Students:
Christos Vlassis, **AM**: f2822204

Task 1 Redis

1.1

In order to import the data on redis we created a loop and inserted the data. Then, we counted and found 9969 users which had modified their lists.

1.2

To find the users that did not modify their listing in January we used, firstly the BITOP NOT and later BITCOUNT. We found 10031 users that did not modify their lists. This number is not correct, we should have found 10030 users.

1.3

To find the users that received at least one email per month, we created three different bitmaps. Each bitmap contains on the ‘offset’ the ID of the user and we use ‘1’ if the user has received one email for the respective month. Finally, we used the BITOP AND to find the users that had one email per month. We found 2668 users.

1.4

To find the users that received an email in January and March but not in February we performed one bitwise operation with AND to find the users that received email in March and January. Secondly, we made an inversion for the bitmap that contains the users that received an email in February. Finally, we performed a BITOP AND operation for those two bitmaps and found 2417 users.

1.5

To find the users that received an email in January that they did not open but updated their list we created two separate data frames which later were merged. Firstly, we selected the data only for January and those users that had opened at least one email. For the second data frame, we kept only the data for the January and for those users that had modified their lists. We merged the two data frames and we imported that data to redis using the IDs as the offset. We found 2417 users.

1.6

To solve this Task, we used the same commands and logic as the 1.5 Task. We did the same thing for February and March. We imported the data to redis and we used the OR operator. We found 5249 users.

1.7

To evaluate if the strategy works we will create the following metrics:

- Y = Users to whom we send an email, they opened it and modified their list, 10704
- Sample_space = users to whom we send an email and they opened it, 12196

We create the percentage : perc = 10704/12196 * 100 , perc = **87%**
We can conclude that 87% of the users that received and opened the email have made a modification. I believe that the strategy works and should NOT be removed from the company's plan!

Also, we find that **66%** of the users to whom we send an email have made a modification.

Redis, R Code

R Code for Redis:
Note: R Code should be copied and pasted in R studio to have a better visual representation.

```
###################################################################
library(dplyr)
library(redux)



#Remote Connection
r <-
redux::hiredis( redux:
:redis_config(
   host = "redis-10013.c250.eu-central-1-1.ec2.cloud.redislabs.com",
   port = "10013",
   password = "sm7JxN8ruUE9gAwoQkAe9GSPumYPZJgY"))

###################################################################
###################################################################
#######
############################################### 1.1    How many users
modified their listing on January?
###############################################
###################################################################
###################################################################
#######
# We keep only the data for January:
new_mod_list <- subset(modified_listings, MonthID == 1)
# we keep only the unique rows. We do this because some users may have
opened an email more than once:
new_mod_list<-unique(new_mod_list)

# We create a loop to take only the relevant information.
# And we import the data to redis

y <- 0
```

```
for (i in 1:nrow(new_mod_list)) { x <-
  new_mod_list[i,3]
  r$SETBIT("ModificationsJanuary",y,x)
  y <- y + 1
}



# And we count:
r$BITCOUNT("ModificationsJanuary") #  9969 users have modified their list on
January
```

```
#################################################################
#################################################################
#######
############################################# 1.2    How many users
did NOT modify their listing on January?
#############################################
#################################################################
#################################################################
#######
# We count the Zeros for the ModificationsJanuary BITMAP
r$BITOP('NOT','Zeros','ModificationsJanuary')
r$BITCOUNT("Zeros") #  10031 Non Zeros
```

```
################################################################
################################################################
#############
############################################# 1.3    How many users
received at least one e-mail per month
#############################################
################################################################
################################################################
######

# Lets keep only the relative columns:
emails_send_reduced <- subset(emails_sent, select = -EmailID)

######################################################  January
######################################################


jan_data <- subset(emails_send_reduced, MonthID == 1, select = UserID)
jan_data <- unique(jan_data)

for (i in 1:nrow(jan_data)) { y <-
  jan_data[i,1]
  r$SETBIT("EmailsJanuary",y,'1')
}

r$BITCOUNT("EmailsJanuary") # 9617 emails were received by users for
January


######################################################  February
######################################################
```

```
feb_data <- subset(emails_send_reduced, MonthID == 2, select = UserID)
feb_data <- unique(feb_data)

for (i in 1:nrow(feb_data)) { y <-
  feb_data[i,1]
  r$SETBIT("EmailsFebruary",y,'1')
}

r$BITCOUNT("EmailsFebruary") #  9666 emails were received by users for
February



###################################################### March
######################################################

march_data <- subset(emails_send_reduced, MonthID == 3, select = UserID)
march_data <- unique(march_data)

for (i in 1:nrow(march_data)) { y
  <- march_data[i,1]
  r$SETBIT("EmailsMarch",y,'1')
}

r$BITCOUNT("EmailsMarch") # 9520 emails were received by users for February


# We perform a bitwise AND operation:
r$BITOP("AND","results",c("EmailsJanuary","EmailsFebruary", 'EmailsMarch'))

r$BITCOUNT("results") # 2668 have received at least one email per month !!



################################################################
################################################################
#############
```

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

```
# We perform a bitwise AND operation for :
r$BITOP("AND","Jan_March_And",c("EmailsJanuary","EmailsMarch"))

r$BITCOUNT("Jan_March_And")

# We make an inversion for the February. That way we will have '1's when the user did NOT received an email:
r$BITOP('NOT','inv_Feb','EmailsFebruary')

# We use the and to take only those users that had emails on January and March but NOT on February
r$BITOP("AND","March_Jan_Not_Febr",c("Jan_March_And","inv_Feb"))

# We count:
r$BITCOUNT("March_Jan_Not_Febr") # 2417 users received emails on 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?
###
##
##
######

We will have to find those users that had ONLY zeros on the EmailOpened column and zeros on the ModifiedListing for January(1)

First i keep only the data that i need for the table emails_sent:

```
emails_sent_vol2 <- subset(emails_sent,  MonthID == 1, select =
-c(EmailID,MonthID))
```

```
# i have to remove all users that had ATLEAST one zero:
# here we have these users:
users_to_be_removed <- (subset(emails_sent_vol2, EmailOpened == 1, select =
UserID))
```

```
# We remove them from the data frame:
emails_sent_clean <- emails_sent_vol2[!(emails_sent_vol2$UserID %in%
users_to_be_removed$UserID), ]
```

```
# And we keep the unique values:
emails_sent_clean <- unique(emails_sent_clean)
```

We continue with the second data frame. We need to remove the users that have '1' in the modified listing column:

```
# We first clean the data frame:
modified_listings_clean <- subset(modified_listings, MonthID == 1 &
ModifiedListing == 1, select = -MonthID)
```

Now that the data frames are ready, we will merge them:

```
merged_df <- merge(emails_sent_clean, modified_listings_clean, by = "UserID",
all = FALSE)
```

now we continue with redit:

```
for (i in 1:nrow(merged_df)) { y <-
  merged_df[i,1]
  r$SETBIT("not_opened_updated_Jan",y,'1')
}
```

We count:
r$BITCOUNT("not_opened_updated_Jan") # 1961 users have received an email on Jan that they did not open, BUT they updated their listing.

##
############## 1.6 How many users received an e-mail on January that they did not open but......
##

We will do the same thing for the other two months !

February

We will have to find those users that had ONLY zeros on the EmailOpened column and zeros on the ModifiedListing for February(2)

First i keep only the data that i need for the table emails_sent:

emails_sent_vol2 <- subset(emails_sent, MonthID == 2, select = -c(EmailID,MonthID))

i have to remove all users that had ATLEAST one zero:
here we have these users:

```
users_to_be_removed <- (subset(emails_sent_vol2, EmailOpened == 1, select =
UserID))

# We remove them from the data frame:
emails_sent_clean <- emails_sent_vol2[!(emails_sent_vol2$UserID %in%
users_to_be_removed$UserID), ]

# And we keep the unique values:
emails_sent_clean <- unique(emails_sent_clean)



# We continue with the second data frame. We need to remove the users that
have '1' in the modified listing column:

# We first clean the data frame:
modified_listings_clean <- subset(modified_listings, MonthID == 2 &
ModifiedListing == 1, select = -MonthID)


# Now that the data frames are ready, we will merge them:

merged_df <- merge(emails_sent_clean, modified_listings_clean, by = "UserID",
all = FALSE)

# now we continue with redit:

for (i in 1:nrow(merged_df)) { y <- merged_df[i,1]
  r$SETBIT("not_opened_updated_February",y,'1')
}

# We count:
r$BITCOUNT("not_opened_updated_February") # 1971 users have received an
email on Feb that they did not open, BUT they updated their listing.
```

March
###
####

We will have to find those users that had ONLY zeros on the EmailOpened column and zeros on the ModifiedListing for March(3)

First i keep only the data that i need for the table emails_sent:

emails_sent_vol2 <- subset(emails_sent, MonthID == 3, select = -c(EmailID,MonthID))

# i have to remove all users that had ATLEAST one zero:
# here we have these users:
users_to_be_removed <- (subset(emails_sent_vol2, EmailOpened == 1, select = UserID))

# We remove them from the data frame:
emails_sent_clean <- emails_sent_vol2[!(emails_sent_vol2$UserID %in% users_to_be_removed$UserID),]

# And we keep the unique values:
emails_sent_clean <- unique(emails_sent_clean)

We continue with the second data frame. We need to remove the users that have '1' in the modified listing column:

# We first clean the data frame:
modified_listings_clean <- subset(modified_listings, MonthID == 3 & ModifiedListing == 1, select = -MonthID)

Now that the data frames are ready, we will merge them:

```
merged_df <- merge(emails_sent_clean, modified_listings_clean, by = "UserID",
all = FALSE)

# now we continue with redit:

for (i in 1:nrow(merged_df)) { y <-
  merged_df[i,1]
  r$SETBIT("not_opened_updated_March",y,'1')
}

# We count:
r$BITCOUNT("not_opened_updated_March") # 1966 users have received an
email on March that they did not open, BUT they updated their listing.


# Finally we will use BITOP OR
r$BITOP("OR","Jan_March_And_not_opened_updated",c('not_opened_updated_
Jan',"not_opened_updated_February","not_opened_updated_March"))

# We count
r$BITCOUNT("Jan_March_And_not_opened_updated") # 5249 users have ....
either blablabla .... either blablabla .... either blablabla


##########################################################################
##########################################################################
###################################### 1.7       Does it make any sense to
keep sending e-mails .... ############################################
##########################################################################
##########################################################################



# We want to find out if the emails 'create' modifications on the listings, so we will
create the following metrics
# to evaluate the performance of our strategy
```

To understand if this strategy works we will create the following metrics to evaluate the performance of the strategy:
We will find the users to whom we send an email, they opened it and modified their list (y)
we will find the users to whom we send an email and they opened it (sample_space)

Because an email can have been send more than one time to each user, we will find the unique users to calculate the above metrics

# Users to whom we send an email and they opened it (sample_space)
sample_space_df<- unique(subset(emails_sent, EmailOpened == 1 , select = UserID))
sample_space <-nrow(sample_space_df) # 12196 from 16006 to whom we send an email they opened it.

lets find the total users to whom we send an email
nrow(unique(subset(emails_sent, select = UserID))) # To 16006 users we have send emails.

Firstly, we need to clean the modified_listings data frame and then merge it with the sample_space

# Cleaning:
mod_list <- subset(modified_listings, select = -MonthID)

# lets keep only the users that modified their list:
mod_list <- subset(mod_list, ModifiedListing == 1, select = -ModifiedListing)
mod_list <- unique(mod_list)

# Lets merge:
merged_df <- merge(mod_list, sample_space_df, by = "UserID")
nrow(merged_df) # 10704 users that received an email have maid a modification in their lists

perc<- 10704/12196 * 100 # 87% of the users that received and opened the email have maid a modification
# i believe that the strategy works and should NOT be removed from the companys plan!
perc_2 <- 10704/16006 * 100 # 66% of the total users that we send an email have made a modification

mod_list$UserID

Task 2, MongoDB

We use the following command on the PowerShell to login in our database:
C:\Program Files\MongoDB\Server\4.0\bin> .\mongo.exe --quiet

Our file paths are located here: "C:\
DATA_FOR_MONGO_ASSIGMENT\BIKES\Paths_to_files.txt"

To get the file paths we used the following command, after we have used the cd to go to the directory were our files are located:
dir -Recurse -Name -File > Paths_to_files.txt

We had to remove the first line, since it contained the headline of the file.We used the following command:
Get-Content Paths_to_files.txt | Select-Object -Skip 1 | Out-File
Paths_to_files_2.txt -Encoding utf8

In this stage our Paths_to_files_2.txt contains the following information(i give an example of the first lines):
10\00\06\10000682.json
10\00\60\10006063.json
10\00\88\10008842.json
10\00\96\10009608.json
10\01\10\10011059.json
We need to include the full path because, as you can see, we don't have the full path in the text file. To do that we use the Powershell:

With this command i added the **full path** and created the full_paths txt file:
(Get-Content Paths_to_files_2.txt) -replace '^',
'C:\DATA_FOR_MONGO_ASSIGMENT\BIKES\' | Out-File Paths_to_files_3.txt
-Encoding utf8

Now we import the data to R in order to do some cleaning.Firstly, for the Price, we use NA for those dictionaries that contain alphabetic characters. We also remove the euro sign and we change the data type to numeric. Secondly, we find those bikes that their price is negotiable. This information can be found in the bike_data$metadata$model. For those bikes which are negotiable we create a new attribute called 'Negotiable' and we set it to TRUE if the price is Negotiable. We also create the Age of the bike by substring the current year from the Registration year. Also, we clean the Mileage attribute to contain only numeric values. Finally, we import the data into MongoDB. The relative code can be found in the last pages of this Report.

We will use the UI of R to complete all the assignments

2.2

By simply counting the number of documents we found **29701 bikes**.

2.3

We calculated the average price of the bikes, which was found to be 2962€ for the 29701 bikes that we had in our database. **But** this number is misleading.

From the 2.4 assignment we found that many bikes had a min price of 1€ and two bikes had a price of 89000€. For those bikes with a price of 1€ we can conclude that the data are wrong and maybe we can consider them as 'Negotiable'. Also, after some research, we found that many bikes that are for sale are just being sold for their parts and they are not operational. For the two bikes with high prices we found that the model of the bike is Bmw HP4. These bikes are racing bikes and after some research we found that the price is reasonable for this kind of models.

With the above being said, we **recalculated** the average price and we excluded bikes that cost less than 300€. So the new and more realistic results
are the following. **An average price of 3074€ for 28049 bikes.**

2.4

We found a **max** price of **89000€**. As it was mentioned before these are the racer bikes which have very high prices. For the **min** prices, we found many bikes with a price of **1€**. As it was mentioned before these might be considered as 'Negotiable' but we cannot be certain of this.

2.5

To solve this task we simply counted the frequency of the 'Negotiable' attribute and we found **1348 bikes** with negotiable prices.

2.7

To solve this Task we found the Average Price per brand, sorted and kept only the first ‘line‘. We found that the **Semog brand** had the higher average price.

2.9

To solve this task we filtered from the ‘extras’ array only those bikes with ‘ABS’. We found **4025 bikes** with ABS.

2.6

To solve this task we, first, calculated the total number of bikes per brand and those bikes that are considered as negotiable. Then we made the following calculation to find the percentage of bikes that are considered as negotiable, for each brand.

Number of Negotiable Bikes / Total Number of Bikes * 100

MongoDB, R Code

R Code for Redis:

Note: R Code should be copied and pasted in R studio to have a better visual representation.

```
# Load mongolite
install.packages("mongolite")
library("mongolite")
install.packages("jsonlite")
library("jsonlite")
install.packages("stringr")
library("stringr")

# Open a connection to MongoDB
m <- mongo(collection = "main_data",  db = "assigment_1", url =
"mongodb://localhost")

# read the file paths: files_list<- read.delim("C:\\
DATA_FOR_MONGO_ASSIGMENT\\BIKES\\Paths_to_files_3.txt ", header =
FALSE)


# Lets see how the data are presented:
example_1 <- fromJSON(readLines(files_list[1,]))
example_2 <- fromJSON(readLines(files_list[2,]))
example_3 <- fromJSON(readLines(files_list[3,]))
########################################################### 2.1       Add
your data to MongoDB.
##################################################################
########################################################### Cleaning and
importing the data:
###############################################################
```

```
for (i in 1:nrow(files_list)) {
  bike_data <- fromJSON(readLines(files_list[i,], encoding = "UTF-8"))
  # if the price is Negotiable we create a new Attribute:
  if (grepl("Negotiable", bike_data$metadata$model, ignore.case = TRUE)) {
    bike_data$metadata$Negotiable <- TRUE
  }
  # if price contains letters, set it to NA. if it contains any non-numeric characters
remove them and set price to numeric
  if (grepl("[a-z]", bike_data$ad_data$Price)) {
    bike_data$ad_data$Price <- NA
  }
  else {
    bike_data$ad_data$Price <- as.numeric(gsub("[^0-9]+", "",
bike_data$ad_data$Price))
  }
  # We create the Age of the bike. from current year we substract the year of the
Registration
  bike_data$ad_data$Age <- as.numeric(format(Sys.Date(), "%Y")) -
as.numeric((strsplit(bike_data$ad_data$Registration, " / "))[[1]][[2]])
  # Clean and make mileage numeric:
bike_data$ad_data$Mileage <- as.numeric(gsub("[^0-9]", "",
bike_data$ad_data$Mileage))
  # we change the format to JSON
  bike_data <- toJSON(bike_data, auto_unbox = TRUE)
  # We insert the data
  m$insert(bike_data)
  print(i)
}


########################################## 2.2    How many bikes are
there for sale?  #######################################################
m$count() # 29701 bikes are there for sale
```

```
############################################ 2.3    What is the average price of a motorcycle ?
#########################################################

# We found 2962.701 the average price of a motorycle. With 29701 being the total number of bikes(number of listings) in our database
m$aggregate('[{"$group": {"_id": null,"Average_Price": {"$avg": "$ad_data.Price"},"number_of_bikes": {"$sum": 1}}}]')


# Note: On question 2.4 we found some Bikes with a Price of 1.
# This means that some bikes have wrong values and they should be excluded from the calculation of the average price.
# Maybe, but we cannot be sure about that, when the Price is equal to 1 it means that the price of the bike is negotiable.


m$aggregate('[{"$match": {"ad_data.Price": {"$gt": 300}}},
          {"$group": {"_id": null,"Average_Price": {"$avg": "$ad_data.Price"},"number_of_bikes": {"$sum": 1}}}]')

# Now that the prices have been filtered to be more realistic we found an Average price of 3074 for 28049 bikes !


############################################ 2.4    What is the maximum and minimum price of a motorcycle...?
#########################################################

# We find the max price of a bike to be 89000 and the min price equal to 1.
m$aggregate('[{"$group":{"_id":null, "Max_Price":{"$max":"$ad_data.Price"}, "Min_Price":{"$min":"$ad_data.Price"}}}]')

# Both numbers don't make sense. Lets investigate and find out those listings with these Prices

# For the MAX Price:
```

```
# the two bikes with this price are Bmw HP4. Which ,on internet, can be found for a price of 80000 plus on average.
# We can conclude that the prices make sense.
# These bikes are racer type bikes and are very expensive!
m$find('{"ad_data.Price":{ "$eq": 89000}}', '{ "ad_data.Make/Model": 1, "_id": 0 }')


# For the MIN Price:
# We can find many listings with a price of 1.
# These models cost a lot more in the market and we should NOT take these values into consideration
# In order to find out the reason behind this price we browsed the car.gr site
# We found out that many of the listings contain bikes that are sold only for their spare parts. These bikes have very low prices.
m$find('{"ad_data.Price":{ "$eq": 1}}', '{ "ad_data.Make/Model": 1, "_id": 0,
"ad_data.Price": 1 }')


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

# Found 1348 bikes with negotiable price
m$aggregate('[{"$match": {"metadata.Negotiable": {"$eq": true}}},
        {"$group": {"_id": null,"number_of_bikes": {"$sum": 1}}}]')


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

# these are our brands!
m$distinct('metadata.brand')


# The first line finds the Average Price for each brand
# The second line renames the _id to Brand
```

The third line sorts in descending order the average price
The forth line gives us the max average price of the brand. (the first line)
m$aggregate('[{"$group": {"_id": "$metadata.brand", "Average_Price": {"$avg": "$ad_data.Price"}}},
	{"$project": {"Brand": "$_id", "Average_Price": 1, "_id": 0}},
	{"$sort": {"Average_Price": -1}},
	{"$limit": 1}]') # We find that Semogs have the higher average price of 15600 euros.

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

m$find('{"extras": {"$eq": "ABS"}}', '{"extras": 1, "_id": 0}')

The first line filters the data to bring only information for these documents that in 'extras' array contain ABS.
The second line counts the number of bikes that have ABS
The third line keeps only the counter. (We remove the _id)
m$aggregate('[{"$match": {"extras": {"$eq": "ABS"}}},
	{"$group": {"_id": null, "Number_of_Bikes_with_ABS": {"$sum": 1}}},
	{"$project": {"_id": 0}}]') # 4025 bikes have ABS

##################### 2.6 (Optional) For each Brand, what percentage of its listings is listed as negotiable?
##

The first line brings the total number of bikes for each brand.
The second line brings the number of bikes with negotiable price per brand.
The third line keeps as an output the brand, neg_bikes and number_of_bikes.
The forth line does the following calculation neg_bikes/number_of_bikes * 100.

```
m$aggregate('[{"$group": {"_id": "$metadata.brand","number_of_bikes": {"$sum":
1},
	"neg_Bikes": {"$sum": {"$cond": {"if": {"$eq":
["$metadata.Negotiable",true]}, "then": 1, "else": 0}}}}}},
	{"$project": {"_id": 1, "neg_Bikes": 1, "number_of_bikes": 1,
	"Neg_Bikes_Perc": {"$multiply": [{"$divide": ["$neg_Bikes",
"$number_of_bikes"]}, 100] }}}]')
```