**Pyspark and Sparksql Coding Challenge**

**Name: Vikas Reddy Gorantla**

**Date: 22-11-2024**

**Explain ETL (Extract, Transform, Load) with PySpark(in your own words)**

ETL (Extract, Transform, Load) is a process used in data integration and analytics, and with PySpark, it involves the following steps:

1. **Extract**: PySpark reads data from various sources like databases, flat files, APIs, or streaming services. This step focuses on gathering raw data into a centralized system for processing. PySpark's **DataFrameReader** can handle formats like CSV, JSON, Parquet, and more.
2. **Transform**: In this step, the raw data is cleaned, enriched, and structured for analysis. Using PySpark, you can apply transformations like filtering, joining, aggregations, or complex computations using the **DataFrame API** or **Spark SQL**. This step ensures the data is optimized and meaningful for downstream processes.
3. **Load**: After transformation, the processed data is written to its target destination, such as a data warehouse, database, or storage system like HDFS or cloud services.

A diagram of a process

Description automatically generated

**Using Spark SQL and Pyspark - Transformations such as Filter, Join, Simple Aggregations, GroupBy on the case study dataset**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, regexp\_replace

# Initialize Spark session

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

# Load loan dataset

loan\_df = spark.read.csv("/FileStore/tables/loan.csv", header=True, inferSchema=True)

# Preprocess: Remove commas from "Loan Amount" and "Debt Record" for numerical operations

loan\_df = loan\_df.withColumn("Loan Amount", regexp\_replace(col("Loan Amount"), ",", "").cast("double")) \

                 .withColumn("Debt Record", regexp\_replace(col(" Debt Record"), ",", "").cast("double"))

# Register loan DataFrame as a temporary SQL view

loan\_df.createOrReplaceTempView("loan")

# Show first few rows

loan\_df.show(5)

display(loan\_df)

A screenshot of a document

Description automatically generated

# PySpark

filtered\_df = loan\_df.filter(col("Income") > 50000)

print("Filter: Customers with Income > 50,000 (PySpark)")

filtered\_df.show(5)

# SparkSQL

print("Filter: Customers with Income > 50,000 (SparkSQL)")

spark.sql("SELECT \* FROM loan WHERE Income > 50000").show(5)

A screenshot of a computer

Description automatically generated

# 1. SUM (Total Loan Amount) in pyspark

loan\_df.agg({"Loan Amount": "sum"}).show()

# 1. SUM (Total Loan Amount) in sparksql

print("SUM of Loan Amount (SparkSQL):")

spark.sql("SELECT SUM(`Loan Amount`) AS `Total Loan Amount` FROM loan").show()

A screenshot of a computer

Description automatically generated

# 2. AVG (Average Loan Amount) in pyspark

loan\_df.agg({"Loan Amount": "avg"}).show()

# 2. AVG (Average Loan Amount) in sparksql

print("AVG of Loan Amount (SparkSQL):")

spark.sql("SELECT AVG(`Loan Amount`) AS `Average Loan Amount` FROM loan").show()

A screenshot of a computer

Description automatically generated

# 3. MIN (Minimum Loan Amount) in pyspark

loan\_df.agg({"Loan Amount": "min"}).show()

# 3. MIN (Minimum Loan Amount) in sparksql

print("MIN of Loan Amount (SparkSQL):")

spark.sql("SELECT MIN(`Loan Amount`) AS `Min Loan Amount` FROM loan").show()

A screenshot of a computer screen

Description automatically generated

# 4. MAX (Maximum Loan Amount) in pysaprk

loan\_df.agg({"Loan Amount": "max"}).show()

# 4. MAX (Maximum Loan Amount) in sparksql

print("MAX of Loan Amount (SparkSQL):")

spark.sql("SELECT MAX(`Loan Amount`) AS `Max Loan Amount` FROM loan").show()

A screenshot of a computer

Description automatically generated

# 5. COUNT (Number of Records) in pyspark

loan\_df.agg({"Loan Amount": "count"}).show()

# 5. COUNT (Number of Records) in sparksql

print("COUNT of Records (SparkSQL):")

spark.sql("SELECT COUNT(`Loan Amount`) AS `Record Count` FROM loan").show()

A screenshot of a computer program

Description automatically generated

# 6. MEAN (Mean Loan Amount) in pyspark

loan\_df.agg({"Loan Amount": "mean"}).show()

# 6. MEAN (Mean Loan Amount) in sparksql

print("MEAN of Loan Amount (SparkSQL):")

spark.sql("SELECT AVG(`Loan Amount`) AS `Mean Loan Amount` FROM loan").show()

A close-up of a number

Description automatically generated

# 7. AGG (Multiple Aggregations) in pyspark

from pyspark.sql import functions as F

# Assuming 'loan\_df' is the DataFrame

loan\_df.agg(

    F.sum("Loan Amount").alias("Total Loan Amount"),

    F.avg("Loan Amount").alias("Average Loan Amount"),

    F.min("Loan Amount").alias("Min Loan Amount"),

    F.max("Loan Amount").alias("Max Loan Amount"),

    F.count("Loan Amount").alias("Loan Count")

).show()

# 7. AGG (Multiple Aggregations) in sparksql

print("Multiple Aggregations (SparkSQL):")

spark.sql("""

    SELECT

        SUM(`Loan Amount`) AS `Total Loan Amount`,

        AVG(`Loan Amount`) AS `Average Loan Amount`,

        MIN(`Loan Amount`) AS `Min Loan Amount`,

        MAX(`Loan Amount`) AS `Max Loan Amount`,

        COUNT(`Loan Amount`) AS `Record Count`

    FROM loan

""").show()

A white sheet of paper with black text

Description automatically generated

# Group by Loan Category and sum Loan Amount using Pyspark

loan\_df.groupBy("Loan Category").sum("Loan Amount").show()

# Group by Loan Category and sum Loan Amount using Spark SQL

print("Group By Loan Category and Sum Loan Amount (SparkSQL):")

spark.sql("""

    SELECT

        `Loan Category`,

        SUM(`Loan Amount`) AS `Total Loan Amount`

    FROM loan

    GROUP BY `Loan Category`

""").show()

A white text on a white background

Description automatically generated

**# JOINS (Sample Dataset Preparation)**

# Create a sample DataFrame for risk levels

risk\_data = [

    ("HOUSING", "High Risk"), ("SHOPPING", "Low Risk"),

    ("TRAVELLING", "Medium Risk"), ("GOLD LOAN", "Medium Risk"),

    ("AUTOMOBILE", "High Risk"),]

risk\_df = spark.createDataFrame(risk\_data, ["Loan Category", "Risk Level"])

# Register risk DataFrame as a temporary SQL view

risk\_df.createOrReplaceTempView("risk")

**INNER JOIN**

# PySpark Joins

print("Inner Join (PySpark)")

loan\_df.join(risk\_df, on="Loan Category", how="inner").show(5)

# SparkSQL Joins

print("Inner Join (SparkSQL)")

spark.sql("""

    SELECT l.\*, r.`Risk Level` FROM loan l

    JOIN risk r ON l.`Loan Category` = r.`Loan Category`

""").show(5)

A screenshot of a computer

Description automatically generated

**OUTER JOIN**

print("Outer Join (PySpark)")

loan\_df.join(risk\_df, on="Loan Category", how="outer").show(5)

print("Outer Join (SparkSQL)")

spark.sql("""

    SELECT l.\*, r.`Risk Level`

    FROM loan l

    FULL OUTER JOIN risk r

    ON l.`Loan Category` = r.`Loan Category`

""").show(5)

A screenshot of a document

Description automatically generated

**LEFT JOIN**

print("Left Join (PySpark)")

loan\_df.join(risk\_df, on="Loan Category", how="left").show(5)

print("Left Join (SparkSQL)")

spark.sql("""

    SELECT l.\*, r.`Risk Level`

    FROM loan l

    LEFT JOIN risk r

    ON l.`Loan Category` = r.`Loan Category`

""").show(5)

A screenshot of a computer

Description automatically generated

**RIGHT JOIN**

print("Right Join (PySpark)")

loan\_df.join(risk\_df, on="Loan Category", how="right").show(5)

print("Right Join (SparkSQL)")

spark.sql("""

    SELECT l.\*, r.`Risk Level`

    FROM loan l

    RIGHT JOIN risk r

    ON l.`Loan Category` = r.`Loan Category`

""").show(5)

A screenshot of a document

Description automatically generated