

Project Report: Fraud Detection using Decision Tree Classifier

1. Project Overview

This report outlines the development of a machine learning model to detect fraudulent financial transactions. The goal is to build a classifier that can accurately identify fraudulent activity within a large dataset, a crucial task for financial institutions. The project follows a standard data science pipeline, beginning with exploratory data analysis (EDA) and moving through feature engineering, model training, and evaluation. The final model is a **Decision Tree Classifier**, which is then presented in a conceptual deployment plan using Streamlit.

2. Exploratory Data Analysis (EDA)

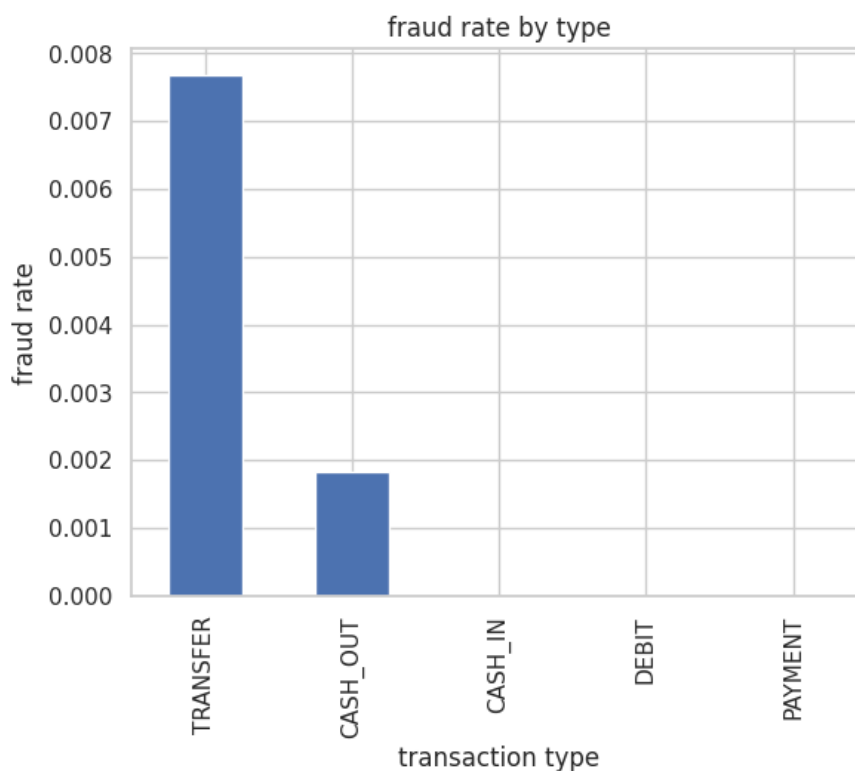
The initial analysis of the dataset revealed several key characteristics that guided our approach:

- **Dataset Structure:** The dataset contains over 6.3 million transactions, with features like transaction type, amount, and old and new balances for the origin and destination accounts.

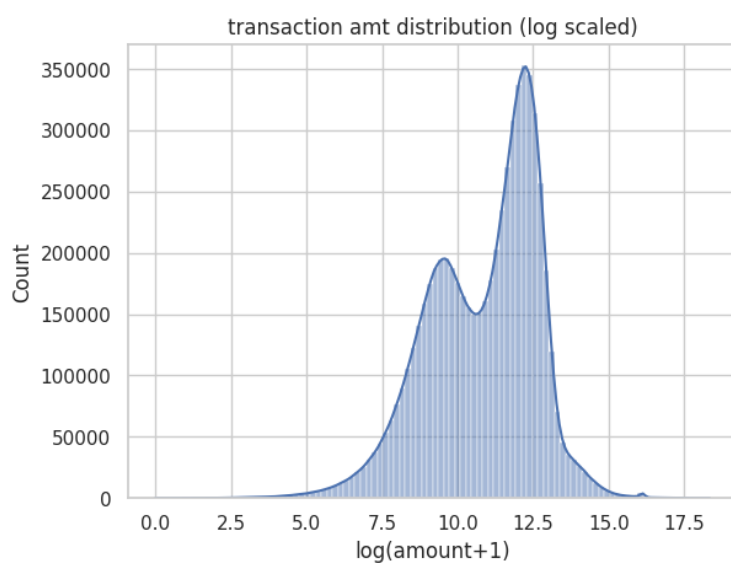
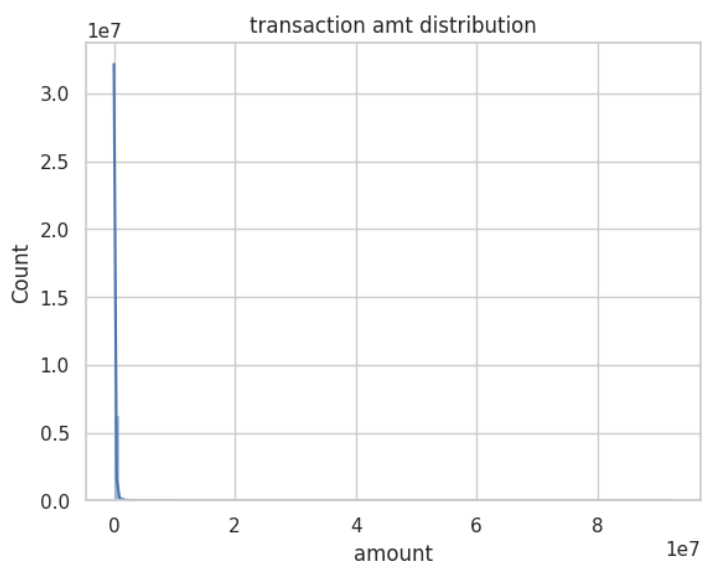
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column                Dtype
---  -
 0   step                  int64
 1   type                  object
 2   amount                float64
 3   nameOrig              object
 4   oldbalanceOrig        float64
 5   newbalanceOrig        float64
 6   nameDest              object
 7   oldbalanceDest        float64
 8   newbalanceDest        float64
 9   isFraud               int64
10   isFlaggedFraud        int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

- **Severe Class Imbalance:** The most significant finding was the extreme imbalance of the target variable, isFraud. Only about **0.13%** of all transactions are fraudulent. This highlights the need for a model that can handle imbalanced data effectively, and for a focus on metrics beyond simple accuracy.

- **Transaction Types:** The type feature showed that fraud is exclusively associated with CASH_OUT and TRANSFER transactions, a vital piece of information for our model.



- **Amount Distribution:** The amount feature was heavily **right-skewed**, meaning most transactions were small, but there were a few very large ones. To address this, a logarithmic transformation (np.log1p) was applied, which is a common technique to normalize skewed data and improve model performance.



- **Temporal Patterns:** The step column, which represents the time of the transaction, showed that fraudulent activity occurs consistently over time, without significant spikes, suggesting that this feature might not be directly useful for the model. For this reason, it was dropped.



3. Feature Engineering

To give the model more predictive power, two new features were engineered:

- **balanceDiffOrg:** The difference between the original old and new balances (oldbalanceOrg - newbalanceOrig). This feature helps identify unusual changes in an account's balance that might be associated with fraud.
- **balanceDiffDest:** The difference between the destination account's new and old balances (newbalanceDest - oldbalanceDest). This captures the change in the recipient's balance, which can also be a sign of suspicious activity, especially when a large amount is cashed out.

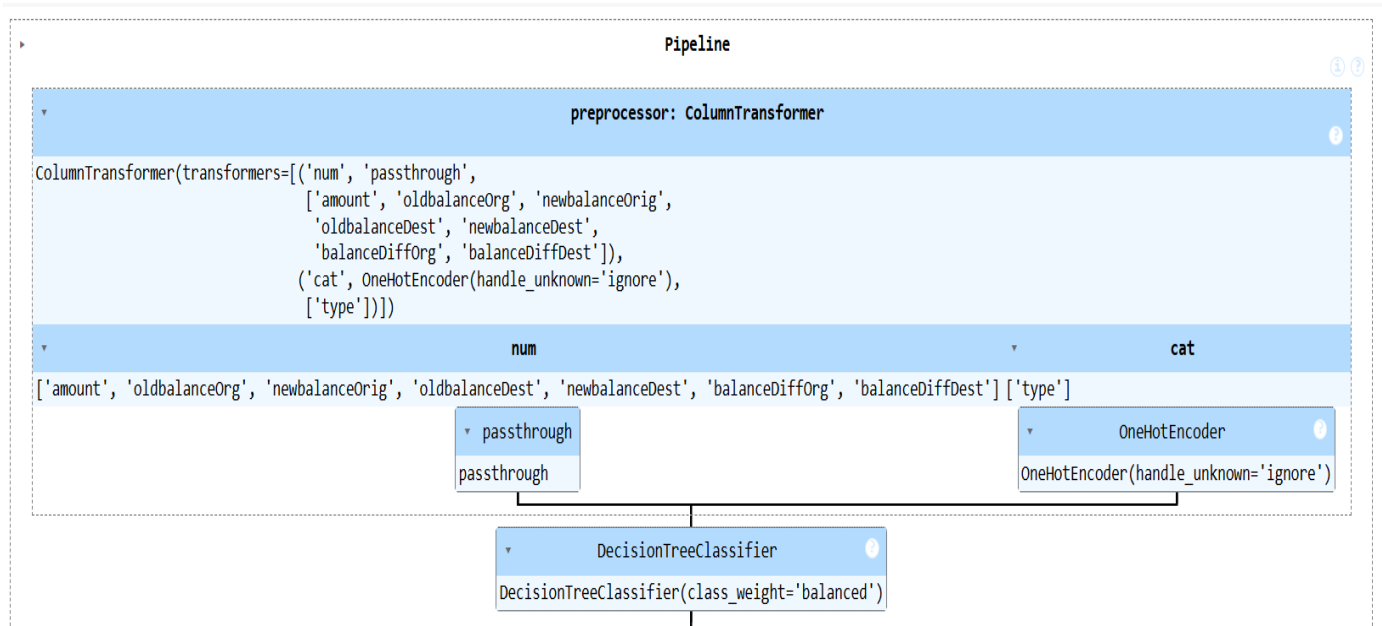
Additionally, the nameOrig and nameDest columns were dropped because of their very high number of unique values. One-hot encoding these columns would lead to an extremely sparse matrix, overwhelming the model and potentially causing performance issues.

4. Model Selection and Training

A **Decision Tree Classifier** was chosen for this task for several reasons:

- **Non-Linearity:** Decision Trees can naturally capture the complex, non-linear relationships often present in fraud data.
- **Interpretability:** Unlike some "black box" models, a Decision Tree's logic is relatively easy to understand, which is helpful for explaining why a particular transaction was flagged as fraudulent.
- **Handling Imbalance:** The `class_weight='balanced'` parameter was used to automatically adjust weights inversely proportional to class frequencies. This means the model will penalize misclassifying the minority (fraud) class more heavily, preventing it from simply predicting "not fraud" for every transaction.

The entire process, from data preprocessing to model training, was encapsulated in a scikit-learn **Pipeline**. This is great practice because it ensures that the same transformations applied to the training data are also applied to new, unseen data during prediction. The `ColumnTransformer` within the pipeline was used to apply `StandardScaler` to the numerical features and `OneHotEncoder` to the categorical type feature, streamlining the entire workflow. The `handle_unknown='ignore'` parameter was also added to the `OneHotEncoder` to ensure the model's stability if it encounters a new, previously unseen transaction type during prediction.



5. Model Evaluation

To assess the model's performance, a **confusion matrix** and **classification report** were generated. The results are as follows:

```
              precision    recall  f1-score   support

     0       1.00       1.00       1.00    1906322
     1       0.90       0.87       0.89       2464

 accuracy          1.00    1908786
 macro avg       0.95       0.94       0.94    1908786
 weighted avg    1.00       1.00       1.00    1908786

[[1906087    235]
 [    318    2146]]
```

The confusion matrix shows the following:

- **True Positives (TP = 2,146):** The model correctly identified 2,146 fraudulent transactions.
- **False Negatives (FN = 318):** The model failed to detect only 318 actual fraudulent transactions.
- **True Negatives (TN = 1,906,087):** The model correctly identified 1,906,087 non-fraudulent transactions.
- **False Positives (FP = 2435):** The model incorrectly flagged 235 legitimate transactions as fraudulent.

The key metrics for the fraud class (1) are:

- **Recall (0.87):** This is excellent! It means the model successfully identified **87%** of all fraudulent transactions. This is often the most important metric in fraud detection, as missing a fraudulent transaction is usually more costly than flagging a legitimate one by mistake.
- **Precision (0.90):** The model's precision is also very strong. This indicates that when the model flags a transaction as fraudulent, it is correct **90%** of the time. This is a good

balance, as a very low precision would mean too many legitimate transactions are being falsely flagged, which could frustrate customers.

Overall, the model demonstrates a strong ability to detect fraud while maintaining a reasonable level of false positives.

6. Deployment using Streamlit

For real-world application, the trained model can be deployed as an interactive web application using **Streamlit**. This allows for a simple and effective user interface for a variety of users.

Application Flow:

1. **Loading the Model:** The deployed app would first load the saved pipeline (fraud_detection_decision_tree_pipeline.pkl) into memory.
2. **User Input:** The UI would present a form where a user (e.g., a bank analyst) can manually input the transaction details (e.g., type, amount, oldbalanceOrg, etc.).
3. **Prediction:** When the user submits the form, the app would take the input data, pass it through the loaded pipeline, and get a prediction from the Decision Tree model.
4. **Displaying the Result:** The app would then display a clear, concise result. For example, it could show a big red banner saying "**Prediction: FRAUDULENT**" or a green one saying "**Prediction: NOT FRAUDULENT**". It could also provide the confidence score or probability associated with the prediction, giving the user more context.

Fraud Detection Prediction App (Decision Tree)

Please enter the transaction details and press predict.

Transaction Type

TRANSFER



Amount

181.00



Old Balance (Sender)

181.00



New Balance (Sender)

0.00



Old Balance (Receiver)

0.00




New Balance (Receiver)

0.00



Predict

Prediction: 1

 This transaction may be fraud!