

```

import numpy as np
import pandas as pd

## importing important libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings

warnings.filterwarnings("ignore")

%matplotlib inline

df = pd.read_csv("Travel.csv")
df.head()

```

	CustomerID	ProdTaken	Age	TypeofContact	CityTier
DurationOfPitch \					
0	200000	1	41.0	Self Enquiry	3
6.0					
1	200001	0	49.0	Company Invited	1
14.0					
2	200002	1	37.0	Self Enquiry	1
8.0					
3	200003	0	33.0	Company Invited	1
9.0					
4	200004	0	NaN	Self Enquiry	1
8.0					

	Occupation	Gender	NumberOfPersonVisiting
NumberOfFollowups \			
0	Salaried	Female	3
			3.0
1	Salaried	Male	3
			4.0
2	Free Lancer	Male	3
			4.0
3	Salaried	Female	2
			3.0
4	Small Business	Male	2
			3.0

	ProductPitched	PreferredPropertyStar	MaritalStatus
NumberOfTrips \			
0	Deluxe	3.0	Single
			1.0
1	Deluxe	4.0	Divorced
			2.0

2	Basic	3.0	Single	7.0
3	Basic	3.0	Divorced	2.0
4	Basic	4.0	Divorced	1.0

	Passport	PitchSatisfactionScore	OwnCar	NumberOfChildrenVisiting
0	1	2	1	0.0
1	0	3	1	2.0
2	1	3	0	0.0
3	1	5	1	1.0
4	0	5	1	0.0

	Designation	MonthlyIncome
0	Manager	20993.0
1	Manager	20130.0
2	Executive	17090.0
3	Executive	17909.0
4	Executive	18468.0

```
# Data Cleaning
#Handling Missing values
#Handling Missing values
#Handling Duplicates
#Check data type
#Understand the dataset
df.isnull().sum()
```

CustomerID	0
ProdTaken	0
Age	226
TypeofContact	25
CityTier	0
DurationOfPitch	251
Occupation	0
Gender	0
NumberOfPersonVisiting	0
NumberOfFollowups	45
ProductPitched	0
PreferredPropertyStar	26
MaritalStatus	0
NumberOfTrips	140
Passport	0
PitchSatisfactionScore	0

```
OwnCar                0
NumberOfChildrenVisiting  66
Designation           0
MonthlyIncome         233
dtype: int64
```

Check all the categories

```
df['Gender'].value_counts()
```

```
Gender
Male      2916
Female    1817
Fe Male    155
Name: count, dtype: int64
```

```
df['MaritalStatus'].value_counts()
```

```
MaritalStatus
Married      2340
Divorced      950
Single        916
Unmarried     682
Name: count, dtype: int64
```

```
df['MaritalStatus'].value_counts()
```

```
MaritalStatus
Married      2340
Divorced      950
Single        916
Unmarried     682
Name: count, dtype: int64
```

```
df['TypeofContact'].value_counts()
```

```
TypeofContact
Self Enquiry      3444
Company Invited    1419
Name: count, dtype: int64
```

```
df['Gender'] = df['Gender'].replace('Fe Male', 'Female')
df['MaritalStatus'] = df['MaritalStatus'].replace('Single',
'Unmarried')
```

Check all the categories

```
df['Gender'].value_counts()
```

```
Gender
Male      2916
Female    1972
Name: count, dtype: int64
```

df.head()

	CustomerID	ProdTaken	Age	TypeofContact	CityTier
0	200000	1	41.0	Self Enquiry	3
1	200001	0	49.0	Company Invited	1
2	200002	1	37.0	Self Enquiry	1
3	200003	0	33.0	Company Invited	1
4	200004	0	NaN	Self Enquiry	1

	Occupation	Gender	NumberOfPersonVisiting
0	Salaried	Female	3
1	Salaried	Male	3
2	Free Lancer	Male	3
3	Salaried	Female	2
4	Small Business	Male	2

	ProductPitched	PreferredPropertyStar	MaritalStatus
0	Deluxe	3.0	Unmarried
1	Deluxe	4.0	Divorced
2	Basic	3.0	Unmarried
3	Basic	3.0	Divorced
4	Basic	4.0	Divorced

	Passport	PitchSatisfactionScore	OwnCar	NumberOfChildrenVisiting
0	1	2	1	0.0
1	0	3	1	2.0
2	1	3	0	0.0
3	1	5	1	1.0

4	0	5	1	0.0
---	---	---	---	-----

	Designation	MonthlyIncome
0	Manager	20993.0
1	Manager	20130.0
2	Executive	17090.0
3	Executive	17909.0
4	Executive	18468.0

Check Misssing Values

```
##these are the features with nan value
```

```
features_with_na=[features for features in df.columns if
df[features].isnull().sum()>=1]
```

```
for feature in features with na:
```

```
print(feature,np.round(df[feature].isnull().mean()*100,5), '%  
missing values')
```

Age 4.62357 % missing values

TypeofContact 0.51146 % missing values

```
DurationOfPitch 5.13502 % missing values
```

NumberOfFollowups 0.92062 % missing values

PreferredPropertyStar 0.53191 % missing values

NumberOfTrips 2.86416 % missing values

NumberOfChildrenVisiting 1.35025 % missing values

MonthlyIncome 4.76678 % missing values

```
# statistics on numerical columns (Null cols)
```

```
df[features with na].select dtypes(exclude='object').describe()
```

	Age	DurationOfPitch	NumberOfFollowups
PreferredPropertyStar \ count	4662.000000	4637.000000	4843.000000
mean	37.622265	15.490835	3.708445
std	9.316387	8.519643	1.002509
min	18.000000	5.000000	1.000000
25%	31.000000	9.000000	3.000000
50%	36.000000	13.000000	4.000000
75%	44.000000	20.000000	4.000000
max	61.000000	127.000000	6.000000

NumberOfTrips NumberOfChildrenVisiting MonthlyIncome

count	4748.000000	4822.000000	4655.000000
mean	3.236521	1.187267	23619.853491
std	1.849019	0.857861	5380.698361
min	1.000000	0.000000	1000.000000
25%	2.000000	1.000000	20346.000000
50%	3.000000	1.000000	22347.000000
75%	4.000000	2.000000	25571.000000
max	22.000000	3.000000	98678.000000

#Age

```
df.Age.fillna(df.Age.median(), inplace=True)
```

#TypeofContract

```
df.TypeofContract.fillna(df.TypeofContract.mode()[0], inplace=True)
```

#DurationOfPitch

```
df.DurationOfPitch.fillna(df.DurationOfPitch.median(), inplace=True)
```

#NumberOfFollowups

```
df.NumberOfFollowups.fillna(df.NumberOfFollowups.mode()[0],
inplace=True)
```

#PreferredPropertyStar

```
df.PreferredPropertyStar.fillna(df.PreferredPropertyStar.mode()[0],
inplace=True)
```

#NumberOfTrips

```
df.NumberOfTrips.fillna(df.NumberOfTrips.median(), inplace=True)
```

#NumberOfChildrenVisiting

```
df.NumberOfChildrenVisiting.fillna(df.NumberOfChildrenVisiting.mode()
[0], inplace=True)
```

#MonthlyIncome

```
df.MonthlyIncome.fillna(df.MonthlyIncome.median(), inplace=True)
```

```
df.head()
```

```
df.isnull().sum()
```

CustomerID	0
ProdTaken	0
Age	0
TypeofContract	0
CityTier	0
DurationOfPitch	0
Occupation	0
Gender	0
NumberOfPersonVisiting	0
NumberOfFollowups	0
ProductPitched	0
PreferredPropertyStar	0

```

MaritalStatus      0
NumberOfTrips      0
Passport           0
PitchSatisfactionScore  0
OwnCar             0
NumberOfChildrenVisiting  0
Designation        0
MonthlyIncome      0
dtype: int64

```

```
df.drop('CustomerID', inplace=True, axis=1)
```

```
# create new column for feature
```

```
df['TotalVisiting'] = df['NumberOfPersonVisiting'] +
```

```
df['NumberOfChildrenVisiting']
```

```
df.drop(columns=['NumberOfPersonVisiting',
'NumberOfChildrenVisiting'], axis=1, inplace=True)
```

```
## get all the numeric features
```

```
num_features = [feature for feature in df.columns if df[feature].dtype
!= '0']
```

```
print('Num of Numerical Features :', len(num_features))
```

```
##categorical features
```

```
cat_features = [feature for feature in df.columns if df[feature].dtype
== '0']
```

```
print('Num of Categorical Features :', len(cat_features))
```

```
## Discrete features
```

```
discrete_features=[feature for feature in num_features if
len(df[feature].unique())<=25]
```

```
print('Num of Discrete Features :',len(discrete_features))
```

```
## coontinuous features
```

```
continuous_features=[feature for feature in num_features if feature
not in discrete_features]
```

```
print('Num of Continuous Features :',len(continuous_features))
```

```
Num of Numerical Features : 12
```

```
Num of Categorical Features : 6
```

```
Num of Discrete Features : 9
```

```
Num of Continuous Features : 3
```

```
df.head()
```

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	\
0	1	41.0	Self Enquiry	3	6.0	
1	0	49.0	Company Invited	1	14.0	
2	1	37.0	Self Enquiry	1	8.0	
3	0	33.0	Company Invited	1	9.0	
4	0	36.0	Self Enquiry	1	8.0	

	Occupation	Gender	NumberOfFollowups	ProductPitched	\
0	Salaried	Female	3.0	Deluxe	

1	Salaried	Male	4.0	Deluxe
2	Free Lancer	Male	4.0	Basic
3	Salaried	Female	3.0	Basic
4	Small Business	Male	3.0	Basic

	PreferredPropertyStar	MaritalStatus	NumberOfTrips	Passport \
0	3.0	Unmarried	1.0	1
1	4.0	Divorced	2.0	0
2	3.0	Unmarried	7.0	1
3	3.0	Divorced	2.0	1
4	4.0	Divorced	1.0	0

	PitchSatisfactionScore	OwnCar	Designation	MonthlyIncome
TotalVisiting				
0	2	1	Manager	20993.0
3.0				
1	3	1	Manager	20130.0
5.0				
2	3	0	Executive	17090.0
3.0				
3	5	1	Executive	17909.0
3.0				
4	5	1	Executive	18468.0
2.0				

#Train Test Split And Model Training

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop(['ProdTaken'], axis=1)
```

```
y = df['ProdTaken']
```

```
y.value_counts()
```

```
ProdTaken
```

```
0    3968
```

```
1     920
```

```
Name: count, dtype: int64
```

```
X.head()
```

	Age	TypeofContact	CityTier	DurationOfPitch	Occupation
Gender \					
0	41.0	Self Enquiry	3	6.0	Salaried
Female					
1	49.0	Company Invited	1	14.0	Salaried
Male					
2	37.0	Self Enquiry	1	8.0	Free Lancer
Male					
3	33.0	Company Invited	1	9.0	Salaried
Female					
4	36.0	Self Enquiry	1	8.0	Small Business
Male					

	NumberOfFollowups	ProductPitched	PreferredPropertyStar
0	3.0	Deluxe	3.0
1	4.0	Deluxe	4.0
2	4.0	Basic	3.0
3	3.0	Basic	3.0
4	3.0	Basic	4.0

	NumberOfTrips	Passport	PitchSatisfactionScore	OwnCar	Designation
0	1.0	1	2	1	Manager
1	2.0	0	3	1	Manager
2	7.0	1	3	0	Executive
3	2.0	1	5	1	Executive
4	1.0	0	5	1	Executive

	MonthlyIncome	TotalVisiting
0	20993.0	3.0
1	20130.0	5.0
2	17090.0	3.0
3	17909.0	3.0
4	18468.0	2.0

separate dataset into train and test

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape
```

```
((3910, 17), (978, 17))
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 4888 entries, 0 to 4887
```

```
Data columns (total 17 columns):
```

#	Column	Non-Null Count	Dtype
0	Age	4888 non-null	float64
1	TypeofContact	4888 non-null	object

2	CityTier	4888	non-null	int64
3	DurationOfPitch	4888	non-null	float64
4	Occupation	4888	non-null	object
5	Gender	4888	non-null	object
6	NumberOfFollowups	4888	non-null	float64
7	ProductPitched	4888	non-null	object
8	PreferredPropertyStar	4888	non-null	float64
9	MaritalStatus	4888	non-null	object
10	NumberOfTrips	4888	non-null	float64
11	Passport	4888	non-null	int64
12	PitchSatisfactionScore	4888	non-null	int64
13	OwnCar	4888	non-null	int64
14	Designation	4888	non-null	object
15	MonthlyIncome	4888	non-null	float64
16	TotalVisiting	4888	non-null	float64

dtypes: float64(7), int64(4), object(6)

memory usage: 649.3+ KB

Create Column Transformer with 3 types of transformers

cat_features = X.select_dtypes(include="object").columns

num_features = X.select_dtypes(exclude="object").columns

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()

oh_transformer = OneHotEncoder(drop='first')

preprocessor = ColumnTransformer(

```
[
    ("OneHotEncoder", oh_transformer, cat_features),
    ("StandardScaler", numeric_transformer, num_features)
]
```

preprocessor

ColumnTransformer(transformers=[('OneHotEncoder',

OneHotEncoder(drop='first'),

Index(['TypeofContact', 'Occupation',

'Gender', 'ProductPitched',

'MaritalStatus', 'Designation'],

dtype='object')),

(('StandardScaler', StandardScaler(),

Index(['Age', 'CityTier',

'DurationOfPitch', 'NumberOfFollowups',

'PreferredPropertyStar', 'NumberOfTrips', 'Passport',

'PitchSatisfactionScore', 'OwnCar', 'MonthlyIncome',

'TotalVisiting'],

dtype='object'))])

```
## applying Trnsformation in training(fit_transform)
```

```
X_train=preprocessor.fit_transform(X_train)
```

```
pd.DataFrame(X_train)
```

	0	1	2	3	4	5	6	7	8	9	...	16
\												
0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	-0.721400
2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
3	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	...	-0.721400
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
...
3905	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
3906	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.455047
3907	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.455047
3908	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	...	1.455047
3909	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
	17	18	19	20	21	22						
23 \												
0	-1.020350	1.284279	-0.725271	-0.127737	-0.632399	0.679690						
0.782966												
1	0.690023	0.282777	-0.725271	1.511598	-0.632399	0.679690						
0.782966												
2	-1.020350	0.282777	1.771041	0.418708	-0.632399	0.679690						
0.782966												
3	-1.020350	1.284279	-0.725271	-0.127737	-0.632399	1.408395	-					
1.277194												
4	2.400396	-1.720227	-0.725271	1.511598	-0.632399	-0.049015	-					
1.277194												
...					
...												
3905	-0.653841	1.284279	-0.725271	-0.674182	-0.632399	-1.506426						
0.782966												
3906	-0.898180	-0.718725	1.771041	-1.220627	-0.632399	1.408395						
0.782966												
3907	1.545210	0.282777	-0.725271	2.058043	-0.632399	-0.777720						
0.782966												
3908	1.789549	1.284279	-0.725271	-0.127737	-0.632399	-1.506426						

```
0.782966
3909 -0.776011  0.282777 -0.725271 -1.220627  1.581280 -0.049015 -
1.277194
```

```

      24      25
0  -0.382245 -0.774151
1  -0.459799  0.643615
2  -0.245196 -0.065268
3   0.213475 -0.065268
4  -0.024889  2.061382
...
3905 -0.536973  0.643615
3906  1.529609 -0.065268
3907 -0.360576  0.643615
3908 -0.252799  0.643615
3909 -1.082511 -1.483035
```

```
[3910 rows x 26 columns]
```

```
## apply transformation on test(transform)
```

```
X_test=preprocessor.transform(X_test)
```

```
X_test
```

```
array([[ 0.          ,  0.          ,  0.          , ..., -1.2771941 ,
        -0.73751038, -0.77415132],
       [ 1.          ,  0.          ,  0.          , ..., -1.2771941 ,
        -0.6704111 , -0.06526803],
       [ 1.          ,  0.          ,  0.          , ...,  0.78296635,
        -0.4208322 , -0.77415132],
       ...,
       [ 0.          ,  1.          ,  0.          , ...,  0.78296635,
         0.69001249,  0.64361526],
       [ 1.          ,  0.          ,  0.          , ...,  0.78296635,
        -0.22827818, -0.77415132],
       [ 1.          ,  1.          ,  0.          , ...,  0.78296635,
        -0.44611323,  2.06138184]], shape=(978, 26))
```

```
pd.DataFrame(X_train)
```

```

      0      1      2      3      4      5      6      7      8      9      ...      16
\
0      1.0      0.0      0.0      1.0      1.0      0.0      0.0      0.0      0.0      0.0      ... -0.721400
1      1.0      0.0      1.0      0.0      1.0      0.0      0.0      0.0      0.0      1.0      ... -0.721400
2      1.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      ... -0.721400
3      1.0      0.0      1.0      0.0      1.0      1.0      0.0      0.0      0.0      1.0      ... -0.721400
4      0.0      0.0      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      ... -0.721400
```

...
3905	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400
3906	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.455047
3907	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.455047
3908	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	...	1.455047
3909	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	-0.721400

	17	18	19	20	21	22
23 \						
0	-1.020350	1.284279	-0.725271	-0.127737	-0.632399	0.679690
0.782966						
1	0.690023	0.282777	-0.725271	1.511598	-0.632399	0.679690
0.782966						
2	-1.020350	0.282777	1.771041	0.418708	-0.632399	0.679690
0.782966						
3	-1.020350	1.284279	-0.725271	-0.127737	-0.632399	1.408395 -
1.277194						
4	2.400396	-1.720227	-0.725271	1.511598	-0.632399	-0.049015 -
1.277194						
...

...						
3905	-0.653841	1.284279	-0.725271	-0.674182	-0.632399	-1.506426
0.782966						
3906	-0.898180	-0.718725	1.771041	-1.220627	-0.632399	1.408395
0.782966						
3907	1.545210	0.282777	-0.725271	2.058043	-0.632399	-0.777720
0.782966						
3908	1.789549	1.284279	-0.725271	-0.127737	-0.632399	-1.506426
0.782966						
3909	-0.776011	0.282777	-0.725271	-1.220627	1.581280	-0.049015 -
1.277194						

	24	25
0	-0.382245	-0.774151
1	-0.459799	0.643615
2	-0.245196	-0.065268
3	0.213475	-0.065268
4	-0.024889	2.061382
...
3905	-0.536973	0.643615
3906	1.529609	-0.065268
3907	-0.360576	0.643615
3908	-0.252799	0.643615

```
3909 -1.082511 -1.483035
```

```
[3910 rows x 26 columns]
```

```
y_train
```

```
3995    0
2610    0
3083    0
3973    0
4044    0
```

```
..
```

```
4426    0
466     0
3092    0
3772    0
860     1
```

```
Name: ProdTaken, Length: 3910, dtype: int64
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report, ConfusionMatrixDisplay, \
                                precision_score, recall_score, f1_score,
roc_auc_score, roc_curve
```

```
models={
    "Logisitic Regression":LogisticRegression(),
    "Decision Tree":DecisionTreeClassifier(),
    "Random Forest":RandomForestClassifier(),
    "Gradient Boost":GradientBoostingClassifier(),
    "Adaboost":AdaBoostClassifier()
}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Training set performance
    model_train_accuracy = accuracy_score(y_train, y_train_pred) #
Calculate Accuracy
    model_train_f1 = f1_score(y_train, y_train_pred,
average='weighted') # Calculate F1-score
    model_train_precision = precision_score(y_train, y_train_pred) #
```

```

Calculate Precision
    model_train_recall = recall_score(y_train, y_train_pred) #
Calculate Recall
    model_train_rocauc_score = roc_auc_score(y_train, y_train_pred)

    # Test set performance
    model_test_accuracy = accuracy_score(y_test, y_test_pred) #
Calculate Accuracy
    model_test_f1 = f1_score(y_test, y_test_pred, average='weighted')
# Calculate F1-score
    model_test_precision = precision_score(y_test, y_test_pred) #
Calculate Precision
    model_test_recall = recall_score(y_test, y_test_pred) # Calculate
Recall
    model_test_rocauc_score = roc_auc_score(y_test, y_test_pred)
#Calculate Roc

print(list(models.keys())[i])
print('Model performance for Training set')
print("- Accuracy: {:.4f}".format(model_train_accuracy))
print("- F1 score: {:.4f}".format(model_train_f1))

print("- Precision: {:.4f}".format(model_train_precision))
print("- Recall: {:.4f}".format(model_train_recall))
print("- Roc Auc Score: {:.4f}".format(model_train_rocauc_score))

print('-----')

print('Model performance for Test set')
print("- Accuracy: {:.4f}".format(model_test_accuracy))
print("- F1 score: {:.4f}".format(model_test_f1))
print("- Precision: {:.4f}".format(model_test_precision))
print("- Recall: {:.4f}".format(model_test_recall))
print("- Roc Auc Score: {:.4f}".format(model_test_rocauc_score))

print('='*35)
print('\n')

```

Logisitic Regression

Model performance for Training set

- Accuracy: 0.8460
- F1 score: 0.8202
- Precision: 0.7016
- Recall: 0.3032
- Roc Auc Score: 0.6368

Model performance for Test set

- Accuracy: 0.8364
 - F1 score: 0.8087
 - Precision: 0.6914
 - Recall: 0.2932
 - Roc Auc Score: 0.6307
- =====

Decision Tree

Model performance for Training set

- Accuracy: 1.0000
 - F1 score: 1.0000
 - Precision: 1.0000
 - Recall: 1.0000
 - Roc Auc Score: 1.0000
-

Model performance for Test set

- Accuracy: 0.9182
 - F1 score: 0.9168
 - Precision: 0.8171
 - Recall: 0.7487
 - Roc Auc Score: 0.8540
- =====

Random Forest

Model performance for Training set

- Accuracy: 1.0000
 - F1 score: 1.0000
 - Precision: 1.0000
 - Recall: 1.0000
 - Roc Auc Score: 1.0000
-

Model performance for Test set

- Accuracy: 0.9274
 - F1 score: 0.9219
 - Precision: 0.9615
 - Recall: 0.6545
 - Roc Auc Score: 0.8240
- =====

Gradient Boost

Model performance for Training set

- Accuracy: 0.8939
- F1 score: 0.8819
- Precision: 0.8756
- Recall: 0.5021

- Roc Auc Score: 0.7429

Model performance for Test set

- Accuracy: 0.8589
- F1 score: 0.8398
- Precision: 0.7732
- Recall: 0.3927
- Roc Auc Score: 0.6824

=====

Adaboost

Model performance for Training set

- Accuracy: 0.8478
- F1 score: 0.8146
- Precision: 0.7815
- Recall: 0.2551
- Roc Auc Score: 0.6194

Model performance for Test set

- Accuracy: 0.8354
- F1 score: 0.7987
- Precision: 0.7500
- Recall: 0.2356
- Roc Auc Score: 0.6083

=====

Hyperparameter Training

```
rf_params = {"max_depth": [5, 8, 15, None, 10],
             "max_features": [5, 7, "auto", 8],
             "min_samples_split": [2, 8, 15, 20],
             "n_estimators": [100, 200, 500, 1000]}
gradient_params={"loss": ['log_loss', 'deviance', 'exponential'],
                 "criterion": ['friedman_mse', 'squared_error', 'mse'],
                 "min_samples_split": [2, 8, 15, 20],
                 "n_estimators": [100, 200, 500],
                 "max_depth": [5, 8, 15, None, 10]
                 }
```

gradient_params

```
{'loss': ['log_loss', 'deviance', 'exponential'],
 'criterion': ['friedman_mse', 'squared_error', 'mse'],
 'min_samples_split': [2, 8, 15, 20],
 'n_estimators': [100, 200, 500],
 'max_depth': [5, 8, 15, None, 10]}
```

rf_params

```
{'max_depth': [5, 8, 15, None, 10],
 'max_features': [5, 7, 'auto', 8],
 'min_samples_split': [2, 8, 15, 20],
 'n_estimators': [100, 200, 500, 1000]}

# Models list for Hyperparameter tuning
randomcv_models = [
    ("RF", RandomForestClassifier(), rf_params),
    ("GradientBoost", GradientBoostingClassifier(), gradient_params)
]
```

```
randomcv_models
```

```
[('RF',
  RandomForestClassifier(),
  {'max_depth': [5, 8, 15, None, 10],
   'max_features': [5, 7, 'auto', 8],
   'min_samples_split': [2, 8, 15, 20],
   'n_estimators': [100, 200, 500, 1000]}),
 ('GradientBoost',
  GradientBoostingClassifier(),
  {'loss': ['log_loss', 'deviance', 'exponential'],
   'criterion': ['friedman_mse', 'squared_error', 'mse'],
   'min_samples_split': [2, 8, 15, 20],
   'n_estimators': [100, 200, 500],
   'max_depth': [5, 8, 15, None, 10]})]
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
model_param = {}
for name, model, params in randomcv_models:
    random = RandomizedSearchCV(estimator=model,
                                param_distributions=params,
                                n_iter=100,
                                cv=3,
                                verbose=2,
                                n_jobs=-1)

    random.fit(X_train, y_train)
    model_param[name] = random.best_params_

for model_name in model_param:
    print(f"----- Best Params for {model_name}
    -----")
    print(model_param[model_name])
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
----- Best Params for RF -----
{'n_estimators': 100, 'min_samples_split': 2, 'max_features': 8,
 'max_depth': None}
```

```

----- Best Params for GradientBoost -----
{'n_estimators': 500, 'min_samples_split': 20, 'max_depth': 15,
'loss': 'exponential', 'criterion': 'squared_error'}

models={

    "Random
Forest":RandomForestClassifier(n_estimators=1000,min_samples_split=2,
max_features=7,max_depth=None),

    "GradientBoostClassifier":GradientBoostingClassifier(n_estimators=500,
min_samples_split=20,
                                                    max_depth=15,

    loss='exponential',
    criterion='mse')
}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Training set performance
    model_train_accuracy = accuracy_score(y_train, y_train_pred) #
Calculate Accuracy
    model_train_f1 = f1_score(y_train, y_train_pred,
average='weighted') # Calculate F1-score
    model_train_precision = precision_score(y_train, y_train_pred) #
Calculate Precision
    model_train_recall = recall_score(y_train, y_train_pred) #
Calculate Recall
    model_train_rocauc_score = roc_auc_score(y_train, y_train_pred)

    # Test set performance
    model_test_accuracy = accuracy_score(y_test, y_test_pred) #
Calculate Accuracy
    model_test_f1 = f1_score(y_test, y_test_pred, average='weighted')
# Calculate F1-score
    model_test_precision = precision_score(y_test, y_test_pred) #
Calculate Precision
    model_test_recall = recall_score(y_test, y_test_pred) # Calculate
Recall
    model_test_rocauc_score = roc_auc_score(y_test, y_test_pred)

```

```

#Calculate Roc
print(list(models.keys())[i])

print('Model performance for Training set')
print("- Accuracy: {:.4f}".format(model_train_accuracy))
print("- F1 score: {:.4f}".format(model_train_f1))

print("- Precision: {:.4f}".format(model_train_precision))
print("- Recall: {:.4f}".format(model_train_recall))
print("- Roc Auc Score: {:.4f}".format(model_train_rocauc_score))

print('-----')

print('Model performance for Test set')
print("- Accuracy: {:.4f}".format(model_test_accuracy))
print("- F1 score: {:.4f}".format(model_test_f1))
print("- Precision: {:.4f}".format(model_test_precision))
print("- Recall: {:.4f}".format(model_test_recall))
print("- Roc Auc Score: {:.4f}".format(model_test_rocauc_score))

print('='*35)
print('\n')

```

Random Forest

Model performance for Training set

- Accuracy: 1.0000
- F1 score: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- Roc Auc Score: 1.0000

Model performance for Test set

- Accuracy: 0.9335
- F1 score: 0.9291
- Precision: 0.9632
- Recall: 0.6859
- Roc Auc Score: 0.8398

=====

InvalidParameterError Traceback (most recent call last)

Cell In[42], line 13

```
11 for i in range(len(list(models))):
```

```

12     model = list(models.values())[i]
--> 13     model.fit(X_train, y_train) # Train model
15     # Make predictions
16     y_train_pred = model.predict(X_train)

```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:1358, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)

```

1353 partial_fit_and_fitted = (
1354     fit_method.__name__ == "partial_fit" and
_is_fitted(estimator)
1355 )
1357 if not global_skip_validation and not partial_fit_and_fitted:
-> 1358     estimator._validate_params()
1360 with config_context(
1361     skip_parameter_validation=(
1362         prefer_skip_nested_validation or
global_skip_validation
1363     )
1364 ):
1365     return fit_method(estimator, *args, **kwargs)

```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:471, in BaseEstimator._validate_params(self)

```

463 def _validate_params(self):
464     """Validate types and values of constructor parameters
465
466     The expected type and values must be defined in the
`_parameter_constraints`
469     accepted constraints.
470     """
--> 471     validate_parameter_constraints(
472         self._parameter_constraints,
473         self.get_params(deep=False),
474         caller_name=self.__class__.__name__,
475     )

```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils_param_validation.py:98, in validate_parameter_constraints(parameter_constraints, params, caller_name)

```

92 else:
93     constraints_str = (
94         f"{'', '.join([str(c) for c in constraints[:-1]])} or"
95         f" {constraints[-1]}"
96     )
--> 98 raise InvalidParameterError(
99     f"The {param_name!r} parameter of {caller_name} must be"
100     f" {constraints_str}. Got {param_val!r} instead."
101 )

```

```
InvalidParameterError: The 'criterion' parameter of
GradientBoostingClassifier must be a str among {'squared_error',
'friedman_mse'}. Got 'mse' instead.
```

```
## Plot ROC AUC Curve
from sklearn.metrics import roc_auc_score, roc_curve
plt.figure()

# Add the models to the list that you want to view on the ROC plot
auc_models = [
    {
        'label': 'Gradient Boost Classifier',
        'model': GradientBoostingClassifier(n_estimators=500,
min_samples_split=20,
max_depth=15,
loss='exponential',
criterion='mse'),
        'auc': 0.9026
    },
]

# create loop through all model
for algo in auc_models:
    model = algo['model'] # select the model
    model.fit(X_train, y_train) # train the model
# Compute False positive rate, and True positive rate
    fpr, tpr, thresholds = roc_curve(y_test,
model.predict_proba(X_test)[:,:1])
# Calculate Area under the curve to display on the plot
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (algo['label'],
algo['auc']))
# Custom settings for the plot
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.savefig("auc.png")
plt.show()
```

[illegible]

Cell In[43], line 21

```
19 for algo in auc_models:
20     model = algo['model'] # select the model
--> 21     model.fit(X_train, y_train) # train the model
22 # Compute False positive rate, and True positive rate
23     fpr, tpr, thresholds = roc_curve(y_test,
model.predict_proba(X_test)[: ,1])
```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:1358, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)

```
1353 partial_fit_and_fitted = (
1354     fit_method.__name__ == "partial_fit" and
_is_fitted(estimator)
1355 )
1357 if not global_skip_validation and not partial_fit_and_fitted:
-> 1358     estimator._validate_params()
1360 with config_context(
1361     skip_parameter_validation=(
1362         prefer_skip_nested_validation or
global_skip_validation
1363     )
1364 ):
1365     return fit_method(estimator, *args, **kwargs)
```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:471, in BaseEstimator._validate_params(self)

```
463 def _validate_params(self):
464     """Validate types and values of constructor parameters
465
466     The expected type and values must be defined in the
`_parameter_constraints`
467     (...) 469     accepted constraints.
470     """
--> 471     validate_parameter_constraints(
472         self._parameter_constraints,
473         self.get_params(deep=False),
474         caller_name=self.__class__.__name__,
475     )
```

File C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils_param_validation.py:98, in

```
validate_parameter_constraints(parameter_constraints, params,
caller_name)
92 else:
93     constraints_str = (
94         f"{'', '.join([str(c) for c in constraints[:-1]])} or"
95         f" {constraints[-1]}"
96     )
--> 98 raise InvalidParameterError(
```

```
99     f"The {param_name!r} parameter of {caller_name} must be"  
100     f" {constraints_str}. Got {param_val!r} instead."  
101 )
```

InvalidParameterError: The 'criterion' parameter of GradientBoostingClassifier must be a str among {'squared_error', 'friedman_mse'}. Got 'mse' instead.

<Figure size 640x480 with 0 Axes>