Name: Bhargavi Vasudev Jahagirdar
University ID: 2001253654

Big Data Applications, Mini Project

Introduction

In the era of digital transformation, big data analytics plays a pivotal role in uncovering insights from massive datasets. The ability to process and visualize data at scale is a cornerstone of modern data-driven decision-making. This project focuses on designing and implementing a complete big data pipeline leveraging the scalable capabilities of AWS and the distributed processing power of PySpark. The project also addresses critical ethical considerations, such as bias in training data and privacy concerns, ensuring that the outcomes are robust and responsible.

Project Goals and Dataset

Project Goals

The primary objectives of this project are:

1. To develop and implement a scalable big data pipeline using AWS services and a Linux-based PySpark environment.
2. To perform distributed data processing and derive meaningful insights through Spark SQL queries and aggregations.
3. To utilize AWS SageMaker Autopilot for automated machine learning model training and evaluation.
4. To create dynamic dashboards using AWS QuickSight or Power BI for visual representation of data-driven insights.
5. To address ethical challenges, including bias in data and maintaining privacy standards throughout the process.

Dataset Overview

The dataset selected for this project pertains to mobile phone specifications and price categorization. It contains the following key attributes:

-

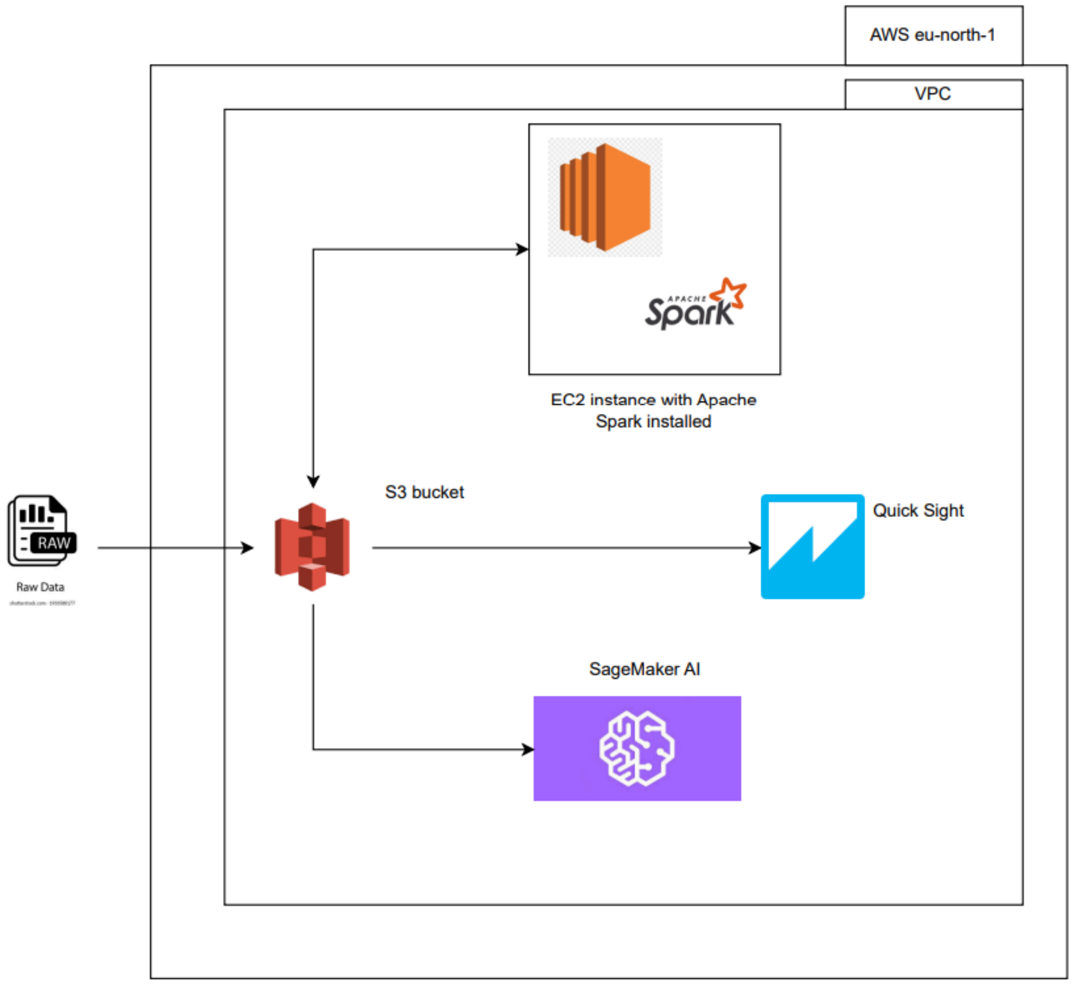
Column Name	Description
battery_power	Battery capacity of the device (mAh).

blue	Binary indicator (0 or 1) if the device supports Bluetooth.
clock_speed	Processor speed of the device (GHz).
dual_sim	Binary indicator (0 or 1) if the device supports dual SIM functionality.
fc	Resolution of the front camera (megapixels).
four_g	Binary indicator (0 or 1) if the device supports 4G connectivity.
int_memory	Internal memory of the device (GB).
m_dep	Mobile depth (cm).
mobile_wt	Weight of the mobile phone (grams).
n_cores	Number of cores in the processor.
pc	Resolution of the primary camera (megapixels).
px_height	Pixel resolution height of the screen.
px_width	Pixel resolution width of the screen.
ram	Random Access Memory (RAM) capacity (MB).
sc_h	Screen height (cm).
sc_w	Screen width (cm).
talk_time	Maximum talk time on a single charge (hours).
three_g	Binary indicator (0 or 1) if the device supports 3G connectivity.
touch_screen	Binary indicator (0 or 1) if the device has a touch screen.
wifi	Binary indicator (0 or 1) if the device supports WiFi.
price_range	Categorical variable indicating price range (0 = low cost, 1 = medium cost, etc.).

The dataset is large and diverse, providing ample opportunity for cleaning, transformation, aggregation, and analysis to derive meaningful insights.

Methodology

Architecture Diagram:



1. Raw Data

- **Source**: This is the starting point where raw data is ingested into the pipeline.
- **Purpose**: The raw data represents unprocessed input, potentially from various sources like IoT sensors, transactional systems, or external data providers.
- **Flow**: The raw data is uploaded to an **AWS S3 bucket** for centralized storage.

2. S3 Bucket

- **Purpose**: AWS S3 serves as a scalable storage solution to hold both raw and processed data.
- **Functions**:
    - Stores raw data for ingestion into processing systems.

    - Saves processed and aggregated data for downstream tasks like visualization and machine learning.
- **Flow**:
    - **Input**: Raw data flows into the S3 bucket from the data source.
    - **Output**: Processed data flows back into the bucket after being transformed by the EC2 instance and Apache Spark.

3. EC2 Instance with Apache Spark Installed

- **Purpose**: Acts as the computational engine for processing and transforming the data.
- **Features**:
    - Apache Spark installed on the EC2 instance is used for distributed data processing.
    - PySpark scripts clean, transform, and aggregate the raw data from S3.
- **Flow**:
    - **Input**: Raw data is pulled from the S3 bucket into the EC2 instance.
    - **Output**: Transformed and aggregated data is written back to the S3 bucket.

4. SageMaker AI

- **Purpose**: AWS SageMaker automates the machine learning workflow, including model training and evaluation.
- **Features**:
    - Uses processed data stored in the S3 bucket as input for training models.
    - Automates feature selection, model selection, and hyperparameter tuning using SageMaker Autopilot.
    - Outputs performance metrics and trained models.
- **Flow**:
    - **Input**: Processed data is retrieved from the S3 bucket.
    - **Output**: Model results and insights are available for further analysis or deployment.

5. AWS QuickSight

- **Purpose**: Provides interactive dashboards and visualizations for the processed data.
- **Features**:
    - Fetches processed data from the S3 bucket to create visual reports.
    - Dashboards can include charts for trends, distributions, and patterns derived from the data.
    - Useful for stakeholders to derive actionable insights.
- **Flow**:

    - **Input**: Processed data from the S3 bucket is connected to QuickSight for visualization.
    - **Output**: Visual dashboards are presented to end users and decision-makers.

6. Region and VPC Context (AWS eu-north-1 and VPC)

- **AWS Region (eu-north-1)**: Specifies that the resources (S3, EC2, SageMaker, and QuickSight) are hosted in the AWS Europe North region.
- **VPC (Virtual Private Cloud)**: Ensures that all AWS services used in the pipeline operate within a secure, isolated network environment.

Flow Summary

1. Raw data is ingested into the **S3 bucket**.
2. Data is processed on an **EC2 instance with Apache Spark** and written back to **S3**.
3. Processed data is used by **SageMaker AI** for machine learning and by **QuickSight** for visualization.
4. End-users and stakeholders interact with the system through dashboards in QuickSight.

Workflow Breakdown

1. Dataset Selection

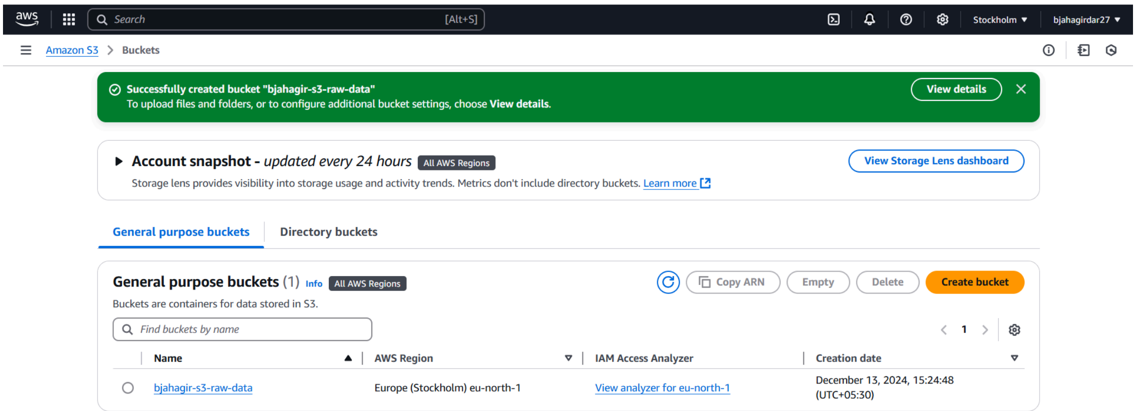
- For this project, I have chosen the Mobile Price Prediction dataset from Kaggle.
- Here is the data dictionary for the dataset:

Column Name	Description
battery_power	Battery capacity of the device (mAh).
blue	Binary indicator (0 or 1) if the device supports Bluetooth.
clock_speed	Processor speed of the device (GHz).
dual_sim	Binary indicator (0 or 1) if the device supports dual SIM functionality.
fc	Resolution of the front camera (megapixels).
four_g	Binary indicator (0 or 1) if the device supports 4G connectivity.
int_memory	Internal memory of the device (GB).
m_dep	Mobile depth (cm).
mobile_wt	Weight of the mobile phone (grams).
n_cores	Number of cores in the processor.
pc	Resolution of the primary camera (megapixels).
px_height	Pixel resolution height of the screen.
px_width	Pixel resolution width of the screen.
ram	Random Access Memory (RAM) capacity (MB).
sc_h	Screen height (cm).
sc_w	Screen width (cm).
talk_time	Maximum talk time on a single charge (hours).
three_g	Binary indicator (0 or 1) if the device supports 3G connectivity.
touch_screen	Binary indicator (0 or 1) if the device has a touch screen.
wifi	Binary indicator (0 or 1) if the device supports WiFi.
price_range	Categorical variable indicating price range (0 = low cost, 1 = medium cost, etc.).

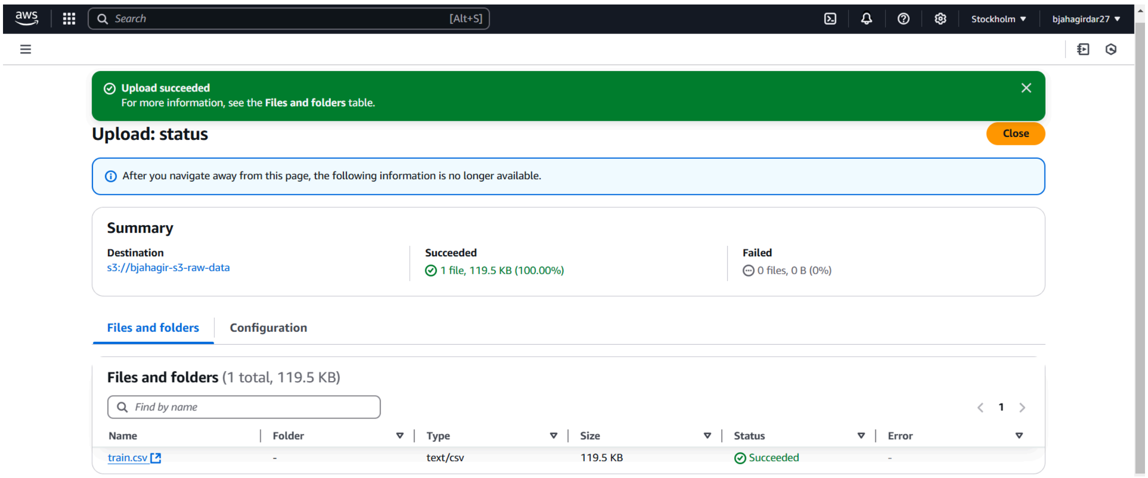
2. Environment Setup

1. AWS S3 for Data Storage

- Create and configure an S3 bucket for raw and processed data.



- Upload raw data to S3.



2. Linux Environment with PySpark:

- Set up a Linux-based environment (local or AWS EC2).
- Install PySpark.
- Configure AWS CLI for S3 interaction.

Commands:

1. Update System and Install Essentials

sudo yum update -y

sudo yum install wget tar python3 python3-pip -y

2. Install Java

wget https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-jdk.tar.gz

tar -xvf amazon-corretto-8-x64-linux-jdk.tar.gz

sudo mv amazon-corretto-8.* /usr/local/java

echo "export JAVA_HOME=/usr/local/java" >> ~/.bashrc

echo "export PATH=\$JAVA_HOME/bin:\$PATH" >> ~/.bashrc

source ~/.bashrc

java -version

3. Install Apache Spark

wget https://archive.apache.org/dist/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz

tar xvf spark-3.0.1-bin-hadoop2.7.tgz

sudo mv spark-3.0.1-bin-hadoop2.7 /opt/spark-3.0.1

sudo ln -s /opt/spark-3.0.1 /opt/spark

4. Configure Spark Environment Variables

echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc

echo "export PATH=\$SPARK_HOME/bin:\$PATH" >> ~/.bashrc

echo "export PYSPARK_PYTHON=python3" >> ~/.bashrc

echo "export PYSPARK_DRIVER_PYTHON=jupyter" >> ~/.bashrc

echo "export PYSPARK_DRIVER_PYTHON_OPTS='notebook --ip 0.0.0.0
--port=8888'" >> ~/.bashrc

source ~/.bashrc

5. Install Required Python Libraries

pip3 install py4j findspark

3. Data Pipeline Tasks

Task 1: Data Ingestion from S3:

- Pull raw data using AWS CLI or PySpark's S3 support.
- Validate ingestion by inspecting the dataset.

Codes for the same are added to the zip file

Data Reading and Overview:

```
[ec2-user@ip-172-31-31-147 ~]$ python3 data_overview.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/14 07:20:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Reading data from s3a://bjahagir-s3-raw-data/train.csv...
24/12/14 07:20:10 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties
Data successfully loaded into Spark DataFrame:
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+----+----+----+---------+-------+------------+----+------
-----+
|battery_power|blue|clock_speed|dual_sim| fc|four_g|int_memory|m_dep|mobile_wt|n_cores| pc|px_height|px_width| ram|sc_h|sc_w|talk_time|three_g|touch_screen|wifi|price_
range|
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+----+----+----+---------+-------+------------+----+------
-----+
|          842|   0|        2.2|       0|  1|     0|         7|  0.6|      188|      2|  2|       20|     756|2549|   9|   7|       19|      0|           0|   1|
    1|
|         1021|   1|        0.5|       1|  0|     1|        53|  0.7|      136|      3|  6|      905|    1988|2631|  17|   3|        7|      1|           1|   0|
    2|
|          563|   1|        0.5|       1|  2|     1|        41|  0.9|      145|      5|  6|     1263|    1716|2603|  11|   2|        9|      1|           1|   0|
    2|
|          615|   1|        2.5|       0|  0|     0|        10|  0.8|      131|      6|  9|     1216|    1786|2769|  16|   8|       11|      1|           0|   0|
    2|
|         1821|   1|        1.2|       0| 13|     1|        44|  0.6|      141|      2| 14|     1208|    1212|1411|   8|   2|       15|      1|           1|   0|
    1|
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+----+----+----+---------+-------+------------+----+------
-----+
```

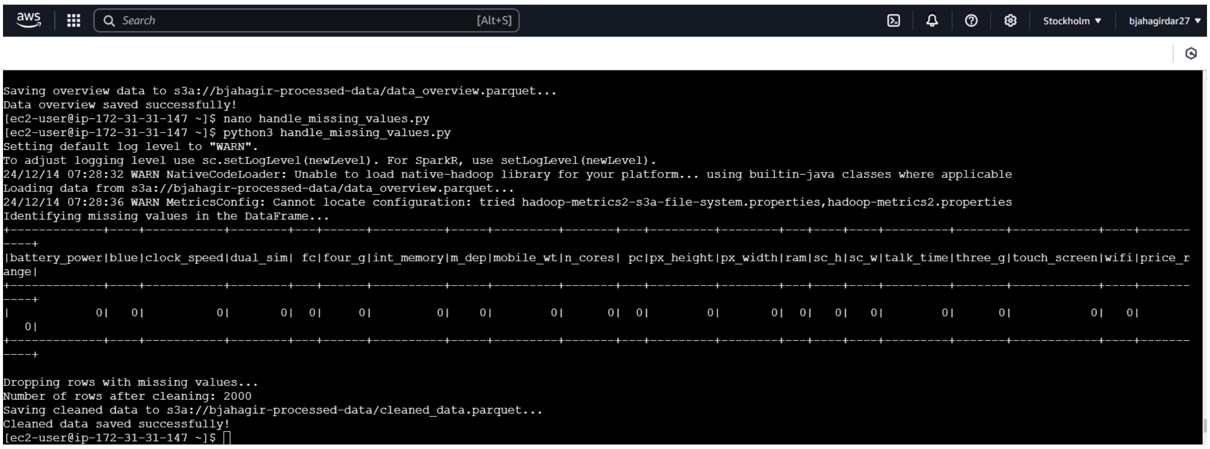
aws Search [Alt+S] Stockholm bjahagirdar27

```
Schema of the DataFrame:
root
 |-- battery_power: integer (nullable = true)
 |-- blue: integer (nullable = true)
 |-- clock_speed: double (nullable = true)
 |-- dual_sim: integer (nullable = true)
 |-- fc: integer (nullable = true)
 |-- four_g: integer (nullable = true)
 |-- int_memory: integer (nullable = true)
 |-- m_dep: double (nullable = true)
 |-- mobile_wt: integer (nullable = true)
 |-- n_cores: integer (nullable = true)
 |-- pc: integer (nullable = true)
 |-- px_height: integer (nullable = true)
 |-- px_width: integer (nullable = true)
 |-- ram: integer (nullable = true)
 |-- sc_h: integer (nullable = true)
 |-- sc_w: integer (nullable = true)
 |-- talk_time: integer (nullable = true)
 |-- three_g: integer (nullable = true)
 |-- touch_screen: integer (nullable = true)
 |-- wifi: integer (nullable = true)
 |-- price_range: integer (nullable = true)

Saving overview data to s3a://bjahagir-processed-data/data_overview.parquet...
Data overview saved successfully!
[ec2-user@ip-172-31-31-147 ~]$
```

Task 2: Data Processing with PySpark:

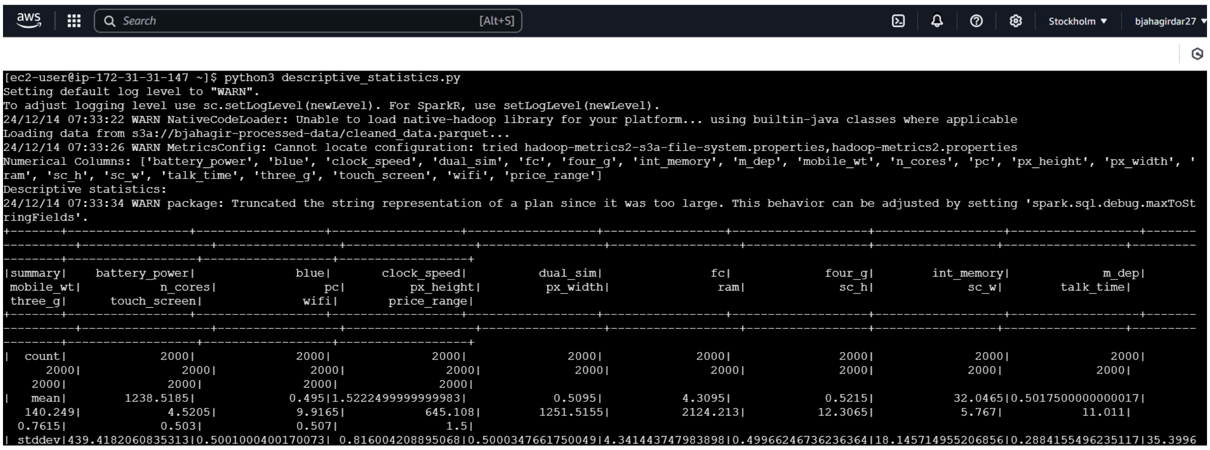
- Clean, transform, and aggregate data:

  Checking for missing values:

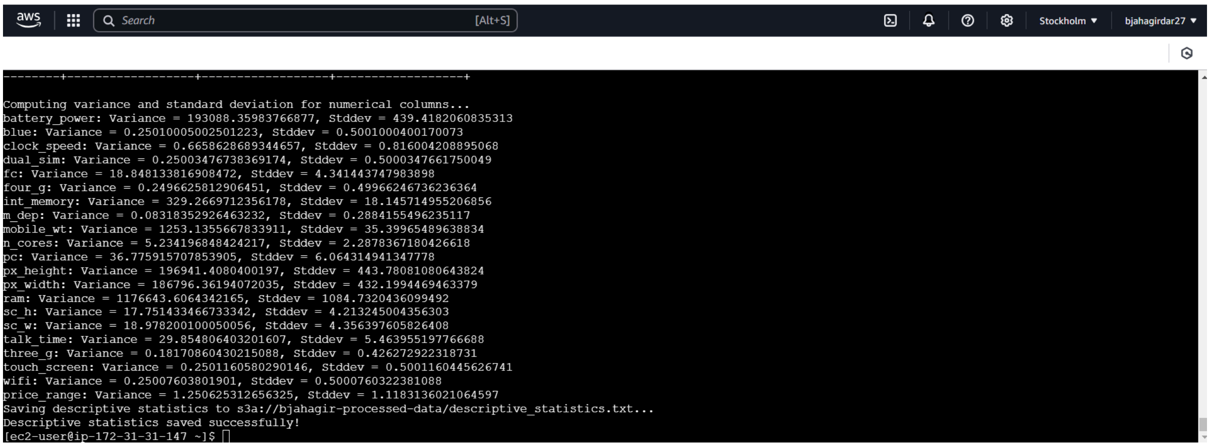


```
Saving overview data to s3a://bjahagir-processed-data/data_overview.parquet...
Data overview saved successfully!
[ec2-user@ip-172-31-31-147 ~]$ nano handle_missing_values.py
[ec2-user@ip-172-31-31-147 ~]$ python3 handle_missing_values.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/14 07:28:32 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Loading data from s3a://bjahagir-processed-data/data_overview.parquet...
24/12/14 07:28:36 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties
Identifying missing values in the DataFrame...
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+---+----+----+---------+-------+------------+----+-------
----+
|battery_power|blue|clock_speed|dual_sim| fc|four_g|int_memory|m_dep|mobile_wt|n_cores| pc|px_height|px_width|ram|sc_h|sc_w|talk_time|three_g|touch_screen|wifi|price_r
ange|
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+---+----+----+---------+-------+------------+----+-------
----+
|            0|   0|          0|       0|  0|     0|         0|    0|        0|      0|  0|        0|       0|  0|   0|   0|        0|      0|           0|   0|
   0|
+-------------+----+-----------+--------+---+------+----------+-----+---------+-------+---+---------+--------+---+----+----+---------+-------+------------+----+-------
----+

Dropping rows with missing values...
Number of rows after cleaning: 2000
Saving cleaned data to s3a://bjahagir-processed-data/cleaned_data.parquet...
Cleaned data saved successfully!
[ec2-user@ip-172-31-31-147 ~]$
```

  Descriptive Statistics:



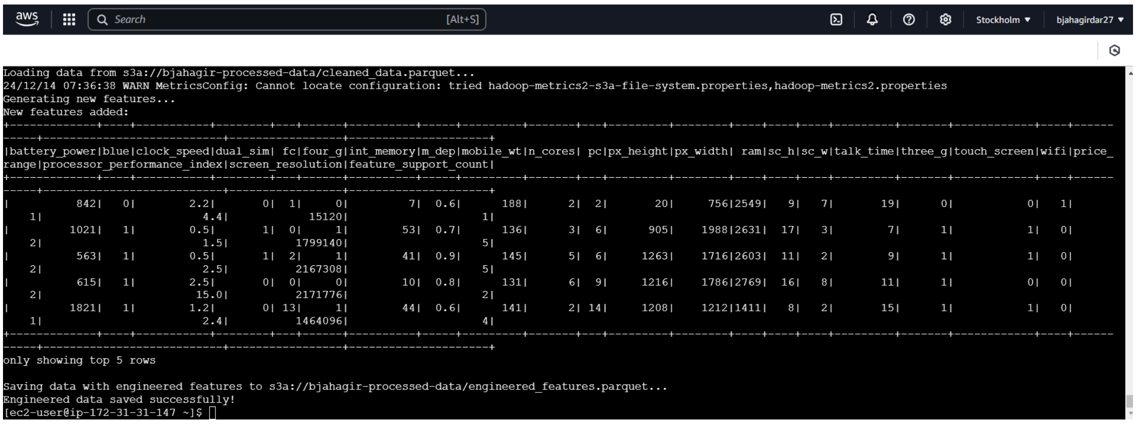
```
[ec2-user@ip-172-31-31-147 ~]$ python3 descriptive_statistics.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/14 07:33:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Loading data from s3a://bjahagir-processed-data/cleaned_data.parquet...
24/12/14 07:33:26 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties
Numerical Columns: ['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height', 'px_width', '
ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'price_range']
Descriptive statistics:
24/12/14 07:33:34 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToSt
ringFields'.
|summary|    battery_power|              blue|       clock_speed|          dual_sim|                fc|            four_g|        int_memory|             m_dep|
 mobile_wt|          n_cores|                pc|         px_height|          px_width|               ram|              sc_h|              sc_w|         talk_time|
 three_g|     touch_screen|              wifi|       price_range|
|  count|             2000|              2000|              2000|              2000|              2000|              2000|              2000|              2000|
    2000|             2000|              2000|              2000|              2000|              2000|              2000|              2000|              2000|
   2000|             2000|              2000|              2000|
|   mean|        1238.5185|             0.495|1.5222499999999983|            0.5095|            4.3095|            0.5215|           32.0465|0.5017500000000017|
  140.249|           4.5205|            9.9165|           645.108|         1251.5155|          2124.213|           12.3065|             5.767|            11.011|
  0.7615|            0.503|             0.507|               1.5|
| stddev|439.4182060835313|0.5001000400170073| 0.816004208895068|0.5000347661750049|4.341443747983898|0.49966246736236364|18.145714955206856|0.2884155496235117|35.3996
```



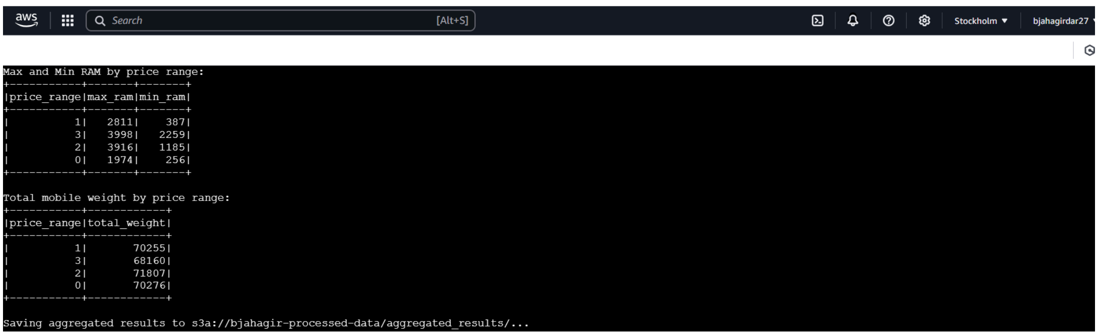
```
--------+------------------+------------------+------------------+

Computing variance and standard deviation for numerical columns...
battery_power: Variance = 193088.35983766877, Stddev = 439.4182060835313
blue: Variance = 0.25010005002501223, Stddev = 0.5001000400170073
clock_speed: Variance = 0.6658628689344657, Stddev = 0.816004208895068
dual_sim: Variance = 0.25003476738369174, Stddev = 0.5000347661750049
fc: Variance = 18.848133816908472, Stddev = 4.341443747983898
four_g: Variance = 0.2496625812906451, Stddev = 0.49966246736236364
int_memory: Variance = 329.2669712356178, Stddev = 18.145714955206856
m_dep: Variance = 0.08318352926463232, Stddev = 0.2884155496235117
mobile_wt: Variance = 1253.1355667833911, Stddev = 35.39965489638834
n_cores: Variance = 5.234196848424217, Stddev = 2.2878367180426618
pc: Variance = 36.775915707853905, Stddev = 6.064314941347778
px_height: Variance = 196941.4080400197, Stddev = 443.78081080643824
px_width: Variance = 186796.36194072035, Stddev = 432.1994469463379
ram: Variance = 1176643.6064342165, Stddev = 1084.7320436099492
sc_h: Variance = 17.751433466733342, Stddev = 4.213245004356303
sc_w: Variance = 18.978200100050056, Stddev = 4.356397605826408
talk_time: Variance = 29.854806403201607, Stddev = 5.463955197766688
three_g: Variance = 0.18170860430215088, Stddev = 0.426272922318731
touch_screen: Variance = 0.2501160580290146, Stddev = 0.5001160445626741
wifi: Variance = 0.25007603801901, Stddev = 0.5000760322381088
price_range: Variance = 1.250625312656325, Stddev = 1.1183136021064597
Saving descriptive statistics to s3a://bjahagir-processed-data/descriptive_statistics.txt...
Descriptive statistics saved successfully!
[ec2-user@ip-172-31-31-147 ~]$
```

- **Transformation**: Add at least 2 new columns (e.g., Year, Month).

  Feature Engineering:

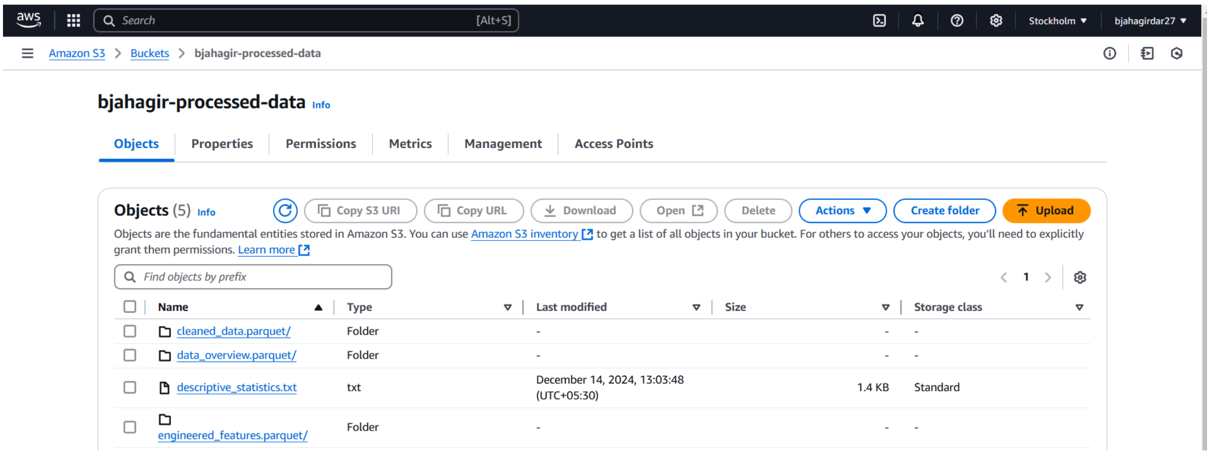


- **Aggregation**:



Task 3: Store Processed Data Back to S3:

- Save processed data as CSV/Parquet and upload it to S3.



Task 4: Data Analysis Using Spark SQL:

- Write 5 queries for insights:

Queries:

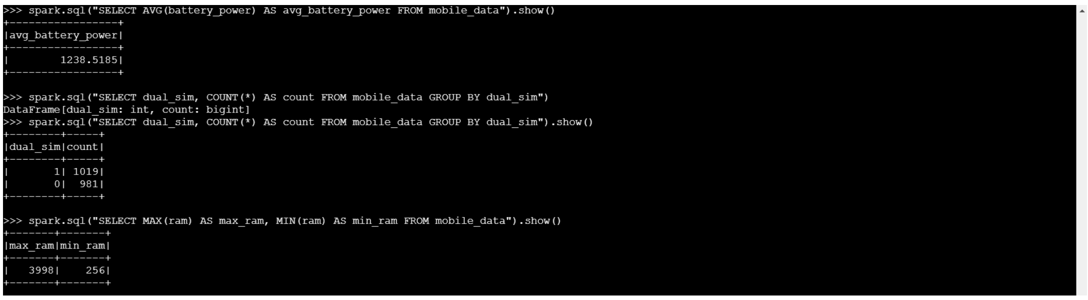
spark.sql("SELECT AVG(battery_power) AS avg_battery_power FROM mobile_data")

spark.sql("SELECT dual_sim, COUNT(*) AS count FROM mobile_data GROUP BY dual_sim")

spark.sql("SELECT MAX(ram) AS max_ram, MIN(ram) AS min_ram FROM mobile_data")

spark.sql("SELECT price_range, AVG(mobile_wt) AS avg_weight FROM mobile_data GROUP BY price_range")

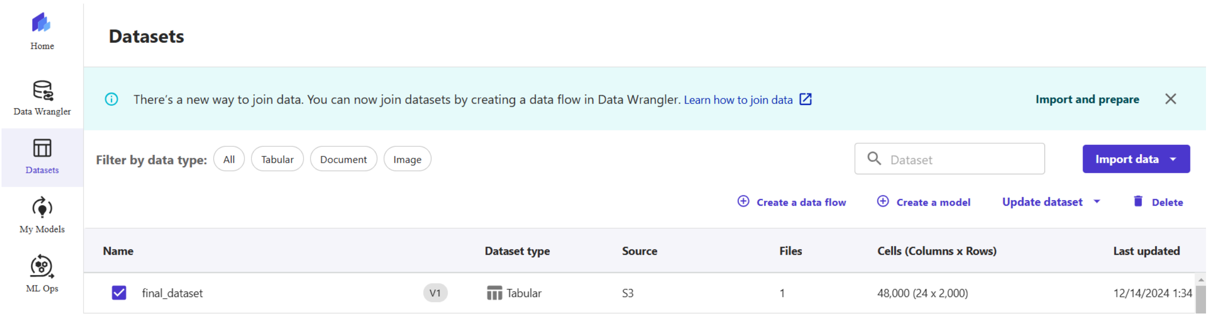
spark.sql("SELECT n_cores, AVG(clock_speed) AS avg_clock_speed FROM mobile_data GROUP BY n_cores")



```
>>> spark.sql("SELECT price_range, AVG(mobile_wt) AS avg_weight FROM mobile_data GROUP BY price_range").show()
+-----------+----------+
|price_range|avg_weight|
+-----------+----------+
|          1|    140.51|
|          3|    136.32|
|          2|   143.614|
|          0|   140.552|
+-----------+----------+

>>> spark.sql("SELECT n_cores, AVG(clock_speed) AS avg_clock_speed FROM mobile_data GROUP BY n_cores")
DataFrame[n_cores: int, avg_clock_speed: double]
>>> spark.sql("SELECT n_cores, AVG(clock_speed) AS avg_clock_speed FROM mobile_data GROUP BY n_cores")
DataFrame[n_cores: int, avg_clock_speed: double]
>>> spark.sql("SELECT n_cores, AVG(clock_speed) AS avg_clock_speed FROM mobile_data GROUP BY n_cores").show()
+-------+-----------------+
|n_cores|  avg_clock_speed|
+-------+-----------------+
|      1| 1.586776859504133|
|      6|1.5321739130434786|
|      3|1.5609756097560983|
|      5|1.4227642276422767|
|      4|1.5452554744525546|
|      8|1.5132812499999988|
|      7|1.5610038610038606|
|      2|1.4534412955465588|
```

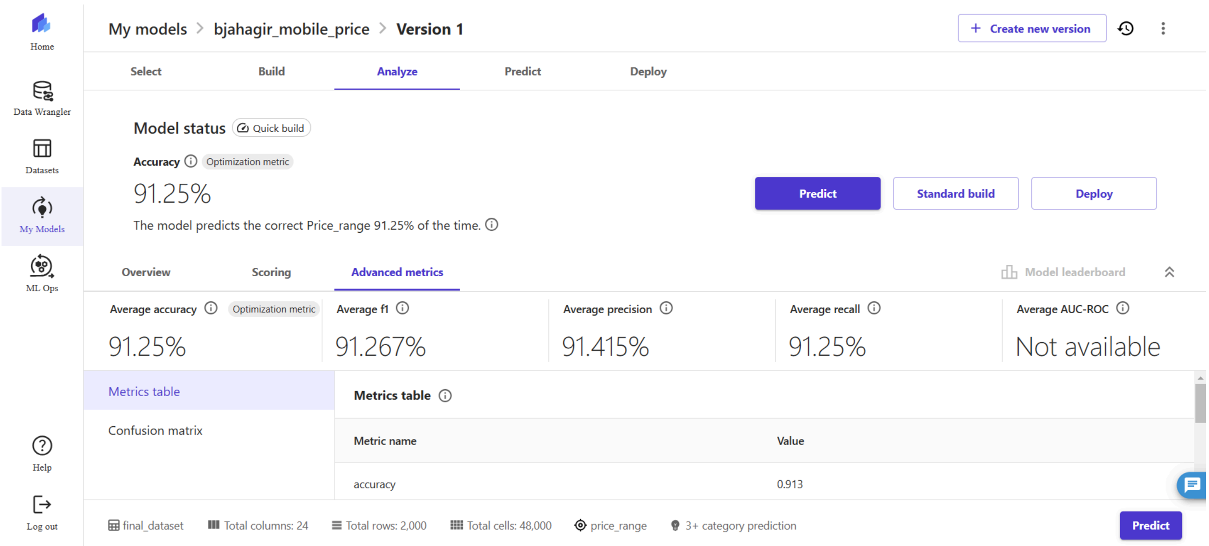
Task 5: Machine Learning with AWS SageMaker Autopilot:

- Load processed data into SageMaker Autopilot.



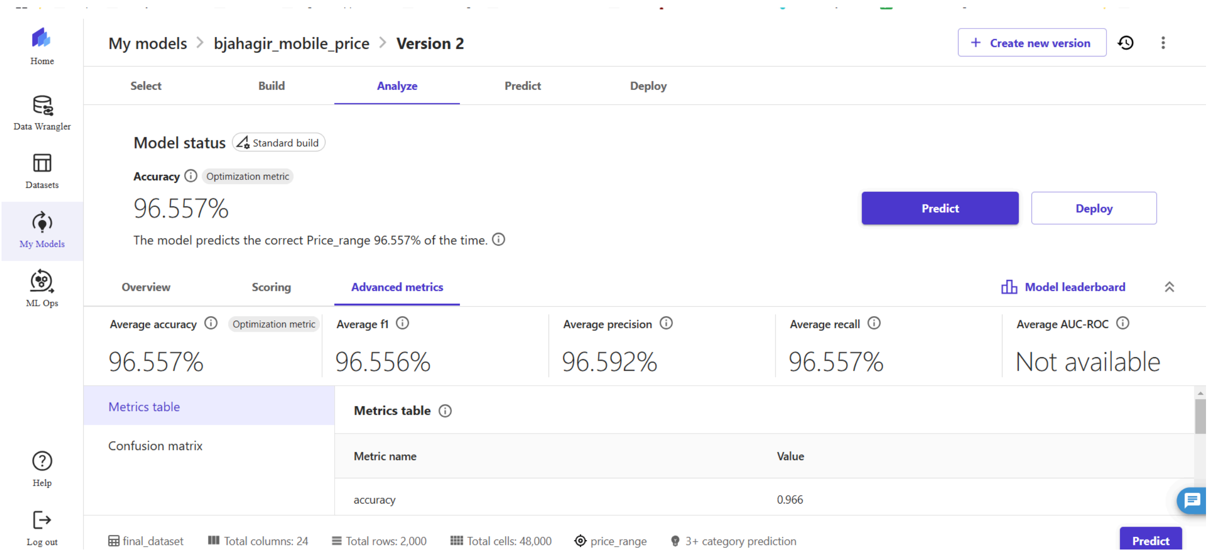
- Train and evaluate ML models.

  Version 1:

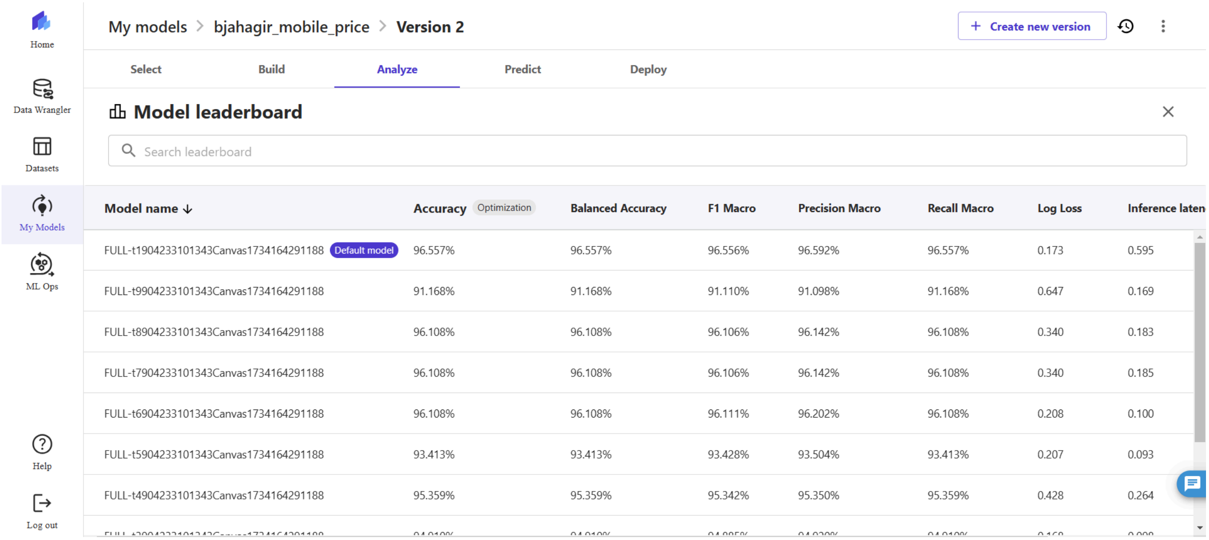


Version 2:

- Ensemble techniques used.
- The algorithms used were: XGBoost, Linear Models, LightGBM, CatBoost, Random Forest, Extra Trees, Neural Networks Built using PyTorch, Neural Networks Built using Fast.ai
- The data split was 70% and 30% for train and test



Model Leadership Board:



Version 3:

- Ensemble techniques used.
- The algorithms used were: XGBoost, Linear Models, LightGBM, CatBoost, Random Forest, Extra Trees, Neural Networks Built using PyTorch, Neural Networks Built using Fast.ai
- The data split was 80% and 20% for train and test

Analysis of the model performance:

Metric	SageMaker Default Model	Custom Ensemble Model	Custom Ensemble Model 2
Accuracy	91%	96%	97.25%
Precision	Higher variance	More consistent	Consistent
Generalization	Moderate	Moderate	Superior

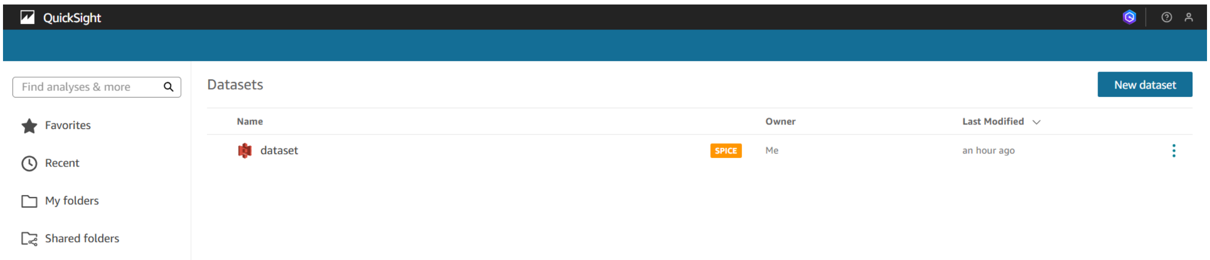
The ensemble model trained with a custom train-test split outperformed the default SageMaker Autopilot model, achieving an accuracy of 96% compared to 91%. The improvement highlights the value of ensemble techniques and proper data partitioning.

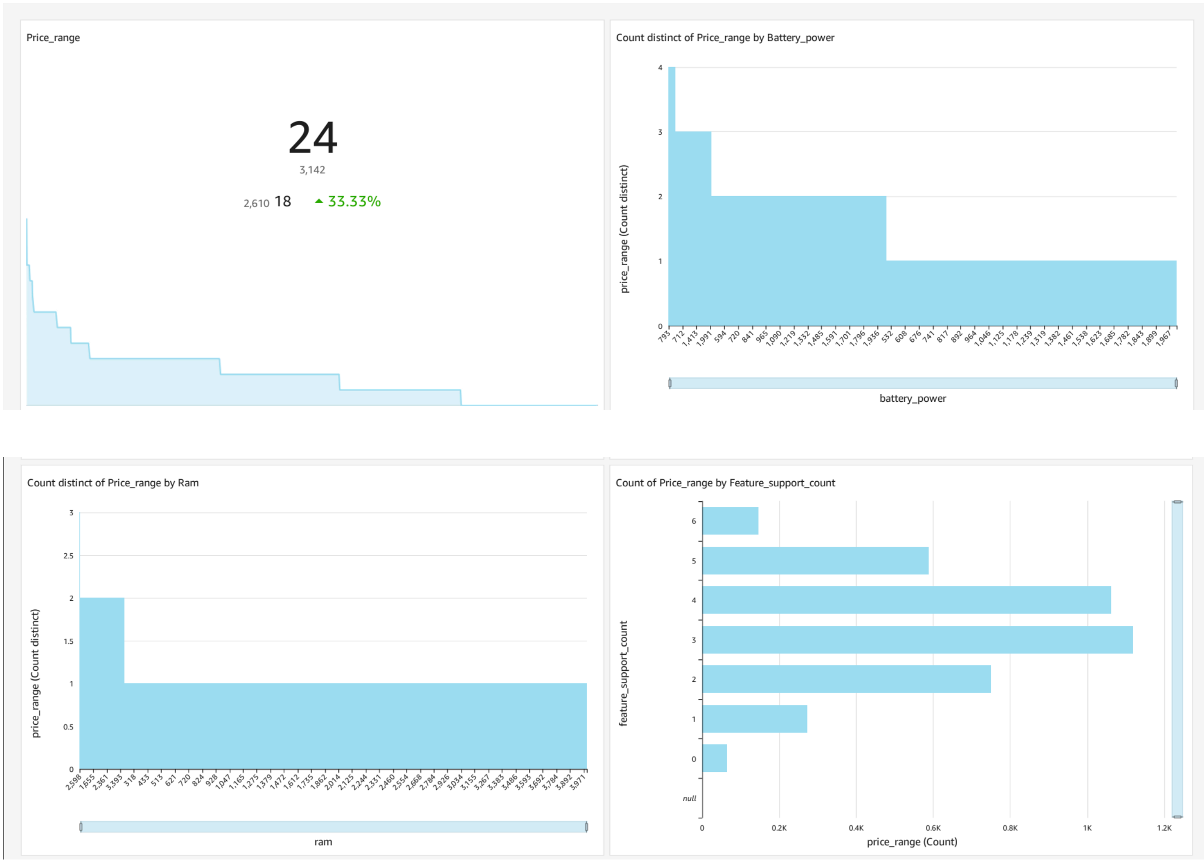
- Address ethical issues like data bias and privacy

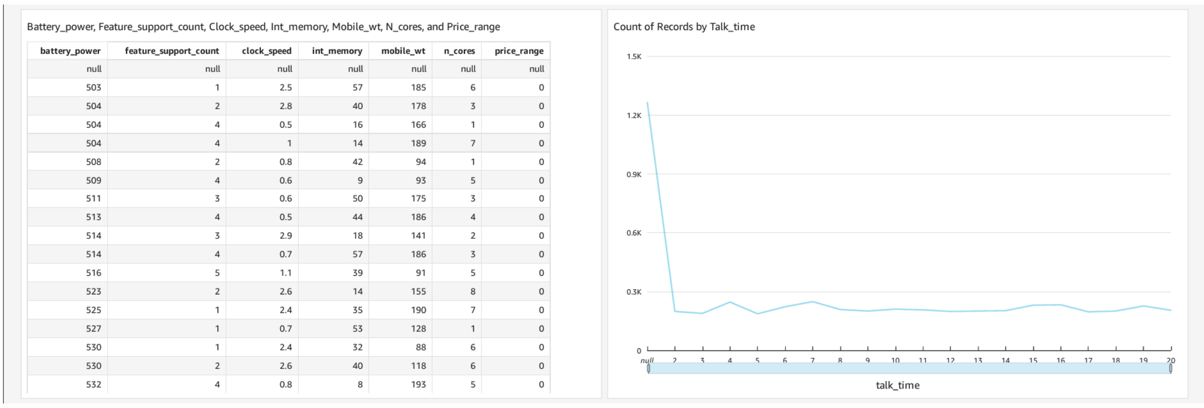
  Ethical concerns were addressed by identifying and mitigating data bias and ensuring privacy compliance. Conducted exploratory data analysis (EDA) to identify any imbalances in target column and all the associate features for ensure no data bias was created. This ensures that the solution is not only technically robust but also ethically sound. Security and Privacy was managed using the user roles provided by IAM on AWS.

4. Visualization

- **Tools**: AWS QuickSight (preferred) or Power BI.
- Create a dashboard with at least 4 visualizations connected to S3 data.







Battery_power, Feature_support_count, Clock_speed, Int_memory, Mobile_wt, N_cores, and Price_range

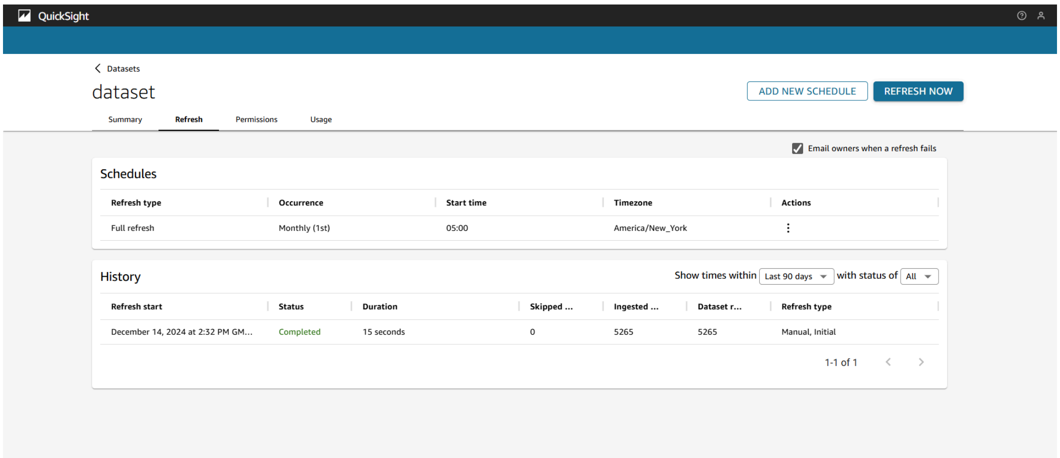
battery_power	feature_support_count	clock_speed	int_memory	mobile_wt	n_cores	price_range
null	null	null	null	null	null	null
503	1	2.5	57	185	6	0
504	2	2.8	40	178	3	0
504	4	0.5	16	166	1	0
504	4	1	14	189	7	0
508	2	0.8	42	94	1	0
509	4	0.6	9	93	5	0
511	3	0.6	50	175	3	0
513	4	0.5	44	186	4	0
514	3	2.9	18	141	2	0
514	4	0.7	57	186	3	0
516	5	1.1	39	91	5	0
523	2	2.6	14	155	8	0
525	1	2.4	35	190	7	0
527	1	0.7	53	128	1	0
530	1	2.4	32	88	6	0
530	2	2.6	40	118	6	0
532	4	0.8	8	193	5	0

5. Bonus Task: Automation of the Pipeline

- **Automation Script (1 Point):**
  - Automate ingestion, processing, and storage steps.

    Added the script to the zip folder

- **Dashboard Automation (1 Point):**
  - Cofigure QuickSight/Power BI for periodic data refresh.



Results

1. The data pipeline successfully processed the dataset, enabling distributed transformations and aggregations in PySpark.
2. Spark SQL queries provided actionable insights, such as the correlation between RAM and price range and the prevalence of 4G devices in higher price categories.
3. Machine learning models generated using SageMaker Autopilot accurately predicted the price range.
4. Dynamic dashboards were created to visualize key metrics and trends, providing a comprehensive view of the dataset's insights.

Conclusion

This project demonstrated the implementation of a complete big data pipeline, leveraging AWS and PySpark to handle large-scale datasets efficiently. The use of AWS SageMaker and QuickSight enabled advanced analysis and visualization capabilities. Ethical considerations, such as bias in data and privacy concerns, were addressed throughout the process. Overall, this project highlights the potential of cloud-based distributed systems in big data analytics and their applicability in solving real-world problems.

Reference:

[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

[2] https://docs.aws.amazon.com/quicksight/latest/user/create-a-data-set-s3.html

[3] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html