Lecture materials | granolarr

Stefano De Sabbata

2021-03-23

Contents

Preface **5**
Session info 5

1 Introduction to R **7**
1.1 About this module 7
1.2 R programming language 7
1.3 Schedule 8
1.4 Reference books 8
1.5 R 9
1.6 Interpreting values 9
1.7 Basic types 9
1.8 Numeric operators 10
1.9 Logical operators 10
1.10 Summary 10

2 Core concepts **13**
2.1 Recap 13
2.2 Variables 13
2.3 Algorithms and functions 14
2.4 Functions 14
2.5 Functions and variables 14
2.6 Naming 15
2.7 Libraries 15
2.8 stringr 15
2.9 Summary 16

3 Tidyverse **17**
3.1 Recap 17
3.2 Tidyverse 17
3.3 Tidyverse core libraries 18
3.4 Tidyverse core libraries 18
3.5 Tidyverse core libraries 18
3.6 The pipe operator 19

3.7 Pipe example . . . 19
3.8 Pipe example . . . 19
3.9 Coding style . . . 20
3.10 Summary . . . 20

4 Data types **21**
4.1 Recap . . . 21
4.2 Vectors . . . 21
4.3 Defining vectors . . . 21
4.4 Creating vectors . . . 22
4.5 Selection . . . 22
4.6 Functions on vectors . . . 23
4.7 Any and all . . . 23
4.8 Factors . . . 23
4.9 table . . . 24
4.10 Specified levels . . . 24
4.11 (Unordered) Factors . . . 24
4.12 Ordered Factors . . . 25
4.13 Matrices . . . 25
4.14 Arrays . . . 26
4.15 Selection . . . 26
4.16 Lists . . . 26
4.17 Named Lists . . . 27
4.18 Recap . . . 27

5 Control structures **29**
5.1 Recap . . . 29
5.2 If . . . 29
5.3 Else . . . 30
5.4 Code blocks . . . 30
5.5 Loops . . . 30
5.6 While . . . 31
5.7 For . . . 31
5.8 For . . . 31
5.9 Loops with conditional statements . . . 32
5.10 Summary . . . 32

6 Functions **33**
6.1 Summary . . . 33
6.2 Defining functions . . . 33
6.3 Defining functions . . . 33
6.4 Defining functions . . . 34
6.5 More parameters . . . 34
6.6 Functions and control structures . . . 34
6.7 Scope . . . 35
6.8 Example . . . 35

6.9 Summary 36

7 Data Frames **37**
7.1 Recap 37
7.2 Lists and named lists 37
7.3 Data Frames 38
7.4 Selection 38
7.5 Selection 38
7.6 Table manipulation 39
7.7 Column processing 39
7.8 tibble 39
7.9 Summary 40

8 Selection and filtering **41**
8.1 Recap 41
8.2 dplyr 41
8.3 Example dataset 42
8.4 Selecting table columns 42
8.5 dplyr::select 42
8.6 dplyr::select 43
8.7 Logical filtering 43
8.8 Conditional filtering 44
8.9 Filtering data frames 44
8.10 dplyr::filter 44
8.11 Select and filter 45
8.12 Summary 45

9 Data manipulation **47**
9.1 Recap 47
9.2 Example 47
9.3 dplyr::arrange 48
9.4 dplyr::summarise 48
9.5 dplyr::group_by 49
9.6 dplyr::tally and dplyr::count 49
9.7 dplyr::mutate 49
9.8 Full pipe example 50
9.9 Full pipe example 50
9.10 Summary 51

10 Join operations **53**
10.1 Recap 53
10.2 Example 53
10.3 Example 54
10.4 Joining data 54
10.5 Join types 54
10.6 dplyr joins 55

10.7 dplyr::full_join . . . 55
10.8 Pipes and shorthands . . . 55
10.9 dplyr::left_join . . . 56
10.10dplyr::right_join . . . 56
10.11dplyr::inner_join . . . 56
10.12dplyr::semi_join and anti_join . . . 57
10.13Summary . . . 57

11 Tidy data **59**
11.1 Recap . . . 59
11.2 Long data . . . 59
11.3 Wide data . . . 60
11.4 Example . . . 60
11.5 tidyr . . . 60
11.6 tidyr::pivot_wider . . . 61
11.7 tidyr::pivot_wider . . . 61
11.8 tidyr::pivot_longer . . . 62
11.9 tidyr::pivot_longer . . . 62
11.10tidyr::pivot_longer . . . 62
11.11tidyr . . . 63
11.12tidyr::replace_na . . . 63
11.13tidyr::fill . . . 63
11.14tidyr::drop_na . . . 64
11.15tidyr::complete . . . 64
11.16tidyr::complete . . . 65
11.17Summary . . . 65

12 Read and write data **67**
12.1 Summary . . . 67
12.2 Text file formats . . . 67
12.3 Comma Separated Values . . . 67
12.4 readr . . . 68
12.5 readr::read_csv . . . 68
12.6 Read options . . . 69
12.7 Column specifications . . . 69
12.8 readr::read_csv . . . 69
12.9 readr::read_csv . . . 70
12.10readr::read_csv . . . 70
12.11readr::write_csv . . . 70
12.12readr::write_tsv . . . 71
12.13Other data imports . . . 71
12.14Summary . . . 72

13 Reproducibility **73**
13.1 Recap . . . 73
13.2 Reproduciblity . . . 73

13.3 Why? . . . 74
13.4 Reproducibility and software engineering . . . 74
13.5 Reproducibility and "big data" . . . 74
13.6 Reproducibility in GIScience . . . 74
13.7 Document everything . . . 75
13.8 Document well . . . 75
13.9 Workflow . . . 75
13.10granolarr Mark.R . . . 76
13.11Future-proof formats . . . 76
13.12Store and share . . . 76
13.13Summary . . . 77

14 RMarkdown **79**
14.1 Recap . . . 79
14.2 Markdown . . . 79
14.3 Markdown example code . . . 79
14.4 Markdown example output . . . 80
14.5 RMarkdown . . . 80
14.6 RMarkdown example . . . 81
14.7 RMarkdown example . . . 81
14.8 The Definitive Guide . . . 82
14.9 Summary . . . 82

15 Git **83**
15.1 Recap . . . 83
15.2 What's git? . . . 83
15.3 How git works . . . 83
15.4 Three stages . . . 84
15.5 Basic git commands . . . 84
15.6 Git and RStudio . . . 85
15.7 What's Docker? . . . 86
15.8 Virtual machines . . . 87
15.9 Docker containers . . . 87
15.10Docker and reproducibility . . . 87
15.11granolarr Dockerfile . . . 88
15.12Summary . . . 88

16 Data visualisation **89**
16.1 Recap . . . 89
16.2 Grammar of graphics . . . 89
16.3 Visual variables . . . 90
16.4 ggplot2 . . . 90
16.5 Aesthetics . . . 90
16.6 Graphical primitives . . . 91
16.7 ggplot2::geom_line . . . 91
16.8 ggplot2::geom_line . . . 92

16.9 ggplot2::geom_col . . . 92
16.10ggplot2::geom_col . . . 93
16.11ggplot2::geom_col . . . 93
16.12ggplot2::geom_col . . . 94
16.13Histograms . . . 94
16.14Histograms . . . 95
16.15Boxplots . . . 95
16.16Boxplots . . . 96
16.17Jittered points . . . 96
16.18Jittered points . . . 97
16.19Violin plot . . . 97
16.20Violin plot . . . 98
16.21Scatterplots . . . 98
16.22Scatterplots . . . 99
16.23Overlapping points . . . 99
16.24Overlapping points . . . 100
16.25Bin counts . . . 100
16.26Bin counts . . . 101
16.27Coordinates transformations . . . 101
16.28Coordinates transformations . . . 102
16.29Summary . . . 102

17 Descriptive statistics **103**
17.1 Summary . . . 103
17.2 Meet the Palmer penguins . . . 103
17.3 Descriptive statistics . . . 104
17.4 stat.desc output . . . 104
17.5 stat.desc: basic . . . 104
17.6 stat.desc: basic . . . 105
17.7 stat.desc: desc . . . 105
17.8 Sample statistics . . . 106
17.9 Estimating variation . . . 106
17.10dplyr::across . . . 106
17.11dplyr::across . . . 107
17.12dplyr::across . . . 107
17.13Summary . . . 107

18 Exploring assumptions **109**
18.1 Recap . . . 109
18.2 Normal distribution . . . 109
18.3 Density histogram . . . 110
18.4 Q-Q plot . . . 110
18.5 Normality . . . 111
18.6 Significance . . . 111
18.7 Example . . . 111
18.8 Example . . . 112

18.9 Skewness and kurtosis . . . 113
18.10Example . . . 113
18.11Example . . . 113
18.12Homogeneity of variance . . . 114
18.13Summary . . . 114

19 Comparing groups **115**
19.1 Recap . . . 115
19.2 Iris . . . 116
19.3 Independent T-test . . . 116
19.4 Independent T-test . . . 116
19.5 Example: Petal lengths . . . 117
19.6 Assumptions: normality . . . 117
19.7 stats::t.test . . . 118
19.8 ANalysis Of VAriance . . . 118
19.9 ANalysis Of VAriance . . . 119
19.10Example: Petal lengths . . . 119
19.11Assumptions: normality . . . 119
19.12stats::aov . . . 120
19.13Summary . . . 120

20 Correlation **121**
20.1 Recap . . . 121
20.2 Correlation . . . 121
20.3 Correlation . . . 122
20.4 Example . . . 122
20.5 Pearson's r . . . 122
20.6 Assumptions: normality . . . 123
20.7 stats::cor.test . . . 123
20.8 Example . . . 123
20.9 Assumptions: normality . . . 124
20.10Spearman's rho . . . 124
20.11stats::cor.test (method = "spearman") . . . 125
20.12Correlation with ties . . . 125
20.13Kendall's tau . . . 126
20.14stats::cor.test (method = "kendall") . . . 126
20.15psych::pairs.panels . . . 127
20.16Chi-squre . . . 127
20.17gmodels::CrossTable . . . 127
20.18Summary . . . 128

21 Data transformations **129**
21.1 Recap . . . 129
21.2 Z-scores . . . 129
21.3 Example . . . 130
21.4 base::scale . . . 130

21.5 base::scale . . . 131
21.6 Comparison . . . 131
21.7 Log transformation . . . 132
21.8 Example . . . 133
21.9 Example . . . 133
21.10Inverse hyperbolic sine . . . 133
21.11Example . . . 134
21.12Example . . . 134
21.13Example . . . 135
21.14Summary . . . 135

22 Simple Regression **137**
22.1 Recap . . . 137
22.2 Regression analysis . . . 137
22.3 Example . . . 138
22.4 Example . . . 138
22.5 Least squares . . . 139
22.6 Assumptions . . . 139
22.7 stats::lm . . . 140
22.8 Overall fit . . . 140
22.9 Outliers and influential cases . . . 141
22.10Checking assumptions: normality . . . 141
22.11Checking assumpt.: homoscedasticity . . . 141
22.12Checking assumptions: independence . . . 142
22.13Example . . . 142
22.14Summary . . . 143

23 Multiple Regression **145**
23.1 Recap . . . 145
23.2 Multiple regression . . . 145
23.3 Assumptions . . . 146
23.4 Boston housing . . . 146
23.5 Example . . . 147
23.6 stats::lm . . . 147
23.7 Overall fit . . . 148
23.8 Standardised coefficients . . . 148
23.9 Confindence intervals . . . 148
23.10Outliers and influential cases . . . 149
23.11Checking assumptions: normality . . . 149
23.12Checking assumpt.: homoscedasticity . . . 150
23.13Checking assumptions: independence . . . 150
23.14Checking assumpt.: multicollinearity . . . 151
23.15Example . . . 151
23.16Summary . . . 151

24 Comparing regression models **153**

24.1 Recap . . . 153
24.2 Multiple regression . . . 153
24.3 Example . . . 154
24.4 stats::lm . . . 154
24.5 Checking assumptions . . . 155
24.6 Model 1 . . . 155
24.7 stats::lm . . . 155
24.8 Logarithmic transformations . . . 156
24.9 Checking assumptions . . . 157
24.10Model 2 . . . 157
24.11Comparing R-squared . . . 158
24.12Model difference with ANOVA . . . 158
24.13Information criteria . . . 158
24.14Stepwise selection . . . 159
24.15MASS::stepAIC . . . 159
24.16Model 3 . . . 160
24.17Checking assumptions . . . 160
24.18Validation . . . 161
24.19caret::train . . . 161
24.20Crossvalidate Model 3 . . . 161
24.21Summary . . . 162

25 Machine Learning **163**
25.1 Recap . . . 163
25.2 Definition . . . 163
25.3 Origines . . . 163
25.4 Types of machine learning . . . 164
25.5 Supervised . . . 164
25.6 Unsupervised . . . 165
25.7 Semi-supervised learning . . . 165
25.8 Reinforcement learning . . . 166
25.9 Limits . . . 166
25.10Overfitting . . . 166
25.11Algorithmic bias . . . 167
25.12Summary . . . 167

26 Artificial Neural Networks **169**
26.1 Recap . . . 169
26.2 Neural networks . . . 170
26.3 Artificial neurons . . . 170
26.4 Logistic regression . . . 170
26.5 Example . . . 171
26.6 Example . . . 171
26.7 Example . . . 172
26.8 stats::glm . . . 172
26.9 Logistic regression . . . 173

26.10Network topology . . . 174
26.11Defining a network . . . 174
26.12Deep neural networks . . . 175
26.13Convolutional neural networks . . . 175
26.14neuralnet::neuralnet . . . 175
26.15Performance . . . 177
26.16Summary . . . 177

27 Support vector machines **179**
27.1 Recap . . . 179
27.2 Classification task . . . 179
27.3 Support vector machines . . . 180
27.4 Nearest Neighbours (k-NN) . . . 180
27.5 Performance . . . 181
27.6 Hyperplanes . . . 182
27.7 e1071::svm . . . 182
27.8 Performance . . . 182
27.9 Not linearly separable . . . 184
27.10e1071::svm . . . 184
27.11Performance . . . 185
27.12e1071::svm . . . 185
27.13Performance . . . 186
27.14Summary . . . 187

28 Principal Component Analysis **189**
28.1 Recap . . . 189
28.2 Principal components . . . 189
28.3 Dimensionality reduction . . . 190
28.4 stats::prcomp . . . 190
28.5 PCA results . . . 191
28.6 Plotting PCA . . . 191
28.7 Summary . . . 192

29 Centroid-based clustering **193**
29.1 Recap . . . 193
29.2 Clustering task . . . 193
29.3 Example . . . 194
29.4 k-means algorithm . . . 194
29.5 stats::kmeans . . . 194
29.6 k-means result . . . 195
29.7 stats::kmeans . . . 195
29.8 k-means result . . . 196
29.9 Limitations . . . 196
29.10Fuzzy c-means . . . 197
29.11Fuzzy c-means . . . 197
29.12Fuzzy c-means result . . . 198

29.13Geodemographic classifications 198
29.14Summary 198

30 Hierarchical and density-based clustering 201
30.1 Recap 201
30.2 Example 202
30.3 Hierarchical clustering 202
30.4 stats::hclust 203
30.5 clustering tree 203
30.6 Hierarchical clustering result 204
30.7 Bagged clustering 205
30.8 e1071::bclust 205
30.9 Bagged clustering result 206
30.10Density based clustering 206
30.11dbscan::dbscan 206
30.12DBSCAN result 207
30.13DBSCAN result 207
30.14DBSCAN result 208
30.15Not alwasy that easy… 209
30.16Summary 209

Preface

Stefano De Sabbata

This book contains the *lectures* component of granolarr, a repository of reproducible materials to teach geographic information and data science in R. Part of the materials are derived from the lectures for the module GY7702 Practical Programming in R of the MSc in Geographic Information Science at the School of Geography, Geology, and the Environment of the University of Leicester, by Dr Stefano De Sabbata.

This book was created using R, RStudio, RMarkdown, Bookdown, and GitHub.

Session info

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04 LTS
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-openmp/libopenblasp-r0.3.8.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
```

```
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.0.2  magrittr_1.5    bookdown_0.20   htmltools_0.5.0
##  [5] tools_4.0.2     yaml_2.2.1      stringi_1.4.6   rmarkdown_2.3
##  [9] knitr_1.29      stringr_1.4.0   digest_0.6.25   xfun_0.16
## [13] rlang_0.4.7     evaluate_0.14
```

Chapter 1

Introduction to R

1.1 About this module

This module will provide you with the fundamental skills in

- basic programming in R
- data wrangling
- data analysis
- reproducibility

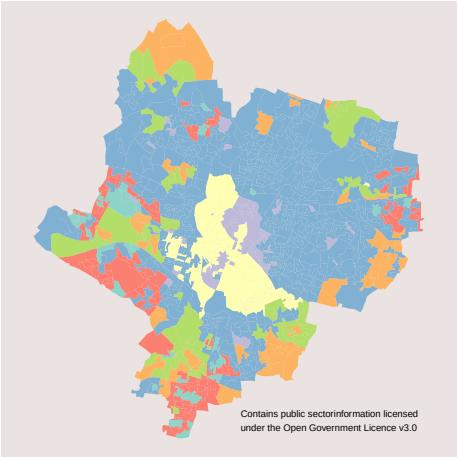
basis for

- *Geospatial Data Analysis*
- *Geospatial Databases and Information Retrieval*

1.2 R programming language

One of the most widely used programming languages and an effective tool for *(geospatial)* data science

- data wrangling
- statistical analysis
- machine learning
- data visualisation and maps
- processing spatial data
- geographic information analysis



1.3 Schedule

The lectures and practical sessions have been designed to follow the schedule below

- **1 R coding**
  - 100 Introduction
  - 110 R programming
- **2 Data wrangling**
  - 200 Selection and manipulation
  - 210 Table operations
  - 220 Reproducibility
- **3 Data analysis**
  - 300 Exploratory data analysis
  - 310 Comparing data
  - 320 Regression models
- **4 Machine learning**
  - 400 Unsupervised
  - 410 Supervised

1.4 Reference books

Suggested reading

- *Programming Skills for Data Science: Start Writing Code to Wrangle, Analyze, and Visualize Data with R* by Michael Freeman and Joel Ross, Addison-Wesley, 2019. See book webpage and repository.
- *R for Data Science* by Garrett Grolemund and Hadley Wickham, O'Reilly Media, 2016. See online book.
- *Discovering Statistics Using R* by Andy Field, Jeremy Miles and Zoë Field, SAGE Publications Ltd, 2012. See book webpage.
- *Machine Learning with R: Expert techniques for predictive modeling* by Brett Lantz, Packt Publishing, 2019. See book webpage.

Further reading

- *The Art of R Programming: A Tour of Statistical Software Design* by Norman Matloff, No Starch Press, 2011. See book webpage

- *An Introduction to R for Spatial Analysis and Mapping* by Chris Brunsdon and Lex Comber, Sage, 2015. See book webpage
- *Geocomputation with R* by Robin Lovelace, Jakub Nowosad, Jannes Muenchow, CRC Press, 2019. See online book.

1.5 R

Created in 1992 by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand

- Free, open-source implementation of *S*
  - statistical programming language
  - Bell Labs
- Functional programming language
- Supports (and commonly used as) procedural (i.e., imperative) programming
- Object-oriented
- Interpreted (not compiled)

1.6 Interpreting values

When values and operations are inputted in the *Console*, the interpreter returns the results of its interpretation of the expression

```
2
```

```
## [1] 2
```

```
"String value"
```

```
## [1] "String value"
```

```
# comments are ignored
```

1.7 Basic types

R provides three core data types

- numeric
  - both integer and real numbers
- character
  - i.e., text, also called *strings*
- logical
  - `TRUE` or `FALSE`

1.8 Numeric operators

R provides a series of basic numeric operators

Operator	Meaning	Example	Output
+	Plus	`5 + 2`	7
-	Minus	`5 - 2`	3
*	Product	`5 * 2`	10
/	Division	`5 / 2`	2.5
%/%	Integer division	`5 %/% 2`	2
%%	Module	`5 %% 2`	1
^	Power	`5^2`	25

```
5 + 2
```

```
## [1] 7
```

1.9 Logical operators

R provides a series of basic logical operators to test

Operator	Meaning	Example	Output
==	Equal	`5 == 2`	FALSE
!=	Not equal	`5 != 2`	TRUE
> (>=)	Greater (or equal)	`5 > 2`	TRUE
< (<=)	Less (or equal)	`5 <= 2`	FALSE
!	Not	`!TRUE`	FALSE
&	And	`TRUE & FALSE`	FALSE
|	Or	`TRUE | FALSE`	TRUE

```
5 >= 2
```

```
## [1] TRUE
```

1.10 Summary

An introduction to R

- Basic types
- Basic operators

Next: Core concepts

- Variables

- Functions
- Libraries

Chapter 2

Core concepts

2.1 Recap

Prev: An introduction to R

- Basic types
- Basic operators

Now: Core concepts

- Variables
- Functions
- Libraries

2.2 Variables

Variables **store data** and can be defined

- using an *identifier* (e.g., `a_variable`)
- on the left of an *assignment operator* `<-`
- followed by the object to be linked to the identifier
- such as a *value* (e.g., `1`)

```
a_variable <- 1
```

The value of the variable can be invoked by simply specifying the **identifier**.

```
a_variable
```

```
## [1] 1
```

2.3 Algorithms and functions

An **algorithm** *or effective procedure is a mechanical rule, or automatic method, or programme for performing some mathematical operation* (Cutland, 1980).

A **program** is a specific set of instructions that implement an abstract algorithm.

The definition of an algorithm (and thus a program) can consist of one or more **function**s

- set of instructions that preform a task
- possibly using an input, possibly returning an output value

Programming languages usually provide pre-defined functions that implement common algorithms (e.g., to find the square root of a number or to calculate a linear regression)

2.4 Functions

Functions execute complex operations and can be invoked

- specifying the *function name*
- the *arguments* (input values) between simple brackets
  - each *argument* corresponds to a *parameter*
  - sometimes the *parameter* name must be specified

```
sqrt(2)
```

```
## [1] 1.414214
```

```
round(1.414214, digits = 2)
```

```
## [1] 1.41
```

2.5 Functions and variables

- functions can be used on the right side of `<-`
- variables and functions can be used as *arguments*

```
sqrt_of_two <- sqrt(2)
sqrt_of_two
```

```
## [1] 1.414214
```

```
round(sqrt_of_two, digits = 2)
```

```
## [1] 1.41
```

```
round(sqrt(2), digits = 2)
```

```
## [1] 1.41
```

2.6 Naming

When creating an identifier for a variable or function

- R is a **case sensitive** language
  - UPPER and lower case are not the same
  - `a_variable` is different from `a_VARIABLE`
- names can include
  - alphanumeric symbols
  - `.` and `_`
- names must start with
  - a letter

2.7 Libraries

Once a number of related, reusable functions are created

- they can be collected and stored in **libraries** (a.k.a. *packages*)
  - `install.packages` is a function that can be used to install libraries (i.e., downloads it on your computer)
  - `library` is a function that *loads* a library (i.e., makes it available to a script)

Libraries can be of any size and complexity, e.g.:

- `base`: base R functions, including the `sqrt` function above
- `rgdal`: implementation of the GDAL (Geospatial Data Abstraction Library) functionalities

2.8 stringr

R provides some basic functions to manipulate strings, but the `stringr` library provides a more consistent and well-defined set

```
library(stringr)

str_length("Leicester")
```

```
## [1] 9
```

```
str_detect("Leicester", "e")
```

```
## [1] TRUE
```

```
str_replace_all("Leicester", "e", "x")
```

```
## [1] "Lxicxstxr"
```

2.9 Summary

Core concepts

- Variables
- Functions
- Libraries

Next: Tidyverse

- Tidyverse libraries
- *pipe* operator

Chapter 3

Tidyverse

3.1 Recap

Prev: Core concepts

- Variables
- Functions
- Libraries

Now: Tidyverse

- Tidyverse libraries
- *pipe* operator

3.2 Tidyverse

The Tidyverse was introduced by statistician Hadley Wickham, Chief Scientist at RStudio (worth following him on twitter).

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures." (Tidyverse homepage).

Core libraries

- `tibble`
- `tidyr`
- `stringr`
- `dplyr`
- `readr`
- `ggplot2`
- `purrr`
- `forcats`

Also, imports `magrittr`, which plays an important role.

3.3 Tidyverse core libraries

The meta-library Tidyverse includes:

- `tibble` is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles are data.frames that are lazy and surly: they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code.
- `tidyr` provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable.
- `stringr` provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations.

3.4 Tidyverse core libraries

The meta-library Tidyverse includes:

- `dplyr` provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges.
- `readr` provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.
- `ggplot2` is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

3.5 Tidyverse core libraries

The meta-library Tidyverse contains the following libraries:

- `purrr` enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. Once you master the basic concepts, purrr allows you to replace many for loops with code that is easier to write and more expressive.
- `forcats` provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values.

3.6 The pipe operator

The Tidyverse (via `magrittr`) also provide a clean and effective way of combining multiple manipulation steps

The pipe operator `%>%`

- takes the result from one function
- and passes it to the next function
- as the **first argument**
- that doesn't need to be included in the code anymore

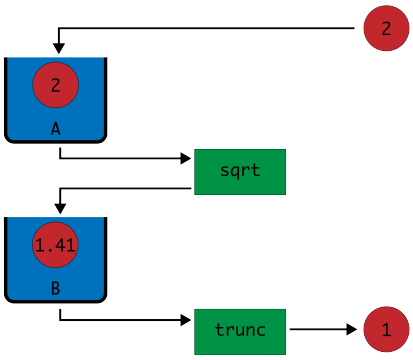
3.7 Pipe example

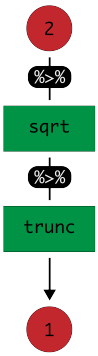
```
tmp_variable_A <- 2
tmp_variable_B <- sqrt(tmp_variable_A)
trunc(tmp_variable_B)

## [1] 1
```

```
2 %>%
  sqrt() %>%
  trunc()

## [1] 1
```





3.8 Pipe example

The two codes below are equivalent

- the first simply invokes the functions
- the second uses the pipe operator `%>%`

```
round(sqrt(2), digits = 2)
```

```
## [1] 1.41
```

```
library(tidyverse)

sqrt(2) %>%
 round(digits = 2)
```

```
## [1] 1.41
```

3.9 Coding style

A *coding style* is a way of writing the code, including

- how variable and functions are named
  - lower case and _
- how spaces are used in the code
- which libraries are used

```
# Bad
X<-round(sqrt(2),2)

#Good
sqrt_of_two <- sqrt(2) %>%
 round(digits = 2)
```

Study the Tidyverse Style Guid and use it consistently!

3.10 Summary

Tidyverse

- Tidyverse libraries
- *pipe* operator
- Coding style

Next: Practical session

- The R programming language
- Interpreting values
- Variables
- Basic types
- Tidyverse
- Coding style

Chapter 4

Data types

4.1 Recap

Prev: Introduction

- 101 Lecture: Introduction to R
- 102 Lecture: Core concepts
- 103 Lecture: Tidyverse
- 104 Practical session

Now: Data types

- vectors
- factors
- matrices, arrays
- lists

4.2 Vectors

Vectors are ordered list of values.

- Vectors can be of any data type
  - numeric
  - character
  - logic
- All items in a vector have to be of the same type
- Vectors can be of any length

4.3 Defining vectors

A vector variable can be defined using

- an **identifier** (e.g., `a_vector`)
- on the left of an **assignment operator** `<-`
- followed by the object to be linked to the identifier
- in this case, the result returned by the function `c`
- which creates a vector containing the provided elements

```
a_vector <- c("Birmingham", "Derby", "Leicester",
  "Lincoln", "Nottingham", "Wolverhampton")
a_vector
```

```
## [1] "Birmingham"    "Derby"         "Leicester"     "Lincoln"
## [5] "Nottingham"    "Wolverhampton"
```

4.4 Creating vectors

- the operator `:`
- the function `seq`
- the function `rep`

```
4:7
```

```
## [1] 4 5 6 7
```

```
seq(1, 7, by = 0.5)
```

```
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
```

```
seq(1, 10, length.out = 7)
```

```
## [1]  1.0  2.5  4.0  5.5  7.0  8.5 10.0
```

```
rep("Ciao", 4)
```

```
## [1] "Ciao" "Ciao" "Ciao" "Ciao"
```

4.5 Selection

Each element of a vector can be retrieved specifying the related **index** between square brackets, after the identifier of the vector. The first element of the vector has index 1.

```
a_vector[3]
```

```
## [1] "Leicester"
```

A vector of indexes can be used to retrieve more than one element.

```
a_vector[c(5, 3)]
```

```
## [1] "Nottingham" "Leicester"
```

4.6 Functions on vectors

Functions can be used on a vector variable directly

```
a_numeric_vector <- 1:5
a_numeric_vector + 10
```

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

```
sqrt(a_numeric_vector)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
a_numeric_vector >= 3
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

4.7 Any and all

Overall expressions can be tested using the functions:

- **any**, TRUE if any of the elements satisfies the condition
- **all**, TRUE if all of the elements satisfy the condition

```
any(a_numeric_vector >= 3)
```

```
## [1] TRUE
```

```
all(a_numeric_vector >= 3)
```

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

4.8 Factors

A **factor** is a data type similar to a vector. However, the values contained in a factor can only be selected from a set of **levels**.

```
houses_vector <- c("Bungalow", "Flat", "Flat",
  "Detached", "Flat", "Terrace", "Terrace")
houses_vector
```

```
## [1] "Bungalow" "Flat"     "Flat"     "Detached" "Flat"     "Terrace"  "Terrace"
```

```
houses_factor <- factor(c("Bungalow", "Flat", "Flat",
  "Detached", "Flat", "Terrace", "Terrace"))
houses_factor
```

```
## [1] Bungalow Flat     Flat     Detached Flat     Terrace  Terrace
## Levels: Bungalow Detached Flat Terrace
```

4.9 table

The function **table** can be used to obtain a tabulated count for each level.

```
houses_factor <- factor(c("Bungalow", "Flat", "Flat",
  "Detached", "Flat", "Terrace", "Terrace"))
houses_factor
```

```
## [1] Bungalow Flat     Flat     Detached Flat     Terrace  Terrace
## Levels: Bungalow Detached Flat Terrace
```

```
table(houses_factor)
```

```
## houses_factor
## Bungalow Detached     Flat  Terrace
##        1        1        3        2
```

4.10 Specified levels

A specific set of levels can be specified when creating a factor by providing a **levels** argument.

```
houses_factor_spec <- factor(
  c("People Carrier", "Flat", "Flat", "Hatchback",
      "Flat", "Terrace", "Terrace"),
  levels = c("Bungalow", "Flat", "Detached",
       "Semi", "Terrace"))

table(houses_factor_spec)
```

```
## houses_factor_spec
## Bungalow     Flat Detached     Semi  Terrace
##        0        3        0        0        2
```

4.11 (Unordered) Factors

In statistics terminology, (unordered) factors are **categorical** (i.e., binary or nominal) variables. Levels are not ordered.

```
income_nominal <- factor(
  c("High", "High", "Low", "Low", "Low",
      "Medium", "Low", "Medium"),
  levels = c("Low", "Medium", "High"))
```

The *greater than* operator is not meaningful on the `income_nominal` factor defined above

```
income_nominal > "Low"
```

```
## Warning in Ops.factor(income_nominal, "Low"): '>' not meaningful for factors
```

```
## [1] NA NA NA NA NA NA NA NA
```

4.12 Ordered Factors

In statistics terminology, ordered factors are **ordinal** variables. Levels are ordered.

```
income_ordered <- ordered(
  c("High", "High", "Low", "Low", "Low",
      "Medium", "Low", "Medium"),
  levels = c("Low", "Medium", "High"))

income_ordered > "Low"
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE
```

```
sort(income_ordered)
```

```
## [1] Low    Low    Low    Low    Medium Medium High   High
## Levels: Low < Medium < High
```

4.13 Matrices

Matrices are collections of numerics arranged in a two-dimensional rectangular layout

- the first argument is a vector of values
- the second specifies number of rows and columns
- R offers operators and functions for matrix algebra

```
a_matrix <- matrix(c(3, 5, 7, 4, 3, 1), c(3, 2))
a_matrix
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    5    3
## [3,]    7    1
```

4.14 Arrays

Variables of the type **array**are higher-dimensionalmatrices.

- the first argument is avector containing thevalues
- the second argument isavector specifying thedepth of each dimension

```
a3dim_array <- array(1:24, dim=c(4, 3, 2))
```

```
a3dim_array
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

4.15 Selection

Subsets of matrices (and arrays) can be selected as seen for vectors.

```
a_matrix[2, c(1, 2)]
```

```
## [1] 5 3
```

```
a3dim_array[c(1, 2), 2, 2]
```

```
## [1] 17 18
```

4.16 Lists

Variables of the type **list** can contain elements of different types (including vectors and matrices), whereas elements of vectors are all of the same type.

```
employee <- list("Stef", 2015)
employee
```

```
## [[1]]
## [1] "Stef"
##
## [[2]]
## [1] 2015
```

```
employee[[1]] # Note the double square brackets for selection
```

```
## [1] "Stef"
```

4.17 Named Lists

In **named lists** each element has a name, and elements can be selected using their name after the symbol `$`.

```
employee <- list(employee_name = "Stef", start_year = 2015)
employee
```

```
## $employee_name
## [1] "Stef"
##
## $start_year
## [1] 2015
```

```
employee$employee_name
```

```
## [1] "Stef"
```

4.18 Recap

Data types

- Vectors
- Factors
- Matrices, arrays
- Lists

Next: Control structures

- Conditional statements
- Loops

Chapter 5

Control structures

5.1 Recap

Prev: Data types

- Vectors
- Factors
- Matrices and arrays
- Lists

Now: Control structures

- Conditional statements
- Loops

5.2 If

Format: **if** (*condition*) *statement*

- *condition*: expression returning a logic value (`TRUE` or `FALSE`)
- *statement*: any valid R statement
- *statement* only executed if *condition* is `TRUE`

```
a_value <- -7
if (a_value < 0) cat("Negative")
```

```
## Negative
```

```
a_value <- 8
if (a_value < 0) cat("Negative")
```

5.3 Else

Format: **if** (*condition*) *statement1* **else** *statement2*

- *condition*: expression returning a logic value (`TRUE` or `FALSE`)
- *statement1* and *statement2*: any valid R statements
- *statement1* executed if *condition* is `TRUE`
- *statement2* executed if *condition* is `FALSE`

```
a_value <- -7
if (a_value < 0) cat("Negative") else cat("Positive")
```

```
## Negative
```

```
a_value <- 8
if (a_value < 0) cat("Negative") else cat("Positive")
```

```
## Positive
```

5.4 Code blocks

Code blocks allow to encapsulate **several** statements in a single group

- `{` and `}` contain code blocks
- the statements are execute together

```
first_value <- 8
second_value <- 5
if (first_value > second_value) {
  cat("First is greater than second\n")
  difference <- first_value - second_value
  cat("Their difference is ", difference)
}
```

```
## First is greater than second
## Their difference is  3
```

5.5 Loops

Loops are a fundamental component of (procedural) programming.

There are two main types of loops:

- **conditional** loops are executed as long as a defined condition holds true
  - construct `while`
  - construct `repeat`
- **deterministic** loops are executed a pre-determined number of times
  - construct `for`

5.6 While

The *while* construct can be defined using the `while` reserved word, followed by the conditional statement between simple brackets, and a code block. The instructions in the code block are re-executed as long as the result of the evaluation of the conditional statement is `TRUE`.

```
current_value <- 0

while (current_value < 3) {
  cat("Current value is", current_value, "\n")
  current_value <- current_value + 1
}
```

```
## Current value is 0
## Current value is 1
## Current value is 2
```

5.7 For

The *for* construct can be defined using the `for` reserved word, followed by the definition of an **iterator**. The iterator is a variable which is temporarily assigned with the current element of a vector, as the construct iterates through all elements of the vector. This definition is followed by a code block, whose instructions are re-executed once for each element of the vector.

```
cities <- c("Derby", "Leicester", "Lincoln", "Nottingham")
for (city in cities) {
  cat("Do you live in", city, "?\n")
}
```

```
## Do you live in Derby ?
## Do you live in Leicester ?
## Do you live in Lincoln ?
## Do you live in Nottingham ?
```

5.8 For

It is common practice to create a vector of integers on the spot in order to execute a certain sequence of steps a pre-defined number of times.

```
for (i in 1:3) {
  cat("This is exectuion number", i, ":\n")
  cat("    See you later!\n")
}
```

```
## This is exectuion number 1 :
```

```
##     See you later!
## This is exectuion number 2 :
##     See you later!
## This is exectuion number 3 :
##     See you later!
```

5.9 Loops with conditional statements

```
3:0
```

```
## [1] 3 2 1 0
```

```
#Example: countdown!
for (i in 3:0) {
  if (i == 0) {
    cat("Go!\n")
  } else {
    cat(i, "\n")
  }
}
```

```
## 3
## 2
## 1
## Go!
```

5.10 Summary

Control structures

- Conditional statements
- Loops

Next: Functions

- Defining functions
- Scope of a variable

Chapter 6

Functions

6.1 Summary

Prev:Control structures

- Conditional statements
- Loops

Now: Functions

- Defining functions
- Scope of a variable

6.2 Defining functions

A function can be defined

- using an **identifier** (e.g., `add_one`)
- on the left of an **assignment operator** `<-`
- followed by the corpus of the function

```
add_one <- function (input_value) {
  output_value <- input_value + 1
  output_value
  }
```

6.3 Defining functions

The corpus

- starts with the reserved word `function`

- followed by the **parameter(s)** (e.g., `input_value`) between simple brackets
- and the instruction(s) to be executed in a code block
- the value of the last statement is returned as output

```
add_one <- function (input_value) {
  output_value <- input_value + 1
  output_value
  }
```

6.4 Defining functions

After being defined

- a function can be invoked by specifying
  - the **identifier**
  - the necessary **parameter(s)**

```
add_one(3)
```

```
## [1] 4
```

```
add_one(1024)
```

```
## [1] 1025
```

6.5 More parameters

- A function can be defined as having two or more **parameters**
  - by specifying more than one parameter name (separated by **commas**) in the function definition
- A function always take as input as many values as the number of parameters specified in the definition
  - otherwise an error is generated

```
area_rectangle <- function (hight, width) {
  area <- hight * width
  area
}

area_rectangle(3, 2)
```

```
## [1] 6
```

6.6 Functions and control structures

Functions can contain both loops and conditional statements

```r
factorial <- function (input_value) {
  result <- 1
  for (i in 1:input_value) {
    cat("current:", result, " | i:", i, "\n")
    result <- result * i
  }
  result
}
factorial(3)
```

```
## current: 1  | i: 1
## current: 1  | i: 2
## current: 2  | i: 3
```

```
## [1] 6
```

6.7 Scope

The **scope of a variable** is the part of code in which the variable is "visible"

In R, variables have a **hierarchical** scope:

- a variable defined in a script can be used referred to from within a definition of a function in the same script
- a variable defined within a definition of a function will **not** be referable from outside the definition
- scope does **not** apply to `if` or loop constructs

6.8 Example

In the case below

- `x_value` is **global** to the function `times_x`
- `new_value` and `input_value` are **local** to the function `times_x`
  - referring to `new_value` or `input_value` from outside the definition of `times_x` would result in an error

```r
x_value <- 10
times_x <- function (input_value) {
  new_value <- input_value * x_value
  new_value
}
times_x(2)
```

```
## [1] 20
```

6.9 Summary

Functions

- Defining functions
- Scope of a variable

Next: Practical session

- Conditional statements
- Loops
  - While
  - For
- Functions
  - Loading functions from scripts
- Debugging

Chapter 7

Data Frames

7.1 Recap

Prev: R programming

- 111 Lecture: Data types
- 112 Lecture: Control structures
- 113 Lecture: Functions
- 114 Practical session

Now: Data Frames

- Data Frames
- Tibbles

7.2 Lists and named lists

List

- can contain elements of different types
  - whereas elements of vectors are all of the same type
- in **named lists**, each element has a name
  - elements can be selected using the operator `$`

```
employee <- list(employee_name = "Stef", start_year = 2015)
employee[[1]]
```

```
## [1] "Stef"
```

```
employee$employee_name
```

```
## [1] "Stef"
```

7.3 Data Frames

A **data frame** is equivalent to a *named list* where all elements are *vectors of the same length*.

```
employees <- data.frame(
  EmployeeName = c("Maria", "Pete", "Sarah"),
  Age = c(47, 34, 32),
  Role = c("Professor", "Researcher", "Researcher"))
employees
```

```
##   EmployeeName Age       Role
## 1        Maria  47  Professor
## 2         Pete  34 Researcher
## 3        Sarah  32 Researcher
```

Data frames are the most common way to represent tabular data in R. Matrices and lists can be converted to data frames.

7.4 Selection

Selection is similar to vectors and lists.

```
employees[1, 1] # value selection
```

```
## [1] "Maria"
```

```
employees[1, ] # row selection
```

```
##   EmployeeName Age      Role
## 1        Maria  47 Professor
```

```
employees[, 1] # column selection
```

```
## [1] "Maria" "Pete"  "Sarah"
```

7.5 Selection

Selection is similar to vectors and lists.

```
employees$EmployeeName # column selection, as for named lists
```

```
## [1] "Maria" "Pete"  "Sarah"
```

```
employees$EmployeeName[1]
```

```
## [1] "Maria"
```

7.6 Table manipulation

- Values can be assigned to cells
  - using any selection method
  - and the assignment operator `<-`
- New columns can be defined
  - assigning a vector to a new name

```
employees$Age[3] <- 33
employees$Place <- c("Leicester", "Leicester","Leicester")
employees
```

```
##   EmployeeName Age       Role     Place
## 1        Maria  47  Professor Leicester
## 2         Pete  34 Researcher Leicester
## 3        Sarah  33 Researcher Leicester
```

7.7 Column processing

Operations can be performed on columns as they where vectors

```
10 - c(1, 2, 3)
```

```
## [1] 9 8 7
```

```
# Use Sys.Date to retrieve the current year
current_year <- as.integer(format(Sys.Date(), "%Y"))

# Calculate employee year of birth
employees$Year_of_birth <- current_year - employees$Age
employees
```

```
##   EmployeeName Age       Role     Place Year_of_birth
## 1        Maria  47  Professor Leicester          1974
## 2         Pete  34 Researcher Leicester          1987
## 3        Sarah  33 Researcher Leicester          1988
```

7.8 tibble

A tibble is a modern reimagining of the data.frame within `tidyverse`

- they do less
  - don't change column names or types
  - don't do partial matching
- complain more
  - e.g. when referring to a column that does not exist

That forces you to confront problems earlier, typically leading to cleaner, more expressive code.

7.9 Summary

Data Frames

- Data Frames
- Tibbles

Next: Data selection and filtering

- dplyr
- dplyr::select
- dplyr::filter

Chapter 8

Selection and filtering

8.1 Recap

Prev: Data Frames

- Data Frames
- Tibbles

Now: Data selection and filtering

- dplyr
- dplyr::select
- dplyr::filter

8.2 dplyr

The `dplyr` (pronounced *dee-ply-er*) library is part of `tidyverse` and it offers a grammar for data manipulation

- `select`: select specific columns
- `filter`: select specific rows
- `arrange`: arrange rows in a particular order
- `summarise`: calculate aggregated values (e.g., mean, max, etc)
- `group_by`: group data based on common column values
- `mutate`: add columns
- `join`: merge tables (`tibbles` or `data.frames`)

```
library(tidyverse)
```

8.3 Example dataset

The library `nycflights13` contains a dataset storing data about all the flights departed from New York City in 2013

```
library(nycflights13)

nycflights13::flights
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## # ... with 336,773 more rows, and 12 more variables:
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

8.4 Selecting table columns

Columns of **data frames** and **tibbles** can be selected

- specifying the column index

```
nycflights13::flights[, c(13, 14)]
```

- specifying the column name

```
nycflights13::flights[, c("origin", "dest")]
```

```
## # A tibble: 336,776 x 2
##   origin dest
##   <chr>  <chr>
## 1 EWR    IAH
## 2 LGA    IAH
## 3 JFK    MIA
## # ... with 336,773 more rows
```

8.5 dplyr::select

`select` can be used to specify which columns to retain

```
nycflights13::flights %>%
  dplyr::select(
```

```
    origin, dest, dep_delay, arr_delay, year:day
  )
```

```
## # A tibble: 336,776 x 7
##   origin dest  dep_delay arr_delay  year month   day
##   <chr>  <chr>     <dbl>     <dbl> <int> <int> <int>
## 1 EWR    IAH           2        11  2013     1     1
## 2 LGA    IAH           4        20  2013     1     1
## 3 JFK    MIA           2        33  2013     1     1
## 4 JFK    BQN          -1       -18  2013     1     1
## 5 LGA    ATL          -6       -25  2013     1     1
## # ... with 336,771 more rows
```

8.6 dplyr::select

... or whichones to drop, using - in front of the column name

```
nycflights13::flights %>%
  dplyr::select(origin, dest, dep_delay, arr_delay, year:day) %>%
  dplyr::select(-arr_delay)
```

```
## # A tibble: 336,776 x 6
##   origin dest  dep_delay  year month   day
##   <chr>  <chr>     <dbl> <int> <int> <int>
## 1 EWR    IAH           2  2013     1     1
## 2 LGA    IAH           4  2013     1     1
## 3 JFK    MIA           2  2013     1     1
## # ... with 336,773 more rows
```

8.7 Logical filtering

Conditional statements can be used to filter a vector

- i.e. to retain only certain values
- where the specified value is `TRUE`

```
a_numeric_vector <- -3:3
a_numeric_vector
```

```
## [1] -3 -2 -1  0  1  2  3
```

```
a_numeric_vector[c(FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)]
```

```
## [1] 0 1 2 3
```

8.8 Conditional filtering

As a conditional expression results in a logic vector...

```
a_numeric_vector > 0
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

... conditional expressions can be used for filtering

```
a_numeric_vector[a_numeric_vector > 0]
```

```
## [1] 1 2 3
```

8.9 Filtering data frames

The same approach can be applied to **data frames** and **tibbles**

```
nycflights13::flights$month
```

```
##     [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1...
```

```
nycflights13::flights$month == 11
```

```
##     [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE...
```

```
nycflights13::flights[nycflights13::flights$month == 11, ]
```

```
## # A tibble: 27,268 x 19
##    year month   day dep_time sched_dep_time
##   <int> <int> <int>    <int>          <int>
## 1  2013    11     1        5           2359
## # ... with 27,267 more rows, and 14 more variables:
## #   dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

8.10 dplyr::filter

```
nycflights13::flights %>%
  # Flights in November
  dplyr::filter(month == 11)
```

```
## # A tibble: 27,268 x 19
##    year month   day dep_time sched_dep_time
##   <int> <int> <int>    <int>          <int>
```

```
## 1  2013    11     1        5           2359
## 2  2013    11     1       35           2250
## 3  2013    11     1      455            500
## # ... with 27,265 more rows, and 14 more variables:
## #   dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

8.11 Select and filter

```
nycflights13::flights %>%
  # Select the columns you need
  dplyr::select(origin, dest, dep_delay, arr_delay, year:day) %>%
  # Drop arr_delay... because you don't need it after all
  dplyr::select(-arr_delay) %>%
  # Filter in only November flights
  dplyr::filter(month == 11)
```

```
## # A tibble: 27,268 x 6
##   origin dest  dep_delay  year month   day
##   <chr>  <chr>     <dbl> <int> <int> <int>
## 1 JFK    PSE           6  2013    11     1
## 2 JFK    SYR         105  2013    11     1
## 3 EWR    CLT          -5  2013    11     1
## # ... with 27,265 more rows
```

8.12 Summary

Data selection and filtering

- dplyr
- dplyr::select
- dplyr::filter

Next: Data manipulation

- dplyr::arrange
- dplyr::summarise
- dplyr::group_by
- dplyr::mutate

Chapter 9

Data manipulation

9.1 Recap

Prev: Data selection and filtering

- dplyr
- dplyr::select
- dplyr::filter

Now: Data manipulation

- dplyr::arrange
- dplyr::summarise
- dplyr::group_by
- dplyr::mutate

9.2 Example

```
library(tidyverse)
library(nycflights13)

nov_dep_delays <-
  nycflights13::flights %>%
  dplyr::select(origin, dest, dep_delay, year:day) %>%
  dplyr::filter(month == 11)

nov_dep_delays
```

```
## # A tibble: 27,268 x 6
##   origin dest  dep_delay  year month   day
##   <chr>  <chr>     <dbl> <int> <int> <int>
```

```
## 1 JFK    PSE           6  2013    11     1
## 2 JFK    SYR         105  2013    11     1
## 3 EWR    CLT          -5  2013    11     1
## # ... with 27,265 more rows
```

9.3 dplyr::arrange

Arranges rows in a particular order

- descending orders specified by using - (minus symbol)

```
nov_dep_delays %>%
  dplyr::arrange(
    # Ascending destination name
    dest,
    # Descending delay
    -dep_delay
  )
```

```
## # A tibble: 27,268 x 6
##   origin dest  dep_delay  year month   day
##   <chr>  <chr>     <dbl> <int> <int> <int>
## 1 JFK    ABQ          25  2013    11    29
## 2 JFK    ABQ          21  2013    11    22
## # ... with 27,266 more rows
```

9.4 dplyr::summarise

Calculates aggregated values

- e.g., using functions such as mean, max, etc.

```
nov_dep_delays %>%
  # Need to filter out rows where delay is NA
  dplyr::filter(!is.na(dep_delay)) %>%
  # Create two aggregated columns
  dplyr::summarise(
    avg_dep_delay = mean(dep_delay),
    tot_dep_delay = sum(dep_delay)
  )
```

```
## # A tibble: 1 x 2
##   avg_dep_delay tot_dep_delay
##           <dbl>         <dbl>
## 1          5.44        146945
```

9.5 dplyr::group_by

Groups rows based on common values for specified column(s)

- combined with `summarise`, aggregated values per group

```
nov_dep_delays %>%
  # First group by same destination
  dplyr::group_by(dest) %>%
  # Then calculate aggregated value
  dplyr::filter(!is.na(dep_delay)) %>%
  dplyr::summarise(tot_dep_delay = sum(dep_delay))
```

```
## # A tibble: 90 x 2
##   dest  tot_dep_delay
##   <chr>         <dbl>
## 1 ABQ             -66
## 2 ALB             636
## # ... with 88 more rows
```

9.6 dplyr::tally and dplyr::count

- `dplyr::tally` short-hand for `summarise` with `n`
  - number of rows
- `dplyr::count`short-hand for `group_by` and `tally`
  - number of rows per group

```
nov_dep_delays %>%
  # Count flights by same destination
  dplyr::count(dest)
```

```
## # A tibble: 90 x 2
##   dest      n
##   <chr> <int>
## 1 ABQ      30
## 2 ALB      46
## 3 ATL    1384
## # ... with 87 more rows
```

9.7 dplyr::mutate

Calculate values for new columns based on current columns

```
nov_dep_delays %>%
  dplyr::mutate(
    # Combine origin and destination into one column
    orig_dest = str_c(origin, dest, sep = "->"),
```

```
    # Departure delay in days (rather than minutes)
    delay_days = ((dep_delay / 60) /24)
  )
```

```
## # A tibble: 27,268 x 8
##   origin dest  dep_delay  year month   day orig_dest delay_days
##   <chr>  <chr>     <dbl> <int> <int> <int> <chr>          <dbl>
## 1 JFK    PSE           6  2013    11     1 JFK->PSE     0.00417
## 2 JFK    SYR         105  2013    11     1 JFK->SYR     0.0729
## 3 EWR    CLT          -5  2013    11     1 EWR->CLT    -0.00347
## # ... with 27,265 more rows
```

9.8 Full pipe example

```
nycflights13::flights %>%
  dplyr::select(
    origin, dest, dep_delay, arr_delay,
    year:day
  ) %>%
  dplyr::select(-arr_delay) %>%
  dplyr::filter(month == 11) %>%
  dplyr::filter(!is.na(dep_delay)) %>%
  dplyr::arrange(dest, -dep_delay) %>%
  dplyr::group_by(dest) %>%
  dplyr::summarise(
    tot_dep_delay = sum(dep_delay)
  ) %>%
  dplyr::mutate(
    tot_dep_delay_days = ((tot_dep_delay / 60) /24)
  )
```

9.9 Full pipe example

```
## # A tibble: 90 x 3
##    dest  tot_dep_delay tot_dep_delay_days
##    <chr>         <dbl>              <dbl>
##  1 ABQ             -66            -0.0458
##  2 ALB             636             0.442
##  3 ATL            8184             5.68
##  4 AUS             574             0.399
##  5 AVL             239             0.166
##  6 BDL              80             0.0556
##  7 BGR             437             0.303
##  8 BHM             412             0.286
```

```
##  9 BNA            3943            2.74
## 10 BOS            2968            2.06
## # ... with 80 more rows
```

9.10 Summary

Data manipulation

- dplyr::arrange
- dplyr::summarise
- dplyr::group_by
- dplyr::mutate

Next: Practical session

- Creating R projects
- Creating R scripts
- Data wrangling script

Chapter 10

Join operations

10.1 Recap

Prev: Selection and manipulation

- Data frames and tibbles
- Data selection and filtering
- Data manipulation

Now: Join operations

- Joining data
- dplyr join functions

10.2 Example

```
cities <- data.frame(
  city_name = c("Barcelona", "London", "Rome", "Los Angeles"),
  country_name = c("Spain", "UK", "Italy", "US"),
  city_pop_M = c(1.62, 8.98, 4.34, 3.99)
)

cities_area <-data.frame(
  city_name = c("Barcelona", "London", "Rome", "Munich"),
  city_area_km2 = c(101, 1572, 496, 310)
)
```

10.3 Example

city_name	country_name	city_pop_M
Barcelona	Spain	1.62
London	UK	8.98
Rome	Italy	4.34
Los Angeles	US	3.99

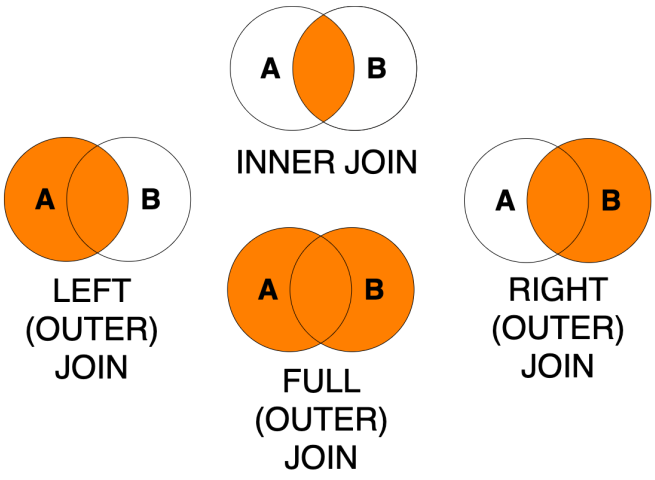
city_name	city_area_km2
Barcelona	101
London	1572
Rome	496
Munich	310

10.4 Joining data

Tables can be joined (or 'merged')

- information from two tables can be combined
- specifying **column(s) from two tables with common values**
  - usually one with a unique identifier of an entity
- rows having the same value are joined
- depending on parameters
  - a row from one table can be merged with multiple rows from the other table
  - rows with no matching values in the other table can be retained
- `merge` base function or join functions in `dplyr`

10.5 Join types



10.6 dplyr joins

`dplyr` provides a series of join verbs

- **Mutating joins**
  - `inner_join`: inner join
  - `left_join`: left join
  - `right_join`: right join
  - `full_join`: full join
- **Nesting joins**
  - `nest_join`: all rows columns from left table, plus a column of tibbles with matching from right
- **Filtering joins** (keep only columns from left)
  - `semi_join`: , rows from left where match with right
  - `anti_join`: rows from left where no match with right

10.7 dplyr::full_join

- `full_join` combines all the available data

```
dplyr::full_join(
  # first argument, left table
  # second argument, right table
  cities, cities_area,
  # specify which column to be matched
  by = c("city_name" = "city_name")
  )
```

city_name	country_name	city_pop_M	city_area_km2
Barcelona	Spain	1.62	101
London	UK	8.98	1572
Rome	Italy	4.34	496
Los Angeles	US	3.99	NA
Munich	NA	NA	310

10.8 Pipes and shorthands

When using (all) join verbs in `dplyr`

```
# using pipe, left table is "coming down the pipe"
cities %>%
  dplyr::full_join(cities_area, by = c("city_name" = "city_name"))

# if no columns specified, columns with the same name are matched
cities %>%
  dplyr::full_join(cities_area)
```

city_name	country_name	city_pop_M	city_area_km2
Barcelona	Spain	1.62	101
London	UK	8.98	1572
Rome	Italy	4.34	496
Los Angeles	US	3.99	NA
Munich	NA	NA	310

10.9 dplyr::left_join

- keeps all the data from the **left** table
  - first argument or *"coming down the pipe"*
- rows from the right table without a match are dropped
  - second argument (or first when using *pipes*)

```
cities %>%
  dplyr::left_join(cities_area)
```

city_name	country_name	city_pop_M	city_area_km2
Barcelona	Spain	1.62	101
London	UK	8.98	1572
Rome	Italy	4.34	496
Los Angeles	US	3.99	NA

10.10 dplyr::right_join

- keeps all the data from the **right** table
  - second argument (or first when using *pipes*)
- rows from the left table without a match are dropped
  - first argument or *"coming down the pipe"*

```
cities %>%
  dplyr::right_join(cities_area)
```

city_name	country_name	city_pop_M	city_area_km2
Barcelona	Spain	1.62	101
London	UK	8.98	1572
Rome	Italy	4.34	496
Munich	NA	NA	310

10.11 dplyr::inner_join

- keeps only rows that have a match in **both** tables
- rows without a match either way are dropped

```
cities %>%
  dplyr::inner_join(cities_area)
```

city_name	country_name	city_pop_M	city_area_km2
Barcelona	Spain	1.62	101
London	UK	8.98	1572
Rome	Italy	4.34	496

10.12 dplyr::semi_join and anti_join

```
cities %>%
  dplyr::semi_join(cities_area)
```

city_name	country_name	city_pop_M
Barcelona	Spain	1.62
London	UK	8.98
Rome	Italy	4.34

```
cities %>%
  dplyr::anti_join(cities_area)
```

city_name	country_name	city_pop_M
Los Angeles	US	3.99

10.13 Summary

Join operations

- Joining data
- dplyr join functions

Next: Tidy-up your data

- Wide and long data
- Re-shape data
- Handle missing values

Chapter 11

Tidy data

CONTENT WARNING: Some of the data used in these slides discuss issues that some people might find distressing: **disease**.

11.1 Recap

Prev: Join operations

- Joining data
- dplyr join functions

Now: Tidy-up your data

- Wide and long data
- Re-shape data
- Handle missing values

11.2 Long data

Each real-world entity is represented by *multiple rows*

- each one reporting only one of its attributes
- one column indicates which attribute each row represent
- another column is used to report the value

Common approach for temporal series

city	week_ending	cases
Derby	2020-10-03	NA
Leicester	2020-10-03	473
Nottingham	2020-10-03	1701
Derby	2020-10-10	320
Leicester	2020-10-10	616
Nottingham	2020-10-10	NA

11.3 Wide data

Each real-world entity is represented by *one single row*

- its attributes are represented through different columns

city	cases_2020_10_03	cases_2020_10_10
Derby	NA	320
Leicester	473	616
Nottingham	1701	NA

- **Long data** can be more flexible
  - new attributes add new rows where necessary
- **Wide data** require more structure
  - new attributes need new column for all entities

11.4 Example

```
city_info_long <- data.frame(
    city = c("Derby", "Leicester" , "Nottingham",
             "Derby", "Leicester", "Nottingham"),
    week_ending = c("2020-10-03", "2020-10-03", "2020-10-03",
                    "2020-10-10", "2020-10-10", "2020-10-10"),
    cases = c(NA, 473, 1701, 320, 616, NA)
  ) %>%
  tibble::as_tibble()
```

city	week_ending	cases
Derby	2020-10-03	NA
Leicester	2020-10-03	473
Nottingham	2020-10-03	1701
Derby	2020-10-10	320
Leicester	2020-10-10	616
Nottingham	2020-10-10	NA

11.5 tidyr

The `tidyr` (pronounced *tidy-er*) library is part of `tidyverse`

Provides a series of functions to *"tidy-up"* your data, including

- re-shape your data
  - `tidyr::pivot_wider`: pivot from long to wide
  - `tidyr::pivot_longer`: pivot from wide to long
- handle missing values
  - `tidyr::drop_na`: remove rows with missing data
  - `tidyr::replace_na`: replace missing data
  - `tidyr::fill`: fill missing data
  - `tidyr::complete`: add missing value combinations

11.6 tidyr::pivot_wider

Re-shape from **long** to **wide** format

```
city_info_wide <-
  city_info_long %>%
  tidyr::pivot_wider(
    # Column from which to extract new column names
    names_from = week_ending,
    # Column from which to extract values
    values_from = cases
  )
```

city	2020-10-03	2020-10-10
Derby	NA	320
Leicester	473	616
Nottingham	1701	NA

11.7 tidyr::pivot_wider

It might be useful (or indeed necessary) to **format** the values that will become the names of the new columns

```
city_info_wide <- city_info_long %>% dplyr::mutate(
    # Change "-" to "_" in the string representing the dates
    week_ending = stringr::str_replace_all(week_ending, "-", "_")
  ) %>%
  tidyr::pivot_wider(
    names_from = week_ending, values_from = cases, # As before
    names_prefix = "cases_" # Add a prefix
  ) # Apologies for bad coding style, need to fit code in slide :)
```

city	cases_2020_10_03	cases_2020_10_10
Derby	NA	320
Leicester	473	616
Nottingham	1701	NA

11.8 tidyr::pivot_longer

Re-shape from **wide** to **long** format

```
city_info_back_to_long <- city_info_wide %>%
  tidyr::pivot_longer(
    cols = -city, # Pivot all columns, excluding city
    names_to = "week_ending", # Name column for column names
    values_to = "cases" # Name column for values
  ) # Again, not best formatting, sorry -_-'
```

city	week_ending	cases
Derby	cases_2020_10_03	NA
Derby	cases_2020_10_10	320
Leicester	cases_2020_10_03	473
Leicester	cases_2020_10_10	616
Nottingham	cases_2020_10_03	1701
Nottingham	cases_2020_10_10	NA

11.9 tidyr::pivot_longer

It might be useful (or indeed necessary) to **format** the values extracted from the column names

```
city_info_back_to_long <- city_info_wide %>%
  tidyr::pivot_longer(
    # As before
    cols = -city, names_to = "week_ending",  values_to = "cases",
    # Remove name prefix
    names_prefix = "cases_",
    # Transform the values that will become column names
    # list of new column names <-> functions to apply
    names_transform = list(
      # Provide a function name or define one
      week_ending = function (x) {
        stringr::str_replace_all(x, "_", "-")
        }
    )
  ) # I usually format my code decently, I promise
```

11.10 tidyr::pivot_longer

... which brings us back exactly where we started.

city	week_ending	cases
Derby	2020-10-03	NA
Derby	2020-10-10	320
Leicester	2020-10-03	473
Leicester	2020-10-10	616
Nottingham	2020-10-03	1701
Nottingham	2020-10-10	NA

11.11 tidyr

The `tidyr` (pronounced *tidy-er*) library is part of `tidyverse`

Provides a series of functions to *"tidy-up"* your data, including

- re-shape your data
  - `tidyr::pivot_wider`: pivot from long to wide
  - `tidyr::pivot_longer`: pivot from wide to long
- handle missing values
  - `tidyr::drop_na`: remove rows with missing data
  - `tidyr::replace_na`: replace missing data
  - `tidyr::fill`: fill missing data
  - `tidyr::complete`: add missing value combinations

11.12 tidyr::replace_na

If the data allow for a baseline value, missing values can be **replaced**

```
city_info_long %>%
  tidyr::replace_na(
    # List of columns <-> values to replace NA
    list(cases = 0)
  )
```

city	week_ending	cases
Derby	2020-10-03	0
Leicester	2020-10-03	473
Nottingham	2020-10-03	1701
Derby	2020-10-10	320
Leicester	2020-10-10	616
Nottingham	2020-10-10	0

11.13 tidyr::fill

Sometimes it can make sense to **fill** missing values using *"nearby"* values, but **caution**, order and grouping matter!

```
city_info_long %>%
  dplyr::group_by(city) %>%
  dplyr::arrange(week_ending) %>%
  # Columns to fill
  tidyr::fill(cases)
```

city	week_ending	cases
Derby	2020-10-03	NA
Leicester	2020-10-03	473
Nottingham	2020-10-03	1701
Derby	2020-10-10	320
Leicester	2020-10-10	616
Nottingham	2020-10-10	1701

11.14 tidyr::drop_na

In other cases, it might be simpler or safer to just **remove** all the rows with missing data

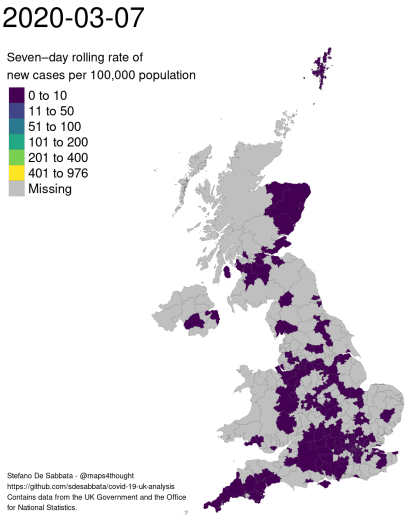
```
city_info_long_noNAs <-
  city_info_long %>%
  # Columns to drop where NA
  tidyr::drop_na(cases)
```

city	week_ending	cases
Leicester	2020-10-03	473
Nottingham	2020-10-03	1701
Derby	2020-10-10	320
Leicester	2020-10-10	616

11.15 tidyr::complete

Finally, some analysis or visualisation procedures might require a *complete* table

- where missing values are represented as NAs
- for instance, when creating a map (as in this example)
  - you might want to use a specific colour for missing values
  - rather than a missing polygon



11.16 tidyr::complete

Complete table by turning implicit missing values into explicit missing values

```
city_info_long_noNAs %>%
  # Complete table with all week_ending and city combinations
  # making missing values for remaining columns explicit
  tidyr::complete(week_ending, city)
```

city	week_ending	cases
Derby	2020-10-03	NA
Derby	2020-10-10	320
Leicester	2020-10-03	473
Leicester	2020-10-10	616
Nottingham	2020-10-03	1701
Nottingham	2020-10-10	NA

11.17 Summary

Tidy-up your data

- Wide and long data
- Re-shape data
- Handle missing values

Next: Read and write data

- file formats
- read

- write

Chapter 12

Read and write data

12.1 Summary

Tidy-up your data

- Wide and long data
- Re-shape data
- Handle missing values

Next: Read and write data

- file formats
- read
- write

12.2 Text file formats

A series of formats based on plain-text files

For instance

- comma-separated values files `.csv`
- semi-colon-separated values files `.csv`
- tab-separated values files `.tsv`
- other formats using custom delimiters
- fix-width files `.fwf`

12.3 Comma Separated Values

The file `2011_OAC_supgrp_Leicester.csv` contains

- one row for each Output Area (OA) in Leicester

- Lower-Super Output Area (LSOA) containing the OA
- code and name of the supergroup assigned to the OA by the 2011 Output Area Classification
- total population of the OA

Extract showing only the first few rows

```
OA11CD,LSOA11CD,supgrpcode,supgrpname,Total_Population
E00069517,E01013785,6,Suburbanites,313
E00069514,E01013784,2,Cosmopolitans,323
E00169516,E01013713,4,Multicultural Metropolitans,341
E00169048,E01032862,4,Multicultural Metropolitans,345
```

12.4 readr

The `readr` (pronounced *read-er*) library is part of `tidyverse`

Provides functions to read and write text files

- `readr::read_csv`: comma-separated files `.csv`
- `readr::read_csv2`: semi-colon-separated files `.csv`
- `readr::read_tsv`: tab-separated files `.tsv`
- `readr::read_fwf`: fix-width files `.fwf`
- `readr::read_delim`: files using a custom delimiter

and their *write* counterpart, such as

- `readr::write_csv`: comma-separated files `.csv`

12.5 readr::read_csv

The `readr::read_csv` function of the `readr` library reads a *csv* file from the path provided as the first argument

```
leicester_2011OAC <-
  readr::read_csv("2011_OAC_supgrp_Leicester.csv")

leicester_2011OAC
```

```
## # A tibble: 969 x 5
##   OA11CD  LSOA11CD supgrpcode supgrpname    Total_Population
##   <chr>   <chr>         <dbl> <chr>                     <dbl>
## 1 E00069~ E010137~          6 Suburbanites                313
## 2 E00069~ E010137~          2 Cosmopolitans               323
## 3 E00169~ E010137~          4 Multicultura~               341
## # ... with 966 more rows
```

12.6 Read options

Read functions provide options about how to interpret a file contents

- For instance, `readr::read_csv`
  - `col_names`:
    - `TRUE` or `FALSE` whether top row is column names
    - or a vector of column names
  - `col_types`:
    - a `cols()` specification or a string
  - `skip`: lines to skip before reading data
  - `n_max`: max number of record to read

12.7 Column specifications

- `col_logical()` or `l` as logic values
- `col_integer()` or `i` as integer
- `col_double()` or `d` as numeric (double)
- `col_character()` or `c` as character
- `col_factor(levels, ordered)` or `f` as factor
- `col_date(format = "")` or `D` as data type
- `col_time(format = "")` or `t` as time type
- `col_datetime(format = "")` or `T` as datetime
- `col_number()` or `n` as numeric (dropping marks)
- `col_skip()` or `_` or `-` don't import
- `col_guess()` or `?` use best type based on the input

12.8 readr::read_csv

Using `readr::read_csv` as in the previous example with no further options will generate the following warning

```
leicester_2011OAC <-
  readr::read_csv("2011_OAC_supgrp_Leicester.csv")

leicester_2011OAC
```

```
Parsed with column specification:
cols(
  OA11CD = col_character(),
  LSOA11CD = col_character(),
  supgrpcode = col_double(),
  supgrpname = col_character(),
  Total_Population = col_double()
)
```

12.9 readr::read_csv

```
leicester_2011OAC <- readr::read_csv(
    "2011_OAC_supgrp_Leicester.csv",
    col_types = cols(
      OA11CD  = col_character(),
      LSOA11CD  = col_character(),
      supgrpcode  = col_character(),
      supgrpname  = col_character(),
      Total_Population  = col_integer()
    )
  )
```

```
## # A tibble: 969 x 5
##   OA11CD  LSOA11CD supgrpcode supgrpname    Total_Population
##   <chr>   <chr>    <chr>      <chr>                     <int>
## 1 E00069~ E010137~ 6          Suburbanites                313
## 2 E00069~ E010137~ 2          Cosmopolitans               323
## 3 E00169~ E010137~ 4          Multicultura~               341
## # ... with 966 more rows
```

12.10 readr::read_csv

```
leicester_2011OAC <- readr::read_csv(
    "2011_OAC_supgrp_Leicester.csv",
    col_types = "cccci"
  )
```

```
## # A tibble: 969 x 5
##   OA11CD  LSOA11CD supgrpcode supgrpname    Total_Population
##   <chr>   <chr>    <chr>      <chr>                     <int>
## 1 E00069~ E010137~ 6          Suburbanites                313
## 2 E00069~ E010137~ 2          Cosmopolitans               323
## 3 E00169~ E010137~ 4          Multicultura~               341
## 4 E00169~ E010328~ 4          Multicultura~               345
## 5 E00169~ E010328~ 4          Multicultura~               322
## 6 E00069~ E010136~ 4          Multicultura~               334
## 7 E00169~ E010328~ 4          Multicultura~               336
## # ... with 962 more rows
```

12.11 readr::write_csv

The function `write_csv` can be used to save a dataset to `csv`

Example:

1. **read** the 2011 OAC dataset
2. **select** a few columns
3. **filter** only those OA in the supergroup *Suburbanites* (code 6)
4. **write** the results to a file named *2011_OAC_supgrp_Leicester_supgrp6.csv*

```
readr::read_csv("2011_OAC_supgrp_Leicester.csv") %>%
  dplyr::select(OA11CD, supgrpcode, Total_Population) %>%
  dplyr::filter(supgrpcode == "6") %>%
  readr::write_csv("2011_OAC_supgrp_Leicester_supgrp6.csv")
```

12.12 readr::write_tsv

```
readr::read_csv("2011_OAC_supgrp_Leicester.csv") %>%
  dplyr::select(OA11CD, supgrpcode, Total_Population) %>%
  dplyr::filter(supgrpcode == "6") %>%
  readr::write_tsv("2011_OAC_supgrp_Leicester_supgrp6.tsv")
```

```
OA11CD     supgrpcode Total_Population
E00069517 6          313
E00069468 6          251
E00069528 6          270
E00069538 6          307
E00069174 6          321
E00069170 6          353
E00069171 6          351
E00068713 6          265
E00069005 6          391
E00069014 6          316
E00068989 6          354
```

12.13 Other data imports

Tidyverse also imports other packages for reading data

- Tabular formats
  - `readxl` for Excel (`.xls` and `.xlsx`)
  - `haven` for SPSS, Stata, and SAS data.
- Databases
  - `DBI` for relational databases
- NoSQL
  - `jsonlite` for JSON
  - `xml2` for XML
- Web
  - `httr` for web APIs

12.14 Summary

Read and write data

- file formats
- read
- write

Next: Practical session

- Read and write data
- Tidy data
- Join operations

Chapter 13

Reproducibility

13.1 Recap

Prev: Table operations

- 211 Join operations
- 212 Data pivot
- 213 Read and write data
- 214 Practical session

Now: Reproduciblity

- Reproduciblity and software engineering
- Reproduciblity in GIScience
- Guidelines

13.2 Reproduciblity

In quantitative research, an analysis or project are considered to be **reproducible** if:

- *"the data and code used to make a finding are available and they are sufficient for an independent researcher to recreate the finding."* Christopher Gandrud, *Reproducible Research with R and R Studio*

That is becoming more and more important in science:

- as programming and scripting are becoming integral in most disciplines
- as the amount of data increases

13.3 Why?

In **scientific research**:

- verificability of claims through replication
- incremental work, avoid duplication

For your **working practice**:

- better working practices
  - coding
  - project structure
  - versioning
- better teamwork
- higher impact (not just results, but code, data, etc.)

13.4 Reproducibility and software engineering

Core aspects of **software engineering** are:

- project design
- software **readibility**
- testing
- **versioning**

As programming becomes integral to research, similar necessities arise among scientists and data analysts.

13.5 Reproducibility and "big data"

There has been a lot of discussions about **"big data"**…

- volume, velocity, variety, …

Beyond the hype of the moment, as the **amount** and **complexity** of data increases

- the time required to replicate an analysis using point-and-click software becomes unsustainable
- room for error increases

Workflow management software (e.g., ArcGIS ModelBuilder) is one answer, reproducible data analysis based on script languages like R is another.

13.6 Reproducibility in GIScience

Singleton *et al.* have discussed the issue of reproducibility in GIScience, identifying the following best practices:

1. Data should be accessible within the public domain and available to researchers.
2. Software used should have open code and be scrutable.
3. Workflows should be public and link data, software, methods of analysis and presentation with discursive narrative
4. The peer review process and academic publishing should require submission of a workflow model and ideally open archiving of those materials necessary for replication.
5. Where full reproducibility is not possible (commercial software or sensitive data) aim to adopt aspects attainable within circumstances

13.7 Document everything

In order to be reproducible, every step of your project should be documented in detail

- data gathering
- data analysis
- results presentation

Well documented R scripts are an excellent way to document your project.

13.8 Document well

Create code that can be **easily understood** by someone outside your project, including yourself in six-month time!

- use a style guide (e.g. tidyverse) consistently
- also add a **comment** before any line that could be ambiguous or particularly difficult or important
- add a **comment** before each code block, describing what the code does
- add a **comment** at the beginning of a file, including
  - date
  - contributors
  - other files the current file depends on
  - materials, sources and other references

13.9 Workflow

Relationships between files in a project are not simple:

- in which order are file executed?
- when to copy files from one folder to another, and where?

A common solution is using **make files**

- commonly written in *bash* on Linux systems

- they can be written in R, using commands like
  - *source* to execute R scripts
  - *system* to interact with the operative system

13.10 granolarr Mark.R

Section of the *granolarr* project make file Make.R that generates the current slides for the lecture session 221

```
cat("\n\n>>> Rendering 221_L_Reproducibility.Rmd <<<\n\n")
rmarkdown::render(
    paste0(
        Sys.getenv("GRANOLARR_HOME"),
        "/src/lectures/221_L_Reproducibility.Rmd"
    ),
    quiet = TRUE,
    output_dir = paste0(
        Sys.getenv("GRANOLARR_HOME"),
        "/docs/lectures/html"
    )
)
```

13.11 Future-proof formats

Complex formats (e.g., .docx, .xlsx, .shp, ArcGIS .mxd)

- can become obsolete
- are not always portable
- usually require proprietary software

Use the simplest format to **future-proof** your analysis.**Text files** are the most versatile

- data: .txt, .csv, .tsv
- analysis: R scrpts, python scripts
- write-up: LaTeX, Markdown, HTML

13.12 Store and share

Reproducible data analysis is particularly important when working in teams, to share and communicate your work.

- Dropbox
  - good option to work in teams, initially free
  - no versioning, branches
- Git

  - free and opensource control system
  - great to work in teams and share your work publically
  - can be more difficult at first
  - GitHub public repositories are free, private ones are not
  - GitLab offers free private repositories

13.13 Summary

Reproduciblity

- Reproduciblity and software engineering
- Reproduciblity in GIScience
- Guidelines

Next: RMarkdown

- Markdown
- RMarkdown

Chapter 14

RMarkdown

14.1 Recap

Prev: Reproduciblity

- Reproduciblity and software engineering
- Reproduciblity in GIScience
- Guidelines

Now: RMarkdown

- Markdown
- RMarkdown

14.2 Markdown

Markdown is a simple markup language

- allows to mark-up plain text
- to specify more complex features (such as *italics text*)
- using a very simple syntax

Markdown can be used in conjunction with numerous tools

- to produce HTML pages
- or even more complex formats (such as PDF)

These slides are written in Markdown

14.3 Markdown example code

```
### This is a third level heading
```

```
Text can be specified as *italic* or **bold**

- and list can be created
    - very simply

1. also numbered lists
    1. [add a link like this](http://le.ac.uk)

|Tables |Can         |Be       |
|-------|------------|---------|
|a bit  |complicated |at first |
|but    |it gets     |easier   |
```

14.4 Markdown example output

14.4.1 This is a third level heading

Text can be specified as *italic* or **bold**

- and list can be created
  - very simply

1. also numbered lists
   1. add a link like this

Tables	Can	Be
a bit	complicated	at first
but	it gets	easier

14.5 RMarkdown

The rmarkdown library and its RStudio plug-in

- provide functionalities to *compile* scripts containing
  - **Markdown** text
    - rendered to documents (e.g., *.pdf* and *.doc*)
  - chunks of **R** code (other supported, e.g., Python, SQL)
    - included in output document
    - interpreted
    - results included in output document

```
```{r, echo=TRUE}
# Example of R chunck
sqrt(2)
```
```

14.6 RMarkdown example

Content of an RMarkdown file: `First_example.Rmd`

````
This is an **RMarkdown** document. The *code chunk* below:

- loads the necessary libraries
- loads the flights from New York City in 2013
- presents a few columns from the first row

```{r, echo=TRUE, message=FALSE, warning=FALSE}
library(tidyverse)
library(nycflights13)

nycflights13::flights %>%
  dplyr::select(year:day, origin, dest, flight) %>%
  dplyr::slice_head(1) %>%
  knitr::kable()
```
````

14.7 RMarkdown example

This is an **RMarkdown** document. The *code chunk* below:

- loads the necessary libraries
- loads the flights from New York City in 2013
- presents a few columns from the first row

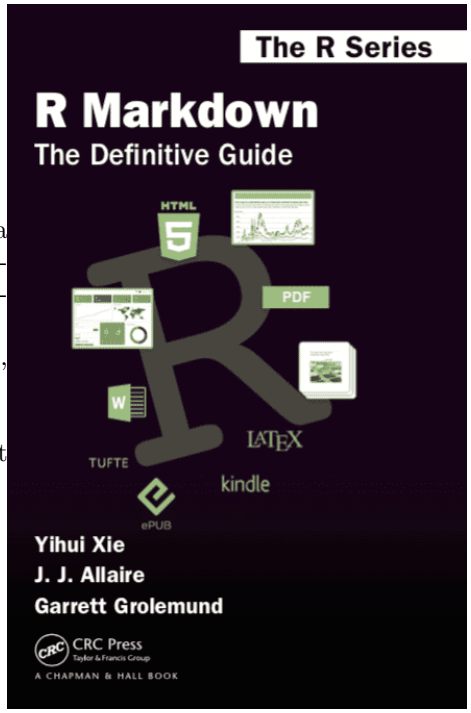
```r
library(tidyverse)
library(nycflights13)

nycflights13::flights %>%
  dplyr::select(year:day, origin, dest, flight) %>%
  dplyr::slice_head(1) %>%
  knitr::kable()
```

year	month	day	origin	dest	flight
2013	1	1	EWR	IAH	1545

14.8 The Definitive Guide

Markdown is a rather simple for a markup language, but still fairly complex, especially when used in combination with R.
For an complete guide to RMarkdown, please see:
R Markdown:The Definitive Guide
by Yihui Xie, J. J. Allaire, Garrett Grolemund.



14.9 Summary

RMarkdown

- Markdown
- RMarkdown

Next: Git and Docker

- Git operations
- Git and RStudio
- Docker

Chapter 15

Git

15.1 Recap

RMarkdown

- Markdown
- RMarkdown

Next: Git and Docker

- Git operations
- Git and RStudio
- Docker

15.2 What's git?

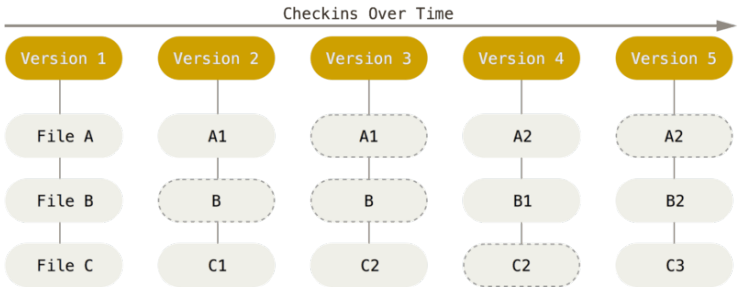
Git is a free and opensource version control system

- commonly used through a server
  - where a master copy of a project is kept
  - can also be used locally
- allows storing versions of a project
  - syncronisation
  - consistency
  - history
  - multiple branches

15.3 How git works

A series of snapshots

- each commit is a snapshot of all files
- if no change to a file, link to previous commit
- all history stored locally

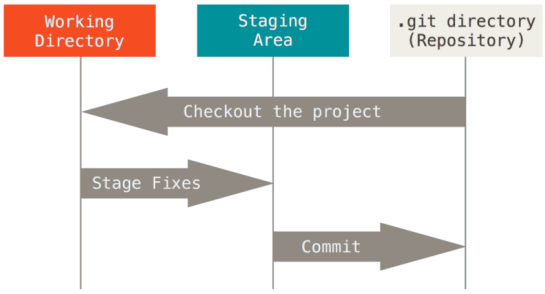


by Scott Chacon and Ben Straub, licensed under CC BY-NC-SA 3.0

15.4 Three stages

When working with a git repository

- first checkout the latest version
- select the edits to stage
- commit what has been staged in a permanent snapshot



by Scott Chacon and Ben Straub, licensed under CC BY-NC-SA 3.0

15.5 Basic git commands

- `git clone`
  - copy a repository from a server
- `git fetch`
  - get the latest version from a branch
- `git pull`
  - incorporate changes from a remote repository
- `git add`
  - stage new files

- `git commit`
  - create a commit
- `git push`
  - upload commits to a remote repository

15.6 Git and RStudio

Git can be used

- from the system terminal or shell, using the `git` command
- dedicated apps such as GitHub Desktop
- RStudio git plug-in (top-right panel, `Git` tab)

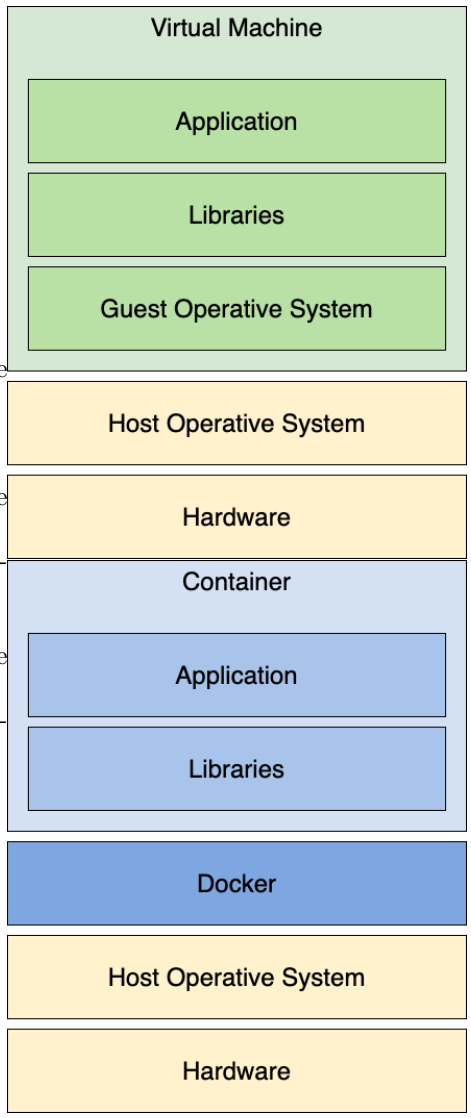
All approaches allow to

- clone R projects from repositories
- stage and commit changes
- push to remote copy
- pull changes from remote copy

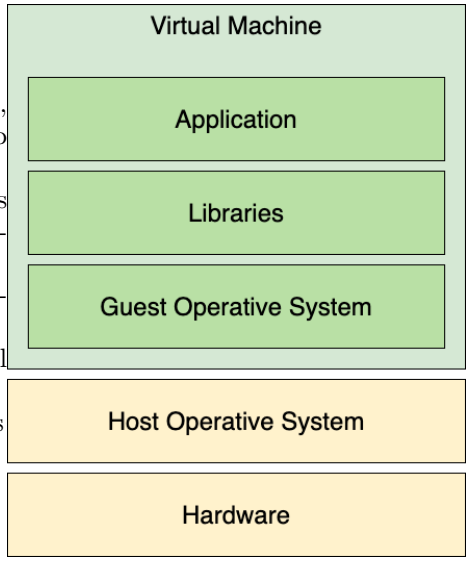
15.7 What's Docker?

Docker allows to encapsulate and share computational environments

- First released in 2013
- Similar to virtual machines
  - simulates a guest operative system
  - within a host operative system
- Lightweight
  - doesn't simulate an entire system
  - only the *"user space"* is simulated



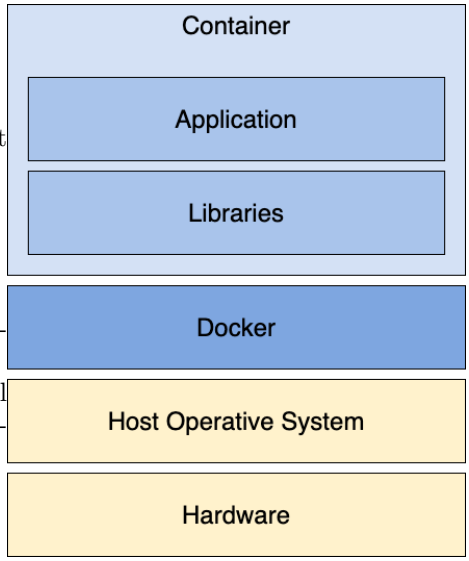
15.8 Virtual machines



Virtual machines software (e.g., VMWare) simulate a computer on top of your operative system

- allows **virtual machine** to access physical resources (e.g., disk, keyboard) of a **host**
- allows to run full operative systems
- e.g., run a full Windows virtual machine on a Mac host
- have been around since the 1970s
- can be *heavy* to run

15.9 Docker containers



Docker runs *containers*

- developed for flexible deployment of (web) services
  - compartimentalised
  - lightweight
  - (frequently) transient
- **kernel** is not simulated
  - kernels are the bulk of operative systems
  - containers share host's kernel
  - can also share binaries and libraries

15.10 Docker and reproducibility

Why are dockers useful for reproducibility?

One of the key issues of reproducing a study is replicating the computational environment used

- e.g., all the libraries in their correct version

Creating a Docker image (from which a container is instantiated)

- defined using a *Dokerfile*
- requires to list a full system configuration
  - version of programming language, libraries, etc
- once created / defined
  - other researchers or developers can run your script **in the exact same computational environment**

15.11 granolarr Dockerfile

```
# Base image https://hub.docker.com/r/rocker/ml
FROM rocker/geospatial:4.0.2

# create an R user
ENV USER rstudio

## Install additional required R libraries
COPY ./DockerConfig/Requirements.R /tmp/Requirements.R
RUN Rscript /tmp/Requirements.R

## Install additional required TeX libraries
RUN tlmgr install amsmath
RUN tlmgr install latex-amsmath-dev
RUN tlmgr install iftex
RUN tlmgr install euenc
RUN tlmgr install fontspec
```

[... continues]

15.12 Summary

Git and Docker

- Git operations
- Git and RStudio
- Docker

Next: Practical

- Reproducibile data analysis
- RMarkdown
- Git

Chapter 16

Data visualisation

16.1 Recap

Prev: Reproducibility

- 221 Reproducibility
- 222 R and Markdown
- 223 Git
- 224 Practical session

Now: Data visualisation

- Grammar of graphics
- ggplot2

16.2 Grammar of graphics

Grammars provide rules for languages

"The grammar of graphics takes us beyond a limited set of charts (words) to an almost unlimited world of graphical forms (statements)" (Wilkinson, 2005)

Statistical graphic specifications are expressed in six statements:

1. **Data** manipulation
2. **Variable** transformations (e.g., rank),
3. **Scale** transformations (e.g., log),
4. **Coordinate system** transformations (e.g., polar),
5. **Element**: mark (e.g., points) and visual variables (e.g., color)
6. **Guides** (axes, legends, etc.).

16.3 Visual variables

A **visual variable** is an aspect of a **mark** that can be controlled to change its appearance.

Visual variables include:

- Size
- Shape
- Orientation
- Colour (hue)
- Colour value (brightness)
- Texture
- Position (2 dimensions)

16.4 ggplot2

The `ggplot2` library offers a series of functions for creating graphics **declaratively**, based on the Grammar of Graphics.

To create a graph in `ggplot2`:

- provide the data
- specify elements
  - which visual variables (`aes`)
  - which marks (e.g., `geom_point`)
- apply transformations
- guides

16.5 Aesthetics

The `aes` element provides a *"mapping"* from the data *columns* (attributes) to the graphic's *visual variables*, including:

- `x` and `y`
- `fill` (fill colour) and `colour` (border colour)
- `shape`
- `size`

```
data %>%
  ggplot2::ggplot(
    aes(
      x = column_1,
      y = column_2
    )
  )
```

16.6 Graphical primitives

Marks (graphical primitives) can be specified through a series of functions, such as `geom_line`, `geom_bar` or `geom_point`

These can be added to the construction of the graph using `+`

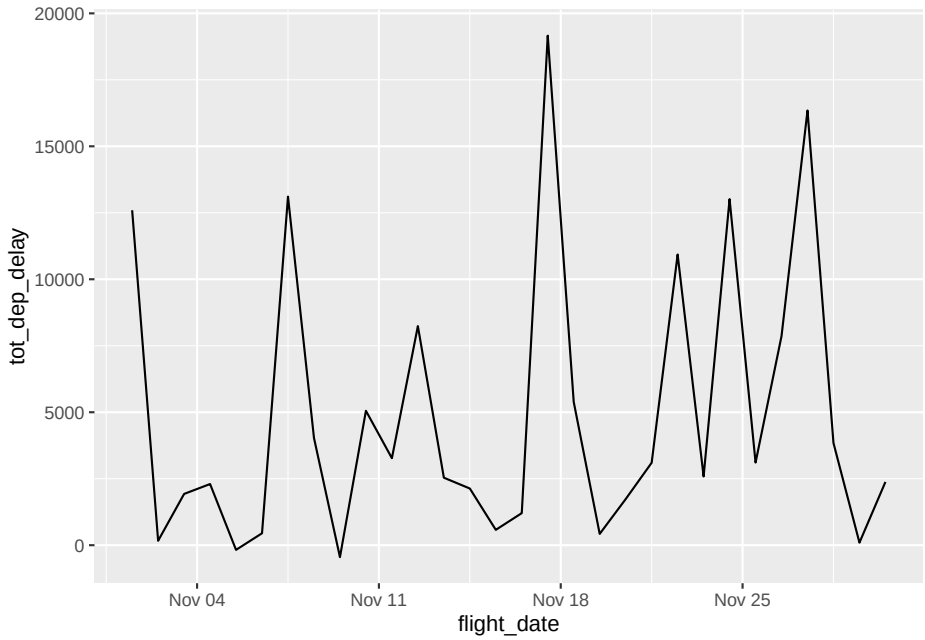
```
ggplot2::ggplot(
  aes(
    x = column_1, y = column_2
  )
) +
ggplot2::geom_line()
```

16.7 ggplot2::geom_line

- `x`: a column to *"map"* to the x-axis, e.g. days (category)
- `y`: a column to *"map"* to the y-axis, e.g. delay (continuous)
- `ggplot2::geom_line`: line mark (graphical primitive)

```
nycflights13::flights %>%
  dplyr::filter(!is.na(dep_delay) & month == 11) %>%
  dplyr::mutate(flight_date = ISOdate(year, month, day)) %>%
  dplyr::group_by(flight_date) %>%
  dplyr::summarize(tot_dep_delay = sum(dep_delay)) %>%
  ggplot2::ggplot(aes(
    x = flight_date,
    y = tot_dep_delay
  )) +
  ggplot2::geom_line()
```

16.8 ggplot2::geom_line

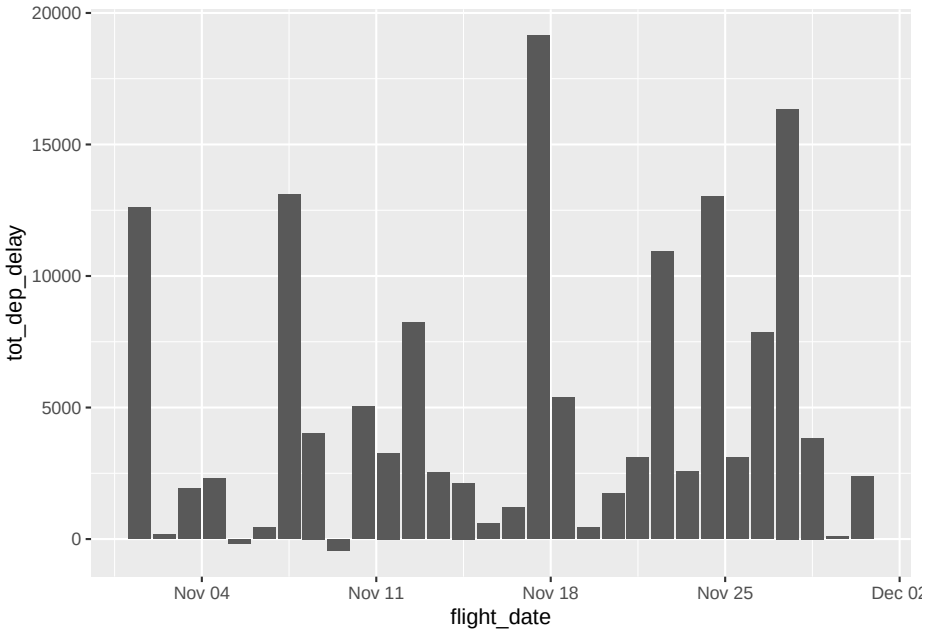


16.9 ggplot2::geom_col

- `x`: a column to *"map"* to the x-axis, e.g. days (category)
- `y`: a column to *"map"* to the y-axis, e.g. delay (continuous)
- `ggplot2::geom_col`: bar mark (graphical primitive)
  - `ggplot2::geom_bar` instead illustrates count per category

```
nycflights13::flights %>%
  dplyr::filter(!is.na(dep_delay) & month == 11) %>%
  dplyr::mutate(flight_date = ISOdate(year, month, day)) %>%
  dplyr::group_by(flight_date) %>%
  dplyr::summarize(tot_dep_delay = sum(dep_delay)) %>%
  ggplot2::ggplot(aes(
    x = flight_date,
    y = tot_dep_delay
  )) +
  ggplot2::geom_col()
```

16.10 ggplot2::geom_col



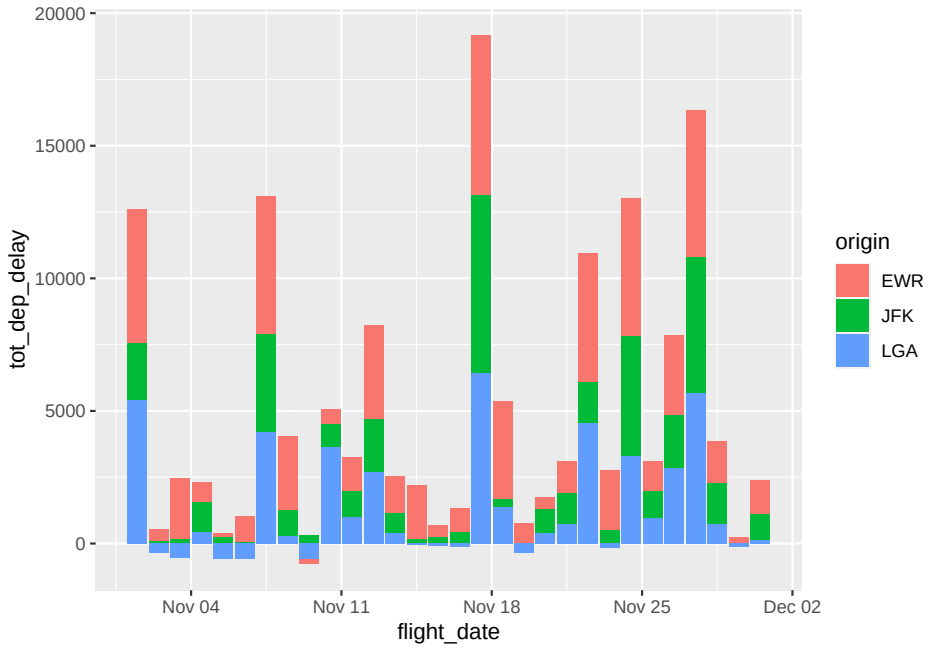
16.11 ggplot2::geom_col

... then, why not add some colour?

- `fill`: a column to *"map"* to the visual variable *colour* as fill of the mark, e.g. origin (category)
  - `colour` can be used to *"map"* a column to the visual variable *colour* as border of the mark

```
nycflights13::flights %>%
  dplyr::filter(!is.na(dep_delay) & month == 11) %>%
  dplyr::mutate(flight_date = ISOdate(year, month, day)) %>%
  dplyr::group_by(flight_date, origin) %>%
  dplyr::summarize(tot_dep_delay = sum(dep_delay)) %>%
  ggplot2::ggplot(aes(
    x = flight_date,
    y = tot_dep_delay,
    fill = origin
  )) +
  ggplot2::geom_col()
```

16.12 ggplot2::geom_col

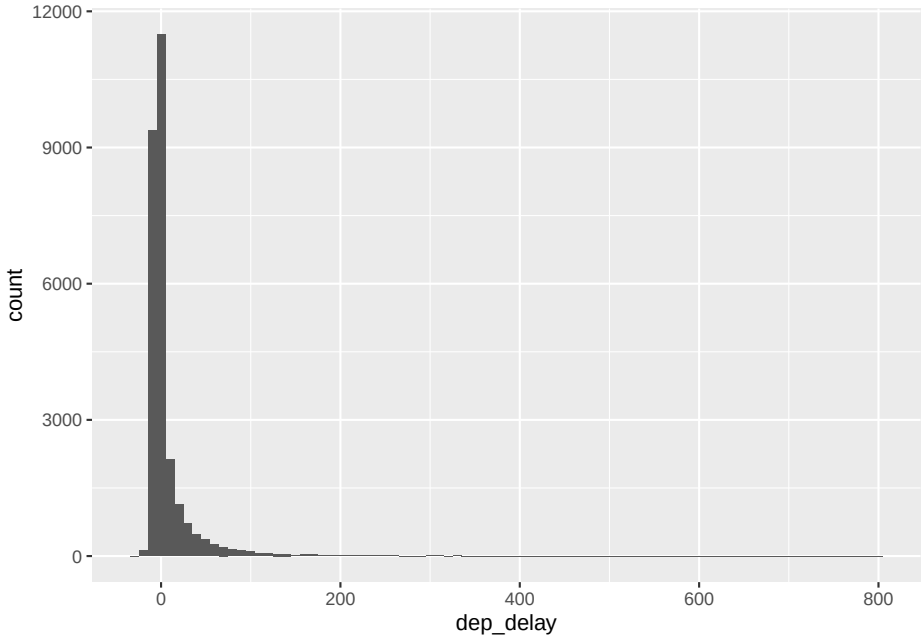


16.13 Histograms

- `x` a column to *"map"* to the x-axis, e.g. delay (continuous)
- `ggplot2::geom_histogram` to illustrate count over intervals of continuous variable on x-axis
  - `ggplot2::geom_bar` instead illustrates count per category

```
nycflights13::flights %>%
  dplyr::filter(month == 11) %>%
  ggplot2::ggplot(
    aes(
      x = dep_delay
    )
  ) +
  ggplot2::geom_histogram(
    binwidth = 10
  )
```

16.14 Histograms



:::

```
nycflights13::flights %>%
  filter(month == 11) %>%
  ggplot2::ggplot(
    aes(
      x = distance
    )
  ) +
  ggplot2::geom_histogram() +
  scale_x_log10()
```

::::::

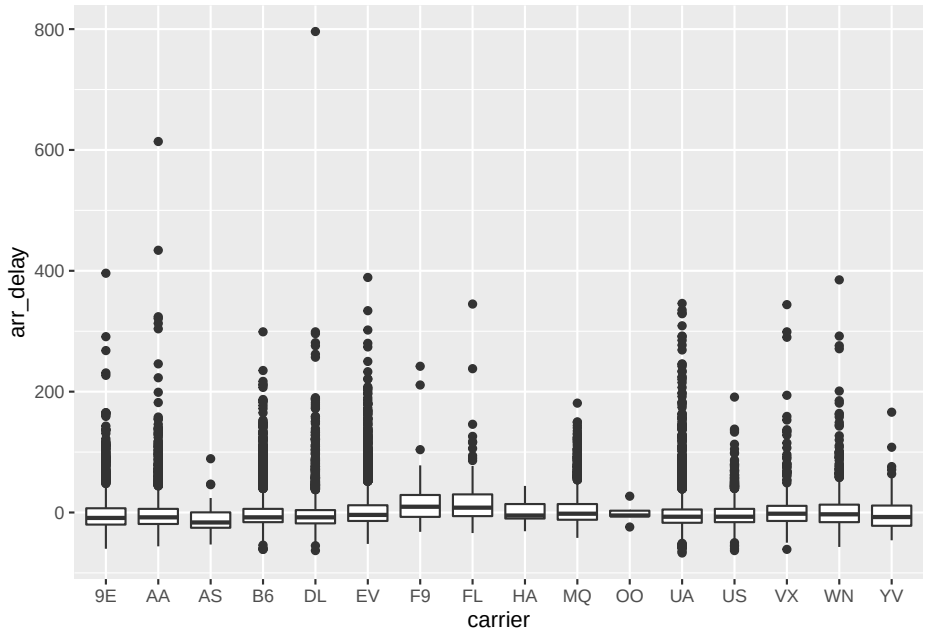
16.15 Boxplots

- `x`: a column to *"map"* to the x-axis, e.g. carrier (category)
- `y`: a column to *"map"* to the y-axis, e.g. delay (continuous)
- `geom_boxplot`: to illustrate distribution of continuous variable on y-axis per each category on x-axis

```
nycflights13::flights %>%
  dplyr::filter(month == 11) %>%
  ggplot2::ggplot(
    aes(
      x = carrier,
```

```
      y = arr_delay
    )
  ) +
  ggplot2::geom_boxplot()
```

16.16 Boxplots

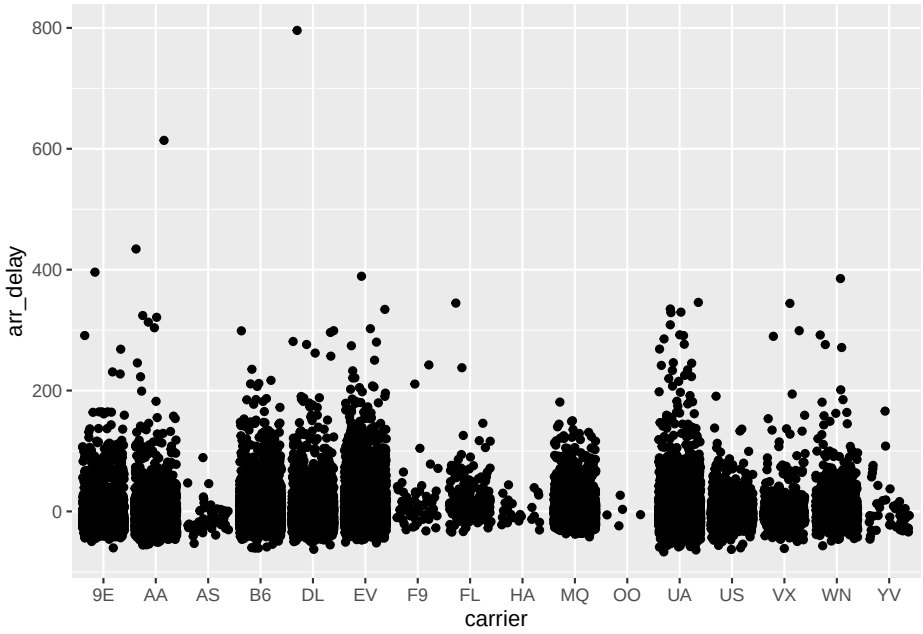


16.17 Jittered points

- `x` categorical variable
- `y` variable to plot
- `geom_jitter`

```
nycflights13::flights %>%
  dplyr::filter(month == 11) %>%
  ggplot2::ggplot(
    aes(
      x = carrier,
      y = arr_delay
    )
  ) +
  ggplot2::geom_jitter()
```

16.18 Jittered points

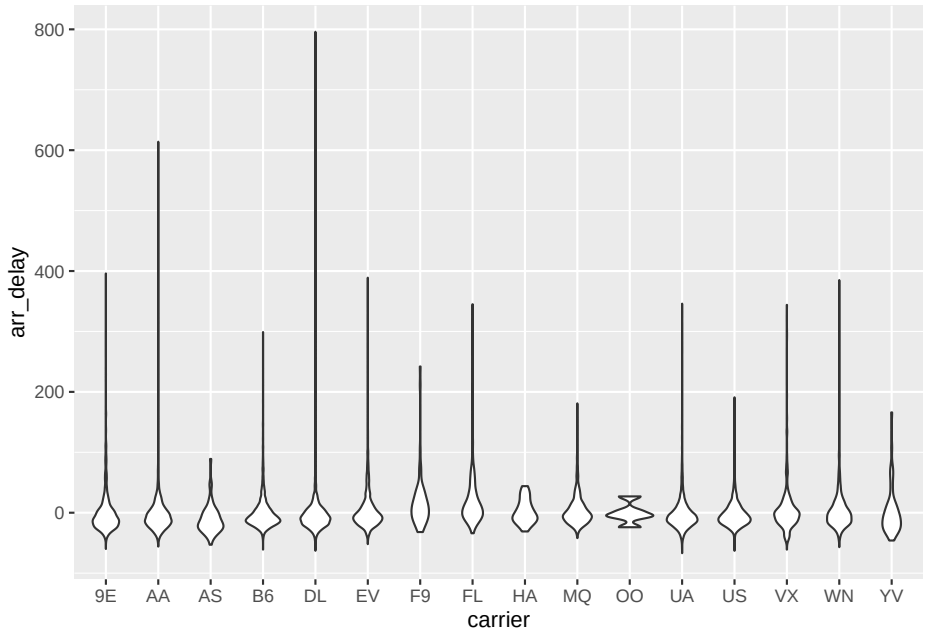


16.19 Violin plot

- x categorical variable
- y variable to plot
- geom_violin

```
nycflights13::flights %>%
  dplyr::filter(month == 11) %>%
  ggplot2::ggplot(
    aes(
      x = carrier,
      y = arr_delay
    )
  ) +
  ggplot2::geom_violin()
```

16.20 Violin plot



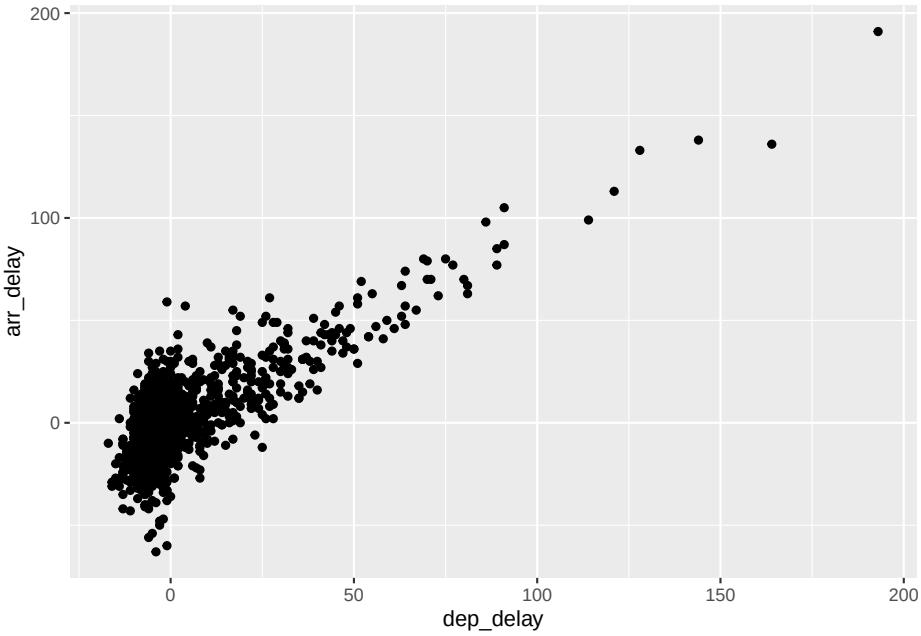
–>

16.21 Scatterplots

- `x` and `y` variables to plot
- `ggplot2::geom_point`

```
nycflights13::flights %>%
  dplyr::filter(
    month == 11,
    carrier == "US",
    !is.na(dep_delay),
    !is.na(arr_delay)
  ) %>%
  ggplot2::ggplot(aes(
    x = dep_delay,
    y = arr_delay
  )) +
  ggplot2::geom_point()
```

16.22 Scatterplots

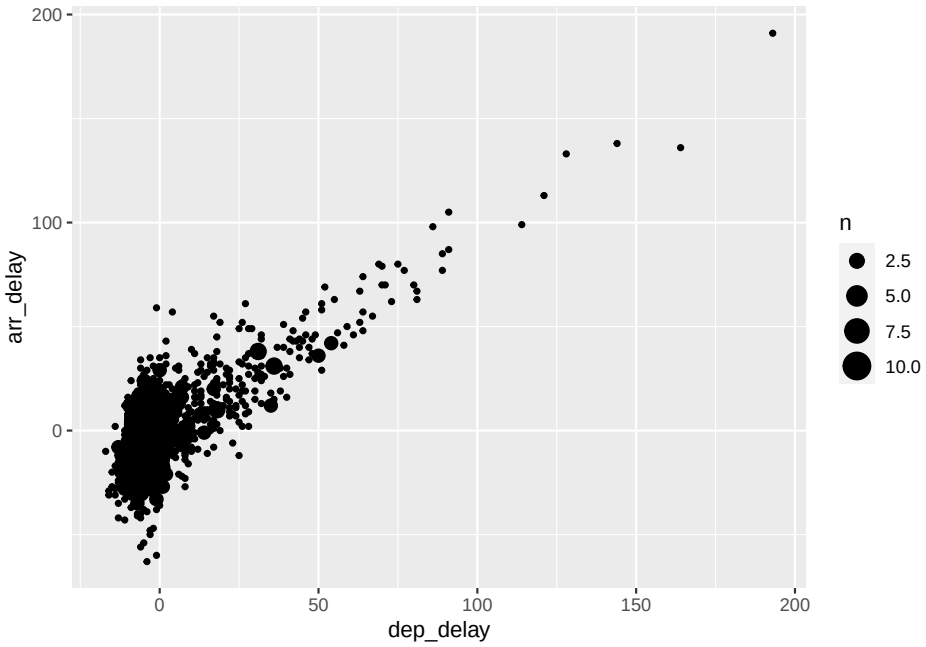


16.23 Overlapping points

- `x` and `y` variables to plot
- `ggplot2::geom_count` counts overlapping points and maps the count to size

```
nycflights13::flights %>%
  dplyr::filter(
    month == 11, carrier == "US",
    !is.na(dep_delay), !is.na(arr_delay)
  ) %>%
  ggplot2::ggplot(aes(
    x = dep_delay,
    y = arr_delay
  )) +
  ggplot2::geom_count()
```

16.24 Overlapping points

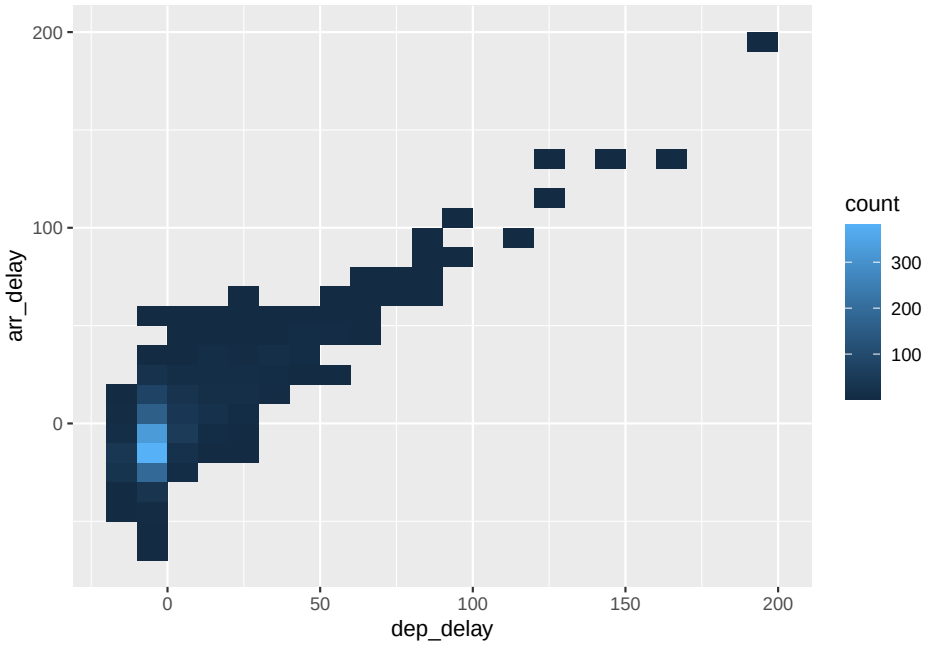


16.25 Bin counts

- `x` and `y` variables to plot
- `ggplot2::geom_bin2d` with 10 minutes binwidth

```
nycflights13::flights %>%
  dplyr::filter(
    month == 11,
    carrier == "US",
    !is.na(dep_delay),
    !is.na(arr_delay)
  ) %>%
  ggplot2::ggplot(aes(
    x = dep_delay,
    y = arr_delay
  )) +
  ggplot2::geom_bin2d(binwidth = 10)
```

16.26 Bin counts

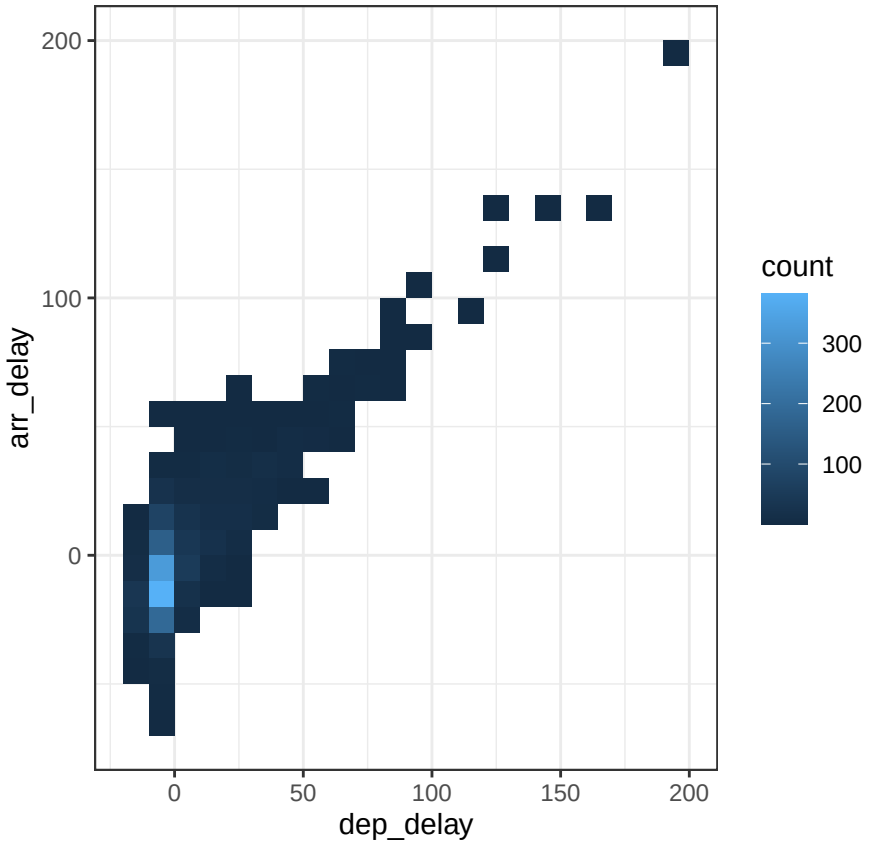


16.27 Coordinates transformations

- `ggplot2::coord_fixed` manipulates coordinates property
- `ggplot2::theme_bw` classic dark-on-light theme

```
nycflights13::flights %>%
  dplyr::filter(
    month == 11,
    carrier == "US",
    !is.na(dep_delay),
    !is.na(arr_delay)
  ) %>%
  ggplot2::ggplot(aes(
    x = dep_delay,
    y = arr_delay
  )) +
  ggplot2::geom_bin2d(binwidth = 10) +
  ggplot2::coord_fixed(ratio = 1) +
  theme_bw()
```

16.28 Coordinates transformations



16.29 Summary

Data visualisation

- Grammar of graphics
- ggplot2

Next: Descriptive statistics

- pastecs::stat.desc
- dplyr::across

Chapter 17

Descriptive statistics

17.1 Summary

Data visualisation

- Grammar of graphics
- ggplot2

Next: Descriptive statistics

- pastecs::stat.desc
- dplyr::across

17.2 Meet the Palmer penguins

Original data collected and released by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, a member of the Long Term Ecological Research Network.
Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. doi:10.5281/zenodo.3960218.

```
library(palmerpenguins)
```

Artwork by @allison_horst

17.3 Descriptive statistics

Quantitatively describe or summarize variables

- `stat.desc` from `pastecs` library
  - `base` includes counts
  - `desc` includes descriptive stats
  - `norm` (default is `FALSE`) includes distribution stats

```
library(pastecs)

palmerpenguins::penguins %>%
  dplyr::select(bill_length_mm, bill_depth_mm) %>%
  pastecs::stat.desc() %>%
  knitr::kable(digits = c(2, 2))
```

17.4 stat.desc output

	bill_length_mm	bill_depth_mm
nbr.val	342.00	342.00
nbr.null	0.00	0.00
nbr.na	2.00	2.00
min	32.10	13.10
max	59.60	21.50
range	27.50	8.40
sum	15021.30	5865.70
median	44.45	17.30
mean	43.92	17.15
SE.mean	0.30	0.11
CI.mean.0.95	0.58	0.21
var	29.81	3.90
std.dev	5.46	1.97
coef.var	0.12	0.12

17.5 stat.desc: basic

- `nbr.val`: overall number of values in the dataset
- `nbr.null`: number of `NULL` values – NULL is often returned by expressions and functions whose values are undefined
- `nbr.na`: number of `NA`s – missing value indicator

	bill_length_mm	bill_depth_mm
nbr.val	342.0	342.0
nbr.null	0.0	0.0
nbr.na	2.0	2.0
min	32.1	13.1
max	59.6	21.5
range	27.5	8.4
sum	15021.3	5865.7

17.6 stat.desc: basic

- `min` (also `min()`): **minimum** value in the dataset
- `max` (also `max()`): **maximum** value in the dataset
- `range`: difference between `min` and `max` (different from `range()`)
- `sum` (also `sum()`): sum of the values in the dataset

	bill_length_mm	bill_depth_mm
nbr.val	342.0	342.0
nbr.null	0.0	0.0
nbr.na	2.0	2.0
min	32.1	13.1
max	59.6	21.5
range	27.5	8.4
sum	15021.3	5865.7

17.7 stat.desc: desc

- `mean` (also `mean()`): **arithmetic mean**, that is `sum` over the number of values not `NA`
- `median` (also `median()`): **median**, that is the value separating the higher half from the lower half the values
- `mode()`function is available: **mode**, the value that appears most often in the values

	bill_length_mm	bill_depth_mm
median	44.45	17.30
mean	43.92	17.15
SE.mean	0.30	0.11
CI.mean.0.95	0.58	0.21
var	29.81	3.90
std.dev	5.46	1.97
coef.var	0.12	0.12

17.8 Sample statistics

Assuming that the data in the dataset are a sample of a population

- `SE.mean`: **standard error of the mean** – estimation of the variability of the mean calculated on different samples of the data (see also *central limit theorem*)
- `CI.mean.0.95`: **95% confidence interval of the mean** – indicates that there is a 95% probability that the actual mean is within that distance from the sample mean

17.9 Estimating variation

- `var`: **variance** (σ^2), it quantifies the amount of variation as the average of squared distances from the mean

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(\mu - x_i)^2$$

- `std.dev`: **standard deviation** (σ), it quantifies the amount of variation as the square root of the variance

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mu - x_i)^2}$$

- `coef.var`: **variation coefficient** it quantifies the amount of variation as the standard deviation divided by the mean

17.10 dplyr::across

The `dplyr` verb `across` allows to apply `summarise` verbs on multiple columns. Instead of...

```
palmerpenguins::penguins %>%
  # filter out raws with missing data
  dplyr::filter(!is.na(bill_length_mm)) %>%
  # summarise
  dplyr::summarise(
    avg_bill_len_mm = mean(bill_length_mm),
    avg_bill_dpt_mm = mean(bill_depth_mm),
    avg_flip_len_mm = mean(flipper_length_mm),
    avg_body_mass_g = mean(body_mass_g)
  ) %>%
  knitr::kable(digits = c(2, 2, 2, 2))
```

avg_bill_len_mm	avg_bill_dpt_mm	avg_flip_len_mm	avg_body_mass_g
43.92	17.15	200.92	4201.75

17.11 dplyr::across

The verb `across` can also be used with `mutate`, to apply the same function to a number of columns

```
palmerpenguins::penguins %>%
  # mutate cross columns
  dplyr::mutate(
    dplyr::across(
      c(bill_length_mm, bill_depth_mm, flipper_length_mm),
      # add 1 to all values in the columns above
      function(x){ x / 25.4 }
    )
  ) %>%
  rename(
    bill_length_in = bill_length_mm,
    bill_depth_in = bill_depth_mm,
    flipper_length_in = flipper_length_mm
  )
```

17.12 dplyr::across

Old columns:

```
## # A tibble: 344 x 3
##   bill_length_mm bill_depth_mm flipper_length_mm
##            <dbl>         <dbl>             <int>
## 1           39.1          18.7               181
## 2           39.5          17.4               186
## # ... with 342 more rows
```

New columns:

```
## # A tibble: 344 x 3
##   bill_length_in bill_depth_in flipper_length_in
##            <dbl>         <dbl>             <dbl>
## 1           1.54         0.736              7.13
## 2           1.56         0.685              7.32
## # ... with 342 more rows
```

17.13 Summary

Descriptive statistics

- pastecs::stat.desc
- dplyr::across

Next: Exploring assumptions

- Normality
- Skewness and kurtosis
- Homogeneity of variance

Chapter 18

Exploring assumptions

18.1 Recap

Prev: Descriptive statistics

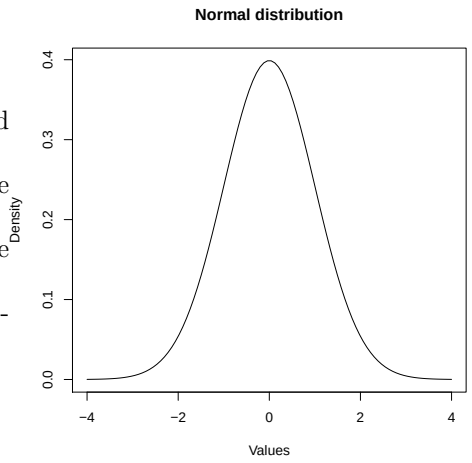
- stat.desc
- dplyr::across

Next: Exploring assumptions

- Normality
- Skewness and kurtosis
- Homogeneity of variance

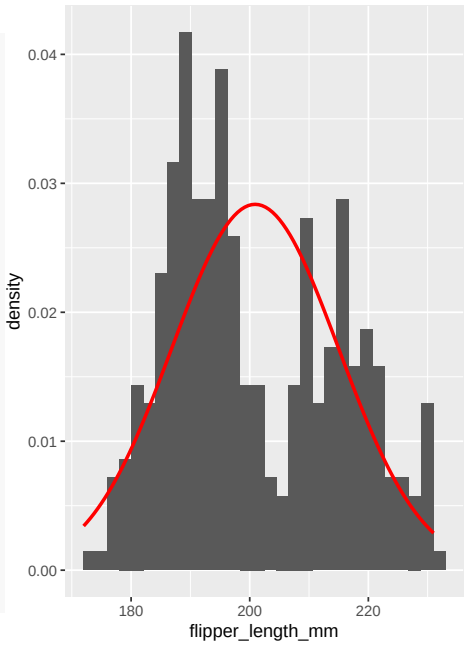
18.2 Normal distribution

- characterized by the bell-shaped curve
- majority of values lie around the centre of the distribution
- the further the values are from the centre, the lower their frequency
- about 95% of values within 2 standard deviations from the mean



18.3 Density histogram

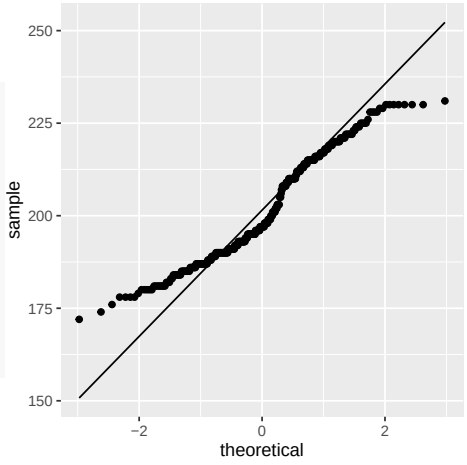
```
palmerpenguins::penguins %>%
  ggplot2::ggplot(
    aes(x = flipper_length_mm)
  ) +
  ggplot2::geom_histogram(
    aes(
      y =..density..
    )
  ) +
  ggplot2::stat_function(
    fun = dnorm,
    args = list(
      # mean and stddev
      # calculations
      # omitted here
      mean = ...,
      sd = ... ),
    colour = "black", size = 1)
```



18.4 Q-Q plot

Values against the cumulative probability of a particular distribution (in this case, *normal* distribution)

```
palmerpenguins::penguins %>%
  ggplot2::ggplot(
    aes(
      sample =
        flipper_length_mm
    )
  ) +
  ggplot2::stat_qq() +
  ggplot2::stat_qq_line()
```



18.5 Normality

Shapiro–Wilk test compares the distribution of a variable with a normal distribution having same mean and standard deviation

- If significant, the distribution is not normal
- `shapiro.test` function in `stats`
- or `normtest` values in `pastecs::stat.desc`

```
palmerpenguins::penguins %>%
  dplyr::pull(flipper_length_mm) %>%
  stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.95155, p-value = 3.54e-09
```

18.6 Significance

Most statistical tests are based on the idea of hypothesis testing

- a **null hypothesis** is set
- the data are fit into a statistical model
- the model is assessed with a **test statistic**
- the **significance** is the probability of obtaining that test statistic value by chance

The threshold to accept or reject an hypotheis is arbitrary and based on conventions (e.g., $p < .01$ or $p < .05$)

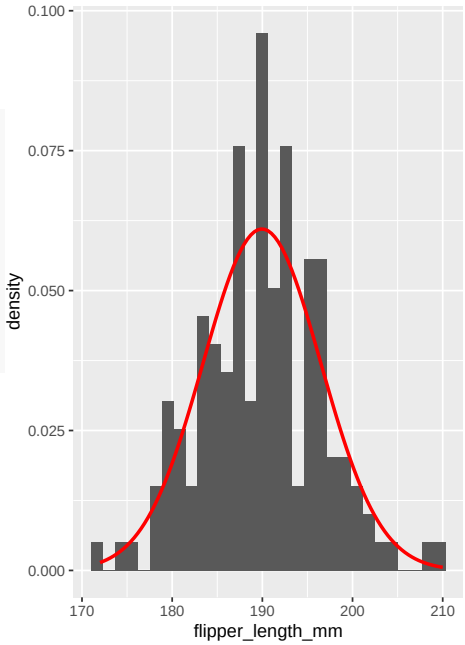
Example: The null hypotheis of the Shapiro–Wilk test is that the sample is normally distributed and $p < .01$ indicates that the probability of that being true is very low. So, the *flipper length* of penguins in the Palmer Station dataset **is not** normally distributed.

18.7 Example

The *flipper length* of **Adelie** penguins **is normally distributed**

```
palmerpenguins::penguins %>%
  filter(
    species == "Adelie"
  ) %>%
  dplyr::pull(
    flipper_length_mm
  ) %>%
  stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.99339, p-value = 0.72
```

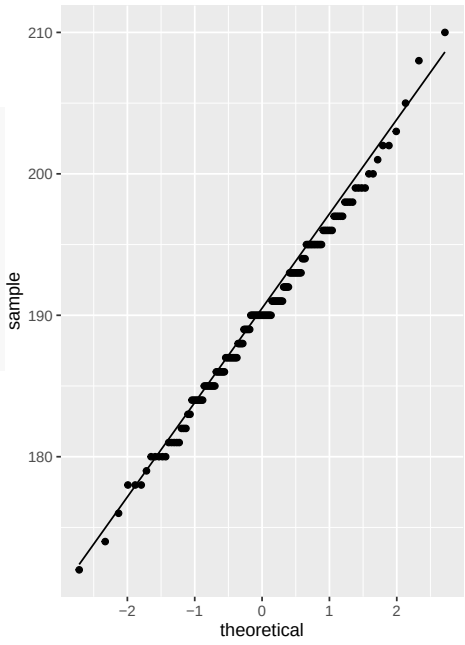


18.8 Example

The *flipper length* of **Adelie** penguins **is normally distributed**

```
palmerpenguins::penguins %>%
  filter(
    species == "Adelie"
  ) %>%
  dplyr::pull(
    flipper_length_mm
  ) %>%
  stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.99339, p-value = 0.72
```



18.9 Skewness and kurtosis

In a normal distribution, *skewness* and *kurtosis* should be **zero**

- `skewness`: **skewness** value indicates
  - positive: the distribution is skewed towards the left
  - negative: the distribution is skewed towards the right
- `kurtosis`: **kurtosis** value indicates
  - positive: heavy-tailed distribution
  - negative: flat distribution
- `skew.2SE` and `kurt.2SE`: skewness and kurtosis divided by 2 standard errors. Therefore
  - if `> 1` (or `< -1`) then the stat significant *($p < .05$)*
  - if `> 1.29` (or `< -1.29`) then stat significant *($p < .01$)*

18.10 Example

Flipper length is not normally distributed

- skewed left (skewness positive, `skew.2SE > 1.29`)
- flat distribution (kurtosis negative, `kurt.2SE < -1.29`)

```
palmerpenguins::penguins %>%
  dplyr::select(bill_length_mm, bill_depth_mm, flipper_length_mm) %>%
  pastecs::stat.desc(basic = FALSE, desc = FALSE, norm = TRUE)
```

	bill_length_mm	bill_depth_mm	flipper_length_mm
skewness	0.0526530	-0.1422086	0.3426554
skew.2SE	0.1996290	-0.5391705	1.2991456
kurtosis	-0.8931397	-0.9233523	-0.9991866
kurt.2SE	-1.6979696	-1.7554076	-1.8995781
normtest.W	0.9748548	0.9725838	0.9515451
normtest.p	0.0000112	0.0000044	0.0000000

18.11 Example

Values are instead not significant for **Adelie** penguins

- both `skew.2SE` and `kurt.2SE` between `-1` and `1`

```
palmerpenguins::penguins %>%
  filter(species == "Adelie") %>%
  dplyr::select(bill_length_mm, bill_depth_mm, flipper_length_mm) %>%
  pastecs::stat.desc(basic = FALSE, desc = FALSE, norm = TRUE)
```

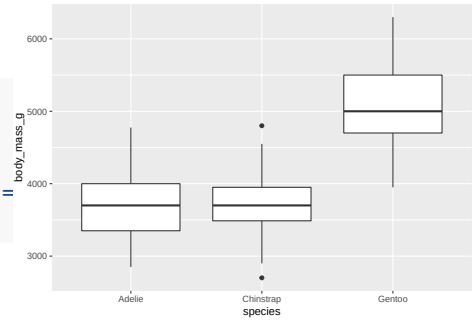
	bill_length_mm	bill_depth_mm	flipper_length_mm
skewness	0.1584764	0.3148847	0.0856093
skew.2SE	0.4014211	0.7976035	0.2168485
kurtosis	-0.2285951	-0.1361153	0.2382734
kurt.2SE	-0.2913388	-0.1734755	0.3036734
normtest.W	0.9933618	0.9846683	0.9933916
normtest.p	0.7166005	0.0924897	0.7200466

18.12 Homogeneity of variance

Levene's test for equality of variance in different levels

- If significant, the variance is different in different levels

```
library(car)
palmerpenguins::penguins %>%
  car::leveneTest(
    body_mass_g ~ species, data =
  )
```




```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value   Pr(>F)
## group   2  5.1203 0.006445 **
##       339
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

18.13 Summary

Exploring assumptions

- Normality
- Skewness and kurtosis
- Homogeneity of variance

Next: Practical session

- Data visualisation
- Descriptive statistics
- Exploring assumptions

Chapter 19

Comparing groups

19.1 Recap

Prev: Exploratory data analysis

- 301 Lecture Data visualisation
- 302 Lecture Descriptive statistics
- 303 Lecture Exploring assumptions
- 304 Practical session

Now: Comparing groups

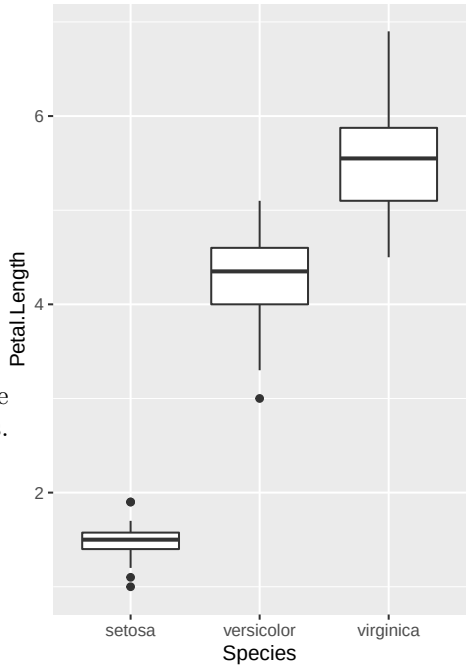
- T-test
- ANOVA

19.2 Iris

A classic R dataset

- 3 species of iris
- 50 flowers per species
- 4 measurements
  - sepal length and width
  - petal length and width

Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2), pp.179-188.



19.3 Independent T-test

Are two group means different?

- null hypothesis
  - there is **no difference** between the groups
- if *p-value* (significance) below threshold (e.g., `0.05` or `0.01`)
  - **group means are different**
- assumptions
  - normally distributed values in groups
  - homogeneity of variance of values in groups
    - if groups have different sizes
  - independence of groups
    - e.g. different conditions of an experiment

19.4 Independent T-test

Independent T-test as a general linear model

General linear model

- observation i can be predicted by a *model* (predictors)
- accounting for some amount of error

$$outcome_i = (model) + error_i$$

Independent T-test

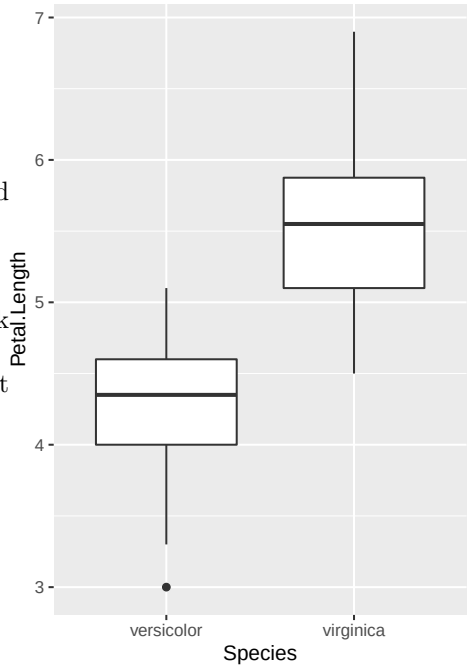
- groups is the predictor (categorical variable)
- single observation value as group mean plus error

$$outcome_i = (group\ mean) + error_i$$

19.5 Example: Petal lengths

Are the petal lengths of *versicolor* and *virginica* different?

1. Check assumptions
    1. Indipendent groups: ok
    2. normal distribution: check using Shapiro-Wilk test
    3. homogeneity of variance: not necessary
2. Run T-test
    1. `stats::t.test`



19.6 Assumptions: normality

Values are normally distributed for both groups

```
iris %>% dplyr::filter(Species == "versicolor") %>%
  dplyr::pull(Petal.Length) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.966, p-value = 0.1585
```

```
iris %>% dplyr::filter(Species == "virginica") %>%
  dplyr::pull(Petal.Length) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.96219, p-value = 0.1098
```

19.7 stats::t.test

The test is significant, the group means are different

```
iris %>%
  dplyr::filter(Species %in% c("versicolor", "virginica")) %$%
  stats::t.test(Petal.Length ~ Species)
```

```
##
##  Welch Two Sample t-test
##
## data:  Petal.Length by Species
## t = -12.604, df = 95.57, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.49549 -1.08851
## sample estimates:
## mean in group versicolor  mean in group virginica
##                    4.260                    5.552
```

How to report:

- $t(95.57) = -12.6$, $p < .01$

19.8 ANalysis Of VAriance

ANOVA is similar to the T-tests, but more than two groups

- null hypothesis
  - there is **no difference** between the groups
- if *p-value* (significance) below threshold (e.g., `0.05` or `0.01`)
  - **group means are different**
- assumptions
  - normally distributed values in groups
    - especially if groups have different sizes
  - homogeneity of variance of values in groups
    - if groups have different sizes
  - independence of groups
    - e.g. different conditions of an experiment

19.9 ANalysis Of VAriance

ANOVA as a general linear model

General linear model

- observation i can be predicted by a *model* (predictors)
- accounting for some amount of error

$$outcome_i = (model) + error_i$$

ANOVA

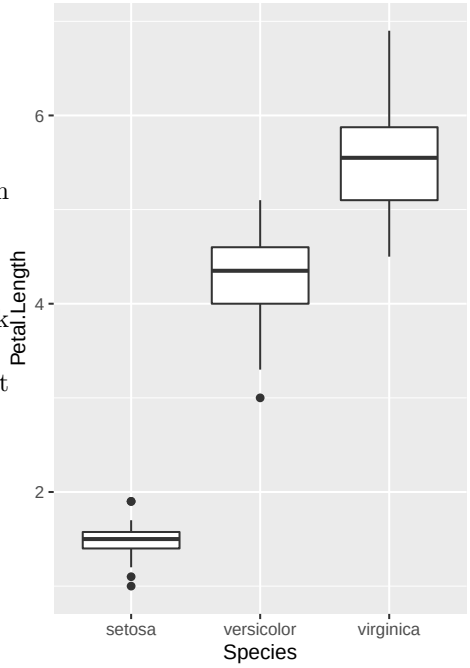
- groups is the predictor (categorical variable)
- single observation value as group mean plus error

$$outcome_i = (group\ mean) + error_i$$

19.10 Example: Petal lengths

Are the petal lengths different between all three species?

1. Check assumptions
    1. Indipendent groups: ok
    2. normal distribution: check using Shapiro-Wilk test
    3. homogeneity of variance: not necessary
2. Run ANOVA
    1. `stats::aov`



19.11 Assumptions: normality

We already checked normality for *versicolor* and *virginica*.

Are values for *setosa* normally distributed?

```
iris %>% dplyr::filter(Species == "setosa") %>%
  dplyr::pull(Petal.Length) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.95498, p-value = 0.05481
```

Values are normally distributed for all three groups

- although significance for *setosa* is borderline

19.12 stats::aov

The test is significant, the group means are different

```
iris %$%
  stats::aov(Petal.Length ~ Species) %>%
  summary()
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Species       2  437.1  218.55    1180 <2e-16 ***
## Residuals   147   27.2    0.19
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

How to report:

- $F(2, 147) = 1180.16$, $p < .01$

19.13 Summary

Comparing groups

- T-test
- ANOVA

Next: Correlation

- Pearson's r
- Spearman's rho
- Kendall's tau
- Pairs panel
- Chi-square

Chapter 20

Correlation

20.1 Recap

Prev: Comparing groups

- T-test
- ANOVA

Now: Correlation

- Pearson's r
- Spearman's rho
- Kendall's tau
- Pairs panel
- Chi-square

20.2 Correlation

Two continuous variables can be

- not related at all
- related
  - positively:
    - entities with *high values* in one
    - tend to have *high values* in the other
  - negatively:
    - entities with *high values* in one
    - tend to have *low values* in the other

Correlation is a standardised measure of covariance

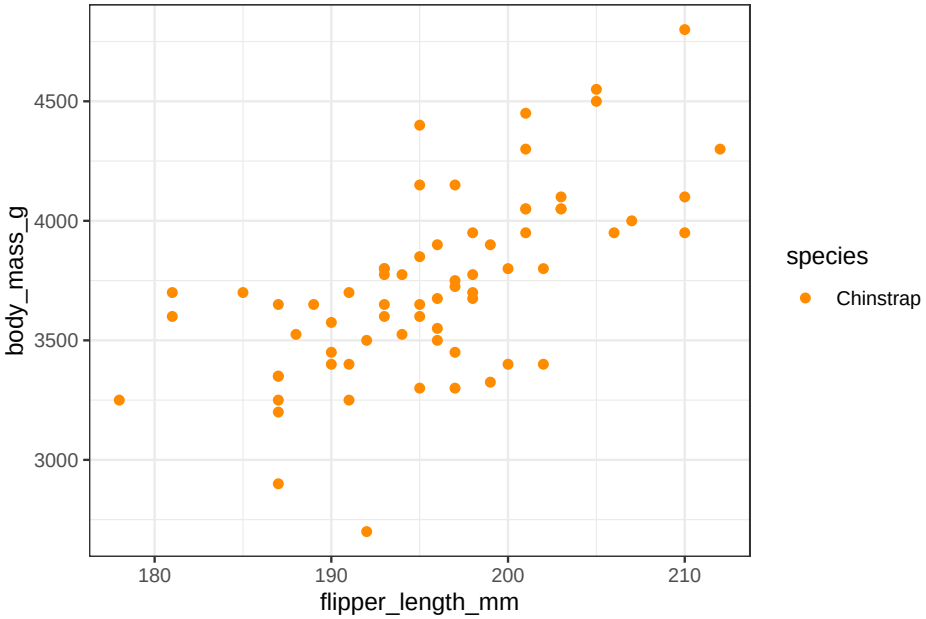
20.3 Correlation

Three different approaches

- **Pearson's r**
  - if two variables are **normally distributed**
- **Spearman's rho**
  - if two variables are **not normally distributed**
- **Kendall's tau**
  - if **not normally distributed**
  - and there are a **large number of ties**

20.4 Example

Are flipper length and body mass related in Chinstrap penguins?



20.5 Pearson's r

If two variables are **normally distributed**, use **Pearson's r**

- null hypothesis
  - there is no relationship between the variables
- assumptions
  - variables are normally distributed

The square of the correlation value indicates the percentage of shared variance

20.6 Assumptions: normality

Flipper length and body mass are normally distributed in Chinstrap penguins

```
palmerpenguins::penguins %>%
  dplyr::filter(species == "Chinstrap") %>%
  dplyr::pull(flipper_length_mm) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.98891, p-value = 0.8106
```

```
palmerpenguins::penguins %>%
  dplyr::filter(species == "Chinstrap") %>%
  dplyr::pull(body_mass_g) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.98449, p-value = 0.5605
```

20.7 stats::cor.test

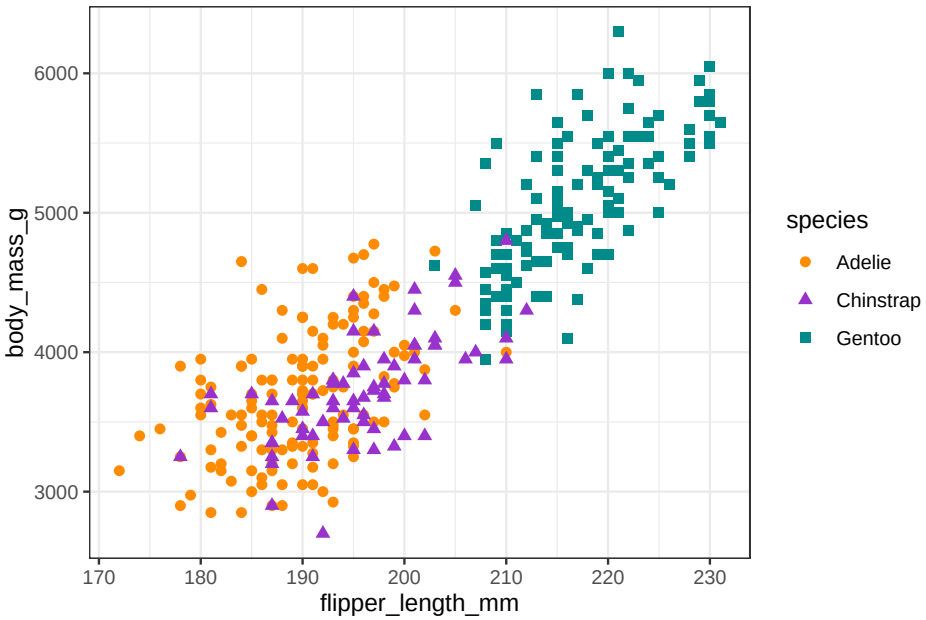
- Flipper length and body mass are related
  - `p-value < 0.01`
- sharing 41.2% of variance
  - `0.642 ^ 2 = 0.412`

```
palmerpenguins::penguins %>%
  dplyr::filter(species == "Chinstrap") %$%
  stats::cor.test(flipper_length_mm, body_mass_g)
```

```
##
##  Pearson's product-moment correlation
##
## data:  flipper_length_mm and body_mass_g
## t = 6.7947, df = 66, p-value = 3.748e-09
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4759352 0.7632368
## sample estimates:
##       cor
## 0.6415594
```

20.8 Example

But, are flipper length and body mass related in penguins (without considering species as separated groups)?



20.9 Assumptions: normality

Flipper length and body mass are not normally distributed when all penguins are taken into account as a single group

```
palmerpenguins::penguins %>%
  dplyr::pull(flipper_length_mm) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.95155, p-value = 3.54e-09
```

```
palmerpenguins::penguins %>%
  dplyr::pull(body_mass_g) %>% stats::shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.95921, p-value = 3.679e-08
```

20.10 Spearman's rho

If two variables are **not normally distrib.**, use **Spearman's rho**

- null hypothesis
  - there is no relationship between the variables

- non-parametric
  - uses full dataset to calculate the statistics
  - rather than estimate key parameters of distributions from data
- based on rank difference
- assumptions
  - ties are uncommon

The square of the correlation value indicates the percentage of shared variance

20.11 stats::cor.test (method = "spearman")

- Flipper length and body mass are related
  - `p-value < 0.01`
- sharing 70.6% of variance
  - `0.84 ^ 2 = 0.706`

```
palmerpenguins::penguins %$%
  stats::cor.test(flipper_length_mm, body_mass_g, method = "spearman")
```

```
##
##  Spearman's rank correlation rho
##
## data:  flipper_length_mm and body_mass_g
## S = 1066875, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.8399741
```

20.12 Correlation with ties

Spearman's rho

- `Cannot compute exact p-value with ties`

```
palmerpenguins::penguins %$%
  stats::cor.test(flipper_length_mm, body_mass_g, method = "spearman")
```

```
## Warning in cor.test.default(flipper_length_mm, body_mass_g, method = "spearman"):
##  Cannot compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  flipper_length_mm and body_mass_g
## S = 1066875, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.8399741
```

20.13 Kendall's tau

If two variables are **not normally distributed** and there are **many ties**, use **Kendall's tau**

- null hypothesis
  - there is no relationship between the variables
- non-parametric
- based on rank difference
- no assumptions
- less *powerful*
  - even if there is a relationship, significance might be high

The square of the correlation value indicates the percentage of shared variance

20.14 stats::cor.test (method = "kendall")

- Flipper length and body mass are related
  - `p-value < 0.01`
- sharing 43.6% of variance
  - 0.66 ^ 2 = 0.436

```
palmerpenguins::penguins %$%
  stats::cor.test(flipper_length_mm, body_mass_g, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  flipper_length_mm and body_mass_g
## z = 17.898, p-value < 2.2e-16
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.6604675
```

20.15 psych::pairs.panels

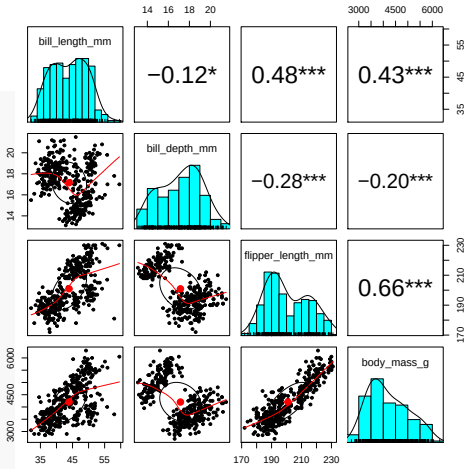
Combining:

- histograms
- scatter plots
- correlations

```
library(psych)

palmerpenguins::penguins %>%
  dplyr::select(
    bill_length_mm,
    bill_depth_mm,
    flipper_length_mm,
    body_mass_g
  ) %>%
  psych::pairs.panels(
    method = "kendall",
    stars = TRUE
  )

## Signif.: 0 '***' 0.001 '**' 0.01
## 0.01 '*' 0.05 0.05 '.' 0.1 ' ' 1
```



20.16 Chi-squre

How to test the correlation between two **categorical** variables?

Chi-square test:

- null hypothesis
  - there is no relationship between the variables
- non-parametric
- based on cross-tabulated expected counts
- no assumptions

```
library(gmodels)
palmerpenguins::penguins %$%
  gmodels::CrossTable(
    island, species, chisq = TRUE, expected = TRUE, prop.c = FALSE,
    prop.t = FALSE, prop.chisq = FALSE, sresid = TRUE, format = "SPSS")
```

20.17 gmodels::CrossTable

There is a relationship (`p-value < 0.01`), different islands have different amounts of penguins from different species

```
##
##    Cell Contents
## |-------------------------|
## |                   Count |
## |         Expected Values |
## |             Row Percent |
## |            Std Residual |
## |-------------------------|
```

```
##
## Total Observations in Table:  344
##
##              | species
##       island |    Adelie  | Chinstrap  |    Gentoo  | Row Total |
## -------------|------------|------------|------------|-----------|
##       Biscoe |        44  |         0  |       124  |      168  |
##              |   74.233  |   33.209  |   60.558  |           |
##              |   26.190% |    0.000% |   73.810% |   48.837% |
##              |   -3.509  |   -5.763  |    8.152  |           |
## -------------|------------|------------|------------|-----------|
##        Dream |        56  |        68  |         0  |      124  |
##              |   54.791  |   24.512  |   44.698  |           |
##              |   45.161% |   54.839% |    0.000% |   36.047% |
##              |    0.163  |    8.784  |   -6.686  |           |
## -------------|------------|------------|------------|-----------|
##    Torgersen |        52  |         0  |         0  |       52  |
##              |   22.977  |   10.279  |   18.744  |           |
##              |  100.000% |    0.000% |    0.000% |   15.116% |
##              |    6.055  |   -3.206  |   -4.329  |           |
## -------------|------------|------------|------------|-----------|
## Column Total |       152  |        68  |       124  |      344  |
## -------------|------------|------------|------------|-----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  299.5503     d.f. =  4     p =  1.354574e-63
##
##
##
##        Minimum expected frequency: 10.27907
```

20.18 Summary

Correlation

- Pearson's r
- Spearman's rho
- Kendall's tau
- Pairs plot
- Chi-square

Next: Data transformations

- Z-scores
- Logarithmic transformations
- Inverse hyperbolic sine transformations

Chapter 21

Data transformations

21.1 Recap

Prev: Correlation

- Pearson's r
- Spearman's rho
- Kendall's tau
- Pairs panel
- Chi-square

Now: Data transformations

- Z-scores
- Logarithmic transformations
- Inverse hyperbolic sine transformations

21.2 Z-scores

Transform the values as relative to

- the distribution's mean
- and standard deviation
- the z-score of a value i-th x_i is calculated as below, where
  - μ is the distribution's mean
  - σ is the distribution's standard deviation

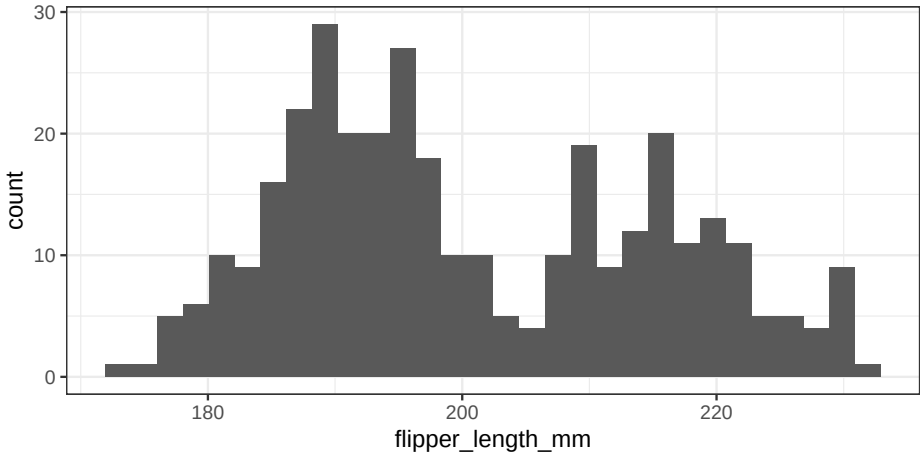
$$z_i = \frac{x_i - \mu}{\sigma}$$

Commonly used to render two variables easier to compare

21.3 Example

Distribution of flipper lengths in Palmer's penguins

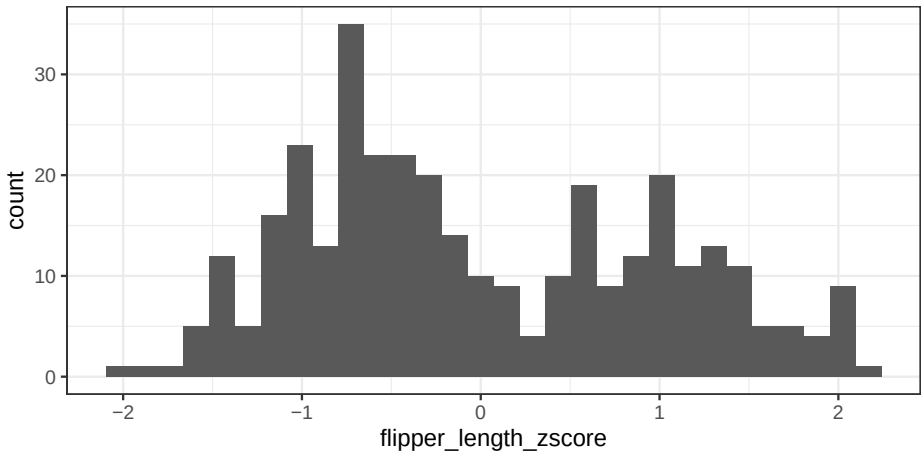
```
palmerpenguins::penguins %>%
  ggplot2::ggplot(aes(x = flipper_length_mm)) +
  ggplot2::geom_histogram() + ggplot2::theme_bw()
```



21.4 base::scale

Distribution of **zscores** derived from flipper lengths

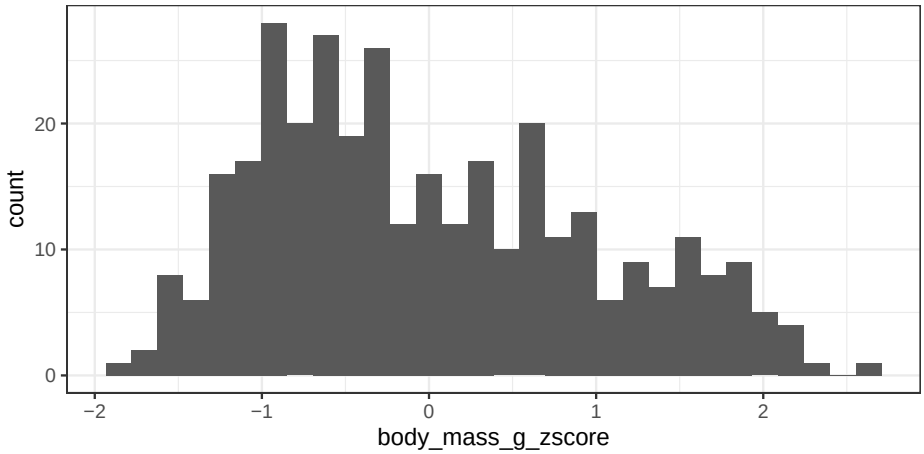
```
palmerpenguins::penguins %>%
  dplyr::mutate(flipper_length_zscore = scale(flipper_length_mm)) %>%
  ggplot2::ggplot(aes(x = flipper_length_zscore)) +
  ggplot2::geom_histogram() + ggplot2::theme_bw()
```



21.5 base::scale

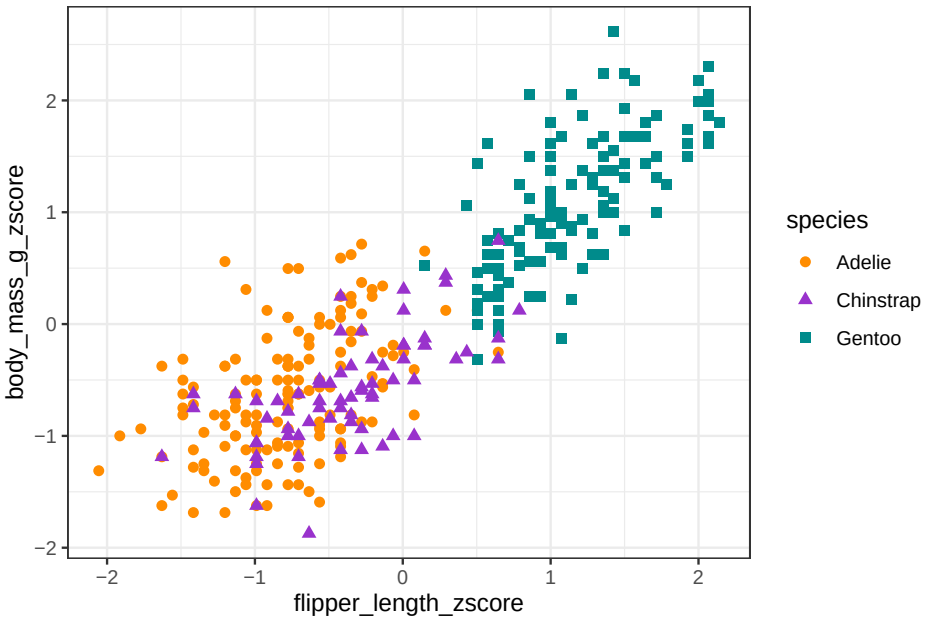
Distribution of **zscores** derived from body mass

```
palmerpenguins::penguins %>%
  dplyr::mutate(body_mass_g_zscore = scale(body_mass_g)) %>%
  ggplot2::ggplot(aes(x = body_mass_g_zscore)) +
  ggplot2::geom_histogram() + ggplot2::theme_bw()
```



21.6 Comparison

But, are flipper length and body mass related in penguins (without considering species as separated groups)?



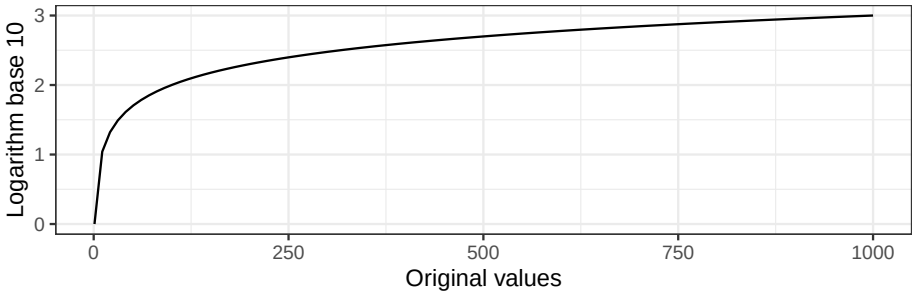
21.7 Log transformation

Logarithmic transformations are useful to *"un-skew"* variables

Common approaches include:

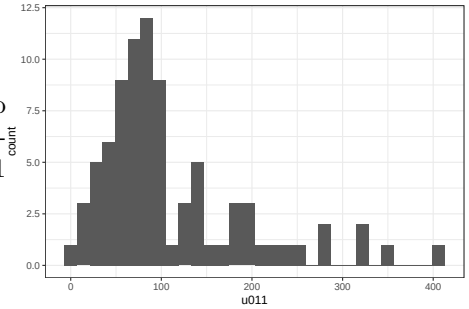
- natural logarithm (`log`)
- binary logarithm (`log2`)
- logarithm base 10 (`log10`)

Only possible on values `> 0`



21.8 Example

The number of residents aged 20 to 24 (`u011`) in the areas of Leicester described as *"Cosmopolitans"* by the 2011 Output Area Classification is skewed

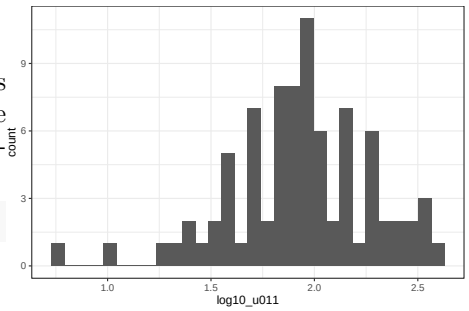


	u011
skewness	1.521
skew.2SE	2.879
kurtosis	2.089
kurt.2SE	1.999
normtest.W	0.847
normtest.p	0.000

21.9 Example

However, it's logarithm base 10 is normally distributed, thus it can be used with tests requiring normally distributed values

```
mutate(log10_u011 = log10(u011))
```



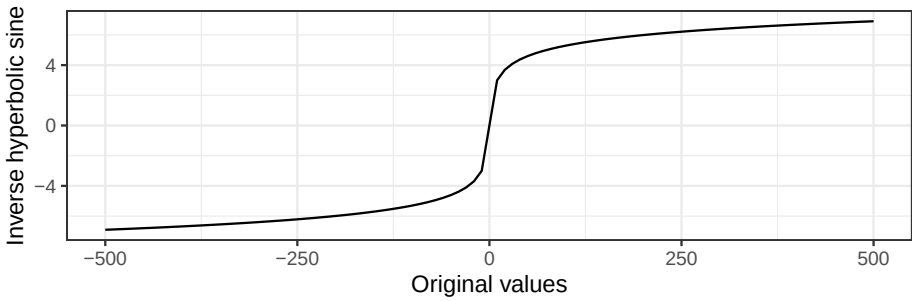
	log10_u011
skewness	-0.504
skew.2SE	-0.953
kurtosis	0.872
kurt.2SE	0.834
normtest.W	0.976
normtest.p	0.118

21.10 Inverse hyperbolic sine

Inverse hyperbolic sine transformations are useful to *"un-skew"* variables

- similar to logarithmic transformations

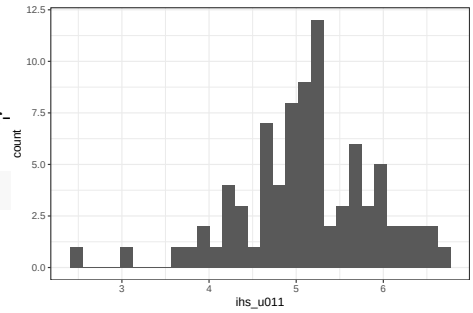
- defined on all values
- in R: `asinh`



21.11 Example

The Inverse hyperbolic sine is also normally distributed

```
mutate(ihs_u011 = asinh(u011))
```

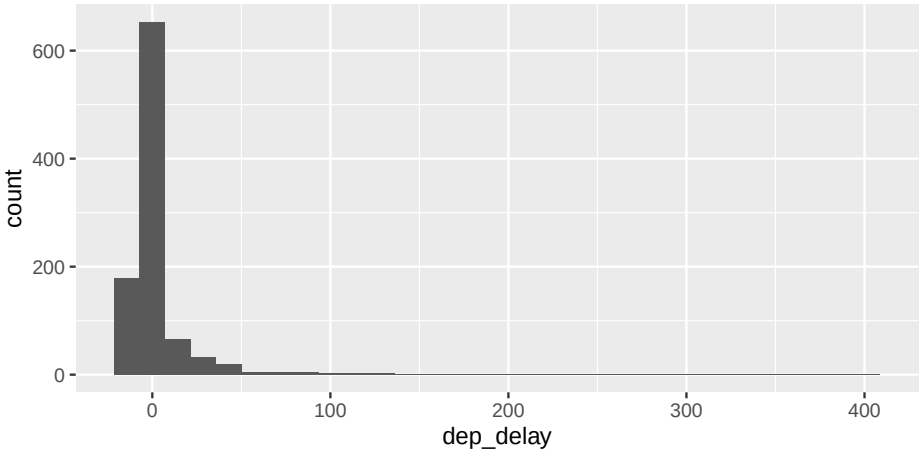


	log10_u011
skewness	-0.504
skew.2SE	-0.953
kurtosis	0.872
kurt.2SE	0.834
normtest.W	0.976
normtest.p	0.118

21.12 Example

Logarithmic transformation can't be applied to arrival delays in the New York City 2013 flights dataset

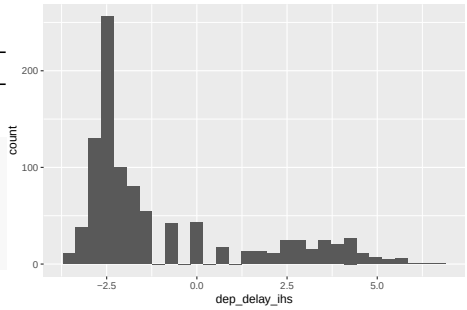
- skewed towards left
- but there are values lower or equal to zero



21.13 Example

Inverse hyperbolic sine can still be applied. Here it partially unskews the distribution

```
mutate(
  dep_delay_ihs =
    asinh(dep_delay)
)
```



	dep_delay_ihs
skewness	1.273
skew.2SE	8.122
kurtosis	0.242
kurt.2SE	0.773
normtest.W	0.778
normtest.p	0.000

21.14 Summary

Data transformations

- Z-scores
- Logarithmic transformations
- Inverse hyperbolic sine transformations

Next: Practical session

- Comparing means

- Correlation

Chapter 22

Simple Regression

22.1 Recap

Prev: Comparing data

- 311 Lecture Comparing groups
- 312 Lecture Correlation
- 313 Lecture Data transformations
- 314 Practical session

Now: Simple Regression

- Regression
- Ordinary Least Squares
- Interpretation
- Checking assumptions

22.2 Regression analysis

Regression analysis is a supervised machine learning approach

Special case of the general linear model

$$outcome_i = (model) + error_i$$

Predict (estimate) value of one outcome (dependent) variable as

- one predictor (independent) variable: **simple / univariate**

$$Y_i = (b_0 + b_1 * X_{i1}) + \epsilon_i$$

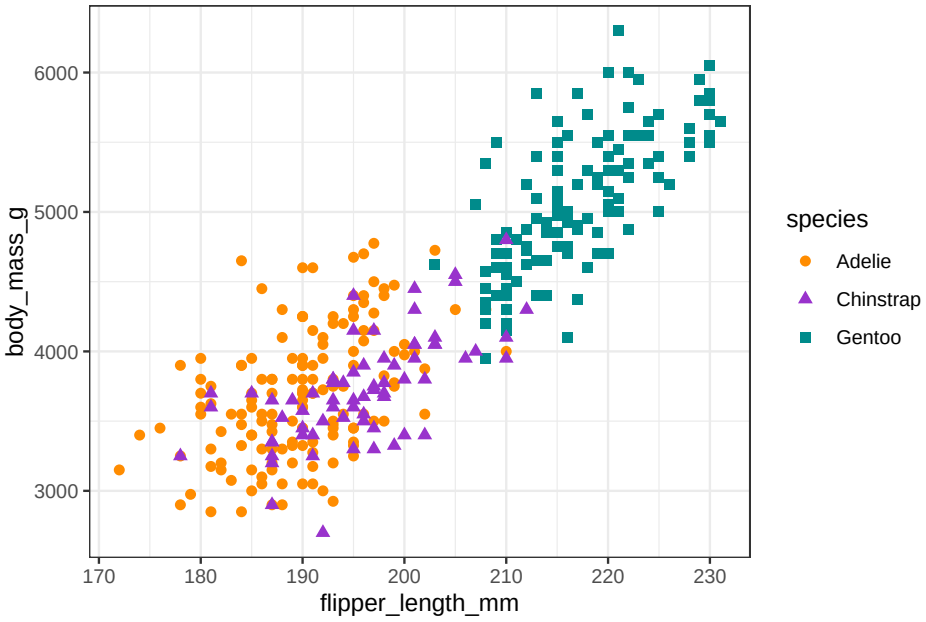
- more predictor (independent) variables: **multiple / multivar.**

$$Y_i = (b_0 + b_1 * X_{i1} + b_2 * X_{i2} + \cdots + b_M * X_{iM}) + \epsilon_i$$

22.3 Example

Can we predict a penguin's body mass from flipper length?

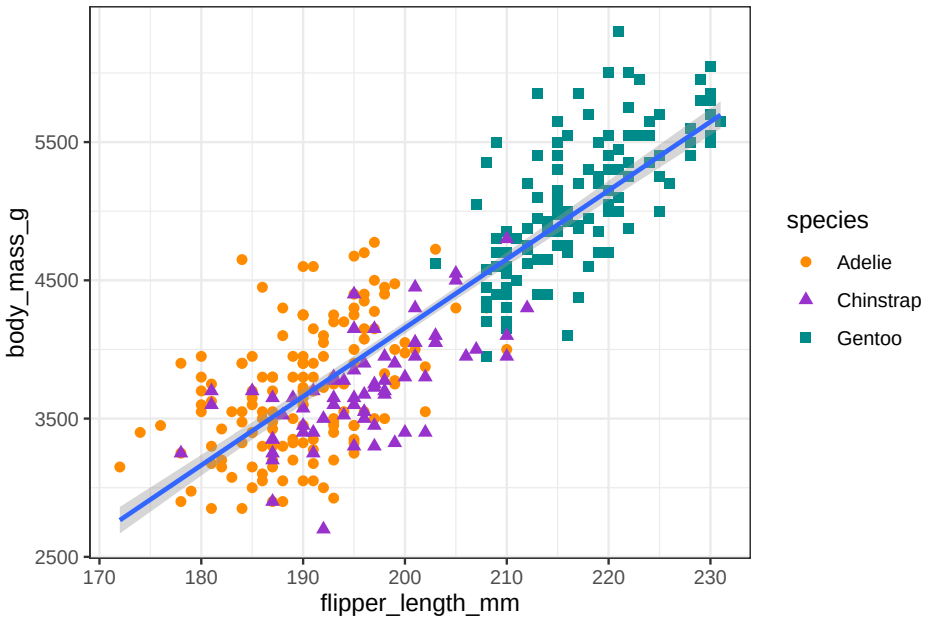
$$body\ mass_i = (b_0 + b_1 * flipper\ length_i) + \epsilon_i$$



22.4 Example

Can we predict a penguin's body mass from flipper length?

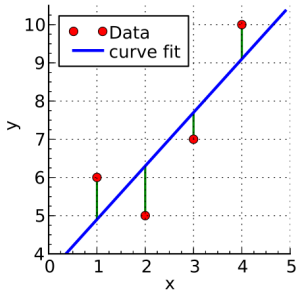
$$body\ mass_i = (b_0 + b_1 * flipper\ length_i) + \epsilon_i$$



22.5 Least squares

Least squares is the most commonly used approach to generate a regression model
The model fits a line

- to **minimise** the squared values of the **residuals** (errors)
- that is squared difference between
  - **observed values**
  - **model**



by Krishnavedala via Wikimedia Commons,CC-BY-SA-3.0

$$residual_i = observed_i - model_i$$

$$deviation = \sum_i (observed_i - model_i)^2$$

22.6 Assumptions

- **Linearity**
  - the relationship is actually linear
- **Normality** of residuals

  - standard residuals are normally distributed with mean `0`
- **Homoscedasticity** of residuals
  - at each level of the predictor variable(s) the variance of the standard residuals should be the same (*homo-scedasticity*) rather than different (*hetero-scedasticity*)
- **Independence** of residuals
  - adjacent standard residuals are not correlated

22.7 stats::lm

```
bm_fl_model <-
  palmerpenguins::penguins %>%
  dplyr::filter(!is.na(body_mass_g) | !is.na(flipper_length_mm)) %$%
  stats::lm(body_mass_g ~ flipper_length_mm)

bm_fl_model %>%
  summary()
```

```
##
## Call:
## stats::lm(formula = body_mass_g ~ flipper_length_mm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1058.80  -259.27   -26.88   247.33  1288.69
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -5780.831    305.815  -18.90   <2e-16 ***
## flipper_length_mm    49.686      1.518   32.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 394.3 on 340 degrees of freedom
## Multiple R-squared:  0.759,Adjusted R-squared:  0.7583
## F-statistic:  1071 on 1 and 340 DF,  p-value: < 2.2e-16
```

22.8 Overall fit

The output indicates

- **p-value: < 2.2e-16**: $p < .01$ the model is significant
  - derived by comparing **F-statistic** (1070.74) to F distribution having specified degrees of freedom (1, 340)
  - Report as: F(1, 340) = 1070.74
- **Adjusted R-squared: 0.7583**:
  - flipper length can account for 75.83% variation in body mass
- **Coefficients**
  - Intercept estimate -5780.8314 is significant
  - `flipper_length_mm` (slope) estimate 49.6856 is significant

22.9 Outliers and influential cases

```
penguins_output <-
  palmerpenguins::penguins %>%
  dplyr::filter(!is.na(body_mass_g) | !is.na(flipper_length_mm)) %>%
  mutate(
    model_stdres = bm_fl_model %>% stats::rstandard(),
    model_cook_dist = bm_fl_model %>% stats::cooks.distance()
  )

penguins_output %>%
  dplyr::select(body_mass_g, model_stdres, model_cook_dist) %>%
  dplyr::filter(abs(model_stdres) > 2.58 | model_cook_dist > 1)
```

```
## # A tibble: 4 x 3
##   body_mass_g model_stdres model_cook_dist
##         <int>        <dbl>           <dbl>
## 1        4650         3.28          0.0388
## 2        5850         2.66          0.0182
## 3        6300         2.80          0.0353
## 4        2700        -2.69          0.0149
```

No influential cases (Cook's distance `> 1`) but there are a handful of outliers (4 abs std res `> 2.58`)

22.10 Checking assumptions: normality

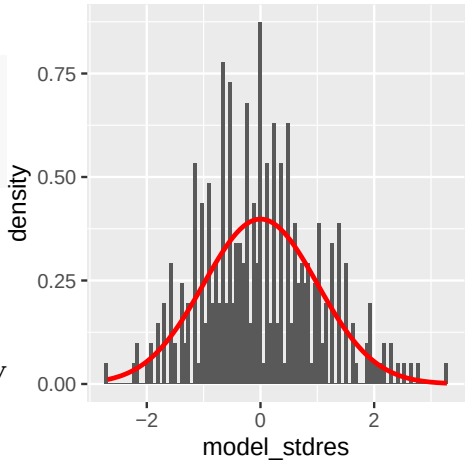
Shapiro-Wilk test for normality of standard residuals,

- robust models: should be **not** significant

```
penguins_output %$%
  stats::shapiro.test(
    model_stdres
  )
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_stdres
## W = 0.99295, p-value = 0.1085
```

Standard residuals are normally distributed!



22.11 Checking assumpt.: homoscedasticity

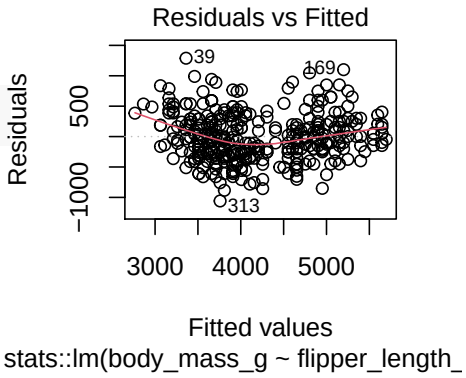
Breusch-Pagan test for homoscedasticity of standard residuals

- robust models: should be **not** significant

```
bm_fl_model %>%
  lmtest::bptest()
```

```
##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 2.1579, df = 1, p-value = 0.1418
```

Standard residuals are homoscedastic!



22.12 Checking assumptions: independence

Durbin-Watson test for the independence of residuals

- robust models: statistic should be close to 2 (advised between 1 and 3) and **not** significant

```
bm_fl_model %>%
  lmtest::dwtest()
```

```
##
##  Durbin-Watson test
##
## data:  .
## DW = 2.1896, p-value = 0.9572
## alternative hypothesis: true autocorrelation is greater than 0
```

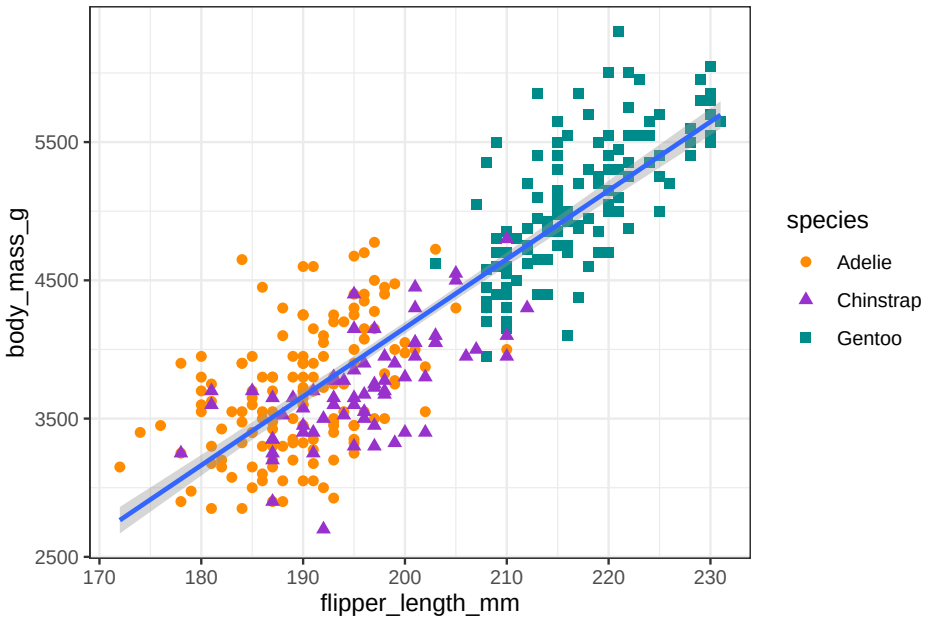
Standard residuals are independent!

Note: the result depends on the order of the data.

22.13 Example

Yes, we can predict a penguin's body mass from flipper length!

$$body\ mass_i = (-5780.83 + 49.69 * flipper\ length_i) + \epsilon_i$$



22.14 Summary

Simple Regression

- Regression
- Ordinary Least Squares
- Interpretation
- Checking assumptions

Next: Multiple Regression

- Multiple regression
- Interpretation
- Checking assumptions

Chapter 23

Multiple Regression

23.1 Recap

Prev: Simple Regression

- Regression
- Ordinary Least Squares
- Interpretation
- Checking assumptions

Now: Multiple Regression

- Multiple regression
- Interpretation
- Checking assumptions

23.2 Multiple regression

Regression analysis is a supervised machine learning approach

Special case of the general linear model

$$outcome_i = (model) + error_i$$

Predict (estimate) value of one outcome (dependent) variable as

- one predictor (independent) variable: **simple / univariate**

$$Y_i = (b_0 + b_1 * X_{i1}) + \epsilon_i$$

- more predictor (independent) variables: **multiple / multivar.**

$$Y_i = (b_0 + b_1 * X_{i1} + b_2 * X_{i2} + \cdots + b_M * X_{iM}) + \epsilon_i$$

23.3 Assumptions

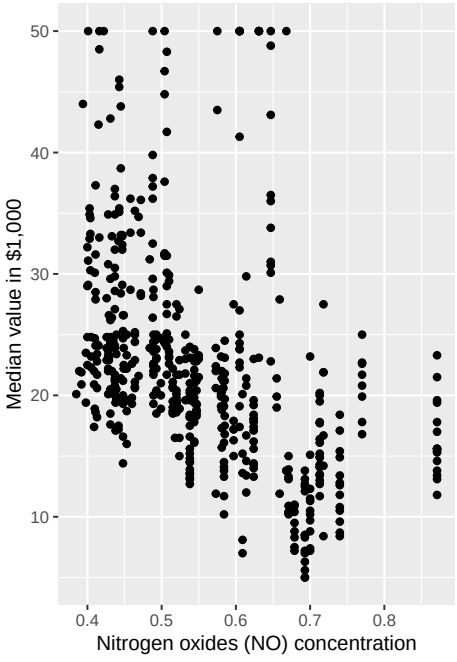
- **Linearity**
  - the relationship is actually linear
- **Normality** of residuals
  - standard residuals are normally distributed with mean `0`
- **Homoscedasticity** of residuals
  - at each level of the predictor variable(s) the variance of the standard residuals should be the same (*homo-scedasticity*) rather than different (*hetero-scedasticity*)
- **Independence** of residuals
  - adjacent standard residuals are not correlated
- When more than one predictor: **no multicollinearity**
  - if two or more predictor variables are used in the model, each pair of variables not correlated

23.4 Boston housing

A classic R dataset

- price of houses in Boston
- in relation to:
  - house characteristics
  - neighborhood
  - air quality

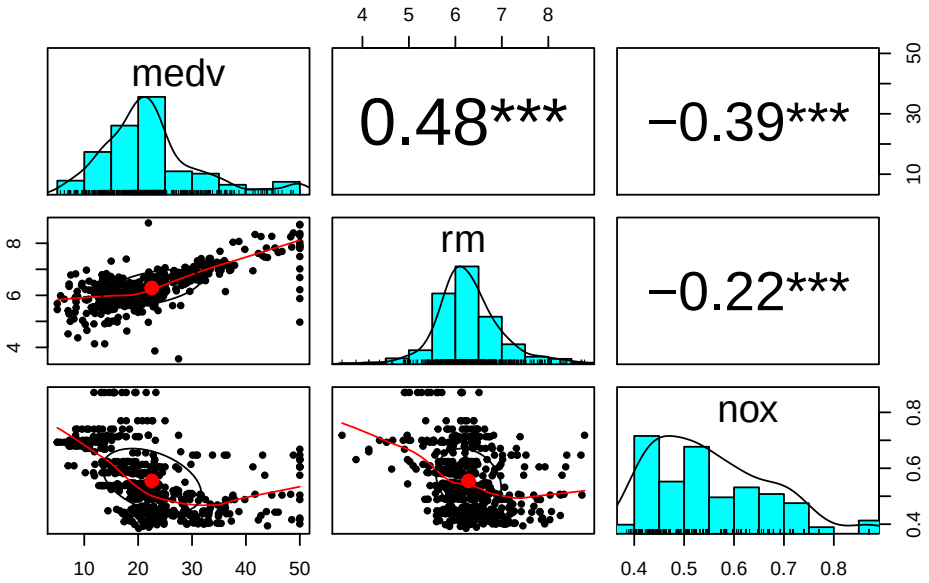
Harrison, D., and D. L. Rubinfeld. 1978. Hedonic Housing Prices and the Demand for Clean Air. Journal of Environmental Economics and Management 5 (1): 81–102.



23.5 Example

Can we predict price based on number of rooms and air quality?

$$house\ value_i = (b_0 + b_1 * rooms_i + b_2 * NO\ conc_i) + \epsilon_i$$



23.6 stats::lm

```
MASS::Boston %$%
  stats::lm(medv ~ rm + nox) ->
  medv_model

medv_model %>%
  summary()
```

```
##
## Call:
## stats::lm(formula = medv ~ rm + nox)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.889  -3.287  -0.636   2.518  39.638
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.2059     3.3393  -5.452 7.82e-08 ***
## rm            8.1567     0.4173  19.546  < 2e-16 ***
## nox         -18.9706     2.5304  -7.497 2.97e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 6.281 on 503 degrees of freedom
## Multiple R-squared:  0.5354,Adjusted R-squared:  0.5336
## F-statistic: 289.9 on 2 and 503 DF,  p-value: < 2.2e-16
```

23.7 Overall fit

The output indicates

- **p-value: < 2.2e-16**: $p < .01$ the model is significant
  - derived by comparing **F-statistic** to F distribution 289.87 having specified degrees of freedom (2, 503)
  - Report as: F(2, 503) = 289.87
- **Adjusted R-squared: 0.5336**:
  - number of rooms and air quality can account for 53.36% variation in house prices
- **Coefficients**
  - Intercept estimate -18.2059 is significant
  - `rm` (slope) estimate -18.9706 is significant
  - `nox` (slope) estimate 8.1567 is significant

23.8 Standardised coefficients

- Indicate amount of change
  - in the outcome variable
  - per one standard deviation change in the predictor variable
- Can also be interpreted as importance of predictor

```
medv_model %>%
  lm.beta::lm.beta()
```

```
##
## Call:
## stats::lm(formula = medv ~ rm + nox)
##
## Standardized Coefficients::
## (Intercept)          rm         nox
##   0.0000000   0.6231316  -0.2390178
```

23.9 Confindence intervals

- Coefficients' 95% confidence intervals
  - can be interpreted as interval containing true coefficient values
  - good models should result in small intervals

```
medv_model %>%
  stats::confint()
```

```
##                  2.5 %     97.5 %
## (Intercept) -24.76666 -11.645106
## rm            7.33676   8.976551
## nox         -23.94200 -13.999233
```

23.10 Outliers and influential cases

```
MASS::Boston %>%
  mutate(
    model_stdres = medv_model %>% stats::rstandard(),
    model_cook_dist = medv_model %>% stats::cooks.distance()
  ) ->
  boston_output

boston_output %>%
  dplyr::select(medv, model_stdres, model_cook_dist) %>%
  dplyr::filter(abs(model_stdres) > 2.58 | model_cook_dist > 1)
```

```
##    medv model_stdres model_cook_dist
## 1  50.0     2.975511      0.02911770
## 2  21.9    -2.907328      0.11856922
## 3  27.5     4.899644      0.26329818
## 4  23.1     3.511137      0.10891047
## 5  50.0     6.339016      0.12065579
## 6  50.0     4.094574      0.02308311
## 7  50.0     3.665060      0.02786674
## 8  50.0     4.699311      0.02109333
## 9  50.0     5.257825      0.03746931
## 10  7.5    -2.672594      0.01576071
```

No influential cases (Cook's distance `> 1`) but there are many outliers (7 abs std res `> 3.29`, 2% `> 2.58`)

23.11 Checking assumptions: normality

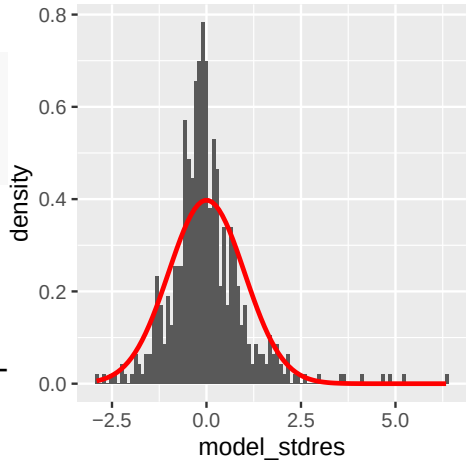
Shapiro-Wilk test for normality of standard residuals,

- robust models: should be **not** significant

```
boston_output %$%
  stats::shapiro.test(
    model_stdres
  )
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_stdres
## W = 0.8979, p-value < 2.2e-16
```

Standard residuals are NOT normally distributed



23.12 Checking assumpt.: homoscedasticity

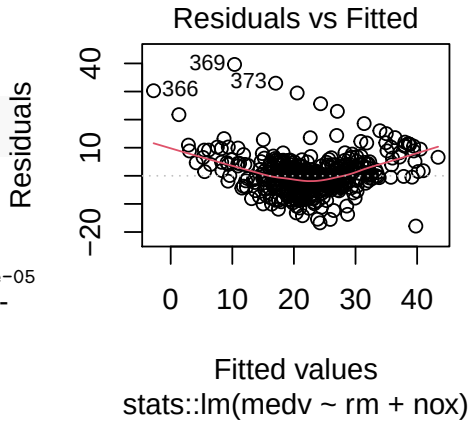
Breusch-Pagan test for homoscedasticity of standard residuals

- robust models: should be **not** significant

```
medv_model %>%
  lmtest::bptest()
```

```
##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 22.342, df = 2, p-value = 1.407e-05
```

Standard residuals are NOT homoscedastic



23.13 Checking assumptions: independence

Durbin-Watson test for the independence of residuals

- robust models: statistic should be close to 2 (advised between 1 and 3) and **not** significant

```
medv_model %>%
  lmtest::dwtest()
```

```
##
##  Durbin-Watson test
##
```

```
## data:  .
## DW = 0.68451, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

Standard residuals are NOT independent

Note: the result depends on the order of the data.

23.14 Checking assumpt.: multicollinearity

Checking the variance inflation factor (VIF)

- robust models should have no multicollinearity:
  - largest VIF should be lower than 10 or the average VIF should not be greater than 1

```
library(car)

medv_model %>%
  car::vif()
```

```
##       rm      nox
## 1.100495 1.100495
```

There is no multicollinearity

23.15 Example

No, we can't predict house prices based only on number of rooms and air quality.

- predictors are statistically significant
- but model is not robust, as it doesn't satisfy most assumptions
  - Standard residuals are NOT normally distributed
  - Standard residuals are NOT homoscedastic
  - Standard residuals are NOT independent
  - (although there is no multicollinearity)

We seem to be on the right path, but something is missing...

23.16 Summary

Multiple Regression

- Multiple regression
- Interpretation
- Checking assumptions

Next: Comparing regression models

- Information criteria

- Model difference
- Systematic variable choice

Chapter 24

Comparing regression models

24.1 Recap

Prev: Multiple Regression

- Multiple regression
- Interpretation
- Checking assumptions

Now: Comparing regression models

- Information criteria
- Model difference
- Stepwise selection
- Validation

24.2 Multiple regression

Regression analysis is a supervised machine learning approach

Special case of the general linear model

$$outcome_i = (model) + error_i$$

Predict (estimate) value of one outcome (dependent) variable as

- one predictor (independent) variable: **simple / univariate**

$$Y_i = (b_0 + b_1 * X_{i1}) + \epsilon_i$$

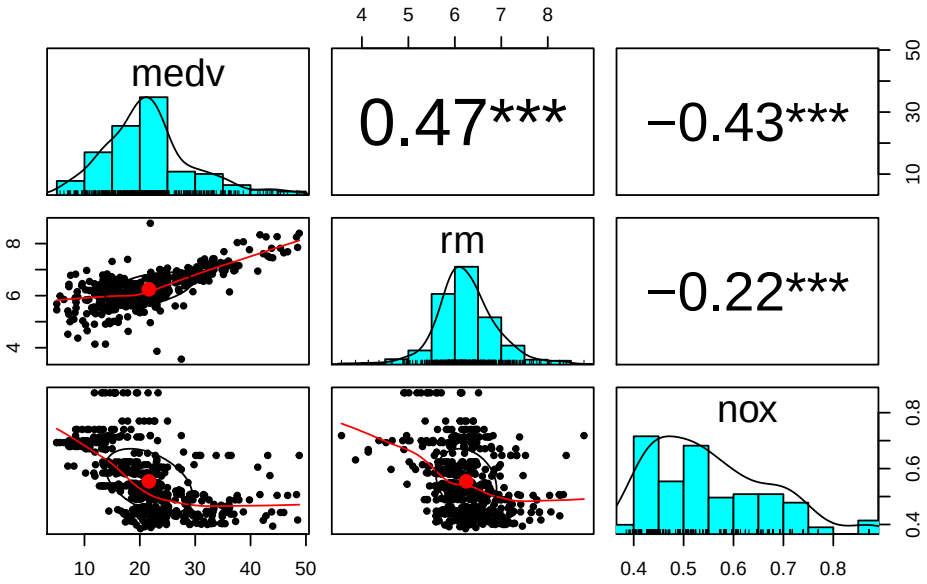
- more predictor (independent) variables: **multiple / multivar.**

$$Y_i = (b_0 + b_1 * X_{i1} + b_2 * X_{i2} + \cdots + b_M * X_{iM}) + \epsilon_i$$

24.3 Example

Can we predict price based on number of rooms and air quality?

$$house\ value_i = (b_0 + b_1 * rooms_i + b_1 * NO\ conc_i) + \epsilon_i$$



24.4 stats::lm

```
MASS::Boston %>% filter(medv < 50) %$%
  stats::lm(medv ~ rm + nox) ->
  medv_model1

medv_model1 %>%
  summary()
```

```
##
## Call:
## stats::lm(formula = medv ~ rm + nox)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.0255  -2.8916  -0.3794   2.6363  28.2653
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -9.1677     2.9516  -3.106  0.00201 **
## rm            6.9550     0.3763  18.481  < 2e-16 ***
## nox         -22.7914     2.1064 -10.820  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.145 on 487 degrees of freedom
## Multiple R-squared:  0.5739,Adjusted R-squared:  0.5721
## F-statistic:   328 on 2 and 487 DF,  p-value: < 2.2e-16
```

24.5 Checking assumptions

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.96913, p-value = 1.222e-08

##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 38.776, df = 2, p-value = 3.8e-09

##
##  Durbin-Watson test
##
## data:  .
## DW = 0.80228, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0

##       rm      nox
## 1.116167 1.116167
```

24.6 Model 1

No, we can't predict house prices based only on number of rooms and air quality.

- predictors are statistically significant
- but model is not robust, as it doesn't satisfy most assumptions
  - Standard residuals are NOT normally distributed
  - Standard residuals are NOT homoscedastic
  - Standard residuals are NOT independent
  - (although there is no multicollinearity)

We seem to be on the right path, but something is missing...

24.7 stats::lm

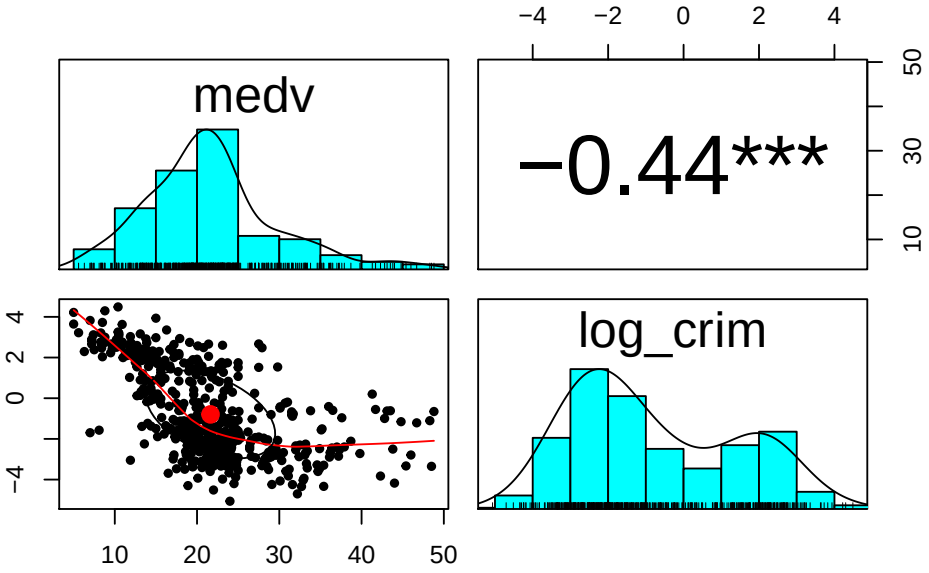
```
MASS::Boston %>% filter(medv < 50) %$%
  stats::lm(medv ~ rm + nox + ptratio + log(crim)) ->
  medv_model2

medv_model2 %>%
  summary()
```

```
##
## Call:
## stats::lm(formula = medv ~ rm + nox + ptratio + log(crim))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.1957  -2.7435  -0.1094   2.2879  26.9646
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  10.4676     4.0846   2.563 0.010687 *
## rm            5.9404     0.3421  17.366  < 2e-16 ***
## nox         -13.0203     2.9408  -4.427 1.18e-05 ***
## ptratio      -1.0344     0.1107  -9.345  < 2e-16 ***
## log(crim)    -0.5558     0.1676  -3.316 0.000983 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.526 on 485 degrees of freedom
## Multiple R-squared:  0.6715,Adjusted R-squared:  0.6688
## F-statistic: 247.9 on 4 and 485 DF,  p-value: < 2.2e-16
```

24.8 Logarithmic transformations

- 10% change in criminality score leads to
  - $log(110/100) * b_{crim} = 0.0953 * b_{crim}$ change
  - `0.0953 * -0.5558 = 0.0529`



24.9 Checking assumptions

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.95777, p-value = 1.231e-10
```

```
##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 14.78, df = 4, p-value = 0.005179
```

```
##
##  Durbin-Watson test
##
## data:  .
## DW = 0.99915, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

```
##        rm       nox   ptratio log(crim)
##  1.191368  2.810505  1.302618  3.125288
```

24.10 Model 2

No, we still can't robustly predict house prices based on number of rooms, air quality, student/teacher ratio and crime level.

- predictors are statistically significant
- but model is not robust
  - Standard residuals are NOT normally distributed

  - Standard residuals are NOT homoscedastic
  - Standard residuals are NOT independent
  - There is some sign of multicollinearity

Still possibly on the right path, not quite there yet…

Is there a difference between:

- Model1's $R^2 = 0.5721$ and Model2's $R^2 = 0.6688$?

24.11 Comparing R-squared

- R^2
  - measure of correlation between
    - values predicted by the model (fitted values)
    - observed values for outcome variable
- Adjusted R^2
  - adjusts the R^2 depending on
    - number of cases
    - number of predictor (independent) variables
  - *"unnecessary"* variables lower the value

The model with the highest adjusted R^2 has the best fit

24.12 Model difference with ANOVA

Can be used to test whether adjusted R^2 are signif. different

- if models are hierarchical
  - one uses all variables of the other
  - plus some additional variables

```
stats::anova(medv_model1, medv_model2)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ rm + nox
## Model 2: medv ~ rm + nox + ptratio + log(crim)
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    487 12890.0
## 2    485  9936.7  2    2953.3 72.073 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Still, neither model is robust

24.13 Information criteria

- Akaike Information Criterion (**AIC**)
  - measure of model fit

        * penalising model with more variables
      - not interpretable per-se, used to compare similar models
        * lower value, better fit
- Bayesian Information Criterion (**BIC**)
    - similar to AIC

```
stats::AIC(medv_model1)
```

```
## [1] 3000.763
```

```
stats::AIC(medv_model2)
```

```
## [1] 2877.258
```

24.14 Stepwise selection

Stepwise selection of predictor (independent) variables

- iteratively adding and/or removing predictors
- to obtain best performing model

Three approaches

- forward: from no variable, iteratively add variables
- backward: from all variables, iteratively remove variables
- both (a.k.a. step-wise):
    - from no variable
    - one step forward, add most promising variable
    - one step backward, remove any variable not improving

24.15 MASS::stepAIC

```
MASS::Boston %$%
  MASS::stepAIC(
    object =
      lm(medv ~ 1),
    scope =
      medv ~
        crim + zn + indus + chas + rm + nox + age +
        dis + rad + tax + ptratio + black + lstat,
    direction = "both",
    trace = FALSE
  ) ->
  medv_model3

medv_model3 %>%
  summary()
```

24.16 Model 3

```
##
## Call:
## lm(formula = medv ~ lstat + rm + ptratio + dis + nox + chas +
##     black + zn + crim + rad + tax)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5984  -2.7386  -0.5046   1.7273  26.2373
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  36.341145   5.067492   7.171 2.73e-12 ***
## lstat        -0.522553   0.047424 -11.019  < 2e-16 ***
## rm            3.801579   0.406316   9.356  < 2e-16 ***
## ptratio      -0.946525   0.129066  -7.334 9.24e-13 ***
## dis          -1.492711   0.185731  -8.037 6.84e-15 ***
## nox         -17.376023   3.535243  -4.915 1.21e-06 ***
## chas          2.718716   0.854240   3.183 0.001551 **
## black         0.009291   0.002674   3.475 0.000557 ***
## zn            0.045845   0.013523   3.390 0.000754 ***
## crim         -0.108413   0.032779  -3.307 0.001010 **
## rad           0.299608   0.063402   4.726 3.00e-06 ***
## tax          -0.011778   0.003372  -3.493 0.000521 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.736 on 494 degrees of freedom
## Multiple R-squared:  0.7406,Adjusted R-squared:  0.7348
## F-statistic: 128.2 on 11 and 494 DF,  p-value: < 2.2e-16
```

24.17 Checking assumptions

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.89904, p-value < 2.2e-16

##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 59.907, df = 11, p-value = 9.647e-09

##
##  Durbin-Watson test
##
## data:  .
## DW = 1.0779, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0

##    lstat       rm  ptratio      dis      nox     chas    black
## 2.581984 1.834806 1.757681 3.443420 3.778011 1.059819 1.341559
##       zn     crim      rad      tax
## 2.239229 1.789704 6.861126 7.272386
```

24.18 Validation

Can the model be generalised?

- split data into
  - training set: used to train the model
  - test set: used to test the model

Approaches:

- Validation
  - simple split: e.g. 80% traning, 20% test
- Cross-validation
  - leave-p-out: repeated split, leaving out p cases for test
    - leave-1-out
  - k-fold: repeated split, k equal size samples

24.19 caret::train

Use caret::train to cross-validate Model 3

```
library(caret)

train(
  formula(medv_model3),
  data = MASS::Boston,
  trControl = trainControl(
    method = "cv", # crossvalidate
    number = 5 # folds
  ),
  method = "lm", # regression model
  na.action = na.pass
) ->
medv_model3_crossv
```

24.20 Crossvalidate Model 3

```
medv_model3_crossv
```

```
## Linear Regression
##
## 506 samples
##  11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 403, 406, 405, 406, 404
## Resampling results:
##
```

```
##   RMSE     Rsquared  MAE
##   4.79003  0.729601  3.332228
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
medv_model3_crossv$resample
```

```
##       RMSE  Rsquared      MAE Resample
## 1 4.521453 0.7398752 3.190820    Fold1
## 2 5.343629 0.7144533 3.408707    Fold2
## 3 4.248468 0.8004829 2.981942    Fold3
## 4 4.780203 0.6512789 3.189664    Fold4
## 5 5.056398 0.7419147 3.890007    Fold5
```

24.21 Summary

Comparing regression models

- Information criteria
- Model difference
- Stepwise selection
- Validation

Next: Practical session

- Simple regression
- Multiple regression

Chapter 25

Machine Learning

25.1 Recap

Prev: Regression models

- 321 Lecture Simple regression
- 322 Lecture Assessing regression assumptions
- 323 Lecture Multiple regression
- 324 Practical session

Now: Machine Learning

- What's Machine Learning?
- Types
- Limitations

25.2 Definition

"The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience."

Mitchell, T. (1997). Machine Learning. McGraw Hill.

25.3 Origines

- **Computer Science**:
  - how to manually program computers to solve tasks
- **Statistics**:
  - what conclusions can be inferred from data
- **Machine Learning**:
  - intersection of **computer science** and **statistics**

  - how to get computers to **program themselves** from experience plus some initial structure
  - effective data capture, store, index, retrieve and merge
  - computational tractability

Mitchell, T.M., 2006. The discipline of machine learning (Vol. 9). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science, Machine Learning Department.

25.4 Types of machine learning

Machine learning approaches are divided into two main types

- **Supervised**
  - training of a *"predictive"* model from data
  - one (or more) attribute of the dataset is used to "predict" another attribute
  - e.g., classification
- **Unsupervised**
  - discovery of *descriptive* patterns in data
  - commonly used in data mining
  - e.g., clustering

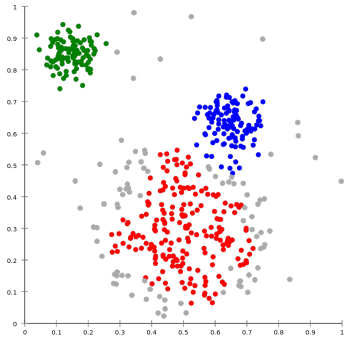
25.5 Supervised

- Training dataset
  - input attribute(s)
  - attribute to predict
- Testing dataset
  - input attribute(s)
  - attribute to predict
- Type of learning model
- Evaluation function
  - evaluates difference between prediction and output in testing data

by Josef Steppan via Wikimedia Commons, CC-BY-SA-4.0

25.6 Unsupervised

- Dataset
  - input attribute(s) to explore
- Type of model for the learning process
  - most approaches are iterative
  - e.g., hierarchical clustering
- Evaluation function
  - evaluates the quality of the pattern under consideration during one iteration



by Chire via Wikimedia Commons, CC-BY-SA-3.0

25.7 Semi-supervised learning

Supervised learning requires *"labelled data"*

- which can be expensive to acquire

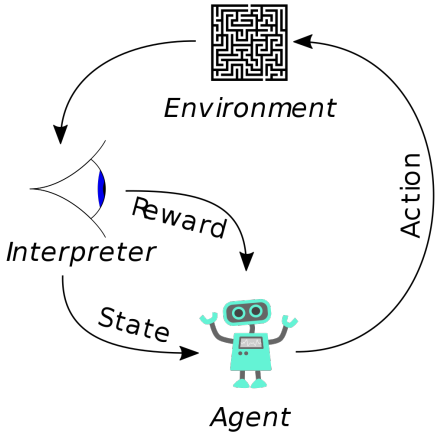
Semi-supervised learning

- combines a small amount of labelled data with a larger un-labelled dataset
  - train on small labelled dataset
  - apply model to larger unlabled dataset generating *"pseudo-labels"*
  - re-train the model with all data (including *"pseudo-labels"*)
- assumptions: continuity, cluster, and manifold (lower dimensionality)

25.8 Reinforcement learning

Based on the idea of training agents to learn how to

- take actions
  - which affect
    - agent state
    - environment
  - to maximize reward
  - balancing
    - exploration (new paths/options)
    - exploitation (of current knowledge)



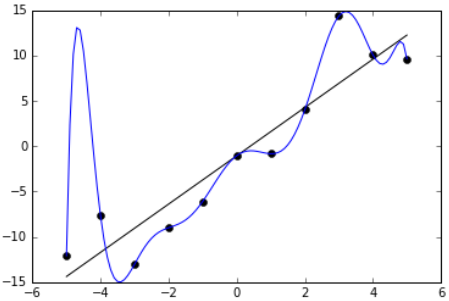
by Megajuice via Wikimedia Commons, CC0 1.0

25.9 Limits

- Complexity
- Creating a model requires hundreds of decisions
  - variable selection and normalisation
  - model, components, algorithm
  - hyper-parameters
  - evaluation
- Black-boxes
  - recent developments in explainable artificial intelligence

25.10 Overfitting

- creating a model
  - perfect for the training data
  - but not generic enough
  - to be useful for prediction
- An issue for machine learning
  - e.g., regression
    - n predictors can generate a line fitting the data exactly n cases
  - Occam's razor
  - one in ten rule
    - 10 cases per predictor



by Ghilesia Wikimedia Commons,CC-BY-SA-4.0

25.11 Algorithmic bias

Assumptions and training dataset quality still matter!

- garbage in, garbage out

Joy Buolamwini and Timnit Gebru's work on facial recognition

- black women were 35% less likely to be recognised than white men.
- Buolamwini, J. and Gebru, T., 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Conference on fairness, accountability and transparency (pp. 77-91).
- see also, Facial Recognition Is Accurate, if You're a White Guy by Steve Lohr (New York Times, Feb. 9, 2018)

25.12 Summary

Machine Learning

- What's Machine Learning?
- Types
- Limitations

Next: Artificial Neural Networks

- Logistic regression
- Artificial neural networks
- Deep learning

Chapter 26

Artificial Neural Networks

26.1 Recap

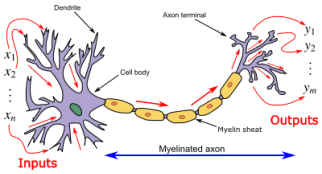
Prev: Machine Learning

- What's Machine Learning?
- Types
- Limitations

Now: Artificial Neural Networks

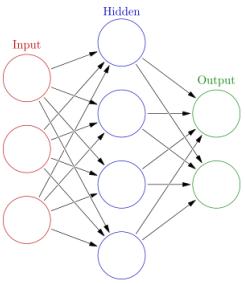
- Logistic regression
- Artificial neural networks
- Deep learning

26.2 Neural networks



Supervised learning approach simulating simplistic neurons

- Classic model with 3 sets
  - input neurons
  - output neurons
  - hidden layer(s)
    - combines input values using **weights**
    - **activation function**
- The **traning algorithm** is used to define the best weights



by Egm4313.s12 and Glosser.ca via Wikimedia Commons, CC-BY-SA-3.0

26.3 Artificial neurons

A model of the relationship between

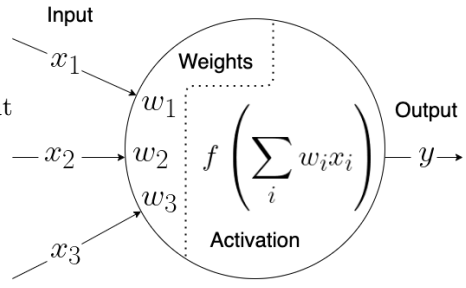
- a series of **input** values (predictors, independent variables)
- one **output** value (outcome, dependent variable)

Learns

- one **weight** (coefficient) per input

Using

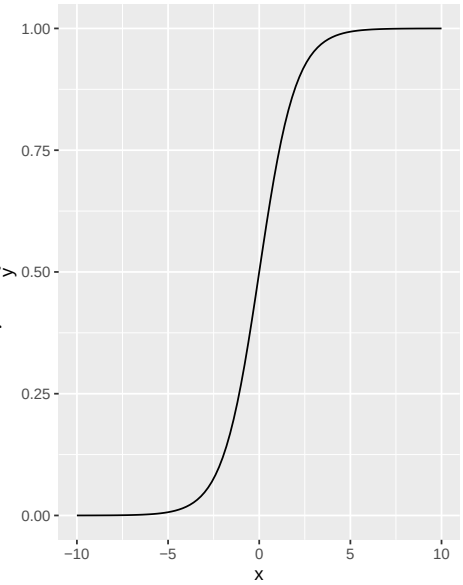
- an **activation function**
- a **training algorithm**



26.4 Logistic regression

The most common activation function is the *logistic sigmoid*

$$f(x) = \frac{1}{1 + e^{-x}}$$



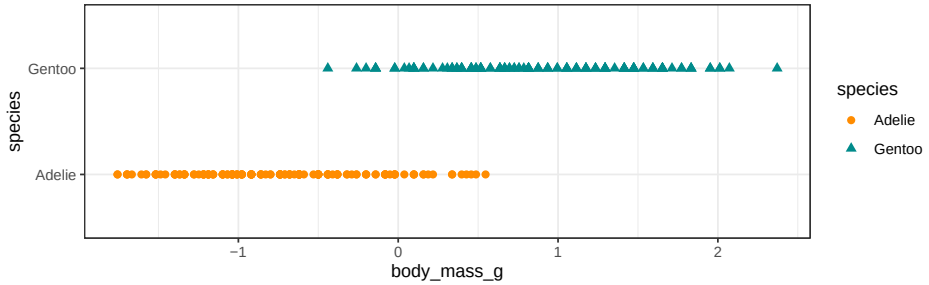
That would render each neuron a **logistic regression model**

- special case of the general linear model
- categorical outcomes

26.5 Example

Can we automatically identify the two species based on the penguins' body mass?

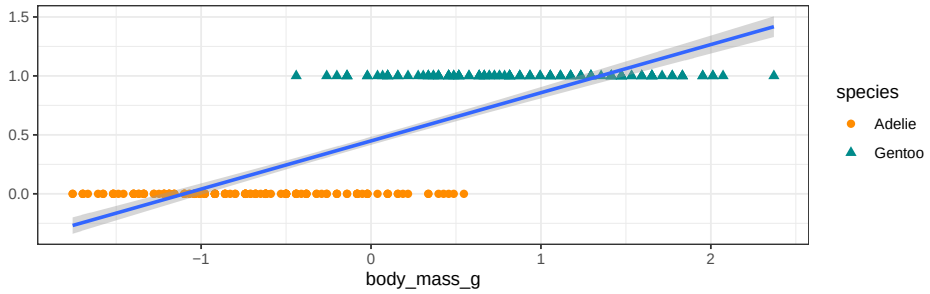
```
penguins_to_learn <-
  palmerpenguins::penguins %>%
  dplyr::filter(species %in% c("Adelie", "Gentoo")) %>%
  dplyr::mutate(species = forcats::fct_drop(species)) %>%
  dplyr::filter(!is.na(body_mass_g) | !is.na(bill_depth_mm)) %>%
  dplyr::mutate(dplyr::across(bill_length_mm:body_mass_g, scale))
```



26.6 Example

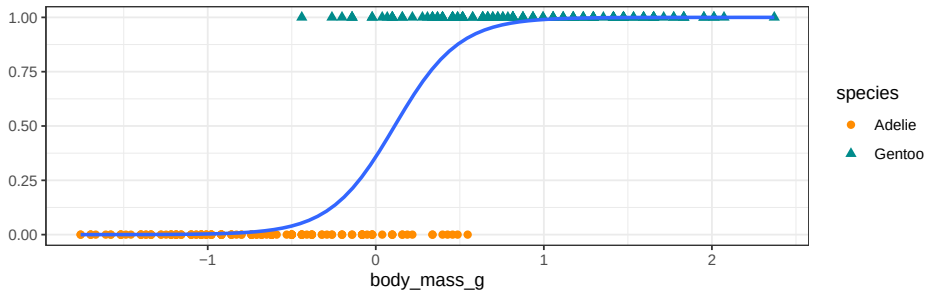
```
penguins_to_learn <-
  palmerpenguins::penguins %>%
```

```
  dplyr::filter(species %in% c("Adelie", "Gentoo")) %>%
  dplyr::mutate(species = forcats::fct_drop(species)) %>%
  dplyr::filter(!is.na(body_mass_g) | !is.na(bill_depth_mm)) %>%
  dplyr::mutate(dplyr::across(bill_length_mm:body_mass_g, scale)) %>%
  dplyr::mutate(
    species_01 = dplyr::recode(species, Adelie = 0, Gentoo = 1)
  )
```



26.7 Example

```
penguins_to_learn <-
  palmerpenguins::penguins %>%
  dplyr::filter(species %in% c("Adelie", "Gentoo")) %>%
  dplyr::mutate(species = forcats::fct_drop(species)) %>%
  dplyr::filter(!is.na(body_mass_g) | !is.na(bill_depth_mm)) %>%
  dplyr::mutate(dplyr::across(bill_length_mm:body_mass_g, scale)) %>%
  dplyr::mutate(
    species_01 = dplyr::recode(species, Adelie = 0, Gentoo = 1)
  )
```



26.8 stats::glm

```
sp_bm_model <- penguins_to_learn %$%
  stats::glm(species_01 ~ body_mass_g, family = binomial())
```

```
sp_bm_model %>%
  summary()
```

```
##
## Call:
## stats::glm(formula = species_01 ~ body_mass_g, family = binomial())
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.17324  -0.16597  -0.02288   0.14154   2.42133
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.5839     0.2666  -2.190   0.0285 *
## body_mass_g   5.2072     0.7271   7.161 7.98e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 376.98  on 273  degrees of freedom
## Residual deviance: 102.07  on 272  degrees of freedom
## AIC: 106.07
##
## Number of Fisher Scoring iterations: 7
```

26.9 Logistic regression

Assumptions

- **Linearity** of the logit
  - predictors have linear relationship with log of outcome
- When more than one predictor: **no multicollinearity**
  - if two or more predictor variables are used in the model, each pair of variables not correlated

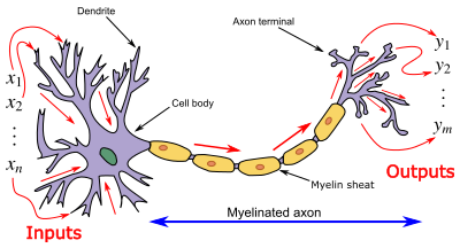
Pseudo-R2

- Approaches to calculating model quality (power)

Adding complexity

- Multiple logistic regression: multiple predictors
- Multinomial logistic regression: several categories as outcome

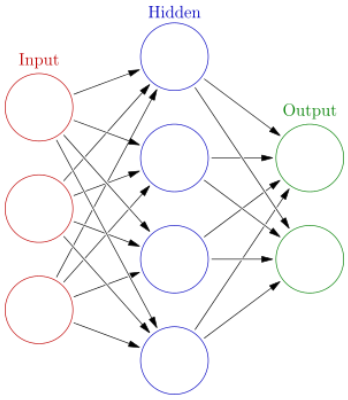
26.10 Network topology



Number of layers

- Single-layer network
  - effectively a logistic regression
- Multi-layer network
  - usually add one hidden layer
- Deep neural networks

Number of nodes



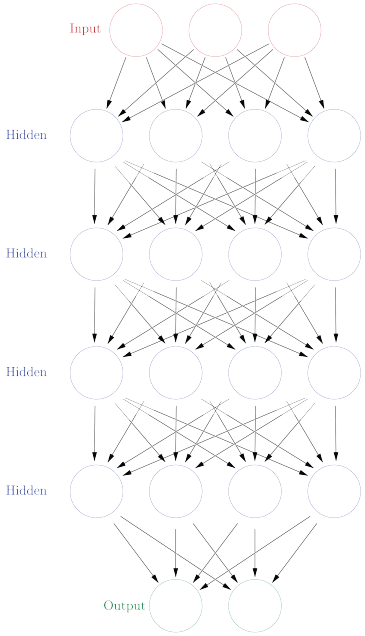
by Egm4313.s12 and Glosser.ca via Wikimedia Commons, CC-BY-SA-3.0

26.11 Defining a network

- Activation function
  - sigmoids
  - Rectified Linear Unit (ReLU)
- Training algorithm
  - Stochastic Gradient Descent
  - Adam
  - L-BFGS (quasi-Newton method)
- Training approach
  - feedforward (*"simple"* iterative training)
  - recurrent (*"short-memory"* of previous values)
  - backpropagation (of errors)

26.12 Deep neural networks

Neural networks with **multiple hidden layers**
The fundamental idea is that *"deeper"* neurons allow for the encoding of more complex characteristics
Example: De Sabbata, S. and Liu, P. (2019). Deep learning geodemographics with autoencoders and geographic convolution. In proceedings of the 22nd AGILE Conference on Geographic Information Science, Limassol, Cyprus.

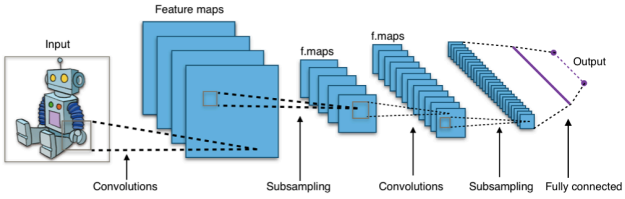


derived from work by Glosser.ca via Wikimedia Commons, CC-BY-SA-3.0

26.13 Convolutional neural networks

Deep neural networks with **convolutional hidden layers**

- used very successfully on image object recognition
- convolutional hidden layers *"convolve"* the images
  - a process similar to applying smoothing filters



by Aphex34 via Wikimedia Commons, CC-BY-SA-4.0

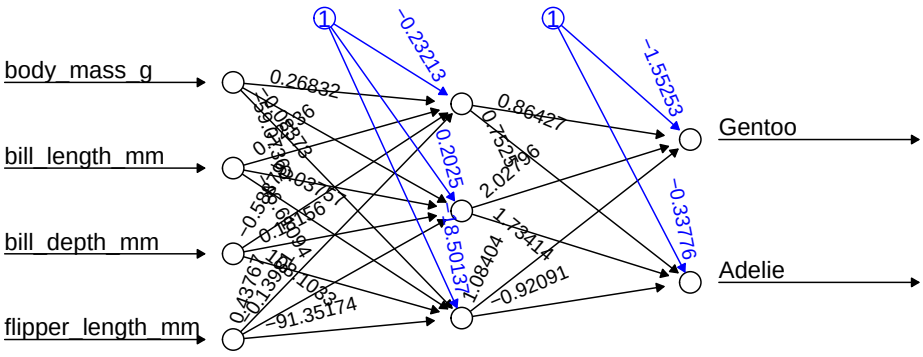
26.14 neuralnet::neuralnet

```
penguins_for_training <- penguins_to_learn %>% slice_sample(prop = 0.8)

penguins_for_test <- penguins_to_learn %>% anti_join(penguins_for_training)

species_nnet <-
  neuralnet::neuralnet(
    species ~ body_mass_g + bill_length_mm + bill_depth_mm +flipper_length_mm,
    hidden = 3, data = penguins_for_training
  )

species_nnet %>%  plot(rep = "best")
```



26.15 Performance

```
# Use the model to predict species
penguins_predicted <-
  neuralnet::compute(
    species_nnet,
    penguins_for_test
)

# Add predicted species to table
penguins_for_test <-
  penguins_for_test %>%
  dplyr::mutate(
    predicted_species =
      penguins_predicted %$%
      net.result %>%
      max.col %>%
      recode(
        `1` = "Adelie",
        `2` = "Gentoo"
      )
  )

# Calculate confusion matrix
caret::confusionMatrix(
  penguins_for_test %>%
    pull(predicted_species) %>%
    forcats::as_factor(),
  penguins_for_test %>%
    pull(species) %>%
    forcats::as_factor()
  )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Adelie Gentoo
##     Adelie     27      0
##     Gentoo      0     28
##
##                Accuracy : 1
##                  95% CI : (0.9351, 1)
##     No Information Rate : 0.5091
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.4909
##          Detection Rate : 0.4909
##    Detection Prevalence : 0.4909
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : Adelie
##
```

26.16 Summary

Artificial Neural Networks

- Logistic regression
- Artificial neural networks
- Deep learning

Next: Support vector machines

- Hyperplanes
- Linear separability
- Kernels

Chapter 27

Support vector machines

27.1 Recap

Prev: Artificial Neural Networks

- Logistic regression
- Artificial neural networks
- Deep learning

Now: Support vector machines

- Hyperplanes
- Linear separability
- Kernels

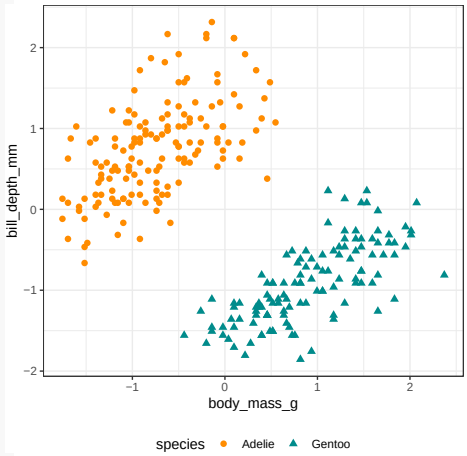
27.2 Classification task

Can we learn to distinguish the two species from body mass and bill depth?

```
penguins_to_learn <-
  palmerpenguins::penguins %>%
  dplyr::filter(
    species %in% c("Adelie", "Gentoo")
  ) %>%
  dplyr::mutate(
    species = forcats::fct_drop(species)
  ) %>%
  dplyr::filter(
    !is.na(body_mass_g)
    | !is.na(bill_depth_mm)
  ) %>%
  dplyr::mutate(dplyr::across(
      bill_length_mm:body_mass_g,
      scale
  ))

penguins_for_training <-
  penguins_to_learn %>%
  slice_sample(prop = 0.8)

penguins_for_testing <-
  penguins_to_learn %>%
  anti_join(penguins_for_training)
```



27.3 Support vector machines

Supervised learning approach to classification

- a series of **input** values (predictors, independent variables)
- one **output** categorical value (outcome, dependent variable)

Partition of multidimensional space

- finding boundaries (*"hyperplanes"*)
- between homogenous groups of observations
- approach akin to
  - linear regression modelling
  - nearest neighbours approaches

27.4 Nearest Neighbours (k-NN)

One of the simplest approaches to classification

Classification of a new observation:

- select `k` closest observations in multidimensional space
- new observation classified as most frequent class

```
library(class)

species_3nn <- class::knn(
    train = penguins_for_training %>% dplyr::pull(body_mass_g, bill_depth_mm),
    test = penguins_for_testing %>% dplyr::pull(body_mass_g, bill_depth_mm),
```

```
    cl = penguins_for_training %>% dplyr::pull(species),
    k = 3
  )

penguins_for_testing <- penguins_for_testing %>%
  tibble::add_column(predicted_species_3nn = species_3nn)

caret::confusionMatrix(
  penguins_for_testing %>% dplyr::pull(predicted_species_3nn),
  penguins_for_testing %>% dplyr::pull(species)
)
```

27.5 Performance

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Adelie Gentoo
##     Adelie     30      1
##     Gentoo      4     20
##
##                Accuracy : 0.9091
##                  95% CI : (0.8005, 0.9698)
##     No Information Rate : 0.6182
##     P-Value [Acc > NIR] : 1.198e-06
##
##                   Kappa : 0.8125
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.8824
##             Specificity : 0.9524
##          Pos Pred Value : 0.9677
##          Neg Pred Value : 0.8333
##              Prevalence : 0.6182
##          Detection Rate : 0.5455
##    Detection Prevalence : 0.5636
##       Balanced Accuracy : 0.9174
##
##        'Positive' Class : Adelie
##
```

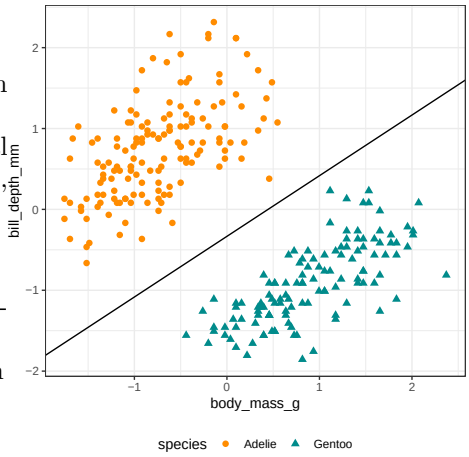
27.6 Hyperplanes

If a hyperplane can be drawn between classes

- e.g., a line in bi-dimensional space, a plane in three dimensions, etc
- classes are linearly separable

Find maximum-margin hyperplane

- line that maximises separation between classes
- conceptually similar to regression



27.7 e1071::svm

```
library(e1071)

species_svm <- penguins_for_training %$%
  e1071::svm(
    species ~
      body_mass_g + bill_depth_mm,
    kernel = "linear",
    scale = FALSE
  )

penguins_for_testing <-
  penguins_for_testing %>%
  tibble::add_column(
    predicted_species_svn =
      stats::predict(
        species_svm,
        penguins_for_testing %>%
          dplyr::select(body_mass_g, bill_depth_mm)
      )
  )

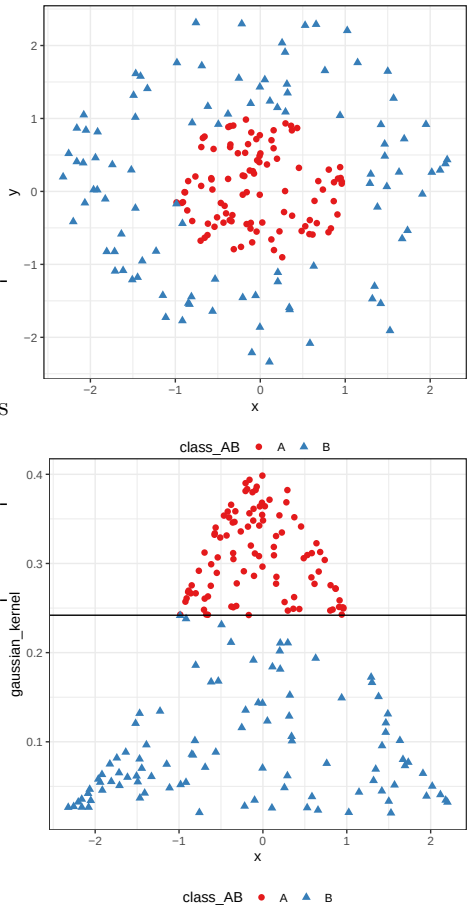
caret::confusionMatrix(
  penguins_for_testing %>%
    dplyr::pull(predicted_species_svn),
  penguins_for_testing %>%
    dplyr::pull(species)
  )
```

27.8 Performance

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction Adelie Gentoo
##     Adelie     34      0
##     Gentoo      0     21
##
##                Accuracy : 1
##                  95% CI : (0.9351, 1)
##     No Information Rate : 0.6182
##     P-Value [Acc > NIR] : 3.246e-12
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6182
##          Detection Rate : 0.6182
##    Detection Prevalence : 0.6182
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : Adelie
##
```

27.9 Not linearly separable



What if classes are *not* linearly separable?

- slack variable `C`
  - soft margin between classes
  - a *"cost"* is applied to cases beyond margins
- kernels *"trick"*
  - functions used to create additional dimensions
  - as functions of input values
    - linear, polinomial, sigmoids, Gaussian

27.10 e1071::svm

```
class_AB_svm <-
  data_AB_for_training %$%
  e1071::svm(
    class_AB ~
      x + y,
    kernel = "linear",
    scale = FALSE
  )

data_AB_for_testing <-
  data_AB_for_testing %>%
  tibble::add_column(
    predicted_AB_svm =
      stats::predict(
        class_AB_svm,
```

```
        data_AB_for_testing %>%
          dplyr::select(x, y)
      )
  )

caret::confusionMatrix(
  data_AB_for_testing %>%
    dplyr::pull(predicted_AB_svm),
  data_AB_for_testing %>%
    dplyr::pull(class_AB)
  )
```

27.11 Performance

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B
##          A 18 14
##          B  4  4
##
##                Accuracy : 0.55
##                  95% CI : (0.3849, 0.7074)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 0.56505
##
##                   Kappa : 0.0426
##
##  Mcnemar's Test P-Value : 0.03389
##
##             Sensitivity : 0.8182
##             Specificity : 0.2222
##          Pos Pred Value : 0.5625
##          Neg Pred Value : 0.5000
##              Prevalence : 0.5500
##          Detection Rate : 0.4500
##    Detection Prevalence : 0.8000
##       Balanced Accuracy : 0.5202
##
##        'Positive' Class : A
##
```

27.12 e1071::svm

```
class_AB_svm_radial <-
  data_AB_for_training %$%
```

```
  e1071::svm(
    class_AB ~ x + y,
    kernel = "radial",
    scale = FALSE,
    cost = 10
  )

data_AB_for_testing <-
  data_AB_for_testing %>%
  tibble::add_column(
    predicted_AB_svm_radial =
      stats::predict(
        class_AB_svm_radial,
        data_AB_for_testing %>%
          dplyr::select(x, y)
      )
  )

caret::confusionMatrix(
  data_AB_for_testing %>%
    dplyr::pull(predicted_AB_svm_radial),
  data_AB_for_testing %>%
    dplyr::pull(class_AB)
  )
```

27.13 Performance

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  A  B
##          A 22  1
##          B  0 17
##
##                Accuracy : 0.975
##                  95% CI : (0.8684, 0.9994)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 1.388e-09
##
##                   Kappa : 0.9492
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9444
##          Pos Pred Value : 0.9565
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5500
##          Detection Rate : 0.5500
```

```
##    Detection Prevalence : 0.5750
##       Balanced Accuracy : 0.9722
##
##        'Positive' Class : A
##
```

27.14 Summary

Support vector machines

- Hyperplanes
- Linear separability
- Kernels

Next: Practical session

- Artificial neural networks
- Support vector machines

Chapter 28

Principal Component Analysis

28.1 Recap

Prev: Comparing data

- 401 Lecture Introduction to Machine Learning
- 402 Lecture Artificial Neural Networks
- 403 Lecture Support vector machines
- 404 Practical session

Now: Principal Component Analysis

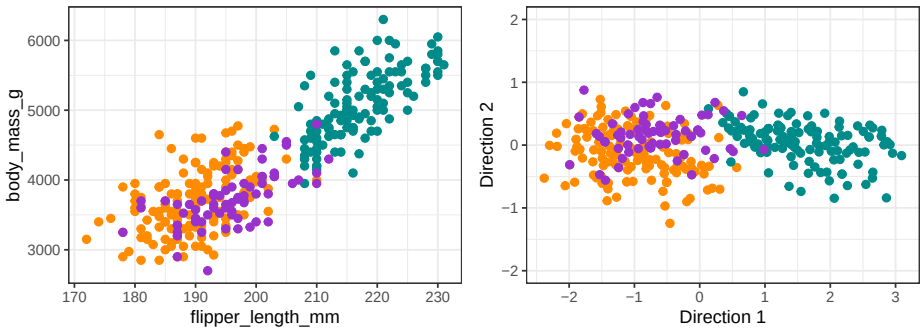
- Principal components
- `stats::prcomp`
- Dimensionality reduction

28.2 Principal components

Principal component are

- a set of directions orthogonal to each other
- that best fit a set of data

Can be interpreted as a *"re-projection"* of the data



28.3 Dimensionality reduction

Alternatively, principal components can be interpreted as

- **lower-dimensional** representation of the data

Especially useful when working numerous variables

- a limited number of principal components can be retained
  - most variance maintained
  - distance in data space approximated
  - high-dimensional data can be more easily plotted
- commonly used as dimensionality reduction step
  - supervised learning models
    - linear regression
  - clustering

28.4 stats::prcomp

Principal component analysis on body mass, flipper length, and bill length and depth

```
penguins_pca <-
  palmerpenguins::penguins %>%
  dplyr::select(bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g) %>%
  # remove missing data
  dplyr::filter(
    !is.na(bill_length_mm) | !is.na(bill_depth_mm) |
    !is.na(flipper_length_mm) | !is.na(body_mass_g)
  ) %>%
  stats::prcomp(center = TRUE, scale. = TRUE)

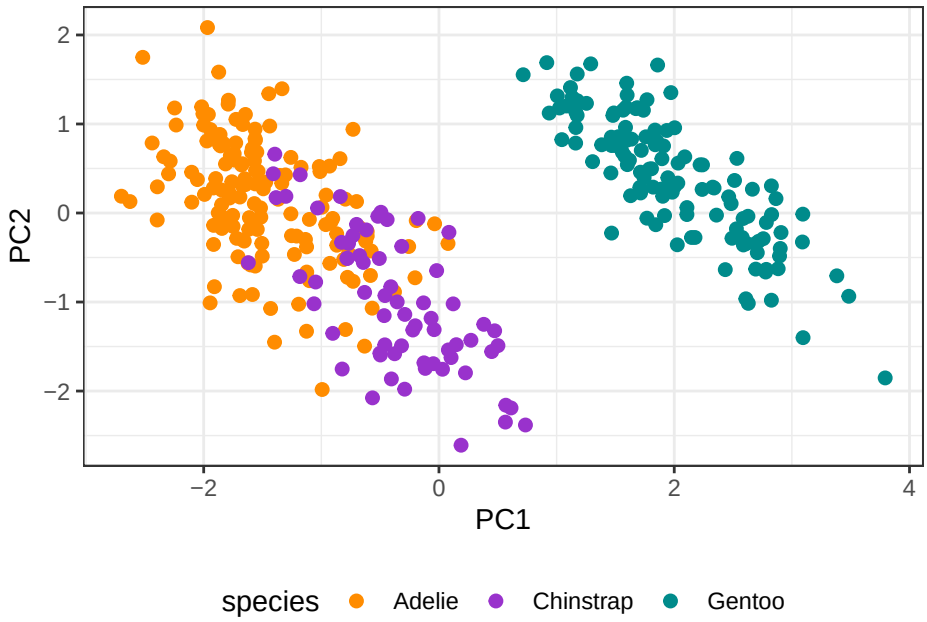
summary(penguins_pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.6594 0.8789 0.60435 0.32938
## Proportion of Variance 0.6884 0.1931 0.09131 0.02712
## Cumulative Proportion  0.6884 0.8816 0.97288 1.00000
```

The first component alone explains 68.84% of variance, and the first two together explain 88.16% of variance

28.5 PCA results

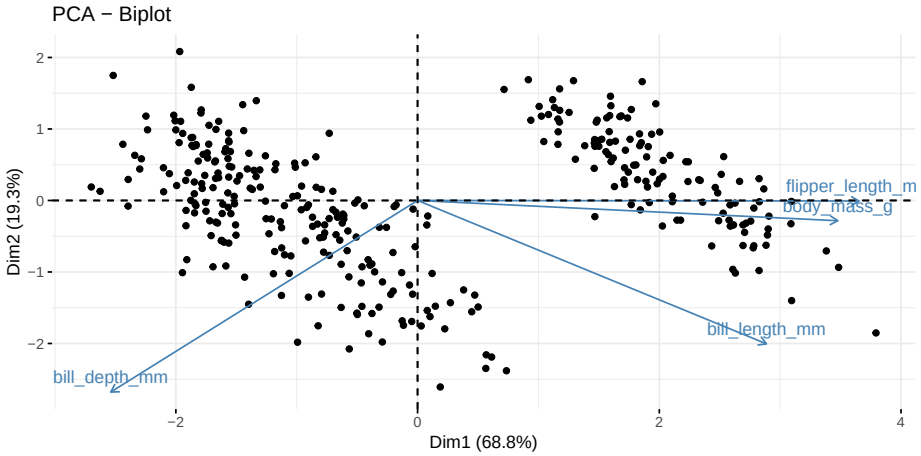
```
penguins_with_pca <- palmerpenguins::penguins %>%
  dplyr::filter(!is.na(bill_length_mm) | !is.na(bill_depth_mm) |
                !is.na(flipper_length_mm) | !is.na(body_mass_g)) %>%
  dplyr::bind_cols(
      penguins_pca %$% x %>% as.data.frame()
  )
```



28.6 Plotting PCA

```
library(factoextra)

penguins_pca %>% fviz_pca_biplot(label = "var")
```



28.7 Summary

Principal Component Analysis

- Principal components
- `stats::prcomp`
- Interpretation

Next: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

Chapter 29

Centroid-based clustering

29.1 Recap

Prev: Principal Component Analysis

- Principal Components
- Interpretation

Now: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

29.2 Clustering task

*"Clustering is an unsupervised machine learning task that automatically divides the data into **clusters** , or groups of similar items".* (Lantz, 2019)

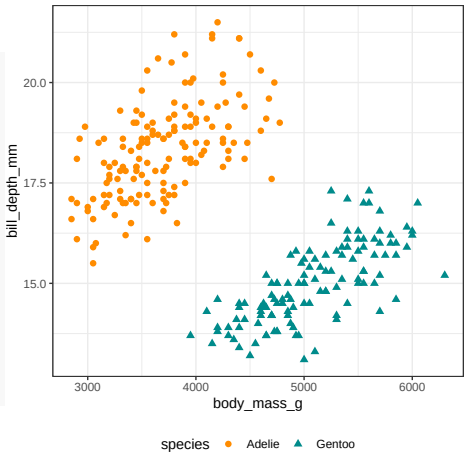
Methods:

- Centroid-based
  - k-means
  - fuzzy c-means
- Hierarchical
- Mixed
  - bootstrap aggregating
- Density-based
  - DBSCAN

29.3 Example

Can we automatically identify the two groups visible in the scatterplot, without any previous knowledge of the groups?

```
# Prepared data
penguins_to_cluster <-
  palmerpenguins::penguins %>%
  dplyr::filter(
    species %in%
      c("Adelie", "Gentoo")
  ) %>%
  dplyr::filter(
    !is.na(body_mass_g) |
      !is.na(bill_depth_mm)
  )
```



29.4 k-means algorithm

k-mean clusters n observations (x) in k clusters (c), minimising the within-cluster sum of squares (WCSS)

$$WCSS = \sum_{c=1}^{k} \sum_{x \in c} (x - \overline{x}_c)^2$$

Algorithm: `k` observations a randomly selected as initial centroids, then repeat

- **assignment step**: observations assigned to closest centroids
- **update step**: calculate means for each cluster, as new centroid

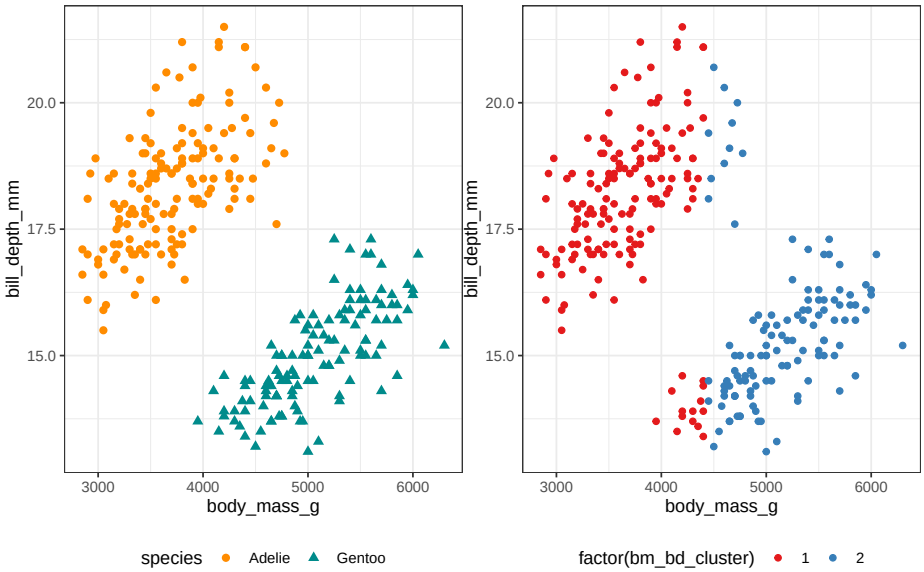
until centroids don't change anymore, the algorithm has **converged**

29.5 stats::kmeans

```
# Execute k-means
bm_bd_clusters <-
  penguins_to_cluster %>%
  dplyr::select(body_mass_g, bill_depth_mm) %>%
  stats::kmeans(
    centers = 2,  # number of clusters (k)
    iter.max = 50 # max number of iterations
  )
```

```
# Add clusters to table
penguins_clustered_bm_bd <-
  penguins_to_cluster %>%
  tibble::add_column(
    bm_bd_cluster = bm_bd_clusters %$% cluster
  )
```

29.6 k-means result



29.7 stats::kmeans

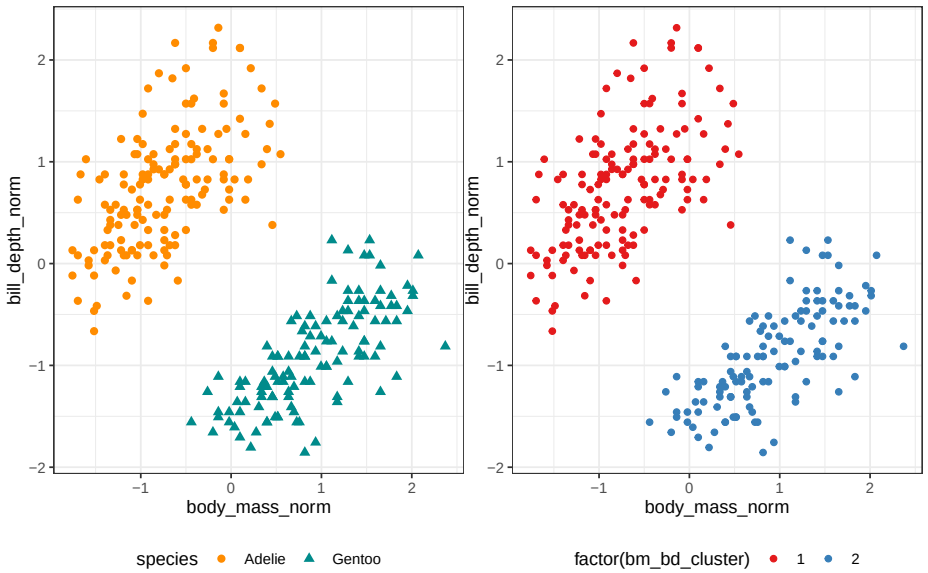
```
# First, normalise values
penguins_norm <-
  penguins_to_cluster %>%
  dplyr::mutate(
          body_mass_norm = scale(body_mass_g),
          bill_depth_norm = scale(bill_depth_mm)
  )

bm_bd_norm_clusters <-
  penguins_norm %>%
  dplyr::select(body_mass_norm, bill_depth_norm) %>%
  stats::kmeans(centers = 2, iter.max = 50)

penguins_clustered_bm_bd_norm <- penguins_norm %>%
```

```
  tibble::add_column(
    bm_bd_cluster = bm_bd_norm_clusters %$% cluster
  )
```

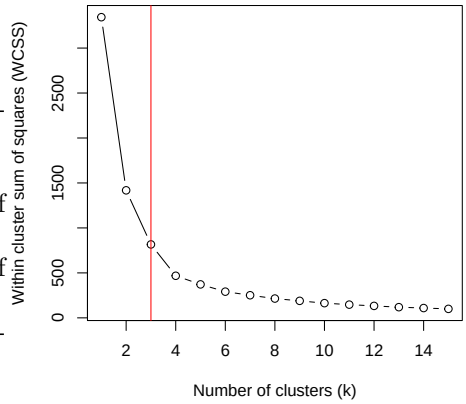
29.8 k-means result



29.9 Limitations

K-means requires to select a fixed number of clusters **in advance**
Elbow method:

- calculate clusters for a range of number of clusters
- select the minimum number of clusters that minimises WCSS
- before increasing number of clusters leads minimal benefit



Example for random data generated to be in 3 clusters

29.10 Fuzzy c-means

Similar to k-means but allows for *"fuzzy"* membership to clusters

Each observation is assigned with a value per each cluster

- usually from `0` to `1`
- indicates how well the observation fits within the cluster
- i.e., based on the distance from the centroid

```
library(e1071)

bm_bd_norm_fclusters <- penguins_norm %>%
  dplyr::select(body_mass_norm, bill_depth_norm) %>%
  e1071::cmeans(centers = 2, iter.max = 50)

penguins_clustered_bm_bd_fuzzy <- penguins_norm %>%
  tibble::add_column(bm_bd_fuzzy_cluster = bm_bd_norm_fclusters %$% cluster)
```

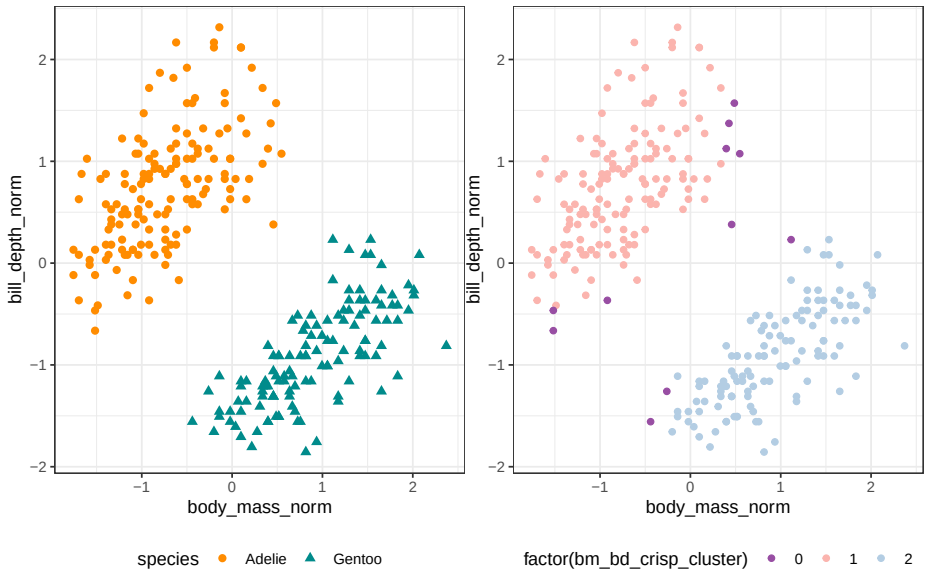
29.11 Fuzzy c-means

A *"crisp"* classification can be created by picking the highest membership value.

- that also allows to set a membership threshold (e.g., `0.75`)
- leaving some observations without a cluster

```
penguins_clustered_bm_bd_fuzzy <-
  penguins_clustered_bm_bd_fuzzy %>%
  tibble::add_column(
    bm_bd_fuzzy_cluster_membership =
      apply(bm_bd_norm_fclusters %$% membership, 1, max)
  ) %>%
  dplyr::mutate(
    bm_bd_crisp_cluster = ifelse(
      bm_bd_fuzzy_cluster_membership < 0.75,
      0, bm_bd_fuzzy_cluster
    )
  )
```

29.12 Fuzzy c-means result



29.13 Geodemographic classifications

In GIScience, clustering is used to create *geodemographic classifications* such as the 2011 Output Area Classification from the UK Census 2011 (Gale *et al.*, 2016)

- initial set of 167 prospective variables
  - 86 were removed,
  - 41 were retained as they are
  - 40 were combined
  - final set of 60 variables.
- k-means clustering approach to create
  - 8 supergroups
  - 26 groups
  - 76 subgroups

29.14 Summary

Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

Next: Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

Chapter 30

Hierarchical and density-based clustering

30.1 Recap

Prev: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

Now: Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
```

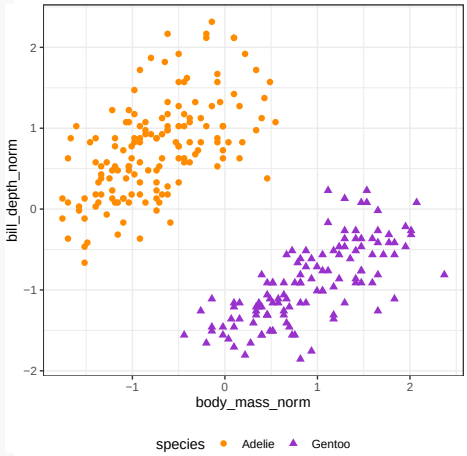
```
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

30.2 Example

Can we automatically identify the two groups visible in the scatterplot, without any previous knowledge of the groups?

```
penguins_norm <-
  palmerpenguins::penguins %>%
  dplyr::filter(
    species %in%
      c("Adelie", "Gentoo")
  ) %>%
  dplyr::filter(
    !is.na(body_mass_g) |
      !is.na(bill_depth_mm)
  ) %>%
  dplyr::mutate(
          body_mass_norm =
            scale(body_mass_g),
          bill_depth_norm =
            scale(bill_depth_mm)
  )
```



30.3 Hierarchical clustering

Bottom-up approach

- rather than splitting objects into clusters
- aggregate from single objects upwards

Algorithm:

- each object is initialised as it's own cluster
- then repeat
  - join the two most similar clusters
    - based on a distance-based metric
    - e.g., Ward's (1963) approach is based on variance
  - until only one single cluster is achieved

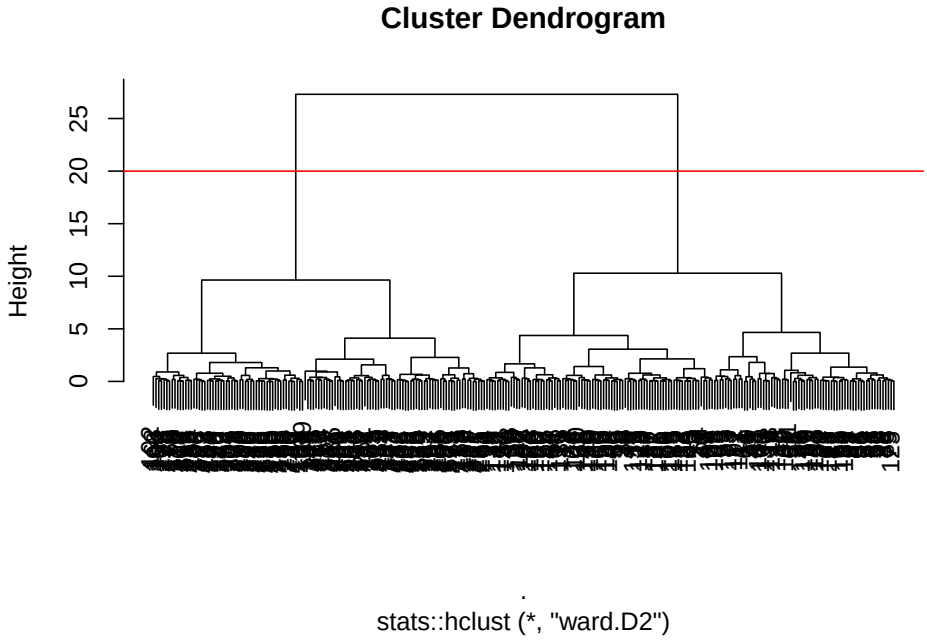
Limitation: computationally expensive

```r
penguins_hclust_result <-
  penguins_norm %>%
  dplyr::select(
    body_mass_norm,
    bill_depth_norm
  ) %>%
  # Calculate distance matrix
  stats::dist(method="euclidean") %>%
  # Cluster data
  stats::hclust(method="ward.D2")

penguins_bm_bd_hclust <- penguins_norm %>%
  tibble::add_column(
    bm_bd_hclust = stats::cutree(
      penguins_hclust_result,
      k = 2
    )
  )
```

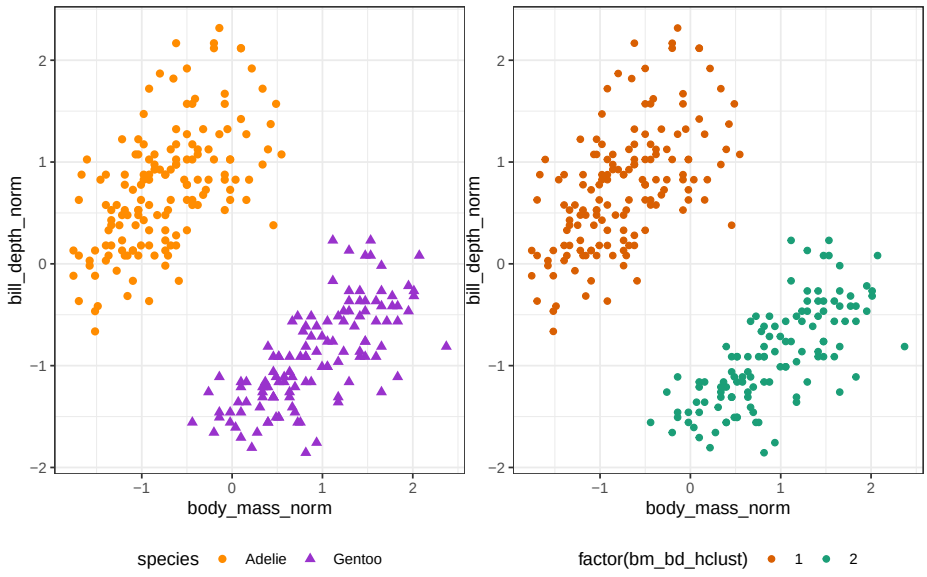
30.5 clustering tree

Generates a clustering tree (dendrogram), which can then be *"cut"* at the desired height




```
## integer(0)
```

30.6 Hierarchical clustering result



30.7 Bagged clustering

Bootstrap aggregating (*b-agg-ed*) clustering approach

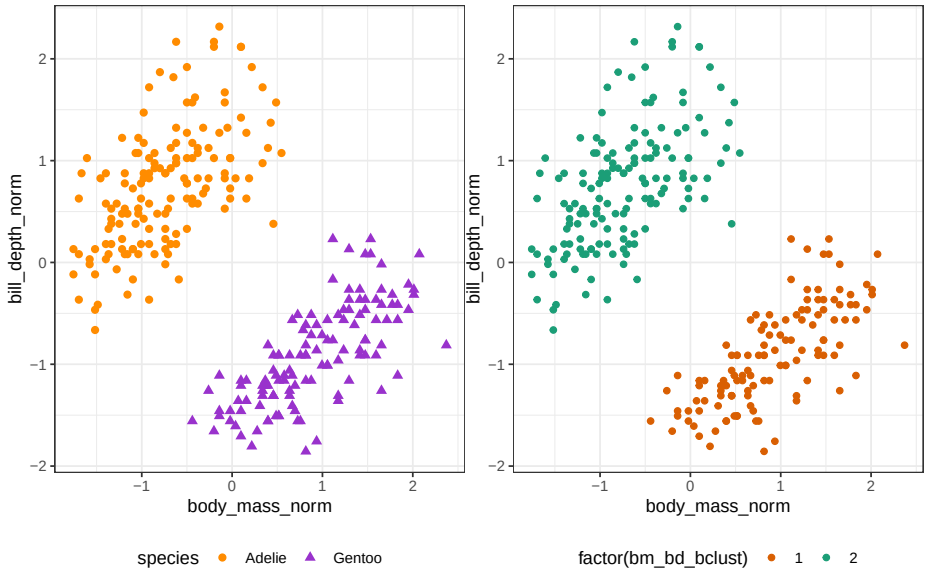
- first, k-means
  - randomly select a sample
  - calculate K-means
  - repeat on *many* samples
- then, hierarchical
  - execute hierarchical clustering on the centroids of the clusters generated in the previous step
- finally
  - select required number of clusters
  - assign object to closest centroid

Leisch, F., 1999. Bagged clustering.

```
penguins_bclust_result <-
  penguins_norm %>%
  dplyr::select(body_mass_norm, bill_depth_norm) %>%
  e1071::bclust(
    hclust.method = "ward.D2",
    resample = TRUE
  )

penguins_bm_bd_bclust <-
  penguins_norm %>%
  tibble::add_column(
    bm_bd_bclust = e1071::clusters.bclust(
      penguins_bclust_result,
      2
    )
  )
```

30.9 Bagged clustering result



30.10 Density based clustering

Density-based spatial clustering of applications with noise (DBSCAN)

- start from a random unclustered point
  - proceed by aggregating its neighbours to the same cluster
    - as long as they are within a certain distance `eps`
- once no more objects can be added
  - select another random point
  - and start aggregating again to a new cluster

Limitation: selection of `eps`

Ester, M., Kriegel, H.P., Sander, J. and Xu, X., 1996, August. Density-based spatial clustering of applications with noise. In Int. Conf. Knowledge Discovery and Data Mining (Vol. 240, p. 6).

30.11 dbscan::dbscan

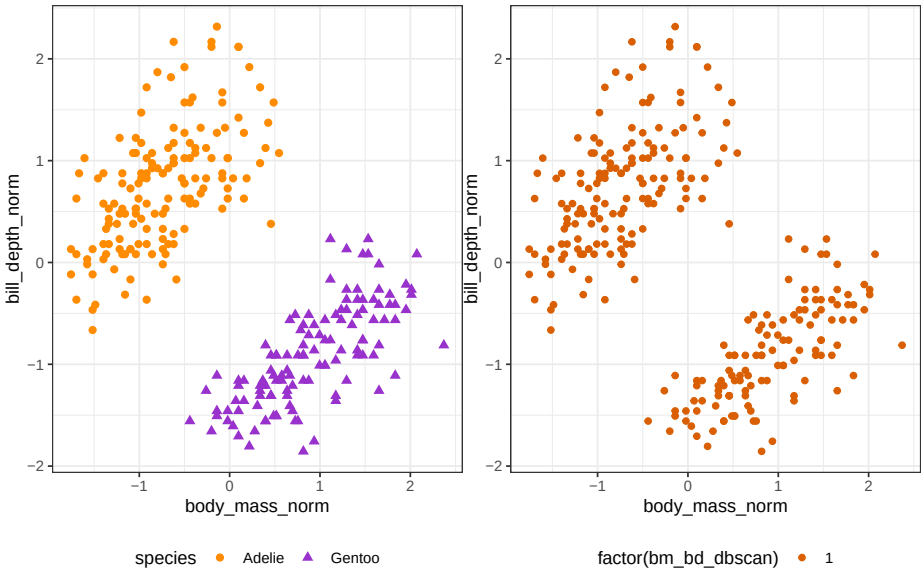
```
penguins_dbscan_result <-
  penguins_norm %>%
  dplyr::select(body_mass_norm, bill_depth_norm) %>%
  dbscan::dbscan(
    eps = 1,
    minPts = 5
```

```
  )

penguins_bm_bd_dbscan <-
  penguins_norm %>%
  tibble::add_column(
    bm_bd_dbscan =
      penguins_dbscan_result %$%
      cluster
  )
```

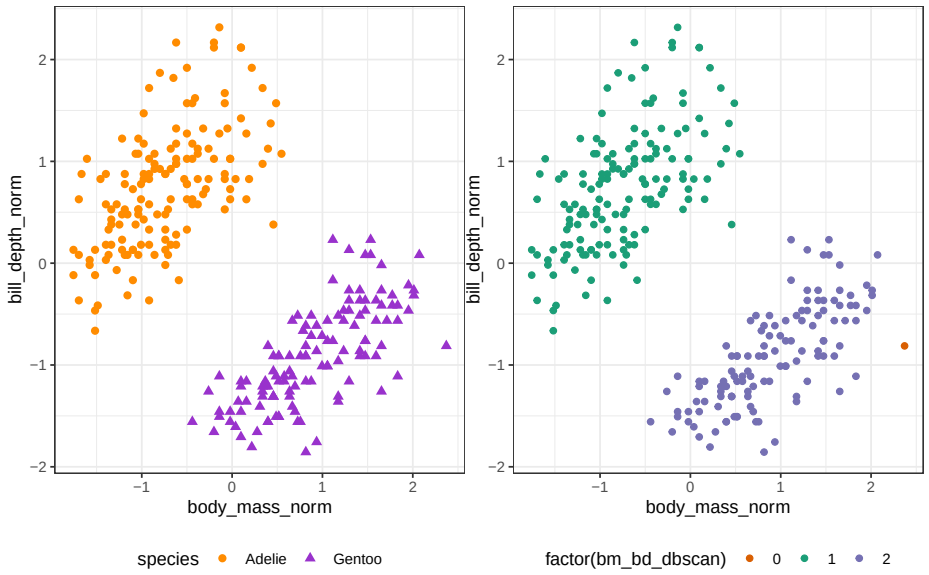
30.12 DBSCAN result

Using: `dbscan(eps = 1, minPts = 5)`



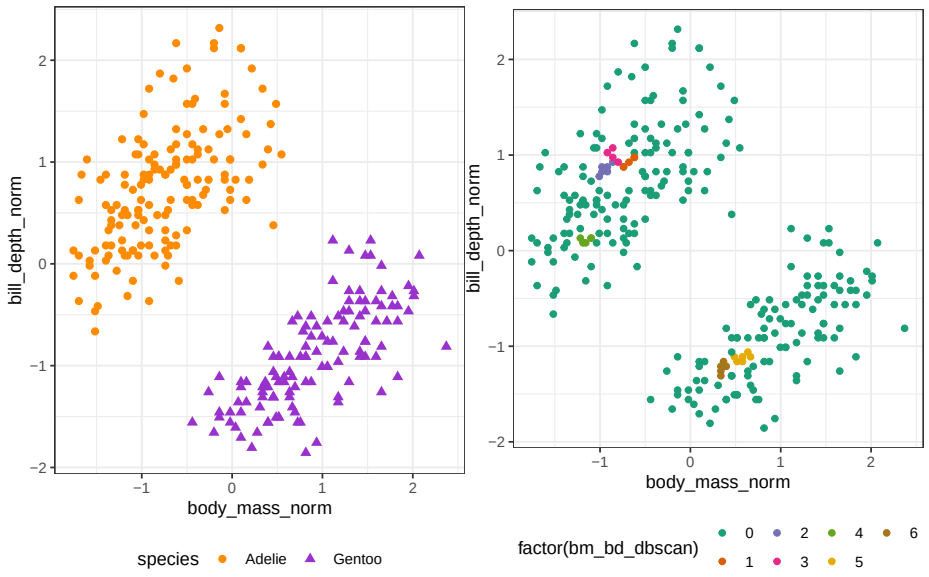
30.13 DBSCAN result

Using: `dbscan(eps = 0.5, minPts = 5)`

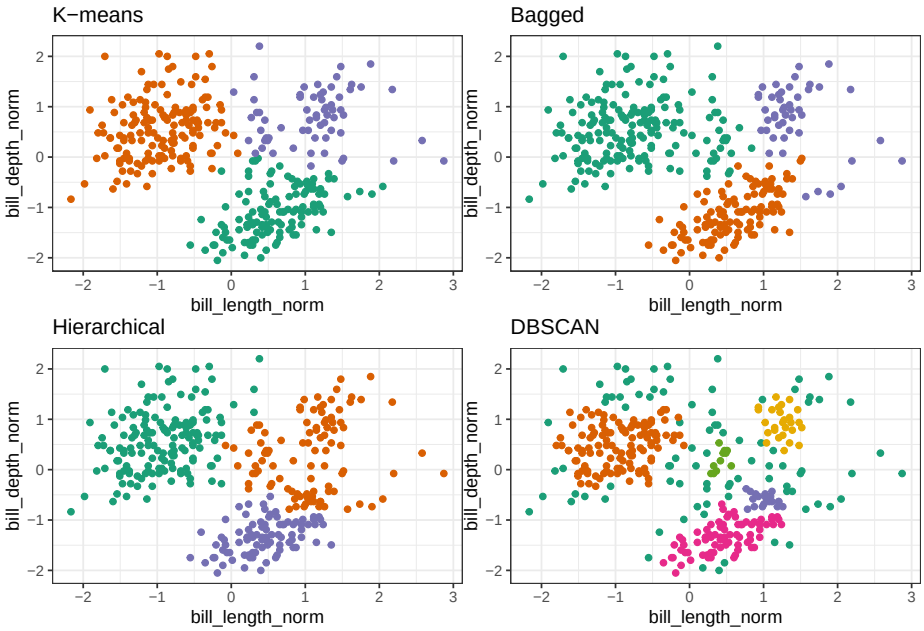


30.14 DBSCAN result

Using: `dbscan(eps = 0.1, minPts = 5)`



30.15 Not alwasy that easy…



30.16 Summary

Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

Next: Practical session

- Geodemographic classification