R Workshop Chandler House: Part 1

Masa Vujovic

14th November 2018

Contents

Installing packages

```
install.packages("wordbankr")
```

Loading libraries

```
library(wordbankr)
library(plyr)
```

Data structures in R

Vectors: character, logical, or numeric

```
char <- c("hellow", "world")
typeof(char)
```

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

```
length(char)
```

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

```
logic <- c(TRUE, FALSE, TRUE)
typeof(logic)
```

```
## [1] "logical"
```

```
num <- c(1,2,3,4)
typeof(num)
```

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

```
is.numeric(num)
```

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

Lists

A list is a special type of vector. Each element can be a different type.

```
l <- list(1, TRUE, "hello")
typeof(l)
```

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

```
l[[3]]
```

```
## [1] "hello"
```

Think about square versus round brackets!

Data frames

We are now going to look at data frames - this is pretty much *the* data structure for data analysis. It is essentialy a secial type of list where every element of the list has the same length.

Data frames are created by default when you load data from a csv file using the read.csv() function:

```
wg <- read.csv("WG_Part1.csv", sep="\t")
```

We can now start to manipulate and analyse our data. For that, we need functions.

Functions in R

We will look at three types of functions: in-built functions, which are part of R, functions which are parts of R packages that need to be installed separately, and finally, custom-written functions which you write yourself.

In-built functions

Functions take *arguments*. The simplest case, the in-built mean() function, *takes* a vector, x, and *returns* the arithmetic mean:

```
# Notice the dollar sign operator - it is used to refer to specific columns within a dataframe.
mean(wg$age)
```

```
## Warning in mean.default(wg$age): argument is not numeric or logical:
## returning NA
```

```
## [1] NA
```

The error "argument is not numeric or logical: returning NA" does not make sense as "age" *is* a numeric variable. Let's take a closer look at our dataset.

```
dim(wg)
```

```
## [1] 10035     1
```

This suggests that our dataset has 10035 rows and 1 column, but this is wrong - it definitely has more than 1 column! Let's take a look at the actual file:

```
head(wg)
```

```
##   data_id.lang.form.age.sex.mom_ed.comprehension.production
## 1                     57475,Norwegian,WG,8,Male,Graduate,3,0
## 2                   57476,Norwegian,WG,9,Female,Graduate,0,0
## 3                   57477,Norwegian,WG,8,Male,Secondary,25,4
## 4                   57478,Norwegian,WG,9,Male,Secondary,18,2
## 5                  57479,Norwegian,WG,10,Female,College,12,1
## 6                   57480,Norwegian,WG,8,Male,Secondary,21,0
```

The file doesn't seem to be comma-separated! This is because we read it as a tab separated file -

```
wg <- read.csv("WG_Part1.csv", sep = "\t")
```

Let's change it back to a comma separated file:

```
wg <- read.csv("WG_Part1.csv", sep = ",")
```

```
dim(wg)
```

```
## [1] 10035     8
```

```
mean(wg$age)
```

```
## [1] 13.49537
```

That looks better!

Actually, in the read.csv() function, sep is set to "," by default, so you don't need to specify it. The following gives the same result:

```
wg <- read.csv("WG_Part1.csv")
```

Functions from a package

Functions can come from specific packages. Let's take a look at the select function:

```
wg <- select(wg, data_id, language, sex, production)
```

```
## Error in eval(expr, envir, enclos): could not find function "select"
```

The problem here is that we are calling a function without having installed and loaded the package this function comes from. So let's do that first:

```
install.packages("dplyr")
```

```
library(dplyr)
```

Let's try calling select() again:

```
wg <- select(wg, data_id, language, sex, production)
```

```
## Error in FUN(X[[i]], ...): object 'language' not found
```

What does the error message "object 'language' not found" mean?

Let's look at our dataset again to see why select() can't find language.

```
head(wg)
```

```
##   data_id      lang form age    sex    mom_ed comprehension production
## 1   57475 Norwegian   WG   8   Male  Graduate             3          0
## 2   57476 Norwegian   WG   9 Female  Graduate             0          0
## 3   57477 Norwegian   WG   8   Male Secondary            25          4
## 4   57478 Norwegian   WG   9   Male Secondary            18          2
## 5   57479 Norwegian   WG  10 Female   College            12          1
## 6   57480 Norwegian   WG   8   Male Secondary            21          0
```

The problem is that I deliberately changed the column name from "language" to "lang" beforehand. I did this to illustrate a common problem when loading datasets. Here is how we can rename the column:

```
colnames(wg)[colnames(wg)=="lang"] <- "language"
```

Let's check if this worked. There are many ways to do this. Here is one:

```
"language" %in% names(wg)
```

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

Alternatively, we could do this:

```
names(wg)
```

```
## [1] "data_id"       "language"      "form"          "age"
## [5] "sex"           "mom_ed"        "comprehension" "production"
```

This might not be a good idea if you are working with large data sets though.

Custom functions

You can also write your own functions. Here, I have written a very simple function which takes a numeric vector and returns the arithmetic mean. This is just to illustrate writing and using custom-written functions - of course, you can always just use the in-built one!

```
my_mean <- function(x){
  sum <- 0
  for(i in 1:length(x)){
    sum <- sum + x[i]
  }
  mean <- sum/length(x)
  return(mean)
}
```

Let's now call our function:

```
my_vector <- c(345, 1, 23, 456, 518)
my_mean(my_vector)
```

```
## [1] 268.6
```

And just to check that our function is correct:

```
mean(my_vector)
```

```
## [1] 268.6
```

Ta-dah!

Activity 1

Load your own data set using the read.csv("filename.csv") function, where "filename" is replaced by the actual name of your file. Note: this function will only work if your data are in a csv format. It might be worth converting your file into a csv file - csv files are great for loading in data.

If possible, copy your data in the same folder where your R script is. A common problem with loading datasets is not specifying the correct path: your R script may be in one folder, but your data file is in another, so simply calling "filename.csv" won't work - you need to tell R the full path that leads to your data file. If, for example, your R script is in one folder, but the data file is in a "data" sub-folder, when you load your data, you should do the following: read.csv("data/filename.csv") - now you have told R where the file is.

Tip: if you have a datafile from SPSS, you can read it using the read.spss() function. In order to use that function, you need to load the library "foreign". This library is built into the R base, so you don't need to install it as a package first.

```
library(foreign)
```

Activity 2

After you have successfully loaded in the data, we can start thinking about functions which might be useful in data analysis and manipulation. Lee and Cat will give you lots of examples of specific functions you can use in data manipulation, analysis and visualization. For now, let's try some simple in-built functions:

head()

tail()

dim()

str()

What do you think each of these functions do?

You can also calculate the mean of some of your variables, for example.