Homework Assignment #3

(MongoDB)

After completing the previous assignment successfully, your reputation as a ShopALot.com data engineer has been skyrocketing! Based on your fast learning skills you have been approached for a part time job at a startup that wants to perform some queries on top of a MongoDB cluster. You were asked to set up a cluster for them in the cloud and get it running and ready to go, load the pre-acquired data, and run some initial analytical queries to give them some insights into the data that they have.

Get Ready...

You should start by reading through the setup instructions **carefully** in order to create an Atlas account with MongoDB and set up a cluster using the provided steps. Afterwards, you will be asked to install the MongoDB shell + MongoDB Compass in order to complete this assignment successfully. You will also be using a Jupyter notebook for this assignment, which comes with Anaconda.

Get (Data) Set...

Your next step is to grab yourself a copy of the relevant ShopALot data. The data has been provided along with a script to load the data so you can be more focused on the analytics rather than on loading the data into the system. Since MongoDB is a document-based database, the schema is self -describing within the data itself; you will be unveiling some information about the schema as the assignment goes along. Some sample data is also provided, along with some example queries, to help to start your interaction with the MongoDB query language.

Go...!

It's time to get to work. You will be using Compass to answer the schema analysis questions and the rest of the queries will be done using a Jupyter notebook where you will write the queries using python (and pymongo). You may also want to use Compass to "unit test" your filters.

1. You will start by exploring the schema for several of the ShopALot collections here. In order to do that you will need to connect to your cluster using Compass and click on the ShopALot database on the left panel. After that, choose the desired collection. Once a collection is chosen, you can then navigate to the schema tab to analyze a sample of the stored data. This will help you answer the following questions:
    A. In the **users** collection, what is the data type for the field *name*?
    B. From that same collection, what is the type for the field named *phones*?
    C. In the **orders** collection, what is the type for the field *time_placed*?
    D. From the same collection, what is the type for the field name *selling_price*?

E. In the **stores** collection, what are the min, average and max array length for the *categories* field?

2. Now we will begin exploring the ShopALot data with the help of the **find** API of MQL. To do so, you will answer the following questions using find.
   Note: You can export any queries that you choose to create and run on Compass first as language-specific (Python) code that can be run on Jupyter if you want. (Not required.)
   A. Your boss wants you to find the total price of the order that was placed at "2020-05-26T00:30:28.000Z".
   B. We would now like to get a better understanding of the products collection. You are asked to find products that are in the “Bread & Bakery” category and have a list price greater than 3. Sort your answer in descending order of their list price and print the first 10 documents.
   C. You have noticed that there are some missing descriptions in the products collection. We would like to update the missing descriptions (i.e., those documents without description fields). Add a description field for each of these documents and set the values to be “Amazing product”.
   D. A store employee found a wallet in the parking lot. Unfortunately, there is no ID in the wallet to help find the owner. The employee claims that a gray Ford car drove away from where he found the wallet. The car was made in, or perhaps before, 2008. It is likely that the driver ordered from ShopALot.com, you have been asked to find all the license plates of vehicles that meet the requirements above. (Make sure you only show their license plates and no other fields.)
   E. We would like to **count** the number of baby care products in the medium price range and pet care products in the high price range that could be targeted for business promotion. Your task is to count the total number of products that are either “Baby Care” products with list prices greater than 3 and less than 10 or “Pet Care” products with list prices greater than 10.
   F. Your boss wants to have a list of shoppers who can be contacted when a large number of orders comes in. You are asked to search for shoppers that have a home phone on file and also have a ‘capacity’ field. Return just their user id and name fields. Sort your results in ascending order of last name and print only the first 5 documents.
   G. We will now be shifting gears and instead using the **pipeline** feature that MongoDB provides for users. Using the MongoDB pipeline API we want to sum up the total prices for all orders placed by the customer with id “GS65P”.
   H. Aggregated reports are easier to read and present compared to raw collected data. Your boss wants to see -- for every store -- its total number of orders, its average order price, and its maximum order price. Print just the store_id, count of orders, average order price, and max order price. Sort your results in descending order of order counts and print only the first 10 documents.
   I. To understand how joins work in MongoDB, your boss wants you to join the orders that have total prices greater than 100 with the customers that they are linked to. Return just the order id, total price, and customer emails. Print out a **sample** of 5 documents.

J. **[Extra credit]** In order to demonstrate your impressive mastery of pipeline API stages, your boss would like for you to convert the following SQL statement to an equivalent MQL pipeline and execute it.

```
SELECT u.user_id, u.name, u.email, COUNT(o.order_id) AS num_orders,
MIN(o.total_price) AS min_price
FROM Users u, Orders o
WHERE u.user_id = o.customer_id
AND o.time_placed BETWEEN '2018-02-16T02:56:00.000Z'
                      AND '2019-01-15T22:06:43.000Z'
GROUP BY u.user_id
HAVING MIN(o.total_price) >= 20
ORDER BY COUNT(o.order_id) DESC
LIMIT 5;
```

What To Turn In

When you have finished your assignment you should use Gradescope to turn in a **PDF** file that lists all of the answers to the questions together with the **results** of running each cell ***without*** including the code to load the data into Atlas. (You can comment that out if you prefer to keep it in your notebook for reference.) Please follow the following steps in order to generate the file for submission.

1. Develop and keep your answers in a .ipynb file using Jupyter.
2. For the first question, put your answers in one or more cells as comments. (Recall that comments in Python begin with a hashtag for each line.)
3. Put each of your queries for the second question in a separate appropriately commented cell and make sure that they actually run and print out their results by inserting list(result) or print(result) after each query (depending on the query). You should restart the kernel and run all the queries once more in your notebook, to run them together and in sequence from top-to-bottom, before turning it in!
4. Remove (or just obfuscate) your Atlas login information from the notebook.
5. Make sure all your code can be seen without scrolling to the right. Break your code into multiple lines if necessary.
6. Save your .ipynb file, with all of the cells and their results, as a PDF. To do that nicely in Jupyter, if you have LaTex available, you should use File -> Download as -> PDF via LaTex (.pdf). If not, you can use File -> Print Preview and then PDF-print the result. Once done, double check that all the code is visible in your PDF file.
7. Turn in the PDF!