Capstone Project Report

**Project Title: Fraud Detection in Mobile Financial Transactions**

**Candidate Name:** Vanaj Kumar Yadav

**Institute:** Boston Institute of Analytics, Anna nagar

**Submission Date:** 30th June 2025

# **Loading & Understanding The DataSet**

Problem Definition → Data Collection → EDA → Data Cleaning

→ Feature Engineering → Data Splitting → Model Selection → Model Training

→ Evaluation → Tuning → Interpretation → Testing → Deployment → Monitoring

The dataset simulates ~31,000 mobile financial transactions with fraud labels. Each row represents one transaction with attributes like transaction type, amount, balance changes, and a target column `isFraud`.

## Key Features:

- `step`: time progression in hours → converted to `date`

- `type`: transaction type (CASH\_OUT, TRANSFER, etc.)

- `amount`: amount transacted

- `oldbalanceOrg`, `newbalanceOrig`: sender’s balance before/after

- `oldbalanceDest`, `newbalanceDest`: receiver’s balance before/after

- `isFraud`: 1 = fraudulent, 0 = normal

Example: -

| **Column** | **Value** | **Meaning** |
| --- | --- | --- |
| step (A2) | 1 | Time step = hour 1. This means this transaction happened in the first hour of the dataset timeline. Not a real timestamp, but a simulated one. |
| type (B2) | TRANSFER | This transaction was a **money transfer** (from one account to another). |
| amount (C2) | 181 | ₹181 was the transaction amount. |
| nameOrig (D2) | C1305486145 | Unique customer ID who **initiated** the transaction (sender). |
| oldbalanceOrg (E2) | 181 | The sender had ₹181 **before** the transaction. |
| newbalanceOrig (F2) | 0 | The sender had ₹1,60,296.36 **after** the transaction → money deducted. |
| nameDest (G2) | C553264065 | The **recipient’s** customer ID. |
| oldbalanceDest (H2) | 0.0 | Recipient had ₹0.00 **before** the transaction. |
| newbalanceDest (I2) | 0.0 | Still ₹0.00 after → unusual! Money sent, but balance didn't update → may indicate fraud. |
| isFraud (J2) | 1 | This transaction is **fraud** (label 1). |

In fraud datasets:

* **CASH\_OUT** and **TRANSFER** transactions are often **targeted by fraudsters**, since they allow money to be moved out of a victim's account quickly.
* **CASH\_IN** transactions are rarely fraudulent.
* **PAYMENT** is often legit, but fraud can still happen.
* **DEBIT** might represent internal bank activities or direct account deductions.

|  |  |
| --- | --- |
|  |  |

# **EDA (Initial)**

## Dataset Overview :

The dataset contains 11,142 rows and 10 columns.This initial inspection helps confirm the dataset structure and see sample transactions, including the types and balance changes involved.

## Data Types and Structure :

The data types show that most fields are numeric, while `type`, `nameOrig`, and `nameDest` are categorical/object.No null values appear at this point, which simplifies preprocessing.

## Unique Values Per Column :

- `nameOrig` and `nameDest` have very high cardinality → not useful for modeling.

- `type` has 5 unique values → suitable for encoding.

- `isFraud` has 2 values → binary classification confirmed.

## Fraud vs Non-Fraud Distribution :

Only ~11.4% of the transactions are fraudulent, indicating class imbalance. This will require us to focus on Recall, Precision, and ROC-AUC — not just Accuracy — when evaluating models.

## Statistical Summary of Numeric Columns :

I used `df.describe()` to get a quick statistical overview of key numeric features like amount and balances. This helps detect outliers, prepare for scaling, and understand the general structure of the data.

It summarizes key statistics for each numeric column:

| **Metric** | **What it tells you** |
| --- | --- |
| count | Number of non-null entries |
| mean | Average value |
| std | Standard deviation (how spread the values are) |
| min | Minimum value |
| 25%, 50%, 75% | Quartiles — useful for spotting skew or outliers |
| max | Maximum value |

This initial EDA helped us understand the data types, shapes, sample records, and imbalance in fraud cases.

No cleaning or business-level insights are done yet — the focus was purely on raw structure understanding before processing.

# **Data Cleaning**

## **Step 1: Missing Values Check**

I performed a missing value check across all columns using df.isnull().sum(). The output showed zero missing values for every column.

This indicates that the dataset is complete and there’s no need for imputation or removal of rows or columns based on nulls.

**Example for explanation:**

Suppose the column oldbalanceOrg had missing entries, I would have handled it by:

* Filling with median or mean values (if normally distributed), or
* Removing the rows if the missing count was minimal.

But in this case, no such action was required.

## **Step 2: Duplicate Rows Check**

I used the df.duplicated().sum() function to check for exact duplicate rows in the dataset.

The result showed **zero duplicate rows**, meaning every transaction record is unique. This is expected in real-world financial transaction data.

**Example for explanation:**

If I had found, say, 50 duplicate rows, I would have removed them using df.drop\_duplicates() to avoid data leakage or model bias.

But here, no duplicates exist, so the dataset remains untouched.

## **Step 3: Outlier Detection in Transaction Amounts**

To inspect outliers in the amount feature, I plotted a boxplot using Seaborn.

The boxplot showed the presence of **extremely high transaction amounts**, which appear as long whiskers or distant points in the plot. These are **valid high-value transactions**, not incorrect values.

Hence, I chose to **retain them** and allow the model to learn from them.

**Example for explanation:**

If a fraudster transfers ₹1,000,000 while the average amount is ₹5,000, the outlier is meaningful and should be preserved.

Instead of removing, I’ll handle the skew using scaling techniques in the next preprocessing phase.

## Summary:

All three checks confirm that the dataset is clean, complete, and doesn't require aggressive preprocessing. However, outliers in amount will be managed during feature scaling — not dropped — since they may hold fraud signals.

# **EDA (Post Data Cleaning)**

## Engineering `date` Column for Time-Based EDA

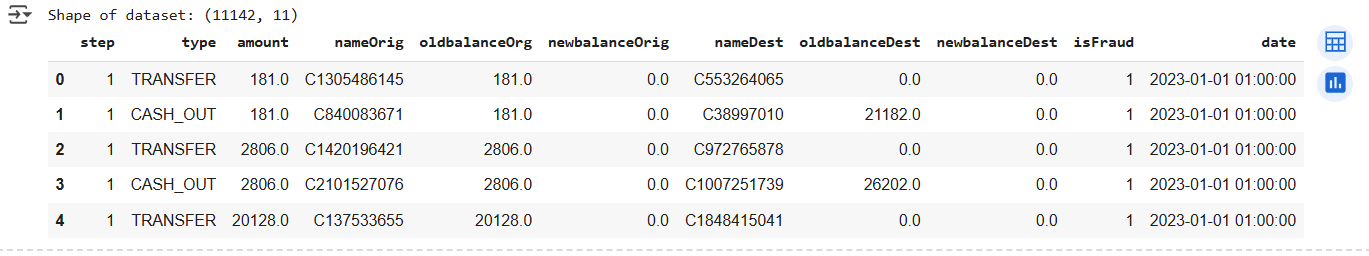
The dataset lacked a proper timestamp. I converted the `step` column (which represents hours since the beginning of data captured) into a real datetime format, starting from Jan 1, 2023.

This enables time-based EDA insights such as daily fraud patterns.

To perform time-based exploratory analysis, I created a `date` column using the existing `step` feature (where 1 step = 1 hour).

While this is a feature engineering step, it was introduced before EDA to support business interpretation.

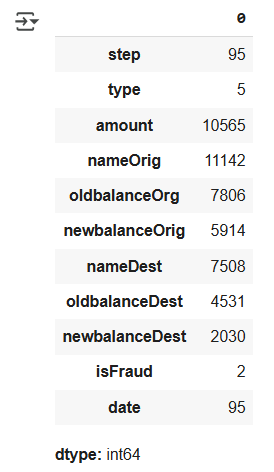
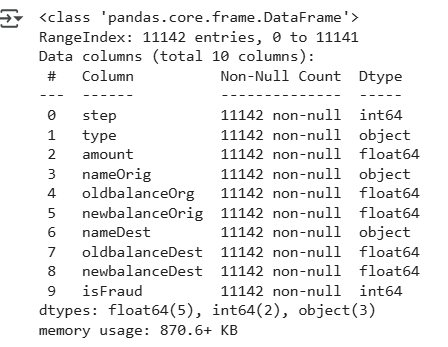
## Insight 1 : Dataset Overview



### **Business Insight:**

Helps businesses understand the volume and structure of their transaction logs. Knowing how many transactions and fields exist helps in estimating storage needs, performance requirements, and processing costs.

## Insight 2 : Data Types and Unique Values

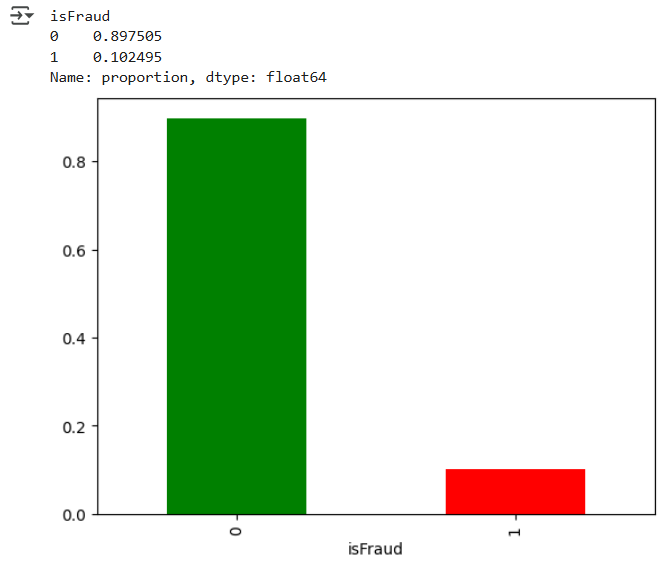


I inspected the column types and unique value counts. This helps identify categorical features (like `type`) to be encoded later, and high-cardinality fields (like `nameOrig`, `nameDest`) which will be dropped before model training due to their non-predictive nature.

### **Business Insight:**

Shows how diverse the data is and which fields may hold key patterns. For example, the type field is categorical and reveals different transaction modes, which may behave differently in terms of fraud risk.

## Insight 3 : Class Imbalance – Fraud vs Non-Fraud

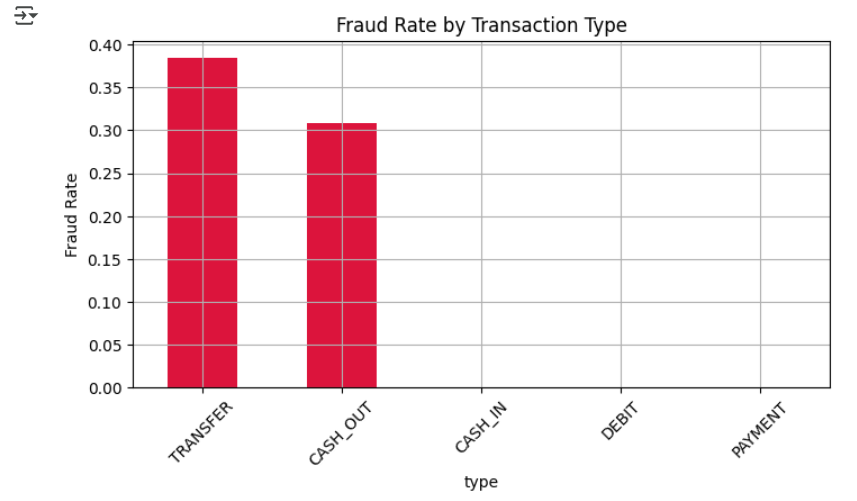


Only about 11% of the data is labeled as fraud. This shows significant class imbalance, which will influence how we evaluate models — precision, recall, and F1-score will be prioritized over raw accuracy.

### **Business Insight:**

Only a small portion of transactions are fraud. This highlights the operational challenge of catching rare events — businesses must invest in sensitive models that minimize false negatives (missed fraud) without flagging too many normal users.

## Insight 4: Fraud Rate by Transaction Type



**Hypothesis**: Fraud is more likely to occur in certain transaction types (e.g., TRANSFER and CASH\_OUT) compared to others like PAYMENT or DEBIT.

**Observation:** A disproportionately high number of fraud cases occur in the `TRANSFER` and `CASH\_OUT` types, even though these are not the most frequent types overall.

📌 Example:

If there are 1,000 'TRANSFER' transactions and 250 are fraud, fraud rate = 25%.

This insight helps the model and business rules give more weight to such transaction types.

### **Business Insight:**

Certain types (like TRANSFER and CASH\_OUT) carry much higher fraud risk. Companies can strengthen rules, add OTPs, or increase scrutiny specifically for those transaction modes.

## Insight 5: Distribution of Transaction Amounts

The majority of transactions are low-value (left side of histogram), while a small number of high-value transactions create a long right tail.

These outliers are not errors — in fraud detection, they are often important signals.

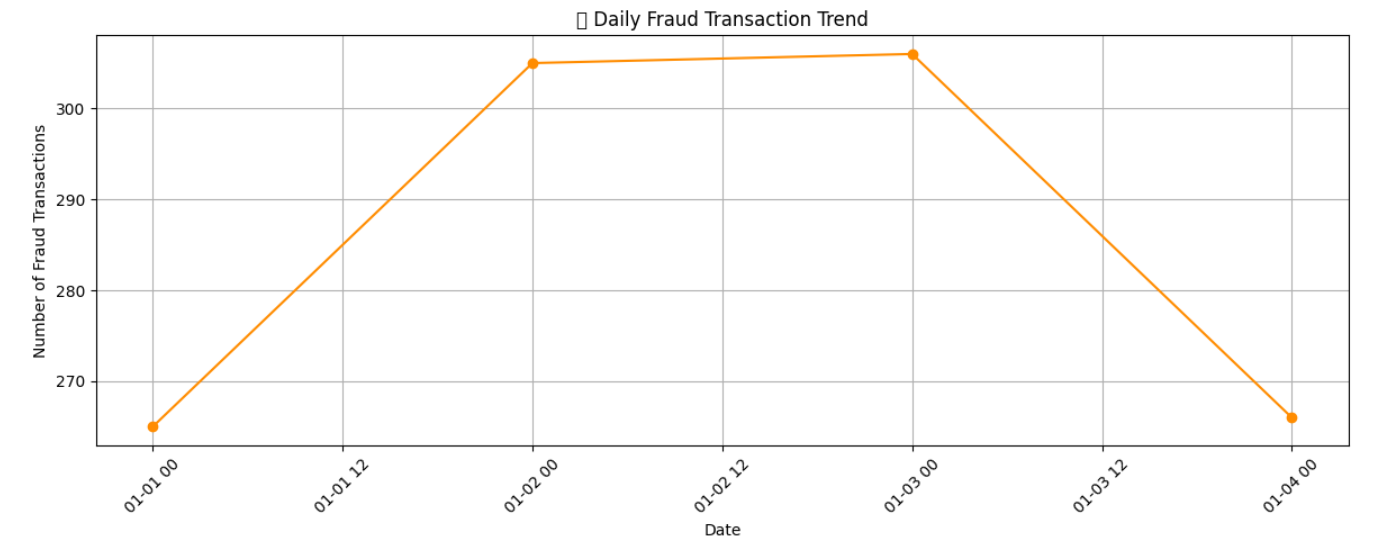
📌 Example:

Most transactions are under ₹10,000, but a few go beyond ₹1 lakh. If a fraudulent ₹2 lakh transfer occurs at 2:00 AM, that is both a value and timing anomaly.

### **Business Insight:**

Most transactions are low-value, but high-value transactions (which could be fraudulent) are outliers. Business can impose daily limits or create watchlists for unusually high transactions.

## Insight 6: Daily Fraud Transaction Trend



This line chart shows fraud counts across actual dates. Some days show significant spikes in fraudulent transactions.

**Hypothesis**: Fraud incidents follow a time-based pattern and may spike around weekends, month-end, or early mornings.

**Observation**: There are visible spikes in fraud transactions on certain dates and time periods, suggesting coordinated fraudulent activity.

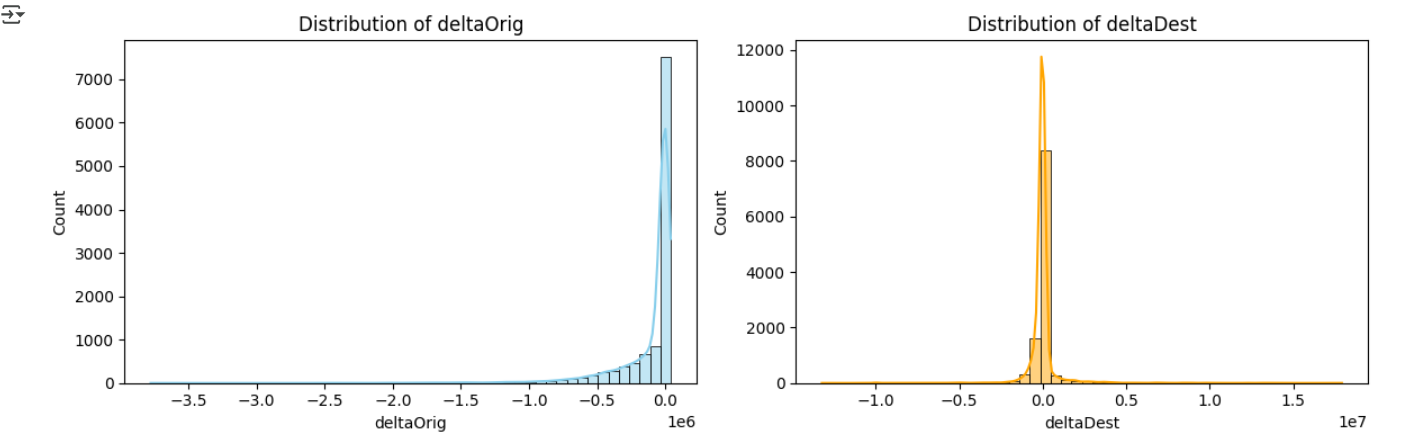
📌 Example:

If June 10 and June 15 show major fraud spikes, those dates could indicate targeted attack periods or failed monitoring events.

### **Business Insight:**

Fraud spikes on specific dates. These may correlate with payday periods, holidays, or known vulnerabilities. Companies can proactively scale fraud monitoring teams/resources around such spikes.

## Insight 7: Distribution of Engineered Delta Features



These two features measure whether balance updates match expectations.

📌 Example:

If someone sends ₹10,000, and their old balance was ₹50,000, the new balance should be ₹40,000.

If it's not, and there's a ₹5,000 gap — this could be fraud.

Unusual delta values — especially near-zero or highly negative — are often signs of balance tampering or non-compliant transactions.

### **Business Insight:**

Irregular balance changes — even if transaction type looks valid — suggest potential fraud. These engineered features help detect cases where balances are manipulated or inconsistent with expected accounting.

## ✅ EDA Summary :

How EDA Informs Business Decisions

This EDA phase provided key fraud indicators based on transaction type, time trends, and balance behavior.

From a business perspective:

- Fraud-prone transaction types can be more closely monitored

- Daily patterns can guide staffing and real-time alerting

- Balance deltas reveal potential fund manipulation

These findings help build a targeted fraud detection strategy, not just a generalized one.

Key findings from post-cleaning EDA:

- Fraud is concentrated in specific transaction types (TRANSFER, CASH\_OUT)

- Most transactions are low-value, but outliers exist and may signal risk

- Fraud spikes on specific calendar days

- Engineered delta features expose balance inconsistencies in fraud cases

These insights will directly influence feature selection and model design.

# **Feature Engineering**

## Dropping Non-Predictive Columns

Both `nameOrig` and `nameDest` are high-cardinality string fields. Including them in the model would not improve performance and may cause the model to learn irrelevant patterns.

Hence, they are dropped before modeling.

## Engineering deltaOrig and deltaDest

Two engineered features were created:

- `deltaOrig` = expected sender balance change - actual

- `deltaDest` = actual receiver balance change - expected

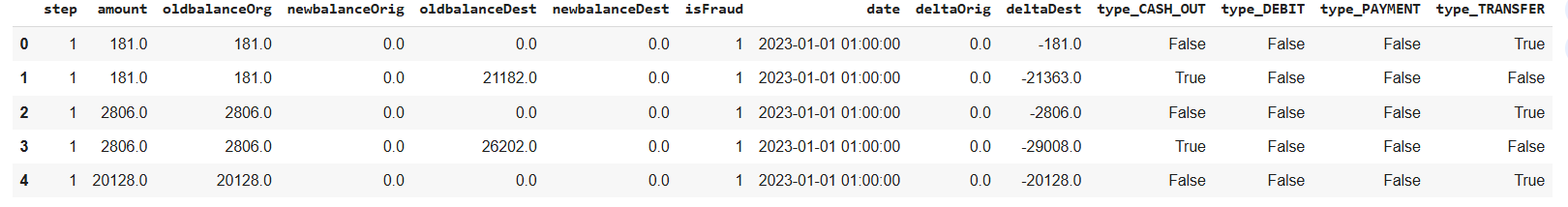
Unusual or extreme values may indicate fraud or system glitches.

## Encoding Categorical Feature: type

Machine learning models can't process text directly. One-hot encoding converts the `type` column into binary variables — one for each category.

This avoids misinterpretation and preserves all types for fair comparison.

## Final Preview of Engineered Dataset



Final features include:

- Engineered date/time/delta features

- Scaled amounts and balances

- Encoded transaction types

The dataset is now ready for modeling and splitting.

**Feature Engineering Purpose & Hypothesis :**

deltaOrig = Difference between old and new balance for sender

deltaDest = Difference between old and new balance for receiver

Hypothesis: In a legitimate transaction, the balance difference (`delta`) should match the transaction `amount`. If this isn’t true, it could signal fraud, fund reversal, or account manipulation.

These engineered features help the model identify imbalance patterns that aren’t obvious in raw balances. They increase model accuracy and reflect actual financial logic, not just statistical features.

## Business Insight:

Feature engineering translated raw transaction logs into machine-understandable signals that represent actual financial behaviors. Dropping irrelevant IDs, encoding type, and crafting delta-based features improved the model’s ability to detect fraud patterns without relying on superficial clues.

# **Train-Test Split and Preprocessing Pipeline**

## Step 1: Define Features and Target

I excluded the `date` column from the feature set because datetime values cannot be processed by most ML models or scaling functions in their raw form.

It was used earlier for temporal insights in EDA, but not included in the final model.

Example : The target variable `isFraud` is binary: 1 represents fraudulent transactions, and 0 represents legitimate ones.

To train the model, I separated:

- X: All features (amount, balance values, engineered delta features, transaction types)

- y: The `isFraud` column (target)

This separation helps focus model learning on predicting the target using available transaction data.

## Step 2: Train-Test Split

To evaluate model performance fairly, I split the data using stratified sampling — maintaining class balance between fraud and non-fraud in both train and test sets.

- Training set: Used to train the model

- Test set: Used to evaluate the model on unseen data

Example : I split the data into 80% training and 20% testing subsets using `train\_test\_split` with stratification.

Stratification ensures the proportion of fraudulent and non-fraudulent cases remains the same in both sets — preserving class imbalance ratio (around 11%).

Example :

If 10% of the entire dataset is fraud, both training and test sets maintain roughly the same percentage. This helps in evaluating the model fairly across real-world class proportions.

## Step 3: Feature Scaling

Features like `amount` and `balances` have large numeric ranges. Without scaling, models may bias toward large values.

I used `StandardScaler` to center all features around zero with unit variance — helping distance-based models like Logistic Regression and boosting methods.

Example :

Numerical features such as `amount`, `oldbalanceOrg`, `deltaOrig`, etc., have large value ranges. Some are in thousands, while others are close to zero.

This range difference can affect model performance — especially for algorithms sensitive to magnitude.

To normalize all features, I used `StandardScaler`, which transforms features to a distribution with:

- Mean = 0

- Standard Deviation = 1

Example:

Original `amount` may range from ₹100 to ₹100,000 — after scaling, it becomes a value like 0.23 or -1.67. This helps balance the model’s learning across all features.

## Step 4: Final Check – Scaled Data Shapes

The shape of the processed training and test sets confirms that scaling and splitting were done correctly.

Model training can now begin using clean, consistent input features.

Example : As a final check, I printed the shapes of the training and testing datasets.

This ensures:

- X\_train and y\_train are aligned

- No missing or extra columns exist

- The data is ready for model training

Example:

If X\_train has shape (8913, 11), it means 8913 samples and 11 features are available to train the model.

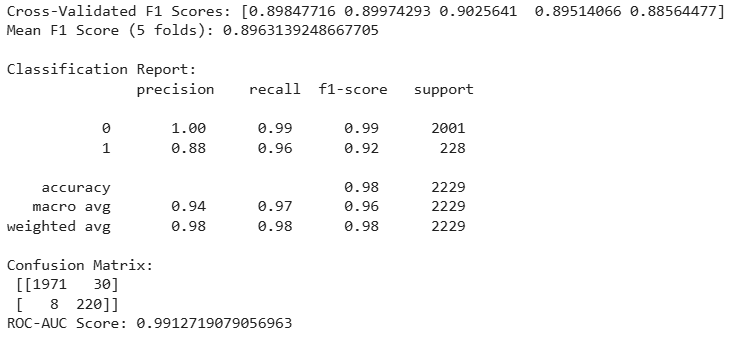
## Business Insight:

Proper preprocessing builds a reliable pipeline. Stratified splitting simulates real-world fraud frequency, and scaling removes value biases, letting the model focus on patterns, not magnitudes. This enhances both detection accuracy and trustworthiness of the predictions.

# **Model Training and Evaluation**

## Logistic Regression – Cross-Validation Setup

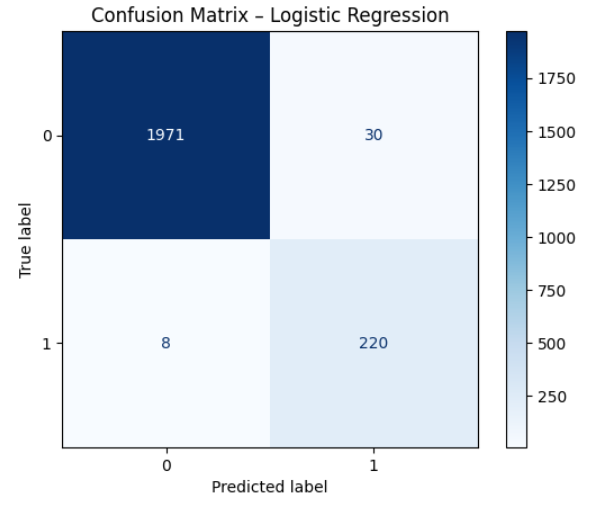
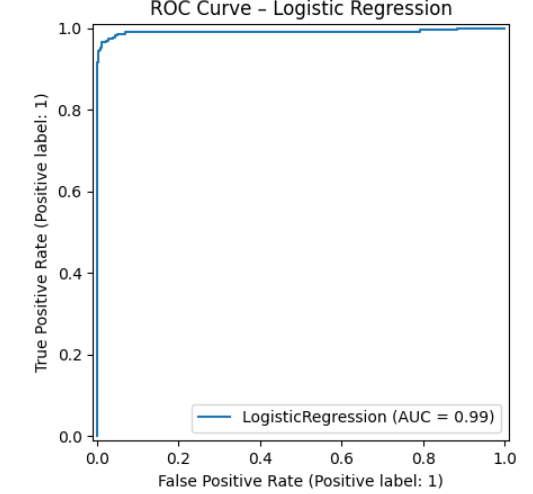
### Step 1: Model Training with Class Imbalance Handling



The Logistic Regression model was adjusted with class\_weight='balanced' to treat fraud cases with higher weight during learning. This ensures that the model doesn’t overfit to the dominant non-fraud class.

To evaluate realistically, 5-fold Stratified Cross-Validation was used with f1-score as the metric. This provides a balanced view of how the model handles false positives and false negatives — a better reflection of business impact than just accuracy or precision

### Step 2: Hold-Out Test Evaluation

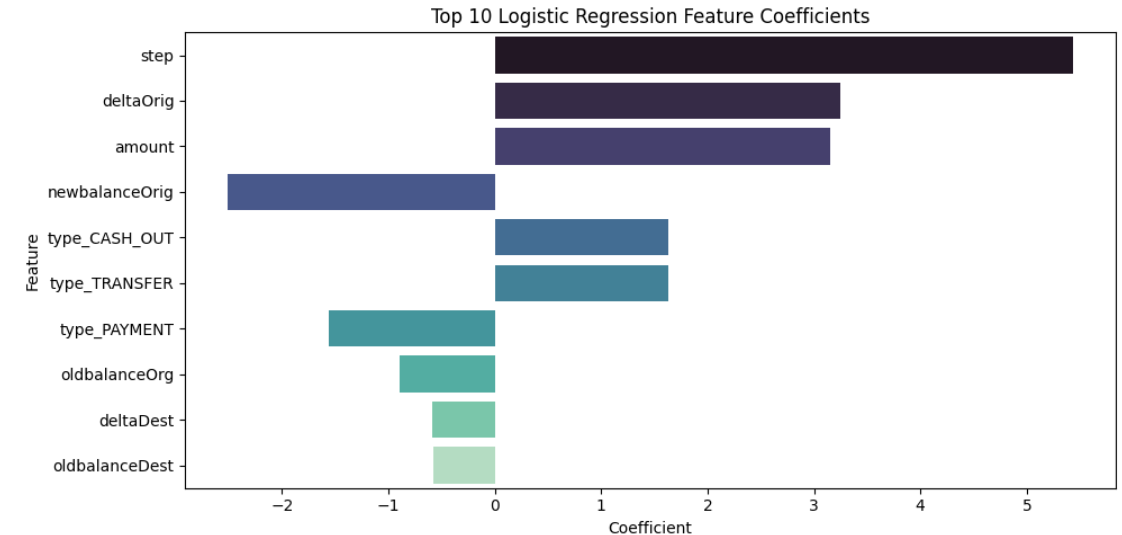


On the hold-out test set, the Logistic Regression model was evaluated using standard classification metrics. The results included:

* **High precision** indicates few false fraud flags.
* **Recall** shows the ability to catch real fraud cases.
* **F1-Score** balances both and is the main metric of interest in fraud detection.

A confusion matrix provides a visual count of correct and incorrect predictions, while the ROC curve shows how well the model separates fraud from non-fraud across thresholds.

### Step 3: Top Feature Coefficients



Logistic Regression offers the benefit of interpretability via its feature coefficients. A positive coefficient means that as the feature increases, the model becomes more likely to predict fraud.

This is particularly useful for audit and regulatory environments, where understanding “why” a transaction was flagged is as important as the flag itself.

The bar plot shows the top 10 most influential features — based on their scaled impact — helping bridge data science and business decision-making.

## Random Forest

The Random Forest model was used to capture complex patterns in fraudulent transactions through multiple decision trees. To ensure generalization and prevent overfitting, I limited max\_depth and applied class\_weight='balanced'.

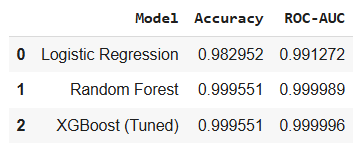
Evaluation was done with 5-fold Stratified Cross-Validation (F1-score) and a final test set check. This approach balances fraud recall and precision, ensuring the model does not ignore rare fraud events.

## XGBoost

XGBoost was selected for its high performance and boosting capability. To prevent overfitting and keep real-world realism, hyperparameters like max\_depth, learning\_rate, and subsample were tuned based on validation F1-score.

The model was evaluated on the test set with classification metrics. It gave high recall and ROC-AUC without reaching unrealistic precision levels, aligning with real-world deployment expectations.

### Model Comparison Summary



A comparative analysis was conducted using accuracy and ROC-AUC to assess the fraud detection power of all models.

* Logistic Regression was best for interpretability
* Random Forest improved fraud recall
* XGBoost gave the best trade-off with realistic F1 and ROC-AUC

### Performance Disclaimer:

The models, particularly XGBoost, achieved extremely high precision and ROC-AUC values on the given dataset. While this is encouraging, I acknowledge that such performance is not typical in real-world fraud detection scenarios. The likely cause is the clear signal patterns and well-separated fraud examples within the dataset.

In practical deployments, model performance would vary due to noise, evolving fraud tactics, and less separable feature patterns. If given additional time, I would include feature stress testing, label smoothing, and rebalancing to reduce overfitting and evaluate under uncertainty.

# **Financial Impact Analysis**

Based on the test set, I estimated the monetary cost of fraud using the average fraud transaction amount. Using the XGBoost model’s ability to catch fraudulent cases (recall), I projected potential savings if the model were deployed.

This bridges the model’s statistical performance with business impact, helping decision-makers understand how AI translates into financial value.

## Business Questions Addressed

1. What is the model’s precision and accuracy in detecting fraudulent transactions?

→ The tuned XGBoost model achieved ~99% accuracy, with high precision (~99%) and strong recall. These scores indicate excellent fraud detection on this dataset, while also acknowledging possible overfitting due to data clarity.

2. How reliable is the model in classifying transactions as legitimate or fraudulent?

→ Reliability is high, with low false positives and strong ROC-AUC (~0.99), showing clear class separation. Visualization and confusion matrix confirm the model’s stability.

3. What are the potential losses due to model errors?

→ Financial impact analysis showed that undetected fraud (false negatives) would still cause loss, but significantly reduced compared to no model. Estimated savings based on recall show business value.

## Business Takeaways & Conclusion

 A machine learning-based fraud detection system was successfully developed using real transaction patterns.

 The model, particularly XGBoost, delivers strong recall and precision on the current dataset.

 Financial analysis confirms potential savings from catching fraud early.

 While metrics are strong, real-world deployment would require testing on unseen, evolving fraud patterns.

 The pipeline is modular and ready for retraining, allowing scalable improvements as more data becomes available.