PROJECT REPORT

Customer Attrition Classification on Telco Sector

Submitted towards the partial fulfillment of the criteria for award of PGA by Imarticus

Submitted By:
Aravindkumar R

Course and Batch: PGA 20 –Feb 2022

Abstract

Customer churn or attrition is a crucial problem and one among the foremost principal concerns for giant telecom companies. Thanks to the undeviating effect on the revenues of the telecom companies, they're seeking to evolve the means to predict a customer to churn. Therefore, finding factors that increase customer attrition may be a major issue to require necessary actions to scale back this attrition. The principal contribution is to develop a customer attrition prediction model which helps telecom operators to predict customers who are subjected to churn from that operator.

In this project, I created a model that can find whether a customer will churn or not. With the help of this model we can solve some of the Business Challenges pertaining to customer attrition like the likelihood of an existing customer leaving and identifying the indicators for churn and impose some new retention strategies to diminish the prospective of customer churn in near future.

Acknowledgement

I’m using this opportunity to express my gratitude to everyone who supported me throughout the course of this project. I’m thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I’m sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills.

I wish to thank all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

I certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date:

Place: Chennai ARAVINDKUMAR R

Table of Contents

Chapter 1

Introduction

Title & Objective of the Study:

This dataset is related to telecom sector IBM's customers attrition classification. Telephone operator companies, Internet service providers, TV broadcasting channel companies, often use customer attrition or churn analysis and customer attrition rates are one among the major business metrics because the value of retaining an existing customer is way but acquiring a replacement one. Companies from these sectors often have customer service branches which strive to get back turn against clients, because recuperated long-term customers are often worth far more to a corporation than newly joined clients. To build a better model, I will be using few Supervised Machine Learning classification algorithms and performance improvement techniques to create a predictive modeling which can be used to in relation to my objective.

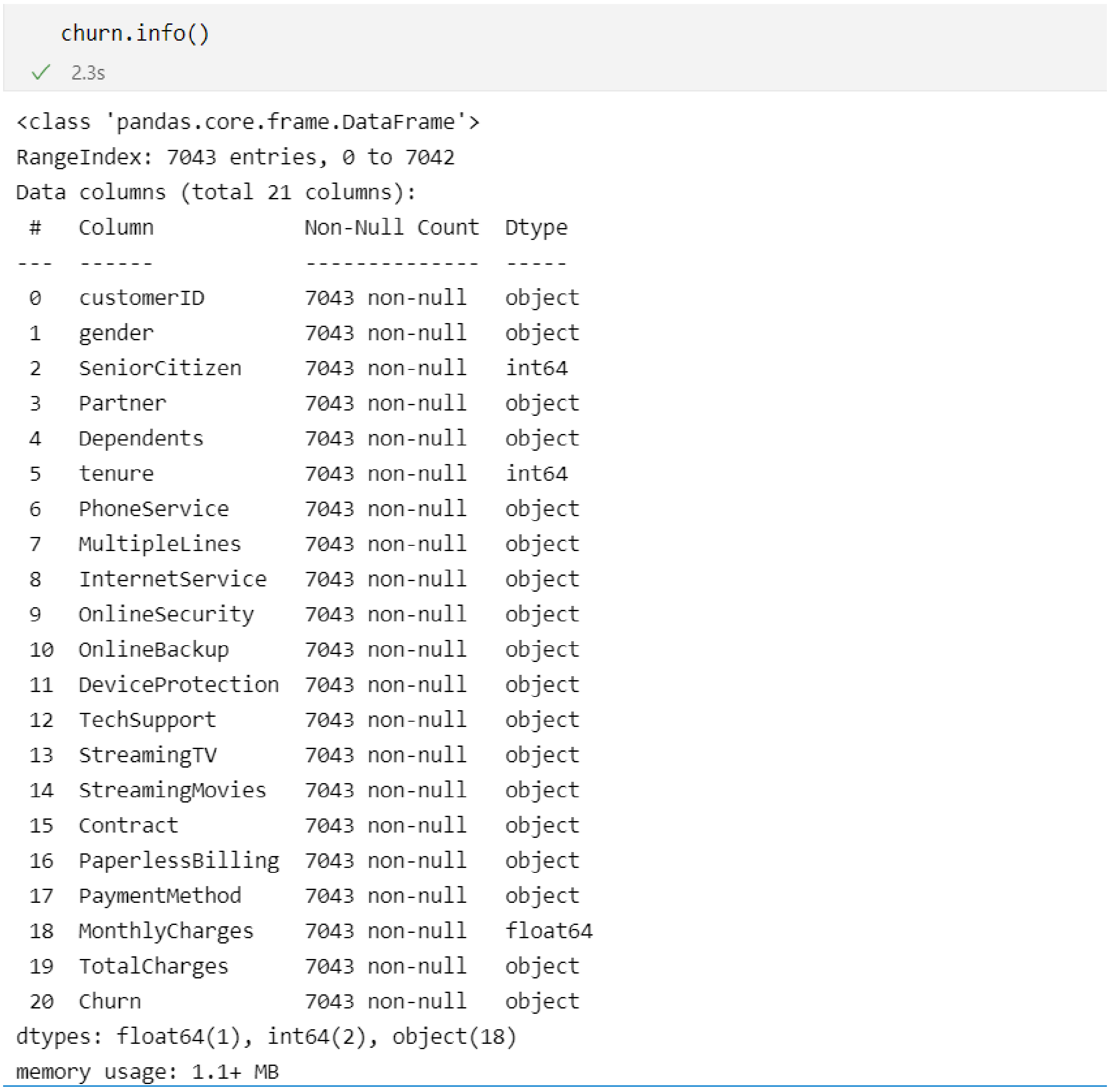
Need of the study

IBM being a Telco giant has its own strategy and knows what it takes to find a new customer than to make an existing customer retain. Customer churn is one of the biggest expenditure of any organization and one among the foremost principal concerns for giant telecom companies and if we are able figure out why a customer leaves and when they leave with reasonable accuracy, it would immensely help the organization to strategize their retention initiatives.

Here by solving this problem we can solve some of the Business Challenges pertaining to customer attrition like the likelihood of an existing customer leaving and identifying the indicators for churn and impose some new retention strategies to diminish the prospective of customer churn in near future

Data Description

Attributes	Description	Type of Data
Customer ID	Customer Identification Number	Numerical
Gender	Whether the customer is a male or a female	Categorical
Senior Citizen	Whether the customer is a senior citizen or not	Yes, No
Partner	Whether the customer has a partner or not	Yes, No
Dependents	Whether the customer has dependents or not	Yes, No
Tenure	No of months the customer has stayed with the company	Numerical
Phone Service	Whether the customer has a phone service or not	Yes, No
Multiple Lines	Whether the customer has multiple lines or not	Yes, No, NPC
Internet Service	Customer's internet service provider	DSL, Fiber optic, No
Online Security	Whether the customer has online security or not	Yes, No, NIS
Online Backup	Whether the customer has online backup or not	Yes, No, NIS
Device Protection	Whether the customer has device protection or not	Yes, No, NIS
Tech Support	Whether the customer has tech support or not	Yes, No, NIS
Streaming TV	Whether the customer has streaming TV or not	Yes, No, NIS
Streaming Movies	Whether the customer has streaming movies or not	Yes, No, NIS
Contract	The contract term of the customer	Per Month, 1 or 2 year
Paperless Billing	Whether the customer has paperless billing or not	Yes, No
Payment Method	The customer's payment method	EC, MC, BT, CC
Monthly Charges	The amount charged to the customer monthly	Numerical
Total Charges	The total amount charged to the customer	Numerical
Churn	Whether the customer churned or not	Yes, No



```
churn.info()
```

✓ 2.3s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Data Source: (Kaggle.com)

https://www.kaggle.com/blastchar/telco-customer-churn?select=WA_Fn-UseC_-Telco-Customer-Churn.csv

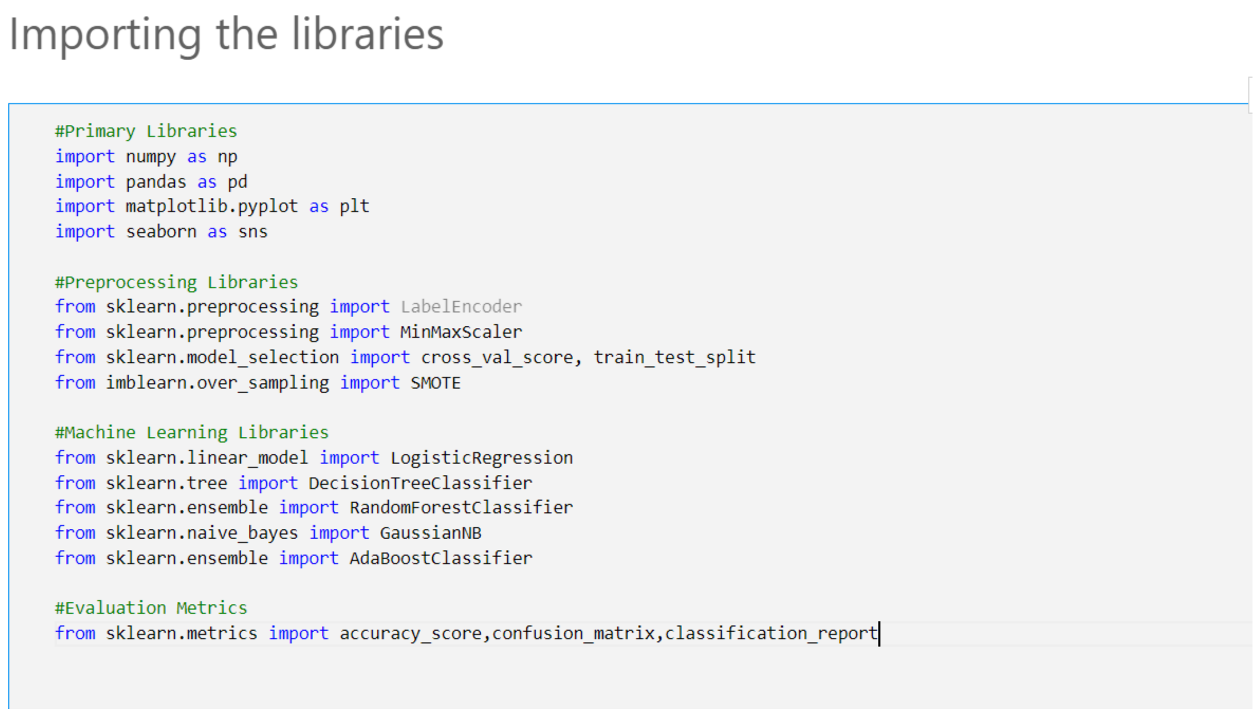
Tools & Techniques:

- ✓ Python Libraries: NumPy, Pandas, Scikit-Learn
- ✓ Data Visualization: Matplotlib, Seaborn
- ✓ Jupyter Notebook, VS code
- ✓ Techniques: Imbalancing Technique(SMOTE)

Chapter 2

Data Exploring & Analyzing

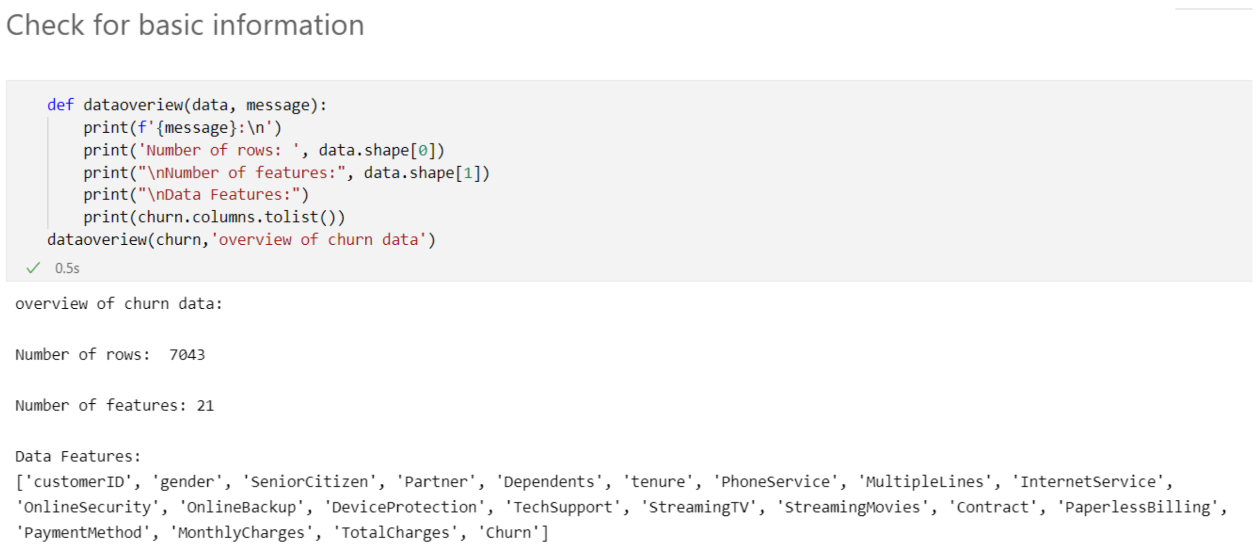
2.1 Data view



Importing the libraries

```python
#Primary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Preprocessing Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score, train_test_split
from imblearn.over_sampling import SMOTE

#Machine Learning Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier

#Evaluation Metrics
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

Data Dimension



Check for basic information

```python
def dataoveriew(data, message):
    print(f'{message}:\n')
    print('Number of rows: ', data.shape[0])
    print("\nNumber of features:", data.shape[1])
    print("\nData Features:")
    print(churn.columns.tolist())
dataoveriew(churn,'overview of churn data')
```

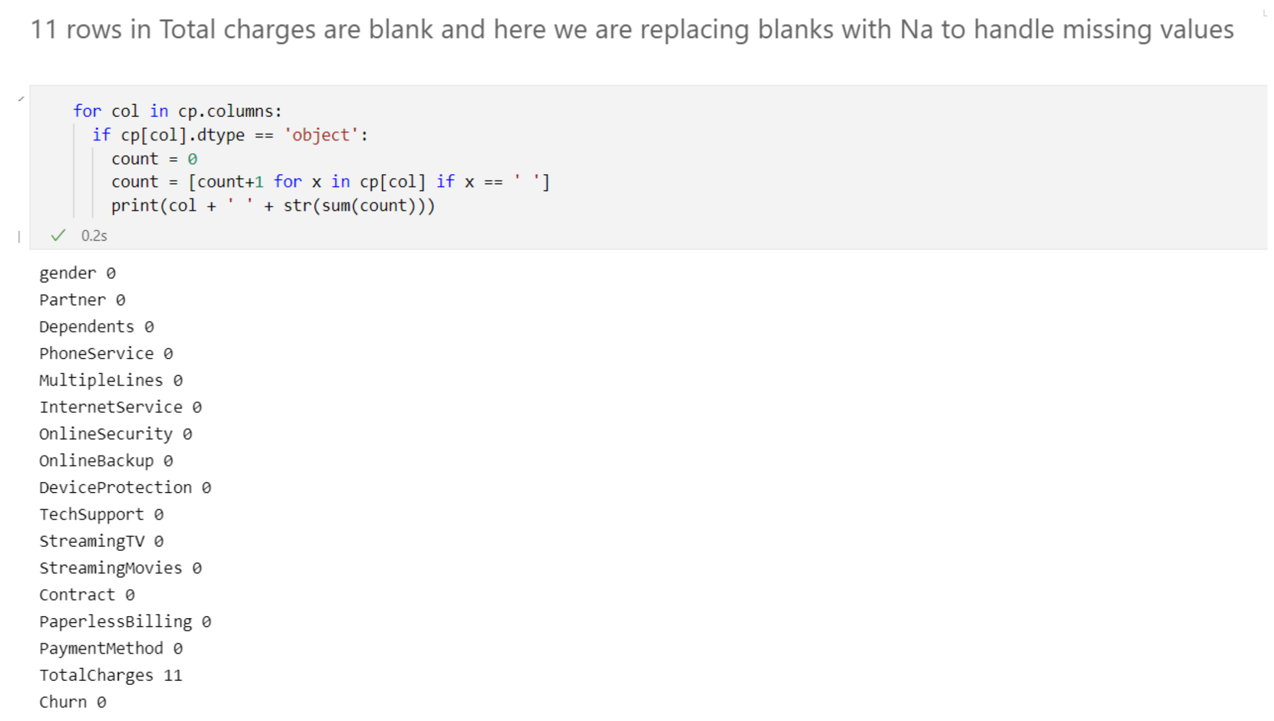
✓ 0.5s

```
overview of churn data:

Number of rows:  7043

Number of features: 21

Data Features:
['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn']
```

2.2 Data Analysis

Missing value Analyses:

There are missing values as “ blank spaces ” in the total charges column and I have found the number of blank spaces totally found in the dataset.



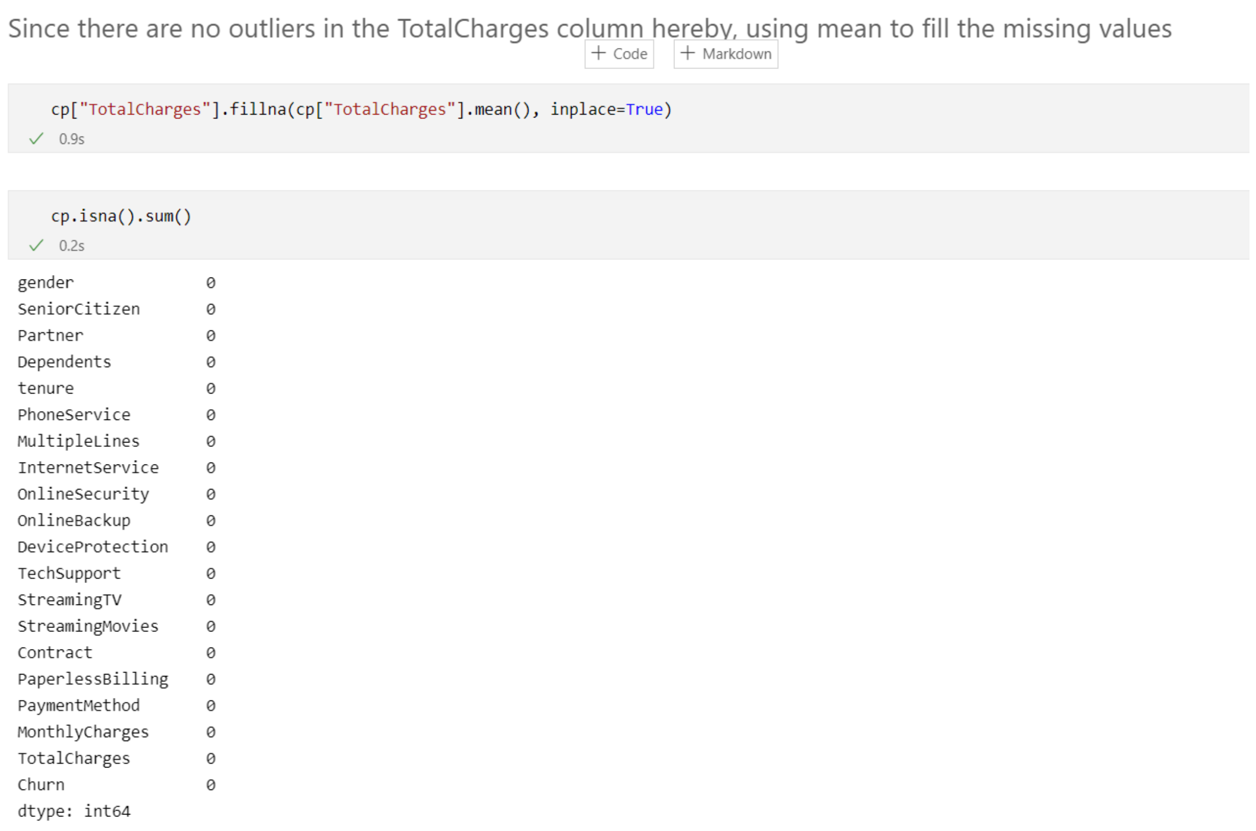
11 rows in Total charges are blank and here we are replacing blanks with Na to handle missing values

```
for col in cp.columns:
    if cp[col].dtype == 'object':
        count = 0
        count = [count+1 for x in cp[col] if x == ' ']
        print(col + ' ' + str(sum(count)))
```

✓ 0.2s

```
gender 0
Partner 0
Dependents 0
PhoneService 0
MultipleLines 0
InternetService 0
OnlineSecurity 0
OnlineBackup 0
DeviceProtection 0
TechSupport 0
StreamingTV 0
StreamingMovies 0
Contract 0
PaperlessBilling 0
PaymentMethod 0
TotalCharges 11
Churn 0
```

Now, after finding the number of blank spaces, I have applied mean to impute those blank spaces since there were no outlier in that column.



Since there are no outliers in the TotalCharges column hereby, using mean to fill the missing values

```
cp["TotalCharges"].fillna(cp["TotalCharges"].mean(), inplace=True)
```

✓ 0.9s

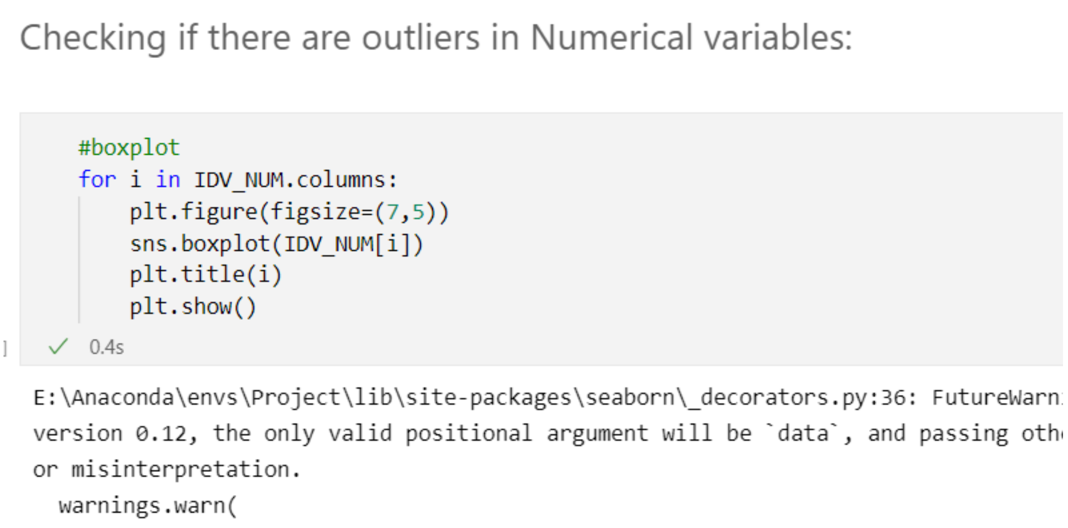
```
cp.isna().sum()
```

✓ 0.2s

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

Outlier Analysis:

Now, after imputing the missing values, we should treat if there are any outliers in our numerical columns.

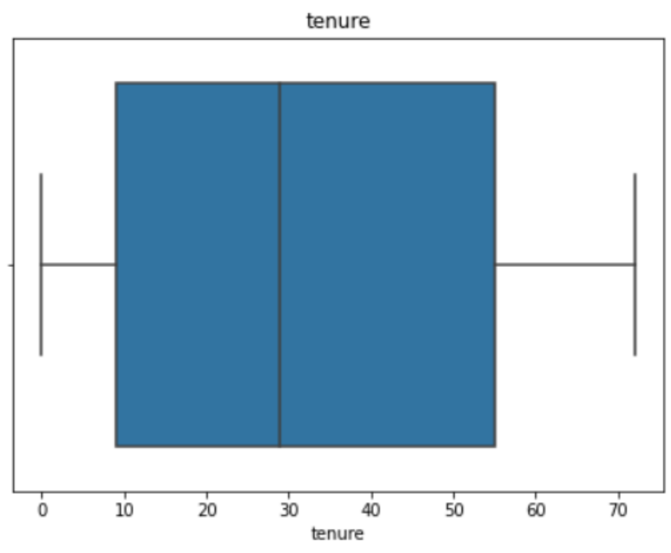


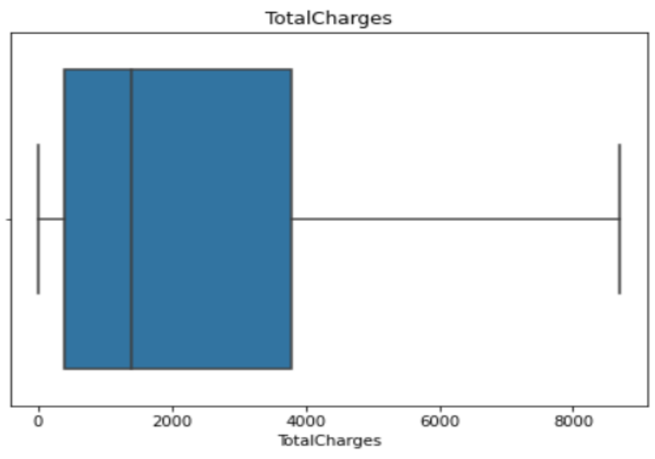
Checking if there are outliers in Numerical variables:

```
#boxplot
for i in IDV_NUM.columns:
    plt.figure(figsize=(7,5))
    sns.boxplot(IDV_NUM[i])
    plt.title(i)
    plt.show()
```

✓ 0.4s

```
E:\Anaconda\envs\Project\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
version 0.12, the only valid positional argument will be `data`, and passing oth
or misinterpretation.
  warnings.warn(
```





From the boxplot it is clear that there are no outliers in our numerical columns.

Feature Analyses:

Note:

*) There are 17 categorical variables in the dataset.
*) There are 3 Numerical variables in the dataset.

The 19 Independent variable are categorized into 3 groups:

Demographic customer information:

Gender , SeniorCitizen , Partner , Dependents

Services that each customer has signed up for:

PhoneService , MultipleLines , InternetService , OnlineSecurity , OnlineBackup , DeviceProtection , TechSupport , StreamingTV , StreamingMovies

Customer account information:

Tenure , Contract , PaperlessBilling , PaymentMethod , MonthlyCharges , TotalCharge

Analyses on Dependent Variable:

Analysis on target dependent variable- Churn

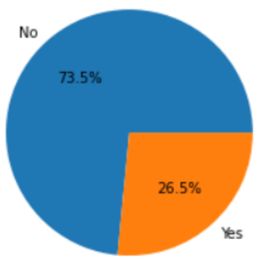
```
target = DV.value_counts()
target = target.reset_index()
target = target.rename(columns={'index': 'Category'})
target
```

✓ 0.2s

	Category	Churn
0	No	5174
1	Yes	1869

```
DV_fig=plt.pie(target['Churn'],labels=target["Category"],autopct='%1.1f%%')
```

✓ 0.2s



Inference:

This above pie chart helps us to understand the churn rate of the customers, which is provided in terms of percentage as shown in the pie chart.

Analyses on Categorical Variables:

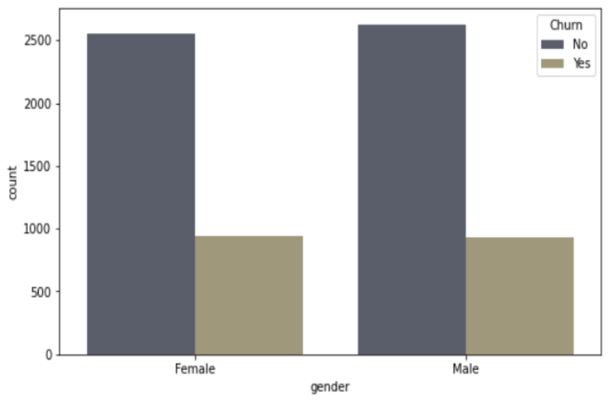
<u>Demographic customer information</u>

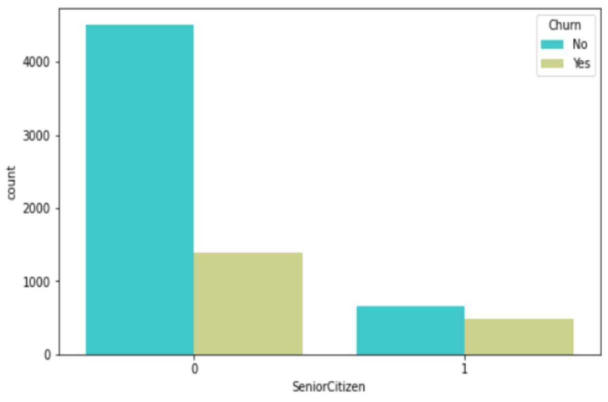
Analyzing Gender and Senior Citizen

```
B1a=df['gender'].value_counts(normalize=True)
print(B1a)

B2a=df['SeniorCitizen'].value_counts(normalize=True)
print(B2a)


fig, g = plt.subplots(1, 2, figsize=(19,5))


B1a=sns.countplot(x='gender',data=df,ax=g[0],hue='Churn',palette='cividis')

B2a=sns.countplot(x='SeniorCitizen',data=df,hue='Churn',ax=g[1],palette='rainbow')
```

✓ 0.2s

```
Male      0.504756
Female    0.495244
Name: gender, dtype: float64
0    0.837853
1    0.162147
Name: SeniorCitizen, dtype: float64
```





Inference:

Gender:

From above plot we can understand out of 50% and 49% of male and female respectively how many are tending to churn and not churn

Senior Citzen:

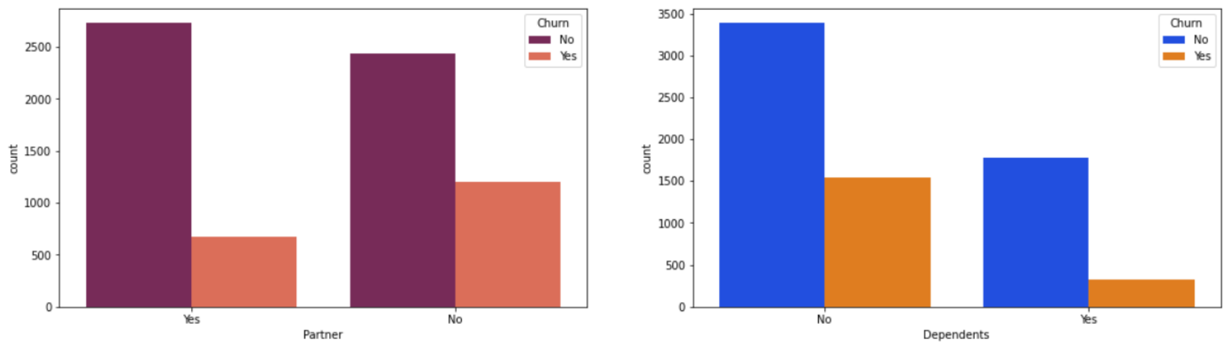
from above plot we can understand out of 83.7% and 16.2% of Non senior Citizen and Seniro Citizen respectively how many are tending to churn and not churn.

Analyzing Partners and Dependents:

Partners and Dependents

```
B3a=df['Partner'].value_counts(normalize=True)
print(B3a)

B4a=df['Dependents'].value_counts(normalize=True)
print(B4a)


fig, g = plt.subplots(1, 2, figsize=(19,5))


B3a=sns.countplot(x='Partner',data=df,hue='Churn',ax=g[0],palette="rocket")

B4a=sns.countplot(x='Dependents',data=df,hue='Churn',ax=g[1],palette='bright')
```
✓ 0.2s

```
No      0.516967
Yes     0.483033
Name: Partner, dtype: float64
No      0.700412
Yes     0.299588
Name: Dependents, dtype: float64
```



Inference:

Partner:

From above plot we can understand out of 51.6% and 48.3% of customers not having partners and customers having partners respectively how many are tending to churn and not churn

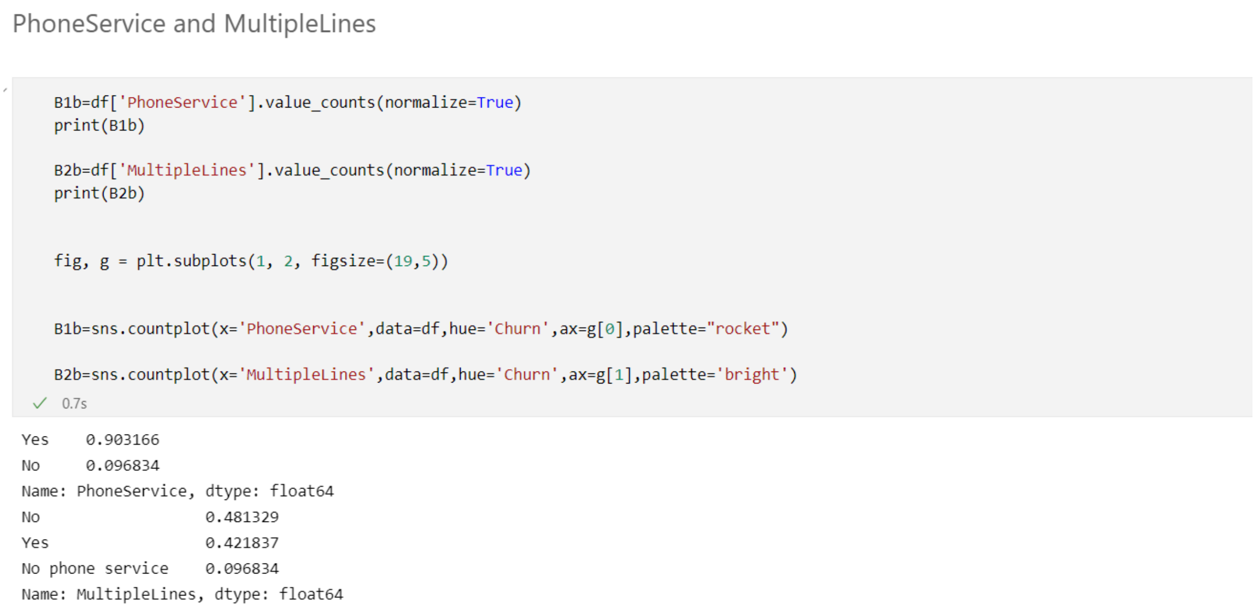
Dependents:

From above plot we can understand out of 70.7% and 30% of customers who are not having any dependents and customers who are having any dependents respectively how many are tending to churn and not churn.

<u>Overall Inference on Demographical section:</u>

Gender and partner are evenly distributed with approximate percentage values. The difference in churn is slightly higher in females, but the small difference can be ignored. There’s a higher proportion of churn in younger customers (SeniorCitizen = No), customers with no partners, and customers with no dependents. The demographic section of data highlights on-senior citizens with no partners and dependents as a particular segment of customers likely to churn.

<u>Analysis on Services that each customer has signed up for:</u>

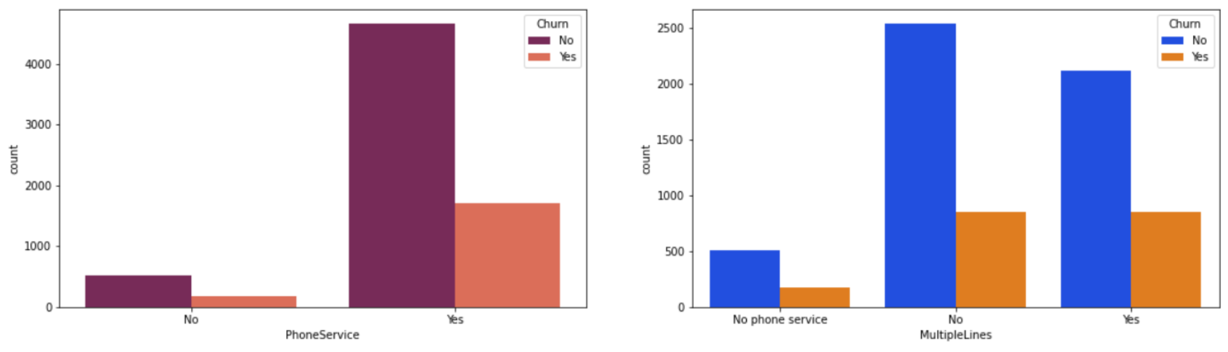


PhoneService and MultipleLines

```
B1b=df['PhoneService'].value_counts(normalize=True)
print(B1b)

B2b=df['MultipleLines'].value_counts(normalize=True)
print(B2b)


fig, g = plt.subplots(1, 2, figsize=(19,5))


B1b=sns.countplot(x='PhoneService',data=df,hue='Churn',ax=g[0],palette="rocket")

B2b=sns.countplot(x='MultipleLines',data=df,hue='Churn',ax=g[1],palette='bright')
```

✓ 0.7s

```
Yes    0.903166
No     0.096834
Name: PhoneService, dtype: float64
No                  0.481329
Yes                 0.421837
No phone service    0.096834
Name: MultipleLines, dtype: float64
```



<u>Inference:</u>

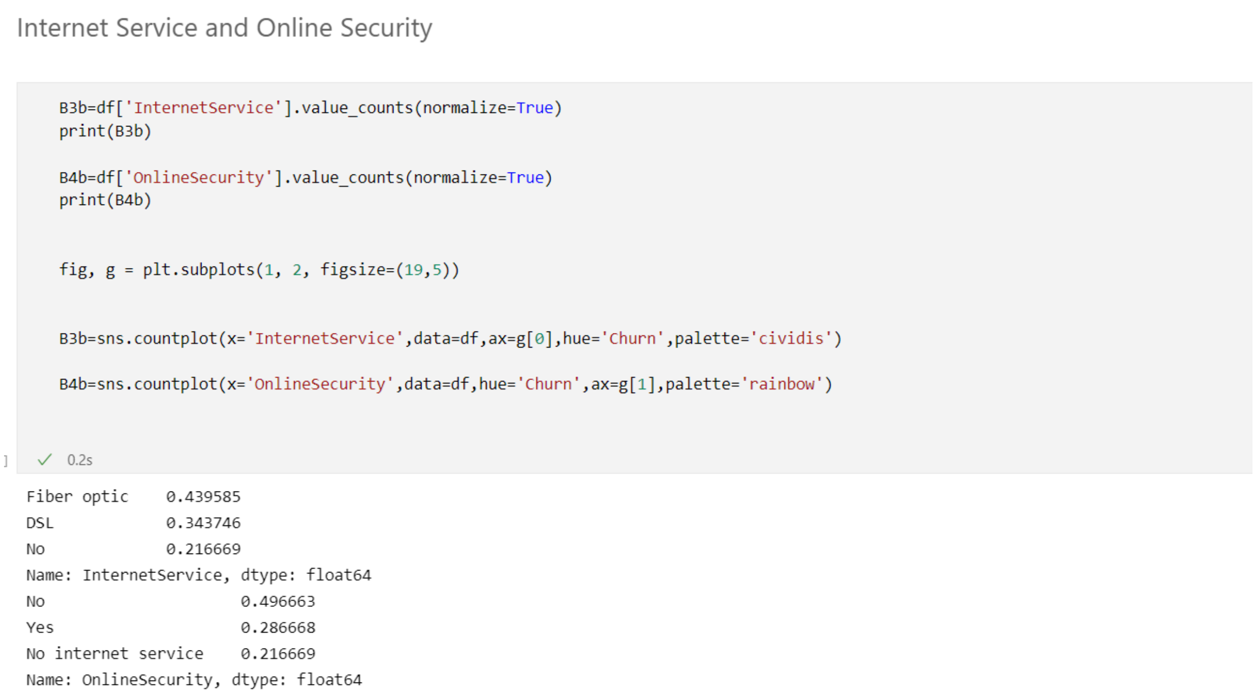
Phone Service:

From above plot we can understand out of 10% and 90% of customers not having phone service and customers having phone Service respectively how many are tending to churn and not churn

Multiple Lines:

From above plot we can understand out of 48%, 42% and 10% of customers who are not having multiple Lines, having multiple lines and customers who are not having phone Service respectively how many are tending to churn and not churn.

Internet services and Online Security

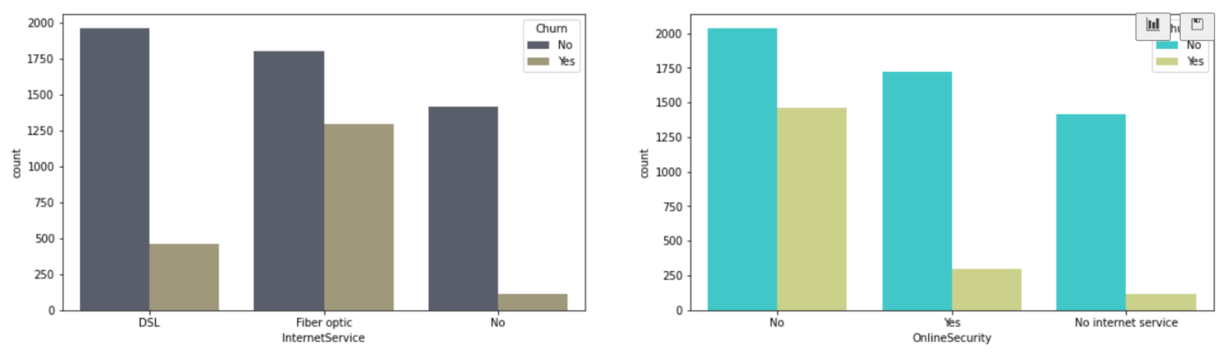


Internet Service and Online Security

```
B3b=df['InternetService'].value_counts(normalize=True)
print(B3b)

B4b=df['OnlineSecurity'].value_counts(normalize=True)
print(B4b)


fig, g = plt.subplots(1, 2, figsize=(19,5))


B3b=sns.countplot(x='InternetService',data=df,ax=g[0],hue='Churn',palette='cividis')

B4b=sns.countplot(x='OnlineSecurity',data=df,hue='Churn',ax=g[1],palette='rainbow')
```

✓ 0.2s

```
Fiber optic    0.439585
DSL            0.343746
No             0.216669
Name: InternetService, dtype: float64
No                     0.496663
Yes                    0.286668
No internet service    0.216669
Name: OnlineSecurity, dtype: float64
```



Inference:

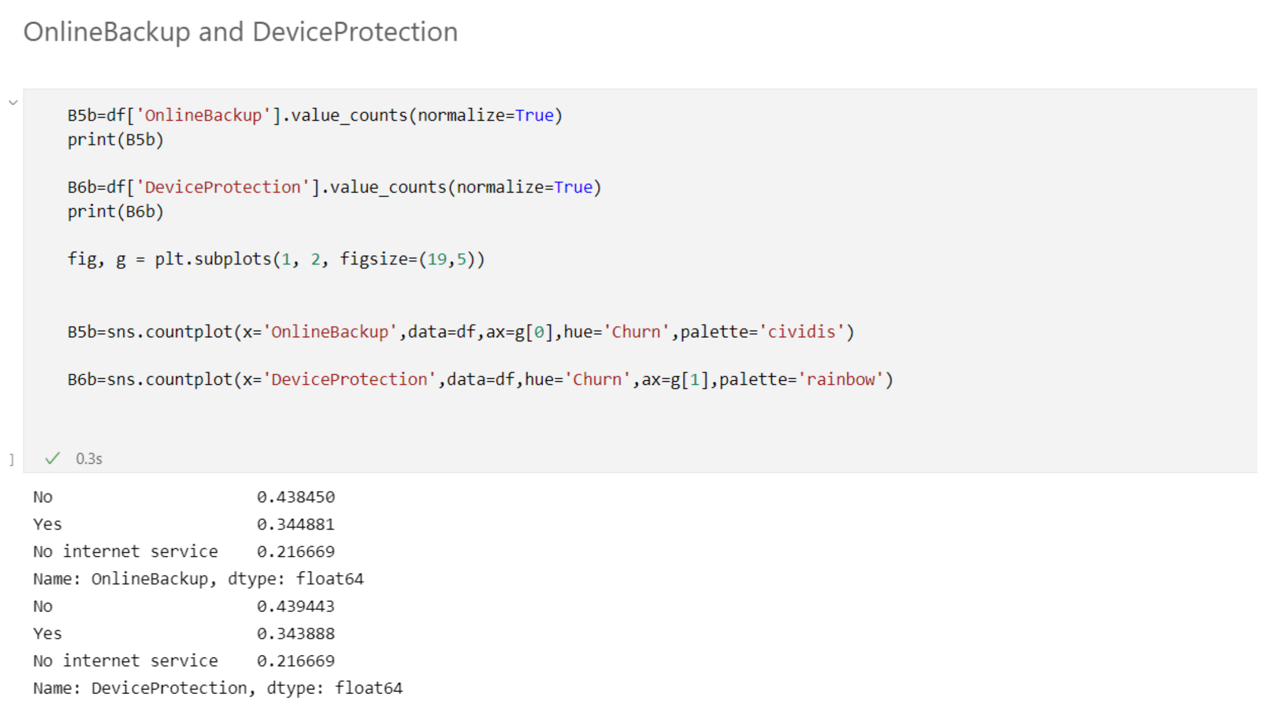
Internet Service:

From above plot we can understand out of 44%,34% and 22% of customers having fibre optic, DSL customers not having Internet Service respectively how many are tending to churn and not churn

Online Security :

From above plot we can understand out of 50%, 29% and 21% of customers who are not having Online Security, having Online Security and customers who are not having Internet Service respectively how many are tending to churn and not churn.

Online Backup and Device Protection:

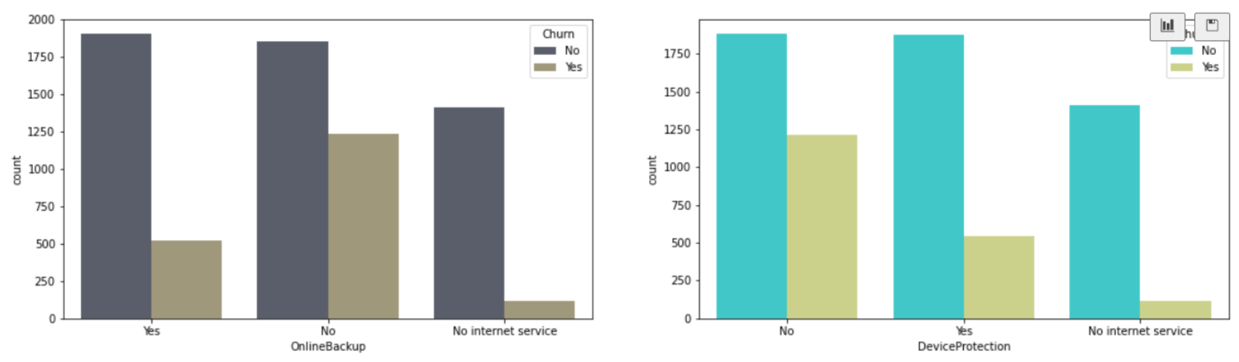


OnlineBackup and DeviceProtection

```
B5b=df['OnlineBackup'].value_counts(normalize=True)
print(B5b)

B6b=df['DeviceProtection'].value_counts(normalize=True)
print(B6b)

fig, g = plt.subplots(1, 2, figsize=(19,5))


B5b=sns.countplot(x='OnlineBackup',data=df,ax=g[0],hue='Churn',palette='cividis')

B6b=sns.countplot(x='DeviceProtection',data=df,hue='Churn',ax=g[1],palette='rainbow')
```

✓ 0.3s

```
No                     0.438450
Yes                    0.344881
No internet service    0.216669
Name: OnlineBackup, dtype: float64
No                     0.439443
Yes                    0.343888
No internet service    0.216669
Name: DeviceProtection, dtype: float64
```



Inference:

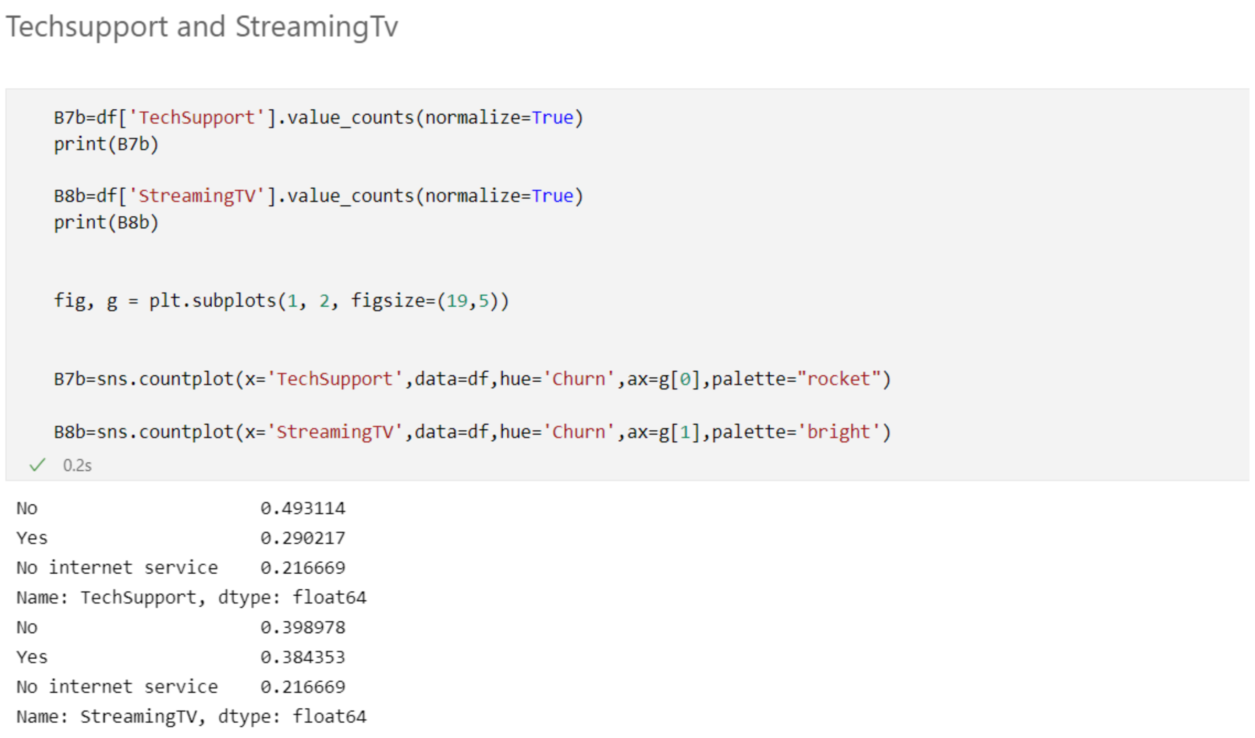
Online Backup:

From above plot we can understand out of 44%,34% and 22% of customers not having Online Backup, customers having Online Backup and customers who are not having Internet Service respectively how many are tending to churn and not churn

Online Security :

From above plot we can understand out of 43.9%,34.5% and 21.7% of customers who are not having Device Protection, having Device Protection and customers who are not having Internet Service respectively how many are tending to churn and not churn.

Tech support and Streaming Tv:

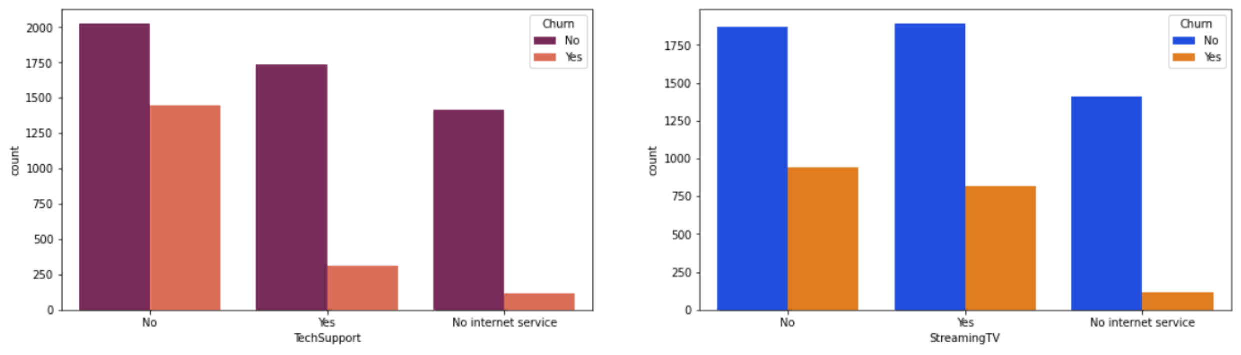


Techsupport and StreamingTv

```
B7b=df['TechSupport'].value_counts(normalize=True)
print(B7b)

B8b=df['StreamingTV'].value_counts(normalize=True)
print(B8b)


fig, g = plt.subplots(1, 2, figsize=(19,5))


B7b=sns.countplot(x='TechSupport',data=df,hue='Churn',ax=g[0],palette="rocket")

B8b=sns.countplot(x='StreamingTV',data=df,hue='Churn',ax=g[1],palette='bright')
```

✓ 0.2s

```
No                     0.493114
Yes                    0.290217
No internet service    0.216669
Name: TechSupport, dtype: float64
No                     0.398978
Yes                    0.384353
No internet service    0.216669
Name: StreamingTV, dtype: float64
```



Inference:

TechSupport:

From above plot we can understand out of 50%,30% and 20% of customers not having TechSupport, customers having TechSupport and customers who are not having Internet Service respectively how many are tending to churn and not churn

StreamingTV :

From above plot we can understand out of 40%,38.5% and 21.5% of customers who are not having StreamingTV, having StreamingTV and customers who are not having Internet Service respectively how many are tending to churn and not churn.

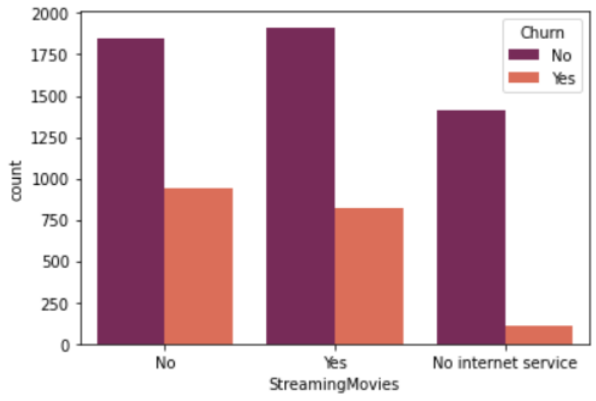
Streaming Movies:

StreamingMovies

```
B9b=df['StreamingMovies'].value_counts(normalize=True)
print(B9b)

B9b=sns.countplot(x='StreamingMovies',data=df,hue='Churn',palette="rocket")
```

✓ 0.1s

```
No                     0.395428
Yes                    0.387903
No internet service    0.216669
Name: StreamingMovies, dtype: float64
```



Inference:

StreamingMovies:

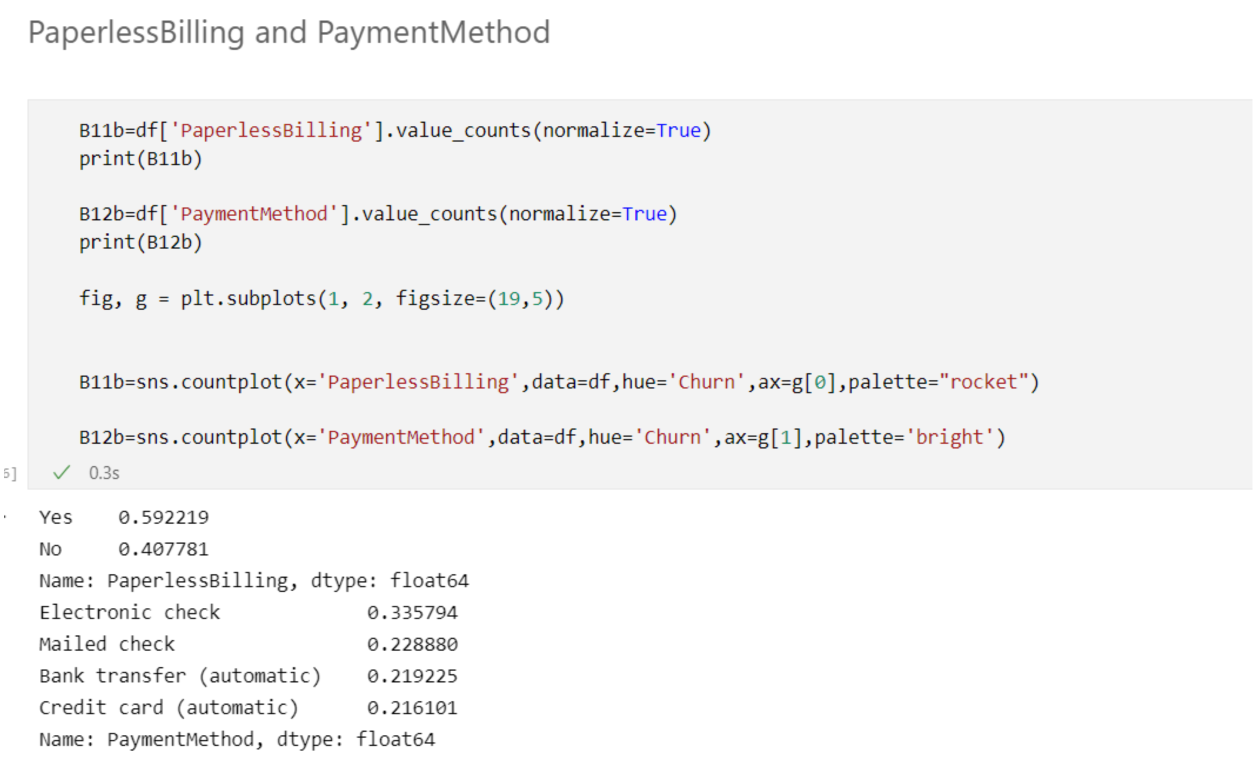
From above plot we can understand out of 39.5%,39% and 21.5% of customers not StreamingMovies, customers StreamingMovies and customers who are not having Internet Service respectively how many are tending to churn and not churn

Overall Inference for Services that each customer has signed up for:

These features show significant variations across their values. If a customer doesn't have phone service, they can't have multiple lines. About 90.3% of the customers have phone services and have a higher rate to churn. Customers who have fibre optic as an internet service are more likely to churn. This can happen due to high prices, competition, customer service, and many other reasons. Fiber optic service is much more expensive than DSL, which may be one of the reasons why customers churn. Customers with OnlineSecurity, OnlineBackup, DeviceProtection, and TechSupport are more unlikely to churn. Streaming service is not predictive for churn as it's evenly distributed to yes and no options.

<u>Analyses on Customer Account information:</u>

Paperless Billing and Payment Method:

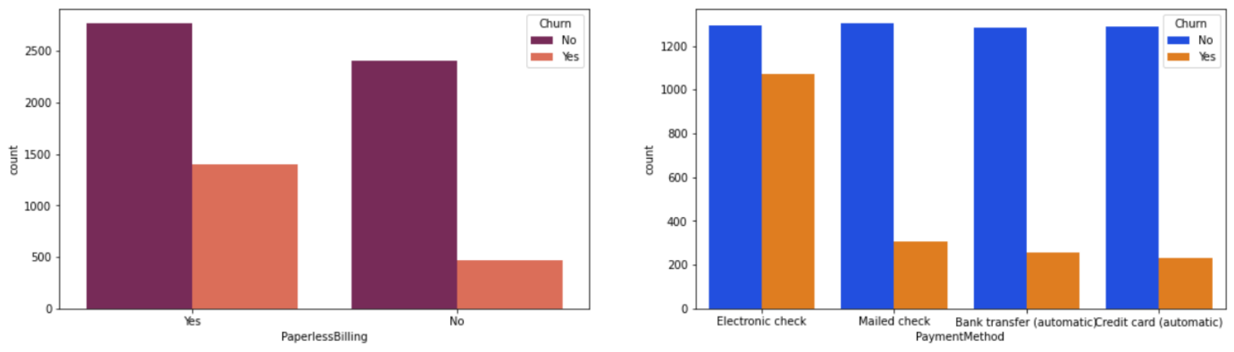


PaperlessBilling and PaymentMethod

```
B11b=df['PaperlessBilling'].value_counts(normalize=True)
print(B11b)

B12b=df['PaymentMethod'].value_counts(normalize=True)
print(B12b)

fig, g = plt.subplots(1, 2, figsize=(19,5))


B11b=sns.countplot(x='PaperlessBilling',data=df,hue='Churn',ax=g[0],palette="rocket")

B12b=sns.countplot(x='PaymentMethod',data=df,hue='Churn',ax=g[1],palette='bright')
```

5] ✓ 0.3s

```
Yes    0.592219
No     0.407781
Name: PaperlessBilling, dtype: float64
Electronic check             0.335794
Mailed check                 0.228880
Bank transfer (automatic)    0.219225
Credit card (automatic)      0.216101
Name: PaymentMethod, dtype: float64
```



Inference:

PaperlessBilling:

From above plot we can understand out of 60% and 40% of customers who do paperless billing and customers who dont do PaperlessBilling respectively how many are tending to churn and not churn

Contract :

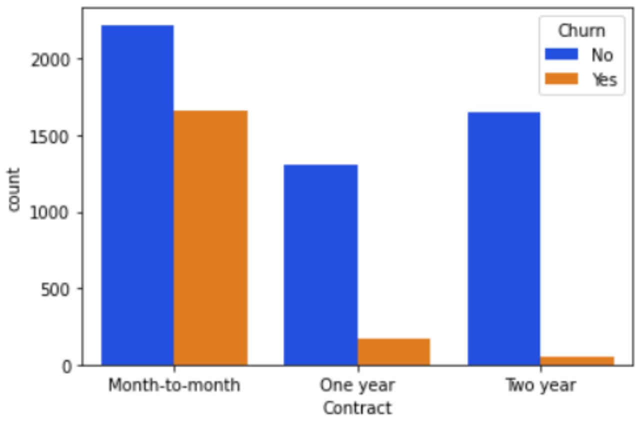
From above plot we can understand out of 33%,23%,22% and 21% of customers who use electronic check, Mailed check, Bank transfer and Credit card as their Payment Method respectively how many are tending to churn and not churn.

Contract:

```
B1c=df['Contract'].value_counts(normalize=True)
print(B1c)

B1c=sns.countplot(x='Contract',data=df,hue='Churn',palette='bright')
```

✓ 0.1s

```
Month-to-month    0.550192
Two year          0.240664
One year          0.209144
Name: Contract, dtype: float64
```



Inference:
Contract :
From above plot we can understand out of 55%,24% and 21% of customers who are not having Contract, having Contract and customers who are not having Internet Service respectively how many are tending to churn and not churn.

Overall inference on Customer Account information:
The shorter the contract, the higher the churn rate. Those with more extended plans face additional barriers when canceling early. This clearly explains the motivation for companies to have long-term relationships with their customers. Churn Rate is higher for the customers who opted for paperless billing. About 59.2% of customers use paperless billing. Customers who pay with electronic checks are more likely to churn, and this kind of payment is more common than other payment types.

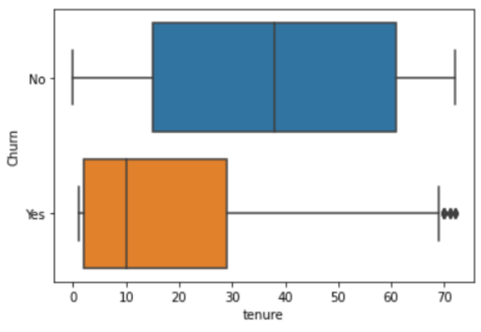
Analysis on Numerical Variable:

Tenure:

tenure

```
sns.boxplot(x='tenure',y='Churn',data=df)
✓ 0.1s
```

```
<AxesSubplot:xlabel='tenure', ylabel='Churn'>
```



Inference:

The tenure boxplot is inferred and shows that most customers have been with the telecom company for just the first few months (0-9 months). The highest rate of churn is also in the first few months (0-9months). most of customers who end up leaving the Telco company do so within their first 30 months.

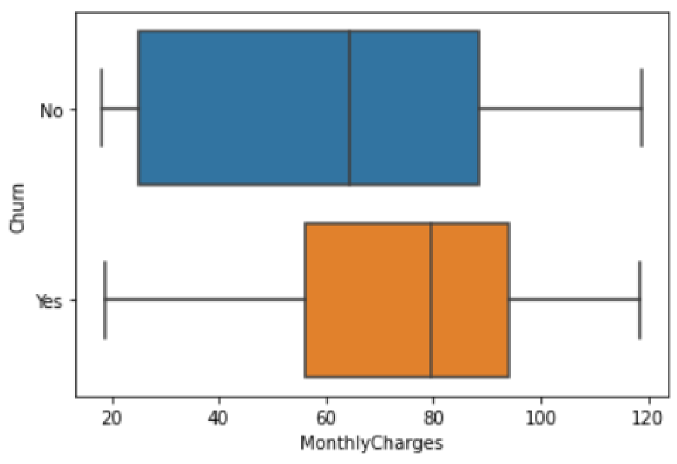
Monthly Charges:

Monthly Charges

+ Code + Markdown

```
sns.boxplot(x='MonthlyCharges',y='Churn',data=df)
✓ 0.1s
```

```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='Churn'>
```



Inference:

The monthly charge plot shows that clients with higher monthly charges have a higher churn rate. This suggests that discounts and promotions can be an enticing reason for customers to stay.

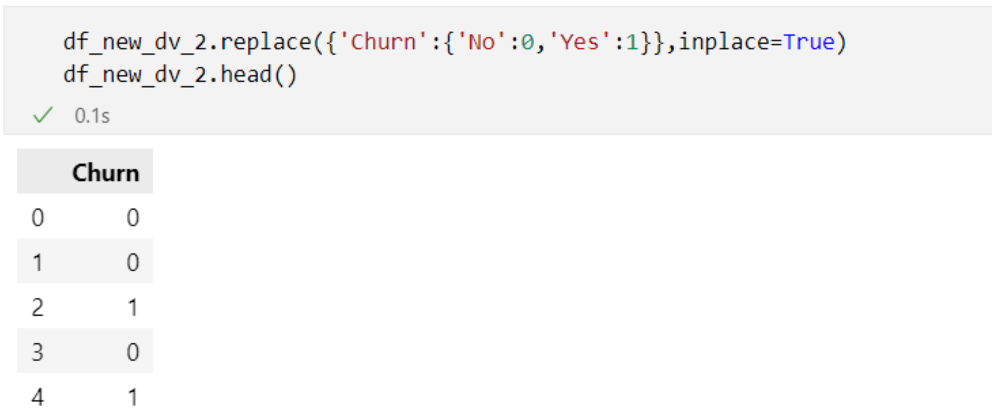
2.3 Feature Engineering:

Classified Weight Approach Encoding:

Note:

- ✓ df_new_dv_2 = refers the encoded part of DV
- ✓ df_new_IDV_1= encoded part of categories
- ✓ df_new_IDV_2 = remaining columns from numerical

<u>Encoding the Dependent variable:</u>

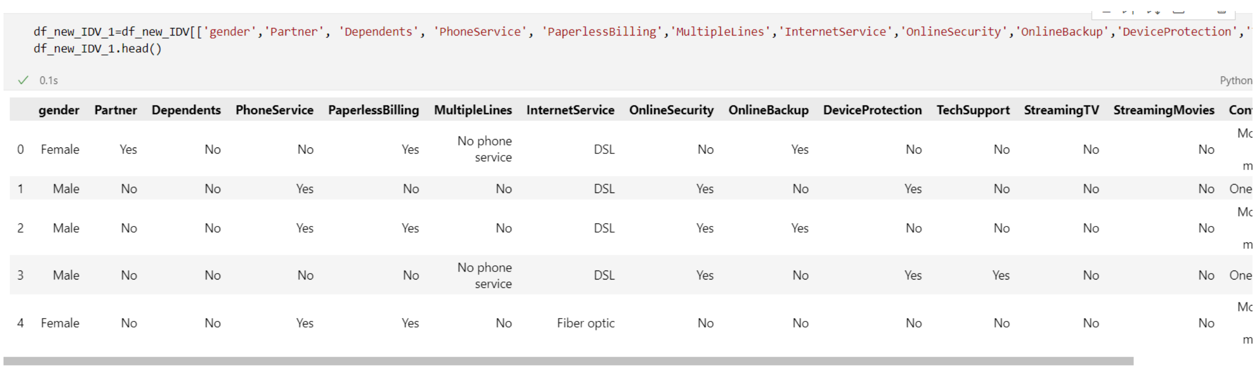


```
df_new_dv_2.replace({'Churn':{'No':0,'Yes':1}},inplace=True)
df_new_dv_2.head()
```

✓ 0.1s

	Churn
0	0
1	0
2	1
3	0
4	1

<u>Encoding the Independent Categorical Variable:</u>

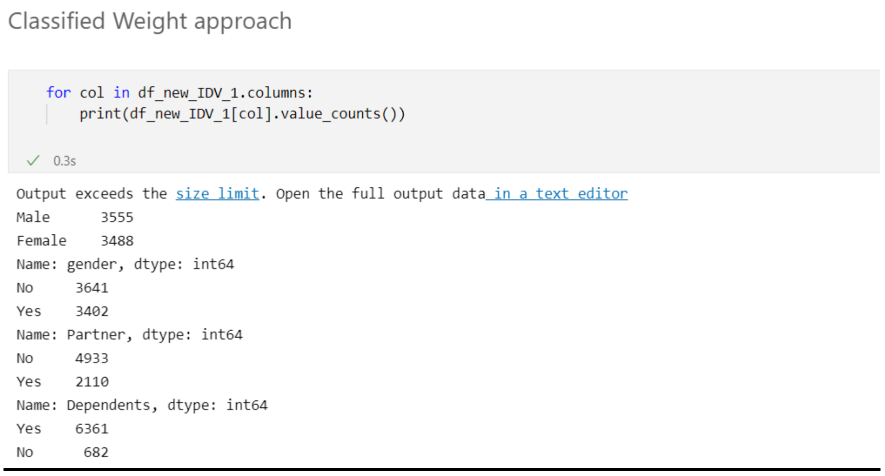


```
df_new_IDV_1=df_new_IDV[['gender','Partner', 'Dependents', 'PhoneService', 'PaperlessBilling','MultipleLines','InternetService','OnlineSecurity','OnlineBackup','DeviceProtection','
df_new_IDV_1.head()
```

✓ 0.1s Python

	gender	Partner	Dependents	PhoneService	PaperlessBilling	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Con
0	Female	Yes	No	No	Yes	No phone service	DSL	No	Yes	No	No	No	No	Mc m
1	Male	No	No	Yes	No	No	DSL	Yes	No	Yes	No	No	No	One
2	Male	No	No	Yes	Yes	No	DSL	Yes	Yes	No	No	No	No	Mc m
3	Male	No	No	No	No	No phone service	DSL	Yes	No	Yes	Yes	No	No	One
4	Female	No	No	Yes	Yes	No	Fiber optic	No	No	No	No	No	No	Mc m

<u>Using classified weight approach to encode the IDV_Numerical variables:</u>



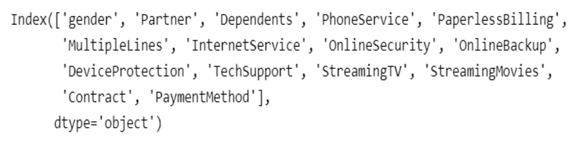
Classified Weight approach

```
for col in df_new_IDV_1.columns:
    print(df_new_IDV_1[col].value_counts())
```

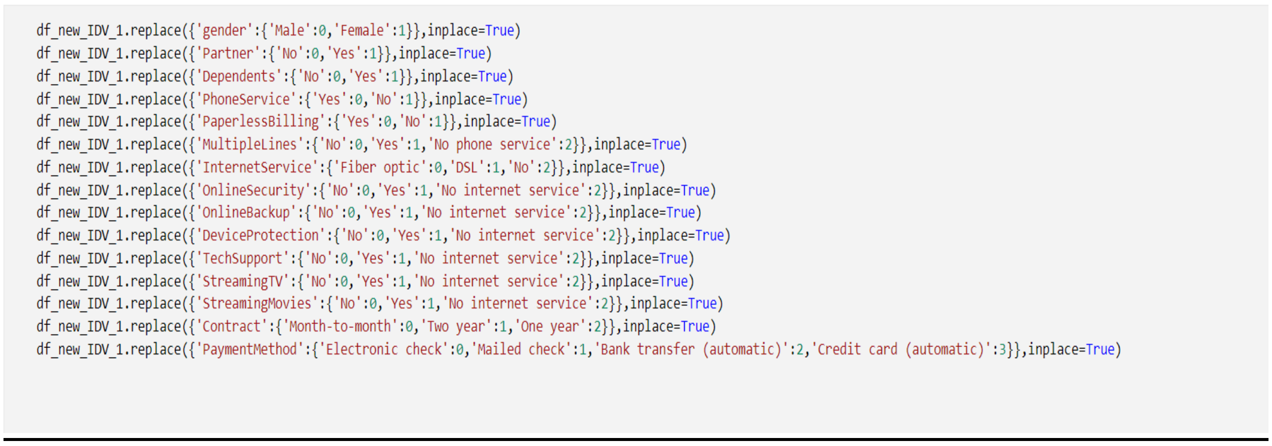
✓ 0.3s

```
Output exceeds the size limit. Open the full output data in a text editor
Male      3555
Female    3488
Name: gender, dtype: int64
No     3641
Yes    3402
Name: Partner, dtype: int64
No     4933
Yes    2110
Name: Dependents, dtype: int64
Yes    6361
No      682
```

Encoding the variables:

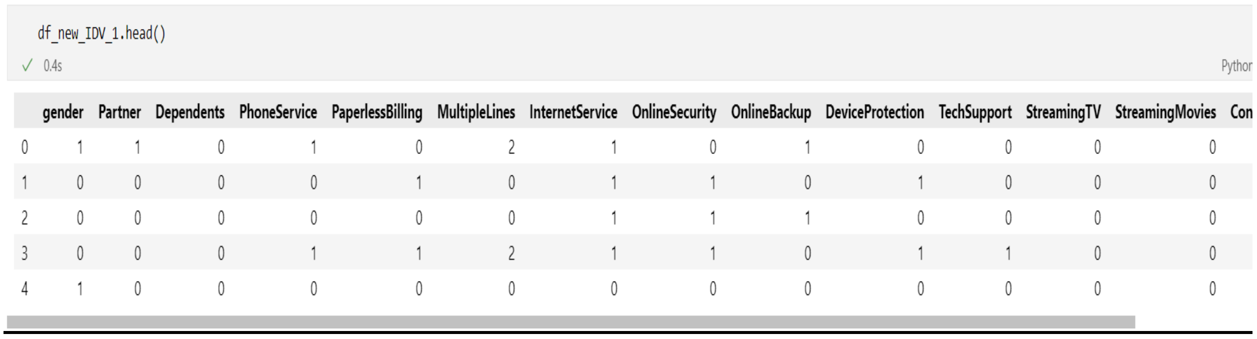


```
Index(['gender', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaymentMethod'],
      dtype='object')
```



```
df_new_IDV_1.replace({'gender':{'Male':0,'Female':1}},inplace=True)
df_new_IDV_1.replace({'Partner':{'No':0,'Yes':1}},inplace=True)
df_new_IDV_1.replace({'Dependents':{'No':0,'Yes':1}},inplace=True)
df_new_IDV_1.replace({'PhoneService':{'Yes':0,'No':1}},inplace=True)
df_new_IDV_1.replace({'PaperlessBilling':{'Yes':0,'No':1}},inplace=True)
df_new_IDV_1.replace({'MultipleLines':{'No':0,'Yes':1,'No phone service':2}},inplace=True)
df_new_IDV_1.replace({'InternetService':{'Fiber optic':0,'DSL':1,'No':2}},inplace=True)
df_new_IDV_1.replace({'OnlineSecurity':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'OnlineBackup':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'DeviceProtection':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'TechSupport':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'StreamingTV':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'StreamingMovies':{'No':0,'Yes':1,'No internet service':2}},inplace=True)
df_new_IDV_1.replace({'Contract':{'Month-to-month':0,'Two year':1,'One year':2}},inplace=True)
df_new_IDV_1.replace({'PaymentMethod':{'Electronic check':0,'Mailed check':1,'Bank transfer (automatic)':2,'Credit card (automatic)':3}},inplace=True)
```

Encoded Independent Variables:

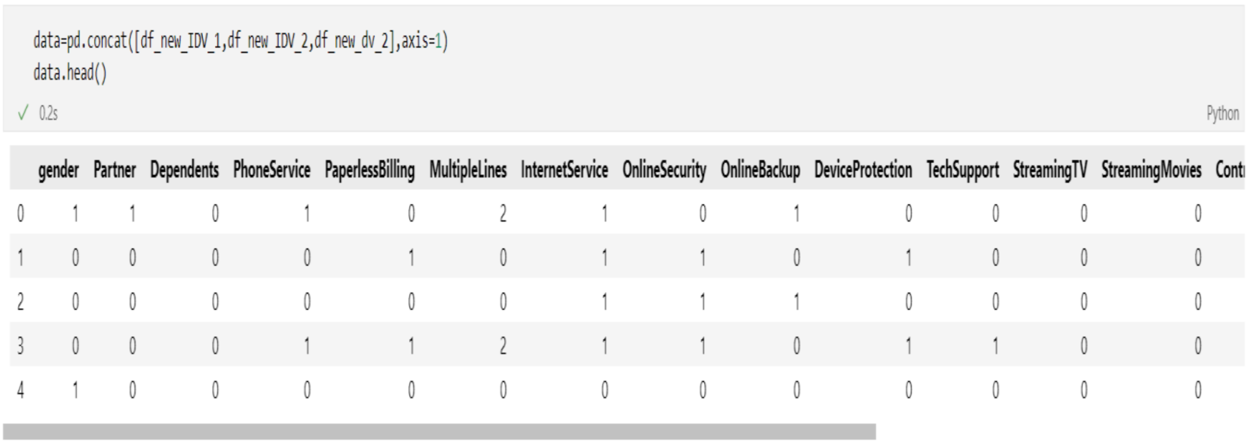


```
df_new_IDV_1.head()
```

✓ 0.4s Pythor

	gender	Partner	Dependents	PhoneService	PaperlessBilling	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Con
0	1	1	0	1	0	2	1	0	1	0	0	0	0	
1	0	0	0	0	1	0	1	1	0	1	0	0	0	
2	0	0	0	0	0	0	1	1	1	0	0	0	0	
3	0	0	0	1	1	2	1	1	0	1	1	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	0	

Dataset after encoding : Data



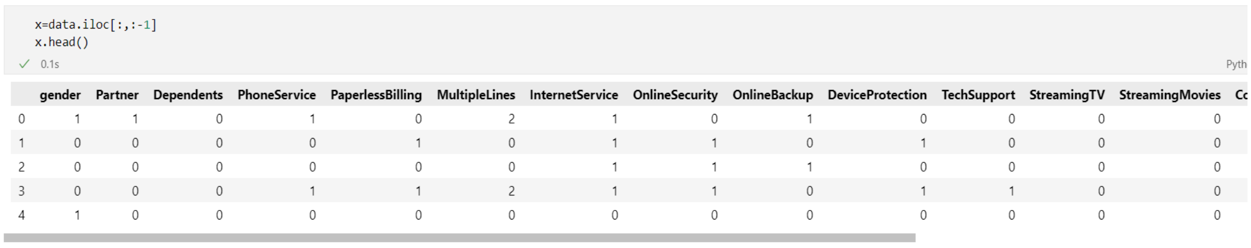
```
data=pd.concat([df_new_IDV_1,df_new_IDV_2,df_new_dv_2],axis=1)
data.head()
```

✓ 0.2s Python

	gender	Partner	Dependents	PhoneService	PaperlessBilling	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Cont
0	1	1	0	1	0	2	1	0	1	0	0	0	0	
1	0	0	0	0	1	0	1	1	0	1	0	0	0	
2	0	0	0	0	0	0	1	1	1	0	0	0	0	
3	0	0	0	1	1	2	1	1	0	1	1	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	0	

Our dataset = data is finally encoded and is ready to be scaled.

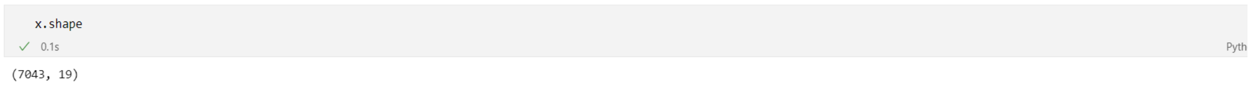
Split data into Independent and Dependent.



```
x=data.iloc[:,:-1]
x.head()
```

✓ 0.1s Pyth

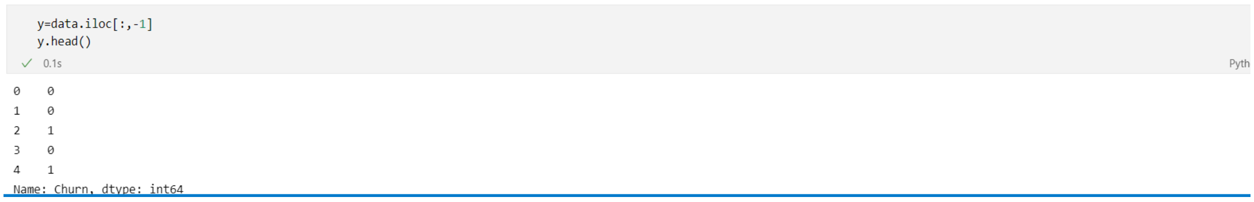
	gender	Partner	Dependents	PhoneService	PaperlessBilling	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Cc
0	1	1	0	1	0	2	1	0	1	0	0	0	0	
1	0	0	0	0	1	0	1	1	0	1	0	0	0	
2	0	0	0	0	0	0	1	1	1	0	0	0	0	
3	0	0	0	1	1	2	1	1	0	1	1	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	0	



```
x.shape
```

✓ 0.1s Pyth

```
(7043, 19)
```



```
y=data.iloc[:,-1]
y.head()
```

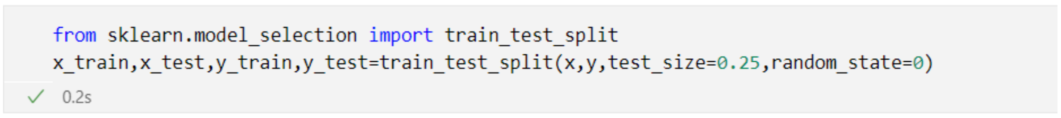
✓ 0.1s Pyth

```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

Feature Scaling :

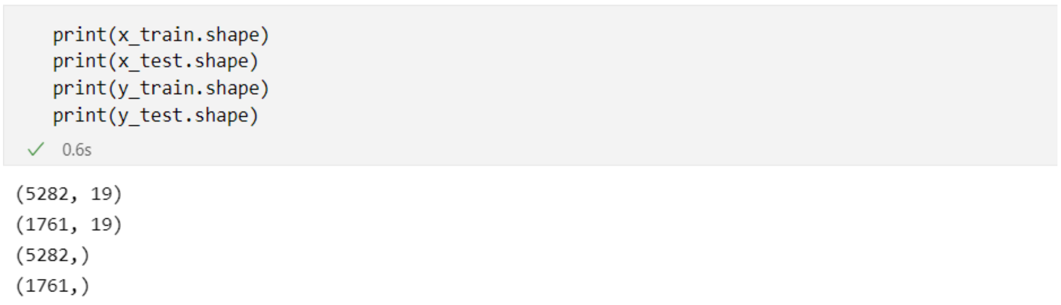
Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. Before doing Feature Scaling we have to split the dataset into train and test.

Train and Test split:



```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

✓ 0.2s

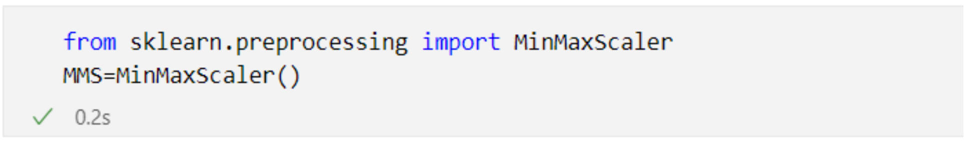


```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

✓ 0.6s

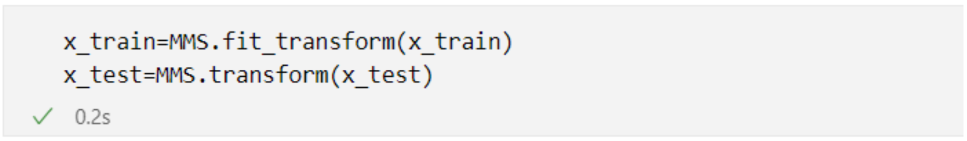
```
(5282, 19)
(1761, 19)
(5282,)
(1761,)
```

Using MinMax Scaler to scale the features.



```
from sklearn.preprocessing import MinMaxScaler
MMS=MinMaxScaler()
```

✓ 0.2s

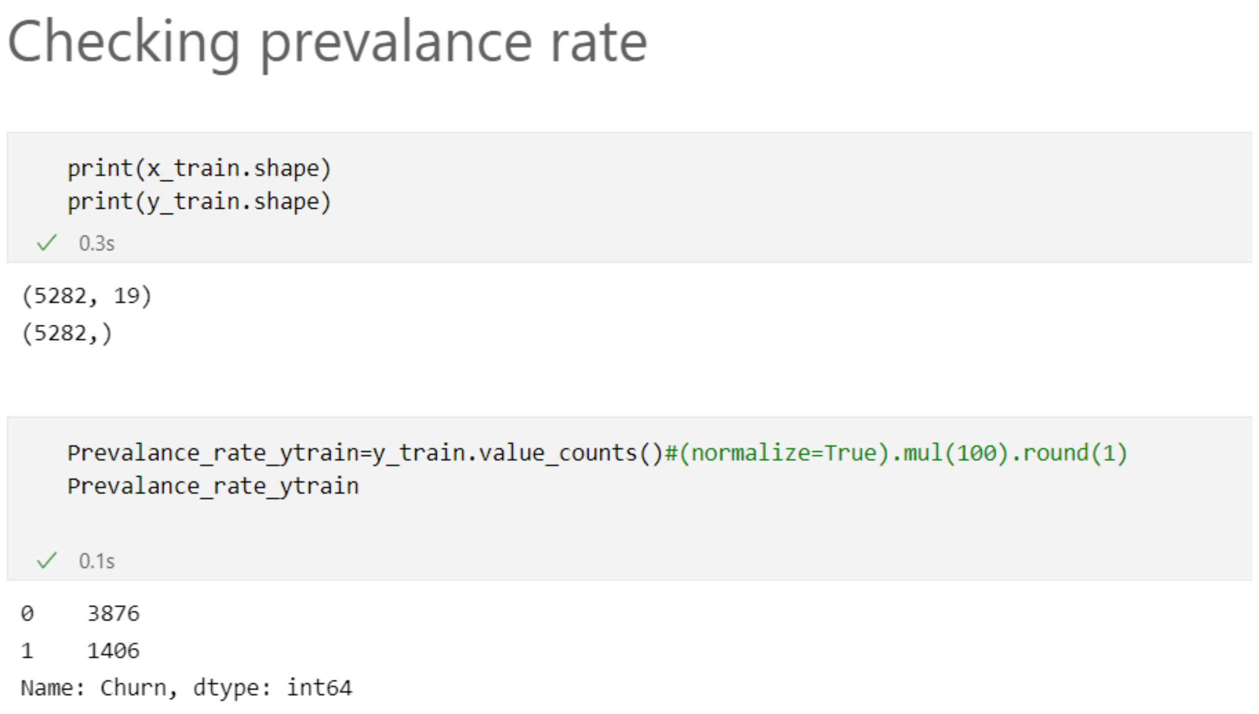


```
x_train=MMS.fit_transform(x_train)
x_test=MMS.transform(x_test)
```

✓ 0.2s

Prevalence Rate check:

To understand if the dataset is balanced or not, we need to find prevalence rate such that it shows on what percentage the classes has been segregated.



Checking prevalance rate

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

✓ 0.3s

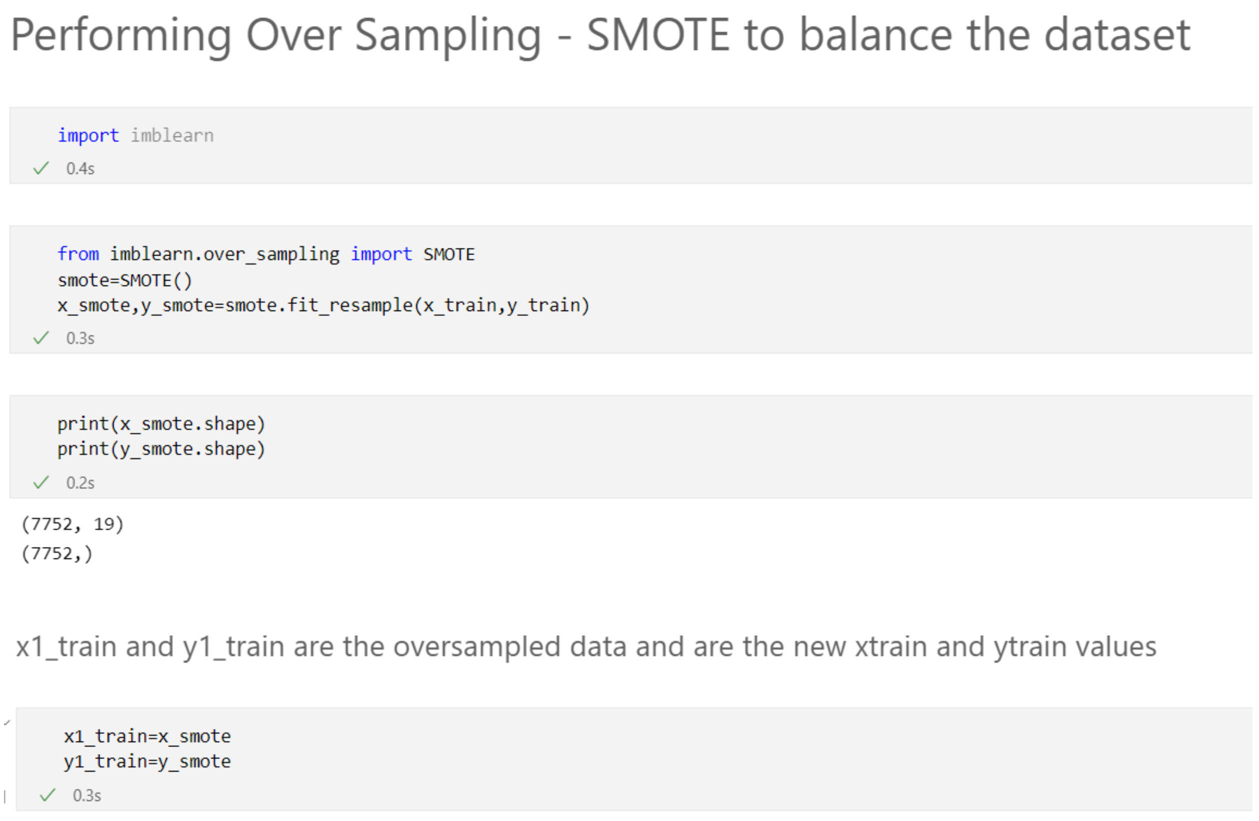
```
(5282, 19)
(5282,)
```

```
Prevalance_rate_ytrain=y_train.value_counts()#(normalize=True).mul(100).round(1)
Prevalance_rate_ytrain
```

✓ 0.1s

```
0    3876
1    1406
Name: Churn, dtype: int64
```

The prevalence rate shows that the dataset is imbalanced and we need to apply SMOTE technique to make the dataset balanced.



Performing Over Sampling - SMOTE to balance the dataset

```
import imblearn
```

✓ 0.4s

```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
x_smote,y_smote=smote.fit_resample(x_train,y_train)
```

✓ 0.3s

```
print(x_smote.shape)
print(y_smote.shape)
```

✓ 0.2s

```
(7752, 19)
(7752,)
```

x1_train and y1_train are the oversampled data and are the new xtrain and ytrain values

```
x1_train=x_smote
y1_train=y_smote
```

✓ 0.3s

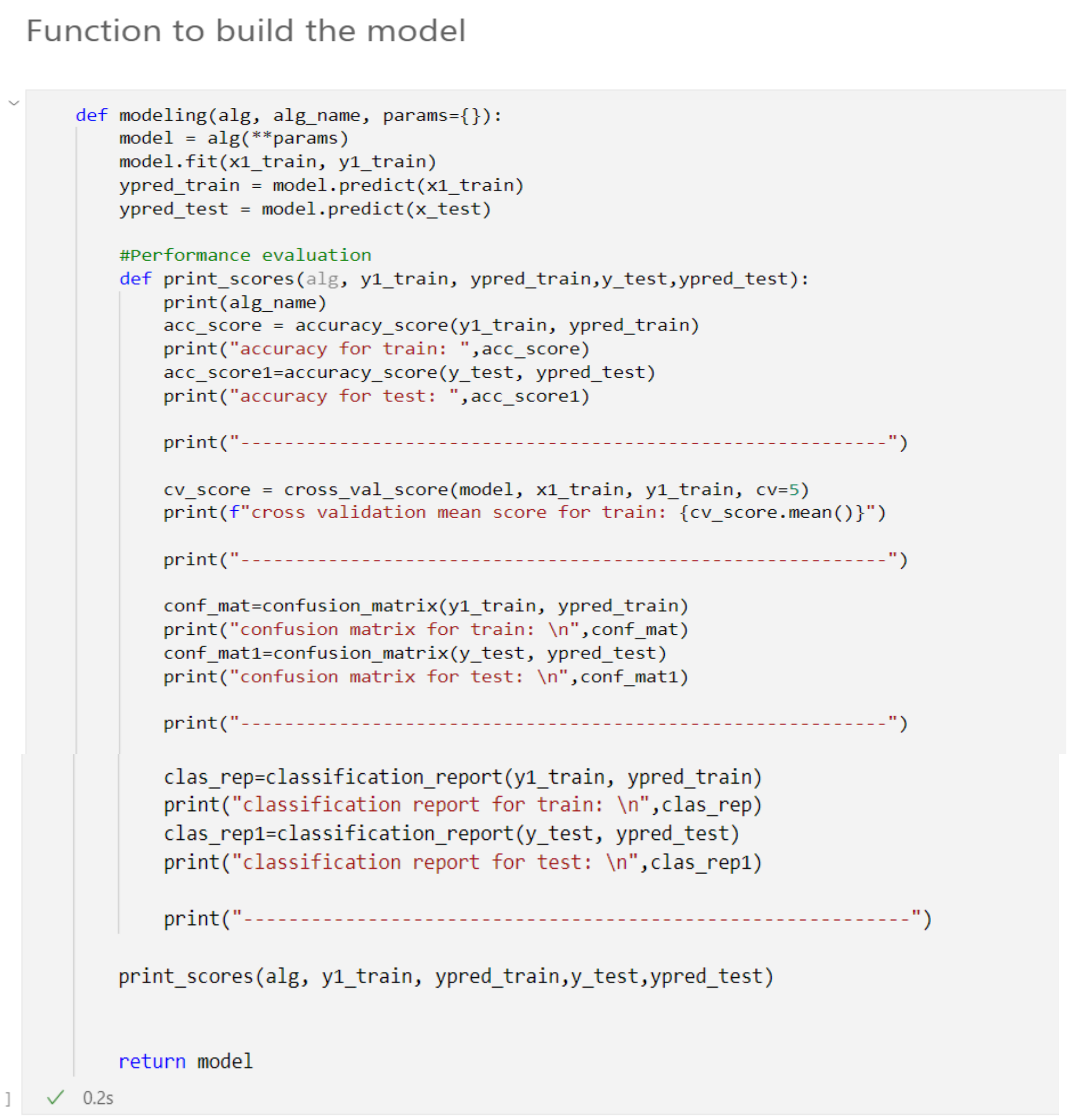
Chapter 3

Model Building

Data Preprocessing is finished and the data is ready to be fed into the model such that prediction can be made by analyzing the underlying pattern what it learnt from the data. Here, I have used 5 classification algorithms for the study to be used for prediction and will choose the best model out of it.

Approaches used :

- ✓ **Logistic Regression**
- ✓ **Decision Tree**
- ✓ **Random Forest**
- ✓ **Naïve Bayes**
- ✓ **Ada Boosting**



Function to build the model

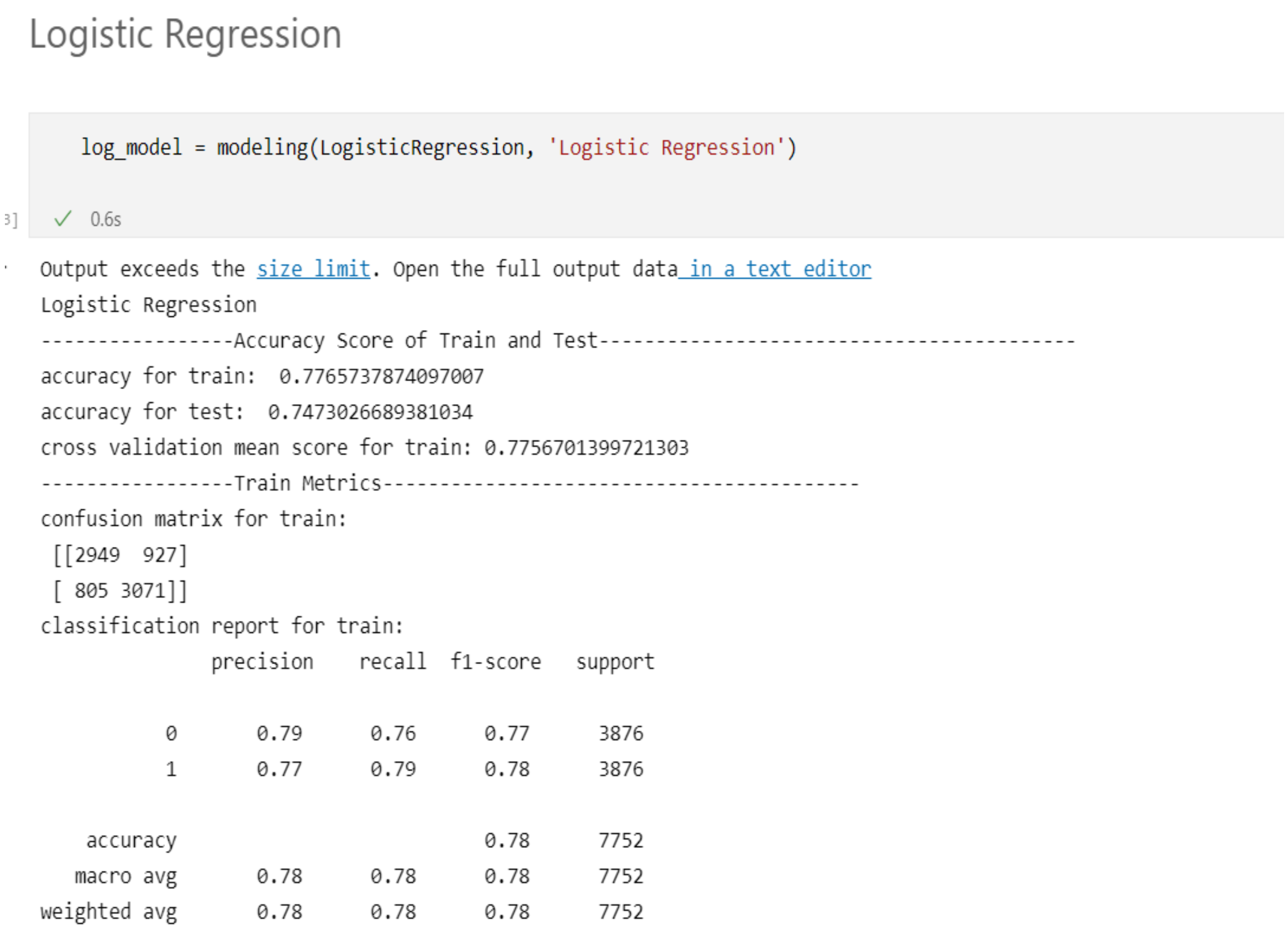
```python
def modeling(alg, alg_name, params={}):
    model = alg(**params)
    model.fit(x1_train, y1_train)
    ypred_train = model.predict(x1_train)
    ypred_test = model.predict(x_test)

    #Performance evaluation
    def print_scores(alg, y1_train, ypred_train,y_test,ypred_test):
        print(alg_name)
        acc_score = accuracy_score(y1_train, ypred_train)
        print("accuracy for train: ",acc_score)
        acc_score1=accuracy_score(y_test, ypred_test)
        print("accuracy for test: ",acc_score1)

        print("-----------------------------------------------------------")

        cv_score = cross_val_score(model, x1_train, y1_train, cv=5)
        print(f"cross validation mean score for train: {cv_score.mean()}")

        print("-----------------------------------------------------------")

        conf_mat=confusion_matrix(y1_train, ypred_train)
        print("confusion matrix for train: \n",conf_mat)
        conf_mat1=confusion_matrix(y_test, ypred_test)
        print("confusion matrix for test: \n",conf_mat1)

        print("-----------------------------------------------------------")

        clas_rep=classification_report(y1_train, ypred_train)
        print("classification report for train: \n",clas_rep)
        clas_rep1=classification_report(y_test, ypred_test)
        print("classification report for test: \n",clas_rep1)

        print("-----------------------------------------------------------")

    print_scores(alg, y1_train, ypred_train,y_test,ypred_test)


    return model
```

✓ 0.2s

I have used a function to fit all the model such that all the functions will be called by its model name to find out the accuracy, confusion matrix and classification report.

Logistic Regression:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

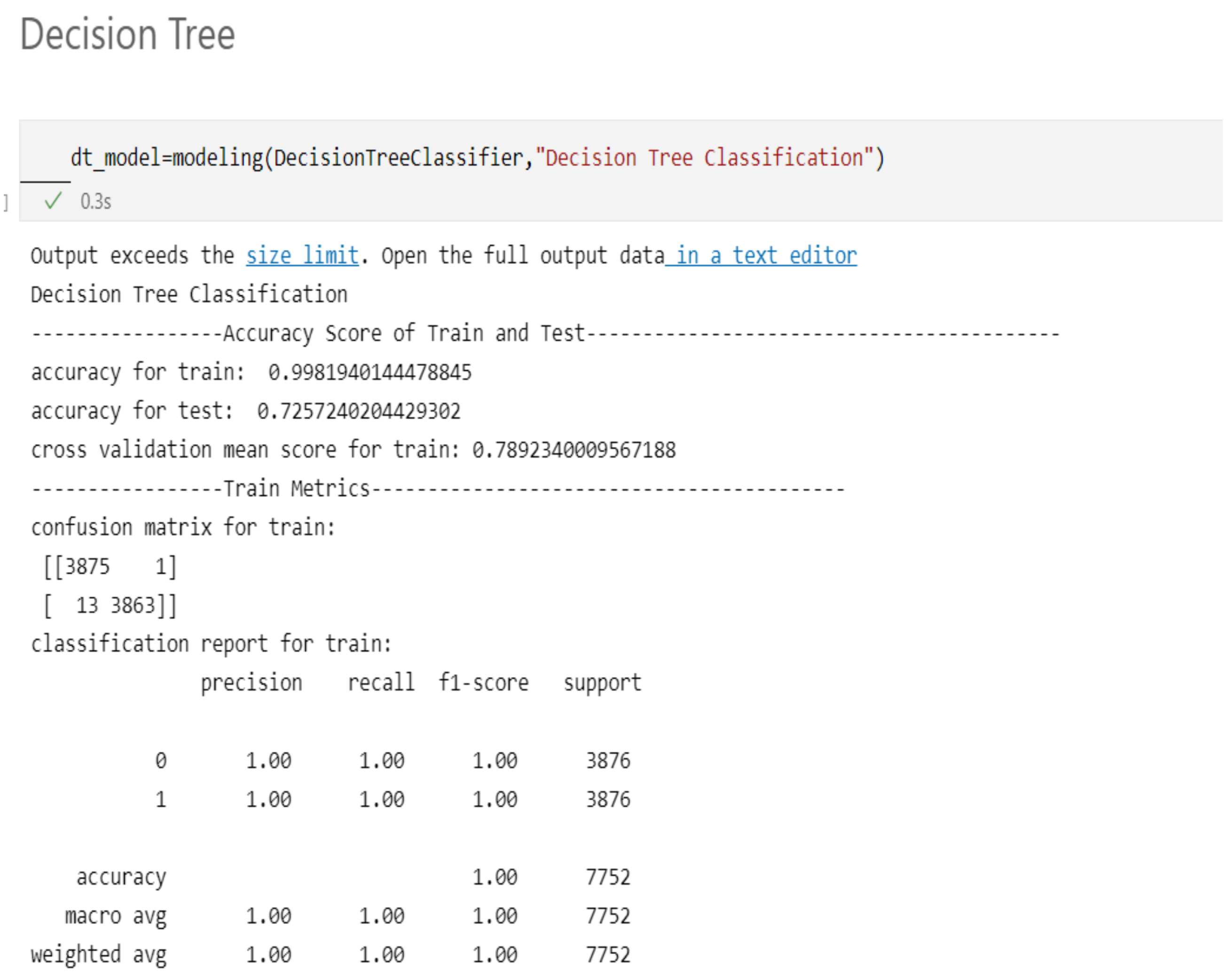

Logistic Regression

```
log_model = modeling(LogisticRegression, 'Logistic Regression')
```

```
Output exceeds the size limit. Open the full output data in a text editor
Logistic Regression
-----------------Accuracy Score of Train and Test-----------------------------------------
accuracy for train:  0.7765737874097007
accuracy for test:  0.7473026689381034
cross validation mean score for train: 0.7756701399721303
-----------------Train Metrics-----------------------------------------
confusion matrix for train:
 [[2949  927]
 [ 805 3071]]
classification report for train:
               precision    recall  f1-score   support

           0       0.79      0.76      0.77      3876
           1       0.77      0.79      0.78      3876

    accuracy                           0.78      7752
   macro avg       0.78      0.78      0.78      7752
weighted avg       0.78      0.78      0.78      7752
```

Decision Tree:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.



Decision Tree

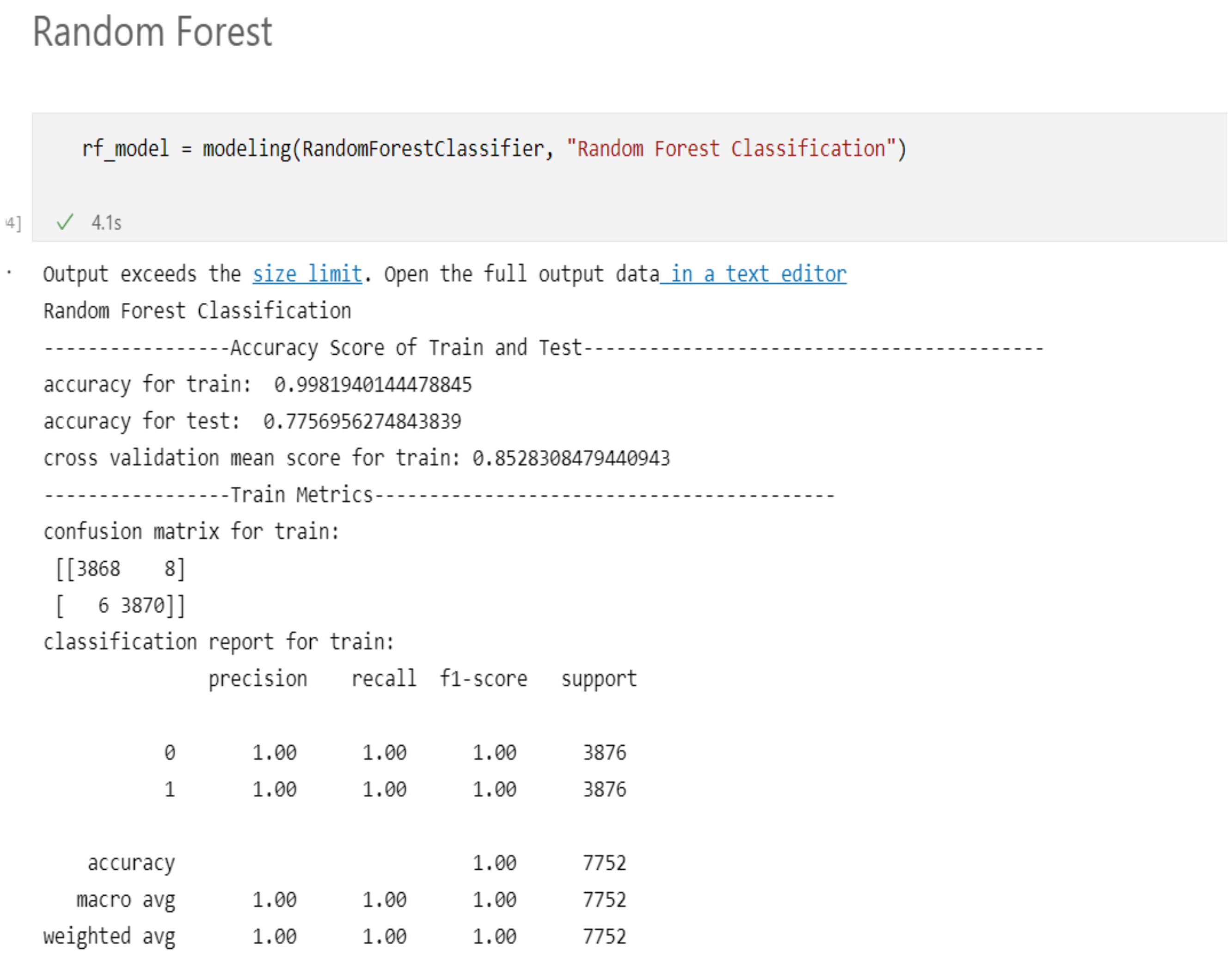
```
dt_model=modeling(DecisionTreeClassifier,"Decision Tree Classification")
```

✓ 0.3s

```
Output exceeds the size limit. Open the full output data in a text editor
Decision Tree Classification
-----------------Accuracy Score of Train and Test-----------------------------------------
accuracy for train:  0.9981940144478845
accuracy for test:  0.7257240204429302
cross validation mean score for train: 0.7892340009567188
-----------------Train Metrics-----------------------------------------
confusion matrix for train:
 [[3875    1]
 [  13 3863]]
classification report for train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3876
           1       1.00      1.00      1.00      3876

    accuracy                           1.00      7752
   macro avg       1.00      1.00      1.00      7752
weighted avg       1.00      1.00      1.00      7752
```

Random Forest:

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.



Random Forest

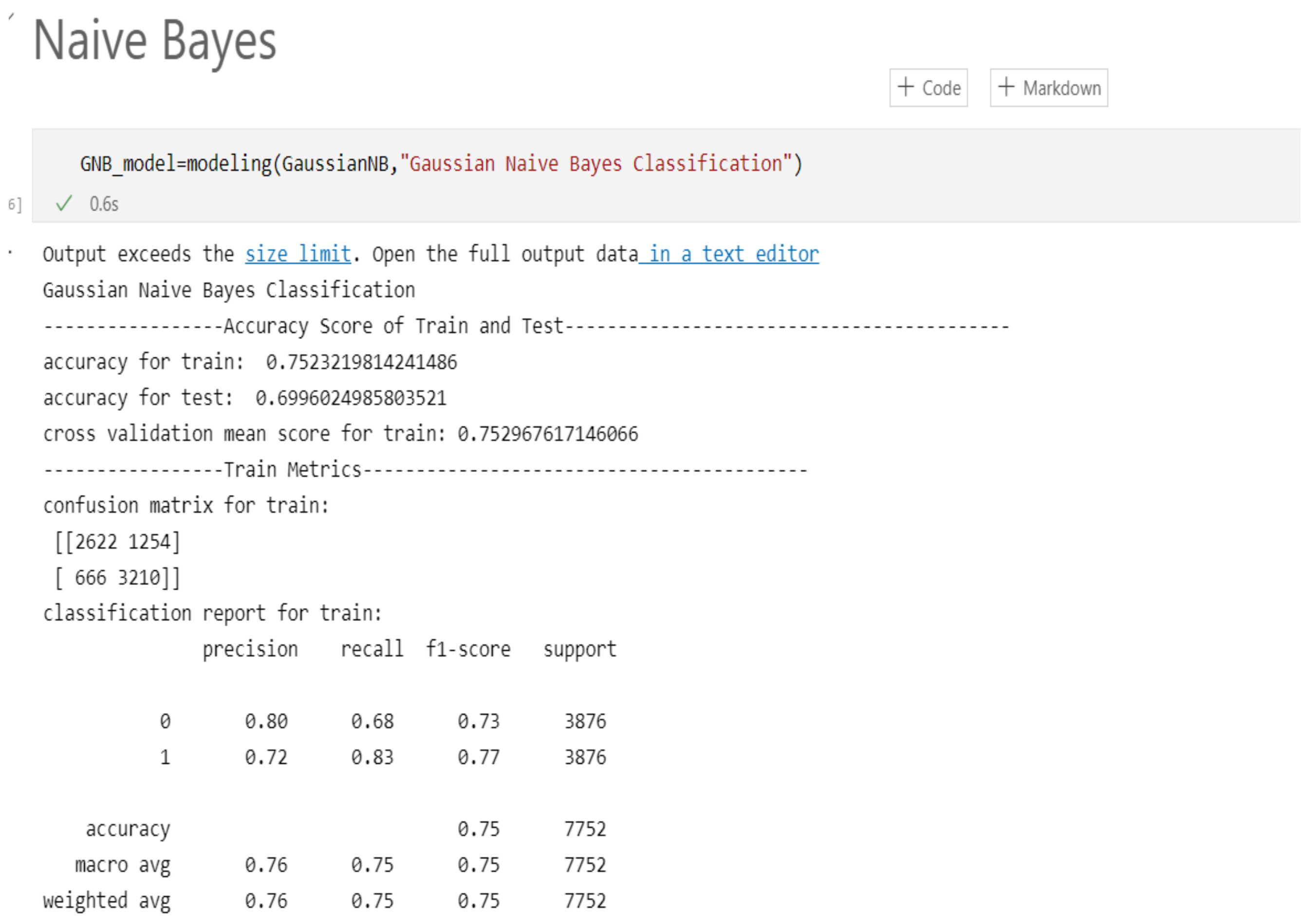
```
rf_model = modeling(RandomForestClassifier, "Random Forest Classification")
```

4] ✓ 4.1s

```
Output exceeds the size limit. Open the full output data in a text editor
Random Forest Classification
-----------------Accuracy Score of Train and Test-----------------------------------------
accuracy for train:  0.9981940144478845
accuracy for test:  0.7756956274843839
cross validation mean score for train: 0.8528308479440943
-----------------Train Metrics-----------------------------------------
confusion matrix for train:
 [[3868    8]
 [   6 3870]]
classification report for train:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      3876
           1       1.00      1.00      1.00      3876

    accuracy                           1.00      7752
   macro avg       1.00      1.00      1.00      7752
weighted avg       1.00      1.00      1.00      7752
```

Naive Bayes:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes’ Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

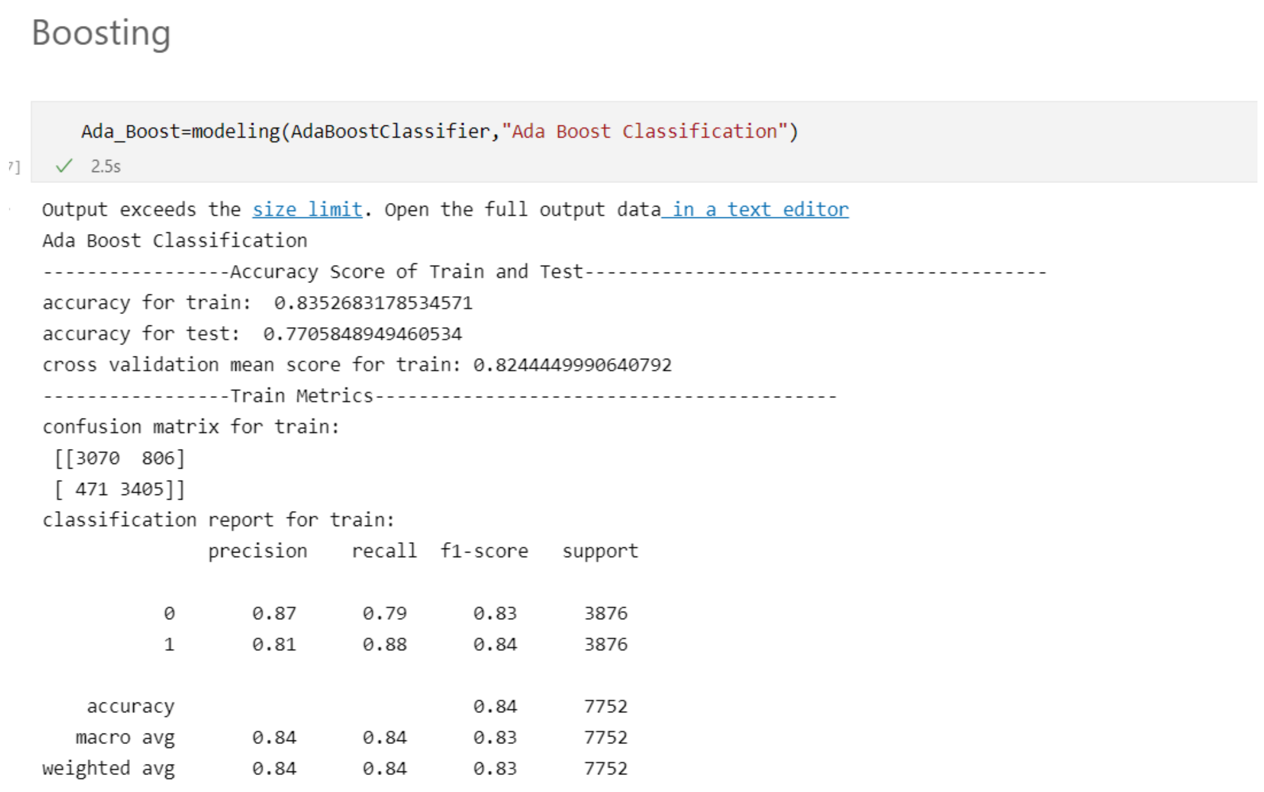


```
GNB_model=modeling(GaussianNB,"Gaussian Naive Bayes Classification")
```

```
Output exceeds the size limit. Open the full output data in a text editor
Gaussian Naive Bayes Classification
-----------------Accuracy Score of Train and Test-----------------------------------------
accuracy for train:  0.7523219814241486
accuracy for test:  0.6996024985803521
cross validation mean score for train: 0.752967617146066
-----------------Train Metrics-----------------------------------------
confusion matrix for train:
 [[2622 1254]
 [ 666 3210]]
classification report for train:
              precision    recall  f1-score   support

           0       0.80      0.68      0.73      3876
           1       0.72      0.83      0.77      3876

    accuracy                           0.75      7752
   macro avg       0.76      0.75      0.75      7752
weighted avg       0.76      0.75      0.75      7752
```

Ada Boosting:

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called Decision Stumps.



Boosting

```
Ada_Boost=modeling(AdaBoostClassifier,"Ada Boost Classification")
```

✓ 2.5s

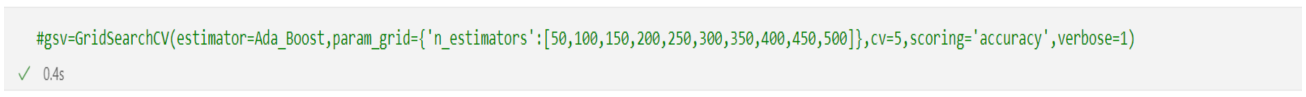
```
Output exceeds the size limit. Open the full output data in a text editor
Ada Boost Classification
-----------------Accuracy Score of Train and Test-----------------------------------------
accuracy for train:  0.8352683178534571
accuracy for test:  0.7705848949460534
cross validation mean score for train: 0.8244449990640792
-----------------Train Metrics-----------------------------------------
confusion matrix for train:
 [[3070  806]
 [ 471 3405]]
classification report for train:
              precision    recall  f1-score   support

           0       0.87      0.79      0.83      3876
           1       0.81      0.88      0.84      3876

    accuracy                           0.84      7752
   macro avg       0.84      0.84      0.83      7752
weighted avg       0.84      0.84      0.83      7752
```

Chapter 4

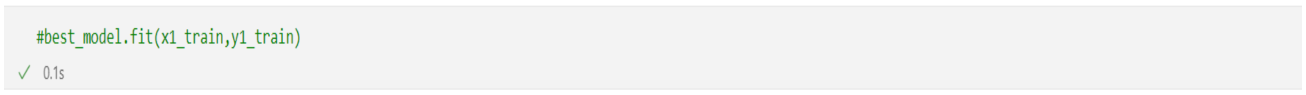
Hyper Parameter tuning

From the above models it is clear that Ada Boost classifier gives the highest Accuracy and hereby, fine tuning the best model in order to check if accuracy can be improved.

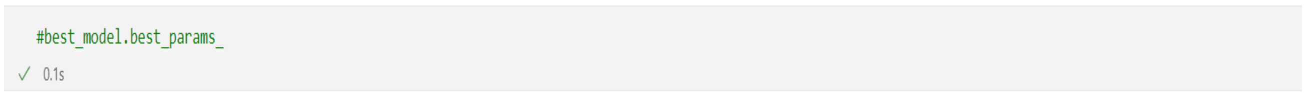
Finding the Best parameter for fine tuning the model:



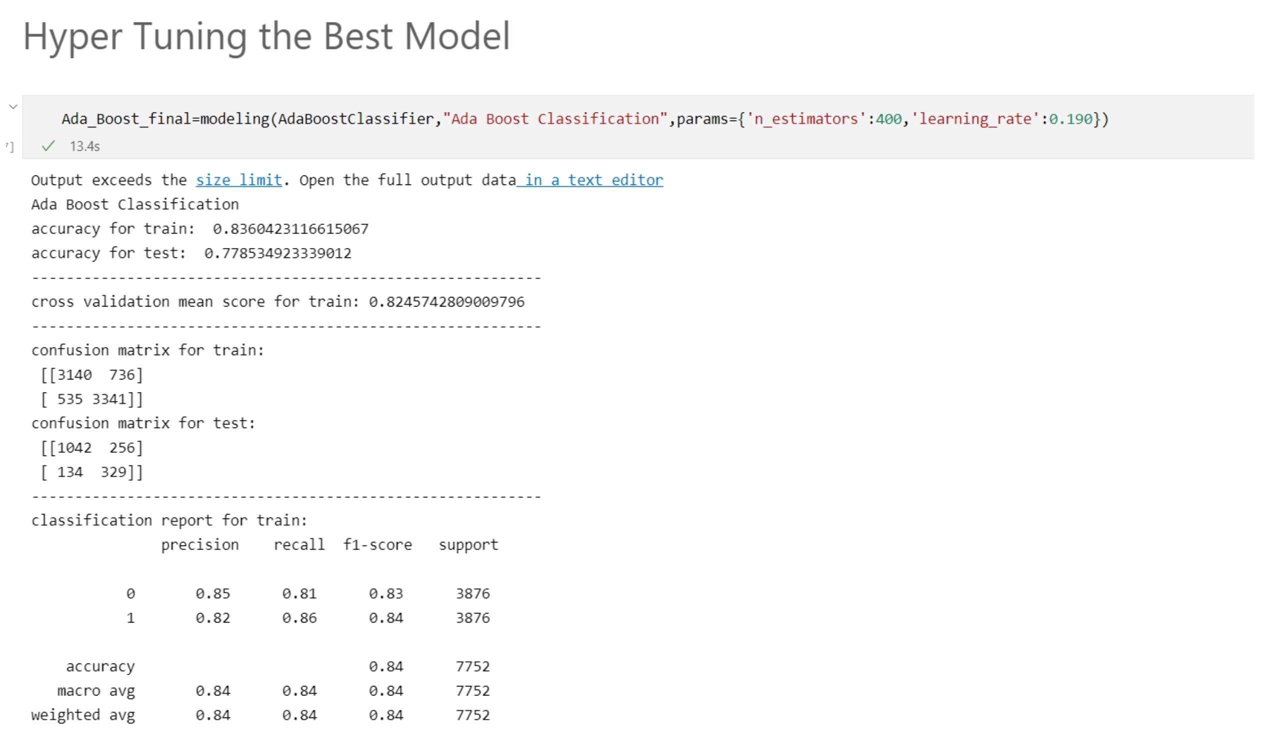
```
#gsv=GridSearchCV(estimator=Ada_Boost,param_grid={'n_estimators':[50,100,150,200,250,300,350,400,450,500]},cv=5,scoring='accuracy',verbose=1)
✓ 0.4s
```



```
#best_model.fit(x1_train,y1_train)
✓ 0.1s
```



```
#best_model.best_params_
✓ 0.1s
```



Hyper Tuning the Best Model

```
Ada_Boost_final=modeling(AdaBoostClassifier,"Ada Boost Classification",params={'n_estimators':400,'learning_rate':0.190})
✓ 13.4s
```

```
Output exceeds the size limit. Open the full output data in a text editor
Ada Boost Classification
accuracy for train:  0.8360423116615067
accuracy for test:  0.778534923339012
------------------------------------------------------------
cross validation mean score for train: 0.8245742809009796
------------------------------------------------------------
confusion matrix for train:
 [[3140  736]
 [ 535 3341]]
confusion matrix for test:
 [[1042  256]
 [ 134  329]]
------------------------------------------------------------
classification report for train:
               precision    recall  f1-score   support

           0       0.85      0.81      0.83      3876
           1       0.82      0.86      0.84      3876

    accuracy                           0.84      7752
   macro avg       0.84      0.84      0.84      7752
weighted avg       0.84      0.84      0.84      7752
```

The Best Model is been Hyper Tuned and accuracy is been improved further 0.1 % from its previous accuracy.

Chapter 6

Conclusion

The Best Pick Model is the Ada Boost Classifier for this problem, where the model is again Hyper Tuned to yield the best result possible.

The parameters used to adjust the model to yield better accuracy are:

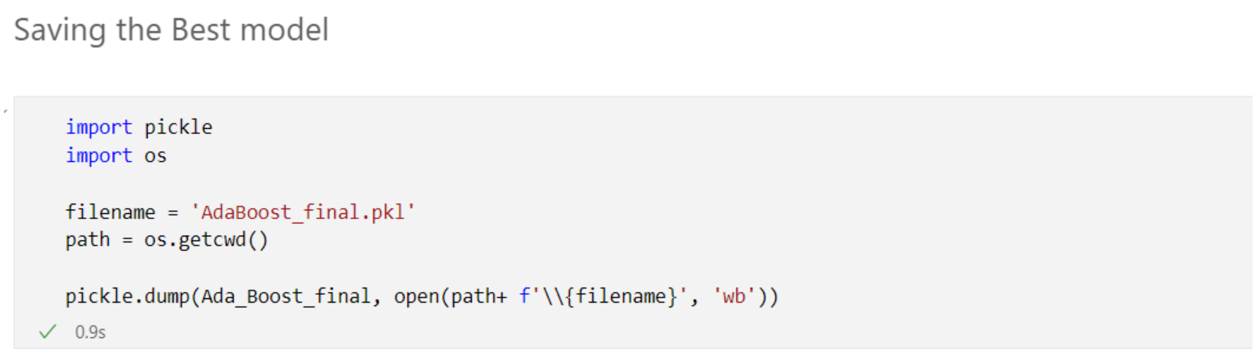
n estimators= 400

Learning rate = 0.190

The model result is based on 95% confidence interval, where the Mean Score of Training Accuracy is 82.45% and that of Test Accuracy is 77.85%.

Saving the Model:

The trained models in a file and restore them in order to reuse it to compare the model with other models, to test the model on a new data. The saving of data is called Serialization, while restoring the data is called Deserialization. This can be done with the help of Pickling.



Saving the Best model

```
import pickle
import os

filename = 'AdaBoost_final.pkl'
path = os.getcwd()

pickle.dump(Ada_Boost_final, open(path+ f'\\{filename}', 'wb'))
```

✓ 0.9s

Pickling:

The pickle module implements binary protocols for serializing and deserializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshaling," 1 or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling". In real world sceanario, the use pickling and unpickling are widespread as they allow us to easily transfer data from one server/system to another and then store it in a file or database

Chapter 7

Model Deployment

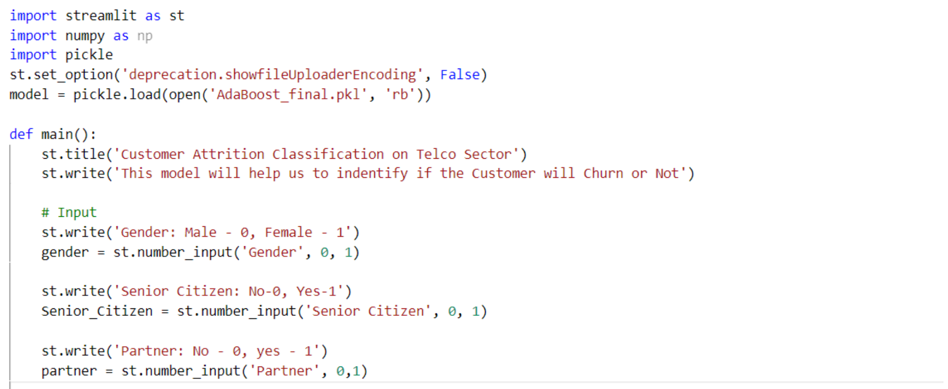
Model Deployment:

Deploying a machine learning model, known as model deployment, simply means to integrate a machine learning model and integrate it into an existing production environment where it can take in an input and return an output. The purpose of deploying your model is so that you can make the predictions from a trained ML model available to others, whether that be users, management, or other systems. Model deployment is closely related to ML systems architecture, which refers to the arrangement and interactions of software components within a system to achieve a predefined goal.

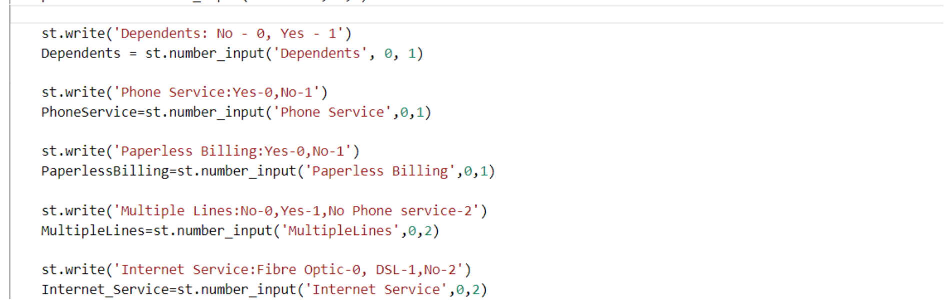
Before we deploy a model, there are a couple of criteria that your machine learning model needs to achieve before it’s ready for deployment:

- ✓ Portability: this refers to the ability of your software to be transferred from one machine or system to another. A portable model is one with a relatively low response time and one that can be rewritten with minimal effort.
- ✓ Scalability: this refers to how large your model can scale. A scalable model is one that doesn’t need to be redesigned to maintain its performance

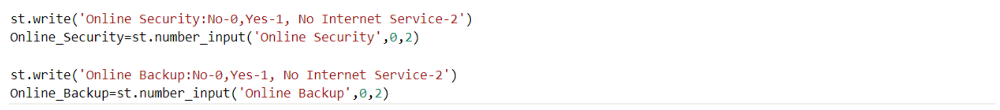
Real-time: Model deployed in Real-time in Local Server using Streamlit



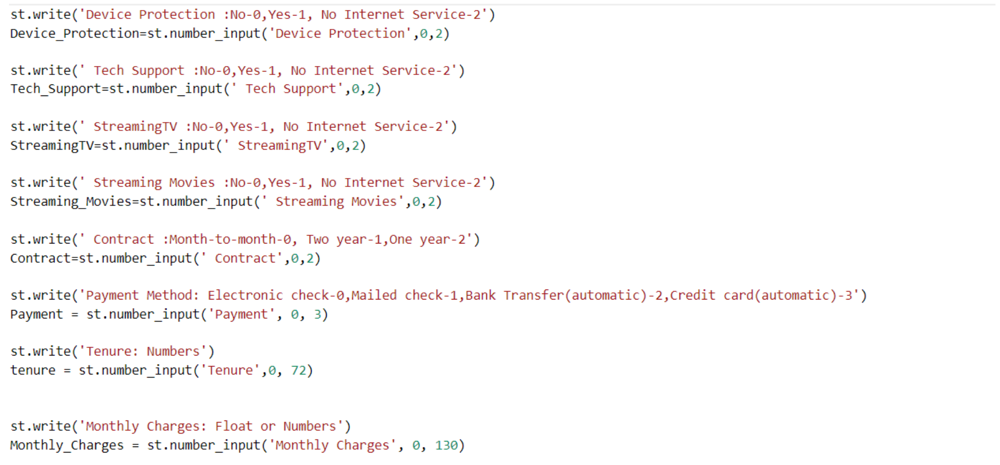
```python
import streamlit as st
import numpy as np
import pickle
st.set_option('deprecation.showfileUploaderEncoding', False)
model = pickle.load(open('AdaBoost_final.pkl', 'rb'))

def main():
    st.title('Customer Attrition Classification on Telco Sector')
    st.write('This model will help us to indentify if the Customer will Churn or Not')

    # Input
    st.write('Gender: Male - 0, Female - 1')
    gender = st.number_input('Gender', 0, 1)

    st.write('Senior Citizen: No-0, Yes-1')
    Senior_Citizen = st.number_input('Senior Citizen', 0, 1)

    st.write('Partner: No - 0, yes - 1')
    partner = st.number_input('Partner', 0,1)
```



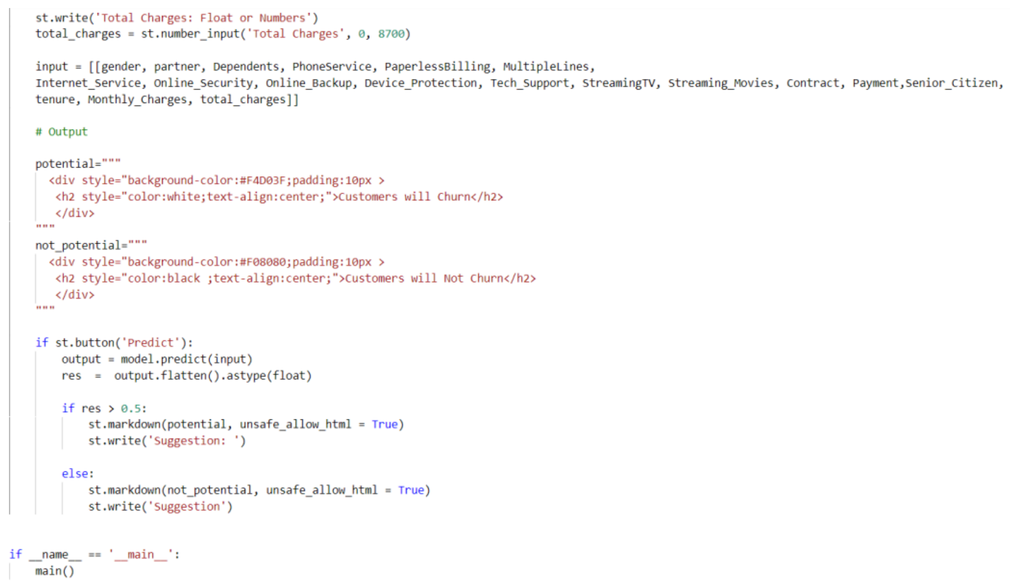
```python
    st.write('Dependents: No - 0, Yes - 1')
    Dependents = st.number_input('Dependents', 0, 1)

    st.write('Phone Service:Yes-0,No-1')
    PhoneService=st.number_input('Phone Service',0,1)

    st.write('Paperless Billing:Yes-0,No-1')
    PaperlessBilling=st.number_input('Paperless Billing',0,1)

    st.write('Multiple Lines:No-0,Yes-1,No Phone service-2')
    MultipleLines=st.number_input('MultipleLines',0,2)

    st.write('Internet Service:Fibre Optic-0, DSL-1,No-2')
    Internet_Service=st.number_input('Internet Service',0,2)
```



```python
st.write('Online Security:No-0,Yes-1, No Internet Service-2')
Online_Security=st.number_input('Online Security',0,2)

st.write('Online Backup:No-0,Yes-1, No Internet Service-2')
Online_Backup=st.number_input('Online Backup',0,2)
```

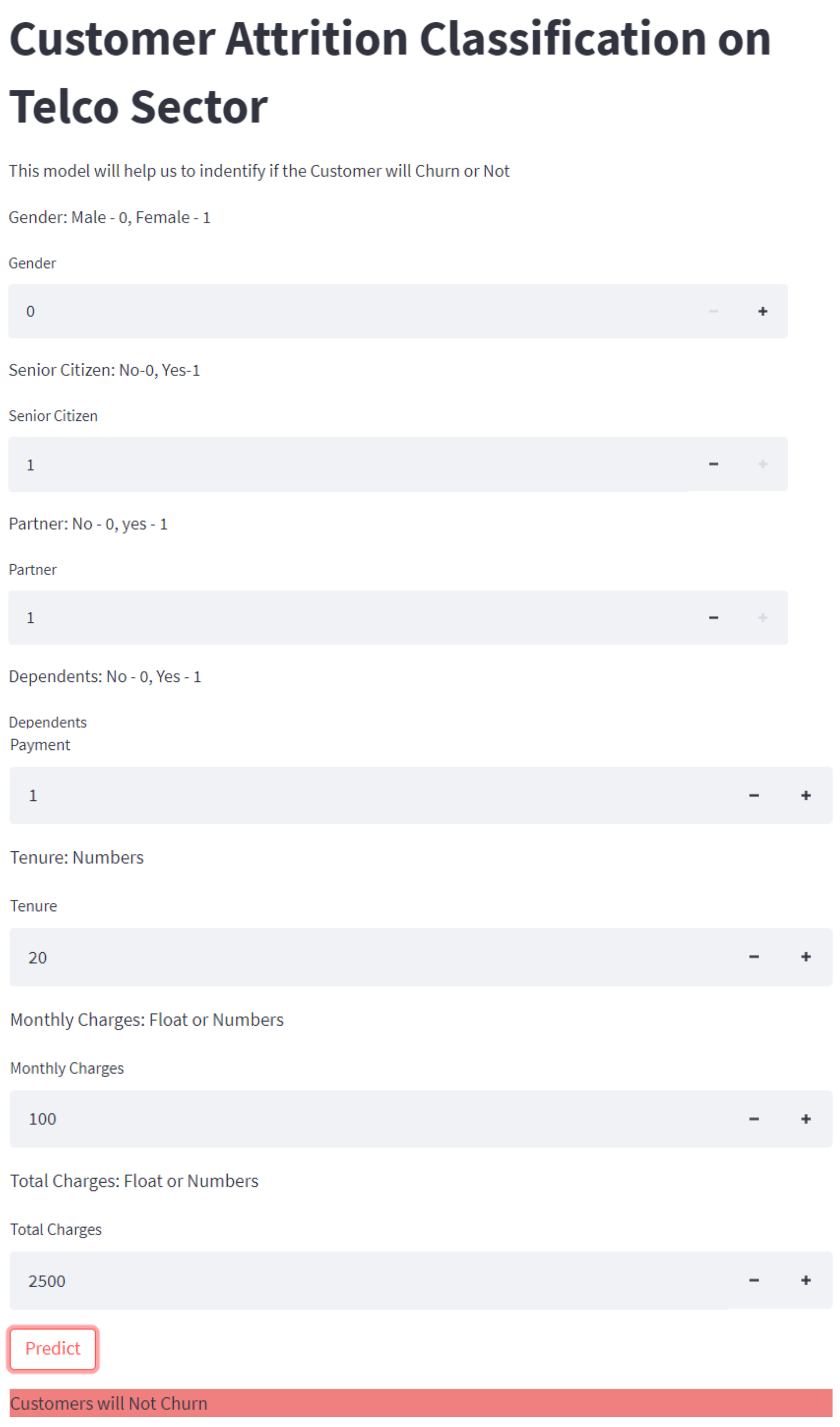


```python
st.write('Device Protection :No-0,Yes-1, No Internet Service-2')
Device_Protection=st.number_input('Device Protection',0,2)

st.write(' Tech Support :No-0,Yes-1, No Internet Service-2')
Tech_Support=st.number_input(' Tech Support',0,2)

st.write(' StreamingTV :No-0,Yes-1, No Internet Service-2')
StreamingTV=st.number_input(' StreamingTV',0,2)

st.write(' Streaming Movies :No-0,Yes-1, No Internet Service-2')
Streaming_Movies=st.number_input(' Streaming Movies',0,2)

st.write(' Contract :Month-to-month-0, Two year-1,One year-2')
Contract=st.number_input(' Contract',0,2)

st.write('Payment Method: Electronic check-0,Mailed check-1,Bank Transfer(automatic)-2,Credit card(automatic)-3')
Payment = st.number_input('Payment', 0, 3)

st.write('Tenure: Numbers')
tenure = st.number_input('Tenure',0, 72)


st.write('Monthly Charges: Float or Numbers')
Monthly_Charges = st.number_input('Monthly Charges', 0, 130)
```



```python
    st.write('Total Charges: Float or Numbers')
    total_charges = st.number_input('Total Charges', 0, 8700)

    input = [[gender, partner, Dependents, PhoneService, PaperlessBilling, MultipleLines,
    Internet_Service, Online_Security, Online_Backup, Device_Protection, Tech_Support, StreamingTV, Streaming_Movies, Contract, Payment,Senior_Citizen,
    tenure, Monthly_Charges, total_charges]]

    # Output

    potential="""
      <div style="background-color:#F4D03F;padding:10px >
       <h2 style="color:white;text-align:center;">Customers will Churn</h2>
       </div>
    """

    not_potential="""
      <div style="background-color:#F08080;padding:10px >
       <h2 style="color:black ;text-align:center;">Customers will Not Churn</h2>
       </div>
    """

    if st.button('Predict'):
        output = model.predict(input)
        res  =  output.flatten().astype(float)

        if res > 0.5:
            st.markdown(potential, unsafe_allow_html = True)
            st.write('Suggestion: ')

        else:
            st.markdown(not_potential, unsafe_allow_html = True)
            st.write('Suggestion')


if __name__ == '__main__':
    main()
```

Output:


Customer Attrition Classification on Telco Sector

This model will help us to indentify if the Customer will Churn or Not

Gender: Male - 0, Female - 1

Gender

0

Senior Citizen: No-0, Yes-1

Senior Citizen

1

Partner: No - 0, yes - 1

Partner

1

Dependents: No - 0, Yes - 1

Dependents
Payment

1

Tenure: Numbers

Tenure

20

Monthly Charges: Float or Numbers

Monthly Charges

100

Total Charges: Float or Numbers

Total Charges

2500

Customers will Not Churn

The model is Successfully deployed using platform called Streamlit. By giving the input parameters in real time we can earn result and suggestion based on the result what we earned.

References:

- https://pypi.org/
- https://pandas.pydata.org/docs/
- https://numpy.org/doc/stable/
- https://scikit-learn.org/stable/
- https://streamlit.io/