**GityHub Link: https://github.com/kazimahmed4u/team-5-project.git**

# Project Title: Guarding transaction with AI_Powered Credit card fraud detection and prevention

**Phase-3**

## 1. PROBLEM STATEMENT

**Dynamically adapt to new and evolving fraud techniques**, ensuring continuous protection against emerging threats that traditional systems fail to recognize.

- **Provide a holistic fraud management solution** encompassing real-time transaction analysis, risk scoring, immediate alerts for suspicious activities, and proactive prevention mechanisms.
- **Minimize disruptions to legitimate cardholders** by drastically reducing false positives and ensuring a seamless and trustworthy transaction experience.
- **Effectively address the inherent class imbalance in transaction data, where fraudulent transactions are a small minority compared to legitimate ones, without compromising detection of the minority class.**

- **Apache Spark:** A powerful and widely used distributed computing framework for large-scale data processing, feature engineering, and machine learning.
-
- **Apache Hadoop (HDFS, MapReduce, YARN):** While Spark is often preferred for its speed, Hadoop remains relevant for distributed storage and batch processing of massive datasets.

- **Apache Flink: A stream processing framework ideal for real-time feature engineering and analysis of streaming transaction data.**

## 2. PROJECT OBJECTIVES

**i. Develop a highly accurate AI-powered fraud detection model:**

- Achieve a significant improvement in fraud detection accuracy compared to traditional rule- based systems, aiming for a substantial reduction in both false positive and false negative rates.

- (e.g., precision, recall, F1-score, AUC)

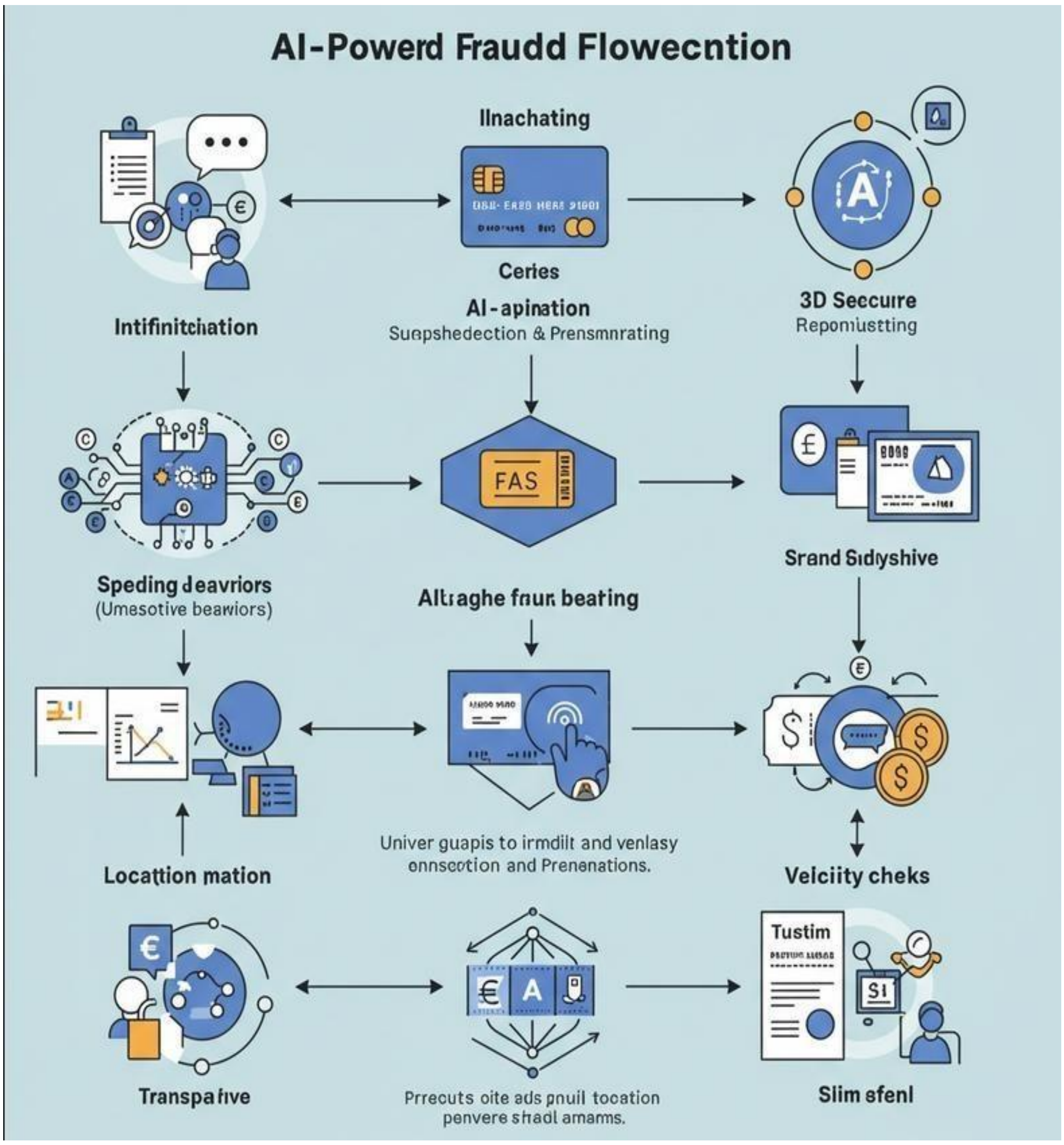
**ii. Build an adaptive system capable of learning and evolving:**

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

Iii. **Achieve scalability and high performance:**

- Design the system architecture to handle the increasing volume of credit card transactions efficiently and with low latency.
- **Ensure the system can scale to accommodate future growth in transaction volume.**

## 3. FLOWCHART OF THE WORKFLOW



- **Apache Hadoop (HDFS, MapReduce, YARN):** While Spark is often preferred for its speed, Hadoop remains relevant for distributed storage and batch processing of massive datasets.
- **Apache Flink:** A stream processing framework ideal for real-time feature engineering and analysis of streaming transaction data.

## 4. DATA DESCRIPTION

**Transaction Data:** This is the most crucial data source and includes details of every credit card transaction, such as:

- **Amount: The monetary value of the transaction. For example, a sudden increase in transaction amount compared to the cardholder's usual spending habits can be a red flag.**

rns or cancellations.

2. **Temporal and Geographic Trends:**

- **Timestamp:** The exact time and date of the transaction. Multiple transactions occurring within a very short time frame but from geographically distant locations can be suspicious.
- **Merchant Information:** Details about the seller or service provider. Transactions with high- risk or blacklisted merchants can be flagged.

### II. Data Processing and Feature Engineering:

- **Big Data Processing Frameworks:**
  - **Apache Spark:** A powerful and widely used distributed computing framework for large-scale data processing, feature engineering, and machine learning.
  - **Apache Hadoop (HDFS, MapReduce, YARN):** While Spark is often preferred for its speed, Hadoop remains relevant for distributed storage and batch processing of massive datasets.
  - **Apache Flink:** A stream processing framework ideal for real-time feature engineering and analysis of streaming transaction data.

The dataset used in this project comprises historical transaction records, including both legitimate and fraudulent instances. Key features include:

- **Transaction ID**: Unique identifier for each transaction
- **Timestamp**: Date and time of the transaction
- **Customer ID**: Anonymized customer identifier

- **Credit Details**: Information such as transaction amount, credit score, credit limit, and outstanding balance
- **Product Information**: Type of product purchased, product ID, category, price, and merchant details
- **Location Data**: Transaction origin, such as city, region, or IP address
- **Device and Channel Info: Device used, browser type, platform (e.g., mobile, web), and payment method**

  **Fraud Label**: Binary indicator (1 = fraudulent, 0 = legitimate)

This data enables the training and testing of AI models to accurately detect anomalies and guard against fraud by learning patterns associated with high-risk transactions across both credit-related and product-level behaviors.

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

## 5. DATA PREPROCESSING

### 1. Data Collection and Ingestion:

- **Real-time Streaming:** Transaction data is captured in real-time as it occurs through various channels (e.g., online purchases, in-store transactions, ATM withdrawals). This often involves streaming data platforms like Apache Kafka or Amazon Kinesis for efficient and low-latency data transfer

### 2. Data Preprocessing:

- **Data Cleaning:** Identifying and handling missing values (imputation using mean, median, or more advanced techniques), correcting errors, and removing irrelevant or duplicate data.

### 3. Data Analysis and Modeling:

- **Exploratory Data Analysis (EDA):** Analyzing the preprocessed data to understand its characteristics, identify patterns, and gain insights into potential fraud indicators. This involves visualizations (histograms, scatter plots), statistical summaries, and correlation analysis.

## 6. EXPLORATORY DATA ANALYSIS(EDA)

Exploratory Data Analysis (EDA) is the foundational step in building effective AI-powered credit card fraud detection systems.

**High-Level Overview:** Examine the dataset's structure:

- Number of rows (transactions) and columns (features).
- Data types of each column (e.g., integer, float, object/string, datetime). This helps identify potential data type issues that need to be addressed.
- Initial rows of the data to get a feel for the content and format.

### Geospatial Analysis (If Location Data is Available):

- **Mapping Transactions:** Visualize transaction locations on a map. Unusual transaction locations for a cardholder can be a strong indicator of fraud.
- **Identify Potential Fraud Indicators:** Discover features or combinations of features that seem to differ significantly between legitimate and fraudulent transactions.
- **Formulate Hypotheses:** Develop initial ideas about which features might be most predictive of fraud.

To understand the underlying patterns and anomalies within the dataset, we conducted an in-depth exploratory data analysis focused on transaction behavior, credit attributes, and product-related insights:

**Missing Values and Data Types:**

- Checked for null or inconsistent entries in critical fields like credit_score, product_category, and transaction_amount.

- Ensured appropriate data types for numerical (e.g., amount, credit limit) and categorical (e.g., product type, device) features.

3. **Fraud Distribution:**

   - Examined class imbalance in the `fraud_label` column.
   - Found that fraudulent transactions account for a small fraction of the data, requiring strategies like oversampling or SMOTE for model training.

4. **Transaction Amount Patterns:**

   - Visualized transaction amounts by fraud status.
   - **Mapping Transactions:** Visualize transaction locations on a map. Unusual transaction locations for a cardholder can be a strong indicator of fraud.
   - **Identify Potential Fraud Indicators:** Discover features or combinations of features that seem to differ significantly between legitimate and fraudulent transactions.
   - **Formulate Hypotheses:** Develop initial ideas about which features might be most predictive of fraud.
   - Detected higher variability and unusual spikes in amounts associated with fraudulent cases.

   - Examined class imbalance in the `fraud_label` column.
   - Found that fraudulent transactions account for a small fraction of the data, requiring strategies like oversampling or SMOTE for model training.

5. **Credit Behavior Insights:**

   - Analyzed the relationship between credit score, credit limit, and fraud probability.
   - Observed that lower credit scores and near-limit usage often correlated with fraudulent transactions.

6. **Product and Category Trends:**

   - Identified specific product categories (e.g., electronics or high-value items) with a higher fraud ratio.
   - Explored merchant behavior and frequency of returns or cancellations.

7. **Temporal and Geographic Trends:**

   Time-series plots revealed fraud peaks during non-business hours.

   - Geographic heatmaps indicated clusters of fraudulent activity in specific regions or IP zones.

8. **Device and Access Channel Analysis:**

    - Compared fraud rates across devices (mobile vs. desktop) and browsers.
    - Noted that unfamiliar or spoofed devices contributed disproportionately to fraud.

9. **Correlation Matrix:**

    - Computed pairwise correlations to identify significant relationships between features like credit_utilization and transaction_risk_score.

    - **Identify Potential Fraud Indicators:** Discover features or combinations of features that seem to differ significantly between legitimate and fraudulent transactions.
    - **Formulate Hypotheses:** Develop initial ideas about which features might be most predictive of fraud.
    - Detected higher variability and unusual spikes in amounts associated with fraudulent cases.

    - Examined class imbalance in the fraud_label column.
    - Found that fraudulent transactions account for a small fraction of the data, requiring strategies like oversampling or SMOTE for model training.

## 7. FEATURE ENGINEERING

**Transaction-Based Features: These features are derived directly from the attributes of individual transactions.**

- **Basic Transaction Attributes:**
    - **Transaction Amount:** The raw amount of the transaction. Often, fraudulent transactions have different amount distributions (e.g., tend to be higher or lower than typical)
    - **Cardholder-Based FeatureGeographic distribution of fraud.**
    - **s:** these features aggregate information about the cardholder's past behavior.

**Historical Spending Patterns:**

    - **Average Transaction Amount (over different time windows):** Captures the typical spending habit.
    - **Standard Deviation of Transaction Amounts:** Measures the variability in spending.
    - **Most Frequent Transaction Amounts/Merchants:** Establishes the "normal" behavior.
- **Recency of Activity:**
    - **Time Since Last Login to Online Banking:** Unusual inactivity followed by transactions could be suspicious.
    - **Time Since Card Activation:** Newly activated cards might be more vulnerable.

## 8. MODEL BUILDING

**Model Selection**

Supervised Learning Models

Logistic Regression

Random

Forests

Decision

Trees

### I. Data Preparation for Model Training

Training

set

Validation

set

Testing set

## II. Data Processing and Feature Engineering:

- **Big Data Processing Frameworks:**
    - **Apache Spark:** A powerful and widely used distributed computing framework for large-scale data processing, feature engineering, and machine learning.
    - **Apache Hadoop (HDFS, MapReduce, YARN):** While Spark is often preferred for its speed, Hadoop remains relevant for distributed storage and batch processing of massive datasets.
    - **Apache Flink:** A stream processing framework ideal for real-time feature engineering and analysis of streaming transaction data.
    - **Average Transaction Amount (over different time windows):** Captures the typical spending habit.
    - **Standard Deviation of Transaction Amounts:** Measures the variability in spending.
    - **Most Frequent Transaction Amounts/Merchants:** Establishes the "normal" behavior.
    - **Recency of Activity:**
    - **Time Since Last Login to Online Banking:** Unusual inactivity followed by transactions could be suspicious.
    - **Time Since Card Activation:** Newly activated cards might be more vulnerable.

## III. Model Evaluation

Confusion Matrix

Precision

**Recall (Sensitivity)**

**Average Precision**

**(AP)**

## 9. ISUALIZATION OF RESULTS & MODEL INSIGHTS

### I. Visualizing Model Performance:

Heatmaps: Display the counts or percentages of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) in a visually intuitive way. Different color intensities can represent the magnitude of each category. This helps quickly assess the types of errors the model is making.

**II. Visualizing Model Insights (Understanding What the Model Learned):**

Bar Charts or Dot Plots: Show the relative importance of different features as determined by the model (e.g., using techniques like Gini importance for tree-based models, coefficient magnitudes for linear models, or permutation importance). This helps understand which factors the AI considers most influential in predicting fraud

.

**III. Interactive Visualizations and Dashboards:**

**Geographic distribution of fraud.**

- False positive rate.
- Number of blocked fraudulent transactions.
- Trends in fraud patterns over time.

This data enables the training and testing of AI models to accurately detect anomalies and guard against fraud by learning patterns associated with high-risk transactions across both credit-related and product-level behaviors.

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

## 10. TOOLS AND TECHNOLOGIES USED

### I. Data Storage and Management:

**Databases:**

- **Relational Databases (SQL):** PostgreSQL, MySQL, Oracle, Microsoft SQL Server are used for structured transaction data, cardholder information, and historical records.

3. This data enables the training and testing of AI models to accurately detect anomalies and guard against fraud by learning patterns associated with high-risk transactions across both credit-related and product-level behaviors.

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

**NoSQL Databases:** Cassandra, MongoDB, HBase are employed for handling large volumes of unstructured or semi-structured data, real-time data streams, and scalability.

- **Data Warehouses:** Amazon Redshift, Google BigQuery, Snowflake are used for storing and analyzing large historical datasets for model training and batch processing.

Bar Charts or Dot Plots: Show the relative importance of different features as determined by the model (e.g., using techniques like Gini importance for tree-based models, coefficient magnitudes for linear models, or permutation importance). This helps understand which factors the AI considers most influential in predicting fraud

### II. Data Processing and Feature Engineering:

- **Big Data Processing Frameworks:**

    - **Apache Spark:** A powerful and widely used distributed computing framework for large-scale data processing, feature engineering, and machine learning.

    - **Apache Hadoop (HDFS, MapReduce, YARN):** While Spark is often preferred for its speed, Hadoop remains relevant for distributed storage and batch processing of massive datasets.

      **Apache Flink:** A stream processing framework ideal for real-time feature engineering and analysis of streaming transaction data.

Bar Charts or Dot Plots: Show the relative importance of different features as determined by the model (e.g., using techniques like Gini importance for tree-based models, coefficient magnitudes for linear models, or permutation importance). This helps understand which factors the AI considers most influential in predicting fraud

1. Data Cleaning and Preprocessing

- **Handling Missing Values:**
  Imputed missing credit_score and product_category using median/mode imputation or predictive modeling.

- **Data Type Conversion:**
  Converted timestamps to datetime objects and encoded categorical features like device_type, product_category, and payment_method.

Bar Charts or Dot Plots: Show the relative importance of different features as determined by the model (e.g., using techniques like Gini importance for tree-based models, coefficient magnitudes for linear models, or permutation importance). This helps understand which factors the AI considers most influential in predicting fraud

    - **Relational Databases (SQL):** PostgreSQL, MySQL, Oracle, Microsoft SQL Server are used for structured transaction data, cardholder information, and historical records.

4. This data enables the training and testing of AI models to accurately detect anomalies and guard against fraud by learning patterns associated with high-risk transactions across both credit-related and product-level behaviors.

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

  **NoSQL Databases:** Cassandra, MongoDB, HBase are employed for handling large volumes of unstructured or semi-structured data, real-time data streams, and scalability.

- **Data Warehouses:** Amazon Redshift, Google BigQuery, Snowflake are used for storing and analyzing large historical datasets for model training and batch processing.

**Outlier Detection:**
Applied IQR and z-score methods to detect and optionally cap extreme values in **transaction_amount, credit_utilization, etc.**

2. Feature Engineering

Credit-Based Features

- **Credit Utilization Ratio:**
  ‘creditutilization‘=‘transactionamount‘/‘creditlim
  it‘ High utilization often signals risky behavior.

- **Credit Score Binning:**
  Grouped credit_score into ranges (e.g., poor, average, good) to simplify analysis.

Product-Based Features

- **High-Risk Product Flag:**

  Created a binary feature to flag purchases in categories frequently linked to fraud (e.g., electronics, luxury items).

- **Product Price Deviation:**
  Compared current transaction price to a rolling mean of previous purchases by the same user to detect anomalies.

Transaction Behavior Features

- **Transaction Frequency:**
  Number of transactions by a user within a certain time window (e.g., last hour, day).

- **Time-Based Features:**
  Extracted hour, day, and weekday from timestamp to capture off-hour or holiday activity.

  Geo-Distance:

  Calculated distance between current and previous transaction locations for the same user to flag unusual access patterns.

Device and Channel Features

- **New Device or Location Indicator:**
  Boolean flags for whether the transaction came from a new device or location.

- **Channel Risk Score:**
  **Assigned weights to transaction channels (e.g., web, mobile) based on historical fraud likelihood.**

  - **Relational Databases (SQL):** PostgreSQL, MySQL, Oracle, Microsoft SQL Server are used for structured transaction data, cardholder information, and historical records.

    This data enables the training and testing of AI models to accurately detect anomalies and guard against fraud by learning patterns associated with high-risk transactions across both credit-related and product-level behaviors.

- Implement mechanisms for continuous learning and model retraining using new transaction data to ensure the system adapts to emerging fraud patterns and techniques.
- Incorporate feedback loops to refine the AI models based on the outcomes of transaction investigations and fraud confirmations.

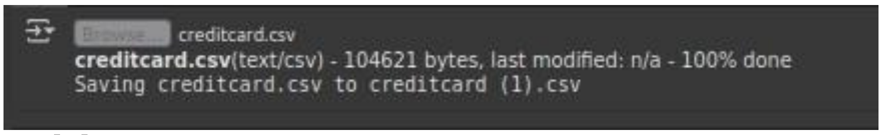
  **NoSQL Databases:** Cassandra, MongoDB, HBase are employed for handling large volumes of unstructured or semi-structured data, real-time data streams, and scalability.

- **Data Warehouses:** Amazon Redshift, Google BigQuery, Snowflake are used for storing and analyzing large historical datasets for model training and batch processing.

**Bar Charts or Dot Plots: Show the relative importance of different features as determined by the model (e.g., using techniques like Gini importance for tree-based models, coefficient magnitudes for linear models, or permutation importance). This helps understand which factors the AI considers most influential in predicting fraud**

**Upload the Dataset**

```
from google.colab import files
uploaded = files.upload()
```



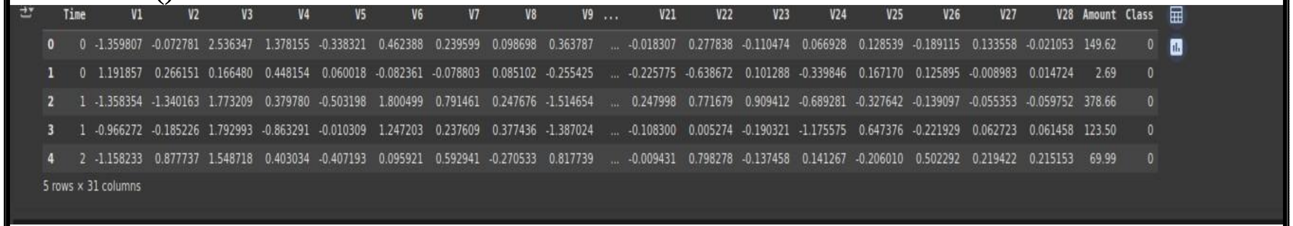
```
Browse... creditcard.csv
creditcard.csv(text/csv) - 104621 bytes, last modified: n/a - 100% done
Saving creditcard.csv to creditcard (1).csv
```

**Load the Dataset**

```
import pandas as pd
df = pd.read_csv('creditcard.csv')
```

**Data Exploration**

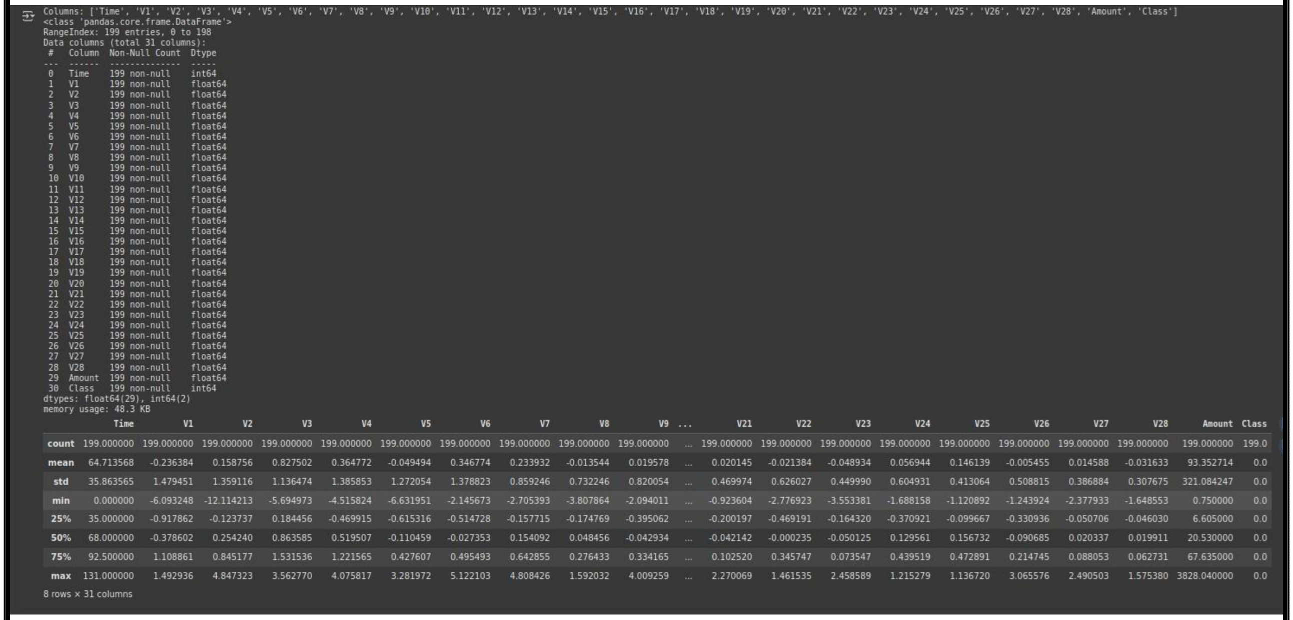
```
df.head()
```



| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| **1** | 0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| **2** | 1 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| **3** | 1 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| **4** | 2 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |

5 rows × 31 columns

**print("Shape:", df.shape)**
**print("Columns:", df.columns.tolist())**
**df.info()**



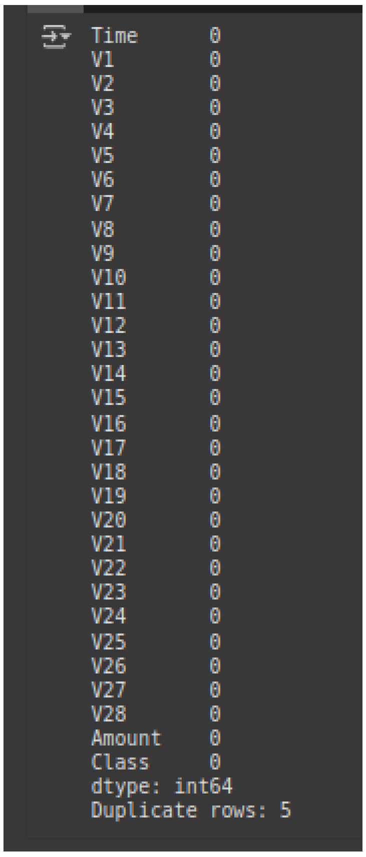
```
Columns: ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199 entries, 0 to 198
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    199 non-null    int64
 1   V1      199 non-null    float64
 2   V2      199 non-null    float64
 3   V3      199 non-null    float64
 4   V4      199 non-null    float64
 5   V5      199 non-null    float64
 6   V6      199 non-null    float64
 7   V7      199 non-null    float64
 8   V8      199 non-null    float64
 9   V9      199 non-null    float64
 10  V10     199 non-null    float64
 11  V11     199 non-null    float64
 12  V12     199 non-null    float64
 13  V13     199 non-null    float64
 14  V14     199 non-null    float64
 15  V15     199 non-null    float64
 16  V16     199 non-null    float64
 17  V17     199 non-null    float64
 18  V18     199 non-null    float64
 19  V19     199 non-null    float64
 20  V20     199 non-null    float64
 21  V21     199 non-null    float64
 22  V22     199 non-null    float64
 23  V23     199 non-null    float64
 24  V24     199 non-null    float64
 25  V25     199 non-null    float64
 26  V26     199 non-null    float64
 27  V27     199 non-null    float64
 28  V28     199 non-null    float64
 29  Amount  199 non-null    float64
 30  Class   199 non-null    int64
dtypes: float64(29), int64(2)
memory usage: 48.3 KB
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | ... | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.000000 | 199.0 |
| **mean** | 64.713568 | -0.236384 | 0.158756 | 0.827502 | 0.364772 | -0.049494 | 0.346774 | 0.233932 | -0.013544 | 0.019578 | ... | 0.020145 | -0.021384 | -0.048934 | 0.056944 | 0.146139 | -0.005455 | 0.014588 | -0.031633 | 93.352714 | 0.0 |
| **std** | 35.863565 | 1.479451 | 1.359116 | 1.136474 | 1.385853 | 1.272054 | 1.378823 | 0.859246 | 0.732246 | 0.820054 | ... | 0.469974 | 0.626027 | 0.449990 | 0.604931 | 0.413064 | 0.508815 | 0.386884 | 0.307675 | 321.084247 | 0.0 |
| **min** | 0.000000 | -6.093248 | -12.114213 | -5.694973 | -4.515824 | -6.631951 | -2.145673 | -2.705393 | -3.807864 | -2.094011 | ... | -0.923604 | -2.776923 | -3.553381 | -1.688158 | -1.120892 | -1.243924 | -2.377933 | -1.648553 | 0.750000 | 0.0 |
| **25%** | 35.000000 | -0.917862 | -0.123737 | 0.184456 | -0.469915 | -0.615316 | -0.514728 | -0.157715 | -0.174769 | -0.395062 | ... | -0.200197 | -0.469191 | -0.164320 | -0.370921 | -0.099667 | -0.330936 | -0.050706 | -0.046030 | 6.605000 | 0.0 |
| **50%** | 68.000000 | -0.378602 | 0.254240 | 0.863585 | 0.519507 | -0.110459 | -0.027353 | 0.154092 | 0.048456 | -0.042934 | ... | -0.042142 | -0.000235 | -0.050125 | 0.129561 | 0.156732 | -0.090685 | 0.020337 | 0.019911 | 20.530000 | 0.0 |
| **75%** | 92.500000 | 1.108861 | 0.845177 | 1.531536 | 1.221565 | 0.427607 | 0.495493 | 0.642855 | 0.276433 | 0.334165 | ... | 0.102520 | 0.345747 | 0.073547 | 0.439519 | 0.472891 | 0.214745 | 0.088053 | 0.062731 | 67.635000 | 0.0 |
| **max** | 131.000000 | 1.492936 | 4.847323 | 3.562770 | 4.075817 | 3.281972 | 5.122103 | 4.808426 | 1.592032 | 4.009259 | ... | 2.270069 | 1.461535 | 2.458589 | 1.215279 | 1.136720 | 3.065576 | 2.490503 | 1.575380 | 3828.040000 | 0.0 |

8 rows × 31 columns

**Check for Missing Values and Duplicates**

```
print(df.isnull().sum())
print("Duplicate rows:", df.duplicated().sum())
```

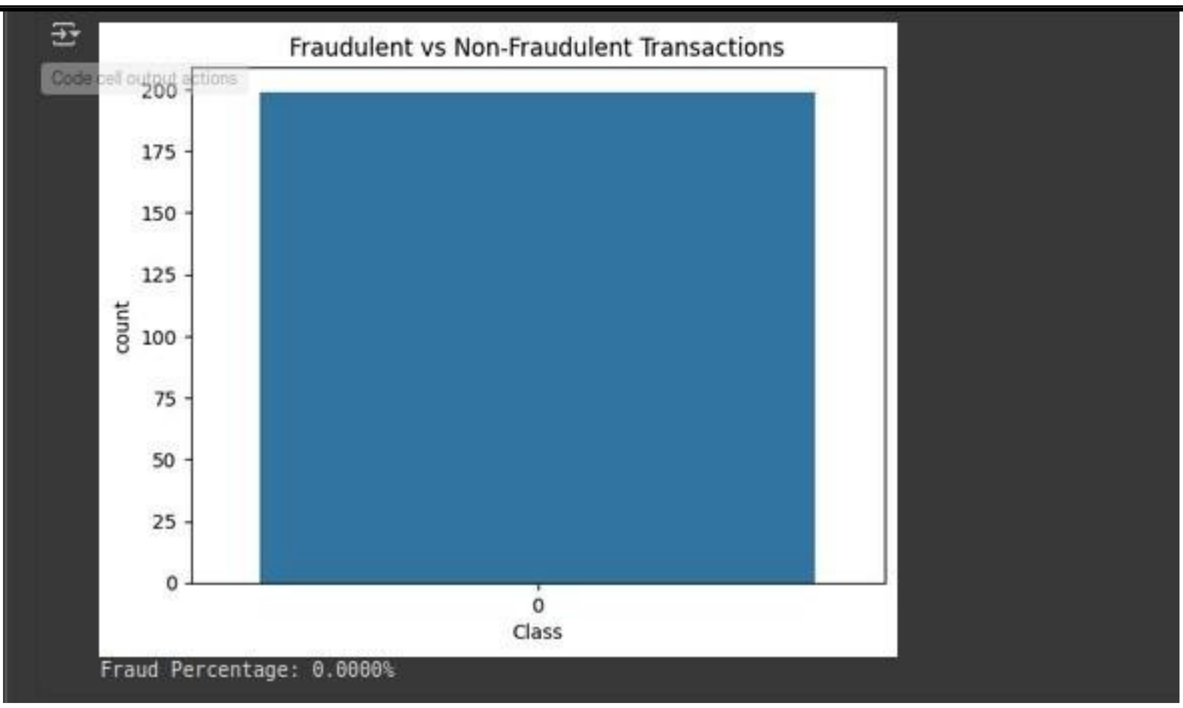


```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
Duplicate rows: 5
```

**Visualize a Few Features**

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='Class', data=df)
plt.title("Fraudulent vs Non-Fraudulent Transactions")
plt.show()
```

**fraud_ratio = df['Class'].value_counts(normalize=True).get(1, 0) * 100 print(f"Fraud Percentage: {fraud_ratio:.4f}%")**



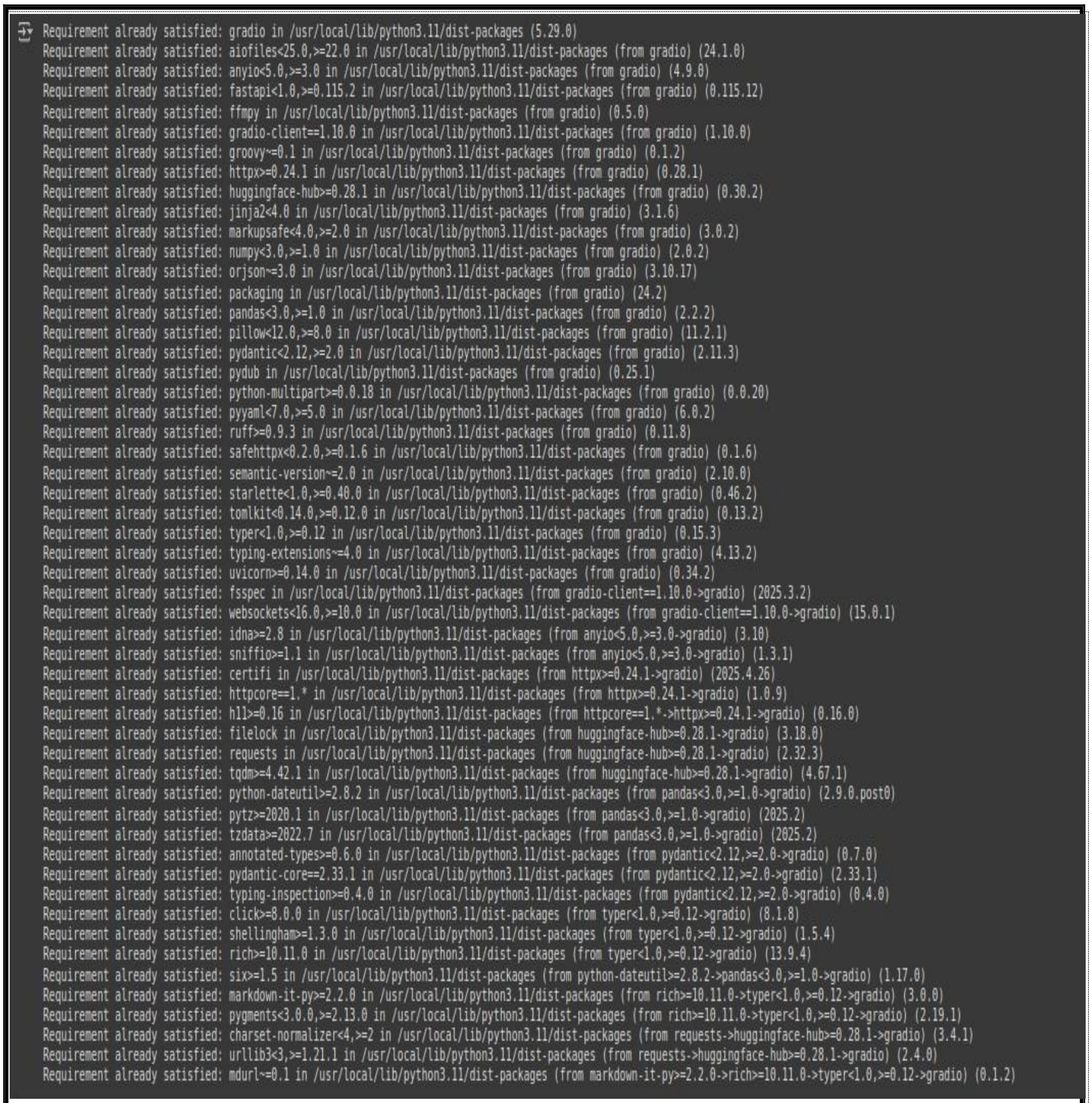
```
Fraud Percentage: 0.0000%
```

**Feature Scaling**

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import
StandardScaler X = df.drop('Class', axis=1)
y = df['Class']
X['Amount'] = StandardScaler().fit_transform(X['Amount'].values.reshape(-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

**Model Building**

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import
StandardScaler X = df.drop('Class', axis=1)
y = df['Class']
X['Amount'] = StandardScaler().fit_transform(X['Amount'].values.reshape(-1, 1))
**X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)**



```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.29.0)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.10.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.10.0)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.17)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.3)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.11.8)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.34.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (15.0.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33.1)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.4.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.19.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (2.4.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio) (0.1.2)
```

## 11. TEAM MEMBERS AND CONTRIBUTION

**Data Cleaning :kazimahmed**

**EDA: kazhim ahmed T.M**

**MODEL development : Lakshmanan**

**Documentation: Mokesh S**