

**ΟΙΚΟΝΟΜΙΚΟ  
ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΑΘΗΝΩΝ**



**ATHENS UNIVERSITY  
OF ECONOMICS  
AND BUSINESS**

**SCHOOL OF BUSINESS**

**DEPARTMENT OF MANAGEMENT SCIENCE & TECHNOLOGY**

**ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS**

**ACADEMIC YEAR OF 2021-2022**

**BIG DATA SYSTEMS & ARCHITECTURES**

**Author:**

**Konstantinos Ninas (f2822108)**

**REDIS & MONGODB ASSIGNMENT**

**Supervising Professor: Spiros Safras**



## PART A

### Scenario

We are data analysts at a consulting firm and we have access to a dataset of ~30K classified ads from the used motorcycles market. We also have access to some seller related actions that have been tracked in the previous months. The data present past activity of the firm and its salesmen (January, February and March of 2021). We are asked to create several programs/queries to provide our firm with some critical information about its strategy's effectiveness.

Specifically, at the end of each month the firm sends to some of its salesmen emails with recommendations aiming to help to increase their listings. Due to technical issues, at times, more than one mails may be sent to a salesman in a month. It should be mentioned that many salesmen do not open their recommendation emails, which is an information that the firm is tracking. An additional file is provided that shows for each salesman and for each month recorded whether they updated their listing, regardless of whether they received a mail from the firm. To answer the questions asked from the firm, Redis bitmaps will be used through an R interpreter.

### Programs/Queries

The first question required the number of salesmen that updated their listing in January regardless of whether they received a recommendation email. It is noticed that 9969 salesmen received recommendation emails in January. (Figure 1)

```
> #count of salesmen that modified their list in January - 9969
> redis$BITCOUNT('ModificationsJanuary')
[1] 9969
```

Figure 1. Salesmen that modified their listing in January

The second question required the number of salesmen that did not update their listing in January. It is noticed that 10031 salesmen did not modify their listing in January. (Figure 2) It should be mentioned that the number of salesmen that did modify their listing, and that of those who didn't, do not add up to the total number of salesmen that are occupied by the firm. (Figures 1,2 & 3) This occurred because Redis bitmaps sometimes create additional bits (positions) to round up to the nearest number of bits, that is divisible by 8.

```
> #Numbers of modified and non-modified add up to the total amount of salesmen - 9969 + 10031 = 19999
> redis$BITCOUNT('NoModificationsJanuary')
[1] 10031
```

Figure 2. Salesmen that did not modify their listing in January

```
> length(unique(modified_listings$UserID))
[1] 19999
```

Figure 3. Total number of salesmen occupied by the firm

The third question required the number of salesmen that received at least one email in January, February and March. In detail, it is noticed that 9617, 9666 and 9520 users received at least one mail in January, February and March respectively. (Figure 4) The number of salesmen that received at least one mail per month is equal to 2668. (Figure 5)

```
> redis$BITCOUNT('EmailsJanuary') # 9617
[1] 9617
> redis$BITCOUNT('EmailsFebruary') # 9666
[1] 9666
> redis$BITCOUNT('EmailsMarch') # 9520
[1] 9520
```

Figure 4. Salesmen that received emails per distinct month

```
> #2668 salesmen received at least one mail in all 3 months
> redis$BITCOUNT('EmailsAllMonths')
[1] 2668
```

Figure 5. Salesmen that received emails in all 3 months

The fourth question required the number of salesmen that received at least one email in both January and March, but not in February. It is found that the number of those salesmen is equal to 2417. (Figure 6)

```
> #2417 salesmen received at least one mail in January and March but not in February
> redis$BITCOUNT('EmailsJanMarNoFeb')
[1] 2417
```

Figure 6. Salesmen that received emails in January and March, but not in February

The next question required the number of salesmen that received a mail, did not open it but updated their listing anyway in January. It is observed that the number of those salesmen is equal to 1961. (Figure 7)

```
> redis$BITCOUNT('updateAndNotOpenedJAN') # 1961 salesmen
[1] 1961
```

Figure 7. Salesmen that received a mail in January, did not open it and update their list

The sixth question required the number of salesmen that received a mail, did not open it but updated their listing in January, February or March. It is noted that the number of those salesmen is equal to 5249. (Figure 8)

```
> redis$BITCOUNT('updateAndNotOpenAllMonths')
[1] 5249
```

Figure 8. Salesmen that received a mail in any month, did not open it and update their list

The seventh question regarded the effectiveness of the firm's strategy. Specifically, the firm wanted to examine the monthly percentiles of its salesmen that opened their mails and of its salesmen that updated their listings. It is found that, in January, 59% of the firm's salesmen opened their recommendation mails. It is also worth mentioning that almost half (49%) of the salesmen that received

a recommendation mail updated their listing, regardless of the whether they opened it, while almost 30% of the salesmen that received an email and opened it, updated their listing in the same month. (Figure 9)

```
> February_open_pc <- redis$BITCOUNT('openedFebruary')/redis$BITCOUNT('EmailsFebruary')
> print(cat(paste('January: Mail opening percent -',round(January_open_pc,2), '%',
+ , paste('(',redis$BITCOUNT('openedJan'), '/',redis$BITCOUNT('EmailsJanuary') ,')',sep='')
+ , ', List update-per-mail-sent -',round(January_sentMod_pc,2), '%',
+ , paste('(',redis$BITCOUNT('JanModandSent'), '/',redis$BITCOUNT('EmailsJanuary') ,')',sep='')
+ , paste(' , List update-per-mail-opened -',round(January_update_pc,2), '%',
+ , paste('(',redis$BITCOUNT('JanModandRead'), '/',redis$BITCOUNT('EmailsJanuary') ,')',sep='')),sep="\n"))
January: Mail opening percent - 0.59 % (5645/9617) , List update-per-mail-sent - 0.49 % (4758/9617)
, List update-per-mail-opened - 0.29 % (2797/9617)
```

Figure 9. January Statistics

In February, it is observed that 59% of the firm's salesmen opened their recommendation mails, while half (50%) of those that received a mail updated their listing. Of those salesmen that received a mail and opened it, 30% updated their listing in the same month. (Figure 10)

```
> print(cat(paste('February: Mail opening percent -',round(February_open_pc,2), '%',
+ , paste('(',redis$BITCOUNT('openedFebruary'), '/',redis$BITCOUNT('EmailsFebruary') ,')',sep='')
+ , ', List update-per-mail-sent -',round(February_sentMod_pc,2), '%',
+ , paste('(',redis$BITCOUNT('FebModandSent'), '/',redis$BITCOUNT('EmailsFebruary') ,')',sep='')
+ , paste(' , List update-per-mail-opened -',round(February_update_pc,2), '%',
+ , paste('(',redis$BITCOUNT('FebModandRead'), '/',redis$BITCOUNT('EmailsFebruary') ,')',sep='')),sep=
+"\n"))
February: Mail opening percent - 0.59 % (5721/9666) , List update-per-mail-sent - 0.5 % (4845/9666)
, List update-per-mail-opened - 0.3 % (2874/9666)
```

Figure 10. February Statistics

In March, it is also observed that 59% of the firm's salesmen opened their recommendation mails, while also half of those that received a mail updated their listing. Of those salesmen that received a mail and opened it, almost 30% updated their listing in the same month. (Figure 11)

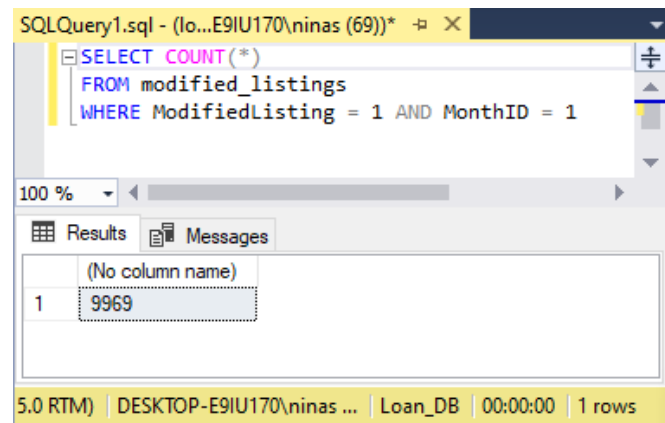
```
> print(cat(paste('March: Mail opening percent -',round(March_open_pc,2), '%',
+ , paste('(',redis$BITCOUNT('openedMarch'), '/',redis$BITCOUNT('EmailsMarch') ,')',sep='')
+ , ', List update-per-mail-sent -',round(March_sentMod_pc,2), '%',
+ , paste('(',redis$BITCOUNT('MarModandSent'), '/',redis$BITCOUNT('EmailsMarch') ,')',sep='')
+ , paste(' , List update-per-mail-opened -',round(March_update_pc,2), '%',
+ , paste('(',redis$BITCOUNT('MarModandRead'), '/',redis$BITCOUNT('EmailsMarch') ,')',sep='')),sep="\n"))
March: Mail opening percent - 0.59 % (5572/9520) , List update-per-mail-sent - 0.5 % (4749/9520)
, List update-per-mail-opened - 0.29 % (2783/9520)
```

Figure 11. March Statistics

It can be inferred that the effectiveness of the firm's strategy does not improve over time and it is relatively weak. In detail, the percent of those salesmen that updated their listing, upon receiving and opening a recommendation mail is steady around 30% for all the months. This percent can be regarded as relatively small, and the strategy cannot be regarded as effective because the percent does not increase throughout the months examined. As a result, the firm's strategy about sending those monthly emails to its salesmen should be reconsidered, because most of its salesmen change their listings based on their own criteria and experience. For example, in January, almost 10.000 salesmen updated their listing, while only 2.800 of them received a recommendation mail. A similar behavior is also observed in the rest months examined.

The final question regards the determination of the most efficient program/query language for the above questions, comparing the execution time and the coding complexity of Redis bitmaps through an R interpreter with that of a Relational Database (SQL Queries). (Figures 12, 13, 14, 15, 16 & 17)

Observing the coding structure in the relational database queries, it is obvious that as the questions are getting more advanced, the queries appear to have a more and more complex structure. In specific, it was necessary to implement slightly more advanced techniques, and larger pieces of code to answer most of the questions asked (such as creating views and using union queries and nested joins). At the same time, the same questions were answered with way less effort and less complex code through Redis bitmaps. Though, it should be mentioned that, for this scale of data (up to 60.000 observations) the response times is the same in both approaches. Unfortunately, due to the pc's low performance, there won't be conducted any replication of the observations to compare their execution times in bigger scales of data.



```
SQLQuery1.sql - (lo...E9IU170\ninas (69))* - X
```

```
SELECT COUNT(*)
FROM modified_listings
WHERE ModifiedListing = 1 AND MonthID = 1
```

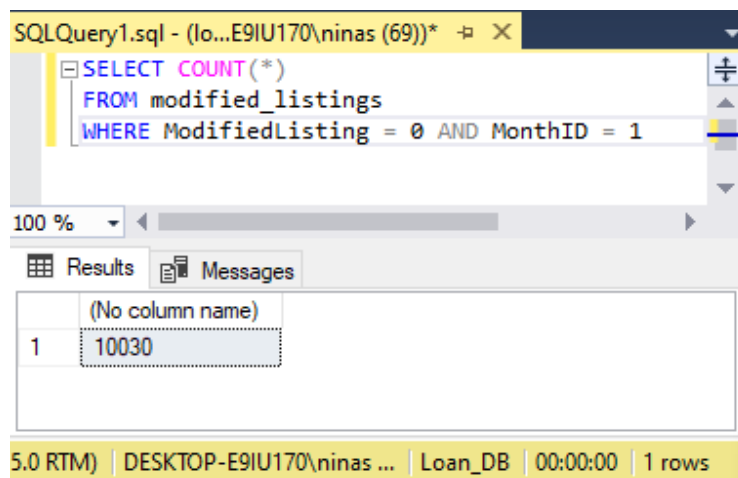
100 %

Results Messages

	(No column name)
1	9969

5.0 RTM) | DESKTOP-E9IU170\ninas ... | Loan\_DB | 00:00:00 | 1 rows

Figure 12. Question 1 – SQL Query



```
SQLQuery1.sql - (lo...E9IU170\ninas (69))* - X
```

```
SELECT COUNT(*)
FROM modified_listings
WHERE ModifiedListing = 0 AND MonthID = 1
```

100 %

Results Messages

	(No column name)
1	10030

5.0 RTM) | DESKTOP-E9IU170\ninas ... | Loan\_DB | 00:00:00 | 1 rows

Figure 13. Question 2 – SQL Query

```
SQLQuery1.sql - (lo...E9IU170\ninas (53))* -> X
CREATE VIEW Jan_Users_AS
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 1

CREATE VIEW Feb_Users_AS
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 2

CREATE VIEW Mar_Users_AS
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 3

SELECT COUNT(*)
FROM Mar_Users T0
INNER JOIN Jan_Users T1 ON T0.UserID = T1.UserID
INNER JOIN Feb_Users T2 ON T0.UserID = T2.UserID
```

100 %

Results Messages

(No column name)
1 2668

Q (local) (15.0 RTM) DESKTOP-E9IU170\ninas ... Loan\_DB 00:00:00 1 rows

Figure 14. Question 3 – SQL Query

```
SQLQuery1.sql - (lo...E9IU170\ninas (53))* -> X
CREATE VIEW Jan_Users_AS
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 1

CREATE VIEW Feb_Users_NoEmail AS
SELECT DISTINCT (T0.UserID)
FROM modified_listings T0
WHERE T0.MonthID = 2 AND T0.UserID NOT IN (
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 2)

CREATE VIEW Mar_Users_AS
SELECT DISTINCT (T0.UserID)
FROM emails_sent T0
WHERE T0.MonthID = 3

SELECT COUNT(*)
FROM Mar_Users T0
INNER JOIN Jan_Users T1 ON T0.UserID = T1.UserID
INNER JOIN Feb_Users_NoEmail T2 ON T0.UserID = T2.UserID
```

100 %

Results Messages

(No column name)
1 2417

Q (local) (15.0 RTM) DESKTOP-E9IU170\ninas ... Loan\_DB 00:00:00 1 rows

Figure 15. Question 4 – SQL Query

```

SQLQuery1.sql - (lo...E9IU170\ninas (53))* -> X
SELECT COUNT(*)
FROM(
  SELECT DISTINCT T0.UserID
  FROM emails_sent T0
  INNER JOIN modified_listings T1 ON T0.UserID = T1.UserID
  AND 1 = T1.MonthID
  WHERE T0.MonthID = 1 AND T0.UserID NOT IN(
    SELECT DISTINCT T0.UserID
    FROM emails_sent T0
    WHERE T0.MonthID = 1 AND T0.EmailOpened=1)
  AND T1.ModifiedListing = 1) Z0

```

100 %

Results Messages

(No column name)
1 1961

Q (local) (15.0 RTM) DESKTOP-E9IU170\ninas ... Loan\_DB 00:00:00 1 rows

Figure 16. Question 5 – SQL Query

```

SQLQuery1.sql - (lo...E9IU170\ninas (53))* -> X
SELECT COUNT(*)
FROM(
  SELECT DISTINCT T0.UserID
  FROM emails_sent T0
  INNER JOIN modified_listings T1 ON T0.UserID = T1.UserID
  AND 1 = T1.MonthID
  WHERE T0.MonthID = 1 AND T0.UserID NOT IN(
    SELECT DISTINCT T0.UserID
    FROM emails_sent T0
    WHERE T0.MonthID = 1 AND T0.EmailOpened=1)
  AND T1.ModifiedListing = 1

  UNION

  SELECT DISTINCT T0.UserID
  FROM emails_sent T0
  INNER JOIN modified_listings T1 ON T0.UserID = T1.UserID
  AND 2 = T1.MonthID
  WHERE T0.MonthID = 2 AND T0.UserID NOT IN(
    SELECT DISTINCT T0.UserID
    FROM emails_sent T0
    WHERE T0.MonthID = 2 AND T0.EmailOpened=1)
  AND T1.ModifiedListing = 1

  UNION

  SELECT DISTINCT T0.UserID
  FROM emails_sent T0
  INNER JOIN modified_listings T1 ON T0.UserID = T1.UserID
  AND 3 = T1.MonthID
  WHERE T0.MonthID = 3 AND T0.UserID NOT IN(
    SELECT DISTINCT T0.UserID
    FROM emails_sent T0
    WHERE T0.MonthID = 3 AND T0.EmailOpened=1)
  AND T1.ModifiedListing = 1) Z0

```

100 %

Results Messages

(No column name)
1 5249

Q (local) (15.0 RTM) DESKTOP-E9IU170\ninas ... Loan\_DB 00:00:05 1 rows

Figure 17. Question 6 – SQL Query



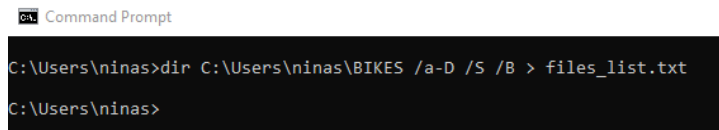
## PART B

### Scenario

A dataset of bike ads from a vehicles commercial site (<https://www.car.gr>) is provided so a data analysis can be conducted. The scope of the analysis is to answer some critical question that may provide valuable insights. Specifically, the dataset contains a plethora of data regarding bike ads that have been posted on the website. Some of these data are the contact information of the seller (name, cellphone, email), and data related to the bike (such as color, age, price and more). To conduct the analysis and to answer those critical questions, first the data will be cleaned thoroughly using R. It should be mentioned that, only the data that are related to the questions that are aimed to be answered will be cleaned. Then, the clean data will be loaded to a Mongo Database, where, through queries, the critical questions will be attempted to be answered.

### Data cleaning & Insertion to MongoDB

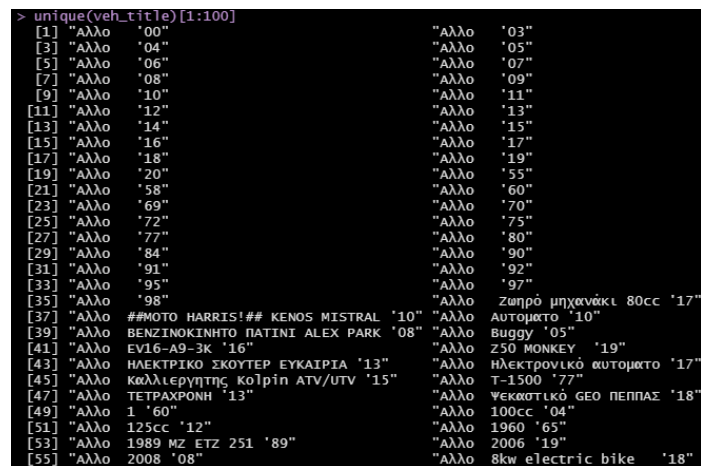
First, since all the files that contain each ad are separated in different folders, a text file that contains the path to each and every one of them is created through the Windows Command Prompt. (Figure 18) Then, the file is read through R to start conducting the data cleaning to the files that are included in it. All the individual actions of data cleaning that are described below, were later included in a R function that applies them to each observation before inserting it to the Mongo Database.



```
C:\Users\ninas>dir C:\Users\ninas\BIKES /a-D /S /B > files_list.txt
C:\Users\ninas>
```

Figure 18. Creation of a file list using Windows Command Prompt

The first variable that was examined was the 'Vehicle Title'. Initially it was examined for null and NAs (this was applied for all variables examined), and none were found. It was detected that some variables had abnormal values that did not make any sense, such as "άλλο '16", though they weren't removed or updated. (Figure 19)



```
> unique(veh_title)[1:100]
[1] "άλλο '00" "άλλο '03"
[3] "άλλο '04" "άλλο '05"
[5] "άλλο '06" "άλλο '07"
[7] "άλλο '08" "άλλο '09"
[9] "άλλο '10" "άλλο '11"
[11] "άλλο '12" "άλλο '13"
[13] "άλλο '14" "άλλο '15"
[15] "άλλο '16" "άλλο '17"
[17] "άλλο '18" "άλλο '19"
[19] "άλλο '20" "άλλο '55"
[21] "άλλο '58" "άλλο '60"
[23] "άλλο '69" "άλλο '70"
[25] "άλλο '72" "άλλο '75"
[27] "άλλο '77" "άλλο '80"
[29] "άλλο '84" "άλλο '90"
[31] "άλλο '91" "άλλο '92"
[33] "άλλο '95" "άλλο '97"
[35] "άλλο '98" "άλλο Ζωηρό μηχανάκι 80cc '17"
[37] "άλλο ##MOTO HARRIS!## KENOS MISTRAL '10" "άλλο Αυτομάτο '10"
[39] "άλλο BENZINOKINHTO PATINI ALEX PARK '08" "άλλο Buggy '05"
[41] "άλλο EV16-A9-3K '16" "άλλο Z50 MONKEY '19"
[43] "άλλο ΗΛΕΚΤΡΙΚΟ ΣΚΟΥΤΕΡ ΕΥΚΑΙΡΙΑ '13" "άλλο Ηλεκτρονικό αυτομάτο '17"
[45] "άλλο Καλλιεργητής κοίτην ATV/UTV '15" "άλλο T-1500 '77"
[47] "άλλο ΤΕΤΡΑΧΡΟΝΗ '13" "άλλο Ψεκαστικό GEO ΠΕΠΠΑΣ '18"
[49] "άλλο 1 '60" "άλλο 100cc '04"
[51] "άλλο 125cc '12" "άλλο 1960 '65"
[53] "άλλο 1989 MZ ETZ 251 '89" "άλλο 2006 '19"
[55] "άλλο 2008 '08" "άλλο 8kw electric bike '18"
```

Figure 19. Unique values of Vehicle Titles

The second variable that was examined was the price. It was discovered that most price values (with 1 exception) included the euro symbol (€) which would make it impossible to perform any calculations with them. Also, they had the dot (.) as a thousand separator, which messed up their transformation to numbers, since they initially were of character type. Both symbols were removed from all values. Additionally, it was observed that some ads had 'Askforprice' as their price value, which were updated to null values. Finally, it is noticed that some prices have a value lower than 250€ or greater than 70.000€, which seem pretty unreasonable, and probably will distort the results of the calculations in the analysis, which is why they were also replaced with null values. (Figure 20)

```
> Price[which(Price < 250 | Price > 70000)] <- NA
> unique(sort(Price, decreasing = TRUE))
[1] 60000 58000 46500 43000 38000 37000 35999 35400 35000 33500 33000 30000 29750 28000 27900 27500 27000 26900 26700 26500
[21] 26000 25700 25500 25000 24990 24000 23900 23800 23500 23000 22800 22500 22000 21989 21900 21700 21500 21000 20900 20500
[41] 20100 20000 19999 19950 19900 19700 19500 19200 19000 18999 18900 18800 18600 18550 18500 18400 18300 18200 18000 17900
[61] 17850 17800 17700 17500 17300 17200 17199 17150 17100 17000 16990 16950 16900 16800 16799 16700 16500 16490 16450 16400
```

Figure 20. Unique prices

The next variable examined was the registration date of the bike. It was noticed that no registration dates were missing, but all of them were in the format (9 / 2018). In this specific analysis, the age of the vehicle in months is not needed, so it was dropped from the variable to only retain the registration's year. Also, it was detected that some ads had registration years prior to 1930. After cross-examining those same ads on the website, it was discovered that they were wrong ads, and had no meaning for this current analysis, so they were replaced with null values. (Figure 21)

```
> unique(sort(Registration, decreasing = TRUE))
[1] "2020" "2019" "2018" "2017" "2016" "2015" "2014" "2013" "2012" "2011" "2010" "2009" "2008" "2007" "2006" "2005" "2004"
[18] "2003" "2002" "2001" "2000" "1999" "1998" "1997" "1996" "1995" "1994" "1993" "1992" "1991" "1990" "1989" "1988" "1987"
[35] "1986" "1985" "1984" "1983" "1982" "1981" "1980" "1979" "1978" "1977" "1976" "1975" "1974" "1973" "1972" "1971" "1970"
[52] "1969" "1968" "1967" "1966" "1965" "1964" "1963" "1962" "1961" "1960" "1959" "1958" "1957" "1956" "1955" "1954" "1953"
[69] "1952" "1951" "1950" "1949" "1945" "1942" "1941" "1940" "1939" "1938" "1934"
```

Figure 21. Unique registration years

Following, the mileage variable was examined. It is observed that all the ads include the km character as a unit of measure, and the comma symbol ',' as a thousand separator. Keeping those characters would make it impossible to conduct any calculations on the variable, so both of them were removed. Additionally, mileage values that were greater than 999.999 km were updated to null values, since they seem pretty unreasonable, and might prove problematic to deal with. (Figure 22)

```
> Mileage[which(Mileage >= 999999)] <- NA
> unique(sort(Mileage, decreasing = TRUE))
[1] 860000 820667 777714 774154 750000 737373 717177 700000 676778 670000 644223 574239 566250 534389 524602 510373 500050
[18] 500000 463752 450000 406541 400000 391194 376879 374444 359114 346290 333333 323000 308500 300000 290000 287647 273860
[35] 250000 242700 240000 235623 234567 232323 230261 225000 220000 218200 211954 207470 200000 199257 198003 198000 194779
```

Figure 22. Unique mileage values

The fifth variable examined was the bikes colors. No missing values (or NAs) were recorded, though it should be mentioned that it was found that all colors existed twice. Specifically, some bikes were described with a color that already existed, but also included the characterization 'Metallic'. It seemed unnecessary to keep that characterization, so it was removed from all observations. (Figure 23)

```
> unique(sort(Color))
[1] "Beige" "Black" "Blue" "Bordeaux" "Brown" "Chromium" "Dark blue"
[8] "Dark Green" "Dark Red" "Gold" "Green" "Grey" "Lemon" "Light blue"
[15] "Orange" "Other" "Pink" "Purple-Violet" "Red" "Silver" "White"
[22] "Yellow"
```

Figure 23. Unique colors

It should be mentioned that a new variable was created. This variable is an index of whether a price negotiable or not, based on whether the publisher mentions it anywhere in the post. Also, the data type of the variable that indicates the number of previous owners per bike, is updated to numeric. This action took place because it will probably prove useful in the upcoming analysis. Finally, the data were inserted to the Mongo Database to initiate the analysis. (Figure 24)

```
> mongo_Conn$insert(data)
List of 6
 $ nInserted   : int 29701
 $ nMatched    : int 0
 $ nModified   : int 0
 $ nRemoved    : int 0
 $ nUpserted   : int 0
 $ writeErrors : list()
```

Figure 24. Data Insertion to Mongo Database

## Programs/Queries

The second task required the calculation of the number of sales ads for bikes that were posted on the website. As expected, it was confirmed that 29701 different ads are included in the website's sample that was provided. (Figure 25)

```
//Question 2
db.BikesColl.find({},{}).count()
```

0 sec.

29701

Figure 25. Count of bikes sales ads

The third task required the calculation of the average price of the bikes that are on sale on the website. It is found that the average sale price is almost at 3.045€. Also, it is worth mentioning that since some prices were set as null from the cleaning process, only the prices of 28290 out of 29701 posts (from Figure 25) were used to calculate the average price. (Figure 26)

```
db.BikesColl.aggregate([
{
  $match: {
    "ad_data.Price":
      {$exists: true
      }
  },
{
  $group: {
    "_id": null,
    "Avg_price": {
      $avg: "$ad_data.Price"
    },
    "Count": {
      $sum: 1
    }
  }
})
```

BikesColl 0.102 sec.

Key	Value
(1)	{ 2 fields }
Avg_price	3044.59402615765
Count	28290.0

Figure 26. Average price of bikes for sale

The next task required the detection of the minimum (excluding the null values) and the maximum prices that are set on posts on the website. It is found that the cheapest bike on the website costs 250€, while the most expensive one is sold at 60.000€. (Figure 27)

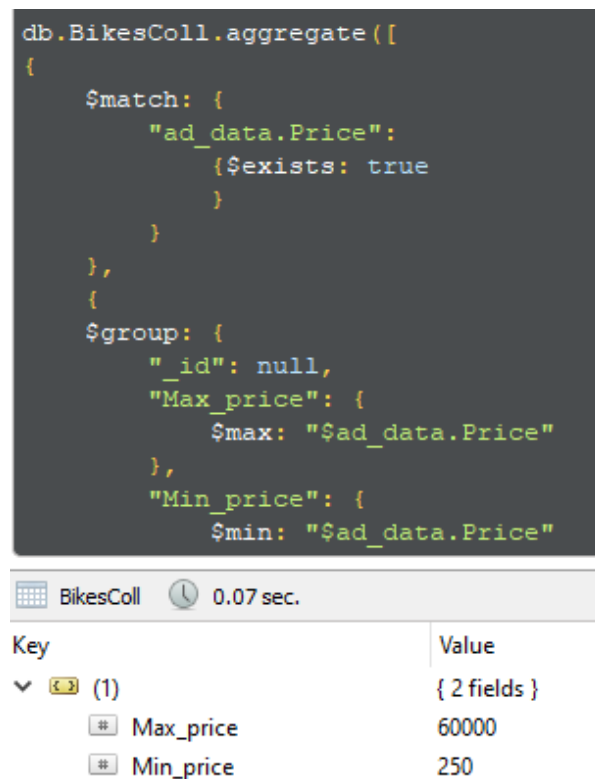


Figure 27. Min and max price of ads

The fifth task required the number of bikes ads with negotiable prices. Using the index that was created in the cleaning process, it was found that 1348 bikes are posted and have a negotiable price. (Figure 28)

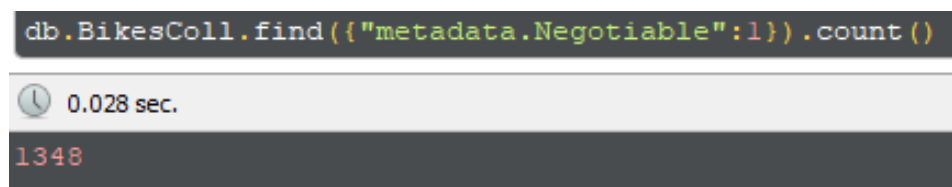


Figure 28. Bikes with Negotiable price

The sixth task required the calculation of the percentages of negotiable prices per bike brand. To calculate those figures, the total number of bike ads per brand had to be calculated along with the number of negotiable prices per ad for each brand. Then, these two calculations were divided to produce the results. (Figure 29)

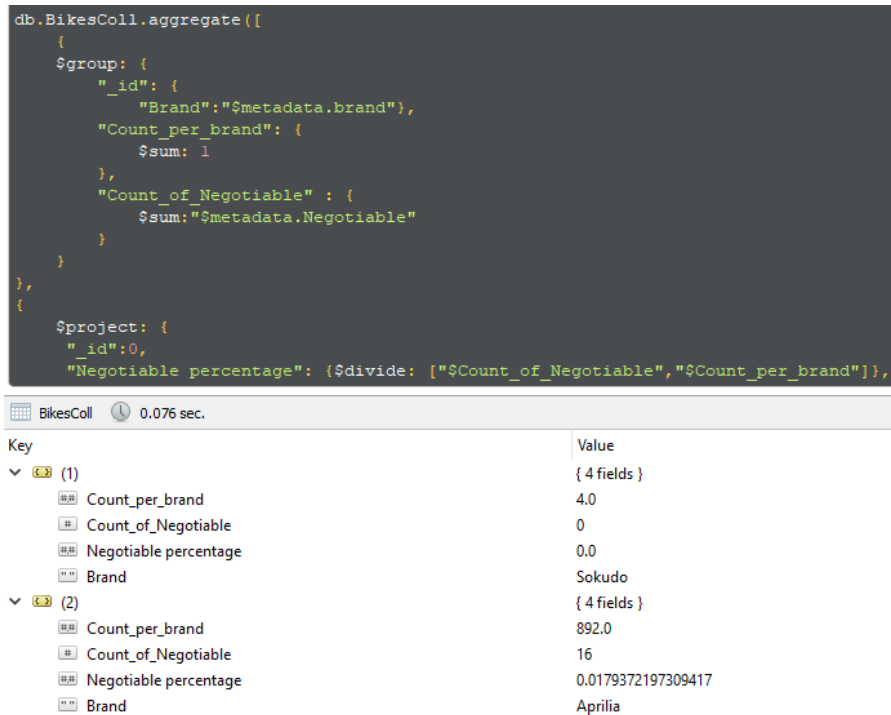


Figure 29. Percentages of bike brands with negotiable prices

The following task required the detection of the brand with the highest average price on the website. It is observed that the brand 'Semog' has the highest average selling price that is equal to 15.600€. (Figures 30)

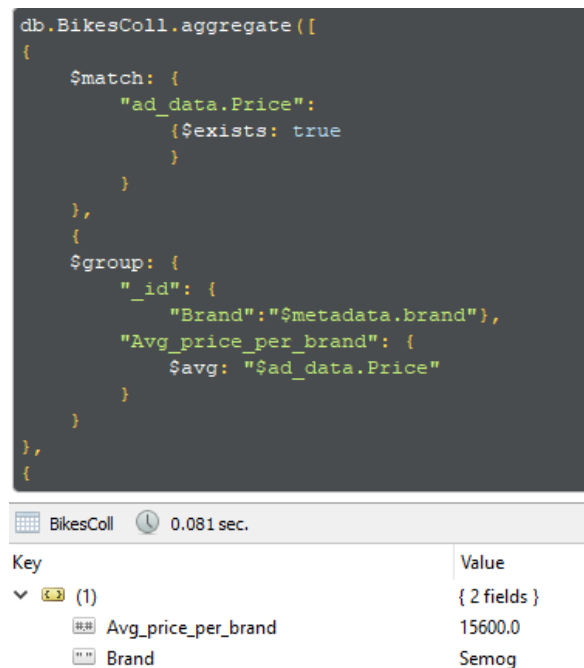


Figure 30. Brand with the highest average price

The eighth task required the detection of the 10 brands with the highest average ages. It is observed that the oldest brand is 'Bsa' with an average age of almost 74 years old. Following, 'Norton' is the second oldest brand on the website, with an average age of almost 71 years old, and so on and so forth. (Figure 31)

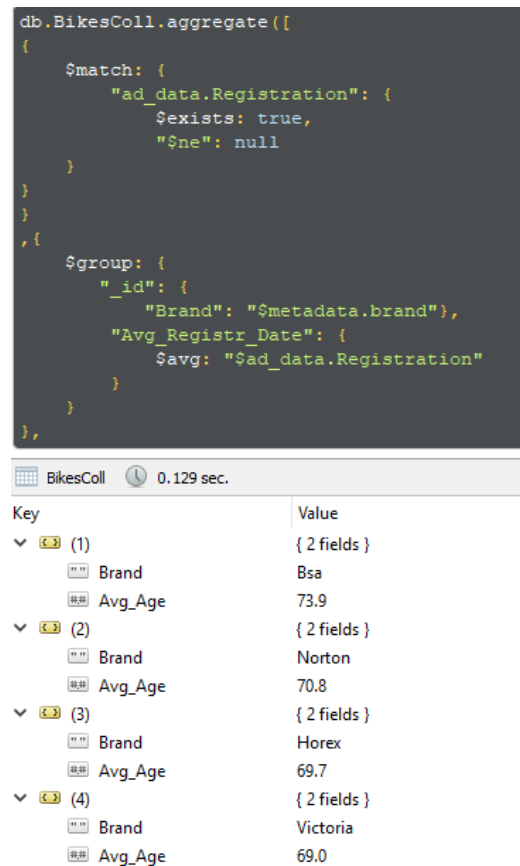


Figure 31. Top 10 brands with the highest average ages

The ninth question regarded the number of bikes that included ABS as an extra feature. It is found that out of 29700 bikes, only 4025 included ABS. (Figure 32)

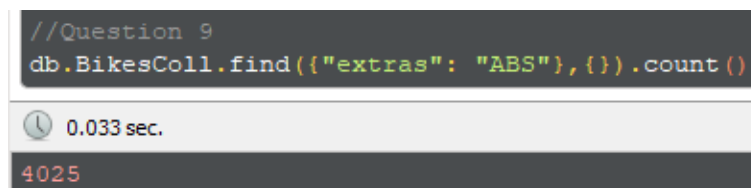


Figure 32. Number of bikes with ABS

The tenth question required the average mileage distances that have been travelled for bikes sold on the website that include 'ABS' and 'Led lights' as extra features. It is found that the average mileage distance travelled for those vehicles is equal to 29.231km. (Figure 33)

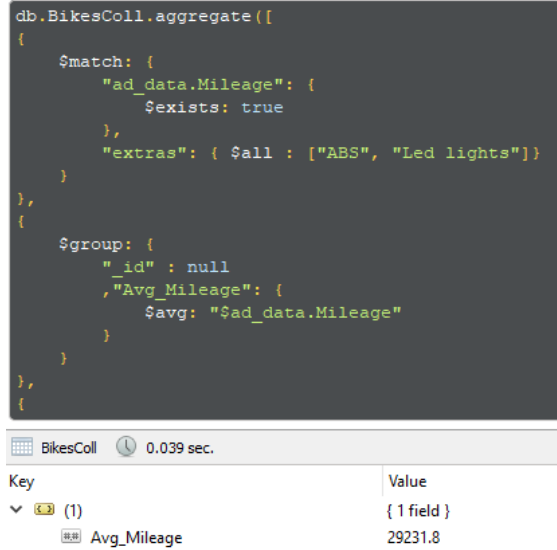


Figure 33. Average mileage distance for bikes with ABS and Led lights

The eleventh question required the detection of the 3 most frequent colors per bike categories. It is observed that UTV Side by Side bikes are most frequently sold in black, blue or white colors, while Sport Touring bikes are most frequently sold in black, silver or white colors, and so on and so forth. (Figure 34)

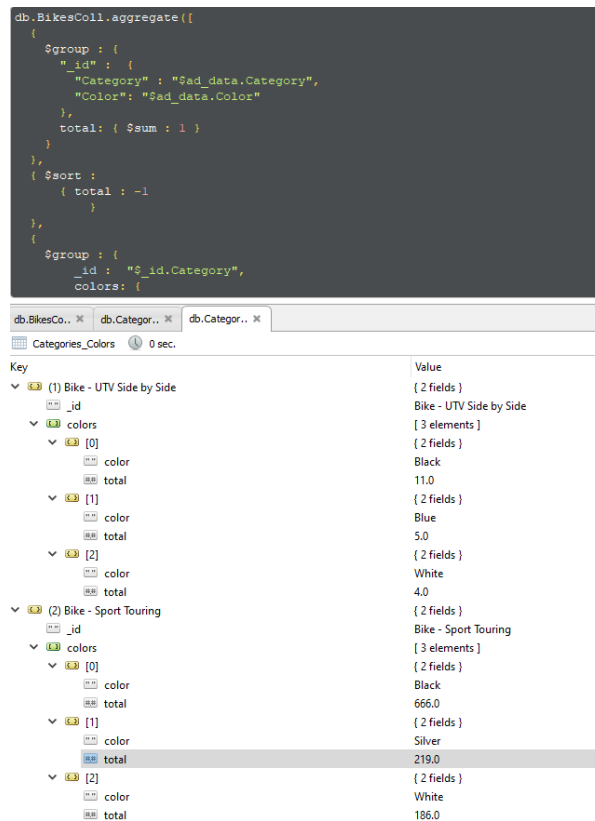


Figure 34. Top 3 colors per bike category

The final task required the identification of ads that can be considered as the best deals. The criteria that were considered to identify those ads were the selling price, the mileage the vehicle has travelled along with its registration date and the number of previous owners. The average values for each of the above figures were calculated and used as a threshold in order to rank evaluate the ads. The average values of each of those variables examined are provided in figure 35 for reference purposes. Specifically, the ads that were considered good deals in the above process were the ones with prices, mileage distances travelled and number of owners that were lower than their corresponding average. At the same time, the bikes that had registration years greater or equal to the mean registration year, were also considered as good deals. (Figure 36) In figure 36, an example of such an ad can be observed where all the criteria set are fulfilled. In fact, a total of 1.556 ads on the website fulfilled those criteria and can be considered as very good possible deals.

```
print("Average price:", Avg_price, ",Average Mileage:", Avg_Mileage, ",Average Registration Year:",Avg_Year, ",Average Number of Owners:",Avg_owners)
```

0 sec.

Average price: 3044.594026157653 ,Average Mileage: 33624.49855434589 ,Average Registration Year: 2005.22384239771 ,Average Number of Owners: 1.1806495263870094

Figure 35. Average values

```
var Avg_Mileage = db.BikesColl.aggregate([
  { "$group": { "_id": "null", Avg_Mileage: { $avg: "$ad_data.Mileage" } } }
]).toArray()[0]["Avg_Mileage"];

var Avg_Year = db.BikesColl.aggregate([
  { "$group": { "_id": "null", Avg_Year: { $avg: "$ad_data.Registration" } } }
]).toArray()[0]["Avg_Year"];

var Avg_owners = db.BikesColl.aggregate([
  { "$group": { "_id": "null", Avg_owners: { $avg: "$ad_data.Previous owners" } } }
]).toArray()[0]["Avg_owners"];

db.BikesColl.find({ $and :[{ "ad_data.Price": { $lte: Avg_price } }
, { "ad_data.Mileage": { $lte: Avg_Mileage } }
, { "ad_data.Registration": { $gte: Avg_Year } }
, { "ad_data.Previous owners": { $lte: Avg_owners } } ] })
```

BikesColl 0.005 sec.

Key	Value	Type
(1) ObjectId("61f92c46225c00007e006072")	{ 9 fields }	Object
_id	ObjectId("61f92c46225c00007e006072")	Objectid
query	{ 4 fields }	Object
title	Honda SH 150i AWOF0 '07	String
ad_id	10050869	String
ad_data	{ 15 fields }	Object
Make/Model	Honda SH 150i AWOF0 '07	String
Classified number	10050869	String
Price	1595	Int32
Category	Bike - Roller/Scooter	String
Registration	2007	Int32
Mileage	32000	Int32
Fuel type	Gasoline	String
Cubic capacity	150 cc	String
Power	16 bhp	String
Color	Black	String
Previous owners	1	Int32
Modified	5 days	String
Times clicked	19	String
Short link	www.car.gr/10050869	String
Telephone	+30 2310886200	String

Figure 36. Ads that are considered as 'Best Deals'