# CREDIT CARD FRAUD DETECTION

*By- Muqaddaspreet Singh*

## 1. Introduction

Detecting fraudulent transactions is of great importance for any credit card company. We are tasked by a well-known company to detect potential fraud so that customers are not charged for items that they did not purchase. So, the goal is to build a classifier that tells if a transaction is afraud or not.

## 2. Need of the System

From the moment payment systems came into existence, there have always been people who will find new ways to access someone's finances illegally. This has become a major problem in the modern era, as all transactions can easily be completed online by only entering your credit card information. Even in the 2010s, many American retail website users were the victims of online transaction fraud right before two-step verification was used for shopping online. Organizations, consumers, banks, and merchants are put at risk when a data breach leads to monetary theft and ultimately the loss of customers' loyalty along with the company's reputation.

## 3. Methodology

### 3.1 Dataset Description:

It has 8 rows and 31 columns as shown in the image below:

```
[3]: dataset=pd.read_csv('creditcard.csv')
```

```
[4]: dataset.describe()
```

[4]:

| | 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 | ... | 2.84 |
| 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 | ... | 1.53 |
| 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 | ... | 7.34 |
| 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 | ... | -3.48 |
| 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 | ... | -2.28 |
| 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 | ... | -2.94 |
| 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 | ... | 1.86 |
| 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 | ... | 2.72 |

8 rows × 31 columns

## 3.2 Checking for null values:

```
[7]: dataset.isnull().sum()
```

```
[7]: 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
```

So, there are no null values in the dataset.

```
[14]: dataset['Class'].value_counts()[0]/len(dataset['Class'])
```

```
[14]: 0.9982725143693799
```

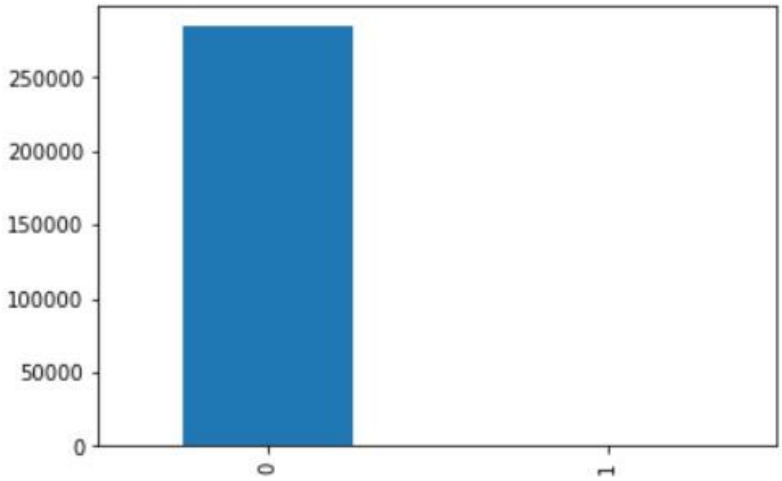
```
[15]: dataset['Class'].value_counts()[1]/len(dataset['Class'])
```

```
[15]: 0.001727485630620034
```

This implies, no fraud i.e. – 0 class the value comes out to be 99.82%, while for fraud cases the value comes out to be 0.17%.

```
In [6]: dataset['Class'].value_counts().plot(kind='bar')
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x27df27a35b0>
```




### 3.3 Scaling the Time and Amount columns using Standard Scaler:

```
In [52]: from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
dataset['scaled_amount'] = standard_scaler.fit_transform(dataset['Amount'].values.reshape(-1,1))
dataset['scaled_time'] = standard_scaler.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset.drop(['Time','Amount'], axis=1, inplace=True)
```

Now on applying the standard scaler, what we obtain is shown below:

```
In [53]: dataset
```

Out[53]:

| /6 | V7 | V8 | V9 | V10 | ... | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Class | scaled_amount | scaled_time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 08 | -0.623276 | 0.161894 | 0.413498 | 0.016350 | ... | 0.742704 | -0.081967 | 0.582521 | 0.477891 | -0.290331 | 0.070501 | 0.028411 | 0 | -0.346593 | -0.824752 |
| 31 | -0.135931 | 0.084315 | -0.186756 | 0.067928 | ... | -0.504163 | 0.183091 | 0.210636 | 0.077413 | 0.095275 | -0.017937 | 0.006714 | 0 | -0.349271 | -1.041754 |
| 32 | -0.585602 | -0.835088 | -0.664213 | 0.519685 | ... | 0.575446 | 2.455147 | -0.153624 | 0.934399 | 0.015352 | 0.887629 | -0.323593 | 0 | 0.160085 | 0.842383 |
| 38 | -0.924876 | 0.436790 | 1.706103 | -0.967010 | ... | 0.955489 | -0.074131 | -0.258075 | 0.421365 | -0.486822 | 0.119259 | 0.014301 | 0 | -0.349231 | -0.246796 |
| 59 | 0.099411 | -0.238454 | 0.899456 | -0.070752 | ... | 0.827146 | 0.022528 | 0.730417 | 0.322633 | -0.449630 | -0.002732 | -0.049188 | 0 | -0.349231 | 0.383910 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 60 | -0.635796 | 0.153732 | 0.857160 | -0.096441 | ... | 0.517560 | -0.275132 | -0.460264 | 0.411262 | -0.213036 | 0.047677 | 0.055150 | 0 | 0.176397 | -0.366805 |
| 11 | 0.135889 | -0.220195 | 0.394268 | 0.189238 | ... | 0.886726 | -0.070417 | -0.459723 | 0.295745 | -0.098154 | -0.025310 | -0.073800 | 0 | -0.342475 | 0.961592 |
| 48 | 0.299516 | 0.553846 | -0.296482 | -0.667123 | ... | -0.519899 | 0.563969 | -0.076632 | -0.348780 | 0.135857 | -0.216601 | -0.091044 | 0 | -0.313328 | 0.767480 |
| 32 | -0.862592 | -0.433468 | -1.444106 | 1.541229 | ... | 0.276233 | 0.306563 | 0.941818 | -0.264075 | -0.224091 | 0.011642 | -0.035852 | 0 | -0.193306 | 0.893722 |
| 74 | -0.342989 | -0.124194 | -1.140019 | 0.840091 | ... | -0.639834 | -0.072795 | 0.036551 | 0.252421 | 1.036004 | -0.086260 | 0.004016 | 0 | -0.013432 | -1.408014 |

So, from this, we infer that the dataset we are using is highly imbalanced as most of the cases in our dataset are pointing to “No fraud” outcome and very few are pointing towards “Fraud”. So, this would lead to overfitting and predicting wrong results. Now using the Random Under-sampling technique i.e., the Class attribute which was imbalanced in our dataset can be balanced by taking the equal size of samples i.e., distributing them equally.

## 4. Random under-sampling technique

In this technique, we generate the class 0 records equivalent to class 1 records such that we can get true results instead of biased results thereby helping to balance the dataset.

```
In [25]: dataset = dataset.sample(frac=1)
         fraud=dataset.loc[dataset['Class'] == 1]
         legit=dataset.loc[dataset['Class'] == 0][:492]
         norm_sample=pd.concat([fraud,legit])
         new_dataset=norm_sample.sample(frac=1, random_state=42)
```

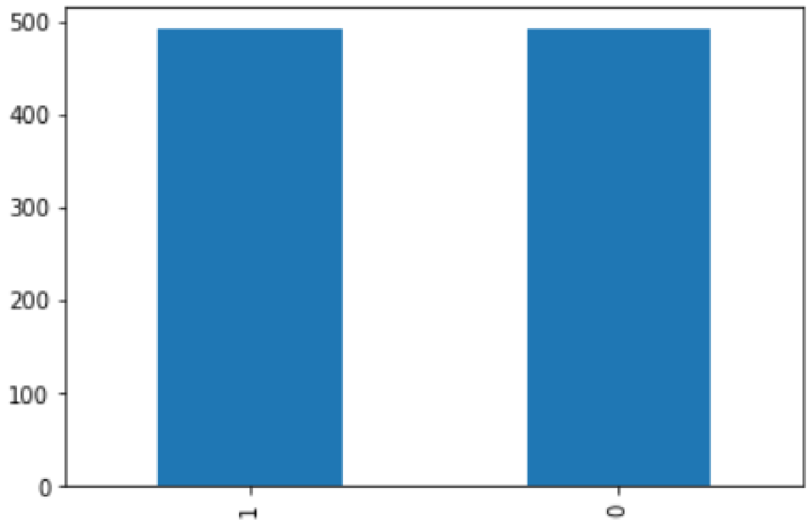
Thus, we have created a new data frame containing equal no of class 0 and class 1 cases i.e., equal no of fraud and legit transactions So output here looks like the following:

Out[56]:

| V6 | V7 | V8 | V9 | V10 | ... | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Class | scaled_amount | scaled_t |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3054 | 0.453986 | 0.254021 | -0.266435 | 0.157432 | ... | -0.005807 | -0.283351 | -0.525708 | 0.011182 | -0.213814 | 0.399174 | 0.185946 | 0 | -0.328761 | -0.761 |
| 2193 | -3.968593 | 1.063728 | -0.486097 | -4.624985 | ... | 0.109541 | 0.601045 | -0.364700 | -1.843078 | 0.351909 | 0.594550 | 0.099372 | 1 | -0.349231 | -1.836 |
| 7731 | 0.363293 | 0.462064 | 1.276429 | -1.220575 | ... | -0.455648 | -0.355893 | -0.036081 | 0.542714 | -0.451115 | -0.326282 | -0.062341 | 0 | -0.137653 | -1.579 |
| 2404 | -16.701694 | 7.517344 | -8.507059 | -14.110184 | ... | -1.127670 | -2.358579 | 0.673461 | -1.413700 | -0.462762 | -2.018575 | -1.042804 | 1 | 1.102834 | -1.122 |
| 5855 | 1.017732 | -0.544704 | -1.703378 | -3.739659 | ... | 0.092073 | -1.492882 | -0.204227 | 0.532511 | -0.293871 | 0.212663 | 0.431095 | 1 | 4.984216 | 0.807 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 4934 | -1.340739 | -0.555548 | -1.184468 | -3.245109 | ... | 0.277612 | 0.019266 | 0.508529 | -0.201183 | -0.249600 | 0.562239 | 0.075309 | 1 | 0.330123 | -0.815 |
| 9966 | 1.767760 | -2.451050 | 0.069736 | 3.245086 | ... | 0.334971 | 0.172106 | 0.623590 | -0.527114 | -0.079215 | -2.532445 | 0.311177 | 1 | 0.065810 | -1.346 |
| 9021 | -1.029212 | 0.608098 | 1.045846 | 0.005142 | ... | 0.751789 | 0.203550 | -0.877386 | -0.666852 | 1.349753 | -0.058620 | -0.070854 | 0 | -0.100470 | 1.217 |
| 6665 | -2.312223 | 0.961014 | -1.896001 | -4.919348 | ... | 0.126576 | 0.203953 | 0.008495 | -0.174501 | 0.575295 | 0.152876 | -0.098173 | 1 | 0.025869 | 1.085 |
| 4806 | -18.261393 | 17.052566 | -3.742605 | -8.233721 | ... | -1.917759 | -1.235787 | 0.161105 | 1.820378 | -0.219359 | 1.388786 | 0.406810 | 1 | 0.046539 | -1.422 |

## 4.1 Visualizing with help of a bar graph:

```
In [26]: new_dataset['Class'].value_counts().plot(kind='bar')
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1a03559c5e0>
```




i.e., 984 rows – 492 (Fraud) & 492 (Legit). Firstly, split the dataset into test and train.

```
X=new_dataset.iloc[:,:-1].values
y=new_dataset.iloc[:,-1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.25, random_state=0)
```

## 4.2. Applying classification Models:

### 4.2.1 Logistic Regression

Importing the Logistic Regression model and fitting the values.

```
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

### 4.2.1.1 Generating confusion matrix:

```
In [31]: y_pred= classifier.predict(X_test)
```

```
In [33]: from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test,y_pred)
         cm

Out[33]: array([[125,   7],
                [ 12, 102]], dtype=int64)
```

### 4.2.1.2 Checking for Precision, Recall, and F1-score:

```
In [90]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

                       precision    recall  f1-score   support

                   0       0.91      0.95      0.93       132
                   1       0.94      0.89      0.91       114

            accuracy                           0.92       246
           macro avg       0.92      0.92      0.92       246
        weighted avg       0.92      0.92      0.92       246
```

### 4.2.1.3 Checking for ROC metrics:

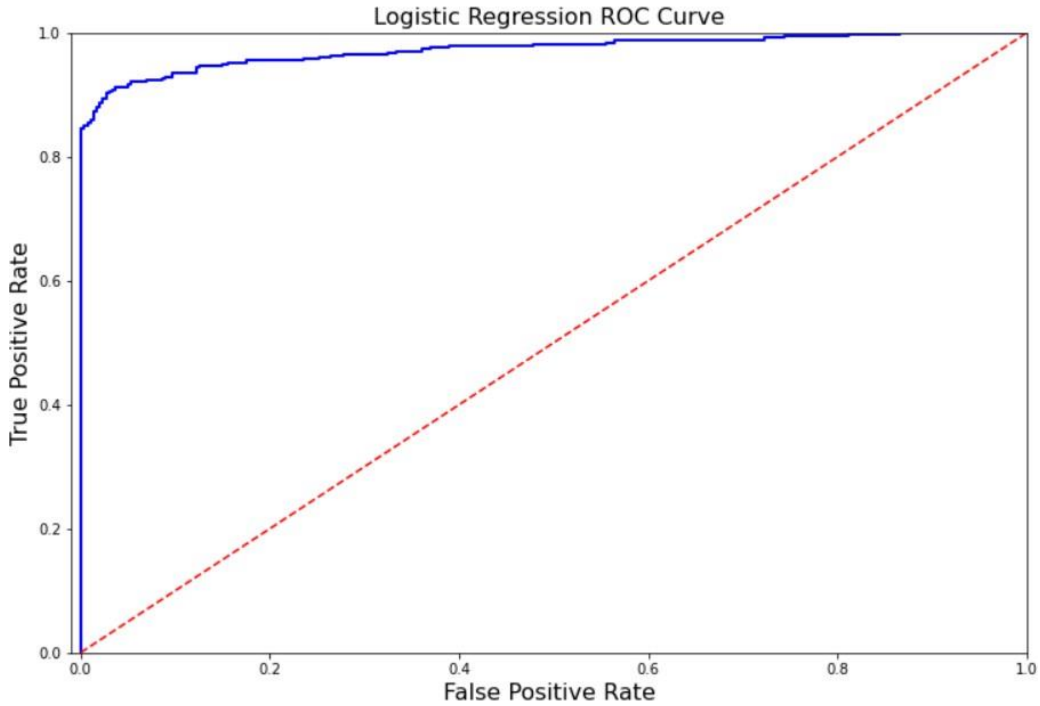
```
In [36]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         lr_pred = cross_val_predict(classifier, X_train, y_train, cv=5,method="decision_function")
         print('Logistic Regression: ', roc_auc_score(y_train, lr_pred))
         lr_fpr, lr_tpr, lr_thresold = roc_curve(y_train, lr_pred)

         Logistic Regression:  0.972854203409759
```

So, ROC accuracy comes out to be 97.28%.

### 4.2.1.4 Plotting ROC curve:

```
In [37]: def logistic_roc_curve(lr_fpr, lr_tpr):
             plt.figure(figsize=(12,8))
             plt.title('Logistic Regression ROC Curve', fontsize=16)
             plt.plot(lr_fpr, lr_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         logistic_roc_curve(lr_fpr, lr_tpr)
         plt.show()
```

#### 4.2.1.5 Plot obtained:




### 4.2.2 KNN

Importing the KNN model and fitting the values.

```
In [36]: from sklearn.neighbors import KNeighborsClassifier
         classifier1= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
         classifier1.fit(X_train, y_train)

Out[36]: KNeighborsClassifier()
```

#### 4.2.2.1 Generating confusion matrix:

```
In [94]: from sklearn.metrics import confusion_matrix
         cm1= confusion_matrix(y_test, y_pred)
         cm1

Out[94]: array([[130,   2],
                [ 14, 100]], dtype=int64)
```

### 4.2.2.2 Checking for precision, recall, and F1-score:

```
In [95]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.90      0.98      0.94       132
           1       0.98      0.88      0.93       114

    accuracy                           0.93       246
   macro avg       0.94      0.93      0.93       246
weighted avg       0.94      0.93      0.93       246
```

### 4.2.2.3 Checking for ROC metrics:

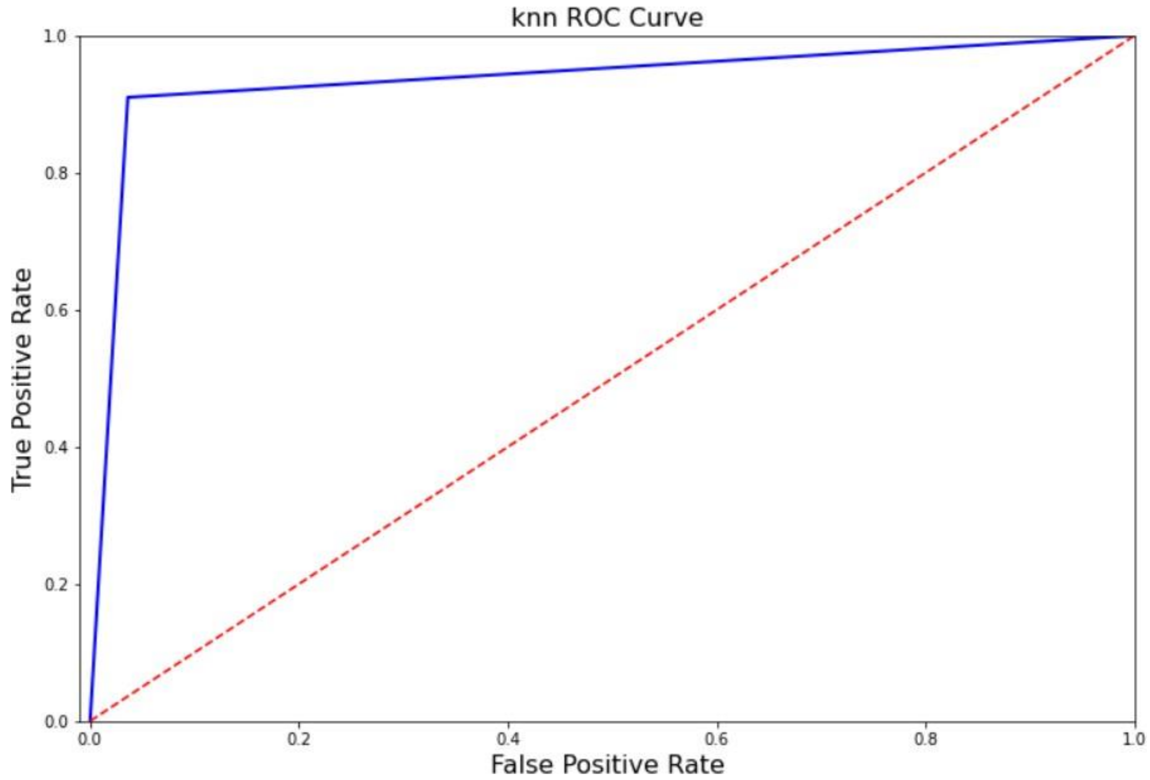
```
In [43]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         knn_pred = cross_val_predict(classifier, X_train, y_train, cv=5)
         print('KNN: ', roc_auc_score(y_train, knn_pred))
         knn_fpr, knn_tpr, knn_thresold = roc_curve(y_train, knn_pred)

         KNN:  0.9369708994708994
```

The roc accuracy score comes out to 93.69%.

### 4.2.2.4 Plotting the ROC curve:

```
In [44]: def knn_roc_curve(knn_fpr, knn_tpr):
             plt.figure(figsize=(12,8))
             plt.title('knn ROC Curve', fontsize=16)
             plt.plot(knn_fpr, knn_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         knn_roc_curve(knn_fpr, knn_tpr)
         plt.show()
```

#### 4.2.2.5 Plot obtained:




### 4.2.3 SVM

Importing the SVM model and fitting the values.

```
In [115]: from sklearn.svm import SVC
          classifier2 = SVC(kernel='rbf', random_state=0)
          classifier2.fit(X_train, y_train)
          y_pred= classifier2.predict(X_test)
```

#### 4.2.3.1 Generating confusion matrix:

```
In [116]: from sklearn.metrics import confusion_matrix
          cm2= confusion_matrix(y_test, y_pred)
          cm2

Out[116]: array([[130,   2],
                 [ 15,  99]], dtype=int64)
```

### 4.2.3.2 Checking for precision, recall, and F1-score:

```
In [117]: from sklearn.metrics import classification_report
          print(classification_report(y_test,y_pred))

                        precision    recall  f1-score   support

                     0       0.90      0.98      0.94       132
                     1       0.98      0.87      0.92       114

              accuracy                           0.93       246
             macro avg       0.94      0.93      0.93       246
          weighted avg       0.94      0.93      0.93       246
```

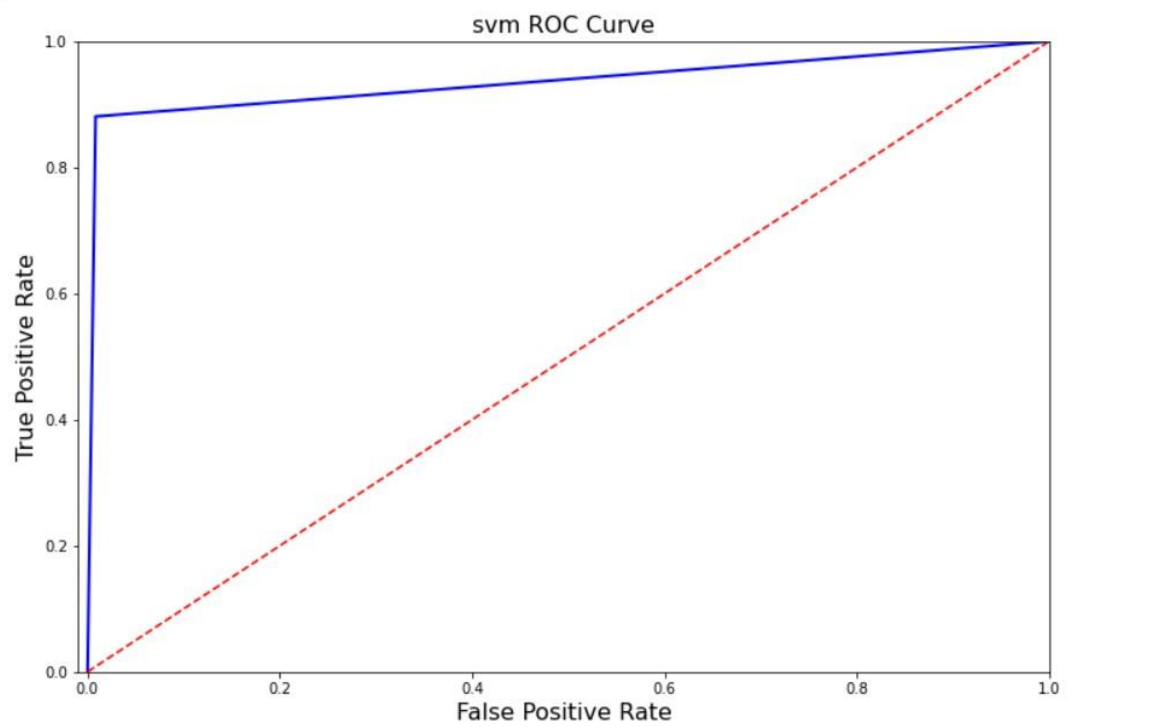
### 4.2.3.3 Checking for ROC metrics:

```
In [59]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         svm_pred = cross_val_predict(classifier2, X_train, y_train, cv=5,method="decision_function")
         print('SVM: ', roc_auc_score(y_train, knn_pred))
         svm_fpr, svm_tpr, svm_thresold = roc_curve(y_train, svm_pred)

         SVM:  0.9363095238095238
```

### 4.2.3.4 Plotting ROC curve:

```
In [60]: def svm_roc_curve(svm_fpr, svm_tpr):
             plt.figure(figsize=(12,8))
             plt.title('svm ROC Curve', fontsize=16)
             plt.plot(knn_fpr, knn_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         svm_roc_curve(svm_fpr, svm_tpr)
         plt.show()
```

### 4.2.3.5 Plot obtained:




### 4.2.4 Naïve Bayes

Importing the SVM model and fitting the values.

```
In [46]: from sklearn.naive_bayes import GaussianNB
         classifier = GaussianNB()
         classifier.fit(X_train,y_train)

Out[46]: GaussianNB()
```

### 4.2.4.1 Generating confusion matrix:

```
In [47]: y_pred = classifier.predict(X_test)

In [48]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[48]: array([[125,   7],
                [ 15,  99]], dtype=int64)
```

### 4.2.4.2 Checking for precision, recall, and F1-score: -

```
In [130]: from sklearn.metrics import classification_report
          print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92       132
           1       0.93      0.87      0.90       114

    accuracy                           0.91       246
   macro avg       0.91      0.91      0.91       246
weighted avg       0.91      0.91      0.91       246
```

### 4.2.4.3 Generating ROC metrics:

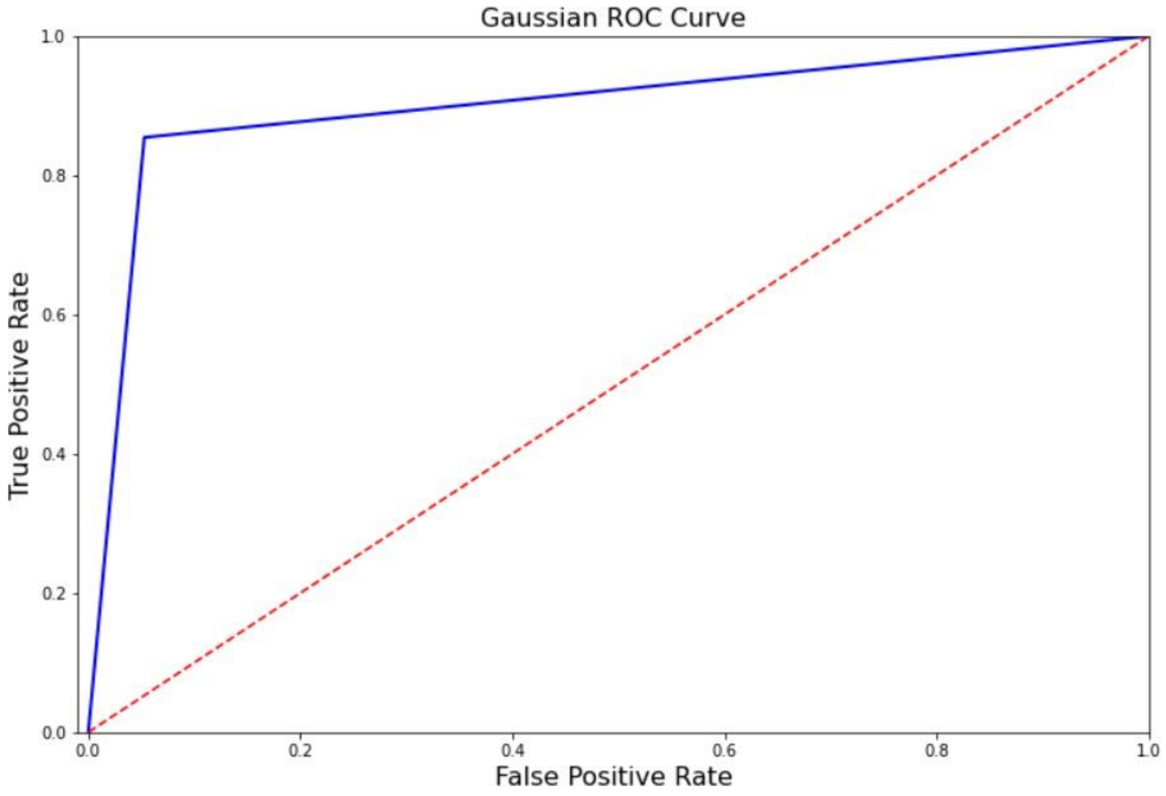
```
In [69]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         gauss_pred = cross_val_predict(classifier, X_train, y_train, cv=5)
         print('Gaussian: ', roc_auc_score(y_train, gauss_pred))
         gauss_fpr, gauss_tpr, gauss_thresold = roc_curve(y_train, gauss_pred)
```

```
Gaussian:  0.9008597883597883
```

### 4.2.4.4 Generating the ROC curve:

```
def gauss_roc_curve(gauss_fpr, gauss_tpr):
    plt.figure(figsize=(12,8))
    plt.title('Gaussian ROC Curve', fontsize=16)
    plt.plot(gauss_fpr, gauss_tpr, 'b-', linewidth=2)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.axis([-0.01,1,0,1])


gauss_roc_curve(gauss_fpr, gauss_tpr)
plt.show()
```

#### 4.2.4.5 Curve generated and obtained:




### 4.2.5 Decision Tree Classifier:

Importing the Decision Tree classifier and fitting the values.

```
In [145]: from sklearn.tree import DecisionTreeClassifier
          classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
          classifier.fit(X_train, y_train)

Out[145]: DecisionTreeClassifier(criterion='entropy', random_state=0)
```

#### 4.2.5.1 Generating confusion matrix:

```
In [146]: y_pred= classifier.predict(X_test)

In [147]: from sklearn.metrics import confusion_matrix
          cm= confusion_matrix(y_test, y_pred)
          cm

Out[147]: array([[116,  16],
                 [ 12, 102]], dtype=int64)
```

### 4.2.5.2 Checking for precision, recall, andF1-score:

In [148]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.89       132
           1       0.86      0.89      0.88       114

    accuracy                           0.89       246
   macro avg       0.89      0.89      0.89       246
weighted avg       0.89      0.89      0.89       246
```

### 4.2.5.3 Generating ROC metrics:

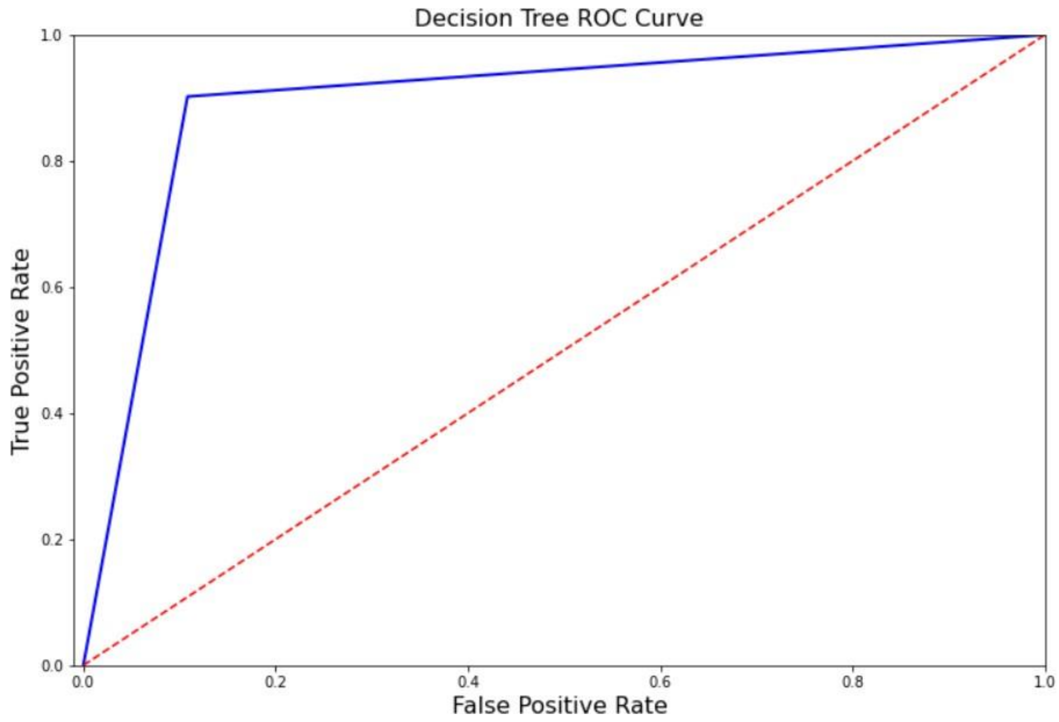
In [74]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
dt_pred = cross_val_predict(classifier, X_train, y_train, cv=5)
print('Decision Tree: ', roc_auc_score(y_train, dt_pred))
dt_fpr, dt_tpr, dt_thresold = roc_curve(y_train, dt_pred)
```

```
Decision Tree:  0.8968915343915342
```

### 4.2.5.4 Generating ROC curve:

In [75]:
```
def dt_roc_curve(dt_fpr, dt_tpr):
    plt.figure(figsize=(12,8))
    plt.title('Decision Tree ROC Curve', fontsize=16)
    plt.plot(dt_fpr, dt_tpr, 'b-', linewidth=2)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.axis([-0.01,1,0,1])


dt_roc_curve(dt_fpr, dt_tpr)
plt.show()
```

### 4.2.5.5 Curve generated and obtained:




## 4.3. Applying cross-validation on models:

### 4.3.1 Logistic Regression

```
In [155]: from sklearn.model_selection import cross_val_score
          training_score = cross_val_score(classifier, X_train, y_train, cv=5)
          print(training_score)
          print(training_score.mean())

          [0.9527027  0.93243243 0.93918919 0.93197279 0.93877551]
          0.9390145247288103
```

### 4.3.2 KNN

```
In [159]: from sklearn.model_selection import cross_val_score
          training_score = cross_val_score(classifier1, X_train, y_train, cv=5)
          print(training_score)
          print(training_score.mean())

          [0.9527027  0.92567568 0.91216216 0.93197279 0.91836735]
          0.9281761353189925
```

### 4.3.3 SVM

```
In [161]: from sklearn.model_selection import cross_val_score
          training_score = cross_val_score(classifier2, X_train, y_train, cv=5)
          print(training_score)
          print(training_score.mean())

          [0.93918919 0.93243243 0.93918919 0.93197279 0.94557823]
          0.9376723662437948
```

### 4.3.4 Naïve Bayes

```
In [164]: from sklearn.model_selection import cross_val_score
          training_score = cross_val_score(classifier, X_train, y_train, cv=5)
          print(training_score)
          print(training_score.mean())

          [0.93918919 0.93243243 0.90540541 0.89795918 0.9047619 ]
          0.9159496230924802
```

### 4.3.5 Decision Tree

```
In [168]: from sklearn.model_selection import cross_val_score
          training_score = cross_val_score(classifier, X_train, y_train, cv=5)
          print(training_score)
          print(training_score.mean())

          [0.91216216 0.89189189 0.92567568 0.89795918 0.88435374]
          0.9024085309799595
```

# 5. RANDOM OVERSAMPLING

It is the second technique used for balancing. Performing random oversampling on data to balance the dataset:

```
: X=dataset.iloc[:,:-1].values
  y=dataset.iloc[:,-1].values
  from imblearn.over_sampling import RandomOverSampler
  ros = RandomOverSampler(random_state=42)
  X_over, y_over = ros.fit_resample(X, y)
  X_train_over, X_test_over, y_train_over, y_test_over = train_test_split(X_over, y_over, test_size=0.25,random_state=0)
```

## 5.1 Applying Classifiers

### 5.1.1 Logistic Regression

#### 5.1.1.1 Generating a confusion matrix:

```
In [91]: from sklearn.linear_model import LogisticRegression
         classifier= LogisticRegression(random_state=0)
         classifier.fit(X_train_over, y_train_over)

Out[91]: LogisticRegression(random_state=0)

In [92]: y_pred_over= classifier.predict(X_test_over)
         from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test_over,y_pred_over)
         print(cm)

         [[69248  1608]
          [ 5680 65622]]
```

### 5.1.1.2 Generating cross-validation score and report:

```
In [94]: from sklearn.model_selection import cross_val_score
         training_score = cross_val_score(classifier, X_train_over, y_train_over,cv=5)
         print(training_score)
         print(training_score.mean())

[0.94983293 0.95026672 0.94970338 0.951122   0.94916407]
0.9500178204805634
```

```
In [95]: from sklearn.metrics import classification_report
         print(classification_report(y_test_over,y_pred_over))

              precision    recall  f1-score   support

           0       0.92      0.98      0.95     70856
           1       0.98      0.92      0.95     71302

    accuracy                           0.95    142158
   macro avg       0.95      0.95      0.95    142158
weighted avg       0.95      0.95      0.95    142158
```

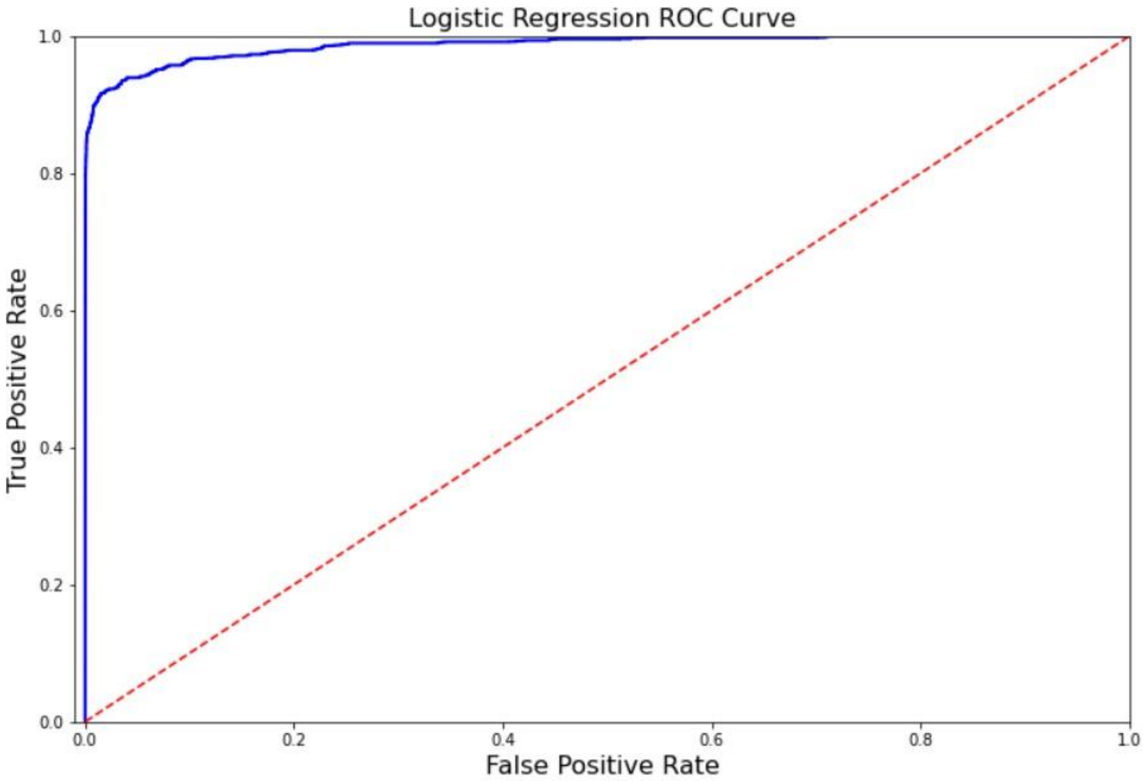
### 5.1.1.3 ROC metric evaluation:

```
In [97]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         lr_pred = cross_val_predict(classifier, X_train_over, y_train_over, cv=5,method="decision_function")
         print('Logistic Regression: ', roc_auc_score(y_train_over, lr_pred))
         lr_fpr, lr_tpr, lr_thresold = roc_curve(y_train_over, lr_pred)

Logistic Regression:  0.9872378648674063
```

### 5.1.1.4 Generating ROC Curve:

```
: def logistic_roc_curve(lr_fpr, lr_tpr):
      plt.figure(figsize=(12,8))
      plt.title('Logistic Regression ROC Curve', fontsize=16)
      plt.plot(lr_fpr, lr_tpr, 'b-', linewidth=2)
      plt.plot([0, 1], [0, 1], 'r--')
      plt.xlabel('False Positive Rate', fontsize=16)
      plt.ylabel('True Positive Rate', fontsize=16)
      plt.axis([-0.01,1,0,1])


  logistic_roc_curve(lr_fpr, lr_tpr)
  plt.show()
```

### 5.1.1.5 ROC Curve obtained:




**KNN-**

### 5.1.2 KNN

### 5.1.2.1 Generating a confusion matrix:

```
In [7]: from sklearn.neighbors import KNeighborsClassifier
        classifier1= KNeighborsClassifier(n_neighbors=8, metric='minkowski', p=2 )
        classifier1.fit(X_train_over, y_train_over)

Out[7]: KNeighborsClassifier(n_neighbors=8)

In [8]: y_pred_over= classifier1.predict(X_test_over)

In [9]: from sklearn.metrics import confusion_matrix
        cm1= confusion_matrix(y_test_over, y_pred_over)
        print(cm1)

        [[70787    69]
         [    0 71302]]
```

### 5.1.2.2 Checking for cross-validation score:

In [12]:
```
from sklearn.model_selection import cross_val_score
training_score = cross_val_score(classifier1, X_train_over, y_train_over, cv=5)
print(training_score)
print(training_score.mean())
```

```
[0.99937863 0.99947242 0.99933172 0.99946069 0.99924965]
0.999378622526681
```

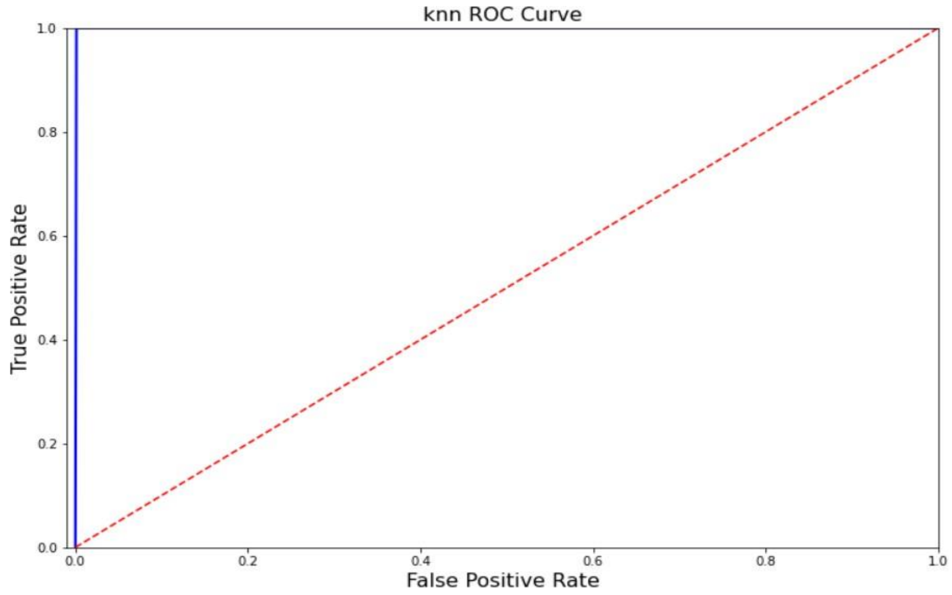
### 5.1.2.3 Checking for ROC metrics and generating roc curve: -

In [9]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
knn_pred_over = cross_val_predict(classifier1, X_train_over, y_train_over, cv=5)
print('knn: ', roc_auc_score(y_train_over, knn_pred_over))
knn_fpr, knn_tpr, knn_thresold = roc_curve(y_train_over, knn_pred_over)
```

```
knn:  0.9993792718976479
```

In [10]:
```
def knn_roc_curve(knn_fpr, knn_tpr):
    plt.figure(figsize=(12,8))
    plt.title('knn ROC Curve', fontsize=16)
    plt.plot(knn_fpr, knn_tpr, 'b-', linewidth=2)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.axis([-0.01,1,0,1])


knn_roc_curve(knn_fpr, knn_tpr)
plt.show()
```

### 5.1.2.4 ROC curve generated and obtained:



### 5.1.3 Gaussian Classifier:

#### 5.1.3.1 Generating a confusion matrix:

```
In [5]: from sklearn.naive_bayes import GaussianNB
        classifier = GaussianNB()
        classifier.fit(X_train_over,y_train_over)
        y_pred_over = classifier.predict(X_test_over)
```

```
In [6]: from sklearn.metrics import confusion_matrix
        cm = confusion_matrix(y_test_over, y_pred_over)
        cm

Out[6]: array([[68978,  1878],
               [ 9995, 61307]], dtype=int64)
```

#### 5.1.3.2 Checking for validation score:

```
In [7]: from sklearn.model_selection import cross_val_score
        training_score = cross_val_score(classifier, X_train_over, y_train_over, cv=5)
        print(training_score)
        print(training_score.mean())

        [0.91813119 0.916865   0.9168054  0.91732127 0.91678195]
        0.9171809623275033
```

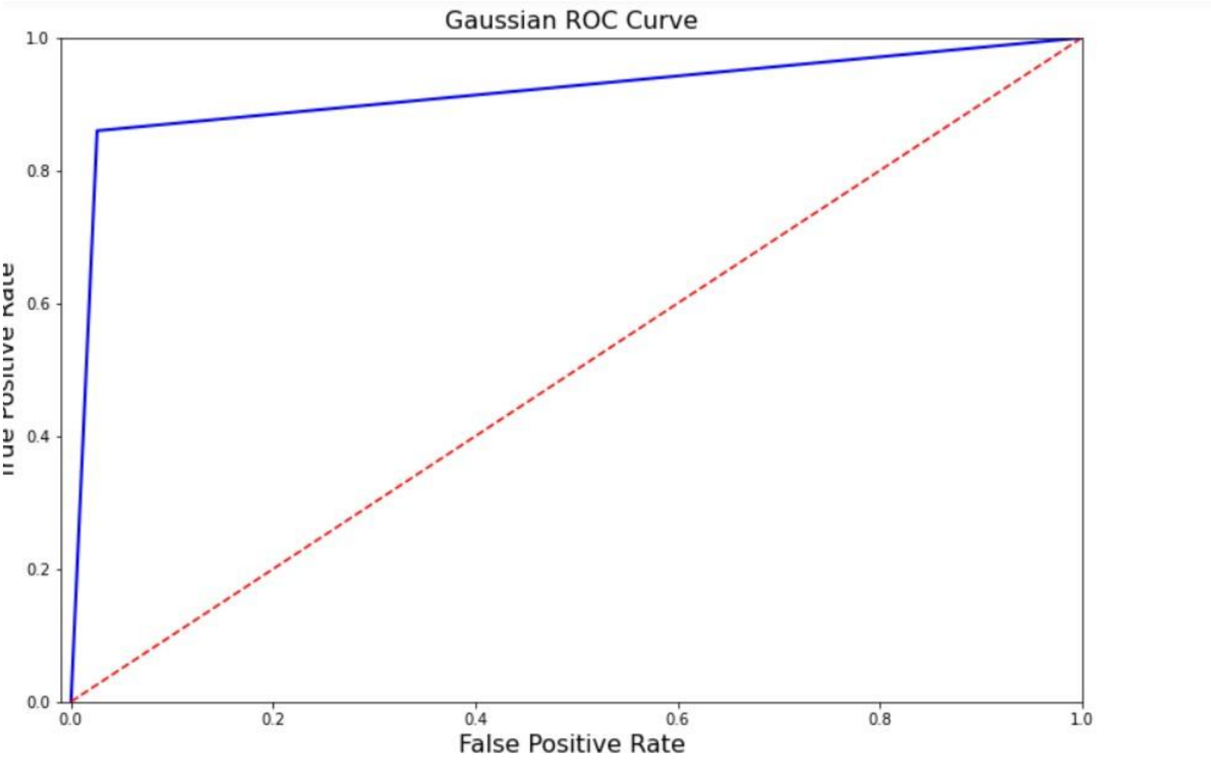
#### 5.1.3.3 Applying ROC metrics:

```
In [9]: from sklearn.metrics import roc_auc_score
        from sklearn.metrics import roc_curve
        from sklearn.model_selection import cross_val_predict
        gauss_pred = cross_val_predict(classifier, X_train_over, y_train_over, cv=5)
        print('Gaussian: ', roc_auc_score(y_train_over, gauss_pred))
        gauss_fpr, gauss_tpr, gauss_thresold = roc_curve(y_train_over, gauss_pred)

        Gaussian:  0.9171212899013806
```

#### 5.1.3.4 Generating ROC Curve:

```
In [10]: def gauss_roc_curve(gauss_fpr, gauss_tpr):
             plt.figure(figsize=(12,8))
             plt.title('Gaussian ROC Curve', fontsize=16)
             plt.plot(gauss_fpr, gauss_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         gauss_roc_curve(gauss_fpr, gauss_tpr)
         plt.show()
```

### 5.1.3.5 Plot obtained:




## 5.1.4 Decision Tree Classifier

### 5.1.4.1 Generating a confusion matrix:

```
In [11]: from sklearn.tree import DecisionTreeClassifier
         classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
         classifier.fit(X_train_over, y_train_over)

Out[11]: DecisionTreeClassifier(criterion='entropy', random_state=0)

In [12]: y_pred_over= classifier.predict(X_test_over)

In [13]: from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test_over, y_pred_over)
         cm

Out[13]: array([[70827,    29],
                [    0, 71302]], dtype=int64)
```

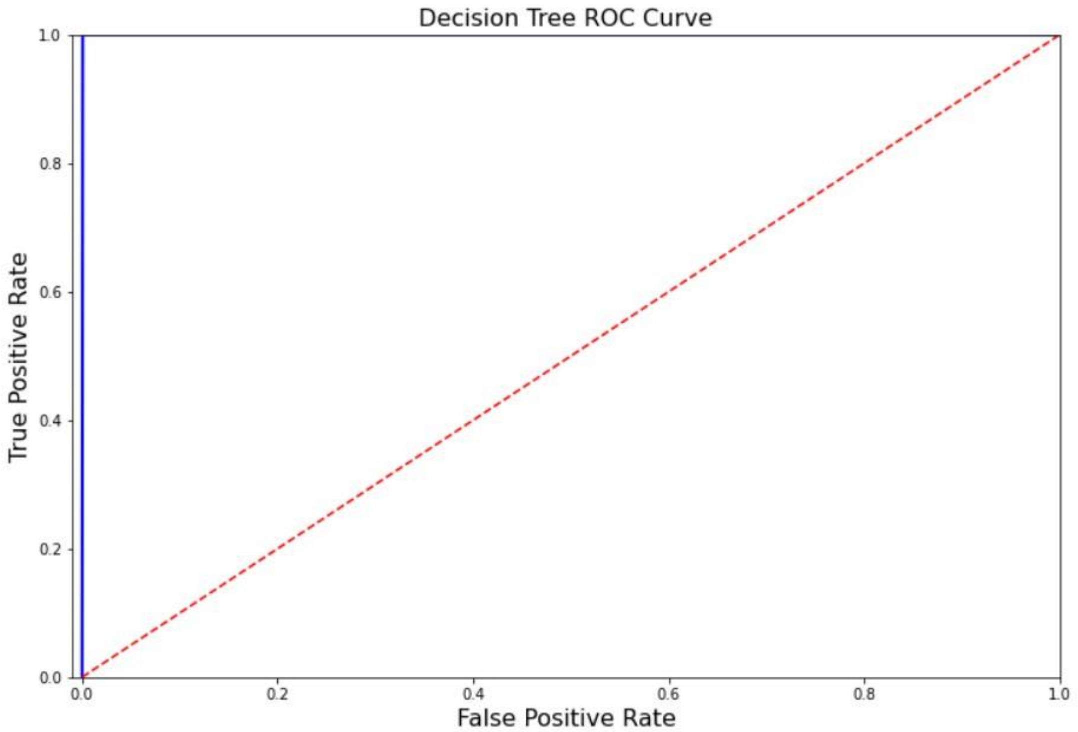
## 5.1.4.2 Applying ROC metrics:

```
In [14]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         dt_pred = cross_val_predict(classifier, X_train_over, y_train_over, cv=5)
         print('Decision Tree: ', roc_auc_score(y_train_over, dt_pred))
         dt_fpr, dt_tpr, dt_thresold = roc_curve(y_train_over, dt_pred)

         Decision Tree:  0.9996580139511568
```

## 5.1.4.3 Generating ROC curve:

```
In [15]: def dt_roc_curve(dt_fpr, dt_tpr):
             plt.figure(figsize=(12,8))
             plt.title('Decision Tree ROC Curve', fontsize=16)
             plt.plot(dt_fpr, dt_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         dt_roc_curve(dt_fpr, dt_tpr)
         plt.show()
```

## 5.1.4.4 The plot obtained:



# 6. SMOTE TECHNIQUE OF BALANCING

Smote is an **oversampling technique** where synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling.

## 6.1 Applying Classification Models:

### 6.1.1 Logistic Regression

#### 6.1.1.1 Generating a confusion matrix:

```
In [10]: from sklearn.linear_model import LogisticRegression
         classifier= LogisticRegression(random_state=0)
         classifier.fit(X_train_smote, y_train_smote)

Out[10]: LogisticRegression(random_state=0)

In [11]: y_pred_smote= classifier.predict(X_test_smote)
         from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test_smote,y_pred_smote)
         print(cm)

         [[69111  1745]
          [ 5682 65620]]
```

#### 6.1.1.2 Cross-validation Score and Classification Report:

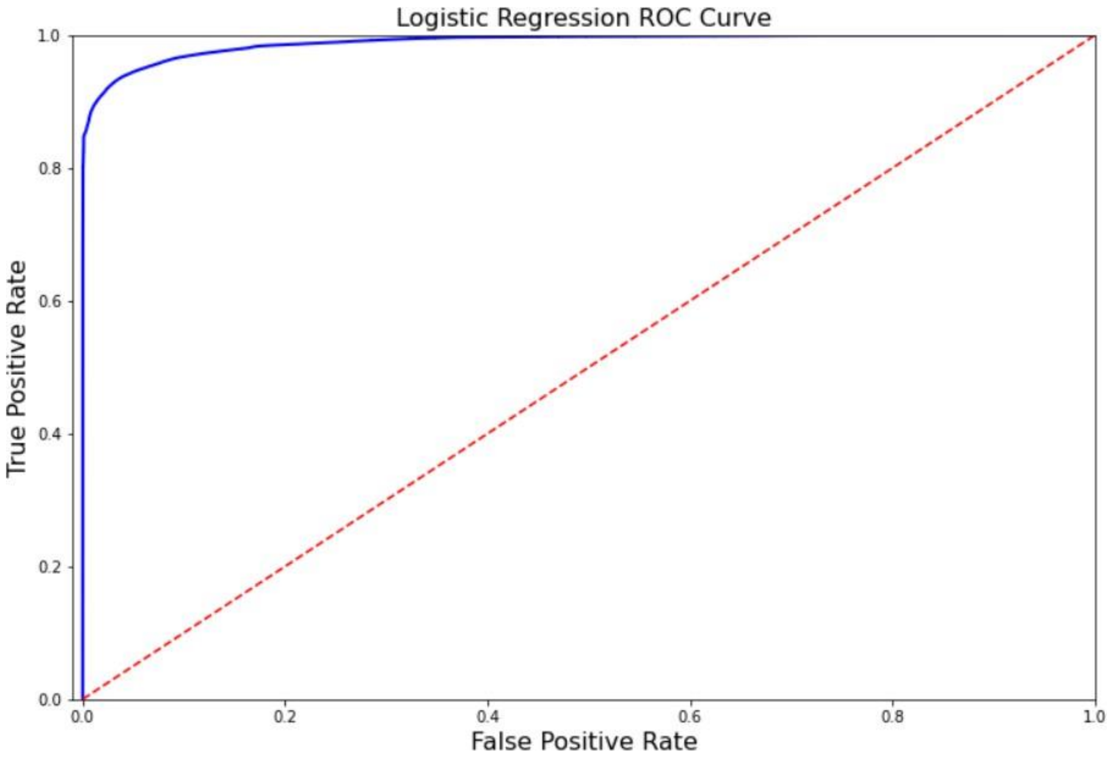
```
In [12]: from sklearn.model_selection import cross_val_score
         training_score = cross_val_score(classifier, X_train_smote, y_train_smote,cv=5)
         print(training_score)
         print(training_score.mean())

         [0.94893018 0.94739434 0.94794476 0.94732338 0.94739372]
         0.9477972745456402

In [13]: from sklearn.metrics import classification_report
         print(classification_report(y_test_smote,y_pred_smote))

                       precision    recall  f1-score   support

                    0       0.92      0.98      0.95     70856
                    1       0.97      0.92      0.95     71302

             accuracy                           0.95    142158
            macro avg       0.95      0.95      0.95    142158
         weighted avg       0.95      0.95      0.95    142158
```

## 6.1.1.3 ROC metrics and ROC curve generation:

```
In [14]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         lr_pred = cross_val_predict(classifier, X_train_smote, y_train_smote, cv=5,method="decision_function")
         print('Logistic Regression: ', roc_auc_score(y_train_smote, lr_pred))
         lr_fpr, lr_tpr, lr_thresold = roc_curve(y_train_smote, lr_pred)

         Logistic Regression:  0.9893171523840611
```

```
In [15]: def logistic_roc_curve(lr_fpr, lr_tpr):
             plt.figure(figsize=(12,8))
             plt.title('Logistic Regression ROC Curve', fontsize=16)
             plt.plot(lr_fpr, lr_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         logistic_roc_curve(lr_fpr, lr_tpr)
         plt.show()
```

## 6.1.1.4 The plot generated and obtained:



### 6.1.2 KNN

#### 6.1.2.1 Generating a confusion matrix:

```
In [6]: from sklearn.neighbors import KNeighborsClassifier
        classifier1= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
        classifier1.fit(X_train_smote, y_train_smote)

Out[6]: KNeighborsClassifier()
```

```
In [8]: y_pred_smote= classifier1.predict(X_test_smote)
        from sklearn.metrics import confusion_matrix
        cm1= confusion_matrix(y_test_smote, y_pred_smote)
        print(cm1)

        [[70720   136]
         [    0 71302]]
```

#### 6.1.2.2 Generating Cross-Validation Score:

```
In [9]: from sklearn.model_selection import cross_val_score
        training_score = cross_val_score(classifier1, X_train_smote, y_train_smote, cv=5)
        print(training_score)
        print(training_score.mean())

        [0.99872208 0.99876898 0.99879241 0.99894483 0.99858138]
        0.9987619352098379
```

#### 6.1.2.3 Generating ROC matrices and ROC Curve:

```
In [10]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.model_selection import cross_val_predict
         knn_pred_smote = cross_val_predict(classifier1, X_train_smote, y_train_smote, cv=5)
         print('knn: ', roc_auc_score(y_train_smote, knn_pred_smote))
         knn_fpr, knn_tpr, knn_thresold = roc_curve(y_train_smote, knn_pred_smote)

         knn:  0.9987632285356908
```

```
In [11]: def knn_roc_curve(knn_fpr, knn_tpr):
             plt.figure(figsize=(12,8))
             plt.title('knn ROC Curve', fontsize=16)
             plt.plot(knn_fpr, knn_tpr, 'b-', linewidth=2)
             plt.plot([0, 1], [0, 1], 'r--')
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)
             plt.axis([-0.01,1,0,1])


         knn_roc_curve(knn_fpr, knn_tpr)
         plt.show()
```

### 6.1.2.4 The plot obtained: