SHUBHASREE SARKAR

PROJECT NAME- CREDIT CARD FRAUD DETECTION

## ***OBJECTIVE***

As mentioned before, the datasets contain 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.
The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
It is one of the unique datasets containing an unbalanced data of fraudulent and legitimate transactions.

1. Why this problem needs to be solved -

This problem is used to model past credit card transactions which are known to be fraud. It is then used to identify whether a new transaction is fraudulent or not. We are here to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

2. The impact of a great solution to the Industry-

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account.
Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behavior, which consist of fraud, intrusion, and defaulting.
This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated.

**Main challenges involved in credit card fraud detection are:**

- Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
- Imbalanced Data i.e most of the transactions *(99.8%)* are not fraudulent which makes it really hard for detecting the fraudulent ones
- Data availability as the data is mostly private.
- Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
- Adaptive techniques used against the model by the scammers.

Understading the Data , Exploring it's length , breadth and height ( How BIG , How DIVERSE , How GRANULAR ) Called : **EDA**

In order to understand and explore the data for its classification, we need to follow the respective steps as required.

## DATA PROCESSING AND MACHINE LEARNING STEPS

### Step 1a- IMPORT ALL THE BASIC LIBRARIES

#first import all the necessary libraries as required on the initial part

**import numpy as np** #provides high-performance multidimensional array with manipulation tool

**import pandas as pd** #Python-based data analysis toolkit

**import matplotlib.pyplot as plt** #plotting library for the Python and its numerical mathematics extension NumPy.

**import seaborn as sns** #Python data visualization library based on matplotlib

**import math** #To use mathematical functions

**import matplotlib**

**import datetime** #supplies classes for manipulating dates and times.

**import sklearn** #most useful library for machine learning and statistical modelling

**import scipy** #uses NumPy for more mathematical functions

**import warnings** #provided to warn the developer of situations that aren't necessarily exceptions

**warnings.filterwarnings ('ignore')**

### Step 1b- GET THE VERSIONS OF THE MODULES AND LIBRARIES

```
! python --version
print(f"Pandas.version : Pandas {pd.__version__}")
print(f"Matplotlib.version : Matplotlib {matplotlib.__version__}")
print(f"Numpy.version : Numpy {np.__version__}")
print(f"Seaborn.version : Seaborn {sns.__version__}")
print(f"Scipy.version : Scipy {scipy.__version__}")
print(f"Skklearn.version : Sklearn {sklearn.__version__}")
```

**RESULTS**

| | |
|---|---|
| **Python 3.8.3** | **Pandas.version : Pandas 1.0.5** |
| **Matplotlib.version : Matplotlib 3.2.2** | **Numpy.version : Numpy 1.18.5** |
| **Seaborn.version : Seaborn 0.10.1** | **Scipy.version : Scipy 1.5.0** |
| **Skklearn.version : Sklearn 0.23.1** | |

### Step 2 – DATA COLLECTION

**#import the dataset**

```
cc = pd.read_csv("creditcard.csv")
```

**#top 5 dataset**
#used to make all the columns visible, use max_rows for row visibility
pd.set_option('display.max_columns',100)
cc.head()

**#bottom 5 dataset**
cc.tail()

**#length of the dataset**
cc.shape

**RESULTS**

**Exploring its length**

Rows (284807)
Columns (31)

**Step 3 – DATA PREPARATION**

It is comprised of the following parts-

**PART 1- DATA DESCRIPTION**

#sample dataset
cc.sample(10)

#check for the data types
cc.info ()

**RESULTS**

```
<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)
```

**#Filtering the data into fraud and legit transactions**

```
fraud = cc.loc[cc['Class'] == 1] #filter the data on the basis of Class as Fraud
legit = cc.loc[cc['Class'] == 0] #filter the data on the basis of Class as Fraud

print (f"total fraud : {len(fraud)}") #count of fraud data
print (f"total legit : {len(legit)}") #count of legit data
```

#check for outlier fraction

```
print (f"outlier_fraction : {len(fraud)/float(len(legit))}")
```

**RESULTS**

**total fraud: 492**
**total legit: 284315**
**outlier_fraction: 0.0017304750013189597**

**Finding the Complexities and Roadblocks for Modelling this particular Use Case**

The dataset has been imbalanced, which can be witnessed from the variations in the fraud and legit transaction data. So, it will be required to balance it using the different sampling techniques.
But before that it is important to understand the presence of Null values, duplicates, outliers, etc

## PART 2- DATA CLEANING

#different types of data cleaning processes
check for missing data
check for duplicate values
check for outliers

### Part 2.1 of Data Cleaning - Deal with missing values

**#check for missing values**

```
data = cc[cc.columns[cc.isnull().any()]].isnull().sum()
for y in data:
   print(y)
cc.isnull().sum()
```

**RESULTS**

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

**# to return all the null containing columns and the total number**

```
cc[cc.columns[cc.isnull().any()]].isnull().sum()
```

**RESULTS**

Series([], dtype: float64)

**Part 2.2 of Data Cleaning - Deal with outliers**

**#check for outliers - part 1**

```
outliers = []
def detect_outliers(data):
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)
    for y in data:
        z_score = (y-mean)/std
        if np.abs(z_score)>threshold:
            outliers.append(y)
    return outliers
```

**#detection of outliers in the amount column**

```
outliers_datapoints = detect_outliers(cc['Amount'])
print(outliers_datapoints)
```

**#detection of outliers in the class column**

```
outliers_datapoints = detect_outliers(cc['Class'])
print(outliers_datapoints)
```

**OUTCOME**

**Outliers are present in both the columns**

**#Outlier removal part 2**

```
def indices_of_outliers(x):
 q1, q3 = np.percentile(x, [25,75])
 iqr = q3-q1
lower_bound = q1 -(iqr * 1.5)
upper_bound = q3 + (iqr * 1.5)
return np.where((x > upper_bound) | (x < lower_bound)) #limit has been set
```

**#Outlier removal part 3 #on the basis of a set max value as limit**

```
def outlier_removal(max_val):
 print("Values lost on thr basis of max based removal :{}".format(len(cc[(cc['Class']==0) &
(cc['Amount']>max_val)])))
print("proportion of data lost :{}".format(len(cc[(cc['Class']==0) & (cc['Amount']>max_val)])/len(cc)))
 temp_cc = cc[cc['Amount']< max_val] #outlier removed
 print(temp_cc['Class'].value_counts(normalize = True))
```

**#check max for fraud transactions**

```
cc[cc['Class']==1]['Amount'].max()
```

**RESULTS**

**2125.87**

**#considering max value as 3000 set as limit of amount, count of how many data_rows are to be dropped**

```
outlier_removal(3000)
```

**RESULTS**

| |
|---|
| Values lost on thr basis of max based removal :284 |
| proportion of data lost :0.0009971665022278245 |
| 0    0.998271 |
| 1    0.001729 |
| Name: Class, dtype: float64 |

**# Part_2.3 of Data Cleaning - Deal with the duplicate values**

#check for duplicates

```
len(cc[cc.duplicated()])
print("total number of duplicates =",len(cc[cc.duplicated()]))
```

**RESULTS**

| |
|---|
| total number of duplicates = 1081 |

#after removing duplicates

```
cc = cc.drop_duplicates()
```

#check the total nos of rows and columns presently

```
cc.shape
```

**RESULTS**

| |
|---|
| (283726, 31) |

**OUTCOME**

With respect to the data cleaning process only the duplicates(1081) are being removed. As the data can contain high values in terms of the Amount of the transactions, hence the presence of the outliers is there as well. But they are not being removed as they will be required to understand the data better.

## PART 3- EXPLORATORY DATA ANALYSIS

### Part 3.1.- Numerical Analysis

**#rows and columns of respective type of transactions**

```
print("fraud(rows,columns)= ", fraud.shape,
      "legit(rows,columns)= ",legit.shape)
```

**RESULTS**

| |
|---|
| **fraud(rows,columns)= (492, 31)** |
| **legit(rows,columns)= (284315, 31)** |

**# check for different statistics and metrics**

```
cc.describe()
```

**#Time , Amount and Class wise statistics**

```
cc[['Time', 'Amount', 'Class']].describe().T
```

**RESULTS**

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Time | 284807.0 | 94813.859575 | 47488.145955 | 0.0 | 54201.5 | 84692.0 | 139320.500 | 172792.00 |
| Amount | 284807.0 | 88.349619 | 250.120109 | 0.0 | 5.6 | 22.0 | 77.165 | 25691.16 |
| Class | 284807.0 | 0.001727 | 0.041527 | 0.0 | 0.0 | 0.0 | 0.000 | 1.00 |

**Points noted**

Time, Amount are the only measurable attributes, While Class is Categorical. In the Amount colmn, Variation of data highly signifies, presence of outliers in them.

**#stats related to the legit type transactions- Amount**

```
print(legit.Amount.describe())
```

**RESULTS**

```
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
```

**#stats related to the fraud type transactions- Amount**

```
print(fraud.Amount.describe())
```

**RESULTS**

```
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
```

**Points noted-** The variation of results that can be clearly be seen from here indicates that the Average Amount transaction for the fraudulent ones is more as compared to the Legit ones. Hence, it makes it complicated to deal with

---

**Part 3.B.- Graphical Analysis**

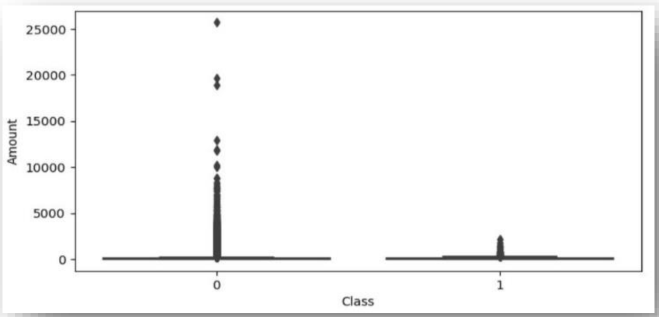
**Plot 1- Boxplot for the type of Transactions**

```
plt.figure(figsize= (8,4), dpi = 100) #helps to create a figure object

#the figure is chosen to be boxplot for the two Class variation ie, fraud and legit

sns.boxplot(x = 'Class',
            y = 'Amount',
            data = cc)
plt.show()
```

**RESULTS**



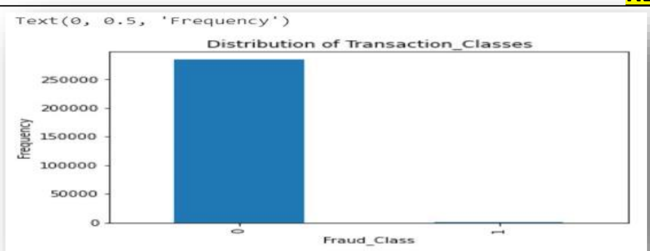

**Points noted**

This box plot shows presence of outliers among the legit transactions, whereas absence of outliers among the fraudulent transactions

**Plot 2- Barplot for the type of Transaction**

```
#distribution of classes
value =pd.value_counts(cc['Class'], sort =True)
value.plot(kind = 'bar')
plt.title("Distribution of Transaction_Classes")
plt.xlabel("Fraud_Class")
plt.ylabel("Frequency")
```

**RESULTS**



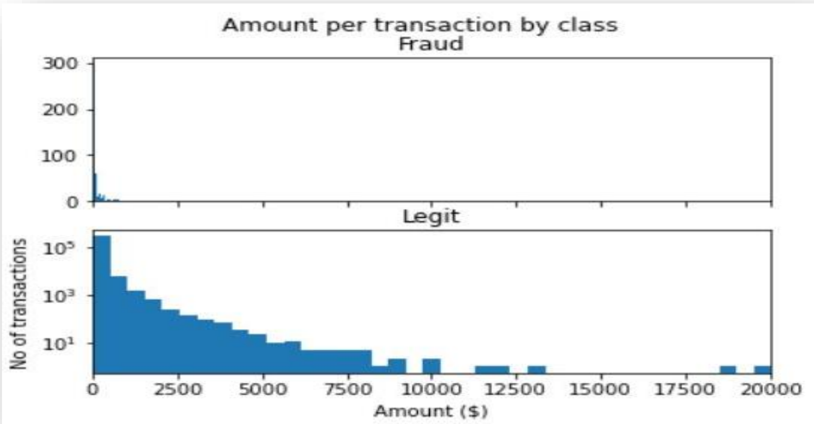

**Points noted**

We can witness from here, that the most transactions recorded had been legit (0), while few of them are fraud(1)

**Plot 3 - Histogram plots of Amounts by Class of Transactions**

```
#transactions with respect to the amount in terms of histogram plots
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(legit.Amount, bins = bins)
ax2.set_title('Legit')
plt.xlabel('Amount ($)')
plt.ylabel('No of transactions')
plt.xlim((0,20000))
plt.yscale('log')
plt.show()
```

**RESULTS**




**Points noted**

The histogram showing the frequency wise distribution of legit and fraud datasets and also if presence of outliers among the different amounts.

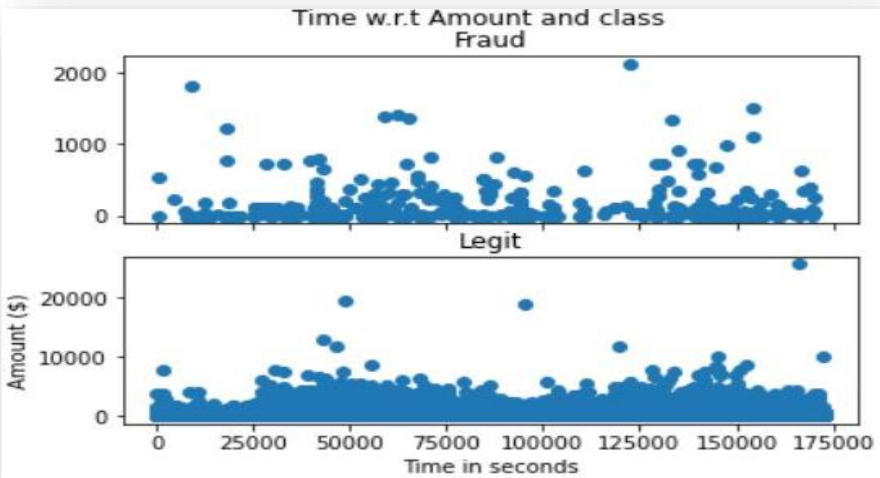
**Most no. of transaction**- Legit

**Higher amount involved**- Legit transaction

**Plot 4 - Scatter plots of Time in secs w.r.t.Amounts by Class of Transactions**

```
#transactions with respect to the amount in terms of scatter plots
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time w.r.t Amount and class')
bins = 50
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(legit.Time, legit.Amount)
ax2.set_title('Legit')
plt.xlabel('Time in seconds')
plt.ylabel('Amount ($)')
plt.show()
```

**RESULTS**



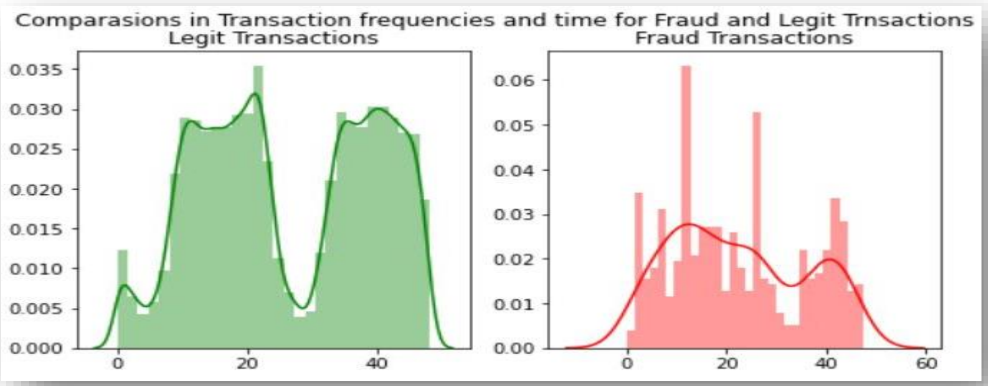

**Points noted**

**Legit Transactions** – Higher Amount have taken more time to process, in few of the cases.But in general we **can't draw a direct or indirect relation** between, the **Amount** and **Time** taken.

**Fraud Transaction**- Time taken for process, has been **more or less uniform**, except for a few large no. involved.

```
#convert time into hour
cc['hour']= cc['Time']/(60*60)
# time plots
#created a subplot with rows = 1 n columns = 2
fig,axs = plt.subplots(1,2, figsize = (8,4))
#using distribution plot ,where we have filtered using class - 0 for legit transactions, considered the hours' column with the values of the same
sns.distplot(cc[cc['Class']==0]['hour'].values, color ='green', bins = 30, ax = axs[0])
#here ax is used from matplotlib
axs[0].set_title("Legit Transactions")
# setting axs[0] for the legit transaction
sns.distplot(cc[cc['Class']==1]['hour'].values, color ='red', bins = 30, ax = axs[1])
#here ax is used from matplotlib
axs[1].set_title("Fraud Transactions")   # setting axs[0] for the fraud transaction
fig.suptitle('Comparisons in Transaction frequencies and time for Fraud and Legit Transactions')
plt.show()
```

**RESULTS**




**Points noted**

The time and frequencies of fraud and legit transactions are being shown here.

Their distribution shows that fraud transactions are not done too often, they had been abrupt.

**Plot 6- Visualising all the data in histogram plots to find all the anomalies**

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

**RESULTS**


**Points noted**

This histogram plot of all the attributes together show the presence of any anomaly.

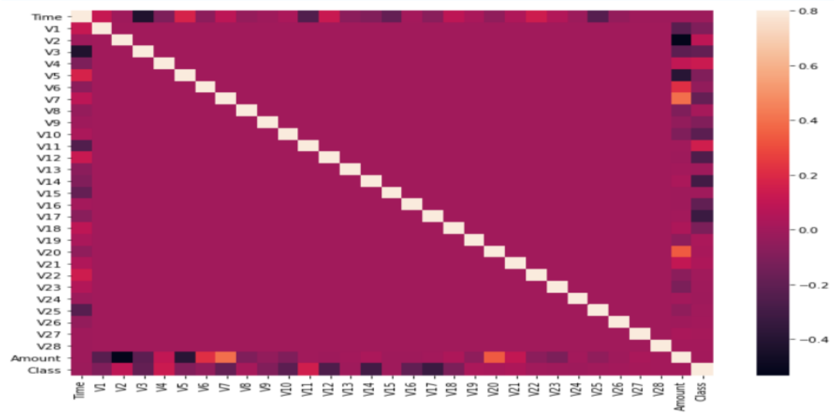
**#Check for Heatmap using correlation**

**#Plotting the Correlation Matrix**

The correlation matrix provides us with graphically visualization about how features correlate with each other which in turn also help us predict what are the features that are most relevant for the prediction.

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

**RESULTS**



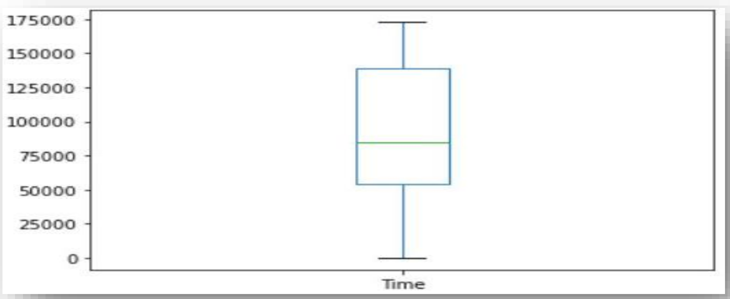

**Points noted**

This Heatmap indicates that most of the features do not correlate with each other but there are some features that either has a positive or a negative correlation. For example, **V2** and **V5** are highly negatively correlated with the feature called **Amount**. We also see some correlation with **V20** and **Amount**. This gives us a deeper understanding of the Data available to us.

**Plot 8- Boxplot of all the variables**

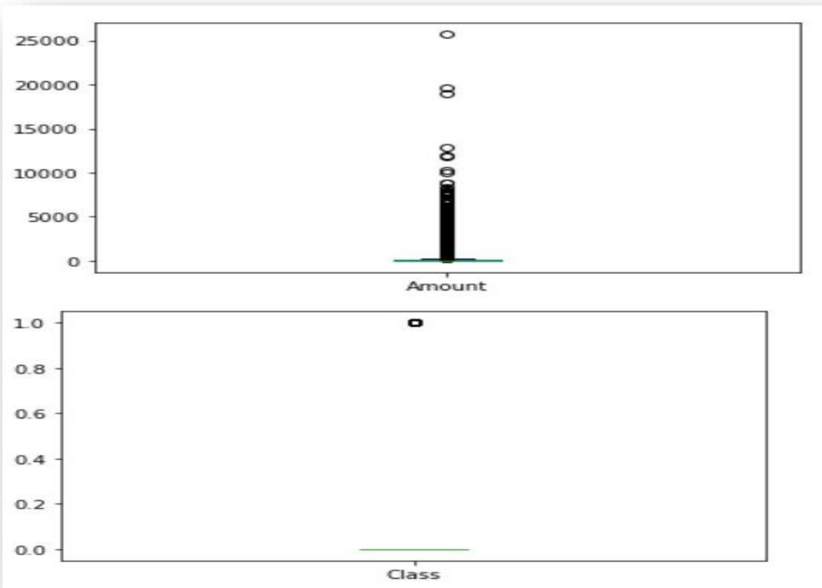
```
for col in cc.columns:
   plt.figure()
   cc[col].plot(kind = 'box')
```

**RESULTS**




**Points noted**

The Box plot of time shows no presence of outliers as only 2 day data is being plotted




**Points noted**

Amount related to transactions as can be witnessed from here contains outliers because of **HIGH values**.

Class being categorical represents only two types.

**# check for per hour transactions on the basis of amount**

```
# check for per hour transactions on the basis of amount and class
cc['hour'] = cc['Time'].apply(lambda x: np.ceil(float(x)/3600) % 24)
cc.pivot_table(values= 'Amount',index='hour',columns='Class',aggfunc='count')
```

**RESULTS**

| Class | 0 | 1 |
|---|---|---|
| hour | | |
| 0.0 | 10868 | 17 |
| 1.0 | 7639 | 6 |
| 2.0 | 4200 | 10 |
| 3.0 | 3258 | 48 |
| 4.0 | 3471 | 17 |
| 5.0 | 2180 | 23 |
| 6.0 | 2977 | 11 |
| 7.0 | 4074 | 9 |
| 8.0 | 7209 | 23 |
| 9.0 | 10222 | 9 |
| 10.0 | 15753 | 16 |
| 11.0 | 16543 | 8 |
| 12.0 | 16729 | 53 |

| Class | 0 | 1 |
|---|---|---|
| hour | | |
| 13.0 | 15358 | 17 |
| 14.0 | 15308 | 17 |
| 15.0 | 16495 | 23 |
| 16.0 | 16347 | 26 |
| 17.0 | 16378 | 22 |
| 18.0 | 16100 | 28 |
| 19.0 | 16928 | 28 |
| 20.0 | 15549 | 19 |
| 21.0 | 16688 | 18 |
| 22.0 | 17618 | 16 |
| 23.0 | 15361 | 9 |

*******************************************************************************

**STEP- 4-Choosing a Model**

Among all the Machine Learning models, we will be considering only the following to understand the one which best suits this case and provides maximum accuracy (Because the data is **imbalanced**)

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- K-Neighbours Classifier
- XGBOOST Classifier
- Naïve Bayes Classifier

*******************************************************************************

**STEP- 5-Preparing the data through filtering process for the Model**

**Part 5.1.Splitting the data as per the features and target variables**

```
#get all the columns in terms of list
columns = cc.columns.tolist()

#filter the columns we do not want
columns = [c for c in columns if c not in ["Class"]]

#filtered out all the columns using a for loop to consider all except the 'Class' column
#created a target variable where we have placed all the data related to 'Class' column
target = "Class"
#Splitting data into feature and target variables
state = np.random.RandomState(42)
```

```
y = cc[target]
x = cc[columns]
x_outliers = state.uniform(low=0, high=1, size=(x.shape[0], x.shape[1]))
#check for the shape of the individual categories
print(x.shape)
print(y.shape)
```

**RESULT**

| |
|---|
| **X- (283726, 30)** |
| **Y- (283726,)** |

---

## Part 5.2. Importing packages as required step by step

```
#import the classes associated with the modules required for training the dataset
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, auc, roc_curve
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

---

## Part 5.3. Splitting the data into training and test dataset

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size = 0.30, random_state = 0)
```

#### Check for Class imbalance

As we can witness, that there has been quite an imbalance in the transaction data based on Class ie,
**total fraud: 492**
**total legit: 284315**

Hence, in order to reduce such class imbalance, we must opt for two processes

1. **Random Under Sampling**
2. **Random Over Sampling**

---

## Part 5.4. Importing packages as required step by step for sampling process

```
#to deal with sampling techniques, we need to install the following package
!pip install imblearn

#Install the modules associated with this package
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler,SMOTE, ADASYN

# counter takes values returns value_counts dictionary
from collections import Counter

# to generate a random n class classification problem
from sklearn.datasets import make_classification
#x, y = make_classification(n_classes=2) #, class_sep=2,weights=[0.1, 0.9], n_informative=3,
n_redundant=1, flip_y=0,n_features=20, n_clusters_per_class=1, n_samples=1000, random_state=10)
print('Original dataset shape %s' % Counter(y))
```

**RESULT**

Original dataset shape Counter ({0: 283253, 1: 473})

- **RANDOM UNDER SAMPLING**

```
rus = RandomUnderSampler(random_state=42)
x_rus, y_rus = rus.fit_resample(x, y)
print('Resampled dataset shape %s' % Counter(y_rus))
```

**RESULT**

| |
|---|
| Resampled dataset shape Counter ({0: 473, 1: 473}) |

- **RANDOM OVER SAMPLING**

```
ros = RandomOverSampler (random_state= 42)
x_ros, y_ros = ros.fit_resample (x, y)
print ('Resampled dataset shape %s' % Counter(y_ros))
```

**RESULT**

| |
|---|
| Resampled dataset shape Counter ({0: 283253, 1: 283253}) |

- **SMOTE**

```
ros = SMOTE (random_state= 42)
x_ros1, y_ros1 = ros.fit_resample(x, y)
print ('Resampled dataset shape %s' % Counter(y_ros))
```

**RESULT**

| |
|---|
| Resampled dataset shape Counter ({0: 283253, 1: 283253}) |

---

**Part 5.5. Types of Metrics to be used to check for accuracy and fitness**

- **Confusion Matrix**

  One of the key concepts in classification performance is **confusion matrix** (or error matrix), which is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

  There are 4 important terms:

I. **True Positives**: The cases in which we predicted YES and the actual output was also YES.
II. **True Negatives**: The cases in which we predicted NO and the actual output was NO.
III. **False Positives**: The cases in which we predicted YES and the actual output was NO.
IV. **False Negatives**: The cases in which we predicted NO and the actual output was YES.

Accuracy for the matrix can be calculated by taking average of the values lying across the **"main diagonal"** i.e

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalSample}$$

Confusion Matrix forms the basis for the other types of metrics.

---

- **Classification Accuracy**

  Classification accuracy is perhaps the simplest metrics one can imagine, and is defined as the **number of correct predictions divided by the total number of predictions,** multiplied by 100**.** Or in other words, it is also **the ratio of number of correct predictions to the total number of input samples.**

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

It works well only if there are equal number of samples belonging to each class.

- **Precision**

  There are many cases in which classification accuracy is not a good indicator of the model performance. In those scenarios, where class distribution is imbalanced (one class is more frequent than others), even after predicting all samples as the most frequent class one would get a high accuracy rate, which does not make sense at all (because the model is not learning anything, and is just predicting everything as the top class).

  Therefore, we need to look at class specific performance metrics too. Precision is one of such metrics, which is defined as:

  It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

---

- **Recall**

  Recall is another important metric, which is defined as the fraction of samples from a class which are correctly predicted by the model. More formally:

  It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive).

$$recall = \frac{true\ positives}{true\ positives\ +false\ negatives}$$

---

- **F1 Score**

  Depending on application, we may want to give higher priority to recall or precision. But there are many applications in which both are important. Therefore, a combination of both as a single metric is required, which is exactly being defined by F1 Score.

  I. **F1-score** is the harmonic mean of precision and recall defined as:
  II. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).
  III. F1 Score tries to find the balance between precision and recall.
  IV. F1 Score is used to measure a test's accuracy

$$\textbf{F1-score} = \frac{\textbf{2*Precision*Recall}}{\textbf{(Precision+Recall)}}$$

---

- **Sensitivity & Specificity**

  **I.** **Sensitivity** measures the proportion of positives that are correctly identified (i.e. the proportion of those who have some condition (affected) who are correctly identified as having the condition).

  II. **Specificity** measures the proportion of negatives that are correctly identified (i.e. the proportion of those who do not have the condition (unaffected) who are correctly identified as not having the condition).

These are two other popular metrics mostly used in medical and biology related fields, and are also defined as:

Sensitivity = Recall= $\frac{TP}{(TP+FN)}$

Specificity = True Negative Rate = $\frac{TN}{(TN+FP)}$

---

- **ROC Curve**

  The **receiver operating characteristic curve** is plot which shows the performance of a binary classifier as function of its cut-off threshold.

  I. **It essentially shows the true positive rate (TPR) against the false positive rate (FPR) for various threshold values.**

  II. ROC curve is a popular curve to look at overall model performance and pick a good cut-off threshold for the model.

---

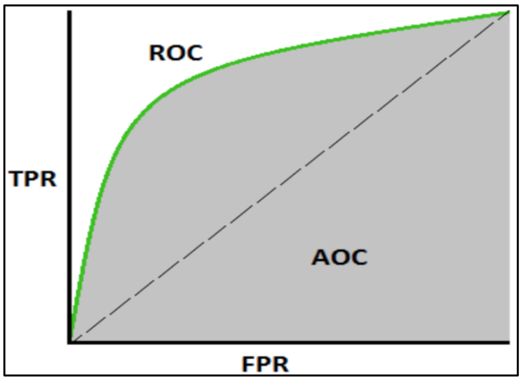
- **AUC**

  The **area under the curve** (AUC), is an aggregated measure of performance of a binary classifier on all possible threshold values (and therefore it is threshold invariant**)**.

  I. AUC calculates the area under the ROC curve, and therefore it is between 0 and 1.

  II. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

  III. On high-level, the higher the AUC of a model the better it is. But sometimes threshold independent measure is not what we want, e.g. the model recall can be expected to be higher than 99% (while it has a reasonable precision or FPR). In that case, it requires to tune the model threshold such that it meets the minimum requirement on those metrics




---

**Step- 6 -Model Selection And Training The Model**

**Classifier 1 - Logistics Regression**

#import the regression module

from sklearn.linear_model import LogisticRegression

**#Split the dataset into test train**

x_train,x_test,y_train,y_test = train_test_split(x,y, test_size = 0.3, random_state = 0)

**#train the model**

log_reg = LogisticRegression()

log_reg.fit(x_train, y_train)

```
#predict the model
y_pred = log_reg.predict(x_test)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score for Logistics Regression: {accuracy_score(y_pred, y_test)}")
```

**RESULT**

**Accuracy_Score for Logistics Regression : 0.9990483798961441**

**STEPS FOR CREATING ROC CURVE**

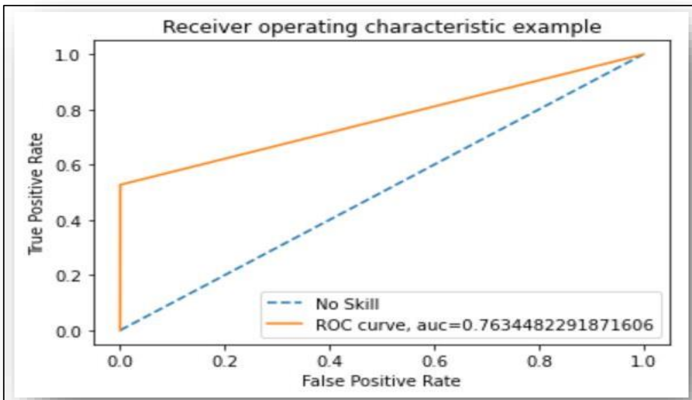
**# predict probabilities**

```
yhat = log_reg.predict_proba(x_test)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs = yhat[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test,  pos_probs) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc = roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **76.34%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.90%**

---

**FACTS**

**We can observe here that the model Accuracy score and roc_auc_score are no way similar in terms of values.**

**we can expect that the high accuracy score is because of high imbalance dataset**

**#check for the confusion matrix**

```
Confusion_Matrix = confusion_matrix(y_test,y_pred)
print (Confusion_Matrix)
```

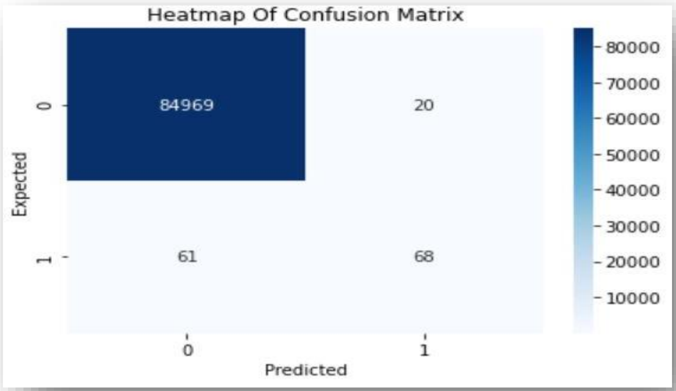
**RESULT**

```
[[84969    20]
 [   61    68]]
```

**#create the heatmap of the confusion matrix**

```
sns.heatmap(pd.DataFrame(Confusion_Matrix), annot = True, cmap = 'Blues', fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.title('Heatmap Of Confusion Matrix')
plt.show()
```

**RESULT**




**Points noted**

The heatmap of confusion shows that most of the transactions are being accurately identified, except for a few.

**#check for the classification report**

```
print(classification_report(y_pred,y_test))
```

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84989 |
| 1 | 0.68 | 0.59 | 0.63 | 129 |
| accuracy | | | 1.00 | 85118 |
| macro avg | 0.84 | 0.79 | 0.82 | 85118 |
| weighted avg | 1.00 | 1.00 | 1.00 | 85118 |

---

**1.1 Machine Learning Model - Logistic Regression - using Random under sampler**

**#Split the dataset into test train**

```
x_train_ru, x_test_ru, y_train_ru, y_test_ru = train_test_split(x_rus,y_rus, test_size = 0.3, random_state =
0)
```

**#train the model**

```
log_reg = LogisticRegression()
log_reg.fit(x_train_ru, y_train_ru)
#predict the model
y_pred_ru= log_reg.predict(x_test_ru)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of RUS : {accuracy_score(y_pred_ru, y_test_ru)}")
```

**RESULT**

**Accuracy_Score for Logistics Regression (RUS) : 0.9401408450704225**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**

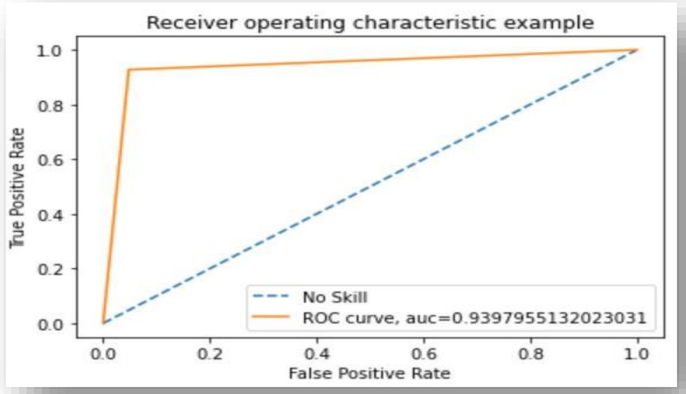
```
yhat_ru = log_reg.predict_proba(x_tes_ru)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_ru = yhat_ru[:, 1]
# plot no skill roc curve
```

```
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test_ru,  y_pred_ru) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_ru = roc_auc_score(y_test_ru, y_pred_ru)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_ru))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **93.97%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **94.01%**

------------------------------------------------------------------------------------------------------------------------------------

**1.2 Machine Learning Model - Logistic Regression - using Random Over sampler**

**#Split the dataset into test train**

```
x_train_ro, x_test_ro, y_train_ro, y_test_ro = train_test_split(x_ros,y_ros, test_size = 0.3, random_state =0)
```

**#train the model**

```
log_reg = LogisticRegression()
log_reg.fit(x_train_ro, y_train_ro)
#predict the model
y_pred_ro = log_reg.predict(x_test_ro)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of ROS : {accuracy_score(y_pred_ro, y_test_ro)}")
```

**RESULT**

**Accuracy_Score for Logistics Regression (ROS): 0.9377942007154961**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**

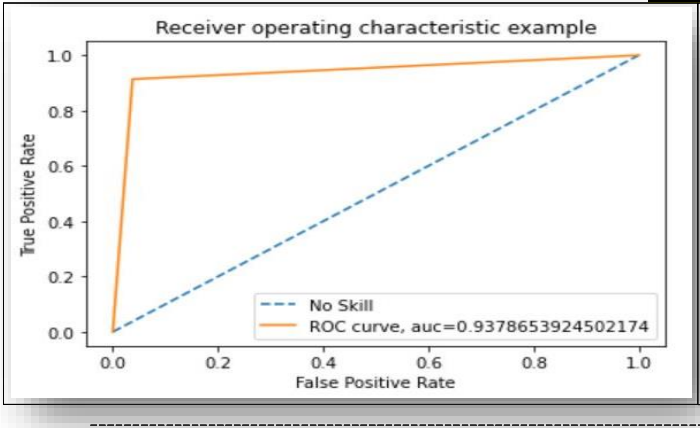
```
yhat_ro = log_reg.predict_proba(x_test_ro)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_ro = yhat_ro[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test_ro,  y_pred_ro) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_ro = roc_auc_score(y_test_ro, y_pred_ro)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_ro))
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **93.78%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **93.77%**

---

**1.2 Machine Learning Model - Logistic Regression - using Random Over sampler(SMOTE)**

**#Split the dataset into test train**

```
x_train_ro2, x_test_ro2, y_train_ro2, y_test_ro2 = train_test_split(x_ros1,y_ros1, test_size = 0.3,
random_state = 0)
```

**#train the model**

```
log_reg = LogisticRegression()
log_reg.fit(x_train_ro2, y_train_ro2)
#predict the model
y_pred_ro2 = log_reg.predict(x_test_ro2)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of ROS using SMOTE : {accuracy_score(y_pred_ro2, y_test_ro2)}")
```

**RESULT**

**Accuracy_Score Logistics Regression (SMOTE): 0.9377942007154961**

**STEPS FOR CREATING ROC CURVE**

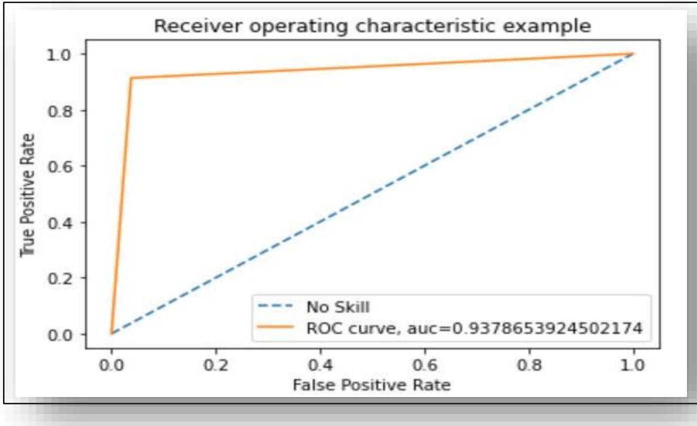
**# predict probabilities**

```
yhat = log_reg.predict_proba(x_test)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs = yhat[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test,  y_pred) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc = roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **93.78%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **93.77%**

*****************************************************************************************

**Step- 7 - Now applying different models and evaluating the dataset**

!pip3 install xgboost  (to use xgboost classifier)

**#checking for different packages for different classifiers**

from sklearn.svm import SVC # Support Vector Classifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.naive_bayes import GaussianNB

from xgboost import XGBClassifier

---

**Classifier 2 - Decision Tree Classifier**

**#Split the dataset into test train**

x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(x, y, test_size=0.3, random_state=0)

**#train the model**

dte = DecisionTreeClassifier()

dte.fit(x_train_1, y_train_1)

#predict the model

y_pred_1 = dte.predict(x_test_1)

**#check for the accuracy score**

print(f"Accuracy_Score of Decision Tree Classifier : {accuracy_score(y_pred_1, y_test_1)}")

**RESULT**

**Accuracy_Score of Decision Tree Classifier : 0.9992715994266782**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**

yhat_1 = dte.predict_proba(x_test_1)#[::,1]

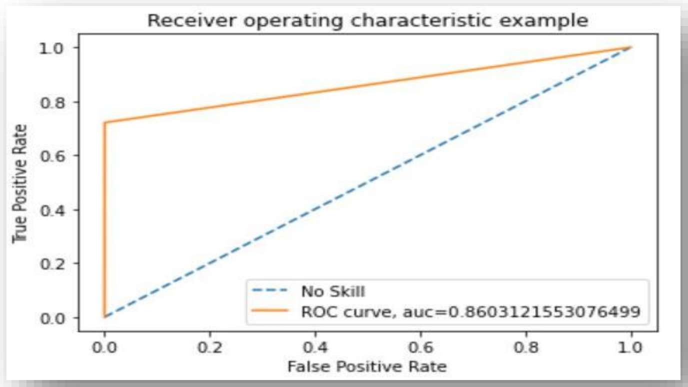
# retrieve just the probabilities for the positive class

pos_probs_1 = yhat_1[:, 1]

```
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

# calculate roc curve for model
fpr, tpr, _ = roc_curve(y_test_1,  y_pred_1) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_1 = roc_auc_score(y_test_1, y_pred_1)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_1))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**

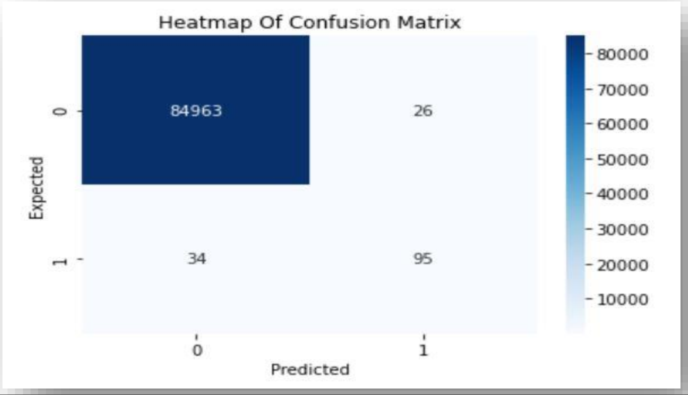



**Points noted**

This ROC -AUC curve shows the accuracy to be **86.03%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.92%**

```
#create the heatmap of the confusion matrix
sns.heatmap(pd.DataFrame(confusion_matrix(y_test_1,y_pred_1)), annot = True, cmap = 'Blues', fmt = 
'd')
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.title('Heatmap Of Confusion Matrix')
plt.show()
```

**RESULT**




**Points noted**

The heatmap of the confusion matrix also shows that most of the dataset has been rightly classified except a few

```
#check for the classification report
print(classification_report(y_test_1,y_pred_1))
```

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84989 |
| 1 | 0.78 | 0.72 | 0.75 | 129 |
| accuracy | | | 1.00 | 85118 |
| macro avg | 0.89 | 0.86 | 0.87 | 85118 |
| weighted avg | 1.00 | 1.00 | 1.00 | 85118 |

**2.1 Machine Learning Model - Decision Tree Classifier - using Random Under sampler**

**#Split the dataset into test train**

```
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(x_rus, y_rus, test_size=0.3, random_state=0)
```

**#train the model**

```
dte = DecisionTreeClassifier()
dte.fit(x_train_2, y_train_2)
#predict the model
y_pred_2 = dte.predict(x_test_2)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of Decision Tree Classifier(RUS) : {accuracy_score(y_pred_2, y_test_2)}")
```

**RESULT**

**Accuracy_Score of Decision Tree Classifier(RUS) : 0.8626760563380281**

**STEPS FOR CREATING ROC CURVE**

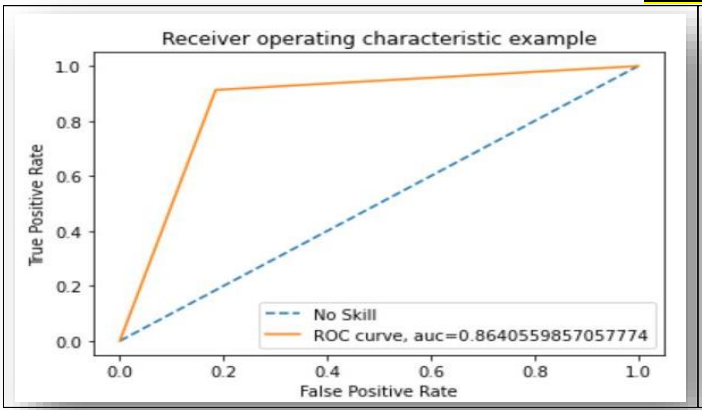
**# predict probabilities**

```
yhat_2 = dte.predict_proba(x_test_2)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_2 = yhat_2[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test_2,  y_pred_2) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_2 = roc_auc_score(y_test_2, y_pred_2)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_2))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **86.40%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **86.26%**

---

**2.1 Machine Learning Model - Decision Tree Classifier - using Random Over sampler**

**#Split the dataset into test train**

```
x_train_3, x_test_3, y_train_3, y_test_3 = train_test_split(x_ros, y_ros, test_size=0.3, random_state=0)
```

**#train the model**

```
dte = DecisionTreeClassifier()
dte.fit(x_train_3, y_train_3)
```

#predict the model
y_pred_3 = dte.predict(x_test_3)

**#check for the accuracy score**
print(f"Accuracy_Score of Decision Tree Classifier(ROS) : {accuracy_score(y_pred_3, y_test_3)}")

**RESULT**

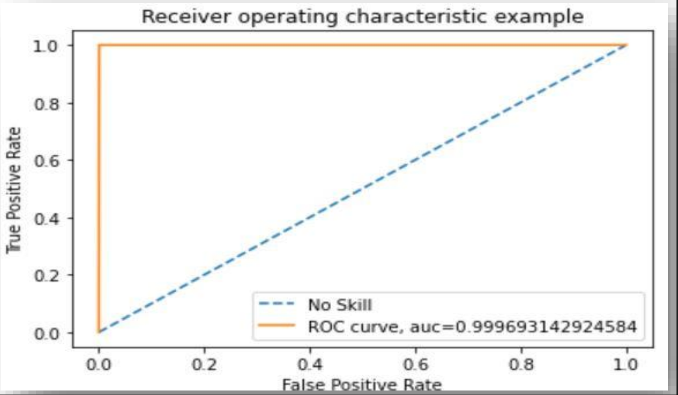
**Accuracy_Score of Decision Tree Classifier(ROS) : 0.9996704951986444**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**
yhat_3 = dte.predict_proba(x_test_3)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_3 = yhat_3[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**
fpr, tpr, _ = roc_curve(y_test_3, y_pred_3) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_3 = roc_auc_score(y_test_3, y_pred_3
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_3))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**
This ROC -AUC curve shows the accuracy to be **99.96%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.96%**

#check for the classification report
print(classification_report(y_test_3,y_pred_3))

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84730 |
| 1 | 1.00 | 1.00 | 1.00 | 85222 |
| accuracy | | | 1.00 | 169952 |
| macro avg | 1.00 | 1.00 | 1.00 | 169952 |
| weighted avg | 1.00 | 1.00 | 1.00 | 169952 |

-----------------------------------------------------------------------------------------------------------------------------

**Classifier 3 - Random forest Classifier**

**#Split the dataset into test train**

```
x_train_4, x_test_4, y_train_4, y_test_4 = train_test_split(x, y, test_size=0.3, random_state=0)
```

**#train the model**

```
rfc = RandomForestClassifier()
rfc.fit( x_train_4, y_train_4 )
#predict the model
y_pred_4 = rfc.predict(x_test_4)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of Random Forest Classifier : {accuracy_score(y_pred_4, y_test_4)}")
```

**RESULT**

**Accuracy_Score of Random Forest Classifier : 0.9995535609389319**

**STEPS FOR CREATING ROC CURVE**

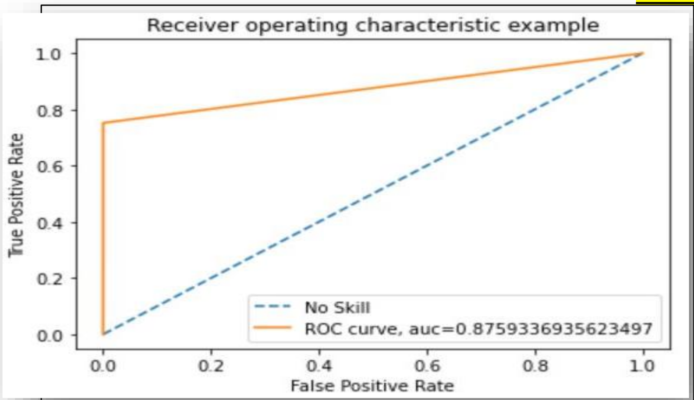
**# predict probabilities**

```
yhat_4 = rfc.predict_proba(x_test_4)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_4 = yhat_4[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test_4,  y_pred_4) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_4 = roc_auc_score(y_test_4, y_pred_4)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_4))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **87.59%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.95%**

**3.1 Machine Learning Model - Random Forest Classifier - using Random Under sampler**

**#Split the dataset into test train**

```
x_train_5, x_test_5, y_train_5, y_test_5 = train_test_split(x_rus, y_rus, test_size=0.3, random_state=0)
```

**#train the model**

```
rfc = RandomForestClassifier()
```

```
rfc.fit( x_train_5, y_train_5 )
#predict the model
y_pred_5 = rfc.predict(x_test_5)
```

**#check for the accuracy score**

```
print(f"Accuracy_Score of Random Forest Classifier(RUS) : {accuracy_score(y_pred_5, y_test_5)}")
```

**RESULT**

Accuracy_Score of **Random Forest Classifier(RUS) : 0.9401408450704225**

**STEPS FOR CREATING ROC CURVE**

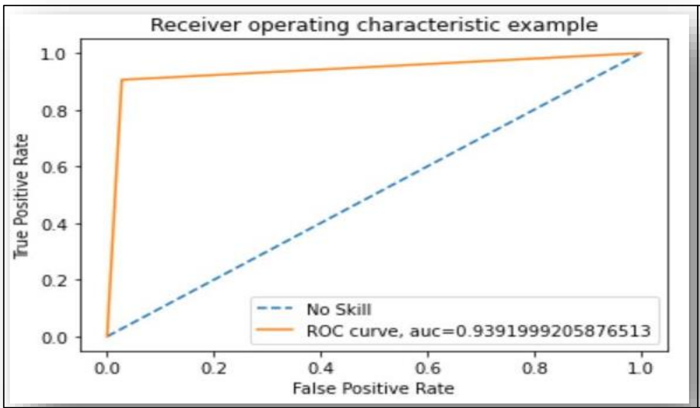
**# predict probabilities**

```
yhat_5 = rfc.predict_proba(x_test_5)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_5 = yhat_5[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
```

**# calculate roc curve for model**

```
fpr, tpr, _ = roc_curve(y_test_5,  y_pred_5) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_5 = roc_auc_score(y_test_5, y_pred_5)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_5))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()
```

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **93.91%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **94.01%**

---

**3.2 Machine Learning Model - Random Forest Classifier - using Random Over sampler**

**#Split the dataset into test train**

```
x_train_6, x_test_6, y_train_6, y_test_6 = train_test_split(x_ros, y_ros, test_size=0.3, random_state=0)
```

**#train the model**

```
rfc = RandomForestClassifier()
rfc.fit( x_train_6, y_train_6 )

#predict the model
y_pred_6 = rfc.predict(x_test_6)
```

**#check for the accuracy score**

print(f"Accuracy_Score of Random Forest Classifier(ROS) : {accuracy_score(y_pred_6, y_test_6)}")

**RESULT**

Accuracy_Score of **Random Forest Classifier(ROS) : 0.9999117397853512**

**STEPS FOR CREATING ROC CURVE**

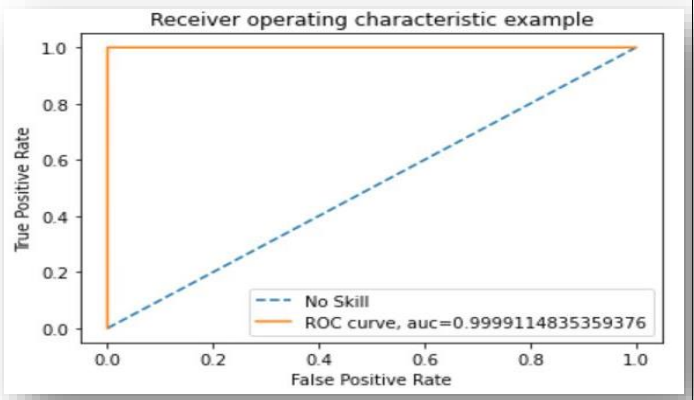
**# predict probabilities**

yhat_6 = rfc.predict_proba(x_test_6)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_6 = yhat_6[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_6,  y_pred_6) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_6 = roc_auc_score(y_test_6, y_pred_6)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_6))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **99.99%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.99%**

#check for the classification report

print(classification_report(y_test_6,y_pred_6))

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84730 |
| 1 | 1.00 | 1.00 | 1.00 | 85222 |
| accuracy | | | 1.00 | 169952 |
| macro avg | 1.00 | 1.00 | 1.00 | 169952 |
| weighted avg | 1.00 | 1.00 | 1.00 | 169952 |

----------------------------------------------------------------------------------------------------------------------------

# Classifier 4 -K Neighbours Classifier

**#Split the dataset into test train**

x_train_7, x_test_7, y_train_7, y_test_7 = train_test_split(x, y, test_size=0.3, random_state=0)

**#train the model**

knc =KNeighborsClassifier(n_neighbors=3)

knc.fit( x_train_7, y_train_7 )
#predict the model
y_pred_7 = knc.predict(x_test_7)

**#check for the accuracy score**

print(f"Accuracy_Score of K Neighbours Classifier : {accuracy_score(y_pred_7, y_test_7)}")

**RESULT**

**Accuracy_Score of K Neighbours Classifier: 0.9985314504570126**

**STEPS FOR CREATING ROC CURVE**

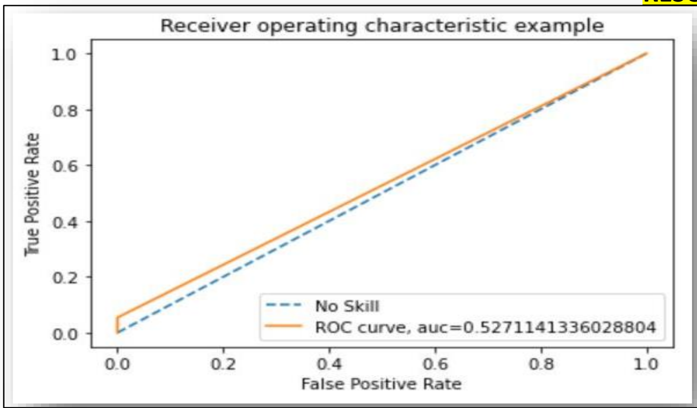
**# predict probabilities**

yhat_7 = knc.predict_proba(x_test_7)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_7 = yhat_7[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_7,  y_pred_7) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_7 = roc_auc_score(y_test_7, y_pred_7)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_7))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **52.71%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.85%**

---

**4.1 Machine Learning Model - K Neighbouring Classifier - using Random Under sampler**

**#Split the dataset into test train**

x_train_8, x_test_8, y_train_8, y_test_8 = train_test_split(x_rus, y_rus, test_size=0.3, random_state=0)

**#train the model**
knc =KNeighborsClassifier(n_neighbors=3)
knc.fit( x_train_8, y_train_8 )
#predict the model
y_pred_8 = knc.predict(x_test_8)

**#check for the accuracy score**
print(f"Accuracy_Score of K Neighbours Classifier(RUS) : {accuracy_score(y_pred_8, y_test_8)}")

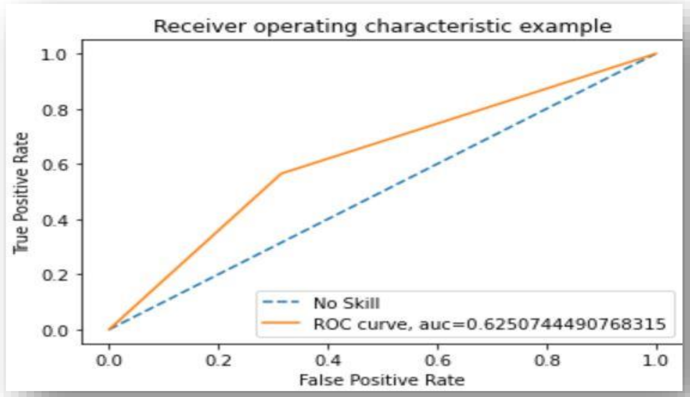
**RESULT**

**Accuracy_Score of K Neighbours Classifier(RUS): 0.6267605633802817**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**
yhat_8 = knc.predict_proba(x_test_8)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_8 = yhat_8[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
**# calculate roc curve for model**
fpr, tpr, _ = roc_curve(y_test_8, y_pred_8) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_8 = roc_auc_score(y_test_8, y_pred_8
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_8))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**
This ROC -AUC curve shows the accuracy to be **62.50%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **62.67%**

---

**4.2 Machine Learning Model - K Neighbouring Classifier - using Random Over sampler**

**#Split the dataset into test train**
x_train_9, x_test_9, y_train_9, y_test_9 = train_test_split(x_ros, y_ros, test_size=0.3, random_state=0)
**#train the model**
knc =KNeighborsClassifier(n_neighbors=3)
knc.fit( x_train_9, y_train_9 )
#predict the model
y_pred_9 = knc.predict(x_test_9)

**#check for the accuracy score**

print(f"Accuracy_Score of K Neighbours Classifier(ROS) : {accuracy_score(y_pred_9, y_test_9)}")

**RESULT**

**Accuracy_Score of K Neighbours Classifier(ROS): 0.9993998305403878**

**STEPS FOR CREATING ROC CURVE**

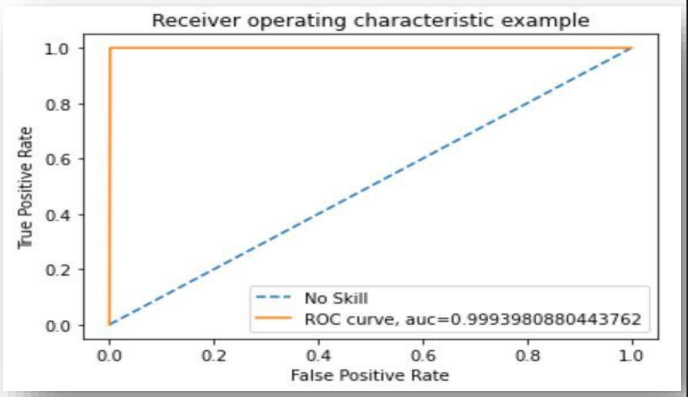
**# predict probabilities**

yhat_9 = knc.predict_proba(x_test_9)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_9 = yhat_9[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_9,  y_pred_9) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_9 = roc_auc_score(y_test_9, y_pred_9)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_9))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **99.93%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.93%**

---

**# Classifier 5- XGBoost Classifier**

#convert x, y dataset into an optimized data structure called Dmatrix that XGBoost supports and which increases productivity
#data_dmatrix = XGBClassifier.DMatrix(data=x,label=y)

**#Split the dataset into test train**

x_train_10, x_test_10, y_train_10, y_test_10 = train_test_split(x, y, test_size=0.3, random_state=0)

**#train the model**

xgbc =XGBClassifier(n_estimators=100)
xgbc.fit( x_train_10, y_train_10 )
#predict the model
y_pred_10 = xgbc.predict(x_test_10)

**#check for the accuracy score**

print(f"Accuracy_Score of XGBoost Classifier : {accuracy_score(y_pred_10, y_test_10)}")

**RESULT**

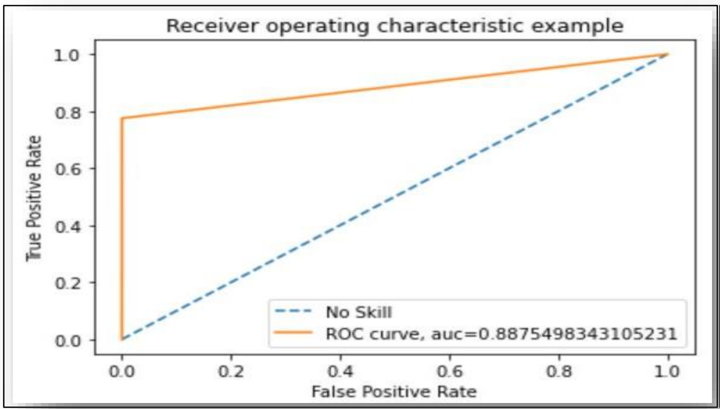
**Accuracy_Score of XGBoost Classifier: 0.9995653093352758**

**STEPS FOR CREATING ROC CURVE**

**# predict probabilities**
yhat_10 = xgbc.predict_proba(x_test_10)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_10 = yhat_10[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**
fpr, tpr, _ = roc_curve(y_test_10,  y_pred_10) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_10 = roc_auc_score(y_test_10, y_pred_10) #should we be using predicted value or predicted prob instead to build the auc?
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_10))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**
This ROC -AUC curve shows the accuracy to be **88.75%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.95%**

#check for the classification report
print(classification_report(y_test_10,y_pred_10))

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84989 |
| 1 | 0.93 | 0.78 | 0.84 | 129 |
| accuracy | | | 1.00 | 85118 |
| macro avg | 0.96 | 0.89 | 0.92 | 85118 |
| weighted avg | 1.00 | 1.00 | 1.00 | 85118 |

---

## 5.1 Machine Learning Model - XGBoost Classifier - using Random Under sampler

#convert x, y dataset into an optimized data structure called Dmatrix that XGBoost supports and which increases productivity
#data_dmatrix = XGBClassifier.DMatrix(data=x,label=y)

**#Split the dataset into test train**

x_train_11, x_test_11, y_train_11, y_test_11 = train_test_split(x_rus, y_rus, test_size=0.3, random_state=0)

**#train the model**

xgbc =XGBClassifier(n_estimators=100)
xgbc.fit( x_train_11, y_train_11 )
#predict the model
y_pred_11 = xgbc.predict(x_test_11)

**#check for the accuracy score**

print(f"Accuracy_Score of XGBoost Classifier(RUS) : {accuracy_score(y_pred_11, y_test_11)}")

**RESULT**

**Accuracy_Score of XGBoost Classifier(RUS) : 0.9401408450704225**

**STEPS FOR CREATING ROC CURVE**

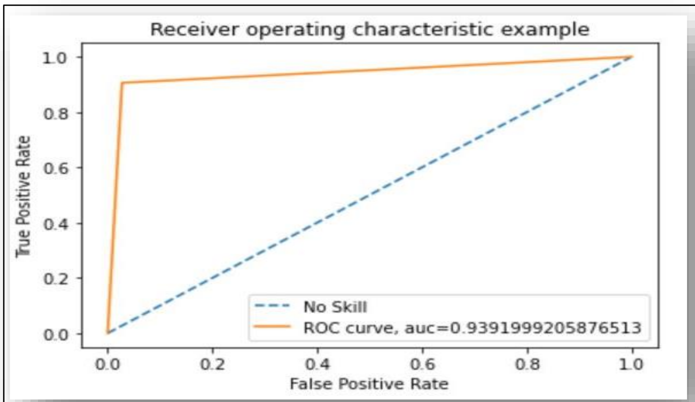
**# predict probabilities**

yhat_11 = xgbc.predict_proba(x_test_11)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_11 = yhat_11[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_11, y_pred_11) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_11 = roc_auc_score(y_test_11, y_pred_11) #should we be using predicted value or predicted prob instead to build the auc?
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_11))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **93.91%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **94.01%**

---

**5.2 Machine Learning Model - XGBoost Classifier - using Random Over sampler**

#convert x, y dataset into an optimized data structure called Dmatrix that XGBoost supports and which increases productivity
#data_dmatrix = XGBClassifier.DMatrix(data=x,label=y)

**#Split the dataset into test train**

x_train_12, x_test_12, y_train_12, y_test_12 = train_test_split(x_ros, y_ros, test_size=0.3, random_state=0)

**#train the model**

xgbc =XGBClassifier(n_estimators=100)
xgbc.fit( x_train_12, y_train_12 )
#predict the model
y_pred_12 = xgbc.predict(x_test_12)

**#check for the accuracy score**

print(f"Accuracy_Score of XGBoost Classifier(ROS) : {accuracy_score(y_pred_12, y_test_12)}")

**RESULT**

**Accuracy_Score of XGBoost Classifier(ROS) : 0.9998882037281115**

**STEPS FOR CREATING ROC CURVE**

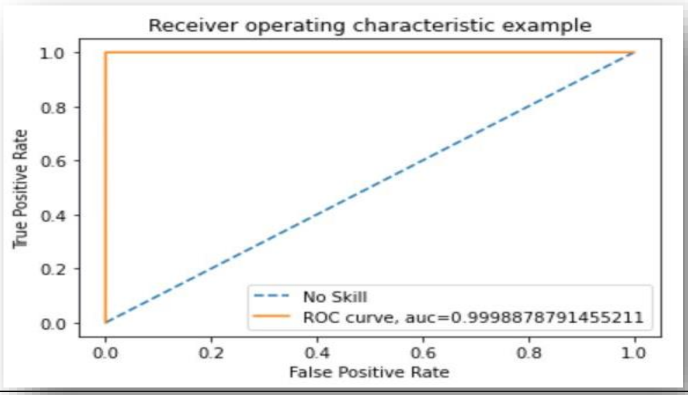
**# predict probabilities**

yhat_12 = xgbc.predict_proba(x_test_12)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_12 = yhat_12[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_12, y_pred_12) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_12 = roc_auc_score(y_test_12, y_pred_12) #should we be using predicted value or predicted prob instead to build the auc?
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_12))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **99.98%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.98%**

#check for the classification report

print(classification_report(y_test_12,y_pred_12))

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 84730 |
| 1 | 1.00 | 1.00 | 1.00 | 85222 |
| accuracy | | | 1.00 | 169952 |
| macro avg | 1.00 | 1.00 | 1.00 | 169952 |
| weighted avg | 1.00 | 1.00 | 1.00 | 169952 |

---

**6. Machine Learning Model--Naive Bayes Classifier**

**#Split the dataset into test train**

x_train_13, x_test_13, y_train_13, y_test_13 = train_test_split(x, y, test_size=0.3, random_state=0)

**#train the model**

gnb = GaussianNB()
gnb.fit( x_train_13, y_train_13)
#predict the model
y_pred_13 = gnb.predict(x_test_13)

**#check for the accuracy score**

print(f"Accuracy_Score of Naive Bayes Classifier: {accuracy_score(y_pred_13, y_test_13)}")

**RESULT**

**Accuracy_Score Naive Bayes Classifier 0.9928804718155971**

**STEPS FOR CREATING ROC CURVE**

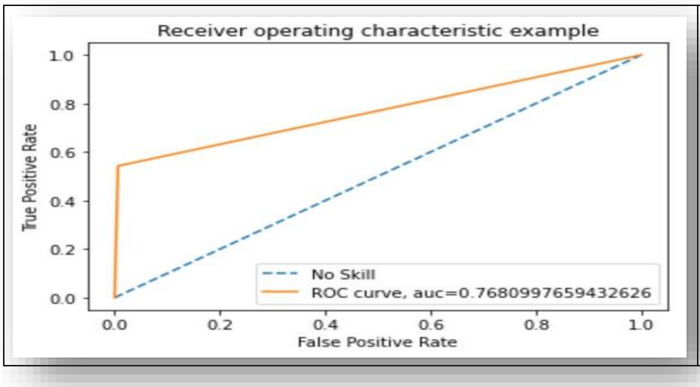
**# predict probabilities**

yhat_13 = gnb.predict_proba(x_test_13)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_13 = yhat_13[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_13, y_pred_13) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_13 = roc_auc_score(y_test_13, y_pred_13)
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_13))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **76.80%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **99.28%**

**6.1 Machine Learning Model - Naive Bayes Classifier - using Random Under sampler**

**#Split the dataset into test train**

x_train_14, x_test_14, y_train_14, y_test_14 = train_test_split(x_rus, y_rus, test_size=0.3, random_state=0)

**#train the model**

gnb = GaussianNB()
gnb.fit( x_train_14, y_train_14)
#predict the model
y_pred_14 = gnb.predict(x_test_14)

**#check for the accuracy score**

print(f"Accuracy_Score of Naive Bayes Classifier(RUS) : {accuracy_score(y_pred_14, y_test_14)}")

**RESULT**

**Accuracy_Score of Naive Bayes Classifier(RUS) : 0.8767605633802817**

**STEPS FOR CREATING ROC CURVE**

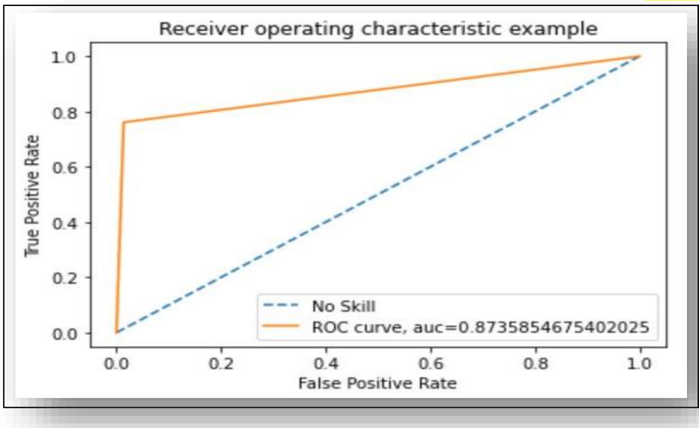
**# predict probabilities**

yhat_14 = gnb.predict_proba(x_test_14)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_14 = yhat_14[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_14,  y_pred_14) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_14 = roc_auc_score(y_test_14, y_pred_14
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_14))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **87.35%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **87.67%**

-----------------------------------------------------------------------------------------------------------------------------

**6.2 Machine Learning Model - Naive Bayes Classifier - using Random Over sampler**

**#Split the dataset into test train**

x_train_15, x_test_15, y_train_15, y_test_15 = train_test_split(x_ros, y_ros, test_size=0.3, random_state=0)

**#train the model**

gnb = GaussianNB()
gnb.fit( x_train_15, y_train_15)
#predict the model
y_pred_15 = gnb.predict(x_test_15)

**#check for the accuracy score**

print(f"Accuracy_Score of Naive Bayes Classifier(ROS) : {accuracy_score(y_pred_15, y_test_15)}")

**RESULT**

**Accuracy_Score of Naive Bayes Classifier(ROS) : 0.8757590378459801**

**STEPS FOR CREATING ROC CURVE**

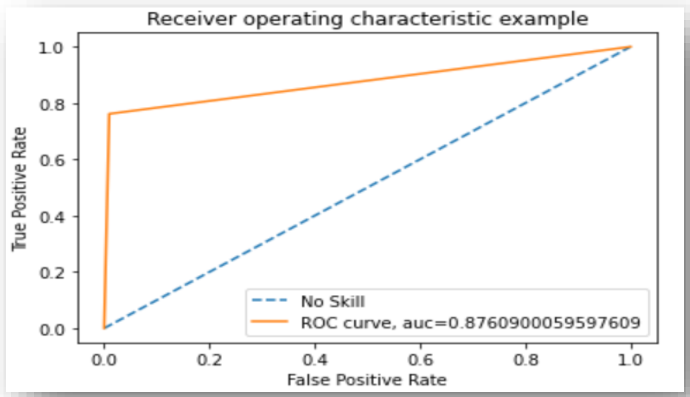
**# predict probabilities**

yhat_15 = gnb.predict_proba(x_test_15)#[::,1]
# retrieve just the probabilities for the positive class
pos_probs_15 = yhat_15[:, 1]
# plot no skill roc curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')

**# calculate roc curve for model**

fpr, tpr, _ = roc_curve(y_test_15,  y_pred_15) #fpr, tpr = false positive rate , true positive rate
# calculate roc score for model
auc_15 = roc_auc_score(y_test_15, y_pred_15
plt.plot(fpr,tpr,label="ROC curve, auc="+str(auc_15))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc=4)
plt.show()

**RESULT**




**Points noted**

This ROC -AUC curve shows the accuracy to be **87.60%** from the plot, whereas actual score from Logistic Regression shows the accuracy to be **87.57%**

#check for the classification report
print(classification_report(y_test_15,y_pred_15))

**RESULT**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.81 | 0.99 | 0.89 | 84730 |
| 1 | 0.99 | 0.76 | 0.86 | 85222 |
| accuracy | | | 0.88 | 169952 |
| macro avg | 0.90 | 0.88 | 0.87 | 169952 |
| weighted avg | 0.90 | 0.88 | 0.87 | 169952 |

*************************************************************************************

| Step- 8 - Check with all the metrics in order to compare at once for each classifier | |
|---|---|
| **Codes** | **RESULT** |
| **#Check for Accuracy score per classifier**<br>log_reg_acc = accuracy_score(y_pred_ro, y_test_ro)*100<br>dtc_acc = accuracy_score(y_pred_3, y_test_3)*100<br>rfc_acc = accuracy_score(y_pred_6, y_test_6)*100<br>knn_acc = accuracy_score(y_pred_9, y_test_9)*100<br>xgb_acc = accuracy_score(y_pred_12, y_test_12)*100<br>gnb_acc = accuracy_score(y_pred_15, y_test_15)*100<br>print(f"log_reg_acc : {log_reg_acc}")<br>print(f"dtc_acc: {dtc_acc}")<br>print(f"rfc_acc: {rfc_acc}")<br>print(f"knn_acc: {knn_acc}")<br>print(f"xgb_acc: {xgb_acc}")<br>print(f"gnb_acc : {gnb_acc}") | log_reg_acc : 93.77942007154961<br>dtc_acc: 99.97057992845039<br>rfc_acc: 99.9929391828281<br>knn_acc: 99.93998305403878<br>xgb_acc: 99.98882037281115<br>gnb_acc : 87.57590378459801 |
| **#Check for roc_auc_score per classifier**<br>log_reg_roc = roc_auc_score(y_pred_ro, y_test_ro)<br>dtc_roc = roc_auc_score(y_pred_3, y_test_3)<br>rfc_roc = roc_auc_score(y_pred_6, y_test_6)<br>knn_roc = roc_auc_score(y_pred_9, y_test_9)<br>xgb_roc = roc_auc_score(y_pred_12, y_test_12)<br>gnb_roc = roc_auc_score(y_pred_15, y_test_15)<br>print(f"log_reg_roc : {log_reg_roc}")<br>print(f"dtc_roc : {dtc_roc}")<br>print(f"rfc_roc : {rfc_roc}")<br>print(f"knn_roc : {knn_roc}")<br>print(f"xgb_roc : {xgb_roc}")<br>print(f"gnb_roc : {gnb_roc}") | log_reg_roc : 0.9388166226357336<br>dtc_roc : 0.9997068205272539<br>rfc_roc : 0.999296055564681<br>knn_roc : 0.9994022783741972<br>xgb_roc : 0.999885512840066<br>gnb_roc : 0.8964197158728477 |
| **#Check for precision score per classifier**<br>log_reg_prc = precision_score(y_pred_ro, y_test_ro)<br>dtc_prc = precision_score(y_pred_3, y_test_3)<br>rfc_prc = precision_score(y_pred_6, y_test_6)<br>knn_prc = precision_score(y_pred_9, y_test_9)<br>xgb_prc = precision_score(y_pred_12, y_test_12)<br>gnb_prc = precision_score(y_pred_15, y_test_15)<br>print(f"log_reg_prc : {log_reg_prc}")<br>print(f"dtc_prc : {dtc_prc}")<br>print(f"rfc_prc : {rfc_prc}")<br>print(f"knn_prc : {knn_prc}")<br>print(f"xgb_prc : {xgb_prc}")<br>print(f"gnb_prc : {gnb_prc}") | log_reg_prc : 0.913273567858064 8<br>dtc_prc : 1.0<br>rfc_prc : 1.0<br>knn_prc : 1.0<br>xgb_prc : 1.0<br>gnb_prc : 0.761763394428668 7 |

| | |
|---|---|
| **#Check for recall score per classifier**<br>log_reg_rcs = recall_score(y_pred_ro, y_test_ro)<br>dtc_rcs = recall_score(y_pred_3, y_test_3)<br>rfc_rcs = recall_score(y_pred_6, y_test_6)<br>knn_rcs = recall_score(y_pred_9, y_test_9)<br>xgb_rcs = recall_score(y_pred_12, y_test_12)<br>gnb_rcs = recall_score(y_pred_15, y_test_15)<br>print(f"log_reg_rcs : {log_reg_rcs}")<br>print(f"dtc_rcs : {dtc_rcs}")<br>print(f"rfc_rcs : {rfc_rcs}")<br>print(f"knn_rcs : {knn_rcs}")<br>print(f"xgb_rcs : {xgb_rcs}")<br>print(f"gnb_rcs : {gnb_rcs}") | log_reg_rcs : 0.9607342122154743<br>dtc_rcs : 0.9994136410545079<br>rfc_rcs : 0.9998592111129362<br>knn_rcs : 0.9988045567483943<br>xgb_rcs : 0.999777102568013<br>gnb_rcs : 0.987646620316137 |
| **#Check for F1 score**<br>log_reg_f1sc = f1_score(y_pred_ro, y_test_ro)<br>dtc_f1sc = f1_score(y_pred_3, y_test_3)<br>rfc_f1sc = f1_score(y_pred_6, y_test_6)<br>knn_f1sc = f1_score(y_pred_9, y_test_9)<br>xgb_f1sc = f1_score(y_pred_12, y_test_12)<br>gnb_f1sc = f1_score(y_pred_15, y_test_15)<br>print(f"log_reg_f1sc : {log_reg_f1sc}")<br>print(f"dtc_f1sc : {dtc_f1sc}")<br>print(f"rfc_f1sc : {rfc_f1sc}")<br>print(f"knn_f1sc : {knn_f1sc}")<br>print(f"xgb_f1sc : {xgb_f1sc}")<br>print(f"gnb_f1sc : {gnb_f1sc}") | log_reg_f1sc : 0.9364029019334191<br>dtc_f1sc : 0.9997067345478434<br>rfc_f1sc : 0.9999296006007415<br>knn_f1sc : 0.9994019208893787<br>xgb_f1sc : 0.9998885388618057<br>gnb_f1sc : 0.8601220247361763 |

**#Compare results of different metrics based on each classifier**

```
results = pd.DataFrame({
   'Model': ['Logistic Regression', 'Decision Tree',
         'Random Forest','KNN','XGBoost' ,'Naive
Bayes'],

   'Accuracy_Score': [log_reg_acc,
              dtc_acc,
              rfc_acc,
              knn_acc,
              xgb_acc,
              gnb_acc],
   'ROC_AUC_Score': [log_reg_roc,
         dtc_roc,
         rfc_roc,
         knn_roc,
         xgb_roc,
         gnb_roc],
   'Precision_Score': [log_reg_prc,
         dtc_prc,
         rfc_prc,
         knn_prc,
         xgb_prc,
         gnb_prc],
```

```
 'Recall_Score': [log_reg_rcs,
          dtc_rcs,
          rfc_rcs,
          knn_rcs,
          xgb_rcs,
          gnb_rcs],
   'F1_Score': [log_reg_f1sc,
          dtc_f1sc,
          rfc_f1sc,
          knn_f1sc,
          xgb_f1sc,
          gnb_f1sc]})
result_df =
results.sort_values(by='Accuracy_Score',
ascending=False)
result_df = result_df.set_index('Model')
result_df
```

**RESULT**

| | Accuracy_Score | ROC_AUC_Score | Precision_Score | Recall_Score | F1_Score |
|---|---|---|---|---|---|
| **Model** | | | | | |
| **Random Forest** | 99.992939 | 0.999930 | 1.000000 | 0.999859 | 0.999930 |
| **XGBoost** | 99.988820 | 0.999889 | 1.000000 | 0.999777 | 0.999889 |
| **Decision Tree** | 99.970580 | 0.999707 | 1.000000 | 0.999414 | 0.999707 |
| **KNN** | 99.939983 | 0.999402 | 1.000000 | 0.998805 | 0.999402 |
| **Logistic Regression** | 93.779420 | 0.938817 | 0.913274 | 0.960734 | 0.936403 |
| **Naive Bayes** | 87.575904 | 0.896420 | 0.761763 | 0.987647 | 0.860122 |

**Part 8.1. Evaluating models based on various metrics**

As we can see from the comparisons above done for all the different models based on several metrics

The most accurate result has been provided by **Random Forest (Over Sampling method)**
Here, the oversampling method has been used because the data being imbalanced and most of the data belonged to Legit cases.

## WAYS TO IMPROVE THE MODEL

- As the dataset provided had been imbalanced, it was important to balance it either by under sampling or by over sampling method.
- Mostly in the models used, oversampling seemed to be useful and more efficient