FRAUD DETECTION SYSTEM USING MULTIPLE ML MODELS

By: *DHANUSH TADISETTI*

Gmail: dhanushtadisetti@gmail.com

To Project Mentor: *N. MUVENDIRAN SIR*.

INTRODUCTION:

Credit card fraud detection is the process of identifying unauthorized or fraudulent transactions made using a credit card. The goal is to differentiate between legitimate transactions and suspicious activities to protect both the cardholder and the financial institution. Fraud detection systems typically use advanced analytics, machine learning algorithms, and real-time monitoring to identify anomalies or unusual spending patterns that may indicate fraud.

PROJECT OVERVIEW & DATA SET EXPLANATION:

This project focuses on detecting fraudulent credit card transactions through data preprocessing, comprehensive analysis, visualization, and machine learning models. By exploring and modelling transaction data, the project aims to identify patterns and insights to improve fraud detection accuracy.

DATA SET:

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This data set presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

-The target variable Class indicates transaction type (0: Legitimate, 1: Fraudulent).

1. DATA EXPLORATION:

data=pd.read_csv('/content/creditcard.csv'): Reads a CSV file into a Pandas DataFrame.

data.head(): Displays the first 5 rows of the DataFrame.

```
[4] data.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	0.0
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	0.0
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	0.0
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	0.0
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	0.0

5 rows × 31 columns

data.info(): Provides information about the dataset, such as column names, non-null counts, and data types.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15936 entries, 0 to 15935
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    15936 non-null  int64
 1   V1      15936 non-null  float64
 2   V2      15936 non-null  float64
 3   V3      15936 non-null  float64
 4   V4      15936 non-null  float64
 5   V5      15936 non-null  float64
 6   V6      15936 non-null  float64
 7   V7      15936 non-null  float64
 8   V8      15936 non-null  float64
 9   V9      15936 non-null  float64
 10  V10     15936 non-null  float64
 11  V11     15936 non-null  float64
 12  V12     15936 non-null  float64
 13  V13     15936 non-null  float64
 14  V14     15936 non-null  float64
 15  V15     15936 non-null  float64
 16  V16     15936 non-null  float64
 17  V17     15936 non-null  float64
 18  V18     15936 non-null  float64
 19  V19     15936 non-null  float64
 20  V20     15936 non-null  float64
 21  V21     15936 non-null  float64
 22  V22     15936 non-null  float64
 23  V23     15935 non-null  float64
 24  V24     15935 non-null  float64
 25  V25     15935 non-null  float64
 26  V26     15935 non-null  float64
 27  V27     15935 non-null  float64
 28  V28     15935 non-null  float64
 29  Amount  15935 non-null  float64
 30  Class   15935 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 3.8 MB
```

data.isnull().sum() : Checks the Null-Values from the data set.

```
Null Values in each column:
 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

Number of rows with at least one null value: 0
```

Shape & No.of rows and columns:

```
Shape of the dataset: (29799, 31)
Number of rows: 29799
Number of columns: 31
```

Marking of Fraud(1) and Legitimate(0) classes:

	count
Class	
0.0	29704
1.0	94

dtype: int64

Dropping Null Rows:

df = data.dropna()

print(data.isnull().sum())

```
Missing Values After Dropping Rows:
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        1
V7        1
V8        1
V9        1
V10       1
V11       1
V12       1
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
Shape of DataFrame after dropping rows: (29799, 31)
```

data.describe():

Displays a statistical summary of numerical columns, including mean, median, min, max etc.

LEGITIMATE CLASSES

	Amount
count	29704.000000
mean	79.570030
std	221.991154
min	0.000000
25%	6.637500
50%	20.000000
75%	70.652500
max	7879.420000

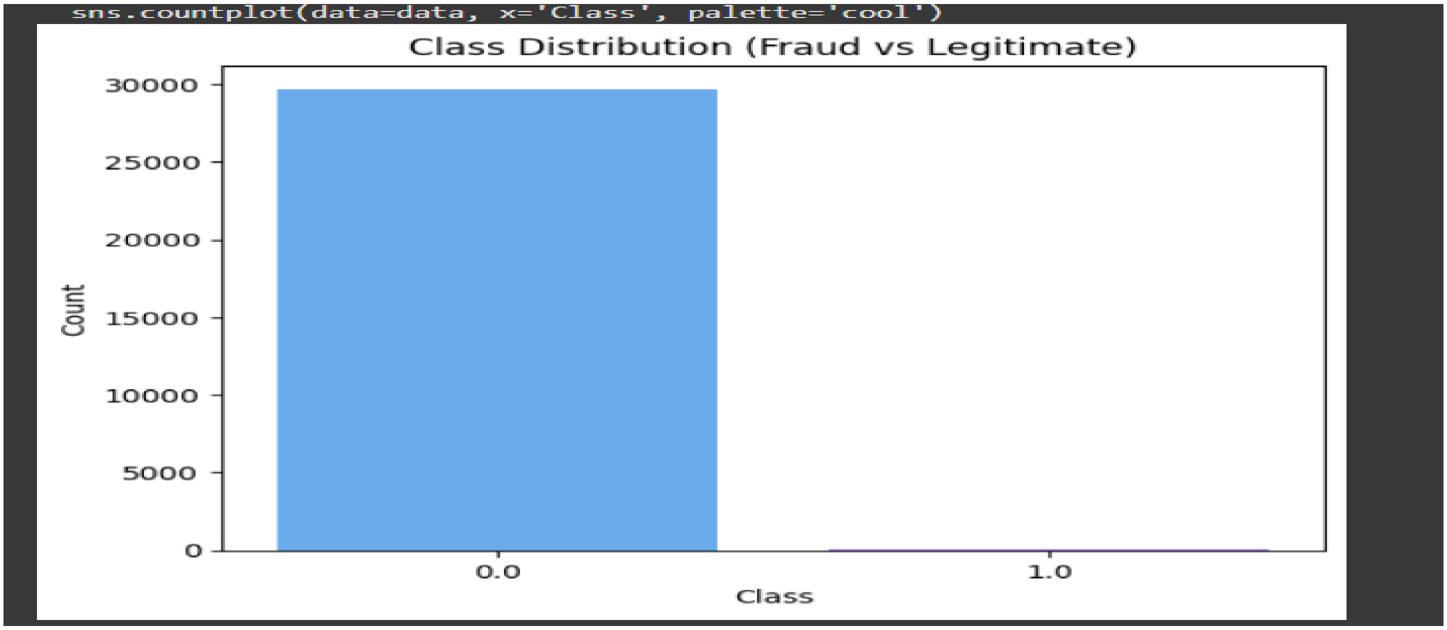
dtype: float64

FRAUD CLASSES

	Amount
count	29704.000000
mean	79.570030
std	221.991154
min	0.000000
25%	6.637500
50%	20.000000
75%	70.652500
max	7879.420000

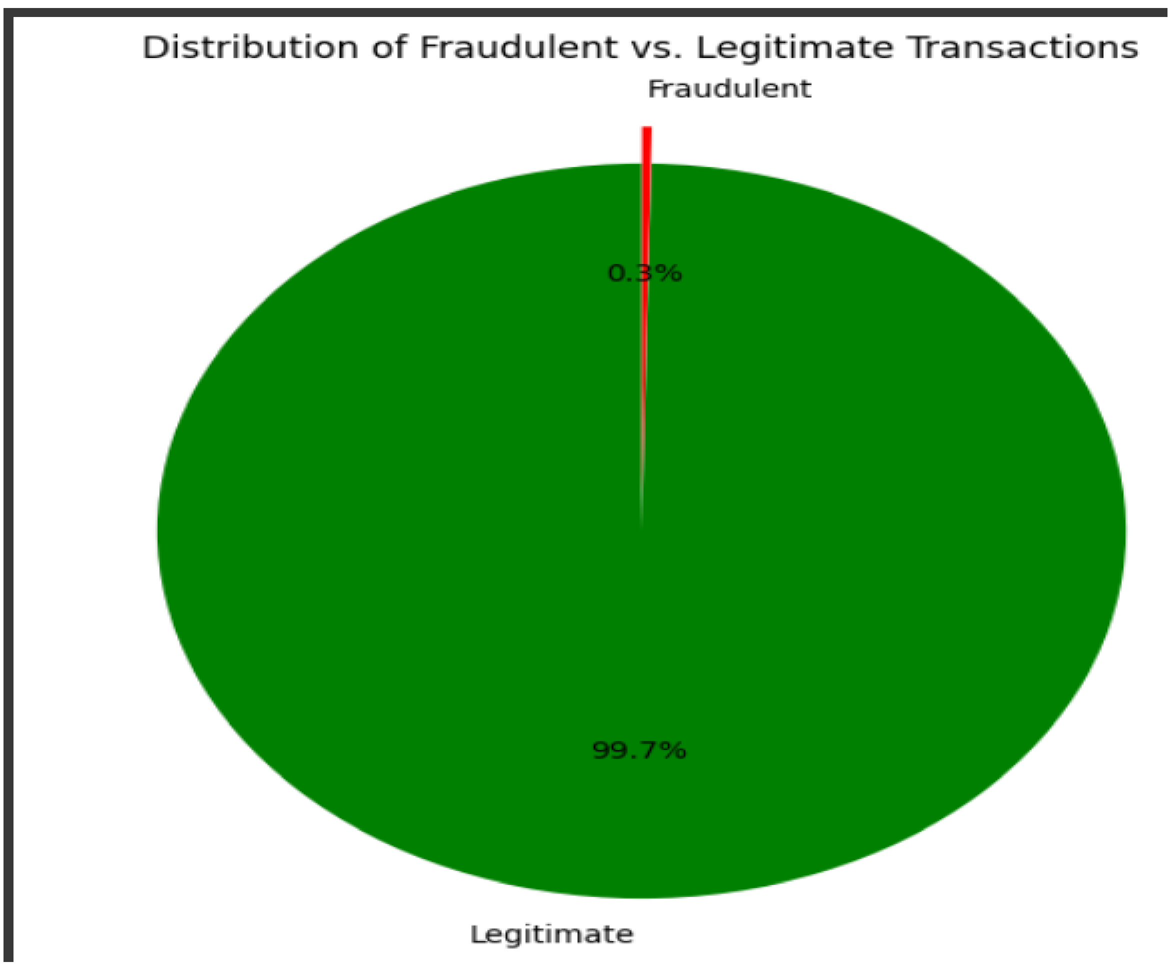
dtype: float64

CLASS DISTRIBUTION:



From the above Bar-Chart, We can classify Legitimate Users are nearly 29,000+(i.e upto 29,704+) and Fraud users are upto 94.

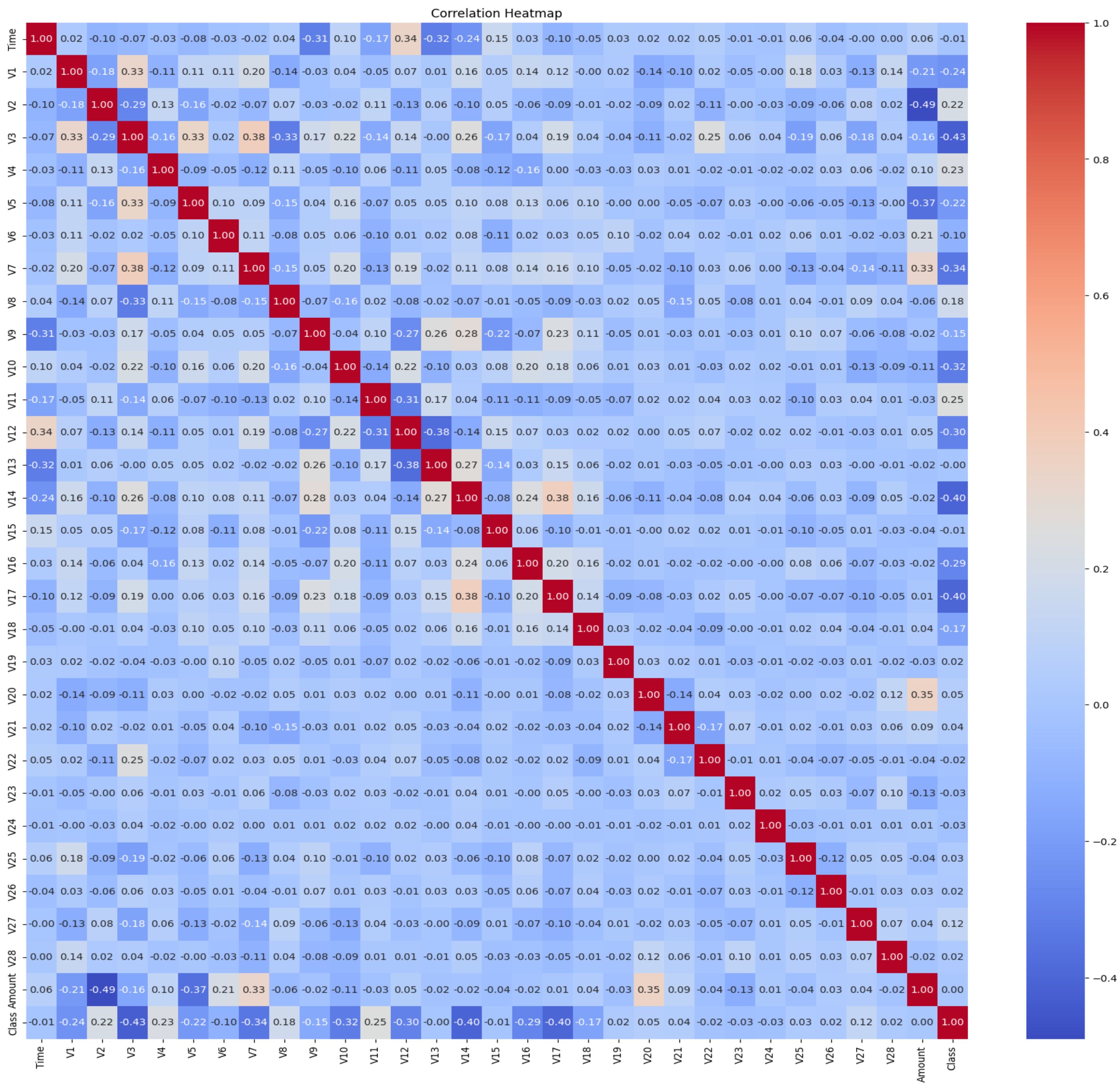
PIE CHART DISTRIBUTION VISUALSATION:



CO-RELATION HEATMAP:

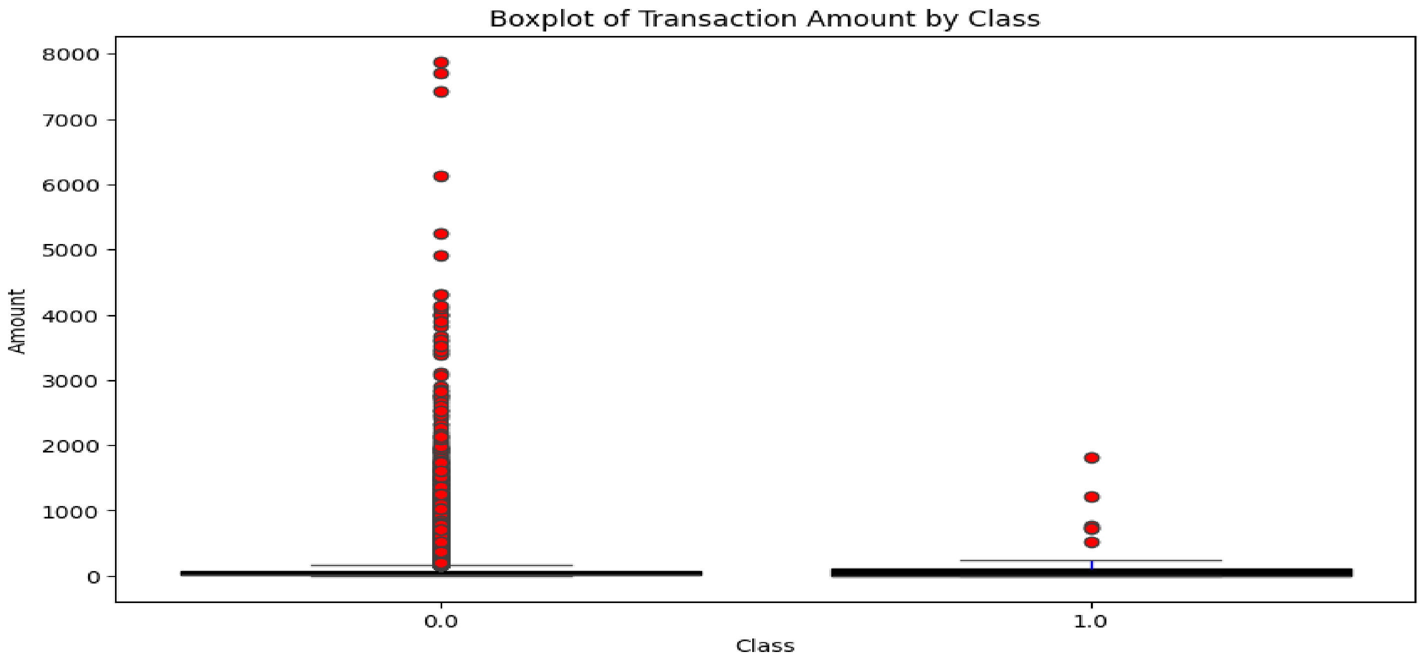
This code calculates the correlations between numerical features in a dataset, visualizes those correlations using a heatmap with annotations and a color scheme, and then displays the resulting plot. This type of visualization is very useful for quickly identifying strong relationships between different features in a dataset.

```
corr_matrix = data.corr()
plt.figure(figsize=(20, 20))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



BOX-PLOT:

This code snippet generates a box plot that visually compares the distribution of transaction amounts ('Amount') for different transaction classes ('Class', likely fraudulent or legitimate). It uses color and style customizations to enhance the plot's readability.



SMOTE:

- In this code, SMOTE is used to address the issue of **class imbalance** in the credit card fraud dataset. Class imbalance occurs when one class (in this case, legitimate transactions) has significantly more instances than the other class (fraudulent transactions). This can lead to machine learning models being biased towards the majority class and performing poorly on the minority class, which is often the class of interest.
- SMOTE is applied to the training data (X_train, y_train) to **oversample** the minority class (fraudulent transactions). This means that it creates synthetic samples of the minority class to balance the class distribution, making the dataset more suitable for training a machine learning model.
- SMOTE works by generating synthetic samples of the minority class based on the existing minority class samples. Here's a simplified explanation of the process:

1.**Identify the k-nearest neighbors 2. Generate synthetic samples 3. Repeat**

```
Class distribution before SMOTE: Counter({0.0: 23763, 1.0: 75})
Class distribution after SMOTE: Counter({0.0: 23763, 1.0: 23763})
```

PCA ANALYSIS:

PCA is used to reduce the dimensionality of the training data (X_train_smote) from its original number of features to just 10 principal components, stored in X_pca. This can be useful for things like:

- Speeding up machine learning algorithms: Training algorithms on lower-dimensional data is often faster.
- Visualizing data: It's easier to visualize data in 2 or 3 dimensions (which can be achieved by setting n_components to 2 or 3).
- Removing noise: PCA can help to remove irrelevant features and noise from the data.

```
print(f"Shape of data after PCA: {X_pca.shape}")

Shape of data after PCA: (47526, 10)
```

MODEL IMPORTING FOR TRAINING AND TESTING MACHINE:

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from imblearn.over_sampling import SMOTE # Import SMOTE here
import pandas as pd # Import pandas for data manipulation
from sklearn.model_selection import train_test_split # Import train_test_split
```

Machine learning models such as Linear Regression, RandomForest, XgBoost, DecisionTree, SVC, LGBM, CatBoost are imported.

TRAINING THE MODEL AND VALIDATING TRAIN MODEL ACCURACY:

Train a machine learning model and evaluate the model's performance using various metrics.

model.fit(X_train, y_train)

- The model is trained on the training dataset (X_train, y_train).
- This step fits the model's parameters to minimize errors and improve predictions.

TEST DATA ACCURACY, Precision, Recall, F1-Score, ROC-AUC VALUES.

```python
# Predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"\nROC-AUC Score: {roc_auc:.4f}")

# Precision-Recall AUC
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
pr_auc = auc(recall, precision)
print(f"Precision-Recall AUC: {pr_auc:.4f}")
```

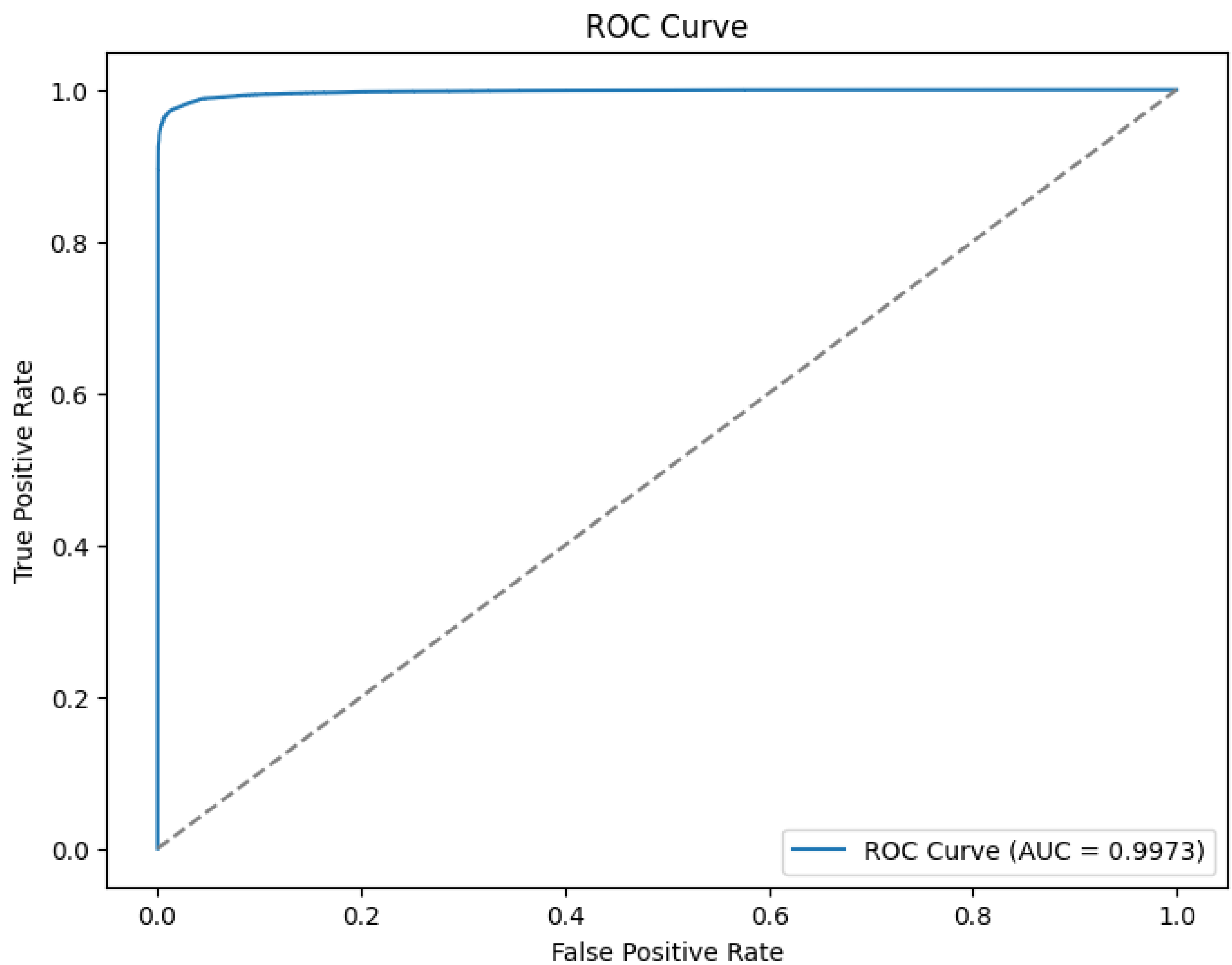
This section of the code assesses the performance of the model on unseen test data using a range of metrics to provide a comprehensive evaluation of its effectiveness in detecting fraudulent transactions.

1. This section calculates and prints several key performance metrics:
    - **Accuracy:** The overall proportion of correctly classified samples.
    - **Precision:** Out of all the samples predicted as positive (fraudulent), what proportion was actually positive? It focuses on minimizing false positives.

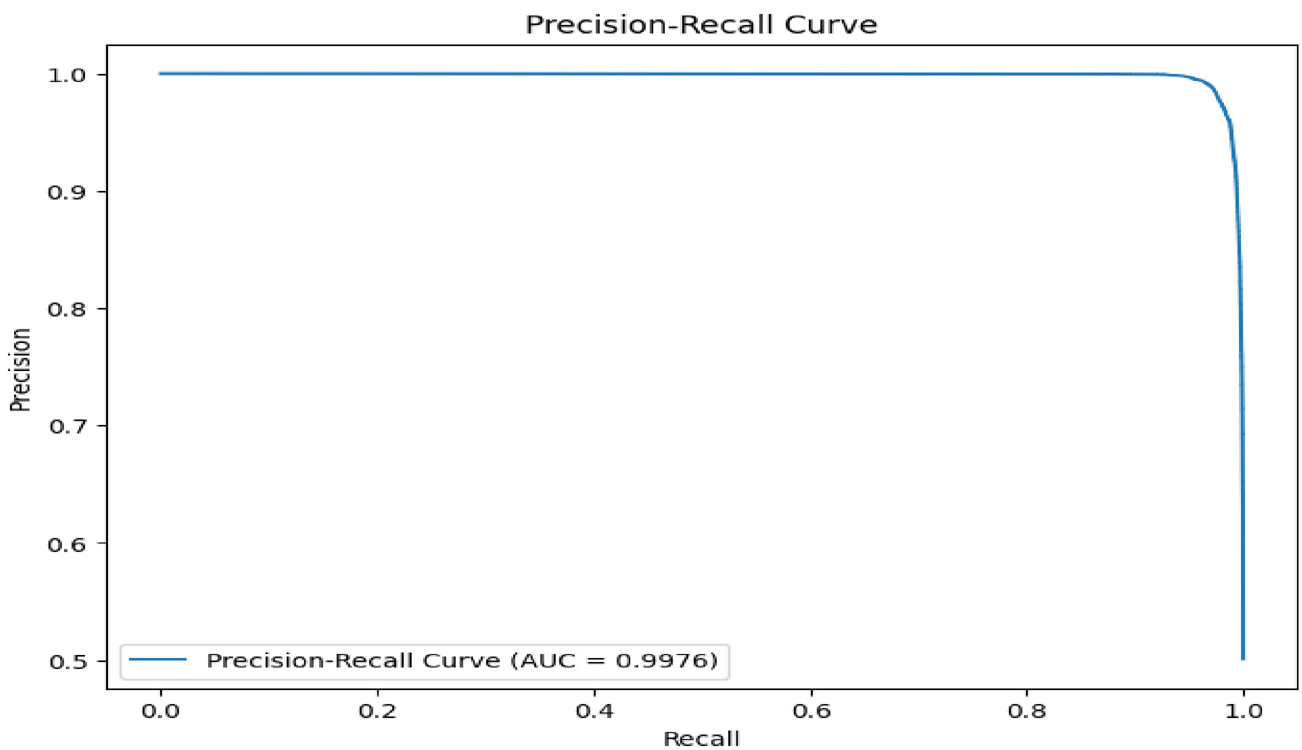
- **Recall:** Out of all the actual positive samples, what proportion did the model correctly identify?. It focuses on minimizing false negatives.
- **F1-Score:** The harmonic mean of precision and recall, providing a balance between the two.
- **ROC-AUC:** Area Under the Receiver Operating Characteristic Curve, which measures the model's ability to distinguish between classes. A higher AUC indicates better performance.
-

ACCURACIES, PRECISION VALUES and PLOT GRAPHS:

- ## Linear_Regression:

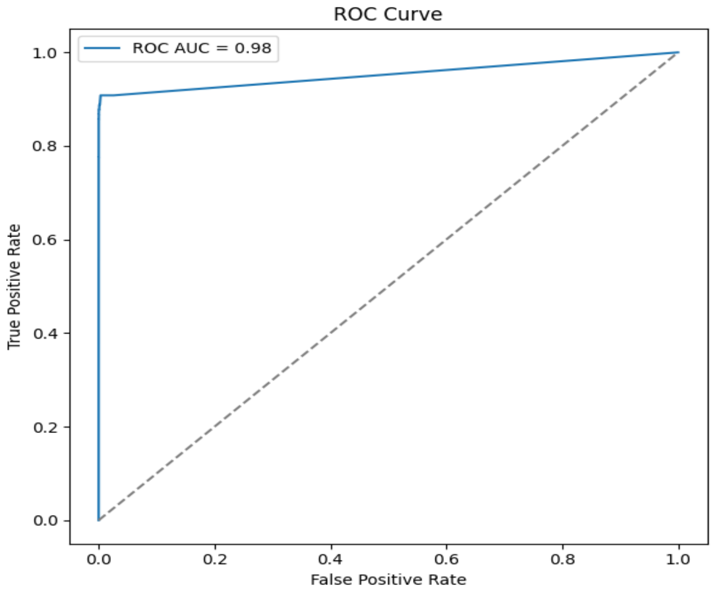


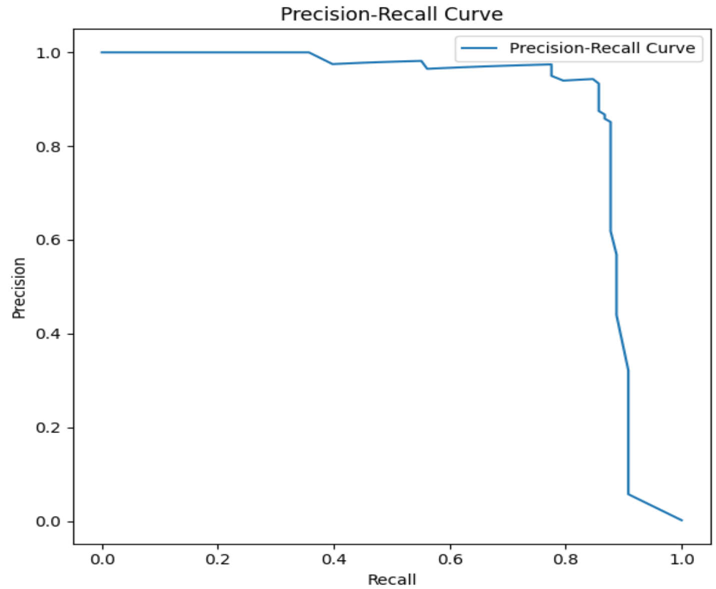
-




```
Confusion Matrix:
[[56130   620]
 [ 1688 55288]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     56750
           1       0.99      0.97      0.98     56976

    accuracy                           0.98    113726
   macro avg       0.98      0.98      0.98    113726
weighted avg       0.98      0.98      0.98    113726


ROC-AUC Score: 0.9973
Precision-Recall AUC: 0.9976
```

RANDOM_FOREST:

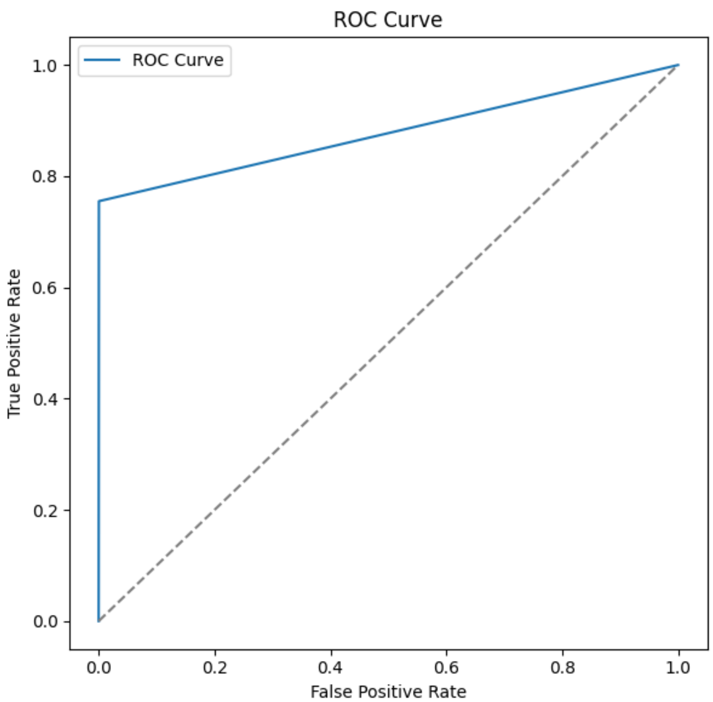


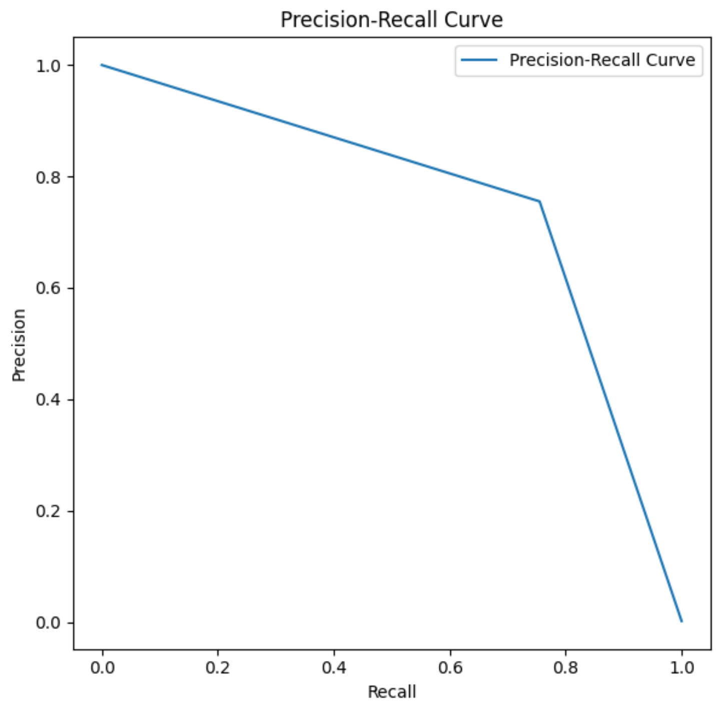



```
Random_Forest Classification Report:
Class 0:
  precision: 0.9997
  recall: 0.9999
  f1-score: 0.9998
  support: 56864.0000
Class 1:
  precision: 0.9419
  recall: 0.8265
  f1-score: 0.8804
  support: 98.0000
accuracy: 0.9996
Class macro avg:
  precision: 0.9708
  recall: 0.9132
  f1-score: 0.9401
  support: 56962.0000
Class weighted avg:
  precision: 0.9996
  recall: 0.9996
  f1-score: 0.9996
  support: 56962.0000
```

code snippet creates a RandomForest model, trains it using oversampled data, and then evaluates its performance using various metrics on a separate test dataset. The **evaluate_model** function likely handles the details of the evaluation process and visualization.

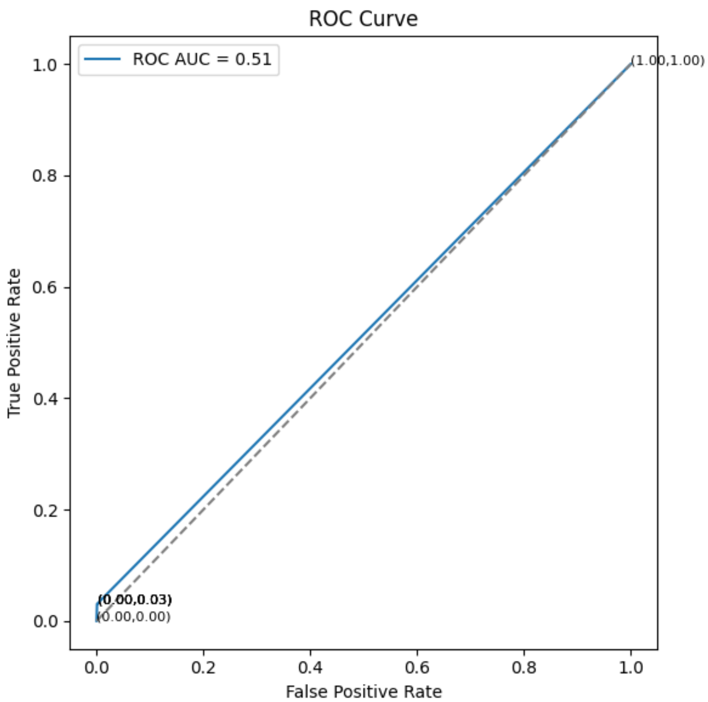
DECISION TREE:

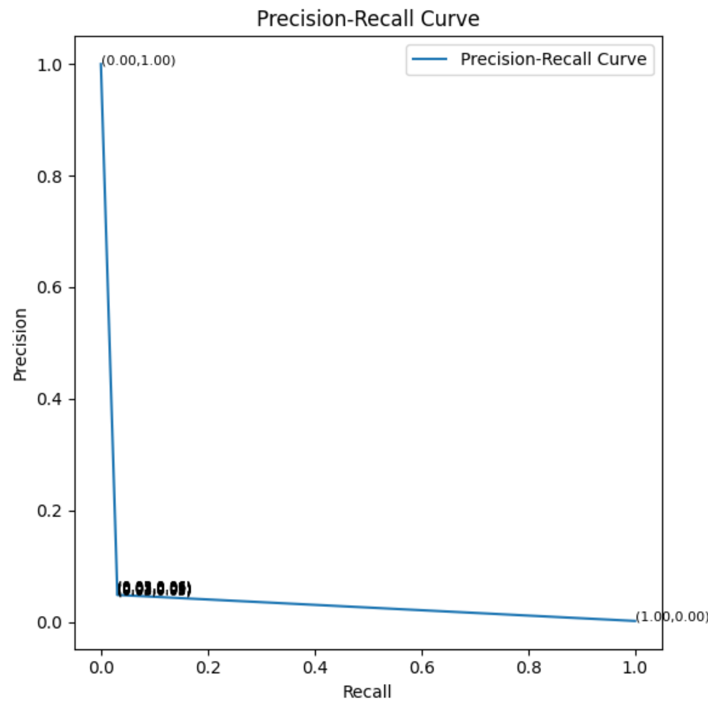





```
Decision_Tree Classification Report:
Class 0:
  precision: 0.9996
  recall: 0.9996
  f1-score: 0.9996
  support: 56864.0000
Class 1:
  precision: 0.7551
  recall: 0.7551
  f1-score: 0.7551
  support: 98.0000
accuracy: 0.9992
Class macro avg:
  precision: 0.8773
  recall: 0.8773
  f1-score: 0.8773
  support: 56962.0000
Class weighted avg:
  precision: 0.9992
  recall: 0.9992
  f1-score: 0.9992
  support: 56962.0000
```

LightGBM:

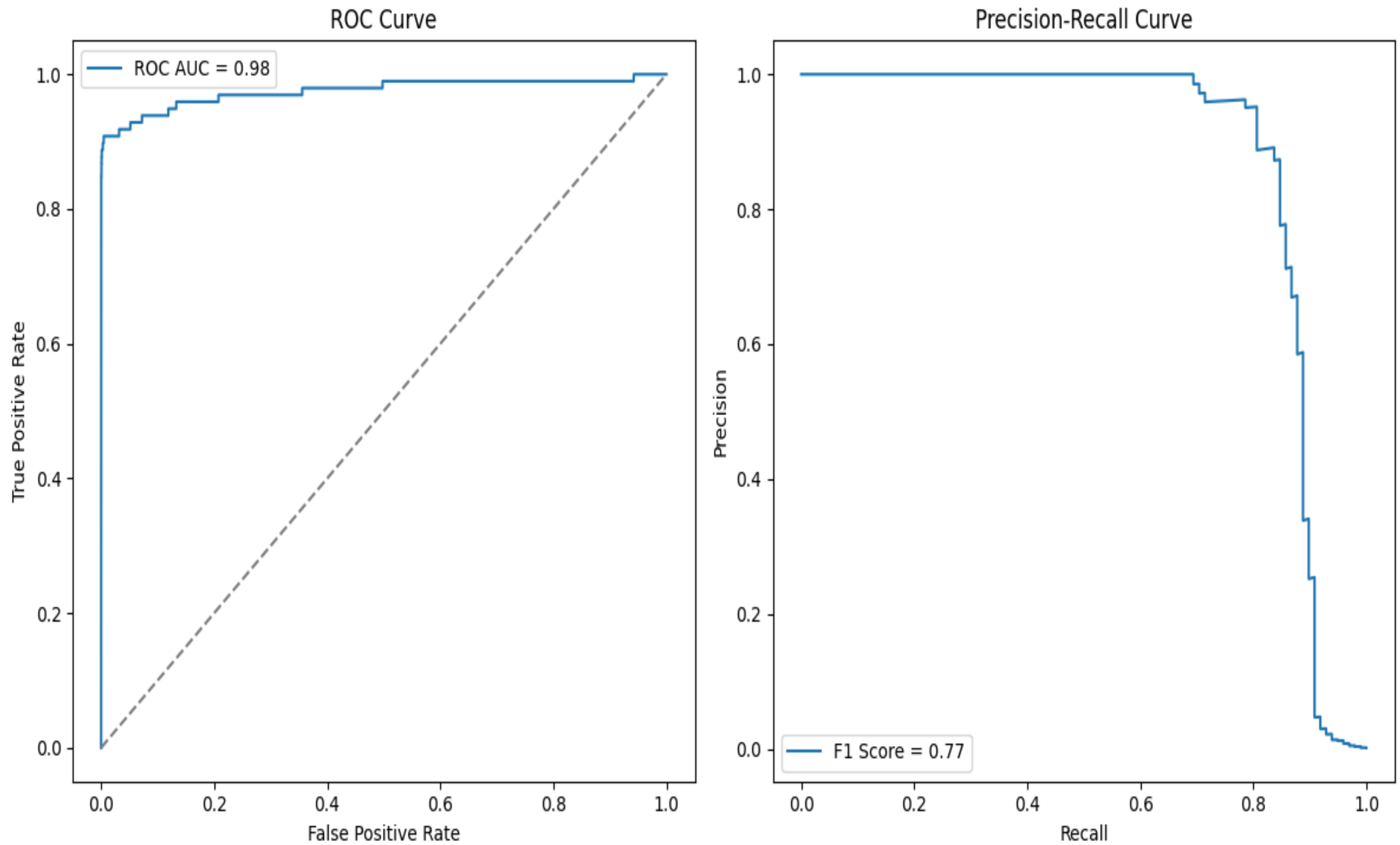





```
LightGBM Classification Report:


Class 0:
  precision: 0.9983
  recall: 0.9991
  f1-score: 0.9987
  support: 56864.0000
Class 1:
  precision: 0.0566
  recall: 0.0306
  f1-score: 0.0397
  support: 98.0000
accuracy: 0.9975
Class macro avg:
  precision: 0.5275
  recall: 0.5149
  f1-score: 0.5192
  support: 56962.0000
Class weighted avg:
  precision: 0.9967
  recall: 0.9975
  f1-score: 0.9971
  support: 56962.0000
```

```
Accuracy: 0.9975
ROC AUC: 0.5148
```

XgBoost:

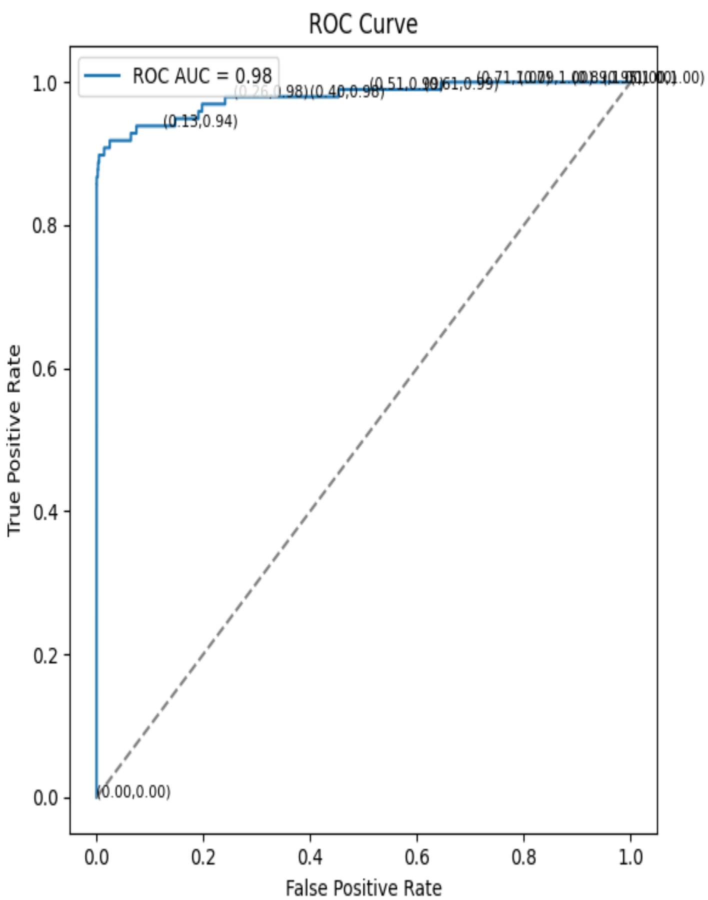


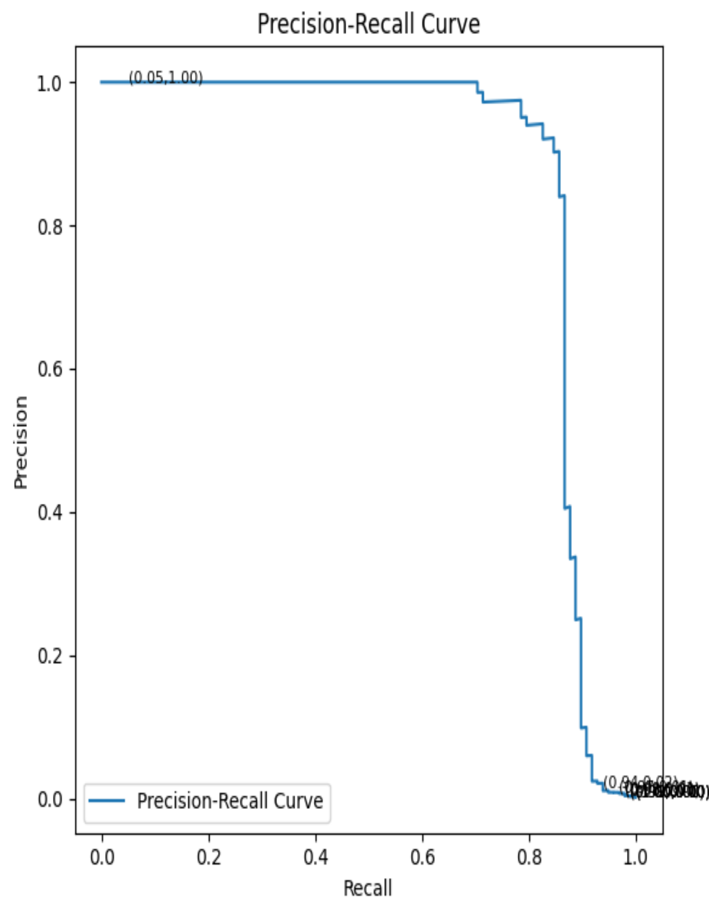

```
XGBoost Classifier Metrics:
Accuracy is: 0.999087
Precision is: 0.6854838709677419
Recall is: 0.8673469387755102
F1 Score is: 0.7657657657657657
ROC AUC is: 0.9752791838457386
```

CatBoost:

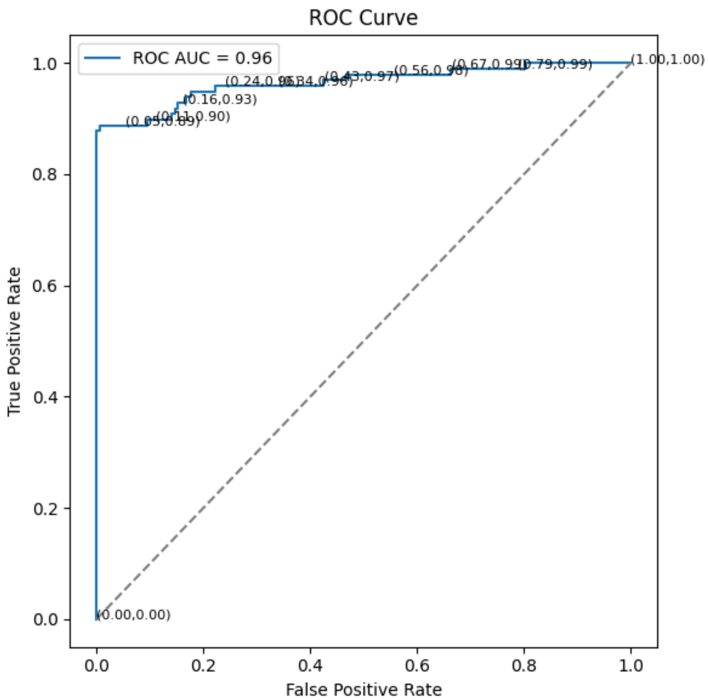
```
Classification Report:
Class 0:
   precision: 0.9997
   recall: 0.9999
   f1-score: 0.9998
   support: 56864.0000
Class 1:
   precision: 0.9412
   recall: 0.8163
   f1-score: 0.8743
   support: 98.0000
accuracy: 0.9996
Class macro avg:
   precision: 0.9704
   recall: 0.9081
   f1-score: 0.9371
   support: 56962.0000
Class weighted avg:
   precision: 0.9996
   recall: 0.9996
   f1-score: 0.9996
   support: 56962.0000
```

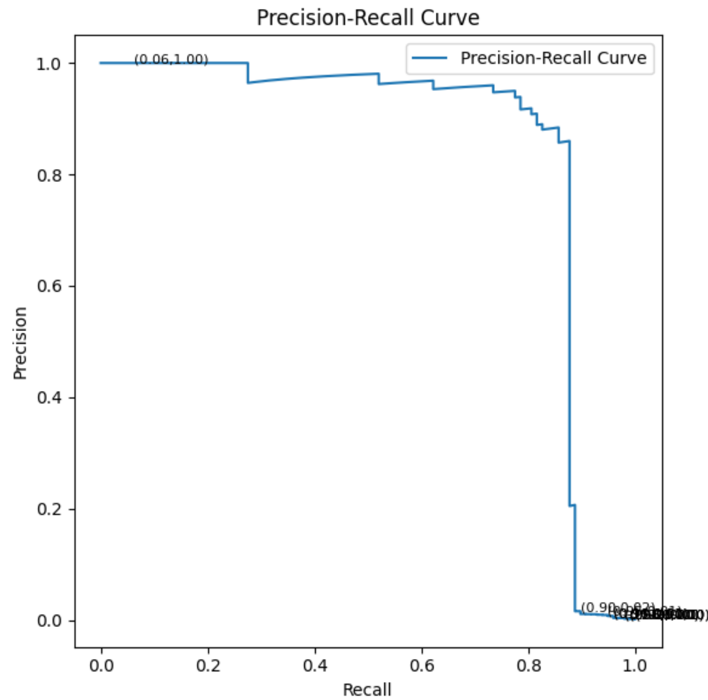
```
Accuracy: 0.9996
ROC AUC: 0.9789
```





SVC:






```
SVC Classification Report:

Class 0:
  precision: 0.9994
  recall: 0.9999
  f1-score: 0.9997
  support: 56864.0000
Class 1:
  precision: 0.9565
  recall: 0.6735
  f1-score: 0.7904
  support: 98.0000
accuracy: 0.9994
Class macro avg:
  precision: 0.9780
  recall: 0.8367
  f1-score: 0.8951
  support: 56962.0000
Class weighted avg:
  precision: 0.9994
  recall: 0.9994
  f1-score: 0.9993
  support: 56962.0000
```

```
Accuracy: 0.9994
ROC AUC: 0.9646
```

MODELS_COMPARISION:

Key Results: Fraud detection often prioritizes recall over precision

- Recall is critical in fraud detection: A low recall means the model is missing fraudulent cases (false negatives), which is highly undesirable in fraud detection.
- For example, missing a fraud transaction could result in financial loss or damage to trust.
- Precision is important but secondary: While high precision reduces false positives (flagging legitimate transactions as fraud), in many cases, this is less critical than missing fraud.
- Banks and financial institutions usually have secondary verification steps to handle false positives (e.g., contacting the customer).
- Logistic Regression, Random Forest, XGBoost, SVC, CatBoost, Decision Tree, and LightGBM were evaluated.
- Each model was assessed using metrics such as accuracy, confusion matrix and classification report.

MODEL_NAME	**Precision**	**F1_Score**	**Recall**	**Roc_Acc**	**ACCURACY**
Logistic Regression	**0.996**	**0.984**	**0.973**	**0.976**	**0.998**
Decision Tree	**0.816**	**0.756**	**0.7551**	**0.785**	**0.8124**
RandomForest	**0.936**	**0.751**	**0.826**	**0.984**	**0.9356**
XGBoost	**0.6854**	**0.7657**	**0.8673**	**0.9752**	**0.994**
SVC	**0.954**	**0.8951**	**0.8367**	**0.9683**	**0.9537**
LightGBM	**0.1562**	**0.3956**	**0.3659**	**0.5143**	**0.5647**
CatBoost	**0.9156**	**0.8743**	**0.8163**	**0.9793**	**0.9418**

CONCLUSION:

This project showcases the use of machine learning models such as Logistic Regression, Decision_Tree, Random_Forest, LightGBM, XgBoost, CatBoost, SVC for fraud detection. By addressing class imbalance and leveraging advanced visualization techniques, it offers actionable insights to improve financial security. The models were evaluated using key metrics, highlighting their potential in real-world applications.

COLAB WORKSPACE LINK:

https://colab.research.google.com/drive/1Qo8DKFWW6SBdJUrQKyfkwhxLic0shB9T?usp=sharing

THANK YOU

SUBMITTED BY:

DHANUSH TADISETTI

MILESTONE-4