Experiment 9

Aim: To perform Exploratory data analysis using Apache Spark and Pandas

Theory:

1. **What is Apache Spark and How Does It Work?**

Apache Spark is an open-source, distributed computing framework designed for big data processing and analytics. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Key Features of Apache Spark:

- In-Memory Processing: Spark stores intermediate data in RAM, making it significantly faster than Hadoop MapReduce.
- Lazy Evaluation: Spark optimizes execution by delaying computation until necessary.
- Fault Tolerance: Uses Resilient Distributed Datasets (RDDs) to recover lost data.
- Multiple Language Support: Works with Python (PySpark), Scala, Java, and R.
- Rich Libraries: Includes Spark SQL (structured data), MLlib (machine learning), GraphX (graph processing), and Spark Streaming (real-time data).

How Apache Spark Works?

- Spark applications run as independent processes coordinated by a SparkContext in the driver program.
- The Cluster Manager (e.g., YARN, Mesos, or Spark Standalone) allocates resources.
- Executors run on worker nodes to perform computations and store data.
- Data is partitioned across nodes for parallel processing.

2. **How is Data Exploration Done in Apache Spark?**

Exploratory Data Analysis (EDA) in Spark involves examining datasets to summarize their main characteristics, often using visual methods and statistical summaries.

Steps for EDA in Apache Spark:

1. Loading the Dataset
    a. Read data from CSV, JSON, Parquet, or other formats using spark.read.

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

2. Viewing Data Structure
    a. Check schema (column names and data types) using printSchema().
    b. Display sample records with show().

```
df.printSchema()
df.show(5)
```

3. Basic Statistics
    a. Compute summary statistics (count, mean, stddev, min, max) using describe().

```
df.describe().show()
```

4. Handling Missing Values
    a. Identify null values:

```
from pyspark.sql.functions import col, isnan, when, count
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

    b. Drop or fill missing values:

```
df_clean = df.na.drop()  # Drop rows with nulls
df_filled = df.na.fill(0)  # Fill nulls with 0
```

5. Data Aggregation and Grouping
    a. Group data and compute aggregations:

```
df.groupBy("category").agg({"price": "avg", "quantity": "sum"}).show()
```

6. Data Visualization (Using Pandas Integration)
    a. Convert Spark DataFrame to Pandas DataFrame for visualization:

```
import matplotlib.pyplot as plt
pandas_df = df.toPandas()
pandas_df["price"].hist()
plt.show()
```

7. Correlation Analysis
    a. Compute correlations between numerical columns:

```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="features")
df_vector = assembler.transform(df).select("features")
matrix = Correlation.corr(df_vector, "features").collect()[0][0]
print(matrix.toArray())
```

8. Handling Categorical Data
    a. Use StringIndexer or OneHotEncoder for categorical variables.

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
```

indexed_df = indexer.fit(df).transform(df)

9. Saving Processed Data
    a. Write the cleaned/processed data back to storage:

df.write.parquet("processed_data.parquet")

Conclusion:

In this experiment, we learned about Exploratory Data Analysis (EDA) using Apache Spark and Pandas.

- Spark provides a distributed framework for handling large-scale datasets efficiently.
- Key steps included data loading, schema inspection, missing value handling, statistical summaries, and visualization.
- Pandas integration helped in plotting and further analysis.
- Spark's parallel processing makes it suitable for big data EDA, while Pandas is useful for smaller datasets.