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/project\_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,recall\_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.xlsx",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

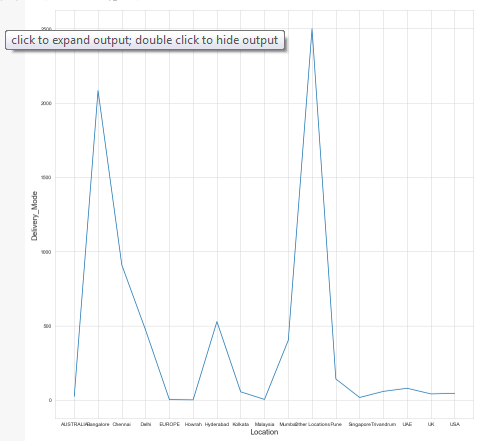
Name: Delivery\_Mode, dtype: int64

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

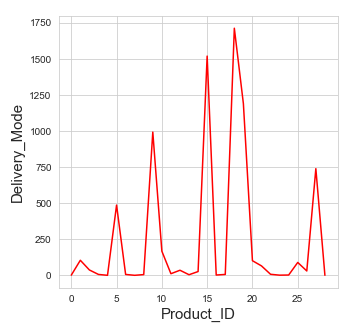
Name: Delivery\_Mode, dtype: int64

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

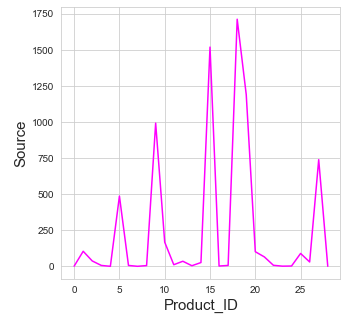
Name: Source, dtype: int64

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

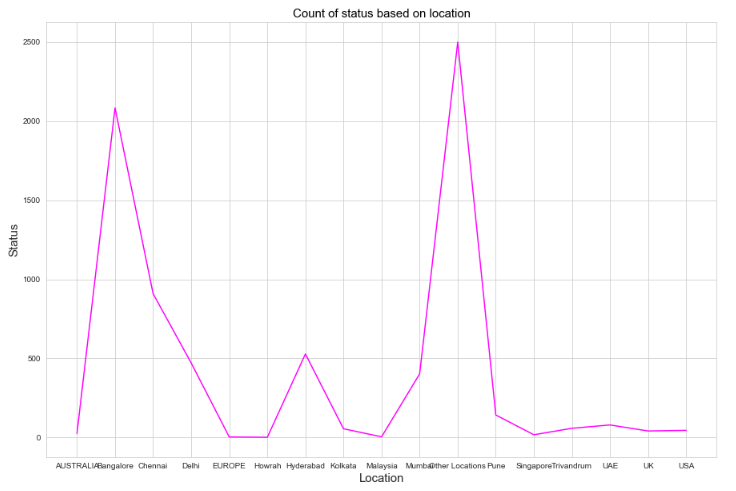
plt.figure(figsize=(15,10))

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()

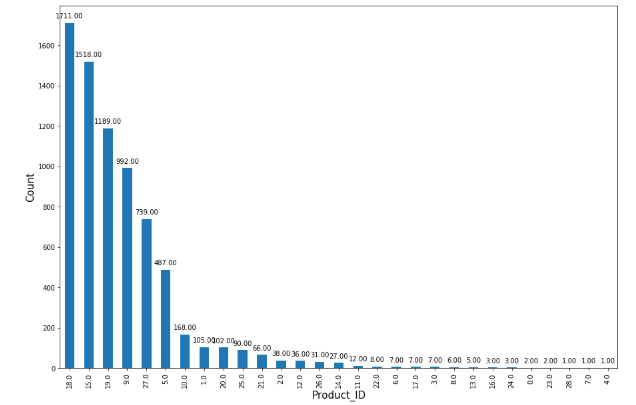
plt.xlabel('Product\_ID',fontsize=15)

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()

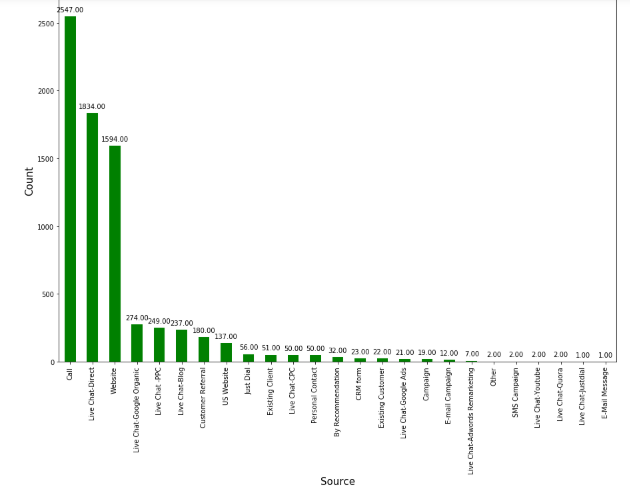
plt.xlabel('Source',fontsize=15)

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()

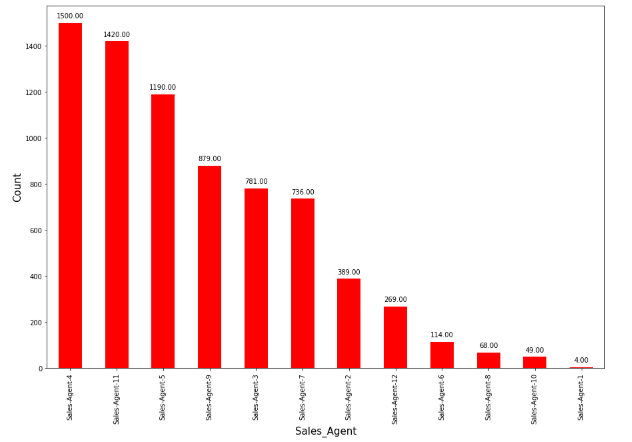
plt.xlabel('Sales\_Agent',fontsize=15)

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()

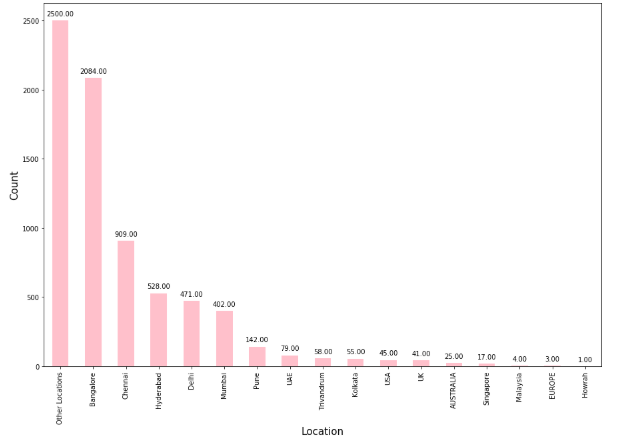
plt.xlabel('Location',fontsize=15)

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()

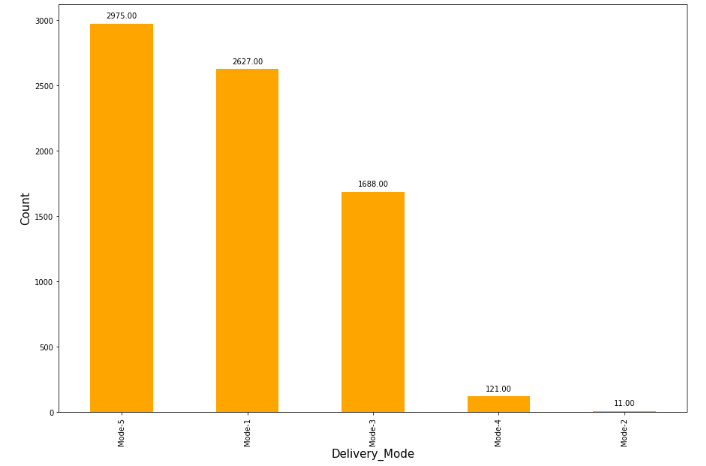
plt.xlabel('Delivery\_Mode',fontsize=15)

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()

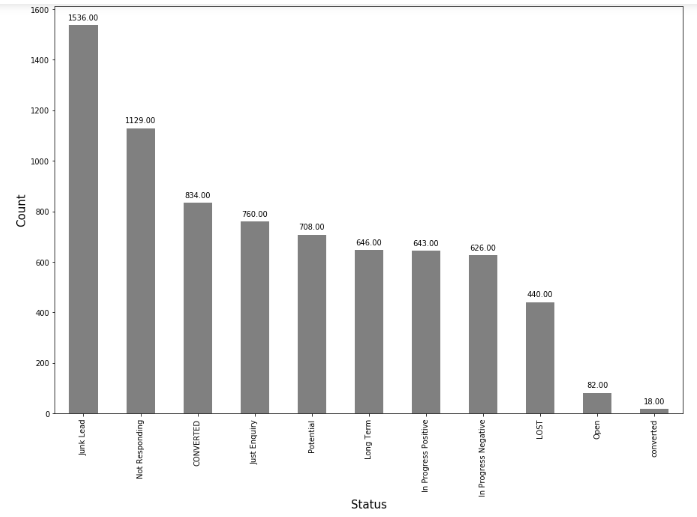
plt.xlabel('Status',fontsize=15)

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)

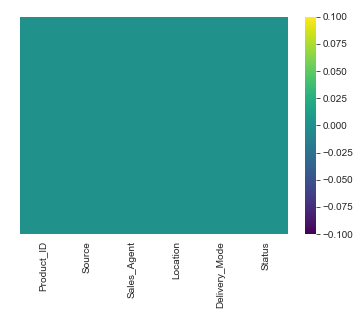
data.Status=enc.fit\_transform(data.Status)

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')



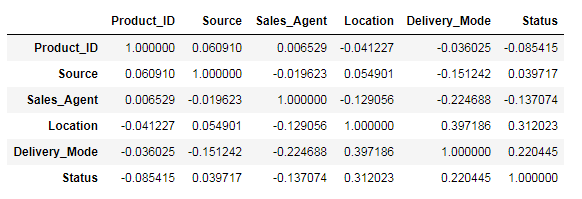
**Figure 12: Heatmap showing data fields**

8. Correlation Matrix

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

corr=data.corr()

corr



**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**

1. **ML model to predict the Lead Category (High Potential , Low  
   Potential)**
2. Define X and y variables

X=data[['Product\_ID','Source','Sales\_Agent','Location','Delivery\_Mode']]

y=data.Status

1. Using train-test split

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)

1. Using Standard Scaler

from sklearn.preprocessing import StandardScaler

s = StandardScaler()

X\_train = s.fit\_transform(X\_train)

X\_test = s.transform(X\_test)

1. Using SMOTE Technique

from imblearn.over\_sampling import SMOTE

smote=SMOTE()

X\_train, y\_train = smote.fit\_sample(X\_train,y\_train)

1. Using PCA Technique

from sklearn.decomposition import PCA

pca=PCA()

X=pd.DataFrame(pca.fit\_transform(X))

pca.explained\_variance\_ratio\_

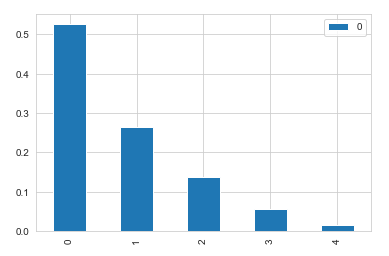
**Output: array([0.52549458, 0.26459349, 0.13698127, 0.05666528, 0.01626539])**

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)

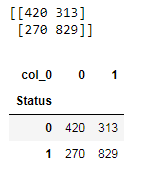
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy of Training = 86.35371179039302

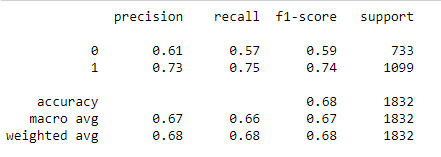
Accuracy of Testing = 68.17685589519651

Precision score = 67.90171272606725

Recall score = 68.17685589519651

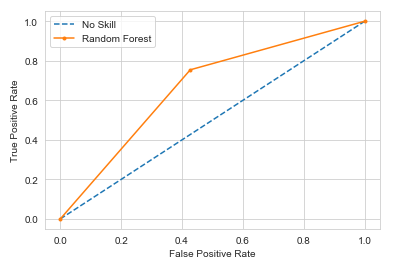
F1 score = 68.0013514614794

Classification report using PCA is as follows:

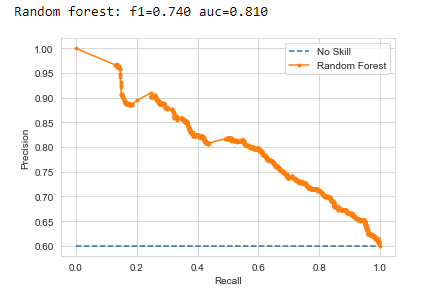


The following are the ROC and Precision-Recall Curves.





**Figure 15: ROC Curve for Random Forest Classifier**



**Figure 16: Precision-Recall Curve for Random Forest Classifier**

1. **Using Random Forest Classifier**
   1. **Using Grid Search Cross-validation (CV)**
2. 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

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. 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)

1. Find out the grid parameter values like params, best score
2. Predict the model

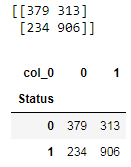
y\_train\_predict=grid.predict(X\_train)

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:



We get the following performance metrics:

Accuracy of Training = 84.35225618631732

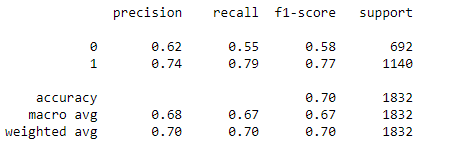
Accuracy of Testing = 70.14192139737992

Precision score = 69.60305964549477

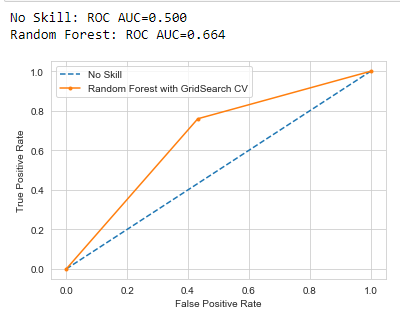
Recall score = 70.14192139737992

F1 score = 69.73812623563275

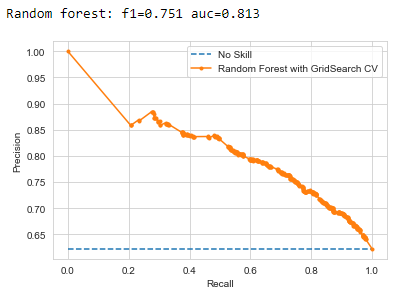
Classification report using GridSearchCV in Random Forest Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



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



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

* 1. **Using RandomizedSearch Cross Validation (CV)**

1. 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]}

randomized=RandomizedSearchCV(model,parameters,scoring='accuracy',cv=15)

randomized.fit(X\_train,y\_train)

1. Find the best parameters and best score.
2. Predict the model

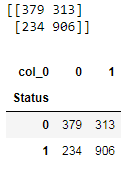
y\_train\_predict=randomized.predict(X\_train)

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:



We get the following performance metrics:

Accuracy of Training = 84.35225618631732

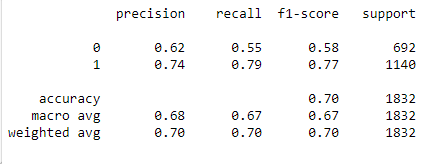
Accuracy of Testing = 70.14192139737992

Precision score = 69.60305964549477

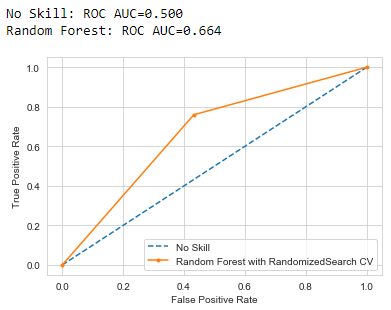
Recall score = 70.14192139737992

F1 score = 69.73812623563275

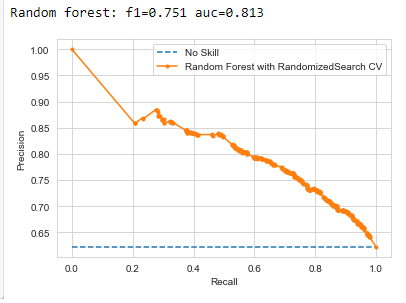
Classification report using RandomizedSearchCV in Random Forest Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



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



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

* 1. **Using Feature Engineering**

data.corr()['Status'].sort\_values()

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. Define and Train the model

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)

1. Predict the model

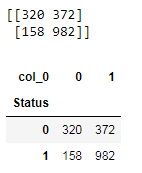
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy of Training = 69.30494905385734

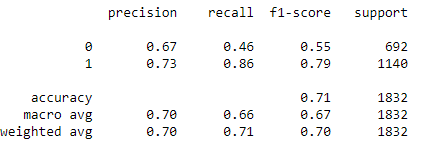
Accuracy of Testing = 71.06986899563319

Precision score = 70.41802842515348

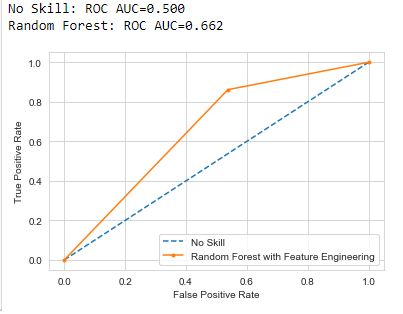
Recall score = 71.06986899563319

F1 score = 69.66531043144609

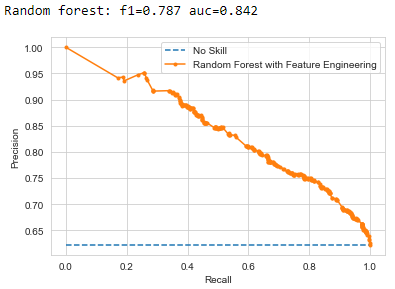
Classification report using feature engineering is as follows:



The following are the ROC and Precision-Recall Curves.



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



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

1. **Using XGBoosting Classifier**
2. Import the necessary package

from xgboost import XGBClassifier

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. 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.

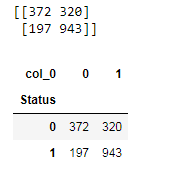
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 72.48908296943232

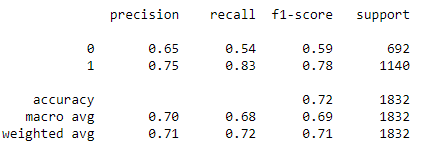
Accuracy score of Testing = 71.77947598253274

Precision score = 71.15604029632028

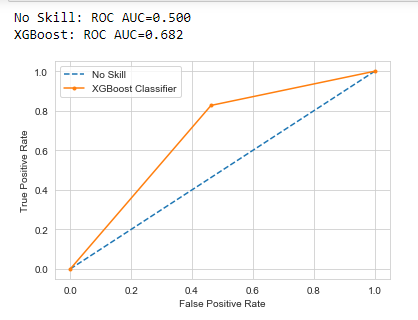
Recall score = 71.77947598253274

F1 score = 71.12538609313704

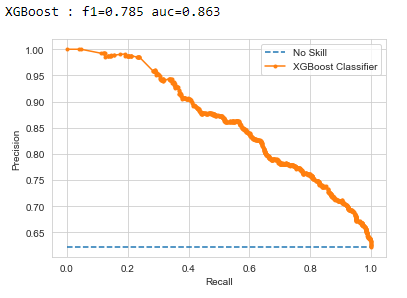
Classification report using XGBoost Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 23: ROC Curve for XGBoost Classifier**



**Figure 24: Precision-Recall Curve for XGBoost Classifier**

1. **Using Gradient Boosting classifier**
2. 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)

1. 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\_)

1. Predict the model

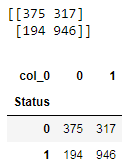
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy of Training = 74.59970887918487

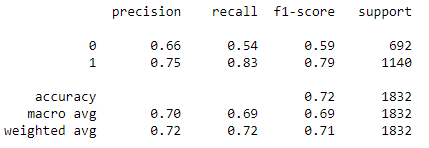
Accuracy of Testing = 72.10698689956332

Precision score = 71.50300233949454

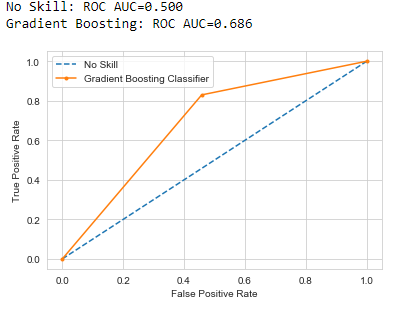
Recall score = 72.10698689956332

F1 score = 71.46048799534435

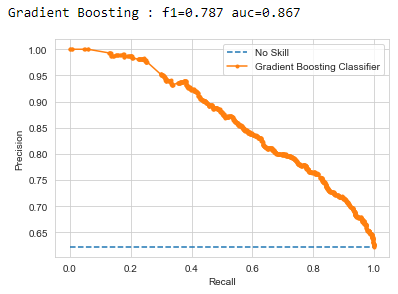
Classification report using Gradient Boosting Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 25: ROC Curve for Gradient Boosting Classifier**



**Figure 26: Precision-Recall Curve for Gradient Boosting Classifier**

1. **Using Support Vector machine (SVM)**
2. Import the package

from sklearn.svm import SVC

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. 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)

1. Predict the model

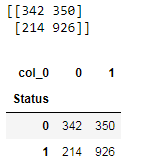
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 72.45269286754002

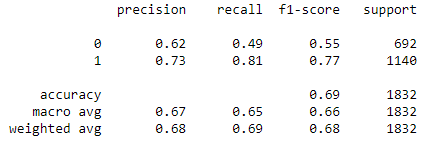
Accuracy score of Testing = 69.21397379912663

Precision score = 68.39294495055145

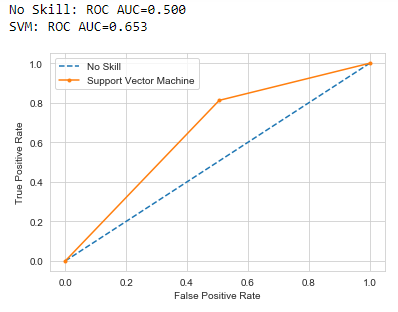
Recall score = 69.21397379912663

F1 score = 68.40302417874788

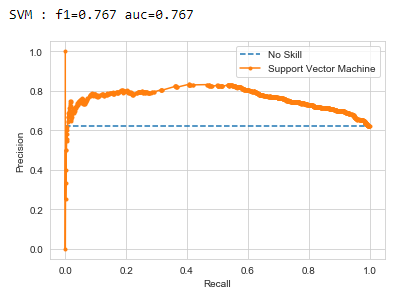
Classification report using SVM is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 27: ROC Curve for Support Vector Machine**



**Figure 28: Precision-Recall Curve for Support Vector Machine**

1. **Using Artificial Neural Networks (ANN)**
2. Import the package

from sklearn.neural\_network import MLPClassifier

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. 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)

1. Predict the model

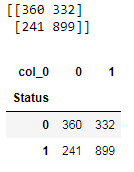
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 68.10407569141194

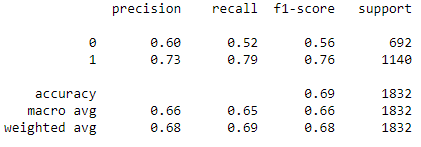
Accuracy score of Testing = 68.72270742358079

Precision score = 68.0705130823014

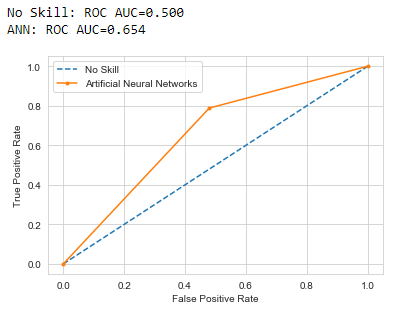
Recall score = 68.72270742358079

F1 score = 68.22229367718045

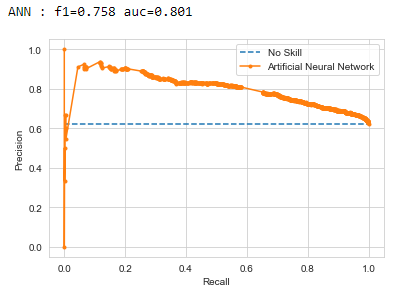
Classification report using ANN is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 29: ROC Curve for Artificial Neural Networks**



**Figure 30: Precision-Recall Curve for Artificial Neural Networks**

1. **Using Decision Tree classifier**
2. Import the package

from sklearn.tree import DecisionTreeClassifier

1. Using Train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. 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\_)

1. Predict the model

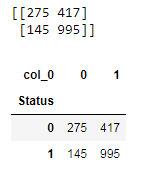
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 68.79548762736536

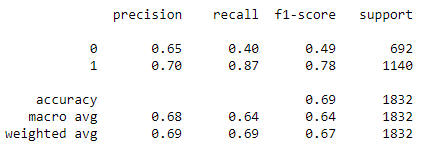
Accuracy score of Testing = 69.32314410480349

Precision score = 68.58208788172789

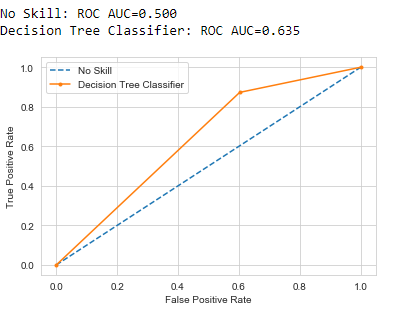
Recall score = 69.32314410480349

F1 score = 67.2061151916238

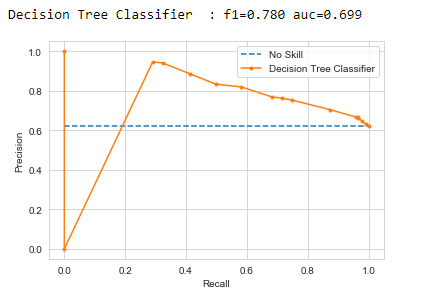
Classification report using Decision Tree Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 31: ROC Curve for Decision Tree Classifier**



**Figure 32: Precision-Recall Curve for Decision Tree Classifier**

1. **Using Logistic Regression**
2. Import the necessary package

from sklearn.linear\_model import LogisticRegression

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. Define and Train the model

model=LogisticRegression(random\_state=5,C=2.0,multi\_class='ovr')

model.fit(X\_train,y\_train)

1. Predict the model

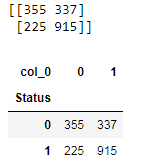
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 66.64847161572052

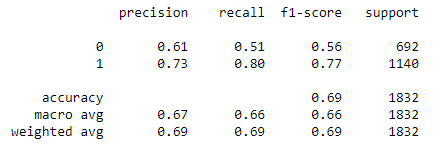
Accuracy score of Testing = 69.32314410480349

Precision score = 69.32314410480349

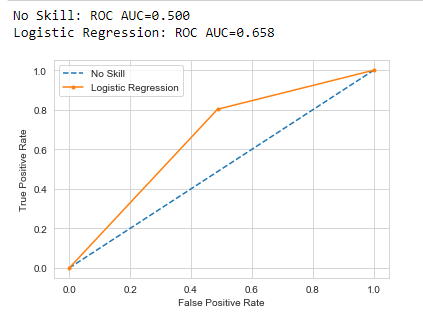
Recall score = 69.32314410480349

F1 score = 69.32314410480349

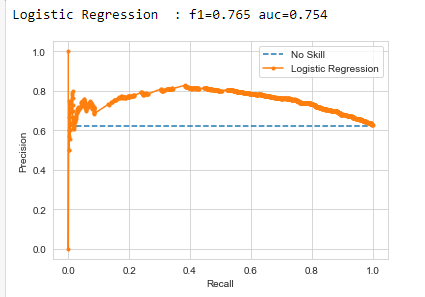
Classification report using logistic regression is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 33: ROC Curve for Logistic Regression**



**Figure 34: Precision-Recall Curve for Logistic Regression**

1. **Using K-Nearest Neighbors (KNN)**
2. Import the package

from sklearn.neighbors import KNeighborsClassifier

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. Define and Train the model

model= KNeighborsClassifier(n\_neighbors=100, metric=’minkowski’)

model.fit(X\_train,y\_train)

1. Predict the model

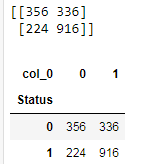
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 69.28675400291121

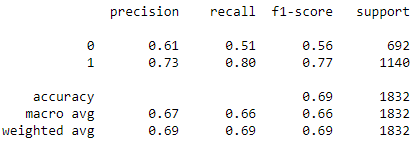
Accuracy score of Testing = 69.43231441048034

Precision score = 68.7119178806456

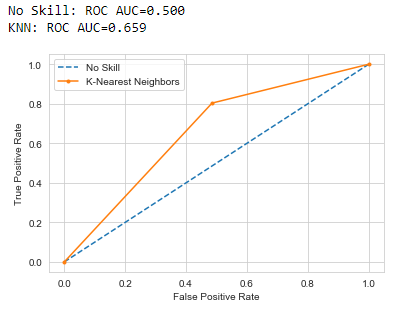
Recall score = 69.43231441048034

F1 score = 68.80219868136803

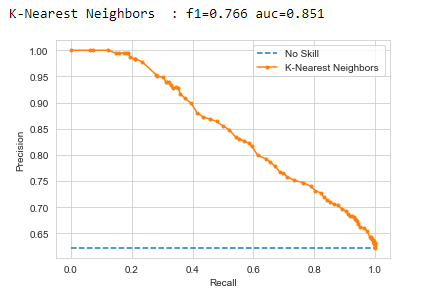
Classification report using KNN is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 35: ROC Curve for K-Nearest Neighbors**



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

1. **Using Naïve Bayes Classifier**
2. Import the package

from sklearn.naive\_bayes import BernoulliNB, GaussianNB

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. Define and Train the model for Bernoulli Naïve Bayes

model\_bernoulli = BernoulliNB(alpha=2.0)

model\_bernoulli.fit(X\_train,y\_train)

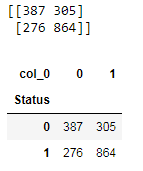
1. Predict the model

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:



We get the following performance metrics:

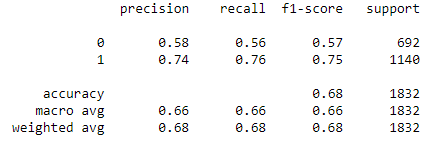
Accuracy score of Testing = 68.28602620087337

Precision score = 68.04005992300351

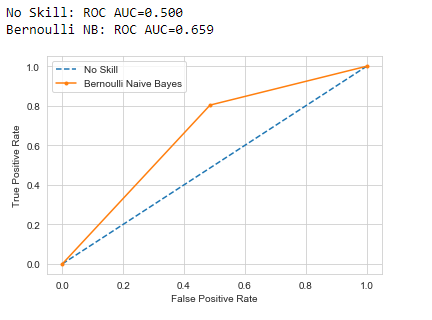
Recall score = 68.28602620087337

F1 score = 68.14580818679296

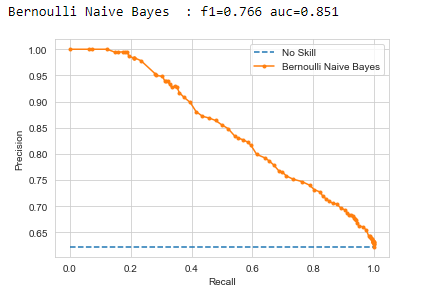
Classification report using Bernoulli Naïve Bayes is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 37: ROC Curve for Bernoulli Naïve Bayes**



**Figure 38: Precision-Recall Curve for Bernoulli Naïve Bayes**

1. Define and Train the model for Gaussian Naïve Bayes

model\_gaussian = GaussianNB()

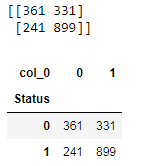
model\_gaussian.fit(X\_train,y\_train)

1. Predict the model

y\_predict\_gaussian = model\_gaussian.predict(X\_test)

print(confusion\_matrix(y\_test,y\_predict\_gaussian))

pd.crosstab(y\_test,y\_predict\_gaussian)



We get the following performance metrics:

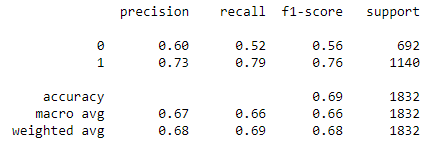
Accuracy score of Testing = 68.77729257641921

Precision score = 68.13262073487424

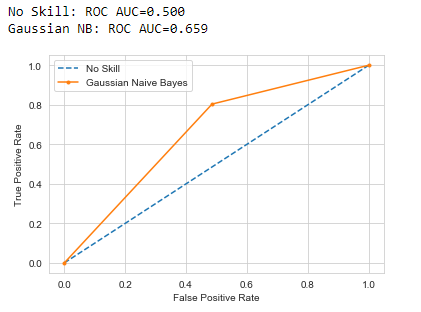
Recall score = 68.77729257641921

F1 score = 68.28433139451188

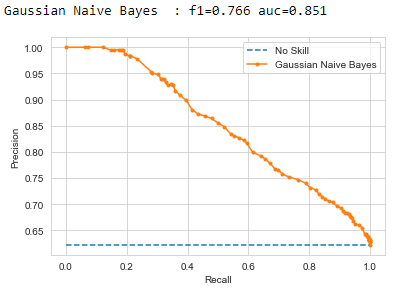
Classification report using Gaussian Naïve Bayes is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 39: ROC Curve for Gaussian Naïve Bayes**



**Figure 40: Precision-Recall Curve for Gaussian Naïve Bayes**

1. **Using ExtraTrees Classifier**
2. Import the package

from sklearn.ensemble import ExtraTreesClassifier

1. Using train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=5)

1. Define and Train the model

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\_)

1. Predict the model

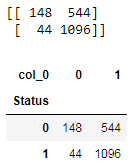
y\_train\_predict=model.predict(X\_train)

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:



We get the following performance metrics:

Accuracy score of Training = 66.41193595342067

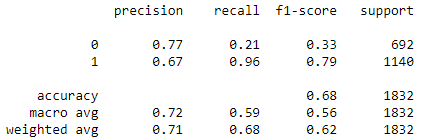
Accuracy score of Testing = 67.90393013100436

Precision score = 70.70252866829978

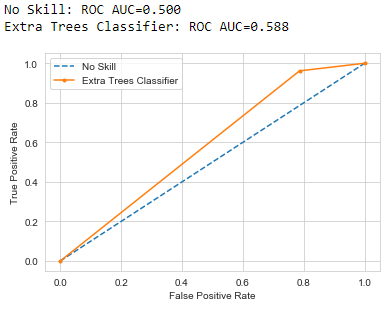
Recall score = 67.90393013100436

F1 score = 61.713324513184794

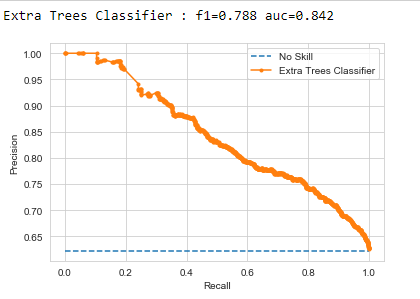
Classification report using Extra Trees Classifier is as follows:



The following are the ROC and Precision-Recall Curves.



**Figure 41: ROC Curve for Extra Trees Classifier**



**Figure 42: Precision-Recall Curve for Extra Trees Classifier**