Intro2R

R Environment and Syntax

Song, Xiao Ping
xp.song@u.nus.edu

Course materials: https://github.com/xp-song/Intro2R
updated 2022-10-18

Before we begin...

1. Navigate to course webpage and read background information
   https://github.com/xp-song/Intro2R

2. Ensure that you have installed **R** on your computer, followed by **R Studio**
   (follow links under 'Instructions' section of webpage)

3. Download workshop materials
   (green button on webpage)

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

What is R?

- Programming language and software environment with a command line interface
- RStudio is often used as a software client
- Both R and RStudio are open source software
- Huge library of packages created by the R community

About this crash course

What it IS

- Designed for those with minimal coding experience
- Give you a taste of what R can do

What it is NOT

- A substitute to practicing the fundamentals of the language
- A lesson in statistics

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Course materials

Intro2R

/notes [1]

/data

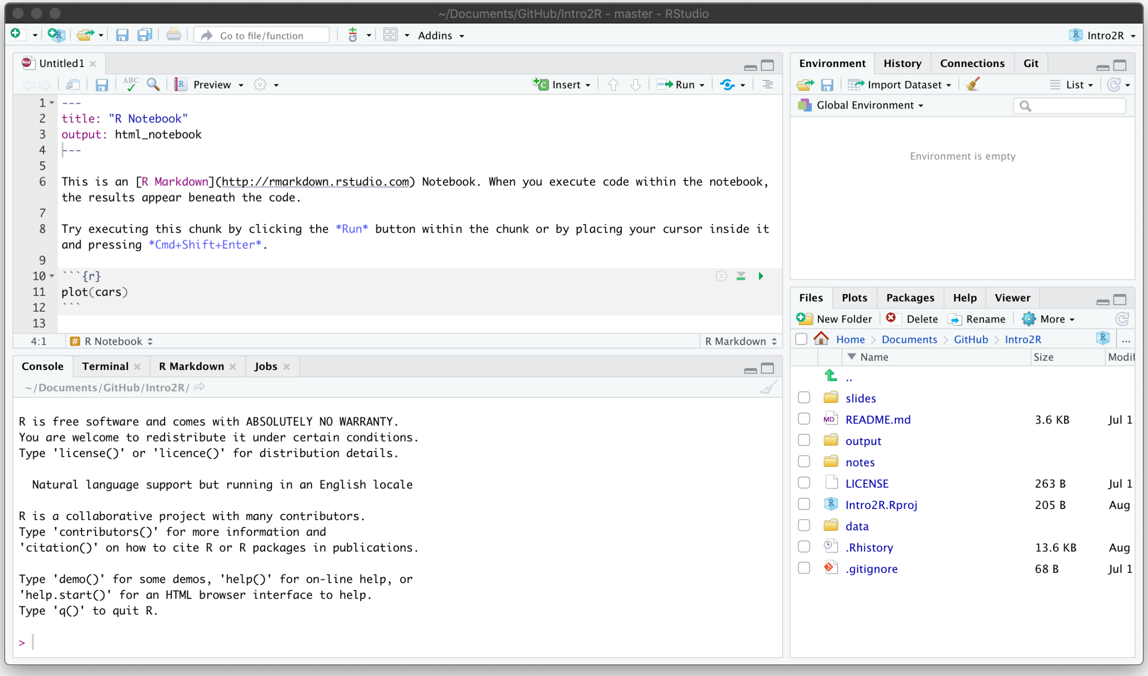
/output

PDF slide decks

Intro2R.Rproj

[1] View in your web browser by opening the '*.html*' files

R Studio Client



~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook,
the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it
and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

~~~

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Environment is empty

Name	Size	Modif
..		
slides		
README.md	3.6 KB	Jul 1
output		
notes		
LICENSE	263 B	Jul 1
Intro2R.Rproj	205 B	Aug
data		
.Rhistory	13.6 KB	Aug
.gitignore	68 B	Jul 1

- **Console:** Command line input/output
- **Script editor:** View/edit files that contain code
- **Environment/History**
- **Files/Plots/Packages/Help/Viewer**

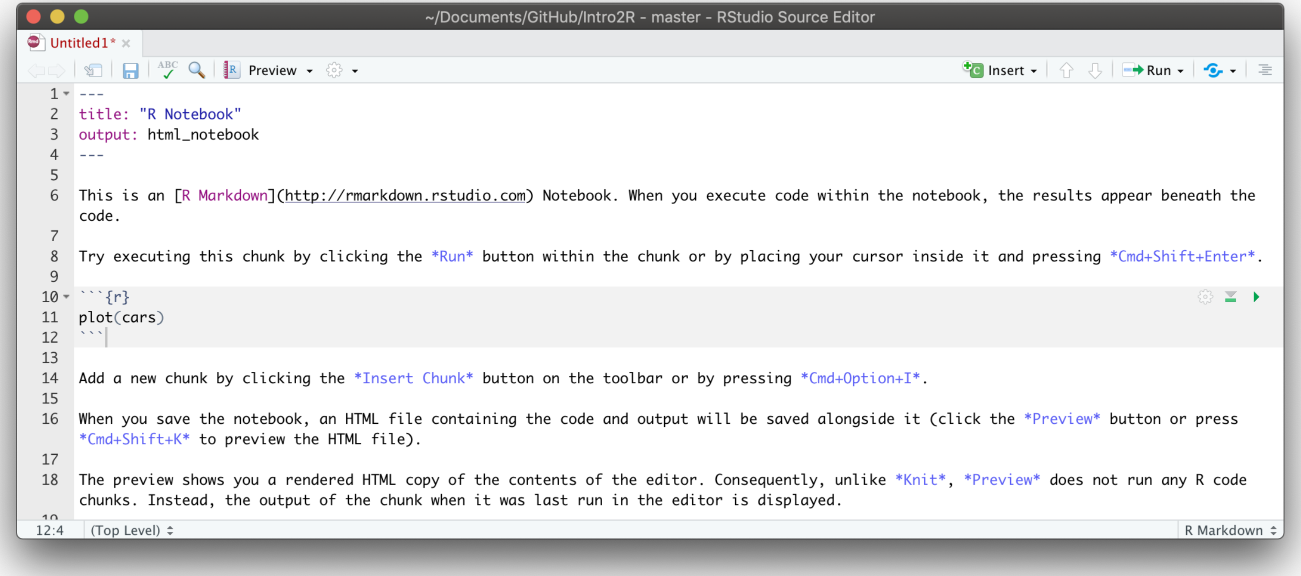
R Notebooks

What are R Notebooks?

- R Notebooks (a.k.a. R Markdown Notebooks) are files ending with '*.Rmd*'.
- Compared to basic '*.R*' scripts, they allows us to:
  - Write normal text alongside code
  - Interact with code within a single document
  - Generate (i.e. '*knit*') different types of files

Try creating a new R Notebook `File > New File > R Notebook`

R Notebooks



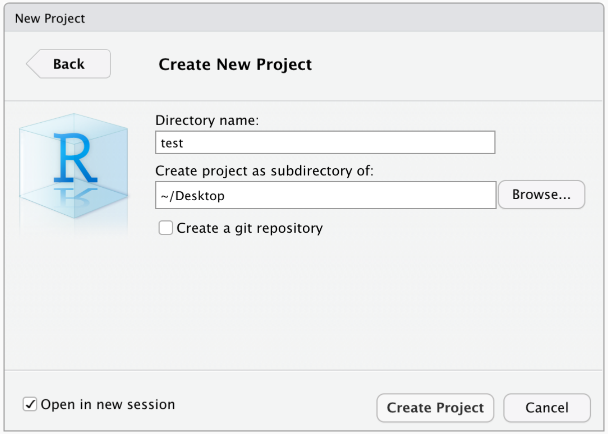
~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
~~~

- **Header section:** specify document parameters
- **Normal text**
- **Code chunk:** write code and specify code parameters

Save our new file as '*myanalysis.Rmd*'

RStudio Projects

Try creating a new RStudio Project `File > New Project > New Directory > New Project`



Directory name: test

Create project as subdirectory of: ~/Desktop

□ Create a git repository

☑ Open in new session

RStudio Projects

What are RStudio Projects?

- RStudio Projects help organise your work into separate 'R sessions'.
- Each project has it's own workspace a.k.a. 'working directory' (separate configuration, history, etc.)
- The location of the '*.RProj*' file defines the 'working directory'
    - **Type `getwd()` in the *console* of our new project**
    - This returns the absolute path to our working directory e.g. `/Users/<computer_username>/Desktop/test`

RStudio Projects

🌟 Best Practice

- Use *relative* paths in your script, based on *.RProj* file location
  - **Try reading in data in your R Notebook**

    ```
    read.csv("<path to Intro2R folder>/Intro2R/data/ozone_data.csv")
    ```

    ```
    read.csv("data/ozone_data.csv")
    ```

- Keep all project items in the working directory

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

General Syntax

Operators

Arithmetic

Solve the following:

$$\frac{(1+2)*(4-5)}{50}$$

```
(1+2)*(4-5)/50
```

```
## [1] -0.06
```

General Syntax

Operators

Logical

Check if `1e3` is larger or equal to `1*10^3`
What is the output?

```r
1e3 >= 1*10^3
```

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

General Syntax

Operators

Variables

Variables are named objects used to store data

- `<-` is used to assign variable names in R (e.g. `x <- 4`)
- Print variables by name (`x` vs. `"x"`)
- Assigning data to an existing variable overwrites it (`x <- 10`)

🌟 **Best Practice**

- Clear and consistent names
- Avoid numbers/symbols/whitespace

General Syntax

Operators

Variables

Data types and examples:

- Numeric (`3.142`), Integer (`5L`)
- Character (`"hello"`)
- Logical (`TRUE, FALSE`)
- Complex

Let's assign new variables `name`, `age`, and `weight`

Check the data type for each variable using the function

`is.numeric()`, `is.integer()`, `is.character()`

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Data Structures

Vectors

- Linear collection of data
- Must be of the *same* data type

Assign a *vector* of names to the variable `name`
(use the concatenate function `c()`)

```
name <- c("Me", "Tom", "Dick", "Harry", "Susan") # character vector
```

Assign a *vector* of numbers to the variable `age`

```
age <- c(20, 25, 30, 35, 40) # numeric vector
```

Data Structures

Vectors

- Linear collection of data
- Must be of the *same* data type
- *Operations in R are vectorised*

Subtract 5 from the vector `age`

```
age-5
```

```
## [1] 15 20 25 30 35
```

Add together two vectors

```
age+age
```

```
## [1] 40 50 60 70 80
```

Data Structures

Vectors

Lists

- Linear collection of data
- Can contain of different *types* and *structure* of data

Create a list with a mix of data types and variables

```
myteam <- list(name, age, "Group 1", 2019)
```

Data Structures

Vectors

Lists

- Linear collection of data
- Can contain of different *types* and *structure* of data

```
myteam
```

```
## [[1]]
## [1] "Me"    "Tom"   "Dick"  "Harry" "Susan"
##
## [[2]]
## [1] 20 25 30 35 40
##
## [[3]]
## [1] "Group 1"
##
## [[4]]
## [1] 2019
```

Data Structures

Vectors

Lists

Factors

- A special kind of vector that represents categorical data with discrete levels

Let's code the sex of each person in the variable `name`
(use the functions `factor()` and `c()`)

```
sex <- factor(c("M","M","M","M","F"))
```

```
sex
```

```
## [1] M M M M F
## Levels: F M
```

Data Structures

Vectors

Lists

Factors

- A special kind of vector that represents categorical data with discrete levels

Let's code the performance of each person in `name`

```r
perform <- factor(c("High", "Low", "Med", "Med", "High"))
```

```r
perform
```

```
## [1] High Low  Med  Med  High
## Levels: High Low Med
```

What is wrong with this output?

Data Structures

Vectors

Lists

Factors

Define the order using the `levels=` argument in `factor()`

```r
perform <- factor(c("High", "Low", "Med", "Med", "High"),
                  levels = c("Low", "Med", "High"))
perform
```

```
## [1] High Low  Med  Med  High
## Levels: Low Med High
```

Data Structures

Vectors

Lists

Factors

Matrices

- Tabular data (rows & columns)
- Must be of the *same* data type

Create a 4 by 3 matrix of sequential numbers

Use `matrix()` and the `:` operator to create a sequence

```
m <- matrix(1:12, nrow = 4)
```

```
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

- Tabular data (rows & columns)
- Rows represent data entries, columns represent different variables

Import the dataset `ozone_data.csv` into your R Notebook using `read.csv()`

```r
ozone <- read.csv("data/ozone_data.csv") # column headers in first row
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

View the first few rows of `ozone`

```
head(ozone) #print first few rows
```

```
##   rad temp wind ozone
## 1 190   67  7.4    41
## 2 118   72  8.0    36
## 3 149   74 12.6    12
## 4 313   62 11.5    18
## 5 299   65  8.6    23
## 6  99   59 13.8    19
```

Check the dimensions of `ozone`

```
dim(ozone)
```

```
## [1] 111   4
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

Check the names of `ozone` using `dimnames()`, `rownames()` and `colnames()`

```
dimnames(ozone)
```

```
## [[1]]
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12" 
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24" 
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36" 
##  [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48" 
##  [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60" 
##  [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72" 
##  [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84" 
##  [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96" 
##  [97] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108"
## [109] "109" "110" "111"
## 
## [[2]]
## [1] "rad"   "temp"  "wind"  "ozone"
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

Extract data by colnames using `$`
(output is a vector)

```
ozone$temp
```

```
##   [1] 67 72 74 62 65 59 61 69 66 68 58 64 66 57 68 62 59 73 61 61 67 81 79 76 82
##  [26] 90 87 82 77 72 65 73 76 84 85 81 83 83 88 92 92 89 73 81 80 81 82 84 87 85
##  [51] 74 86 85 82 86 88 86 83 81 81 81 82 89 90 90 86 82 80 77 79 76 78 78 77 72
##  [76] 79 81 86 97 94 96 94 91 92 93 93 87 84 80 78 75 73 81 76 77 71 71 78 67 76
## [101] 68 82 64 71 81 69 63 70 75 76 68
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

Create a dataframe with the vectors `name`, `sex`, `age` and `perform`

```
team_details <- data.frame(name, age, sex, perform)
```

```
team_details
```

```
##    name age sex perform
## 1    Me  20   M    High
## 2   Tom  25   M     Low
## 3  Dick  30   M     Med
## 4 Harry  35   M     Med
## 5 Susan  40   F    High
```

Back to operators…

Subsetting in R

What is 5th element in the vector `name`?

```r
name[5]
```

```
## [1] "Susan"
```

What is the 4th element of the column `name` in the dataframe `team_details`?
(hint: use `$` to extract columns as vectors)

```r
team_details$name[4]
```

```
## [1] "Harry"
```

Back to operators...

Subsetting in R

Extract the element in the 2nd row and 4th col in `team_details`

```r
team_details[2,4]
```

```
## [1] Low
## Levels: Low Med High
```

Extract 2nd row and all cols in `team_details`

```r
team_details[2,]
```

```
##   name age sex perform
## 2  Tom  25   M     Low
```

Extract the 4th col and all rows except the 2nd in `team_details`

```r
team_details[-2,4]
```

```
## [1] High Med  Med  High
## Levels: Low Med High
```

Back to operators...

Subsetting in R

Extract rows 1 to 3 in `team_details`

```
team_details[1:3,]
```

```
##   name age sex perform
## 1   Me  20   M    High
## 2  Tom  25   M     Low
## 3 Dick  30   M     Med
```

Extract rows 1 and 3 in `team_details`

```
team_details[c(1,3),]
```

```
##   name age sex perform
## 1   Me  20   M    High
## 3 Dick  30   M     Med
```

Back to operators...

Subsetting in R: Quick test!⚡

Extract R's built-in dataset `data(mtcars)`

Extract data on cars with a fuel efficiency of at least 20 mpg, and that are more than 108 hp

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- **Hint:** `mtcars[ & , ]`
- **Hint:** `mtcars[mtcars$mpg >= 20 & , ]`
- **Answer:** `mtcars[mtcars$mpg >= 20 & mtcars$hp > 108, ]`

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Functions

Overview

Functions have inputs and outputs (look up details with `?`)

E.g. Plot the performance distribution in `team_details`

```
plot(team_details$perform)
```

Functions

Overview

Functions have inputs and outputs (look up details with `?`)

E.g. Find the mean age in `team_details`

```
mean(team_details$age)
```

```
## [1] 30
```

E.g. Find the number of people (rows) in `team_details`

```
nrow(team_details)
```

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

Functions

Overview

User-defined

General structure when defining a function:

```
functionname <- function(inputs){
  # calculations...
  output
  }
```

Subsequent calls to the function:

`functionname(inputs)`

Functions

Overview

User-defined

E.g. Load `data/grades.csv` and assign it the name `grades`

```
##     subject grade grade_point credits
## 1      Math     A         4.5       5
## 2   English     B         3.5       5
## 3 Economics     C         2.0       4
## 4  Mandarin    B+         4.0       5
## 5     Music     F         1.0       0
## 6   History    C+         2.5       5
## 7   Intro2R    A+         5.0       1
```

Functions

Overview

User-defined

$$\frac{\sum_{i=1}^{n} gradepoint_i \times credits_i}{\sum_{i=1}^{n} credits_i}$$

Manually calculate the GPA in R using the formula

```
sum(grades$grade_point * grades$credits) / sum(grades$credits)
```

```
## [1] 3.42
```

Functions

Overview

User-defined

Create a function named `scorer` that:

- Takes a dataframe as input
- Outputs a calculation based on the colnames `grade_point` and `credits`

```
scorer <- function(x){
  sum(x$grade_point*x$credits) / sum(x$credits)
}
```

```
scorer(grades) #use function
```

```
## [1] 3.42
```

Functions

Overview

User-defined

Loops

- Loop functions repeat code `i` number of times
- Most common type: `for` loop

Functions

Overview

User-defined

Loops

Prepare our data inputs to the `for` loop:

Get the grades of other team members within `/data` folder

```
grades_tom <- read.csv("data/grades_tom.csv")
grades_dick <- read.csv("data/grades_dick.csv")
grades_harry <- read.csv("data/grades_harry.csv")
grades_susan <- read.csv("data/grades_susan.csv")
```

Put all these dataframes into a list named `team_grades`

```
team_grades <- list(grades, grades_tom, grades_dick, grades_harry, grades_sus
```

Functions

Overview

User-defined

Loops

For every item (person) in the list `team_grades`, use the function `scorer()` and append results to new column "GPA" in `team_details`

```
for(i in 1:length(team_grades)){
  team_details$GPA[i] <- scorer(team_grades[[i]])
  }
#the named object "i" changes in value with iteration of the loop
```

Who has the best grades in the team?

```
##    name age sex perform      GPA
## 1    Me  20   M    High 3.420000
## 2   Tom  25   M     Low 3.710526
## 3  Dick  30   M     Med 4.342105
## 4 Harry  35   M     Med 5.000000
## 5 Susan  40   F    High 4.342105
```

Functions

Overview

User-defined

Loops

Loop functions in base R

`lapply(x, FUN)`: Apply a function on each element of `x`, returns a *list*

`apply(x, MARGIN, FUN)`: Apply a function to tabular data by rows (`1`), cols (`2`), or both `c(1,2)`

Find the mean value for *each* numeric column in `team_details`

```
apply(team_details[,c(2,5)], 2, mean) #apply mean() function across columns
```

```
##       age       GPA
## 30.000000  4.162947
```

Functions

Overview

User-defined

Loops

Quick test!⚡

Calculate the mean for each numeric variable in `data(mtcars)`

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Answer:

```
apply(mtcars, 2, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec 
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750 
##         vs         am       gear       carb 
##   0.437500   0.406250   3.687500   2.812500
```

Questions?

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Useful Resources

Online tutorials

- R for Data Science
- Quick R
- R for cats (blog post)
- swirl (good for practice)
- R markdown cookbook

Online Q&A

- Stack Overflow
- How to ask a good question online
- Remember to check your `sessionInfo()` when troubleshooting!

Others

- Use R/RStudio from an external drive (if you don't have admin rights to install software)