# Basic R and how to read in data

This guide is partly based on online material from Amy Willis, Kiirsti Owen and Amelia McNamara, and the book "R for Data Science" by Hadley Wickham and Garrett Grolemund. Thank you amazing R community!

### R as a calculator

In the Console window below, type: 2+2 and press Enter Also try:

```
2^5
```

```
## [1] 32
```

```
3/10
```

```
## [1] 0.3
```

```
(3+5)^2
```

```
## [1] 64
```

```
sqrt(4)
```

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

Tip: To run a line (or multiple lines) of code from a script without typing them into the Console, select the line(s) you want to run and press Ctrl+Enter (Command+Enter on a Mac)

### Objects

R stores data as objects. You create new objects when you assign a value to them using "<-":

```
x <- 3  # Check the "Environment" window!
```

Tip: use the R studio shortcut Alt+ - (Alt and the minus sign) to easily create the assignment symbol <-

```
y <- 6
x+y
```

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

Tip: R is case sensitive so if you've defined your object as x, it will not recognise (capital) X. Similarly, the function for square root is sqrt, R will give you an error if you try to use Sqrt.

### Packages

Packages extend the functionality of base R. They are distributed via CRAN: the Comprehensive R Archive Network

To install a package, use: install.packages("packagename") You then need to load it, using library(packagename)

We will be using a collection of packages called the Tidyverse:

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.1     v stringr   1.5.2
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

When you load the tidyverse, you'll see a message about conflicts. As there is an (increasingly) large number of packages in R, it is possible to have functions with the same name in more than one package. The message tells you that packages dplyr and stats both have a function called filter and the one that will be used is the one from dplyr. It is the one that was loaded last.

If you want to use a function from a particular package, you need to include packagename:: before the name of the function.

In this example, you can use stats::filter() instead of just filter() to use filter from the stats package.

```
find("filter") # this shows you the packages a function belongs to, in order of priority
```

```
## [1] "package:dplyr" "package:stats"
```

The tidyverse packages we will be using mostly in this course are readr (for reading in data), dplyr (for transforming data) and ggplot2 (for plotting).

### Functions

When using the Tidyverse, you can call functions in two ways:

```
sqrt(4) # base R
```

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

```
4 %>%
  sqrt  # "pipe" operator (you can read is as "and then...")
```

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

Tip: use the R Studio shortcut Ctrl + Sft + M to create the pipe operator %>%

Tip: If you are not sure what a function does, type ?functionname in the Console, e.g. ?sqrt

### Reading in data

Before we read in our data, let's consider where we have saved our data file. Since we want our code to be reusable (by us and other people), the last thing we want is to include the location of the file in our code, something like:

"C:/dimitra/data/datafile.csv"

The above would only work for me, and only for the particular computer where folder "dimitra" contains a folder called "data".

To avoid these issues, we need to do two things:

1. Use R projects. (I hope you are doing that already!) Save the data and R markdown file inside the R project. Exactly where you save your code doesn't matter, you just need to note the location of your data with respect to the .Rproj file.

2. Use the R package "here". "Here" points to the location of the .Rproj file (which is the working directory for your project), so you just need to add "here" in front of the relative path to your data file.

For example, if your data file (a comma-separated value (csv) file) was saved inside a "data" directory, you would say:

library(here)

fev_data <- read_csv(here("data/fev.csv"))

To read in a file that is saved in the same directory as the .Rproj file:

```
library(here)
```

```
## here() starts at /home/myke/Desktop/Intro2HDS_R_WEEK5_MAIN
```

```
#fev_data <- read_csv(here("fev.csv"))
fev_data <- read_csv(here("INPUTS/fev.csv")) # Edit the path to reflect the correct folder structure
```

```
## Rows: 654 Columns: 7
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (7): seqnbr, subjid, age, fev, height, sex, smoke
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

(Remember to install the "here" package the first time.)

–> How would you use read_csv with the pipe operator?

```
# Add your code here!
here("INPUTS/fev.csv") %>%
  read_csv
```

```
## Rows: 654 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (7): seqnbr, subjid, age, fev, height, sex, smoke
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 654 x 7
##    seqnbr subjid   age   fev height   sex smoke
##     <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
##  1      1    301     9  1.71   57       0     0
##  2      2    451     8  1.72   67.5     0     0
##  3      3    501     7  1.72   54.5     0     0
##  4      4    642     9  1.56   53       1     0
##  5      5    901     9  1.90   57       1     0
##  6      6   1701     8  2.34   61       0     0
##  7      7   1752     6  1.92   58       0     0
##  8      8   1753     6  1.42   56       0     0
##  9      9   1901     8  1.99   58.5     0     0
## 10     10   1951     9  1.94   60       0     0
## # i 644 more rows
```

Look at the top few rows of the data:

```r
head(fev_data)
```

```
## # A tibble: 6 x 7
##   seqnbr subjid   age   fev height   sex smoke
##    <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1      1    301     9  1.71   57       0     0
## 2      2    451     8  1.72   67.5     0     0
## 3      3    501     7  1.72   54.5     0     0
## 4      4    642     9  1.56   53       1     0
## 5      5    901     9  1.90   57       1     0
## 6      6   1701     8  2.34   61       0     0
```

fev_data is a tibble - this is a tidyverse structure similar to a data frame (from base R) but with some differences:

- default printing is shorter
- tells you the column types (character, double, etc.)
- doesn't change the types of inputs

Tip: if your data is in a Microsoft Excel spreadsheet, you will need a different package to read it in, such as readxl. So you'll need:

install.packages("readxl")

library(readxl)

excel_data <- read_xlsx(filename, sheet = 1) #(to read the first sheet)

–> How would you read in a text file? (Check the data import cheat sheet!) There is a text file in your dataset so you can practice: psa.txt

```r
# Add your code here!
psa_data <- read_table(here("INPUTS/psa.txt"))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   ptid = col_double(),
##   nadirpsa = col_double(),
##   pretxpsa = col_double(),
##   ps = col_double(),
##   bss = col_double(),
##   grade = col_double(),
##   age = col_double(),
##   obstime = col_double(),
##   inrem = col_character()
## )
```

–> Have a look at the "Useful arguments" section of the data import cheat sheet. Use a few of them when you read in fev.csv and look at the data, is that what you expected?

```r
# Add your code here!
fev_data <- read_csv(here("INPUTS/fev.csv"), col_names = TRUE, n_max = 10)
```

```
## Rows: 10 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (7): seqnbr, subjid, age, fev, height, sex, smoke
##
```

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

–> Apply the summary function to a tibble. What does it do?

```
# Add your code here!
summarise(psa_data)
```

```
## # A tibble: 1 x 0
```

### Operating on data: columns

Individual columns are identified using the $ symbol:

```
head(fev_data$fev)
```

```
## [1] 1.708 1.724 1.720 1.558 1.895 2.336
```

```
summary(fev_data$fev)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.415   1.711   1.810   1.820   1.936   2.336
```

```
length(fev_data$fev)
```

```
## [1] 10
```

Other useful functions for tibbles and data frames:

```
names(fev_data)
```

```
## [1] "seqnbr" "subjid" "age"    "fev"    "height" "sex"    "smoke"
```

```
dim(fev_data)
```

```
## [1] 10  7
```

Other useful functions for columns:

```
max(fev_data$fev)
```

```
## [1] 2.336
```

```
mean(fev_data$fev)
```

```
## [1] 1.8204
```

```
sd(fev_data$fev)
```

```
## [1] 0.2557956
```