Political Analysis: Lab 1

Hilary Term 2016

1 Typographical conventions

- Text in `monospace` font usually represents `R` code.
- Text in grey boxes represents the executed `R` code together with its corresponding output.

2 R programming

Running R

- Enter `R` code directly into the command line or save it to a script and send it to `R`.
- A command line is an interface where you type a command and press the `Return` key to execute that command.
- Once you close `R` all the commands that you typed in the command line will be lost.
- A script is a textfile where you save the sequence of the commands that you want to execute together with comments to describe what you want to do.
- Writing scripts has two benefits: they help you remember what you did, they allow other people to replicate your work, and you can usually re-use old scripts in new projects, which saves times.
- To send code from the script to the console, select the line of code in `RStudio` and click on `Run`.
- Separate commands either by a `;` or by a new line.
- `R` is case sensitive.
- The `#` character at the beginning of a line signifies a comment, which is not executed.
- Access help files for `R` functions by preceding the name of the function with `?` (e.g., `?table`).

Working with objects

- R stores both data and output from data analysis in objects.
- Assign data or output to objects using the `<-` or `=` operator.
- See a list of all objects in the current session by typing `ls()`.

Exercise

- List all objects in current session:

```
ls()

## character(0)
```

- Assign the number 1 to object called `a`:

```
a <- 1
```

- List all objects of the current session:

```
ls()

## [1] "a"
```

- See the content of object `a`:

```
a

## [1] 1
```

3 Working with data

Data set files

- R can read pretty much any data format (e.g., Excel, Stata).
- Ljiphart's datasets is saved as a text file, where the values are separated by commas (CSV).
- The first five rows of one of the text file look like this (do not execute):

```
"country","exec_parties_1945_2010","exec_parties_1981_2010","federal_unitary_194
"ARG",-0.93,-1.01,1.38,1.34,3.15,3.15,82.4,82.4,8,8,17.98,17.98,2.7,2.7,4.5,4.5,
"AUL",-0.73,-0.65,1.63,1.58,2.22,2.19,80.7,86.5,9.1,7.37,9.44,10.07,2.12,1.88,5,
"AUT",0.43,0.64,1.07,0.97,2.68,3.23,43.3,47.4,8.07,5.9,2.51,2.02,0.38,0.38,4.5,4
"BAH",-1.5,-1.33,-0.15,-0.18,1.69,1.74,100,100,9.44,7.37,16.48,15.9,3,3,1,1,2,2,
```

Reading in data

- The function `read.csv()` can read the text file.
- Although we retrieve the text file from the internet, the function also works for files saved on the harddrive.
- We assign the data set to an object.

```
Gov <- read.csv("http://andy.egge.rs/data/L.csv")
```

Viewing data

- Usually it is best to first look at parts of the data set (output omitted):

```
head(Gov)  # first six rows
tail(Gov)  # last six rows
names(Gov) # variable names
str(Gov)   # structure of the data set
```

- We can also look at the whole data set by typing (output omitted):

```
Gov
```

Data frames

- A data set like this one is usually saved as a data frame.
- A data frame is a particular type of object that has a matrix structure.
- The rows list the numerical or categorical observations, the columns list the variables.
- We can access rows, columns, and cells by indexing the object.
- A common notation is: `object[row,column]`.
- For instance, to access the name of the first country we can type:

```
Gov[1,1]

## [1] ARG
## 36 Levels: ARG AUL AUT BAH BAR BEL BOT CAN CR DEN ... US
```

- To access all variable values of the first five countries, we could type (output omitted):

```
Gov[1:5,]
```

- To access the first two variable values of the first country, we could type:

```
Gov[1,1:2]

##   country exec_parties_1945_2010
## 1     ARG                  -0.93
```

Variable indexing

- We can also index variables directly by using their names, either with `object[,"variable"]` or `object$variable` notation.
- Get all the country names:

```
Gov$country

##  [1] ARG AUL AUT BAH BAR BEL BOT CAN CR  DEN FIN FRA GER GRE
## [15] ICE IND IRE ISR ITA JAM JPN KOR LUX MAL MAU NET NOR NZ
## [29] POR SPA SWE SWI TRI UK  URU US
## 36 Levels: ARG AUL AUT BAH BAR BEL BOT CAN CR DEN ... US

Gov[,"country"]

##  [1] ARG AUL AUT BAH BAR BEL BOT CAN CR  DEN FIN FRA GER GRE
## [15] ICE IND IRE ISR ITA JAM JPN KOR LUX MAL MAU NET NOR NZ
## [29] POR SPA SWE SWI TRI UK  URU US
## 36 Levels: ARG AUL AUT BAH BAR BEL BOT CAN CR DEN ... US
```

- The advantage of the `object[,"variable"]` notation is that we can select several variable at once by creating a vector of variable names.

- The function `c()` combines values to a vector.
- See for every country its effective number of parliamentary parties between 1981 and 2010 (output omitted):

```
Gov[,c("country","eff_num_parl_parties_1981_2010")]
```

Variable names

- If there were no variable names or we wanted to change them, we could use `names()`.
- To change the name of the first variable from `country` to `CountryName` we type:

```
names(Gov)[1] <- "CountryName"
names(Gov)[1]

## [1] "CountryName"
```

Variable values

- To change the factor labels of the country variable, use the `levels()` command.
- For instance, to change the country name `ARG` to `Argentina` and `AUL` to `Australia`, use the `levels()` command and indexing:

```
levels(Gov$CountryName)[1:2] <- c("Argentina", "Australia")
```

Saving data

- We can save our changed data set in a number of formats (e.g., Stata and Excel).
- For convenience, we save it again as a CSV file.
- The command below is commented so it will not run.

```
# write.csv(Gov, file = "path/filename.csv")
```

Saving the script

- We can save our script to keep a record of what we did.
- To do so, in `RStudio` click on `File > Save As...`, then type a filename such as `Lab1.R`, and press the `Return` key.