Reading data in R

Work flow within RStudio

You can type on the "command line" in the console window. However, just as with bash, you will want to save your code in a script. Saving code is easy in R Studio.

- Select `File - New Project`
- Select `New Directory` and `Empty Project`

- Give the project a name

- Browse to the subdirectory you would like to do your work

To keep track of the commands you are running in a script:

- Select `File - New File - R Script`

Now you have a fourth window to keep track of commands.

R Packages

It is possible to add functions to R by obtaining a package written by someone else. As of this writing, there are over 7,000 packages available on CRAN (the comprehensive R archive network). R and RStudio have functionality for managing packages:

- You can see what packages are installed by typing `installed.packages()`
- You can install packages by typing `install.packages("packagename")`, where `packagename` is the package name, in quotes.
- You can update installed packages by typing `update.packages()`
- You can remove a package with `remove.packages("packagename")`
- You can make a package you have installed available for use in your session with `library(packagename)`

Install the following packages: `tidyverse`, `gapminder`.

###Loading the data

- Load the packages you installed

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0       v purrr   0.3.0
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.2       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.3.0

## Warning: package 'tibble' was built under R version 3.5.2

## Warning: package 'dplyr' was built under R version 3.5.2

## Warning: package 'stringr' was built under R version 3.5.2

## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(gapminder)
```

If you're curious about where this data comes from you might like to look at the Gapminder website.

Unlike bash scripts RStudio offers you great flexibility in running code from within the editor window. To run a line of code from a script click on the line and click on the `Run` button just above the editor panel, or hit Ctrl-Enter in Windows / Linux or Command-Enter on OS X. To run a block of code, select it and then `Run`.

As in bash, we can use head to view the first few lines of the file.

```r
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Examining the data

Let's have a look at our example data (life expectancy in various countries for various years).

There are a few functions we can use to interrogate data structures in R. Let's use them to explore the gapminder dataset.

The first thing you should do when reading data in, is check that it matches what you expect, even if the command ran without warnings or errors. The `str` function, short for "structure", is really useful for this:

```r
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 222
##  $ gdpPercap: num  779 821 853 836 740 ...
```

We can see that the object is a `tbl_df`, `tbl` and `data.frame` with 1,704 observations (rows), and 6 variables (columns). Below that, we see the name of each column, followed by a ":", followed by the type of variable in that column, along with the first few entries.

The `class` function will show us the type of data structure (a data frame) without additional information.

```r
class(gapminder)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

The `typeof` function shows us the type of data in the variable.

```r
typeof(gapminder)
```

```
## [1] "list"
```

Here we see that data frames are lists 'under the hood'.

The gapminder data is stored in a “data.frame”. This is the default data structure when you read in data; it is useful for storing data with mixed types of columns.

Let’s look at some of the columns individually. We can access each column using the name of the dataset followed by a $ and then the name of the column. For example:

```
gapminder$year
```

```
## [1] 1952 1957 1962 1967 1972 1977
```

We can also look at the characteristics of the data in the column:

```
typeof(gapminder$year)
```

```
## [1] "integer"
```

```
typeof(gapminder$lifeExp)
```

```
## [1] "double"
```

A “double” is a decimal value.

Can anyone guess what we should expect the type of the continent column to be?

```
typeof(gapminder$continent)
```

```
## [1] "integer"
```

If you were expecting a the answer to be “character”, you would rightly be surprised by the answer. Let’s take a look at the class of this column:

```
class(gapminder$continent)
```

```
## [1] "factor"
```

In R text columns often represent categorical data, which need to be factors to be handled appropriately by the statistical modeling functions in R.

However it’s not obvious behaviour, and something that trips many people up. We can disable this behaviour when we read in data and we’ll discuss this later. Remember, if you do turn off automatic conversion to factors you will need to explicitly convert the variables into factors when running statistical models. This can be useful, because it forces you to think about the question you’re asking, and makes it easier to specify the ordering of the categories.

We can view the column names of the data.frame using the function `colnames`:

```
colnames(gapminder)
```

```
## [1] "country"   "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
```

You can also change the column names this way (use a copy of the data rather than changing the column names of the original data).

```
copy <- gapminder
colnames(copy) <- letters[1:6]
head(copy, n=3)
```

```
## # A tibble: 3 x 6
##   a           b         c     d        e     f
##   <fct>       <fct> <int> <dbl>    <int> <dbl>
## 1 Afghanistan Asia   1952  28.8  8425333  779.
## 2 Afghanistan Asia   1957  30.3  9240934  821.
## 3 Afghanistan Asia   1962  32.0 10267083  853.
```

Similarly, we can view the row names of the data.frame using the function `rownames`:

```
rownames(gapminder)[1:20]
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20"
```

See those numbers in the square brackets on the left? That tells you the number of the first entry in that row of output. So we see that for the 5th row, the rowname is "5". In this case, the rownames are simply the row numbers.