ads-phase4

October 25, 2023

[18]:
```python
# import the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from matplotlib import gridspec
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, roc_auc_score, auc
```

1 Loading the Data

[2]:
```python
# Load the dataset from the csv file using pandas
# best way is to mount the drive on colab and
# copy the path for the csv file
data = pd.read_csv("creditcard.csv")
```

[3]:
```python
# Grab a peek at the data
data.head()
```

[3]:
```
   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]
```

2 Exploitary Data Analysis

[4]:
```python
# Print the shape of the data
# data = data.sample(frac = 0.1, random_state = 48)
print(data.shape)
print(data.describe())
```

```
(284807, 31)
                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]
```

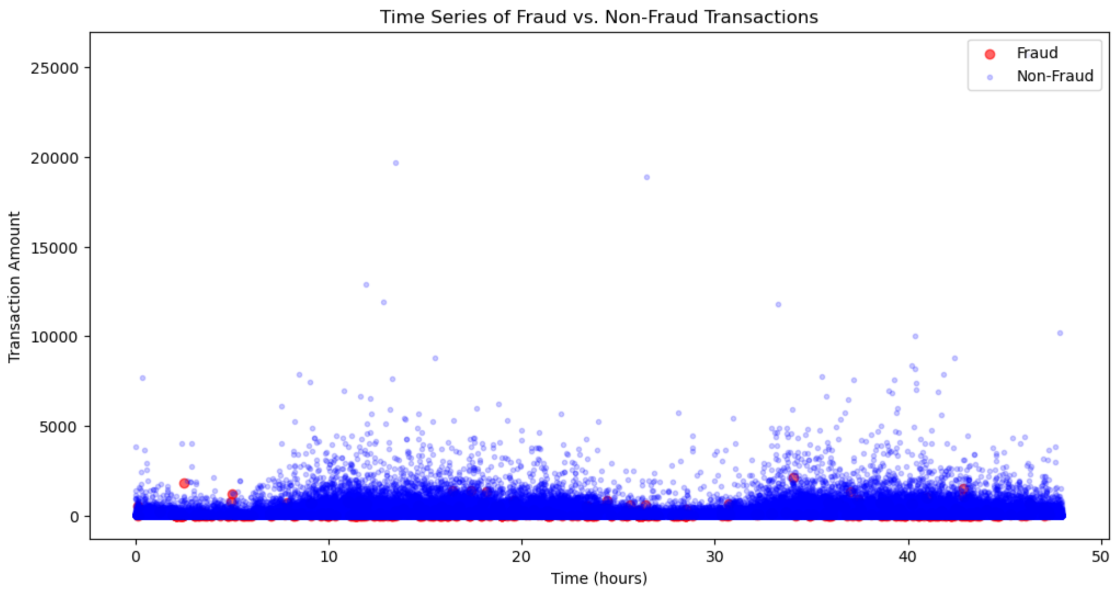
[5]:
```
# Determine number of fraud cases in dataset
data['Time'] = data['Time'] / 3600
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

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

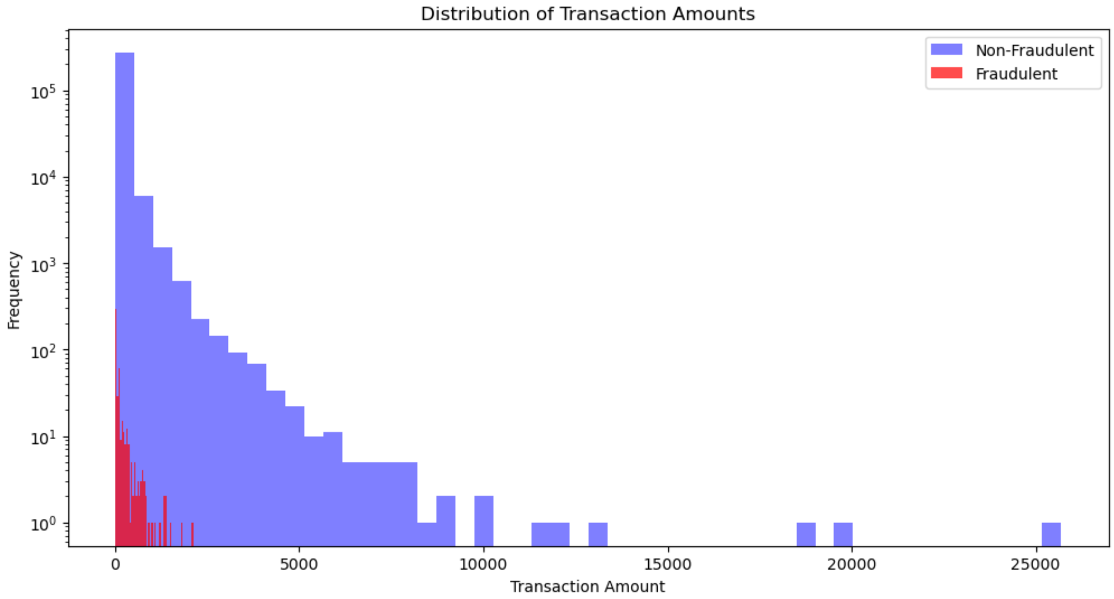
3 Visualization

[6]:
```
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)')
```

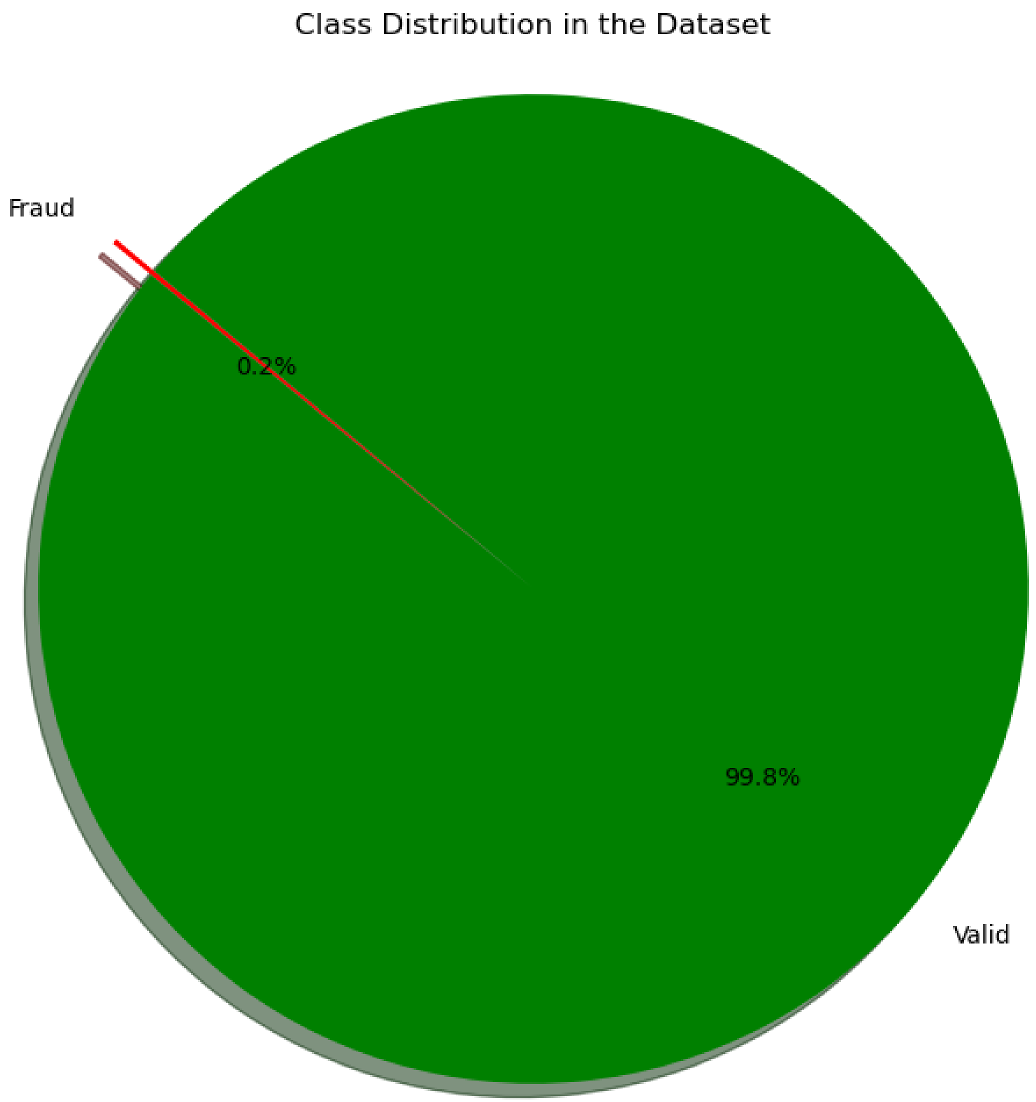
```
plt.ylabel('Transaction Amount')
plt.legend(loc='upper right')
plt.show()
```



[7]:
```
# Create a figure with subplots
plt.figure(figsize=(12, 6))

# Plot histograms for transaction amounts
plt.hist(valid['Amount'], bins=50, alpha=0.5, label='Non-Fraudulent',␣
 ↪color='blue')
plt.hist(fraud['Amount'], bins=50, alpha=0.7, label='Fraudulent', color='red')

plt.title('Distribution of Transaction Amounts')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.yscale('log')  # Use a logarithmic scale on the y-axis for better␣
 ↪visualization

plt.show()
```



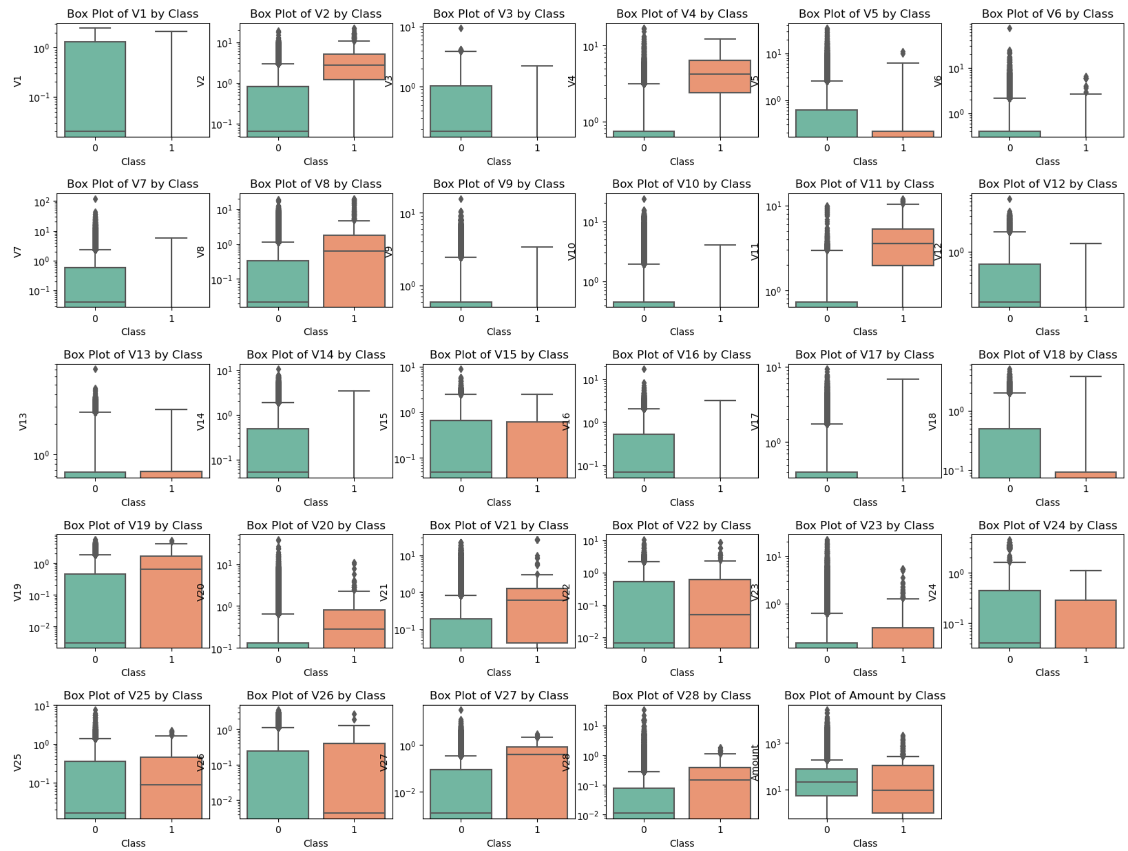
[8]:
```python
import matplotlib.pyplot as plt

# Count the number of fraud and valid transactions
fraud_count = len(data[data['Class'] == 1])
valid_count = len(data[data['Class'] == 0])

# Create data for the pie chart
labels = 'Fraud', 'Valid'
sizes = [fraud_count, valid_count]
colors = ['red', 'green']
explode = (0.1, 0)  # Explode the 'Fraud' slice for emphasis (optional)

# Create the pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.
↪1f%%', shadow=True, startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that the pie chart is circular.

# Add a title
plt.title("Class Distribution in the Dataset")

# Display the pie chart
plt.show()
```



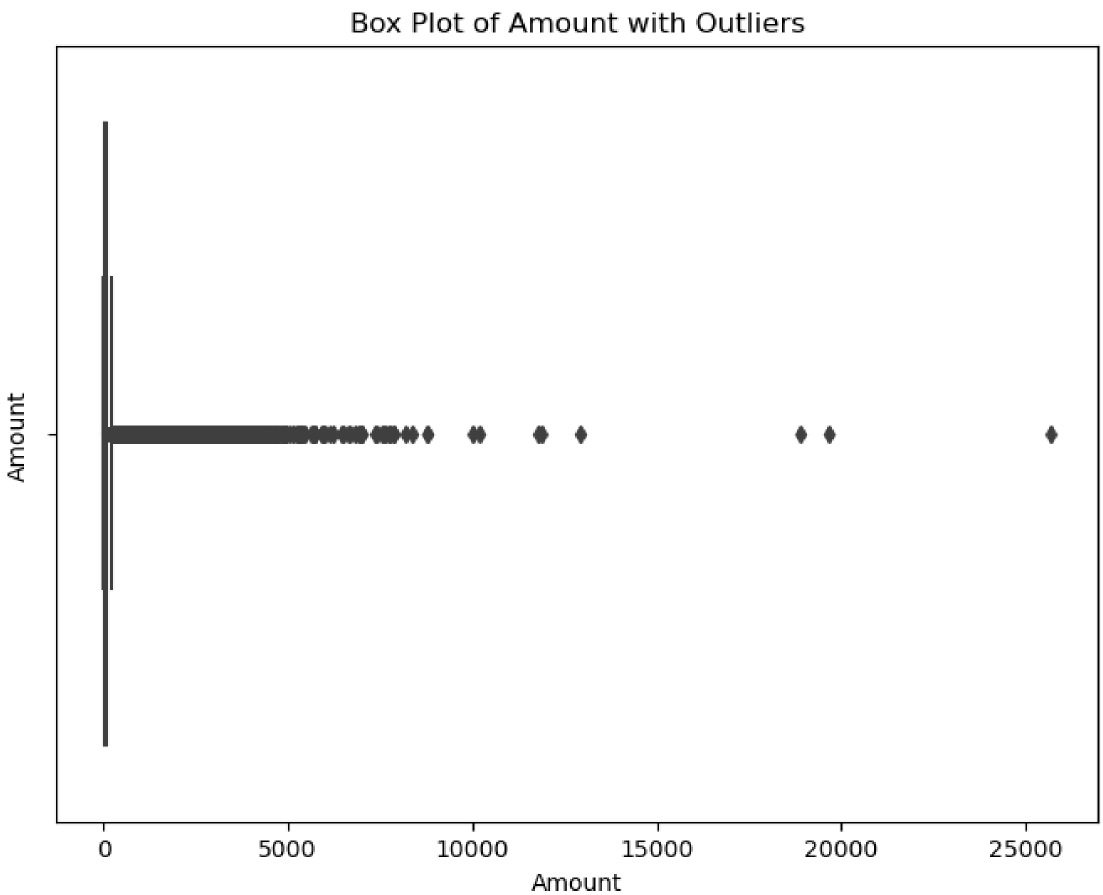
From the above information we came to know that dataset is imbalanced

[9]:
```
# Create a figure with subplots
plt.figure(figsize=(20, 15))
plt.subplots_adjust(hspace=0.5)

# List of features you want to create box plots for
features = ['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']  # Add more features as needed
```

```
# Create box plots for each feature
for i, feature in enumerate(features):
    plt.subplot(5, 6, i + 1)
    sns.boxplot(x='Class', y=feature, data=data, palette="Set2")
    plt.title(f'Box Plot of {feature} by Class')
    plt.yscale('log')  # Use a logarithmic scale on the y-axis for better visualization

plt.show()
```



[10]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a boxplot for the 'Amount' column with outliers displayed
plt.figure(figsize=(8, 6))
sns.boxplot(x=data['Amount'], showfliers=True)
plt.title('Box Plot of Amount with Outliers')
plt.ylabel('Amount')
plt.show()
```



[11]:
```
#Fraudulant transaction
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

```
Amount details of the fraudulent transaction
```

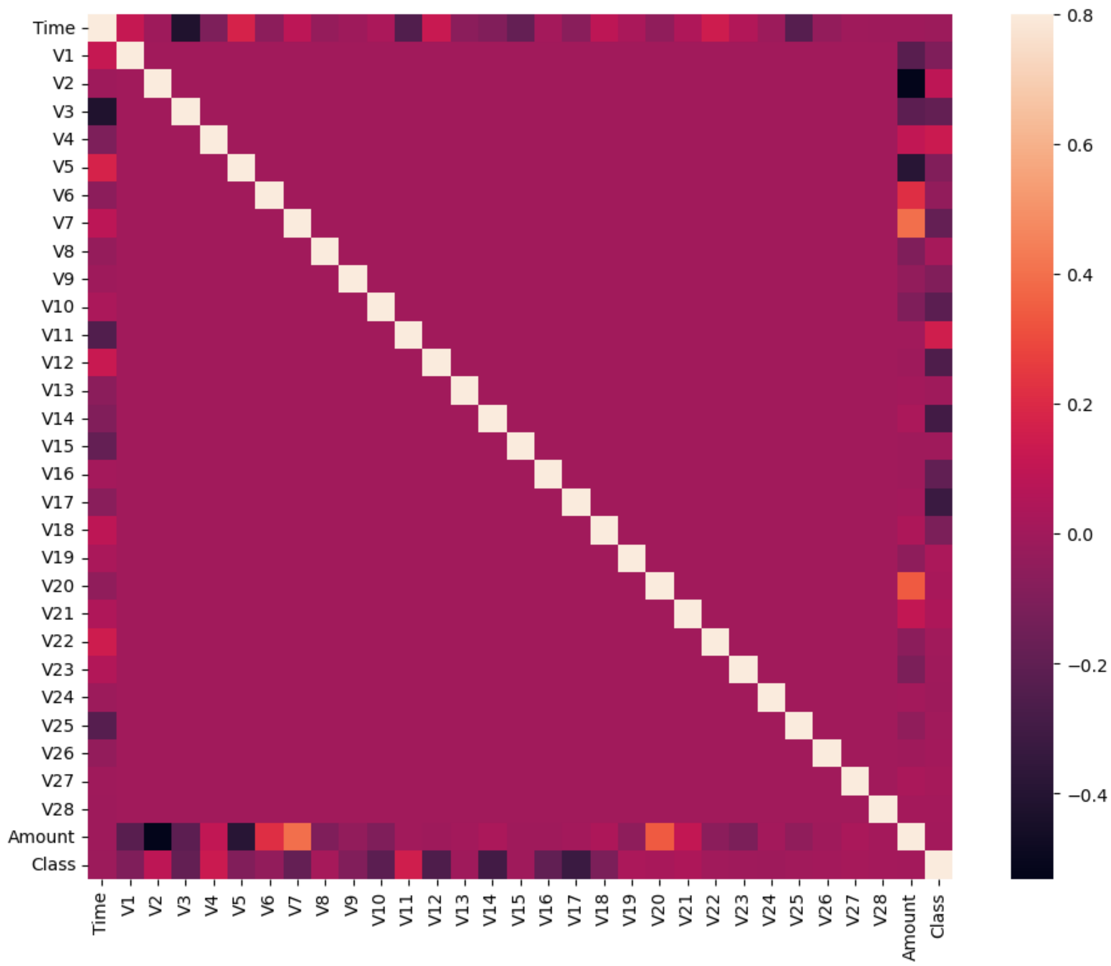
[11]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

[12]:
```
#Normal Transaction
print("details of valid transaction")
valid.Amount.describe()
```

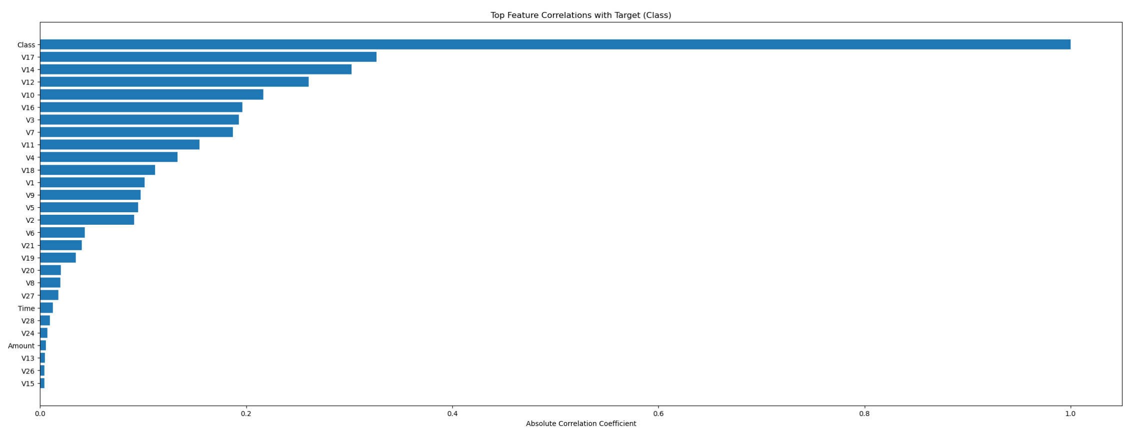
details of valid transaction

[12]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

[13]:
```
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



[14]:
```
# Calculate the absolute Pearson correlation coefficient between features and␣
↪the target variable
correlations = data.corrwith(data['Class']).abs()

# Sort feature importances in descending order
sorted_correlations = correlations.sort_values(ascending=False)

# Get the top n important features (e.g., top 10)
n = 28
top_correlations = sorted_correlations[:n]
top_feature_names = top_correlations.index

# Create a bar chart to visualize feature importance
plt.figure(figsize=(28, 10))
plt.barh(range(n), top_correlations, align='center')
plt.yticks(range(n), top_feature_names)
plt.xlabel('Absolute Correlation Coefficient')
plt.title('Top Feature Correlations with Target (Class)')
plt.gca().invert_yaxis()  # Invert the y-axis for better visualization
plt.show()
```



[16]:
```
# dividing the X and the Y from the dataset
X = data.drop(['Class'], axis = 1)
Y = data["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, 30)
```

```
(284807,)
```

4 Feature Engineering

Standardizing Features

[28]:
```python
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Principal Component Analysis(PCA)

[31]:
```python
# Dimensionality reduction (PCA)
pca = PCA(n_components=10)
X_pca = pca.fit_transform(X_scaled)

# Create a new DataFrame with the engineered features
X_engineered = pd.DataFrame(X_pca, columns=[f'PC{i}' for i in range(1, 11)])
```

[30]:
```python
# Split the data into training and testing sets
XTrain, XTest, YTrain, Test = train_test_split(X_engineered, Y, test_size=0.2,␣
↪random_state=42)
```

5 Model Building and Training

[33]:
```python
# Building the Random Forest Classifier (RANDOM FOREST)
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(XTrain, YTrain)
# predictions
yPred = rfc.predict(XTest)
```

6 Accuracy of the model

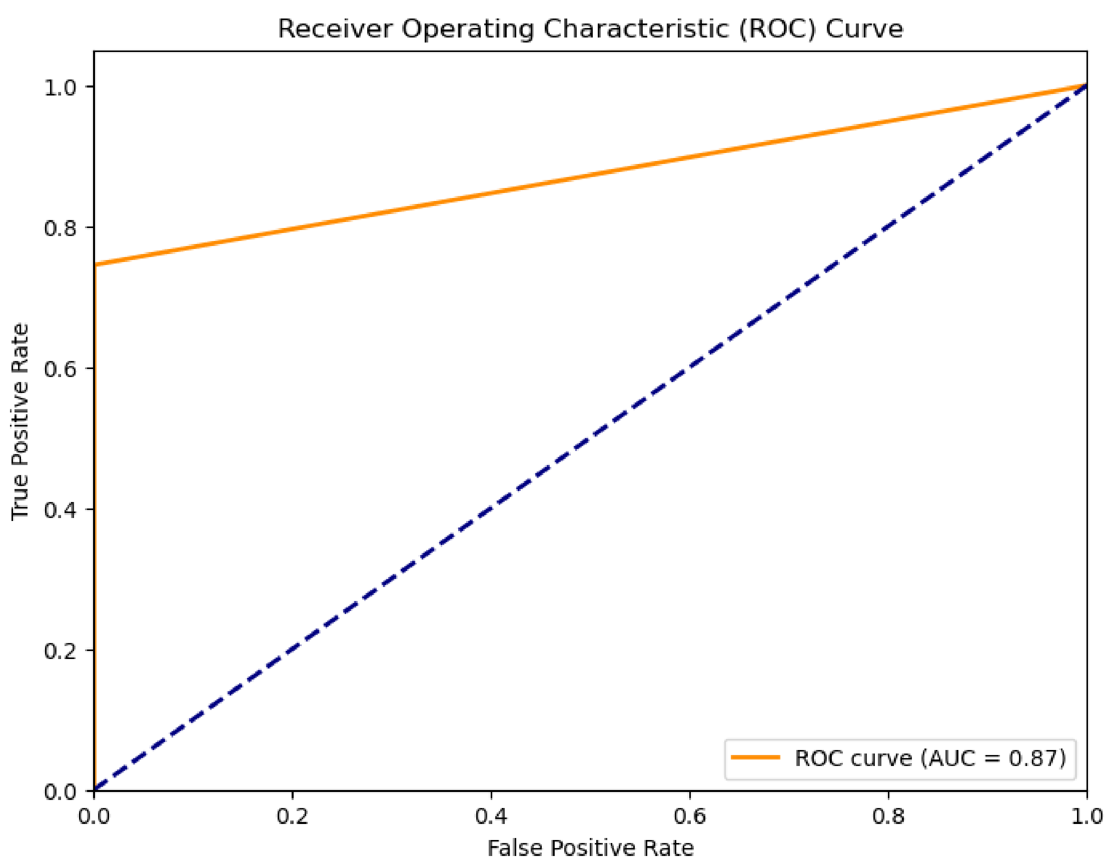
[34]:
```python
n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")

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%

[35]:

```python
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(yTest, yPred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.
 ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Output the AUC score
print('AUC (Area Under the Curve): {:.2f}'.format(roc_auc))
```



```
AUC (Area Under the Curve): 0.87
```

We are getting 0.87 out of 1, which is pretty good model accuracy

[ ]:

```

```