Lecture X: Outliers and Missing Data Imputation

Amaury de Vicq

5/16/2020

Introduction

In this lecture, we will occupy ourselves with (i) identifying outliers and (ii) missing value imputation (NA).

This is a crucial step in the data cleaning process and is often necessary for any kind of succesful data analysis.

This is because (i) missing data might prevent functions to work properly, and (ii) outliers could corrupt and/or bias to your results.

Part 1: Missing Data Handling

Basic guidelines

Raw, unclean data often has many missing data (NAs). Especially in economic history, this is very common and could be due to a variety of different reasons.

It might be that some of the sources you are using to create your dataset has several unreadable entries, that you cannot access the sources and/or that some years are missing.

While it might be possible to simply overlook these missing observations or delete them for your dataset, this is not always the best approach.

Basic guidelines [2]

In fact it is important to distinguish between three categories of missing data:

- MCAR (missing completely at random)
- MAR (missing at random)
- MNAR (missing not at random)

Basic guidelines [3]

These categories determine how to best solve the missing data issues. Some basic guidelines include:

- If the amount of NAs $< 5\%$ and they are MCAR, it generally acceptable to simply delete them and/or to use functions with a built-in NA handling feature
- If the amount of NAs $> 5\%$ and they are MCAR, it is better to use simple imputation methods such as mean imputation and interpolation
- If they are MAR or MNAR it is often advised to rely on more complex imputation methods such as MICE (multiple imputation by chained equations)

Simple methods for missing data treatment

Uncovering missing data

Let us import some historical data using the `readxl` package, which we introduced in a previous lecture.

As we will soon find out, this datasets contains many missing values (NAs). We will then discuss some simple methods to address this issue.

First, let us install the `readxl` package.

```r
install.packages('readxl')
```

And don't forget to 'turn it on':

```r
library(readxl)
```

Reading the data

Now, let us import the dataset using `read_excel` command:

```
data <- "data/Netherlands_GDPperCapita_TerritorialRef_1946_2012_CCode_528(1).xlsx"
NL <- read_excel(data)
```

The ‘NL’ dataset provides historical information on GDP per Capita for the Netherlands (1800-2010).

It is derived direcly from the Clio-infra website, with some minor alterations for the purpose of this lecture.

Before we go any further, let us have a closer look at the actual dataset. By doing so, we might uncover some common issues that could arise in your own projects as well.

Inspecting the data

Recommended first steps in the analysis of all sorts of datasets are:

- the `str` function, which tells us how the dataeset is structured.
- the `summary` function, which provides some summary statistics regarding the dataset.
- and finally the `head` function, which (if set as default) provides an overview of the first 5 rows of the dataset.

```
str(NL)
```

```
## tibble [211 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Country Code: num [1:211] 528 528 528 528 528 528 528 528 528 528 ...
##  $ Country Name: chr [1:211] "Netherlands" "Netherlands" "Netherlands" "Netherlands" ...
##  $ Indicator   : chr [1:211] "GDP per Capita" "GDP per Capita" "GDP per Capita" "GDP per Capita" ...
##  $ Year        : num [1:211] 1800 1801 1802 1803 1804 ...
##  $ Data        : chr [1:211] "2609" "2563" "2704" "2646" ...
```

Inspecting the data [2]

```
summary(NL)
```

```
##   Country Code Country Name        Indicator             Year     
##  Min.   :528   Length:211         Length:211         Min.   :1800  
##  1st Qu.:528   Class :character   Class :character   1st Qu.:1852  
##  Median :528   Mode  :character   Mode  :character   Median :1905  
##  Mean   :528                                         Mean   :1905  
##  3rd Qu.:528                                         3rd Qu.:1958  
##  Max.   :528                                         Max.   :2010  
##      Data          
##  Length:211        
##  Class :character  
##  Mode  :character  
##                    
##                    
##                    
```

```
head(NL)
```

```
## # A tibble: 6 x 5
##   `Country Code` `Country Name` Indicator       Year Data 
##            <dbl> <chr>          <chr>          <dbl> <chr>
## 1            528 Netherlands    GDP per Capita  1800 2609 
## 2            528 Netherlands    GDP per Capita  1801 2563 
## 3            528 Netherlands    GDP per Capita  1802 2704 
## 4            528 Netherlands    GDP per Capita  1803 2646 
## 5            528 Netherlands    GDP per Capita  1804 2772 
## 6            528 Netherlands    GDP per Capita  1805 2609 
```

Inspecting the data [3]

Using these functions, gives us detailed and necessary information about this dataset. We find that it consists of 211 observations accross 5 different variables.

`Country Code`, `Country name` and `Indicator` are somewhat redudant as we already know that this dataset contains information on ‘GDP per Capita’ for the Netherlands, so we could remove them.

More imporantly however, R recognises the variable ‘Data’ as a character instead of number or an integer, so we have to change this.

Inspecting the data [4]

Using the following code, we will change ‘Data’ into a numeric variable, and drop column 1:3 (albeit this is optional!). We will save the new dataset as `NLclean`, so we can easily revert back to the old dataset when we chose to do so. We will then once more use the ‘summary’ function on the newly created NLclean dataset.

```
library(dplyr)
NLclean <- NL %>%
            mutate(data = as.numeric(Data)) %>%
            select(4,6)

str(NLclean)
```

```
## tibble [211 x 2] (S3: tbl_df/tbl/data.frame)
##  $ Year: num [1:211] 1800 1801 1802 1803 1804 ...
##  $ data: num [1:211] 2609 2563 2704 2646 2772 ...
```

```
summary(NLclean)
```

```
##       Year           data
##  Min.   :1800   Min.   :  1640
##  1st Qu.:1852   1st Qu.:  2566
##  Median :1905   Median :  3531
##  Mean   :1905   Mean   :  8140
##  3rd Qu.:1958   3rd Qu.:  8287
##  Max.   :2010   Max.   :166955
##                 NA's   :14
```

Inspecting the data [5]

Success! We now have much easier to interpret dataset which shows the GDP per Capita for the Netherlands on a yearly basis for the period 1800-2010.

More importantly, it tells us that there are 14 missing values (NAs) in the ‘Data’ variable.

In other words, for 14 years out of the 211, we do not have any information regarding the GDP per Capita. Uncovering these NAs is a crucial step.

Deleting data, relying on built-in NA handling features

As we briefly mentioned, we can opt to simply ignore these 14 NAs, or we can simply dete them using the ‘na.omit’ function

```
NLomit <- na.omit(NLclean)

summary(NLomit)
```

```
##       Year           data       
##  Min.   :1800   Min.   :  1640  
##  1st Qu.:1857   1st Qu.:  2566  
##  Median :1906   Median :  3531  
##  Mean   :1907   Mean   :  8140  
##  3rd Qu.:1958   3rd Qu.:  8287  
##  Max.   :2010   Max.   :166955  
```

As you can see, the dataset now consists of only 197 observations, and no longer has any NAs.

Handling NA's

In additon many functions in R have a built-in NA handling feature designed specificially for this purpose.

It is important to note that per default this function is often set to ignore the missing data.

The function ‘boxplot’ for example has an argument ‘na.action’ which by default is set to NULL, indicating that it will ignore NAs.

Handling NA's [2]

If you feel confident you can ignore NA (i.e. because they are $< 5\%$ of all observations and MCAR) you can always check the documentation of any given function using '?'

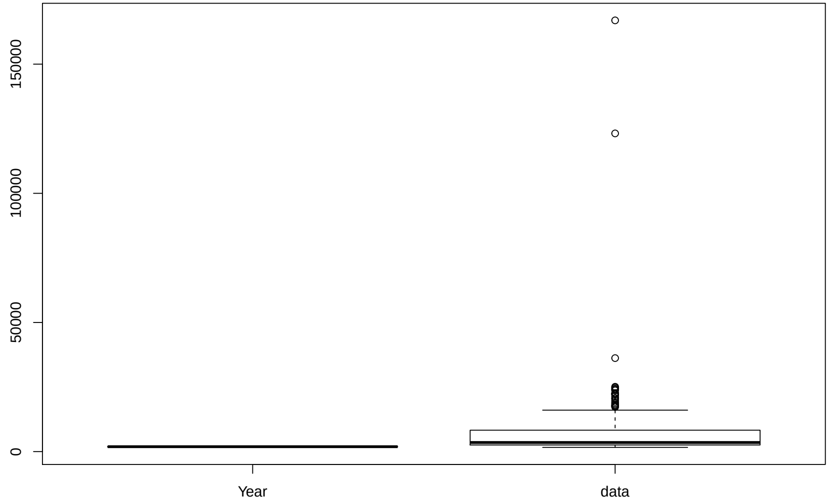
Look whether the function you wish to use has a built-in NA handling feature and what its default settings are.

You can then proceed to use the function like you intended to.

Example of handling NA's

Let's now look at a concrete example:

```
p1 <- boxplot(NLclean)
```



```
p1
```

Handling NA's

While the boxplot might look rather strange, due to reasons which we will ignore for now, the function itself worked just fine.

It ignored the missing values and plotted the boxplot regardless. So this is one, very basic way to cope with NAs in your dataset.

Simple imputation methods dedicated to NA handling

Now, we introduce some simple imputation methods to deal with missing data. As long as the NAs are MCAR and below, or slighly above, 5% of all observations, these methods provide a reasonable solution.

There are several packages available which offer such a solution, but the most commonly used package is the `zoo` package. It contains many functions for missing data handling, such as:

- `na.locf()`, which is a generic function for replacing each NA with the most recent non-NA prior to it
- `na.spline()` & `na.approx()`, which are generic functions for replacing each NA with interpolated values
- `na.aggregate()`, which is a generic function for replacing each NA with aggregated values

As always you can find more information about these functions by reading the documentation.

Example

Let us start by installing the zoo package.

```
install.packages('zoo')
```

And don't forget to 'turn it on':

```
library(zoo)
```

While all of these function have their advantages and disadvantages which you should carefully take into consideration when using them in your own analysis, a detailed discussion of all of these, will take us far beyond the introductory scope of this course. Therefore, we will focus our attention to the `na.locf` function which is primary used for time series analysis.

Example [2]

Let us now take a closer look at what this function does.

```
NLfull <- na.locf(NLclean)

summary(NLclean)
```

```
##       Year           data       
##  Min.   :1800   Min.   :  1640  
##  1st Qu.:1852   1st Qu.:  2566  
##  Median :1905   Median :  3531  
##  Mean   :1905   Mean   :  8140  
##  3rd Qu.:1958   3rd Qu.:  8287  
##  Max.   :2010   Max.   :166955  
##                 NA's   :14
```

```
summary(NLfull)
```

```
##       Year           data       
##  Min.   :1800   Min.   :  1640  
##  1st Qu.:1852   1st Qu.:  2456  
##  Median :1905   Median :  3508  
##  Mean   :1905   Mean   :  8025  
##  3rd Qu.:1958   3rd Qu.:  8244  
##  Max.   :2010   Max.   :166955
```

As you can see, when we compare the summary statistics of NLfull compared to NLclean, it no longer has any NAs. Clearly, the imputation function worked as it was intended to.

Recap

Always check the presence of NAs using `str` and `summary`.

Address NAs, by:

- relying on built-in handling features: e.g. ‘na.action’ = NULL
- omitting / deleting NAs from the data: e.g. ‘na.omit’
- simple imputationg methods: e.g. ‘na.locf’ for time series

Part 2: Outliers

Basic guidelines

> *An outlier is an observation which deviates from the other observations as to arouse suspicions that is was generated by a different mechanism*

Consequently, ignorning outliers might greatly bias your results and should therefore be handled accordingly.

Basic guidelines [2]

Most outliers are in actuality faulty data points, i.e. a wrong measurement due to human mistakes.

In Economic history, it is likely that there are faulty data entries in the underlying sources and/or faulty transcriptions of these sources.

In some occasions however, outliers can be valid measurements. Clearly, it is very important to distinguish between both of these types of outliers.

Common ways to deal with outliers

The most common ways to deal with outliers are:

- Deleting the outlier, but this can greatly reduce the statistical power of your (small) sample
- Ignoring the outlier, but this can bias your results
- Subsituting the outlier for a different value, but this runs the risk of overly excessive data manipulation

For now, we focus on actually detecting the outliers.

Simple approaches to detect outliers: Three Sigma Rule and Boxplots,

Three Sigma Rule

One of the most basics ways to detect outliers mathematically, is the Three Sigma Rule.

We start by simply calculating an interval around the mean: - Lower boundry (lb):

$\mu - n \cdot \sigma$

where μ is the mean, and σ the standard deviation.

- Upper boundry (ub):

$\mu + n \cdot \sigma$

All data points that are with the range are considered to be ‘normal’, whereas observation outside of this range are conidered to be outliers.

The parameter n

Usually *n* is set to 3, hence the name Three Sigma. This is because for normally distributed data the probability of observing a value more than three standard deviations from the mean is only 0.3%.

Let us now apply this simple method to detect outliers in the `NLfull` dataset.

We set n = 3 and the subsequently calculate the mean and standard deviation for GDP per Capita (i.e. NLfull$Data), and the lower -, and upper-boundries of our data.

We use NLfull instead of NLclean simply because the funtion 'mean' and 'sd' do not work with NAs. Another solution would simply be to rely on the built-in NA handling features of these functions (i.e. set `na.rm = TRUE`).

Approach

We then ask R to provide us with list of 0 and 1, where 1 equals an outliers. Finally, we plot our data, but ask R to colour outliers differently using the ‘col’ argument.

Please note that we will discuss ifelse functions and plots much more extensively in upcoming lectures, so there is no need to panic if something is unclear for now. You are doing great!

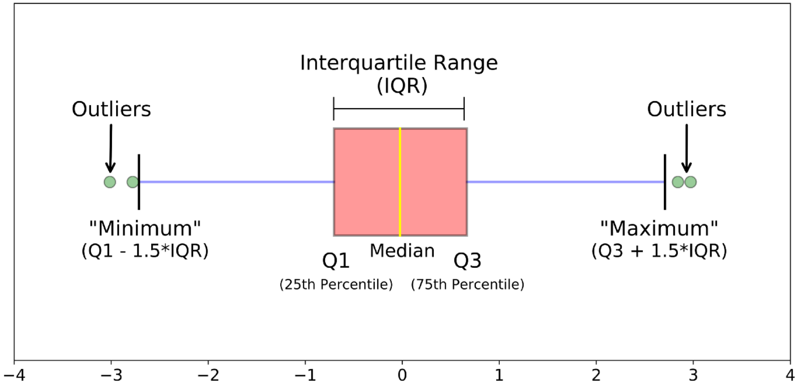
```
n <- 3
mean <- mean(NLfull$data)
sd <- sd(NLfull$data)

lb = mean - n * sd
ub = mean + n * sd

outliers = ifelse(NLfull$data >= lb & NLfull$data <= ub,0,1)

str(outliers)
```

```
##  num [1:211] 0 0 0 0 0 0 0 0 0 0 ...
```

```
#plot(NLclean$Data, col = outliers+1)
```

Boxplots

Another commonly used, basic approach to detect outliers, is to rely on boxplots, which provide a very visible way to detect outliers.



What are boxplots?

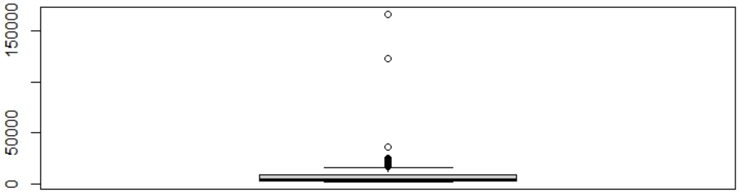
Boxplots are a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”).

- median (Q2/50th Percentile): the middle value of the dataset.
- first quartile (Q1/25th Percentile): the middle number between the smallest number (not the “minimum”) and the median of the dataset.
- third quartile (Q3/75th Percentile): the middle value between the median and the highest value (not the “maximum”) of the dataset.
- interquartile range (IQR): 25th to the 75th percentile.
- “maximum”: Q3 + 1.5*IQR
- “minimum”: Q1 -1.5*IQR
- whiskers: hown in blue
- outliers: shown as green circles

Boxplots

We already made a boxplot for our dataset and observed that it looked rather strange. Let us recapitulate, but this time we also rely on the ‘boxplot.stats’ function to have a closer look at the underlying data.

```
boxplot(NLfull$Data)
boxplot.stats(NLfull$Data)
```



No picture here

`![](data/lecture1d_6.png)`

This shows the aformentioned summary statistics (stats), e.g. median, first and thid quartile.

It also shows the number of observations (n), the IQR (conf) and the outliers ($out).

Please be aware however that many of these values might be false positives.

They could be well within the distribution and should not be marked as outliers which one should ignore or omit from the dataset.

More complicated approaches to detect outliers: Outliers package

The ‘outliers’ package provides much more complicated tools to detect outliers. These include:

- chisq.out.test, which preforms a a chisquared test for detection of one outlier in a vector
- dixon.test, which performs several variants of Dixon test for detecting outlier in data sample
- outlier, which finds the value with largest difference between it and sample mean, which can be an outlier.

Simple approaches to detect outliers: Three Sigma Rule and Boxplots,

By far the most intuitive of these function is the last one, ‘outlier’, let us therefore look at this function more closely. If you wish to know more about these functions, you can always read their corresponding vignettes.

As always, we have to start by installing the `outliers` package.

```r
install.packages('outliers')
```

And then ‘turn it on’:

```r
library(outliers)
```

In practice

Let us now look wat this function does in practice.

```r
outlier(NLfull)
```

```
##   Year   data
##   2010 166955
```

Voila, it tells us that the GDP Per Capita for 2010, 166955 is an outlier, it could therefore potentially be ignored, omitted or replaced for further data analysis.

Summary

We have presented three relative simple ways to detect outliers for an univariate dataset.

- The Three Sigma Rule
- Boxplots
- Tests from the ‘outliers’ package

It is important to keep in mind that all of these methods have their downsides and should only be used with extreme caution!