Trent University

A Report on

Credit Fraud Detection

Under the subject of
Data Mining
M.E., Semester-I
(Big Data Analytics)

Submitted By

Sr.	Name of Student	Enrolment No.
1.	Aakash Patel	0700763
2.	Nisarg Patel	0692153
3.	Malav Joshi	0703330

Guided By
Sabine McConnell
Instructor

Academic Year
2021-2022

PROJECT WRITE UP

Group Members :

1) **AAKASH PATEL : 0700763**
2) **NISARG PATEL : 0692153**
3) **MALAV SHAH : 0703330**

Dataset Name : CREDIT FRAUD DETECTION

Abstract :

Frauds in credit card transactions are common today as most of us are using the credit card payment methods more frequently. This is due to the advancement of Technology and increase in online transactions resulting in frauds causing huge financial loss. Therefore, there is a need for effective methods to reduce the loss. In addition, fraudsters find ways to steal the credit card information of the user by sending fake SMS and calls, also through masquerading attack, phishing attack and so on. This paper aims in using the multiple algorithms of Machine learning such as support vector machine (SVM), Decision Trees in predicting the occurrence of fraud.

Overview of Credit Card Fraud Detection :

Fraud is an offensive activity, carried out by an unauthorized person by cheating innocent. Credit card fraud involves stealing the essential credentials from the cardholder and using it unauthorized manner by the fraudsters either by using phone calls or SMS. This fraud in credit cards may also happen using some software applications that are under the control of fraudsters. The credit card fraud detection takes place as: the user or the customer enters the necessary credentials in order to make any transaction using the credit card and the transaction should get approved only upon being checked for any fraud activity. For this to happen, we first pass the transaction details to the verification module where it is classified under fraud and non-fraud categories. Any transaction that is put under fraud category is rejected. Otherwise, the transaction gets approved.

Problem Statement :

Now -a-days, most of them are using credit cards for buying the goods which are so much in need but can't afford at the moment. In order to meet the needs credit cards are used and the fraud associated with it is also increasing so there is a need of developing a model that fits well and predicts at higher accuracy.

Objectives :

The main objective of this project is to find fraudulent transactions in credit card transactions and comparison between the different machine learning algorithms to check which of the algorithms used in this project shows promising results and generates the best possible output. The comparison is done on the basis of the accuracy of each model.

DATA EXPLORATION :

Due to the Data Privacy of sensitive payment data, which contains private information about customers and businesses, access to such data is highly restricted, making it very difficult to find a publicly available real-world data set.

- There are 31 features out of which 28 features (V1, V2, ... V28) are numerical input variables resulting from a PCA transformation. The other two features, 'Amount' and 'Time,' are not PCA transformed.
- Feature 'Time' holds the seconds elapsed between each transaction and the first transaction in the dataset.
- Feature 'Class' is the target variable that takes value 1 in fraudulent transactions and 0 otherwise.

Packages and libraries used :

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Scikit Learn

Here the group members have performed individual analysis on this dataset using different machine learning algorithms and have at the end compared the accuracy of the models to check which machine learning algorithm works well and is capable of finding the fraudulent transactions.

From here on we have created four sections below, in the first three we have discussed the work performed by each of the individual team members and at the end is the conclusion.

We have discussed our individual work separately because it was difficult for us to integrate the all the work in one single python file because each one of us have used different pre-processing techniques , different machine learning models and different plots so it will become very clumsy to understand so we have performed the task individually and then at the end all of our work has been explained in this single document.

Data Analysis performed by Aakash Patel:

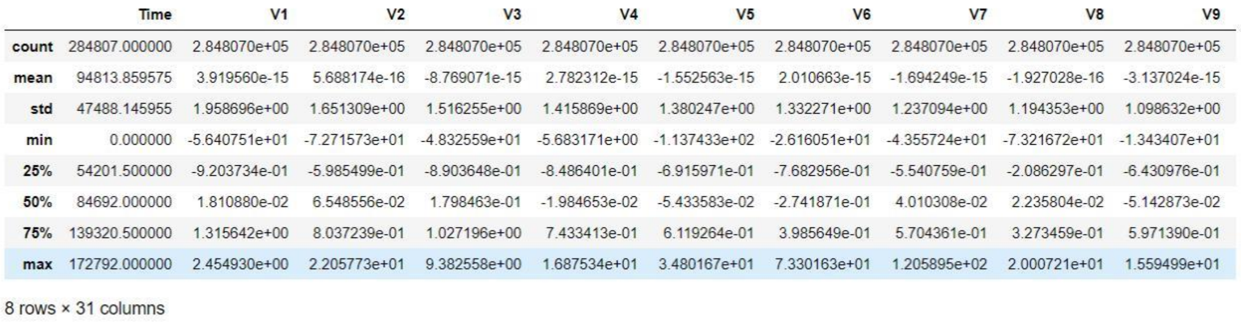
The analysis is started firstly by importing the basic libraries which are necessary for every data analysis project which are as follows.

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

These libraries are the most basic ones. Numpy is used to work with multi-dimensional arrays and matrices. Pandas is a data manipulation library which is used to read the data file , whether it be csv, json or any other format. It is used to convert that file into a dataframe and then the further operations can be carried out on that file , and it has many more functions. Matplotlib and seaborn are data visualization libraries which are used to create beautiful grapes and plots which makes the understanding of the data easy for the user.

```
dataset = pd.read_csv("F:\\TRENT COURSE\\DM\\PROJECT\\creditcard.csv")
dataset.head(5)
dataset.info()
dataset.describe()
```

The file is read from its storage location and is saved into the dataframe named dataset , and the dataset.head() function allows us to view the first 5 rows of the dataset.

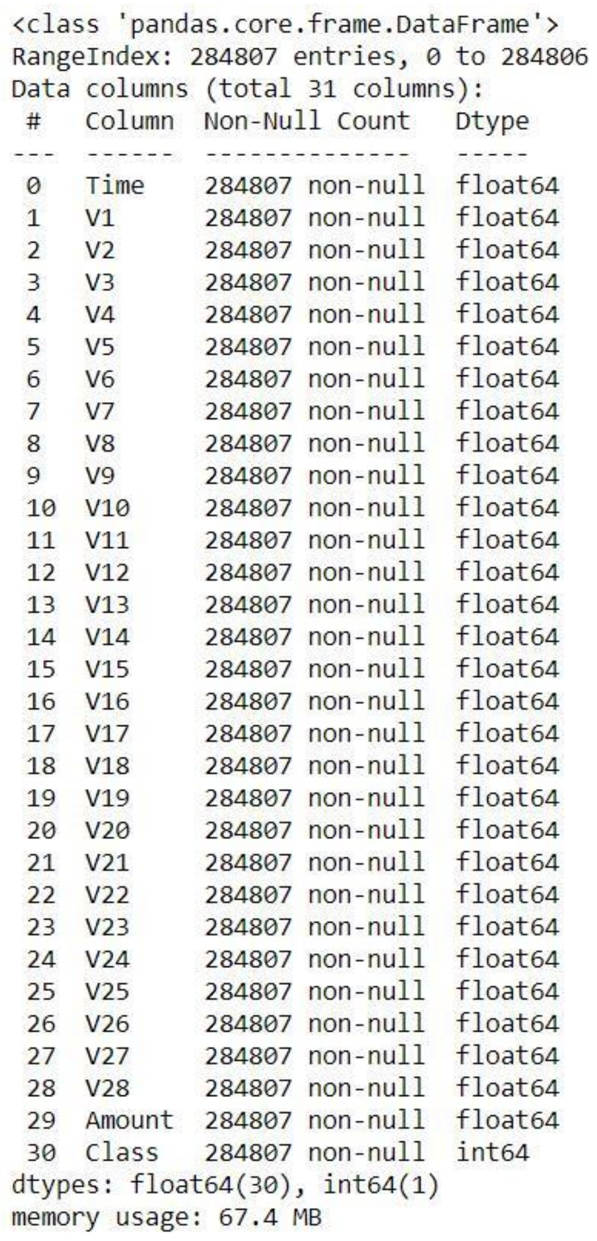


	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

8 rows × 31 columns

As you can see there are in total 8 rows and 31 columns but the image is incomplete as there are more columns the image can be captured as a single image.

Then the dataset.info() function is used to get the information about the data in the dataset.



```
<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
```

As we can see there are a total of 28406 entries in total which is an extremely huge amount of data. Also we can see there are a total 31 columns namely , Time , V1 - V28 , Amount and Class. Also besides those columns name it also shows that if there is any null value in those columns , and we can see it shows non-null which means that none of the columns have null values so we don’t need to do any preprocessing to eliminate the null values. And at the end the last column shows the type of data in those columns.

Next we have an iloc function which is used for slicing or indexing. We have used this function here to see the shape of the x and y.

```
x = dataset.iloc[:,1:30].values
y = dataset.iloc[:,30].values
print("Input Range : " , x.shape)
print("Output Range : " , y.shape)
print("Class Labels : \n",y)
```

Here the iloc function is used on the class column which contains the values 0 and 1. 0 means that the transaction is not fraudulent while 1 means that it is fraudulent. So for the X we have used all the values from 1 to 30 while for the Y we have used just the entire class columns.

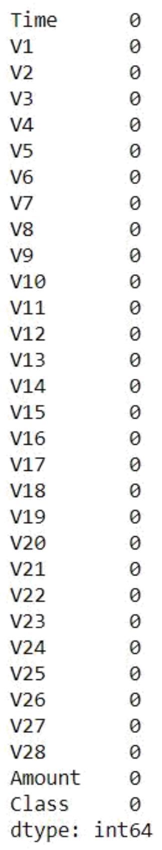
```
Input Range : (284807, 29)
Output Range : (284807,)
Class Labels :
[000...000]
```

So front the above output we can determine that class labels show 0 and 1 values. As the number of columns as more 1 is not visible.

So to view this perfectly we can plot it on a graph to have a better understanding of this.

Before that lets check if the dataset has any null values or not.

```
null = dataset.isna().sum()
print(null)
```



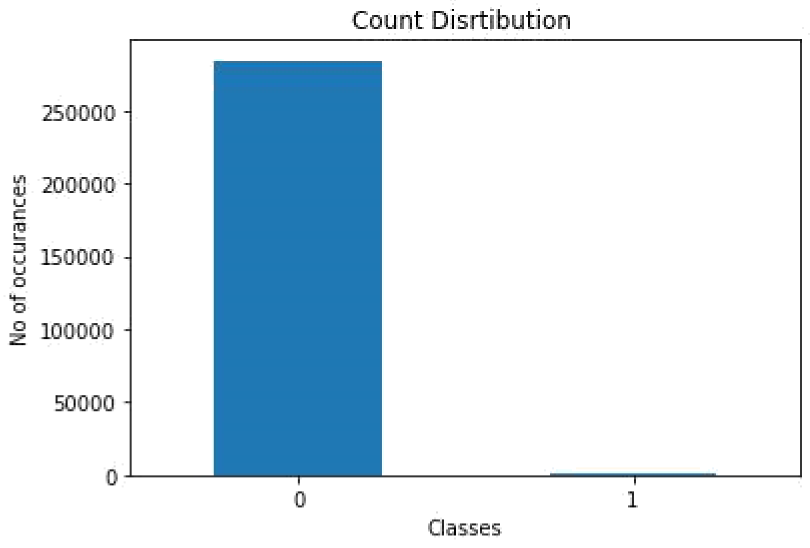
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

As we can see in the output that the dataset does not have any null values.

Now we can go forward and construct our first plot. Checking the dataset for null values is extremely important because having null values in the dataset can give wrong visuals and it can lead to improper data analysis.

```
set_class = pd.value_counts(dataset['Class'] , sort = True)
set_class.plot(kind = 'bar' , rot =0)
plt.title("Count Distribution ")
plt.xlabel("Classes")
plt.ylabel("No of occurances")
plt.xticks(range(2))
plt.show()
```

Now we have constructed a barplot to check there are how many legit transactions and how many fraud transactions.



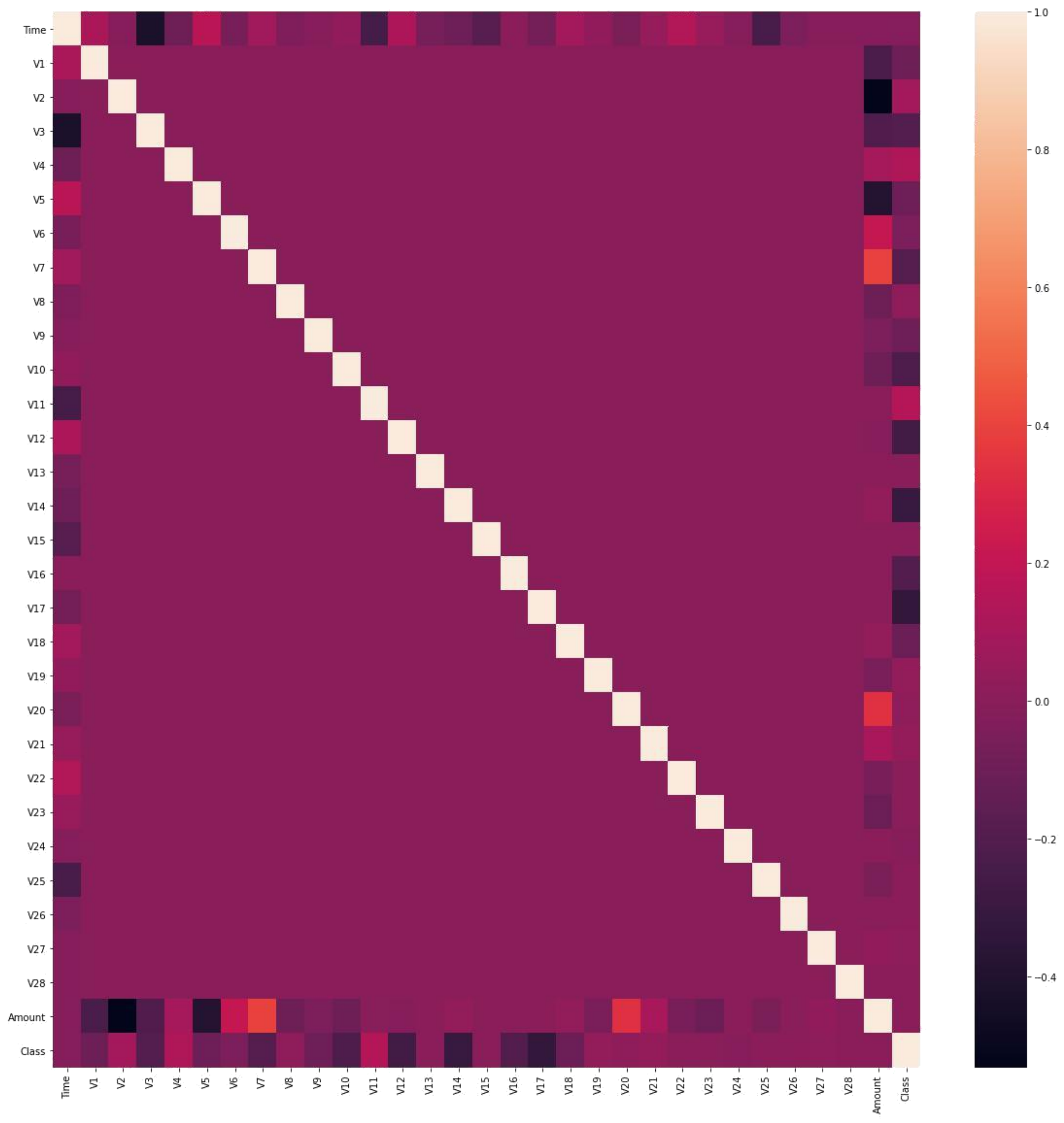
So from the graph above we can see the number of fraudulent cases are way less as compared to the number of fraud cases.

Next I have plotted a heatmap to find the correlation between all the rows.
For using heatmap to find correlation we need to make sure that all the values are of numeric type , and as we checked above that we don't have any categorical values so heatmap can be used here to find the correlation. The correlation is used to measure the dependencies between the two variables. It also measures how the variables move together and how strongly they are related to each other. It helps in deeper understanding of the data.

In the correlation plot we can see the values are correlated to each other. The white line in the between indicates stronger correlation and all the other color indicates weaker correlation.

We can identify the correlation between the variable form the bar on the right. As the color gets darker the correlation becomes weaker and as the color gets darker the correlation becomes stronger.

```
dataset.hist(figsize = (20, 20))
plt.show()
```

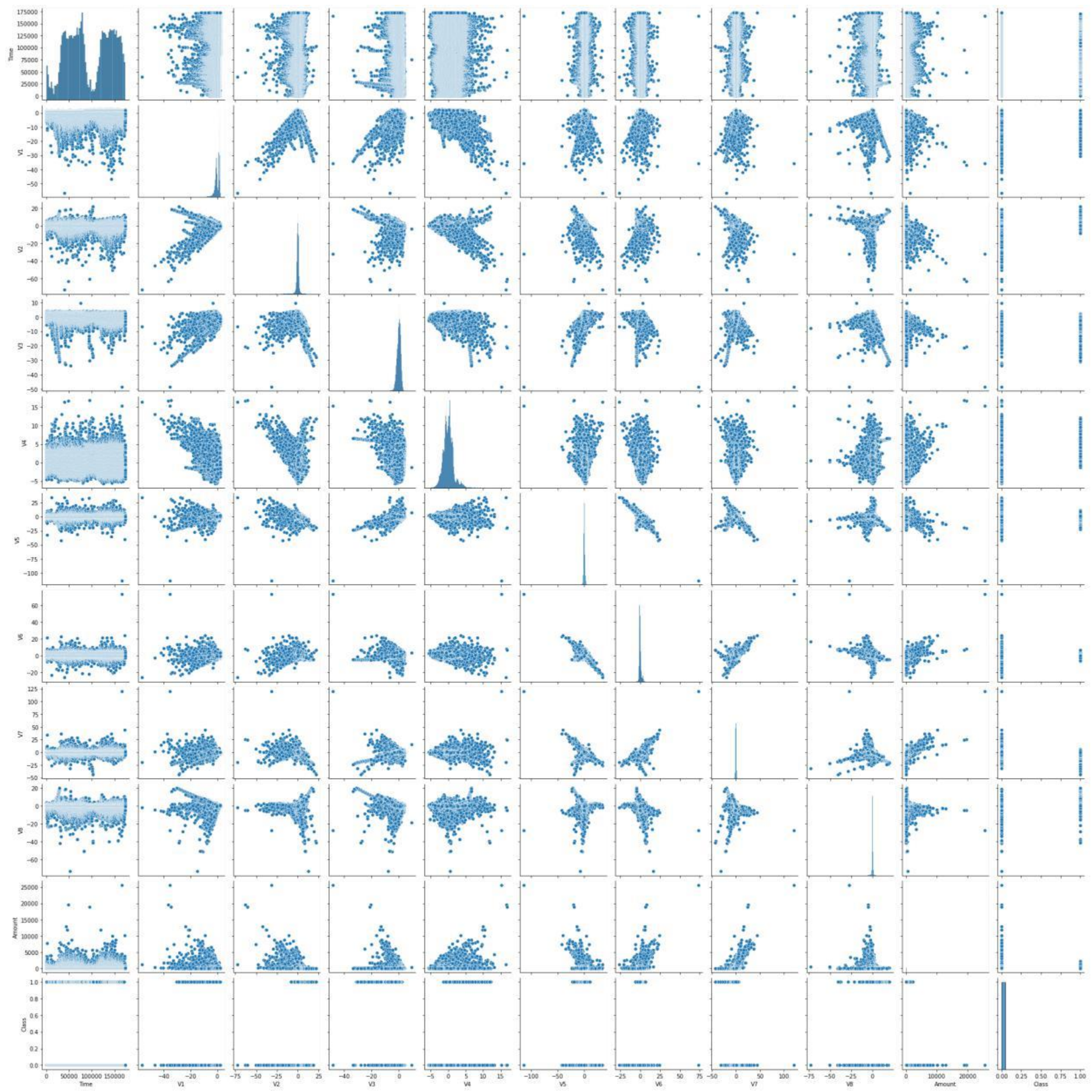


Now we can use pariplot over this to get an even better understanding of the data. Pairplot is like the above correlation matrix plot only but the major difference is that unlike the above heatmap in pairplot a plot is constructed for each of the variables. Pairplot is used to plot multiple pairwise bivariate distributions in a dataset. This shows the relationship for (n, 2) combination of variables in a DataFrame as a matrix of plots and the diagonal plots are the univariate plots.

```
colss = ['Time' , 'V1' , 'V2' , 'V3' , 'V4', 'V5' , 'V6', 'V7' , 'V8' , 'Amount' ,
'Class'] sns.pairplot(dataset[colss] , height = 2.5) plt.show()
```

Here, I have used just V1 to V8 for constructing the pariplot because if I take all the rows till V28 then it will become very clumsy and it will be difficult for us to understand the correlation and derive some results.

The height 2.5 indicates the height of each individual plot.



So the beautiful pairplot is constructed and it is easier to derive a better understanding of the relationship and dependencies between the two variables.

Now we will derive some valuable information about the fraud and the normal data.

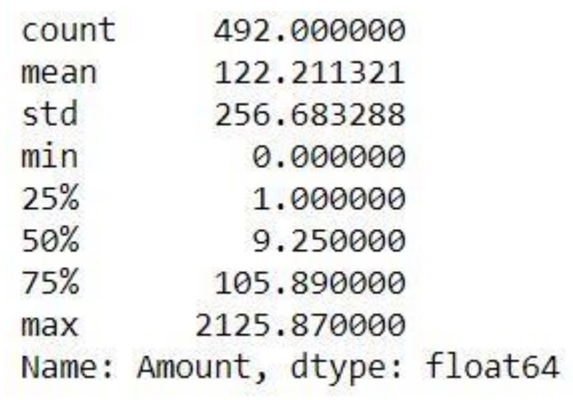
```
fraud_data = dataset[dataset["Class"] == 1]
normal_data = dataset[dataset["Class"] ==
0] print(fraud_data.shape ,
normal_data.shape)
fraud_data.Amount.describe()
```

We will first count the fraud and the normal data and store them in the variable fraud_data and normal_data respectively and then we will print out the shape of the data.

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

So this is the output of it which shows that there are 492 fraudulent data entries and 284315 normal data entries.

Now we will use the describe function on the fraudulent data on the amount column to check how much amount is spent when it comes to the fraudulent cases.



```
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
```

The above output shows that there are 492 cases of fraudulent data , the average amount spent by the fraudsters is 122.211 and the minimum and maximum fraud amount is 0 and 2125 respectively.

Now after all the necessary analysis now it's time to develop our machine learning model.
I have developed the Decision Tree Model to capture the fraudulent data.

But first in order to run the decision tree model we need to first split our data into two halves. Train data and Test data. Train data is used to train our machine learning model , while the test data is used to test our machine learning model , and check how well our model is performing.

So to perform the train and test split on our data , we will be using sklearn. Sklearn has an inbuilt library to split the data in to train and test the dataset.

```
from sklearn.model_selection import train_test_split
```

```
xtrain , xtest , ytrain , ytest = train_test_split(x, y , test_size = 0.30 , random_state = 0)
```

Now we have to specify the xtrain , xtest , ytrain and ytest , so that the model is able to select the value for training and testing and also to split the data into the two categories.
The test size is set to 0.3 which means 30 % of the data will be used for testing our machine learning model and the random state is set to 0 because we want the train_test_split library to randomly select the data for training and testing.

Now let's look at the shape of the training and testing variables .

```
print('xtrain.shape :' , xtrain.shape)
print("xtest.shape :" , xtest.shape)
print("ytrain.shape :" , ytrain.shape)
print("ytest.shape : " , ytest.shape)
```

This above code prints out the shape of the train and test data. Below is the output of the above code.

```
xtrain.shape : (199364, 29)
xtest.shape : (85443, 29)
ytrain.shape : (199364,)
ytest.shape : (85443,)
```

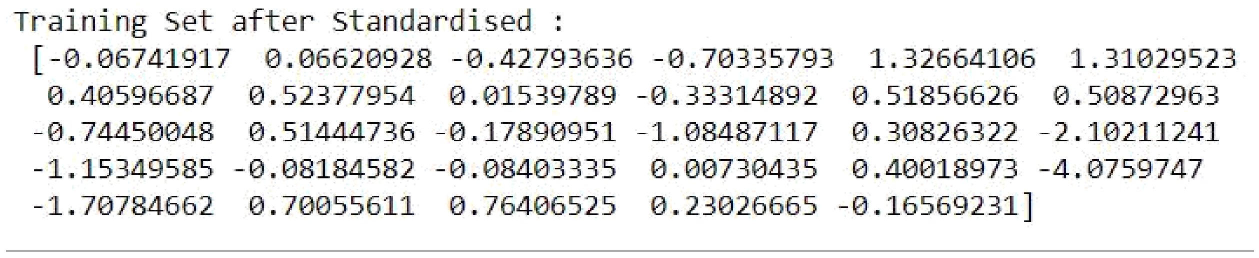
Now as we can see in the above output that for training 199364 entries are used while the test data counts to 85443.

Now the next step will be to scale our data as the data is highly imbalanced.
So to scale our data there is a library available for this as well in the sklearn library.

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
xtrain = stdsc.fit_transform(xtrain)
xtest = stdsc.transform(xtest)
```

print("Training Set after Standardised : \n" , xtrain[0])

The StarndardScaler library is used to scale the data while the fit_transform and transform functions are used on train and test data respectively. And after that the train data is printed.



```
Training Set after Standardised :
 [-0.06741917  0.06620928 -0.42793636 -0.70335793  1.32664106  1.31029523
  0.40596687  0.52377954  0.01539789 -0.33314892  0.51856626  0.50872963
 -0.74450048  0.51444736 -0.17890951 -1.08487117  0.30826322 -2.10211241
 -1.15349585 -0.08184582 -0.08403335  0.00730435  0.40018973 -4.0759747
 -1.70784662  0.70055611  0.76406525  0.23026665 -0.16569231]
```

So we can see in the output that our data is scaled and all the data is on the same scale.

Now we will run our decision tree model.
For that first we need to import the decision tree model from sklearn package.

from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(criterion = 'entropy' , random_state = 0)
dt_classifier.fit(xtrain , ytrain)
y_pred_decision_tree = dt_classifier.predict(xtest)
print("y_pred_decision_tree : \n" ,y_pred_decision_tree)

So the DecisionTreeClassifier has been imported and is run on the train data. And then the predicted output is printed.

y_pred_decision_tree :
[000...000]

The output of the decision tree model is above.

Now we don’t know the accuracy of the model, accuracy is an important factor to determine how good our model is and is also used to compare our model with other machine learning models . So to check the accuracy of the model we need to construct a confusion matrix.

from sklearn.metrics import confusion_matrix
con_decision = confusion_matrix(ytest , y_pred_decision_tree)
print("confusion matrix : \n" , con_decision)

```
accuracy = ((con_decision[0][0] + con_decision[0][1]) / con_decision.sum())*100
print(accuracy)
error_rate = ((con_decision[0][1] + con_decision[1][0]) / con_decision.sum())*100
print(error_rate)
```

So from sklearn.metrics , confusion_matrix is imported.
The result of the confusion matrix is stored into a variable con_decesion and then the accuracy and the error rate is calculated and printed out.

```
confusion matrix :
[[85268    28]
 [   39  108]]
99.82795547909133
0.07841484966585911
```

So from the output we can see that the accuracy of the decision tree model is 99.827 and the error rate is 0.07. So this says that the model is very much accurate and it is able to correctly detect the fraudulent data.

Data Analysis performed by Nisarg Patel:

In the starting of program, I have included very basic but powerful libraries like numy and pandas. Pandas allow users to import data from various file format like JSON, CSV, database etc. I have imported creditcard dataset in this program with the help of read_csv which is one of the features of pandas library. Other than that, it also allows users to perform different data manipulation like reshaping, selecting , data cleaning and many more.
I have used numpy library because it increase the performance and make execution much more faster. It also allow us to store values of same data type and which help us to perform math operation on arrays easier.
Seaborn is a library in the python which have features to make statistical data in to graphical data in a way that it easily understandable. It provide many functionalities like to calculate automatic estimation, plotting the graphs and to find relationship between variables. matplotlib library also provide same kind of features.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

import seaborn as sns
from sklearn.model_selection import train_test_split

data = pd.read_csv("C:\\\\Users\\\\dell\\\\Desktop\\\\creditcard.csv")

Before starting any analysis on datset, we need some information about data like how the data is distributed, what kind of data that we working on. So here I have print the first 5 row of dataset. So we have total 31 columns in our data.

print("First 5 row of our data :"+"\n",data.head())

```
First 5 row of our data :
    Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -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
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

Columns is used to print all the columns of our dataset so we can see we have 31 columns on our data in which class is our target variable.

print("Total Number of column : "+"\n",data.columns)

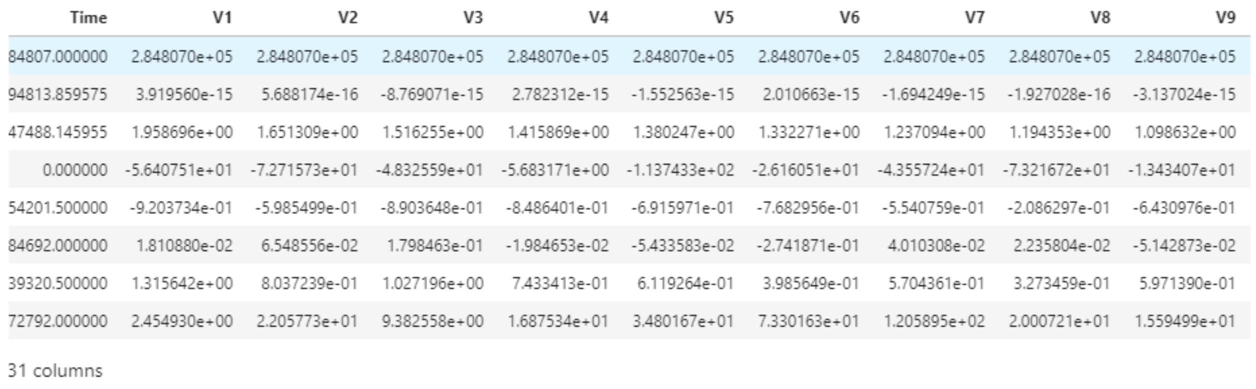
```
Total Number of column :
 Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

print(data.info())
From data.info, we can see the index for all the columns, then name of the columns. We can see how many row of data we have in that columns and their type of data.

```
<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
```

print(data.describe())

Describe() is one of the features of pandas library and it provides basic statistical details like standard deviation, mean, 3 percentile, then minimum and maximum value of data



Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
84807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-15
47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
39320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
72792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

31 columns

print(data['Class'].describe())

Now, I have basic understanding of the data and I want to see how data is distributed in our target variable. So I used describe for that and I saw that data is not distributed normally because class is a categorical variable and have only two value: 0 and 1. So, if data is distributed normally then we should have mean around 0.5 which is very far away from 0.001. So, we will further visualize it for better understanding.

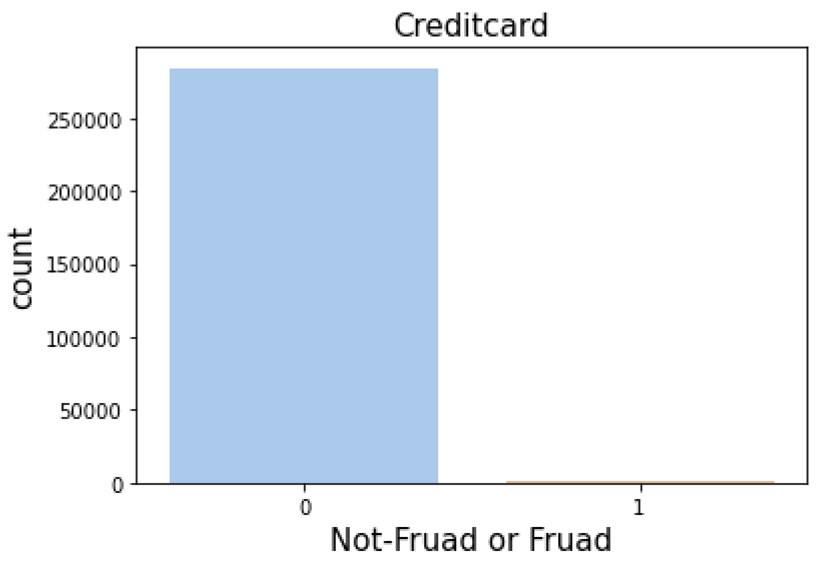
```
count    284807.000000
mean          0.001727
std           0.041527
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: Class, dtype: float64
```

print("Value count for Class attribute : ","\n",data['Class'].value_counts())

```
Value count for Class attribute :
 0    284315
1       492
```

As we know class attribute is not normally distributed. So, now I tried to calculate the values for both attributes of class. And I found that count of 1 has only 492 sample which is 577 times smaller than count of 1.

```
#visulising the data

sns.countplot(x=data['Class'], palette = 'pastel')
plt.title('Creditcard', fontsize = 15)
plt.xlabel('Not-Fruad or Fruad', fontsize = 15)
plt.ylabel('count', fontsize = 15)
```



After class attribute, I performed same features to the time column to check how the data is distributed in this and I found that time columns is not much have impact on the predicting class variable. Most of the time is unique and have no useful patterns inside that.

```
print(data['Time'].describe())
```

```
count    284807.000000
mean      94813.859575
std       47488.145955
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: Time, dtype: float64
```

```
print("Time : "+"\n",data['Amount'].value_counts())
```

```
Name: Class, dtype: int64
Time :
 1.00       13688
1.98        6044
0.89        4872
9.99        4747
15.00        3280
           ...
192.63         1
218.84         1
195.52         1
793.50         1
1080.06        1
Name: Amount, Length: 32767, dtype: int64
```

Now, most important thing in any data analytics to look for a null data and if we have null values in our dataset then we need to pre-process them like there are various techniques to handle missing values. For example, we can remove them directly or we can replace this values with other values or put average value in the null value and many more.
But luckily, we don't have any null values in our dataset which saves lots of times.

```
#looking for null values in the data
print(data.isnull().sum()) #there is no null values in the data
```

```
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
```

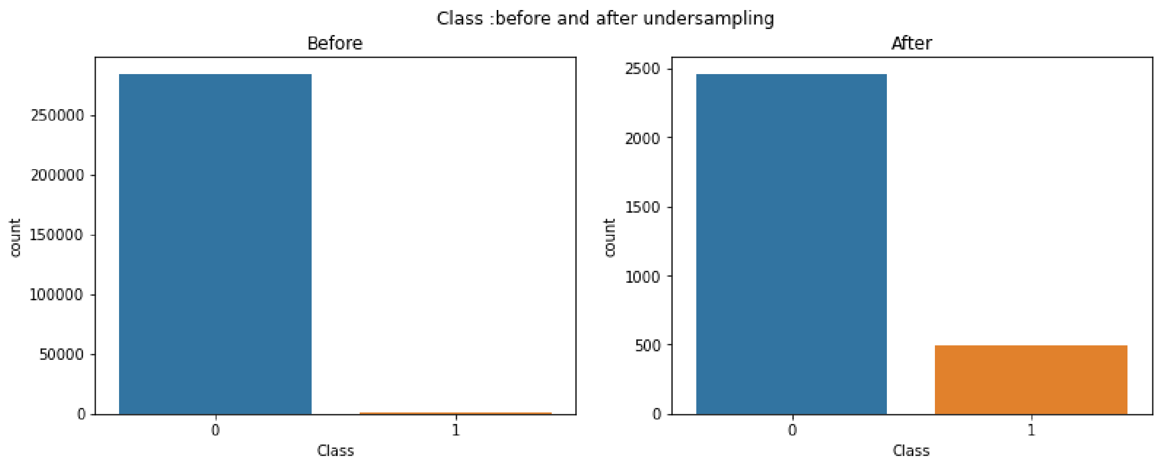
```
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Now, after getting some information of data, I have scaled the data with standardscaler which is used to remove the mean and scales of each variable to unit variance. I have dropped the column named “Time” as we know it does not have much importance in creating our model. I have performed standard scaler on amount and create a new column named std_amount in dataset. So, I removed the ‘Amount’ column from the database.

```
from sklearn import preprocessing

data = data.drop(['Time'],axis = 1)
scaler = preprocessing.StandardScaler()

#standard scaling
data['std_Amount'] = scaler.fit_transform(data['Amount'].values.reshape (-1,1))

#removing Amount
data = data.drop("Amount", axis=1)
```

From the above information we see that the distribution of fraud and notfraud data are highly imbalanced. So, we should not directly create a model because even if we get high accuracy for our model, it will introduce bias because it will take more sample from one class which have more frequency.
For that, I have used random under sampling technique:
There are some techniques for under sampling like random undersampling, cluster, tomek links etc. in which I have selected random under sampling which will remove the samples from majority class with or without replacement. In out case, it will remove samples from class 0.

```
# so we need to perform undersampling technique :
import imblearn
from imblearn.under_sampling import RandomUnderSampler


undersample = RandomUnderSampler(sampling_strategy=0.2)

cols = data.columns.tolist()
cols = [c for c in cols if c not in ["Class"]]
target = "Class"
```



X = data[cols]
Y = data[target]

#undersample

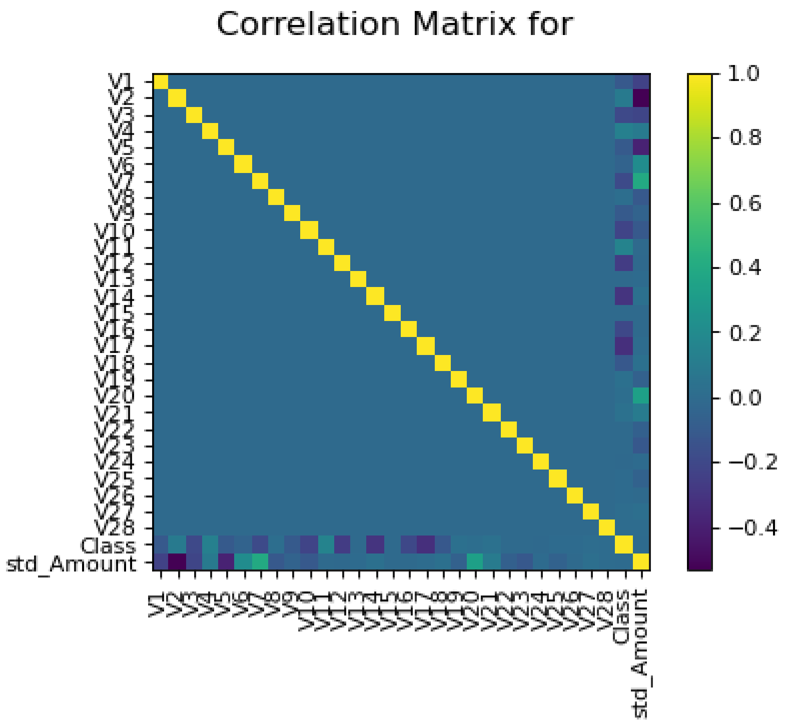
from pandas import DataFrame
X_under, Y_under = undersample.fit_resample(X, Y)
test = pd.DataFrame(Y_under, columns = ['Class'])

I have compared the result of under sampling and data that we have previously and we can find noticeable difference between them.

#visualizing undersampling results
fig, axs = plt.subplots(ncols=2, figsize=(13,4.5))
sns.countplot(x="Class", data=data, ax=axs[0])
sns.countplot(x="Class", data=test, ax=axs[1])

fig.suptitle("Class :before and after undersampling")
a1=fig.axes[0]
a1.set_title("Before")
a2=fig.axes[1]
a2.set_title("After")

corr = data.corr()
plt.figure(num=None, dpi=80, facecolor='w', edgecolor='k')
corrMat = plt.matshow(corr, fignum = 1)
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.columns)), corr.columns)
plt.gca().xaxis.tick_bottom()
plt.colorbar(corrMat)
plt.title('Correlation Matrix for ', fontsize=15)



I have used correlation matrix to check the changes between two variables. We can get the idea about which variable have a high or low correlation in respect to other variable. In our case, from the output, it can be clearly seen that output of correlation matrix is symmetrical by seeing that output of bottom and top is identical and each variable is positively correlated with each other.

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_under, Y_under, test_size=0.2, random_state=1)

Now, I have completed the preprocessing of data. So I will make model but before making model from we have to split our data in to train and test dataset. From the train data we will create a model and with the help of test dataset we will check the accuracy of our model**.**

from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import precision_recall_curve

For our model, I have selected multilayer perceptron known ad MLP, which is a feedforward artificial nueral network. It generate a set of output from a setoff inputs.
MLP have several layers of input node inside it and these input node are connected as a directed graph between input and output layers. MLP use backpropogation for training the data for network. If our dataset have class or label and we have a classification predict problem then we can use this algorithm.

One of the most important thing to check while preproccesing is to indentify any outliners in the dataset. There are many techniques available to identify it. I have used 3 techniques to identify it which are Isolation forest, ellipticenvelope and localoutlinerfactor and compared the result for it. But result are not good so I decide to make model without them**.**

```
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor

# identify outliers in the training dataset
#lof = LocalOutlierFactor()
#yhat = lof.fit_predict(X_train)
#ee = EllipticEnvelope(contamination=0.01)
#yhat = ee.fit_predict(X_train)
#iso = IsolationForest(contamination=0.1)
#yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
#mask = yhat != -1
#X_train, y_train = X_train.iloc[mask, :], y_train.iloc[mask]
# summarize the shape of the updated training dataset
#print(X_train.shape, y_train.shape)

#train the model
```

‘lbfgs’ is an optimizer in the family of quasi-Newton methods.

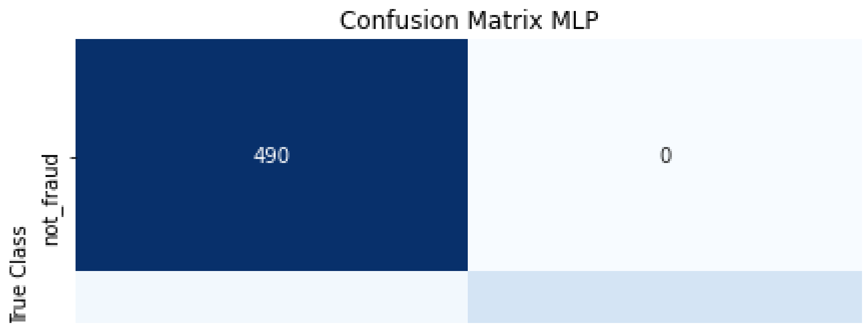
```
model = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(100,100), random_state=2)
mlp = model.fit(X_train, y_train)

print(model.get_params(deep=True))

#predictions
y_pred_mlp = model.predict(X_test)
```

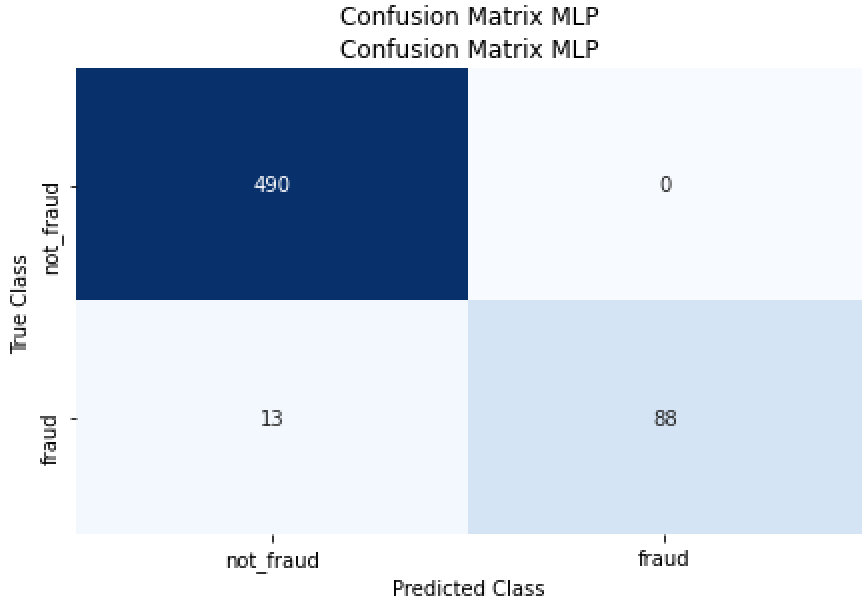
output:

```
{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999,
'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': (100, 100), 'learning_rate':
'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 200, 'momentum': 0.9,
'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': 2, 'shuffle':
True, 'solver': 'lbfgs', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}
```

```
#scores
print("Accuracy MLP:",metrics.accuracy_score(y_test, y_pred_mlp))
print("Precision MLP:",metrics.precision_score(y_test, y_pred_mlp))
print("Recall MLP:",metrics.recall_score(y_test, y_pred_mlp))
print("F1 Score MLP:",metrics.f1_score(y_test, y_pred_mlp))
```

```
Accuracy MLP: 0.9780033840947546
Precision MLP: 1.0
Recall MLP: 0.8712871287128713
F1 Score MLP: 0.9312169312169313
```





olumns=['not_fraud',

of two rows and two
e, true negative and
uracy.

```
sns.heatmap(cm_mlp, annot=True, cbar=None, cmap="Blues", fmt = 'g')
plt.title("Confusion Matrix MLP"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

Heat map is a data visualization graphical software where it uses color the way bar graph uses height and width. We can get idea from a heatmap that where we need most attention with dark color. From heat map we can easily visualize and make decision.

```
from sklearn.model_selection import KFold, cross_val_score

cvs1 = cross_val_score(MLPClassifier(), X_under, Y_under, scoring='accuracy',
cv=KFold(n_splits=10))
print("MLP Classifier - Accuracy : ",cvs1)
```

```
#MLP Classifier - Accuracy :  [0.96959459 0.98648649 0.99322034 0.99661017 0.99322034
0.98644068
#0.99661017 0.98983051 0.8779661  0.78983051]
```

If we want to check how good our model then there is a technique called cross validation which is responsible to evaluate machine learning model by several times. It is very useful in assessing the effectiveness of our model in overfitting case. Another use of cross validation is that, we can determine hyper parameters of our model which will result in lowest test error.
I used cross validation and gave the kfold 10 times.

Data Analysis performed by Malav Joshi:
Here , I will use logistic regression for credit dataset. The following steps I will follow.

1. Reading, understanding and visualising the data
2. Preparing the data for modelling
3. Building the model
4. Evaluate the model

Step -1 : Reading ,understanding and visualizing the data

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

- These are the basic library to load the dataset and to make graph. BY using this we can see that what distribution data follow or data is balance or not or is there any correlation between variables or not.

data = pd.read_csv("card.csv")

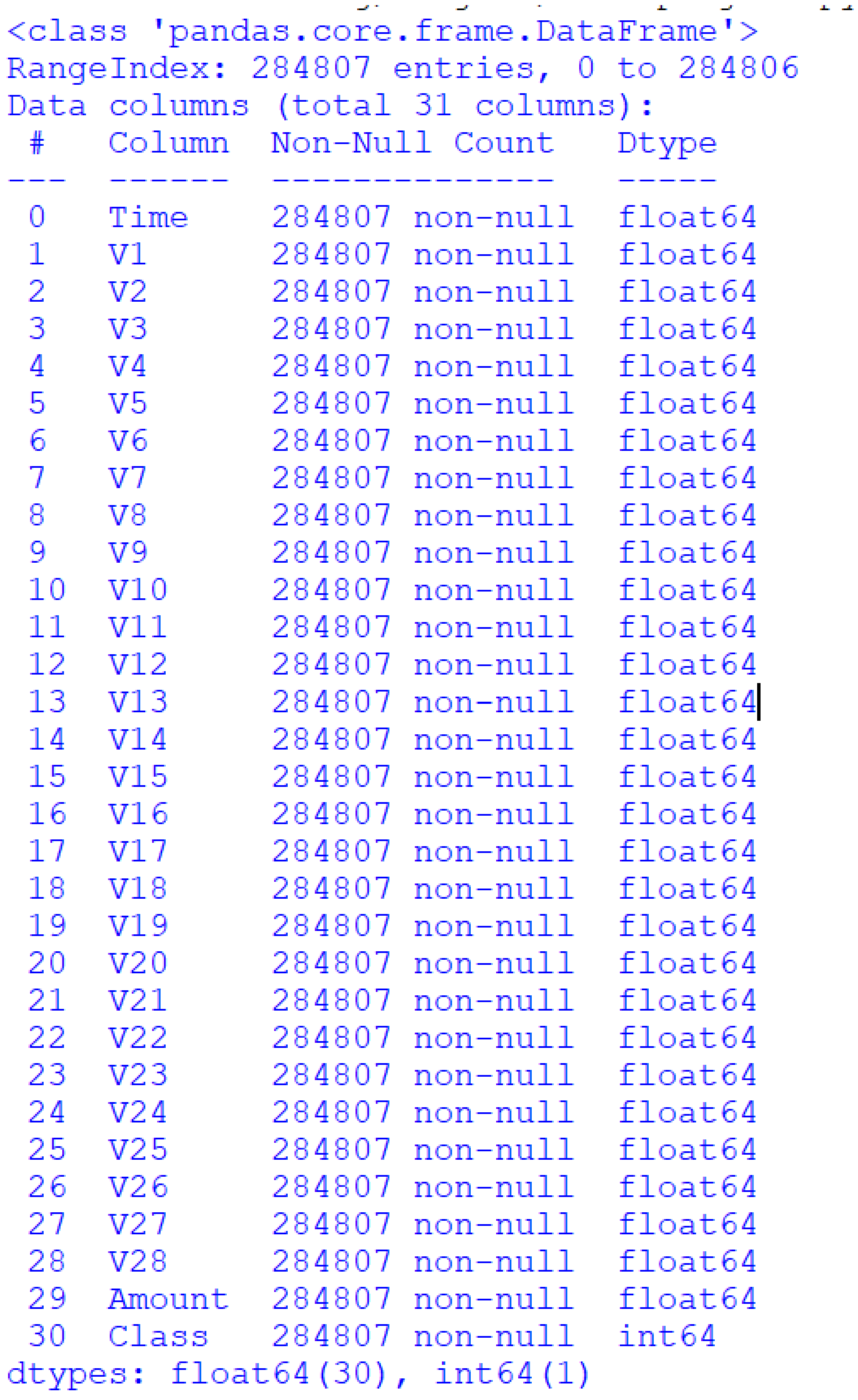
data.head()

data.info()

data.shape

(284807, 31)

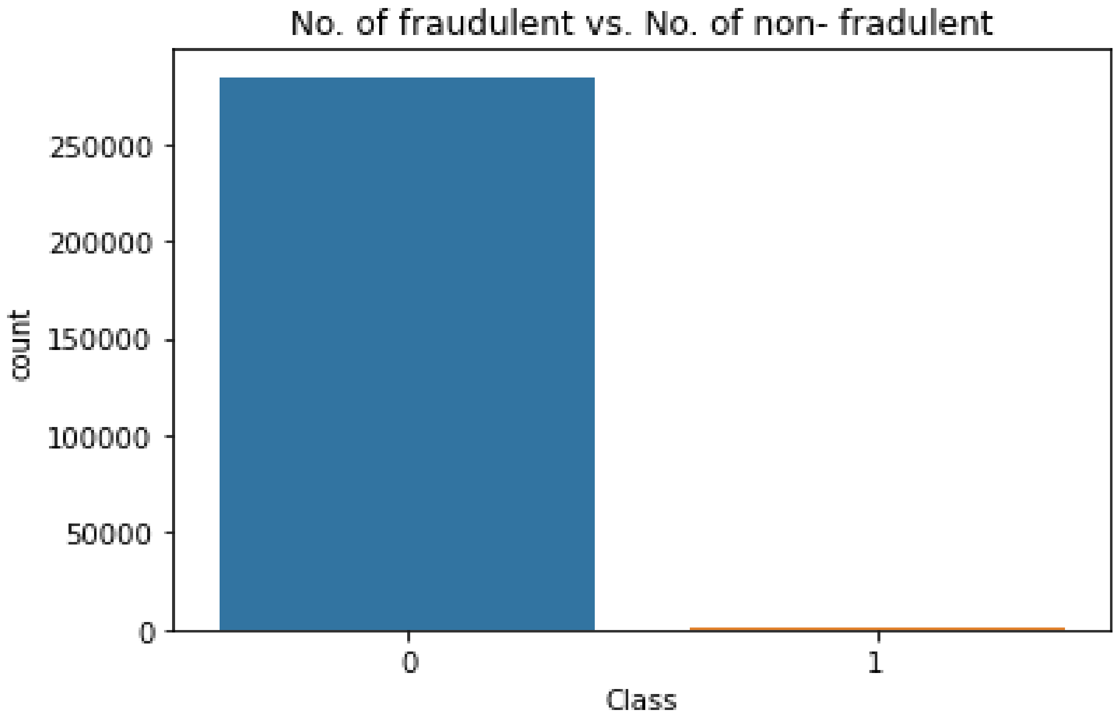
- Here data.head() will give us first 5 rows of the dataset , data.info() will give us the data type of each attributes and also whether is there any null values in the dataset or not. data.shape will give us number of record and number of columns in the dataset. We can see that there are 284807 records in total which is very big dataset. And also there are 31 columns namely time , v1, v2,…v28,amount and class.



```
<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)
```

Checking the distribution of the class :

```
classes = data['Class'].value_counts()
print(classes)
0    284315
1       492
Name: Class, dtype: int64
normal = round((classes[0]/data['Class'].count()*100),2)
print(normal)
99.83
fraud = round((classes[1]/data['Class'].count()*100),2)
print(fraud)
0.17

sns.countplot(x="Class",data=data)
plt.title("No. of fraudulent vs. No. of non- fradulent")
```

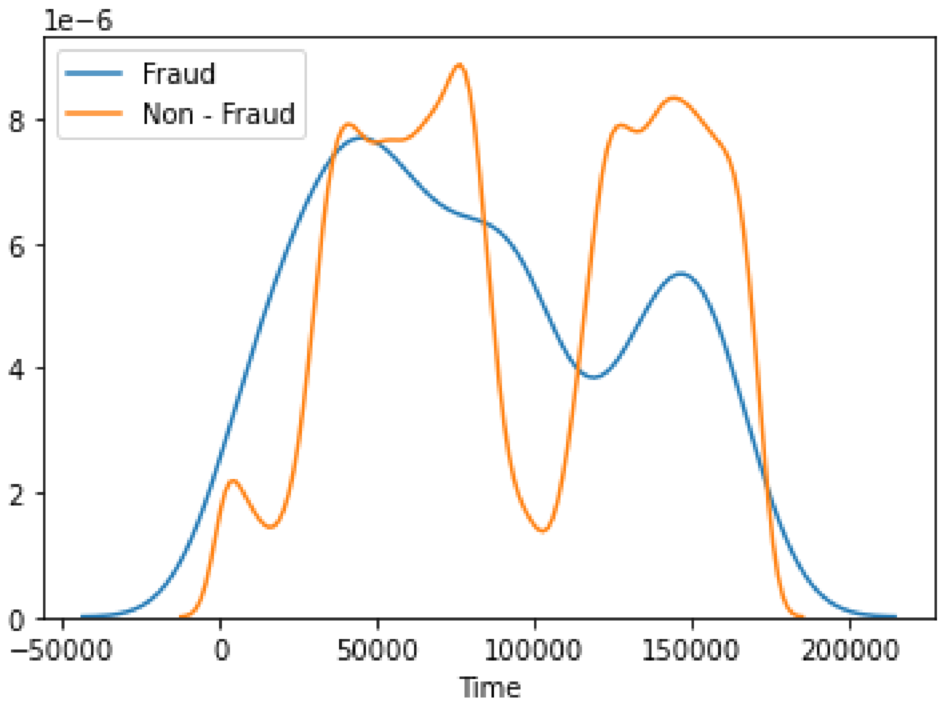


- Here from the code and also from the distribution plot we can see that data is very imbalanced. There are 284315 records belong to class 0 and 492 belong to class 1.In percentage almost 100% data belongs to class 0 and only 0.17% data belongs to 1. So we have to make it balance by using best method which I will be using in the next part.

- Here , we will check Time attribute to check whether it is affected our data or not . if not then we will drop it.

```
#let us create new variables by fraud and non- fraud transactions.

data_fraud = data[data['Class']==1]
data_non_fraud = data[data['Class']==0]

#Distribution Plot

ax = sns.distplot(data_fraud['Time'],label="Fraud",hist=False)
ax = sns.distplot(data_non_fraud['Time'],label="Non - Fraud",hist=False)


data.drop('Time',axis=1)
data.shape
```

- Here form the distribution plot we can not see any kind of pattern in the graph so there is not use meaning of keeping this variable in the dataset. So I have dropped it.



Step – 2 : Preparing the data for the model

- so first part is done and now we will split the data , scale the data and as I mentioned in the first part that we have to make data balance here in this step we will make it balance dataset.

#now let us split data

```
X = data.drop("Class",axis=1)
X.shape
X.info()

y = data['Class']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3,random_state=0)

from sklearn.preprocessing import StandardScaler
sc= StandardScaler()

X_train['Amount'] = sc.fit_transform(X_train[['Amount']])
X_test['Amount'] = sc.transform(X_test[['Amount']])
```

- Now here we will define our data into X and y . X contains all the variables but not include class variable and y contains only target variable which is class. After that we will split the data into train and test using train_test_split method. In that we will take 30% test size and 70% data we will use to train our logistic regression model.

- After that we will scale our data using standardscaler and as we can see that our data is almost normal but the attribute amount is not so we have normalize that attribute too.

- As we have seen that there are several methods to deal with imbalance dataset which are given below :

  1. **Under Sampling :** Under sampling aims to balance class distribution by randomly removing the majority class. This procedure continues until the class become balance.

  2. **Over Sampling :** Over sampling aims to increase the number of records of the minority class by randomly repeating them in order to represent minority class in the data.

  3. **SMOTE :** this is synthetic minority technique to avoid overfitting when we add exact records of minority class in the main dataset.

  4. **Adasyn :** This method is very similar to SMOTE but the advantage of this method is that it generated data and it does not copy our same minority class data. And also generate more to data for model so it becomes hard to learn. Which will give us better results.

- so, for our dataset we will use Adasyn to make balance dataset .

```
from imblearn.over_sampling import ADASYN

ada = ADASYN(random_state=0)
X_train_adasyn, y_train_adasyn = ada.fit_resample(X_train, y_train)

from collections import Counter
# Befor sampling class distribution
print('Before sampling class distribution:-',Counter(y_train))
Before sampling class distribution:- Counter({0: 199019, 1: 345})

# new class distribution
print('New class distribution:-',Counter(y_train_adasyn))
New class distribution:- Counter({0: 199019, 1: 198957})
```

- So , here we have used Adasyn method which we import from over_sampling . we have created adasyn model and fit it using X_train and y_train. After fitting we have checked whether it worked or not. As we can see that we have use counter method to check the number of classes are equal or not.

- Before applying this sampling method there were 199019 records of class 0 and only 345 records of class 1. After applying we could see that there are 199019 record of class 0 and 198957 records of class 1.

Step – 3 : Building the model

- Now , we are done with all the pre processing steps like checking the null values , dropping Time attribute , divide the data into train and test , scale the data and last we made dataset balance using Adasyn. Its time to build a model using LogisticRegression. Here , first we will import some required packages .

```
from sklearn.linear_model import LogisticRegression
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

- First we have import our main model from linear_model after that we have imported all the packages which are useful to evaluate our model and they are confusion matrix f1_score and classification report.

- Here I am going to use parameter tuning along with KFold to build more accurate model which can help to classify data correctly.

```
# Creating KFold object with 3 splits
folds = KFold(n_splits=10, shuffle=True, random_state=4)


# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_adasyn, y_train_adasyn)
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
The highest test roc_auc is 0.9953533903855286 at C = 10
```

- Firstly, I did KFold method with n_Splits=10 and I define random state to get the same output every time. After that , I define params and I randomly stored values of C in it. After that using roc _auc score while using gridsearchcv method. After that I have fitted model using X_train_adasyn and y_train_adasyn which are balanced data. Then I found best score for each combination and and also the best value of parameter C . so , the highest score of roc_auc is 0.99 at C = 10

```
logistic_bal_adasyn = LogisticRegression(C=10)
```

# Fit the model on the train set
logistic_bal_adasyn_model = logistic_bal_adasyn.fit(X_train_adasyn, y_train_adasyn)

- Here I have fitted model using C=10 and used X_train_adasyn, y_train_adasyn .

Step – 4 : Evaluating the model
y_test_pred = logistic_bal_adasyn_model.predict(X_test)

# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)

```
83602  1694
19          128
```

TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

Sensitivity:- 0.8707482993197279
Specificity:- 0.9801397486400301
Accuracy:- 0.9799515466451318

- So from the confusion matrix we can see that there are 83730 records are classified correctly by our model. And 1694 are false positive which means model incorrectly predicted positive class and 128 are false negatives which means they are not correctly predicted as negative class.

Now , I will use Logistic Regression model again but without using parameter tuning and KFold method to check whether is there any difference can be observed or not .

- So , all the parts are the same but I have to change the model part and the evaluating part.

logistic_bal_adasyn = LogisticRegression()

# Fit the model on the train set

```
logistic_bal_adasyn_model = logistic_bal_adasyn.fit(X_train_adasyn, y_train_adasyn)

# Predictions on the train set
#y_train_pred = logistic_bal_adasyn_model.predict(X_test_adasyn)
y_test_pred = logistic_bal_adasyn_model.predict(X_test)

# Confusion matrix
#confusion = metrics.confusion_matrix(y_train_adasyn, y_train_pred)
#print(confusion)
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)



TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))
```

```
[[83790  1506]
 [   16   131]]
Sensitivity:- 0.891156462585034
Specificity:- 0.982343837929094
Accuracy:- 0.9821869550460541
```

- So from the confusion matrix we can see that there are 83921 records are classified correctly by our model. And 150 are false positive which means model incorrectly predicted positive class and 116 are false negatives which means they are not correctly predicted as negative class.

- Here the accuracy is almost same if we go use KFold and parameter tuning or not and it is 98% . Specificity is also the same and which is 98%. But minor change observed in sensitivity with KFold and parameter tuning it is 87% and without it is 89%. So there not much changes I could observe in both the cases.
-

So , now we have reached to the conclusion part. Here in this we will compare the accuracy , sensitivity and specificity of every model to see which one is better.

Models :	**Decision Tree Classifier**	**MLP Classifier**	**Logistic Regression(KFOld and parameter tuning)**	**Logistic Regression**
Accuracy	99%	97.8%	97.99~98%	98.21 ~98%

- From the all the accuracy it can be seen that almost all the models given same accuracy or minor change in accuracy we could observe. So from the above models any model can be used to find the fraud or non – fraudulent data.

CONCLUSION : So in this project we have prepared different machine learning algorithms and have used them on the credit card fraud detection dataset. We have used the algorithms like Decision Tree , Neural Network and Logistic Regression . We have also made use of a confusion matrix in every algorithm we used , to find out the accuracy and error rate of each machine learning model. So after implementing the confusion matrix on each algorithm the accuracy we got goes like this. Decision Tree scored an accuracy of 99.82 , then Neural Network with an accuracy of 97.8 , and lastly Logistic Regression model with an accuracy of 98.21. All the models scored pretty well on the basis of the accuracy , but Decision Tree proved to be the best.