Credit Card Fraud Detection

1.Collection of data

2.Data Preprocessing

3.Training data

4.Model Training

5.Model Evaluation

Understanding the Problem:

- Credit card fraud detection is a binary classification problem where the goal is to accurately classify transactions as fraudulent (1) or non-fraudulent (0). The dataset is highly imbalanced, with fraudulent transactions being rare, which makes detecting fraud challenging. Effective detection models help financial institutions prevent financial loss and protect customers.
- The project focuses on detecting fraudulent credit card transactions using machine learning. The dataset includes anonymized transaction features (V1–V28), Time, Amount, and a Class label, where 0 denotes legitimate and 1 indicates fraudulent transactions.

Importing Dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
```

Loading dataset

```
data = pd.read_csv('creditcard.csv')
```

Data Analysis

- Dataset Structure: Contains 284,807 rows and 31 columns, with features
- V1–V28 derived via PCA for confidentiality, along with Time, Amount, and Class.
- Class Imbalance: Only a small fraction of transactions are fraudulent, leading to an imbalance that can bias the model toward predicting the majority class.
- Duplicate and Missing Values: Duplicates were removed and null values checked to ensure data integrity.
- Statistical Summary: Descriptive statistics (mean, std, min, max) highlighted feature distributions and potential outliers.

```
print("\nHead of the dataset:")
print(data.head())
```

```
Head of the dataset:
   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]
```

```
print("\nTail of the dataset:")
print(data.tail())
```

```
Tail of the dataset:
            Time         V1         V2        V3        V4        V5  \
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
284802      0
284803      0
284804      0
284805      0
284806      0

[5 rows x 31 columns]
```

```
print("\nDataset Description:")
print(data.describe())
```

```
Dataset Description:
                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]
```

```
print("\nDataset Shape:", data.shape)
```

```
Dataset Shape: (284807, 31)
```

```
print("\nDataset Information:", data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 275663 entries, 0 to 284806
Data columns (total 30 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   V1      275663 non-null  float64
 1   V2      275663 non-null  float64
 2   V3      275663 non-null  float64
 3   V4      275663 non-null  float64
 4   V5      275663 non-null  float64
 5   V6      275663 non-null  float64
 6   V7      275663 non-null  float64
 7   V8      275663 non-null  float64
 8   V9      275663 non-null  float64
 9   V10     275663 non-null  float64
 10  V11     275663 non-null  float64
 11  V12     275663 non-null  float64
 12  V13     275663 non-null  float64
 13  V14     275663 non-null  float64
 14  V15     275663 non-null  float64
 15  V16     275663 non-null  float64
 16  V17     275663 non-null  float64
 17  V18     275663 non-null  float64
 18  V19     275663 non-null  float64
 19  V20     275663 non-null  float64
 20  V21     275663 non-null  float64
 21  V22     275663 non-null  float64
 22  V23     275663 non-null  float64
 23  V24     275663 non-null  float64
 24  V25     275663 non-null  float64
 25  V26     275663 non-null  float64
 26  V27     275663 non-null  float64
 27  V28     275663 non-null  float64
 28  Amount  275663 non-null  float64
 29  Class   275663 non-null  int64
dtypes: float64(29), int64(1)
memory usage: 65.2 MB

Dataset Shape: None
```

```
data.describe()
```

	V1	V2	V3	V4	V5	V6	V7	V8	
count	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.0
mean	-0.037460	-0.002430	0.025520	-0.004359	-0.010660	-0.014206	0.008586	-0.005698	-0.0
std	1.952522	1.667260	1.507538	1.424323	1.378117	1.313213	1.240348	1.191596	1.1
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.4
25%	-0.941105	-0.614040	-0.843168	-0.862847	-0.700192	-0.765861	-0.552047	-0.209618	-0.6
50%	-0.059659	0.070249	0.200736	-0.035098	-0.060556	-0.270931	0.044848	0.022980	-0.0
75%	1.294471	0.819067	1.048461	0.753943	0.604521	0.387704	0.583885	0.322319	0.5
max	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.5

8 rows × 30 columns

```
print("\nNull values in the dataset:")
print(data.isnull().sum())
```

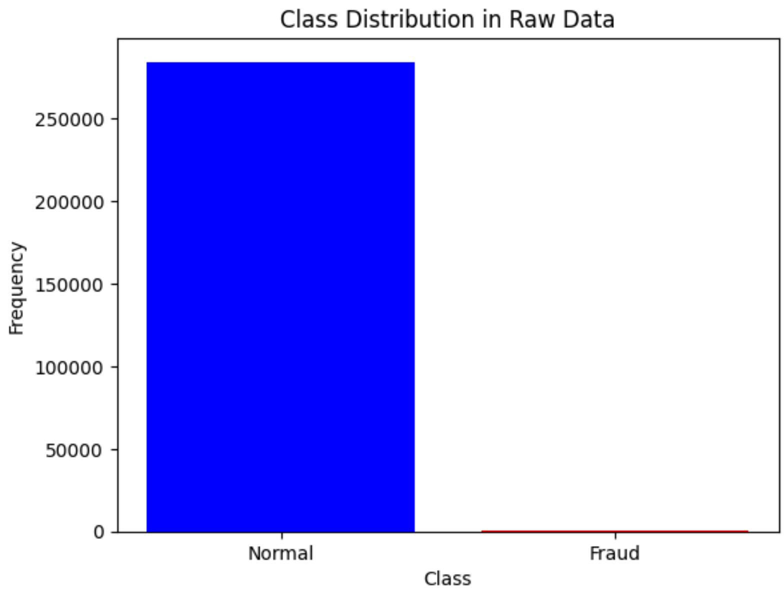
```
Null values in the dataset:
Time      0
V1        0
V2        0
V3        0
```

```
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
```

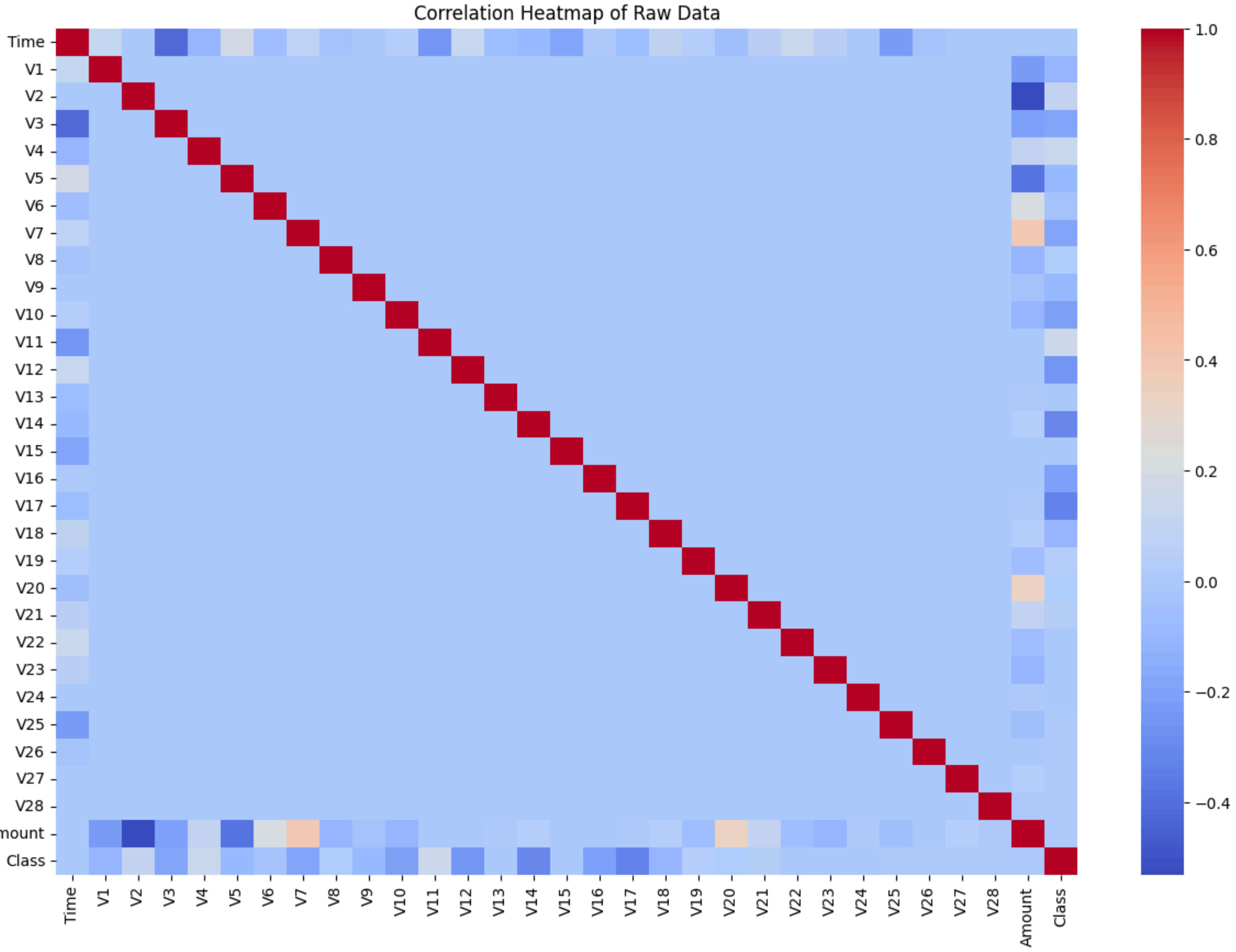
Data Visualization:

- Class Distribution (Bar Plot): Displayed the overwhelming number of legitimate transactions compared to fraudulent ones.
- Correlation Heatmap: Identified feature relationships and dependencies, helping in feature selection.
- Scatter Plot: Visualized the distribution of transaction amounts across both classes, indicating that fraud often involves smaller amounts.

```
# Visualization of  bar plot
class_counts = data['Class'].value_counts()
plt.bar(class_counts.index, class_counts.values, color=['blue', 'red'])
plt.xticks([0, 1], ['Normal', 'Fraud'])
plt.title('Class Distribution in Raw Data')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```




```
# Heatmap to visualize correlations
plt.figure(figsize=(15, 10))
sns.heatmap(data.corr(), cmap='coolwarm', annot=False)
plt.title('Correlation Heatmap of Raw Data')
plt.show()
```



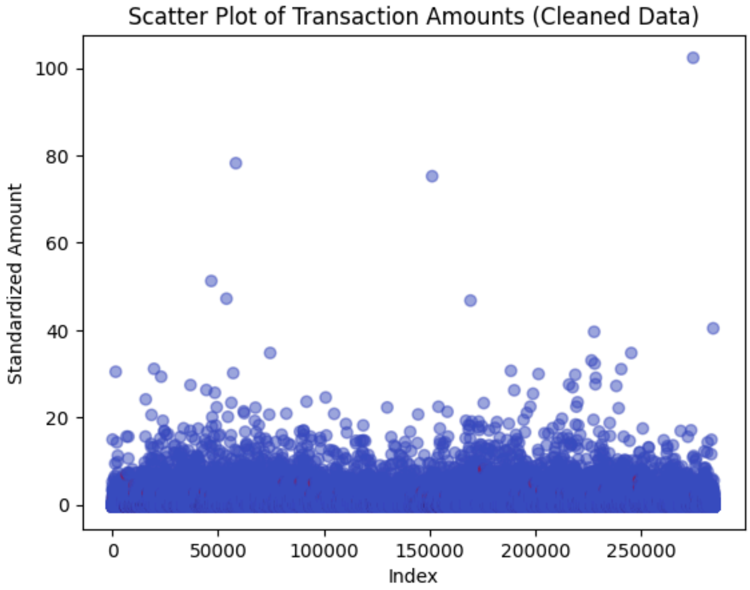

```
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))


data = data.drop(['Time'], axis=1)
data = data.drop_duplicates()
print("\nDataset Shape after cleaning:", data.shape)
print(data['Class'].value_counts())
```

```
Dataset Shape after cleaning: (275663, 30)
Class
0    275190
1       473
Name: count, dtype: int64
```


```
print("\nDataset Shape after cleaning:", data.shape)
print(data['Class'].value_counts())
```

```
Dataset Shape after cleaning: (275663, 30)
Class
0    275190
1       473
Name: count, dtype: int64
```


```
# Scatter plot of cleaned data
plt.scatter(data.index, data['Amount'], alpha=0.5, c=data['Class'], cmap='coolwarm')
plt.title('Scatter Plot of Transaction Amounts (Cleaned Data)')
plt.xlabel('Index')
plt.ylabel('Standardized Amount')
plt.show()
```




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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree Classifier": DecisionTreeClassifier()
}
```

```
# Evaluate classifiers on the raw data
print("\nEvaluation on Raw Data:")
raw_scores = []
for name, clf in classifiers.items():
    print(f"\n========== {name} ===========")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    raw_scores.append([name, acc, prec, rec, f1])
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

```
Evaluation on Raw Data:

========== Logistic Regression ===========
Accuracy: 0.9993
Precision: 0.8906
Recall: 0.6264
F1 Score: 0.7355

========== Decision Tree Classifier ===========
Accuracy: 0.9990
Precision: 0.6837
Recall: 0.7363
F1 Score: 0.7090
```

Training Data Preparation:

- Feature Scaling: Standardized the Amount feature to bring it to the same scale as other features.
- Feature Reduction: Dropped the Time feature due to its low relevance.
- Data Splitting: Divided the dataset into 80% training and 20% testing sets to ensure unbiased model evaluation.

Balancing the Dataset:

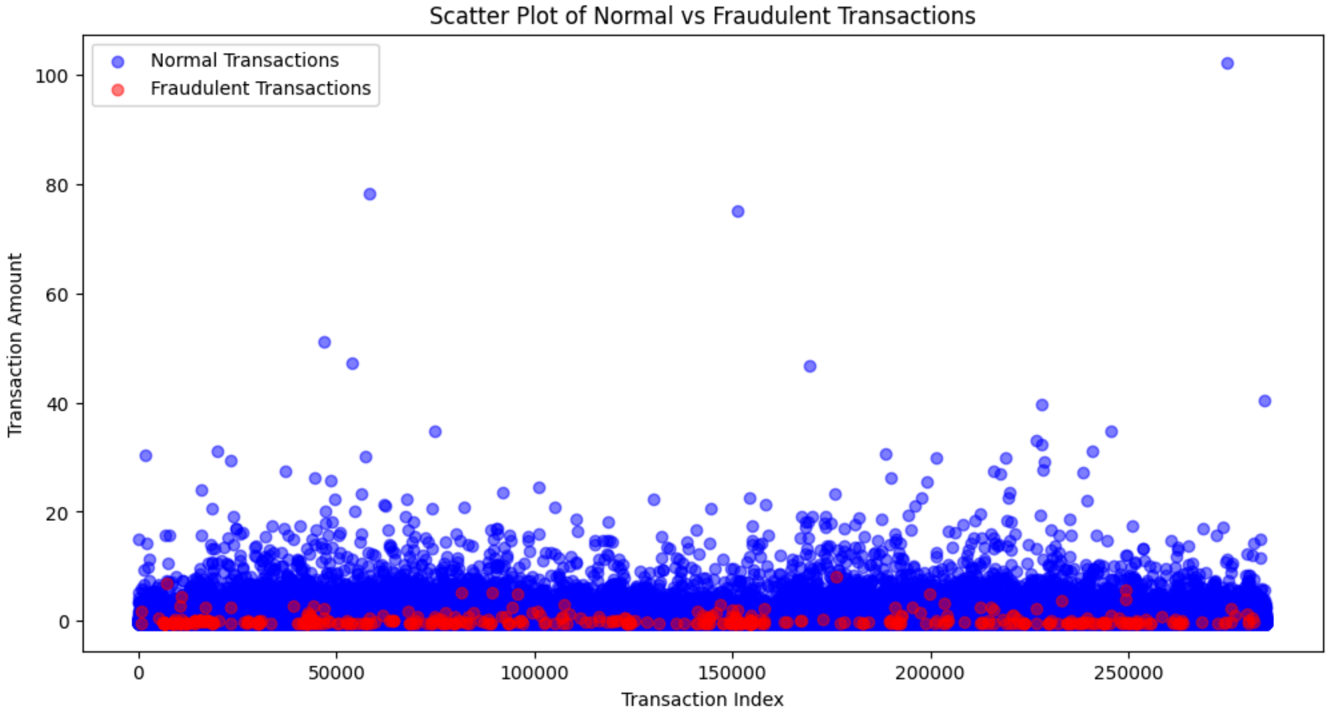
- *Under-Sampling: Reduced non-fraudulent samples to balance the dataset.*

```
normal = data[data['Class'] == 0]
fraud = data[data['Class'] == 1]
```

```
normal_sample = normal.sample(n=fraud.shape[0])
new_data = pd.concat([normal_sample, fraud], ignore_index=True)
```

```
plt.figure(figsize=(12, 6))
plt.scatter(normal.index, normal['Amount'], alpha=0.5, label='Normal Transactions', color='blue')
plt.scatter(fraud.index, fraud['Amount'], alpha=0.5, label='Fraudulent Transactions', color='red')
plt.title('Scatter Plot of Normal vs Fraudulent Transactions')
plt.xlabel('Transaction Index')
plt.ylabel('Transaction Amount')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Creating legend with loc="best" can be slow wi
  fig.canvas.print_figure(bytes_io, **kw)
```




```
# Split the balanced dataset
X = new_data.drop('Class', axis=1)
y = new_data['Class']
```

```
new_data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
0	-1.475315	1.049001	1.781780	1.101956	0.461021	-0.484097	1.052987	-0.321786	-0.515330	0.649340	...	-0.544247	-1.036156	0.1
1	0.956891	-1.403289	-0.037293	-1.440457	-0.816330	0.193657	-0.476316	-0.172668	1.767556	-0.866729	...	-0.396968	-0.612041	-0.3
2	1.107298	-0.170076	1.499309	0.610809	-1.214932	-0.196013	-0.822915	0.157357	0.345125	0.038578	...	0.306800	0.909278	-0.0
3	1.194352	-0.633998	0.033808	-0.241007	-0.615379	-0.397688	-0.162676	-0.272583	-1.093880	0.650803	...	-0.432383	-0.975768	-0.1
4	1.445825	-1.167408	0.065631	-1.476644	-1.363367	-0.640204	-0.929969	-0.071838	-1.989845	1.657485	...	-0.112756	-0.098297	-0.1

5 rows × 30 columns

```
new_data.tail()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
941	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	-5.587794	...	0.778584	-0.319189	0.
942	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	-3.232153	...	0.370612	0.028234	-0.
943	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	-3.463891	...	0.751826	0.834108	0.
944	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	-5.245984	...	0.583276	-0.269209	-0.
945	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	-0.888722	...	-0.164350	-0.295135	-0.

5 rows × 30 columns

```
new_data.describe()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	
count	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	...	946.0
mean	-2.250028	1.675657	-3.352272	2.179847	-1.475085	-0.725359	-2.581325	0.453395	-1.297265	-2.729448	...	0.2
std	5.319163	3.561901	6.017910	3.194467	4.118270	1.694788	5.550428	4.037324	2.257084	4.357244	...	1.9
min	-30.552380	-14.811940	-31.103685	-4.011848	-22.105532	-7.216809	-43.557242	-41.044261	-13.434066	-24.588262	...	-22.7
25%	-2.752126	-0.168618	-4.951307	-0.198957	-1.754592	-1.601628	-3.022498	-0.222571	-2.274985	-4.480683	...	-0.1
50%	-0.751764	0.979787	-1.219251	1.197590	-0.456031	-0.635555	-0.658624	0.157855	-0.733363	-0.876926	...	0.1
75%	0.950579	2.653723	0.334797	4.127228	0.557988	0.069545	0.325020	0.852353	0.164724	0.067226	...	0.6
max	2.332026	22.057729	3.145596	12.114672	11.095089	6.474115	5.802537	20.007208	4.401053	6.318517	...	27.2

8 rows × 30 columns

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Evaluate classifiers on the undersampled data
print("\nEvaluation on Undersampled Data:")
undersample_scores = []
for name, clf in classifiers.items():
    print(f"\n========== {name} (Undersampling) ===========")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    undersample_scores.append([name, acc, prec, rec, f1])
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

```
Evaluation on Undersampled Data:

========== Logistic Regression (Undersampling) ===========
Accuracy: 0.9526
Precision: 0.9895
Recall: 0.9216
F1 Score: 0.9543

========== Decision Tree Classifier (Undersampling) ===========
Accuracy: 0.8895
Precision: 0.9010
Recall: 0.8922
F1 Score: 0.8966
```

Balancing the Dataset:

- *Over-Sampling (SMOTE):Synthetic samples of fraudulent transactions were generated to address class imbalance.*

Model Training

```
# Oversampling using SMOTE
X_res, y_res = SMOTE().fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
```

Model Evaluation:

- Performance Metrics:
  - Accuracy: Overall correctness of predictions.
  - Precision: How many predicted frauds were actual frauds.
  - Recall: Ability to detect actual frauds.
  - F1-Score: Balance between precision and recall.

```
# Evaluate classifiers on the oversampled data
print("\nEvaluation on Oversampled Data:")
oversample_scores = []
for name, clf in classifiers.items():
    print(f"\n========== {name} (Oversampling) ===========")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    oversample_scores.append([name, acc, prec, rec, f1])
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

```
Evaluation on Oversampled Data:

========== Logistic Regression (Oversampling) ===========
Accuracy: 0.9526
Precision: 0.9895
Recall: 0.9216
F1 Score: 0.9543

========== Decision Tree Classifier (Oversampling) ===========
Accuracy: 0.9105
Precision: 0.9293
Recall: 0.9020
F1 Score: 0.9154
```

Final Result

```
def create_summary_table(scores, title):
    summary_table = pd.DataFrame(scores, columns=["Model", "Accuracy", "Precision", "Recall", "F1 Score"])
    summary_table.set_index("Model", inplace=True)
    print(f"\n{title}:")
    print(summary_table)


create_summary_table(raw_scores, "Summary Table for Raw Data")
create_summary_table(undersample_scores, "Summary Table for Undersampled Data")
create_summary_table(oversample_scores, "Summary Table for Oversampled Data")
```

```
Summary Table for Raw Data:
                          Accuracy  Precision    Recall  F1 Score
Model
Logistic Regression       0.999256   0.890625  0.626374  0.735484
Decision Tree Classifier  0.999002   0.683673  0.736264  0.708995

Summary Table for Undersampled Data:
                          Accuracy  Precision    Recall  F1 Score
Model
Logistic Regression       0.952632   0.989474  0.921569  0.954315
Decision Tree Classifier  0.889474   0.900990  0.892157  0.896552

Summary Table for Oversampled Data:
                          Accuracy  Precision    Recall  F1 Score
Model
Logistic Regression       0.952632   0.989474  0.921569  0.954315
Decision Tree Classifier  0.910526   0.929293  0.901961  0.915423
```