**1.** **Implement Processing JSON and CSV data with PySpark**

PySpark is the Python API for Apache Spark, a powerful distributed computing framework. It allows you to process large-scale data sets in a distributed and parallelized manner. To work with JSON and CSV data using PySpark, you'll need to use the pyspark.sql module.

Here is a basic example of how to process JSON and CSV data with PySpark:

**from pyspark.sql import SparkSession**

**# Initialize a Spark session**

**spark = SparkSession.builder \**

**.appName("JSON and CSV Processing") \**

**.getOrCreate()**

**# JSON Data**

**json\_data\_path = "path/to/json/data.json"**

**json\_data = spark.read.json(json\_data\_path)**

**# Show the JSON schema**

**json\_data.printSchema()**

**# Show the JSON data**

**json\_data.show()**

**# CSV Data**

**csv\_data\_path = "path/to/csv/data.csv"**

**csv\_data = spark.read.csv(csv\_data\_path, header=True, inferSchema=True)**

**# Show the CSV schema**

**csv\_data.printSchema()**

**# Show the CSV data**

**csv\_data.show()**

**# Perform some operations on the data (e.g., join, filter, etc.)**

**# For example, let's join the JSON and CSV data on a common column**

**joined\_data = json\_data.join(csv\_data, "common\_column")**

**# Show the result of the join**

**joined\_data.show()**

**# Perform other operations as needed**

**# Stop the Spark session**

**spark.stop()**

In this example:

1.We create a Spark session using SparkSession.builder.

2. We read JSON data using spark.read.json and CSV data using spark.read.csv. You need to replace the path/to/json/data.json and path/to/csv/data.csv with the actual paths to your JSON and CSV data files.

3. We print the schema of the JSON and CSV data using printSchema

4. We display the first few rows of the JSON and CSV data using show.

5. We perform some operations on the data (e.g., join, filter) as needed for your use case. In this example, we demonstrate a join operation.

6. Finally, we stop the Spark session using spark.stop().

**2.Explain ETL (Extract, Transform, Load) with PySpark**

ETL (Extract, Transform, Load) is a common process in data integration and data warehousing. PySpark, as part of the Apache Spark ecosystem, is a powerful tool for performing ETL tasks at scale. Let's break down each step of the ETL process in the context of PySpark:

1. EXTRACT:

This step involves extracting data from various sources such as databases, files, APIs, or streaming systems. PySpark supports reading data from a variety of sources, including:

**File Formats:** CSV, JSON, Parquet, Avro, ORC, and more.

1. TRANSFORM:

Transformation involves cleaning, aggregating, enriching, or otherwise modifying the extracted data to suit the analysis or storage requirements. PySpark provides a rich set of transformations using the DataFrame API. Common transformations include filtering, grouping, joining, and adding new columns.

1. LOAD:

Loading is the process of writing the transformed data into a target destination, such as a data warehouse, database, or file system. PySpark supports writing data to various output formats and systems.

3.Using Spark SQL – Creating databases, tables Using Spark SQL- Transformations such as Filter, Join, Simple Aggregations, GroupBy

from pyspark.sql import SparkSession

# Create a SparkSession

spark = SparkSession.builder.appName("SparkSQLExample").getOrCreate()

# Create a Database

spark.sql("CREATE DATABASE IF NOT EXISTS mydatabase")

# Use the Database

spark.sql("USE mydatabase")

# Create a Table

data = [("Alice", 25), ("Bob", 30), ("Charlie", 22)]

columns = ["Name", "Age"]

df = spark.createDataFrame(data, columns)

df.createOrReplaceTempView("mytable")

# Querying Data with Spark SQL

# Filtering

result = spark.sql("SELECT \* FROM mytable WHERE Age > 25")

result.show()

# Joining Tables

data2 = [("Alice", "Engineer"), ("Bob", "Doctor"), ("Charlie", "Teacher")]

columns2 = ["Name", "Occupation"]

df2 = spark.createDataFrame(data2, columns2)

df2.createOrReplaceTempView("occupation")

joined\_result = spark.sql("SELECT \* FROM mytable JOIN occupation ON mytable.Name = occupation.Name")

joined\_result.show()

# Simple Aggregations

avg\_age\_result = spark.sql("SELECT AVG(Age) as avg\_age FROM mytable")

avg\_age\_result.show()

# GroupBy

grouped\_result = spark.sql("SELECT Age, COUNT(\*) as count FROM mytable GROUP BY Age")

grouped\_result.show()

# Stop the SparkSession

spark.stop()

