A reproducible analysis in R

Stefano De Sabbata

This document is an small example of a reproducible analysis. This is an R Markdown script created in RStudio, which uses the *rmarkdown*, *knitr*, and *dyplr* libraries of the Tidyverse. Data from the UK Department for Transport are analysed and the output is compiled in a *pdf* document.

Data

This analysis document depends on *Gather_DfT_data_2015.R*, which retrieves some data from the Department for Transport and saves them in the *Data* folder. It is assumed that *Gather_DfT_data_2015.R* has been executed before this document is compiled.

The data file is relatively large and it is available in compressed *zip* format. It is thus necessary to download the file and process it locally before reading it in to R. The *download.file* and *unzip* functions can be used to read in data: *download.file* takes two arguments, the URL of the file to download, and the name of a local file into which it will be stored. The function *unzip* takes at minimum one argument, which is the file to be unzipped. It is possible to specify the folder in which the files should be extracted to (see *?unzip*). In this case, once the function has been executed, the files contained in the zip file are available in the working *Data* folder.

The *Gather_DfT_data_2015.R* script is executed by the *Make.R* script. For your own understanding of the process, read the code in *Gather_DfT_data_2015.R* and then execute it by clicking on the *Source* button on the top-right of the code when the file is open on RStudio.

Libraries and Packages

As mentioned in earlier lectures, libraries are collections of functions. Libraries can be installed in R using the function *install.packages* or in RStudio via the *Tools > Install Packages* menu. This document assumes that the whole Tidyverse is installed, that includes a number of different libraries, including *knitr*. The *Make.R* script includes the instruction to install the Tidyverse if not currently installed. You can check whether the Tidyverse is installed by clicking on the *Packages* tab in the bottom-right pane in RStudio, and search for "tidyverse". If not installed, either install it manually or run the *Make.R* script.

You can load a library using the function *library*, as shown below. Once a library is installed on a computer you don't need to install it again, but every script needs to load all the library that it uses. Once a library is loaded all its functions can be used. Before staring, let's then load the *knitr* library that we need to use *kable* to create nice tables in R Markdown.

```
# Load knitr
library(knitr)
```

Loading the data

R can import a number of different types of data files. One of the most commonly used format is the *.csv* (comma separated values). Before continuing, make sure the *Data* folder contains the *Accidents_2015.csv* file. If not, you can execute the *Gather_DfT_data_2015.R* script in the *Dat* folder, or the *Make.R* script.

You can open the *Accidents_2015.csv* file using Notepad (or another text editor) to see its text format, and using Microsoft Excel (or similar software) to see the tabular structure. The code below loads the data from the file using the *read.csv* function into a *data.frame* variable named *accidents*. The function *head* is used to grab only the top rows to be displayed in this document. In this case, the code specifies to grab only the first five rows ($n = 5$). Those five row are the displayed using the *kable* function mentioned above, which create a well-formatted table.

Note that when this document is compiled, the *working directory* is the folder where the document is. Thus, the path to the *Data* files needs to start from within the *Analysis* folder, up one level to the main folder (using *../*) and then in the *Data* folder.

```
# Load data about accidents
accidents <- read.csv("../Data/Accidents_2015.csv")

# Inspect the content in data frame
# Only a subset of columns is selected
kable(head(
  accidents[,
    c("Accident_Index", "Date", "Time", "Local_Authority_.District.")
  ],
  n = 5
))
```

Accident_Index	Date	Time	Local_Authority_.District.
201501BS70001	12/01/2015	18:45	12
201501BS70002	12/01/2015	07:50	12
201501BS70004	12/01/2015	18:08	12
201501BS70005	13/01/2015	07:40	12
201501BS70008	09/01/2015	07:30	12

A simple analysis

The analysis below uses *dplyr*, which is another useful library that is part of the Tidyverse. The functions provided by *dplyr* are frequently used in scripts that employ a pipe-based programming style. This is not covered in this lecture, but *Gather_Ofcom_data_2012.R* provides an example of that approach. The chunk of code below loads the *dyplr* library.

```
# The warn.conflicts and quietly options can be used
# to suppress the message on loading
library(dplyr, warn.conflicts = FALSE, quietly = TRUE)
```

The function *count* of the *dplyr* library can be used to count the number rows of a *data.frame*, grouped by a variable (in the example below, the local authority district). As you can see, the code below is formatted in a way similar to a code block, although it is not a code block. This is very common in R programming (especially when functions have a lot of parameters) as it makes the code more readable.

```
# Count number of accident by local authority
accidents_count <- count(
  accidents,
  Local_Authority_.District.
  )

# Inspect the content in data frame
kable(head(accidents_count, n=5))
```

Local_Authority_.District.	n
1	1578
2	936
3	844
4	857
5	1066

The function *summarise* of the same library *dplyr* can be used in combination with the function *group_by* to summarise the values of the rows of a *data.frame*. Rows can be grouped by a variable (in the example below, whether an area is urban or rural), and then aggregated based on a function over one or more columns. In this example the function is *mean*, over the column *Accident_Severity* (where numeric values range from 1, for the most sever, to 3, for the less sever accidents).

```
# Group accidents by Urban and Rural areas
accidents_urb_rrl <-group_by(
    accidents,
    Urban_or_Rural_Area
  )

# Average severity per Urban and Rural areas
accidents_urb_rrl_severity <- summarise(
  accidents_urb_rrl,
  avg_severity = mean(Accident_Severity)
)

# Inspect the content in data frame
kable(accidents_urb_rrl_severity)
```

Urban_or_Rural_Area	avg_severity
1	2.861045
2	2.781662

The same result can be achieved by inserting the call to the function *group_by* directly into the call to the function *summarise*, as first argument. Observe the code below and compare to the code above.

```
# Group and average severity per Urban and Rural areas
accidents_urb_rrl_severity <- summarise(
  group_by(
    accidents,
    Urban_or_Rural_Area
  ),
  avg_severity = mean(Accident_Severity)
)

# Inspect the content in data frame
kable(accidents_urb_rrl_severity)
```

Urban_or_Rural_Area	avg_severity
1	2.861045
2	2.781662

The same result can also be achieved by using the *aggregate* function of the *stats* library, which does not need to be loaded as it is part of the base R libraries. Observe the code below and compare to the two chunks of code above.

Note that in the code below, a copy of the *data.frame* containing only the *Urban_or_Rural_Area* and *Accident_Severity* column is first created, as the aggregation function specified as argument for *FUN* is applied to all columns in the table except the group-by column specified as argument for *by*.

```
# Average severity per Urban and Rural areas
accidents_copy <- accidents[, c("Urban_or_Rural_Area", "Accident_Severity")]
accidents_urb_rrl_severity <- aggregate(
  accidents_copy,
  by = list(accidents_copy$Urban_or_Rural_Area),
  FUN = mean
)

# Inspect the content in data frame
kable(accidents_urb_rrl_severity)
```

Group.1	Urban_or_Rural_Area	Accident_Severity
1	1	2.861045
2	2	2.781662

Mergind datasets

As mentioned in previous lectures, two tables can be merged using a common column of identifiers. We can thus load a second dataset, containing the information about the vehicles involved in the accidents. In this case, one or more vehicles will be involved for each accidents.

```
# Load data about vehicles
vehicles <- read.csv("../Data/Vehicles_2015.csv")

# Inspect the content in data frame
# Only a subset of columns is selected
kable(head(
  vehicles[,
    c("Accident_Index", "Vehicle_Type", "Age_of_Vehicle")
  ],
  n = 5
))
```

Accident_Index	Vehicle_Type	Age_of_Vehicle
201506E098757	9	11
201506E098766	9	1
201506E098766	9	-1
201506E098777	20	1
201506E098780	9	-1

The two tables can be merged using the function *merge*, and specify the common column (or columns, if more than one is to be used as identifier) as argument of *by*.

```r
# Merge the data
accidents_vehicles <- merge(accidents, vehicles, by = "Accident_Index")

# Inspect the content in data frame
# Only a subset of columns is selected
kable(head(
  accidents_vehicles[,
    c("Accident_Index", "Urban_or_Rural_Area", "Accident_Severity", "Vehicle_Type", "Age_of_Vehicle")
  ],
  n = 5
))
```

Accident_Index	Urban_or_Rural_Area	Accident_Severity	Vehicle_Type	Age_of_Vehicle
201501BS70001	1	3	19	4
201501BS70002	1	3	9	3
201501BS70004	1	3	9	10
201501BS70005	1	3	9	-1
201501BS70008	1	2	1	-1

The average and standard deviation age of the vehicles involved in an accident per accident severity can then be calculated using the code below. Note how the second aggregation function *sd* (standard deviation) is added after the first aggregation function, to create a second column of aggregated values.

```r
# Group and average severity per Urban and Rural areas
accidents_vehicles_severity_age <- summarise(
  group_by(
    accidents_vehicles,
    Accident_Severity
  ),
  avg_age = mean(Age_of_Vehicle),
  stddev_age = sd(Age_of_Vehicle)
)

# Inspect the content in data frame
kable(accidents_vehicles_severity_age)
```

Accident_Severity	avg_age	stddev_age
1	5.929286	6.227915
2	5.123257	6.280692
3	5.148115	5.815070

Finally, images can be included in *RMarkdown* documents by simply adding the corresponding code. The boxplot can be created using the *boxplot* function of the base library *graphics*. The first arguments requires the name of the variable to be shown on the y-axis and the name of the variable to be used to categorise the cases on the x-axis, separate by a *tilde* symbol. The data to be used in specified using the *data* parameter. The example below also illustrate (in the RMarkdown code) how to specify the caption and size of the figure using the arguments of the code chunk.

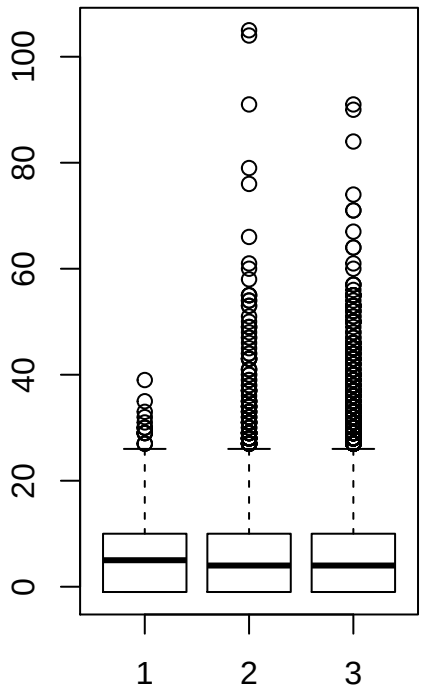

Figure 1: Age of vehicles per accident severity

```
boxplot(Age_of_Vehicle ~ Accident_Severity, data = accidents_vehicles)
```

Exercise

Add another section of R code below, that creates a new column for the *accidents* dataset, which represents the hour of the accident – e.g., from *18:45* to *18*. That can be achieved using the function *substr* (to select a section of a character variable, see *?substr*) on the column *Time* and the converting the result *as.numeric*.

Once the first part above is completed, calculate the number of accidents per hour of the day in which the accident happened, and then the average severity per hour of the day. Then, merge the *accidents* and *vehicles* again, and calculate the average age of the driver per hour of the day.

Solutions are provided in the script *Reproducible_analysis_Exercise_solution.R* in the *Analysis* folder.