IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Session 2

EDA and Data Cleaning in R

Course Name:

Supervised Machine Learning

By Prof. Vala A. Rohani
vala.ali.rohani@estsetubal.ips.pt

Session outline

1. Reviewing the essential topics in R Programming
2. Exploratory Data Analysis in R
3. Preparation and Data Cleaning for ML (Machine Learning) in R

Reviewing the essential topics in R Programming

R is a programming language for **statistical analysis** and **data visualization**. Using R, you can take raw data and understand the trends and patterns in it.
You can also use R to build and validate the machine learning models.

RStudio is an integrated development environment (IDE) for R.

1.Installing R

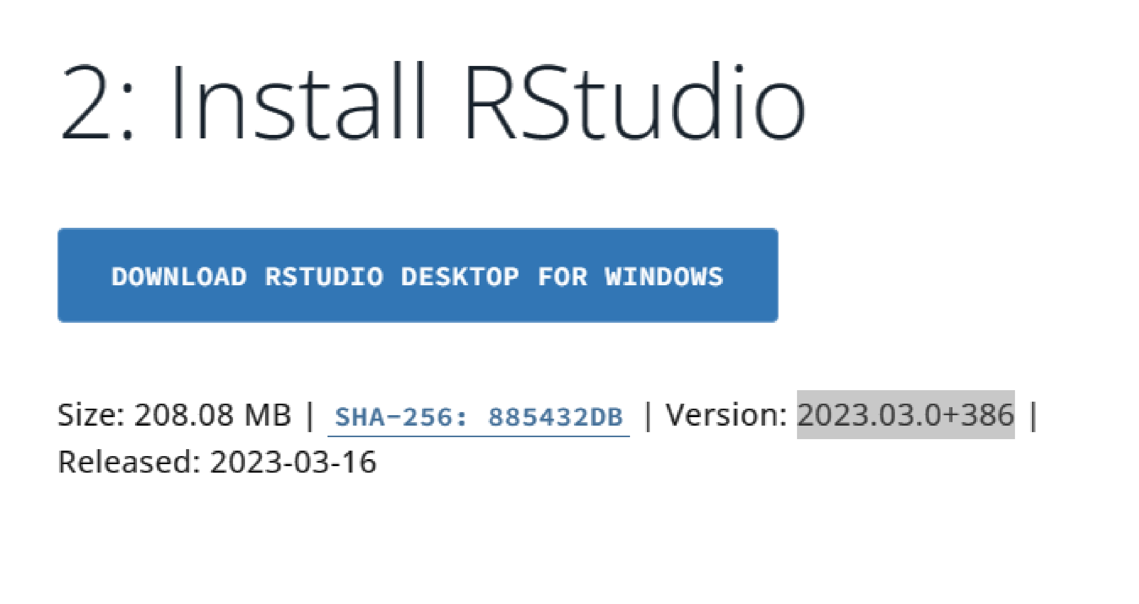
Latest version: 4.2.3 for windows

https://cran.r-project.org/bin/windows/base/

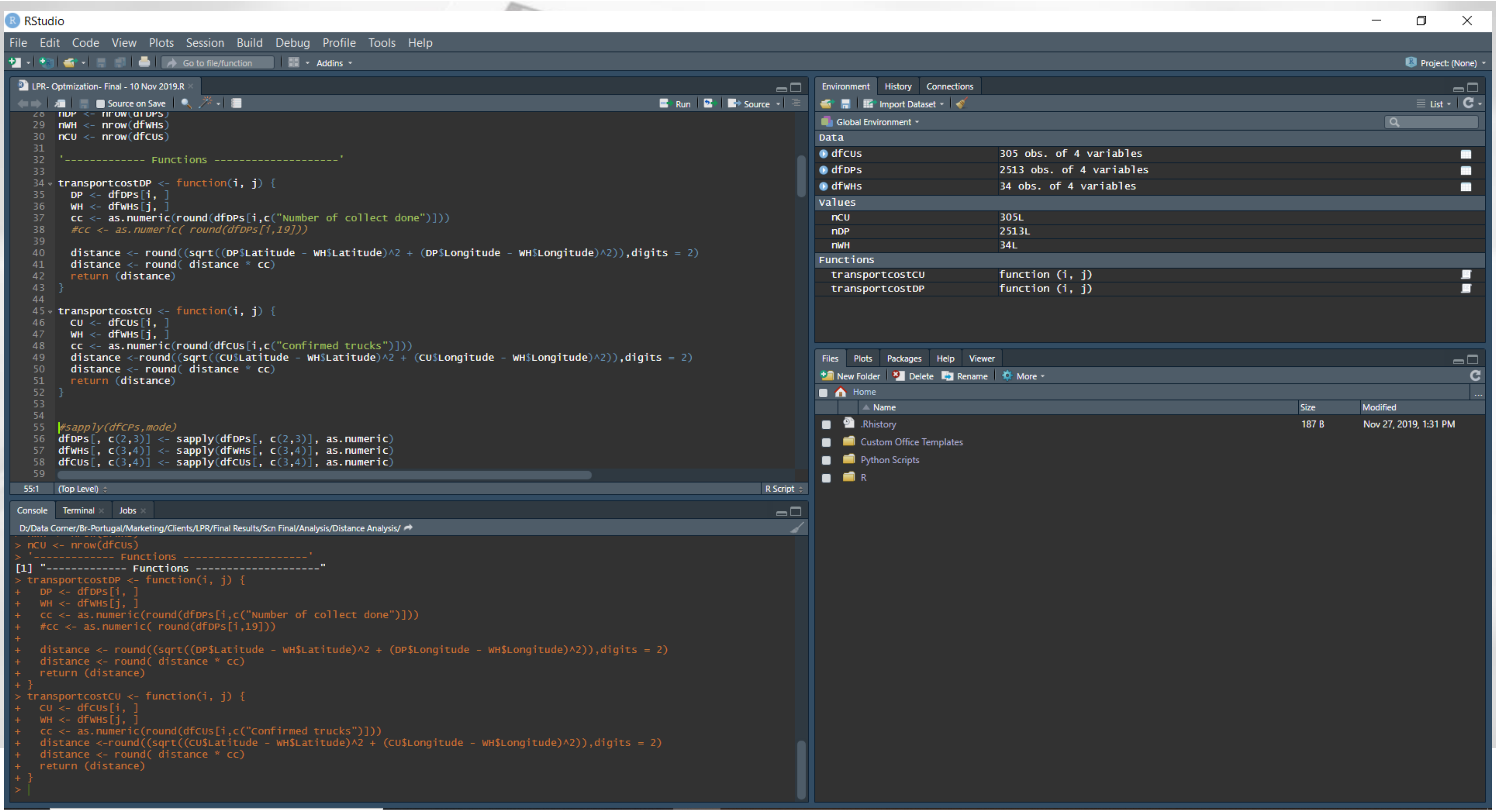
2.Installing R Studio

RStudio Desktop 2023.03.0+386

https://rstudio.com/products/rstudio/download/#download



RStudio environment [you can change the theme through **Tools > Global Option > Appearance**]



First things first!

Create the course folder in your laptop:

/IPS-ESCE SP ML

/Session 2

/Session 3

/Session 4

/Session 5

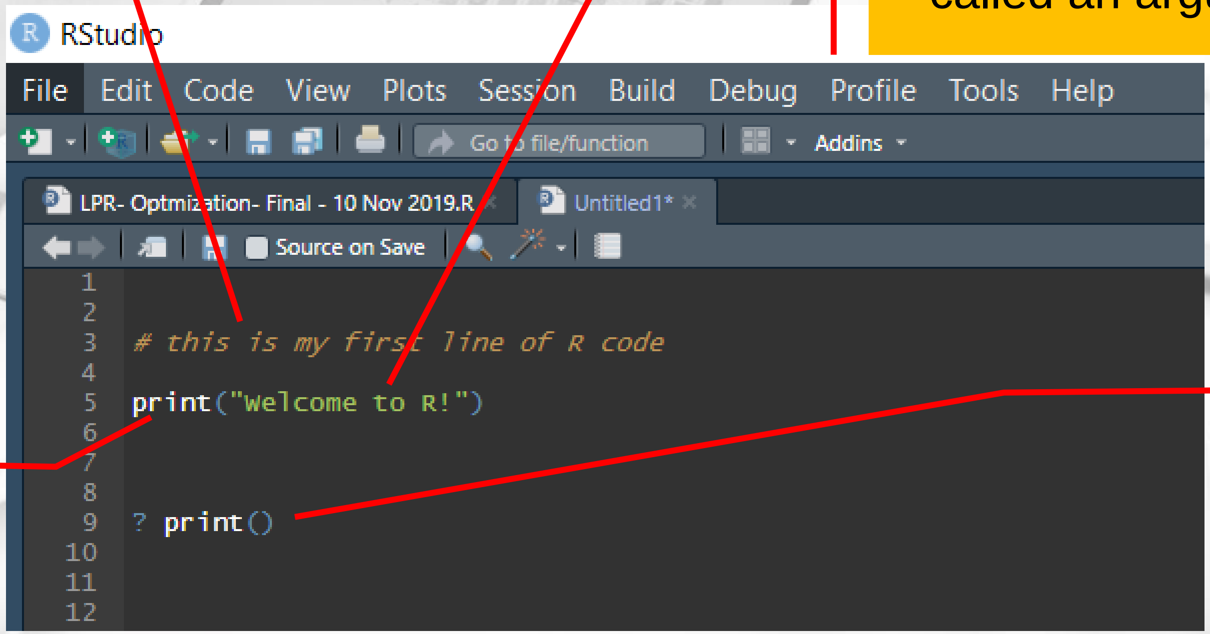
/Session 6

Your first line of code in R!

Ctrl+ENTER to run each line

Comments helps to make your codes more readable

"Welcome to R!" is called an argument



? Print() shows help about print() function

Print() is a function

Variables

Your turn! Create a variable named "aSentence" and store a sentence in it

```r
# in R, you can store data in a variable by using "<-". You can name a
# variable anything you want as long as it's not already the name of something else.
# I find that a short phrase (without spaces) is generally best.

textToPrint <- "this is some text to print"

# if you give R the name of a variable, it will print whatever is in that variable

textToPrint

# note that capitalization does matter! This line will generate an error becuase
# there is nothing called "texttoprint"

texttoprint

####### Your turn! Create a variable named "aSentence" and store a sentence in it

# our old friend print()
print(textToPrint)

# the nchar() function tells you the number of characters in a variable
nchar(textToPrint)

# the c() function concatenates (strings together) all its arguments
c(textToPrint, textToPrint, textToPrint)
```

Lab Activity:

Your turn! Try the following three exercises on your own.

1) print the variable **"aSentence"** you made previously

2) find out the number of characters in **"aSentence"**

3) concatenate the **"textToPrint"** and **"aSentence"** variables

Data Types

R has 6 basic data types:

- **character:** "a", "swc"
- **numeric:** 2, 15.5
- **integer:** 2L (the L tells R to store this as an integer)
- **logical:** TRUE, FALSE
- **complex:** 1+4i (complex numbers with real and imaginary parts)

Data Types

```r
# What we've seen so far are characters. This is the type of data you'll use for text
varStr <- "someText"

# we can check the data type of a variable using the function str() (like "structure")
str(varStr)
# we can tell this is a character because it's structure is "chr"
```

Your turn!
Check the data type of the "aSentence" variable you made above.

Data Types

```r
# let's create some numeric variables
hoursPerDay <- 24
daysPerWeek <- 7

# we can check to make sure that these actually are numeric
class(hoursPerDay)
class(daysPerWeek)

# since this is numeric data, we can do math with it!
# "*" is the symbol for multiplication
hoursPerWeek <- hoursPerDay * daysPerWeek
hoursPerWeek
```

Your turn!
Create a numeric variable "minutesPerHour" and use it to calcuate a new variable called
"minutesPerWeek" that has the number of minutes per week in it

Data Types

Your turn!

Now try this:

a <- 5
b <- "6"

a * b

```r
# You can change character data to numeric data using the as.numeric() function.
# This will let you do math with it again. :)
a * as.numeric(b)

a * b


# check out the stucture: note that b changes from "chr" to "num
str(b)
str(as.numeric(b))

# to fix b to be a number permentantly
b <- as.numeric(b)

str (b)

```

Data Types

So far we've learned about two data types: **character** and **numeric**. But there's a third common data type you'll encounter a lot in R: **logical or boolean** data. Booleans can only take two values, TRUE and FALSE.

In [15]:

```
#--- Boolean types

# You'll get a boolean back if you ask R "are these two things the same?" using "=="

var1 <- "a" == "b"
var2 <- 1 == 1

var1
var2

str (var1)
```

Your turn!

First, take a guess: will 6 == "6" return TRUE or FALSE?
Then test your prediction. Are you surprised by the outcome?
What does this tell you about datatypes in R?

Vectors

In programming, a vector is **list of data that is all of the same data type**.

```r
# vectors

# let's make a vector!
listOfNumbers <- c(1,5,91,42.8,100008.41)
listOfNumbers

str (listOfNumbers)

# becuase this is a numeric vector, we can do math on it! When you do math to a vector,
# it happens to every number in the vector. (If you're familiar with matrix
# mutiplication, it's the same thing as multiplying a 1x1 matrix by a 1xN matrix.)

# multiply every number in the vector by 5
5 * listOfNumbers


# add one to every number in the vector
listOfNumbers + 1

listOfNumbers

```

Your turn!

divide every number in the vector listOfNumbers by 2 and store it in v2.

[In R, you do division using the / symbol]

Vectors

How to access the elements of vector?

You can look inside a vector using square brackets (**[]**). The number inside the square brackets tells R what **index** to look at. *In R, indexes start at one.*

```r
# get the third item from "listofNumbers"
listofNumbers[3]

# and store it in another varibale

varA <- listofNumbers[3]

varA
```

Your turn!

1. Print the sum of first and second item of “listOfNumbers” and,

2. Check if the first item of “listOfNumbers” is bigger than the second one or not?

Managing Data Frames

The **data frame** is a key data structure in statistics and in R. The basic structure of a data frame is that there is one observation **per row** and each **column** represents a variable of that observation.

The **dplyr** package is designed to provide a highly optimized set of routines specifically for dealing with data frames.

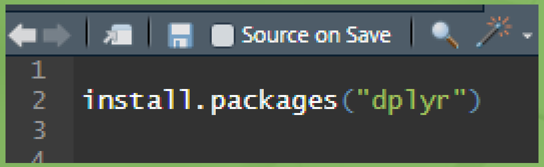
So first,
Lets' see how to install packages in R

Managing Data Frames

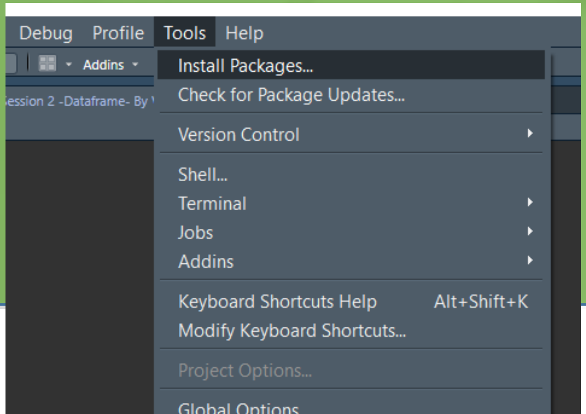
Packages are collections of R functions, data, and compiled code in a well-defined format.

The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used.

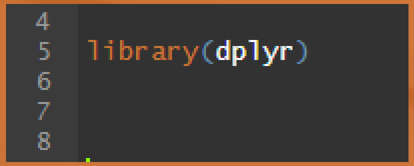
How to install a new package?



Or,



How to use an installed package?



Managing Data Frames

dplyr Grammar:

Some of the key "verbs" provided by the dplyr package are:

select: return a subset of the columns of a data frame, using a flexible notation

filter: extract a subset of rows from a data frame based on logical conditions

arrange: reorder rows of a data frame

rename: rename variables in a data frame

mutate: add new variables/columns or transform existing variables

summarise / summarize: generate summary statistics of different variables in the data frame, possibly within strata

%>%: the "pipe" operator is used to connect multiple verb actions together into a pipeline

Managing Data Frames

Now, we need to read the data :

For the examples in this chapter we will be using a dataset containing air pollution and temperature data for the city of Chicago in the U.S. The dataset is available in the shared folder.

1. **Setting working directory:**

```r
# --------------- Setting the working directory

# First let's see the current working directory

getwd()

# now, we can set our desired directory as the working directory

setwd("D:/Academia/IPS/2019 Analytics for Master Students/Training Material/Season 2/data/chicago_data")

# let's check it

getwd()
```

2. **Reading data into the dataframe**

```r
dfChicago <- readRDS("chicago.rds")
```

Managing Data Frames

Let’s see some basic characteristics of the dataset with the **dim()** and **str()** functions.

```r
dim(dfChicago)

str (dfChicago)
```

31:1 (Top Level)

Console Terminal Jobs

D:/Academia/IPS/2019 Analytics for Master Students/Training Material/Season 2/data/chicago_data/

```
> dim(dfChicago)
[1] 6940    8
> str (dfChicago)
'data.frame':	6940 obs. of  8 variables:
 $ city      : chr  "chic" "chic" "chic" "chic" ...
 $ tmpd      : num  31.5 33 33 29 32 40 34.5 29 26.5 32.5 ...
 $ dptp      : num  31.5 29.9 27.4 28.6 28.9 ...
 $ date      : Date, format: "1987-01-01" "1987-01-02" "1987-01-03" "1987-01-04" ...
 $ pm25tmean2: num  NA NA NA NA NA NA NA NA NA NA ...
 $ pm10tmean2: num  34 NA 34.2 47 NA ...
 $ o3tmean2  : num  4.25 3.3 3.33 4.38 4.75 ...
 $ no2tmean2 : num  20 23.2 23.8 30.4 30.3 ...
>
```

Managing Data Frames

The **select()** function can be used to select columns of a data frame that you want to focus on. Often you'll have a large data frame containing "all" of the data, but any given analysis might only use a subset of variables or observations. The select() function allows you to get the few columns you might need.

```
#--- select

names(dfChicago)


# -- Results: "city"        "tmpd"        "dptp"        "date"        "pm25tmean2" "pm10tmean2" "o3tmean2"    "no2tmean2"

dfSubset <- select(dfChicago, city:dptp)

names(dfSubset)

# -- Results: "city" "tmpd" "dptp"

head (dfSubset)

```

You can also **omit variables** using the select() function by using the negative sign.

```
# -- Omit columns

select(dfChicago, -(city:dptp))

```

"city"

Managing Data Frames

Your turn!

1. Write a line of code to select columns (“city”, “date”, and “pm10tmean2”) and store them in df3

2. Can you implement it with pipe (%>%)

Managing Data Frames

The **filter()** function is used to extract subsets of rows from a data frame.

```
#--- Filter

dfFiltered <- filter(dfChicago, pm25tmean2 > 30)

head(dfFiltered)

dfFiltered2 <- filter(dfChicago, pm25tmean2 > 30 & tmpd > 80)

head(dfFiltered2)

```

Managing Data Frames

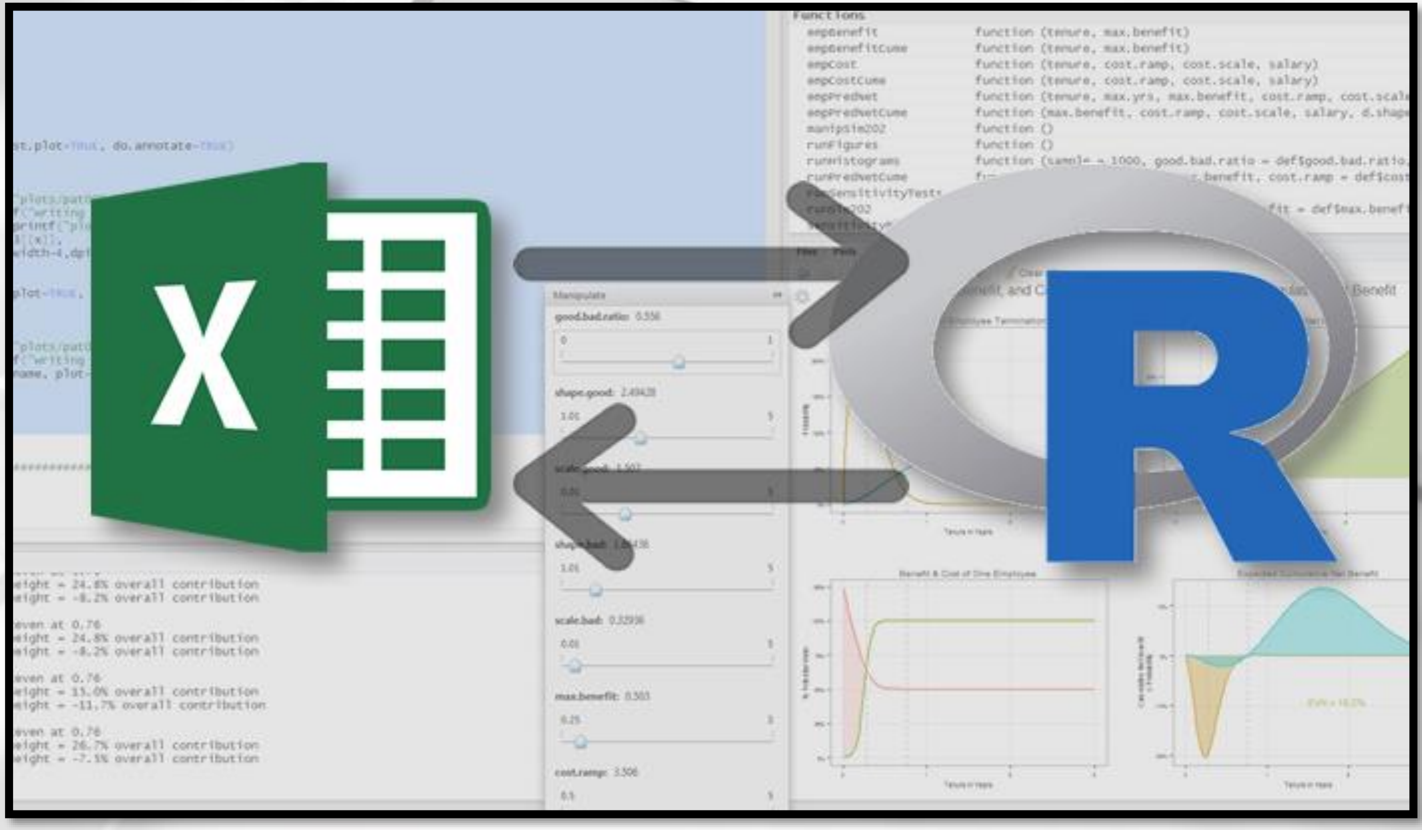
The **arrange()** function is used to sort rows of a data frame according to one of the variables/columns.

```r
#---- Sort

dfSorted <- arrange(dfChicago,date)

dfSorted2 <- arrange(dfChicago, desc (date))

```

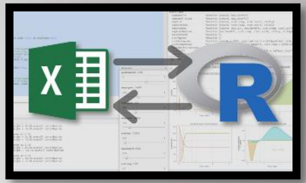
Your turn!
Check the functions **rename()** and **mutate()** from dplyr package.
What is their usage? Apply them on dfChicago

Managing Data Frames

How to communicate with excel?



Managing Data Frames



To communicate with excel you need **readxl** package, so let’s first install it:

```
# ----- Read and write to Excel/CSV

install.packages("readxl")

library("readxl")


# now, we can set our desired directory as the working directory

setwd("D:/Academia/IPS/2019 Analytics for Master Students/Training Material/Season 2/data/chocolate-bar-ratings")

getwd()

#-- reading excel file

dfCacao <- read_excel("flavors_of_cacao.xlsx")


#--- Analysing the data


#--- Storing the dada in csv format

write.csv(dfCacao, file = "flavors_of_cacao_editted.csv")
```

Lab assignment:

1. Read the flavors_of_cacao. Xlsx into the dfCacao data frame.
2. How many columns and rows does it have? What are the column names? What is the datatype of each column?
3. Sort the dataset descending based on Ratings. Store them in separate data frame .
4. How many records exists in Portugal, Spain, Germany, and USA? Store them in separate data frame .
5. How many records exists in Switzerland in 2010 with rating of 3.5? Store them in separate data frame .
6. Store the records of above countries (in 4^{th} step) in separate data frames and compare the average ratings for each of them.
7. Store all generated datafarmes in excel format.

Data cleaning is one of the most important aspects of data science.

As a data scientist, you can expect to spend up to **80% of your time cleaning data**.

In this post you'll learn how to detect missing values using the tidyr and dplyr packages from the Tidyverse.

NA Stands for not available. NA is a placeholder for a missing value.

Your turn!
Try the following codes and see the results

```
# what is NA?


NA + 1
sum(c(NA, 1, 2))
median(c(NA, 1, 2, 3), na.rm = TRUE)
length(c(NA, 2, 3, 4))
3 == NA
NA == NA
TRUE | NA
```

NULL means no class (its class is NULL) and has length 0 so it does not take up any space in a vector.

Your turn!
Try the following codes and see the results

```
# what is NULL?


length(c(1, 2, NULL, 4))
sum(c(1, 2, NULL, 4))
x <- NULL
c(x, 2)
```

Data Cleaning > Graphs Principles > Graphs in R

In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

Missing Values

Statistical analysis error

```r

# ---- Statistical analysis error


age <- c(23, 16, NA)
mean(age)

## [1] NA

mean(age, na.rm = TRUE)
## [1] 19.5

```

Identify and remove the NA value

Missing Values

```
# --- Identify the NA value

complete.cases(age)

#or

is.na(age)

# --- Remove NA values

na.omit(age)

age
```

Data Cleaning > Graphs Principles > Graphs in R

recode missing values with the mean

Missing Values

```r
#--- Recode missing values with mean

x <- c(1:5, NA, 9:11, NA)
is.na(x)

df <- data.frame(col1 = c(1:3, NA),
                 col2 = c("this", NA,"is", "text"),
                 col3 = c(TRUE, FALSE, TRUE, TRUE),
                 col4 = c(2.5, 4.2, 3.2, NA),
                 stringsAsFactors = FALSE)
is.na(df)

# -- To identify the location of the NA.


which(is.na(x))
sum(is.na(df))

#or for data frame

colSums(is.na(df))


# recode missing values with the mean
x

x[is.na(x)] <- mean(x, na.rm = TRUE)

# do it for dataframe

df

df$col4[is.na(df$col4)] <- mean(df$col4, na.rm = TRUE)

```

showing complete and incomplete observations

Missing Values

```
#-- For complete observations.


df

complete.cases(df)


# subset with complete.cases to get complete cases
df[complete.cases(df), ]

# or use na.omit() to get same as above
na.omit(df)

# or subset with `!` operator to get incomplete cases
df[!complete.cases(df), ]
```

What can we do with missing values in datasets?

Missing Values

1. **Deleting the observations**
    - Have sufficient data points, so the model doesn't lose power.
    - Not to introduce bias (meaning, disproportionate or non-representation of classes).

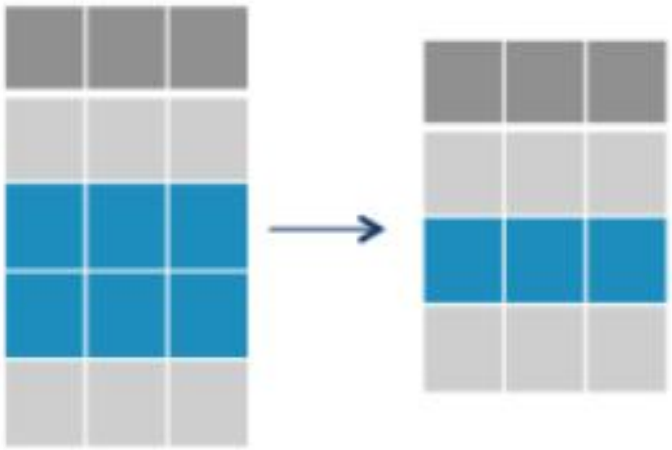
2. Deleting the variable
- When we have lots of missing values for a specific variable

3. Imputation with mean / median / mode
- Replacing the missing values with the mean / median / mode is a crude way of treating missing values. Depending on the context, like if the variation is low or if the variable has low leverage over the response, such

4. Prediction
- Prediction is most advanced method to impute your missing values and includes different approaches such as: kNN Imputation, rpart, and mice.

Duplicate values

Remove Duplicate Data in R



duplicated(): *Identify duplicate elements (R base)*
unique(): *Keep only unique elements (R base)*
distinct(): *Efficient solution to remove duplicate in a data table (dplyr)*

IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

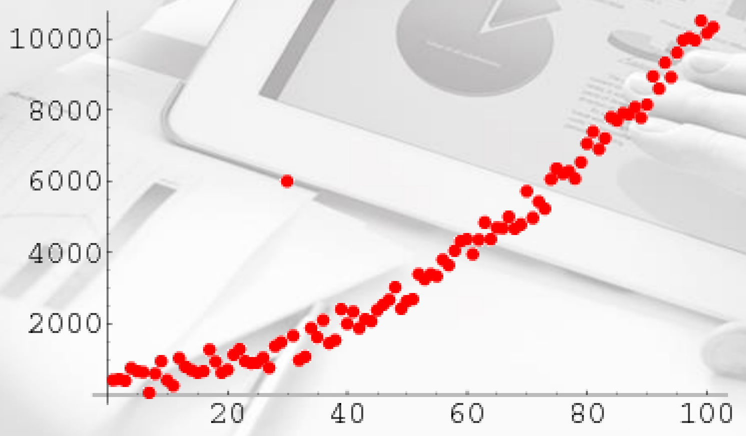
Data Cleaning > Graphs Principles > Graphs in R

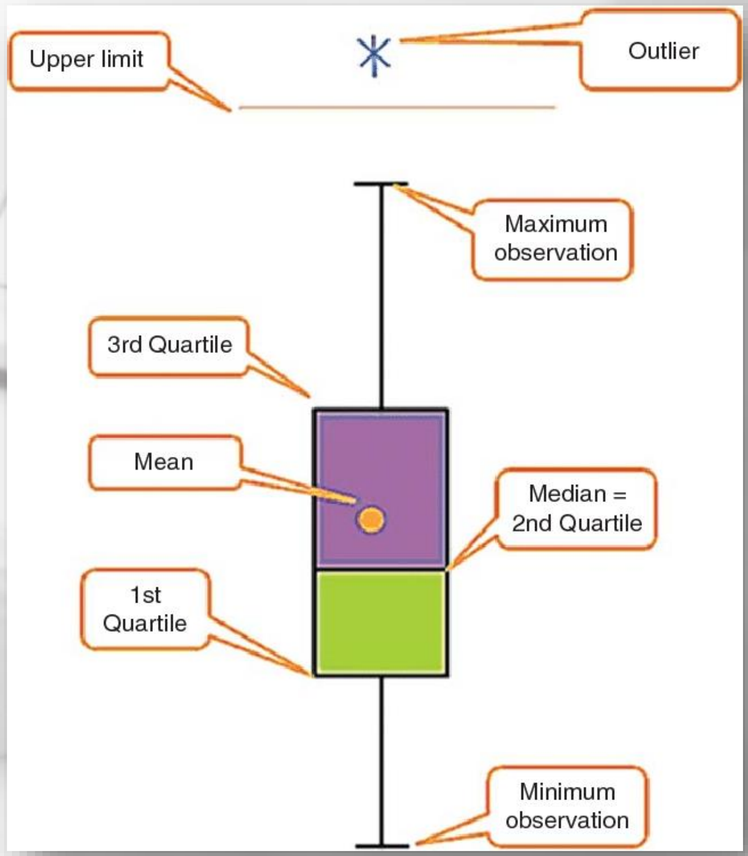
Duplicate values

```
######### Removing dublicates ###############

#--- Given the following vector:

x <- c(1, 1, 4, 5, 4, 6)

#--- To find the position of duplicate elements in x, use this:

duplicated(x)

# --- Extract duplicate elements:

x[duplicated(x)]

#--- If you want to remove duplicated elements, use !duplicated()

x[!duplicated(x)]

x <- x[!duplicated(x)]

x


#-- Following this way, you can remove duplicate rows from a data frame based on a column values, as follow:

library(tidyverse)

my_data <- as_tibble(iris)
my_data

# Remove duplicates based on Sepal.Width columns

my_data[!duplicated(my_data$Sepal.Width), ]


# --- Extract unique elements

x <- c(1, 1, 4, 5, 4, 6)

unique(x)

unique(my_data)


#--- Remove duplicate rows in a data frame


my_data %>% distinct()

my_data %>% distinct(Sepal.Length, .keep_all = TRUE)
```

Outliers

In statistics, an **outlier** is a data point that differs significantly from other observations.
An outlier may be due to variability in the measurement or it may indicate experimental error. An outlier can cause serious problems in statistical analyses.





Data Cleaning > Graphs Principles > Graphs in R

Outliers

In this part, we use mtcars dataset. It comes with the base package, so no need to import anything)

mtcars {datasets} R Documentation

Motor Trend Car Road Tests

Description

The data was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Usage

```
mtcars
```

Format

A data frame with 32 observations on 11 (numeric) variables.

[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs)
[, 7]	qsec	1/4 mile time
[, 8]	vs	Engine (0 = V-shaped, 1 = straight)
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

Data Cleaning > Graphs Principles > Graphs in R

Outliers

```r

#########  Removing Outliers ###############

# First of all, we insert a couple of outliers to the $disp column of the mtcars dataset
# (mtcars comes with the base package, so no need to import anything)
# In order to have a couple of outliers in this dataset, we simply multiply the values in mtcars$disp that are higher than 420 by *2

mtcars$disp[which(mtcars$disp >420)] <- c(mtcars$disp[which(mtcars$disp >420)]*2)

# (This is just a random way of inserting a couple of outlier values, you could also assign a couple of high values in a milion different

# Now we have a look at $disp column of the mtcars dataset with boxplot

boxplot(mtcars$disp)

# You can get the actual values of the outliers with this

boxplot(mtcars$disp)$out

# Now you can assign the outlier values into a vector

outliers <- boxplot(mtcars$disp, plot=FALSE)$out

# Check the results

print(outliers)


###### Removing the outliers

mtcars[which(mtcars$disp %in% outliers),]


# Now you can remove the rows containing the outliers, one possible option is:

mtcars <- mtcars[-which(mtcars$disp %in% outliers),]

# If you check now with boxplot, you will notice that those pesky outliers are gone

boxplot(mtcars$disp)

```

Lab Activity:

Data cleaning the Singapore Airbnb dataset

You can download the raw data in the shared folder :

/Session 3/data

Source:

https://www.kaggle.com/jojoker/singapore-airbnb

Lab Activity:

Data cleaning the Singapore Airbnb dataset

idroom id
nameroom names
host_idhost id
host_namehost names
neighbourhood_groupSingapore regions
neighbourhoodspecific place
latitudelatitude
longitudelongtitude
room_typeroom type

pricesingapore dollar per night
minimum_nightsminimum nights
number_of_reviewsnumber of review
last_reviewlast review
reviews_per_monthI don't know exaclty
calculated_host_listings_counttotal room or house in host catalog on Airbnb
availability_365availability

Lab Activity #1:

Data cleaning the Singapore Airbnb dataset

Apply all required data cleaning techniques that you learned in this session to clean the Singapore Airbnb dataset.
Then submit the following files:

- your source code (name: Session3-[your name]) in R
- The cleaned excel file

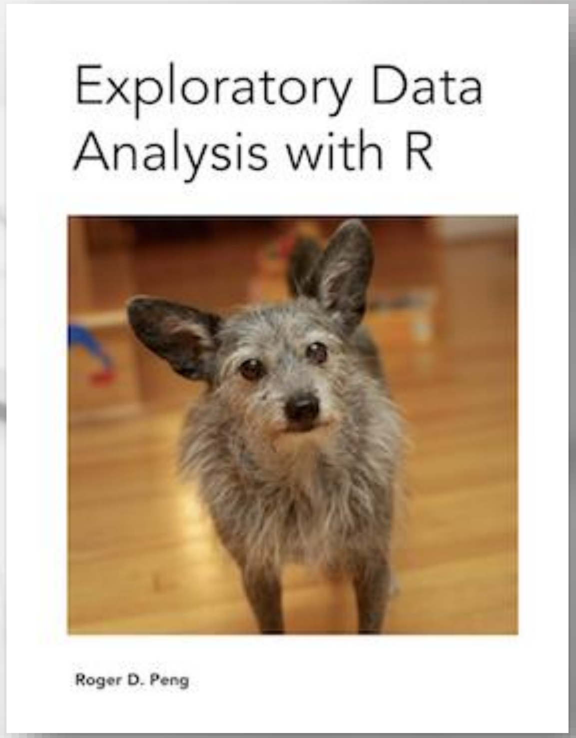
Some useful sources for further reading:

1. Exploratory Data Analysis with R

https://bookdown.org/rdpeng/exdata/

2. Programming with R

https://swcarpentry.github.io/r-novice-inflammation/



Any questions?

Vala@data-corner.com