information technology institute

Project Team Members:

Abdulrahman Mahmoud
Hazem Gamal Abdallah
Ahmed Magdy Ghazy
Sheref Hamdy

Under Supervision of:
Eng. Ibrahim Mohamed

october 2024

ACKNOWLEDGMENT

We would like to express our sincere gratitude to all those who supported and guided us throughout the development of this **Real-Time Fraud Detection System** project. Without their invaluable assistance and insight, this project would not have been possible.

First and foremost, we would like to thank our **supervisors and academic mentors** for their continuous guidance, encouragement, and expert knowledge. Their insightful advice, critical feedback, and unwavering support helped shape the direction of this project. Their expertise in the areas of big data technologies, machine learning, and cloud infrastructure provided us with a solid foundation on which to build our project.

We are especially grateful to our **institution and faculty members** for providing the necessary resources and learning environment that facilitated our research and development. The access to cutting-edge tools and technologies, such as **Google Cloud Platform (GCP)**, **Apache Kafka**, **Apache Spark**, and **machine learning frameworks**, allowed us to work on real-world problems and develop innovative solutions. The practical experience we gained through this project has significantly enhanced our skills and knowledge.

We would also like to extend our thanks to the **technical support teams** at various stages of the project, who helped resolve technical issues and provided timely assistance with cloud services, data processing, and machine learning tools. Their prompt responses and solutions played a crucial role in keeping the project on track.

Our gratitude goes to the **fraud analysts and industry experts** who provided valuable insights into the current challenges in financial fraud detection. Their expertise helped us understand the complexity of real-time fraud detection systems and inspired us to develop a scalable and efficient solution that addresses these challenges.

We are also thankful to our **peers and colleagues** for their constructive feedback and collaborative discussions throughout the course of this project. Their ideas and suggestions helped us refine our approach and continuously improve the project.

Finally, we would like to acknowledge the unconditional support of our **families and friends**. Their patience, understanding, and encouragement allowed us to stay focused and motivated during the challenging phases of the project. Without their emotional support, this project would not have been possible.

This project is the result of the collective efforts of many individuals and teams. We are deeply grateful to all those who contributed in various ways, and we hope that the work we have done will contribute to the ongoing efforts in fighting financial fraud and improving security in the digital economy.

Thank you all.

Thanks for all your encouragement!

Introduction

Fraud detection systems are critical tools in the financial industry, used to identify and prevent fraudulent activities in financial transactions. With the increasing adoption of online banking, digital payments, and e-commerce, the volume of transactions has grown exponentially, leading to a rise in fraudulent activities such as credit card fraud, identity theft, and unauthorized transactions. Traditional fraud detection methods rely heavily on manual review processes and static rules-based systems, which are not sufficient for handling the complexity and scale of modern financial systems.

A **Real-Time Fraud Detection System** utilizes big data technologies and machine learning to analyze vast amounts of transaction data in real time. These systems can detect patterns and anomalies that indicate fraudulent activities by continuously monitoring transaction flows. The goal of this project is to develop a scalable, automated fraud detection system that ingests real-time transaction data, processes it efficiently, applies machine learning models to detect potential fraud, and provides actionable insights to prevent financial losses.

This system is designed to detect fraud immediately after it occurs, enabling organizations to take preventative measures. It can handle high volumes of data, adapt to new fraud patterns, and ensure the accuracy and speed required to protect users and businesses.

Abstract

The **Real-Time Fraud Detection System** is designed to analyze financial transactions as they occur and flag suspicious activities for further review. By leveraging a combination of **Apache Kafka** for data ingestion, **Apache Spark** for data processing, and **machine learning models** for fraud detection, this system is capable of processing large volumes of transactions in real time. The project utilizes **Google Cloud Platform (GCP)** for scalable storage, processing, and reporting.

The core of the system is a machine learning pipeline that identifies fraudulent behavior by examining transaction patterns. These models are trained on historical data to detect anomalies such as unusual transaction amounts, unfamiliar merchant locations, or rapid consecutive transactions. Once a potential fraud is detected, the system triggers alerts and provides detailed reports for fraud analysts to investigate.

The system ensures real-time detection by processing data as it is ingested, reducing latency, and improving detection accuracy. The use of cloud infrastructure allows for automatic scaling, ensuring that the system can handle increases in transaction volumes without performance degradation.

Motivation

Fraud detection has become a significant concern in today's digital economy. As financial transactions shift from traditional in-person payments to online and mobile platforms, the number and complexity of fraudulent activities have increased. Businesses are losing billions of dollars annually due to fraud, and the methods used by fraudsters are becoming increasingly sophisticated.

One of the key motivators for this project is the **growing demand for real-time fraud detection**. Traditional systems that rely on batch processing and manual review cannot keep pace with the speed at which transactions occur in today's digital world. Delays in detecting fraud can lead to substantial financial losses, damage to a company's reputation, and loss of customer trust.

Additionally, the **complexity of modern fraud** demands more advanced solutions. Fraudsters often employ techniques such as synthetic identities, account takeovers, and automated attacks that bypass simple rule-based detection systems. Machine learning models can recognize these complex patterns by learning from historical data and continuously adapting to new fraud behaviors.

The **motivation** for this project stems from the need for a solution that:

1. **Detects fraud in real time**, as transactions are processed.
2. **Scales to accommodate increasing volumes of transaction data**.
3. **Adapts to emerging fraud patterns**, ensuring that new fraud schemes can be detected and mitigated.

By addressing these challenges, the Real-Time Fraud Detection System provides a proactive approach to fraud prevention, reducing financial losses and enhancing security.

Problem Statement

The financial industry faces several challenges when it comes to detecting and preventing fraud. As digital transactions continue to rise, fraudsters are exploiting vulnerabilities in payment systems, leading to financial losses for businesses and customers. Current fraud detection systems often suffer from several limitations, including:

1. **Delayed Detection**: Many existing fraud detection systems operate in batch mode, analyzing transactions after they have occurred. This delay can result in fraud going unnoticed until it is too late to prevent financial losses.
2. **Manual Reviews**: Fraud detection in many organizations still relies on manual reviews or rule-based systems that flag transactions for human inspection. These processes are time-consuming, prone to errors, and struggle to handle large transaction volumes.
3. **Static Rules**: Traditional fraud detection systems rely on fixed rules, such as flagging transactions above a certain amount or originating from unfamiliar locations. These static rules are easily bypassed by fraudsters who adapt their tactics to avoid detection.
4. **Scalability**: As transaction volumes increase, many fraud detection systems struggle to scale, leading to performance bottlenecks and reduced detection accuracy.
5. **Emerging Fraud Tactics**: Fraudsters continuously evolve their methods, utilizing sophisticated techniques such as account takeovers, synthetic identities, and AI-driven attacks. Traditional detection systems are often ill-equipped to detect these emerging threats.

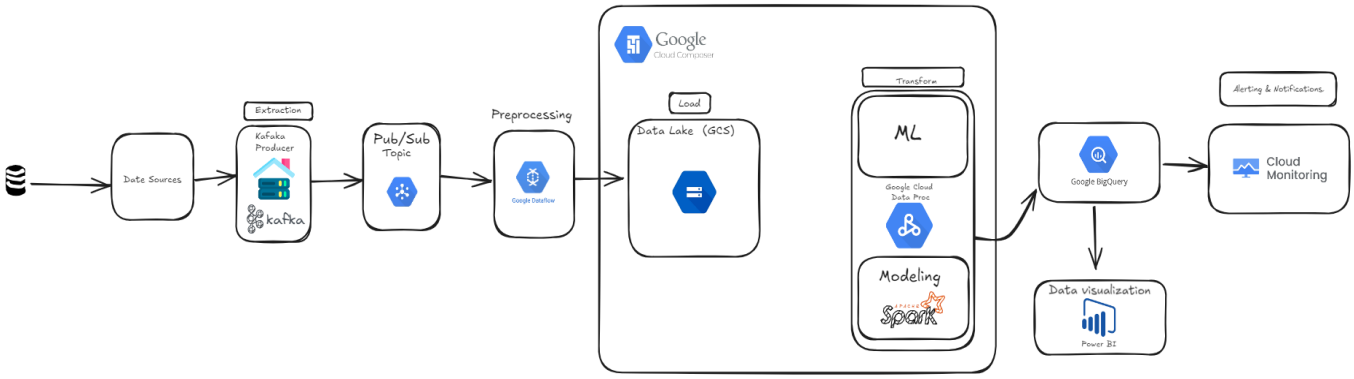
The Real-Time Fraud Detection System aims to address these problems by:

- Ingesting and analyzing transactions in real time providing immediate detection of suspicious activities.
- Leveraging machine learning models to detect complex fraud patterns and reduce false positives.
- Utilizing cloud infrastructure for scalability, ensuring the system can handle large transaction volumes.
- Continuously retraining models to adapt to new fraud tactics and evolving threats.

The system will provide businesses with a proactive approach to fraud prevention, reducing the financial and reputational damage caused by fraud.

Project Architecture

The Real-Time Fraud Detection System architecture is designed to provide a scalable, efficient, and automated solution for detecting fraudulent activities in real time. The system leverages cloud-native technologies, big data processing frameworks, and machine learning to monitor transaction data, detect anomalies, and prevent financial fraud.



Key Components:

1. Data Sources:

- The system ingests transaction data from various sources such as credit card terminals**,** banking systems**,** and online payment platforms**.** These sources generate high-velocity streams of transactional data, which are the foundation for fraud detection.

2. Data Ingestion and Streaming:

- Apache Kafka acts as the real-time data ingestion layer. Kafka captures transaction data from the sources and streams it into the system.
    - Kafka Producers**:** These components are responsible for sending data from the transaction sources to the Kafka topics in real time.
    - The high throughput and fault tolerance of Kafka ensure that large volumes of transaction data are ingested efficiently.

3. Pub/Sub Topic:

- The ingested data from Kafka is streamed into Google Pub/Sub for event-driven processing. Pub/Sub allows the system to decouple data producers from consumers, enabling real-time messaging and event notification across the system.

4. Data Preprocessing:

- Google Dataflow is used to preprocess the transaction data in real time. This preprocessing includes cleaning, filtering, and transforming the raw data before it is passed on for analysis.
    - Dataflow ensures that the data is properly structured and formatted, preparing it for subsequent stages of processing and machine learning.
    - This step is critical for ensuring data quality, as any inconsistencies or errors in the raw data could compromise the accuracy of fraud detection.

5. Data Lake:

- Preprocessed data is stored in a Data Lake **on** Google Cloud Storage (GCS)**.** The data lake serves as a centralized repository for both raw and processed data, allowing for scalable storage and easy access.
    - GCS provides high durability and scalability, allowing the system to handle growing volumes of transaction data over time.
    - The data stored here will be used by subsequent stages of the pipeline, such as machine learning and analysis.

6. Data Processing and Machine Learning:

- Google Cloud Dataproc orchestrates the data transformation and machine learning tasks.
    - Apache Spark**:** Spark is used for distributed data processing and modeling. It transforms the preprocessed data and applies machine learning models to detect fraudulent patterns in transactions.
    - Machine Learning (ML)**:** The system leverages pre-trained machine learning models to detect anomalies and identify potential fraud. These models analyze transaction attributes such as transaction amount, time, location, and customer behavior to flag suspicious activities.
    - The machine learning models are continuously retrained on historical data to improve their accuracy and adapt to new fraud patterns.

7. Data Warehouse (Google BigQuery):

- The results of the data processing and machine learning models are loaded into Google BigQuery, which serves as the data warehouse.
    - BigQuery stores aggregated and summarized data that can be used for real-time analysis, reporting, and visualization.
    - This data includes detailed records of flagged transactions, risk scores, and fraud detection metrics, which are essential for further analysis and decision-making.

8. Visualization and Reporting:

- **Power BI** is used for visualizing the fraud detection insights and generating interactive dashboards.
    - The visualization layer allows fraud analysts to monitor transaction trends, detection accuracy, and fraud rates in real time.
    - These dashboards provide actionable insights that enable organizations to respond to fraud incidents quickly and effectively.

9. Monitoring and Alerts:

- **Google Cloud Monitoring** is integrated to track the health and performance of the system. It monitors key metrics such as transaction volume, processing time, and model performance.
    - **Alerts and Notifications**: The system is configured to send alerts when certain thresholds are reached, such as when a high-risk transaction is detected or when the system experiences performance issues.

Scalability and Automation**:**

- The entire pipeline is orchestrated by Google Cloud Composer (Apache Airflow)**,** which schedules and automates the various stages of the system.
    - **Auto-scaling**: The cloud infrastructure automatically scales to handle varying loads of transaction data, ensuring the system can handle peak transaction times without performance degradation.

This flow represents the seamless movement of data from ingestion through preprocessing, machine learning, and finally reporting and visualization. Each component in the architecture is designed for real-time performance, scalability, and reliability, ensuring that fraud detection happens as transactions are processed.

Project Phases

1. Identify the Problem

The first step is to clearly define the problem that the project aims to solve. In the context of financial transactions, fraud has become a significant concern due to the increasing digitalization of payments. Here are the key issues that were identified:

- **Rise in Digital Payments**: With the increase in online banking, e-commerce, and mobile payments, the number of transactions has skyrocketed. This has provided more opportunities for fraudsters to exploit vulnerabilities in payment systems.
- **Traditional Fraud Detection Limitations**: Many organizations still rely on traditional fraud detection systems that use static rules to flag suspicious activities. These systems often result in delays in detecting fraud, meaning that fraudulent transactions may be completed before they are flagged.
- **High False Positive Rates**: Rule-based systems are notorious for generating a high number of false positives. For example, a simple rule might flag a transaction as suspicious if it exceeds a certain threshold or originates from an unfamiliar location. However, these rules often trigger unnecessary alerts, wasting resources and time for fraud analysts.
- **Evolving Nature of Fraud**: Fraudsters are constantly evolving their tactics, using more sophisticated techniques like synthetic identities, account takeovers, and automated fraud attempts. Static rules cannot keep up with these changes, making it difficult to detect new types of fraud.
- **Scalability Challenges**: As transaction volumes increase, traditional systems struggle to scale. The challenge is to create a fraud detection system that can handle the increasing load without sacrificing performance.

In conclusion, the key problem is the inability of traditional fraud detection systems to provide real-time**,** scalable, and accurate fraud detection, especially as fraud patterns evolve.

2. Research the Problem

The second step involves conducting in-depth research to better understand the problem, explore current solutions, and identify possible alternatives. Here are the key research areas:

- **Current Fraud Detection Methods**:
    - Traditional fraud detection methods primarily use static rules or manual reviews. For example, systems may flag transactions based on pre-defined thresholds (e.g., transactions above $1,000 from a foreign

country). However, these systems are slow to respond to new fraud techniques and often generate too many false positives.

- **Limitations of Manual Reviews**:
  - In many financial institutions, fraud detection still relies heavily on manual review processes. Human analysts are required to review flagged transactions, which is not scalable and leads to delayed detection.
- **Advantages of Machine Learning in Fraud Detection**:
  - Unlike static rule-based systems, machine learning models can detect complex fraud patterns by analyzing a wide range of features, such as transaction amount, location, time, and customer behavior.
  - **Supervised Learning**: Models can be trained on labeled data (i.e., past transactions marked as "fraud" or "not fraud") to predict fraud in future transactions.
  - **Unsupervised Learning**: In cases where fraud labels are unavailable, unsupervised techniques like clustering and anomaly detection can be used to identify outliers or unusual patterns in the data.
- **Scalability of Big Data Technologies**:
  - **Apache Kafka** provides a scalable solution for ingesting high volumes of transaction data in real time.
  - **Apache Spark** allows distributed processing, enabling the system to analyze and process large datasets efficiently.
  - **Google Cloud Platform (GCP)** offers scalable storage (e.g., Google Cloud Storage) and processing (e.g., Google Dataproc) to handle the growing volume of transactions.
- **Fraud Patterns and Features**:
  - Research into common fraud patterns, such as frequent small transactions, transactions from multiple locations in a short time, or usage of stolen card information, is essential to define the features that will be used in the machine learning model.

---

3. Formulate a Hypothesis

After thoroughly researching the problem, the next step is to formulate a hypothesis. This hypothesis should propose a potential solution to the identified problem.

Hypothesis: "A real-time fraud detection system that uses machine learning models, big data processing frameworks, and scalable cloud infrastructure will significantly improve the accuracy and speed of fraud detection, reduce false positives, and adapt to emerging fraud patterns."

Key Assumptions of the Hypothesis:

1. Machine learning models trained on historical transaction data can identify complex fraud patterns better than static rule-based systems.
2. Big data technologies such as Apache Kafka and Apache Spark can handle the large volume of transactions and provide real-time analysis.
3. The system will be able to automatically scale as transaction volumes increase, ensuring consistent performance.
4. The system will be able **to** reduce false positives by learning from the actual fraud cases in the training data.

This hypothesis establishes the basis for the project's experimental phase.

4. Conduct an Experiment

This phase involves building and deploying the Real-Time Fraud Detection System to test the hypothesis. The goal is to implement the system in a real-world environment and collect data to evaluate its performance.

Step 1: Data Ingestion

- Set up Apache Kafka to ingest transaction data from multiple sources, such as credit card terminals, online payment gateways, and banking systems.
    - Kafka is used to handle the large volume of incoming transactions in real time. Each transaction is represented as a Kafka message, which is streamed into the system.

Step 2: Preprocessing

- Google Dataflow is used to clean and preprocess the transaction data. This involves removing duplicates, handling missing values, and formatting the data so that it can be used in machine learning models.
    - For example, transactions may be filtered to remove those with incomplete information, or certain features (such as transaction amount and location) may be transformed to standardize values.

Step 3: Machine Learning Model

- A pre-trained machine learning model (using algorithms such as random forests**,** gradient boosting**, or** deep learning) is applied to detect fraudulent activities.
    - The model is trained on historical transaction data, where each transaction is labeled as “fraud” or “not fraud.”

- Once trained, the model is deployed to predict the likelihood of fraud for each incoming transaction in real time.
- Key features considered by the model may include transaction amount, location, merchant type, time of day, and user history.

Step 4: Data Storage

- The preprocessed and processed data are stored in Google Cloud Storage (GCS) as a data lake, allowing for scalable and secure storage.
  - Google BigQuery is used to store summarized and aggregated data, enabling real-time querying and reporting.

Step 5: Experimentation

- The system continuously monitors and processes transaction data to detect fraud. The key metrics evaluated during the experiment include:
  - **Accuracy**: The percentage of fraud cases correctly identified by the system.
  - **False Positive Rate**: The number of legitimate transactions that were incorrectly flagged as fraud.
  - **Real-time Performance**: The system's ability to detect fraud within milliseconds of the transaction being processed.

---

5. Reach a Conclusion

After conducting the experiment, the final step is to analyze the results and determine whether the system meets the expectations set forth in the hypothesis.

Key Questions to Answer:

1. **Has the accuracy improved?**
   - Compare the fraud detection accuracy of the new system with the baseline accuracy of traditional systems. The goal is to achieve higher accuracy, meaning that more fraud cases are correctly detected.
2. **Are false positives reduced?**
   - Examine whether the machine learning model has successfully reduced the number of false positives, ensuring that fewer legitimate transactions are incorrectly flagged as fraud.
3. **Does the system detect fraud in real time?**

- Measure the latency between the time a transaction occurs and the time the system flags it as fraud. The system must operate within strict time constraints to ensure that fraud can be prevented before the transaction is completed.

4. **Can the system scale?**
    - Assess whether the system can handle increasing transaction volumes without degrading performance. The use of cloud infrastructure and distributed processing should allow the system to scale automatically.

Conclusion: Based on the performance metrics, the team can either confirm the hypothesis, adjust the system to improve its performance, or propose new approaches for future experiments. If the hypothesis is confirmed, the Real-Time Fraud Detection System will be deployed in a production environment, providing financial institutions with a scalable, accurate, and real-time fraud detection solution.

Literature Review

Fraud detection has been a topic of research for several decades, especially with the advent of digital payments and e-commerce. Numerous methods and technologies have been explored to prevent financial fraud, including rule-based systems, anomaly detection, machine learning, and hybrid systems. Here's a detailed look into the existing literature and techniques used for fraud detection.

1. Rule-Based Fraud Detection

One of the earliest approaches to fraud detection involves the use of rule-based systems. These systems flag suspicious transactions based on predefined rules, such as:

- Transactions exceeding a specific amount.
- Transactions originating from unfamiliar locations.
- A sudden increase in transaction volume from a single account.

Limitations:

- **Static Rules**: The fraudsters can easily adapt their behavior to evade detection by understanding these static rules.
- **High False Positives**: These systems tend to generate a high number of false positives, where legitimate transactions are flagged as suspicious.
- **Difficulty in Adapting to Emerging Fraud**: These systems struggle to handle new, evolving fraud patterns that do not match pre-defined rules.

2. Anomaly Detection

Anomaly detection techniques are often used in fraud detection to identify transactions that deviate from normal behavior. This involves building a profile of legitimate transactions and flagging those that differ significantly from these profiles.

Challenges:

- **Behavioral Variability**: Customer behaviors change over time, making it difficult to define "normal" transaction patterns.
- **Complexity in Real-Time Detection**: Anomaly detection systems can be computationally expensive and may not work well in real-time scenarios.

3. Machine Learning in Fraud Detection

Machine learning has become increasingly popular in fraud detection due to its ability to identify complex patterns and adapt to new fraud behaviors. Models such as logistic regression**,** decision trees, random forests**,** support vector machines (SVMs)**,** and deep learning have been widely used in recent research.

Advantages:

- **Pattern Recognition**: Machine learning models can detect complex fraud patterns that rule-based systems miss.
- **Adaptability**: The models can learn from new fraud cases and continuously improve their detection capabilities.
- **Reduction in False Positives**: Machine learning models can significantly reduce the number of false positives by learning from the data rather than relying on static rules.

Challenges:

- **Data Availability**: Machine learning models require large amounts of labeled data to train accurately.
- **Real-Time Processing**: Implementing machine learning in a real-time fraud detection system requires significant computational power and efficiency to ensure fast processing.

4. Hybrid Systems

Recent advancements in fraud detection involve combining rule-based systems with machine learning models. These hybrid systems use predefined rules to filter out obvious cases, while more complex cases are processed by machine learning models.

Benefits:

- **Efficiency**: Hybrid systems can process large amounts of data efficiently by combining the strengths of both rule-based and machine learning systems.
- **Improved Detection Rates**: By combining multiple detection methods, hybrid systems can achieve higher detection rates and lower false positive rates.

Conclusion from Literature Review: The literature suggests that machine learning provides a more flexible and accurate approach to fraud detection compared to traditional rule-based systems. However, for a comprehensive solution, hybrid systems that leverage both rule-based and machine learning approaches are considered the most

effective. This project aims to build a Real-Time Fraud Detection System that incorporates the lessons learned from the literature by leveraging machine learning, scalable big data technologies, and real-time processing capabilities.

Analysis and Requirements

In this section, we define the requirements and specifications of the **Real-Time Fraud Detection System**, which are essential for the development and implementation phases of the project.

1. Requirements

1.1 Functional Requirements

Functional requirements define the specific behavior or functionality of the fraud detection system. These are essential to ensure that the system operates as intended and meets user needs. The key functional requirements are:

1. **Data Ingestion**:
    - The system must be able to ingest real-time transaction data from multiple sources, including credit card terminals, online payment gateways, and banking systems.
    - Data must be ingested with minimal delay to ensure real-time fraud detection.
2. **Data Preprocessing**:
    - The system must preprocess the raw transaction data by cleaning, formatting, and filtering it.
    - Any missing or corrupted data should be handled appropriately to ensure data quality.
3. **Fraud Detection**:
    - The system must apply a pre-trained machine learning model to detect potentially fraudulent transactions.
    - The system should assign a fraud score to each transaction, representing the likelihood of it being fraudulent.
    - The system must be capable of real-time fraud detection with minimal latency.
4. **Data Storage**:
    - Transaction data must be stored in a scalable and durable data lake for further analysis.
    - Aggregated data should be stored in a data warehouse to support reporting and visualization.
5. **Alerts and Notifications**:
    - If a fraudulent transaction is detected, the system should generate alerts for fraud analysts.
    - Alerts must be sent in real-time, enabling fraud analysts to take immediate action.
6. **Data Visualization**:
    - The system must provide a visualization dashboard for real-time monitoring of fraud detection metrics.

- Fraud analysts should be able to view detailed reports on flagged transactions, fraud scores, and historical data.

1.2 Non-Functional Requirements

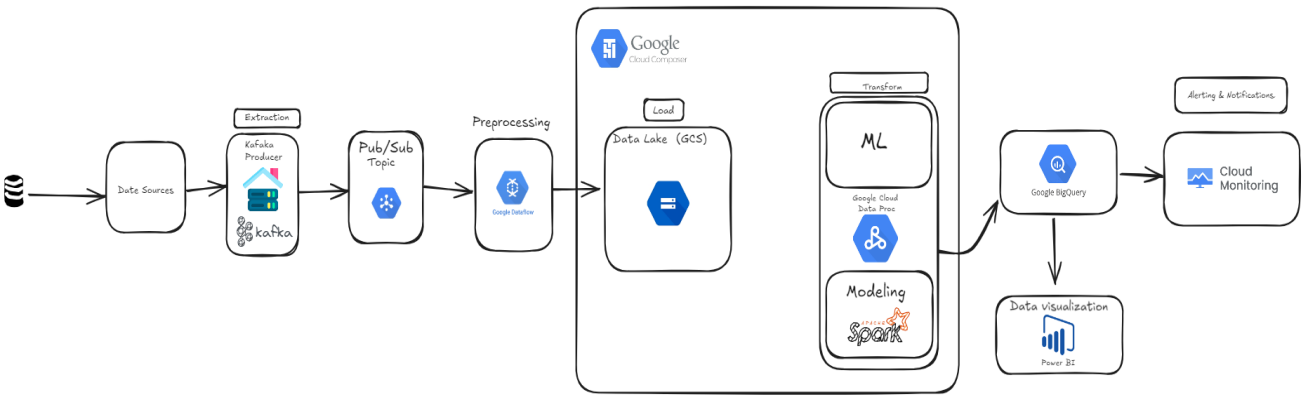
Non-functional requirements describe the performance, scalability, and reliability aspects of the system. These requirements ensure that the system is scalable, secure, and performs well under various conditions.

1. **Scalability**:
   - The system must scale automatically to accommodate increases in transaction volume without degrading performance.
   - The system should be able to process millions of transactions per second with minimal latency.
2. **Performance**:
   - The system must perform real-time fraud detection within milliseconds of receiving a transaction.
   - The system should have low latency in both data ingestion and detection to prevent delays in fraud detection.
3. **Reliability**:
   - The system must be fault-tolerant and able to recover from failures with minimal downtime.
   - Redundancy and backup mechanisms should be in place to ensure data is not lost during system failures.
4. **Security**:
   - The system must ensure that sensitive customer information (e.g., credit card data, transaction details) is encrypted during transmission and storage.
   - Role-based access controls must be implemented to restrict access to sensitive data and features.

2. Functional Requirement Specification (FRS)

Below is **a** Functional Requirement Specification (FRS) diagram that captures the flow and interactions within the system:

FRS Diagram



- Data Ingestion: Transaction data from various sources is collected via Apache Kafka.
- Preprocessing: Google Dataflow cleans and transforms the raw data.
- Fraud Detection: Google Dataproc and Apache Spark apply machine learning models to detect potential fraud.
- Data Storage: Processed and aggregated data is stored in Google Cloud Storage (GCS) and BigQuery.
- Visualization: Power BI provides real-time dashboards for fraud analysts.
- Alerts: Alerts are triggered when suspicious transactions are flagged.

Key Functions Defined by the FRS:

1. Real-time Data Ingestion: The system can collect and stream real-time transaction data with minimal delay.
2. Fraud Detection Engine: The machine learning-based fraud detection engine analyzes transaction patterns in real time.
3. Visualization and Reporting: The system generates detailed reports and dashboards for fraud analysts.
4. Alerting System: The system sends real-time alerts whenever fraud is detected.

Dataset Overview:

This dataset includes a variety of transaction-related fields for building a fraud detection system. It provides detailed information about transactions, the merchants involved, the customers making the transactions, and a range of metadata useful for evaluating potential fraud risks. Each row represents a single transaction, capturing multiple aspects of the event that can be analyzed to identify fraudulent activity.

Dataset Features :

1. **Transaction Information:**
    - **Transaction ID:** A unique identifier for each transaction.
    - **Transaction Date/Time:** The date and time when the transaction occurred.
    - **Transaction Amount:** The amount involved in the transaction.
    - **Currency:** The currency used in the transaction.
    - **Transaction Type:** The type of transaction (e.g., purchase, withdrawal).
    - **Authorization Code:** A code used to authorize the transaction.
    - **Fraud Flag:** A flag indicating whether the transaction was identified as fraudulent.
2. **Merchant Information:**
    - **Merchant Name:** The name of the merchant involved in the transaction.
    - **Merchant Category Code (MCC):** A code that classifies the type of goods or services sold by the merchant.
    - **Location:** The merchant's location.
    - **Latitude & Longitude:** Geolocation coordinates of the merchant's location.
    - **Merchant ID:** A unique identifier for the merchant.
    - **Terminal ID:** The ID of the point of sale (POS) terminal used.
    - **Merchant Rating:** Rating of the merchant based on customer feedback.

3. **Customer Information:**
    - **Customer ID:** A unique identifier for the customer making the transaction.
    - **First Name, Last Name, Email, Phone Number, Address:** Personal details of the customer.
    - **City, Country:** The customer's geographic location.
    - **Birth Date:** The date of birth of the customer, useful for deriving the age.
    - **Income Bracket:** The income category of the customer.
    - **Account Balance:** The balance in the customer's account before the transaction.
    - **Remaining Balance:** The remaining balance after the transaction.
    - **Cardholder's Age:** The age of the cardholder at the time of the transaction.
    - **Cardholder's Gender:** The gender of the cardholder.
4. **Transaction Risk and Fraud Detection:**
    - **Transaction Risk Score:** A numerical score that represents the likelihood of the transaction being fraudulent.
    - **Fraud Detection Method:** The method used to determine if the transaction was fraudulent (e.g., rule-based, machine learning).
    - **Response Time:** The time taken to evaluate whether the transaction was fraudulent.
5. **Card Information:**
    - **Card Type:** The type of card used (e.g., debit, credit).
    - **Card Expiration Date:** The expiration date of the card.
    - **Card Number:** The card number used in the transaction.
    - **Card CVV:** The CVV number used to authorize the transaction.
6. **Device and Location Information:**
    - **POS Entry Mode:** How the card information was entered (e.g., chip, swipe).
    - **Device Information:** Information about the device used for the transaction.
    - **IP Address:** The IP address associated with the transaction.

7. **Other Features:**
    - **Loyalty Points Earned:** Any loyalty points the customer earned from the transaction.
    - **Installment Information:** If the transaction was part of an installment payment plan.
    - **Recurring Transaction Flag:** Indicates whether the transaction was part of a recurring payment (e.g., subscriptions).

Data Analysis:

1. Data Exploration:

- **Summary statistics for numeric columns :**

summary	Transaction Amount	Transaction Risk Score	Account Balance	Cardholder's Age
count	2512668	2512668	2512668	2512668
mean	4278.73634896054	0.49989534232138944	25929.931929512706	48.45255561021193
stddev	3227.1428356495867	0.2885457092100204	13790.576948566844	18.502575809842
min	60.52	0.0	2008.15	18
max	14993.34	1.0	49980.56	81

- **Count missing values in each column:**

Latitude	Longitude
7262	7262

2. Transaction Analysis :

- **Transaction Amount by Merchant Category (MCC):**

Merchant Category Code (MCC)	sum(Transaction Amount)
4785	5972620.74
5881	5927297.18
5813	5913432.379999999
4381	5904003.77
4548	5901090.5600000005

- **Count of transactions by customer :**

Customer ID	First Name	Last Name	Transaction Count
91de9441-378f-43f...	Larry	Nguyen	2998
c573a70c-564f-4b2...	Ann	Monroe	2997
ffefc8bc-4aad-48a...	Jason	Williams	2997
1197aa7e-5edf-466...	Mark	Chen	2996
1740405e-dca1-485...	Sara	Evans	2995
7decc72e-63f6-4a6...	James	Johnston	2992
acd20eee-c10d-4b4...	Glenn	Robertson	2990

- **Transaction Count by Type :**

Transaction Type	Transaction Count
Purchase	838105
Refund	837073
Withdrawal	837490

- **Total count transaction amount by year and month :**

Year	Month	count
2023	9	206077
2023	10	212872
2023	11	206341
2023	12	212908
2024	1	213016
2024	2	198992
2024	3	212264
2024	4	205576
2024	5	213213
2024	6	205418
2024	7	213058
2024	8	212933

3. Fraud Detection Insights :

- **Count of Fraudulent vs Non-Fraudulent Transactions :**

Fraud Flag	count
true	75942
false	2436726

- **Count of Fraudulent vs Non-Fraudulent Transactions :**

Fraud Flag	avg(Transaction Amount)
true	4241.743290537512

- **Fraud is true and false over months :**

Year	Month	Fraud Flag	count
2023	9	false	199859
2023	9	true	6218
2023	10	false	206389
2023	10	true	6483
2023	11	false	200158
2023	11	true	6183
2023	12	false	206322
2023	12	true	6586
2024	1	false	206758
2024	1	true	6258
2024	2	false	193006
2024	2	true	5986
2024	3	false	205994
2024	3	true	6270
2024	4	false	199381
2024	4	true	6195
2024	5	false	206629
2024	5	true	6584
2024	6	false	199169
2024	6	true	6249

only showing top 20 rows

- **Fraud is true and false over months Top Merchants by Transaction Amount :**

Merchant Name	sum(Transaction Amount)
Smith LLC	1.373039921E7
Smith PLC	1.3008163779999997E7
Smith and Sons	1.2980866149999999E7
Smith Group	1.2898090749999998E7
Smith Ltd	1.2879751819999998E7
Smith Inc	1.2182976850000003E7
Johnson and Sons	1.0300928120000001E7
Johnson LLC	1.028184927E7
Johnson PLC	1.0225173810000002E7
Johnson Ltd	1.0102643959999997E7

Cardholder's Age	avg(Transaction Amount)
62	5876.376621411277
18	5551.258742677369
78	5382.635892355801
70	5335.624560000001
42	5271.927797626068
35	5118.406846356603
61	5050.354368719457
41	5036.980337340144
36	4990.1193549248155
66	4973.873576623785
55	4973.113638909513
24	4910.126646599232
47	4896.022538820057
44	4872.754557836135
30	4838.014931331386
77	4825.492419913538
20	4823.410166882552
27	4778.979084523287
64	4720.681737603595
50	4712.522354765475

4. Customer Insights :

- **Average Transaction Amount by Age Group :**

- **Transaction Count by Income Bracket :**

Cardholder's Income Bracket	count
['Middle']	1553547
['Low']	709323
['High']	249798

- **Customer has transaction in in the same day**

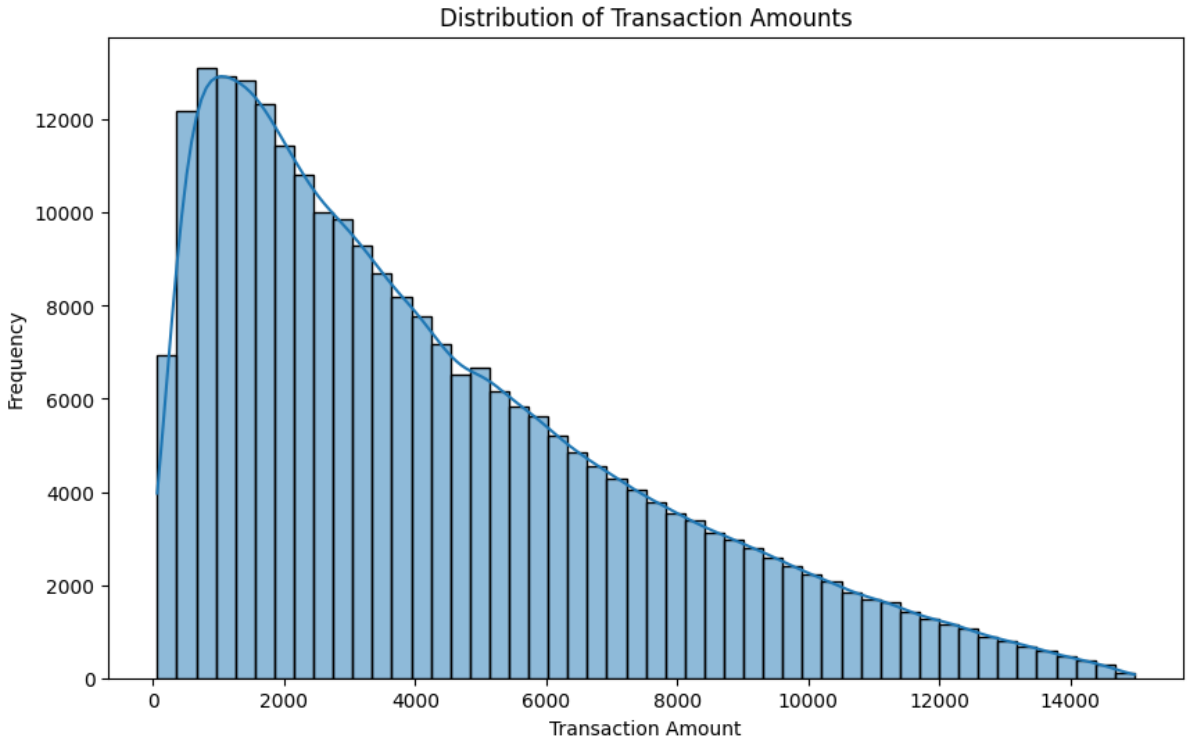
Customer ID	Transaction Date	count
fd2eb4c4-da8f-460...	2024-04-17	22
ea8d6987-645b-450...	2024-04-29	22
e3d04c55-fc7d-40f...	2023-09-09	21
5eea2bfd-53d5-44f...	2024-01-07	21
0f212fdc-7eba-474...	2024-05-20	21
b20ba2a3-1426-4b8...	2023-12-15	21
6d64ddaf-0c3c-487...	2023-12-10	21
d9f46258-cb5e-4c0...	2024-06-27	21
df97e887-dc92-4a7...	2024-08-03	21
b1838724-a0a5-45a...	2023-11-24	21
3b3a6822-9493-426...	2024-01-23	21
360c0c1c-39f7-4a7...	2024-06-11	21
49aadcde-cc79-475...	2024-08-31	20
e1521535-8b76-406...	2024-07-23	20
86dc00fc-b51f-4c4...	2024-03-26	20
5d0b8352-ac90-4f8...	2024-06-26	20
3708f274-ad30-48e...	2023-09-11	20
ea315bec-bdb3-4e0...	2023-11-03	20
90a1369d-9d3e-400...	2024-02-14	20
b0a1c68d-17f2-471...	2023-09-28	20

5. Geographical Analysis and Visualizations :

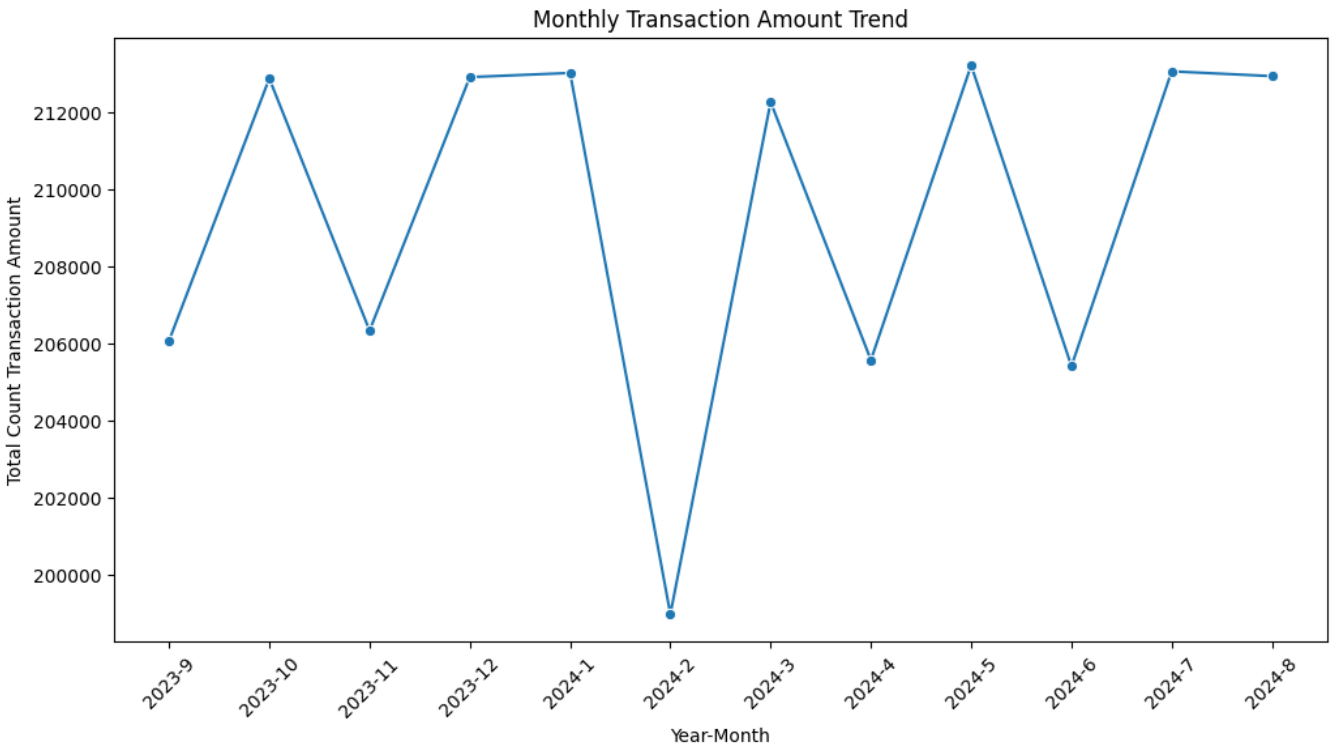
- **Transaction Amount by City and Country :**

City	Country	sum(Transaction Amount)
South Robert	United States	4.810052881E7
West Christopher	United States	3.368437878000001E7
New Jeffrey	United States	3.340881656E7
Howardberg	United States	3.1353720800000004E7
West Christian	United States	3.059180405E7
East Hannah	United States	3.0411852930000003E7
East Richard	United States	2.979896752999999E7
East Christopher	United States	2.8623728189999998E7
Lake Amanda	United States	2.784353902E7
Jamesburgh	United States	2.6484626849999998E7
Port Michael	United States	2.6382598460000005E7
Heatherberg	United States	2.585352629999999E7
East Kevin	United States	2.4855993289999995E7
Andrewshire	United States	2.4512128420000006E7
Port Ethanmouth	United States	2.4070589339999996E7
South Michael	United States	2.404862187E7
Allenland	United States	2.3968476900000006E7
Lake Meganchester	United States	2.3957112889999997E7
South Tracey	United States	2.394675185E7
South Michaeltown	United States	2.394118564E7

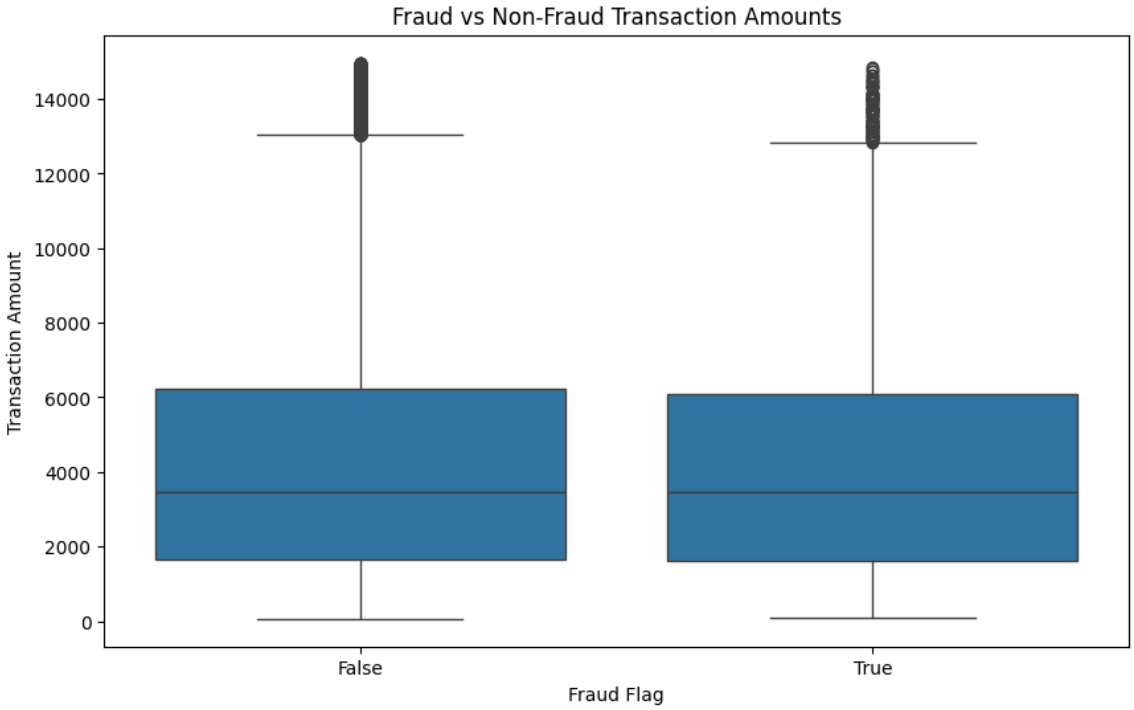
The distribution of transaction amounts :



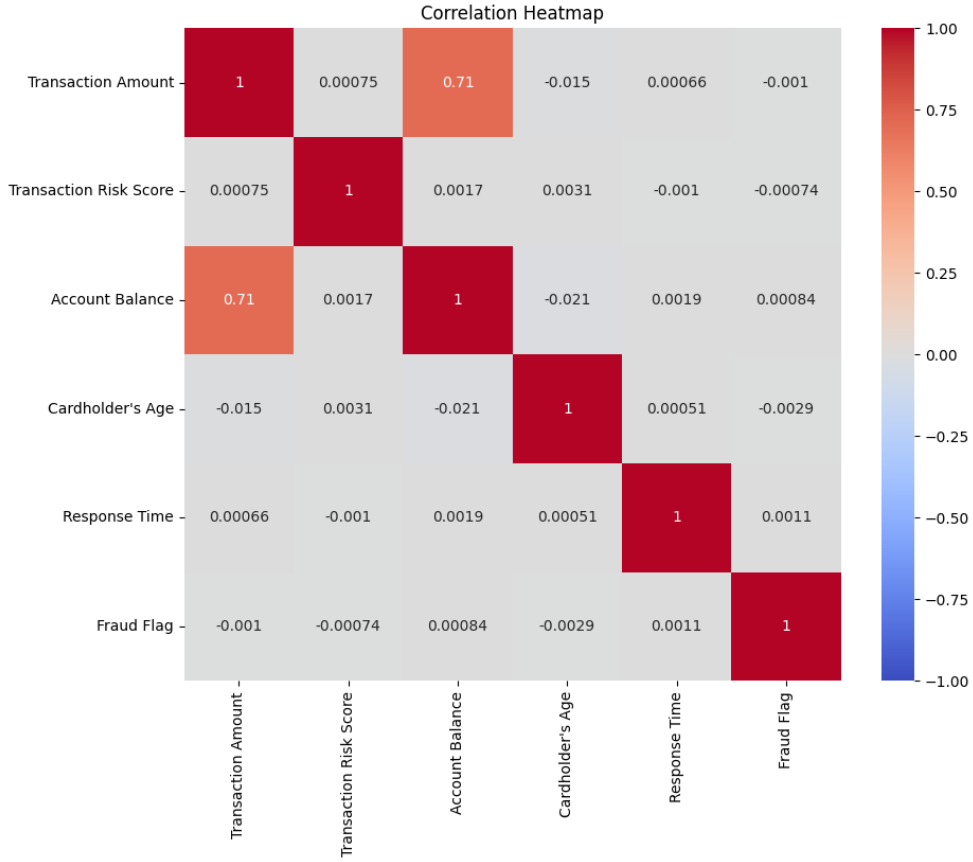
- **Monthly Transaction Amount Trend :**



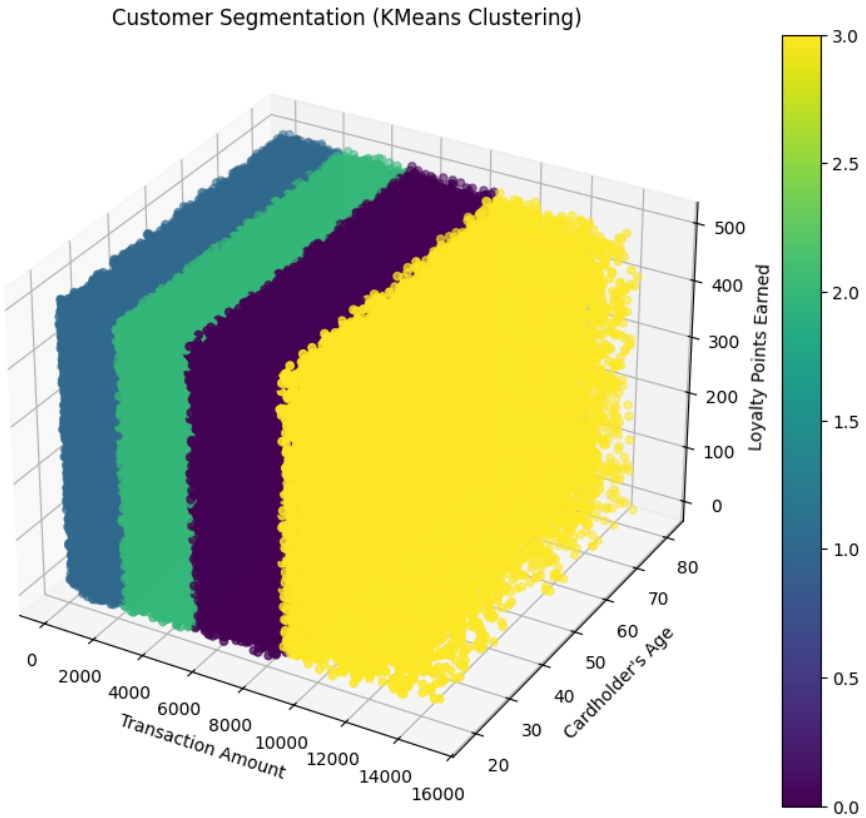
- **Boxplot of Transaction amounts for fraud vs non-frad transactions :**



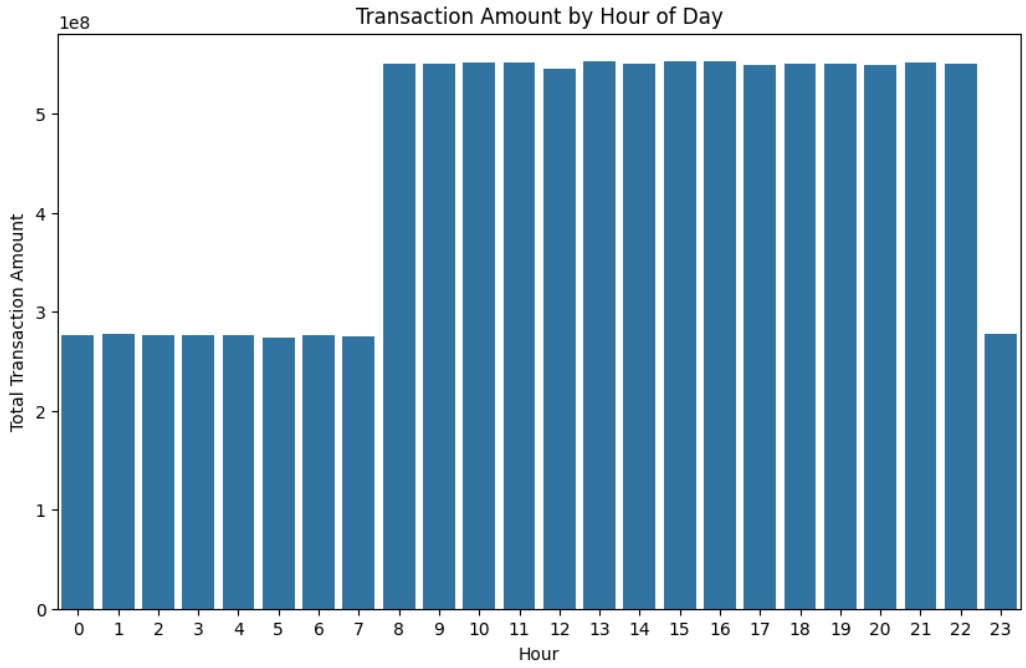
- **Heatmap of Correlation between Features :**



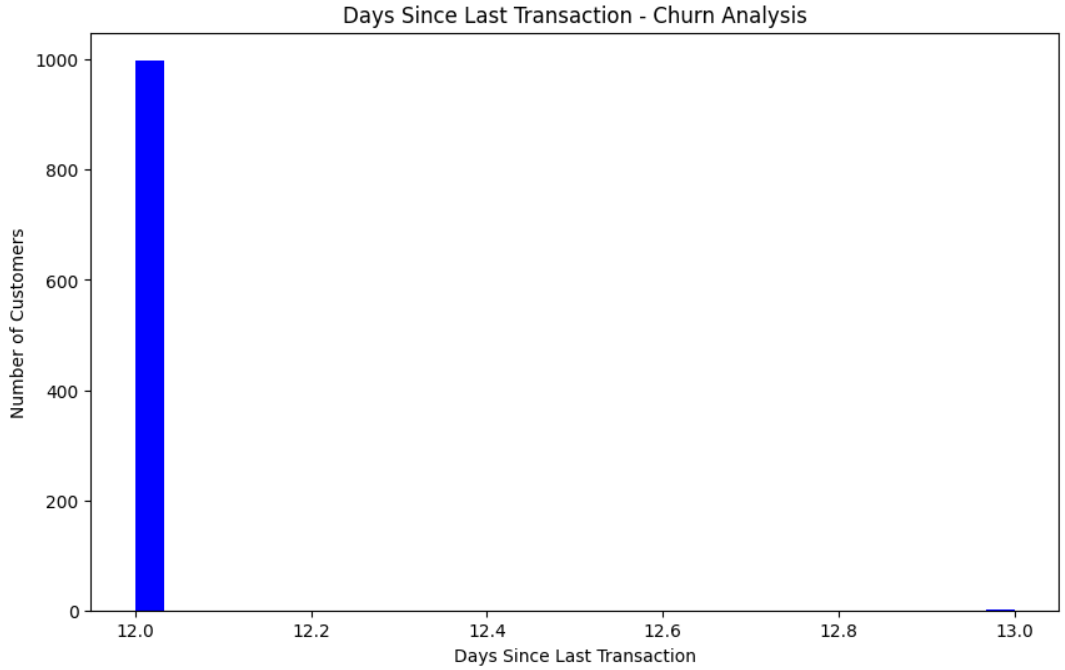
- **Customer Segmentation Analysis :**



- **Transaction Amount by Hour of Day :**



- **Days Since Last Transaction :**



Data Cleaning and Preparing :

- **Handling Missing Values :**

```
[ ]  df = df.fillna({'latitude': 0.0, 'longitude': 0.0,'Email': 'unknown'})
```

- **Cleaning and Extracting values on columns:**

```
from pyspark.sql.functions import expr,split, col

df = df.withColumn(
    "Transaction Status",
    split(col("Transaction Status"), "'")[1]
).withColumn(
    "Income Bracket",
    split(col("Income Bracket"), "'")[1]
)
df.show(5)
```

```
+------------------+--------------+
|Transaction Status|Income Bracket|
+------------------+--------------+
|          Approved|           Low|
|          Approved|           Low|
|          Approved|          High|
|          Approved|           Low|
|          Approved|          High|
+------------------+--------------+
```

- **Extract Columns From “Device Information”**

```
# Sample regular expressions to extract device_type, os, and browser
df['device_type'] = df['Device Information'].str.extract(r"'device_type': '([^']+)'")
df['os'] = df['Device Information'].str.extract(r"'browser': '\(?([a-zA-Z]+)")
df['browser'] = df['Device Information'].str.extract(r"'os': '([^/]+)")
```

- **Extract Columns From “Transaction Date/Time”**

```
# Step 6: Extract year, month, day, hour, and minute from the date
df['Year'] = df['Transaction Date/Time'].dt.year
df['Month'] = df['Transaction Date/Time'].dt.month
df['Day'] = df['Transaction Date/Time'].dt.day
df['Hour'] = df['Transaction Date/Time'].dt.hour
df['Minute'] = df['Transaction Date/Time'].dt.minute
```

Feature Engineering :

- **Dealing with Imbalanced Data :**

  The balance_fraud_data function addresses class imbalance in fraud detection datasets by ensuring an equal number of fraud and non-fraud cases.

  Purpose:

  To balance the dataset by either:

  - Downsampling the majority class (non-fraud) if there are more non-fraudulent transactions.
  - Upsampling the minority class (fraud) if there are fewer fraudulent transactions.

  Parameters:

  - df_sample (DataFrame): The dataset containing transaction data.
  - fraud_column (str): Column indicating fraud (1 for fraud, 0 for non-fraud).

  Process:

  1. Split the dataset into fraud and non-fraud DataFrames.
  2. Compare the counts of fraud and non-fraud cases.
  3. Downsample non-fraud cases or upsample fraud cases to achieve balance.
  4. Return a balanced DataFrame with an equal number of fraud and non-fraud cases.

```python
def balance_fraud_data(df_sample, fraud_column='Fraud Flag'):
    # Separate the fraud and non-fraud cases
    fraud_df = df_sample[df_sample[fraud_column] == 1]
    non_fraud_df = df_sample[df_sample[fraud_column] == 0]

    # Get the counts of fraud and non-fraud cases
    fraud_count = fraud_df.shape[0]
    non_fraud_count = non_fraud_df.shape[0]

    # If non-fraud is larger, we downsample it to match fraud count, else we upsample fraud
    if non_fraud_count > fraud_count:
        non_fraud_df_downsampled = resample(non_fraud_df,
                                            replace=False,    # sample without replacement
                                            n_samples=fraud_count,   # match fraud count
                                            random_state=42)
        balanced_df = pd.concat([fraud_df, non_fraud_df_downsampled])
    else:
        fraud_df_upsampled = resample(fraud_df,
                                      replace=True,    # sample with replacement
                                      n_samples=non_fraud_count,   # match non-fraud count
                                      random_state=42)
        balanced_df = pd.concat([fraud_df_upsampled, non_fraud_df])

    return balanced_df
```

```
data['Fraud Flag'].value_counts()
```

```
Fraud Flag
True     75942
False    75942
Name: count, dtype: int64
```

- **Calculate Distance between Latitude and Longitude :**

```
# Calculate distance between Latitude and Longitude
data['Distance'] = 0
for index, row in data.iterrows():
  if pd.notna(row['Latitude']) and pd.notna(row['Longitude']):
    data.loc[index, 'Distance'] = geodesic((row['Latitude'], row['Longitude']), (0, 0)).kilometers
```

- **Handle Categorical features :**

```
le = LabelEncoder()
categorical_cols = ['POS Entry Mode', 'Transaction Status', 'Merchant Category Code (MCC)', 'Merchant Rating', 'Transaction Type','os','browser',
                    'browser','Income Bracket','Response Time','Transaction Notes']
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
```

Modeling :

- **Features and Target :**

```
# Split data into features and target variable:
X = data[['Transaction Amount', 'Transaction Risk Score', 'POS Entry Mode', 'Hour', 'Day','Minute','os','browser','Income Bracket',
          'Transaction Status', 'Merchant Category Code (MCC)', 'Merchant Rating', 'Distance',
          'Cardholder\'s Age', 'Account Balance', 'Recurring Transaction Flag',
          'Transaction Type','Transaction Notes','Remaining Balance']]
y = data['Fraud Flag']  # Assuming 'Fraudulent' is the target column
```

- **Split data into training and testing sets :**

```
# Split data into training and testing sets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Standardize our Features :**

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- **Define the Model and set its parameters :**

```
rm=RandomForestClassifier(n_estimators=500, max_depth=50, min_samples_split=10, min_samples_leaf=4, max_features='sqrt', random_state=42)
```

- **Training the model on the Scaled Features :**

```
rm.fit(X_train_scaled, y_train)
```

- **Predicting our Target :**

```
y_pred = rm.predict(X_test_scaled)
```

Evaluate the model :

- **Evaluate and Rating The Model:**

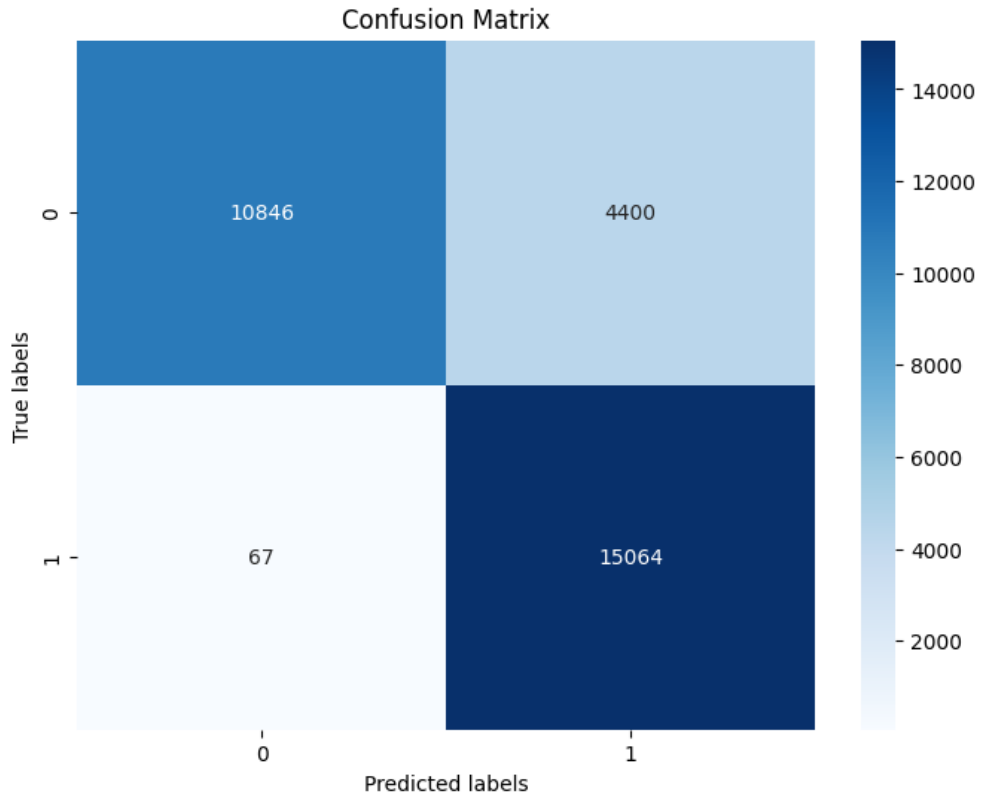
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)


print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
Accuracy: 0.8529479540441781
Precision: 0.7739416358405261
Recall: 0.995572004494085
F1-Score: 0.8708772944067062
```

- **Plot Confusion Matrix :**

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



- **Saving Our Model as Format (pkl) :**

```
#save the model
import joblib
joblib.dump(rm, 'fraud_detection_model.pkl')
```

```
['fraud_detection_model.pkl']
```

Tools and Technologies

Programming Languages

In your real-time fraud detection system project, the use of Python and SQL plays a significant role in different aspects of the system's architecture:

Python:

1. **Data Ingestion**: Python, along with libraries such as Apache Kafka or Apache Flink, can help in ingesting high-velocity data from multiple sources like transaction systems or financial services. Python scripts can be used to set up data pipelines and handle real-time streaming data.
2. **Data Processing**: Python is often used for real-time processing of transaction data. With frameworks like Apache Spark Streaming or Flink, Python allows for distributed processing of data, enabling real-time analysis. Pandas and NumPy can handle smaller-scale batch processing, while PySpark is useful for large datasets in distributed environments.
3. **Machine Learning**: Python's extensive machine learning libraries like scikit-learn, TensorFlow, PyTorch, and XGBoost are used for building fraud detection models. These models can be trained to detect patterns of fraud by analyzing historical data and can also be applied in real-time to detect anomalies or fraudulent behavior.
4. **Model Deployment**: Python can be used to serve machine learning models for fraud detection in real-time. Flask or FastAPI can be employed to create APIs that take in transaction data and return predictions in real-time.
5. **Monitoring & Logging**: Python is used for logging transaction data and monitoring the system's performance, leveraging libraries like Loguru or the built-in logging module.

SQL:

1. **Data Storage**: SQL plays a crucial role in storing transaction data, user profiles, and other related data in relational databases such as MySQL, PostgreSQL, or cloud-based solutions like Google BigQuery or Amazon RDS. It is also used to query large historical datasets for training machine learning models.
2. **Data Aggregation**: SQL is useful for performing aggregations, summarizing transaction data over time to identify trends or anomalies that may indicate fraud.
3. **Data Analytics**: SQL queries can be used to generate reports or dashboards that provide insights into the volume of transactions, detected fraud cases, and other business metrics. It is an effective tool for business analysts to assess the performance of the fraud detection system.

4. **Feature Engineering**: SQL is often used to extract features from raw data stored in databases, which can then be used in machine learning models. Complex joins, subqueries, and window functions are commonly used to create the right features.

By combining the strengths of both Python for real-time processing and machine learning, and SQL for data storage and manipulation, you can build a scalable, robust fraud detection system.

Online Tools

1. Data Sources:

- **Description**: This refers to the sources of transaction data that need to be analyzed for fraud detection. It could be databases or real-time transaction feeds from multiple financial institutions or services.
- **Role**: Serve as the entry point for the entire fraud detection process by providing raw data.

2. Kafka Producer:

- **Description**: Apache Kafka is used as a data streaming platform that ingests real-time transaction data.
- **Role**: The Kafka producer continuously extracts transaction data from the data sources and sends it to a Kafka topic for real-time processing.

3. Pub/Sub Topic (Google Cloud Pub/Sub):

- **Description**: Google Cloud Pub/Sub is a messaging service for exchanging messages between independent applications.
- **Role**: Pub/Sub topics act as intermediaries to pass data from Kafka to the downstream components. It enables asynchronous communication and ensures data is published and consumed in real-time.

4. Google Dataflow:

- **Description**: A managed service for stream and batch processing using Apache Beam. It’s used for processing large data sets.
- **Role**: Preprocesses incoming data by cleaning, transforming, and normalizing it before it gets stored in a data lake. Dataflow handles data at scale and provides low-latency processing for real-time applications.

5. Google Cloud Storage (Data Lake):

- **Description**: Google Cloud Storage (GCS) is a service for storing and accessing data in the cloud.
- **Role**: Acts as a data lake where processed or preprocessed data is stored. This allows for long-term storage of large datasets that may be used for further analysis or model training.

6. Google Cloud Composer:

- **Description**: A managed workflow orchestration tool based on Apache Airflow.
- **Role**: Orchestrates different tasks within the data pipeline, such as scheduling jobs, managing dependencies, and automating ETL workflows.

7. Apache Spark (Machine Learning and Modeling):

- **Description**: Apache Spark is a distributed processing system used for big data workloads, and it includes machine learning libraries.
- **Role**: Spark is used for both machine learning (ML) and data modeling tasks. This can include training fraud detection models and transforming transaction data to identify fraudulent patterns.

8. Google Cloud Dataproc:

- **Description**: A fully managed and scalable service for running Apache Spark, Hadoop, and other open-source big data tools.
- **Role**: Enables distributed data processing for machine learning tasks, including model training and deployment on large datasets.

9. Google BigQuery:

- **Description**: A fully-managed data warehouse that allows SQL-like queries on large datasets.
- **Role**: Stores the transformed data and allows querying for analytics and reporting. It is also integrated with tools for data visualization and real-time querying for insights.

10. Power BI:

- **Description**: A business analytics service by Microsoft that provides data visualization and business intelligence capabilities.
- **Role**: Provides dashboards and visual reports based on the data stored in Google BigQuery, helping stakeholders understand transaction patterns and identify fraud trends.

11. Google Cloud Monitoring:

- **Description**: A service for monitoring, logging, and alerting in real-time for Google Cloud-based resources.
- **Role**: Monitors the health of the fraud detection pipeline, tracks system performance, and triggers alerts or notifications in case of suspicious activity or system issues.

12. Alerting & Notifications:

- **Description**: Notification systems integrated with Google Cloud Monitoring.
- **Role**: Sends real-time alerts to relevant teams whenever potential fraud is detected or if there's an issue with the system, allowing for immediate action.

Libraries

Here's a detailed description of the libraries used in the project:

1. PySpark SQL Functions

- **Library**: pyspark.sql.functions
- **Functions Used**: expr, split, col, udf, regexp_replace
- **Description**: These functions are part of the PySpark SQL module and are used for data manipulation and transformation on distributed Spark DataFrames.
    - expr: Allows for SQL expressions to be used on Spark DataFrames.
    - split: Splits a string into an array based on a delimiter.
    - col: Refers to a column in a DataFrame for transformations or filters.
    - udf: User-Defined Function, used to create custom functions to apply on DataFrames.
    - regexp_replace: Replaces a substring in a column based on a regular expression.

2. PySpark Window Functions

- **Library**: pyspark.sql.Window
- **Description**: Used for creating windowing operations in Spark DataFrames. It allows you to perform operations like ranking, cumulative sums, or calculating moving averages over a specified window (set of rows).

3. PySpark Data Types

- **Library**: pyspark.sql.types

- **Types Used**: StructType, StructField, StringType
- **Description**: Defines the schema for a Spark DataFrame.
    - StructType and StructField: Used to specify the structure of a DataFrame, especially useful when reading or transforming data with specific column types.
    - StringType: Represents string data types in a DataFrame schema.

4. Requests

- **Library**: requests
- **Description**: A simple HTTP library for Python, used to send HTTP requests and handle responses. In this project, it could be used to fetch external data (e.g., from APIs) for real-time fraud detection.

5. Pandas

- **Library**: pandas
- **Description**: A powerful data manipulation and analysis library. Provides data structures like DataFrames to work with structured data, which is useful for exploratory data analysis and preprocessing before feeding data into machine learning models.

6. Scikit-learn - Train/Test Split

- **Library**: sklearn.model_selection
- **Function Used**: train_test_split
- **Description**: This function is used to split the dataset into training and testing sets, which is essential for evaluating machine learning models.

7. Scikit-learn - Preprocessing

- **Library**: sklearn.preprocessing
- **Functions Used**: LabelEncoder, StandardScaler
- **Description**:
    - LabelEncoder: Encodes categorical features or labels into numerical values.
    - StandardScaler: Standardizes features by removing the mean and scaling them to unit variance, which is important for machine learning algorithms that rely on distance metrics.

8. Scikit-learn - Ensemble Learning

- **Library**: sklearn.ensemble

- **Model Used**: RandomForestClassifier
- **Description**: A robust and widely-used ensemble learning algorithm. The Random Forest classifier is an ensemble of decision trees, typically used to improve the accuracy and reduce overfitting. It's commonly used for classification tasks like fraud detection.

9. Scikit-learn - Evaluation Metrics

- **Library**: sklearn.metrics
- **Metrics Used**: accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
- **Description**: This library provides various metrics to evaluate the performance of machine learning models:
    - accuracy_score: Measures the proportion of correctly predicted instances.
    - precision_score: The ratio of true positive predictions to the total predicted positives.
    - recall_score: The ratio of true positives to the actual positives in the data.
    - f1_score: The harmonic mean of precision and recall.
    - roc_auc_score: Measures the area under the ROC curve, useful for binary classification performance.
    - confusion_matrix: Provides a breakdown of true positives, true negatives, false positives, and false negatives.

10. Scikit-learn - Resampling

- **Library**: sklearn.utils
- **Function Used**: resample
- **Description**: This function can be used to oversample or undersample a dataset to handle class imbalances, which is common in fraud detection where fraudulent transactions are much less frequent than legitimate ones.

11. Geopy

- **Library**: geopy.distance
- **Function Used**: geodesic
- **Description**: This library is used to calculate geographical distances between two points (based on their latitude and longitude). It is useful in fraud detection to check if a transaction happens far from the cardholder's usual location.

12. Matplotlib

- **Library**: matplotlib.pyplot

- **Description**: A widely used library for creating visualizations in Python. It's used to create static, interactive, and animated plots to visualize patterns in the data or model performance (e.g., confusion matrices or ROC curves).

13. Seaborn

- **Library**: seaborn
- **Description**: A data visualization library built on top of Matplotlib. It offers a higher-level interface for drawing attractive and informative statistical graphics, like heatmaps or distribution plots, which are useful for visualizing fraud detection model results and patterns in the dataset.

These libraries together support data preprocessing, model building, evaluation, visualization, and real-time data handling, forming the backbone of a comprehensive fraud detection system.

Kpi

- **Customer Fraud Statistics:**
    - Calculate the number of fraudulent customers and the fraud rate.
- **Average Transactions per Customer:**
    - Compute the average number of transactions conducted per customer.
- **Fraud Incidence by Card Type:**
    - Determine the ratio of fraudulent transactions based on card type.
- **Average Response Time for Fraud vs. Non-Fraud Cases**:
    - Compare the average response time between fraudulent and non-fraudulent transactions.
- **Average Transaction Amount by Income Bracket**:
    - Analyze the average transaction amount for customer groups segmented by income bracket.

- **Average Account Balance for Fraudulent Customers:**
    - Calculate the average account balance for customers flagged as fraudulent.

- **Hourly Fraud Rate**:
    - Analyze the ratio of fraudulent transactions by hour of the day.

- **Fraud Incidence Among Top 20 Merchants:**
    - Determine the fraud ratio among the top 20 merchant names by transaction volume.

- **Transactions by Merchant Category:**
    - Calculate the ratio of transactions by merchant category.

- **Quarterly Fraud vs. Non-Fraud Transaction Trends**:
    - Examine the relationship between the quarter of the year and the number of fraud and non-fraud transactions.

- **Operating System Usage and Fraud Rates:**
    - Explore the relationship between operating systems and the incidence of fraud and non-transactional events.

- **Browser Usage and Fraud Rates**:
    - Analyze the correlation between browser type and the frequency of fraud and non- transactional events.

- **Transaction Channel and Fraud Rates:**
    - Investigate the relationship between transaction channels and the incidence of fraud and non-transactional events.

- **POS Entry Mode and Fraud Rates:**
    - Study the correlation between Point-of-Sale (POS) entry modes and fraud detection rates.

- **Hourly Trends for Fraud and Non-Fraud Transactions**:
    - Assess the relationship between the hour of the day and the frequency of fraud and non-fraud transactions.

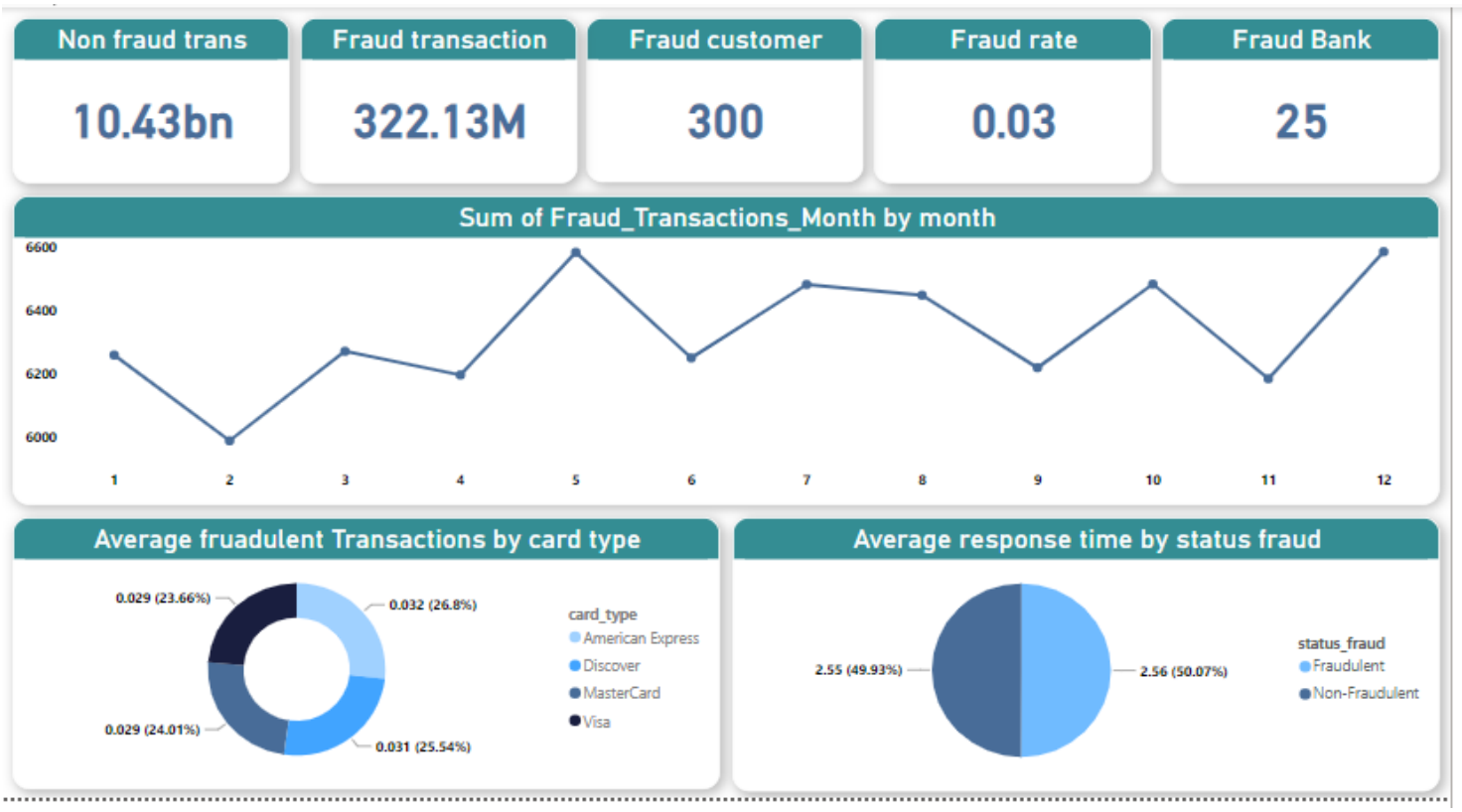
- **Total Non-Fraud Transaction Amount**:
    - Calculate the total transaction amount for non-fraudulent transactions compared to fraudulent ones.

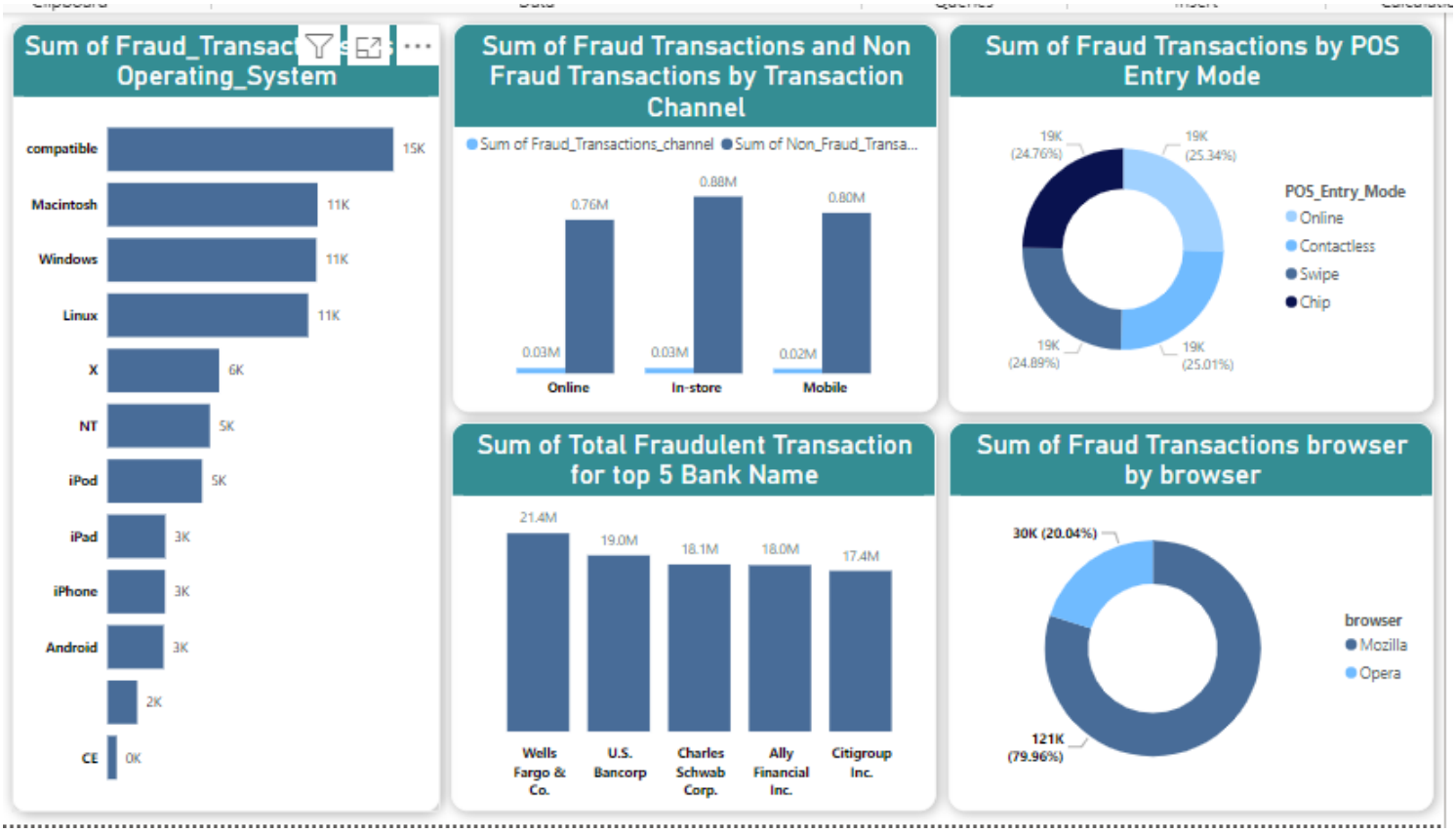
- **Fraud Incidence by Top 5 Banks:**
    - Analyze the total transaction amounts of fraudulent cases across the top 5 banks.

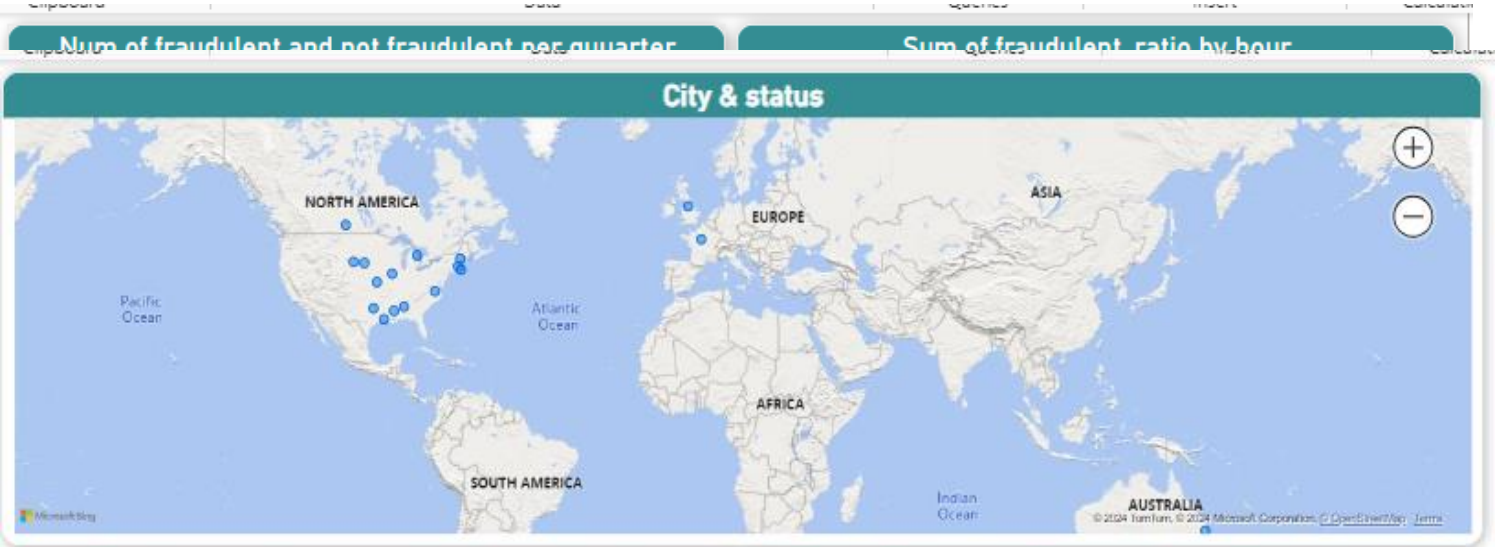
- **Geographical Fraud Analysis (State & City):**
    - Examine the relationship between states, cities, and the number of fraudulent customers in each region.

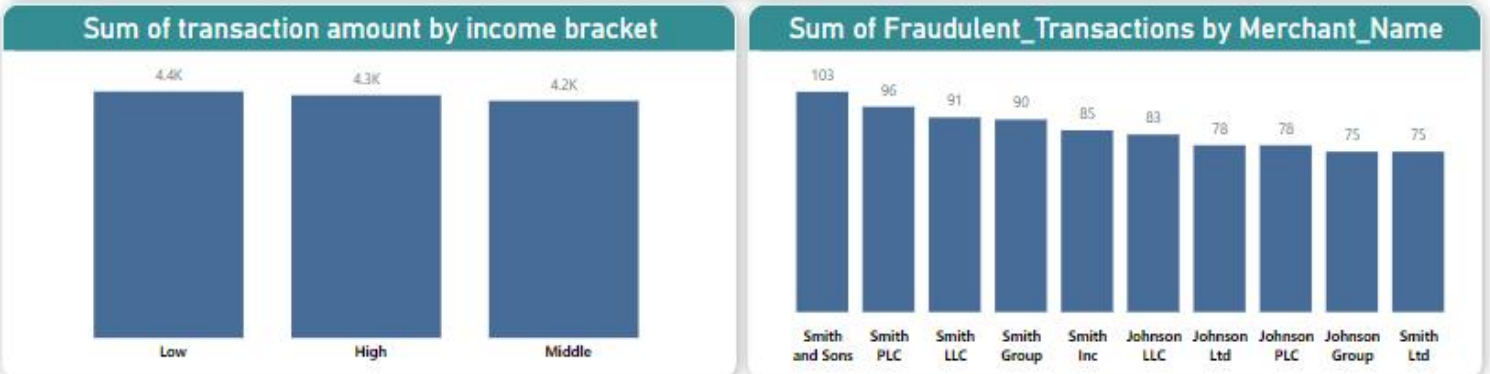
Data Visualization

Data visualization transforms complex data into visual formats like charts and graphs, making patterns and insights easier to understand. It simplifies analysis, enhances decision-making, and improves communication of data insights.









Data description

The dataset provided is focused on detecting fraudulent transactions by analyzing various features related to cardholder behavior, transaction details, and merchant information. It contains both numerical and categorical fields, offering a comprehensive view of transactional activity.

Key attributes include:

- **Personal Information**: Fields such as Phone Number, Address, City, Customer State, Country, and Birth Date provide details about the cardholder's identity and location. These help in verifying if transactions align with the cardholder's usual behavior and geography.
- **Transaction Details**: Information like Transaction Amount, Transaction Channel, Card Number, and POS Entry Mode captures specifics about each transaction. These features help detect unusual patterns, such as large amounts or transactions through uncommon channels that could signal fraud.
- **Banking and Security**: Fields such as Card Issuing Bank, CVV, and IP Location offer insight into the security aspects of transactions. Anomalies in these areas often serve as red flags for fraud.
- **Fraud Indicator**: The Fraud Flag column labels whether a transaction is fraudulent or legitimate, making this dataset suitable for building and training machine learning models to detect fraud in real-time scenarios.

Describe the data in detail

1. Transaction ID**:**

- **Type**: Categorical (String)
- **Description**: A unique identifier assigned to each transaction. This is crucial for tracking and differentiating between different transactions.

2. Transaction Date/Time**:**

- **Type**: DateTime
- **Description**: The date and time at which the transaction was made. It helps in identifying transaction patterns over time, such as during peak transaction hours, which could be critical for detecting fraud trends.

3. Transaction Amount**:**

- **Type**: Numerical (Float)
- **Description**: The monetary value of the transaction. Large or unusual transaction amounts could be indicators of fraudulent behavior.

4. Currency

- **Type**: Categorical (String)
- **Description**: The currency in which the transaction was made (e.g., USD, EUR). This is important for understanding cross-border transactions and potential foreign exchange-related fraud.

5. Merchant Name

- **Type**: Categorical (String)
- **Description**: The name of the merchant where the transaction occurred. Analysis of transaction volumes and fraud incidents by merchant can help flag high-risk merchants.

6. Merchant Category Code (MCC)

- **Type**: Numerical (Integer)
- **Description**: A four-digit code used to classify businesses by the types of goods or services they provide (e.g., 5411 for grocery stores). Certain categories might have higher fraud risks (e.g., online retailers).

7. Location**:**

- **Type**: Categorical (String)
- **Description**: The geographical location of the transaction, often city-level, which helps in identifying transactions in unusual locations for the cardholder, a potential fraud signal.

8. Latitude **/** Longitude**:**

- **Type**: Numerical (Float)
- **Description**: Geographic coordinates of the transaction location. Although the data has no entries here, it could be used for geospatial analysis in detecting fraud (e.g., card used in two distant locations within a short time frame).

9. Transaction Type**:**

- **Type**: Categorical (String)
- **Description**: Specifies the type of transaction (e.g., Purchase, Withdrawal, Refund). Fraudulent patterns may differ between transaction types.

10. Authorization Code**:**

- **Type**: Numerical (Integer)
- **Description**: A unique code authorizing the transaction. Authorization failures or irregularities can signal potential fraud.

11. Fraud Flag**:**

- **Type**: Boolean (True/False)
- **Description**: Indicates whether a transaction has been identified as fraudulent. This column is crucial for building and testing fraud detection models.

12. Transaction**:**

- **Type**: Categorical (String)
- **Description**: Status of the transaction, such as Completed, Pending, or Rejected. Fraudulent transactions may often be pending or rejected by systems.

13. POS Entry Mode**:**

- **Type**: Categorical (String)
- **Description**: Describes how the card information was entered at the point of sale, e.g., by swiping, chip insert, or manual entry. Certain entry modes, like manual, could be riskier.

14. Installment Information**:**

- **Type**: Categorical (String)
- **Description**: Details if the transaction was part of an installment plan. Fraudulent actors may take advantage of installment payments for high-ticket items.

15. Merchant ID **(Merchant ID):**

- **Type**: Categorical (String)
- **Description**: A unique identifier for each merchant. This can help track merchants with higher instances of fraud.

16. Terminal ID**:**

- **Type**: Categorical (String)

- **Description**: The ID of the point-of-sale (POS) terminal where the transaction was made. Unusual terminals (especially in high-risk locations) could be a flag for fraud.

17. IP Address**:**

- **Type**: Categorical (String)
- **Description**: The IP address used during the transaction, relevant for identifying online transaction fraud or unauthorized logins.

18. Device Information**:**

- **Type**: Categorical (String)
- **Description**: Details about the device used for the transaction (e.g., mobile, desktop). Unrecognized devices used by frequent customers could signal fraud.

19. Transaction Risk Score**:**

- **Type**: Numerical (Float)
- **Description**: A calculated score indicating the risk level of each transaction. Higher scores may be assigned based on historical patterns and anomaly detection.

20. Loyalty Points Earned**:**

- **Type**: Numerical (Integer)
- **Description**: The number of loyalty points earned during the transaction. Unusual point accumulation could be a sign of rewards fraud.

21. Transaction Notes**:**

- **Type**: Categorical (String)
- **Description**: Free-text notes about the transaction, possibly containing important information about unusual circumstances or customer complaints.

22. Cardholder's Income Bracket**:**

- **Type**: Categorical (String)
- **Description**: The income bracket of the cardholder, which could help in assessing if a transaction is out of character based on usual spending behavior.

23. Fraud Detection Method**:**

- **Type**: Categorical (String)
- **Description**: The method used to detect fraud, e.g., machine learning, rule-based systems, or manual reviews.

24. Response Time**:**

- **Type**: Numerical (Float)
- **Description**: The time taken to respond to the transaction (usually in seconds). Slower response times could indicate potential issues, such as fraudulent behavior.

25. Merchant Rating**:**

- **Type**: Numerical (Float)
- **Description**: The rating of the merchant, which could help identify high-risk merchants with poor reputations.

26. Recurring Transaction**:**

- **Type**: Boolean (True/False)
- **Description**: Indicates whether the transaction is part of a recurring series. Fraud detection systems should treat recurring transactions differently than one-off purchases.

27. Customer ID**:**

- **Type**: Categorical (String)
- **Description**: A unique identifier for the cardholder/customer. This helps track customer behavior over time and across transactions.

28. First Name / Last Name**:**

- **Type**: Categorical (String)
- **Description**: The cardholder's name. Important for personalized fraud detection and investigation.

29. Email**:**

- **Type**: Categorical (String)
- **Description**: The cardholder's email address, which can be cross-verified with account information to prevent email-based fraud.

30. Cardholder's Age**:**

- **Type**: Numerical (Integer)
- **Description**: The age of the cardholder. Transactions that are unusual for a particular age group could indicate fraud.

31. Cardholder's Gender **(Cardholder's Gender):**

- **Type**: Categorical (String)
- **Description**: The gender of the cardholder, which could be used to track demographics associated with certain fraud patterns.

32. Account Balance **(Account Balance):**

- **Type**: Numerical (Float)
- **Description**: The balance in the account at the time of the transaction. This helps in analyzing whether a transaction was feasible or suspicious based on available funds.

33. Remaining Balance **(Remaining Balance):**

- **Type**: Numerical (Float)
- **Description**: The balance remaining in the account after the transaction is completed.

34. Customer State / Zip Code **(Customer State, Customer Zip Code):**

- **Type**: Categorical (String)
- **Description**: The cardholder's address details, which can be useful for verifying transactions that occur far from the cardholder's usual location.

35. Card Type**:**

- **Type**: Categorical (String)
- **Description**: The type of card used (e.g., Visa, MasterCard, Discover). Different cards may have different levels of security and fraud protection.

36. Card Expiration Date**:**

- **Type**: Categorical (String)

- **Description**: The expiration date of the card used in the transaction. Fraudulent transactions might use expired or soon-to-expire cards.

37. Card Number**:**

- **Type**: Categorical (String)
- **Description**: The credit card number used in the transaction. This is often tokenized for privacy and security.

38. Card Verification Value (CVV)**:**

- **Type**: Categorical (String)
- **Description**: The CVV code associated with the credit card. This three- or four-digit code provides an additional layer of security for online transactions. Invalid or mismatched CVV values could indicate fraudulent behavior.

39. Card Issuing Bank**:**

- **Type**: Categorical (String)
- **Description**: The bank that issued the card used for the transaction. Fraudsters may target specific banks or exploit vulnerabilities in their fraud detection systems.

40. Country of Issue**:**

- **Type**: Categorical (String)
- **Description**: The country where the card was issued. Cross-border transactions, or those occurring far from the cardholder's country, may raise fraud risk flags.

41. Phone Number:

- **Type**: Categorical (String)
- **Description**: The cardholder’s phone number associated with the transaction. This is often used for verification and security purposes (e.g., two-factor authentication). Anomalies like a sudden change in phone numbers or multiple accounts using the same phone number can indicate fraudulent activity.

42. phone number:

- **Type**: Categorical (String)

- **Description**: This appears to be a duplicate of the Phone Number field. It likely serves the same purpose—tracking the contact number associated with the cardholder. Care should be taken to ensure there is no data redundancy or inconsistency between this and the Phone Number column.

43. Address:

- **Type**: Categorical (String)
- **Description**: The cardholder’s residential or billing address. Any mismatch between the billing and shipping address or discrepancies in address changes might raise flags for fraud detection.

44. City:

- **Type**: Categorical (String)
- **Description**: The city associated with the cardholder's billing or shipping address. Transactions that occur far from this city, especially when involving high-value purchases, can be indicators of fraudulent activity.

45. Customer State:

- **Type**: Categorical (String)
- **Description**: The state in which the cardholder resides or is billed. This field provides geographic context, which is important for detecting anomalies, such as purchases made outside the cardholder's usual state.

46. Country:

- **Type**: Categorical (String)
- **Description**: The country where the cardholder resides or is billed. Transactions made outside the cardholder's home country, especially in high-risk regions, can trigger fraud alerts.

47. Birth Date:

- **Type**: Date
- **Description**: The birth date of the cardholder. This information can be used to assess fraud risk by comparing it with age-related spending patterns or detecting anomalies in the cardholder's behavior (e.g., sudden large purchases by an elderly cardholder).

48. Income Bracket:

- **Type**: Categorical (String)

- **Description**: The cardholder's income range, typically categorized (e.g., Low, Medium, High). This can be used to assess the legitimacy of high-value transactions—transactions that significantly exceed a cardholder's typical spending pattern based on their income bracket might be suspicious.

49. Transaction Channel:

- **Type**: Categorical (String)
- **Description**: This indicates the medium or platform through which the transaction was made, such as online, in-store, mobile app, or phone. Some channels, like online or mobile transactions, are more prone to fraud, and patterns of transactions through a new or unusual channel can raise flags.

50. Transaction Channel :

- **Type**: Categorical (String)
- **Description**: This appears to be a duplicate of the transaction channel column, serving the same purpose. It's important to resolve any discrepancies or duplicates to maintain data integrity.

DWH and models

The diagram you've shared illustrates a star schema design for a fraud detection system, containing dimension tables and a fact table. Here's a description of the schema and an explanation of why a snowflake schema could be beneficial:

Schema Breakdown:

1. **Fact Table (Transaction_Fact)**:
    - Contains the core transactional data related to fraud detection.
    - Key fields include:
        - pk-transaction_id: Primary key for the transaction.
        - fk-transaction_date(sk): Foreign key linking to the Date dimension.
        - fk-response_time(sk): Foreign key linking to the Time dimension.
        - fk-POS Entry Mode(sk): Foreign key linking to POS Entry Mode.
        - fk-card(sk): Foreign key linking to the Card dimension.
        - fk-merchant(sk): Foreign key linking to the Merchant dimension.
        - fk-Fraud(sk): Foreign key linking to the Fraud dimension.

- Other fields like transaction_amount, transaction_status, transaction_type, and Transaction Risk Score are also captured.

2. **Dimension Tables**:
   - **Customer Dimension (Customer_Dim)**: Stores customer-related information (e.g., Customer_ID, First Name, Last Name, Income, etc.).
   - **Card Dimension (Card_Dim)**: Details about the customer's card (e.g., Card Type, Card Expiration Date).
   - **Merchant Dimension (Merchant_Dim)**: Information related to merchants (e.g., Merchant Category, Merchant Name, Merchant Rating).
   - **Time Dimension (Time_Dim)**: Tracks time of transactions (e.g., hours, minutes, seconds).
   - **Date Dimension (Date_Dim)**: Tracks dates (e.g., month, quarter, year).
   - **POS Entry Mode Dimension (POS Entry Mode_Dim)**: Tracks the point-of-sale entry mode (e.g., device type, method of transaction).
   - **Fraud Dimension (Fraud_Flag)**: Captures fraud-related flags and detection methods.

Why Use Snowflake Schema?

1. **Normalization**:
   - The snowflake schema is a normalized version of a star schema, where dimension tables are split into additional tables. This reduces redundancy and improves data integrity.
   - For example, in your schema, Customer_Dim could be further normalized by splitting Address, Country, and State into separate tables, reducing repetition across multiple rows for the same customer.
2. **Space Efficiency**:
   - By normalizing, you store related attributes in separate dimension tables, saving storage space. Snowflake designs are beneficial when dealing with large datasets where denormalization would cause redundancy.
3. **Query Optimization**:
   - Snowflake schemas can provide faster join performance when querying smaller dimension tables (e.g., joining a normalized customer address table with the Customer_Dim). Modern databases are optimized for these normalized joins.
4. **Maintenance and Updates**:
   - Data maintenance becomes easier because updates occur only in one place (for example, updating a merchant category would only require a change in the Merchant_Dim, which is more normalized in a snowflake schema).
5. **Flexibility**:

- As the data model evolves, the snowflake schema provides more flexibility to add or modify attributes in dimension tables without affecting the fact table or related dimensions.

Potential Drawback:

While snowflake schemas improve storage efficiency and maintainability, they can make querying more complex, requiring more joins than a star schema. However, for large-scale, complex systems like real-time fraud detection, the trade-off is often worth it.