**#Databricks Exercise**

**#1. Introduction to Databricks**

**#Task: Creating a Databricks Notebook**

# Read the CSV file into a DataFrame

df1 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/teacher\_data-1.csv")

#Performing DataFrame Operations

# Show the first few rows of the DataFrame

df1.show(5)

# Display the schema of the DataFrame

df1.printSchema()

# Count the number of rows in the DataFrame

row\_count = df1.count()

print(f"Number of rows in the DataFrame: {row\_count}")

# Filter the DataFrame for teachers with more than 10 years of experience

experienced\_teachers = df1.filter(df1["YearsOfExperience"] > 10)

print("experienced teachers")

experienced\_teachers.show()

# Group by subject and calculate the average salary for each subject

avg\_salary = df1.groupBy('Subject').agg({'Salary': 'avg'})

print("Avg salary")

avg\_salary.show()

# Sort the DataFrame by salary in descending order

df1.orderBy(df1["Salary"].desc()).show()

**#2. Setting Up Azure Databricks Workspace and Configuring Clusters**

**#Task: Configuring Clusters**

spark

**SparkSession - hive**

**SparkContext**

[Spark UI](https://community.cloud.databricks.com/?o=4215158048388427#setting/sparkui/0915-120127-7rswl9xc/driver-3838952452065686731)

Version

v3.3.2

Master

local[8]

AppName

Databricks Shell

**#3. Real-Time Data Processing with Databricks**

**#Task: Implementing Databricks for Real-Time Data Processing**

# Read the CSV file into a DataFrame

df2 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_1.csv")

df2.show()

# Define the schema for the streaming data

from pyspark.sql.types import StructType, StructField, StringType, TimestampType, FloatType

schema = StructType([

StructField("event\_time", TimestampType(), True),

StructField("event\_type", StringType(), True),

StructField("user\_id", StringType(), True),

StructField("amount", FloatType(), True)

])

# Read the CSV file as a streaming DataFrame

input\_path = "/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_1.csv"

# Read the file as if it's a stream

df\_stream = spark.readStream \

.option("header", "true") \

.schema(schema) \

.csv(input\_path)

# Display the streaming DataFrame (for debugging purposes)

df\_stream.printSchema()

# Real-time aggregation: Sum amount by event\_type and per minute

schema="event\_time TIMESTAMP, event\_type STRING, user\_id STRING, amount DOUBLE"

streamind\_data=spark.readStream.format("csv").schema(schema).option("header","true").load("dbfs:/FileStore/")

# Real-time aggregation

aggregated\_data=(streamind\_data.groupBy("event\_type").agg({"amount":"sum"}).withColumnRenamed("sum(amount)","total\_amount"))

query = (aggregated\_data.writeStream

.outputMode("complete")

.format("console")

.start())

query.stop()

**#4. Data Exploration and Visualization in Databricks**

**#Task: Visualizing Data in Databricks**

df3 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/Sales\_data\_1.csv")

df3.show()

# Display schema of the DataFrame

df3.printSchema()

# Describe to see summary statistics of numerical columns

df3.describe().show()

# Count the total number of rows

row\_count = df3.count()

print(f"Total rows in the dataset: {row\_count}")

from pyspark.sql.functions import col, sum

# Calculate total sales by category

sales\_by\_category = df3.withColumn("total\_sales", col("price") \* col("quantity")) \

.groupBy("category") \

.agg(sum("total\_sales").alias("total\_sales")) \

.orderBy("total\_sales", ascending=False)

# Display the result as a table

display(sales\_by\_category)

df3.select("price", "quantity").display()

import matplotlib.pyplot as plt

import seaborn as sns

# Convert Spark DataFrame to Pandas for visualization

sales\_by\_category\_pd = sales\_by\_category.toPandas()

# Create a bar plot

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

sns.barplot(x='category', y='total\_sales', data=sales\_by\_category\_pd)

plt.title("Total Sales by Category")

plt.xticks(rotation=45)

plt.show()

**#6. Reading and Writing Data in Databricks**

**#Task: Reading and Writing Data in Various Formats**

# Reading CSV file

df4 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/sales\_data.csv")

df4.show()

# Writing to JSON

df4.write.json("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/sales\_data\_json")

print("Written to json file format")

# Writing to Parquet

df4.write.parquet("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/sales\_data\_parquet")

print("Written to parquet file format")

# Writing to Delta format

df4.write.format("delta").save("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/sales\_data\_delta")

print("Written to delta table format")

**#7. Analyzing and Visualizing Streaming Data with Databricks**

**#Task: Analyzing Streaming Data**

df5 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_1.csv")

# Path to the saved CSV file

streaming\_path = "/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_1.csv"

# Reading the dataset in streaming mode

streaming\_df = (spark.readStream.format("csv").option("header", "true").schema(schema).load(streaming\_path))

# Show streaming schema

streaming\_df.printSchema()

from pyspark.sql.functions import window

# Aggregating data by event type and time window

aggregated\_streaming\_df = streaming\_df \

.groupBy(window("event\_time", "1 minute"), "event\_type") \

.agg({"amount": "sum"}) \

.withColumnRenamed("sum(amount)", "total\_amount")

# Write the results to the console to verify

query = aggregated\_streaming\_df.writeStream \

.outputMode("complete") \

.format("console") \

.start()

query.stop()

**#8. Introduction to Databricks Delta Lake**

**#Task: Using Delta Lake for Data Versioning**

df6 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_1.csv")

#Write DataFrame to Delta format

df6.write.format("delta").save("dbfs:/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_delta")

print("Data written to Delta format")

delta\_table\_path = "/FileStore/shared\_uploads/varshinie.1006@gmail.com/event\_data\_delta"

# Perform updates by loading the Delta table

delta\_table = spark.read.format("delta").load(delta\_table\_path)

# Simulating an update: Changing 'amount' for 'purchase' event types

from pyspark.sql.functions import expr

updated\_data = delta\_table.withColumn("amount", expr("case when event\_type = 'purchase' then amount \* 1.10 else amount end"))

# Overwrite the Delta table with the updated data

updated\_data.write.format("delta").mode("overwrite").save(delta\_table\_path)

updated\_data.show()

# Reading the Delta table as of version 0 (before the update)

df\_version\_0 = spark.read.format("delta").option("versionAsOf", 0).load(delta\_table\_path)

df\_version\_0.show(5)

# Reading the Delta table as of version 1 (after the update)

df\_version\_1 = spark.read.format("delta").option("versionAsOf", 1).load(delta\_table\_path)

df\_version\_1.show(5)

# Optimize the Delta table

spark.sql(f"OPTIMIZE delta.`{delta\_table\_path}`")

# Vacuum the Delta table to delete old data versions

spark.sql(f"VACUUM delta.`{delta\_table\_path}` RETAIN 0 HOURS")

# Query the optimized Delta table

optimized\_df = spark.read.format("delta").load(delta\_table\_path)

optimized\_df.show(5)

**#9. Managed and Unmanaged Tables**

**#Task: Creating Managed and Unmanaged Tables**

# Save the sales data as a managed Delta table

df4.write.format("delta").mode("overwrite").saveAsTable("managed\_sales\_table")

# Define the external location to save the data for unmanaged table

external\_path = "/FileStore/tables/unmanaged\_sales\_data"

# Save the sales data to the external location

df4.write.format("delta").mode("overwrite").save(external\_path)

# Create an unmanaged Delta table pointing to this external location

spark.sql(f"""

CREATE TABLE unmanaged\_sales\_table

USING DELTA

LOCATION '{external\_path}'

""")

# Select the first 5 records from the managed table

spark.sql("SELECT \* FROM managed\_sales\_table LIMIT 5").show()

# Select the first 5 records from the unmanaged table

spark.sql("SELECT \* FROM unmanaged\_sales\_table LIMIT 5").show()

**#10. Views and Temporary Views**

**#Task: Working with Views in Databricks**

# Create a view from the sales data

df4.createOrReplaceTempView("sales\_view")

# Query the view

spark.sql("SELECT \* FROM sales\_view LIMIT 5").show()

# Create a temporary view

df4.createOrReplaceTempView("temp\_sales\_view")

# Query the temporary view

spark.sql("SELECT product, SUM(price) AS total\_sales FROM temp\_sales\_view GROUP BY product").show()

# Create a global temporary view

df4.createOrReplaceGlobalTempView("global\_sales\_view")

# Query the global temporary view

spark.sql("SELECT \* FROM global\_temp.global\_sales\_view LIMIT 5").show()