Lecture materials | granolarr

Stefano De Sabbata

2020-09-10

Contents

Preface

Stefano De Sabbata

This work is licensed under the GNU General Public License v3.0. Contains public sector information licensed under the Open Government Licence v3.0.

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

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

Session info

```
sessionInfo()
```

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

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

Chapter 1

Introduction to R

1.1 About this module

This module will provide you with the fundamental skills in

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

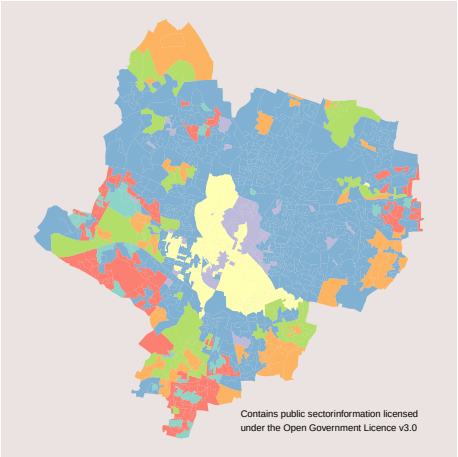
basis for

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

1.2 R programming language

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

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



1.3 Schedule

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

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

1.4 Reference books

Suggested reading

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

Further reading

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

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

1.5 R

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

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

1.6 Interpreting values

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

```
2
```

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

```
"String value"
```

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

```
# comments are ignored
```

1.7 Basic types

R provides three core data types

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

1.8 Numeric operators

R provides a series of basic numeric operators

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

```
5 + 2
```

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

1.9 Logical operators

R provides a series of basic logical operators to test

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

```
5 >= 2
```

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

1.10 Summary

An introduction to R

- Basic types
- Basic operators

Next: Core concepts

- Variables

- Functions
- Libraries

Chapter 2

Core concepts

2.1 Recap

Prev: An introduction to R

- Basic types
- Basic operators

Now: Core concepts

- Variables
- Functions
- Libraries

2.2 Variables

Variables **store data** and can be defined

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

```
a_variable <- 1
```

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

```
a_variable
```

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

2.3 Algorithms and functions

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

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

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

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

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

2.4 Functions

Functions execute complex operations and can be invoked

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

```
sqrt(2)
```

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

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

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

2.5 Functions and variables

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

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

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

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

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

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

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

2.6 Naming

When creating an identifier for a variable or function

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

2.7 Libraries

Once a number of related, reusable functions are created

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

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

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

2.8 stringr

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

```
library(stringr)

str_length("Leicester")
```

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

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

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

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

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

2.9 Summary

Core concepts

- Variables
- Functions
- Libraries

Next: Tidyverse

- Tidyverse libraries
- *pipe* operator

Chapter 3

Tidyverse

3.1 Recap

Prev: Core concepts

- Variables
- Functions
- Libraries

Now: Tidyverse

- Tidyverse libraries
- *pipe* operator

3.2 Tidyverse

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

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

Core libraries

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

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

3.3 Tidyverse core libraries

The meta-library Tidyverse includes:

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

3.4 Tidyverse core libraries

The meta-library Tidyverse includes:

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

3.5 Tidyverse core libraries

The meta-library Tidyverse contains the following libraries:

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

3.6 The pipe operator

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

The pipe operator `%>%`

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

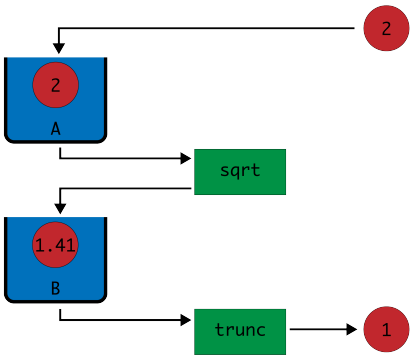
3.7 Pipe example

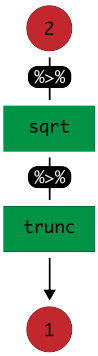
```
tmp_variable_A <- 2
tmp_variable_B <- sqrt(tmp_variable_A)
trunc(tmp_variable_B)

## [1] 1
```

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

## [1] 1
```





3.8 Pipe example

The two codes below are equivalent

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

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

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

```
library(tidyverse)

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

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

3.9 Coding style

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

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

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

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

Study the Tidyverse Style Guid and use it consistently!

3.10 Summary

Tidyverse

- Tidyverse libraries
- *pipe* operator
- Coding style

Next: Practical session

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

Chapter 4

Data types

4.1 Recap

Prev: Introduction

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

Now: Data types

- vectors
- factors
- matrices, arrays
- lists

4.2 Vectors

Vectors are ordered list of values.

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

4.3 Defining vectors

A vector variable can be defined using

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

```
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"))
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-dimensional matrices.

- the first argument is a vector containing the values
- the second argument is a vector specifying the depth of each dimension

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

4.15 Arrays

```
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
```

4.16 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
```

4.17 apply

`apply` applies another function to each level of a set dimension of an array

```
apply(a3dim_array, 3, min) # apply on third dimension
```

```
## [1]  1 13
```

```
apply(a3dim_array, 1, min) # apply on first dimension
```

```
## [1] 1 2 3 4
```

```
apply(a3dim_array, 2, min) # apply on second dimension
```

```
## [1] 1 5 9
```

4.18 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("Stefano", 2015)
employee
```

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

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

```
## [1] "Stefano"
```

4.19 Named Lists

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

```
employee <- list(name = "Stefano", start_year = 2015)
employee
```

```
## $name
## [1] "Stefano"
##
## $start_year
## [1] 2015
```

```
employee$name
```

```
## [1] "Stefano"
```

4.20 lapply

With `lapply` take care that the function makes sense for *any* element in the list

```
various <- list(
  "Some text",
  matrix(c(6, 3, 1, 2), c(2, 2))
)
lapply(various, is.numeric)
```

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

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

4.21 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
- Arrays
- Lists

Now: Control structures

- Conditional statements
- Loops

5.2 If

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

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

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

```
## Negative
```

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

5.3 Else

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

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

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

```
## Negative
```

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

```
## Positive
```

5.4 Code blocks

Suppose you want to execute **several** statements within a function, or if a condition is true

- Such a group of statements are called **code blocks**
- `{` and `}` contain code blocks

```
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 list. 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**

```
add_one (3)
```

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

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 in their corpus

```
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 scrip
- 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

7.1.1 Data Frames

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

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

```
##    Name 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.

7.2 Selection

Selection is similar to vectors and lists.

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

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

```
employees[, 1] # column selection, as for matrices
```

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

7.3 Selection

Selection is similar to vectors and lists.

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

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

```
employees$Name[1]
```

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

7.4 Value assignment

Values can be assigned to cells through filtering and <-

```
employees$Age[3] <- 33
employees
```

```
##    Name Age       Role
## 1 Maria  47  Professor
## 2  Pete  34 Researcher
## 3 Sarah  33 Researcher
```

7.5 Column processing

Operations can be performed on columns, and new columns created.

```
current_year <- as.integer(format(Sys.Date(), "%Y"))
employees$Year_of_birth <- current_year - employees$Age
employees
```

```
##    Name Age       Role Year_of_birth
## 1 Maria  47  Professor          1973
## 2  Pete  34 Researcher          1986
## 3 Sarah  33 Researcher          1987
```

7.6 tibble

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

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

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

7.7 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 data frames

```
library(tidyverse)
```

8.3 Example dataset

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

```
install.packages("nycflights13")
```

```
library(nycflights13)

flights_from_nyc <- nycflights13::flights

colnames(flights_from_nyc)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

8.4 dplyr::select

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

```
delays <- select(flights_from_nyc,
  origin, dest, dep_delay, arr_delay,
  year:day
  )

# Drop column arr_delay using - in front of the column name
dep_delays <- select(delays, -arr_delay)

delays[1:3, ]
```

```
## # A tibble: 3 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
```

8.5 dplyr::select

... using the pipe operator

```
dep_delays <- flights_from_nyc %>%
  select(origin, dest, dep_delay, arr_delay, year:day) %>%
  select(-arr_delay)
```

```
delays[1:3, ]
```

```
## # A tibble: 3 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
```

8.6 Logical filtering

Conditional statements can be used to filter a vector, i.e. to retain only certain values

```
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
```

8.7 Conditional filtering

As a condition expression results in a logic vector, that condition can be used for filtering

```
a_numeric_vector > 0
```

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

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

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

8.8 Filtering data frames

The same can be applied to data frames

```
nov_dep_delays <- dep_delays[dep_delays$month == 11, ]

nov_dep_delays[1:3, ]
```

```
## # A tibble: 3 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
```

8.9 dplyr::filter

```
nov_dep_delays <- dep_delays %>%
  filter(month == 11) # Flights in November

nov_dep_delays[1:3, ]
```

```
## # A tibble: 3 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
```

8.10 Summary

Data selection and filtering

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

Next:

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 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 data frames

9.3 Libraries

```
library(tidyverse)
library(nycflights13)

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

9.4 dplyr::arrange

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

nov_dep_delays[1:3, ]
```

```
## # A tibble: 3 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
## 3 JFK    ABQ          17  2013    11    21
```

9.5 dplyr::summarise

`summarise`: calculate aggregated values (e.g., mean, max, etc)

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

aggr_dep_delays_nov
```

```
## # A tibble: 1 x 2
```

```
##   avg_dep_delay tot_dep_delay
##           <dbl>         <dbl>
## 1          5.44        146945
```

9.6 dplyr::group_by

```
dest_dep_delays_nov <- nov_dep_delays %>%
  # Need to filter out rows where delay is NA
  filter(!is.na(dep_delay)) %>%
  # Frist group by same destination
  group_by(dest) %>%
  # Then caluculate aggregated value
  summarize(
    tot_dep_delay = sum(dep_delay)
  )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
dest_dep_delays_nov[1:3, ]
```

```
## # A tibble: 3 x 2
##   dest  tot_dep_delay
##   <chr>         <dbl>
## 1 ABQ             -66
## 2 ALB             636
## 3 ATL            8184
```

9.7 dplyr::mutate

```
dest_dep_delays_nov <- dest_dep_delays_nov %>%
  mutate(
    tot_dep_delay_days = ((tot_dep_delay / 60) /24)
  )

dest_dep_delays_nov[1:3, ]
```

```
## # A tibble: 3 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
```

9.8 Full pipe example

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
dest_dep_delays_nov[1:3, ]
```

```
## # A tibble: 3 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
```

9.9 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
- Data selection and filtering
- Data manipulation

Now: Join operations

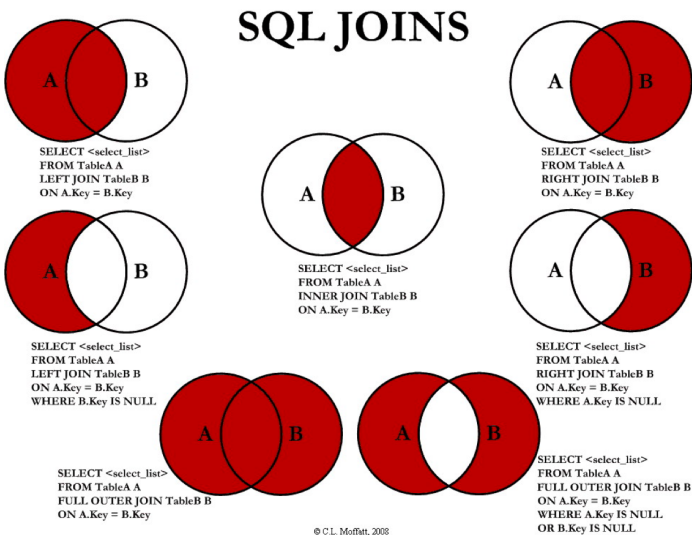
- Joining data
- dplyr join functions

10.2 Joining data

Data frames can be joined (or 'merged')

- information from two data frames can be combined
- specifying a **column 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 data frame can be merged with multiple rows from the other data frame
  - rows with no matching values in the other data frame can be retained
- `merge` base function or join functions in `dplyr`

10.3 Join types



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

10.4 Example

```
employees <- data.frame(
  Name = c("Maria", "Pete", "Sarah", "Jo"),
  Age = c(47, 34, 32, 25),
  Role = c("Professor", "Researcher", "Researcher", "Postgrad")
)

city_of_birth <-data.frame(
  Name = c("Maria", "Pete", "Sarah", "Mel"),
  City = c("Barcelona", "London", "Boston", "Los Angeles")
)
```

10.5 Example

Name	Age	Role
Maria	47	Professor
Pete	34	Researcher
Sarah	32	Researcher
Jo	25	Postgrad

Name	City
Maria	Barcelona
Pete	London
Sarah	Boston
Mel	Los Angeles

10.6 dplyr::full_join

`dplyr` provides a series of join functions

- `full_join` combines all the available data

```
employees %>% full_join(
  city_of_birth,
  by = c("Name" = "Name") # join columns
  ) %>%
  kable()
```

Name	Age	Role	City
Maria	47	Professor	Barcelona
Pete	34	Researcher	London
Sarah	32	Researcher	Boston
Jo	25	Postgrad	NA
Mel	NA	NA	Los Angeles

10.7 dplyr::left_join

- `left_join` keeps all the data from the **left** table
  - using `%>%`, that's the data *"coming down the pipe"*
- rows from the right table without a match are dropped

```
employees %>% left_join(
  city_of_birth,
  by = c("Name" = "Name") # join columns
  ) %>%
  kable()
```

Name	Age	Role	City
Maria	47	Professor	Barcelona
Pete	34	Researcher	London
Sarah	32	Researcher	Boston
Jo	25	Postgrad	NA

10.8 dplyr::right_join

- `right_join` keeps all the data from the **right** table
  - using `%>%`, that's the data provided as an argument

- rows from the left table without a match are dropped

```
employees %>% right_join(
  city_of_birth,
  by = c("Name" = "Name") # join columns
  ) %>%
  kable()
```

Name	Age	Role	City
Maria	47	Professor	Barcelona
Pete	34	Researcher	London
Sarah	32	Researcher	Boston
Mel	NA	NA	Los Angeles

10.9 dplyr::inner_join

- `inner_join` keeps only rows that have a match in both tables
- rows without a match are dropped

```
employees %>% inner_join(
  city_of_birth,
  by = c("Name" = "Name") # join columns
  ) %>%
  kable()
```

Name	Age	Role	City
Maria	47	Professor	Barcelona
Pete	34	Researcher	London
Sarah	32	Researcher	Boston

10.10 Summary

Join operations

- Joining data
- dplyr join functions

Next: Table pivot

Chapter 11

Data pivot

11.1 Recap

Prev: Selection and manipulation

- Data Frames
- Data selection and filtering
- Data manipulation

Now: Data pivot

- Wide and long data
- tidyr::pivot_longer
- tidyr::pivot_wider

11.2 Wide data

This is the most common approach

- each real-world entity is represented by *one single row*
- its attributes are represented through different columns

City	Population	Area	Density
Leicester	329,839	73.3	4,500
Nottingham	321,500	74.6	4,412

11.3 Long data

This is probably a less common approach, but still necessary in many cases

- 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

City	Attribute	Value
Leicester	Population	329,839
Leicester	Area	73.3
Leicester	Density	4,500
Nottingham	Population	321,500
Nottingham	Area	74.6
Nottingham	Density	4,412

11.4 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(knitr)
```

11.5 tidyr

The `tidyr` (pronounced *tidy-er*) library is part of `tidyverse` and it provides functions to re-shape your data

```
city_info_wide <- data.frame(
  City = c("Leicester", "Nottingham"),
  Population = c(329839, 321500),
  Area = c(73.3, 74.6),
  Density = c(4500, 4412)
)

kable(city_info_wide)
```

City	Population	Area	Density
Leicester	329839	73.3	4500
Nottingham	321500	74.6	4412

11.6 tidyr::gather

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

```
city_info_long <- city_info_wide %>%
  gather(
    -City, # exclude city names from gathering
    key = "Attribute", # name for the new key column
    value = "Value" # name for the new value column
  )
```

City	Attribute	Value
Leicester	Population	329839.0
Nottingham	Population	321500.0
Leicester	Area	73.3
Nottingham	Area	74.6
Leicester	Density	4500.0
Nottingham	Density	4412.0

11.7 tidyr::spread

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

```
city_info_back_to_wide <- city_info_long %>%
  spread(
    key = "Attribute", # specify key column
    value = "Value" # specify value column
  )
```

City	Area	Density	Population
Leicester	73.3	4500	329839
Nottingham	74.6	4412	321500

11.8 Summary

Table pivot

- Wide and long data
- tidyr::pivot_longer
- tidyr::pivot_wider

Next: Read and write data

Chapter 12

Read and write data

12.1 Summary

Table pivot

- Wide and long data
- tidyr::pivot_longer
- tidyr::pivot_wider

Next: Read and write data

- file formats
- read
- write

12.2 Comma Separated Values

The file `2011_OAC_Raw_uVariables_Leicester.csv` - contains data used for the 2011 Output Area Classificagtion - 167 variables, as well as the resulting groups - for the city of Leicester

Extract showing only some columns

```
OA11CD,LSOA11CD, ... supgrpcode,supgrpname,Total_Population, ...
E00069517,E01013785, ... 6,Suburbanites,313, ...
E00169516,E01013713, ... 4,Multicultural Metropolitans,341, ...
E00169048,E01032862, ... 4,Multicultural Metropolitans,345, ...
```

The full variable names can be found in the file - `2011_OAC_Raw_uVariables_Lookup.csv`.

12.3 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(knitr)
```

12.4 Read

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

```
leicester_2011OAC <- read_csv("2011_OAC_Raw_uVariables_Leicester.csv")

leicester_2011OAC %>%
  select(OA11CD,LSOA11CD, supgrpcode,supgrpname,Total_Population) %>%
  top_n(3) %>%
  kable()
```

OA11CD	LSOA11CD	supgrpcode	supgrpname	Total_Population
E00169553	E01013648	2	Cosmopolitans	714
E00069303	E01013739	4	Multicultural Metropolitans	623
E00168096	E01013689	2	Cosmopolitans	708

12.5 Write

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 *Leicester_Suburbanites.csv* in your home folder

```
read_csv("2011_OAC_Raw_uVariables_Leicester.csv") %>%
  select(OA11CD, supgrpcode, supgrpname) %>%
```

```
  filter(supgrpcode == 6) %>%
  write_csv("~/Leicester_Suburbanites.csv")
```

12.6 Summary

Read and write data

- file formats
- read
- write

Next: Practical session

- Join operations
- Table pivot
- Read and write data

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 and excellent way to document your project.

13.8 Document well

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

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

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 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

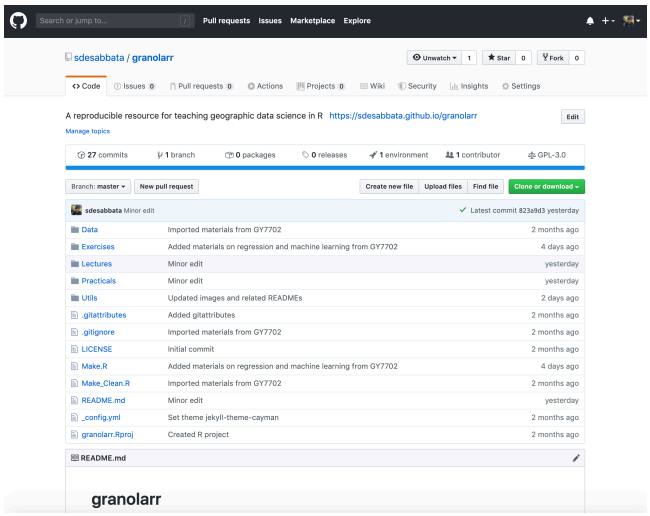
- data: .txt, .csv, .tsv
- analysis: R scrpts, python scripts
- write-up: LaTeX, Markdown, HTML

13.11 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

13.12 This repository



github.com/sdesabbata/granolarr

13.13 Summary

Reproduciblity

- Reproduciblity and software engineering
- Reproduciblity in GIScience
- Guidelines

Next: RMarkdown

- Markdown
- RMarkdown

Chapter 14

RMarkdown

14.1 Recap

Prev: Reproduciblity

- Reproduciblity and software engineering
- Reproduciblity in GIScience
- Guidelines

Now: RMarkdown

- Markdown
- RMarkdown

14.2 Markdown

Markdown is a simple markup language

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

Markdown can be used in conjunction with numerous tools

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

These slides are written in Markdown

14.3 Markdown example code

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

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

- and list can be created
    - very simply

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

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

14.4 Markdown example output

14.4.1 This is a third level heading

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

- and list can be created
  - very simply

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

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

14.5 RMarkdown example code

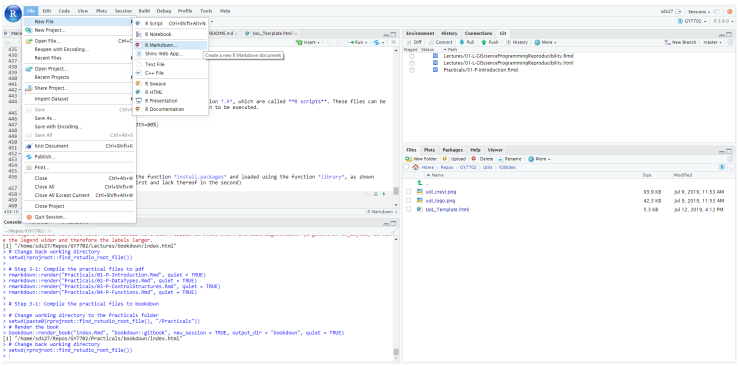
```
Let's write an example of **R** code including

- a variable `a_variable`
- an assignment operation (i.e., `<-`)
- a mathematical operation (i.e., `+`)

```{r, echo=TRUE}
a_variable <- 0
a_variable <- a_variable + 1
a_variable <- a_variable + 1
a_variable <- a_variable + 1
a_variable
```
```

14.6 Writing RMarkdown docs

RMarkdown documents contain both Markdown and R code. These files can be created in RStudio, and compiled to create an html page (like this document), a pdf, or a Microsoft Word document.



14.7 Summary

RMarkdown

- Markdown
- RMarkdown

Next: Git

- Git operations
- Git and RStudio

Chapter 15

Git

15.1 Recap

RMarkdown

- Markdown
- RMarkdown

Next: Git

- Git operations
- Git and RStudio

15.2 What's git?

Git is a free and opensource version control system

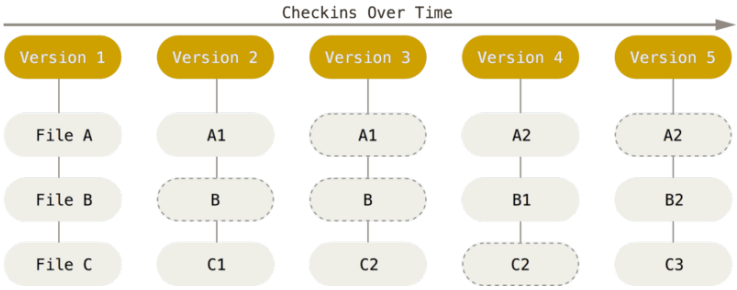
- commonly used through a server
  - where a master copy of a project is kept
  - can also be used locally
- allows storing versions of a project
  - syncronisation
  - consistency
  - history
  - multiple branches

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

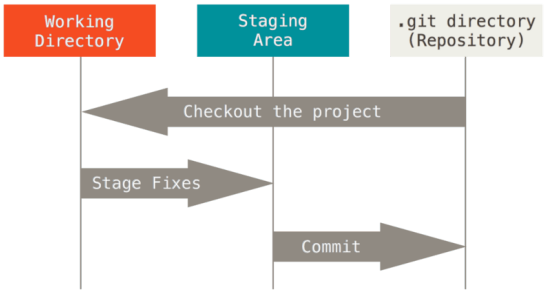


by Scott Chacon and Ben Straub, licensed under CC BY-NC-SA 3.0

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

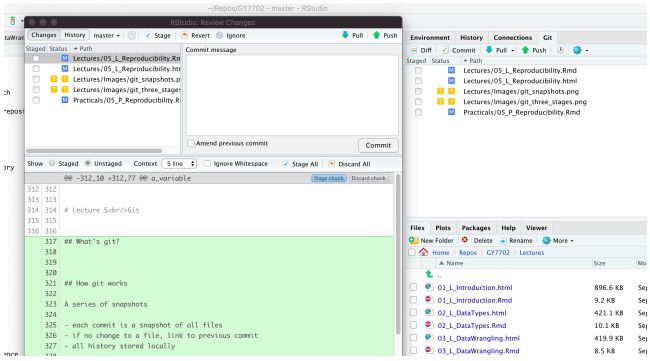
- `git clone`
  - copy a repository from a server
- `git fetch`
  - get the latest version from a branch
- `git pull`
  - incorporate changes from a remote repository
- `git add`
  - stage new files
- `git commit`

  - 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 Summary

Git

- Git operations
- Git and RStudio

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 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)

16.3 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.).

16.4 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

```
library(tidyverse)
library(nycflights13)
library(knitr)
```

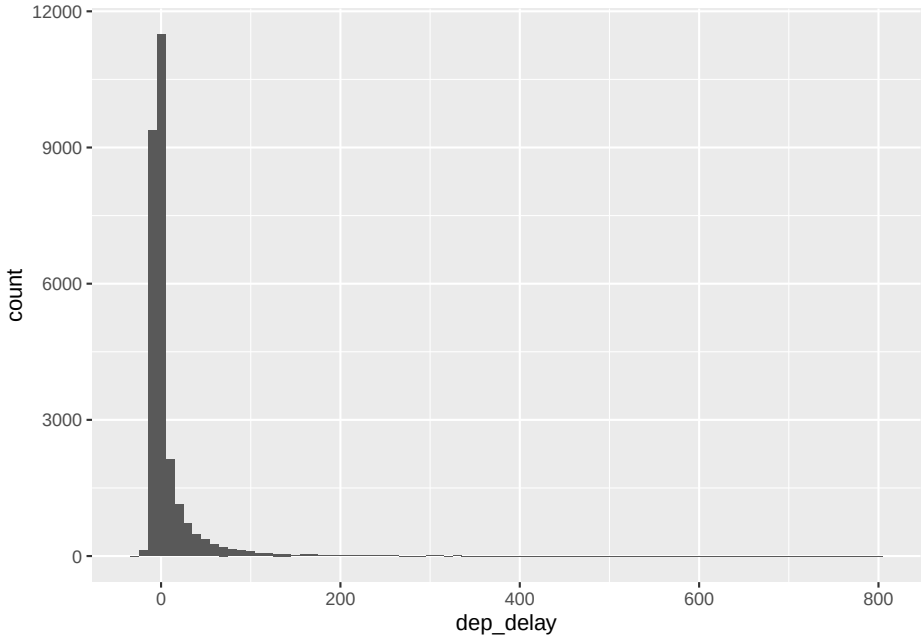
16.5 Histograms

- `x` variable to plot
- `geom_histogram`

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

```
    binwidth = 10
  )
```

16.6 Histograms



:::

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

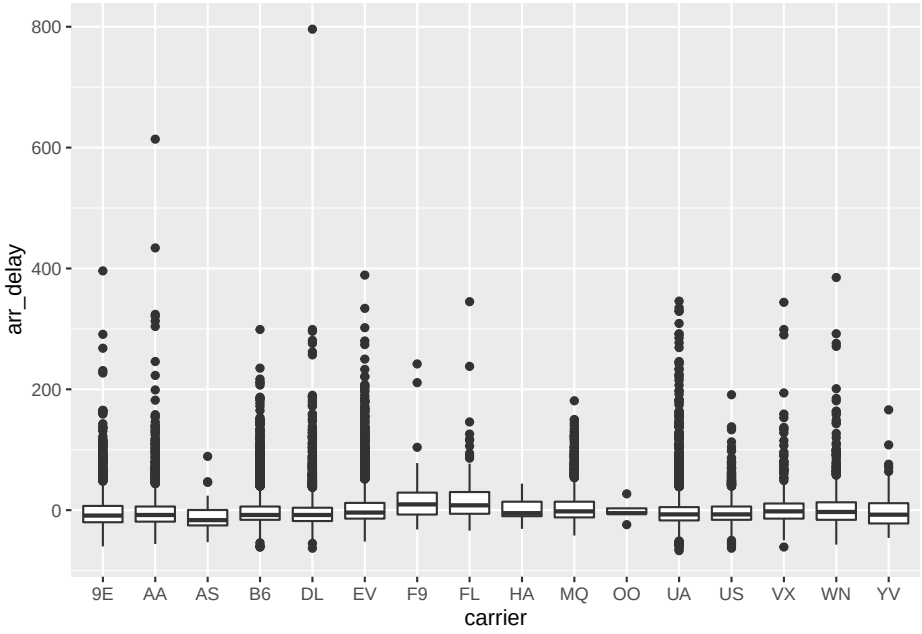
:::::: –>

16.7 Boxplots

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

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

16.8 Boxplots



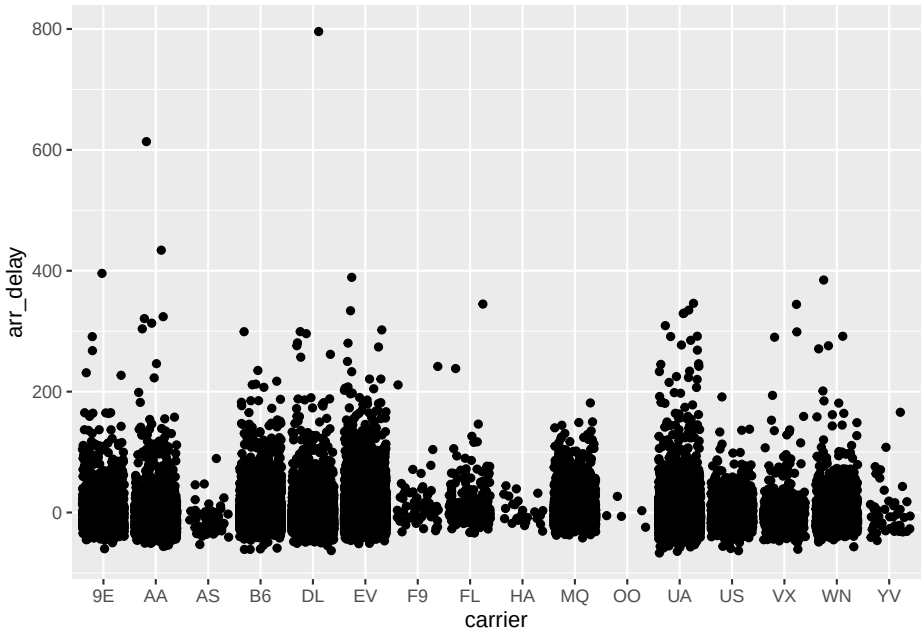
16.9 Jittered points

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

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

```
      y = arr_delay
    )
  ) +
  geom_jitter()
```

16.10 Jittered points

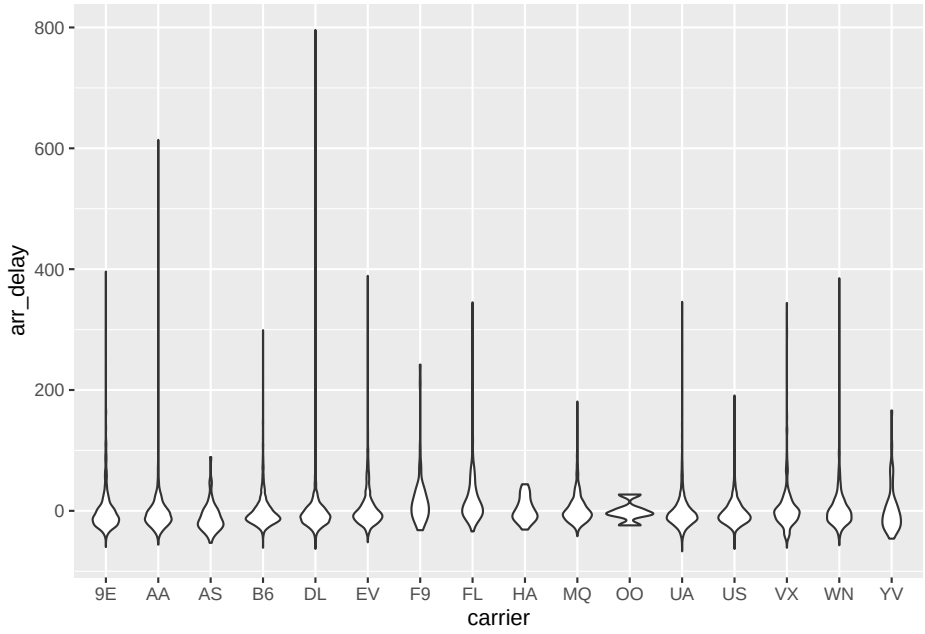


16.11 Violin plot

- x categorical variable
- y variable to plot
- geom_violin

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

16.12 Violin plot

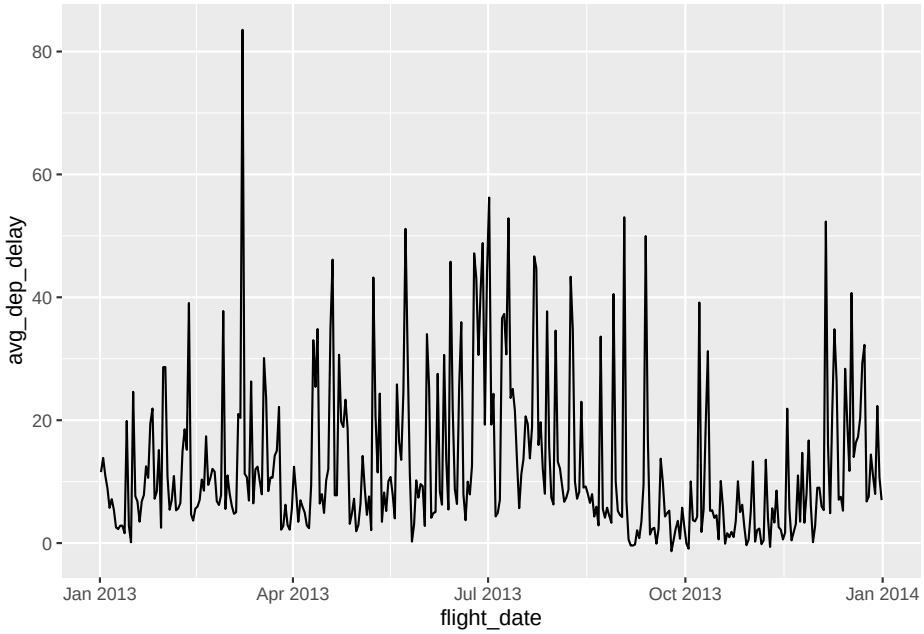


16.13 Lines

- `x` e.g., a temporal variable
- `y` variable to plot
- `geom_line`

```
nycflights13::flights %>%
  filter(!is.na(dep_delay)) %>%
  mutate(flight_date = ISOdate(year, month, day)) %>%
  group_by(flight_date) %>%
  summarize(avg_dep_delay = mean(dep_delay)) %>%
  ggplot(aes(
    x = flight_date,
    y = avg_dep_delay
  )) +
  geom_line()
```

16.14 Lines

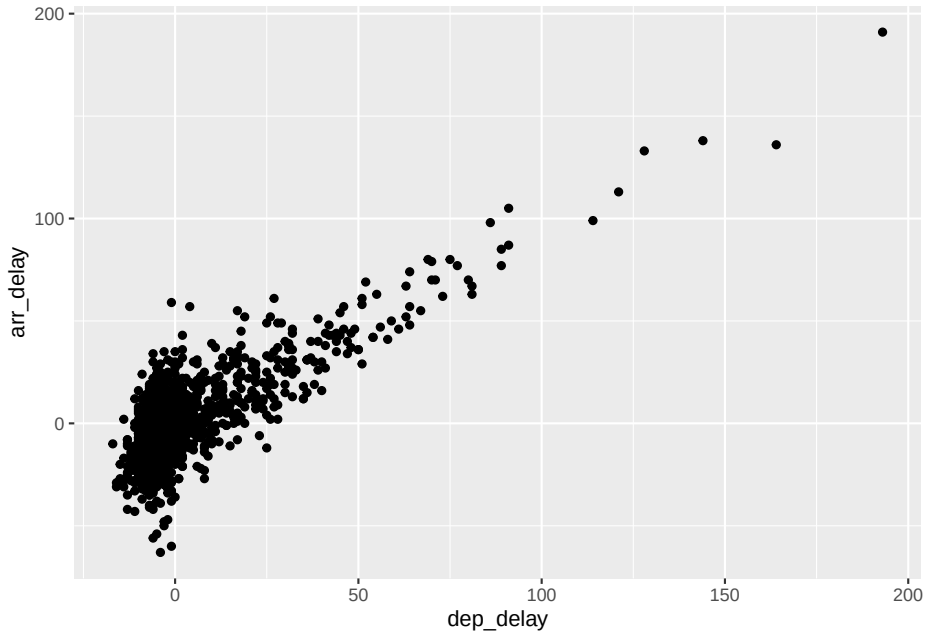


16.15 Scatterplots

- x and y variable to plot
- geom_point

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

16.16 Scatterplots

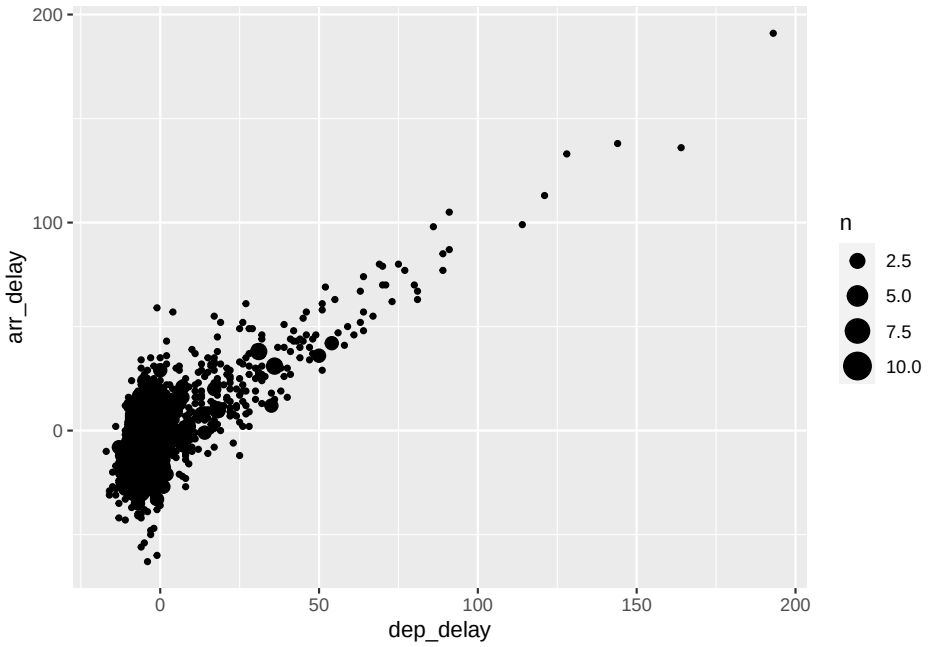


16.17 Overlapping points

- `x` and `y` variable to plot
- `geom_count` counts overlapping points and maps the count to size

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

16.18 Overlapping points

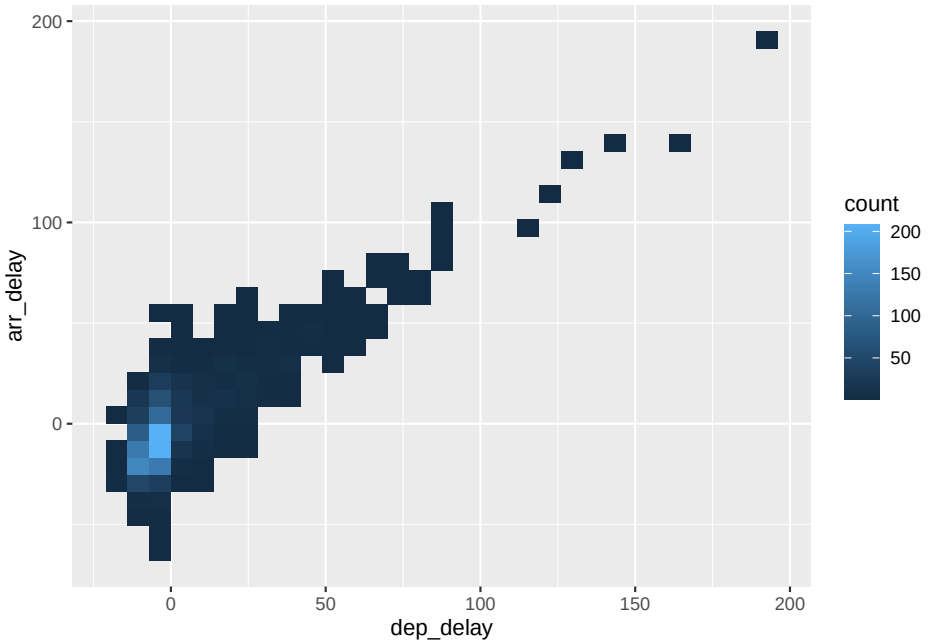


16.19 Bin counts

- x and y variable to plot
- geom_bin2d

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

16.20 Bin counts



16.21 Summary

Data visualisation

- Grammar of graphics
- ggplot2

Next: Descriptive statistics

- stat.desc
- dplyr::across

Chapter 17

Descriptive statistics

17.1 Summary

Data visualisation

- Grammar of graphics
- ggplot2

Next: Descriptive statistics

- stat.desc
- dplyr::across

17.2 Libraries and data

```
library(tidyverse)
library(magrittr)
library(knitr)

library(pastecs)

library(nycflights13)

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

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

```
nycflights13::flights %>%
  filter(month == 11, carrier == "US") %>%
  select(dep_delay, arr_delay, distance) %>%
  stat.desc() %>%
  kable()
```

17.4 stat.desc output

	dep_delay	arr_delay	distance
nbr.val	1668.0000000	1667.000000	1.699000e+03
nbr.null	58.0000000	35.000000	0.000000e+00
nbr.na	31.0000000	32.000000	0.000000e+00
min	-17.0000000	-63.000000	9.600000e+01
max	193.0000000	191.000000	2.153000e+03
range	210.0000000	254.000000	2.057000e+03
sum	961.0000000	-4450.000000	9.715580e+05
median	-4.0000000	-7.000000	5.290000e+02
mean	0.5761391	-2.669466	5.718411e+02
SE.mean	0.4084206	0.518816	1.464965e+01
CI.mean.0.95	0.8010713	1.017600	2.873327e+01
var	278.2347513	448.706408	3.646264e+05
std.dev	16.6803702	21.182691	6.038430e+02
coef.var	28.9519850	-7.935179	1.055963e+00

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

17.6 stat.desc: desc

- `min` (also `min()`): **minimum** value in the dataset
- `max` (also `max()`): **minimum** value in the dataset
- `range`: difference between `min` and `max` (different from `range()`)
- `sum` (also `sum()`): sum of the values in the dataset
- `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()`functio is available: **mode**, the value that appears most often in the values

17.7 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

17.8 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

17.9 dplyr::across

TODO

17.10 Summary

Descriptive statistics

- 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 Libraries and data

```
library(tidyverse)
library(magrittr)
library(knitr)

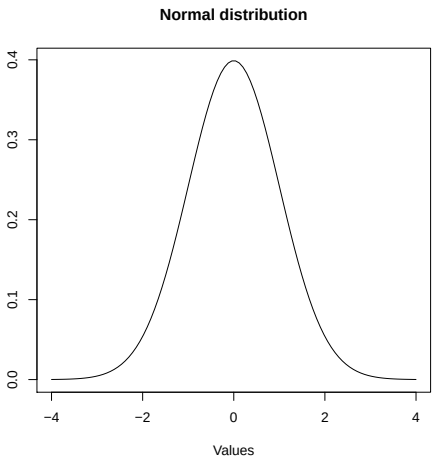
library(pastecs)

library(nycflights13)

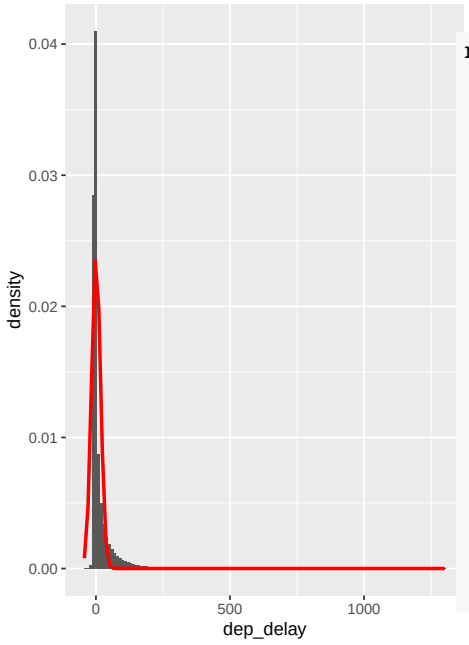
flights_nov_20 <- nycflights13::flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay), month == 11, day ==20)
```

18.3 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



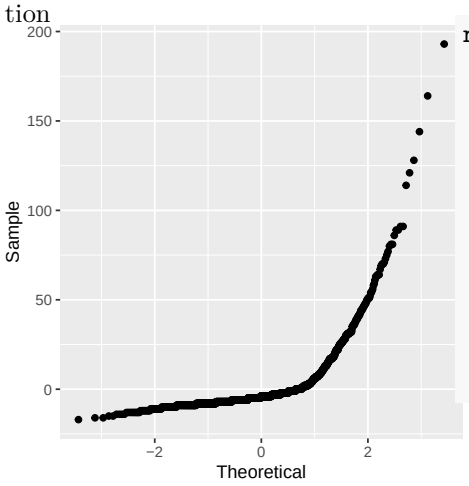
18.4 Density histogram




```
nycflights13::flights %>%
  ggplot(
    aes(
      x = dep_delay
    )
  ) +
  geom_histogram(
    aes(
      y =..density..
    ),
    binwidth = 10
  ) +
  stat_function(
    fun = dnorm,
    args = list(
      mean = dep_delay_mean,
      sd = dep_delay_sd),
    colour = "black", size = 1)
```

18.5 Q-Q plot

Cumulative values against the cumulative probability of a particular distribution




```
nycflights13::flights %>%
  filter(
    month == 11,
    carrier == "US"
  ) %>%
  qplot(
    sample = dep_delay,
    data = .,
    stat = "qq",
    xlab = "Theoretical",
    ylab = "Sample"
  )
```

18.6 stat.desc: norm

```
nycflights13::flights %>%
  filter(month == 11, carrier == "US") %>%
  select(dep_delay, arr_delay, distance) %>%
  stat.desc(basic = FALSE, desc = FALSE, norm = TRUE) %>%
  kable()
```

	dep_delay	arr_delay	distance
skewness	4.4187763	2.0716291	2.0030249
skew.2SE	36.8709612	17.2808242	16.8678747
kurtosis	28.8513206	9.5741004	2.6000743
kurt.2SE	120.4418092	39.9557893	10.9542887
normtest.W	0.5545326	0.8657894	0.6012442
normtest.p	0.0000000	0.0000000	0.0000000

18.7 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
- `normtest.W` (test statistics) and `normtest.p` (significance)
- also, `shapiro.test` function is available

```
nycflights13::flights %>%
  filter(month == 11, carrier == "US") %>%
  pull(dep_delay) %>%
  shapiro.test()
```

```
##
##  Shapiro-Wilk normality test
##
## data:  .
## W = 0.55453, p-value < 2.2e-16
```

18.8 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.

18.9 Skewness and kurtosis

In a normal distribution, the values of *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. If greater than 1, the respective statistics is significant ($p < .05$).

18.10 Homogeneity of variance

Levene's test for equality of variance in different levels

- If significant, the variance is different in different levels

```r
dep_delay_carrier <- nycflights13::flights %>%
  filter(month == 11) %>%
  select(dep_delay, carrier)

library(car)
leveneTest(dep_delay_carrier$dep_delay, dep_delay_carrier$carrier)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##          Df F value    Pr(>F)
## group    15  20.203 < 2.2e-16 ***
##       27019
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

18.11 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

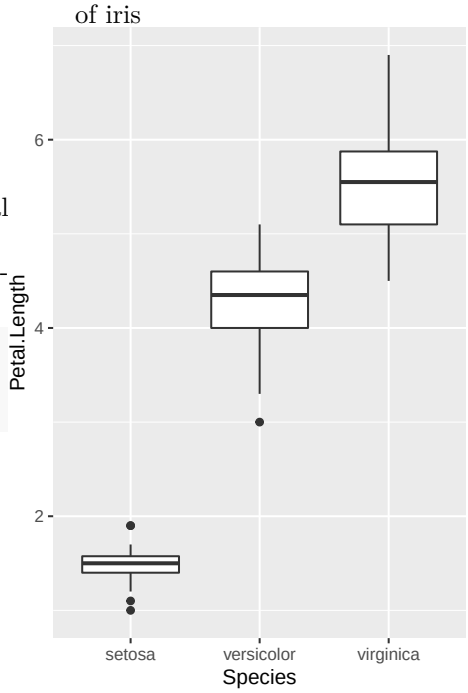
Today's libraries

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

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

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

- 50 flowers from each of 3 species of iris



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

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

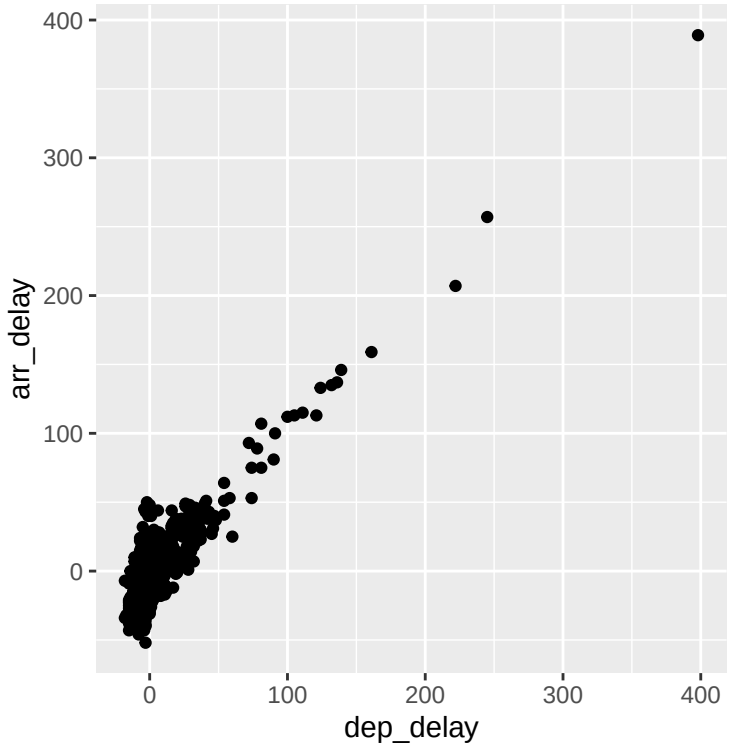
- related
  - positively: entities with high values in one tend to have high values in the other
  - negatively: entities with high values in one tend to have low values in the other
- not related at all

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

- $0.396 \hat{} 2 = 0.157$
- **15.7% of variance**

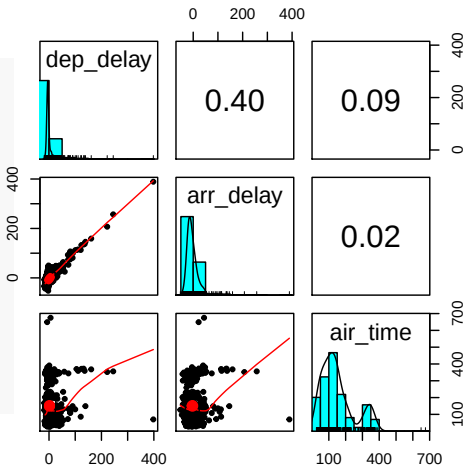
```
flights_nov_20 %$%
  cor.test(
    dep_delay, arr_delay,
    method = "kendall")
```

```
##
##  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

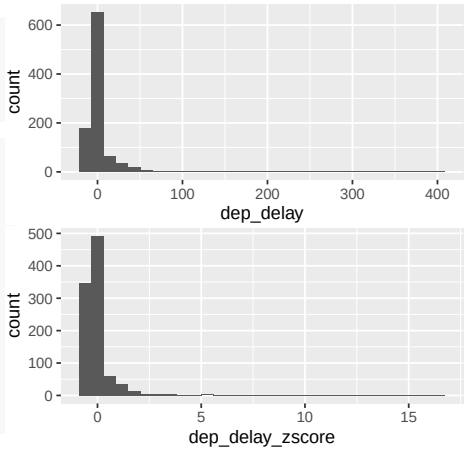
```
library(tidyverse)
library(magrittr)
library(nycflights13)

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()
```

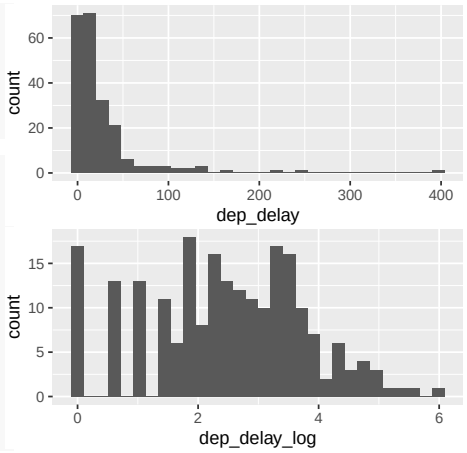


21.4 Log transformation

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()
```

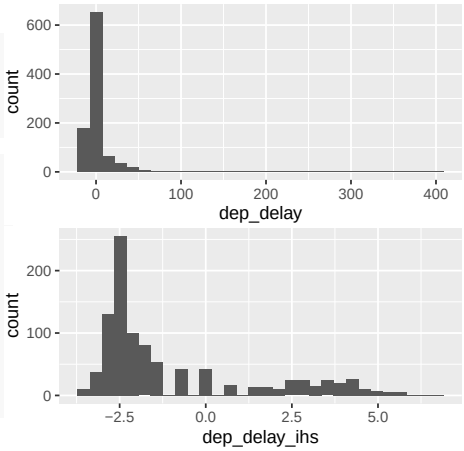


21.5 Inverse hyperbolic sine

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

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

```
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)

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

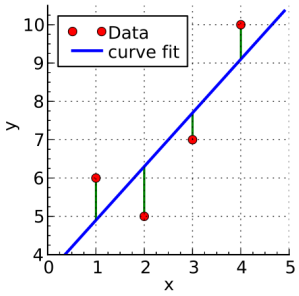
- more predictor variables (**multiple / multivariate** regression)

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

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

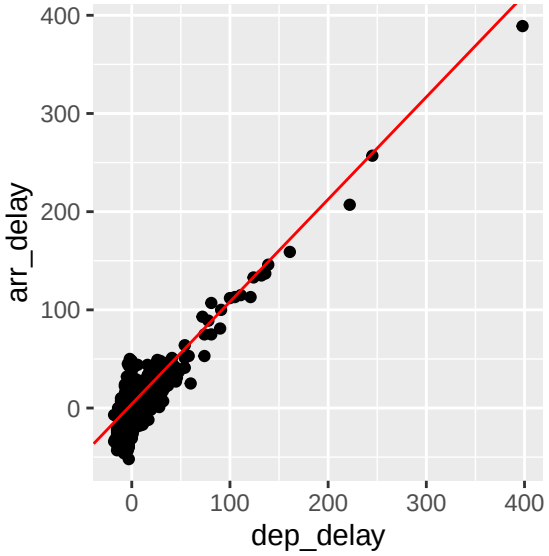
The output indicates

- **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

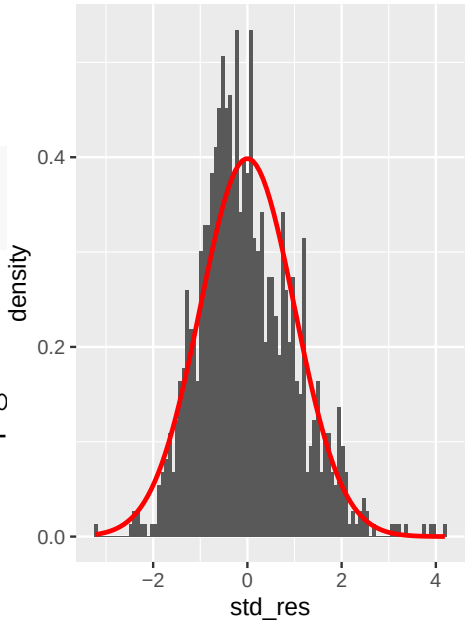
Shapiro-Wilk test for normality of standard residuals,

- robust models: should be not significant

```
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

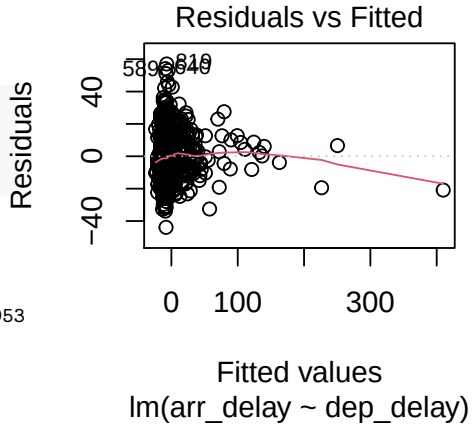
Breusch-Pagan test for homoscedasticity of standard residuals

- robust models: should be not significant

```
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

```
# 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

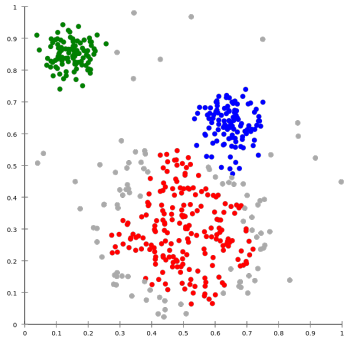
25.5 Supervised

- 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, CC-BY-SA-4.0

25.6 Unsupervised

- Dataset
  - input attribute(s) to explore
- Type of model for the learning process
  - most approaches are iterative
  - e.g., hierarchical clustering
- Evaluation function
  - evaluates the quality of the pattern under consideration during one iteration

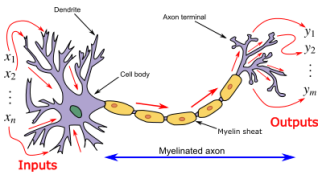


by Chire via Wikimedia Commons, CC-BY-SA-3.0

25.7 ... more

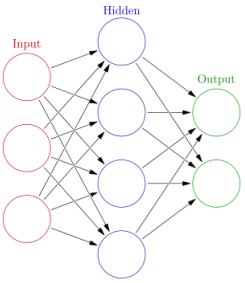
- **Semi-supervised learning**
  - between unsupervised and supervised learning
  - combines a small amount of labelled data with a larger un-labelled dataset
  - continuity, cluster, and manifold (lower dimensionality) assumption
- **Reinforcement learning**
  - training agents take actions to maximize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

25.8 Neural networks



Supervised learning approach simulating simplistic neurons

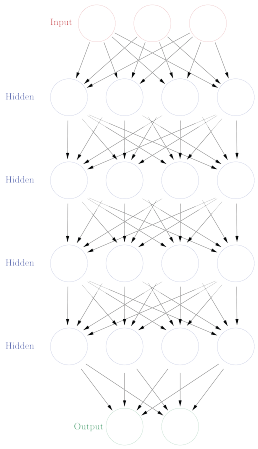
- Classic model with 3 sets
  - input neurons
  - output neurons
  - hidden layer
    - combines input values using **weights**
    - **activation function**
- The **traning algorithm** is used to define the best weights



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

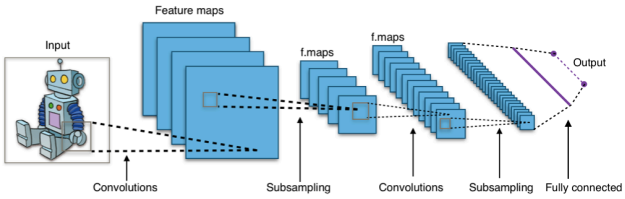
25.10 Convolutional neural networks

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

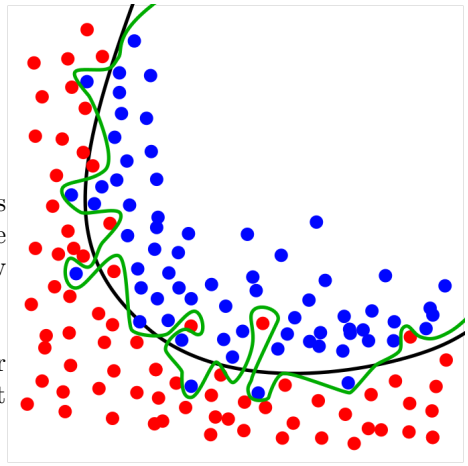
Example: Liu, P. and De Sabbata, S. (2019). Learning Digital Geographies through a Graph-Based Semi-supervised Approach. In proceedings of the 15th International Conference on GeoComputation, Queenstown, New Zealand.



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

Centroid-based clustering

26.1 Recap

Prev: Machine Learning

- What's Machine Learning?
- Types
- Limitations

Now: Centroid-based clustering

- K-means
- Fuzzy c-means
- Geodemographic classification
- Hierarchical
- Mixed
- Density-based

26.2 Clustering task

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

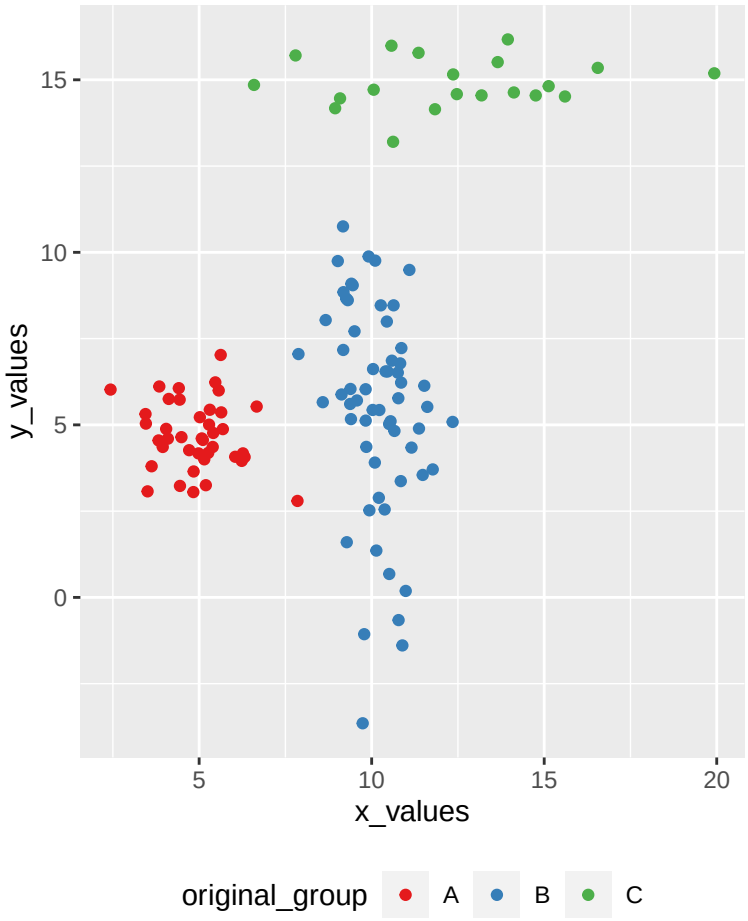
Methods:

- Centroid-based
  - k-means
  - fuzzy c-means
- Hierarchical

- Mixed
  - bootstrap aggregating
- Density-based
  - DBSCAN

26.3 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)) )
```



26.4 k-means

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

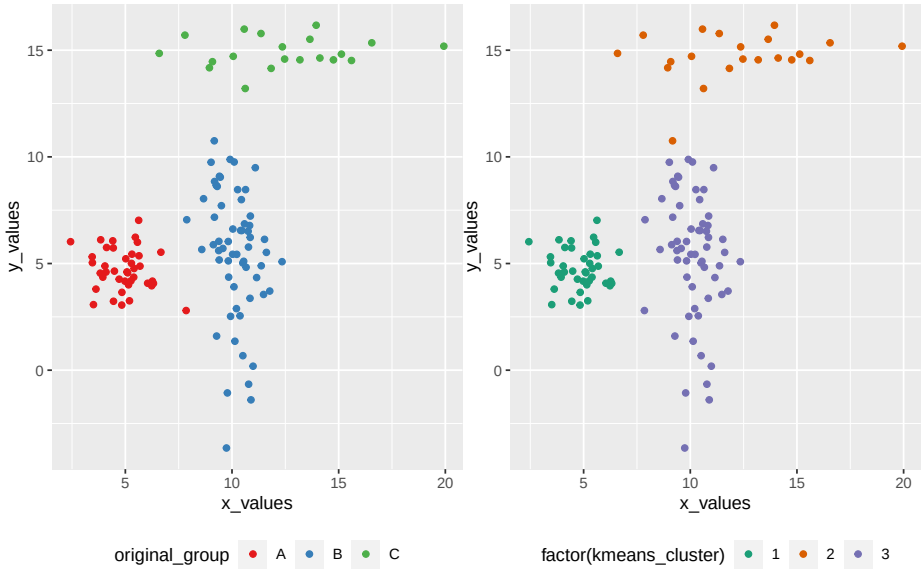
Algorithm: `k` observations a randomly selected as initial centroids, then repeat

- **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)
```

26.5 K-means result



26.6 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)
```

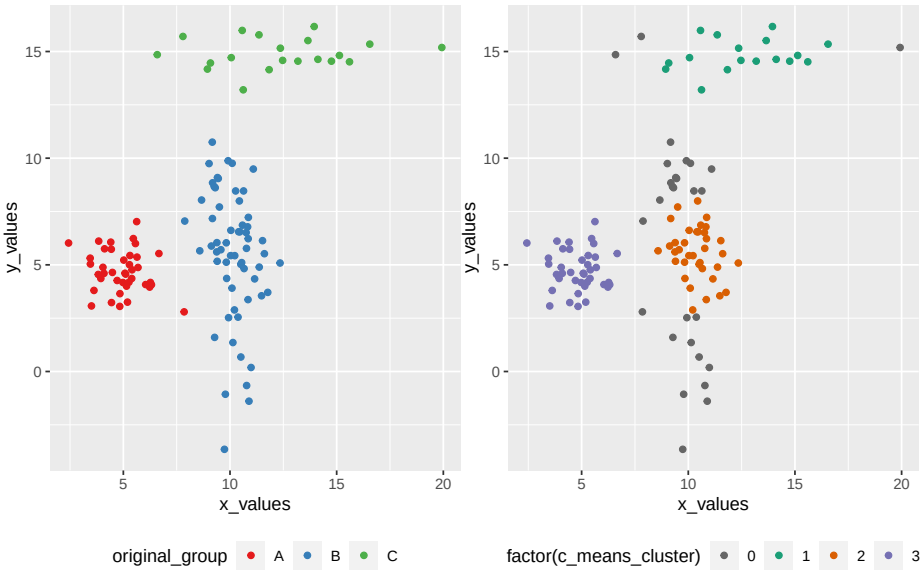
26.7 Fuzzy c-means

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
    )
  )
```

26.8 Fuzzy c-means result



26.9 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.

26.10 Summary

Centroid-based clustering

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

Next: Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

Chapter 27

Hierarchical and density-based clustering

27.1 Recap

Prev: Centroid-based clustering

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

Now: Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

27.2 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)
```

27.3 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)) )
```

27.4 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

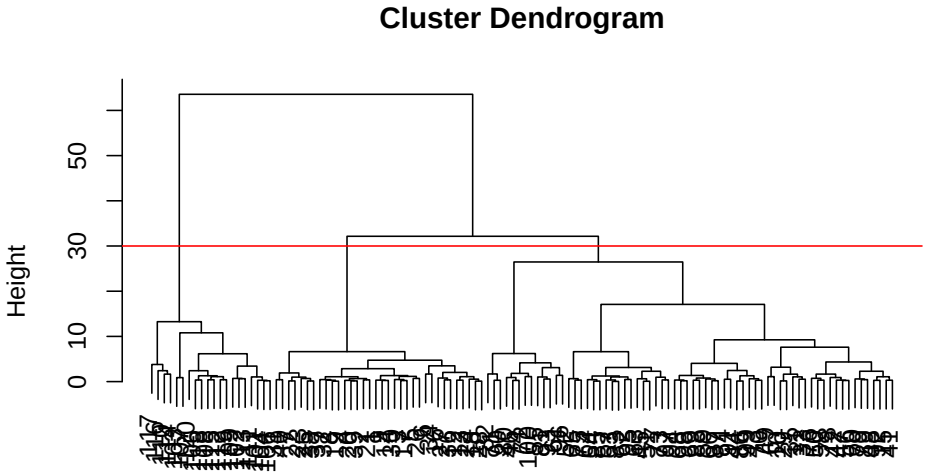
until only one single cluster is achieved

```
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))
```

27.5 Clustering tree

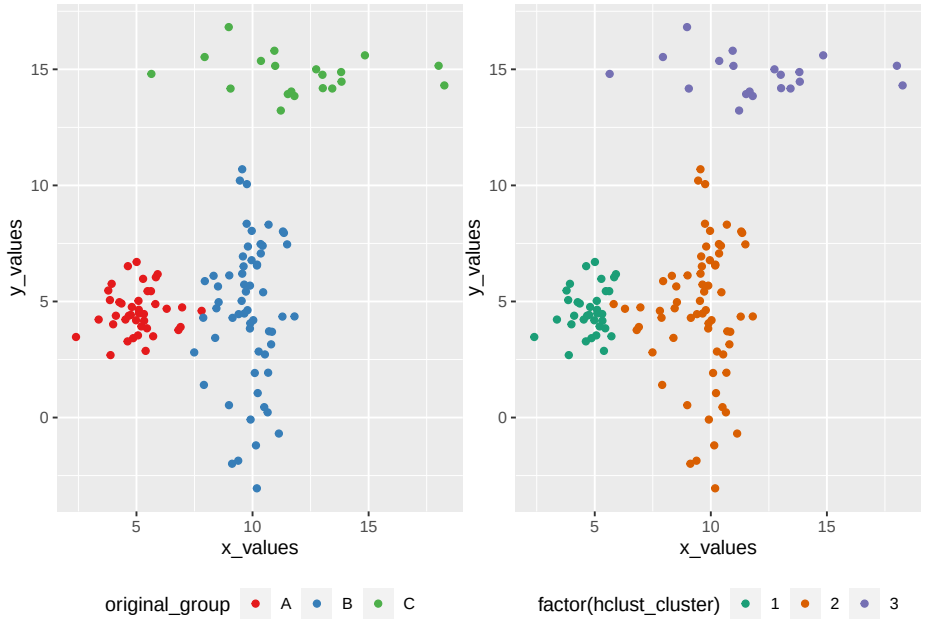
This approach generates a clustering tree (dendrogram), which can then be *"cut"* at the desired height

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




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

27.6 Hierarchical clustering result



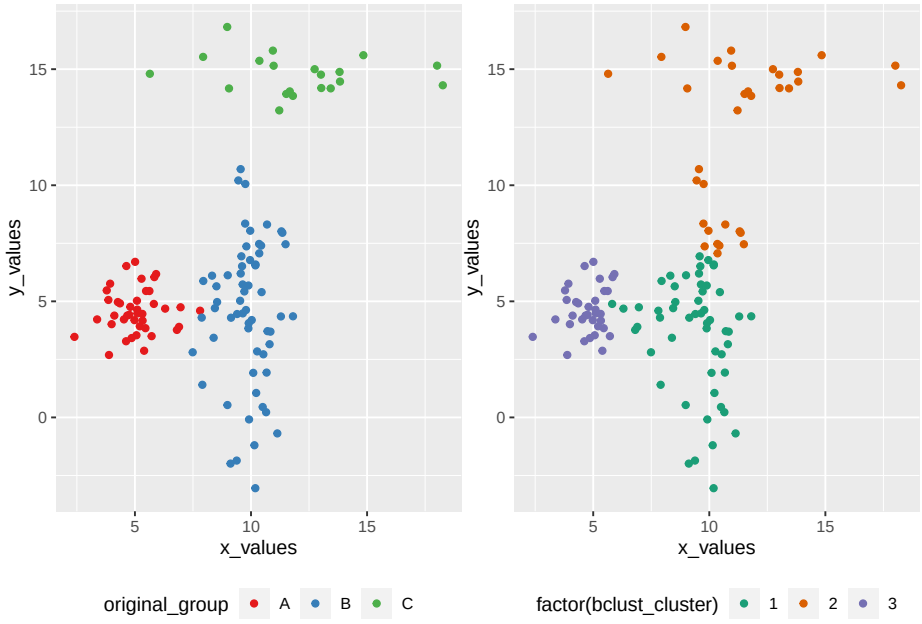
27.7 Bagged clustering

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))
```

27.8 Bagged clustering result



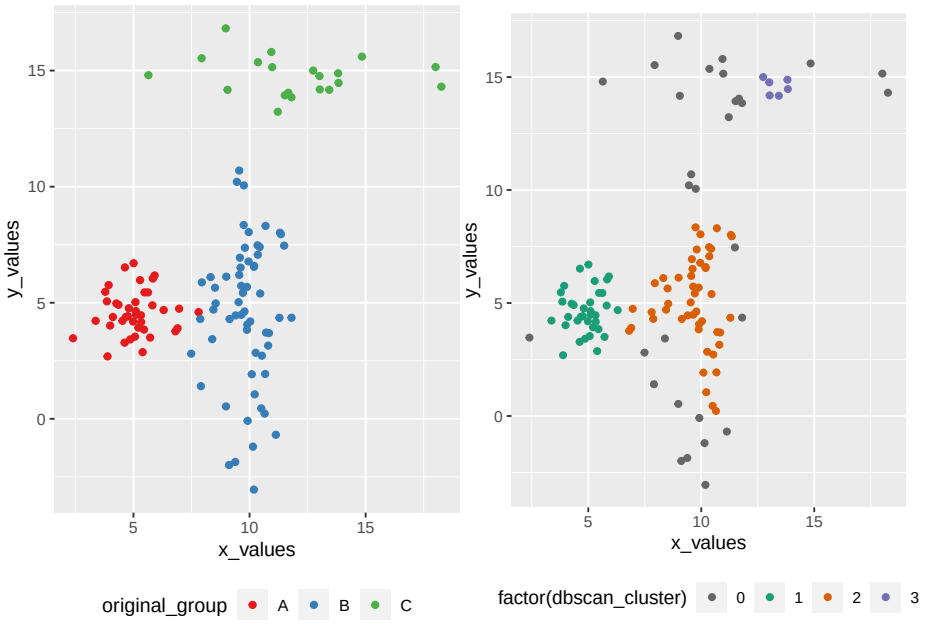
27.9 Density based clustering

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)
```

27.10 DBSCAN result



27.11 Summary

Hierarchical and density-based clustering

- Hierarchical
- Mixed
- Density-based

Next: Practical session

- Geodemographic classification

Chapter 28

kNN

28.1 Recap

Prev: Comparing data

- 321 Lecture Introduction to Machine Learning
- 322 Lecture Centroid-based clustering
- 323 Lecture Hierarchical and density-based clustering
- 324 Practical session

Now: kNN

- X
- Y
- Z

28.2 TO-DO

28.3 Summary

Support vector machines

- X
- Y
- Z

Next: TBC

- X
- Y
- Z

Chapter 29

Support vector machines

29.1 Recap

Prev: kNN

- X
- Y
- Z

Now: Support vector machines

- X
- Y
- Z

29.2 TO-DO

29.3 Summary

TBC

- X
- Y
- Z

Next: Deep learning

- X
- Y
- Z

Chapter 30

Deep learning

30.1 Recap

Prev: Support vector machines

- X
- Y
- Z

Now: Deep learning

- X
- Y
- Z

30.2 TO-DO

30.3 Summary

TBC

- X
- Y
- Z

Next: Practical session

- Support vector machines