Course: Big Data Systems and Architectures (Part Time)

Assignment #1: Redis & MongoDB

Professor: Spyros Safras

Students

Despotis Spyridon: p2822111

Papailiou Thanasis: p2822128

2022

Contents

Introduction 3

Section One: Generating Analytics with Redis 4

1.1 How many users modified their listing on January? 4

1.2 How many users did NOT modify their listing on January? 4

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)? . 5

1.4 How many users received an e-mail on January and March but NOT on February? 5

1.5 How many users received an e-mail on January that they did not open but they updated their listing anyway? 5

Section Two: Generating Analytics with MongoDB 8

2.1 Add your data to MongoDB. 8

2.2 How many bikes are there for sale? 9

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 1.2? Why? 9

2.4 What is the maximum and minimum price of a motorcycle currently available in the market? 9

2.5 How many listings have a price that is identified as negotiable? 9

Introduction

In this assignment, in section one, we are trying to experiment with an advanced NoSQL key-value data store, named “Remote Dictionary Server” (Redis). Redis is ideal for developing high performance, scalable applications. Also, Redis can be used as the backend for queueing background jobs and can provides the capability to distribute data utilizing the Publish/Subscribe messaging. Companies, such as Twitter, Pinterest, and GitHub for creating background jobs, placing those jobs on multiple queues, and processing them later.

In section two, we are trying to experiment with is a source-available cross-platform document-oriented database program, named MongoDB. MongoDB uses collections of JSON-like documents with optional schemas, just like the MYSQL database contains tables. Specifically, Mongo DB, enables faster access of the data due to its nature of using the internal memory for the storage and helps in easier way of load balancing. Companies, such as Adobe, eBay and Facebook are using MongoDB, for higher availability and scalability.

Finally, the tasks of this assignment are implemented with R programming language in R Studio environment.

Figure 0-1 Redis and MongoDB logos

Section One: Generating Analytics with Redis

For the following tasks, we installed “**redux**” package in R Studio in order to create a connection to the local instance of REDIS.

1.1 How many users modified their listing in January?

Firstly, in order to calculate the number of users modified their listing on January, we created a new bitmap called “ModificationsJanuary”.

Then we used “SETBIT” command equal to “1” for each user that modified his/her listing in January. The bitmaps key is the UserID and the value is either “1”, if the user modified his/her listing, or “0” if the user did not modify his/her listing.

After, with “BITCOUNT” command, we calculated the total number of users modified theirs listings in January (equal to “1” in the bitmap).

-The result is that **9.969 users** modified their listing in January.

1.2 How many users did NOT modify their listing in January?

We used “BITOP” command, with “NOT” argument, in order invert the bits of the January modifications and store them in a new bitmap. After the inversion, the value “1” represents the users who did not modify their listings. Then, using “BITCOUNT” command, we can see that **10.031 users** did not modify their listing in January.

The total number of the users from the question 1.1 and 1.2, (10.031 + 9.969) is **20.000** and does not much with the total number of users **19.999**. Due to the fact that each byte has 8 bits and the result of BITOP operation, would always be an integer multiple of 8, the results do not match by 1 unit. For the 19999 users Redis stores the bits with a length of 20000 bits (1 byte = 8 bits and 20000/8 = 2500 bytes). In other words, because all BITOP operations happen at a byte-level increments, Redis for one extra bit will store a full byte of 8 bits. So, we have one extra bit at the end that does not correspond to a user and its value is 0. Since we perform the NOT operation that bit becomes 1 and increments the total users that did not modify in January by 1. So, in our case, we have actually 10.030 users, but the operation does a rounding to fulfil the bytes structure and it returns 10.031 users.

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)?

In this case, we created 3 new bitmaps, for users that received at least one email per month ("EmailsJanuary","EmailsFebruary","EmailsMarch"). Then with "BITOP" command with the "AND" argument, we created a new bitmap named "out3" followed by "BITCOUNT" command, to calculate the intersection between the three bitmaps of the three months.

-We found out that **2.668 users** received at least one e-mail per month.

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

We used "BITOP" command, with the "NOT" argument, in order invert the bits of the "EmailsFebruary" bitmap (from 1.3 question) and create a new bitmap named "NotEmailsFebruary" for users who did not receive an email in February. Afterwards, we used "BITOP" command followed by the "AND" argument to calculate the intersection between the bitmaps.

-The answer is that **2.417 users** received an email in January and March but not on February.

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

Firstly, we created a new dataframe named "data" that was the merge from the two given datasets "emails_sent.csv" and "read_csv". Then, we created a new bitmap for those users who did not open their emails ("EmailOpened" equal to "0") and for the month of January ("MonthID.X ,Y equal to "1").

The final output was **2.807 users**.

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

In this case we implemented the same methodology as question 1.5, but for also February and March. So, we created two more bitmaps named

"EmailsNotOpenedUpdatedFebruary" and "EmailsNotOpenedUpdatedMarch". With "BITOP" command followed by the "OR" argument and after the "BITCOUNT" command we calculated the total number of the users.

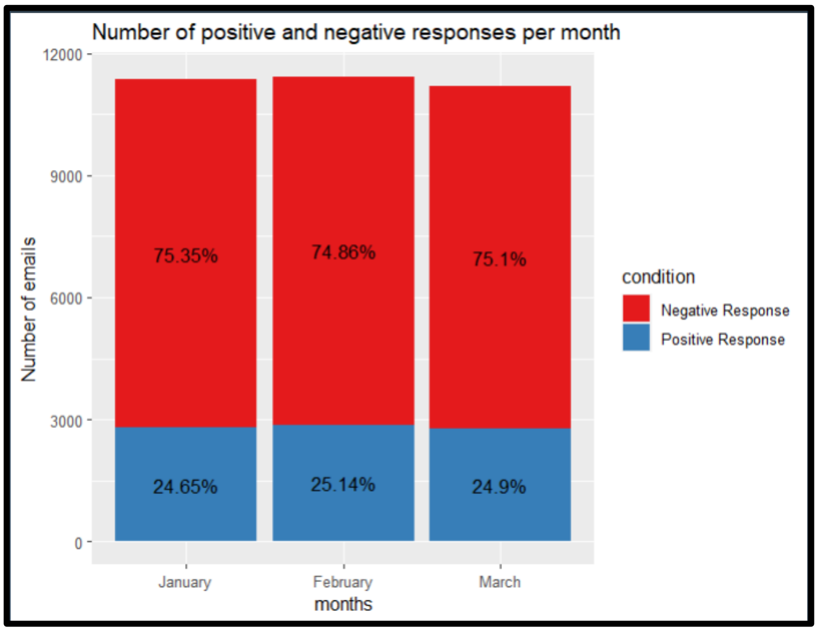
-The output was **7.221** users.

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

In order to understand if the strategy really works, we segmented the users according to their action when they receive and email. Specifically, there are four scenarios:

- Users who read the email and updated their listing ("EmailOpenedUpdated")
- Users who did not read their email and did not update ("EmailNotOpenedNotUpdated")
- Users that read the email and did not update their listing ("EmailOpenedNotUpdated)
- Users who did not read and updated their listing ("EmailNotOpenedUpdated)

The following graph shows (figure 1) the positive and negative responses of the users (if they proceed to modification or not) that actually read the email recommendations. We can conclude that approximately 25% of the users have positive response to the emails, meaning that they read their email and modified their listings. The majority of the users, despite reading the email did not proceed to modification.



The following graph (figure 2) provides more details about the user's behaviour. It is clear that roughly 50% of the users did not even open the email, while only 24,897% of the users actually opened the email and proceed to the appropriate modifications. Last but not least the percentage of 24.985% confirms our findings from figure 1, that ¼ of the emails induced the users to modify their listing. To summarize it does not make sense to keep sending emails to the sellers with the same strategy.

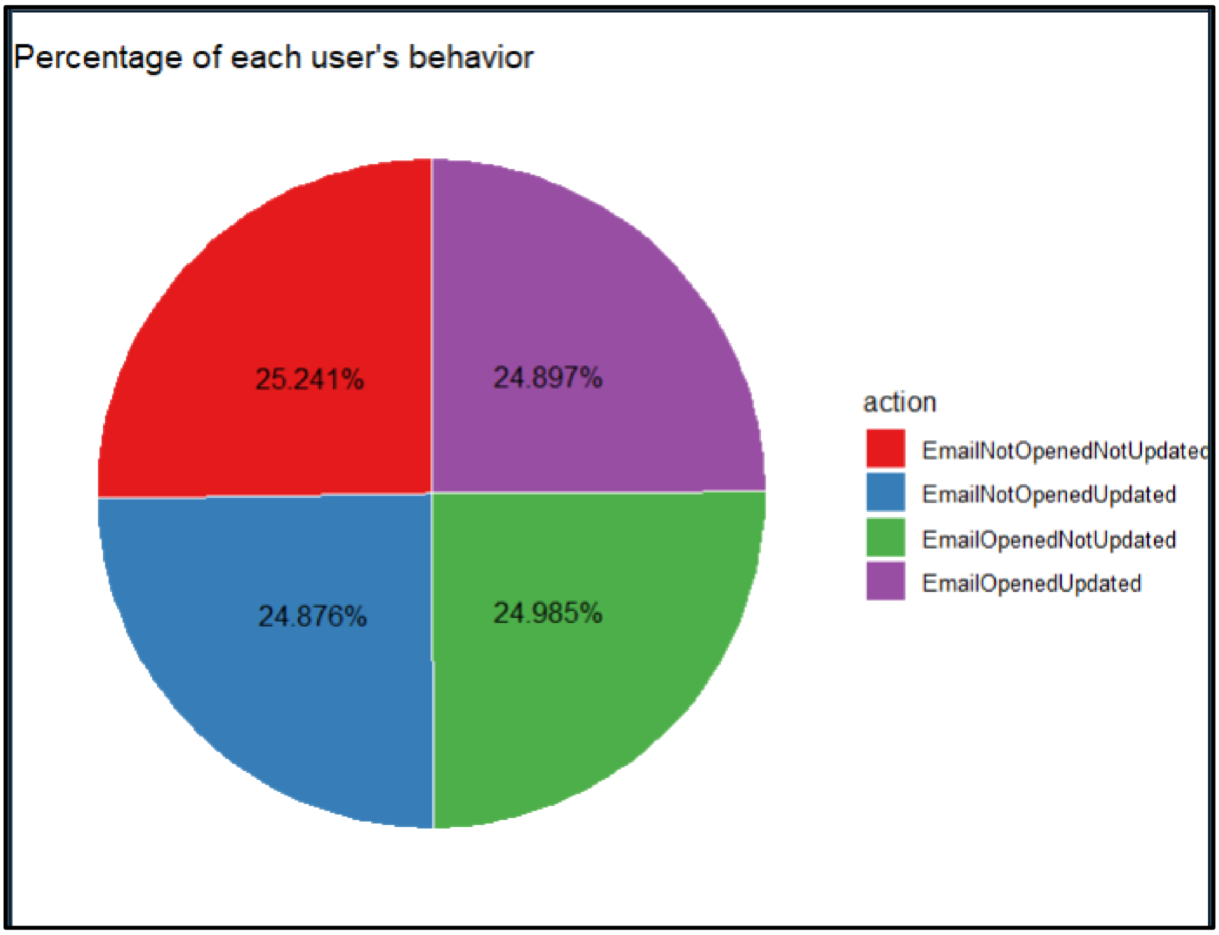


Figure 0-2 Percentage of Each Users Behavior

2.1 Add your data to MongoDB.

At first, we create a connection with the MongoDB server, by running a mongo shell to connect to a MongoDB instance running on localhost with default port 27017. Instead of working on the shell, we managed and queried our data with R, using “jsonlite” and “mongolite” packages. We have also used Robo3T as a visual tool to help manage and manipulate the data in MongoDB.

In order to insert the data into MongoDB, we aim to create a clean dataframe using the json files inside bikes folder. To achieve this, we have created a list with the paths of the json files in the bikes folder, using Windows PowerShell and “dir -Recurse -Name -File > files_list.txt” command.

After reading the files_list.txt through R and saving the paths in a dataframe, we used the “fromJSON” function that reads the content in JSON format and de-serializes it into R objects.

The next step is to insert these objects into a MongoDB collection named “Full_Dataset”. In order to clean the data, we create a nested dataframe using mongolite command “findAll”. This dataframe is composed of other dataframes and lists. To simplify the database structure, we split the nested dataframe and we create small dataframes, one for each key that defines a full of records, and one that store the simple keys. The final step is to merge them all in one, which has a simpler structure as the starting one.
For the cleaning process the next 3 steps are implemented:

✓ Price column: Removed “€” symbol, removed the “.” character and transformed the column to numeric type.

✓ Mileage column: Removed “km” character, removed the commas “,” character and transformed the column to numeric type

✓ Registration column: Removed all characters before and up to ":”, so as to keep only the year and transformed the column to numeric type.

The final step is to drop the old collection named “Full_Dataset” and insert the clean dataframe in a new MongoDB collection, named “Cleaned_Dataset”. We have now successfully extrapolated the information in each sub-dataframe and we have set properly a cleaned dataframe, which can be worked with “mongolite” package.

2.2 How many bikes are there for sale?

To answer this question, we just had to count the total number of listings since each listing represents a bike. There **are 29.701 bikes** available for sale.

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 1.2? Why?

Before we make the aggregation, we have to match first all bikes with a price greater than 100 because all the listings with price less than this usually are for parts and not for actual bikes. This along with the fact that some records stored the price with a string “AskForprice” results in a smaller number of listings used in comparison with the previous question. The average price **is 3.033,611.**
In order to calculate this quantity, 28461 records have been used. This is because there 1240 records stored the price with a string “AskForprice”, or the price was less than 100.

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

For the min price we can safely pick 101 euros as the minimum for the same reason we described in the previous question. The maximum price is **89.000**.

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

To find this we search for the word “Negotiable” in the “model” document, and the total number of bikes identified as negotiable are **1.348**.