Introduction to R: Day 2
Descriptive Statistics

Instructor: Yara Abu Awad

Workshop Schedule

- In this workshop, the following topics will be covered:
  - Mastering R basics: will include an introduction to the R environment, packages and data types
  - Describing data: will demonstrate how to generate descriptive statistics, table outputs and simple statistical tests (i.e. t tests)
  - Visualizing data: will show participants how to generate multiple types of plots and charts

But.. descriptive stats are boring

- Actually… it’s important to understand your data:
  - How many responses are missing in my variables of interest?
  - How are my variables coded? Do I need to recode them?
  - What do the distributions look like?
- Don’t make the mistake of skipping this, noticing something wrong in your results and then having to repeat your analysis all over again!
- Also: descriptive statistics are valuable in and of themselves and can yield very interesting information: e.g. prevalence of ADHD in my population of interest, % of people living in poverty etc…

Plan for today

- First 15 minutes will consist of a quick review
- Go to https://github.com/YaraRAA/data-scientifique
- Scroll down to: **README.md**
- You will see some poll links with numbers
- Let’s begin!

Plan for today ctd.

- Then….
- I will talk for an hour (if you’re lucky, maybe less).
- During this time ***resist*** the urge to open R and follow along with me.
- I will be giving you a lot of information and it will be easy to get lost (deep breaths).
- You will get a chance to practice afterwards with the exercises provided. Again, I recommend that you work in pairs.

Step 1: Import your data (rows & columns)

Source	File extension	Package	Function
Comma delimited Excel, simple text editor (e.g. notepad)	.csv	Base R	read.csv()
Simple text editor	.txt	Base R	read.table()

SPSS	.sav	foreign
		haven
			read.spss()
			read_spss()

SAS	.sas7bdat	haven	read_sas()
R	.rds	Base R	readRDS()
PostgreSQL, ArcGIS	.dbf	foreign	read.dbf()

Excel	.xlsx	readxl
		xlsx
			read_excel()
			read.xlsx()

	.xml	XML	

Imported data is different!

etc..

Step 1: Some words of advice

- Check your data to make sure that missing values, character variables and numeric variables were imported correctly
  - You can do this by: visually inspecting your data, the str() and summary() functions
- If you have leading zeros, import that variable as a character or R will assume it is numeric and delete those zeros
- I recommend converting your ID variable to character if it is numeric to avoid loss of important characters
- Trial and error!
- Google!
- Let’s look at read.csv

Step 1: Import .csv

Read.csv is the function you are using to import your data

```
df = read.csv('C:\\User\\Yara\\mydata.csv')
```

df is the name you are assigning to your dataset and it will be an object in R.
If you skip df = then R will import your data and print it in the console.

The path to your file and the full name of your file (note the quotation marks). You don't need the full path if you have already set your working directory to the folder where this file is stored.
Rstudio Menu: Session -> Set Working Directory -> Choose Directory

If you have set the working directory to the folder where the file is stored then this becomes

```
df = read.csv('mydata.csv')
```

Step 1: Troubleshooting

- Character variable accidentally imported as factor?

Note: you are replacing an existing column! Do this with care

$ tells R to look for a column inside the object df

df is the name you assigned your dataset and it is an object in R

```
df$variablename = as.character(df$variablename)
```

‘variablename’ is the name of your column / variable in df

this function converts data to character format

The data that you want to convert. In this example, it is the column named ‘variablename’

Note that in R data.frame objects: column names are in character format

Step 1: Troubleshooting

- Numeric variable accidentally imported as factor?

this function converts
data to numeric format

```
df$variablename = as.numeric(as.character(df$variablename))
```

Rule of thumb: the innermost function is executed first. So the column is converted to character and then to numeric. I recommend this for factor variables so there is no loss of information.

Why do we go to the trouble of converting data from one type to another?

Step 2: Recode variables as needed

- What does *recode* mean?
  - It means to transform a variable
- Correctly code missing as NA in R
  - Example 1: if all cells containing ‘777’ are actually missing in your data, you can set them to NA as follows (only do this if ‘777’ is not a valid value in *any* cell of your data)

```
df[df == ‘777’] <- NA
```

  - Example 2: if you want to code all “I don’t know” answers to a question as missing:

```
df$questionr = ifelse(df$question == “I don’t know”, NA, df$question)
```

Step 2: Recode variables as needed

- Categorize continuous variables
  - Example: this is done often in Nutritional epi for frequency of consumption

```
df$fruitsperday_cat = cut(df$fruitsperday, 4)
```

OR

```
df$fruitsperday_cat = cut(df$fruitsperday, breaks = c(0,0.5,2,5,9), labels =
c('none', 'Up to 2', 'three to five', 'six to nine' ), include.lowest = T)
```

Step 2: Recode variables as needed

- Change category names in a variable:
  - Example, turn string answers on a Likert scale (‘strongly agree’, ‘agree’, ‘disagree’, ‘strongly disagree’) to numbers (1,2,3,4)

```
df$question_numeric = ifelse(df$original_question == ‘strongly agree’, 1, NA)
df$question_numeric = ifelse(df$original_question == ‘agree’, 2, df$question_numeric)
df$question_numeric = ifelse(df$original_question == ‘disagree’, 3, df$question_numeric)
```

How would you complete this transformation?)

Step 2: Recode variables as needed

- Arithmetically transform a variable
  - Fahrenheit to Celsius

```
df$celsius = (df$fahrenheit – 32) * 5/9
```

  - Log transform

```
df$lvariable = log(df$variable)
```

  - Exponentiate

```
df$expvariable = exp(df$variable)
```

etc…

Step 3: Start Generating Descriptive Data!

- Continuous variables: mean, median, range, standard deviation
- Categorical variables: frequency tables
- T test: to compare two populations
- A picture is worth a thousand words but we will save plots for next time!

Continuous variables

- Functions: summary
  - Gives you the mean, median, min, max and number missing for continuous and factor variables
  - Can apply to entire data frame: `summary(df)`
  - Or an individual column: `summary(df$variable)`

- Other functions: mean, quantile, sd, min, max
  - Apply to individual columns:

```
mean(df$variable)
quantile(df$variable, c(0.05, 0.95))
```

  - Need special arguments if you have missing data!!

```
mean(df$variable, na.rm = T)
```

Continuous variables

- Want summary statistics by group?
  - The aggregate function. Example: height by sex

continuous variable

categorical variable

function

```
aggregate(height ~ sex, data = df, mean)
```

this is a formula,
e.g. a regression formula
y = a + bx
where ~ is equivalent to =

name of data frame
object in your workspace

- If data is missing:

```
aggregate(height ~ sex, data = df, function(x) mean(x,na.rm = T))
```

T test: comparing means of two groups

- The assumptions are:
  - that the data are quantitative and plausibly Normal
  - that the two samples come from distributions that may differ in their mean value, but not in the standard deviation
  - that the observations are independent of each other

H_0:	$\mu1=\mu2$
H_a:	$\mu1\neq\mu2$

Sources: https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/7-t-tests, https://www.itl.nist.gov/div898/handbook/eda/section3/eda353.htm

T test: checking that the variances are homogenous

- Back to our height and sex example
- Function: var.test

```
var.test(height ~ sex, data = df)
```

T test: comparing population means

- Function t.test:

```
t.test(height ~ sex, data = df)
```

- What if the variances are not equal? Let’s check the help file of the t test function

Categorical variables

- Functions: table
- Applies to one column: table(df$variable)
- Or more columns: table(df$variable1, df$variable2)
- To see number missing: table(df$variable, useNA = 'always')
- Functions of table objects:
  - If mytable = table(df$variable1, df$variable2)
  - ftable makes tables flat i.e. look nicer, also accepts formulas: ftable(mytable)
  - margin.table returns sums of rows and columns
  - prop.table gives percentages
  - summary of a table 'includes a Chi-square test for independence of all the factors if the table object has more than 1 dimension'