Classification algorithm project

Algorithms in this notebook

1) Logistic Regression 2) KNN (K-nearest neighbors) 3) Decision Tree 4) Random Forest 5) SVM (Support Vector Machine) 6) Naive Bayes

In [1]:
```
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Read data from csv file
df=pd.read_csv(r"C:\Users\Sanjay Lohar\Downloads\Investment.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutc
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	...	1	999	0	nonexi
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	...	1	999	0	nonexi
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	...	3	6	2	suc
3	39	services	married	high.school	no	no	no	cellular	apr	fri	...	2	999	0	nonexi
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	...	1	3	1	suc

5 rows × 21 columns

In [4]:
```
#Target variable
df['Invested']=df['Invested'].replace(['Yes','No'],[1,0])
```

In [5]:
```
df.dtypes
```

Out[5]:
```
age                  int64
job                 object
marital             object
education           object
default             object
housing             object
loan                object
contact             object
month               object
day_of_week         object
duration             int64
campaign             int64
pdays                int64
previous             int64
poutcome            object
emp_var_rate       float64
cons_price_idx     float64
cons_conf_idx      float64
euribor3m          float64
nr_employed        float64
Invested             int64
dtype: object
```

In [6]:
```
#check missing values
#treat missing values (if any)
df.isnull().sum()
```

Out[6]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp_var_rate      0
cons_price_idx    0
cons_conf_idx     0
euribor3m         0
nr_employed       0
Invested          0
dtype: int64
```

In [7]:
```
#print column names and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp_var_rate    41188 non-null  float64
 16  cons_price_idx  41188 non-null  float64
 17  cons_conf_idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr_employed     41188 non-null  float64
 20  Invested        41188 non-null  int64  
dtypes: float64(5), int64(6), object(10)
memory usage: 6.6+ MB
```

In [8]:
```
#check unique entries in job column
df['job'].unique()
```

Out[8]:
```
array(['blue-collar', 'technician', 'management', 'services', 'retired',
       'admin.', 'housemaid', 'unemployed', 'entrepreneur',
       'self-employed', 'unknown', 'student'], dtype=object)
```

In [9]:
```
#Replace
# technician: blue-collar
# management: white-collar
# services : white-collar
# admin. : white-collar
# housemaid : blue-collar
# entrepreneur : white-collar
# self-employed: white-collar
# unknown : unemployed
# student : unemployed
# Retired : Retired
```

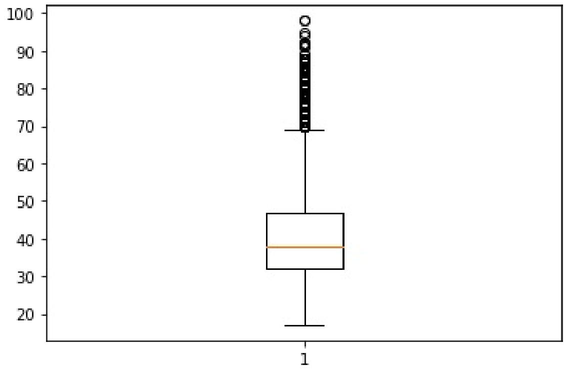
In [10]:
```
df['job']=df['job'].replace(['technician','housemaid',
                                           'management','services','admin.',
                                           'entrepreneur','self-employed',
                                           'unknown','student'],
                                          ['blue-collar','blue-collar',
                                           'white-collar','white-collar',
                                           'white-collar','white-collar',
                                           'white-collar','unemployed',
                                           'unemployed'])
```

In [11]:
```
#Check marital status
df['marital'].unique()
```

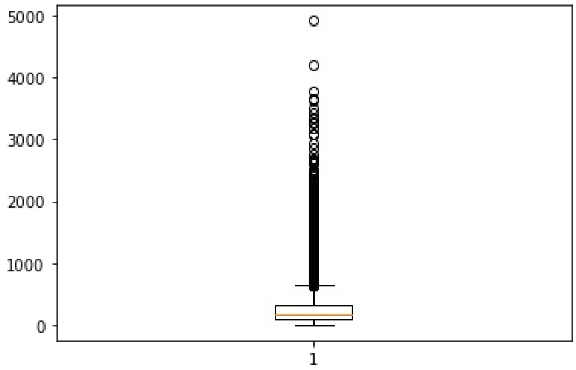
Out[11]:
```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
```

In [12]:
```
df['marital']=df['marital'].replace(['divorced','unknown'],['single','marital_unknown'])
df['marital'].unique()
```
Out[12]:
```
array(['married', 'single', 'marital_unknown'], dtype=object)
```
In [13]:
```
#Check unique entries in education column
df['education'].unique()
```
Out[13]:
```
array(['basic.4y', 'unknown', 'university.degree', 'high.school',
       'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
      dtype=object)
```
In [14]:
```
#Replace
#basic.4y. basic.6y,basic.9y with basic
#unknown with edu_unknown
df['education']=df['education'].replace(['basic.4y','basic.6y','basic.9y'],
                                        ['basic','basic','basic'])
df['education']=df['education'].replace(['unknown'],
                                        ['edu_unknown'])
```
In [15]:
```
#Check unique entries in education column
df['education'].unique()
```
Out[15]:
```
array(['basic', 'edu_unknown', 'university.degree', 'high.school',
       'professional.course', 'illiterate'], dtype=object)
```
In [16]:
```
df['default'].unique()
df['default']=df['default'].replace(['unknown','yes','no'],
                          ['default_unknown','default_yes','default_no'])
```
In [17]:
```
df['default'].unique()
```
Out[17]:
```
array(['default_unknown', 'default_no', 'default_yes'], dtype=object)
```
In [18]:
```
df['housing'].unique()
```
Out[18]:
```
array(['yes', 'no', 'unknown'], dtype=object)
```
In [19]:
```
df['housing']=df['housing'].replace(['unknown','yes','no'],
                            ['housing_unknown','housing_yes','housing_no'])
df['housing'].unique()
```
Out[19]:
```
array(['housing_yes', 'housing_no', 'housing_unknown'], dtype=object)
```
In [20]:
```
df['loan'].unique()
df['loan']=df['loan'].replace(['unknown','yes','no'],
                                ['loan_unknown','loan_yes','loan_no'])
df['loan'].unique()
```
Out[20]:
```
array(['loan_no', 'loan_yes', 'loan_unknown'], dtype=object)
```
In [21]:
```
df['contact'].unique()
```
Out[21]:
```
array(['cellular', 'telephone'], dtype=object)
```
In [22]:
```
df['poutcome'].unique()
```
Out[22]:
```
array(['nonexistent', 'success', 'failure'], dtype=object)
```
In [23]:
```
#Check missing values
df.isnull().sum()
```
Out[23]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp_var_rate      0
cons_price_idx    0
cons_conf_idx     0
euribor3m         0
nr_employed       0
Invested          0
dtype: int64
```

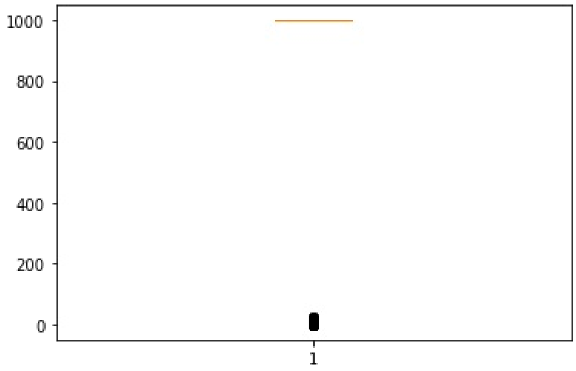
In [24]:
```
#Check outliers
plt.boxplot(df['age']) #has outlier
plt.show()
```



In [25]:
```
#Check outliers
plt.boxplot(df['duration']) #has outlier
plt.show()
```



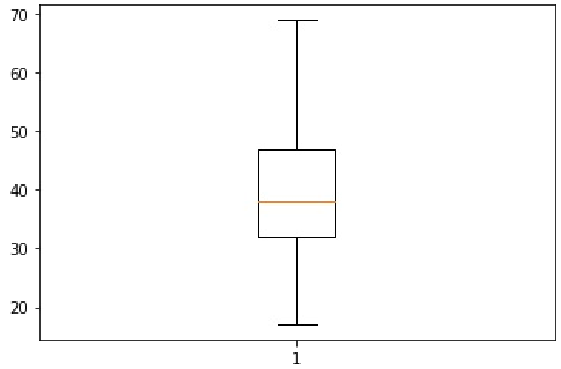
In [26]:
```
#Check outliers
plt.boxplot(df['pdays']) #has outlier
plt.show()
```



In [30]:
```
#remove outliers
#user defined function for outlier treatment
def rm_out(d,c):
    #find q1 and q3
    q1=d[c].quantile(0.25)
    q3=d[c].quantile(0.75)

    #find iqr
    iqr=q3-q1

    #upper bound and lower bound
    ub=q3+1.5*iqr
    lb=q1-1.5*iqr

    output_data=d.loc[(d[c]>lb) & (d[c]<ub)]

    return output_data
```

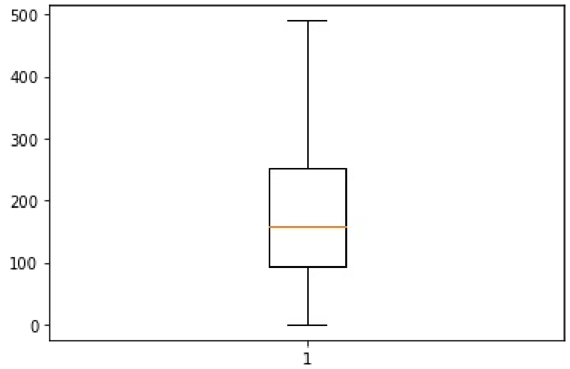
In [35]:
```
#Outlier treatment: age column
df=rm_out(df,'age')
plt.boxplot(df['age'])
```

Out[35]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x22dbe480610>,
  <matplotlib.lines.Line2D at 0x22dbe4808e0>],
 'caps': [<matplotlib.lines.Line2D at 0x22dbe480bb0>,
  <matplotlib.lines.Line2D at 0x22dbe480e80>],
 'boxes': [<matplotlib.lines.Line2D at 0x22dbe480340>],
 'medians': [<matplotlib.lines.Line2D at 0x22dbe48d190>],
 'fliers': [<matplotlib.lines.Line2D at 0x22dbe48d460>],
 'means': []}
```



In [45]:
```
#Outlier treatment: duration column
df=rm_out(df,'duration')
plt.boxplot(df['duration'])
```

Out[45]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x22dbe65ad00>,
  <matplotlib.lines.Line2D at 0x22dbe65afd0>],
 'caps': [<matplotlib.lines.Line2D at 0x22dbe6692e0>,
  <matplotlib.lines.Line2D at 0x22dbe6695b0>],
 'boxes': [<matplotlib.lines.Line2D at 0x22dbe65aa30>],
 'medians': [<matplotlib.lines.Line2D at 0x22dbe669880>],
 'fliers': [<matplotlib.lines.Line2D at 0x22dbe669b50>],
 'means': []}
```



In [48]:
```
#Replace pdays 999 with NaN (missing values)
df['pdays']=df['pdays'].replace(999,np.nan)
```

In [49]:
```
#Replace pdays  NaN (missing values) with median
df['pdays']=df['pdays'].fillna(df['pdays'].median())
```

In [50]:
```
df.columns
```

Out[50]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
       'cons_conf_idx', 'euribor3m', 'nr_employed', 'Invested'],
      dtype='object')
```

In [51]:
```
# split data into x and y
x = df.loc[:, ['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
       'cons_conf_idx', 'euribor3m', 'nr_employed']]

y = df['Invested']
```

In [52]:
```
# create dummy variables for categorical variables

# subset all categorical variables
df_categorical = x.select_dtypes(include=['object'])
df_categorical.head()
```

Out[52]:

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome
0	blue-collar	married	basic	default_unknown	housing_yes	loan_no	cellular	aug	thu	nonexistent
1	blue-collar	married	edu_unknown	default_no	housing_no	loan_no	cellular	nov	fri	nonexistent
2	white-collar	single	university.degree	default_no	housing_yes	loan_no	cellular	jun	thu	success
3	white-collar	married	high.school	default_no	housing_no	loan_no	cellular	apr	fri	nonexistent
4	retired	married	basic	default_no	housing_yes	loan_no	cellular	aug	fri	success

In [53]:
```
# convert into dummies
df_dummies = pd.get_dummies(df_categorical)
df_dummies.head()
```

Out[53]:

	job_blue-collar	job_retired	job_unemployed	job_white-collar	marital_marital_unknown	marital_married	marital_single	education_basic	education_ed
0	1	0	0	0	0	1	0	1	
1	1	0	0	0	0	1	0	0	
2	0	0	0	1	0	0	1	0	
3	0	0	0	1	0	1	0	0	
4	0	1	0	0	0	1	0	1	

5 rows × 42 columns

In [54]:
```
# drop categorical from x
x = x.drop(list(df_categorical.columns), axis=1)
```

In [55]:
```
# concat dummy variables with x
x = pd.concat([x, df_dummies], axis=1)
x
```

Out[55]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	...	month_oct	mo
0	44	210	1	6.0	0	1.4	93.444	-36.1	4.963	5228.1	...	0	
1	53	138	1	6.0	0	-0.1	93.200	-42.0	4.021	5195.8	...	0	
2	28	339	3	6.0	2	-1.7	94.055	-39.8	0.729	4991.6	...	0	
3	39	185	2	6.0	0	-1.8	93.075	-47.1	1.405	5099.1	...	0	
4	55	137	1	3.0	1	-2.9	92.201	-31.4	0.869	5076.2	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	59	222	1	6.0	0	1.4	94.465	-41.8	4.866	5228.1	...	0	
41184	31	196	2	6.0	0	1.1	93.994	-36.4	4.860	5191.0	...	0	
41185	42	62	3	6.0	0	1.1	93.994	-36.4	4.857	5191.0	...	0	
41186	48	200	2	6.0	0	-3.4	92.431	-26.9	0.742	5017.5	...	1	
41187	25	112	4	6.0	0	1.1	93.994	-36.4	4.859	5191.0	...	0	

35688 rows × 52 columns

In [56]:
```
# split data into train and test
from sklearn.model_selection import train_test_split

xtrain,xtest, ytrain, ytest = train_test_split(x, y,train_size=0.7,test_size = 0.3,
                                               random_state=0)
```

In [57]:
```
#-------------- Model Development ---------------
```

--------------- Logistic Regression ---------------

In [58]:
```
#Import Logistic Regression Library
from sklearn.linear_model import LogisticRegression
```

In [59]:
```
#create  model object
logreg=LogisticRegression()
```

In [60]:
```
#Fit the model using training data
train_model_fit=logreg.fit(xtrain,ytrain)
```

In [61]:
```
#check the accuracy of training model
logreg.score(xtrain,ytrain)
```

Out[61]: 0.9465994155558224

In [62]:
```
#predict y using test data
y_pred=logreg.predict(xtest)
```

In [63]:
```
#check the accuracy of test model
logreg.score(xtest,ytest)
```

Out[63]: 0.9518072289156626

In [64]:
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
acc = accuracy_score(ytest, y_pred)
prec = precision_score(ytest, y_pred)
rec = recall_score(ytest, y_pred)
f1 = f1_score(ytest,y_pred)

print('Accuracy:', acc,'\nPrecision:', prec,'\nRecall:',rec, '\nF1 Score:',f1)

results = pd.DataFrame([['Logistic Regression', acc,prec,rec,f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

```
Accuracy: 0.9518072289156626
Precision: 0.6782178217821783
Recall: 0.41515151515151516
F1 Score: 0.5150375939849625
```

Out[64]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.951807	0.678218	0.415152	0.515038

In [65]:
```
prec = precision_score(ytest, y_pred)
prec
```

Out[65]: 0.6782178217821783

In [66]:
```
#import confusion matrix library
from sklearn.metrics import confusion_matrix
```

In [67]:
```
#create confusion matrix
confusion_matrix(ytest,y_pred)
```

Out[67]:
```
array([[9917,  130],
       [ 386,  274]], dtype=int64)
```

In [68]:
```
#Precision: Out of all predicted positives, how many are really positive
#Precision=TP/(TP+FP)
274/(274+130)
```

Out[68]: 0.6782178217821783

In [69]:
```
#Accuracy   =(TP+TN)/(TP+TN+FN+FP)
(9917+274)/(9917+274+386+130)
```

Out[69]: 0.9518072289156626

In [70]:
```
#TPR or sensitivity or recall  TPR=TP/TP+FN
274/(386+274) #sensitivity
```

Out[70]: 0.41515151515151516

In [71]:
```
#TNR or Specificity or selectivity  TNR =TN/TN+FP
9917/(9917+ 130)
```

Out[71]: 0.9870608141733851

-------------------------KNN ----------------------

In [72]:
```
#import knn library from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

In [73]:
```
#Create model object
knn=KNeighborsClassifier(n_neighbors=5, metric='euclidean')
```

In [74]:
```
#fit the training model
train_model_fit=knn.fit(xtrain,ytrain)
```

In [75]:
```
#test the accuracy of training model
knn.score(xtrain,ytrain)
```

Out[75]: 0.9559265041431488

In [76]:
```
#test the model using xtest
y_pred=knn.predict(xtest)
```

In [77]:
```
#Check accuracy of test model
knn.score(xtest,ytest)
```

Out[77]: 0.9462034183244606

In [78]:
```
#import confusion matrix from sklearn
from sklearn.metrics import confusion_matrix
```

In [79]:
```
confusion_matrix(ytest,y_pred)
```

Out[79]:
```
array([[9853,  194],
       [ 382,  278]], dtype=int64)
```

In [80]:
```
#TPR or Sensitivity or recall
278/(382+ 278)
```

Out[80]: 0.4212121212121212

In [81]:
```
#TNR or Specificity or Selectivity
9852/(9852+ 195)
```

Out[81]: 0.9805912212600776

In [82]:
```
#Calculate Recall, Precision, Sensitivity
knn_acc=(9854+279)/(9854+279+193+381)
knn_rec=279/(279+381)
knn_prec=(279/(279+193))
knn_spec=9854/(9854+193)
knn_f1=2*(knn_prec*knn_rec)/(knn_prec + knn_rec)
```

In [83]:
```
results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[83]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.951807	0.678218	0.415152	0.515038
1	KNN	0.946390	0.591102	0.422727	0.492933

In [ ]:

In [84]:
```
# predict probabilities
pred_prob1 = logreg.predict_proba(xtest)
pred_prob2 = knn.predict_proba(xtest)
```

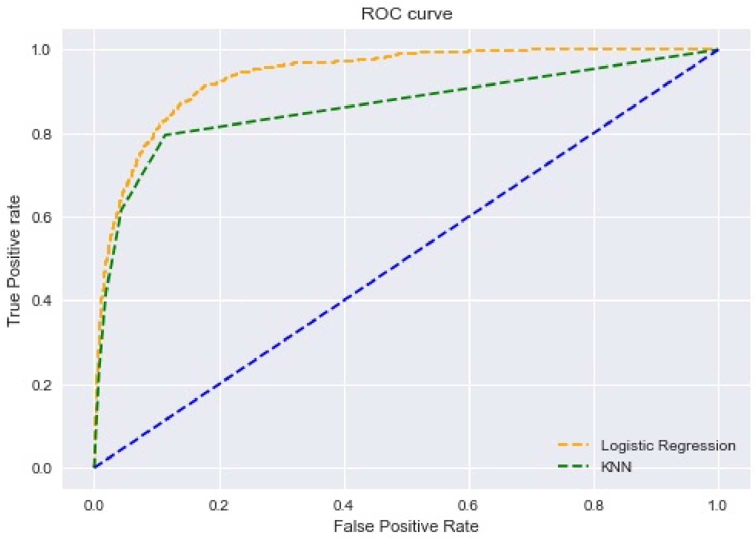
In [85]:
```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(ytest, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(ytest, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(ytest))]
p_fpr, p_tpr, _ = roc_curve(ytest, random_probs, pos_label=1)
```

In [86]:
```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(ytest, pred_prob1[:,1])
auc_score2 = roc_auc_score(ytest, pred_prob2[:,1])

print(auc_score1, auc_score2)
```

0.937673992839714 0.8623289478843376

In [87]:
```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
```

```python
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC')
plt.show()
```



-------------- Decision Tree ------------------

In [105…
```python
#import Decision Tree classifier library from sklearn
from sklearn.tree import DecisionTreeClassifier
```

In [106…
```python
#Create model object
dtree=DecisionTreeClassifier(max_depth=5)
```

In [107…
```python
#fit the training model
model=dtree.fit(xtrain,ytrain)
```

In [108…
```python
#test the accuracy of training model
dtree.score(xtrain,ytrain)
```

Out[108]:
```
0.9525639486009367
```

In [109…
```python
#test the model using xtest
y_pred=dtree.predict(xtest)
```

In [110…
```python
#Check accuracy ot test model
dtree.score(xtest,ytest)
```

Out[110]:
```
0.9537685626225834
```

In [111…
```python
#import confusion matrix from sklearn
from sklearn.metrics import confusion_matrix
```

In [112…
```python
confusion_matrix(ytest,y_pred)
```

Out[112]:
```
array([[9813,  234],
       [ 261,  399]], dtype=int64)
```

In [113…
```python
#TPR or Sensitivity or recall
399/(261+399)
```

Out[113]:
```
0.6045454545454545
```

In [114…
```python
#TNR Spcificity or selectivity
9813/(9813+234)
```

Out[114]:
```
0.9767094655120931
```

In [115…
```python
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
dt_acc = accuracy_score(ytest, y_pred)
dt_prec = precision_score(ytest, y_pred)
dt_rec = recall_score(ytest, y_pred)
dt_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1]],
```

```
             columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[115]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.951807	0.678218	0.415152	0.515038
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.881760	0.312268	0.763636	0.443272
3	D-Tree	0.953769	0.630332	0.604545	0.617169

------------------- Random FOrest -------------------------

In [ ]:

In [116…
```
#import Random Forest classifier library from sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
```

In [137…
```
#Create Model Object for Random Forest
rfc = RandomForestClassifier(random_state=101)

#Recursive Feature Elimination with Cross-Validation
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(7),
              scoring='accuracy')
rfecv
```

Out[137]:
```
RFECV(cv=StratifiedKFold(n_splits=7, random_state=None, shuffle=False),
      estimator=RandomForestClassifier(random_state=101), scoring='accuracy')
```

In [138…
```
#fit the training model
model=rfc.fit(xtrain,ytrain)
```

In [139…
```
#test the accuracy of training model
rfc.score(xtrain,ytrain)
```

Out[139]:
```
1.0
```

In [140…
```
#test the model using xtest
y_pred=rfc.predict(xtest)
```

In [141…
```
#Check accuracy ot test model
rfc.score(xtest,ytest)
```

Out[141]:
```
0.9545157373680769
```

In [142…
```
#import confusion matrix from sklearn
from sklearn.metrics import confusion_matrix
```

In [143…
```
confusion_matrix(ytest,y_pred)
```

Out[143]:
```
array([[9887,  160],
       [ 327,  333]], dtype=int64)
```

In [144…
```
#TPR or Sensitivity or recall
334/(334+326)
```

Out[144]:
```
0.5060606060606061
```

In [145…
```
#TNR Spcificity or selectivity
9885/(9813+162)
```

Out[145]:
```
0.9909774436090225
```

In [146…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
rf_acc = accuracy_score(ytest, y_pred)
rf_prec = precision_score(ytest, y_pred)
rf_rec = recall_score(ytest, y_pred)
rf_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],
                       ['Randm Forest', rf_acc,rf_prec,rf_rec,rf_f1]],
             columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[146]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.951807	0.678218	0.415152	0.515038
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.881760	0.312268	0.763636	0.443272
3	D-Tree	0.953769	0.630332	0.604545	0.617169
4	Randm Forest	0.954516	0.675456	0.504545	0.577624

-------Support Vector Machine (SVM) --------

In [127…
```
#Support Vector Machine (SVC: Support Vector Classifier)
from sklearn.svm import SVC
svm = SVC(kernel='linear')
```

In [128…
```
#Fit the model
model=svm.fit(xtrain,ytrain)
```

In [129…
```
#check accuracy of training model
svm.score(xtrain,ytrain)
```

Out[129]:
```
0.9360313838517273
```

In [130…
```
#Predict the response from xtest
y_pred=svm.predict(xtest)
```

In [131…
```
#Check accuracy of test model
svm.score(xtest,ytest)
```

Out[131]:
```
0.9343420192397497
```

In [132…
```
#import confusion matrix from sklearn
from sklearn.metrics import confusion_matrix
```

In [133…
```
confusion_matrix(ytest,y_pred)
```

Out[133]:
```
array([[9698,  349],
       [ 354,  306]], dtype=int64)
```

In [134…
```
#Sensitivity
306/(354+306)
```

Out[134]:
```
0.4636363636363636
```

In [135…
```
#Specificity
9718/(9718+329)
```

Out[135]:
```
0.9672539066387976
```

In [136…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
svm_acc = accuracy_score(ytest, y_pred)
svm_prec = precision_score(ytest, y_pred)
svm_rec = recall_score(ytest, y_pred)
svm_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],
                       ['Randm Forest', rf_acc,rf_prec,rf_rec,rf_f1],
                       ['SVM', svm_acc,svm_prec,svm_rec,svm_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[136]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.951807	0.678218	0.415152	0.515038
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.881760	0.312268	0.763636	0.443272
3	D-Tree	0.953769	0.630332	0.604545	0.617169
4	Randm Forest	0.954516	0.675456	0.504545	0.577624
5	SVM	0.934342	0.467176	0.463636	0.465399

--------------- Naive Bayes Classifier --------------

In [99]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

# Train the model using the training sets
gnb.fit(xtrain,ytrain)

#Predict Output
y_pred = gnb.predict(xtest)
```

In [100…
```
#Check accuracy ot test model

#use model.score(xtest,ytest)
gnb.score(xtest,ytest)
#or
# Model Accuracy, how often is the classifier correct?
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(ytest, y_pred))
```

```
Accuracy: 0.8817595965256374
```

In [101…
```
#TPR
from sklearn.metrics import confusion_matrix

confusion_matrix(ytest,y_pred)
#TPR=0.85
```

Out[101]:
```
array([[8937, 1110],
       [ 156,  504]], dtype=int64)
```

In [102…
```
#sensitivity
504/(156+504)
```

Out[102]:
```
0.7636363636363637
```

In [103…
```
#specificity
8937/(8937+1110)
```

Out[103]:
```
0.889519259480442
```

In [104…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
nb_acc = accuracy_score(ytest, y_pred)
nb_prec = precision_score(ytest, y_pred)
nb_rec = recall_score(ytest, y_pred)
nb_f1 = f1_score(ytest,y_pred)

#print('Accuracy:', nb_acc,'\nPrecision:', nb_prec,'\nRecall:',nb_rec, '\nF1 Score:',nb_f1)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[104]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.951807	0.678218	0.415152	0.515038
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.881760	0.312268	0.763636	0.443272

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js