Fraudulent Transaction Detector

Part 3

Arya Goyal (ag9961)

Gitesh Chinawalkar (gc3410)

Introduction

With the increasing prevalence of financial transactions conducted digitally, detecting and mitigating fraudulent activities has become paramount. This project integrates a robust database schema with a state-of-the-art machine learning model to identify fraudulent transactions in real-time. The end goal is to enhance security, reduce financial losses, and improve trust in digital financial ecosystems.

The database schema was carefully designed to store transactional, user, and fraud detection log data efficiently. A machine learning model analyzes patterns within this data to flag suspicious activities. The system operates seamlessly, leveraging both structured and unstructured data for analytics and predictions.

Machine Learning Model Selection and Training

1. Data Selection

The dataset used for training the model consists of transactional data, including features like transaction type, amount, sender and receiver balances, and flags for suspected fraud. Missing and anomalous values were handled carefully to ensure model integrity.

Features:

- amount: Transaction amount.
- oldbalanceOrg and newbalanceOrig: Initial and post-transaction balances of the sender.
- oldbalanceDest and newbalanceDest: Initial and post-transaction balances of the recipient.
- type: Type of transaction (e.g., CASH_IN, CASH_OUT, PAYMENT).
- isFraud: Label indicating whether the transaction is fraudulent.

The dataset was split into training, validation, and test sets in an 80:10:10 ratio.

2. Algorithm Selection

The following algorithms were considered:

1. **Logistic Regression**: For its simplicity and interpretability.
2. **Random Forest**: For handling feature interactions and imbalanced data.
3. **SVM**: For its ability to handle large datasets efficiently and deliver high accuracy.

The final model selected was **Random forest**, chosen for its superior performance in fraud detection tasks.

3. Model Training and Insights Extraction

The model was trained using:

- Training Data: To fit the model.
- Validation Data: For hyperparameter tuning.
- Test Data: To evaluate real-world performance.

Key Metrics:

- Precision, Recall, and F1 Score for imbalanced class evaluation.
- AUC-ROC for overall model effectiveness.

Features were engineered to include:

- Differences between old and new balances.
- Ratio of transaction amount to account balances.

The trained model effectively identified patterns associated with fraudulent behavior, such as high-value transactions with zero account balances.

4. Business Use Cases

1. Fraud Detection: Automatically flagging high-risk transactions.
2. Risk Profiling: Assigning fraud risk scores to accounts.
3. Regulatory Reporting: Providing reports on flagged transactions for compliance.

5. Fraud Detection Model

- **Data Import**

Data Import and description

```
[6] df1 = pd.read_csv('/content/fraud_0.1origbase.csv')
```

```
[7] df1.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	283	CASH_IN	210329.84	C1159819632	3778062.79	3988392.64	C1218876138	1519266.60	1308936.76	0.0	0.0
1	132	CASH_OUT	215489.19	C1372369468	21518.00	0.00	C467105520	6345756.55	6794954.89	0.0	0.0
2	355	DEBIT	4431.05	C1059822709	20674.00	16242.95	C76588246	80876.56	85307.61	0.0	0.0
3	135	CASH_OUT	214026.20	C1464960643	46909.73	0.00	C1059379810	13467450.36	13681476.56	0.0	0.0
4	381	CASH_OUT	8858.45	C831134427	0.00	0.00	C579876929	1667180.58	1676039.03	0.0	0.0

```
] df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 216746 entries, 0 to 216745
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   step              216746 non-null  int64
 1   type              216746 non-null  object
 2   amount            216746 non-null  float64
 3   name_orig         216745 non-null  object
 4   oldbalance_org    216745 non-null  float64
 5   newbalance_orig   216745 non-null  float64
 6   name_dest         216745 non-null  object
 7   oldbalance_dest   216745 non-null  float64
 8   newbalance_dest   216745 non-null  float64
 9   is_fraud          216745 non-null  float64
 10  is_flagged_fraud  216745 non-null  float64
dtypes: float64(7), int64(1), object(3)
memory usage: 18.2+ MB
```

Data Cleaning

```
df1.isna().mean()
```

	0
step	0.000000
type	0.000000
amount	0.000000
name_orig	0.000005
oldbalance_org	0.000005
newbalance_orig	0.000005
name_dest	0.000005
oldbalance_dest	0.000005
newbalance_dest	0.000005
is_fraud	0.000005
is_flagged_fraud	0.000005

dtype: float64

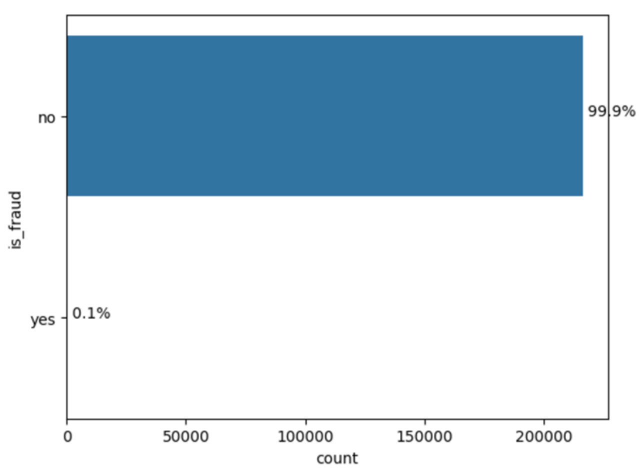
```
[17] # Handle missing values
     df1['name_orig'] = df1['name_orig'].fillna(0).astype(str).apply(lambda i: i[0])
     df1['name_dest'] = df1['name_dest'].fillna(0).astype(str).apply(lambda i: i[0])
```

```
[20] df1['is_fraud'] = df1['is_fraud'].map({1: 'yes', 0: 'no'})
     df1['is_flagged_fraud'] = df1['is_flagged_fraud'].map({1: 'yes', 0: 'no'})
```

- ## Exploratory data analysis

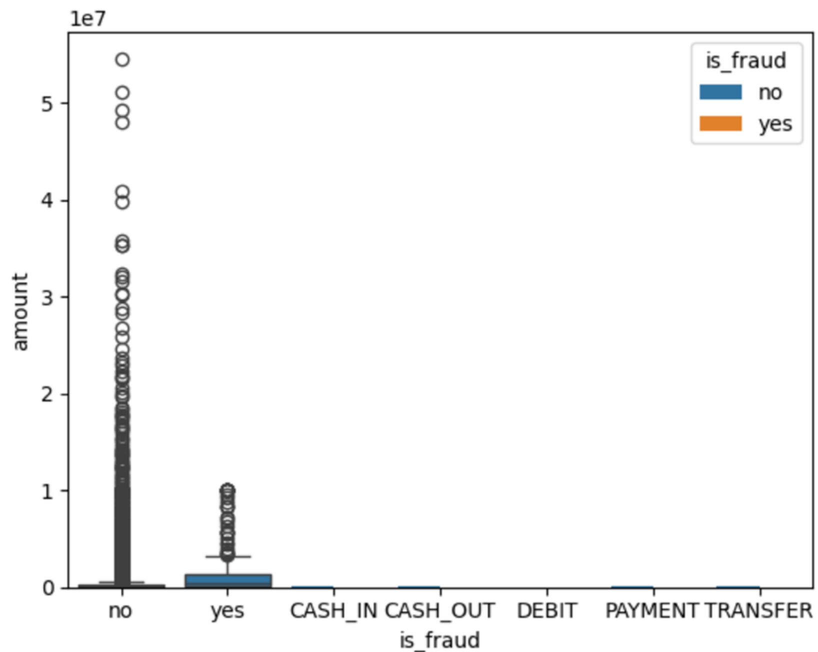
Expolatory data analysis

```
ax = sns.countplot(y='is_fraud', data=df2);

total = df2['is_fraud'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```



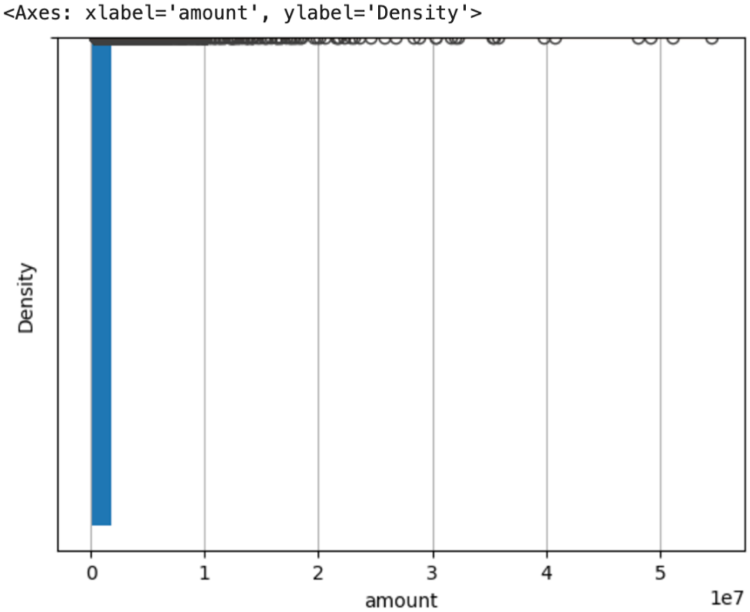
```
sns.boxplot(x='is_fraud', y='amount', data=df2)
sns.countplot(x='type', hue='is_fraud', data=df2)
```

```
<Axes: xlabel='is_fraud', ylabel='amount'>
```

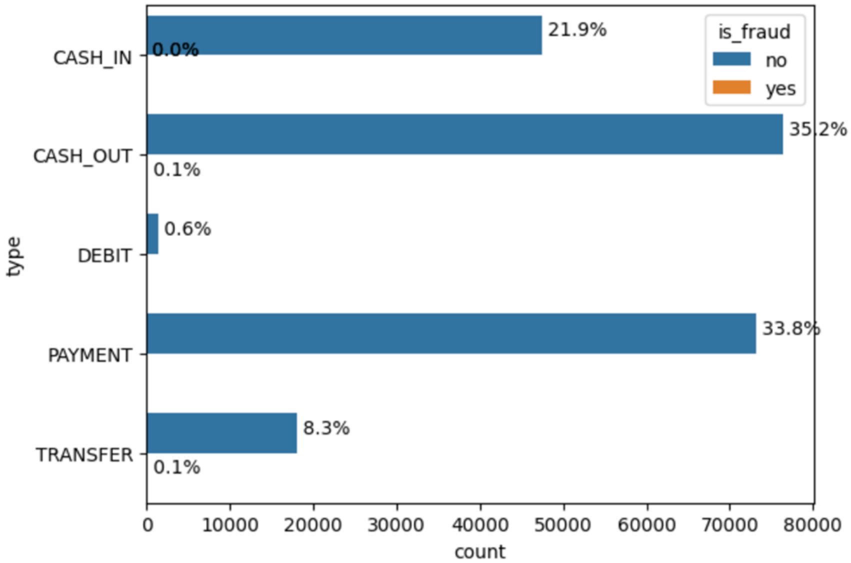



```
df2['amount'].hist(bins=30)
sns.boxplot(x=df2['amount'])
sns.kdeplot(df2['amount'])
```

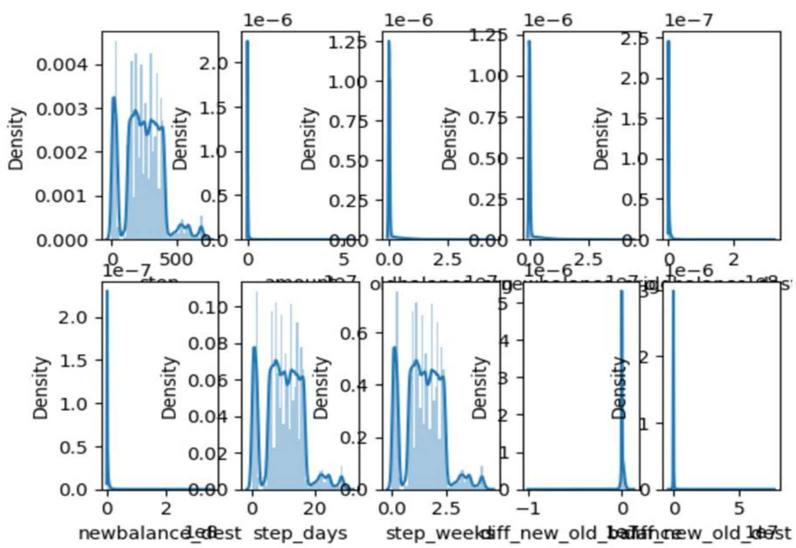
```
<Axes: xlabel='amount', ylabel='Density'>
```



```
ax = sns.countplot(y='type', hue='is_fraud', data=df2)

total = df2['type'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```




```
columns = num_attributes.columns.tolist()
j = 1

for column in columns:
    plt.subplot(2, 5, j)
    sns.distplot(num_attributes[column]);

    j += 1
```



- ## Data Preparation

Data Preparation

```
[38] X = df2.drop(columns=['is_fraud', 'is_flagged_fraud', 'name_orig', 'name_dest',
                          'step_weeks', 'step_days'], axis=1)
     y = df2['is_fraud'].map({'yes': 1, 'no': 0})
```

```
print(y.isna().sum())
print(X.isna().sum())
y = y.fillna(0)
X = X.fillna(0)
```

```
0
step                    0
type                    0
amount                  0
oldbalance_org          1
newbalance_orig         1
oldbalance_dest         1
newbalance_dest         1
diff_new_old_balance    1
diff_new_old_destiny    1
dtype: int64
```

```
[55] X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=.2, stratify=y)
     X_train, X_valid, y_train, y_valid = train_test_split(X_temp, y_temp, test_size=.2, stratify=y_temp)
```

```
num_columns = ['amount', 'oldbalance_org', 'newbalance_orig', 'oldbalance_dest', 'newbalance_dest',
               'diff_new_old_balance', 'diff_new_old_destiny']
mm = MinMaxScaler()
X_params = X_temp.copy()

X_train[num_columns] = mm.fit_transform(X_train[num_columns])
X_valid[num_columns] = mm.transform(X_valid[num_columns])

X_params[num_columns] = mm.fit_transform(X_temp[num_columns])
X_test[num_columns] = mm.transform(X_test[num_columns])
```

```
final_columns_selected = ['step', 'oldbalance_org',
                          'newbalance_orig', 'newbalance_dest',
                          'diff_new_old_balance', 'diff_new_old_destiny',
                          'type_TRANSFER']
```

```
X_train_cs = X_train[final_columns_selected]
X_valid_cs = X_valid[final_columns_selected]

X_temp_cs = X_temp[final_columns_selected]
X_test_cs = X_test[final_columns_selected]

X_params_cs = X_params[final_columns_selected]
```

```
5] X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=.2, stratify=y)
   X_train, X_valid, y_train, y_valid = train_test_split(X_temp, y_temp, test_size=.2, stratify=y_temp)
```

```
!pip install category_encoders
from category_encoders import OneHotEncoder

ohe = OneHotEncoder(cols=['type'], use_cat_names=True)

X_train = ohe.fit_transform(X_train)
X_valid = ohe.transform(X_valid)

X_temp = ohe.fit_transform(X_temp)
X_test = ohe.transform(X_test)
```

- ## Machine learning model

```
[ ] #SVM
```

```
[62] def ml_scores(model_name, y_true, y_pred):

         accuracy = balanced_accuracy_score(y_true, y_pred)
         precision = precision_score(y_true, y_pred)
         recall = recall_score(y_true, y_pred)
         f1 = f1_score(y_true, y_pred)
         kappa = cohen_kappa_score(y_true, y_pred)

         return pd.DataFrame({'Balanced Accuracy': np.round(accuracy, 3),
                              'Precision': np.round(precision, 3),
                              'Recall': np.round(recall, 3),
                              'F1': np.round(f1, 3),
                              'Kappa': np.round(kappa, 3)},
                             index=[model_name])
```

```
[63] svm = SVC()
     svm.fit(X_train_cs, y_train)

     y_pred = svm.predict(X_valid_cs)
```

```
svm_results = ml_scores('SVM', y_valid, y_pred)
svm_results
```

	Balanced Accuracy	Precision	Recall	F1	Kappa
SVM	0.5	0.0	0.0	0.0	0.0

```
[65] print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     34634
         1.0       0.00      0.00      0.00        46

    accuracy                           1.00     34680
   macro avg       0.50      0.50      0.50     34680
weighted avg       1.00      1.00      1.00     34680
```

```
[ ] #Logistic regression
```

```
[66] lg = LogisticRegression()
     lg.fit(X_train_cs, y_train)

     y_pred = lg.predict(X_valid_cs)
```

```
[67] lg_results = ml_scores('Logistic Regression', y_valid, y_pred)
     lg_results
```

	Balanced Accuracy	Precision	Recall	F1	Kappa
Logistic Regression	0.5	0.0	0.0	0.0	0.0

```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     34634
         1.0       0.00      0.00      0.00        46

    accuracy                           1.00     34680
   macro avg       0.50      0.50      0.50     34680
weighted avg       1.00      1.00      1.00     34680
```

```
[ ] #Random forest
```

```
[69] rf = RandomForestClassifier(class_weight='balanced')
     rf.fit(X_train_cs, y_train)

     y_pred = rf.predict(X_valid_cs)
```

```
[70] rf_results = ml_scores('Random Forest', y_valid, y_pred)
     rf_results
```

	Balanced Accuracy	Precision	Recall	F1	Kappa
Random Forest	0.859	1.0	0.717	0.835	0.835

```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     34634
         1.0       1.00      0.72      0.84        46

    accuracy                           1.00     34680
   macro avg       1.00      0.86      0.92     34680
weighted avg       1.00      1.00      1.00     34680
```

- ## Compare results

Compare results

```
[73] modeling_performance = pd.concat([lg_results, rf_results,svm_results])
     modeling_performance.sort_values(by="F1", ascending=True)
```

	Balanced Accuracy	Precision	Recall	F1	Kappa
Logistic Regression	0.500	0.0	0.000	0.000	0.000
SVM	0.500	0.0	0.000	0.000	0.000
Random Forest	0.859	1.0	0.717	0.835	0.835

- ## Github Link

https://github.com/Aryagoy/Fraud-transaction-detection