<u>Analysis Report</u>

Domain: Banking

Prepared By:

Tohfa Siddika Barbhuiya,

EXL Analytics

24 Jan’2021

EXECUTIVE SUMMARY:

BACKGROUND:

Context:

Leveraging customer information is paramount for most businesses. In the case of a bank, attributes of customers like the ones mentioned below can be crucial in strategizing a marketing campaign when launching a new product.

Data Description:

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to assess if the product (bank term deposit) would be ('yes') or not ('no') subscribed

OBJECTIVE:

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

1. age (numeric)

2. job: type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','selfemployed','services','student','technician','unemployed','unknown')

3. marital: marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4. education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5. default: has credit in default? (categorical: 'no','yes','unknown')

6. balance: average yearly balance, in euros (numeric)

7. housing: has a housing loan? (categorical: 'no','yes','unknown')

8. loan: has personal loan? (categorical: 'no','yes','unknown')

9. contact: contact communication type (categorical: 'cellular','telephone')

10. day: last contact day of the month (numeric 1 -31)

11. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

12. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

13. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

14. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

15. previous: number of contacts performed before this campaign and for this client (numeric)

16. poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

17. target: has the client subscribed a term deposit? (binary: "yes","no")

KEY FINDINGS:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


%matplotlib inline


#Supress warnings

import warnings

warnings.filterwarnings('ignore')


df=pd.read_csv("C:/Users/user/Downloads/bank-full.csv")

df.head()
```

2. Perform basic EDA which should include the following and print out your insights at every step. (20 marks)

a. Shape and data type of the data (1 marks)

```
df.shape #Finding Shape
```

df.dtypes #Finding Datatype

```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
Target       object
dtype: object
```

c. Report statistical summary of the dataset. (3 marks)

df.describe()

b. Check info of the dataset (1 marks)

df.info() #checking info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
age          45211 non-null int64
job          45211 non-null object
marital      45211 non-null object
education    45211 non-null object
default      45211 non-null object
balance      45211 non-null int64
housing      45211 non-null object
loan         45211 non-null object
contact      45211 non-null object
day          45211 non-null int64
month        45211 non-null object
duration     45211 non-null int64
campaign     45211 non-null int64
pdays        45211 non-null int64
previous     45211 non-null int64
poutcome     45211 non-null object
Target       45211 non-null object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

d. Check the presence of missing values and impute if there is any (3 marks)

df.isnull().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
poutcome     0
Target       0
dtype: int64
```

No missing values present

!pip install pandas_profiling

import pandas_profiling

df.profile_report()

e. Checking the presence of outliers and impute if there is any (3 marks)

plt.figure(figsize= (25,25))

plt.subplot(5,2,1)

sns.boxplot(x= df.balance, color='orange')

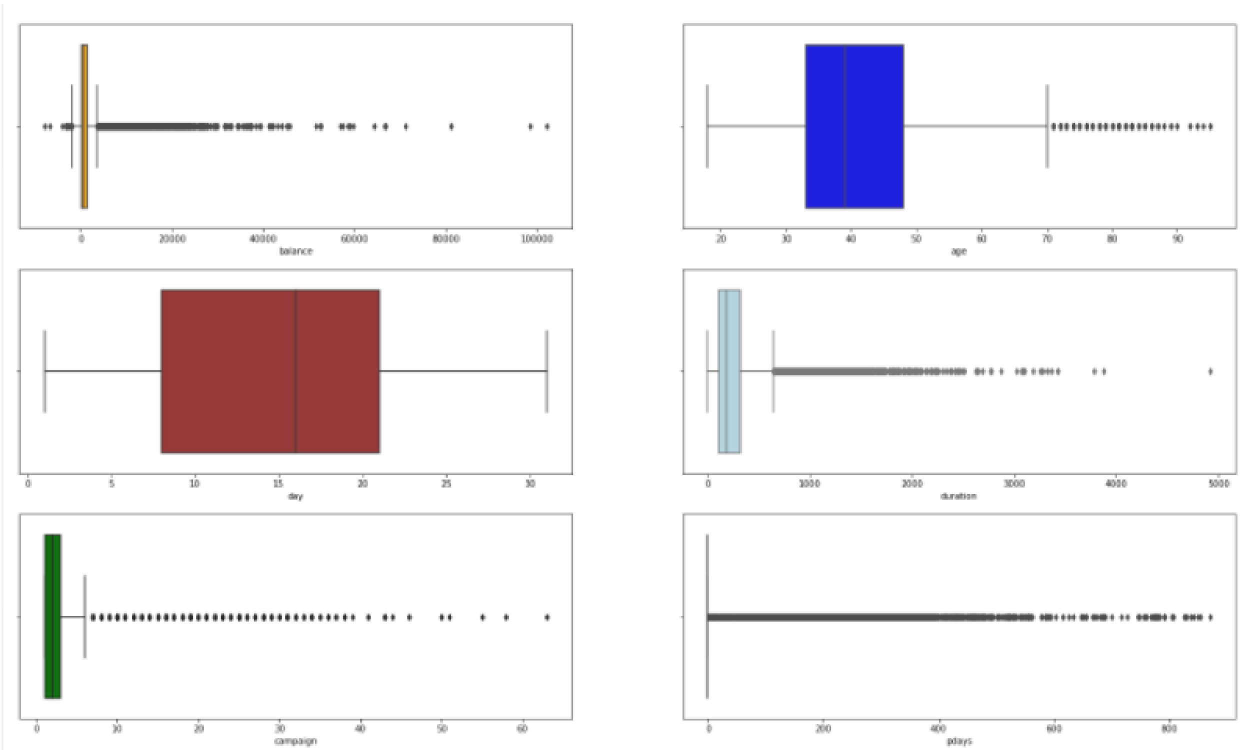
plt.subplot(5,2,2)

sns.boxplot(x= df.age, color='blue')

plt.subplot(5,2,3)

sns.boxplot(x= df.day, color='brown')

```
plt.subplot(5,2,4)
sns.boxplot(x= df.duration, color='lightblue')


plt.subplot(5,2,5)
sns.boxplot(x= df.campaign, color='green')


plt.subplot(5,2,6)
sns.boxplot(x= df.pdays, color='yellow')
```




```
## Binning the variables to overcome the skweness, to deal with outliers


#Binning Balance
def bal_group(series):
    if series < 0:
```

```
        return "negative balance"
    elif 0 <= series < 500:
        return "low balance"
    elif 500 <= series < 2000:
        return "moderate balance"
    elif 2000 <= series < 10000:
        return "high balance"
    elif 10000 <= series:
        return "very high balance"

df['bal_group'] = df['balance'].apply(bal_group)

df['bal_group'].value_counts()
pd.crosstab(df['bal_group'],df['Target'])

def camp_group(series):
    if series <= 1:
        return "1st Contact"
    elif 2 <= series <= 5:
        return "2-5 Contacts"
    elif 6 <= series <= 10:
        return "6-10 Contacts"
    elif 10 < series:
        return "More than 10 Contacts"

df['camp_group'] = df['campaign'].apply(camp_group)

#Most of people subscribe with lower number of contacts
```

```
df['camp_group'].value_counts()
pd.crosstab(df['camp_group'],df['Target'])


def dura_group(series):
    if series <= 120:
        return "<2 mints Duration"
    elif 120 < series <= 600:
        return "<2-10 mints Duration"
    elif 600 < series <= 1800:
        return "<10-30 mints Contacts"
    elif 1800 < series:
        return "More than 30 mints Contacts"


df['dura_group'] = df['duration'].apply(dura_group)


df['dura_group'].value_counts()
pd.crosstab(df['dura_group'],df['Target'])
df_1=df.copy()
df.drop(['balance','campaign'],axis=1,inplace=True)
```

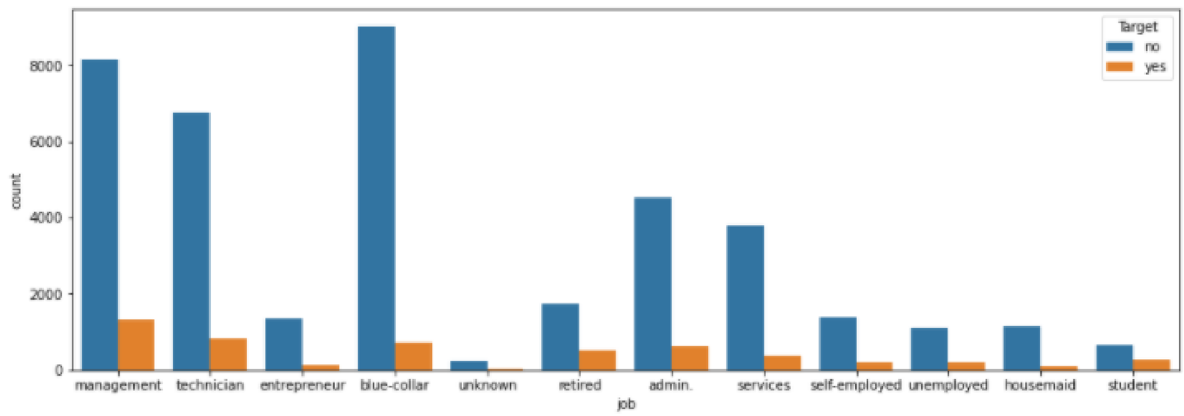
h. Perform bivariate analysis using pairplot and mention your findings. (3 marks)

```
df.groupby(['Target']).agg(['mean','median'])
```

#job - blue collar jobs was the most contacted in marketing campaign however they are the least who subscribed it.

Target	no	yes
job		
student	71.321962	28.678038
retired	77.208481	22.791519
unemployed	84.497314	15.502686
management	86.244449	13.755551
admin.	87.797331	12.202669
self-employed	88.157061	11.842939
unknown	88.194444	11.805556
technician	88.943004	11.056996
services	91.116996	8.883004
housemaid	91.209677	8.790323
entrepreneur	91.728312	8.271688
blue-collar	92.725031	7.274969

<matplotlib.axes._subplots.AxesSubplot at 0x2957cb2fac0>



#Student have mostly subscribed term deposit, however they are least contacted in marketing campaign

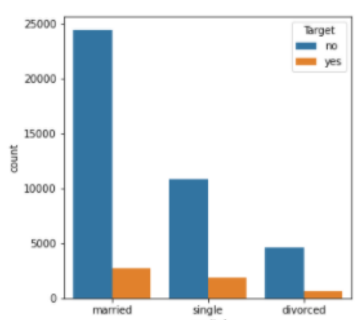
print(pd.crosstab(df['job'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(15,5))

sns.countplot(x='job',hue='Target',data=df)

Target	no	yes
marital		
single	85.050821	14.949179
divorced	88.054542	11.945458
married	89.876534	10.123466

<matplotlib.axes._subplots.AxesSubplot at 0x2957cb18820>



#Married was the most conatcted group however they are the least who subscribed it.

#Single are the most who subscribed for term deposit

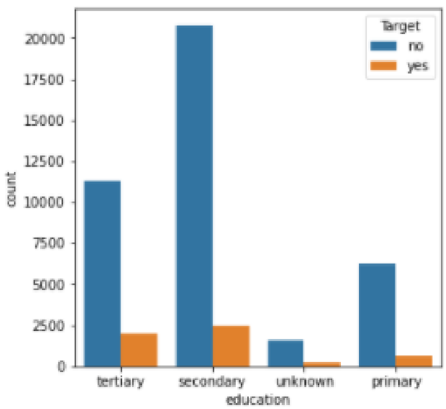
print(pd.crosstab(df['marital'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

sns.countplot(x='marital',hue='Target',data=df)

```
Target            no        yes
education
tertiary   84.993610  15.006390
unknown    86.429725  13.570275
secondary  89.440565  10.559435
primary    91.373522   8.626478
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2957ce4c0a0>
```



#tertiary education mostly subscribed to term deposit

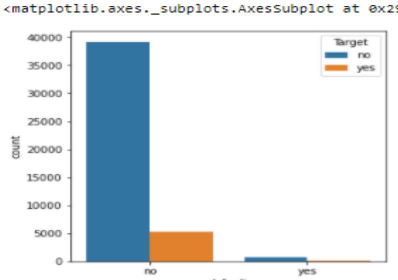
print(pd.crosstab(df['education'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

sns.countplot(x='education',hue='Target',data=df)

```
Target          no        yes
default
no       88.203892  11.796108
yes      93.619632   6.380368
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2957c860b20>
```



#defaulter/non defaultor- those who subscribed to term deposit is very low. dont believe will add much to our analysis

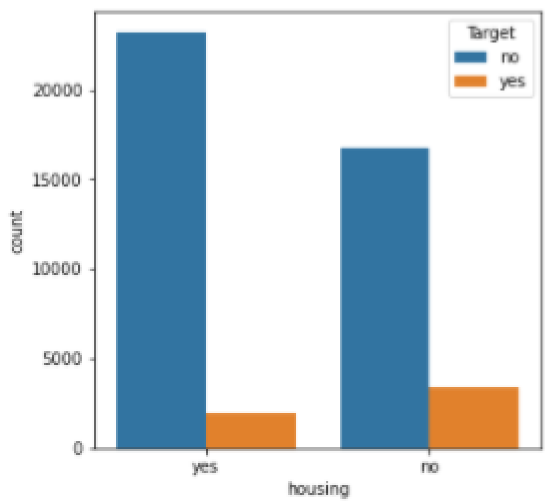
print(pd.crosstab(df['default'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

sns.countplot(x='default',hue='Target',data=df)

```
Target          no        yes
housing
no       83.297645  16.702355
yes      92.300040   7.699960
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2957cfab730>
```



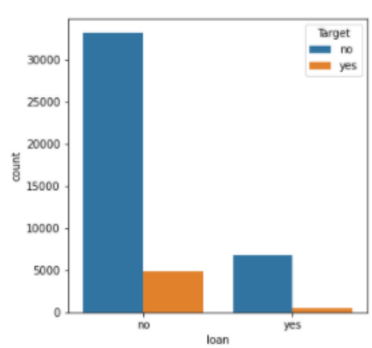
print(pd.crosstab(df['housing'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

sns.countplot(x='housing',hue='Target',data=df)

```
Target          no        yes
loan
no       87.344273  12.655727
yes      93.318609   6.681391
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2957ce61940>
```



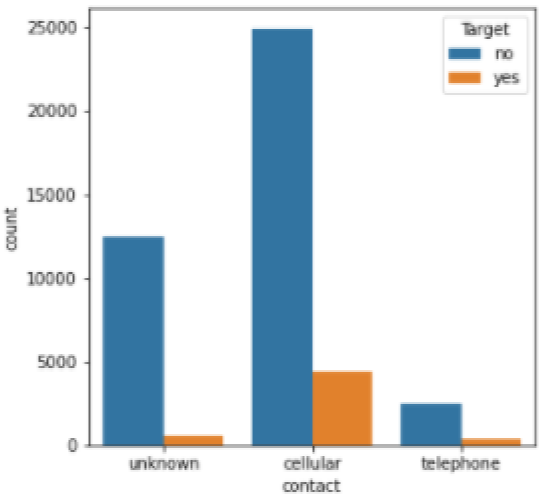
print(pd.crosstab(df['loan'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

sns.countplot(x='loan',hue='Target',data=df)

```
Target            no        yes
contact
cellular   85.081100  14.918900
telephone  86.579491  13.420509
unknown    95.929339   4.070661
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2957cea3340>
```



print(pd.crosstab(df['contact'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(5,5))

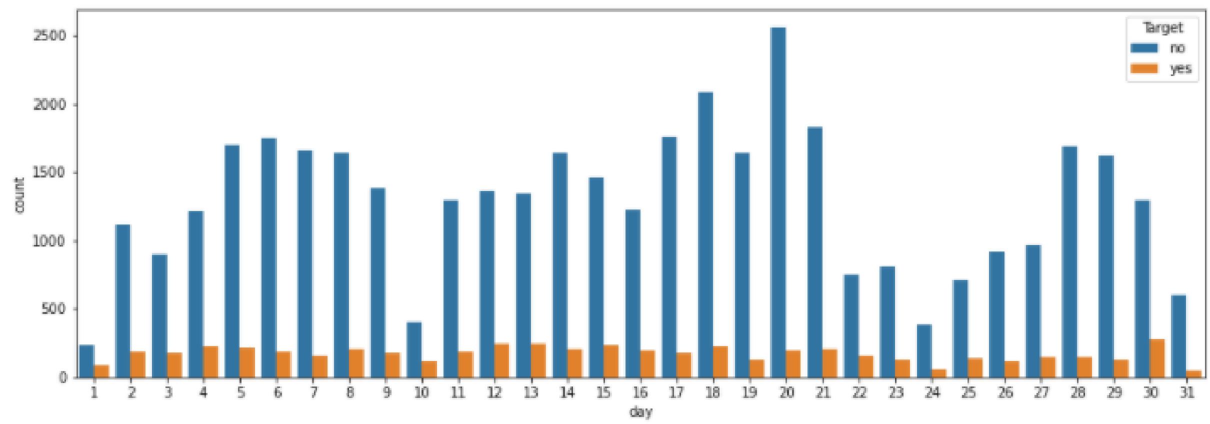
sns.countplot(x='contact',hue='Target',data=df)

print(pd.crosstab(df['day'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(15,5))

sns.countplot(x='day',hue='Target',data=df)

<matplotlib.axes._subplots.AxesSubplot at 0x2957cebd0a0>



#those who subscribed to previous campaign subscribed to term deposit as well.

#however the number is quite low in data for those subscribed to previous capaign.

print(pd.crosstab(df['poutcome'],df['Target'],normalize='index').mul(100).sort_values(by='yes',ascending=False))

plt.figure(figsize=(15,5))

sns.countplot(x='poutcome',hue='Target',data=df)

#campaign with lower numver of conatcts have higher chance of getting converted

sns.pairplot(df,hue='Target')

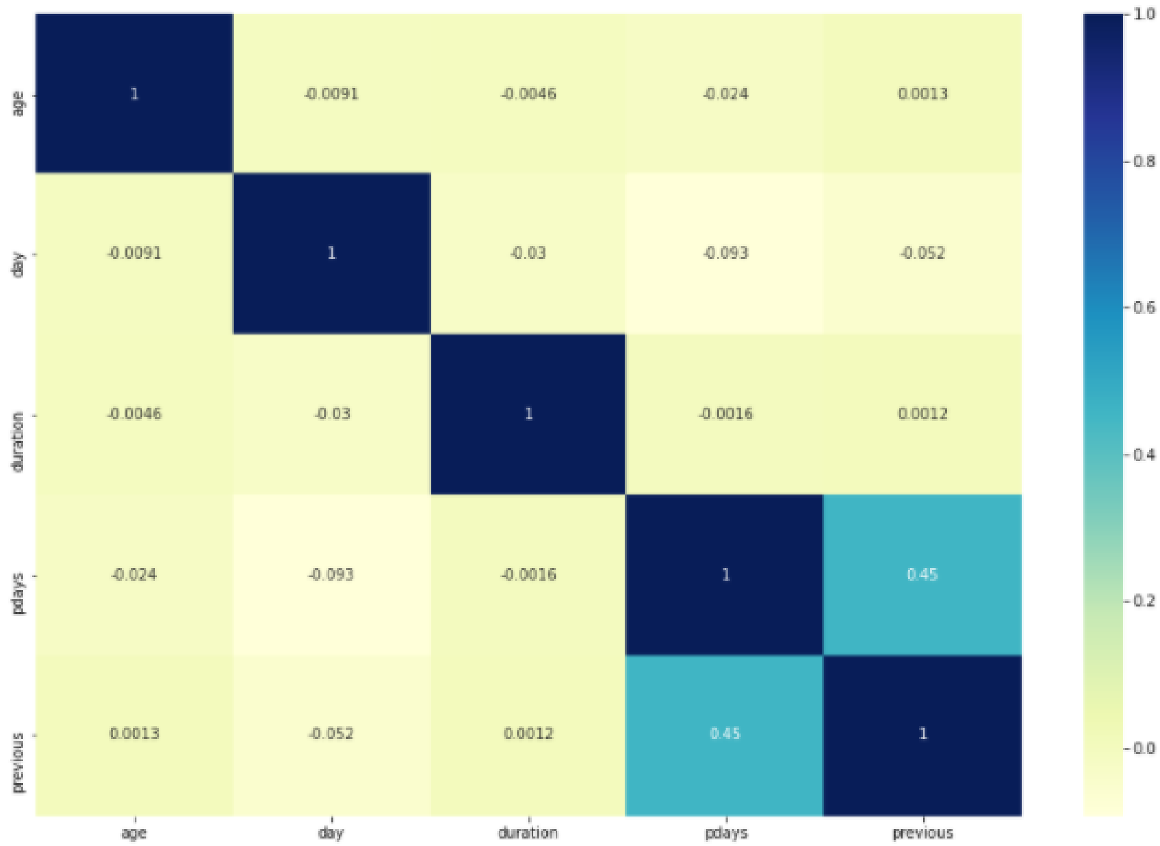
i. Check correlation among independent features and mention if there is any collinearity. (2 marks)

bank_corr=df.corr()

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

sns.heatmap(bank_corr,cmap="YlGnBu",annot=True)

#none of the variables seems to be highly correlated to each other, lets try dividing them into categories




```
df_1=df.copy()
df.drop(['balance','campaign'],axis=1,inplace=True)

df.columns



#Converting day to categorical
df['day']=df['day'].astype('category')
df['Target']=df['Target'].astype('category')
df.dtypes

# creating a dict file
label = {'yes': 1,'no': 0}
```

```
df.loan = [label[item] for item in df.loan]
df.default = [label[item] for item in df.default]
df.Target = [label[item] for item in df.Target]

df.head()

df_obj=pd.DataFrame(df.select_dtypes(include='object'))
df_obj=pd.get_dummies(df_obj)

df_2 = pd.concat([df,df_obj], axis=1)

print(df_2.columns)

df_3=df_2.drop(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'day',
               'month','bal_group', 'camp_group','duration','poutcome','dura_group'],axis=1)

df_3.info()

df_3.shape
```

3. Prepare the data to train a model – check if data types are appropriate, get rid of the missing values etc.(3 marks)

```
#Test Train Split
X=df_3.drop(['Target'],axis=1)
y=df_3['Target']

#Test Train Split
from sklearn.model_selection import train_test_split
```

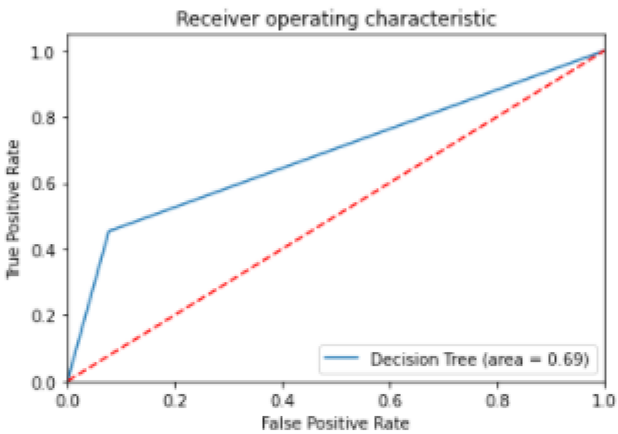
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3,
random_state=100)

X_train.columns

y_train.name
```

4. Train a decision tree model, note and comment on their performances across different classification metrics. (5 marks)

```
from sklearn.tree import DecisionTreeClassifier

dt_algo = DecisionTreeClassifier(random_state=101)
dt_algo.fit(X_train, y_train)



y_pred = dt_algo.predict(X_test)
print(dt_algo.score(X_train, y_train))
print(dt_algo.score(X_test , y_test))

#the model overfits, its accuracy on test data is good. However recall and precision are low. We
want model
#to predict better so as we don't waste time in handling customers which are not likely to
subscribe
# metrics
from sklearn import metrics

#confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print("confusion matrix: \n",confusion_matrix)
```

```
# accuracy
print("accuracy", metrics.accuracy_score(y_test, y_pred))
# precision
print("precision", metrics.precision_score(y_test, y_pred))
# recall/sensitivity
print("recall", metrics.recall_score(y_test, y_pred))
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred))

#ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
dt_roc_auc = roc_auc_score(y_test, dt_algo.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, dt_algo.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Decision Tree (area = %0.2f)' % dt_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
confusion matrix:
 [[11059   926]
 [  864   715]]
accuracy 0.8680330286051312
precision 0.4357099329677026
recall 0.4528182393920226
area-under-curve metric:  0.6877774968340994
```

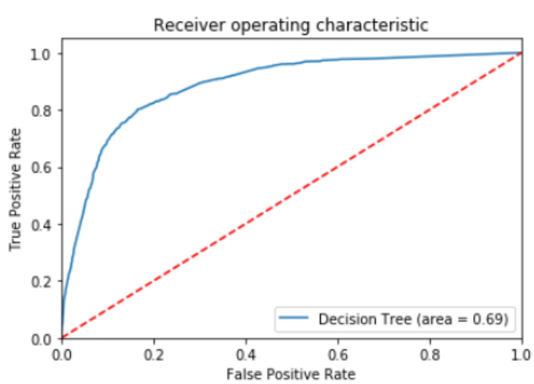



```
#Regularization using GridSearchCV

dt_algo1 = DecisionTreeClassifier(random_state = 102)

params = {"max_depth":np.arange(8,20),"max_features":np.arange(15,55,5),'min_samples_leaf': range(45, 65, 5),

    'min_samples_split': range(2,5),

    'criterion': ["entropy", "gini"]}


from sklearn.model_selection import GridSearchCV

model_cv = GridSearchCV(estimator = dt_algo1, param_grid = params,

                        scoring= 'accuracy',

                        cv=3,

                        verbose = 1,

                       return_train_score=True)


model_cv.fit(X_train, y_train)


# results of grid search CV

cv_results = pd.DataFrame(model_cv.cv_results_)
```

```
#cv_results



#parameters best value

best_score = model_cv.best_score_

best = model_cv.best_params_

best


#using best parameter values

dt_algo_best = DecisionTreeClassifier(max_depth= 14, max_features=
40,random_state=103,min_samples_leaf=50,

                                      min_samples_split=2,criterion='entropy')

dt_algo_best.fit(X_train, y_train)

# predict

y_pred1 = dt_algo_best.predict(X_test)


#accuracy and precision increase but recall decreases though by less amount and also roc_auc
metric

#decreases by .04.


# metrics

# confusion matrix

print("confusion matrix: \n", metrics.confusion_matrix(y_test, y_pred1))

# accuracy

print("accuracy: ", metrics.accuracy_score(y_test, y_pred1))

# precision

print("precision: ", metrics.precision_score(y_test, y_pred1))

# recall/sensitivity

print("recall: ", metrics.recall_score(y_test, y_pred1))
```

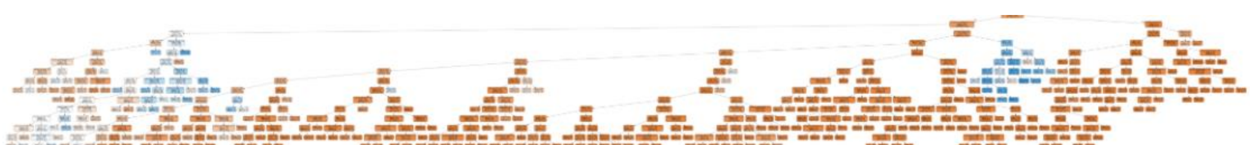
```
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred1))


#ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
dt_roc_auc1 = roc_auc_score(y_test, dt_algo_best.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, dt_algo_best.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Decision Tree (area = %0.2f)' % dt_roc_auc1)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('ROC_DecisionTree')
plt.show()
```

```
confusion matrix:
 [[11468   517]
 [  924   655]]
accuracy:  0.8937629017988794
precision:  0.5588737201365188
recall:  0.4148195060164661
area-under-curve metric:  0.6858411255572527
```



#Feature Importance

```
dt_imp_feature=pd.DataFrame(dt_algo_best.feature_importances_, columns = ["Imp"], index = X_train.columns)

dt_imp_feature.sort_values(by="Imp",ascending=False)[:15]

#dt_imp_feature.sort_values(by="Imp",ascending=False)


import os

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/graphviz-2.38/release/bin/'


#Tree structure

from sklearn import tree

from sklearn.tree import export_graphviz

#from sklearn.externals.six import StringIO

from IPython.display import Image

#import pydotplus

#import graphviz


features = X_train.columns

dot_data = StringIO()

export_graphviz(dt_algo_best, out_file=dot_data,feature_names=features,filled=True,rounded=True)


graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('tree.png')

Image(graph.create_png()
```



5. Build the ensemble models (random forest, bagging classifier, Adaboosting, and gradient boosting, and

stacking classifier) and compare the results. (15 marks)

#Bagging

from sklearn.ensemble import BaggingClassifier

bg_bank = BaggingClassifier(random_state=150)

bg_bank.fit(X_train, y_train)

y_pred_bg = bg_bank.predict(X_test)

print(bg_bank.score(X_train, y_train))

print(bg_bank.score(X_test , y_test))

```
0.9869181912977534
0.8891182542023002
```

#this algo seems overfit, also its recall is low

#metrices

#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_bg)

print("confusion matrix: \n",confusion_matrix)

accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_bg))

precision

print("precision", metrics.precision_score(y_test, y_pred_bg))

recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_bg))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_bg))

```
confusion matrix:
 [[11418   567]
 [  937   642]]
accuracy 0.8891182542023002
precision 0.5310173697270472
recall 0.4065864471184294
area-under-curve metric:  0.6796386553489517
```

#Regularization using GridSearchCV

bg_bank1 = BaggingClassifier(random_state=151)

params = {"n_estimators": np.arange(30,50,2),"max_features":[0.78,0.8,0.82,0.84],

'max_samples': [0.45,0.5,0.55,0.6],'oob_score':['True']}

model_cv_bg = GridSearchCV(estimator = bg_bank1, param_grid = params,

scoring= 'accuracy',

cv=3,

verbose = 1,

return_train_score=True)

model_cv_bg.fit(X_train, y_train)

results of grid search CV

cv_results_bg = pd.DataFrame(model_cv_bg.cv_results_)

#cv_results_bg

#parameters best value

best_score_bg = model_cv_bg.best_score_

best_bg = model_cv_bg.best_params_

best_bg

```
{'max_features': 0.8,
 'max_samples': 0.45,
 'n_estimators': 44,
 'oob_score': 'True'}
```

bg_algo_best = BaggingClassifier(max_features= 0.8, max_samples=0.8,n_estimators=42,oob_score=True,random_state=152)

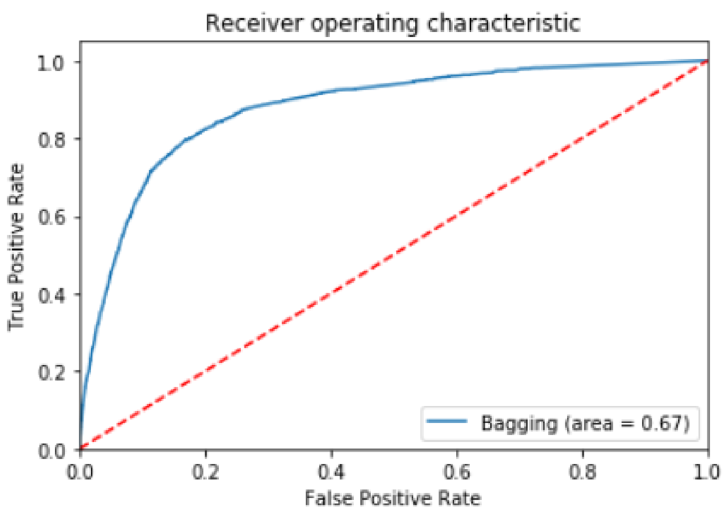
bg_algo_best.fit(X_train, y_train)

```
# predict
y_pred1_bg = bg_algo_best.predict(X_test)


#Accuracy improves and so does precision, but recall drops .
# metrics
# confusion matrix
print("confusion matrix: \n", metrics.confusion_matrix(y_test, y_pred1_bg))
# accuracy
print("accuracy: ", metrics.accuracy_score(y_test, y_pred1_bg))
# precision
print("precision: ", metrics.precision_score(y_test, y_pred1_bg))
# recall/sensitivity
print("recall: ", metrics.recall_score(y_test, y_pred1_bg))
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred1_bg))
```

```
confusion matrix:
 [[11510   475]
 [  966   613]]
accuracy:  0.8937629017988794
precision:  0.5634191176470589
recall:  0.3882203926535782
area-under-curve metric:  0.6742937591136059
```

```
#ROC
bg_roc_auc = roc_auc_score(y_test, bg_algo_best.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, bg_algo_best.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Bagging (area = %0.2f)' % bg_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('ROC_Bagging')
plt.show()



#Feature Importance
feature_importances = np.mean([ tree.feature_importances_ for tree in
bg_algo_best.estimators_], axis=0)
bg_imp_feature=pd.DataFrame(feature_importances, columns = ["Imp"])
bg_imp_feature.sort_values(by="Imp",ascending=False)
```




```
#RandomForest Algo
from sklearn.ensemble import RandomForestClassifier
rf_bank = RandomForestClassifier(random_state=200)
rf_bank.fit(X_train, y_train)


y_pred_rf = rf_bank.predict(X_test)
print(rf_bank.score(X_train, y_train))
```

print(rf_bank.score(X_test , y_test))

```
0.9861282270041394
0.887717487466824
```

#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_rf)

print("confusion matrix: \n",confusion_matrix)

accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_rf))

precision

print("precision", metrics.precision_score(y_test, y_pred_rf))

recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_rf))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_rf))

```
confusion matrix:
 [[11467   518]
 [ 1005   574]]
accuracy 0.887717487466824
precision 0.5256410256410257
recall 0.363521215959468
area-under-curve metric:  0.660150261713568
```

#Regularization using GridSearchCV

rf_bank1 = RandomForestClassifier(random_state = 201,oob_score="True",bootstrap=True)

params = {"n_estimators": np.arange(12,18,2),'criterion': ["entropy"],"max_depth": np.arange(9, 15,2),

"max_features":np.arange(15,30,5),'min_samples_leaf': range(26, 32, 2),

'min_samples_split': range(26, 32, 2)}

model_cv_rf = GridSearchCV(estimator = rf_bank1, param_grid = params,

scoring= 'accuracy',

cv=3,

verbose = 1,

return_train_score=True)

model_cv_rf.fit(X_train, y_train)

results of grid search CV

cv_results_rf = pd.DataFrame(model_cv_rf.cv_results_)

#cv_results_rf

#parameters best value

best_score_rf = model_cv_rf.best_score_

best_rf = model_cv_rf.best_params_

best_rf

```
{'criterion': 'entropy',
 'max_depth': 11,
 'max_features': 25,
 'min_samples_leaf': 26,
 'min_samples_split': 26,
 'n_estimators': 16}
```

rf_bank_best = RandomForestClassifier(max_depth= 13, max_features= 25,random_state=202,

n_estimators=16,criterion='entropy',

min_samples_leaf=30,min_samples_split=30)

rf_bank_best.fit(X_train, y_train)

predict

y_pred1_rf = rf_bank_best.predict(X_test)

metrics

confusion matrix

print("confusion matrix: \n", metrics.confusion_matrix(y_test, y_pred1_rf))

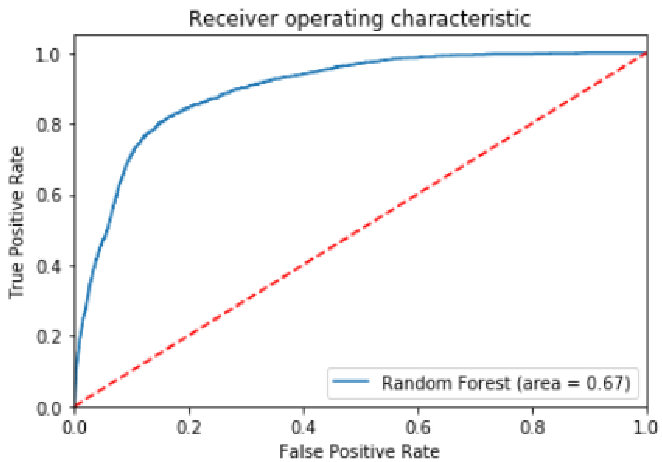
accuracy

```
print("accuracy: ", metrics.accuracy_score(y_test, y_pred1_rf))

# precision

print("precision: ", metrics.precision_score(y_test, y_pred1_rf))

# recall/sensitivity

print("recall: ", metrics.recall_score(y_test, y_pred1_rf))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred1_rf))
```

```
confusion matrix:
 [[11604   381]
 [  991   588]]
accuracy:  0.898849896785609
precision:  0.6068111455108359
recall:  0.3723875870804306
area-under-curve metric:  0.6702989249544832
```

```
#ROC

rf_roc_auc = roc_auc_score(y_test, rf_bank_best.predict(X_test))

fpr, tpr, thresholds = roc_curve(y_test, rf_bank_best.predict_proba(X_test)[:,1])

plt.figure()

plt.plot(fpr, tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

plt.savefig('ROC_RandomForest')

plt.show()



#Feature Importance
```

```
rf_imp_feature=pd.DataFrame(rf_bank_best.feature_importances_, columns = ["Imp"], index = X_train.columns)

rf_imp_feature.sort_values(by="Imp",ascending=False)[:15]
```




```
#AdaBoost Algo

from sklearn.ensemble import AdaBoostClassifier

# base estimator

tree = DecisionTreeClassifier(max_depth=2)


# adaboost with the tree as base estimator

ada_bank = AdaBoostClassifier(

    base_estimator=tree,

    algorithm="SAMME",random_state=250)

ada_bank.fit(X_train, y_train)


y_pred_ada = ada_bank.predict(X_test)

print(ada_bank.score(X_train, y_train))

print(ada_bank.score(X_test , y_test))
```

```
0.898758176130439
0.894426422884105
```

#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_ada)

print("confusion matrix: \n",confusion_matrix)

accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_ada))

precision

print("precision", metrics.precision_score(y_test, y_pred_ada))

recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_ada))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_ada))

```
confusion matrix:
 [[11504   481]
 [  951   628]]
accuracy 0.894426422884105
precision 0.5662759242560865
recall 0.3977200759974667
area-under-curve metric:  0.6787932878944363
```

#Regularization using GridSearchCV

ada_bank1 = AdaBoostClassifier(base_estimator=tree,algorithm="SAMME",random_state=251)

parameter grid

params = {"base_estimator__max_depth" : np.arange(2, 8,2),"n_estimators": [150,200,250],

"learning_rate":[0.2,0.3,0.4]}

model_cv_ada = GridSearchCV(estimator = ada_bank1, param_grid = params,

scoring= 'accuracy',

cv=3,

verbose = 1,

```
                   return_train_score=True)



model_cv_ada.fit(X_train, y_train)

# results of grid search CV
cv_results_ada = pd.DataFrame(model_cv_ada.cv_results_)
#cv_results_ada

#parameters best value
best_score_ada = model_cv_ada.best_score_
best_ada = model_cv_ada.best_params_
best_ada

# base estimator
tree = DecisionTreeClassifier(max_depth=4)
ada_bank_best = AdaBoostClassifier(base_estimator=tree , n_estimators=200,
                             random_state=252,learning_rate=0.3)
ada_bank_best.fit(X_train, y_train)
# predict
y_pred1_ada = ada_bank_best.predict(X_test)

# metrics
# confusion matrix
print("confusion matrix: \n", metrics.confusion_matrix(y_test, y_pred1_ada))
# accuracy
print("accuracy: ", metrics.accuracy_score(y_test, y_pred1_ada))
# precision
```

print("precision: ", metrics.precision_score(y_test, y_pred1_ada))

recall/sensitivity

print("recall: ", metrics.recall_score(y_test, y_pred1_ada))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred1_ada))

```
confusion matrix:
 [[11487   498]
 [  918   661]]
accuracy:  0.8956060159245061
precision:  0.5703192407247627
recall:  0.41861937935402155
area-under-curve metric:  0.6885337197145577
```

#ROC

ada_roc_auc = roc_auc_score(y_test, ada_bank_best.predict(X_test))

fpr, tpr, thresholds = roc_curve(y_test, ada_bank_best.predict_proba(X_test)[:,1])

plt.figure()

plt.plot(fpr, tpr, label='Ada Boost (area = %0.2f)' % ada_roc_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

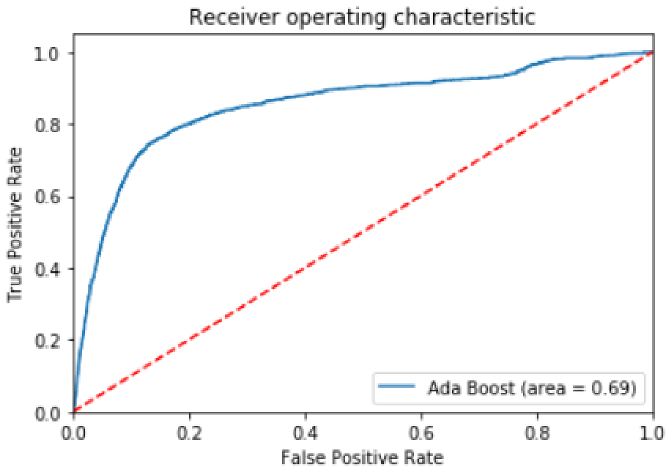
plt.savefig('ROC_AdaBoost')

plt.show()

#Feature Importance

ada_imp_feature=pd.DataFrame(ada_bank_best.feature_importances_, columns = ["Imp"], index = X_train.columns)

ada_imp_feature.sort_values(by="Imp",ascending=False)[:15]




```
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier
gbc_bank = GradientBoostingClassifier(random_state=300)
gbc_bank.fit(X_train, y_train)

y_pred_gbc = gbc_bank.predict(X_test)
print(gbc_bank.score(X_train, y_train))
print(gbc_bank.score(X_test , y_test))
```

```
0.9065630233513445
0.8989236213506341
```

```
#confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred_gbc)
print("confusion matrix: \n",confusion_matrix)
# accuracy
```

print("accuracy", metrics.accuracy_score(y_test, y_pred_gbc))

precision

print("precision", metrics.precision_score(y_test, y_pred_gbc))

recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_gbc))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_gbc))

```
confusion matrix:
 [[11630   355]
 [ 1016   563]]
accuracy 0.8989236213506341
precision 0.6132897603485838
recall 0.3565547815072831
area-under-curve metric:  0.6634672113627362
```

###As this run is taking very long, adjusting 1 parameter in grid search. However use trial and error by passing single

###value in fit

#Regularization using GridSearchCV

gbc_bank = GradientBoostingClassifier(random_state=301)

params = {"n_estimators": [200,210,220]}#,"learning_rate":[0.1,0.2],"max_depth": np.arange(10, 16)}

"max_features":np.arange(36,50,2),'min_samples_leaf': range(45, 60, 5)}

model_cv_gbc = GridSearchCV(estimator = gbc_bank, param_grid = params,

scoring= 'accuracy',

cv=3,

verbose = 1,

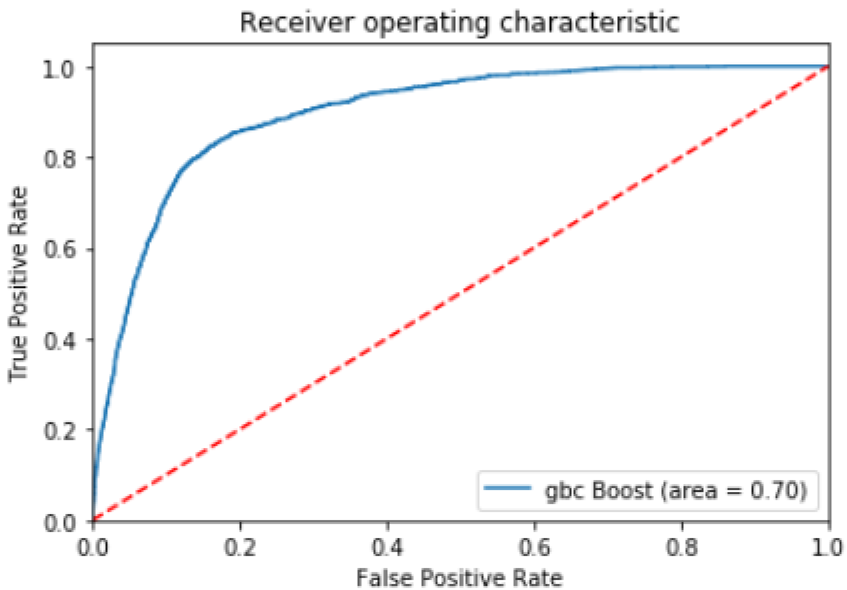
return_train_score=True)

model_cv_gbc.fit(X_train, y_train)

```
# results of grid search CV
cv_results_gbc = pd.DataFrame(model_cv_gbc.cv_results_)
#cv_results_gbc

#parameters best value
best_score_gbc = model_cv_gbc.best_score_
best_gbc = model_cv_gbc.best_params_
best_gbc

#After fitting best parameters
gbc_bank_best = GradientBoostingClassifier(learning_rate= 0.1, n_estimators= 220,max_depth= 14,
                                  max_features= 42,random_state=103,min_samples_leaf=50,min_samples_split=50)

gbc_bank_best.fit(X_train, y_train)
# predict
y_pred1_gbc = gbc_bank_best.predict(X_test)

# metrics
# confusion matrix
print("confusion matrix: \n", metrics.confusion_matrix(y_test, y_pred1_gbc))
# accuracy
print("accuracy: ", metrics.accuracy_score(y_test, y_pred1_gbc))
# precision
print("precision: ", metrics.precision_score(y_test, y_pred1_gbc))
# recall/sensitivity
print("recall: ", metrics.recall_score(y_test, y_pred1_gbc))
#Area under curve
```

```
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred1_gbc))


#ROC
gbc_roc_auc = roc_auc_score(y_test, gbc_bank_best.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, gbc_bank_best.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='gbc Boost (area = %0.2f)' % gbc_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('ROC_GradientBoost')
plt.show()
```

```
confusion matrix:
 [[11429   556]
 [  861   718]]
accuracy:  0.895532291359481
precision:  0.5635792778649922
recall:  0.45471817606079795
area-under-curve metric:  0.7041634267871782
```



#Feature Importance

gbc_imp_feature=pd.DataFrame(gbc_bank_best.feature_importances_, columns = ["Imp"], index = X_train.columns)

gbc_imp_feature.sort_values(by="Imp",ascending=False)[:15]

#xgboost

import xgboost as xgb

from xgboost.sklearn import XGBClassifier

train_data = np.array(X_train)

test_data = np.array(X_test)

xgb_bank = XGBClassifier(random_state=400)

xgb_bank.fit(train_data, y_train)

```
y_pred_xgb = xgb_bank.predict(test_data)

print(xgb_bank.score(train_data, y_train))

print(xgb_bank.score(test_data , y_test))
```

```
0.9032135747464215
0.8994396933058095
```

```
#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb)

print("confusion matrix: \n",confusion_matrix)

# accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb))

# precision

print("precision", metrics.precision_score(y_test, y_pred_xgb))

# recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_xgb))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb))
```

```
confusion matrix:
 [[11719   266]
 [ 1098   481]]
accuracy 0.8994396933058095
precision 0.643908969210174
recall 0.3046231792273591
area-under-curve metric:  0.641214384774297
```

```
#Regularization using GridSearchCV - 1st Iteration


params1 = {

   "colsample_bytree": [i/100.0 for i in range(78,82,2)],

   "learning_rate": [0.2,0.3],

   "n_estimators": [142,144,146],

   "subsample": [i/100.0 for i in range(80,84,2)]

}
```

```
model_cv_xgb1 = GridSearchCV(estimator = xgb_bank, param_grid = params1,
                        scoring= 'accuracy',
                        cv=3,
                        verbose = 1,
                       return_train_score=True)



model_cv_xgb1.fit(train_data,y_train)


# results of grid search CV
cv_results_xgb1 = pd.DataFrame(model_cv_xgb1.cv_results_)
cv_results_xgb1


#parameters best value
best_score_xgb1 = model_cv_xgb1.best_score_
best_xgb1 = model_cv_xgb1.best_params_
best_xgb1


#Choosing best parameter from 1st Iteration
xgb_bank_best1 =
XGBClassifier(colsample_bytree=0.8,learning_rate=0.2,n_estimators=144,subsample=0.82)
xgb_bank_best1.fit(train_data, y_train)
# predict
y_pred_xgb1 = xgb_bank_best1.predict(test_data)


#confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb1)
print("confusion matrix: \n",confusion_matrix)
```

```
# accuracy
print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb1))
# precision
print("precision", metrics.precision_score(y_test, y_pred_xgb1))
# recall/sensitivity
print("recall", metrics.recall_score(y_test, y_pred_xgb1))
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb1))

#Regularization using GridSearchCV - 2nd Iteration

params2 = {
    'min_child_weight':[4,5,6,7],"max_depth": [2,4,6],
}

model_cv_xgb2 = GridSearchCV(estimator = xgb_bank_best1, param_grid = params2,
                        scoring= 'accuracy',
                        cv=3,
                        verbose = 1,
                       return_train_score=True)

#Choosing best parameter obtained from 2nd Iteration an apply to model of 1st iteration
model_cv_xgb2.fit(train_data,y_train)

# results of grid search CV
cv_results_xgb2 = pd.DataFrame(model_cv_xgb2.cv_results_)
cv_results_xgb2
```

```
#parameters best value

best_score_xgb2 = model_cv_xgb2.best_score_

best_xgb2 = model_cv_xgb2.best_params_

best_xgb2



xgb_bank_best2 =
XGBClassifier(colsample_bytree=0.8,learning_rate=0.2,n_estimators=144,subsample=0.82,

                         min_child_weight=6,max_depth=4)

xgb_bank_best2.fit(train_data, y_train)

# predict

y_pred_xgb2 = xgb_bank_best1.predict(test_data)


#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb2)

print("confusion matrix: \n",confusion_matrix)

# accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb2))

# precision

print("precision", metrics.precision_score(y_test, y_pred_xgb2))

# recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_xgb2))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb2))


#Regularization using GridSearchCV - 3rd Iteration


params3 = {

    'gamma':[0.3,0.35,0.4,0.45]
```

```
}

model_cv_xgb3 = GridSearchCV(estimator = xgb_bank_best2, param_grid = params3,
                        scoring= 'accuracy',
                        cv=3,
                        verbose = 1,
                       return_train_score=True)



model_cv_xgb3.fit(train_data,y_train)

# results of grid search CV
cv_results_xgb3 = pd.DataFrame(model_cv_xgb3.cv_results_)



#parameters best value
best_score_xgb3 = model_cv_xgb3.best_score_
best_xgb3 = model_cv_xgb3.best_params_
best_xgb3

xgb_bank_best3 =
XGBClassifier(colsample_bytree=0.8,learning_rate=0.2,n_estimators=144,subsample=0.82,
                              min_child_weight=6,max_depth=4,gamma=0.4)
xgb_bank_best3.fit(train_data, y_train)
# predict
y_pred_xgb3 = xgb_bank_best3.predict(test_data)



#confusion matrix
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb3)
print("confusion matrix: \n",confusion_matrix)
# accuracy
print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb3))
# precision
print("precision", metrics.precision_score(y_test, y_pred_xgb3))
# recall/sensitivity
print("recall", metrics.recall_score(y_test, y_pred_xgb3))
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb3))

#Regularization using GridSearchCV - 4th Iteration

params4 = {
    'reg_lambda':[1e-2,0.05,0.1]
}

model_cv_xgb4 = GridSearchCV(estimator = xgb_bank_best3, param_grid = params4,
                        scoring= 'accuracy',
                        cv=3,
                        verbose = 1,
                       return_train_score=True)



model_cv_xgb4.fit(train_data,y_train)

# results of grid search CV
cv_results_xgb4 = pd.DataFrame(model_cv_xgb4.cv_results_)
```

```
#parameters best value
best_score_xgb4 = model_cv_xgb4.best_score_
best_xgb4 = model_cv_xgb4.best_params_
best_xgb4


xgb_bank_best4 = XGBClassifier(colsample_bytree=0.8,learning_rate=0.2,n_estimators=144,subsample=0.82,
                               min_child_weight=6,max_depth=4,gamma=0.4,reg_lambda=0.05)
xgb_bank_best4.fit(train_data, y_train)
# predict
y_pred_xgb4 = xgb_bank_best4.predict(test_data)


#confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb4)
print("confusion matrix: \n",confusion_matrix)
# accuracy
print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb4))
# precision
print("precision", metrics.precision_score(y_test, y_pred_xgb4))
# recall/sensitivity
print("recall", metrics.recall_score(y_test, y_pred_xgb4))
#Area under curve
print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb4))
```

```
xgb_bank_best4 =
XGBClassifier(colsample_bytree=0.8,learning_rate=0.2,n_estimators=144,subsample=0.82,
                          min_child_weight=6,max_depth=4,gamma=0.4,reg_lambda=0.05)

xgb_bank_best4.fit(train_data, y_train)

# predict

y_pred_xgb4 = xgb_bank_best4.predict(test_data)



#confusion matrix

confusion_matrix = metrics.confusion_matrix(y_test, y_pred_xgb4)

print("confusion matrix: \n",confusion_matrix)

# accuracy

print("accuracy", metrics.accuracy_score(y_test, y_pred_xgb4))

# precision

print("precision", metrics.precision_score(y_test, y_pred_xgb4))

# recall/sensitivity

print("recall", metrics.recall_score(y_test, y_pred_xgb4))

#Area under curve

print("area-under-curve metric: ", metrics.roc_auc_score(y_test, y_pred_xgb4))


#Feature Importance

gbc_imp_feature=pd.DataFrame(xgb_bank_best4.feature_importances_, columns = ["Imp"],
index = X_train.columns)

gbc_imp_feature.sort_values(by="Imp",ascending=False)[:16]
```

6. Compare performances of all the models and comment on your findings. (5 marks)

Gradient Boost is appearing to be best among all the models as it has an accuracyof 89.6% and precision of 56.4% is good considering data given.

CONCLUSION:

Following can be concluded from the bivariate analysis, the plots and also from the whole case study analysis:

- Gradient Boost is appearing to be best among all the models as it has an accuracy of 89.6% and precision of 56.4% is good considering data given.
- job - blue collar jobs was the most contacted in marketing campaign however they are the least who subscribed it.
- Student have mostly subscribed term deposit, however they are least contacted in marketing campaign
- Married was the most contacted group however they are the least who subscribed it.
- Single are the most who subscribed for term deposit
- Tertiary education mostly subscribed to term deposit
- Faulter/non defaultor- those who subscribed to term deposit is very low. dont believe will add much to our analysis