Data Pre-processing and Cleaning in R

Step-by-Step Tutorial

Matteo Manca (matteo.manca@eurecat.org)

February 22, 2016

Introduction and usefull functions

- R is case sensitive : `VARIABLE` is different from `variable`
- All R functions have some mandatory parameters and some optional parameters.
- Use ***`?function`*** or ***`help(function)`*** to get help for a specified `function` ***Each time you need to use a new function, first check how it works and which are all possible parameters***.
- During the Pre-processing and data cleaning verify often temporary results to see if your doing right
- ***c(v1,v2,...)***: Create a vector (A vector is a sequence of data elements of the same basic type - numeric, char, etc.)
- Missing values are identified by `NA` value

Introduction

- Create a directory where you'll store all files of this tutorial (this will be your working directory -WD-)
- Open RStudio
- Create new RScript and save it your directory WD
- Set your working directory

```
setwd("PATH_OF_YOUR_WD")
setwd("C:/Users/User Name/Documents/FOLDER")
```

```
setwd("~/Dropbox/Sync/Research-Projects/Courses/tutorial/")
```

Datasets

A ***data set*** is a collection of data that describes attribute values (variables) of a number of real-world objects (units).

- Create a new ***data frame*** (In R a data frame is an object used to store data. It is a list of (column) vectors of equal length.)
- create a vector for each column: `vector <- c(value1, value2, ...)  = creates a vector with specified values`
- create the dataset from the just created vectors

```
person <- c("Matteo","Jhon", "Alice","Maria") #Vector of person names
age <- c(18, NA, 22,15) #Vector of person ages
gender <- c("M","M","F","F")
df1 <- data.frame(person, age, gender) #Create the dataframe with the vecto data
```

Explore the Datasets

- ***head()/tail()*** Returns the first or last parts of a vector, matrix, table, data frame or function
- ***summary()*** Produce result summaries of the results of various model fitting function
- try the "?head"' command

```
head(df1)
```

```
##   person age gender
## 1 Matteo  18      M
## 2   Jhon  NA      M
## 3  Alice  22      F
## 4  Maria  15      F
```

```
summary(df1)
```

```
##     person       age        gender
##  Alice :1   Min.   :15.00   F:2
##  Jhon  :1   1st Qu.:16.50   M:2
##  Maria :1   Median :18.00
##  Matteo:1   Mean   :18.33
##             3rd Qu.:20.00
##             Max.   :22.00
##             NA's   :1
```

Create dataset df2

- In the same way create a secon data frames containg information about the country of these pesrsons

```
person <- c("Matteo","Jhon", "Alice","Maria") #Vector of person names
country <- c("IT","US","US","SP")
df2 <- data.frame(person, country) #Create the dataframe with the vecto data
head(df2)
```

```
##   person country
## 1 Matteo      IT
## 2   Jhon      US
## 3  Alice      US
## 4  Maria      SP
```

```
summary(df2)
```

```
##     person  country
##  Alice :1   IT:1
##  Jhon  :1   SP:1
##  Maria :1   US:2
##  Matteo:1
```

Merge the two dataset in a unique dataset

Now we have two daasets that contain information of the same persons. We want to crate a uniqie dataset df

```
df <- merge(df1, df2, by = "person")
head(df)
```

```
##   person age gender country
## 1  Alice  22      F      US
## 2   Jhon  NA      M      US
## 3  Maria  15      F      SP
## 4 Matteo  18      M      IT
```

Access a given column/field of the dataframe

There are two main ways: - using the $ operator - specifying df[,"field_name"] - Example

```
df$person
```

```
## [1] Alice  Jhon   Maria  Matteo
## Levels: Alice Jhon Maria Matteo
```

```
#OR
df[,"person"]
```

```
## [1] Alice  Jhon   Maria  Matteo
## Levels: Alice Jhon Maria Matteo
```

Explore the dataset

- Count how many male and femal there are in our dataframe

```
table(df$gender)
```

```
##
## F M
## 2 2
```

Cleaning dataset

We have a ***missing value*** (see age field)! We can:

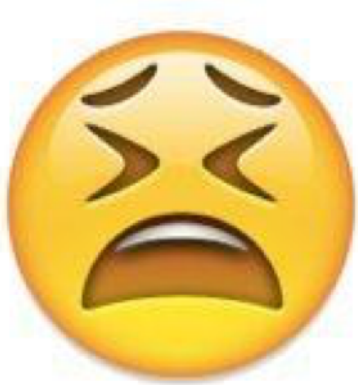
- Ignore the record containing the missing value
- Fill the value with the mean (or some other statistic value)
- Try to predict the missing value
- Apply some heuristic (an approach that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals)

Fill the missing value with the mean

- Using the $ we can access a specified column `col1` of the dataframe: `dataframe$col1`
- `dataframe$column[is.na(dataframe$column)]` select all records in `dataframe` that contain NA values in `column`
- `mean(vector)` computes the mean of the values contained in `vector`

```
#head(df)
df$age[is.na(df$age)] <- mean(df$age)
head(df)
```

```
##   person age gender country
## 1  Alice  22      F      US
## 2   Jhon  NA      M      US
## 3  Maria  15      F      SP
## 4 Matteo  18      M      IT
```



SURPRISE!! Nothing happens!!! Why??

Fill the missing value with the mean

In previous slide we were trying to compute the mean of a vector containg also non-numeric value (there is the NA value!!): `age ==> 22, NA, 15 , 18`

R is not able to do that!!

We need to specify to IGNORE NA values when computing the mean by passing `na.rm = TRUE` to the mean function

```
df$age[is.na(df$age)] <- mean(df$age, na.rm = TRUE)
head(df)
```

```
##   person      age gender country
## 1  Alice 22.00000      F      US
## 2   Jhon 18.33333      M      US
## 3  Maria 15.00000      F      SP
## 4 Matteo 18.00000      M      IT
```

Round the age values: `round(value/vector,NUMBER_OF_DECIMAL_DIGITS)`

```
df$age <- round(df$age,0)
head(df)
```

```
##   person age gender country
## 1  Alice  22      F      US
## 2   Jhon  18      M      US
## 3  Maria  15      F      SP
## 4 Matteo  18      M      IT
```

Dataframe subsetting

- Select first two rows of the dataframe and all columns

```
df[1:2,]
```

```
##   person age gender country
## 1  Alice  22      F      US
## 2   Jhon  18      M      US
```

- Select rows 1 and 3 of the dataframe and all columns

```
df[c(1,3),]
```

```
##   person age gender country
## 1  Alice  22      F      US
## 3  Maria  15      F      SP
```

Dataframe subsetting

- Select rows 1 and 3 of the dataframe and columns 1 and 3

```
df[c(1,3),c(1,3)]
```

```
##   person gender
## 1  Alice      F
## 3  Maria      F
```

- Select all rows where gender = female

See function which `?which`

```
df[which(df$gender=='F'),]
```

```
##   person age gender country
## 1  Alice  22      F      US
## 3  Maria  15      F      SP
```

```
df[df$gender=='F',]
```

```
##   person age gender country
## 1  Alice  22      F      US
## 3  Maria  15      F      SP
```

```
df[df["gender"]=='F',]
```

```
##   person age gender country
## 1  Alice  22      F      US
## 3  Maria  15      F      SP
```

Save the dataframe in a file

We'll save it in a .csv (Coma Separate Value) file

```
write.csv(df, file = "test_df.csv")
```

check if the file is in your WD

Read dataset from file and load it in a new data frame

- Download data (Electric power consumption) from: https://www.dropbox.com/s/2b3tnp9svups3zq/household_power_consumption.txt?dl=0 and save it in your WD

Description: Measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are available.

The following descriptions of the 9 variables in the dataset are taken from the UCI web site:

- ***Date***: Date in format dd/mm/yyyy
- ***Time***: time in format hh:mm:ss
- ***Global_active_power***: household global minute-averaged active power (in kilowatt)
- ***Global_reactive_power***: household global minute-averaged reactive power (in kilowatt)
- ***Voltage***: minute-averaged voltage (in volt)
- ***Global_intensity***: household global minute-averaged current intensity (in ampere)
- ***Sub_metering_1***: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
- ***Sub_metering_2***: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
- ***Sub_metering_3***: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.
- Note that in this dataset missing values are coded as ?.
- The separator character is ';' .

Read data from file

- load the data in a dataframe (functions: read.table, read.csv)

Main parameters of `read.table` and `read.csv`:

- *file:* the name of the file which the data are to be read from.
- *header:* a logical value indicating whether the file contains the names of the variables as its first line.
- *sep:* the field separator character. Values on each line of the file are separated by this character.
- *na.strings:* a character vector of strings which are to be interpreted as NA values.
- first see how the funcion `read.csv` works using `?read.csv`

```
df <- read.csv("household_power_consumption.txt", sep = ';', na.strings = '?', header = T)
```

Explore dataset

We can use functions like head, summary, dim, etc.

```
head(df)
```

```
##         Date     Time Global_active_power Global_reactive_power Voltage
## 1 16/12/2006 17:24:00               4.216                 0.418  234.84
## 2 16/12/2006 17:25:00               5.360                 0.436  233.63
## 3 16/12/2006 17:26:00               5.374                 0.498  233.29
## 4 16/12/2006 17:27:00               5.388                 0.502  233.74
## 5 16/12/2006 17:28:00               3.666                 0.528  235.68
## 6 16/12/2006 17:29:00               3.520                 0.522  235.02
##   Global_intensity Sub_metering_1 Sub_metering_2 Sub_metering_3
## 1             18.4              0              1             17
## 2             23.0              0              1             16
## 3             23.0              0              2             17
## 4             23.0              0              1             17
## 5             15.8              0              1             17
## 6             15.0              0              2             17
```

```
summary(df)
```

```
##         Date              Time        Global_active_power
##  1/1/2007 :  1440   17:41:00:   322   Min.   : 0.082
##  1/10/2007:  1440   17:42:00:   322   1st Qu.: 0.268
##  1/11/2007:  1440   17:43:00:   322   Median : 0.462
##  1/2/2007 :  1440   00:00:00:   321   Mean   : 1.086
##  1/3/2007 :  1440   00:01:00:   321   3rd Qu.: 1.514
##  1/4/2007 :  1440   00:02:00:   321   Max.   :10.670
##  (Other)  :452595   (Other) :459306   NA's   :3932
##  Global_reactive_power    Voltage      Global_intensity Sub_metering_1
##  Min.   :0.000         Min.   :223.5   Min.   : 0.400   Min.   : 0.000
##  1st Qu.:0.000         1st Qu.:236.7   1st Qu.: 1.200   1st Qu.: 0.000
```

```
##  Median :0.102         Median :239.5   Median : 2.200   Median : 0.000
##  Mean   :0.120         Mean   :239.1   Mean   : 4.646   Mean   : 1.192
##  3rd Qu.:0.190         3rd Qu.:241.6   3rd Qu.: 6.400   3rd Qu.: 0.000
##  Max.   :1.148         Max.   :251.7   Max.   :46.400   Max.   :78.000
##  NA's   :3932          NA's   :3932    NA's   :3932     NA's   :3932
##  Sub_metering_2   Sub_metering_3
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 0.000
##  Mean   : 1.633   Mean   : 5.523
##  3rd Qu.: 1.000   3rd Qu.:17.000
##  Max.   :78.000   Max.   :20.000
##  NA's   :3932     NA's   :3932
```

```
dim(df) #dimensions of the dataset rows x columns
```

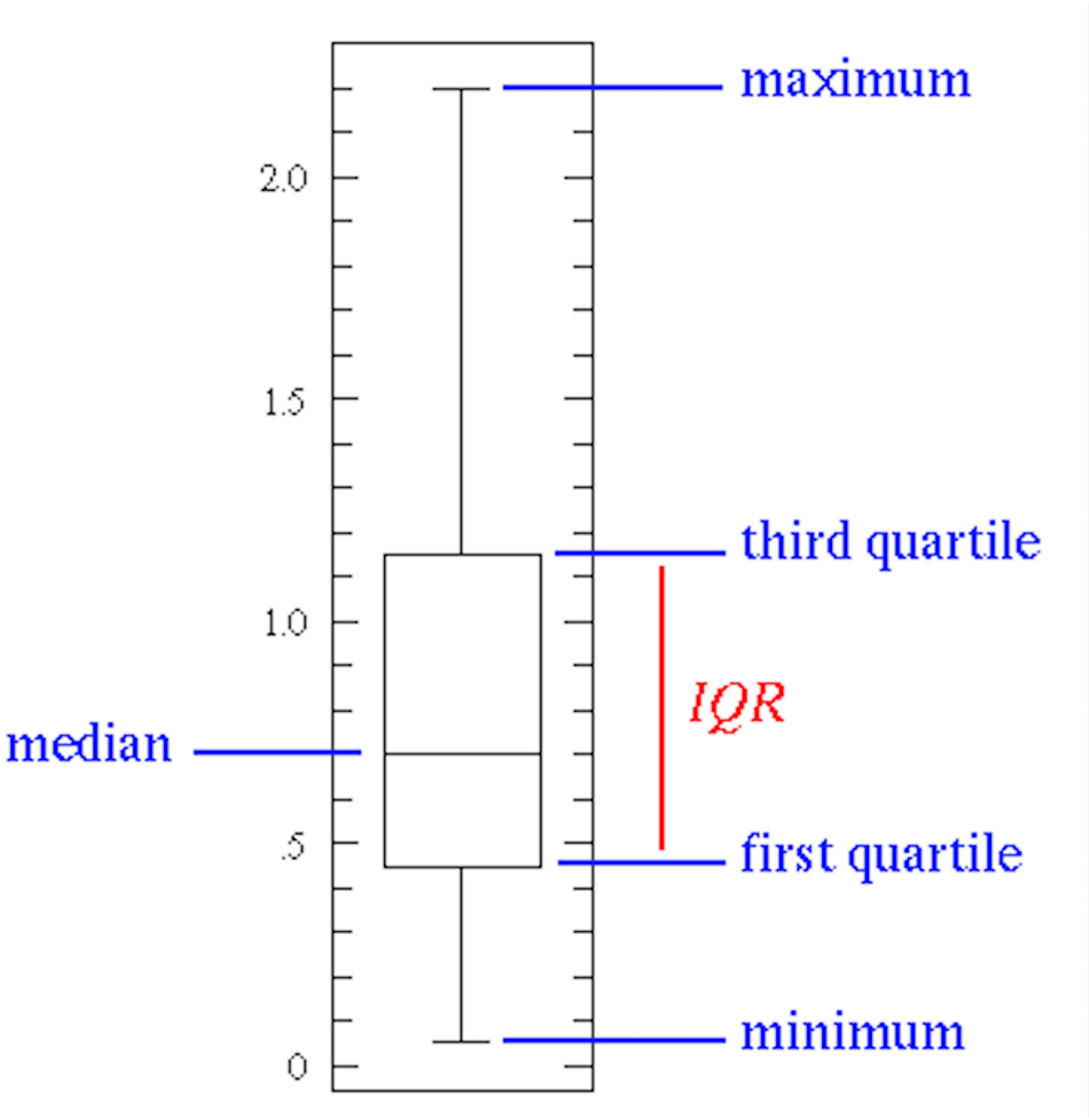
```
## [1] 461235      9
```

```
names(df) ##see the column names of the dataframe
```

```
## [1] "Date"                  "Time"                  "Global_active_power"
## [4] "Global_reactive_power" "Voltage"               "Global_intensity"
## [7] "Sub_metering_1"        "Sub_metering_2"        "Sub_metering_3"
```

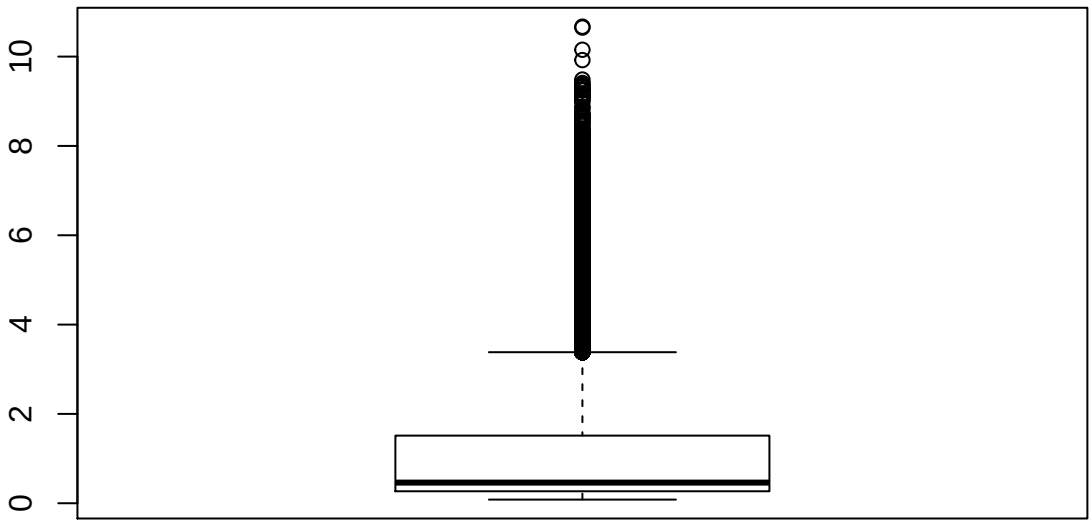
Explore dataset

In addiction to functions we can explore the data also with plots. Suppose you want to explore the `Global_active_power` variable. We can build a box plot to obtain an overview of the values. A ***boxplot*** is a plot that shows the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum.



Explore dataset

```
boxplot(df$Global_active_power)
```



Rename the column names

```
names(df) <- c("date","time", "active_power", "reactive_power", "Voltage", "intensity", "Sub_met_1", "Su
head(df)
```

```
##         date     time active_power reactive_power Voltage intensity
## 1 16/12/2006 17:24:00        4.216          0.418  234.84      18.4
## 2 16/12/2006 17:25:00        5.360          0.436  233.63      23.0
## 3 16/12/2006 17:26:00        5.374          0.498  233.29      23.0
## 4 16/12/2006 17:27:00        5.388          0.502  233.74      23.0
## 5 16/12/2006 17:28:00        3.666          0.528  235.68      15.8
## 6 16/12/2006 17:29:00        3.520          0.522  235.02      15.0
##   Sub_met_1 Sub_met_2 Sub_met_3
## 1         0         1        17
## 2         0         1        16
## 3         0         2        17
## 4         0         1        17
## 5         0         1        17
## 6         0         2        17
```

Count the number of distinct date

See:

- ?length
- ?unique

```
length(unique(df$date))
```

```
## [1] 322
```

Check for duplicates record in the dataset

- if there are duplicates, verify how many
- create a new dataset containg only the duplicates records and save in a .csv file
- remove the duplicates from the original dataset

```
dim(df[duplicated(df), ])
```

```
## [1] 3 9
```

```
dup_df <- df[duplicated(df), ]
write.csv(dup_df, file="duplicates.csv")
df <- df[!duplicated(df), ]
dim(dup_df)
```

```
## [1] 3 9
```

```
dim(df)
```

```
## [1] 461232      9
```

Check if there are records containing NA values

?complete.cases Return a logical vector indicating which cases are complete, i.e., have no missing values. Example:

```
> airquality[1:7,]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
7    23     299  8.6   65     5   7
> complete.cases(airquality[1:7,])
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
                           5     6
```

Check if there are records containing NA values

```
head(complete.cases(df))
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

```
dim(df[!complete.cases(df),]) ##NOTE THE "!" that means NOT
```

```
## [1] 3932    9
```

The result of `df[!complete.cases(df),]` is a dataframe containing all record with NA

Check if there are records containing NA values

- Remove all values with NA (in the first toy dataset we used we replace NA with mean)

```
df_withoutna <- df[complete.cases(df),] ##keep just complete records
```

- Verify the obtained result (the dimension of the dataset without NA is equal to dim(original dataframe) - dim(NAs) ??)

```
dim(df) ##Original datasets
```

```
## [1] 461232      9
```

```
dim(df[!complete.cases(df),]) ##records containing NAs
```

```
## [1] 3932    9
```

```
dim(df_withoutna) ##new datasets without NAs
```

```
## [1] 457300      9
```

It's a good pratice to make these type of checks to verify if your pre-processing steps are doing well.

Add column to dataframe

- extract the "month" from the date field and add a column called "month"
- first convert the date object in the R object used to represent dates ***as.POSIXct function***
- to extract the our use the function ***as.POSIXlt function*** (see ?as.POSIXlt)

```
##The as.POSIXct functions convert the time object to the class used to represent times in R
##df_withoutna$month add a column month to the dataframe
df_withoutna$date <- as.POSIXct(df_withoutna$date, format = "%d/%m/%Y")

##add a field/column called month to the dataframe (remember that now we are working on the new datafra
df_withoutna$month <- format(as.POSIXlt(df_withoutna$date), "%Y-%m")

#print an overview of the dataset
head(df_withoutna)
```

```
##         date     time active_power reactive_power Voltage intensity
## 1 2006-12-16 17:24:00        4.216          0.418  234.84      18.4
## 2 2006-12-16 17:25:00        5.360          0.436  233.63      23.0
## 3 2006-12-16 17:26:00        5.374          0.498  233.29      23.0
## 4 2006-12-16 17:27:00        5.388          0.502  233.74      23.0
## 5 2006-12-16 17:28:00        3.666          0.528  235.68      15.8
## 6 2006-12-16 17:29:00        3.520          0.522  235.02      15.0
##   Sub_met_1 Sub_met_2 Sub_met_3   month
## 1         0         1        17 2006-12
## 2         0         1        16 2006-12
## 3         0         2        17 2006-12
## 4         0         1        17 2006-12
## 5         0         1        17 2006-12
## 6         0         2        17 2006-12
```

Aggregate values

Data can be aggregated in different ways based on different functions (length, mean, etc).

- ***?aggregate function:*** Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

EXAMPLE 1 - Compute the average active_power per month (we are aggregating by month) and save the result ina new dataframe "avg_active_power_xmonth"

```
avg_active_power_xmonth <-  aggregate(active_power ~ month , df_withoutna, FUN=mean)
head(avg_active_power_xmonth)
```

```
##     month active_power
## 1 2006-12    1.9012951
## 2 2007-01    1.5460339
## 3 2007-02    1.4010835
## 4 2007-03    1.3186270
## 5 2007-04    0.8911889
## 6 2007-05    0.9858618
```

Aggregate values

EXAMPLE 2

- Compute the number of active_power measurements per month (So, we are aggregating by month) and save the res in anew df "meas_xmonth"

```
meas_xmonth <-  aggregate(active_power ~ month , df_withoutna, FUN=length)
names(meas_xmonth) <- c("month","meas_count")
head(meas_xmonth)
```

```
##     month meas_count
## 1 2006-12      21992
## 2 2007-01      44638
```

```
## 3 2007-02      40318
## 4 2007-03      44639
## 5 2007-04      39477
## 6 2007-05      44640
```

R do not change the ***column names*** and this, sometimes, can be ***misleading***; in that case it's always better to rename the columns (Rename the column names).

Sort dataframe by a specific column

```
avg_active_power_xmonth <-  avg_active_power_xmonth[order(avg_active_power_xmonth$active_power),]
head(avg_active_power_xmonth)
```

```
##      month active_power
## 12 2007-11    0.3427087
## 8  2007-07    0.6673668
## 9  2007-08    0.7641862
## 7  2007-06    0.8268144
## 5  2007-04    0.8911889
## 10 2007-09    0.9693182
```

- Sort DESC

```
avg_active_power_xmonth <-  avg_active_power_xmonth[order(-avg_active_power_xmonth$active_power),]
head(avg_active_power_xmonth)
```

```
##      month active_power
## 1  2006-12    1.9012951
## 2  2007-01    1.5460339
## 3  2007-02    1.4010835
## 4  2007-03    1.3186270
## 11 2007-10    1.1039108
## 6  2007-05    0.9858618
```

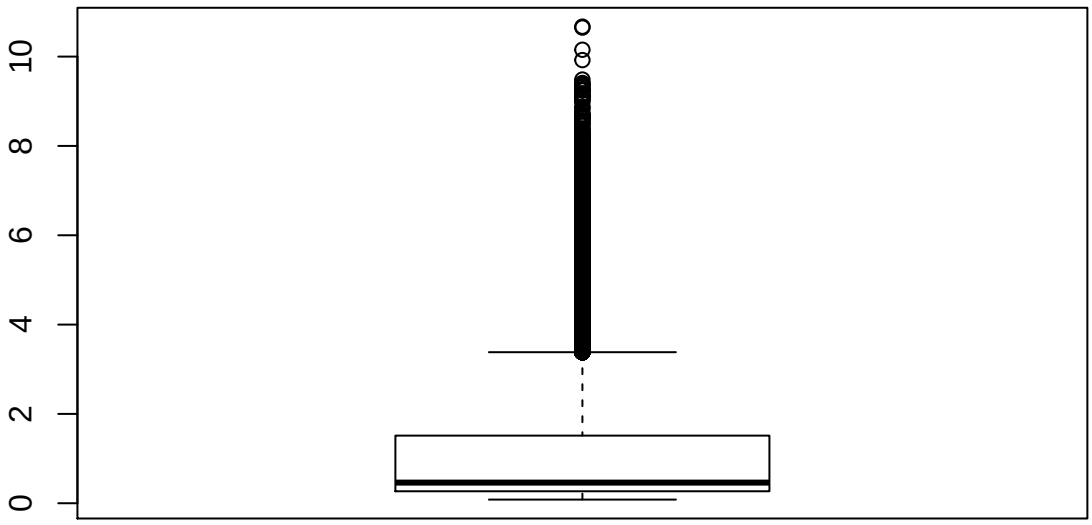
Introduction to outliers

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism"[Hawkins 1980]

There exist several ways to discover outliers. A first step may be to plot the values to see if there are objects that deviate significantly from the rest of the dataset.

Example: See if there are outliers in the active power measurements of our dataset

```
boxplot(df_withoutna$active_power)
```



For more detalis about the outliers: https://www.siam.org/meetings/sdm10/tutorial3.pdf

References

1. https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf
2. https://www.coursera.org/learn/data-cleaning
3. https://www.coursera.org/learn/r-programming
4. http://www.r-bloggers.com
5. https://github.com/matteomanca/DataScienceSpCourseNotes/blob/master/2_RPROG/R_Programming_Course_Notes.pdf
6. https://www.siam.org/meetings/sdm10/tutorial3.pdf