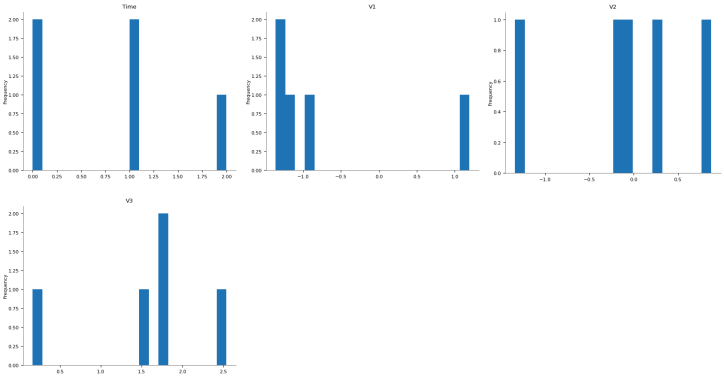
```
# import the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec


# 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()
```

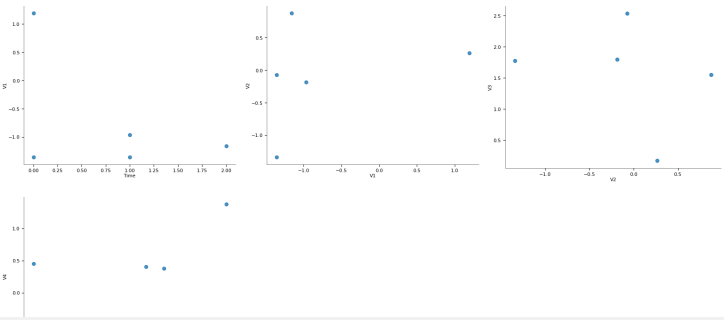
	Time	V1	V2	V3	V4	V5	V6	V7	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

Distributions



2-d distributions



```
data.dropna(axis = 0, inplace = True)
print(data.head(), data.shape)
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -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    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns] (17917, 31)
```

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

```
(17917, 31)
                Time            V1            V2            V3            V4  \
count  17917.000000  17917.000000  17917.000000  17917.000000  17917.000000
mean   13904.432048     -0.245042      0.258176      0.777821      0.291555
std     9867.544555      1.893189      1.508337      1.766920      1.479539
min        0.000000    -30.552380    -40.978852    -31.103685     -5.172595
25%     3781.000000     -0.959859     -0.305382      0.338302     -0.629984
50%    12346.000000     -0.306847      0.235109      0.924305      0.229959
75%    23772.000000      1.164015      0.876556      1.557400      1.155742
max    29030.000000      1.960497     16.713389      4.101716     11.927512

                 V5            V6            V7            V8            V9  \
count  17917.000000  17917.000000  17917.000000  17917.000000  17917.000000
mean      -0.146329      0.099878     -0.150970      0.012969      0.734707
std        1.423917      1.327756      1.342027      1.318460      1.273597
min      -32.092129    -23.496714    -26.548144    -23.632502     -7.175097
25%       -0.729796     -0.651820     -0.599770     -0.175261     -0.102798
50%       -0.192681     -0.169764     -0.076457      0.020846      0.726588
75%        0.347812      0.493661      0.447398      0.272076      1.480254
max       34.099309     21.393069     34.303177     20.007208     10.392889

       ...           V21           V22           V23           V24  \
count  ...  17917.000000  17917.000000  17917.000000  17917.000000
mean   ...     -0.052197     -0.146256     -0.038504      0.014510
std    ...      0.826158      0.634077      0.526168      0.588938
min    ...    -11.468435     -8.593642    -26.751119     -2.687773
25%    ...     -0.262581     -0.556175     -0.173509     -0.330602
50%    ...     -0.119493     -0.118803     -0.046656      0.063623
75%    ...      0.041561      0.250886      0.072758      0.398391
max    ...     22.614889      4.534454     13.876221      3.695503

                V25           V26           V27           V28        Amount  \
count  17917.000000  17917.000000  17917.000000  17917.000000  17917.000000
mean       0.119974      0.036454      0.014558      0.007047     67.504000
std        0.438850      0.537046      0.397673      0.248679    189.184677
min       -7.495741     -1.338556     -8.567638     -3.575312      0.000000
25%       -0.140146     -0.345451     -0.071486     -0.011786      5.490000
50%        0.158356     -0.030200      0.003094      0.018529     15.950000
75%        0.397765      0.342446      0.098968      0.077345     56.670000
max        5.525093      3.517346      8.254376      4.860769   7712.430000

             Class
count  17917.000000
mean       0.004521
std        0.067087
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        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])))
```

```
0.004541376990356582
Fraud Cases: 81
Valid Transactions: 17836
```

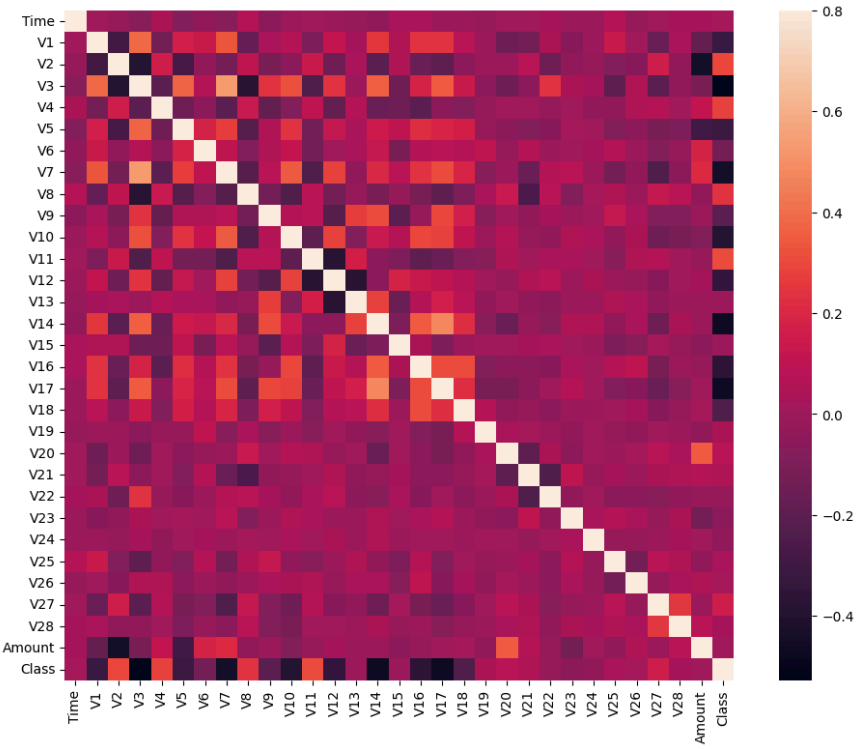
```
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

```
Amount details of the fraudulent transaction
count      81.000000
mean       98.105926
std       267.464067
min         0.000000
25%         1.000000
50%         1.000000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

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

```
details of valid transaction
count    17836.000000
mean        67.365025
std        188.754429
min          0.000000
25%          5.490000
50%         15.950000
75%         56.232500
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
```

```
(17917, 30)
(17917,)
```

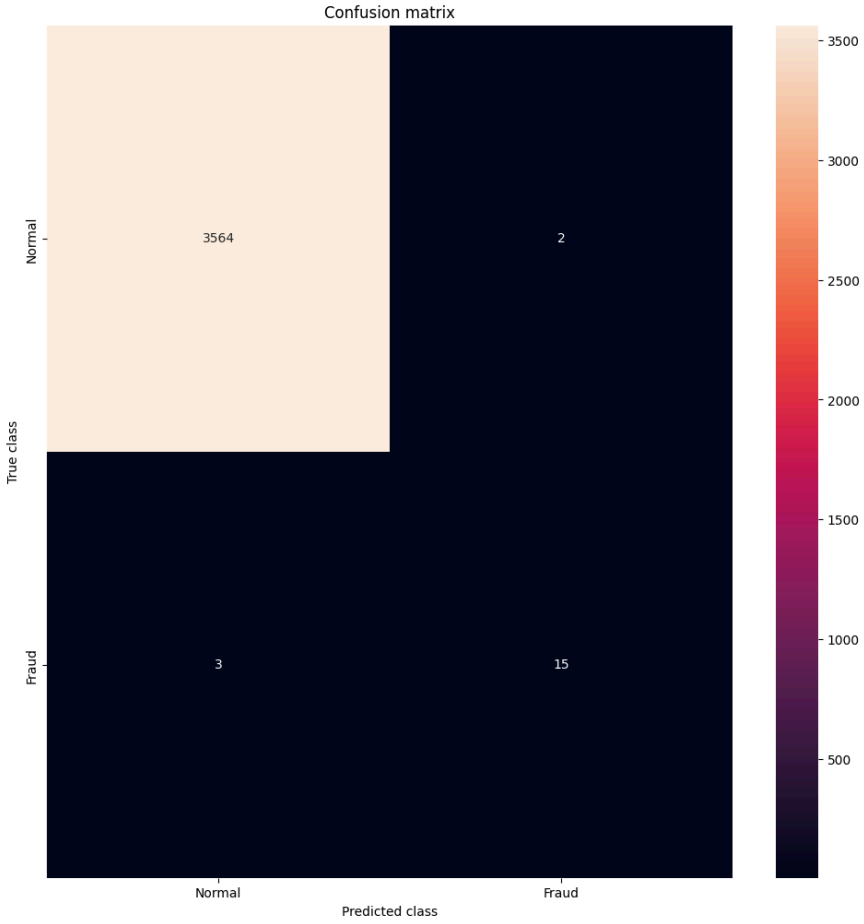
```
# Using Scikit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
xTrain, xTest, yTrain, yTest = train_test_split(
        xData, yData, test_size = 0.2, random_state = 42)
```

```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
# predictions
yPred = rfc.predict(xTest)
```

```
# 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 0.9986049107142857
The precision is 0.8823529411764706
The recall is 0.8333333333333334
The F1-Score is 0.8571428571428571
The Matthews correlation coefficient is0.856795687676577
```

```
# 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
3564
2
3
15
Normal
Fraud
True class
Normal
Fraud
Predicted class
3500
3000
2500
2000
1500
1000
500