Customer Churn Prediction

20-10-2023

Intern Assessment Assignment

Siddharth Agrawal

Siddharthagrawal2408@gmail.com

Description

In this project I have created a predictive based machine learning model that predict the customer churn based on different features in the dataset. Our main goal is to predict the churn of new customer using machine learning model which is trained and tested on similar type of data

About Dataset

Our dataset contains total 9 features

	A	B	C	D	E	F	G	H	I
1	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
2	1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
3	2	Customer_2	62	Female	New York	1	48.76	172	0
4	3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
5	4	Customer_4	36	Female	Miami	3	97.94	297	1
6	5	Customer_5	46	Female	Miami	19	58.14	266	0
7	6	Customer_6	67	Male	New York	15	82.65	456	1
8	7	Customer_7	30	Female	Chicago	3	73.79	269	0
9	8	Customer_8	67	Female	Miami	1	97.7	396	1
10	9	Customer_9	20	Female	Miami	10	42.45	150	1
11	10	Customer_10	53	Female	Los Angeles	12	64.49	383	1

In which churn is our target variable
Dataset contains 3 categorical features that includes gender, name, location
Rest 6 are our numerical data

Importing libraries

IMPORTING THE LIBRAIRES

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import  LabelEncoder,StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    ConfusionMatrixDisplay,
    classification_report,
    confusion_matrix,
)
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV, cross_val_score,train_test_split,RepeatedStratifiedKF
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.preprocessing import OneHotEncoder
```

Loading the dataset

After importing the libraries, we have to load our dataset using pandas and setting index column as customer id

```
[ ] df=pd.read_excel('/content/customer_churn_large_dataset.xlsx',index_col='CustomerID')
```

EDA(Exploratory data analysis)

```
df.head()
```

CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
2	Customer_2	62	Female	New York	1	48.76	172	0
3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
4	Customer_4	36	Female	Miami	3	97.94	297	1
5	Customer_5	46	Female	Miami	19	58.14	266	0

```
df.tail()
```

CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
99996	Customer_99996	33	Male	Houston	23	55.13	226	1
99997	Customer_99997	62	Female	New York	19	61.65	351	0
99998	Customer_99998	64	Male	Chicago	17	96.11	251	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 1 to 100000
Data columns (total 8 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Name                        100000 non-null  object
 1   Age                         100000 non-null  int64
 2   Gender                      100000 non-null  object
 3   Location                    100000 non-null  object
 4   Subscription_Length_Months  100000 non-null  int64
 5   Monthly_Bill                100000 non-null  float64
 6   Total_Usage_GB              100000 non-null  int64
 7   Churn                       100000 non-null  int64
dtypes: float64(1), int64(4), object(3)
memory usage: 6.9+ MB
```

Here we can see we have 1 to 10000 entries and data type of each features

```
df.describe()
```

	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	44.027020	12.490100	65.053197	274.393650	0.497790
std	15.280283	6.926461	20.230696	130.463063	0.499998
min	18.000000	1.000000	30.000000	50.000000	0.000000
25%	31.000000	6.000000	47.540000	161.000000	0.000000
50%	44.000000	12.000000	65.010000	274.000000	0.000000
75%	57.000000	19.000000	82.640000	387.000000	1.000000
max	70.000000	24.000000	100.000000	500.000000	1.000000

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

```
Name                          0
Age                           0
Gender                        0
Location                      0
Subscription_Length_Months    0
Monthly_Bill                  0
Total_Usage_GB                0
Churn                         0
dtype: int64
```

NO NULL VALUES!!!

```
df.duplicated().sum()
```

```
0
```

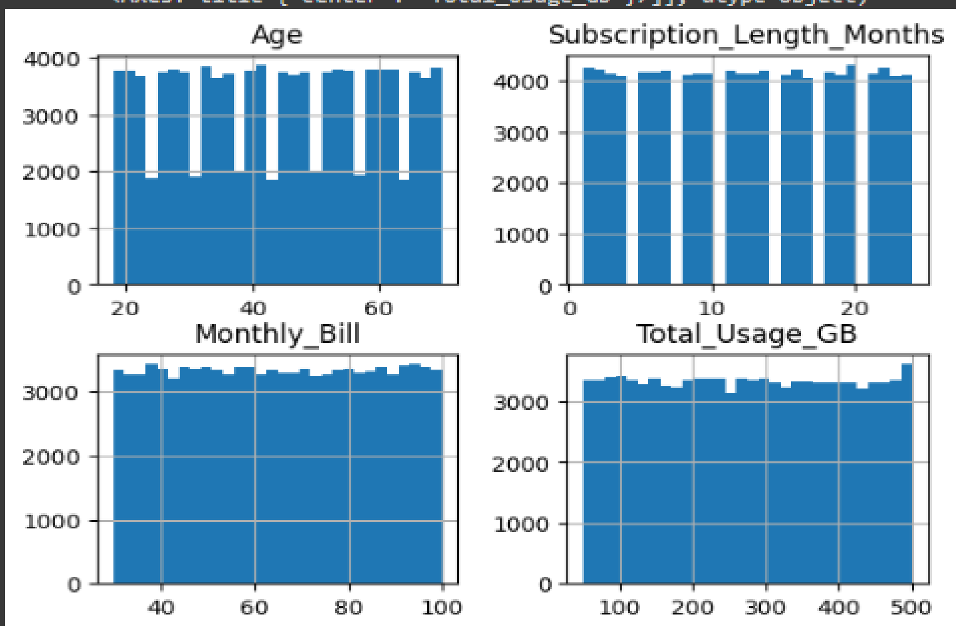
```
df.corr(numeric_only=True)
```

	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
Age	1.000000	0.003382	0.001110	0.001927	0.001559
Subscription_Length_Months	0.003382	1.000000	-0.005294	-0.002203	0.002328
Monthly_Bill	0.001110	-0.005294	1.000000	0.003187	-0.000211
Total_Usage_GB	0.001927	-0.002203	0.003187	1.000000	-0.002842
Churn	0.001559	0.002328	-0.000211	-0.002842	1.000000

Distribution of numerical features

```
cols=['Age','Subscription_Length_Months','Monthly_Bill', 'Total_Usage_GB',]
df[cols].hist(bins=30)
```

```
array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Subscription_Length_Months'}>],
       [<Axes: title={'center': 'Monthly_Bill'}>,
        <Axes: title={'center': 'Total_Usage_GB'}>]], dtype=object)
```

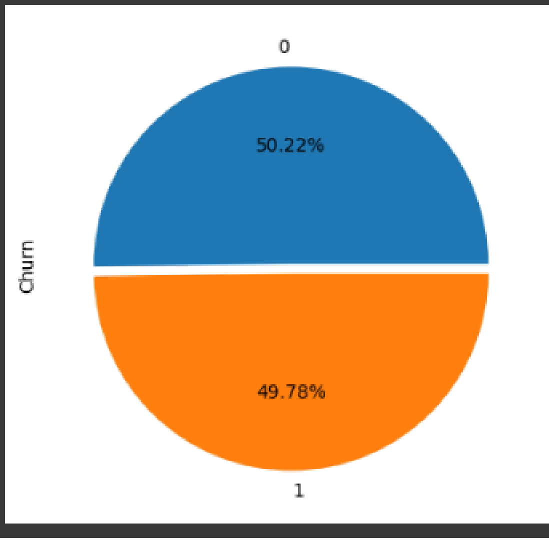


Distribution of categorical features

```
a1=df.groupby(['Gender','Churn']).apply(lambda x:x['Churn'].count()).reset_index(name='Counts')
print(a1)
```

```
   Gender  Churn  Counts
0  Female      0   25272
1  Female      1   24944
2    Male      0   24949
3    Male      1   24835
```

```
[ ]  (df['Churn'].value_counts()*100.0 /len(df)).plot.pie(autopct='%.2f%%' , explode = [0,0.05]);
```

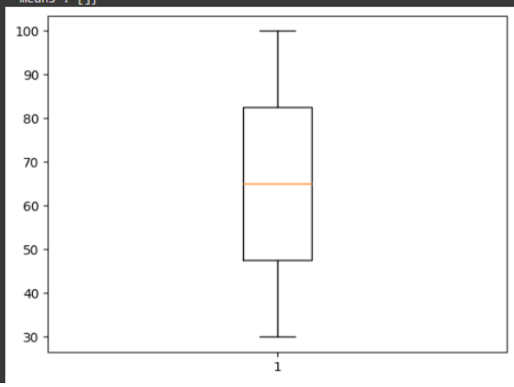


Outliers Analysis with IQR method

```
[ ]  x = ['Subscription_Length_Months','Monthly_Bill','Total_Usage_GB']
     def count_outliers(data,col):
             q1 = data[col].quantile(0.25,interpolation='nearest')
             q2 = data[col].quantile(0.5,interpolation='nearest')
             q3 = data[col].quantile(0.75,interpolation='nearest')
             q4 = data[col].quantile(1,interpolation='nearest')
             IQR = q3 -q1
             global LLP
             global ULP
             LLP = q1 - 1.5*IQR
             ULP = q3 + 1.5*IQR
             if data[col].min() > LLP and data[col].max() < ULP:
                 print("No outliers in",i)
             else:
                 print("There are outliers in",i)
                 x = data[data[col]<LLP][col].size
                 y = data[data[col]>ULP][col].size
                 a.append(i)
                 print('Count of outliers are:',x+y)
     global a
     a = []
     for i in x:
         count_outliers(df,i)

     No outliers in Subscription_Length_Months
     No outliers in Monthly_Bill
     No outliers in Total_Usage_GB
```

```
plt.boxplot(df[['Monthly_Bill']])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x78450d791450>,
  <matplotlib.lines.Line2D at 0x78450d791570>],
 'caps': [<matplotlib.lines.Line2D at 0x78450d791810>,
  <matplotlib.lines.Line2D at 0x78450d791ab0>],
 'boxes': [<matplotlib.lines.Line2D at 0x78450d7911b0>],
 'medians': [<matplotlib.lines.Line2D at 0x78450d791d50>],
 'fliers': [<matplotlib.lines.Line2D at 0x78450d791ff0>],
 'means': []}
```



Cleaning/Removing irrelevant features

Removing the customer's name features as it is useless

```
[ ] df.drop(['Name'],axis = 1,inplace = True)
```

```
df.head()
```

CustomerID	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
1	63	Male	Los Angeles	17	73.36	236	0
2	62	Female	New York	1	48.76	172	0
3	24	Female	Los Angeles	5	85.47	460	0
4	36	Female	Miami	3	97.94	297	1
5	46	Female	Miami	19	58.14	266	0

```
[ ] df.columns
```

```
Index(['Age', 'Gender', 'Location', 'Subscription_Length_Months',
       'Monthly_Bill', 'Total_Usage_GB', 'Churn'],
      dtype='object')
```

One Hot Encoding

Used one hot Encoding to convert categorical to numerical data and then rearranging the columns

On Hot Encoding:for transforming categorical data to numeric

```
[ ] df1=pd.get_dummies(data=df,columns=['Gender','Location'],drop_first=True)
```

```
df1.head()
```

CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	Gender_Male	Location_Houston	Location_Los Angeles	Location_Miami	Location_New York
1	63	17	73.36	236	0	1	0	1	0	0
2	62	1	48.76	172	0	0	0	0	0	1
3	24	5	85.47	460	0	0	0	1	0	0
4	36	3	97.94	297	1	0	0	0	1	0
5	46	19	58.14	266	0	0	0	0	1	0

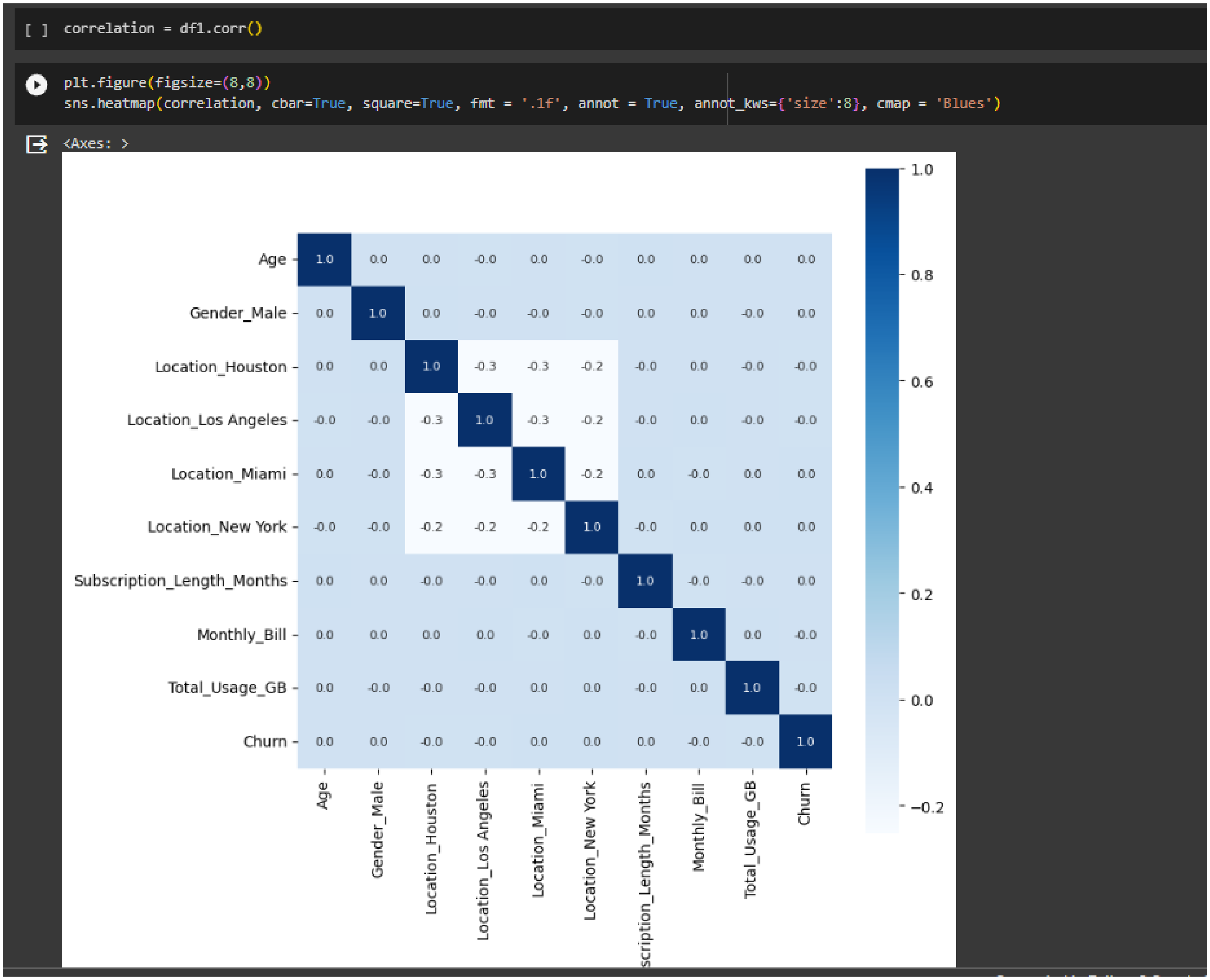
```
[ ] df1.columns

Index(['Age', 'Subscription_Length_Months', 'Monthly_Bill', 'Total_Usage_GB',
       'Churn', 'Gender_Male', 'Location_Houston', 'Location_Los Angeles',
       'Location_Miami', 'Location_New York'],
      dtype='object')
```

```
[ ] df1=df1[['Age','Gender_Male', 'Location_Houston', 'Location_Los Angeles','Location_Miami', 'Location_New York','Subscription_Length_Months', 'Monthly_Bill', 'Total_Usage_GB','Churn']]
```

```
[ ] df1.head()
```

CustomerID	Age	Gender_Male	Location_Houston	Location_Los Angeles	Location_Miami	Location_New York	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
1	63	1	0	1	0	0	17	73.36	236	0
2	62	0	0	0	0	1	1	48.76	172	0
3	24	0	0	1	0	0	5	85.47	460	0
4	36	0	0	0	1	0	3	97.94	297	1
5	46	0	0	0	1	0	19	58.14	266	0

Correlation between features



Feature Scaling

As we can see from above correlation matrix and distributions of inputs our dataset is not very sparse so we don't need to apply any features scaling

Feature selection

So, we store all columns accept target in X and only target in Y

Feature selection

```
X = df1.drop('Churn',axis='columns')
y = df1['Churn']
print('Shape of X:',X.shape)
print('Shape of y:',y.shape)
```

```
Shape of X: (100000, 9)
Shape of y: (100000,)
```

Splitting on dataset in Train and test

We have split the dataset in 80% in training and 20% in testing

Spliting of Dataset in Train and Test

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=5)
print('Shape of X_train:',X_train.shape)
print('Shape of X_test:',X_test.shape)
print('Shape of y_train:',y_train.shape)
print('Shape of y_test:',y_test.shape)
```

```
Shape of X_train: (80000, 9)
Shape of X_test: (20000, 9)
Shape of y_train: (80000,)
Shape of y_test: (20000,)
```

Prediction using Different models

During the period of the training part. I have used multiple algorithm for training the data like logistic regression, decision tree, Random forest, Artificial neural network and support vector machine.
For details refer the .py file attached together
So among different models decision tree is giving optimal answer

Prediction using Decision Tree Classifier

- **Fitting the model**

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

dtc.fit(X_train, y_train)
```

- **Predict values using X-test**

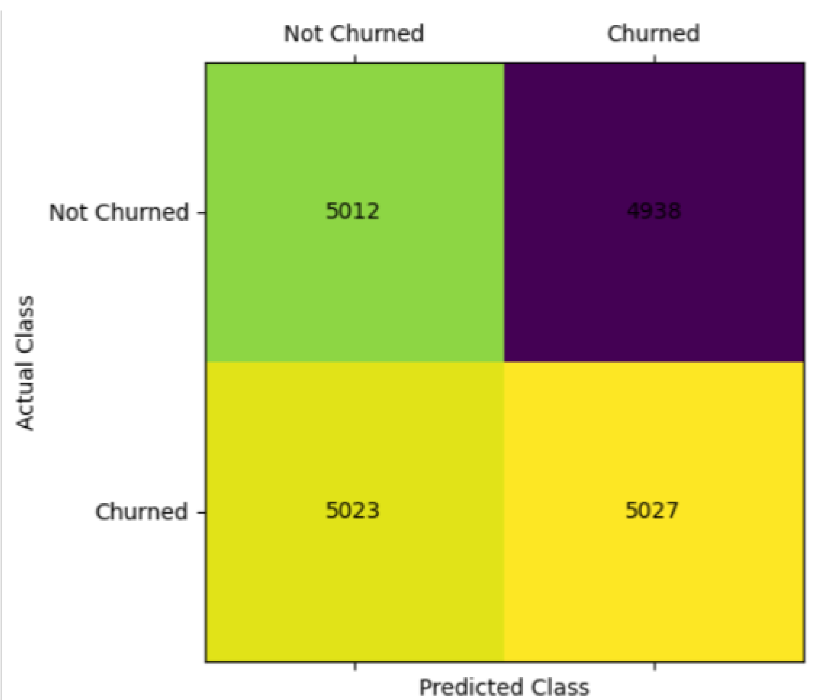
```
pickle.dump(dtc, open('dtc.pickle','wb'))
y_pred_dtc = dtc.predict(X_test)
```

- **Accuracy score / classification report**

```
print(classification_report(y_test, y_pred_dtc))

              precision    recall  f1-score   support

           0       0.50      0.50      0.50      9950
           1       0.50      0.50      0.50     10050

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.50     20000
weighted avg       0.50      0.50      0.50     20000
```

```
[ ] accuracy_score(y_test, y_pred_dtc)

0.50195
```

- **Confusion matrix**



Hyper Parameter Tuing in Decision tree

This process of calibrating our model by finding the right hyperparameters to generalize our model is called Hyperparameter Tuning. In this project I have used for loop to find the optimal value for parameter of decision tree called max_depth

```
[ ] dtc.tree_.max_depth

58
```

```
for max_d in range(1,50):
  model = DecisionTreeClassifier(max_depth=max_d, random_state=42)
  model.fit(X_train,y_train)
  y_pred_ldtc=model.predict(X_test)
  print('The Training Accuracy for max_depth {} is:'.format(max_d), accuracy_score(y_test,y_pred_ldtc))
  print('')
```

```
The Training Accuracy for max_depth 1 is: 0.49695

The Training Accuracy for max_depth 2 is: 0.49695

The Training Accuracy for max_depth 3 is: 0.49595

The Training Accuracy for max_depth 4 is: 0.4987

The Training Accuracy for max_depth 5 is: 0.50195

The Training Accuracy for max_depth 6 is: 0.50095
```

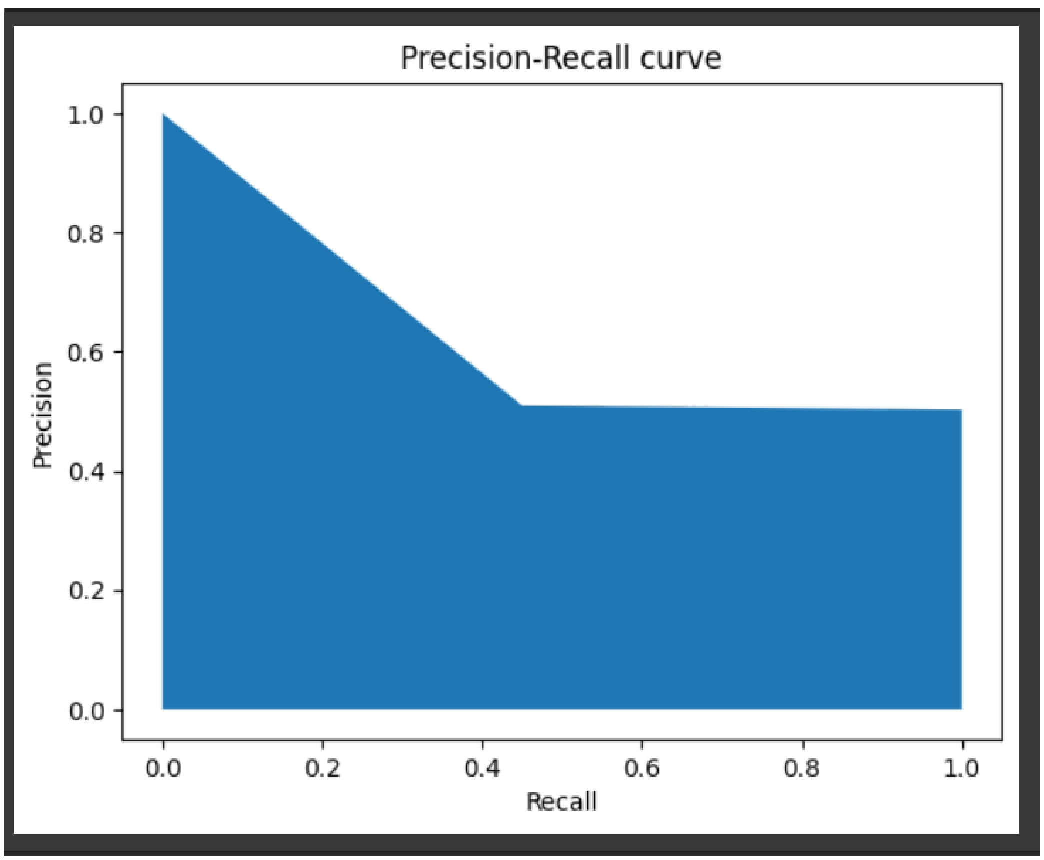
So by doing this I have determined the value should be 18 of max_depth.

```
dtc = DecisionTreeClassifier(max_depth=18)

dtc.fit(X_train.values, y_train.values)

pickle.dump(dtc, open('dtc.pickle','wb'))
y_pred_dtc = dtc.predict(X_test.values)

print(accuracy_score(y_test,y_pred_dtc))
```

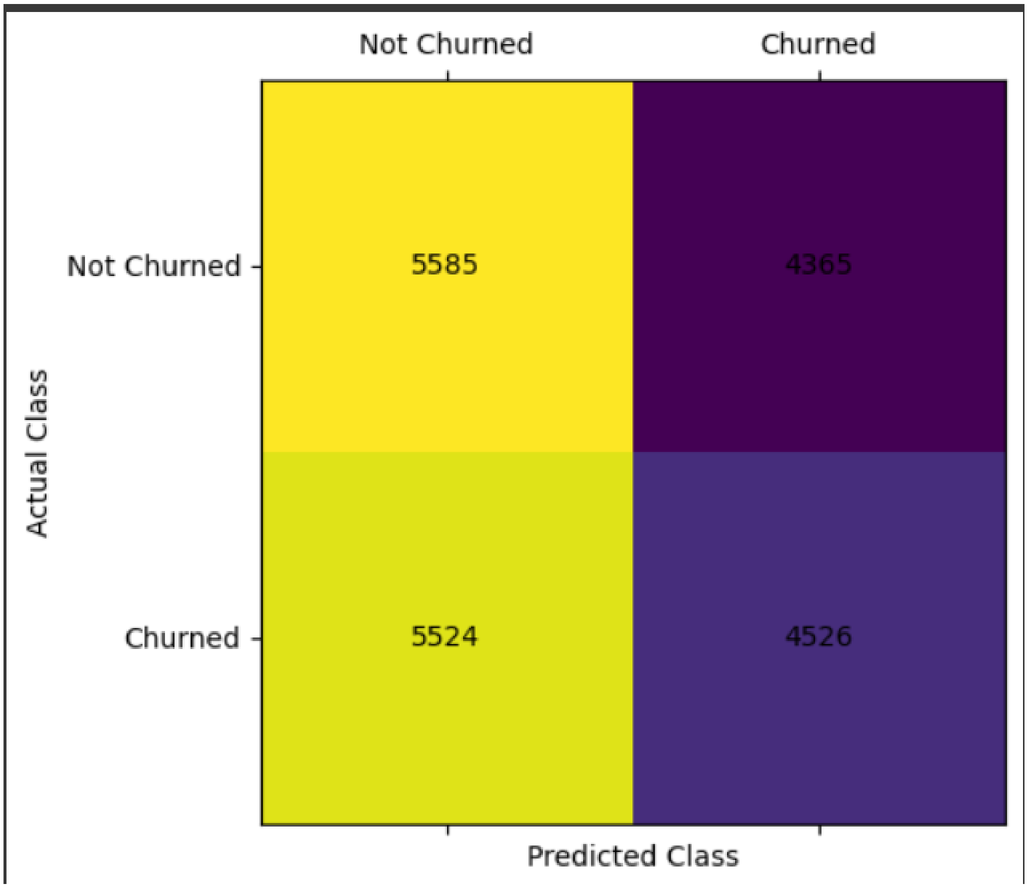
```
0.50595
```

So our model accuracy is increased by 0.04%   which is not very significant improvement but still its works

PR-CURVE



Confusion matrix



Model Deployment

As now we have trained and tested our model its time to deploy our model So I have used pickle so that our model can predict values for new input

```
[ ] import pickle
    import numpy as np

    lc=pickle.load(open('/content/dtc.pickle','rb'))
    ls=pickle.load(open('/content/scaler_dtc.pickle','rb'))

    new_pred=lc.predict(np.array([[27,1,1,0,0,0,2,59.82,364]]))
    new_pred

    array([1])
```

CONCLUSION

After deployment of the model we can predict the output of new input. Here to conclude I have used different model and find the best model for this dataset and implemented it.