R Intro

SESSION 1

Introduction to R and Rstudio

Session 1.1

Installation

R can be downloaded from http://cran.r-project.org/mirrors.html. Choose your nearest location and whether you need the download for Windows, Mac or Linux.

RStudio is possibly the most user friendly Integraded Development Environment for R and can be downloaded from https://www.rstudio.com/products/rstudio/download/ by choosing the installer for the right platform (Windows, Mac etc).

RStudio layout

- **Command window (bottom left)** This is where you can write simple commands when you see the prompt (>) . To run a command here you just press enter.
- **Editor window (top left)** This is where you write scripts (among other things). I.e. a collection of commands that you want to save in a file to be able to run it again. You can open a new R script by File -> New -> R script. To run a command here you highlight it and hit Run (top right of Editor window). If you want to write a comment that will not be processed by R you can use # followed by your comment as follows.

```
# Anything I write after this sign will be ingored by R as long as it's on the same line
```

- **Environment/History window (top right)** In the environment window you can see which data/values are in the memory of the current R session. In the History window you can see what you have run before.
- **Files/Plots/Packages/Help window (bottom right)** In this window you can open files, see the plots you created, install and load packages and get help.

R-projects

R projects is an excellent way for the data scientist to organise each analysis/project using a meaningful file structure.

A simple file structure could be the following

Let's practice creating our own!

1. In RStudio go to File, "New Project"
2. Choose "Existing directory"
3. Choose the folder where you have downloaded the files for this course.
4. Click "Create Project"

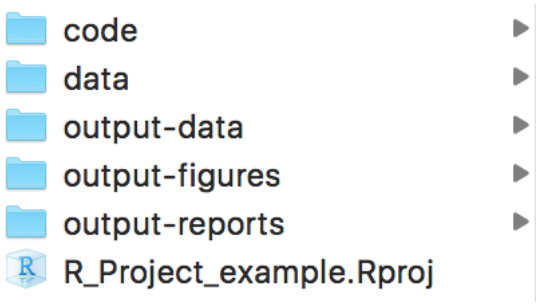

Figure 1:

5. You can create the folders that you would like in your directory either by using "New Folder" found in the Files tab of the bottom right window of R-studio or just by going to your directory on your computer.
6. **Important:** Once you have set up your project always start work on the project by opening the project from within RStudio or by double clicking on the project file in your computer's file explorer.

R as a calculator

```
2+3
```

1. Copy above in the command window and press enter
2. Copy in the editor window, highlight and press run

Variable assignment

You can assign "values" to variables using the assignment operator "<-". Assign value 5 to x and value "George" to y.

```
x <- 5
y <- "George"
```

As five is a numerical value no "" are necessary. Now type x and press enter. Then y and press enter to see what values are returned.

```
x
y
```

You can overwrite current entries by assigning a new value to a variable. Assign value "I have a dog" to x, then type x and press enter to see what you get.

```
x <- "I have a dog"
x
```

You can assign any type of value you want to a variable and we will now assign a sequence of numbers.

```
x <- 10:13
x
```

You can even create a spreadsheet-like grid of data using the function **data.frame**. Let's create a data.frame called **y** which will contain 2 variables in columns.

```
y <- data.frame(variable1=x, variable2=c("dog","cat","rat","mouse"))
y
```

Functions

R users commonly use existing functions to carry out analysis. Think of functions as **verbs** that 'do something' to variables and objects. For example you can use the function **sum** to obtain the sum of variable **x**. In order to do this you need to write the function followed by a parenthesis, which can include zero or more arguments (the objects that the function works on). In this case the parenthesis will include one argument which will be the variable we want the function to act on.

```
sum(x)
```

Data types

Different variable types in R are called classes. For example, the data types can be 'integer' (whole numbers), 'numeric' (continuous numbers), 'factor' (categorical data), 'character' (strings), and 'logical' (true/false). You can check what class a variable you are dealing with is by using the function **class**. Remember you have created 2 variables x and y. First type these in to remind yourselves of what they look like and then let's check what classes they are. Sometimes an object can have several classes just like a person can be 'Scottish' and an 'epidemiologist'.

```
x
y
class(x)
class(y)
```

You can also check what variable types are included in a data.frame, for example **y**, using the **$** sign.

```
class(y$variable1)
class(y$variable2)
```

As the functions used with each variable will depend on the variable class we sometimes need to change from one class to another. For example if you wanted to change **x** from an integer variable to a numeric variable you can use the function **as.numeric**. Check if it has worked using the **class** function.

```
x <- as.numeric(x)
class(x)
```

R help

There are many ways to get help and a search engine like Google is possibly your best friend. A quick way to get help within R when you know the name of a function is the **?** function. To obtain information about function **sum** type **?sum**. Results will be shown in your bottom right window. Also see(https://xkcd.com/627/)

```
?sum
```

Similary if you are unsure about whether a function to summarise data is called summary or summarise you can use the function **apropos** to see what functions are available including the term "summar"

```
apropos("summar")
```

Lastly, you can use the **tab key** to auto-complete. Type **aprop** and press the tab key.

R extension packages

Many R functions are included in Base R which is what you get when you first download R. Plenty more functions are included within extension packages created by users. The official repository where R packages

are stored is called CRAN. In order for a package to be published on CRAN it needs to pass several tests that ensure the package is following CRAN policies.

In order to get a package you need to download it and then load it into your current R session. Let's install package **epiR** which we will use in following sessions.

```
install.packages("epiR")
library(epiR)
```

Other repositories where packages can be found

Github is another repository R packages are often stored in. To install packages stored on github you need to use another package called **devtools** as shown below. Note: Unlike CRAN packages, there is no review process for github packages.

```
install.packages("devtools")
library(devtools)
install_github("ianhandel/epidemr")
library(epidemr)
```

Tidyverse

Tidyverse is a collection of packages designed to make the life of a data scientist easier and faster! Let's install them (if they are not already installed in our R session) and load them.

```
if (!require("tidyverse")) install.packages("tidyverse")
library(tidyverse)
```

The pipe operator "%>%"

Great! Now that we've learned about R packages, let's talk about the pipe operator, which is often used in conjuction with tidyverse to make our code easier to read. The pipe operator takes the output of one function and feeds it into the next function as its first argument. It might be easy to think of it as a "THEN". For example let's calculate the mean of "x", then round the result to the nearest whole number and then find the square root of that.

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

An alternative way to do that would be the following. Notice that they both give the same result, but the first example makes it much easier for the reader to follow the steps

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

Import datasets

There are different ways to load data into R depending on how they are stored. The most common will be importing data from a csv file. In order to do this we will use the functions **read_csv** or **read_excel**. We will try these later.

Example datasets

You can get a list of example datasets included in R for practice using **data()**. To look at any of the datasets just type their name.

```
data()
```

Working directory

Working directory is where any process running on your computer happens. This is for example where R will look/save things if you ask it to load or save a dataset when you don't specify it yourself.

In order to check the current working directory you need the function type.

```
getwd()
```

Saving datasets

In R you can save datasets created as a .csv using the function **write_csv**. Let's save dataframe **y** created earlier as a csv giving the name **y_dataset_2018-01-10.csv**. It's often good to include a creation date in a file name. The best format is YYYY-MM-DD. See (https://xkcd.com/1179/).

```
write_csv(y, path="y_dataset_2018-01-10.csv")
```

Session 1.2

Exercises

We will use one of the example datasets called **mtcars** to try out some of the functions already mentioned and more.

Ex 1

First have a look at the first 8 rows of this dataset using the function **head** and its dimensions using the function **dim**. Get a summary of each column in the dataset using function **summary** and look at the structure of your dataset using function **str**. The tidyverse has updated dataframes to an object called a **tibble**. Use **as_tibble** to convert a dataframe to a tibble and see how it looks.

```
head(mtcars, 8)
dim(mtcars)
summary(mtcars)
str(mtcars)
as_tibble(mtcars)
```

Ex 2

Use **$** to have a look at the first variable of mtcars.

```
mtcars$mpg
```

Ex 3

Let's now check the classes of mtcars and each variable in it. For example:

```
class(mtcars)
class(mtcars$mpg)
map(mtcars, class)
```

Ex 4

Try functions **summary**, **sum**, **max**, **mean**, **median** on each variable to obtain more information for each variable. Hint: use `map(mtcars, summary)` to do a summary on every column etc. Use help as explained above to see how you could make a table of variable **cyl**, find the minimun value of variable **wt** and the first quartile of variable **hp**.

Ex 5

Expore the whole dataset using **skim** from package **skimr**. Hint: Download and load package first!