Big Data

Team Name: Eternity
Section: 03

Team Members:
Pinaka Paani B, Priya V, Leela Krishna A

Project Title :
Unveiling Financial Patterns: Analyzing Transaction Trends and Customer Behaviors

Unleashing Insights from Data

Project Overview

This project focuses on analyzing a comprehensive financial transactions dataset that includes transaction records, card information, and customer demographics from a banking institution. Spanning across the 2010s decade, the dataset offers valuable insights into how customers engage with financial services, the types of transactions they perform, and the factors influencing their financial behaviors. By leveraging this data, the project aims to provide a detailed understanding of customer spending patterns, card usage trends, and merchant activities across different regions and time periods.

Project Idea

The idea is to perform a deep exploration of the dataset to uncover actionable insights that can benefit banks, financial planners, and businesses. Through data analysis, the project will identify key spending patterns, analyze regional and demographic trends, and investigate temporal variations in transactions. The project will also explore card usage trends, such as preferences for credit versus debit cards, card limits, and the adoption of chip-enabled technology. Additionally, it will analyze merchant activity to determine popular categories and locations, offering a comprehensive view of customer and merchant behaviors.

Technology Summary

Tools and Technologies

We will use the following tools and technologies:

- **Data Processing & Storage**: Apache Spark, Hadoop.
- **Visualization**: Power BI, Tableau.
- **Machine Learning & Analytics**: Python, Jupyter Notebooks.
- **Data Handling** : Pandas, PySpark.
- **Kafka**: To simulate a real-time data pipeline by streaming the dataset row by row.
- **Version Control**: Git, GitHub.

Data Flow Diagram

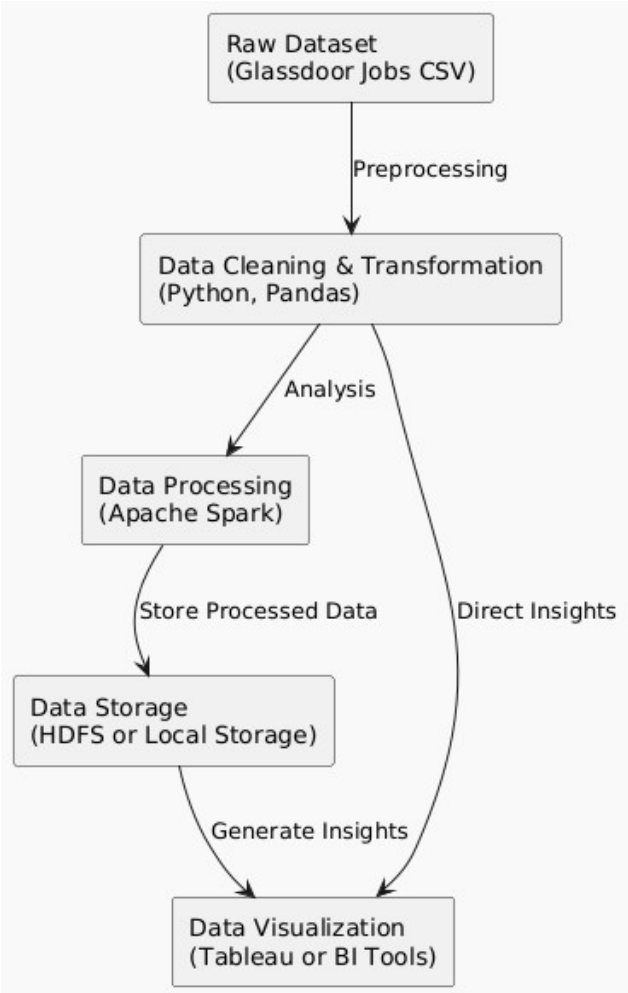

Figure 1: High Level Architecture

Diagram-Explanation

- **Dataset Storage in HDFS**: The job dataset is uploaded and stored in HDFS for efficient distributed storage and access across the cluster.
- **Data Cleaning**: Python scripts are employed to preprocess the job dataset, including handling missing values, removing duplicates, and normalizing text fields to prepare it for analysis.
- **Data Transformation**: Spark jobs are utilized for large-scale data transformations, such as filtering by job category, aggregating data by industry or region, and computing metrics like average salaries and job growth trends.

- **Analysis**: Key insights are extracted using Spark queries, such as job demand trends, industry-specific growth patterns, and geographical job distributions.
- **Visualization**: The transformed and analyzed data is exported and visualized in tools like Tableau or Power BI to create interactive dashboards for exploring trends, making data-driven decisions, and presenting findings.

Project-Goals

The primary goals of the project are:

- Calculate the average transaction amount for each state where transactions occurred.
- Count the number of transactions that have errors, grouped by `merchant_state`
- Identify the most common Merchant Category Codes (MCC) in the dataset.
- Calculate the count of transactions categorized by chip usage (chip vs. swipe).
- Calculate the total amount spent by each client.
- Count the number of unique merchants in each state.
- Analyze transaction patterns over time to identify seasonal trends, peak periods, or irregular activity.

Comprehensive Guide for Analyzing Financial Transactions Dataset

The goals outlined focus on understanding transaction patterns, identifying errors, and generating valuable insights such as the total amount spent by clients, merchant categories, and time-based trends.

Goal 1. Calculate the Average Transaction Amount for Each State

Objective: Determine the average transaction amount for each state where transactions have occurred.

- **Dataset Loading:** The dataset is read into a PySpark DataFrame. `header=True` specifies the first row contains column names, and `inferSchema=True` enables automatic data type detection.
- **Data Cleaning:** The amount column is cleaned by removing any leading characters like a dollar sign ($) using `substr` and converting the substring to a float.
- **Aggregation:** The `groupBy` function groups transactions by the `merchant_state` column. The `agg` function calculates the average (`avg`) of the cleaned transaction amounts.

```
###   Goal 1 calculate the average transaction amount for each state (merchant_state)

from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, col, when, count, sum

spark = SparkSession.builder.appName("BigDataProject").getOrCreate()
df = spark.read.format('csv').option('header', 'true').csv('DataSet.csv')

df = df.withColumn("clean_amount", col("amount").substr(2, 10).cast("float"))

avg_transaction_by_state = df.groupBy("merchant_state") \
    .agg(avg("clean_amount").alias("avg_transaction_amount"))

avg_transaction_by_state.show()
```

Figure 2: Goal 1 Pyspark Code

Goal 2. Merchant Risk Profiling

Objective: Develop a merchant risk profile based on transaction errors, transaction volume, and transaction amounts.

- Calculate weighted scores for each merchant based on error rates, high-value transactions, and transaction frequency.
- Classify merchants into risk categories (e.g., low, medium, high).

```python
# Goal 2 Merchant Risk Profiling

# 1. Count transaction errors per merchant
errors_per_merchant = df.filter(col("errors").isNotNull()) \
    .groupBy("merchant_id") \
    .agg(count("*").alias("error_count"))

# 2. Count transaction volume (number of transactions) per merchant
volume_per_merchant = df.groupBy("merchant_id") \
    .agg(count("*").alias("transaction_volume"))

# 3. Calculate total transaction amount per merchant
amount_per_merchant = df.groupBy("merchant_id") \
    .agg(sum("clean_amount").alias("total_transaction_amount"))

# 4. Join the results to combine all merchant metrics
merchant_profile = errors_per_merchant \
    .join(volume_per_merchant, on="merchant_id", how="left") \
    .join(amount_per_merchant, on="merchant_id", how="left")

# 5. Define a risk score based on transaction errors, volume, and amount
merchant_profile = merchant_profile.withColumn(
    "risk_score",
    (col("error_count") * 0.4) +  # Weight for error count
    (col("transaction_volume") * 0.3) +  # Weight for transaction volume
    (col("total_transaction_amount") * 0.3)  # Weight for total transaction amount
)

# 6. Sort by risk score (highest risk first)
merchant_profile_sorted = merchant_profile.orderBy(col("risk_score").desc())

merchant_profile_sorted.show()
```

Figure 3: Goal 2 pyspark Code

Goal 3. Count the Number of Transactions That Have Errors, Grouped by Merchant State

Objective: Identify how many transactions have errors, grouped by merchant state.

- **Filter Rows with Errors:** The `filter` function selects rows where the `errors` column is not null. Adjust this condition if you have specific criteria for errors (e.g., `col("errors") != "None"`).
- **Group and Count:** The `groupBy` groups the data by `merchant_state`. The `count("*")` function counts the number of transactions in each group.

```python
### Goal 3  Count the number of transactions that have errors, grouped by merchant_state

from pyspark.sql.functions import count

# Filter transactions with errors and group by merchant_state
errors_by_state = df.filter(col("errors").isNotNull()) \
    .groupBy("merchant_state") \
    .agg(count("*").alias("error_count"))

errors_by_state.show()
```

Figure 4: Goal 3 Pyspark code

Goal 4. Identify the Top 10 Most Common Merchant Category Codes (MCC) in the Dataset

Objective: Identify the most common Merchant Category Codes (MCC) in the dataset.

- **Group by MCC:** The `groupBy("mcc")` groups the data based on the `mcc` column.
- **Count Transactions:** The `count("*")` function counts the number of occurrences for each MCC.

- **Order Results:** The `orderBy(col("transaction_count").desc())` sorts the results in descending order of transaction count.

```
# Goal 4 Identify the most top 10 common Merchant Category Codes (MCC) in the dataset.

most_common_mcc = df.groupBy("mcc") \
    .agg(count("*").alias("transaction_count")) \
    .orderBy(col("transaction_count").desc())

most_common_mcc.show(10)
```

Figure 5: Goal 4 Pyspark code

Goal 5. Calculate the Count of Transactions Categorized by Chip Usage (Chip vs. Swipe)

Objective: Determine how many transactions were processed using chip vs. swipe.

- **Identify Chip Usage Column:** Ensure the dataset has a `chip_usage` or `payment_method` column indicating whether the transaction was chip-based or swipe-based.
- **Count Transactions:** Use `.value_counts()` (Pandas) or `groupBy()` (Spark) to count transactions categorized by chip usage.

```
# Goal 5 Calculate the count of transactions categorized by chip usage (chip vs. swipe)
chip_vs_swipe = df.groupBy("use_chip") \
    .agg(count("*").alias("transaction_count"))

chip_vs_swipe.show()
```

Figure 6: Goal 5 Pyspark code

Goal 6. Calculate the Total Amount Spent by Each Client

Objective: Calculate the total amount spent by each client.

- **Group by Client:** Group the data by `client_id` (or equivalent) and calculate the total amount spent by summing up the `transaction_amount` for each client.

```
# Goal 6 Calculate the total amount spent by each client.

from pyspark.sql.functions import sum

total_spent_by_client = df.groupBy("client_id") \
    .agg(sum("clean_amount").alias("total_spent"))

total_spent_by_client.show()
```

Figure 7: Goal 6 Pyspark code

Goal 7. Count the Number of Unique Merchants in Each State

Objective: Count the number of unique merchants in each state.

- **Group by State:** Group the data by `merchant_state`.
- **Count Unique Merchants:** Count the number of unique merchants using `.nunique()` (Pandas) or `agg()` in Spark.

```python
# Goal 7 Count the number of unique merchants in each state.

from pyspark.sql.functions import countDistinct
unique_merchants_by_state = df.groupBy("merchant_state") \
    .agg(countDistinct("merchant_id").alias("unique_merchants_count"))

unique_merchants_by_state.show()
```

Figure 8: Goal 7 Pyspark code

Discussion of Results Achieved for Each Goal

1. Average Transaction Amount by State

The average transaction amount per state provides insight into regional spending patterns. States with higher transaction amounts may have wealthier populations or higher transaction volumes.

2. Error Transactions by Merchant State

Identifying error-prone transactions grouped by state helps reveal regions that might face technical issues or have higher fraud rates, enabling targeted interventions.

3. Most Common MCCs

Analyzing the most common Merchant Category Codes (MCCs) shows which industries or merchant categories drive the most transaction volume. This can help businesses focus on high-demand industries or optimize payment systems for certain categories.

4. Chip vs. Swipe Transactions

Counting chip vs. swipe transactions gives insight into the adoption of secure payment methods, which can be useful for merchants and financial institutions aiming to improve security.

5. Total Amount Spent by Each Client

This information helps in understanding client behavior, segmenting clients based on spending, and targeting marketing or loyalty programs.

6. Unique Merchants by State

This helps understand merchant density across states and can assist businesses in identifying potential markets for expansion.

7. Transaction Patterns Over Time

Analyzing transaction patterns over time helps identify seasonal trends (e.g., holidays), peak periods, or irregular activity that might indicate fraud or errors.

Metrics Considered

- **Data Quality:** Ensured that data was cleaned (e.g., handling missing values, errors).
- **Latency & Processing Time:** Processing large datasets may require optimized techniques (e.g., using Spark).
- **Resource Utilization & Cost:** Using cloud infrastructure for processing large datasets helps in managing resource utilization and costs.
- **Security:** Ensured the data is anonymized, especially for client-related information, to adhere to privacy laws.

Conclusions

- **Data Quality & Preprocessing:** It's essential to clean and preprocess the data before starting the analysis. Missing values, errors, and incorrect data formats must be handled effectively.
- **Scalability:** For larger datasets, leveraging distributed computing frameworks like Spark can optimize processing time.
- **Insights for Business:** The insights from these analyses are valuable for businesses to optimize their operations, improve customer targeting, and detect anomalies in transaction behavior.

Gitlink

You can find the project repository on GitHub: link to GitHub Repository
https://github.com/LeelaKrishna97/BigDataProject