Statistics for Data Science / Exercise 01

Pierpaolo Brutti

Part I: R Syntax & Functions

Take a shot at the following questions.

a. For each of the following commands, either explain why they should be errors, or explain the non-erroneous result.

```
vector1 <- c("5", "12", "7", "32")
max(vector1)
sort(vector1)
sum(vector1)
```

b. For the next series of commands, either explain their results, or why they should produce errors.

```
vector2   <- c("5", 7, 12)
vector2[2] + vector2[3]
```

```
dataframe3 <- data.frame(z1 = "5", z2 = 7, z3 = 12)
dataframe3[1,2] + dataframe3[1,3]
```

```
list4 <- list(z1 = "6", z2 = 42, z3 = "49", z4 = 126)
list4[[2]] + list4[[4]]
list4[2]   + list4[4]
```

c. The colon operator `:` creates a sequence of integers in order and it is a special case of the function `seq()`. Using the help command `?seq` to learn about the function, design an expression that will give you the sequence of numbers from 1 to 10000 in increments of 372. Design another that will give you a sequence between 1 and 10000 that is exactly 50 numbers in length.

d. The function `rep()` repeats a vector some number of times. Using the help command `?rep`, explain the difference between `rep(1:3, times = 3)` and `rep(1:3, each = 3)`.

Part II: "Massage" your data

The data set you.tsv contains some data in a `\tab` separated format. Start downloading the `.tsv` file into your project/working directory.

a. First, we need to load the dataset into R using the command `read.table()` or, even better in this case, `read.delim()`. Use the help function `?read.table` to learn what arguments this function takes. If you go for `read.delim()`, you should be happy with the very basic call already, **but** please, take a look also at the `stringsAsFactors` option. Once you have sorted out the necessary input, load the data set into an R data.frame called `youraw`. At this point use the command `str()` to take a look. You should see something like this (if not, check your code)

```
str(youraw)
```

```
## 'data.frame':    68 obs. of  6 variables:
##  $ Where.are.you.from.                                              : chr  "Spain" "Italy" "Italy
##  $ Previous..academic..life.                                        : chr  "Physics" "Statistics"
##  $ How.well.do.you.know..Probability                                : int  3 4 4 2 2 4 2 2 3 3 ..
```

```
##  $ How.well.do.you.know..Statistics                                  : int  3 4 4 2 2 2 4 4 3 2 ..
##  $ How.well.do.you.know..R                                           : int  1 4 3 3 2 0 3 0 4 4 ..
##  $ Besides.R..do.you.fluently.know.use.other.programming.languages.: chr  "Yes, C++, Matlab, Bas
```

b. How many rows and columns does `youraw` have? (If there are not 68 rows and 6 columns, something is wrong; check the previous part).

c. What are the names of the columns of `youraw`?

d. What is the value of row 7, column 5 of `youraw`?

e. Display the second row of `youraw` in its entirety.

From now on, we will try to clean `youraw` a little bit.

First of all, lets make a working copy of `youraw`

```
youclean <- youraw
```

f. Now explain what this command does

```
names(youclean) <- c("where", "prev.life",
                     "good.prob", "good.stat",
                     "good.R", "other.lan")
```

by running it on your data and examining (with suitable functions!) the object `youclean`.

g. As you may have noticed, the variable now called `other.lan` is of type `character` and it stores a `No` or an unstructured sequence of programming languages that people in this survey declare to know beside `R`. Your goal is to create two new variables inside the dataset `youclean` – named `lan1` and `lan2` – which contains the first two options listed in `other.lan`. In case less than two options are provided, just fill in the new variables with `NA` as needed. To tackle this points you may wanna know a bit more about *string manipulation* in `R`. The `stringr` package can be useful, together with functions in `base R` like `grep()`, `gsub()`, `strsplit()`, `paste()`, etc.

h. Very last step! Do you remember what a `factor` is? Good, run the following code on your data and explain what it does. In particular, explain the difference you see between the first three and the last three lines of code.

```
# Before
class(youclean$where)
typeof(youclean$where)
mode(youclean$where)
```

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

```
# Change to factor
youclean$where <- as.factor(youclean$where)
levels(youclean$where)
nlevels(youclean$where)
```

```
##  [1] "Argentina"  "Azerbaijan" "Belgium"    "Brazil"     "China"
##  [6] "Egypt"      "Georgia"    "Germany"    "Iran"       "Italy"
## [11] "Italy "     "Montenegro" "Pakistan"   "Russia"     "Spain"
## [16] "Togo"       "Turkey"     "Turkey "    "Venezuela"
## [1] 19
```

```
# After
class(youclean$where)
```

```
typeof(youclean$where)
mode(youclean$where)
```

```
## [1] "factor"
## [1] "integer"
## [1] "numeric"
```

Recycle the relevant part of this code to turn into factors also `prev.life`.

i. Lets finally save the result of our hard work in a R friendly file format (`.RData`). To this end, read the help files of the `save()` and `load()` functions and use the first one to save the two objects named `youclean` and `youraw` into a file called `you.RData`.
