**Exploring Data Analysis using Databricks**

*By Om Gaikwad*

Question 1

# TODO: Read the Parquet files for each city

crimeDataNewYorkDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-New-York-2016.parquet")

crimeDataLosAngelesDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-Los-Angeles-2016.parquet")

crimeDataChicagoDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-Chicago-2016.parquet")

crimeDataPhiladelphiaDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-Philadelphia-2016.parquet")

crimeDataDallasDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-Dallas-2016.parquet")

crimeDataBostonDF = spark.read.parquet("/mnt/training/crime-data-2016/Crime-Data-Boston-2016.parquet")

# Calculate total count of crimes for each city

totalCrimesNewYork = crimeDataNewYorkDF.count()

totalCrimesLosAngeles = crimeDataLosAngelesDF.count()

totalCrimesChicago = crimeDataChicagoDF.count()

totalCrimesPhiladelphia = crimeDataPhiladelphiaDF.count()

totalCrimesDallas = crimeDataDallasDF.count()

totalCrimesBoston = crimeDataBostonDF.count()

# Display the total count of crimes for each city

print("THe Total crimes in New York:", totalCrimesNewYork)

print("THe Total crimes in Los Angeles:", totalCrimesLosAngeles)

print("THe Total crimes in Chicago:", totalCrimesChicago)

print("The Total crimes in Philadelphia:", totalCrimesPhiladelphia)

print("THe Total crimes in Dallas:", totalCrimesDallas)

print("The Total crimes in Boston:", totalCrimesBoston)

A screenshot of a computer code

Description automatically generated

Question 2

# TODO

from pyspark.sql.functions import lower, col

# Calculate different types of crimes for New York

totalCrimesNewYork = crimeDataNewYorkDF.groupBy(lower(col("offenseDescription")).alias("crime\_type")).count()

# Calculate different types of crimes for Los Angeles

totalCrimesLosAngeles = crimeDataLosAngelesDF.groupBy(lower(col("crimecodedescription")).alias("crime\_type")).count()

# Calculate different types of crimes for Chicago

totalCrimesChicago = crimeDataChicagoDF.groupBy(lower(col("primarytype")).alias("crime\_type")).count()

# Calculate different types of crimes for Philadelphia

totalCrimesPhiladelphia = crimeDataPhiladelphiaDF.groupBy(lower(col("ucr\_general\_description")).alias("crime\_type")).count()

# Calculate different types of crimes for Dallas

totalCrimesDallas = crimeDataDallasDF.groupBy(lower(col("typeofincident")).alias("crime\_type")).count()

# Calculate different types of crimes for Boston

totalCrimesBoston = crimeDataBostonDF.groupBy(lower(col("offense\_code\_group")).alias("crime\_type")).count()

# Display the results

print("Different types of crimes for New York:")

display(totalCrimesNewYork)

print("Different types of crimes for Los Angeles:")

display(totalCrimesLosAngeles)

print("Different types of crimes for Chicago:")

display(totalCrimesChicago)

print("Different types of crimes for Philadelphia:")

display(totalCrimesPhiladelphia)

print("Different types of crimes for Dallas:")

display(totalCrimesDallas)

print("Different types of crimes for Boston:")

display(totalCrimesBoston)

A screenshot of a computer

Description automatically generated

Question 3

# Filter the DataFrame

robberyLosAngelesDF = crimeDataLosAngelesDF.filter(lower(col("crimeCodeDescription")).contains("robbery"))

robberyPhiladelphiaDF = crimeDataPhiladelphiaDF.filter(lower(col("ucr\_general\_description")).contains("robbery"))

robberyDallasDF = crimeDataDallasDF.filter(lower(col("typeofincident")).contains("robbery"))

# Calculate the total Robbery count

totalRobberyLosAngeles = robberyLosAngelesDF.count()

totalRobberyPhiladelphia = robberyPhiladelphiaDF.count()

totalRobberyDallas = robberyDallasDF.count()

# Display the results

print("Total Robbery count for Los Angeles: ", totalRobberyLosAngeles)

print("Total Robbery count for Philadelphia: ", totalRobberyPhiladelphia)

print("Total Robbery count for Dallas: ", totalRobberyDallas)

A screenshot of a computer

Description automatically generated

Question 4

# TASK: Extract the month from the date column

from pyspark.sql.functions import month

robberiesByMonthLosAngelesDF = robberyLosAngelesDF.withColumn("month", month("timeOccurred"))

robberiesByMonthPhiladelphiaDF = robberyPhiladelphiaDF.withColumn("month", month("dispatch\_date\_time"))

robberiesByMonthDallasDF = robberyDallasDF.withColumn("month", month("startingDateTime"))

# Group by month and count the number of robberies

robberiesByMonthLosAngelesDF = robberiesByMonthLosAngelesDF.groupBy("month").count()

robberiesByMonthPhiladelphiaDF = robberiesByMonthPhiladelphiaDF.groupBy("month").count()

robberiesByMonthDallasDF = robberiesByMonthDallasDF.groupBy("month").count()

# Rename the "count" column to "robberies"

robberiesByMonthLosAngelesDF = robberiesByMonthLosAngelesDF.withColumnRenamed("count", "robberies")

robberiesByMonthPhiladelphiaDF = robberiesByMonthPhiladelphiaDF.withColumnRenamed("count", "robberies")

robberiesByMonthDallasDF = robberiesByMonthDallasDF.withColumnRenamed("count", "robberies")

# Find the month with the highest robbery count for Los Angeles

highestMonthLosAngeles = robberiesByMonthLosAngelesDF.orderBy("count", ascending=False).first()

# Find the month with the lowest robbery count for Los Angeles

lowestMonthLosAngeles = robberiesByMonthLosAngelesDF.orderBy("count").first()

print("Highest robbery count month in Los Angeles:", highestMonthLosAngeles)

print("Lowest robbery count month in Los Angeles:", lowestMonthLosAngeles)

# Find the month with the highest robbery count for Philadelphia

highestMonthPhiladelphia = robberiesByMonthPhiladelphiaDF.orderBy("count", ascending=False).first()

# Find the month with the lowest robbery count for Philadelphia

lowestMonthPhiladelphia = robberiesByMonthPhiladelphiaDF.orderBy("count").first()

print("\nHighest robbery count month in Philadelphia:", highestMonthPhiladelphia)

print("Lowest robbery count month in Philadelphia:", lowestMonthPhiladelphia)

# Find the month with the highest robbery count for Dallas

highestMonthDallas = robberiesByMonthDallasDF.orderBy("count", ascending=False).first()

# Find the month with the lowest robbery count for Dallas

lowestMonthDallas = robberiesByMonthDallasDF.orderBy("count").first()

print("\nHighest robbery count month in Dallas:", highestMonthDallas)

print("Lowest robbery count month in Dallas:", lowestMonthDallas)

A screenshot of a computer code

Description automatically generated

Question 5

# TODO

from pyspark.sql.functions import lit

# Add a new column "city"

robberiesByMonthLosAngelesDF = robberiesByMonthLosAngelesDF.withColumn("city", lit("Los Angeles"))

robberiesByMonthPhiladelphiaDF = robberiesByMonthPhiladelphiaDF.withColumn("city", lit("Philadelphia"))

robberiesByMonthDallasDF = robberiesByMonthDallasDF.withColumn("city", lit("Dallas"))

combinedRobberiesByMonthDF = robberiesByMonthLosAngelesDF.union(robberiesByMonthPhiladelphiaDF).union(robberiesByMonthDallasDF).orderBy("month", "city")

# Find the month with the highest robbery count

highestMonth = combinedRobberiesByMonthDF.orderBy("robberies", ascending=False).first()

# Find the month with the lowest robbery count

lowestMonth = combinedRobberiesByMonthDF.orderBy("robberies").first()

# Display

print("Combined robberies by month:")

display(combinedRobberiesByMonthDF)

print("Month with highest combined robbery count:", highestMonth)

print("Month with lowest combined robbery count:", lowestMonth)

A screenshot of a computer

Description automatically generated

Question 6

# TODO

import matplotlib.pyplot as plt

import numpy as np

combinedRobberiesByMonthDF = combinedRobberiesByMonthDF.orderBy("month", "city")

plot\_data = combinedRobberiesByMonthDF.toPandas()

# Extract unique cities

cities = plot\_data["city"].unique()

# Group the data by month and city

grouped\_data = plot\_data.groupby(["month", "city"]).sum().reset\_index()

# Create a list to store the x coordinates

x\_values = np.arange(1, 13)

# Calculate the width

bar\_width = 0.20

# Create the histogram

plt.figure(figsize=(10, 6))

for i, city in enumerate(cities):

    city\_data = grouped\_data[grouped\_data["city"] == city]

    plt.bar(x\_values + i \* bar\_width, city\_data["robberies"], width=bar\_width, label=city)

# Configure the plot

plt.title("No of robberies per Month for Each City")

plt.xlabel("Month")

plt.ylabel("No of Robberies")

plt.xticks(x\_values + bar\_width, range(1, 13))  # Assuming months are represented as integers from 1 to 12

plt.legend()

plt.grid(axis='y')

# Show the plot

plt.show()

A graph of different colored lines

Description automatically generated with medium confidence

Question 7

# TASK: Load city data from Parquet file

cityDataDF = spark.read.parquet("/mnt/training/City-Data.parquet")

# Extract population values for the desired cities

desired\_cities = ["Los Angeles", "Philadelphia", "Dallas"]

population\_data = cityDataDF.filter(cityDataDF["city"].isin(desired\_cities)).select("city", "estPopulation2016")

# Join population data with grouped data

joined\_data = combinedRobberiesByMonthDF.join(population\_data, combinedRobberiesByMonthDF["city"] == population\_data["city"], "inner")

# Calculate per capita robbery rates

joined\_data = joined\_data.withColumn("robberyRate", joined\_data["robberies"] / joined\_data["estPopulation2016"])

# Create DataFrame for per capita robbery rates

robberyRatesByCityDF = joined\_data.select(combinedRobberiesByMonthDF["city"], "month", "robberies", "estPopulation2016", "robberyRate").orderBy("month","city")

# Display combined DataFrame

print("Per Capita Robbery Rates by City: ")

display(robberyRatesByCityDF)

# Convert the Spark DataFrame to Pandas DataFrame

plot\_data = robberyRatesByCityDF.toPandas()

# Extract unique cities

cities = plot\_data["city"].unique()

# Create a list to store the x coordinates for each city's bars

x\_values = np.arange(1, 13)

# Calculate the width of each bar

bar\_width = 0.20

# Create the histogram

plt.figure(figsize=(10, 6))

for i, city in enumerate(cities):

    city\_data = plot\_data[plot\_data["city"] == city]

    plt.bar(x\_values + i \* bar\_width, city\_data["robberyRate"], width=bar\_width, label=city)

# Configure the plot

plt.title("Robbery Rates per Month for Each City")

plt.xlabel("Month")

plt.ylabel("Robbery Rate (per capita)")

plt.xticks(x\_values + (len(cities) / 2) \* bar\_width, range(1, 13))  # Assuming months are represented as integers from 1 to 12

plt.legend()

plt.grid(axis='y')

# Show the plot

plt.show()

A screenshot of a computer

Description automatically generated

Question 8

# Filter the DataFrame to include only rows related to homicide

homicideNewYorkDF = crimeDataNewYorkDF.filter(lower(col("offenseDescription")).contains("homicide") | lower(col("offenseDescription")).contains("murder"))

homicideBostonDF = crimeDataBostonDF.filter(lower(col("OFFENSE\_CODE\_GROUP")).contains("homicide") | lower(col("OFFENSE\_CODE\_GROUP")).contains("murder"))

# Extract the month from the date column

homicidesByMonthNewYorkDF = homicideNewYorkDF.withColumn("month", month("reportDate"))

homicidesByMonthBostonDF = homicideBostonDF.withColumn("month", col("MONTH"))

# Group by month and count the number of homicides

homicidesByMonthNewYorkDF = homicidesByMonthNewYorkDF.groupBy("month").count()

homicidesByMonthBostonDF = homicidesByMonthBostonDF.groupBy("month").count()

# Rename the "count" column to "homicides"

homicidesByMonthNewYorkDF = homicidesByMonthNewYorkDF.withColumnRenamed("count", "homicides")

homicidesByMonthBostonDF = homicidesByMonthBostonDF.withColumnRenamed("count", "homicides")

# Add a new column "city" to each view to identify the city associated with each row

homicidesByMonthNewYorkDF = homicidesByMonthNewYorkDF.withColumn("city", lit("New York"))

homicidesByMonthBostonDF = homicidesByMonthBostonDF.withColumn("city", lit("Boston"))

# Combine the two views

combinedHomicidesByMonthDF = homicidesByMonthNewYorkDF.union(homicidesByMonthBostonDF).orderBy("month", "city")

# Find the month with the highest combined homicide count

highestMonth = combinedHomicidesByMonthDF.orderBy("homicides", ascending=False).first()

# Find the month with the lowest combined homicide count

lowestMonth = combinedHomicidesByMonthDF.orderBy("homicides").first()

# Display the combined DataFrame

print("Combined homicides by each month:")

display(combinedHomicidesByMonthDF)

print("Month with highest combined homicide count:", highestMonth)

print("Month with lowest combined homicide count:", lowestMonth)

A screenshot of a computer

Description automatically generated

Question 9

cityDataDF = spark.read.parquet("/mnt/training/City-Data.parquet")

# Extract population values

desired\_cities2 = ["New York", "Boston"]

population\_data2 = cityDataDF.filter(cityDataDF["city"].isin(desired\_cities2)).select("city", "estPopulation2016")

# Join population data with grouped data

joined\_data2 = combinedHomicidesByMonthDF.join(population\_data2, combinedHomicidesByMonthDF["city"] == population\_data2["city"], "inner")

# Calculate per capita robbery rates

joined\_data2 = joined\_data2.withColumn("homicideRate", joined\_data2["homicides"] / joined\_data2["estPopulation2016"])

# Create DataFrame

homicideRatesByCityDF = joined\_data2.select(combinedHomicidesByMonthDF["city"], "month", "homicides", "estPopulation2016", "homicideRate").orderBy("month","city")

# Display

print("Homicide Rates by City: ")

display(homicideRatesByCityDF)

# Convert the Spark DataFrame

plot\_data2 = homicideRatesByCityDF.toPandas()

# Extract unique cities

cities2 = plot\_data2["city"].unique()

# Create a list to store the x coordinates

x\_values2 = np.arange(1, 13)

# Calculate the width of each bar

bar\_width2 = 0.20

# Create the histogram

plt.figure(figsize=(10, 6))

for i, city in enumerate(cities2):

    city\_data2 = plot\_data2[plot\_data2["city"] == city]

    plt.bar(x\_values2 + i \* bar\_width2, city\_data2["homicideRate"], width=bar\_width2, label=city)

# Configure the plot

plt.title("Homicide Rates per Month")

plt.xlabel("Month")

plt.ylabel("Homicide Rate per capita")

plt.xticks(x\_values2 + (len(cities) / 2) \* bar\_width2, range(1, 13))

plt.legend()

plt.grid(axis='y')

# Show the plot

plt.show()

A screenshot of a computer

Description automatically generated

Question 10

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

# Create features vector

assembler = VectorAssembler(inputCols=["month"], outputCol="features")

featureVectorDF = assembler.transform(robberiesByMonthLosAngelesDF)

trainingData = featureVectorDF

# Train the linear regression model

lr = LinearRegression(featuresCol="features", labelCol="robberies")

lrModel = lr.fit(trainingData)

# Generate DataFrame and create feature vectors for the months of 2017

months\_2017 = spark.range(1, 13).withColumnRenamed("id", "month")

assembler\_2017 = VectorAssembler(inputCols=["month"], outputCol="features")

features\_2017 = assembler\_2017.transform(months\_2017)

predictions\_2017 = lrModel.transform(features\_2017).select("month", "prediction")

# Display the result

display(predictions\_2017)

A screenshot of a computer

Description automatically generated