```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec

data = pd.read_csv("creditcard.csv")
data.head()
data= data.fillna(0)

data.dropna(axis= 1,how='any')
```

```
      Time        V1        V2        V3        V4        V5        V6
\
0        0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388

1        0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361

2        1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499

3        1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203

4        2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921

...    ...       ...       ...       ...       ...       ...       ...

1981  1524 -0.340622  1.132232  1.291494  0.062313  0.016387 -0.977070

1982  1525 -1.842696  1.740641  0.861526 -0.856315 -0.655376 -0.842786

1983  1525 -0.480693  0.646091  1.577264 -0.084411 -0.305958 -0.534739

1984  1525 -0.342132  1.091125  1.282729  0.068076 -0.022498 -0.996727

1985  1526 -0.854343  1.382948  1.278665  2.914727 -0.183139 -0.349329


            V7        V8        V9  ...       V21       V22       V23
\
0     0.239599  0.098698  0.363787  ... -0.018307  0.277838 -0.110474

1    -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288

2     0.791461  0.247676 -1.514654  ...  0.247998  0.771679  0.909412

3     0.237609  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321

4     0.592941 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458

...        ...       ...       ...  ...       ...       ...       ...
```

```
1981  0.723755 -0.074630 -0.396655  ... -0.262948 -0.688785 -0.010937
1982  0.198563  0.602764  0.455595  ... -0.213609 -0.400617  0.030013
1983  0.860346 -0.028569 -0.800705  ...  0.121681  0.175190  0.035986
1984  0.676304 -0.042250 -0.312036  ... -0.269850 -0.734148 -0.007354
1985  0.274566  0.435277 -1.576521  ...  0.000000  0.000000  0.000000

           V24       V25       V26       V27       V28  Amount  Class
0     0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    0.0
1    -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    0.0
2    -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    0.0
3    -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    0.0
4     0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    0.0
...        ...       ...       ...       ...       ...     ...    ...
1981  0.334061 -0.160025  0.071779  0.245128  0.098336    5.35    0.0
1982  0.512611 -0.077087  0.286218  0.586012  0.352610    1.00    0.0
1983  0.557665 -0.112301  0.337154 -0.015602  0.051504   80.70    0.0
1984  0.319161 -0.179146  0.073683  0.241932  0.097139    3.59    0.0
1985  0.000000  0.000000  0.000000  0.000000  0.000000    0.00    0.0

[1986 rows x 31 columns]
```

```
print(data['Class'])
```

```
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
1981    0.0
1982    0.0
1983    0.0
1984    0.0
```

```
1985    0.0
Name: Class, Length: 1986, dtype: float64

print(data.shape)
print(data.describe())

(1986, 31)
              Time            V1            V2            V3            V4
\
count  1986.000000  1986.000000  1986.000000  1986.000000  1986.000000

mean    761.035750    -0.284195     0.266886     0.848005     0.151216

std     451.034025     1.353508     1.142026     1.012645     1.264932

min       0.000000   -11.140706   -12.114213   -12.389545    -4.657545

25%     366.000000    -1.045512    -0.204111     0.280517    -0.670513

50%     750.000000    -0.437621     0.314294     0.864505     0.190698

75%    1161.000000     1.095047     0.926126     1.486942     1.002546

max    1526.000000     1.685314     6.118940     4.017561     6.013346


                 V5           V6           V7           V8           V9
...  \
count  1986.000000  1986.000000  1986.000000  1986.000000  1986.000000
...
mean     -0.077457     0.050205     0.138347    -0.058795     0.012145
...
std       1.272512     1.274204     1.140750     0.966493     0.900828
...
min     -32.092129    -3.498447    -4.925568   -12.258158    -3.110515
...
25%      -0.576269    -0.691393    -0.286991    -0.172322    -0.479310
...
50%      -0.154843    -0.198063     0.117535     0.037598    -0.034097
...
75%       0.376901     0.389714     0.569262     0.279513     0.449706
...
max       7.672544    21.393069    34.303177     3.877662     6.450992
...

                V21          V22          V23          V24          V25
\
count  1986.000000  1986.000000  1986.000000  1986.000000  1986.000000

mean     -0.011605    -0.144246    -0.043024     0.013857     0.108318
```

```
std       0.653036     0.588062     0.352803     0.601219     0.407778
min      -4.709977    -2.776923    -4.020300    -2.162523    -1.577384
25%      -0.226879    -0.547314    -0.181141    -0.350094    -0.150713
50%      -0.087276    -0.151669    -0.056958     0.092543     0.131114
75%       0.083325     0.252640     0.064850     0.428657     0.383289
max       6.765928     1.957759     4.095021     1.215279     1.629684

                V26          V27          V28       Amount        Class
count  1986.000000  1986.000000  1986.000000  1986.000000  1986.000000
mean      0.049383     0.027183    -0.002017    68.567925     0.001007
std       0.454138     0.369392     0.272795   241.621039     0.031726
min      -1.243924    -5.336289    -2.738566     0.000000     0.000000
25%      -0.280862    -0.049397    -0.020939     4.950000     0.000000
50%       0.036473     0.022920     0.022645    15.085000     0.000000
75%       0.303332     0.140279     0.090899    63.467500     0.000000
max       3.463246     3.852046     4.157934  7712.430000     1.000000

[8 rows x 31 columns]
```

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

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

```
0.0010080645161290322
```

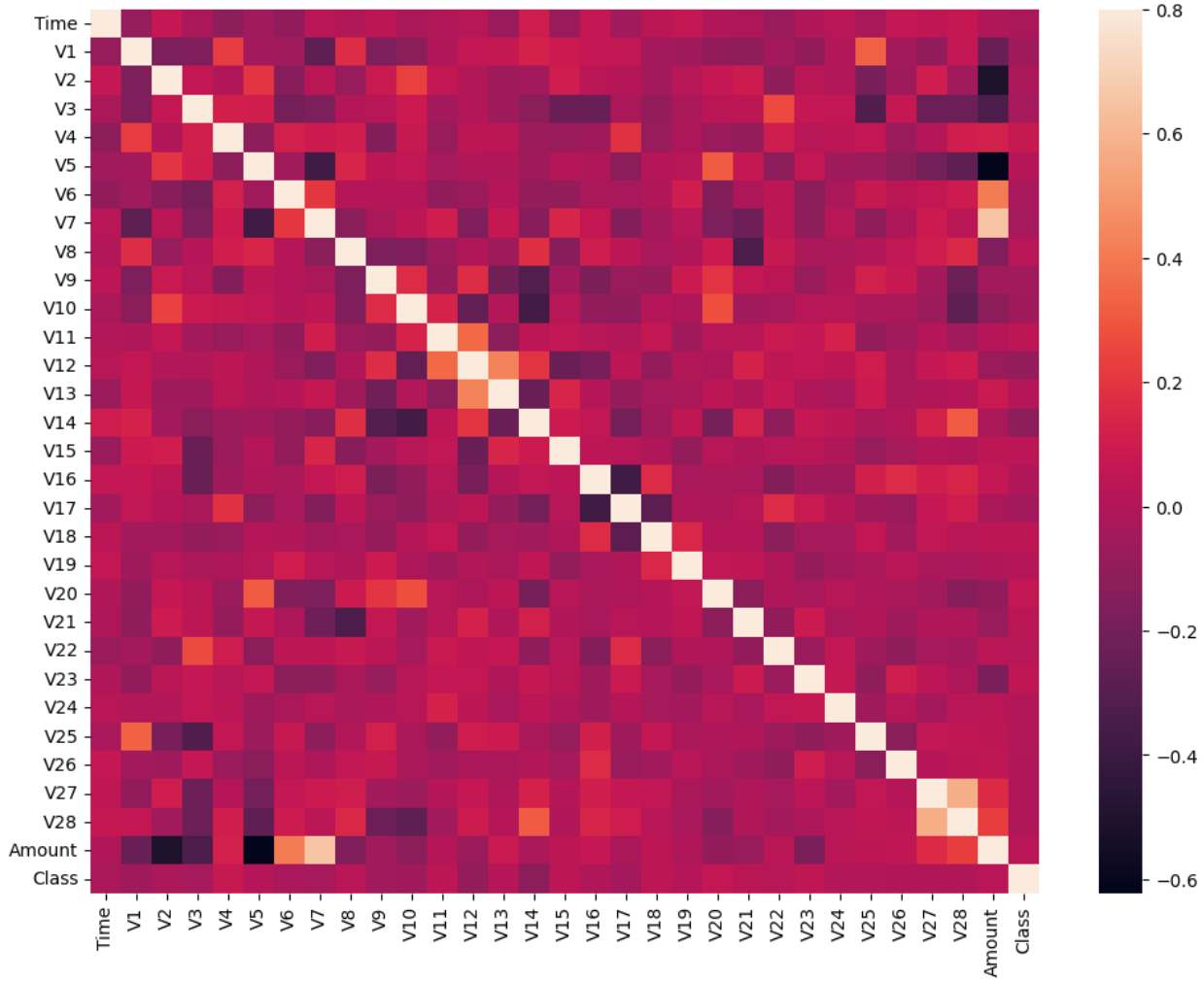
```
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
print('Amount details of the fraudulent transaction')
fraud.Amount.describe()
print('details of valid transaction')
valid.Amount.describe()
```

```
Fraud Cases: 2
Valid Transactions: 1984
```

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

count    1984.000000
mean       68.370413
std       241.516646
min         0.000000
25%         4.950000
50%        15.085000
75%        63.102500
max      7712.430000
Name: Amount, dtype: float64

corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = False)
plt.show()
```



```
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)
xData = X.values
yData = Y.values
from sklearn.model_selection import train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(
           xData, yData, test_size = 0.2, random_state = 42)
```

```
(1986, 30)
(1986,)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
yPred = rfc.predict(xTest)
```

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

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

```
The model used is Random Forest classifier
```

```
acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))
```

```
prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))
```

```
rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))
```

```
f1 = f1_score(yTest, yPred)
print("The F1-Score is {}".format(f1))
```

```
The accuracy is 1.0
The precision is 0.0
The recall is 0.0
The F1-Score is 0.0
```

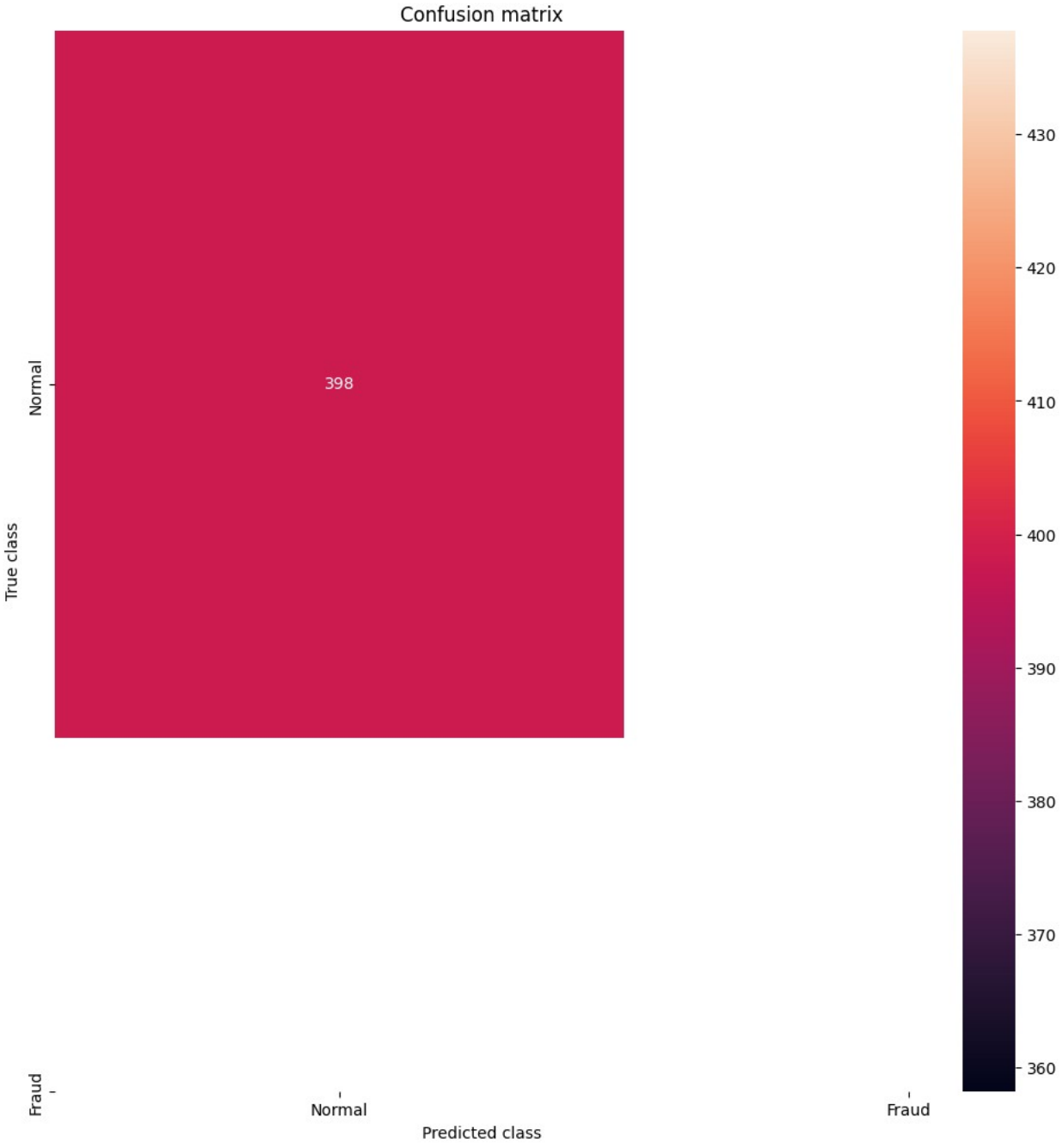
```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Recall is ill-defined and being set
to 0.0 due to no true samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1609: UndefinedMetricWarning: F-score is ill-defined and being
set to 0.0 due to no true nor predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, "true nor predicted", "F-score is",
len(true_sum))
```

```
MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is {}".format(MCC))
```

```
The Matthews correlation coefficient is 0.0
```

```
# printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize =(12, 12))
sns.heatmap(conf_matrix, xticklabels = LABELS,
                 yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```


Confusion matrix
398
Normal
Fraud
True class
Normal
Fraud
Predicted class
430
420
410
400
390
380
370
360