Problem Statement 1(Predict the customer retention)

One of the topmost banks in Europe wants to understand their customer data in-detail and trying to extract the patterns causing for customer churn (i.e., bank is losing its existing customers). In order to solve this problem, they have retrieved the customer data from the database which contains the information of both churned and active customers.

```
#Importing libraries

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
# Reading the dataset

data=pd.read_csv("BankCustomerChurn.csv")


# Dimensions of the dataset

data.shape
```

```
(10000, 14)
```

```
# Viewing the column names of the dataset

data.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
# Obtaining the first 5 rows
```

```
data.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balan
0	1	15634602	Hargrave	619	France	Female	42	2	0.
1	2	15647311	Hill	608	Spain	Female	41	1	83807.
2	3	15619304	Onio	502	France	Female	42	8	159660.
3	4	15701354	Boni	699	France	Female	39	1	0.
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.

```
# Obtaining data types

data.dtypes
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

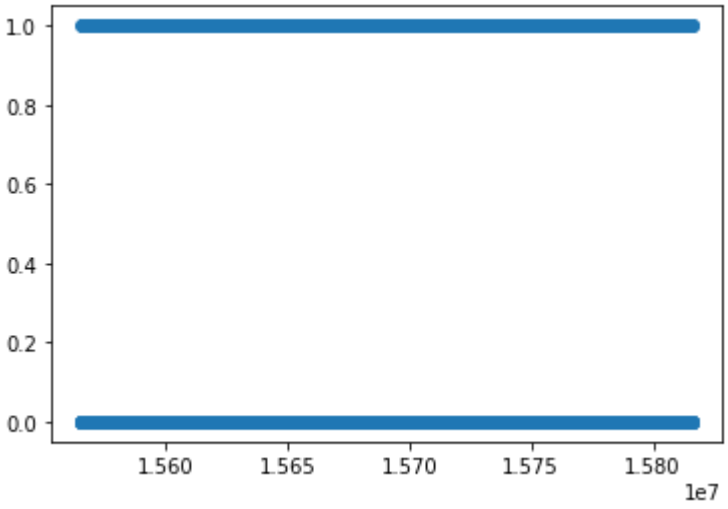
```
# Summary Statistics

data.describe(include='all')
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Ag
count	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.00000
unique	NaN	NaN	2932	NaN	3	2	NaN
top	NaN	NaN	Smith	NaN	France	Male	NaN
freq	NaN	NaN	32	NaN	5014	5457	NaN

```
# "Exited" is the dependent variable

y=np.array(data['Exited'])
```

```
# Scatter plot between CustomerID and the dependent variable

plt.scatter(data['CustomerId'],y)
```

```
<matplotlib.collections.PathCollection at 0x7fdbb7b2e048>
```




```
# Scatter plot between CreditScore and the dependent variable


plt.scatter(data['CreditScore'],y)
```

```
<matplotlib.collections.PathCollection at 0x7fdbb76761d0>
```


According to the scatter plots, it can be concluded that the dependent variable is categorical.


```
# Obtaining the variables having unique values less than 10

ins=[]
for a in data.columns:
  if(data[a].nunique()<10):
    ins.append(a)
    print(a,data[a].nunique())
    print(data[a].unique())
```

```
Geography 3
['France' 'Spain' 'Germany']
Gender 2
['Female' 'Male']
NumOfProducts 4
[1 3 2 4]
HasCrCard 2
[1 0]
IsActiveMember 2
[1 0]
Exited 2
[1 0]
```

```
# Converting the required attributes into categorical

for col in ['Geography','Gender','HasCrCard','IsActiveMember','Exited','CustomerId','RowNumbe
  data[col]=data[col].astype('category')
```

```
# Checking to see if type has been converted

data.dtypes
```

```
RowNumber          category
CustomerId         category
Surname              object
CreditScore           int64
Geography          category
Gender             category
Age                   int64
Tenure                int64
Balance             float64
NumOfProducts         int64
HasCrCard          category
IsActiveMember     category
EstimatedSalary     float64
Exited             category
dtype: object
```

```
# Dropping duplicate values

data_unique=data.drop_duplicates(keep='first')


# Dimensions of dataset

data_unique.shape
```

```
(10000, 14)
```

There are no duplicate values as the dimensions have not changed.

```
# Identifying missing values

data.isna().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
#taking all independent variables into x

x = data[['CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary']]


# taking dependent variable "Exited" into y

y = data[['Exited']]


# Printing the shape of x and y

print(x.shape)
print(y.shape)
```

```
(10000, 10)
```

```
#import libraries to build decision tree

from sklearn.tree import DecisionTreeClassifier


#obtain the dummies and put them in x1

x1 = pd.get_dummies(x)


# printing first few rows of x1

x1.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Geography_Franc
0	619	42	2	0.00	1	101348.88	
1	608	41	1	83807.86	1	112542.58	
2	502	42	8	159660.80	3	113931.57	
3	699	39	1	0.00	2	93826.63	
4	850	43	2	125510.82	1	79084.10	

```
# we need to split into train and test

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x1,y,test_size=0.2,random_state=123)


y.head()
```

	Exited
0	1
1	0
2	1
3	0
4	0

```
# Printing  dimensions of train and test data of x and y

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
```

```
print(y_train.shape)
print(y_test.shape)
```

```
(8000, 15)
(2000, 15)
(8000, 1)
(2000, 1)
```

Defining the model with depth 3 (gini index)

```
dtc1 = DecisionTreeClassifier(max_depth=3)


#fit the model of x_train and y_train

dtc1.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
dtc1.predict(x_train)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#lets store x_train predictions in a variable y_train_pred

y_train_pred=dtc1.predict(x_train)


#lets store x_test predictions in a variable y_test_pred

y_test_pred = dtc1.predict(x_test)


print(y_train_pred)
print(y_test_pred)
```

```
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
```

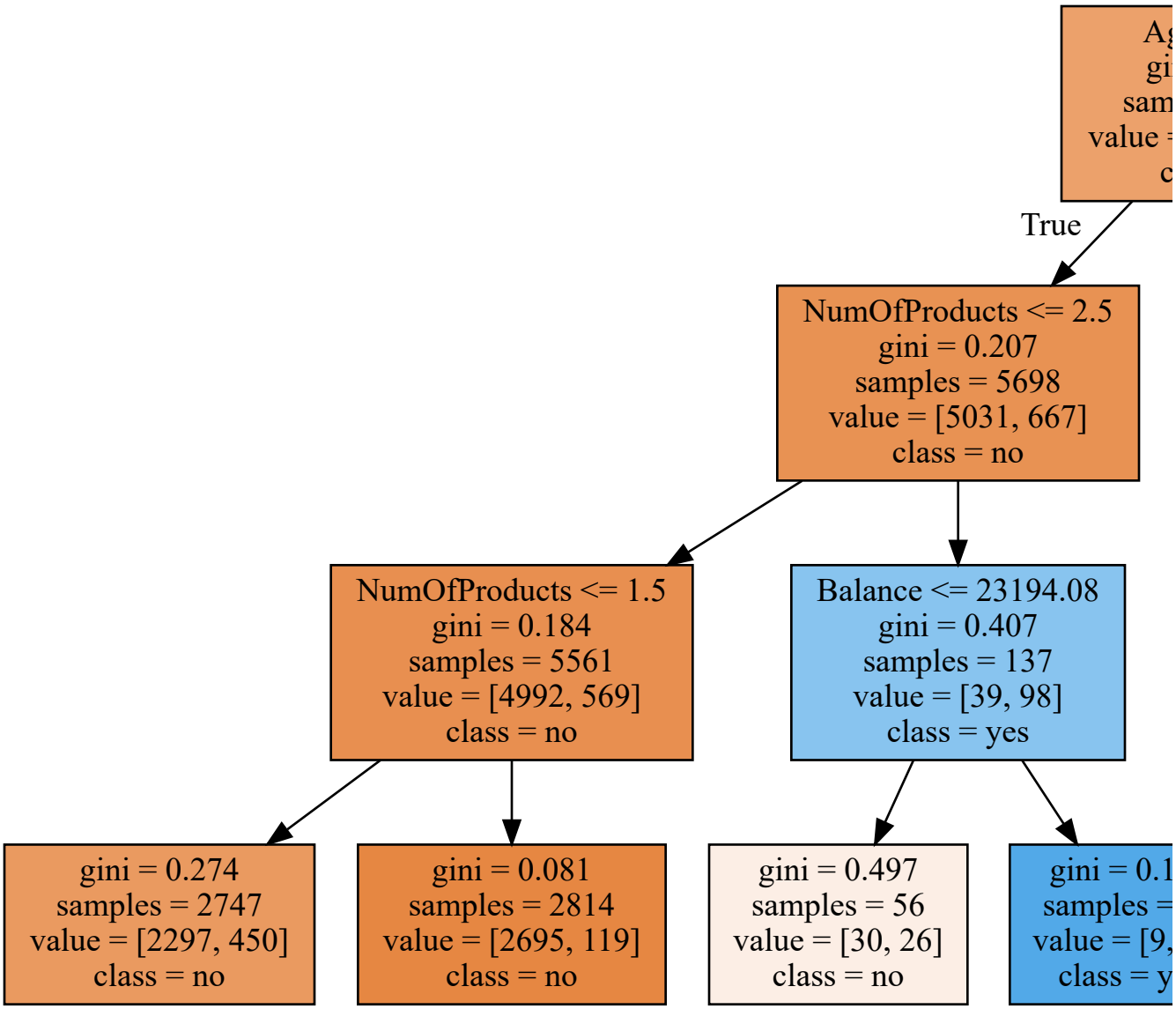
```
# importing libraries to find accuracy score

from sklearn.metrics import accuracy_score


accuracy_score(y_train,y_train_pred)
```

```
0.8425
```

```
accuracy_score(y_test,y_test_pred)
```

```
0.837
```

```
# importing libraries to draw graph

import graphviz
from sklearn.tree import export_graphviz


graphviz.Source(export_graphviz(dtc1,
                                out_file=None,
                                feature_names=x_train.columns,
                                class_names=["no","yes"],
                                filled=True))
```



Defining the model with depth 3 (entropy)

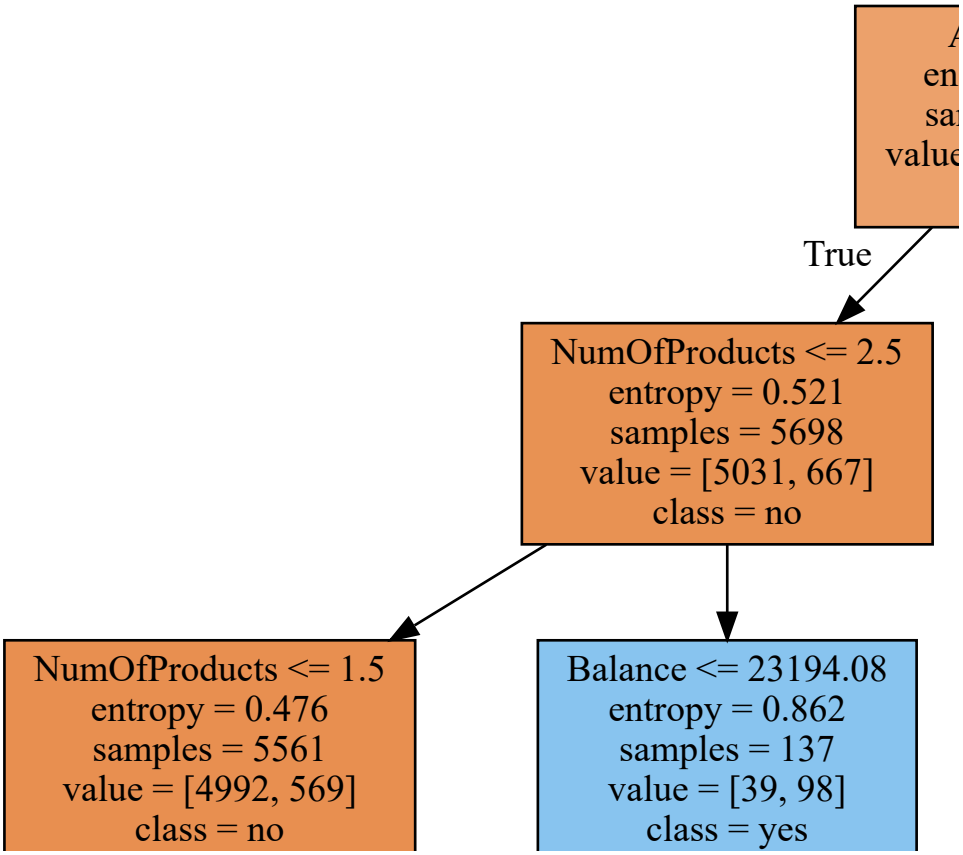
```
dtc2 = DecisionTreeClassifier(max_depth=3,criterion='entropy')



#fit the model of x_train and y_train

dtc2.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
#lets store x_train predictions in a variable y2_train_pred
#lets store x_test predictions in a variable y2_test_pred

y2_train_pred = dtc2.predict(x_train)
y2_test_pred = dtc2.predict(x_test)


# Obtaining accuracy scores of train and test data

print(accuracy_score(y_test,y2_test_pred))
print(accuracy_score(y_train,y2_train_pred))
```

```
0.837
0.840625
```

```
# importing libraries to draw graph

graphviz.Source(export_graphviz(dtc2,
                                out_file=None,
                                feature_names=x_train.columns,
                                class_names=["no","yes"],
                                filled=True))
```



Defining the model with depth 5 (gini index)

```
dtc3 = DecisionTreeClassifier(max_depth=5)
```

```
#fit the model of x_train and y_train

dtc3.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
#lets store x_train predictions in a variable y3_train_pred
#lets store x_test predictions in a variable y3_test_pred

y3_train_pred = dtc3.predict(x_train)
y3_test_pred = dtc3.predict(x_test)


# Obtaining accuracy scores of train and test data

print(accuracy_score(y_test,y3_test_pred))
print(accuracy_score(y_train,y3_train_pred))
```

```
0.857
0.857875
```

```
# importing libraries to draw graph

graphviz.Source(export_graphviz(dtc3,
                                out_file=None,
                                feature_names=x_train.columns,
                                class_names=["no","yes"],
                                filled=True))
```

We have observed that there has been an increase in the accuracy as the depth increases. Also, entropy takes more time to execute when compared to gini index.

Defining the model with depth 5 (entropy)

```
dtc4 = DecisionTreeClassifier(max_depth=5,criterion='entropy')



#fit the model of x_train and y_train

dtc4.fit(x_train,y_train)
```

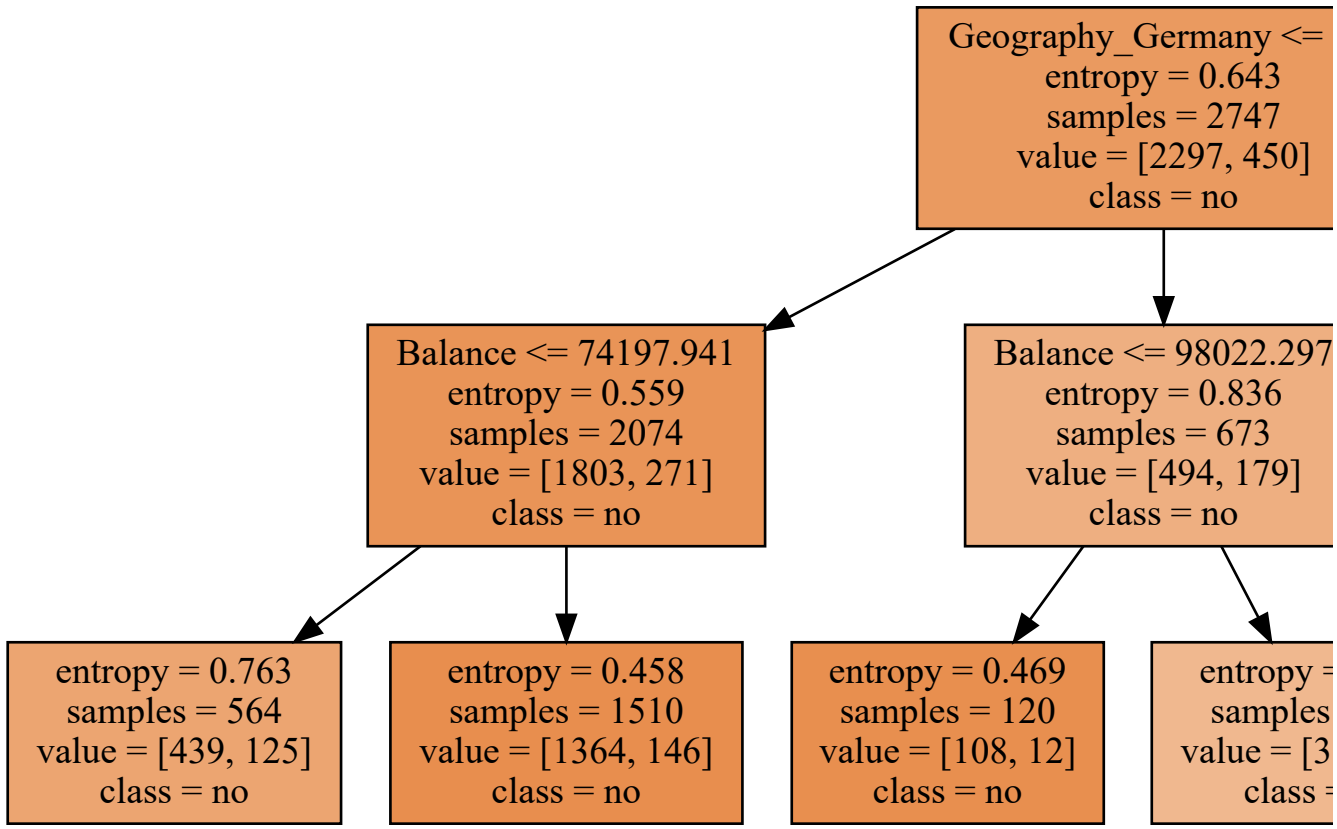
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```


```
#lets store x_train predictions in a variable y4_train_pred
#lets store x_test predictions in a variable y4_test_pred

y4_train_pred = dtc4.predict(x_train)
y4_test_pred = dtc4.predict(x_test)
```

```
# Obtaining accuracy scores of train and test data

print(accuracy_score(y_test,y4_test_pred))
print(accuracy_score(y_train,y4_train_pred))
```

```
0.856
0.857375
```

```
# importing libraries to draw graph

graphviz.Source(export_graphviz(dtc4,
                                out_file=None,
                                feature_names=x_train.columns,
                                class_names=["no","yes"],
                                filled=True))
```



OBSERVATION