## ETL OPERATIONS AND BIGDATA ANALYSIS

A part of GDDA707 Advanced Data Engineering Assessment-2

Prepared by: Aju Peter

Date: 22-03-2024

Contents

**PART A: ETL Operations**

* + Selection of Dataset and Tools
  + Load and Pre-processing
  + ETL Operations and Integration

**PART B: Big Data Analysis**

* + Data pipeline to HDFS
  + Data storage and querying solution
  + Building real-time data clustering system
  + Data streaming pipeline

**Project Report**

* + Executive Summary
  + Technical Summary
  + Conclusion
  + References

### PART A

**Task 1**

I am employed as a data analyst at a giftware e-commerce company, my primary responsibility is ETL operation and choosing the appropriate tools for the operation. The bigdata, which I sourced from Kaggle, the dataset having 8 variables includes 1.InvoiceNo, 2.StockCode, 3.Description, 4.Quantity, 5.InvoiceDate, 6.UnitPrice, 7.CustomerID , and 8.Country.

Both the bigdata having similar attributes:

* InvoiceNo – a unique number for each sales order,
* StockCode – a 5-digit number assigned to each product.
* Description - Product name,
* Quantity - Total quantity of products
* InvoiceDate - Date and Time of the transaction,
* UnitPrice - Price for each product
* CustomerID - A 5 digit unique ID for each customer,
* Country - Customer location

Tool selection for the ETL operation:

* I have chosen Python, a well-known tool for big data engineering and analysis. For this task, I used the Python-based IDE, Jupyter Notebook, supported by the Pandas and NumPy libraries for tasks such as reading data, data exploration, data transformation, data cleaning, merging data, and loading the data into the desired database.

used libraries;

* Pandas: a library for data manipulation and analysis.
* NumPy: a library for statistical analysis.
* Kaggle is an online platform that offers a vast array of datasets for data science projects and data analysis.
* For the semi-structured big data, I utilized MongoDB, a flexible NoSQL database known for its schema flexibility and scalability, making it well suited for storing and analysing big data due to its powerful querying capabilities.
* GCP (Google Cloud Platform) is a collection of cloud services offered by Google, providing computing, databases, storage, machine learning, etc.
* GCP Dataflow is a GCP service that simplifies data processing, allowing for the transformation and analysis of big data.
* BigQuery is another GCP service for big data, enabling the storage and complex analysis of large datasets in a fast manner.

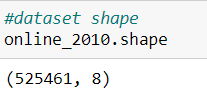
**Task 2**

I started by importing the required libraries to work with the bigdata. After downloading two datasets from Kaggle - the "2010 Online Sales Data" and the "2011 Online Sales Data," both featuring the same attributes, I saved them on my local drive. To read the data, I used the pd.read\_csv() command. However, when loading this semi-structured Big Data, I encountered an error. To address this issue, I utilized the encoding='latin1' parameter, which successfully allowed me to retrieve the entire dataset without any errors.

Python libraries used:

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

To get a quick look of the 2010 online dataset, I utilized the head() function, which displayed a few rows of data. I then checked the dimensions of the dataset using the shape attribute and discovered that the 2010 online dataset consists of 525,461 rows and 8 columns. To explore deeper into the dataset, I used the isnull().sum() command, which revealed that there are 2,928 missing values in the 'Description' column and 107,927 missing values in the 'Customer ID' column. Finally, I used the info() command to get a quick overview of the dataset.

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Similarly, I followed the same process for the 2011 online dataset, I utilized the head() function, which displayed a few rows of data. I then checked the dimensions of the dataset using the shape attribute and discovered that the 2010 online dataset consists of 541910 rows and 8 columns. To explore deeper into the dataset, I used the isnull().sum() command, which revealed that there are 1,454 missing values in the 'Description' column and 135,080 missing values in the 'Customer ID' column. Finally, I used the info() command to get a quick overview of the dataset.

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Data cleaning:

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

* I have used the “shape” function to find size of the dataset.
* I have used the “isnull().sum()” method to identifying and counting null values in the dataset.
* I have used the “info()” method to check the schema of the dataset and type of each variable and the number of non-null values recorded for those variables.

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

A close up of text

Description automatically generatedA white background with black text

Description automatically generatedTo further clean the data, the “duplicated ()” function was employed to identify duplicate records within the dataset. 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.

I found that the 'InvoiceDate' column was of the object data type, so I utilized the “pd.to\_datetime” function to convert the column's data type to Datetime.

A black text on a white background

Description automatically generated

A close-up of a text

Description automatically generated

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.

A close-up of a text

Description automatically generated

A close up of a word

Description automatically generated

After completing the preprocessing the dataset, I have displayed some rows of the cleaned dataset, the head() command was used.

A screenshot of a data

Description automatically generated

A screenshot of a computer

Description automatically generated

Task 3:

I executed an ETL (Extract, Transform, Load) operation to consolidate data from the "2010 Online Sales Data" and the "2011 Online Sales Data" into a unified dataset. Given that both datasets shared similar attributes, I utilized the “concatenate” function to merge them effectively, thereby unifying the datasets into one comprehensive dataset for analysis.



To effectively manage and process the large volumes of data from the unified dataset, Initially, I used Jupyter Notebook as a primary tool for unifying the 2010 and 2011 online sales data due to its versatility in data manipulation and analysis.

After successfully merging the datasets, I chose MongoDB, a NoSQL database, to handle the semi-structured nature of the big data. I began by creating a MongoDB cluster on Atlas, which allowed for scalable and flexible data storage. Eventually, I established a connection from the Jupyter Notebook to the MongoDB cluster.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Once the connection was set up, I proceeded to create a database in MongoDB and within it, a collection where I inserted the unified dataset.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedTo further integrate and analyze the big data at scale, I utilized Google Cloud Platform's BigQuery. The first step involved enabling IP access in MongoDB to allow connections from any location, enhancing the interoperability between MongoDB and GCP.A screenshot of a computer

Description automatically generated

I logged into Google Cloud Platform (GCP) with my credentials and then created a new project on Google Cloud Storage, named as"707A2PartA."

A screenshot of a computer screen

Description automatically generated

Following this, I established a service account through IAM and admin, providing the necessary permissions for data manipulation and analysis.

A screenshot of a service account

Description automatically generated

Then, I created a dataset in BigQuery with the ID "707part\_a" and selected "asia-southeast2 (Jakarta)" as the region to optimize data access and processing speeds locally.

A screenshot of a computer

Description automatically generated

A table was then created within this dataset, with a schema that mirrored the MongoDB collection, including fields such as "\_id" and "timestamp," ensuring data consistency and integrity.

A screenshot of a computer

Description automatically generated

To support the data integration process from MongoDB to BigQuery, I implemented Google Cloud Dataflow. I initiated this by creating a job from a template, naming it "unifieddataset707," and configuring the required parameters. This included pasting the MongoDB connection string, updating the database name, and specifying the collection name. I set the user option to "Flatten" to ensure a smooth data transformation process. Finally, by selecting the project ID "unified\_dataset," I ran the job, enabling the seamless integration and analysis of the unified dataset across MongoDB and GCP BigQuery.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The Dataflow job I set up was executed successfully, ensuring a seamless transfer and processing of data.

A screenshot of a computer

Description automatically generated

Eventually, this processed data was effectively integrated into BigQuery, enabling us to move forward with our data analysis tasks without any issues.

A screenshot of a computer

Description automatically generated

During the ETL and data integration process, particularly while integrating MongoDB with GCP Dataflow and BigQuery, I encountered several challenges. Although there were numerous obstacles, space limitations allow me to highlight only a few:

1. Dataflow Job Naming Issue: Initially, I named my Dataflow job "707\_unifieddataset", which led to a failure because Dataflow job names must start with a letter, not a number. To resolve this, I renamed the job to "unifieddataset707", which allowed the creation process to proceed successfully.

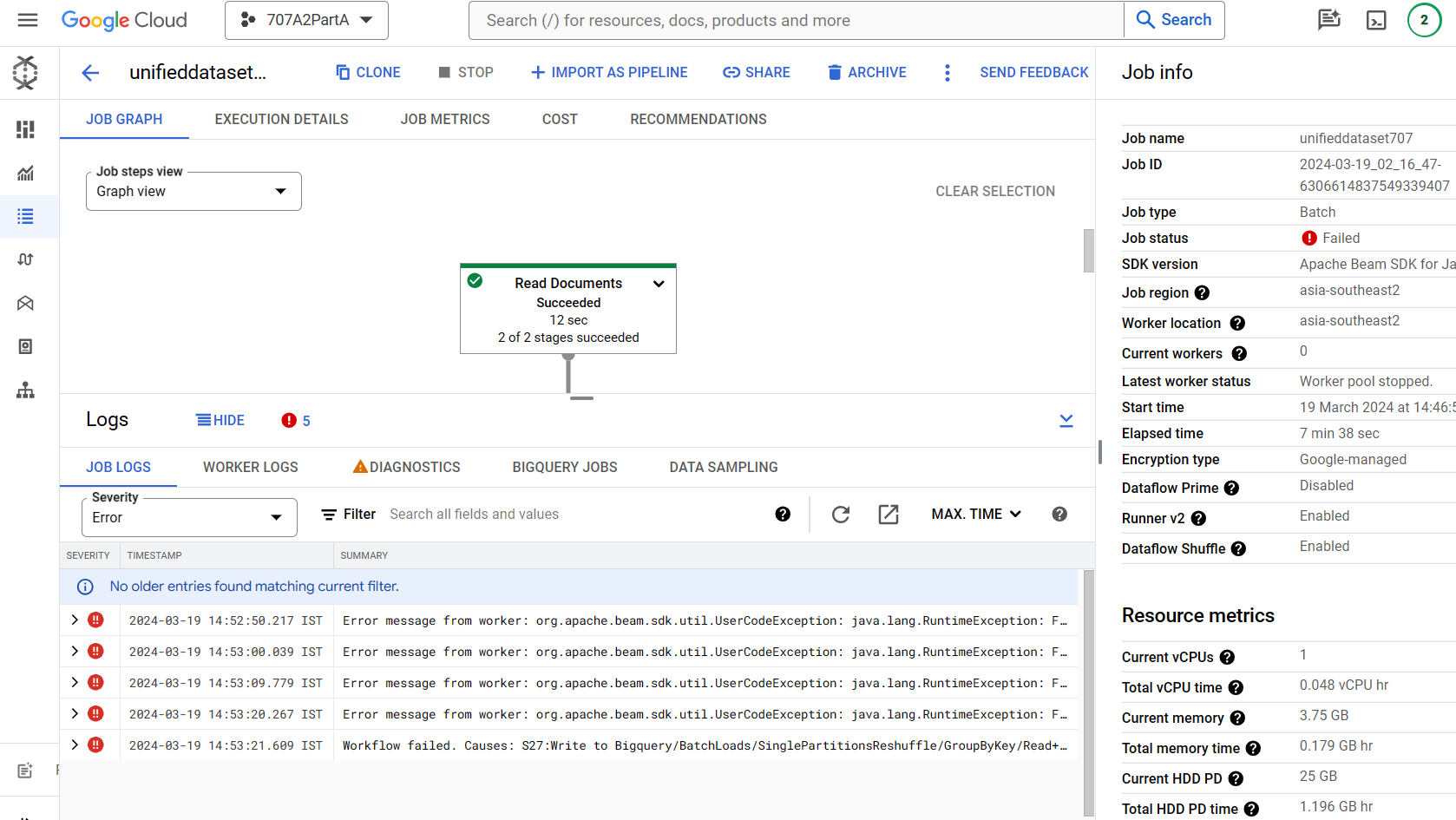
A screenshot of a computer screen

Description automatically generated

A screenshot of a search box

Description automatically generated

1. Schema Mismatch in BigQuery: I faced an issue when creating a table in BigQuery; the schema for the InvoiceDate column was imported from MongoDB as a Date type. However, during the execution of the Dataflow job, an error occurred, indicating the need to convert this column to a string type. BigQuery does not allow direct schema editing, so I had to drop the InvoiceDate column and recreate it as a new field with the String type. After making this adjustment, I reran the Dataflow job, which then integrated successfully into BigQuery.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

These solutions helped overcome the challenges faced, enabling the successful integration of data into the desired platforms.

### PART B: Big Data Analysis

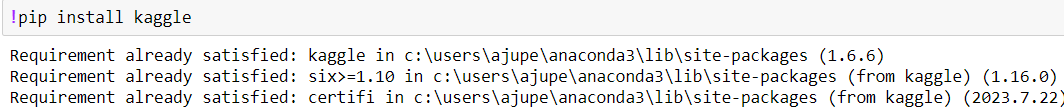
### Task-1

### Two well-known sources were selected for this task: GitHub and Kaggle. To suport the data ingestion process, Python was employed as the primary tool, Jupyter Notebook, a Python-based IDE, for the development environment. The Pandas and Numpy libraries were imported for various stages of the pipeline, including downloading, reading, exploring, transforming, cleaning, merging, and ultimately loading the data into HDFS.

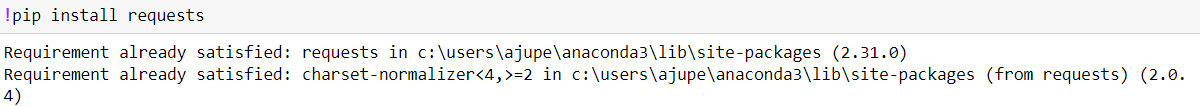
### The data ingestion process commenced with the downloading of customer transaction data from Kaggle. The steps were as follows:

### Kaggle API Installation and Configuration:

### Installation of the Kaggle API in Jupyter Notebook was achieved using the command! pip install kaggle.



### The requests library was also installed using! pip install requests to handle HTTP requests.



### The os library was imported to configure the Kaggle API by setting up environment variables with my Kaggle credentials.

A computer screen shot of a computer code

Description automatically generated

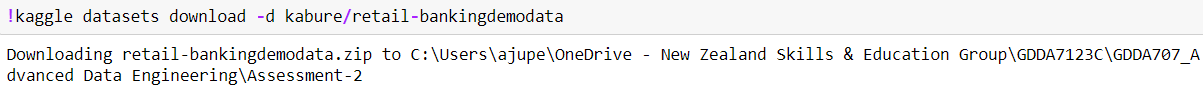
### A Kaggle API token was generated from my Kaggle account, facilitating secure access to Kaggle datasets.

A screenshot of a computer

Description automatically generated

### Dataset Download:

### With Kaggle API configured, the desired dataset was downloaded directly from its Kaggle directory to the local drive of the development environment.



### Data Preparation:

After downloading, the dataset, which came in a ZIP file format, was unpacked using the zipfile library.A computer screen shot of text

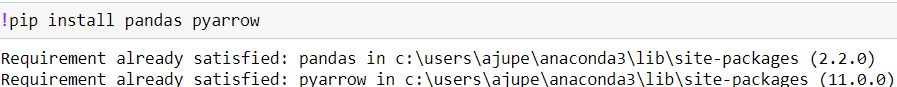
Description automatically generated

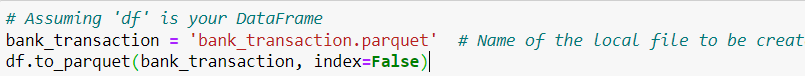
1. The extracted dataset was then read into a Pandas DataFrame using the pd.read\_csv() function for further processing.

A screenshot of a computer screen

Description automatically generated

1. I converted the DataFrame to a Parquet file to efficiently manage the large dataset. For this purpose, I installed the pyarrow library by running pip install pyarrow. Subsequently, I created a Parquet file named bank\_transaction.parquet.





1. To establish a pipeline for loading customer transaction data into HDFS, I began by logging into Google Cloud Platform (GCP) using my credentials. I initiated the process by creating a new project named "GDDA707assessment2." Following this, I set up a Hadoop cluster within Google Cloud Dataproc, opting for a virtual machine (VM) engine that included components such as Zookeeper and Docker.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. To store the data, I created a bucket in Google Cloud Storage, which I named "707bucket."

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Next, I proceeded to integrate Google Cloud Storage with Jupyter Notebook by installing the necessary libraries.



1. I navigated to IAM & Admin within GCP to create a service account, from which I generated a service account key. This key was downloaded in JSON format and saved on my local drive. To facilitate access to Google Cloud Storage from Jupyter Notebook, I configured the operating system to recognize the service account key stored on the local drive.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

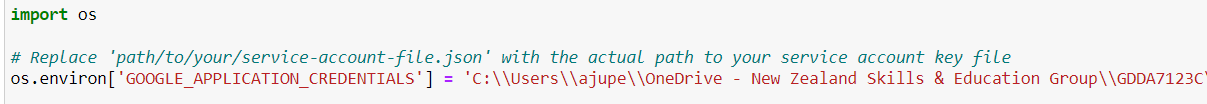
A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



1. I imported the necessary modules for Google Cloud Storage within my Jupyter Notebook environment, setting the stage for data transfer and manipulation directly from the cloud. I established a connection to a Google Cloud Storage bucket from my Jupyter Notebook and then successfully uploaded both the big data to the bucket.

A computer screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

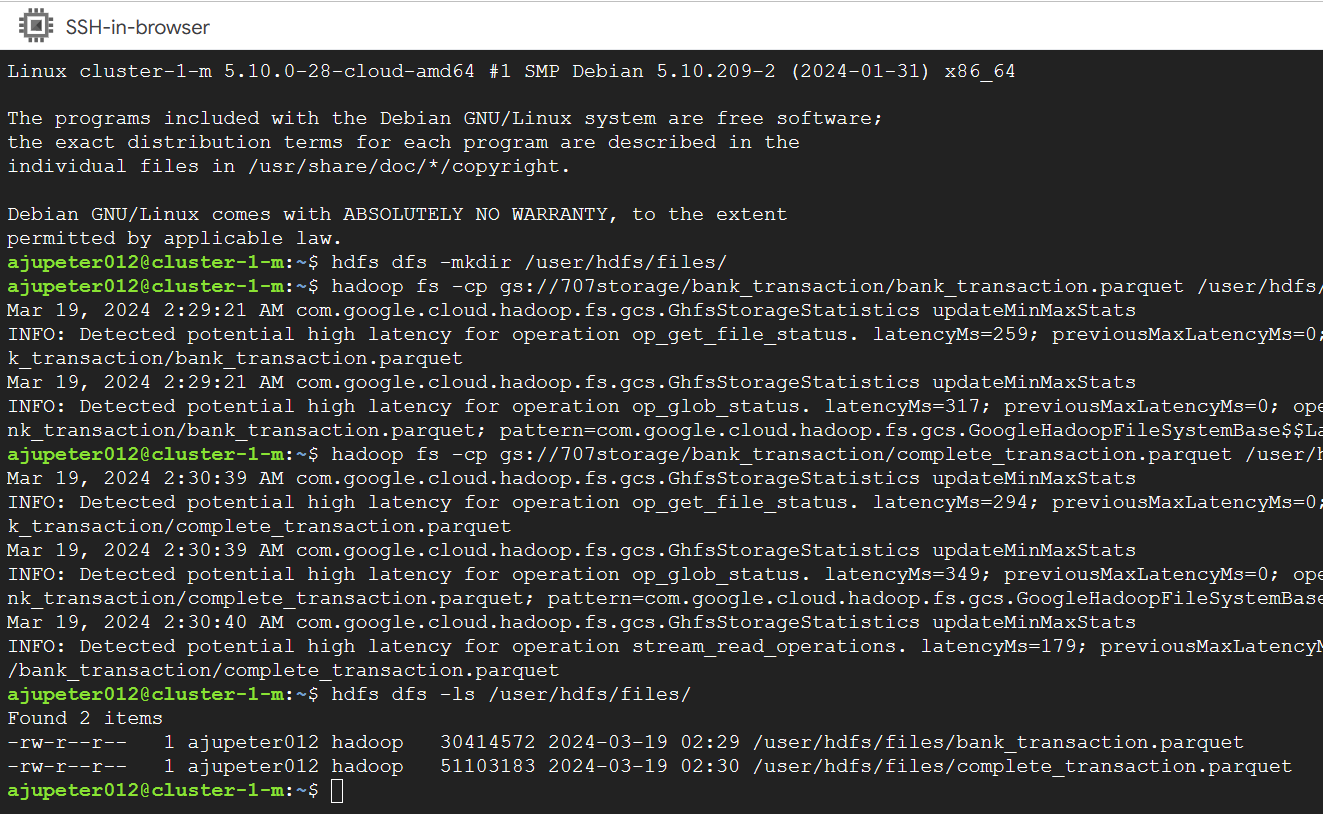
A screenshot of a computer

Description automatically generated

1. From the cluster's VM instance, I accessed SSH in the browser, created a destination file location in Hadoop, and successfully copied the file to that location.

A screenshot of a computer

Description automatically generated



This methodical approach enabled the efficient creation of a data ingestion pipeline into HDFS, leveraging GCP's robust cloud infrastructure and tools.

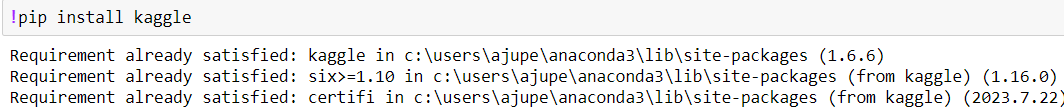
Task-2:

### To implement data storage and querying solution I have chosen MongoDB and selected a customer transaction bigdata from Kaggle, initially Python was employed as the primary tool, Jupyter Notebook, a Python-based IDE, for the development environment.

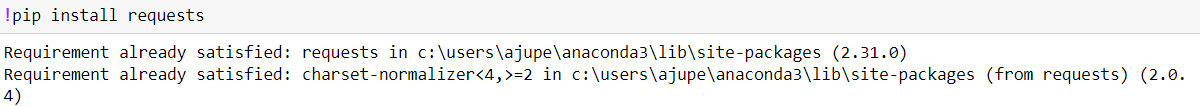
### The data storage and querying solution process started with the downloading of customer transaction data from Kaggle. The steps were as follows:

### Kaggle API Installation and Configuration:

### Installation of the Kaggle API in Jupyter Notebook was achieved using the command “pip install kaggle”.



### The requests library was also installed using “pip install requests” to handle requests.



### The os library was imported to configure the Kaggle API

A computer screen shot of a computer code

Description automatically generated

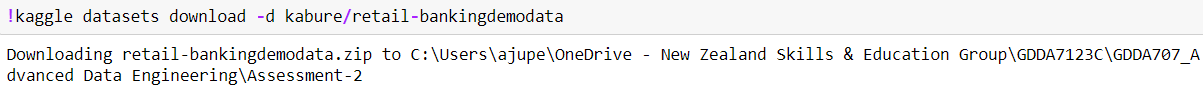
### A Kaggle API token was generated from my Kaggle account, for secure access to Kaggle datasets.

A screenshot of a computer

Description automatically generated

### Dataset Download:

### With Kaggle API configured, the desired dataset was downloaded directly from its Kaggle directory to the local drive of the development environment.



1. Data Preparation:

Upon downloading, the dataset, packaged as a ZIP file, was extracted using the zipfile library.

A computer screen shot of text

Description automatically generated

1. The extracted dataset was then read into a Pandas DataFrame using the pd.read\_csv() function for further processing.

A screenshot of a computer screen

Description automatically generated

1. 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 and columns.
* Identifying and counting null values in the dataset by applying the isnull().sum() method.
* 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.

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated A screenshot of a computer code

Description automatically generated

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

1. To further clean the data, the “drop()” function was employed to drop unwanted columns.

A close-up of a computer screen

Description automatically generated

1. After completing the preprocessing the dataset, I have displayed few rows of the cleaned dataset, the head(2) command was used.

A screenshot of a calendar

Description automatically generated

1. I chose MongoDB, a NoSQL database, to handle the semi-structured nature of the big data. I began by creating a MongoDB cluster on Atlas, which allowed for scalable and flexible data storage. Subsequently, I established a connection from the Jupyter Notebook to the MongoDB cluster. Connection string: “mongodb+srv://ajupeter:student2023@mongodbcluster.s7xdaor.mongodb.net/"

A screenshot of a computer

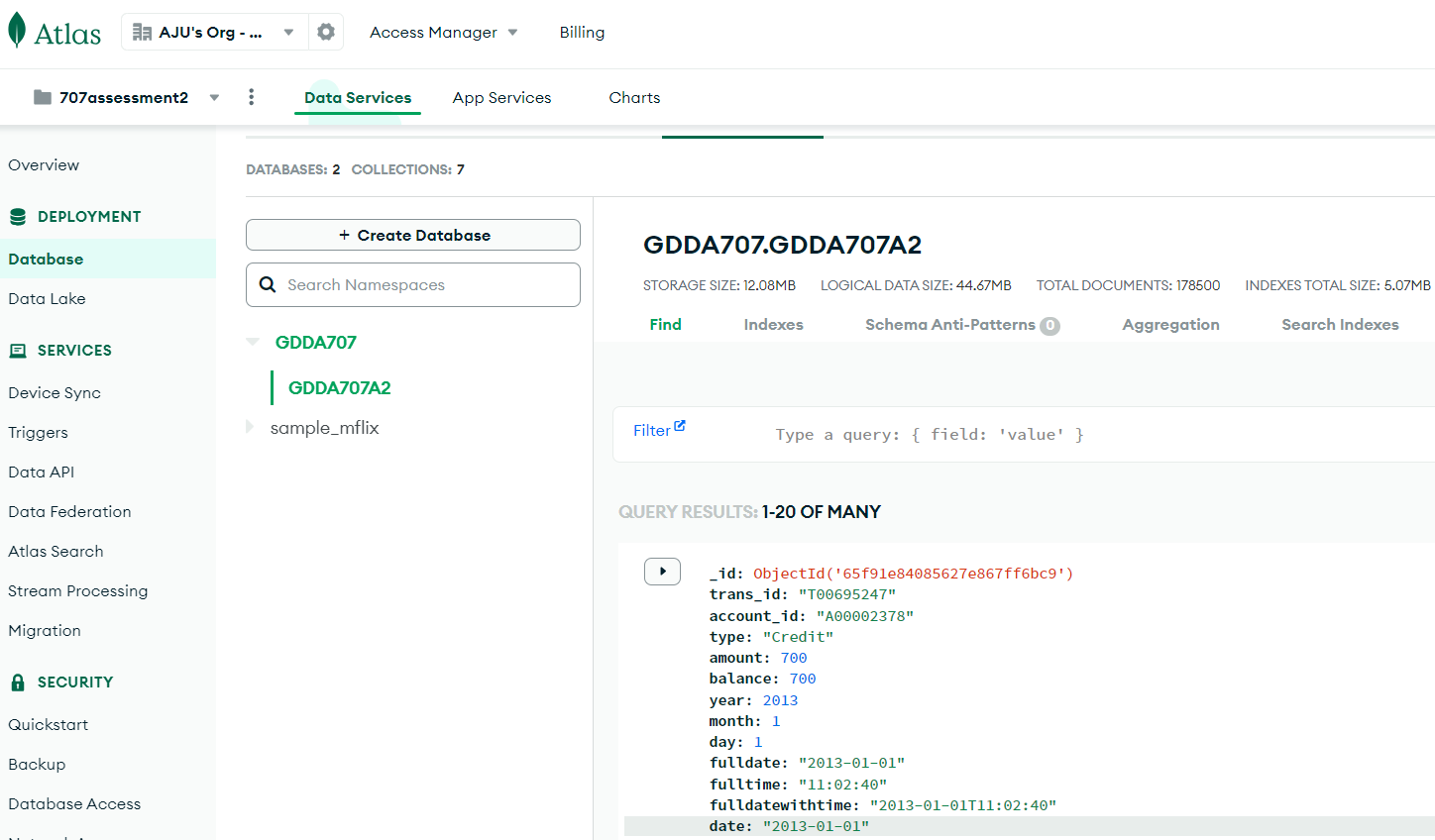
Description automatically generatedA screenshot of a computer

Description automatically generated

1. Once the connection was set up, I proceeded to create a database in MongoDB and within it, a collection where I inserted.

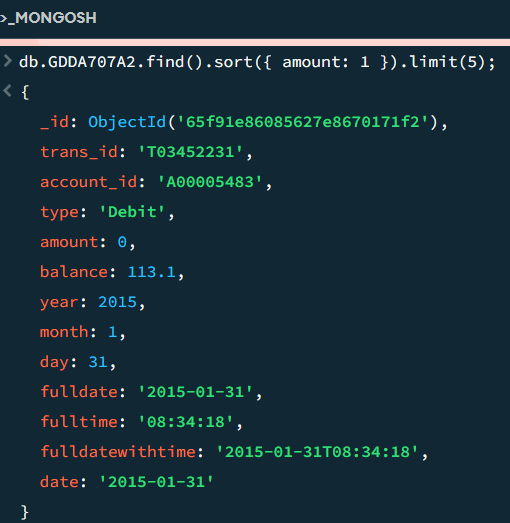
A screenshot of a computer

Description automatically generated



1. A computer screen shot of a code

   Description automatically generatedAfter inserting the collection from Jupyter Notebook into MongoDB, I performed multiple queries in MongoDB, as shown in the screenshot below.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

Task-3:

Created a real-time data clustering framework using Apache Spark on Google Cloud Platform (GCP) for aggregating and analysing data streams from social media networks.

1. I have created a new project as GDDA707assessment2 on the Google Cloud Console.

A screenshot of a computer

Description automatically generated

1. I have enabled and created a new VM instances named as instance-1

A screenshot of a computer

Description automatically generated

1. Enabled Dataproc and created a cluster named as cluster-1

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

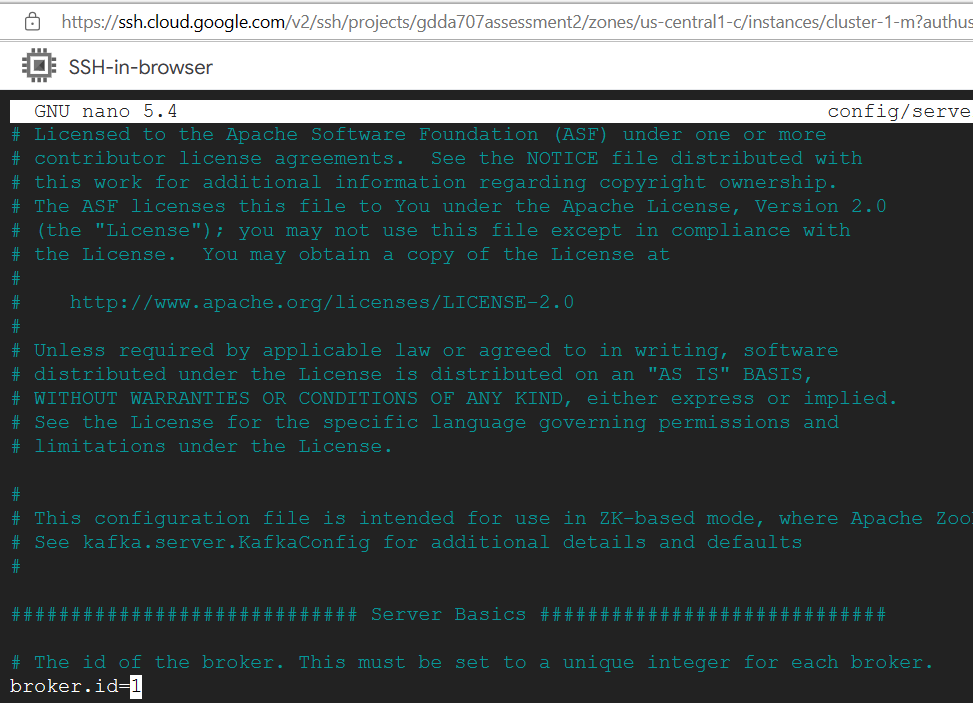
1. Installed Apache Kafka on GCP by Configuring Compute Engine Instances:

I navigated to the Compute Engine section in the Google Cloud Console and set up new instances for my Kafka brokers, ensuring they were equipped with sufficient CPU and memory resources for the needs of my Kafka cluster. Following that, I downloaded the preferred version of Apache Kafka from the official website directly onto these instances.

A screenshot of a computer program

Description automatically generated

1. I edited the server.properties file to configure Kafka, adjusting the broker.id to 1 from the original 0 and updating the listeners property to use the instance IP 34.27.45.168. This was done using the command nano config/server.properties.







1. I started the built-in Zookeeper using the command ~/kafka\_2.12-3.5.2/bin/zookeeper-server-start.sh ~/kafka\_2.12-3.5.2/config/zookeeper.properties.

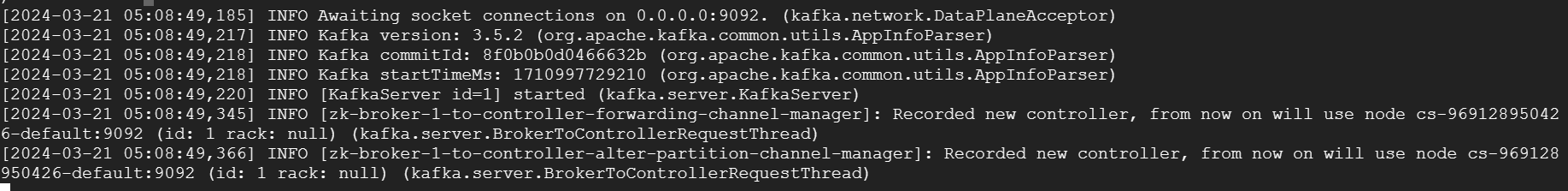
A screenshot of a computer screen

Description automatically generated

A black and white screen

Description automatically generated

7. I initiated the Kafka server and launched the Kafka brokers as planned.



1. I have opened the necessary ports, created a new VPC network, and established a firewall rule named "kafkafirewall1".

A screenshot of a computer

Description automatically generated

\

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. I attempted to create a Kafka topic named facebook-stream for storing Facebook streaming data using the command:

However, I encountered an error indicating bash: ./kafka-topics.sh: No such file or directory. This error occurred when executing the command from the Google Cloud Shell.

A black and white screen

Description automatically generated

1. I've integrated with the Facebook Graph API by setting up and using the facebook-sdk library in Python. This included installing the library with pip install facebook-sdk and creating an account for the Facebook Graph API, which was necessary for access.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

1. Script to Establish a Connection with the Facebook Graph API.

import facebook

access\_token = 'EAAP16axGY0MBO7jewZBZBy9AJUau9VR36y7xIS9gB1Sms1xRbcqt6ZCZBzDP6DIK9p7hAzGYjmhHtNewgZCJBDVFyCZClE6a1rhNDCJHiE8JxaZBR7qw7Aaq0fujmEMZBWaC48vTObLZAiJ6fKtn48oOWx8q57b9lC9giopjCBkAehrCa9gR0q1tMnNfzRYmPZBZBXQXPCm3d0G8GEYThJsKb4qZBBYhofNhst4WFdtZBqwxVu4cXzEEntEXwHwo6zP69VgZDZD'

graph = facebook.GraphAPI(access\_token, version="3.0")

# Example: Get information about the current user

me = graph.get\_object('me')

print(f"User ID: {me['id']}, Name: {me['name']}")

A screenshot of a computer

Description automatically generated

1. I opened SSH in a new window and attempted to initiate the Spark installation using the command gs://dataproc-initialization-actions/spark/spark.sh. However, upon execution, I encountered an error indicating "no such directory". This issue occurred while trying to verify the Spark installation on the master node.

A screenshot of a computer screen

Description automatically generated

Finally, I have paused the construction of the real-time data clustering system using Apache Spark for streaming and incorporating data from social media platforms. The decision comes after facing continuous issues and recognizing a need for further research and study, which the timeframe of my current assessment does not allow.

Task-4:

I have developed a data streaming pipeline using Kafka to facilitate data integration for analytical purposes.

1. I have created a new project as GDDA707assessment2 on the Google Cloud Console.

A screenshot of a computer

Description automatically generated

1. I have enabled and created a new VM instances named as instance-1

A screenshot of a computer

Description automatically generated

1. Enabled Dataproc and created a cluster named as cluster-1

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

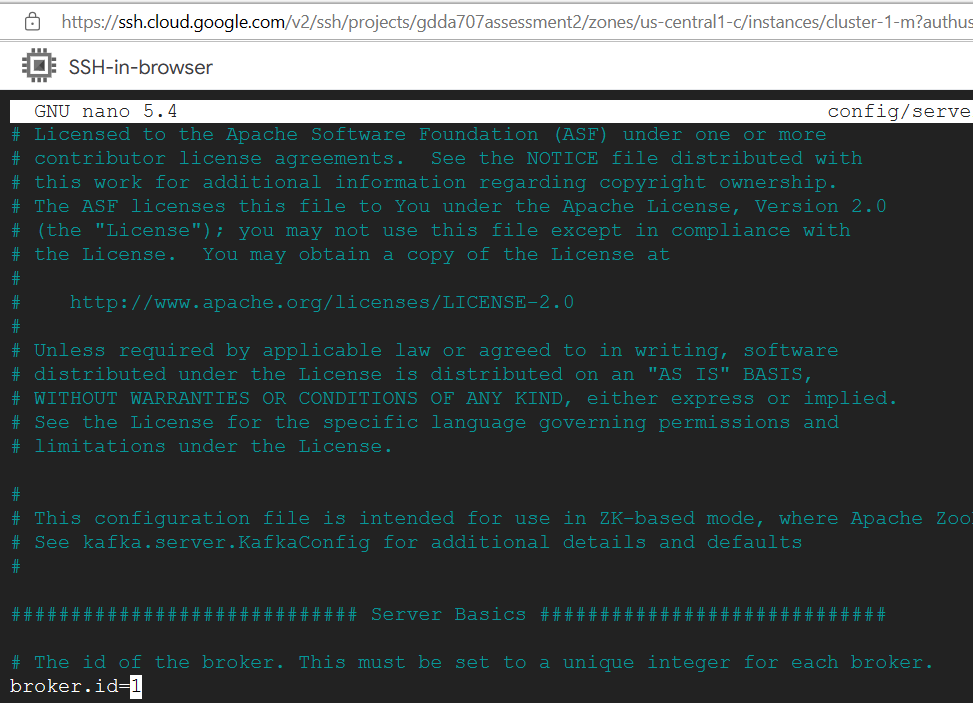
1. Install Apache Kafka on GCP by Configuring Compute Engine Instances:

I navigated to the Compute Engine section in the Google Cloud Console and set up new instances for my Kafka brokers, ensuring they were equipped with sufficient CPU and memory resources for the needs of my Kafka cluster. Following that, I downloaded the preferred version of Apache Kafka from the official website directly onto these instances.

A screenshot of a computer program

Description automatically generated

1. I edited the server.properties file to configure Kafka, adjusting the broker.id to 1 from the original 0 and updating the listeners property to use the instance IP 34.27.45.168. This was done using the command nano config/server.properties.



A computer screen shot of a computer code

Description automatically generated

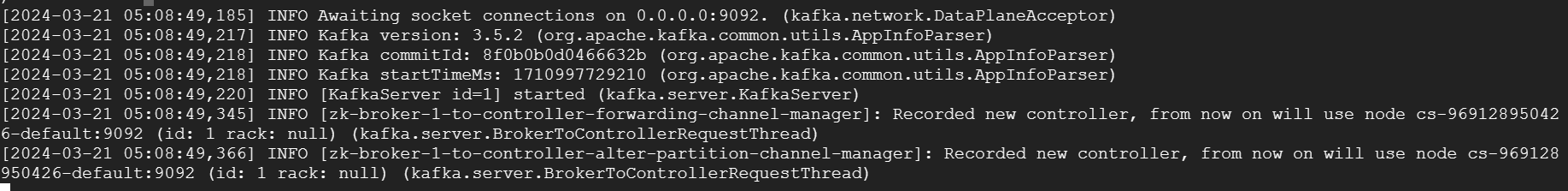


1. I started the built-in Zookeeper using the command ~/kafka\_2.12-3.5.2/bin/zookeeper-server-start.sh ~/kafka\_2.12-3.5.2/config/zookeeper.properties.

A screenshot of a computer screen

Description automatically generated

1. I initiated the Kafka server and launched the Kafka brokers as planned.



1. I have opened the necessary ports, created a new VPC network, and established a firewall rule named "kafkafirewall1".

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. I attempted to create a Kafka topic named facebook-stream for storing Facebook streaming data using the command:

However, I encountered an error indicating bash: ./kafka-topics.sh: No such file or directory. This error occurred when executing the command from the Google Cloud Shell.

A black and white screen

Description automatically generated

1. I've integrated with the Facebook Graph API by setting up and using the facebook-sdk library in Python. This included installing the library with pip install facebook-sdk and creating an account for the Facebook Graph API, which was necessary for access.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

1. Script to Establish a Connection with the Facebook Graph API.

import facebook

access\_token = 'EAAP16axGY0MBO7jewZBZBy9AJUau9VR36y7xIS9gB1Sms1xRbcqt6ZCZBzDP6DIK9p7hAzGYjmhHtNewgZCJBDVFyCZClE6a1rhNDCJHiE8JxaZBR7qw7Aaq0fujmEMZBWaC48vTObLZAiJ6fKtn48oOWx8q57b9lC9giopjCBkAehrCa9gR0q1tMnNfzRYmPZBZBXQXPCm3d0G8GEYThJsKb4qZBBYhofNhst4WFdtZBqwxVu4cXzEEntEXwHwo6zP69VgZDZD'

graph = facebook.GraphAPI(access\_token, version="3.0")

# Example: Get information about the current user

me = graph.get\_object('me')

print(f"User ID: {me['id']}, Name: {me['name']}")

A screenshot of a computer

Description automatically generated

1. I opened SSH in a new window and attempted to initiate the Spark installation using the command gs://dataproc-initialization-actions/spark/spark.sh. However, upon execution, I encountered an error indicating "no such directory". This issue occurred while trying to verify the Spark installation on the master node.

A screenshot of a computer screen

Description automatically generated

I have put the development of the data streaming pipeline using Kafka for data integration and analysis on hold due to insufficient time for research and to meet the submission deadline of the assessment.

**Section 1:**

This project was focused on big data from an e-commerce company selling giftware, using data sourced from Kaggle that covered transactions from 2010 and 2011. The main goal was to Extract, Transform, and Load. I chose Python for handling the data, MongoDB for storing it, and Google Cloud Platform's services for further processing and analysis.

The background of this project lies in the need to understand customer behaviour and market trends through data. The online sales data included details like order numbers, product codes, product names, quantities sold, transaction dates, prices, customer IDs, and countries.

I started with an ETL (Extract, Transform, Load) process using Python libraries like Pandas and NumPy. I faced some initial errors like encoding issues and missing values but resolved them by applying appropriate data cleaning techniques. After cleaning, I merged the 2010 and 2011 datasets into one.

Then, I loaded the cleaned data into MongoDB, a flexible database suitable for our semi-structured big data. Finally, I used Google Cloud Platform's BigQuery for detailed analysis and Dataflow for efficiently transferring our data between MongoDB and BigQuery.

I faced challenges, especially with integrating MongoDB with Google Cloud services. For example, I had to change the data type of some columns to match BigQuery's requirements and rename our Dataflow job due to naming structures. Despite these challenges, I successfully cleaned and merged the datasets for analysing them to extract useful insights.

In summary, this project showed the power of combining modern data processing tools and cloud services to analyse big data. I overcome technical challenges and provide a dataset ready for in-depth analysis.

**Section 2:**

**Part A: Execution of ETL Operations**

Objectives: The step ETL operations on a bigdata from Kaggle, which represented sales transactions for a giftware e-commerce company across 2010 and 2011. The process involved data cleaning, transforming, merging, and loading the dataset into MongoDB.

Methodological Approach:

Data Extraction: Used the Pandas library in Python to import the bigdata from Kaggle.

Data Transformation and Cleaning: Utilized Pandas for data cleaning tasks such as identifying missing values, eliminating duplicate records, and converting the data types (changing 'InvoiceDate' to datetime type).

Data Merging: Combined the 2010 and 2011 datasets into a single dataset using Pandas' concatenate function.

Data Loading: Uploaded the processed dataset into MongoDB, selected for its schema flexibility, and proceeded with data integration into GCP for enhanced processing and analytical capabilities.

Technical Specifics:

Python (Jupyter Notebook): Jupyter Notebook IDE used for executing code, manipulating data, and analytical tasks.

Pandas and NumPy: for data manipulation and data cleaning processes.

MongoDB: Served as the storage solution for the unified dataset, maximise on its schema adaptability and querying strength.

GCP Dataflow: the processing needs for the voluminous data, navigating through naming convention challenges.

GCP BigQuery: Handled complex data analysis and storage, resolving schema discrepancies through data type adjustments and schema editing.

**Part B: Big Data Analysis**

Objectives: This phase focused on applying data engineering methods to big data platforms for efficient data ingestion, storage, and querying.

Methodological Approach:

Data Ingestion: Employed Kaggle APIs for downloading customer transaction data.

Data Storage and Querying: Choosen MongoDB due to its effectiveness in managing semi-structured data, establishing a MongoDB cluster on Atlas for data storage.

Integration and Analysis: After transferring the data to Google Cloud Platform's BigQuery via Dataflow, comprehensive analysis of the data was enabled.

Technical Specification:

Kaggle API: Enabled seamless dataset acquisition from Kaggle, crucial for the initial data ingestion phase.

Jupyter Notebook with Pandas and NumPy: Crucial for the data preparation, cleansing, and transformation stages.

MongoDB Atlas: Managed the database and collections, ensuring effective data storage and management.

Google Cloud Storage: Acted as the intermediary data storage before GCP Dataflow processing and BigQuery analysis.

GCP Dataflow and BigQuery: Used large-scale data processing and analysis challenges, fulfilling the project's technical requisites.

This section details the specific tasks, approaches, and technical aspects of the project. It emphasizes the deliberate use of certain tools and platforms to meet the project's goals in Parts A and B.

**Conclusion**

Through this project Part A, I gone through into the sales data of a giftware e-commerce company, employing ETL operations to uncover significant insights. Utilizing Python, Jupyter Notebook, Pandas, NumPy, MongoDB, and Google Cloud Platform (GCP) services like Dataflow and BigQuery, we navigated through the complexities of big data from Kaggle. In addition to ETL operations, Part B focused on the analysis of unified big data to provide deeper insights. The importance of data engineering techniques on big data platforms, supports the analytical capabilities of MongoDB, Dataflow, and BigQuery.

Challenges Encountered:

Sourcing Big Datasets: Initially, finding a dataset that was both large enough and relevant caused a challenge. Kaggle and GitHub proved to be an invaluable resource in this search.

Technical challenges: I faced difficulties with schema mismatches in BigQuery and adhering to naming conventions in Dataflow, which were critical in the ETL process.

Solutions Implemented:

To overcome the dataset challenge, I refined search strategy on Kaggle, focusing on datasets that met our project's scope and depth requirements.

For the technical challenges:

BigQuery Schema Mismatch: I modified the schema by adjusting the 'InvoiceDate' column, ensuring compatibility with BigQuery's requirements.

Dataflow Naming Convention: Renaming the Dataflow job to meet the platform's specifications allowed us to proceed without further issues.

**Reference:**

Part A : The bigdata collected from Kaggle

* <https://www.kaggle.com/datasets/mathchi/online-retail-ii-data-set-from-ml-repository>

(“The UK registered online Retail bigdata contains complete transaction from 01-12-2009 till 09-12-2011. The company sells a complete range of giftware. Majority of customers are wholesalers.”)

Part B : The bigdata collected from Kaggle and GitHub

* <https://github.com/microsoft/DataStoriesSamples/blob/master/samples/FraudDetectionOnADL/Data/transactions.csv>
* <https://www.kaggle.com/datasets/kabure/retail-bankingdemodata?select=completedtrans.csv>
* Week 16\_S3\_Lab Tasks Spark and kafka on GCP
* GitHub link: <https://github.com/ajupeter23/GDDA707>