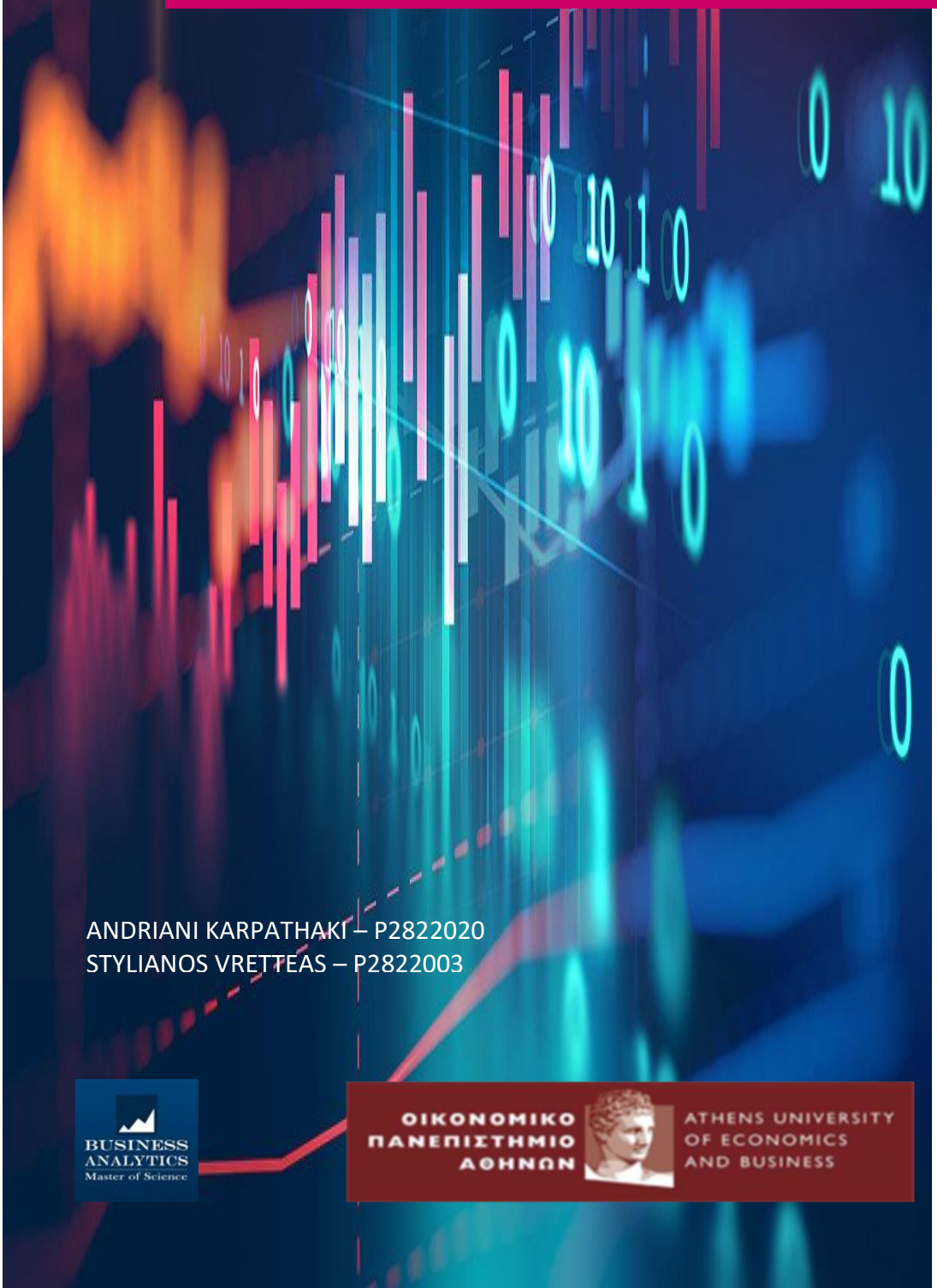
# BIG DATA SYSTEMS AND ARCHITECTURES

2020-2021 PART TIME

## REDIS AND MONGO ASSIGNMENT



ANDRIANI KARPATHAKI – P2822020
STYLIANOS VRETTEAS – P2822003

BUSINESS ANALYTICS
Master of Science

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

# Contents

# Tasks

## Task 1

For the scope of this report, we worked on R programming language. First thing to do was to install the "redux" package in order to connect to Redis through R.

```
#install.packages("redux")

library("redux")
r <- redux::hiredis(
  redux::redis_config(
    host = "127.0.0.1",
    port = "6379"))
```

### 1.1 How many users modified their listing in January?

```
January<- modified_listings[which(modified_listings$MonthID==1),-2]
February<- modified_listings[which(modified_listings$MonthID==2),-2]
March <- modified_listings[which(modified_listings$MonthID==3),-2]

for (i in 1:length(January$UserID)){
     r$SETBIT("January_modified",January[i,1],January[i,2])
  }

r$BITCOUNT("January_modified") # 9969
```

Firstly, we created three subsets of the "modified_listings" data set, one for users modifying their listing on January, one for users modifying their listing in February and one on March.
In order to figure how many users modified their listing in January, we have created a bitmap for January subset and fill it with "SETBIT" command. In this way the bitmap has for "key" the UserID and as value either "0", indicating that the user has not modified his/her listing, or "1" indicating that the user has modified his/her listing.

Lastly, with "BITCOUNT" command we count the number of "1"s in the bitmap, which represents the number of users modifying their list in January and is equal to 9969.

### 1.2 How many users did NOT modify their listing on January?

```
r$BITOP("NOT","January_not_modified","January_modified")
r$BITCOUNT("January_not_modified") # 10031

length(January$UserID)
table(January$ModifiedListing)
```

In this case we are interested in counting the "0"s of the January bitmap as it represents the number of users not modifying their list in January. Thus, the "BITOP" command was used with "NOT" as argument in order to invert the bits value and store it to another bitmap. As before by using the "BITCOUNT" command we see that 10.031 users did not modify their listing in January.

It is worth mentioning that the "BITCOUNT" command calculates one extra unit regarding users not modifying their listing. This is obvious by creating a table with frequencies of "0" and "1" for January subset. The reason why this happens is that all BITOP operations happen at byte-level increments so the system by default if it is missing a bit in order to complete a byte element of 8 bits, auto increments it. In our case we have actually 10.030 users, but the system does a rounding by itself in order to complete the bytes structure and returns 10.031.

### 1.3 How many users received at least one e-mail per month (at least one e-mail in January and at least one e-mail in February and at least one e-mail in March)?

For this task we used the "emails_sent" data set. At first, we created a new data set which shows the number of emails received by each user for each month. To be noted that the months during which a user has not received any email are not included.

Afterwards, three subsets were created from the new data set, one for each month. These were later used for creating bitmaps. Taking into consideration that if a client opened at least one of the e-mails that she/he received in the same month then we classify this client as having opened this month's newsletter, we assign to every key the value "1".

```
require(tidyr)
pivot13<- emails_sent %>%
  select(UserID, MonthID, EmailID) %>%
  group_by(UserID,MonthID) %>%
  summarise(Count_EmailID = n())
pivot13 <- as.data.frame(pivot13)
Jan_rec <- subset(pivot13, MonthID==1, c(1,3))
Feb_rec <- subset(pivot13, MonthID==2, c(1,3))
Mar_rec <- subset(pivot13, MonthID==3, c(1,3))

for (i in 1:nrow(Jan_rec)){
  r$SETBIT("EmailsJanuary",Jan_rec[i,1], "1")
}
for (i in 1:nrow(Feb_rec)){
  r$SETBIT("EmailsFebruary",Feb_rec[i,1],  "1")
}
for (i in 1:nrow(Mar_rec)){
  r$SETBIT("EmailsMarch",Mar_rec[i,1], "1")
}
r$BITOP("AND","Total",c("EmailsJanuary","EmailsFebruary","EmailsMarch"))
r$BITCOUNT("Total") # 2668
```

At this stage we use the “BITOP” command with “AND” as argument in order to find the intersection between the three bitmaps. Finally, the “BITCOUNT” command indicates that 2.668 users received at least one e-mail per month.

### 1.4 How many users received an e-mail in January and March but NOT in February?

We conducted the “BITOP” command with “NOT” as argument in order to reverse the EmailsFebruary bitmap and afterwards we found the intersection between EmailsJanuary and EmailsMarch bitmap.

```
r$BITOP("NOT","February_not_rec","EmailsFebruary")
r$BITCOUNT("February_not_rec")
r$BITCOUNT("EmailsMarch")
r$BITCOUNT("EmailsJanuary")


r$BITOP("AND","Jan_Mar",c("EmailsJanuary","EmailsMarch"))

r$BITOP("AND","Jan_Mar_not_Feb",c("Jan_Mar","February_not_rec"))
r$BITCOUNT("Jan_Mar_not_Feb") # 2417
```

The bitmaps produced by the above operations are included as input in “BITOP” command with “AND” as argument so as to calculate the total number of users that received an e-mail in January and March but NOT in February. The output is equal to 2.417 users.

### 1.5 How many users received an e-mail in January that they did not open but they updated their listing anyway?

For this task we took a subset of the “emails_sent” data set only for January period. Next, we grouped by “UserID” and calculated the sum of emails opened. We were interested only for the values that are equal to zero as this means that the user has not opened any email during the month. Given that, we assigned for each value other than zero the value “0” and zero values are replaced with value “1”.

From the object created above we constructed the bitmap showing the users who did not open their emails and we found its intersection with the JanuaryModified bitmap created in Task 1.1.

By using the “BITCOUNT” command we see that 1.961 users received an e-mail on January that they did not open but they updated their listing anyway.

```
EmailsGenJanuary <- subset(emails_sent, MonthID==1,c(UserID,EmailOpened))

library(dplyr)
library(tidyverse)

EmailsGenJanuary <- EmailsGenJanuary %>%
  select(UserID,EmailOpened) %>%
  group_by(UserID) %>%
  summarise(Total_Emails_Opened = sum(EmailOpened))

EmailsGenJanuary <- as.data.frame(EmailsGenJanuary)

EmailsGenJan <- data.frame()

for (i in 1:length(EmailsGenJanuary$UserID)){
  if(EmailsGenJanuary[i,2]!=0){EmailsGenJan[i,1]=EmailsGenJanuary[i,1];
  EmailsGenJan[i,2]=0}
  else{EmailsGenJan[i,1]=EmailsGenJanuary[i,1]; EmailsGenJan[i,2]=1}
}

for (i in 1:nrow(EmailsGenJan)){
  r$SETBIT("NotOpenedJan",EmailsGenJan[i,1],EmailsGenJan[i,2])
}

r$BITCOUNT("NotOpenedJan")
# Intercection
r$BITOP("AND","EmailsRecJanNotModified",c("January_modified",
"NotOpenedJan"))
r$BITCOUNT("EmailsRecJanNotModified") # 1961
```

### 1.6 How many users received an e-mail in January that they did not open but they updated their listing anyway on January OR they received an e-mail in February that they did not open but they updated their listing anyway in February OR they received an e-mail in March that they did not open but they updated their listing anyway in March?

For this task we followed the same procedure as we did in the previous task but for periods February and March, respectively. The only difference is that we calculated the bitmaps February_modified and March_modified that had not been created before.

At the end we calculated the union between the three bitmaps by using the “BITOP OR” command. The output shows that the number of users that received an e-mail on January that they did not open but they updated their listing anyway in January or they received an e-mail in February that they did not open but they updated their listing anyway in February or they received an e-mail in March that they did not open but they updated their listing anyway in March is equal to 5.249.

```
#---February
EmailsGenFebruary <- subset(emails_sent, MonthID==2,
c(UserID,EmailOpened))

library(dplyr)
library(tidyverse)

EmailsGenFebruary <- EmailsGenFebruary %>%
  select(UserID,EmailOpened) %>%
  group_by(UserID) %>%
  summarise(Total_Emails_Opened = sum(EmailOpened))

EmailsGenFebruary <- as.data.frame(EmailsGenFebruary)

EmailsGenFeb <- data.frame()
for (i in 1:length(EmailsGenFebruary$UserID)){
  if(EmailsGenFebruary[i,2]!=0){EmailsGenFeb[i,1]=EmailsGenFebruary[i,1];
      EmailsGenFeb[i,2]=0}
  else{EmailsGenFeb[i,1]=EmailsGenFebruary[i,1]; EmailsGenFeb[i,2]=1}
}


for (i in 1:nrow(EmailsGenFeb)){
  r$SETBIT("NotOpenedFeb",EmailsGenFeb[i,1],EmailsGenFeb[i,2])
}

r$BITCOUNT("NotOpenedFeb")

# February Modified

for (i in 1:length(February$UserID)){
  r$SETBIT("February_modified",February[i,1],February[i,2])
}

r$BITCOUNT("February_modified")
# Intercection
r$BITOP("AND","EmailsRecFebNotModified",c("February_modified",
"NotOpenedFeb"))
r$BITCOUNT("EmailsRecFebNotModified") #1971

# ---March
EmailsGenMarch <- subset(emails_sent, MonthID==3, c(UserID,EmailOpened))

library(dplyr)
library(tidyverse)

EmailsGenMarch <- EmailsGenMarch %>%
  select(UserID,EmailOpened) %>%
  group_by(UserID) %>%
  summarise(Total_Emails_Opened = sum(EmailOpened))

EmailsGenMarch <- as.data.frame(EmailsGenMarch)

EmailsGenMar <- data.frame()
for (i in 1:length(EmailsGenMarch$UserID)){
  if(EmailsGenMarch[i,2]!=0){EmailsGenMar[i,1]=EmailsGenMarch[i,1];
      EmailsGenMar[i,2]=0}
  else{EmailsGenMar[i,1]=EmailsGenMarch[i,1]; EmailsGenMar[i,2]=1}
}
```

```
for (i in 1:nrow(EmailsGenMar)){
  r$SETBIT("NotOpenedMar",EmailsGenMar[i,1],EmailsGenMar[i,2])
}

r$BITCOUNT("NotOpenedMar")

# March Modified

for (i in 1:length(March$UserID)){
  r$SETBIT("March_modified",March[i,1],March[i,2])
}

r$BITCOUNT("March_modified")
# Intercection
r$BITOP("AND","EmailsRecMarNotModified",
c("March_modified","NotOpenedMar"))
r$BITCOUNT("EmailsRecMarNotModified") #1966

r$BITOP("OR","Jan_Feb_Mar",
c("EmailsRecJanNotModified","EmailsRecFebNotModified",
"EmailsRecMarNotModified"))
r$BITCOUNT("Jan_Feb_Mar") # 5249
```

### 1.7 Does it make any sense to keep sending e-mails with recommendations to sellers? Does this strategy really work? How would you describe this in terms a business person would understand?

From the above analysis and tasks , we can conclude the following:

A total of 40.148 emails were sent from the company to the respective number of 16.006 users during the January - March period. Specifically, 13.459 emails were sent in January, 13.535 emails were sent in February, 13.154 emails were sent in March.

Cumulatively, 20.009 emails were not opened at all in contrast to 20.139 emails that were read. This is an approximately 50% percent ratio of not opened emails .Thus, we shall conclude that 1 of 2 emails that were sent from the company was not opened.

Regarding the users, a total number of 2.668 received at least one email per month and we can see from the data that the ratio of the users that have not opened their newsletter at least one month but still made modifications (for this month) in their selling approach was around 32% - 33%. In addition, almost 45 % of users (7190) opened their email and also made recommended modifications at least one month.

Subsequently, since the percentage of users who made changes and opened their emails is greater than the percentage of users who modified without checking their newsletter (37% difference) the emails feedback process should remain as it is.

**1.8 Do the previous subtasks again by using any type of relational or non-relational database. Compare the complexity of the solutions. Then benchmark the query execution time for the dataset that you have. At last, boost the number of entries to 1 billion rows (create your own dummy entries). Perform the benchmark again.**

At this point we do the same tasks as before but in this case using MySQL relational database. The queries used come as follows.

```
CREATE DATABASE IF NOT EXISTS redis;
use redis;

-- Q1--
SELECT count(UserID) as No_of_users_Jan_modified from
modified_listings where MonthID = 1 AND ModifiedListing = 1;
-- LIMIT 0, 1000       1 row(s) returned      0.078 sec / 0.000 sec

-- Q2 --
SELECT count(UserID) as No_of_users_Jan_modified from
modified_listings where MonthID = 1 AND ModifiedListing = 0;
-- LIMIT 0, 1000       1 row(s) returned      0.046 sec / 0.000 sec


-- Q3 --

select count(Emails_Jan) as Intersection_All_months_emails_UserId
from
(select Emails_Jan
from (SELECT distinct(UserID) as Emails_Jan from emails_sent where
MonthID = 1) as t1
INNER JOIN (SELECT distinct(UserID) as Emails_Feb from emails_sent
where MonthID = 2) as t2 ON t1.Emails_Jan = t2.Emails_Feb) as t3
INNER JOIN (SELECT distinct(UserID) as Emails_Mar from emails_sent
where MonthID = 3) as t4 on t3.Emails_Jan = t4.Emails_Mar;
-- LIMIT 0, 1000       1 row(s) returned      0.125 sec / 0.000 sec

-- Q4 --
select COUNT(UserID)
FROM
(select *
from
(select t1.UserID
from (SELECT distinct(UserID) from emails_sent where MonthID = 1) as
t1
INNER JOIN (SELECT distinct(UserID) from emails_sent where MonthID =
3) as t2 ON t1.UserID = t2.UserID) as t3
LEFT JOIN (SELECT distinct(UserID) AS FEB from emails_sent where
MonthID = 2) as t4 on t3.UserID = t4.FEB) as t5
where t5.FEB IS NULL;
-- LIMIT 0, 1000       1 row(s) returned      0.141 sec / 0.000 sec
```

```sql
-- Q5 --
select count(*)
from
(select UserID FROM modified_listings where MonthID=1 AND
ModifiedListing=1) as t3
inner join
(select UserID from (SELECT UserID, sum(EmailOpened) as Sum from
emails_sent where MonthID = 1 group by UserID) as t1 where t1.Sum =
0) as t2
on t2.UserID = t3.UserID ;
-- LIMIT 0, 1000       1 row(s) returned      0.094 sec / 0.000 sec

-- Q6 --
CREATE VIEW JAN AS
select t3.UserID
from
(select UserID FROM modified_listings where MonthID=1 AND
ModifiedListing=1) as t3
inner join
(select UserID from (SELECT UserID, sum(EmailOpened) as Sum from
emails_sent where MonthID = 1 group by UserID) as t1 where t1.Sum =
0) as t2
on t2.UserID = t3.UserID ;

CREATE VIEW FEB AS
select t3.UserID
from
(select UserID FROM modified_listings where MonthID=2 AND
ModifiedListing=1) as t3
inner join
(select UserID from (SELECT UserID, sum(EmailOpened) as Sum from
emails_sent where MonthID = 2 group by UserID) as t1 where t1.Sum =
0) as t2
on t2.UserID = t3.UserID;

CREATE VIEW MAR AS
select t3.UserID
from
(select UserID FROM modified_listings where MonthID=3 AND
ModifiedListing=1) as t3
inner join
(select UserID from (SELECT UserID, sum(EmailOpened) as Sum from
emails_sent where MonthID = 3 group by UserID) as t1 where t1.Sum =
0) as t2
on t2.UserID = t3.UserID ;

select count(*)
from
(select t1.UserID
FROM
(SELECT UserID FROM JAN
UNION
SELECT UserID FROM FEB) AS t1
UNION
select UserID FROM MAR
ORDER BY UserID) as t2;
-- LIMIT 0, 1000       1 row(s) returned      0.344 sec / 0.000 sec
```

The results provided by MySQL verify those provided by Redis. Generally, regarding processing massive amounts of data preferable tool is Redis from any other relational database because it is storing data in primary memory, meaning little time execution.

Although time execution of queries in MySQL was not significant for the provided data, it would need much more time for the database to execute the queries if the rows were 1 billion.

# Task 2

## 2.1 Add your data to MongoDB.

In this task we have been provided with data, split in many files (29.701 json files), regarding bike advertisement scraped from "car.gr" site. First of all, in order to read and import the data into R, for further data manipulation, we have used the command "`PS C:\Users\svret> dir -Recurse -Name -File > files_list.txt` in Windows PowerShell. Thus, a text file has been created, containing every path for each json file stored locally.

Before importing the data into MongoDB, we used R programming language as a staging environment in order to proceed with basic data cleansing process. Then, we created a collection called "bikes1" in which we inserted the data and we established a connection to MongoDB.

As presented below, the "files_list" object stands for the file containing the path for the split data. After loading the file, the two first lines have been removed because they were not a path to json file and slashes have been converted appropriately for being used as an argument in R command.

```
setwd("C:/Users/svret/BIKES_DATASET/") # set directory
# read the file with the paths
files_list<- read.csv(file.choose(),sep = '\n', header = FALSE)
files_list<- files_list[-c(1,2),] # remove first two lines
files_list<- as.data.frame(files_list) # make dataframe
# string manipulation
files_list[,1] <- gsub('\\', '//', files_list[,1], fixed = TRUE)
files_list[,1] <- gsub('//', '/', files_list[,1], fixed = TRUE)
head(files_list) # ok for reading
```

Before starting the cleaning process, we have created a data frame containing only the data needed for performing other tasks and take a look of its structure.

```
# List with all json files
y<- list()
for (i in 1:nrow(files_list)) {
  y[[i]] <- fromJSON(readLines(files_list[i,]))
}
```

```
# make empty vectors
ad_id <- c()
price <- c()
category<- c()
registration<- c()
mileage<- c()
type<- c()
brand<- c()
model<- c()
# check the column we want for the project
for (i in 1:length(y)){
  if(is.null(y[[i]]$ad_id)){mileage[i] <- "NA"}
  else {mileage[i]<- y[[i]]$ad_id} # ad_id
}

for (i in 1:length(y)){
  if(is.null(y[[i]]$ad_data$Price)){Price[i] <- "NA"}
  else {price[i]<- y[[i]]$ad_data$Price} # Price
}

for (i in 1:length(y)){
  if(is.null(y[[i]]$ad_data$Category)){Category[i] <- "NA"}
  else {category[i]<- y[[i]]$ad_data$Category} # category
}

for (i in 1:length(y)){
  if(is.null(y[[i]]$ad_data$Registration)){registration[i]<-"NA"}
  else {registration[i]<-y[[i]]$ad_data$Registration} # registration
}
for (i in 1:length(y)){
  if(is.null(y[[i]]$ad_data$Mileage)){mileage[i] <- "NA"}
  else {mileage[i]<- y[[i]]$ad_data$Mileage} # Mileage
}

for (i in 1:length(y)){
  if(is.null(y[[i]]$metadata$brand)){brand[i]<-"NA"}
  else{brand[i]<-y[[i]]$metadata$brand} # brand
}

for (i in 1:length(y)){
  if(is.null(y[[i]]$metadata$model)){model[i]<-"NA"}
  else{model[i]<-y[[i]]$metadata$model} # model
}

# unlist
ad_id1 <- unlist(ad_id, use.names=FALSE)
price1 <- unlist(price, use.names=FALSE)
category1 <- unlist(category, use.names=FALSE)
registration1 <- unlist(registration, use.names=FALSE)
mileage1<- unlist(mileage, use.names=FALSE)
brand1 <- unlist(brand, use.names=FALSE)
model1 <- unlist(model, use.names=FALSE)
# create df in order to check the data
df <-
cbind(ad_id1,price1,category1,registration1,mileage1,brand1,model1)
df <- as.data.frame(df)
```

At a first glance, we noticed that the “Price” column was not set appropriately as it contained some irrelevant special characters. In addition to that, some fields did not have a price value but an “Askforprice” string (Figure. 1).

| | price1 | category1 | registration1 | mileage1 | brand1 | model1 |
|---|---|---|---|---|---|---|
| 1 | β,¬10 | Bike - Naked | 10 / 1992 | 25,000 km | Jawa | Jawa 350CC 634-638-640 '92 - β,¬ 10 EUR |
| 2 | β,¬2.300 | Bike - Roller/Scooter | 4 / 2011 | 59,000 km | Piaggio | Beverly 300i |
| 3 | Askforprice | Bike - Roller/Scooter | 1 / 2019 | NA | Kymco | Agility CITY 150 |
| 4 | β,¬3.000 | Bike - Underbone | 10 / 1985 | 7,000 km | Honda | C |
| 5 | β,¬6.500 | Bike - Naked | 1 / 2005 | 30,000 km | KTM | KTM SUPER DUKE 990 '05 - β,¬ 6.500 EUR |
| 6 | β,¬5.490 | Bike - On/Off | 3 / 1990 | 109,000 km | Bmw | R 100 GS |
| 7 | β,¬3.350 | Bike - Ξ•nduro | 3 / 2008 | 100 km | KTM | 450 EXC |
| 8 | β,¬1.350 | Bike - Roller/Scooter | 7 / 2004 | 7,039 km | Kymco | B & W 250 |
| 9 | β,¬6.000 | Bike - Cafe Racer | 1 / 1979 | 30 km | Honda | CB 400 |
| 10 | β,¬1.200 | Bike - Super Motard | 1 / 2000 | 13,000 km | Yamaha | TTR |
| 11 | β,¬2.200 | Bike - Roller/Scooter | 4 / 2003 | 28,500 km | Yamaha | T-MAX 500 |
| 12 | β,¬2.200 | Bike - Roller/Scooter | 11 / 2010 | 38,000 km | Kymco | Kymco Downtown 300 ABS '10 - β,¬ 2.200 EUR |
| 13 | β,¬3.000 | Bike - Naked | 9 / 2007 | 33,000 km | Suzuki | GSR 600 |

*Figure 1 - Structure of some of the data*

Regarding the "Registration" column which contained date values, the primary format was "mm/yyyy" as character. Given that we were only interested in the year we removed the month information at a later point. The "Mileage" column also seemed to need some changes with reference to its final format.

All actions required for cleaning the data have been incorporated into one function called "clean". As said previously, we chose to modify three columns "Price", "Mileage" and "Registration".

```
# function for cleaning
clean <- function(x) {
  # Price
  if (x$ad_data$Price == "Askforprice") { x$ad_data$Price <- NA}
  else {x$ad_data$Price <- gsub("β,¬","", x$ad_data$Price);
  x$ad_data$Price <- as.numeric(gsub("[\200.]","",x$ad_data$Price))};
  # Mileage
  if (is.null(x$ad_data$Mileage)) { x$ad_data$Mileage <- NA}
  x$ad_data$Mileage <- gsub("km","", x$ad_data$Mileage);
  x$ad_data$Mileage <- as.numeric(gsub(",","", x$ad_data$Mileage));
  # Registration
  year <- 2021
  x$ad_data$Registration <-
as.numeric(str_sub(x$ad_data$Registration,-4,-1));
  x$ad_data$age <- year - x$ad_data$Registration
  return(x)
}
```

To summarize, referring to "Price" column, we replaced the "AskforPrice" value with NA value. Then, we kept a subset of the fields, price only, and convert it to numeric. Regarding the "Mileage" column we had noticed that there were many Null values so we replaced them with NA values. Afterwards, we removed the "km" string, replaced

the "," character with "." and convert it to numeric. Finally, we kept only the year of the "Registration" column and created a new column called "age" indicating the difference in years up to today.

At this point, we are ready to import all data into MongoDB. As seen below, every line of the "files_list" file which corresponds to a different json file, is read and assigned to "x" as a list object. After, applying the clean function to the data we convert them again to json files and store them all in one vector.

```
# prepare final_data
final_data <- c()

for (i in 1:nrow(files_list)) {
x <- fromJSON(readLines(files_list[i,], warn=FALSE, encoding ="UTF-
8"))
x <- clean(x)
j <- toJSON(x, auto_unbox = TRUE)
final_data <- c(final_data, j)
}
# Open a connection to MongoDB
library("mongolite")
m <- mongo(collection = "bikes1", db = "mydb", url =
"mongodb://localhost")
m$remove('{}')
# insert the data in mongo db
m$insert(final_data)
```

A view of the data in MongoDB is presented in the following figure (Figure. 2).

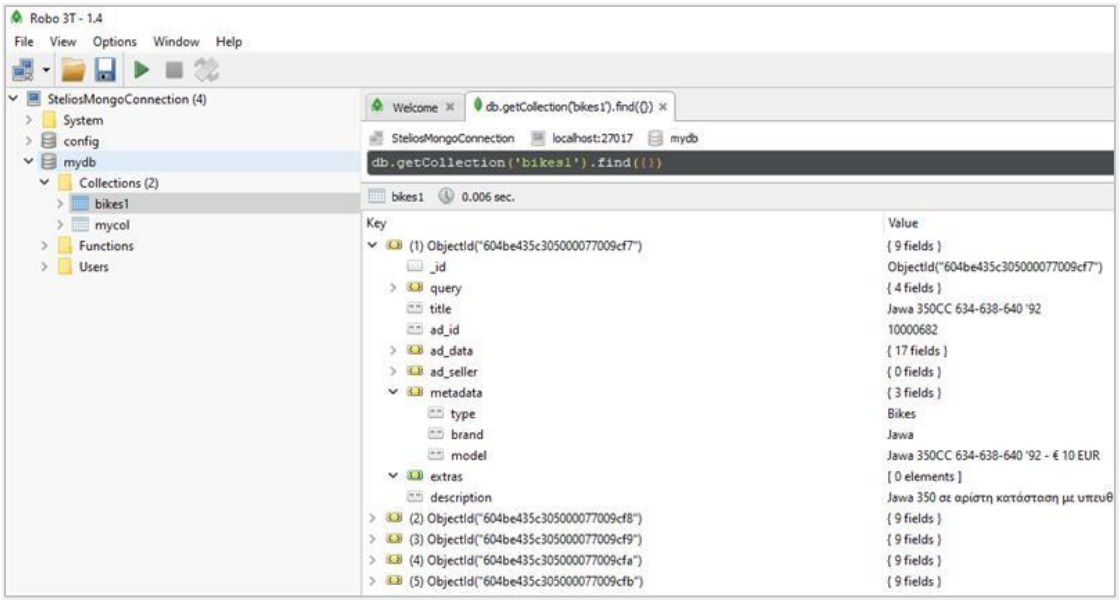



*Figure 2 - Data in MongoDB*

## 2.2 How many bikes are there for sale?

```
m$count()
```

```
> m$count() # 29701
[1] 29701
```

*Figure 3 - Output for Task 2.2*

The number of bikes is equal to 29.701, as the total number of lists provided.

## 2.3 What is the average price of a motorcycle (give a number)? What is the number of listings that were used in order to calculate this average (give a number as well)? Is the number of listings used the same as the answer in 2.2? Why?

To be noted that for this task only values greater than 100 have been taken into account as it has been noticed that some values were extremely small, hence not representative for bikes price.

```
m$aggregate(
'[{
"$match": {
"ad_data.Price": {
"$gt": 100
}
}
},
{
"$group": {
"_id": null,
"AvgPrice": {
"$avg": "$ad_data.Price"
},
"count": {
"$sum": 1
}
}
}]')
```

```
  _id AvgPrice count
1  NA 3033.611 28461
>
```

*Figure 4 - Output for Task 2.3*

The average number of price is equal to € 3.033,61. The total number of lists used is 28.461, different than the findings of Task 2.2 which is rational considering NA values and those excluded (<100).

## 2.4 What is the maximum and minimum price of a motorcycle currently available in the market?

```
# maximum
m$aggregate(
'[{
"$group": {
"_id": null,
"Max_Price": {
"$max": "$ad_data.Price"
},
"count": {
"$max": 1
}
}
}]')
```

```
  _id Max_Price count
1  NA     89000     1
```

*Figure 5 - Output for Task 2.4 (Maximum)*

The maximum price of a motorcycle is equal to € 89.000. Although not being a common case, there are indeed such prices regarding specific category of bikes.

```
# minimum above 100
m$aggregate(
'[{
"$match": {
"ad_data.Price": {
"$gt": 100
}
}
},
{
"$group": {
"_id": null,
"MinPrice": {
"$min": "$ad_data.Price"
},
"count": {
"$sum": 1
}
}
}]')
```

```
  _id MinPrice count
1  NA      101 28461
>
```

*Figure 6 - Output for Task 2.4 (Minimum)*

The minimum price of a motorcycle is equal to € 101 regarding values that are greater than 100. The result is considered to be logical for second handed bikes which are not in a very good situation or have damage.

### 2.5 How many listings have a price that is identified as negotiable?

```
m$aggregate(
'[{
"$match": {
"metadata.model": {
"$regex": "Negotiable",
"$options": "i"
}
}
},
{
"$group": {
"_id": null,
"count": {
"$sum": 1
}
}
}]')
```

```
  _id count
1  NA  1348
```

*Figure 7 - Output for Task 2.5*

The number of listings that have a price that is identified as negotiable is equal to 1.348.

### 2.7 (Optional) What is the motorcycle brand with the highest average price?

```
m$aggregate( '[{
"$group": { "_id": "$metadata.brand",
"AveragePrice": { "$avg": "$ad_data.Price" }
} },
{"$sort" : { "AveragePrice": -1 } },
{ "$limit" : 1 }
]')
```

```
    _id AveragePrice
1 Semog        15600
```

*Figure 8 - Output for Task 2.7*

The motorcycle brand with the highest average price is equal to € 15.600.

### 2.8 (Optional) What are the TOP 10 models with the highest average age? (Round age by one decimal number)

```
m$aggregate
( '[{ "$group": {
         "_id":"$metadata.brand",
         "AverageAge":{"$avg":"$ad_data.age" }
         }
        },
      {"$sort":
      { "AverageAge": -1}
        },
     {"$limit":10},
     {"$project":{
        "AverageAge":{
        "$round":["$AverageAge", 1 ]}
}
}]')
```

| | _id | AverageAge |
|---|---|---|
| 1 | Bsa | 72.9 |
| 2 | Norton | 69.8 |
| 3 | Horex | 68.7 |
| 4 | Victoria | 68.0 |
| 5 | Nsu | 65.7 |
| 6 | Adler | 65.0 |
| 7 | Heinkel | 61.8 |
| 8 | Kuberg | 61.0 |
| 9 | Dkw | 60.4 |
| 10 | Maico | 57.5 |

*Figure 9 - Output for Task 2.8*

The top 10 models with the highest Average age are presented in Figure 9.

### 2.9 (Optional) How many bikes have “ABS” as an extra?

```
m$aggregate(
'[{
"$match": {
"extras": {
"$regex": "ABS",
"$options": "i"
}
}
},
{
"$group": {
"_id": null,
"count": {
"$sum": 1
}
}}]')
```

```
  _id count
1  NA  4025
```

*Figure 10 - Output for Task 2.9*

The number of bikes that have "ABS" as an extra is equal to 4.025.

### 2.10 (Optional) What is the average Mileage of bikes that have "ABS" AND "Led lights" as an extra?

```
m$aggregate(
'[{
"$match": {
"extras": {
"$all" : ["ABS", "Led lights"]
}
}
},
{
"$group": {
"_id": null,
"AverageMilage": {
"$avg": "$ad_data.Mileage"
}
}
}]')
```

```
  _id AverageMilage
1  NA       30125.7
```

*Figure 11 - Output for Task 2.10*

The average Mileage of bikes that have "ABS" and "Led lights" as an extra is equal to 30.125,7 km.

*To be noted that we tried all Optional Tasks but we could not execute three of them 2.6, 2.11, 2.12*