Google Cloud

# Fraud Detection in Banking

# Table of contents

# Introduction

This article provides a comprehensive overview of fraud and fraud detection within the banking sector, emphasizing a real-time credit card fraud detection use case. It covers various fraud types, different lines of defense, and diverse detection and protection approaches.

The article also details how banks are leveraging machine learning (ML) to detect transactional fraud, a critical component of their overall fraud prevention strategy.

Through a real-time use case, the article demonstrates the multiple components of a fraud detection system, including data ingestion, feature engineering, feature stores, model training, deployment, monitoring, and MLOps, and illustrates how these function on Google Cloud.

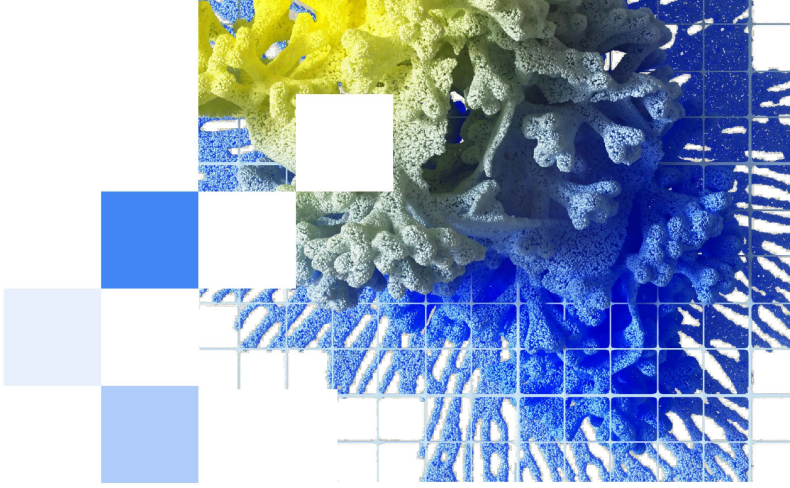
This guide aims to assist in building a scalable real-time fraud detection solution on Google Cloud and is intended for developers, data engineers, data scientists, and solution architects with foundational knowledge of Google Cloud, machine learning, and DevOps.


**Saeed Aghabozorgi**

AI/ML Specialist,
Financial Services

**Polong Lin**

Developer Advocate,
Generative AI



# Fraud types in banking

**Understanding the diverse landscape of fraud within the banking system is crucial. Each line of business (LOB) in a bank encounters distinct fraud risks, shaped by their specific operations and customer demographics. Credit cards, loans, customer KYC (Know Your Customer), deposit accounts, investment accounts, and the ever-present threat of cybercrime each present unique vulnerabilities. Consequently, banks must contend with a wide array of fraud types and use cases:**

## Customer on-boarding

During the process of opening a new account, fraudsters may provide false information to create fake identities. This could be to launder money, obtain credit they don't intend to repay, or engage in other illicit activities. Banks need robust verification processes to combat this.

Notably, with recent advances in generative AI, it is now even easier for fraudsters to generate, edit or fabricate data to impersonate or to create materials that appear genuine when they are not.

## Loan Application Fraud

Loan application fraud occurs when individuals provide false information to obtain loans they don't intend to repay, or not get approved for without those. This can include falsifying income or employment details, misrepresenting assets, identity theft, using straw borrowers, or providing fake addresses. Lenders use various methods to detect and prevent this fraud, but it remains a persistent challenge due to evolving fraud tactics.

## Credit Card payment

Stolen credit card information, obtained through phishing scams, data breaches, or physical theft, can be used to make fraudulent purchases. This can happen online or in person, leaving the victim with unauthorized charges.

## EMT (Electronic Money Transfer)

Scammers use social engineering tactics to deceive victims into sending them money through EMT. They might impersonate a friend in need, a government agency demanding payment, or a lottery claiming a prize requires a fee.

## Money Laundering

Disguising the origins of illegally obtained money by passing it through legitimate financial systems. For example, Google Anti Money Laundering, detects suspicious, potential money laundering activity faster and more precisely with AI.

## Identity Theft

Stealing someone's personal information to open accounts, apply for credit, or make purchases in their name.

## Insider Fraud

Bank employees abusing their position to steal money, manipulate transactions, or engage in other illegal activities.

## Account Take-over

Once fraudsters gain access to a victim's account (through stolen passwords, phishing, or malware), they can drain funds, change account details, or initiate unauthorized transactions, leaving the victim with a financial mess.

## Bank Kiting

In this scheme, an individual writes a check from an account with insufficient funds (Account A) at one bank and deposits it into another account (Account B) at a different bank. Before the check from Account A clears, the individual writes another check from Account B and deposits it into Account A to cover the first check. This process can be repeated multiple times, creating a cycle of bad checks.

## Check Fraud

Check fraud is a pervasive issue that encompasses various deceptive tactics aimed at illicitly obtaining funds. One common method involves the creation of counterfeit checks, which are fabricated documents designed to mimic legitimate checks. These counterfeit checks may feature forged signatures, altered payee information, or fictitious account details, all intended to deceive financial institutions and individuals. Another prevalent technique is the alteration of legitimate checks. This can involve changing the payee name, modifying the amount payable, or manipulating other key details on the check. These alterations are often subtle and difficult to detect, making them a potent tool for fraudsters. Furthermore, criminals may attempt to write checks from closed accounts, exploiting the delay between account closure and the updating of banking systems. This allows them to temporarily withdraw funds that are no longer available, causing financial harm to both the account holder and the recipient of the fraudulent check.

# Defining the Lines of Defense

**Fraud detection in banking is a multi-layered approach, employing various techniques to identify and prevent fraudulent activities.**

**These techniques can be broadly classified into two categories:**

## Fraud Prevention

Aims to prevent fraudulent activities before they occur. This is achieved through methods like multi-factor authentication, device verification, and user behavior analysis. By implementing these measures, banks can create a more secure environment and deter potential fraudsters.

## Fraud Detection

Focused on identifying occurrences of fraud after they have happened, usually through transaction monitoring and analysis. This involves analyzing patterns in spending and account activity to flag suspicious transactions. Tools like rule-based systems and anomaly detection algorithms are often used.

# Methodologies

The specific methodology employed by a bank will depend on various factors, such as its size, customer base, and technological infrastructure. However, some common methods used in fraud detection include:

## Machine Learning

Powerful algorithms capable of analyzing vast amounts of data and identifying patterns that may indicate fraudulent activity. This allows for real-time detection and quicker responses to potential threats.

## Data Analytics

Scrutinizing customer data, transaction history, and other relevant information to identify anomalies and red flags. This can involve techniques like statistical analysis, predictive modeling, and data visualization.

## Biometrics

Utilizing physical characteristics, such as fingerprints and facial recognition, to authenticate users and prevent unauthorized access. This adds an extra layer of security to sensitive financial transactions.

## Risk Scoring

Assigning a risk score to each customer or transaction based on a variety of factors. This helps prioritize resources and focus on areas with a higher probability of fraudulent activity.

## The Human Element

Despite technological advancements, the human element remains vital in fraud prevention. Continuously training employees to recognize potential scams and suspicious activities is crucial. Additionally, knowing your customer, maintaining open communication with customers and encouraging them to report suspicious activity can significantly contribute to timely fraud detection.

## The Importance of Collaboration

Combating fraud effectively requires collaboration between banks, regulatory authorities, and law enforcement agencies. Sharing information and best practices can help ensure coordinated efforts in tackling evolving fraudulent schemes.

# Fraud Detection in Credit Card Payments

With the increasing prevalence of real-time and online payments, the need for effective fraud detection solutions capable of providing real-time, actionable alerts has never been greater. This applies to various transaction types, including credit card payments, ATM withdrawals, money transfers, and more. There are numerous approaches to building fraud detection applications. In this section, we will focus on credit card payment fraud detection, but the underlying concepts can be readily adapted to other real-time fraud detection use cases such as electronic money transfers (EMTs), wire transfers, and ATM transaction fraud.

Prior to delving into the various methods and solutions for credit card fraud detection, let's examine some common examples of fraud in payment systems. Broadly speaking, credit card fraud patterns can be classified into the following categories:

## Transaction-Based Patterns

- Unusual Spending Amounts or Frequency: Sudden large purchases or an increase in the number of transactions can be indicators of fraud, especially if they deviate from the cardholder's typical spending habits.
- Multiple Transactions in a Short Time: Rapid-fire transactions across different merchants or locations could indicate a stolen card being used quickly before it's reported.
- Purchases from Unfamiliar Locations: Transactions from countries or regions the cardholder doesn't frequent can be a red flag.
- Online Purchases vs. In-Store: A sudden shift to online purchases or a high volume of online transactions can be suspicious, as online transactions are generally easier for fraudsters to carry out.
- Declined Transactions Followed by Successful Ones: This could indicate a fraudster testing a stolen card or trying different card details.

## Cardholder-Based Patterns

- New Accounts with High Spending: Newly opened accounts that immediately engage in high-value transactions can be a sign of fraudulent activity.
- Multiple Cards from the Same Address: Fraudsters might obtain multiple cards using the same address.
- Changes in Personal Information: Sudden changes to the cardholder's name, address, or contact details could be a sign that their identity has been compromised.
- Device Fingerprinting: Each device has unique characteristics (OS, browser, IP address, etc.) that create a "fingerprint." If multiple accounts are originating from the same device, suggesting potential fraud like account takeover or fake account creation.

## Merchant-Based Patterns

- High-Risk Merchants: Certain types of merchants, such as those selling electronics or jewelry, are more likely to be targeted by fraudsters.
- International Merchants: Transactions with merchants in countries known for fraudulent activity can be suspicious.
- Suspicious Merchant Category Codes (MCCs): Some MCCs are associated with higher fraud risk, such as those for gambling or adult entertainment.

## Emerging Patterns

- Account Takeover Fraud: Fraudsters gain access to existing accounts through phishing or other methods and then change the account details to make fraudulent transactions.
- Synthetic Identity Fraud: This involves creating fake identities using a combination of real and fabricated information to open new accounts and commit fraud.
- Card-Not-Present (CNP) Fraud: This refers to fraud that occurs without the physical card being present, such as online or phone transactions.

Learning from all these patterns over years, banks employ a range of specialized techniques to detect and prevent fraudulent transactions. These key methods include:

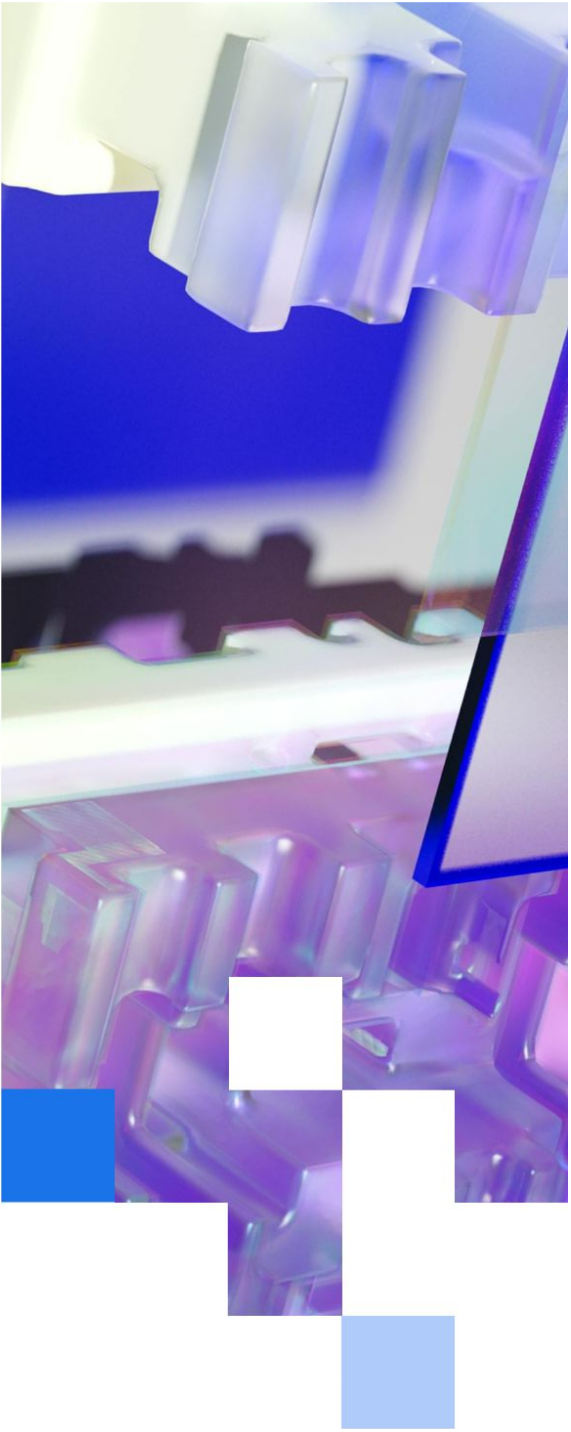
| Category | Technique | Description |
|---|---|---|
| Data-driven Techniques | Machine Learning | Algorithms analyze data to identify anomalies indicative of fraud. |
| | Rule-based Systems | Pre-defined rules flag suspicious activity based on specific criteria. |
| | Statistical Analysis | Outlier detection and regression analysis help identify unusual spending patterns. |
| | Network Analysis | Identifying connections between accounts, devices, and IP addresses can uncover fraud rings. |
| Transaction Verification | Security protocols (e.g. 3-D Secure) | Additional authentication layer requiring a verification code for online transactions (e.g. OTP Number). |
| | Address Verification Service (AVS) | Compares billing and registered addresses, flagging inconsistencies. |
| | CVV/CVC Verification | Ensures the physical card is present for online transactions by checking the security code |
| Behavioral Analysis | User Behavior Analytics (UBA) | Monitoring user behavior patterns to identify suspicious deviations (type speed, errors, events). |
| | Biometrics | Fingerprint and facial recognition add security to transactions. |
| Collaborative Efforts | Fraud Detection Networks | Sharing information about known fraudsters and tactics to prevent fraud. |
| | Customer Reporting | Encouraging customers to report suspicious activity for faster detection. |
| Other techniques | Multi-factor Authentication | Adds additional security layers beyond passwords. |
| | Device Fingerprinting | Identifying specific devices used for transactions to detect suspicious activity. |
| | Active Account Monitoring | Banks actively monitor accounts for unusual activity and may contact cardholders to verify transactions. |

Naturally, most banks incorporate a combination of these techniques to effectively combat fraud and protect both banks and cardholders from financial losses. That is, machine learning models, rule-based systems, human review, or a combination of these methods can be employed to identify fraudulent patterns. In the following section, we will concentrate on the machine learning approach.

# Machine learning approaches

From a data science point of view, the focus usually is on machine learning techniques or algorithms. Various machine learning approaches are employed for credit card fraud detection, each with its strengths and limitations:

| Category | Algorithm | Description | Pros | Cons |
|---|---|---|---|---|
| Supervised Learning | XGBoost (Extreme Gradient Boosting) | An ensemble method combining multiple decision trees for high performance. | High accuracy, efficiency, scalability, handles missing values, provides feature importance ranking. | Can be prone to overfitting, requires careful hyperparameter tuning, may not be easily interpretable. |
| | Logistic Regression | Predicts the probability of fraud based on pre-labeled data. | Easy to interpret and implement | May not capture complex relationships in data. |
| | Support Vector Machines (SVMs) | Effective for fraud detection in high- dimensional data with complex relationships. | Good at handling high-dimensional data and complex relationships. | Computationally expensive, requires careful parameter tuning. |
| | Decision Trees | Interpretable and handles categorical data effectively. | Interpretable, handles categorical data effectively. | Prone to overfitting, may not capture complex relationships. |
| | Random Forests | Combines multiple decision trees to improve accuracy and reduce overfitting. | Robust and powerful. | Difficult to interpret. |
| Unsupervised Learning | K-Means Clustering | Identifies groups of similar transactions to highlight anomalies. | Fast and scalable. | Requires prior knowledge of the number of clusters. |
| | Isolation Forests | Isolates anomalies by repeatedly dividing data points. | Good for high-dimensional data, handles unknown fraud patterns well. | Computationally expensive. |
| | Autoencoders | Learns patterns in data and reconstructs normal transactions. Deviations from the reconstructed version indicate fraud. | Handles complex relationships in data. | Requires large amounts of training data, difficult to interpret. |
| Deep Learning | Convolutional Neural Networks (CNNs) | Excels at extracting features from data, effective for detecting fraud patterns in large datasets. | Effective for detecting fraud patterns in large datasets. | Requires significant computational resources and expertise. |
| | Recurrent Neural Networks (RNNs) | Captures temporal dependencies in data, suitable for analyzing transaction sequences. | Suitable for analyzing transaction sequences and identifying patterns over time. | Training can be complex and computationally expensive. |
| Hybrid Approaches | Combining multiple algorithms | Leverages strengths of different techniques for improved performance (e.g., anomaly detection with supervised learning). | Can improve overall performance and accuracy. | Requires careful selection and integration of algorithms. |
| | Ensemble Learning | Combines multiple models into a single ensemble for improved accuracy and robustness. | Improved accuracy and robustness compared to individual models. | Increased complexity and potential for overfitting if not managed carefully. |



To tackle this issue, data scientists in banks often begin with an efficient algorithm such as XGBoost or Random Forest, then supplement it with neural network or deep learning approaches. Combining different techniques and customizing models often yields the most effective results. Banks also utilize unsupervised algorithms for profiling and anomaly detection, particularly for channels, industries, or geographical locations that are new to them and lack sufficient labeled data.

In the past, data scientists and machine learning practitioners had to manually develop the aforementioned algorithms for fraud detection. This process was time-consuming and required specialized knowledge and expertise. However, advancements in machine learning have led to the development of out-of-the-box algorithms that can be easily integrated into existing fraud detection workflows. These algorithms come with pre-defined hyperparameters and can be further tuned to optimize performance, making it easier and faster to build and deploy machine learning models.

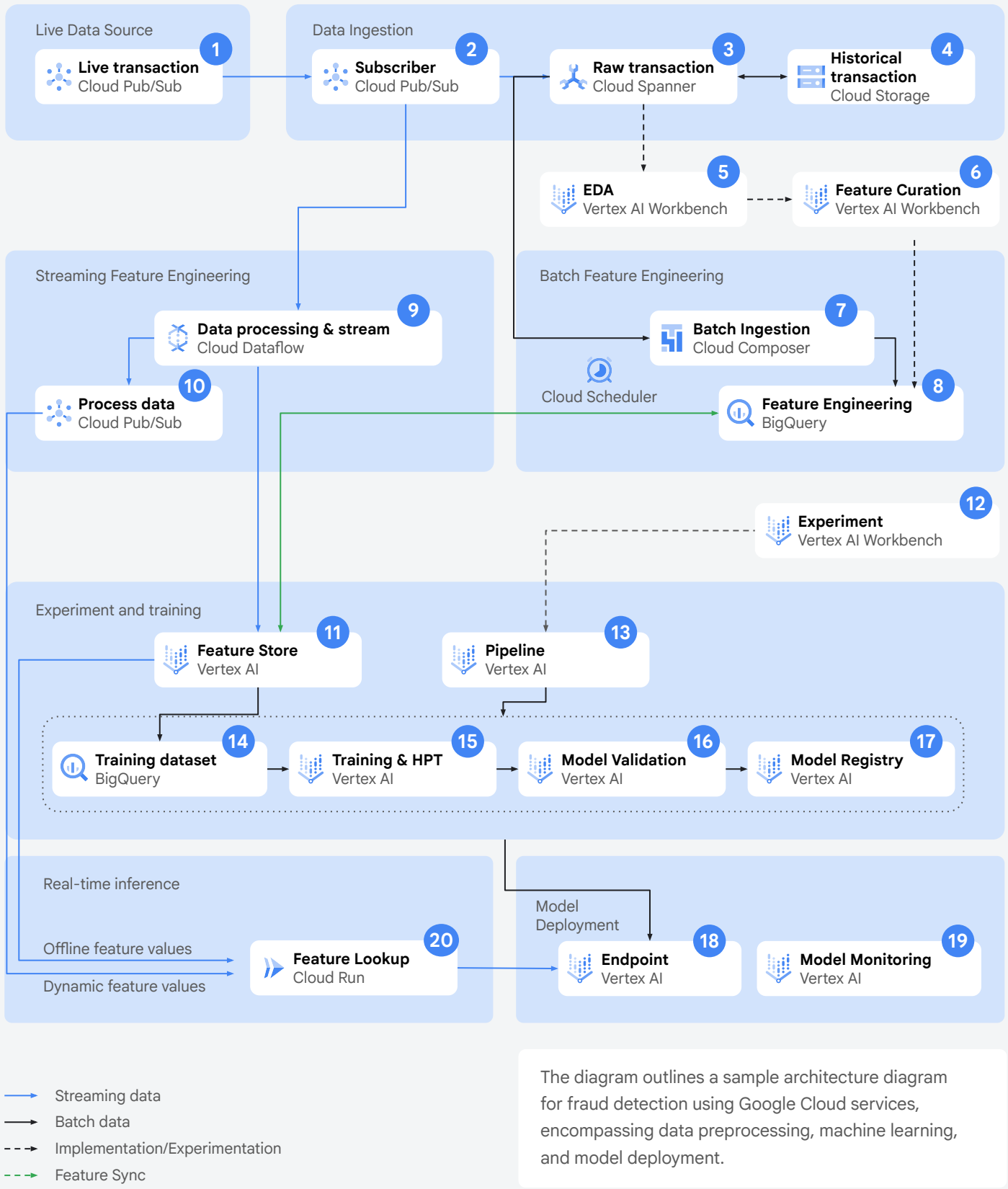
For example, Google Cloud offers a variety of classification algorithms through the BigQuery ML service, including boosted tree model, logistic regression, XGBoost, deep neural networks, or AutoML. These algorithms are designed to handle large datasets and provide accurate and reliable results. The benefits of using out-of-the-box algorithms include speed, accuracy, and ease of use. They can save time and resources, and accelerate the development process.

# Sample Architecture

The following is a sample architecture for fraud detection in Google Cloud, focusing on transactional data. This blueprint is designed to handle large volumes of data in real time, effectively identifying fraudulent transactions across various banking activities. It encompasses essential components for data ingestion, processing, feature engineering, model training, deployment, and monitoring. This architecture offers a solid foundation for building a scalable real-time fraud detection solution on Google Cloud.

This architecture represents the minimum requirements and tooling for both build and operationalization of the model. Specific use cases may necessitate additional or modified components, depending on the transaction types, such as, Money transfers in online banking, ATM withdrawals, Credit card payments at point-of-sale (POS) terminals, or other financial transactions.

The sample architecture illustrated below outlines the flow of data and processes involved in real-time credit card fraud detection:




The diagram outlines a sample architecture diagram for fraud detection using Google Cloud services, encompassing data preprocessing, machine learning, and model deployment.

## Here's a breakdown of the process:

### 01. Data Ingestion & Storage

- Live Data Source (1): Real-time transaction data is streamed into Cloud Pub/Sub.
- Data ingestion (2): The transactional data is ingested into a database via Cloud Pub/Sub.
- Raw Data Storage (3 & 4): The raw transaction data is stored in Cloud Spanner (3) as a transactional database. Optionally, historical transaction data is ingested from Cloud Storage (4).

### 02. Data Preprocessing & Feature Engineering

- Batch Feature Engineering (5-8):
  - Exploratory Data Analytics (5): Initial exploration and analysis of raw data using Vertex AI Workbench.
  - Feature Curation (6): Features are designed and selected within Vertex AI Workbench.
  - Batch Ingestion (7): Cloud Composer orchestrates a pipeline to read data from Cloud Spanner, transform it, and then load the results into BigQuery.
  - Feature Engineering (8): Features are generated using BigQuery ML, and stored in the offline feature store for training.
- Real-time data is processed and features are generated using Cloud Dataflow (9). Then, the processed data is sent to Cloud Pub/Sub (10) for inference downstream.
- Feature Store (11): Features are stored and managed in Vertex AI Feature Store. Additionally, the features get synchronized with the offline feature store in BigQuery.

### 03. Model Training & Evaluation

Experiment & Training (12-17):

- Experiment (12): Experiments are conducted using Vertex AI Workbench to build and train fraud detection models, before training at scale.
- Pipeline (13): A machine learning pipeline is built on Vertex AI to automate the training and deployment process.
- Training Dataset (14): Training data is pulled from BigQuery, which in this case is the offline feature store.
- Training & HPT (15): Models are trained and hyperparameter tuning is performed using Vertex AI.
- Model Validation (16): Trained models are validated on Vertex AI.
- Model Registry (17): Successful models are registered in Vertex AI Model Registry.

### 04. Model Deployment & Monitoring

- Model Deployment (18 & 19):
  - Endpoint (18): The best performing model is deployed to a Vertex AI Endpoint for real-time predictions.
  - Model Monitoring (19): Deployed models are continuously monitored on Vertex AI for performance and potential drift.
- Real-time Inference (20):
  - Feature Lookup (20): For each incoming transaction, corresponding calculated features in real-time (dynamic features), and features retrieved from the online Feature Store are combined using Cloud Run.
  - The model deployed on the Vertex AI Endpoint then uses these features to provide real-time fraud predictions.

In summary, this architecture leverages various Google Cloud services to create a robust end-to-end fraud detection system, covering data ingestion, preprocessing, feature engineering, model training, deployment, and monitoring. In the following sections, each component of the architecture are discussed further.

# Architecture Components

## Data Requirement

The availability of relevant data is crucial for effective credit card and payment fraud detection in the banking sector. Today's banks typically gather extensive data, often exceeding what is strictly necessary for this purpose. The following table illustrates common data sources utilized in online banking fraud detection, accompanied by specific examples:

| Data Source Category | Sample Data Points | How It Helps Detect Fraud |
|---|---|---|
| **Customer demographic information** | Nationality, gender, DOB, address, income, occupation, spending behavior, education, type (company/consumer), join date, assets value, events (change of address, email, etc) | It is used as the base information that helps with unusual behaviors. For example, any update to the customer information might be a signal for fraud activity, such as profile updates, contact updates, etc. |
| **Banking accounts data** | Billing account (checking/saving) connected with credit card, balance, transaction history, card type (debit/credit), active country, card or payment events (validity) | Inconsistent spending patterns or unusual changes in the connected account to the customer/credit card can be an indicator of a compromised account. Additionally, any update to the card or payment information. |
| **User authentication information** | Username, password, account number, events (change of password, username, email, etc) | Unusual login locations, frequent password changes, might indicate compromised accounts or unauthorized use. |
| **Device Information** | Device type, operating system, browser type, screen resolution, unique device ID | Linking devices to users helps track if multiple accounts are used from the same device (potential account takeover). |
| **Transactional Data** | Date, time, amount, merchant, location, payment method | Anomalies like large transactions, international purchases, specific merchants, or multiple transactions in a short period can signal fraud. |
| **Behavioral Biometrics** | Typing speed, mouse movements, scrolling patterns | Deviation from a user's typical behavior (e.g., unusually slow typing) may indicate someone else is using the account. |
| **Network Information** | IP address, geolocation, ISP, network traffic | Unusual network traffic patterns (e.g., sudden spikes) or connections from high-risk countries can be signs of fraudulent activity. |
| **External Data Sources** | Public records, social media, merchant blacklists, rules | Verifying user identity against public data or checking if they're associated with known fraudsters adds another layer of security. Also, using signals from 3rd party fraud systems can be beneficial. |

Some of this data is transformed into features for machine learning models. Other data, specific to individual users, is only used in rule-based systems. For instance, a high "number of transactions in the last 15 minutes" could indicate fraud and is therefore an input feature for the ML model. "Device Count," representing the number of unique devices linked to an account during a transaction, can also be a potential flag. However, this may not be suspicious for users who genuinely use multiple devices. As a result, combining ML-based and rule-based detection helps reduce false positives in such cases. As another example for rule-based data, the "MAC-address" of a user's typical devices, while unsuitable as an ML input feature, can serve as a secondary security layer for specific customers.

**Now, let us look at a simple example of raw payment data in banking:**

```
{
  "transaction_id": "TXN1234567890",
  "device_id": "POS456",
  "merchant_id": "MERCH123",
  "timestamp": "2024-07-17T15:26:00Z",
  "transaction_type": "purchase",
  "amount": 19.99,
  "currency": "USD",
  "card_data": {
    "card_number": "411111XXXXXX1111",
    "expiry_date": "08/26",
    "cvv": "123"
  },
  "authorization_code": "AUTH789",
  "additional_data": {
    "invoice_number": "INV001",
    "customer_id": "CUST1234"
  }
}
```

You can think of this data (.json) as the data that comes to the issuing bank to verify the card details and approve or decline a transaction. Of course these payloads can be very different, depending on the type of transaction and also the channel, whether it is happening on POS, ITM, Mobile APP, or Web App.

While banks typically possess vast amounts of data, this information may not be readily accessible on modern services on Google Cloud, necessitating proper migration, loading, or transfer. Additionally, data augmentation plays a crucial role in enhancing fraud detection capabilities.
This can involve leveraging raw data from external sources or incorporating existing fraud scores as supplementary signals. Notably, generative AI is emerging as a powerful tool for augmenting fraud data and simulating synthetic scenarios, offering several key benefits:

### Data Augmentation

Generative AI models can create synthetic data that closely resembles real-world fraudulent patterns. This enhances training datasets for machine learning algorithms, particularly when real-world data on fraudulent activities may be limited due to its sensitive nature. This enriched data can improve the accuracy and robustness of fraud detection models.

### Fraud Scenario Generation

Generative AI can be used to generate realistic scenarios of potential fraud, allowing institutions to test and improve their detection systems. This proactive approach can help identify vulnerabilities and anticipate new threats before they impact real-world operations.

### Fraud Investigation

Generative AI can analyze existing fraudulent transactions and generate similar data points, providing insights into the methods and motives behind the fraud. This helps investigators understand what fraudsters do and develop strategies to prevent future occurrences.

### Reduced Bias

Generative AI models can help reduce bias in fraud detection algorithms by generating data that reflects the diversity of population groups and behavior patterns. This can lead to fairer and more accurate identification of fraudulent activity across various demographics.

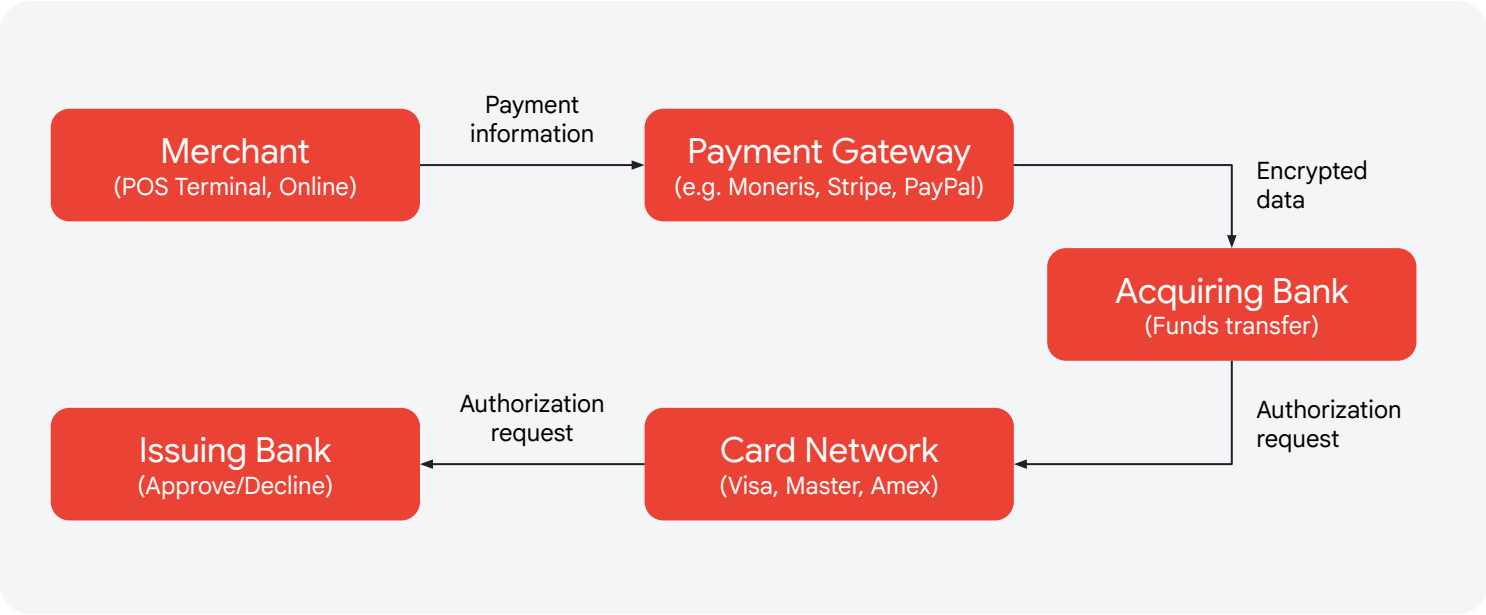
# Data ingestion

As mentioned before, data used for fraud detection comes from different sources with various frequencies. Some might be streamed (e.g. transactional data), some might be static/historical data (e.g. demographic information about customers or labeled data) or some might be data generated or processed as a batch daily/weekly (e.g. customer's average amount of transactions in a week).

Let's first look at the live data ingestion process in this section. The simplest data can be transaction data, as we discussed before, for example, "a customer spending X amount at a point-of-sale terminal Y", being streamed to the bank. However, the source of data in the banking system can go beyond transactional data though, for example, the type of device used by customers, geographical information, type of transaction and interactions (e.g. user- behavior in the merchant website), login information, etc.

Banks have different mechanisms for ingestion of these data in their on-prem or cloud-based environments.

Specifically, in payment processing, first, a merchant's POS terminal sends encrypted transaction data (authorization requests, merchant details, and card information) to a payment gateway, which acts as a secure link between the merchant and their acquiring bank. The acquiring bank then forwards an authorization request to the relevant card network (e.g., Visa, Mastercard, Amex). This network facilitates communication with the cardholder's issuing bank, which verifies the card details, available funds, and credit limits before approving or declining the transaction. The authorization response is relayed back through the network to the acquiring bank, then the payment gateway, and finally to the merchant. For approved transactions, the acquiring bank initiates the settlement process, transferring funds from the issuing bank to the merchant's account.



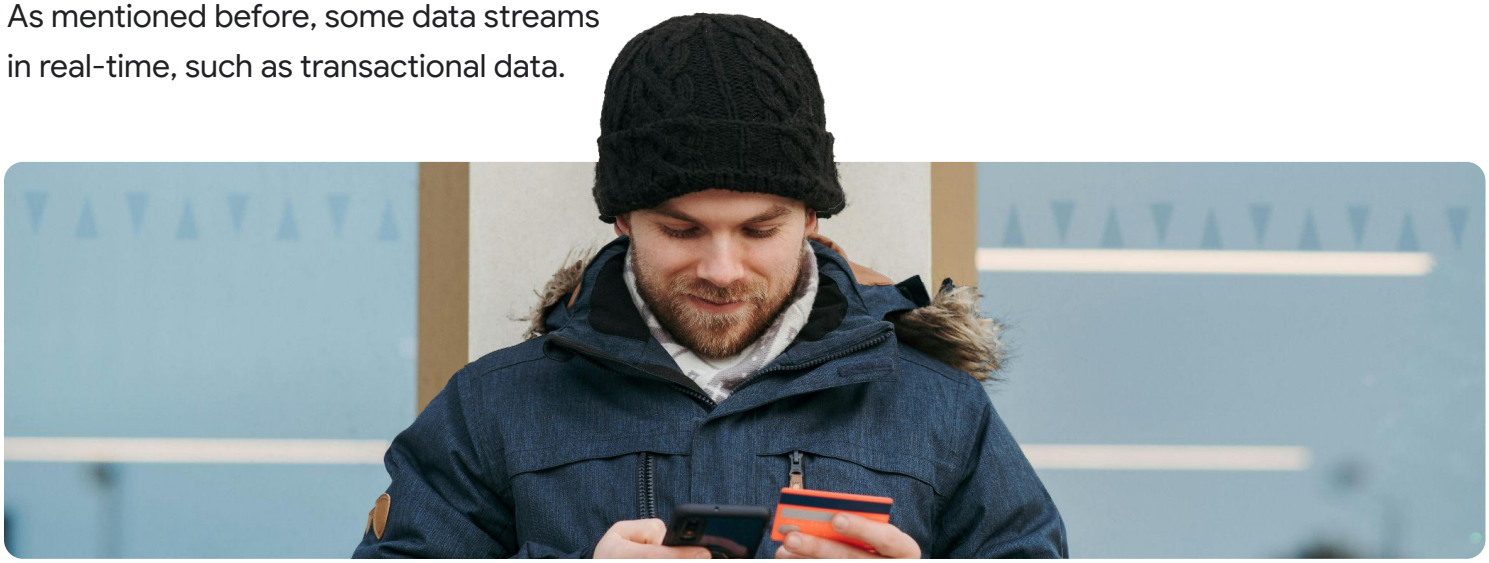
Please note that fraud detection happens at multiple stages in the transaction process, at the payment gateway, acquiring and issuing banks. As soon as the acquiring bank receives the transaction data, it runs through its fraud detection system. For example, the system assesses the merchant's risk profile, considering factors like their industry, transaction history, and chargeback rate. It also happens on the issuing bank. For example, the issuing bank verifies if the transaction is a duplicate of a previous one. Furthermore, card networks like Visa and Mastercard have their own fraud detection systems that monitor transactions across their networks. They also share information about fraudulent activities and suspicious patterns with member banks.

Banks employ various robust mechanisms on the consumer side to receive, store and process streaming data. In the case of Google Cloud, for instance, data can be ingested into a Google Cloud project via publishing into Pub/Sub (1), a messaging service enabling scalable, asynchronous communication between applications.

As mentioned before, some data streams in real-time, such as transactional data. This type of data provides a continuous flow of information that enables immediate analysis and response to suspicious activities. On the other hand, some data remains relatively static or historical, like demographic information about customers or labeled data. This type of data serves as a valuable foundation for understanding customer behavior patterns and identifying potential red flags.

Furthermore, there is also data that is generated or processed in batches, typically on a daily or weekly basis. An example of this could be a customer's average amount of transactions in a week. This data provides insights into behavioral patterns over time, helping to establish baselines and detect anomalies that may indicate fraudulent activity.

Each of these data sources has its own unique characteristics and can contribute valuable insights to the fraud detection process. By combining data from different sources and frequencies, organizations can create a comprehensive view of customer behavior, identify suspicious patterns, and mitigate the risk of financial losses due to fraud.



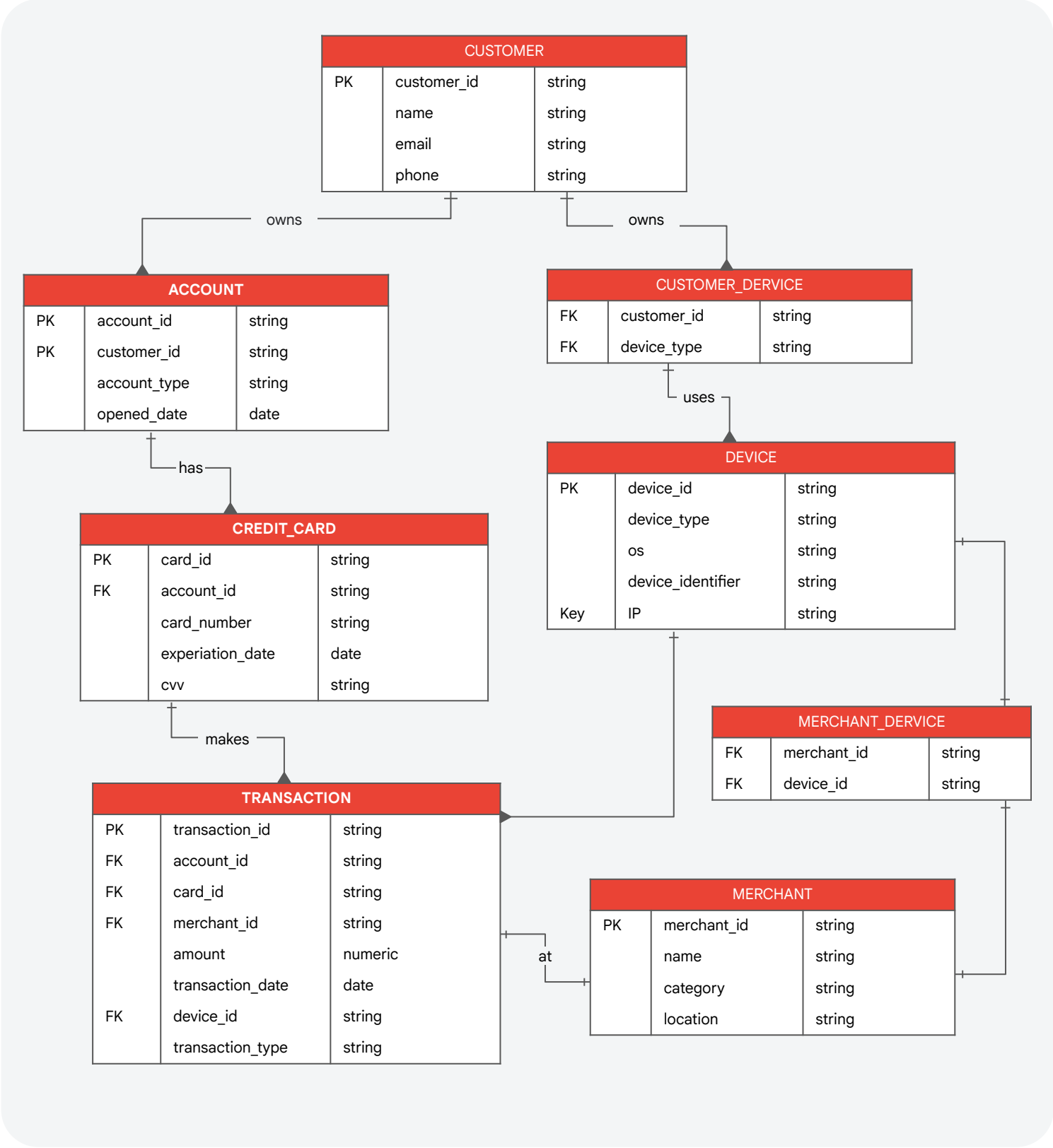
## Here is an example of these type of data:

| Data Type | Example | Source | Frequency |
|---|---|---|---|
| Streamed Data | Transaction amount, merchant category code, location | Point-of-sale terminals, online payment gateways | Real-time, per transaction |
| | IP address, device ID | Network logs, transaction metadata | Real-time, per transaction |
| Static/ Historical Data | Customer age, permanent country, account history, tenure, asset value | Customer onboarding information, banking records | Onboarding, occasional updates |
| | Labeled fraud data (past confirmed fraud cases) | Internal fraud reports, chargebacks | As cases are identified |
| Batch Processed Data | Average weekly transaction amount | Transaction history aggregated daily/weekly | Daily, weekly |
| | Number of failed login attempts per week | Authentication logs aggregated daily/weekly | Daily, weekly |
| | Spending pattern changes over time | Transaction history analyzed over longer periods | Weekly, monthly |

The storage choice of data for transactions in Google Cloud can be a database such as Cloud Spanner which can easily scale to handle large amounts of data and high write throughput, AlloyDB which is specifically designed for high-performance transactional workloads, or Bigtable which is optimized for large analytical and operational workloads. In the proposed architecture, just for sake of demonstration, for the Data Ingestion section, the streamed raw data is stored through a Cloud Pub/Sub subscription (2) into Cloud Spanner (3). You might have many tables in the database that collectively represent your historical data. At minimum, a transaction table, which contains historical transactions, either published by live transactions, or from another source such as historical data in Google Cloud Storage (4):

| Row | transaction_id | card_number | transaction_amount | transaction_datetime | currency_code | merchant_id | device_id | channel |
|---|---|---|---|---|---|---|---|---|
| 1 | 3fb80c8b-72a4-4243-a0c3-db8... | 7704747631488286 | 69.09 | 2023-12-16T09:40:19 | CAD | 9dbf2c80-9abf-41e7-8a34-e52... | ece19de2-00ac-4435-8ea3-4a7... | online-website |
| 2 | b15e098d-2ebc-4649-970d-22... | 9062369731630170 | 87.28 | 2023-06-06T12:03:16 | CAD | c476fdf2-d17f-4d68-b2b1-2000... | bf441e9b-47c4-4ecd-b373-b17... | online-mobile |
| 3 | 0453b4c1-5da6-45c8-9cea-e5b... | 7606134327644398 | 133.89 | 2023-03-13T09:22:45 | CAD | 9dbf2c80-9abf-41e7-8a34-e52... | e2d5c7f9-d205-43f0-a4ad-277... | in-Store-mpos |
| 4 | f9b28684-caae-490b-943f-9fc4... | 9521340057795409 | 130.1 | 2023-01-10T14:39:34 | CAD | aef8b7f3-7151-4a02-8b88-f114... | 7bbf1d2a-4550-48d4-a106-f03... | online-website |
| 5 | 0b252aa3-1e00-4059-a485-ce6... | 6038182140034377 | 139.7 | 2023-04-30T04:06:25 | CAD | cc07be4d-0746-4117-ae0d-fb1... | 607516e8-5b7b-471b-808e-ff2... | in-Store-pos |
| 6 | 92daeb3d-7c79-4e42-93b9-b0b... | 6840958879875419 | 167.58 | 2023-06-04T02:37:52 | CAD | aef8b7f3-7151-4a02-8b88-f114... | 806ccc32-b414-4e53-849e-9ed... | online-website |
| 7 | c939272e-2f30-4ef5-aaa0-987... | 5759440056787611 | 34.33 | 2023-07-12T22:51:13 | CAD | 6f49f374-81fa-47ff-bc50-1234f... | 94730b0d-4648-4bb7-8ec8-a5e... | online-mobile |
| 8 | 2b87afbc-e0f9-400e-b5bd-044... | 4554053981933288 | 32.98 | 2023-09-16T06:43:47 | CAD | 0fb6bb20-38b6-4b15-9bd6-53d... | 5876d32b-6a7a-42cb-b6c8-fcf... | in-Store-pos |
| 9 | b38e405b-85e7-4811-ab30-c91... | 4035645947166296 | 40.61 | 2023-05-02T18:46:24 | CAD | 9dbf2c80-9abf-41e7-8a34-e52... | ac804d0d-59c3-499e-972b-b5f... | online-mobile |
| 10 | 718441a9-4b5f-4a1b-8912-877... | 5045090672855264 | 51.48 | 2023-12-06T02:02:23 | CAD | 8b2b1ac8-31d0-40db-bdf6-32d... | 57a9db2c-c845-4d71-9f10-2d2... | in-Store-mpos |

Below is a basic Entity Relationship Diagram (ERD) illustrating the relationships between entities within credit card payment systems. Please note that the number of tables and relationships between the tables in the banks, are much more complicated than this diagram though:

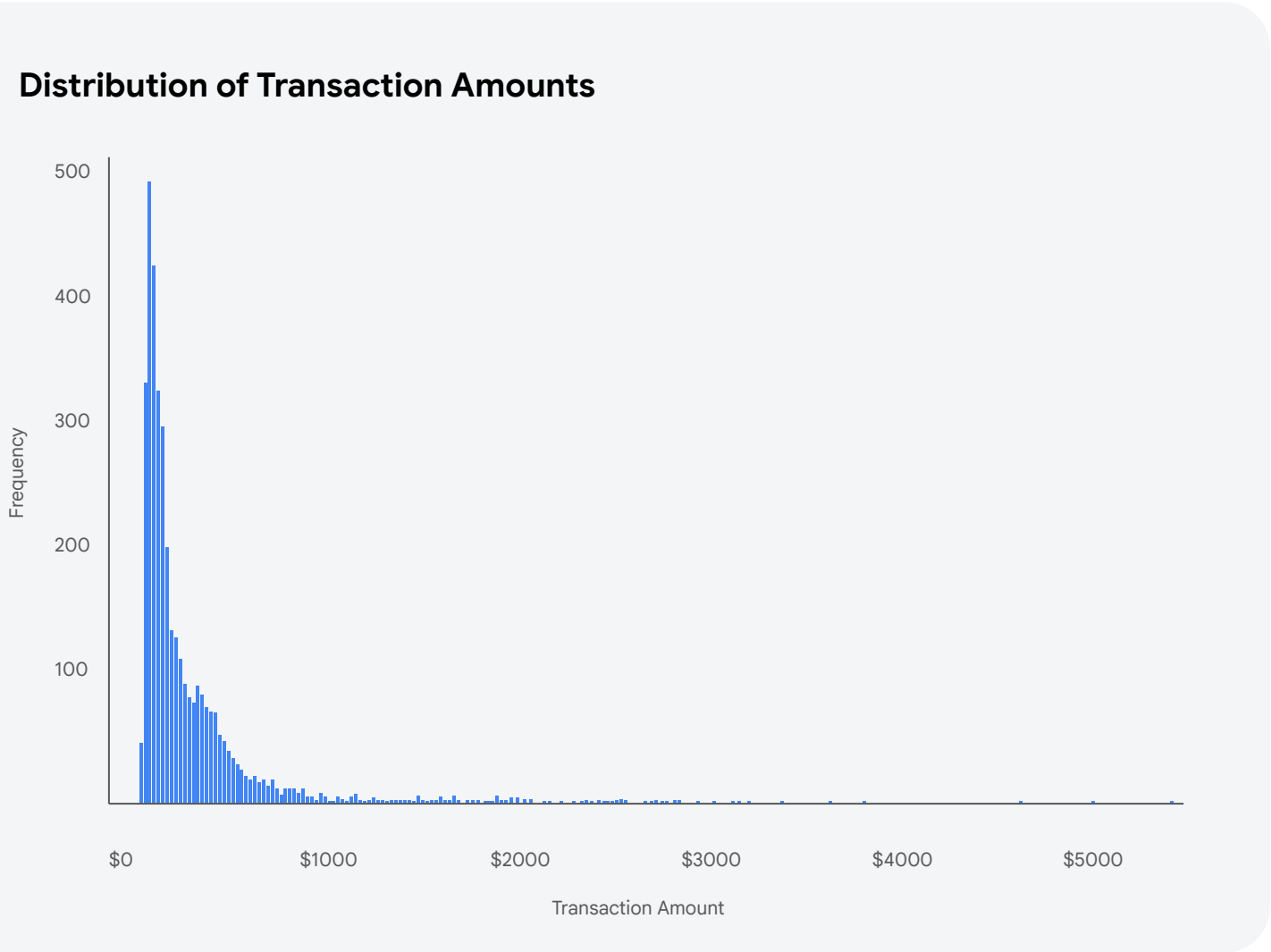


# Exploratory Data Analysis (EDA)

Similar to most machine learning tasks, the next stage involves comprehending the data (5). This typically entails running queries and creating interactive plots to explore the data.

Understanding the statistical attributes of data for feature engineering is the primary goal of this step. For instance, comprehending the total number of transactions, the proportion of fraudulent transactions, customer-level aggregates of transaction data, and the distribution of fraud across other columns, such as transaction amount, is crucial. This information helps in understanding the nature of the data and identifying potential patterns and relationships that can be leveraged for fraud detection.

Here is a very simple example of visualization to better understand the distribution of transactions amounts on some sample data for a short period of time:



Another goal of this step is to find any issue with the data. This includes identifying missing data, unbalanced data, wrong data, etc. Missing data can be imputed using various techniques such as mean, median, or mode. Wrong data can be corrected or removed depending on the situation.

Some data issues can be detected in this step, but get addressed programmatically through a ML pipeline, for example, the imbalanced issue, which is crucial for building effective models.

**For example, to understand the imbalance issue, it is recommended to take 2 steps:**

### Quantification

Determine the ratio between fraudulent and non-fraudulent transactions. This helps you gauge the severity of the imbalance.

### Visualization

Use histograms or bar plots to visually illustrate the class distribution. This can provide further insights into the data.

**It's important to note that this step usually happens in the build phase, similar to training and feature engineering, and not in the production phase. Later, in the MLOps section, we will explain how these steps would be orchestrated through a mature MLOps practice.**

There are multiple tools for EDA in Google Cloud, however we recommend using Vertex AI Workbench as a single notebook-based development environment for connecting to the source of data, statistical description analysis, and visualization. Vertex AI Workbench offers notebooks and built-in integrations with multiple sources of data. You can select your data from BigQuery (or other sources such GCS, Dataproc, etc) and run your exploratory analysis with Python, SQL, or Spark in the notebook effortlessly. Also, you can leverage generative AI capabilities to write codes for analysis of data and visualization. For example, Gemini in BigQuery comprehends the structure and relationships within your data, so you can receive tailored code suggestions from a straightforward natural-language prompt. For example, you can ask it to:

> "Generate a SQL query to calculate the total number of fraud transactions for a specific location."
>
> "Write Python code to correlate between the number of fraud cases and industry using pandas"

The output of EDA is basically knowing the needed features for training, possible cleaning processes (dealing with missing values, dropping columns), and required transformations through feature engineering processes.

# Feature Engineering

Based on exploratory data analysis, at this point you may have some ideas on what kinds of features to create. So the task becomes how can you create these features at scale, and in a way that can be automated in a pipeline?

At the highest level, architecturally you need two pipelines: A pipeline to generate features that are calculated in batch (6 to 8), and a pipeline for the features that are calculated in real-time (streaming) (9 and 10).

# Batch feature engineering

During the batch feature engineering phase, tasks like data aggregation, outlier removal, missing value imputation, and numerical column scaling are performed. The computed features are then stored and regularly updated in an offline feature store, primarily for training purposes. Google Cloud offers tools like Vertex AI Workbench (6) for curating feature engineering logic and BigQuery (8) for executing the feature calculations. This approach offers the advantages of processing data where it resides (eliminating data movement) and leveraging the enterprise-level performance of BigQuery for feature calculation. Additionally, advances in generative AI, like Gemini in BigQuery, helps you write and modify SQL or Python code using natural language prompts, which can reduce the cycle time of curating features, and data pipelines.

Feature values used in fraud detection, such as the number of card transactions within a specific timeframe, are constantly changing and require frequent recalculation. The frequency of these calculations can vary; for instance, while the number of transactions per card might be calculated daily, the number of detected fraud transactions per merchant might only be calculated weekly.

Similarly, changes to customer information like email addresses might only be checked monthly. To manage this dynamic feature engineering process, a task orchestrator and scheduler are essential. Tools like Cloud Composer (7) or Dataflow can help formalize and orchestrate these pipelines, ensuring that feature calculations are performed consistently and on schedule across different environments.

**Note:** The optimal toolset for feature engineering will depend on several factors, including the source of your existing data, your organization's preferred technology, and the skill sets of your team. For instance, if you have experience with Spark, Dataproc might be a suitable choice. Similarly, if you've worked with Beam, you might opt for Dataflow. This principle also applies to the orchestrator: you could consider using Cloud Composer, Vertex AI Pipelines, or even Cloud Workflows.

# Streaming feature engineering

The key objective of this step is to process real-time data and calculate feature values with minimal latency. These features must be calculated dynamically to be available for the ML model. For instance, when a transaction occurs, in milliseconds, we need to calculate the average number of transactions made by the customer in the last 10 minutes. The calculated feature values will be forwarded to the Vertex AI endpoints for prediction downstream and simultaneously pushed into feature stores to update features with the most recent values for future use. To ingest data and calculate feature values, Dataflow **(9)** is utilized for on-the-fly transformation and feature creation.

In the proposed architecture data aggregation using SQL/Beam capability in Dataflow is recommended. The processed data would be published to a Cloud Pub/Sub topic **(10)**, to facilitate the inference pipeline. In parallel, data ingestion into the online Feature Store is performed by a streaming ingestion call of Vertex AI Feature Store.

# Feature Store

A feature store (11) is one of the most important components of real-time fraud detection. When you calculate the feature values, you need to store those in a feature store:

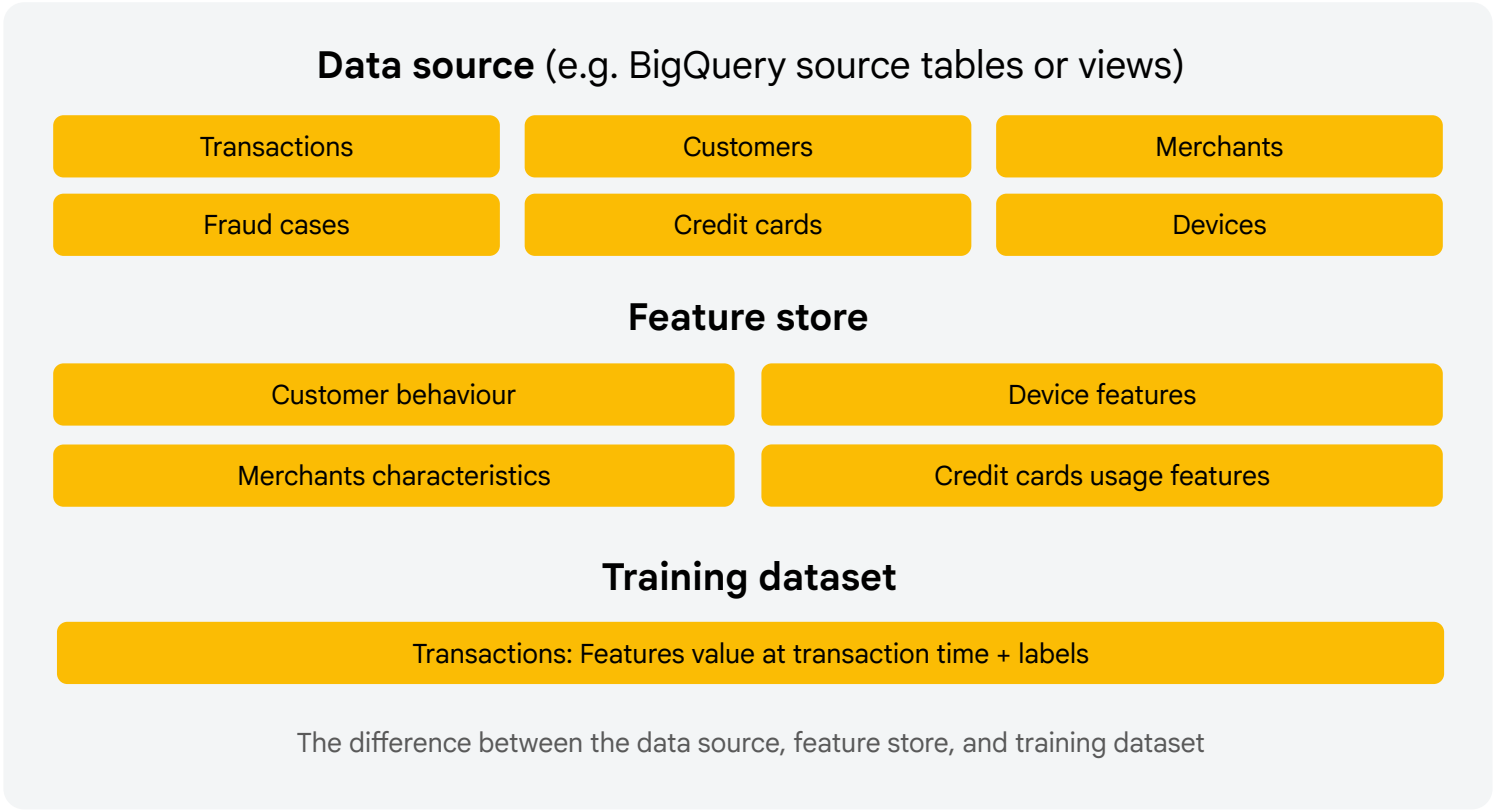
For point-in-time lookups to fetch historical data for training (instead of re-creation of features). In the training part we explain why point-in-time lookup is important for training.

To retrieve feature values in real-time to perform fast online predictions, in the couple of milliseconds.

To ensure that a feature value is ingested once into a feature store and that same value is reused for both training and serving.

The following diagram illustrates the distinction between data sources, features, and training datasets. It demonstrates how elements like "transaction amount" from the “Transactions” table, can be transformed into calculated features like "Avg. Transaction Amount in last 7 days" under a feature view (e.g., "credit card usage features").

**Multiple features then are combined to construct the training dataset used for model development.**




The difference between the data source, feature store, and training dataset

You can maintain feature values, including historical feature data, in the offline store, in BigQuery. Basically these features are used to train ML models. For example, you can register the **Customer behavior** and **Device** features, as two feature groups in Vertex Feature Store, that hold the aggregated features at customer level and device level. The features are built and updated nightly, and are used for training and prediction. The following are some examples of the features in each Feature group, driven from source tables:

## Feature Group: Credit Card

- Update/ingest timestamp (FS timestamp) - updated in batch daily
- Credit Card identification - updated in batch
- Number of transactions in last 1/7/14 days - updated in batch
- Average amount of transactions last 1/7/14 days - updated in batch
- Number of transactions in last 10/7/14 days - updated in batch
- Number of transactions last last 10/60 minutes - updated in real-time
- Average amount of transactions last last 10/60 minutes - updated in real-time
- Chrome/Safari usage in last 1/6/12 month
- Number of logins

## Feature Group: Device

- Update/ingest timestamp (FS timestamp) - updated in batch daily
- Device ID - updated in batch
- Number of transactions by the device in the 1/7/14 days - updated in batch
- Risk score calculated average number of frauds on the device in the 1/7/14 days - updated in batch

## Feature Group: Customer

- Update/ingest timestamp (FS timestamp) - updated in batch monthly
- Customer ID - updated in batch
- Number of times customer changed its address - updated in batch
- New accounts opened in last 3 days - updated in batch
- Applied for new card - updated in batch

## Feature Group: Merchants

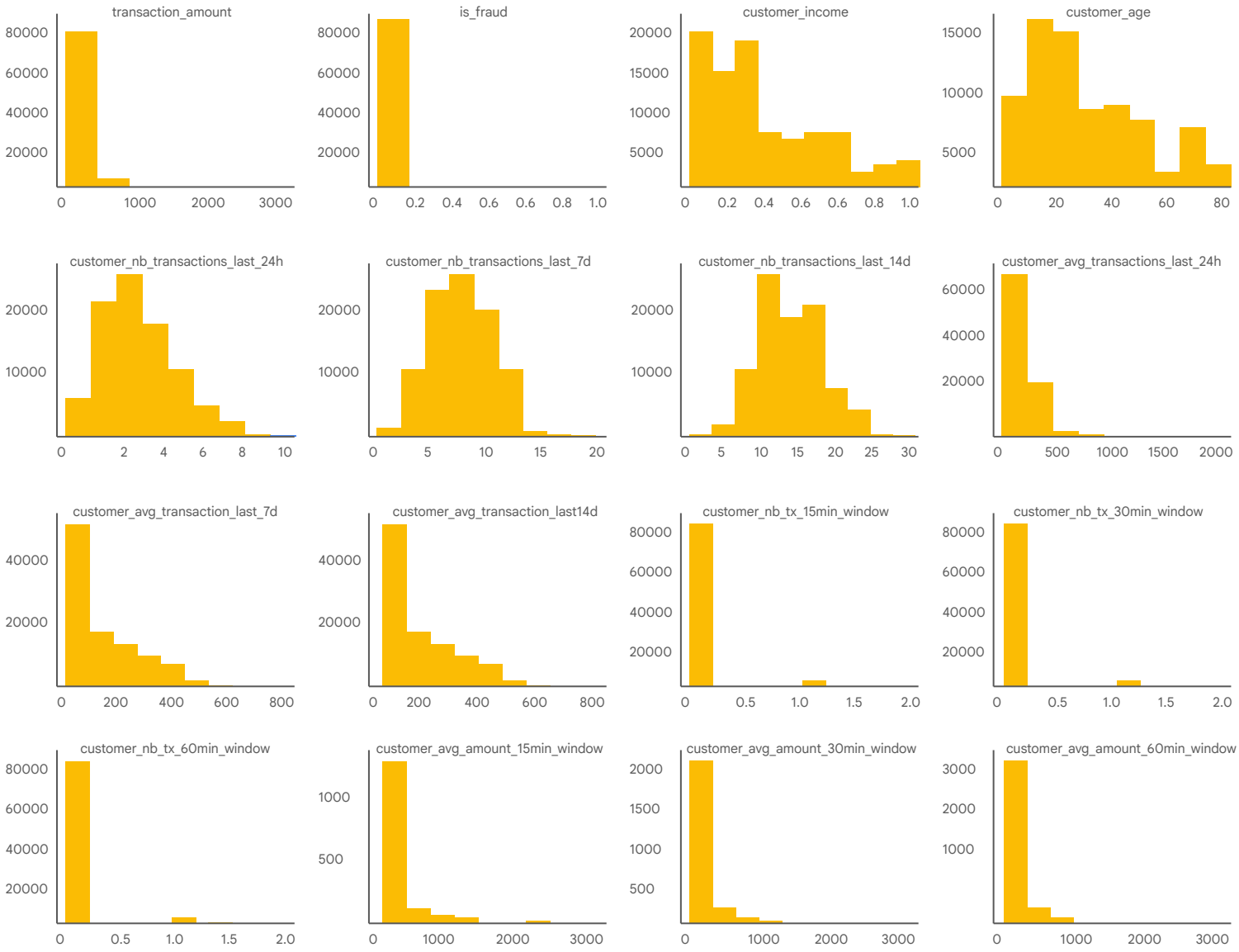
- Update/ingest timestamp (FS timestamp) - updated in batch daily
- Merchant ID - updated in batch
- Number of terminals
- Merchant size
- Industry
- Number of transactions per month
- Number of fraud cases per month

You also need an online feature store, as a high performance real-time serving infrastructure that can perform low-latency serving, for example, under 3 milliseconds.

Vertex AI Feature Store provides resources that let you set up online serving by specifying your feature data sources. Vertex AI Feature Store lets you online serve feature values in real-time from a feature view within an online store.

**Recommendation:** Before commencing the modeling process, it is crucial to examine the extracted feature values. Although a comprehensive exploration of feature validation and checking techniques falls outside the scope of this paper, we provide a demonstrative example below, showcasing the plotted distribution of features.

**Distribution of feature values as an example of feature validation before modeling.**



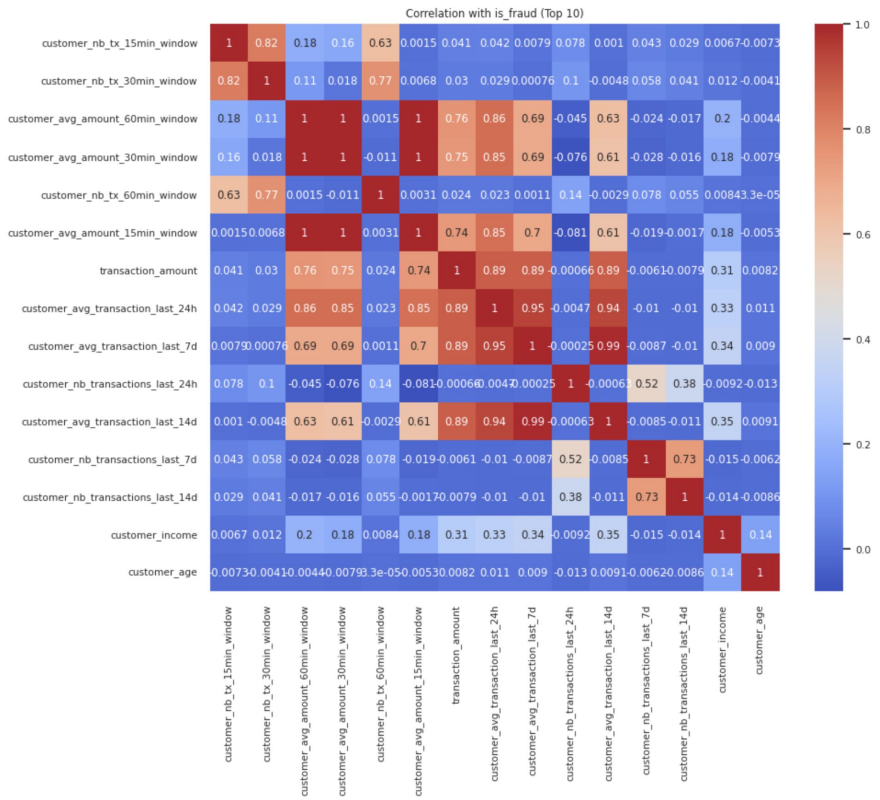
# Experimentation

Typically, the prior step before training at scale is an iterative process of selecting the right features (a.k.a training dataset), experimenting with various algorithms, building machine learning models locally, tuning it with parameters and feature combinations, and evaluating it. You can think of these steps as an experimental lab process for the “Training at scale” stepe. The experimentation phase is crucial when building fraud detection models, as it allows you to explore various options and discover the most effective approach for your specific use case.

In terms of tooling, Vertex AI Workbench (12) can be used to examine different models and evaluate those before formailizing it into a pipeline. There are some typical activities in this step:

## Feature Selection

The primary goal at this stage is to select the most appropriate features for the training dataset. There are various methods to achieve this. A commonly employed approach are statistical measures like correlation or mutual information, algorithm performance based on feature subsets, embedded methods and  incorporating domain knowledge. For example, correlation study can serve multiple purposes, such as validating features, detecting feature leakage, and determining the linearity or non-linearity of the problem.




**Recommendation:** To identify potentially relevant features, draw upon your domain expertise. Iterate through the process of feature selection, ensuring that you have right features. There's no need to experiment with the entire dataset; you can select a representative subsample that accurately reflects the whole.

It's expected that you're familiar with the process of splitting your data into training, validation, and test sets. This is an essential step in machine learning. As a quick reminder, the training set is used to train the model, the validation set is employed to tune the model's hyperparameters, and the test set is utilized to assess the model's performance.

Another important step in this process is to handle imbalanced data. You can use one of Google Cloud managed services (e.g. BigQuery, Dataproc, etc) for this step. The typical techniques to address this issue are:

## Data-Level Techniques

- **Resampling:**
  - **Undersampling:** Randomly remove samples from the majority (non-fraudulent) class to match the minority class.
  - **Oversampling:** Duplicate samples from the minority (fraudulent) class or create synthetic samples using techniques like SMOTE (Synthetic Minority Over-sampling Technique).
  - **Hybrid:** Combine oversampling and undersampling for a more balanced approach.
- **Data Augmentation:** Generate additional synthetic fraud examples by perturbing existing ones, introducing noise, or using generative models.

## Algorithm-Level Techniques

- **Cost-Sensitive Learning:** Assign higher misclassification costs to the fraudulent class during model training. This incentivizes the model to focus on fraud detection.
- **Ensemble Methods:** Combine multiple models (e.g., decision trees, random forests) to improve overall performance. Ensemble methods can often handle imbalanced data more effectively.
- **Anomaly Detection:** Treat fraud detection as an anomaly detection problem, focusing on identifying outliers in the data.

Among the aforementioned techniques, a combination of hybrid resampling and data augmentation is frequently employed. Data augmentation is typically constructed using known fraud patterns or third-party data.

Please note that you should split your dataset into training, validation, and test sets before balancing. Here's why:

## Avoid Data Leakage

Balancing techniques often involve creating synthetic data points or duplicating existing ones. If you balance the entire dataset and then split it, information from the test set might leak into the training set, leading to overly optimistic evaluation results.

## Hyperparameter Tuning

The validation set is used for tuning hyperparameters. By splitting before balancing, you can tune your model on the imbalanced validation set, which more closely resembles the real-world scenario your model will encounter.

## Training and Hyper Parameter Optimization

During the experiment phase, carefully select the algorithm for training. Consider options such as Random Forest, XGboost, Logistic Regression, or Neural Network. The best algorithm for your specific fraud detection problem will depend on your data characteristics, resources, and priorities. As mentioned before, we recommend experimenting with different options and choose the one that delivers the best results for your particular use case. Notably, the XGboost algorithm is often favored for fraud detection as it exhibits superior performance compared to many other algorithms. Additionally, XGboost requires minimal feature engineering and hyperparameter tuning while maintaining a reasonable level of model interpretability.

## Recommendation

Take an iterative approach. Begin with AutoML to establish a performance baseline. This provides a reference point for more complex models. Increase model complexity, experimenting with different algorithms (random forests, XGBoost, neural networks), feature engineering techniques, and hyperparameter tuning. Evaluate each change systematically and keep track of results in the Vertex AI Experiment and Tensorboard.

To select the best algorithm, choose evaluation metrics that align with your objectives. This typically involves prioritizing recall (sensitivity) while maintaining acceptable precision (we explain it further in the evaluation section.). Consider cost-based metrics for monetary values, e.g. false positive or false negative cases. Accordingly, adjust the probability threshold for classifying a transaction as fraud based on your business needs. A higher threshold will increase precision (fewer false positives) but might decrease recall (more false negatives).

Use stratified k-fold cross-validation to ensure that each fold maintains the same class distribution as the original dataset. This is crucial for imbalanced fraud detection datasets.

Hyperparameter tuning (HPT) is crucial for fraud detection because it helps optimize the performance of machine learning models, leading to more accurate identification of fraudulent activities. In Vertex AI, hyperparameter tuning works by systematically exploring different combinations of hyperparameter values (e.g., learning rate, number of trees) to find the set that results in the best model performance on a validation dataset. This process leverages automated algorithms like Bayesian optimization or grid search to efficiently search the hyperparameter space, reducing the need for manual trial and error and ensuring that the model is tailored to the specific fraud detection task, thus improving its ability to detect subtle patterns and anomalies in the data.

Please note that the goal of HPT in the experimentation step is not to tune your model, but to find the right tuning strategy for your chosen algorithm by testing on a subset of data. Keep in mind that you will do this step at scale through an orchestrator pipeline later on all your data after each training.

## Model Evaluation and Registration

Evaluating fraud detection models requires specific considerations due to the inherent class imbalance and the high cost associated with false negatives (missing actual fraud).

### Here are some best practices:

#### Choose Recall (Sensitivity) metric

This metric prioritizes minimizing the occurrence of false negatives, or fraudulent transactions that go undetected by the model. It's crucial as the cost of missing fraud is high. However, the optimal sensitivity level can vary based on the bank's cost of investigating false positive cases. To address this, many banks adopt a Cost-Based Evaluation approach. In this approach, monetary values or weights are assigned to different types of errors (false positives and false negatives) based on their impact on the business. By calculating a cost-based metric that incorporates these weights, banks can comprehensively evaluate a model's overall financial performance and make informed decisions.

#### Use AUC-ROC Curve

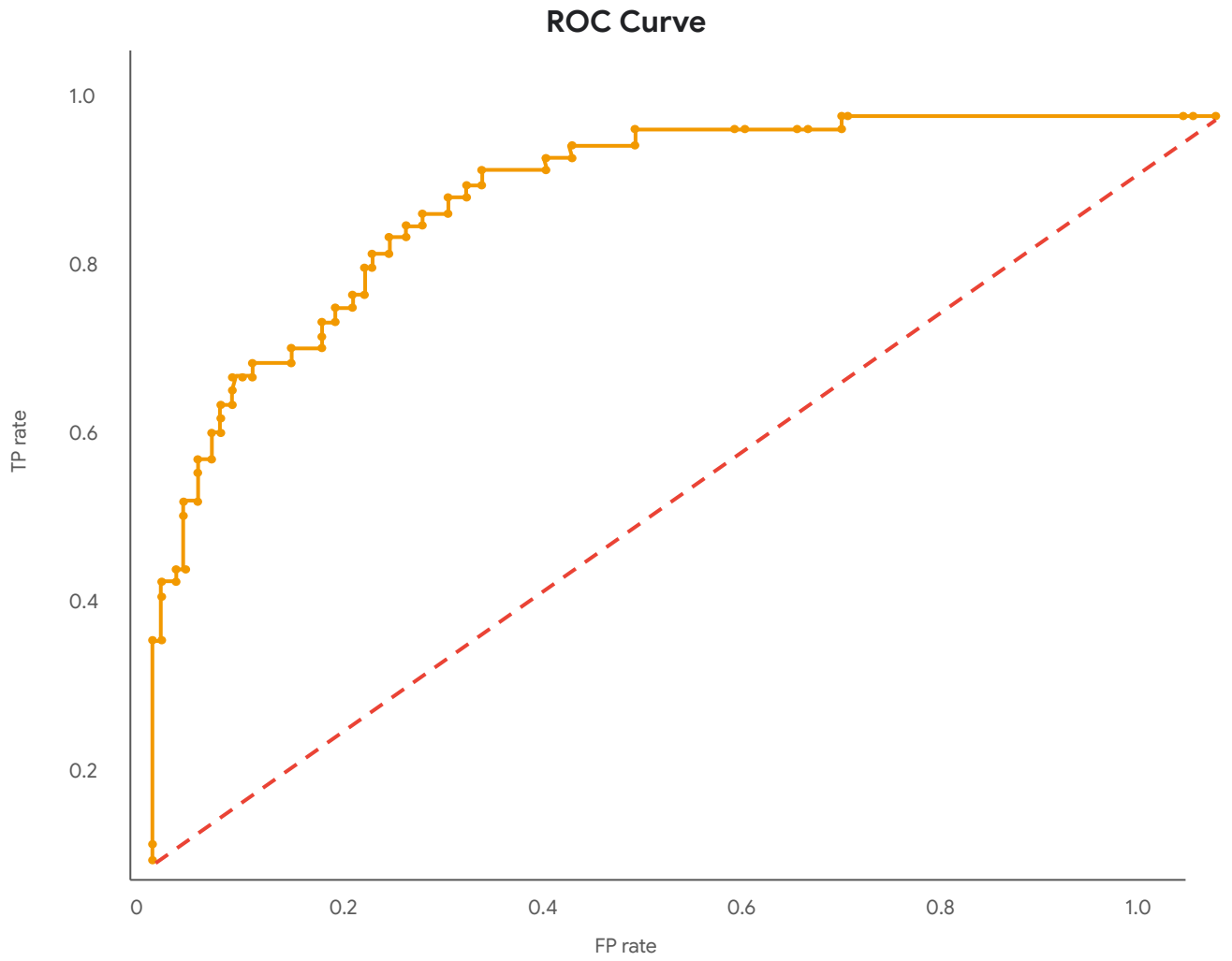
A graphical representation of the model's ability to discriminate between fraudulent and legitimate transactions across different thresholds.

#### Use a Representative Test Set

Ensure that the test set reflects the real-world distribution of fraud and non-fraud cases.

In the fraud detection process, training models usually assign a probability score to each transaction, indicating the likelihood of it being fraudulent or legitimate. A critical parameter that needs to be established at this stage is the threshold that determines when a transaction is flagged as potentially fraudulent. We might get insight into the optimal threshold by examining a Receiver Operator Characteristic (ROC) Curve. This will plot the true positive (TP) vs. false positive (FP) rates at different classification thresholds. Lowering the classification threshold classifies more items as positive/fraud, thus increasing both False Positives and True Positives.

## The following figure shows a sample ROC curve.




For example, in this ROC curve chart, the optimal classification threshold for the trained model with XGBoost, is around 0.36 to get the highest TP rate and minimum FP rate.

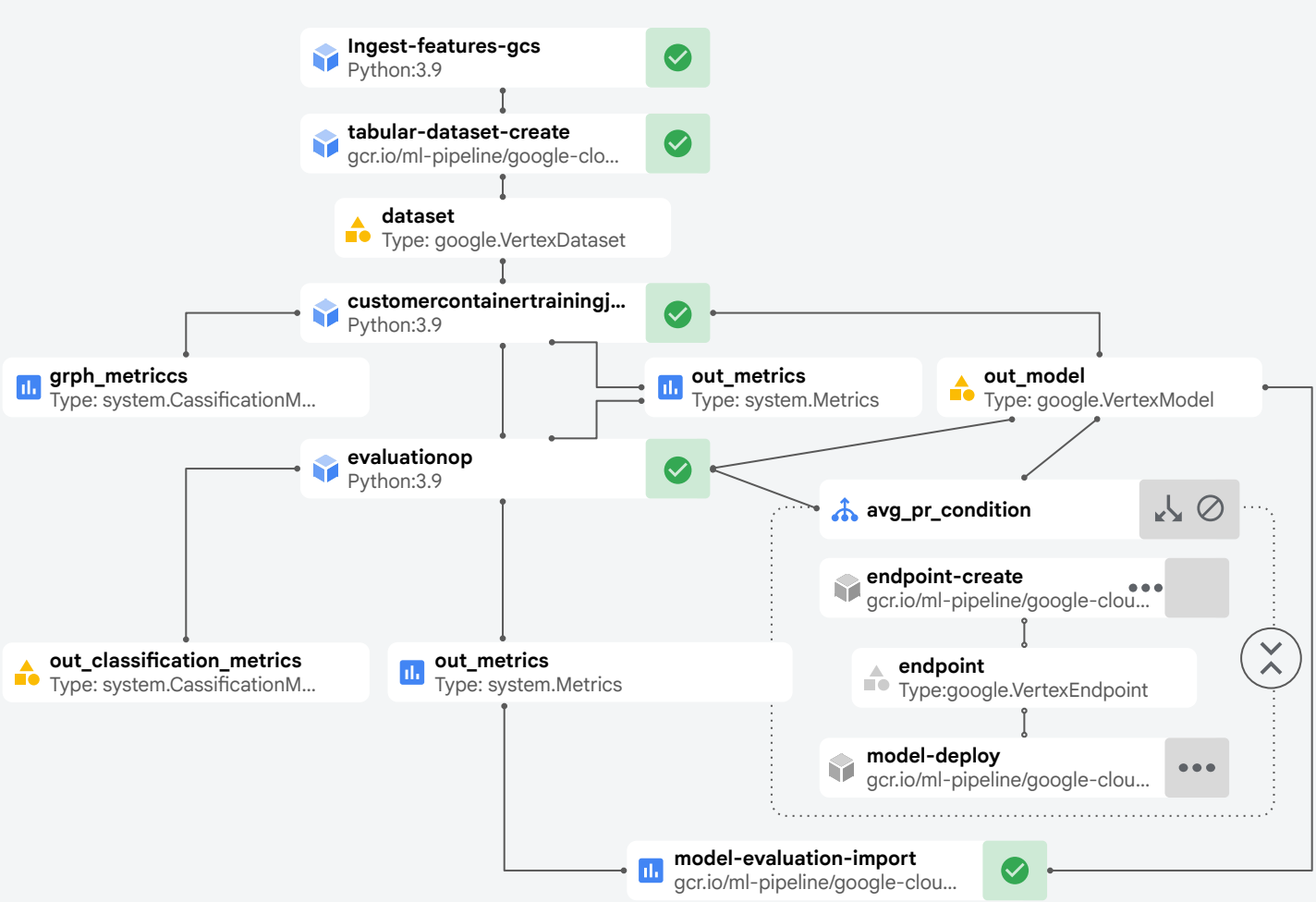
Additionally, we recommend calculating the Area Under the Curve (AUC) metric for each trained model. The AUC metric provides a useful way to compare different models (e.g. models trained with different algorithms, features, or hyperparameters). The AUC metric has a value range of 0 to 1.0. A model making entirely incorrect predictions would have an AUC of 0.0, while a model making entirely correct predictions would have an AUC of 1.0. In the  example above, the AUC for the model is 0.90.

# Formalization / Orchestration

After experimenting with various algorithms, the next step is to organize those steps into a cohesive pipeline. We will delve deeper into the advantages of orchestrating all these steps later, in the MLOps section, but it's worth noting that having a robust orchestrator can accelerate both the experimentation and productionalization phases of the entire process. Vertex AI Pipelines (13) helps you to automate, monitor, and govern your ML step by orchestrating your ML workflow in a serverless manner. Though the pipeline can have many components in the real-world, the minimum steps that you need to add to the pipeline are training and hyperparameter selection, model validation and registering model into a model registry.

**Please notice a pipeline can have as many as the following components:**

| | | | | |
|---|---|---|---|---|
| Data ingestion | Data validation | Feature creation | Feature selection | Training |
| Hyperparameter Tuning | Metadata Tracking | Model Evaluation | Criteria for deployment | Deployment |




The subsequent sections elaborate on the process of formalizing each step and integrating it into the pipeline. Additionally, they offer best practices and tooling recommendations for each component of the pipeline.

# Training dataset

To prepare a training dataset, you need to extract training data from the Feature Store and use it to train your model. This process involves gathering data related to the customer behavior, devices, and relevant features that you have already determined in the experimentation stage.

**Note:** You need to join the features with transactional level data with labels.

The goal is to create a flat table, known as a training dataset, by joining these entities. Each label (fraudulent or not-fraudulent) represents an observation at a specific moment in time. Therefore, it's crucial to retrieve feature values that align with the time when the observation was made. For instance, if a fraudulent transaction occurred on June 21, 2022, for a particular customer, the model should be trained using the average transaction amount for that customer at that precise time, not the current or previous average amounts. Vertex AI Feature Store enables point-in-time lookups, allowing you to fetch feature values at specific moments, ensuring accurate model training.

The training dataset **(14)** can be fetched directly into a BigQuery or Cloud Storage from the Feature Store. We recommend building a training dataset as one of the components of the machine learning pipeline (e.g. Vertex AI Pipeline). We recommend using SQL, and a Python wrapper or BQ components to execute this job.

Optionally a Vertex AI managed dataset can be built using Vertex AI dataset to streamline the training pipeline.

**Note:** For fraud detection, integrating feature preparation steps (data ingestion, validation, transformation, feature engineering, etc) into the machine learning pipeline is generally not recommended, although it may be effective for certain ML applications. Due to its complexity, resource intensiveness, and frequent execution, feature engineering for fraud presents challenges. Tightly coupling feature engineering and model training can hinder reproducibility and maintenance. Moreover, separating these processes enables optimized and parallelized feature generation, potentially enhancing scalability.

**Note:** When building fraud detection models, the number of datasets and models required should be taken into account. Different channels (e.g., POS vs. mobile payment vs. online shopping) and data sources (e.g., countries or currencies) may necessitate specialized models for each. This introduces the need to select appropriate features and construct multiple training pipelines. Consequently, there are additional costs associated with maintaining multiple datasets, models, pipelines, and potentially endpoints during serving.

# Training and HPT

Google Cloud provides several options for training models. You can use BigQuery ML, AutoML, or custom models in Vertex AI. In this architecture, we recommend using Vertex AI training (15). It offers full control over the training process, which is particularly important when migrating a fraud workload to the cloud. Vertex AI training allows you to train multiple models, compare their performance, and store them in the model registry. These models can then be seamlessly deployed to endpoints on Vertex AI.

In the formalization process, you need to package your training code, to be able to run it using Vertex AI training. It involved containerizing your code using docker files. We recommend using one of the pre-built ML containers and injecting your code into the container. For example, you can use the Custom Training Job from Google Cloud Pipeline Components. This component in the Kubeflow Pipelines SDK lets you define a custom training job within your pipeline.

# Model evaluation and registration

To select the optimal features and algorithms, model evaluation should be performed during the experimentation stage. Furthermore, after training the model in the pipeline, a systematic evaluation is required. The primary objective of this evaluation is to make an informed decision regarding model deployment.

Eventually, model evaluation metrics (16) provide quantitative measurements of how your model performed on the test set. You can use your pipeline to emit this metric to Vertex AI metadata and evaluate the model performance. Also, you can use it as a criteria to register the trained model as a new version to be deployed to an endpoint for serving.

The Vertex AI Model Registry (17) is a central repository where you can manage the lifecycle of your ML models.

From the Vertex AI Model Registry, you have an overview of your models so you can better organize, track, and train new versions. When you have a model version you would like to deploy, create a Vertex AI endpoint to host your model. Finally, deploy the model to the endpoint, configuring settings like machine type and traffic splitting as needed.

# Model deployment and monitoring

You must deploy a model to an endpoint before that model can be used to serve online predictions. Deploying a model associates physical resources with the model so it can serve online predictions with low latency. An undeployed model can serve batch predictions, which do not have the same low latency requirements.

**Note:** You can deploy more than one model to an endpoint, which is very useful for A/B testing and blue/Green deployment for fraud models.

We recommend using Vertex AI private endpoints (18) to serve online predictions for transactions as it provides a low-latency, secure connection to the Vertex AI online prediction service.

**Note:** Model deployment would happen through the MLOps process and usually with manual approval.

When you deploy the fraud detection model in production, it performs best on prediction input data that is similar to the training data. When the input data deviates from the data used to train the model, the model's performance can deteriorate, even if the model itself hasn't changed.

To maintain a model's performance, Vertex AI Model Monitoring (19) monitors the model's prediction input data for feature skew and drift:

Training-serving skew occurs when the feature data distribution in production deviates from the feature data distribution used to train the model. If the original training data is available, you can enable skew detection to monitor your models for training-serving skew.

Prediction drift occurs when feature data distribution in production changes significantly over time. If the original training data isn't available, you can enable drift detection to monitor the input data for changes over time.

You can enable both skew and drift detection, when you deploy the model into the endpoint.

You can monitor and debug your Model Monitoring job through alerts. Model Monitoring automatically notifies you of job updates through email, but you can also set up alerts through Cloud Logging. Upon receiving a notification, you should take action, for example, automatically trigger the pipeline for re-training the model, or investigate the cause of the issue or change of payloads data.

## Real time inference

In the real-time inference step **(20)**, incoming transactions trigger the retrieval of pre-computed offline features from the Feature Store. These are combined with on-the-fly processed features **(9 and 10)** and sent as a payload to the Vertex AI endpoint for fraud detection.

We suggest using Cloud Run for this feature combination task. As a managed compute, Cloud Run allows you to execute containerized processes triggered by requests or events. Its serverless nature eliminates infrastructure management overhead, letting you concentrate on constructing the fraud detection payload. In this scenario, Cloud Run's primary function is to efficiently retrieve and merge pre-calculated features from online feature store with real-time features before forwarding them to the Vertex AI Endpoint.

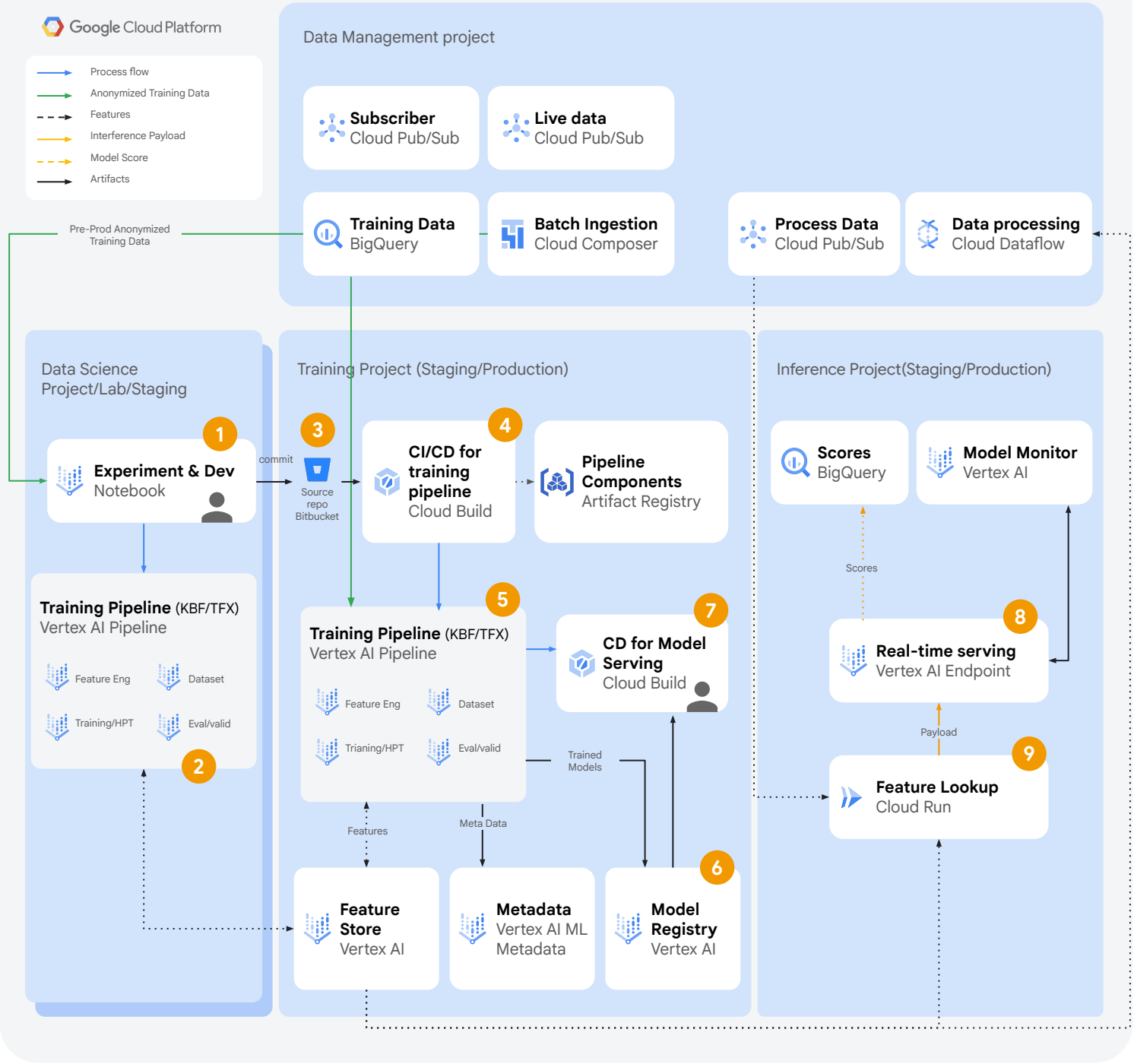
# MLOps

Fraud detection pipelines typically need to be retrained to capture new patterns of fraud. You need to monitor the models in production, and upon observing drift or proactively on a schedule, retrain the model. To streamline this process you need a mature MLOps. The aim of showing MLOps here is to help data scientists, ML engineers and DevOps teams to achieve continuous training and continuous delivery of pipeline implementations. Some of the motivations for reaching this level of sophistication comes from the need to frequently update Fraud Detection pipelines with new implementations and new data.

In such a scenario, an automated robust Continuous Integration (CI) and Continuous Deployment (CD) system should be designed, to ensure rapid and reliable update of the pipelines in production. The objective is to allow data scientists to rapidly explore new ideas around feature engineering, model architecture, hyper parameters, etc. to implement them, and automatically build, test and deploy the new implementation of the pipeline components to the target environment.

For further guidance on aligning MLOps practices with your organization's specific tools and maturity level, we recommend consulting the resource "MLOps: Continuous delivery and automation pipelines in machine learning" or "Build and deploy generative AI and machine learning models in an enterprise"

**The following diagram illustrates the sample flow and steps involved in deploying a simple fraud detection pipeline through an MLOps process:**



The following outlines the steps involved in a typical MLOps diagram. For each step, we'll explore the tasks involved, typical inputs and outputs, and offer recommendations on tooling.

## 01. ML Development

- Use Vertex AI Workbench for model development and experiments.
- Track experiments with Vertex AI ML metadata (data, hyperparameters, evaluation metrics).

## 02. Formalize ML Training Procedure for Retraining

- Package ML code and use Vertex AI training.
- Utilize Kubeflow pipeline templates for ML code development/migration.
- Build/Run/Test training and batch inference pipelines locally (Vertex AI Pipeline).
- Commit the source code of ML pipeline to a source control repository.

## 03. Merge Source Code

- Merge source code for training pipeline operationalization.
- Source code includes:
  - Python scripts (model building, evaluation, etc.)
  - Build specs for components, pipeline compilation, and execution.
  - Configuration file (parameters).

## 04. Deploy Training Pipeline Using Standard CI/CD Processes and Tools

- CI/CD triggered upon code commit.
- The Training project is used for CI/CD (first in staging and then production).
- Environment operates autonomously without human intervention.
- Continuous Integration (CI) Stage:
  - The source code includes training pipeline component codes (packages, images, etc).
  - Artifacts are stored in Artifact Registry (pipeline components, containers).
  - Pipeline is compiled and added as a template to Artifact Registry.
  - The output of this step are components and pipeline definition for deployment.
- Continuous Deployment (CD) Stage:
  - Tested training pipeline artifacts are deployed to the target environment for end-to-end testing.
  - Pipelines are tested in non-production environments on a subset of production data.
  - New pipeline is tested, fallback to current model if new one fails.
  - The output of this stage is a deployed pipeline.
- Successful CI/CD pipeline execution results in deployment of new ML Continuous Training pipeline.

## 05. Training Pipeline (Vertex AI Pipeline)

- Provides Continuous Training capability.
- The pipeline utilizes non-production data in staging project, and production data in production project.
- Data validation and model validation are critical in this step.
- Continuous training pipeline triggers:
  - On-demand/On-schedule.
  - On availability of new training data or code.
  - On model performance degradation (feedback loop).
- Successful execution registers a model candidate in Vertex AI Model Registry.
- Training metadata and artifacts are stored in Vertex AI ML metadata

## 06. Model Registration

Vertex AI Model registry is used to register each version of the model.

## 07. Model Deployment

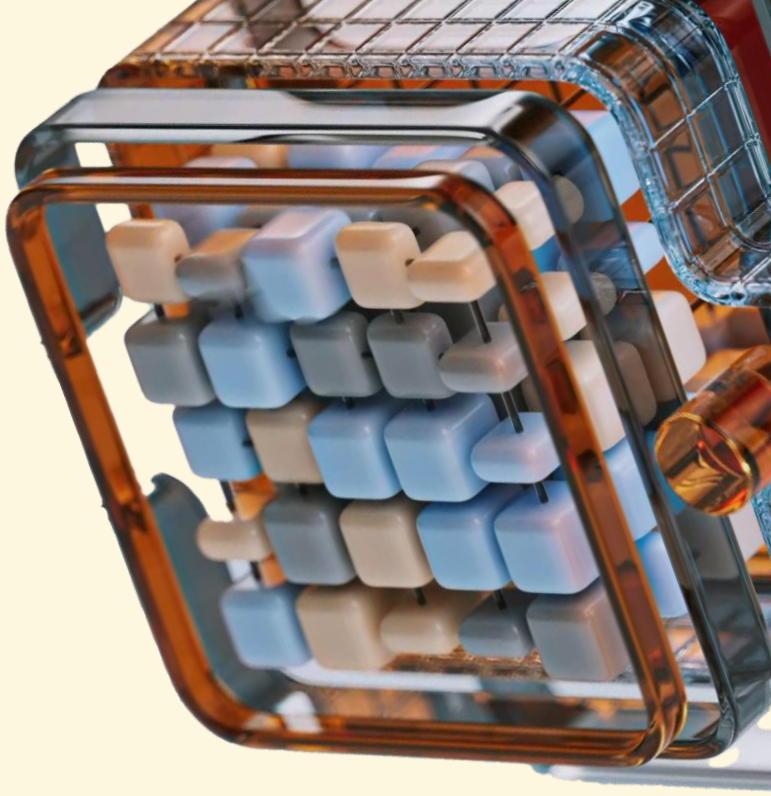
- The model is packaged, tested, and deployed to the target serving environment (staging and the production project).
- Continuous Deployment (CD):
  - Fetches model from Vertex Model Registry and the serving container from Artifact Registry.
  - Runs tests on subset of data (e.g., through batch).
  - Releases model into production upon manual approval:
    - Deploys batch pipeline for scheduled execution with production data.
    - Deploys model as an endpoint for real-time inference.
- Logs inference metrics and prediction requests for continuous monitoring.

## 08. Model Serving (Endpoint)

- Online inference in near real-time for high-frequency requests (REST or gRPC).
- Regular monitoring and updating of models with fresh data is crucial.

## 09. Prediction

- Cloud Run is used for feature lookup from Feature Store.
- Cloud Run deployment can be done with a Cloud Build.



# Looking Forward

Fraud detection in banking is an ongoing process that requires constant adaptation and innovation. As technology advances, new tools and methodologies are continually being developed to address emerging threats. By staying ahead of the curve and utilizing a multi-layered approach, banks can create a safer financial environment for everyone.

By understanding the methods and approaches used in fraud detection, you can gain valuable insights into how your financial information is protected. Additionally, it's crucial to be vigilant and report any suspicious activity you encounter to your bank immediately. By working together, we can create a more secure financial ecosystem and minimize the impact of fraudulent activities.

Contact us to learn more

The authors gratefully acknowledge the contributions of **Ivan Nardini, Erwin Huizenga, Lavi Nigam, Thu Ya Kyaw, Alok Pattani, Marc Cohen, and Neama Dadkhahnikoo** for their code contributions, reviews, and valuable feedback.

Last update: Oct 25, 2024