CUSTOMER BEHAVIOR ANALYSIS

A part of GDDA612\_Data Transformation and Management Assessment

Prepared by: Aju Peter

New Zealand School of Education College (NZSE)

**Content:**

**Task A – Data Preparation and Cleansing in Python**

1. **Data preparation in python**
2. **Data cleaning in python**
3. **Displaying cleaned data**
4. **Filtering data**
5. **Establishing connection with MongoDB**
6. **Import dataset into a MongoDB collection**
7. **Displaying collection contents**
8. **Sorting collection**
9. **Counting total number of documents in collection**
10. **Grouping operations**
11. **Updating collection**

**Task B – Data migration and Backup**

1. **Exporting data from MongoDB**
2. **Establishing connection with GCP**
3. **Automated backup with GCP**
4. **Error handling**

**Reference**

**Tutor approval**

**Github**

**Scenario:**

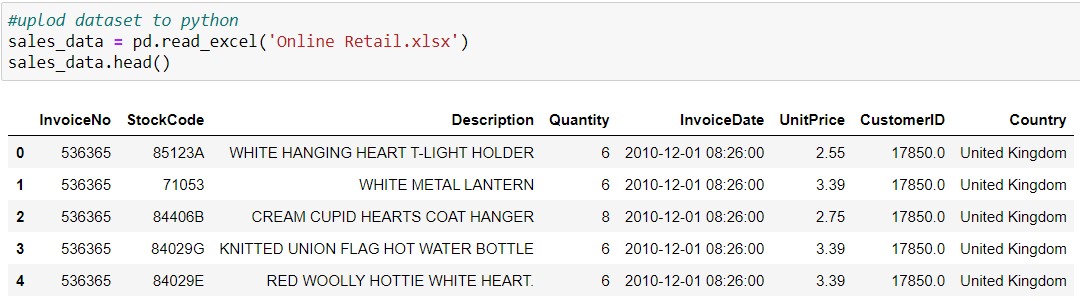
I am employed as a data analyst at a homeware e-commerce company, my primary responsibility is to analyse sales data to uncover insights into customer behaviour and refine marketing strategies accordingly. The dataset provided encompasses detailed information regarding customer transactions, including purchase specifics, product attributes, and customer demographics. The dataset, which I sourced from Kaggle dataset having 8 variables includes InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID , and Country.

# Task A – Data Preparation and Database Integration

**a) Used Python to load and analyse dataset.**

To perform the data cleaning process, I have selected python software based IDE “Jupiter Notebook”. I started with importing the required libraries and then uploaded the “Online Retail” dataset using “pd.read\_excel( )” command. (See figure 1)

* Pandas: a library for data manipulation and analysis.
* NumPy: a library for statistical analysis.



*Figure 1*

**b) Data cleaning**

To begin the data cleaning process, several pandas functions were utilized to understand and prepare the dataset. The steps followed include:

* I have used the “shape” function to find size of the dataset: number of rows (541909) and columns (8). (See figure 3)

A close up of a text

Description automatically generated

Figure 2

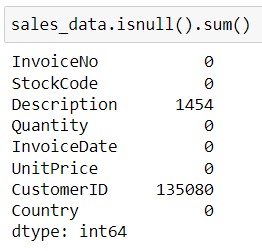
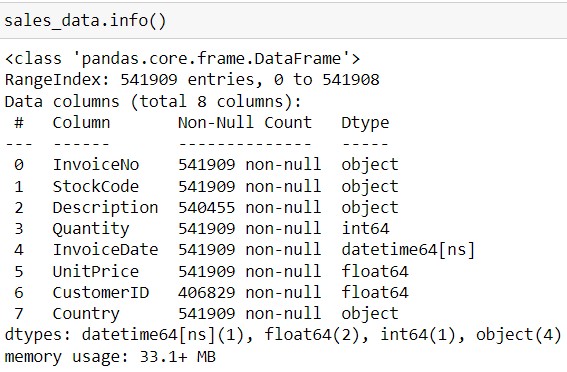
* Identifying and counting null values in the dataset by applying the isnull().sum() method. This step helps in understanding the missing data across different variables. (See figure 2)

Figure 3

* Examining the dataset schema using the info() method to reveal the data type of each variable and the number of non-null values recorded for those variables. (See figure 4)

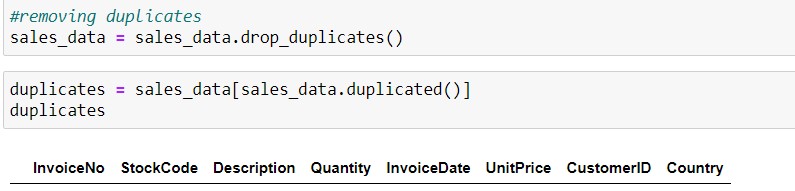
 *Figure 4*

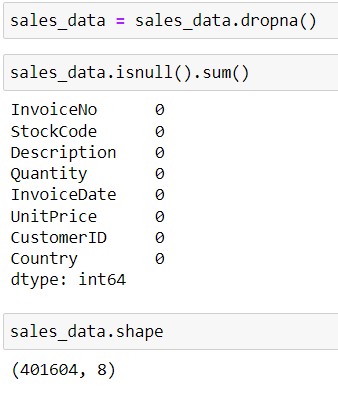
These initial steps are crucial for assessing the quality of the dataset and planning subsequent cleaning and preprocessing tasks.

To further clean the data, the “duplicated ()” function was employed to identify duplicate records within the dataset, revealing a total of 5268 duplicate rows. (See figure 5)

 *Figure 5*

To eliminate these redundant entries, the “drop\_duplicates()” method was utilized. Following this operation, it was verified that the dataset was free of duplicate records.

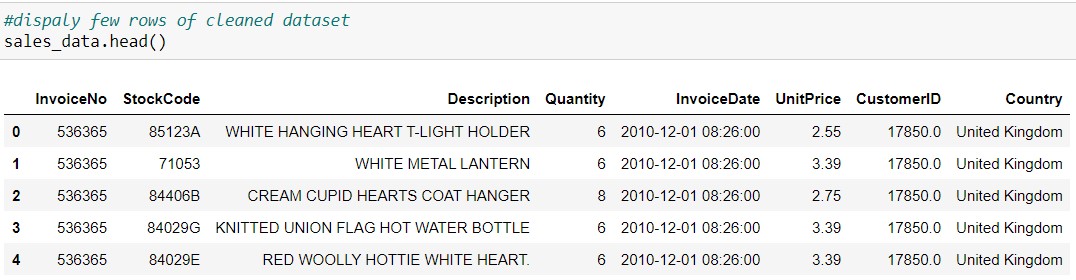


In the final step of the data cleaning process, all null values were eliminated using the dropna() method. This action resulted in a cleaned and processed dataset consisting of 401,604 rows and 8 columns, ready for further analysis. (See figure 6)

*Figure 6*

1. **Displaying the cleaned data**

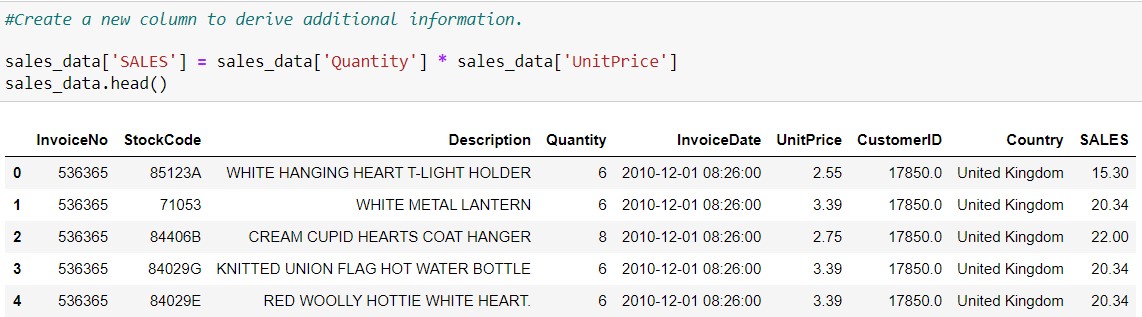
To display some rows of the cleaned dataset, the head() command was used, its default displays the first 5 rows of data. This provided a quick overview of the dataset's structure and content after the cleaning process. (See figure 7)

 *Figure 7*

1. **Filtering dataset.**

Upon reviewing the dataset, it was observed that a crucial "Total Sales" column was absent, which is essential for quickly deriving insights through sorting or querying. To address this, a

"SALES" column was added to the dataset. (See figure 8)



*Figure 8*

To integrate the dataset for more efficient analysis, the data was aggregated by "Country" to summarize key metrics such as Total Sales, Invoice Count, Total Quantity Ordered, and

Average Price Each. This was achieved using the aggregate function. (See figure 9)



*Figure 9*

I filtered the records with a sales value exceeding $20,000 were identified and organized in descending order to prioritize higher-value insights. (See figure 10)

A screenshot of a computer

Description automatically generated*Figure 10*

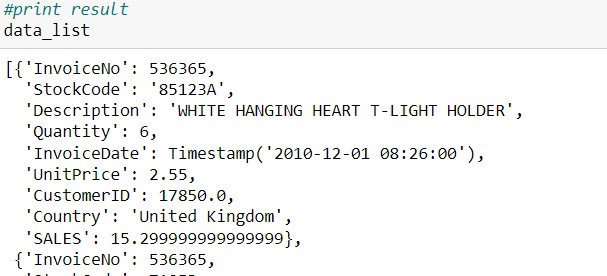
1. **Establishing a connection with MongoDB (NoSQL) database.**

For managing the online retail dataset, which is large and does not include relational tables, the decision was made to transform the data into a collection format to facilitate easier querying and manipulation. MongoDB was chosen as the database for this purpose. To establish a connection between Python and MongoDB, necessary libraries including pymongo, pandas, and json were imported. To initiate the connection within a Jupyter Notebook to MongoDB, a variable named client was defined by utilizing pymongo.MongoClient(). Subsequently, a collection named Assessment\_2 was created. Furthermore, a variable called data\_list was introduced to convert the dataframe into a dictionary format, achieved by applying the to\_dict() method, allowing for efficient data importation into MongoDB. (See figure 11)



*Figure 11*

To ensure the accuracy of the data transformation, the dictionary was displayed to cross-verify the contents of the data collection. (See figure 12)



*Figure 12*

A new database named GDDA612 was successfully created within MongoDB. The creation of the database was verified, confirming that the setup was completed without any issues. (See figure 13)



*Figure 13*

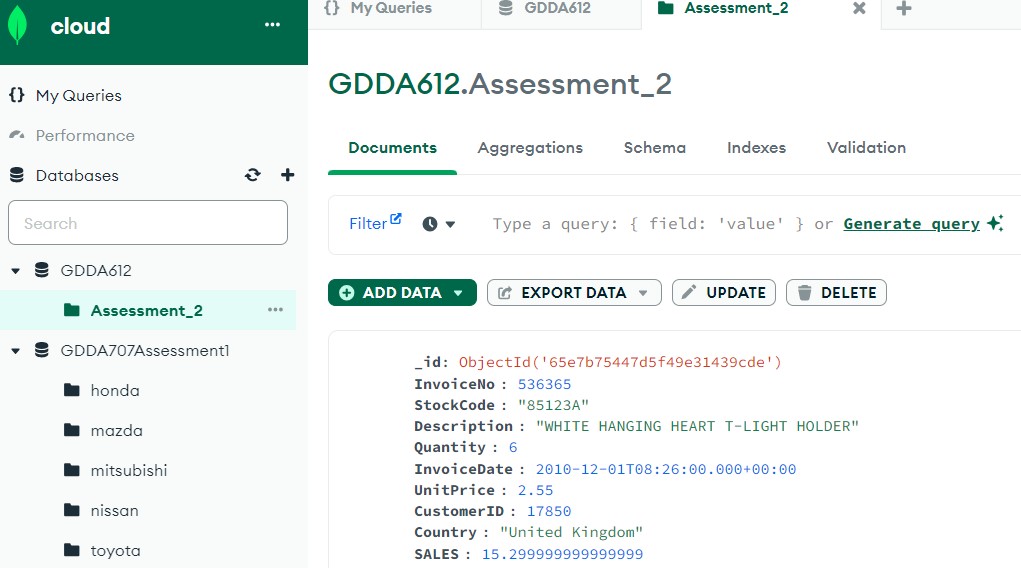
1. **Import the dataset into a collection within the database.**

After the successful creation of the database GDDA612 in MongoDB, the data collections were transferred into this newly established database utilizing the insert\_many() function. This operation facilitated the bulk insertion of documents into the specified collection. After the successful data transfer, the connection between the Jupyter Notebook and MongoDB was terminated using the close() function. (See figure 14)

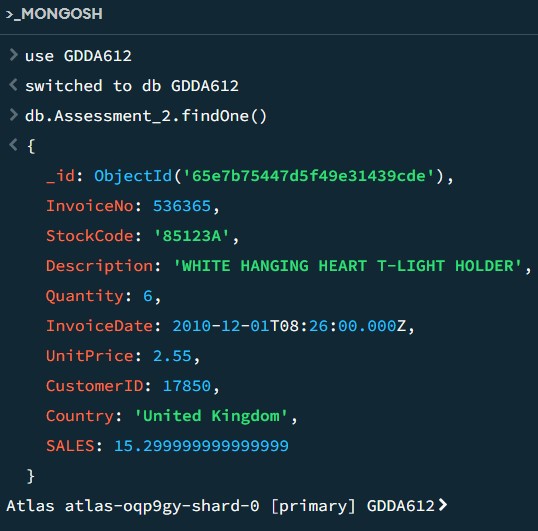


*Figure 14*

Later, MongoDB Compass was opened, and a connection was established with the MongoDB cloud database. This step was undertaken to verify that the dataset had been correctly inserted into the MongoDB cloud database. (See figure 15)



*Figure 15*

1. **Retrieve and display records from the table or collection.**

Utilizing the MongoDB Shell (mongosh) for queries and data manipulation, I initially connected mongosh to the MongoDB cloud database and accessed the collection. To preview a single record from

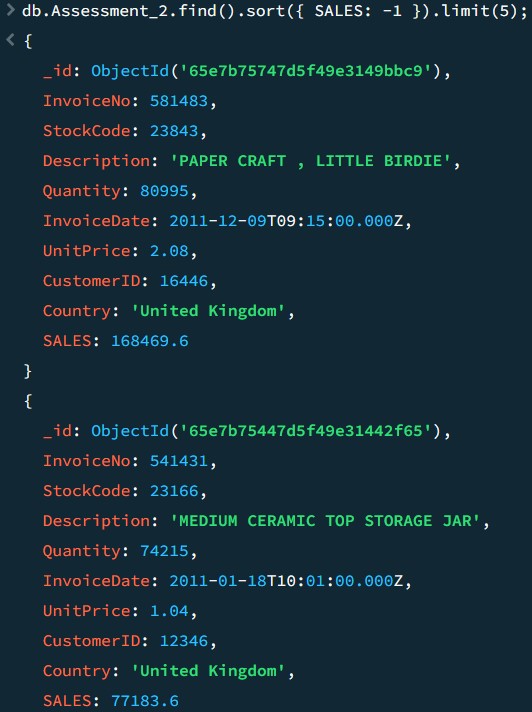
the collection, the db.Assessment\_.findOne() function was used to successfully displaying a document. (See figure 16)

*Figure 16*

**h) Sorting collection based on condition.**

I have sorted the collection by the SALES field in descending order, I used the following

MongoDB query: (See figure 17)



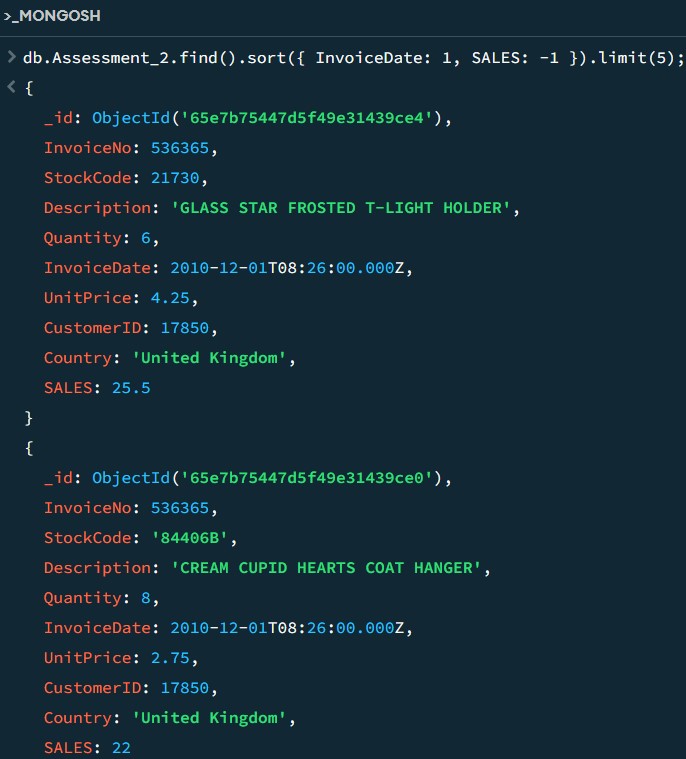
*Figure 17*

I have then sorted the collection by InvoiceDate in ascending order to see the earliest sales first and limit the results to the top 5 oldest records, , I used the following MongoDB query:

db

*Figure*

*18*

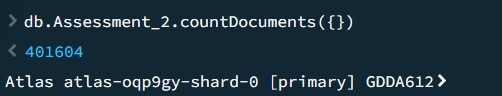


1. **Count the number of records present in the collection.**

I have counted the number of records in the collection by using the following mongodb query

which resulted the same result as having the jupyter notebook, which ensured the dataset transformation was successful. (See figure 19)

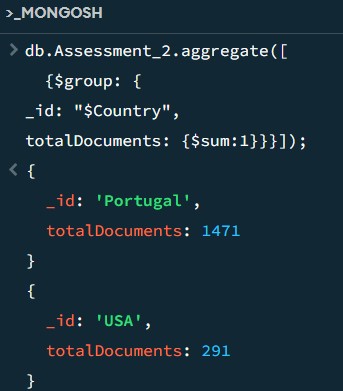
|  |  |  |  |
| --- | --- | --- | --- |
| db. | Assessment\_2. | countDocuments | ({}); , |



*Figure 19*

1. **Perform grouping operations on records within the collection.**

I have used the $group function to group documents by Country field and count the number of documents in each country. (See figure 20)



*Figure 20*

I have performed another grouping operation, such as calculating the total sales (SALES) per country. (See figure 21)



*Figure 21*

**k) Update collection.**

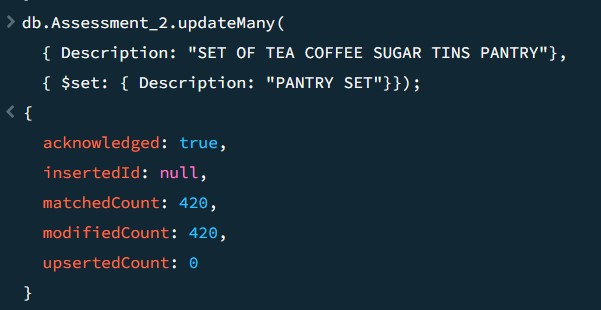
The first record's Country field was updated to "United Kingdom" using the updateOne() function.(See figure 22)



*Figure 22*

The description was changed to "PANTRY SET" from the longer "SET OF TEA COFFEE SUGAR TINS

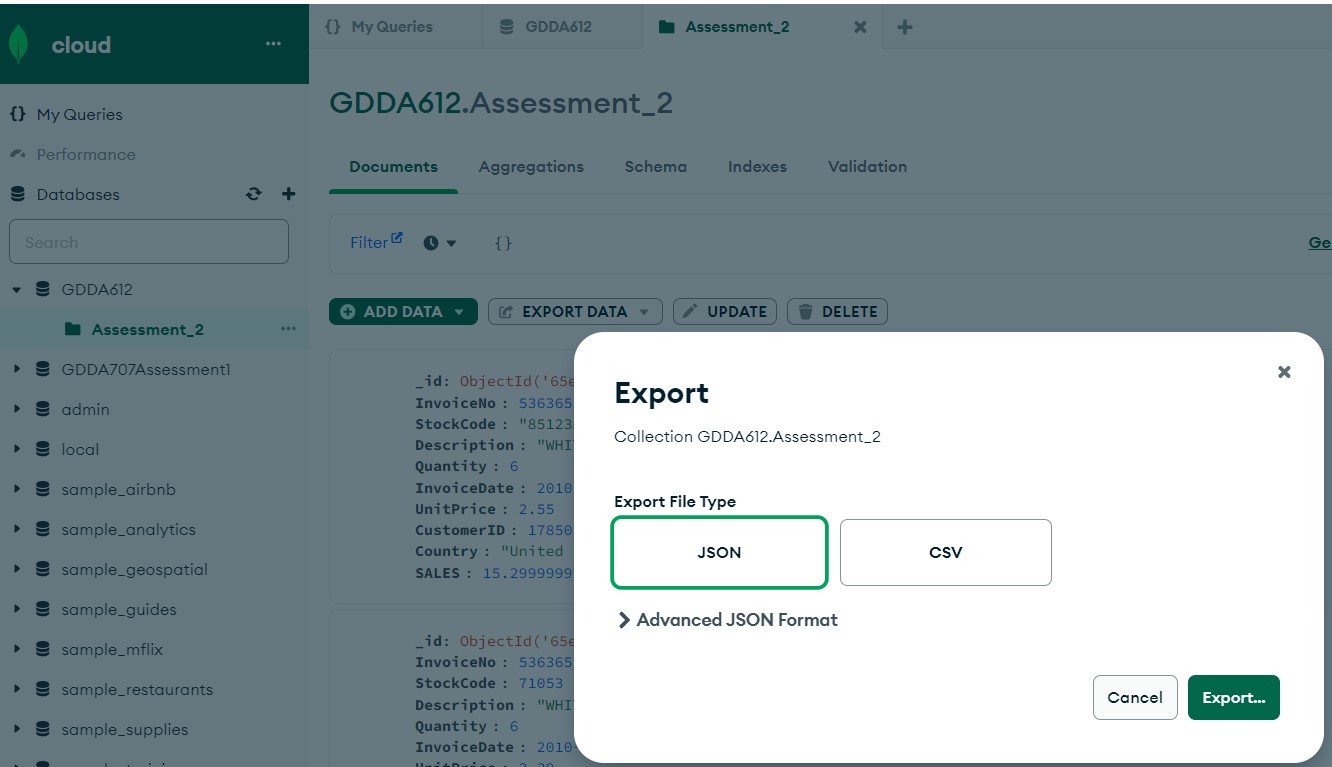
PANTRY" using updateMany function. (See figure 23)



*Figure 23*

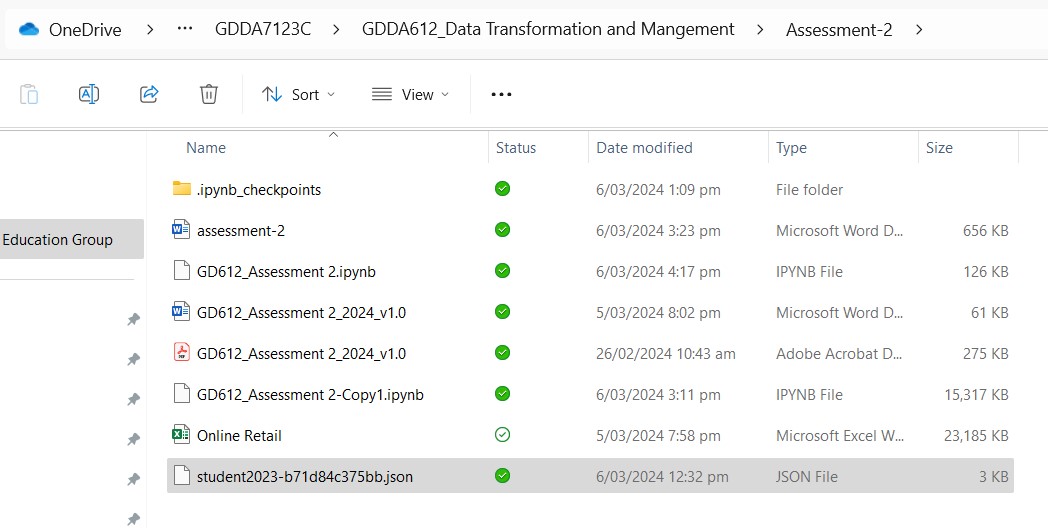
# Task B – Data Migration and Backup

1. The data collection was exported to my local drive as a JSON file, a format well-suited for saving collections from MongoDB cloud. (See figure 24)



*Figure 24*

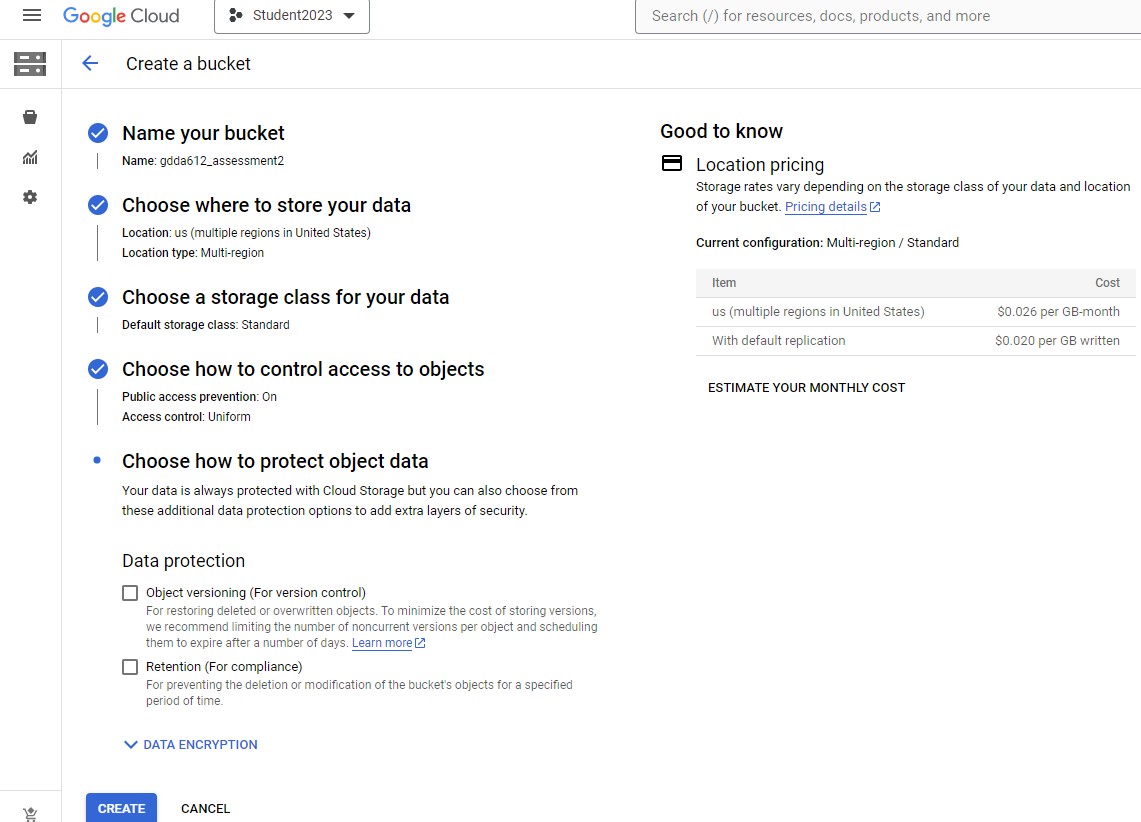
The data was stored in a OneDrive location to ensure the file's security. (See figure 25)



*Figure 25*

1. **Establishing connection with Google cloud storage (GCP).**

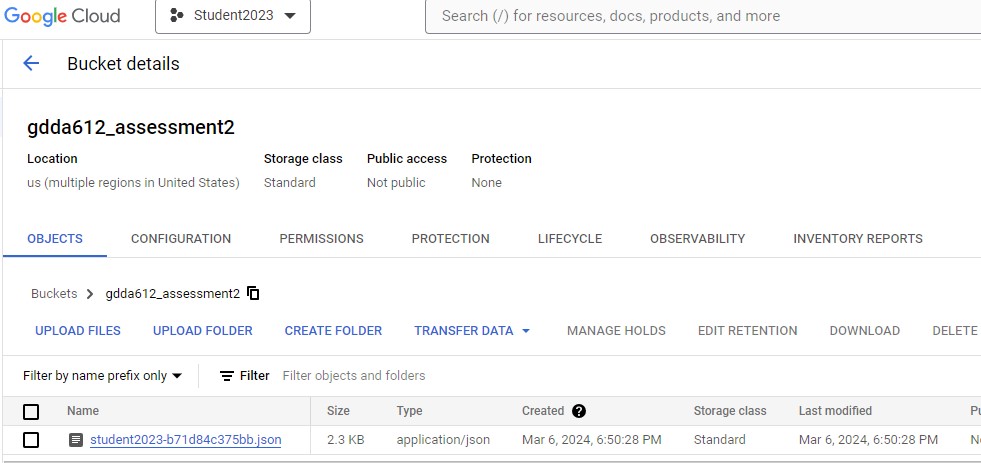
I am familiar with the Google Cloud Platform (GCP), so I chose it for storing the online sales data collection. Having an existing GCP account, I first logged in and selected Google Cloud Platform to create a storage bucket. I initially created a bucket and named it gdda612\_assessment2. (See figure 26)



*Figure 26*

Then, I imported the dataset from my local drive to the Google Cloud bucket using the "Upload

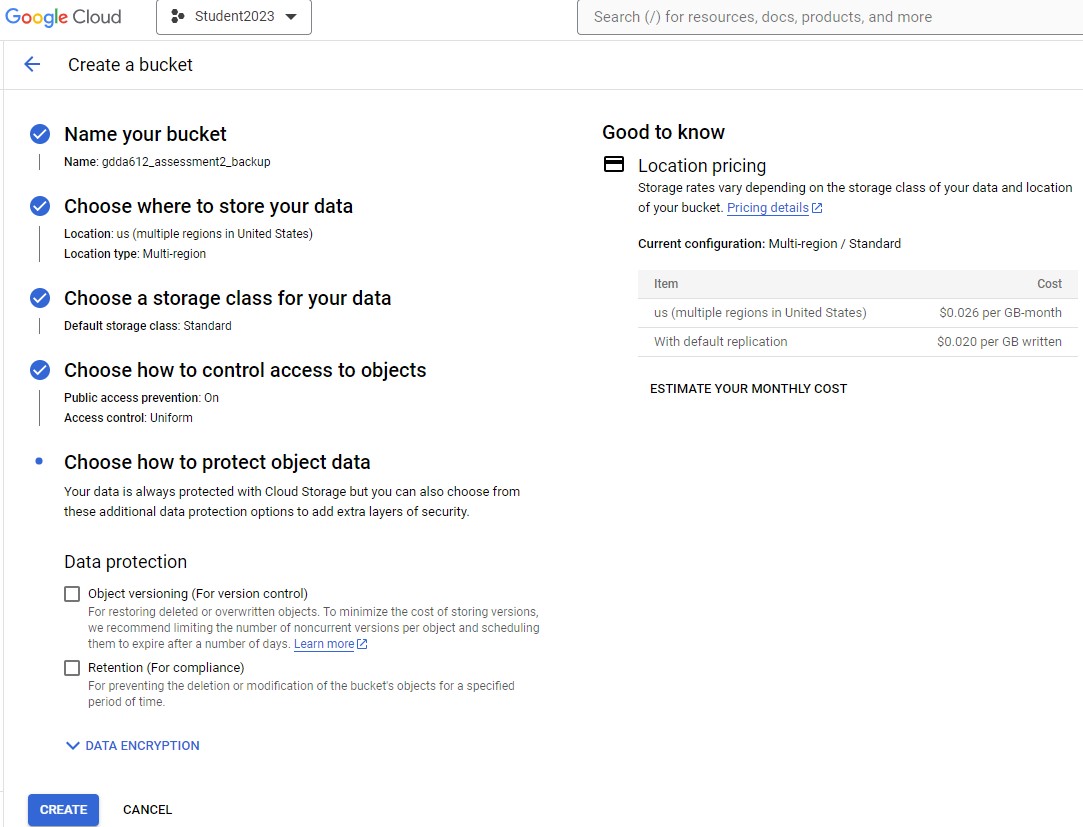
Files" function. I successfully uploaded the JSON file to my Google Cloud bucket. (See figure 27)



*Figure 27*

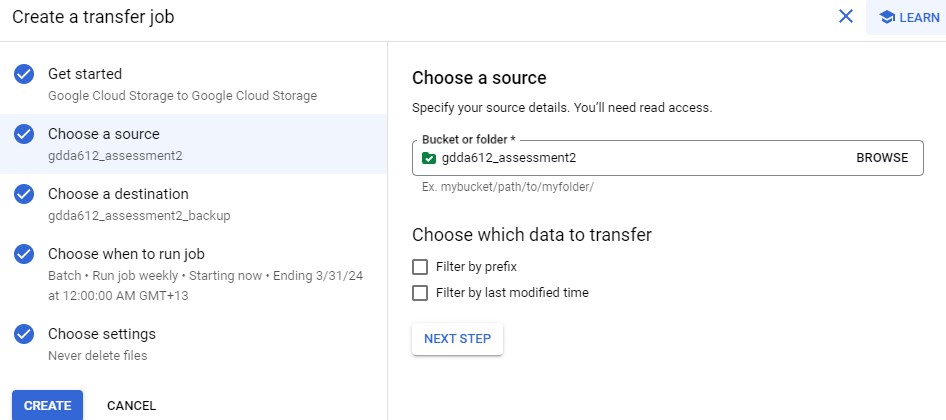
**c) Scheduling automated backups in GCP.**

To create a backup of the imported JSON file, I created another Google Cloud bucket and named it gdda612\_assessment2\_backup. (See figure 28)

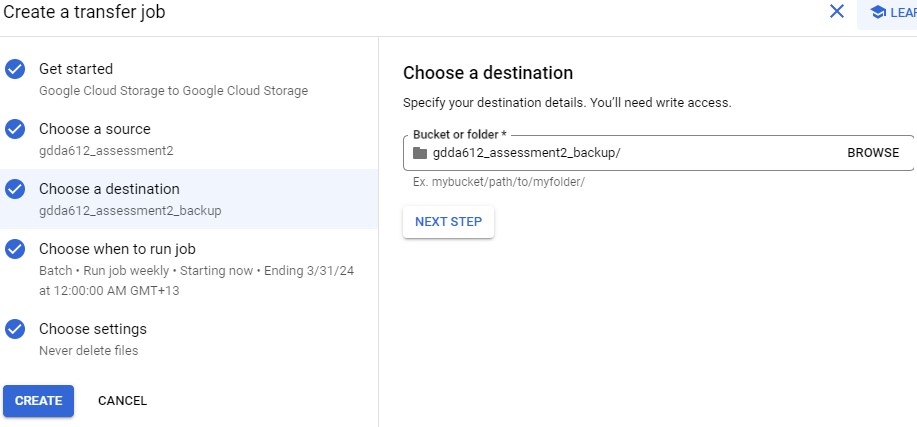


*Figure 28*

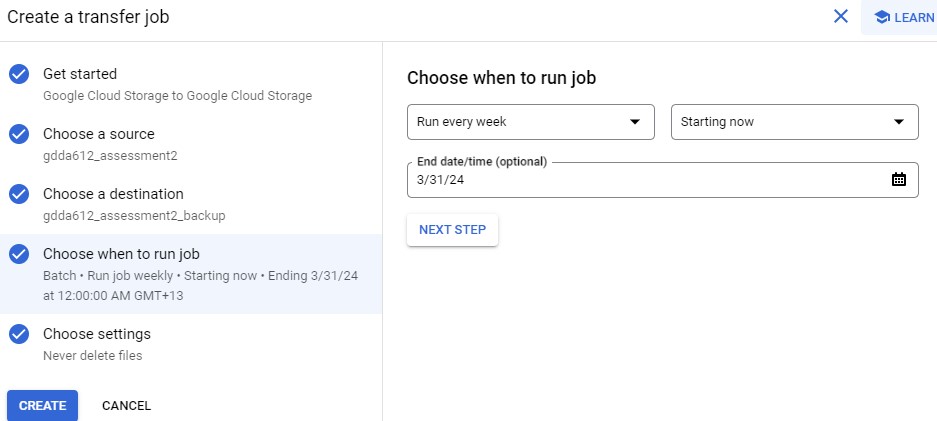
From the Google Cloud backup bucket, I created a transfer job for backing up the collection. I selected gdda612\_assessment2 as the source and gdda612\_assessment2\_backup as the destination. Additionally, I scheduled an automated backup of the specified data collection to occur weekly until March 30th. (See figure 29, 30,31)



*Figure 29*



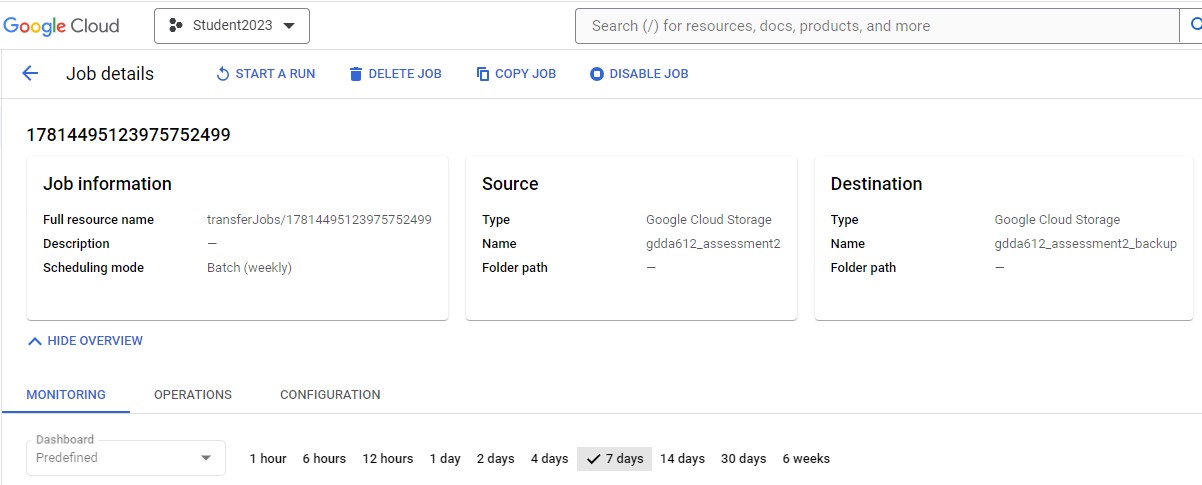
*Figure 30*



*Figure 31*

The figure below demonstrates the successful completion of the backup storage on Google

Cloud, including the automation setup to run until March 31st. (See figure 32)

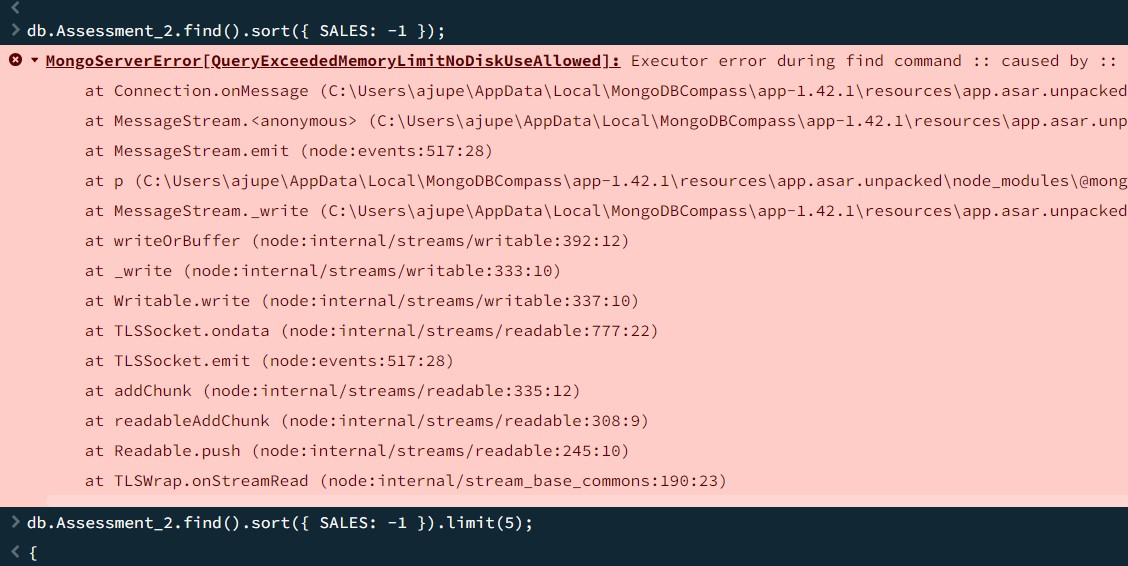


*Figure 32*

**d) Error handling**

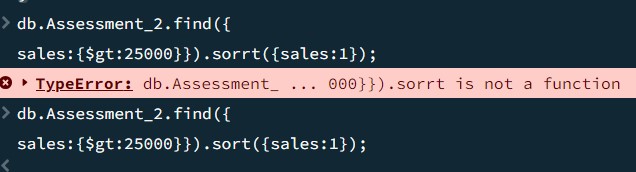
To avoid connectivity issues and file upload errors, I referred to GDDA612 Assessment1. During Assessment 2, I encountered several errors, but unfortunately, I did not save many of them.

Error 1: While executing the sort function, I encountered an error stating that the sort operation exceeded the memory limit of 33554432 bytes. I resolved this issue by appending a limit command to the sort query. (See figure 33)



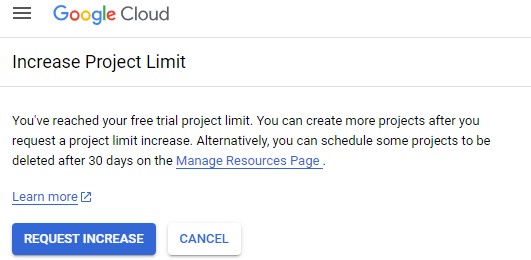
*Figure 33*

Error 2: MongoDB is case-sensitive and sensitive to spelling errors. I encountered this error due to a mistyped sort function. I resolved it by correcting the spelling of the sort function and then executed it successfully. (See figure 34)



*Figure 34*

Error 3: When attempting to create a new Google Cloud bucket, I initiated the creation of a new project in GCP but encountered an "increase project limit" error. This was because, as a user of the free version of GCP, I was limited in the number of new projects I could create. To resolve this issue, I utilized an existing project that had been created earlier in GCP. (See figure 35)

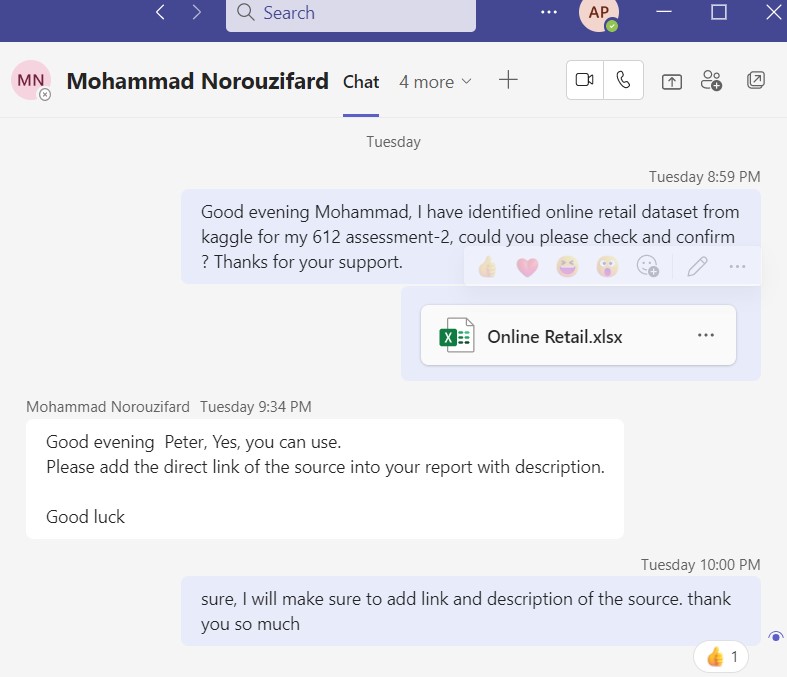


*Figure 35*

Reference: Online Retail dataset downloaded from Kaggle <https://www.kaggle.com/datasets/lakshmi25npathi/online-retail-dataset/data>

Tutor approval:

I have received the Tutor approval on 27th February 2024. (See figure 36)



*Figure 36*

GitHub reference for source code: https://github.com/ajupeter23/GDDA612