```
# 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
data = pd.read_csv("/content/credit.csv")
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V2
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.11047
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.10128
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.90941
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.19032
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.13745

5 rows × 31 columns

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

```
(11683, 31)
<bound method NDFrame.describe of         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
...      ...       ...       ...       ...       ...       ...       ...
11678  19921  1.208802 -0.401943  0.901086 -0.697416 -1.014073 -0.281932
11679  19924 -1.723814  1.389327  1.411353 -0.716019 -1.561864  1.505156
11680  19926  1.192037 -0.357840  1.002156 -0.558666 -1.017703 -0.321732
11681  19927 -7.773912  4.249596 -5.985636  1.450199 -4.709726 -1.302327
11682  19929  1.024814 -1.179948  1.702954 -0.833752 -1.598620  1.270610

             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
...         ...       ...       ...  ...       ...       ...       ...
11678 -0.828187  0.108108  3.018142  ... -0.137886 -0.007707 -0.075264
11679 -2.024937 -5.393713  2.156363  ...  5.161661 -1.997550 -0.202928
11680 -0.780201  0.062111  3.206257  ... -0.138692  0.082080  0.007441
11681 -2.807678  4.890516  0.807323  ...  0.104104 -0.188352 -0.302390
11682 -1.786000       NaN       NaN  ...       NaN       NaN       NaN

            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
...         ...       ...       ...       ...       ...     ...    ...
11678 -0.065906  0.454008 -0.719787  0.054139  0.015084   11.85    0.0
11679 -0.040939  1.221328  1.155446  0.334758  0.104672  155.38    0.0
11680  0.018529  0.378177 -0.693956  0.077499  0.025269    8.35    0.0
11681  0.299249 -0.185131 -0.445921  0.143783 -0.061396   89.99    0.0
11682       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[11683 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])))
```

```
0.004212155076076678
Fraud Cases: 49
Valid Transactions: 11633
```

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

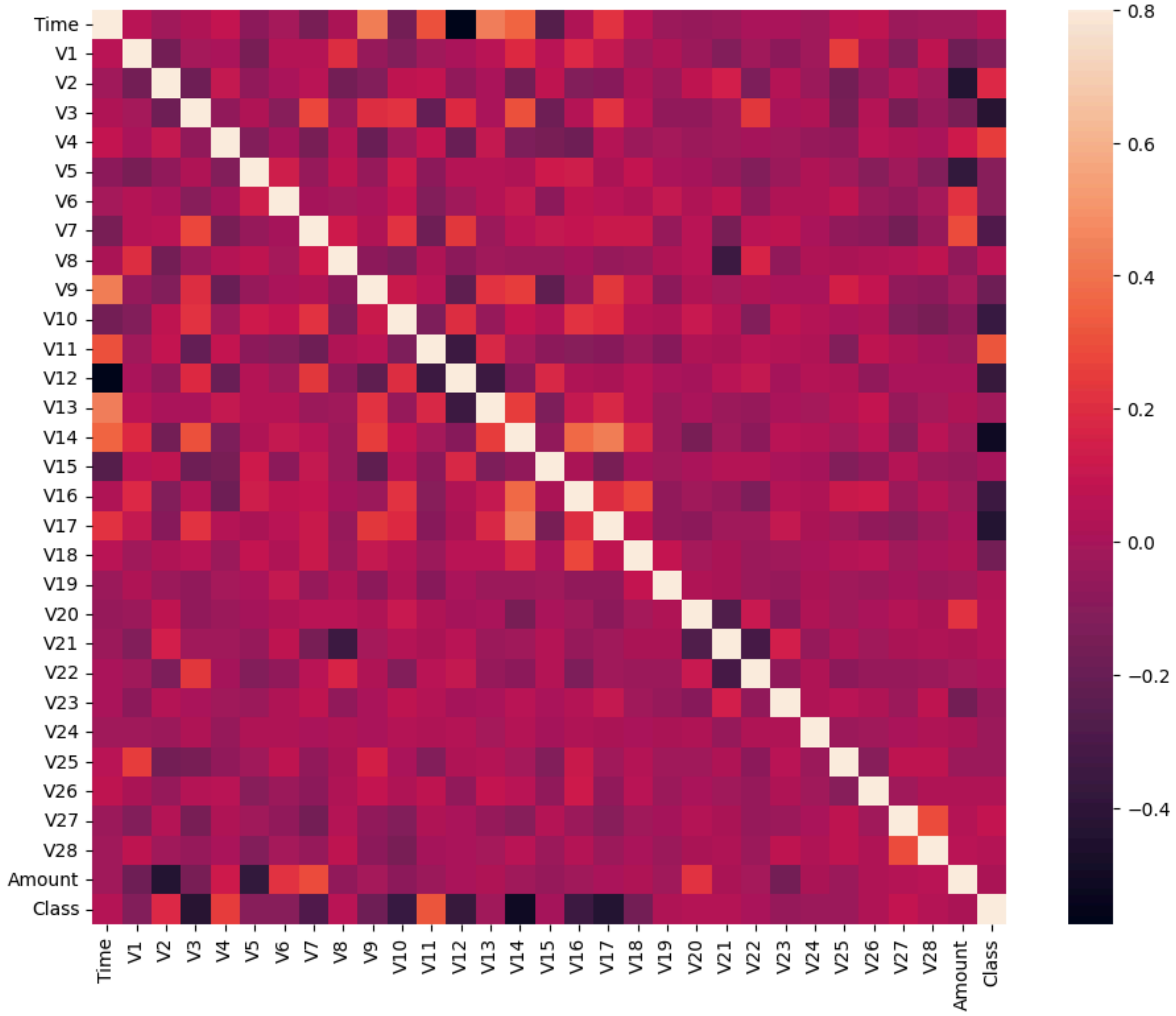
```
Amount details of the fraudulent transaction
count      49.000000
mean      103.646735
std       330.135333
min         0.000000
25%         1.000000
50%         1.000000
75%         3.790000
max      1809.680000
Name: Amount, dtype: float64
```

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

```
details of valid transaction
count    11633.000000
mean        62.572337
std        178.797878
min          0.000000
25%          5.180000
50%         15.950000
75%         50.000000
max       7712.430000
Name: Amount, dtype: float64
```

Plotting the Correlation Matrix

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



Separating the X and the Y values

```
# 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
```

```
(11683, 30)
(11683,)
```

Training and Testing Data Bifurcation

We will be dividing the dataset into two main groups. One for training the model and the other for Testing our trained model's performance.

```
# 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 a Random Forest Model using scikit learn

```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer  # Import the imputer

# Create an imputer to fill missing values (e.g., with the mean)
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform both training and testing data
xTrain = imputer.fit_transform(xTrain)
xTest = imputer.transform(xTest)

# Handle missing values in yTrain (if any)
# Assuming 'Class' is a categorical feature, we'll use the most frequent value
from sklearn.impute import SimpleImputer
y_imputer = SimpleImputer(strategy='most_frequent')
yTrain = y_imputer.fit_transform(yTrain.reshape(-1, 1)).ravel()  # Reshape for the imputer and flatten back

# 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
import numpy as np # Import numpy for handling NaNs

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")
# Handle NaNs in yTest (replace with a suitable value, e.g., 0)
yTest_no_nan = np.nan_to_num(yTest, nan=0)
```

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

```
# 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
```

```
import numpy as np # Import numpy for handling NaNs

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")
# Handle NaNs in yTest (replace with a suitable value, e.g., 0)
yTest_no_nan = np.nan_to_num(yTest, nan=0) # Replace NaNs in yTest

# printing the confusion matrix
LABELS = ['Normal', 'Fraud']
# Use yTest_no_nan which has no NaNs
conf_matrix = confusion_matrix(yTest_no_nan, 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()
```

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