MSc in Business Analytics - Part time

BIG DATA SYSTEMS AND ARCHITECTURES

Redis and MongoDB Assignment

Professor:

Mr. Safras Spyridon

Gkourioti Panagiota – P2822109

Koletsi Thaleia – P2822120

Table of Contents

1. Redis

In this section the "recorded actions" dataset will be used in order to generate some analytics with REDIS. This dataset contains two files, emails_sent and modified_listings. The first one contains User IDs that have received an e-mail at least once. The second one contains all the User IDs of the classifieds provider and a flag that indicates whether the user performed a modification on his/her listing. Both datasets contain entries for the months January, February and March.

After connecting remotely to REDIS via R, we proceed to answer some queries using BITMAPs.

```
#Remote Connection
library("redux")
r <- redux::hiredis(redux::redis_config(host = "127.0.0.1",port = "6379"))
emails_sent<-read.csv(file.choose(),header=T)
mod_listings<-read.csv(file.choose(),header=T)
```

Task 1.1.

In this task, we would like to find how many users modified their listing in January. To answer that, we create a separate data frame, from modified listings data frame, that includes only the respective data for January. Then, in order to create a BITMAP, we use the SETBIT command, which sets one to user who modified their listing and zero in the opposite situation. The BITCOUNT command calculates that 9,969 users modified their listing. The code used is displayed below.

```
#we create a separate data frame, from modified listings data frame, that includes only the respective data for January.
Jan_mod<- mod_listings[which(mod_listings$MonthID==1),]

for (i in 1:nrow(Jan_mod)){

    if (Jan_mod$ModifiedListing[i]==1) { r$SETBIT("ModificationsJanuary",i,"1")}

}

r$BITCOUNT("ModificationsJanuary") #9969 users modified their listing on January
```

Task 1.2.

In this task, we would like to find the users who did not modify their listing in January. Therefore, we will perform inversion on the "ModificationsJanuary" BITMAP and use BITCOUNT to calculate the answer. The result is that 10,031 users did not modify their listing in January.

```
r$BITOP("NOT","NOT_ModificationsJanuary","ModificationsJanuary")
r$BITCOUNT("NOT_ModificationsJanuary") #10031 users did not modify their listing in January
```

It should be noted that the data frame with January listings has 19,999 observations but if we add the result of the BITCOUNTs, we find 20,000. The reason why there is one more observation is that BITOP operations happen at byte-level increments. One byte consists of 8 bits, thus, to interpret the output of BITCOUNT operation from bits to bytes, we need to divide with 8, so system creates one more bit to have zero remainder (19,999/8 =2499,67, 20,000/8= 2500).

Task 1.3.

For this task, the "emails_sent" dataset is used. Following the below process, we change the data frame "Emails" grouping by UserID and MonthID and creating an extra column "Total emails" that depicts the number of received emails per UserID. Then, we also add three extra columns, one per month, we populate the emails received per UserID and per month, so, in the final data frame, we keep only UserID column and columns for each of the months (we delete Total emails & MonthID). Having created that, we replace with one when the user has received at least one e-mail per month and with zero when they have not received any. Afterwards, we create three BITMAPS, one for each month, by setting one when user received at least one email at the corresponding month. Finally, to calculate the total users who received at least one email in January, February and March, we use BITOP AND operation and find that 2,668 users received at least one email per month.

```
library(dplyr)

emails_sent["Total_emails"]<-1

#group emails by user and month
Emails<-emails_sent %>% group_by(UserID,MonthID) %>% summarize_at("Total_emails",sum)

#create three new columns
Emails["January"]<-0
Emails["February"]<-0
Emails["March"]<-0

#assign emails to columns according to month
for (i in 1:nrow(Emails)){

  if (Emails$MonthID[i]==1) {Emails$January[i]=Emails$Total_emails[i]}
  else if (Emails$MonthID[i]==2) {Emails$February[i]=Emails$Total_emails[i]}
  else if (Emails$MonthID[i]==3) {Emails$March[i]=Emails$Total_emails[i]}

}

Emails<-Emails[,-c(2,3)]

#summarize by userID
Emails<-Emails %>% group_by(UserID) %>% summarize_all(sum)

#replace with one if the user has received at least one e-mail per month and zero if they have not received any
for (i in 1:nrow(Emails)) {

  if (Emails$January[i]>0) {Emails$January[i]=1}
  else {Emails$January[i]=0}
}
for (i in 1:nrow(Emails)) {

  if (Emails$February[i]>0) {Emails$February[i]=1}
  else {Emails$February[i]=0}
}
```

```
for (i in 1:nrow(Emails)) {

  if (Emails$March[i]>0) {Emails$March[i]=1}
  else {Emails$March[i]=0}
}

#create BITMAPs for each month
for (i in 1:nrow(Emails)){

  if (Emails$January[i]==1) { r$SETBIT("Emails_in_January",i,"1")}

}
for (i in 1:nrow(Emails)){

  if (Emails$February[i]==1) { r$SETBIT("Emails_in_February",i,"1")}

}
for (i in 1:nrow(Emails)){

  if (Emails$March[i]==1) { r$SETBIT("Emails_in_March",i,"1")}

}

r$BITOP("AND","One_or_more_in_3M",c("Emails_in_January","Emails_in_February","Emails_in_March"))
r$BITCOUNT("One_or_more_in_3M") # 2,668 users with at least one email per month
```

Task 1.4.

In this task, we would like to find the users who received an e-mail in January and March but not in February. Therefore, we use the BITOP NOT operation in order to reverse the Emails_in_February bitmap and the BITOP AND operation to find the intersection between Emails_in_January and Emails_in_March BITMAPs. With the use of BITCOUNT command, it is found that 2,417 users received an e-mail in January and March but not in February.

```
r$BITOP("NOT","NOT_Emails_in_February","Emails_in_February")
r$BITOP("AND","Jan&Mar",c("Emails_in_January","Emails_in_March"))

r$BITOP("AND","Jan&March_AND_NOT_Feb",c("NOT_Emails_in_February","Jan&Mar"))
r$BITCOUNT("Jan&March_AND_NOT_Feb") #2417 users received an e-mail on January and March but not on February
```

Task 1.5.

For this task, we would like to calculate the users who received an e-mail in January that they did not open but they updated their listing anyway. At first, we take a subset of the “emails_sent” dataset for January and create a data frame with the sum of opened emails for each user in January. Then, we replace the sum of opened emails with zero in case they have opened at least one email and with one in case they have not opened any. Afterwards, we construct a BITMAP showing the users who did not open their emails and we find that they are 3,972. Similarly, we construct another BITMAP showing the users who modified their listings and we find that they are 9,969. Therefore, using BITOP AND operation, it is concluded that 1,961 users received an e-mail in January that they did not open but updated their listing anyway.

```
#subset of emails sent in January
Jan<- emails_sent[which(emails_sent$MonthID==1),-c(1,3)]
#create a data frame with the sum of opened emails for each user in January
Emails.Jan<-Jan %>% group_by(UserID) %>% summarize_at("EmailOpened",sum)
Emails.Jan<-as.data.frame(Emails.Jan)

#replace the sum of opened emails with 0 in case they have opened at least one email
#and with 1 in case they have not opened any
for (i in 1:nrow(Emails.Jan)){
  if(Emails.Jan[i,2]!=0){Emails.Jan[i,2]=0}
  else{Emails.Jan[i,2]=1}
}

#bitmap for not opened emails in January
for (i in 1:nrow(Emails.Jan)){
  r$SETBIT("EmailsNotOpenedInJanuary",Emails.Jan[i,1],Emails.Jan[i,2])
}

r$BITCOUNT("EmailsNotOpenedInJanuary") #3972 users have not opened any emails in January

#subset of modified listings in January
Jan_mod2<- mod_listings[which(mod_listings$MonthID==1),-2]

#bitmap for modified listings in January
for (i in 1:nrow(Jan_mod2)){
  r$SETBIT("ModificationsJanuary2",Jan_mod2[i,1],Jan_mod2[i,2])
}

r$BITCOUNT("ModificationsJanuary2") #9969 users have modified their listing in January

#perform AND operation in order to consolidate the information derived from the previous two bitmaps and
#calculate the users that received an email on January which they did not open but they updated their listing anyway
r$BITOP("AND","JanNotOpenedButUpdated",c("ModificationsJanuary2","EmailsNotOpenedInJanuary"))
j1<- r$BITCOUNT("JanNotOpenedButUpdated") #1961 users
```

Task 1.6.

For this task, we follow the same procedure as previously, but for February and March. 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 in 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.

```
#subset of emails sent in February
Feb<- emails_sent[which(emails_sent$MonthID==2),-c(1,3)]
#create a data frame with the sum of opened emails for each user in February
Emails.Feb<-Feb %>% group_by(UserID) %>% summarize_at("EmailOpened",sum)
Emails.Feb<-as.data.frame(Emails.Feb)

#replace the sum of opened emails with 0 in case they have opened at least one email
#and with 1 in case they have not opened any
for (i in 1:nrow(Emails.Feb)){
  if(Emails.Feb[i,2]!=0){Emails.Feb[i,2]=0}
  else{Emails.Feb[i,2]=1}
}

#bitmap for not opened emails in February
for (i in 1:nrow(Emails.Feb)){
  r$SETBIT("EmailsNotOpenedInFebruary",Emails.Feb[i,1],Emails.Feb[i,2])
}

r$BITCOUNT("EmailsNotOpenedInFebruary") #3945 users have not opened any emails in February


#subset of modified listings in February
Feb_mod<- mod_listings[which(mod_listings$MonthID==2),-2]

#bitmap for modified listings in February
for (i in 1:nrow(Feb_mod)){
  r$SETBIT("ModificationsFebruary",Feb_mod[i,1],Feb_mod[i,2])
}

r$BITCOUNT("ModificationsFebruary") #10007 users have modified their listing in February

#perform AND operation in order to consolidate the information derived from the previous two bitmaps and
#calculate the users that received an email on February which they did not open but they updated their listing anyway
r$BITOP("AND","FebNotOpenedButUpdated",c("ModificationsFebruary","EmailsNotOpenedInFebruary"))
f1<- r$BITCOUNT("FebNotOpenedButUpdated") #1971 users


Mar<- emails_sent[which(emails_sent$MonthID==3),-c(1,3)]
#create a data frame with the sum of opened emails for each user in March
Emails.Mar<-Mar %>% group_by(UserID) %>% summarize_at("EmailOpened",sum)
Emails.Mar<-as.data.frame(Emails.Mar)

#replace the sum of opened emails with 0 in case they have opened at least one email
#and with 1 in case they have not opened any
for (i in 1:nrow(Emails.Mar)){
  if(Emails.Mar[i,2]!=0){Emails.Mar[i,2]=0}
  else{Emails.Mar[i,2]=1}
}

#bitmap for not opened emails in March
for (i in 1:nrow(Emails.Mar)){
  r$SETBIT("EmailsNotOpenedMarch",Emails.Mar[i,1],Emails.Mar[i,2])
}

r$BITCOUNT("EmailsNotOpenedMarch") #3948 users have not opened any emails in March


#subset of modified listings in March
Mar_mod<- mod_listings[which(mod_listings$MonthID==3),-2]

#bitmap for modified listings in February
for (i in 1:nrow(Mar_mod)){
  r$SETBIT("ModificationsMarch",Mar_mod[i,1],Mar_mod[i,2])
}

r$BITCOUNT("ModificationsMarch") #9991 users have modified their listing in March

#perform AND operation in order to consolidate the information derived from the previous two bitmaps and
#calculate the users that received an email on March which they did not open but they updated their listing anyway
r$BITOP("AND","MarNotOpenedButUpdated",c("ModificationsMarch","EmailsNotOpenedMarch"))
m1<- r$BITCOUNT("MarNotOpenedButUpdated") #1966 users


#perform OR operation in order to consolidate the information derived from the three bitmaps regarding January,February,March and
#calculate the users that received an email 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
r$BITOP("OR","JanFebMarNotOpenedButUpdated",c("JanNotOpenedButUpdated","FebNotOpenedButUpdated","MarNotOpenedButUpdated"))
r$BITCOUNT("JanFebMarNotOpenedButUpdated") #5249 users
```

Task 1.7.

To answer the question if it makes sense to continue sending emails to the users, we construct the bitmaps for users that opened their email and updated their listing. By using the previous results, we then calculate the percentages of the users that have not opened their emails but still did modifications (33.6%) and of the users that have opened emails and did modifications (48.2%). Since the percentage of users who made changes and opened their emails is greater than the percentage of users who modified without checking their newsletters, the emails feedback process should remain as it is.

```
#bitmap for opened emails in January
for (i in 1:nrow(Emails.Jan)){
  if (Emails.Jan[i,2]=="0"){
  r$SETBIT("EmailsOpenedInJanuary",Emails.Jan[i,1],"1")
  }
  else{r$SETBIT("EmailsOpenedInJanuary",Emails.Jan[i,1],"0")}
}
r$BITCOUNT("EmailsOpenedInJanuary")

r$BITOP("AND","JanOpenedButUpdated",c("ModificationsJanuary2","EmailsOpenedInJanuary"))
j2<- r$BITCOUNT("JanOpenedButUpdated") #2797 users opened email and updated their listing on January

#bitmap for opened emails in February
for (i in 1:nrow(Emails.Feb)){
  if (Emails.Feb[i,2]=="0"){
    r$SETBIT("EmailsOpenedInFebruary",Emails.Feb[i,1],"1")
  }
  else{r$SETBIT("EmailspOpenedInFebruary",Emails.Feb[i,1],"0")}
}
r$BITCOUNT("EmailsOpenedInFebruary")

r$BITOP("AND","FebOpenedButUpdated",c("ModificationsFebruary","EmailsOpenedInFebruary"))
f2<- r$BITCOUNT("FebOpenedButUpdated") #2874 users opened email and updated their listing on February

#bitmap for opened emails in March
for (i in 1:nrow(Emails.Mar)){
  if (Emails.Mar[i,2]=="0"){
    r$SETBIT("EmailsOpenedInMarch",Emails.Mar[i,1],"1")
  }
  else{r$SETBIT("EmailsOpenedInMarch",Emails.Mar[i,1],"0")}
}
r$BITCOUNT("EmailsOpenedInMarch")

r$BITOP("AND","MarOpenedButUpdated",c("ModificationsMarch","EmailsOpenedInMarch"))
m2<- r$BITCOUNT("MarOpenedButUpdated") #2783 users opened email and updated their listing on March


#sum the previous results for all three months
TotalNotOpenedButUpdated <- j1+f1+m1 #5898 users did not open any email and updated their listing
TotalOpenedButUpdated <- j2+f2+m2 #8454 users opened email and updated their listing

#find total users who modified their listings
mod<- mod_listings[which(mod_listings$ModifiedListing==1),]
uniq.user<- length(unique(mod$UserID)) #17541 users

#find percentages
perc.not.opened <- (TotalNotOpenedButUpdated/uniq.user)*100 #33.6%
perc.opened <- (TotalOpenedButUpdated/uniq.user)*100 #48.2%
```

2. Mongo DB

For this section the Mongo Distributed database will be used in order to extract useful aggregations and analytics from a range of JSON files that contain information about motorcycles for sale. Mongo DB will be used because it supports data store in a JSON-like format that provides the user the opportunity to change files' structure over time if needed and have a flexible schema.

Initial step of the specific analysis is to read the available data (29,695 JSON files) and proceed with a data cleaning so that inaccurate and "dirty" elements will be identified, corrected, and removed. During this procedure, raw data types, formats or structure can be modified in order for the data to be fully prepped for elaboration.

Task 2.1.

Specifically, for the presented project the following command in Windows CMD is used so that the data can be imported, read and elaborated by R programming language: *"C:\Users\...\Msc BA\Big Data & Architectures\BIKES_DATASET> dir /a-D /S /B > files_list.txt",* and thus a file that includes a list with the paths for each of the available JSON files is created and saved in a data frame in R, as per below:

```
m <- mongo(collection = "bikes",  db = "mydb", url = "mongodb://localhost")
files_list<- read.delim("C:\\Users\\thale\\OneDrive\\files_list.txt", header = FALSE)
files_list[,1]<- gsub("\\", "\\\\", files_list[,1], fixed=TRUE)
```

Next step is to proceed with the data cleaning and make necessary transformations. Especially, as indicated below:

- Replace prices with NAs in cases when the price is unknown.
- Remove the symbol of currency in "Price" column.
- Convert "Price" to numeric values, so we can make aggregations.
- Remove "km" and comma in "Mileage" column and convert the values to numeric.
- Delete "cc" and comma in "Capacity" column and transform the data to numeric type.
- Remove "bhp" and comma in "Power" column and convert the values to numeric.
- Convert columns "Times clicked" and "Registration" to numeric as well.
- Add another column named "Age" that calculates the age of the motorcycle.

```r
#read the file with the paths of json files
files_list<- read.delim("C:\\Users\\DELL\\Documents\\Msc R\\lab files\\BIKES_DATASET\\BIKES\\files_list.txt", header = FALSE)
files_list[,1]<- gsub("\\", "\\\\", files_list[,1], fixed=TRUE)

#cleaning and insert data to MongoDB
library(jsonlite)
library(stringr)
for (i in 1:nrow(files_list)) {
  file <- fromJSON(readLines(files_list[i,],encoding = "UTF-8"))
  if (file$ad_data$Price == 'Askforprice')
  {
    file$ad_data$Price <- "NA"
  }
  else {
    file$ad_data$Price <- gsub("[[:punct:]]", "", file$ad_data$Price) #remove the punctuation
    file$ad_data$Price <- gsub("\200", "", file$ad_data$Price) #remove \200 which represents euro
    file$ad_data$Price <- as.numeric(file$ad_data$Price)
  }
  file$ad_data$Mileage <- gsub(" km", "", file$ad_data$Mileage)
  file$ad_data$Mileage <- gsub(",", "", file$ad_data$Mileage)
  file$ad_data$Mileage <- as.numeric(file$ad_data$Mileage)
  file$ad_data$`Cubic capacity` <- gsub(" cc", "", file$ad_data$`Cubic capacity`)
  file$ad_data$`Cubic capacity` <- gsub(",", "", file$ad_data$`Cubic capacity`)
  file$ad_data$`Cubic capacity` <- as.numeric(file$ad_data$`Cubic capacity`)
  file$ad_data$Power <- gsub(" bhp", "", file$ad_data$Power)
  file$ad_data$Power <- gsub(",", "", file$ad_data$Power)
  file$ad_data$Power <- as.numeric(file$ad_data$Power)
  file$ad_data$`Times clicked` <- as.numeric(file$ad_data$`Times clicked`)
  file$ad_data$Registration <- str_sub(file$ad_data$Registration,-4,-1)
  file$ad_data$Registration <- as.numeric(file$ad_data$Registration)
  file$ad_data$Age <- 2022 - file$ad_data$Registration
  file <- toJSON(file, auto_unbox = TRUE)
  m$insert(file)
}

str(fromJSON(file))
```

After the above changes, we insert the transformed data to MongoDB and check the structure of the last file inserted to make sure the cleaning is correct.

Task 2.2.

To calculate the number of the total bikes that are for sale, we used the following command through mongolite:

```r
m$count()
```

As calculated, the number of the total bikes for sale is 29,701 bikes.

Task 2.3.

For the calculation of the average price of a motorcycle, it is important to define the appropriate price range. There are many items in the data having an extremely low price that is not representative for a motorcycle sale. Thus, only the items with price over 100 € are used for the computation.

Through “aggregate” Mongo command we can determine the range we would like to use and find the average price, which is estimated at 3,033.61 €, while by using “count” the total number of listed used bikes is also calculated (28,461 motorcycles). The number differs from the total of 29,701 (Task 2.2) because many listings have been excluded since their price is less than 100 €.

```
#we define price over 100 € to exclude any unrealistic prices
m$aggregate(
  '[{
  "$match": {
    "ad_data.Price": {
      "$gt": 100
    }
  }
},
{
  "$group": {
    "_id": null,
    "AvgPrice": {
      "$avg": "$ad_data.Price"
    },
    "count": {
      "$sum": 1
    }
  }
}]')  #the average price of a motorcycle is 3,033.61 € and the number of listings that were used is 28,461
```

The output is the following:

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

Task 2.4.

"Aggregate" Mongo command is used again in order to find the maximum and minimum price of a motorcycle which is currently available in the market. Through these commands we can filter, again, the specific range that we will use (items with price over 100 €) and by using "max" and "min" commands we calculate the requested numbers. Maximum price seems to be 89,000 € while minimum price is 101 €, regarding values that are greater than 100.

```
m$aggregate(
  '[{
  "$match": {
    "ad_data.Price": {
      "$gt": 100
    }
  }
},
{
    "$group" : {
      "_id" : null,
      "MaxPrice" : {
        "$max" : "$ad_data.Price"
      },
    "count": {
      "$max": 1
      }
    }
  }]')  #the maximum price of a motorcycle is 89,000 €
```

```
m$aggregate(
  '[{
  "$match": {
    "ad_data.Price": {
      "$gt": 100
    }
  }
},
{
  "$group": {
    "_id": null,
    "MinPrice": {
      "$min": "$ad_data.Price"
    },
    "count": {
      "$sum": 1
    }
  }
}]')  #the minimum price of a motorcycle is 101 €, given that we only used prices above 100 €
```

The outputs are the following:

```
  _id MaxPrice count
1  NA    89000     1
```

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

Task 2.5.

By using "regex" command in an aggregation we can filter those motorcycles that include the word "Negotiable" (in any format) in "metadata" category and "model" field. Counting those items, we can see that the total number of bikes with a price characterized as negotiable are 1,348.

```
m$aggregate(
  '[{
  "$match": {
    "metadata.model": {
      "$regex": "Negotiable",
      "$options": "i"
    }
  }
},
{
  "$group": {
    "_id": null,
    "count": {
      "$sum": 1
    }
  }
}]')  #the total number of bikes with a price characterized as negotiable are 1,348
```

The output is the following:

```
  _id count
1  NA  1348
```

Task 2.8. (Optional)

To define which are the top ten models with the highest average age we use the “avg” in an aggregation as before by grouping by brand and then sorting in descending order and taking the first ten listings. Thus, we find the below models in the top ten:

```
Top10ByAge<- m$aggregate(
  '[{
  "$group": {
    "_id": {
      "brand": "$metadata.brand"
    },
    "AvgAgePerBrand": {
      "$avg": "$ad_data.Age"
    }
  }
},
{"$sort": {
    "AvgAgePerBrand": -1
  }
},
{
  "$limit": 10
},
{
  "$project": {
    "AvgAgePerBrand": {
      "$round": ["$AvgAgePerBrand",1]
    }
  }
}]') # top ten models with the highest average age
```

	_id.brand	AvgAgePerBrand
1	Bsa	73.9
2	Norton	70.8
3	Horex	69.7
4	Victoria	69.0
5	Nsu	66.7
6	Adler	66.0
7	Heinkel	62.8
8	Kuberg	62.0
9	Dkw	61.4
10	Maico	58.5

Task 2.9. (Optional)

By using, again, “regex” command we can source the motorcycles that include the “ABS” characteristic in the “extras” category and in any format (“options: i”) and by making a group by we calculate the number of those. So, we can see that there are 4,025 bikes with “ABS”.

```
m$aggregate(
  '[{
  "$match": {
    "extras": {
      "$regex": "ABS",
      "$options": "i"
    }
  }
},
{
  "$group": {
    "_id": null,
    "count": {
      "$sum": 1
    }
  }
}]')  # 4,025 motorcycles have "ABS" as an extra
```

The output is the following:

```
  _id count
1  NA  4025
```

Task 2.10. (Optional)

To calculate the average Mileage of bikes that have "ABS" and "Led lights" as an extra, we use the following commands and we find that it is 30.125,7 km.

```
m$aggregate(
  '[{
  "$match": {
    "extras": {
      "$all" : ["ABS", "Led lights"]
    }
  }
},
{
  "$group": {
    "_id": null,
    "AvgMileage": {
      "$avg": "$ad_data.Mileage"
    }
  }
}]')  #the average Mileage of bikes that have "ABS" and "Led lights" as an extra is 30,125.7 km
```

The output is the following:

```
  _id AvgMileage
1  NA    30125.7
```