This document outlines the approach and implementation of large-scale data processing using PySpark on the **Credit Card Fraud Detection dataset**..

**1. Objective**

To process, transform, and analyze the dataset efficiently using PySpark. The tasks include:

* Performing DataFrame operations for data transformation.
* Implementing partitioning strategies to enhance performance.
* Computing statistical summaries and aggregations.
* Optimizing performance with Spark configurations.
* Generating data quality reports.

**2. Tools and Technologies**

**2.1 Tools**

* **PySpark**: For distributed data processing.
* **HDFS/S3**: For storage (if handling very large datasets).
* **Jupyter Notebook/Databricks**: For interactive analysis.
* **Pandas**: For smaller summary-level operations (if required).

**2.2 PySpark Setup**

1. Install Spark:

pip install pyspark

1. Initialize SparkSession:

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Credit Card Fraud Detection") \

.config("spark.executor.memory", "4g") \

.config("spark.driver.memory", "2g") \

.config("spark.sql.shuffle.partitions", "200") \

.getOrCreate()

**3. Data Loading**

**3.1 Load the Dataset**

* Load the CSV file into a PySpark DataFrame:

data = spark.read.csv('creditcard.csv', header=True, inferSchema=True)

data.printSchema()

**3.2 Inspect Data**

* Display sample data and column statistics:

data.show(5)

data.describe().show()

**4. Data Transformation Using PySpark**

**4.1 Column Renaming and Data Cleaning**

* Rename columns if necessary and drop duplicates:

from pyspark.sql.functions import col

data = data.withColumnRenamed("Class", "isFraud") \

.dropDuplicates()

**4.2 Feature Engineering**

* Create additional columns (e.g., fraud ratio, log-transformed amounts):

from pyspark.sql.functions import log, when

data = data.withColumn("LogTransactionAmt", log(col("Amount") + 1)) \

.withColumn("FraudLabel", when(col("isFraud") == 1, "Fraud").otherwise("Non-Fraud"))

**5. Data Partitioning**

**5.1 Partition Strategy**

Partition the data based on isFraud to segregate fraudulent and non-fraudulent transactions:

data.write.partitionBy("isFraud").parquet("output\_path/partitioned\_data")

**5.2 Repartition for Performance**

Repartition to balance data processing across nodes:

data = data.repartition(100, "isFraud")

**6. Aggregations and Statistical Summaries**

**6.1 Compute Basic Aggregations**

* Aggregate fraud counts and transaction statistics:

fraud\_summary = data.groupBy("isFraud").agg(

{"Amount": "mean", "Amount": "stddev", "Amount": "max", "Amount": "min"}

)

fraud\_summary.show()

**6.2 Fraud Ratio**

* Compute fraud percentage by transaction type:

total\_count = data.count()

fraud\_ratio = data.groupBy("isFraud").count().withColumn(

"FraudPercentage", (col("count") / total\_count) \* 100

)

fraud\_ratio.show()

**6.3 Correlation Analysis**

* Compute correlation between features and fraud:

correlation = data.stat.corr("Amount", "isFraud")

print(f"Correlation between Amount and Fraud: {correlation}")

**7. Performance Optimization**

**7.1 Spark Configuration**

* Optimize Spark settings:

spark.conf.set("spark.sql.shuffle.partitions", "100") # Default is 200; adjust based on cluster size.

spark.conf.set("spark.executor.memory", "4g")

spark.conf.set("spark.executor.cores", "2")

**7.2 Caching**

* Cache intermediate results for repeated queries:

data.cache()

**7.3 Predicate Pushdown**

* Use filters to reduce data movement:

filtered\_data = data.filter(data.Amount > 1000)

**8. Data Quality Report**

**8.1 Missing Values**

* Check for nulls or missing values:

from pyspark.sql.functions import isnan, isnull

data.select([col(c).isNull().alias(c) for c in data.columns]).show()

**8.2 Data Distribution**

* Generate histograms for numeric columns:

data.select("Amount").summary("count", "mean", "stddev", "min", "25%", "50%", "75%", "max").show()

**8.3 Data Balance**

* Compute class distribution:

class\_distribution = data.groupBy("isFraud").count()

class\_distribution.show()

**8.4 Save Report**

* Save the quality report as a CSV or JSON file:

quality\_report = class\_distribution.toPandas()

quality\_report.to\_csv("data\_quality\_report.csv", index=False)

**9. Visualization**

* Use external libraries like matplotlib or seaborn for visual summaries after converting PySpark DataFrames to Pandas:

import matplotlib.pyplot as plt

from pyspark.sql.functions import col

# Assuming 'data' is your PySpark DataFrame containing the transaction data

class\_distribution = data.groupBy("isFraud").count()

# Convert the PySpark DataFrame to a Pandas DataFrame for plotting

fraud\_counts = class\_distribution.toPandas()

# Create the bar plot

fraud\_counts.plot(kind='bar', x='isFraud', y='count', legend=False)

plt.title("Fraud vs Non-Fraud Transaction Counts")

plt.show()

