# Sakina Banu

**Databricks**

# 🚗 Databricks PySpark Execution Using cars.csv

## Step 1: Configure Google Cloud Storage (GCS) Access

spark.conf.set("spark.hadoop.google.cloud.auth.service.account.enable", "true")  
spark.conf.set("spark.hadoop.fs.gs.auth.service.account.email", "databricks-compute@calcium-field-455700-p2.iam.gserviceaccount.com")  
spark.conf.set("spark.hadoop.fs.gs.project.id", "calcium-field-455700-p2")  
spark.conf.set("spark.hadoop.fs.gs.auth.service.account.private.key", "<your\_private\_key\_here>")  
spark.conf.set("spark.hadoop.fs.gs.auth.service.account.private.key.id", "cc792d2f8ef8022756e4bdf8e1f3c2105aa4ec1c")

## Step 2: Load the Dataset from GCS

df = spark.read.csv("gs://heart\_raw-bronze/cars.csv", sep=";", header=True, inferSchema=True)

## Step 3: Display and Inspect the Data

df.show()  
df.printSchema()  
df.count()  
df.describe().show()  
print(df.columns)

## Step 4: Remove Duplicate Cars (if any)

df\_clean = df.dropDuplicates(['Car'])  
df\_clean.show()

## Step 5: Create a New Column - Power to Weight Ratio

from pyspark.sql.functions import col  
  
df = df.withColumn("Power\_to\_Weight", col("Horsepower") / col("Weight"))  
df.select("Car", "Horsepower", "Weight", "Power\_to\_Weight").show()

## Step 6: Filter High MPG Cars

df.filter(col("MPG") > 25).show()

## Step 7: Pivot Table – Average MPG by Origin and Cylinders

from pyspark.sql import functions as F  
  
df.groupBy("Origin").pivot("Cylinders").agg(F.avg(F.col("MPG").cast("double")).alias("avg\_MPG")).show()

## Step 8: Register Temp View and Run SQL Query

df.createOrReplaceTempView("cars")  
  
spark.sql("SELECT Origin, AVG(MPG) as avg\_mpg FROM cars GROUP BY Origin").show()

## Step 9: Use UDF to Convert Car Names to Uppercase

from pyspark.sql.functions import udf  
from pyspark.sql.types import StringType  
  
def upper\_case(name):  
 return name.upper()  
  
upper\_udf = udf(upper\_case, StringType())  
  
df = df.withColumn("Car\_Upper", upper\_udf("Car"))  
df.select("Car", "Car\_Upper").show()

## Step 10: Write Output Partitioned by Origin

df.write.partitionBy("Origin").mode("overwrite").parquet("/tmp/cars\_partitioned/")

## Step 11: Repartition and Coalesce to Optimize Output

df\_repartitioned = df.repartition(5)  
df\_coalesced = df\_repartitioned.coalesce(2)  
df\_coalesced.write.mode("overwrite").parquet("/tmp/cars\_compact/")

## Step 12: Save and Merge Data Using Delta Lake

from delta.tables import DeltaTable  
  
delta\_path = "/tmp/delta\_cars"  
  
# Save as Delta Table  
df.write.format("delta").mode("overwrite").save(delta\_path)  
  
# Merge (upsert) into Delta Table based on Car column  
deltaTable = DeltaTable.forPath(spark, delta\_path)  
  
deltaTable.alias("existing").merge(  
 df.alias("new"),  
 "existing.Car = new.Car"  
).whenNotMatchedInsertAll().execute()

## Summary

This notebook:  
- Connected to GCS using service account.  
- Loaded and explored the cars.csv file.  
- Cleaned and transformed data (e.g., derived metrics, filtered, pivoted).  
- Applied SQL, UDFs, partitioning, and data optimizations.  
- Performed Delta Lake merge (upsert) operations for scalable data storage.

A screenshot of a computer

AI-generated content may be incorrect.