Machine Learning Project Credit Card Fraud Detection

By Aadarsh Dwivedi

✴ Problem Definition

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. The aim is to build a machine learning system in python that can detect whether a credit card transaction is legit or fraud.

✴ Dataset

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

It contains only numerical input variables. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise..

```
#importing dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import Normalizer
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn import svm
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
#loading dataset to a pandas dataframe
credit_card_data=pd.read_csv('/content/drive/MyDrive/credit_card_datas
et/creditcard.csv')
```

✴ Prepare Data

Summarization

```
#print dimension of dataset
print(credit_card_data.shape)
```

```
(284807, 31)
```

```
#first 5 rows of data set
credit_card_data.head()
```

```
   Time        V1        V2        V3  ...       V27       V28  Amount  
Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62
0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69
0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66
0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50
0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99
0

[5 rows x 31 columns]
```

```
#last 5 rows
credit_card_data.tail()
```

```
            Time         V1         V2  ...       V28  Amount  Class
284802  172786.0 -11.881118  10.071785  ...  0.823731    0.77      0
284803  172787.0  -0.732789  -0.055080  ... -0.053527   24.79      0
284804  172788.0   1.919565  -0.301254  ... -0.026561   67.88      0
284805  172788.0  -0.240440   0.530483  ...  0.104533   10.00      0
284806  172792.0  -0.533413  -0.189733  ...  0.013649  217.00      0

[5 rows x 31 columns]
```

```
#get some information about dataset
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
```

```
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
#there is also another method to find the number of missing values
credit_card_data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
```

```
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
#checking the distribution of legit transaction and fradulant transaction
credit_card_data['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

This dataset is highly unbalanced.

Preprocessing

```
#seperate normal and fradulant transaction from the data frame.
#create 2 variables legit and fraud

legit = credit_card_data[credit_card_data.Class == 0] #if class value is 0, that entire row will be stored in legit variable
fraud = credit_card_data[credit_card_data.Class == 1]

print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```
#to get statistical measures of legit data
legit.Amount.describe()
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
#to get statistical measures of fraud data
fraud.Amount.describe()
```

```
count      492.000000
mean       122.211321
std        256.683288
min          0.000000
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
```

```
#compare the values of both transation
credit_card_data.groupby('Class').mean()
```

```
               Time         V1         V2  ...        V27        V28
Amount
Class                                      ...

0      94838.202258   0.008258  -0.006271  ... -0.000295 -0.000131
88.291022
1      80746.806911  -4.771948   3.623778  ...  0.170575  0.075667
122.211321

[2 rows x 30 columns]
```

Under Sampling

Build a sample dataset containing similar distribution of normal and fradulant transaction

```
# in the 284315 legit transactions, randomly pick 492 transactions and
join with 492 fradulant transactions ,thus the data set can be
balanced.

legit_sample=legit.sample(n=492)
new_dataset = pd.concat([legit_sample , fraud], axis=0)   #concatenate
492 legit and 492 fradulant dataframe into new dataset
#if axis=0, data frame to be added 1 by 1
#if axis=1, the values will be added column wise



#check first 5 rows of new dataset
new_dataset.head()

#here the serial numbers , we can see they are randomly..
```

```
            Time        V1        V2  ...       V28  Amount  Class
213674  139347.0 -0.465789  0.474828  ...  0.233988   77.50      0
84967    60542.0  1.293079  0.089837  ...  0.014171    7.83      0
197214  131899.0  2.053947 -0.678694  ... -0.007479   69.00      0
134961   81048.0 -1.782741  0.381524  ... -0.087533   39.45      0
```

```
242703  151609.0 -2.962208 -3.718305  ...  0.100407  360.77      0

[5 rows x 31 columns]
```

```
#check last 5 rows
new_dataset.tail()
```

```
             Time        V1        V2  ...       V28  Amount  Class
279863  169142.0 -1.927883  1.125653  ...  0.147968  390.00      1
280143  169347.0  1.378559  1.289381  ...  0.186637    0.76      1
280149  169351.0 -0.676143  1.126366  ...  0.194361   77.89      1
281144  169966.0 -3.113832  0.585864  ... -0.253700  245.00      1
281674  170348.0  1.991976  0.158476  ... -0.015309   42.53      1

[5 rows x 31 columns]
```

```
new_dataset['Class'].value_counts()
```

```
1    492
0    492
Name: Class, dtype: int64
```

```
new_dataset.groupby('Class').mean()
# here the different in mean is same as that of last time ,thus we
find that nature of dataset is not changed.
#since the mean values are similar ,our sample is good.
```

```
                Time        V1        V2  ...       V27       V28
Amount
Class                                     ...

0      95371.963415  0.037499 -0.024024  ... -0.003366  0.016366
91.727520
1      80746.806911 -4.771948  3.623778  ...  0.170575  0.075667
122.211321

[2 rows x 30 columns]
```

```
#splitting data into features and target {to feed to machinelearning
model}

X = new_dataset.drop(columns = 'Class', axis = 1)
Y = new_dataset['Class']

print(X)
#here the class column is removed
```

```
             Time        V1        V2  ...       V27       V28  Amount
213674  139347.0 -0.465789  0.474828  ...  0.242662  0.233988   77.50
84967    60542.0  1.293079  0.089837  ...  0.015628  0.014171    7.83
197214  131899.0  2.053947 -0.678694  ...  0.001502 -0.007479   69.00
134961   81048.0 -1.782741  0.381524  ...  0.137311 -0.087533   39.45
```

```
242703  151609.0 -2.962208 -3.718305  ... -0.133632  0.100407  360.77
...          ...       ...       ...  ...       ...       ...     ...
279863  169142.0 -1.927883  1.125653  ...  0.292680  0.147968  390.00
280143  169347.0  1.378559  1.289381  ...  0.389152  0.186637    0.76
280149  169351.0 -0.676143  1.126366  ...  0.385107  0.194361   77.89
281144  169966.0 -3.113832  0.585864  ...  0.884876 -0.253700  245.00
281674  170348.0  1.991976  0.158476  ...  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

```
print(Y)
```

```
213674    0
84967     0
197214    0
134961    0
242703    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

```
# Split data into training and testing data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2 , stratify=Y, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Data Standardization

```
scaler = StandardScaler()
```

```
scaler.fit(X)
```

```
StandardScaler()
```

```
standardized_data = scaler.transform(X)
```

```
print(standardized_data)
```

```
[[ 1.06963592  0.34609309 -0.35762725 ...  0.15823523  0.45578914
  -0.1078734 ]
 [-0.57389262  0.6662363  -0.46153556 ... -0.06762509 -0.07721771
  -0.36290183]
 [ 0.91430313  0.80472695 -0.66896009 ... -0.08167818 -0.12971459
  -0.13898782]
 ...
```

```
 [ 1.69538848  0.30780516 -0.18177891 ...  0.29994412  0.35970405
  -0.1064458 ]
 [ 1.70821469 -0.13589486 -0.32765898 ...  0.79712844 -0.72674728
   0.50526369]
 [ 1.71618155  0.79344726 -0.44300997 ... -0.08020021 -0.1487003
  -0.23588179]]
```

```
X = standardized_data
Y = new_dataset['Class']
```

```
print(X)    #data
print(Y)    #label
```

```
[[ 1.06963592  0.34609309 -0.35762725 ...  0.15823523  0.45578914
  -0.1078734 ]
 [-0.57389262  0.6662363  -0.46153556 ... -0.06762509 -0.07721771
  -0.36290183]
 [ 0.91430313  0.80472695 -0.66896009 ... -0.08167818 -0.12971459
  -0.13898782]
 ...
 [ 1.69538848  0.30780516 -0.18177891 ...  0.29994412  0.35970405
  -0.1064458 ]
 [ 1.70821469 -0.13589486 -0.32765898 ...  0.79712844 -0.72674728
   0.50526369]
 [ 1.71618155  0.79344726 -0.44300997 ... -0.08020021 -0.1487003
  -0.23588179]]
213674    0
84967     0
197214    0
134961    0
242703    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Training the model

```
classifier = svm.SVC(kernel='linear')


classifier.fit(X_train, Y_train)
```

```
SVC(kernel='linear')
```

Model Evaluation

```
#Model Training
```

```
#Logistic Regression

#create variable as model
model = LogisticRegression()

#training the logistic regression model with training data

model.fit(X_train, Y_train)
#x_train contain all features of training data, y_train contain
corresponding labels (0 /1)
#this will fit data in logistic regression model and then we can make
some predictionsfrom it.
```

```
LogisticRegression()
```

```
#accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy on Training Data : ', training_data_accuracy)
```

```
Accuracy on Training Data :  0.9199491740787802
```

```
#accuracy score on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy on Test Data : ', test_data_accuracy)
```

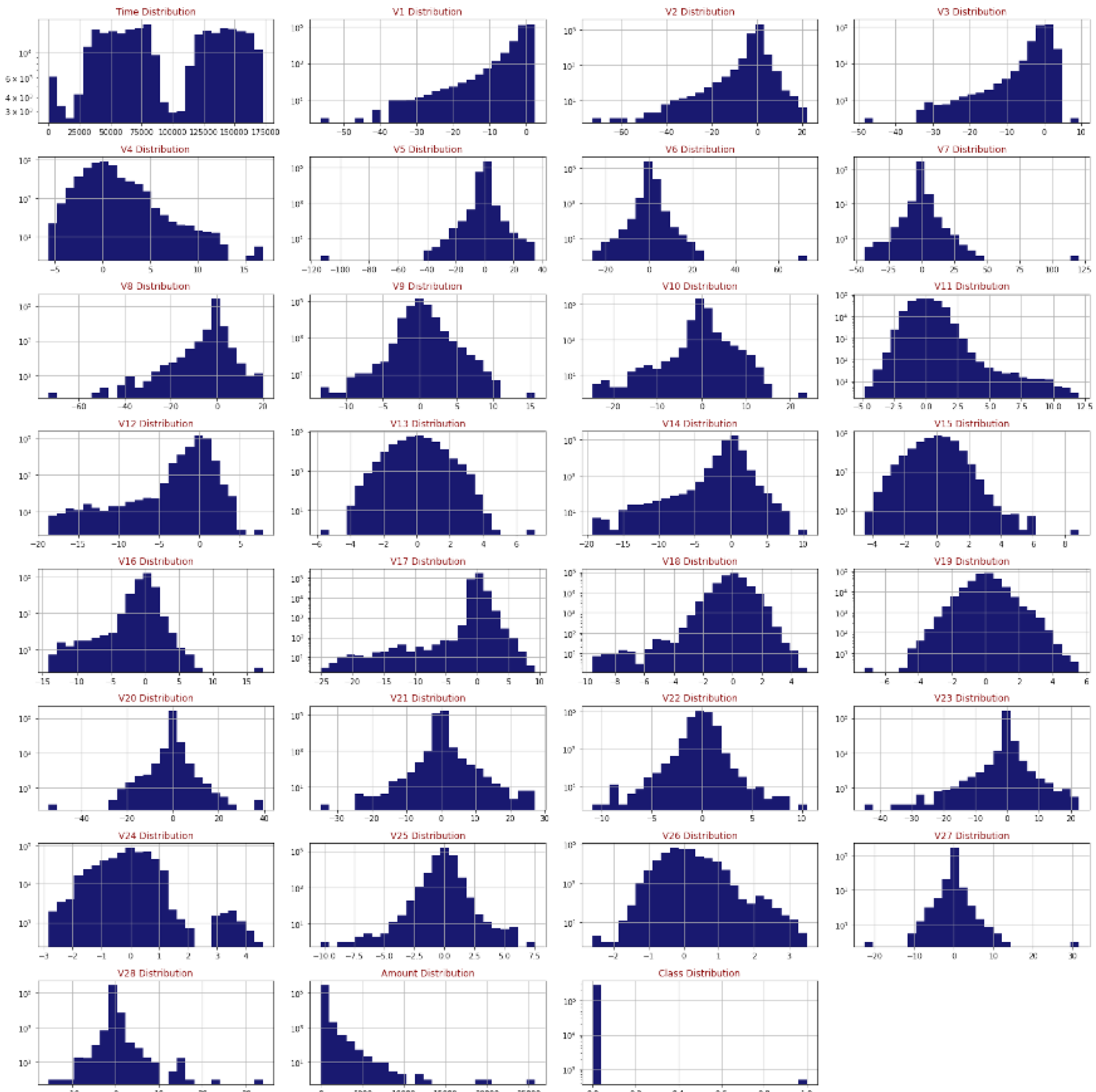
```
Accuracy on Test Data :  0.8883248730964467
```

Note : If Accuracy score on training data is very different from that on test data, then our model is over fitted or under fitted.

Visualization

```
#HISTOGRAM
#credit_card dataset
def draw_histograms(dataframe, features, rows, cols):    #call draw
histogram funtction
    fig=plt.figure(figsize=(20,20))                      #give figure
size
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)

dataframe[feature].hist(bins=25,ax=ax,facecolor='midnightblue')

        ax.set_title(feature+" Distribution",color='DarkRed')
        ax.set_yscale('log')
    fig.tight_layout()
    plt.show()
```

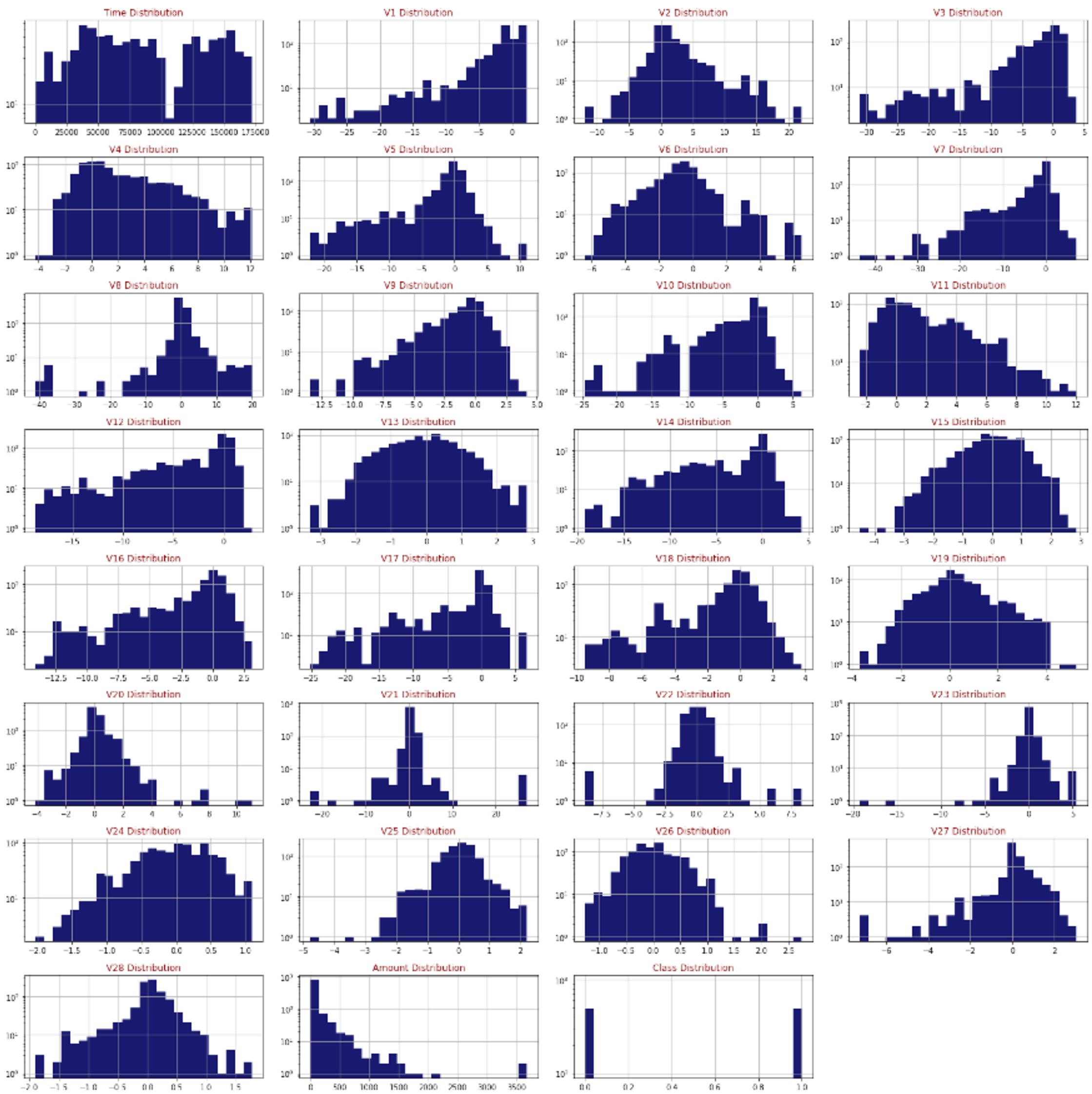
```
draw_histograms(credit_card_data,credit_card_data.columns,8,4)
```




```
#HISTOGRAM
#new_dataset
def draw_histograms(dataframe, features, rows, cols):
    fig=plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)

dataframe[feature].hist(bins=25,ax=ax,facecolor='midnightblue')

        ax.set_title(feature+" Distribution",color='DarkRed')
        ax.set_yscale('log')
```

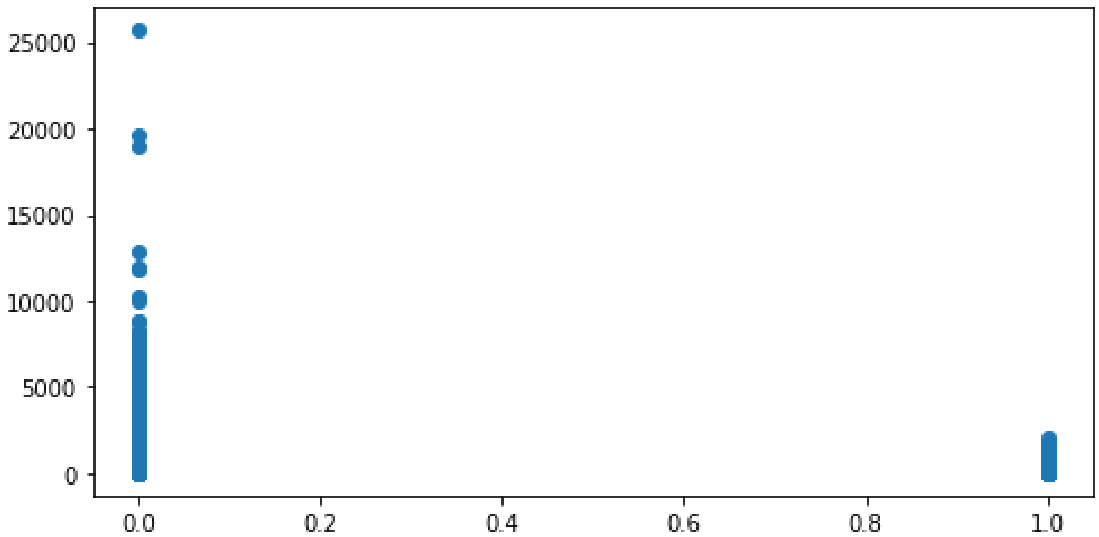
```
    fig.tight_layout()
    plt.show()
```

```
draw_histograms(new_dataset,new_dataset.columns,8,4)
```



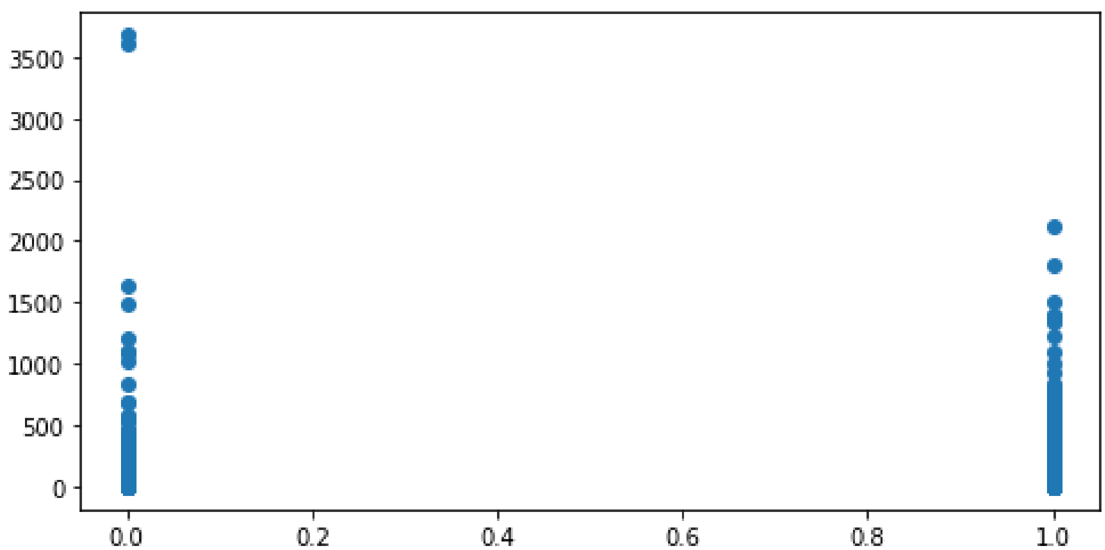

```
#SCATTER PLOT
#credit_card_dataset
from matplotlib import rcParams
rcParams['figure.figsize']= 8,4
plt.scatter(credit_card_data.Class, credit_card_data.Amount)
```

```
<matplotlib.collections.PathCollection at 0x7f795894f390>
```




```
#SCATTER PLOT
#new_dataset
rcParams['figure.figsize']= 8,4
plt.scatter(new_dataset.Class, new_dataset.Amount)
```

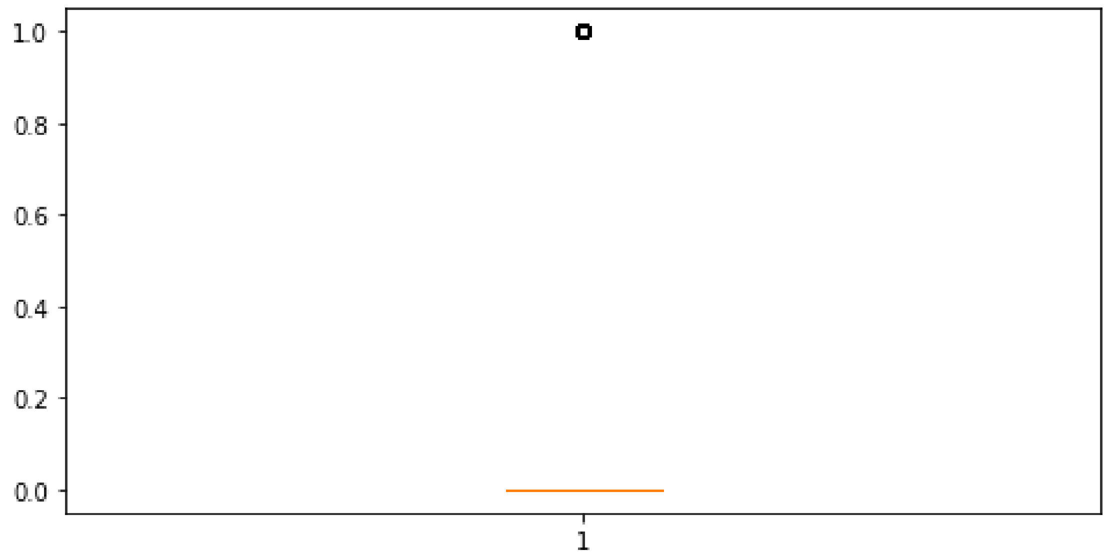
```
<matplotlib.collections.PathCollection at 0x7f7958734810>
```




```
#BOX PLOT
#Credit_card_data
plt.boxplot(credit_card_data.Class)
```

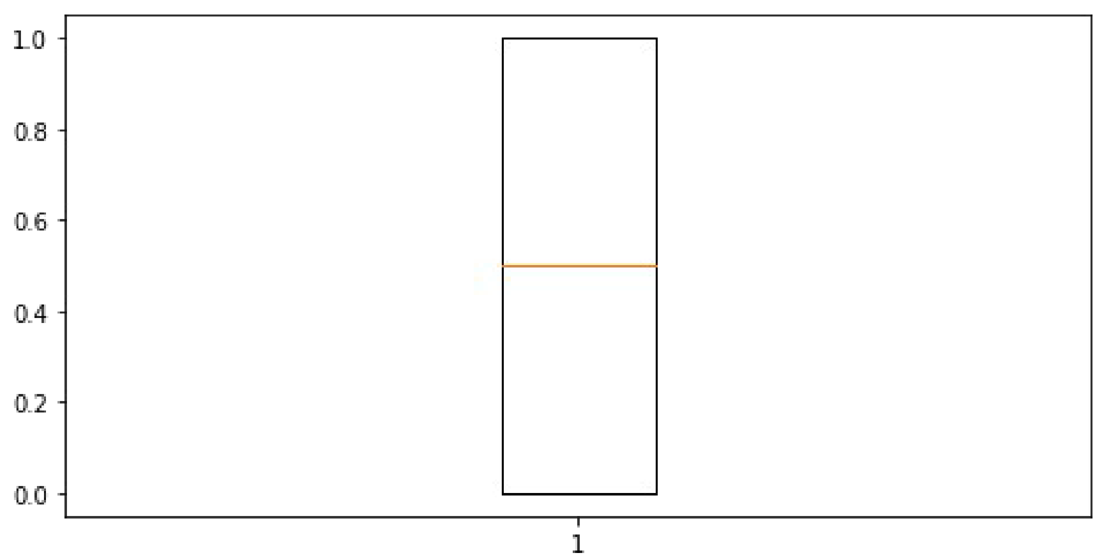
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f79586c4050>],
 'caps': [<matplotlib.lines.Line2D at 0x7f79586f3210>,
  <matplotlib.lines.Line2D at 0x7f79586f3750>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f79586f9250>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f79586f3cd0>],
```

```
 'whiskers': [<matplotlib.lines.Line2D at 0x7f79586ea750>,
  <matplotlib.lines.Line2D at 0x7f79586eac90>]}
```




```
#BOX PLOT
#new_dataset
plt.boxplot(new_dataset.Class)
```

```
{'boxes': [<matplotlib.lines.Line2D at 0x7f79586e27d0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f795865c250>,
  <matplotlib.lines.Line2D at 0x7f795865c790>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f7958663290>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f795865cd10>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f7958655790>,
  <matplotlib.lines.Line2D at 0x7f7958655cd0>]}
```



PREDICT A TRANSACTION IS FRADULANT OR NOT

```
input_data = (7,-0.89428608220282,0.286157196276544,-
0.113192212729871,-
0.271526130088604,2.6695986595986,3.72181806112751,0.370145127676916,0
.851084443200905,-0.392047586798604,-0.410430432848439,-
0.705116586646536,-0.110452261733098,-
0.286253632470583,0.0743553603016731,-0.328783050303565,-
0.210077268148783,-
0.499767968800267,0.118764861004217,0.57032816746536,0.052735669114969
7,-0.0734251001059225,-0.268091632235551,-
0.204232669947878,1.0115918018785,0.373204680146282,-
0.384157307702294,0.0117473564581996,0.14240432992147,93.2)
input_data_as_numpy_array = np.asarray(input_data)

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The transaction is not fradulant')
else:
  print('The transaction is fradulant')
```

```
[[-1.83638865  0.26809943 -0.40854912  0.54536024 -0.81113642
0.99290242
   2.49178649  0.53945503  0.11929346  0.40222062  0.54019003 -
0.92717709
   0.66003629 -0.18329757  0.77711329 -0.30092893  0.51954896
0.47029608
   0.51179774  0.18477125 -0.13710722 -0.1623053  -0.22317363 -
0.15236167
   1.91696581  0.53628856 -0.88728109 -0.07148573  0.23371939 -
0.05040324]]
[0]
The transaction is not fradulant

/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  "X does not have valid feature names, but"
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446:
UserWarning: X does not have valid feature names, but SVC was fitted
```

```
with feature names
  "X does not have valid feature names, but"
```

✳ Python packages

NumPy : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Use of NumPy in the dataset : for creating array

Pandas: Pandas is the most popular machine learning library written in python, for data manipulation and analysis.

In the dataset, Pd.read_csv() function imports CSV file to DataFrame format.

Matplotlib: a great library for Data Visualization.

Sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

✳ Supervised Learning Algorithms

Logistic Regression

```
#Model Training


#create variable as model
model = LogisticRegression()

#training the logistic regression model with training data

model.fit(X_train, Y_train)
#x_train contain all features of training data, y_train contain
corresponding labels (0 /1)
#this will fit data in logistic regression model and then we can make
some predictionsfrom it.

LogisticRegression()

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

print(confusion_matrix(Y_train, y_train_pred))
print(classification_report(Y_train, y_train_pred))
```

```
[[371  22]
 [ 41 353]]
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       393
           1       0.94      0.90      0.92       394

    accuracy                           0.92       787
   macro avg       0.92      0.92      0.92       787
weighted avg       0.92      0.92      0.92       787
```

```
print(confusion_matrix(Y_test, y_test_pred))
print(classification_report(Y_test, y_test_pred))
```

```
[[91  8]
 [14 84]]
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        99
           1       0.91      0.86      0.88        98

    accuracy                           0.89       197
   macro avg       0.89      0.89      0.89       197
weighted avg       0.89      0.89      0.89       197
```

Decision Tree

```
#Building Decision Tree Model
from sklearn.tree import DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(criterion = 'gini', max_depth = 20,
random_state=0)
dt_clf.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(max_depth=20, random_state=0)
```

Decision Tree Model Evaluation

```
print("Train Results")
pred_train = dt_clf.predict(X_train)

print(confusion_matrix(Y_train, pred_train))
print(classification_report(Y_train, pred_train))
```

```
Train Results
[[393   0]
 [  0 394]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       393
```

```
           1       1.00      1.00      1.00       394

    accuracy                           1.00       787
   macro avg       1.00      1.00      1.00       787
weighted avg       1.00      1.00      1.00       787
```

```
print("Test Results")
pred_test = dt_clf.predict(X_test)

print(confusion_matrix(Y_test, pred_test))
print(classification_report(Y_test, pred_test))
```

```
Test Results
[[90  9]
 [10 88]]
              precision    recall  f1-score   support

           0       0.90      0.91      0.90        99
           1       0.91      0.90      0.90        98

    accuracy                           0.90       197
   macro avg       0.90      0.90      0.90       197
weighted avg       0.90      0.90      0.90       197
```

Random Forest Classifier

```
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=345)

param_grid = {
    'n_estimators': [50],
    'max_depth' : [8,16,20]
}

rf_clf = RandomForestClassifier(n_estimators = 50,max_depth = 20,
                                random_state=345, verbose = 1)
rf_clf.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    0.2s finished

RandomForestClassifier(max_depth=20, n_estimators=50,
random_state=345,
                       verbose=1)
```

Random Forest Classifier - Model Evaluation

```
print("Train Results")
pred_train = rf_clf.predict(X_train)

print(confusion_matrix(Y_train, pred_train))
print(classification_report(Y_train, pred_train))
```

```
Train Results
[[393   0]
 [  0 394]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       393
           1       1.00      1.00      1.00       394

    accuracy                           1.00       787
   macro avg       1.00      1.00      1.00       787
weighted avg       1.00      1.00      1.00       787


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    0.0s finished
```

```
print("Test Results")
pred_test = rf_clf.predict(X_test)

print(confusion_matrix(Y_test, pred_test))
print(classification_report(Y_test, pred_test))
```

```
Test Results
[[95  4]
 [12 86]]
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        99
           1       0.96      0.88      0.91        98

    accuracy                           0.92       197
   macro avg       0.92      0.92      0.92       197
weighted avg       0.92      0.92      0.92       197


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    0.0s finished
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

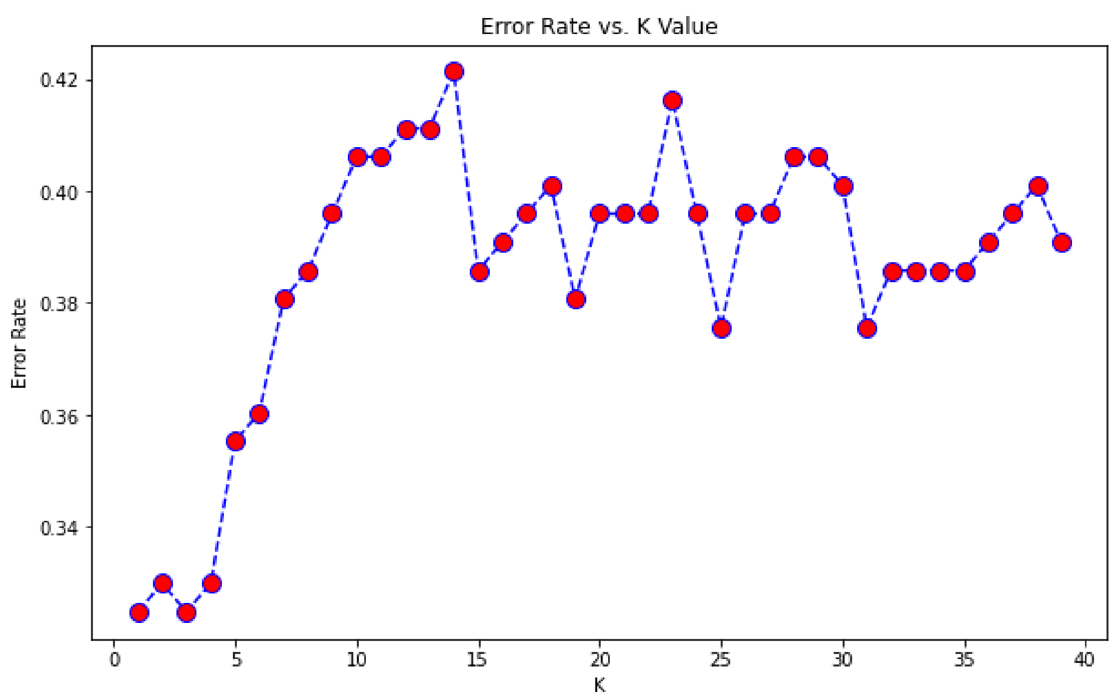
```
from sklearn.model_selection import cross_val_score
```

```
error_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,Y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != Y_test))

knn.fit(X_train,Y_train)
```

```
KNeighborsClassifier(n_neighbors=39)
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```




```
# NOW WITH K=30
knn = KNeighborsClassifier(n_neighbors=30)

knn.fit(X_train,Y_train)
pred = knn.predict(X_test)

print('WITH K=30')
print('\n')
```

```
print(confusion_matrix(Y_test,pred))
print('\n')
print(classification_report(Y_test,pred))
```

WITH K=30

```
[[63 36]
 [43 55]]
```

```
              precision    recall  f1-score   support

           0       0.59      0.64      0.61        99
           1       0.60      0.56      0.58        98

    accuracy                           0.60       197
   macro avg       0.60      0.60      0.60       197
weighted avg       0.60      0.60      0.60       197
```

✴ Unsupervised Learning Algorithms

K-Mean Clustering

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import
confusion_matrix,classification_report,accuracy_score,roc_curve,roc_au
c_score,f1_score, precision_score,  recall_score, cohen_kappa_score

kmeans=KMeans(n_clusters=2,random_state=0,algorithm="elkan",max_iter=1
0000)
kmeans.fit(X_train)
kmeans_predicted_train_labels=kmeans.predict(X_train)

print("tn --> true negatives")
print("fp --> false positives")
print("fn --> false negatives")
print("tp --> true positives")
tn,fp,fn,tp=confusion_matrix(Y_train,kmeans_predicted_train_labels).ra
vel()
reassignflag=False
if tn+tp<fn+fp:
     # clustering is opposite of original classification
     reassignflag=True
kmeans_predicted_test_labels=kmeans.predict(X_test)
if reassignflag:
```

```
kmeans_predicted_test_labels=1-kmeans_predicted_test_labels
#calculating confusion matrix for kmeans
tn,fp,fn,tp=confusion_matrix(Y_test,kmeans_predicted_test_labels).rave
l()
#scoring kmeans
kmeans_accuracy_score=accuracy_score(Y_test,kmeans_predicted_test_labe
ls)
kmeans_precison_score=precision_score(Y_test,kmeans_predicted_test_lab
els)
kmeans_recall_score=recall_score(Y_test,kmeans_predicted_test_labels)
kmeans_f1_score=f1_score(Y_test,kmeans_predicted_test_labels)
kmeans_kappa_score=cohen_kappa_score(Y_test,kmeans_predicted_test_labe
ls)
fpr, tpr, _=roc_curve(Y_test,kmeans_predicted_test_labels)
kmeans_auc=roc_auc_score(Y_test,kmeans_predicted_test_labels)
#printing
print("")
print("K-Means")
print("Confusion Matrix")
print("tn =",tn,"fp =",fp)
print("fn =",fn,"tp =",tp)
print("Scores")
print("Accuracy -->",kmeans_accuracy_score)
print("Precison -->",kmeans_precison_score)
print("Recall -->",kmeans_recall_score)
print("F1 -->",kmeans_f1_score)
print("Kappa -->",kmeans_kappa_score)
print('Area Under Curve:',kmeans_auc)
print("fpr = ", fpr , "tpr = ", tpr)
```

```
tn --> true negatives
fp --> false positives
fn --> false negatives
tp --> true positives
```

```
K-Means
Confusion Matrix
tn = 50 fp = 49
fn = 33 tp = 65
Scores
Accuracy --> 0.583756345177665
Precison --> 0.5701754385964912
Recall --> 0.6632653061224489
F1 --> 0.6132075471698113
Kappa --> 0.1681771369721936
Area Under Curve: 0.584157905586477
fpr =  [0.         0.49494949 1.        ] tpr =  [0.
0.66326531 1.        ]
```

Gaussian Mixture Model Clustering

```
from sklearn.mixture import GaussianMixture

gm= GaussianMixture(n_components=2, n_init=10,
covariance_type='spherical')
gm.fit(X_train)

GaussianMixture(covariance_type='spherical', n_components=2,
n_init=10)

gaussian_predicted_train_labels= gm.predict(X_train)

print("tn --> true negatives")
print("fp --> false positives")
print("fn --> false negatives")
print("tp --> true positives")
tn,fp,fn,tp=confusion_matrix(Y_train,gaussian_predicted_train_labels).
ravel()
reassignflag=False
if tn+tp<fn+fp:
     # clustering is opposite of original classification
     reassignflag=True
gaussian_predicted_test_labels=gm.predict(X_test)
if reassignflag:
     gaussian_predicted_test_labels=1-gaussian_predicted_test_labels
#calculating confusion matrix for kmeans
tn,fp,fn,tp=confusion_matrix(Y_test,gaussian_predicted_test_labels).ra
vel()
#scoring kmeans
gaussian_accuracy_score=accuracy_score(Y_test,gaussian_predicted_test_
labels)
gaussian_precison_score=precision_score(Y_test,gaussian_predicted_test
_labels)
gaussian_recall_score=recall_score(Y_test,gaussian_predicted_test_labe
ls)
gaussian_f1_score=f1_score(Y_test,gaussian_predicted_test_labels)
gaussian_kappa_score=cohen_kappa_score(Y_test,gaussian_predicted_test_
labels)
fpr, tpr, _=roc_curve(Y_test,gaussian_predicted_test_labels)
gaussian_auc=roc_auc_score(Y_test,gaussian_predicted_test_labels)
#printing
print("")
print("gaussian")
print("Confusion Matrix")
print("tn =",tn,"fp =",fp)
print("fn =",fn,"tp =",tp)
print("Scores")
print("Accuracy -->",gaussian_accuracy_score)
print("Precison -->",gaussian_precison_score)
print("Recall -->",gaussian_recall_score)
print("F1 -->",gaussian_f1_score)
print("Kappa -->",gaussian_kappa_score)
```

```
print('Area Under Curve:',gaussian_auc)
print("fpr = ", fpr , "tpr = ", tpr)
```

```
tn --> true negatives
fp --> false positives
fn --> false negatives
tp --> true positives
```

```
gaussian
Confusion Matrix
tn = 45 fp = 54
fn = 30 tp = 68
Scores
Accuracy --> 0.5736040609137056
Precison --> 0.5573770491803278
Recall --> 0.6938775510204082
F1 --> 0.6181818181818182
Kappa --> 0.14823965410747375
Area Under Curve: 0.5742115027829313
fpr =  [0.         0.54545455 1.        ] tpr =  [0.
0.69387755 1.        ]
```