CREDIT CARD FRAUD DETECTION

- Credit card fraud detection is a process and set of
- techniques used by financial institutions, merchants, and
- credit card companies to identify and prevent
- unauthorized or fraudulent use of credit cards. It involves
- the use of various methods to analyze and monitor
- credit card transactions in real-time to detect any
- unusual or suspicious activity. Some common:

PROBLEM STATEMENT

- DEVELOP AN EFFECTIVE AND SCALABLE CREDIT CARDFRAUD DETECTION SYSTEM USING MACHINE LEARNING TECHNIQUES TO IDENTIFY AND PREVENT FRAUDULENT TRANSACTIONS IN REAL-TIME, MINIMIZING FINANCIAL LOSSES AND ENSURING A SEAMLESS USER EXPERIENCE FOR LEGITIMATE CUSTOMERS.

DESIGN THINKING

- DESIGN THINKING IS A USER-CENTERED APPROACH TO PROBLEM-SOLVING THAT INVOLVES EMPATHIZING WITH USERS, DEFINING THE PROBLEM, IDEATING SOLUTIONS, PROTOTYPING, AND TESTING.
- PHASES OF DESIGN THINKING IN CREDIT CAURD FRAUD DETECTION:
- 1.EMPATHIZE
- 2.DEFINE
- 3.IDEATE
- 4.PROTOTYPE
- 5.TEST.

EMPATHIZING WITH USER

- UNDERSTAND THE NEEDS AND PAIN POINTS OF BOTH CUSTOMERS AND FINANCIAL INSTITUTIONS.
- COLLECT AND ANALYZE DATA ON PAST FRAUDULENT TRANSACTIONS AND THEIR IMPACT ON CUSTOMERS AND BUSINESSES.
- INTERVIEW FRAUD ANALYSTS, SECURITY EXPERTS, AND CUSTOMERS TO GAIN INSIGHTS INTO THEIR PERSPECTIVES AND CHALLENGES.

DEFINING THE PROBLEM:

- Clearly define the problem by identifying the main issues in credit card fraud detection.
- Create a problem statement that addresses the key concerns, such as reducing false positives, improving detection accuracy, and enhancing customer experience.

IDEATING SOLUTIONS:

- Developing advanced anomaly detection algorithms.
- Implementing real-time transaction monitoring.
- Enhancing user authentication and verification processes.
- Exploring behavior-based modeling for fraud detection.
- Using deep learning models for pattern recognition.

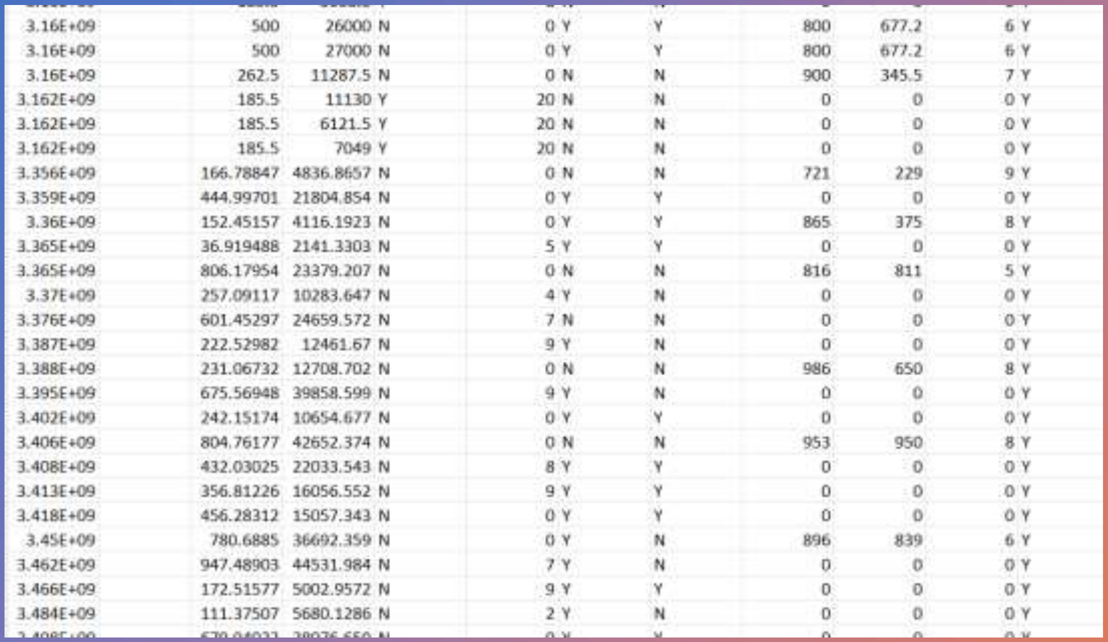
- Encourage cross-functional collaboration among data scientists, engineers, domain experts, and UX designers to brainstorm innovative solutions.

PROTOTYPE

- Create prototypes or mock-ups of potential solutions. For instance:
- Develop a user interface for fraud analysts to investigate suspicious transactions efficiently.
  - Create a machine learning model prototype for fraud detection and prevention.
- These prototypes should be low-cost and

TESTING

- Gather feedback from stakeholders, including fraud analysts, customers, and technical experts, on the prototypes.
- Iterate on the prototypes based on the feedback received.
- Conduct simulations or pilot tests to evaluate the effectiveness of the proposed solutions.



3.16E+09	500	26000	N	0	Y	Y	800	677.2	6	Y
3.16E+09	500	27000	N	0	Y	Y	800	677.2	6	Y
3.16E+09	262.5	11287.5	N	0	N	N	900	345.5	7	Y
3.162E+09	185.5	11130	Y	20	N	N	0	0	0	Y
3.162E+09	185.5	6121.5	Y	20	N	N	0	0	0	Y
3.162E+09	185.5	7049	Y	20	N	N	0	0	0	Y
3.356E+09	166.78847	4836.8657	N	0	N	N	721	229	9	Y
3.359E+09	444.99701	21804.854	N	0	Y	Y	0	0	0	Y
3.36E+09	152.45157	4116.1923	N	0	Y	Y	865	375	8	Y
3.365E+09	36.919488	2141.3303	N	5	Y	Y	0	0	0	Y
3.365E+09	806.17954	23379.207	N	0	N	N	816	811	5	Y
3.37E+09	257.09117	10283.647	N	4	Y	N	0	0	0	Y
3.376E+09	601.45297	24659.572	N	7	N	N	0	0	0	Y
3.387E+09	222.52982	12461.67	N	9	Y	N	0	0	0	Y
3.388E+09	231.06732	12708.702	N	0	N	N	986	650	8	Y
3.395E+09	675.56948	39858.599	N	9	Y	N	0	0	0	Y
3.402E+09	242.15174	10654.677	N	0	Y	Y	0	0	0	Y
3.406E+09	804.76177	42652.374	N	0	N	N	953	950	8	Y
3.408E+09	432.03025	22033.543	N	8	Y	Y	0	0	0	Y
3.413E+09	356.81226	16056.552	N	9	Y	Y	0	0	0	Y
3.418E+09	456.28312	15057.343	N	0	Y	Y	0	0	0	Y
3.45E+09	780.6885	36692.359	N	0	Y	N	896	839	6	Y
3.462E+09	947.48903	44531.984	N	7	Y	N	0	0	0	Y
3.466E+09	172.51577	5002.9572	N	9	Y	Y	0	0	0	Y
3.484E+09	111.37507	5680.1286	N	2	Y	N	0	0	0	Y

DATA SET COLLECTION

The first step is to collect reliable data so that your machine learning

model can find the correct patterns.The quality of the data that you feed to

the machine will determine how accurate your model

DATASET LINK:

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

DATA PREPROCESSING

clean and preprocess the data to remove noice and inconsistencies handle missing values and outliers

FEAUTURE ENGINEERING

- Create a relevant features from the data such as transaction, frequency, Location based features and user behavior patterns

MODEL SELECTION

- hoose appropriate machine learning or deep learning models for fraud detection, such as logistic regression, decision trees, random forests, or neural networks.Consider ensemble methods for improved performance

TRAIN AND TEST

- To train and test a model for credit card fraud detection, you should follow these steps, which include splitting your dataset into training and testing sets:
- Data Preparation:Gather and preprocess your credit card transaction dataset. This dataset should include labeled data, where each transaction is classified as either legitimate or fraudulent.
- Data Splitting:Divide your dataset into three subsets: training, validation, and testing. A common split might be 60% for training, 20% for validation, and 20% for testing. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set is used for the final evaluation

- Training Set:The training set is used to train your machine learning or deep learning model. It should contain a diverse representation of both legitimate and fraudulent transactions to ensure the model learns to distinguish between the two classes.
- Validation Set:The validation set is used during the training process to fine-tune model hyperparameters and assess the model's performance. You can use it to prevent overfitting.

- Testing Set: The testing set is kept separate and used only after model training is complete. It is used to evaluate the model's performance on unseen data, simulating real-world scenarios.
- Model Training:Train your chosen model using the training data, adjusting hyperparameters and architecture as needed. Common models for fraud detection include logistic regression, decision trees, random forests, and deep neural networks.

October 23, 2023

1. Dataset Information

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset 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.

It contains only numerical input variables which are the result of a PCA transformation. Unfortu- nately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Fea- ture 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example- dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

2. Import modules

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

0.3 Loading the dataset

[ ]:
```
df = pd.read_csv("C:\Users\sdeva\Downloads\archive(1)\creditcard.csv")
df.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]
```

```
[3]: # statistical info
df.describe()
```

```
[3]:                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-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  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-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   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-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   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-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]
```

[4]:
```
# datatype info
df.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
```

0.4 Preprocessing the dataset

[5]:
```
# check for null values
df.isnull().sum()
```

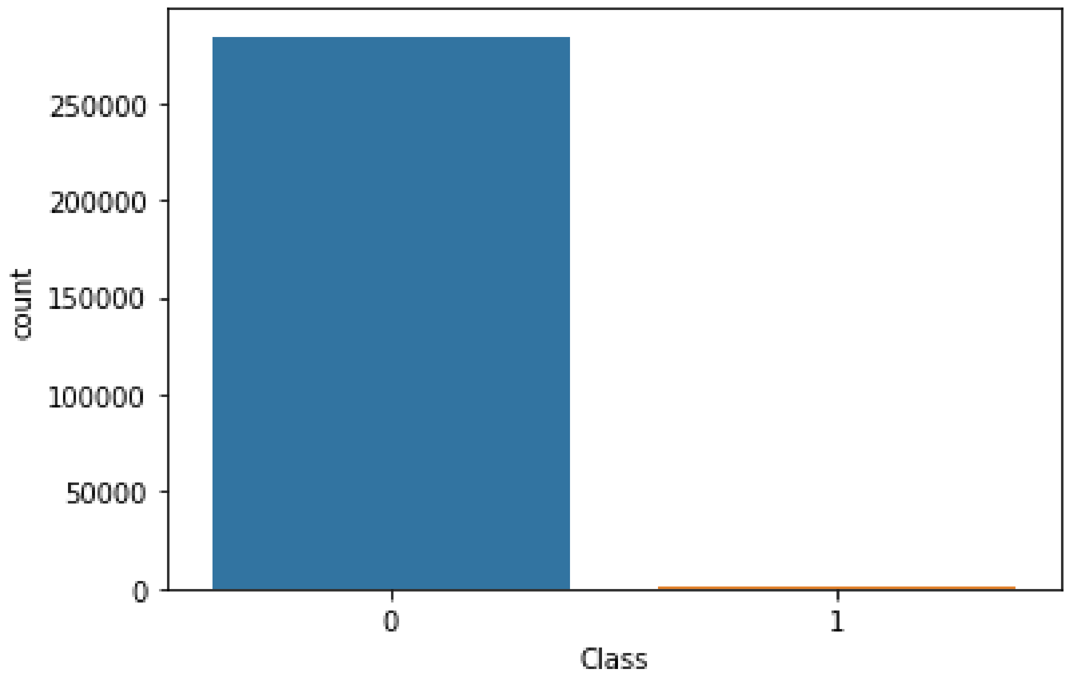
[5]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
```

```
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

0.5 Exploratory Data Analysis

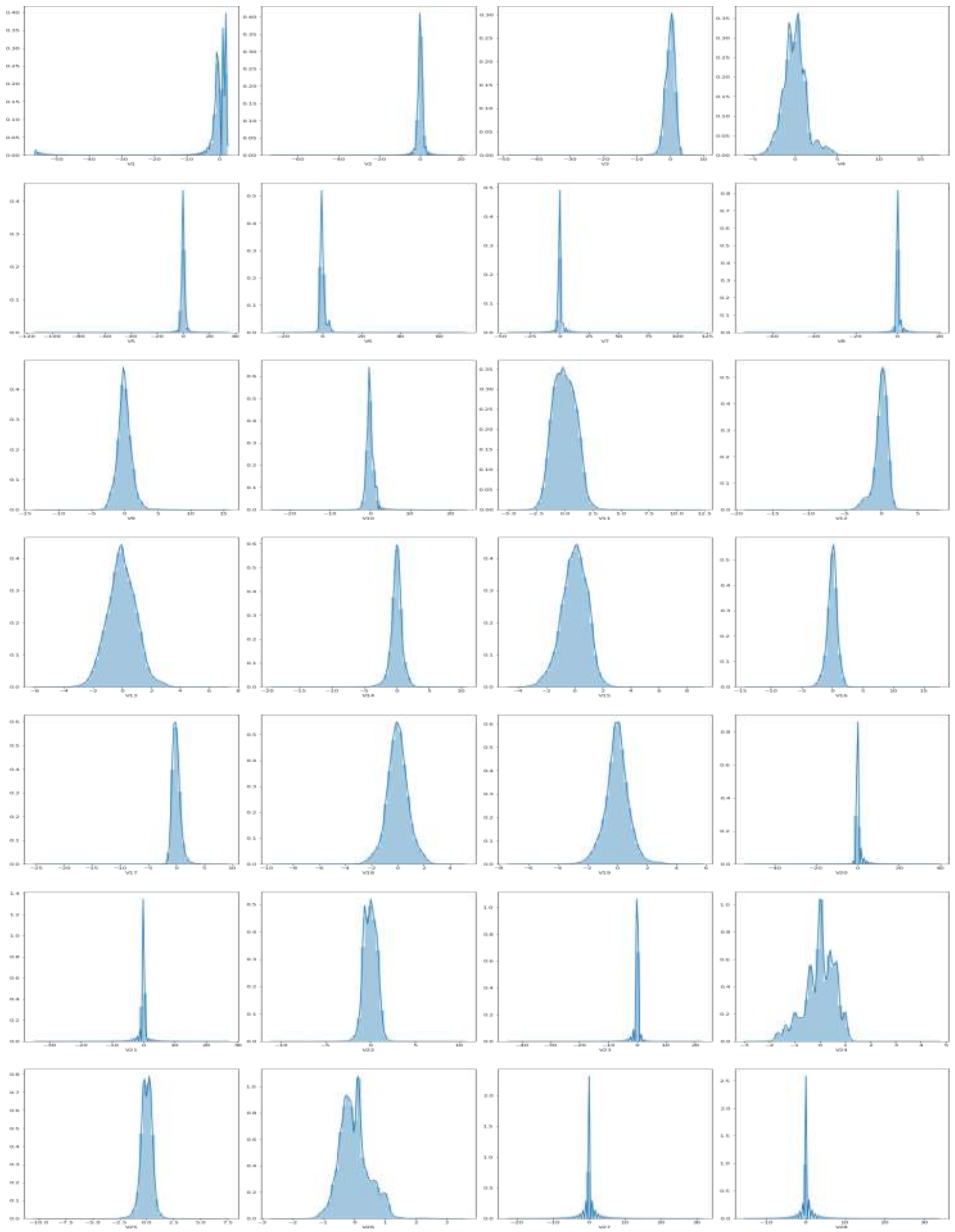
[6]:
```
sns.countplot(df['Class'])
```

[6]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



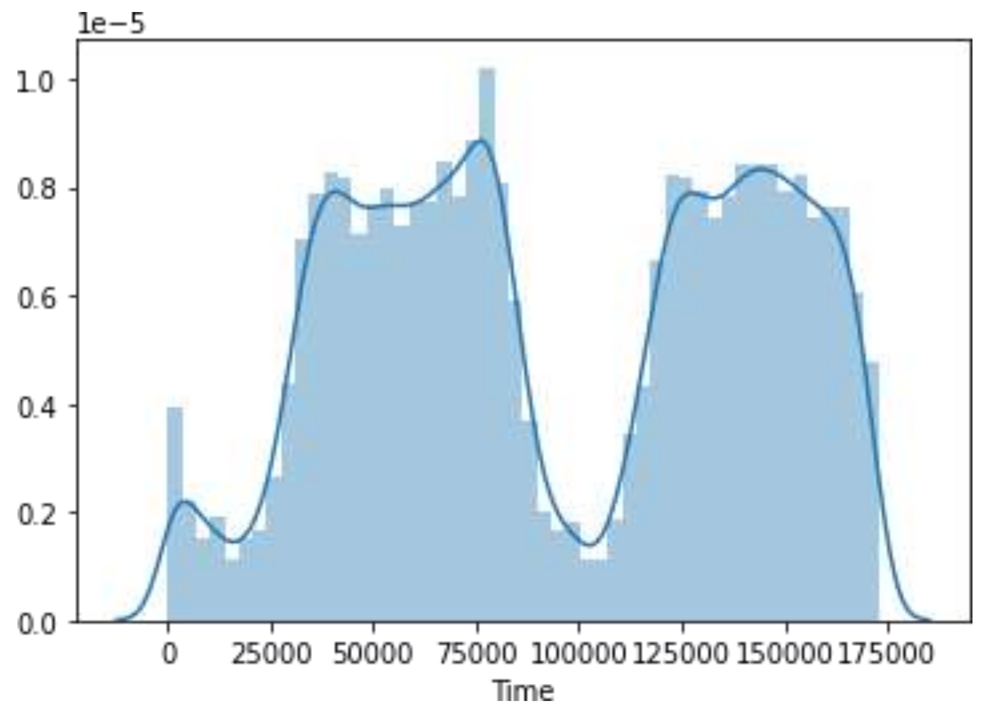
[10]:
```
df_temp = df.drop(columns=['Time', 'Amount', 'Class'], axis=1)

# create dist plots
fig, ax = plt.subplots(ncols=4, nrows=7, figsize=(20, 50))
index = 0
```

```
ax = ax.flatten()

for col in df_temp.columns:
    sns.distplot(df_temp[col], ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=5)
```



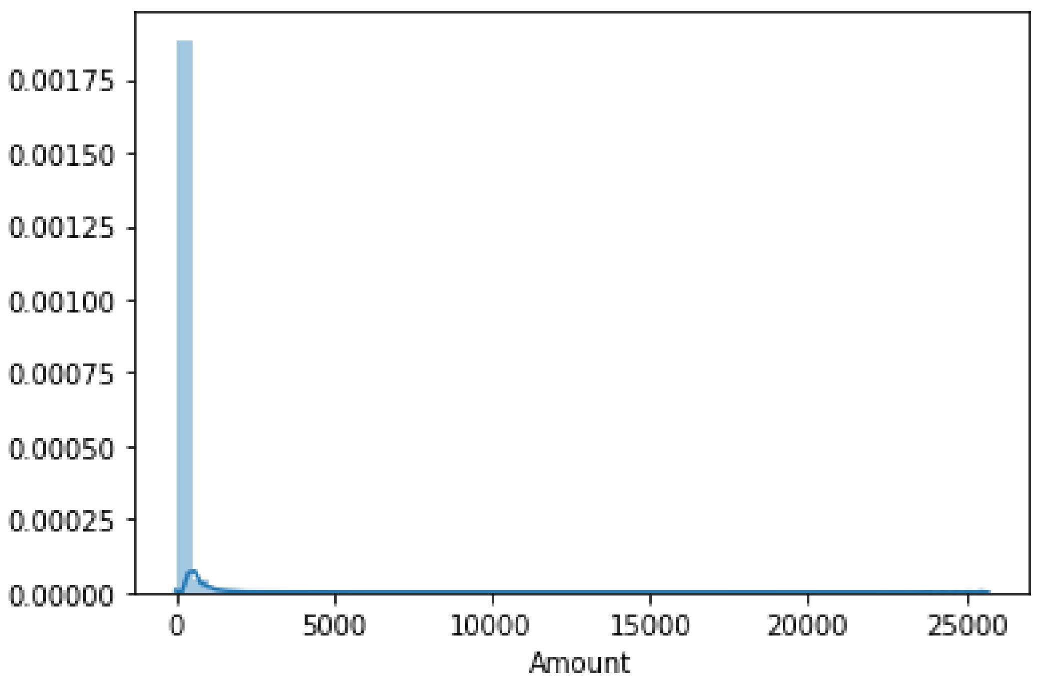
```
[11]: sns.distplot(df['Time'])
```

```
[11]: <AxesSubplot:xlabel='Time'>
```



```
[12]: sns.distplot(df['Amount'])
```

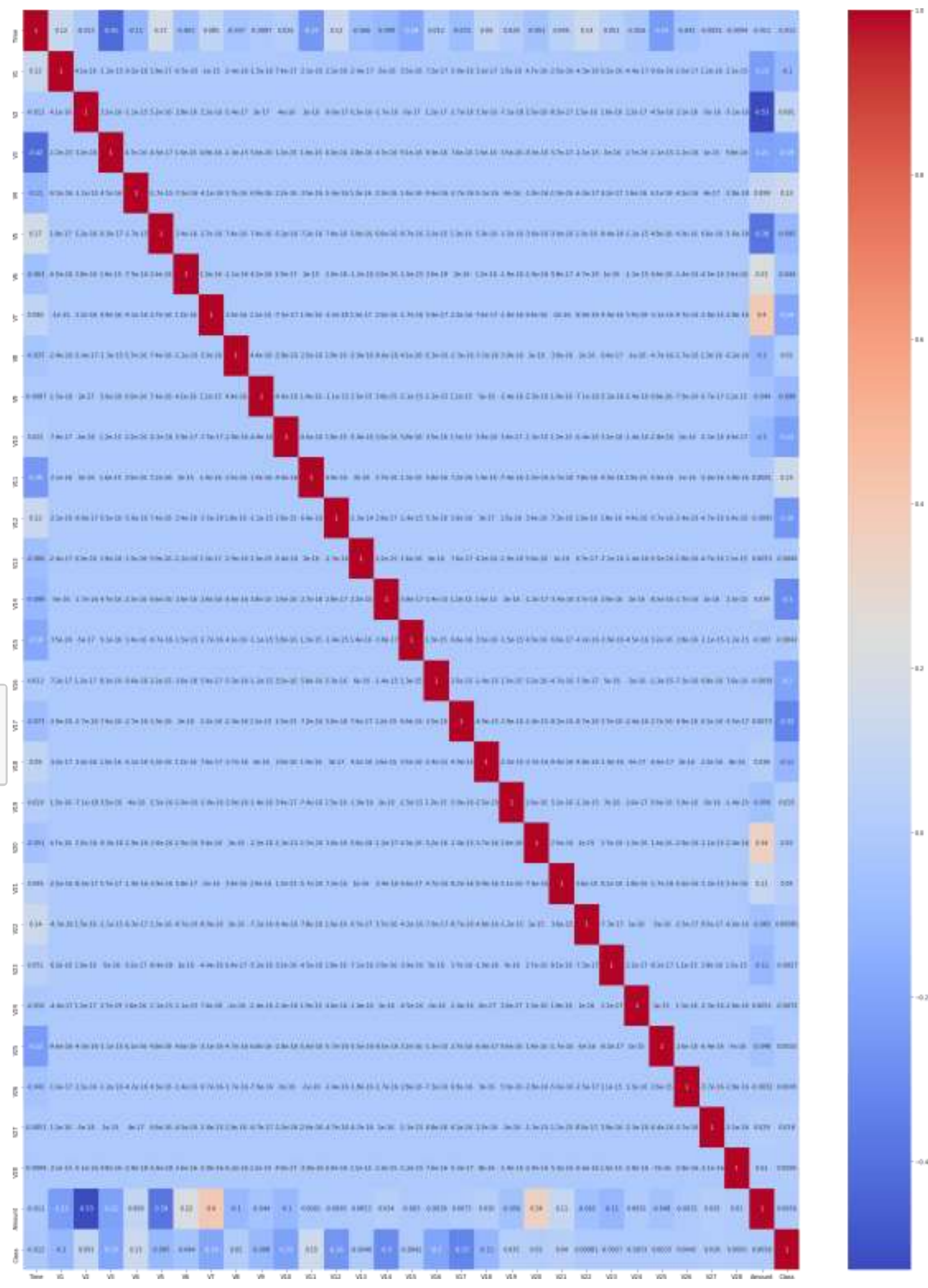
```
[12]: <AxesSubplot:xlabel='Amount'>
```



0.6 Coorelation Matrix

[14]:
```
corr = df.corr()
plt.figure(figsize=(30,40))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

[14]:
```
<AxesSubplot:>
```



0.7 Input Split

```
[15]: X = df.drop(columns=['Class'], axis=1)
      y = df['Class']
```

0.8 Standard Scaling

```
[16]: from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      x_scaler = sc.fit_transform(X)
```

```
[18]: x_scaler[-1]
```

```
[18]: array([ 1.64205773, -0.27233093, -0.11489898,  0.46386564, -0.35757    ,
             -0.00908946, -0.48760183,  1.27476937, -0.3471764 ,  0.44253246,
             -0.84072963, -1.01934641, -0.0315383 , -0.18898634, -0.08795849,
              0.04515766, -0.34535763, -0.77752147,  0.1997554 , -0.31462479,
              0.49673933,  0.35541083,  0.8861488 ,  0.6033653 ,  0.01452561,
             -0.90863123, -1.69685342, -0.00598394,  0.04134999,  0.51435531])
```

0.9 Model Training

```
[23]: # train test split
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import classification_report, f1_score
      x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.
       ↪25, random_state=42, stratify=y)
```

```
[25]: from sklearn.linear_model import LogisticRegression
      model = LogisticRegression()
      # training
      model.fit(x_train, y_train)
      # testing
      y_pred = model.predict(x_test)
      print(classification_report(y_test, y_pred))
      print("F1 Score:",f1_score(y_test, y_pred))
```

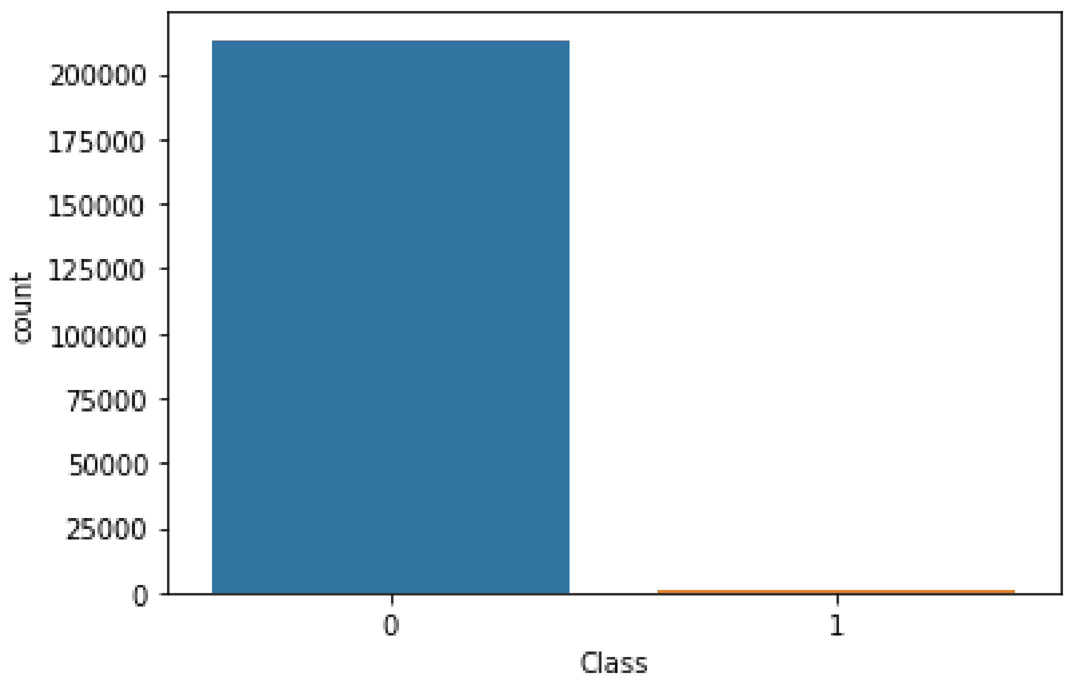
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.85      0.63      0.72       123

    accuracy                           1.00     71202
   macro avg       0.92      0.81      0.86     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.719626168224299
```

[26]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
#training
model.fit(x_train, y_train)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.95      0.76      0.85       123

    accuracy                           1.00     71202
   macro avg       0.97      0.88      0.92     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.846846846846847
```

[37]:
```
from xgboost import XGBClassifier
model = XGBClassifier(n_jobs=-1)
#training
model.fit(x_train, y_train)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

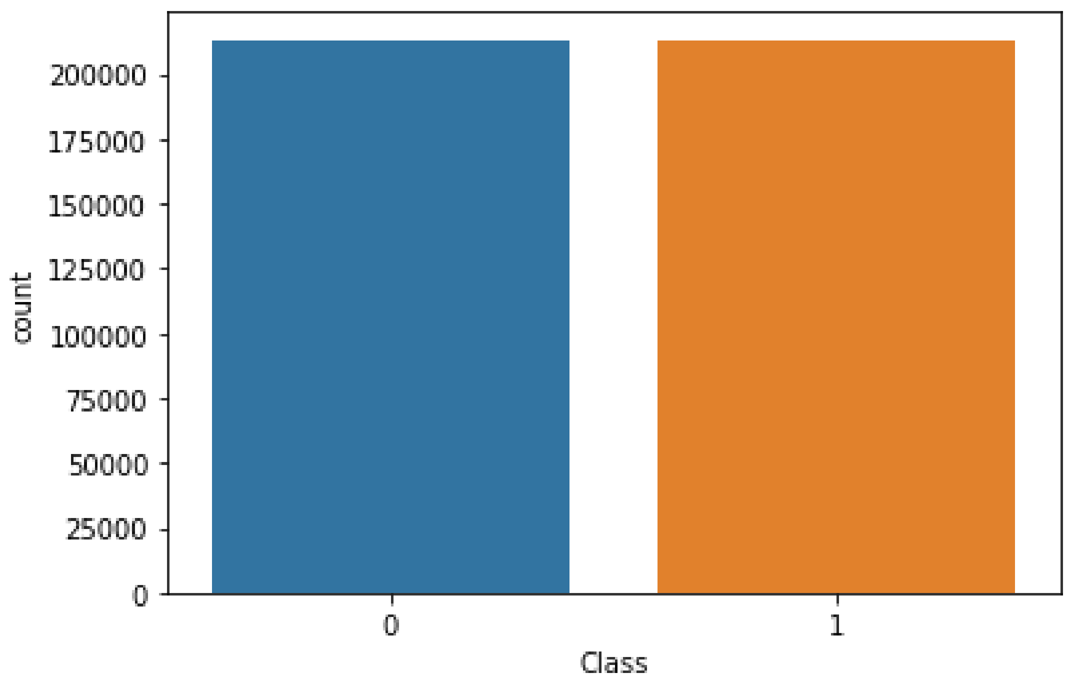
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.94      0.80      0.86       123

    accuracy                           1.00     71202
   macro avg       0.97      0.90      0.93     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.8634361233480178
```



```
[29]: # hint - use combination of over sampling and under sampling
      # balance the class with equal distribution
      from imblearn.over_sampling import SMOTE
      over_sample = SMOTE()
      x_smote, y_smote = over_sample.fit_resample(x_train, y_train)
```

```
[30]: sns.countplot(y_smote)
```

```
[30]: <AxesSubplot:xlabel='Class', ylabel='count'>
```



```
[33]: from sklearn.linear_model import LogisticRegression
      model = LogisticRegression()
      # training
      model.fit(x_smote, y_smote)
      # testing
      y_pred = model.predict(x_test)
      print(classification_report(y_test, y_pred))
      print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     71079
           1       0.06      0.89      0.11       123

    accuracy                           0.98     71202
   macro avg       0.53      0.93      0.55     71202
weighted avg       1.00      0.98      0.99     71202

F1 Score: 0.11202466598150052
```

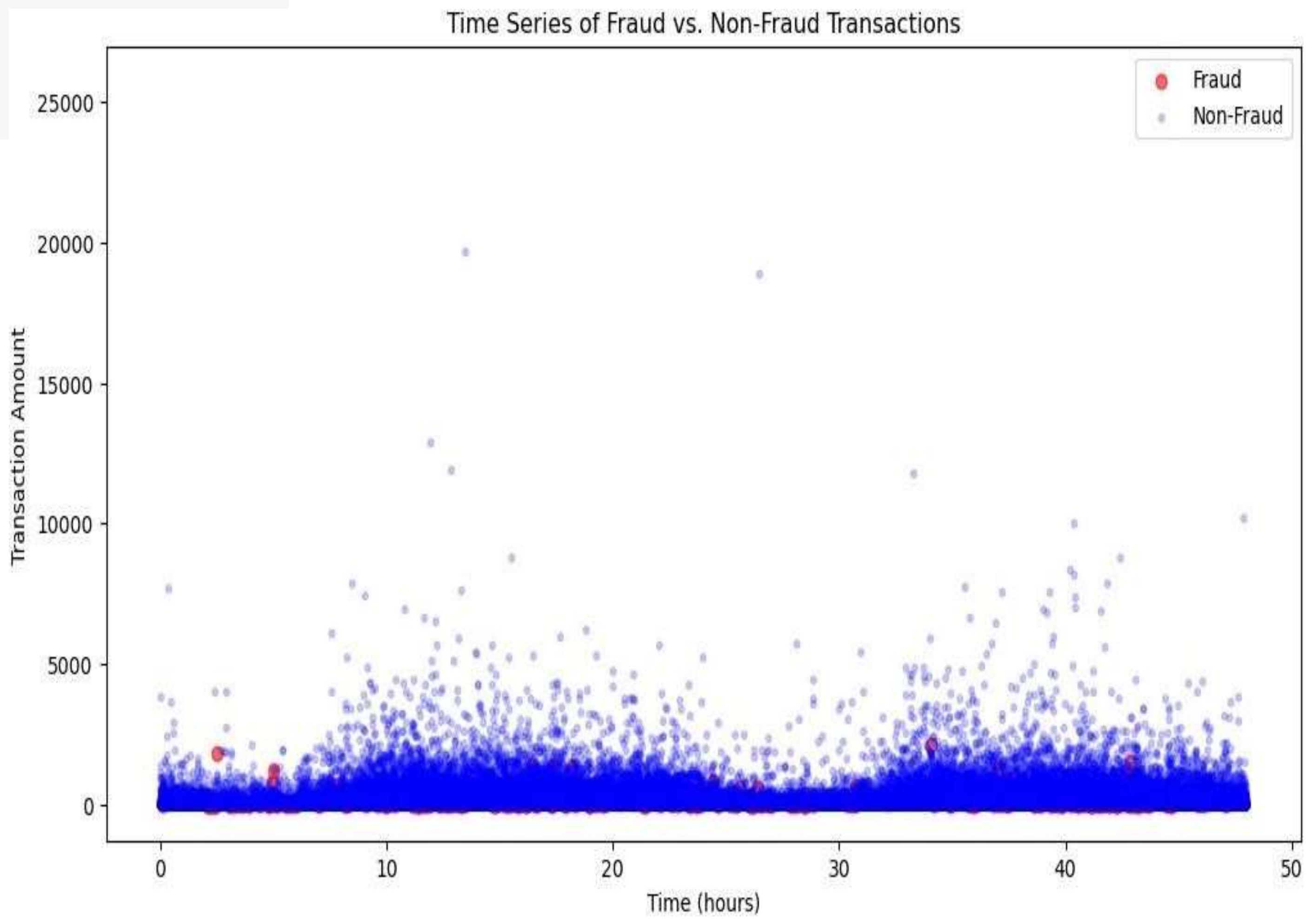
```
[34]: from sklearn.ensemble import RandomForestClassifier
      model = RandomForestClassifier(n_jobs=-1)
      #training
      model.fit(x_smote, y_smote)
      # testing
# Determine number of fraud cases in dataset

df['Time'] = df['Time'] / 3600 fraud = df[df['Class']
== 1] valid = df[df['Class'] == 0]

outlierFraction = len(fraud)/float(len(valid))

print(outlierFraction) print('Fraud Cases:
  {}'.format(len(df[df['Class'] == 1])))

   print('Valid Transactions:
  {}'.format(len(df[df['Class'] == 0])))
```

[49]:

```
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

1.1 visualization

```
[50]: plt.figure(figsize=(12, 6))
      plt.scatter(fraud['Time'], fraud['Amount'],
      color='red',␣marker='o',label='Fraud',
      alpha=0.6) plt.scatter(valid['Time'],
      valid['Amount'], color='blue',
      marker='.',label='Non-Fraud', alpha=0.2)
      plt.title('Time Series of Fraud vs. Non-Fraud
 Transactions')
      plt.xlabel('Time (hours)')
      plt.ylabel('Transaction Amount')
      plt.legend(loc='upper right')
      plt.show()
```



```
# dividing the X and the Y from the dataset
X = df.drop(['Class'], axis = 1)
Y = df["Class"]
print(X.shape)
print(Y.shape)
# getting just the values for the sake of processing
# (its a numpy array with no columns)
xData = X.values
yData = Y.values
```

```
(284807,)
```

ACCURACY

```
n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier ")
acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))
```

```
The model used is Random Forest classifier
The accuracy is 0.9994908886626171
So the Accuracy of the Model  is 99.94
1
```

CONCLUSION:

- WE INVESTIGATED THE DATA, CHECKED FOR DATA UNBALANCING, VISUALIZED, AND UNDERSTOOD THE RELATIONSHIP BETWEEN DIFFERENT FEATURES. WE THEN USED FOUR PREDICTIVE MODELS TO PERFORM VALIDATION BY SPLITTING DATASET INTO 3 PARTS, A TRAIN SET, A VALIDATION SET AND A TEST SET. FOR THE FIRST TWO MODELS, WE ONLY USED THE TRAIN AND TEST SET.

THANK YOU