**Assignment 01**

The chosen dataset isthe *Youtube Videos Dataset (~3400 videos)*. This dataset contains around 3400 videos divided into four categories: Travel vlogs, Food, Art and Music, and History. The things that were scraped for this video dataset are their Video ID links, title, description, and category. I decided to use this dataset for the assignment because it is one of the datasets I want to use and explore in my final project. I want to test the water with this dataset to see the potential of my project and if I’ll need to adjust and make some changes accordingly in this artistic process.

I faced several issues during the package installation and data importation process. I decided to install the *mongoshell/atlas* commandline and the *mongoimport* tool. It was a bit difficult to follow through all the installation steps. There were a lot of precise steps to follow, and I was a bit nervous while doing it for fear of messing things up on my computer. I had to re-install *Chocolatey* because I forgot a step, but in the end, I was able to install everything correctly. The second challenge was importing the data; it took me a while to understand the *Mongoimport* function; the handshake was not happening. Ultimately, I understood how to write the path for importing my CSV file. It required my connection string at the beginning of the code line to make the handshake happen. The third struggle was to connect my *MongoDB* profile with my code. It took me a while to realize I forgot to add my URI to the env file.

I made eight functioning queries with this dataset. The first one was to count the number of documents in the collection. I used the .*estimatedDocumentCount()* function for this query. I discovered there were 3599 documents in this collection. The creator of this dataset did not lie about the number mentioned in the description and title of this dataset, which was around 3400 videos collected.

In the second query, I asked to display all the collected data in the terminal. I used the *.find({})* and *.toArray()* functions for this query. Only the first 100 documents are displayed in the terminal. The other 3499 documents were just mentioned at the end of the result. It has shown how heavy this dataset is.

The third query served to retrieve the data from the history category and only displayed their videos’ titles. I used the .*distinct()* function for this query. I discovered 587 historical videos in this dataset, many of which were not in English. I have noticed some of them had their titles in Hindi or Mandarin. It shows the variety of this dataset.

The fourth query was to find documents of videos with over 100 subscribers on their channel. I used the *.find()* and *.countDocuments()* functions for this query. I found some issues with the dataset through this query. Because the number of subscribers is written in the description (i.e., everything is written as a string in this input) and does not have a subcollection, I needed to pull the numbers as a string from the description. Because of that, the results of the query were inaccurate. I did not receive an accurate number range when using comparison operators. Sometimes, the numbers do not match my request. For example, I found a lot of decimal numbers between 1 and 100, but those that were not in decimals were over 100.

I asked for a video with a specific ID for the fifth query. I used the *.findOne()* function for this query I looked through the data collection in the *MongoDB Atlas*, took an ID randomly, and asked for it in the code. It worked because I verified that the pulled data matched the document in my *MongoDB Atlas*.

For the sixth query, I filtered the art and music category by displaying only the videos with less than 400 subscribers on the channel, then giving only the title and its description in the results. I used the *.aggregate()* function for this query. As mentioned in the fourth query, the results were again inaccurate because the subscribers’ numbers are in a string and do not have their subcollection. Plus, the numbers are written with letters too, such as 90K or 1.1M, so it messes up the accuracy of the results. However, I discovered popular videos in this collection by seeing *NSYNC*’s *Pop* Music video as part of the results, which creates more variety in this dataset.

The seventh query was finding the food category videos, sorting them alphabetically, and limiting the results to five videos. I9 used the *.find()*, *.sort()*, and *.limit()* functions for this query. It worked, and again, Japanese writings and the *BuzzFeed Video* channel appeared in the results.

The last query was finding one video according to the input categories. I used the *.findOne()* function for this query. I mixed categories that were not part of this dataset and some existing categories, and it could still pull one document that matched one of the existing categories.

The thumbnail images of each video visualized the data the query pulled, accompanied by their video links. I used the aggregation query where only videos coming from the arts and music category with a number of subscribers below 400 are pulled. Because of the format of this dataset, my options for queries and search criteria were minimal. So, I decided to create a search engine where the user can discover YouTube videos by searching through these four categories: food, travel, history and art. It took me a while to figure out how to add multiple categories in this search engine because *res.send()* could not handle more than one variable. With the help of Sabine, I was able to create four different queries for each category. I then assigned each thumbnail to their associated video description.

1. The user must answer the question to access the data.

A red rectangular object with white text

Description automatically generated

If they provide the wrong answer, they receive “Wrong answer, try again.”

A red rectangular object with white text

Description automatically generatedA red rectangular object with white text

Description automatically generated

1. If they provide the right answer but with the wrong spelling they receive “You’re close!!”. A red rectangular object with white lines

   Description automatically generatedA red rectangular object with white text

   Description automatically generated
2. And if they provide the right answer they receive the data.

A red rectangular object with white lines

Description automatically generated

As seen here, the dataset format was badly written because some videos have over 400 subscribers (eg., *NSYNC YouTube* channel). I also used different thumbnails for the inaccessible videos (i.e., which is something that I’ve discovered while inserting the video links), one for those that are now private and another one for those whose channels or videos are now deleted. I found it very interesting that the dataset still has videos unavailable to the public; it shows that the date was not cleaned recently.

Overall, I discovered that the dataset used for this assignment might not be a good fit for my final project or is at least very limited in its options. I tried various MongoDB functions, and only a few could pull data from this dataset. Plus, some document fields were too descriptive, so it was hard to pull data from them. I always received empty arrays or null if I couldn’t find a document by searching for a specific word within a string in the title or description fields. For example, if I want to search all the documents containing the word “India” in their titles, it is impossible because the query is not precise enough. I would need to give the full title or description to receive something in the terminal. I think it is in how things were categorized; there are only a few categories and no subcollections in this dataset, limiting the possibilities of pulling out data. I also looked at the CSV file and noticed that some elements in the categories could have been categorized. For example, in the description category, many elements written in it could have been divided, such as the content creators’ names or the total of subscribers. The only category working well in this dataset is “category” because it is the only category that requires few keyword inputs for the MongoDB functions. All the other categories are too descriptive. For obscure reasons, I cannot insert the request alone in the *.find()* function; I need to pass them into a variable and use that variable to make it work. If I want to use this dataset, I must edit it by creating subcollections in its categories to pull data effectively. However, I see that this dataset has a variety of collected videos, from the language to the popularity of the channels they’re coming from. It is very disappointing that the dataset does not have many divisions between its elements because the diversity in its collections had a lot of potential for my final project.