In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

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

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
0	51435.0	1.197490	-0.352125	-0.135904	0.222100	0.231128	1.086617	-0.420363	0.391464	0.672499	...	-0.337999
1	78049.0	0.976047	-0.289947	1.465321	1.300002	-1.382887	-0.479586	-0.632572	0.064533	0.710743	...	0.322829
2	157168.0	-1.395302	0.478266	-0.584911	-1.201527	0.928544	-0.743618	0.755504	-0.141397	-2.118499	...	0.282803
3	69297.0	1.276014	-0.672705	-0.425494	-0.777398	-0.582088	-0.880396	-0.103505	-0.203036	-1.241653	...	0.256003
4	144504.0	-0.312745	-1.202565	2.249806	-0.297210	-0.963389	1.207532	-0.837776	-0.057654	1.121421	...	-0.274386

5 rows × 31 columns

In [3]:

```
data.shape
```

Out[3]:

```
(186000, 31)
```

In [4]:

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 186000 entries, 0 to 185999
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    186000 non-null  float64
 1   V1      186000 non-null  float64
 2   V2      186000 non-null  float64
 3   V3      186000 non-null  float64
 4   V4      186000 non-null  float64
 5   V5      186000 non-null  float64
 6   V6      186000 non-null  float64
 7   V7      186000 non-null  float64
 8   V8      186000 non-null  float64
```

```
 9   V9      186000 non-null  float64
 10  V10     186000 non-null  float64
 11  V11     186000 non-null  float64
 12  V12     186000 non-null  float64
 13  V13     186000 non-null  float64
 14  V14     186000 non-null  float64
 15  V15     186000 non-null  float64
 16  V16     186000 non-null  float64
 17  V17     186000 non-null  float64
 18  V18     186000 non-null  float64
 19  V19     186000 non-null  float64
 20  V20     186000 non-null  float64
 21  V21     186000 non-null  float64
 22  V22     186000 non-null  float64
 23  V23     186000 non-null  float64
 24  V24     186000 non-null  float64
 25  V25     186000 non-null  float64
 26  V26     186000 non-null  float64
 27  V27     186000 non-null  float64
 28  V28     186000 non-null  float64
 29  Amount  186000 non-null  float64
 30  Class   186000 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 44.0 MB
```

In [5]:

```
data.describe()
```

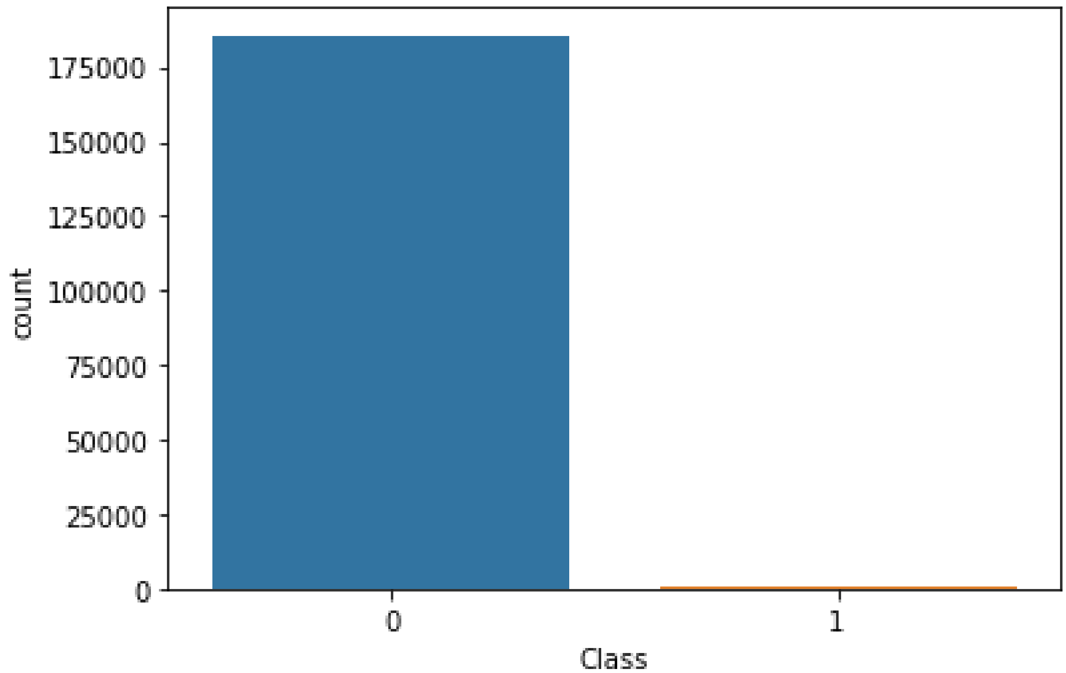
Out[5]:

	Time	V1	V2	V3	V4	V5	
count	186000.000000	186000.000000	186000.000000	186000.000000	186000.000000	186000.000000	186000.00
mean	94829.405134	-0.003211	0.001000	0.000697	-0.000765	0.001546	0.000522
std	47498.820375	1.960298	1.658303	1.510441	1.412797	1.401870	1.342198
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.16050
25%	54169.000000	-0.921761	-0.594771	-0.888268	-0.846676	-0.690550	-0.767943
50%	84669.500000	0.012918	0.069063	0.179846	-0.023288	-0.053598	-0.274187
75%	139336.000000	1.313242	0.807190	1.028361	0.740716	0.615143	0.396718
max	172792.000000	2.454930	19.167239	9.382558	16.875344	34.801666	73.30162

8 rows × 31 columns

In [6]:

```
sns.countplot(x='Class', data=data)
print("Fraud: ",data.Class.sum()/data.Class.count())
Fraud:  0.001586021505376344
```

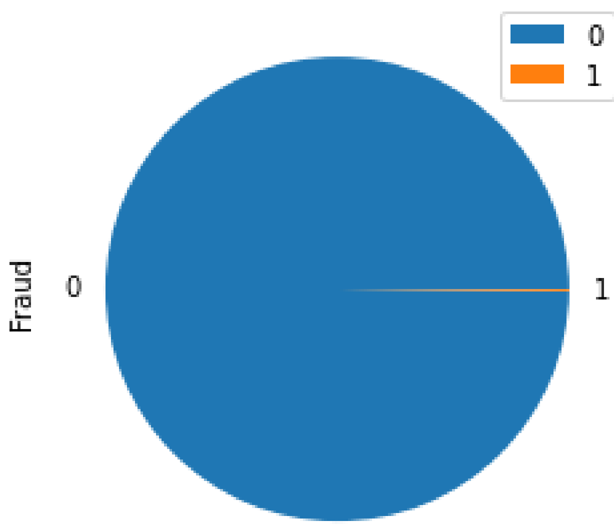


In [7]:

```
Fraud_class = pd.DataFrame({'Fraud': data['Class']})
Fraud_class.apply(pd.value_counts).plot(kind='pie',subplots=True)
```

Out[7]:

```
array([<AxesSubplot:ylabel='Fraud'>], dtype=object)
```



In [8]:

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

In [9]:

```
fraud.Amount.describe()
```

Out[9]:

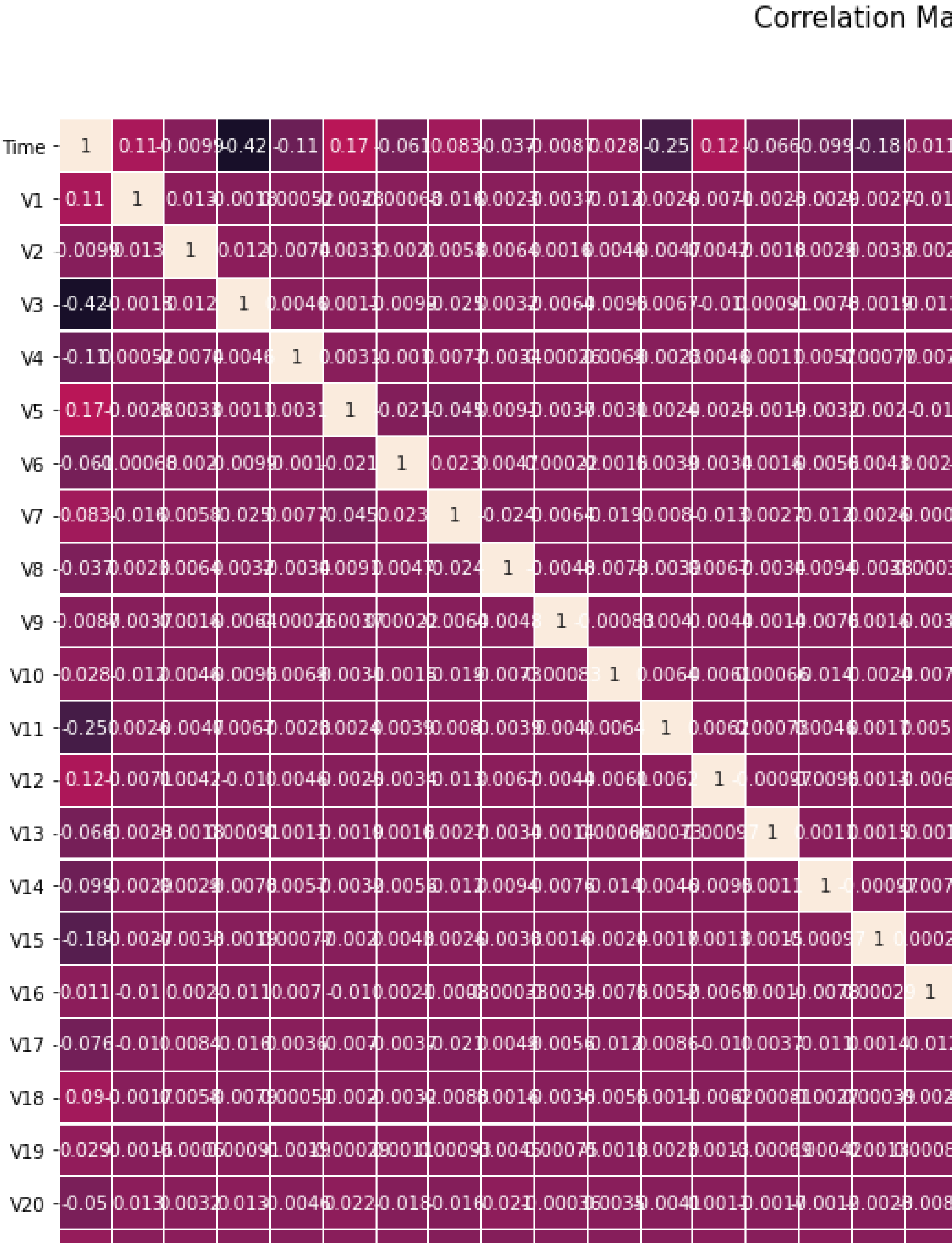
```
count     295.000000
mean      126.596983
std       251.386390
min         0.000000
25%         1.000000
50%        14.460000
75%       112.015000
max      1809.680000
Name: Amount, dtype: float64
```

In [12]:

```
plt.figure(figsize=(20,20))
plt.title('Correlation Matrix', y=1.05, size=15)
sns.heatmap(data.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, linecolor='white', annot=True)
```

Out[12]:

```
<AxesSubplot:title={'center':'Correlation Matrix'}>
```


Correlation Ma
Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20

In [13]:

```
from sklearn.preprocessing import RobustScaler
rs = RobustScaler()

data['Amount'] = rs.fit_transform(data['Amount'].values.reshape(-1, 1))
data['Time'] = rs.fit_transform(data['Time'].values.reshape(-1, 1))
```

In [14]:

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

In [15]:

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 1)
```

In [16]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report, accuracy_score,
precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, precision_recall_curve,
roc_auc_score
```

In [17]:

```
def evaluate(Y_pred, Y_pred_prob):
    print("Accuracy: ",accuracy_score(Y_test, Y_pred))
    print("Precision: ",precision_score(Y_test, Y_pred))
    print("Recall: ",recall_score(Y_test, Y_pred))
    print("F1-Score: ",f1_score(Y_test, Y_pred))
    print("AUC score: ",roc_auc_score(Y_test, Y_pred))

    print(classification_report(Y_test, Y_pred, target_names = ['Normal',
'Fraud']))

    conf_matrix = confusion_matrix(Y_test, Y_pred)
    plt.figure(figsize =(6, 6))
    sns.heatmap(conf_matrix, xticklabels = ['Normal', 'Fraud'],
            yticklabels = ['Normal', 'Fraud'], annot = True, fmt ="d");
    plt.title("Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()


    p, r, t = precision_recall_curve(Y_test, Y_pred_prob)
```

```
    plt.plot(p, r)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision Recall Curve')
```

In [18]:

```
#logistic regression

lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_pred_lr_i = lr.predict(X_test)
```

In [19]:

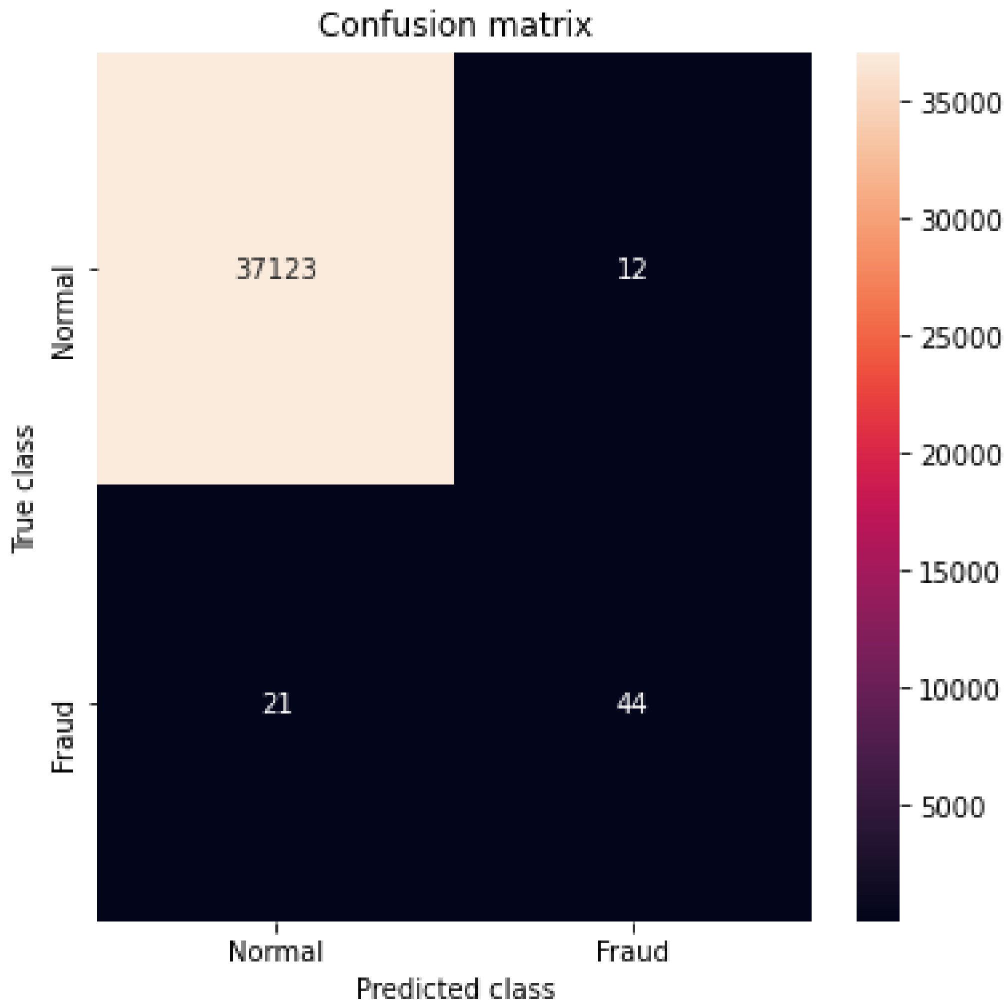
```
Y_pred_prob_lr_i = lr.predict_proba(X_test)[:,1]
```

In [20]:

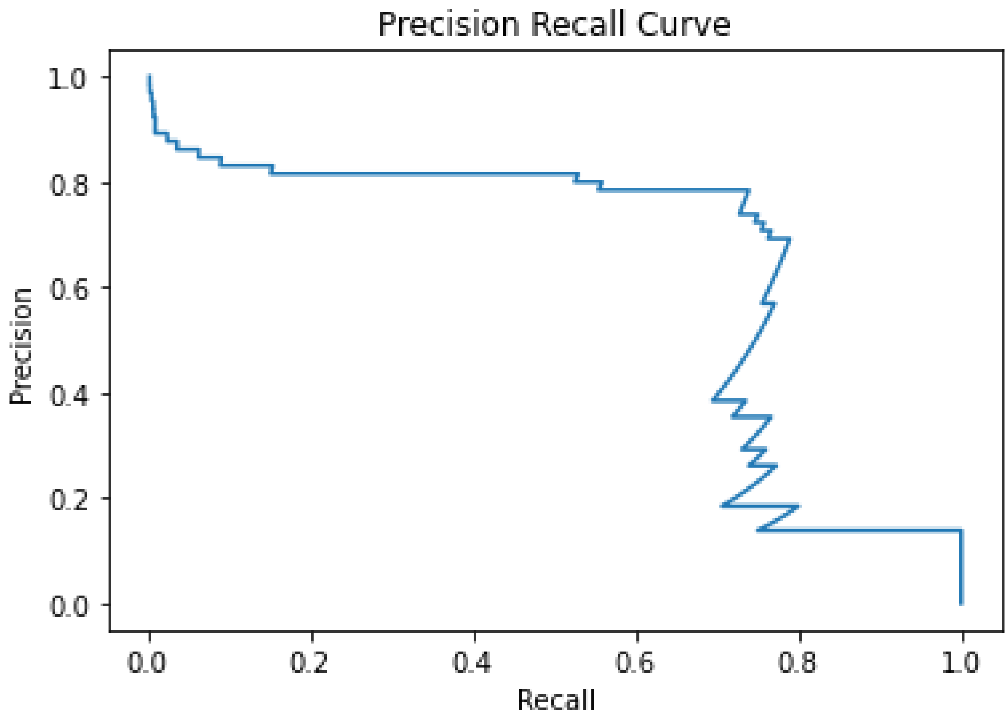
```
evaluate(Y_pred_lr_i, Y_pred_prob_lr_i)
Accuracy:  0.9991129032258065
Precision:  0.7857142857142857
Recall:  0.676923076923077
F1-Score:  0.7272727272727272
AUC score:  0.8382999658211723
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37135
       Fraud       0.79      0.68      0.73        65

    accuracy                           1.00     37200
   macro avg       0.89      0.84      0.86     37200
weighted avg       1.00      1.00      1.00     37200
```



Confusion matrix
37123
12
21
44
Normal
Fraud
True class
Normal
Fraud
Predicted class
35000
30000
25000
20000
15000
10000
5000



In [21]:

```
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(X_train, Y_train)
# predictions
Y_pred_rf_i = rfc.predict(X_test)
```

In [22]:

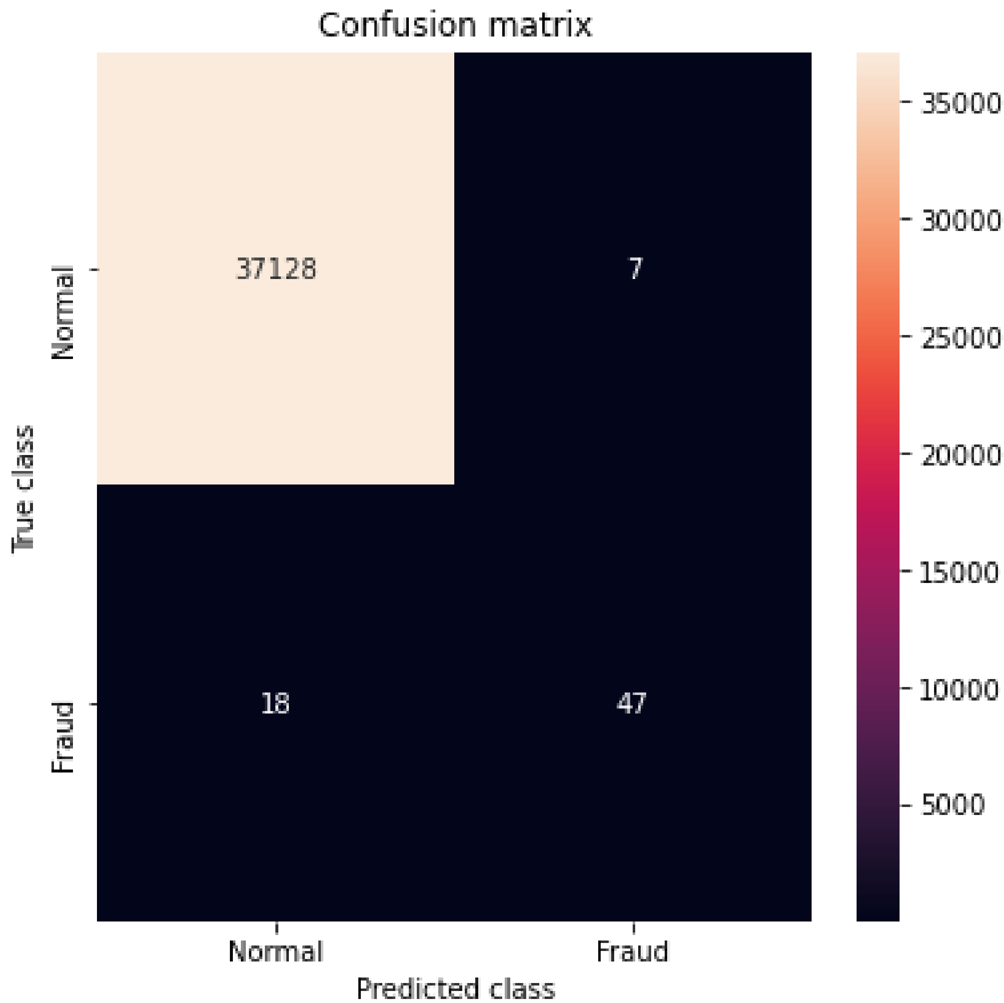
```
Y_pred_prob_rf_i = rfc.predict_proba(X_test)[:,1]
```

In [23]:

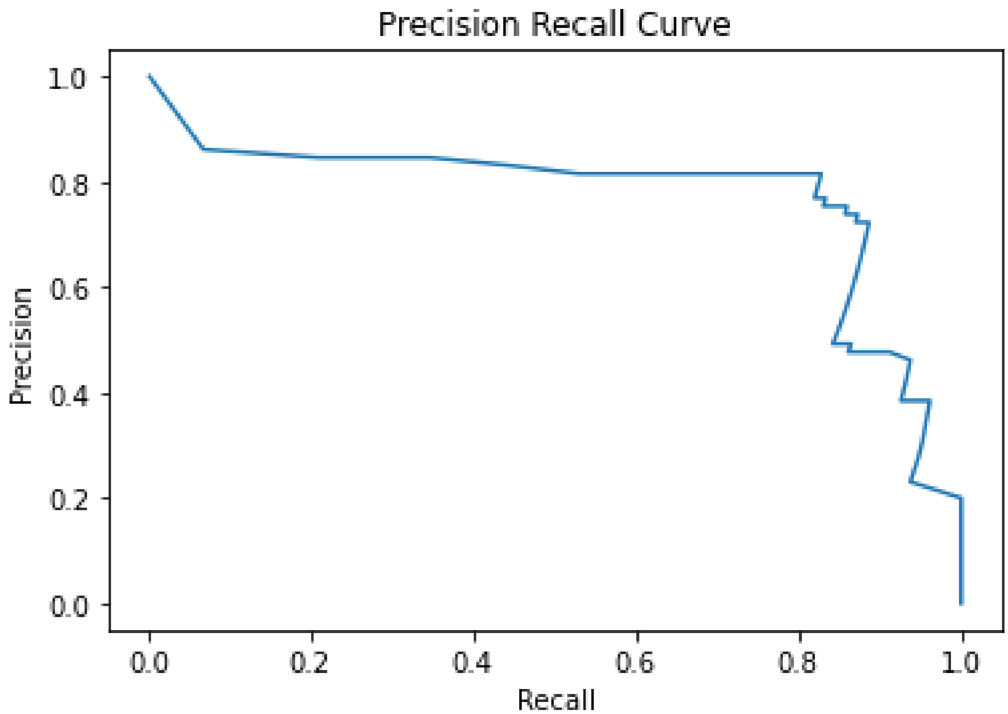
```
evaluate(Y_pred_rf_i, Y_pred_prob_rf_i)
Accuracy:  0.9993279569892473
Precision:  0.8703703703703703
Recall:  0.7230769230769231
F1-Score:  0.7899159663865546
AUC score:  0.8614442108315813
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37135
       Fraud       0.87      0.72      0.79        65

    accuracy                           1.00     37200
   macro avg       0.93      0.86      0.89     37200
weighted avg       1.00      1.00      1.00     37200
```



Confusion matrix
37128
7
18
47
Normal
Fraud
True class
Normal
Fraud
Predicted class
35000
30000
25000
20000
15000
10000
5000



In [24]:

```
# decision tree model creation
dtc = DecisionTreeClassifier()
dtc.fit(X_train, Y_train)
# predictions
Y_pred_dt_i = dtc.predict(X_test)
```

In [25]:

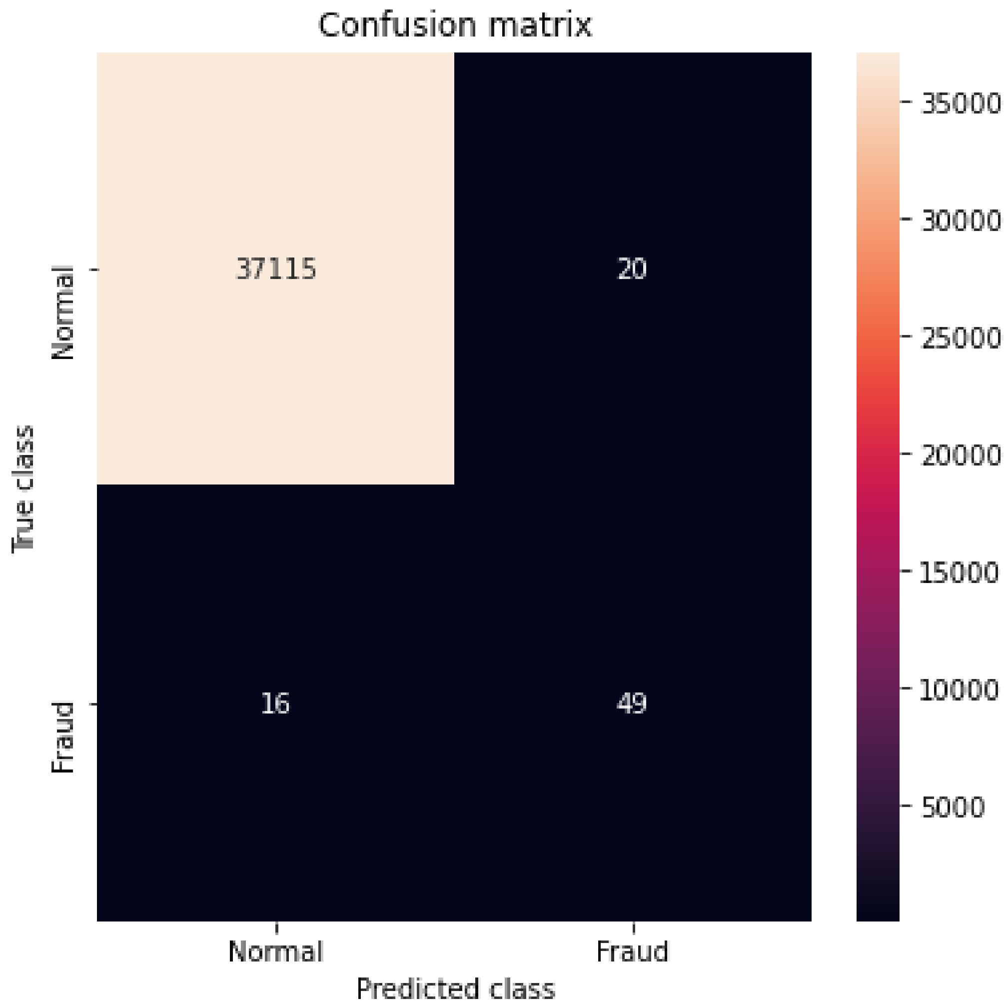
```
Y_pred_prob_dt_i = dtc.predict_proba(X_test)[:,1]
```

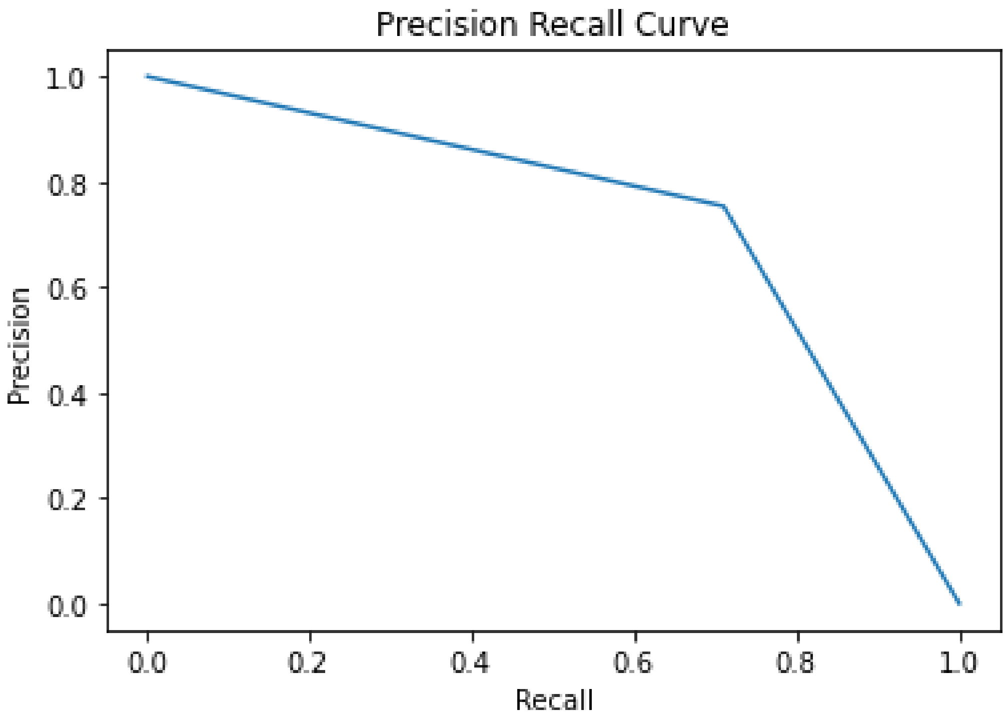
In [26]:

```
evaluate(Y_pred_dt_i, Y_pred_prob_dt_i)
Accuracy:  0.9990322580645161
Precision:  0.7101449275362319
Recall:  0.7538461538461538
F1-Score:  0.7313432835820897
AUC score:  0.8766537891891332
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37135
       Fraud       0.71      0.75      0.73        65

    accuracy                           1.00     37200
   macro avg       0.85      0.88      0.87     37200
weighted avg       1.00      1.00      1.00     37200
```


Confusion matrix
True class
Normal
Fraud
Predicted class
Normal
Fraud
37115
20
16
49
35000
30000
25000
20000
15000
10000
5000



In [27]:

```
#random forest balanced weights
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfb = RandomForestClassifier(class_weight='balanced')
rfb.fit(X_train, Y_train)
# predictions
Y_pred_rf_b = rfb.predict(X_test)
```

In [28]:

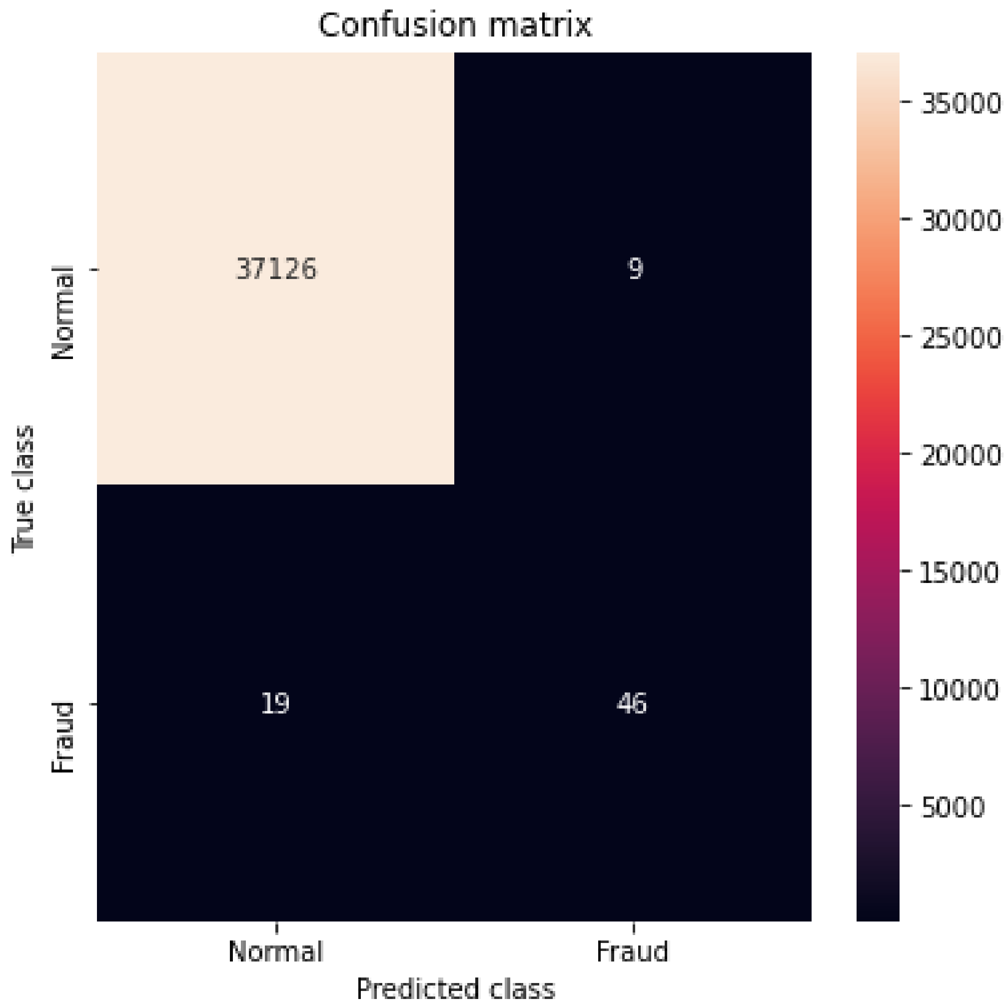
```
Y_pred_prob_rf_b = rfb.predict_proba(X_test)[:,1]
```

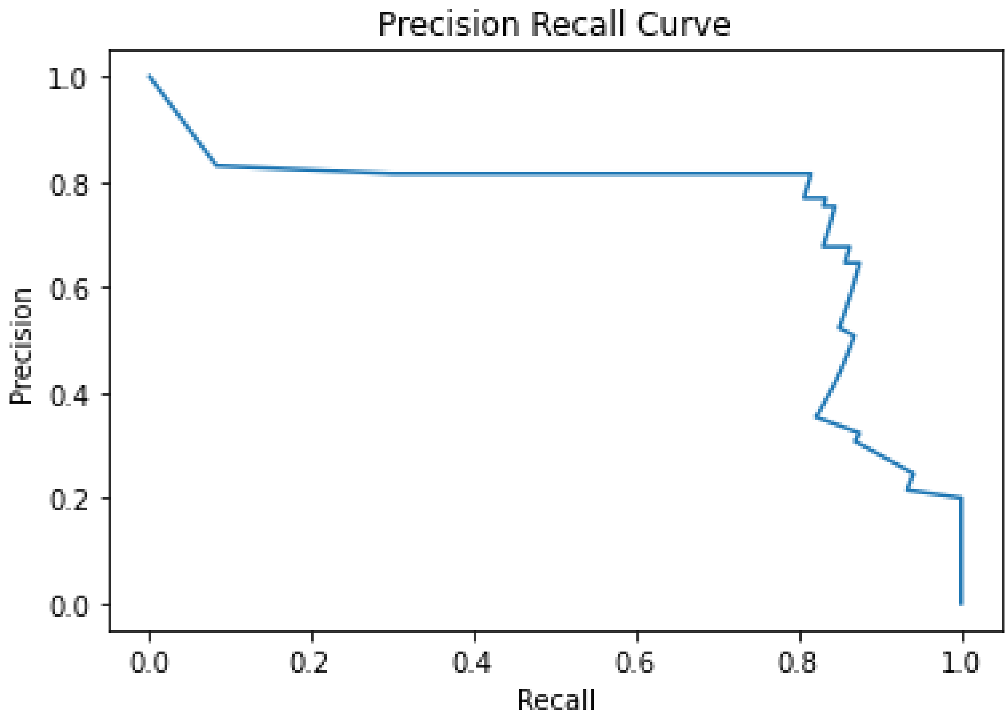
In [29]:

```
evaluate(Y_pred_rf_b, Y_pred_prob_rf_b)
Accuracy:  0.999247311827957
Precision:  0.8363636363636363
Recall:  0.7076923076923077
F1-Score:  0.7666666666666666
AUC score:  0.8537249743658792
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37135
       Fraud       0.84      0.71      0.77        65

    accuracy                           1.00     37200
   macro avg       0.92      0.85      0.88     37200
weighted avg       1.00      1.00      1.00     37200
```


Confusion matrix
37126
9
19
46
Normal
Fraud
True class
Normal
Fraud
Predicted class
35000
30000
25000
20000
15000
10000
5000



In [31]:

```
pip install -U imbalanced-learn
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.7.0-py3-none-any.whl (167 kB)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in
c:\users\revan\anaconda3\lib\site-packages (from imbalanced-learn) (1.19.2)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
c:\users\revan\anaconda3\lib\site-packages (from imbalanced-learn) (0.17.0)
Requirement already satisfied, skipping upgrade: scikit-learn>=0.23 in
c:\users\revan\anaconda3\lib\site-packages (from imbalanced-learn) (0.23.2)
Requirement already satisfied, skipping upgrade: scipy>=0.19.1 in
c:\users\revan\anaconda3\lib\site-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in
c:\users\revan\anaconda3\lib\site-packages (from scikit-learn>=0.23-
>imbalanced-learn) (2.1.0)
Installing collected packages: imbalanced-learn
Successfully installed imbalanced-learn-0.7.0
Note: you may need to restart the kernel to use updated packages.
```

In [32]:

```
from imblearn.over_sampling import SMOTE
```

In [33]:

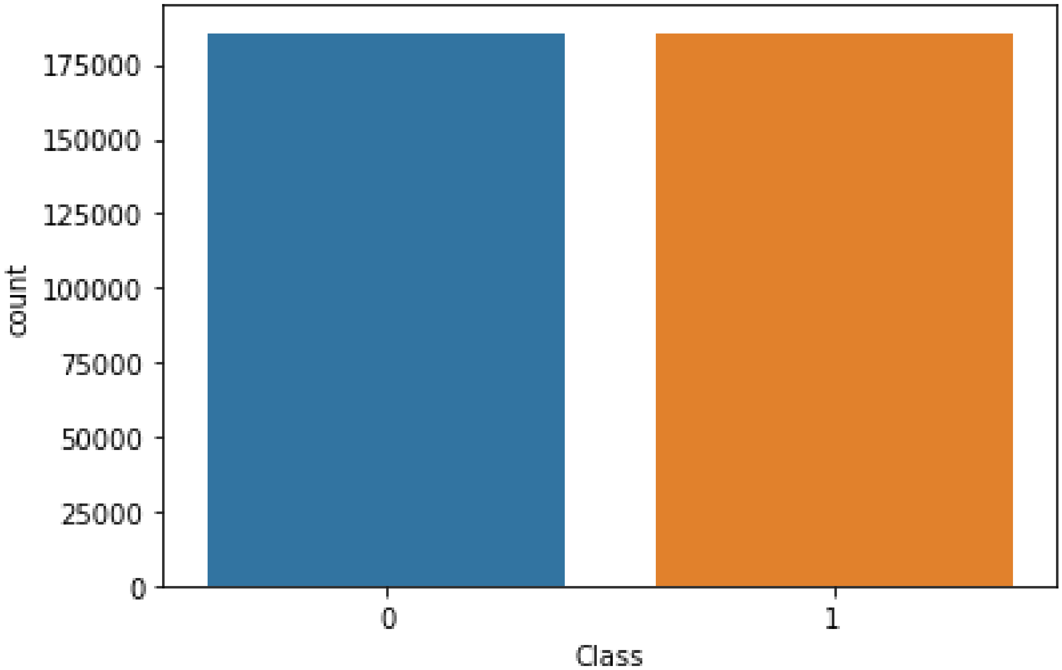
```
smote = SMOTE(random_state=56)
smote_X, smote_Y = smote.fit_resample(X, Y)
```

In [34]:

```
sns.countplot(smote_Y)
```

Out[34]:

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



In [35]:

```
X_train, X_test, Y_train, Y_test = train_test_split(smote_X, smote_Y,
test_size=0.2, random_state=1)
```

In [36]:

```
#using smote
lr_smote = LogisticRegression()
lr_smote.fit(X_train, Y_train)
Y_pred_lr_smote = lr_smote.predict(X_test)
```

In [37]:

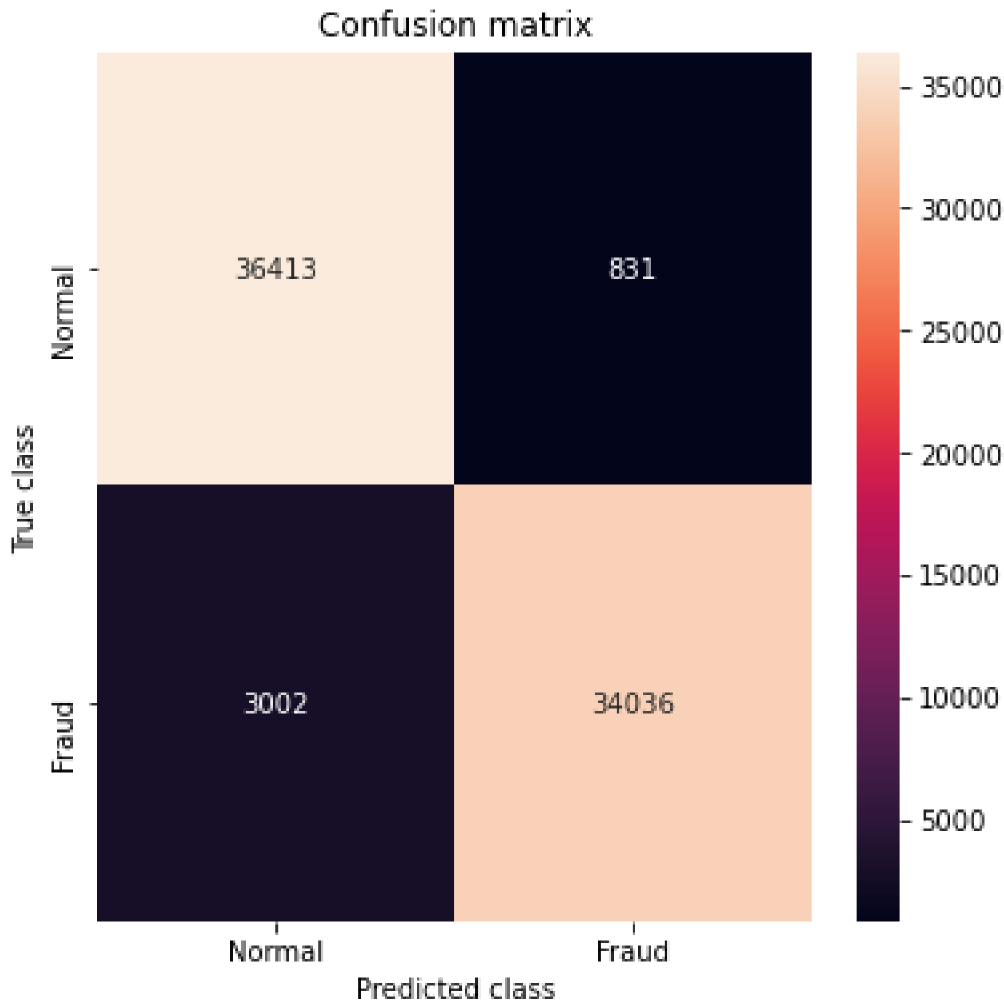
```
Y_pred_prob_lr_smote = lr_smote.predict_proba(X_test)[:,1]
```

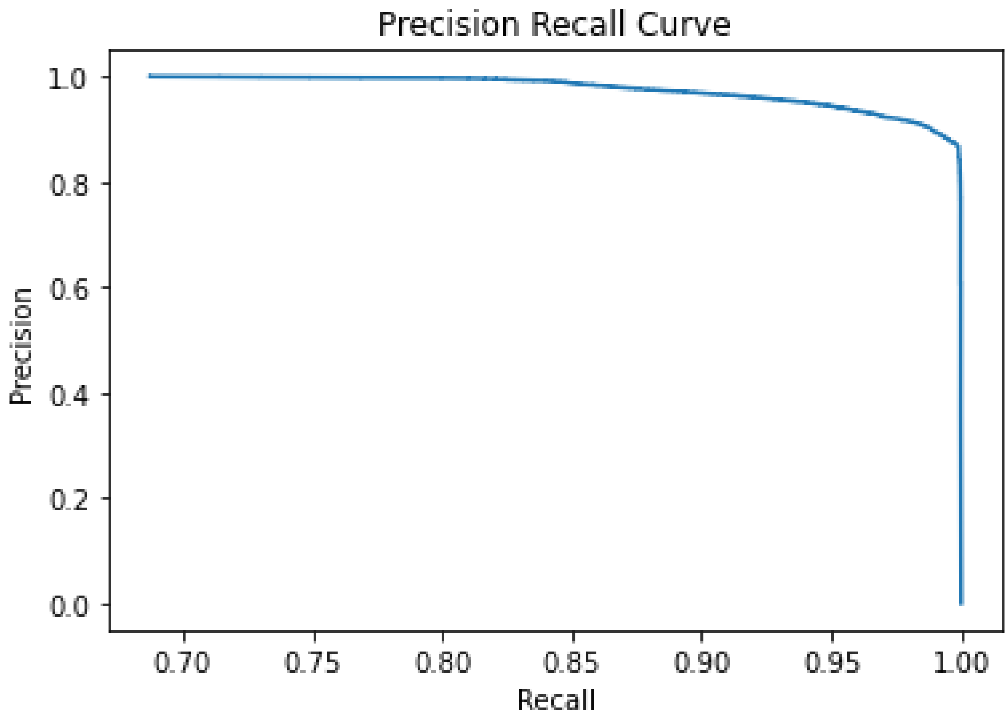
In [38]:

```
evaluate(Y_pred_lr_smote, Y_pred_prob_lr_smote)
Accuracy:  0.9483993430440752
Precision:  0.9761665758453552
Recall:  0.918948107349209
F1-Score:  0.9466935539948543
AUC score:  0.9483178942932277
              precision    recall  f1-score   support

      Normal       0.92      0.98      0.95     37244
       Fraud       0.98      0.92      0.95     37038

    accuracy                           0.95     74282
   macro avg       0.95      0.95      0.95     74282
weighted avg       0.95      0.95      0.95     74282
```


Confusion matrix
36413
831
3002
34036
Normal
Fraud
True class
Normal
Fraud
Predicted class
35000
30000
25000
20000
15000
10000
5000



In [39]:

```
#using randomforest
rf_smote = RandomForestClassifier()
rf_smote.fit(X_train, Y_train)
Y_pred_rf_smote = rf_smote.predict(X_test)
```

In [40]:

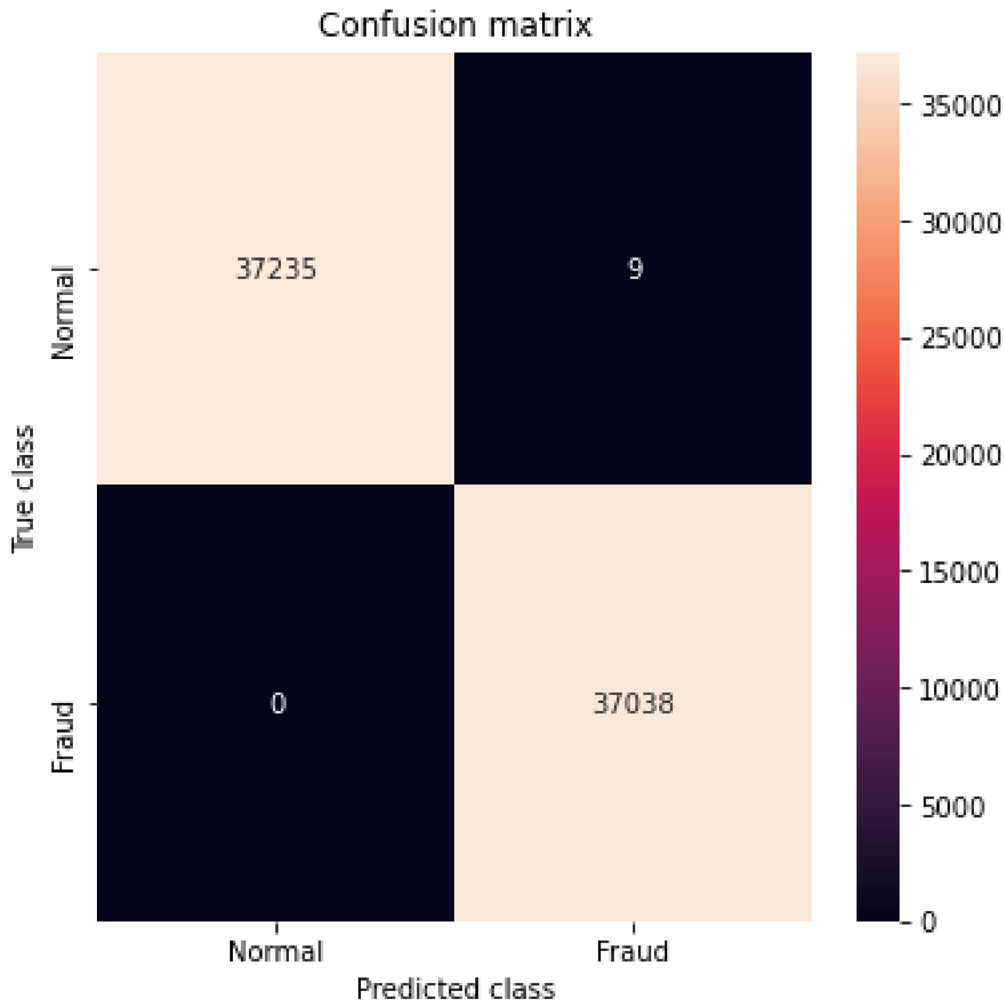
```
Y_pred_prob_rf_smote = rf_smote.predict_proba(X_test)[:,1]
```

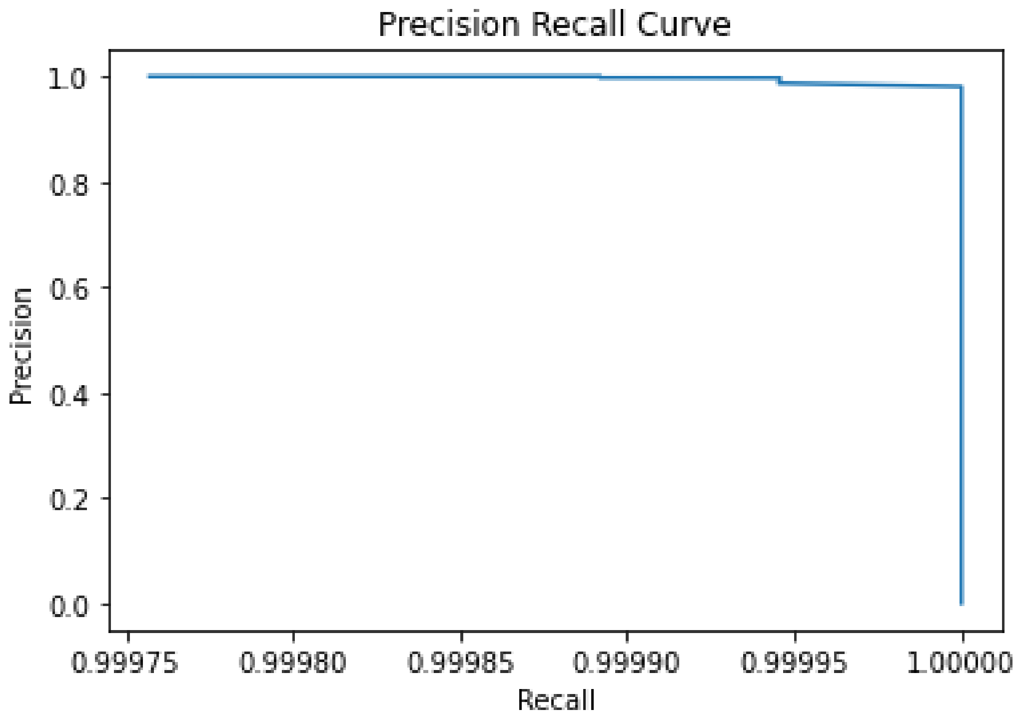
In [41]:

```
evaluate(Y_pred_rf_smote, Y_pred_prob_rf_smote)
Accuracy:  0.9998788400958509
Precision:  0.999757065349421
Recall:  1.0
F1-Score:  0.999878517918607
AUC score:  0.9998791751691547
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37244
       Fraud       1.00      1.00      1.00     37038

    accuracy                           1.00     74282
   macro avg       1.00      1.00      1.00     74282
weighted avg       1.00      1.00      1.00     74282
```


Confusion matrix
37235
9
0
37038
Normal
Fraud
True class
Predicted class
35000
30000
25000
20000
15000
10000
5000
0



In [43]:

```
#using decision tree
dt_smote = DecisionTreeClassifier()
dt_smote.fit(X_train, Y_train)
Y_pred_dt_smote = dt_smote.predict(X_test)
```

In [45]:

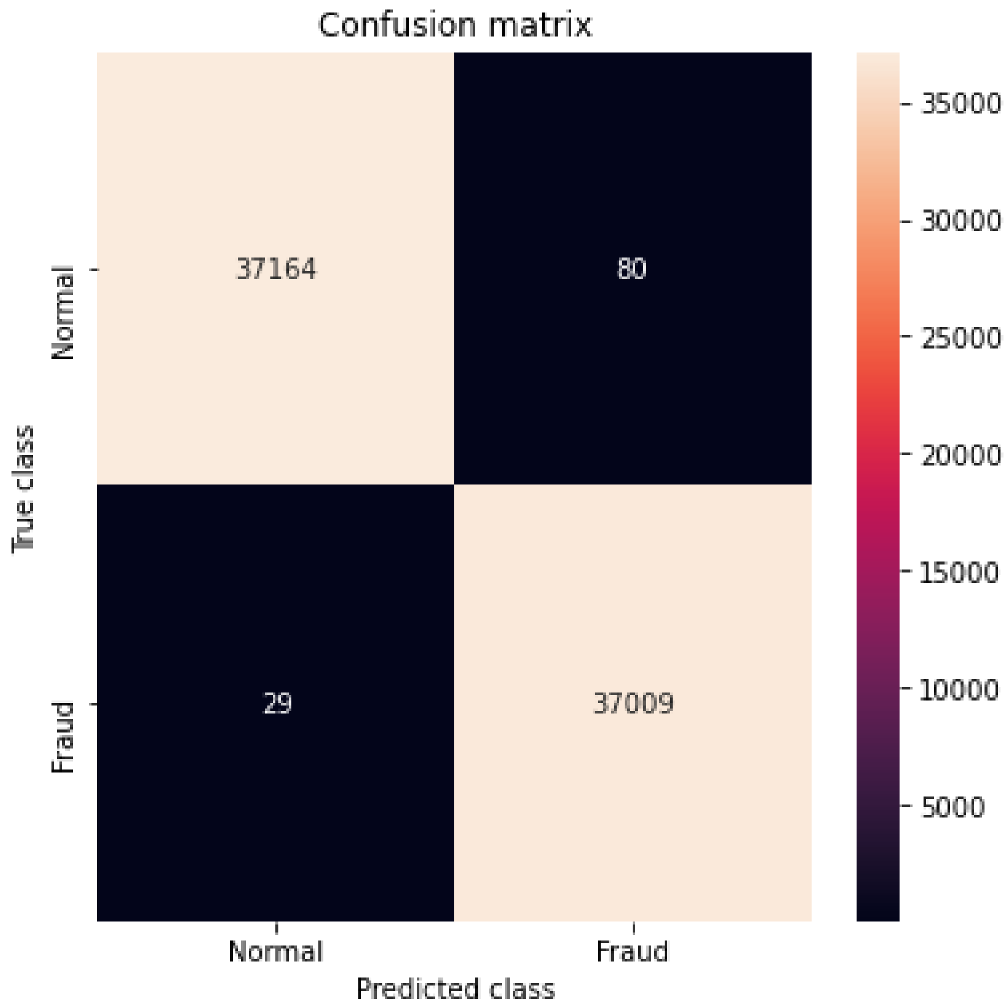
```
Y_pred_prob_dt_smote = dt_smote.predict_proba(X_test)[:,1]
```

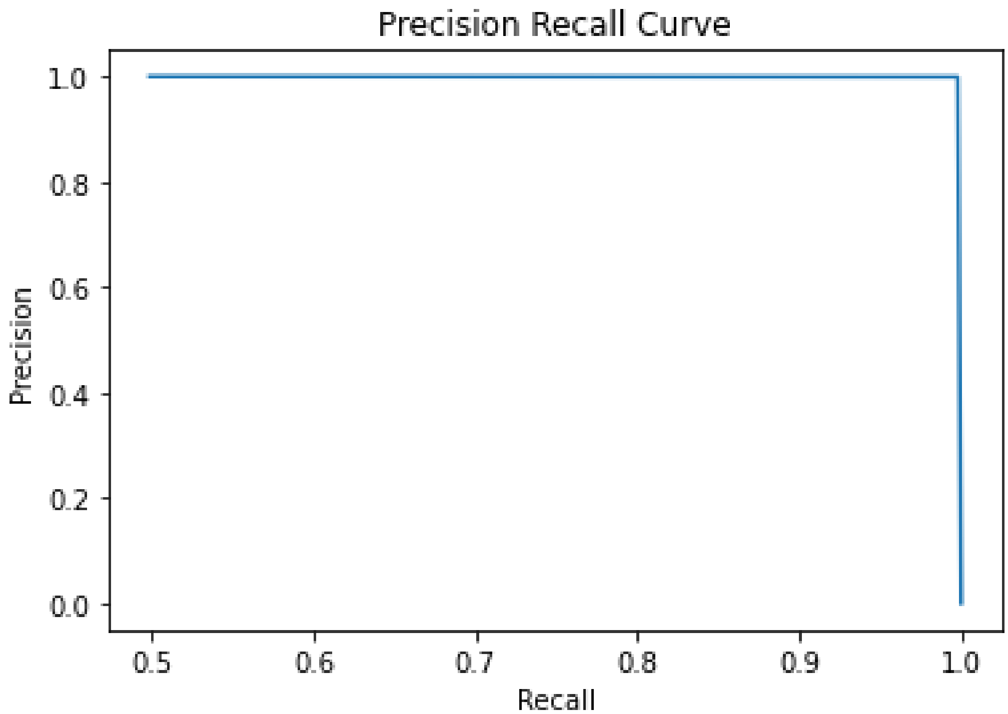
In [46]:

```
evaluate(Y_pred_dt_smote, Y_pred_prob_dt_smote)
Accuracy:  0.9985326189386392
Precision:  0.9978430262341934
Recall:  0.9992170203574707
F1-Score:  0.9985295506360705
AUC score:  0.9985345116823332
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     37244
       Fraud       1.00      1.00      1.00     37038

    accuracy                           1.00     74282
   macro avg       1.00      1.00      1.00     74282
weighted avg       1.00      1.00      1.00     74282
```


Confusion matrix
37164
80
29
37009
Normal
Fraud
True class
Normal
Fraud
Predicted class
35000
30000
25000
20000
15000
10000
5000


Precision Recall Curve
1.0
0.8
0.6
0.4
0.2
0.0
Precision
0.5
0.6
0.7
0.8
0.9
1.0
Recall

In [ ]: