ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Master of Science (MSc) in Business Analytics

Big Data Systems and Architectures - FT

-Redis and Mongodb Assignment-

Instructor: Safras Spyridon

Students Names:

Eleni Ralli (Student ID: f2822312)

Viktor Kalatzis (Student ID: f2822318)

March, 2024

Contents

Data

For the completion of the assignment, we downloaded these datasets:

- We Download the BIKES_DATASET.zip from Google Drive. This dataset contains around 30,000 classified ads from the used motorcycles market.
- We Download the RECORDED_ACTIONS.zip from Google Drive. This dataset includes data on seller-related actions tracked over the previous months.

Scenario

As data analysts at a leading consulting firm, we have access to comprehensive datasets from the used motorcycles market. These datasets include classified advertisements for used motorcycles and track actions taken by sellers in recent months. Our objective is to utilize this information to carry out in-depth analyses and address the requirements outlined in the "TASKS" section.

Redis

We implement code in R to complete our tasks.

```
> # Load the redux package into the R session.
> # The redux package provides a set of functions to interact with Redis databases using the hiredis library.
> library("redux")
>
> # Establishing a local connection to a Redis server
> # The `hiredis` function from the redux package is used to create a connection object.
> # This connection object will be used for subsequent Redis operations (commands).
> # The `redis_config` function is used to specify the configuration settings for the connection.
> # Here, 'host' is set to "127.0.0.1" which is the IP address for localhost, indicating that
> # the Redis server is running on the same machine as the R session.
> # The 'port' is set to "6379", which is the default port number that Redis servers listen on.
> r <- redux::hiredis(
+   redux::redis_config(
+     host = "127.0.0.1",
+     port = "6379"))
>
> # The result, 'r', is a Redis connection object that you can use to interact with the Redis server.
> # For example, to set a key-value pair, retrieve data, or perform other Redis commands.
>
>
> # Installation of necessary packages (if they have not already been installed)
> #if (!require(bit64)) install.packages("bit64")
> #if (!require(dplyr)) install.packages("dplyr")
>
> # Loading packages
> library(bit64)
> library(dplyr)
>
>
> # Loading data from CSV
> # Loads the 'emails_sent.csv' dataset into R
> emails_sent <- read.csv("D:\\redis\\ergasia_redis\\RECORDED_ACTIONS\\emails_sent.csv")
> # Loads the 'modified_listings.csv' dataset into R
> modified_list <- read.csv("D:\\redis\\ergasia_redis\\RECORDED_ACTIONS\\modified_listings.csv")
>
```

1.1)

```
> #1.1
>
> # Subsetting 'modified_list' to include only entries from January
> # Selecting only 'UserID' and 'ModifiedListing' columns
> January <- subset(modified_list, MonthID == 1, select=c(UserID, ModifiedListing))
>
> # Subsetting 'modified_list' to include only entries from February
> # Selecting only 'UserID' and 'ModifiedListing' columns
> February <- subset(modified_list, MonthID == 2, select=c(UserID, ModifiedListing))
>
> # Subsetting 'modified_list' to include only entries from March
> # Selecting only 'UserID' and 'ModifiedListing' columns
> March <- subset(modified_list, MonthID == 3, select=c(UserID, ModifiedListing))
>
> # Loop through each row of the 'January' data frame
> for (i in 1:nrow(January)){
+   # For each user (based on 'UserID'), set a bit at the position of 'ModifiedListing' value
+   # in the bitmap named "ModificationsJanuary" in Redis.
+   # This creates or modifies a bitmap for each user where each bit represents whether
+   # a modification occurred (1) or not (0).
+   r$SETBIT("ModificationsJanuary", January[i,1], January[i,2])
+ }
>
> # Count and return the number of set bits (value of 1) in the bitmap "ModificationsJanuary".
> # This provides the total number of modifications that happened in January.
> r$BITCOUNT("ModificationsJanuary")
[1] 9969
>
>
>
> #1.2
>
> # Perform a bitwise NOT operation on the bitmap "ModificationsJanuary" and store the result in "JanuaryNotModified".
> # The NOT operation will invert the bits in the bitmap: bits that are 0 become 1, and bits that are 1 become 0.
> r$BITOP("NOT", "JanuaryNotModified", "ModificationsJanuary")
[1] 2500
>
> # Count and return the number of set bits (value of 1) in the bitmap "JanuaryNotModified".
> # After the NOT operation, this count will represent the number of "non-modifications" or the inverse of the original modifications bitmap for January.
> r$BITCOUNT("JanuaryNotModified")
[1] 10031
>
>
>
```

1.2)

```
> #1.2
>
> # Perform a bitwise NOT operation on the bitmap "ModificationsJanuary" and store the result in "JanuaryNotModified".
> # The NOT operation will invert the bits in the bitmap: bits that are 0 become 1, and bits that are 1 become 0.
> r$BITOP("NOT", "JanuaryNotModified", "ModificationsJanuary")
[1] 2500
>
> # Count and return the number of set bits (value of 1) in the bitmap "JanuaryNotModified".
> # After the NOT operation, this count will represent the number of "non-modifications" or the
> #inverse of the original modifications bitmap for January.
> r$BITCOUNT("JanuaryNotModified")
[1] 10031
>
```

1.3)

```
> #1.3
> # Load the dplyr package for data manipulation
> library("dplyr")
>
> # Select distinct rows from the 'emails_sent' data frame based on the UserID and MonthID
> # Keep all columns from the second to the fourth
> emails <- emails_sent[, 2:4] %>% distinct(UserID, MonthID, .keep_all = TRUE)
>
> # Subset the 'emails' data frame for each of the first three months of the year
> EmailsJanuary <- subset(emails, MonthID == 1)
> EmailsFebruary <- subset(emails, MonthID == 2)
> EmailsMarch <- subset(emails, MonthID == 3)
>
> # Insert data into Redis. For each month, set a bit for each UserID that received an email
> # The loop goes through each row of the January emails and sets a bit in Redis
> for (i in 1:nrow(EmailsJanuary)) {
+   r$SETBIT("EmailsSentJanuary", EmailsJanuary$UserID[i], "1")
+ }
> # Repeat the process for February
> for (i in 1:nrow(EmailsFebruary)) {
+   r$SETBIT("EmailsSentFebruary", EmailsFebruary$UserID[i], "1")
+ }
> # And repeat the process for March
> for (i in 1:nrow(EmailsMarch)) {
+   r$SETBIT("EmailsSentMarch", EmailsMarch$UserID[i], "1")
+ }
>
> # Perform a bitwise AND operation on the three Redis keys (bitmaps) representing
> # emails sent in January, February, and March, respectively
> # Store the result in a new Redis key named "Task 1.3"
> r$BITOP("AND", "Task 1.3", c("EmailsSentJanuary", "EmailsSentFebruary", "EmailsSentMarch"))
[1] 2500
>
> # Count and return the number of set bits in the "Task 1.3" bitmap
> # This represents the number of users who received an email in all three months
> r$BITCOUNT("Task 1.3")
[1] 2668
>
> # Print out the count of users who received at least one email each month
> print(paste("The number of users who received at least one email every month is:", r$BITCOUNT("Task 1.3")))
[1] "The number of users who received at least one email every month is: 2668"
```

1.4)

```
> #1.4
>
> # Perform a bitwise NOT operation on the bitmap for emails sent in February, creating an inverted bitmap.
> # In the inverted bitmap, a set bit (1) will now represent a user who did NOT receive an email in February.
> r$BITOP("NOT", "InvertedEmailsSentFebruary", "EmailsSentFebruary")
[1] 2500
>
> # Perform a bitwise AND operation between the bitmaps for emails sent in January, the inverted bitmap for February,
> # and the bitmap for emails sent in March. This will result in a bitmap where a set bit represents users who
> # received emails in January and March but NOT in February.
> r$BITOP("AND", "Task 1.4", c("EmailsSentJanuary", "InvertedEmailsSentFebruary", "EmailsSentMarch"))
[1] 2500
>
> # Count and return the number of set bits in the "Task 1.4" bitmap.
> # This count represents the number of users who received an email in January and March but NOT in February.
> r$BITCOUNT("Task 1.4")
[1] 2417
>
> # Print out the result with a descriptive message.
> # This prints the number of users who met the criteria of receiving an email in January and March but not in February.
> print(paste("The number of users who received an email in January and March but NOT in February is:", r$BITCOUNT("Task 1.4")))
[1] "The number of users who received an email in January and March but NOT in February is: 2417"
```

1.5)

```
> #1.5
>
>
> # Subset 'emails_sent' for January only and select the 'UserID' and 'EmailOpened' columns
> EmailsJanuary_agg <- subset(emails_sent, MonthID == 1, select = c(UserID, EmailOpened))
>
> # Aggregate the data by 'UserID' and sum up the 'EmailOpened' column
> # This gives us the total number of emails opened by each user in January
> EmailsJanuary_agg <- aggregate(EmailsJanuary$EmailOpened, by = list(UserID = EmailsJanuary$UserID), FUN = sum)
>
> # Convert the 'x' column to a binary indicator: 1 if any email was opened, 0 if no emails were opened
> EmailsJanuary_agg$x <- if_else(EmailsJanuary_agg$x == 0, 0, 1)
>
> # Loop through the aggregated January emails and set a bit for each user ID based on whether they opened any emails
> # We're using the result of the aggregation to set the bits in the bitmap "EmailsOpenedJanuary" in Redis
> for (i in 1:nrow(EmailsJanuary_agg)) {
+   r$SETBIT("EmailsOpenedJanuary", EmailsJanuary_agg$UserID[i], EmailsJanuary_agg$x[i])
+ }
>
> # Perform a bitwise NOT operation on "EmailsOpenedJanuary" to get "EmailsNotOpenedJanuary"
> # In the resulting bitmap, a set bit will now represent a user who did NOT open an email in January
> r$BITOP("NOT", "EmailsNotOpenedJanuary", "EmailsOpenedJanuary")
[1] 2500
>
> # Perform a bitwise AND operation between "EmailsNotOpenedJanuary" and "ModificationsJanuary"
> # The resulting bitmap, "Task 1.5", will represent users who did not open any emails and who updated their listing in January
> r$BITOP("AND", "Task 1.5", c("EmailsNotOpenedJanuary", "ModificationsJanuary"))
[1] 2500
>
> # Count and return the number of set bits in the "Task 1.5" bitmap
> # This count represents the number of users who received but did not open an email and also made a listing update
> r$BITCOUNT("Task 1.5")
[1] 7577
>
> # Print out the count with a descriptive message
> print(paste("The number of users who received but did not open an email in January and updated their listing is:", r$BITCOUNT("Task 1.5")))
[1] "The number of users who received but did not open an email in January and updated their listing is: 7577"
>
```

1.6)

```
> #1.6
>
> # For February, subset the 'emails_sent' data frame to get user IDs and whether they opened emails.
> EmailsFebruary_agg <- subset(emails_sent, MonthID == 2, select = c(UserID, EmailOpened))
> # Aggregate by UserID to sum up EmailOpened values, which indicates whether a user opened any email.
> EmailsFebruary_agg <- aggregate(EmailsFebruary$EmailOpened, by = list(UserID = EmailsFebruary$UserID), FUN = sum)
> # Convert the sum into a binary indicator where 0 remains 0 and any positive number becomes 1.
> EmailsFebruary_agg$x <- if_else(EmailsFebruary_agg$x == 0, 0, 1)
>
> # For March, repeat the process of subsetting and aggregating email open data.
> EmailsMarch_agg <- subset(emails_sent, MonthID == 3, select = c(UserID, EmailOpened))
> EmailsMarch_agg <- aggregate(EmailsMarch$EmailOpened, by = list(UserID = EmailsMarch$UserID), FUN = sum)
> EmailsMarch_agg$x <- if_else(EmailsMarch_agg$x == 0, 0, 1)
>
> # For each user in February, set a bit in Redis based on whether they modified their listing.
> for (i in 1:nrow(February)) {
+   r$SETBIT("ModificationsFebruary", February$UserID[i], February$ModifiedListing[i])
+ }
>
> # For each user in February, set a bit in Redis based on whether they opened an email.
> for (i in 1:nrow(EmailsFebruary_agg)) {
+   r$SETBIT("EmailsOpenedFebruary", EmailsFebruary_agg$UserID[i], EmailsFebruary_agg$x[i])
+ }
>
> # Perform a NOT operation on February's opened emails to track emails not opened.
> r$BITOP("NOT", "EmailsNotOpenedFebruary", "EmailsOpenedFebruary")
[1] 2500
> # Perform an AND operation to find users who did not open emails and modified listings in February.
> r$BITOP("AND", "February", c("EmailsNotOpenedFebruary", "ModificationsFebruary"))
[1] 2500
> # Count the number of users who fit the criteria in February.
> r$BITCOUNT("February")
[1] 7528
>
> # Repeat the process for March for modifications and email openings.
> for (i in 1:nrow(March)) {
+   r$SETBIT("ModificationsMarch", March$UserID[i], March$ModifiedListing[i])
+ }
> for (i in 1:nrow(EmailsMarch_agg)) {
+   r$SETBIT("EmailsOpenedMarch", EmailsMarch_agg$UserID[i], EmailsMarch_agg$x[i])
+ }
> r$BITOP("NOT", "EmailsNotOpenedMarch", "EmailsOpenedMarch")
[1] 2500
> r$BITOP("AND", "March", c("EmailsNotOpenedMarch", "ModificationsMarch"))
[1] 2500
> r$BITCOUNT("March")
[1] 7635
>
> # Perform an OR operation between the results of January (from previous script), February, and March.
> r$BITOP("OR", "Task 1.6", c("Task 1.5", "February", "March"))
[1] 2500
> # Count the number of users who fit the OR criteria across all three months.
> r$BITCOUNT("Task 1.6")
[1] 15152
>


>
> # Print the final count of users who received but did not open emails and updated their listings across the three months.
> print(paste("The number of users who received but did not open an email in January and updated their listing is:", r$BITCOUNT("Task 1.6")))
[1] "The number of users who received but did not open an email in January and updated their listing is: 15152"
>
```

1.7)

```
> #1.7
>
> # Perform a bitwise AND operation between the bitmap for users who opened emails in January and
> # the bitmap for users who modified their listings in January.
> # The resulting bitmap "OpenedModJan" will have bits set only for users who did both.
> r$BITOP("AND", "OpenedModJan", c("EmailsOpenedJanuary", "ModificationsJanuary"))
[1] 2500
>
> # Perform a bitwise AND operation between the bitmap for users who opened emails in February and
> # the bitmap for users who modified their listings in February.
> # The resulting bitmap "OpenedModFeb" will have bits set only for users who did both.
> r$BITOP("AND", "OpenedModFeb", c("EmailsOpenedFebruary", "ModificationsFebruary"))
[1] 2500
>
> # Perform a bitwise AND operation between the bitmap for users who opened emails in March and
> # the bitmap for users who modified their listings in March.
> # The resulting bitmap "OpenedModMarch" will have bits set only for users who did both.
> r$BITOP("AND", "OpenedModMarch", c("EmailsOpenedMarch", "ModificationsMarch"))
[1] 2500
>
> # Count the number of bits set in the "OpenedModJan" bitmap. This number represents users who both
> # opened an email and modified their listing in January.
> r$BITCOUNT("OpenedModJan")
[1] 2392
>
> # Count the number of bits set in the "OpenedModFeb" bitmap. This number represents users who both
> # opened an email and modified their listing in February.
> r$BITCOUNT("OpenedModFeb")
[1] 2479
>
> # Count the number of bits set in the "OpenedModMarch" bitmap. This number represents users who both
> # opened an email and modified their listing in March.
> r$BITCOUNT("OpenedModMarch")
[1] 2356
```

Engagement with Emails: When we look at who's opening emails and making changes to their listings, a high count means people are really paying attention to our emails and acting on them.

Impact of Email Campaigns: If we see a lot of users updating their listings in the months when they've opened our emails, it means our email campaigns are working well to get people to be more active.

Potential Overreach: If many users get our emails but don't open them, especially if this happens a lot over several months, it might mean they're tired of our emails. This suggests we might need to better choose who we're sending emails to or change what we're sending.

Optimization Opportunity: There are users who don't open our emails but still update their listings. We should think about other ways to engage with these users, since they're active but not through email.

Business Decision Based on Findings:
Effective Strategy: If opening emails leads to more listing updates, then sending emails is a good move. It shows that emails are motivating people to stay engaged with the platform.

Need for Personalization: If a lot of emails are not being opened, it's a sign we need to make our emails more tailored to our users to get them interested.

Cost-Benefit Analysis: We need to weigh the costs of sending out emails against the benefits of more listings being updated. If sending emails costs less than the benefits we get from more active listings, then it's a cost-effective strategy.

Mongo

In [1]:
```
#!pip install pymongo pandas

import pymongo
import json

# Σύνδεση με την MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["assignment"]
collection = db["assignment_collection"]


files_list_path = "D:/redis/ergasia_redis/BIKES_DATASET/files_list.txt"
try:
    with open(files_list_path, 'r', encoding='utf-16') as file:
        files_list = file.read().splitlines()
except UnicodeDecodeError:
    with open(files_list_path, 'r', encoding='cp1252') as file:
        files_list = file.read().splitlines()

# Εκτυπώστε τον αριθμό των διαδρομών αρχείων που βρέθηκαν για επιβεβαίωση
print(f"Βρέθηκαν {len(files_list)} διαδρομές αρχείων.")
```

```
Βρέθηκαν 29701 διαδρομές αρχείων.
```

In [2]:
```
import json
import os

# Βάση διαδρομής όπου βρίσκονται τα αρχεία JSON
base_path = "D:/redis/ergasia_redis/BIKES_DATASET"

# Αναγνώστης της λίστας διαδρομών αρχείων
files_list_path = "D:/redis/ergasia_redis/BIKES_DATASET/files_list.txt"
with open(files_list_path, 'r', encoding='utf-16') as file:  # Προσαρμόστε την κωδικοποίηση αν χρειάζεται
    files_list = file.read().splitlines()

# Διαδικασία φόρτωσης δεδομένων από κάθε αρχείο JSON
for file_path in files_list:
    full_path = os.path.join(base_path, file_path)  # Πλήρης διαδρομή του αρχείου
    try:
        with open(full_path, 'r', encoding='utf-8') as file:  # Άνοιγμα και ανάγνωση του αρχείου
            data = json.load(file)
            # Εδώ μπορείτε να κάνετε οτιδήποτε θέλετε με τα δεδομένα, π.χ. εκτύπωση
            print(data)  # Εκτυπώνει τα δεδομένα JSON φορτωμένα στην Python
    except Exception as e:
        print(f"Σφάλμα κατά την επεξεργασία του αρχείου {full_path}: {e}")
```

```
{'query': {'url': 'https://www.car.gr/10000682-jawa-350cc-634-638-640', 'type': 'CAR_ADDE', 'trial_count': 1, 'last_p
rocessed': 1553105207}, 'title': "Jawa  350CC 634-638-640 '92", 'ad_id': '10000682', 'ad_data': {'Make/Model': "Jawa
350CC 634-638-640 '92", 'Classified number': '10000682', 'Price': '€10', 'Category': 'Bike - Naked', 'Registration':
'10 / 1992', 'Mileage': '25.000 km', 'Fuel type': 'Gasoline', 'Cubic capacity': '350 cc', 'Power': '25 bhp', 'Color':
```

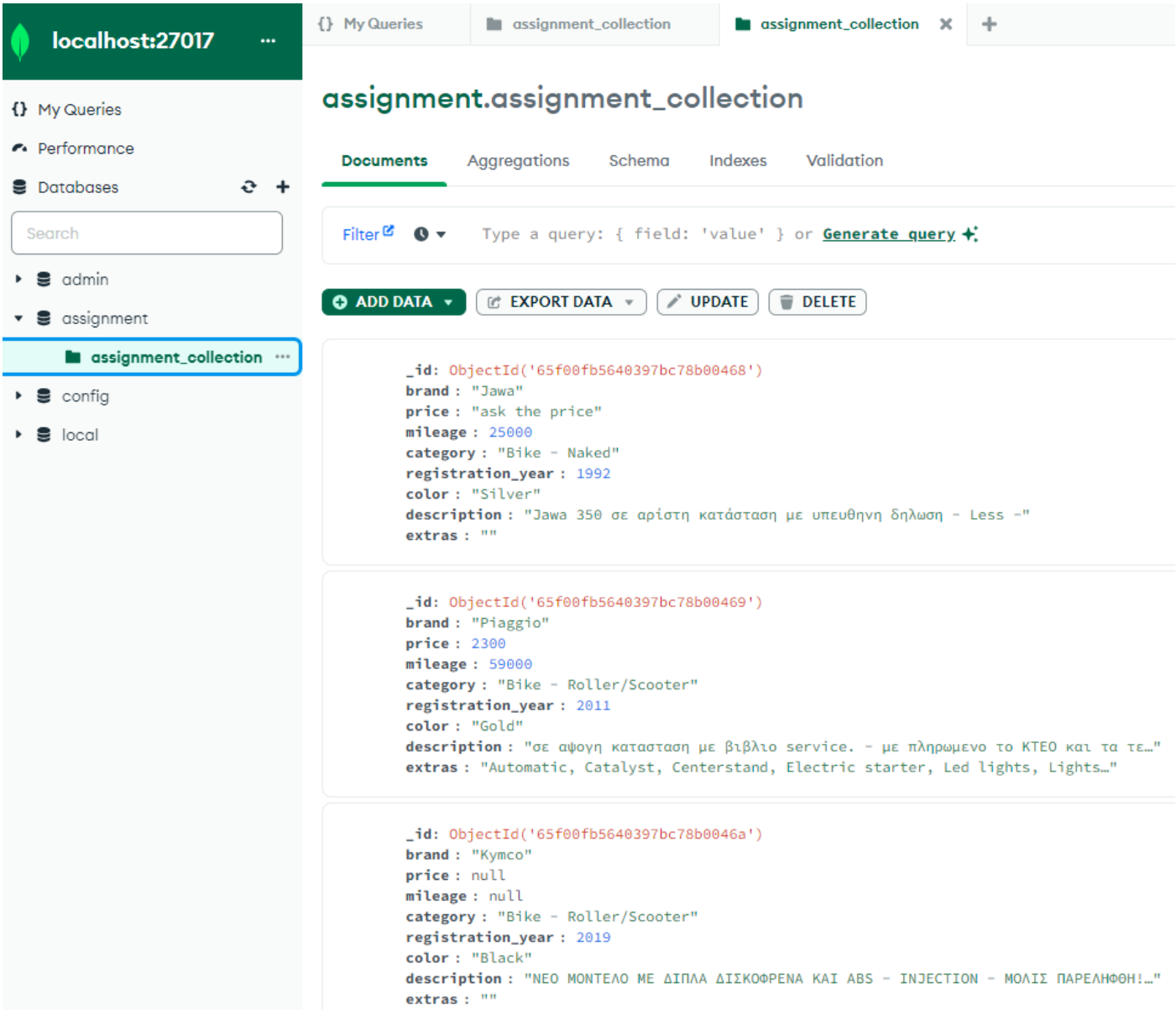
```python
import json
import os
import re
import pymongo

# Σύνδεση με την MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["assignment"]
collection = db["assignment_collection"]

# Βάση διαδρομής όπου βρίσκονται τα αρχεία JSON
base_path = "D:/redis/ergasia_redis/BIKES_DATASET"

# Αναγνώστης της λίστας διαδρομών αρχείων
files_list_path = "D:/redis/ergasia_redis/BIKES_DATASET/files_list.txt"
with open(files_list_path, 'r', encoding='utf-16') as file:
    files_list = file.read().splitlines()

cleaned_data = []  # Λίστα για την αποθήκευση των καθαρισμένων δεδομένων

for file_path in files_list:
    full_path = os.path.join(base_path, file_path)
    try:
        with open(full_path, 'r', encoding='utf-8') as file:
            data = json.load(file)
            # Καθαρισμός τιμής
            price = data["ad_data"].get("Price", "")
            price = re.sub(r'[^\d]', '', price)  # Αφαίρεση μη αριθμητικών χαρακτήρων
            price = int(price) if price else None  # Μετατροπή σε ακέραιο
            if price is not None and price < 150:
                price = "ask the price"

            # Καθαρισμός και προσθήκη Mileage
            mileage = data["ad_data"].get("Mileage", "")
            mileage = re.sub(r'[^\d]', '', mileage)  # Αφαίρεση μη αριθμητικών χαρακτήρων
            mileage = int(mileage) if mileage else None  # Μετατροπή σε ακέραιο

            # Καθαρισμός χρονολογίας εγγραφής
            registration_year = data["ad_data"].get("Registration", "")
            registration_year = re.findall(r'\d{4}', registration_year)
            registration_year = int(registration_year[0]) if registration_year else None

            # Προσθήκη χρώματος
            color = data["ad_data"].get("Color", "").strip()

            # Προσθήκη περιγραφής
            description = data.get("description", "").strip()

            # Μετατροπή extras από λίστα σε string για απλοποίηση
            extras = ', '.join(data.get("extras", []))
```

```
            # Προσθήκη brand
            brand = data["metadata"].get("brand", "").strip()

            cleaned_data.append({
                "brand": brand,
                "price": price,
                "mileage": mileage,
                "category": data["ad_data"].get("Category", ""),
                "registration_year": registration_year,
                "color": color,
                "description": description,
                "extras": extras
            })
    except Exception as e:
        print(f"Σφάλμα κατά την επεξεργασία του αρχείου {full_path}: {e}")

# Εκτυπώστε τα καθαρισμένα δεδομένα για να ελέγξετε
for item in cleaned_data[:5]:  # Εκτυπώνουμε τις πρώτες 5 καταχωρήσεις για επισκόπηση
    print(item)
```

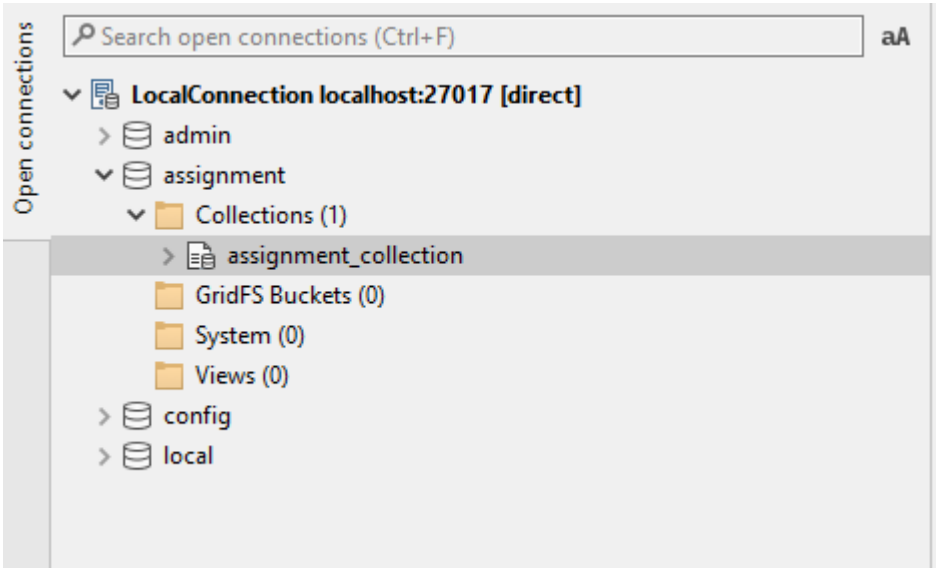
```
{'brand': 'Jawa', 'price': 'ask the price', 'mileage': 25000, 'category': 'Bike - Naked', 'registration_year': 1992,
'color': 'Silver', 'description': 'Jawa 350 σε αρίστη κατάσταση με υπευθηνη δηλωση - Less -', 'extras': ''}
{'brand': 'Piaggio', 'price': 2300, 'mileage': 59000, 'category': 'Bike - Roller/Scooter', 'registration_year': 2011,
'color': 'Gold', 'description': 'σε αψογη κατασταση με βιβλιο service. - με πληρωμενο το ΚΤΕΟ και τα τελη και εχει γί
νει και το μεγάλο service αλλαγή ιμάντα - Less -', 'extras': 'Automatic, Catalyst, Centerstand, Electric starter, Led
lights, Lights Xenon, Service Book, Sidestand, Spoke wheels'}
{'brand': 'Kymco', 'price': None, 'mileage': None, 'category': 'Bike - Roller/Scooter', 'registration_year': 2019, 'c
olor': 'Black', 'description': 'ΝΕΟ ΜΟΝΤΕΛΟ ΜΕ ΔΙΠΛΑ ΔΙΣΚΟΦΡΕΝΑ ΚΑΙ ABS - INJECTION - ΜΟΛΙΣ ΠΑΡΕΛΗΦΘΗ! EURO 4! - ΔΙΑΚ
ΑΝΟΝΙΣΜΟΣ ΔΕΚΤΟΣ, ΑΤΟΚΕΣ ΔΟΣΕΙΣ - ΔΙΑΘΕΤΟΥΜΕ ΕΞΟΥΣΙΟΔΟΤΗΜΕΝΟ ΣΥΝΕΡΓΕΙΟ ΓΙΑ ΟΛΑ ΣΑΣ ΤΑ SERVICE ΚΑΙ ΓΙΑ ΚΑΛΥΨΗ ΕΓΓΥΗΣΕΩ
Ν - ΠΛΗΡΗ ΓΚΑΜΑ ΕΤΟΙΜΟΠΑΡΑΔΟΤΩΝ ΑΝΤΑΛΛΑΚΤΙΚΩΝ ΓΙΑ ΤΟ ΟΧΗΜΑ ΣΑΣ - ΔΙΑΤΙΘΕΤΑΙ ΚΑΙ ΣΕ 125 i CBS - Less -', 'extras': ''}
{'brand': 'Honda', 'price': 3000, 'mileage': 7000, 'category': 'Bike - Underbone', 'registration_year': 1985, 'colo
r': 'Red', 'description': 'ΣΕ ΑΡΙΣΤΗ ΚΑΤΑΣΤΑΣΗ ΓΝΗΣΙΟ C-50 ΑΝΤΙΠΡΟΣΩΠΕΙΑΣ ΑΠΟ ΠΛΗΡΗ ΑΝΑΚΑΤΑΣΚΕΥΗ:ΚΙΝΗΤΗΡΑΣ ΜΕ 4ΑΡΙ ΣΑ
ΖΜΑΝ-ΨΑΛΙΔΙ GLX 50,ΣΤΡΟΦΑΛΟΣ-ΜΠΙΕΛΑ-ΕΞΑΤΜΙΣΗ-ΚΑΜΠΑΝΑ ΔΙΣΚΩΝ ΑΜΠΡΑΓΙΑΖ-ΦΙΛΤΡΟΚΟΥΤΙ GLX 90,ΚΥΛΙΝΔΡΟΣ-ΠΙΣΤΟΝΙ-ΚΟΜΠΛΕ ΚΕΦ
ΑΛΗ-ΚΑΡΜΠΥΡΑΤΕΡ ASTREA 100,ΟΛΑ ΓΝΗΣΙΑ!!ΚΑΙΝΟΥΡΙΑ ΕΠΙΣΗΣ ΖΑΝΤΕΣ-ΛΑΣΤΙΧΑ-ΦΡΕΝΑ-ΓΡΑΝΑΖΙΑ-ΑΛΥΣΙΔΑ-ΜΠΑΤΑΡΙΑ-ΦΛΑΣ ΕΜΠΡΟΣ Κ
ΠΙΣΩ-ΠΟΔΙΑ-ΠΑΤΑΚΙΑ ΣΥΝΟΔΗΓΟΥ-ΠΡΟΣΤΑΤΕΥΤΙΚΑ ΑΛΥΣΙΔΟΣ!! - Less -', 'extras': ''}
{'brand': 'KTM', 'price': 6500, 'mileage': 30000, 'category': 'Bike - Naked', 'registration_year': 2005, 'color': 'Bl
ack', 'description': '-ΚΑΙΝΟΥΡΓΙΑ ΛΑΣΤΙΧΑ DUNLOP SPORT SMART 2. -ΚΑΙΝΟΥΡΓΙΑ ΓΡΑΝΑΖΙΑ ΕΜΠΡΟΣ - ΠΙΣΩ. -ΑΛΥΣΙΔΑ DID ZVMX
```

```
# Σύνδεση με την MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["assignment"]  # Όνομα βάσης δεδομένων
collection = db["assignment_collection"]  # Όνομα συλλογής

# Εισαγωγή των καθαρισμένων δεδομένων στην MongoDB
try:
    collection.insert_many(cleaned_data)
    print("Τα δεδομένα εισήχθησαν επιτυχώς.")
except Exception as e:
    print(f"Σφάλμα κατά την εισαγωγή δεδομένων: {e}")
```

```
Τα δεδομένα εισήχθησαν επιτυχώς.
```



In studio 3T we run the queries.



```
1  db.getCollection("assignment_collection").find({})
2
3
```

Raw shell output | Find Query (line 1)

50 | Documents 1 to 50 | Show delete dialog (F8)

assignment_collection > brand

_id	brand	price	mileage	category	registration_year	color	description	extras
65f00fb5640397...	Jawa	ask the price	25000	Bike - Naked	1992	Silver	Jawa 350 σε αρ...	
65f00fb5640397...	Piaggio	2300	59000	Bike - Roller/Sc...	2011	Gold	σε αψογη κατα...	Automatic, Cat...
65f00fb5640397...	Kymco	null	null	Bike - Roller/Sc...	2019	Black	ΝΕΟ ΜΟΝΤΕΛ...	
65f00fb5640397...	Honda	3000	7000	Bike - Underbo...	1985	Red	ΣΕ ΑΡΙΣΤΗ ΚΑΤ...	
65f00fb5640397...	KTM	6500	30000	Bike - Naked	2005	Black	-ΚΑΙΝΟΥΡΓΙΑ ...	
65f00fb5640397...	Bmw	5490	109000	Bike - On/Off	1990	Black	Πολύ καλή κατ...	Exchange with ...
65f00fb5640397...	KTM	3350	100	Bike - Enduro	2008	Orange	Σε άριστη κατά...	
65f00fb5640397...	Kymco	1350	7039	Bike - Roller/Sc...	2004	Silver (Metallic)	Η μοτοσυκλέτ...	Automatic, Ca...
65f00fb5640397...	Honda	6000	30	Bike - Cafe Racer	1979	Other	Honda CB400...	Car Exchange
65f00fb5640397...	Yamaha	1200	13000	Bike - Super M...	2000	White (Metallic)	Χωρίς κανένα ...	Exchange with ...
65f00fb5640397...	Yamaha	2200	28500	Bike - Roller/Sc...	2003	Black		

2.2) There are 29701 bikes for sale.

```
4  // 2.2
5  {db.assignment_collection.countDocuments()};
6
```

Raw shell output

Restart the MongoDB Shell | Use legacy shell | Clear raw she

```
1  29701
2
```

2.3)

The average price of a motorcycle is 3035.7 euros.

The number of listings that were used in order to calculate this average is different from task 2.2. and is equal to 29145.0. This $match stage filters out any documents where the price field does not exist or is null. This means that only documents with a valid, non-null price field are passed to the $group stage and included in the calculation of the average price.

The count of 29,701 documents you see in the first image is the total number of documents in the collection without any filters applied. So, it appears that 29,701 - 29,145 = 556 documents either do not have the price field or have it set to null, and therefore, were not included in the average price calculation.

```
// 2.3
db.assignment_collection.aggregate([
  {
    $match: {
      price: { $exists: true, $ne: null }
    }
  },
  {
    $group: {
      _id: null, // Group all documents together
      averagePrice: { $avg: "$price" }, // Calculate the average price
      count: { $sum: 1 } // Count the number of documents
    }
  }
]);
```

Raw shell output | Aggregate Query (line 11)

50 | Documents 1 to 1

assignment_collection > averagePrice

_id	averagePrice	count
null	3035.75748945...	29145.0

2.4) The maximum price is 89000 euro of a motorcycle and minimum price is 150 euro currently available in the market.

```
//  2.4
// Find the maximum price
db.assignment_collection.aggregate([
    {
        $match: {
            price: { $ne: "ask the price" } // Εξαιρούνται οι τιμές με την ένδειξη "ask the price"
        }
    },
    {
        $group: {
            _id: null,
            maxPrice: { $max: "$price" },
            minPrice: { $min: "$price" }
        }
    }
])
```

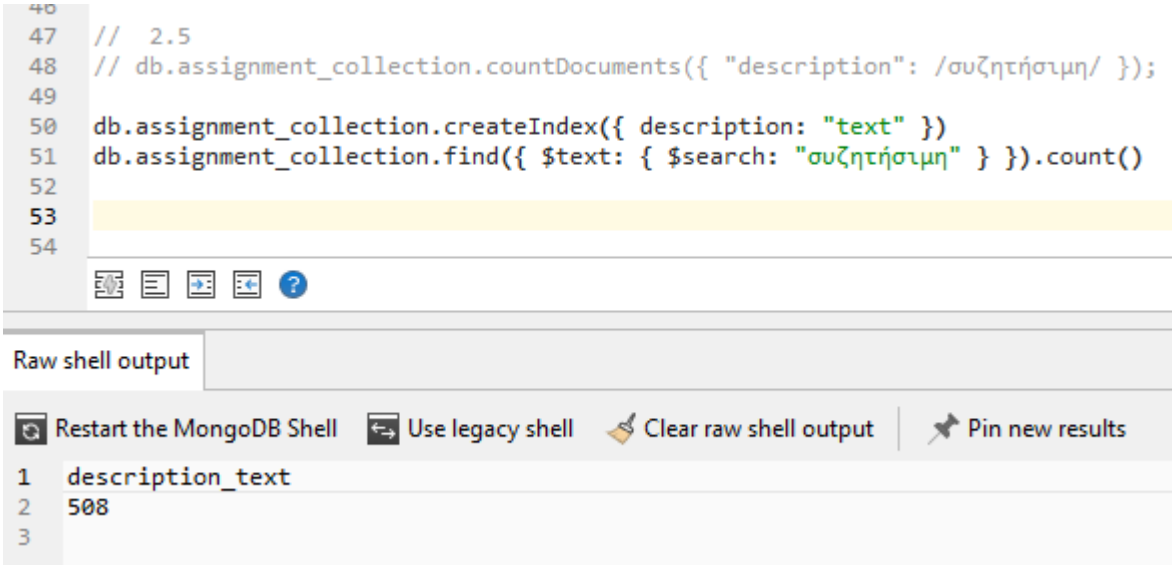
Raw shell output | Aggregate Query (line 40)

50 | Documents 1 to 1

assignment_collection > maxPrice

_id	maxPrice	minPrice
null	89000	150

2.5) The number of listings that have a price that is identified as negotiable is 508.



```
47  //  2.5
48  // db.assignment_collection.countDocuments({ "description": /συζητήσιμη/ });
49
50  db.assignment_collection.createIndex({ description: "text" })
51  db.assignment_collection.find({ $text: { $search: "συζητήσιμη" } }).count()
52
53
54
```

Raw shell output

Restart the MongoDB Shell | Use legacy shell | Clear raw shell output | Pin new results

```
1  description_text
2  508
3
```

2.6) For each Brand, the percentage of its listings is listed as negotiable is:
Dayang: 3.33%
Derbi: 2.32%
Harley Davidson: 0.32%
Kawasaki: 0.256%
Daytona: 0.254%
Bmw: 0.14%
Suzuki: 0.12%
Honda: 0.09%

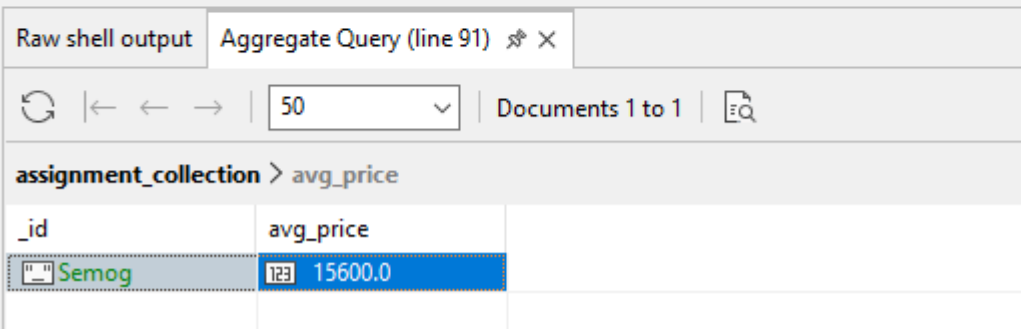
```
//  2.6
db.assignment_collection.aggregate([
  {
    "$group": {
      "_id": "$brand",
      "totalCount": { "$sum": 1 },
      "matchedCount": {
        "$sum": {
          "$cond": [
            { "$ne": [ { "$indexOfCP": [ "$description", "Συζητήσιμη" ] }, -1 ] }, 1,  0]
        }
      }
    }
  },
  {
    "$addFields": {
      "percentage": {
        "$cond": {
          "if": { "$ne": [ "$matchedCount", 0 ] },
          "then": {
            "$multiply": [
              { "$divide": [ "$matchedCount", "$totalCount" ] },
              100
            ]
          },
          "else": 0
        }
      }
    }
  },
  { "$sort": { "percentage": -1 } }
]);
```

assignment_collection > totalCount

_id	totalCount	matchedCount	percentage
Dayang	30.0	1.0	3.33333333333...
Derbi	86.0	2.0	2.32558139534...
Harley Davidson	309.0	1.0	0.32362459546...
Kawasaki	1953.0	5.0	0.25601638504...
Daytona	393.0	1.0	0.25445292620...
Bmw	1394.0	2.0	0.14347202295...
Suzuki	2365.0	3.0	0.12684989429...
Honda	6190.0	6.0	0.09693053311...
JetMoto	4.0	0.0	0.0
Kxd	7.0	0.0	0.0
Baotian	14.0	0.0	0.0
Goes	1.0	0.0	0.0
Haojin	10.0	0.0	0.0
Garelli	24.0	0.0	0.0
Aprilia	892.0	0.0	0.0
Enfield	4.0	0.0	0.0

2.7) The motorcycle brand with the highest average price is Semong, with average price 15600.0

```
90  //  2.7
91  db.assignment_collection.aggregate([
92    { $group: { _id: "$brand", "avg_price": { $avg: "$price" } } },
93    { $sort: { "avg_price": -1 } },
94    { $limit: 1 }
95  ]);
96
97
```



Raw shell output | Aggregate Query (line 91)

50 | Documents 1 to 1

assignment_collection > avg_price

_id	avg_price
Semog	15600.0

2.8)

```
99  //  2.8
100 db.assignment_collection.aggregate([
101   {
102     $group: {
103       _id: "$model",  // Group by model
104       AverageAge: {
105         $avg: {
106           $subtract: [new Date().getFullYear(), "$registration_year"]
107         }
108       }
109     }
110   },
111   {
112     $project: {
113       Model: "$_id",
114       AverageAge: { $round: ["$AverageAge", 1] }  // Round to one decimal place
115     }
116   },
117   { $sort: { AverageAge: -1 } },  // Sort by average age in descending order
118   { $limit: 10 }  // Limit to top 10
119 ]);
120
121
```

Raw shell output | Aggregate Query (line 100)

50 | Documents 1 to 1

assignment_collection > Model

_id	Model	AverageAge
null	null	18.8

2.9) 1018 bikes have “ABS” as an extra.

```
122 //  2.9
123 db.assignment_collection.createIndex({ extra: "text" })
124 db.assignment_collection.find({ $text: { $search: "ABS" } }).count()
125
126
```

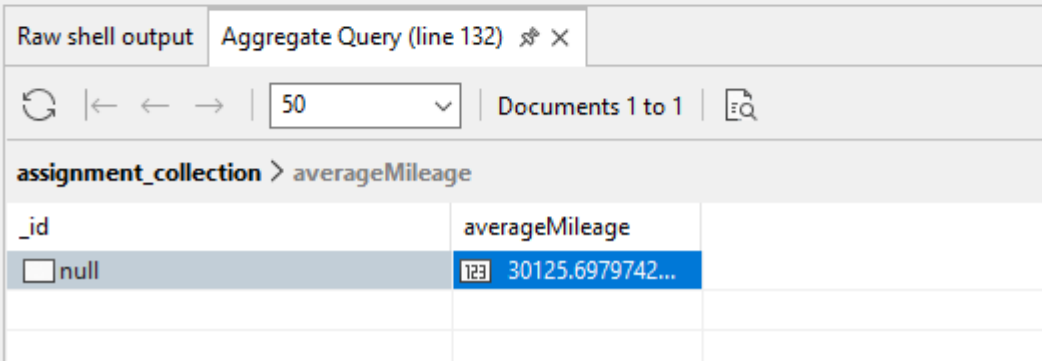
Raw shell output

Restart the MongoDB Shell | Use legacy shell | Clear raw shell output | Pin new results

```
1  1018
2
```

2.10) The average Mileage of bikes that have “ABS” AND “Led lights” as an extra is 30125 Km

```
// 2.10
db.assignment_collection.aggregate([
    {
        $match: {
            $and: [
                { extras: { $regex: "ABS", $options: "i" } },
                { extras: { $regex: "Led lights", $options: "i" } }
            ]
        }
    },
    {
        $group: {
            _id: null,
            averageMileage: { $avg: "$mileage" }
        }
    }
])
```



Raw shell output | Aggregate Query (line 132)

50 | Documents 1 to 1

assignment_collection > averageMileage

_id	averageMileage
null	30125.6979742...

2.11) For example the TOP 3 colors for the category street bike is black , black (metallic) , red .

```
// 2.11
db.assignment_collection.aggregate([
  {
    $group: {
      _id: { category: "$category", color: "$color" },
      count: { $sum: 1 }
    }
  },
  {
    $sort: { "_id.category": 1, "count": -1 }
  },
  {
    $group: {
      _id: "$_id.category",
      colors: { $push: { color: "$_id.color", count: "$count" } }
    }
  },
  {
    $project: {
      topColors: { $slice: ["$colors", 3] }
    }
  }
]);
```

Raw shell output | Aggregate Query (line 144)

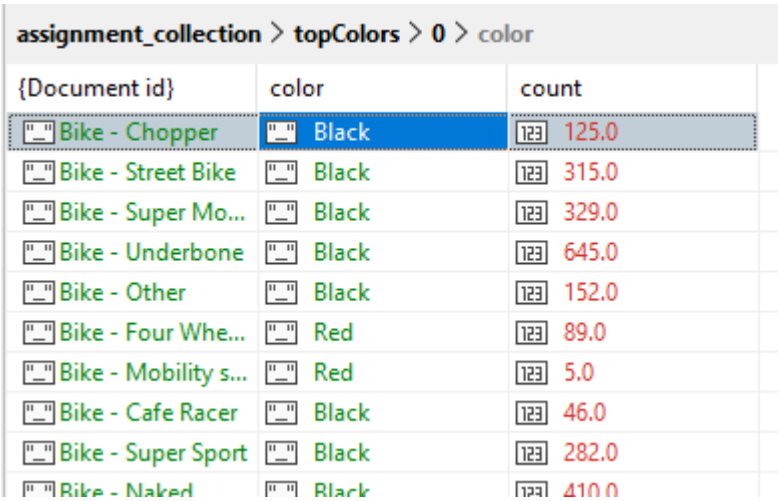
50 | Documents 1 to 22

assignment_collection > topColors

_id	topColors
Bike - Custom	[3 elements]
Bike - Cafe Racer	[3 elements]
Bike - Four Whe...	[3 elements]
Bike - Trial	[3 elements]
Bike - UTV Side ...	[3 elements]
Bike - Buggy	[3 elements]
Bike - Enduro	[3 elements]
Bike - Super Sport	[3 elements]

assignment_collection > **topColors** > 0

{Document id}	0	1	2
Bike - Chopper	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Street Bike	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Super Mo...	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Underbone	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Other	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Four Whe...	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Mobility s...	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Cafe Racer	{ 2 fields }	{ 2 fields }	{ 2 fields }
Bike - Super Sport	{ 2 fields }	{ 2 fields }	{ 2 fields }



assignment_collection > topColors > 0 > color

{Document id}	color	count
Bike - Chopper	Black	125.0
Bike - Street Bike	Black	315.0
Bike - Super Mo...	Black	329.0
Bike - Underbone	Black	645.0
Bike - Other	Black	152.0
Bike - Four Whe...	Red	89.0
Bike - Mobility s...	Red	5.0
Bike - Cafe Racer	Black	46.0
Bike - Super Sport	Black	282.0
Bike - Naked	Black	410.0

2.12)

```
//2.12
db.assignment_collection.aggregate([
  {
    $match: {
      condition: "good",
      mileage: { $lte: 10000 }
    }
  },
  {
    $group: {
      _id: "$category",
      avgPrice: { $avg: "$price" }
    }
  },
  {
    $lookup: {
      from: "assignment_collection",
      let: { category: "$_id", avgPrice: "$avgPrice" },
      pipeline: [
        { $match:
          { $expr:
            { $and:
              [
                { $eq: [ "$category",  "$$category" ] },
                { $lt: [ "$price", "$$avgPrice" ] }
              ]
            }
          }
        },
        { $project: { model: 1, price: 1, mileage: 1 } } // Modify to include
      ],
      as: "deals"
    }
  },
  { $unwind: "$deals" },
  { $limit: 10 } // Adjust based on how many deals you want to fetch
]);
```