Li Tao

850001413

GDDA707

Assessment2 e-Portfolio

Table of Contents

[PART A: ETL Operations 2](#_Toc174562963)

[Task 1: Selection of Dataset and Tools 2](#_Toc174562964)

[Clarify and describe the chosen scenario and datasets 2](#_Toc174562965)

[Select appropriate platforms for ETL operations and data engineering on big data 3](#_Toc174562966)

[Task 2: Load and Pre-processing 5](#_Toc174562967)

[Load data from selected datasets to process 5](#_Toc174562968)

[Cleanse, format, and transform data for further operations 6](#_Toc174562969)

[Task 3: ETL Operations and Integration 9](#_Toc174562970)

[Perform ETL operations to integrate data from different datasets into a unified dataset 9](#_Toc174562971)

[Use chosen tools/platforms to effectively manage and process large volumes of data. 11](#_Toc174562972)

[Document challenges faced during ETL and integration, along with solutions implemented. 11](#_Toc174562973)

[PART B: Big Data Analysis and Application of Engineering Techniques 13](#_Toc174562974)

[Task 1: Develop a data ingestion pipeline to load data from various sources into HDFS 13](#_Toc174562975)

[Task 2: Implement a data storage and querying solution using MongoDB 20](#_Toc174562976)

[Task 3: Build a real-time data clustering system using Apache Spark 24](#_Toc174562977)

[Task 4: Set up a data streaming pipeline with Kafka 30](#_Toc174562978)

[Project Report 34](#_Toc174562979)

[Section 1: Summary of the project 34](#_Toc174562980)

[Section 2: Task-Specific Description and Technical Details 36](#_Toc174562981)

[Section 3: Ethical Data Analysis Considerations 39](#_Toc174562982)

[Section 4: Conclusion 40](#_Toc174562983)

[Section 5: References 45](#_Toc174562984)

# PART A: ETL Operations

## Task 1: Dataset Selection and Tool Justification

### Clarify and describe the chosen scenario and datasets

In this task, I chose a scenario in the customer service industry, focusing on integrating customer support information and agent performance data to analyse and optimise customer service quality and efficiency. This involves understanding customer interactions, evaluating agent performance, and identifying areas for improvement.

The datasets ‘customer\_support.csv’ and ‘agent\_info.csv’ were downloaded from Kaggle.

Customer support dataset:

This dataset contains information about customer support records, including their unique ID identifying each customer support record, channel name of the medium, category or type of the issue, remarks relating to customer comments and notes, issue reported and response time, agent name, and other relevant fields.

Agent information dataset:

This dataset contains information about agent records, including agent name, supervisor name, manager name, agent shift timing and working hours, CSAT score, and other pertinent details.

I performed ETL operations on the two datasets by cleaning, transforming, and integrating the datasets into a unified dataset as necessary by joining the common column ‘Agent\_name’ utilising Spark in the Python environment.

### Select ETL tools for operations and big data engineering

I chose Apache Spark and PySpark as the platform for ETL operations and big data processing. PySpark is a tool by the Apache Spark community for using Python with Spark, offering a PySpark shell to link Python APIs with Spark core to initiate Spark context.

This selection is based on the following key factors:

* Performance and capabilities:

Spark can handle large-scale datasets, providing efficient distributed data processing capabilities. In addition, Spark offers powerful data loading, transformation, and integration functions. Spark’s in-memory computing capabilities speed up the ETL process and reduce data processing time. PySpark is faster and more efficient than libraries like Pandas for handling big data, improving productivity and reducing processing time.

* Compatibility:

Spark supports various data sources and formats, such as (CSV, JSON, Parquet, etc.), facilitating data loading and processing. Moreover, Spark integrates well with Hadoop’s ecosystem, leveraging HDFS for storage and YARN for resource management, providing a cohesive big-data processing environment. PySpark enables seamless integration with Python’s libraries like Pandas, Numpy, and Scikit-learn, empowering and leveraging tools for big data analytics.

* Advanced analytics:

Spark’s advanced built-in libraries allow for analytics directly within the ETL pipeline. In addition, Spark streaming allows for real-time processing of data streams, making it suitable for applications that require immediate insights from continuously generated data. PySpark provides fault tolerance, and in-memory computation, and runs on machines without hard drives, enhancing reliability and scalability for big data applications.

Hence, Apache Spark is an ideal platform for ETL operations and data engineering on big data due to its performance, compatibility with big data ecosystems, and advanced analytics capabilities.

## Task 2: Data Loading and Pre-processing

### Import data from the chosen datasets to process

I used Google Colab to perform the following tasks with PySpark in Python.

|  |
| --- |
| After I installed Java, Spark and set up the environment, I imported the required libraries.  A screenshot of a computer program  Description automatically generated |
| I initialized the Spark session successfully. The application ID is a unique identifier showing I am running Spark in local machine, with the application name helping to identify the job.  A screen shot of a computer program  Description automatically generated |
| I loaded the CSV files into Spark DataFrame and displayed their contents. |
| A screenshot of a computer  Description automatically generated |

### Cleanse, format, and transform data for further operations

I started by Inspecting the dataset using the ‘printSchema()’ method from the ‘pyspark.sql’ module.

|  |
| --- |
| I found that some time-related data was in string format, and I would perform data transformations to convert these data types to date-time in the next steps.  A screenshot of a computer  Description automatically generated |
| I checked for missing values in each column.  I found some columns, ‘Customer\_City’, ‘Product\_category’, and ‘Item\_price’, had missing values and were deemed useless, so I decided to drop them. The column ‘Customer Remarks’ contains important information, so I would fill in the null value. The column ‘connected\_handling\_time’ equals to ‘issue\_responded’ time minus ‘issue\_reported\_at’ time, so I will calculate it accordingly.  A screenshot of a computer  Description automatically generated |
| I used ‘to\_timestamp()’ function to convert the ‘Issue\_reported\_at’ and ‘issue\_responded’ columns from string representation of dates and time into timestamps data types.  A screenshot of a computer  Description automatically generated |
| I calculate the ‘connected\_handling\_time’ by subtracting the ‘issue\_reported at’ time from the ‘issue\_responded’ time and then converted the result to minutes.  A screenshot of a computer program  Description automatically generated |
| I droped the irrelevant columns ‘Order\_id’, ‘order\_date\_time’, ‘Survey\_response\_Date’, ‘Customer\_City’, ‘Product\_category’, and ‘Item\_price’ that are not needed for the future analysis from ‘customer\_df’ and filled null values in the ‘Customer remarks’ column with ‘No Remarks’. |
| I checked for missing values again and found that some rows have a very small amount of missing values.  A screenshot of a computer  Description automatically generated |
| I decided to drop rows where any value was missing.  The dataset is now cleaned, with 0 missing values.  A screenshot of a computer  Description automatically generated |

## Task 3: ETL Operations and Integration

### Perform ETL operations to integrate data from different datasets into a unified dataset

|  |
| --- |
| I checked the shape of each dataset.  A screenshot of a computer code  Description automatically generated |
| I grouped the dataset by the ‘Agent\_name’ column and then aggregated the columns in ‘agent\_df’, so that I could find out if these columns were having unique values for each agent.  I found that, except for ‘CSAT Score’ which varied among different agents, each agent had only one unique value for ‘Tenure Bucket’, ‘Agent shift’, ‘Supervisor’, and ‘Manager’.  A screenshot of a computer  Description automatically generated |
| Based on the findings, I decided to aggregate the ‘agent\_df’ dataset by using the first occurrence of ‘Tenure Bucket’, ‘Agent shift’, ‘Supervisor’, and ‘Manager’ for each agent, while computing the average of the ‘CSAT Score’ for each agent.  A screenshot of a computer  Description automatically generated |
| I joined the ‘customer\_df’ dataset with the ‘aggregated\_agents’ dataset on the ‘Agent\_name’ column using a left join operation. This ensures that all rows from ‘customer\_df’ will be kept, and matching rows from ‘aggregated\_agents’ will be added where available.  A screenshot of a computer  Description automatically generated |
| I counted and printed the shape of the ‘merged\_df’. There were 85896 rows and 14 columns in the ‘merged\_df’ dataset.  A screenshot of a computer code  Description automatically generated |

### Use chosen tools to manage and process large volumes of data.

In this task, I went with Apache Spark and performed the data integration in the previous task, because it is great at handling and processing large-scale data. By using Apache Spark, I was able to manage and process the large dataset efficiently. For example, Spark’s ability to distribute the processing load across a cluster of nodes ensures that the ‘customer\_df’ dataset can be handled swiftly. Spark’s in-memory computing spends up data processing, especially when I need to access data frequently. Plus, its distributed processing capabilities make it efficient and reliable, which is perfect for working with really big datasets.

When saving the processed data, I chose to save the data as a Parquet file instead of a CSV file. Unlike row-based formats like CSV, Parquet stores data in a columnar format, making it faster to read specific columns. This approach is optimized for both performance and storage efficiency.

### Document challenges faced during ETL and integration along with solutions

During the ETL and data integration process, I encountered various challenges. For instance, setting up and configuring the Spark and Java environments required a certain level of understanding and practice, which took me some time. Additionally, some minor issues arose during the aggregation and integration operations, but they were eventually resolved. Here I would describe the issues and solutions in detail.

When cleaning the data, I found that many columns in the ‘customer\_df’ dataset had missing values. Making decisions on how to handle these missing values is crucial and should be based on several factors, relevance to data analysis and future use. Since the research purpose is focused on customer experience and agent performance, 6 columns were identified as not directly related: ‘Order\_id’, ‘order\_date\_time’, ‘Survey\_response’, ‘Customer\_City’, ‘Product\_category’, and ‘Item\_price’. I decided to drop these columns. On the other hand, some of the data types were inappropriate. For example, time-related data was represented as a string. To ensure accurate analysis, I needed to convert these data types to datetime types.

After cleaning the data, I encountered another issue with attempting to merge the datasets. The shape of the two datasets was different, with inconsistent numbers of columns and rows. Additionally, there were duplicate entries in the agent dataset. To merge the two datasets, I needed to address these discrepancies. I found that the agent dataset should be aggregated, grouping by the ‘Agent\_name’ column to compile information for each agent. For example, each agent has a single and unique supervisor, manager, shift, and tenure. However, their CSAT (customer satisfaction) scores varied, as these scores are assigned based on individual phone calls. To consolidate the tables, I took the average of the CSAT scores and assigned them to each agent. This allowed me to merge the agent dataset into the customer dataset by the ‘Agent\_name’ column using the left join operation.

# PART B: Big Data Analysis and Engineering

## Task 1: Create a pipeline to load data into HDFS

In this task, I set up and configured Hadoop in Docker to load the dataset. The dataset, ‘online\_customer\_transaction’ which I downloaded from Kaggle, includes 52,955 rows and 21 columns. It contains detailed information on customer purchases, such as customer details, product information, and transaction specifics. With a large volume of customer interactions, it helps to capture the customers’ purchasing patterns and preferences. Additionally, the dataset offers valuable insights into buying behaviour through detailed information on product SKUs, descriptions, and pricing.

* Environment setup:

First, I downloaded and installed Docker to manage and run containerized applications. Docker simplifies the deployment and management of Hadoop services.

Next, I utilized the ‘docker-compose.yml’ file to define and start the Hadoop cluster with HDFS. This file specifies services for NameNode, DataNode, ResourceManager, and NodeManager, ensuring that all necessary components of the Hadoop ecosystem are properly configured and operational.

NameNode: manage the metadata of HDFS.

DataNode: store the actual data blocks.

ResourceManager: manage resources and job scheduling.

NodeManager: manage individual node resources.

|  |
| --- |
| Please see the docker-compose file below:  A screenshot of a computer  Description automatically generated  A screenshot of a computer program  Description automatically generated  A close-up of a computer  Description automatically generated |

Then, I navigated to the directory containing the ‘docker-compose.yml’ file and executed the ‘docker-compose up’ command. This command started the Hadoop containers in detached mode, allowing them to run in the background.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

After executing the command, I checked the containers in the Docker desktop and confirmed that they were running properly.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* Cluster monitoring:

I accessed the HDFS web interface ‘http://localhost:50070’ to check the Hadoop cluster status.

|  |
| --- |
| A screenshot of a computer  Description automatically generated  A screenshot of a login  Description automatically generated |

I also accessed the YARN resource manager web UI ‘http://localhost:8088’ to monitor and manage cluster resources and applications.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* Data interaction:

Then I entered the NameNode container using the command to interact with HDFS to manage and verify files. For example, I listed the files in the HDFS root directory.

|  |
| --- |
| A close-up of a computer code  Description automatically generated |
| A screenshot of a computer  Description automatically generated |

* Upload and verification:

I uploaded a local file to the HDFS under ‘mydirectory’.

|  |
| --- |
| I copied the local file to the docker container using ‘docker cp’ command. |
| Then I went back inside my NameNode container using ‘hdfs dfs -put’ command to upload the file from ‘/tmp’ directory of the container to HDFS. The directory ‘/mydirectory’ was created in the classroom lab. |
| At last, I verified the file uploaded. I can see the file ‘online\_customer\_transaction.csv’ listed in the output.    When I navigated to ‘http://localhost:50070’, and went to ‘Utilities/mydirectory’, I found there was a file I just uploaded ‘online\_customer\_transaction.csv’.  A screenshot of a computer  Description automatically generated |

Hence, I successfully set up and configured a Hadoop cluster using Docker to ingest the ‘online\_customer\_transaction’ dataset into HDFS. By deploying the Hadoop services through a ‘docker-compose.yml’ file and managing the containers, I ensured proper fault tolerance and data integrity. The dataset was successfully loaded into HDFS, where it will be available for further analysis.

## Task 2: Data storage and Querying

In this task, I used MongoDB as the storage solution for the ‘online\_customer\_transaction’ dataset. MongoDB, a NoSQL database, is designed to handle unstructured or semi-structured data. Firstly, It is well suited for scenarios where flexibility and high operation performance are needed while requiring a schema-less, documented-oriented approach. For example, there is a possibility that the ‘online\_customer\_transaction’ dataset will change the structure or attributes over time. Hence, storing these data in MongoDB is suitable for handling such variability. Secondly, MongoDB provides real-time read and write operations, which is beneficial for applications requiring immediate access to up-to-date data. The dataset ‘online\_customer\_transaction’ might require real-time querying and analyzing for the marketing and product departments. Thirdly, MongoDB supports large volumes of transaction data and ensures high availability and fault tolerance. The dataset in this task is quite big with over 52000 rows. Finally, with MongoDB’s aggregation framework, I can perform complex queries and analytics on transaction operations.

* Data conversion:

|  |
| --- |
| To use MongoDB effectively, I needed to convert the CSV file ‘online\_customer\_transaction’ to JSON format. I performed this conversion using the Pandas library in Python and saved the resulting file on the desktop for further use.  A screenshot of a computer program  Description automatically generated |

* Data import:

|  |
| --- |
| Then I imported the JSON file into MongoDB using the ‘mongoimport’ tool, which efficiently handles large datasets and maintains data integrity during the import.  I confirmed the ‘mongoimport’ path in the terminal.  A screenshot of a computer  Description automatically generated |
| I created a database ‘707\_Assessment\_2’ and a collection ‘PartB’ in MongoDB to store the data. Then I inserted the JSON file into the collection using the ‘mongoimport’ command.  A screenshot of a computer program  Description automatically generated |

|  |
| --- |
| I can confirm that the dataset is successfully inserted into MongoDB.  A screenshot of a computer  Description automatically generated |

* Querying and analysis:

Using a Python environment to interact with MongoDB provides various advantages in terms of ease of use, flexibility, and the ability to leverage plenty of libraries for data analysis.

|  |
| --- |
| I established a connection to the MongoDB server using the connection string, selected the relevant database and collection.  A close-up of a text  Description automatically generated |

I performed queries in the Python environment.

|  |
| --- |
| I defined the product SKU to calculate the total amount spent.  A screenshot of a computer code  Description automatically generated |
| I found the top 5 products by total sales number.  A screenshot of a computer code  Description automatically generated |
| I found the total amount spent by the customer with ID 17850.  A screenshot of a computer code  Description automatically generated |

Choosing MongoDB for this task was driven by its ability to handle the volume and variability of the ‘online\_customer\_transaction’ dataset. Its Flexibility in managing unstructured data, combined with analytical capabilities, makes it an ideal solution for storing and analyzing large-scale transaction data. By leveraging MongoDB, I was able to perform analyses such as finding the total amount spent on a specific product, identifying top-selling products and understanding specific customer spending behaviour.

## Task 3: Real-time data clustering using Apache Spark

In this task, I employed Apache Spark to develop a real-time data clustering system, integrating and processing data from Facebook for further insights.

* Facebook API setup:

The first step was to set up the Facebook API, I created a Facebook App to obtain an access token, which is essential for accessing Facebook’s data. This token serves as a key to authenticate requests made to the Facebook API, enabling me to fetch real-time data from the platform.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* Docker compose setup:

Next, I configured a Spark environment using Docker compose, setting up both master and worker nodes. I created and saved a ‘docker-compose.yml’ file under my chosen directory to define the services.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

To initiate the environment, I ran the ‘docker-compose up’ command in the terminal to start the Spark master and worker nodes.

|  |
| --- |
| A screenshot of a computer screen  Description automatically generated |

|  |
| --- |
| I can confirm that Spark master and worker nodes were running successfully.  A screenshot of a computer  Description automatically generated |

Once the nodes were up and running, I accessed ‘localhost:8081’ in a web browser which is the web UI for the Spark Master node, used to monitor the overall status of the Spark cluster. This web interface displays the status of all Spark worker nodes and lists all running and completed jobs, including status, execution time, and progress.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* Python script for data extraction and streaming:

The final step involved running a Python script to extract and stream data from Facebook API to the Spark cluster. I used a script named ‘facebook\_streaming.py’ provided in the lab test and provided my API token and user ID for this purpose.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated  A screenshot of a computer program  Description automatically generated |

After executing the script via the command line, it continuously fetched and streamed data from Facebook to the Spark cluster in real time. The batch interval is set using the ‘time.sleep(30)’ statement within the ‘fetch\_and\_stream\_data’ function, meaning the statement paused the execution of the loop for 30 seconds before fetching new data from the Facebook API again.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

The script seemed to be working correctly but was not receiving new posts because my user ID did not have any new post during that period.

|  |
| --- |
| A white screen with black text  Description automatically generated |

* Monitoring:

To monitor this process, I accessed ‘localhost:4040’ which is the Spark application web UI, used to monitor and debug specific Spark applications. This interface displayed detailed information about the currently running Spark application ‘facebook\_streaming.py’, including the status, task distribution, and run times.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

In summary, by setting up the Facebook API and configuring a Spark environment through Docker Compose, I established a robust system for streaming real-time social media data. The Python script efficiently streamed data to Spark. The system’s real-time monitoring interfaces ensured effective oversight and management of the data streaming process. This setup has the potential for analyzing and aggregating social media data, offering valuable insights into trends and user behaviour as new data becomes available.

## Task 4: Data streaming with Kafka

In this task, I established a data streaming pipeline using Apache Kafka. Kafka’s features ensures that the streaming pipeline remains reliable and consistent, even in the face of failures or high data volumes, supporting effective real-time data analysis.

* Set up Kafka and Zookeeper with Docker:

Firstly, I created a ‘docker-compose.yml’ file, which outlined the configuration necessary to set up Kafka and Zookeeper so that I can run them in a containerized environment using Docker. This file defines the services and their respective configurations, allowing them to run within Docker containers.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

By running the ‘docker-compose up’ comman, I launched the Kafka and Zookeeper on Docker containers. This set up allowed Kafka to handle incoming and outgoing data streams, while Zookeeper managed the distributed nature of Kafka’s environment.

|  |
| --- |
| A white screen with black text  Description automatically generated |
| A screenshot of a computer program  Description automatically generated |

* Create a Kafka topic:

Secondly, with Kafka running, I created a Kafka topic named ‘707Assessment2’. A Kafka topic is essentially a stream of records, where producers send data and consumers read it. I executed the command to create the topic, specifying its configuration such as number of partitions and replication factor.

|  |
| --- |
|  |

* Run the Kafka producer script ‘kafka\_producer.py’:

Next, I ran the Kafka producer script ‘kafka\_producer.py’. This script was to send messages to the Kafka topic ‘707Assessment2’.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

The script sent a series of messages to the Kafka topic. Each message was automatically assigned to partition 0 and given a unique offset within that partition. The offset started at 0 and incremented with each message, serving as a unique identifier for each message within the partition. These messages were sent sequentially, and each was logged to confirm successful transmission.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

* Run the Kafka consumer to verify the message consumption:

To ensure that messages were correctly sent and stored, I used the Kafka consumer to read back the message from the ‘707Assessment2’ topic. I ran the ‘kafka-console-consumer’ command to consume the messages from the beginning of the topic and displayed them in the console. The output indicated that I used the ‘Kafka-console-consumer’ command to read 10 messages from the ‘707Assessment2’ topic starting from the beginning. The messages are correctly displayed, showing the values ‘0’ to ‘9’ as expected.

|  |
| --- |
| A screen shot of a computer  Description automatically generated |

By completing these steps, I successfully set up a robust data streaming pipeline using Kafka. This pipeline ensures that messages can be reliably produced, stored, and consumed in real time, with fault tolerance and data integrity maintained throughout the process. The use of Kafka topics and partitions ensured that the data was efficiently organized and accessible for further processing or analysis.

# Project Report

## Section 1: Summary of the project

Every aspect of our lives is intertwined with data, and all information holds value with the potential to play a critical role at some point. Therefore, storing information securely and ensuring flexible access to it is crucial. Meanwhile, with the rise of social media, technology has become an integral part of people’s lives. Understanding customer behaviour and patterns on social media can help companies develop more effective strategies, making the acquisition of real-time information significantly important.

This project aimed to develop a comprehensive data pipeline by performing ETL operations and applying data engineering techniques on big data platforms, utilizing various tools, technologies, libraries, and platforms such as MongoDB for data storage, Hadoop for distributed data processing, Spark for large scale data analytics, Kafka for real-timing data streaming, and Python for programming and data manipulation so that to create an efficient and scalable pipeline capable for handling and analyzing complex datasets to drive insights and optimize data-driven decision-making process.

In Part A, I selected a scenario regarding a customer service centre, two datasets ‘customer\_support’ and ‘agent\_info’ were used in this task. The objective is to integrate the customer service centre’s datasets for analysing the agents’ performance, identifying areas for improvement, and optimizing the service quality and efficiency. I performed ETL operations by extracting the ‘customer support’ and ‘agent info’ datasets from the online resource Kaggle, cleaning, transforming, and integrating the datasets into a unified dataset as necessary for further operations using Spark in Python.

In Part B, I downloaded a customer transaction dataset from Kaggle with a large volume of customer interactions that captures their purchasing patterns and preferences providing valuable insights into purchasing behaviour for loading, querying, and analyzing operations for Task 1 and Task 2, aiming to prepare and store the customer transaction data for further operations. I loaded the ‘customer transaction’ data into HDFS, and employed a data storage and querying solution using MongoDB. In terms of Task 3 and Task 4, I implemented a real-time data clustering and streaming system for handling the Facebook social media data integration, aiming to prepare the real-time data clustering system for further study. I used Apache Spark to build a real-time data clustering system to analyse the social media data on Facebook and implemented a data streaming pipeline using Kafka to handle real-time data processing.

Through the tasks in Part A, I completed data storage and integration using the ETL method and analyzed the performance of agents in the service centre. By comparing performance metrics with customer feedback, I gained valuable insights that can help the company develop management strategies more aligned with real situations, such as understanding the impact of talk time on service quality and its effect on household consumption. In Part B, I completed data storage and query, leveraging Hadoop’s HDFS to reduce computational workload and accelerate the process. Additionally, I implemented real-time information integration and obtained Facebook interaction data, which could be beneficial for formulating effective marketing strategies.

The tasks in Part A and Part B significantly enhanced my understanding of the course material. They provided an opportunity to review and apply concepts. Through this process, I consolidated my knowledge and strengthened my data engineering and analysis skills.

## Section 2: Task-Specific Description and Technical Details

Below, I will describe the tasks involved in this project and provide their respective technical details.

* ETL operation and integration:

The objective was to integrate data from multiple sources into a unified dataset. I used Spark for extracting, transforming, and loading data by applying transformations to clean and merge data from different sources into a single dataset suitable for analysis. I used Spark’s ‘printSchema()’ method to examine the dataset’s structure and identified that some time-related data was in string format, requiring conversion to date-time format for accurate processing by using the ‘to\_timestamp()’ function and some missing values needed to be dropped as they were deemed irrelevant for further analysis. I grouped the dataset by ‘Agent\_name’ and aggregated columns in ‘agent\_df’ and performed a left join of the ‘customer\_df’ dataset with the aggregated ‘agent\_df’ on the ‘Agent\_name’ column. This ensured that all rows from ‘customer\_df’ were retained and corresponding rows from ‘aggregated\_agent’ were included where available. Spark was chosen due to its powerful capabilities for handling large-scale data. Its in-memory computing and distributed processing features significantly speed up data processing. Spark’s efficiency and reliability make it well-suited for managing and analyzing large datasets. Instead of a CSV file, the final cleaned data was saved as a Parquet file. Parquet is optimized for performance and storage efficiency, which is ideal for big data processing.

* Data ingestion pipeline with HDFS:

I established an ingestion pipeline for loading customer transaction data into Hadoop HDFS leveraging Docker for deployment and management. To begin, I downloaded and installed Docker, a tool that simplifies the deployment and management of containerized applications. This was particularly useful for managing Hadoop services. With Docker in place, I proceeded to set up the Hadoop cluster. By navigating to the directory containing this ‘docker-compose.yml’ file and executing the appropriate command, I successfully started the Hadoop containers in detached mode. I then confirmed that the containers were running properly by checking the Docker Desktop interface. Once the Hadoop cluster was up and running, I accessed the HDFS web interface at ‘http://localhost:50070’ to check the overall status of the Hadoop cluster. Additionally, I visited the YARN Resource Manager web UI at ‘http://localhost:8088’ to monitor and manage the cluster’s resources and applications. The next step was to upload the local ‘online\_customer\_transaction.csv’ file to HDFS. I placed this file in a directory I created named ‘mydirectory’ within HDFS. After the upload, I navigated back to the HDFS web interface at ‘http://localhost:50070’ to verify that the file had been successfully transferred. By accessing the ‘Utilities/mydirectory’ path, I confirmed the presence of the uploaded file, ensuring that the data ingestion pipeline was successfully established.

* Real-time data clustering with Spark:

I utilized Apache Spark to stream and integrate real-time data from Facebook. First, I set up the Facebook API by creating an app and generating an access token. Next, I configured a Spark environment using Docker Compose, defining both Spark master and worker nodes in a ‘docker-compose.yml’ file. After starting the nodes with the ‘docker-compose up’ command, I monitored the cluster through the Spark Master node’s web UI at ‘localhost:8081’. Finally, I ran a Python script (‘facebook\_streaming.py’) to fetch and stream data from the Facebook API into the Spark cluster. The process was monitored using the Spark application web UI at ‘localhost:4040’, where I could view detailed information about the streaming jobs and ensure the process ran smoothly.

* Real-time data streaming with Kafka:

I established a data streaming pipeline using Kafka. To start, I configured Kafka and Zookeeper by creating a ‘docker-compose.yml’ file and running it with Docker. This started Kafka and Zookeeper on Docker containers. Next, I created a Kafka topic named ‘707Assessment2’ to organize the incoming messages. I then ran the Kafka producer script (kafka\_producer.py), which successfully sent messages to the ‘707Assessment2’ topic. All messages were directed to partition 0, with each message assigned a unique offset ranging from 0 to 9, allowing for precise tracking by consumers. Finally, I verified message consumption by running the Kafka consumer using the Kafka-console-consumer command. The output confirmed that the messages were correctly consumed, displaying values from 0 to 9 as expected.

## Section 3: Ethical Data Analysis Considerations

The datasets utilized in this project, namely, ‘customer\_support.csv’, ‘agent\_info.csv’, and ‘online\_customer\_transaction’, were downloaded from Kaggle, a platform known for providing public datasets. These datasets are publicly available and intended for use in academic and research projects, ensuring that no personal or sensitive information is misused. As these datasets are public, no explicit permissions or consents were required for their use. However, it is essential to note that the data would be used within the scope allowed by the Kaggle platform’s terms of use and not be used for commercial purposes.

For the real-time data clustering, I utilised my personal Facebook API. This involved accessing data through the API subject to Facebook’s data usage policies. I ensured that my use of the API complied with Facebook’s terms and conditions, particularly regarding the collection and handling of user data. Since the Facebook API may involve data from other users, care was taken to comply with Facebook’s data usage policies, ensuring no personal or sensitive information of others was collected or stored without consent. All the data I have collected on Facebook is confidential and used only for the purpose of this academic report.

I strictly adhered to ethical guidelines during the project to maintain privacy and anonymity. Any personal identifiers within the Facebook data were anonymized where applicable. Only the necessary data attributes were used for analysis, and any irrelevant information was either omitted or removed during the cleaning process. All data used was either anonymized or publicly available, ensuring that no identifiable information could be tracked back to any individual. All data was stored and processed securely, with access restricted to authorized personnel only.

## Section 4: Conclusion

The project successfully integrated multiple data sources, employed big data techniques, and set the foundation for future data engineering and analysis improvements.

Throughout this project, several valuable insights were derived:

* By integrating the ‘customer\_support.csv’ and ‘agent\_info.csv’ datasets. I could evaluate agent performance metrics and customer feedback. Aggregated data such as CSAT score provided a clear view of agent effectiveness and customer satisfaction.
* The ‘online\_customer\_transaction’ dataset analysis revealed purchasing patterns and preferences, including popular products and spending patterns which can help in targeted market and inventory management.
* The real-time streaming setup using Spark and Facebook API demonstrated the system’s capability to handle live data clusters effectively. This setup holds the potential for analyzing social media data, offering insights into trends and patterns as new data is continuously processed. This can be crucial for real-time decision-making and immediate insight into interactions and trends.

During the project, I faced several challenges that required careful consideration and problem-solving:

* Environment integration:

Setting up and configuring the Spark and Java environments required a certain level of understanding and practice, which took me some time. Integrating Docker to manage and run containerized applications posed a significant challenge, particularly in configuring the environment correctly. Ensuring all services, such as Kafka, Hadoop, and Spark, were running smoothly in containers required careful attention to Docker’s networking and resource allocation settings. To overcome Docker integration challenges, I reviewed different configurations in the ‘docker-compose.yml’ file and tested various setups that leveraged Docker’s network features, and I ensured that each service was correctly connected and accessible.

* Port identification:

Identifying and configuring the correct ports for various services was another major challenge. With multiple services running simultaneously, conflicts between ports or incorrect configurations could lead to failure in accessing web interfaces and system errors. For example, when I was running Spark Master in Docker, I encountered an error indicating that port 8080 was occupied. After determining that the port was used by a Java application, I decided to change the Spark port to 8081, which resolved the issue. Furthermore, to identify and avoid port conflicts, I used a map-out approach to locate the ports for each service, checking for any conflicts with existing applications on my machine. This approach was effective as it ensured that all services were accessible through the correct ports without interference.

Suggestions for future work:

* Predictive analysis:

In Part A, after integrating the data through ETL, subsequent analysis can include both descriptive and inferential methods. Descriptive analysis can be used to summarize overall customer satisfaction. For instance, calculating averages and frequencies helps gauge customers’ overall satisfaction with the service. On the other hand, inferential analysis can involve building models to predict future trends, allowing for hypothesis testing to explore correlations between various features, and understanding the relationship between satisfaction and sales by examining distributions. Implementing predictive analytics models can provide deeper insights, such as predicting future purchasing trends and identifying potential service bottlenecks.

* Real-time data ingestion and automation:

In Part B, automation can be implemented to better organize the entire process. For example, customer interaction data can be directly obtained from the CRM or company systems and set up to update weekly by uploading it to HDFS, with MongoDB being used to query the data. When interacting with social media, Spark can be employed to cluster with real-time data either once a week or daily and stream it to Kafka for storage. This approach allows people to monitor data changes and assess the impact of social media on customer interaction in a timely manner, which can help the marketing department make quick and effective decisions to improve sales and market share.

Hence, I created an integrated ‘docker-compose.yml’ file that includes the big data analysis tools and engineering setups.

|  |
| --- |
| A screenshot of a computer  Description automatically generated  A screenshot of a computer program  Description automatically generated  A screenshot of a computer code  Description automatically generated |

To sum up, the project successfully achieved its goal by integrating multiple data sources, performing ETL operations, and applying data engineering techniques to big data platforms such as setting up a real-time data flow pipeline. The insights gained are useful for improving customer service and fine-tuning marketing strategies. Even though some challenges were faced, the solutions worked well, and s strong foundation was set for future improvement. Overall, the project showed using big data tools and techniques effectively to build efficient data pipelines and prepare for future upgrades and expansions.

I have provided my GitHub link below for reference:

<https://github.com/YiliaTao0122/707Assessment2>

## Section 5: References

Docker. (n.d.). Docker Manual. Retrieved from <https://docs.docker.com/manuals/>

Kaggle. (n.d.). eCommerce Customer service satisfaction. Retrieved from <https://www.kaggle.com/datasets/ddosad/ecommerce-customer-service-satisfaction>

Kaggle. (n.d.). Online Retail Transaction Data. Retrieved from <https://www.kaggle.com/datasets/thedevastator/online-retail-transaction-data>

Apache Spark. (n.d.). Spark connect. Retrieved from <https://spark.apache.org/spark-connect/>

Python documentation. (n.d.). The Python Standard Library. Retrieved from <https://docs.python.org/3/library/index.html>

Apache Hadoop. (n.d.). Hadoop documentation. Retrieved from <https://apache.github.io/hadoop/>

Apache Kafka. (n.d.). Kafka documentation. Retrieved from <https://kafka.apache.org/documentation/>