BIO144 Course Book (version 2027, draft)

Owen Petchey

2026-10-13

Table of contents

Preface **1**
How to get a copy of this book . . . 1
Datasets are not real . . . 2
Getting the datasets . . . 2
Packages used in this book . . . 2
If you think you found a mistake in this book . . . 2
How this book was made . . . 2
Acknowledgements . . . 3

Introduction (L1) **5**
About this chapter . . . 6
Goals of the course . . . 6
Why data analysis matters . . . 7
High-quality data analysis: four principles . . . 7
How the course is organised . . . 8
The final examination . . . 9
How we will work together . . . 9
Using AI assistants . . . 10
RStudio Server . . . 19
A complete data analysis: reaction times . . . 20
Activate your prior learning . . . 22
Notation and some definitions . . . 22
References and further reading . . . 23

R and RStudio (L2) **25**
Getting R and RStudio . . . 26
Getting to know the RStudio IDE . . . 27
Getting to know R . . . 27
Getting help . . . 30
Finding and fixing errors . . . 30
Add-on packages . . . 32
R Version and add-on package versions . . . 33
R Projects . . . 34

Importing data . . . 35
Viewing the data . . . 35
Data wrangling . . . 37
Visualisation . . . 51
Review . . . 57
Further reading . . . 57
Extras . . . 58

Regression Part 1 (L3) **67**
Introduction . . . 68
Calculating the intercept and slope . . . 71
Dealing with the error . . . 78
Is the model good enough to use? . . . 81
What can go "wrong" during the modeling process? . . . 104
Its a kind of magic… . . . 112
Review . . . 114
Further reading . . . 114
Extras . . . 115

Regression Part 2 (L4) **127**
Introduction . . . 128
How good is the regression model (R^2)? . . . 129
Is the relationship *statistically significant*? . . . 135
Confidence interval of slope . . . 143
Confidence band . . . 145
Prediction band . . . 147
Extrapolation: predicting beyond the data . . . 148
Reporting . . . 149
Association is not causation . . . 150
Relationship between R^2 and the p-value of the slope . . . 151
Review . . . 152
Further reading . . . 153
Extras . . . 153

Analysis of variance (L5) **161**
Introduction . . . 162
How does it look like in R? . . . 164
What is ANOVA? . . . 165
ANOVA as a linear model . . . 166
Doing ANOVA in R . . . 178
Difference between pairs of groups . . . 180
Communicating the results of ANOVA . . . 187
Review . . . 190
Further reading . . . 190
Extras . . . 190

Multiple regression (L6) **193**
Introduction . . . 194
Multiple regression . . . 195
Question 1: As an ensemble, are the explanatory variables useful? . . . 202
Question 2: Which variables are associated with the response? . . . 204
Question 3: What proportion of variability is explained? . . . 205
Question 4: Are some explanatory variables more important than others? 207
Question 5: How do we make predictions? . . . 208
Reporting . . . 211
Collinearity . . . 212
Confounding: why we include other explanatory variables . . . 225
Review . . . 226
Further reading . . . 227
Extras . . . 227

Interactions (L7) **231**
Thought for the week . . . 232
Introduction . . . 232
Implementing interactions . . . 243
Three models . . . 249
General points . . . 259
Review . . . 262
Further reading . . . 262
Extras . . . 262

Count data (L8) **269**
Introduction . . . 270
Example: Soay sheep . . . 271
From LM to GLM . . . 275
Poisson GLM . . . 277
R - Poisson GLM . . . 280
Well done! . . . 288
Overdispersion . . . 288
Zero inflation . . . 289
Multiple explanatory variables . . . 290
Review . . . 293
Further reading . . . 293
Extras . . . 293

Binomial data (L9) **297**
Introduction . . . 298
Binary and binomial data . . . 299
A simple binomial GLM . . . 304
Aggregated binomial data . . . 308
Individual-level binary data (non-aggregated) . . . 317
Two common practical issues . . . 322

Naming models (logistic regression) . . . 322
Review . . . 323
Further reading . . . 323

Ordination (L10) 325
Introduction . . . 326
Ordination I: PCA (Principal Components Analysis) . . . 329
When ordination helps . . . 334
Ordination II: NMDS (Non-metric Multidimensional Scaling) . . . 335
Hypothesis testing: avoid "four separate regressions" . . . 341
PCA in multiple regression . . . 346
Review . . . 347
Further reading . . . 347

Mixed models (L11-1) 349
Introduction . . . 350
Why not just average? . . . 351
The problem: non-independence and pseudoreplication . . . 351
Random intercept models . . . 352
Random slope models . . . 354
Nested and crossed random effects . . . 354
Hands-on example: plant growth with repeated measures . . . 355
Review . . . 363
Further reading (optional): mixed models . . . 364

What next (L11-2) 365

Review: planning and implementing statistical analyses (L12) 369
Introduction . . . 370
Part 1: Planning the analysis . . . 371
Part 2: Implementing the analysis . . . 378
Example 1: Linear model with an interaction . . . 379
Example 2: Binomial GLM . . . 387
Example 3: Count GLM . . . 392
Final checklist . . . 396

Preface

This book contains the content of the course BIO144 Data Analysis in Biology at the University of Zurich. It is intended to be used as a companion to the lectures and practical exercises of the course. All of the required content of the course (i.e., what could be in the final exam) is included in this book. Additional content is included for those who want to learn more.

Sections headed "Extras" at the end of chapters are not examinable. They are there for those who want to learn more. The same applies to the "Further reading" sections: their content is not examined unless it is also in the main text of the book.

Beware that Owen sometimes makes updates to the book during the semester, so if you have downloaded a copy or taken screenshots, your copy may not exactly match the most current version. However, all of the required content will be the same, and any changes will be correcting typos or improving explanations. If content does change in a way that would change the answer to a question in the final exam, Owen will announce this in the lectures and on OLAT.

How to get a copy of this book

If you'd like a copy of this book for yourself, there are a few ways. But beware: if you take a local copy then it will not be updated when Owen makes changes to the online version!

- You can download a PDF version of the entire book: Download PDF
- You can download a complete copy of the HTML version of the book, to read offline: Download HTML book (zip). Unzip it, then open the file `index.html` (in the folder `BIO144_Course_Book_html`) in your web browser.
- You can get all of the source code for the book from the GitHub repository. However, you may find it a little complicated to do anything useful with it!

Datasets are not real

The datasets used in this book are not real datasets. They were created to illustrate the methods taught in the course. Any resemblance to real data is purely coincidental. Please do not use these datasets for any purpose other than learning the methods taught in this course. The patterns in the data may not reflect real-world patterns, and should not be used to draw any conclusions about real-world phenomena.

Getting the datasets

The datasets used in this book are available for download as a zip file here: course_book_datasets.zip. You can download this file and unzip it to get all of the datasets used in the book. The datasets are in CSV format, which can be opened in R or other statistical software.

Packages used in this book

This book uses a number of R add-on packages for data analysis and visualization. You will need to install these packages in order to run the code in the book. The required packages are listed in the `_common.r` file in the GitHub repository for the book. Here is a link to that file: _common.R. You can copy and paste the list of packages from that file into your R console to install them.

If you think you found a mistake in this book

If you think you have found a mistake in the book, please say. A really nice way is to submit an issue on the GitHub repository for the book: Issues page of the GitHub repository. You will need a GitHub account to do this, but they are free and easy to set up. Otherwise tell Owen in person sometime, or in the OLAT Forum, or by email.

When reporting a mistake, please be as specific as possible about where the mistake is. A screenshot works well. Or give the chapter and section number, and copy a chunk of text, as well as a description of the issue problem of course!

How this book was made

The book was written using a type of RMarkdown. It allows a script with a mix of normal text and R code to produce chapters and a book that has a mixture of text, R code, and R output. Rmarkdown is very useful for making reports, books, presentations, and even websites.

This book is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

Acknowledgements

The content was based on lectures originally written by Dr Stefanie Muff.

The content of the book was written with the assistance of Github Copilot, an AI tool that helps write code and text.

Introduction (L1)

Learning objectives

After this unit, students will be able to:

1. Describe how the lectures, course book, practicals, homework, weekly quizzes and final exam of BIO144 fit together. *(not examined)*
2. Identify the prior knowledge and skills assumed at the start of the course, and where to revise them. *(not examined)*
3. Describe the steps of a data analysis workflow, from formulating a question to communicating the answer, and explain why the presentation and analysis should be planned before the data are collected.
4. Describe the four principles for high-quality data analysis: a focused question with planned sampling and analysis; a model that accounts for the distribution and dependence of the data; emphasis on effect sizes and biological relevance; and reporting that makes the research reproducible.
5. Explain the advantages of doing a data analysis with a script in R, such as being repeatable, readable and sharable.
6. Distinguish appropriate from inappropriate uses of AI assistants in learning data analysis and in BIO144 assessments.
7. Recognise and use the course notation for samples, populations and linear models (e.g. the sample mean, sample and population standard deviation, parameter estimates and residuals).

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch01_intro.R
This R script contains the code of this chapter, with comments that explain each step. This is the code from the analysis of the class reaction time data in the first lecture. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 1 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch01_intro.R")
```

Video walkthrough: coming soon.

About this chapter

The first week of BIO144 is different from all the others. It is the only lecture that uses slides, and much of it is about how the course works rather than about statistical methods. This chapter is the written version of that lecture. It covers:

- the goals of the course, and why data analysis matters;
- what makes a data analysis high quality;
- how the course is organised, and how the final examination works;
- how we will work together, including how to use AI assistants;
- RStudio Server;
- a complete data analysis, from question to answer, using data that you collect in the lecture;
- the knowledge it is useful to have already, and the notation used throughout the course.

Goals of the course

The overarching goals of BIO144 are to:

- provide a solid foundation for answering biological questions with quantitative data;
- grow your confidence in your ability to understand and perform data analysis;
- help you plan research projects so that they are easy to analyse;
- excite you about the wonders of data analysis;
- prepare you for the final exam;
- have fun!

Why data analysis matters

Why should you care about data analysis? Only you can answer that question. A strong sense of purpose usually comes from doing something that is meaningful to you and useful to others. Here are some of the ways in which data analysis is both:

- **Showing that a problem exists.** Good analysis can make invisible problems visible, and help others take them seriously.
- **Testing solutions.** Does an intervention work? Does it work for everyone? Does it have unintended consequences?
- **Evaluating evidence that affects people's lives.** Showing whether a drug works, does not work, or has serious side effects is not abstract: it shapes real decisions.
- **Questioning powerful claims.** Marketing, policy and public messaging often rely on selective statistics. You can learn to ask: what is missing? What is exaggerated?
- **Protecting others, especially those with less power.** From children targeted by unhealthy products to communities affected by environmental harm, good analysis can act as a form of protection.
- **Serving something bigger than yourself.** Science, public trust, health, biodiversity, fairness: data analysis is one way people contribute to their community and society.

In this course you will work on questions from many areas of biology, from animal behaviour and plant growth to soil organisms and human health. Each time, the pattern is the same: we start with a **question** (or puzzle, or problem), combine **data** with **analysis**, and arrive at an **answer** (or solution).

But how do we make sure the analysis is a good one?

High-quality data analysis: four principles

Popovic et al. (2024) proposed four principles that lead to reliable findings. They were written for ecology, but they apply to data analysis in all of biology, and they run through the whole of this course.

1. **Define a focused research question, then plan sampling and analysis to answer it.**
    - Define the research question *before* collecting new data or exploring existing data.
    - Match the data collection to the question.
    - Plan (and ideally register) the analysis in advance.
2. **Develop a model that accounts for the distribution and dependence of your data.**
    - Use a distribution appropriate for the response variable (for example, counts are not normally distributed).

   - Model all dependence among observations (for example, repeated measurements of the same individual).
   - Check the assumptions of the model.
3. **Emphasise effect sizes, to replace statistical significance with biological relevance.**
   - Emphasise and report effect sizes, with confidence intervals.
   - Discuss whether the effects are biologically relevant.
   - De-emphasise p-values.
4. **Report methods and findings in enough detail that the research is valid and reproducible.**
   - Publish all data.
   - Provide code that others can run.
   - Visualise the outputs of the model to communicate the results.

You will meet each principle many times. Principle 1 is the first step of the workflow below. Principle 2 is why the course moves from linear models to generalised linear models (for counts and binary data) and mixed models (for grouped data). Principle 3 is why we put so much weight on interpreting estimates and their uncertainty. Principle 4 is one reason we do all analyses with scripts in R.

How the course is organised

> **Note**
>
> The course information website has the current times, dates, and other details that can change from year to year.

The team. You are at the centre of the course, supported by the lectures, the practicals, the discussion forum on OLAT and, when needed, email. The course is taught by Prof. Owen Petchey, with a head teaching assistant, an examinations coordinator, and numerous teaching assistants.

The conceptual side: lectures

- Apart from this first week, the lectures do not use slides.
- The **course book contains what you need to know.** Each week has a chapter.
- In lectures, the lecturer talks, draws, and asks you questions.

Many of the questions in lectures use **think, pair, share**: you first think about the question on your own, then discuss it with a neighbour, then share with the class. Peer discussion improves understanding, even when none of the students in a discussion group originally knew the correct answer (Smith et al. 2009).

The practical side.

- **Practicals**, where you work on data analysis with help from the teaching assistants.
- **Homework**, best done between the lecture and the practical.
- **Weekly practice quizzes**, which you can do at any time.

Please keep up with the course material, and reach out to us if you feel you are falling behind. The weeks do not all need the same amount of time, so look ahead and plan early.

Where to find everything. The course uses four websites:

- **OLAT**, the hub, which links to everything else and hosts the discussion forum. Please ask questions and make requests in the forum.
- The **course information website**, with general information, the learning objectives, and a page for each week.
- The **course book** (this book).
- The site with the **homework, practicals and weekly quizzes** (on the UZH RStudio Connect server: off campus, connect to the UZH VPN first).

Much of the course material is new or recently revised. If you find something that is unclear, too brief, or wrong, please let us know.

The final examination

> **Note**
>
> Check the course information website for the current examination details.

One comment here: The exam consists of **case studies**. In each, you analyse a dataset and answer questions about it. Many conceptual questions are embedded in the case studies. Because the exam asks you to analyse data in R and give numerical answers, the best preparation is to practise doing analyses yourself, every week.

How we will work together

We are here to support you: to give you the knowledge of how to learn more, and the confidence that you can.

- Everyone here can learn to think analytically and statistically.
- We will hold you to high standards, because we believe you can meet them. Our job is to support you in reaching them.
- Struggle is part of growth, not a sign of failure or deficiency.
- Feeling stressed before a challenge is normal. A higher heart rate, faster breathing and a tight stomach are your body giving you more resources to meet the challenge.

High standards and wise feedback. Feedback in this course follows the idea of "wise feedback" (Cohen et al. 1999): *"I'm giving you these comments because I have high standards and I know that you can meet them."* Feedback will be honest and specific. It comes from belief in your potential, not criticism of you as a person. If it ever feels like you are being criticised, try to reframe it as being challenged to grow. This class is a safe space for you, and it is a challenging space for you.

Using AI assistants

AI assistants such as ChatGPT, Claude, Gemini and Copilot can help you learn, if you use them well.

Asking for help understanding is usually a good use. For example:

- "What is the mode of a distribution?"
- "I didn't understand that explanation. Can you give an easier one?"
- "Can you show me the mode graphically?"
- "Can you give me an example of getting the mode of a distribution in R?"

Asking for help writing code needs more care. A request like "write code to import data and give the number of missing values in each variable" can save time. A request like "import the data, fit a regression, check the diagnostics, summarise the results, make a nice figure, put it all in a Quarto file, and do it while I go and chill" means you learn nothing, and you cannot check whether the result is correct.

Your relationship with AI assistants. In a strong relationship, the AI is an empowering tool: a knowledgeable collaborator that complements your skills, speeds up your work, and supports your growth. You provide the context, direction and critical judgement. In a poor relationship, you use the AI's output without understanding it, copying and pasting solutions. This leads to errors, inefficient or misaligned solutions, and your own skills stagnating.

The knowledge and expertise *you* bring is what makes a good relationship with an AI assistant possible. So we suggest you **minimise your use of AI assistants during the course** (and minimise your use of the example solutions too). Do the work, and get the benefit.

Guidance and good practice

> **!** Important
>
> For the final examination, you will use your own computer, but the test will run inside the Safe Exam Browser, which will be configured to block all access to generative AI tools, browser-based assistants, external software,

> online services, and any AI code copilots inside RStudio or other IDEs. This means **no form of generative AI will be available during the exam**. Because of this, please avoid becoming overly reliant on GenAI—such as ChatGPT, Claude, Gemini, Copilot, or similar tools for answering quiz questions, explaining results, fixing errors, guiding your analysis, or writing code. You must be able to perform all tasks independently. We also strongly recommend that you do not use RStudio with Copilot integration during the course, as it will not function in the exam environment and may leave you under prepared. Throughout the semester, be sure to practice writing your own R code, interpreting outputs yourself, and applying statistical reasoning without AI assistance, as your exam performance will depend entirely on your own knowledge and skills.

Generative AI (GenAI) tools can support learning, exploration, and coding in R. They can be powerful assistants, but they must be used with care. This section introduces the types of tools available, provides guidelines for responsible use, highlights red flags for problematic usage, and gives examples of good and poor practice.

Typical uses:

- asking conceptual questions
- summarizing methods
- generating example code
- explaining error messages

Strengths:

- flexible and conversational
- good for brainstorming
- can generate starter code

Limitations:

- often wrong in subtle ways
- may hallucinate functions
- cannot see your working R session

Guidelines for Good Use of Generative AI

Use GenAI as a Helper, Not a Source of Truth

Best uses:

- drafting
- explanation
- syntax reminders
- scaffolding

Not reliable for:

- model choice
- statistical inference
- interpreting coefficients
- designing analysis workflows
- checking assumptions

Always Verify AI-Generated Code and Explanations

Check:

- does the code run?
- do variable names match?
- is the model appropriate?
- are assumptions addressed?
- is the explanation logically correct?

Keep Human Judgement Central

GenAI cannot:

- understand scientific questions
- evaluate model assumptions
- know ecological/biological reasoning

- determine appropriate models

Provide Context Carefully

When asking GenAI:

- describe variables
- provide example data
- specify your goal
- show your existing code

Better context = better answers.

Use GenAI to Improve Understanding, Not Bypass It

Helpful:

- *"Explain logistic regression."*
- *"Why do residuals fan out?"*

Not helpful:

- *"Do my assignment for me."*

Indicators of Problematic Usage

Code That Does Not Reflect Ability

Signs:

- unfamiliar advanced syntax
- unexplained packages
- inconsistent style

Hallucinated Functions or Nonsensical Code

Examples:

- `slope(x)` in mixed models
- missing arguments
- fabricated packages

Statistical Errors Typical of AI

Common issues:

- wrong model family
- wrong inference logic
- invented assumptions
- incorrect explanation of coefficients

Lack of Understanding

Indicators:

- cannot explain model
- inconsistent interpretations
- identical phrasing to AI output

Over-Reliance on AI

Signs:

- using AI for every step
- no debugging effort
- stagnation in skill development

Examples of Good and Problematic Use

Good Use Examples

A. Syntax help
"How do I specify a random slope in `lme4`?"

B. Clarification
"How does adding an interaction change interpretation?"

C. Debugging
"What does 'object not found' usually mean?"

D. Brainstorming
"How can I visualise a logistic regression?"

Problematic Use Examples

A. Blindly copying model code

```
lm(y ~ x1 + x2 * x3 * x1)
```

B. Incorrect statistical logic
AI code labelled as a bootstrap but is actually a permutation test.

C. Misleading interpretation
Claims that coefficients assume explanatory variable independence.

D. Presenting AI-generated plots without understanding

E. Outsourcing entire workflow
"Write a script that loads data, cleans it, runs models, interprets, and writes the report."

Summary

Generative AI can:

- help learning
- support debugging
- provide code scaffolds
- explain concepts

But it can also:

- hallucinate
- produce incorrect models
- misinterpret statistics

Use GenAI as a supportive tool—never as an unquestioned authority.
Good use of GenAI *supports* learning. Problematic use *replaces* it.

Common GenAI Errors in R and Statistical Modelling

Generative AI tools can be helpful for writing R code, exploring ideas, and learning syntax.
However, they sometimes produce *plausible but incorrect* code or explanations. This section provides real examples of typical GenAI mistakes, with correct solutions and learning points.

Why this matters: GenAI is a **pattern-matching system**, not a statistical reasoning engine. It does not understand assumptions, inference, or modelling logic.Therefore, students should **never accept code or explanations without checking them**.

Incorrect formula structure in `lm()`

Prompt: *Fit a linear model with main effects and a two-way interaction between `x2` and `x3`.*

Incorrect GenAI output:

```
lm(y ~ x1 + x2 * x3 * x1, data = df)
```

This includes an unintended **three-way interaction** and extra terms.

Correct:

```
lm(y ~ x1 + x2 * x3, data = df)
```

Learning point: Always check model formulas carefully. AI often adds or removes interactions.

Confusing bootstrap and permutation tests

Documented case: GenAI was asked for a *bootstrap t-test*.

Incorrect GenAI code (actually a permutation test):

```
t_stats <- replicate(1000, {
  perm <- sample(df$group)
  t.test(df$value ~ perm)$statistic
})
```

Correct bootstrap approach:

```
t_stats <- replicate(1000, {
  sample_df <- df[sample(nrow(df), replace = TRUE), ]
  t.test(value ~ group, data = sample_df)$statistic
})
```

Learning point: The logic of inference matters. Code that runs is not necessarily correct.

Incorrect explanation of linear-model coefficients

Incorrect claim: *"Coefficients assume independence among explanatory variable."*

This is false. Linear model coefficients describe **conditional effects within the model**, regardless of collinearity.

Learning point: Interpretations come from the model structure, not from simplistic assumptions GenAI sometimes invents.

Hallucinated functions in mixed models

Incorrect GenAI output:

```
lmer(y ~ x + (slope(x) | group), data = df)
```

`slope()` does not exist.

Correct random-slope specification:

```
lmer(y ~ x + (x | group), data = df)
```

Learning point: Always verify syntax in package documentation.

Wrong variable names

The dataset has variables `height` and `age`.

Incorrect GenAI output:

```
lm(Height ~ Age, data = df)
```

Correct:

```
lm(height ~ age, data = df)
```

Learning point: GenAI often guesses variable names. Check against your data.

Wrong model family for binary data

Incorrect GenAI output (linear regression):

```
lm(y ~ x, data = df)
```

Correct logistic regression:

```
glm(y ~ x, data = df, family = binomial)
```

Learning point: For binary response variables, specify the model family explicitly.

Incorrect explanation of random intercepts

Incorrect claim:
"Random intercepts eliminate correlation among repeated measures."

Incorrect — they **model** correlation, not eliminate it.

Learning point: Random effects structure determines the implied correlation. AI explanations are often vague or wrong here.

Omitting interaction terms in ANOVA

Prompt: *Two-way ANOVA with interaction.*

Incorrect:

```
aov(y ~ factor1 + factor2, data = df)
```

Correct:

```
aov(y ~ factor1 * factor2, data = df)
```

Learning point: Confirm that the model matches the experimental design.

Incorrect use of `predict()`

Prompt: *Predict for new x values.*

Incorrect GenAI output:

```
predict(model)
```

This gives **in-sample fitted values**, not predictions for new data.

Correct:

```
predict(model, newdata = data.frame(x = c(1, 2, 3)))
```

Learning point: Always specify `newdata` for predictions.

Poor explanations of multicollinearity

Incorrect GenAI claim:
"Multicollinearity is indicated when the model p-value is low but the individual explanatory variable p-values are high."

This is an unreliable and incomplete diagnostic.

Better diagnostics:

```
car::vif(model)
cor(df)
model.matrix(model)
```

Learning point: AI often repeats common internet tropes rather than robust statistical principles.

GenAI Summary

GenAI can:

- write useful scaffolding code,
- provide quick reminders,
- help with simple tasks.

But it can also:

- hallucinate functions,
- give subtly incorrect models,
- invent statistical logic,
- provide plausible but wrong explanations.

Advice for students:
Use GenAI as a starting point, not an authority.
Always check:

- function names,
- model formulas,
- assumptions,
- interpretations,
- and logic.

In statistics, clarity of reasoning matters more than code that merely *runs*.

RStudio Server

You can use R and RStudio either on your own laptop or in a web browser, through the UZH RStudio Server (https://rstudio.mnf.uzh.ch/).

- **You will have to use RStudio Server in the exam**, so whichever you use during the course, make sure you can log in to RStudio Server and work there (the Unit 1 practical starts with this).

- RStudio Server: much of the setup is already done for you, but you have to be connected to the internet (and to the UZH network or VPN for some resources).
- Your own laptop: works offline, but you need to install R, RStudio and the add-on packages yourself (see the "Getting R & RStudio" section of the Previous Knowledge self-tests).

Only registered users can log in. Everyone in the OLAT course is registered, so you must be in the OLAT course to get access.

A complete data analysis: reaction times

To give you a feel for what is involved in data analysis, the first lecture includes a live demonstration of the whole workflow, using data collected from you during the lecture. You will understand some of the demonstration, and not other parts of it. That is expected: keep notes about what you do not understand, and look for the answers as the course goes on.

The steps of the workflow are:

1. Question
2. Expectation
3. Planned presentation and analysis
4. Selection of subjects
5. How the data will be collected
6. Ethics and permissions
7. Data collection
8. Data wrangling
9. Visualisation
10. Statistical test
11. Critical thinking
12. Reporting and communicating

Notice that the first six steps all happen *before* any data are collected. This is Popovic et al.'s first principle in action.

Question

What should our question be? As always, there are some influences and some constraints. We should ask a question that is of interest to us, and of some importance. And we must be able to collect the data needed to answer it, within our current constraints.

The question we address is: **do the reaction times of male and female students at the University of Zurich differ?** Reaction times matter for safety (for example, when driving) and in sport (for example, at the start of a sprint race).

Expectation

There has already been quite a lot of research on reaction times. In general, males tend to have faster reaction times than females, so we expect the same for students at the University of Zurich.

Planned presentation and analysis

Before collecting any data, we decide what graph we will make and what statistical test we will use.

- **Graph:** box plots of reaction time (on the horizontal axis) for females and males. This shows the centre and spread of the two groups side by side.
- **Statistical test:** we are comparing the mean of a continuous response variable (reaction time) between two groups (the explanatory variable, sex), so a two-sample t-test is a natural choice. Its assumptions include independent observations, and roughly normally distributed values with similar variances in the two groups.

Selection of subjects

Ideally, the subjects would be a random sample from the population we want to draw conclusions about. Are the students in this lecture a random selection of all humans? Of all students at the University of Zurich? The way subjects are selected limits the population to which our conclusions can be generalised.

How the data are collected, and ethics

You measure your own reaction time using an online test, and record it, along with some other information about yourself, in a form linked from the course information website. Because these are data about people, we need to think about ethical issues, such as consent, anonymity, and how the data will be used and shared.

Data wrangling, visualisation and statistical test

These steps are done live in RStudio, with a script.

Doing the analysis with a script in R, rather than by pointing and clicking, makes it:

- efficient and scalable: it is quick to re-run, including on more or new data;
- consistent, repeatable and reliable: the same steps give the same result every time;
- readable, sharable and persistent: others (and your future self) can see exactly what was done, check it, and re-use it;
- robust: mistakes can be found and fixed, and the whole analysis re-run.

This is Popovic et al.'s fourth principle in action.

Critical thinking, and reporting

Finally, we think critically about the result. How big is the difference in reaction times, and is it biologically relevant (principle 3)? How precise is our estimate? Could something other than sex explain the difference? To whom do the conclusions apply, given how the subjects were selected? Then we report the result clearly, with a figure and a short description of the estimate, its uncertainty, and the test.

Activate your prior learning

BIO144 builds on knowledge you should already have from earlier courses. The "previous knowledge" resources on OLAT help you review it. If you have not worked through them yet, there is still time. It is useful to already be familiar with:

- **Summarising distributions of data:** central tendency (mean, median, mode); spread (range, quartiles, standard deviation, standard error); symmetry (skew); pointedness (kurtosis); number of peaks (unimodal, bimodal, multimodal).
- **Types of variable:** numeric, continuous, discrete, categorical; response (dependent) and explanatory (independent) variables.
- **Statistical ideas:** expected value, parameter, parameter estimate, variable, the central limit theorem, hypothesis testing, degrees of freedom, p-value; the difference between a population and a sample.
- **Probability:** for example, the probability of getting five heads when tossing a fair coin seven times; probability distributions (normal, binomial, Poisson, exponential); frequency distributions (histograms).
- **Basic methods and graphs:** the t-test, linear regression and the equation of a straight line; box-and-whisker plots and scatterplots.
- **Notation**, described in the next section.

Notation and some definitions

Throughout the course, we will use the following notation:

- x for a variable. Typically this variable contains a set of observations. These observations are said to represent a sample of all the possible observations that could be made of a *population*.
- $x_1, x_2, ...$ for the values of a variable
- x_i for the ith value of a scalar variable. This is often spoken as "x sub i" or the "i-th value of x".
- $x^{(1)}$ for variable 1, $x^{(2)}$ for variable 2, etc.
- The mean of the sample x is $\bar{x}$. This is usually spoken as "x-bar".
- The mean of x is calculated as $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.
- n is the number of observations in a sample.

- The summation symbol $\sum$ is used to indicate that the values of x are summed over all values of i from 1 to n.
- The standard deviation of the sample is s. The standard deviation of the population is σ.
- The variance is s^2. The variance of the population is σ^2.
- The variance of the sample is calculated as $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$.
- The standard deviation of the sample is calculated as $s = \sqrt{s^2}$.
- y is usually used to represent a dependent / response variable.
- x is usually used to represent an independent / predictor / explanatory variable.
- β_0 is usually used to denote the intercept of a linear model.
- β_1, β_2, etc. are usually used to denote the coefficients of the independent variables in a linear model.
- Estimates are denoted with a hat, so $\hat{\beta}_0$ is the estimate of the intercept of a linear model.
- Hence, the estimated value of y_i in a linear regression model is $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i^{(1)}$.
- e_i is the residual for the ith observation in a linear model. The residual is the difference between the observed value of y_i and the predicted value of y_i ($\hat{y}_i$).
- Often we assume errors are normally distributed with mean 0 and variance σ^2. This is written as $e_i \sim N(0, \sigma^2)$.
- SST is the total sum of squares. It is the sum of the squared differences between the observed values of y and the mean of y. It is calculated as $\sum_{i=1}^{n}(y_i - \bar{y})^2$.
- SSM is the model sum of squares. It is the sum of the squared differences between the predicted values of y and the mean of y. It is calculated as $\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$.
- SSE is the error sum of squares. It is the sum of the squared differences between the observed values of y and the predicted values of y. It is calculated as $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
- The variance of x can be written as $Var(x)$. The covariance between x and y can be written as $Cov(x, y)$.
- Covariance is calculated as $Cov(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$.
- H_0 is the null hypothesis.
- α is the significance level.
- df is the degrees of freedom.
- p is the p-value.

References and further reading

Cohen, G.L., Steele, C.M. & Ross, L.D. (1999). The mentor's dilemma: providing critical feedback across the racial divide. *Personality and Social Psychology Bulletin*, 25, 1302–1318.

Popovic, G., Mason, T.J., Drobniak, S.M., Marques, T.A., Potts, J., Joo, R., et al. (2024). Four principles for improved statistical ecology. *Methods in Ecology and Evolution*, 15, 266–281.

Smith, M.K., Wood, W.B., Adams, W.K., Wieman, C., Knight, J.K., Guild, N. & Su, T.T. (2009). Why peer discussion improves student performance on in-class concept questions. *Science*, 323, 122–124.

Students who are curious and would like to explore these topics further (purely for their own interest) may find the following resources useful. Material from these resources will not be examined in the final exam, unless it is also already present in the course book.

If you would like to read more about reaction time differences between men and women, this is a quite interesting paper: On the Implications of a Sex Difference in the Reaction Times of Sprinters at the Beijing Olympics, by Lipps et al (2011). The analyses are relatively simple, and the implications explored are quite interesting. Unfortunately, the data used in the paper is not publicly available, so you cannot use it for practice here.

A more detailed data analysis workflow suggestion is here on the Insights from data website.

Here is an article about Exploring The Ethical Implications Of Ai In Data Analytics: Challenges And Strategies For Responsible Implementation. It is rather brief and high-level, but may be of interest when you seek an ethical perspective on the use of AI tools in data analysis.

R and RStudio (L2)

Learning objectives

After this unit, students will be able to:

1. Explain the purpose of the script, the console and the environment in RStudio.
2. Write and run R code in a script, using objects, functions and arguments correctly.
3. Install and load add-on packages, and explain the difference between installing and loading a package.
4. Use R's help system and other resources to find out how a function works and what arguments it takes.
5. Import a dataset into R and inspect its structure, including the variable names and variable types.
6. Convert variables to factors, and set or change the reference level of a categorical variable.
7. Manipulate data in R: select variables, filter observations, create new variables, and calculate summaries by group.
8. Produce exploratory graphs (e.g. histograms, scatterplots, box plots) to inspect distributions of and relationships between variables.
9. Identify and fix common errors in R code (e.g. typos, misspelled variable names, packages not loaded).
10. Organise an analysis as a script within an RStudio project, with meaningful names for datasets and model objects, that reproduces all results when run from top to bottom.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch02_r_and_rstudio.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 2 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch02_r_and_rstudio.R")
```

Video walkthrough: coming soon.

Getting R and RStudio

R is a programming language and software environment for statistical computing and graphics. RStudio is an integrated development environment (IDE) for R. ***RStudio*** provides a user-friendly interface for working with R, including a console, a script editor, and tools for managing packages and projects.

We highly recommend using **RStudio** to work with **R**.

There are two ways to use **RStudio**:

1. **RStudio Desktop**: a standalone application that you can install on your computer. If you choose this option, you will need to install R first, and then install RStudio. This usually requires administrator privileges on your computer. If you have problems installing add-on packages, they will have to be fixed by you or with our help (rarely we cannot find a solution). Follow the instructions on this website about how to install R and RStudio: https://posit.co/download/rstudio-desktop/.

2. **Rstudio Cloud**: a web-based version of RStudio that you can use in your web browser. You don't need to install anything on your computer, and you can access your work from any computer with an internet connection. The Faculty of Science has a RStudio Cloud here that you can use (and will have to use during the final exam).

What do we recommend? Try the cloud first. If you like it then continue to use it.

> **!** Important
>
> Whether you use the RStudio application on your computer, or use RStudio on the Cloud, you are responsible for the safety and persistence of your files (data, code, etc.). Just because you're using RStudio on the Cloud does not mean your files are automatically saved forever. Make sure to download and back up your important files regularly!

Getting to know the RStudio IDE

When you open RStudio, you will see a window with four main panes:

1. **Source pane**: where you can write and edit R scripts, R Markdown documents, and other files. This pane can have multiple tabs, so you can have several files open at the same time.
2. **Console pane**: where you can type and execute R commands directly. This pane has multiple tabs, including: **Console**, **Terminal**, and **Jobs**. During this course we will mostly use the **Console** tab.
3. **Environment pane**: where you can see the objects (data frames, vectors, etc.) that are currently in your R session. This pane has multiple tabs, including: **Environment**, **History**, **Connections**, and **Tutorial**. During this course we will mostly use the **Environment** tab.
4. **Files/Plots/Packages/Help pane**: where you can manage files, view plots, manage packages, and access help documentation. This pane has multiple tabs, including: **Files**, **Plots**, **Packages**, **Help**, and **Viewer**. During this course we will mostly use the **Files**, **Plots**, **Packages**, and **Help** tabs.

Our **Scripts** are in the Source pane tabs. The code / script we write in R is usually saved in a file with the extension `.R`. This file can be opened and edited in the Source pane. Creating a new R script: File > New File > R Script.

You can run code from the script by selecting the code and clicking the "Run" button, or by using the keyboard shortcut `Ctrl + Enter` (Windows) or `Cmd + Enter` (Mac).

There is so much more to learn about the RStudio IDE, but we will cover that as we go along in the course.

Getting to know R

In our newly opened script file, type the following code:

```
# This is a comment. Comments are ignored by R.
# They are useful for explaining what your code does.
```

Then type the following code:

```
1 + 1
```

```
[1] 2
```

```
exp(2)
```

```
[1] 7.389056
```

```
sqrt(16)
```

```
[1] 4
```

Select all the code and run it (using the "Run" button or `Ctrl + Enter` / `Cmd + Enter`). You should see the results of the calculations in the Console pane.

Now try assigning values to named object:

```
a <- 5
b <- 10
c <- a + b
```

> **Note**
>
> In fact, just about everything in R is an object! These objects live in your R session (i.e., in the memory of your computer), and you can see them in the Environment pane. You can create objects to store data, functions, and other information. Your data files are something different. They are just like other files on your computer, such as documents, images, and music files. They live on your hard drive (or in the cloud), and you can import them into R when you need to work with them.

And then print the value of `c`:

```
c
```

```
[1] 15
```

You should see the value `15` printed in the Console pane.

We can also create vectors:

```
my_vector <- c(1, 2, 3, 4, 5)
my_vector
```

```
[1] 1 2 3 4 5
```

And do maths on vectors:

```
my_vector * 2
```

```
[1]  2  4  6  8 10
```

```
my_vector + 10
```

```
[1] 11 12 13 14 15
```

```
my_vector ^ 2
```

```
[1]  1  4  9 16 25
```

We can vectors of strings (text):

```
my_strings <- c("apple", "banana", "cherry")
my_strings
```

```
[1] "apple"  "banana" "cherry"
```

And can perform operations on strings:

```
paste("I like", my_strings)
```

```
[1] "I like apple"  "I like banana" "I like cherry"
```

```
toupper(my_strings)
```

```
[1] "APPLE"  "BANANA" "CHERRY"
```

And we can create data frames, which are like tables of data

```
my_data <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 35),
  Height = c(165, 180, 175)
)
my_data
```

```
     Name Age Height
1   Alice  25    165
2     Bob  30    180
3 Charlie  35    175
```

Above we have numerous examples of functions: `exp()`, `sqrt()`, `c()`, `paste()`, `toupper()`, and `data.frame()`. Functions are a fundamental part of R programming. They are used to perform specific tasks, such as calculations, data manipulation, and data analysis. All functions have a name and can take arguments (inputs) and return values (outputs). They are called by writing the function name followed by parentheses, with any arguments inside the parentheses.

You likely guessed that there is much much more to learn about R, but we will cover that as we go along in the course.

Getting help

R has a built-in help system that you can use to get information about functions, packages, and other topics. To access the help system, you can use the `?` operator followed by the name of the function or topic you want to learn about. For example, to get help on the `mean()` function, you would type:

```
?mean
```

This will open the help documentation for the `mean()` function in the Help pane of RStudio. The documentation includes a description of the function, its arguments, and examples of how to use it. Some of the help documentation is very useful and accessible, other is less so. Over time you will learn which functions and packages have good documentation, and you will get better and better at understanding R help files.

Of course you can use any other resources to get help with R, including online forums, tutorials, and books. Some popular online resources for R help include:

- Stack Overflow
- RStudio Community
- R-bloggers
- The R Graph Gallery

You can also use search engines like Google to find answers to your R questions. Just be sure to include "R" in your search query to get relevant results.

AI assistants like ChatGPT can also be useful for getting help with R programming. You can ask specific questions about R code, functions, and packages, and get instant responses.

And of course there is always your course instructors and fellow students to help you out when you get stuck.

Finding and fixing errors

Everyone who writes R code gets errors. Error messages are not a sign that you are doing badly; they are information. Learning to read them and fix the problem quickly is one of the most useful skills you can develop, and it matters in the exam, where you have to get your code working on your own.

Errors and warnings

An **error** stops R: the code did not run, and nothing was produced. A **warning** means the code ran, but R noticed something you should check. Warnings are easy to ignore, but they can point to real problems (for example, missing values that were silently removed).

Read the message

The message usually tells you what went wrong, and sometimes where. For example:

```
my_values <- c(3, 5, 8)
sum(my_value)
```

```
Error:
! object 'my_value' not found
```

"object 'my_value' not found" tells us that R does not know anything called `my_value`. Here it is a typo: the object is called `my_values`.

Common errors and what they usually mean

Message (or symptom)	Usual cause	What to check
`object 'x' not found`	Typo in the name; wrong capitalisation (R is case-sensitive); the line that creates the object has not been run	Spelling and capitals; run the script from the top
`could not find function "f"`	Typo in the function name; the package containing it is not loaded	Spelling; add `library(package)`
`there is no package called 'p'`	The package is not installed	Install it once with `install.packages("p")`
`unexpected symbol`, `unexpected ')'`, or `+` in the console that never goes away	Syntax error: a missing comma, bracket or quotation mark	Count brackets and quotes; press Esc to clear the `+` prompt
`cannot open file ...` `No such file or directory`	Wrong file name or path; working directory not where you think	File name and folder; use an RStudio project
`non-numeric argument to binary operator`	Doing arithmetic on text or a factor	`str()` of the data; is the variable numeric?
A variable has unexpected `NA` values	Missing data in the file; text in a numeric column	`summary()` of the data
A model has fewer observations than expected	Rows with missing values were dropped	`nobs(model)`, `summary()` of the data

A strategy for fixing errors

1. **Read the error message**, and find the line that caused it.
2. **Run the code line by line**, from the top of the script, to find the first line that fails. Often the real mistake is earlier than the line that produces the error.
3. **Look at your objects.** Use `str()`, `head()`, `names()` and `summary()` to check that the data are what you think they are: the right variable names, numeric variables that really are numeric, no unexpected missing values.
4. **Restart R and run the whole script from the top** (in RStudio: Session > Restart R, then run all). This finds problems caused by objects that exist in your environment but are not created by your script.
5. **Make the problem small.** Remove everything that is not needed to produce the error. Often the cause becomes obvious.
6. **Search for the error message**, or ask for help. When you ask (for example in the course forum), include the code, the exact error message, and what you expected to happen.

Add-on packages

R has a vast ecosystem of add-on packages that extend its functionality. These packages are collections of functions, data, and documentation that can be installed and loaded into your R session. There are thousands of packages available on CRAN (the Comprehensive R Archive Network) and other repositories like Bioconductor and GitHub.

We will be using several packages throughout this course. To install a package, you can use the `install.packages()` function. For example, to install the `ggplot2` package, you would type:

```
install.packages("ggplot2")
```

You can also use the RStudio interface to install packages. Go to the "Packages" tab in the bottom right pane, click on "Install", type the name of the package you want to install, and click "Install".

You can see which packages are currently installed by looking in the "Packages" tab.

> **Tip**
>
> You only need to install a package once. After it is installed, you can load it into your R session using the `library()` function. Do not install packages every time you want to use them; just load them with `library()`.

R Version and add-on package versions

(This section concerns the Desktop version of R and RStudio, and not so much the Cloud version, because version management is handled for you in the Cloud.)

R and its add-on packages are constantly being updated and improved. This can cause problems when trying to install or use packages that depend on specific versions of R or other packages.

Imagine that the online version of a package has been updated and now only works with the lastest version of R. If you are using an older version of R, you may not be able to install or use that package.

Or if a package depends on another package that has been updated, you may need to update that package as well to use the first package.

This sounds complicated, but there are some simple steps you can take to reduce the chances of running into version-related problems:

1. **Keep your R version up to date.** New versions of R are released every 6 months or so, and they often include important bug fixes and new features. You can check your current R version by typing `R.version.string` in the Console. To update R, you can download the latest version from the CRAN website.
2. **Keep your add-on packages up to date.** You can update all your installed packages by using the `update.packages()` function. This will check for updates for all installed packages and install the latest versions. You can also use the RStudio interface to update packages by going to the "Packages" tab, clicking on "Update", selecting the packages you want to update, and clicking "Install Updates".
3. **Do this well before critical deadlines or important events (e.g., exams).** Updating R and packages can sometimes lead to unexpected issues, so it's best to do it well in advance of when you need everything to work perfectly.

Nevertheless, even with these precautions, you may still encounter version-related issues from time to time. When this happens, don't panic!

A common problem you might see is an error message when trying to install or load a package, indicating that the package requires a newer version of R or another package. The error / warning message might look like:

```
warning: package 'xyz' requires R version >= 4.2.0
```

```
Warning in install.packages: package ‘XYZ’ is not available (for
R version 4.2.0)
```

These messages indicate that the package you are trying to install or load requires a newer version of R than the one you currently have. To fix this, you will need

to update your R installation to the required version or higher. Then its also a good idea to update your packages as well.

> **Note**
>
> RStudio is also regularly updated, with new version released every several months or so. Your version of RStudio is independent of your version of R, so you can update RStudio without changing your R version. Note that usually your version of RStudio is not as important as your version of R and the packages you are using. So updating RStudio is usually not a high priority and doesn't often help solve problems related to add on package versions.

R Projects

I always work within R Projects. R Projects help you to organise your work and keep all files related to a project in one place. They also make importing data a breeze.

But what is an R Project? An R Project is a directory (folder) that contains all the files related to a specific project. When you open an R Project, RStudio automatically sets the working directory to the project directory, so you don't have to worry about setting the working directory manually.

To see if you're working within an R Project, look at the top right of the RStudio window. If you see the name of your project there, you're good to go. If you see "Project: (None)", then you're not working within an R Project.

If you click on the project name, a dropdown menu will appear. From there, you can create a new project, open an existing project, or switch between projects.

Create a new R Project: File > New Project > New Directory or Existing Directory > New Project > Choose a name and location for your project > Create Project.

> **Important**
>
> **Get organised!** Put all files for a project in one folder. For example, I made a folder called `BIO144_2027` and put all files related to this course in that folder. Within that folder, I have subfolders for `datasets`, `scripts`, and `results`. I then create an R Project in the `BIO144_2027` folder. This way, all files related to the course are in one place, and I can easily find them later.

Now, always open and ensure you're working within the R Project for your project. As mentioned, you can see the project name at the top right of the RStudio window. And if its not the correct project, click on the name to get the

drop-down list of available projects from which you can switch to the correct one.

Importing data

First, get some datasets for us to work with. The datasets used in this book are available for download as a zip file here: course_book_datasets.zip. You can download this file and unzip it to get all of the datasets used in the book. The datasets are in CSV format, which can be opened in R or other statistical software.

Put the datasets in a `datasets` folder in your project folder. All code in this book reads data from `datasets/...`.

We will use the `readr` package to import data into R. The `readr` package provides functions to read data from various file formats, including CSV (comma-separated values) files, tab-separated values files, and others.

To read a CSV file, we can use the `read_csv()` function from the `readr` package. For example, to read a CSV file called `my_data_file.csv`, we can use the following code:

```
library(readr)
my_data <- read_csv("datasets/my_data_file.csv")
```

This code will read the `my_data_file.csv` file from the `datasets` folder within the current working directory (which should be the R Project directory) and store it in a data frame called `my_data`.

> Tip
>
> **Easily getting the file path** In RStudio, you can easily get the file path by putting the cursor in the parentheses of the `read_csv()` function, the press the tab key. A drop-down menu will appear with options to navigate to the file. This way, you don't have to type the file path manually.

Viewing the data

Once you've imported your data, you can view it in several ways:

- Click on the data frame in the Environment tab in RStudio to open it in a new tab.
- Use the `View()` function to open the data frame in a new tab in RStudio.
- Use the `head()` function to view the first few rows of the data frame.
- Use the `str()` function to view the structure of the data frame, including the variable names and types.

- Use the `summary()` function to get a summary of the data frame, including basic statistics for each variable.

Another useful function is `glimpse()` from the `dplyr` package, which provides a quick overview of the data frame.

```
library(dplyr)
glimpse(my_data)
```

There are many checks you can do to ensure your data was imported correctly. For example checking if there are duplicated values in a variable when there shouldn't be:

```
any(duplicated(my_data$Name))
```

```
[1] FALSE
```

The function `any()` will return `TRUE` if there are any duplicated values in the `Name` variable, and `FALSE` otherwise. The function `duplicated()` returns a logical vector indicating which values are duplicates. We use the dollar sign `$` to access a specific variable (column) in the data frame. A logical vector is a vector that contains only `TRUE` or `FALSE` values:

```
duplicated(my_data$Name)
```

```
[1] FALSE FALSE FALSE
```

All three logicals are `FALSE`, meaning none of the three are duplicates. If there were duplicates, the corresponding positions in the logical vector would be `TRUE`. For example:

```
example_vector <- c("A", "B", "A", "C", "B")
```

What do you expect the output of `duplicated(example_vector)` to be?

A final check (though not the final one we could do–there are many others). Let us check for missing values and get a count of how many there are in each variable. We can do this with the following *tidyverse* code:

```
library(tidyverse)
my_data |>
  summarise(across(everything(), ~ sum(is.na(.))))
```

```
  Name Age Height
1    0   0      0
```

Looks complicated eh! Well, that's because it is, for sure. But let's break it down:

- `summarise()` creates a new data frame with summary statistics.
- `across(everything(), ~ sum(is.na(.)))` applies the function `sum(is.na(.))` to every variable in the data frame.

- The `is.na()` function returns a logical vector indicating which values are missing (`NA`), and the `sum()` function counts the number of `TRUE` values in that vector (i.e., the number of missing values).

> **!** Important
>
> **Let's assume your data was imported incorrectly.** This means you have to inspect it carefully. Check that the variable names are correct, that the data types are correct (e.g., numeric, character, factor), that there are the correct number of rows and columns. If you find any issues, you need to find out what caused them, fix them, and re-import the data (see below).

Common data import problems:

- Incorrect delimiter: If your data file uses a different delimiter (e.g., tab, semicolon), you need to specify it in the `read_csv()` function using the `delim` argument (e.g., `read_delim("data.csv", delim = "\t")` for tab-delimited files).
- Missing values: If your data file uses a specific value to represent missing data (e.g., "NA", "-999"), you need to specify it in the `read_csv()` function using the `na` argument (e.g., `read_csv("data.csv", na = c("NA", "-999"))`).
- Only one column: If your data file has only one column, it may be because the delimiter is incorrect. Check the delimiter and re-import the data with the correct delimiter.
- You opened the downloaded file in Excel and then saved it: Excel may have changed the format of the file when you opened and saved it. Always work with the original downloaded file.
- Wrong path or file name: Make sure the file path and name are correct. Remember, when you work in an R Project, you can place the cursor in the parentheses of the `read_csv()` function and press the tab key to navigate to the file.

Data wrangling

Now we have our data imported and checked, and we're ready to start working with it. This process is called data wrangling, and it involves cleaning, transforming, and reshaping the data to make it suitable for visualisation and analysis.

Clean the variable names

The first thing I like to do is standardise and clean up the variable names. I like to use the `janitor` package for this:

```
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```
my_data <- my_data |>
  clean_names()
```

The `clean_names()` function from the `janitor` package will convert variable names to a consistent format (lowercase, spaces replaced by underscores, no special characters).

> **Note**
>
> When we ran the code `library(janitor)` we got the message: **Attaching package: 'janitor' The following objects are masked from 'package:stats': chisq.test, fisher.test** This sometimes happens when two packages have functions with the same name. In this case, the `janitor` package has functions called `chisq.test()` and `fisher.test()`, which are also in the base `stats` package. When we load the `janitor` package, it masks (hides) the functions from the `stats` package. This is usually not a problem, but if you want to use the functions from the `stats` package, you can specify the package name when calling the function, like this: `stats::chisq.test()`.

Manipulate the data frame

Functions in the `dplyr` package are used to manipulate data frames:

- `select()`: select columns by position, or by name, or by other methods
- `filter()`: select rows that meet a logical condition
- `slice()`: select rows by position
- `arrange()`: reorder rows
- `mutate()`: add new variables

The `dplyr` package also provides functions to group data frames and to summarize data:

- `group_by()`: add to a data frame a grouping structure
- `summarize()`: summarize data, respecting any grouping structure specified by `group_by()`

The pipe operator `|>` is used to chain together multiple operations on a data frame.

> **Tip**
>
> Note that you will often see another pipe operator `%>%` used in examples. The pipe operator `|>` is a newer version of `%>%` that is more efficient and easier to use. The pipe operator `|>` is available in R version 4.1.0 and later.

Lets work through some examples with a sample data frame:

```
my_data1 <- tibble(
  name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  age = c(25, 30, 35, 40, 45),
  score = c(90, 85, 95, 80, 70))
```

Here is the same dataset with 100 rows:

```
set.seed(123)
my_data2 <- tibble(name = paste0("Person_", sprintf("%03d", 1:100)),
  age = sample(20:50, 100, replace = TRUE),
  score = rnorm(100, mean = 75, sd = 10))
```

Select columns

We can select columns by name

```
my_data2 |>
  select(name, score)
```

```
# A tibble: 100 x 2
   name       score
   <chr>      <dbl>
 1 Person_001  91.9
 2 Person_002  87.3
 3 Person_003  77.8
 4 Person_004  64.5
 5 Person_005  69.8
 6 Person_006  91.2
 7 Person_007  64.3
 8 Person_008  91.9
 9 Person_009  72.6
10 Person_010  70.3
# i 90 more rows
```

We can select columns by position

```
my_data2 |>
  select(1, 3)
```

```
# A tibble: 100 x 2
```

```
   name       score
   <chr>      <dbl>
 1 Person_001  91.9
 2 Person_002  87.3
 3 Person_003  77.8
 4 Person_004  64.5
 5 Person_005  69.8
 6 Person_006  91.2
 7 Person_007  64.3
 8 Person_008  91.9
 9 Person_009  72.6
10 Person_010  70.3
# i 90 more rows
```

We can select columns by a condition, for example select only the numeric columns:

```
my_data2 |>
  select(where(is.numeric))
```

```
# A tibble: 100 x 2
     age score
   <int> <dbl>
 1    50  91.9
 2    34  87.3
 3    38  77.8
 4    33  64.5
 5    22  69.8
 6    29  91.2
 7    37  64.3
 8    41  91.9
 9    30  72.6
10    24  70.3
# i 90 more rows
```

We can select a column by pattern matching, using helper functions, for example select columns that contain the letter "a":

```
my_data2 |>
  select(contains("a"))
```

```
# A tibble: 100 x 2
   name         age
   <chr>      <int>
 1 Person_001    50
 2 Person_002    34
 3 Person_003    38
 4 Person_004    33
```

```
 5 Person_005    22
 6 Person_006    29
 7 Person_007    37
 8 Person_008    41
 9 Person_009    30
10 Person_010    24
# i 90 more rows
```

Other helpers include `starts_with()`, `ends_with()`, `matches()`, and `everything()`.

Filter: Getting particular rows of data

To get particular rows of data, we can use the `filter()` function. This function takes a *logical condition* as an argument and returns only the rows that meet that condition. For example, to get all rows where the Age is greater than 30:

```
my_data2 |>
  filter(age > 30)
```

```
# A tibble: 66 x 3
   name         age score
   <chr>      <int> <dbl>
 1 Person_001    50  91.9
 2 Person_002    34  87.3
 3 Person_003    38  77.8
 4 Person_004    33  64.5
 5 Person_007    37  64.3
 6 Person_008    41  91.9
 7 Person_011    39  67.3
 8 Person_012    33  96.5
 9 Person_013    41  61.7
10 Person_014    44  80.0
# i 56 more rows
```

Here, the logical condition is `age > 30`.

We can combine multiple conditions using the logical operators `&` (and), `|` (or), and `!` (not). For example, to get all rows where the Age is greater than 30 and the Score is less than 90:

```
my_data2 |>
  filter(age > 30 & score < 90)
```

```
# A tibble: 60 x 3
   name         age score
   <chr>      <int> <dbl>
 1 Person_002    34  87.3
 2 Person_003    38  77.8
```

```
 3 Person_004    33  64.5
 4 Person_007    37  64.3
 5 Person_011    39  67.3
 6 Person_013    41  61.7
 7 Person_014    44  80.0
 8 Person_015    45  87.3
 9 Person_016    46  81.3
10 Person_018    38  82.9
# i 50 more rows
```

Other logical operators include `==` (equal to), `!=` (not equal to), `<=` (less than or equal to), and `>=` (greater than or equal to).

Slice: Getting rows by position

The `slice()` function allows us to get rows by their position in the data frame. For example, to get the first two rows:

```
my_data2 |>
  slice(1:2)
```

```
# A tibble: 2 x 3
  name         age score
  <chr>      <int> <dbl>
1 Person_001    50  91.9
2 Person_002    34  87.3
```

I very rarely use this function, as I prefer to use `filter()` with logical conditions. I can't think of a good use case for this function right now! Perhaps you can?

Arrange: Reordering rows

The `arrange()` function allows us to reorder the rows of a data frame based on the values in one or more columns. For example, to reorder the rows by Age in ascending order:

```
my_data2 |>
  arrange(age)
```

```
# A tibble: 100 x 3
   name         age score
   <chr>      <int> <dbl>
 1 Person_064    20  88.8
 2 Person_056    21  79.5
 3 Person_091    21  67.4
 4 Person_005    22  69.8
 5 Person_025    22  74.9
 6 Person_033    23  67.3
```

```
 7 Person_092    23  72.4
 8 Person_010    24  70.3
 9 Person_017    24  79.1
10 Person_058    24  72.7
# i 90 more rows
```

I f we want to reorder the rows by Age in descending order, we can use the `desc()` function:

```
my_data2 |>
  arrange(desc(age))
```

```
# A tibble: 100 x 3
   name         age score
   <chr>      <int> <dbl>
 1 Person_001    50  91.9
 2 Person_061    50  79.9
 3 Person_075    50  67.3
 4 Person_085    50  50.1
 5 Person_096    50  86.8
 6 Person_031    49  71.8
 7 Person_078    49  72.6
 8 Person_024    48  81.2
 9 Person_057    48  80.3
10 Person_021    47  66.0
# i 90 more rows
```

It's unusual to need the rows of a dataset to be arranged in a specific order, but it can be useful when looking at the data directly.

> **Tip**
>
> Note that when you view the data in RStudio, it will always be arranged by the row number. In the viewer you can sort by clicking on the column headers.

Mutate: Adding new variables

The `mutate()` function allows us to add new variables to a data frame. For example, to add a new variable called `Age_in_5_years` that is the Age plus 5:

```
my_data2 |>
  mutate(age_in_5_years = age + 5)
```

```
# A tibble: 100 x 4
   name         age score age_in_5_years
   <chr>      <int> <dbl>          <dbl>
 1 Person_001    50  91.9             55
```

```
 2 Person_002    34  87.3            39
 3 Person_003    38  77.8            43
 4 Person_004    33  64.5            38
 5 Person_005    22  69.8            27
 6 Person_006    29  91.2            34
 7 Person_007    37  64.3            42
 8 Person_008    41  91.9            46
 9 Person_009    30  72.6            35
10 Person_010    24  70.3            29
# i 90 more rows
```

We can add multiple new variables at once:

```
my_data2 |>
  mutate(
    age_in_5_years = age + 5,
    percentage_score = score / 100
  )
```

```
# A tibble: 100 x 5
   name         age score age_in_5_years percentage_score
   <chr>      <int> <dbl>          <dbl>            <dbl>
 1 Person_001    50  91.9             55            0.919
 2 Person_002    34  87.3             39            0.873
 3 Person_003    38  77.8             43            0.778
 4 Person_004    33  64.5             38            0.645
 5 Person_005    22  69.8             27            0.698
 6 Person_006    29  91.2             34            0.912
 7 Person_007    37  64.3             42            0.643
 8 Person_008    41  91.9             46            0.919
 9 Person_009    30  72.6             35            0.726
10 Person_010    24  70.3             29            0.703
# i 90 more rows
```

Grouping and summarising data

The `group_by()` function allows us to add a grouping structure to a data frame, and the `summarise()` function allows us to summarise the data, respecting any grouping structure specified by `group_by()`.

First let us read in a dataset to work with. The dataset is simulated about factors that could be associated with blood pressure

Study design:

- 60 individuals
- Sex assigned at birth: Male / Female
- Treatment: Placebo / Drug
- Exercise program: Yes / No

- Four visits (repeated measures per person)
- Age varies between individuals
- Outcome: systolic blood pressure (mmHg)

Read in and view the data:

```
bp_multifactor <- read_csv("datasets/bp_multifactor.csv")
```

```
Rows: 240 Columns: 7
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (3): sex, treatment, exercise
dbl (4): id, age, visit, systolic_bp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(bp_multifactor)
```

```
Rows: 240
Columns: 7
$ id          <dbl> 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5~
$ sex         <chr> "Female", "Female", "Female", "Female", "Female", "Female"~
$ treatment   <chr> "Drug", "Drug", "Drug", "Drug", "Drug", "Drug", "Drug", "D~
$ exercise    <chr> "No", "No", "No", "No", "No", "No", "No", "No", "Yes", "Ye~
$ age         <dbl> 57, 57, 57, 57, 47, 47, 47, 47, 35, 35, 35, 35, 54, 54, 54~
$ visit       <dbl> 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4~
$ systolic_bp <dbl> 133.2036, 136.9701, 122.2254, 120.0836, 130.7802, 126.2976~
```

First, a simple summarise. We will get the mean blood pressure across all visits and all people:

```
bp_multifactor |>
  summarise(mean_bp = mean(systolic_bp))
```

```
# A tibble: 1 x 1
  mean_bp
    <dbl>
1    131.
```

How does this work? The `summarise()` function creates a new data frame with summary statistics. The argument `mean_bp = mean(systolic_bp)` calculates the mean of the `systolic_bp` variable and names it `mean_bp`. (The mean of all the observations is sometimes called the "grand mean".)

Now let's get the mean blood pressure for each person. To do this, we need to tell the summarise to calculate the mean for each person. We can do this by **adding a grouping structure** to the data frame with the `group_by()` function:

```
bp_multifactor |>
  group_by(id) |>
  summarise(mean_bp = mean(systolic_bp))
```

```
# A tibble: 60 x 2
      id mean_bp
   <dbl>   <dbl>
 1     1    128.
 2     2    127.
 3     3    127.
 4     4    137.
 5     5    128.
 6     6    118.
 7     7    133.
 8     8    120.
 9     9    128.
10    10    119.
# i 50 more rows
```

The `group_by(id)` function adds a grouping structure to the data frame based on the `id` variable. This tells the `summarise()` function to calculate the mean blood pressure for each person (i.e., for each unique value of `id`).

We can see the grouping structure when we look at the output of the `group_by()` function:

```
bp_multifactor |>
  group_by(id)
```

```
# A tibble: 240 x 7
# Groups:   id [60]
      id sex    treatment exercise   age visit systolic_bp
   <dbl> <chr>  <chr>     <chr>    <dbl> <dbl>       <dbl>
 1     1 Female Drug      No          57     1        133.
 2     1 Female Drug      No          57     2        137.
 3     1 Female Drug      No          57     3        122.
 4     1 Female Drug      No          57     4        120.
 5     2 Female Drug      No          47     1        131.
 6     2 Female Drug      No          47     2        126.
 7     2 Female Drug      No          47     3        120.
 8     2 Female Drug      No          47     4        132.
 9     3 Male   Drug      Yes         35     1        133.
10     3 Male   Drug      Yes         35     2        132.
# i 230 more rows
```

The output has the line `# Groups: id [60]`, which indicates that the data frame is now grouped by the `id` variable, and there are 60 unique groups (i.e.,

60 people).

It is that simple :)

We can simultaneously group by multiple variables. For example, to get the mean blood pressure for each combination of sex and treatment. Before we do this, we can check how many unique combinations we think we should have. We already know there are two sexes and two treatments (Placebo and Drug), so we should have four unique combinations, and therefore four means.

```
bp_multifactor |>
  group_by(sex, treatment) |>
  summarise(mean_bp = mean(systolic_bp))
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by sex and treatment.
i Output is grouped by sex.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(sex, treatment))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```

```
# A tibble: 4 x 3
# Groups:   sex [2]
  sex    treatment mean_bp
  <chr>  <chr>       <dbl>
1 Female Drug         126.
2 Female Placebo      134.
3 Male   Drug         132.
4 Male   Placebo      138.
```

Perfect.

We can also calculate multiple summary statistics at once. For example, to get the mean and standard deviation of blood pressure for each combination of sex and treatment:

```
bp_multifactor |>
  group_by(sex, treatment) |>
  summarise(
    mean_bp = mean(systolic_bp),
    sd_bp = sd(systolic_bp)
  )
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by sex and treatment.
i Output is grouped by sex.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(sex, treatment))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```

```
# A tibble: 4 x 4
# Groups:   sex [2]
  sex    treatment mean_bp sd_bp
  <chr>  <chr>       <dbl> <dbl>
1 Female Drug         126.  8.55
2 Female Placebo      134.  6.90
3 Male   Drug         132.  7.41
4 Male   Placebo      138.  7.70
```

Working with categorical variables

Variables in a data frame in R have a *type*. The most common types of variables are numeric and categorical. Numeric variables are variables that take on numerical values, such as age or score. Categorical variables are variables that take on a limited number of values, often representing categories or groups. In R, categorical variables are typically have *type* `<chr>` which is `character`. Or they can be of type `<fct>` which is `factor`.

When we import data categorical variable is usually imported as a `character` variable. For example, the variable `name` in our example dataset is a categorical variable of type `character`. Look at the first few rows of the dataset again, and see that below the variable name it says `<chr>` for the `name` variable:

```
my_data2
```

```
# A tibble: 100 x 3
   name         age score
   <chr>      <int> <dbl>
 1 Person_001    50  91.9
 2 Person_002    34  87.3
 3 Person_003    38  77.8
 4 Person_004    33  64.5
 5 Person_005    22  69.8
 6 Person_006    29  91.2
 7 Person_007    37  64.3
 8 Person_008    41  91.9
 9 Person_009    30  72.6
10 Person_010    24  70.3
# i 90 more rows
```

This is all totally fine. There are, however, use cases where we might want to convert a `character` variable to a `factor` variable. Factors are useful when we have a categorical variable with a fixed number of levels, and we want to specify the order of those levels. For example, if we had a variable called `education_level` with the values "High School", "Bachelor's", "Master's", and "PhD", we might want to convert this variable to a factor and specify the order of the levels.

Let's make a new dataset to illustrate this:

```
my_data3 <- tibble(
  name = c("Alice", "Bob", "Charlie", "David", "Eve"),
  education_level = c("Bachelor's", "Master's", "PhD", "High School", "Bachelor's"),
  age = c(19, 23, 25, 16, 20)
)
```

Look at the structure of this new dataset:

```
my_data3
```

```
# A tibble: 5 x 3
  name    education_level   age
  <chr>   <chr>           <dbl>
1 Alice   Bachelor's         19
2 Bob     Master's           23
3 Charlie PhD                25
4 David   High School        16
5 Eve     Bachelor's         20
```

We can see that the `education_level` variable is of type `<chr>`, which is `character`.

We can convert the `education_level` variable to a factor:

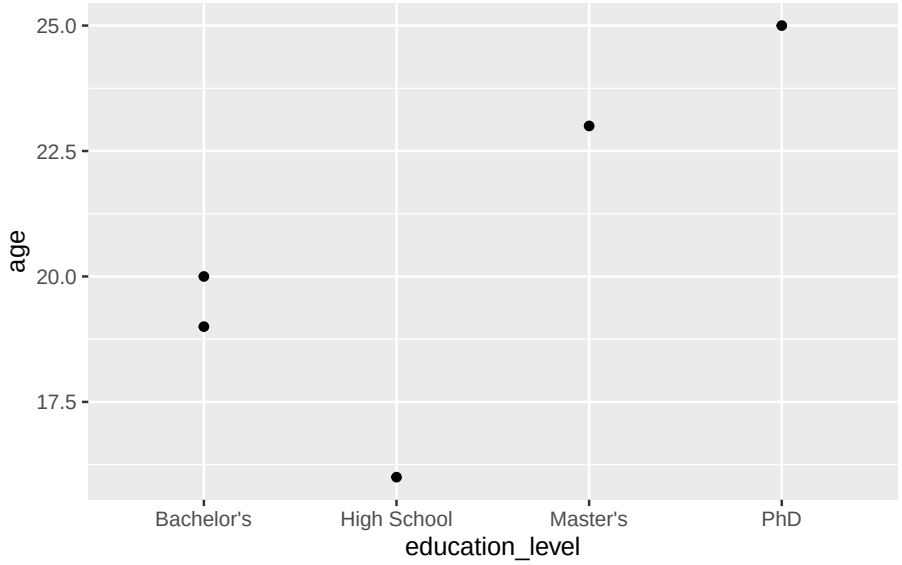
```
my_data3 <- my_data3 |>
  mutate(education_level = factor(education_level))
my_data3
```

```
# A tibble: 5 x 3
  name    education_level   age
  <chr>   <fct>           <dbl>
1 Alice   Bachelor's         19
2 Bob     Master's           23
3 Charlie PhD                25
4 David   High School        16
5 Eve     Bachelor's         20
```

Now, the `education_level` variable is of type `<fct>`, which is `factor`.

Here is a graph of age by education level:

```
ggplot(my_data3, aes(x = education_level, y = age)) +
  geom_point()
```



We have a problem here: the education levels are not in a sensible order. The first level is "Bachelor's", followed by "High School", "Master's", and "PhD".

> **Note**
>
> Why do you think the levels are in this order? We didn't tell R to order them like this! The answer is that R orders factor levels alphabetically by default. So when we convert a character variable to a factor without specifying the order of the levels, R will order them alphabetically.

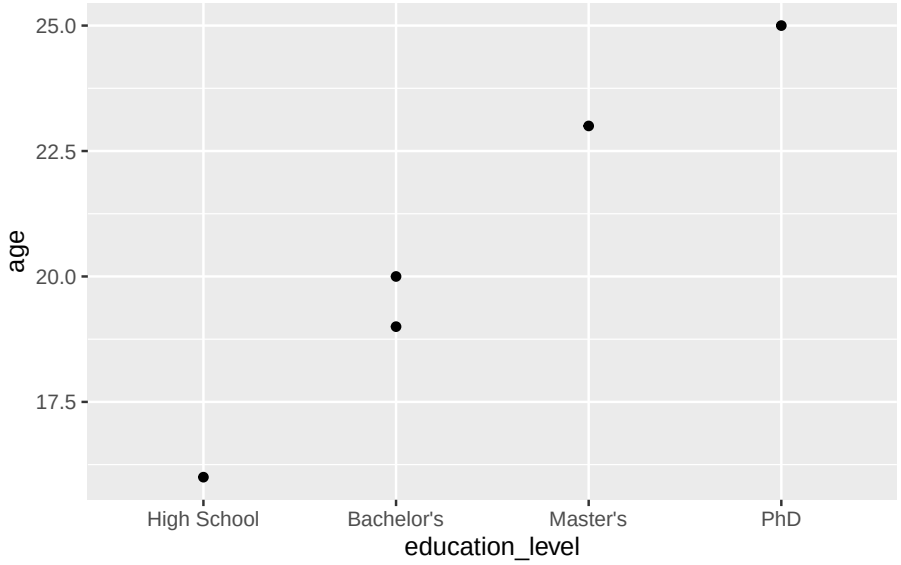
It would be much better to have the levels ordered as "High School", "Bachelor's", "Master's", and "PhD".

We can fix this by specifying the order of the levels when we convert the variable to a factor:

```
my_data3 <- my_data3 |>
  mutate(education_level = factor(education_level,
                                  levels = c("High School", "Bachelor's", "Master's",
```

Now when we plot the data again, the education levels are in the correct order:

```
ggplot(my_data3, aes(x = education_level, y = age)) +
  geom_point()
```



Another use case is when we are making a linear model and want to specify the reference level for a categorical variable. We will look at this when we get to linear models. If you want to skip ahead, you can see how this works in a section at the end of this chapter.

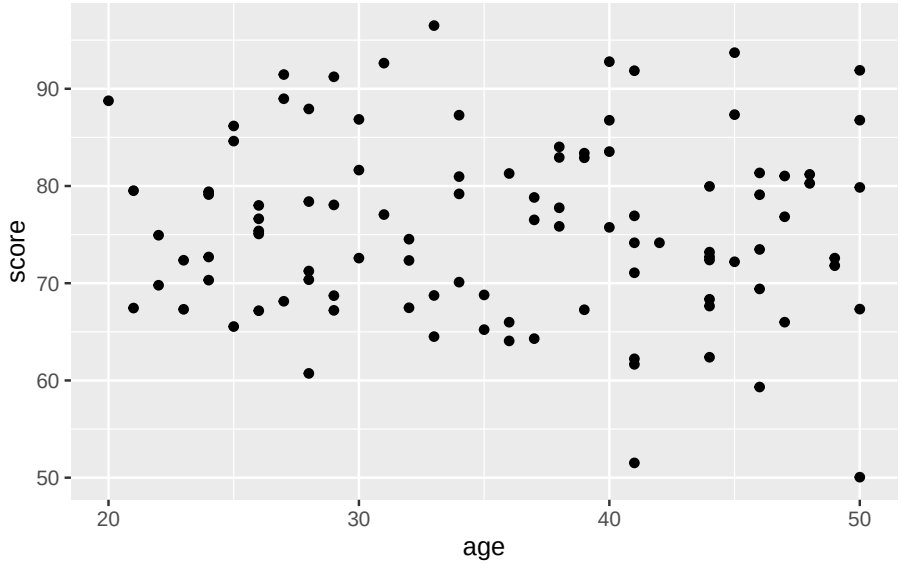
Visualisation

There are many many many types of data visualisation. We will not explore them all in this course! In fact, we will use only a few basic types of visualisation, but we will use them well and critically. The three types of visualisation we will focus on are scatter plots, histograms, and box and whisker plots.

Three basic types of visualisation

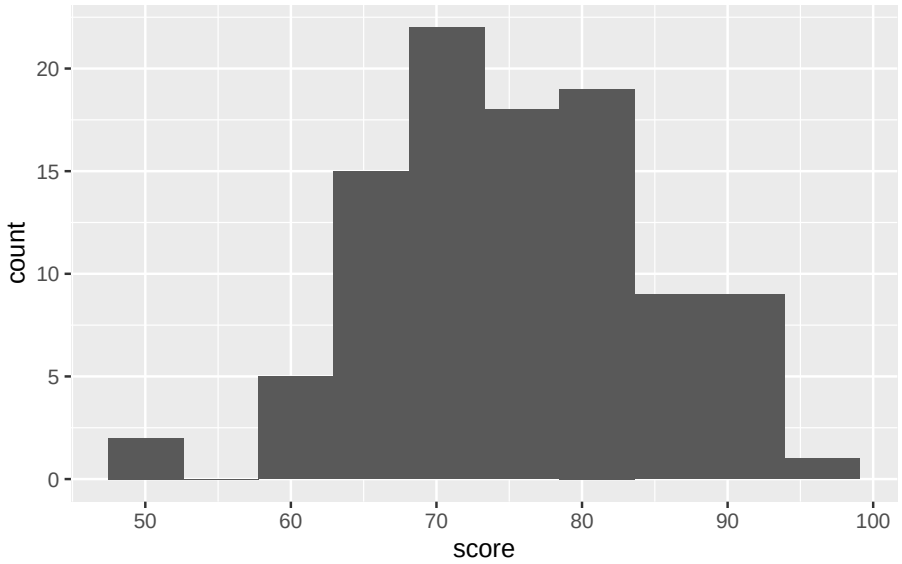
Scatterplots are used to visualise the relationship between two continuous variables. Here is an example of a scatterplot:

```
library(ggplot2)
ggplot(my_data2, aes(x = age, y = score)) +
  geom_point()
```



Histograms are used to visualise the distribution of a single continuous variable. The axiss are different to scatterplots: the x-axis is the variable being measured, and the y-axis is the count (or frequency) of observations in each bin. A bin is a range of values. Here is an esample of a histogram:

```
ggplot(my_data2, aes(x = score)) +
  geom_histogram(bins = 10)
```

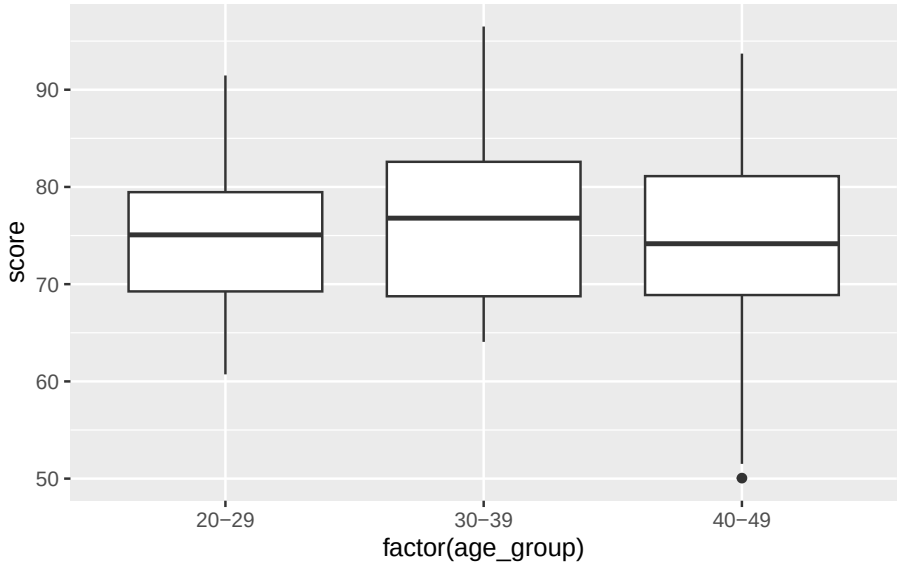


Box and whisker plots are used to visualise the distribution of a continuous variable across different categories. Here is an example of a box and whisker plot. First we add a new variable that is age group:

```
my_data2 <- my_data2 |>
  mutate(age_group = case_when(
    age < 30 ~ "20-29",
    age >= 30 & age < 40 ~ "30-39",
    age >= 40 ~ "40-49"
  ))
```

The new variable `age_group` is a categorical variable with three levels: "20-29", "30-39", and "40-49". We make this using the `case_when()` function. This function works by checking each condition (which are given as the arguments to the function) in turn, and assigning the corresponding value when the condition is true. Now we can make the box and whisker plot:

```
ggplot(my_data2, aes(x = factor(age_group), y = score)) +
  geom_boxplot()
```



Understanding ggplot2 syntax

We have used the `ggplot2` package to create visualisations. The `ggplot2` package is based on the grammar of graphics, which provides a consistent way to create visualisations. It is amazing, and when it was created it revolutionised data visualisation in R.

You can see that for each of the three visualisations, we use the `ggplot()`

function to create the base plot, and then we add layers to the plot using the `+` operator.

The first argument to the `ggplot()` function is the data frame that we want to visualise. The layers that we add to the plot each have two main components. The first component is the *aesthetic mappings*, which specify how the variables in the data frame are mapped to the visual properties of the plot (e.g., x-axis, y-axis, color, size). The second component is the *geometric object*, which defines how the data is represented in the plot (e.g., points, lines, bars).

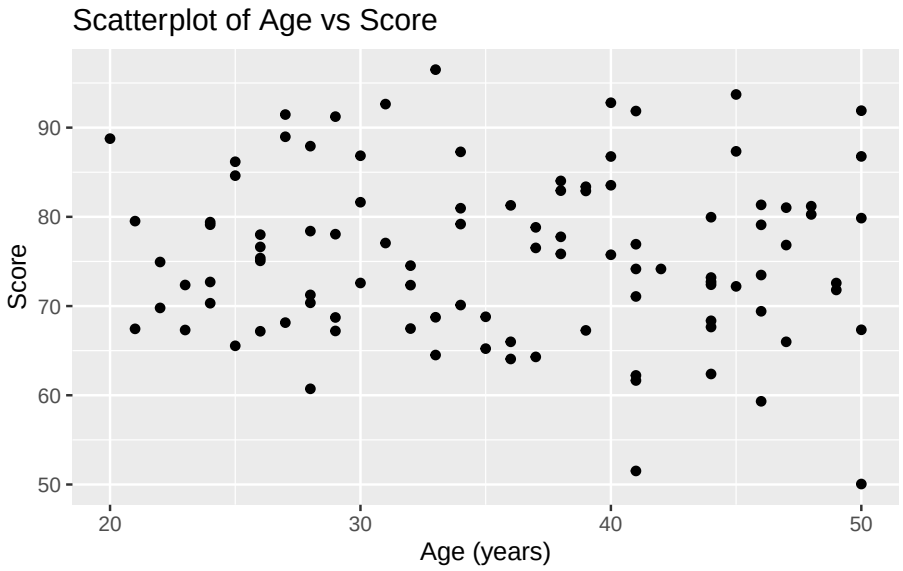
The *aesthetic mappings* are specified using the `aes()` function, which takes arguments that define the mappings. Inside the `aes()` function, we specify the variables from the data frame that we want to map to the visual properties of the plot. For example, in the scatterplot, we map the `age` variable to the x-axis and the `score` variable to the y-axis using `aes(x = age, y = score)`.

The *geometric object* is specified using functions that start with `geom_`, such as `geom_point()`, `geom_histogram()`, and `geom_boxplot()`.

You will notice that for the scatterplot and the box and whisker plot, we specify both an x- and a y-variable, but for the histogram we only specify an x-variable. This is because histograms only have one variable, which is the variable being measured. The y-axis is automatically calculated as the count (or frequency) of observations in each bin.

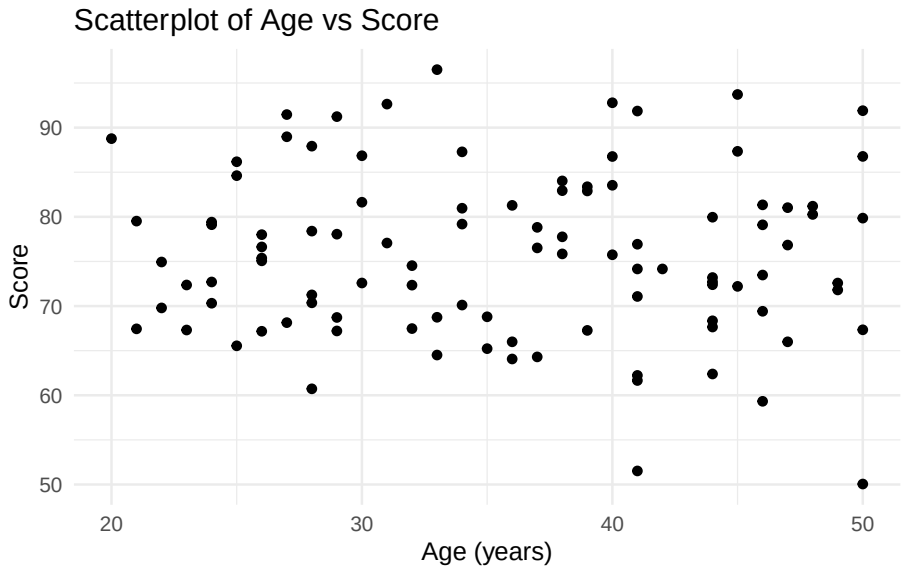
We can customise many features of the graph using additional arguments to the `ggplot()` function and the `geom_` functions. For example, we can add titles and labels to the axes using the `labs()` function:

```
ggplot(my_data2, aes(x = age, y = score)) +
  geom_point() +
  labs(
    title = "Scatterplot of Age vs Score",
    x = "Age (years)",
    y = "Score"
  )
```



We can also change the theme of the plot using the `theme_` functions. For example, to use a minimal theme, and add it the customisations we already made:

```
ggplot(my_data2, aes(x = age, y = score)) +
  geom_point() +
  labs(
    title = "Scatterplot of Age vs Score",
    x = "Age (years)",
    y = "Score"
  ) +
  theme_minimal()
```



There are a million and one ways to customise visualisations in `ggplot2`. We will explore many of them during the course in a rather ad-hoc way. In this course we do not *assess* your skill and competence in making clear and beautiful visualisations. We will, however, be very happy to help you make beautiful and effective visualisations for your assignments and projects. And please be sure that making beautiful and effective visualisations is a skill that is very highly valued in the workplace.

Saving ggplot visualisations

Another feature that is very useful is to save ggplot visualisations to objects and then save to a file (for example a pdf). First, here is how we save a ggplot to an object:

```
plot1 <- ggplot(my_data2, aes(x = age, y = score)) +
  geom_point() +
  labs(
    title = "Scatterplot of Age vs Score",
    x = "Age (years)",
    y = "Score"
  )
```

Now we can save the plot to a file using the `ggsave()` function:

```
ggsave("scatterplot_age_vs_score.pdf", plot = plot1, width = 8, height = 6)
```

Note two things about the `ggsave()` function. First, the first argument is the file name (including the file extension). The file extension determines the file

type (e.g., pdf, png, jpeg). Second, we can specify the width and height of the plot in inches.

Also note that the file is saved to the current working directory. When you're working in an R project, this is usually the base directory of the project. If you want to save your plots in a folder named `plots` you would first need to create the folder (if it doesn't already exist) and then specify the path in the file name:

```
# Use argument showWarnings = FALSE, so no warning when the folder already exists
dir.create("plots", showWarnings = FALSE)  # Create the folder if it doesn't exist
ggsave("plots/scatterplot_age_vs_score.pdf", plot = plot1, width = 8, height = 6)
```

Review

In this chapter we learned about using R and RStudio for data analysis. We covered the basics of R programming, including data types, variables, functions, and packages. We also learned about importing data into R, cleaning and wrangling data using the `dplyr` package, and visualising data using the `ggplot2` package. A great starting point for your journey into data analysis with R!

Further reading

The data wrangling packages we use, such as `dplyr` and `tidyr`, are part of the *tidyverse*, a collection of R packages designed for data science. These packages were developed by Hadley Wickham and his team at RStudio. Moreover, Hadley Wickham is also the creator of `ggplot2`, the package we use for data visualisation.

A great place to consolidate your learning so far, and to learn more about the *tidyverse* and data science with R is the book *R for Data Science* by Hadley Wickham and Garrett Grolemund (https://r4ds.had.co.nz/). This book is available for free online and covers many topics related to data science with R, including:

- Scripts and projects
- Data import
- Data wrangling with dplyr and tidyr
- Data visualisation with ggplot2
- And much more…

Base R is a term used to describe the core functionality of R, without any additional packages. While the *tidyverse* packages are very useful and powerful, it can be useful to learn about the base R functions, especially if someone else is using base R in their code. A starting point for learning about base R is the chapter "A field guide to base R" in the *R for Data Science* book – A field guide to base R.

Cheat sheets

There are a series of excellent cheat sheets for R and RStudio, and for packages in the *tidyverse*. They're pretty dense and packed with information, but they can be very useful as a quick reference. You can find them here: RStudio Cheat Sheets.

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Joining data frames

In this chapter we have only worked with one data frame at a time. However, in real life you will often have multiple data frames that you need to join together. For example, you might have one data frame with demographic information about people (e.g., age, sex, education level) and another data frame with information about their health outcomes (e.g., blood pressure, cholesterol levels). You can use the `dplyr` package to join these data frames together using functions like `left_join()`, `right_join()`, `inner_join()`, and `full_join()`. We won't cover these during the course, but you can read about them in the *R for Data Science* book – Joining data frames.

Making reports directly using Quarto

We don't explicitly ask you to make reports using Quarto in this course, but it is a very useful skill to have, and I highly recommend you explore it further in your own time. Here are a few basics to get you started.

One of the great features of R and RStudio is the ability to create reports that combine text, code, and visualisations. One of the most popular tools for this is Quarto (https://quarto.org/), which allows you to create documents in various formats (HTML, PDF, Word, etc.) using a combination of *Markdown* and R code.

**Why is this so great???* If you want to show someone your analysis and visualisation, say a team member or supervisor, it is often good to prepare a report that explains what you did, perhaps shows the code you used, and presents the results (including visualisations). One way to go about this is to prepare a powerpoint presentation or a word document, and then copy and paste code and visualisations into the document. Its what I used to do. It works. But it is tedious, error prone, and when you change something in your code or data, you have to remember to go back and update the powerpoint or word document.

With Quarto, you can create a report that automatically includes the code and visualisations directly from your R script. This way, if you change the code or data, you can simply re-render the report and everything is automatically updated. It takes away a lot of the tediousness and potential for errors. And it makes updating reports much easier.

If you'd like to get started with Quarto, check out the Quarto website (https://quarto.org/) and the RStudio Quarto documentation (https://quarto.org/docs/get-started/). There are also many tutorials and resources available online to help you learn how to use Quarto effectively.

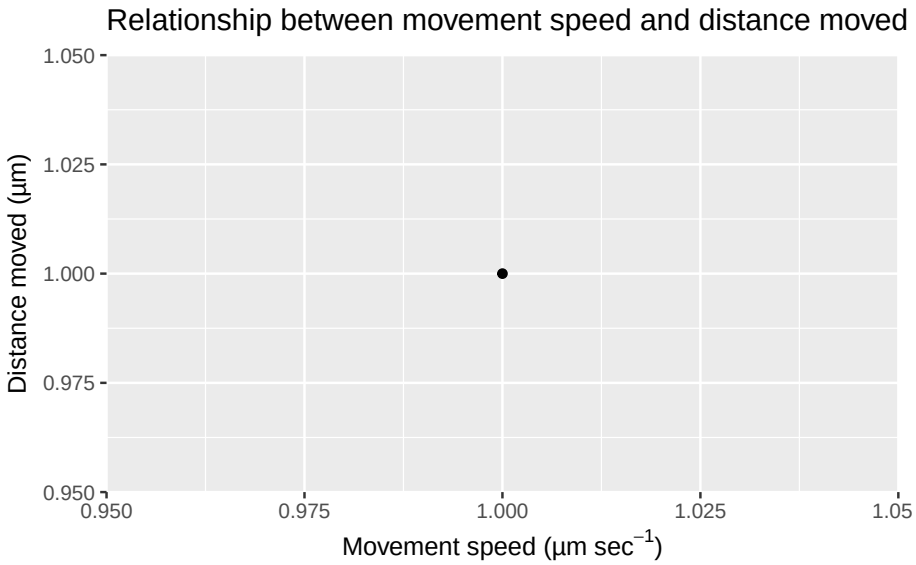
If you have questions about Quarto, feel free to ask me or TAs during the practicals, though note that any particular TAs may or may not be experienced with Quarto themselves.

Quarto reports are also covered in the *R for Data Science* book – Quarto.

Mathematical notation in ggplot2

When creating visualisations with `ggplot2`, you may want to include mathematical notation in your titles, axis labels, or legends. This can be done using the `latex2exp` package, which allows you to use LaTeX syntax to create mathematical expressions in your plots. Here is an example of how to use `latex2exp` to include mathematical notation in a ggplot:

```
ggplot(mapping = aes(x = 1, y = 1)) +
  geom_point() +
  labs(x = latex2exp::TeX("Movement speed ($\mu m$ $sec^{-1}$)"),
       y = latex2exp::TeX("Distance moved ($\mu m$)"),
       title = latex2exp::TeX("Relationship between movement speed and distance moved"))
```



Combining ggplots with patchwork

We often make multiple ggplots in our analyses. Sometimes it is useful to combine multiple plots into a single figure for easier comparison or presentation. We can do with ggplots and the lovely add-on package called `patchwork`. The `patchwork` package allows us to combine multiple ggplots into a single plot layout. Here is an example of how to use `patchwork` to combine the three plots we made earlier (scatterplot, histogram, and boxplot):

First, load the `patchwork` package:

```
library(patchwork)
```

Next make the first plot and assign it to an object:

```
plot1 <- ggplot(my_data2, aes(x = age, y = score)) +
  geom_point() +
  labs(
    title = "Scatterplot of Age vs Score",
    x = "Age (years)",
    y = "Score"
  )
```

Now make the second plot and assign it to an object:

```
plot2 <- ggplot(my_data2, aes(x = score)) +
  geom_histogram(binwidth = 5) +
  labs(
    title = "Histogram of Scores",
```

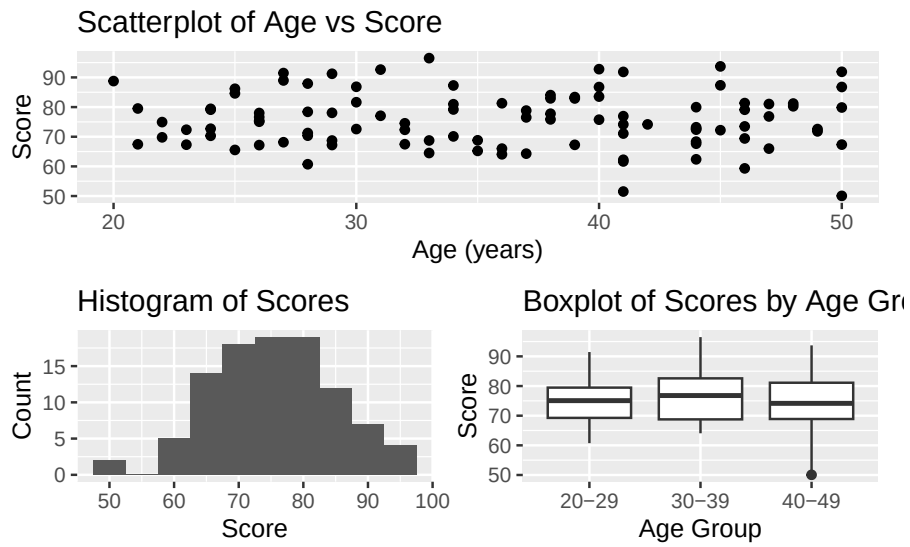
```
    x = "Score",
    y = "Count"
  )
```

Now make the third plot and assign it to an object:

```
plot3 <- ggplot(my_data2, aes(x = factor(age_group), y = score)) +
  geom_boxplot() +
  labs(
    title = "Boxplot of Scores by Age Group",
    x = "Age Group",
    y = "Score"
  )
```

Now we can combine the three plots into a single layout using the `patchwork` syntax. Here, we arrange `plot1` on the top row, and `plot2` and `plot3` side by side on the bottom row:

```
combined_plot <- plot1 / (plot2 | plot3)
combined_plot
```



Amazing eh! OK, lets leave it there for now. We'll use ggplot2 throughout the course, and explore more features as we go along.

If you'd like to read more about the patchwork package, check out the package website – patchwork.

Setting a reference level in a linear model

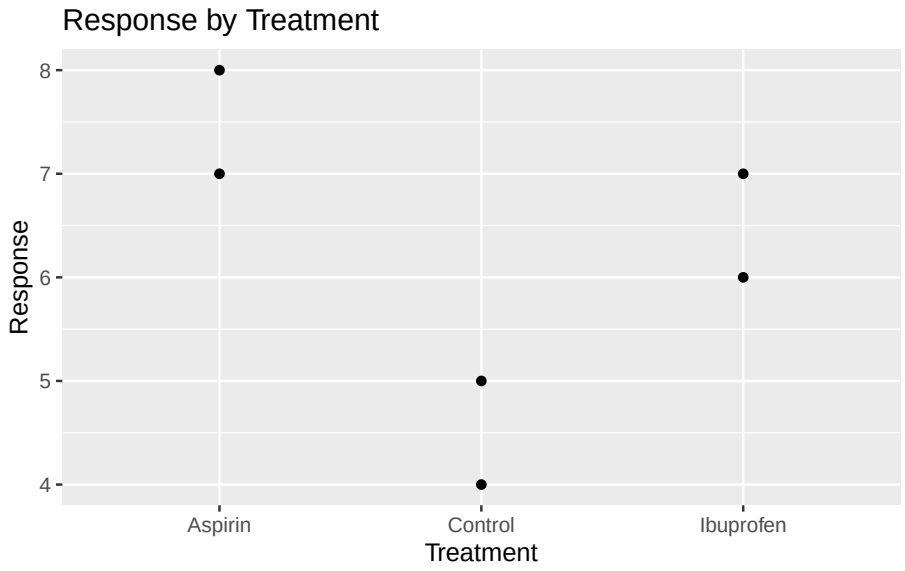
Sometimes when fitting linear models with categorical explanatory (independent) variables, it is useful to set a specific reference level for the categorical variable. This can help in interpreting the model coefficients. In R, we can set the reference level using the `relevel()` function or by using the `factor()` function with the `levels` argument.

First, let's create a simple dataset:

```
my_data4 <- tibble(
  treatment = factor(c("Control", "Aspirin", "Ibuprofen", "Control", "Aspirin", "Ibupr
  response = c(5, 7, 6, 4, 8, 7)
)
```

By default, R will set the first level of the factor (in alphabetical order) as the reference level. In this case, "Aspirin" would be the reference level. Therefore when we visualise the data:

```
ggplot(my_data4, aes(x = treatment, y = response)) +
  geom_point() +
  labs(
    title = "Response by Treatment",
    x = "Treatment",
    y = "Response"
  )
```



It would be nicer to have the "Control" group as the first level on the left of the x-axis.

Likewise, when we make a linear model:

```
m_treatment_default_ref <- lm(response ~ treatment, data = my_data4)
summary(m_treatment_default_ref)
```

```

Call:
lm(formula = response ~ treatment, data = my_data4)

Residuals:
   1    2    3    4    5    6
 0.5 -0.5 -0.5 -0.5  0.5  0.5

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)          7.5000     0.5000  15.000 0.000643 ***
treatmentControl    -3.0000     0.7071  -4.243 0.023981 *
treatmentIbuprofen  -1.0000     0.7071  -1.414 0.252215
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7071 on 3 degrees of freedom
Multiple R-squared:  0.8615,    Adjusted R-squared:  0.7692
F-statistic: 9.333 on 2 and 3 DF,  p-value: 0.05152
```

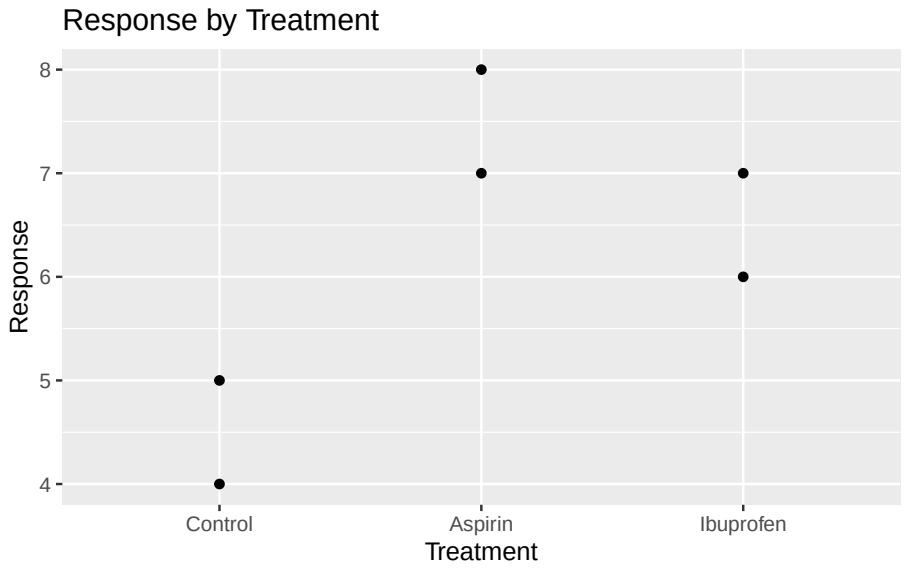
The (Intercept) term corresponds to the "Aspirin" group, and the coefficients for "Control" and "Ibuprofen" are relative to "Aspirin". *R* has done this because in the factor levels, "Aspirin" comes first alphabetically and was therefore set as the reference level when the factor variable was created.

If we want to set "Control" as the reference level, we can do so using `relevel()`:

```
my_data4 <- my_data4 |>
  mutate(treatment = relevel(treatment, ref = "Control"))
```

Now when we visualise the data again:

```
ggplot(my_data4, aes(x = treatment, y = response)) +
  geom_point() +
  labs(
    title = "Response by Treatment",
    x = "Treatment",
    y = "Response"
  )
```



Magic! The "Control" group is now the first level on the left of the x-axis.

And when we fit the linear model again:

```
m_treatment_control_ref <- lm(response ~ treatment, data = my_data4)
summary(m_treatment_control_ref)
```

```
Call:
lm(formula = response ~ treatment, data = my_data4)

Residuals:
   1    2    3    4    5    6
 0.5 -0.5 -0.5 -0.5  0.5  0.5

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)          4.5000     0.5000   9.000   0.0029 **
treatmentAspirin     3.0000     0.7071   4.243   0.0240 *
treatmentIbuprofen   2.0000     0.7071   2.828   0.0663 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7071 on 3 degrees of freedom
Multiple R-squared:  0.8615,    Adjusted R-squared:  0.7692
F-statistic: 9.333 on 2 and 3 DF,  p-value: 0.05152
```

The (Intercept) term now corresponds to the "Control" group, and the coefficients

for "Aspirin" and "Ibuprofen" are relative to "Control". This makes interpretation of the model coefficients more intuitive.

Also see the chapter about factors in the R for Data Science book – Factors.

Finding observations with extreme values

Imagine we have a dataset about people's ages and their blood pressure, so that the dataset has three variables: `id`, `age`, and `blood_pressure`. We might want to find the `id` of people with extreme values of blood pressure, or age. How can we do this? And do this using code?

The first step is to define a criterion for what we consider to be an extreme value. For example, we might define an extreme value as being more than 3 standard deviations away from the mean. This is equivalent to saying that we define extreme value as ones with a z-score greater than 3 or less than -3. Here, a z-score is calculated as (value - mean) / standard deviation.

Let's do this in code, and make the code as general (i.e., works for other datasets) as possible. Let's say our dataset is called `my_data`, and it has three variables: `ID`, and two other variables

```
my_data |>
  pivot_longer(cols = !matches("ID"),
               names_to = "var_name",
               values_to = "var_value") |>
  group_by(var_name) |>
  mutate(temp = var_value - mean(var_value),
         z_val = temp / sd(temp)) |>
  filter(z_val < -3 | z_val > 3) |>
  pull(ID) |>
  unique()
```

How does this work?

First, we use `pivot_longer()` to reshape the data from wide format to long format. This allows us to work with all the variables (except ID) in a consistent way. The `matches("ID")` function is used to specify that we want to keep the ID variable as is, and pivot the other variables.

Second, we use `group_by(var_name)` to group the data by variable name. This allows us to calculate the mean and standard deviation for each variable separately.

Third, we use `mutate()` to calculate the z-score for each value of each variable. We first calculate the difference between the value and the mean (temp), and then divide by the standard deviation to get the z-score (z_val).

Fourth, we use `filter()` to keep only the rows where the z-score is less than -3 or greater than 3, which corresponds to our definition of extreme values.

Finally, we use `pull(ID)` to extract the ID variable from the filtered data, and `unique()` to get the unique IDs of people with extreme values.

Regression Part 1 (L3)

Learning objectives

After this unit, students will be able to:

1. Explain when simple linear regression is an appropriate tool for a biological question.
2. Write the simple linear regression model as an equation, and identify the intercept, slope and error term.
3. Explain the least squares principle by which the intercept and slope are estimated.
4. Calculate fitted values and residuals from a regression equation and data.
5. Explain what degrees of freedom are: the number of independent pieces of information left for estimating variation after the model parameters have been estimated, so that the residual degrees of freedom equal the number of observations minus the number of estimated parameters.
6. Explain why the residual variance of a simple linear regression is estimated by dividing the residual sum of squares by n − 2, and calculate the residual degrees of freedom.
7. Explain why few residual degrees of freedom are a problem: imprecise estimates of the residual variance, larger critical t-values and so wider confidence intervals and less power, a risk of fitting noise, and difficulty checking assumptions.
8. Describe how the error term is modelled: normally distributed, with mean zero and constant variance.
9. State the assumptions of simple linear regression.
10. Use R to fit a simple linear regression with `lm()` and obtain the estimated intercept and slope.
11. Use diagnostic plots to judge whether the assumptions of a linear regression are reasonably met, and identify which assumption is violated if not.
12. Explain leverage and Cook's distance, and identify influential obser-

vations.
13. Identify common problems in a regression (e.g. non-linearity, non-constant variance, influential observations) and select an appropriate remedy (e.g. a transformation).
14. Interpret the coefficients of a regression with a log-transformed response and/or explanatory variable (e.g. as proportional or multiplicative effects).
15. Explain what makes a model linear (linear in its parameters), so that models with transformed variables can still be fitted with `lm()`.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch03_regression_part1.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 3 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch03_regression_part1.R")
```

Video walkthrough: coming soon.

Introduction

Linear regression is a common statistical method that models the relationship between a dependent (response) variable and one or more independent (explanatory) variables. The relationship is modeled with the equation for a straight line ($y = a + bx$).

With linear regression we can answer questions such as:

- How does the dependent (response) variable change with respect to the independent (explanatory) variable?
- What amount of variation in the dependent variable can be explained by the independent variable?
- Is there a statistically significant relationship between the dependent variable and the independent variable?

- Does the linear model fit the data well?

In this chapter / lesson we will explore what is linear regression and how to use it to answer these questions. We'll cover the following topics:

- Why use linear regression?
- What is the linear regression model?
- Fitting the regression model (= finding the intercept and the slope).
- Is linear regression a good enough model to use?
- What do we do when things go wrong?
- Transformation of variables/the response.
- Identifying and handling odd data points (aka outliers).

In this chapter / lesson we will not discuss the statistical significance of the model. We will cover this topic in the next chapter / lesson.

Why use linear regression?

- It's a good starting point because it is a relatively simple model.
- Relationships are sometimes close enough to linear.
- It's easy to interpret.
- It's easy to use.
- It's actually quite flexible (e.g. can be used for non-linear relationships, e.g., a quadratic model is still a linear model!!!).

An example - blood pressure and age

There are lots of situations in which linear regression can be useful. For example, consider hypertension. Hypertension is a condition in which the blood pressure in the arteries is persistently elevated. Hypertension is a major risk factor for heart disease, stroke, and kidney disease. It is estimated that hypertension affects about 1 billion people worldwide. Hypertension is a complex condition that is influenced by many factors, including age. In fact, it is well known that blood pressure increases with age. But how much does blood pressure increase with age? This is a question that can be answered using linear regression.

Here is an example of a study that used linear regression to answer this question Association of age and blood pressure among 3.3 million adults: insights from China PEACE million persons project (2021) *The China PEACE Collaborative Group*

In this study, the authors used linear regression to model the relationship between age and blood pressure. They found that systolic blood pressure increased by 0.28–0.85 mmHg/year. This is a small increase, but it is statistically significant. This means that the observed relationship between age and blood pressure is unlikely to be due to chance.

Lets look at some simulated example data:

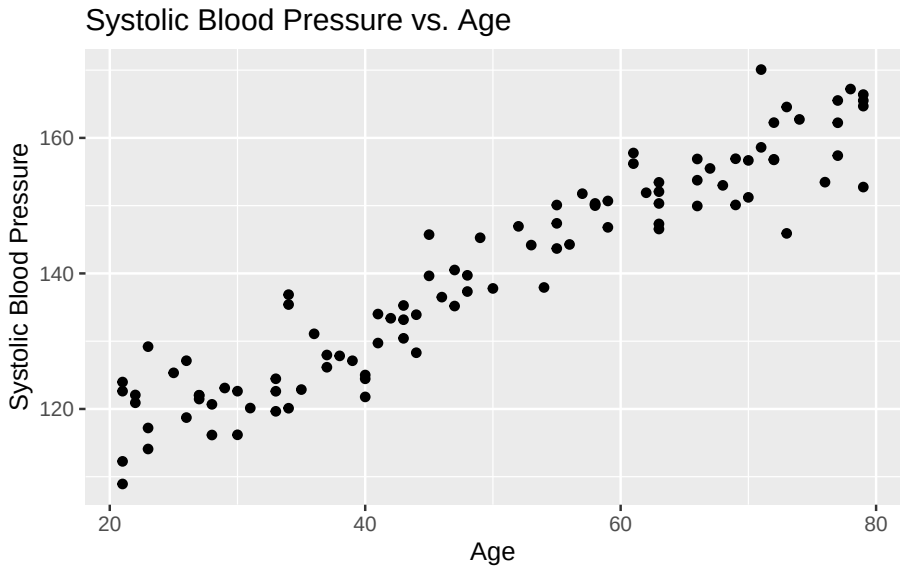
```
# Load the data
bp_age_data <- read_csv("datasets/Simulated_Blood_Pressure_and_Age_Data.csv")
```

```
Rows: 100 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): Age, Systolic_BP

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# How many data points with both age and blood pressure?
bp_age_data <- na.omit(bp_age_data)
# No rows with missing values

# Visualize the data
ggplot(bp_age_data, aes(x = Age, y = Systolic_BP)) +
  geom_point() +
  labs(title = "Systolic Blood Pressure vs. Age",
       x = "Age",
       y = "Systolic Blood Pressure")
```



Well, that is pretty conclusive. We hardly need statistics. There is a clear positive relationship between age and systolic blood pressure. But how can we quantify this relationship? And in less clear-cut cases what is the strength of evidence for a relationship? This is where linear regression comes in. Linear regression models the relationship between age and systolic blood pressure. With

linear regression we can answer the following questions:

- What is the value of the intercept and slope of the relationship?
- Is the relationship different from what we would expect if there were no relationship?
- How well does the mathematical representation match the observed values?
- How much uncertainty is there in predictions?

Lets try to figure some of these out from the visualisation.

Calculating the intercept and slope

Regression from a mathematical perspective

Given an **independent/explanatory variable** (X) and a **dependent/response variable** (Y) all points (x_i, y_i), $i = 1, \ldots, n$, on a straight line follow the equation

$$y_i = \beta_0 + \beta_1 x_i \ .$$

- β_0 is the **intercept** - the value of Y when $x_i = 0$
- β_1 the **slope** of the line, also known as the regression coefficient of X.
- If $\beta_0 = 0$ the line goes through the origin $(x, y) = (0, 0)$.
- **Interpretation** of linear dependency: proportional increase in y with increase (decrease) in x.

Finding the intercept and the slope

In a regression analysis, one task is to estimate the intercept and the slope. These are known as the **regression coefficients** β_0, β_1.

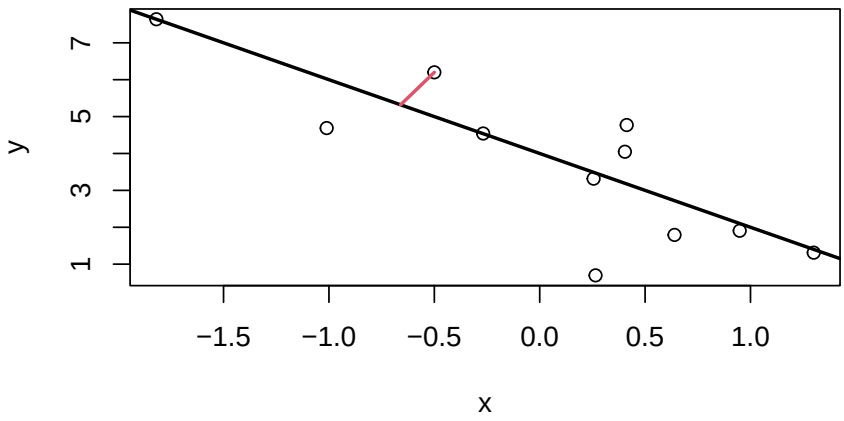
- **Problem**: For more than two points (x_i, y_i), $i = 1, \ldots, n$, there is generally no perfectly fitting line.

- **Aim:** We want to estimate the parameters (β_0, β_1) of the **best fitting** line $Y = \beta_0 + \beta_1 x$.

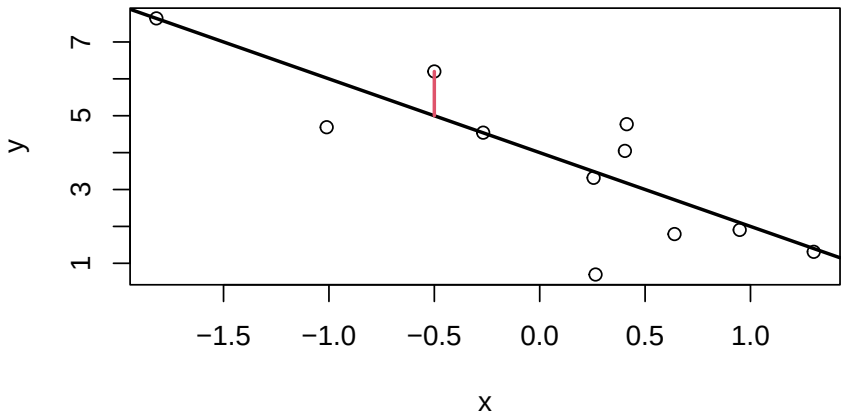
- **Idea:** Find the **best fitting line** by minimizing the deviations between the data points (x_i, y_i) and the regression line. I.e., minimising the residuals.

But which deviations?

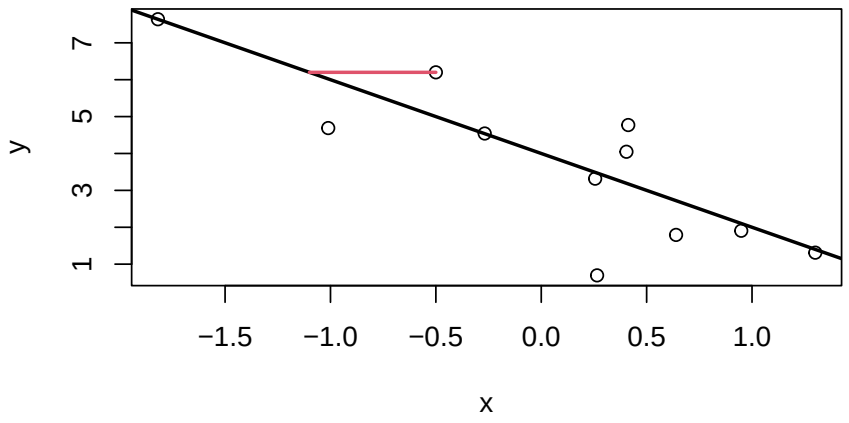
These ones?



Or these?



Or maybe even these?



Well, actually its none of these!!!

Least squares

For multiple reasons (theoretical aspects and mathematical convenience), the intercept and slope are estimated using the **least squares** approach. In this, yet something else is minimized:

The parameters β_0 and β_1 are estimated such that the **sum of squared vertical distances** (sum of squared residuals / errors) is minimised.

SSE means **S**um of **S**quared **E**rrors:

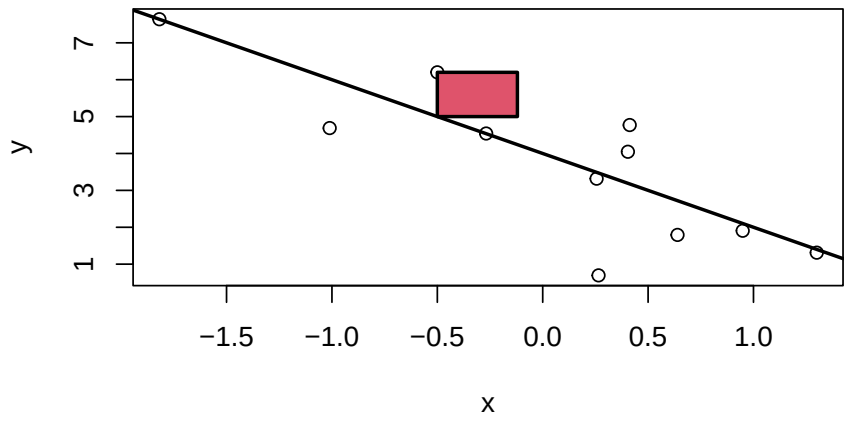
$$SSE = \sum_{i=1}^{n} e_i^2$$

where,

$$e_i = y_i - \underbrace{(\beta_0 + \beta_1 x_i)}_{=\hat{y}_i}$$

Note: $\hat{y}_i = \beta_0 + \beta_1 x_i$ are the *predicted values*.

In the graph just below, one of these squares is shown in red.



> **Note**
>
> Residuals are model-based quantities, not properties of the raw data.

Least squares estimates

With a linear model, we can calculate the least squares estimates of the parameters β_0 and β_1 directly using the following formulas.

For a given sample of data $(x_i, y_i), i = 1, .., n$, with mean values $\overline{x}$ and $\overline{y}$, the least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are computed as

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(y_i - \overline{y})(x_i - \overline{x})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} = \frac{cov(x, y)}{var(x)}$$

This formula for $\hat{\beta}_1$ can be interpreted as the ratio of the covariance between x and y to the variance of x. It captures how much y changes on average for a one-unit change in x. The derivation of this formula involves calculus and the minimization of the sum of squared residuals, but the key point is that it provides a direct (analytical) way to calculate the slope of the best-fitting line.

The intercept $\hat{\beta}_0$ can then be calculated using the formula:

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x}$$

Moreover, we can also calculate a measure of the residual variance. This is a measure of how much variability is not explained by the model. The formula for the residual variance is:

$$\hat{\sigma}^2 = \frac{1}{n-2}\sum_{i=1}^{n} e_i^2 \quad \text{with residuals } e_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

The division by $n-2$ (instead of n) is a correction for the fact that we have estimated two parameters (β_0 and β_1) from the data. This correction ensures that $\hat{\sigma}^2$ is an unbiased estimate of the residual variance σ^2.

Why division by $n-2$ ensures an unbiased estimator

When estimating parameters (β_0 and β_1), the square of the residuals is minimised. This fitting process inherently *uses up* two *degrees of freedom*, as the model forces the residuals to sum to zero and aligns the slope to best fit the data. I.e., one degree of freedom is lost due to the estimation of the intercept, and another due to the estimation of the slope.

The adjustment (division by $n-2$ instead of n) compensates for the loss of variability due to parameter estimation, ensuring the estimator of the residual variance is unbiased. Mathematically, dividing by n - 2 adjusts for this loss and gives an accurate estimate of the population variance when working with sample data.

Degrees of freedom, and why having few of them is a problem

The **residual degrees of freedom** are the number of observations minus the number of parameters estimated by the model. In simple linear regression we estimate two parameters (the intercept and the slope), so the residual degrees of freedom are $n-2$. You can think of them as the number of independent pieces of information left over, after fitting the model, for estimating how much the data vary around the model (the residual variance).

Having few residual degrees of freedom is a problem, for several connected reasons:

- **The residual variance is estimated imprecisely.** It is estimated from the residuals, and with few residual degrees of freedom there is little information to estimate it from. By chance it could be much too small or much too large.
- **Confidence intervals become wider and tests less powerful.** Because the residual variance is uncertain, the t-distribution used for confidence intervals and p-values has heavier tails when there are few degrees of freedom. The value we multiply the standard error by to get a 95% confidence interval (the critical t-value) is then larger, so confidence intervals are wider,

and a real effect is less likely to be detected. The table below shows how quickly the critical value grows as the degrees of freedom shrink. (With very many degrees of freedom it approaches 1.96, the value for the normal distribution.)

- **The model can fit the noise.** If the number of parameters gets close to the number of observations, the model can fit the data closely just by chance. In the extreme, a straight line through two points fits perfectly, leaves zero residual degrees of freedom, and tells us nothing about the residual variance.
- **Assumptions are hard to check.** With few residuals, diagnostic plots show very little, so it is difficult to judge whether the model's assumptions are met.

Residual degrees of freedom	Critical t-value for a 95% confidence interval
1	12.71
2	4.30
3	3.18
5	2.57
10	2.23
20	2.09
30	2.04
100	1.98
1000	1.96

So each parameter a model estimates uses up one degree of freedom. Adding explanatory variables, adding levels to a categorical explanatory variable, or adding interactions all increase the number of parameters, and so reduce the residual degrees of freedom. This is why, when planning a study, it is worth checking that the planned number of observations leaves enough residual degrees of freedom for the model you intend to fit. We will meet degrees of freedom again in every chapter that follows.

Let's do it in R

First we read in the dataset:

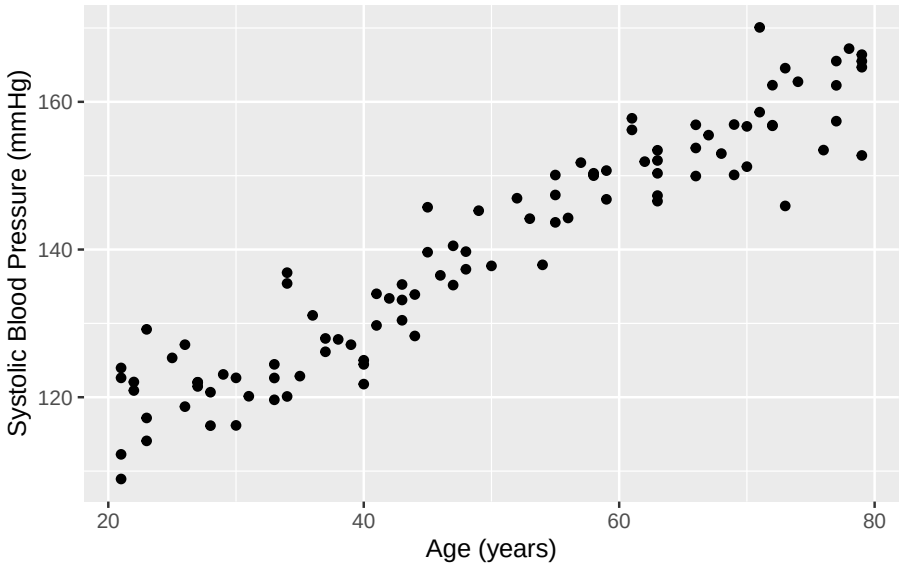
```
bp_age_data <- read_csv("datasets/Simulated_Blood_Pressure_and_Age_Data.csv")
```

```
Rows: 100 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): Age, Systolic_BP

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The we make a graph of the data:

```
ggplot(bp_age_data, aes(x = Age, y = Systolic_BP)) +
  geom_point() +
  labs(x = "Age (years)", y = "Systolic Blood Pressure (mmHg)")
```



Then we make the linear model, using the `lm()` function:

```
bp_age_model <- lm(Systolic_BP ~ Age, data = bp_age_data)
```

Then we can look at the summary of the model. It contains a lot of information, so can be a bit confusing at first.

```
summary(bp_age_model)
```

```
Call:
lm(formula = Systolic_BP ~ Age, data = bp_age_data)

Residuals:
     Min       1Q   Median       3Q      Max
-13.2195  -3.4434  -0.0808   3.1383  12.6025

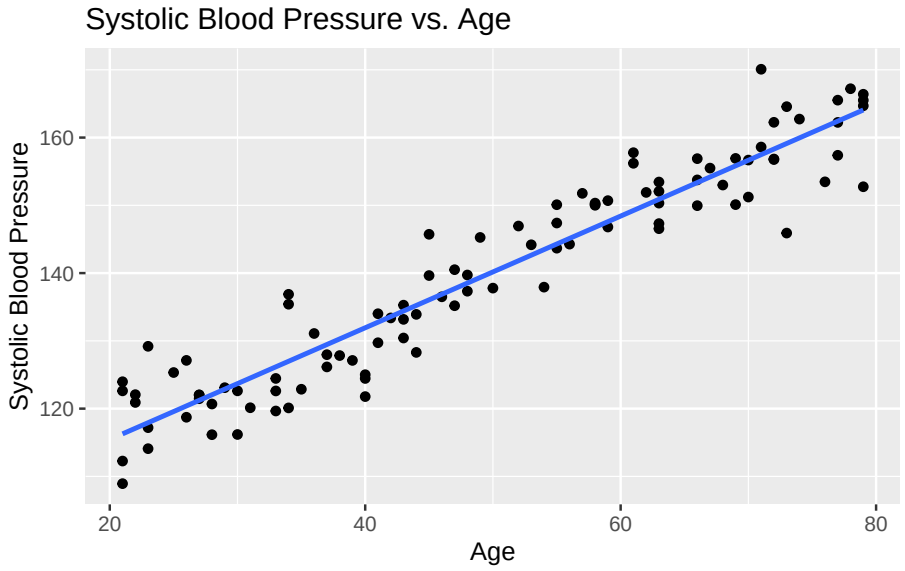
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 98.96874    1.46102   67.74   <2e-16 ***
Age          0.82407    0.02771   29.74   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 4.971 on 98 degrees of freedom
Multiple R-squared:  0.9002,    Adjusted R-squared:  0.8992
F-statistic: 884.4 on 1 and 98 DF,  p-value: < 2.2e-16
```

How do our guesses of the intercept and slope compare to the guesses we made earlier?

Recal that the units of the *Age* coefficient are in mmHg per year. This means that for each additional year of age, the systolic blood pressure increases by 0.82 mmHg.

Dealing with the error



The line is not a perfect fit to the data. There is scatter around the line.

Some of this scatter could be caused by other factors that influence blood pressure, such as diet, exercise, and genetics. Also, the there could be differences due to the measurement instrument (i.e., some measurement error).

These other factors are not included in the model (only age is in the model), so they create variation that can only appear in error term.

In the linear regression model the dependent variable Y is related to the independent variable x as

$$Y = \beta_0 + \beta_1 x + \epsilon$$

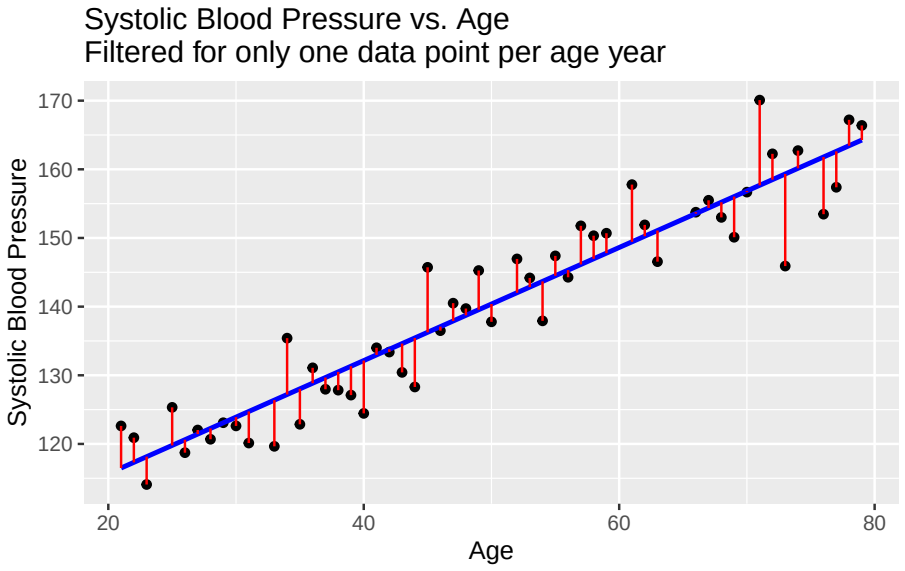
where

- β_0 is the intercept
- β_1 is the slope
- ϵ is the error term.

The error term captures the difference between the observed value of the dependent variable and the value predicted by the model. The error term includes the effects of other factors that influence the dependent variable, as well as measurement error.

$$Y = \underbrace{\text{expected value}}_{E(Y)=\beta_0+\beta_1 x} + \underbrace{\text{random error}}_{\epsilon} .$$

Graphically the error term is the vertical distance between the observed value of the dependent variable and the value predicted by the model.



The error term is also known as the residual. It is the variation that *resides* (is left over / is left unexplained) after accounting for the relationship between the dependent and independent variables.

Example in R

Let's look at observed values, expected (predicted) values, and residuals (error) in R.

The observed values of the response (dependent) variable are already in the dataset:

```
head(bp_age_data$Systolic_BP)
```

```
[1] 150.3277 170.0801 139.7174 135.4089 151.9041 122.0296
```

To get the expected values, we need to find the intercept and slope of the linear model. We can do this using the `lm()` function in R.

```
bp_age_model <- lm(Systolic_BP ~ Age, data = bp_age_data)
```

And we can get the intercept and slope using the `coef()` function:

```
coef(bp_age_model)
```

```
(Intercept)         Age
 98.9687381   0.8240678
```

We can then use the mutate function from the dplyr package to add the expected values to the dataset:

```
bp_age_data <- bp_age_data |>
  mutate(Expected_BP = coef(bp_age_model)[1] + coef(bp_age_model)[2] * Age)
```

And we can get the residuals by subtracting the expected values from the observed values:

```
bp_age_data <- bp_age_data |>
  mutate(Residuals = Systolic_BP - Expected_BP)
```

> **Tip**
>
> We can also get the expected values and residuals directly from the `lm` object using the `fitted()` (or `predicted()`) and `residuals()` functions:
>
> ```
> bp_age_data <- bp_age_data |>
>   mutate(Expected_BP = fitted(bp_age_model),
>          Residuals = residuals(bp_age_model))
> ```

Now we have a model that gives the expected values (on the regression line) and that gives us a residual. Because the expected value plus the residual equals the observed value, if we use each of the residuals as the error for each respective data point, we end up with a perfect fit to the data. All we are doing is describing the observed data in a different way. This is known as over-fitting. In fact, we have gained very little by fitting the model. We have simply memorized / copied the data!!!

In order to avoid this, we need to assume something about the residuals – we need to *model* the residuals. The most common model for the residuals is a normal distribution with mean 0 and constant variance.

$$\epsilon \sim N(0, \sigma^2)$$

This is known as the normality assumption. The normality assumption is important because it allows us to make inferences about the *population parameters* based on the *sample data*.

The linear regression model then becomes:

$$Y = \beta_0 + \beta_1 x + N(0, \sigma^2)$$

where σ^2 is the variance of the error term. The variance of the error term is the amount of variation in the dependent variable that is not explained by the independent variable. The variance of the error term is also known as the residual variance.

An alternate and equivalent formulation is that Y is a random variable that follows a normal distribution with mean $\beta_0 + \beta_1 x$ and variance σ^2.

$$Y \sim N(\beta_0 + \beta_1 x, \sigma^2)$$

So, the answer to the question "how do we deal with the error term" is that we model the error term as normally distributed with mean 0 and constant variance. Put another way, the error term is assumed to be normally distributed with mean 0 and constant variance.

Back to blood pressure and age

The mathematical model in this case is:

$$SystolicBP = \beta_0 + \beta_1 \times Age + \epsilon$$

where: *SystolicBP* is the dependent (response) variable, β_0 is the intercept, β_1 is the coefficient of Age, *Age* is the independent (explanatory) variable, ϵ is the error term.

Let's ensure we understand this, by thinking about the units of the variables in this model. This can be very useful because it can help us to understand the model better and to check that the model makes sense.

Is the model good enough to use?

- All models are wrong, but is ours good enough to be useful?
- Are the assumption of the model justified?

- It would be very unwise to use the model before we know if it is good enough to use.
- *Don't jump out of an aeroplane until you know your parachute is good enough!*

What assumptions do we make?

We already heard about one. We assume that the residuals follow a $N(0, \sigma^2)$ distribution (that is, a Gaussian / Normal distrution with mean of zero and variance of σ^2). We make this assumption because it is often well enough met, and it gives great mathematical tractability.

This assumption implies that:

(a) The ϵ_i are normally distributed.
(b) ϵ_i has constant variance: $Var(\epsilon_i) = \sigma^2$.
(c) The ϵ_i are independent of each other.

> **Note**
>
> These assumptions are about the residuals, not the observed data!!!

Furthermore:

(d) we assumed a linear relationship.
(e) implies there are no outliers (implied by (a) above)

Lets go through each five assumptions.

(a) Normally distributed residuals

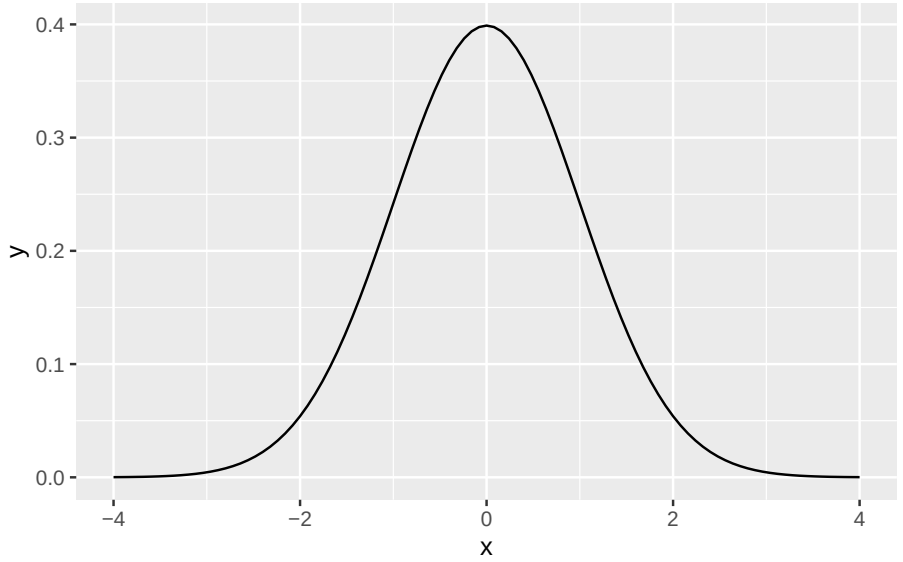
Recall that we make the assumption that the residuals are normally distributed with mean 0 and constant variance:

$$\epsilon \sim N(0, \sigma^2)$$

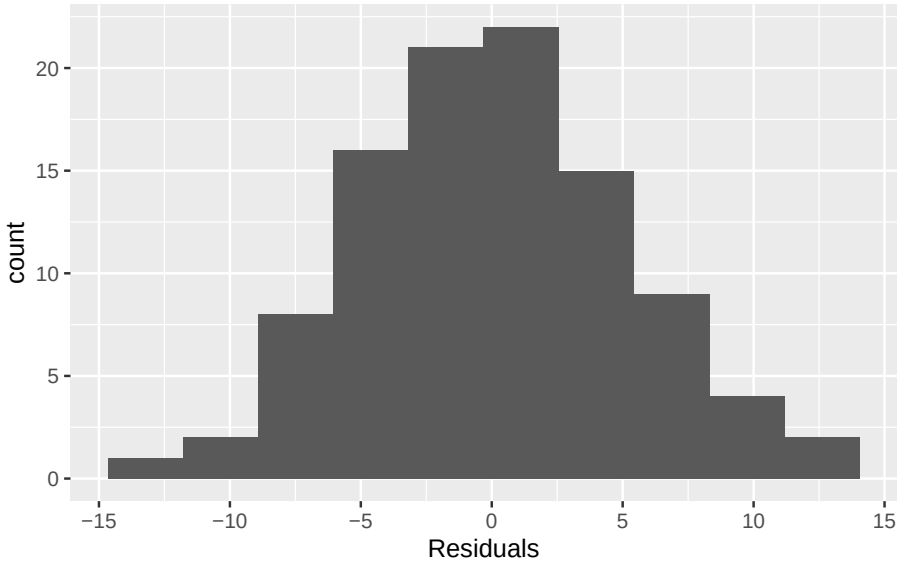
Here we are concerned with the first part of this assumption, that the residuals are normally distributed.

What does this mean? How can we check it?

A normal distribution is symmetric and bell-shaped…



Lets look at the frequency distribution of the residuals of the linear regression of blood pressure and age:

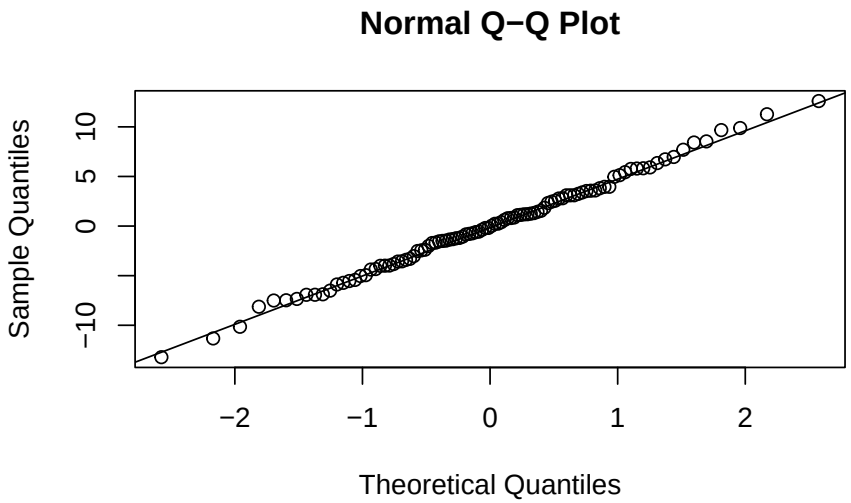


The normal distribution assumption (a) seems ok as well.

(a) Normally distributed residuals: The QQ-plot

Usually, not the histogram of the residuals is plotted, but the so-called **quantile-quantile** (QQ) plot. The quantiles of the observed distribution are plotted against the quantiles of the respective theoretical (normal) distribution:

```
qqnorm(residuals(bp_age_model))
qqline(residuals(bp_age_model))
```

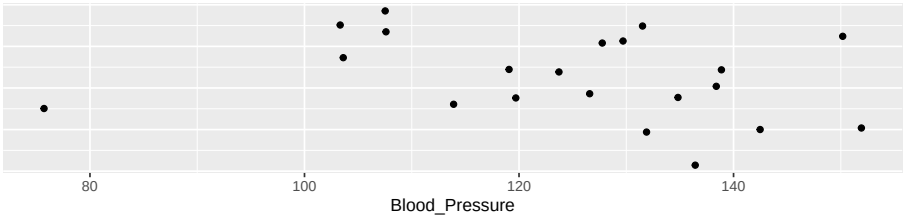


If the points lie approximately on a straight line, the data is fairly normally distributed.

This is often "tested" by eye, and needs some experience.

But what on earth is a quantile???

Imagine we make 21 measures of something, say 21 blood pressures:

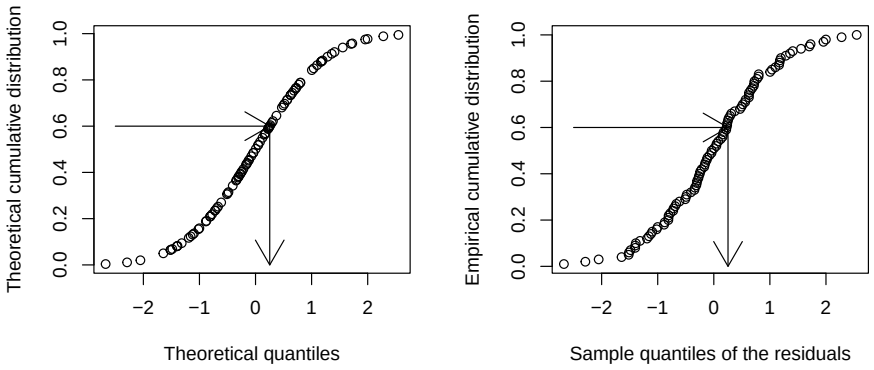


The median of these is 127.8. The median is the 50% or 0.5 quantile, because half the data points are above it, and half below.

```
quantile(dd$Blood_Pressure, 0.5)
```

```
  50%
127.8
```

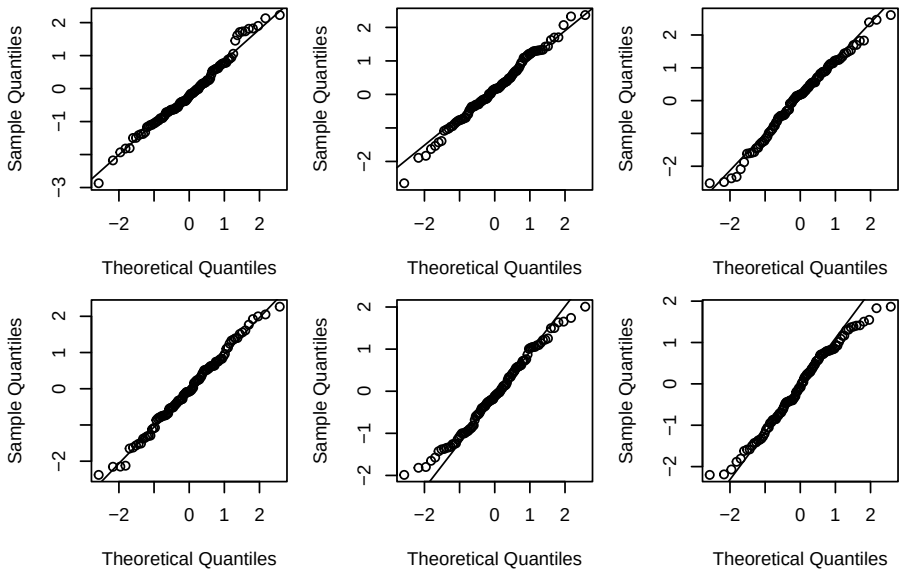
The *theoretical quantiles* come from the normal distribution. The *sample quantiles* come from the distribution of our residuals.



How do I know if a QQ-plot looks "good"?

There is **no quantitative rule** to answer this question. Instead experience is needed. You can gain this experience from simulations. To this end, we can generate the same number of data points of a normally distributed variable and compare this simulated qqplot to our observed one.

Example: Generate 100 points $\epsilon_i \sim N(0, 1)$ each time:



Each of the graphs above has data points that are randomly generated from a normal distribution. In all cases the data points are close to the line. This is what we would expect if the data were normally distributed. The amount of deviation from the line is what we would expect from random variation, and so seeing this amount of variation in a QQ-plot of your model should not be cause for concern.

(b) Constant error variance (homoscedasticity)

Recall that we assume the errors are normally distributed with constant variance σ^2:

$$\epsilon_i \sim N(0, \sigma^2)$$

Here we're concerned with the second part of this assumption, that the variance is constant.That is, variance of the residuals is a constant: $\text{Var}(\epsilon_i) = \sigma^2$. And not, for example $\text{Var}(\epsilon_i) = \sigma^2 \cdot x_i$.

Put another way, we're interested if the size of the residuals tends to show a pattern with the fitted values. By *size* of the residuals we mean the *absolute value* of the residuals. In fact, we often look at the square root of the absolute value of the standardized residuals:

$$R_i = \frac{\epsilon_i}{\hat{\sigma}}$$

Where $\hat{\sigma}$ is the estimated standard deviation of the residuals:

$$\hat{\sigma} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}\epsilon_i^2}$$

So that the full equation of the square root of the standardised residuals is:

$$\sqrt{|R_i|} = \sqrt{\left|\frac{\epsilon_i}{\hat{\sigma}}\right|}$$

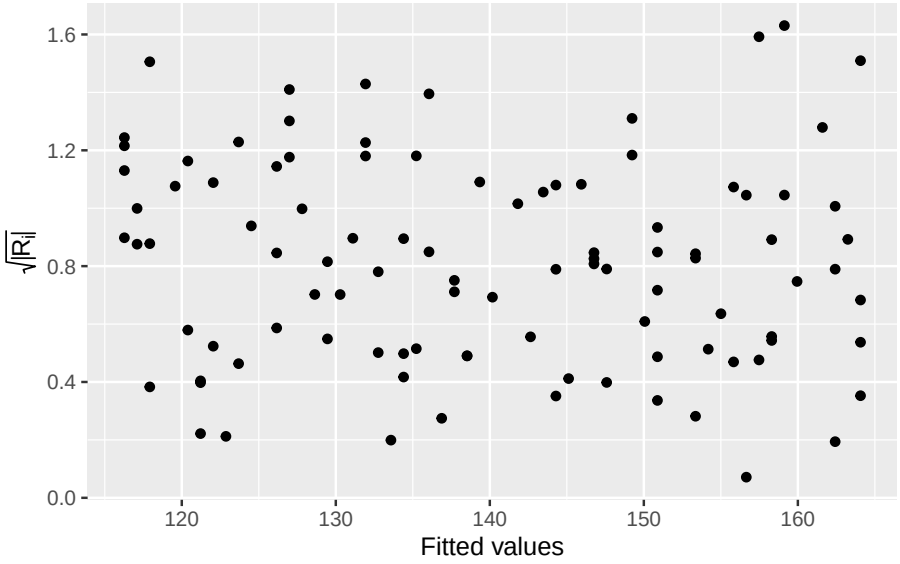
To look to see if the variance of the residuals is constant, we need to see if there is any relationship between the size of the residuals and the fitted values. A commonly used visualistion for this is a plot of the size of the residuals against the fitted values.

Lets first calculated the $\sqrt{|R_i|}$ values for our blood pressure model:

```
bp_age_data <- bp_age_data |>
  mutate(fitted = predict(bp_age_model),
         residuals = residuals(bp_age_model),
         sigma_hat = sqrt(sum(residuals^2)/(n()-2)),
         R_i = residuals / sigma_hat,
         sqrt_abs_R_i = sqrt(abs(R_i)))
```

And now visualise the relationship between the fitted values and the size of the residuals:

```
ggplot(bp_age_data, aes(x = fitted, y = sqrt_abs_R_i)) +
  geom_point() +
  labs(x = "Fitted values", y = expression(sqrt(abs(R[i]))))
```



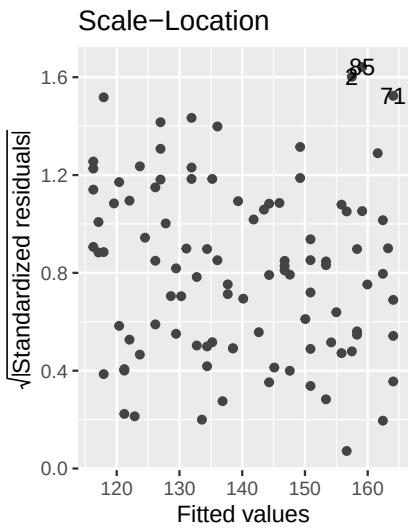
This graph is known as the scale-location plot. It is particularly suited to check the assumption of equal variances (**homoscedasticity / Homoskedastizität**). There should be **no trend** or pattern.

Tip

We can also use the `autoplot()` function from the `ggfortify` package to create this plot. It is the third plot in the set of model checking plots, so we use `which = 3`. The argument `smooth.colour = NA` removes the smoothed line.

```
library(ggfortify)
autoplot(bp_age_model, which = 3, smooth.colour = NA)
```

```
Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.
i Please use `broom::augment(<lm>)` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
Warning: Removed 100 rows containing missing values or values outside the scale rang
(`geom_line()`).
```

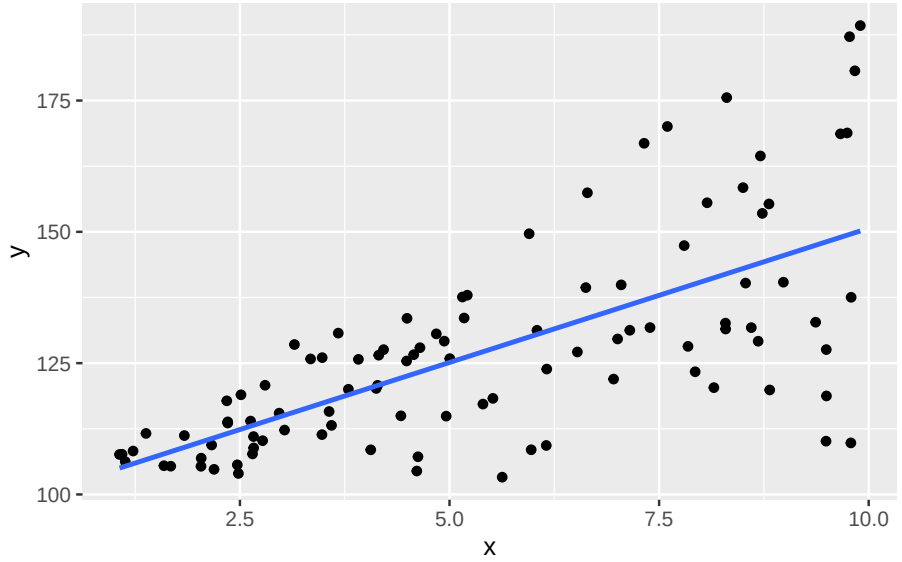


How it looks with the variance increasing with the fitted values

Here's a graphical example of how it would look if the variance of the residuals increases with the fitted values.

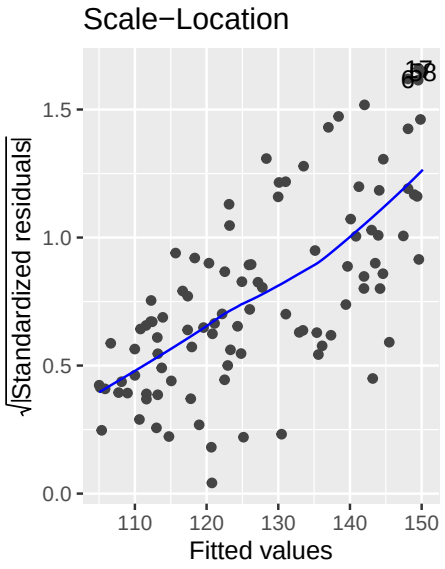
First here is a graph of the relationship:

```
set.seed(2)
x <- runif(100, 1, 10)
y <- 100 + 5*x + 2*x*rnorm(100,0,1)
ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method="lm", se=FALSE)
```



And here the scale-location plot for a linear model of that data:

```
m_sim <- lm(y~x)
autoplot(m_sim, which = 3)
```



(c) Independence (residuals are independent of each other)

We assume that the residuals (ϵ_i) are independent of each other. This means that the value of one residual is not somehow related to the value of another.

The dataset about blood pressure we looked at contained 100 observations, each one made from a different person. In such a study design, we could be safe in the assumption that the people are independent, and therefore the assumption that the residuals are independent.

Imagine, however, if we had 100 observations of blood pressure collected from 50 people, because we measured the blood pressure of each person twice. In this case, the residuals would not be independent, because two measures of the blood pressure of the same person are likely to be similar. A person is likely to have a high blood pressure in both measurements, or a low blood pressure in both measurements. This would mean they have a high residual in both measurements, or a low residual in both measurements.

In this case, we would need to account for the fact that the residuals are not independent. We would need to use a more complex model, such as a mixed effects model, to account for the fact that the residuals are not independent. We will talk about this again in the last week of this course.

In general, you should always think about the study design when you are analysing data. You should always think about whether the residuals are likely to be independent of each other. If they are not, you should think about how you can account for this in your analysis.

A good way to assess if there could be dependencies in the residuals is to be critical about what is the unit of observation in the data. In the blood pressure example, the unit of observation is the person. Count the number of persons in the study. If there are fewer persons than observations, then at least some people must have been measured at least twice. Repeating measures on the same person is a common way to get dependent residuals.

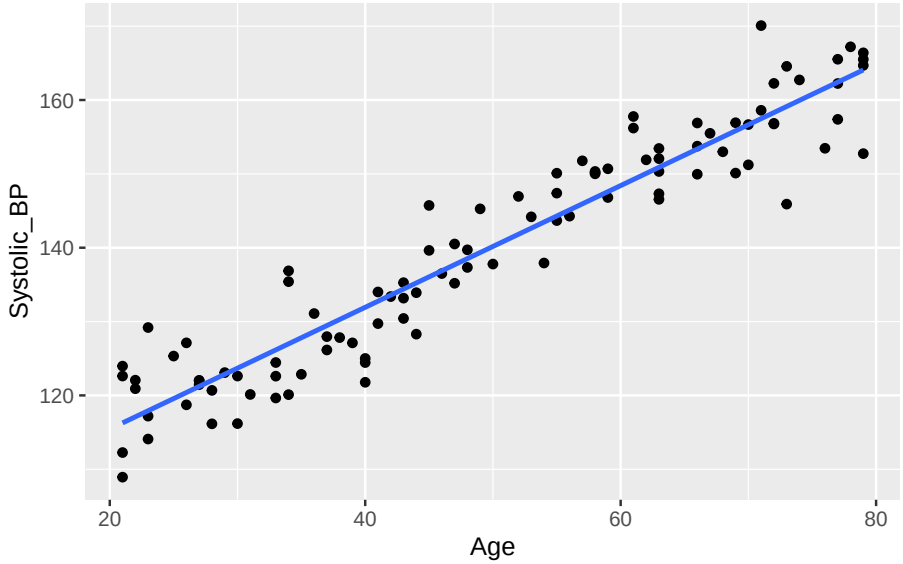
So, to check the assumption of independence, you should:

- Think carefully about the study design.
- Think carefully about the unit of observation in the data.
- Compare the number of observations to the number of units of observation.

(d) Linearity assumption

The linearity assumption states that the relationship between the independent variable and the dependent variable is linear. This means that the dependent variable changes by a constant amount for a one-unit change in the independent variable. And that this slope is does not change with the value of the independent variable.

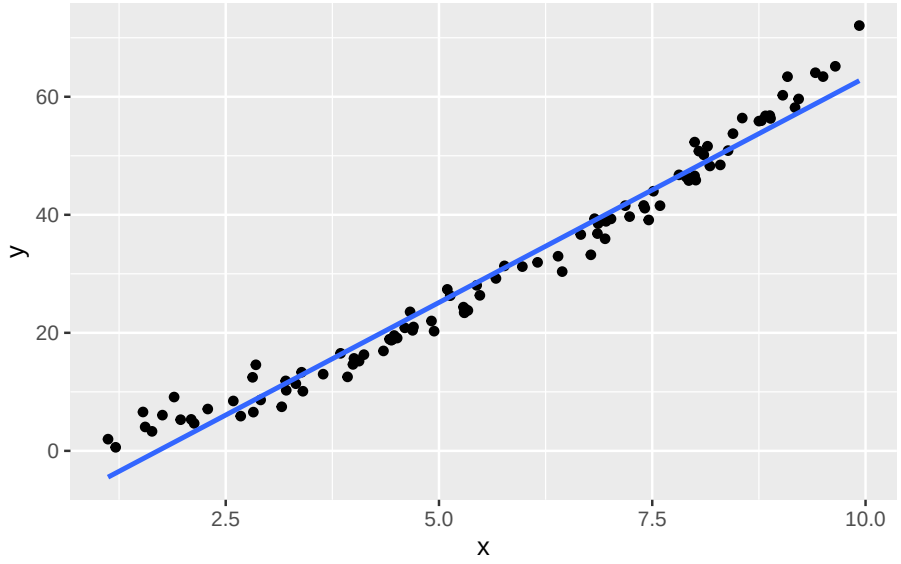
The blood pressure data seems to be linear:




```
ggplot(bp_age_data, aes(x = Age, y=Systolic_BP)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method="lm", se=FALSE)
```

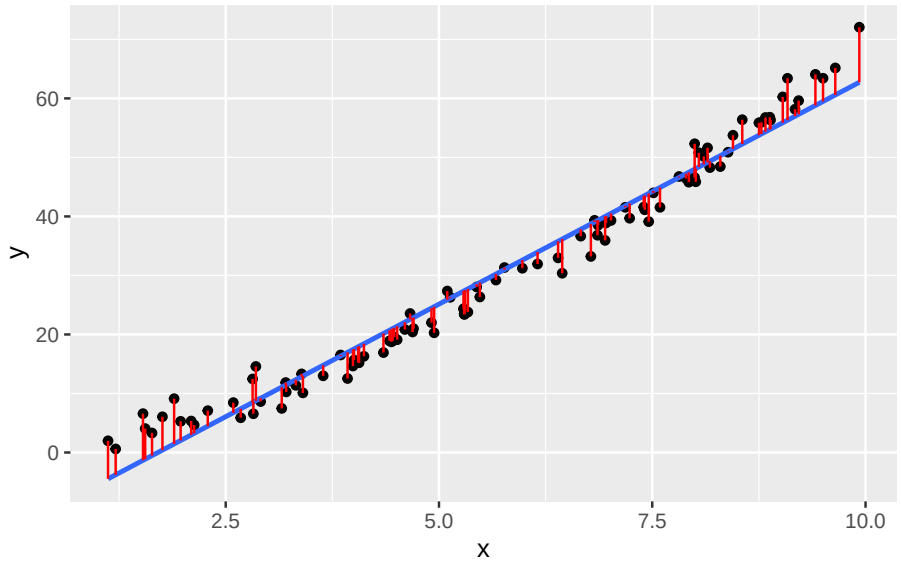
In contrast, look at this linear regression through data that appears non-linear:

```
set.seed(1)
x <- runif(100, 1, 10)
y <- 2*x + rnorm(100,0,2) + 0.5*x^2
m_sim <- lm(y~x)
ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method="lm", se=FALSE)
```



And with the residuals shown as red lines:

```
ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method="lm", se=FALSE) +
  geom_segment(aes(xend=x, yend=m_sim$fitted), color="red")
```

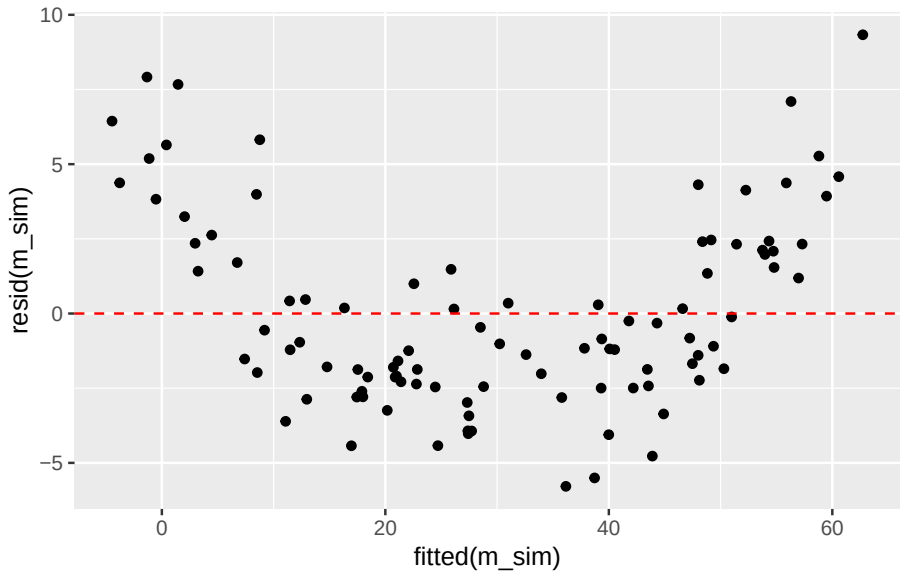


At low values of y, the residuals are positive, at intermediate values of y the

residuals are negative, and at high values of y the residuals are positive. This pattern in the residuals is a sign that the relationship between x and y is not linear.

We can plot the value of the residuals against the y value directly, instead of looking at the pattern in the graph above. This is called a **Tukey-Anscombe plot**. It is a graph of the residuals versus the fitted y values:

```
ggplot(mapping = aes(x = fitted(m_sim), y = resid(m_sim))) +
  geom_point() +
  geom_hline(yintercept = 0, linetype="dashed", color="red")
```

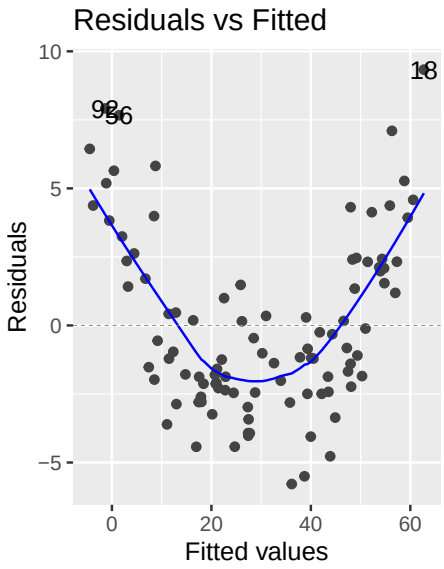


We can very clearly see pattern in the residuals in this Tukey-Anscombe plot. The residuals are positive, then negative, then positive, as the fitted y value gets larger.

We can also make this Tukey-Anscombe plot using the `autoplot()` function (it is plot number 1 of the model checking plots):

```
autoplot(m_sim, which = 1)
```

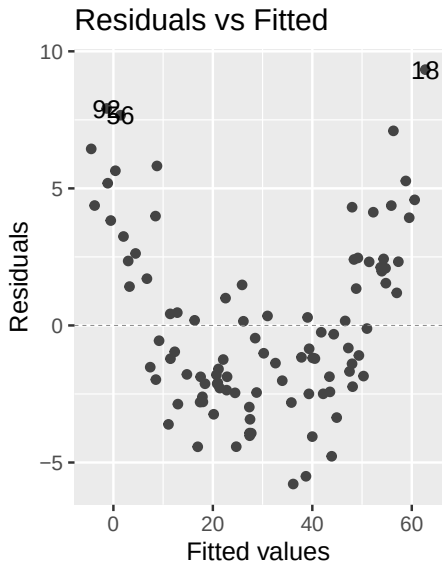
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```



The blue line in the Tukey-Anscombe plot is a loess smooth. It is automatically added to the plot. It is a way of estimating the pattern in the residuals. If the blue line is not flat, then there is a pattern in the residuals. However, the loess smooth is not always reliable. It is a good idea to look at the residuals directly, without this smooth.

```
autoplot(m_sim, which = 1, smooth.colour = NA)
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```

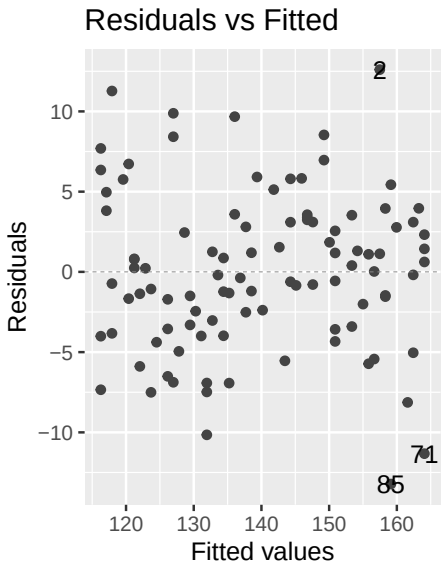


The data here is simulated to show a very clear pattern in the residuals. In real data, the pattern might not be so clear. But if you suspect you see a pattern in the residuals, it could be a sign that the relationship between the independent and dependent variable is not linear.

Here is the Tukey-Anscombe plot for the blood pressure data:

```
autoplot(bp_age_model, which = 1, smooth.colour = NA)
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



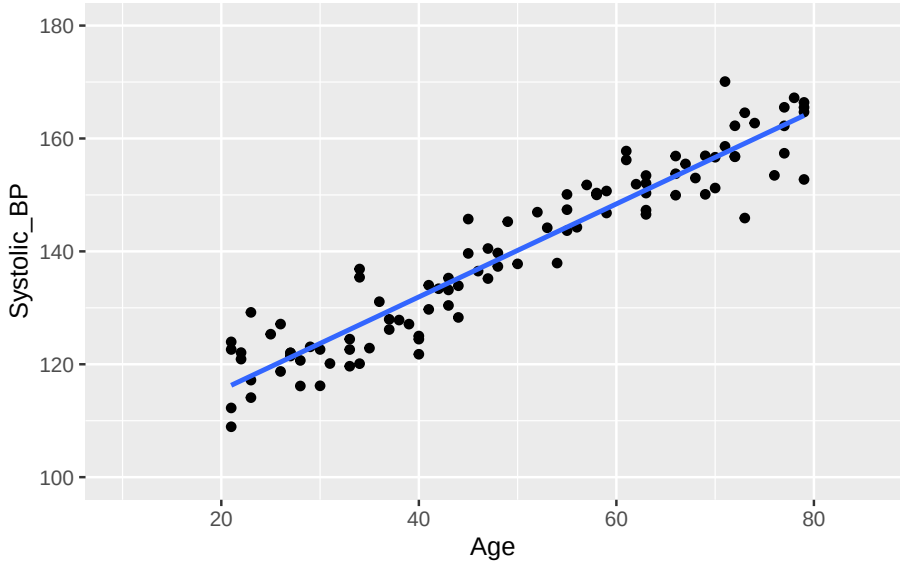
There is very little evidence of any pattern in the residuals. This data is simulated with a truly linear relationship, so we would not expect to see any pattern in the residuals.

(e) No outliers

An outlier is a data point that is very different from the other data points. Outliers can have a big effect on the results of a regression analysis. They can pull the line of best fit towards them, and make the line of best fit a poor representation of the data.

Lets again look at the blood pressure versus age data:

```
ggplot(bp_age_data, aes(x = Age, y=Systolic_BP)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method="lm", se=FALSE) +
  xlim(10, 85) +
  ylim(100, 180)
```



There are no obvious outliers in this data. The data points are all close to the line of best fit. This is a good sign that the line of best fit is a good representation of the data.

Data points that are far from the mean of the independent variable have a large effect on the value of the slope. These data points have a large leverage. They are data points that are far from the other data points in the x direction.

We can think of this with the analogy of a seesaw. The slope of the line of best fit is like the pivot point of a seesaw. Data points that are far from the pivot point have a large effect on the slope. Data points that are close to the pivot point have a small effect on the slope.

A measure of distance from the pivot point is called the *leverage* of a data point. In simple regression, the leverage of individual i is defined as

$h_i = (1/n) + (x_i - \overline{x})^2/SSX.$

where $SSX = \sum_{i=1}^{n}(x_i - \overline{x})^2$. (**S**um of **S**quares of X)

So, the leverage of a data point is inversely related to n (the number of data points). The leverage of a data point is also inversely related to the sum of the squares of the x values. The leverage of a data point is directly related to the square of the distance of the x value from the mean of the x values.

More intuitively perhaps, the leverage of a data point will be greater when the are fewer other data points. It will also be greater when the distance from the mean value of x is greater.

Going back to the analogy of a seesaw, with data points as children on the

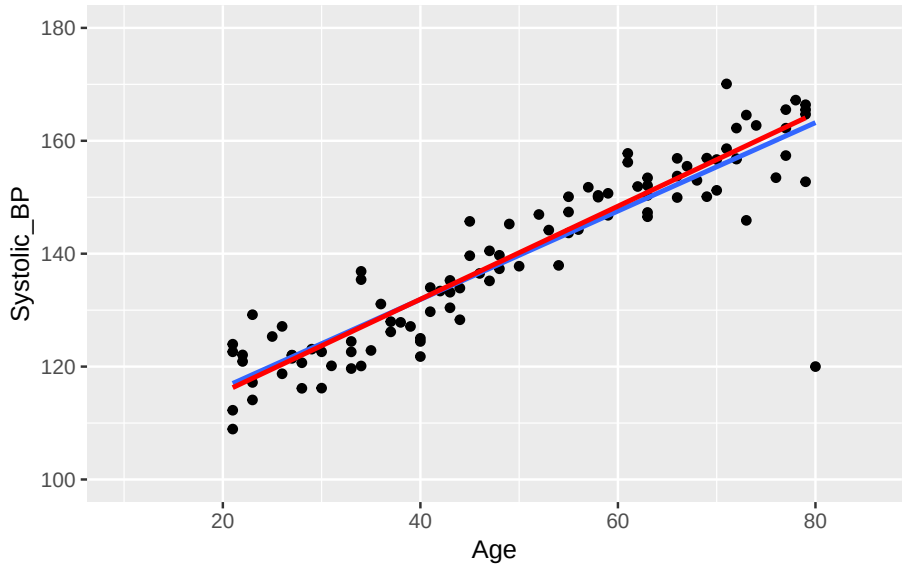
seesaw, the leverage of a data point is like the distance from the pivot a child sits. But we also have children of different weights. A lighter child will have less effect on the tilt of the seesaw. A heavier one will have a greater effect on the tilt. A heavier child sitting far from the pivot will have a very large effect.

The size of the residuals are like the weight of the child. Data points with large residuals have a large effect on the slope of the line of best fit. Data points with small residuals have a small effect on the slope of the line of best fit.

So the overall effect of a data point on the slope of the line of best fit is a combination of the leverage and the residual. This quantity is called the *influence* of a data point.

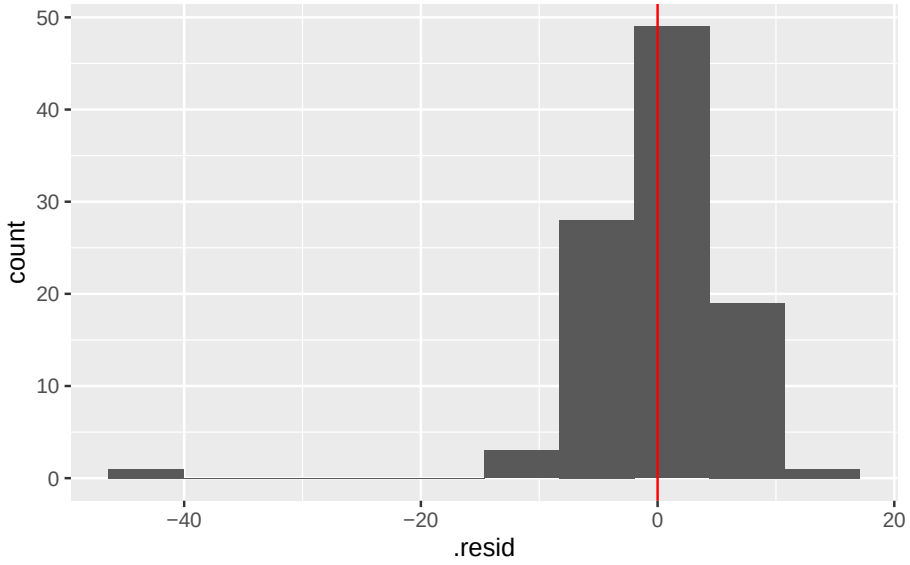
Let's add a rather extreme data point (age 80, blood pressure 120) to the blood pressure versus age data, and fit the model to the data with this extra point:

```
bp_age_data_outlier <- bp_age_data |>
  select(Age, Systolic_BP) |>
  bind_rows(tibble(Age = 80, Systolic_BP = 120))
m_bp_age_outlier <- lm(Systolic_BP ~ Age, data = bp_age_data_outlier)
```



This is a bit ridiculous, but it is a good example of an outlier. The data point is far from the other data points. It has a large residual. And it is a long way from the pivot (the middle of the x data) so has large leverage.

We can make a histogram of the residuals and see that the outlier has a large residual:



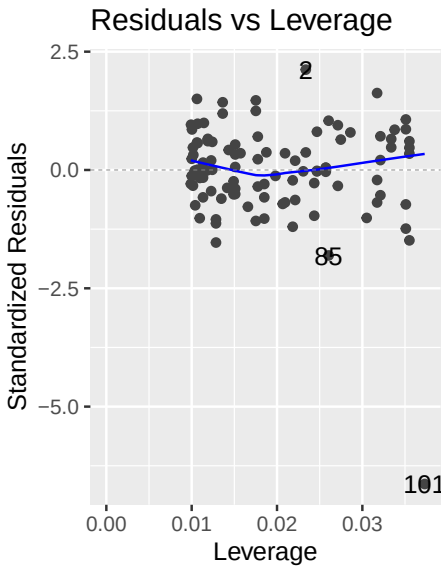
And we can see that the leverage is large.

There is a graph that we can look at to see the influence of a data point. This is called a *Cook's distance* plot. The Cook's distance of a data point is a measure of how much the slope of the line of best fit changes when that data point is removed. The Cook's distance of a data point is defined as

$D_i = \sum_{j=1}^{n} (\hat{y}_j - \hat{y}_{j(i)})^2 / (p \times MSE)$.

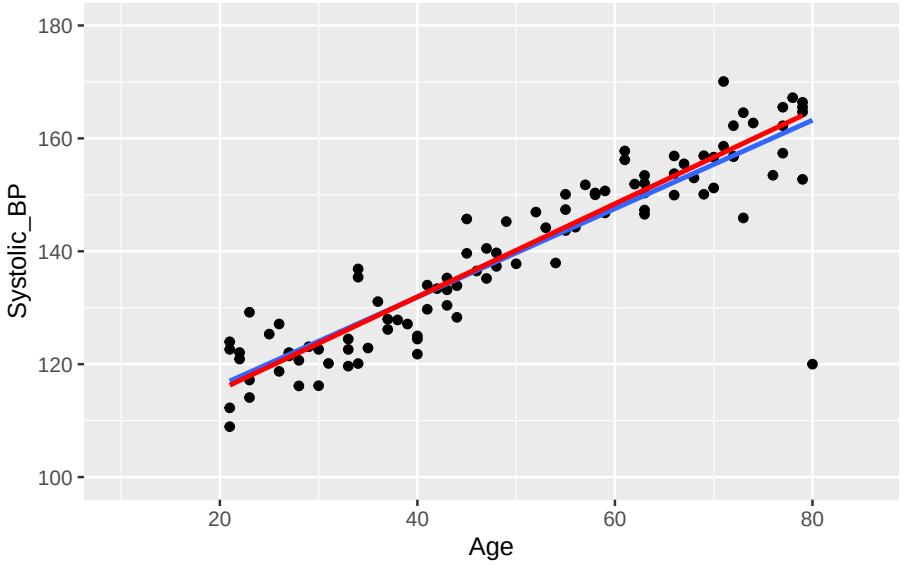
where $\hat{y}_j$ is the predicted value of the dependent variable for data point j, $\hat{y}_{j(i)}$ is the predicted value of the dependent variable for data point j when data point i is removed, p is the number of parameters in the model (2 in this case), MSE is the mean squared error of the model.

```
autoplot(m_bp_age_outlier, which = 5)
```



Observation 101 (the added point) is far to the right (large leverage) and far from zero (large standardised residual). Its Cook's distance is 0.85, much larger than that of any of the original data points (the largest of which is 0.1).

But does it have a large influence on the value of the slope? In the next graph we show the line of best fit with the outlier (blue line) and without the outlier (red line).



We can compare the estimated coefficients with and without the outlier:

```
coef(bp_age_model)
```

```
(Intercept)         Age
 98.9687381   0.8240678
```

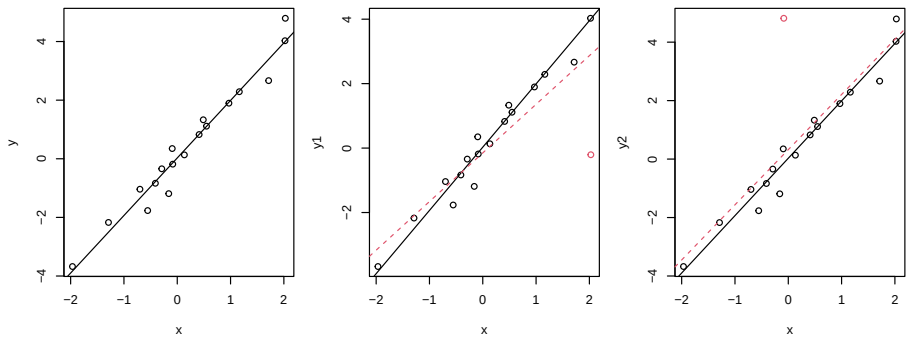
```
coef(m_bp_age_outlier)
```

```
(Intercept)         Age
100.5616423   0.7832227
```

The outlier has a moderate influence on the slope: it changes from 0.824 to 0.783 mmHg per year. The outlier has a large leverage. It is far from the pivot. But it does not have a very large effect (influence) on the slope. This is in large part because there are a lot data points (100) that are quite tightly arranged around the regression line.

Graphical illustration of the leverage effect

Data points with x_i values far from the mean have a stronger leverage effect than when $x_i \approx \overline{x}$:



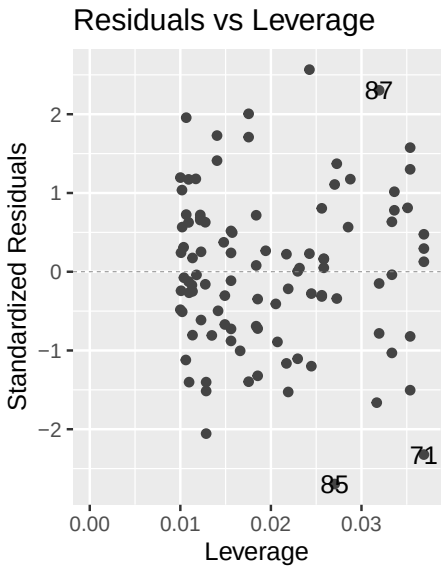
The outlier (red circle) in the middle plot "pulls" the regression line in its direction and has large influence on the slope. THe outlier (red circle) in the right plot has less influence on the slope because it is closer to the mean of x.

Leverage plot (Hebelarm-Diagramm)

In the leverage plot, (standardized) residuals $\tilde{R}_i$ are plotted against the leverage H_{ii} :

```
autoplot(bp_age_model, which = 5, smooth.colour = NA)
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



Critical ranges are the top and bottom right corners!!

Here, observations 71, 85, and 87 are labelled as potential outliers.

Some texts will give a rule of thumb that points with Cook's distances greater than 1 should be considered influential, while others claim a reasonable rule of thumb is $4/(n-p-1)$ where n is the sample size, and p is the number of *beta* parameters.

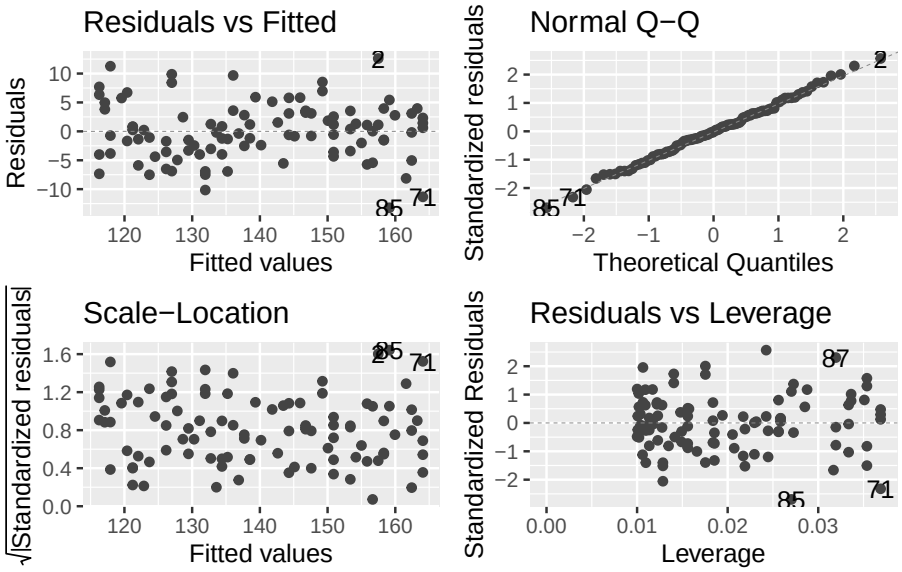
The `autoplot()` function from the `ggfortify` package

So far we have used the `autoplot()` function from the `ggfortify` package to make one model checking plot at a time, choosing the plot with the `which` argument. If we leave out `which`, `autoplot()` makes the four standard model checking plots (residuals vs fitted, QQ-plot, scale-location, and residuals vs leverage) arranged in a 2x2 grid. This is what you will usually do. Of course, you do need to ensure that the `ggfortify` package is installed and loaded. (You may also see the built-in `plot()` function used for this in other texts, for example `plot(bp_age_model, which = 1)`; it makes the same plots with base R graphics.)

```
library(ggfortify)
autoplot(bp_age_model, smooth.colour = NA)
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```

```
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



(At the time of writing, this use of `autoplot()` causes a warning message to be printed. This is a known issue with the `ggfortify` package. You can safely ignore this warning message for now.)

What can go "wrong" during the modeling process?

Answer: a lot of things!

- Non-linearity. We assumed a linear relationship between the response and the explanatory variables. But this is not always the case in practice. We might find that the relationship is curved and not well represtented by a straight line.
- Non-normal distribution of residuals. The QQ-plot data might deviate from the straight line so much that we get worried!
- Heteroscadisticity (non-constant variance). We assumed homoscadisticity, but the residuals might show a pattern.
- Data point with high influence. We might have a data point that has a large influence on the slope of the line of best fit.

What to do when things "go wrong"?

1. Now: Transform the response and/or explanatory variables.
2. Now: Take care of outliers.

3. Later in the course: Improve the model, e.g., by adding additional terms or interactions.
4. Later in the course: Use another model family (generalized or nonlinear regression model).

Dealing with non-linearity

Here's another example of y and x that are not linearly related:

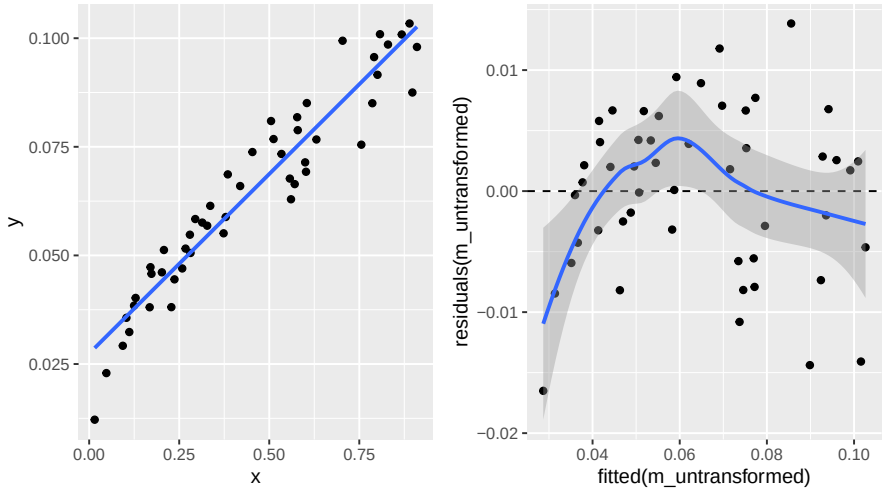
```
set.seed(3)
nonlinear_data <- tibble(x = runif(50)) |>
  mutate(log_y = log(0.1) + 0.5* log(x) + rnorm(50,0,0.1),
         y = exp(log_y),
         log_y = log(y),
         log_x = log(x),
         sqrt_y = sqrt(y))

p1 <- ggplot(nonlinear_data, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "lm", se = FALSE)

m_untransformed <- lm(y ~ x, nonlinear_data)

p2 <- ggplot(mapping = aes(x = fitted(m_untransformed), y = residuals(m_untransformed))) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(formula = y ~ x, method = "loess")

p1 + p2
```

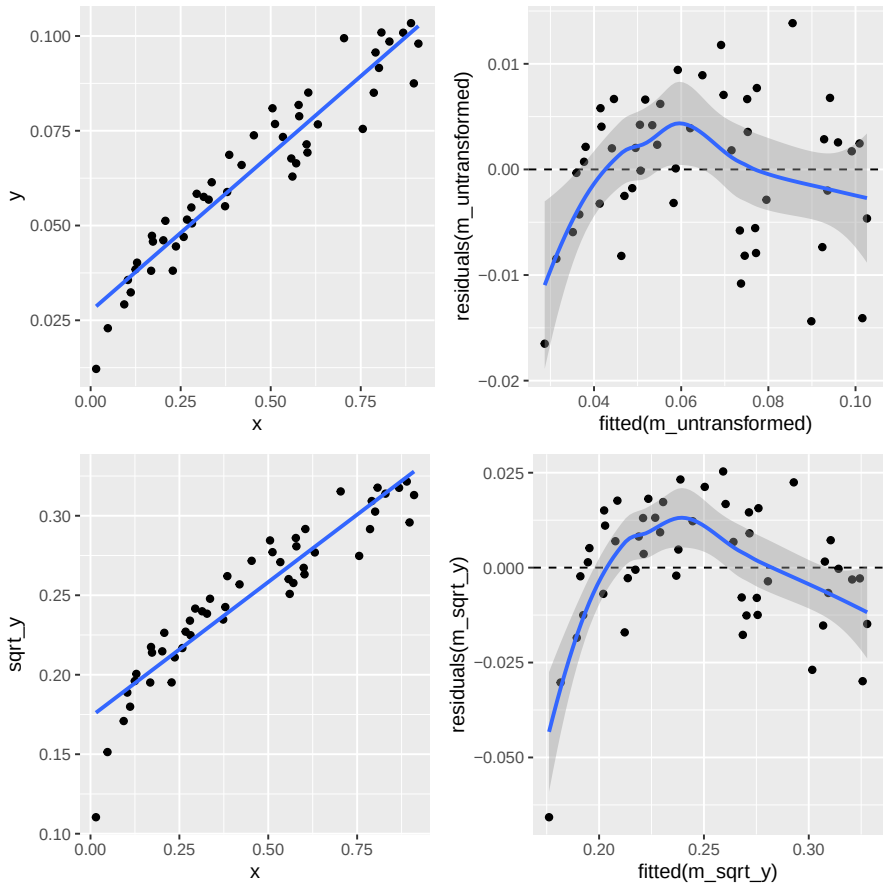


One way to deal with this is to transform the response variable Y. Here we try two different transformations: $\log(Y)$ (the natural logarithm, which is what the `log()` function in R calculates) and $\sqrt{Y}$.

Square root transform of the response variable Y:

```
p3 <- ggplot(nonlinear_data, aes(x = x, y = sqrt_y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "lm", se = FALSE)
m_sqrt_y <- lm(sqrt_y ~ x, nonlinear_data)
p4 <- ggplot(mapping = aes(x = fitted(m_sqrt_y), y = residuals(m_sqrt_y))) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(formula = y ~ x, method = "loess")

(p1 + p2) / (p3 + p4)
```

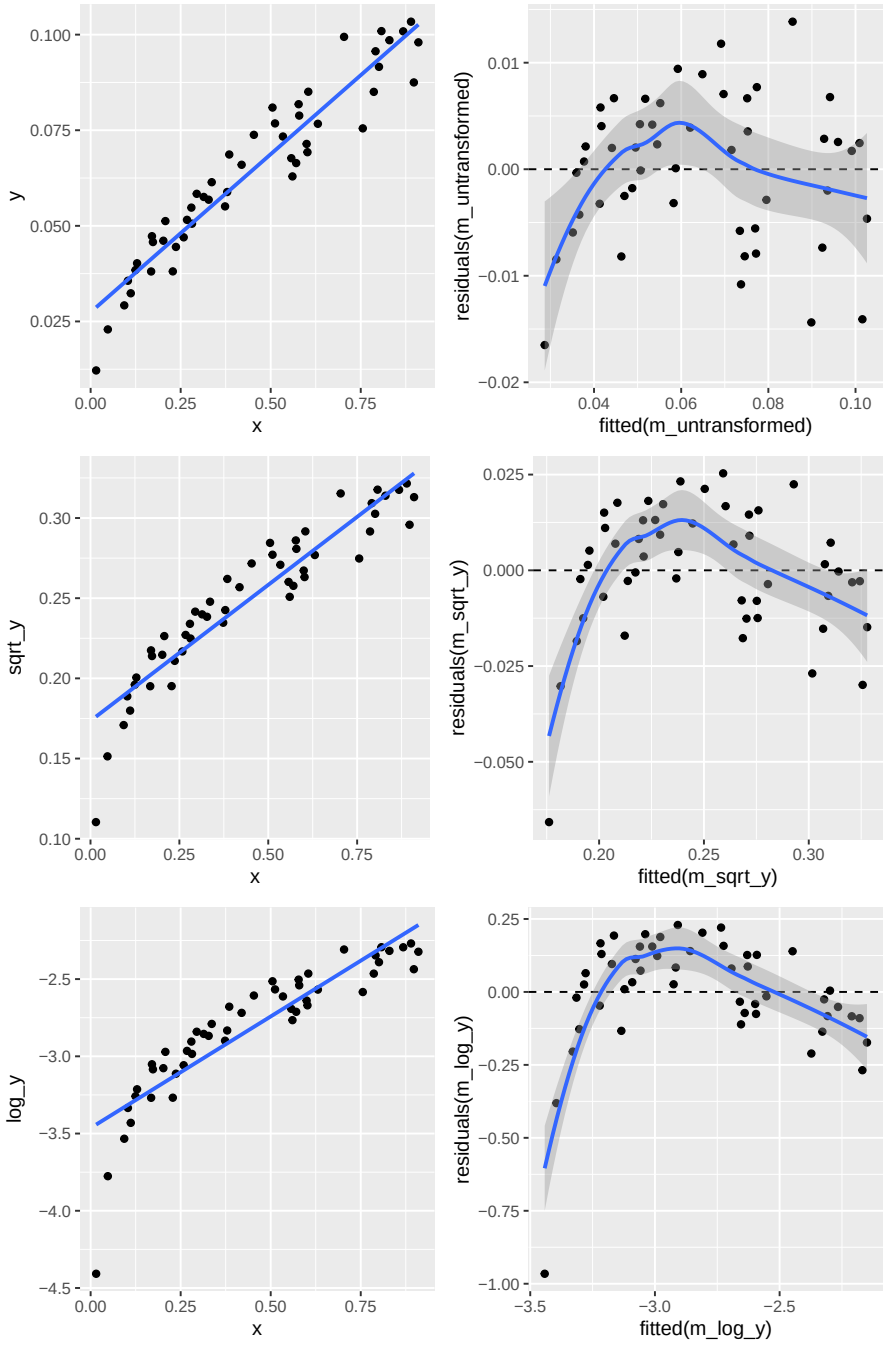


Not great.

Log transformation of the response variable Y:

```
p5 <- ggplot(nonlinear_data, aes(x = x, y = log_y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "lm", se = FALSE)
m_log_y <- lm(log_y ~ x, nonlinear_data)
p6 <- ggplot(mapping = aes(x = fitted(m_log_y), y = residuals(m_log_y))) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(formula = y ~ x, method = "loess")

(p1 + p2) / (p3 + p4) / (p5 + p6)
```

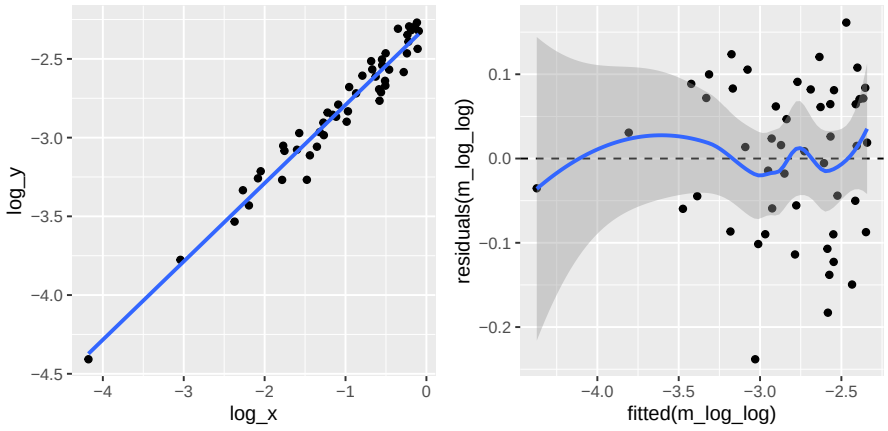


Nope. Still some evidence of non-linearity.

What about transforming the explanatory variable X as well?

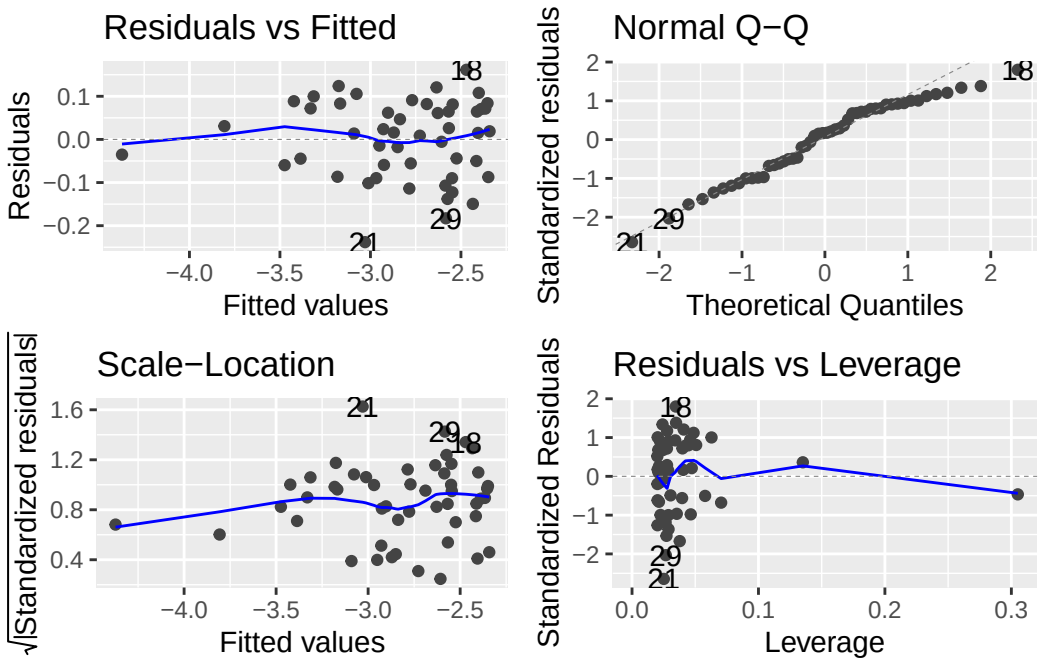
```
p7 <- ggplot(nonlinear_data, aes(x = log_x, y = log_y)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "lm", se = FALSE)
m_log_log <- lm(log_y ~ log_x, nonlinear_data)
p8 <- ggplot(mapping = aes(x = fitted(m_log_log), y = residuals(m_log_log))) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(formula = y ~ x, method = "loess")

p7 + p8
```



Let's look at the four model checking plots for the log-log-transformed data:

```
autoplot(m_log_log)
```



All looks pretty good except for the scale-location plot, which shows a bit of a pattern. But overall, this looks much better than our original model.

But… how to know which transformation to use…? It's a bit of trial and error. But we can use the model checking plots to help us.

Very very important is that we do this trial and error before we start using the model. E.g., we don't want to jump from the aeroplane and then find out that our parachute is not working properly! And then try to fix the parachute while we are falling….

Likewise, we must not start using the model and then try to fix it. We need to make sure our model is in good working order before we start using it.

One of the traps we could fall into is called "p-hacking". This is when we try different transformation until we find one that gives us the **result we want**, for example significant relationship. This is a big no-no in statistics. We need to decide on the model (including any transformations) before we start using it.

Common transformations

Which transformations could be considered? There is no simple answer. But some guidelines. E.g. if we see non-linearity and increasing variance with increasing fitted values, then a log transform may improve matter.

Some common and useful transformations are:

- The log transformation for concentrations and absolute values.
- The square-root transformation for count data.
- The arcsin square-root $\arcsin(\sqrt{\cdot})$ transformation for proportions/percentages.

Transformations can also be applied on explanatory variables, as we saw in the example above.

Interpreting a model with log-transformed variables

When we transform a variable, the coefficients of the model are on the transformed scale. With a square-root transformation they are hard to interpret. With a log transformation, however, they have useful interpretations in terms of *multiplicative* (proportional) changes. This matters a lot in practice: the slope of a model with a log-transformed response is not "the change in y per unit x".

Here are the three common cases, using natural logarithms (`log()` in R):

1. Log-transformed response: $\log(y) = \beta_0 + \beta_1 x$**.** Increasing x by one unit adds β_1 to $\log(y)$, which *multiplies* the expected y by e^{β_1}. For example, if $\beta_1 = 0.1$, then $e^{0.1} = 1.105$, so each unit increase in x is associated with a 10.5% increase in y. For small values of β_1 (say, between -0.1 and 0.1), $100 \cdot \beta_1$ is a good approximation of the percentage change.

2. Log-transformed explanatory variable: $y = \beta_0 + \beta_1 \log(x)$**.** Multiplying x by a factor k changes y by $\beta_1 \log(k)$. So doubling x is associated with a change in y of $\beta_1 \log(2) \approx 0.69 \cdot \beta_1$.

3. Both log-transformed (a "log–log" model): $\log(y) = \beta_0 + \beta_1 \log(x)$**.** This corresponds to the power law $y = e^{\beta_0} x^{\beta_1}$. A 1% increase in x is associated with an (approximately) β_1% change in y, and doubling x multiplies y by 2^{β_1}. Log–log models are common in biology, for example when describing how metabolic rate or other traits scale with body mass.

In the log–log model we fitted above (`m_log_log`), the estimated slope is 0.5. So doubling x is associated with y being multiplied by $2^{0.5} = 1.41$, that is, an increase of about 41%. (The data were simulated with a true slope of 0.5.)

Some practical points:

- **Back-transforming predictions.** To get predictions on the original scale, back-transform with `exp()`. For a log-transformed response, `exp()` of a predicted value estimates the *median* (more precisely, the geometric mean) of y at that value of x, not its mean. Back-transform confidence intervals by applying `exp()` to their limits; the resulting intervals are not symmetric around the prediction.
- **Which logarithm?** If base-10 logarithms are used (`log10()`), replace e^{β_1} with 10^{β_1} in case 1, and $\log(2)$ with $\log_{10}(2) \approx 0.30$ in case 2. The log–log interpretation (case 3) is the same whatever the base, as long as the same base is used for both variables. Always state which logarithm you used.

- **Zeros.** The logarithm of zero is undefined, so a log transformation cannot be applied directly to data containing zeros. Adding a constant (e.g. $\log(y+1)$) is sometimes done, but it changes the interpretation of the coefficients. For count data, a generalised linear model (Chapter 8) is usually a better solution.

Outliers

What do we do when we identify the presence of one or more outliers?

1. Start by checking the "correctness" of the data. Is there a typo or a decimal point that was shifted by mistake? Check both the response and explanatory variables.
2. If not, ask whether the model could be improved. Do reasonable transformations of the response and/or explanatory variables eliminate the outlier? Do the residuals have a distribution with a long tail (which makes it more likely that extreme observations occur)?
3. Sometimes, an outlier may be the most interesting observation in a dataset! Was the outlier created by some interesting but different process from the other data points?
4. Consider that outliers can also occur just by chance!
5. Only if you decide to report the results of both scenario can you check if inclusion/exclusion changes the qualitative conclusion, and by how much it changes the quantitative conclusion.

Removing outliers

It might seem tempting to remove observations that apparently don't fit into the picture. However:

- Do this **only with greatest care** e.g., if an observation has extremely implausible values!

- Before deleting outliers, check points 1-5 above.
- When removing outliers, **you must mention this in your report**.

During the course we'll see many more examples of things going at least a bit wrong. And we'll do our best to improve the model, so we can be confident in it, and start to use it. Which we will start to do in the next lesson. But before we wrap up, some good news…

Its a kind of magic...

Above, we learned about linear regression, the equation for it, how to estimate the coefficients, and how to check the assumptions. There was a lot of information, and it might seem a bit overwhelming.

You might also be aware that there are quite a few other types of statistical model, such as multiple regression, t-test, ANOVA, two-way ANOVA, and ANCOVA. It could be worrying to think that you need to learn so much new information for each of these types of tests.

But this is where the kind-of-magic happens. The good news is that the linear regression model is a special case of what is called a *general linear model*, or just *linear model* for short. And that all the tests mentioned above are also types of *linear model*. So, once you have learned about linear regression, you have learned a lot about linear models, and therefore also a lot about all of these other tests as well.

Moreover, the same function in R 'lm' is used to make all those statistical models Awesome.

So what is a linear model?

A common misconception is that a linear model is a model where the relationship between the dependent variable and the independent variables must be linear linear. Actually, we can have non-linear relationships in a linear model. For example, we can have quadratic terms in a linear model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

If this is still a linear model, what then is the meaning of "linear" in "linear model"? The answer is a little nuanced: a linear model is a mathematical model in which the expected value of the response variable is expressed as a **linear combination of parameters (coefficients).**

When we say a model has "a linear combination of parameters", we mean that the unknown coefficients (the β's) are only multiplied by things and added together… they are not exponentiated, for example.

That is, the dependent variable can be expressed as a linear combination of the independent variables. An example of a linear model is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon$$

where: y is the dependent variable, β_0 is the intercept, $\beta_1, \beta_2, ..., \beta_p$ are the coefficients of the independent variables, $x_1, x_2, ..., x_p$ are the independent variables, ϵ is the error term.

In contrast, a non-linear model is one where a parameter enters in a non-linear way. An example of a non-linear model is:

$$y = \beta_0 + \beta_1 e^{\beta_2 x} + \epsilon$$

where: y is the dependent variable, β_0 is the intercept, β_1, β_2 are the coefficients of the independent variables, x is the independent variable, ϵ is the error term.

In this model, changing the value of

$$\beta_2$$

does not change the value of

$$y$$

linearly. This happens because exponentiation is a non-linear operation.

Review

In this chapter we learned about first steps of regression analysis. Specifically, we learned about to:

- Motivation for regression analysis.
- The simple linear regression model.
- Estimation of the coefficients (intercept and slope) using the least squares method.
- How to model the errors (residuals) and the assumptions of linear regression.
- How to check the assumptions of linear regression using model checking plots.
- What can go wrong during the modeling process, and how to deal with it.

With all that in place, we can be sure that when we use a linear regression model, we can trust the results it gives us. In the next chapter, we will start to use linear regression models to answer questions! This involves things such as hypothesis testing, confidence intervals, and prediction. Exciting times ahead!

Further reading

There are many good books and online resources about regression models, fitting them, and checking their assumptions. Here are some suggestions for further reading:

- For a more mathematical treatment, including matrix representation, see Chapter 2 of Faraway, J. J. (2016). Linear models with R (2nd ed.). Chapman and Hall/CRC. Link
- The same book (Faraway) covers Diagnostics and model checking in Chapter 5.
- For a complementary perspective on regression, see Chapter 7 of The New Statistics with R, by Andy Hector.

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Making a QQ plot

With the quantile-quantile plot we are investigating the distribution of the residuals. We do this by comparing the quantiles of the residuals (the sample quantiles) to the quantiles of a normal distribution (the theoretical quantiles). If the residuals are normally distributed, then the points in the QQ-plot will approximately lie on a straight line.

What are the sample quantiles? Let us assume a sample of 20 residuals. We can order these residuals from smallest to largest. The we can calculate the proportion of data points that are smaller than each of these residuals. For example, the smallest residual will have 0% of the data points smaller than it, the second smallest residual will have 5% of the data points smaller than it, and so on. The largest residual will have 100% of the data points smaller than it. These proportions are called the sample quantiles.

Example for residuals sampled from a uniform distribution:

```
set.seed(121)

n <- 200
sim_residuals <- runif(n)
sim_residuals <- sim_residuals - mean(sim_residuals)

ordered_residuals <- sort(sim_residuals)

# plotting positions
p <- (1:n - 0.5) / n

# theoretical normal quantiles
theoretical_quantiles <- qnorm(p)

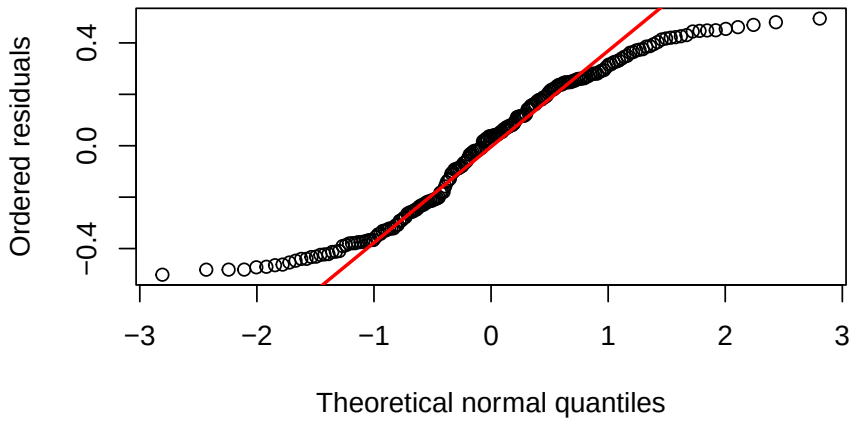
plot(theoretical_quantiles, ordered_residuals,
     xlab = "Theoretical normal quantiles",
     ylab = "Ordered residuals")

# --- replicate qqline() ---

sample_q <- quantile(sim_residuals, probs = c(0.25, 0.75))
theoretical_q <- qnorm(c(0.25, 0.75))
```

```
b <- diff(sample_q) / diff(theoretical_q)
a <- sample_q[1] - b * theoretical_q[1]

abline(a = a, b = b, col = "red", lwd = 2)
```




```
#qqnorm(sim_residuals)
#qqline(sim_residuals, col = "red", lwd = 2)
```

Tests for normality

We often want to know if our data or our residuals are not normally distributed. You've now done a lot with QQ-plots to examine how normally distributed your data are. Some people also like to have a statistical test for normality. Here I'll show some tests.

Shapiro–Wilk test on normally distributed data

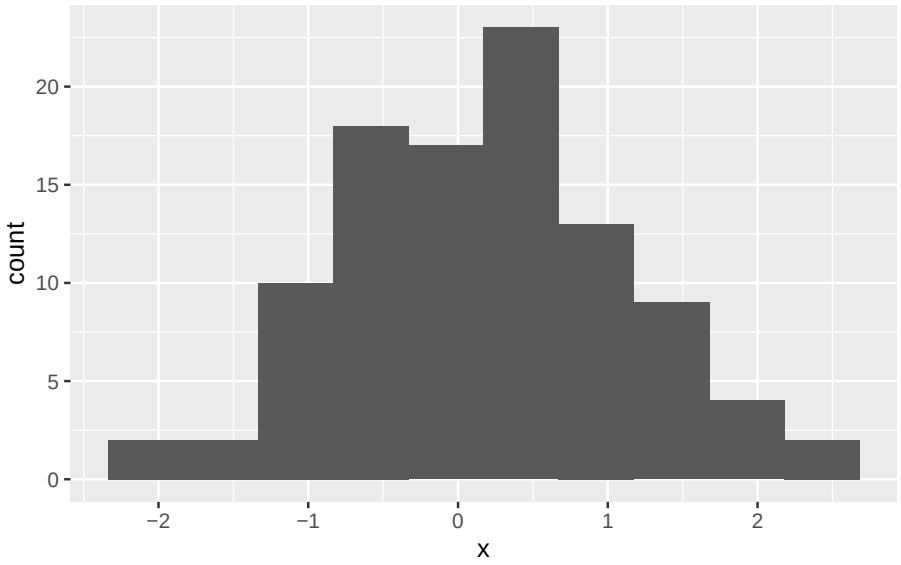
The Shapiro–Wilk test is frequently used.

- **Null hypothesis:** the data are normally distributed
- **Alternative hypothesis:** the data are not normally distributed

Let's make some data and look at its distribution and QQ-plot:

```
set.seed(145)
x <- rnorm(100)
```

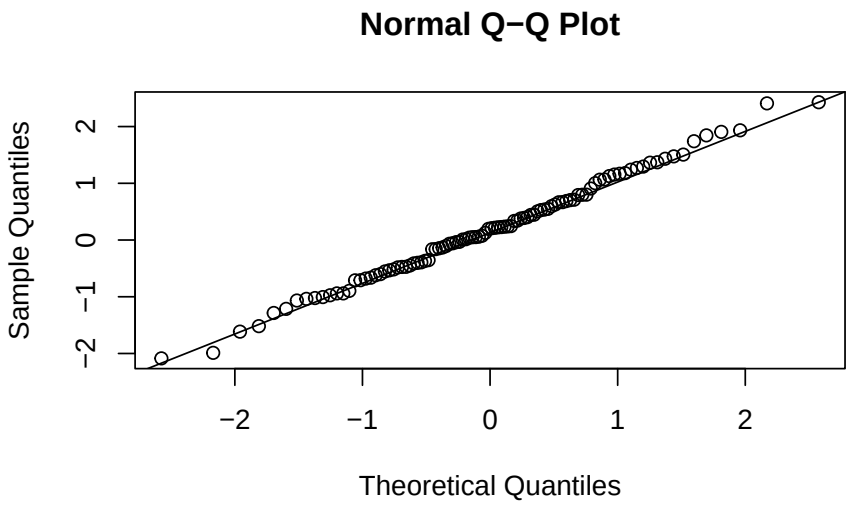
```
ggplot(data.frame(x=x), aes(x=x)) +
  geom_histogram(bins = 10)
```



Yes, this is the distribution of data that come from a normal distribution.

And the QQ-plot:

```
qqnorm(x)
qqline(x)
```



Looks pretty normal. Let's see what the Shapiro–Wilk test tells us:

```
shapiro.test(x)
```

```
    Shapiro-Wilk normality test

data:  x
W = 0.99467, p-value = 0.9658
```

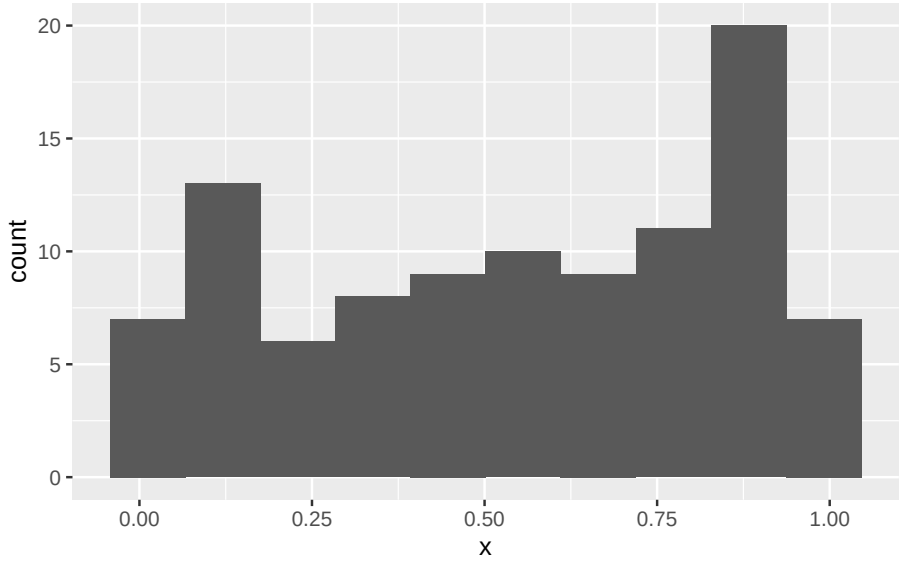
The p-value for the test is large (e.g. p 0.99). When we get a large p-value like this, it means we **cannot reject the null hypothesis**. (A small p-value, say p $<$ 0.05, would cause us to reject the null hypothesis.) Hence, here, we cannot reject the null hypothesis that the data are normally distributed. (Note that, as usual, technically we cannot *accept* the null hypothesis.)

So we can happily continue with any work that assumes normally distributed residuals.

Shapiro–Wilk test on non-normally distributed data

Now let's repeat this exercise using data that we *know* are not normally distributed.

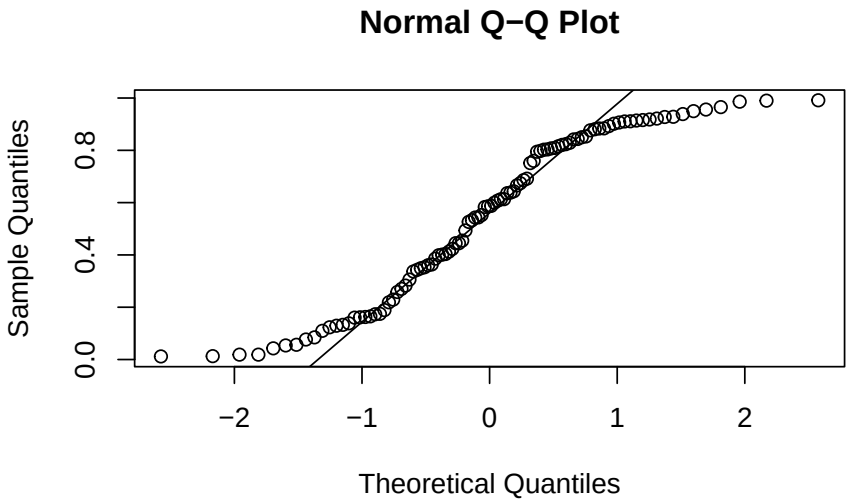
```
x <- runif(100)
ggplot(data.frame(x=x), aes(x=x)) +
  geom_histogram(bins = 10)
```



This is clearly not normally distributed data.

And the QQ-plot:

```
qqnorm(x)
qqline(x)
```



OK, we don't need to do a test to see that this is not normally distributed data. But let's do it anyway:

```
shapiro.test(x)
```

```
    Shapiro-Wilk normality test

data:  x
W = 0.92167, p-value = 1.728e-05
```

Now the p-value is small, so we **reject the null hypothesis** that the data are normally distributed. Now we can't continue with methods, such as linear models, that assume normally distributed residuals.

What if we have fewer data points?

What about if we have fewer data points, say 30? And let's do the test lots of times and count how often we reject the null hypothesis.

```
x <- replicate(1000, shapiro.test(runif(30))$p.value)
sum(x < 0.05)
```

```
[1] 371
```

So we reject the null hypothesis in about **38% of the trials**. That is, we *don't* reject the hypothesis that the data are normally distributed in nearly **40% of cases**, even though the data came from a uniform distribution!

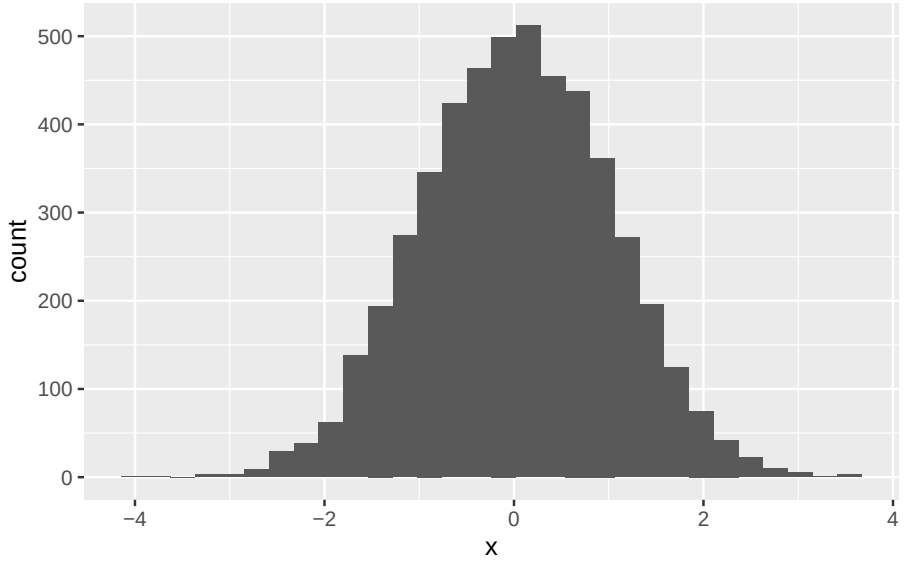
Shapiro–Wilk test on real data

How useful this (or any) test of normality is depends on our data.

- If we have a **large sample**, the test has very **high power**, and we can easily reject the null hypothesis due to tiny deviations from normality.
- If we have a **small sample**, we could easily fail to reject the null hypothesis even when the generating process does not create normally distributed data. The test has **low power** for small samples.

Here's an example of the latter point. We make data that has 5000 numbers sampled from a normal distribution, and then add a bit of non-normality:

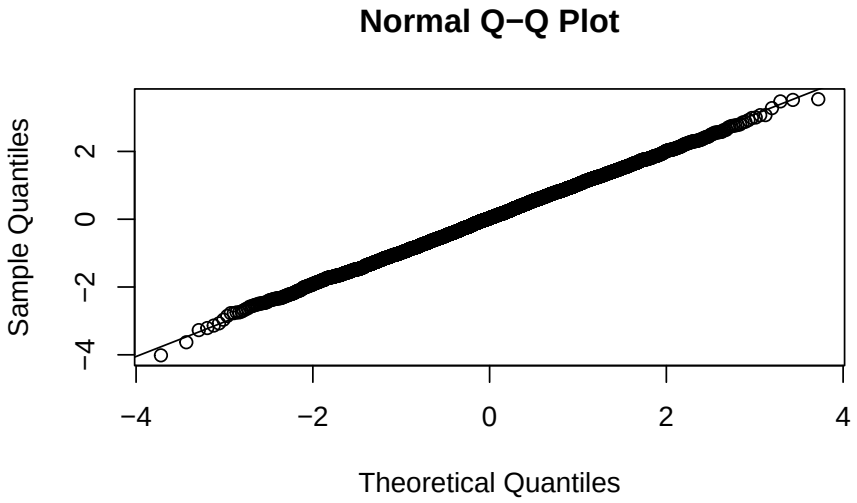
```
x <- rnorm(5000) #+ c(1, 0, 2, 0, 1)
ggplot(data.frame(x=x), aes(x=x)) +
  geom_histogram(bins = 30)
```



Looks pretty normal.

And the QQ-plot:

```
qqnorm(x)
qqline(x)
```



And the test.

```
shapiro.test(x)
```

```
    Shapiro-Wilk normality test

data:  x
W = 0.99966, p-value = 0.5779
```

Uh oh! The test rejects the null hypothesis.

So what do we do?

Actually, we might be focusing on the wrong question. Instead of asking *how close to normally distributed* our data are, we should consider:

- how **robust** our analysis is to deviations from normality, and
- which types of deviations from normality actually matter.

Generally speaking, **linear models are robust** to deviations from normality.

This means:

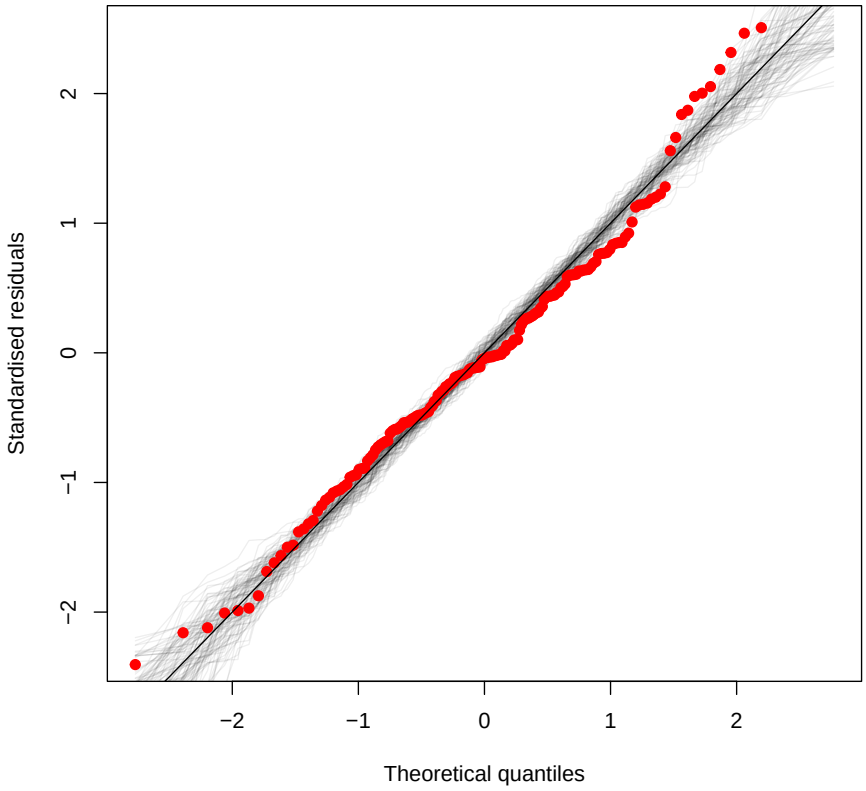
- we can be relatively happy that the Shapiro–Wilk test lacks power with small samples, and
- we should be cautious about rejecting normality when sample sizes are large.

More insight from QQ-plots

It is quite subjective to decide if a QQ-plot looks ok, or not. And if the tests are not so useful, what can we do?

One option is to make use of simulation to see what kind of variation we can expect in QQ-plots, even when the data really do come from a normal distribution.

Here is an example of such a plot (the code to make it is below):



The red dots are is the qqplot of the observed data. The grey lines are qqplots of normally distributed random data, with characteristics equal to those assumed by the model. Each grey line is a separate draw of random numbers. So the grey lines give an idea of what kind of spread one can expect even when the data really do come from a normal distribution. If the red points lie within the grey region, we're good to go!

Some of the upper red dots are at the edge of grey region, suggesting there might be minor deviations from normality. Hence, even with some help given by showing what we can expect to see, we still have to make an arbitrary / rather subjective decision about when deviations are bad enough to warrant doing something, and when they are not so bad, and we can continue with the assumption of normality. Much of data analysis is more subjective that we might expect given that it is quantitative!

Here is the code to make the graph, using some example data:

```
## function adapted from
## http://www.nate-miller.org/1/post/2013/03/how-normal-is-normal-a-q-q-plot-approach.
qqfunc <- function(model, num.reps) {

  N <- length(resid(model))
  sigma <- summary(model)$sigma

  x <- rnorm(N, 0, sigma)
  xx <- qqnorm(x, plot.it=F)
  xx$y <- xx$y[order(xx$x)]
  xx$x <- xx$x[order(xx$x)]
  plot(xx$x, scale(xx$y), pch=19, col="#00000011", type="l",
       xlab="Theoretical quantiles",
       ylab="Standardised residuals")
  ##qqline(x)

  for(i in 2:num.reps) {

    x <- rnorm(N, 0, sigma)
    xx <- qqnorm(x, plot.it=F)
    xx$y <- xx$y[order(xx$x)]
    xx$x <- xx$x[order(xx$x)]
    points(xx$x, scale(xx$y), pch=19, col="#00000011", type="l")

  }

  xx <- qqnorm(m_child_mort$residuals, plot.it=F)
  points(xx$x, scale(xx$y), col="red", pch=19)

}

## load some useful packages
library(readr)
library(tidyverse)
library(ggfortify)

## load the data
## Here I'm loading it direct from online where the dataset are stored
healthcare_data <- suppressMessages(readr::read_csv("https://raw.githubusercontent.com/

## tell me which countries are in the dataset
#unique(healthcare_data$country)

## tell me which years are in the dataset
#unique(healthcare_data$year)
```

```
## give me the rows in which both child mortality and health care expenditure are not NA
#filter(healthcare_data, !is.na(child_mort) & !is.na(health_exp_total))

## Wrange the data
healthcare_2013 <- healthcare_data |>
  filter(year==2013) |> ## only data fro 2013 please
  select(year:continent, health_exp_total, child_mort, life_expectancy) |> ## only these columns
  drop_na() ## drop rows with any NAs

## From last weeks work we know we need to log transform the data
healthcare_2013 <- mutate(healthcare_2013,
                log10_health_exp_total=log10(health_exp_total),
                log10_child_mort=log10(child_mort))

## plot the relationship between health care expenditure and child mortality
#ggplot(data=healthcare_2013, aes(x=health_exp_total, y=child_mort)) + geom_point()

## fit the linear model of the log transformed data and assign it to object named m_child_mort
m_child_mort <- lm(log10_child_mort ~ log10_health_exp_total, data=healthcare_2013)

## Here we run the function that makes the qqplot with some random samples from a normal distribu
qqfunc(m_child_mort, 100)
abline(0,1)
```

Regression Part 2 (L4)

Learning objectives

After this unit, students will be able to:

1. Explain what the correlation coefficient (r) and R^2 each measure, and how they differ.
2. Explain R^2 as the proportion of the total sum of squares explained by the model, and calculate it from the sums of squares.
3. Use R to obtain r and R^2 for a simple linear regression, and interpret their values.
4. Identify the null hypothesis tested by the t-test of a regression slope.
5. Explain what a p-value is, and calculate the t-statistic of a slope as its estimate divided by its standard error.
6. Explain how the number of degrees of freedom changes the t-distribution, and why fewer residual degrees of freedom give wider confidence intervals and larger p-values for the same t-statistic.
7. Use R to test whether the estimated slope is compatible with zero or another specified value, and draw the appropriate conclusion.
8. Use the residual degrees of freedom $(n - 2)$ to obtain the critical t-value for a confidence interval and the p-value for a slope, and identify the degrees of freedom in R output.
9. Use R to obtain a 95% confidence interval for the slope and intercept, and interpret it.
10. Distinguish a confidence band from a prediction band, and explain why the prediction band is wider.
11. Use `predict()` to calculate the expected value of the response, with its confidence interval or prediction interval, for a new value of the explanatory variable.
12. Interpret the slope and intercept in the biological units of the study, including whether the intercept is biologically meaningful.
13. Recognise a correct and complete statement of regression results that reports the estimate, its uncertainty and the test result.
14. Evaluate whether a regression analysis supports a stated biological

claim, considering R^2 versus statistical significance, extrapolation beyond the data, model diagnostics, and correlation versus causation.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch04_regression_part2.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 4 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch04_regression_part2.R")
```

Video walkthrough: coming soon.

Introduction

This chapter builds on the previous chapter on simple linear regression. There we learned how to fit a regression model to data, and how to check if the assumptions of the regression model are met. When we are satisfied that the assumptions of linear regression are adequately met, we can move on to the next stage: interpreting the model and using it to answer our biological question.

Interpretation of the model means two main things in simple regression:

- Is there a statistically significant relationship between the response and explanatory variable?
- Does the model provide a good (or not so good) explanation of the variation in the response variable?

And there are couple of tools useful for communicating and visualising the uncertainty in the regression model, such as *confidence interval*, *confidence bands*, and *prediction bands*.

So, in this chapter, we cover:

- How to measure how good is the regression (R^2).

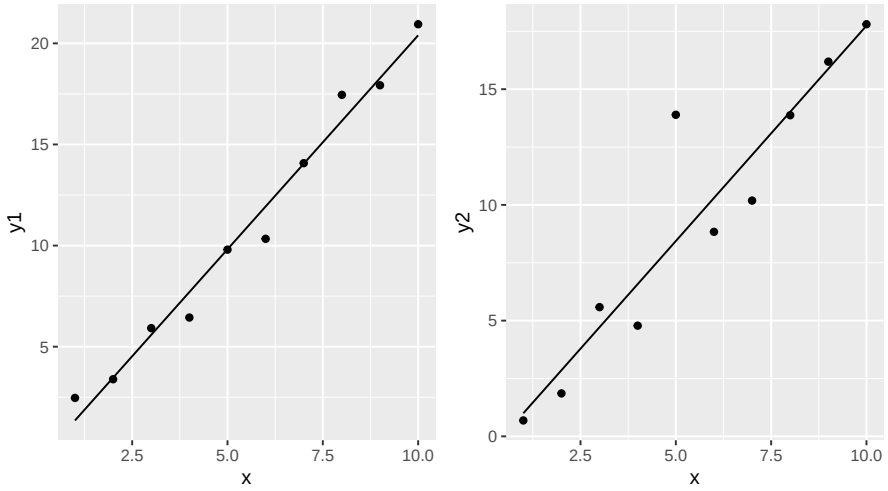
- Testing if relationship is statistically significant (i.e., if the slope is significantly different from zero).
- What is the confidence interval of the slope.
- What is the the confidence band of the regression line.
- What is the prediction band of the regression line.

How good is the regression model (R^2)?

What would a good regression model look like? What would a bad one look like? One could say that a good regression model is one that explains a lot the variability in the response variable. But what could we mean by "variability in the response variable" and "explains a lot"?

Take these two examples. The code makes two example datasets: in `y1` the data are close to a straight line (residual standard deviation 1), in `y2` they are more scattered (residual standard deviation 5). You do not need to understand the `rnorm()` part.

```
set.seed(1233)
x <- 1:10
y1 <- 2*x + rnorm(10,0,1)
y2 <- 2*x + rnorm(10,0,5)
```



The left regression model seems to fit the data relatively well. The regression line is quite close to the data points.

The right regression model does not fit the data well. The regression line is quite far from the data points.

But how can we quantify this?

Let's say that we will measure the goodness of the model by the proportion of variability of the response variable that is explained by the explanatory variable. To do this we need to do the following:

1. Measure the total variability of the response variable (total sum of squares, SST).
2. Measure the amount of variability of the response variable that is explained by the explanatory variable (model sum of squares, SSM).
3. Measure the variability of the response variable that is not explained by the explanatory variable (error sum of squares, SSE).
4. Calculate the proportion of variability of the response variable that is explained by the explanatory variable (R^2, pronounced as "r-squared") (also known as the coefficient of determination) ($R^2 = SSM/SST$).

Importantly, note that we will calculate SSM and SSE so that they sum up to SST. I.e., $SST = SSM + SSE$. That is, the total variability is the sum of what is explained by the model and what remains unexplained.

Let's take each in turn:

SST

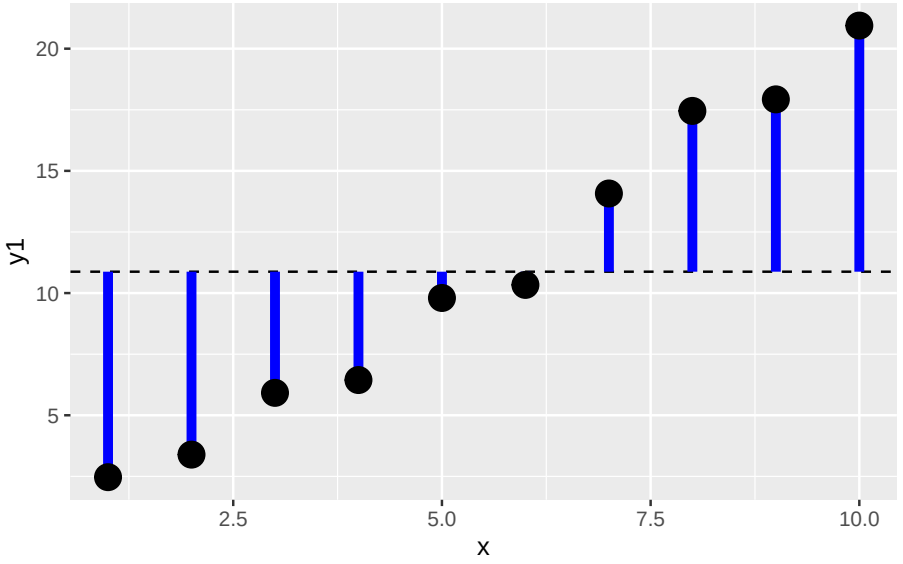
1. The total variability of the response variable is the sum of the squared differences between the response variable and its mean. This is called the total sum of squares (SST).

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where: y_i is the response variable, $\bar{y}$ is the mean of the response variable, n is the number of observations.

Note that sometimes SST is referred to as SSY (sum of squares of y).

Graphically, this is the sum of the square of the blue residuals as shown in the following graph, where the horizontal dashed line is at the value of the mean of the response variable.



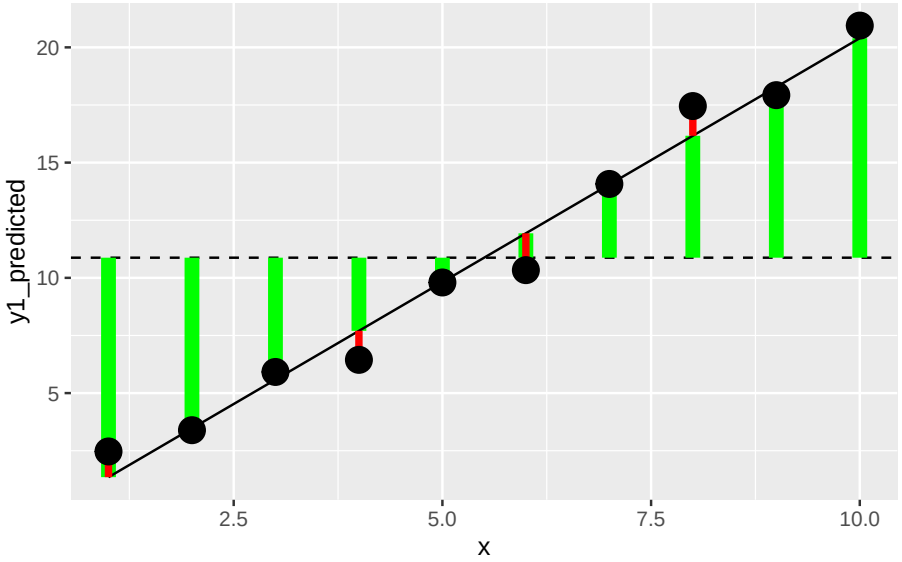
We can calculate this in R as follows:

```
SST <- sum((y1 - mean(y1))^2)
```

SSM and SSE

Now the next two steps, that is getting the model sum of squares (SSM) and the error sum of squares (SSE) are a bit more complicated. To do this we need to fit a regression model to the data. Let's see this graphically, and divide the data into the explained and unexplained parts.

Make a graph with vertical lines connecting the data to the mean of the data, but with each line two parts, one from the mean to the data, and one from the data to the predicted value.



In this graph, the square of the length of the green lines is the model sum of squares (SSM). The square of the length of the red lines is the error sum of squares (SSE).

In a better model the length of the green lines will be **longer** (the square of these gives the SSM, the variability explained by the model). And the length of the red lines will be **shorter** (the square of these gives the SSE, the variability not explained by the model).

SSM

Next we will do the second step, that is calculate the model sum of squares (SSM).

2. The amount of variability of the response variable that is explained by the explanatory variable is called the model sum of squares (SSM).

This is the difference between the predicted value of the response variable and the mean of the response variable, squared and summed:

$$SSM = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$$

where: $\hat{y}_i$ is the predicted value of the response variable,

In R, we calculate this as follows:

```
m_sd1 <- lm(y1 ~ x)
y1_predicted <- predict(m_sd1)
```

```
SSM <- sum((y1_predicted - mean(y1))^2)
SSM
```

```
[1] 369.4306
```

SSE

Third, we calculate the error sum of squares (SSE) with either of two methods. We could calculate it as the sum of the squared residuals, or as the difference between the total sum of squares and the model sum of squares:

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = SST - SSM$$

Let's calculate this in R uses both approaches:

```
SSE <- sum((y1 - y1_predicted)^2)
SSE
```

```
[1] 7.586091
```

Or...

```
SSE <- SST - SSM
SSE
```

```
[1] 7.586091
```

R^2

Finally, we calculate the proportion of variability of the response variable that is explained by the explanatory variable (R^2):

$$R^2 = \frac{SSM}{SST}$$

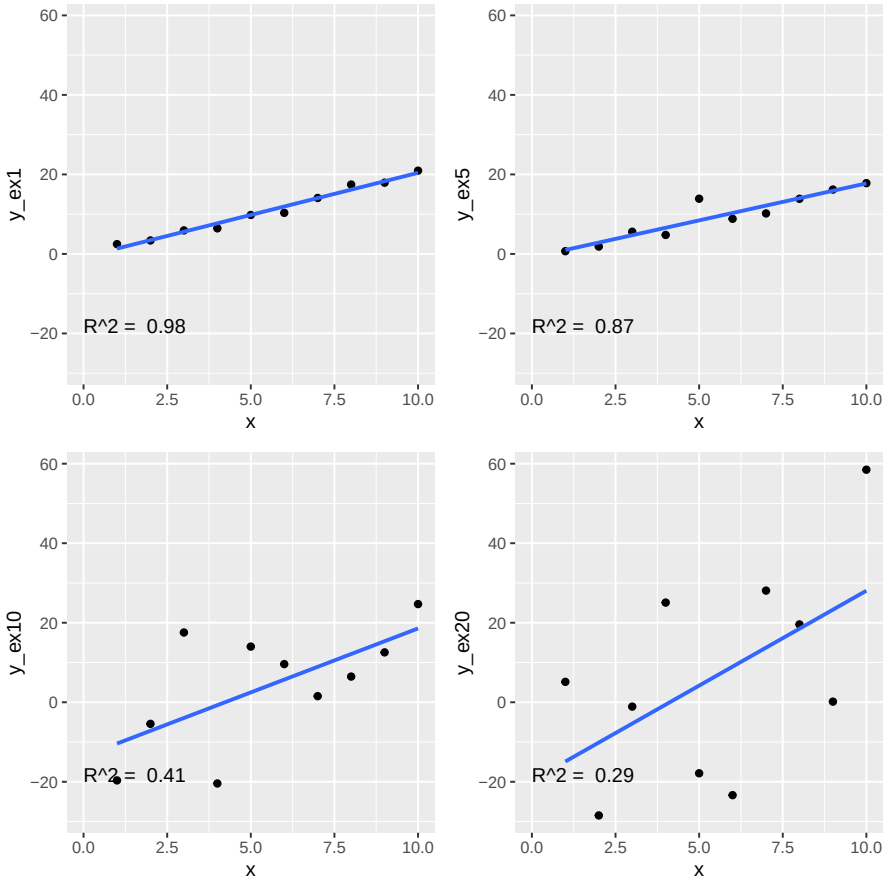
```
R.squared <- SSM/SST
R.squared
```

```
[1] 0.9798786
```

Is my R squared high (or low)?

What value of R^2 is considered good? In ecological research, R^2 values are often low (less than 0.3), because ecological systems are complex and many factors influence the response variable. However, in other fields, such as physiology, R^2 values are often higher. Therefore, the answer of what values of R^2 are good depends on the field of research.

Here are the four examples and their r-squared.



Where is the R^2 value in the regression output?

Using the blood pressure data, we can fit a regression model and look at the output. We use the `summary` function to get the regression output, and we can see that the R^2 value is given in the output. Look for "Multiple R-squared" in the output, which is the R^2 value of the model.

```
bp_data <- read_csv("datasets/Simulated_Blood_Pressure_and_Age_Data.csv")
```

```
Rows: 100 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): Age, Systolic_BP

i Use `spec()` to retrieve the full column specification for this data.
```

```
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
bp_age_model <- lm(Systolic_BP ~ Age, data = bp_data)
summary(bp_age_model)
```

```

Call:
lm(formula = Systolic_BP ~ Age, data = bp_data)

Residuals:
     Min       1Q   Median       3Q      Max
-13.2195  -3.4434  -0.0808   3.1383  12.6025

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 98.96874    1.46102   67.74   <2e-16 ***
Age          0.82407    0.02771   29.74   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.971 on 98 degrees of freedom
Multiple R-squared:  0.9002,    Adjusted R-squared:  0.8992
F-statistic: 884.4 on 1 and 98 DF,  p-value: < 2.2e-16
```

The value is 0.9, which means that the model explains about 90% of the variability in the response variable (systolic blood pressure).

Is the relationship *statistically significant*?

A common question is "Is there a relationship between the response and explanatory variable?" Or, more formally, "Is the slope of the regression line significantly different from zero?"

We often hear this expressed as "is the relationship statistically significant?" And maybe we heard that the relationship is significant if the p-value is less than 0.05. But what does all this actually mean? In this section we'll figure all this out. The first step to is to formulate a null hypothesis.

The null hypothesis

What is a meaningful null hypothesis for a regression model?

As mentioned, often we're interested in whether there is a relationship between the dependent (response) and independent (explanatory) variable. Therefore, the null hypothesis is that there is no relationship between the response and explanatory variable. This means that the null hypothesis is that the slope of the regression line is zero.

Recall the regression model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

The null hypothesis is that the slope of the regression line is zero:

$$H_0 : \beta_1 = 0$$

What is the alternative hypothesis?

$$H_1 : \beta_1 \neq 0$$

So, how do we test the null hypothesis? We are going to calculate the probability of observing the data we have, given that the null hypothesis is true. If this probability is very low, then we can reject the null hypothesis.

> **Note**
>
> We never prove the null hypothesis is true. Instead, we calculate the probability of observing our data given that the null hypothesis is true. If this probability is very low, we reject the null hypothesis.

To calculation this probability we use the fact that the slope of the regression line is an estimate and this estimate has uncertainty associated with it.

We can see that the slope estimate (the x row) has uncertainty by looking at the regression output:

```
summary(m_sd1)$coefficients[1:2, 1:2]
```

```
             Estimate Std. Error
(Intercept) -0.7638353  0.6652233
x            2.1161160  0.1072104
```

We see the estimate of the parameter (slope) and a measure of uncertainty in the estimate: the standard error of the estimate.

The standard error is calculated as:

$$\sigma^{(\beta_1)} = \sqrt{\frac{\hat{\sigma}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

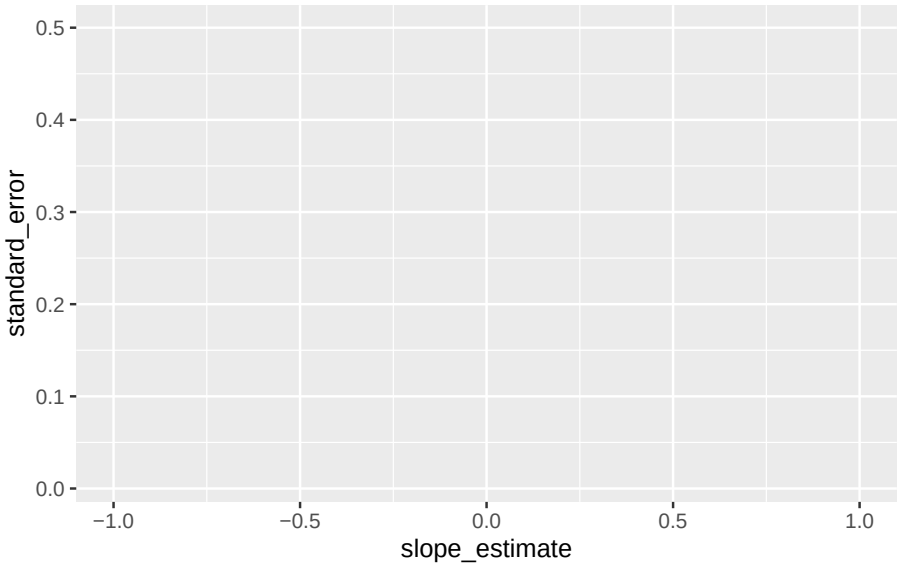
Where $\hat{\sigma}^2$ is the expected residual variance of the model. This is calculated as:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-2}$$

Where $\hat{y}_i$ is the predicted value of y_i from the regression model, and n is the number of observations. We divide by $n - 2$ because we have estimated two parameters (the intercept and the slope) from the data.

OK, let's take a look at this intuitively. We have the estimate of the slope and the standard error of the estimate of the slope (when this standard error is larger, we are less certain about the estimate).

Here is an empty graph of the value of the slope estimate versus the standard error of the estimate:



Have a careful think about what combinations of slope estimate value and standard error value are more likely to have been observed by chance, and which combinations are less likely to have been observed by chance. Draw a copy of the graph and label areas where the slope estimate is more likely to have been observed by chance, and areas where it is less likely to have been observed by chance. (The answer is at the end of this chapter, so think about it before you look at the answer).

When the slope estimate is larger, it is less likely to have been observed by chance. And when the standard error is larger, it is more likely to have been observed by chance. How can we put these together into a single measure?

The *t*-statistic

If we divide the slope estimate by the standard error, we get a measure of how many standard errors the slope estimate is from the null hypothesis slope of zero. This is the *t*-statistic:

$$t = \frac{\hat{\beta}_1 - \beta_{1,H_0}}{\sigma^{(\beta_1)}}$$

Where β_{1,H_0} is the null hypothesis value of the slope, usually zero, so that

$$t = \frac{\hat{\beta}_1}{\sigma^{(\beta_1)}}$$

The t-statistic is a measure of how many standard errors the slope estimate is from the null hypothesis value of the slope. The larger the t-statistic, the less likely the slope estimate was observed by chance.

The p-value

Recap: Formal definition of the p-value

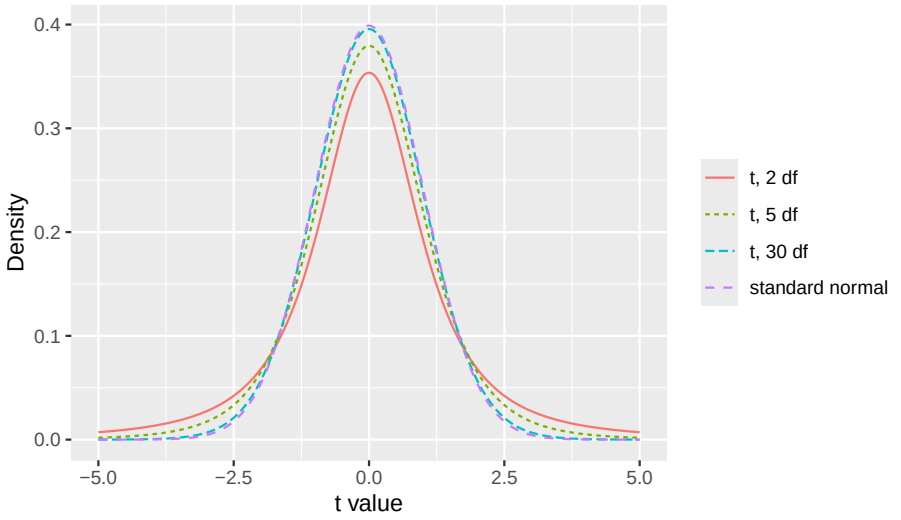
The formal definition of p-value is the probability to observe a data summary that is at least as extreme as the one observed, given that the null hypothesis is correct.

When performing a t-test, the data summary is the value of the t-statistic. Therefore, the p-value is the probability of observing a value of the t-statistic at least as extreme as the one observed, given that the null hypothesis is correct.

But how can we calculate the p-value from the t-statistic? The answer is to use the **t-distribution**, which quantifies the probability of observing a value of the t-statistic under the null hypothesis.

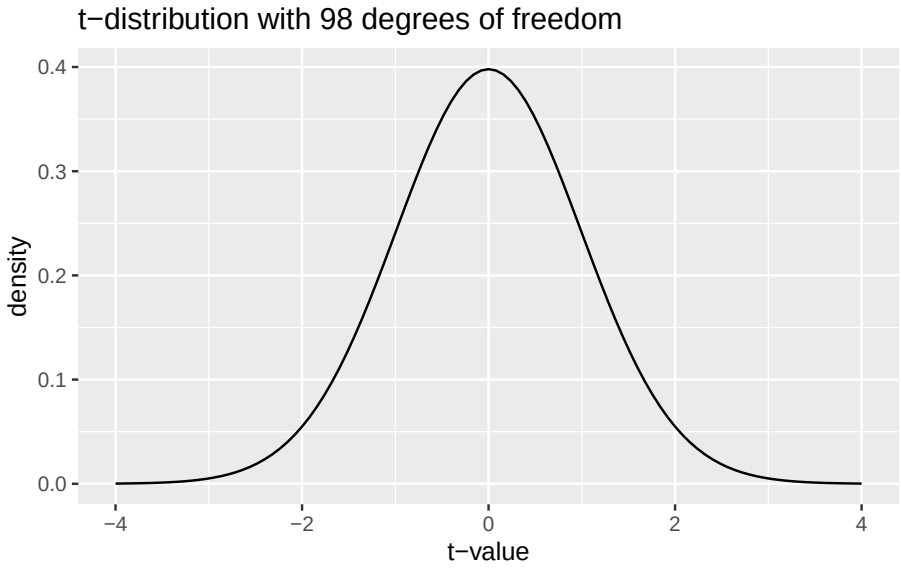
But what is the t-distribution? It is a distribution of the t-statistic under the null hypothesis. It is a bell-shaped distribution that is centered on zero. The shape of the distribution is determined by the degrees of freedom, which is $n-2$ for a simple linear regression model.

With few degrees of freedom the t-distribution has heavier tails than the normal distribution: extreme values of t are more likely, because the residual variance (and so the standard error) is estimated less precisely. As the degrees of freedom increase, the t-distribution gets closer and closer to the standard normal distribution:



This is why, as we saw in the table of critical t-values in Chapter 3, fewer residual degrees of freedom give larger critical values, wider confidence intervals, and larger p-values for the same t-statistic.

Here is the distribution of t values under the null hypothesis for a sample size of 100 (i.e., 98 degrees of freedom):



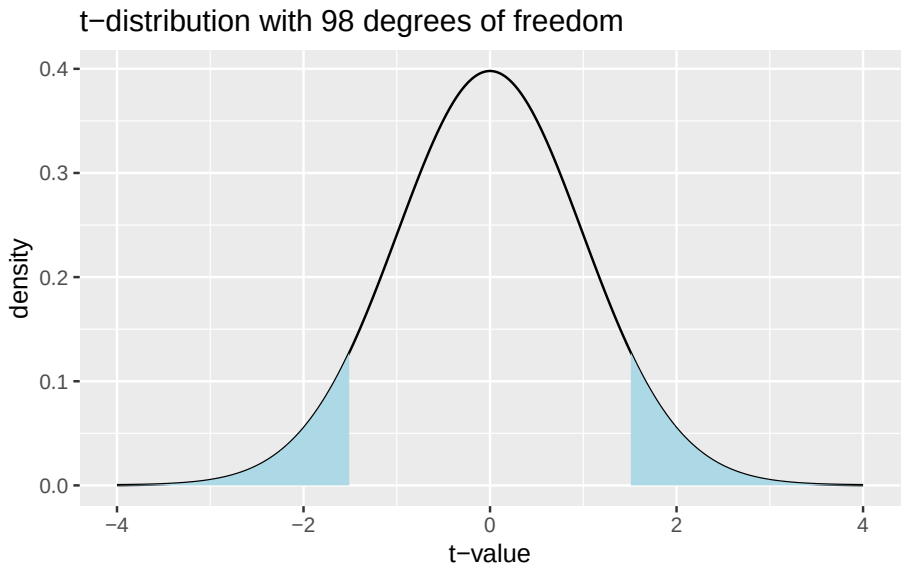
The height of the line is the "density" and not a probability. The probability of observing a value of the t-statistic in a given range is the area under the curve of the t-distribution in that range. The area under the curve of the t-distribution

is equal to 1, which means that the total probability of observing a value of the t-statistic is 1.

> **Tip**
>
> By the way, it is named the t-distribution by it's developer, William Sealy Gosset, who worked for the Guinness brewery in Dublin, Ireland. In his 1908 paper, Gosset introduced the t-distribution but he didn't explicitly explain his choice of the letter t. The choice of the letter t could be to indicate "Test", as the t-distribution was developed specifically for hypothesis testing.

Now, recall that the p-value is the probability of observing the value of the test statistic (so here the t-statistic) at least as extreme as the one we have, given the null hypothesis is true. We can calculate this probability by integrating the t-distribution from the observed t-statistic to the tails of the distribution.

Let's see this graphically. Say we have an observed t-statistic of 1.5. The p-value is the area under the curve of the t-distribution in the tails beyond 1.5 and -1.5 (more extreme means both less than or greater than). Graphically this looks like:



This blue area is the p-value, which is the probability of observing a value of the t-statistic at least as extreme as 1.5, given that the null hypothesis is true.

This area (p-value) can be calculated using the cumulative distribution function of the t-distribution.

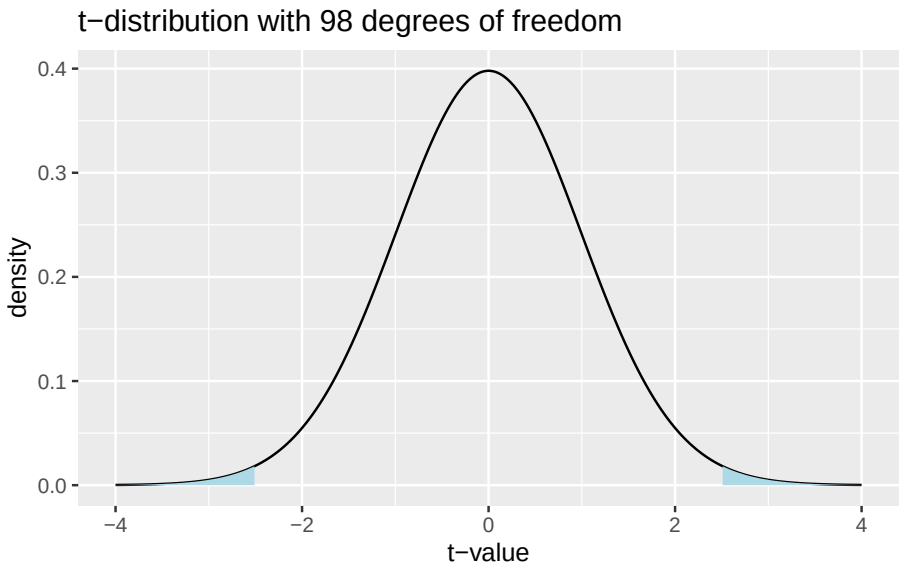
```
n <- 100
t_value <- 1.5
p_value <- 2 * pt(-abs(t_value), df = n - 2)
p_value
```

```
[1] 0.1368292
```

So, with a t-statistic of 1.5, and 98 degrees of freedom, the p-value is 0.14, which means that there is a 0.14 probability of observing a value of the t-statistic at least as extreme as 1.5, given that the null hypothesis is true.

Often we say that there is "statistically significant evidence" for a relationship between the response and explanatory variable if the p-value is less than 0.05. This means that if the p-value is less than 0.05, we reject the null hypothesis of no relationship between the response and explanatory variable.

Let's do the same graph and calculation for a t-statistic of 2.5:



With the larger t-statistic the blue areas are smaller, which means that the p-value is smaller.

```
t_value <- 2.5
p_value <- 2 * pt(-abs(t_value), df = n - 2)
p_value
```

```
[1] 0.01407976
```

The p-value is 0.014, which means that there is a 0.014 probability of observing a value of the t-statistic at least as extreme as 2.5, given that the null hypothesis

is true. Therefore, we reject the null hypothesis of no relationship between the response and explanatory variable.

Where is the p-value in the regression output?

Using the blood pressure data again, we can fit a regression model and look at the output. We use the `summary` function to get the regression output.

```
summary(bp_age_model)
```

```
Call:
lm(formula = Systolic_BP ~ Age, data = bp_data)

Residuals:
     Min       1Q   Median       3Q      Max
-13.2195  -3.4434  -0.0808   3.1383  12.6025

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 98.96874    1.46102   67.74   <2e-16 ***
Age          0.82407    0.02771   29.74   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.971 on 98 degrees of freedom
Multiple R-squared:  0.9002,    Adjusted R-squared:  0.8992
F-statistic: 884.4 on 1 and 98 DF,  p-value: < 2.2e-16
```

In the coefficients table, look for the row corresponding to the slope (the second row, which is the "Age" row). In this row we see the estimate of the slope, the standard error of the slope, the t-value for the slope,and in the last column "Pr(>|t|)". This means "the probability of observing a value of the t-statistic at least as extreme as the one observed, given that the null hypothesis is true".

> **Note**
>
> The vertical bars around t indicate that we are looking at the absolute value of the t-statistic, which is what we use for a two-tailed test, which is the most common test for the slope of a regression line

Conclusion: there is **very strong evidence** that the blood pressure is associated with age, because the p-value is extremely small (thus it is very unlikely that the observed slope value or a large one would be seen if there was really no association). Thus, we can reject the null hypothesis that the slope is zero.

A cautionary note on the use of p-values

Maybe you have seen that in statistical testing, often the criterion $p \leq 0.05$ is used to test whether H_0 should be rejected. This is often done in a black-or-white manner. However, we will put a lot of attention to a more reasonable and cautionary interpretation of p-values in this course!

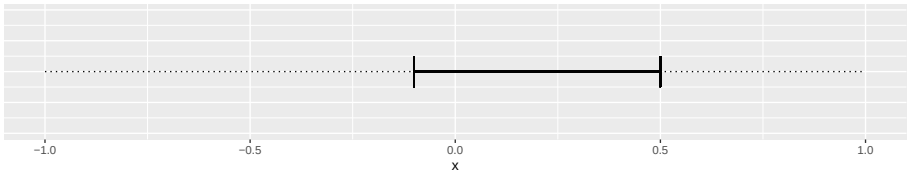
Confidence interval of slope

The actual value of the slope has practical meaning. The slope of the regression line tells us how much the response variable changes when the explanatory variable changes by one unit. The slope is one measure of the strength of the relationship between the two variables.

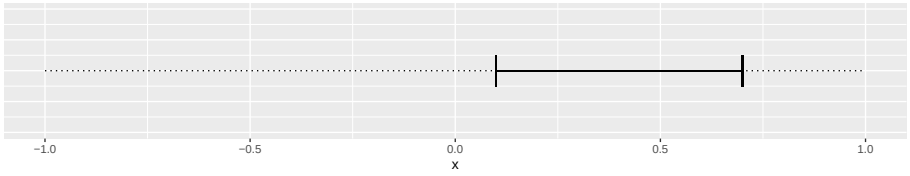
We can ask what values of a parameter estimate are compatible with the data (confidence intervals)? To answer this question, we can determine the confidence intervals of the regression parameters.

A 95% confidence interval is calculated with a procedure that, if we repeated the study many times and calculated an interval from each new sample, would give intervals that contain the true parameter value in 95% of the repetitions. Any one interval either contains the true value or it does not; the 95% refers to the procedure, not to the particular interval we calculated. A useful, informal way to think of a 95% confidence interval of the slope is as the range of values of the slope that are compatible with the data.

We can then imagine two cases. The 95% confidence interval of the slope includes 0:



Or where the confidence interval does not include zero:



How do we calculate the lower and upper limits of the 95% confidence interval of the slope?

Recall that the t-value for a null hypothesis of slope of zero is defined as:

$$t = \frac{\hat{\beta}_1}{\hat{\sigma}^{(\beta_1)}}$$

The first step is to calculate the t-value that corresponds to a p-value of 0.05. This is the t-value that corresponds to the 97.5% quantile of the t-distribution with $n-2$ degrees of freedom.

$t_{0.975} = -t_{0.025} = 1.96$, for large n.

The 95% confidence interval of the slope is then given by:

$$\hat{\beta}_1 \pm t_{0.975} \cdot \hat{\sigma}^{(\beta_1)}$$

In our blood pressure example the estimated slope is 0.8240678 and the standard error of the slope is 0.0277096. We can calculate the 95% confidence interval of the slope in R as follows:

```
n <- 100
t_0975 <- qt(0.975, df = n - 2)
half_interval <- t_0975 * summary(bp_age_model)$coef[2,2]
lower_limit <- coef(bp_age_model)[2] - half_interval
upper_limit <- coef(bp_age_model)[2] + half_interval
ci_slope <- c(lower_limit, upper_limit)
slope <- coef(bp_age_model)[2]
slope
```

```
      Age
0.8240678
```

```
ci_slope
```

```
      Age       Age
0.7690791 0.8790565
```

Or, using the `confint` function:

```
## 95% confidence interval of the slope of bp_age_model
ci_slope_2 <- confint(bp_age_model, level=c(0.95))[2,]
ci_slope_2
```

```
    2.5 %    97.5 %
0.7690791 0.8790565
```

Or we can do it using values from the coefficients table:

```
bp_age_coefs <- summary(bp_age_model)$coef
beta <- bp_age_coefs[2,1]
sdbeta <- bp_age_coefs[2,2]
beta + c(-1,1) * qt(0.975, df = df.residual(bp_age_model)) * sdbeta
```

```
[1] 0.7690791 0.8790565
```

Interpretation: for an increase in the age by one year, roughly 0.82 mmHg increase in blood pressure is expected, and all true values for β_1 between 0.77 and 0.88 are compatible with the observed data.

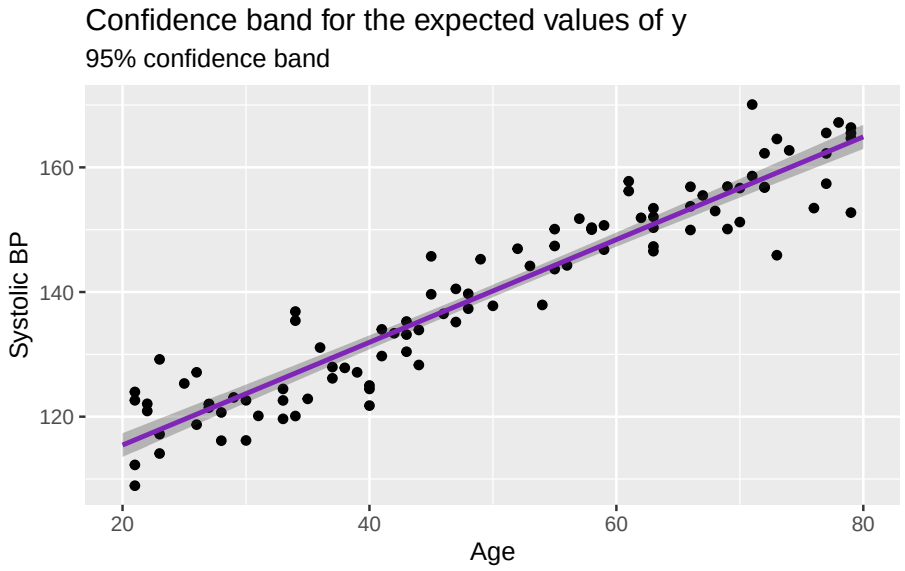
Confidence band

If another sample from the same population was taken, the regression line would look slightly different. But how different? And where could we expect the regression line to be if we took another sample from the same population? The *confidence band* is an illustration of the region in which the regression line could lie if we took another sample from the same population. A confidence band is always specified with a confidence level, such as 95%.

Lets look at this in a graph of our blood pressure data. The black line is the regression line fitted to the data. The purple line is the confidence band for the expected values of y (systolic blood pressure) for each value of x (age). The shaded area around the purple line is the 95% confidence band for the expected values of y.

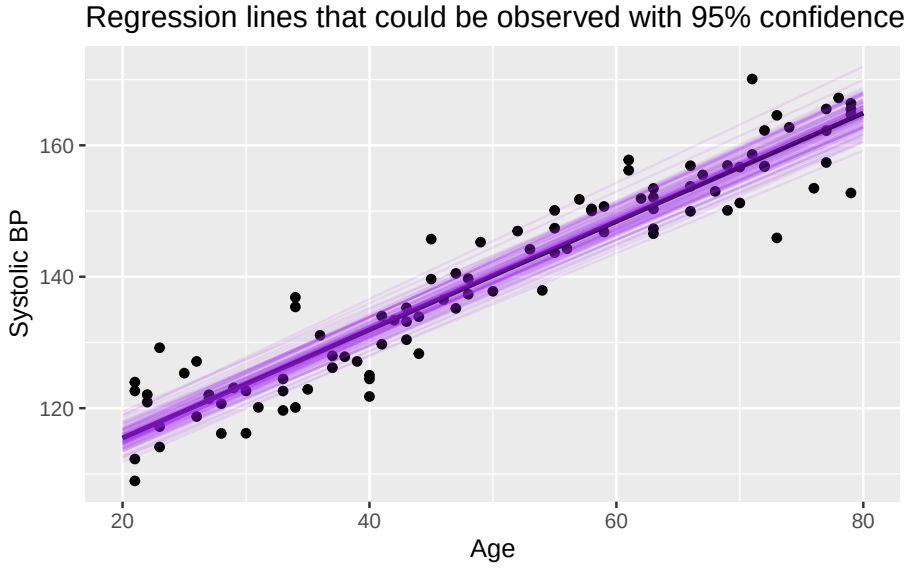
We can calculate the confidence band with the `predict()` function, using `interval = "confidence"`, and then plot it:

```
## calculate the confidence band for the expected values of y
new_data <- data.frame(Age = seq(20, 80, by = 1))
conf_band <- cbind(new_data,
                   predict(bp_age_model, newdata = new_data, interval = "confidence", level = 0.9
ggplot(bp_data, aes(x = Age, y = Systolic_BP)) +
  geom_point() +
  geom_line(aes(y = fit), data = conf_band, color = "purple", lwd = 1) +
  geom_ribbon(aes(ymin = lwr, ymax = upr, y = fit), data = conf_band, alpha = 0.3) +
  labs(title = "Confidence band for the expected values of y",
       subtitle = "95% confidence band",
       x = "Age", y = "Systolic BP")
```



The confidence band is narrow here because the estimates of the slope and intercept are precise (the standard errors of the slope and intercept are small). The confidence band is (slightly) wider at the edges of the graph because there are fewer data points at the edges, which means that the estimates of the slope and intercept are less precise at the edges.

We can view this in a different way. Let us plot regression lines that are consistent with the estimated slope and intercept, given their standard errors.



None of these lines has a slope any where near zero! And all are closely grouped around the estimated regression line. Note that this is a different way of visualising the confidence in the slope, and is not commonly used. Most people would just plot the 95% confidence band.

The details of how to calculate a confidence band are not required for this course. If you would like to see them, please look in the *Extras* section below.

Note: For the confidence band, only the uncertainty in the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ matters.

Prediction band

The prediction band is rarely used, but it is important to understand the difference between a confidence band and a prediction band. As seen above, the confidence band is where expected values of y lie. This means that we only account for the uncertainty in the estimates of the slope and intercept.

The prediction band is where future observations of y lie. This means that we have to account for both the uncertainty in the estimates of the slope and intercept, and the variability of future observations around the predicted value.

That is, we have to *consider not only the uncertainty in the predicted value caused by uncertainty in the parameter estimates* $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$, but also the *error term* $\epsilon_i \sim N(0, \sigma^2)$}.

If you'd like to learn how to calculate the prediction band, the details are given in the *Extras* section below. These details are not necessary parts of this course.

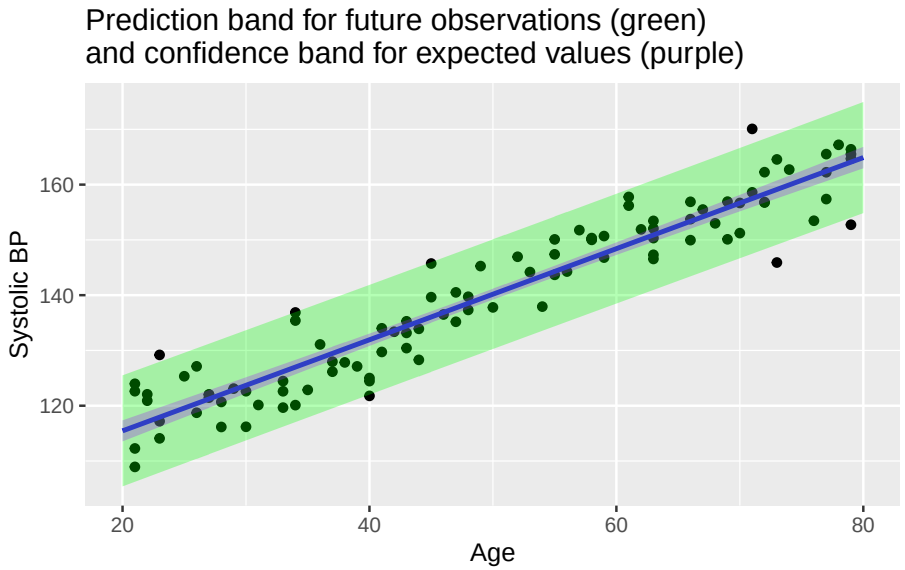
The key point is that the standard error of a future observation is larger than the standard error of the predicted value $\hat{y}_0$ because it includes an additional term that accounts for the variability of future observations around the predicted value.

Here's a graph showing the prediction band in green for the blood pressure data (with the confidence band in purple):

The code is the same as for the confidence band, but with `interval = "prediction"`:

```
prediction_band <- cbind(new_data,
                  predict(bp_age_model, newdata = new_data, interval = "prediction", level = 0.9
ggplot(bp_data, aes(x = Age, y = Systolic_BP)) +
  geom_point() +
  geom_line(aes(y = fit), data = prediction_band, color = "blue", lwd = 1) +
  geom_ribbon(aes(ymin = lwr, ymax = upr, y = fit), data = prediction_band, alpha = 0.3, fill = '
  geom_ribbon(aes(ymin = lwr, ymax = upr, y = fit), data = conf_band, alpha = 0.3,
              fill = "purple") +
```

```
  labs(title = "Prediction band for future observations (green)\nand confidence band fo
       x = "Age", y = "Systolic BP")
```



Another way to think of the 95% confidence band is that it is where we would expect 95% of the regression lines to lie if we were to collect many samples from the same population. The 95% prediction band is where we would expect 95% of the future observations to lie.

Extrapolation: predicting beyond the data

A regression model describes the relationship only over the range of values of the explanatory variable that we observed. In the blood pressure data, ages range from 21 to 79 years. Using the model to make predictions outside this range is called **extrapolation**, and it is risky:

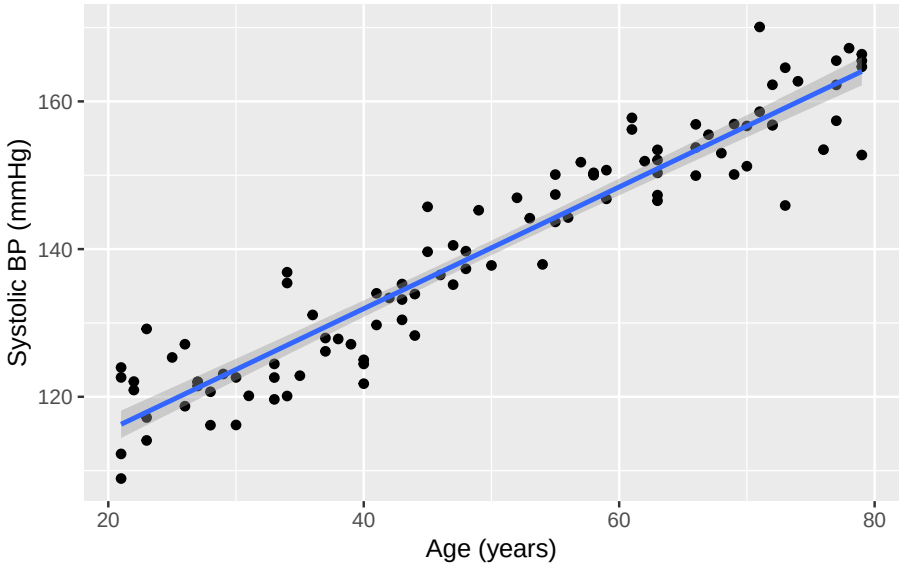
- We have no data outside the observed range, so we have no way to check whether the relationship is still linear there. For example, blood pressure may not keep increasing at the same rate into very old age.
- Confidence and prediction bands become wider the further we move from the mean of the explanatory variable. But even these bands assume that the straight line still holds outside the data, so when extrapolating they understate the real uncertainty.

The intercept is a common example of an unintended extrapolation. It is the expected value of y when $x = 0$. Here the estimated intercept is 99 mmHg: the predicted systolic blood pressure of a newborn baby. Age 0 is far outside the range of the data, so this value is not biologically meaningful; it is simply where

the fitted line crosses the vertical axis. If we want an intercept that can be interpreted, we can centre the explanatory variable, for example by using age minus 50 years as the explanatory variable. The intercept is then the expected blood pressure of a 50-year-old.

Reporting

A figure is very important part of reporting the results of a regression analysis. A good figure can help to understand the relationship between the response and explanatory variable, and can also help to understand the uncertainty in the estimates of the slope and intercept.



In scientific reports we usually don't give a graph a title, but rather make a caption that describes the figure. The caption should be informative and should describe what the figure shows. For example, a caption for the above figure could be:

Figure 1. The relationship of systolic blood pressure (mmHg) against age (years) with a fitted regression line (blue) and 95% confidence band for the regression (grey band).

Then in the text of the Results section we would report the results of the regression analysis. For example:

Systolic blood pressure was positively associated with age (slope = 0.82 mmHg per year, 95% confidence interval [0.77, 0.88], degrees of freedom = 98, t = 29.7, $p < 0.001$). The regression model explained 90% of the variability in systolic blood pressure.

We *DO NOT* report like this:

There was a statistically significant relationship between age and systolic blood pressure ($p < 0.001$).

How big is the effect, and does it matter?

A p-value tells us whether the data are compatible with there being no relationship. It does not tell us how strong the relationship is, or whether it is important. For that we need the **effect size**: here, the estimated slope, with its confidence interval, in the units of the variables.

In the blood pressure example, the slope is 0.82 mmHg per year (95% CI 0.77 to 0.88). It is often helpful to express an effect over a meaningful change in the explanatory variable. Over ten years, for example, the expected increase in systolic blood pressure is 8.2 mmHg (95% CI 7.7 to 8.8 mmHg).

Whether an effect of this size is *biologically* (or medically) important is not a statistical question. It requires knowledge of the system: for example, how large a change in blood pressure has consequences for health. Good reporting gives readers the effect size and its uncertainty, in meaningful units, so that they can make this judgement. Pay attention to the whole confidence interval: if it includes both trivially small and important effects, the data cannot tell us which is the case, even if the p-value is small.

This is the third of the four principles for high-quality data analysis introduced in Chapter 1: emphasise effect sizes and biological relevance, rather than statistical significance alone.

Association is not causation

The blood pressure data come from an *observational* study: we did not assign people their ages. The regression therefore describes an **association**: older people in this sample tend to have higher blood pressure. On its own, it does not show that ageing *causes* the increase. Many other things differ with age, such as diet, exercise and body weight, and any of these could contribute to the association.

A variable that is associated with both the explanatory variable and the response, and so creates or distorts the association between them, is called a **confounding variable**. The strongest evidence for a causal effect comes from *manipulative experiments*, in which the researcher randomly assigns the values of the explanatory variable to the subjects. Random assignment breaks any link between the explanatory variable and potential confounders. In observational studies, we can include potential confounders as additional explanatory variables in a multiple regression (Chapter 6), but only those that we have measured.

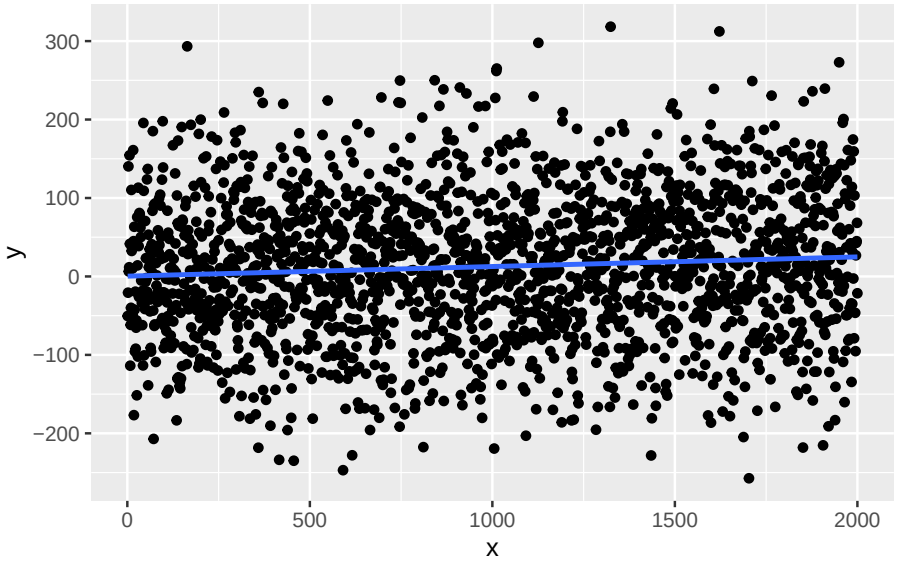
When reporting results from observational data, use language such as "was associated with" rather than "caused", "increased" or "affected".

Relationship between R^2 and the p-value of the slope

Do you think a small R^2 value can be associated with a small p-value for the slope? Or do you think a large R^2 value can be associated with a large p-value for the slope?

Actually, we can get a very small p-value for the slope even when R^2 is very small. This can happen when the sample size is large, because with a large sample size we can get a precise estimate of the slope, which means that the standard error of the slope is small, which means that the t-statistic is large, which means that the p-value is small.

Here is an example of a dataset with a very small R^2 value but a very small p-value for the slope:




```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-278.540  -60.288    0.443   61.720  301.762
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.364568   4.029273   0.090 0.927915
x           0.012271   0.003488   3.518 0.000445 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 90.06 on 1998 degrees of freedom
Multiple R-squared:  0.006156,  Adjusted R-squared:  0.005658
F-statistic: 12.38 on 1 and 1998 DF,  p-value: 0.0004447
```

The R^2 value is about 0.006, which means that the regression model explains less than only 1% of the variability in y. However, the p-value for the slope is very small ($p < 0.001$), which means that there is strong evidence for a relationship between x and y!!!

This is because with a large sample size we can get a precise estimate of the slope, which means that the standard error of the slope is small, which means that the t-statistic is large, which means that the p-value is small.

So, beware of interpreting a small p-value for the slope as evidence of a strong relationship between the response and explanatory variable. A small p-value for the slope can be observed even when the relationship is very weak (i.e., when R^2 is very small).

Review

That is regression done (at least for our current purposes). Here is a summary of what we have covered:

Previous chapter:

- Why use (linear) regression?
- Fitting the line (= parameter estimation)
- Is linear regression good enough model to use?
- What to do when things go wrong?
- Transformation of variables/the response.
- Handling of outliers.

This chapter:

- Sums of squares: SST, SSM, SSE
- R^2 as a measure of goodness of fit
- Hypothesis testing in regression
- Null and alternative hypotheses
- t-statistic and p-values
- Confidence intervals
- Confidence and prediction bands

- Reporting

Further reading

Again, there are many good books and online resources about regression models. The same ones as last week are recommended, also for interpreting results of regression models.

- Faraway, J. J. (2016). Linear models with R (2nd ed.). Chapman and Hall/CRC. Link.
- Chapter 7 of The New Statistics with R, by Andy Hector.

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Calculation of confidence bands

Given a fixed value of x, say x_0. The question is:

Where does $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$ lie with a certain confidence (i.e., 95%)?

The first step is to calculate the standard error of $\hat{y}_0$:

$$\sigma^{(\hat{y}_0)} = \hat{\sigma}\sqrt{\left(\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)}$$

where $\hat{\sigma}^2$ is the expected residual variance of the model, calculated as:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-2}$$

Then, we can calculate the confidence interval of $\hat{y}_0$ as:

$$\hat{y}_0 \pm t_{0.975} \cdot \sigma^{(\hat{y}_0)}$$

Plotting the confidence interval around all $\hat{y}_0$ values one obtains the *confidence band* or *confidence band for the expected values* of y.

Calculation of the prediction band

The standard error of a future observation is given by:

$$\sigma^{(y_0)} = \hat{\sigma}\sqrt{1 + \left(\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)}$$

This is the same as the standard error of the predicted value $\hat{y}_0$ but with an additional 1. This is because the standard error of a future observation is the standard error of the predicted value $\hat{y}_0$, and this includes the uncertainty in the parameter estimates, plus an additional term that accounts for the variability of future observations around the predicted value. This additional term is what makes the prediction band wider than the confidence band.

The prediction interval of a future observation is then given by:

$$\hat{y}_0 \pm t_{0.975} \cdot \sigma^{(y_0)}$$

This is the reason why the **prediction band** is wider than the confidence band.

Randomisation test for the slope of a regression line

Let's use randomisation as another method to understand how likely we are to observe the data we have, given the null hypothesis is true.

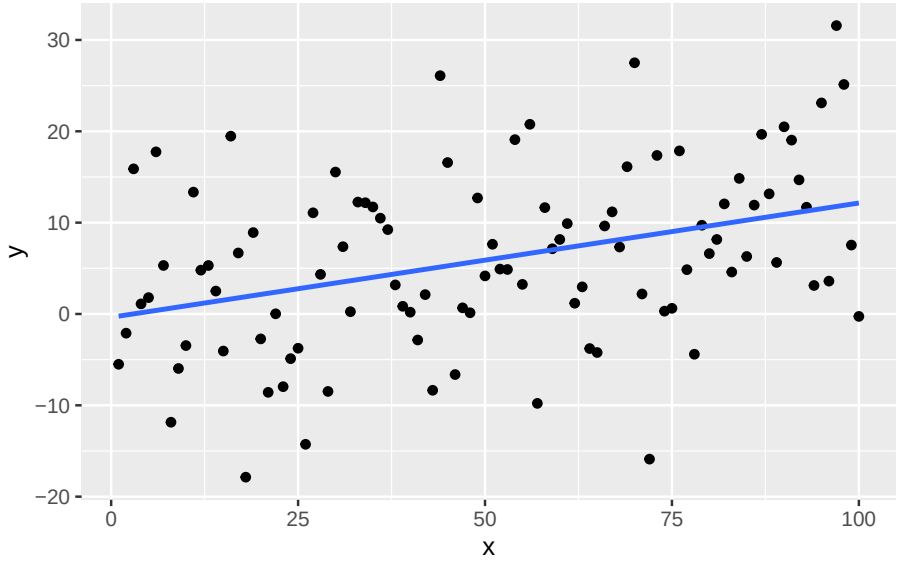
If the null hypothesis is true, we expect no relationship between x and y. Therefore, we can shuffle the y values and fit a regression model to the shuffled data. We can repeat this many times and calculate the slope of the regression line each time. This will give us a distribution of slopes we would expect to observe if the null hypothesis is true.

First, we'll make some data and get the slope of the regression line. Here is the observed slope and relationship:

```
set.seed(123)
n <- 100
x <- 1:n
y <- 0.1*x + rnorm(n, 0, 10)
m_sim <- lm(y ~ x)
coef(m_sim)[2]
```

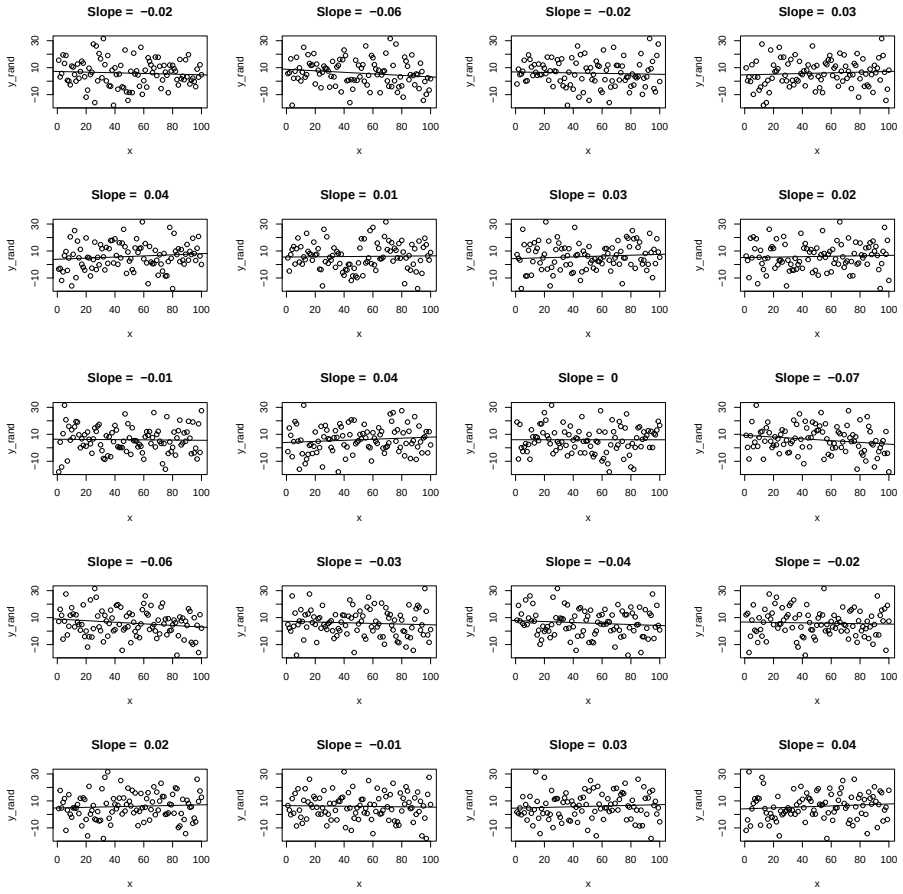
```
       x
0.1251108
```

```
ggplot(data.frame(x = x, y = y), aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, formula = y ~ x)
```

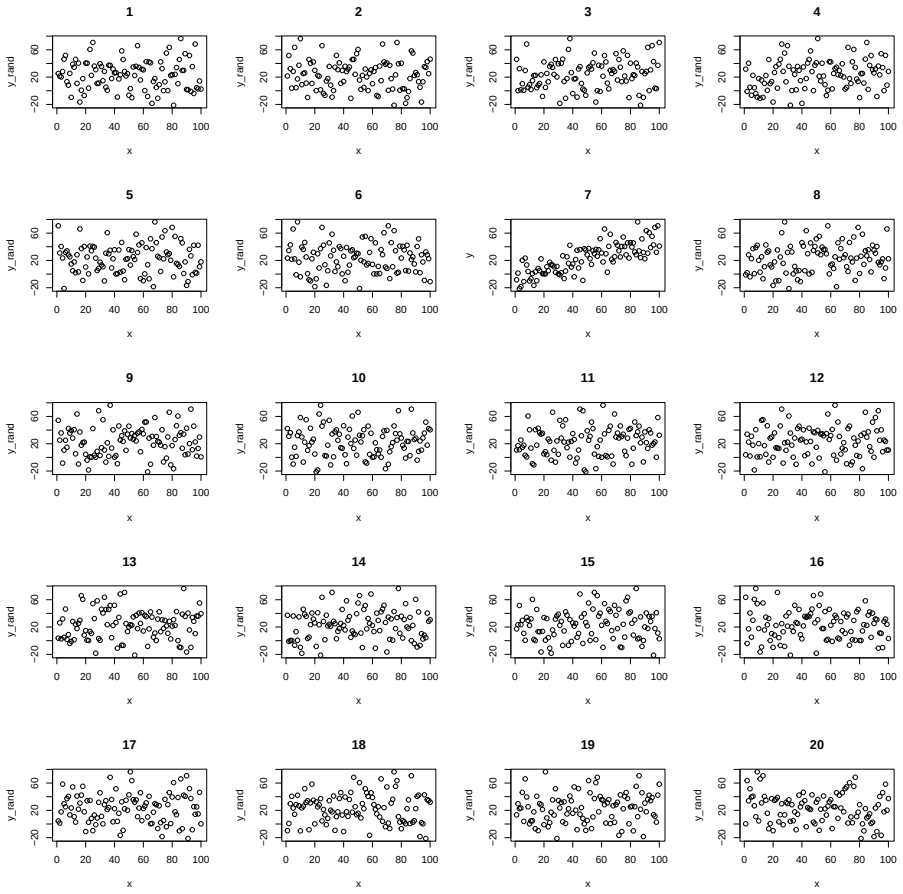


Now we'll use randomisation to test the null hypothesis. We can create lots of examples where the relationship is expected to have a slope of zero by shuffling randomly the y values. Here are 20:

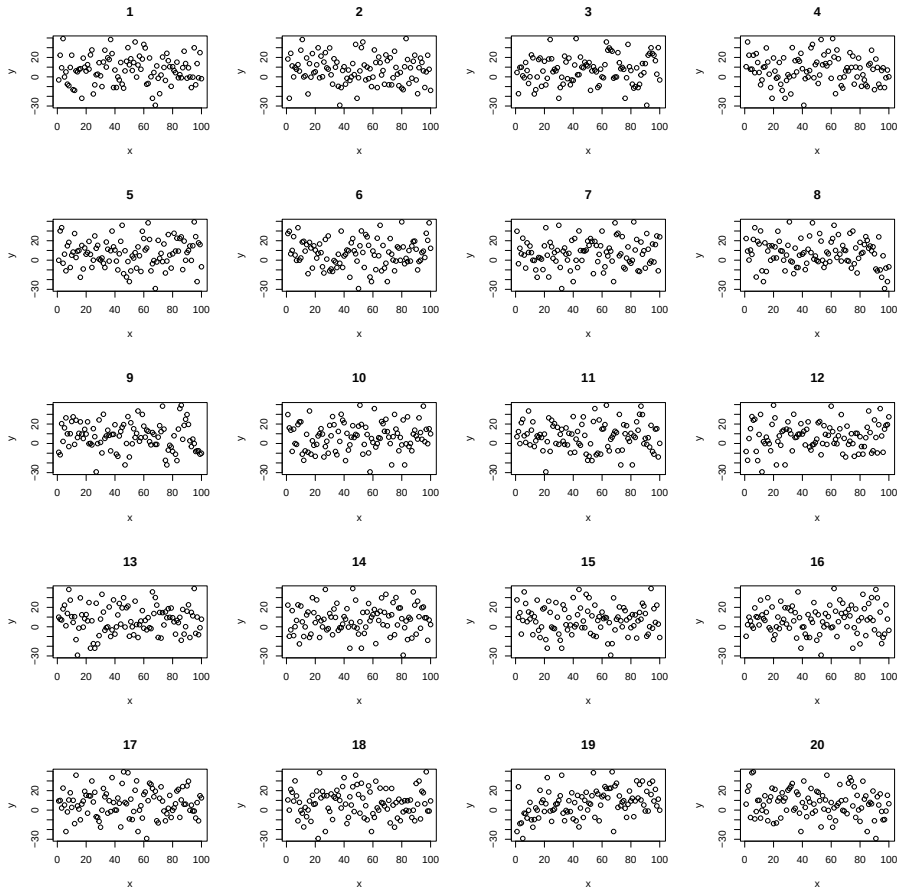
```
par(mfrow = c(5, 4))
for (i in 1:20) {
  y_rand <- sample(y)
  m_rand <- lm(y_rand ~ x)
  plot(x, y_rand, main = paste("Slope = ", round(coef(m_rand)[2], 2)))
  abline(m_rand)
}
```



Now let's create 19 and put the real one in there somewhere random. Here's a case where the real data has a quite strong relationship:



We can confidently find the real data amount the shuffled data. But what if the relationship is weaker?



Now its less clear which is the real data. We can use this idea to test the null hypothesis.

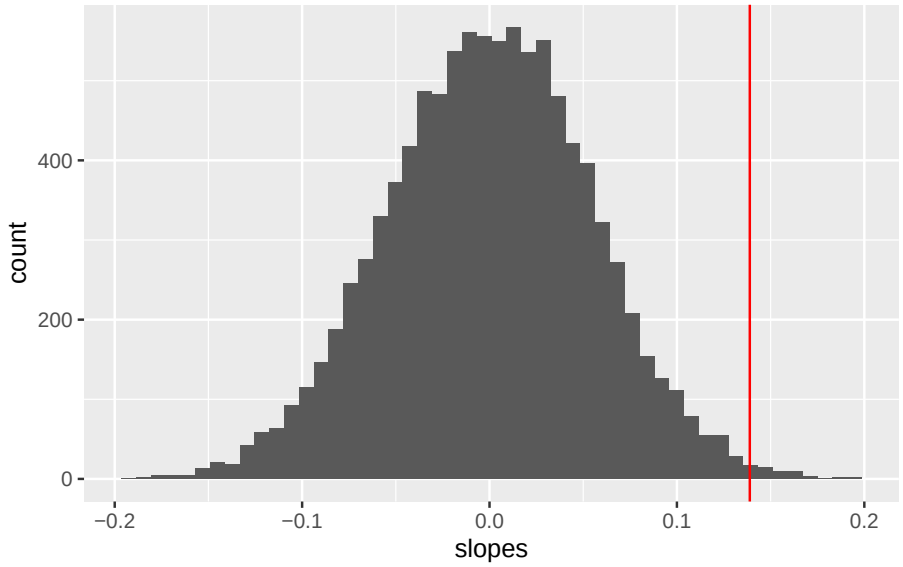
We do the same procedure of but instead of just looking at the graphs, we calculate the slope of the regression line each time. This gives us a distribution of slopes we would expect to observe if the null hypothesis is true. We can then see where the observed slope lies in this distribution of null hypothesis slopes.

```
# The "observed" data: a weak relationship
set.seed(456)
y_weak <- 0.15*x + rnorm(n, 0, 15)
m_weak <- lm(y_weak ~ x)
observed_slope <- coef(m_weak)[2]
observed_slope
```

```
       x
0.1388931
```

```
# Repeat the randomisation 10000 times: shuffle y_weak, fit the model, keep the slope
rand_slopes <- replicate(10000, {
  y_rand <- sample(y_weak)
  m_rand <- lm(y_rand ~ x)
  coef(m_rand)[2]
})

ggplot(data.frame(slopes = rand_slopes), aes(x = slopes)) +
  geom_histogram(bins = 50) +
  geom_vline(xintercept = observed_slope, color = "red")
```



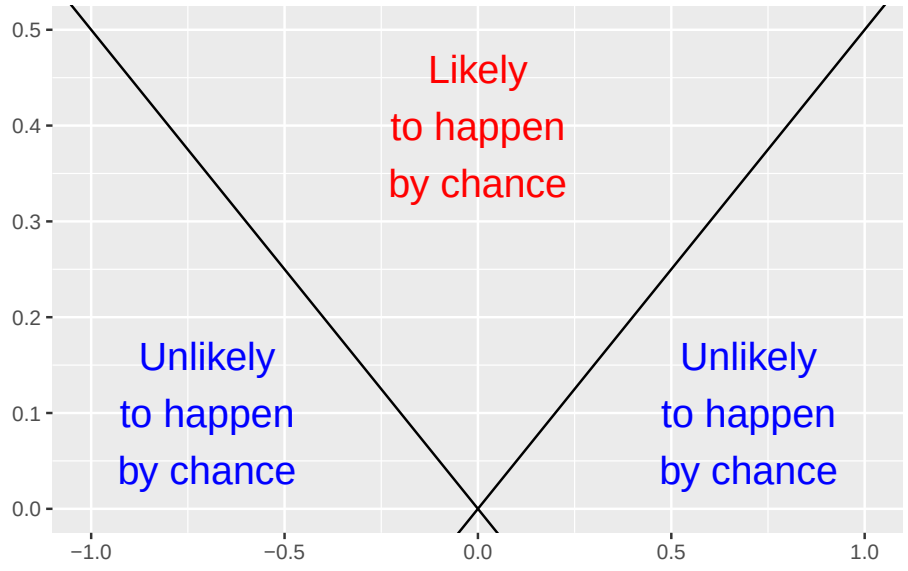
The red line is the slope of the observed data; the histogram shows the slopes we get when the null hypothesis is true (no relationship). We can now calculate the probability of observing a slope at least as extreme as the one we observed (in either direction), given the null hypothesis is true. This is the p-value of the randomisation test:

```
p_value <- mean(abs(rand_slopes) >= abs(observed_slope))
p_value
```

```
[1] 0.0103
```

Compare this with the p-value for the slope from the t-test in `summary(m_weak)`, which is 0.0092. The two methods give similar answers.

Visualising p-values for regression slopes



Analysis of variance (L5)

Learning objectives

After this unit, students will be able to:

1. Explain one-way ANOVA as a linear model with a single categorical explanatory variable.
2. Explain how the total variation is partitioned into between-group and within-group components, and how the F statistic is calculated from them.
3. Calculate the total, model and residual degrees of freedom of a one-way ANOVA ($n - 1$, $k - 1$ and $n - k$), and explain why they add up.
4. Explain why sums of squares are divided by their degrees of freedom to give mean squares, and identify the two degrees of freedom of the F-distribution.
5. Calculate missing entries in an ANOVA table (degrees of freedom, sums of squares, mean squares, F).
6. Use R to fit a one-way ANOVA with `lm()` and obtain the ANOVA table.
7. Use R to test whether the group means differ, and draw the appropriate conclusion.
8. Interpret the coefficients of a model with a categorical explanatory variable: the reference level and the differences from it.
9. Use R to obtain group means and the estimated differences between groups, with their uncertainty.
10. Use diagnostic plots to judge whether the assumptions of a one-way ANOVA are reasonably met.
11. Explain why testing many pairs of groups inflates the probability of a false positive, and use a correction for multiple comparisons (e.g. Bonferroni, Tukey HSD).
12. Decide whether one-way ANOVA is appropriate for a described study design and response variable.
13. Recognise a correct and complete statement of one-way ANOVA

results.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch05_anova.R

This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 5 practical uses but this chapter does not.

To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch05_anova.R")
```

Video walkthrough: coming soon.

The previous two chapters were about linear regression. *Linear regression* is a type of *linear model* – recall that in *R* we used the function `lm()` to make the regression model. In this chapter we will look at a different type of linear model: analysis of variance (ANOVA).

Introduction

Recall that linear regression is a linear model with one continuous explanatory (independent) variable. A continuous explanatory variable is a variable in which values can take any value within a range (e.g., height, weight, temperature).

In contrast, analysis of variance (ANOVA) is a linear model with one or more categorical explanatory variables. We will first look at a one-way ANOVA, which has one categorical explanatory variable. Later (in a following chapter) we will look at two-way ANOVA, which has two categorical explanatory variables.

What is a categorical variable? A categorical explanatory variable is a variable that contains values that fall into distinct groups or categories. For example, habitat type (e.g., forest, grassland, wetland), treatment group (e.g., control, low dose, high dose), or diet type (e.g., vegetarian, vegan, omnivore).

This means that each observation belongs to one of a limited number of categories or groups. For example, in a study of how blood pressure varies with diet type,

diet type is a categorical variable with several levels (e.g., vegetarian, vegan, omnivore). A person can only belong to one diet type category.

Here are the first several rows of a dataset that contains blood pressure measurements for individuals following different diet types:

Reading in the dataset:

```
bp_data_diet <- read_csv("datasets/bp_data_diet.csv")
```

```
Rows: 50 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): diet, person_ID
dbl (1): bp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
bp_data_diet <- select(bp_data_diet, bp, diet, person_ID)
head(bp_data_diet)
```

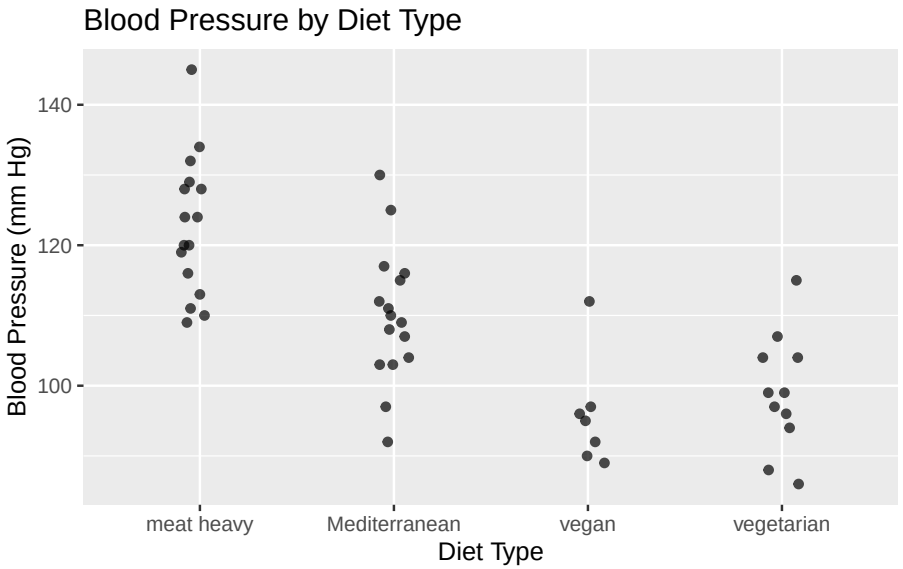
```
# A tibble: 6 x 3
     bp diet          person_ID
  <dbl> <chr>         <chr>
1   120 meat heavy    person_1
2    89 vegan         person_2
3    86 vegetarian    person_3
4   116 meat heavy    person_4
5   115 Mediterranean person_5
6   134 meat heavy    person_6
```

There are three variables: - `bp`: blood pressure (continuous response variable) - `diet`: diet type (categorical explanatory variable) - `person_ID`: unique identifier for each individual (not used in the analysis)

Note that the `diet` variable is of type `<chr>` which is short for `character`. In *R*, categorical variables are often represented as factors.

As usual, its a really good idea to visualise the data in as close to "raw" form as possible before doing any analysis. We'll make a scatterplot of blood pressure versus diet type.

```
ggplot(bp_data_diet, aes(x = diet, y = bp)) +
  geom_jitter(width = 0.1, height = 0, alpha = 0.7) +
  labs(title = "Blood Pressure by Diet Type",
       x = "Diet Type",
       y = "Blood Pressure (mm Hg)")
```



> **Tip**
>
> We just used `geom_jitter()` instead of `geom_point()` to make a scatterplot. This is because `geom_jitter()` adds a small amount of random noise to the points, which helps to prevent overplotting when multiple points have the same value (which is common when the x-axis is categorical). When we use `geom_jitter()`, we can specify the amount of noise to add in the x and y directions using the `width` and `height` arguments, respectively. We must be very careful to not add noise to the y direction if we care about the actual y values (e.g., blood pressure). In this case, we only added noise in the x direction by setting `height = 0` to separate the points just enough, but not so much that we could get confused about which of the diets they belong to.

Looking at this graph it certainly looks like diet type has an effect on blood pressure. But is this effect statistically significant? In other words, are the differences in mean blood pressure between diet types larger than we would expect due to random variation alone?

Analysis of variance (ANOVA) is a statistical method that can help us answer this question, and also others.

How does it look like in R?

We can fit a one-way ANOVA model in *R* using the same `lm()` function that we used for linear regression. The only difference is that the explanatory variable is

categorical.

```
m_bp_diet <- lm(bp ~ diet, data = bp_data_diet)
```

Then instead of using `summary()` to look at the results, we use the `anova()` function.

```
anova(m_bp_diet)
```

```
Analysis of Variance Table

Response: bp
          Df Sum Sq Mean Sq F value    Pr(>F)
diet       3 5274.2 1758.08  20.728 1.214e-08 ***
Residuals 46 3901.5   84.82
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This is an ANOVA table. It shows us the sources of variation in the data, along with their associated degrees of freedom (Df), sum of squares (Sum Sq), mean square (Mean Sq), F value, and p-value (Pr(>F)) associate with a getting a F value the same as or greater than the observed F value if the null hypothesis were true.

The challenge now is to understand what all of these values mean! Let's take it step by step.

What is ANOVA?

Analysis of variance is a method to compare whether the observations (e.g., of blood pressure) differ according to some grouping (e.g., diet) that the subjects (e.g., people) belong to.

We already know a lot about analysing variance: we compared the total sum of squares (SST), model sum of squares (SSM) and the residual sum of squares (SSE) in the context of linear regression. We used these to calculated the R^2 value. The R^2 value tells us how much of the total variance in the response variable (e.g., blood pressure) is explained by the explanatory variable (e.g., diet).

The same applies to analysis of variance (ANOVA) (as well as regression) because ANOVA is a special case of a linear model, just like regression is also a special case of a linear model.

The defining characteristic of ANOVA is that we are comparing the means of groups by analysing variances. Put another way, we will have a single categorical explanatory variable with two or more levels. We will test whether the means of the response variable are the same across all levels of the explanatory variable, and we test this by analysing the variances.

When we have only one categorical explanatory variable, we use a *one-way* ANOVA. When we have two categorical explanatory variables, we will use a *two-way* ANOVA (we'll look at this in a subsequent chapter).

ANOVA as a linear model

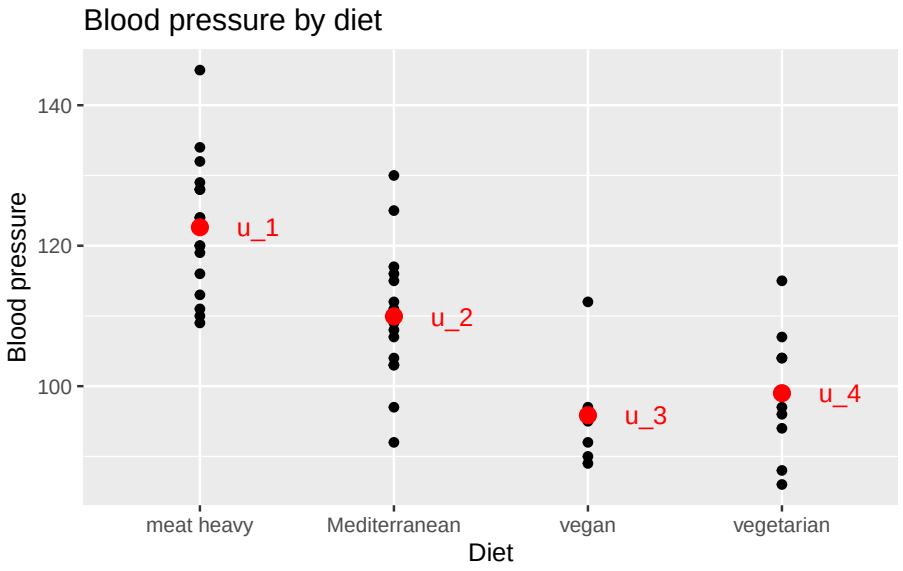
Just like linear regression, ANOVA can be expressed as a linear model. The key difference is that in ANOVA, the explanatory variable is categorical rather than continuous.We formulate the linear model as follows:

$$y_{ij} = \mu_j + \epsilon_i$$

where:

- y_{ij} = Blood pressure of individual i with diet j
- μ_j = Mean blood pressure of an individual with diet j
- $\epsilon_i \sim N(0, \sigma^2)$ is an independent error term.

Graphically, with the blood pressure and diet data, this looks like:



> **Note**
>
> There is lots of hidden code used to create the data used in the graph above, and to make the graph itself. You can see the code by going to the Github repository for this book.

Rewrite the model

We usually use a different formulation of the linear model for ANOVA. This is because we usually prefer to express the estimated parameters in terms of *differences between means* (rather than the means themselves). The reason for this is that then the null hypothesis can be that the differences are zero.

To proceed with this formulation, we define one of the groups as the reference group, and make the mean of that equal to the intercept of the model. For example, if we choose the "meat heavy" diet as the reference group, we can write:

$$\mu_{meat} = \beta_0$$

And then to express the other group means as deviations from the reference group mean:

$$\mu_{Med} = \beta_0 + \beta_1$$

$$\mu_{vegan} = \beta_0 + \beta_2$$

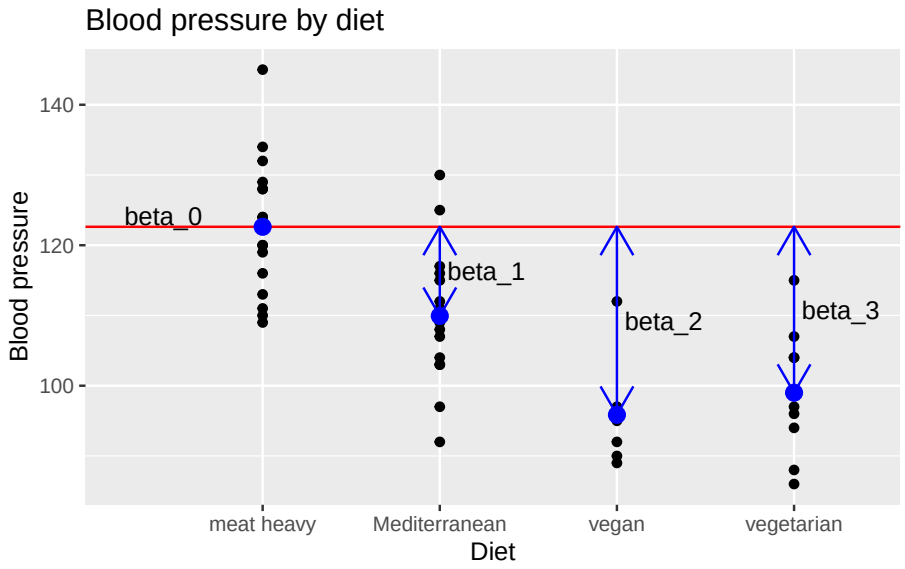
$$\mu_{veggi} = \beta_0 + \beta_3$$

When we write out the entire model, we get:

$$y_i = \beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

where: y_i is the blood pressure of individual i. x_i^1 is a binary variable indicating whether individual i is on the Mediterranean diet. x_i^2 is a binary variable indicating whether individual i is on the vegan diet. x_i^3 is a binary variable indicating whether individual i is on the vegetarian diet.

Graphically, the model now looks like this:



Here is something to warp you mind… we described one-way ANOVA as a linear model with one categorical explanatory variable. But as you can see above, we can also describe it as a linear model with multiple binary explanatory variables (one for each group except the reference group). And when we make a linear model in R it really does create multiple binary explanatory variables behind the scenes. So one-way ANOVA and multiple linear regression with multiple binary explanatory variables are really the same thing! And, even more mind-warping, one-way ANOVA and multiple regression (regression with multiple continuous explanatory variables) are also the same thing! So when we look at multiple regression later in the course, you can think of it as just an extension of one-way ANOVA.

The ANOVA test: The F-test

Aim of ANOVA: to test *globally* if the groups differ. That is we want to test the null hypothesis that all of the group means are equal:

$$H_0 : \mu_1 = \mu_2 = ... = \mu_g$$

This is equivalent to testing if all βs that belong to a categorical variable are $= 0$.

$$H_0 : \beta_1 = ... = \beta_{g-1} = 0$$

The alternate hypothesis is that H_1: The group means are not all the same.

A key point is that we are testing a null hypothesis that concerns all the groups. We are not testing if one group is different from another group (which we could do with a t-test on one of the non-intercept βs).

Because we are testing a null hypothesis that concerns all the groups, we need to use an F-test. It asks if the model with the group means is better than a model with just the overall mean.

> **Note**
>
> The F-test is called the "F-test" because it is based on the F-distribution, which was named after the statistician Sir Ronald A. Fisher. Fisher developed this statistical method as part of his pioneering work in analysis of variance (ANOVA) and other fields of experimental design and statistical inference.

Actually, the F-test does not directly test the null hypothesis that all the group means are equal. Instead, it tests whether the model that includes the *group means* explains significantly more variance in the data than a model that only includes the overall mean (i.e., without considering group differences).

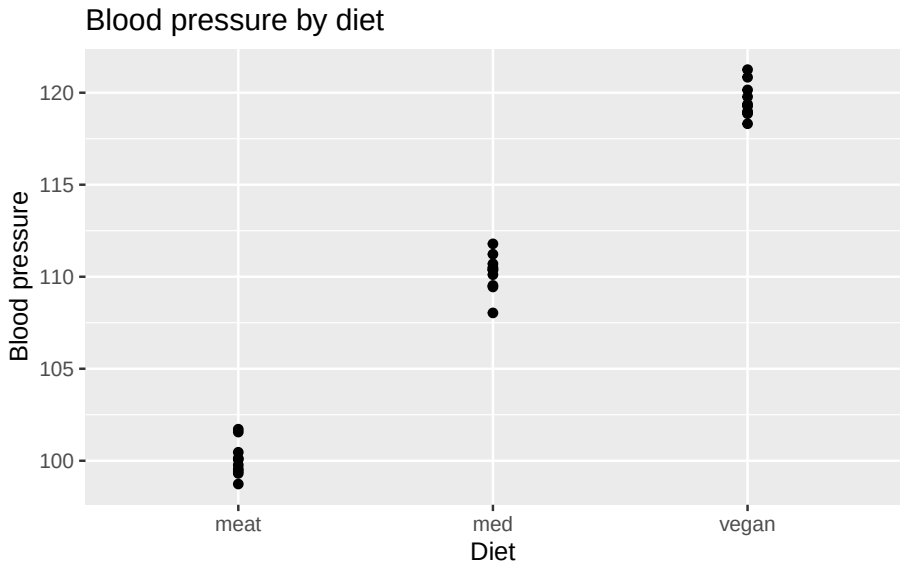
The F-test does this by comparing two variance estimates: the variance explained by the group means (between-group variance) and the variance that remains unexplained within each group (within-group variance).

Interpretation of the F statistic

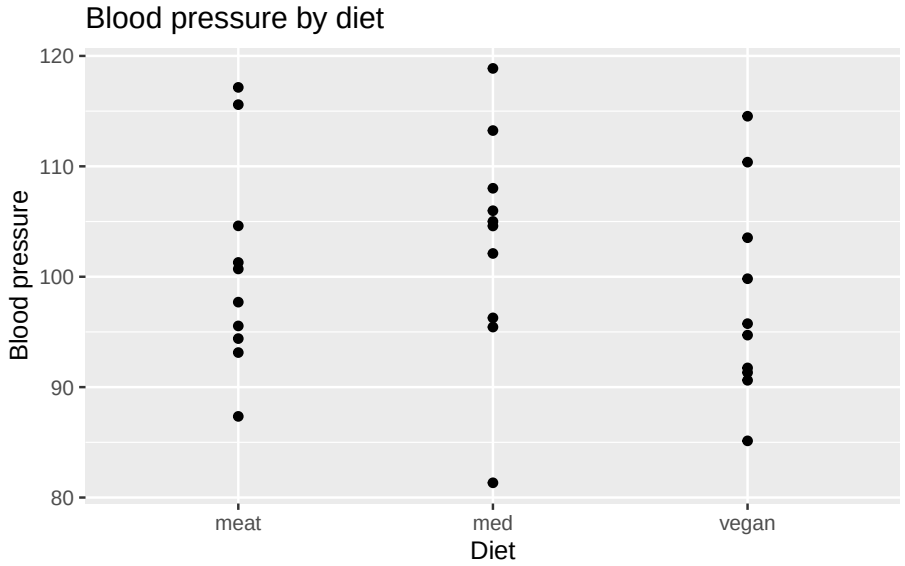
The F-test involves calculating from the observed data the value of the F statistic, and then computing if that value is large enough to reject the null hypothesis.

The F statistic is a ratio of two variances: the variance **between** groups, and the variance **within** groups.

Here is an example with very low within group variability, and high between group variability:



And here's an example with very high within group variability, and low between group variability:



So, when the ratio of between group variance to within group variance is large, the group means are very different compared to the variability within groups. This suggests that the groups are different.

When the ratio is small, the group means are similar compared to the variability

within groups. This suggests that the groups are not different.

- F **increases**
  - when the group means become more different, or
  - when the variability within groups decreases.
- F **decreases**
  - when the group means become more similar, or
  - when the variability within groups increases.

$\rightarrow$ The larger F, the less likely are the data seen under H_0.

Calculating the F statistic

Recall that the F statistic is a ratio of two variances. Specifically, it is the ratio of two mean squares (MS):

- MS_{model}: the variability **between** groups.
- $MS_{residual}$: the variability **within** groups.

MS stands for Mean Square, and is a variance estimate.

The F statistic is calculated as:

$$F = \frac{MS_{model}}{MS_{residual}}$$

To find the mean squares, we need to calculate the within and the between group sums of squares, and the corresponding degrees of freedom. Let's go though this step by step.

Calculating the sums of squares

First we get the total sum of squares (SST), which quantifies the total variability in the data. This is then split into two parts:

1. The explained variability, which is known as the between group sum of squares ($SS_{between}$) or sum of squares of the model (SSM).
2. The residual variability, which is known as the within group sum of squares (SS_{within}) or sum of squares of error / residuals (SSE).

Total variability: $\text{SST} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (y_{ij} - \overline{y})^2$

where:

- y_{ij} is the blood pressure of individual j in group i
- $\overline{y}$ is the overall mean blood pressure
- n_i is the number of individuals in group i
- k is the number of groups

Explained variability (between group variability): $= SS_{between}$\$ = SSM $= \sum_{i=1}^{k} n_i(\overline{y}_i - \overline{y})^2$

where:

- $\overline{y}_i$ is the mean blood pressure of group i

Residual variability (within group variability): $= SS_{within}$ = SSE $= \sum_{i=1}^{k} \sum_{j=1}^{n_i} (y_{ij} - \overline{y}_i)^2$

Calculating the degrees of freedom

And now we need the degrees of freedom for each sum of squares:

SST degrees of freedom: $n - 1$ (total degrees of freedom is number of observations n minus 1)

SSM degrees of freedom: $k - 1$ (model degrees of freedom is number of groups k minus 1)

SSE degrees of freedom: $n - k$ (residual degrees of freedom is total degrees of freedom $n - 1$ minus model degrees of freedom $k - 1$)

Total degrees of freedom

The total degrees of freedom are the degrees of freedom associated with the total sum of squares (SST).

In order to calculate the SST, we need to calculate the mean of the response variable. This implies that we estimate one parameter (the mean of the response variable). As a consequence, we lose one degree of freedom and so there remain $n - 1$ degrees of freedom associated with the total sum of squares (where n is the number of observations).

What do we mean by "lose one degree of freedom"? Imagine we have ten observations. We can calculate the mean of these ten observations. But if we know the mean and nine of the observations, we can calculate the tenth observation. So, in a sense, once we calculate the mean, the value of one of the ten observations is fixed. This is what we mean by "losing one degree of freedom". When we calculate and use the mean, one of the observations "loses its freedom".

For example, take the numbers 1, 3, 5, 7, 9. The mean is 5. The sum of the squared differences between the observations and the mean is $(1-5)^2 + (3-5)^2 + (5-5)^2 + (7-5)^2 + (9-5)^2 = 20$. This is the total sum of squares. The degrees of freedom are $5 - 1 = 4$.

The total degrees of freedom are the total number of observations minus one. That is, the total sum of squares is associated with $n - 1$ degrees of freedom.

Another perspective in which to think about the total sum of squares and total degrees of freedom is to consider the intercept only model. The intercept only model is a model that only includes the intercept term. The equation of this model would be:

$$y_i = \beta_0 + \epsilon_i$$

The sum of the square of the residuals for this model is minimised when the predicted value of the response variable is the mean of the response variable. That is, the least squares estimate of β_0 is the mean of the response variable:

$$\hat{\beta}_0 = \bar{y}$$

Hence, the predicted value of the response variable is the mean of the response variable. The equation is:

$$\hat{y}_i = \bar{y} + \epsilon_i$$

The error term is therefore:

$$\epsilon_i = y_i - \bar{y}$$

And the total sum of squares is:

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where $\hat{y}_i$ is the predicted value of the response variable for the ith observation, $\bar{y}$ is the mean of the response variable, and ϵ_i is the residual for the ith observation.

The intercept only model involves estimating only one parameter, so the total degrees of freedom are the total number of observations minus one $n - 1$.

Therefore, the total degrees of freedom are the total number of observations minus one.

Bottom line: SST is the residual sum of squares when we fit the intercept only model. The total degrees of freedom are the total number of observations minus one.

Model degrees of freedom

The model degrees of freedom are the degrees of freedom associated with the model sum of squares (SSM).

In the case of the intercept only model, we estimated one parameter, the mean of the response variable.

In the case of a categorical variable with k groups, we need $k-1$ parameters (non intercept β parameters), so we lose $k-1$ degrees of freedom. Put another way, when we fit a model with a categorical explanatory variable with k groups, we estimate $k-1$ parameters in addition to the intercept. That is, we estimate the difference between each group and the reference group.

Each time we estimate a new parameter, we lose a degree of freedom.

Residual degrees of freedom

The residual degrees of freedom are the total degrees of freedom $(n-1)$ minus the model degrees of freedom $(k-1)$.

Therefore, the residual degrees of freedom are the degrees of freedom remaining after we estimate the intercept and the other β parameters. There is one intercept and $k-1$ other β parameters, so the residual degrees of freedom are $n-1-(k-1)=n-k$.

Calculating the mean square and F statistic

From these sums of squares and degrees of freedom we can calculate the mean squares and F-statistic:

$$MS_{model} = \frac{SS_{\text{between}}}{k-1} = \frac{SSM}{k-1}$$

$$MS_{residual} = \frac{SS_{\text{within}}}{n-k} = \frac{SSE}{n-k}$$

$$F = \frac{MS_{model}}{MS_{residual}}$$

> **Note**
>
> **Why divide by the degrees of freedom?** The more observations we have, the greater will be the total sum of squares. The more observations we have, the greater will be the residual sum of squares. So it is not very informative to compare totals. Rather, we need to compare the mean of the sums of squares. Except we don't calculate the mean by dividing by the number of observations. Rather we divide by the degrees of freedom. The total mean square is an estimate of the variance of the response variable. And the residual mean square is an estimate of the variance of the residuals.

SST, SSM, SSE, and degrees of freedom

Just a reminder and a summary of some of the material above:

- SST: degrees of freedom $= n - 1$
- SSM: degrees of freedom $= k - 1$
- SSE: degrees of freedom $= n - k$

The sum of squares add up:

$$SST = SSM + SSE$$

and the degrees of freedom add up

$$(n - 1) = (k - 1) + (n - k)$$

Source of variance table

Now we have nearly everything we need. We often express all of this (and a few more quantities) in a convenient table called the **sources of variance table** (or ANOVA table).

The **sources of variance table** is a table that conveniently and clearly gives all of the quantities mentioned above. It breaks down the total sum of squares into the sum of squares explained by the model and the sum of squares due to error. The source of variance table is used to calculate the F-statistic.

Table 1: Sources of variance table

Source	Sum of squares	Degrees of freedom	Mean square	F-statistic
Model	SSM	$k-1$	$MSE_{model} = SSM/k-1$	$\frac{MSE_{model}}{MSE_{error}}$
Error	SSE	$n-1-(k-1)$	$MSE_{error} = SSE/(n-1-(k-1))$	
Total	SST	$n-1$		

Is my F-statistic large or small?

OK, so we have calculated the F statistic. But how do we use it to test our hypothesis?

We can use the F statistic to calculate a p-value, which tells us how likely our data is under the null hypothesis.

Some key points:

1. F-Distribution: The test statistic of the F-test (that is, the F-statistic) follows the F-distribution under the null hypothesis. This distribution arises when comparing the ratio of two independent sample variances (or mean squares).
2. Ronald Fisher's Contribution: Fisher introduced the F-distribution in the early 20th century as a way to test hypotheses about the equality of variances and to analyze variance in regression and experimental designs. The "F" in F-distribution honours him.
3. Variance Ratio: The test statistic for the F-test is the ratio of two variances (termed mean squares in this case), making the F-distribution the natural choice for modeling this ratio when the null hypothesis is true.

The F-test is widely used, including when comparing variances, assessing the significance of multiple regression models (see later chapter), conducting ANOVA to test for differences among group means, and for comparing different models.

Recall that "The F-statistic is calculated as the ratio of the mean square error of the model to the mean square error of the residuals." And that a large F-statistic is evidence against the null hypothesis that the slopes of the explanatory variables are zero. And that a small F-statistic is evidence to not reject the null hypothesis that the slopes of the explanatory variables are zero.

But how big does the F-statistic need to be in order to confidently reject the null hypothesis?

The null hypothesis that the explained variance of the model is no greater than would be expected by chance. Here, "by chance" means that the slopes of the explanatory variables are zero.

$$H_0 : \beta_1 = \beta_2 = ... = \beta_p = 0$$

The alternative hypothesis is that the explained variance of the model is greater than would be expected by chance. This would occur if the slopes of some or all of the explanatory variables are not zero.

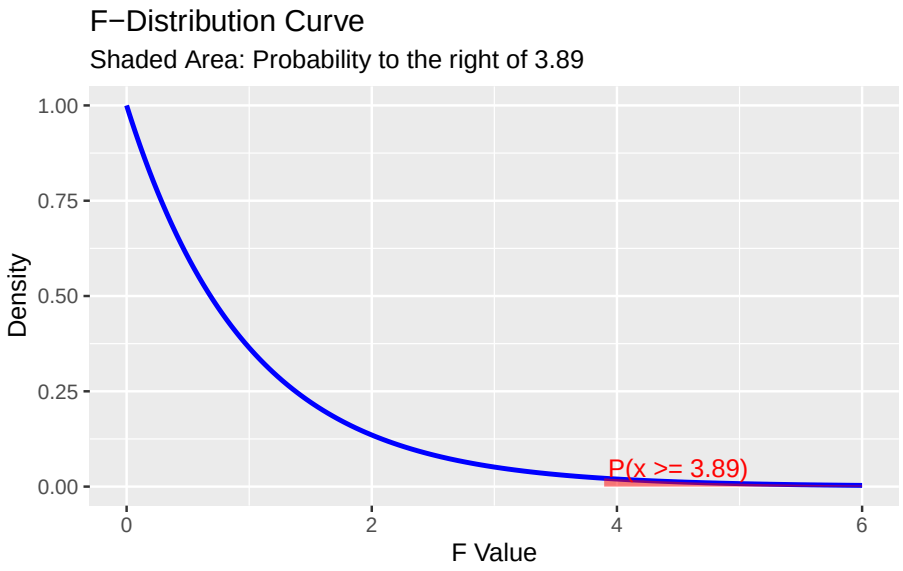
$$H_1 : \beta_1 \neq 0 \text{ or } \beta_2 \neq 0 \text{ or } ... \text{ or } \beta_p \neq 0$$

To test this hypothesis we are going to, as usual, calculate a p-value. The p-value is the probability of observing a test statistic as or more extreme as the one we observed, assuming the null hypothesis is true. To do this, we need to know the distribution of the test statistic under the null hypothesis. The distribution of the test statistic under the null hypothesis is known as the F-distribution.

The F-distribution has two degrees of freedom values associated with it: the degrees of freedom of the model and the degrees of freedom of the residuals. The degrees of freedom of the model are the number of extra parameters that the model estimates compared with the null (intercept-only) model; for a categorical

explanatory variable with k groups this is $k-1$. The degrees of freedom of the residuals are the total degrees of freedom minus the degrees of freedom of the model.

Here is the F-distribution with 2 and 99 degrees of freedom:



The F-distribution is skewed to the right and has a long tail. The area to the right of 3.89 is shaded in red. This area represents the probability of observing an F-statistic as or more extreme as 3.89, assuming the null hypothesis is true. This probability is the p-value of the hypothesis test.

The F-statistic and F-test uses the fact that:

$$\frac{MSE_{model}}{MSE_{residual}} = \frac{SSM/(k-1)}{SSE/(n-k))} \sim F_{k-1,n-k)}$$

follows an F-distribution with $k-1$ and $n-k$ degrees of freedom, where k is the number of groups in the categorical variable, and n the number of data points.

- $SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is the residual sum of squares
- $SSM = SST - SSE$ is the sum of squares of the model
- $SST = \sum_{i=1}^{n}(y_i - \overline{y})^2$ is the total sum of squares
- n is the number of data points
- k is the number of groups in the categorical variable

Well, that is ANOVA conceptually. But how does it actually look like in R?

Doing ANOVA in R

Let's go back again the question of how diet effects blood pressure. Here is the data:

```
bp_data_diet <- select(bp_data_diet, bp, diet, person_ID)
head(bp_data_diet)
```

```
# A tibble: 6 x 3
     bp diet          person_ID
  <dbl> <chr>         <chr>
1   120 meat heavy    person_1
2    89 vegan         person_2
3    86 vegetarian    person_3
4   116 meat heavy    person_4
5   115 Mediterranean person_5
6   134 meat heavy    person_6
```

```
ggplot(bp_data_diet, aes(x = diet, y = bp)) +
  geom_point() +
  labs(title = "Blood pressure by diet",
       x = "Diet",
       y = "Blood pressure")
```



And here is how we fit a linear model to this data:

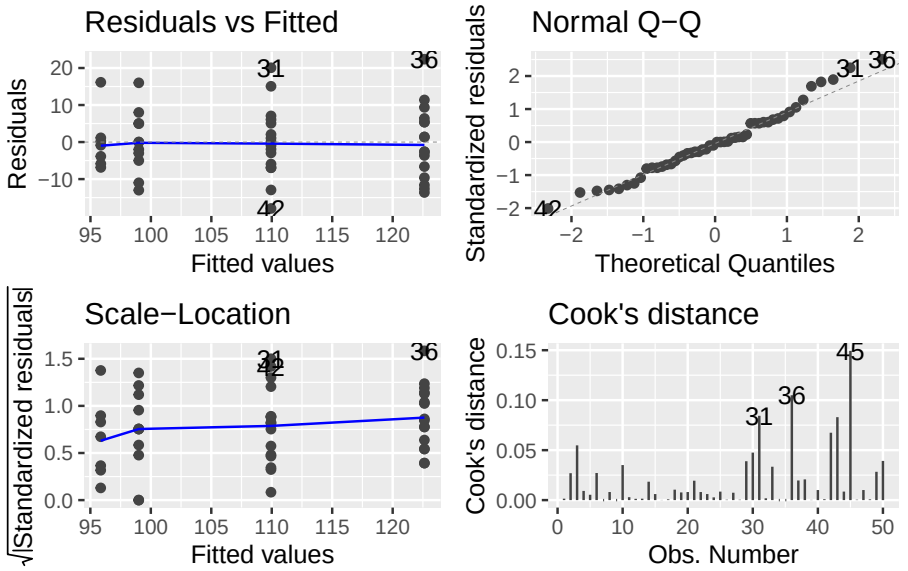
```
m_bp_diet <- lm(bp ~ diet, data = bp_data_diet)
```

IMPORTANT: Since ANOVA is a linear model, it is important to check the

assumptions of linear models before interpreting the results. These are some of the same assumptions we checked for simple linear regression, including: independence of errors, normality of residuals, and homoscedasticity (constant variance of residuals).

As with linear regression, we check the assumptions are not too badly broken by looking at model checking plots:

```
autoplot(m_bp_diet, which = 1:4, add.smooth = FALSE)
```



The Q-Q plot (top right) looks pretty good, indicating that the residuals are approximately normally distributed.

The residuals vs fitted plot (top left) does not show any clear pattern, which is good (though since there is no continuous explanatory variable, we would not expect to see a pattern in this plot).

The scale-location plot (bottom left) does not show any clear pattern, which is good. This suggests that the residual variance is not different in different groups, i.e., that there is little if any heteroscedasticity.

The bottom right plot shows Cook's distance for each observation, that is, how much the fitted values would change if that observation were left out. No observation has a Cook's distance near 1 (the largest is about 0.15), so no single observation has a large influence on the model. We did not ask for the residuals vs leverage plot (`which = 5`). In an ANOVA model with only a categorical explanatory variable, the leverage of a data point depends only on how many observations are in its group ($1/n_i$), so all data points in the same group have the same leverage, and this plot is not very informative.

There are a few residuals that might be a bit extreme. These might be worth investigating, but we will ignore them for now.

Now we can look at the ANOVA table:

```
anova(m_bp_diet)
```

```
Analysis of Variance Table

Response: bp
          Df Sum Sq Mean Sq F value    Pr(>F)
diet       3 5274.2 1758.08  20.728 1.214e-08 ***
Residuals 46 3901.5   84.82
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The ANOVA table shows the sum of squares, degrees of freedom, mean square, F value, and p-value for the model and residuals. As we know, the F value (F statistics) is calculated as the mean square of the model divided by the mean square of the residuals. The p-value is calculated based on the F-distribution with the appropriate degrees of freedom.

A suitable sentence to report our findings would be: "Diet has a significant effect on blood pressure ($F(3, 46) = 20.7, p < 0.0001$)". The two numbers in brackets are the degrees of freedom of the model ($k - 1 = 3$) and of the residuals ($n - k = 46$). This means that the probability of observing such a large F value under the null hypothesis is less than 0.01%.

Difference between pairs of groups

If, and only if the F-test is significant, we can conclude that there is a significant difference between the groups. But this does not tell us which groups differ from each other.

If we'd like to know which pairs of groups differ, we have a few issues to contend with.

1. ANOVA does not tell us this!
2. Controlling error rates.

Let's first look at error rates.

Error-rates

Imagine a fire alarm that goes off when there is a fire. The alarm can make two types of errors:

1. Type I error: The alarm goes off when there is no fire (false positive).
2. Type II error: The alarm does not go off when there is a fire (false negative).

In the context of hypothesis testing, a Type I error occurs when we reject the null hypothesis when it is actually true. A Type II error occurs when we fail to reject the null hypothesis when it is actually false.

When we decide that $p < 0.05$ is the threshold for "significance", we are essentially saying that we are willing to accept a 5% chance of making a Type I error. That is, we are willing to accept a 5% chance of rejecting the null hypothesis when it is actually true.

So when we do one test, e.g., and F-test, and use 0.05 as the threshold p for significance, we are setting a **per test error rate** of 0.05.

What happens is we do multiple tests, each with an 0.05 threshold (i.e., 0.05 type 1 error rate)? The more tests we do, the more likely we are to make at least one Type I error. This is known as the problem of multiple comparisons / tests.

To illustrate this, imagine that we have a fire alarm that goes off 5% of the time when there is no fire (i.e., it has a 5% false positive rate). Imagine that we test the alarm 100 times when there is no fire. On average, we would expect the alarm to go off 5 times (5% of 100) just by chance, even though there is no fire in any test. This illustrates the problem of doing multiple tests: the more tests we do, the more likely we are to get a false positive just by chance.

To illustrate this in the context of ANOVA, imagine that we have 10 groups, and we want to test if any pairs of groups differ from each other. This would involve making 45 pairwise comparisons (since there are $\binom{10}{2} = 45$ pairs of groups). If we use a significance threshold of 0.05 for each individual test, we would expect to get about 2.25 false positives just by chance (5% of 45). This means that even if there were no true differences between the groups, we would still expect to find just over two significantly different pairs, even when there really is no significant difference.

Going a little further, we can calculate something called the *family-wise error rate* (FWER), which is the probability of making at least one Type I error across all the tests. The FWER can be calculated as:

$$FWER = 1 - (1 - \alpha)^m$$

where α is the per test error rate (e.g., 0.05) and m is the number of tests (e.g., 45). In our example, the FWER would be:

$$FWER = 1 - (1 - 0.05)^{45} \approx 0.90$$

Yes, you read that right! In this example, with 45 pairwise comparisons and a per test error rate of 0.05, the family-wise error rate is about 90%. This means that there is a 90% chance of making at least one Type I error across all 45 tests, even if there are no true differences between the groups. **It is almost certain**

that we will find at significant difference even though there is no real difference!

So, when we want to find which pairs of groups differ, we cannot just do lots of t-tests and look for significant results. We need to use a method that takes into account the problem of multiple comparisons.

ANOVA doesn't compare pairs

The second issue is that ANOVA does not itself compare any pairs of groups. Rather, it works by doing an F test. This tests the null hypothesis that all group means are equal. It does not tell us which groups are different from each other. It just tells us that fitting group means explains a significant amount of variance in the data.

In fact, we can find a significant F-statistic even when none of the pairs of groups are significantly different (e.g., when tested with a t-test). And we can find the opposite, a non-significant F-statistic even when one of the pairs of groups is significantly different by t-test.

Comparing pairs of groups

The solution to both issues is to use a method that allows us to compare pairs of groups while controlling for the problem of multiple comparisons. There are several methods to do this. The most prominent ones are:

- Bonferroni correction
- Tukey **H**onest **S**ignificant **D**ifferences (HSD) approach
- Fisher **L**east **S**ignificant **D**ifferences (LSD) approach

The second two when implemented in R also provide all possible pairwise comparisons.

Bonferroni correction

If we find a significant F-statistic, we can then proceed to compare pairs of groups while controlling for the problem of multiple comparisons. One method to do this is the Bonferroni correction.

The idea here is to control the family-wise error rate by adjusting the per test error rate. The Bonferroni correction is a simple method to control the family-wise error rate when performing multiple comparisons. It works by adjusting the significance threshold for each individual test to account for the number of tests being performed.

If a total of m tests are carried out, simply divide the per test type-I error level α_0 (often 5%) by the number of tests:

$$\alpha = \alpha_0/m \ .$$

This gives the FWER as:

$$FWER = 1 - (1 - \alpha)^m = 1 - (1 - \alpha_0/m)^m$$

With the previous example of 45 pairwise comparisons and a per test error rate of 0.05, the Bonferroni correction would adjust the per test error rate to:

$$\alpha = 0.05/45 \approx 0.0011$$

And the FWER would be:

$$FWER = 1 - (1 - 0.0011)^{45} \approx 0.05$$

Note that an equivalent method to control the family-wise error rate is to adjust the p-values of the individual tests instead of adjusting the significance threshold. The Bonferroni correction can also be applied by multiplying the p-values of the individual tests by the number of tests. If the adjusted p-value is less than the original significance threshold (e.g., 0.05), then we reject the null hypothesis for that test.

We can quickly get all the pairwise comparisons with the Bonferroni correction by using the `pairwise.t.test` function in R. This function performs pairwise t-tests between groups and applies the specified p-value adjustment method to control for multiple comparisons.

```
pairwise.t.test(bp_data_diet$bp,
                bp_data_diet$diet,
                p.adjust.method = "bonferroni")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  bp_data_diet$bp and bp_data_diet$diet

             meat heavy Mediterranean vegan
Mediterranean 0.0019     -             -
vegan         4.2e-07    0.0091        -
vegetarian    2.6e-07    0.0239        1.0000

P value adjustment method: bonferroni
```

There are six pairwise comparisons (since there are four groups, and $\binom{4}{2} = 6$ pairs of groups). The output shows the p-values for each pairwise comparison after applying the Bonferroni correction. We can see that all pairwise comparisons have a p-value less than 0.05, except for the comparison of vegan versus vegetarian, which has a p-value that rounds to 1.0000.

Let's check our understanding of the Bonferroni adjustment to the p-values. The adjusted p-value for the Mediterranean versus vegetarian comparison is 0.0239. There are six tests in total. So the unadjusted p-value for this comparison would be $0.0239/6 \approx 0.0040$. Let's check this by asking for the unadjusted p-values:

```
pairwise.t.test(bp_data_diet$bp,
                bp_data_diet$diet,
                p.adjust.method = "none")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  bp_data_diet$bp and bp_data_diet$diet

             meat heavy Mediterranean vegan
Mediterranean 0.00031    -             -
vegan         6.9e-08    0.00151       -
vegetarian    4.3e-08    0.00398       0.48386

P value adjustment method: none
```

Yes, the unadjusted p-value for the Mediterranean versus vegetarian comparison is 0.004, which matches our calculation. The Bonferroni correction has adjusted this p-value to 0.0239 to control for the family-wise error rate across all six comparisons.

Tukey HSD approach

Another approach to comparing group means after ANOVA is Tukey's Honest Significant Difference (HSD) method. In this approach, we first perform the ANOVA F-test. Only if the F-test is significant do we proceed to compare pairs of group means.

Instead of adjusting each test separately, it treats all pairwise comparisons as one set of comparisons and calculates a single threshold for differences between group means that keeps the overall false-positive rate at the chosen level. In practice, this means Tukey HSD determines how large a difference between two means must be before we consider it significant, given the number of groups being compared.

In short, Bonferroni adjusts the significance threshold for each test, while Tukey HSD directly determines the minimum difference between group means needed to declare a significant pairwise difference while controlling the overall error rate.

In R we can use the `multcomp` package to do Tukey HSD tests:

```
bp_data_diet <- bp_data_diet |>
  mutate(diet = as.factor(diet))
m_bp_diet <- lm(bp ~ diet, data = bp_data_diet)
```

```r
suppressMessages(library(multcomp))
tukey_test <- glht(m_bp_diet, linfct = mcp(diet = "Tukey"))
summary(tukey_test)
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = bp ~ diet, data = bp_data_diet)

Linear Hypotheses:
                                Estimate Std. Error t value Pr(>|t|)
Mediterranean - meat heavy == 0  -12.688      3.256  -3.897  0.00168 **
vegan - meat heavy == 0          -26.768      4.173  -6.414  < 0.001 ***
vegetarian - meat heavy == 0     -23.625      3.607  -6.549  < 0.001 ***
vegan - Mediterranean == 0       -14.080      4.173  -3.374  0.00759 **
vegetarian - Mediterranean == 0  -10.938      3.607  -3.032  0.01951 *
vegetarian - vegan == 0            3.143      4.453   0.706  0.89305
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

We get all the pairwise comparisons, along with their estimates, standard errors, t-values, and p-values. We also get a note `Adjusted p values reported -- single-step method`, indicating that the Tukey HSD adjustment has been applied to the p-values.

Again, all pairwise comparisons have a p-value less than 0.05, except for the comparison of vegan versus vegetarian, which has a p-value of 0.89305.

Fisher's LSD approach

Another approach to comparing group means after ANOVA is Fisher's Least Significant Difference (LSD) method. In this approach, we first perform the ANOVA F-test. Only if the F-test is significant do we proceed to compare pairs of group means.

In the Fisher LSD approach, we use a two-sample test, but use a larger variance (namely the pooled variance of all groups) that we would use in a two-sample test. This is because we are using the same variance estimate for all pairwise comparisons, which is the mean square error from the ANOVA.

Which method to use?

The Bonferroni correction is more conservative than Tukey HSD, which is more conservative than Fisher LSD. The Bonferroni correction is less powerful than Tukey HSD, which is less powerful than Fisher LSD. So if we want to be more confident that we are not making any false positives, we can use the Bonferroni correction. If we want to be more confident that we are not missing any true positives, we can use the Fisher LSD approach. The Tukey HSD approach is a good middle ground between the two.

Other contrasts

A contrast is a specific comparison between groups. So far we have only considered pairwise contrasts (i.e., comparing two groups at a time). But we can also design more complex contrasts. For example: are diets that contain meat different from diets that do not contain meat?

```
bp_data_diet <- mutate(bp_data_diet,
                       meat_or_no_meat = ifelse(diet == "meat heavy" |
                                                  diet == "Mediterranean",
                                                  "meat", "no meat"))
head(bp_data_diet)
```

```
# A tibble: 6 x 4
     bp diet          person_ID meat_or_no_meat
  <dbl> <fct>         <chr>     <chr>
1   120 meat heavy    person_1  meat
2    89 vegan         person_2  no meat
3    86 vegetarian    person_3  no meat
4   116 meat heavy    person_4  meat
5   115 Mediterranean person_5  meat
6   134 meat heavy    person_6  meat
```

Here we defined a new explanatory variable that groups the meat heavy and Mediterranean diet together into a single "meat" group and vegetarian and vegan into a single "no meat" group. We then fit a model with this explanatory variable:

```
m_bp_meat_or_no_meat <- lm(bp ~ meat_or_no_meat, data = bp_data_diet)
```

(We should not look at model checking plots here, before using the model. But let us continue as if the assumptions are sufficiently met.)

We now do something a bit more complicated: we compare the variance explained by the model with four diets to the model with two diets. This is done by comparing the two models using an F-test. We are testing the null hypothesis that the two models are equally good at explaining the data, in which case the two diet model will explain as much variance as the four diet model.

Let's look at the ANOVA table of the model comparison:

```
anova(m_bp_diet, m_bp_meat_or_no_meat)
```

```
Analysis of Variance Table

Model 1: bp ~ diet
Model 2: bp ~ meat_or_no_meat
  Res.Df    RSS Df Sum of Sq      F   Pr(>F)
1     46 3901.5
2     48 5231.6 -2     -1330 7.8407 0.001175 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see the residual sum of squares of the model with meat or no meat is 5231.6, while that of the four diet model is 3901.5. That is, the four diet model explains more variance in the data than the two diet model. The F-test is significant ($F(2, 46) = 7.84$, $p = 0.0012$), so we reject the null hypothesis that the two models are equally good at explaining the data. And we conclude that its not just whether people eat meat or not, but rather what kind of diet they eat that affects their blood pressure.

Ideally we do not make a lot of contrasts after we have collected and looked at our data. Rather, we would specify the contrasts we are interested in before we collect the data. This is called a priori contrasts. But sometimes we do exploratory data analysis and then we can make post hoc contrasts. In this case we should be careful to adjust for multiple comparisons.

Communicating the results of ANOVA

When communicating the results of an ANOVA, we usually report the F-statistic, the degrees of freedom of the numerator and denominator, and the p-value. For example, we could say:

> Blood pressure differed significantly between groups, with the mean of a meat heavy diet being 123 mmHg, while the mean blood pressure of the vegan group was 27 mmHg lower (One-way ANOVA, $F(3, 46) = 20.7$, $p < 0.0001$.

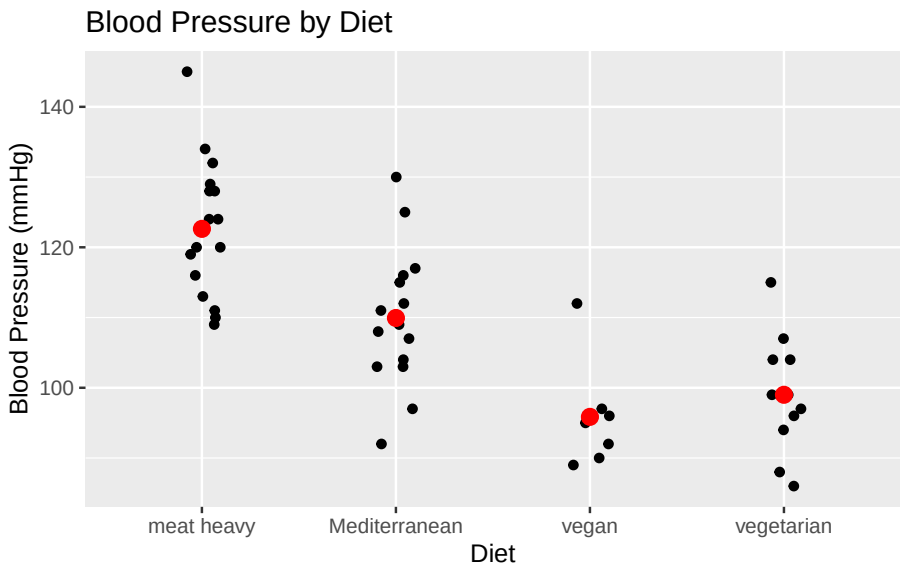
Tip

Report the effect sizes, not only the test. The F-test only tells us whether there is evidence that *some* group means differ. Notice that the sentence above also gives the size of the difference (27 mmHg lower in the vegan group). Even better is to give the uncertainty of the difference too, for example its 95% confidence interval (from `confint()` applied to

the linear model). Readers can then judge whether the difference is large enough to be biologically or medically important.

And we would make a nice graph, in this case showing each individual observation since there are not too many to cause overplotting. We can also add the estimated means of each group if we like:

```
ggplot(bp_data_diet, aes(x = diet, y = bp)) +
  geom_jitter(width = 0.1, height = 0) +
  stat_summary(fun = mean, geom = "point", color = "red", size = 3) +
  labs(title = "Blood Pressure by Diet",
       x = "Diet",
       y = "Blood Pressure (mmHg)")
```

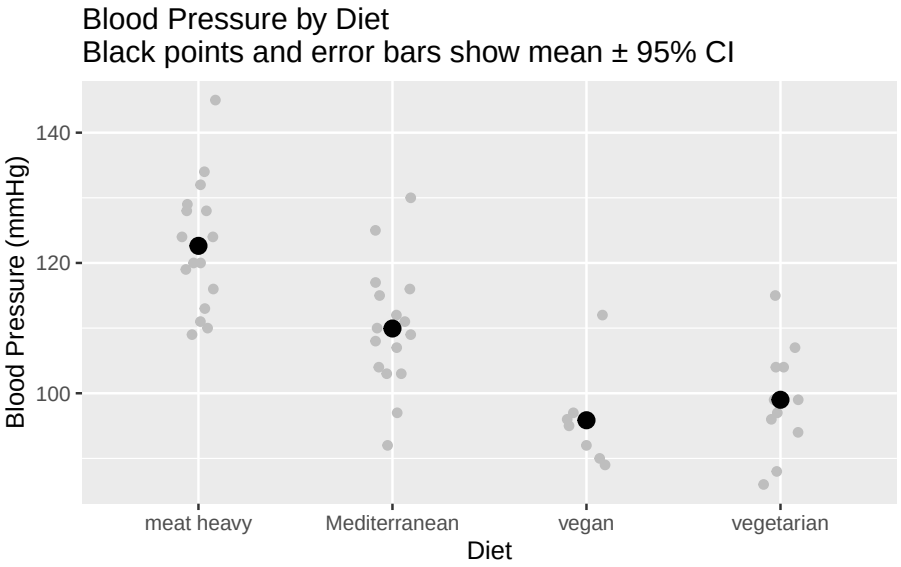


Some people like to see error bars as well, for example showing the 95% confidence intervals of the means:

```
ggplot(bp_data_diet, aes(x = diet, y = bp)) +
  geom_jitter(width = 0.1, height = 0, col = "grey") +
  stat_summary(fun = mean, geom = "point", color = "black", size = 3) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.2, color = "bla
  labs(title = "Blood Pressure by Diet\nBlack points and error bars show mean ± 95% CI
       x = "Diet",
       y = "Blood Pressure (mmHg)")
```

```
Warning: Computation failed in `stat_summary()`.
Caused by error in `fun.data()`:
```

```
! The package "Hmisc" is required.
```



There are many many plotting styles and preferences. The important thing is to clearly communicate the results, and to not mislead the reader. I find that plotting the individual data points is often a good idea, especially when the sample size is not too large.

ANOVA and experimental design

This is not a course about experimental design (this is much too large a subject to cover). You may have noticed, however, that all of the example datasets and questions in your reading about ANOVA have been *manipulative experiments*, where researchers had various treatment groups containing the experimental subjects. That is, the researchers ***manipulated*** something to be different among the groups, and looked for any evidence of resulting differences.

This might lead you to wonder if we always analyse manipulative experiments with ANOVA, and always analyse observational studies (involving no manipulations) with regression. The answer is no, we can use either for either. It just happens that manipulations in experiments are often of the categorical nature: type of food, parasite species, light level (high or low), and so on. Of course some experiments contain manipulations that are continuous (e.g. a range of temperatures or drug concentrations) and these are analysed with regression. And some studies include a manipulation (e.g., drug concentration) and observed variables like gender.

So there isn't necessarily an association between type of explanatory variable (continuous or categorical) and whether a study is experimental or observational.

Essential is for the researcher to clearly be aware of which variables are manipulation and which are observations. Because only the manipulations can be used to infer causation.

Review

Here are the key points from this chapter:

- ANOVA is just another linear model.
- It is used when we have categorical explanatory variables.
- We use F-tests to test the null hypothesis of no difference among the means of the groups (categories).
- We can use contrasts and post-hoc tests to test specific hypotheses about the means of the groups.

Further reading

If you'd like some further reading on ANOVA and ANOVA in R, here are some good resources. There is lots of overlap among them, however, and with the material in this chapter. I suggest to look at them mostly for different perspectives and examples. And only if you'd like to solidify your understanding.

- Chapter 5, Section 5.6, of *Getting Started with R* by Beckerman et al. This section is about 10 pages.
- Chapter 11 of *The New Statistics with R* by Hector, not including the section on two-way ANOVA. This chapter is about 10 pages.
- Chapter 14 of *Linear Models with R* by Faraway. This chapter is eight pages.
- Chapter 9 of **Statistics. An Introduction using R** by Crawley, up to the section *Factorial experiments*. This section is about 16 pages.

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Choosing the reference category

Question: Which diet is the reference (intercept) category in the model `m_bp_diet`, and why?

Answer: R orders the categories alphabetically and takes the first level as the reference category. But "alphabetically" depends on the settings (the "locale")

of the computer. On most computers upper and lower case letters are sorted together, so "meat heavy" comes before "Mediterranean" and is the reference. But some computers (for example some servers) sort all capital letters first, and then "Mediterranean" is the reference. So always check which level is first:

```
levels(factor(bp_data_diet$diet))
```

```
[1] "meat heavy"    "Mediterranean" "vegan"         "vegetarian"
```

If you set the order of the levels yourself (for example with `factor(diet, levels = c("meat heavy", "Mediterranean", "vegan", "vegetarian"))`), the reference is the same on every computer.

Sometimes we may want to override this, for example if we have a treatment that is experimentally the control, then it will usually be useful to set this as the reference / intercept level.

In R we can set the reference level using the `relevel` function:

```
bp_data_diet$diet <- relevel(factor(bp_data_diet$diet), ref = "vegan")
```

And now make the model and look at the estimated coefficients:

```
m_bp_diet_vegan_ref <- lm(bp ~ diet, data = bp_data_diet)
summary(m_bp_diet_vegan_ref)
```

```

Call:
lm(formula = bp ~ diet, data = bp_data_diet)

Residuals:
     Min       1Q   Median       3Q      Max
-17.9375  -5.9174  -0.4286   5.2969  22.3750

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)        95.857      3.481  27.538  < 2e-16 ***
dietmeat heavy     26.768      4.173   6.414 6.92e-08 ***
dietMediterranean  14.080      4.173   3.374  0.00151 **
dietvegetarian      3.143      4.453   0.706  0.48386
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.21 on 46 degrees of freedom
Multiple R-squared:  0.5748,    Adjusted R-squared:  0.5471
F-statistic: 20.73 on 3 and 46 DF,  p-value: 1.214e-08
```

Now we see the estimated coefficients for all diets except the vegan diet. The intercept is the mean individuals with vegan diet.

Multiple regression (L6)

Learning objectives

After this unit, students will be able to:

1. Identify biological questions that require a model with more than one explanatory variable.
2. Write the multiple regression model as an equation, and explain what each coefficient represents.
3. Use R to fit a multiple regression with `lm()`.
4. Use diagnostic plots to judge whether the assumptions of a multiple regression are reasonably met.
5. Calculate the model and residual degrees of freedom of a multiple regression, counting k − 1 degrees of freedom for a categorical explanatory variable with k levels, and use them in the F-test and in adjusted R^2.
6. Use the overall F-test to judge whether the explanatory variables together explain variation in the response.
7. Interpret each coefficient as the association between that explanatory variable and the response with the other explanatory variables held constant, including its direction and size.
8. Use R to obtain R^2 and adjusted R^2, and explain why adjusted R^2 is needed when comparing models with different numbers of explanatory variables.
9. Compare the importance of explanatory variables measured on different scales using standardised coefficients, and explain what is lost by standardising.
10. Use `predict()` to make predictions for given values of all explanatory variables.
11. Explain what collinearity is, detect it (e.g. correlations among explanatory variables, variance inflation factors), and describe its consequences for coefficient estimates and their uncertainty.
12. Use the residual degrees of freedom of two nested models in an F-test comparing them.

13. Explain what a confounding variable is, how including a measured confounder in a multiple regression reduces bias in the estimated effect of another variable, and why this cannot prove causation.
14. Recognise a correct and complete statement of multiple regression results.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch06_multiple_regression.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 6 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch06_multiple_regression.R")
```

Video walkthrough: coming soon.

Introduction

In the previous chapters we covered simple linear regression and one-way analysis of variance. In both we had one response variable and one explanatory variable. In both cases we made a linear model to relate the response variable to the explanatory variable. The two cases differed in the type of explanatory variable. In the case of simple linear regression, the explanatory variable was continuous. In the case of one-way ANOVA, the explanatory variable was categorical.

We will now extend the linear model and analyses to cases with more than one (i.e., multiple) explanatory variables. The explanatory variables can be continuous or categorical, and can be a mixture of the two.

Some combinations of explanatory variables have special names:

- Multiple (more than one) continuous explanatory variables -> Multiple linear regression.
- Two categorical explanatory variables -> Two-way ANOVA.

- One continuous and one categorical explanatory variable -> Analysis of covariance (ANCOVA).

We will look at each of these, and start with multiple linear regression which is usually shortened to just multiple regression.

Multiple regression

We previously looked at whether blood pressure is associated with age. This is an important question, because blood pressure has many health implications. However, blood pressure is not only associated with age, but also with other factors, such as weight, height, and lifestyle. In this chapter, we will look at how to investigate the association between blood pressure and multiple explanatory variables.

When we have multiple explanatory variables, we are often interested in questions such as:

- Question 1: As an ensemble (i.e., all together), are the explanatory variables "useful"?
- Question 2: Are each of the explanatory variables associated with the response?
- Question 3: What proportion of variability is explained?
- Question 4: Are some explanatory variables more important than others?

An example dataset

Blood pressure is again the response variable, with age and lifestyle as two explanatory variables. Lifestyle is a continuous variable that is the number of minutes of exercise per week.

Reading in the dataset:

```
bp_data_multreg <- read_csv("datasets/bp_data_multreg.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): age, mins_exercise, bp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here is a look at the dataset:

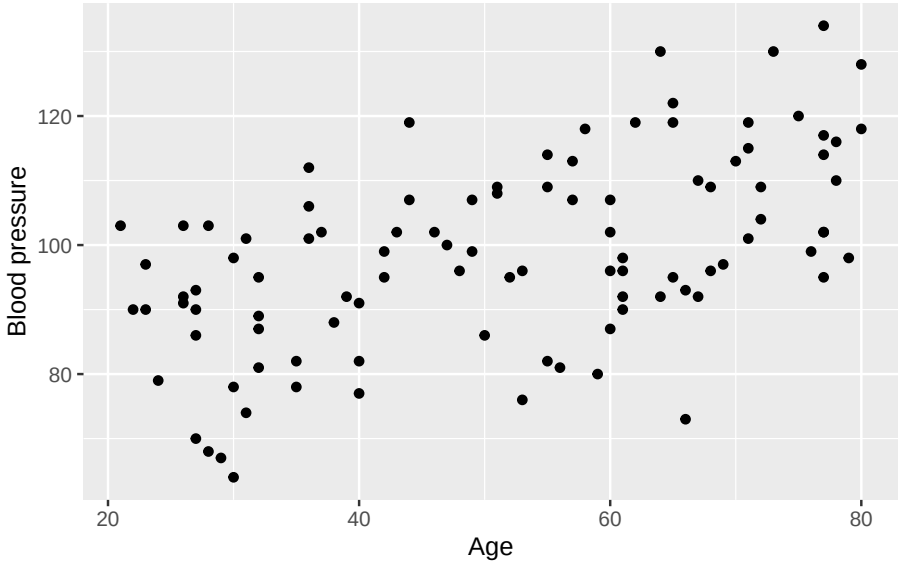
```
head(bp_data_multreg)
```

```
# A tibble: 6 x 3
    age mins_exercise    bp
```

```
  <dbl>          <dbl> <dbl>
1    36            157   101
2    61            249    92
3    29            283    67
4    64            178    92
5    42            115    99
6    23             13    97
```

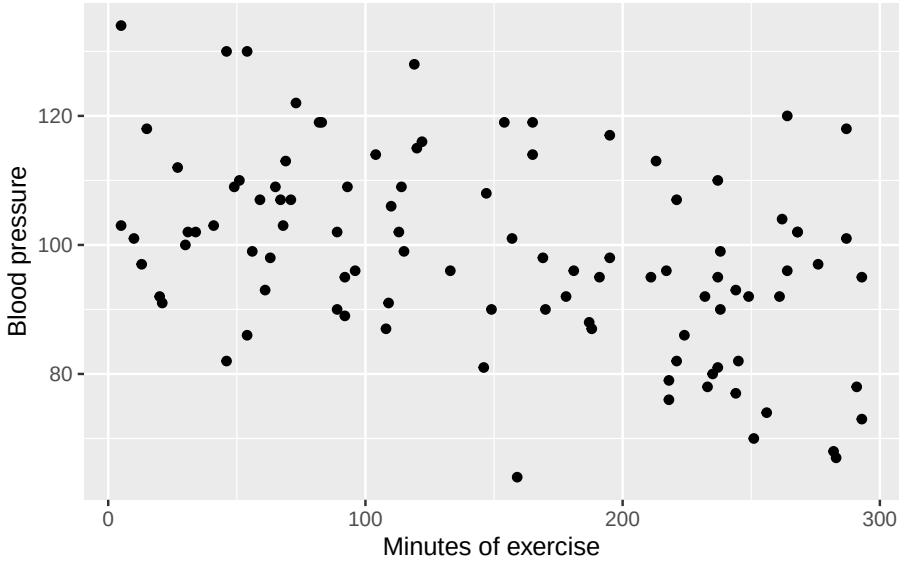
Since there are three variables, we can make three different scatter plots to visualise the relationships:

1. Age vs blood pressure. This is the graph of the response variable (blood pressure) against one of the explanatory variables (age). It looks like there is evidence of a positive relationship.



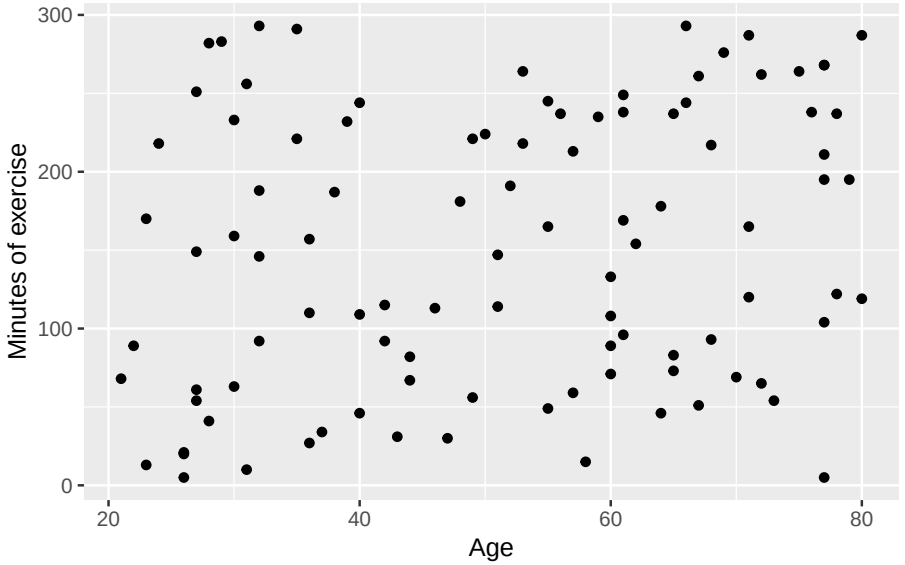
Here we see a positive relationship between age and blood pressure. Blood pressure tends to increase with age.

2. Minutes of exercise vs blood pressure. This is a graph of the response variable (blood pressure) against the other explanatory variable (minutes of exercise). It looks like there is evidence of a negative relationship.



Here we see a negative relationship between minutes of exercise and blood pressure. Blood pressure tends to decrease with more minutes of exercise.

3. Age vs minutes of exercise. This is a graph of the two explanatory variables against each other. It looks like there is no relationship.



And here we see no relationship between age and minutes of exercise. The two explanatory variables appear to be independent.

> **!** Important
>
> The lack of correlation between the two explanatory variables is very important. If the two explanatory variables were correlated, we would have a situation known as multicollinearity. Multicollinearity can greatly complicate the interpretation of the results of a multiple regression analysis. We will discuss multicollinearity later.

The multiple linear regression model

The multiple linear regression model is an extension of the simple linear regression model. Recall the simple linear regression model is:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where:

- y_i is the response variable
- x_i is the explanatory variable
- β_0 is the intercept
- β_1 is the slope
- ϵ_i is the error term.

The multiple linear regression model with two explanatory variables is:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \epsilon_i$$

where:

- $x_i^{(1)}$ and $x_i^{(2)}$ are the two explanatory variables
- β_0 is the intercept
- β_1 is the slope for the first explanatory variable
- β_2 is the slope for the second explanatory variable

Note that the intercept β_0 is the value of the response variable when all explanatory variables are zero. In this example, it would be the blood pressure for someone that is 0 years old and does 0 minutes of exercise per week. This is not a particularly useful scenario, but it is a necessary mathematical construct that helps us to build the model.

We can extend the multiple regression model to have an arbitrary number of explanatory variables:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + ... + \beta_p x_i^{(p)} + \epsilon_i$$

Where:

$x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(p)}$ are the p explanatory variables and all else is as before.

or with summation notation:

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_i^{(j)} + \epsilon_i$$

Just like in simple linear regression, we can estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$ using the method of least squares. The least squares method minimizes the sum of the squared residuals:

$$\sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the predicted value of the response variable for the ith observation:

$$\hat{y}_i = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_i^{(j)}$$

where:

$\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ are the estimated parameters.

Here is a graph of the geometric representation of a multiple linear regression model with two explanatory variables (this interactive figure is shown only in the online, HTML, version of the book):

This interactive 3-D figure is shown only in the online version of the book: see this chapter online.

Let's write the equation for the blood pressure data:

$$bp_i = \beta_0 + \beta_1 \cdot age_i + \beta_2 \cdot mins\_exercise_i + \epsilon_i$$

where:

- bp_i is the blood pressure for the ith observation
- age_i is the age for the ith observation
- $mins\_exercise_i$ is the minutes of exercise for the ith observation
- β_0 is the intercept
- β_1 is the slope for age
- β_2 is the slope for minutes of exercise
- ϵ_i is the error term

and the error term is assumed to be normally distributed with mean 0 and constant variance, just as was the case for simple linear regression:

$$\epsilon_i \sim N(0, \sigma^2)$$

What we already know from simple linear regression also applies here: how to estimate the parameters by least squares, how to check the assumptions of the model, and how to use the model to make predictions (with confidence and prediction bands).

What we don't know yet is how to answer the four questions already mentioned above:

- Question 1: As an ensemble (i.e., all together), are the explanatory variables "useful"?
- Question 2: Are each of the explanatory variables associated with the response?
- Question 3: What proportion of variability is explained?
- Question 4: Are some explanatory variables more important than others?

Let's answer these questions using the blood pressure example.

Fitting the model

We know how to estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$ using the method of least squares.

In R, we can fit a multiple linear regression model using the `lm()` function in a very similar way to the simple linear regression model. Here is the code for the blood pressure example. To fit two explanatory variables, we simply add the second variable to the formula using the `+` sign:

```
m_bp_age_exercise <- lm(bp ~ age + mins_exercise, data = bp_data_multreg)
```

Checking the assumptions

Great news –> the five assumptions of the multiple linear regression model are the same as for the simple linear regression model:

a) Normality of residuals.
b) Homoscedasticity = constant variance of residuals.
c) Independence of residuals.
d) Linearity.
e) No outliers.

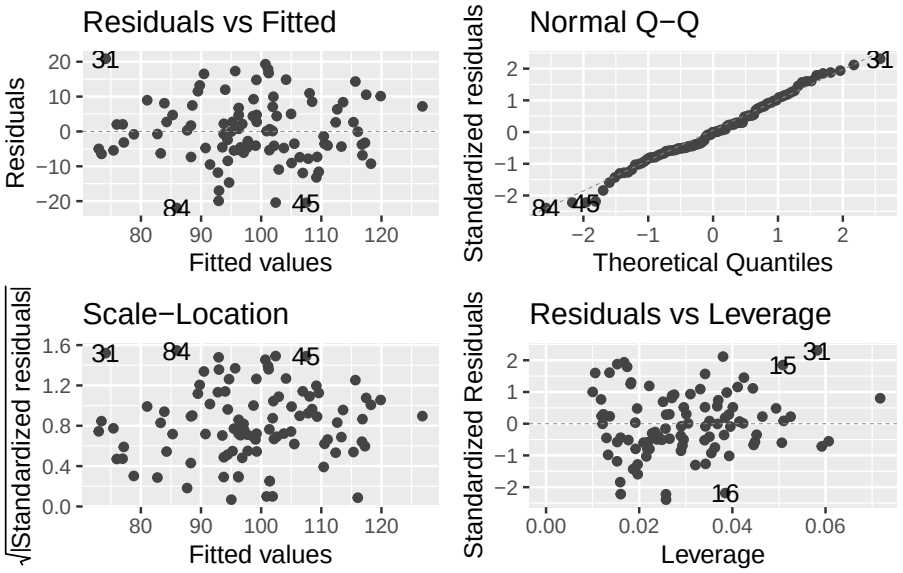
We can check the assumptions of the multiple linear regression model using the same methods as for the simple linear regression model. Here is the code for the blood pressure example:

```
# Check the assumptions
autoplot(m_bp_age_exercise, which = c(1, 2, 3, 5), smooth.colour = NA)
```

```
Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.
i Please use `broom::augment(<lm>)` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



We see that the assumptions are met for the blood pressure example:

a) Normality of residuals: The QQ-plot shows that the residuals are normally distributed.
b) Homoscedasticity: The scale-location plot shows that the residuals have constant variance.
c) Independence of residuals: No evidence of pattern or clustering. But also need to know about study design to properly assess independence.
d) Linearity: The residuals vs. fitted values plot shows no clear pattern in the residuals.
e) No outliers: No points with high leverage or high residuals.

Question 1: As an ensemble, are the explanatory variables useful?

Recall that when we learned about ANOVA we saw that a single categorical explanatory variable with multiple levels can be represented as multiple binary (0/1) explanatory variables. In that case, we used the F-test to test the null hypothesis of no effect / relationship for all binary variables together.

Likewise, when we have multiple continuous explanatory variables, we use the F-test to test the null hypothesis that **together** the explanatory variables have no association with the response variable. That is, we use the F-test to test the null hypothesis that the ensemble of explanatory variables is not associated with the response variable.

This corresponds to the same null hypothesis as we used in one-way ANOVA: The null hypothesis that the explained variance of the model is no greater than would be expected by chance. Here, "by chance" means that the slopes of the explanatory variables are zero:

$$H_0 : \beta_1 = \beta_2 = ... = \beta_p = 0$$

And the alternative hypothesis (just as in one-way ANOVA) is that the explained variance of the model is greater than would be expected by chance. This would occur if the slopes of some or all of the explanatory variables are not zero:

$$H_1 : \beta_1 \neq 0 \text{ or } \beta_2 \neq 0 \text{ or } ... \text{ or } \beta_p \neq 0$$

Recall that the F-test compares the variance explained by the model to the variance not explained by the model (i.e., the variance of the residuals). If the variance explained by the model is significantly greater than the variance not explained by the model, then we can conclude that the explanatory variables are associated with the response variable.

If we reject the null hypothesis, we can conclude that some combination of the explanatory variables is associated with the response variable. However, we cannot conclude which specific explanatory variables are associated with the response variable. To determine which specific explanatory variables are associated with the response variable, we need to perform individual t-tests for each explanatory variable. We will do this in the next section.

OK, back to the F-test.

We know a lot already from the ANOVA chapter. Let's review how we calculate the F-statistic.

The F-statistic is calculated as the ratio of two mean squares:

1. The mean square of the model (MSE_{model}).

2. The mean square of the residuals ($MSE_{residual}$).

Recall that a mean square is a sum of squares divided by the associated degrees of freedom. The formulas for these are the same as for ANOVA.

So, to calculate these two mean squares, we need to calculate three sums of squares:

1. The total sum of squares (SST).
2. The sum of squares of the model (SSM).
3. The sum of squares of the residuals (SSE).

We also need to calculate the degrees of freedom associated with each sum of squares.

1. The total degrees of freedom is $n-1$, where n is the number of observations.
2. The model degrees of freedom is p, where p is the number of parameters estimated in addition to the intercept. Each estimated parameter uses up one degree of freedom. A continuous explanatory variable has one parameter (its slope), so in our example with two continuous explanatory variables, $p = 2$. A categorical explanatory variable with k levels has $k-1$ parameters, so it contributes $k-1$ degrees of freedom.
3. The residual degrees of freedom is $n-1-p$.

The F-statistic in R

We could do all these calculations ourselves (and you might be asked to in the exam), but also we can just ask R! For question 1 we need to know the F-statistic for the multiple linear regression model. We can easily get this from R using the `summary()` function, and by looking in the right place in the output:

```
summary(m_bp_age_exercise)
```

```
Call:
lm(formula = bp ~ age + mins_exercise, data = bp_data_multreg)

Residuals:
     Min       1Q   Median       3Q      Max
-21.9748  -5.3890  -0.0555   6.4472  20.8549

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   84.74631    3.02000  28.062  < 2e-16 ***
age            0.55255    0.05348  10.331  < 2e-16 ***
mins_exercise -0.09653    0.01086  -8.889 3.35e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 9.298 on 97 degrees of freedom
Multiple R-squared:  0.6115,    Adjusted R-squared:  0.6035
F-statistic: 76.35 on 2 and 97 DF,  p-value: < 2.2e-16
```

In the final line of output we see "F-statistic: 76.35 on 2 and 97 DF, p-value: 1.21×10^{-20}".

The model degrees of freedom is 2 (because we have two explanatory variables), and the residual degrees of freedom is 97 (because we have 100 observations and 2 explanatory variables, so $100 - 1 - 2 = 97$).

So the `summary()` function gives us everything we need to answer question 1. It even gives us the p-value for the F-test.

How to report the result. We could write somthing like this: "The combination of age and minutes of exercise is significantly associated with blood pressure ($F(2, 97) = 76.35$, $p = 1.21 \times 10^{-20}$)." Note that this is rather an undesirable statement, because it focuses too much on the statistics and not enough on the science. Indeed, perhaps we care more about the association of each explanatory variable with blood pressure, which we will look at next.

> **Note**
>
> If you want to review how to calculate a p-value from an F-statistic, see the corresponding section of the one-way ANOVA chapter.

Question 2: Which variables are associated with the response?

As we did for simple linear regression, we can perform a t-test for one explanatory variable to determine if it is associated with the response. And we can do this for each of the explanatory variables. As before, the null hypothesis for each t-test is that the slope of the explanatory variable is zero. The alternative hypothesis is that the slope of the explanatory variable is not zero.

Here is the coefficients table, which includes the results of the t-tests for each explanatory variable:

```
summary(m_bp_age_exercise)$coef
```

```
                Estimate Std. Error   t value     Pr(>|t|)
(Intercept)   84.74630657 3.01999756 28.061714 2.380472e-48
age            0.55255207 0.05348274 10.331409 2.584306e-17
mins_exercise -0.09652863 0.01085909 -8.889205 3.349927e-14
```

And we can get the 95% CI for each slope estimate $\hat{\beta}_j$ as follows:

```
confint(m_bp_age_exercise)
```

```
                  2.5 %      97.5 %
(Intercept)   78.7524474 90.74016574
age            0.4464036  0.65870050
mins_exercise -0.1180809 -0.07497635
```

Reminder: The 95% confidence interval is $[\hat{\beta} - c \cdot \sigma^{(\beta)}; \hat{\beta} + c \cdot \sigma^{(\beta)}]$, where c is the 97.5% quantile of the t-distribution with $n - p - 1$ degrees of freedom (the residual degrees of freedom).

> **!** Important
>
> **However** Please insert a note into your brain that we are dealing here with an ideal case of **uncorrelated explanatory variables**. You'll learn later in the course about what happens when explanatory variables are correlated. Hint: interpretation is difficult and unstable!

Question 3: What proportion of variability is explained?

Multiple R^2

We can calculate the R^2 value for the multiple linear regression model just like we already did for a simple linear regression model. The R^2 value is the proportion of variability in the response variable that is explained by the model. As before, the R^2 value ranges from 0 to 1, where 0 indicates that the model does not explain any variability in the response variable, and 1 indicates that the model explains all the variability in the response variable.

For multiple linear regression, we often use the term "multiple R^2" to distinguish it from the R^2 value for simple linear regression. The multiple R^2 is the proportion of variability in the response variable that is explained by the model, taking into account all the explanatory variables in the model.

As before, for simple linear regression, the multiple R^2 value is calculated as the sum of squares explained by the model divided by the total sum of squares:

$$R^2 = \frac{SSM}{SST}$$

```
anova_bp_age_exercise <- anova(m_bp_age_exercise)
SSM <- anova_bp_age_exercise$`Sum Sq`[1] + anova_bp_age_exercise$`Sum Sq`[2]
SST <- sum(anova_bp_age_exercise$`Sum Sq`)
R_squared <- SSM / SST
#R_squared
```

where SSM is the sum of squares explained by the model and SST is the total sum of squares, and $SSM = SST - SSE$.

For the blood pressure data:

```
summary(m_bp_age_exercise)$r.squared
```

```
[1] 0.6115492
```

R^2 for multiple linear regression can also be calculated as the squared correlation between $(y_1, \ldots, y_n)$ and $(\hat{y}_1, \ldots, \hat{y}_n)$, where the $\hat{y}$ are the fitted values from the model. The fitted values are calculated as:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x^{(1)} + \ldots + \hat{\beta}_m x^{(m)}$$

In R:

```
r_squared <- cor(m_bp_age_exercise$fitted.values, bp_data_multreg$bp)^2
r_squared
```

```
[1] 0.6115492
```

Or:

```
anova_bp_age_exercise <- anova(m_bp_age_exercise)
SSM <- anova_bp_age_exercise$`Sum Sq`[1] + anova_bp_age_exercise$`Sum Sq`[2]
SST <- sum(anova_bp_age_exercise$`Sum Sq`)
R_squared <- SSM / SST
R_squared
```

```
[1] 0.6115492
```

Adjusted R^2

However, we have a little problem to address. The R^2 value increases as we add more explanatory variables to the model, even if the additional variables are not associated with the response. This is because the R^2 value is calculated as the proportion of variability in the response variable that is explained by the model. As we add more explanatory variables to the model, the model will always explain more variability in the response variable, even if the additional variables are not associated with the response. Some of the variance will be explained by chance.

Here is an example of this problem. First, here's the explanatory power of the model with only age and minutes of exercise as the explanatory variables:

```
m_bp_age_exercise <- lm(bp ~ age + mins_exercise, data = bp_data_multreg)
summary(m_bp_age_exercise)$r.squared
```

```
[1] 0.6115492
```

Now, we can add a new explanatory variable to the blood pressure model that is not associated with the response:

```
set.seed(1)
bp_data_multreg$random_variable <- rnorm(nrow(bp_data_multreg))
m_bp_age_exercise_random <- lm(bp ~ age + mins_exercise + random_variable, data = bp_data_multreg
summary(m_bp_age_exercise_random)$r.squared
```

```
[1] 0.6241558
```

The R^2 value for the model with the random variable is higher than the R^2 value for the model without the random variable. This is because the model with the random variable explains more variability in the response variable, even though the random variable is not associated with the response.

To address this problem, we can use the adjusted R^2 value. The adjusted R^2 value is calculated as:

$$R^2_{\text{adj}} = 1 - \frac{SSE/(n-p-1)}{SST/(n-1)}$$

where * SSE is the sum of squared errors * SST is the total sum of squares * n is the number of observations * p is the number of explanatory variables in the model.

Or put another way:

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

In this form, we can see that as p increases (as we add explanatory variables) the term $(n-1)/(n-p-1)$ increases, and the adjusted R^2 value will decrease if the additional variables are not associated with the response.

Take home: when we want to compare the explanatory power of models that differ in the number of explanatory variables, we should use the adjusted R^2 value.

Question 4: Are some explanatory variables more important than others?

We know how to test the significance of the parameters using the t-test. We also know how to calculate the confidence intervals for the parameters, and to make a confidence band.

How important are the explanatory variables and how important are they relative to each other?

The importance of an explanatory variable can be assessed by looking at the size of the coefficient for that variable. The larger the coefficient, the more important the variable is in explaining the response variable.

It is, however, important to remember that the size of the coefficient depends on the scale of the explanatory variable. If the explanatory variables are on different scales, then the coefficients will be on different scales and cannot be directly compared.

In our example, the age variable is measured in years, so the coefficient is in units mmHg (pressure) per year. The mins_exercise variable is measured in minutes, so the coefficient is in units mmHg per minute. The coefficients are on different scales and cannot be directly compared. Furthermore, the value of the coefficients would change if we measured age in months or minutes of exercise in hours.

There are other perspectives we can take when we're assessing importance. For example, we cannot change our age, but we can change the number of minutes of exercise. So, the practical importance of the two variables is quite different in that sense also.

To compare the importance of the explanatory variables that are measured on different scales, we can standardize the variables before fitting the model. This means that we subtract the mean of the variable and divide by the standard deviation. This puts all the variables on the same scale, so the coefficients can be directly compared. The coefficients are then in units of the response variable per standard deviation of the explanatory variable.

However, the coefficients are then not in the original units of the explanatory variables, so it is not always easy to interpret the coefficients. So while we can compare the coefficients, they have lost a bit of their original meaning and are not so easy to interpret.

One way to relate the coefficients in this case is to realise that to compensate for the blood pressure increase associated with one year of age, one would need to exercise for a certain number of minutes more.

```
extra_mins_exercise <- coef(m_bp_age_exercise)["age"] / -coef(m_bp_age_exercise)["mins
#extra_mins_exercise
```

Question 5: How do we make predictions?

We already made predictions from a simple linear regression model?

Recall that the equation for multiple linear regression is:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + ... + \beta_p x_i^{(p)} + \epsilon_i$$

Therefore to get a predicted value of y_i, we can use the estimated parameters $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)$ and the values of the explanatory variables $(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(p)})$:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i^{(1)} + \hat{\beta}_2 x_i^{(2)} + \ldots + \hat{\beta}_p x_i^{(p)}$$

In R, we can use the `predict()` function to make predictions from a multiple linear regression model. Here is an example of how to make predictions for the values of the explanatory variables in the original dataset:

```
predictions <- predict(m_bp_age_exercise)
```

Where `m_bp_age_exercise` is the multiple linear regression model fitted earlier. The `predict` function automatically uses the original data because we did not provide any new data.

We can also make predictions for new values of the explanatory variables by providing a new data frame to the `predict()` function. Here is an example of how to make predictions for new values of age and minutes of exercise:

First we need to make some new values of age and minutes of exercise:

```
new_data <- data.frame(age = c(30, 40, 50),
                       mins_exercise = c(50, 100, 150))
```

Then we can use the `predict()` function to make predictions for these new values:

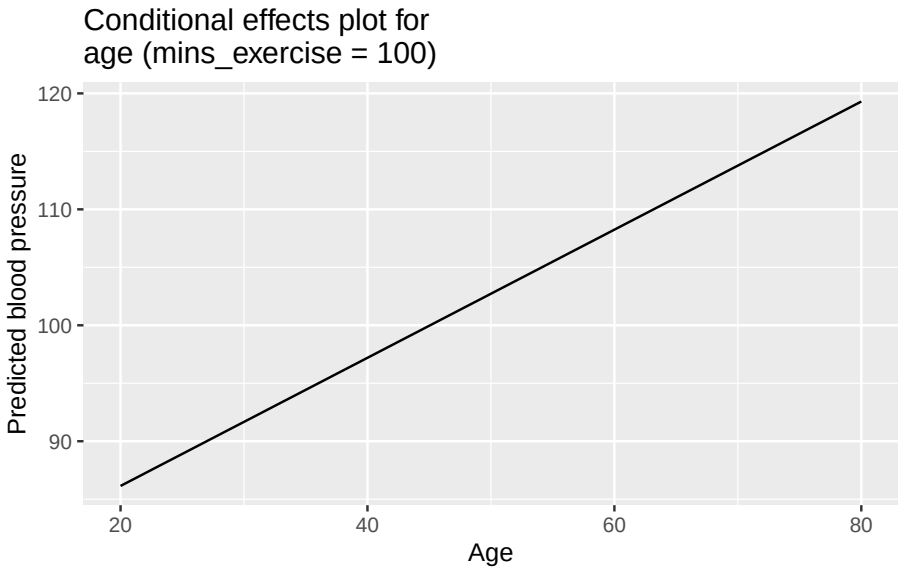
```
new_predictions <- predict(m_bp_age_exercise, newdata = new_data)
```

> **!** Important
>
> The new data frame must have the all the explanatory variables used in the model, and the variable names must match exactly those used in the model.

We can take this to the next level and make what is called a "conditional effects plot" or "effect plot". This is a plot that shows the predicted values of the response variable for different values of one explanatory variable, while holding the other explanatory variables constant. Here is an example of how to make a conditional effects plot for age, while holding minutes of exercise constant at 100:

```
new_data_effects <- data.frame(age = seq(20, 80, by = 1),
                               mins_exercise = 100)
new_data_effects <- new_data_effects |>
  mutate(new_predictions_effects = predict(m_bp_age_exercise, newdata = new_data_effects))
ggplot(data = new_data_effects, aes(x = age, y = new_predictions_effects)) +
  geom_line() +
```

```
  labs(x = "Age", y = "Predicted blood pressure") +
  ggtitle("Conditional effects plot for\nage (mins_exercise = 100)")
```



And lets take this to the next level again and make a conditional effects plot for three different levels of `mins_exercise`:

```
new_data_effects_3 <- expand.grid(age = seq(20, 80, by = 1),
                                  mins_exercise = c(0, 150, 300))
```

A new function! And it's one of Owen's favourites: `expand.grid()`. This function creates a data frame from all combinations of the supplied vectors or factors. Here, we are creating a data frame with all combinations of age (from 20 to 80) and minutes of exercise (0, 150, and 300).

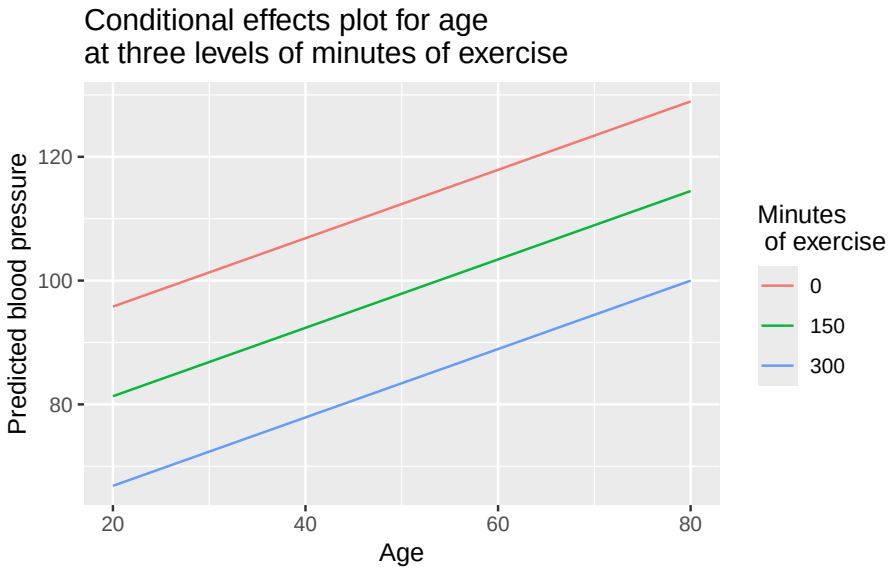
We then give the new data frame to the `predict()` function to get the predicted values for each combination of age and minutes of exercise:

```
new_data_effects_3 <- new_data_effects_3 |>
  mutate(prediction = predict(m_bp_age_exercise, newdata = new_data_effects_3),
         mins_exercise_fac = as.factor(mins_exercise))
```

(Note that we also made a new variable, `mins_exercise_fac`, which is `mins_exercise` converted to a factor. Then ggplot gives each of the three values of minutes of exercise its own colour, and a legend with just these three values, instead of a continuous colour scale.)

And make a graph:

```
ggplot(data = new_data_effects_3, aes(x = age, y = prediction,
                                      col = mins_exercise_fac)) +
  geom_line() +
  labs(x = "Age", y = "Predicted blood pressure", color = "Minutes
 of exercise") +
  ggtitle("Conditional effects plot for age\nat three levels of minutes of exercise")
```



Reporting

As for simple regression, a good report of a multiple regression gives the effect size of each explanatory variable, with its uncertainty, in meaningful units, as well as the test results. For example:

Blood pressure increased with age and decreased with the amount of exercise (multiple regression, $R^2 = 0.61$, $n = 100$). With the amount of exercise held constant, blood pressure was 5.5 mmHg higher for every ten years of age (95% CI 4.5 to 6.6). With age held constant, each additional hour of exercise per week was associated with blood pressure 5.8 mmHg lower (95% CI 4.5 to 7.1).

Notice that the effects are expressed over meaningful amounts (ten years of age, one hour of exercise) rather than per year and per minute, which makes it easier for readers to judge whether they are biologically important.

Collinearity

In the blood pressure example data used previously in this chapter there is no evidence of correlation between the two explanatory variables. However, in practice, it is common for explanatory variables to be correlated with each other. This is known as collinearity. It can be quite problematic for us!

Collinearity, specifically *harmful* collinearity, is extremely common in real dataset that result from observational studies. This is because in observational studies there are often numerous explanatory variables, and they are often correlated with each other. This is a situation that is ripe for collinearity problems. (Collinearity can also happen in data resulting from designed manipulative experiments, but is hopefully relatively rare there because a well-designed experiment will try to avoid collinearity by ensuring that the explanatory variables are independent.)

So what is collinearity, and why is it a problem?

Put simply, collinearity is when one explanatory variable is predictable from a linear combination of others.

This can happen due to strong correlation among pairs of explanatory variables, or due to more complex relationships involving three or more explanatory variables.

For example, if we have three explanatory variables, x_1, x_2, and x_3, and if x_3 can be predicted from a linear combination of x_1 and x_2, then we have collinearity. For example, if:

$$x_3 = 2 \cdot x_1 + 3 \cdot x_2 + \text{small random noise}$$

In this case, variable x_3 is a linear combination of x_1 and x_2, plus some small random noise. This means that if we know the values of x_1 and x_2, we can predict the value of x_3 quite accurately. Also, in this case we might not have strong correlation between any pair of the explanatory variables, but there is still collinearity because x_3 is predictable from x_1 and x_2. So lack of correlation between pairs of explanatory variables does not guarantee that there is no collinearity.

It is a problem because it makes the slope estimates unstable and therefore difficult to interpret.

Let us see this instability in practice. First let's look at the really extreme example of perfect collinearity. Here's a new version of the blood pressure data in which the minutes of exercise variable is perfectly predicted from age:

```
set.seed(123)
n <- 100
age <- ceiling(runif(n, 20, 80))
```

Here is the line of code that makes the perfect correlation between age and minutes of exercise:

```
mins_exercise <- 100 - age
```

Now we generate the blood pressure variable as before:

```
bp <- 100 + 0.5 * age + rnorm(n, 0, 15)
bp_data_perfect <- data.frame(age = age,
                              mins_exercise = mins_exercise,
                              bp = bp)
```

Read in the data:

```
bp_data_perfect <- read_csv("datasets/bp_data_perfect.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): age, mins_exercise, bp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now we fit a simple linear regression model with only age:

```
m1_age <- lm(bp ~ age, data = bp_data_perfect)
summary(m1_age)
```

```

Call:
lm(formula = bp ~ age, data = bp_data_perfect)

Residuals:
    Min      1Q  Median      3Q     Max 
-33.546  -9.147  -0.327   8.938  33.213 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 100.43331    4.54751  22.085  < 2e-16 ***
age           0.47541    0.08549   5.561 2.33e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.54 on 98 degrees of freedom
Multiple R-squared:  0.2399,    Adjusted R-squared:  0.2321 
F-statistic: 30.92 on 1 and 98 DF,  p-value: 2.325e-07
```

The estimated slope (coefficient) is 0.48, which is close to the true value of 0.5. The 95% confidence interval is 0.31 to 0.65, which includes the true value of 0.5. All good then.

Now we fit the multiple linear regression model with both age and minutes of exercise included:

```
m1_both <- lm(bp ~ mins_exercise + age, data = bp_data_perfect)
summary(m1_both)
```

```
Call:
lm(formula = bp ~ mins_exercise + age, data = bp_data_perfect)

Residuals:
    Min      1Q  Median      3Q     Max
-33.546  -9.147  -0.327   8.938  33.213

Coefficients: (1 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   147.97398    4.48276  33.010  < 2e-16 ***
mins_exercise  -0.47541    0.08549  -5.561 2.33e-07 ***
age                NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.54 on 98 degrees of freedom
Multiple R-squared:  0.2399,    Adjusted R-squared:  0.2321
F-statistic: 30.92 on 1 and 98 DF,  p-value: 2.325e-07
```

This output is a bit strange. The estimate for age is now `NA`. This is because the model cannot distinguish between the effects of age and minutes of exercise, since they are perfectly correlated. The model is unable to estimate the coefficient for age, so it returns `NA`.

This is an example of instability of coefficients due to collinearity. The coefficient for age is completely unstable, as it changes from a number to `NA` depending on whether minutes of exercise is included in the model or not.

Actually, you can see that the estimate for `mins_exercise` are the same (except the sign of the coefficient) as the estimate for age in the previous model. This is because minutes of exercise is perfectly correlated with age, so the model is essentially using minutes of exercise as a proxy for age. With perfect collinearity, the model cannot distinguish between the effects of the two variables… they are effectively identical.

You can also see that the R^2 value doesn't change when the second variable is added. That is, the model with only age included is identical to the model with both age and minutes of exercise included. This is because minutes of exercise is perfectly correlated age, so including minutes of exercise in the model does not add any new information.

That is a pretty extreme example of perfect collinearity. In practice, collinearity

is often not perfect, but still strong enough to cause problems.

Let's look at the less extreme example of collinearity we had earlier, with three explanatory variables, x_1, x_2, and x_3, and where x_3 can be predicted from a linear combination of x_1 and x_2 plus some random noise:

```
set.seed(123)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- 2 * x1 + 3 * x2 + rnorm(n, 0, 5)
y <- 5 + 1.5 * x1 - 2 * x2 + 0.5 * x3 + rnorm(n, 0, 1)
data_collinear <- data.frame(x1 = x1, x2 = x2, x3 = x3, y = y)
```

This dataset is available to load as `data_collinear`, in the book dataset collection:

```
data_collinear <- read_csv("datasets/data_collinear.csv")
```

```
Rows: 100 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (4): x1, x2, x3, y

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Look at the correlations between the variables in the dataset:

```
cor(data_collinear)
```

```
            x1          x2        x3          y
x1  1.00000000 -0.04953215 0.1877534  0.6079515
x2 -0.04953215  1.00000000 0.5194100 -0.1488812
x3  0.18775342  0.51940999 1.0000000  0.6434430
y   0.60795153 -0.14888117 0.6434430  1.0000000
```

We see that there is some correlation between pairs of the explanatory variables, and we know there is collinearity because x_3 is predictable from x_1 and x_2.

Let's look for evidence of instability of the coefficients. First, we fit the multiple linear regression model with all three explanatory variables included:

```
m_collinear_123 <- lm(y ~ x1 + x2 + x3, data = data_collinear)
summary(m_collinear_123)$coefficients
```

```
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept)  4.9806739 0.10734366  46.39933 1.490805e-67
x1           1.4675042 0.11972274  12.25752 2.384482e-21
x2          -1.9193496 0.12990373 -14.77517 1.832330e-26
x3           0.4885226 0.02244616  21.76419 6.723265e-39
```

Do these change much if we fit the model with only x_1 and x_2 included?

```
m_collinear_12 <- lm(y ~ x1 + x2, data = data_collinear)
summary(m_collinear_12)$coefficients
```

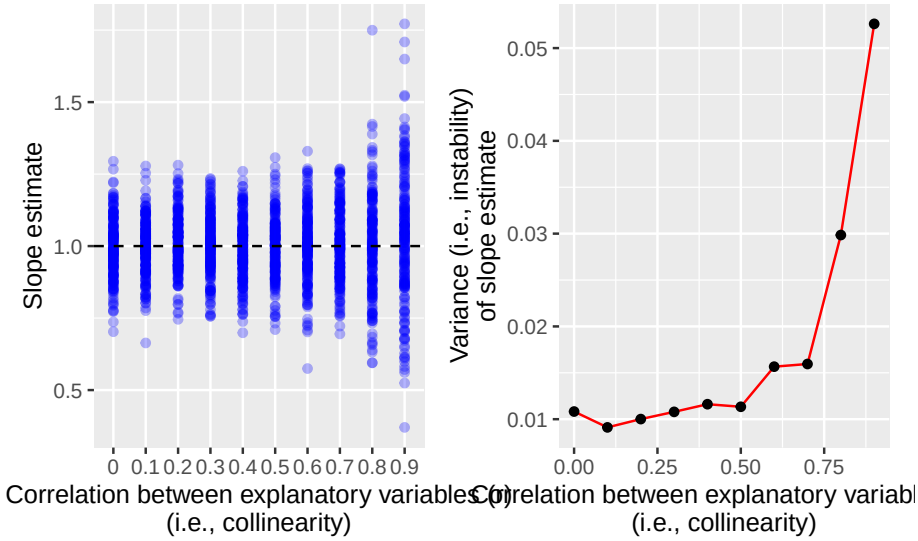
```
             Estimate Std. Error   t value     Pr(>|t|)
(Intercept)  5.3105865  0.2575325 20.621034 3.113931e-37
x1           2.1192629  0.2809162  7.544111 2.464756e-11
x2          -0.3956201  0.2651792 -1.491897 1.389714e-01
```

Yes, they do change quite a bit. The coefficients for x_1 and x_2 are quite different when x_3 is included in the model compared to when it is not included. This is because x_3 is predictable from x_1 and x_2, so including x_3 in the model changes the interpretation of the coefficients for x_1 and x_2.

> **Note**
>
> In a not so extreme case such as this one, the coefficients will not be completely unstable (i.e., they will not change from a number to NA) but they can still change quite a bit depending on which other collinear variables are included in the model.

The bottom line is that collinearity between explanatory variables complicates the interpretation of the model coefficients. If there is collinearity/correlation between the explanatory variables, then the model coefficients can be unstable and difficult to interpret.

Let's have one more look at this instability that is caused by collinearity. And make this demonstration a bit more general. We'll simulate data with two explanatory variables that are correlated with each other to varying degrees. We'll then fit multiple linear regression models with both explanatory variables included, and see how the stability of the coefficients changes as we change the correlation between the explanatory variables.



In the left panel we see the slope estimates for x_1 from the multiple linear regression model with both x_1 and x_2 included, for different levels of correlation between x_1 and x_2. When there is no correlation between x_1 and x_2 (i.e., no collinearity), the slope estimates are quite stable and close to the true value of 1.

As the correlation between x_1 and x_2 increases, the slope estimates become more variable (= less stable). At correlation of 0.9, the slope estimates are quite variable and can be very far from the true value of 1.

The right panel shows the variance of the slope estimates as a function of the correlation between x_1 and x_2. Variance here is a measure of the amount of vertical spread in the left panel.

We see that the variance of the slope estimates increases as the correlation between x_1 and x_2 increases. This shows that greater collinearity between explanatory variables leads to greater instability (variance) of the slope estimates.

This increase in variance caused by collinearity is known as the **variance inflation effect**. The variance inflation effect makes it difficult to interpret the coefficients of the model, because the coefficients can be quite unstable and can change a lot depending on which other collinear variables are included in the model.

Collinearity and interpretation of R^2

Collinearity also affects the interpretation of the R^2 values. Collinearity will cause the collinear explanatory variables to share some of the explained variance. The R^2 value of the multiple regression will then be less than the sum of the

R^2 values of the individual regressions of the response variable on each of the explanatory variables separately.

Let's use a version of the blood pressure age data with strong (but not perfect) collinearity between age and minutes of exercise:

Read in the data:

```
data_collinear_strong <- read_csv("datasets/data_collinear_strong.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): age, mins_exercise, bp

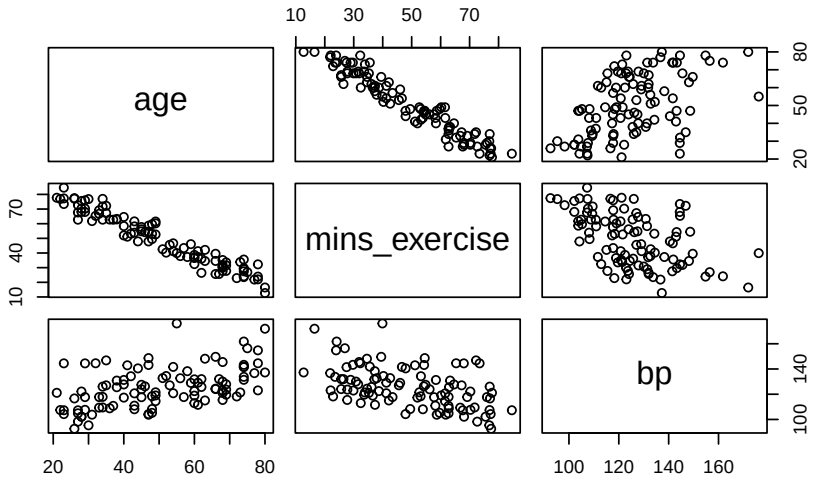
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Look at correlations:

```
cor(data_collinear_strong)
```

```
                    age mins_exercise         bp
age            1.0000000    -0.9630093  0.4998038
mins_exercise -0.9630093     1.0000000 -0.5018301
bp             0.4998038    -0.5018301  1.0000000
```

```
pairs(data_collinear_strong)
```



Fit the two models with one explanatory variable each, and the model with both:

```
m2_age <- lm(bp ~ age, data = data_collinear_strong)
m2_mins_exercise <- lm(bp ~ mins_exercise, data = data_collinear_strong)
m2_both <- lm(bp ~ age + mins_exercise, data = data_collinear_strong)
```

R^2 of the age only model:

```
summary(m2_age)$r.squared
```

```
[1] 0.2498038
```

R^2 of the mins_exercise only model:

```
summary(m2_mins_exercise)$r.squared
```

```
[1] 0.2518335
```

R^2 of the model with both age and mins_exercise:

```
summary(m2_both)$r.squared
```

```
[1] 0.2555995
```

In this case the two explanatory variables are strongly correlated and so share a lot of the explained variance. The R^2 value of the model with both explanatory variables is much less than the sum of the R^2 values of the models with each explanatory variable separately. In fact, either of the models with only one explanatory variable is nearly as good as the model with both explanatory variables. We don't gain much from including another explanatory variable in the model when we already include one explanatory variable that is strongly correlated with the other.

Do I have a problem (with collinearity)?

There are several ways to measuring collinearity between explanatory variables and to assess if it is a problematic. One way is to look at the correlation matrix of the explanatory variables. If there are strong correlations between any pair of explanatory variables, then there is likely to be collinearity.

But recall that collinearity can also occur without strong pairwise correlations. So another way to detect collinearity is to calculate the Variance Inflation Factor (VIF). This is so named because it measures how much the variance of the estimated regression coefficients is increased due to collinearity. (Recall that we saw the variance inflation effect in the simulation above.)

Recall the definition of collinearity: it is when an explanatory variable is predictable from a linear combination of others. To calculate the VIF for a specific explanatory variable, we fit a linear regression model with that explanatory variable as the response variable, and all the other explanatory variables as the explanatory variables. We then calculate the R^2 value for this model. The VIF is then calculated as:

$$VIF_j = \frac{1}{1 - R_j^2}$$

where R_j^2 is the R^2 value from the model with explanatory variable j as the response variable, and all other explanatory variables as the explanatory variables.

In the case where the explanatory variable j is not predictable from the other explanatory variables at all, then $R_j^2 = 0$, and the VIF is 1. This indicates that there is no collinearity.

In the case where the explanatory variable j is perfectly predictable from the other explanatory variables, then $R_j^2 = 1$, and the VIF is infinite. This indicates that there is perfect collinearity.

To get the VIF for a multiple linear regression model with multiple explanatory variables, we calculate the VIF for each explanatory variable separately.

The VIF measures how much the variance of the estimated regression coefficients is increased due to collinearity.

In R, we can calculate the VIF using the `vif()` function from the `car` package. Here is the code to calculate the VIF for the blood pressure model:

```
vif(m_bp_age_exercise)
```

```
          age mins_exercise
     1.050981      1.050981
```

We see that the VIF values for both explanatory variables are close to 1, indicating that there is no collinearity between the explanatory variables.

For the previous example with collinearity among three explanatory variables, we can calculate the VIF as follows:

```
vif(m_collinear_123)
```

```
      x1       x2       x3
1.069365 1.412832 1.460863
```

We see that the VIF values for all three explanatory variables are greater than 1, indicating that there is collinearity between the explanatory variables. The VIF for x_3 is highest, indicating that x_3 is predictable from x_1 and x_2. But none of the VIF values are extremely high, indicating that the collinearity is not severe.

A VIF value greater than 5 or 10 is often used as a rule of thumb to indicate that there is collinearity between the explanatory variables.

Let's make an example of three explanatory variables with more severe collinearity:

```
set.seed(123)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- 2 * x1 + 3 * x2 + rnorm(n, 0, 1)  # less noise, more collinearity
y <- 5 + 1.5 * x1 - 2 * x2 + 0.5 * x3 + rnorm(n, 0, 1)
data_collinear_severe <- data.frame(x1 = x1, x2 = x2, x3 = x3, y = y)
```

This dataset is also available as `data_collinear_severe` in the book dataset collection:

```
data_collinear_severe <- read_csv("datasets/data_collinear_severe.csv")
```

```
Rows: 100 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (4): x1, x2, x3, y

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
m_collinear_severe <- lm(y ~ x1 + x2 + x3, data = data_collinear_severe)
vif(m_collinear_severe)
```

```
       x1        x2        x3
 4.277416 10.644760 13.360572
```

We see that the VIF values of x_2 and x_3 are greater than 10, indicating that there is strong collinearity between the explanatory variables. The coefficients for x_2 and x_3 will be quite unstable and difficult to interpret.

What to do about collinearity?

Imagine we see high VIFs, we can conclud that there is collinearity between explanatory variables, and that the coefficients for those variables are likely to be unstable and difficult to interpret. What can we do???

The answer is we should *think*!

- Are explanatory variables measuring the same concept?
- Is this collinearity expected from the study design?
- Do I care about interpretation, or am I only interested in prediction?

And then consider some possible remedies:

- Combine explanatory variables (index, PCA, biologically meaningful composite) (covered later in the course).
- Look at the explanatory power of a variable once all other variables are accounted for (see below).

- Remove one of a set of redundant explanatory variables (not covered in this course).
- Use regularization if prediction is the goal (not covered in this course).

Assessing the importance of an explanatory variables in the presence of collinearity

Imagine that we want to know the importance of an explanatory variable in the presence of collinearity. For example, we might want to know how important age is in explaining blood pressure. But we also know that we have collinearity with other explanatory variables, such as minutes of exercise. How can we assess the importance of a particular explanatory variable in the presence of collinearity? Let us call the explanatory variable of interest the "focal" explanatory variable.

One way to assess the importance of the focal explanatory variable is to compare two models. Both should have all other explanatory variables included, but one model should have the focal explanatory variable included, and the other model should have the focal explanatory variable removed. We can then make an F-test to compare the two models. The null hypothesis is that the focal explanatory variable does not explain any additional variance in the response variable, once all other explanatory variables are accounted for. The alternative hypothesis is that the focal explanatory variable does explain additional variance in the response variable, once all other explanatory variables are accounted for.

The question is essentially: does including the focal explanatory variable improve the model fit, once all other explanatory variables are already accounted for?

Here is an example of how to do this in R. Lets use the data with three explanatory variables, x_1, x_2, and x_3, where x_3 is predictable from a linear combination of x_1 and x_2 plus some random noise. We will assess the importance of x_1 in explaining the response variable y, once x_2 and x_3 are accounted for. First, we fit the full model with all three explanatory variables included:

```
m_full <- lm(y ~ x1 + x2 + x3, data = data_collinear)
```

Then, we fit the reduced model with x_1 removed:

```
m_reduced <- lm(y ~ x2 + x3, data = data_collinear)
```

Now, we can use the `anova()` function to compare the two models:

```
anova(m_reduced, m_full)
```

```
Analysis of Variance Table

Model 1: y ~ x2 + x3
Model 2: y ~ x1 + x2 + x3
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     97 272.27
```

```
2     96 106.14  1    166.12 150.25 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the R output we see information about two models.

- **Model 1** is the reduced model with x_1 removed.
- **Model 2** is the second model is the full model with all three explanatory variables included.

Then we have a table that looks somewhat like an ANOVA table. The first column has no title and contain 1 and 2. This indicates the two models being compared.

The second column is **Res.Df**, which is the residual degrees of freedom for each model. We can see that the residual degrees of freedom for Model 2 is one less than that for Model 1, because Model 2 has one more explanatory variable (i.e., x_1) than Model 1.

The third column is "RSS", which is the Residual Sum of Squares for each model. The residual sum of squares for Model 2 is less than that for Model 1, because Model 2 has one more explanatory variable (i.e., x_1) than Model 1, and because that explanatory variable helps to explain some of the variance in the response variable.

The fourth column is "Df", which is the difference in degrees of freedom between the two models. This is one in this case, because Model 2 has one more explanatory variable than Model 1.

The fifth column is "Sum of Sq", which is the difference in residual sum of squares between the two models. This the amount of variance in the response variable that is explained by the focal explanatory variable (i.e., x_1) once all other explanatory variables are already accounted for.

The sixth column is "F value", which is the F-statistic for the comparison of the two models. This is calculated as:

$$F = \frac{(RSS_1 - RSS_2)/(df_1 - df_2)}{RSS_2/df_2}$$

where RSS_1 and RSS_2 are the residual sum of squares for Model 1 and Model 2, respectively, and df_1 and df_2 are the residual degrees of freedom for Model 1 and Model 2, respectively.

Put another way, the numerator is the mean square difference between the two models, and the denominator is the mean square error for the full model.

In our example data, the numbers are: 272.3 - 106.1 = 166.1 divided by 97 - 96 = 1 for the numerator.

And for the denominator: 106.1 divided by 96.

This gives an F value of 150.25. This is a very large F value, indicating that the focal explanatory variable (i.e., x_1) explains a significant amount of variance in the response variable, even once all other explanatory variables are already accounted for. This means that even though there is collinearity between the explanatory variables, we can still find that x_1 is important in explaining the response variable.

We can also find the partial R^2 value for the focal explanatory variable (i.e., x_1) from this analysis. The partial R^2 value is the proportion of variance in the response variable that is explained by the focal explanatory variable, as a fraction of the variance that is not explained by the other explanatory variables.

The partial R^2 is calculated as:

$$R^2_{partial} = \frac{RSS_1 - RSS_2}{RSS_1}$$

We can find this in R with:

```
rss1 <- anova(m_reduced, m_full)$"RSS"[1]
rss2 <- anova(m_reduced, m_full)$"RSS"[2]
partial_r2 <- (rss1 - rss2) / rss1
partial_r2
```

```
[1] 0.6101473
```

Don't confuse this partial R^2 with what is sometimes known as the semi-partial R^2. The semi-partial R^2 is the proportion of variance in the response variable that is explained by the focal explanatory variable, as a fraction of the total variance in the response variable. The semi-partial R^2 is calculated as:

$$R^2_{semi-partial} = \frac{RSS_1 - RSS_2}{TSS}$$

where TSS is the total sum of squares of the response variable.

In R in our example this is:

```
tss <- sum((data_collinear$y - mean(data_collinear$y))^2)
semi_partial_r2 <- (rss1 - rss2) / tss
semi_partial_r2
```

```
[1] 0.1625303
```

This means that about 16.25% of the total variance in the response variable is explained by the focal explanatory variable (i.e., x_1), even once all other explanatory variables are already accounted for. This is explanatory power that is unique to the focal explanatory variable.

Confounding: why we include other explanatory variables

In Chapter 4 we saw that an association between two variables in an observational study does not show that one causes the other, because of possible **confounding variables**: variables associated with both the explanatory variable and the response. Multiple regression gives us one way to deal with confounding variables that we have measured: include them as explanatory variables.

Here is an example. Imagine a survey of 100 gardens, in which we measured the height of a particular crop plant (cm), the amount of fertiliser the gardener applies (g per week) and the amount of watering (litres per week). Gardeners who use more fertiliser also tend to water more. In these simulated data, fertiliser truly increases height by 1 cm per g, and water by 3 cm per litre:

```
set.seed(2027)
n <- 100
water <- runif(n, 0, 10)
fertiliser <- 2 + 0.8 * water + rnorm(n, 0, 1.5)
height <- 20 + 1 * fertiliser + 3 * water + rnorm(n, 0, 4)
garden_survey <- tibble(water = water, fertiliser = fertiliser, height = height)
```

```
garden_survey <- read_csv("datasets/garden_survey.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): water, fertiliser, height

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

If we look only at fertiliser, we get:

```
m_fertiliser_only <- lm(height ~ fertiliser, data = garden_survey)
coef(m_fertiliser_only)
```

```
(Intercept)  fertiliser
  20.590578    3.502984
```

The estimated effect of fertiliser is 3.5 cm per g, much larger than the true value of 1. This is because gardens with more fertiliser also receive more water, so in this model fertiliser is "taking the credit" for some of the effect of water. Water is a confounding variable.

If we include water in the model:

```
m_fertiliser_water <- lm(height ~ fertiliser + water, data = garden_survey)
coef(m_fertiliser_water)
```

```
(Intercept)  fertiliser       water
 19.6733452   0.8867562   3.1165654
```

```
confint(m_fertiliser_water)
```

```
                2.5 %    97.5 %
(Intercept) 17.7850212 21.561669
fertiliser   0.3734932  1.400019
water        2.5972822  3.635849
```

the estimated effect of fertiliser, with the amount of water held constant, is 0.89 cm per g, close to the true value.

Some important points:

- **We can only adjust for confounders we have measured.** If an unmeasured variable (for example, soil quality) is associated with both fertiliser use and plant height, the estimate will still be biased, and the model cannot warn us. This is why observational studies, even with multiple regression, cannot prove causation.
- **Adjusting has a cost.** Because the confounder is correlated with the explanatory variable of interest, including it introduces collinearity, which makes the estimates less precise. Here the standard error of the fertiliser effect increases from 0.21 to 0.26 when water is included. This is a price worth paying to remove the bias.
- **Choose the variables in advance.** Which potential confounders to include should be decided using biological knowledge when planning the study and analysis, not by trying different combinations until a result looks interesting.
- **Experiments avoid the problem.** If we had instead done an experiment and *randomly assigned* the amount of fertiliser to the gardens, fertiliser would be unrelated to watering (and to soil quality, and anything else). A simple regression of height on fertiliser would then give an unbiased estimate of its effect. This is why randomised experiments give the strongest evidence for causal effects.

Review

Simple regression:

- How well does the model describe the data: Correlation and R^2
- Are the parameter estimates compatible with some specific value (t-test)?
- What range of parameters values are compatible with the data (confidence intervals)?
- What regression lines are compatible with the data (confidence band)?
- What are plausible values of other data (prediction band)?

Multiple regression:

- Multiple linear regression x_1, x_2, ..., x_m
- Checking assumptions.
- Question 1: As an ensemble, do the explanatory variables explain variation in the response variable? This is done with an overall F-test.
- Question 2: Which of the explanatory variables are important in explaining variation in the response variable? This is done by looking at the coefficients, t-tests, and confidence intervals for each explanatory variable.
- Question 3: How well does the model describe the data? This is done with R^2 and adjusted R^2.
- Question 4: Are some explanatory variables more important than others in explaining variation in the response variable? This is done by looking at the sizes of the coefficients, and comparing those for different explanatory variables. But caution is required.
- Question 5: How do we make predictions from the model? This is done with the `predict()` function, and often requires fixing the values of some explanatory variables to specific values.
- Collinearity between explanatory variables can make the coefficients unstable and difficult to interpret. This can be assessed with the Variance Inflation Factor (VIF). Remedies include combining explanatory variables, removing redundant predictors, or using regularization if prediction is the goal. The importance of an explanatory variable in the presence of collinearity can be assessed by comparing models with and without that variable, using an F-test.

Further reading

If you'd like to solidify your understanding of multiple linear regression, you might like to look at Chapter 11 of **Statistics. An Introduction using R** by Crawley. This section is about 14 pages. You will also see that the chapter includes material not included in this course, such as interactions, polynomial regression, model selection, and regression trees. Fun stuff (of course it will not be in the examination for this course!).

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

3D plot of multiple linear regression

3D plots can help us to visualise multiple linear regression models with two explanatory variables. They are also kind of cool. They can also be difficult to

interpret. So use with caution!

Here is the code to make a 3D plot of the blood pressure data. The y-axis is blood pressure, the x-axis is age, and the z-axis is minutes of exercise. Here is a 3d plot that we can interactive with and rotate (this interactive figure is shown only in the online, HTML, version of the book):

```
plotly::plot_ly(bp_data_multreg, x = ~age, y = ~mins_exercise, z = ~bp,
                type = "scatter3d", mode = "markers", marker = list(size = 5),
                width=500, height=500) |>
  plotly::layout(scene = list(xaxis = list(title = "Age"),
                              yaxis = list(title = "Minutes of exercise"),
                              zaxis = list(title = "Blood pressure")))
```

This interactive 3-D figure is shown only in the online version of the book: see this chapter online.

Publication ready table of results

Sometimes we need to put a table of coefficients and confidence intervals into a report or publication. One option is to make a table in Word, and to manually enter the coefficients and confidence intervals. This is tedious and error prone. A much better option is to use R to make the table for us. One way to do this is to use the `broom` package to tidy up the model output into a data frame, and then use the `knitr` package to make a table from the data frame.

```
tidy_m_bp_age_exercise <- tidy(m_bp_age_exercise, conf.int = TRUE)
kable(tidy_m_bp_age_exercise, digits = 3, caption = "Multiple linear regression results
for blood pressure data")
```

Table 2: Multiple linear regression results for blood pressure data

term	estimate	std.error	statistic	p.value	conf.low	conf.high
(Intercept)	84.746	3.020	28.062	0	78.752	90.740
age	0.553	0.053	10.331	0	0.446	0.659
mins_exercise	-0.097	0.011	-8.889	0	-0.118	-0.075

Another approach is to use `tbl_regression` function within the `gtsummary` package to get a publication ready table of the coefficients and confidence intervals:

```
tbl_regression(m_bp_age_exercise)
```

To use either approach most efficiently you will, however, need to write your report using R Markdown or Quarto so that the table is automatically created

Characteristic	Beta	95% CI	p-value
age	0.55	0.45, 0.66	<0.001
mins_exercise	-0.10	-0.12, -0.07	<0.001

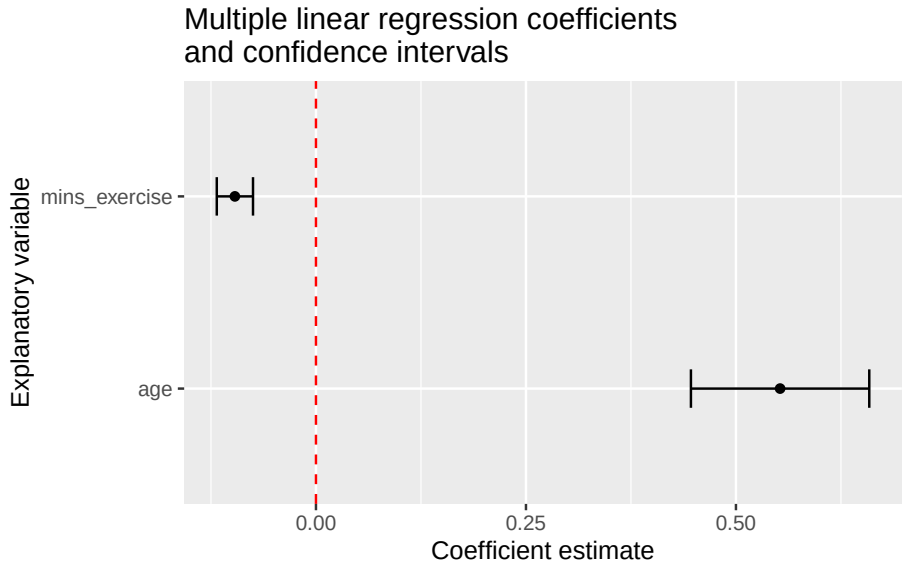
Abbreviation: CI = Confidence Interval

in your report document.

A figure showing the coefficients and confidence intervals

As well as or rather than a table of coefficients, we could make a figure showing the coefficients and confidence intervals for each explanatory variable. This can be a nice way to visualise the results of the multiple linear regression model.

```
tidy_m_bp_age_exercise <- tidy(m_bp_age_exercise, conf.int = TRUE)
tidy_m_bp_age_exercise |>
  filter(term != "(Intercept)") |>
ggplot(aes(x = term, y = estimate)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high), width =
                0.2) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(x = "Explanatory variable", y = "Coefficient estimate") +
  ggtitle("Multiple linear regression coefficients\nand confidence intervals") +
  coord_flip()
```



Interactions (L7)

Learning objectives

After this unit, students will be able to:

1. Explain what an interaction between two explanatory variables means, both biologically and in the model equation.
2. Distinguish additive from interactive effects using graphs of data or of model predictions.
3. Explain why one model with several explanatory variables (and their interaction) is preferable to several separate single-variable analyses.
4. Use R to fit models with interactions between two categorical variables (two-way ANOVA), a categorical and a continuous variable (ANCOVA), and two continuous variables.
5. Interpret interaction coefficients, including how the meaning of the main-effect coefficients changes when an interaction is in the model.
6. Use R to calculate and plot how the effect of one explanatory variable depends on the value of another.
7. Test whether an interaction is needed, and judge whether it is biologically meaningful.
8. Calculate the degrees of freedom of main effects and interaction terms, and the residual degrees of freedom, from the number of parameters a model estimates.
9. Recognise a correct and complete statement of results from a model with an interaction.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch07_interactions.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 7 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:
`download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough`
`destfile = "walkthrough_ch07_interactions.R")`
Video walkthrough: coming soon.

Thought for the week

"*You should arrive at answers on your own, and not rely upon what you get from someone else. Answers from others are nothing more than stopgap measures, they're of no value.*"

- Ichiro Kishimi & Fumitake Koga, The Courage to Be Disliked

They instead encourage answers gained from curiosity, observation, and dialogue. Answers you develop for yourself are more likely to stick with you, and to be meaningful.

When we mentor, ask questions that encourage others to think for themselves, rather than simply giving answers. This is hard, but worthwhile.

Introduction

In the previous chapters we have looked at models with a special type of simplicity: the effect of each explanatory variable on the response variable is independent of the other explanatory variables. This is special because it means that we can understand the effect of each explanatory variable on the response variable separately, without considering the other explanatory variables. Nevertheless, this is often not the case in real life. The effect of one explanatory variable on the response variable may depend on the value of another explanatory variable. This is called an interaction effect, or simply an interaction.

Interactions are some of the most interesting phenomena in science, including biology. We are not talking about interactions between species, like predation, though these are also very interesting. We are talking about effects of one thing, like diet, depending on another thing, like exercise. Let's break that down a bit with an example.

Example 1 (1 cat, 1 cont)

Imagine we make a study of the effect of exercise (minutes per week) on blood pressure for people with a meat heavy diet.

Read in the dataset:

```
bp_meatheavy <- read_csv("datasets/bp_meatheavy.csv")
```

```
Rows: 50 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): diet
dbl (2): bp, mins_per_week

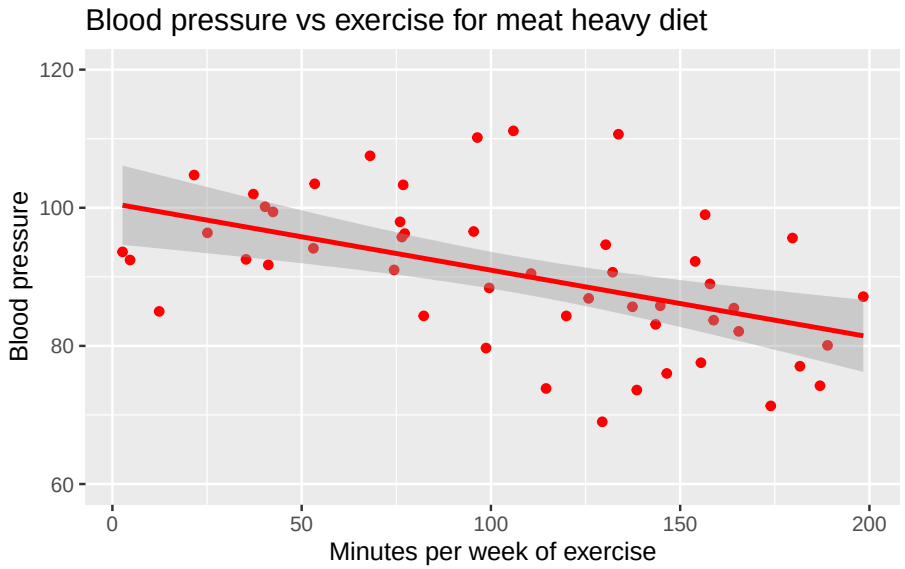
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the data:

```
head(bp_meatheavy)
```

```
# A tibble: 6 x 3
     bp mins_per_week diet
  <dbl>         <dbl> <chr>
1  94.1          53.1 meat heavy
2  91.0          74.4 meat heavy
3  73.8         115.  meat heavy
4  77.1         182.  meat heavy
5 100.          40.3 meat heavy
6  95.6         180.  meat heavy
```

And here is a graph of the relationship between blood pressure and minutes of exercise for people with a meat heavy diet:



We see that exercise seems to lower blood pressure. But what if we look at the effect of exercise on blood pressure for people with a vegetarian diet? The relationship might look something like this:

```
         bp mins_per_week       diet
1  77.56704     122.92899 vegetarian
2  84.30684     111.43191 vegetarian
3  75.23323      65.75546 vegetarian
4  85.67402      90.62629 vegetarian
5  77.45327     100.08819 vegetarian
6 102.31114      36.17327 vegetarian
```

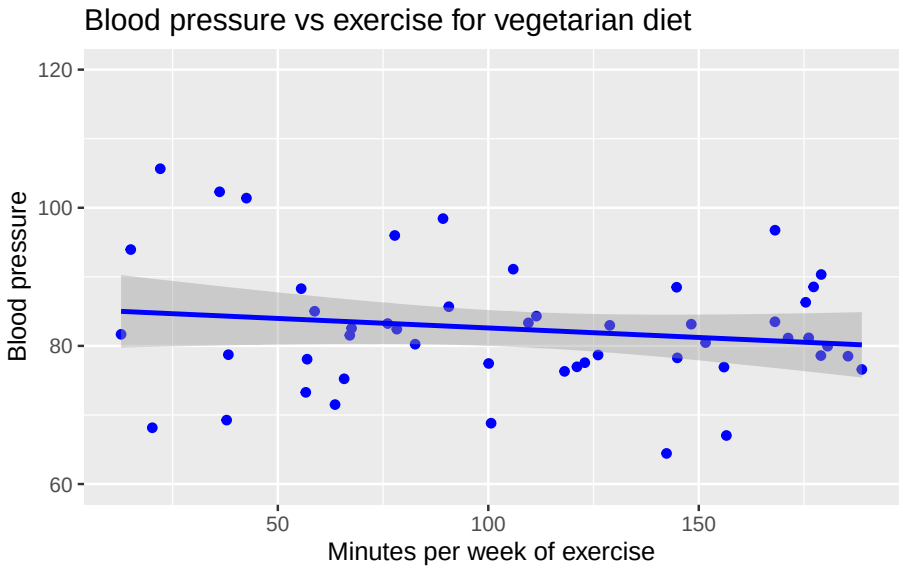
Read in the dataset:

```
bp_vegetarian <- read_csv("datasets/bp_vegetarian.csv")
```

```
Rows: 50 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): diet
dbl (2): bp, mins_per_week

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here is a graph of the relationship between blood pressure and minutes of exercise for people with a vegetarian diet:

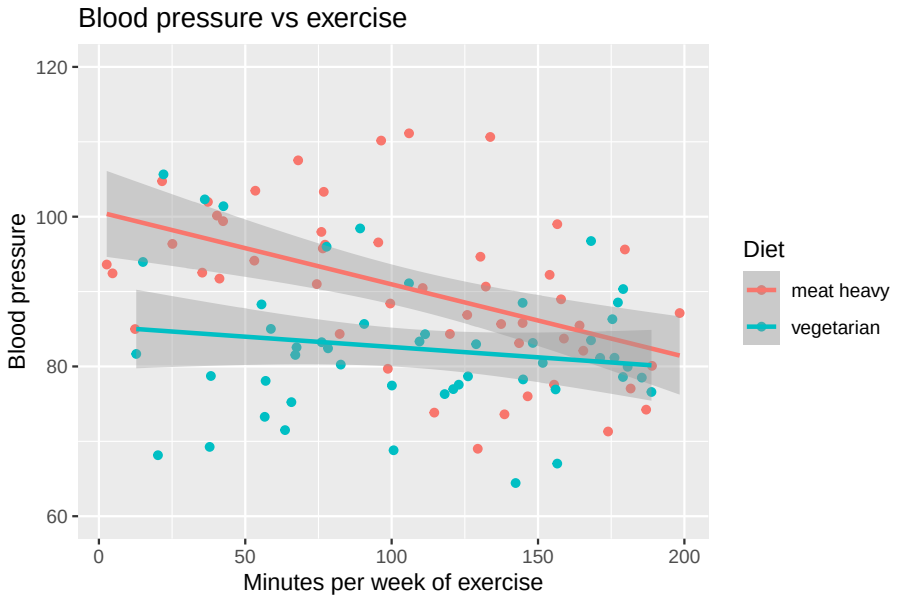


We see that exercise seems to lower blood pressure for vegetarians too, but the effect seems to be weaker.

To summarise this finding, we can say that the effect of exercise on blood pressure is stronger for people with a meat heavy diet than for people with a vegetarian diet. This means that the effect of exercise on blood pressure depends on diet.

This is very clear when we look at the both diets in the same graph. First we combine the two datasets into one, with `bind_rows()`:

```
bp_1cont1cat <- bind_rows(bp_meatheavy, bp_vegetarian)
```



(The dataset with both combined is in the file `bp_1cont1cat.csv`.)

I think it is clear that the interaction was easier to see when we plotted all the data in one graph… it is much easier to visually compare the slopes of the two regression lines when they are on the same graph.

As we will see later in this chapter, the same holds true for statistical tests of interactions: it is much easier to make a statistical test of the interaction when we make a single model with *an interaction term*.

It is harder and is not recommended to make a separate regression for each level of the second variable (diet) and then compare the slopes of the regression lines (it is possible, just not at all efficient).

Parallel and non-parallel effects

In the example above, the effect of exercise on blood pressure was stronger for people with a meat heavy diet than for people with a vegetarian diet. That is, the slope of the regression line was steeper for the meat heavy diet than for the vegetarian diet. Put another way, the regression lines are not parallel.

Parallel = no interaction: Parallel regression lines are evidence of no interaction. This means that the effect of one variable (exercise) is the same for all levels of another variable (diet).

Non-parallel = interaction: When the regression lines are not parallel, there is evidence of an interaction. This means that the effect of one variable (exercise) depends on the level of another variable (diet).

That was an example of an interaction between a continuous explanatory variable (exercise) and a categorical explanatory variable (diet). Interactions can also occur between two categorical explanatory variables, or between two continuous explanatory variables. Let us look at some more examples.

Example 2 (2 categorical)

Two categorical explanatory variables: diet (meat heavy or vegetarian) and exercise (low or high), and one continuous response variable (blood pressure).

Read in the dataset:

```
bp_2cat <- read_csv("datasets/bp_2cat.csv")
```

```
Rows: 40 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): diet, exercise
dbl (2): reps, bp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

R orders the levels of a categorical variable alphabetically, so "high" would come before "low", and "high" would be the reference level. We make `exercise` a factor with the order we want:

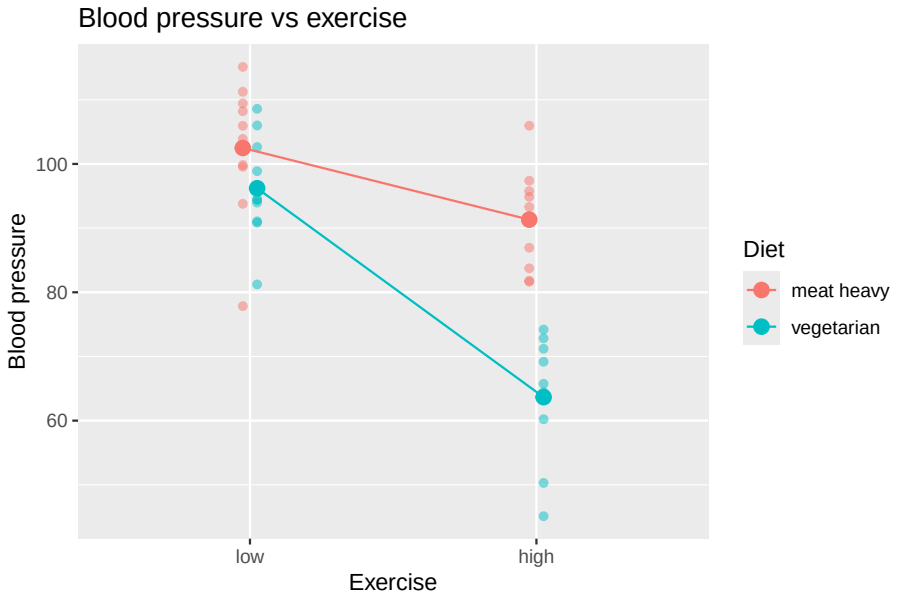
```
bp_2cat <- bp_2cat |>
  mutate(exercise = factor(exercise, levels = c("low", "high")))
```

Here are the first few rows of the data:

```
head(bp_2cat)
```

```
# A tibble: 6 x 4
  diet       exercise  reps    bp
  <chr>      <fct>    <dbl> <dbl>
1 meat heavy high         1  83.7
2 meat heavy low          1 115.
3 vegetarian high         1  74.2
4 vegetarian low          1 109.
5 meat heavy high         2  91.8
6 meat heavy low          2 104.
```

And here is the data in a graph, where we show the individual data points as well as the group means connected by lines. The lines connecting the means are only to help visualize whether there is an interaction or not. If we see non-parallel lines connecting the means, we have evidence of an interaction.



Indeed, the lines connecting the means are not parallel, which is evidence of an interaction between diet and exercise on blood pressure.

Example 3 (two continuous)

Two continuous explanatory variables (age and exercise minutes) and one continuous response variable (blood pressure).

Read in the dataset:

```
bp_2cont <- read_csv("datasets/bp_2cont.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): bp, age, mins_per_week

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

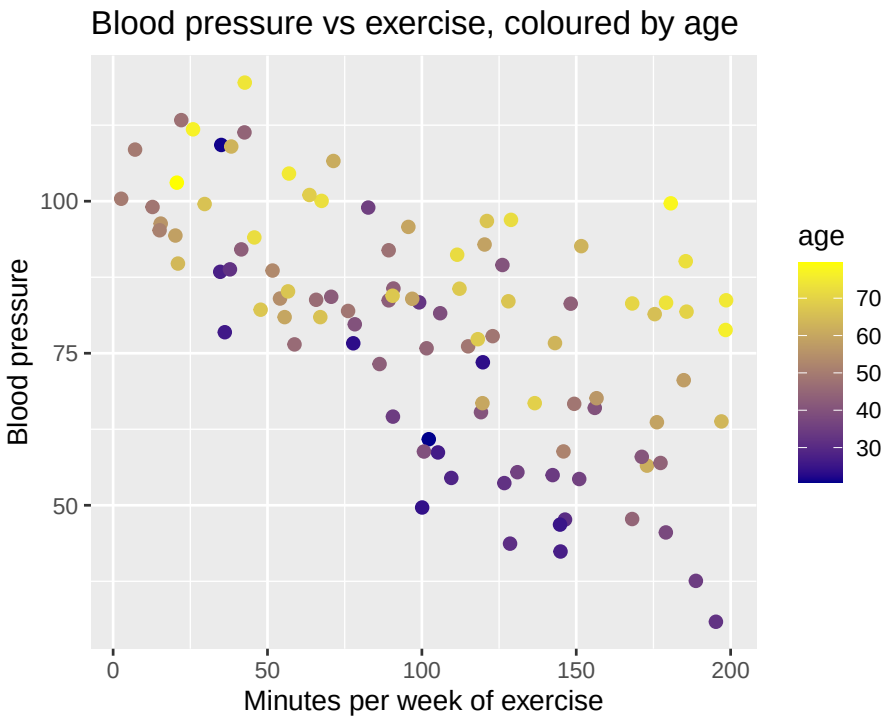
Here is the first few rows of the data:

```
head(bp_2cont)
```

```
# A tibble: 6 x 3
     bp   age mins_per_week
  <dbl> <dbl>         <dbl>
1  55.4  35.9         131.
```

```
2  84.3  42.3          70.6
3  84.0  54.4          54.1
4  83.7  74.5         199.
5  53.7  32.1         127.
6 119.   73.9          42.6
```

Now we have a little challenge, namely that we have two continuous explanatory variables. This means that we need to use three dimensions to visualize the data. Here is a standard 2D scatter plot of blood pressure against minutes of exercise, though with age represented by color grading from young (dark blue) to old (yellow):



And we can make the complementary plot of blood pressure against age, with minutes of exercise represented by color grading from low (green) to high (orange):



What do we see? If we look at a set of points of the similar colour (i.e., similar number of minutes of exercised), we can see that the slope of the relationship between blood pressure and age depends on exercise. The slope is steeper for people that exercise more.

Yet another way to visualise this interaction is to create categorical versions of age and minutes of exercise, and then plot the data with these categorical variables:

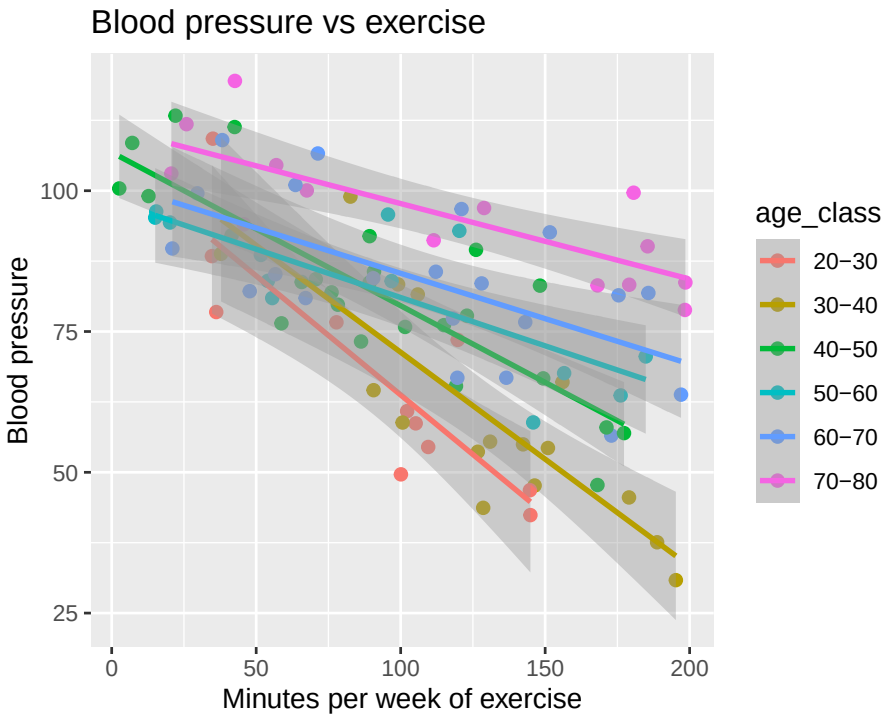
```
# cut() makes a categorical version (classes) of a continuous variable
bp_2cont <- bp_2cont |>
  mutate(age_class = cut(age,
                   breaks = seq(10, 100, 10),
                   labels = c("10-20", "20-30", "30-40",
                              "40-50", "50-60", "60-70",
                              "70-80", "80-90", "90-100")),
         mins_per_week_class = cut(mins_per_week,
                                   breaks = seq(0, 200, 20),
                                   labels = c("0-20", "20-40", "40-60",
                                              "60-80", "80-100", "100-120",
                                              "120-140", "140-160", "160-180",
                                              "180-200")))
```

Here is the first few rows of the modified data:

```
head(bp_2cont)
```

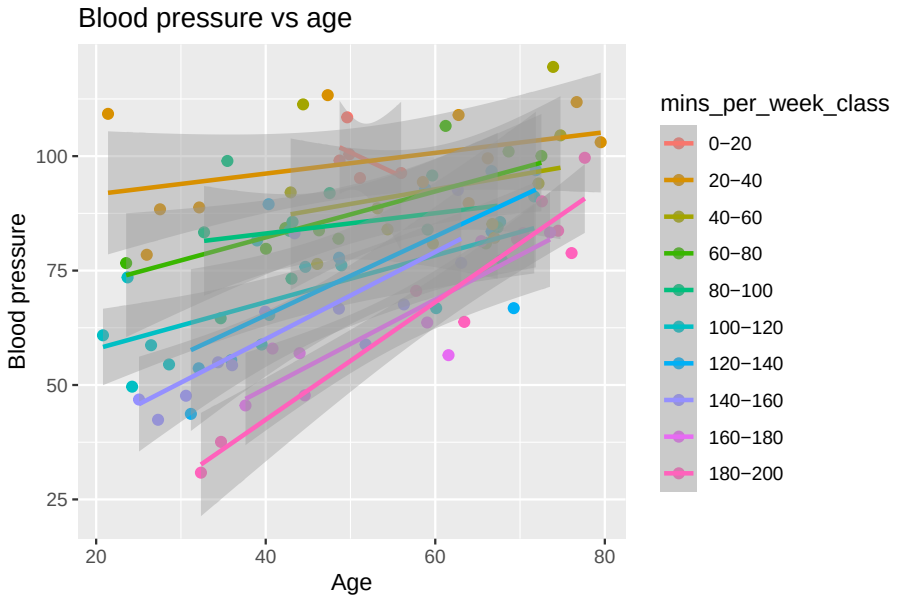
```
# A tibble: 6 x 5
     bp   age mins_per_week age_class mins_per_week_class
  <dbl> <dbl>         <dbl> <fct>     <fct>
1  55.4  35.9         131.  30-40     120-140
2  84.3  42.3          70.6 40-50     60-80
3  84.0  54.4          54.1 50-60     40-60
4  83.7  74.5         199.  70-80     180-200
5  53.7  32.1         127.  30-40     120-140
6 119.   73.9          42.6 70-80     40-60
```

And here is the plot of blood pressure against minutes of exercise, with age class represented by colour. Regression lines have been added, to help visualize any interaction:



It is as we saw in the previous version of the graph. The slope of the relationship between blood pressure and minutes of exercise depends on age. The slope is shallower for older people.

In the complementary graph of blood pressure against age, with minutes of exercise represented by colour, we see the same interaction:



> **!** Important
>
> **There is only one interaction** We made two graphs of the same data, one with age as a categorical variable and one with minutes of exercise as a categorical variable. In both graphs we see an interaction. This is not us seeing two separate interactions, however. There is only one interaction here, namely that the effect of age on blood pressure depends on minutes of exercise, and equivalently, that the effect of minutes of exercise on blood pressure depends on age. The two graphs just look at the same interaction from different perspectives.

Other perspectives

Interactions and additivity of effects

Another way of thinking about interactions is from the perspective of additivity or non-additivity of effects. Imagine we made two separate studies, one of the effect of diet on blood pressure, and one of the effect of exercise on blood pressure. In the first study we only varied diet, and in the second study we only varied exercise. In the first study we found an effect size of diet on blood pressure of say 10 mmHg (e.g., difference between meat heavy and vegetarian). And in the second study we found an effect size of exercise on blood pressure of 15 mmHg (e.g., difference between low and high exercise).

If the effects are non-additive, we would expect the effect size to be different from additive. For example, if we found the combined effect of diet and exercise

on blood pressure to be 40, we would say that the effects are non-additive. Their combined effect is more than the sum of their individual effects. This example is of a synergistic interaction because the combined effect (40) is greater than the sum of the individual effects (25).

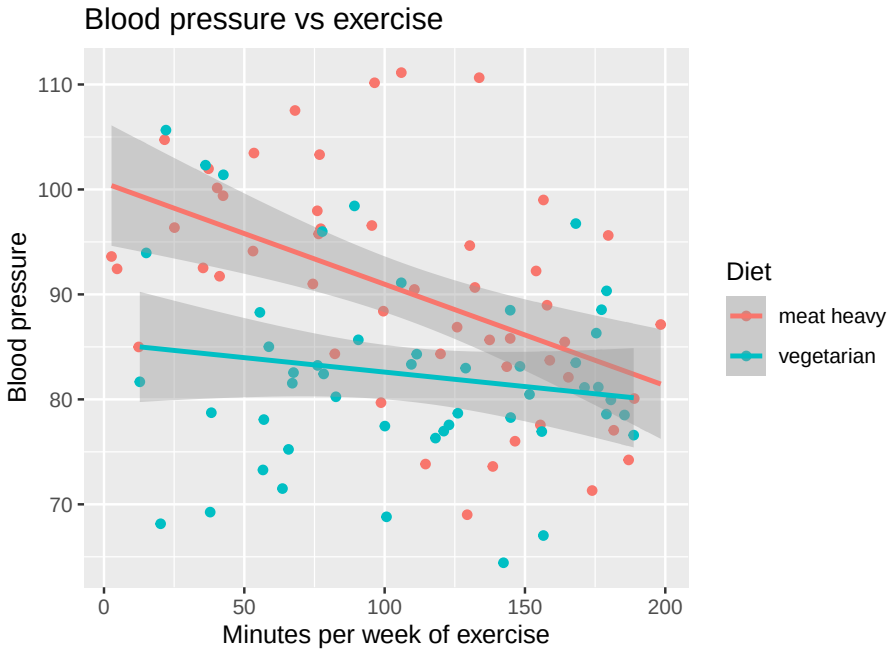
Drug interactions

When a doctor is considering giving us a particular medication, we are asked if we are taking any other medications. This is because the effects of drugs can interact. For example, if we take two drugs that both lower blood pressure and they interfere with each other, the combined effect might be less than the sum of their individual effects. This is an antagonistic interaction. It could be worse than that though, the interaction might actually be harmful, which is why doctors are so careful about drug interactions.

Implementing interactions

The mathematical model

Let us return to Example 1 (1 categorical explanatory variable, and 1 continuous explanatory variable), of the effects of number of minutes of exercise and diet on blood pressure:



We have one continuous explanatory variable (minutes of exercise) and one

binary explanatory variable (diet) and one continuous response variable (blood pressure).

A linear model without interaction would look like this:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \epsilon_i$$

where:

- y_i is the blood pressure of the ith participant
- $x_i^{(1)}$ is the number of minutes of exercise of the ith participant
- $x_i^{(2)}$ is the diet of the ith participant, coded as 0 for "meat heavy" and 1 for "vegetarian"
- β_0 is the intercept
- β_1 is the effect of exercise on blood pressure
- β_2 is the effect of diet on blood pressure
- ϵ_i is the error term for the ith participant.

And here is the model with an interaction term added:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \beta_3 (x_i^{(1)} x_i^{(2)}) + \epsilon_i$$

where:

- $x_i^{(1)} x_i^{(2)}$ is the product of the number of minutes of exercise and the diet of the ith participant.
- β_3 is the coefficient of the interaction term between diet and exercise.

We could also write this model as:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \beta_3 x_i^{(3)} + \epsilon_i$$

where:

- $x_i^{(3)} = x_i^{(1)} x_i^{(2)}$

In this model, the β_3 coefficient represents the interaction effect between diet and exercise on blood pressure. If β_3 is significantly different from 0, we would conclude that there is an interaction between diet and exercise on blood pressure.

Hypothesis testing

If we want to test whether the effect of minutes of exercise on blood pressure is different for people with different diets, we need a null hypothesis to test.

The null hypothesis is that the effect of minutes of exercise on blood pressure is the same for people with different diets. This is a null hypothesis of no

interaction between diet and exercise. In terms of the coefficients of the model, the null hypothesis is that $\beta_3 = 0$.

If we reject the null hypothesis, we conclude that the effect of minutes of exercise on blood pressure is different for people with different diets. This is a non-additive effect.

We can also use the ANOVA table to get an F-test for each of the terms in the model. When our model includes an explanatory variable with an interaction term, the ANOVA table will have a row for the main effect of the variable, a row for the interaction term, and a row for the residuals. The F-test for the interaction term tests the null hypothesis that there is no interaction between the two variables.

If we have one or more explanatory variables with more than two groups (e.g., diet with three levels: meat heavy, vegetarian, vegan), then we need to use the F-test in the ANOVA table rather than the t-test for the coefficients in the model. This is because, just like with ANOVA, the t-test for the coefficients in the model only tests whether the effect of one group is different from the reference group (e.g., vegetarian vs meat heavy), but does not test whether the effects of all groups are different from each other (e.g., vegetarian vs meat heavy vs vegan). The F-test in the ANOVA table tests whether there is a significant difference between any of the groups, which is what we want to know when we have more than two groups.

Doing it in R

Let us fit the model with the interaction term in R. There are two methods to do this and they are equivalent:

```
m_bp_exercise_diet_explicit <- lm(bp ~ mins_per_week + diet + mins_per_week:diet, data=bp_1cont1c
m_bp_exercise_diet <- lm(bp ~ mins_per_week * diet, data=bp_1cont1cat)
```

The second is a shorthand for the first. The `*` operator includes the main effects (main effects are terms in the model that don't include interactions) and the interaction term. The `:` operator includes only the interaction term.

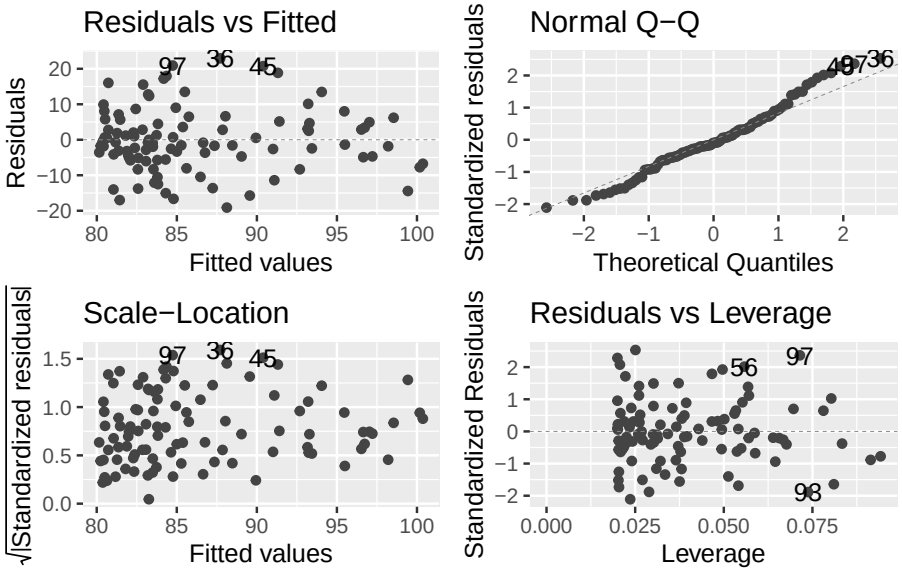
Of course, we check the model assumptions before we interpret the results:

```
autoplot(m_bp_exercise_diet, smooth.colour = NA)
```

```
Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.
i Please use `broom::augment(<lm>)` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



All of the plots look good. Now we can do hypothesis testing of the interaction term using an F-test:

```
anova(m_bp_exercise_diet)
```

```
Analysis of Variance Table

Response: bp
                   Df Sum Sq Mean Sq F value    Pr(>F)
mins_per_week       1 1133.6 1133.55 13.4800 0.0003964 ***
diet                1 1560.8 1560.75 18.5601 3.979e-05 ***
mins_per_week:diet  1  340.4  340.36  4.0475 0.0470408 *
Residuals          96 8072.8   84.09
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the ANOVA table we see four rows. The first row is for the main effect of minutes of exercise, the second row is for the main effect of diet, the third row is for the interaction effect between diet and minutes of exercise, and the fourth row is for the residuals. As always, an interaction term in the R output is shown with a colon `:` between the two variables (here it looks like `mins_per_week:diet`).

In this example, the F-statistic for the interaction term is 4.05 and the p-value is 0.047. The p-value is just below 0.05, so if we use 0.05 as the threshold we reject the null hypothesis that there is no interaction between diet and minutes of exercise on blood pressure. But the evidence is not strong: there is some evidence that the effect of minutes of exercise on blood pressure is different for people with different diets. (With a p-value this close to 0.05, a new study of the same size could quite easily give a p-value above 0.05.)

If we like (and we don't have to), we can look at the coefficients of the model:

```
summary(m_bp_exercise_diet)$coefficients
```

```
                                Estimate Std. Error   t value     Pr(>|t|)
(Intercept)                 100.62716134 2.87228930 35.033783 1.714199e-
56
mins_per_week                -0.09661054 0.02406014 -4.015376 1.177969e-
04
dietvegetarian              -15.27591304 4.09882726 -3.726898 3.275630e-
04
mins_per_week:dietvegetarian  0.06907583 0.03433487  2.011828 4.704076e-
02
```

As expected, there are four coefficients.

The first is `(Intercept)`, which is the expected blood pressure for a person who does 0 minutes of exercise and is on diet "meat heavy".

The second is `mins_per_week`, which is the effect (slope) of minutes of exercise on blood pressure for a person on diet "meat heavy".

The third is `dietvegetarian`, which is the effect of being on a vegetarian diet on blood pressure for a person who does 0 minutes of exercise. This can be thought of as the change in the intercept for a person on a vegetarian diet compared to a person on a "meat heavy" diet.

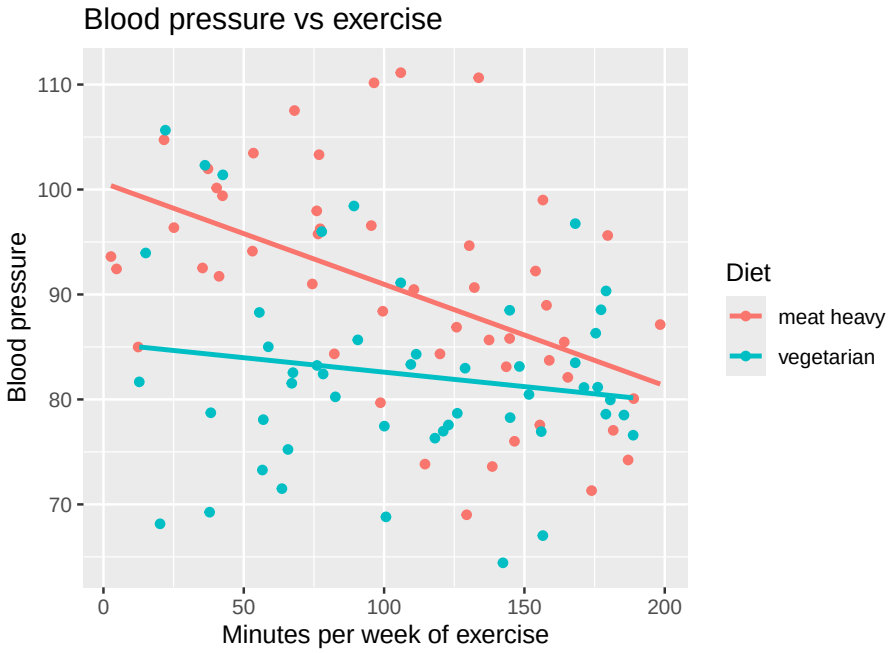
The fourth is the interaction term `mins_per_week:dietvegetarian`, which is the difference in the effect (slope) of minutes of exercise on blood pressure for a person on a vegetarian diet compared to a person on a "meat heavy" diet.

Reporting our findings

Of course a nice graph is always helpful. We already have quite a nice one:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	100.6271613	2.8722893	35.033783	0.0000000
mins_per_week	-0.0966105	0.0240601	-4.015377	0.0001178
dietvegetarian	-15.2759130	4.0988273	-3.726898	0.0003276
mins_per_week:dietvegetarian	0.0690758	0.0343349	2.011828	0.0470408

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
mins_per_week	1	1133.554	1133.55446	13.479979	0.0003964
diet	1	1560.753	1560.75255	18.560124	0.0000398
mins_per_week:diet	1	340.357	340.35702	4.047451	0.0470408
Residuals	96	8072.804	84.09171	NA	NA



We also might want some tables summarizing the model results. Here is a table of the coefficients:

We could also report the R^2 of the model:

```
summary(m_bp_exercise_diet)$r.squared
```

```
[1] 0.2732093
```

And also a table of the variances of the terms in the model:

We also might use a sentence like this to report the results: "The effect of

minutes of exercise is generally negative, and there is some evidence that the effect is stronger for people on a meat heavy diet than for people on a vegetarian diet (F-statistics of interaction term = 4.05, degrees of freedom = 1, degrees of freedom residuals = 96, p-value = 0.047)."

Tip

Report the effect sizes, too. The sentence above tells the reader that the effect of exercise differs between diets, and gives the test of the interaction. It would be more informative to also give the size of the effect of exercise in each diet: the estimated slope of blood pressure against minutes of exercise for each diet, with its 95% confidence interval, preferably expressed over a meaningful amount of exercise (for example, per hour per week). Readers can then judge whether the effects, and the difference between them, are biologically important.

Three models

1. ANCOVA

The example we just worked through (Example 1) was with one continuous explanatory variable (minutes of exercise) and one categorical explanatory variable (diet). This is an example of an analysis of covariance (ANCOVA). ANCOVA is a type of linear model in which there are both continuous and categorical explanatory variables.

ANCOVA is often used in two main ways.

First, it can be used to account for covariates (continuous variables). In this case, the main interest is in comparing groups (as in ANOVA), while one or more continuous variables are included to explain additional variation in the response. These covariates are not the main focus of interpretation; instead, they help adjust group means and improve the precision of group comparisons.

Second, ANCOVA can be used to test whether covariate effects differ between groups. Here, the continuous variable is of real interest, and the question is whether the relationship between the covariate and the response is the same across groups. This is done by including a group $\times$ covariate interaction, which allows the slope of the relationship to differ between groups.

An important distinction is that the first use assumes the covariate has the same effect in all groups (parallel slopes), while the second explicitly tests whether this assumption is valid.

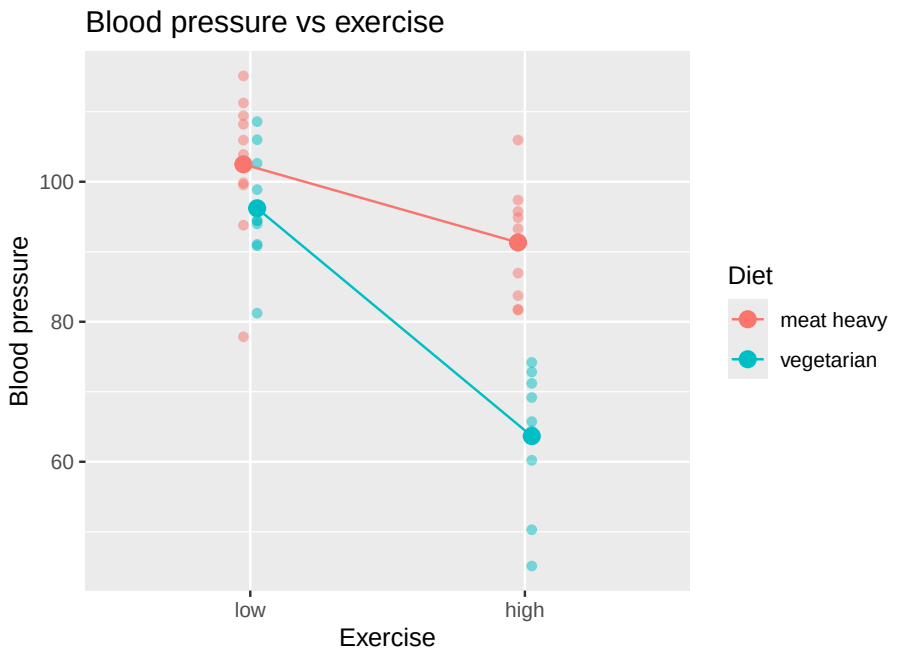
2. Two-way ANOVA

Above we had an example (Example 2) with two categorical explanatory variables (diet [levels: meat heavy or vegetarian] and exercise [levels: low or high]) and one continuous response variable (blood pressure). This is an example of a two-way ANOVA. Two-way ANOVA is used to test for effects of two categorical explanatory variables, as well as their interaction effect on a continuous response variable.

Here are the first few rows of the data again:

```
# A tibble: 6 x 4
  diet       exercise  reps    bp
  <chr>      <fct>    <dbl> <dbl>
1 meat heavy high         1  83.7
2 meat heavy low          1 115.
3 vegetarian high         1  74.2
4 vegetarian low          1 109.
5 meat heavy high         2  91.8
6 meat heavy low          2 104.
```

Here is the graph of the data again:



We can fit a two-way ANOVA model in R as follows:

```
mod_2cat <- lm(bp ~ diet * exercise, data=bp_2cat)
```

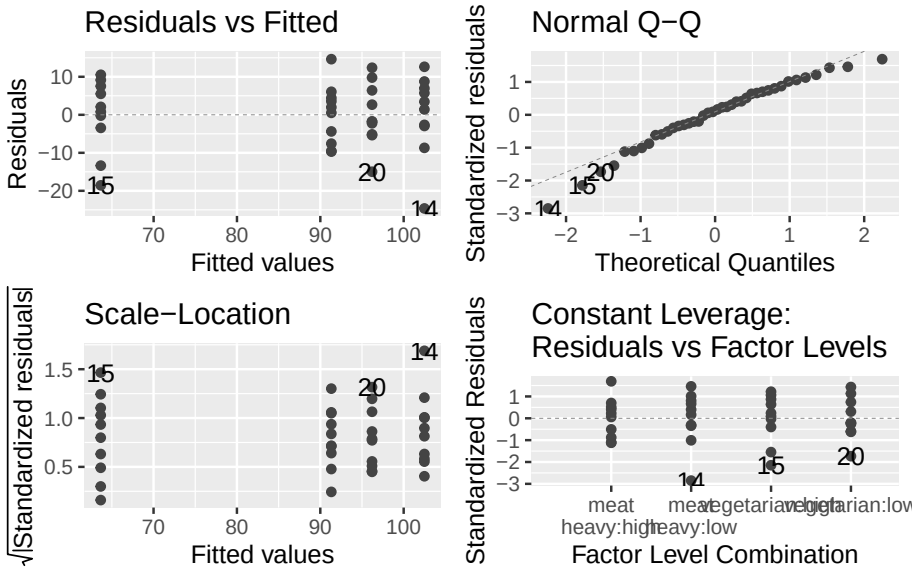
Recall that this fits a model with main effects of diet and exercise, as well as their interaction effect. Recall that we could specify the model equivalently as:

```
mod_2cat <- lm(bp ~ diet + exercise + diet:exercise, data=bp_2cat)
```

We can check the model assumptions:

```
autoplot(mod_2cat, smooth.colour = NA)
```

```
Warning: Removed 40 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 40 rows containing missing values or values outside the scale range
(`geom_line()`).
```



These look ok.

Hypothesis testing the interaction is just as before. We use an F-test on the interaction term to test the null hypothesis that there is no interaction between diet and exercise on blood pressure.

```
anova(mod_2cat)
```

```
Analysis of Variance Table

Response: bp
              Df Sum Sq Mean Sq F value    Pr(>F)
diet           1 2879.8  2879.8  34.695 9.741e-07 ***
exercise       1 4776.5  4776.5  57.546 5.680e-09 ***
diet:exercise  1 1142.5  1142.5  13.765  0.000696 ***
Residuals     36 2988.1    83.0
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the ANOVA table we see four rows. The first row is for the main effect of diet, the second row is for the main effect of exercise, the third row is for the interaction effect between diet and exercise, and the fourth row is for the residuals. As always, an interaction term in the R output is shown with a colon `:` between the two variables (here it looks like `diet:exercise`).

In this example, the F-statistic for the interaction term is quite large (13.76 and the corresponding p-value is quite small (7×10^{-4}. Hence, we conclude that there is a strong evidence of an interaction between diet and exercise on blood pressure. The effect of exercise on blood pressure depends on diet.

> **!** Important
>
> **Reporting two-way ANOVA results** When reporting the results of a two-way ANOVA, it is important to focus on the biological interpretation of the results, rather than the statistical details. Include statistics in parentheses, in support of your statements. For example, you might say something like: "The effect of exercise was different for people on different diets, with a stronger effect for those on a vegetarian diet compared to those on a meat heavy diet ($F(1, 36) = 13.77$, $p < 0.001$)."

More than two levels

What if we had a categorical explanatory variable with more than two levels? For example, what if diet had three levels: meat heavy, vegetarian, and vegan?

Here is an example dataset.

Read in the dataset:

```
bp_2cat_3levels <- read_csv("datasets/bp_3cat.csv") |>
  mutate(exercise = factor(exercise, levels = c("low", "high")))
```

```
Rows: 60 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): diet, exercise
dbl (2): reps, bp

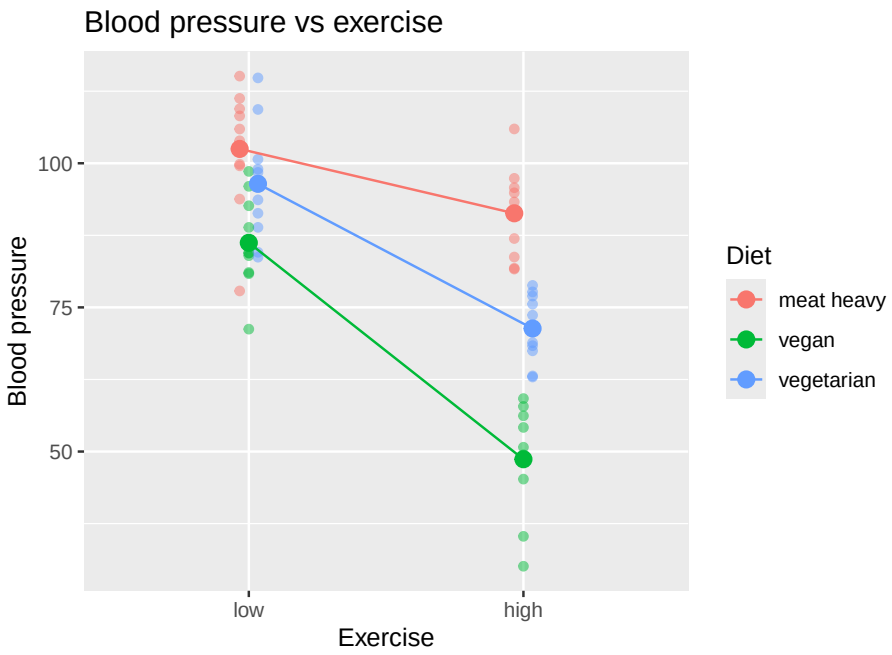
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the data:

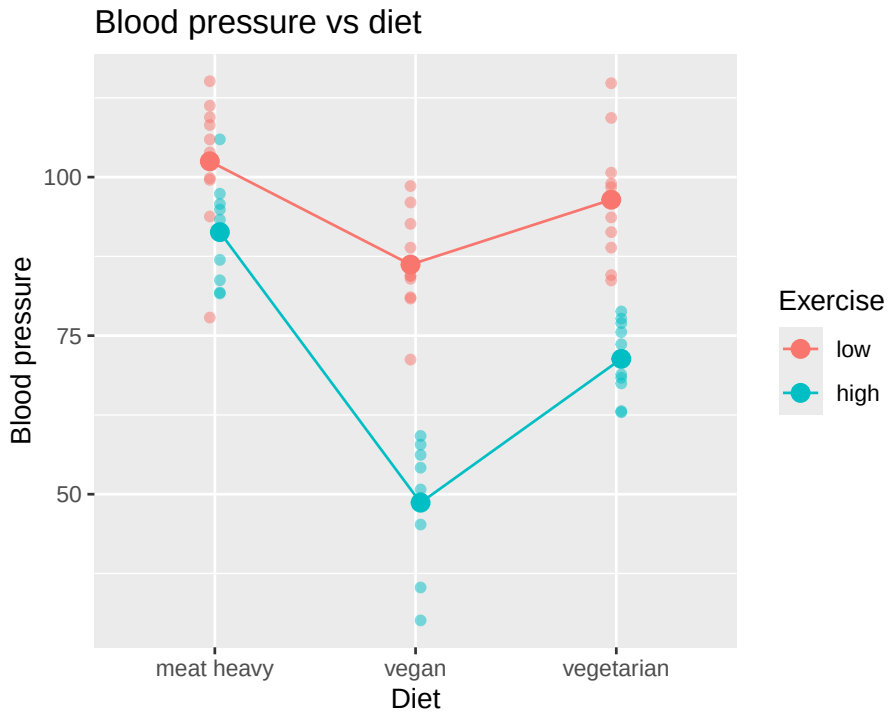
```
head(bp_2cat_3levels)
```

```
# A tibble: 6 x 4
  diet       exercise  reps    bp
  <chr>      <fct>    <dbl> <dbl>
1 meat heavy high         1  83.7
2 meat heavy low          1 115.
3 vegan      high         1  59.2
4 vegan      low          1  98.6
5 vegetarian high         1  68.4
6 vegetarian low          1  99.0
```

Here is the graph of the data:



Or plotted differently:



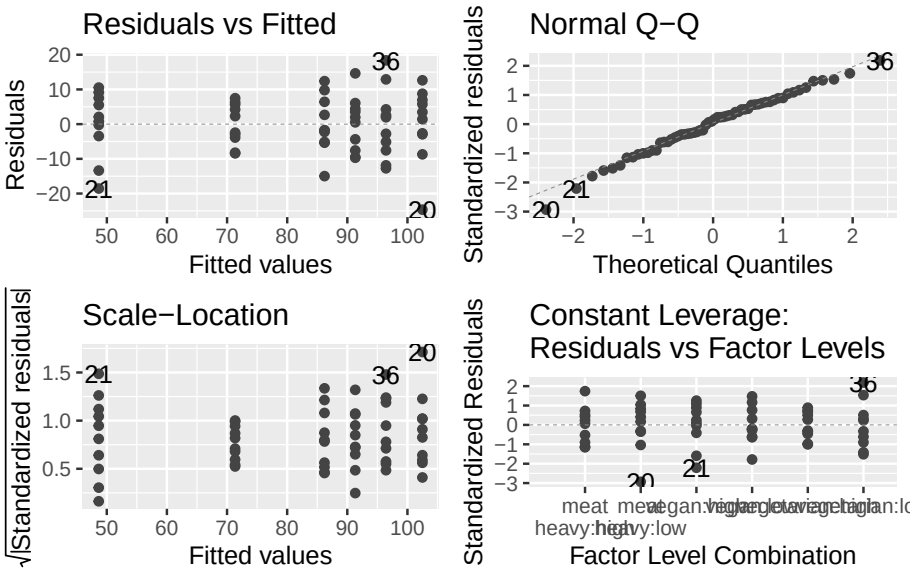
The hypothesis testing is the same as before. We fit the model with interaction term:

```
mod_2cat_3levels <- lm(bp ~ diet * exercise, data=bp_2cat_3levels)
```

We check the model assumptions:

```
autoplot(mod_2cat_3levels, smooth.colour = NA)
```

```
Warning: Removed 60 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 60 rows containing missing values or values outside the scale range
(`geom_line()`).
```



These look ok.

Now we do the ANOVA:

```
anova(mod_2cat_3levels)
```

```
Analysis of Variance Table

Response: bp
              Df Sum Sq Mean Sq F value    Pr(>F)
diet           2 8724.1  4362.1  55.636 7.644e-14 ***
exercise       1 9078.3  9078.3 115.789 4.791e-15 ***
diet:exercise  2 1741.3   870.6  11.104 9.130e-05 ***
Residuals     54 4233.8    78.4
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Important: Although we have a categorical variable with three rather than two levels, we still have only four rows in the ANOVA table. One for each main effect (diet and exercise), one for the interaction effect (diet:exercise), and one for the residuals. This is because the ANOVA table tests each effect as a whole, rather than testing each level of the categorical variable separately.

In this case, we see that there is strong evidence of main effects of diet and exercise on blood pressure, as well as strong evidence of an interaction effect between diet and exercise on blood pressure.

Reporting the patterns and statistics is similar to before, but now we have more levels to consider so the reporting is a bit more complex, and we have to be

careful to not over-interpret the results. For example, when the hypothesis test is on the interaction term via an F-test, we can only say that there is evidence of an interaction between diet and exercise on blood pressure. We cannot say which diets have different effects of exercise on blood pressure. To do that, we would need to do post-hoc tests, such as pairwise comparisons between the levels of diet within each level of exercise.

> Caution
>
> **Degrees of freedom** Look at the ANOVA table and the degrees of freedom column. For diet, the degrees of freedom is 2, because there are three levels of diet (meat heavy, vegetarian, vegan), and the degrees of freedom is number of levels minus 1. For exercise, the degrees of freedom is 1, because there are two levels of exercise (low, high). For the interaction term diet:exercise, the degrees of freedom is 2, which is the product of the degrees of freedom for diet (2) and exercise (1).
> Another way to think about this is how many parameters are estimated? The answer is as follows: six parameters are estimated in total, one for each of the combinations of diet and exercise (3 diets x 2 exercises = 6 combinations). Hence the residual degrees of freedom is total number of observations (60) minus 6.
> Don't worry if this is a bit confusing at first. It will become clearer with practice, and you can ask for it to be explained again and in different ways.

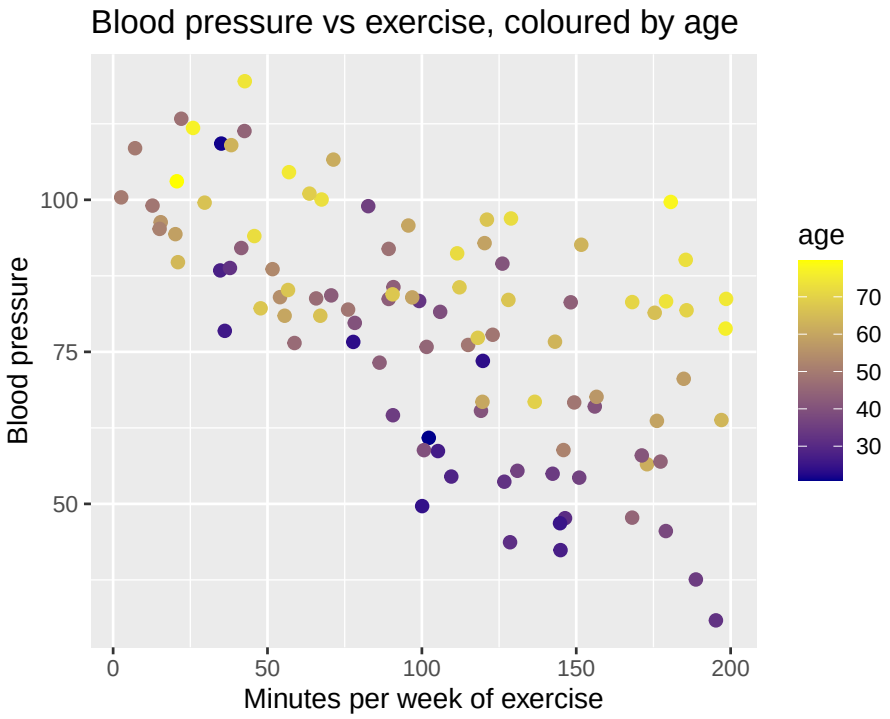
3. Multiple regression with interaction term

Above we had the example (Example 3) of two continuous explanatory variables (age and minutes of exercise) and one continuous response variable (blood pressure). We saw evidence of an interaction between age and minutes of exercise on blood pressure.

Here are the first few rows of the data again:

```
# A tibble: 6 x 5
     bp   age mins_per_week age_class mins_per_week_class
  <dbl> <dbl>         <dbl> <fct>     <fct>
1  55.4  35.9         131.  30-40     120-140
2  84.3  42.3          70.6 40-50     60-80
3  84.0  54.4          54.1 50-60     40-60
4  83.7  74.5         199.  70-80     180-200
5  53.7  32.1         127.  30-40     120-140
6 119.   73.9          42.6 70-80     40-60
```

Here is the data in a graph, with age represented by a colour gradient:



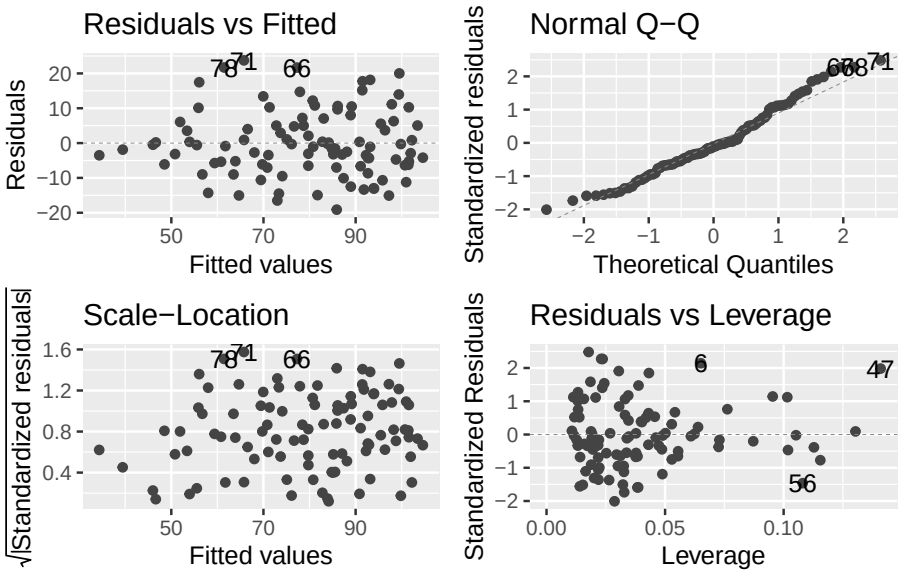
We can fit a multiple regression model with an interaction term in R as follows:

```
mod_2cont <- lm(bp ~ mins_per_week * age, data=bp_2cont)
```

We check the model assumptions:

```
autoplot(mod_2cont, smooth.colour = NA)
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
Removed 100 rows containing missing values or values outside the scale range
(`geom_line()`).
```



These look ok.

Now we do the F-test for the interaction term:

```
anova(mod_2cont)
```

```
Analysis of Variance Table

Response: bp
                  Df  Sum Sq Mean Sq F value    Pr(>F)
mins_per_week      1 14222.8 14222.8 152.509 < 2.2e-16 ***
age                1  9854.3  9854.3 105.666 < 2.2e-16 ***
mins_per_week:age  1  2827.7  2827.7  30.321 3.052e-07 ***
Residuals         96  8952.8    93.3
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It looks the same as in the two-way ANOVA case! This is because we still have two variables and their interaction, so the ANOVA table has one row for each main effect (mins_per_week and age), one row for the interaction effect (mins_per_week:age), and one row for the residuals.

The F-statistic for the interaction term is large (30.32) and the corresponding p-value is very small (3.1×10^{-7}). Hence, we conclude that there is strong evidence of an interaction between minutes of exercise and age on blood pressure. The effect of minutes of exercise on blood pressure depends on age, with a stronger effect for younger people compared to older people.

General points

Multiple regression vs. many single regressions

Question: Why not just fit a separate simple regression model and then test whether the slopes are the same (i.e., if they are parallel)? That is, why not fit the two models:

$$y_i = \beta_{0,veg} + \beta_{1,veg} x_i^{(1)} + \epsilon_i$$

$$y_i = \beta_{0,meat} + \beta_{1,meat} x_i^{(2)} + \epsilon_i$$

and compare the estimate of $\beta_{1,veg}$ to the estimate of $\beta_{1,meat}$?

Well, you could do that, and could probably find a way to test for whether the difference in the slopes is different from 0. This would be a test of the null hypothesis that the effect of minutes of exercise on blood pressure is the same for people with different diets. But, this would be a more complicated way to do it, and would not be as general as the model with the interaction term. The model with the interaction term is more general, more flexible, and more elegant.

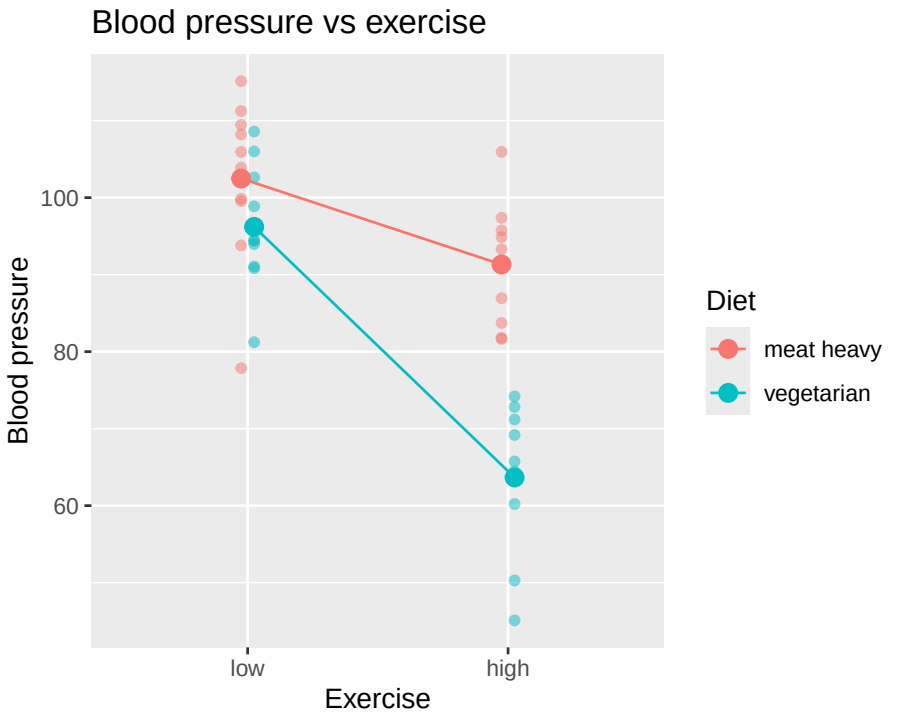
> **i** Note
>
> It is usually better model the whole dataset to test a single hypothesis, rather dividing up the dataset into smaller parts, fitting a model to each part, and then comparing the results of the models. The latter approach is less efficient and less elegant. **One hypothesis = one model.**

Intepreting main effects and interaction effects

The term *main effect* refers to the individual effect of each categorical explanatory variable on the response variable, ignoring the other variable. For example, the main effect of diet would be the difference in blood pressure between meat heavy and vegetarian diets, averaged over both levels of exercise.

However, if there is an interaction between the two categorical explanatory variables, the main effects may not fully capture the relationship. The interaction effect indicates that the effect of one categorical variable on the response variable depends on the level of the other categorical variable. For example, the effect of diet on blood pressure may differ between low and high exercise groups.

We can see this in the example:



There is a main effect of diet on blood pressure, as well as a main effect of exercise on blood pressure. However, there is also an interaction effect between diet and exercise on blood pressure, as the effect of diet on blood pressure depends on the level of exercise.

It could look otherwise. For example, if we found a (albeit rather unlikely pattern) of higher blood pressure for vegetarians who exercise little, and lower blood pressure for vegetarians who exercise a lot, we would have a pattern as follows:



Here there is no main effect of diet on blood pressure, as the average blood pressure for meat heavy and vegetarian diets is the same when averaged over both levels of exercise. There is still a main effect of exercise on blood pressure, as blood pressure is lower for high exercise compared to low exercise. And there is still an interaction effect between diet and exercise on blood pressure, as the effect of diet on blood pressure depends on the level of exercise.

Take home: Interactions are very interesting, but also will require careful and nuanced interpretation.

More than two explanatory variables

All the examples above had two explanatory variables. What if we have more than two explanatory variables? The principles are the same, but the models and interpretations become more complex. For example, if we have three categorical explanatory variables (A, B, C), we can fit a model with main effects of A, B, and C, as well as all possible interaction effects (A:B, A:C, B:C, A:B:C). The ANOVA table will have one row for each main effect, one row for each two-way interaction effect, one row for the three-way interaction effect, and one row for the residuals.

Interpretation of three-way interactions can be quite complex, as it involves understanding how the effect of one variable on the response variable depends on

the levels of the other two categorical variables. Very careful consideration and visualization of the data are often necessary to fully understand and communicate the results.

Review

- Interactions are some of the most interesting effects in biology and medicine. They occur when the effect of one explanatory variable on the response variable depends on the level of another explanatory variable.
- Interactions can occur between continuous and categorical explanatory variables, or between two categorical explanatory variables, or between two continuous explanatory variables.
- Visualization is key to understanding interactions. Use graphs to explore and communicate interactions.
- Hypothesis testing for interactions is done using F-tests on the interaction terms in linear models.
- The degrees of freedom for interaction terms depend on the levels of the categorical variables involved. Each categorical variable takes degrees of freedom equal to the number of levels minus one. Each continuous variable takes one degree of freedom. The degrees of freedom for the interaction term is the product of the degrees of freedom of the individual variables.
- Report interactions carefully, focusing on the biological interpretation and including relevant statistics (i.e., the F-statistic, degrees of freedom for the interaction term, degrees of freedom for error, and p-value).

Further reading

- Wikipedia is always a good place to start for understanding concepts: Interaction term (statistics)
- This is a nice paper for clinicians on understanding interactions: How to interact with interactions: what clinicians should know about statistical interactions
- If you want a challenge, and to see how nuanced interactions and interpretation can be, have a look at this paper Interactions in statistical models: Three things to know

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Degrees of freedom

It can be useful to think about degrees of freedom in terms of how many parameters / coefficients have to be estimated by the model.

In one-way ANOVA, this is the number of groups one has. So if you have a categorical variable with five categories (in another language one would say a factor variable with five levels), there will be five means estimated, and so the model uses five degrees of freedom. So error degrees of freedom will be the total number of observation minus five.

(Note that here we count *all* the parameters the model estimates, including the intercept. In an ANOVA table, the degrees of freedom shown for a term do not include the intercept, so a categorical variable with five levels has $5 - 1 = 4$ degrees of freedom there, as in Chapter 5. Both ways of counting give the same residual degrees of freedom.)

In two-way ANOVA (which includes the interaction of the two categorical variables) we have to estimate a mean for each of the combinations of the categories. So if we have one categorical variable with three levels and another with four, there are 3*4 combinations, so 12 means to estimate. So the model takes 12 degrees of freedom, and the error degrees of freedom will be the number of observations minus 12.

If we made a two-way ANOVA without an interaction (i.e. we included only the main effects) between the two explanatory variables, and we could have (i.e. we had performed a fully factorial experiment), well, this is just weird. Why would we not do the analysis we planned when we designed the experiment?

(I raised that in case anyone wants to know how to calculate degrees of freedom with only main effects. In that case, the two explanatory variables share the same reference level (intercept) so in the case above, only 6 things would need to be estimated: A shared intercept, and the other five means. Its a bit like when we previously made a regression model with two regression lines, both with the same slope. They shared the same slope – so only one needed to be estimated. So often when we use only main effects, getting the degrees of freedom is a bit tricky.)

Bottom line: more explanatory variables, more interactions, more levels in categorical explanatory variables = more things have to be estimated = fewer degrees of freedom for error. And few degrees of freedom for error is not good, generally speaking.

3d scatter plot

Here is a 3D scatter plot of the data with two continuous explanatory variables (age and minutes of exercise) and one continuous response variable (blood pressure). This plot helps to visualise the interaction between age and minutes of exercise on blood pressure.

(This interactive figure is shown only in the online, HTML, version of the book.)

```
plotly::plot_ly(bp_2cont, x = ~mins_per_week, y = ~age, z = ~bp, type = "scatter3d", mo
  plotly::layout(title = "3D Scatter plot of Blood Pressure vs Exercise and Age",
         scene = list(xaxis = list(title = "Minutes per week of exercise"),
                      yaxis = list(title = "Age"),
                      zaxis = list(title = "Blood Pressure"))
         )
```

This interactive 3-D figure is shown only in the online version of the book: see this chapter online.

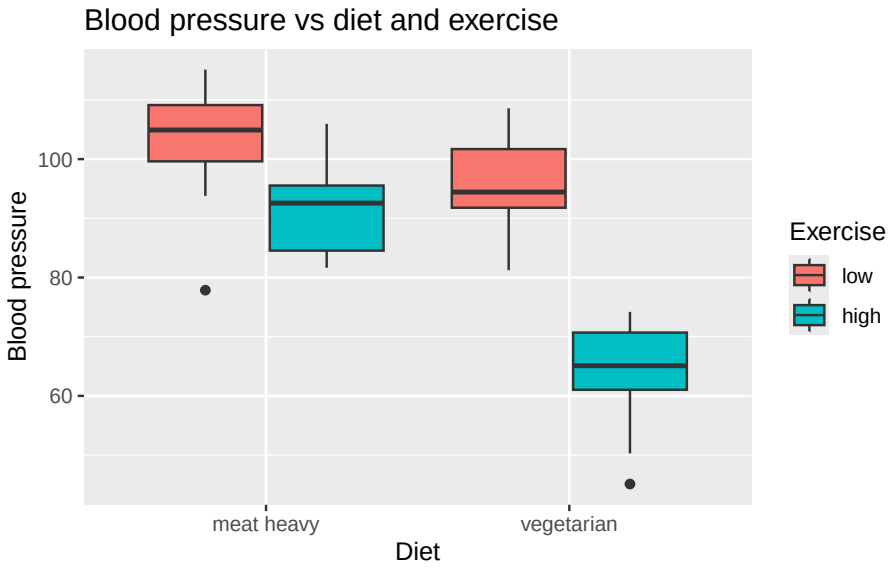
Types of sums of squares

> **!** Important
>
> **A note on anova() and "Type I" (sequential) sums of squares**
> In this course we often use `anova()` to test terms in a linear model. In base R, `anova()` for an `lm` uses Type I (sequential) sums of squares. That means the test for each term is done in the order the terms appear in the model formula: each term is tested after the terms before it have already been included.
> In balanced designs (equal sample sizes in each group/combination), the order usually does not matter much. But in unbalanced designs (unequal sample sizes), the p-value for a " "main effect" or an interaction can change if you change the order of terms in the model formula.
> This is related to collinearity between explanatory variables. When explanatory variables are correlated, the variance they explain in the response variable overlaps. In that case, the order of terms matters because earlier terms get to "claim" more of the shared variance.
> So: when you see an ANOVA table from `anova()`, remember that it is testing terms sequentially, not "all at once".

Box and whisker plot for two cats

Often you will see box and whisker plots used to visualise data with two categorical explanatory variables. Here is how to make such a plot in R:

```
ggplot(bp_2cat, aes(x=diet, y=bp, fill=exercise)) +
  geom_boxplot(position=position_dodge(width=0.8)) +
  labs(title="Blood pressure vs diet and exercise",
       x="Diet",
       y="Blood pressure",
       fill="Exercise")
```



I (Owen) prefer to show the individual data points whenever possible, so here is a box and whisker plot with the individual data points overlaid:

```
ggplot(bp_2cat, aes(x=diet, y=bp, fill=exercise)) +
  geom_boxplot(position=position_dodge(width=0.8), alpha = 0.5) +
  geom_jitter(aes(col=exercise), position=position_jitterdodge(jitter.width=0.2, dodge.width=0.8)
  labs(title="Blood pressure vs diet and exercise",
       x="Diet",
       y="Blood pressure",
       fill="Exercise",
       col="Exercise")
```



I, and others, prefer to not use bar plots with error bars to visualise data with categorical explanatory variables. Bar plots hide the individual data points and can be misleading. Box and whisker plots are better, but still hide some of the data. Scatter plots with jittered points are often the best way to visualise such data.

In case you must use bar plots with error bars, here is how to make one in R:

```
grouped_data <- group_by(bp_2cat, diet, exercise) |>
  summarise(mean_bp = mean(bp), sd_bp = sd(bp), n = n(), se_bp = sd_bp / sqrt(n), .grou
ggplot(grouped_data, aes(x=diet, y=mean_bp, fill=exercise)) +
  geom_bar(stat="identity", position=position_dodge(width=0.8), alpha = 0.5) +
  geom_errorbar(aes(ymin=mean_bp - se_bp, ymax=mean_bp + se_bp), position=position_dodg
  labs(title="Blood pressure vs diet and exercise",
       x="Diet",
       y="Blood pressure",
       fill="Exercise")
```


Blood pressure vs diet and exercise
Blood pressure
100
75
50
25
0
meat heavy
vegetarian
Diet
Exercise
low
high

Count data (L8)

Learning objectives

After this unit, students will be able to:

1. Identify biological questions and study designs in which the response variable is a count.
2. Explain why linear models with normal errors are often inappropriate for count data (counts are non-negative integers, and their variance tends to increase with the mean).
3. Describe the three components of a generalised linear model: the linear predictor, the distribution family and the link function.
4. Use R to fit a Poisson regression with `glm()`.
5. Explain null and residual deviance as measures of fit for a GLM, analogous to sums of squares in a linear model.
6. Check the assumptions of a Poisson GLM using diagnostic plots, recognising that QQ-plots of deviance residuals are interpreted differently from those of a linear model.
7. Interpret coefficients on the log (link) scale, and back-transform them to multiplicative effects on the expected count.
8. Use R to calculate predicted counts for given values of the explanatory variables.
9. Use deviance-based tests to judge whether explanatory variables are associated with the response.
10. Explain that in a chi-squared (likelihood ratio) test between nested GLMs, the degrees of freedom equal the difference in the number of estimated parameters.
11. Detect overdispersion and select an appropriate response to it (e.g. a quasi-Poisson or negative binomial model).
12. Recognise zero inflation in count data, and that it requires models beyond the Poisson GLM.
13. Recognise a correct and complete statement of results from a Poisson regression.

These learning objectives, together with the content of this chapter, define

what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch08_glm_count_data.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 8 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch08_glm_count_data.R")
```

Video walkthrough: coming soon.

In all the previous chapters, we have focused on **linear models**. These are suitable for continuous response variables that can, mathematically, take negative values. However, response variables can be otherwise. For example, we might have a response variable that is the number of offspring produced by a mother. This is a *count* type of response variable. In this chapter, you'll see why the models we used so far are usually inappropriate for count data. Then you'll learn about a more appropriate type of model: **Generalized Linear Models (GLMs)**.

Introduction

So far in BIO144, we have focused on **linear models** fitted using `lm()`. These models assume:

- A **continuous response variable** that can take any real value.
- Normally distributed residuals.
- Constant variance (homoscedasticity).

Linear models are powerful, but these assumptions can be violated in biological data. In this chapter we move beyond linear models to handle an important new type of response variable: **counts**. We introduce **Generalized Linear Models (GLMs)**, which extend linear models by allowing the response variable to follow distributions other than the normal distribution.

By the end of this chapter, you should be able to:

- Recognise when linear regression is inappropriate
- Understand the core components of a GLM
- Fit and interpret Poisson regression models
- Diagnose common problems such as overdispersion and zero inflation

Example: Soay sheep

A feral population of Soay sheep on the island of Hirta (Scotland) has been studied extensively. Ecologists were interested in whether the **body mass of female sheep** influences their fitness, measured as **lifetime reproductive success** (number of offspring produced over a lifetime).

Question: Are heavier females fitter than lighter females?

Read in an example dataset:

```
soay <- read_csv("datasets/soay_sheep.csv")
```

```
Rows: 100 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): body.size, fitness

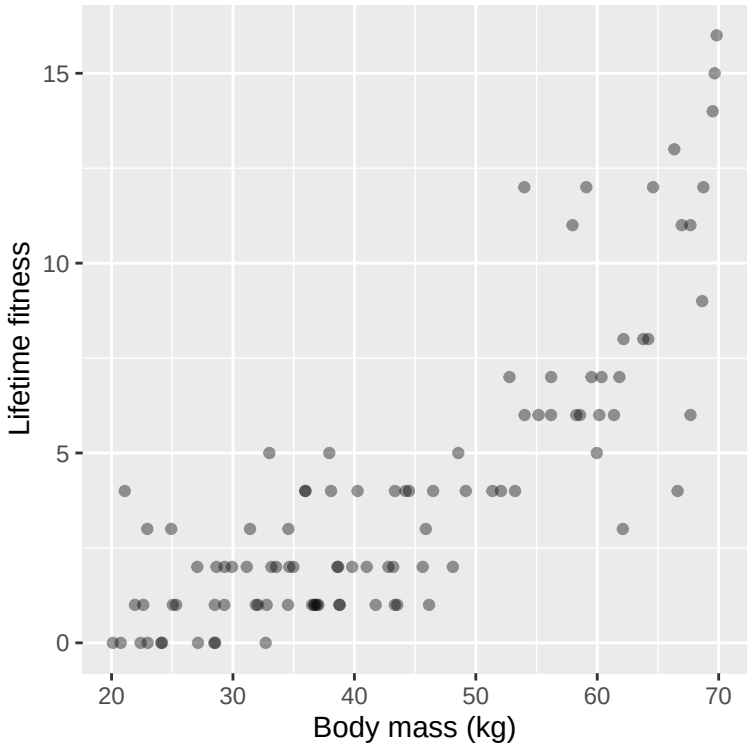
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the dataset:

```
head(soay)
```

```
# A tibble: 6 x 2
  body.size fitness
      <dbl>   <dbl>
1      54.0      12
2      32.7       0
3      33.6       2
4      43.2       2
5      43.3       4
6      59.5       7
```

As always, we start by exploring the data visually:



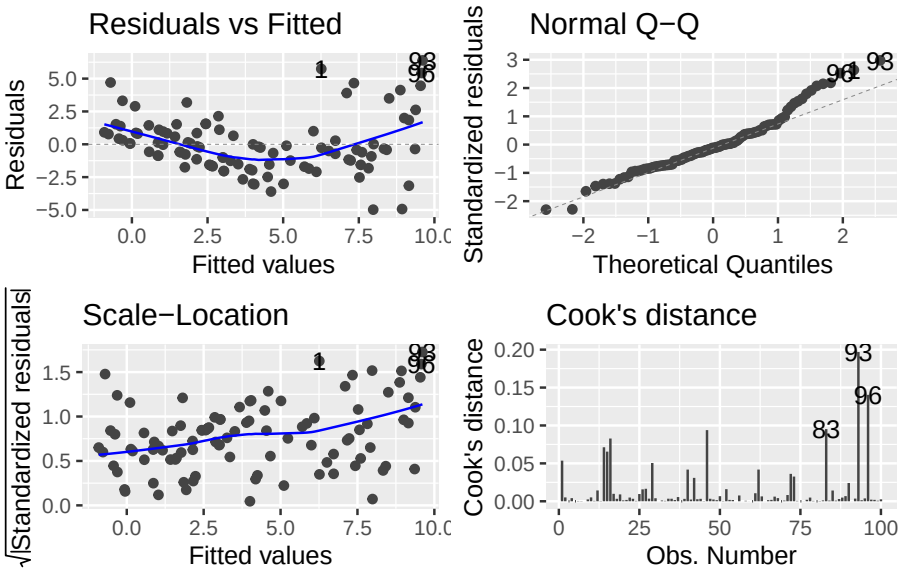
The wrong analysis

Let's analyse the data with linear regression, treating counts as if they were continuous.

```
mod_soay_lm <- lm(fitness ~ body.size, data = soay)
```

The model checking plots show clear violations of linear regression assumptions:

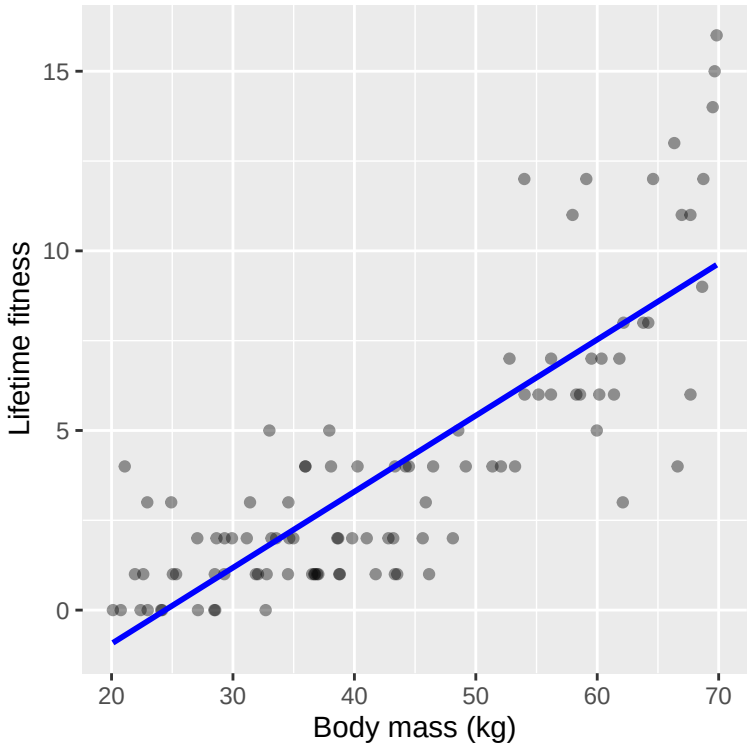
```
#par(mfrow = c(2, 2))
autoplot(mod_soay_lm, which = 1:4, add.smooth = TRUE)
```



The qq-plot looks ok-ish, though the larger residuals show a tendency to be larger than expected. The scale-location plot shows that variance increases with fitted values, violating homoscedasticity. Also, there is a clear non-linear pattern in the residuals vs fitted plot, suggesting that the linear model is not capturing the relationship well.

When we plot the fitted line, we see the linear relationship:

```
`geom_smooth()` using formula = 'y ~ x'
```



Caution

Why this matters
Violating model assumptions can lead to biased estimates, incorrect standard errors, and misleading p-values.

Another issue remains, however: linear regression can predict negative counts, which are impossible. We can see this in a plot of the data and the fitted regression line.

Why linear regression fails for count data

- The normal distribution is for continuous variables.
- It allows negative values.
- It assumes constant variance.

However, many biological response variables are:

- **Counts** (e.g. offspring, parasites, species)
- **Binary responses** (e.g. alive/dead)
- **Proportions**

In these cases, forcing the data into a linear model often leads to invalid predictions and misleading inference. **Generalized Linear Models (GLMs)** solve this problem.

> **!** Important
>
> **Key idea** GLMs are not a replacement for linear models — they are a *generalisation* of them. Linear regression is a special case of a GLM.

From LM to GLM

A linear model is actually a special case of a generalized linear model.

What you already know: in a linear model we write:

$$y_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)} + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

In words, this means an observed response variable y_i is modelled as a linear combination of explanatory variables plus normally distributed error.

This implies that

$$y_i \sim N(\mu_i, \sigma^2)$$

where μ_i is the mean of the normal distribution for observation i, and

$$\mu_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$$

Moving to things you don't know:

A generalised linear model has three components: a random component (*family*), a systematic component (*linear predictor*), and a link function. In a linear model (like the ones you already saw), these are as follows:

- **Random component (family)**: Normal distribution $N(\mu_i, \sigma^2)$
- **Systematic component (linear predictor)**: $\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$
- **Link function**: Identity link, $\mu_i = \eta_i$.

Putting this all together in a description of a generalised linear model we have:

Family: $y_i \sim N(\mu_i, \sigma^2)$

Linear predictor: $\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$

Link function: $\mu_i = \eta_i$

In R, we use the function `glm()` to fit generalised linear models. For a normal linear model of the Soay sheep data, we would write:

```
mod_soay_glm <- glm(fitness ~ body.size, data = soay, family = gaussian(link = "identi
```

Here we have specified the family as Gaussian (i.e., Normal) with an identity link. This model produces the same results as the linear model we fitted earlier with `lm()`. And we can get the following model summary:

```
summary(mod_soay_glm)
```

```
Call:
glm(formula = fitness ~ body.size, family = gaussian(link = "identity"),
    data = soay)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.16768    0.68895  -7.501  2.9e-11 ***
body.size    0.21163    0.01501  14.097  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 4.805055)

    Null deviance: 1425.8  on 99  degrees of freedom
Residual deviance:  470.9  on 98  degrees of freedom
AIC: 444.73

Number of Fisher Scoring iterations: 2
```

Compare that to the linear model summary:

```
summary(mod_soay_lm)
```

```
Call:
lm(formula = fitness ~ body.size, data = soay)

Residuals:
    Min      1Q  Median      3Q     Max
-4.9760 -1.5468 -0.2568  0.9733  6.3887

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.16768    0.68895  -7.501  2.9e-11 ***
body.size    0.21163    0.01501  14.097  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 2.192 on 98 degrees of freedom
Multiple R-squared:  0.6697,    Adjusted R-squared:  0.6664
F-statistic: 198.7 on 1 and 98 DF,  p-value: < 2.2e-16
```

The coefficients table is exactly the same. Other parts of the output differ slightly because `glm()` uses maximum likelihood estimation (MLE) while `lm()` uses least squares estimation. Put that aside for now; the key point is that both R functions fit the same model when using a normal family with an identity link.

So, you now know what is a generalised linear model.

In one sentence, a GLM is a model where the response variable follows a specified distribution (family), the mean of that distribution is linked to a linear combination of explanatory variables (linear predictor) via a link function. We will now look at a specific type of GLM for count data: the Poisson GLM.

Poisson GLM

We need three components for a GLM:

1. A probability distribution (family) for the response variable.
2. A linear predictor.
3. A link function relating the linear predictor to the mean of the distribution.

Let's start with a discussion of the probability distribution (family) for count data. And what we're aiming for is a probability distribution that we expect to be a reasonably good starting point for modelling count data. A common probability model for counts is the **Poisson distribution**. The Poisson distribution is often the default starting point for modelling counts because it is the simplest distribution that respects characteristics of count data: discreteness, non-negativity, and increasing variance.

Unlike the normal distribution, which has two parameters (mean and spread), the Poisson distribution has only one parameter: the mean λ. In the Poisson distribution, the mean and variance are equal.

In a Poisson distribution, the probability of observing a count y is given by:

$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$

where:

- Y is a random variable representing the count.
- $y = 0, 1, 2, ...$
- $\lambda > 0$ is the mean (and variance) of the distribution.

In R, we can get the probability of observing a count of 3, when the expected count is 5, using the `dpois()` function:

```
dpois(3, lambda = 5)
```

```
[1] 0.1403739
```

And we expect the probability of observing a count of 5 when the expected count is 5 to be higher:

```
dpois(5, lambda = 5)
```

```
[1] 0.1754674
```

What happens when we ask for the probability of observing a count of 3.5 when the expected count is 5? Or a count of -1?

```
dpois(3.5, lambda = 5)
```

```
Warning in dpois(3.5, lambda = 5): non-integer x = 3.500000
```

```
[1] 0
```

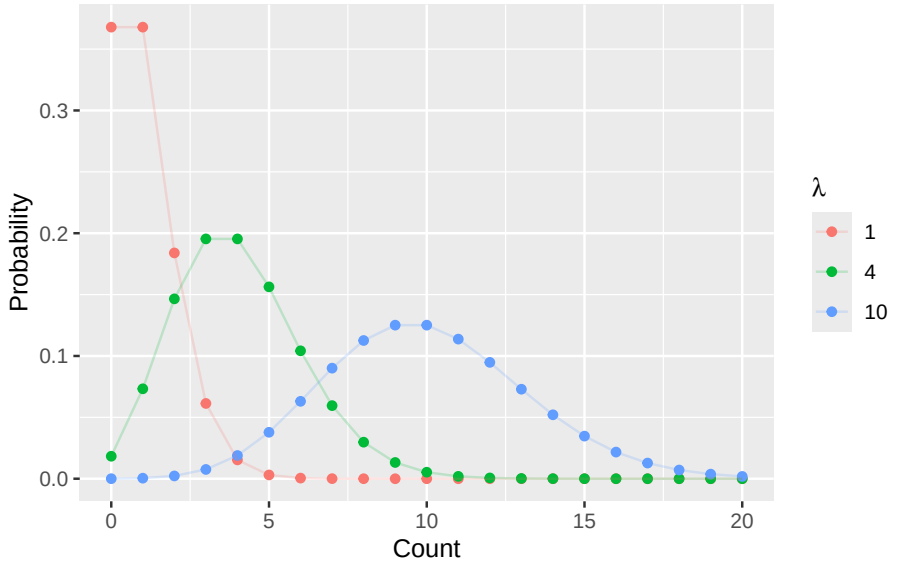
The warning message tells us that the Poisson distribution is only defined for non-negative integers. This is a key property of the Poisson distribution: it is only defined for integers (0, 1, 2, …).

```
dpois(-1, lambda = 5)
```

```
[1] 0
```

And we see that the probability of observing a negative count is zero.

Here are a few of Poisson distributions with different means:



The Poisson distribution has two important properties:

1. It is defined only for **non-negative integers** (0, 1, 2, …)

2. The **mean and variance are equal**.

> **!** Important
>
> **Mean–variance relationship**
> In a Poisson distribution, the mean and variance are equal. This captures an important feature of count data, but it will later lead to the concept of *overdispersion*.

So, in a Poisson GLM we assume a Poisson distribution (family) for the response variable.

$$y_i \sim \text{Poisson}(\mu_i)$$

where $\mu_i = \lambda_i$ is the expected count for observation i.

Next we need to define the linear predictor (and then we finish with the link function).

The linear predictor is the same as in linear regression:

$\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$

It is just the linear combination of explanatory variables and coefficients. Nothing new here.

Finally, we need to define the link function that relates the linear predictor to the expected count μ_i.

We cannot use the identity link here, because that would allow negative expected counts. Instead, we use the **log link**:

$$\eta_i = \log(\mu_i)$$

which implies:

$\mu_i = \exp(\eta_i)$

The log link ensures that the expected count is always positive, regardless of the values of the β parameters and explanatory variables. Whatever the value of the linear predictor η_i, the exponential of it $\exp(\eta_i)$ is always positive.

Putting this all together, a Poisson GLM can be summarised as:

Family: $y_i \sim \text{Poisson}(\mu_i)$

Linear predictor: $\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$

Link function: $\log(\mu_i) = \eta_i$

We could also write this as:

$$y_i \sim \text{Poisson}\Big(\exp\big(\beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}\big)\Big)$$

Parameter estimation

We observe the counts y_i, and the model predicts the expected counts μ_i. The GLM estimates the parameters $\beta_0, \beta_1, \ldots, \beta_p$ that make the observed data most probable under the assumed Poisson model. This is done using maximum likelihood estimation (MLE). The likelihood of the data given the model is maximised.

Recall that we saw that the summary information of the linear model and the GLM with normal family and identity link were different. This is because of differences in estimation methods (least squares vs MLE). In GLMs, we always use MLE for parameter estimation. In this course we will not go into the mathematical details of MLE, but the key idea is that we find parameter values that make the observed data most probable under the assumed model. This is analogous to least squares estimation in linear models, which is a special case of MLE for Normally distributed errors.

R - Poisson GLM

When fitting a Poisson GLM in R, we use the `glm()` function, specifying the family as `poisson`:

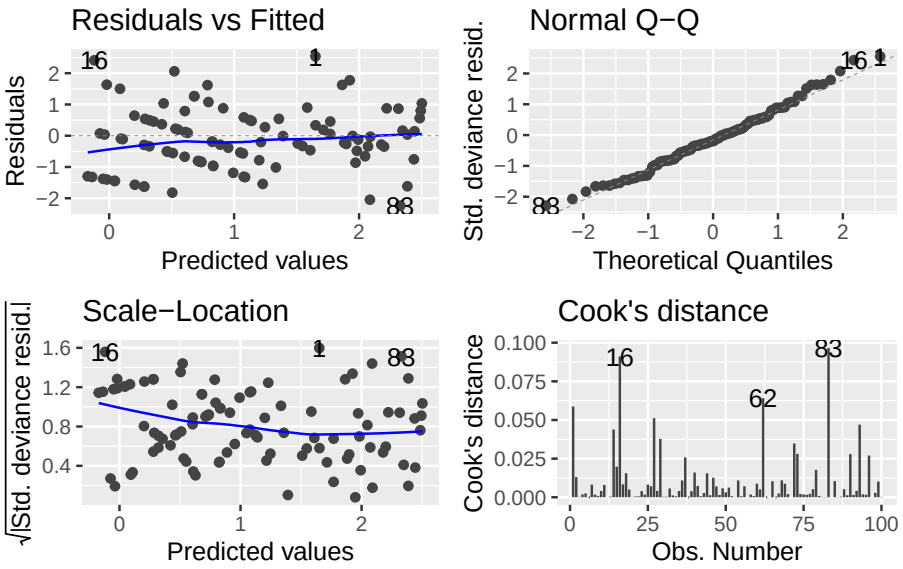
```
soay_glm <- glm(fitness ~ body.size, data = soay, family = poisson(link = "log"))
```

Specifying `family = poisson` tells R to use the Poisson distribution with a log link function by default. We also specify the link explicitly as `link = "log"`. You will, however, often only see `family = poisson` and not the log link, because the log link is the default for the Poisson family.

Checking model assumptions

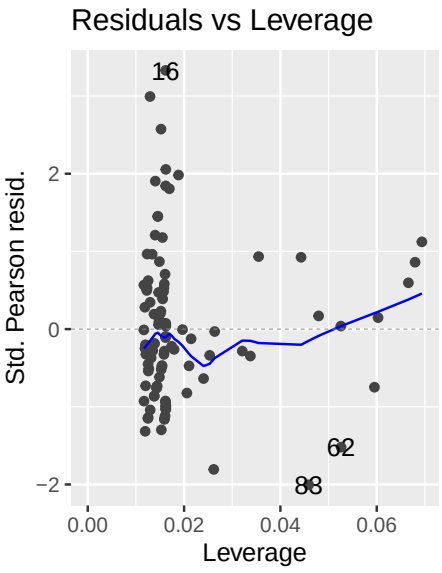
As always, we should check model assumptions:

```
#par(mfrow = c(2, 2))
autoplot(soay_glm, which = 1:4, add.smooth = TRUE)
```



The bottom-right plot above shows Cook's distance for each observation. The checklist below also asks for the residuals vs leverage plot, which we get with `which = 5`:

```
autoplot(soay_glm, which = 5)
```



The QQ-plot is rather concerning. However, QQ-plots in GLMs are not testing normality of residuals in the same way as for linear models, so their interpretation

differs. The *deviance residuals* should approximately follow a normal distribution if the model fits well. Here, there are some deviations from normality, which could be somewhat concerning, but for now we will focus on the other plots. (If you're interested in what are deviance residuals, please see the *Extras* section at the end of this chapter.)

The other plots look much better than for the linear model. The residuals vs fitted plot shows no obvious pattern, and the scale-location plot shows more constant variance.

ℹ Checklist: checking a GLM

1. **Is the family right for the response variable?** Counts: Poisson (or quasi-Poisson / negative binomial if overdispersed). Binary data, or successes out of a number of trials: binomial. No diagnostic plot can rescue a model with the wrong family.
2. **Are the observations independent?** This comes from the study design, not from the plots. If observations are grouped or repeated, a mixed model is needed (Chapter 11).
3. **Residuals vs fitted values.** Look for strong patterns or curvature, which suggest a missing explanatory variable, a missing interaction, or a non-linear relationship. For individual-level 0/1 data this plot shows two bands of points and is hard to interpret.
4. **Dispersion.** The ratio of residual deviance to residual degrees of freedom should be roughly 1. Values above about 1.5–2 suggest overdispersion (consider quasi-Poisson, negative binomial or quasibinomial). This check works for counts and aggregated binomial data, but not for individual-level 0/1 data.
5. **Influential observations.** As for linear models, look at the residuals vs leverage plot (Cook's distance).
6. **QQ-plot of the deviance residuals.** A GLM does not assume normally distributed residuals, so this is only a rough guide. Large departures can point to a wrong family or to overdispersion.
7. **Zeros (count data).** Many more zeros than the Poisson model predicts suggests zero inflation.

❗ Important

You must specify the **family** in `glm()`. If you omit the link function, R uses the default link for that family.

Interpreting coefficients

```
summary(soay_glm)
```

```
Call:
glm(formula = fitness ~ body.size, family = poisson(link = "log"),
    data = soay)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.253092   0.210186  -5.962 2.49e-09 ***
body.size    0.053781   0.003735  14.400  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 333.105  on 99  degrees of freedom
Residual deviance:  97.783  on 98  degrees of freedom
AIC: 378.16

Number of Fisher Scoring iterations: 5
```

So we have an intercept of $\beta_0 = -1.253$. Because of the log link, this means that when body size is 0 kg (which is outside the range of our data), the expected count of fitness is $\exp(-1.253) = 0.285$ offspring.

But what is the biological meaning of the slope for body size: $\beta_1 = 0.054$? It must be describing how something changes as a result of 1 unit change in body size (1 kg). But what?

Lets's find out by calculating the expected fitness for two body sizes that differ by 1 kg.

For a sheep of body size 40 kg, the linear predictor is:

$$\eta = \beta_0 + \beta_1 \times \text{body.size} = -1.253 + 0.054 \times 40 = 0.907$$

But this doesn't mean we predict a fitness of 0.907 offspring. Instead, we need to back-transform this using the link function. Since we used a log link, we exponentiate the linear predictor to get the expected count:

$$E(y_{40}) = \exp(\eta) = \exp(0.907) = 2.48$$

> **Caution**
>
> Some of the calculations in this text are done with rounded values for clarity. In practice, you should use the full precision values from R to avoid rounding errors. You will see below that the predicted value calculated here (2.48) differs from that calculated in R (2.46) due to rounding. Always do calculations with full precision values from R, and only round the final

result if needed.

Let us now calculate the expected fitness for a sheep of body size 41 kg. First calculate the value of the linear predictor:

$\eta = -1.253 + 0.054 \times 40 + 0.054 = 0.907 + 0.054 = 0.961$

The expected count is then $E(y) = \exp(0.961) = 2.61$

Now let's compare the value of the predicted fitness for the 41 and 40 kg sheep:

$\frac{E(y_{41})}{E(y_{40})} = \frac{\exp(0.907+0.054)}{\exp(0.907)}$

And because $\frac{\exp(a)}{\exp(b)} = \exp(a-b)$, we have:

$\frac{E(y_{41})}{E(y_{40})} = \exp(0.054) = 1.055$

This means that each increase in body size of 1 kg multiplies the expected count of fitness by a factor of $\exp(0.054) = 1.055$ (i.e., a 5.5% increase).

This shows that the exponentiated coefficient $\exp(\beta_1)$ gives the multiplicative change in expected count for a one-unit change in the explanatory variable.

Important

A one-unit increase in an explanatory variable multiplies the expected count by $\exp(\beta)$.

This means that for each additional kilogram of body size, the expected count of fitness increases by a factor of approximately 1.055 (i.e., a 5.5% increase). We use the number multiplicatively because of the log link function.

Pleaes note that of course all these calculations can be done in R. For example, for a body size of 40 kg:

```
body_size_example <- 40
linear_predictor <- coef(soay_glm)[1] + coef(soay_glm)[2] * body_size_example
expected_count <- exp(linear_predictor)
expected_count
```

```
(Intercept)
   2.455011
```

Note

Although counts can only be integers, the expected value from a Poisson model can be any positive real number. This is because the expected value is a mean over many possible counts.

Analysis of deviance

There is something technically different that we have glossed over until now: how model fit is assessed. In linear regression we estimate parameters by minimizing the sum of squared residuals. In GLMs, we use **maximum likelihood estimation (MLE)**. In this course we will not go into the mathematical details of MLE, but the key idea is that we find the parameter values that make the observed data most probable under the assumed model (just like in least squares). Instead of minimising sums of squares, we maximise the **likelihood** of the data given the model. Maximising the likelihood is equivalent to minimising the **deviance**, which is a measure of model fit based on likelihoods. Hence, when we fit a GLM in R, we get output including the **deviance** (and not sums of squares):

```
anova(soay_glm, test = "Chisq")
```

```
Analysis of Deviance Table

Model: poisson, link: log

Response: fitness

Terms added sequentially (first to last)

          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                         99     333.11
body.size  1   235.32        98      97.78 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the output we see the deviance for the null model and the fitted model, as well as the change in deviance when adding explanatory variables. We can use a chi-squared test to assess whether adding explanatory variables significantly improves model fit.

We use a chi-squared test here because, under certain regularity conditions, the change in deviance between nested models follows a chi-squared distribution with degrees of freedom equal to the difference in the number of parameters. If you are interested in what this means, you can read more about it in more advanced statistics textbooks.

You may be wondering if we can calculate an r-squared value for GLMs. While there is no direct equivalent of r-squared for GLMs, several pseudo-r-squared measures have been proposed. However, these are not as widely used or interpreted as r-squared in linear regression. If you're interested, check out the *Extras* section at the end of this chapter for more information.

Reporting

When reporting results from a Poisson GLM, we can make a graph and a sentence describing the pattern and related statistics.

When we wish to make a graph of the fitted relationship, we can either make it with the y-axis on the log scale (linear predictor scale) or back-transform to the original count scale. I usually prefer the original scale, as it is easier to interpret.

The first step is to create a new data frame with a sequence of body sizes for prediction, and errors for the 95% confidence intervals:

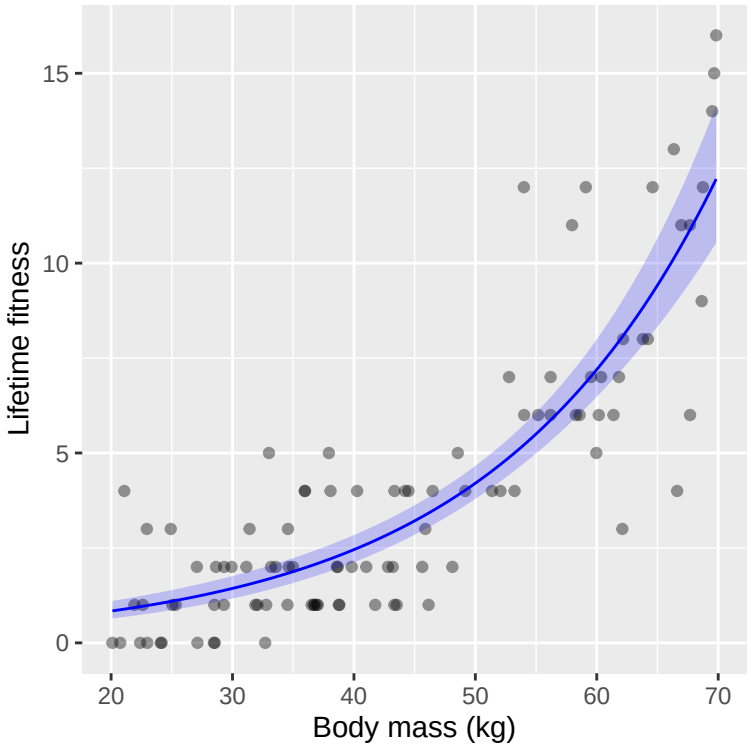
```
new_data <- data.frame(body.size = seq(min(soay$body.size), max(soay$body.size), lengt
soay_predictions <- predict(soay_glm, newdata = new_data, se.fit = TRUE)
new_data$fit <- exp(soay_predictions$fit)
new_data$lower <- exp(soay_predictions$fit - 1.96 * soay_predictions$se.fit)
new_data$upper <- exp(soay_predictions$fit + 1.96 * soay_predictions$se.fit)
```

Note that we back-transform the predictions and confidence intervals using the exponential function, since the link function is the log. And note that we must calculate the confidence intervals on the log scale first, and then back-transform them.

> **Note**
>
> If we only wanted the fitted value (and not confidence intervals) we could have used `type = "response"` in the `predict()` function to get predictions on the original count scale (back-transformed from the log scale). When we don't specified this, we would get predictions on the log scale–this is the default behavior.

Now we can plot the data and the fitted relationship with confidence intervals. Each layer gets its own `data` and `aes()`, because the layers use different data frames (`soay` for the points, `new_data` for the line and band):

```
ggplot() +
  geom_point(data = soay, aes(x = body.size, y = fitness), alpha = 0.4) +
  geom_line(data = new_data, aes(x = body.size, y = fit), color = "blue") +
  geom_ribbon(data = new_data, aes(x = body.size, ymin = lower, ymax = upper),
              alpha = 0.2, fill = "blue") +
  labs(x = "Body mass (kg)", y = "Lifetime fitness")
```



Excellent! We can now write a nice sentence for our results:

Reproductive fitness (in terms of lifetime number of offspring) increased significantly with body mass, with a unit increase in body mass associated with a multiplicative increase in expected fitness of 1.06 (95% CI: 1.05, 1.06; $\chi^2 = 235.32$, $df = 1$, $p < 4.1 \times 10^{-53}$).

Tip

How big is the effect? A multiplicative effect per kg of body mass can be hard to judge. It helps to also express it over the range of the data. In this example, the model predicts an expected lifetime fitness of 0.8 offspring for the lightest sheep (20 kg) and 12.2 offspring for the heaviest (70 kg). Whether a difference of this size is biologically important (for example, for the evolution of body size) is a biological question, but reporting the effect in this way gives readers what they need to judge it.

Well done!

We have made our first steps into the world of GLMs and Poisson regression for count data. You should now be able to:

- Recognise why linear regression is often inappropriate for count data.
- Understand the components of a GLM: linear predictor, link function, and family.
- Fit a Poisson GLM in R using `glm()`.
- Interpret coefficients from a Poisson GLM.

Next, we will look at common issues that arise when fitting Poisson models, such as overdispersion and zero inflation.

Overdispersion

In a Poisson model, mean and variance are assumed equal. In practice, variance often exceeds the mean, a phenomenon called **overdispersion**.

Common causes include:

- Unmeasured explanatory variables. This means important explanatory variables are missing from the model, leading to extra variability.
- Individual heterogeneity. This can arise when individuals differ in ways not captured by measured explanatory variables, leading to extra variability in counts.
- Correlated observations. This can occur when observations are not independent, such as repeated measures on the same individual.

A simple check for overdispersion is to compare the residual deviance to the residual degrees of freedom:

$$\text{Dispersion} = \frac{\text{Residual Deviance}}{\text{Residual DF}}$$

If this ratio is substantially greater than 1 (e.g., > 1.5 or 2), it indicates overdispersion.

Check this for our Soay sheep model:

```
dispersion_soay <- deviance(soay_glm) / df.residual(soay_glm)
dispersion_soay
```

```
[1] 0.9977877
```

Here, the dispersion is quite close to 1 (it is 1), indicating little overdispersion. However, in many real datasets, overdispersion is common.

> **Caution**
>
> Ignoring overdispersion leads to **anti-conservative p-values** (too small). This would be a typical example of **Type I error inflation**. Type I errors occur when we incorrectly reject a true null hypothesis, leading to false positives. In the context of statistical modeling, ignoring overdispersion can result in underestimating standard errors, which in turn leads to smaller p-values. Consequently, we may conclude that an effect is statistically significant when it is not, thereby increasing the likelihood of Type I errors.

Quasi-Poisson and negative binomial models

One solution is the **quasi-Poisson** model, which estimates an additional dispersion parameter:

For this, we can use `family = quasipoisson` in `glm()`:

```
soay_quasi <- glm(fitness ~ body.size, data = soay, family = quasipoisson)
summary(soay_quasi)
```

```

Call:
glm(formula = fitness ~ body.size, family = quasipoisson, data = soay)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.253092   0.209446  -5.983 3.59e-08 ***
body.size    0.053781   0.003722  14.450  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 0.992977)

    Null deviance: 333.105  on 99  degrees of freedom
Residual deviance:  97.783  on 98  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5
```

Zero inflation

A special case of overdispersion arises when there are **more zeros than expected** under a Poisson model. This is called **zero inflation**.

Examples include:

- Number of cigarettes smoked (many people do not smoke, more than can be modelled by a Poisson distribution)
- Parasite counts (many host individual have no parasites, more than can be modelled by a Poission distribution)

> **Note**
>
> Zero inflation often reflects **two processes**: whether an observation can be non-zero at all, and how large it is if it is. E.g., whether an individual smokes at all, and then how many cigarettes they smoke if they do.

In this course we will not describe in detail or practice fitting zero-inflated models, but they are an important tool for count data with many zeros. Examples of models include zero-inflated Poisson (ZIP) and zero-inflated negative binomial (ZINB) models. These models combine a count model (e.g., Poisson or negative binomial) with a separate model for the probability of being a structural zero. There is also a model called the **hurdle model**, which is similar but has a different interpretation.

Multiple explanatory variables

GLMs can include multiple explanatory variables, just like linear models. And they can include only categorical variables, only continuous variables, or a mix of both.

For example, we could include parasite load as an additional predictor of fitness in the Soay sheep data:

Read in the updated dataset:

```
soay <- read_csv("datasets/soay_sheep_with_parasites.csv")
```

```
Rows: 100 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): body.size, fitness, parasite.load

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

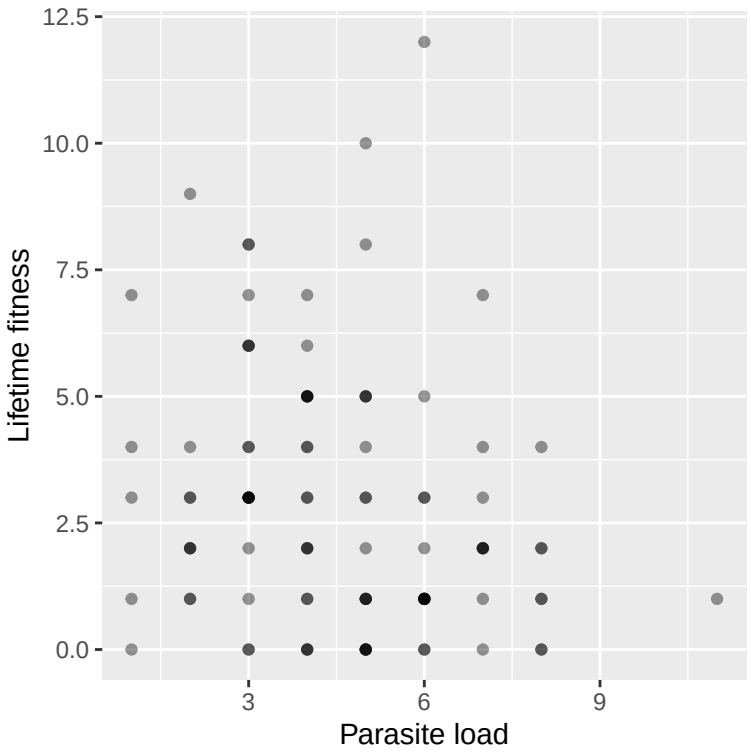
Here is the first few rows of the updated dataset:

```
head(soay)
```

```
# A tibble: 6 x 3
  body.size fitness parasite.load
      <dbl>   <dbl>         <dbl>
1      54.0       1             2
```

```
2       32.7        1             6
3       33.6        0             5
4       43.2        1             6
5       43.3        0             7
6       59.5        3             6
```

We can visualise the relationship between parasite load and fitness:



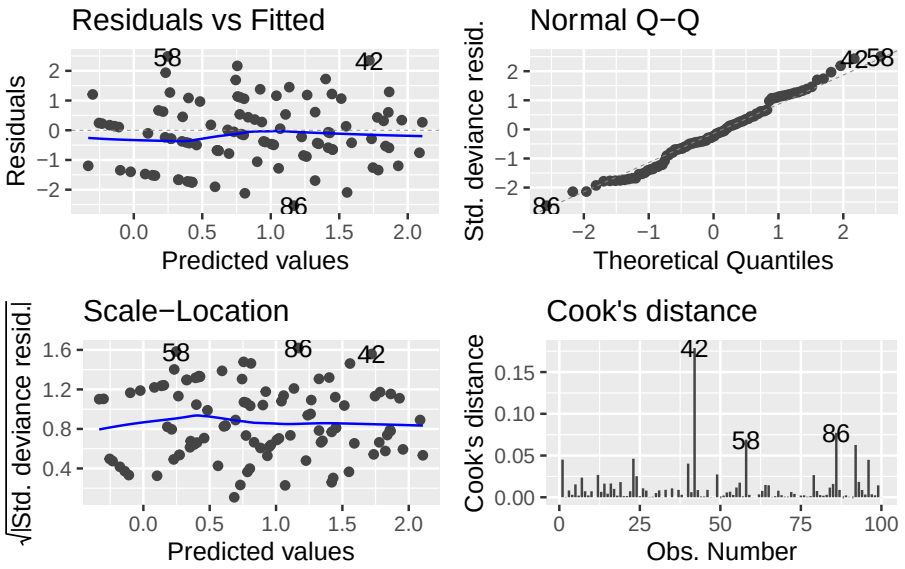
It looks like higher parasite loads are associated with lower fitness.

We can fit a Poisson GLM with both body size and parasite load as explanatory variables:

```
soay_glm_parasites <- glm(fitness ~ body.size + parasite.load, data = soay, family = poisson)
```

As always we check model assumptions:

```
#par(mfrow = c(2, 2))
autoplot(soay_glm_parasites, which = 1:4, add.smooth = TRUE)
```



The model checking plots look good. We can summarise the model:

```
anova(soay_glm_parasites, test = "Chisq")
```

```
Analysis of Deviance Table

Model: poisson, link: log

Response: fitness

Terms added sequentially (first to last)

              Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                            99     226.57
body.size      1   95.723        98     130.85 < 2.2e-16 ***
parasite.load  1   11.063        97     119.78 0.0008809 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that both body size and parasite load significantly affect fitness. Body size has a positive effect, while parasite load has a negative effect. These effects are interpreted conditional on the other explanatory variable being held constant, just as in multiple linear regression.

Review

In this chapter we have introduced Generalized Linear Models (GLMs) for count data, focusing on Poisson regression. Key points include:

- Linear regression is often inappropriate for count data due to violations of assumptions.
- GLMs extend linear models by allowing different distributions and link functions.
- Poisson GLMs use the Poisson distribution with a log link to model counts.
- Coefficients are interpreted on the log scale, with exponentiation giving multiplicative effects.
- Overdispersion and zero inflation are common issues that need to be addressed.
- GLMs can include multiple explanatory variables, just like linear models.

With this foundation, you are now equipped to analyse count data using GLMs in R. In the next chapter, we will explore further extensions and applications of GLMs.

Further reading

- Chapter 14 of *The R Book* by Crawley (2012) provides a comprehensive introduction to GLMs in R, including Poisson regression. The R is a bit old fashioned but the concepts are well explained.
- Generalized Linear Models With Examples in R (2018) by Peter K. Dunn and Gordon K. Smyth.

Extras

> **Note**
>
> The content of this Extras section is not examinable. It is here for those who want to learn more.

Deviance residuals

In linear models, residuals are simply the difference between observed and predicted values. In GLMs, residuals are more complex due to the non-normal distribution of the response variable.

Recall from above that a Poisson GLM is mathematically summarised as:

$y_i \sim \text{Poisson}(\lambda_i)$

where $\log(\lambda_i) = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$

The **deviance residuals** are a type of residual used in GLMs to assess model fit. They are derived from the concept of deviance, which measures the difference between the fitted model and a saturated model (a model that perfectly fits the data).

The deviance residual for each observation is calculated as: $r_i = \text{sign}(y_i - \hat{y}_i)\sqrt{2\left(y_i \log\left(\frac{y_i}{\hat{y}_i}\right) - (y_i - \hat{y}_i)\right)}$

where: - y_i is the observed count - $\hat{y}_i$ is the predicted count from the model

Deviance residuals have the following properties:

- They are approximately normally distributed if the model fits well. **Therefore they can be used in QQ-plots to assess model fit.**
- They can be positive or negative, indicating whether the observed count is above or below the predicted
- They are used in diagnostic plots to assess model fit, similar to residuals in linear models.

Pseudo-R-squared for GLMs

While there is no direct equivalent of r-squared for GLMs, several pseudo-r-squared measures have been proposed. These measures aim to provide an indication of model fit similar to r-squared in linear regression, but they do not have the same interpretation.

To understand how pseudo-r-squared works, we first need to understand the concept of deviance and log-likelihoods in GLMs. Deviance is a measure of model fit based on likelihoods. It compares the fitted model to a null model (a model with only an intercept) and a saturated model (a model that perfectly fits the data). Let's break that down a bit further:

Residual deviance is calculated as:

$D = -2\left(\log L(\text{fitted model}) - \log L(\text{saturated model})\right)$

where - $\log L(\text{fitted model})$ is the log-likelihood of the fitted model and - $\log L(\text{saturated model})$ is the log-likelihood of the saturated model.

The log-likelihood measures how well the model explains the observed data; higher values indicate better fit. The fitted model is the model we have estimated, while the saturated model is a hypothetical model that perfectly fits the data.

Null deviance is calculated similarly, but for the null model:

$D_{null} = -2\left(\log L(\text{null model}) - \log L(\text{fitted model})\right)$

where - $\log L(\text{null model})$ is the log-likelihood of the null model.

(By the way, the -2 factor is included to make the deviance comparable to the chi-squared distribution, with degrees of freedom equal to the difference in the number of parameters between models. This is useful for hypothesis testing.)

So:

- **Null deviance**: This is the deviance of the null model, which includes only an intercept. It represents how well a model with no predictors fits the data.
- **Residual deviance**: This is the deviance of the fitted model. It represents how well the model with predictors fits the data relative to the null model.

One common pseudo-r-squared measure is McFadden's R-squared, defined as:

$R^2_{McFadden} = 1 - \frac{D_{fitted}}{D_{null}}$ where D_{fitted} is the residual deviance of the fitted model and D_{null} is the null deviance.

This measure ranges from 0 to 1, with higher values indicating better model fit. However, it is important to note that pseudo-r-squared values for GLMs are generally lower than r-squared values in linear regression, and they should be interpreted with caution.

There is much more to learn about the methods surrounding GLMs and their diagnostics, but this introduction should give you a solid foundation to start working with count data in R using Poisson regression.

Binomial data (L9)

Learning objectives

After this unit, students will be able to:

1. Identify binary response variables (e.g. presence–absence, success–failure) in described biological studies.
2. Distinguish individual-level binary data from aggregated binomial data (number of successes out of a number of trials).
3. Explain why linear regression on proportions, or Poisson regression on the number of successes, is unsuitable for binary and binomial responses.
4. Explain the logit link and the relationship between probability, odds and log-odds.
5. Convert between probabilities, odds and log-odds.
6. Use R to fit a logistic regression with `glm()`.
7. Use R to fit a binomial GLM to aggregated data, specifying the numbers of successes and failures.
8. Interpret coefficients as changes in log-odds, and back-transform them to odds ratios.
9. Use R to calculate predicted probabilities and plot the fitted relationship.
10. Use deviance-based tests to judge whether explanatory variables are associated with the response.
11. Check a binomial GLM before interpreting it, using the GLM checklist, and recognise which checks are hard to interpret for individual-level 0/1 data.
12. Use the ratio of residual deviance to residual degrees of freedom to check aggregated binomial data for overdispersion, use a quasibinomial model as a pragmatic fix, and recognise that this check is not informative for individual-level binary data.
13. Recognise separation in a logistic regression from its symptoms (e.g. very large coefficient estimates and standard errors).
14. Recognise a correct and complete statement of results from a logistic

regression.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch09_glm_binary_data.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 9 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch09_glm_binary_data.R")
```

Video walkthrough: coming soon.

Introduction

In the previous chapter we moved from linear models and normal errors to generalized linear models (GLMs) to handle count responses with a Poisson GLM. In this chapter we move to another common case: **binary** and **binomial responses**. Because binary data is a special case of binomial data, we will often refer to both as "binomial data".

At the heart of binomial data is an event that happens or does not happen. Examples include:

- Disease status: infected (1) or not infected (0)
- Survival: dead (1) or alive (0)
- Presence/absence of a species at a site: present (1) or absent (0)
- Success/failure of a treatment: success (1) or failure (0)

The central question is therefore:

Which explanatory variables influence the probability of the event happening.

Overview

In this chapter we will cover:

- A review of Bernoulli and binomial distributions (the family).
- Probabilities, odds, log-odds, odds ratios (the link function).
- Two examples of binomial GLM.
- Interpreting coefficients (odds ratios and probabilities).
- Model assumptions, deviances, and (some) model checking.
- Overdispersion in *aggregated* binomial data and the quasibinomial fix.
- Practical complications for individual-level (non-aggregated) 0/1 data.

Binary and binomial data

When we model binary/binomial data, we are interested in the **probability** of an event happening. For example, the probability of a heart attack, or the probability of a species being present at a site. And we are interested in whether such probabilities differ between groups, or vary along a continuous explanatory variable.

Let us label this probability π. So π is the probability of "yes" (e.g., pass a test), and $1 - \pi$ is the probability of "no" (e.g., fail a test).

So, if the probability π is 0.8, then there is an 80% chance of "yes" and a 20% chance of "no". In R we can simulate the pass or fail of each of a group of people with an expected pass rate (probability) of 0.8 like this:

```
set.seed(123)
number_persons <- 10
y_binary <- rbinom(number_persons, size = 1, prob = 0.8)
```

This code says please draw 10 random numbers from a binomial distribution where each number is the result of 1 trial (size = 1) with a probability of success of 0.8 (prob = 0.8). Because size = 1, each number will be either 0 or 1.

Hence, the object `y_binary` contains 10 simulated persons, where each person has an 80% chance of being 1:

```
y_binary
```

```
 [1] 1 1 1 0 0 1 1 0 1 1
```

What we have done is to simulate 10 *Bernoulli trials* with probability 0.8. A Bernoulli trial is one random experiment with exactly two possible outcomes, "success" (1) and "failure" (0), where the probability of success is π.)

You can see that the observations in `y` are either 0 or 1: they are `binary` data.

An alternate method to simulate and express the same kind of data is binomial data, where we count the number of successes (1s) in a number of trials. In R, we simulate this like so:

```
set.seed(123)
number_persons <- 10
y_binomial <- rbinom(1, size = number_persons, prob = 0.8)
```

This code says please draw 1 random number from a binomial distribution where the number of trials is 10 (size = 10) with a probability of success of 0.8 (prob = 0.8). The result is the number of successes (1s) in those 10 trials, in this particular simulation it is:

```
y_binomial
```

```
[1] 9
```

We know to expect that 8 of the 10 persons will be 1 (success), because the probability of success $\pi = 0.8$. But the actual number of successes in any one simulation can vary due to chance. In the binary data example above we by chance got

```
sum(y_binary)
```

```
[1] 7
```

successes (1s) out of 10 persons. In the binomial data example above we got:

```
y_binomial
```

```
[1] 9
```

So in neither case did we get exactly 8 successes! What is the chance of that!

Note also that the two simulations give different numbers of successes (7 and 9), even though both used `set.seed(123)`, the same number of persons and the same probability. This is because they are different random draws: `rbinom(10, size = 1, ...)` and `rbinom(1, size = 10, ...)` use the random numbers in a different way, so they do not produce the same result. Both are valid simulations of the same process.

Well, we can calculate the probability of getting exactly k successes in n trials with success probability π using the binomial probability formula:

$P(Y = k) = \binom{n}{k}\pi^k(1-\pi)^{n-k}$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient, which counts the number of ways to choose k successes from n trials.

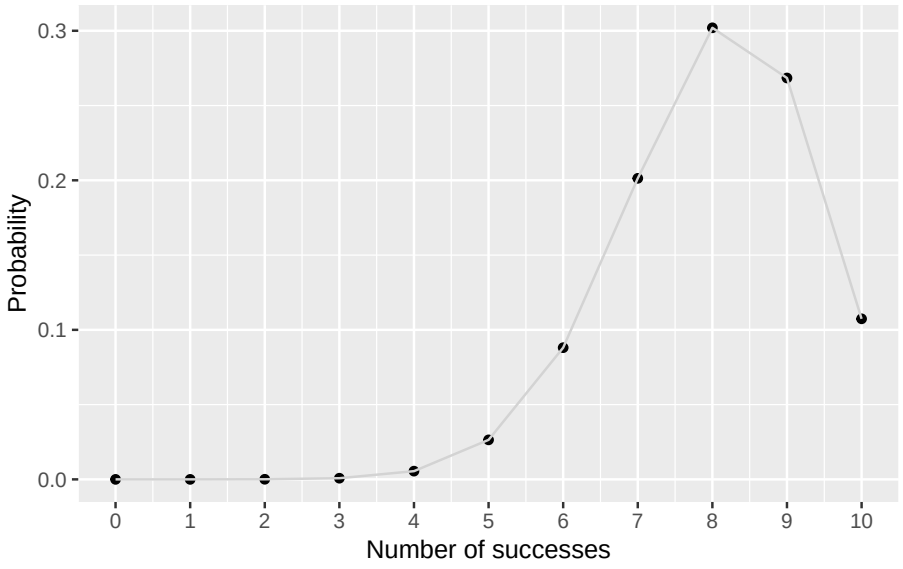
In R, we can calculate the probability of getting exactly 8 successes in 10 trials with success probability 0.8 as:

```
dbinom(8, size = 10, prob = 0.8)
```

```
[1] 0.3019899
```

And we can make a graph of the full probability distribution for getting k successes in 10 trials with success probability 0.8:

```
p_k_successes <- dbinom(0:10, size = 10, prob = 0.8)
ggplot(mapping = aes(x = 0:10, y = p_k_successes)) +
  geom_point() +
  geom_line(col="lightgrey") +
  scale_x_continuous(breaks = seq(0, 10, by = 1)) +
  labs(x = "Number of successes", y = "Probability")
```



So the probability of getting exactly 8 successes is actually quite low, at about 0.3. But it is still the most likely outcome.

! Important

So what??? You should now understand two important things:

- The relationship between binary data (0/1) and binomial data (counts of successes out of trials).
- That the binomial distribution is appropriate for modelling both binary and binomial data. Hint: in a GLM, we will therefore use the `binomial` family.

Probabilities, odds, log-odds, and odds ratios

Odds

OK, so we know that when we analyse binary/binomial data, we are interested in the **probability** of an event happening, denoted by π.

A related quantity is the **odds** of the event happening. The odds is defined as the ratio of the probability of the event happening to the probability of it not happening:

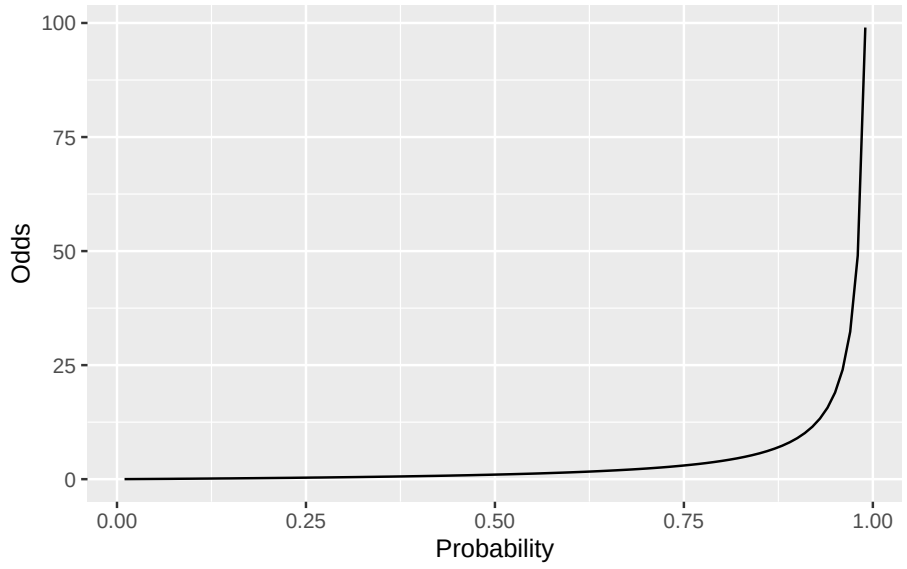
- **Odds** is $\pi/(1-\pi)$

We sometimes say "the odds of an event happening are x to y". For example, a horse has 4 to 1 odds of winning a race. This means that in 5 races we expect the horse to win 4 times and lose once. The odds are $4/1 = 4$. The probability of winning is then $4/(4+1) = 0.8$.

If the horse was really bad and had 1 to 4 odds of winning, then the odds of winning would be $0.2/0.8 = 0.25$, and the probability of winning would be 0.2.

So, odds and probabilities are related mathematically, but they are not the same thing. Lets look at a graph of probability on the x-axis and odds on the y-axis:

```
prob_seq <- seq(0.01, 0.99, by = 0.01)
odds_seq <- prob_seq / (1 - prob_seq)
ggplot(mapping = aes(x = prob_seq, y = odds_seq)) +
  geom_line() +
  labs(x = "Probability", y = "Odds")
```



So, when we transform probabilities to odds, we go from a bounded scale (0 to

1) to an unbounded scale (0 to infinity).

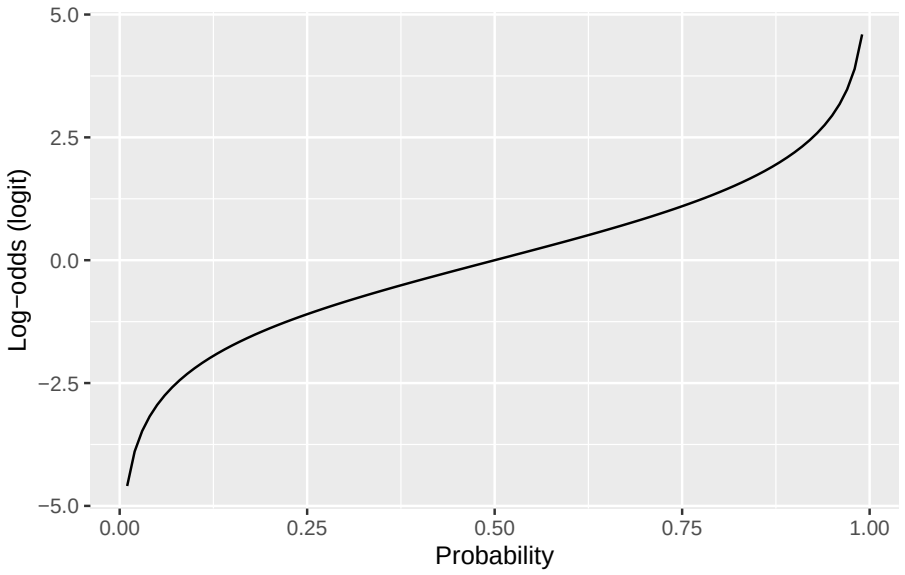
Log-odds (logit)

Another useful transformation is the **log-odds**, also known as the **logit**:

- **Log-odds (logit)** is $\log\left(\frac{\pi}{1-\pi}\right)$

Here is the relationship between probability and log-odds:

```
log_odds_seq <- log(odds_seq)
ggplot(mapping = aes(x = prob_seq, y = log_odds_seq)) +
  geom_line() +
  labs(x = "Probability", y = "Log-odds (logit)")
```



> **!** Important
>
> **So what???**
> The reason why the logit is useful is that it maps probabilities (0 to 1) to the whole real line (-infinity to infinity). This is important because in a GLM, the linear predictor can take any value from -infinity to infinity, so we need a link function that can map between the two scales. The logit does this perfectly.

Odds ratio

Finally, we often want to compare the odds of an event happening in two different groups. Say we have two groups, group 1 and group 2, with probabilities π_1 and

π_2. The odds in each group are:

- Odds in group 1: $\frac{\pi_1}{1-\pi_1}$
- Odds in group 2: $\frac{\pi_2}{1-\pi_2}$

The **odds ratio (OR)** compares the odds in two groups:

Odds Ratio = OR = $\frac{\text{odds in group 1}}{\text{odds in group 2}}$.

Let's go through a couple of examples. First, suppose the probability of success is 0.2 in both groups. Then the odds in both groups are:

- Odds in group 1 and 2: $\frac{0.2}{1-0.2} = \frac{0.2}{0.8} = 0.25$

So the odds ratio is:

$OR = \frac{0.25}{0.25} = 1$

Now take an example where the probability of success in group 1 is 0.4, and in group 2 it is 0.2. Then the odds in each group are:

- Odds in group 1: $\frac{0.4}{1-0.4} = \frac{0.4}{0.6} = 0.667$
- Odds in group 2: $\frac{0.2}{1-0.2} = \frac{0.2}{0.8} = 0.250$

So the odds ratio is:

$OR = \frac{0.667}{0.250} = 2.67$

This means that the odds of success in group 1 are about 2.67 times the odds of success in group 2.

> **!** Important
>
> **So what???**
> You will see below that the coefficients in a binomial GLM with logit link correspond to log-odds ratios. Exponentiating those coefficients gives odds ratios. This is a key part of interpreting the results of a binomial GLM. For example, when we see a coefficient (e.g., slope) close to zero, that means the log-odds ratio is close to zero, which means the odds ratio is close to 1, which means there is little difference in odds between the two groups.

A simple binomial GLM

We now have the pieces to define a binomial GLM:

1. **Family**: $y_i \sim \text{Binomial}(n_i, \pi_i)$
2. **Linear predictor**: $\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \cdots + \beta_p x_i^{(p)}$
3. **Link**: logit link: $\eta_i = \log\left(\frac{\pi_i}{1-\pi_i}\right)$

Let us make a simple example in which there we are interested in how the probability of passing a test differs between children and adults. We will simulate the data, and then fit a binomial GLM.

To simulate that data we will define the number of children and adults, and the probability of passing the test in each group:

```
set.seed(8761)
n_children <- 50
n_adults <- 50
p_pass_children <- 0.8
p_pass_adults <- 0.4
```

Now let us simulate whether each person passes or fails, using the `rbinom()` function:

```
y_children <- rbinom(n_children, size = 1, prob = p_pass_children)
y_adults <- rbinom(n_adults, size = 1, prob = p_pass_adults)
```

Now we can combine the data into a data frame (tibble), and look at the data:

```
pass_data <- tibble(
  age_group = rep(c("child", "adult"), each = 50),
  pass = c(y_children, y_adults)
)
pass_data
```

```
# A tibble: 100 x 2
   age_group  pass
   <chr>     <int>
 1 child         1
 2 child         0
 3 child         1
 4 child         1
 5 child         1
 6 child         1
 7 child         0
 8 child         1
 9 child         1
10 child         1
# i 90 more rows
```

We can summarize the data to see how many passed in each group:

```
pass_summary <- pass_data |>
  group_by(age_group) |>
  summarise(
    n_tested = n(),
    n_passed = sum(pass),
```

```
    pass_rate = mean(pass)
  )
pass_summary
```

```
# A tibble: 2 x 4
  age_group n_tested n_passed pass_rate
  <chr>        <int>    <int>     <dbl>
1 adult           50       30      0.6
2 child           50       39      0.78
```

In the simulated experiment, 78% of the children passed, when the true probability of passing was 0.8. The adults did really well! – 60% of the adults passed, when the true probability was 0.4.

If we had collected that data for real, we might be interested in if the difference in pass rates between children and adults is statistically significant. We can use a binomial GLM to test that. Here is how we specify and fit the model in R:

```
m_pass_age_group <- glm(pass ~ age_group, data = pass_data, family = binomial(link = "l
```

The critical part here is the `family = binomial(link = "logit")` argument, which tells R to fit a binomial GLM with a logit link function. Note that if we do not specify the link function, R will use the logit link by default for the binomial family. This means that the following model specification is equivalent:

```
m_pass_age_group <- glm(pass ~ age_group, data = pass_data, family = binomial)
```

We can look at the estimated coefficients to see the result:

```
coef(summary(m_pass_age_group))
```

```
                Estimate Std. Error  z value   Pr(>|z|)
(Intercept)    0.4054651  0.2886751 1.404572 0.16014849
age_groupchild 0.8602013  0.4470831 1.924030 0.05435084
```

Important: The estimates are on the scale of the linear predictor, which is the logit (log-odds) scale. The intercept corresponds to the log-odds of passing for the reference group (adults), and the coefficient for `age_groupchild` corresponds to the difference in log-odds between adults and children.

So the logit (log-odds) for adults to pass is 0.405. To convert that to a probability, we can use the inverse logit function:

$\pi = \frac{\exp(\eta)}{1+\exp(\eta)}$

Calculating that in R:

```
eta_adults <- coef(m_pass_age_group)[["(Intercept)"]]
p_adults <- exp(eta_adults) / (1 + exp(eta_adults))
p_adults
```

```
[1] 0.6
```

Perfect. It is as we saw when we summarised the data – this is the observed pass rate for adults.

To find the probability of passing for children, we calculate the logit value for children by adding the coefficient for `age_groupchild` to the intercept:

Logit for children = 0.405 + 0.860 = 1.265

Then we convert that to a probability:

```
eta_children <- eta_adults + coef(m_pass_age_group)[["age_groupchild"]]
p_children <- exp(eta_children) / (1 + exp(eta_children))
p_children
```

```
[1] 0.78
```

Perfect again.

The odds ratio

It is useful to know about and use something called the **odds ratio**. The odds ratio compares the odds of an event happening in two situations. In our example, we can calculate the odds ratio for passing the test for children compared to adults.

Let's use the logit values we calculated above, which we have to exponentiate to transform back to odds:

- Odds for adults: exp(0.405)
- Odds for children: exp(1.265)

The odds ratio (OR) for children compared to adults is:

$$\exp(1.265)/\exp(0.405) = \exp(1.265 - 0.405) = \exp(0.860) = 2.363$$

In the line above, notice the exp(0.860) term. The number 0.860 is exactly the coefficient for `age_groupchild` in the model output. So we can get the odds ratio directly from the model coefficients by exponentiating the coefficient for `age_groupchild`. Put another way, the coefficient for `age_groupchild` is the log-odds ratio for children compared to adults.

This isn't magic. Its just a straightforward consequence of how the logit link function works. In the following example of binomial regression, you will see how with a continuous explanatory variable, the coefficient (slope) corresponds to the change in log-odds for a one-unit increase in the explanatory variable, and exponentiating that coefficient gives the odds ratio for a one-unit increase.

Aggregated binomial data

Let's work though an example of binomial regression, with data in what we call *aggregated* format. This simply means that we have counts of successes and failures for groups of individuals, rather than individual-level 0/1 data.

Eight groups of beetles were exposed to an insecticide dose for 5 hours. For each dose level, we know how many beetles were tested and how many died. This is **binomial (aggregated)** data.

Read in the beetle dataset:

```
beetle <- read_csv("datasets/beetle_mortality.csv")
```

```
Rows: 8 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (4): Dose, Number_tested, Number_killed, Mortality_rate

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

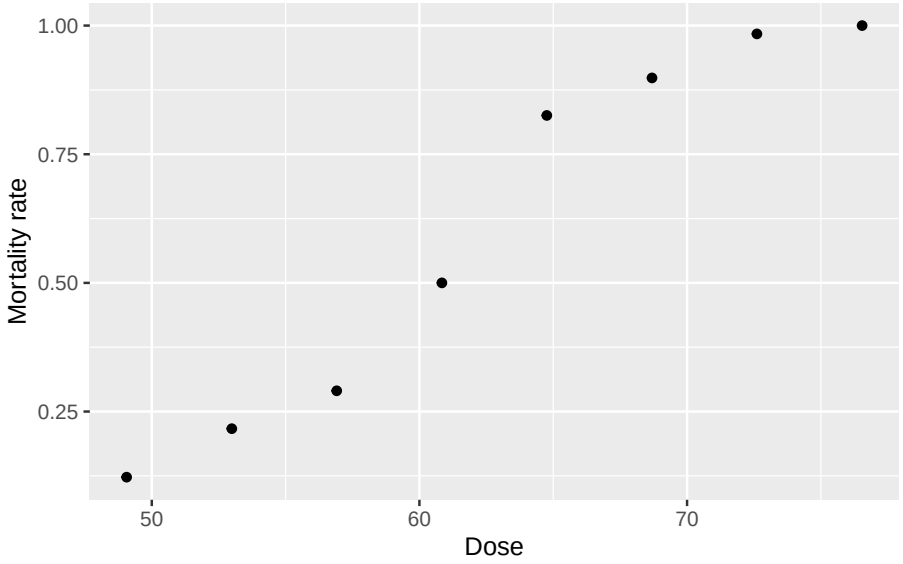
Here are the first few rows of the beetle dataset:

```
head(beetle)
```

```
# A tibble: 6 x 4
   Dose Number_tested Number_killed Mortality_rate
  <dbl>         <dbl>         <dbl>          <dbl>
1  49.1            49             6          0.122
2  53.0            60            13          0.217
3  56.9            62            18          0.290
4  60.8            56            28          0.5
5  64.8            63            52          0.825
6  68.7            59            53          0.898
```

It contains one explanatory variable (Dose) and two pieces of information about the binary response variable (Number_killed and Number_tested). The mortality rate is also included, and is the proportion killed at each dose (Number_killed / Number_tested).

As always, start with a graph.



Wrong analysis 1: linear regression on mortality rate

```
mod_beetle_lm <- lm(Mortality_rate ~ Dose, data = beetle)
```

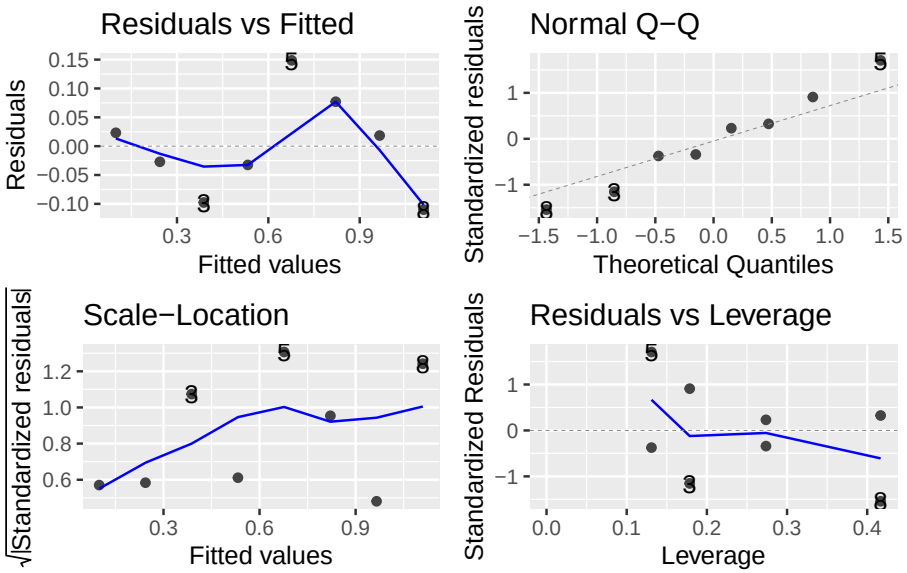
This can predict probabilities below 0 or above 1 and does not respect the mean–variance structure of binomial data.

Check out the model assumptions:

```
#par(mfrow = c(2, 2))
#plot(mod_beetle_lm)
autoplot(mod_beetle_lm)
```

```
Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.
i Please use `broom::augment(<lm>)` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```



We don't have so many data points here, but the residuals vs fitted indicating non-linearity and the Q-Q plot indicating non-normality are both concerning. Also the scale-location plot suggests non-constant variance.

Wrong analysis 2: Poisson regression on counts killed

A Poisson model ignores the fact that deaths are bounded by the number tested at each dose.

```
mod_beetle_pois <- glm(Number_killed ~ Dose, data = beetle, family = poisson)
summary(mod_beetle_pois)$coefficients
```

```
             Estimate  Std. Error   z value     Pr(>|z|)
(Intercept) -0.77418056 0.494639994 -1.565139 1.175502e-01
Dose         0.06678281 0.007240113  9.224002 2.862145e-20
```

One issue that can now happen is that we can get "impossible" predictions: predicted counts killed that exceed the number tested.

```
# Example of the "impossible" issue:
dose_example <- 76.54
pred_killed <- predict(mod_beetle_pois,
                       newdata = data.frame(Dose = dose_example),
                       type = "response")
pred_killed
```

```
       1
76.50652
```

```
num_tested_example <- beetle$Number_tested[beetle$Dose == dose_example]
num_tested_example
```

```
[1] 60
```

At the highest dose (76.54), the model predicts about 76 deaths, but only 60 beetles were tested! Clearly this is an impossible prediction. We can't go on with the Poisson model either.

> Caution
>
> **Poisson vs binomial, when to use each**
> Use a Poisson model when counts have *no known upper limit*.
> Use a binomial model when counts are "k successes out of n trials". Use a binomial/binary model when each observation is a 0/1 response.

Doing it right!

For aggregated binomial data, we provide **successes and failures**:

```
beetle <- beetle |>
  mutate(Number_survived = Number_tested - Number_killed)
head(beetle)
```

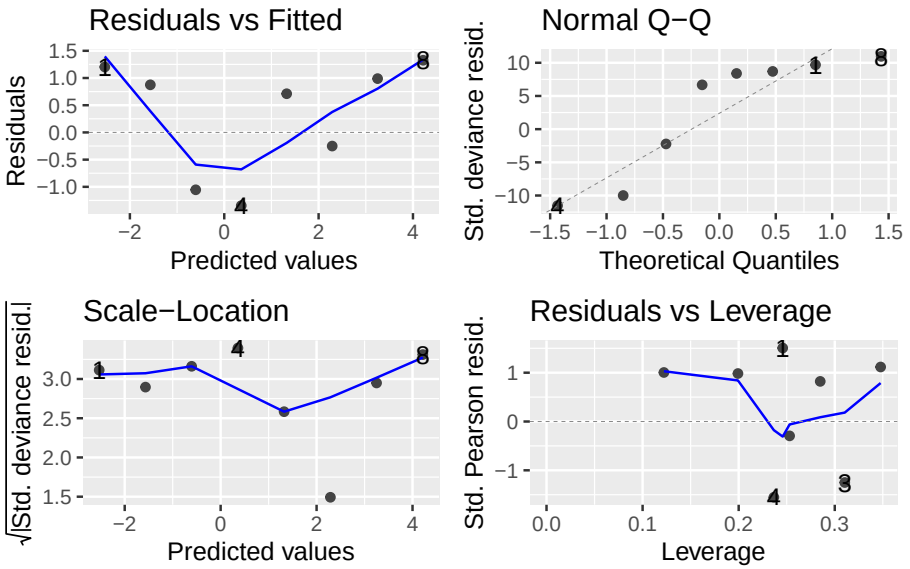
```
# A tibble: 6 x 5
   Dose Number_tested Number_killed Mortality_rate Number_survived
  <dbl>         <dbl>         <dbl>          <dbl>           <dbl>
1  49.1            49             6          0.122              43
2  53.0            60            13          0.217              47
3  56.9            62            18          0.290              44
4  60.8            56            28          0.5                28
5  64.8            63            52          0.825              11
6  68.7            59            53          0.898               6
```

Now we can fit the logistic regression model, being careful to specify `family = binomial` and to use `cbind(successes, failures)` for the response. Because we specify the response this way, R knows we are working with aggregated binomial data. And because we specify `family = binomial`, R knows to use the logit link (the default).

```
beetle_glm <- glm(cbind(Number_killed, Number_survived) ~ Dose,
                  data = beetle, family = binomial)
```

And as usual we can check the model assumptions:

```
#par(mfrow = c(2, 2))
#plot(beetle_glm)
autoplot(beetle_glm)
```



Not great, but again we have only 8 dose levels. Let us continue in any case and look at the ANOVA table and the coefficients.

> **!** Important
>
> **The response in aggregated binomial data**
> Use `cbind(successes, failures)` in `glm(..., family = binomial)`.
> - successes: number of 1s
> - failures: number of 0s

Analysis of deviance (likelihood-based ANOVA)

As for count-data GLMs, the estimation of the coefficients is by maximum likelihood estimation (MLE). The measure of goodness-of-fit is the deviance. We can use an analysis of deviance table to see how much deviance is explained by dose.

```
anova(beetle_glm, test = "Chisq")
```

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: cbind(Number_killed, Number_survived)

Terms added sequentially (first to last)
```

```
     Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                     7    267.662
Dose  1   259.23         6      8.438 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see a table quite similar to that for the Poisson GLM. In this case the Dose row shows a huge amount of the deviance (259.23 units) is explained by dose, and so the p-value is very small.

Recall that we already used a chi-squared test for the Poisson GLM. We do the same here for the binomial GLM, because we are working with models estimated by maximum likelihood. So just as with the Poisson GLM, we use a chi-squared test here because, under certain regularity conditions, the change in deviance between nested models follows a chi-squared distribution with degrees of freedom equal to the difference in the number of parameters. If you are interested in what this means, you can read more about it in more advanced statistics textbooks mentioned in the *Further reading* section at the end of this chapter.

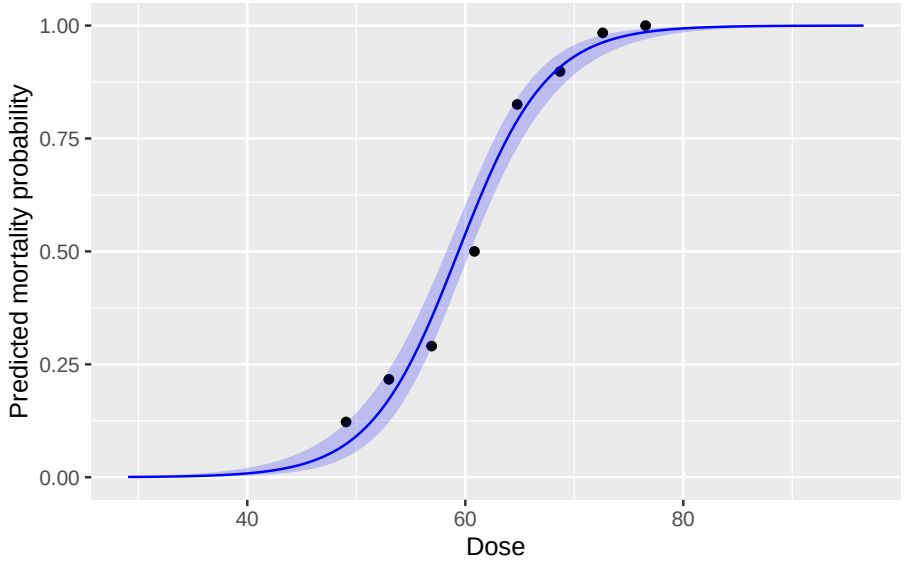
Plotting the fitted relationship

We will plot $P(Y = 1)$ (mortality probability) against dose using model predictions.

```
dose_grid <- tibble(Dose = seq(min(beetle$Dose)-20, max(beetle$Dose)+20, length.out = 400))
beetle_predictions <- predict(beetle_glm, newdata = dose_grid, se.fit = TRUE) |>
  as_tibble() |>
        mutate(p_hat = exp(fit) / (1 + exp(fit)),
        p_hat_upper_2se = exp(fit + 2*se.fit) / (1 + exp(fit + 2*se.fit)),
        p_hat_lower_2se = exp(fit - 2*se.fit) / (1 + exp(fit - 2*se.fit))
       )
dose_grid <- bind_cols(dose_grid, beetle_predictions)
```

And now the plot:

```
ggplot() +
  geom_point(data = beetle, aes(x = Dose, y = Mortality_rate)) +
  geom_line(data = dose_grid, aes(x = Dose, y = p_hat), color = "blue") +
  geom_ribbon(data = dose_grid,
              aes(x = Dose, ymin = p_hat_lower_2se, ymax = p_hat_upper_2se),
              alpha = 0.2, fill = "blue") +
  labs(x = "Dose", y = "Predicted mortality probability")
```



The plot is intentionally made to show what happens at very low and very high dose. At very low dose, the predicted mortality probability approaches 0, and at very high dose it approaches 1. This is a key feature of binomial regression: the predicted probabilities are always between 0 and 1. No matter the value of the linear predictor, the inverse logit transformation always maps it to a probability between 0 and 1.

We also see that the standard errors are larger at intermediate doses, where the slope of the curve is steepest. And the standard errors are smaller at very low and very high doses, where the curve flattens out.

Reporting

To report a predicted probability with its confidence interval, we calculate the prediction and its standard error on the link (log-odds) scale, form the interval there, and then back-transform both to the probability scale with `plogis()` (the inverse of the logit). This is the same approach we used for the Poisson GLM, where we back-transformed with `exp()`. Calculating the interval on the link scale first ensures that it stays between 0 and 1.

```
pred_50 <- predict(beetle_glm, newdata = data.frame(Dose = 50),
                   type = "link", se.fit = TRUE)
prob_50 <- plogis(pred_50$fit)
prob_50_ci <- plogis(pred_50$fit + c(-1.96, 1.96) * pred_50$se.fit)
prob_50
```

```
         1
0.09103457
```

```
prob_50_ci
```

```
[1] 0.05741714 0.14138248
```

When reporting results from a logistic regression, you might say something like:

"A logistic regression was used to model the probability of beetle mortality as a function of insecticide dose. The odds of mortality increased by a factor of 1.28 (95% CI: 1.23 to 1.34) per unit increase in dose. This indicates that higher doses are associated with higher odds of mortality. The model predicts that at a dose of 50, the probability of mortality is approximately 0.09 (95% CI: 0.06 to 0.14). Mortality was strongly associated with dose (likelihood ratio test: change in deviance = 259.2, $df = 1$, $p < 0.001$)."

> Tip
>
> **How big is the effect?** Odds ratios are hard for most readers to interpret. As well as the odds ratio, report predicted probabilities at a few meaningful values of the explanatory variable (for example, the lowest and highest doses used, or a dose of practical interest), with their confidence intervals. Readers can then judge whether the effect is biologically (or practically) important. See also "Probability vs odds ratio in reporting" below.

Overdispersion in aggregated binomial data

Overdispersion means the variability is larger than the binomial model assumes. A quick (rough) check for aggregated binomial data is:

Residual deviance $\approx$ df

Values much larger than 1 for the ratio $\frac{\text{Residual deviance}}{\text{df}}$ suggest overdispersion. In practice we look for values above about 1.5 or 2.

In the beetle mortality example:

```
deviance(beetle_glm)
```

```
[1] 8.437898
```

```
df.residual(beetle_glm)
```

```
[1] 6
```

```
deviance(beetle_glm) / df.residual(beetle_glm)
```

```
[1] 1.406316
```

The ratio is about 1.4, which is suggests some overdispersion, but is on the limit of what is usually considered acceptable.

Caution

Type I error inflation (binomial overdispersion)
If there is overdispersion and we ignore it, standard errors are usually too small and p-values can be too small (anti-conservative). That increases false positives (Type I errors).

Quasibinomial as a pragmatic fix

If we had found overdispersion, a simple fix is to use the `quasibinomial` family. This estimates an extra dispersion parameter from the data. For example:

```
beetle_glm_q <- glm(cbind(Number_killed, Number_survived) ~ Dose,
                    data = beetle, family = quasibinomial)
summary(beetle_glm_q)
```

```
Call:
glm(formula = cbind(Number_killed, Number_survived) ~ Dose, family = quasibinomial,
    data = beetle)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -14.57806    1.46611  -9.943 5.98e-05 ***
Dose          0.24554    0.02427  10.118 5.42e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasibinomial family taken to be 1.274895)

    Null deviance: 267.6624  on 7  degrees of freedom
Residual deviance:   8.4379  on 6  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4
```

The estimates are the same as for `beetle_glm`, but the standard errors are larger (for Dose, 0.0243 instead of 0.0215). This is because they are scaled by the estimated dispersion parameter (1.27; see the line "Dispersion parameter for quasibinomial family taken to be …"). This value is calculated from the Pearson residuals, so it differs a little from the deviance / df ratio of about 1.4 above, but it tells us the same thing. The tests are now t-tests rather than z-tests. Here the effect of Dose remains very clear.

> **Note**
>
> `quasibinomial` estimates an extra dispersion parameter from the data. It is often a good "first fix" when aggregated binomial data look overdispersed.

Individual-level binary data (non-aggregated)

In some datasets we record a single 0/1 observation per individual, rather than aggregated counts for many individuals. The same logistic regression model is used, but:

- simple plots of y against explanatory variables look uninformative (two bands)
- some assumption and dispersion checks need extra care

Example: blood screening (ESR)

Individuals with low ESR (ESR is erythrocyte sedimentation rate and is a measure of general inflammation and infection) are generally considered healthy; ESR $>$ 20 mm/hr indicates possible disease. We will model the probability of high ESR using fibrinogen and globulin concentration.

Read in the plasma dataset:

```
plasma <- read_csv("datasets/plasma_esr_original_HSAUR3.csv")
```

```
Rows: 32 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): ESR
dbl (3): fibrinogen, globulin, y

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the data:

```
plasma
```

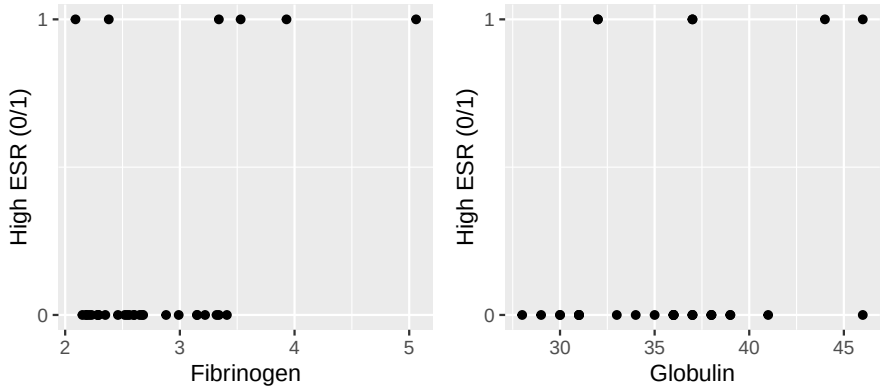
```
# A tibble: 32 x 4
   fibrinogen globulin ESR          y
        <dbl>    <dbl> <chr>    <dbl>
 1       2.52       38 ESR < 20     0
 2       2.56       31 ESR < 20     0
 3       2.19       33 ESR < 20     0
 4       2.18       31 ESR < 20     0
 5       3.41       37 ESR < 20     0
```

```
 6       2.46       36 ESR < 20     0
 7       3.22       38 ESR < 20     0
 8       2.21       37 ESR < 20     0
 9       3.15       39 ESR < 20     0
10       2.6        41 ESR < 20     0
# i 22 more rows
```

The ESR response variable is in two variables. `ESR` is a factor with levels "low" and "high". It is also present as a numeric variable `y`, coded 0 (low) and 1 (high). The explanatory variables are `fibrinogen` and `globulin`, both continuous.

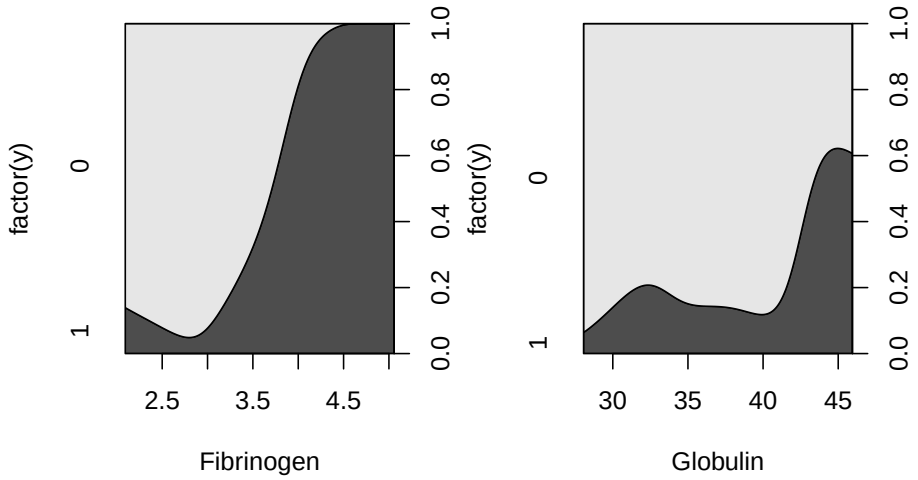
Complication 1: graphical description

Simple scatter plots of 0/1 data are not so informative:



A more informative plot is a conditional density plot. This shows the proportion of 0s and 1s at each value of the explanatory variable.

```
par(mfrow = c(1, 2))
cdplot(factor(y) ~ fibrinogen, data = plasma, xlab = "Fibrinogen")
cdplot(factor(y) ~ globulin, data = plasma, xlab = "Globulin")
```




```
par(mfrow = c(1, 1))
```

It looks like higher fibrinogen and higher globulin are associated with higher probability of high ESR. The pattern appears stronger for fibrinogen.

Fit the binomial regression model

In the case of non-aggregated individual-level 0/1 data, we simply use `y` as the response variable. We have to remember to specify `family = binomial` in `glm()`.

```
plasma_glm <- glm(y ~ fibrinogen + globulin, data = plasma, family = binomial)
```

We look at the results (we check the model below, but in practice you would check the model before interpreting the results):

```
summary(plasma_glm)
```

```
Call:
glm(formula = y ~ fibrinogen + globulin, family = binomial, data = plasma)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -12.7921     5.7963  -2.207   0.0273 *
fibrinogen    1.9104     0.9710   1.967   0.0491 *
globulin      0.1558     0.1195   1.303   0.1925
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 30.885  on 31  degrees of freedom
Residual deviance: 22.971  on 29  degrees of freedom
AIC: 28.971

Number of Fisher Scoring iterations: 5
```

```
exp(coef(plasma_glm))
```

```
 (Intercept)   fibrinogen     globulin
2.782735e-06 6.755579e+00 1.168567e+00
```

The coefficients are on the log odds scale. Higher fibrinogen is associated with higher odds of high ESR: each extra unit of fibrinogen multiplies the odds of high ESR by about 6.8, when globulin is held constant (z-test in the summary, $p = 0.049$). The estimated effect of globulin is also positive (odds multiplied by 1.17 per unit), but there is little evidence for it once fibrinogen is in the model ($p = 0.19$). This agrees with the conditional density plots above. Note that with only 32 people, the estimates are quite uncertain (large standard errors).

Complication 2: model checking and dispersion

The general checklist for checking a GLM, from Chapter 8, applies here too:

i Checklist: checking a GLM

1. **Is the family right for the response variable?** Counts: Poisson (or quasi-Poisson / negative binomial if overdispersed). Binary data, or successes out of a number of trials: binomial. No diagnostic plot can rescue a model with the wrong family.
2. **Are the observations independent?** This comes from the study design, not from the plots. If observations are grouped or repeated, a mixed model is needed (Chapter 11).
3. **Residuals vs fitted values.** Look for strong patterns or curvature, which suggest a missing explanatory variable, a missing interaction, or a non-linear relationship. For individual-level 0/1 data this plot shows two bands of points and is hard to interpret.
4. **Dispersion.** The ratio of residual deviance to residual degrees of freedom should be roughly 1. Values above about 1.5–2 suggest overdispersion (consider quasi-Poisson, negative binomial or quasibinomial). This check works for counts and aggregated binomial data, but not for individual-level 0/1 data.
5. **Influential observations.** As for linear models, look at the residuals

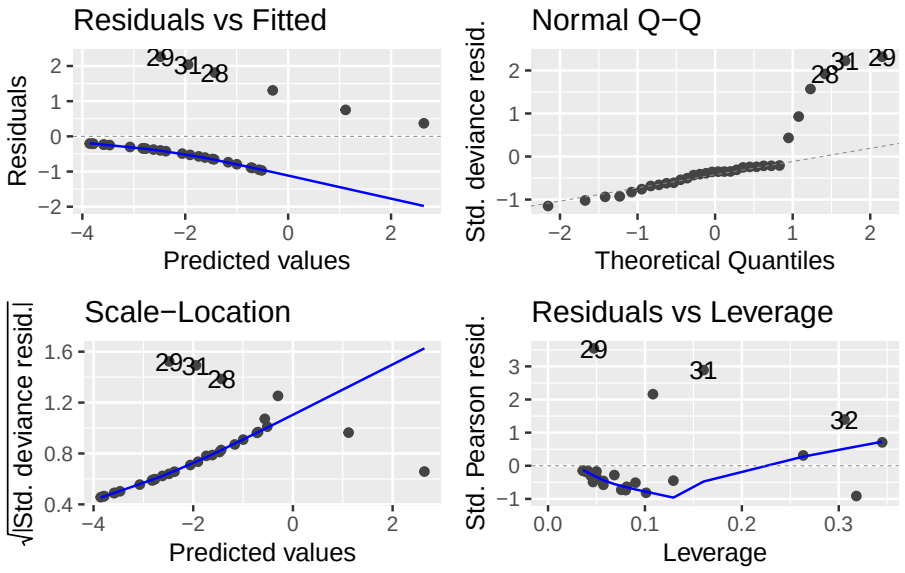
vs leverage plot (Cook's distance).

6. **QQ-plot of the deviance residuals.** A GLM does not assume normally distributed residuals, so this is only a rough guide. Large departures can point to a wrong family or to overdispersion.
7. **Zeros (count data).** Many more zeros than the Poisson model predicts suggests zero inflation.

For individual-level 0/1 data, points 3 and 4 need particular care:

Residual plots exist, but can be hard to interpret for 0/1 data:

```
#par(mfrow = c(2, 2))
#plot(plasma_glm)
autoplot(plasma_glm)
```



Important

Dispersion checks need caution for individual 0/1 data
The simple "residual deviance ≈ residual df" rule-of-thumb is most useful for *aggregated* binomial data. With individual-level 0/1 data, detecting overdispersion is more subtle and often requires additional structure (e.g. grouping, random effects).

Two common practical issues

1. Separation

Sometimes a explanatory variable (or combination of explanatory variables) perfectly predicts the response variable (all 0s on one side, all 1s on the other). This is called **(complete) separation** and can cause extremely large coefficient estimates and warnings.

> Caution
>
> **Separation warning**
> If your logistic regression produces enormous standard errors or warnings about fitted probabilities being 0 or 1, check for separation. Remedies include more data, simpler models, or penalized methods (advanced topic).

2. Probability vs odds ratio in reporting

Log odds ratios are the default "native" scale of the coefficients logistic regression, but probabilities are often easier to interpret. In practice, you may report odds ratios **and** translate to probabilities at meaningful explanatory variable values.

Naming models (logistic regression)

Logistic regression is a very important term. It is a binomial GLM with logit link. I.e., it is what we have just described above!

The reason it (a binomial GLM with logit link) is called **logistic regression** is that the inverse of the logit function is called the logistic function. So logistic regression is just a special case of binomial GLM. The logistic function looks like a "squashed" S-shaped curve that maps the whole real line to the interval (0, 1). This is perfect for modelling probabilities.

Just as with linear models, and with Poisson GLMs, we can include multiple explanatory variables in a binomial GLM. The interpretation of coefficients is similar to that in linear models, but on the log-odds scale. We don't always have well recognised names, like we do with linear models (e.g., ANCOVA, multiple regression). But the principles are the same.

Because we don't have well recognised names for different types of binomial GLM with multiple explanatory variables, we just call them "binomial GLMs" or "logistic regression models", and then describe the explanatory variables included. For example:

- A logistic regression model with one continuous and one categorical explanatory variable.

- A logistic regression model with two continuous explanatory variables and their interaction.
- A logistic regression model with three categorical explanatory variables and all possible interactions (a full factorial model, analogous to a three-way ANOVA).

Review

There is a lot more to logistic regression and binary data. We can of course make models to analyse more complex data and questions, including multiple explanatory variables, different types of explanatory variable (continuous and categorical), interactions, and so on. But the key ideas are all above. To summarize the main points:

- Data can be either aggregated binomial (successes out of trials) or individual-level 0/1.
- The key questions are about how explanatory variables influence the probability of "yes".
- The distribution family is binomial (Bernoulli for individual-level data).
- The link function is the logit (log-odds).
- If we find overdispersion in aggregated binomial data, `quasibinomial` is a pragmatic fix.
- With non-aggregated individual-level 0/1 data, plotting and checking model assumptions require more care.

Further reading

- Chapter 17 of *The R Book* by Crawley (2012) provides a comprehensive introduction to GLMs in R, including binomial regression. The R is a bit old fashioned but the concepts are well explained.
- Generalized Linear Models With Examples in R (2018) by Peter K. Dunn and Gordon K. Smyth.

Ordination (L10)

Learning objectives

After this unit, students will be able to:

1. Recognise when a biological question involves many response variables, and explain the limitations of analysing each one separately.
2. Reshape multivariate data between wide and long format, and explain which format is needed for which task.
3. Explain the aim of ordination: to summarise similarity among observations measured on many variables using a few axes.
4. Explain why variables are centred, and usually scaled, before a PCA.
5. Use R to perform a principal component analysis (PCA) and obtain the proportion of variation explained by each axis.
6. Interpret PCA loadings and biplots to identify which variables contribute most to each axis.
7. Choose a distance measure for NMDS appropriate to the data (e.g. Euclidean distance on scaled numeric variables).
8. Use R to perform a non-metric multidimensional scaling (NMDS) and obtain its stress value.
9. Interpret ordination plots in terms of similarities and differences among observations.
10. Distinguish PCA from NMDS, and choose between them for a given dataset.
11. Evaluate what an ordination does and does not show (e.g. it summarises patterns but does not by itself test hypotheses; tests need separate methods).
12. Choose an appropriate approach to test whether groups differ in multivariate composition (MANOVA, PERMANOVA) and in multivariate spread (PERMDISP), and to relate an explanatory variable to an ordination (envfit, ordisurf).
13. Use principal components as explanatory variables in a regression, as one way to deal with collinearity.

These learning objectives, together with the content of this chapter, define

what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch10_ordination.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 10 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch10_ordination.R")
```

Video walkthrough: coming soon.

Introduction

In earlier chapters, we have analysed variability in **one response variable at a time**. But many biological questions are intrinsically **multivariate**:

- morphology: multiple measurements describe *shape* (not just size)
- communities: abundance of many species describes *composition*
- physiology / omics: many traits or genes change together

When we have **many response variables**, two practical problems appear:

1. How do we **visualize** patterns across many variables?
2. How do we **summarize** the dominant patterns without losing the biology?

Ordination is a family of methods that helps with both.

It can also help in cases where we have many explanatory variables (e.g. environmental gradients). This can be especially useful when predictors are collinear.

! Important

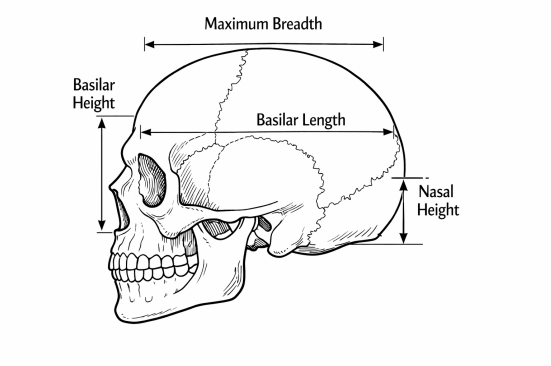
Core idea
Ordination finds a low-dimensional representation (often 2D) of *multivariate* data, so that points that are "similar" in the original multivariate space end up close together in the ordination plot.

Working example: skull shape through time

We will use a classic dataset of **human skull dimensions** measured in millimeters. The biological question is:

> Has skull **shape** changed through time?

Four measurements of skull were made:



We will treat the four skull measurements as multivariate responses, and time as an explanatory variable.

First read the dataset from the csv file:

```
skull <- read_csv("datasets/skull_shape_time.csv")
```

```
Rows: 150 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): id
dbl (5): max.breadth, basi.height, basi.length, nasal.height, thousand.years

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Wide vs long format (and why you should care)

Multivariate data often come in **wide format**:

- one row per object (skull)
- one column per variable (measurement)

That's great for many ordination functions.

But long format is often easier for: - grouped summaries - plotting multiple variables with a single ggplot call

We can easily make a long format version with `pivot_longer()`:

```
skull_long <- skull |>
  pivot_longer(
    cols = c(max.breadth, basi.height, basi.length, nasal.height),
    names_to = "dimension",
    values_to = "value"
  )
```

This will make some of the following exploration easier.

Explore first: univariate views of a multivariate problem

A good habit:

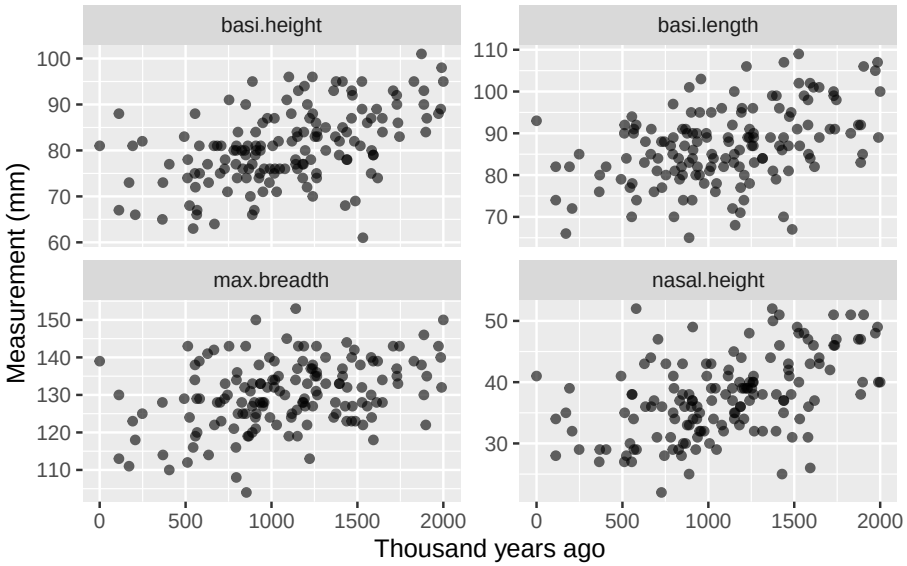
1) visualize each variable,

2) look for patterns and surprises,

3) then use ordination to *summarize*.

Four scatterplots (measurement vs time)

```
p_vars <- skull_long |>
  ggplot(aes(x = thousand.years, y = value)) +
  geom_point(alpha = 0.6) +
  facet_wrap(~ dimension, scales = "free_y", ncol = 2) +
  labs(x = "Thousand years ago", y = "Measurement (mm)")
p_vars
```



Interpretation (qualitative):
If all dimensions change similarly through time, that's mostly a **size** change. If some change differently, that suggests **shape** change.

Why not four separate regressions?

Fitting separate regressions for each measurement has two problems: 1. **Multiple testing**: testing four times increases the chance of false positives. 2. **Ignoring covariance**: measurements may be correlated, so analyzing them separately misses joint patterns

Ordination I: PCA (Principal Components Analysis)

PCA is the most common entry point.

> **!** Important
>
> **What PCA does**
> PCA finds new axes (PC1, PC2, …) that are: - linear combinations of the original variables - orthogonal (uncorrelated) - ordered so that PC1 captures the most variance, PC2 the next most, etc.

Intuitive picture of PCA

Imagine that each skull is a point in a space with one axis per measurement (four dimensions here). Together, the skulls form a cloud of points.

PCA asks from which direction does this cloud show the greatest spread?

To answer this, PCA rotates the coordinate system so that:

- PC1 points in the direction of greatest variation,
- PC2 points in the next greatest direction, orthogonal to PC1,
- and so on.

You can think of PCA as turning the data cloud until you find the view where the points are most spread out. That view becomes PC1. Then PCA finds the best second view at right angles to it (PC2).

The key idea is that PCA does not invent new information: it simply re-expresses the same data using new axes that make dominant patterns easier to see.

> **Note**
>
> Under the hood, PCA is based on the covariance (or correlation) matrix of the variables. The principal components are eigenvectors of this matrix, and the amount of variance they capture is given by the corresponding eigenvalues. You do not need to compute these by hand, but this explains why PCA is fundamentally about variance and correlation.

Centering and scaling

Before PCA, we usually **center** and **scale** the data, or at least think carefully if we should.

- **Centering** subtracts the mean of each variable (so mean = 0).
- **Scaling** divides by the SD of each variable (so SD = 1).

Scaling matters if variables are on different scales or if you care about *relative* variation.

> **Note**
>
> Note that above it is written that PCA does not change the relative positions of the points. However, if we scale the data before performing PCA, this can affect the relative positions of the points in the PCA space. Scaling ensures that all variables contribute equally to the analysis, which can be important when variables are measured on different scales. If we do not scale the data, variables with larger variances can dominate the PCA results, potentially distorting the relative positions of the points in the PCA space.

We can make these transformed variables ourself, or use the built-in options in `prcomp()`.

So, let us do our first PCA. We must be sure to only do the PCA on the four skull measurement variables. To do this we will select only those columns from the original data frame, and pipe these into the `prcomp()` function:

```
skull_pca <- skull |>
  dplyr::select(max.breadth, basi.height, basi.length, nasal.height) |>
  prcomp(center = TRUE, scale. = TRUE)
```

What is this new dataset? Part of it is the new coordinates of each skull in the PC space:

```
head(skull_pca$x)
```

```
           PC1         PC2        PC3        PC4
[1,] -3.621758  0.87923353  0.2413718 -0.2421541
```

```
[2,] -3.201426 -0.38205353  0.3833647 -0.3500371
[3,] -3.004179 -1.00242513  1.5640064 -0.2348435
[4,] -2.683398  0.08615036  1.6069216 -1.0534579
[5,] -3.001132 -0.46314398 -0.2188543 -0.7276811
[6,] -2.774762  0.09654899  0.2109694 -0.8941411
```

Variance represented

Each of the new PC axes captures some of the variance in the original data. We can summarize this with:

```
summary(skull_pca)
```

```
Importance of components:
                          PC1    PC2    PC3     PC4
Standard deviation     1.4916 0.8981 0.7643 0.62007
Proportion of Variance 0.5562 0.2016 0.1460 0.09612
Cumulative Proportion  0.5562 0.7578 0.9039 1.00000
```

This shows the standard deviation of each PC axis, the proportion of variance explained by each PC, and the cumulative proportion of variance explained.

The first PC axis captures 55.6% of the variance in the original data. The first two PC axes together capture 75.8% of the variance.

This is quite a lot, so we can visualize the data well with two PC axes. When we do so, we ignore the remaining axes, which capture less variance (24.2%).

PCA scores plot (PC1 vs PC2)

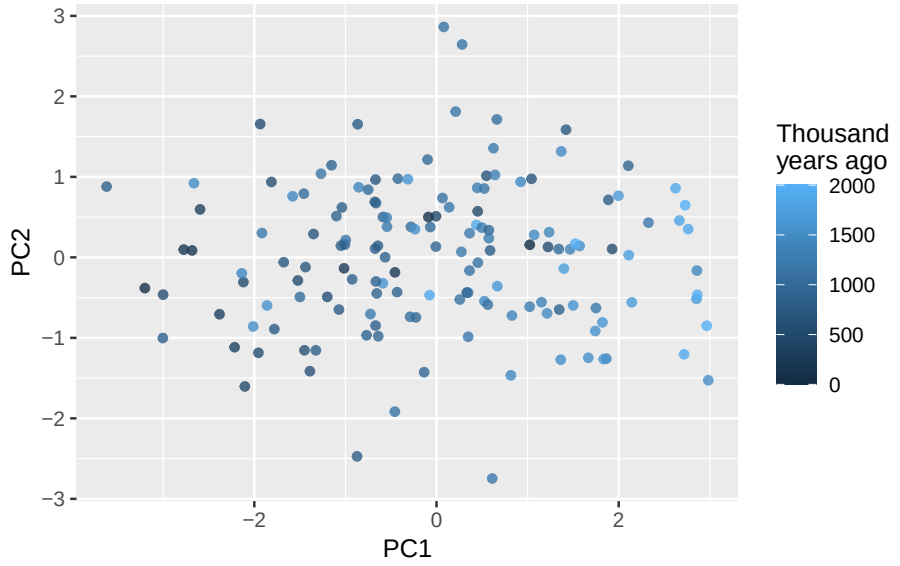
The **scores** are the coordinates of each skull in the new PC space. We do a bit of data wrangling to combine these with the original data (id and time):

```
pca_scores <- as_tibble(skull_pca$x) |>
  mutate(id = skull$id, thousand.years = skull$thousand.years)
pca_scores |> select(id, PC1, PC2, thousand.years) |> slice_head(n = 5)
```

```
# A tibble: 5 x 4
  id      PC1     PC2 thousand.years
  <chr> <dbl>   <dbl>          <dbl>
1 S1    -3.62  0.879             888
2 S2    -3.20 -0.382             112
3 S3    -3.00 -1.00              854
4 S4    -2.68  0.0862            171
5 S5    -3.00 -0.463             544
```

And then make the graph:

```
ggplot(pca_scores, aes(x = PC1, y = PC2, color = thousand.years)) +
  geom_point(alpha = 0.8) +
  labs(color = "Thousand\nyears ago")
```



> **Note**
>
> **Interpretation tip**
> If points separate along PC1 as time increases, that suggests the dominant multivariate trend is associated with time.

What do the axes mean?

The **loadings** tell us how each original variable contributes to each PC. In fact, each PC is a linear combination of the original variables, weighted by the loadings.

```
loadings <- as_tibble(skull_pca$rotation, rownames = "variable")
loadings
```

```
# A tibble: 4 x 5
  variable       PC1     PC2    PC3    PC4
  <chr>        <dbl>   <dbl>  <dbl>  <dbl>
1 max.breadth  0.436   0.689  -0.568  0.107
2 basi.height  0.536   0.108   0.643  0.537
3 basi.length  0.440  -0.712  -0.474  0.272
4 nasal.height 0.573  -0.0783  0.196 -0.792
```

These can help us interpret the PCs. For example, we see a strong positive loading of all four skull measurements on PC1. This suggests that PC1 represents overall size, as all measurements increase together. On the second PC axis, we see a mix of positive and negative loadings, indicating that PC2 captures shape differences where some measurements increase while others decrease. The positive PC2 loadings for head breadth and nasal height, combined with negative loadings for basi height and basi length, suggest that PC2 reflects a shape change where skulls become wider and taller in the nasal region while becoming shorter in the base dimensions.

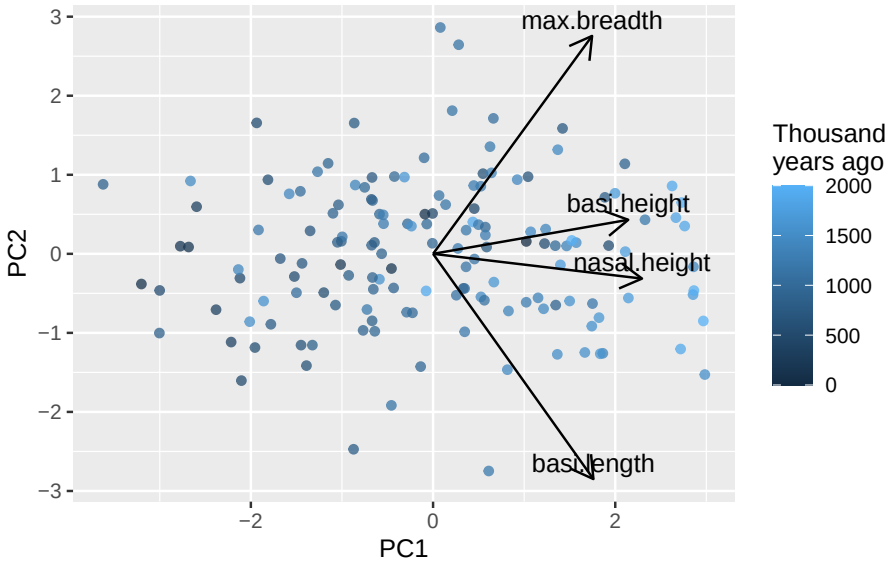
Another way to interpret PCs is to look at the correlations between the original variables and the PCs. This can provide insights into how each original variable relates to the new PC axes. We can calculate these correlations as follows:

```
skull_measurements <- skull |>
  dplyr::select(max.breadth, basi.height, basi.length, nasal.height)
round(cor(skull_measurements, skull_pca$x), 2)
```

```
               PC1   PC2   PC3   PC4
max.breadth   0.65  0.62 -0.43  0.07
basi.height   0.80  0.10  0.49  0.33
basi.length   0.66 -0.64 -0.36  0.17
nasal.height 0.86 -0.07  0.15 -0.49
```

A simple biplot (scores + loadings)

Below is a lightweight biplot-style plot. (There are fancy versions in packages like **factoextra**, but we keep it minimal.)



In this graph we see all arrow point in the same horizontal direction, indicating that PC1 represents overall size. An increase in any of the four measurements will increase PC1.

In contrast, the arrows for PC2 point in different directions, indicating that PC2 represents shape differences where some measurements increase while others decrease. The two longest arrows are for basi.length and max.breadth, suggesting that these measurements contribute most strongly to shape variation captured by PC2.

When ordination helps

We might think that ordination is more useful when we have a greater number of variables, because then the reducing down to two or three dimensions is more helpful. But the key factor is actually the **correlation structure** among the variables. If many variables are correlated, then ordination can capture most of the variance in a few dimensions. If variables are uncorrelated, then ordination may not help much since most of the variance is spread evenly across many dimensions. Let's have a look a this with some simulated data. We will simulate three datasets with three variables each, but with different correlation structures: high correlation, moderate correlation, and low correlation.

Now a 3D plot of the simulated data with different correlation structures (these interactive figures are shown only in the online, HTML, version of the book):

These interactive 3-D figures are shown only in the online version of the book: see this chapter online.

The first plot shows data with high correlation among variables, where the points are clustered along a diagonal line, indicating that the variables are strongly related. The second plot shows moderate correlation, where the points are more spread out but still show some clustering. The third plot shows low correlation, where the points are scattered randomly in space, indicating that the variables are largely independent.

We expect that in the high correlation case, the first principal component will capture a large proportion of the variance, while in the low correlation case, the variance will be more evenly distributed across all principal components. Let's check this by performing PCA on each dataset and examining the variance explained by each principal component.

```
[1] 0.925377783 0.065681354 0.008940863
```

```
[1] 0.6571611 0.1978438 0.1449951
```

```
[1] 0.3874361 0.3586825 0.2538815
```

It turns out that in the high correlation case, the first principal component captures a large proportion of the variance (92.5%). This is a large amount

of variance explained by a single component. It is caused because we made a dataset where the variables are strongly correlated. There is only one axis along which the data vary strongly.

In contrast, in the zero correlation case, the variance is more evenly distributed across all principal components, with each component capturing around 38.7%, 35.9%, and 25.4% of the variance respectively. This indicates that there is no single dominant direction of variance in the data, as the variables are largely independent.

In the intermediate correlation case, the variance explained by each principal component is more balanced, with the first component capturing around 65.7% of the variance, and the remaining components capturing significant portions as well. This reflects the moderate correlation structure in the data.

Ordination is most useful when it can reduce the dimensionality of the data while retaining most of the variance. This means that ordination methods like PCA will be more effective in summarizing the data in the high correlation case, as most of the information can be captured in just one or two dimensions. In contrast, in the low correlation case, ordination may not provide much dimensionality reduction, as each variable contributes independently to the overall variance.

What determines if we have variables with high correlation in real datasets? Often, it is biological or physical relationships among the variables. For example, in morphological datasets, measurements of different body parts may be correlated due to overall size or shape factors. In ecological datasets, species abundances may be correlated due to shared environmental preferences or interactions. Understanding the underlying biology can help us anticipate when ordination methods will be most useful.

Ordination II: NMDS (Non-metric Multidimensional Scaling)

PCA is powerful, but it is a **linear** method and it relies on Euclidean geometry in the original variable space.

NMDS is often used when:

- you want ordination based on **distances/dissimilarities**
- the relationships are not well represented by a linear method
- you want flexibility in the choice of distance (e.g. Bray–Curtis, Gower, …)

> **!** Important
>
> **What NMDS does (conceptually)**
> 1) compute pairwise distances among objects
> 2) place points in a low-dimensional space (usually 2D)

3) try to preserve the **rank order** of distances (non-metric)
4) report **stress**: lower is better (roughly: mismatch between original dissimilarities and ones in the lower-dimensional space)

Step 1: Choose a distance measure

Here our variables are numeric, so Euclidean distance is a reasonable default.

We will scale first (so variables contribute comparably), then compute distances:

```
skull_measurements <- skull |>
  dplyr::select(max.breadth, basi.height, basi.length, nasal.height)
X_scaled <- scale(skull_measurements)
skull_distances <- dist(X_scaled, method = "euclidean")
#skull_distances
```

Distances for community (abundance) data: Bray–Curtis

Euclidean distance on scaled variables suits continuous measurements like these skull dimensions. For **species abundance or community composition data** (counts or relative abundances of many species or taxa at each site or sample), the most widely used dissimilarity is **Bray–Curtis**. For two samples j and k, with abundances y_{ij} and y_{ik} of species i:

$$BC_{jk} = \frac{\sum_i |y_{ij} - y_{ik}|}{\sum_i (y_{ij} + y_{ik})}$$

It is 0 when two samples have identical composition and 1 when they share no species. Two features make it better suited to community data than Euclidean distance:

- **It ignores shared absences.** Two sites both lacking a species are not counted as similar because of it (with many species, most are absent from most sites, and Euclidean distance would make sites with few species look similar just because they share many zeros).
- **It works on the original abundances** (or proportions), without scaling each species. Very abundant species have more influence than rare ones, so abundances are sometimes square-root or log transformed first.

A small example with three sites and four species:

```
communities <- data.frame(species_A = c(10, 8, 0),
                          species_B = c(5, 6, 0),
                          species_C = c(0, 0, 4),
                          species_D = c(0, 0, 3))
vegdist(communities, method = "bray")
```

```
          1         2
2 0.1034483
3 1.0000000 1.0000000
```

```
dist(communities, method = "euclidean")
```

```
          1         2
2  2.236068
3 12.247449 11.180340
```

Sites 1 and 2 share the same species in similar numbers (Bray–Curtis about 0.10), whereas site 3 shares no species with either (Bray–Curtis exactly 1). In R, Bray–Curtis dissimilarities are calculated with `vegdist(..., method = "bray")` from the vegan package, and can be passed to `metaMDS()` just like the Euclidean distances here.

Step 2: Fit NMDS with multiple random starts

In NMDS, we need to choose some random starting configuration of points. And we want to make sure that we choose a good solution, not just a local optimum. Hence, we try multiple random starts and pick the best one.

In practice, use `metaMDS()` (from **vegan**) rather than calling the low-level optimizer directly. It tries multiple starting configurations and does useful housekeeping.

When calling `metaMDS()`, we specify the following:

- `k = 2` for a 2D solution.
- `trymax = 50` to try up to 50 random starts.
- `autotransform = FALSE` because we already scaled the data ourselves.
- `trace = FALSE` to suppress output during fitting.

```
set.seed(1)
skull_nmds <- metaMDS(skull_distances, k = 2, trymax = 50, autotransform = FALSE, trace = FALSE)
skull_nmds
```

```
Call:
metaMDS(comm = skull_distances, k = 2, trymax = 50, autotransform = FALSE,      trace = FALSE)

global Multidimensional Scaling using monoMDS

Data:     skull_distances
Distance: euclidean

Dimensions: 2
Stress:     0.1467224
Stress type 1, weak ties
```

```
Best solution was repeated 1 time in 20 tries
The best solution was from try 5 (random start)
Scaling: centring, PC rotation
Species: scores missing
```

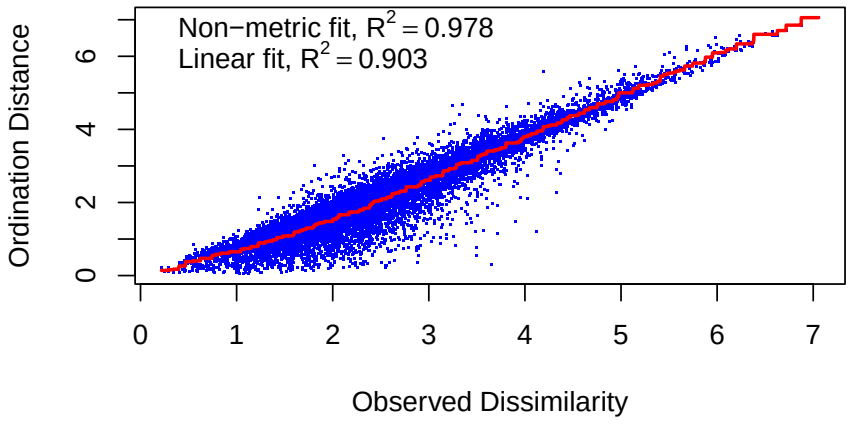
Lots of information here, for example we have:

- the call used
- the type of distance
- the number of dimensions
- the number of random starts tried
- the final stress value
- the scaling of the data

Step 3: Assess NMDS fit with stress

The stress of an NMDS solution quantifies how well the low-dimensional configuration preserves the rank order of the original dissimilarities. That is, it is a measure of mismatch between the distances in the original high-dimensional space and the distances in the reduced low-dimensional space. Here is the stress plot for our NMDS solution:

```
stressplot(skull_nmds)
```



We see a fairly good match between the original distances and the NMDS distances. The correlations are reasonably high, indicating that the NMDS solution captures the rank order of dissimilarities well.

The stress value is another measure of fit. Lower stress values indicate a better

fit. A common rule of thumb is that stress < 0.1 is a good fit, stress between 0.1 and 0.2 is acceptable, and stress > 0.2 indicates a poor fit. A poor fit suggests that the data may not be well represented in two dimensions, and a higher-dimensional solution may be needed.

```
skull_nmds$stress
```

```
[1] 0.1467224
```

We are in the range of okay fit in two dimensions.

What improvement in stress do we get if we go to three dimensions?

```
set.seed(1)
skull_nmds_3d <- metaMDS(skull_distances, k = 3, trymax = 50, autotransform = FALSE, trace = FALS
skull_nmds_3d$stress
```

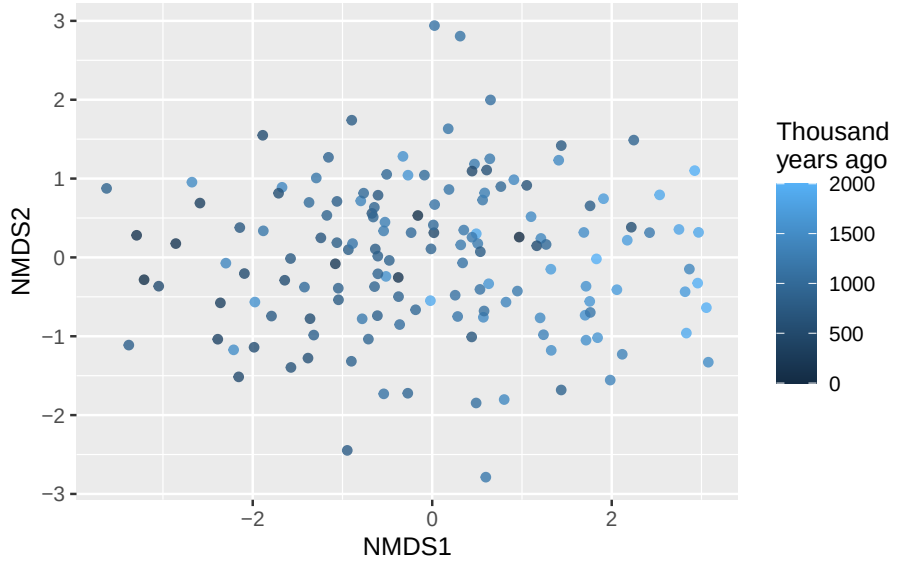
```
[1] 0.07009879
```

The stress decreases when we move to three dimensions, indicating a better fit. However, the improvement may not be substantial enough to justify the added complexity of a three-dimensional solution. In practice, we often prefer two-dimensional solutions for ease of visualization and interpretation, unless the stress reduction is very large.

We can plot the NMDS scores, colored by time:

```
nmds_scores <- as_tibble(scores(skull_nmds, display = "sites")) |>
  mutate(id = skull$id, thousand.years = skull$thousand.years)

ggplot(nmds_scores, aes(x = NMDS1, y = NMDS2, color = thousand.years)) +
  geom_point(alpha = 0.8)  +
  labs(color = "Thousand\nyears ago")
```



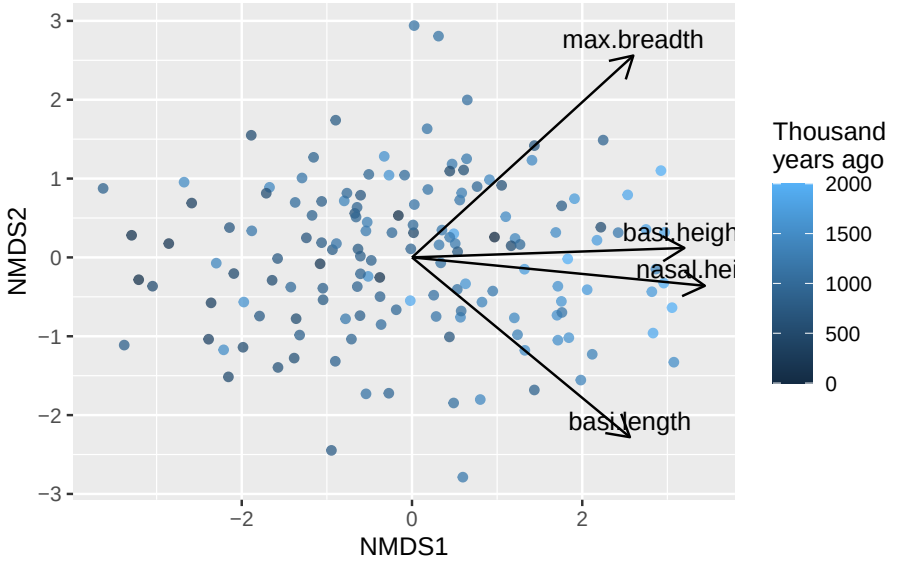
> **Note**
>
> **Axis directions are arbitrary**
> Flipping or rotating an NMDS solution (or a PCA) does not change its meaning. Only relative distances among points matter.

If we wanted to know the biological interpretation of the NMDS axes, we would need to look at correlations between the original variables and the NMDS axes. This is just the same as we did for PCA. We could then plot the correlations as arrows on the NMDS plot to help interpret the axes.

```
cor_nmds <- round(cor(skull_measurements, nmds_scores |> select(NMDS1, NMDS2)), 2)
cor_nmds
```

```
              NMDS1 NMDS2
max.breadth    0.65  0.64
basi.height    0.80  0.03
basi.length    0.64 -0.57
nasal.height   0.86 -0.09
```

And plot these as arrows:



> **Note**
>
> This NMDS graph looks a lot like the PCA biplot. This is because our data are fairly linear and Euclidean distances are appropriate. In more complex datasets, NMDS can reveal patterns that PCA might miss.

Hypothesis testing: avoid "four separate regressions"

If you test time against each skull measurement separately, you face a **multiple testing** problem.

A better match to the question ("do skulls change through time in multivariate space?") is a **multivariate test**.

> **Important**
>
> **What you need to know about these methods.** You do not need to be able to run these tests yourself. You should know that they exist, understand what question each one answers, and be able to choose which one you would use in a given situation:

Question	Approach
Do groups differ in their multivariate mean, assuming (approximately) multivariate normal data?	MANOVA
Do groups differ in their position in multivariate space, using any distance measure and without assuming normality?	PERMANOVA
Do groups differ in how spread out (variable) they are in multivariate space?	PERMDISP (`betadisper`)
Is a continuous explanatory variable related to the positions of observations in an ordination, along a straight direction?	envfit
… or along a curved (non-linear) surface?	ordisurf

Because PERMANOVA can give a significant result when groups differ in spread rather than position, a PERMDISP check is often done alongside it.

Here are four common approaches demonstrated but not deeply explained:

1. MANOVA (parametric, multivariate normality assumptions)
2. PERMANOVA (distance-based, permutation test)
3. Dispersion checks (are groups equally variable?)
4. Fitting time onto an ordination (envfit, ordisurf)

1) MANOVA (parametric)

Multivariate ANOVA (MANOVA) tests whether group **centroids** differ in multivariate space, assuming multivariate normality. The response is a matrix of multiple variables, and the explanatory variables can be categorical or continuous.

```
skull_manova <- manova(cbind(max.breadth, basi.height, basi.length, nasal.height) ~ tho
                  data = skull)
summary(skull_manova, test = "Pillai")
```

```
               Df  Pillai approx F num Df den Df    Pr(>F)
thousand.years   1 0.36751   21.064      4    145 1.042e-13 ***
Residuals      148
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we bin years into categories, MANOVA becomes closer to multivariate ANOVA:

```
skull <- skull |>
  mutate(year_class = cut(thousand.years, breaks = 8) |> fct_inorder())
skull_manova_class <- manova(cbind(max.breadth, basi.height, basi.length, nasal.height) ~ year_cl
                       data = skull)
summary(skull_manova_class, test = "Pillai")
```

```
            Df  Pillai approx F num Df den Df    Pr(>F)
year_class   7 0.52715   3.0792     28    568 3.261e-07 ***
Residuals  142
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

> **Caution**
>
> **MANOVA assumptions matter**
> MANOVA relies on multivariate normality and homogeneity of covariance matrices. In many biological datasets, these are imperfect—so distance-based alternatives are common.

2) PERMANOVA (distance-based)

PERMANOVA tests whether group **centroids** differ in multivariate space, using permutations. In vegan, use `adonis2()`.

```
# Use the same distance matrix skull_distances we used for NMDS
adonis2(skull_distances ~ year_class, data = skull, permutations = 999)
```

```
Permutation test for adonis under reduced model
Permutation: free
Number of permutations: 999

adonis2(formula = skull_distances ~ year_class, data = skull, permutations = 999)
          Df SumOfSqs      R2      F Pr(>F)
Model      7   138.47 0.23234 6.1396  0.001 ***
Residual 142   457.53 0.76766
Total    149   596.00 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3) PERMDISP / betadisper: are groups equally variable?

A common pitfall:

- PERMANOVA can be significant either because **centroids differ**
- or because **dispersion differs** (groups have different spread)

So it's good practice to check dispersion when comparing groups.

```
skull_betadisper <- betadisper(skull_distances, skull$year_class)
#skull_betadisper
anova(skull_betadisper)
```

```
Analysis of Variance Table

Response: Distances
           Df Sum Sq Mean Sq F value Pr(>F)
Groups      7  0.929 0.13274  0.2284  0.978
Residuals 142 82.529 0.58119
```

```
permutest(skull_betadisper, permutations = 999)
```

```
Permutation test for homogeneity of multivariate dispersions
Permutation: free
Number of permutations: 999

Response: Distances
           Df Sum Sq Mean Sq      F N.Perm Pr(>F)
Groups      7  0.929 0.13274 0.2284    999   0.98
Residuals 142 82.529 0.58119
```

4) Fitting time onto an ordination: envfit and ordisurf

Even when you use an unconstrained ordination (PCA/NMDS), you may want to show how an explanatory variable aligns with it.

envfit: linear fit + permutation test

```
skull_envfit_years <- envfit(skull_nmds ~ thousand.years, data = skull, permutations =
skull_envfit_years
```

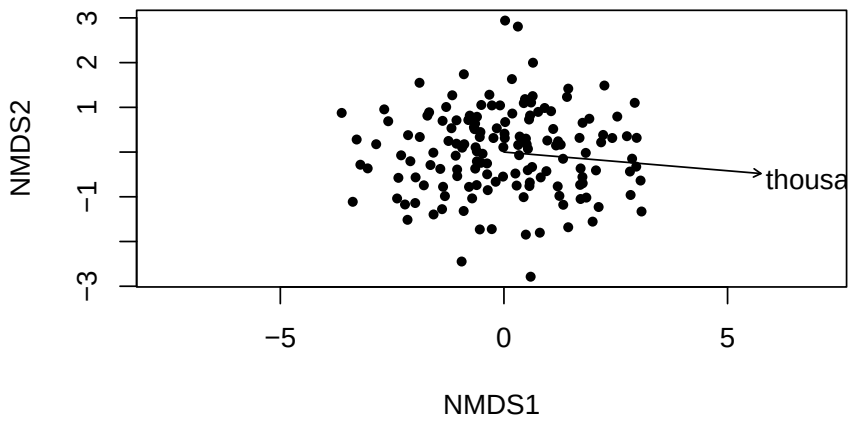
```
***VECTORS

                NMDS1     NMDS2     r2 Pr(>r)
thousand.years  0.996530 -0.083272 0.3643  0.001 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Permutation: free
Number of permutations: 999
```

```
plot(skull_nmds, type = "n")
```

```
species scores not available
```

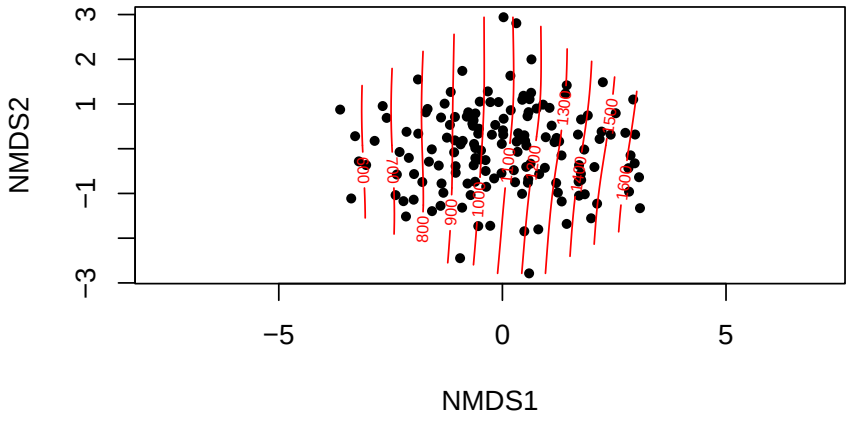
```
points(skull_nmds, display = "sites", pch = 16, cex = 0.8)
plot(skull_envfit_years, col = "black")  # vector direction and significance
```




```
plot(skull_nmds, type = "n")
```

```
species scores not available
```

```
points(skull_nmds, display = "sites", pch = 16, cex = 0.8)
ordisurf(skull_nmds ~ thousand.years, data = skull, add = TRUE)
```




```
Family: gaussian
Link function: identity

Formula:
y ~ s(x1, x2, k = 10, bs = "tp", fx = FALSE)

Estimated degrees of freedom:
2.98  total = 3.98

REML score: 1092.963
```

PCA in multiple regression

PCA can also be used to reduce multiple explanatory variable into a few uncorrelated components, which can then be used in multiple regression models. This is particularly useful when the predictors are highly correlated, as it helps to avoid multicollinearity issues.

The steps for this would be:

1. Perform PCA on the explanatory variables to obtain principal components.
2. Make a linear regression model using the selected principal components as explanatory variables.
3. Interpret the results in terms of the original variables, if needed.

The "win" here is that the principal components are uncorrelated, which simplifies the regression analysis and interpretation. The "lose" is that the principal

components may not have a straightforward biological interpretation, so care is needed when explaining the results.

Review

Ordination is useful when we have multivariate data and when variables within that data are correlated. The correlation structure means that we can represent the data in a lower-dimensional space without losing too much information. If there is a lot of correlation among variables, then ordination becomes very useful. If there is little correlation, then ordination may not help much since most of the variance is spread evenly across many dimensions.

Summary:

- PCA is a linear ordination method that finds orthogonal axes capturing the most variance.
- NMDS is a flexible, distance-based ordination method that preserves rank order of dissimilarities.
- Both methods help visualize and summarize multivariate data.
- Multivariate hypothesis tests (MANOVA, PERMANOVA) assess group differences in multivariate space.
- Checking dispersion (PERMDISP) is important to interpret PERMANOVA results.
- Fitting explanatory variables onto ordinations (envfit, ordisurf) helps interpret patterns.

Further reading

Evelyn Chrystalla "E.C." Pielou was a Canadian statistical ecologist. She began her career as a researcher for the Canadian Department of Forestry and the Canadian Department of Agriculture. Pielou's books are classics that helped establish quantitative ecology. They range from the mathematics of ordination to kinds of (ecological) question ordination helps answer. A great place to start is *The Interpretation of Ecological Data: A Primer on Classification and Ordination*

Mixed models (L11-1)

Learning objectives

After this unit, students will be able to:

1. Recognise grouped, nested or hierarchical data in biological study designs (e.g. repeated measures, individuals within populations, sites within regions).
2. Explain why ignoring grouping in the data leads to pseudoreplication and overconfident conclusions.
3. Explain why averaging the repeated measurements within each group before using `lm()` is sometimes acceptable but often throws away information.
4. Recognise pseudoreplication when a model's residual degrees of freedom are much larger than the number of independent units (e.g. individual plants) in the study.
5. Distinguish fixed effects from random effects, and decide which is appropriate for a variable in a described study.
6. Distinguish random intercept models from random slope models, and nested from crossed random effects, and write the corresponding `lmer()` formula.
7. Explain partial pooling, and how it shrinks group-level estimates towards the overall mean.
8. Use R to fit linear mixed models with random intercepts, and with random slopes, using `lmer()` from the `lme4` package.
9. Use R output to obtain the fixed-effect estimates and the variance components.
10. Interpret mixed model output in biological terms, including what the variance components say about variation among groups.
11. Explain why degrees of freedom, and so p-values, for fixed effects in mixed models are not straightforward, and that several approximations exist (e.g. Satterthwaite's method).
12. Identify limitations and assumptions of mixed models in a described analysis.

13. Recognise a correct and complete statement of results from a mixed model.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch11_mixed_models.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 11 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch11_mixed_models.R")
```

Video walkthrough: coming soon.

Introduction

So far in BIO144 we mostly used models where **each observation is assumed independent**:

- linear models (`lm()`)
- generalized linear models (`glm()`), e.g. Poisson and binomial.

By independent we mean that the value of one observation does not give us any information about the value of another observation.

But many biological datasets violate independence because observations come in **groups**:

- repeated measures on the same individual (before/after, time series, multiple tissues)
- multiple individuals from the same plot / site / stream / lake / cage / family
- students within classes, patients within hospitals, samples within batches

In these cases, treating all rows as independent often leads to **false confidence** (too-small standard errors, too-small p-values). Mixed models are one standard way to handle this.

> **!** Important
>
> **Core idea** A *mixed model* extends over all our previous models by adding **random effects** that represent grouping structure (clusters) in the data.
> - **fixed effects**: effects you want to estimate explicitly (treatments, temperature, time, …)
> - **random effects**: variation among groups (individuals, sites, years, …) that induces correlation within groups

Why not just average?

A common workaround is to average repeated measurements to a single value per group and then use `lm()`.

Sometimes this is OK — but often it throws away information.

Reasons *not* to average include:

1. **Imbalanced sampling** (groups have different numbers of observations): averaging changes the weighting.
2. **Which average?** (mean, median, mode): different choices answer different questions.
3. **False confidence**: pretending "n rows" are independent can make uncertainty look too small.
4. **You may want to study variation among groups** (some individuals respond more strongly than others).
5. **"Sharing information" across groups**: mixed models partially pool group estimates toward the overall mean.
6. **Keeping information**: you can use all observations without collapsing the design.

> **i** Note
>
> A mixed model gives you a principled compromise between:
> - analysing each observation as a independent one, and
> - averaging everything (too coarse).
>
> This is sometimes called **partial pooling**.

The problem: non-independence and pseudoreplication

Imagine we measure the same individual multiple times.

If we fit a simple linear model, the residuals are assumed independent:

$$\varepsilon_i \sim \text{Normal}(0, \sigma^2), \quad \text{independent across } i$$

But repeated measures induce correlation:

- measurements from the same individual tend to be more similar
- so residuals are not independent

This is one common form of **pseudoreplication**: treating repeated measurements as if they were separate independent replicates.

Random intercept models

Random intercept idea

Suppose we measure a response y (e.g. growth) for individuals j at observations i. And that we measure each individual more than once. And that we have some treatment, such as temperature x.

This means that our data have a **grouping structure**: observations are grouped by individual. If we want to calculate the mean growth, we should account for this grouping.

Let's read in the illustrative data:

```
example_repeated <- read_csv("datasets/mixed_model_example.csv")
```

```
Rows: 50 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): individual
dbl (2): temperature, growth

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows:

```
head(example_repeated)
```

```
# A tibble: 6 x 3
  individual temperature growth
  <chr>            <dbl>  <dbl>
1 Ind01             9.15  16.1
2 Ind01             9.37  17.4
3 Ind01             2.86   7.58
4 Ind01             8.30  14.2
5 Ind01             6.42  13.8
6 Ind02             5.19  12.5
```

In this data we have two sources of variation in `growth`:

1. **fixed effect** of `temperature` (same for all individuals)
2. **random effect** of `individual` (different baseline growth for each individual)

A fixed effect is something we want to estimate explicitly (e.g. how growth changes with temperature). Fixed effects are variables in which the values have specific meaning (e.g. temperature = 5°C).

A random effect is something that varies among groups (individuals here), but where we are not interested in the specific values for each group. Instead, we want to estimate the amount of variation among groups. Random effects are variables where the specific values are not of interest, but rather the variation among them (e.g. individual identity). I.e., the value "Ind01" has no specific meaning; we just want to know how much individuals differ from each other on average. That value could be anything.

A random effect is often containing levels that are considered a random sample from a larger population. We did not, for example, choose specific individuals for a reason; they are just a sample of all possible individuals.

An appropriate model for this data is a **random-intercept mixed model**. It is a model in which each group (individual) has its own intercept (baseline), but the intercepts are assumed to come from a common distribution.

A random-intercept mixed model is:

$$y_{ij} = \beta_0 + \beta_1 x_{ij} + b_{0j} + \varepsilon_{ij}$$

where:

- β_0, β_1 are **fixed effects**
- x_{ij} is the explanatory variable for observation i in group j. Also called a **fixed effect**.
- b_{0j} is a **random intercept** for group j, typically $b_{0j} \sim \text{Normal}(0, \sigma_b^2)$
- ε_{ij} is residual error, $\varepsilon_{ij} \sim \text{Normal}(0, \sigma^2)$

Interpretation:

- each group gets its own intercept (baseline), but
- those intercepts are assumed to come from a common distribution

Random intercept syntax in R

To make a mixed model in R we can no longer use `lm()` or `glm()`. Instead, we use the `lmer()` function from the `lme4` package.

In the `lmer` function we have to specify the random effects in a special way. For example, a random-intercept model for `y` with fixed effect `x` and random intercept by `group` is specified as:

```
y ~ x + (1 | group)
```

Read it as: "a model for `y` with fixed effect `x` and a random intercept by `group`".

Random slope models

Sometimes groups differ not only in baseline level, but also in how they respond to an explanatory variable.

A random-slope model:

$$y_{ij} = \beta_0 + \beta_1 x_{ij} + b_{0j} + b_{1j} x_{ij} + \varepsilon_{ij}$$

In R:

```
y ~ x + (1 + x | group)
```

Here each group has its own intercept **and** its own slope, and these can be correlated.

> **Caution**
>
> Random slopes are powerful but can be hard to estimate with small datasets. If the model struggles to fit (singular fit warnings), consider simplifying.

Nested and crossed random effects

Nested

Nested means one grouping factor is contained within another.

Example: measurements within plants within plots:

```
y ~ treatment + (1 | plot/plant)
```

This expands to `(1 | plot) + (1 | plot:plant)`.

Crossed

Crossed means groups are not nested.

Example: repeated measures with multiple observers (each observer measures many individuals; each individual is measured by many observers):

```
y ~ x + (1 | individual) + (1 | observer)
```

Hands-on example: plant growth with repeated measures

Biological story

You are studying how temperature affects plant growth.

- 30 plants are grown at one of three temperatures: 10°C, 15°C, 20°C.
- Each plant is measured weekly for 8 weeks.
- Response: plant height (cm).

Because we repeatedly measure the **same plant**, the observations are not independent. We will compare:

1. a naive linear model (wrong independence assumption),
2. a mixed model with plant as a random effect.

An example dataset

Read in the data:

```
plant_growth <- read_csv("datasets/plant_growth_repeated.csv") |>
  mutate(temp = factor(temp, levels = c("10", "15", "20")))
```

```
Rows: 240 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): plant_id
dbl (3): temp, week, height

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

`read_csv()` reads `temp` as a number (10, 15, 20). Because we treat temperature as a treatment with three levels, we convert it to a factor with `factor()`.

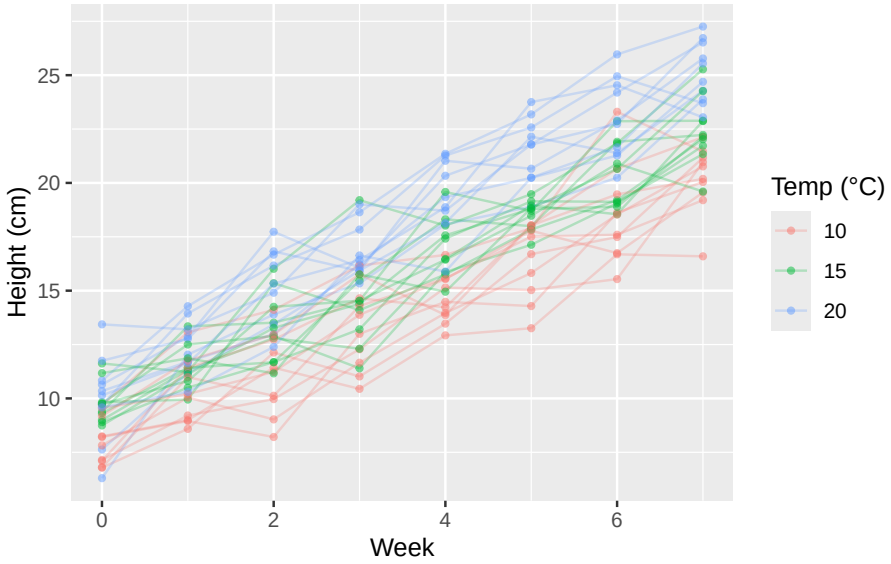
```
head(plant_growth)
```

```
# A tibble: 6 x 4
  plant_id temp   week height
  <chr>    <fct> <dbl>  <dbl>
1 P01      10        0   6.79
2 P01      10        1   8.59
3 P01      10        2  12.1
4 P01      10        3  11.0
5 P01      10        4  13.5
```

```
6 P01      10        5  16.7
```

Explore the data

First we make a plot of the data:



We can see data from each individual plant because it is connected by a line. We also see that plants at higher temperature tend to be taller. And we see that plants differ in their baseline height (week 0).

There are two challenges here: 1. We have repeated measures on the same plants (non-independence). 2. We have variation among plants in baseline height.

A mixed model can handle both of these.

Wrong model: treat all rows as independent

```
m_lm <- lm(height ~ week * temp, data = plant_growth)
anova(m_lm)
```

```
Analysis of Variance Table

Response: height
           Df Sum Sq Mean Sq   F value    Pr(>F)
week        1 4478.8  4478.8 1888.8551 < 2.2e-16 ***
temp        2  586.9   293.5  123.7678 < 2.2e-16 ***
week:temp   2   46.9    23.4    9.8838 7.572e-05 ***
Residuals 234  554.8     2.4
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This model pretends there are $30 \times 8 = 240$ independent data points. But most of that information is repeated measures on the same plants.

We can see that this is pseudoreplication because there are many more residual degrees of freedom (234) than plants (30). This is a classic sign of pseudoreplication. We really cannot trust the p-values or confidence intervals from this model.

Mixed model: random intercept for plant

Now let's fit a mixed model with plant identity as a random effect:

```
m_lmm1 <- lmer(height ~ week * temp + (1 | plant_id), data = plant_growth)
anova(m_lmm1)
```

```
Analysis of Variance Table
          npar Sum Sq Mean Sq  F value
week         1 4478.8  4478.8 3264.193
temp         2   80.3    40.1   29.258
week:temp    2   46.9    23.4   17.081
```

In the anova table, note that there are no degrees of freedom and no p-values for fixed effects. This is because calculating these in mixed models is complicated and there are multiple methods. We will look at this more later in this chapter.

Interpretation:

- fixed effects describe the **average** relationship between height, week, and temperature
- the random intercept captures baseline differences among plants

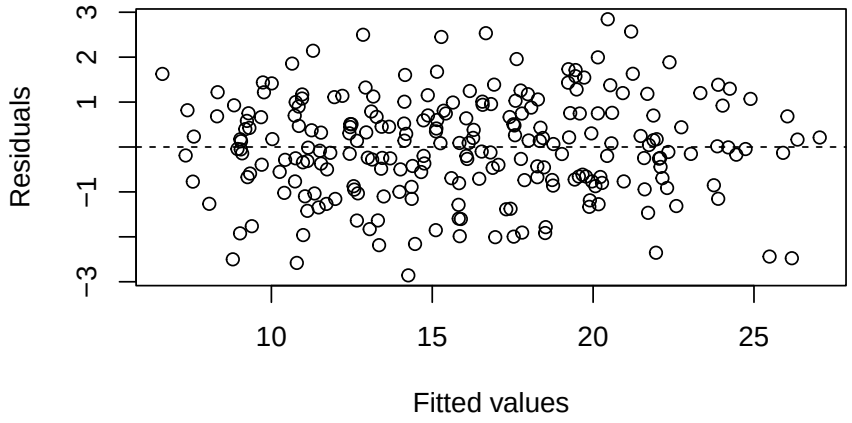
model checking

Mixed-model diagnostics (i.e., model checking) can be more involved than `lm()`, but you can still start with:

- residual vs fitted plot (nonlinearity / heteroscedasticity)
- normal Q–Q of residuals (approximate)
- check random effect estimates for extreme outliers

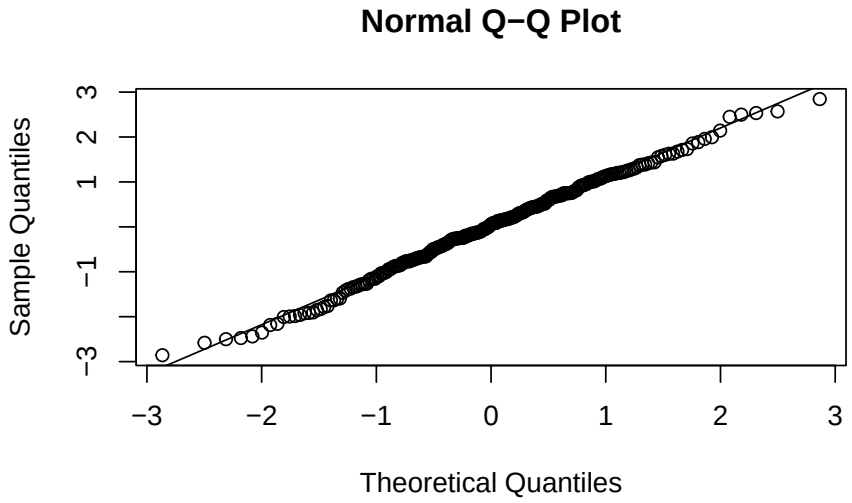
1) Residuals vs fitted

```
plot(fitted(m_lmm1), resid(m_lmm1),
     xlab = "Fitted values", ylab = "Residuals")
abline(h = 0, lty = 2)
```



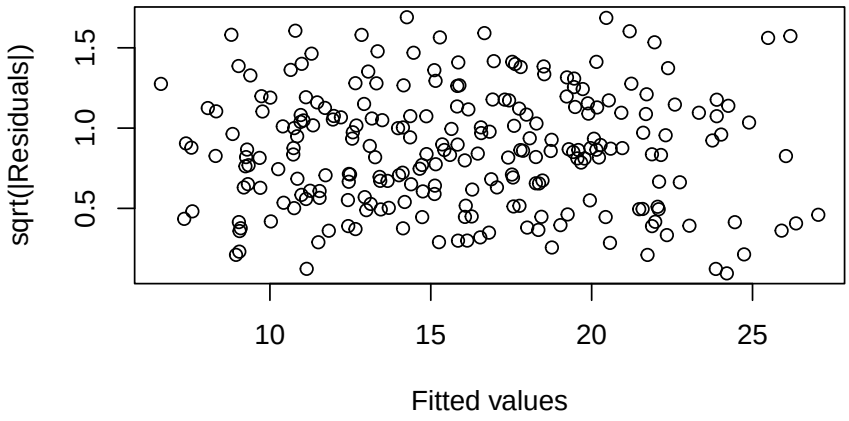
2) Normal Q–Q of residuals

```
qqnorm(resid(m_lmm1)); qqline(resid(m_lmm1))
```



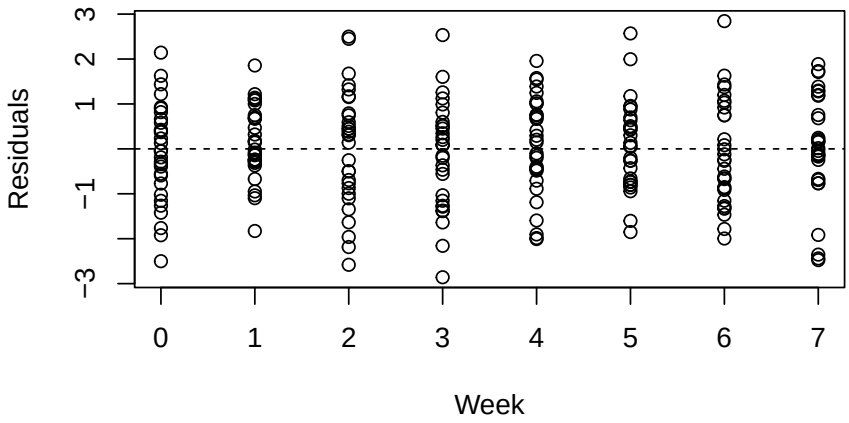
3) Scale–location (sqrt(|resid|) vs fitted)

```
plot(fitted(m_lmm1), sqrt(abs(resid(m_lmm1))),
     xlab = "Fitted values", ylab = "sqrt(|Residuals|)")
```



4) Residuals vs explanatory variable (helpful for nonlinearity with a continuous explanatory variable, here week)

```
plot(plant_growth$week, resid(m_lmm1), xlab = "Week", ylab = "Residuals")
abline(h = 0, lty = 2)
```



> **Caution**
>
> **Don't over-interpret p-values**
> In mixed models, inference depends on how you handle degrees of freedom and uncertainty. In BIO144, focus on: - correct model structure (what must be random?) - effect sizes and uncertainty (CIs) - sensible plots and biological interpretation

Extension: random slopes for week

If you believe plants differ in growth rate (not only baseline), add a random slope:

```
m_lmm2 <- lmer(height ~ week * temp + (1 + week | plant_id), data = plant_growth)
```

```
boundary (singular) fit: see help('isSingular')
```

```
summary(m_lmm2)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: height ~ week * temp + (1 + week | plant_id)
   Data: plant_growth

REML criterion at convergence: 822.6

Scaled residuals:
     Min       1Q   Median       3Q      Max
-2.43220 -0.64862  0.04439  0.63469  2.31852

Random effects:
 Groups   Name        Variance  Std.Dev. Corr
 plant_id (Intercept) 0.9012459 0.94934
          week        0.0006804 0.02608  1.00
 Residual             1.3682646 1.16973
Number of obs: 240, groups:  plant_id, 30

Fixed effects:
            Estimate Std. Error t value
(Intercept)  8.12308    0.38358  21.177
week         1.69948    0.05767  29.469
temp15       1.45872    0.54247   2.689
temp20       2.24843    0.54247   4.145
week:temp15  0.10592    0.08156   1.299
week:temp20  0.45169    0.08156   5.538

Correlation of Fixed Effects:
```

```
            (Intr) week   temp15 temp20 wk:t15
week        -0.404
temp15      -0.707  0.285
temp20      -0.707  0.285  0.500
week:temp15  0.285 -0.707 -0.404 -0.202
week:temp20  0.285 -0.707 -0.202 -0.404  0.500
optimizer (nloptwrap) convergence code: 0 (OK)
boundary (singular) fit: see help('isSingular')
```

Note the message "boundary (singular) fit". In the "Random effects" part of the summary, the standard deviation among plants in their week slope is almost zero (0.03), and the correlation between random intercepts and slopes is 1. This means the data contain almost no evidence that plants differ in growth rate: the random-slope model is more complex than the data can support (see the caution box about random slopes above). A singular fit is a sign that we should simplify the model, here by going back to the random-intercept model.

Compare the two mixed models:

```
m_lmm_comparison <- anova(m_lmm1, m_lmm2)
```

```
refitting model(s) with ML (instead of REML)
```

```
m_lmm_comparison
```

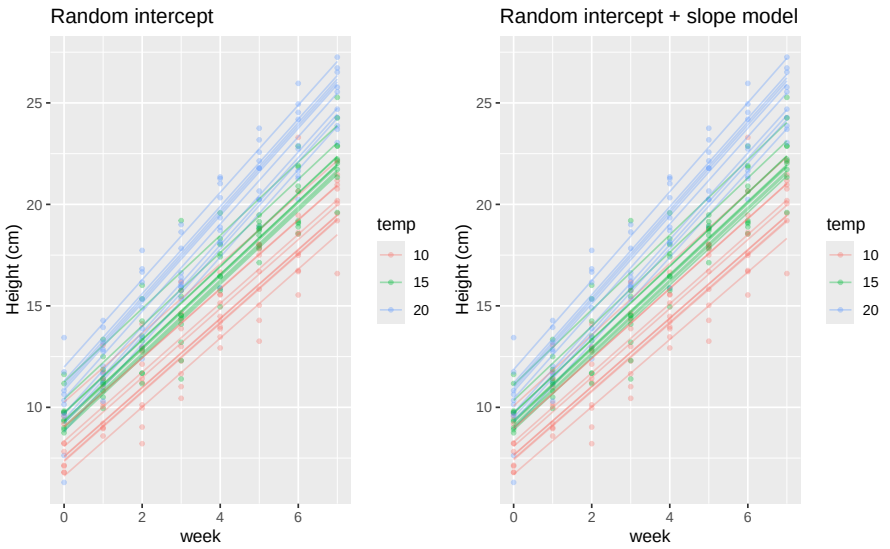
```
Data: plant_growth
Models:
m_lmm1: height ~ week * temp + (1 | plant_id)
m_lmm2: height ~ week * temp + (1 + week | plant_id)
       npar    AIC    BIC  logLik -2*log(L)  Chisq Df Pr(>Chisq)
m_lmm1    8 826.51 854.35 -405.25    810.51
m_lmm2   10 830.01 864.82 -405.00    810.01 0.4974  2     0.7798
```

There is little evidence that adding random slopes improves the model here ($p = 0.78$). But in other datasets it might.

> **Note**
>
> This likelihood ratio test compares nested models. It is widely used, but has subtleties for random effects. In BIO144 you can treat it as a reasonable practical tool, while noting that "testing random effects" is an advanced topic.

Visualising fitted values

We can visualise the fitted values from both mixed models:



There is not much difference between the two models here, but in other datasets random slopes can make a big difference.

Significance testing for fixed effects

Calculating p-values for fixed effects in mixed models is complicated, and there are multiple methods (Satterthwaite, Kenward-Roger, likelihood ratio tests, bootstrapping, Bayesian credible intervals). It is difficult because the degrees of freedom depend on the random effects structure and the data. There is no clear and objective method to get the degrees of freedom.

Nevertheless, we can get p-values for terms using the `lmerTest` package (optional). This changes the `lmer()` function to provide p-values using Satterthwaite's method for degrees of freedom.

```
library(lmerTest)
```

```
Attaching package: 'lmerTest'

The following object is masked from 'package:lme4':

    lmer

The following object is masked from 'package:stats':

    step
```

```
m_lmm1 <- lmer(height ~ week * temp + (1 | plant_id), data = plant_growth)
anova(m_lmm1)
```

```
Type III Analysis of Variance Table with Satterthwaite's method
          Sum Sq Mean Sq NumDF  DenDF   F value    Pr(>F)
week      4478.8  4478.8     1 207.00 3264.1929 < 2.2e-16 ***
temp        21.6    10.8     2  46.37    7.8666  0.001144 **
week:temp   46.9    23.4     2 207.00   17.0806 1.362e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You may notice that the Sum Sq (and F value) for `temp` differ from the `anova()` table we got earlier with `lme4`. This is not because the model changed (it is the same model), but because the two `anova()` functions calculate different types of sums of squares. The `lme4` version gives *sequential* (Type I) sums of squares, like `anova()` for `lm()`: each term is tested after the terms above it. The `lmerTest` version gives Type III sums of squares by default: each term is tested after *all* other terms, including the interaction. The last term (`week:temp`) is the same in both. If you want the sequential version with p-values, use `anova(m_lmm1, type = 1)`. As with `lm()`, when there is an interaction, focus first on the interaction term.

Reporting (template)

- We modelled plant height as a function of week, temperature, and their interaction using a linear mixed model with plant identity as a random effect.

- Height increased with week (fixed effect of week), and plants at higher temperature were taller on average (fixed effect of temperature).
- All fixed effects had a p-value < 0.05, calculated using Satterthwaite's method for degrees of freedom (lmerTest package).
- Including plant as a random effect accounted for non-independence due to repeated measures.

Review

- Mixed models are used when observations are **grouped** (non-independent).
- They combine **fixed effects** (average relationships) with **random effects** (group-to-group variation).
- Random intercepts model different baselines among groups; random slopes allow different responses.
- Nested and crossed random effects reflect study design.
- Mixed models let you keep all observations while avoiding pseudoreplication and false confidence.

Further reading (optional): mixed models

In this course, there is only a short introduction to mixed (multilevel) models. Students who are curious and would like to explore this topic further (purely for their own interest) may find the following resources useful. Material from these resources will not be examined in the final exam, unless it is also already present in the course book.

McElreath, R. – Statistical Rethinking This book offers an excellent conceptual introduction to hierarchical (mixed) models, with a strong focus on understanding why they are useful. It introduces ideas such as partial pooling and shrinkage in a very intuitive way. The book uses a Bayesian perspective, but the conceptual insights are valuable even if you later apply frequentist methods. Strongly recommended by Owen for conceptual understanding.

Gelman, A. & Hill, J. – Data Analysis Using Regression and Multilevel/Hierarchical Models A classic and very practical text on regression and mixed models, with many applied examples. It complements Statistical Rethinking well and is particularly useful if you want to understand mixed models as they are commonly used in practice.

R documentation and tutorials for mixed-model packages If you are interested in implementation in R, the vignettes and documentation for packages such as lme4 (frequentist) or brms (Bayesian) provide hands-on examples of fitting, interpreting, and extending mixed models.

These resources are entirely optional and intended for students who wish to deepen their understanding beyond the scope of the course.

What next (L11-2)

Learning objectives

After this unit, students will be able to:

1. Describe which kinds of data the methods in BIO144 can analyse: data that meet the assumptions of linear models, count and binary data using generalised linear models, grouped or repeated-measures data using mixed models, and many variables at once using ordination.
2. Explain why data collected over time or across space often violate the assumption of independent observations, and so require specialised methods (time series analysis, spatial analysis).
3. Recognise features of a research question or dataset that call for methods beyond BIO144, such as non-independent observations, non-linear relationships, abrupt changes in a relationship, many interrelated variables, time-to-event responses, or results from multiple studies.
4. Match a described research situation to a suitable family of methods: time series analysis, spatial analysis, non-linear regression, breakpoint analysis, generalised additive models, structural equation modelling, machine learning, meta-analysis, survival analysis, non-parametric methods, power analysis or Bayesian statistics.
5. Explain why the analysis, including the sample size needed (power analysis), should be planned when the study is designed.
6. Explain why the choice of analysis should be driven by the research question rather than by the techniques available.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.

The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough

This chapter has no R code. The Unit 11 code walkthrough script and video are at the top of the Mixed models chapter.

During this course, you have learned a variety of data analysis techniques using R, including data manipulation, visualization, and statistical modeling. You have gained a solid foundation in using R for data analysis, you can analyse data that meet the assumptions of linear models, and some types of data that do not: count and binary data using GLMs, and grouped or repeated-measures data using mixed models. You have also learned to summarise many variables at once using ordination.

Of course, there is much more to learn! There are many opportunities for you to further develop your skills in R and data analysis. What is the next step after this course? What are the options to further improve your skills in data analysis in R? What other types of analyses could you learn about, and when might you need them?

Here are a list of other types of problem / question that we might have, and types of analysis that could be relevant. The list is by no means exhaustive, but it should give you some ideas of what to explore next, and what to explore when you encounter specific types of data or research questions.

- **Time series analysis**: If your data are collected over time (e.g., daily, monthly, yearly), you might need to learn about time series analysis techniques such as ARIMA models, seasonal decomposition, and forecasting methods. A key feature of time series data is that observations are not independent, which violates assumptions of many standard statistical methods. This needs to be carefully handled in the analysis.
- **Spatial analysis**: If your data have a spatial component (e.g., locations, regions), you might need to learn about spatial statistics, geostatistics, and spatial modeling techniques. This could include methods such as kriging, spatial autocorrelation analysis, and spatial regression models. Again, spatial data often violate independence assumptions, requiring specialized methods.
- **Non-linear regression**: If the relationship between your explanatory variables and response variable is not linear, you might need to learn about non-linear regression techniques. These can estimate the parameters of specific non-linear functions, and to assess the goodness of fit.
- **Breakpoint analysis**: If you suspect that there are changes in the relationship between variables at certain points (e.g., before and after an intervention), you might need to learn about breakpoint analysis techniques, such as piecewise regression or change point detection methods.
- **Generalized Additive Models (GAMs)**: If you want to model com-

plex, non-linear relationships between explanatory variables and response variables while maintaining some interpretability, you might need to learn about GAMs. These models use smooth functions to capture non-linear effects. They are rather elegant!

- **Structural Equation Modeling (SEM)**: If you want to analyze complex relationships among multiple variables, including latent variables, you might need to learn about SEM techniques. SEM allows for the modeling of direct and indirect effects, as well as measurement error. Effectively, we can build and test complex causal models. Variables can be both explanatory variables and responses at the same time.
- **Machine Learning**: If you want to make predictions or classify data based on patterns, you might need to learn about machine learning techniques such as decision trees, random forests, support vector machines, and neural networks. These methods can handle large datasets and complex relationships but may sacrifice some interpretability.
- **Meta-analysis**: If you want to synthesize results from multiple studies to draw broader conclusions, you might need to learn about meta-analysis techniques. This involves combining effect sizes from different studies and assessing heterogeneity among them.
- **Survival analysis**: If your data involve time-to-event response variables (e.g., time until failure, time until death), you might need to learn about survival analysis techniques such as Kaplan-Meier estimation, Cox proportional hazards models, and parametric survival models.
- **Non-parametric methods**: If your data do not meet the assumptions of parametric tests (e.g., normality, homoscedasticity), and you really can't figure out how to make a parametric model (e.g., LM or GLM) you might need to learn about non-parametric methods such as rank-based tests, bootstrapping, and permutation tests.
- **Power analysis and sample size estimation**: If you want to design studies with adequate statistical power, you might need to learn about power analysis techniques. This involves calculating the required sample size based on effect sizes, significance levels, and desired power.
- **Bayesian statistics**: If you want to incorporate prior knowledge and uncertainty into your analyses, you might need to learn about Bayesian statistical methods. This involves using Bayes' theorem to update prior beliefs based on observed data.

These are just a few examples of the many types of analyses that you might encounter in your data analysis journey. The choice of which techniques to learn next will depend on your specific research questions, data characteristics, and goals. Ideally you will plan your analyses when you design your study, so that you can collect the right type of data to answer your questions. When you don't

or when something changes, you can then explore and discussion with experts which techniques are most appropriate.

A final word of advice… try to not be driven by techniques. Instead, be driven by your research questions. After all, we are not doing data analysis for its own sake, but to answer questions about the world around us. Let your questions guide your learning journey!

Review: planning and implementing statistical analyses (L12)

Learning objectives

After this unit, students will be able to:

1. Write a planning sentence for a described study, stating the question, the type of response variable, the data structure, the explanatory variables, the planned model and the expected residual degrees of freedom.
2. Choose a model (linear model, binomial GLM, Poisson GLM, mixed model or ordination) from the type of response variable, the explanatory variables and the data structure of a described study.
3. Identify the experimental unit in a described study, and recognise pseudoreplication.
4. Explain the roles of replication, randomisation, controls, blocking and balance in study design.
5. Count the parameters a planned model will estimate, calculate the expected residual degrees of freedom, and judge whether they are enough.
6. Explain how sample size affects the precision of estimates and the power of a study, and what a power analysis is for.
7. Decide which terms to include in a model from the question and the design (variables of interest, planned interactions, design variables, potential confounders), and explain why trying many models or using stepwise selection gives misleading p-values.
8. Check a fitted model before interpreting it, make explicit predictions from it, and plot the predictions together with the data.
9. Recognise a report that gives both the statistical result and its biological meaning.

These learning objectives, together with the content of this chapter, define what you need to know for the exam.
The learning objectives for all chapters, and for the course as a whole, are also given on the Learning Objectives page of the course information website.

Code walkthrough for this chapter

Script: walkthrough_ch12_review.R
This R script contains the code of this chapter, with comments that explain each step. Download it, save it in your BIO144 RStudio project folder, open it in RStudio, and run it line by line while you read the chapter or watch the video. The script reads the datasets directly from the internet. Its last section ("Practical toolbox") shows code that the Unit 12 practical uses but this chapter does not.
To download the script straight into your project folder (for example on RStudio Server), run this line in the Console:

```
download.file("https://opetchey.github.io/BIO144_Course_Book/walkthroughs/walkthrough
destfile = "walkthrough_ch12_review.R")
```

Video walkthrough: coming soon.

Introduction

In earlier chapters we learned many different statistical tools:

- linear regression
- ANOVA
- multiple regression
- interactions
- Poisson GLMs for count data
- binomial GLMs for binary data
- ordination
- mixed models

This final chapter pulls those ideas together. The aim is not to memorize a list of tests. The aim is to move carefully from a **biological question** to a **planned model**, and then from a fitted model to a checked, interpreted, plotted, and reported result.

! Important

Core idea
Do not start with "Which test do I use?" Start with the biological question, sketch the expected figure, identify the response variable and data structure,

> then choose the model.

The chapter has two parts:

1. **Planning**: what to decide before collecting or analyzing data.
2. **Implementation**: what to do once the data exist.

Part 1: Planning the analysis

Statistics does not begin after data collection. In a good study, the statistical plan is part of the study design.

Before collecting data, we should already be able to say:

> Here is my biological question.
> Here is the figure I expect to make.
> Here is the response variable.
> Here is the model I plan to fit.
> Here is roughly how many observations I need, and how many residual degrees of freedom I expect.

This does not mean the plan can never change. Real data can surprise us. But without a plan, we often discover too late that the design does not answer the question.

The planning workflow

```
Biological question
        |
        v
Sketch expected figure
        |
        v
Define response variable
        |
        +-- Continuous -> LM / Gaussian model
        +-- Binary / proportion -> binomial GLM
        +-- Count -> Poisson/count GLM
        +-- Multivariate -> ordination / multivariate methods
        |
        v
Check data structure
        |
        +-- independent observations -> ordinary LM / GLM
        +-- grouped / repeated / blocked / nested -> mixed model / random effects
        |
```

```
        v
Define explanatory variables
        |
        +-- categorical -> group differences
        +-- continuous -> slopes
        +-- categorical + continuous -> ANCOVA-style model
        +-- multiple predictors -> multiple model
        +-- interactions -> one effect depends on another
        |
        v
Planned statistical model
        |
        v
Plan sample size and expected residual df
```

Traditional names like ANOVA, regression, and ANCOVA are useful, but they are not the deepest decision. They are mostly different fixed-effect structures inside the linear model.

The deeper questions are:

1. What kind of response variable do I have?
2. Are the observations independent, or is there grouping/repeated structure?
3. What kind of explanatory variables do I have?

The planning sentence

A useful planning sentence is:

> We will test whether **X** affects (or is related with) **Y** using **model Z**, because **Y** is [response type], **X** is [predictor type], and the observations are [independent/grouped]. We plan to collect **n = ...** observations, giving **n minus the number of estimated parameters** residual degrees of freedom.

This sentence is not just a reporting sentence. It is a design sentence. If you cannot complete it, the study plan (including question) is probably not clear enough yet.

> **Note**
>
> Residual degrees of freedom are not magic numbers that appear later in R. They are already implied by the design and the model.

Example 1: Comparing treatment means

Biological question:

Does fertilizer treatment affect plant biomass?

Planning step	Decision
Expected figure	Biomass on the y-axis, fertilizer treatment on the x-axis; probably a boxplot, dot plot, or means with confidence intervals
Response variable	Plant biomass, continuous
Data structure	Each plant is measured once and plants are independent
Explanatory variables	Fertilizer treatment, categorical: control, low fertilizer, high fertilizer
Planned model	Gaussian linear model with biomass as the response and fertilizer treatment as the explanatory variable
Traditional name	One-way ANOVA, also understood as a linear model with one categorical predictor

Example model:

```
m_biomass_fertilizer <- lm(biomass ~ fertilizer, data = plants)
```

Suppose we plan 20 plants per treatment and 3 treatments. That gives $n = 60$ observations.

The model estimates 3 treatment means, so the expected residual df is approximately:

```
60 - 3 = 57
```

Planning sentence:

> We will test whether fertilizer treatment affects plant biomass using a Gaussian linear model, because biomass is continuous, fertilizer treatment is categorical, and each plant is measured independently. We plan to collect 60 observations, 20 per treatment, giving an expected residual df of about 57.

Example 2: A continuous predictor

Biological question:

> Does temperature affect growth rate?

Planning step	Decision
Expected figure	Growth rate on the y-axis, temperature on the x-axis; a scatterplot with a fitted line and uncertainty band
Response variable	Growth rate, continuous
Data structure	Each experimental unit is measured once and observations are independent
Explanatory variables	Temperature, continuous
Planned model	Gaussian linear model with a continuous predictor
Traditional name	Linear regression

Example model:

```
m_growth_temperature <- lm(growth_rate ~ temperature, data = growth)
```

Suppose we plan $n = 50$ observations across the temperature gradient.

The model estimates an intercept and a slope, so the expected residual df is:

```
50 - 2 = 48
```

Planning sentence:

> We will test whether temperature affects growth rate using a Gaussian linear model, because growth rate is continuous, temperature is continuous, and observations are independent. We plan to collect 50 observations, giving an expected residual df of about 48.

The expected figure helps clarify the model. If the figure is a scatterplot with one fitted line, the model is probably a regression-type model.

Example 3: Grouping structure and random effects

Biological question:

> Does a drug treatment affect heart rate when each animal is measured repeatedly over time?

Planning step	Decision
Expected figure	Heart rate on the y-axis, time or treatment on the x-axis, with repeated measurements connected or grouped by animal
Response variable	Heart rate, continuous

Planning step	Decision
Data structure	Observations are not independent, because each animal contributes multiple measurements
Explanatory variables	Treatment is categorical; time may be continuous or categorical, depending on the design
Planned model	Mixed model, because repeated measurements are grouped within animals
Random effect	Animal identity, because measurements are repeated within animal

Example model:

```
m_heart_rate <- lmer(heart_rate ~ treatment + time + (1 | animal_id), data = heart)
```

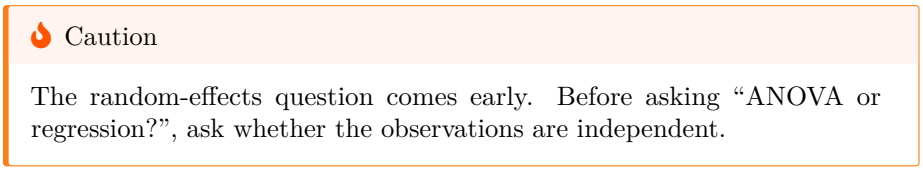
If the biological question is whether the treatment changes the trajectory through time, then plan an interaction:

```
m_heart_rate_interaction <- lmer(heart_rate ~ treatment * time + (1 | animal_id), data = heart)
```

Here we need to plan both the number of animals and the number of measurements per animal. For example, 20 animals measured 5 times each gives 100 observations, but those 100 observations are not equivalent to 100 independent animals.

Planning sentence:

> We will test whether drug treatment affects heart rate over time using a mixed model, because heart rate is continuous, treatment and time are explanatory variables, and repeated measurements are grouped within animals. We plan to measure 20 animals 5 times each, giving 100 total observations, while accounting for animal identity as a random effect.



Example 4: Binary response

Biological question:

> Does habitat type affect whether individuals are infected?

Planning step	Decision
Expected figure	Infection probability on the y-axis and habitat type on the x-axis, perhaps shown as proportions with confidence intervals
Response variable	Infection status: infected or not infected, binary
Data structure	Each individual is sampled once and observations are independent
Explanatory variables	Habitat type, categorical
Planned model	Binomial GLM

Example model:

```
m_infection_habitat <- glm(infected ~ habitat, family = binomial, data = infection)
```

Suppose we plan 50 individuals in each of two habitats, giving $n = 100$ observations. With two habitat groups, the expected residual df is roughly:

```
100 - 2 = 98
```

Planning sentence:

> We will test whether infection probability differs between habitats using a binomial GLM, because infection status is binary, habitat is categorical, and individuals are independent. We plan to sample 100 individuals, 50 per habitat, giving an expected residual df of about 98.

The explanatory variable could be categorical or continuous, just as in a linear model. What changes the model class is the response variable.

Designing the study so that it is easy to analyse

The planning sentence assumes that the data will be collected in a way that can answer the question. A few principles of study design make this much more likely. You have met each of them in earlier chapters; here they are together.

- **Identify the experimental unit.** The experimental unit is the thing to which a treatment is independently applied (or, in an observational study, the independent thing we sample). It is the unit of replication. If fertiliser is applied to whole pots, and we measure five plants per pot, the pot is the experimental unit, not the plant.
- **Replicate.** We need several independent experimental units per treatment (or across the range of a continuous explanatory variable). Replication is what lets us estimate the residual variance and so the uncertainty of our estimates.

- **Avoid pseudoreplication.** Measuring the same unit several times, or measuring several subunits within a unit, does not give independent replicates. Either analyse the units (for example, using averages per unit), or model the grouping with a mixed model (Chapter 11). Treating subunits as independent overstates the residual degrees of freedom and gives over-confident results.
- **Randomise.** Assign treatments to experimental units at random. Randomisation breaks any link between the treatment and other variables that might affect the response, which is why experiments give the strongest evidence for causal effects (Chapters 4 and 6).
- **Include controls.** A comparison group (for example, no fertiliser) is needed to know what would have happened without the treatment.
- **Block when units differ in known ways.** If experimental units differ in a known way (for example, different greenhouse benches, or different days), put a full set of treatments in each block, and include the block in the model, as a fixed or random effect. This removes the variation between blocks from the residual variation and makes treatment comparisons more precise.
- **Aim for balance.** Equal numbers of units per treatment (or per combination of treatments) make the analysis simpler and more robust, and make the estimates as precise as possible for a given total effort.
- **In observational studies, plan for confounders.** List the variables that might be associated with both the explanatory variable of interest and the response, and plan to measure them (Chapter 6).

How many observations? Sample size, precision and power

More observations give more precise estimates. The standard error of an estimate is roughly proportional to $1/\sqrt{n}$, so to halve the width of a confidence interval we need about four times as many observations. More observations also give more residual degrees of freedom, and we saw in Chapter 3 why having few residual degrees of freedom is a problem: the residual variance is estimated imprecisely and the critical t-values are large.

A few practical guidelines:

- Count the parameters the planned model will estimate (including the intercept, each slope, $k-1$ parameters for each categorical variable with k levels, and the parameters for any interactions), and check that the planned number of observations leaves plenty of residual degrees of freedom. A common rough guide is to aim for at least 10–20 observations per estimated parameter.
- Remember that the number that matters is the number of *independent* experimental units, not the number of measurements (see pseudoreplication above).
- The **power** of a study is the probability that it detects an effect of a given size, if that effect really exists. Power increases with the number of

observations and with the size of the effect, and decreases with the residual variation. A formal **power analysis** uses these quantities to calculate the sample size needed. We do not cover how to do a power analysis in this course (see Chapter 11.2), but you should know what it is and why it is done before collecting data.

Deciding which terms go into the model

The terms in the model should follow from the question and the design, and should be decided **before** looking at the results.

- **The explanatory variables of interest** come from the question.
- **Interactions** are included when the question is about whether one effect depends on another (Chapter 7), or when the design is fully crossed (factorial) and an interaction is biologically plausible.
- **Design variables** are included because of how the data were collected: blocks, and grouping variables as random effects (Chapter 11). They are included whether or not they turn out to be "significant".
- **Potential confounders** (in observational studies) are included because of biological knowledge that they may affect both the response and the explanatory variables of interest (Chapter 6).

What to avoid:

- **Trying many models and reporting the one that gives the result you hoped for.** Adding and removing terms, transformations or observations until a p-value becomes small is a form of "p-hacking" (Chapter 3). The p-values from the final model are then misleading, because they do not take into account all the models that were tried.
- **Automatic variable selection** (such as stepwise selection, which adds or removes terms one by one according to their p-values). It suffers from the same problem.

In this course we test terms with F-tests (for linear models) and likelihood ratio (chi-squared) tests (for GLMs), comparing models that differ only in the terms relevant to the planned question. Other approaches to choosing among models, such as information criteria (e.g. AIC), are not covered in this course. If an analysis is exploratory, rather than planned, say so clearly when reporting it.

Part 2: Implementing the analysis

Once we have the data, the key implementation question is:

> Did the model behave well enough that we can interpret it?

The workflow is:

```
Fit planned model
        |
```

```
        v
Check model assumptions / diagnostics
        |
        +-- okay -> interpret model output
        +-- not okay -> revise model / family / structure
        |
        v
Interpret output
        |
        v
Make explicit predictions
        |
        v
Plot data + model predictions
        |
        v
Report statistical and biological conclusion
```

> **!** Important
>
> The order matters. We do not fit a model, look for a small p-value, and only then check whether the model was reasonable. We check first, then interpret.

Good practice is to make predictions explicitly from the fitted model and then plot those predictions. The figure should show the model you fitted, not a model quietly fitted by the plotting function.

Example 1: Linear model with an interaction

Biological question:

> Does body size affect metabolic rate, and does this body-size effect differ among three species?

This example connects several ideas from the course:

- regression
- treatment groups
- interactions
- diagnostic plots
- reading model output
- degrees of freedom
- explicit prediction and plotting

Planning recap

Planning step	Decision
Expected figure	Scatterplot of metabolic rate against body size, with separate fitted lines for each species
Response variable	`metabolic_rate`, continuous
Data structure	Independent observations, assuming each individual is measured once
Explanatory variables	`body_size`, continuous; `species`, categorical
Planned model	Gaussian linear model with a species-by-body-size interaction
Approximate residual df	Number of observations minus number of estimated parameters

Planning sentence:

> We will test whether body size affects metabolic rate differently among species using a Gaussian linear model with a species-by-body-size interaction, because metabolic rate is continuous, body size is continuous, species is categorical, and observations are independent.

An example dataset

Reading in the dataset:

```
metabolism <- read_csv("datasets/metabolism_species_body_size.csv")
```

```
Rows: 135 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): species
dbl (2): body_size, metabolic_rate

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the dataset:

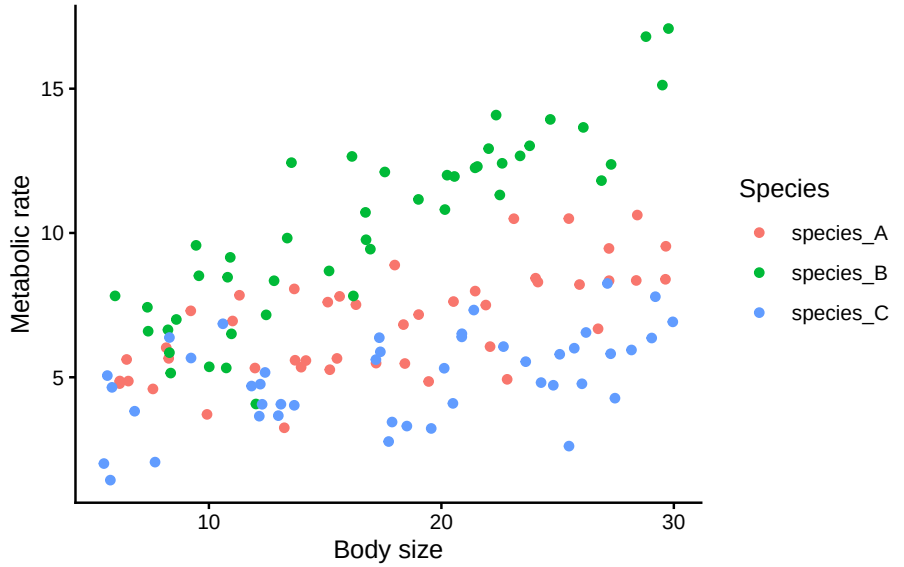
```
metabolism |>
  slice_head(n = 6)
```

```
# A tibble: 6 x 3
  species   body_size metabolic_rate
  <chr>         <dbl>          <dbl>
1 species_A      9.92           3.71
```

```
2 species_A     27.2            8.35
3 species_A     14.2            5.58
4 species_A     13.7            8.06
5 species_A     18.4            6.82
6 species_A     18.4            5.48
```

Planned figure:

```
ggplot(metabolism, aes(body_size, metabolic_rate, colour = species)) +
  geom_point() +
  labs(
    x = "Body size",
    y = "Metabolic rate",
    colour = "Species"
  ) +
  theme_classic()
```



Fit the planned model

The planned model has:

- a continuous response: `metabolic_rate`
- one continuous explanatory variable: `body_size`
- one categorical explanatory variable: `species`
- an interaction: `body_size:species`

```
m_metabolism <- lm(metabolic_rate ~ species * body_size, data = metabolism)
```

The model formula:

```
metabolic_rate ~ species * body_size
```

means:

```
metabolic_rate ~ species + body_size + species:body_size
```

The * means main effects plus the interaction. We use it when the biological question is whether the effect of one predictor depends on another predictor.

Check the model before interpreting

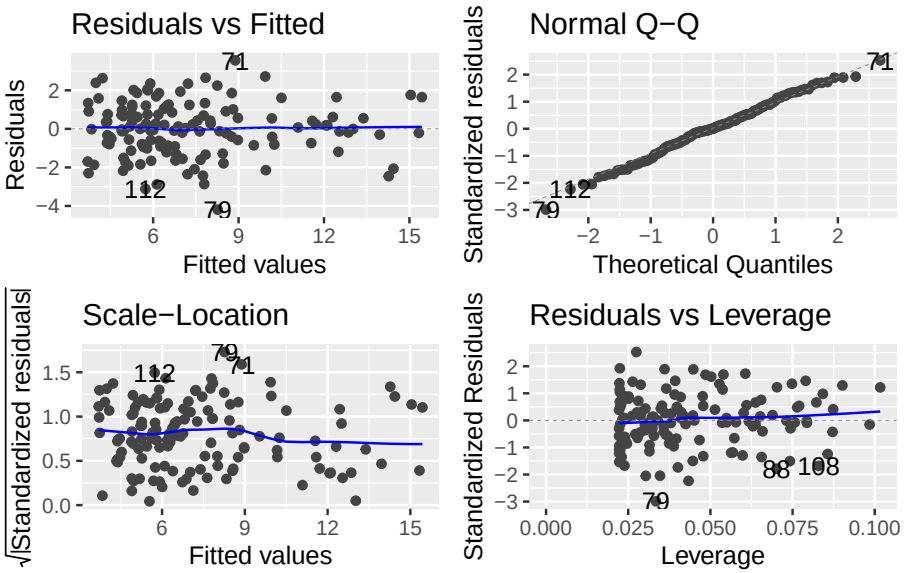
For a linear model, the common diagnostic questions are:

Diagnostic plot	Question
Residuals vs fitted	Is there obvious nonlinearity or unequal variance?
QQ plot	Are residuals approximately normal?
Scale-location	Does residual spread change with fitted values?
Residuals vs leverage	Are a few observations dominating the model?

```
autoplot(m_metabolism)
```

```
Warning: `fortify(<lm>)` was deprecated in ggplot2 4.0.0.
i Please use `broom::augment(<lm>)` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```



We are not looking for perfect plots. We are looking for obvious problems that would make the model inappropriate.

Possible actions if diagnostics look bad:

- reconsider whether the response should be transformed
- reconsider whether the relationship is nonlinear
- check for influential observations
- check whether important grouping structure was ignored
- for non-continuous responses, consider a GLM family instead of an LM

Interpret the output

```
summary(m_metabolism)
```

```
Call:
lm(formula = metabolic_rate ~ species * body_size, data = metabolism)

Residuals:
    Min      1Q  Median      3Q     Max
-4.1918 -0.8628 -0.0026  0.9103  3.5532

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                3.83601    0.56726   6.762 4.25e-
10 ***
```

```
speciesspecies_B           -0.42823    0.80217  -0.534   0.5944
speciesspecies_C           -0.68723    0.79494  -0.865   0.3889
body_size                   0.17335    0.03001   5.777 5.40e-
08 ***
speciesspecies_B:body_size  0.23073    0.04327   5.333 4.20e-
07 ***
speciesspecies_C:body_size -0.07204    0.04127  -1.746   0.0832 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.426 on 129 degrees of freedom
Multiple R-squared:  0.8028,    Adjusted R-squared:  0.7952
F-statistic:   105 on 5 and 129 DF,  p-value: < 2.2e-16
```

```
anova(m_metabolism)
```

```
Analysis of Variance Table

Response: metabolic_rate
                   Df Sum Sq Mean Sq F value    Pr(>F)
species             2 632.80  316.40 155.492 < 2.2e-16 ***
body_size           1 324.39  324.39 159.417 < 2.2e-16 ***
species:body_size   2 111.55   55.78  27.411 1.201e-10 ***
Residuals         129 262.50    2.03
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Use this table to orient yourself:

Question level	Model/output	Common test statistic	What it asks
Model / term level	ANOVA table for LM	`F`	Does this explanatory variable or term explain variation in the response?
Model / term level	Deviance comparison for GLM	Deviance / chi-square-like test	Does adding/removing this term improve the model?
Coefficient level	Linear model coefficients	`t`	Is this estimated coefficient different from zero?

Question level	Model/output	Common test statistic	What it asks
Coefficient level	Binomial/Poisson GLM coefficients	`z`	Is this estimated coefficient different from zero on the link scale?
Coefficient level	Confidence interval	CI	What range of effect sizes is compatible with the data/model?

`summary()` often gives coefficient-level tests. `anova()` often gives term-level tests. These are related, but they are not always asking exactly the same question.

Residual degrees of freedom:

```
df.residual(m_metabolism)
```

```
[1] 129
```

Residual degrees of freedom are the amount of information left for estimating error after fitting the model. Here they are roughly the number of observations minus the number of estimated parameters.

```
nrow(metabolism)
```

```
[1] 135
```

```
length(coef(m_metabolism))
```

```
[1] 6
```

```
nrow(metabolism) - length(coef(m_metabolism))
```

```
[1] 129
```

Make explicit predictions

Do not use `geom_smooth()` as the model.

Instead:

1. fit the model
2. create new data containing the predictor values we want predictions for
3. use `predict()`
4. plot the observed data and model predictions

```
new_metabolism <- expand_grid(
  species = c("species_A", "species_B", "species_C"),
```

```
  body_size = seq(5, 30, length.out = 100)
)

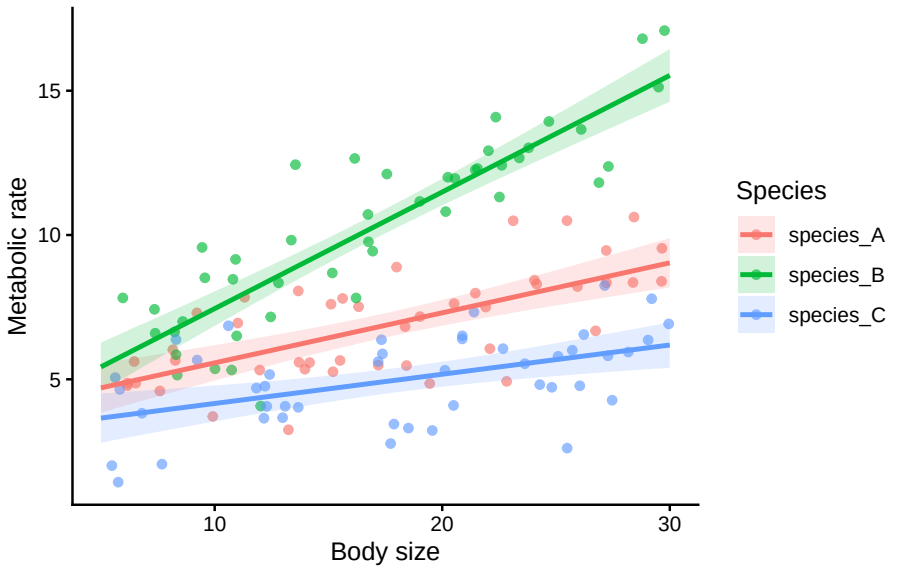
pred_metabolism <- predict(
  m_metabolism,
  newdata = new_metabolism,
  interval = "confidence"
)

plot_metabolism <- bind_cols(
  new_metabolism,
  as_tibble(pred_metabolism)
)

plot_metabolism |>
  slice_head(n = 6)
```

```
# A tibble: 6 x 5
  species   body_size   fit   lwr   upr
  <chr>         <dbl> <dbl> <dbl> <dbl>
1 species_A      5     4.70  3.85  5.56
2 species_A      5.25  4.75  3.91  5.59
3 species_A      5.51  4.79  3.96  5.62
4 species_A      5.76  4.83  4.02  5.65
5 species_A      6.01  4.88  4.08  5.68
6 species_A      6.26  4.92  4.13  5.71
```

```
ggplot(metabolism, aes(body_size, metabolic_rate, colour = species)) +
  geom_point(alpha = 0.65) +
  geom_ribbon(
    data = plot_metabolism,
    aes(x = body_size, ymin = lwr, ymax = upr, fill = species),
    alpha = 0.18,
    colour = NA,
    inherit.aes = FALSE
  ) +
  geom_line(data = plot_metabolism, aes(y = fit), linewidth = 1) +
  labs(
    x = "Body size",
    y = "Metabolic rate",
    colour = "Species",
    fill = "Species"
  ) +
  theme_classic()
```



The model and the figure now match. The lines and ribbons come from `m_metabolism`, the model we fitted and checked.

Report

Example structure:

> We fitted a Gaussian linear model testing whether metabolic rate depended on body size, species, and their interaction. Diagnostic plots did not show major violations of model assumptions. Metabolic rate changed with body size, but the slope differed among species: the relationship was strongest in species B and weakest in species C. This indicates that species changed the relationship between body size and metabolism.

If giving formal results, add the relevant coefficient or term-level test from the output.

Example 2: Binomial GLM

Biological question:

> Does habitat type affect whether individuals are infected?

Planning recap

Planning step	Decision
Expected figure	Infection probability or observed proportion infected by habitat
Response variable	`infected`, binary: 0 = not infected, 1 = infected
Data structure	Independent observations, assuming each individual is sampled once
Explanatory variables	`habitat`, categorical
Planned model	Binomial GLM
Approximate residual df	Number of observations minus number of estimated habitat probabilities

Planning sentence:

> We will test whether infection probability differs between habitats using a binomial GLM, because infection status is binary, habitat is categorical, and observations are independent.

An example dataset

Reading in the dataset:

```
infection <- read_csv("datasets/infection_habitat.csv")
```

```
Rows: 160 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): habitat
dbl (1): infected

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Here are the first few rows of the dataset:

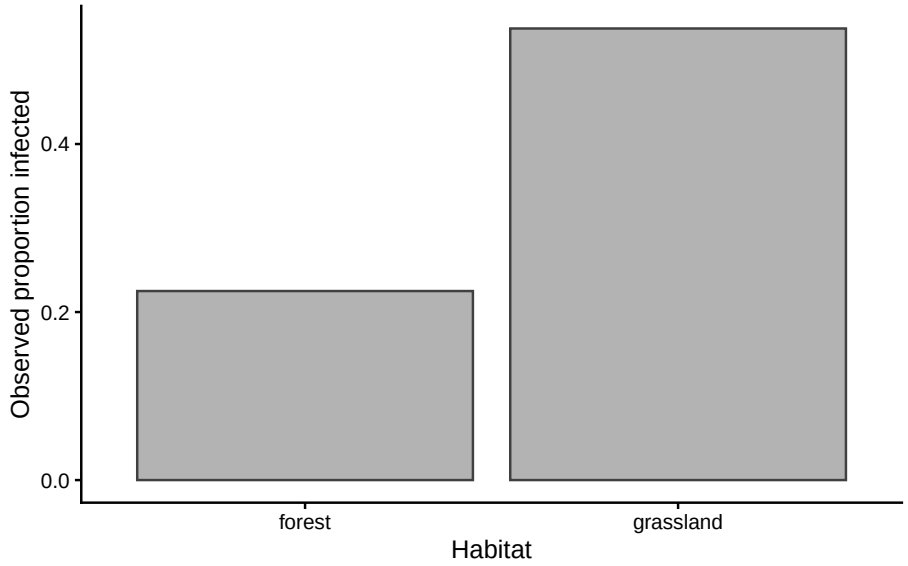
```
infection |>
  slice_head(n = 6)
```

```
# A tibble: 6 x 2
  habitat infected
  <chr>      <dbl>
1 forest         0
2 forest         1
3 forest         0
4 forest         0
5 forest         0
6 forest         0
```

Planned figure:

```
infection_summary <- infection |>
  group_by(habitat) |>
  summarise(
    infected = mean(infected),
    .groups = "drop"
  )

ggplot(infection_summary, aes(habitat, infected)) +
  geom_col(fill = "grey70", colour = "grey25") +
  labs(
    x = "Habitat",
    y = "Observed proportion infected"
  ) +
  theme_classic()
```



Fit the planned model

```
m_infection <- glm(
  infected ~ habitat,
  family = binomial,
  data = infection
)
```

Check before interpreting

For a binomial GLM, ask:

- Did I use the correct response type and family?
- Are observations independent?
- Are there obvious influential observations?
- Is there overdispersion? (This question applies to counts and to grouped binomial data, i.e. successes out of a number of trials.)

The simple overdispersion check (residual deviance divided by residual degrees of freedom, roughly 1 if there is no overdispersion) is **not** useful here. Our response is individual-level 0/1 data (each row is one individual, infected or not). For such data the ratio says nothing about overdispersion, so we do not calculate it. (See the GLM checklist in Chapters 8 and 9.) We will use this check for the count data in the next example.

Interpret the output

```
summary(m_infection)
```

```
Call:
glm(formula = infected ~ habitat, family = binomial, data = infection)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       -1.2368     0.2677  -4.619 3.85e-06 ***
habitatgrassland   1.3870     0.3492   3.972 7.14e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 212.70  on 159  degrees of freedom
Residual deviance: 195.76  on 158  degrees of freedom
AIC: 199.76

Number of Fisher Scoring iterations: 4
```

In a binomial GLM, coefficients are on the log-odds scale by default. That scale is useful for the model, but not always intuitive biologically.

Predicted probabilities are often easier to interpret.

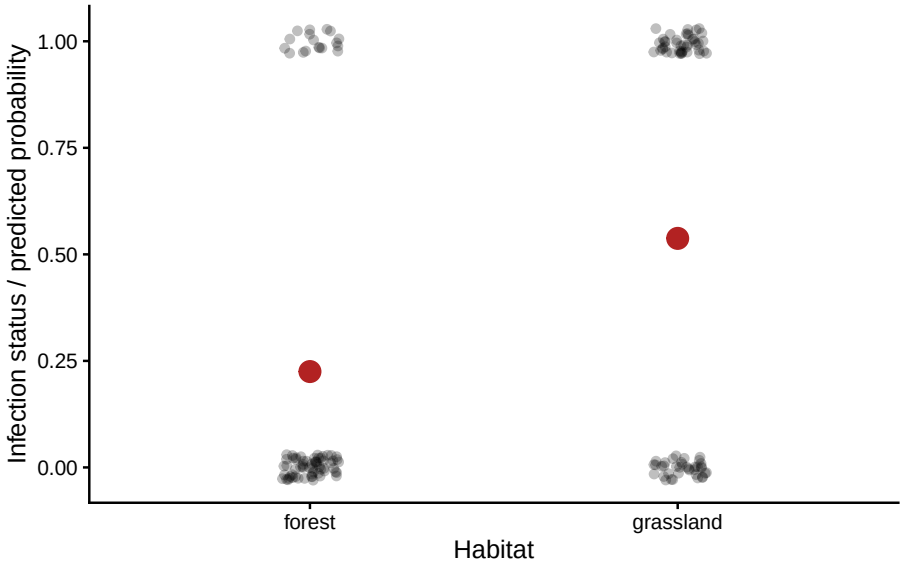
```
    predicted_probability = predict(
      m_infection,
      newdata = pick(everything()),
      type = "response"
    )
  )

new_infection
```


Make explicit predictions

```
# A tibble: 2 x 2
  habitat   predicted_probability
  <chr>                     <dbl>
1 forest                    0.225
2 grassland                 0.538
```

```
ggplot(infection, aes(habitat, infected)) +
  geom_jitter(width = 0.08, height = 0.03, alpha = 0.25) +
  geom_point(
    data = new_infection,
    aes(y = predicted_probability),
    size = 4,
    colour = "firebrick"
  ) +
  labs(
    x = "Habitat",
    y = "Infection status / predicted probability"
  ) +
  theme_classic()
```



For GLMs, `type = "response"` gives predictions on the biological scale. For a binomial GLM, that means predicted probabilities.

Report

Example structure:

> We fitted a binomial GLM testing whether infection probability differed between habitats. The model used infection status as a binary response and habitat as a categorical explanatory variable. After checking model fit, we interpreted the model on the probability scale: predicted infection probability was higher in grassland than forest.

Example 3: Count GLM

Biological question:

> Does food availability affect the number of offspring produced?

Planning recap

Planning step	Decision
Expected figure	Scatterplot of offspring number against food availability, with a fitted prediction line

Planning step	Decision
Response variable	`n_offspring`, count
Data structure	Independent observations, assuming each individual or reproductive unit is measured once
Explanatory variables	`food`, continuous
Planned model	Poisson GLM, with overdispersion checked after fitting
Approximate residual df	Number of observations minus number of estimated parameters

Planning sentence:

> We will test whether food availability affects offspring number using a Poisson GLM, because offspring number is a count, food availability is continuous, and observations are independent.

An example dataset

Reading in the dataset:

```
offspring <- read_csv("datasets/offspring_food.csv")
```

```
Rows: 90 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): food, n_offspring

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

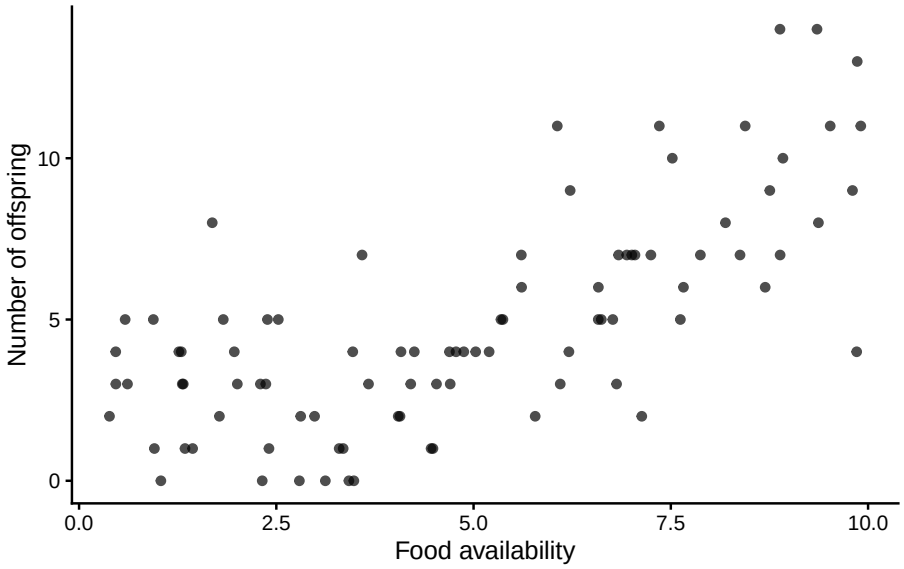
Here are the first few rows of the dataset:

```
offspring |>
  slice_head(n = 6)
```

```
# A tibble: 6 x 2
   food n_offspring
  <dbl>       <dbl>
1  1.97           4
2  8.89           7
3  3.67           3
4  3.47           4
5  5.35           5
6  5.37           5
```

Planned figure:

```
ggplot(offspring, aes(food, n_offspring)) +
  geom_point(alpha = 0.7) +
  labs(
    x = "Food availability",
    y = "Number of offspring"
  ) +
  theme_classic()
```



Fit the planned model

```
m_offspring <- glm(
  n_offspring ~ food,
  family = poisson,
  data = offspring
)
```

Check before interpreting

```
deviance(m_offspring) / df.residual(m_offspring)
```

```
[1] 1.338865
```

For count models, overdispersion is a common issue. If the residual deviance is much larger than the residual df, the Poisson model may be too simple.

Here the ratio is 1.34. This is somewhat above 1, but below the rule-of-thumb

values of about 1.5–2, so there is no strong sign of overdispersion and the Poisson model is acceptable. (If it were much larger, we would consider a quasi-Poisson or negative binomial model.)

Interpret the output

```
summary(m_offspring)
```

```
Call:
glm(formula = n_offspring ~ food, family = poisson, data = offspring)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.61383    0.12057   5.091 3.56e-07 ***
food         0.17327    0.01782   9.721  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 216.70  on 89  degrees of freedom
Residual deviance: 117.82  on 88  degrees of freedom
AIC: 401.27

Number of Fisher Scoring iterations: 5
```

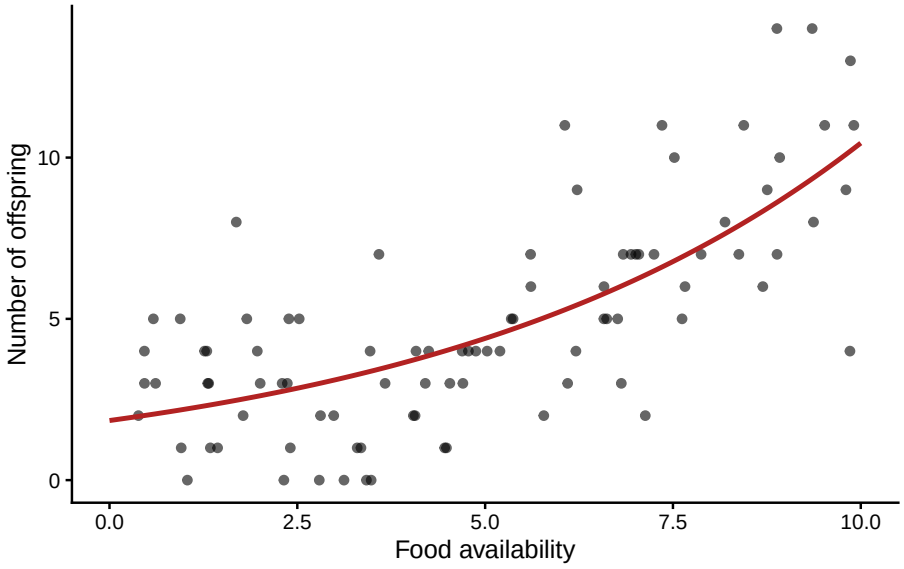
In a Poisson GLM, coefficients are on the log scale. A positive coefficient means the expected count increases as the predictor increases.

Make explicit predictions

```
new_offspring <- tibble(
  food = seq(0, 10, length.out = 100)
) |>
  mutate(
    predicted_count = predict(
      m_offspring,
      newdata = pick(everything()),
      type = "response"
    )
  )
```

```
ggplot(offspring, aes(food, n_offspring)) +
  geom_point(alpha = 0.6) +
  geom_line(
```

```
    data = new_offspring,
    aes(y = predicted_count),
    colour = "firebrick",
    linewidth = 1
  ) +
  labs(
    x = "Food availability",
    y = "Number of offspring"
  ) +
  theme_classic()
```



Report

Example structure:

> We fitted a Poisson GLM testing whether offspring number changed with food availability. Model checks did not suggest major overdispersion. Predicted offspring number increased with food availability, suggesting that food availability was positively associated with reproductive output.

Final checklist

Planning checklist:

1. What is the biological question?
2. What would the figure look like?

3. What is the response variable?
4. Are observations independent, grouped, repeated, blocked, or nested?
5. What are the explanatory variables?
6. What model follows from those choices?
7. What is the experimental unit, and how will treatments be replicated, randomised and (if needed) blocked?
8. Which terms will the model include, and why? (Decided before seeing the results.)
9. How many independent observations will we collect?
10. What residual degrees of freedom do we expect, and are they enough?

Implementation checklist:

1. Did I fit the model I planned?
2. Did I check assumptions or diagnostics before interpretation?
3. Am I reading the correct output for my question?
4. Am I asking a model/term-level question or a coefficient-level question?
5. Did I make explicit predictions from the fitted model?
6. Does my plot show the model I actually fitted?
7. Does my report include both the statistical result and the biological meaning?

The goal is not to memorize every possible test. The goal is to move carefully from biological question to planned model, and then from fitted model to checked, interpreted, plotted, and reported result.