Project Documentation :

This section of the documentation outlines the objectives, requirements, and setup configurations for the data analysis project utilizing PySpark, Hadoop, and other data processing and visualization tools.

Project Requirements :

PySpark version: 3.3.0

Pandas Version : 1.2.5

NumPy version: 1.26.4

Matplotlib version : 3.8.4

Seaborn version: 0.13.2

Python version: 3.9.19

Hadoop version: 3.4.0

Dataset Link: https://www.kaggle.com/code/itankar/bank-customer-segmentation

Dataset Description :

The dataset used in this project, sourced from Kaggle, involves detailed banking transactions and customer demographics aimed at enabling customer segmentation analyses. It comprises various attributes including TransactionID, CustomerID, CustomerDOB, CustGender, CustLocation, CustAccountBalance, TransactionDate, TransactionTime, and TransactionAmount (INR), each providing essential insights into customer behaviors and transaction patterns.

System Setup:

Integration of Hadoop and Spark:

1. Installation and Configuration:

- Installed Hadoop version 3.4.0, ensuring compatibility with the current Spark version.
- Configured SSH for localhost to facilitate passwordless SSH access, essential for Hadoop's proper functioning.
- Generated a new key pair and added it to ~/.ssh/authorized_keys.

- Adjusted permissions for ~/.ssh and authorized_keys to enable passwordless SSH access.

2. HDFS Initialization:

- Formatted the Namenode using the command hdfs namenode -format, a crucial step for initializing HDFS for the first time.
- Started HDFS to ensure proper initialization and functionality.

3. Data Upload and Access:

- Created a text file `temp.txt` and uploaded it to HDFS under the path `/test/hadoop_spark_test.txt` using the command `hdfs dfs -put temp.txt /test/hadoop_spark_test.txt`.

Error : Attempted to read the uploaded file using Spark, initially encountering issues due to an incomplete HDFS URI.

4. Spark Data Processing:

- Wanted to test if I could write a simple text file to hdfs and see if I could read it with spark.
- Accessed the uploaded file '**hdfs://localhost:9000/test/hadoop_spark_test.txt'** using Spark's textFile method, resolving the previous URI issue.
- Successfully displayed the contents of the file using the **'show()'** method.

```
scala> val textFile = spark.read.textFile("hdfs://localhost:9000/test/hadoop_spark_test.txt")
textFile: org.apache.spark.sql.Dataset[String] = [value: string]
scala> textFile.show()
+------------------+
|             value|
+------------------+
|Hello, Hadoop and...|
```

Performing Comprehensive Data Analysis with Jupyter Notebook: Integrating Hadoop for Data Storage and Spark for Processing.

Requirements :

Installation of Jupyter Notebook to facilitate the .ipynb file that I have used for comprehensive.

Implementation Details

Data Preprocessing

- Custom Schema Definition: A custom schema is defined to structure the dataset, specifying data types and handling missing or malformed data.
- Data Loading: The dataset is loaded into a DataFrame with a specified schema to ensure consistency in data format, especially for date fields where a legacy time parser policy is applied.

Implemented custom schema:

```
#Custom Schema for hundred thousand records
custom_schema_full = StructType([
    StructField("TransactionID", StringType(), True),
    StructField("CustomerID", StringType(), True),
    StructField("CustomerDOB", DateType(), True),
    StructField("CustGender", StringType(), True),
    StructField("CustLocation", StringType(), True),
    StructField("CustAccountBalance", DoubleType(), True),
    StructField("TransactionDate", DateType(), True),
    StructField("TransactionTime", IntegerType(), True),
    StructField("TransactionAmount (INR)", DoubleType(), True),
])

df_full = spark.read.format("csv") \
    .option("header", "true") \
    .option("dateFormat", "d/M/yy") \
    .schema(custom_schema_full) \
    .load("hdfs://localhost:9000/project/full_data.csv")
```

Data Analysis and Processing:

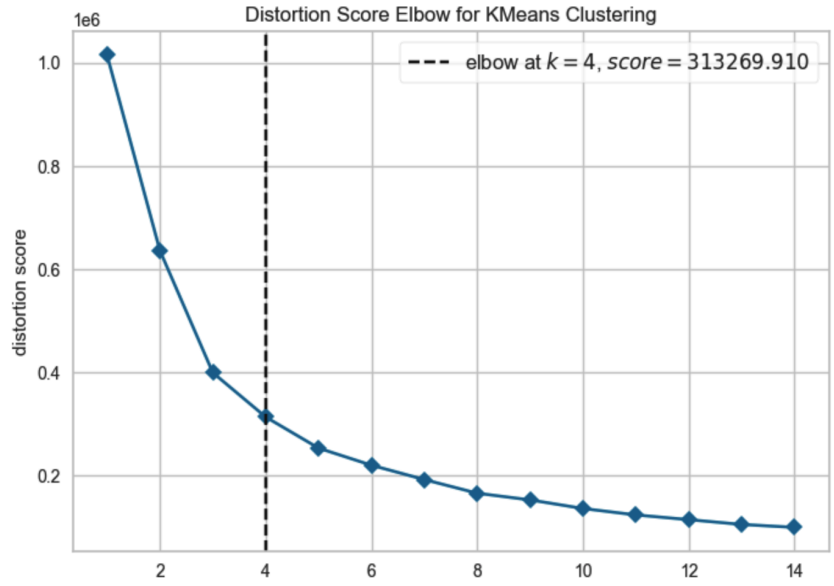
- **Column Renaming:** Columns are renamed for clarity and ease of access during analysis.
- **Data Cleaning:** The data is filtered to remove rows with null or invalid entries, and specific conditions like positive transaction times and amounts are applied.
- **Location Analysis:** Transactions are analyzed based on customer locations to derive insights on transaction distribution across different locations.

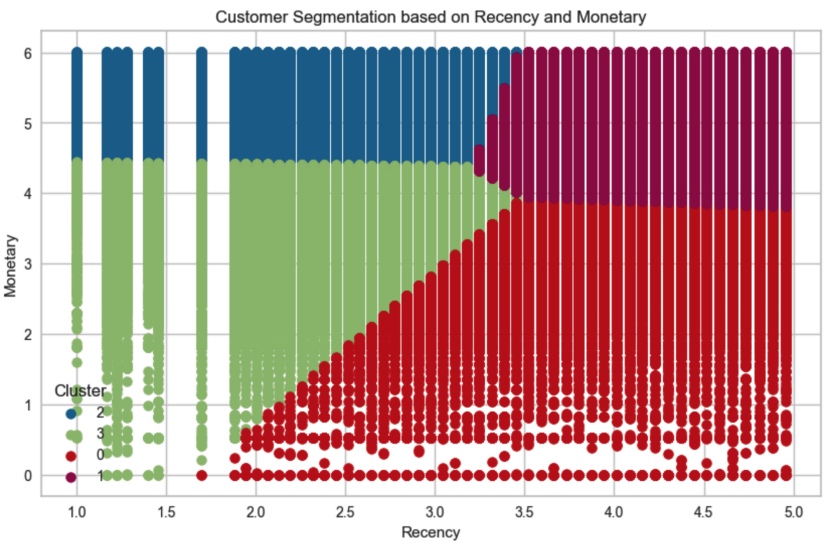
Feature Engineering:

- **Recency, Frequency, and Monetary Analysis:** RFM analysis is conducted to understand customer value based on their transaction history.
- **Transformation:** The RFM features are transformed using Box-Cox and logarithmic transformations to normalize the distributions and improve model performance.

Clustering and Visualization:

- **K-Means Clustering:** The preprocessed data is used to perform clustering to segment customers based on their transaction behavior.
- **Cluster Evaluation:** The silhouette score is calculated to evaluate the effectiveness of the clustering.
- **Visualization:** Scatter plots are generated to visualize the clustering of customers based on recency and monetary values.



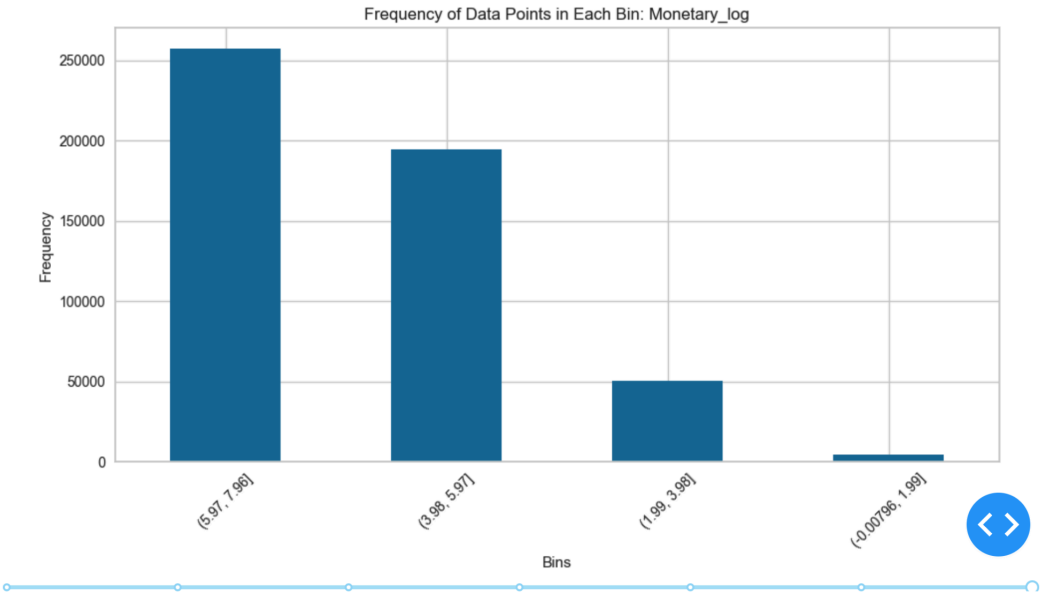


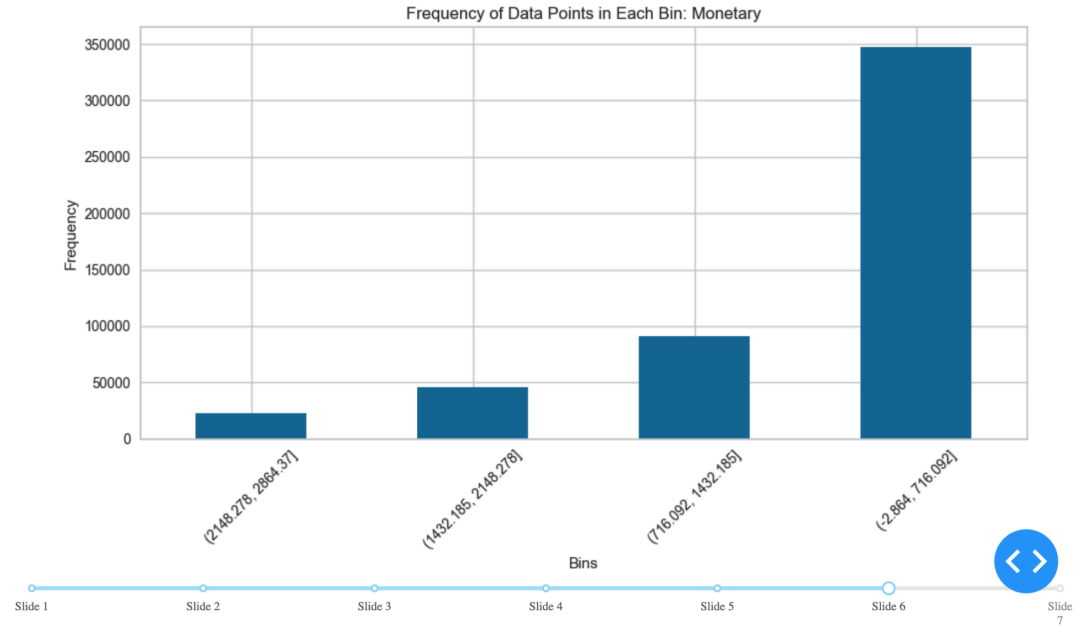
Interactive Visualization Dashboard with Dash:

Implementation Details:

- **Dashboard Framework:** Built with Dash, allowing for real-time interaction and updates.
- **Dynamic Content Loading:** Features a slider-controlled interface that displays different plots based on the selected feature, such as Recency, Frequency, and Monetary values.
- **Image Rendering:** Implements Base64 encoding for efficient image display without external hosting.
- **Deployment:** The dashboard runs on a local development server, facilitating debugging and presentation.

Visualization of Customer Segmentation using Spark





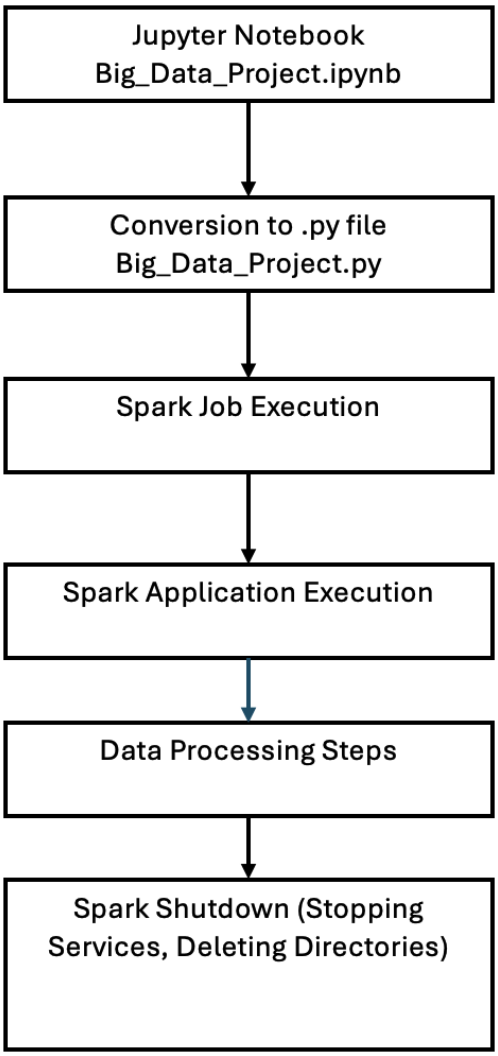
Output and Export

- Data Export: Results from the clustering are exported into CSV files for each cluster for further analysis or reporting.

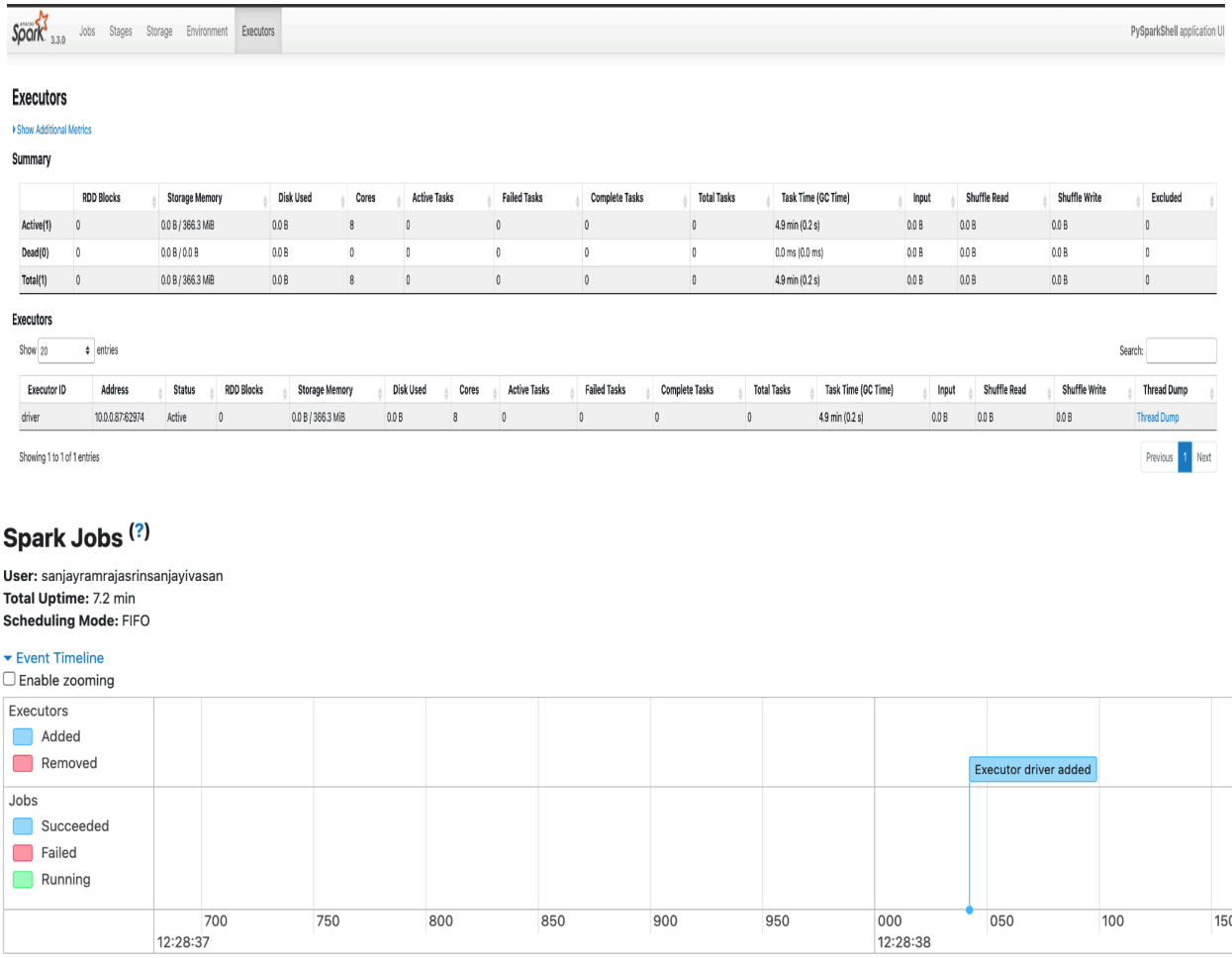
Spark Job Execution :

Command to execute a spark job from hdfs :

(base) user@hostname % spark-submit --master "local[*]" Big_Data_Project.py hdfs://localhost:9000/project/full_data.csv



Monitoring the Spark Job using Spark UI:



Conclusion :

This project showcased the successful integration of Hadoop and Spark for analyzing and segmenting bank customer data. The seamless merging of these technologies enabled robust data handling and customer valuation via RFM analysis. K-Means clustering, validated by the silhouette score, provided insightful segmentation. Future enhancements will focus on incorporating real-time processing and advanced modeling to enhance scalability and adaptability as data volumes and business needs evolve.