STATS 2107
Statistical Modelling and Inference II
Practical 1: Data Cleaning

Jono Tuke

Semester 2 2022

In the practicals for SMI, you will learn how to use the R skills from SAM for data analysis.

Throughout these practicals you will look at how to analyse real datasets. The ideas from todays practical will assist you in your first assignment.

In this practical you will

- recap the basics of R and Rstudio,
- read the data into R,
- learn some of the basics of cleaning data,
- recap dataframes, and
- make some univariate plots.

Opening Rstudio

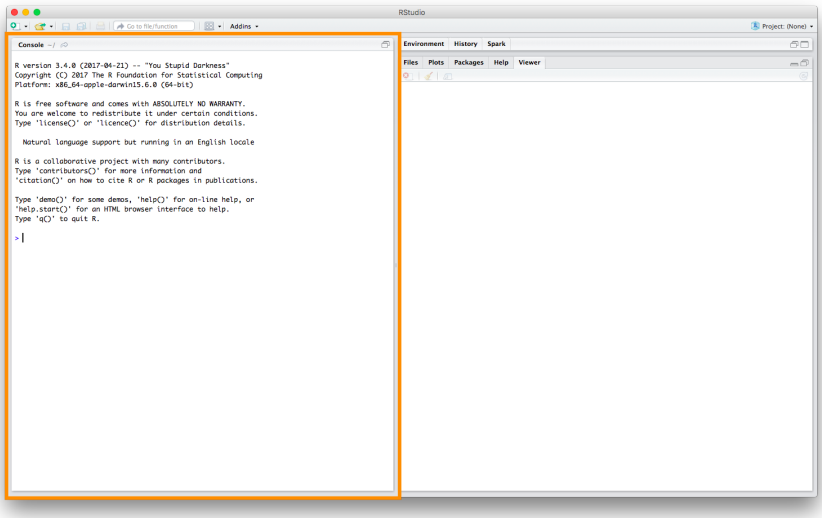
First log-on to your computer and then open Rstudio. If you would like to do this on your home computer you need to install R:

https://cran.r-project.org/

and then Rstudio

https://www.rstudio.com/

Let us have a good look at Rstudio. The main thing to look at is the console. See the orange box in the figure.



This is where you can type commands and get answers. For example, imagine that you are sad enough to want to perform the complicated calculation that is one plus one. Place your cursor into the console next to the prompt `>` and type `1+1`. Hit enter and you should get

```
1+1
```

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

Brilliant - worth twice the amount you paid for it.

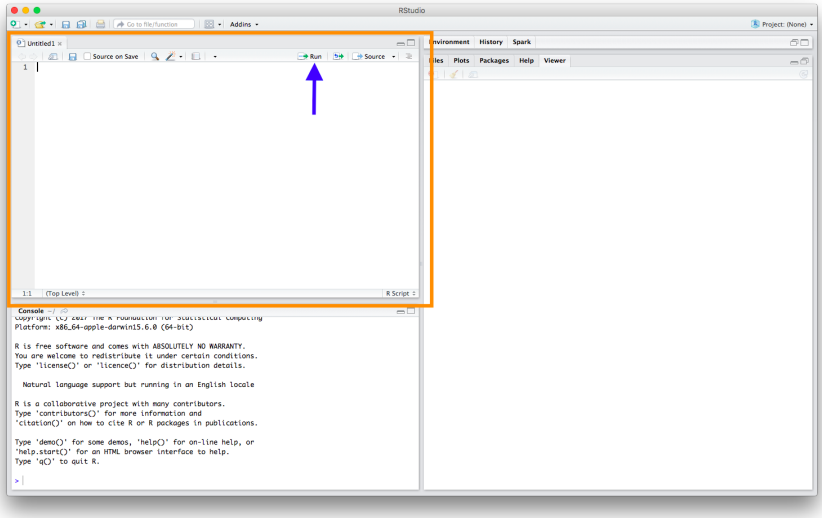
Writing scripts

Two panels - that is not enough - we need more. Well, wait no longer, let's get some more panels.

Go to

`File > New File > R Script`

And now we have



The new window (shown in orange) is a blank script file. This gives us a space to type commands and save them for later.

Click into the file and type

```
print("Hello World")
```

Now while your cursor is on the same line, click the `Run` button (blue arrow). This will send the command down to the console and run it.

```
> print("Hello World")
[1] "Hello World"
```

You can now save this file for later. We would recommend this. You are going to save a copy of your work, and this is how to do it. You should save the file with the extension `.R` so something like: `SMIP1.R`.

Packages

A package or library are extra add-on code that can be installed into R. These give extra functions that can be useful. Using a package is a two-step process.

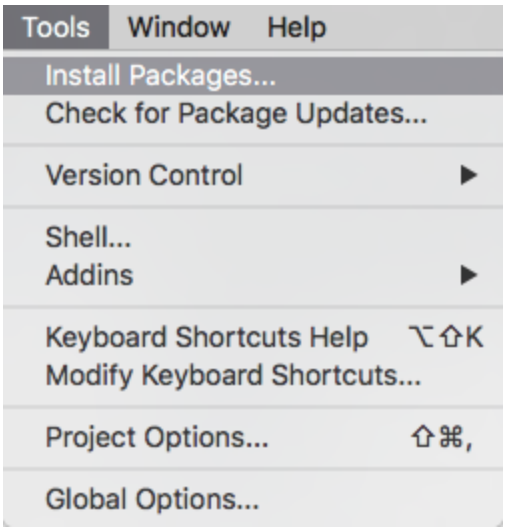
1. If you do not already have the package on your computer, then you must install it.
2. Each time you start R, then you must load the package if you need it.

Installing extra packages

Often you'll need to install extra packages in R in order to do various things. Fortunately, R makes it particularly easy to install packages – it's as simple as going to the top menu bar and clicking

`Tools > Install Packages...`

and then typing the name of the package you want.



In this course we'll be making use of the `tidyverse` package. Make this the first package you install[1]. Alternatively, you can install packages from the console, like this:

```
install.packages("tidyverse")
```

Try it out in the console of Rstudio on your home computer. Don't forget that you'll need to be connected the internet.

[1] This is already installed in EMG13.

Loading packages

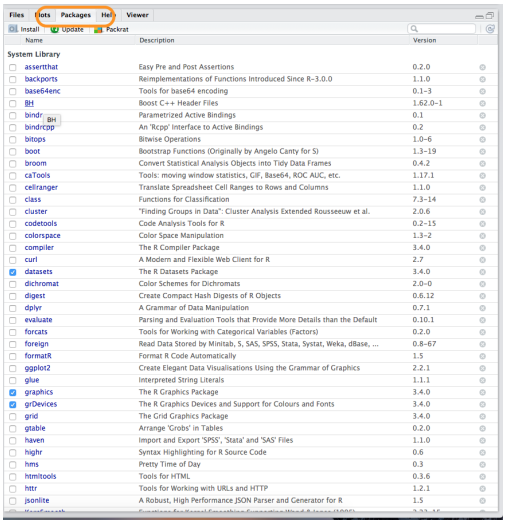
Even if you have a package installed, R does not know about it until you load it. You need to do this every time you restart R. In this practical, we need the `tidyverse` and the `readxl` packages. Load these.

Do not forget to put these commands in your script, not directly into your console!!

```
library(tidyverse)
library(readxl)
```

Viewing installed packages

You can see what packages are installed on your system in the packages tab.

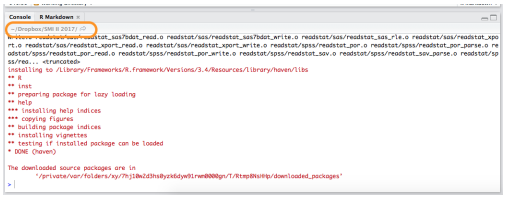


Those with ticks are loaded.

Working directory

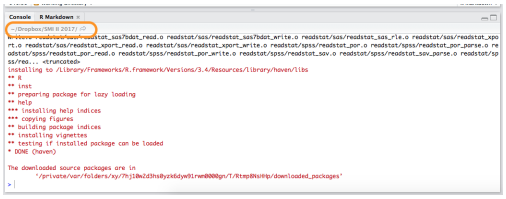
R has a place that it is looking at for your files. It is called the working directory. You can out where this is in two ways:

1. Look at the top of the console.



2. Use the command `getwd()`.

To set the working directory, you can click on the `...` in the Files tab:



I suggest creating a folder in your drive called `Practical 1` and saving all the files there.

Dirty dataset

I have simulated some dirty data for you. This is in an excel spreadsheet called `dirty_data.xlsx`.

Download this spreadsheet and save it somewhere on your `U` drive.

Reading in data

Read the data into R using

```
dirty_data  <- read_excel("dirty_data.xlsx")
```

Note that the pathway `"dirty_data.xlsx"` will only work if you have set the working directory to be the one where you have save the `dirty_data.xlsx` dataset.

```
getwd()
```

```
## [1] "/Users/matthew/Desktop/SMI_2022/practicals"
```

So in my case, my working directory is the practical folder of the SMI folder on my computer. You can set the working director using the

```
setwd()
```

command. We can check the data is loaded with

```
dirty_data
```

```
## # A tibble: 32 x 5
##    gender name     weight             height transport
##    <chr>  <chr>    <chr>               <dbl> <chr>
##  1 female Amber    55.86                160. bike
##  2 female Leah     65.08                168. bike
##  3 female Kellie   56.58                161. car
##  4 female Tamera   66.02                167. car
##  5 Female Courtney 71.319999999999993   171. bike
##  6 female Margaret 66.25                166. bike
##  7 female Amanda   66.290000000000006   167. Bkie
##  8 female Samantha 73.930000000000007   173. bike
##  9 female Tia      58.9                 162. car
## 10 female Shantae  63.8                 169. car
## # ... with 22 more rows
```

Quiz Questions (see solutions for answers)

1. What are the subjects?
2. How many subjects are there?
3. How many variables are there?
4. What type of variable is `gender`?

Dataframes (tibbles)

The standard form for storing data in R is the dataframe. The modern form of the data frame - introduced in `tidyverse` - is the tibble (basically it gives nice looking output). To get individual rows of a dataframe we can use the square bracket notation:

```
dirty_data[1, ]
```

```
## # A tibble: 1 x 5
##   gender name  weight height transport
##   <chr>  <chr> <chr>   <dbl> <chr>
## 1 female Amber 55.86    160. bike
```

This gives us the first row of the `dirty_data` dataframe. To get columns we can use

```
dirty_data[, 1]
```

```
## # A tibble: 32 x 1
##    gender
##    <chr>
##  1 female
##  2 female
##  3 female
##  4 female
##  5 Female
##  6 female
##  7 female
##  8 female
##  9 female
## 10 female
## # ... with 22 more rows
```

which gives the first column, or even better we can use the `$` notation which outputs the values for each variable.

```
dirty_data$name
```

```
##  [1] "Amber"     "Leah"      "Kellie"    "Tamera"    "Courtney"  "Margaret"
##  [7] "Amanda"    "Samantha"  "Tia"       "Shantae"   "Kaitlyn"   "Mattie"
## [13] "Brittney"  "Sarah"     "Rachel"    "Arielle"   "Koleby"    "Jonathan"
## [19] "Michael"   "Andrew"    "Logan"     "Joseph"    "Jared"     "Nicholas"
## [25] "Davis"     "Travis"    "Samuel"    "Mark"      "Lee"       "Nathaniel"
## [31] "Morgan"    "Stephen"
```

Later in these practicals, I will show you how to use `dplyr` to be more precise in manipulating dataframes.

Quiz Questions

5. What is the value of the tenth row fourth column?
6. What is the R command to get the weights of the subjects?

Cleaning the data

As already stated, this data is dirty and needs some cleaning. We will start by looking at the variable `gender`.

Recoding missing data

The first step in data cleaning is to make sure that R knows where there is missing data.

Lets start by looking at a summary table of `gender` using the `table()` command:

```
table(dirty_data$gender)
```

```
##
##      -      F female Female   male   Male
##      1      1     14      1     14      1
```

We see that one of the entries is "-". We have spoken to the researcher and these are how they have entered missing values in the spreadsheet. We need to let R know that these are missing values, which R denotes as `NA`.

```
# Find all prices that are "NA" and set to NA.
dirty_data$gender[dirty_data$gender == "-"] <- NA
```

How does this work:

We select a subset of the values using the square brackets:

```
dirty_data$gender[..]
```

The bit in the square brackets `dirty_data$gender=="-"` tells R which to select, those that have the value "-". Note the use of a double equals.

Check it works

```
table(dirty_data$gender)
```

```
##
##      F female Female   male   Male
##      1     14      1     14      1
```

Notice that they have disappeared as R knows that there are missing values. How do we know that they are still there?

Easy, add the extra `useNA = 'always'` to show them:

```
table(dirty_data$gender, useNA = 'always')
```

```
##
##      F female Female   male   Male   <NA>
##      1     14      1     14      1      1
```

Consolidating levels

Notice that we still have some problems. We have three ways to code female: "F", "Female", and "female". Also we have two ways to code male: "male" and "Male". Lets clean this up. The best way is to use the `fct_recode()` function in the `forcats` package.

```
dirty_data$gender  <- fct_recode(dirty_data$gender,
                                 female = "Female",
                                 female = "F",
                                 male = "Male")
table(dirty_data$gender)
```

```
##
## female   male
##     16     15
```

Redefining the class of the variable

Lets have a look at the variable weight

```
dirty_data
```

```
## # A tibble: 32 x 5
##    gender name     weight             height transport
##    <fct>  <chr>    <chr>               <dbl> <chr>
##  1 female Amber    55.86                160. bike
##  2 female Leah     65.08                168. bike
##  3 female Kellie   56.58                161. car
##  4 female Tamera   66.02                167. car
##  5 female Courtney 71.319999999999993   171. bike
##  6 female Margaret 66.25                166. bike
##  7 female Amanda   66.290000000000006   167. Bkie
##  8 female Samantha 73.930000000000007   173. bike
##  9 female Tia      58.9                 162. car
## 10 female Shantae  63.8                 169. car
## # ... with 22 more rows
```

Noticed it is listed as `weight <chr>`. This can also be checked by asking the variables class

```
class(dirty_data$weight)
```

```
## [1] "character"
```

So R thinks that weight consists of letters (character), but we know from the researcher that weight is a quantitative variable. We need to convert `weight` to a numeric variable.

First check if there are any missing values and convert them to `NA`.

Next we can convert it to a numerical variable.

```
# Convert weight to number
dirty_data$weight  <- as.numeric(dirty_data$weight)
```

So that is the first two steps of cleaning data: set missing values and make sure that R has the correct variable type.

Quiz Questions

7. Have a go at cleaning the remaining columns.

Does the data make sense?

Summary statistics

We can get simple summary statistics with

```
summary(dirty_data$weight)
```

Individual summary statistics can be obtained with functions such as

```
mean(dirty_data$weight, na.rm = TRUE)
sd(dirty_data$weight, na.rm = TRUE)
median(dirty_data$weight, na.rm = TRUE)
```

Univariate plots

Now that the weight data is cleaner, we will produce some univariate plots. We will start with a histogram of weight:

```
ggplot(dirty_data,aes(weight)) + geom_histogram()
```

This uses `ggplot` to produce the plot. The form is

- `ggplot()`: this creates the canvas. The first argument is the data frame, the `aes()` command tells `ggplot` what columns to use, in this case `weight`.
- `+ geom_histogram()`: this tells ggplot to add a histogram.

Have a look at the help page:

http://ggplot2.tidyverse.org/reference/geom_histogram.html

If you have a categorical variable, then a bar-chart is preferable:

```
ggplot(dirty_data, aes(transport)) + geom_bar()
```

Quiz Questions

9. What do you notice?

Saving the data

We can save the data frame in the standard form for R to use with the following command.

```
write_rds(dirty_data, "dirty_data.rds")
```

We can then reload the data next time with the command

```
dirty_data  <- read_rds("dirty_data.rds")
```