Big Data Analytics and Visualization Using AWS and PySpark

ropavu | Rohith Pavuluru

Introduction:

The used car market has witnessed significant growth in recent years, driven by increasing demand for affordable transportation options and the availability of diverse car models. Understanding the factors that influence the pricing and demand for used cars is essential for buyers, sellers, and industry stakeholders. In this project, I aim to analyze a dataset of used car listings to derive actionable insights, such as trends in car pricing, the influence of specific features on value, and the general state of the used car market.

Project Goal

My primary objective in this project is to explore and analyze the given dataset to:

- Identify key factors affecting the price of used cars.
- Provide insights into market trends, such as popular brands, models, or fuel types.
- Develop predictive models to estimate the price of a used car based on its characteristics.

Dataset Overview

The dataset I am using for this project consists of 9,582 records, detailing various attributes of used cars. The columns in the dataset include:

- Brand: The manufacturer of the car (e.g., Honda, Toyota).
- Model: The specific car model.
- Year: Year of manufacture.
- Age: Age of the car in years, derived from the year of manufacture.
- kmDriven: Distance the car has been driven, provided in kilometers.
- Transmission: The type of transmission system (Manual or Automatic).

- Owner: Ownership status, indicating whether the car is first-hand, second-hand, etc.
- FuelType: The type of fuel used by the car (e.g., Petrol, Diesel).
- PostedDate: The date when the car was listed for sale.
- AdditionInfo: Additional descriptive details about the car.
- AskPrice: The asking price for the car, including currency symbols.

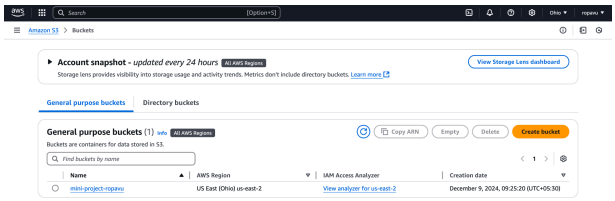
While the dataset provides rich information, certain aspects, such as inconsistent formatting in fields like kmDriven and AskPrice, will require cleaning and preprocessing before analysis.

Methodology:

Environment Setup

1. AWS S3 for Data Storage:

a. Step 1: Created an S3 bucket to store both raw and processed data.



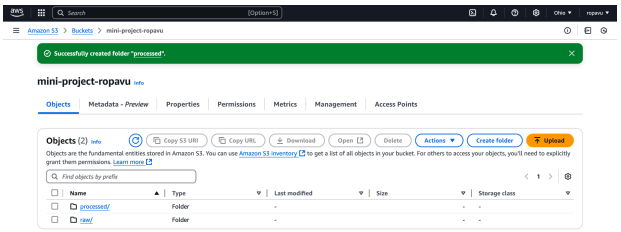
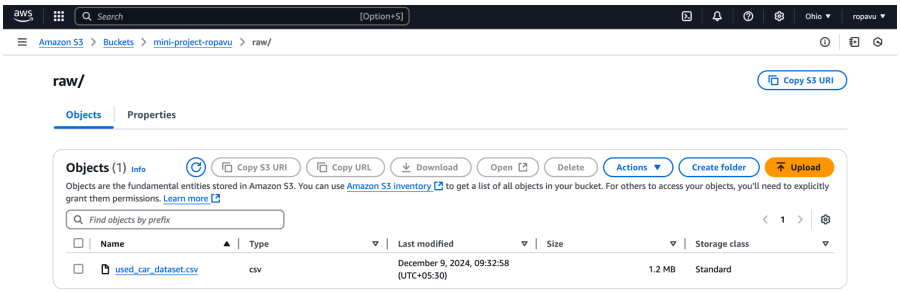


fig-1 fig-2

b. Step 2: Uploaded the raw dataset to the S3 bucket. [fig-3]



Used the dataset from:

https://www.kaggle.com/datasets/mohitkumar282/used-car-dataset

2. Linux Environment with PySpark:

a. Step 1: Used an AWS EC2 instance for better scalability and AWS Integration.

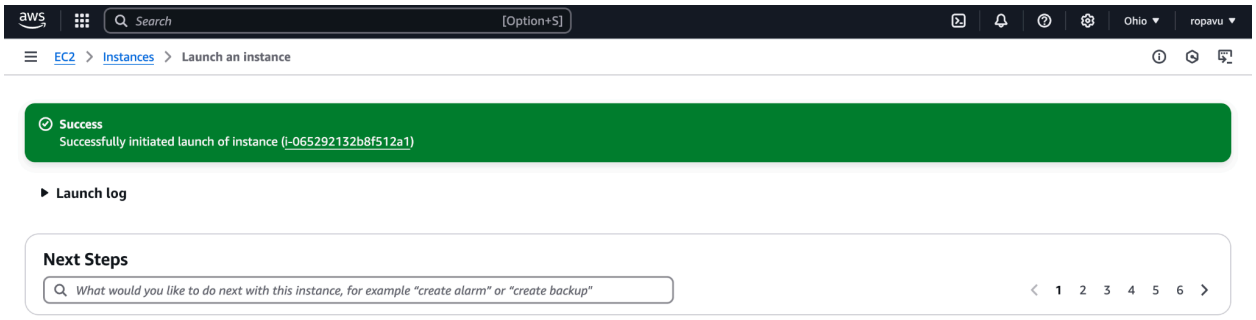


fig-4

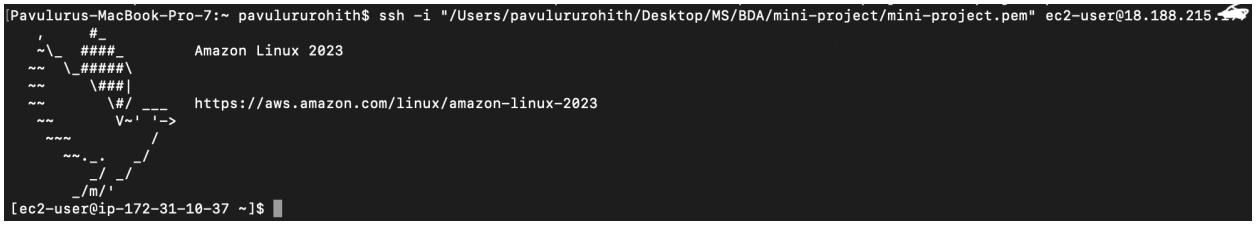


fig-5

Created an EC2 instance and established a SSH connection.

b. Step 2: Installed PySpark and confirmed installation for distributed data processing.

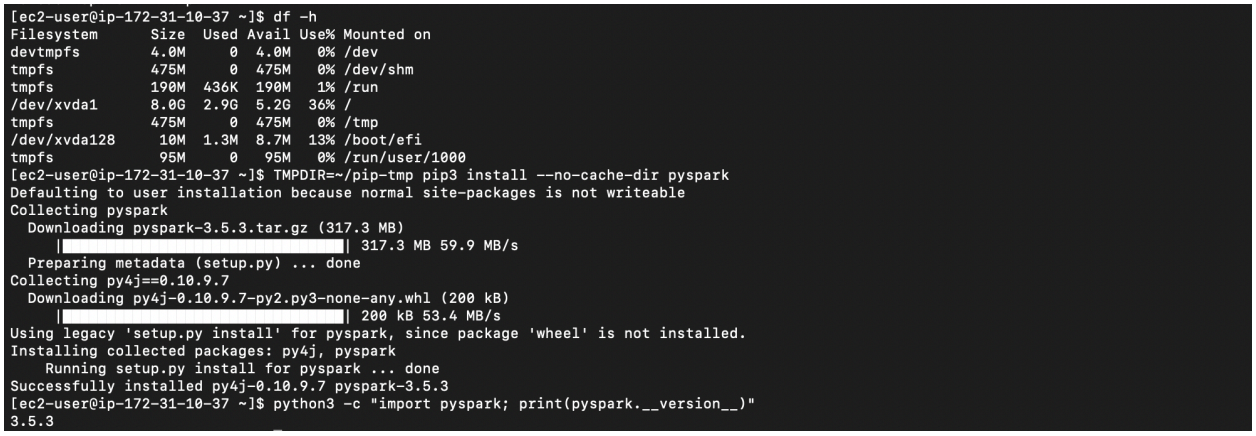


fig-6

There were many other dependencies that had to be installed before PySpark.

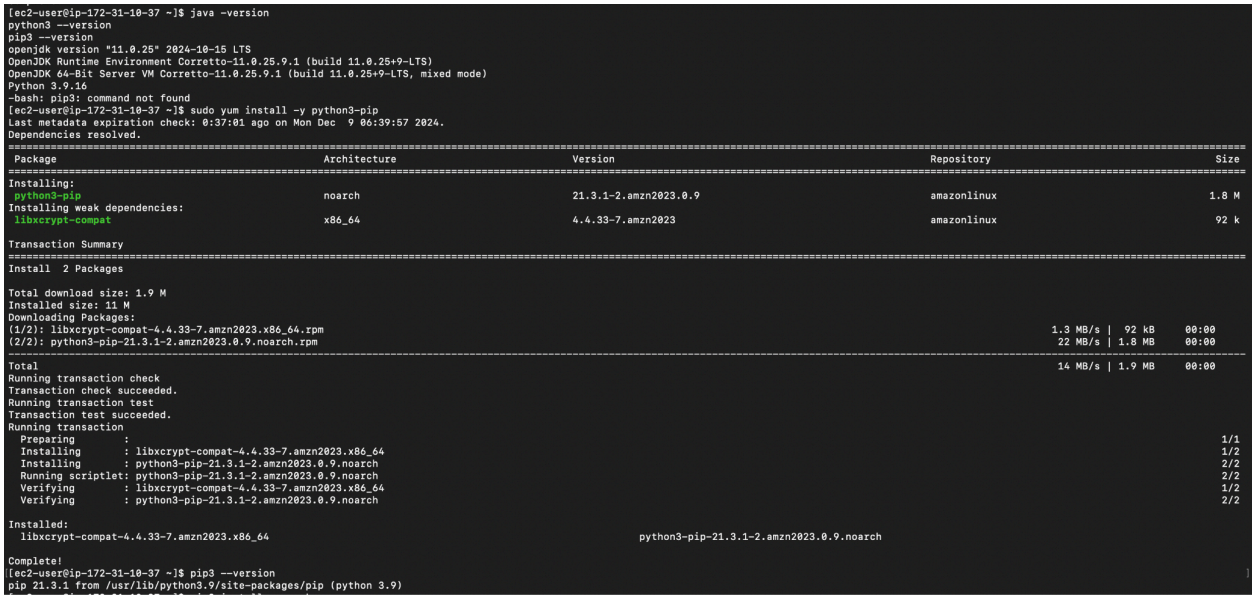


fig-7

c. Step 3: Configured AWS CLI to interact with S3 buckets.

Note: I am not attaching aws configure as there is sensitive information.

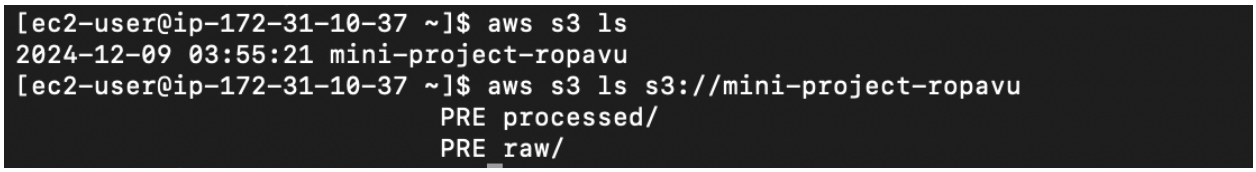


fig-8

fig-9

Downloaded a file from S3 bucket to confirm the connection.

Data Pipeline Tasks

1. Data Ingestion from S3:

a. Step 1: Used AWS CLI to load the dataset directly.

```
[ec2-user@ip-172-31-10-37 ~]$ aws s3 cp s3://mini-project-ropavu/raw/used_car_dataset.csv ./
download: s3://mini-project-ropavu/raw/used_car_dataset.csv to ./used_car_dataset.csv
```

fig-10

b. Step 2: Confirmed successful ingestion by inspecting the dataset using head command.

```
[ec2-user@ip-172-31-10-37 ~]$ head used_car_dataset.csv
Brand,model,Year,Age,kmDriven,Transmission,Owner,FuelType,PostedDate,AdditionInfo,AskPrice
Honda,City,2001,23,"98,000 km",Manual,second,Petrol,Nov-24,"Honda City v teck in mint condition, valid genuine car,","₹ 1,95,000"
Toyota,Innova,2009,15,190000.0 km,Manual,second,Diesel,Jul-24,"Toyota Innova 2.5 G (Diesel) 7 Seater, 2009, Diesel","₹ 3,75,000"
Volkswagen,VentoTest,2010,14,"77,246 km",Manual,first,Diesel,Nov-24,"Volkswagen Vento 2010-2013 Diesel Breeze, 2010, Diesel","₹ 1,84,999"
Maruti Suzuki,Swift,2017,7,"83,500 km",Manual,second,Diesel,Nov-24,Maruti Suzuki Swift 2017 Diesel Good Condition,"₹ 5,65,000"
Maruti Suzuki,Baleno,2019,5,"45,000 km",Automatic,first,Petrol,Nov-24,"Maruti Suzuki Baleno Alpha CVT, 2019, Petrol","₹ 6,85,000"
BMW,X3,2014,10,"83,000 km",Automatic,first,Diesel,Nov-24,"BMW X3 2.0 XDRIVE 20D, 2014, Diesel","₹ 13,50,000"
Toyota,Innova,2014,10,"168,000 km",Manual,second,Diesel,Nov-24,"Toyota Innova 2.5 VX 8 Seater BS IV, 2014, Diesel","₹ 10,25,000"
BMW,5 Series,2019,5,"25,000 km",Automatic,second,Diesel,Nov-24,"BMW 5 Series 3.0 530D M Sport, 2019, Diesel","₹ 59,50,000"
Maruti Suzuki,maruti-suzuki-dzire,2020,4,"33,759 km",Manual,second,Petrol,Nov-24,"Maruti Suzuki Dzire 1.2 VXI, 2020, Petrol","₹ 6,22,000"
```

fig-11

2. Data Processing with PySpark:

a. Step 1: Created 2 new columns month and yearPosted to aid in analysis.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, month, year

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("S3 Data Transformation") \
    .config("spark.hadoop.fs.s3a.access.key", "[redacted]") \
    .config("spark.hadoop.fs.s3a.secret.key", "[redacted]") \
    .config("spark.hadoop.fs.s3a.endpoint", "s3.amazonaws.com") \
    .getOrCreate()

# Read the raw dataset from S3
input_path = "s3://mini-project-ropavu/raw/used_car_dataset.csv"
data = spark.read.csv(input_path, header=True, inferSchema=True)

# Add Month and YearPosted columns
data = data.withColumn("Month", month(to_date(col("PostedDate"), "MMM-yy"))) \
           .withColumn("YearPosted", year(to_date(col("PostedDate"), "MMM-yy")))

# Write the transformed data back to S3
output_path = "s3://mini-project-ropavu/processed/transformed_data.csv"
data.write.csv(output_path, header=True, mode="overwrite")
```

fig-12

```
[ec2-user@ip-172-31-10-37 ~]$ head transformed_data/part-00000-9846e3e3-896d-4081-98f9-32ad4bda8aae-c000.csv
Brand,model,Year,Age,kmDriven,Transmission,Owner,FuelType,PostedDate,AdditionInfo,AskPrice,Month,YearPosted
Honda,City,2001,23,98000,Manual,second,Petrol,Nov-24,"Honda City v teck in mint condition, valid genuine car,",195000,Nov,24
Toyota,Innova,2009,15,1900000,Manual,second,Diesel,Jul-24,"Toyota Innova 2.5 G (Diesel) 7 Seater, 2009, Diesel",375000,Jul,24
Volkswagen,VentoTest,2010,14,77246,Manual,first,Diesel,Nov-24,"Volkswagen Vento 2010-2013 Diesel Breeze, 2010, Diesel",184999,Nov,24
Maruti Suzuki,Swift,2017,7,83500,Manual,second,Diesel,Nov-24,Maruti Suzuki Swift 2017 Diesel Good Condition,565000,Nov,24
Maruti Suzuki,Baleno,2019,5,45000,Automatic,first,Petrol,Nov-24,"Maruti Suzuki Baleno Alpha CVT, 2019, Petrol",685000,Nov,24
BMW,X3,2014,10,83000,Automatic,first,Diesel,Nov-24,"BMW X3 2.0 XDRIVE 20D, 2014, Diesel",1350000,Nov,24
Toyota,Innova,2014,10,168000,Manual,second,Diesel,Nov-24,"Toyota Innova 2.5 VX 8 Seater BS IV, 2014, Diesel",1025000,Nov,24
BMW,5 Series,2019,5,25000,Automatic,second,Diesel,Nov-24,"BMW 5 Series 3.0 530D M Sport, 2019, Diesel",5950000,Nov,24
Maruti Suzuki,maruti-suzuki-dzire,2020,4,33759,Manual,second,Petrol,Nov-24,"Maruti Suzuki Dzire 1.2 VXI, 2020, Petrol",622000,Nov,24
```

fig-13

Confirmed the result using head command.

b. Step 2: Computed 5 key metrics:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, split, regexp_replace, sum, avg

# Initialize Spark Session with S3 support
spark = SparkSession.builder \
    .appName("S3 Data Processing") \
    .config("spark.hadoop.fs.s3a.awsAccessKeyId", "[redacted]") \
    .config("spark.hadoop.fs.s3a.awsSecretAccessKey", "[redacted]") \
    .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .config("spark.hadoop.fs.s3a.endpoint", "s3.amazonaws.com") \
    .config("spark.hadoop.fs.s3a.connection.maximum", "100") \
    .config("spark.hadoop.fs.s3a.attempts.max", "10") \
    .config("spark.hadoop.fs.s3a.paging.maximum", "1000") \
    .getOrCreate()

# Load dataset from S3
file_path = "s3a://mini-project-ropavu/raw/used_car_dataset.csv"
df = spark.read.csv("./used_car_dataset.csv", header=True, inferSchema=True)

# Handle missing values before transformations if necessary
df = df.dropna(subset=["PostedDate", "kmDriven", "AskPrice", "Brand", "Transmission"])

# Transform the data
df = df.withColumn("Month", split(col("PostedDate"), "-")[0]) \
       .withColumn("YearPosted", split(col("PostedDate"), "-")[1]) \
       .withColumn("kmDriven", regexp_replace(col("kmDriven"), "[^0-9]", "").cast("int")) \
       .withColumn("AskPrice", regexp_replace(col("AskPrice"), "[^0-9]", "").cast("int"))

# Optionally, cache the transformed dataframe if reused multiple times
df.cache()

# Perform Aggregations
total_revenue_by_brand = df.groupBy("Brand").agg(sum("AskPrice").alias("TotalRevenue"))
monthly_spending = df.groupBy("Month").agg(sum("AskPrice").alias("TotalMonthlySpending"))
top_transactions = df.orderBy(col("AskPrice").desc()).limit(10)
avg_age_by_brand = df.groupBy("Brand").agg(avg("Age").alias("AverageAge"))
total_distance_by_transmission = df.groupBy("Transmission").agg(sum("kmDriven").alias("TotalDistanceDriven"))

# Save Results to S3
output_path = "s3a://mini-project-ropavu/processed/"
df.write.csv(f"./transformed_data", header=True, mode="overwrite")
total_revenue_by_brand.write.csv(f"./total_revenue_by_brand", header=True, mode="overwrite")
monthly_spending.write.csv(f"./monthly_spending", header=True, mode="overwrite")
top_transactions.write.csv(f"./top_transactions", header=True, mode="overwrite")
avg_age_by_brand.write.csv(f"./avg_age_by_brand", header=True, mode="overwrite")
total_distance_by_transmission.write.csv(f"./total_distance_by_transmission", header=True, mode="overwrite")

print("Data processing completed successfully!")
```

fig-14

```
[ec2-user@ip-172-31-10-37 ~]$ ls
Miniconda3-latest-Linux-x86_64.sh  hadoop-3.3.6          pip-tmp      s3_process_data.py          top_transactions               transformed_data
avg_age_by_brand                   hadoop-3.3.6.tar.gz  s1.py        spark-3.5.3-bin-hadoop3.tgz total_distance_by_transmission used_car_dataset.csv
aws-java-sdk-1.11.901.jar          monthly_spending     s3_access.py spark_jars                  total_revenue_by_brand
```

fig-15

[fig-14] has the code for all 5 metrics computed.

- Average age by brand:

```
[ec2-user@ip-172-31-10-37 ~]$ head avg_age_by_brand/part-00000-da053092-4c44-4e19-a3b4-9b12832a7c9a-c000.csv
Brand,AverageAge
Volkswagen,8.182965299684543
Lexus,4.85
Jaguar,8.382352941176471
Maserati,7.0
Rolls-Royce,7.0
Tata,6.307291666666667
Jeep,5.202702702702703
Mitsubishi,11.38888888888889
Kia,3.0372670807453415
```

fig-16

- Monthly spending:

```
[ec2-user@ip-172-31-10-37 ~]$ head monthly_spending/part-00000-42d956c9-7af2-45fb-b772-696f965fbe69-c000.csv
Month,TotalMonthlySpending
Oct,750559397
Sep,232020651
Dec,4815000
Aug,98502108
May,9416000
Jun,15663000
Feb,1475001
Nov,9034076279
Mar,1069000
```

fig-17

- Top transactions:

```
[ec2-user@ip-172-31-10-37 ~]$ head top_transactions/part-00000-2d697798-8d50-4d49-b5bd-8ac7ab926e20-c000.csv
Brand,model,Year,Age,kmDriven,Transmission,Owner,FuelType,PostedDate,AdditionInfo,AskPrice,Month,YearPosted
Rolls-Royce,Phantom Series II,2015,9,11500,Automatic,second,Petrol,Aug-24,"Rolls-Royce Phantom Series II EWB, 2015, Petrol",42500000,Aug,24
Lexus,LX,2024,0,24000,Automatic,first,Diesel,Nov-24,"Lexus LX 500d, 2024, Diesel",27750000,Nov,24
Aston Martin,Vanquish,2016,8,257000,Automatic,first,Petrol,Nov-24,"Aston Martin Vanquish 6.0 V12, 2016, Petrol",26400000,Nov,24
Mercedes-Benz,G Class,2019,5,59000,Automatic,first,Petrol,Sep-24,"Mercedes-Benz G Class 63 AMG, 2019, Petrol",26000000,Sep,24
Mercedes-Benz,G Class,2019,5,59000,Automatic,first,Petrol,Nov-24,"Mercedes-Benz G Class G63 AMG, 2019, Petrol",24900000,Nov,24
Toyota,Land Cruiser,2023,1,150000,Automatic,first,Diesel,Nov-24,"Toyota Land Cruiser 3.3 ZX Diesel, 2023, Diesel",24700000,Nov,24
Mercedes-Benz,SL-Class,2024,0,1400,Automatic,first,Petrol,Nov-24,"Mercedes-Benz SL-Class 55 AMG Roadster, 2024, Petrol",24500000,Nov,24
Mercedes-Benz,AMG,2024,0,16000,Automatic,first,Petrol,Nov-24,"Mercedes-Benz AMG GT 63 S, 2024, Petrol",21500000,Nov,24
BMW,7 Series,2023,1,70000,Automatic,first,Petrol,Nov-24,"BMW 7 Series 3.0 740Li M Sport, 2023, Petrol",18500000,Nov,24
```

fig-18

- Total distance by transmission:

```
[ec2-user@ip-172-31-10-37 ~]$ head total_distance_by_transmission/part-00000-39b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv
Transmission,TotalDistanceDriven
Automatic,524921079
Manual,618084055
```

fig-19

- Total revenue by brand:

```
[ec2-user@ip-172-31-10-37 ~]$ head total_revenue_by_brand/part-00000-e406465b-03e5-47ee-b62c-5f2f24b5257d-c000.csv
Brand,TotalRevenue
Volkswagen,224852964
Lexus,148275000
Jaguar,81690999
Maserati,8000000
Rolls-Royce,42735000
Tata,260173825
Jeep,127774342
Mitsubishi,14412199
Kia,223555099
```

fig-20

3. Store Processed Data Back to S3:

a. Step 1: Exported data in CSV [fig-21]

```
[ec2-user@ip-172-31-10-37 ~]$ for dir in top_transactions transformed_data avg_age_by_brand total_distance_by_transmission monthly_spending total_revenue_by_brand; do
  aws s3 cp $dir s3://mini-project-ropavu/processed/$dir --recursive
done
upload: top_transactions/_SUCCESS to s3://mini-project-ropavu/processed/top_transactions/_SUCCESS
upload: top_transactions/._SUCCESS.crc to s3://mini-project-ropavu/processed/top_transactions/._SUCCESS.crc
upload: top_transactions/.part-00000-2d697798-8d50-4d49-b5bd-8ac7ab926e20-c000.csv.crc to s3://mini-project-ropavu/processed/top_transactions/.part-00000-2d697798-8d50-4d49-b5bd-
8ac7ab926e20-c000.csv.crc
upload: top_transactions/part-00000-2d697798-8d50-4d49-b5bd-8ac7ab926e20-c000.csv to s3://mini-project-ropavu/processed/top_transactions/part-00000-2d697798-8d50-4d49-b5bd-8ac7ab
926e20-c000.csv
upload: transformed_data/._SUCCESS.crc to s3://mini-project-ropavu/processed/transformed_data/._SUCCESS.crc
upload: transformed_data/.part-00000-9846e3e3-896d-4081-98f9-32ad4bda8aae-c000.csv.crc to s3://mini-project-ropavu/processed/transformed_data/.part-00000-9846e3e3-896d-4081-98f9-
32ad4bda8aae-c000.csv.crc
upload: transformed_data/_SUCCESS to s3://mini-project-ropavu/processed/transformed_data/_SUCCESS
upload: transformed_data/part-00000-9846e3e3-896d-4081-98f9-32ad4bda8aae-c000.csv to s3://mini-project-ropavu/processed/transformed_data/part-00000-9846e3e3-896d-4081-98f9-32ad4b
da8aae-c000.csv
upload: avg_age_by_brand/._SUCCESS.crc to s3://mini-project-ropavu/processed/avg_age_by_brand/._SUCCESS.crc
upload: avg_age_by_brand/_SUCCESS to s3://mini-project-ropavu/processed/avg_age_by_brand/_SUCCESS
upload: avg_age_by_brand/.part-00000-da053092-4c44-4e19-a3b4-9b12832a7c9a-c000.csv.crc to s3://mini-project-ropavu/processed/avg_age_by_brand/.part-00000-da053092-4c44-4e19-a3b4-
9b12832a7c9a-c000.csv.crc
upload: avg_age_by_brand/part-00000-da053092-4c44-4e19-a3b4-9b12832a7c9a-c000.csv to s3://mini-project-ropavu/processed/avg_age_by_brand/part-00000-da053092-4c44-4e19-a3b4-9b1283
2a7c9a-c000.csv
upload: total_distance_by_transmission/_SUCCESS to s3://mini-project-ropavu/processed/total_distance_by_transmission/_SUCCESS
upload: total_distance_by_transmission/.part-00000-39b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv.crc to s3://mini-project-ropavu/processed/total_distance_by_transmission/.part-00
000-39b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv.crc
upload: total_distance_by_transmission/._SUCCESS.crc to s3://mini-project-ropavu/processed/total_distance_by_transmission/._SUCCESS.crc
upload: total_distance_by_transmission/part-00000-39b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv to s3://mini-project-ropavu/processed/total_distance_by_transmission/part-00000-39
b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv
upload: monthly_spending/.part-00000-42d956c9-7af2-45fb-b772-696f965fbe69-c000.csv.crc to s3://mini-project-ropavu/processed/monthly_spending/.part-00000-42d956c9-7af2-45fb-b772-
696f965fbe69-c000.csv.crc
upload: monthly_spending/_SUCCESS to s3://mini-project-ropavu/processed/monthly_spending/_SUCCESS
upload: monthly_spending/._SUCCESS.crc to s3://mini-project-ropavu/processed/monthly_spending/._SUCCESS.crc
upload: monthly_spending/part-00000-42d956c9-7af2-45fb-b772-696f965fbe69-c000.csv to s3://mini-project-ropavu/processed/monthly_spending/part-00000-42d956c9-7af2-45fb-b772-696f96
5fbe69-c000.csv
upload: total_revenue_by_brand/._SUCCESS.crc to s3://mini-project-ropavu/processed/total_revenue_by_brand/._SUCCESS.crc
upload: total_revenue_by_brand/.part-00000-e406465b-03e5-47ee-b62c-5f2f24b5257d-c000.csv.crc to s3://mini-project-ropavu/processed/total_revenue_by_brand/.part-00000-e406465b-03e
5-47ee-b62c-5f2f24b5257d-c000.csv.crc
upload: total_revenue_by_brand/part-00000-e406465b-03e5-47ee-b62c-5f2f24b5257d-c000.csv to s3://mini-project-ropavu/processed/total_revenue_by_brand/part-00000-e406465b-03e5-47ee
-b62c-5f2f24b5257d-c000.csv
upload: total_revenue_by_brand/_SUCCESS to s3://mini-project-ropavu/processed/total_revenue_by_brand/_SUCCESS
```

b. Step 2: Uploaded the processed data to a designated S3 location for easy access.

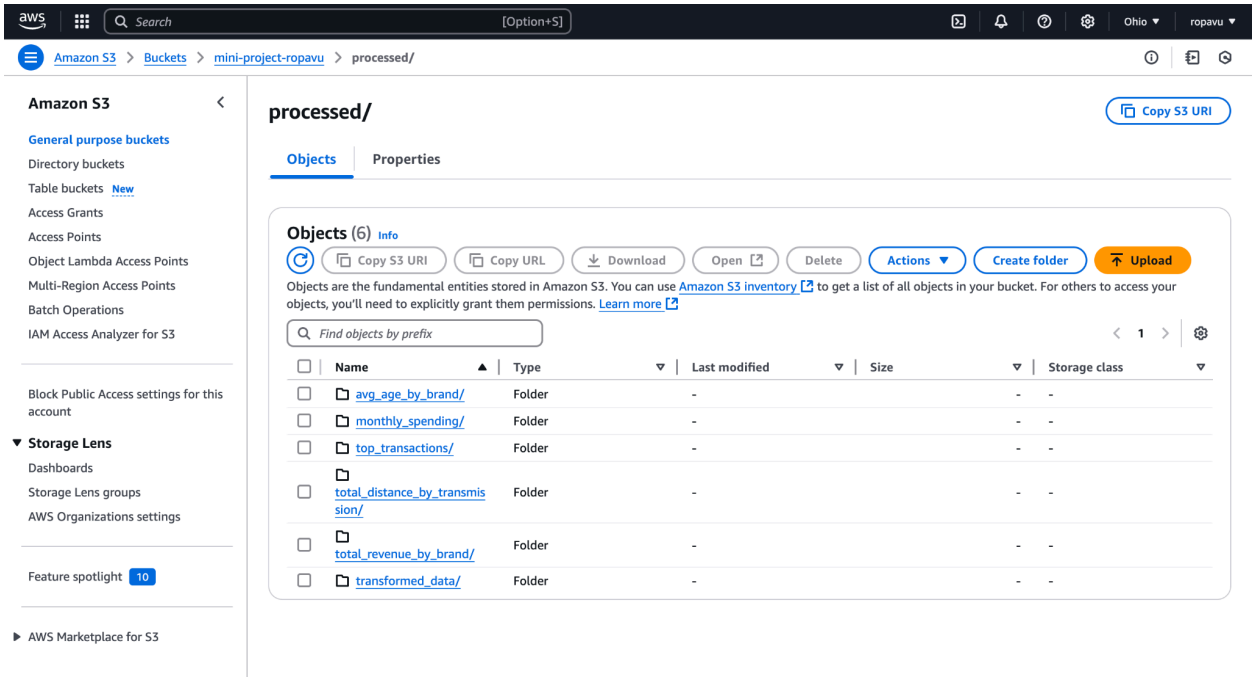


fig-22

4. Data Analysis Using Spark SQL:

All the SQL queries are in a single file. [fig-23]

```python
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder.appName("SQL Insights").getOrCreate()

# Load data into DataFrames
df_top_transactions = spark.read.csv("top_transactions/*.csv", header=True, inferSchema=True)
df_avg_age_by_brand = spark.read.csv("avg_age_by_brand/*.csv", header=True, inferSchema=True)
df_monthly_spending = spark.read.csv("monthly_spending/*.csv", header=True, inferSchema=True)
df_total_distance_by_transmission = spark.read.csv("total_distance_by_transmission/*.csv", header=True, inferSchema=True)
df_total_revenue_by_brand = spark.read.csv("total_revenue_by_brand/*.csv", header=True, inferSchema=True)

# Create temporary views for SQL queries
df_top_transactions.createOrReplaceTempView("top_transactions")
df_avg_age_by_brand.createOrReplaceTempView("avg_age_by_brand")
df_monthly_spending.createOrReplaceTempView("monthly_spending")
df_total_distance_by_transmission.createOrReplaceTempView("total_distance_by_transmission")
df_total_revenue_by_brand.createOrReplaceTempView("total_revenue_by_brand")

# Queries
queries = [
    # Query 1: Identify the top 5 brands by total revenue
    """
    SELECT Brand, TotalRevenue
    FROM total_revenue_by_brand
    ORDER BY TotalRevenue DESC
    LIMIT 5
    """,

    # Query 2: Analyze monthly spending trends
    """
    SELECT Month, TotalMonthlySpending
    FROM monthly_spending
    ORDER BY TotalMonthlySpending DESC
    """,

    # Query 3: Determine the top 5 car brands with the highest average age
    """
    SELECT Brand, AverageAge
    FROM avg_age_by_brand
    ORDER BY AverageAge DESC
    LIMIT 5
    """,

    # Query 4: Analyze the total distance driven grouped by transmission type
    """
    SELECT Transmission, TotalDistanceDriven
    FROM total_distance_by_transmission
    ORDER BY TotalDistanceDriven DESC
    """,

    # Query 5: Find the top 5 most expensive car transactions
    """
    SELECT Brand, model, AskPrice
    FROM top_transactions
    ORDER BY AskPrice DESC
    LIMIT 5
    """
]

# Execute and display results for each query
for i, query in enumerate(queries, 1):
    print(f"Query {i}:")
    result = spark.sql(query)
    result.show()
```

fig-23

- Query 1: Identifies the top 5 car brands by total revenue using total_revenue_by_brand dataset.

```
24/12/12 12:29:46 INFO FileScanRDD: Reading File path: file:///home/ec2-user/total_revenue_by_brand/part-00000-e406465b-03e5-47ee-b62c-5f2f24b5257d-c000.csv, range: 0-660, partit
ion values: [empty row]
24/12/12 12:29:46 INFO CodeGenerator: Code generated in 22.494116 ms
24/12/12 12:29:46 INFO Executor: Finished task 0.0 in stage 10.0 (TID 10). 1777 bytes result sent to driver
24/12/12 12:29:46 INFO TaskSetManager: Finished task 0.0 in stage 10.0 (TID 10) in 230 ms on ip-172-31-10-37.us-east-2.compute.internal (executor driver) (1/1)
24/12/12 12:29:46 INFO TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
24/12/12 12:29:46 INFO DAGScheduler: ResultStage 10 (showString at NativeMethodAccessorImpl.java:0) finished in 0.241 s
24/12/12 12:29:46 INFO DAGScheduler: Job 10 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/12 12:29:46 INFO TaskSchedulerImpl: Killing all running tasks in stage 10: Stage finished
24/12/12 12:29:46 INFO DAGScheduler: Job 10 finished: showString at NativeMethodAccessorImpl.java:0, took 0.247803 s
24/12/12 12:29:46 INFO CodeGenerator: Code generated in 32.437727 ms
24/12/12 12:29:46 INFO CodeGenerator: Code generated in 22.45082 ms
+-------------+------------+
|        Brand|TotalRevenue|
+-------------+------------+
|Mercedes-Benz|  1437065544|
|       Toyota|  1374431516|
|Maruti Suzuki|  1318636754|
|          BMW|   975194548|
|      Hyundai|   914561052|
+-------------+------------+
```

fig-24

- Query 2: Analyzes the monthly spending trends from the monthly_spending dataset.

```
24/12/12 12:29:46 INFO FileScanRDD: Reading File path: file:///home/ec2-user/monthly_spending/part-00000-42d956c9-7af2-45fb-b772-696f965fbe69-c000.csv, range: 0-180, partition va
lues: [empty row]
24/12/12 12:29:46 INFO BlockManagerInfo: Removed broadcast_20_piece0 on ip-172-31-10-37.us-east-2.compute.internal:44103 in memory (size: 34.3 KiB, free: 413.6 MiB)
24/12/12 12:29:46 INFO CodeGenerator: Code generated in 16.31206 ms
24/12/12 12:29:46 INFO Executor: Finished task 0.0 in stage 11.0 (TID 11). 2018 bytes result sent to driver
24/12/12 12:29:46 INFO TaskSetManager: Finished task 0.0 in stage 11.0 (TID 11) in 85 ms on ip-172-31-10-37.us-east-2.compute.internal (executor driver) (1/1)
24/12/12 12:29:46 INFO TaskSchedulerImpl: Removed TaskSet 11.0, whose tasks have all completed, from pool
24/12/12 12:29:46 INFO DAGScheduler: ResultStage 11 (showString at NativeMethodAccessorImpl.java:0) finished in 0.111 s
24/12/12 12:29:46 INFO DAGScheduler: Job 11 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/12 12:29:46 INFO TaskSchedulerImpl: Killing all running tasks in stage 11: Stage finished
24/12/12 12:29:46 INFO DAGScheduler: Job 11 finished: showString at NativeMethodAccessorImpl.java:0, took 0.119421 s
24/12/12 12:29:46 INFO CodeGenerator: Code generated in 10.544267 ms
+-----+--------------------+
|Month|TotalMonthlySpending|
+-----+--------------------+
|  Nov|          9034076279|
|  Oct|           750559397|
|  Sep|           232020651|
|  Aug|            98502108|
|  Jul|            23953000|
|  Jun|            15663000|
|  May|             9416000|
|  Dec|             4815000|
|  Apr|             3245000|
|  Feb|             1475001|
|  Mar|             1069000|
|  Jan|              590000|
+-----+--------------------+
```

fig-25

- Query 3: Finds the top 5 car brands with the highest average age from the avg_age_by_brand dataset.

```
24/12/12 12:29:47 INFO FileScanRDD: Reading File path: file:///home/ec2-user/avg_age_by_brand/part-00000-da053092-4c44-4e19-a3b4-9b12832a7c9a-c000.csv, range: 0-820, partition va
lues: [empty row]
24/12/12 12:29:47 INFO CodeGenerator: Code generated in 15.411178 ms
24/12/12 12:29:47 INFO Executor: Finished task 0.0 in stage 12.0 (TID 12). 1707 bytes result sent to driver
24/12/12 12:29:47 INFO TaskSetManager: Finished task 0.0 in stage 12.0 (TID 12) in 106 ms on ip-172-31-10-37.us-east-2.compute.internal (executor driver) (1/1)
24/12/12 12:29:47 INFO TaskSchedulerImpl: Removed TaskSet 12.0, whose tasks have all completed, from pool
24/12/12 12:29:47 INFO DAGScheduler: ResultStage 12 (showString at NativeMethodAccessorImpl.java:0) finished in 0.114 s
24/12/12 12:29:47 INFO DAGScheduler: Job 12 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/12 12:29:47 INFO TaskSchedulerImpl: Killing all running tasks in stage 12: Stage finished
24/12/12 12:29:47 INFO DAGScheduler: Job 12 finished: showString at NativeMethodAccessorImpl.java:0, took 0.121653 s
24/12/12 12:29:47 INFO CodeGenerator: Code generated in 15.872161 ms
+----------+------------------+
|     Brand|        AverageAge|
+----------+------------------+
|      Opel|              21.0|
|Ambassador|              15.0|
| Chevrolet|11.955056179775282|
|Mitsubishi| 11.38888888888889|
| Ssangyong|              11.0|
+----------+------------------+
```

fig-26

- Query 4: Calculates the total distance driven by each transmission type using the total_distance_by_transmission dataset.

```
24/12/12 12:29:47 INFO FileScanRDD: Reading File path: file:///home/ec2-user/total_distance_by_transmission/part-00000-39b52c71-9368-4451-b1f7-f63e6b5a27e5-c000.csv, range: 0-70,
 partition values: [empty row]
24/12/12 12:29:47 INFO Executor: Finished task 0.0 in stage 13.0 (TID 13). 1536 bytes result sent to driver
24/12/12 12:29:47 INFO TaskSetManager: Finished task 0.0 in stage 13.0 (TID 13) in 30 ms on ip-172-31-10-37.us-east-2.compute.internal (executor driver) (1/1)
24/12/12 12:29:47 INFO TaskSchedulerImpl: Removed TaskSet 13.0, whose tasks have all completed, from pool
24/12/12 12:29:47 INFO DAGScheduler: ResultStage 13 (showString at NativeMethodAccessorImpl.java:0) finished in 0.049 s
24/12/12 12:29:47 INFO DAGScheduler: Job 13 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/12 12:29:47 INFO TaskSchedulerImpl: Killing all running tasks in stage 13: Stage finished
24/12/12 12:29:47 INFO DAGScheduler: Job 13 finished: showString at NativeMethodAccessorImpl.java:0, took 0.057147 s
+------------+------------------+
|Transmission|TotalDistanceDriven|
+------------+------------------+
|      Manual|         618084055|
|   Automatic|         524921079|
+------------+------------------+
```

fig-27

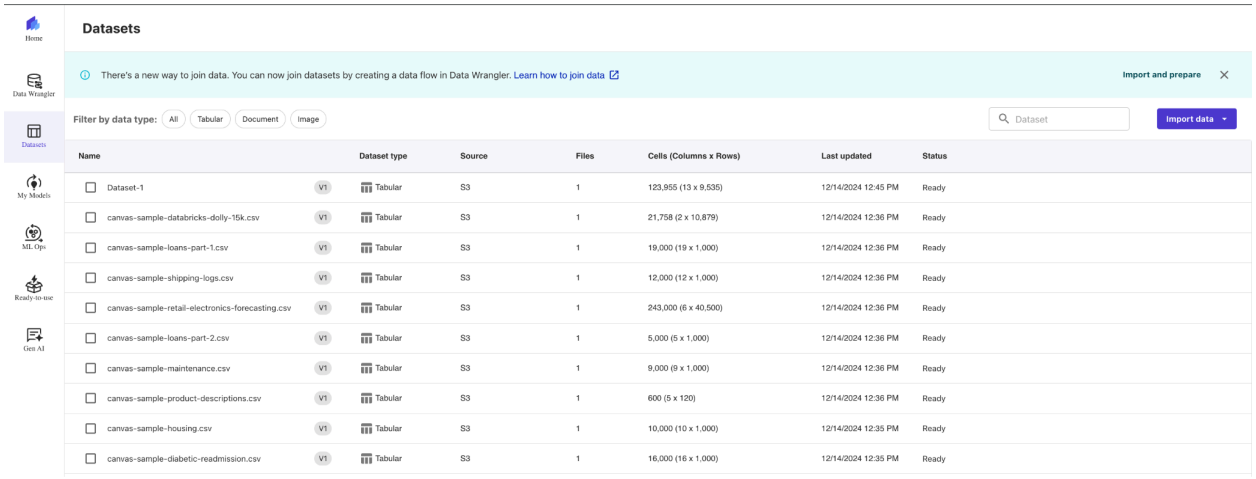
- Query 5: Identifies the top 5 most expensive car transactions from the top_transactions dataset.

```
24/12/12 12:29:47 INFO FileScanRDD: Reading File path: file:///home/ec2-user/top_transactions/part-00000-2d697798-8d50-4d49-b5bd-8ac7ab926e20-c000.csv, range: 0-1361, partition v
alues: [empty row]
24/12/12 12:29:47 INFO CodeGenerator: Code generated in 24.805145 ms
24/12/12 12:29:47 INFO Executor: Finished task 0.0 in stage 14.0 (TID 14). 1803 bytes result sent to driver
24/12/12 12:29:47 INFO TaskSetManager: Finished task 0.0 in stage 14.0 (TID 14) in 86 ms on ip-172-31-10-37.us-east-2.compute.internal (executor driver) (1/1)
24/12/12 12:29:47 INFO TaskSchedulerImpl: Removed TaskSet 14.0, whose tasks have all completed, from pool
24/12/12 12:29:47 INFO DAGScheduler: ResultStage 14 (showString at NativeMethodAccessorImpl.java:0) finished in 0.099 s
24/12/12 12:29:47 INFO DAGScheduler: Job 14 is finished. Cancelling potential speculative or zombie tasks for this job
24/12/12 12:29:47 INFO TaskSchedulerImpl: Killing all running tasks in stage 14: Stage finished
24/12/12 12:29:47 INFO DAGScheduler: Job 14 finished: showString at NativeMethodAccessorImpl.java:0, took 0.106573 s
24/12/12 12:29:47 INFO CodeGenerator: Code generated in 14.565681 ms
24/12/12 12:29:47 INFO CodeGenerator: Code generated in 24.814393 ms
+-------------+-----------------+--------+
|        Brand|            model|AskPrice|
+-------------+-----------------+--------+
|  Rolls-Royce|Phantom Series II|42500000|
|        Lexus|               LX|27750000|
| Aston Martin|         Vanquish|26400000|
|Mercedes-Benz|          G Class|26000000|
|Mercedes-Benz|          G Class|24900000|
+-------------+-----------------+--------+
```

fig-28

5. Machine Learning with AWS SageMaker Autopilot:

a. Step 1: Import Processed Data [fig-29] the dataset loaded is **Dataset-1.**



b. Step 2 - 1: Selected the target column as age in PR-1 model.

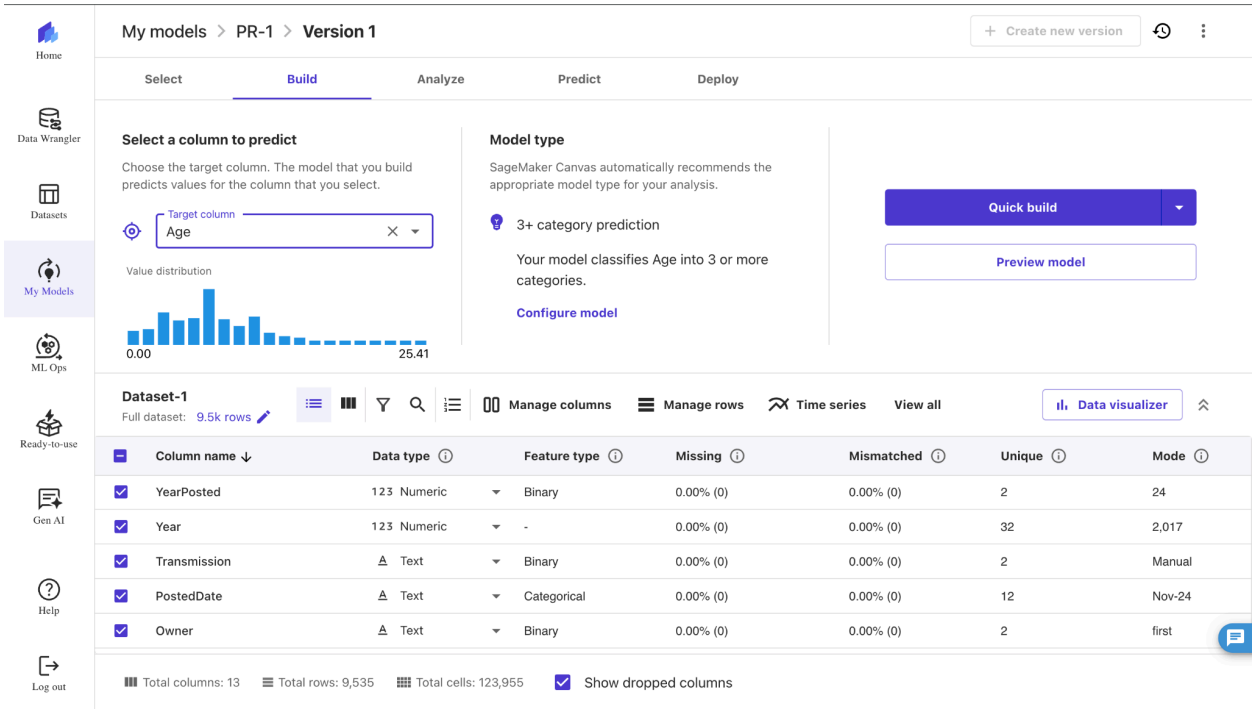
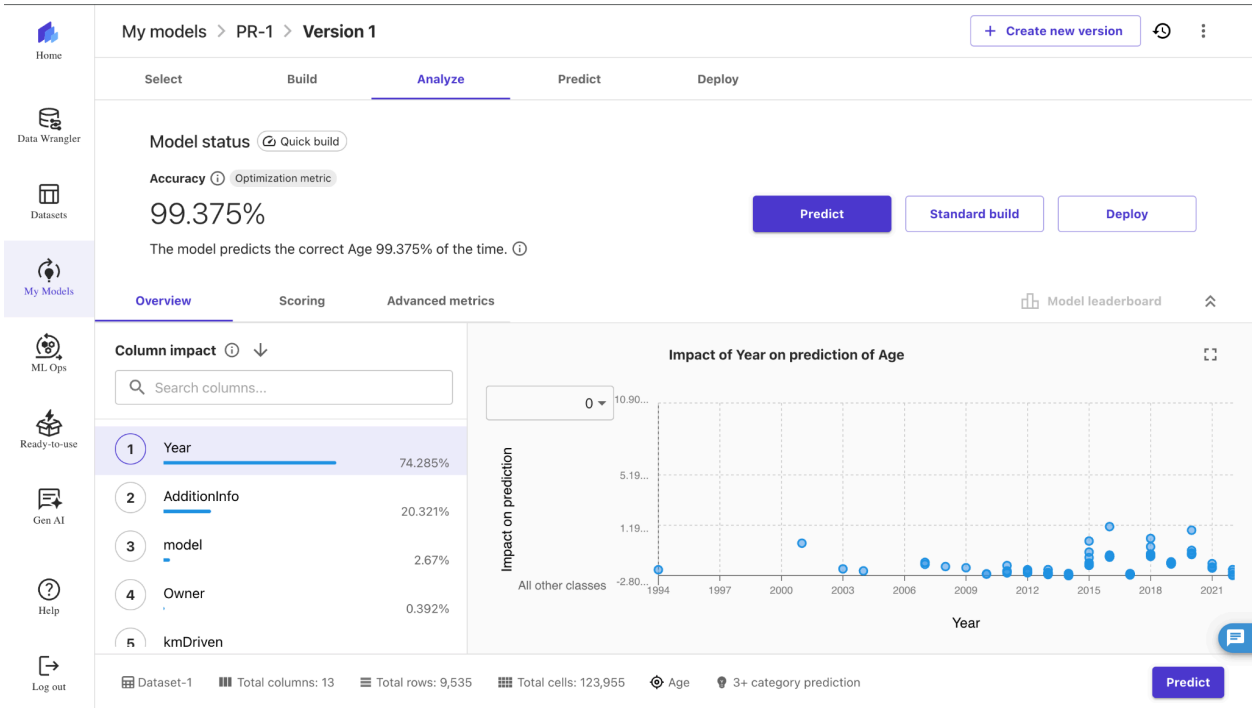


fig-30

b. Step 2 - 2: Ran the AutoML process to train and evaluate multiple models.[fig-31]



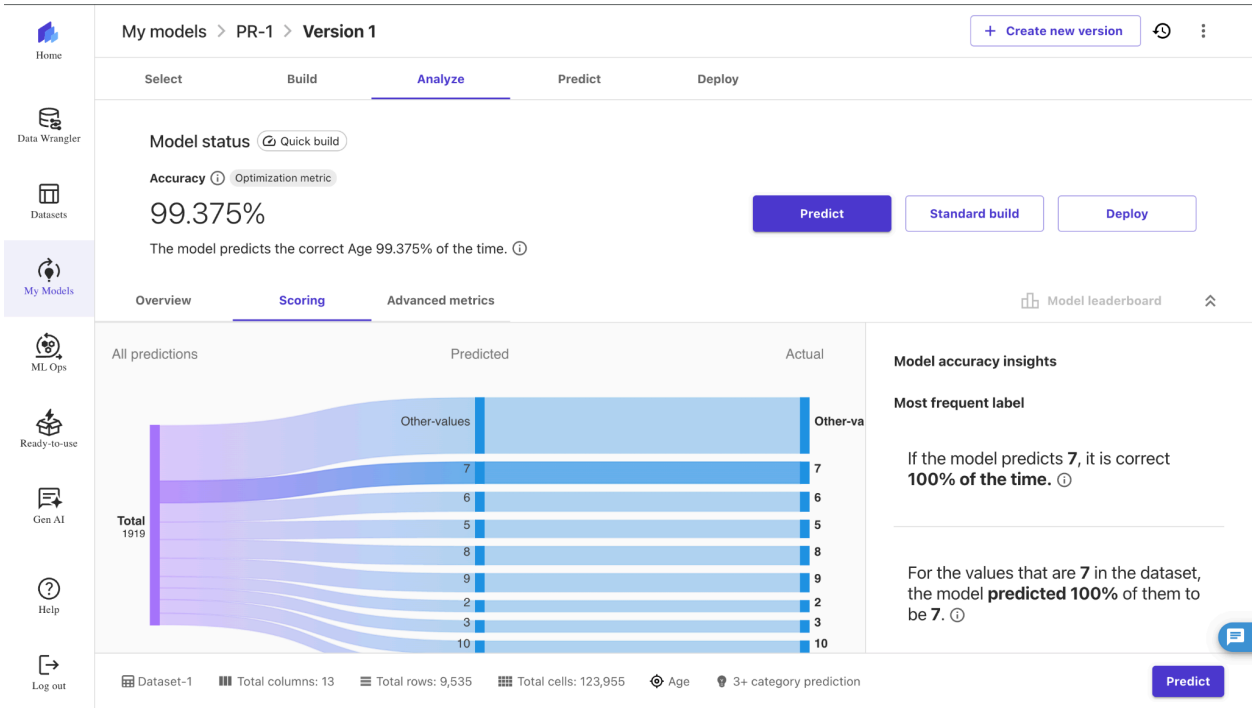

Home
Data Wrangler
Datasets
My Models
ML Ops
Ready-to-use
Gen AI
Help
Log out
My models > PR-1 > Version 1
+ Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Quick build
Accuracy
Optimization metric
99.375%
The model predicts the correct Age 99.375% of the time.
Predict
Standard build
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
All predictions
Predicted
Actual
Other-values
Other-va
Total
1919
7
6
5
8
9
2
3
10
Model accuracy insights
Most frequent label
If the model predicts 7, it is correct 100% of the time.
For the values that are 7 in the dataset, the model predicted 100% of them to be 7.
Dataset-1
Total columns: 13
Total rows: 9,535
Total cells: 123,955
Age
3+ category prediction
Predict

fig-32

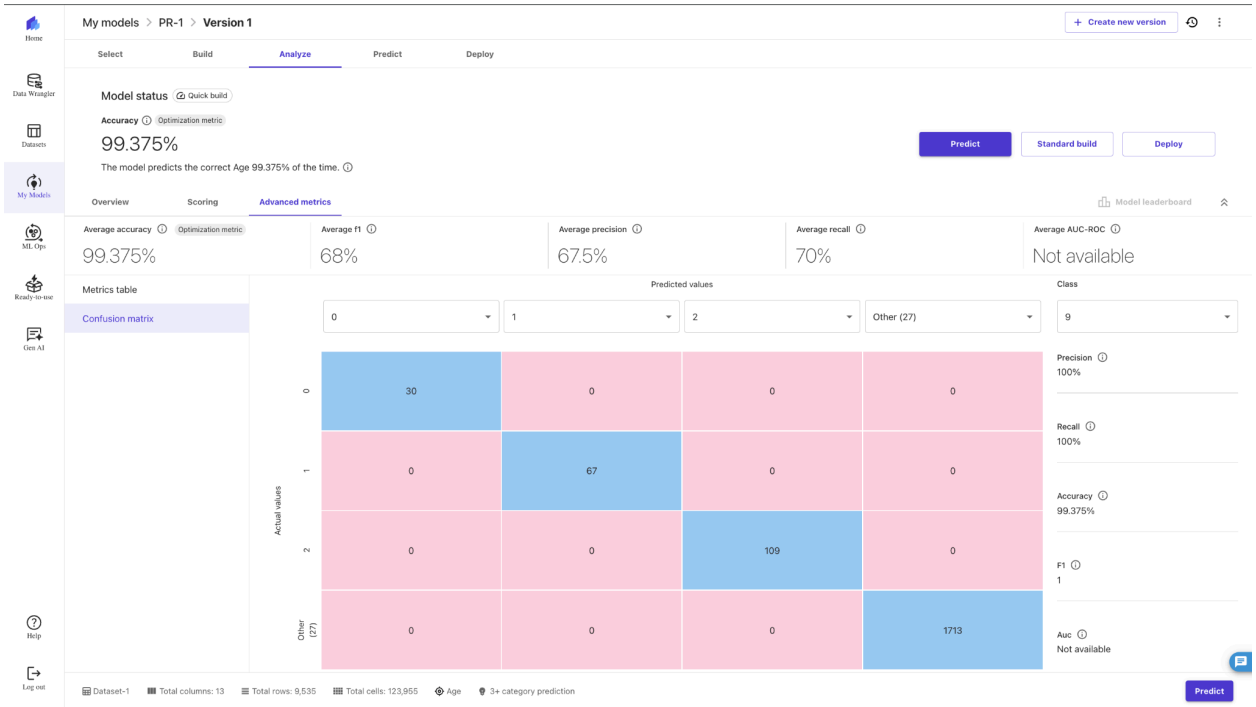

My models > PR-1 > Version 1
+ Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Quick build
Accuracy
Optimization metric
99.375%
The model predicts the correct Age 99.375% of the time.
Predict
Standard build
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
Average accuracy
Optimization metric
99.375%
Average f1
68%
Average precision
67.5%
Average recall
70%
Average AUC-ROC
Not available
Metrics table
Confusion matrix
Predicted values
Class
0
1
2
Other (27)
9
Actual values
30
0
0
0
0
67
0
0
0
0
109
0
0
0
0
1713
Precision
100%
Recall
100%
Accuracy
99.375%
F1
1
Auc
Not available
Dataset-1
Total columns: 13
Total rows: 9,535
Total cells: 123,955
Age
3+ category prediction
Predict

fig-33

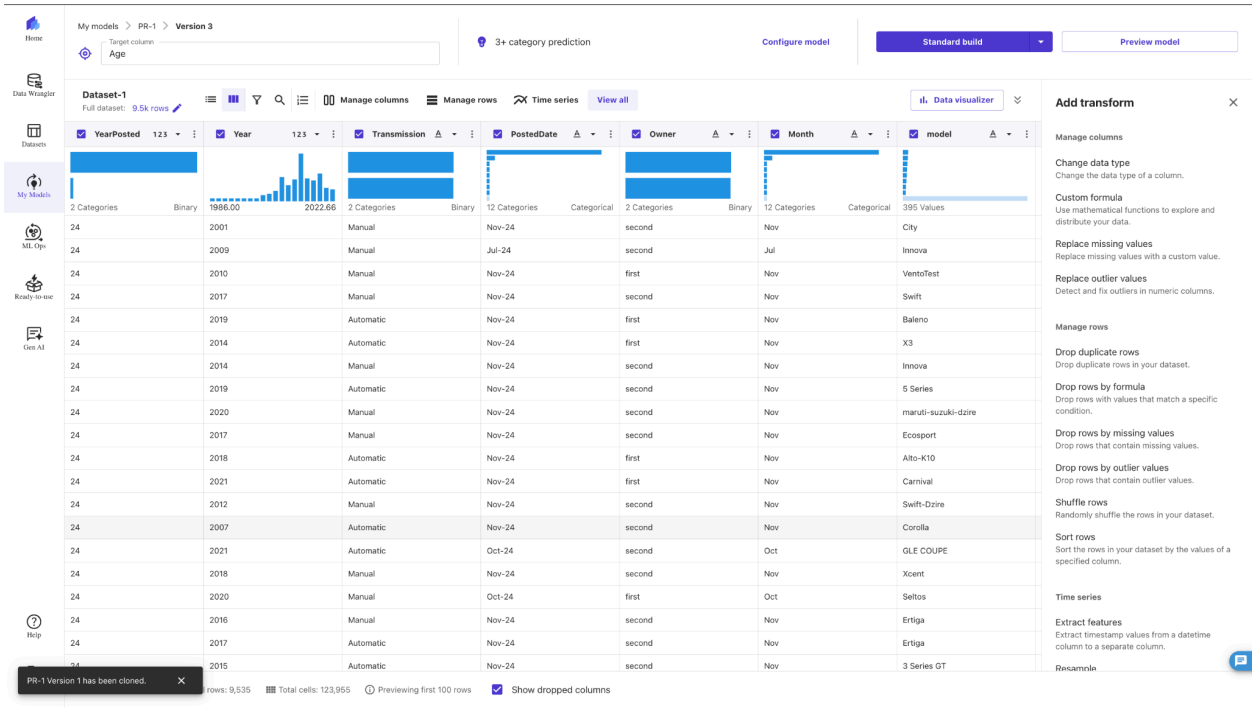


fig-34

b. Step 2 - 1: Selected another target column as brand in PR-2 model,

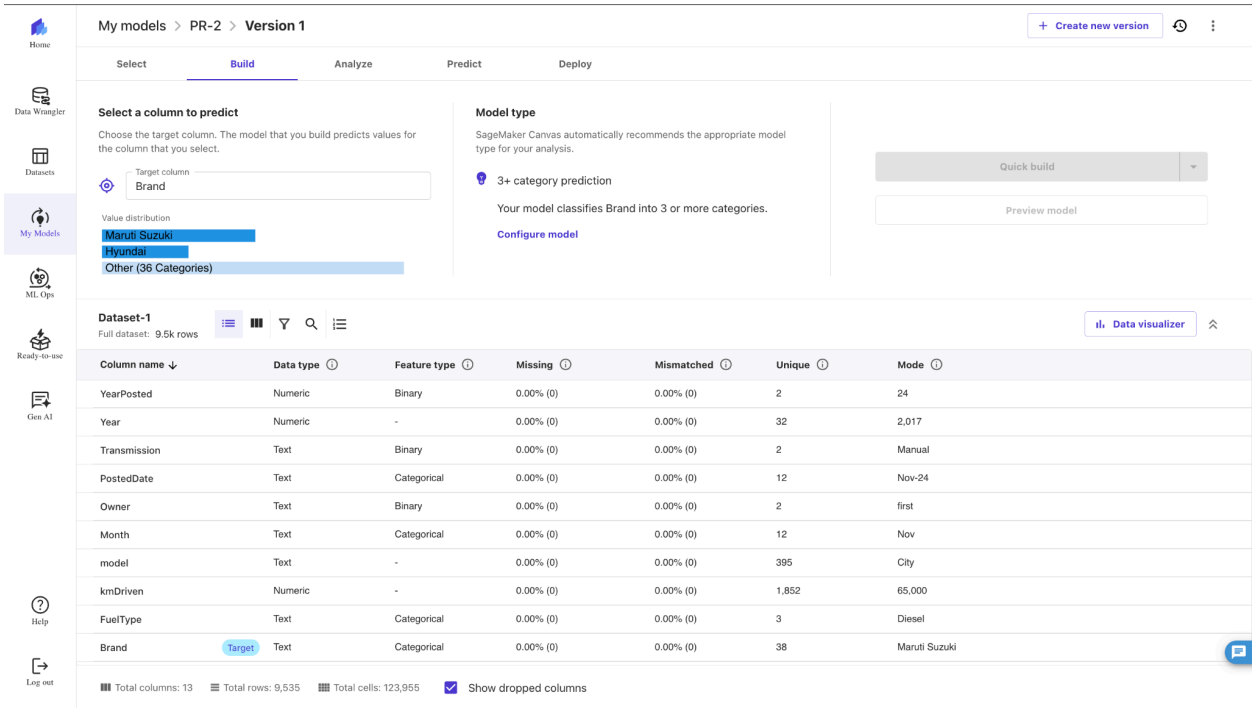


fig-35

b. Step 2 - 2: Ran PR-2 AutoML process to train and evaluate multiple models.

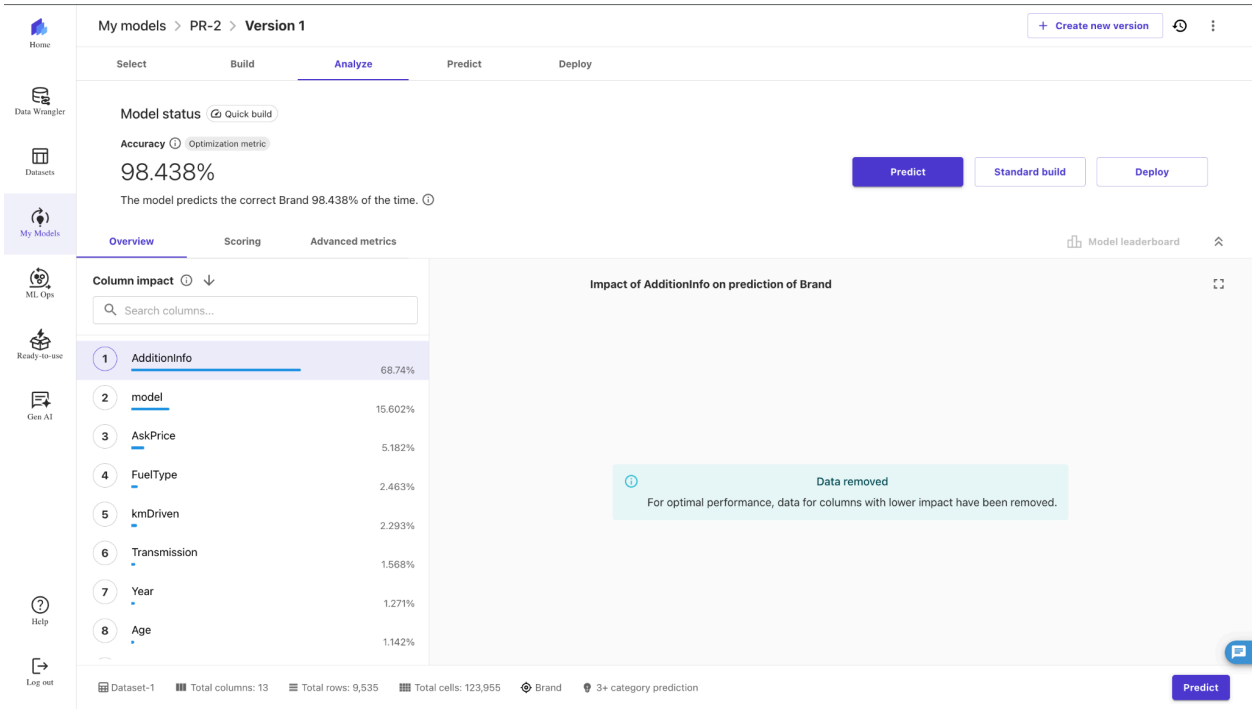
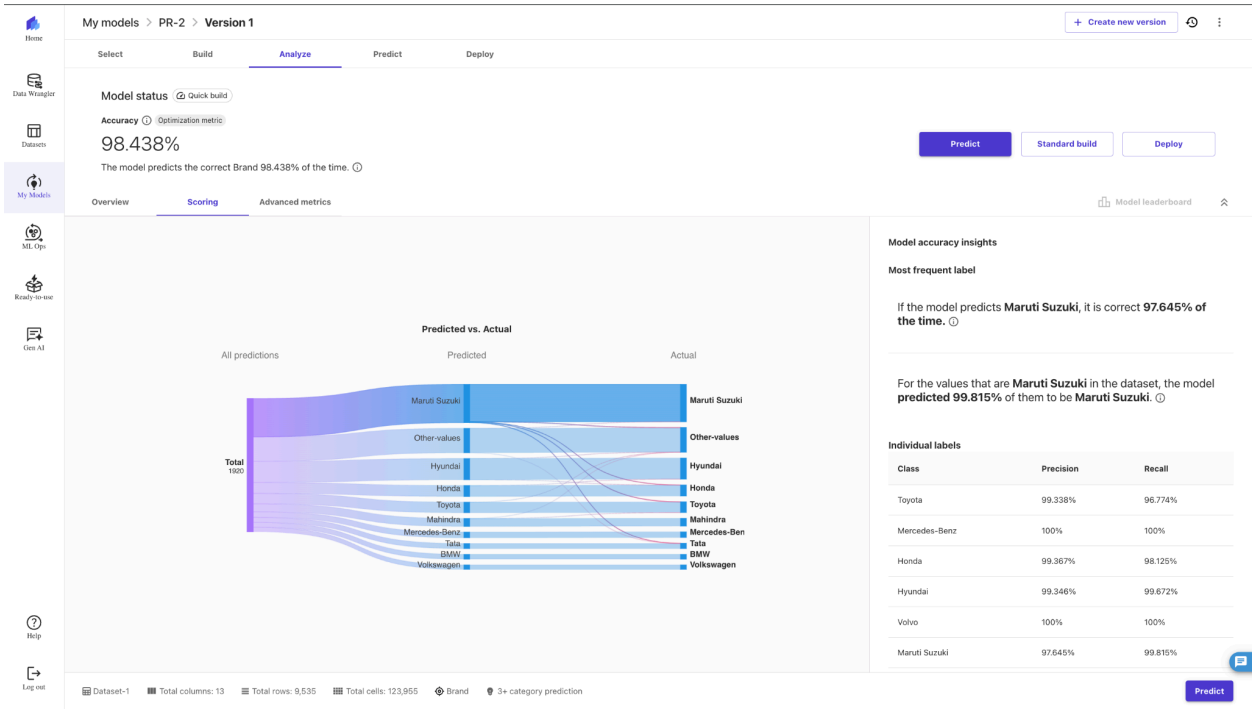


fig-36 | fig-37



The predicted output file for PR-2 is:
https://drive.google.com/file/d/1OPVWEfwLB23RZwujBC5DmEWRx6OYoxRs/view?usp=drive_link

The two models were built.

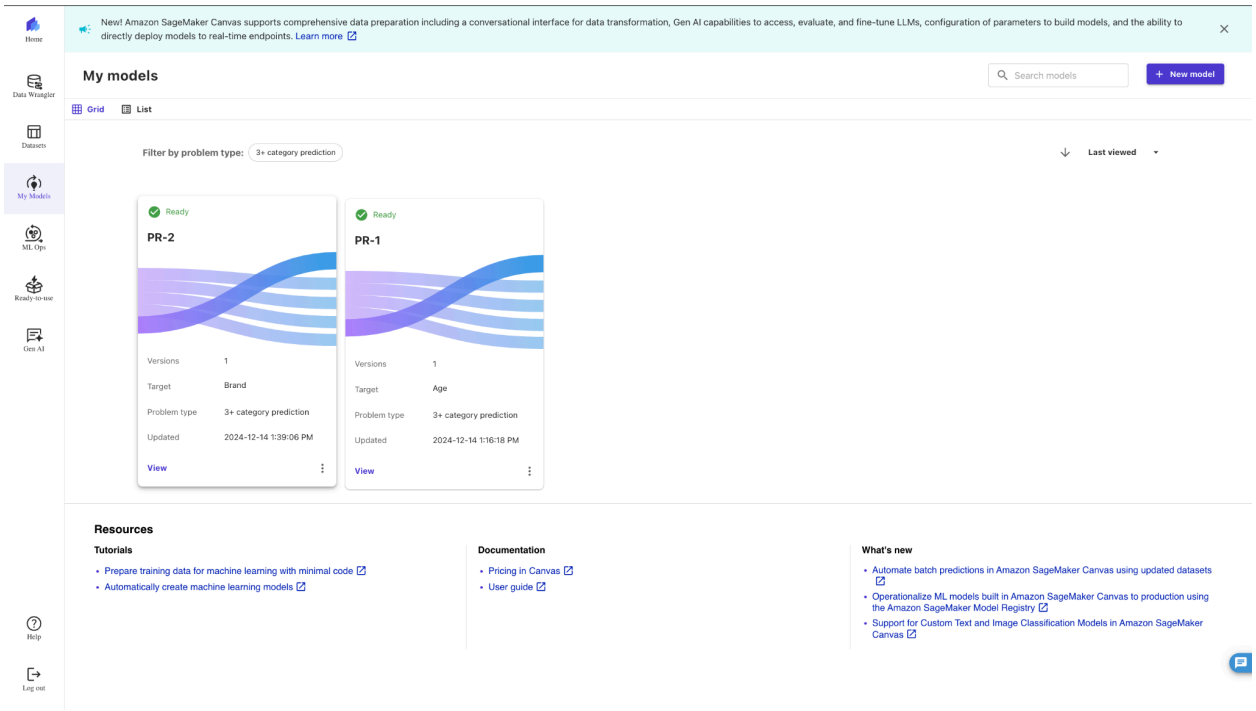


fig-38

Visualization

a. Step 1: Connected QuickSight to the processed data in S3.

Had to create a manifest file i.e, transformed_data.json

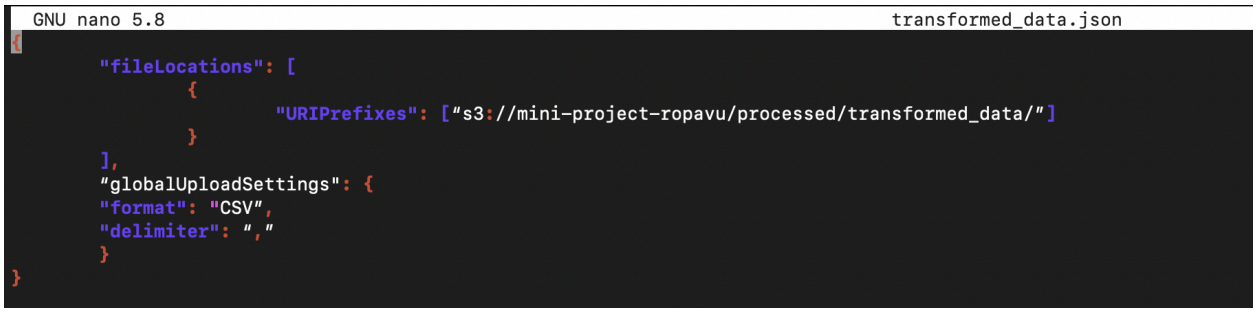


fig-39

Dataset being uploaded as, from S3 to QuickSight

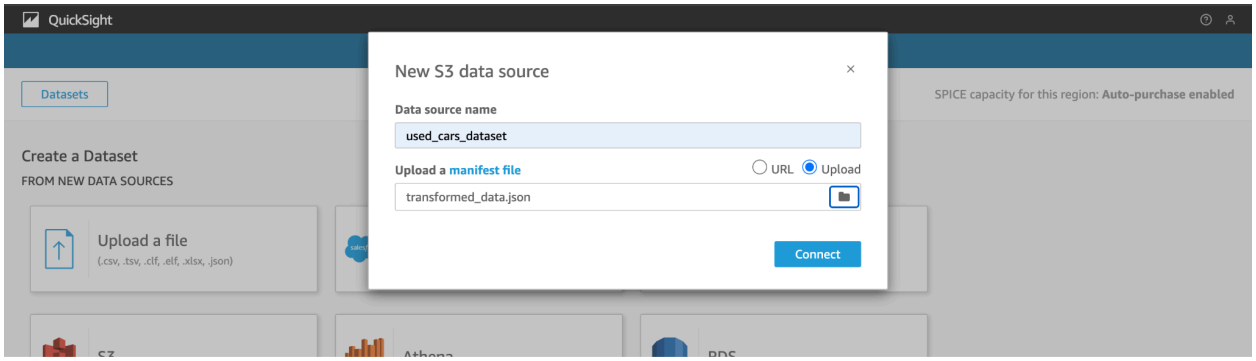


fig-40

b. Step 2: Designed dashboards with insightful visualizations.

- A bar graph on sum of KmDriven and sum of AskPrice by Age.

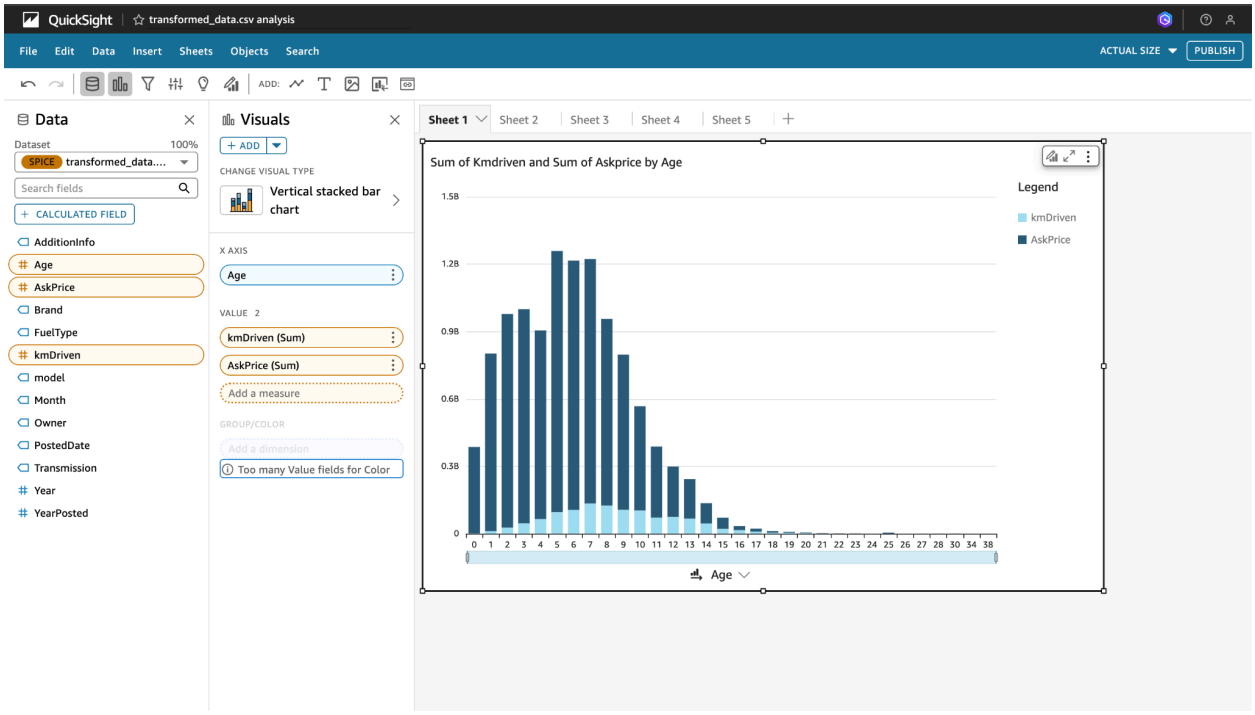


fig-41

Fig-41 demonstrates that there are more cars on sale when the age is between 5-8 years, this is analysed from the sum of KmDriven and AskingPrice.

- A Sankey diagram on count of records by Owner and Brand.

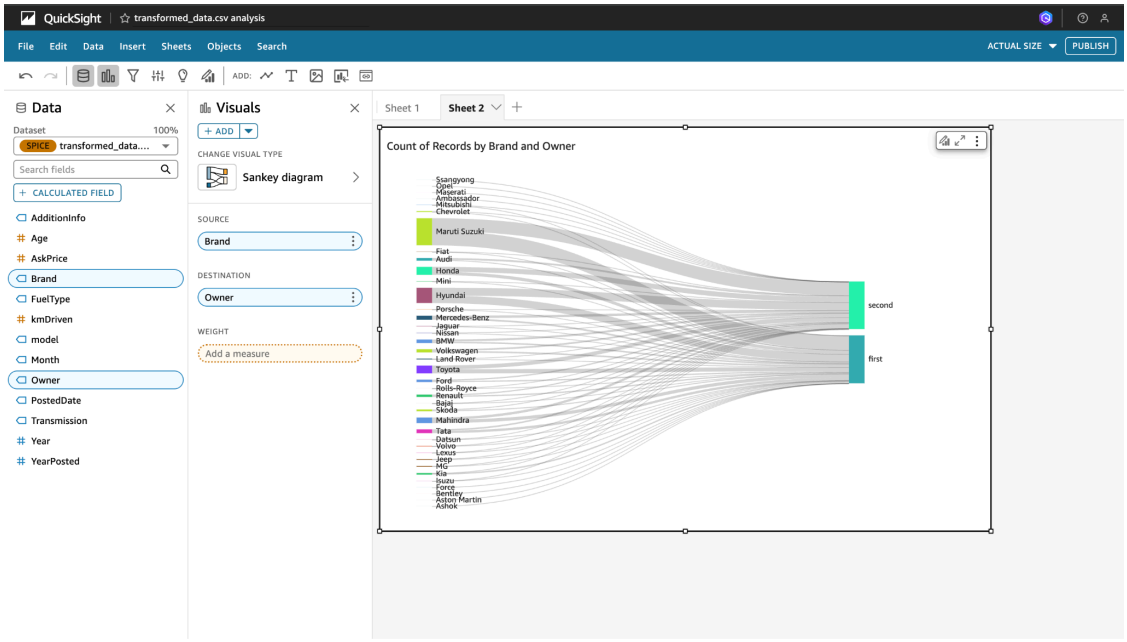


fig-42

Fig-42 illustrates that Maruti Suzuki, Hyundai, Toyota, Mahindra, Honda, Tata are more on the used market than other brands with 1 or more owners.

- A Pie chart on count of records by model and brand. [fig-43]

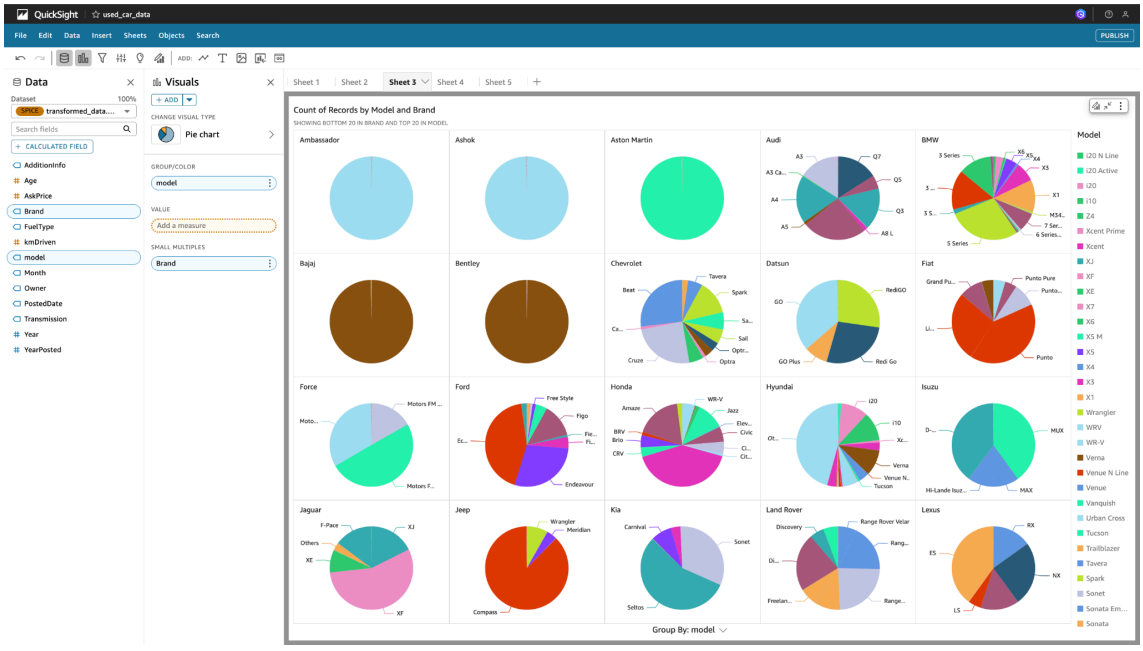


Fig-43 gives us the proportion of each model on the used market from each brand.

- A Waterfall chart for sum of KmDriven by Transmission.

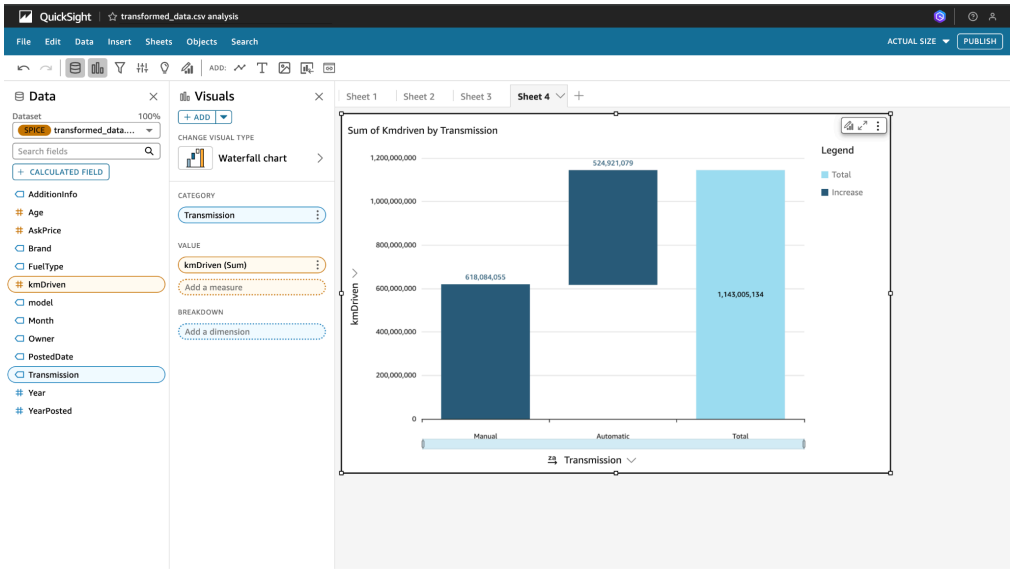
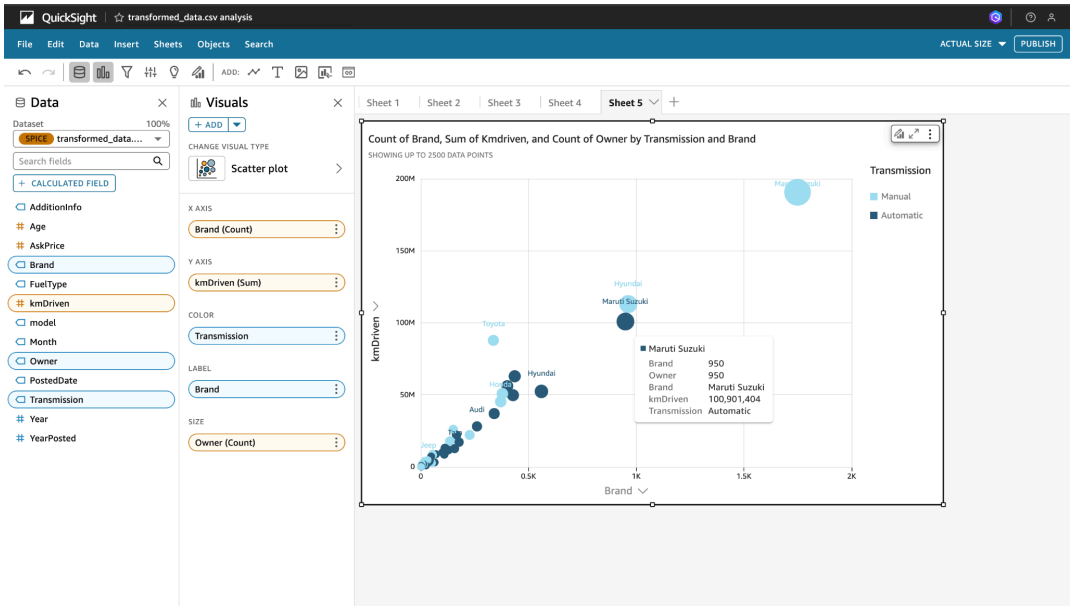


fig-44

Fig-44 shows that cars with manual transmission have been driven more than automatic transmissions.

- A Scatter plot for count of brand, KmDriven, owner by transmission and brand.

fig-45



In fig-45 Maruti Suzuki brand has the most number of cars in manual and automatic transmission. But Toyota has been driven more than average for the number of cars in the market.

Analysis and Results:

Data Aggregation and SQL results:

- Average age by brand: From fig-16 and the query-3 has delivered fig-26 and we can see that brand Opel and Ambassador have the highest average age of cars.
- Monthly spending: From fig-17 and the query-2 has delivered fig-25 and we can see that months of September, October and November have an increasing monthly spending.
- Top transactions: From fig-18 and the query-5 has delivered fig-28 and we can see that Rolls-Royce Phantom Series-ii has the highest asking price.
- Total distance by transmission: From fig-19 and the query-4 has delivered fig-27 and we can see that manual transmission has been driven more than automatic transmission.
- Total revenue by brand: From fig-20 and the query-1 has delivered fig-24 and we can see that Mercedes-Benz, Toyota and Maruti Suzuki are the highest revenue makers in order.

Machine Learning with AWS SageMaker Autopilot results:

- The model PR-1 fig-30 after selecting “age” as the target column, the model predicts 99.375% fig-31 correctly. Interestingly if the car's age is 7 years then the model predicts correctly 100% of the time fig-32.
- The model PR-2 fig-35 after selecting “brand” as the target column, the model predicts 98.438% fig-36 correctly. Interestingly if the value is Maruti Suzuki the model predicts 99.815% correctly fig-37 and model predicts Maruti Suzuki 97.645% correctly fig-37.

Visualization results:

- A bar graph (Fig-41) illustrating the sum of kmDriven and the sum of AskPrice by Age reveals that the majority of cars listed for sale fall within the 5-8 year age range. This conclusion is drawn from the higher cumulative values of kmDriven and AskPrice observed in this age group.
- A Sankey diagram (Fig-42) depicting the count of records by Owner and Brand shows that Maruti Suzuki, Hyundai, Toyota, Mahindra, Honda, and Tata dominate the used car market, with a significant presence across vehicles with one or more previous owners.
- A pie chart (Fig-43) representing the count of records by Model and Brand highlights the proportion of each model available in the used car market for every brand.
- A waterfall chart (Fig-44) illustrating the sum of kmDriven by Transmission reveals that cars with manual transmissions have accumulated more total kilometers compared to those with automatic transmissions.
- A scatter plot (Fig-45) displaying the count of Brand, kmDriven, and Owner by Transmission and Brand highlights that Maruti Suzuki has the highest number of cars in both manual and automatic transmissions. However, Toyota stands out as having a higher-than-average distance driven relative to the number of cars available in the market.

Conclusion:

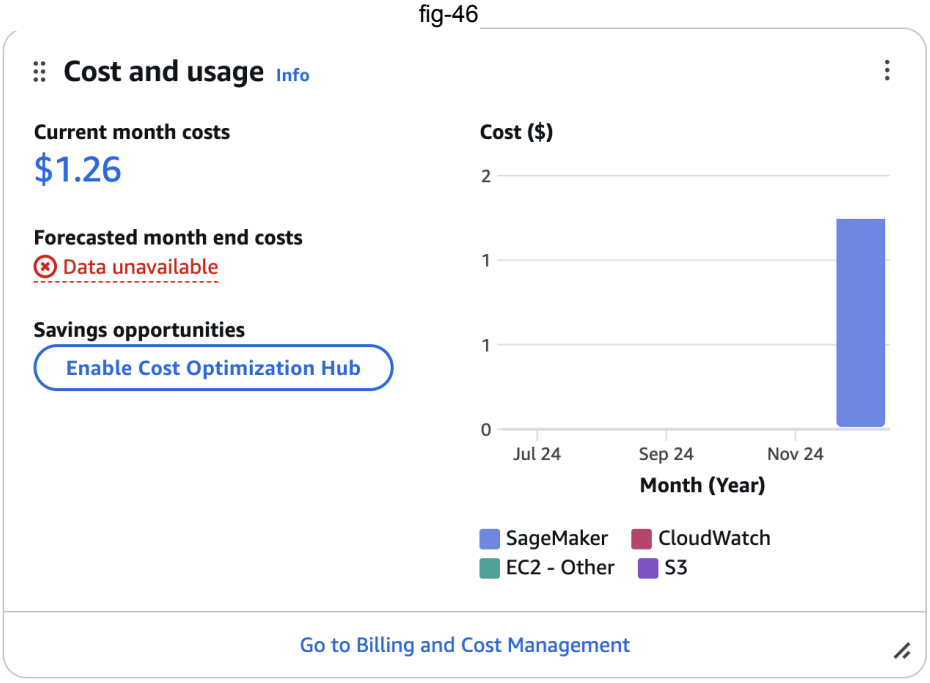
Through this analysis, I have gained valuable insights into the used car market. Brands like Opel and Ambassador have the oldest cars on average, while the majority of cars listed for sale are within the 5-8 year age range, as shown by their cumulative mileage and asking prices. I also observed a notable increase in spending during September, October, and November, suggesting festival seasons and giving offers in the market. The Rolls-Royce Phantom Series-II emerged as the most expensive car on the market, showcasing the upper

echelon of luxury transactions. Additionally, cars with manual transmissions have been driven more extensively than their automatic counterparts, and brands like Mercedes-Benz, Toyota, and Maruti Suzuki have proven to be the highest revenue generators.

Using machine learning models through AWS SageMaker Autopilot, I was able to achieve high predictive accuracy for key attributes. For instance, the model accurately predicted car age with 99.375% accuracy and identified cars aged seven years with 100% precision. Similarly, predictions for the brand attribute showed exceptional performance, particularly for Maruti Suzuki, which was predicted with 99.815% accuracy. These findings highlight the dynamics of the used car market and provide a strong foundation for understanding buyer preferences, vehicle performance, and market trends. This analysis not only benefits industry stakeholders but also provides actionable insights for buyers and sellers aiming to navigate the used car market more effectively.

Road-Blocks:

fig-46



During the course of this project, I encountered several challenges:

- **AWS Service Costs**: A cost of $1.26 was incurred for using AWS services after exceeding the free-tier limits.
- **Installing PySpark**: Installing PySpark required numerous prerequisites, which added complexity to the setup process.
- **AWS SageMaker Issues**: The newer version of AWS SageMaker in Canvas experienced connectivity issues, frequently displaying a "502 Bad Gateway" error. Fortunately, this issue resolved itself after a day.
- **QuickSight Manifest File**: While using QuickSight, I had to create a manifest file that was less than 1MB in size and in .json format. This requirement was initially unclear and required additional effort to implement correctly.

References:

- **AWS SageMaker Documentation:** https://docs.aws.amazon.com/sagemaker/
- **PySpark Installation Guide:** https://spark.apache.org/docs/latest/
- **Amazon QuickSight User Guide:** https://docs.aws.amazon.com/quicksight/
- **Used Car Market Analysis:** Industry Trends and Insights, Market Reports (2023).
- **Dataset Source:** Ihttps://www.kaggle.com/datasets/mohitkumar282/used-car-dataset