Data Science PROJECT

Client: Sales Effectiveness | Category: Product Sales

Project Ref: PM-PR-0019

Business Case:

FicZon Inc is an IT solution provider with products ranging from onpremises products to SAAS based solutions. FicZon major leads generation channel is digital and through their website. FicZon business is majorly dependent on the sales force effectiveness. As the market is maturing and more new competitors entering the market, FicZon is experiencing the dip in sales. An effective sale is dependent on lead quality and as of now, this is based on manual categorization and highly depended on sales staff. Though there is a quality process, which continuously updates the lead categorization, its value is in for post analysis, rather than conversation.
FicZon wants to explore Machine Learning to pre-categorize the lead quality and as result, expecting significant increase in sales effectiveness.

PROJECT GOAL:

1. Data exploration insights – Sales effectiveness.

2. ML model to predict the Lead Category (High Potential, Low Potential)

Database details:

DB Name: project_sales
Table Name: data
Host: 18.136.56.185
Port: 3306
Username: dm_team2
Password: dm_team1118#

Feature Details:

RangeIndex: 7422 entries, 0 to 7421
Data columns (total 9 columns):
Created 7422 non-null object
Product_ID 7364 non-null float64
Source 7405 non-null object
Mobile 5612 non-null object
EMAIL 7422 non-null object
Sales_Agent 7399 non-null object
Location 7364 non-null object
Delivery_Mode 7422 non-null object
Status 7422 non-null object

Created: It consists of lead creation date.

Product_ID: Product ID is a methodology which helps in identifying a product without a full specification specified on the label. In case of shipping and transportation of the items each and every document associated with the product carries this unique Product ID. It helps in tracking the item in any part of the supply chain.

Source: Product sourcing is the process by which a business attains a product to sell. It can be call, email message CRM form, email campaign, customer referral, campaign, Live chat, SMS campaign, website, personal chat, by recommendation, existing customer etc.

Mobile: Contact number of the sales person.

Email: Email address of the sales person.

Sales_Agent: A person or a company that acts as a sales agent on behalf of the exporting company (principal), introducing its products to potential buyers in the external market, in exchange for a commission based on the value of the business deals arranged and paid to the principal.

https://www.globalnegotiator.com/international-trade/dictionary/sales-agent/

Location: The place where the sales activity takes place.

Delivery_Mode: A delivery mode is the way training instructions are delivered to support and enable learning process.

Status: It is title you put on a lead, or groups of leads, in order to plan actions and to improve work-flow. It can be open, potential, converted, Not responding, Junk lead, just inquiry, lost, long term, in progress positive, in progress negative.

Features

The features of the datasets were provided by ***Datamites*** Company.

Assumptions:

- Dropped features like Created, Mobile and Email.
- Used Product_ID, Source, Sales_Agent, Location and Delivery_Mode as input variables.
- Used Status as target variable.
- Scaled the data using standard scaler.
- Used SMOTE (Syntactically Minority Oversampling TEchnique) for handling the imbalanced datasets.
- Used Principal Component Analysis (PCA) for dimensionality reduction.
- Used GridSearch Cross-Validation in Random Forest Classifier as part of hyperparameter tuning to combine an estimator using grid search.
- Used Randomized Search Cross-Validation in Random Forest Classifier as part of hyperparameter tuning by finding the random combinations of the hyperparameters to find the best solution for the built model.

Import the dataset from the server

```
!pip install sqlalchemy

!pip install pymysql

from sqlalchemy import create_engine

import pandas as pd

db_host= '18.136.56.185:3306'

username = 'dm_team2'

user_pass= 'dm_team1118#'

db_name='project_sales'

conn=
create_engine('mysql+pymysql://dm_team2:dm_team1118#@18.136.56.185:3306/pr
oject_sales')

conn.table_names()

query = 'select * from data'

data = pd.read_sql(query,conn)

print(data.shape)
```

data

Download the dataset in xls format

```
data.to_excel('C:\\Users\DELL\Desktop\Datamites
projects\Apr2020\Sales_data.xlsx')
```

Approach:

1) Data exploration insights – Sales effectiveness

1. Import the necessary packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams
%matplotlib inline
from collections import Counter
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder,scale
from sklearn.metrics import
accuracy_score,precision_score,confusion_matrix,classification_report,f1_score,reca
ll_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

2. Load the dataset.

```
data=
pd.read_excel("C:\\Users\DELL\Desktop\Datamitesprojects\Apr2020\Sales_data.xls
x",parse_dates=['Created'])
data
```

2. Creating a new dataframe from the existing dataframe.

```
data=pd.DataFrame(data,columns=['Created','Product_ID','Source','Mobile','EMAIL',
'Sales_Agent','Location','Delivery_Mode','Status'])
data
```

3. Perform Exploratory Data Analysis (EDA) steps.

```
data.describe()
data.info()
data.shape
data.isna().sum().to_frame().T
data.isna().sum().to_frame().any
data.dtypes
data.columns
```

4. Data Exploration Insights

In this section we find the insights with respect to different fields in the data like Location, Product_ID, Delivery_Mode, Sales_Agent, Source and Status.

1. data.groupby(by=['Product_ID'])['Sales_Agent'].count()

Here count of Sales Agent with respect to product_ID is calculated.

```
Product_ID
0.0       2
1.0     104
2.0      38
3.0       7
4.0       1
5.0     485
6.0       7
7.0       1
8.0       6
9.0     992
10.0    168
11.0     12
12.0     36
13.0      4
14.0     27
15.0   1507
16.0      3
17.0      6
18.0   1709
19.0   1188
20.0    102
21.0     65
22.0      8
23.0      2
24.0      2
25.0     90
26.0     31
27.0    737
28.0      1
Name: Sales_Agent, dtype: int64
```

```
new=data.groupby(by=['Location'])['Delivery_Mode'].count()
new
```

Here Count of Deliver_Mode is calculated with respect to location.

```
Location
AUSTRALIA       25
Bangalore     2084
Chennai        909
Delhi          471
```

```
EUROPE            3
Howrah            1
Hyderabad       528
Kolkata          55
Malaysia          4
Mumbai          402
Other Locations  2500
Pune            142
Singapore        17
Trivandrum       58
UAE              79
UK               41
USA              45
Name: Delivery_Mode, dtype: int64
```

```
plt.figure(figsize=(15,15))
plt.plot(new)
plt.xlabel('Location',fontsize=15)
plt.ylabel('Delivery_Mode',fontsize=15)
```

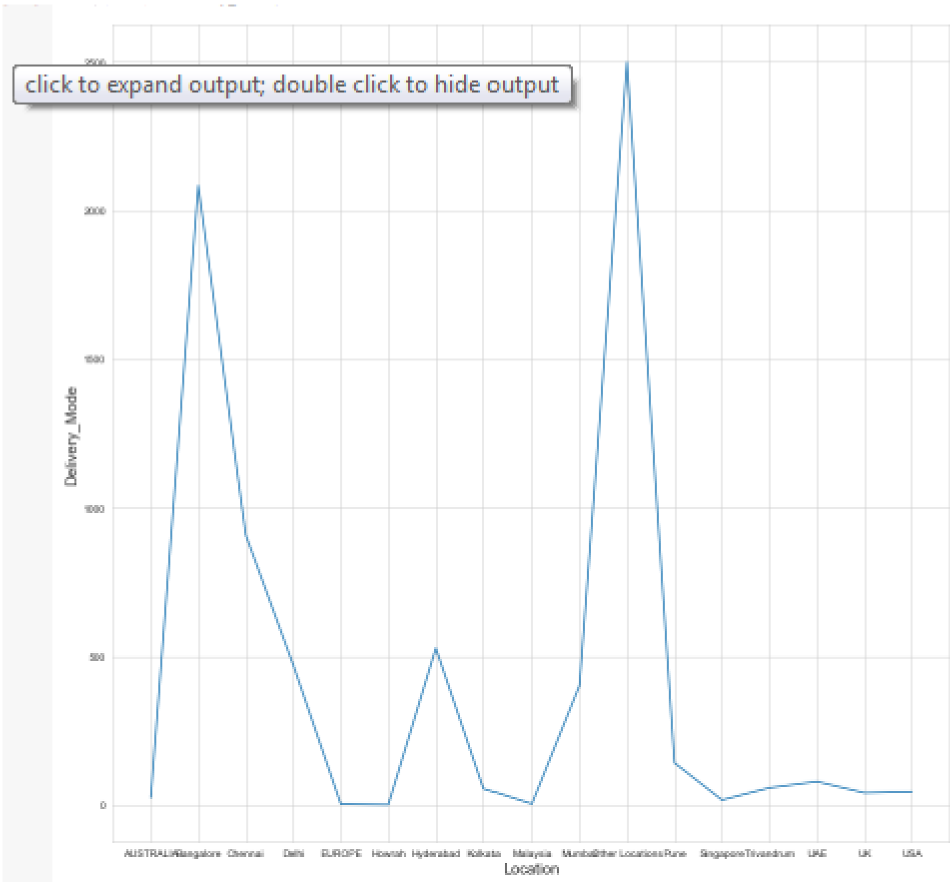

Figure 1: Line graph of Location v/s Delivery Mode

```
data.groupby(by=['Sales_Agent'])['Product_ID'].mean()
```

Here mean of Product_ID with respect to Sales_Agent is calculated.

```
Sales_Agent
Sales-Agent-1         NaN
Sales-Agent-10    14.200000
Sales-Agent-11    15.526464
Sales-Agent-12    16.460967
Sales-Agent-2     16.899743
Sales-Agent-3     15.994878
Sales-Agent-4     15.797730
Sales-Agent-5     16.970489
Sales-Agent-6     11.438596
Sales-Agent-7     14.970109
Sales-Agent-8     15.294118
Sales-Agent-9     16.350797
Name: Product_ID, dtype: float64
```

```
plt.figure(figsize=(15,5))
splot=sns.barplot(data['Sales_Agent'],data['Product_ID'],ci=None)
plt.xticks(rotation=30)
plt.xlabel("Sales Agent ",fontsize=15,color='black')
plt.ylabel(" Product_ID ",fontsize=15,color='black')
plt.title(" Sales Agent with respect to Product_ID ",fontdict={'fontsize':20,'color':'Red'})
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

Figure 2: Bar chart for Sales Agent v/s Product_ID

```
delivery=data.groupby(by=['Product_ID'])['Delivery_Mode'].count()
delivery
```

Here count of Delivery_Mode is calculated with respect to Product_ID.

```
Product_ID
0.0      2
1.0    105
```

```
2.0      38
3.0       7
4.0       1
5.0     487
6.0       7
7.0       1
8.0       6
9.0     992
10.0    168
11.0     12
12.0     36
13.0      5
14.0     27
15.0   1518
16.0      3
17.0      7
18.0   1711
19.0   1189
20.0    102
21.0     66
22.0      8
23.0      2
24.0      3
25.0     90
26.0     31
27.0    739
28.0      1
Name: Delivery_Mode, dtype: int64
```

```
plt.figure(figsize=(5,5))
plt.plot(delivery,color='red')
plt.xlabel('Product_ID',fontsize=15)
plt.ylabel('Delivery_Mode',fontsize=15)
```

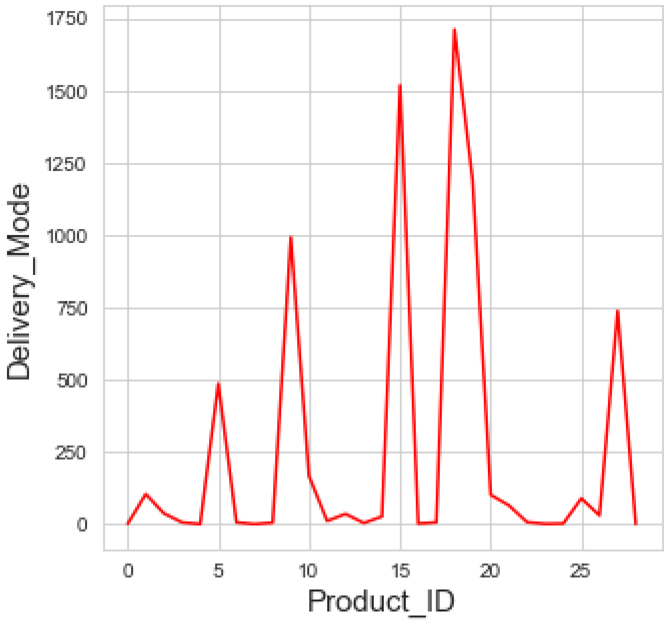

Figure 3: Line graph for Product_ID v/s Delivery Mode

```
source=data.groupby(by=['Product_ID'])['Source'].count()
source
```

Here count of Source with respect to Product_ID is done.

```
Product_ID
0.0      2
1.0    105
2.0     38
3.0      7
4.0      1
5.0    486
6.0      7
7.0      1
8.0      6
9.0    990
10.0    168
11.0     12
12.0     36
13.0      5
14.0     27
15.0   1516
16.0      3
17.0      7
18.0   1709
19.0   1189
20.0    102
21.0     66
22.0      8
23.0      2
```

```
24.0      3
25.0     90
26.0     31
27.0    739
28.0      1
Name: Source, dtype: int64
plt.figure(figsize=(5,5))
plt.plot(delivery,color='magenta')
plt.xlabel('Product_ID',fontsize=15)
plt.ylabel('Source',fontsize=15)
```

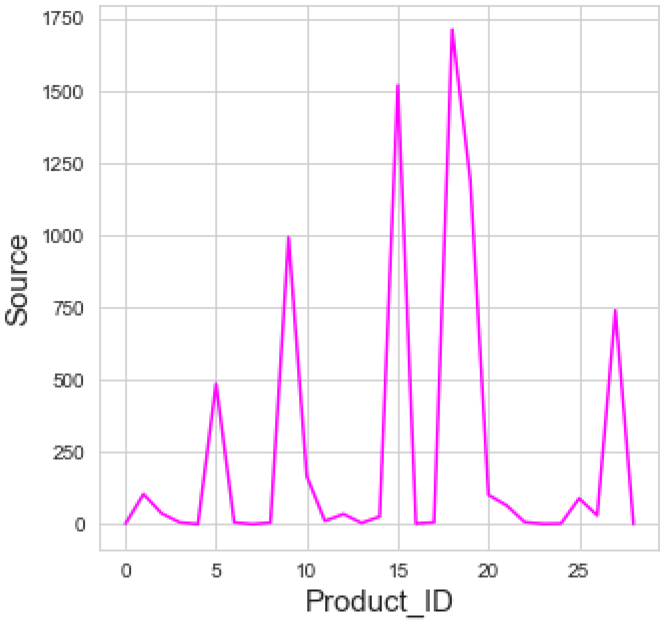

Figure 4: Line graph for Product_ID v/s Source

```
location=data.groupby(by=['Location'])['Product_ID'].count()

location
```

Here count of product_ID is done with respect to location.

```
Location
AUSTRALIA          25
Bangalore        2084
Chennai           909
Delhi             471
EUROPE              3
Howrah              1
Hyderabad         528
Kolkata            55
Malaysia            4
Mumbai            401
Other Locations  2496
Pune              142
Singapore          17
```

```
Trivandrum          58
UAE                 78
UK                  41
USA                 45
Name: Product_ID, dtype: int64
```

```
location_status=data.groupby(by=['Location'])['Status'].count()
location_status
```

Count of status with respect to Location is done.

```
Location
AUSTRALIA           25
Bangalore         2084
Chennai            909
Delhi              471
EUROPE               3
Howrah               1
Hyderabad          528
Kolkata             55
Malaysia             4
Mumbai             402
Other Locations   2500
Pune               142
Singapore           17
Trivandrum          58
UAE                 79
UK                  41
USA                 45
Name: Status, dtype: int64
```

```
plt.figure(figsize=(15,10))
plt.plot(location_status,color='magenta')
plt.xlabel('Location',fontsize=15)
plt.ylabel('Status',fontsize=15)
plt.title("Count of status based on location",color='black',fontsize=15)
```

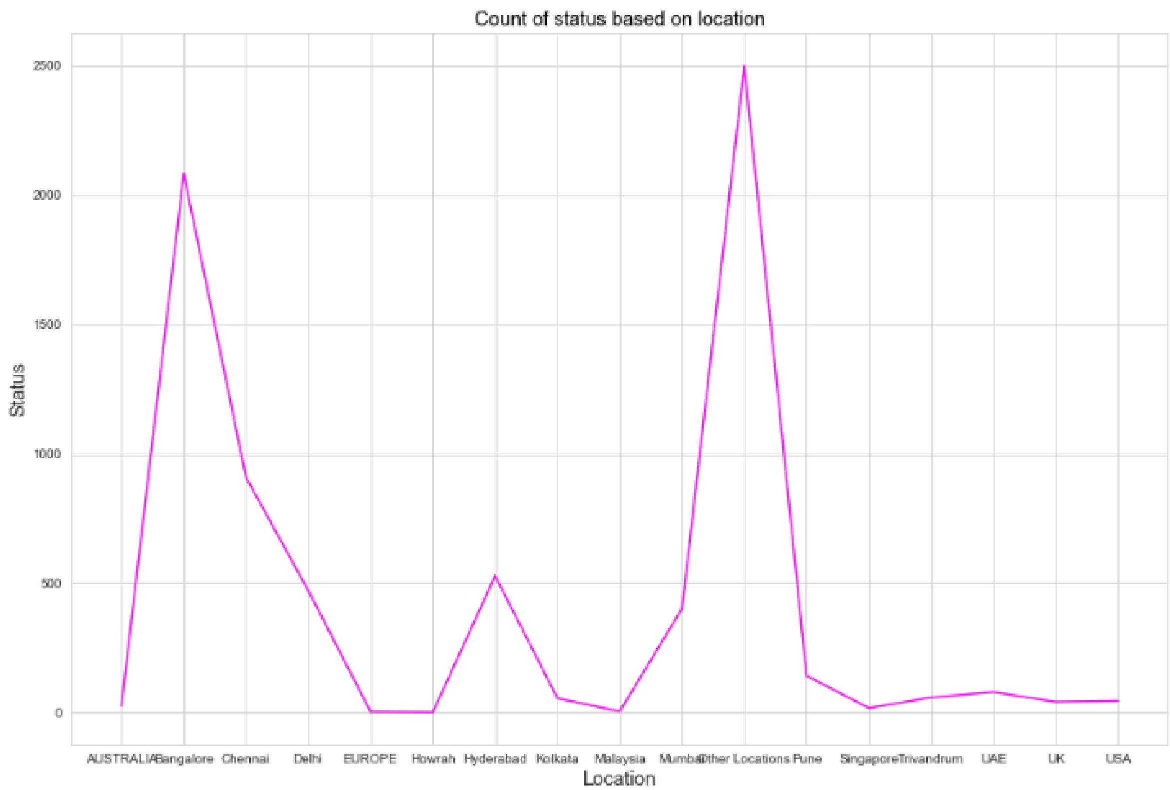

Figure 5: Line graph for Location v/s Status

```
plt.figure(figsize=(15,10))
Product_ID=data.Product_ID.value_counts()
plt.xlabel('Product_ID',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Product_ID.plot(kind='bar')
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

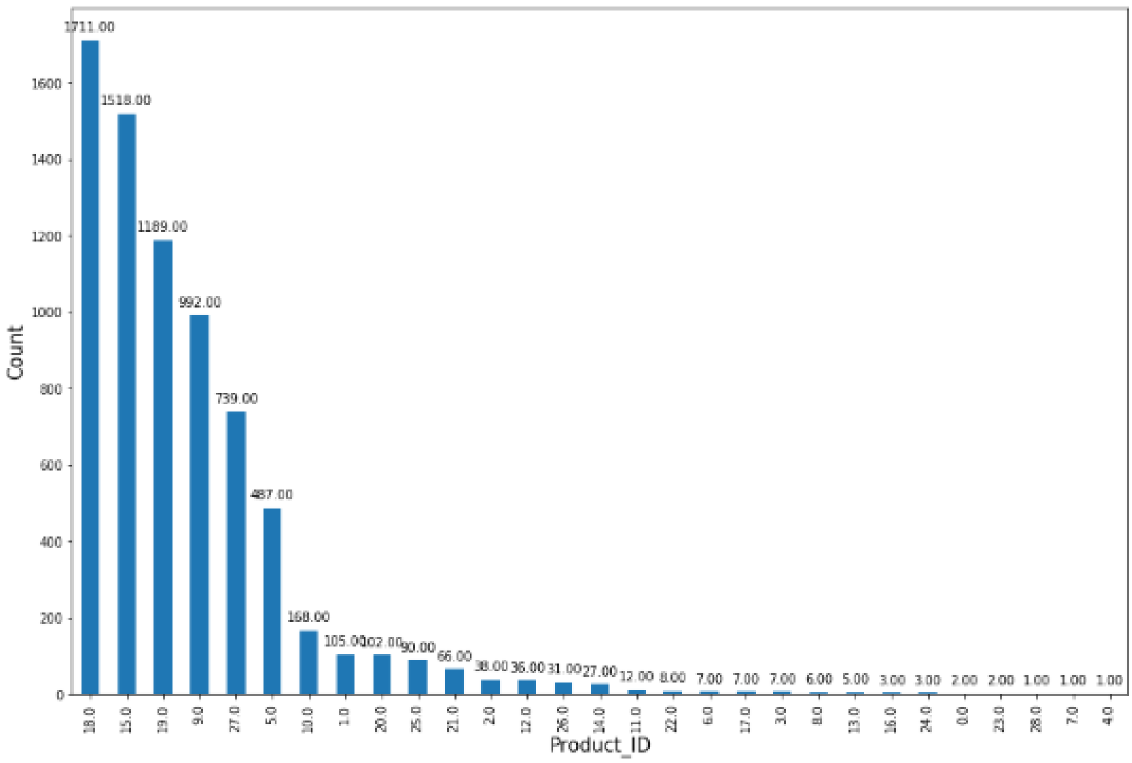

Figure 6: Bar graph for Product_ID v/s Count

```
plt.figure(figsize=(15,10))
Source=data.Source.value_counts()
plt.xlabel('Source',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Source.plot(kind='bar',color='green')
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

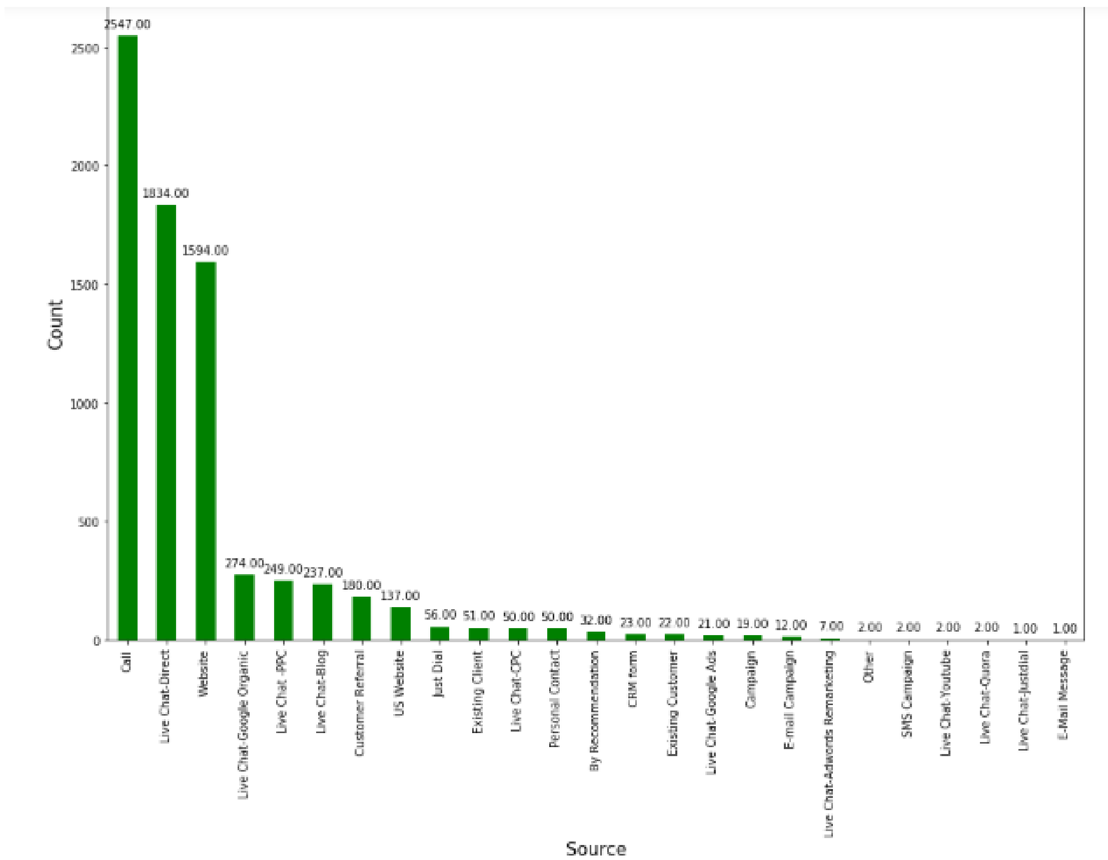

Figure 7: Bar graph for Source count

```
plt.figure(figsize=(15,10))
Sales_Agent=data.Sales_Agent.value_counts()
plt.xlabel('Sales_Agent',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Sales_Agent.plot(kind='bar',color='red')
for p in splot.patches:
   splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

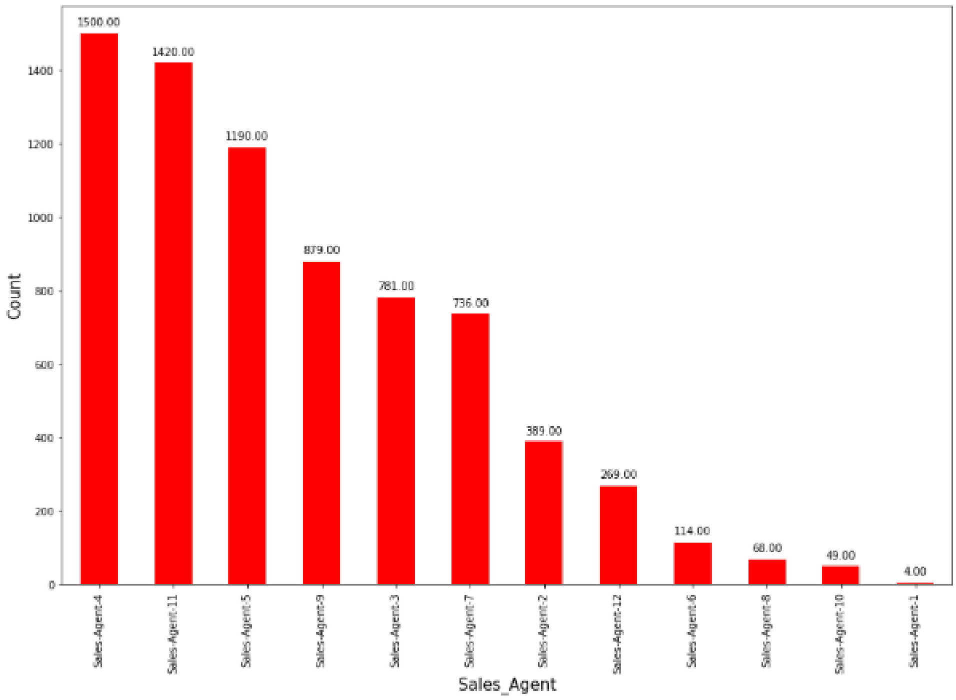

Figure 8: Bar chart for count of sales agent

```
plt.figure(figsize=(15,10))
Location=data.Location.value_counts()
plt.xlabel('Location',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Location.plot(kind='bar',color='pink')

for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

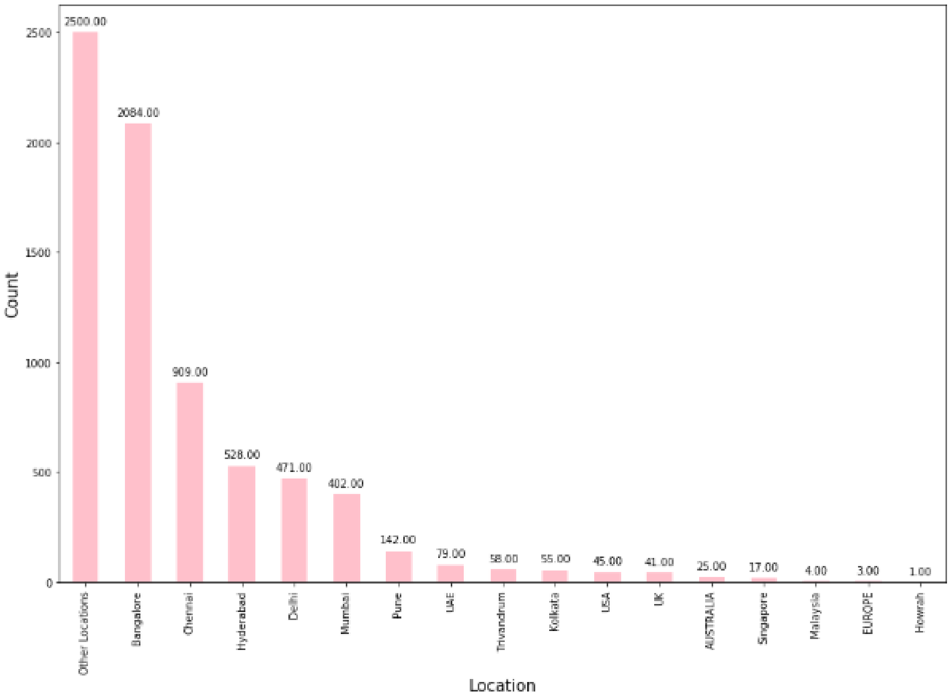

Figure 9: Bar graph for count of sales based on location

```
plt.figure(figsize=(15,10))
Delivery_Mode=data.Delivery_Mode.value_counts()
plt.xlabel('Delivery_Mode',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Delivery_Mode.plot(kind='bar',color='orange')

for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

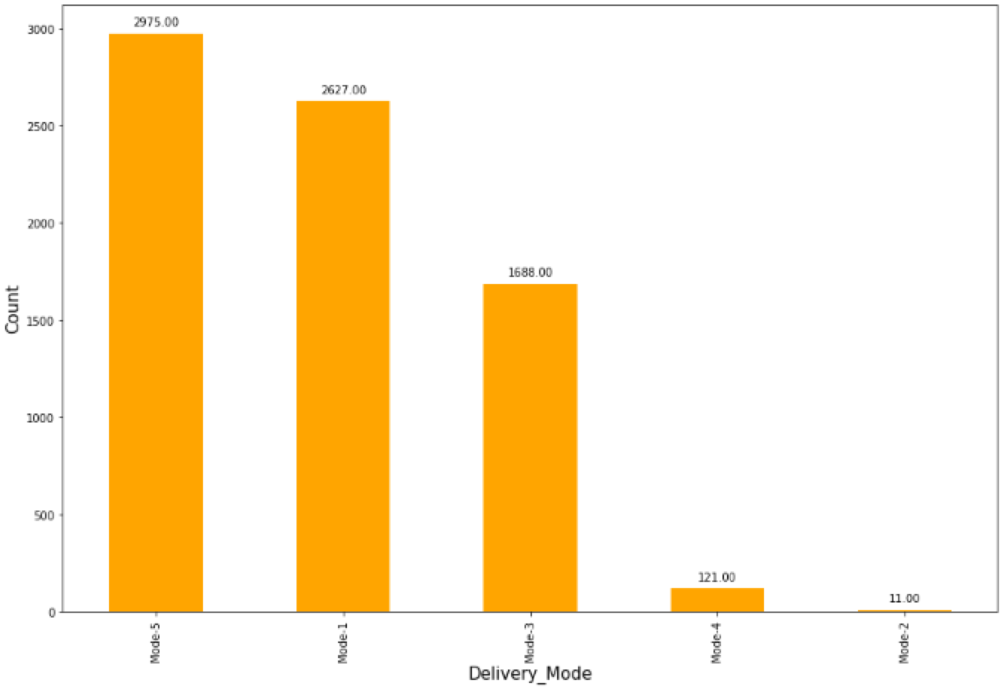

Figure 10: Bar graph of count of delivery mode

```
plt.figure(figsize=(15,10))
Status=data.Status.value_counts()
plt.xlabel('Status',fontsize=15)
plt.ylabel('Count',fontsize=15)
splot=Status.plot(kind='bar',color='Gray')

for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10),textcoords = 'offset points')
```

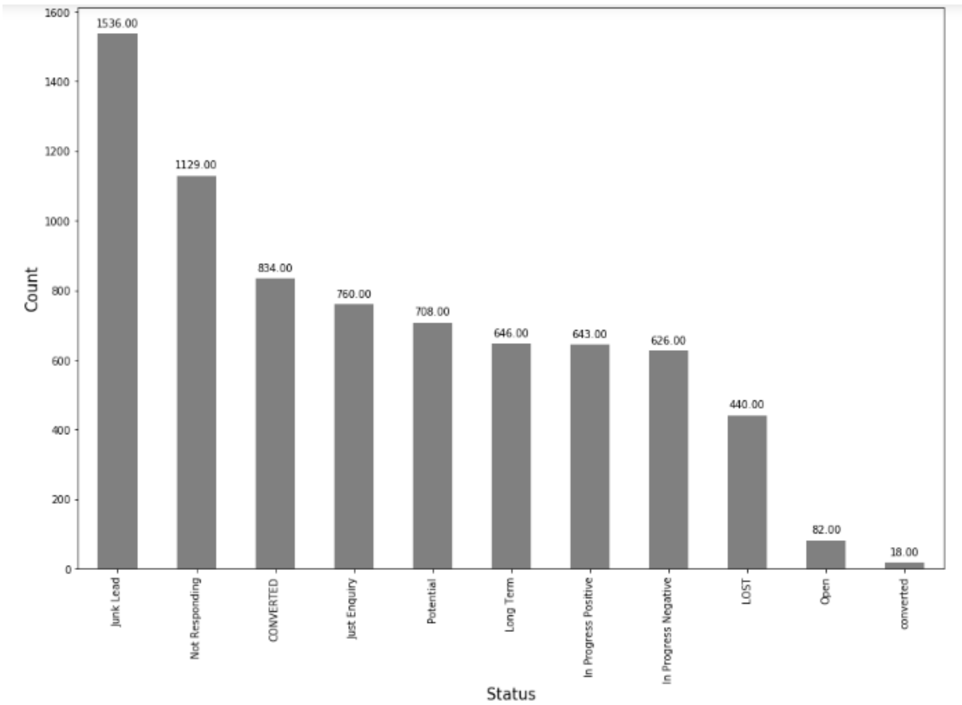

Figure 11: Bar graph for count of status

5. Cleaning the data.
Drop the following columns
data=data.drop(columns=['Mobile','EMAIL','Created'])

data.shape
data.head()

Replace nan values with ' ' character and drop null values. Perform the reset index.
data.replace('',np.nan,inplace=True)
data.dropna(axis=0,inplace=True)
data.reset_index(inplace=True,drop=True)

Check for the null values.
data.isna().sum().to_frame().T

Converting categorical data into dummy/indicator variables.
pd.get_dummies(data.Status,drop_first=True)
pd.get_dummies(data.Delivery_Mode,drop_first=True)
pd.get_dummies(data.Location,drop_first=True)
pd.get_dummies(data.Sales_Agent,drop_first=True)
pd.get_dummies(data.Source,drop_first=True)
pd.get_dummies(data.Delivery_Mode,drop_first=True)

Convert the Status field values into High Potential and Low Potential
data.Status.replace(['CONVERTED','converted','In Progress Positive','Potential','Long Term','Open'],'High Potential',inplace=True)

data.Status.replace(['LOST','In Progress Negative','Not Responding','Junk Lead','Just Enquiry'],'Low Potential',inplace=True)

Using Counter to count key value pairs inside a dictionary for each field.
Counter(data.Location)
Counter(data.Source)
Counter(data.Sales_Agent)
Counter(data.Status)
Counter(data.Product_ID)
Counter(data.Delivery_Mode)

6. Using Label Encoder
It is used to convert the categorical data into numerical.
from sklearn.preprocessing import LabelEncoder

enc=LabelEncoder()
data.Source=enc.fit_transform(data.Source)
data.Sales_Agent=enc.fit_transform(data.Sales_Agent)
data.Location=enc.fit_transform(data.Location)
data.Delivery_Mode=enc.fit_transform(data.Delivery_Mode)
data.Status=enc.fit_transform(data.Status)

7. Checking for the outliers

We use heatmap plot to check if any outliers are present in the data.
sns.set_style('whitegrid')
sns.heatmap(data.isnull(),yticklabels=False,cbar=True,cmap='viridis')

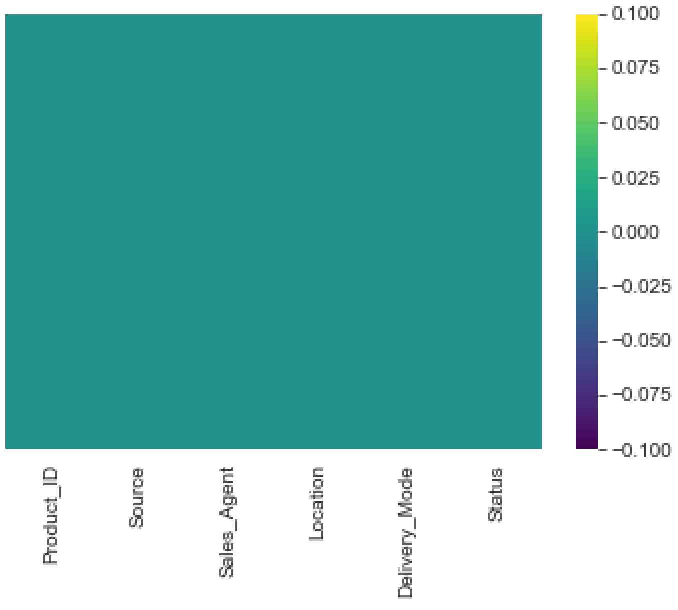

Figure 12: Heatmap showing data fields

8. Correlation Matrix
It is used to find the correlation of each field with respect to one another.

```
corr=data.corr()

corr
```

	Product_ID	Source	Sales_Agent	Location	Delivery_Mode	Status
Product_ID	1.000000	0.060910	0.006529	-0.041227	-0.036025	-0.085415
Source	0.060910	1.000000	-0.019623	0.054901	-0.151242	0.039717
Sales_Agent	0.006529	-0.019623	1.000000	-0.129056	-0.224688	-0.137074
Location	-0.041227	0.054901	-0.129056	1.000000	0.397186	0.312023
Delivery_Mode	-0.036025	-0.151242	-0.224688	0.397186	1.000000	0.220445
Status	-0.085415	0.039717	-0.137074	0.312023	0.220445	1.000000

Table 1: Table representing correlation between different data fields

```
plt.figure(figsize=(10,10))
sns.heatmap(corr,cmap='viridis',        vmax=.3,vmin=.03        ,center=0,square=True,
linewidths=.2, cbar_kws={"shrink": .2}, annot=True)
```

Figure 13: Correlation Matrix

2) <u>ML model to predict the Lead Category (High Potential , Low Potential)</u>

1. <u>Define X and y variables</u>

```
X=data[['Product_ID','Source','Sales_Agent','Location','Delivery_Mode']]
y=data.Status
```

2. <u>Using train-test split</u>

```
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=5)
```

3. <u>Using Standard Scaler</u>

```
from sklearn.preprocessing import StandardScaler

s = StandardScaler()
X_train = s.fit_transform(X_train)
X_test = s.transform(X_test)
```

4. <u>Using SMOTE Technique</u>

```
from imblearn.over_sampling import SMOTE

smote=SMOTE()
X_train, y_train = smote.fit_sample(X_train,y_train)
```

5. <u>Using PCA Technique</u>

```
from sklearn.decomposition import PCA

pca=PCA()
X=pd.DataFrame(pca.fit_transform(X))

pca.explained_variance_ratio_
```

Output: array([0.52549458, 0.26459349, 0.13698127, 0.05666528, 0.01626539])

```
pca.explained_variance_
```

Output: array([72.94237527, 36.72745392, 19.01397138, 7.86554237, 2.25775114])

```
sales_var=pd.DataFrame(pca.explained_variance_ratio_)
sales_var.plot(kind='bar')
```

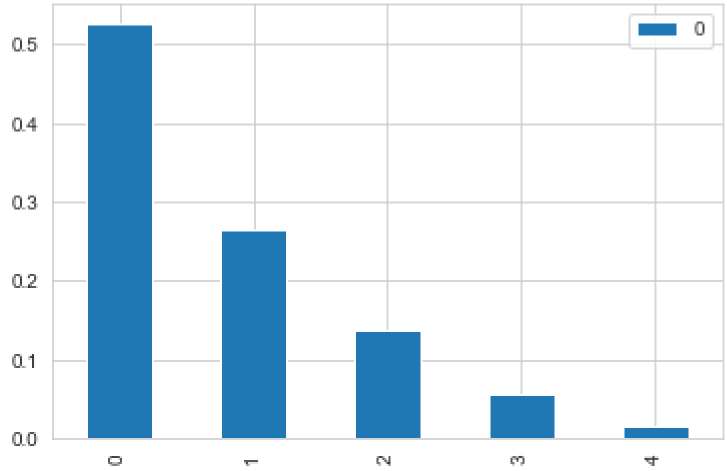

Figure 14: Bar chart representing data after performing PCA

Next steps are train and predict the model.

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25, random_state=10)

model=RandomForestClassifier(random_state=10,n_estimators=100,max_depth=20, criterion='gini')
model.fit(X_train,y_train)

y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)

print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[420 313]
 [270 829]]
```

col_0	0	1
Status		
0	420	313
1	270	829

We get the following performance metrics:

Accuracy of Training = 86.35371179039302
Accuracy of Testing = 68.17685589519651
Precision score = 67.90171272606725

Recall score = 68.17685589519651
F1 score = 68.0013514614794

Classification report using PCA is as follows:

	precision	recall	f1-score	support
0	0.61	0.57	0.59	733
1	0.73	0.75	0.74	1099
accuracy			0.68	1832
macro avg	0.67	0.66	0.67	1832
weighted avg	0.68	0.68	0.68	1832

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Random Forest: ROC AUC=0.664
```

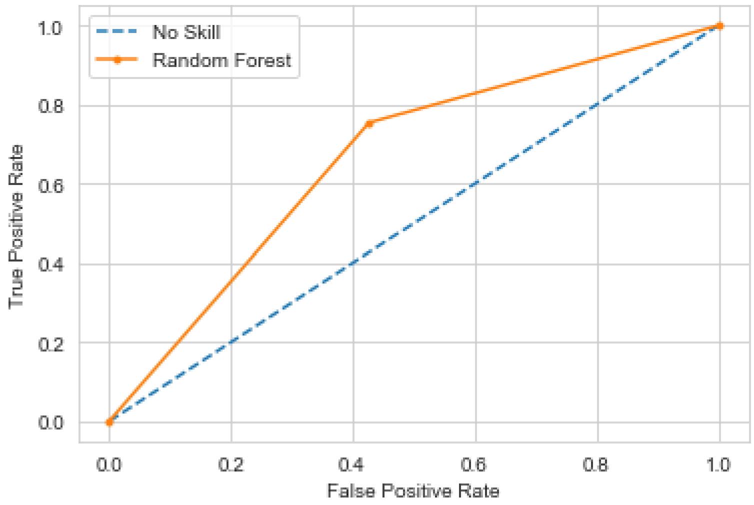

Figure 15: ROC Curve for Random Forest Classifier

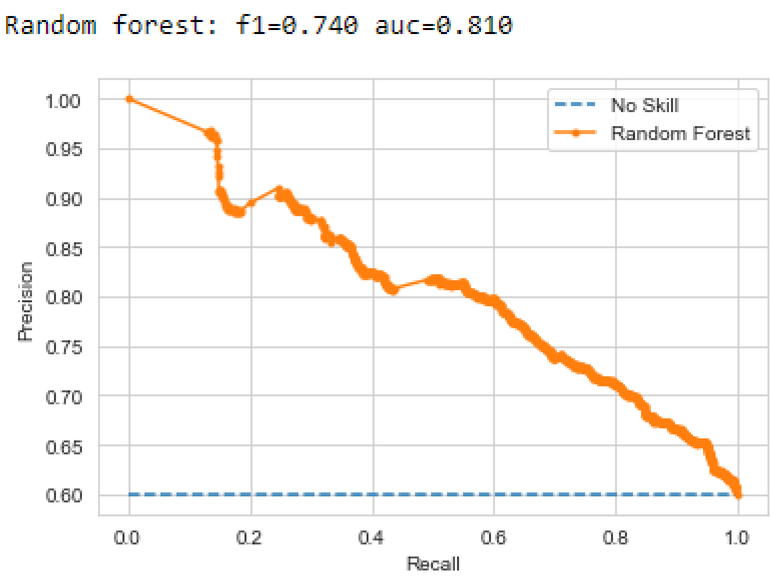

Figure 16: Precision-Recall Curve for Random Forest Classifier

1) Using Random Forest Classifier

1.1) Using Grid Search Cross-validation (CV)

1. Importing the necessary packages

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import
accuracy_score,precision_score,confusion_matrix,classification_report,f1_score,recall_score

2. Using train-test split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

model=RandomForestClassifier(n_estimators=10,random_state=5)
parameters={'min_samples_split':[2,3,4,5],'criterion':['gini','entropy'],'min_samples_leaf':[1,2,3],'n_estimators':[10,20],'random_state' : [5]}
grid=GridSearchCV(model,parameters,scoring='accuracy',cv=15)
grid.fit(X_train,y_train)

4. Find out the grid parameter values like params, best score

5. Predict the model

y_train_predict=grid.predict(X_train)
y_predict=grid.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[379 313]
 [234 906]]
```

col_0	0	1
Status		
0	379	313
1	234	906

We get the following performance metrics:

Accuracy of Training = 84.35225618631732
Accuracy of Testing = 70.14192139737992
Precision score = 69.60305964549477
Recall score = 70.14192139737992
F1 score = 69.73812623563275

Classification report using GridSearchCV in Random Forest Classifier is as follows:

```
              precision    recall  f1-score   support

           0       0.62      0.55      0.58       692
           1       0.74      0.79      0.77      1140

    accuracy                           0.70      1832
   macro avg       0.68      0.67      0.67      1832
weighted avg       0.70      0.70      0.70      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Random Forest: ROC AUC=0.664
```

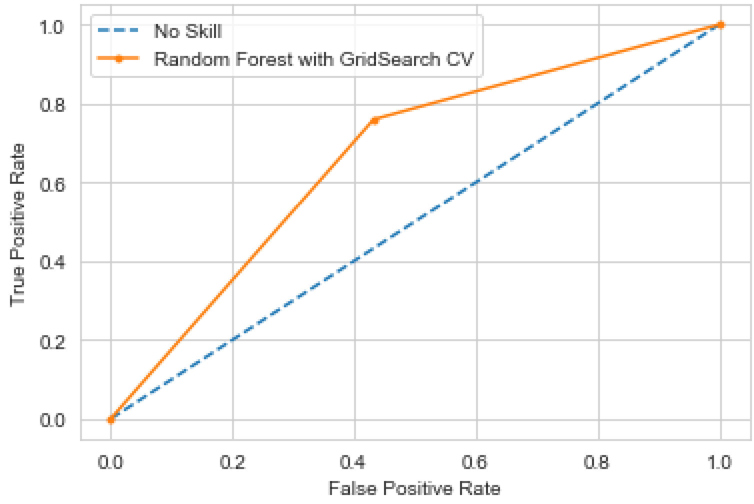

Figure 17: ROC Curve for Random Forest Classifier with GridSearch CV

```
Random forest: f1=0.751 auc=0.813
```

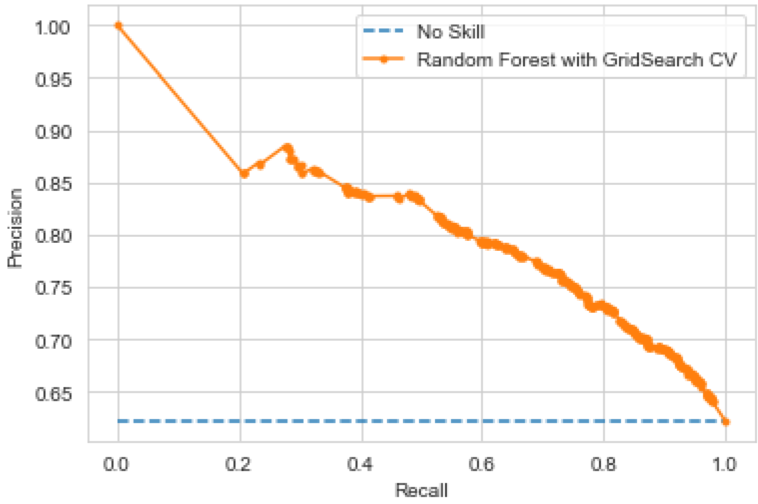

Figure 18: Precision-Recall Curve for Random Forest Classifier with GridSearch CV

1.2) Using RandomizedSearch Cross Validation (CV)

1. <u>Define and Train the model</u>

model=RandomForestClassifier(n_estimators=10,random_state=5)
parameters={'min_samples_split':[2,3,4,5],'criterion':['gini','entropy'],'min_samples_leaf':[1,2,3],'n_estimators':[10,20],'random_state' : [5]}
randomized=RandomizedSearchCV(model,parameters,scoring='accuracy',cv=15)

randomized.fit(X_train,y_train)

2. Find the best parameters and best score.

3. <u>Predict the model</u>

y_train_predict=randomized.predict(X_train)
y_predict=randomized.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
The confusion matrix is as follows:

```
[[379 313]
 [234 906]]
```

col_0	0	1
Status		
0	379	313
1	234	906

We get the following performance metrics:

Accuracy of Training = 84.35225618631732
Accuracy of Testing = 70.14192139737992
Precision score = 69.60305964549477
Recall score = 70.14192139737992
F1 score = 69.73812623563275

Classification report using RandomizedSearchCV in Random Forest Classifier is as follows:

```
              precision    recall  f1-score   support

           0       0.62      0.55      0.58       692
           1       0.74      0.79      0.77      1140

    accuracy                           0.70      1832
   macro avg       0.68      0.67      0.67      1832
weighted avg       0.70      0.70      0.70      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Random Forest: ROC AUC=0.664
```

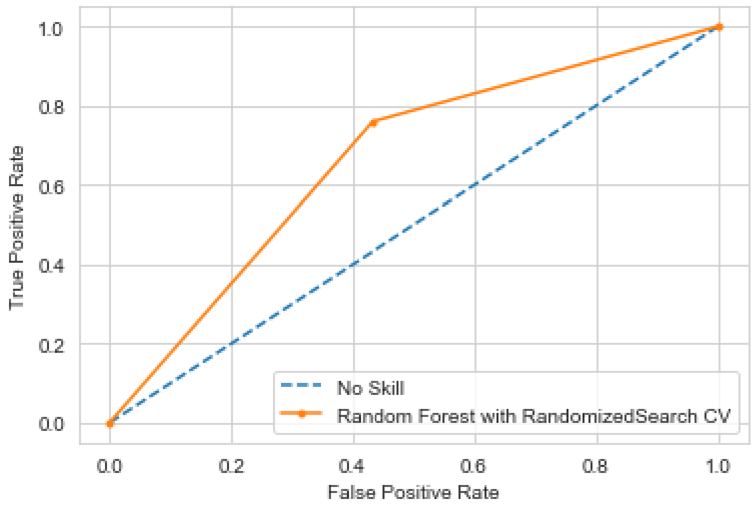

Figure 19: ROC Curve for Random Forest Classifier with RandomizedSearch CV

```
Random forest: f1=0.751 auc=0.813
```

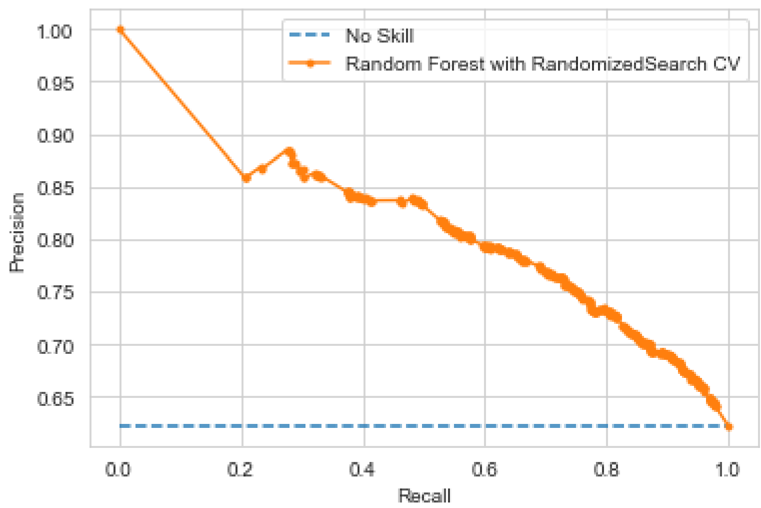

Figure 20: Precision-Recall Curve for Random Forest Classifier with RandomizedSearch CV

1.3) Using Feature Engineering

data.corr()['Status'].sort_values()

1. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

2. <u>Define and Train the model</u>

```
model=RandomForestClassifier(n_estimators=10,criterion='gini',max_depth=3,min_samples_split=2, min_samples_leaf=1,random_state=5)
model.fit(X_train,y_train)
pd.DataFrame(model.feature_importances_,index=X.columns).sort_values(0,ascending=False)
```

3. Predict the model

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

The confusion matrix is as follows:

```
[[320 372]
 [158 982]]
```

col_0	0	1
Status		
0	320	372
1	158	982

We get the following performance metrics:

Accuracy of Training = 69.30494905385734
Accuracy of Testing = 71.06986899563319
Precision score = 70.41802842515348
Recall score = 71.06986899563319
F1 score = 69.66531043144609

Classification report using feature engineering is as follows:

```
              precision    recall  f1-score   support

           0       0.67      0.46      0.55       692
           1       0.73      0.86      0.79      1140

    accuracy                           0.71      1832
   macro avg       0.70      0.66      0.67      1832
weighted avg       0.70      0.71      0.70      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Random Forest: ROC AUC=0.662
```

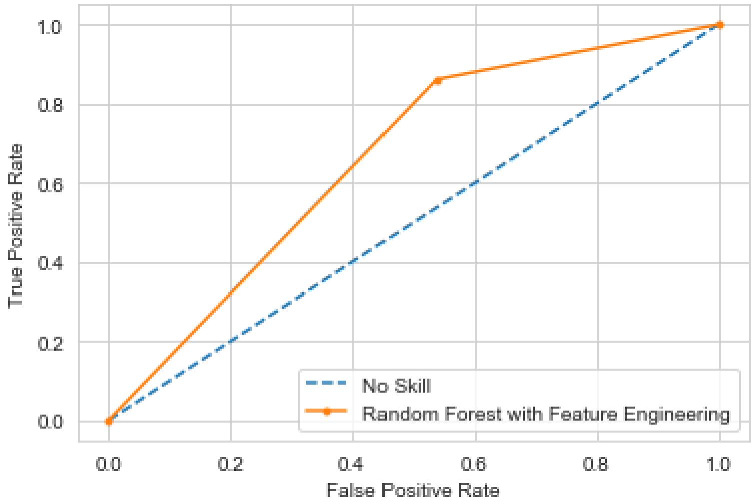

Figure 21: ROC Curve for Random Forest Classifier with Feature Engineering

```
Random forest: f1=0.787 auc=0.842
```

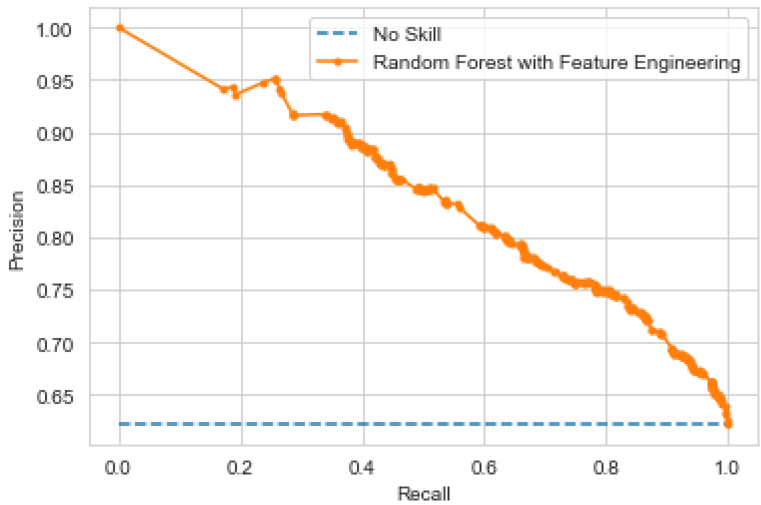

Figure 22: Precision-Recall Curve for Random Forest Classifier with Feature Engineering

2) Using XGBoosting Classifier

1. <u>Import the necessary package</u>

from xgboost import XGBClassifier

2. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

model=XGBClassifier(max_depth=3,learning_rate=0.01,test_size=0.25,n_estimators=500,n_jobs=1,random_state=10,gamma=5)
model.fit(X_train,y_train)

Predict the model.

y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[372 320]
 [197 943]]
```

col_0	0	1
Status		
0	372	320
1	197	943

We get the following performance metrics:

Accuracy score of Training = 72.48908296943232
Accuracy score of Testing = 71.77947598253274
Precision score = 71.15604029632028
Recall score = 71.77947598253274
F1 score = 71.12538609313704

Classification report using XGBoost Classifier is as follows:

	precision	recall	f1-score	support
0	0.65	0.54	0.59	692
1	0.75	0.83	0.78	1140
accuracy			0.72	1832
macro avg	0.70	0.68	0.69	1832
weighted avg	0.71	0.72	0.71	1832

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
XGBoost: ROC AUC=0.682
```

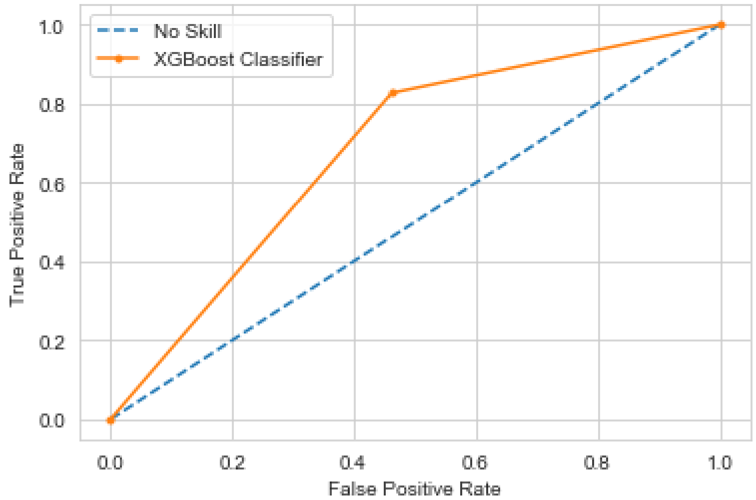

Figure 23: ROC Curve for XGBoost Classifier

```
XGBoost : f1=0.785 auc=0.863
```

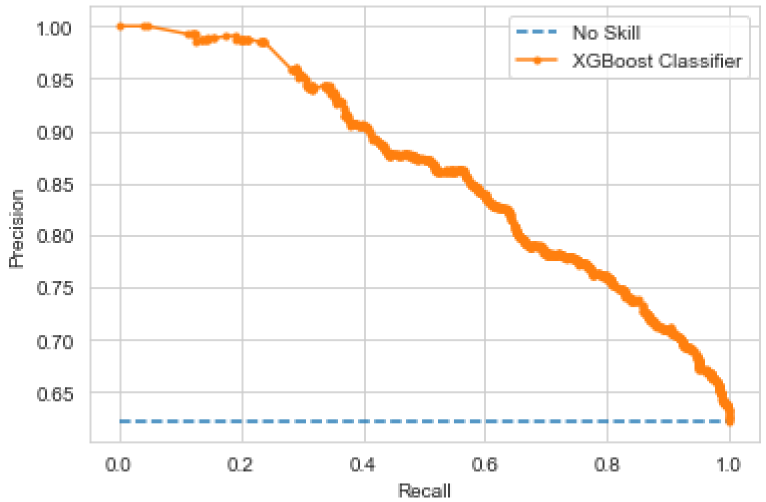

Figure 24: Precision-Recall Curve for XGBoost Classifier

3) Using Gradient Boosting classifier

1. Import the necessary package

from sklearn.ensemble import GradientBoostingClassifier

Using train-test split.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

2. Define and Train the model

model=GradientBoostingClassifier(learning_rate=0.1,n_estimators=100,subsample=1.0,max_depth=3,random_state=10)
model.fit(X_train,y_train)
print(" Model Feature Importances = " ,model.feature_importances_)

3. Predict the model

y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[375 317]
 [194 946]]
```

col_0	0	1
Status		
0	375	317
1	194	946

We get the following performance metrics:

Accuracy of Training = 74.59970887918487
Accuracy of Testing = 72.10698689956332
Precision score = 71.50300233949454
Recall score = 72.10698689956332
F1 score = 71.46048799534435

Classification report using Gradient Boosting Classifier is as follows:

	precision	recall	f1-score	support
0	0.66	0.54	0.59	692
1	0.75	0.83	0.79	1140
accuracy			0.72	1832
macro avg	0.70	0.69	0.69	1832
weighted avg	0.72	0.72	0.71	1832

The following are the ROC and Precision-Recall Curves.

No Skill: ROC AUC=0.500
Gradient Boosting: ROC AUC=0.686

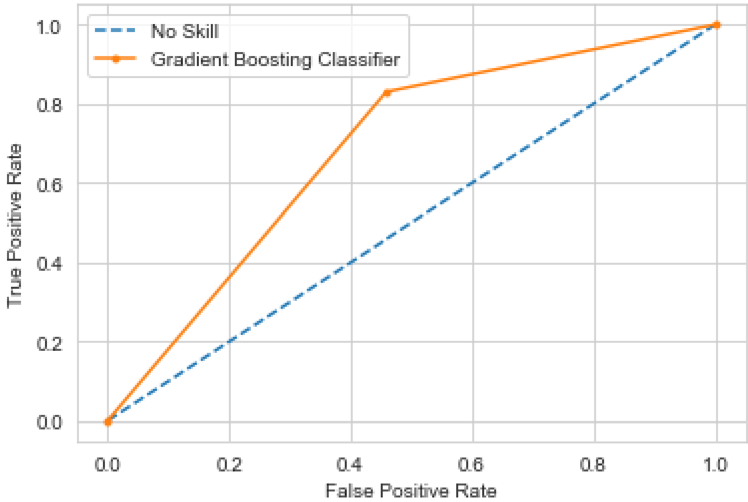

Figure 25: ROC Curve for Gradient Boosting Classifier

Gradient Boosting : f1=0.787 auc=0.867

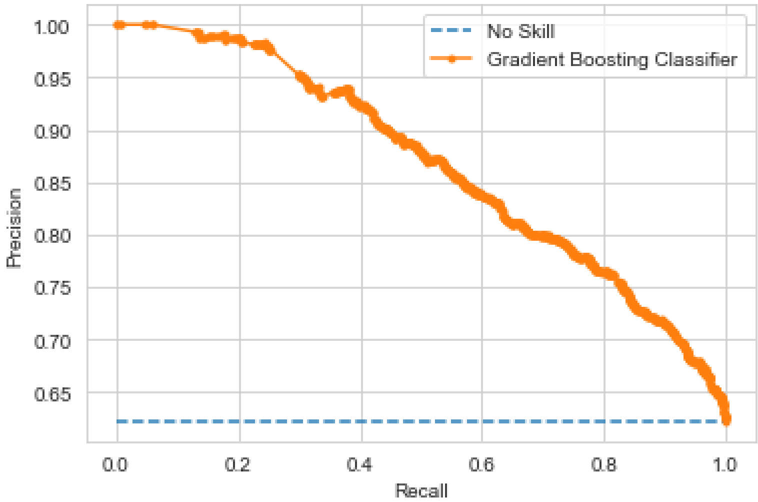

Figure 26: Precision-Recall Curve for Gradient Boosting Classifier

4) Using Support Vector machine (SVM)

1. <u>Import the package</u>

from sklearn.svm import SVC

2. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

```
model=SVC(C=130,kernel = 'rbf', degree=4,gamma='scale',
random_state=10,probability=True)
model.fit(X_train,y_train)
```

4. Predict the model

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

The confusion matrix is as follows:

```
[[342 350]
 [214 926]]
```

col_0	0	1
Status		
0	342	350
1	214	926

We get the following performance metrics:

Accuracy score of Training = 72.45269286754002
Accuracy score of Testing = 69.21397379912663
Precision score = 68.39294495055145
Recall score = 69.21397379912663
F1 score = 68.40302417874788

Classification report using SVM is as follows:

	precision	recall	f1-score	support
0	0.62	0.49	0.55	692
1	0.73	0.81	0.77	1140
accuracy			0.69	1832
macro avg	0.67	0.65	0.66	1832
weighted avg	0.68	0.69	0.68	1832

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
SVM: ROC AUC=0.653
```

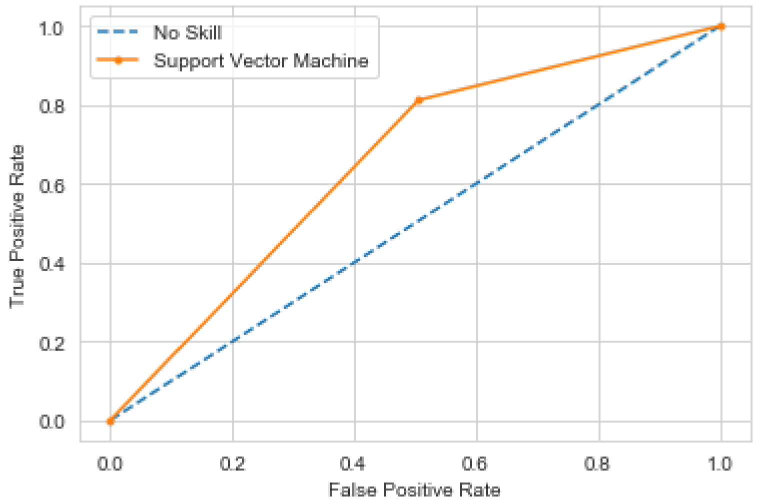

Figure 27: ROC Curve for Support Vector Machine

```
SVM : f1=0.767 auc=0.767
```

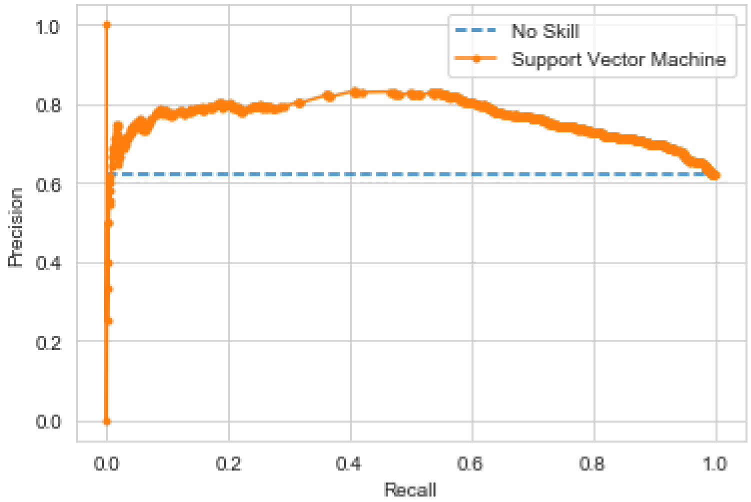

Figure 28: Precision-Recall Curve for Support Vector Machine

5) Using Artificial Neural Networks (ANN)

1. <u>Import the package</u>

from sklearn.neural_network import MLPClassifier

2. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

model=MLPClassifier(hidden_layer_sizes=10,activation='relu',alpha=0.001,batch_size=10,learning_rate_init=0.01,random_state=5)
model.fit(X_train,y_train)

4. Predict the model

y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[360 332]
 [241 899]]
```

col_0	0	1
Status		
0	360	332
1	241	899

We get the following performance metrics:

Accuracy score of Training = 68.10407569141194
Accuracy score of Testing = 68.72270742358079
Precision score = 68.0705130823014
Recall score = 68.72270742358079
F1 score = 68.22229367718045

Classification report using ANN is as follows:

	precision	recall	f1-score	support
0	0.60	0.52	0.56	692
1	0.73	0.79	0.76	1140
accuracy			0.69	1832
macro avg	0.66	0.65	0.66	1832
weighted avg	0.68	0.69	0.68	1832

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
ANN: ROC AUC=0.654
```

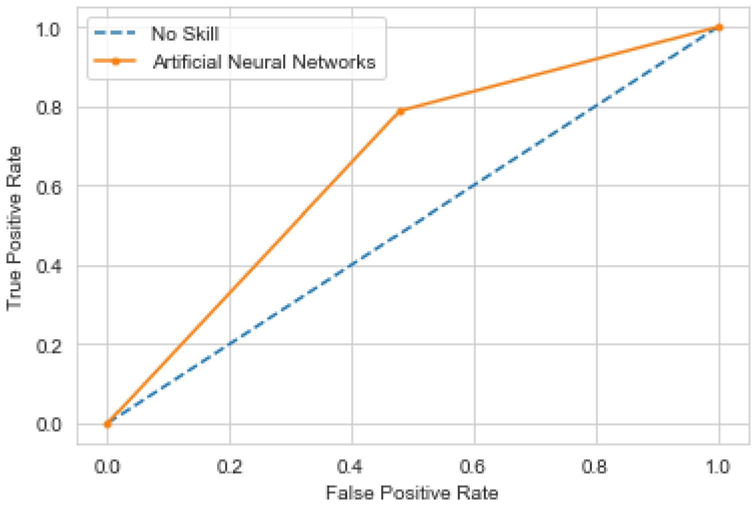

Figure 29: ROC Curve for Artificial Neural Networks

```
ANN : f1=0.758 auc=0.801
```

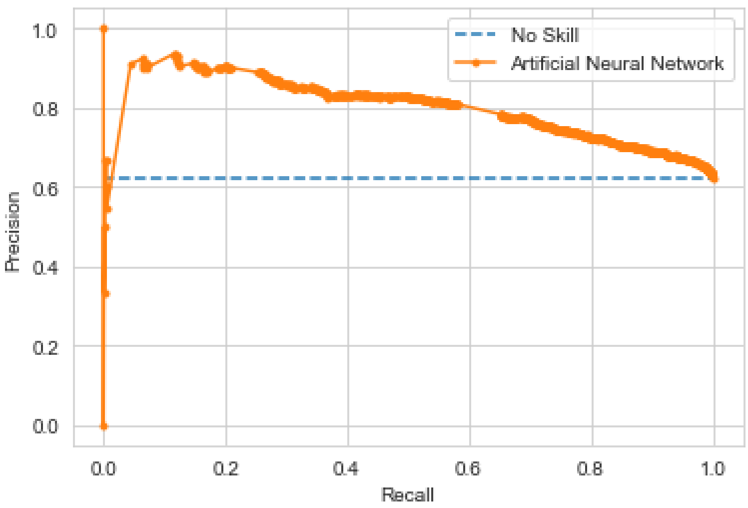

Figure 30: Precision-Recall Curve for Artificial Neural Networks

6) Using Decision Tree classifier

1. Import the package

from sklearn.tree import DecisionTreeClassifier

2. Using Train-test split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

```
model=DecisionTreeClassifier(splitter='best',  random_state=5,min_samples_split=2, max_depth=4, min_samples_leaf=1,criterion='gini')
model.fit(X_train,y_train)
print(" Model Feature Importances = " ,model.feature_importances_)
```

4. <u>Predict the model</u>

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

The confusion matrix is as follows:

```
[[275 417]
 [145 995]]
```

col_0	0	1
Status		
0	275	417
1	145	995

We get the following performance metrics:

Accuracy score of Training = 68.79548762736536
Accuracy score of Testing = 69.32314410480349
Precision score = 68.58208788172789
Recall score = 69.32314410480349
F1 score = 67.2061151916238

Classification report using Decision Tree Classifier is as follows:

	precision	recall	f1-score	support
0	0.65	0.40	0.49	692
1	0.70	0.87	0.78	1140
accuracy			0.69	1832
macro avg	0.68	0.64	0.64	1832
weighted avg	0.69	0.69	0.67	1832

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Decision Tree Classifier: ROC AUC=0.635
```

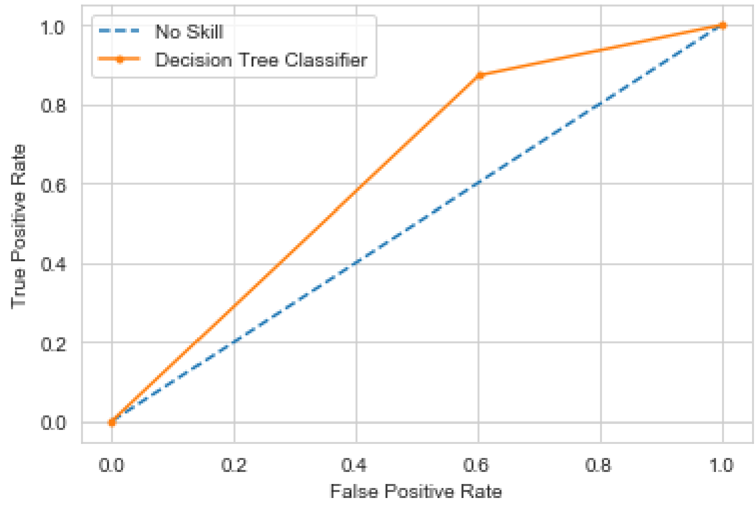

Figure 31: ROC Curve for Decision Tree Classifier

```
Decision Tree Classifier  : f1=0.780 auc=0.699
```

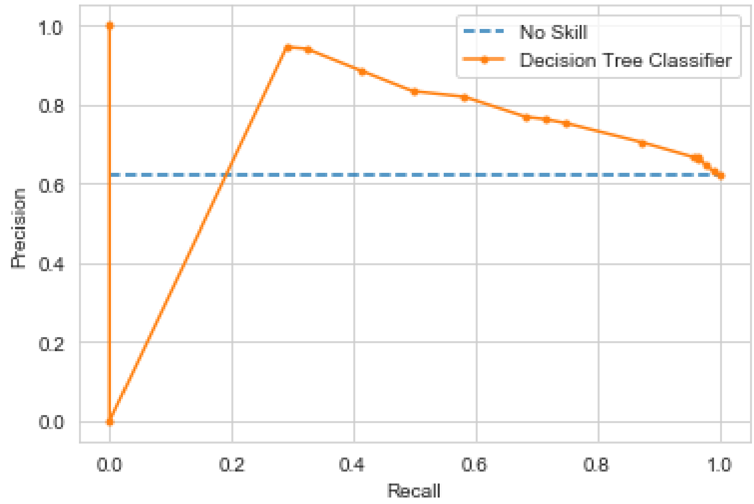

Figure 32: Precision-Recall Curve for Decision Tree Classifier

7) Using Logistic Regression

1. Import the necessary package

from sklearn.linear_model import LogisticRegression

2. Using train-test split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model

```
model=LogisticRegression(random_state=5,C=2.0,multi_class='ovr')
model.fit(X_train,y_train)
```

4. Predict the model

```
y_train_predict=model.predict(X_train)
y_predict= model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

The confusion matrix is as follows:

```
[[355 337]
 [225 915]]
```

col_0	0	1
Status		
0	355	337
1	225	915

We get the following performance metrics:

Accuracy score of Training = 66.64847161572052
Accuracy score of Testing = 69.32314410480349
Precision score = 69.32314410480349
Recall score = 69.32314410480349
F1 score = 69.32314410480349

Classification report using logistic regression is as follows:

```
              precision    recall  f1-score   support

           0       0.61      0.51      0.56       692
           1       0.73      0.80      0.77      1140

    accuracy                           0.69      1832
   macro avg       0.67      0.66      0.66      1832
weighted avg       0.69      0.69      0.69      1832
```

The following are the ROC and Precision-Recall Curves.

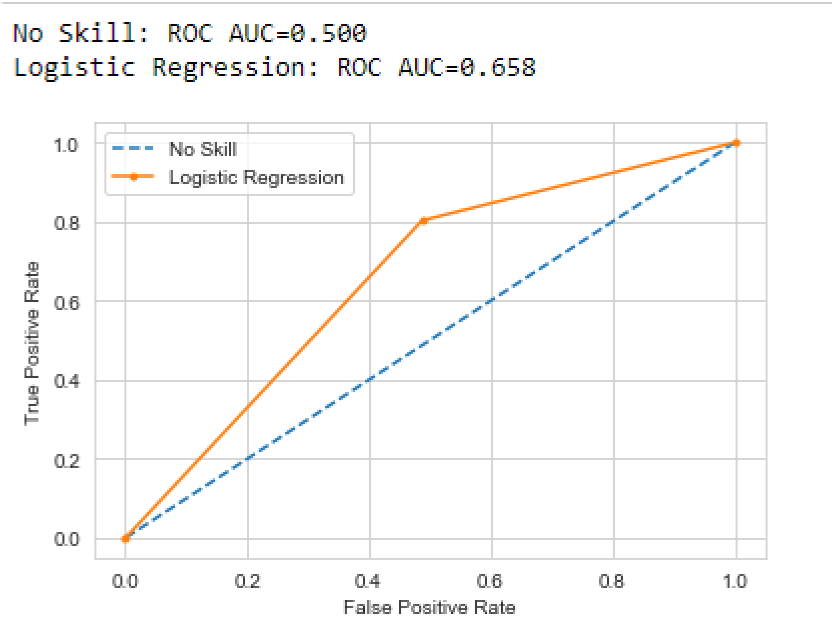

Figure 33: ROC Curve for Logistic Regression

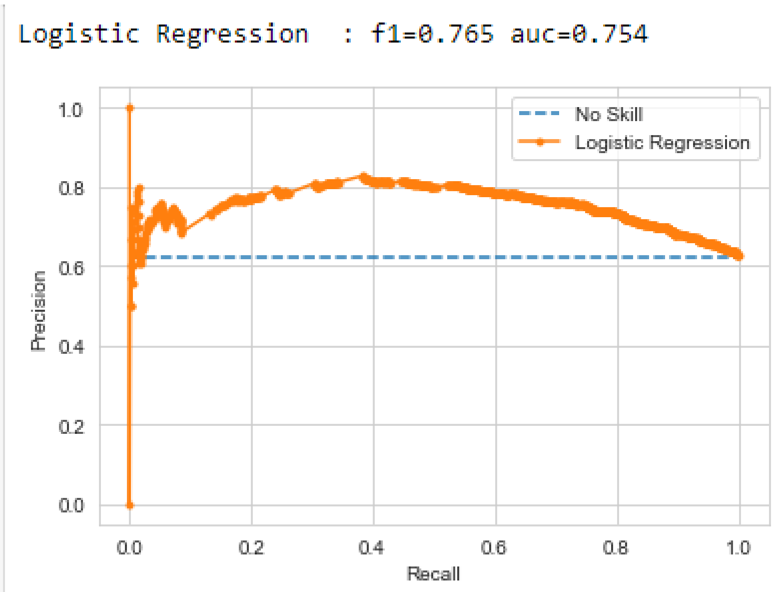

Figure 34: Precision-Recall Curve for Logistic Regression

8) Using K-Nearest Neighbors (KNN)

1. <u>Import the package</u>

from sklearn.neighbors import KNeighborsClassifier

2. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. <u>Define and Train the model</u>

```
model= KNeighborsClassifier(n_neighbors=100, metric='minkowski')
model.fit(X_train,y_train)
```

4. <u>Predict the model</u>

```
y_train_predict=model.predict(X_train)
y_predict = model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

The confusion matrix is as follows:

```
[[356 336]
 [224 916]]
```

col_0	0	1
Status		
0	356	336
1	224	916

We get the following performance metrics:

Accuracy score of Training = 69.28675400291121
Accuracy score of Testing = 69.43231441048034
Precision score = 68.7119178806456
Recall score = 69.43231441048034
F1 score = 68.80219868136803

Classification report using KNN is as follows:

```
              precision    recall  f1-score   support

           0       0.61      0.51      0.56       692
           1       0.73      0.80      0.77      1140

    accuracy                           0.69      1832
   macro avg       0.67      0.66      0.66      1832
weighted avg       0.69      0.69      0.69      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
KNN: ROC AUC=0.659
```

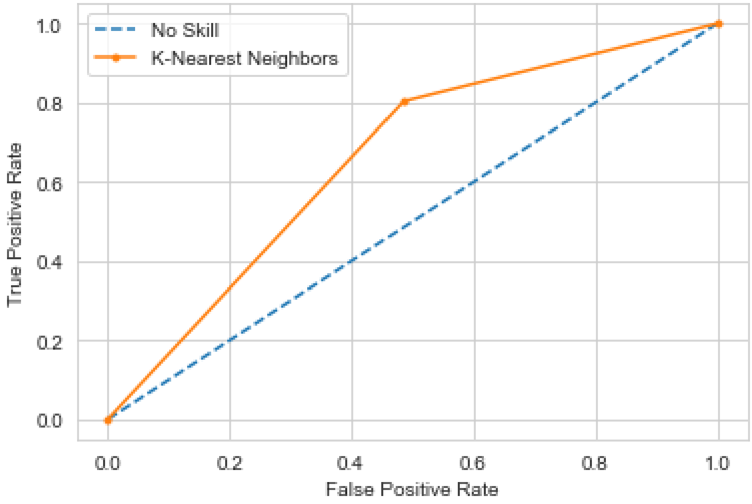

Figure 35: ROC Curve for K-Nearest Neighbors

```
K-Nearest Neighbors  : f1=0.766 auc=0.851
```

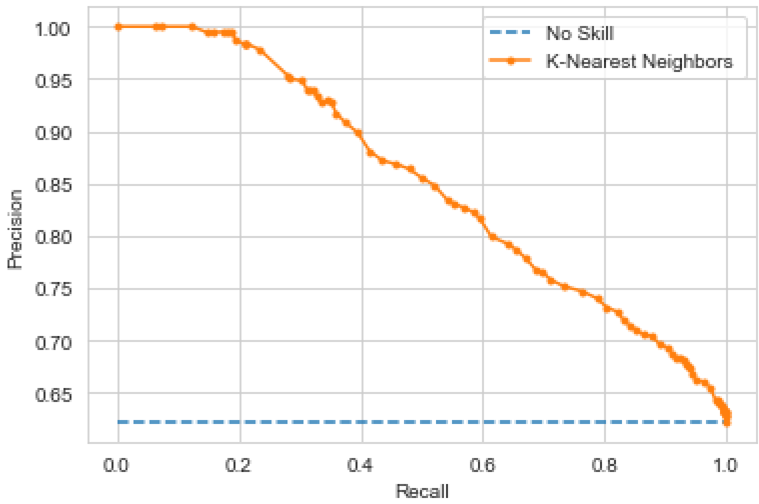

Figure 36: Precision-Recall Curve for K-Nearest Neighbors

9) Using Naïve Bayes Classifier

1. Import the package

from sklearn.naive_bayes import BernoulliNB, GaussianNB

2. Using train-test split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. Define and Train the model for Bernoulli Naïve Bayes

```
model_bernoulli = BernoulliNB(alpha=2.0)
model_bernoulli.fit(X_train,y_train)
```

4. Predict the model

```
y_predict_bernoulli = model_bernoulli.predict(X_test)
print(confusion_matrix(y_test,y_predict_bernoulli))
pd.crosstab(y_test,y_predict_bernoulli)
```

The confusion matrix is as follows:

```
[[387 305]
 [276 864]]
```

col_0	0	1
Status		
0	387	305
1	276	864

We get the following performance metrics:

Accuracy score of Testing = 68.28602620087337
Precision score = 68.04005992300351
Recall score = 68.28602620087337
F1 score = 68.14580818679296

Classification report using Bernoulli Naïve Bayes is as follows:

```
              precision    recall  f1-score   support

           0       0.58      0.56      0.57       692
           1       0.74      0.76      0.75      1140

    accuracy                           0.68      1832
   macro avg       0.66      0.66      0.66      1832
weighted avg       0.68      0.68      0.68      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Bernoulli NB: ROC AUC=0.659
```

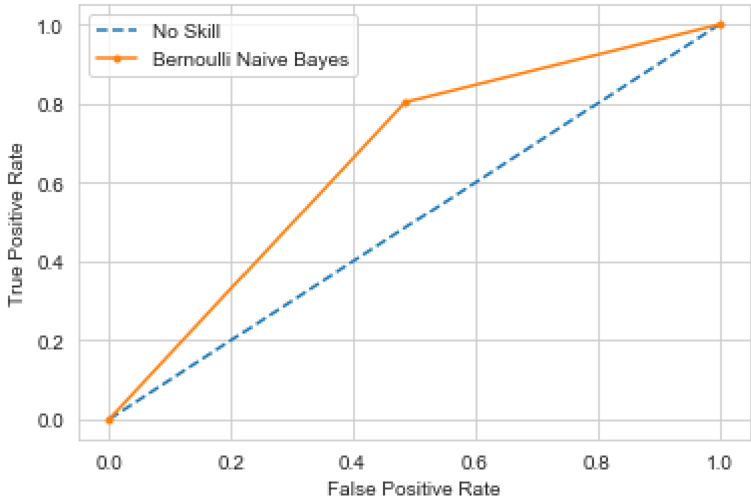

Figure 37: ROC Curve for Bernoulli Naïve Bayes

```
Bernoulli Naive Bayes  : f1=0.766 auc=0.851
```

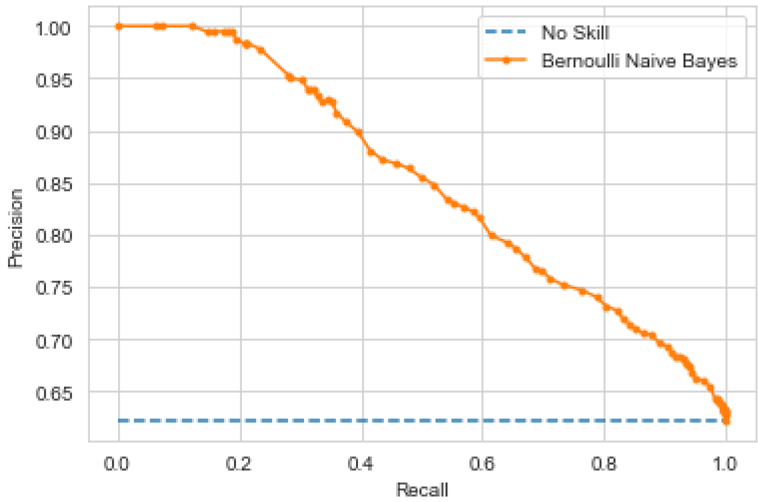

Figure 38: Precision-Recall Curve for Bernoulli Naïve Bayes

5. <u>Define and Train the model for Gaussian Naïve Bayes</u>

model_gaussian = GaussianNB()
model_gaussian.fit(X_train,y_train)

6. <u>Predict the model</u>

y_predict_gaussian = model_gaussian.predict(X_test)
print(confusion_matrix(y_test,y_predict_gaussian))
pd.crosstab(y_test,y_predict_gaussian)

```
[[361 331]
 [241 899]]
```

col_0	0	1
Status		
0	361	331
1	241	899

We get the following performance metrics:

Accuracy score of Testing = 68.77729257641921
Precision score = 68.13262073487424
Recall score = 68.77729257641921
F1 score = 68.28433139451188

Classification report using Gaussian Naïve Bayes is as follows:

```
              precision    recall  f1-score   support

           0       0.60      0.52      0.56       692
           1       0.73      0.79      0.76      1140

    accuracy                           0.69      1832
   macro avg       0.67      0.66      0.66      1832
weighted avg       0.68      0.69      0.68      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Gaussian NB: ROC AUC=0.659
```

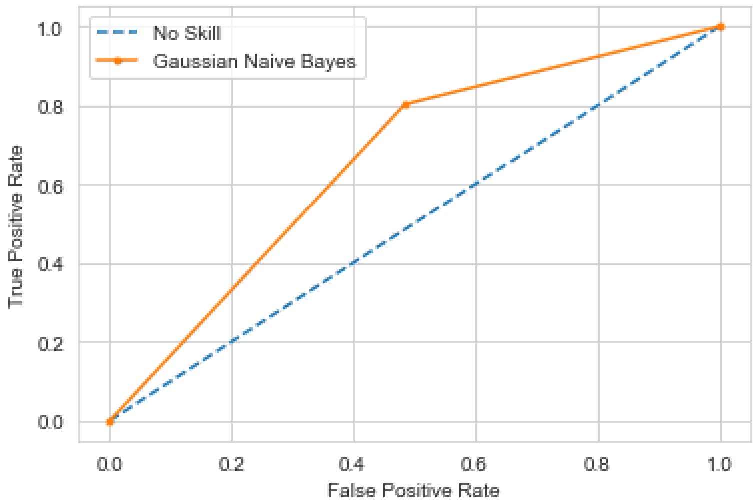

Figure 39: ROC Curve for Gaussian Naïve Bayes

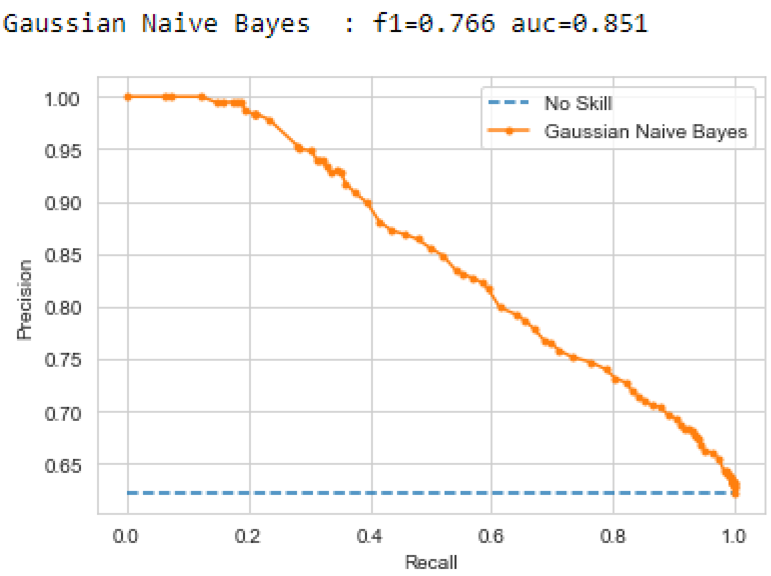

Figure 40: Precision-Recall Curve for Gaussian Naïve Bayes

10) Using ExtraTrees Classifier

1. <u>Import the package</u>

from sklearn.ensemble import ExtraTreesClassifier

2. <u>Using train-test split</u>

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=5)

3. <u>Define and Train the model</u>

model=ExtraTreesClassifier(n_estimators=100,criterion='gini',random_state=10,max_depth=3)
model.fit(X_train,y_train)
print("Model Feature Importances = " ,model.feature_importances_)

4. <u>Predict the model</u>

y_train_predict=model.predict(X_train)
y_predict= model.predict(X_test)
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)

The confusion matrix is as follows:

```
[[ 148  544]
 [  44 1096]]
```

col_0	0	1
Status		
0	148	544
1	44	1096

We get the following performance metrics:

Accuracy score of Training = 66.41193595342067
Accuracy score of Testing = 67.90393013100436
Precision score = 70.70252866829978
Recall score = 67.90393013100436
F1 score = 61.713324513184794

Classification report using Extra Trees Classifier is as follows:

```
              precision    recall  f1-score   support

           0       0.77      0.21      0.33       692
           1       0.67      0.96      0.79      1140

    accuracy                           0.68      1832
   macro avg       0.72      0.59      0.56      1832
weighted avg       0.71      0.68      0.62      1832
```

The following are the ROC and Precision-Recall Curves.

```
No Skill: ROC AUC=0.500
Extra Trees Classifier: ROC AUC=0.588
```

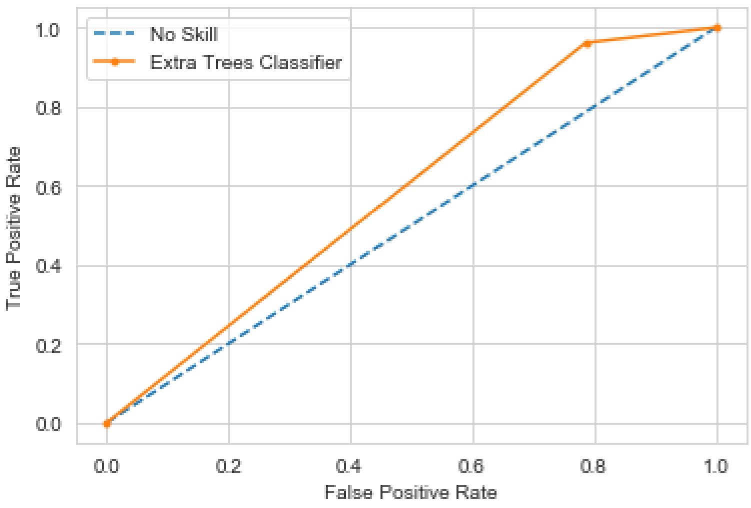

Figure 41: ROC Curve for Extra Trees Classifier

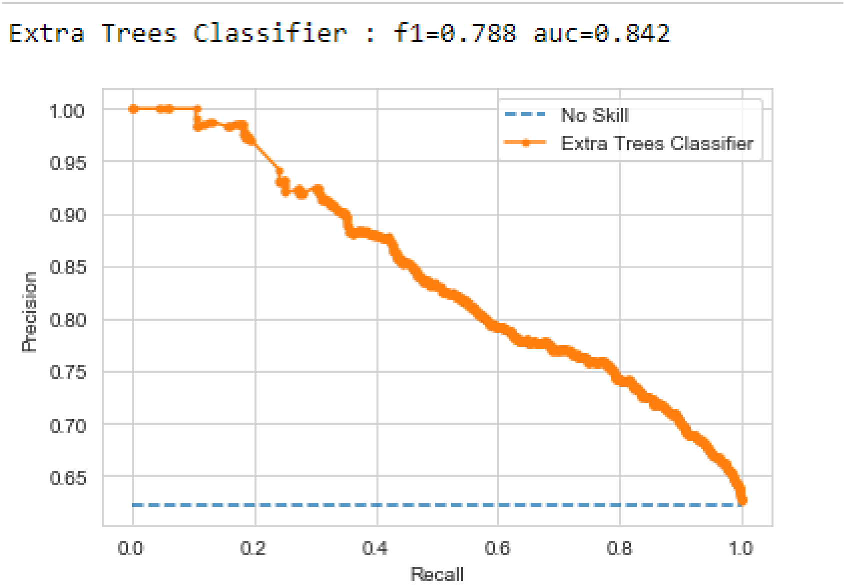

Figure 42: Precision-Recall Curve for Extra Trees Classifier