```
# import the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec

# 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("/content/creditcard.csv")

# Grab a peek at the data
data.head()

{"type":"dataframe","variable_name":"data"}

# Print the shape of the data
# data = data.sample(frac = 0.1, random_state = 48)
print(data.shape)
print(data.describe())

(9965, 31)
               Time            V1            V2            V3
V4  \
count   9965.000000  9965.000000  9965.000000  9964.000000
9964.000000
mean    5934.484897    -0.241681     0.280693     0.906359
0.260247
std     4449.407112     1.522695     1.308882     1.156948
1.440500
min        0.000000   -27.670569   -34.607649   -15.496222    -
4.657545
25%     2061.000000    -1.012157    -0.208591     0.412198    -
0.615591
50%     4547.000000    -0.372624     0.286179     0.943149
0.218492
75%    10163.000000     1.151506     0.900823     1.601871
1.119367
max    14864.000000     1.960497     8.636214     4.101716
10.463020

                V5            V6            V7            V8            V9
...  \
count  9964.000000  9964.000000  9964.000000  9964.000000  9964.000000
...
mean     -0.046342     0.132335    -0.071847    -0.065317     0.801220
...
std       1.183634     1.307586     1.077872     1.260140     1.156424
...
```

```
min     -32.092129   -23.496714   -26.548144   -23.632502    -6.329801
...
25%      -0.643060    -0.630075    -0.542336    -0.190495     0.069439
...
50%      -0.153090    -0.153577    -0.054658     0.012466     0.804930
...
75%       0.371762     0.503376     0.476280     0.273016     1.506066
...
max      34.099309    21.393069    34.303177     5.060381    10.392889
...

                V21          V22          V23          V24          V25
\
count  9964.000000  9964.000000  9964.000000  9964.000000  9964.000000

mean     -0.052099    -0.152091    -0.033426     0.021638     0.087302

std       0.914735     0.631310     0.488203     0.593818     0.428128

min     -11.468435    -8.527145   -15.144340    -2.512377    -2.577363

25%      -0.268191    -0.548412    -0.174222    -0.327438    -0.157704

50%      -0.123101    -0.136078    -0.046009     0.079935     0.121180

75%       0.032707     0.247913     0.081288     0.410877     0.359418

max      22.588989     4.534454    13.876221     3.200201     5.525093


                V26          V27          V28       Amount        Class

count  9964.000000  9964.000000  9964.000000  9964.000000  9964.000000

mean      0.108328     0.005614     0.003051    62.968359     0.003814

std       0.562661     0.411434     0.266532   184.626707     0.061641

min      -1.338556    -7.976100    -3.509250     0.000000     0.000000

25%      -0.328193    -0.084489    -0.015751     5.000000     0.000000

50%       0.043395    -0.004505     0.015904    15.950000     0.000000

75%       0.478249     0.121045     0.077418    50.792500     0.000000

max       3.517346     8.254376     4.860769  7712.430000     1.000000

[8 rows x 31 columns]
```

```
# Determine number of fraud cases in dataset
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])))
```

```
0.0038283296393310496
Fraud Cases: 38
Valid Transactions: 9926
```

```
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

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

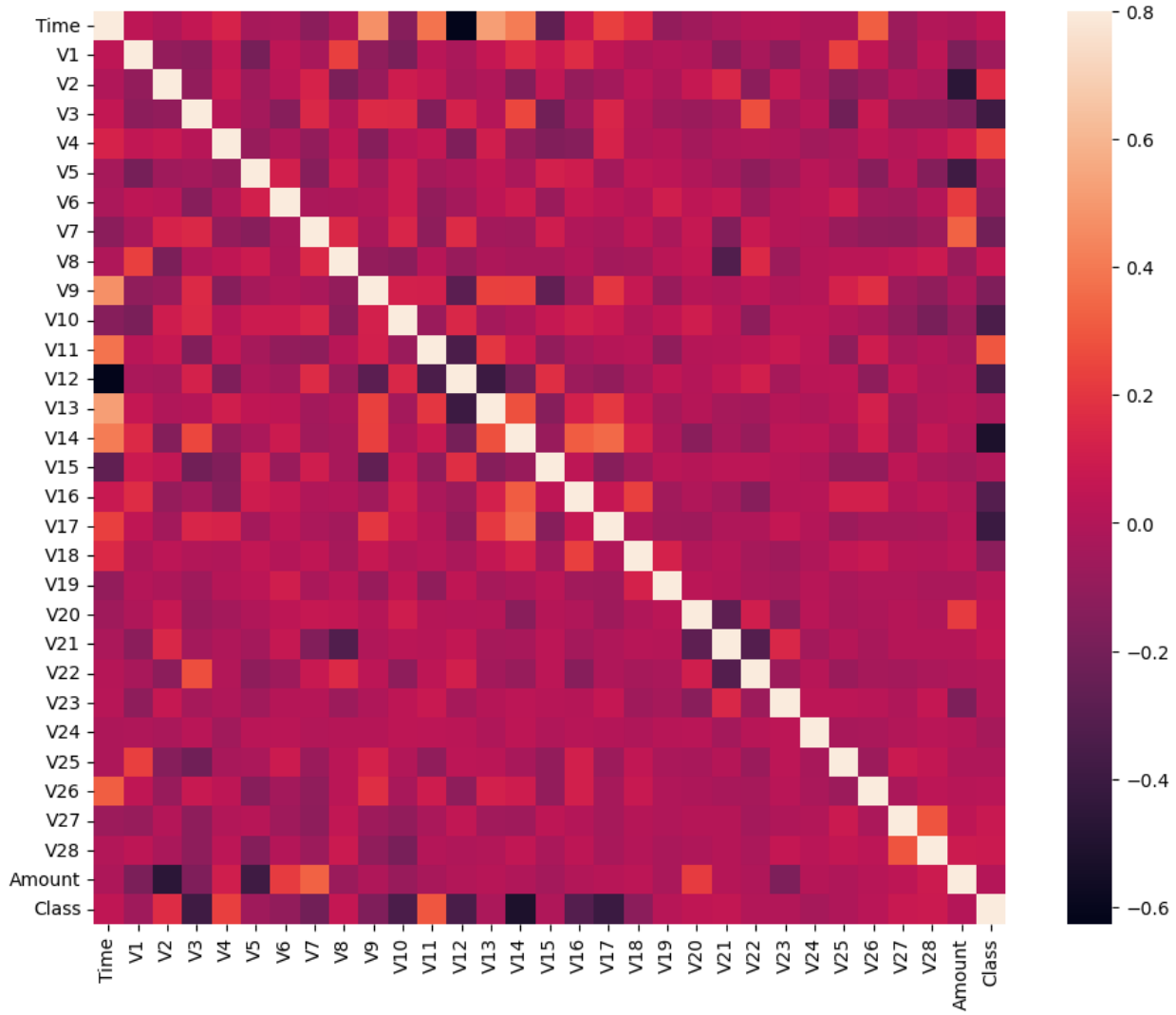
```
count      38.000000
mean       75.730526
std       304.521215
min         0.000000
25%         1.000000
50%         1.000000
75%         1.000000
max      1809.680000
Name: Amount, dtype: float64
```

```
print("details of valid transaction")
valid.Amount.describe()
```

```
details of valid transaction
```

```
count    9926.000000
mean       62.919501
std       184.041297
min         0.000000
25%         5.172500
50%        15.950000
75%        51.067500
max      7712.430000
Name: Amount, dtype: float64
```

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




```
# 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
```

```
(9965, 30)
(9965,)
```

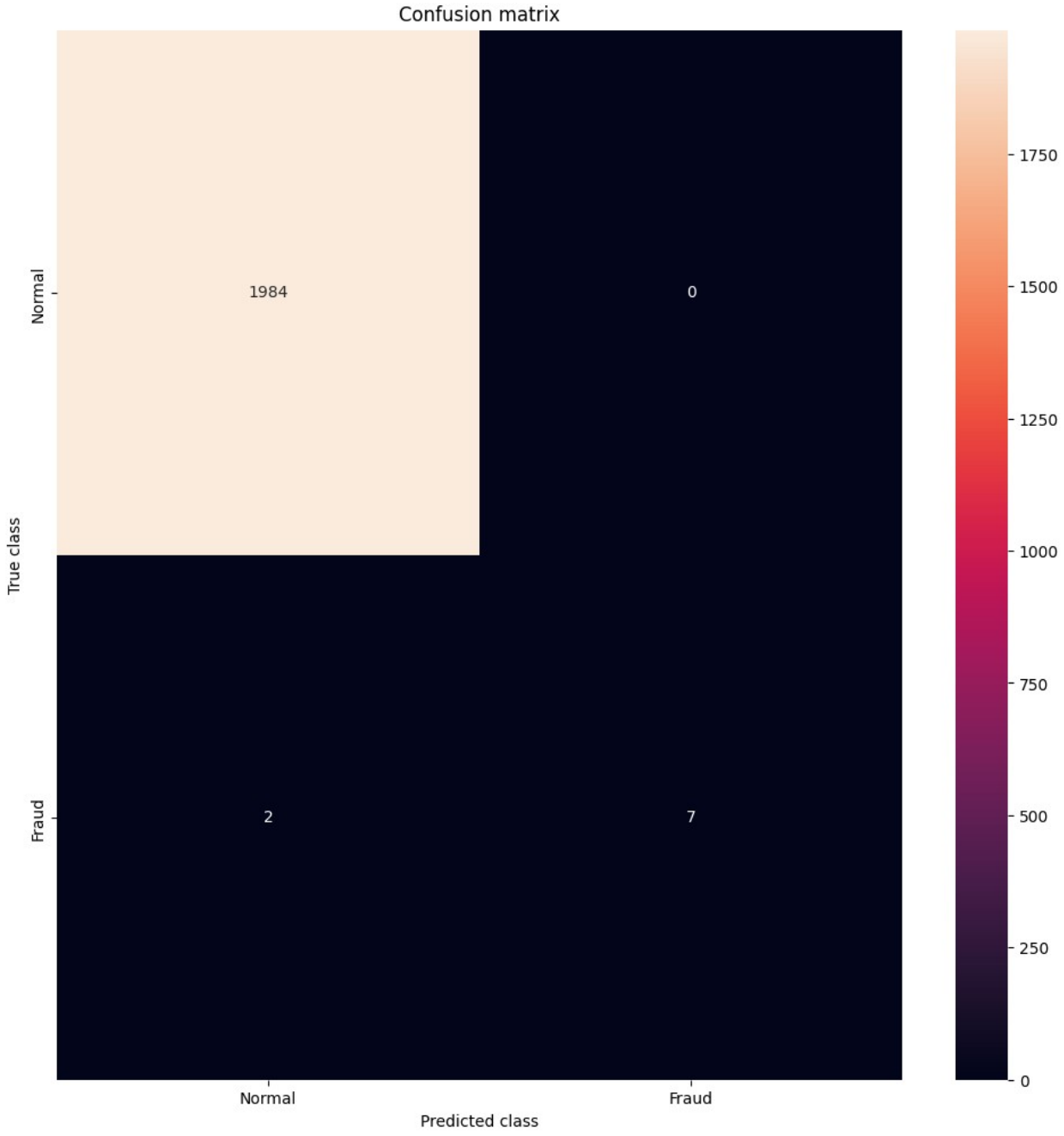
```
# Using Scikit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
```

```
xTrain, xTest, yTrain, yTest = train_test_split(
           xData, yData, test_size = 0.2, random_state = 42)

import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer

# Check for NaN values in yTrain and remove corresponding rows from
xTrain
nan_indices = np.isnan(yTrain)
xTrain_clean = xTrain[~nan_indices]
yTrain_clean = yTrain[~nan_indices]

# Create an imputer instance
imputer = SimpleImputer(strategy='mean')

# Impute missing values in the training and test data
xTrain_imputed = imputer.fit_transform(xTrain_clean)
xTest_imputed = imputer.transform(xTest)

# Create RandomForestClassifier instance
rfc = RandomForestClassifier()

# Train the model
rfc.fit(xTrain_imputed, yTrain_clean)

# Make predictions
yPred = rfc.predict(xTest_imputed)



# Evaluating the classifier
# printing every score of the classifier
# scoring in anything
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")

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))

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

```
The model used is Random Forest classifier
The accuracy is 0.9989964877069744
The precision is 1.0
The recall is 0.7777777777777778
The F1-Score is 0.8750000000000001
The Matthews correlation coefficient is0.881472924811526
```

```
# 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
Normal
Fraud
True class
1984
0
2
7
Normal
Fraud
Predicted class
1750
1500
1250
1000
750
500
250
0