Project Title : Advanced Credit Card Fraud Detection Using Machine Learning

Problem Statement :

Credit card fraud is a major concern in financial systems, with fraudulent transactions often being rare, hidden within large volumes of legitimate ones. The goal of this project is to develop a machine learning pipeline that can accurately detect fraudulent credit card transactions in real-time. Given the severe class imbalance, traditional models struggle with identifying fraud without a high false positive rate.

Objectives

- Analyze transaction patterns to detect anomalies using EDA.
- Handle imbalanced data using techniques like SMOTE.
- Build and compare classification models such as Random Forest and XGBoost.
- Evaluate model performance using ROC-AUC, precision-recall curves, and confusion matrix.
- Explain model predictions using SHAP values for transparency.
- Visualize high-dimensional data using t-SNE to detect possible fraud clusters.

Dataset Description

- Source: Public dataset from Kaggle
- Rows: 284,807
- Columns: 31 (including 28 anonymized PCA components V1 to V28, Time, Amount, and Class)
- Target Variable: Class (0 = Non-Fraud, 1 = Fraud)
- Imbalance: Only ~0.17% transactions are fraudulent

```python
# Import the required Library
import numpy as np
import pandas as pd

# Import Data Visualization Libray
import matplotlib.pyplot as plt
import seaborn as sns

# Import Library for Spliting Traing and testing dataset
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Import library for Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Import library for XGB Classifier
from xgboost import XGBClassifier
```

```
# Import library for Logistic Regression
from sklearn.linear_model import LogisticRegression

# Import library for metrices
from sklearn.metrics import (confusion_matrix, roc_curve,
precision_recall_curve,
    accuracy_score, f1_score, mean_squared_error, mean_absolute_error,
r2_score)


import shap
from sklearn.manifold import TSNE


data = pd.read_csv('Creditcard.csv')

data.head()

   Time        V1        V2        V3        V4        V5        V6
V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -
0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
0.592941

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
```

```
[5 rows x 31 columns]
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
data.describe()
```

```
                Time            V1            V2            V3  
V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  
2.848070e+05
```

```
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15
2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00
1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -
5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -
8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -
1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00
7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00
1.687534e+01

                 V5            V6            V7            V8
V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -
2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01
1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16
88.349619
```

```
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

```
data.duplicated()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
284802    False
284803    False
284804    False
284805    False
284806    False
Length: 284807, dtype: bool
```

```
data.isnull().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
```

```
# Convert the Numeric dataset into standard form
col = data[['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']]
# Convert columns to numeric
for column in col.columns:
    col[column] = pd.to_numeric(col[column], errors='coerce')

# Information about features of class 1 (Fraud)

data[data['Class'] == 1].describe()
```

```
                Time          V1          V2          V3
V4  \
count     492.000000  492.000000  492.000000  492.000000  492.000000

mean    80746.806911   -4.771948    3.623778   -7.033281    4.542029

std     47835.365138    6.783687    4.291216    7.110937    2.873318

min       406.000000  -30.552380   -8.402154  -31.103685   -1.313275
```

```
25%     41241.500000   -6.036063    1.188226   -8.643489    2.373050
50%     75568.500000   -2.342497    2.717869   -5.075257    4.177147
75%    128483.000000   -0.419200    4.971257   -2.276185    6.348729
max    170348.000000    2.132386   22.057729    2.250210   12.114672

               V5          V6          V7          V8          V9  ...
\
count  492.000000  492.000000  492.000000  492.000000  492.000000  ...
mean    -3.151225   -1.397737   -5.568731    0.570636   -2.581123  ...
std      5.372468    1.858124    7.206773    6.797831    2.500896  ...
min    -22.105532   -6.406267  -43.557242  -41.044261  -13.434066  ...
25%     -4.792835   -2.501511   -7.965295   -0.195336   -3.872383  ...
50%     -1.522962   -1.424616   -3.034402    0.621508   -2.208768  ...
75%      0.214562   -0.413216   -0.945954    1.764879   -0.787850  ...
max     11.095089    6.474115    5.802537   20.007208    3.353525  ...

              V21         V22         V23         V24         V25
V26  \
count  492.000000  492.000000  492.000000  492.000000  492.000000
492.000000
mean     0.713588    0.014049   -0.040308   -0.105130    0.041449
0.051648
std      3.869304    1.494602    1.579642    0.515577    0.797205
0.471679
min    -22.797604   -8.887017  -19.254328   -2.028024   -4.781606   -
1.152671
25%      0.041787   -0.533764   -0.342175   -0.436809   -0.314348   -
0.259416
50%      0.592146    0.048434   -0.073135   -0.060795    0.088371
0.004321
75%      1.244611    0.617474    0.308378    0.285328    0.456515
0.396733
max     27.202839    8.361985    5.466230    1.091435    2.208209
2.745261

              V27         V28      Amount  Class
count  492.000000  492.000000  492.000000  492.0
mean     0.170575    0.075667  122.211321    1.0
```

```
std       1.376766    0.547291   256.683288    0.0
min      -7.263482   -1.869290     0.000000    1.0
25%      -0.020025   -0.108868     1.000000    1.0
50%       0.394926    0.146344     9.250000    1.0
75%       0.826029    0.381152   105.890000    1.0
max       3.052358    1.779364  2125.870000    1.0

[8 rows x 31 columns]
```

```
# Information about features of class 0 (Not Fraud)
data[data['Class'] == 0].describe()
```

```
                Time             V1             V2             V3  \
count  284315.000000  284315.000000  284315.000000  284315.000000
mean    94838.202258       0.008258      -0.006271       0.012171
std     47484.015786       1.929814       1.636146       1.459429
min         0.000000     -56.407510     -72.715728     -48.325589
25%     54230.000000      -0.917544      -0.599473      -0.884541
50%     84711.000000       0.020023       0.064070       0.182158
75%    139333.000000       1.316218       0.800446       1.028372
max    172792.000000       2.454930      18.902453       9.382558

                  V4             V5             V6             V7  \
count  284315.000000  284315.000000  284315.000000  284315.000000
mean       -0.007860       0.005453       0.002419       0.009637
std         1.399333       1.356952       1.329913       1.178812
min        -5.683171    -113.743307     -26.160506     -31.764946
25%        -0.850077      -0.689398      -0.766847      -0.551442
50%        -0.022405      -0.053457      -0.273123       0.041138
75%         0.737624       0.612181       0.399619       0.571019
max        16.875344      34.801666      73.301626     120.589494

                  V8             V9  ...            V21            V22
\
count  284315.000000  284315.000000  ...  284315.000000  284315.000000

mean       -0.000987       0.004467  ...      -0.001235      -0.000024

std         1.161283       1.089372  ...       0.716743       0.723668

min       -73.216718      -6.290730  ...     -34.830382     -10.933144

25%        -0.208633      -0.640412  ...      -0.228509      -0.542403

50%         0.022041      -0.049964  ...      -0.029821       0.006736

75%         0.326200       0.598230  ...       0.185626       0.528407

max        18.709255      15.594995  ...      22.614889      10.503090
```

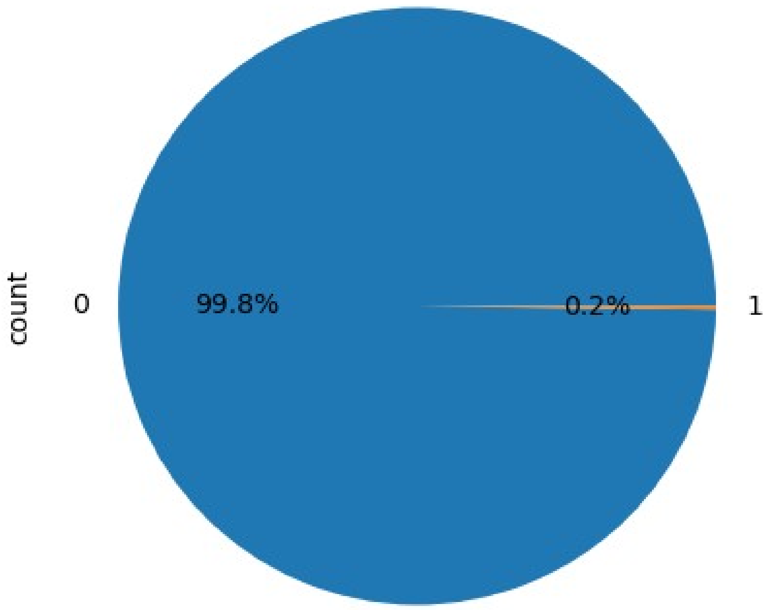
```
                 V23            V24            V25            V26  \
count  284315.000000  284315.000000  284315.000000  284315.000000
mean        0.000070       0.000182      -0.000072      -0.000089
std         0.621541       0.605776       0.520673       0.482241
min       -44.807735      -2.836627     -10.295397      -2.604551
25%        -0.161702      -0.354425      -0.317145      -0.327074
50%        -0.011147       0.041082       0.016417      -0.052227
75%         0.147522       0.439869       0.350594       0.240671
max        22.528412       4.584549       7.519589       3.517346

                 V27            V28         Amount     Class
count  284315.000000  284315.000000  284315.000000  284315.0
mean       -0.000295      -0.000131      88.291022       0.0
std         0.399847       0.329570     250.105092       0.0
min       -22.565679     -15.430084       0.000000       0.0
25%        -0.070852      -0.052950       5.650000       0.0
50%         0.001230       0.011199      22.000000       0.0
75%         0.090573       0.077962      77.050000       0.0
max        31.612198      33.847808   25691.160000       0.0

[8 rows x 31 columns]
```

```python
# Count the number of froud and not froud

data.Class.value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

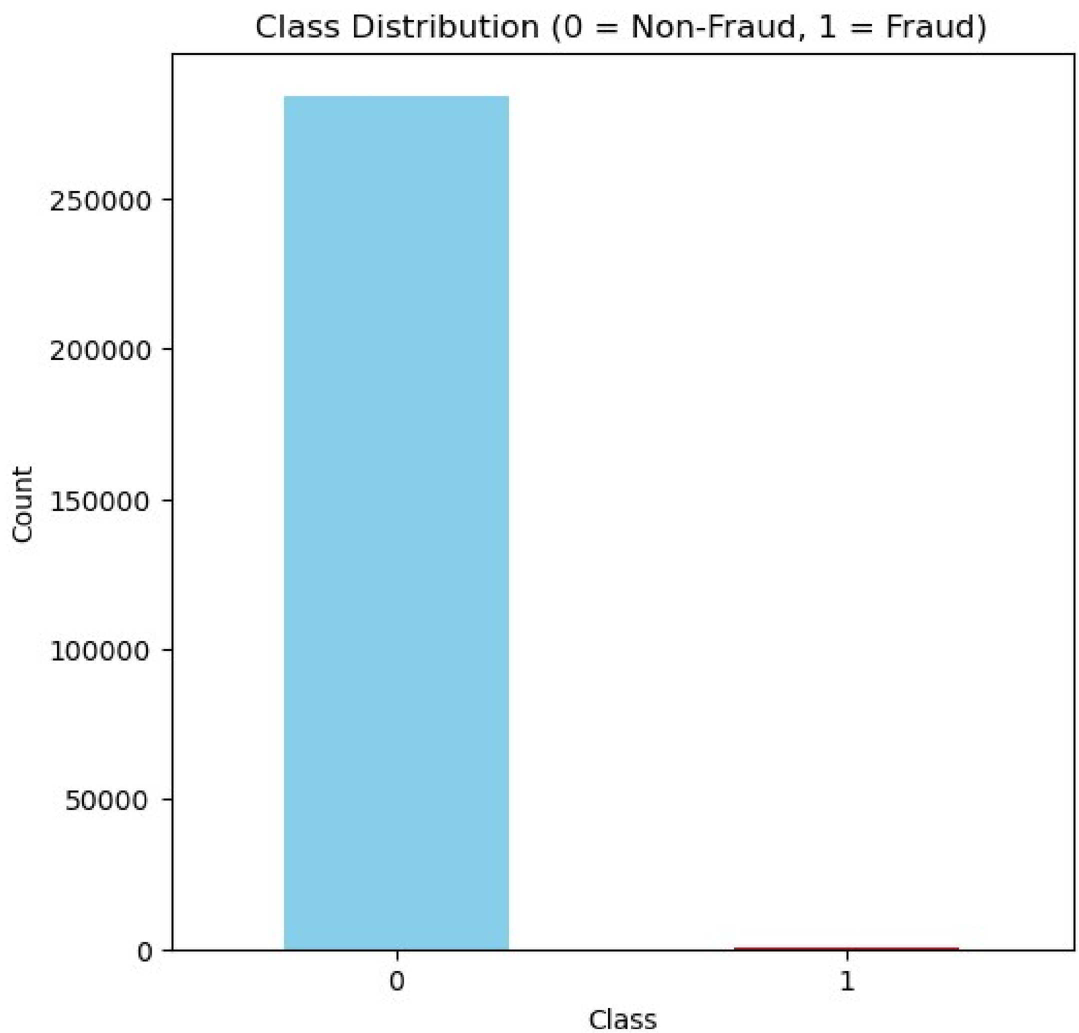
```python
data['Class'].value_counts().plot.pie(autopct = '%.1f%%', figsize =
(5,5) )
```

```
<Axes: ylabel='count'>
```

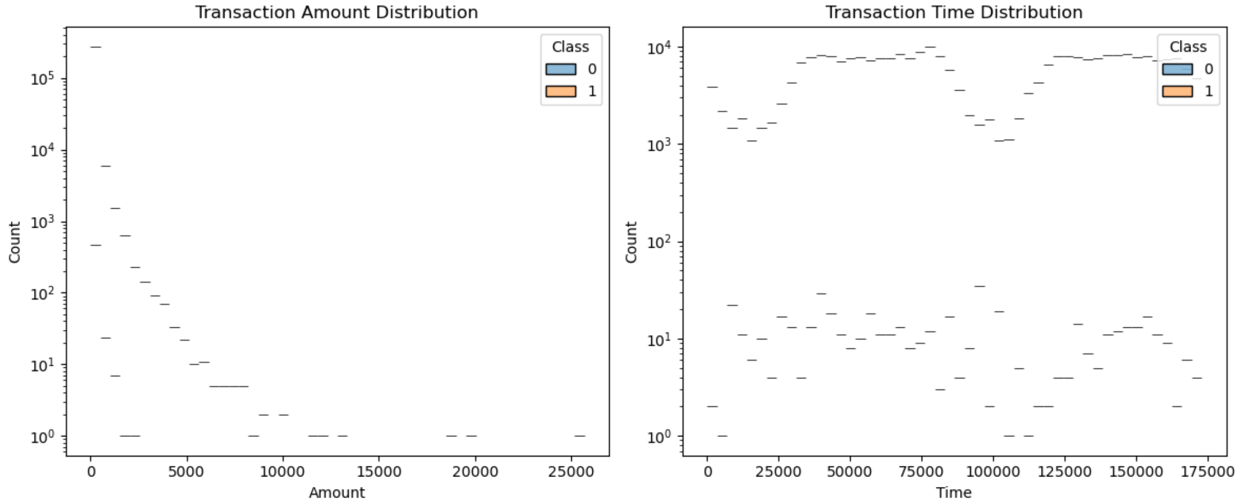


Data Visualization

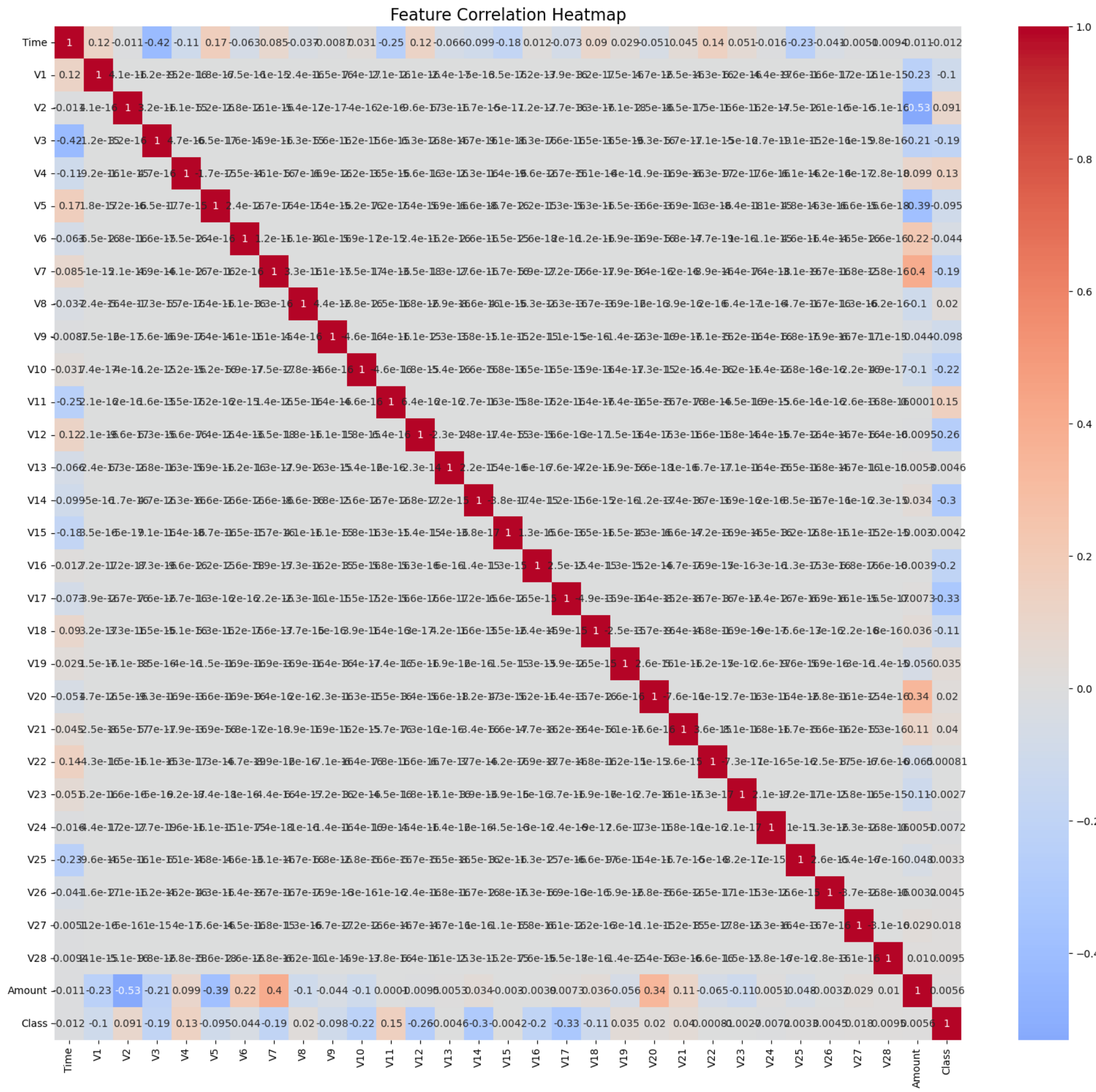
```
# Plot 1: Class imbalance
plt.figure(figsize=(6,6))
data['Class'].value_counts().plot(kind='bar', color=['skyblue',
'red'])
plt.title("Class Distribution (0 = Non-Fraud, 1 = Fraud)")
plt.xlabel("Class")
plt.ylabel("Count")
plt.xticks(rotation=0)
plt.show()
```



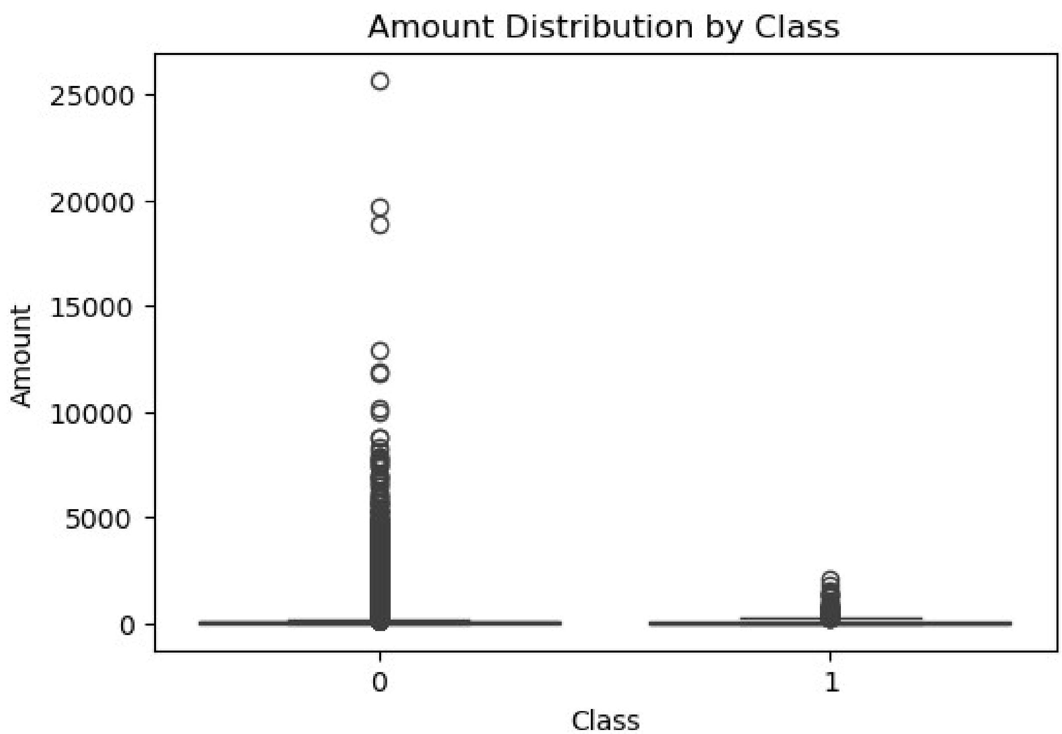

```
# Plot 2: Histograms of Amount and Time by class
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(data=data, x='Amount', hue='Class', bins=50,
log_scale=(False, True))
plt.title('Transaction Amount Distribution')

plt.subplot(1,2,2)
sns.histplot(data=data, x='Time', hue='Class', bins=50,
log_scale=(False, True))
plt.title('Transaction Time Distribution')
plt.tight_layout()
plt.show()
```




```python
# Plot 3: Correlation heatmap
plt.figure(figsize=(20,18))
sns.heatmap(data.corr(), annot = True , cmap='coolwarm', center=0)
plt.title('Feature Correlation Heatmap' , fontsize = 16)
plt.show()
```




```python
# Plot 4: Boxplots of Amount by class
plt.figure(figsize=(6,4))
sns.boxplot(x='Class', y='Amount', data=data)
plt.title('Amount Distribution by Class')
plt.show()
```



Model Bulding

```
# Befor model builing split the dataset for  training and testing
purpose
from  sklearn.model_selection import train_test_split

# Asign the alue for X and y for training and testing

X = data.drop(['Class'] , axis = 1)
y = data['Class']

print(X.shape)
print(y.shape)

(284807, 30)
(284807,)

# Split the dataset into training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)
```

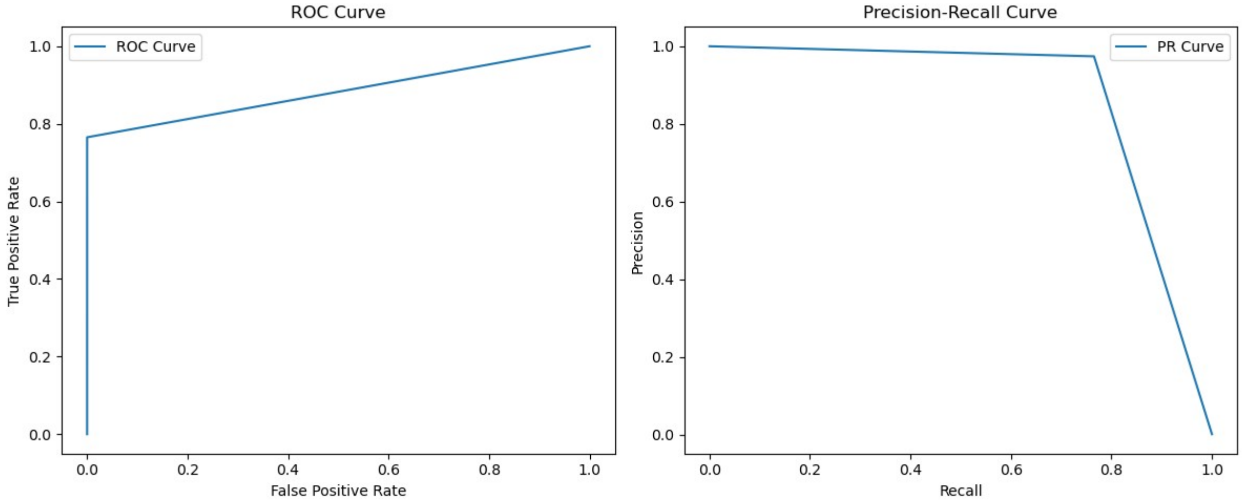
Random Forest Classifier Model

```
# Initialize and fit
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

RandomForestClassifier(random_state=42)
```

```python
# Predict the value for y

y_pred_rf = rf.predict(X_test)

# Evaluate the model
mae_rf = n=mean_absolute_error(y_test,y_pred_rf)
r2_rf = r2_score(y_test,y_pred_rf)
mse_rf = mean_squared_error(y_test,y_pred_rf)
acc_rf = accuracy_score(y_test,y_pred_rf)
f1_rf = f1_score(y_test,y_pred_rf)
print('Random Forest Model Performance :')
print('Mean Absolute Error :' , mae_rf)
print('R^2 Score :' , r2_rf)
print("Mean Squared Error :" ,mse_rf )
print("Accuracy Score :" ,acc_rf)
print("F 1 Score :" , f1_rf)
```

```
Random Forest Model Performance :
Mean Absolute Error : 0.00043888908395070395
R^2 Score : 0.7444583137137805
Mean Squared Error : 0.00043888908395070395
Accuracy Score : 0.9995611109160493
F 1 Score : 0.8571428571428571
```

```python
# ROC and Precision-Recall Curves
fpr, tpr, _ = roc_curve(y_test, y_pred_rf)
precision, recall, _ = precision_recall_curve(y_test, y_pred_rf)

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(fpr, tpr, label='ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()

plt.subplot(1,2,2)
plt.plot(recall, precision, label='PR Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.tight_layout()
plt.show()
```



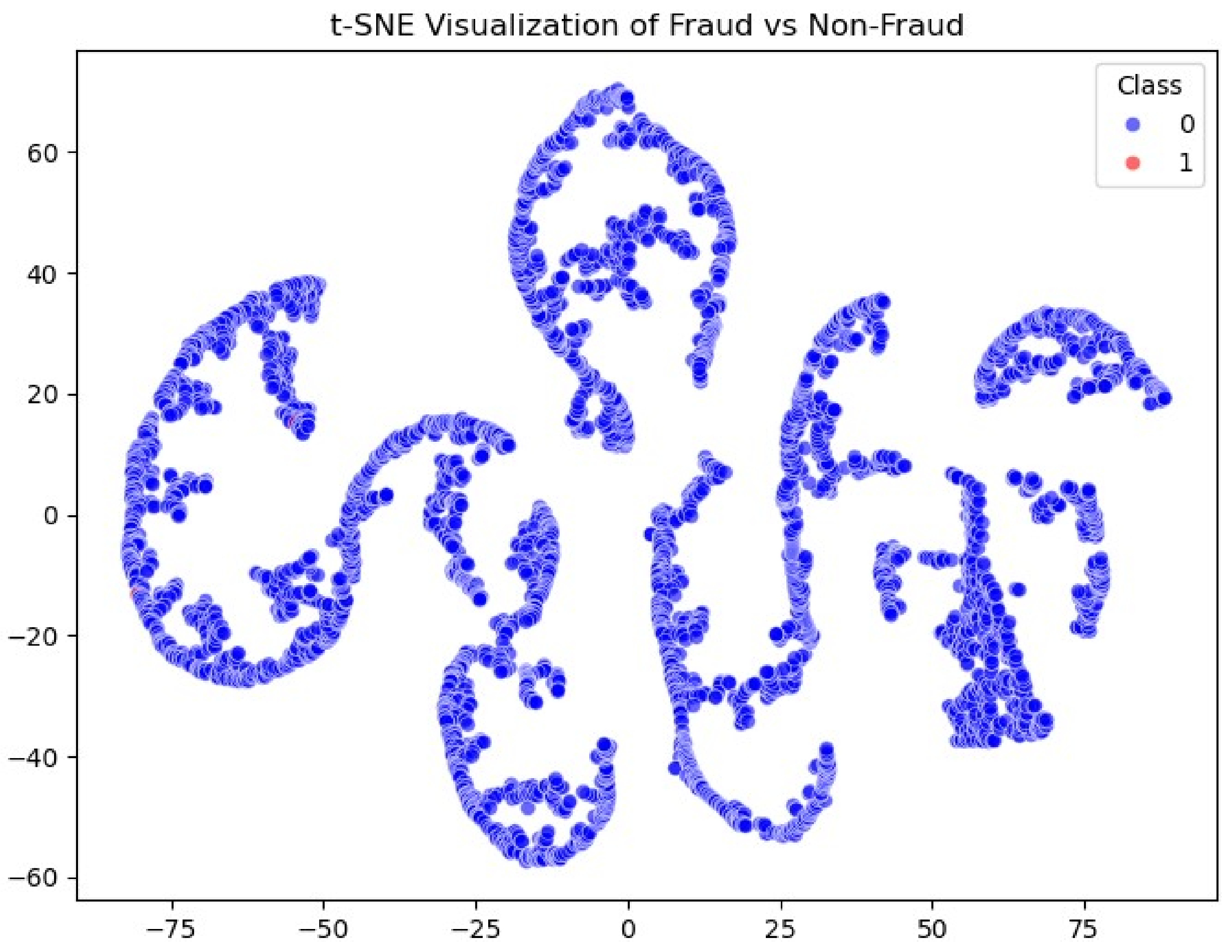
XGB Classifier

```
# Model Training - XGBoost
xgb = XGBClassifier( eval_metric='logloss')
xgb.fit(X_train, y_train)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, ...)

y_pred_xgb = xgb.predict(X_test)

# Evaluate the model
mae_xgb = n=mean_absolute_error(y_test,y_pred_xgb)
r2_xgb = r2_score(y_test,y_pred_xgb)
mse_xgb = mean_squared_error(y_test,y_pred_xgb)
acc_xgb = accuracy_score(y_test,y_pred_xgb)
f1_xgb = f1_score(y_test,y_pred_xgb)
print('XGB Classifier Model Performance :')
print('Mean Absolute Error :' , mae_xgb)
print('R^2 Score :' , r2_xgb)
print("Mean Squared Error :" ,mse_xgb )
```

```
print("Accuracy Score :" ,acc_xgb)
print("F 1 Score :" , f1_xgb)
```

```
XGB Classifier Model Performance :
Mean Absolute Error : 0.0004213335205926758
R^2 Score : 0.7546799811652293
Mean Squared Error : 0.0004213335205926758
Accuracy Score : 0.9995786664794073
F 1 Score : 0.8681318681318682
```

```
# Handle class imbalance using SMOTE
sm = SMOTE(random_state=42)
X_resampled, y_resampled = sm.fit_resample(X, y)

# Dimensionality Reduction (t-SNE)
tsne = TSNE(n_components=2, random_state=42)

X_embedded = tsne.fit_transform(X_resampled[:5000])  # use a subset
for speed
plt.figure(figsize=(8,6))
sns.scatterplot(x=X_embedded[:,0], y=X_embedded[:,1],
hue=y_resampled[:5000], palette=['blue', 'red'], alpha=0.6)
plt.title('t-SNE Visualization of Fraud vs Non-Fraud')
plt.show()
```



Logistic Regression

```
# Initialize and train the model
lr = LogisticRegression()
lr.fit(X_train, y_train)

C:\Users\Dnyanraj\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()
```

```python
# Predictions
y_pred_lr = lr.predict(X_test)

# Evaluate the model
mae_lr = n=mean_absolute_error(y_test,y_pred_lr)
r2_lr = r2_score(y_test,y_pred_lr)
mse_lr = mean_squared_error(y_test,y_pred_lr)
acc_lr = accuracy_score(y_test,y_pred_lr)
f1_lr = f1_score(y_test,y_pred_lr)
print('XGB Classifier Model Performance :')
print('Mean Absolute Error :' , mae_lr)
print('R^2 Score :' , r2_lr)
print("Mean Squared Error :" ,mse_lr)
print("Accuracy Score :" ,acc_lr)
print("F 1 Score :" , f1_lr)
```

```
XGB Classifier Model Performance :
Mean Absolute Error : 0.00133422281521014
R^2 Score : 0.22315327368989268
Mean Squared Error : 0.00133422281521014
Accuracy Score : 0.9986657771847899
F 1 Score : 0.5730337078651685
```

```python
# Model Evalution
Model_evaluation = pd.DataFrame({'Model' : ['Random Forest Classifier'
, 'XGB Classifier' , 'Logistic Regression'],
                                'Mean Absolute Error':
[mae_rf,mae_xgb,mae_lr],
                               'R^2 Score' : [r2_rf,r2_xgb,r2_lr],
                               'Mean Squared Error' :
[mse_rf,mse_xgb,mse_lr],
                               'Accuracy Score' :
[acc_rf,acc_xgb,acc_lr],
                               'F1 Score' :[f1_rf,f1_xgb,f1_lr]})
Model_evaluation
```

```
                      Model  Mean Absolute Error  R^2 Score  \
0  Random Forest Classifier             0.000439   0.744458
1            XGB Classifier             0.000421   0.754680
2       Logistic Regression             0.001334   0.223153

   Mean Squared Error  Accuracy Score  F1 Score
0            0.000439        0.999561  0.857143
1            0.000421        0.999579  0.868132
2            0.001334        0.998666  0.573034
```

Tools and Technologies Used

- Python: Pandas, NumPy, Matplotlib, Seaborn
- Scikit-learn: For preprocessing, modeling, and evaluation

- Imbalanced-learn: SMOTE for handling class imbalance
- XGBoost: High-performance gradient boosting model
- SHAP: Explainability of model decisions
- t-SNE: Dimensionality reduction for visualization

Observations :

- The dataset had 284,807 transactions, with 492 fraud cases (0.17%).
- After applying SMOTE, the minority class was balanced for training purposes.
- ROC and Precision-Recall curves confirmed good model performance, particularly for XGBoost and Random Forest.
- t-SNE visualization revealed distinct clusters of fraud and non-fraud transactions.
- Model explainability (planned via SHAP) will provide further insights into feature importance.

Recommendations

- Integrate model into a real-time fraud detection system.
- Monitor false positives regularly to avoid customer dissatisfaction.
- Update model periodically with new transaction data to adapt to new fraud patterns.

Conclusion

This project successfully demonstrates how a machine learning pipeline can be used to detect fraudulent credit card transactions. It emphasizes not just prediction accuracy, but also model transparency and robustness in handling real-world challenges like data imbalance.

References

- Kaggle Dataset: Credit Card Fraud Detection
- Imbalanced-learn (SMOTE): https://imbalanced-learn.org
- Scikit-learn: https://scikit-learn.org
- XGBoost: https://xgboost.readthedocs.io
- SHAP (Explainability): https://github.com/slundberg/shap
- Matplotlib & Seaborn: Visualization libraries used in the project