Bank Marketing Effectiveness Prediction

```
## Importing necessary libraries

# For scientific computation and processing array elements.
import numpy as np
from scipy.stats import norm

# Importing pandas
import pandas as pd

# For plotting statstical visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# For pretty-printing tabular data
from tabulate import tabulate

# For plotting feature importance
from sklearn.feature_selection import mutual_info_classif

# For handling class imbalance
from imblearn.over_sampling import SMOTE

# For Split dataset into train and test
from sklearn.model_selection import train_test_split

# For Cross-Validation and Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV

# For Scaliing dataset
from sklearn.preprocessing import MinMaxScaler

# Importing algorithams for building model
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Evaluation metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc

# For plotting Decision Tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

# For building Artificial Neural Networks
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.layers import Dropout
from keras import regularizers

# # For model explainability
# import shap

#To ignore warnings
import warnings
warnings.filterwarnings('ignore')


# Loabding Dataset
df=pd.read_csv("/content/bank-full.csv", sep =";")
df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	58	management	married	tertiary	no	2143	yes	no	unknown	
1	44	technician	single	secondary	no	29	yes	no	unknown	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	
4	33	unknown	single	unknown	no	1	no	no	unknown	

Next steps: Generate code with df View recommended plots

```
# First Five Observations
df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	58	management	married	tertiary	no	2143	yes	no	unknown	
1	44	technician	single	secondary	no	29	yes	no	unknown	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	
4	33	unknown	single	unknown	no	1	no	no	unknown	

Next steps: Generate code with df View recommended plots

```
# Last five observations
df.tail()
```

	age	job	marital	education	default	balance	housing	loan	contac
45206	51	technician	married	tertiary	no	825	no	no	cellula
45207	71	retired	divorced	primary	no	1729	no	no	cellula
45208	72	retired	married	secondary	no	5715	no	no	cellula
45209	57	blue-collar	married	secondary	no	668	no	no	telephone
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellula

```
# Bsic description of Dataset
df.describe(include='all')
```

	age	job	marital	education	default	balance	housing	loa
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	4521
unique	NaN	12	3	4	2	NaN	2	
top	NaN	blue-collar	married	secondary	no	NaN	yes	r
freq	NaN	9732	27214	23202	44396	NaN	25130	379
mean	40.936210	NaN	NaN	NaN	NaN	1362.272058	NaN	Na
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	Na
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	Na
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	Na
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	Na
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	Na
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	Na

```
# Checking duplicated values in dataset
count_duplicated = df.duplicated().sum()
print(f'Dataset having {count_duplicated} duplicated values')
```

```
Dataset having 0 duplicated values
```

```
# Checking for number of null values in dataset
count_null_df=pd.DataFrame({'columns':df.columns,'number_of_nulls_values':df.isna().sum()})
count_null_df.set_index('columns').sort_values(by='number_of_nulls_values', ascending = False)
```

columns	number_of_nulls_values
age	0
day	0
poutcome	0
previous	0
pdays	0
campaign	0
duration	0
month	0
contact	0
job	0
loan	0
housing	0
balance	0
default	0
education	0
marital	0
y	0

```
# Basic description of Dataset
df.describe(include='all')
```

	age	job	marital	education	default	balance	housing	loa
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	4521
unique	NaN	12	3	4	2	NaN	2	
top	NaN	blue-collar	married	secondary	no	NaN	yes	r
freq	NaN	9732	27214	23202	44396	NaN	25130	3796
mean	40.936210	NaN	NaN	NaN	NaN	1362.272058	NaN	Na
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	Na
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	Na
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	Na
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	Na
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	Na
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	Na

```
# Finding categorical variables
categorical_variables = [var for var in df.columns if df[var].dtype=='O']
print('There are {} categorical variables'.format(len(categorical_variables)))
print('--'*45)
print(categorical_variables)
```

```
There are 10 categorical variables
------------------------------------------------------------------------------------------
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'y']
```

```
# Finding numerical variables
numerical_variables=[var for var in df.columns if var not in categorical_variables]
print('There are {} numerical variables'.format(len(numerical_variables)))
print('--'*45)
print(numerical_variables)
```

```
There are 7 numerical variables
------------------------------------------------------------------------------------------
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
```

```
# Check Unique Values and its frequency for each variable
for var in df.columns:
    print(df[var].value_counts())
    print('--'*45)
```

```
Name: count, Length: 559, dtype: int64
------------------------------------------------------------------------------------------
previous
0      36954
1       2772
2       2106
3       1142
4        714
5        459
6        277
7        205
8        129
9         92
10        67
11        65
12        44
13        38
15        20
14        19
17        15
16        13
19        11
20         8
23         8
18         6
22         6
24         5
27         5
21         4
29         4
25         4
30         3
38         2
37         2
26         2
28         2
51         1
275        1
58         1
32         1
40         1
55         1
35         1
41         1
Name: count, dtype: int64
------------------------------------------------------------------------------------------
poutcome
unknown    36959
failure     4901
other       1840
success     1511
Name: count, dtype: int64
------------------------------------------------------------------------------------------
y
no     39922
yes     5289
Name: count, dtype: int64
------------------------------------------------------------------------------------------
```

```
# This is formatted as code
```

[1] Handling Duplicate Values

```
# Checking duplicated values in dataset
count_duplicated = df.duplicated().sum()
print(f'Dataset having {count_duplicated} duplicated values')
```

```
Dataset having 0 duplicated values
```

[2] Handling Null / Missing Values

```
# Replacing the unknown values with null across all the dataset
df = df.replace('unknown', np.nan)
```

```
# Checking for number of null values
count_null_df=pd.DataFrame({'columns':df.columns,'number_of_nulls_values':df.isna().sum(),'percentage_null_values':round(df.isna().sum(
count_null_df.set_index('columns').sort_values(by='percentage_null_values', ascending = False)
```

columns	number_of_nulls_values	percentage_null_values
poutcome	36959	81.75
contact	13020	28.80
education	1857	4.11
job	288	0.64
month	0	0.00
previous	0	0.00
pdays	0	0.00
campaign	0	0.00
duration	0	0.00
age	0	0.00
day	0	0.00
loan	0	0.00
housing	0	0.00
balance	0	0.00
default	0	0.00
marital	0	0.00
y	0	0.00

```
# Dropping variables having more than 50% null values
df.drop(columns='poutcome', inplace=True)

# Replacing null values with the most frequent value in a variable
df['contact']=df['contact'].fillna(df['contact'].mode()[0])
df['education']=df['education'].fillna(df['education'].mode()[0])
df['job']=df['job'].fillna(df['job'].mode()[0])
```

```
# Verify for null values are removed
df.isna().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
y            0
dtype: int64
```

```
# Using Inter Quartile Range for removing outliers from numerical variables

# Defining outlier features but remove flat IQR feature pdays and previous
outlier_var=['age', 'balance', 'duration', 'campaign']

# Capping dataset
for i in outlier_var:
    # Findling IQR
    Q1=df[i].quantile(0.25)
    Q3=df[i].quantile(0.75)
    IQR=Q3-Q1

    # Defining upper and lower limit
    lower_limit =df[i].quantile(0.25)-1.5*IQR
    upper_limit =df[i].quantile(0.75)+1.5*IQR

    # Applying lower and upper limit to each variables
    df.loc[(df[i] > upper_limit),i] = upper_limit
    df.loc[(df[i] < lower_limit),i] = lower_limit
```

Data Pre-processing

[1] Categorical Encoding

```
# Checking basic info of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  float64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  y          45211 non-null  object
dtypes: float64(1), int64(6), object(9)
memory usage: 5.5+ MB
```

```
# Addressing categorical variables from the dataset
categorical_variables=df.describe(include=['object']).columns
print(f'Categorical variables are : {list(categorical_variables)}')
```

```
Categorical variables are : ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'y']
```

```
# Checking categories in each categorical features
for var in categorical_variables:
    print(df[var].value_counts())
    print('__'*45)
```

```
entrepreneur      1487
unemployed        1303
housemaid         1240
student            938
Name: count, dtype: int64
__________________________________________________________________________________________
marital
married     27214
single      12790
divorced     5207
Name: count, dtype: int64
```

```
no     44396
yes      815
Name: count, dtype: int64
________________________________________
housing
yes    25130
no     20081
Name: count, dtype: int64
________________________________________
loan
no     37967
yes     7244
Name: count, dtype: int64
________________________________________
contact
cellular     42305
telephone     2906
Name: count, dtype: int64
________________________________________
month
may    13766
jul     6895
aug     6247
jun     5341
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: count, dtype: int64
________________________________________
y
no     39922
yes     5289
Name: count, dtype: int64
________________________________________
```

```
## label encoding

# Mapping the categorical variables whoes having limited categories
df['marital'] = df['marital'].map({'single':0,'married':1,'divorced':2})
df['education'] = df['education'].map({'secondary':0,'tertiary':1, 'primary':2})
df['default'] = df['default'].map({'yes':1,'no':0})
df['housing'] = df['housing'].map({'yes':1,'no':0})
df['loan'] = df['loan'].map({'yes':1,'no':0})
df['contact'] = df['contact'].map({'cellular':1,'telephone':0})
df['y'] = df['y'].map({'yes':1,'no':0})
```

```
## One hot encoding for variable job and month
df=pd.get_dummies(df, columns=['job', "month"], prefix=["job", "month"], drop_first=True)
```

```
# Checking basic information of dataset after feature encoding
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 35 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   age                45211 non-null  float64
 1   marital            45211 non-null  int64  
 2   education          45211 non-null  int64  
 3   default            45211 non-null  int64  
 4   balance            45211 non-null  int64  
 5   housing            45211 non-null  int64  
 6   loan               45211 non-null  int64  
 7   contact            45211 non-null  int64  
 8   day                45211 non-null  int64  
 9   duration           45211 non-null  int64  
 10  campaign           45211 non-null  int64  
 11  pdays              45211 non-null  int64  
 12  previous           45211 non-null  int64  
 13  y                  45211 non-null  int64  
 14  job_blue-collar    45211 non-null  bool   
 15  job_entrepreneur   45211 non-null  bool   
 16  job_housemaid      45211 non-null  bool   
 17  job_management     45211 non-null  bool   
 18  job_retired        45211 non-null  bool   
 19  job_self-employed  45211 non-null  bool   
 20  job_services       45211 non-null  bool   
 21  job_student        45211 non-null  bool   
 22  job_technician     45211 non-null  bool   
 23  job_unemployed     45211 non-null  bool   
 24  month_aug          45211 non-null  bool   
```

```
 25  month_dec          45211 non-null  bool
 26  month_feb          45211 non-null  bool
 27  month_jan          45211 non-null  bool
 28  month_jul          45211 non-null  bool
 29  month_jun          45211 non-null  bool
 30  month_mar          45211 non-null  bool
 31  month_may          45211 non-null  bool
 32  month_nov          45211 non-null  bool
 33  month_oct          45211 non-null  bool
 34  month_sep          45211 non-null  bool
dtypes: bool(21), float64(1), int64(13)
memory usage: 5.7 MB
```

```
# Final Dataset
pd.set_option('display.max_columns', None)
df.head()
```

	age	marital	education	default	balance	housing	loan	contact	day	duration
0	58.0	1	1	0	2143	1	0	1	5	261
1	44.0	0	0	0	29	1	0	1	5	151
2	33.0	1	0	0	2	1	1	1	5	76
3	47.0	1	0	0	1506	1	0	1	5	92
4	33.0	0	0	0	1	0	0	1	5	198

[2] Separating Dependant and Independant variables

```
## Separating independant variables and dependant variable

# Creating the dataset with all dependent variables
dependent_variable = 'y'

# Creating the dataset with all independent variables
independent_variables = list(set(df.columns.tolist()) - {dependent_variable})

# Create the data of independent variables
X = df[independent_variables].copy()
# Create the data of dependent variable
y = df[dependent_variable].copy()
```

[3] Feature Manipulation & Selection

[4] Handling Imbalanced Dataset

```
# Import model imblearn in envirnoment
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

```
# Using Synthetic Minority Oversampling Technique (SMOTE) for handling class imbalance

from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=0)

# fit predictor and target variable
x_smote, y_smote = smote.fit_resample(X,y)
```

[5] Data Splitting

```
# Splitting dataset into training set and test set
from sklearn.model_selection import train_test_split, GridSearchCV
X_train, X_test, y_train, y_test= train_test_split(x_smote, y_smote, test_size=0.2, random_state=42)
```

```
# Checking shape of split
print(f'Shape of X_train : {X_train.shape}')
print(f'Shape of X_test : {X_test.shape}')
print(f'Shape of y_train : {y_train.shape}')
print(f'Shape of y_test : {y_test.shape}')
```

```
Shape of X_train : (63875, 34)
Shape of X_test : (15969, 34)
Shape of y_train : (63875,)
Shape of y_test : (15969,)
```

- We divided the dataset into 20% for model testing and 80% for training.

```
# Checking values of splitted dataset
X_train[0:3]
```

	duration	marital	month_sep	job_services	month_jul	age	job_technici
76180	310	1	False	False	False	70.028256	Fal
36038	67	1	False	False	False	50.000000	Fal
41791	78	1	False	False	False	62.000000	Fal

```
# Checking values of splitted dataset
X_test[0:3]
```

	duration	marital	month_sep	job_services	month_jul	age	job_technici
72809	315	1	False	False	False	39.273337	Tr
71061	643	0	False	False	True	46.690791	Tr
57176	643	1	False	False	False	54.030217	Tr

.

[6] Data Scaling

```
# Transforming data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- MinMaxScaler preserves the shape of the original distribution. It doesn't meaningfully change the information embedded in the original data. So we used MinMaxScaler for scaling the dataset.

```
# Checking values of splitted dataset after normalisation
X_train[0:5]
```

```
array([[0.48211509, 0.5       , 0.        , 0.        , 0.        ,
        0.9910144 , 0.        , 1.        , 0.        , 0.3       ,
        0.        , 0.        , 0.        , 0.5       , 0.        ,
        1.        , 0.        , 0.09977064, 0.        , 0.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.05263158, 0.        , 0.        , 0.97455752],
       [0.10419907, 0.5       , 0.        , 0.        , 0.        ,
        0.60952381, 0.        , 1.        , 0.        , 0.33333333,
        0.        , 0.        , 1.        , 0.5       , 1.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.36246313],
       [0.12130638, 0.5       , 0.        , 0.        , 0.        ,
        0.83809524, 0.        , 0.        , 0.        , 0.4       ,
        0.        , 0.        , 0.        , 0.5       , 0.        ,
        1.        , 0.4       , 0.        , 0.        , 0.        ,
```

```
        0.        , 0.        , 0.        , 1.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.36172566],
       [1.        , 0.        , 0.        , 0.        , 1.        ,
        0.2040902 , 0.        , 0.        , 0.        , 0.46666667,
        0.        , 0.        , 0.        , 0.5       , 0.        ,
        1.        , 0.8       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 1.        ,
        0.        , 0.        , 0.        , 0.36227876],
       [0.35769829, 0.        , 0.        , 0.        , 1.        ,
        0.19047619, 0.        , 0.        , 0.        , 0.9       ,
        0.        , 0.        , 1.        , 0.5       , 1.        ,
        1.        , 0.2       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.38219027]])
```

```
# Checking values of splitted dataset after normalisation
X_test[0:5]
```

```
array([[0.48989114, 0.5       , 0.        , 0.        , 0.        ,
        0.40520641, 1.        , 0.        , 0.        , 0.53333333,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        1.        , 0.2       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 1.        , 0.        ,
        0.        , 0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        , 0.        , 1.        ,
        0.54649125, 1.        , 0.        , 1.        , 0.7       ,
        0.        , 0.        , 0.        , 0.        , 1.        ,
        1.        , 0.2       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.56452802],
       [1.        , 0.5       , 0.        , 0.        , 0.        ,
        0.68628984, 1.        , 1.        , 0.        , 0.1       ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.2       , 0.        , 0.        , 0.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.36172566],
       [0.08864697, 0.5       , 0.        , 0.        , 0.        ,
        0.32380952, 0.        , 0.        , 0.        , 0.13333333,
        0.        , 0.        , 0.        , 0.5       , 0.        ,
        1.        , 0.2       , 0.34977064, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        1.        , 0.        , 1.        , 0.        , 0.        ,
        0.01754386, 0.        , 0.        , 0.49022861],
       [0.46500778, 0.5       , 0.        , 0.        , 0.        ,
        0.98754787, 1.        , 0.        , 0.        , 0.2       ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.2       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 1.        , 1.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.43473451]])
```

.

ML Model Implementation

```
# Defining function which fit classification algoritham, evaluate and visualise model using train test split

# Import evaluation metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc

# Defining function
def classification_model(X_train, X_test, y_train, y_test, clf):
    """
    function fit the algorithm on the training set, evaluate the model, and visualise evaluation metrics
    """
    ## Fit the model using training dataset
    model=clf.fit(X_train, y_train)
    print(model)
    print('=='*45)

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

    ## Evaluate the model
    print('Training set evaluation result :\n')
    cm_train = confusion_matrix(y_train, y_train_pred)
    accuracy_train = accuracy_score(y_train, y_train_pred)
    precision_train = precision_score(y_train, y_train_pred)
    recall_train = recall_score(y_train, y_train_pred)
    f1_train = f1_score(y_train, y_train_pred)
    roc_auc_score_train=roc_auc_score(y_train, y_train_pred)
    print("Confusion Matrix: \n", cm_train)
    print("Accuracy: ", accuracy_train)
    print("Precision: ", precision_train)
    print("Recall: ", recall_train)
    print("F1 Score: ", f1_train)
    print("roc_auc_score: ", roc_auc_score_train)
    print('\n-------------------------------\n')
    print('Test set evaluation result :\n')
    cm_test = confusion_matrix(y_test, y_test_pred)
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred)
    recall_test = recall_score(y_test, y_test_pred)
    f1_test = f1_score(y_test, y_test_pred)
    roc_auc_score_test=roc_auc_score(y_test, y_test_pred)
    print("Confusion Matrix: \n", cm_test)
    print("Accuracy: ", accuracy_test)
    print("Precision: ", precision_test)
    print("Recall: ", recall_test)
    print("F1 Score: ", f1_test)
    print("roc_auc_score: ", roc_auc_score_test)
    print('=='*45)

    ## Visualizes evaluation metrics
    fig,axes = plt.subplots(nrows=2, ncols=2)
    ax1 = sns.heatmap(cm_train, annot=True, ax=axes[0,0], fmt='d')
    ax1.set_title('Confusion Matrix for training set')
    ax1.set_ylabel('True label')
    ax1.set_xlabel('Predicted label')
    ax2 = sns.heatmap(cm_test, annot=True, ax=axes[0,1], fmt='d')
    ax2.set_title('Confusion Matrix for test set')
    ax2.set_ylabel('True label')
    ax2.set_xlabel('Predicted label')
    ax3 = sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1','roc_auc_score'], y=[accuracy_train, precision_train, recall_train, f1
    ax3.set_title('Evaluation Metrics for training set')
    ax3.tick_params(axis='x', rotation=90)
    ax4 = sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1','roc_auc_score'], y=[accuracy_test, precision_test, recall_test, f1_te
    ax4.set_title('Evaluation Metrics for test set')
    ax4.tick_params(axis='x', rotation=90)
    plt.tight_layout()
    plt.show()
    print('=='*45)

    return {'model': model, 'y_train_pred': y_train_pred, 'y_test_pred': y_test_pred, 'cm_test': cm_test, 'accuracy_test': accuracy_tes
            'precision_test': precision_test, 'recall_test': recall_test, 'f1_test': f1_test, 'roc_auc_score_test': roc_auc_score_test}
```

```
# Defining function which fit classification algoritham using GridSearchCV, evaluate and visualise model

# Import necessary dependancy
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc
from sklearn.model_selection import GridSearchCV

# Defining function
def classification_CV_model(X_train, X_test, y_train, y_test, clf, param_grid):
    """
    function fit the algorithm using GridSearchCV on the training set, evaluate the model, and visualise evaluation metrics
    """
    ## Fit the model on training dataset
    classifier = clf
    model = GridSearchCV(classifier, param_grid, verbose=1, scoring='accuracy', cv=3, n_jobs=-1)
    model.fit(X_train, y_train)
    print(model)
    print('=='*45)

    # Print the best parameters and score
    print("Best parameters:", model.best_params_)
    print("Best score:", model.best_score_)
    print('=='*45)

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

    ## Evaluate the model
    print('Training set evaluation result :\n')
    cm_train = confusion_matrix(y_train, y_train_pred)
    accuracy_train = accuracy_score(y_train, y_train_pred)
    precision_train = precision_score(y_train, y_train_pred)
    recall_train = recall_score(y_train, y_train_pred)
    f1_train = f1_score(y_train, y_train_pred)
    roc_auc_score_train=roc_auc_score(y_train, y_train_pred)
    print("Confusion Matrix: \n", cm_train)
    print("Accuracy: ", accuracy_train)
    print("Precision: ", precision_train)
    print("Recall: ", recall_train)
    print("F1 Score: ", f1_train)
    print("roc_auc_score: ", roc_auc_score_train)
    print('\n-------------------------------\n')
    print('Test set evaluation result :\n')
    cm_test = confusion_matrix(y_test, y_test_pred)
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred)
    recall_test = recall_score(y_test, y_test_pred)
    f1_test = f1_score(y_test, y_test_pred)
    roc_auc_score_test=roc_auc_score(y_test, y_test_pred)
    print("Confusion Matrix: \n", cm_test)
    print("Accuracy: ", accuracy_test)
    print("Precision: ", precision_test)
    print("Recall: ", recall_test)
    print("F1 Score: ", f1_test)
    print("roc_auc_score: ", roc_auc_score_test)
    print('=='*45)

    ## Visualizes evaluation metrics
    fig,axes = plt.subplots(nrows=2, ncols=2)
    ax1 = sns.heatmap(cm_train, annot=True, ax=axes[0,0], fmt='d')
    ax1.set_title('Confusion Matrix for training set')
    ax1.set_ylabel('True label')
    ax1.set_xlabel('Predicted label')
    ax2 = sns.heatmap(cm_test, annot=True, ax=axes[0,1], fmt='d')
    ax2.set_title('Confusion Matrix for test set')
    ax2.set_ylabel('True label')
    ax2.set_xlabel('Predicted label')
    ax3 = sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1','roc_auc_score'], y=[accuracy_train, precision_train, recall_train, f1
    ax3.set_title('Evaluation Metrics for training set')
    ax3.tick_params(axis='x', rotation=90)
    ax4 = sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1','roc_auc_score'], y=[accuracy_test, precision_test, recall_test, f1_te
    ax4.set_title('Evaluation Metrics for test set')
    ax4.tick_params(axis='x', rotation=90)
    plt.tight_layout()
    plt.show()
    print('=='*45)

    return {'model': model, 'y_train_pred': y_train_pred, 'y_test_pred': y_test_pred, 'cm_test': cm_test, 'accuracy_test': accuracy_tes
            'precision_test': precision_test, 'recall_test': recall_test, 'f1_test': f1_test, 'roc_auc_score_test': roc_auc_score_test}
```

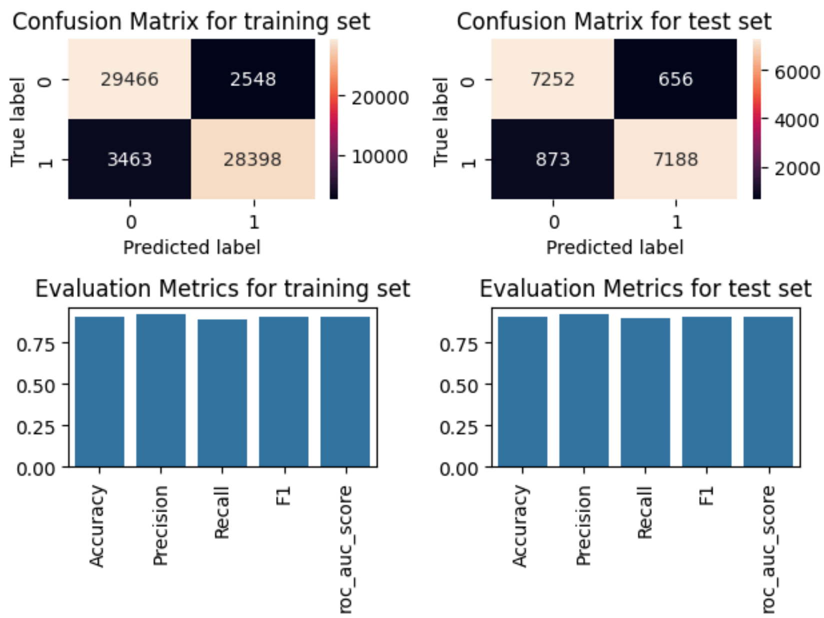
```
# Defining function to plot ROC curve
def plot_roc_curve(y_test, y_pred):
    """
    plots the roc curve
    """
    # Generate a list of false and true positive rates
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    # Calculate the area under the curve (AUC)
    roc_auc = auc(fpr, tpr)
    # Plotting the ROC curve
    plt.figure(figsize=(5,5))
    plt.plot(fpr, tpr, label='ROC curve (AUC = %0.4f)' % roc_auc)
    plt.plot([0, 1], [0, 1],'r--')
    # Labeling the graph
    plt.xlabel('False Positive Rate (Precision)')
    plt.ylabel('True Positive Rate (Recall)')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend(loc="lower right")
    # Show the plot
    plt.show()
```

.

[1] Logistic Regression

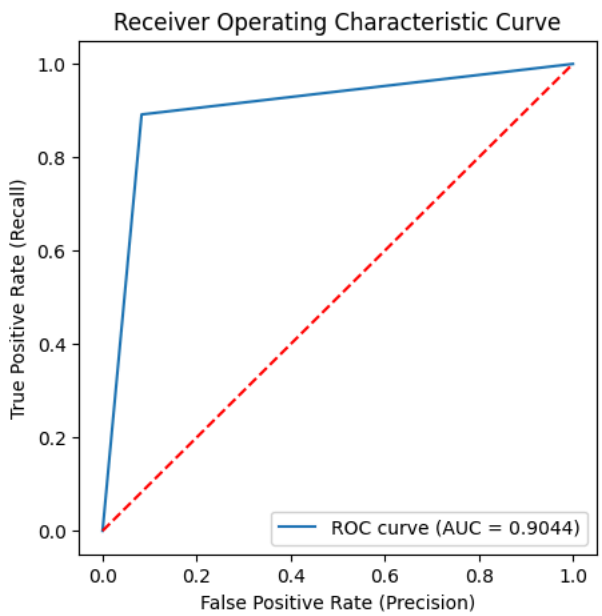
```
# Import Logistic Regression algorithm in envirnoment
from sklearn.linear_model import LogisticRegression
# Fitting Logistic Regression model to training set
Logistic_regression=LogisticRegression(fit_intercept=True, max_iter=10000,random_state=0)
lr=classification_model(X_train, X_test, y_train, y_test, Logistic_regression)
```

```
LogisticRegression(max_iter=10000, random_state=0)
==========================================================================================
Training set evaluation result :

Confusion Matrix:
 [[29466  2548]
 [ 3463 28398]]
Accuracy:  0.905894324853229
Precision:  0.9176630259161119
Recall:  0.8913091240074071
F1 Score:  0.9042941073447226
roc_auc_score:  0.9058594723554246

-------------------------------

Test set evaluation result :

Confusion Matrix:
 [[7252  656]
 [ 873 7188]]
Accuracy:  0.9042519882271902
Precision:  0.9163691993880673
Recall:  0.8917007815407517
F1 Score:  0.9038667085822069
roc_auc_score:  0.9043734054390659
==========================================================================================
```




```
==========================================================================================
```

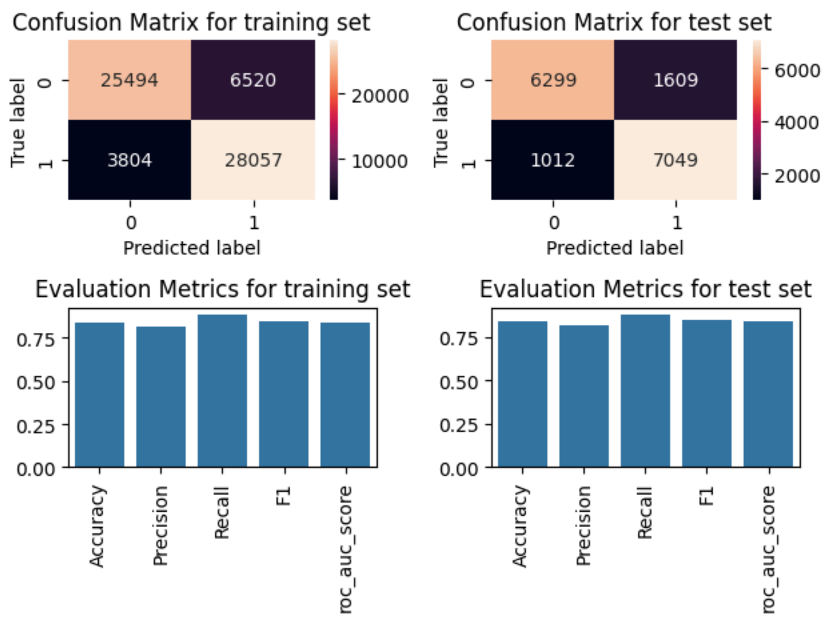
```
# Plot roc curve for Logistic Regression classifier
y_pred=lr['y_test_pred']
plot_roc_curve(y_test, y_pred)
```



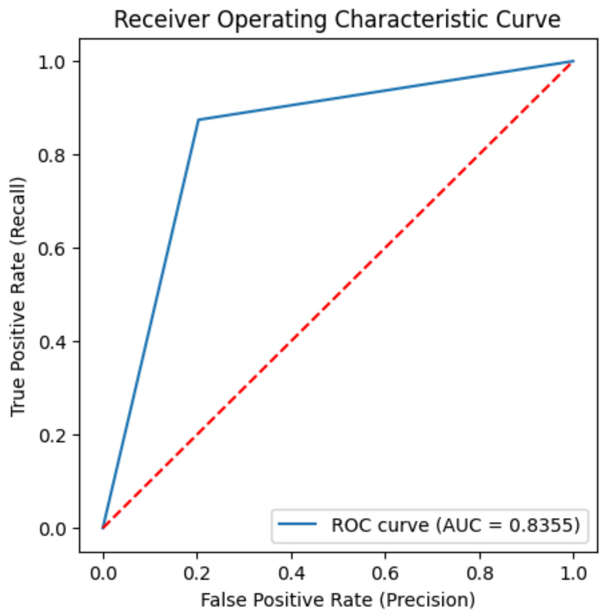
[2] Decision Tree

```
# Import Decision Tree algoritham in environment
from sklearn.tree import DecisionTreeClassifier
# Fitting Decision Tree model to training set
classifier_dt = DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=10, random_state=0)
dt=classification_model(X_train, X_test, y_train, y_test, classifier_dt)
```

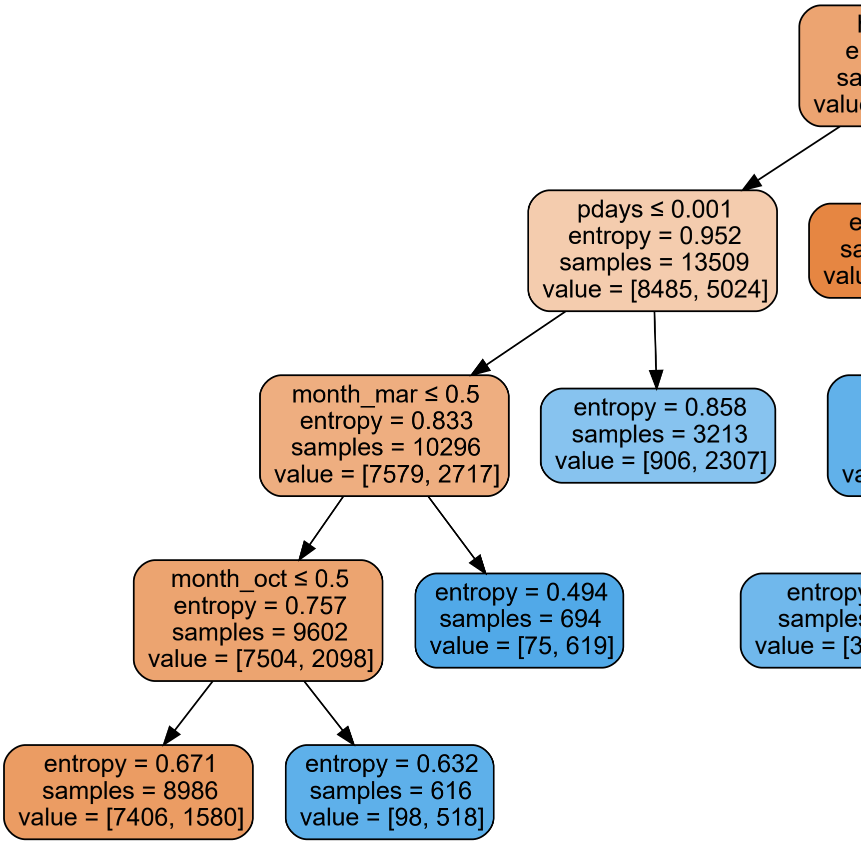
```
DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=10, random_state=0)
==================================================================================
Training set evaluation result :

Confusion Matrix:
 [[25494  6520]
 [ 3804 28057]]
Accuracy:  0.8383718199608611
Precision:  0.811435347196113
Recall:  0.8806063839804149
F1 Score:  0.8446070020169181
roc_auc_score:  0.838472742811723

------------------------------

Test set evaluation result :

Confusion Matrix:
 [[6299 1609]
 [1012 7049]]
Accuracy:  0.8358694971507296
Precision:  0.8141603141603142
Recall:  0.8744572633668279
F1 Score:  0.8432322507326994
roc_auc_score:  0.8354962088204904
==================================================================================
```




```
# Plot ROC curve for Decision Tree classifier
y_pred=dt['y_test_pred']
plot_roc_curve(y_test, y_pred)
```



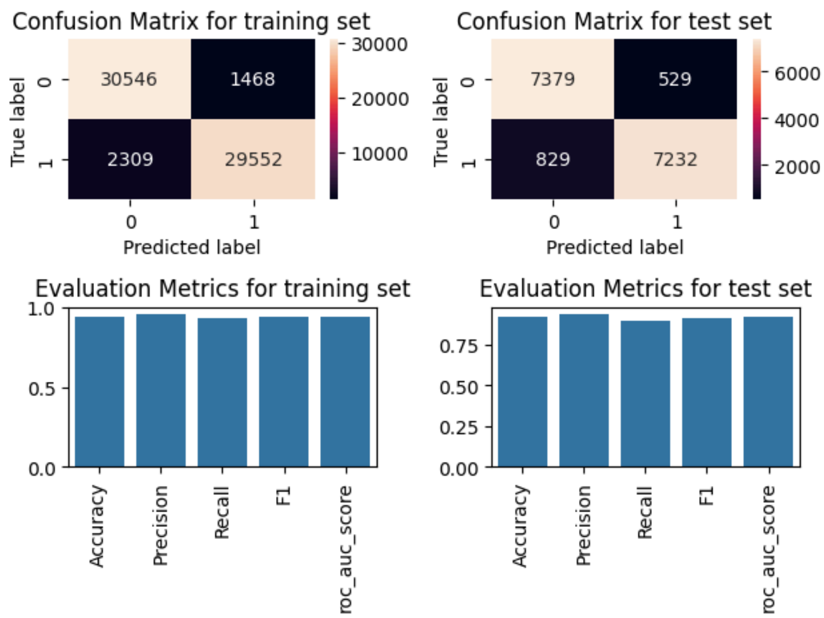

```
# Visulaizing Decision Tree

# Importing libraries
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

# Visualizing graph
graph = Source(tree.export_graphviz(dt['model'], out_file=None,feature_names=df[independent_variables].columns,
                                    rounded=True,special_characters=True, filled = True))
display(SVG(graph.pipe(format='svg')))
```



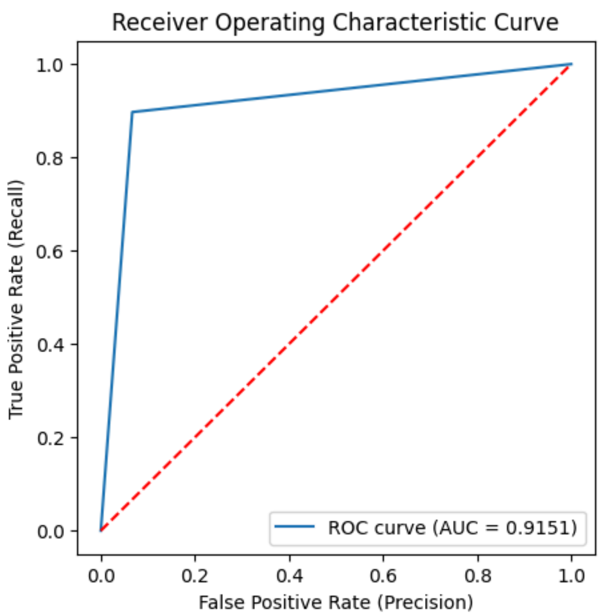
[3] K-Nearest Neighbor(KNN)

```
# Import KNN algoritham in envirnoment
from sklearn.neighbors import KNeighborsClassifier
# Fitting model to training set
classifier_knn = KNeighborsClassifier(n_neighbors=5)
knn=classification_model(X_train, X_test, y_train, y_test, classifier_knn)
```

```
KNeighborsClassifier()
==================================================================================
Training set evaluation result :

Confusion Matrix:
 [[30546  1468]
 [ 2309 29552]]
Accuracy:  0.9408688845401174
Precision:  0.952675693101225
Recall:  0.9275289538934748
F1 Score:  0.9399341613523958
roc_auc_score:  0.9408370077145266

------------------------------

Test set evaluation result :

Confusion Matrix:
 [[7379  529]
 [ 829 7232]]
Accuracy:  0.9149602354561964
Precision:  0.9318386805823992
Recall:  0.8971591613943679
F1 Score:  0.9141701428390847
roc_auc_score:  0.9151324385626367
==================================================================================
```




```
# Plot ROC curve for KNN classifier
y_pred=knn['y_test_pred']
plot_roc_curve(y_test, y_pred)
```



Cross- Validation & Hyperparameter Tuning

```
## Import KNN algoritham in envirnoment
from sklearn.neighbors import KNeighborsClassifier

## Fitting KNN model to training set using cross validation

# Defining param_dict
param_grid = {"n_neighbors": np.arange(1,7), "metric": ["euclidean", "cityblock"]}
# Creating instance of KNN classifier
classifier_knn = KNeighborsClassifier()
# Fitting model
knn_cv=classification_CV_model(X_train, X_test, y_train, y_test, classifier_knn, param_grid)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
GridSearchCV(cv=3, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'metric': ['euclidean', 'cityblock'],
                         'n_neighbors': array([1, 2, 3, 4, 5, 6])},
             scoring='accuracy', verbose=1)
================================================================================
Best parameters: {'metric': 'cityblock', 'n_neighbors': 1}
Best score: 0.9141448022830883
================================================================================
Training set evaluation result :

Confusion Matrix:
 [[32014     0]
 [    0 31861]]
Accuracy:  1.0
Precision:  1.0
Recall:  1.0
```