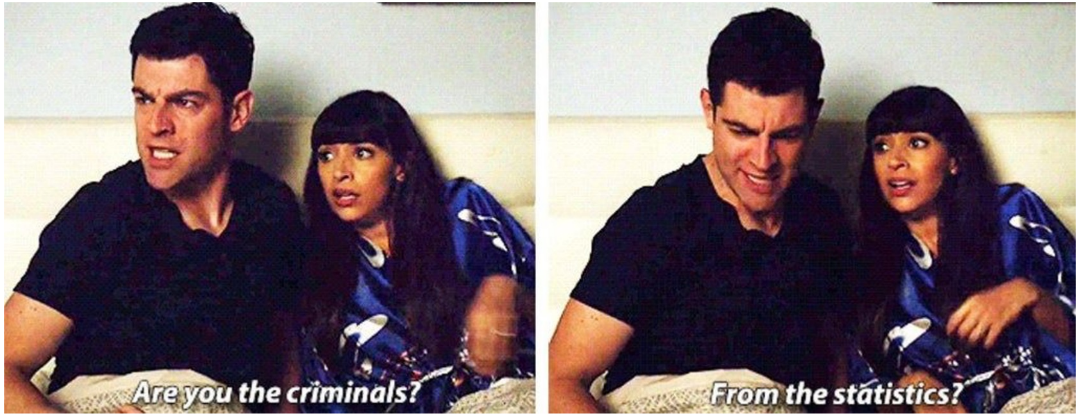
R review session

Stats I

November 2023

Agenda

1. Preparing your project
2. Preparing your data
3. Descriptive statistics
4. Inferential statistics (linear regressions)
5. Visualisation



1. Preparing your project

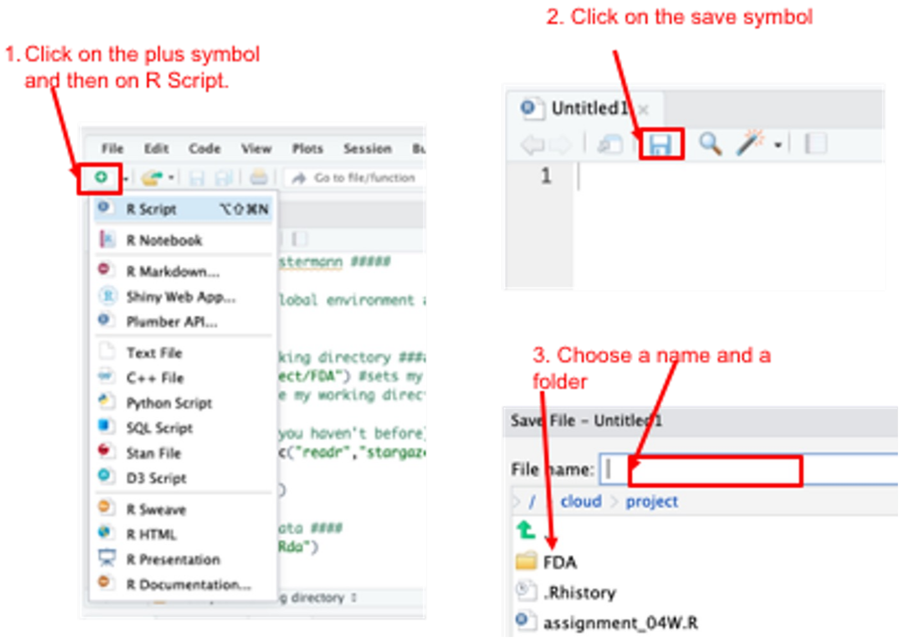
R scripts

- Scripts allow you to:
  - Write and correct your code
  - Document your work (for yourself and others)
  - Backtrack your work

- Use # to write comments (anything after a # is not treated as code in your script)

- In the FDA, you will be required to attach your script (as a pdf)

- Create, name and save a new script:



R scripts: Good practice

```r
#########################
# R review session
# Stats I
# November 2021
#########################

## 1. Clean your global environment
rm(list=ls())
options(scipen=999) # Remove scientific notation

## 2. Set your working directory
setwd("/Documents/Stats I/Review") # sets my working directory
getwd() # shows me my working directory

## 3. Install (if you haven't before) and load packages/libraries
install.packages(c("stargazer", "ggplot2", "dplyr", "readxl", "writexl", "haven"))
library(stargazer)
library(ggplot2)
library(dplyr)
library(readxl)
library(writexl)
library(haven)

## 4.  Load your data
```

- Title (within hashtags)
- Clean your environment
- Set your working directory
- Install and load packages (if any)
- Load your data

Working directory

- A folder on your computer that you attach to your R project

- Contains any file you load into R (e.g. a data set), and any file you save from R (e.g. a regression table)

- You can check your current working directory with the function getwd()

- You can change your working directory with the function setwd(“...”)
  → Inside the brackets, write the file path that leads to your desired folder

Datasets: open and save different formats

Type	From	Command in R	Package

.csv	Excel, Stata,...	`mydata <- read.csv("mydata.csv") #open dataset`
		
		`write.csv(mydata, "mydata.csv") #save dataset`
			Base R
			Base R

.xlsx	Excel	`mydata <- read_xlsx("mydata.xlsx") #open dataset`
		`mydata <- read.xlsx("mydata.xlsx", sheetName = "mysheet")`
		`#open data set from specific sheet`
		
		`write_xlsx(mydata, "mydata.xlsx") #save dataset`
			library(readxl)
			library(writexl)

.dta	Stata	`mydata <-  read_dta("mydata.dta") #open dataset`
		
		`write_dta(df, "c:/mydata.dta") #save dataset`
			library(haven)

.Rda	R	`load("dataset_asia.Rda") #open dataset`
		
		`save(asia_full, file = "dataset_asia.Rda") #save dataset`
			Base R
			Note: here you don't need to assign a name to your dataframe

Now go to the script (Section 1)

2. Preparing your data

Objects

- Most general concept/term
- Definition:
  - An object is a defined “element” in R
  - It’s a “vessel”, i.e. something that contains something else
- How to create an object:
  - You create an object by deciding on a name and using an assignment operator: `=`, `<-`, or `->`
  - Examples:
    - `max <- 10`
    - `gender <- “female”`
- Why use objects?
  - They’re handy, practical and you can use them over and over again → they are saved in the Environment
  - Example: `reg <- lm(prestige ~ education + income + women, data = Prestige)`

Data types and structures: R data types

R has 4 basic data types:

1. Character: `“a”`, `“swc”`
2. Numeric: `2`, `15.5`
3. Integer: `2L` (the L tells R to store this as an integer)
4. Logical: `TRUE`, `FALSE`

R has many data structures, which include:

- Vectors (character, logical, integer, or numeric)
- Matrix
- Data frames
- Factors

	Linnear	**Rectangular**
All same type	vector	matrix
Mixed	list	data frame

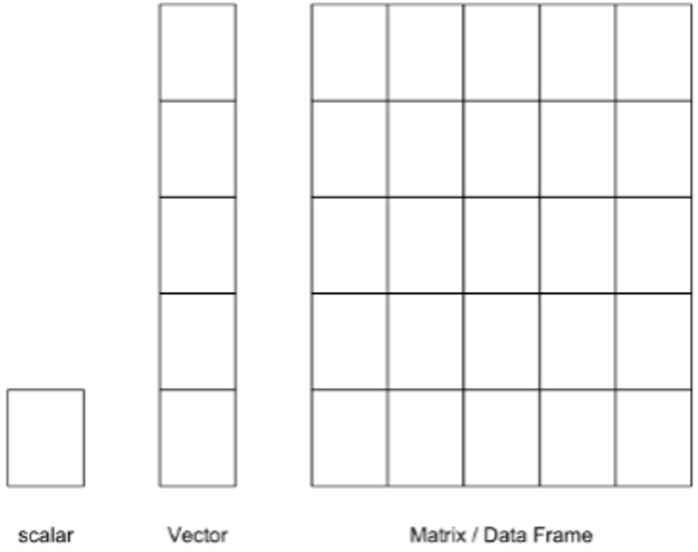
Data types and structures: Vectors

Decide on a name and use the assignment operator (“<-”)

- `x <- c(10.4, 5.6, 3.1, 6.4, 21.7)`

Different ways of creating vectors:

Command in R	**Gives you**
`a <- c(2,5,8)`	2 5 8
`b <- 2:8`	2 3 4 5 6 7 8
`c <- seq(2,4, by=0.5)`	2.0 2.5 3.0 3.5 4.0
`d <- rep(1:2, times=4)`	1 2 1 2 1 2 1 2
`e <- rep(1:2, each=3)`	1 1 1 2 2 2



Source: Phillip (2018)

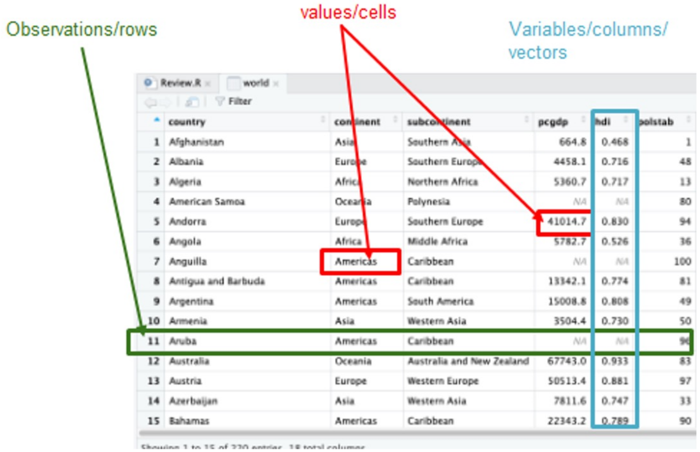
Data types and structures: Factors

- Factors are used to store categorical data
- Can be unordered (when the data is categorical/nominal) or ordered (when the data is ordinal)
- Factors are stored as integers (1,2,3 …) , and have labels associated with these unique integers.
    - We will see an example in R

Data types and structures: Dataframes

- A dataframe is a “rectangular” type of object, where observations are in the rows and variables in the columns
- All columns have the same length
- We usually open a dataset/dataframe that we download from official websites, and we also sometimes merge different datasets given to us
- R commands that gives us a first look at our data:

```
View(world)      # Open the data in a new window
head(world)      # Show me the first few rows
str(world)       # Show me the structure of the data
names(world)     # What are the names of the columns?
nrow(world)      # How many rows are there in the data?
```



Dealing with missing values (NAs)

- How to find them: is.na() gives back missing values in vectors or dataframes
- How to remove them: na.omit()

However…

It’s a bad idea to drop all the NAs in a dataframe! Just delete the NAs in the variables of interest

keeps all values that are NOT missing

```
world_clean <- world[!is.na(world$growth),]
```

name of the dataframe (with NAs)

name of the variable you want to clean

Remember that the order within the brackets is always [Rows, Columns]

Now go to the script (Section 2.1-2.2)

Recategorizing data

You can transform interval level variables into ordinal/nominal level variables using the cut() function

```
world_clean$polstab_cat <- cut(world_clean$polstab,
                               breaks = c(0, 35, 65, 100),
                               labels = c("Very unstable", "Unstable", "Stable"), # assigns labels
                               ordered_result = TRUE) # to get ordinal variable rather than nominal
```

This creates the following categories (the variable is an index from 0 to 100):

- Category 1: from 0 to 35
- Category 2: from 35 to 65
- Category 3: from 65 to 100

Subsetting data: Extracting elements from a vector

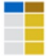
Selecting by…	Code in R	What it extract…
By position	`x[4]`	The fourth element
	`x[-4]`	All except the fourth element
	`x[2:4]`	Elements two to four
	`x[-(2:4)]`	All the elements except two to four
	`x[c(1, 5)]`	Elements one AND five
By value	`x[x == 10]`	Elements which are equal to 10
	`x[x < 0]`	All elements less than zero
By name	`x[“apple”]`	Element with the name “apple”

Subsetting data: Extracting elements from a dataframe

The general code to extract an element from a dataframe is

`nameofdedataframe[row number, column number]`

`df[ , 2]`



`df[2, ]`

`df[2, 2]`

To extract a variable or column from a dataframe:

`nameofthedataframe$nameofthevariable`

`df$x`

Subsetting data: Extracting elements from a dataframe

Applied example of subsetting rows and columns:

Let's imagine the following dataset called `my_df`:

	a	b	c	d
1	0.3146011	10.575473	a	A
2	0.7404668	8.818245	b	B
3	0.4659729	10.017466	c	A
4	0.8692551	9.997965	d	B
5	0.0873134	11.046969	e	A
6	0.9267281	11.315129	f	B
7	0.6326854	10.903308	g	A
8	0.5868624	11.001529	h	B
9	0.6586531	10.157911	i	A
10	0.5244315	10.683044	j	B

- `my_df[1:3]` (with no comma) will subset `my_df`, returning the first three columns of the data frame
- `my_df[1:3, ]` (with comma, numbers to the left of the comma) will subset `my_df` and return the first three ROWS as a data frame
- `my_df[, 1:3]` (with comma, numbers to the right of the comma) will subset `my_df` and return the first three COLUMNS as a data frame

Now go to the script (Section 2.3)

3. Descriptive statistics

Ways to describe your data

Measure	Type of data	R function
Mean	Interval	mean()
Median	Interval, ordinal	median()
Mode	Ordinal, nominal, (interval)	table()
Standard deviation	Interval	sd()
Variance	Interval	var()
Range, interquartile range	Interval	range(), IQR()

OR use the summary() function to describe interval data quickly

Tables

- Present your categorical data (ordinal and nominal) with the table() function
- In tables with one variable, R counts the number of observations in each category
- In tables with two variables, R counts the number of observations in each combination of categories:

	Africa	Americas	Asia	Europe	Oceania
Non-Democratic	38	7	18	1	1
Democratic	6	16	6	24	3

- If you instead use the prop.table() function, R will convert the counts into proportions
  - By row: argument = 1
  - By column: argument = 2

Distributions

Use hist(), density(), and plot() to show how your data is distributed:

- hist() creates histograms for a single interval variable
- plot(density()) creates a smoothed version of the histogram
  - This is usually preferred, since the break points of a histogram can be tweaked to potentially misrepresent the data
  - Density plots always look the same

- plot() shows the relationship between two variables:
  - Interval + interval → **Scatterplot**
  - Ordinal/nominal + internal → **Boxplot**

Now go to the script (Section 3)

3. Inferential statistics (linear regressions)

Correlations and linear regressions

Correlation

- You can check the correlation between two variables by using the cor() function
- It is always a good idea to visualize the association between variables with the plot() command too.

Linear Regressions in R

- Create an object that contains the output of a linear regression.
- Here, we create a bivariate regression with “corrupt” (level of corruption) as the dependent variable and “polstab” (political stability) as the independent variable:

```
bi_reg <- lm(corrupt ~ polstab, data = world.clean)
```

- ...or define a multivariate regression (don’t forget the “+”)...

```
mul_reg <- lm(corrupt ~ polstab + homicide, data = world.clean)
```

- …or a multivariate regression with interaction effects

```
interaction_reg <- lm(corrupt ~ polstab + demo + polstab*demo, data = world.clean)
```

Linear regressions: output

●...use summary() to get your regression output

summary(bi_reg)

```
Call:
lm(formula = corrupt ~ polstab, data = world.clean)

Residuals:
    Min      1Q  Median      3Q     Max 
-48.439 -12.867   0.714  12.243  57.657 

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)    
(Intercept)  7.12157    2.86114   2.489              0.0139 *  
polstab      0.82634    0.05086  16.247 <0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 17.62 on 150 degrees of freedom
Multiple R-squared:  0.6377,	Adjusted R-squared:  0.6352 
F-statistic:   264 on 1 and 150 DF,  p-value: < 0.00000000000000022
```

Regression tables

- We use stargazer() from the stargazer package to create a better-looking table (that you can use in your papers)
- Stargazer can - very usefully - display different regression tables side by side

```
library(stargazer)
stargazer(bi_reg, mul_reg, title = "Regression results", type=“text”)
```

- We can use the ‘out’ argument in the stargazer function to save the regression output in a file (e.g. .txt or .docx)

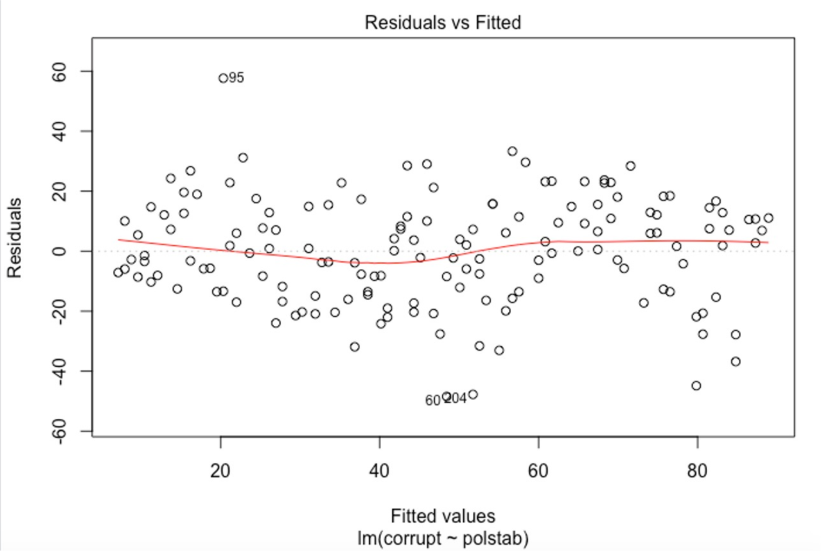
```
stargazer(bi_reg, mul_reg, title = "Regression results", type=”text”, out=“mul_reg.txt”)
```

Regression diagnostics

- R creates regression diagnostics with the plot() function.
  - The first argument is the name of your linear regression model
  - The second argument is a number specifying which kind of diagnostic you are interested in.

- Regression diagnostics can help us check some of the OLS assumptions

plot(bi_reg,1): “Residuals vs. Fitted”: Does it appear that my variables have a linear relationship?



Now go to the script (Section 4)

5. Visualisation

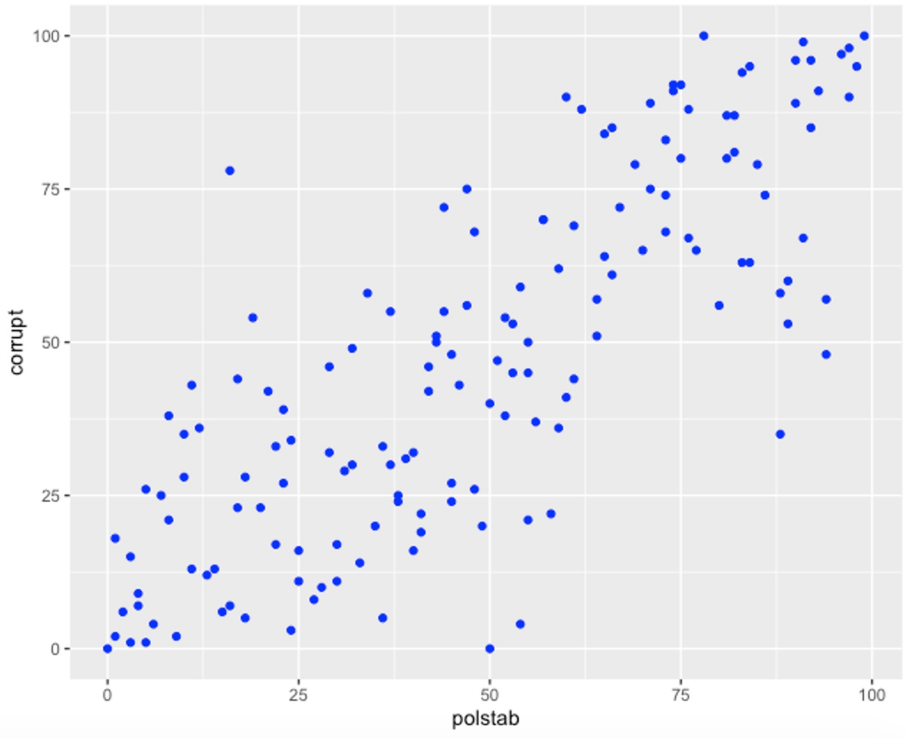
Graphs (I)

- We can plot our data either with the basic plot() function, or by using the ggplot2 library

- Basic idea of ggplot: you put layers upon layers of what you want to plot:

```
ggplot(world.clean, aes(x = polstab, y = corrupt)) +
        geom_point(color= "blue")
```

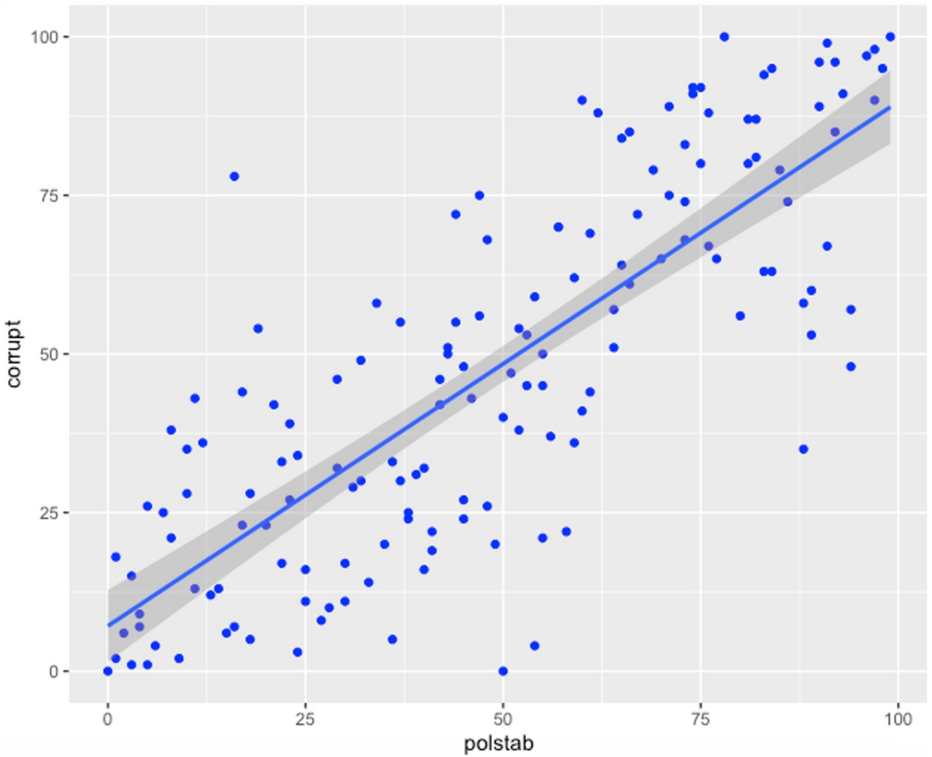
- The “+” is very important!



Graphs (II)

- New layer: Regression line

```
ggplot(world.clean, aes(x = polstab, y = corrupt)) +
        geom_point(color= “blue”) +
        geom_smooth(method = “lm”)
```



Graphs (III)

- New layers: Theme and labels

```
ggplot(world.clean, aes(x = polstab, y = corrupt)) +
        geom_point(color="blue") +
        geom_smooth(method = "lm") +
        theme_minimal() +
        labs(x="Political Stability",
                y="Level of Corruption",
                title = "Effect of Political Stability
        on Corruption")
```