Lecture materials | granolarr

Stefano De Sabbata

2020-11-16

Contents

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

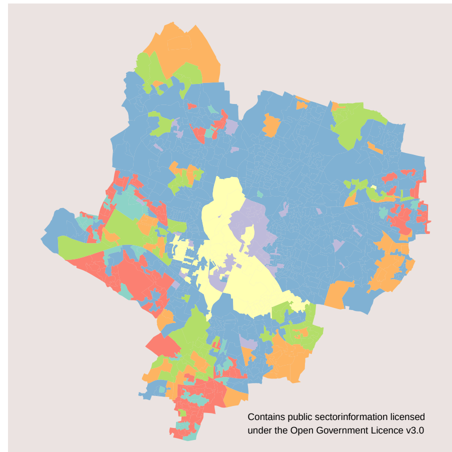
basis for

- *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

```r
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

```r
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
```

```r
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

```r
library(stringr)

str_length("Leicester")
```

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

```r
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

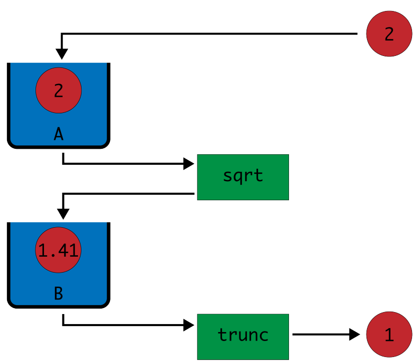
3.7 Pipe example

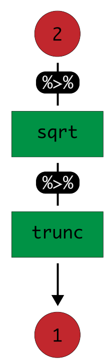
```
tmp_variable_A <- 2
tmp_variable_B <- sqrt(tmp_variable_A)
trunc(tmp_variable_B)

## [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`

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

```
## Negative
```

```r
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

```r
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

```
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

```
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          1973
## 2         Pete  34 Researcher Leicester          1986
## 3        Sarah  33 Researcher Leicester          1987
```

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`)

```r
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

```r
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

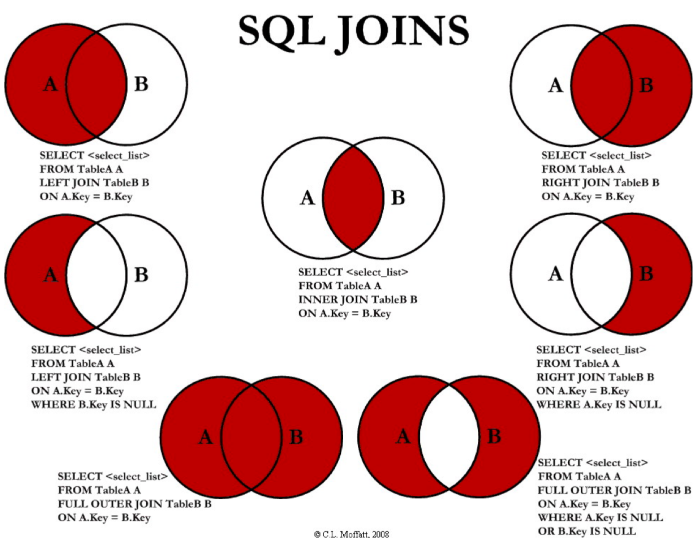
city_name	city_area_km2
Barcelona	101
London	1572
Rome	496
Munich	310

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



by C.L. Moffatt, licensed under The Code Project Open License (CPOL)

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

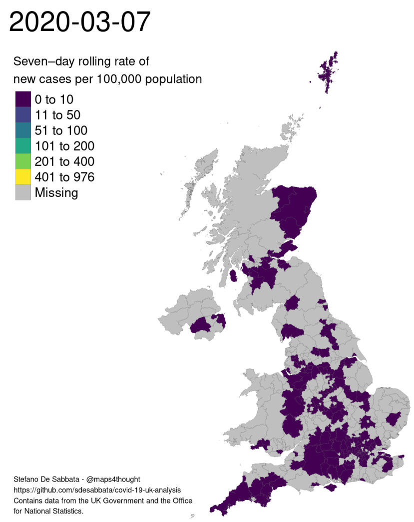
```
city_info_long_noNAs <-
  city_info_long %>%
  # Columns to drop where NA
  tidyr::drop_na(cases)
```

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

```r
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

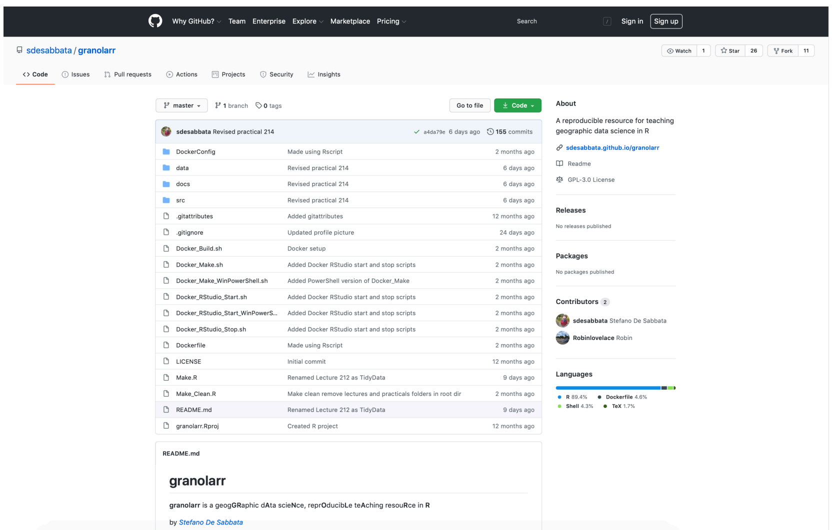
13.12 Store and share

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 This repository



github.com/sdesabbata/granolarr

13.14 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:

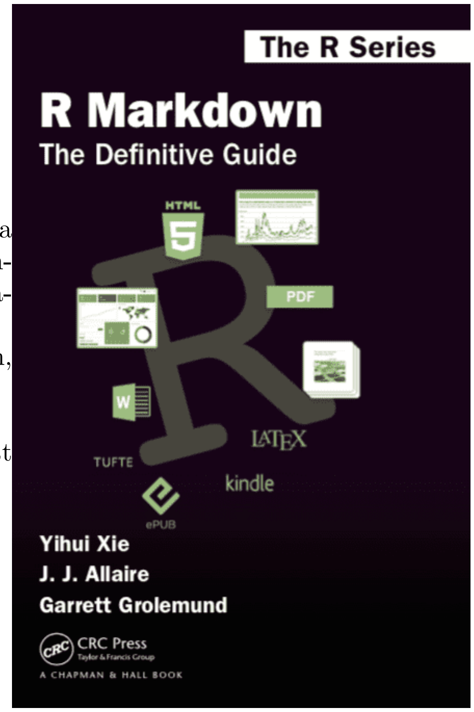
- loads the necessary libraries
- loads the flights from New York City in 2013
- presents a few columns from the first row

```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?

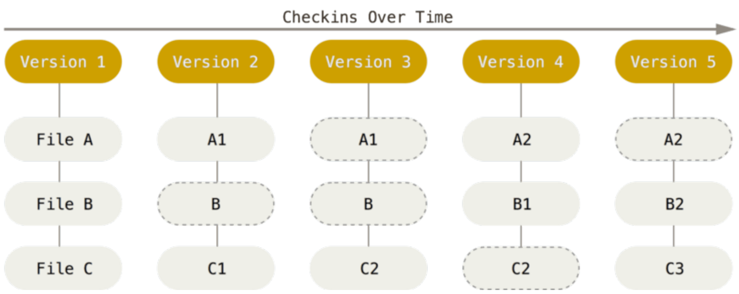
Git is a free and opensource version control system

- 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

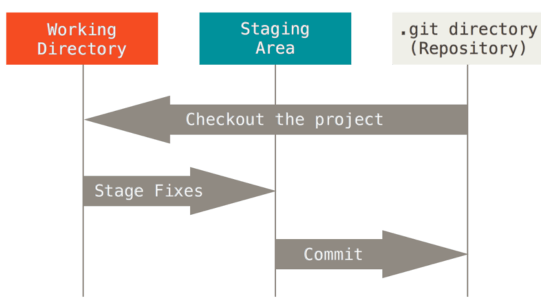


by Scott Chacon and Ben Straub, licensed under CC BY-NC-SA 3.0

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

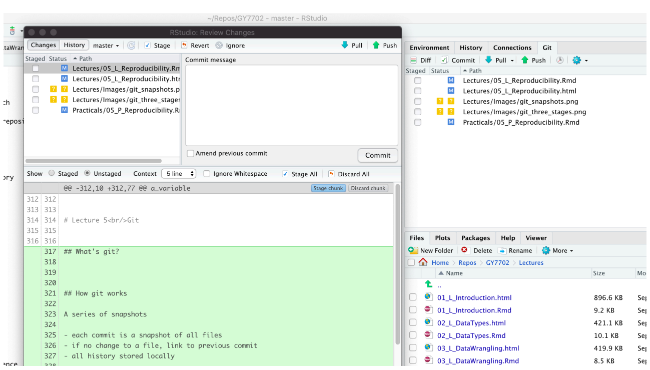
- `git clone`
  - copy a repository from a server
- `git fetch`
  - get the latest version from a branch
- `git pull`
  - incorporate changes from a remote repository
- `git add`
  - stage new files

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

15.6 Git and RStudio

RStudio includes a git plug-in

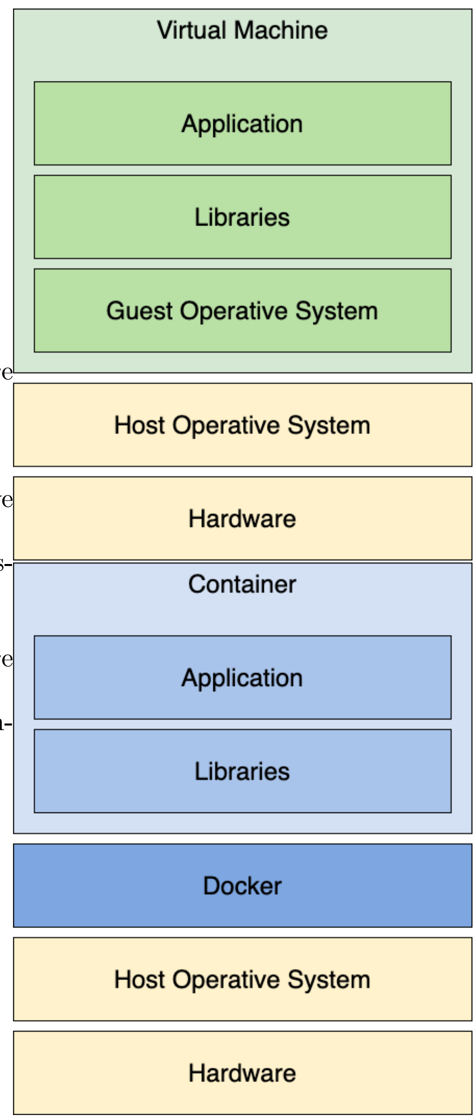
- clone R projects from repositories
- stage and commit changes
- push and pull changes



15.7 What's Docker?

Docker allows to encapsulate and share computational environments

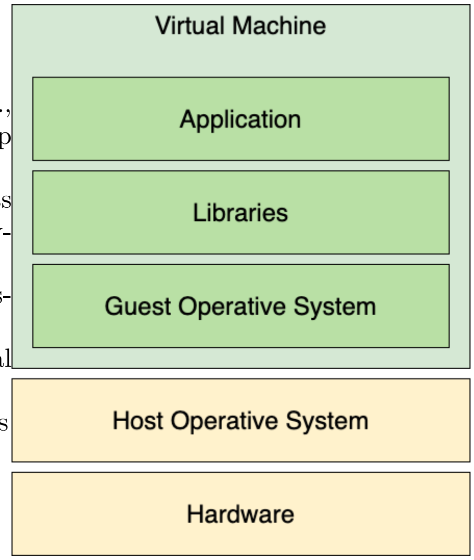
- First released in 2013
- Similar to virtual machines
  - simulates a guest operative system
  - within a host operative system
- Lightweight
  - doesn't simulate an entire system
  - only the *"user space"* is simulated



15.8 Virtual machines

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

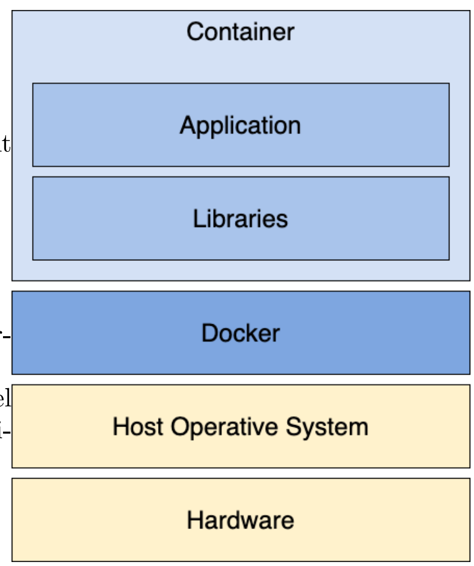
- allows **virtual machine** to access physical resources (e.g., disk, keyboard) of a **host**
- allows to run full operative systems
- e.g., run a full Windows virtual machine on a Mac host
- have been around since the 1970s
- can be *heavy* to run



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 `+`

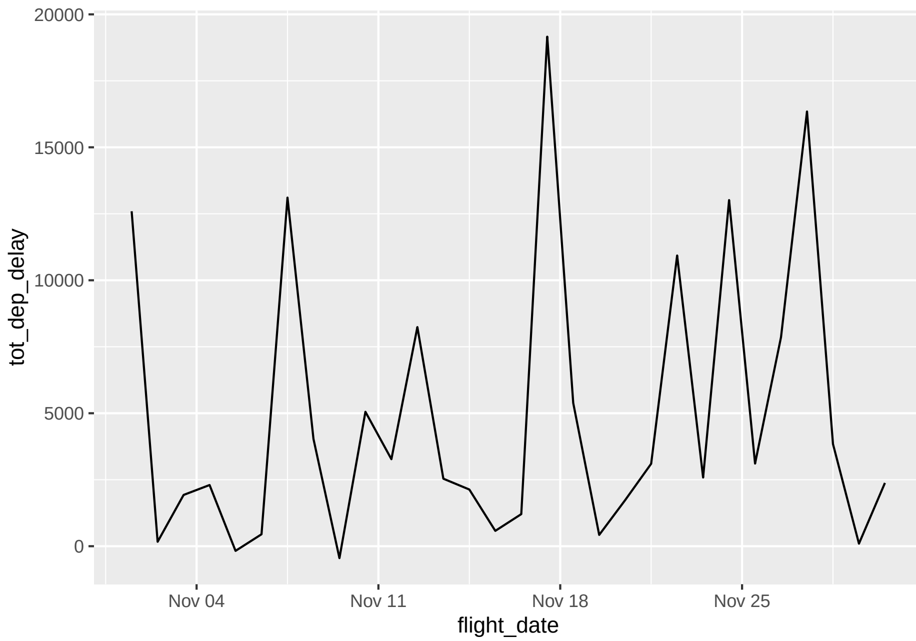
```
ggplot2::ggplot(
  aes(
    x = column_1, y = column_2
  )
) +
ggplot2::geom_line()
```

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

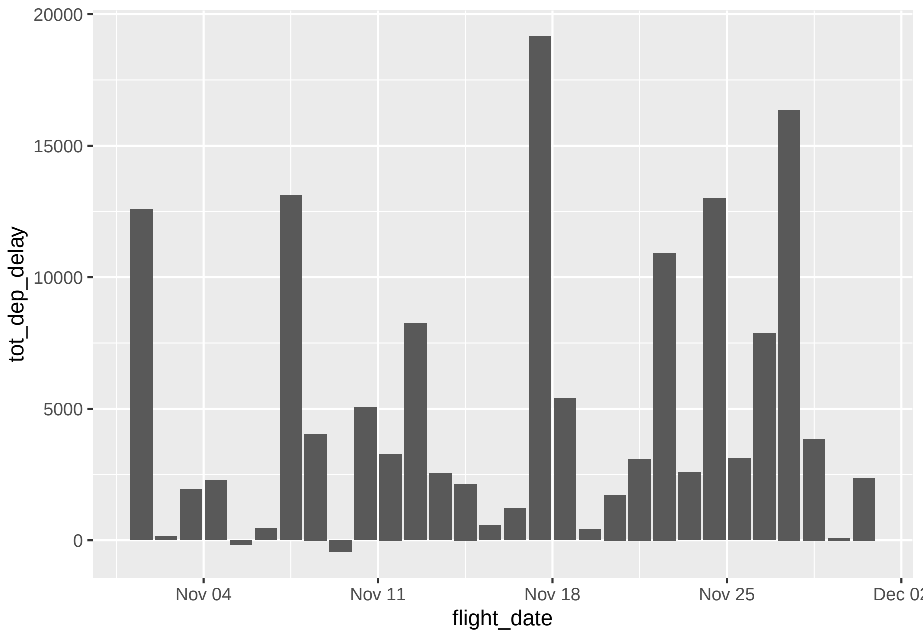


16.9 ggplot2::geom_col

- `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



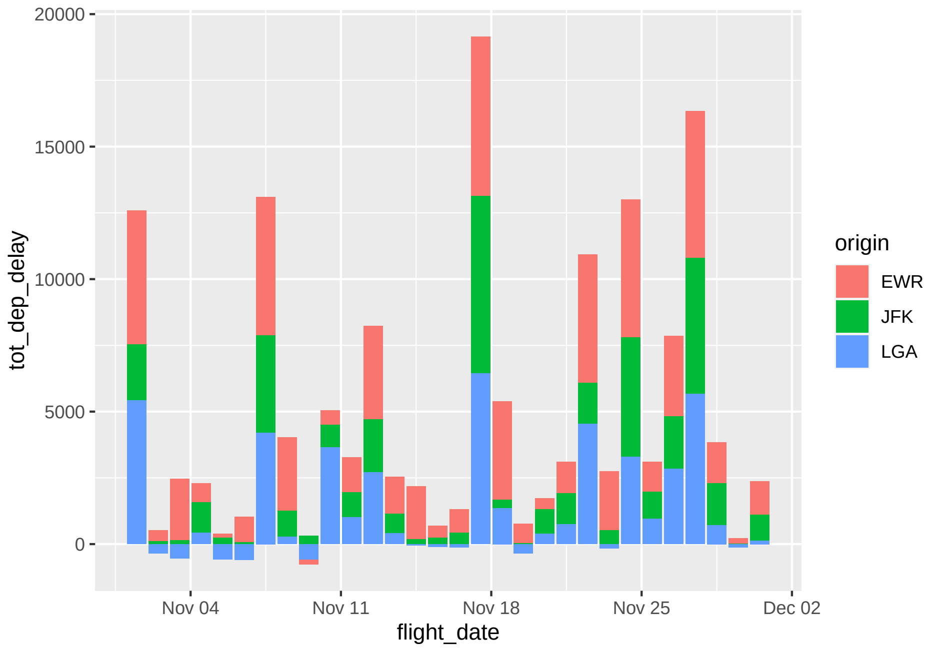
16.11 ggplot2::geom_col

… 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

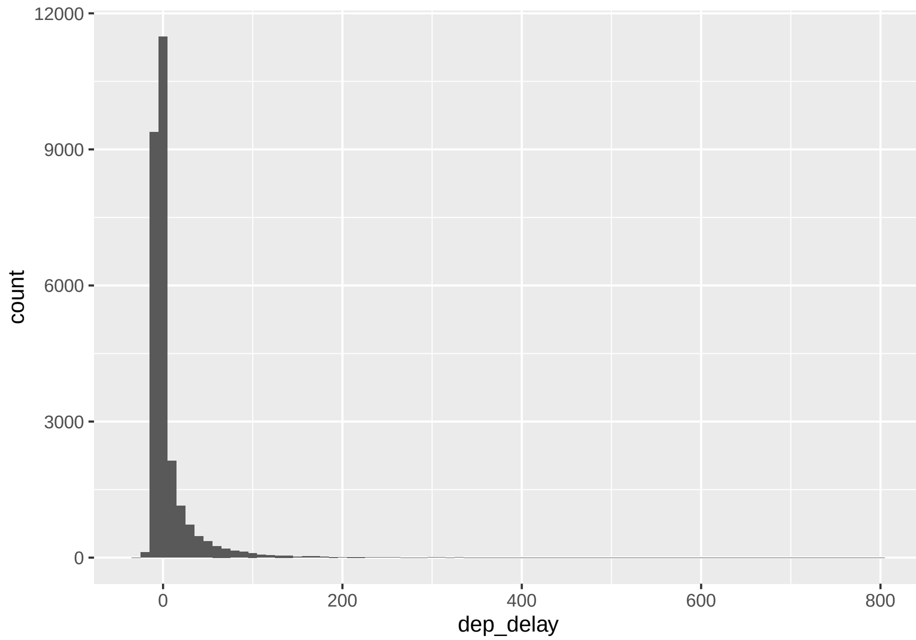


16.13 Histograms

- `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()
```

::::::

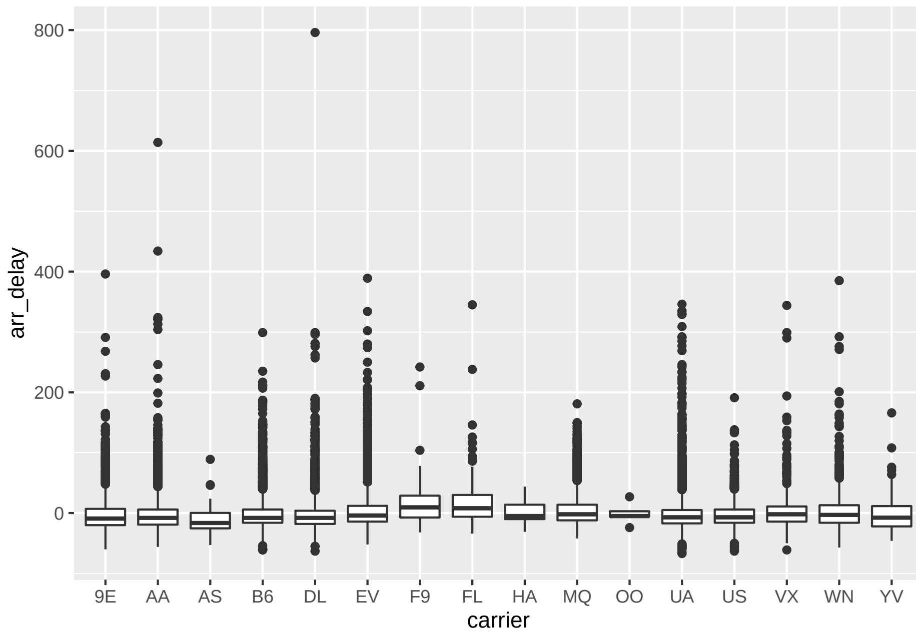
16.15 Boxplots

- `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

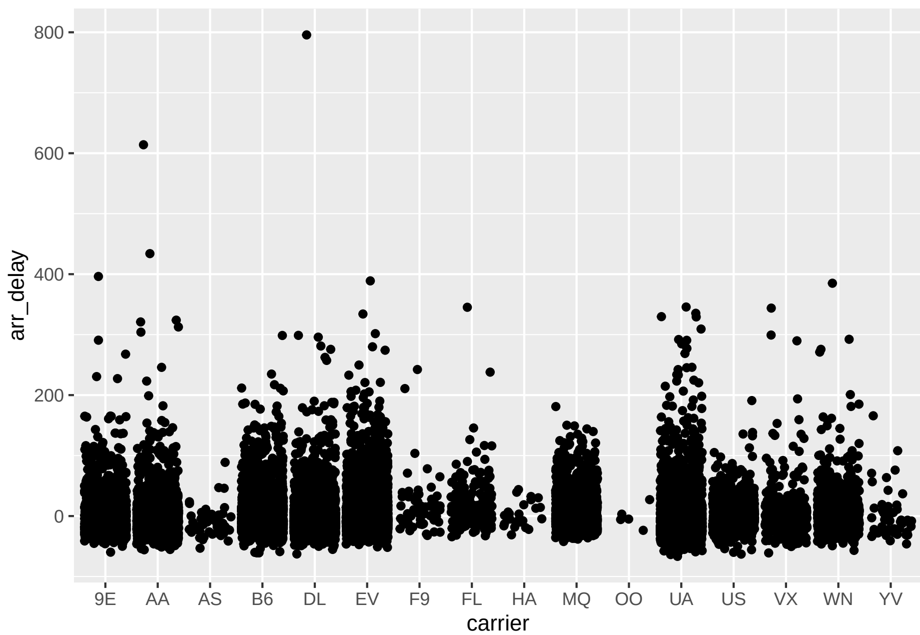


16.17 Jittered points

- 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

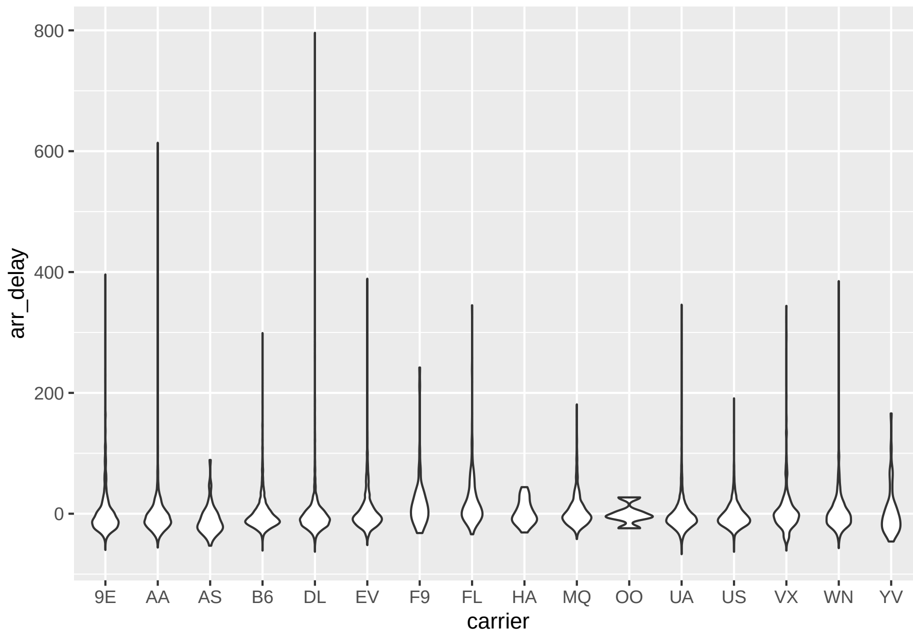


16.19 Violin plot

- 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



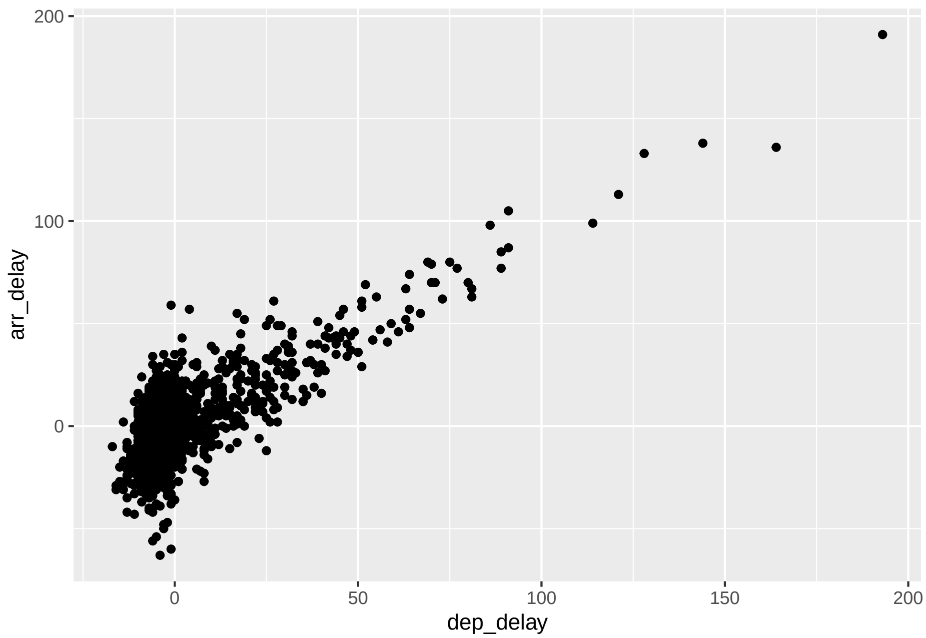
–>

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

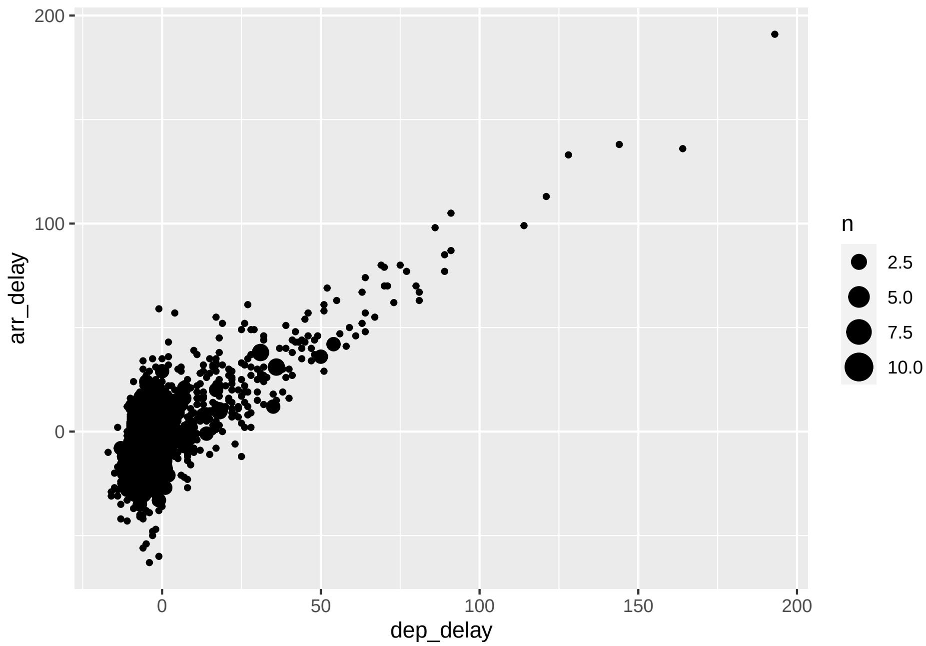


16.23 Overlapping points

- `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

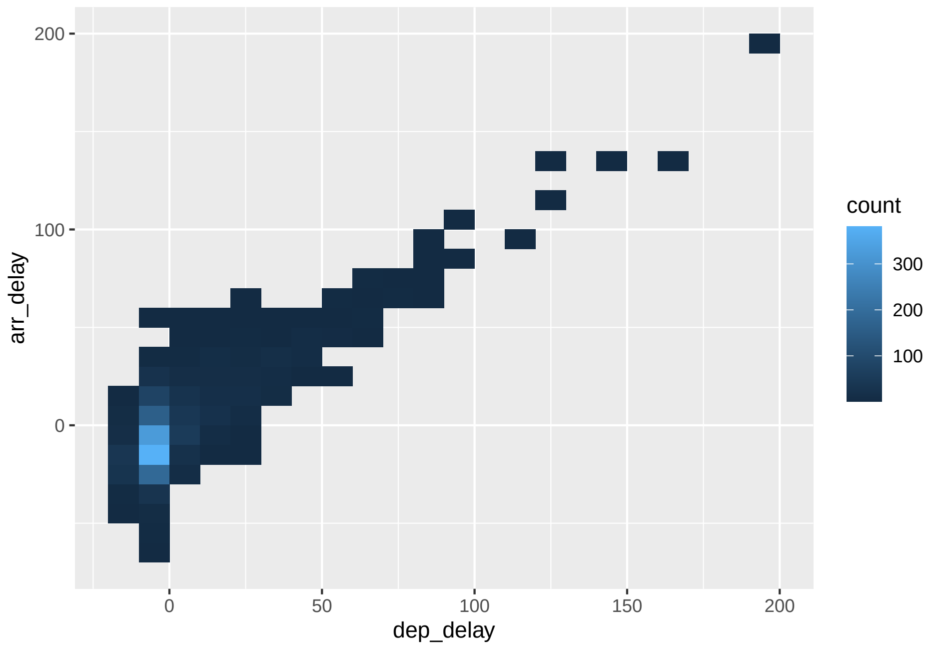


16.25 Bin counts

- 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

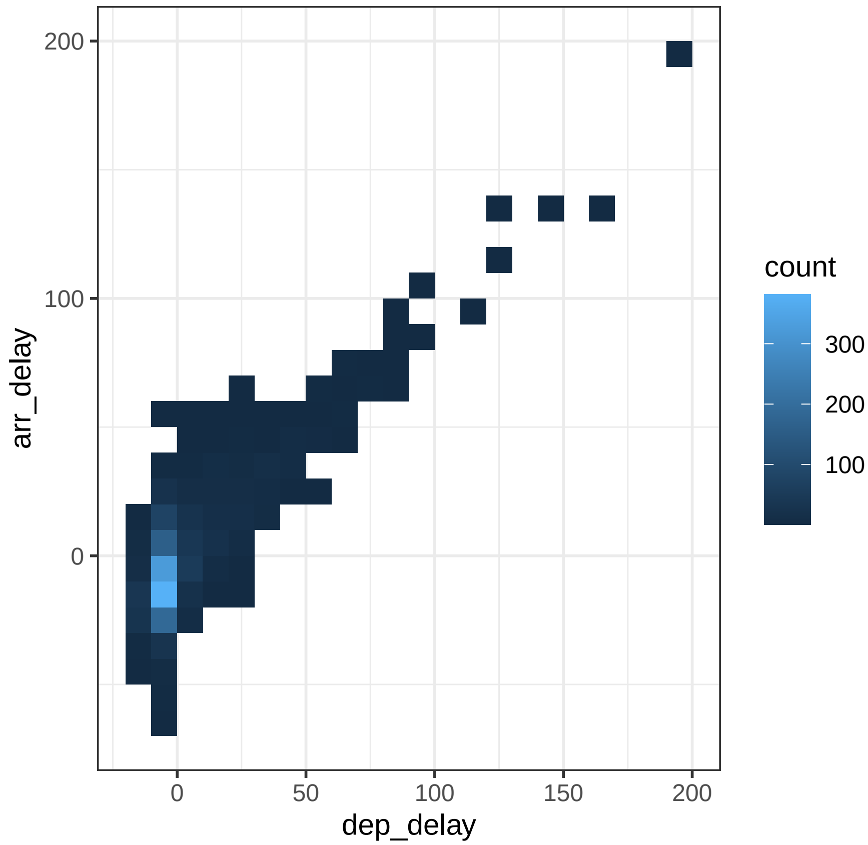


16.27 Coordinates transformations

- `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

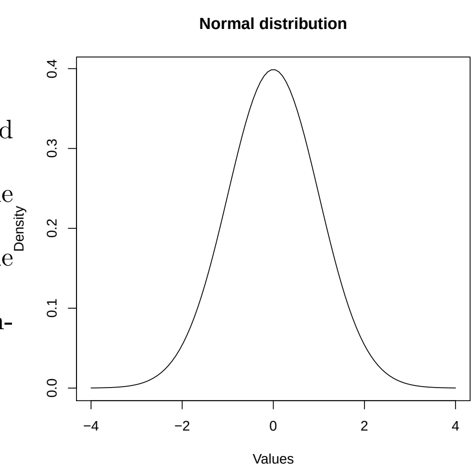
- stat.desc
- dplyr::across

Next: Exploring assumptions

- Normality
- Skewness and kurtosis
- Homogeneity of variance

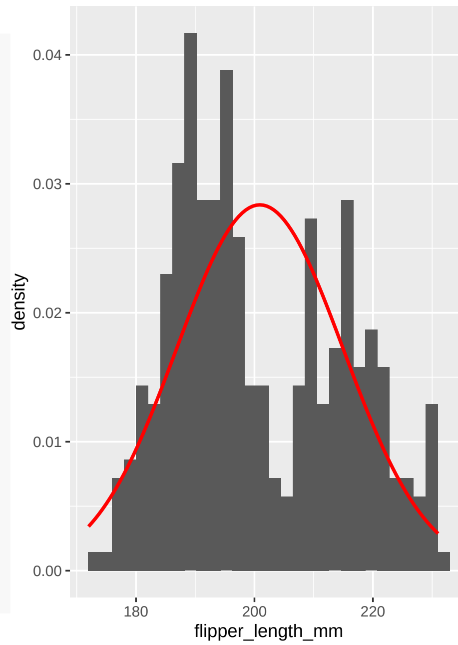
18.2 Normal distribution

- 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

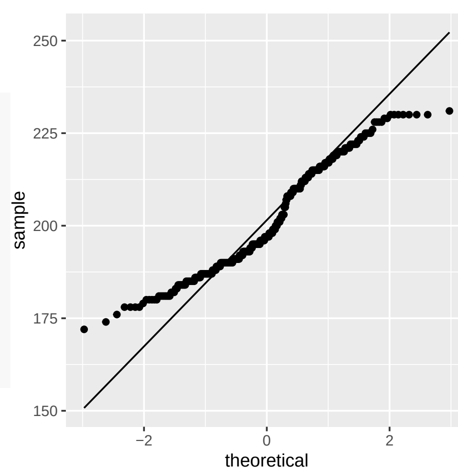
```
palmerpenguins::penguins %>%
  ggplot2::ggplot(
    aes(x = flipper_length_mm)
  ) +
  ggplot2::geom_histogram(
    aes(
      y =..density..
    )
  ) +
  ggplot2::stat_function(
    fun = dnorm,
    args = list(
      # mean and stddev
      # calculations
      # omitted here
      mean = ...,
      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 $p < .05$)

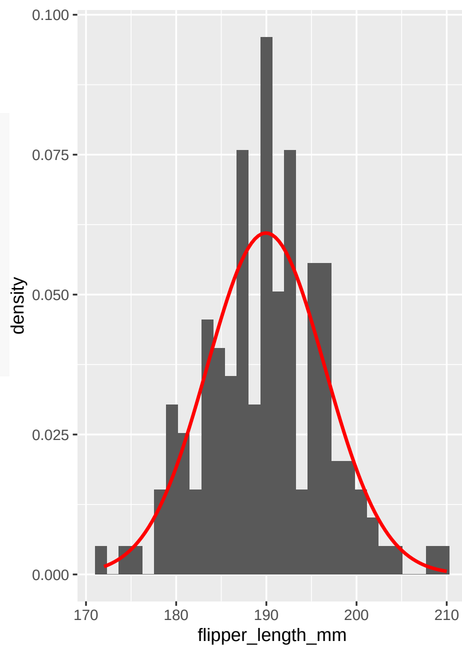
Example: The null hypotheis of the Shapiro–Wilk test is that the sample is normally distributed and $p < .01$ indicates that the probability of that being true is very low. So, the *flipper length* of penguins in the Palmer Station dataset **is not** normally distributed.

18.7 Example

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

```r
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
```

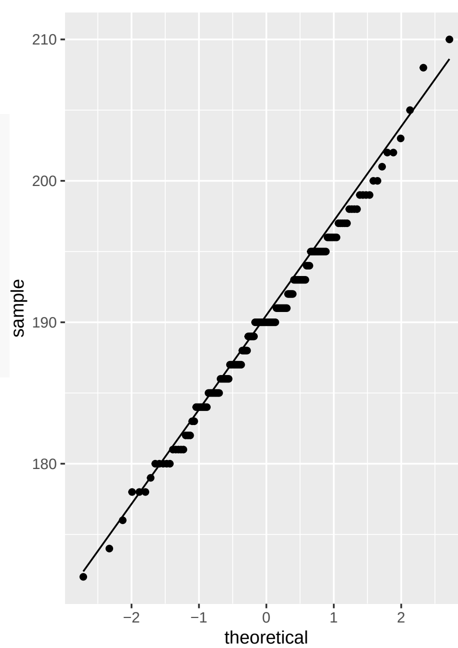


18.8 Example

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

```r
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)
```

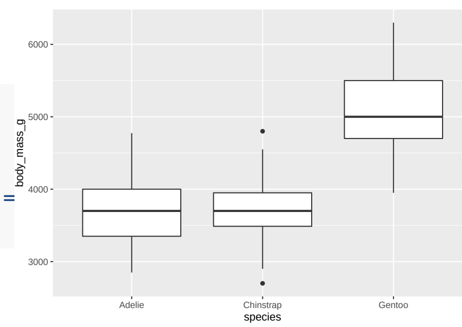
	bill_length_mm	bill_depth_mm	flipper_length_mm
skewness	0.1584764	0.3148847	0.0856093
skew.2SE	0.4014211	0.7976035	0.2168485
kurtosis	-0.2285951	-0.1361153	0.2382734
kurt.2SE	-0.2913388	-0.1734755	0.3036734
normtest.W	0.9933618	0.9846683	0.9933916
normtest.p	0.7166005	0.0924897	0.7200466

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

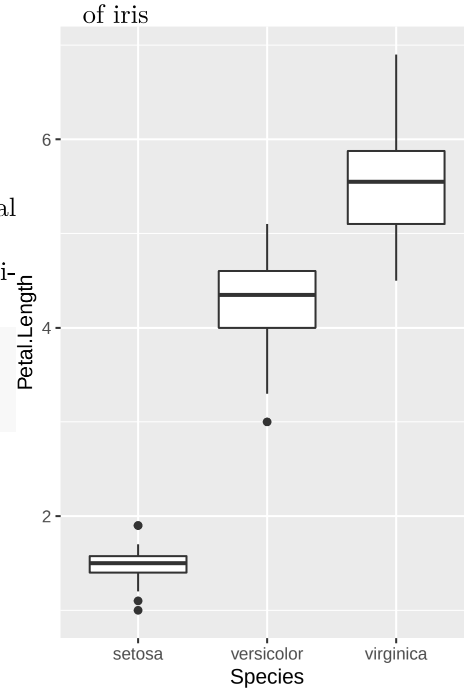
Now: Comparing groups

- T-test
- ANOVA
- Chi-square

19.2 Libraries

But let's start from a simple example from `datasets`

- 50 flowers from each of 3 species of iris



Today's libraries

- mostly working with the usual `nycflights13`
- exposition pipe `%$%` from the library `magrittr`

```
library(tidyverse)
library(magrittr)
library(nycflights13)
```

19.3 Example

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

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

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

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

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

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  .
## W = 0.96219, p-value = 0.1098
```

19.4 T-test

Independent T-test tests whether two group means are different

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

- groups defined by a predictor, categorical variable
- outcome is a continuous variable
- assuming
  - normally distributed values in groups
  - homogeneity of variance of values in groups
    * if groups have different sizes
  - independence of groups

19.5 Example

Values are normally distributed, groups have same size, and they are independent (different flowers, check using `leveneTest`)

```
iris %>%
  filter(Species %in% c("versicolor", "virginica")) %$% # Note %$%
  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
```

The difference is significant t(95.57) = -12.6, $p < .01$

19.6 ANOVA

ANOVA (analysis of variance) tests whether more than two group means are different

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

- groups defined by a predictor, categorical variable

- outcome is a continuous variable
- assuming
  - 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

19.7 Example

Values are normally distributed, groups have same size, they are independent (different flowers, check using `leveneTest`)

```
iris %$%
  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
```

The difference is significant $\mathrm{t}(2, 147) = 1180.16$, $p < .01$

19.8 Summary

Comparing groups

- T-test
- ANOVA
- Chi-square

Next: Correlation

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

Chapter 20

Correlation

20.1 Recap

Prev: Comparing groups

- T-test
- ANOVA
- Chi-square

Now: Correlation

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

20.2 Correlation

Two variables can be related in three different ways

- 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
- not related at all

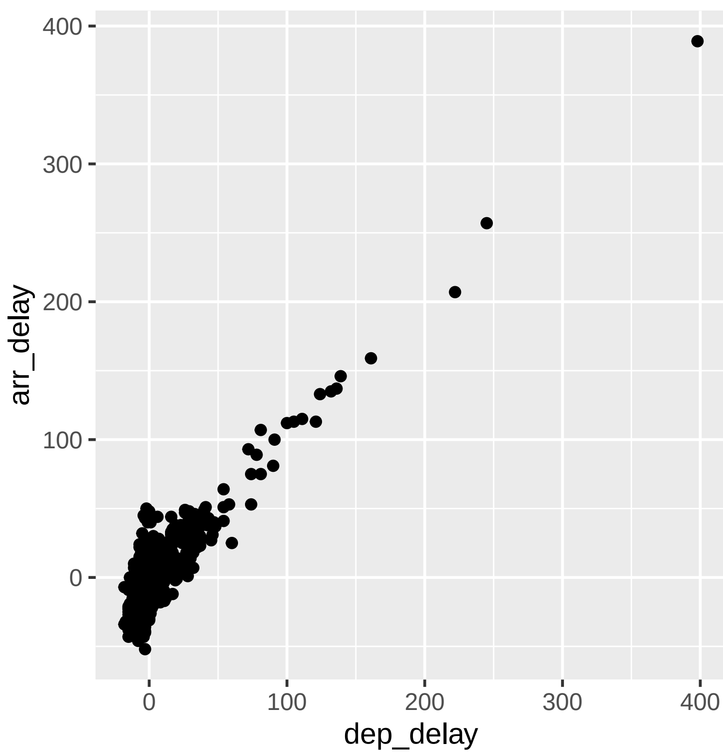
Correlation is a standardised measure of covariance

20.3 Libraries and data

```
library(tidyverse)
library(magrittr)
library(nycflights13)

flights_nov_20 <- nycflights13::flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay), month == 11, day ==20)
```

20.4 Example



20.5 Example

```
flights_nov_20 %>%
  pull(dep_delay) %>% shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.39881, p-value < 2.2e-16
```

```
flights_nov_20 %>%
  pull(arr_delay) %>% shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.67201, p-value < 2.2e-16
```

20.6 Pearson's r

If two variables are **normally distributed** use **Pearson's r**
The square of the correlation value indicates the percentage of shared variance
If they were normally distributed, but they are not

- 0.882 ^ 2 = 0.778
- departure and arrival delay *would* share 77.8% of variance

```
# note the use of %$%
#instead of %>%
flights_nov_20 %$%
  cor.test(dep_delay, arr_delay)
```

```
##
##  Pearson's product-moment correlation
##
## data:  dep_delay and arr_delay
## t = 58.282, df = 972, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8669702 0.8950078
## sample estimates:
##       cor
## 0.8817655
```

20.7 Spearman's rho

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

- non-parametric
- based on rank difference

The square of the correlation value indicates the percentage of shared variance
If few ties, but there are

- 0.536 ^ 2 = 0.287
- departure and arrival delay *would* share 28.7% of variance

```
flights_nov_20 %$%
  cor.test(
    dep_delay, arr_delay,
    method = "spearman")
```

```
## Warning in cor.test.default(dep_delay, arr_delay, method = "spearman"): Cann
## compute exact p-value with ties
##
##  Spearman's rank correlation rho
##
## data:  dep_delay and arr_delay
## S = 71437522, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.5361247
```

20.8 Kendall's tau

If **not normally distributed** and there is a **large number of ties**, use **Kendall's tau**

- non-parametric
- based on rank difference

The square of the correlation value indicates the percentage of shared variance
Departure and arrival delay seem actually to share

- 0.396 ^ 2 = 0.157
- **15.7% of variance**

```
flights_nov_20 %$%
  cor.test(
    dep_delay, arr_delay,
    method = "kendall")
```

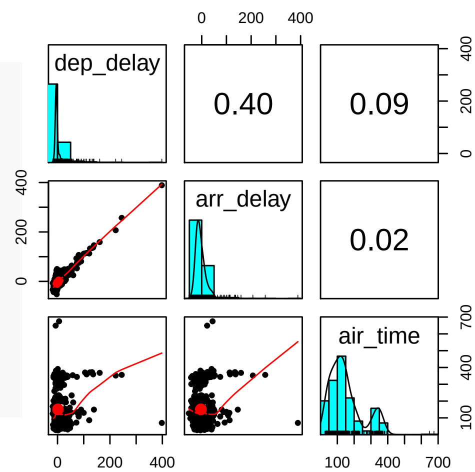
```
##
##  Kendall's rank correlation tau
##
## data:  dep_delay and arr_delay
## z = 17.859, p-value < 2.2e-16
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.3956265
```

20.9 Pairs plot

Combines in one visualisation: histograms, scatter plots, and correlation values for a set of variables

```
library(psych)

flights_nov_20 %>%
  select(
    dep_delay,
    arr_delay,
    air_time
  ) %>%
  pairs.panels(
    method = "kendall"
  )
```



20.10 Summary

Correlation

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

Next: Data transformations

- Z-scores
- Logarithmic transformations

Chapter 21

Data transformations

21.1 Recap

Prev: Correlation

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

Now: Data transformations

- Z-scores
- Logarithmic transformations

21.2 Libraries and data

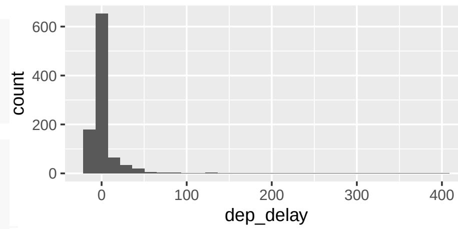
```
library(tidyverse)
library(magrittr)
library(nycflights13)

flights_nov_20 <- nycflights13::flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay), month == 11, day ==20)
```

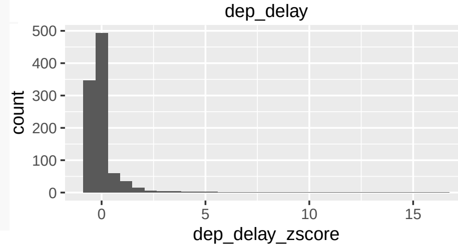
21.3 Z-scores

Z-scores transform the values as relative to the distribution mean and standard deviation

```
flights_nov_20 %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram()
```



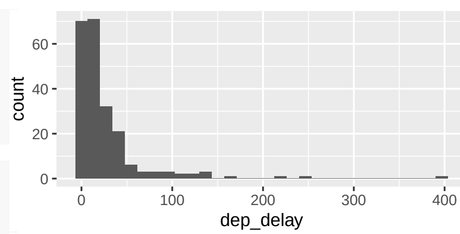

```
flights_nov_20 %>%
  mutate(
    dep_delay_zscore =
      scale(dep_delay)
  ) %>%
  ggplot(
    aes(x = dep_delay_zscore)
  ) +
  geom_histogram()
```



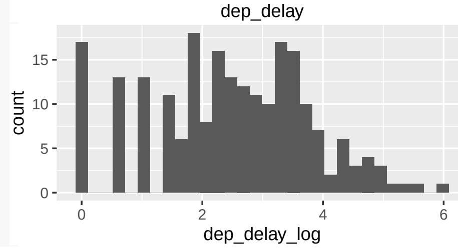
21.4 Log transformation

Logarithmic transformations (e.g., `log` and `log10`) are useful to *"un-skew"* variables, but only possible on values `> 0`

```
flights_nov_20 %>%
  filter(dep_delay > 0) %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram()
```



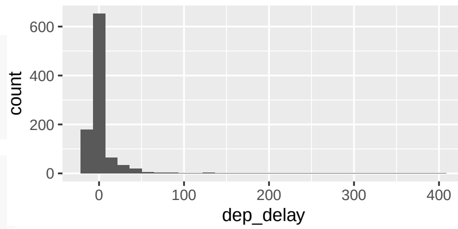

```
flights_nov_20 %>%
  filter(dep_delay > 0) %>%
  mutate(
    dep_delay_log =
      log(dep_delay)
  ) %>%
  ggplot(
    aes(x = dep_delay_log)) +
  geom_histogram()
```



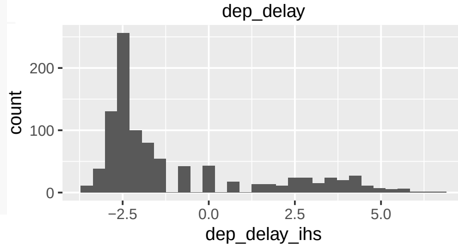
21.5 Inverse hyperbolic sine

Inverse hyperbolic sine (`asinh`) transformations are useful to *"un-skew"* variables, similar to logarithmic transformations, work on all values

```r
flights_nov_20 %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram()
```




```r
flights_nov_20 %>%
  mutate(
    dep_delay_ihs =
      asinh(dep_delay)
  ) %>%
  ggplot(
    aes(x = dep_delay_ihs)) +
  geom_histogram()
```



21.6 Summary

Data transformations

- Z-scores
- Logarithmic 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
- Fit

22.2 Regression analysis

Regression analysis is a supervised machine learning approach

Predict the value of one outcome variable as

$$outcome_i = (model) + error_i$$

- one predictor variable (**simple / univariate** regression)

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

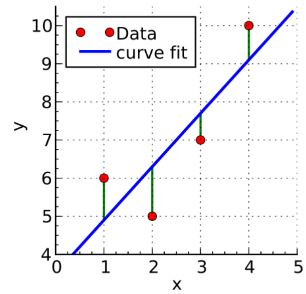
- more predictor variables (**multiple / multivariate** regression)

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

22.3 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

$$deviation = \sum(observed - model)^2$$

22.4 Libraries and data

```
library(tidyverse)
library(magrittr)
library(nycflights13)

flights_nov_20 <- nycflights13::flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay), month == 11, day ==20)
```

22.5 Example

$$arr\_delay_i = (b_0 + b_1 * dep\_delay_{i1}) + \epsilon_i$$

```
delay_model <- flights_nov_20 %$% # Note %$%
  lm(arr_delay ~ dep_delay)

delay_model %>%  summary()
```

```
##
## Call:
## lm(formula = arr_delay ~ dep_delay)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.906  -9.022  -1.758   8.678  57.052
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.96717    0.43748  -11.35   <2e-16 ***
## dep_delay    1.04229    0.01788   58.28   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.62 on 972 degrees of freedom
## Multiple R-squared:  0.7775,Adjusted R-squared:  0.7773
## F-statistic:  3397 on 1 and 972 DF,  p-value: < 2.2e-16
```

22.6 Overall fit

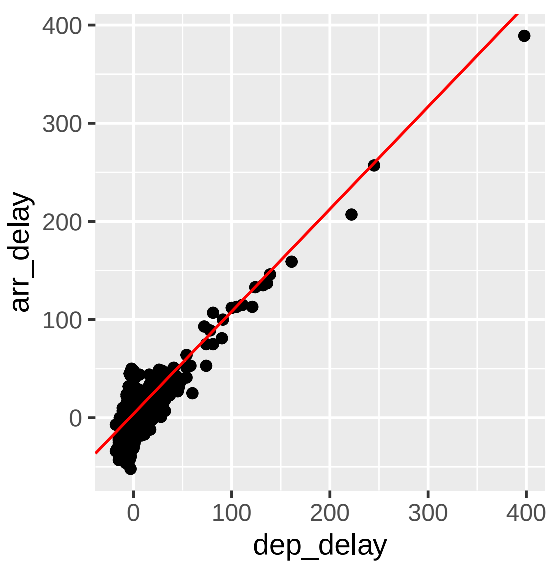
The output indicates

- **p-value: < 2.2e-16**: $p < .001$ the model is significant
  - derived by comparing the calulated **F-statistic** value to F distribution 3396.74 having specified degrees of freedom (1, 972)
  - Report as: F(1, 972) = 3396.74
- **Adjusted R-squared: 0.7773**: the departure delay can account for 77.73% of the arrival delay
- **Coefficients**
  - Intercept estimate -4.9672 is significant
  - `dep_delay` (slope) estimate 1.0423 is significant

22.7 Parameters

$$arr\_delay_i = (Intercept + Coefficient_{dep\_delay} * dep\_delay_{i1}) + \epsilon_i$$

```
flights_nov_20 %>%
  ggplot(aes(x = dep_delay, y = arr_delay)) +
  geom_point() + coord_fixed(ratio = 1) +
  geom_abline(intercept = 4.0943, slope = 1.04229, color="red")
```



22.8 Summary

Simple Regression

- Regression
- Ordinary Least Squares
- Fit

Next: Assessing regression assumptions

- Normality
- Homoscedasticity
- Independence

Chapter 23

Assessing regression assumptions

23.1 Recap

Prev: Simple Regression

- Regression
- Ordinary Least Squares
- Fit

Now: Assessing regression assumptions

- Normality
- Homoscedasticity
- Independence

23.2 Checking 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

23.3 Libraries and data

```
library(tidyverse)
library(magrittr)
library(nycflights13)

flights_nov_20 <- nycflights13::flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay), month == 11, day ==20)
```

23.4 Example

$$arr\_delay_i = (b_0 + b_1 * dep\_delay_{i1}) + \epsilon_i$$

```
delay_model <- flights_nov_20 %$% # Note %$%
  lm(arr_delay ~ dep_delay)

delay_model %>%  summary()
```

```
##
## Call:
## lm(formula = arr_delay ~ dep_delay)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.906  -9.022  -1.758   8.678  57.052
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.96717    0.43748  -11.35   <2e-16 ***
## dep_delay    1.04229    0.01788   58.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.62 on 972 degrees of freedom
## Multiple R-squared:  0.7775,Adjusted R-squared:  0.7773
## F-statistic:  3397 on 1 and 972 DF,  p-value: < 2.2e-16
```

23.5 Normality

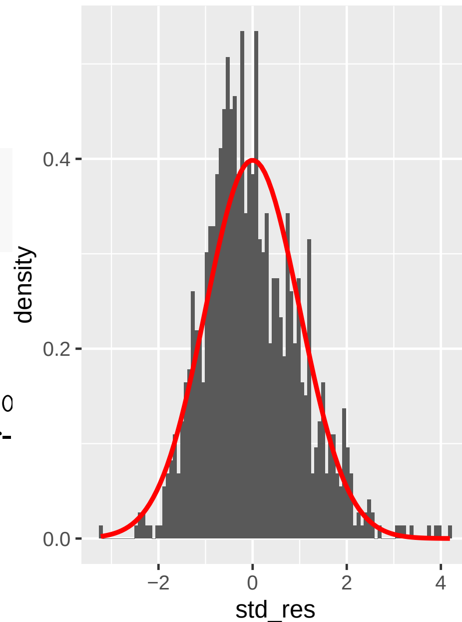
Shapiro-Wilk test for normality of standard residuals,

- robust models: should be not significant

```r
delay_model %>%
  rstandard() %>%
  shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.98231, p-value = 1.73e-0
```

Standard residuals are NOT normally distributed



23.6 Homoscedasticity

Breusch-Pagan test for homoscedasticity of standard residuals

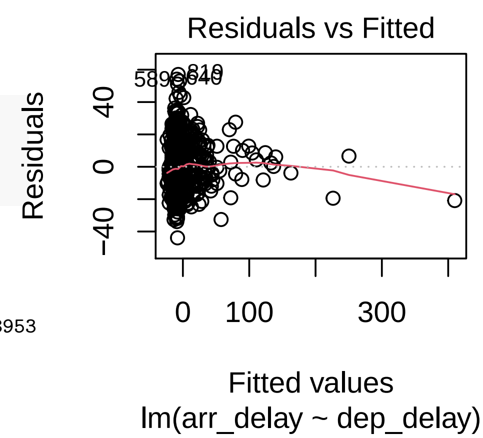
- robust models: should be not significant

```r
library(lmtest)

delay_model %>%
  bptest()
```

```
##
##  studentized Breusch-Pagan test
##
## data:  .
## BP = 0.017316, df = 1, p-value = 0.8953
```

Standard residuals are homoscedastic



23.7 Independence

Durbin-Watson test for the independence of residuals

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

```r
# Also part of the library lmtest
delay_model %>%
  dwtest()
```

```
##
##  Durbin-Watson test
##
## data:  .
## DW = 1.8731, p-value = 0.02358
## alternative hypothesis: true autocorrelation is greater than 0
```

Standard residuals might not be completely indipendent

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

23.8 Summary

Assessing regression assumptions

- Normality
- Homoscedasticity
- Independence

Next: Assessing regression assumptions

- Normality
- Homoscedasticity
- Independence

Chapter 24

Multiple Regression

24.1 Recap

Prev: Assessing regression assumptions

- Normality
- Homoscedasticity
- Independence

Now: Multiple Regression

- Fit
- Multicollinearity
- Comparing models

24.2 TO-DO

24.3 Summary

Multiple Regression

- Fit
- Multicollinearity
- Comparing models

Next: Practical session

- Simple regression
- Testing assumptions
- Multiple regression

Chapter 25

Machine Learning

25.1 Recap

Prev: Comparing data

- 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 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

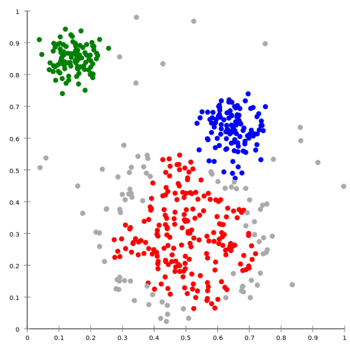
25.5 Supervised

- 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,

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



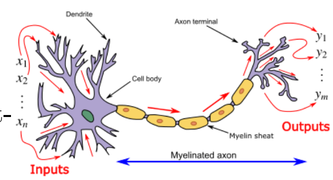
25.7 ... more

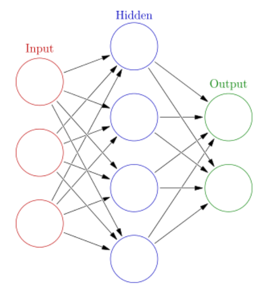
- **Semi-supervised learning**
  - between unsupervised and supervised learning
  - combines a small amount of labelled data with a larger un-labelled dataset
  - continuity, cluster, and manifold (lower dimensionality) assumption
- **Reinforcement learning**
  - training agents take actions to maximize reward
  - balancing
    - exploration (new paths/options)
    - exploitation (of current knowledge)

25.8 Neural networks

Supervised learning approach simulating simplistic neurons

- Classic model with 3 sets
  - input neurons
  - output neurons
  - hidden layer
    - combines input values using **weights**
    - **activation function**
- The **traning algorithm** is used to define the best weights

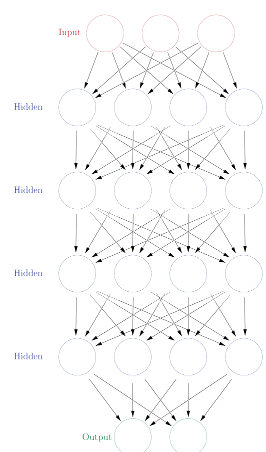




by Egm4313.s12 and Glosser.ca via Wikimedia Commons, CC-BY-SA-3.0

25.9 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

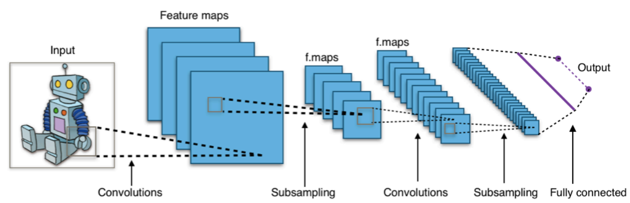
25.10 Convolutional neural networks

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

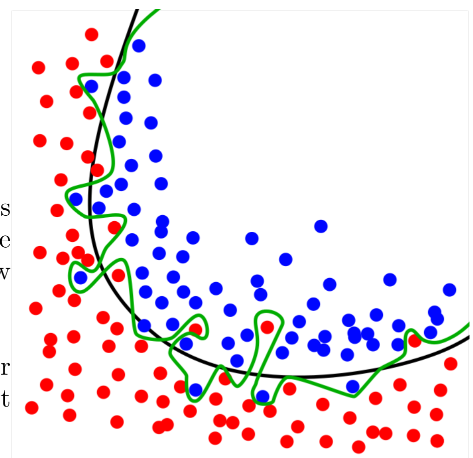
Example: Liu, P. and De Sabbata, S. (2019). Learning Digital Geographies through a Graph-Based Semi-supervised Approach. In proceedings of the 15th International Conference on GeoComputation, Queenstown, New Zealand.



by Aphex34 via Wikimedia Commons, CC-BY-SA-4.0

25.11 Limits

- Complexity
- Training dataset quality
  - garbage in, garbage out
  - e.g., Facial Recognition Is Accurate, if You're a White Guy by Steve Lohr (New York Times, Feb. 9, 2018)
- Overfitting
  - creating a model perfect for the training data, but not generic enough to be useful



by Chabacanoia Wikimedia Commons,CC-BY-SA-4.0

25.12 Summary

Machine Learning

- What's Machine Learning?
- Types
- Limitations

Next: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

Chapter 26

Deep learning

26.1 Recap

Prev: Machine learning

- X
- Y
- Z

Now: Artificial Neural Networks

- X
- Y
- Z

26.2 TO-DO

26.3 Summary

Artificial Neural Networks

- X
- Y
- Z

Next: Support vector machines

-
-
-

Chapter 27

Support vector machines

27.1 Recap

Prev: Artificial Neural Networks

- X
- Y
- Z

Now: Support vector machines

- X
- Y
- Z

27.2 TO-DO

27.3 Summary

Support vector machines

- X
- Y
- Z

Next: Practical session

- X
- Y
- Z

Chapter 28

Principal Component Analysis

28.1 Recap

Prev: Comparing data

- 321 Lecture Introduction to Machine Learning
- 322 Lecture
- 323 Lecture
- 324 Practical session

Now: Principal Component Analysis

- X
- Y
- Z

28.2 TO-DO

28.3 Summary

Principal Component Analysis

- X
- Y
- Z

Next: Centroid-based clustering

- K-means
- Fuzzy c-means

- Geodemographic classification

Chapter 29

Centroid-based clustering

29.1 Recap

Prev: Principal Component Analysis

-
-
-

Now: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification

29.2 Clustering task

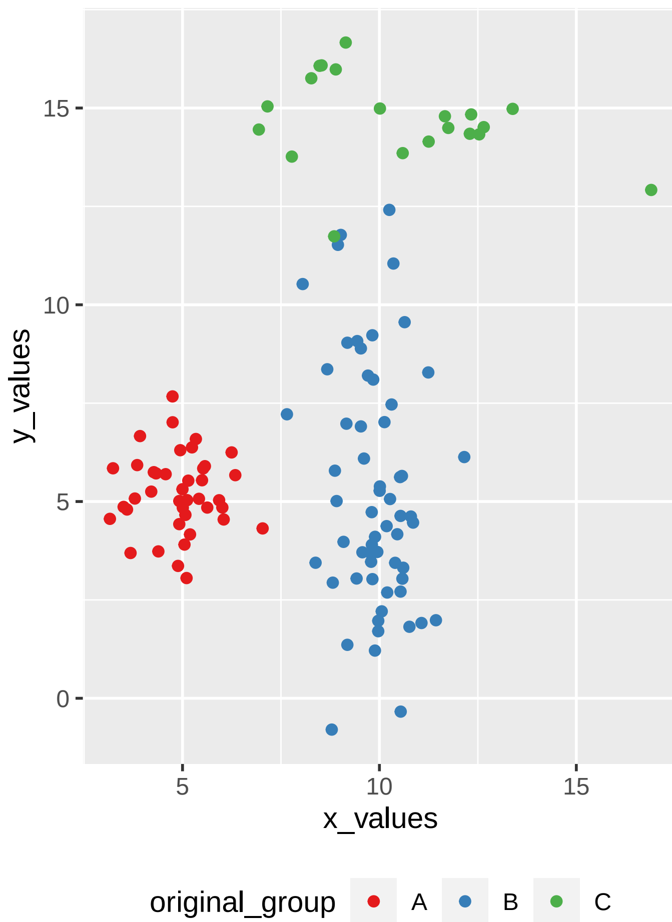
"Clustering is an unsupervised machine learning task that automatically divides the data into ***clusters*** *, or groups of similar items".* (Lantz, 2019)

Methods:

- Centroid-based
  - k-means
  - fuzzy c-means
- Hierarchical
- Mixed
  - bootstrap aggregating
- Density-based
  - DBSCAN

29.3 Example

```
data_to_cluster <- data.frame(
  x_values = c(rnorm(40, 5, 1), rnorm(60, 10, 1), rnorm(20, 12, 3)),
  y_values = c(rnorm(40, 5, 1), rnorm(60, 5, 3), rnorm(20, 15, 1)),
  original_group = c(rep("A", 40), rep("B", 60), rep("C", 20)) )
```



29.4 k-means

k-mean clusters `n` observations in `k` clusters, minimising the within-cluster sum of squares (WCSS)

Algorithm: `k` observations a randomly selected as initial centroids, then repeat

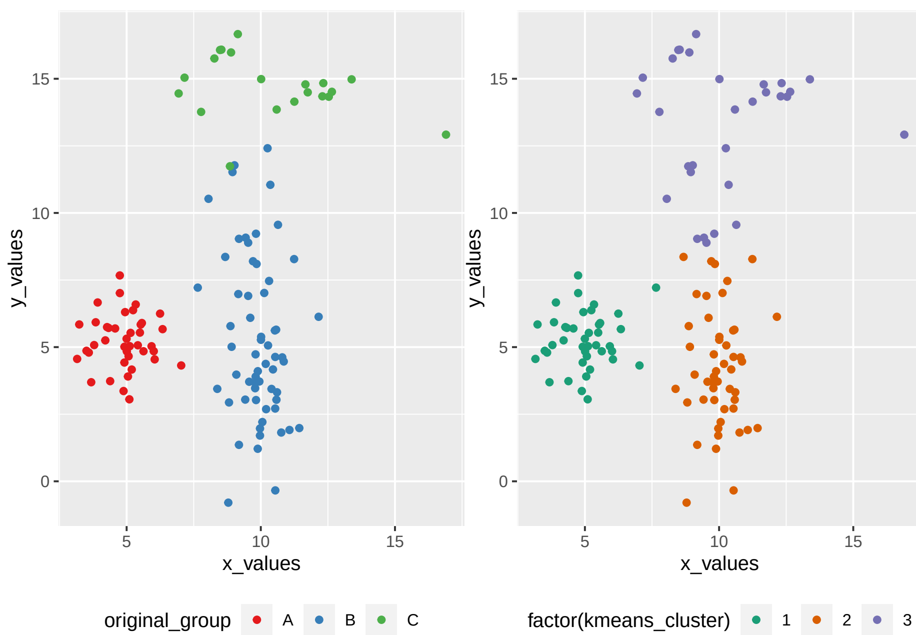
- **assignment step**: observations are assigned to the closest centroids
- **update step**: calculate means for each cluster to use as new the centroid

until centroids don't change anymore, the algorithm has **converged**

```
kmeans_found_clusters <- data_to_cluster %>%
  select(x_values, y_values) %>%
  kmeans(centers=3, iter.max=50)

data_to_cluster <- data_to_cluster %>%
  add_column(kmeans_cluster = kmeans_found_clusters$cluster)
```

29.5 K-means result



29.6 Fuzzy c-means

Fuzzy c-means is 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)

cmeans_result <- data_to_cluster %>%
  select(x_values, y_values) %>%
```

```
  cmeans(centers=3, iter.max=50)

data_to_cluster <- data_to_cluster %>%
  add_column(c_means_assigned_cluster = cmeans_result$cluster)
```

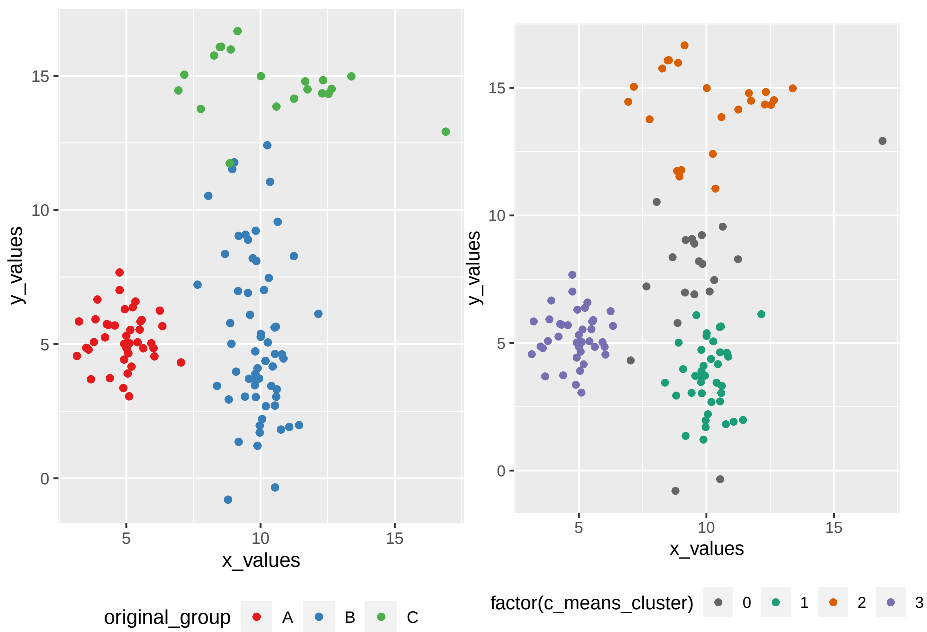
29.7 Fuzzy c-means

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

```
data_to_cluster <- data_to_cluster %>%
  add_column(
    c_means_membership = apply(cmeans_result$membership, 1, max)
  ) %>%
  mutate(
    c_means_cluster = ifelse(
      c_means_membership > 0.75,
      c_means_assigned_cluster,
      0
    )
  )
```

29.8 Fuzzy c-means result



29.9 Geodemographic classifications

In GIScience, the clustering is commonly used to create *geodemographic classifications* such as the 2011 Output Area Classification (Gale *et al.*, 2016)

- initial set of 167 prospective variables from the United Kingdom Census 2011
  - 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.

29.10 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

30.2 Libraries

```
library(tidyverse)
```

```
## -- 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()
```

```
library(e1071)
library(dbscan)
```

30.3 Example

```
data_to_cluster <- data.frame(
  x_values = c(rnorm(40, 5, 1), rnorm(60, 10, 1), rnorm(20, 12, 3)),
  y_values = c(rnorm(40, 5, 1), rnorm(60, 5, 3), rnorm(20, 15, 1)),
  original_group = c(rep("A", 40), rep("B", 60), rep("C", 20)) )
```

30.4 Hierarchical clustering

Algorithm: each object is initialised as, 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

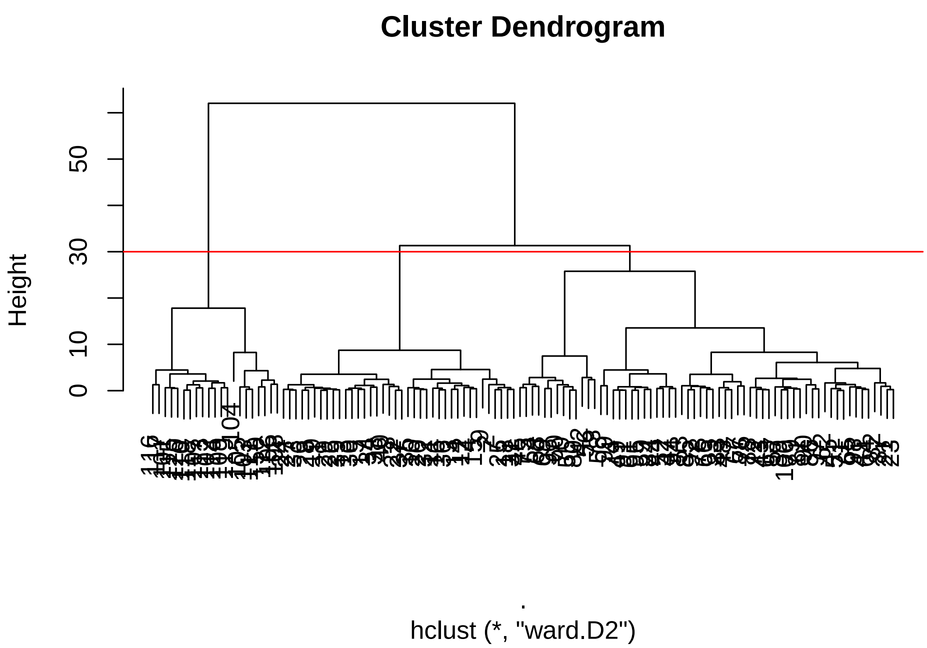
```
hclust_result <- data_to_cluster %>%
  select(x_values, y_values) %>%
  dist(method="euclidean") %>%
  hclust(method="ward.D2")

data_to_cluster <- data_to_cluster %>%
  add_column(hclust_cluster = cutree(hclust_result, k=3))
```

30.5 Clustering tree

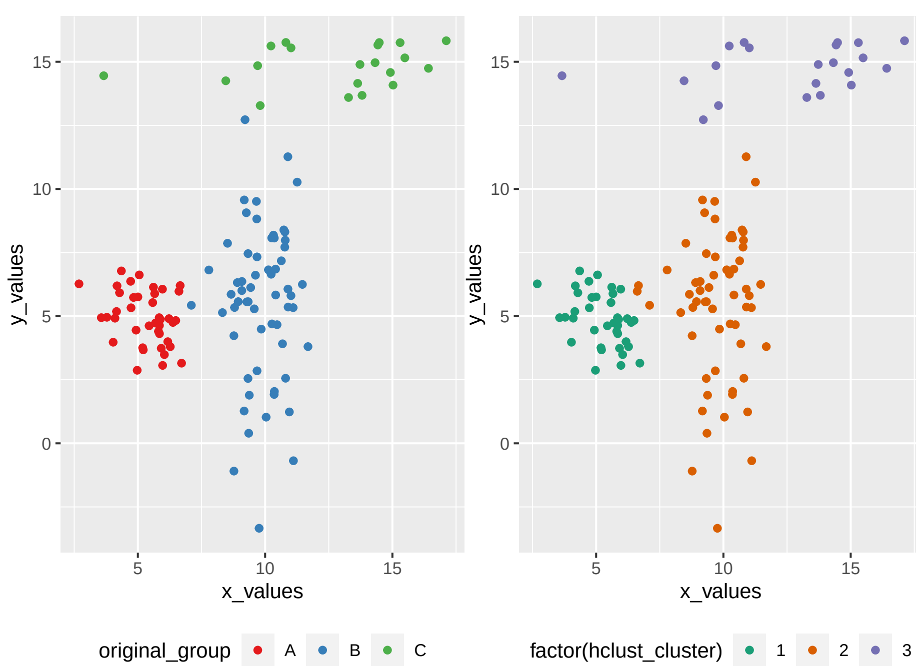
This approach generates a clustering tree (dendrogram), which can then be *"cut"* at the desired height

```
plot(hclust_result) + abline(h = 30, col = "red")
```




```
## integer(0)
```

30.6 Hierarchical clustering result



30.7 Bagged clustering

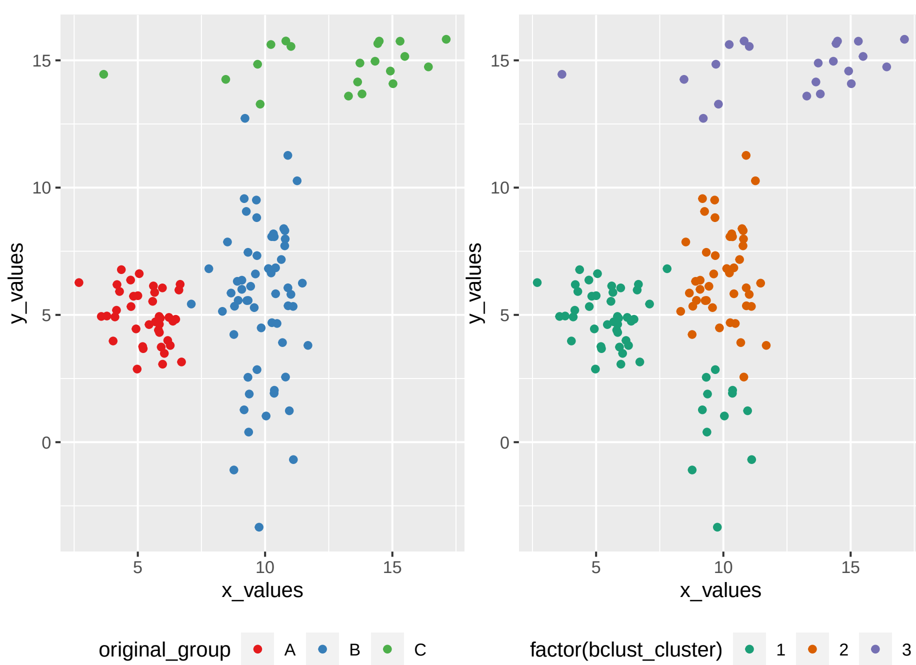
Bootstrap aggregating (*b-agg-ed*) clustering approach (Leisch, 1999)

- first k-means on samples
- then a hierarchical clustering of the centroids generated through the samples

```
bclust_result <- data_to_cluster %>%
  select(x_values, y_values) %>%
  bclust(hclust.method="ward.D2", resample = TRUE)

data_to_cluster <- data_to_cluster %>%
  add_column(bclust_cluster = clusters.bclust(bclust_result, 3))
```

30.8 Bagged clustering result



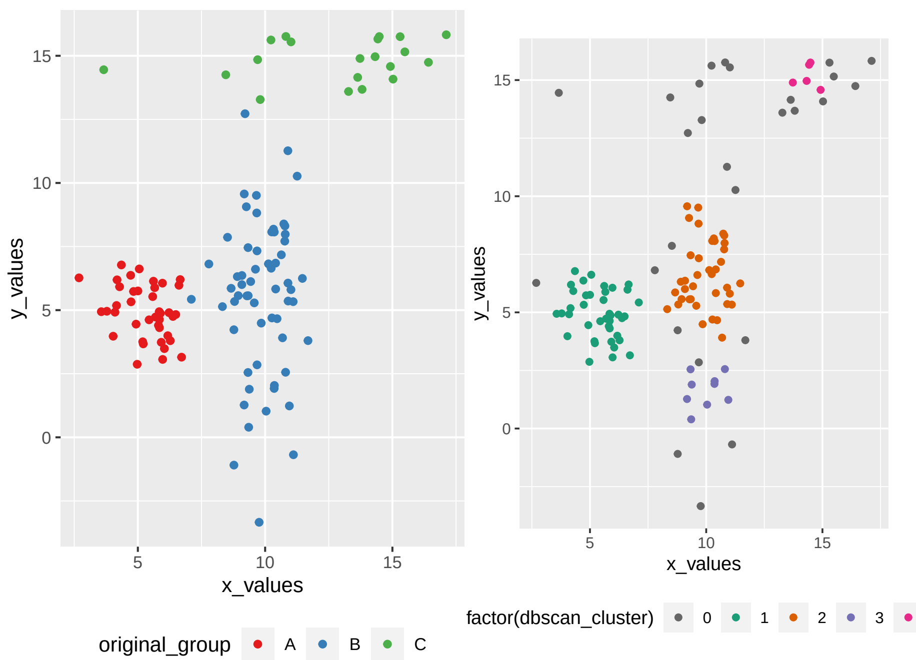
30.9 Density based clustering

DBSCAN (*"density-based spatial clustering of applications with noise"*) starts from an unclustered point and proceeds by aggregating its neighbours to the same cluster, as long as they are within a certain distance. (Ester *et al*, 1996)

```
dbscan_result <- data_to_cluster %>%
  select(x_values, y_values) %>%
   dbscan(eps = 1, minPts = 5)

data_to_cluster <- data_to_cluster %>%
  add_column(dbscan_cluster = dbscan_result$cluster)
```

30.10 DBSCAN result



30.11 Summary

Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

Next: Practical session

- Geodemographic classification