Customer Churn Analysis: Machine Learning Project

Author: **ARCHANA KUMARI**

Topics covered:

I. Problem Definition

II. Data Analysis

III. EDA Concluding Remark

IV. Pre-Processing Pipeline

V. Building Machine Learning Models

VI. Concluding Remarks.

- **What Is Customer Churn?**

Customer churn is the percentage of customers that stopped using your company's product or service during a certain time frame.

- **Why Is Customer Churn Rate Important?**

An increase in customer retention of just 5% can create at least a 25% increase in profit. This is because returning customers will likely spend 67% more on your company's products and services. As a result, your company can spend less on the operating costs of having to acquire new customers.

- **How to Calculate Churn Rate?**

1. Determine a time period: monthly, annual, or quarterly.
2. Determine the number of customers you had at the beginning of the time period.
3. Determine the number of customers that churned by the end of the time period.
4. Divide the number of lost customers by the number of customers you had prior to the churn.
5. Multiply that number by 100.

- **Formula:**

Customer Churn Rate = (Lost Customers ÷ Total Customers at the Start of Time Period) x 100

1. Problem Definition

Customer churn is when a company's customers stop doing business with that company. Businesses are very keen on measuring churn because keeping an existing customer is far less expensive than acquiring a new customer. New business involves working leads through a sales funnel, using marketing and sales budgets to gain additional customers. Existing customers will often have a higher volume of service consumption and can generate additional customer referrals.

Customer retention can be achieved with good customer service and products. But the most effective way for a company to prevent attrition of customers is to truly know them. The vast volumes of data collected about customers can be used to build churn prediction models. Knowing who is most likely to defect means that a company can prioritise focused marketing efforts on that subset of their customer base.

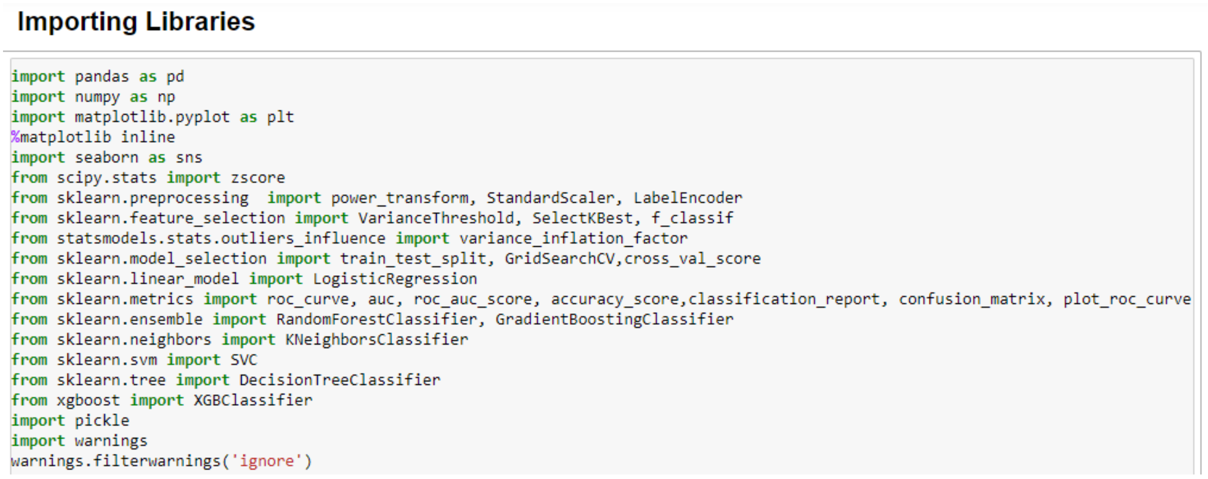
Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

Here we are examining customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

- You can find the original project details on Kaggle: https://www.kaggle.com/c/customer-churn-prediction-2020/overview

2. Data Analysis.

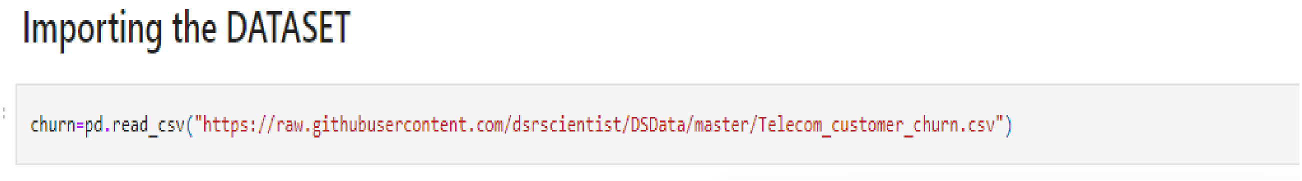
- **First thing before analysis is to import necessary libraries**



Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy.stats import zscore
from sklearn.preprocessing  import power_transform, StandardScaler, LabelEncoder
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score,classification_report, confusion_matrix, plot_roc_curve
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
import pickle
import warnings
warnings.filterwarnings('ignore')
```

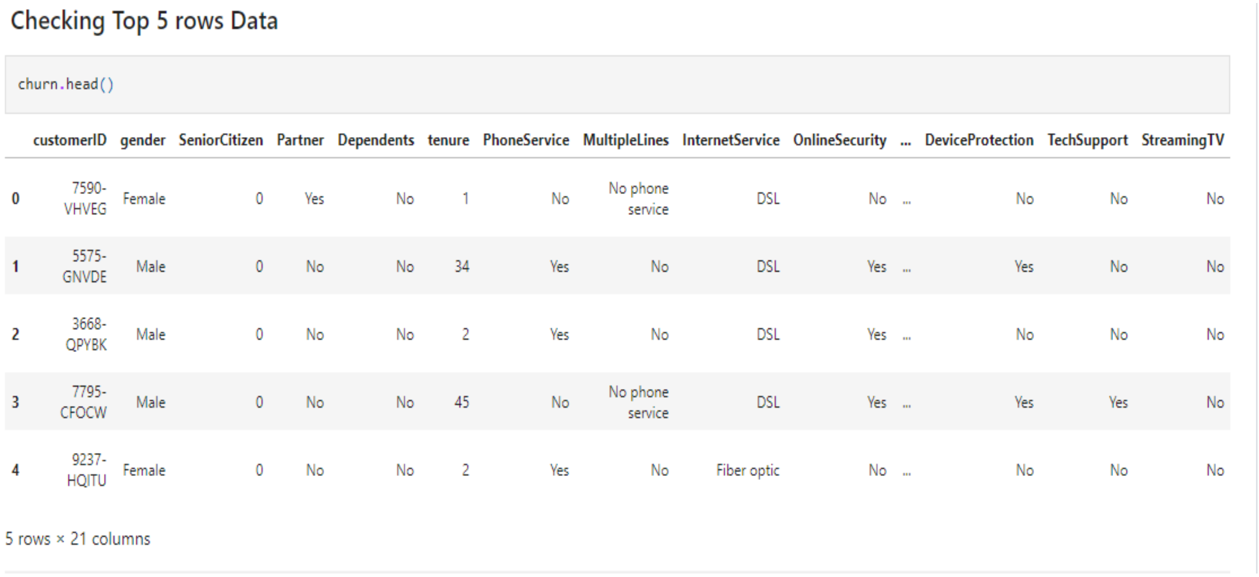
- **Second for analysis import Dataset (Telecom_customer_churn.csv)**



Importing the DATASET

```
churn=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/DSData/master/Telecom_customer_churn.csv")
```

- **Checking Top 5 rows of Dataset to analyse the data/values of dataset**



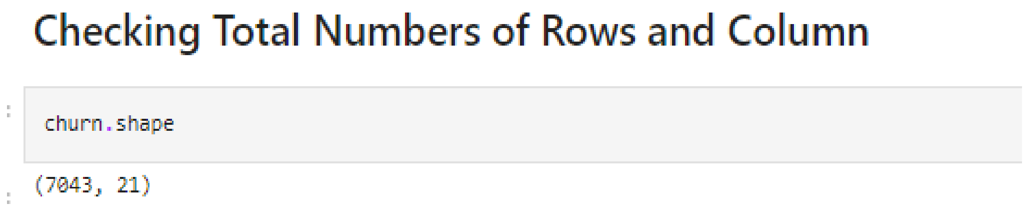
Checking Top 5 rows Data

```
churn.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	No
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	No

5 rows × 21 columns

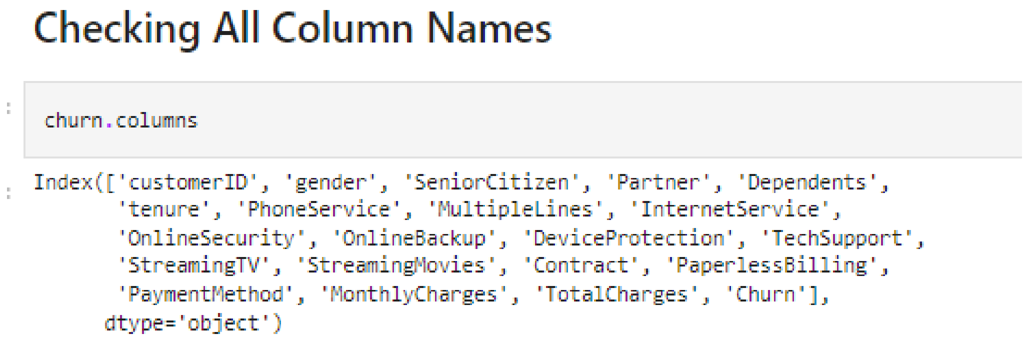
- **Analysing Total Numbers of Rows and Column and All Column Names**



Checking Total Numbers of Rows and Column

```
churn.shape
```

```
(7043, 21)
```



Checking All Column Names

```
churn.columns
```

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

We observed that this dataset contains 7043 rows and 21 columns. Names of column with brief description are:

Features columns:

- **CustomerID:** Customer ID unique for each customer
- **gender:** Whether the customer is a male or a female
- **SeniorCitizen:** Whether the customer is a senior citizen or not (1, 0)
- **Partner:** Whether the customer has a partner or not (Yes, No)
- **Dependent:** Whether the customer has dependents or not (Yes, No)
- **PhoneService:** Whether the customer has a phone service or not (Yes, No)
- **MultipeLines:** Whether the customer has multiple lines or not (Yes, No, No phone service)
- **InternetService:** Customer's internet service provider (DSL, Fiber optic, No)
- **OnlineSecurity:** Whether the customer has online security or not (Yes, No, No internet service)
- **OnlineBackup:** Whether the customer has an online backup or not (Yes, No, No internet service)
- **DeviceProtection:** Whether the customer has device protection or not (Yes, No, No internet service)
- **TechSupport:** Whether the customer has tech support or not (Yes, No, No internet service)
- **StreamingTV:** Whether the customer has streaming TV or not (Yes, No, No internet service)
- **StreamingMovies:** Whether the customer has streaming movies or not (Yes, No, No internet service)
- **Contract:** The contract term of the customer (Month-to-month, One year, Two years)
- **PaperlessBilling:** The contract term of the customer (Month-to-month, One year, Two years)
- **PaymentMethod:** The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- **Tenure:** Number of months the customer has stayed with the company
- **MonthlyCharges:** The amount charged to the customer monthly
- **TotalCharges:** The total amount charged to the customer

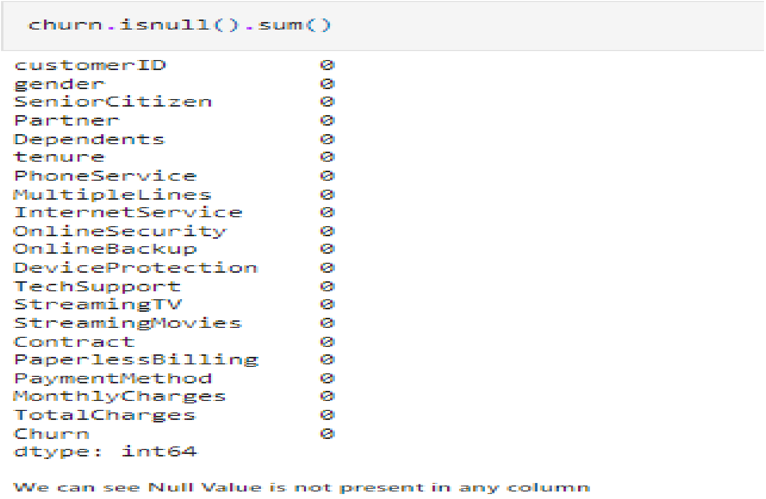
Target column:

- **Churn**: Whether the customer churned or not (Yes or No)

These features can also be subdivided into:

- **Demographic customer information:** gender, SeniorCitizen, Partner, Dependents
- **Services Information:** PhoneService, MultipleLines, InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies
- **Customer account information:** tenure, Contract, PaperlessBilling, PaymentMethod, MonthlyCharges, TotalCharges

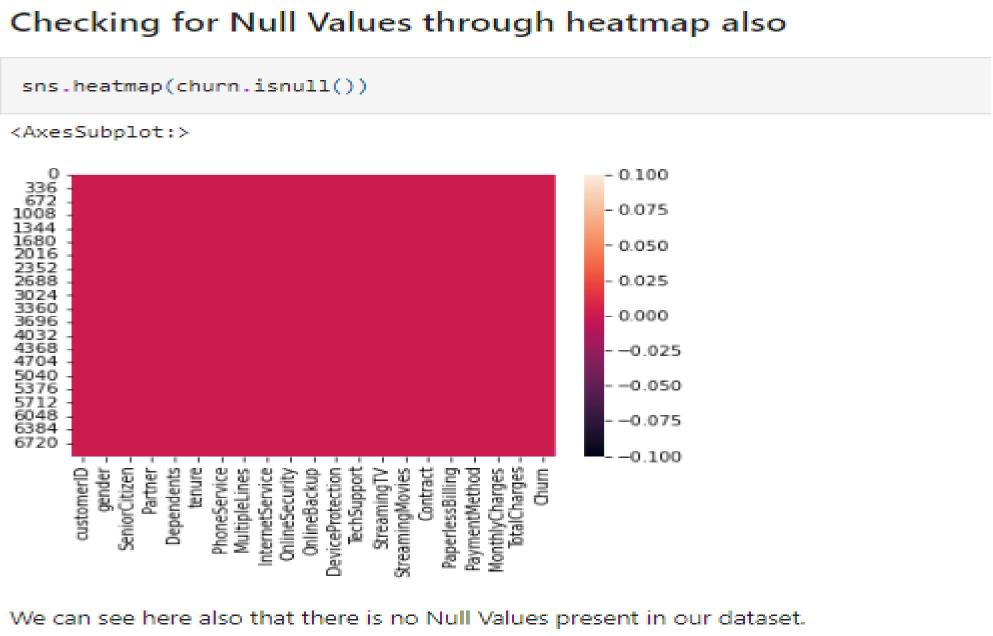
➢ **Analysing Null Values**



```
churn.isnull().sum()
```

```
customerID          0
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
```

We can see Null Value is not present in any column



Checking for Null Values through heatmap also

```
sns.heatmap(churn.isnull())
```

```
<AxesSubplot:>
```

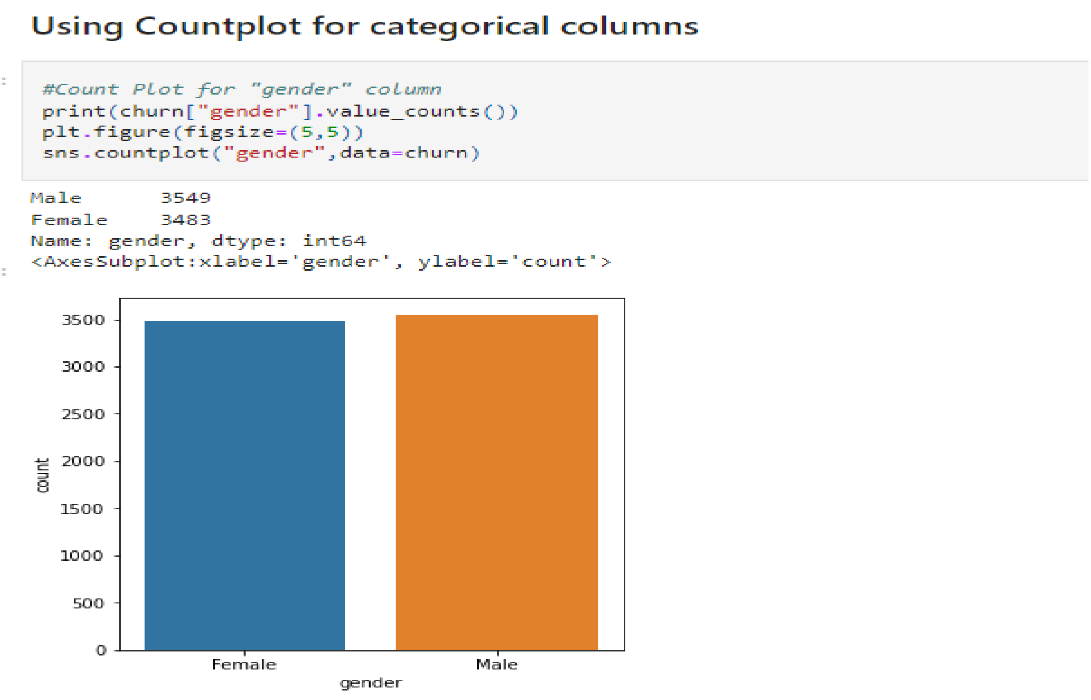
We can see here also that there is no Null Values present in our dataset.

We can observe there are no Null Values in Dataset.

➢ Let’s plot to Analyse each column’s data through Data Visualization

- **<u>Univariant Analysis</u>**

  - Using countplot (for Categorical Columns)

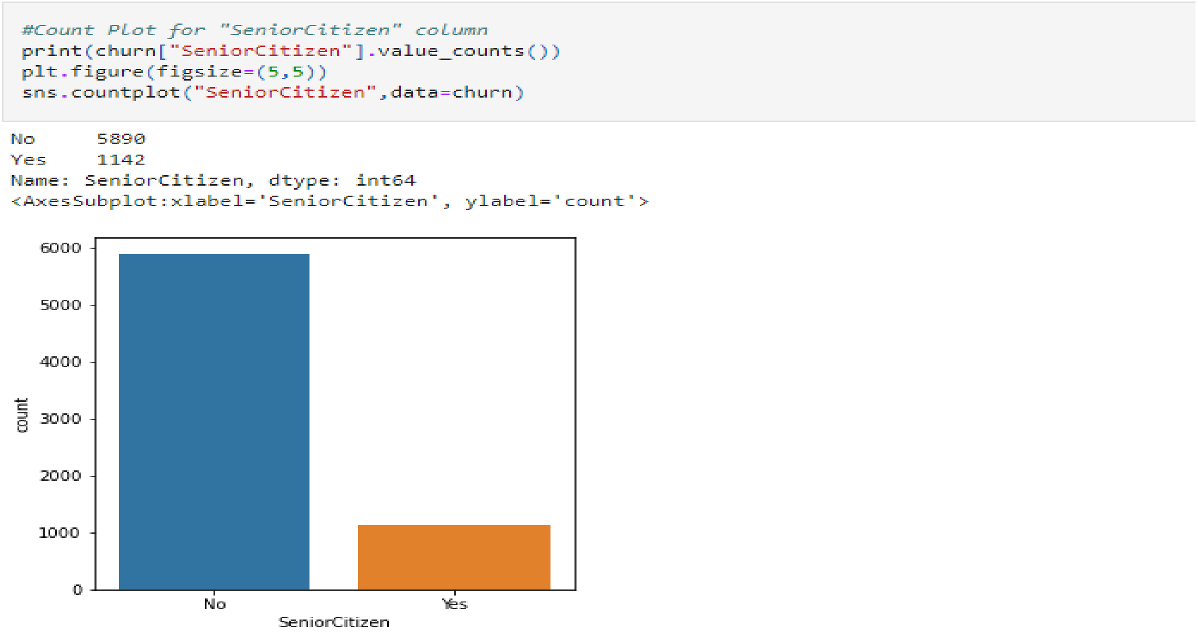


Using Countplot for categorical columns

```
#Count Plot for "gender" column
print(churn["gender"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("gender",data=churn)
```

```
Male      3549
Female    3483
Name: gender, dtype: int64
<AxesSubplot:xlabel='gender', ylabel='count'>
```

- We can see Male (Total No= 3555) are more compare to Female (Total No= 3488).



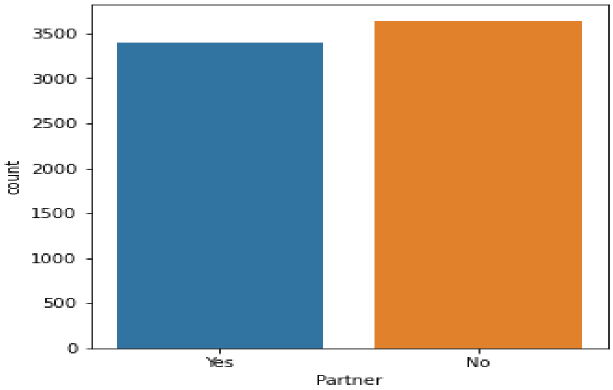
```
#Count Plot for "SeniorCitizen" column
print(churn["SeniorCitizen"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("SeniorCitizen",data=churn)
```

```
No     5890
Yes    1142
Name: SeniorCitizen, dtype: int64
<AxesSubplot:xlabel='SeniorCitizen', ylabel='count'>
```

- We can see SeniorCitizen are less (Total No= 1142) and not SeniorCitizen are more (Total No= 5901).

```
#Count Plot for "Partner" column
print(churn["Partner"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("Partner",data=churn)
```

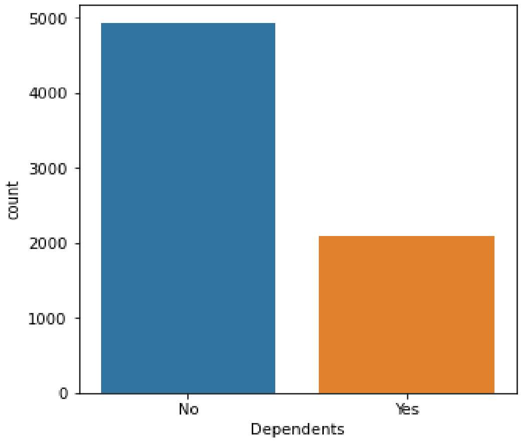
```
No      3639
Yes     3393
Name: Partner, dtype: int64
<AxesSubplot:xlabel='Partner', ylabel='count'>
```



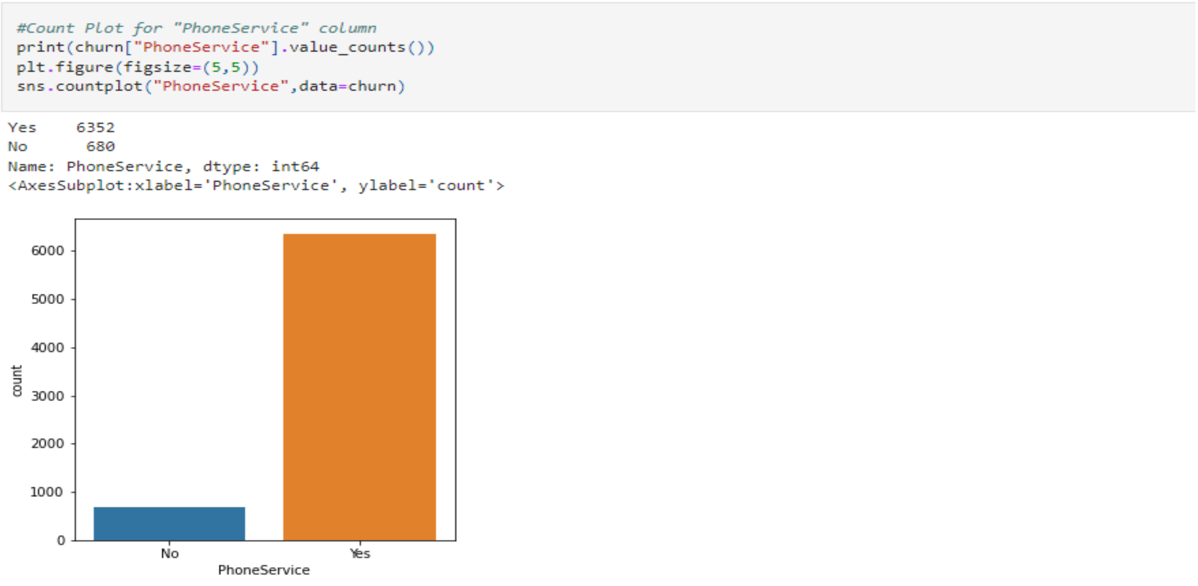
- We can see Partner are less (Total No= 3402) and not having Partner are more (Total No= 3641).

```
#Count Plot for "Dependents" column
print(churn["Dependents"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("Dependents",data=churn)
```

```
No      4933
Yes     2099
Name: Dependents, dtype: int64
<AxesSubplot:xlabel='Dependents', ylabel='count'>
```



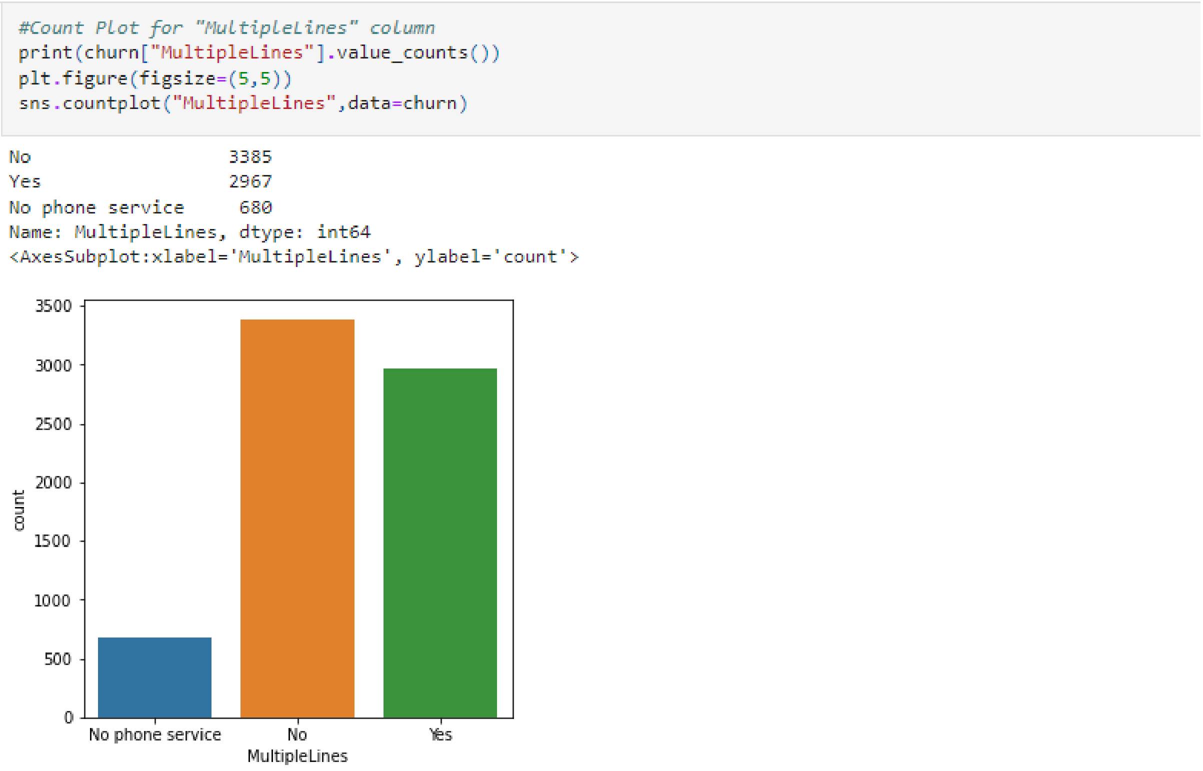
- We can see Dependents are less (Total No= 2110) and not having Dependents are more (Total No= 4933).



```
#Count Plot for "PhoneService" column
print(churn["PhoneService"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("PhoneService",data=churn)
```

```
Yes    6352
No      680
Name: PhoneService, dtype: int64
<AxesSubplot:xlabel='PhoneService', ylabel='count'>
```

- We can see having PhoneService are more (Total No= 6361) and not having PhoneService are less (Total No= 682).



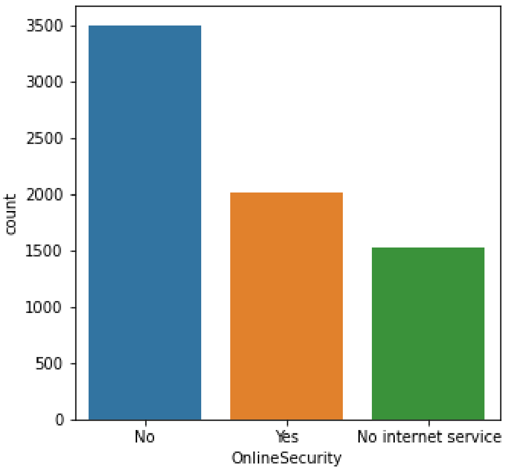
```
#Count Plot for "MultipleLines" column
print(churn["MultipleLines"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("MultipleLines",data=churn)
```

```
No                  3385
Yes                 2967
No phone service     680
Name: MultipleLines, dtype: int64
<AxesSubplot:xlabel='MultipleLines', ylabel='count'>
```

- We can see not having "MultipleLines" is more (Total No= 3390) compare to having "MultipleLines" (Total No= 2971).
- Having "No phone service" is least (Total No= 682)

```
#Count Plot for "OnlineSecurity" column
print(churn["OnlineSecurity"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("OnlineSecurity",data=churn)
```

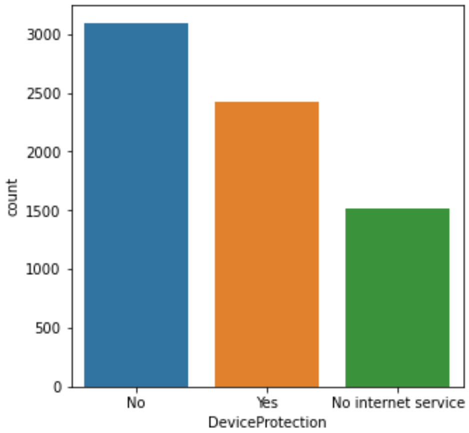
```
No                     3497
Yes                    2015
No internet service    1520
Name: OnlineSecurity, dtype: int64
<AxesSubplot:xlabel='OnlineSecurity', ylabel='count'>
```



- We can see not having "OnlineSecurity" is more (Total No= 3498) compare to having "OnlineSecurity" (Total No= 2019).
- Having "No internet service" is least (Total No= 1526)

```
#Count Plot for "DeviceProtection" column
print(churn["DeviceProtection"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("DeviceProtection",data=churn)
```

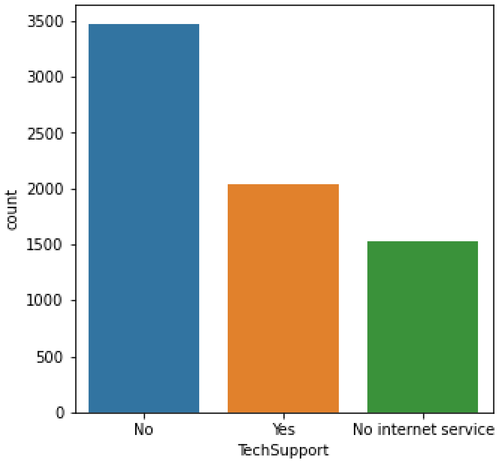
```
No                     3094
Yes                    2418
No internet service    1520
Name: DeviceProtection, dtype: int64
<AxesSubplot:xlabel='DeviceProtection', ylabel='count'>
```



- We can see not having "DeviceProtection" is more (Total No= 3095) compare to having "DeviceProtection" (Total No= 2422).
- Having "No internet service" is least (Total No= 1526)

```
#Count Plot for "TechSupport" column
print(churn["TechSupport"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("TechSupport",data=churn)
```

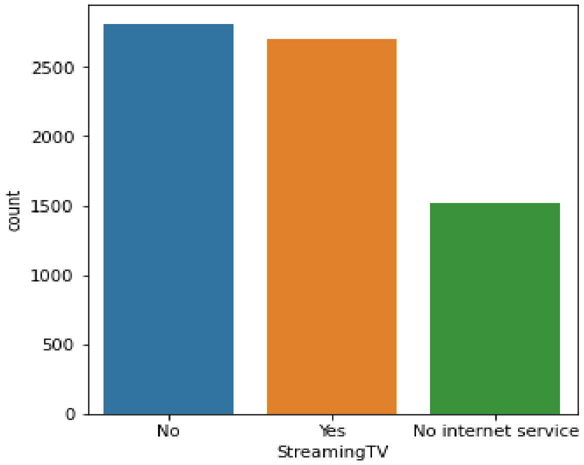
```
No                     3472
Yes                    2040
No internet service    1520
Name: TechSupport, dtype: int64
<AxesSubplot:xlabel='TechSupport', ylabel='count'>
```



- We can see not having "TechSupport" are more (Total No= 3473) compare to having "TechSupport" (Total No= 2044).
- Having "No internet service" is least (Total No= 1526)

```
#Count Plot for "StreamingTV" column
print(churn["StreamingTV"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("StreamingTV",data=churn)
```

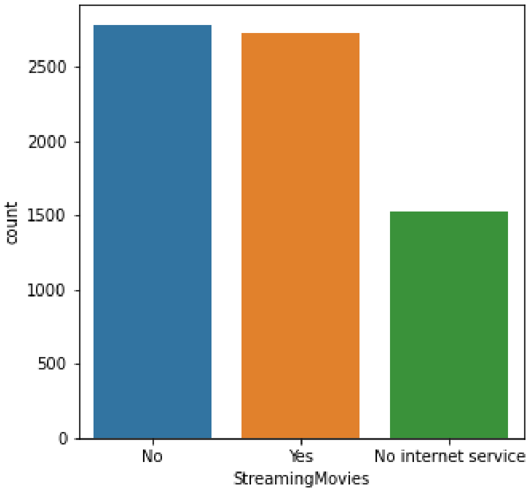
```
No                     2809
Yes                    2703
No internet service    1520
Name: StreamingTV, dtype: int64
<AxesSubplot:xlabel='StreamingTV', ylabel='count'>
```



- We can see not "StreamingTV" are more (Total No= 2810) compare to "StreamingTV" (Total No= 2707).
- Having "No internet service" is least (Total No= 1526)

```
#Count Plot for "StreamingMovies" column
print(churn["StreamingMovies"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("StreamingMovies",data=churn)
```

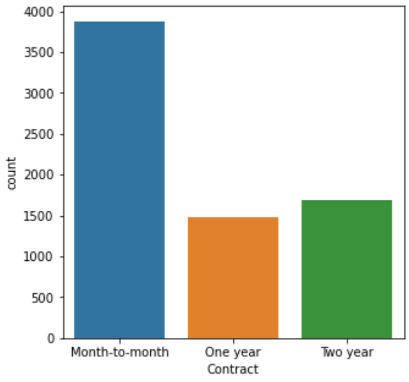
```
No                     2781
Yes                    2731
No internet service    1520
Name: StreamingMovies, dtype: int64
<AxesSubplot:xlabel='StreamingMovies', ylabel='count'>
```



- We can see not "StreamingMovies" are more (Total No= 2785) compare to "StreamingMovies" (Total No= 2732).
- Having "No internet service" is least (Total No= 1526)

```
#Count Plot for "Contract" column
print(churn["Contract"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("Contract",data=churn)
```

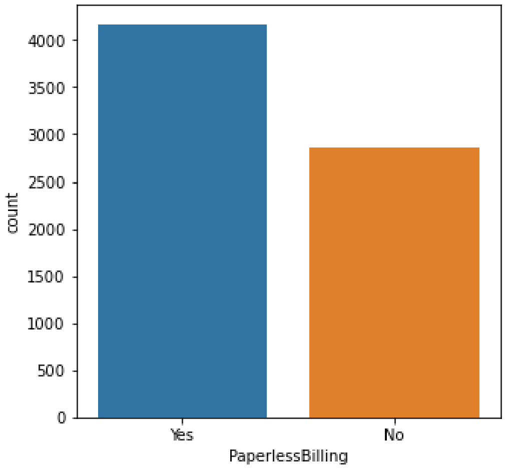
```
Month-to-month    3875
Two year          1685
One year          1472
Name: Contract, dtype: int64
<AxesSubplot:xlabel='Contract', ylabel='count'>
```



- We can see "Contract" done with customers for Month-to-month is more (Total No= 3875) and done for One year is less (Total No= 1473)

```
#Count Plot for "PaperlessBilling" column
print(churn["PaperlessBilling"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("PaperlessBilling",data=churn)
```

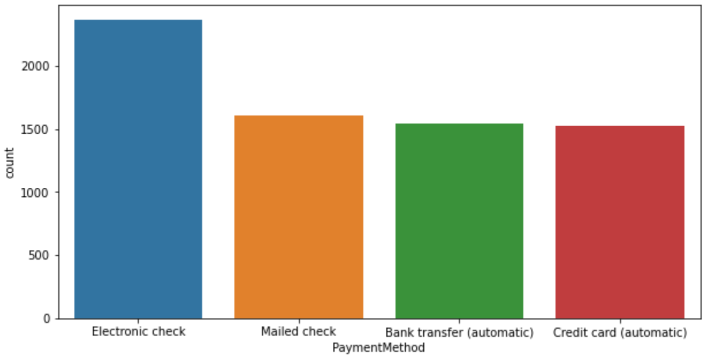
```
Yes    4168
No     2864
Name: PaperlessBilling, dtype: int64
<AxesSubplot:xlabel='PaperlessBilling', ylabel='count'>
```



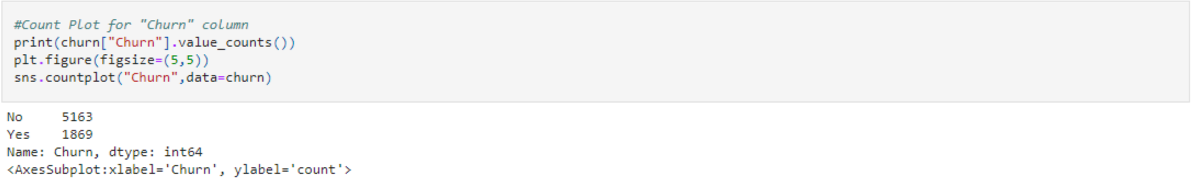
- We can see PaperlessBilling are done more (Total No= 4171) compare to not done PaperlessBilling (Total No= 2872).

```
#Count Plot for "PaymentMethod" column
print(churn["PaymentMethod"].value_counts())
plt.figure(figsize=(10,5))
sns.countplot("PaymentMethod",data=churn)
```

```
Electronic check             2365
Mailed check                 1604
Bank transfer (automatic)    1542
Credit card (automatic)      1521
Name: PaymentMethod, dtype: int64
<AxesSubplot:xlabel='PaymentMethod', ylabel='count'>
```

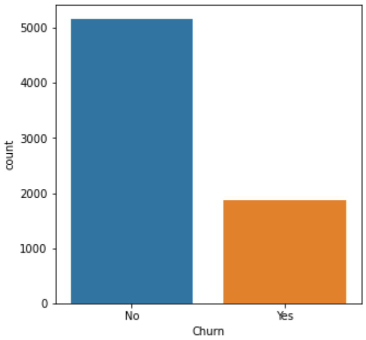


- We can see PaymentMethod "Electronic check" is used more (Total No= 2365) and "Credit card (automatic)" is used less (Total No= 1522).
- "Mailed check" (Total No= 1612) and "Bank transfer (automatic)" (Total No= 1544) is also used as PayemntMethod but less than "Electronic check" and more than "Credit card"



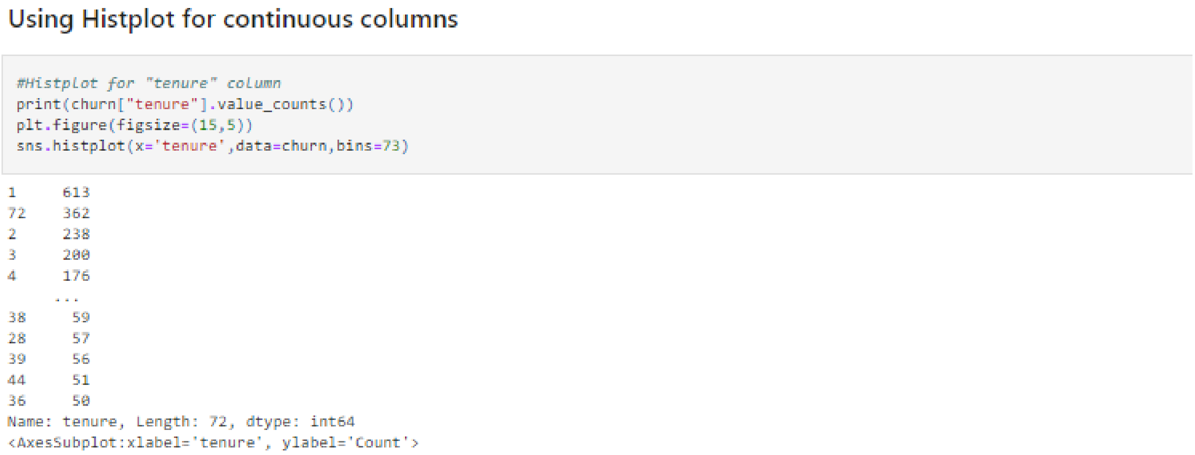
```
#Count Plot for "Churn" column
print(churn["Churn"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("Churn",data=churn)
```

```
No     5163
Yes    1869
Name: Churn, dtype: int64
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



- We can see Churn are less (Total No= 1869) and not having Churn are more (Total No= 5174). It shows 1869 number of customers no longer want to purchase goods and services from the business.

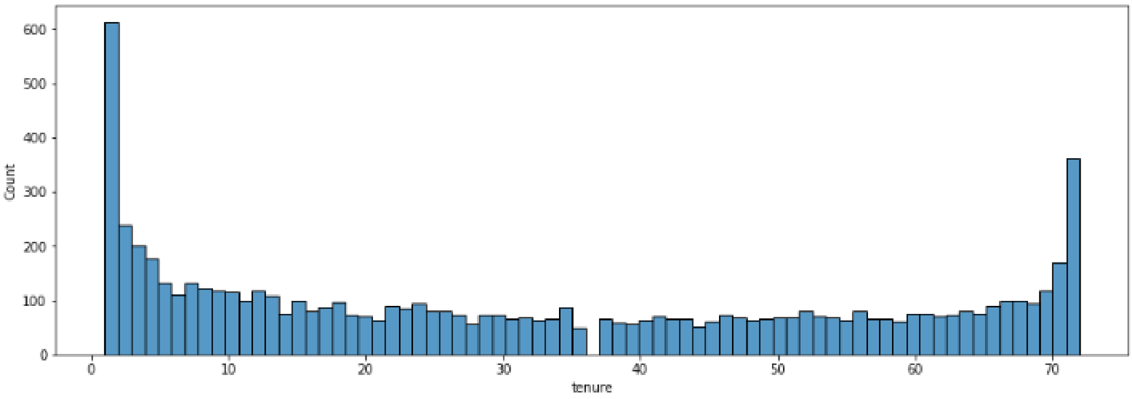
- Using Histplot (for Continuous Columns)



Using Histplot for continuous columns

```
#Histplot for "tenure" column
print(churn["tenure"].value_counts())
plt.figure(figsize=(15,5))
sns.histplot(x='tenure',data=churn,bins=73)
```

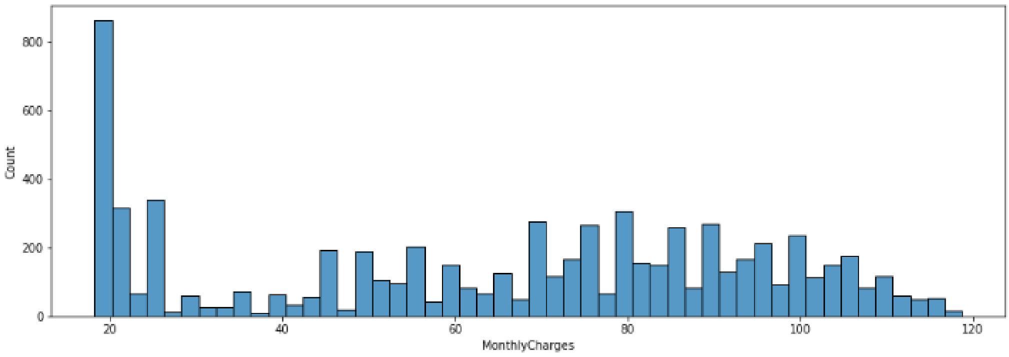
```
1     613
72    362
2     238
3     200
4     176
     ...
38     59
28     57
39     56
44     51
36     50
Name: tenure, Length: 72, dtype: int64
<AxesSubplot:xlabel='tenure', ylabel='Count'>
```



- We can see having tenure 1 is highest (Total No= 613)

```
#Histplot for "MonthlyCharges" column
print(churn["MonthlyCharges"].value_counts())
plt.figure(figsize=(15,5))
sns.histplot(x='MonthlyCharges',data=churn,bins=50)
```

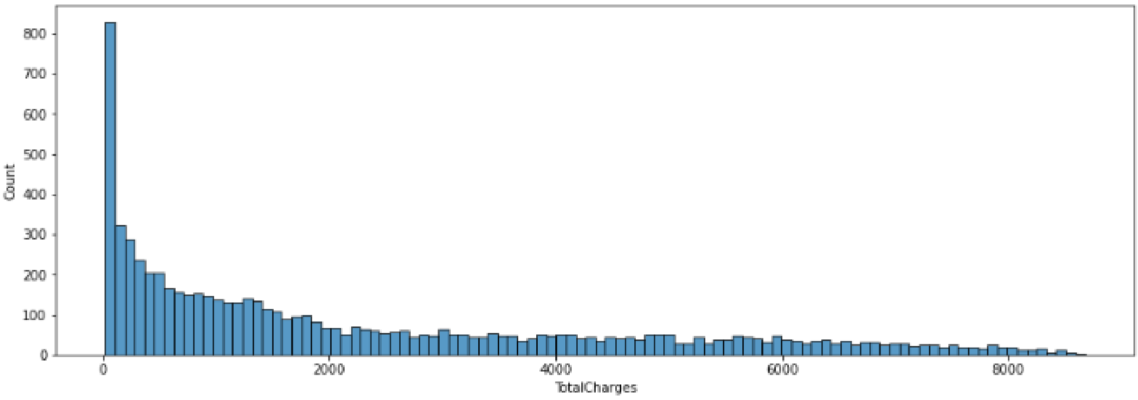
```
20.05     61
19.90     44
19.95     44
19.85     44
19.65     43
          ..
23.65      1
114.70     1
43.65      1
87.80      1
78.70      1
Name: MonthlyCharges, Length: 1584, dtype: int64
<AxesSubplot:xlabel='MonthlyCharges', ylabel='Count'>
```



- We can see having MonthlyCharges 20.05 is highest (Total No= 61)

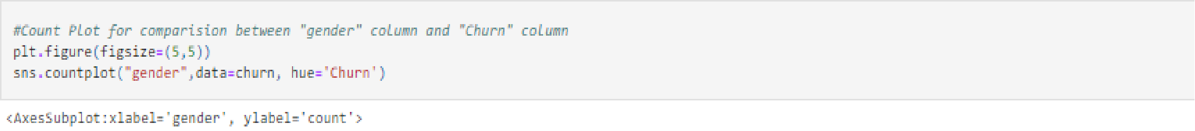
```
#Histplot for "TotalCharges" column
print(churn["TotalCharges"].value_counts())
plt.figure(figsize=(15,5))
sns.histplot(x='TotalCharges',data=churn,bins=100)
```

```
20.20      11
19.75       9
20.05       8
19.90       8
19.65       8
           ..
6849.40     1
692.35      1
130.15      1
3211.90     1
6844.50     1
Name: TotalCharges, Length: 6530, dtype: int64
<AxesSubplot:xlabel='TotalCharges', ylabel='Count'>
```



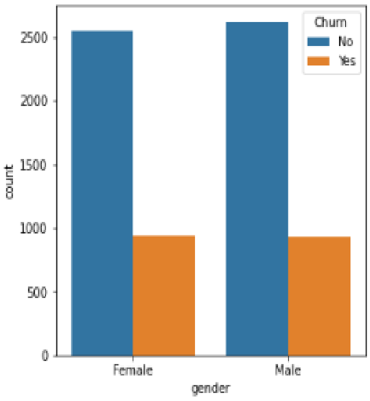
- We can see having TotalCharges 20.20 is highest (Total No= 11)

- Bivariant Analysis (for comparison)
  - Using countplot (for comparison between categorical column and target)

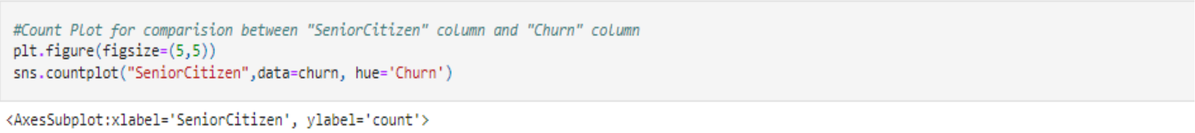


```
#Count Plot for comparision between "gender" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("gender",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```

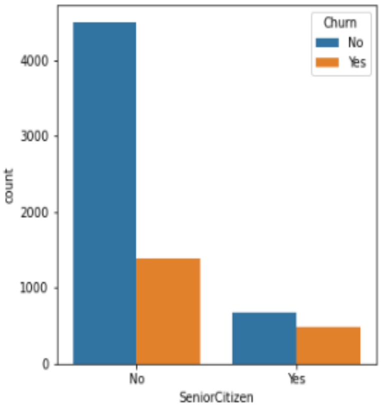


Male customer are few more compare to female and small difference in churn rate is of Male and Female. So, through Gender we cannot determine churn rate and we can say that it is not good measure for churn rate.



```
#Count Plot for comparision between "SeniorCitizen" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("SeniorCitizen",data=churn, hue='Churn')
```

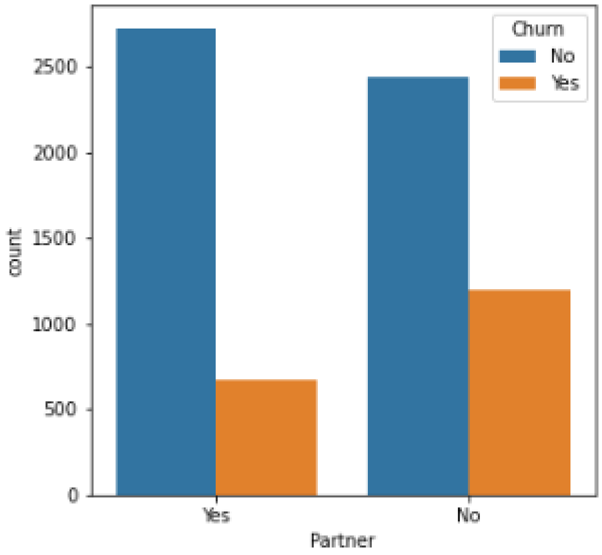
```
<AxesSubplot:xlabel='SeniorCitizen', ylabel='count'>
```



In SeniorCitizen also there is small diffence to determine churn rate but those who are young and not SeniorCitizen are determining churn rate as there is diffence in churn and not churn.

```
#Count Plot for comparision between "Partner" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("Partner",data=churn, hue='Churn')
```

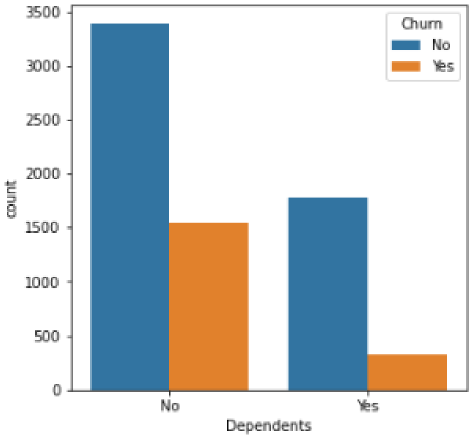
```
<AxesSubplot:xlabel='Partner', ylabel='count'>
```



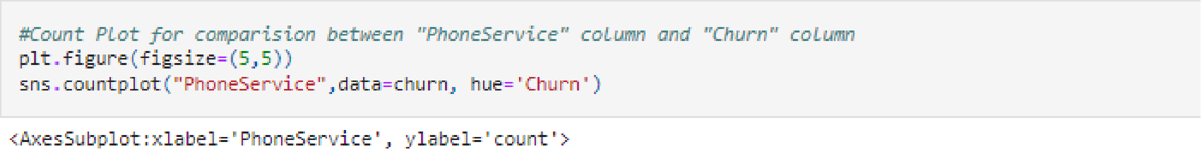
Those customer who have Partner are having less churn compare to those having Partner.

```
#Count Plot for comparision between "Dependents" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("Dependents",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='Dependents', ylabel='count'>
```

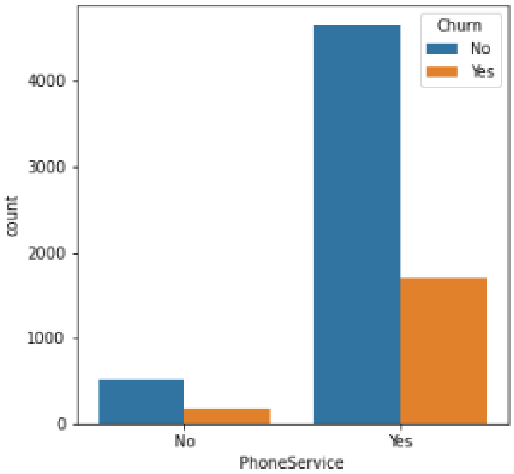


Those customer who have Dependents are having less churn but those customers who have no Dependents are having more churn.

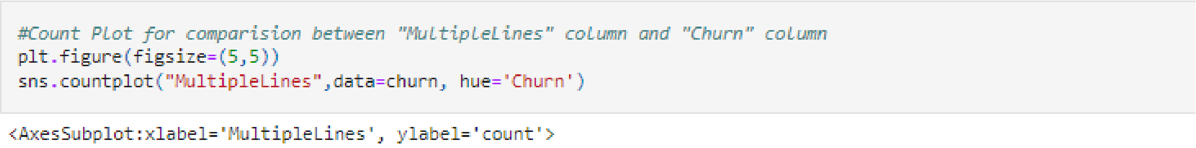


```
#Count Plot for comparision between "PhoneService" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("PhoneService",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='PhoneService', ylabel='count'>
```

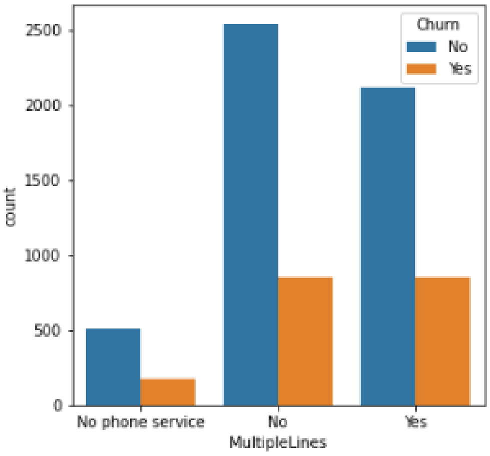


Those customer who have PhoneService are having less churn but more than those who have No PhoneService.

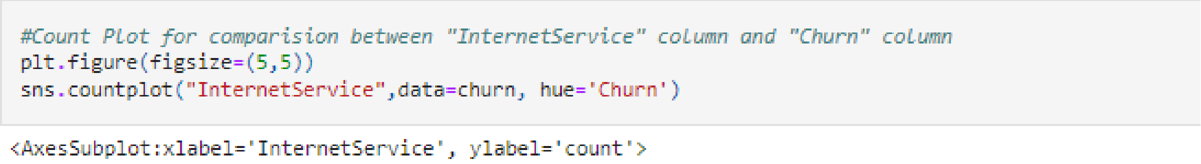


```
#Count Plot for comparision between "MultipleLines" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("MultipleLines",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='MultipleLines', ylabel='count'>
```

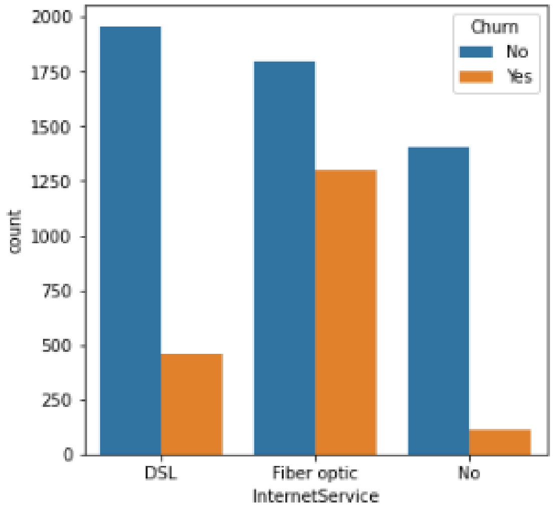


MultipleLines having No Phone Service having less churn and those having or not having MultipleLines have equal churn rate.

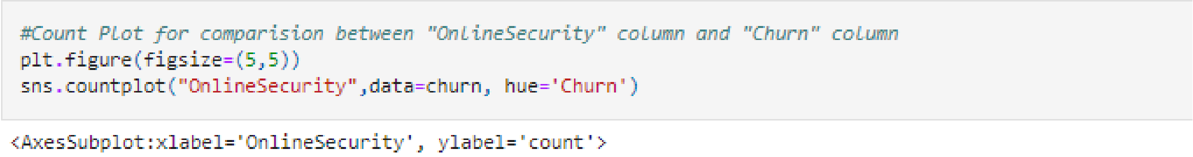


```
#Count Plot for comparision between "InternetService" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("InternetService",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='InternetService', ylabel='count'>
```

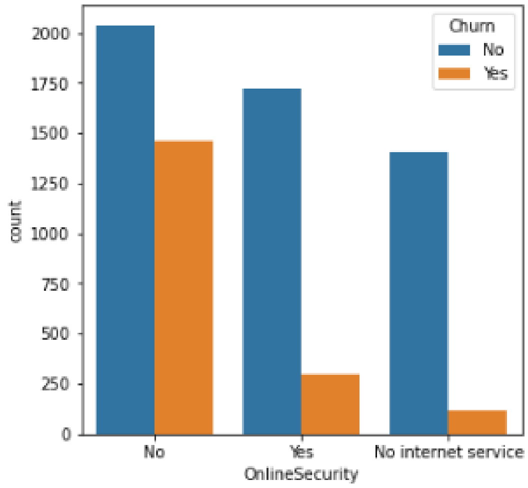


No InternetService having less churn and those having Fiber Optic InternetService are having more churn.



```
#Count Plot for comparision between "OnlineSecurity" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("OnlineSecurity",data=churn, hue='Churn')
```

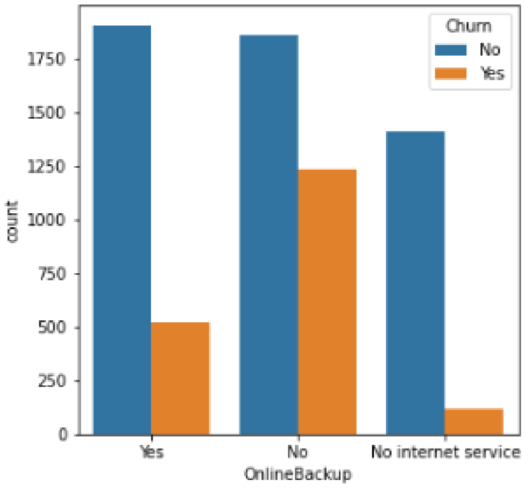
```
<AxesSubplot:xlabel='OnlineSecurity', ylabel='count'>
```



OnlineSecurity having No InternetService are having less churn and not having OnlineSecurity are having more churn.

```
#Count Plot for comparision between "OnlineBackup" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("OnlineBackup",data=churn, hue='Churn')
```

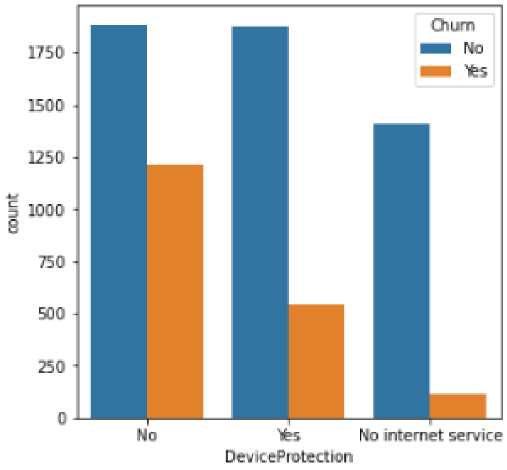
```
<AxesSubplot:xlabel='OnlineBackup', ylabel='count'>
```



OnlineBackup having No InternetService are having less churn and not having OnlineBackup are having more churn.

```
#Count Plot for comparision between "DeviceProtection" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("DeviceProtection",data=churn, hue='Churn')
```

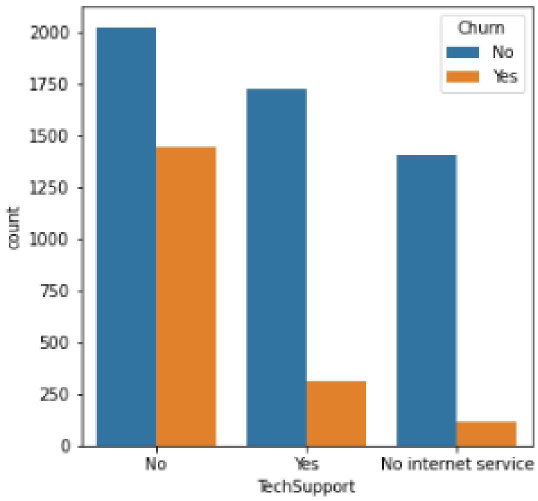
```
<AxesSubplot:xlabel='DeviceProtection', ylabel='count'>
```



DeviceProtection having No InternetService are having less churn and not having DeviceProtection are having more churn.

```
#Count Plot for comparision between "TechSupport" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("TechSupport",data=churn, hue='Churn')
```

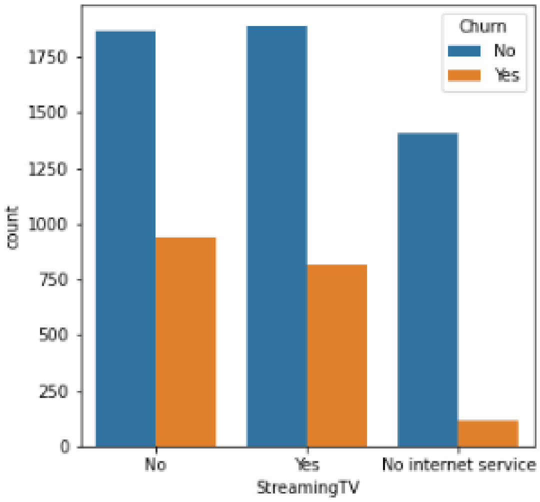
```
<AxesSubplot:xlabel='TechSupport', ylabel='count'>
```



TechSupport having No InternetService are having less churn and not having TechSupport are having more churn.

```
#Count Plot for comparision between "StreamingTV" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("StreamingTV",data=churn, hue='Churn')
```

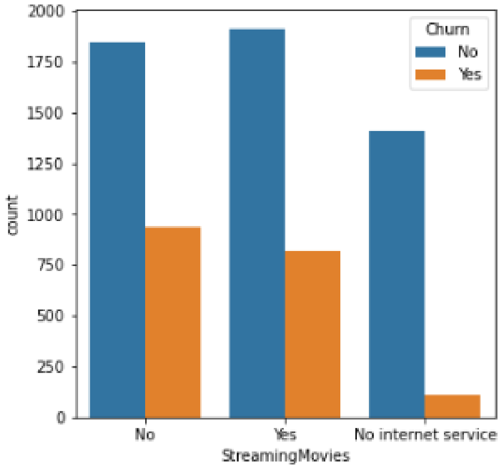
```
<AxesSubplot:xlabel='StreamingTV', ylabel='count'>
```



StreamingTV having No InternetService are having less churn and not having StreamingTV are having more churn.

```python
#Count Plot for comparision between "StreamingMovies" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("StreamingMovies",data=churn, hue='Churn')
```

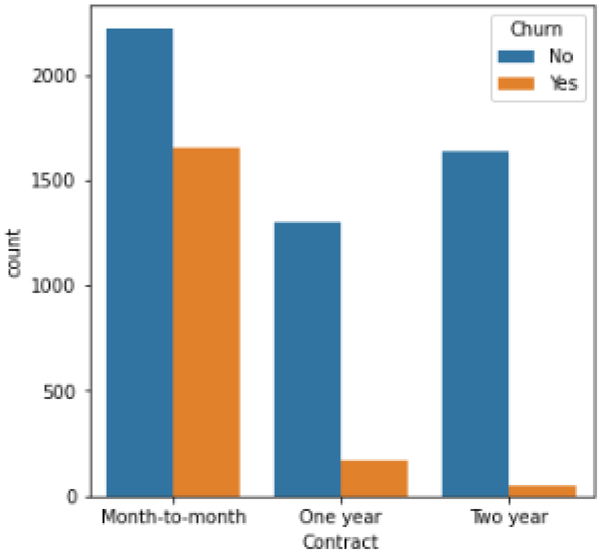
```
<AxesSubplot:xlabel='StreamingMovies', ylabel='count'>
```



StreamingMovies having No InternetService are having less churn and not having StreamingMovies are having more churn.

```python
#Count Plot for comparision between "Contract" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("Contract",data=churn, hue='Churn')
```

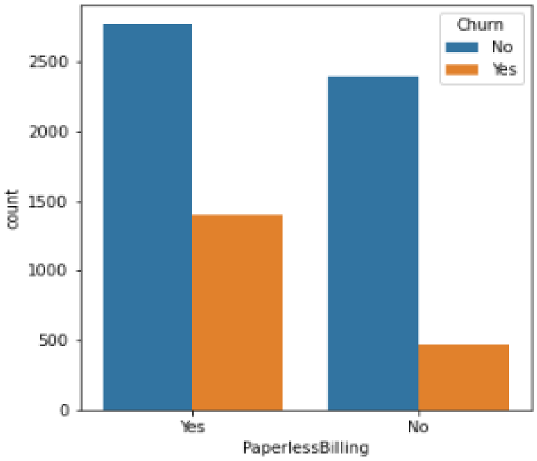
```
<AxesSubplot:xlabel='Contract', ylabel='count'>
```



Contract having Month-tomonth are having more churn and for Two year are having less churn.

```
#Count Plot for comparision between "PaperlessBilling" column and "Churn" column
plt.figure(figsize=(5,5))
sns.countplot("PaperlessBilling",data=churn, hue='Churn')
```

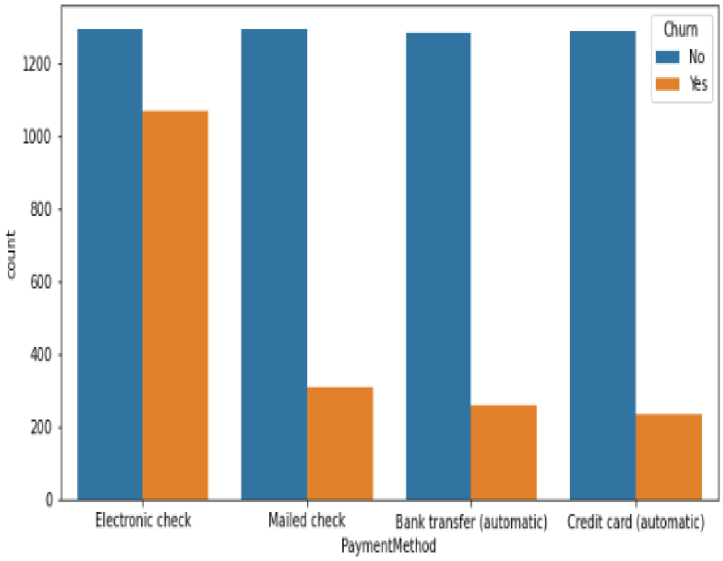
```
<AxesSubplot:xlabel='PaperlessBilling', ylabel='count'>
```



Having PaperlessBilling are having more churn and not having PaperlessBilling are having less churn.

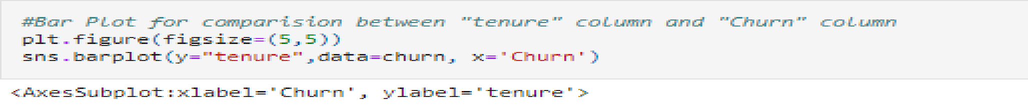
```
#Count Plot for comparision between "PaymentMethod" column and "Churn" column
plt.figure(figsize=(10,5))
sns.countplot("PaymentMethod",data=churn, hue='Churn')
```

```
<AxesSubplot:xlabel='PaymentMethod', ylabel='count'>
```



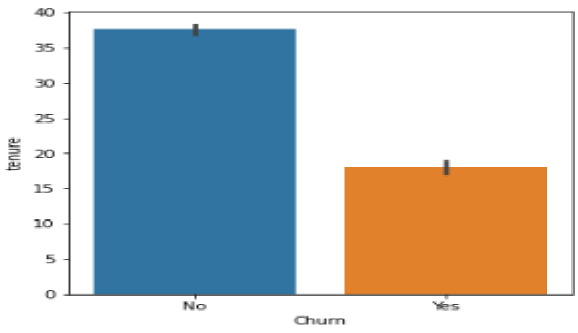
PaymentMethod used Electronic check having more churn compare to other PaymentMethod. And Credit card PaymentMethod have less churn compare to other PaymentMethod.

- Using Barplot (for comparison between continuous column and target)

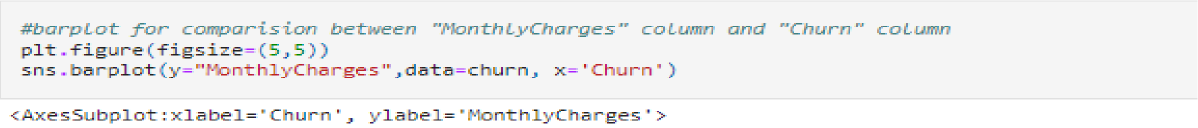


```
#Bar Plot for comparision between "tenure" column and "Churn" column
plt.figure(figsize=(5,5))
sns.barplot(y="tenure",data=churn, x='Churn')
```

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

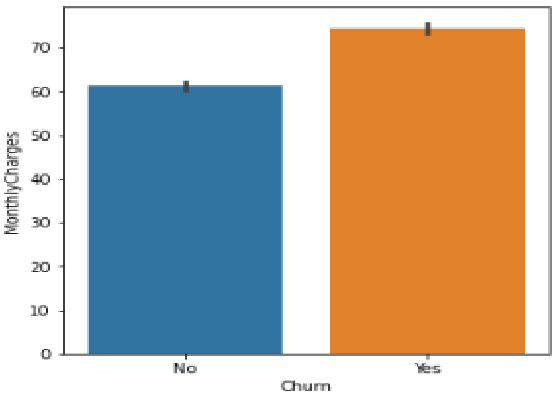


We can see less churn with high tenure. If tenure is less then churn rate is also High.

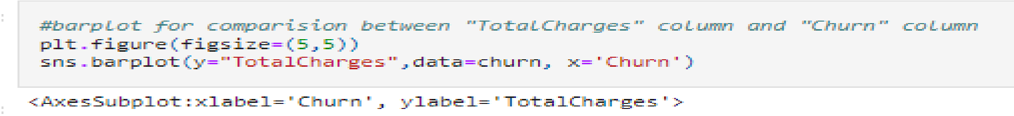


```
#barplot for comparision between "MonthlyCharges" column and "Churn" column
plt.figure(figsize=(5,5))
sns.barplot(y="MonthlyCharges",data=churn, x='Churn')
```

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

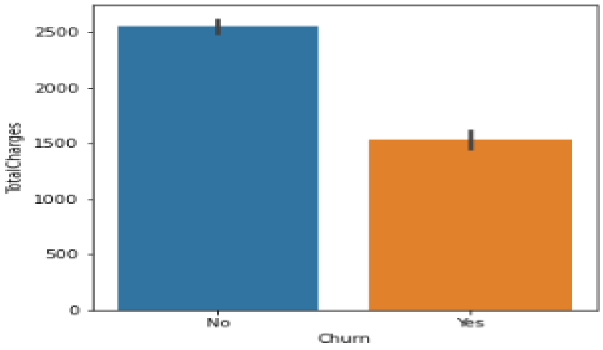


We can see High churn with High MonthlyCharges. Less Churn with less MonthlyCharges.



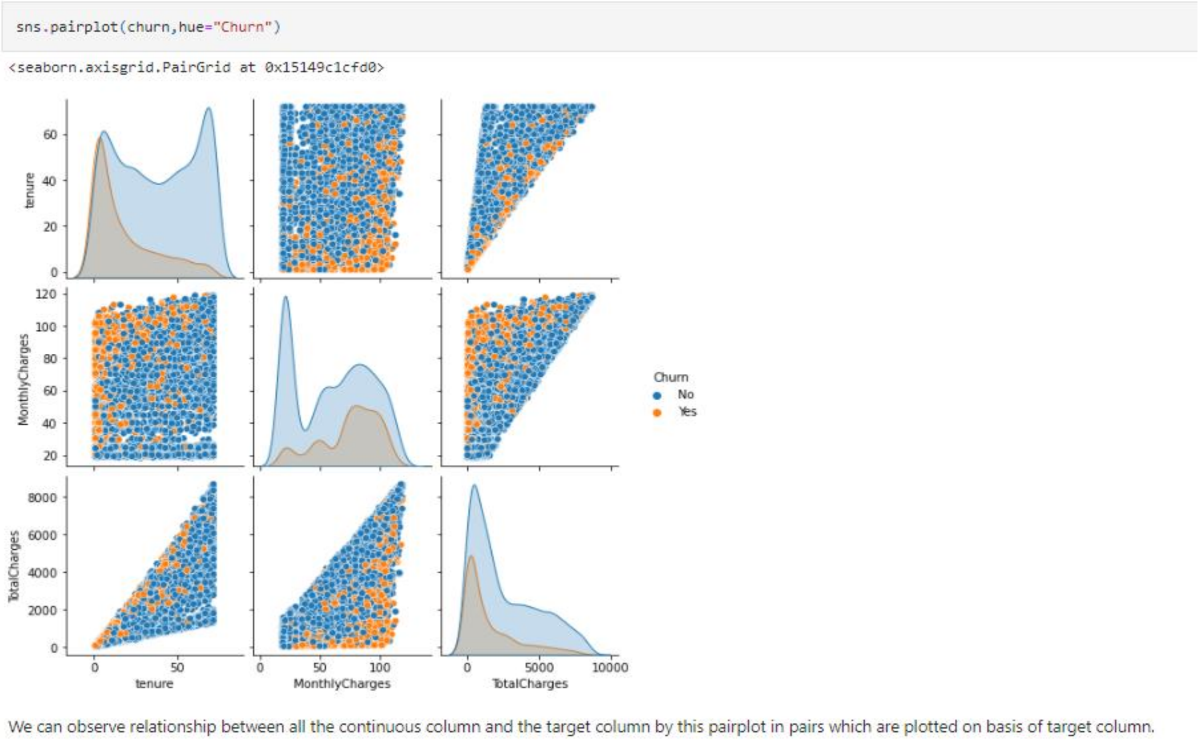
```
#barplot for comparision between "TotalCharges" column and "Churn" column
plt.figure(figsize=(5,5))
sns.barplot(y="TotalCharges",data=churn, x='Churn')
```

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



Less TotalCharges, high churn rate. High TotalCharges less churn rate.

- **Multivariant Analysis:** Using Pairplot
  (Relationship between continuous and target column)



```
sns.pairplot(churn,hue="Churn")
```

```
<seaborn.axisgrid.PairGrid at 0x15149c1cfd0>
```

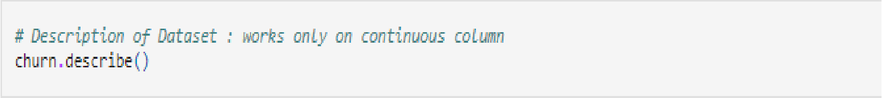
We can observe relationship between all the continuous column and the target column by this pairplot in pairs which are plotted on basis of target column.

Observation of Data Visualisation

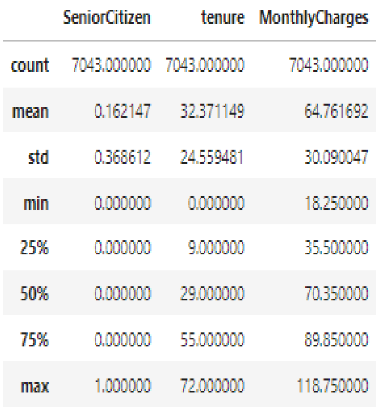
- The churn rate of senior citizens is approx. double of young citizens.
- We cannot determine churn rate through Gender as both male and female are equal.
- Customers with partner is having less churn compare to customers with no partner.
- Customers with month-to-month contracts have higher churn rates compared to Customers having yearly contracts.
- Customers who opted for an electronic check as paying method are having more churn rate.
- Customers subscribed to paperless billing churn more
- The churn rate is high when monthly charges are high.
- New customers churn more.
- Clients with high total charges having less churn.
- We do not expect phone attributes (PhoneService and MultipleLines) to have significant predictive power. The percentage of churn for all classes in both independent variables is nearly the same.
- Clients with online security churn less.
- Customers with no tech support tend to churn more often than those with tech support.

3. EDA Concluding Remark

Descriptive Statistics



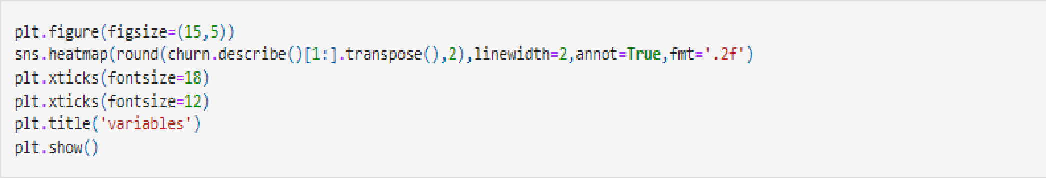
```
# Description of Dataset : works only on continuous column
churn.describe()
```



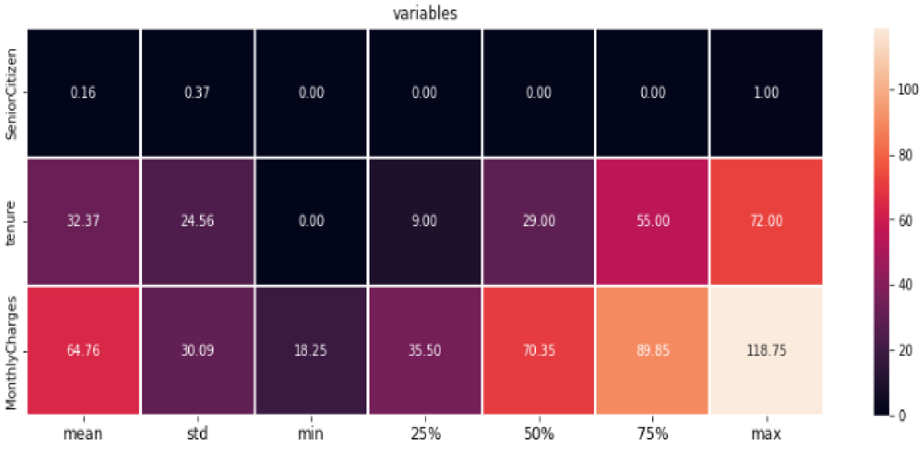
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559461	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

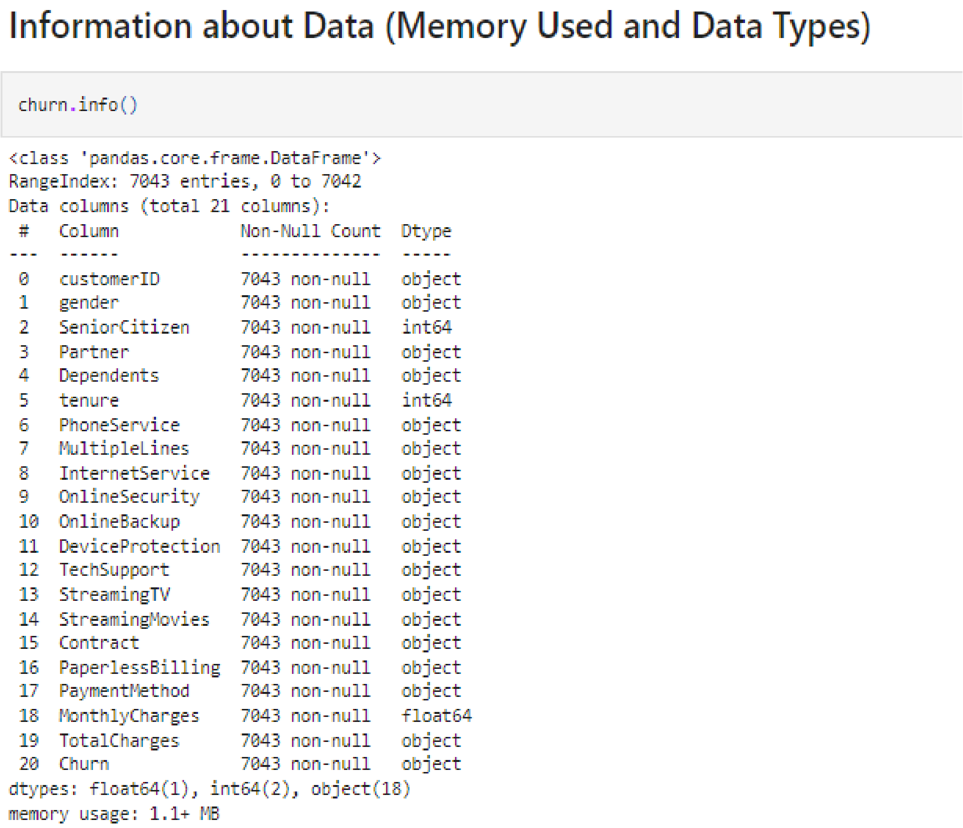
- So, we can see that 3 column is containing continuous data and rest 18 column contains categorical data.

Checking Description through heatmap also.



```
plt.figure(figsize=(15,5))
sns.heatmap(round(churn.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt='.2f')
plt.xticks(fontsize=18)
plt.xticks(fontsize=12)
plt.title('variables')
plt.show()
```





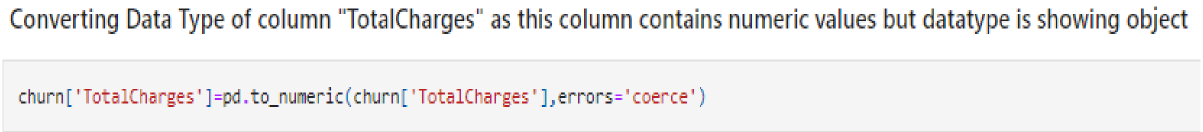
Information about Data (Memory Used and Data Types)

```
churn.info()
```

```
<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
```

There are 18 categorical features including Target Feature and 3 continuous features in Dataset.

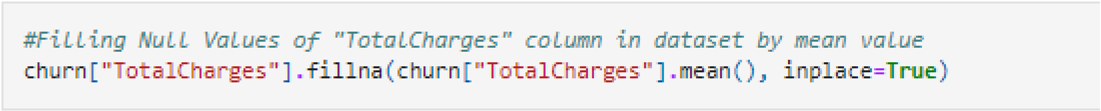
We also observed through visualization that column TotalCharges is wrongly detected as **object data type.** This column represents the total amount charged to the customer and it should be a numeric variable. So, for further analysis, we need to transform this column into a **numeric data type** through function pd.to_numeric.



Converting Data Type of column "TotalCharges" as this column contains numeric values but datatype is showing object

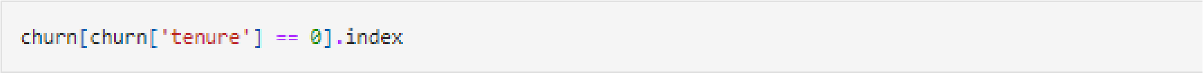
```
churn['TotalCharges']=pd.to_numeric(churn['TotalCharges'],errors='coerce')
```

After converting Datatype of column TotalCharges, we can see there are 11 null values present in column TotalCharges. So handled null values through mean mode.



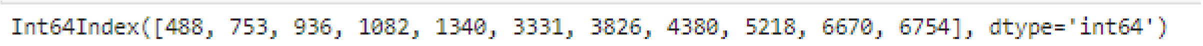
```
#Filling Null Values of "TotalCharges" column in dataset by mean value
churn["TotalCharges"].fillna(churn["TotalCharges"].mean(), inplace=True)
```

And also observed that column tenure is having 0 values. So have to handle this too. So, first check for index number where 0 values are present.



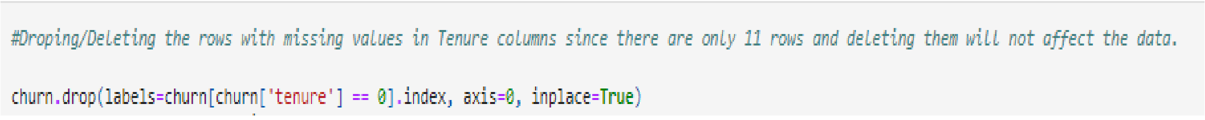
```
churn[churn['tenure'] == 0].index
```

We get these index numbers:

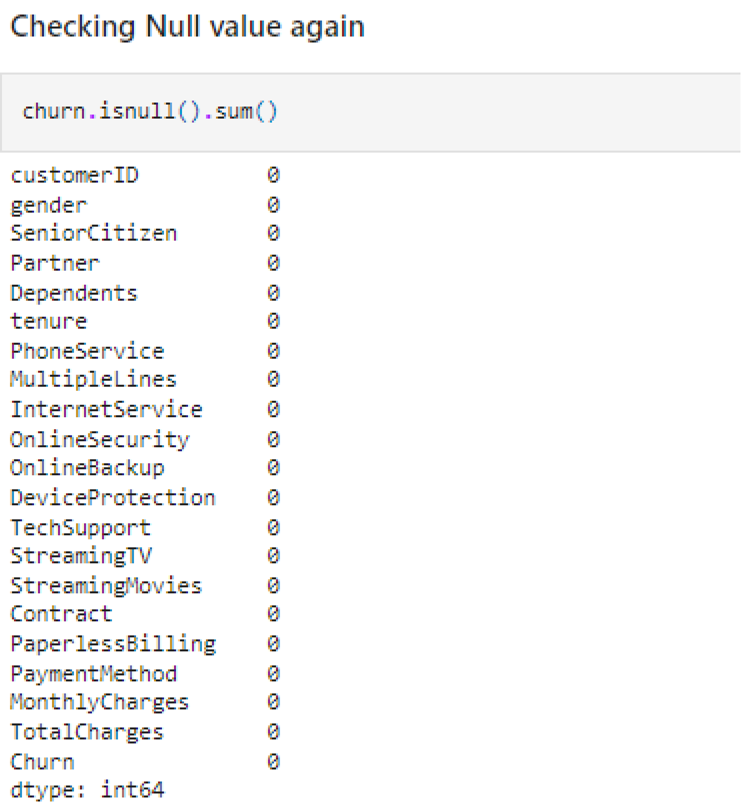


```
Int64Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

Therefore, to handle this we are dropping these rows as it is only 11 and it will not affect prediction.



```
#Droping/Deleting the rows with missing values in Tenure columns since there are only 11 rows and deleting them will not affect the data.

churn.drop(labels=churn[churn['tenure'] == 0].index, axis=0, inplace=True)
```



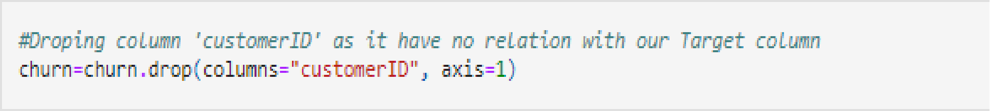
Checking Null value again

```
churn.isnull().sum()
```

```
customerID          0
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
```

4. Pre-Processing Pipeline

Column customerID is having unique value for each data so it has no relation with Target. Hence, will drop this column.

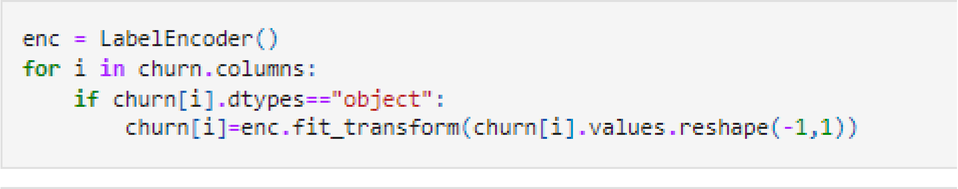


```
#Droping column 'customerID' as it have no relation with our Target column
churn=churn.drop(columns="customerID", axis=1)
```

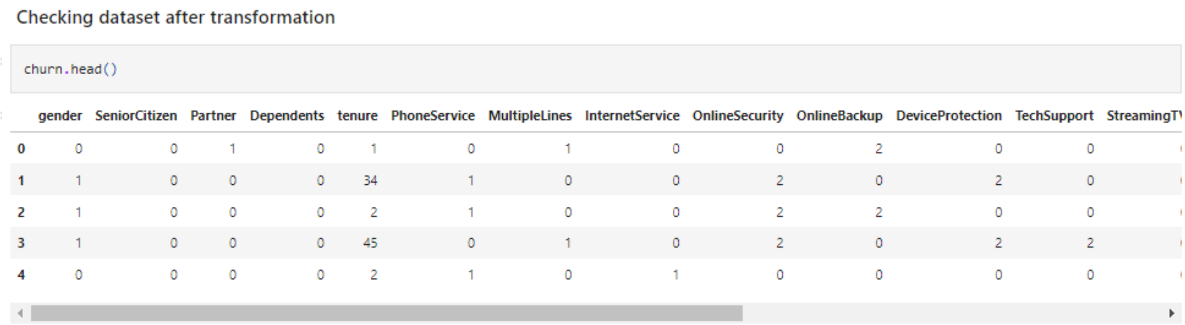
Label Encoding

Label Encoding refers to converting the labels into a numeric form to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

So, we are encoding the data by transforming from categorical to continuous.



```
enc = LabelEncoder()
for i in churn.columns:
    if churn[i].dtypes=="object":
        churn[i]=enc.fit_transform(churn[i].values.reshape(-1,1))
```



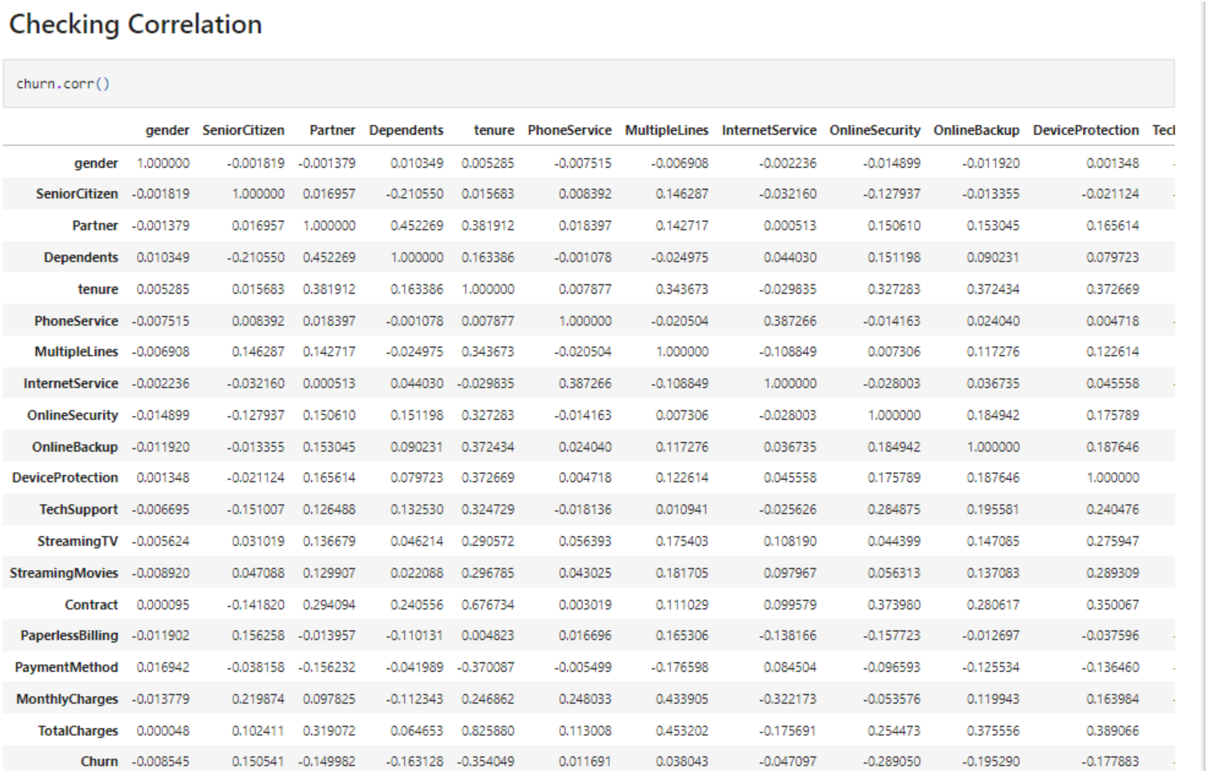
Checking dataset after transformation

```
churn.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingT
0	0	0	1	0	1	0	1	0	0	2	0	0	
1	1	0	0	0	34	1	0	0	2	0	2	0	
2	1	0	0	0	2	1	0	0	2	2	0	0	
3	1	0	0	0	45	0	1	0	2	0	2	2	
4	0	0	0	0	2	1	0	1	0	0	0	0	

Checking Correlation

Data correlation is the way in which one set of data may correspond to another set means measures the linear relationship between two variables. It describes how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable.

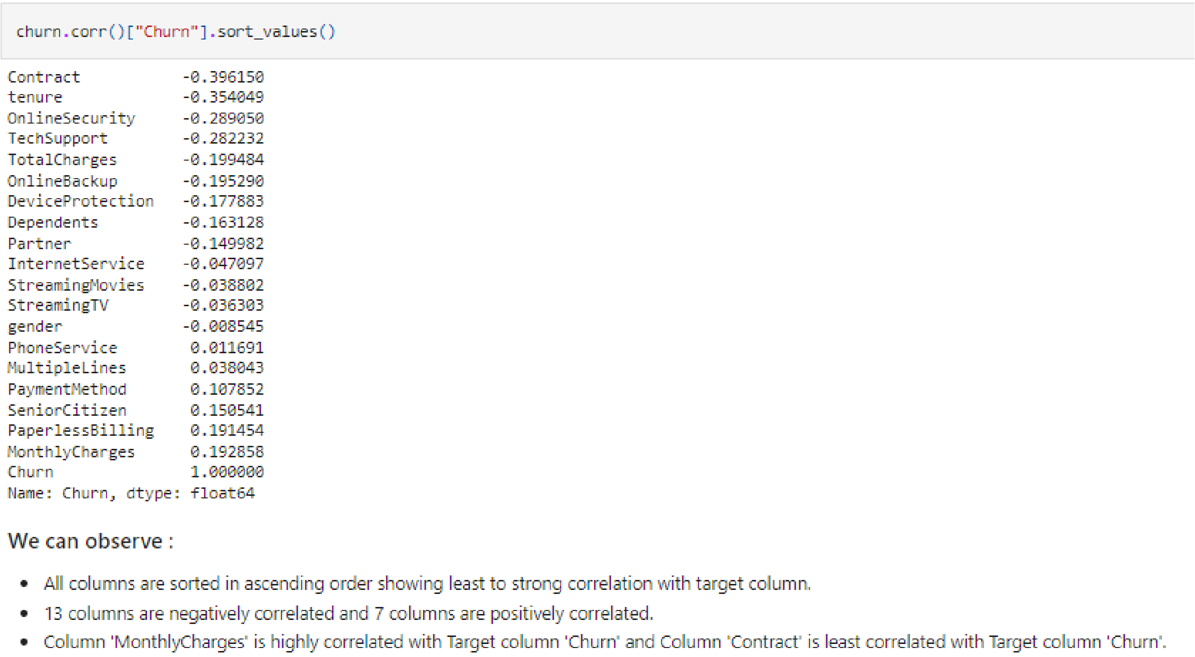


Checking Correlation

```
churn.corr()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	Tecl
gender	1.000000	-0.001819	-0.001379	0.010349	0.005285	-0.007515	-0.006908	-0.002236	-0.014899	-0.011920	0.001348	-
SeniorCitizen	-0.001819	1.000000	0.016957	-0.210550	0.015683	0.008392	0.146287	-0.032160	-0.127937	-0.013355	-0.021124	-
Partner	-0.001379	0.016957	1.000000	0.452269	0.381912	0.018397	0.142717	0.000513	0.150610	0.153045	0.165614	
Dependents	0.010349	-0.210550	0.452269	1.000000	0.163386	-0.001078	-0.024975	0.044030	0.151198	0.090231	0.079723	
tenure	0.005285	0.015683	0.381912	0.163386	1.000000	0.007877	0.343673	-0.029835	0.327283	0.372434	0.372669	
PhoneService	-0.007515	0.008392	0.018397	-0.001078	0.007877	1.000000	-0.020504	0.387266	-0.014163	0.024040	0.004718	-
MultipleLines	-0.006908	0.146287	0.142717	-0.024975	0.343673	-0.020504	1.000000	-0.108849	0.007306	0.117276	0.122614	
InternetService	-0.002236	-0.032160	0.000513	0.044030	-0.029835	0.387266	-0.108849	1.000000	-0.028003	0.036735	0.045558	-
OnlineSecurity	-0.014899	-0.127937	0.150610	0.151198	0.327283	-0.014163	0.007306	-0.028003	1.000000	0.184942	0.175789	
OnlineBackup	-0.011920	-0.013355	0.153045	0.090231	0.372434	0.024040	0.117276	0.036735	0.184942	1.000000	0.187646	
DeviceProtection	0.001348	-0.021124	0.165614	0.079723	0.372669	0.004718	0.122614	0.045558	0.175789	0.187646	1.000000	
TechSupport	-0.006695	-0.151007	0.126488	0.132530	0.324729	-0.018136	0.010941	-0.025626	0.284875	0.195581	0.240476	
StreamingTV	-0.005624	0.031019	0.136679	0.046214	0.290572	0.056393	0.175403	0.108190	0.044399	0.147085	0.275947	
StreamingMovies	-0.008920	0.047088	0.129907	0.022088	0.296785	0.043025	0.181705	0.097967	0.056313	0.137083	0.289309	
Contract	0.000095	-0.141820	0.294094	0.240556	0.676734	0.003019	0.111029	0.099579	0.373980	0.280617	0.350067	
PaperlessBilling	-0.011902	0.156258	-0.013957	-0.110131	0.004823	0.016696	0.165306	-0.138166	-0.157723	-0.012697	-0.037596	-
PaymentMethod	0.016942	-0.038158	-0.156232	-0.041989	-0.370087	-0.005499	-0.176598	0.084504	-0.096593	-0.125534	-0.136460	-
MonthlyCharges	-0.013779	0.219874	0.097825	-0.112343	0.246862	0.248033	0.433905	-0.322173	-0.053576	0.119943	0.163984	-
TotalCharges	0.000048	0.102411	0.319072	0.064653	0.825880	0.113008	0.453202	-0.175691	0.254473	0.375556	0.389066	
Churn	-0.008545	0.150541	-0.149982	-0.163128	-0.354049	0.011691	0.038043	-0.047097	-0.289050	-0.195290	-0.177883	-

- Checking correlation in ascending order also to get highest and lowest correlated values.



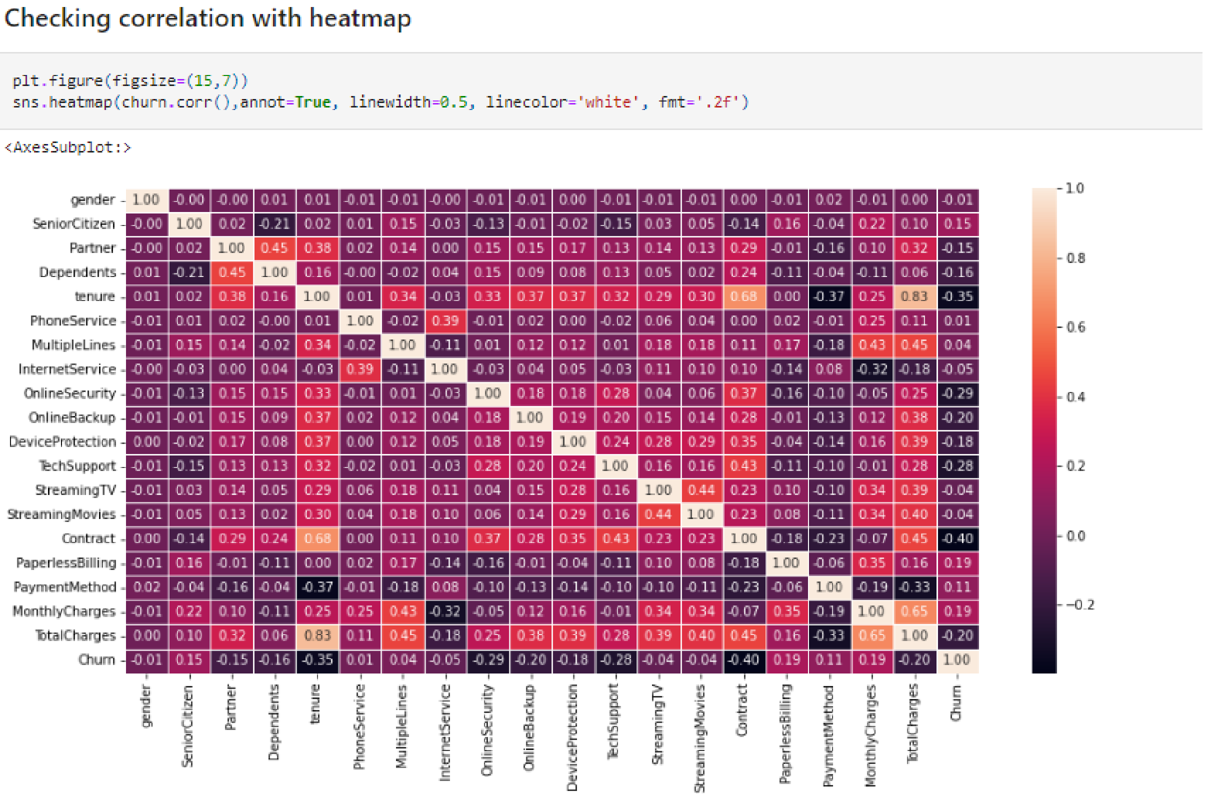
```
churn.corr()["Churn"].sort_values()
```

```
Contract           -0.396150
tenure             -0.354049
OnlineSecurity     -0.289050
TechSupport        -0.282232
TotalCharges       -0.199484
OnlineBackup       -0.195290
DeviceProtection   -0.177883
Dependents         -0.163128
Partner            -0.149982
InternetService    -0.047097
StreamingMovies    -0.038802
StreamingTV        -0.036303
gender             -0.008545
PhoneService        0.011691
MultipleLines       0.038043
PaymentMethod       0.107852
SeniorCitizen       0.150541
PaperlessBilling    0.191454
MonthlyCharges      0.192858
Churn               1.000000
Name: Churn, dtype: float64
```

We can observe :

- All columns are sorted in ascending order showing least to strong correlation with target column.
- 13 columns are negatively correlated and 7 columns are positively correlated.
- Column 'MonthlyCharges' is highly correlated with Target column 'Churn' and Column 'Contract' is least correlated with Target column 'Churn'.

- Observing correlation through data visualization also:

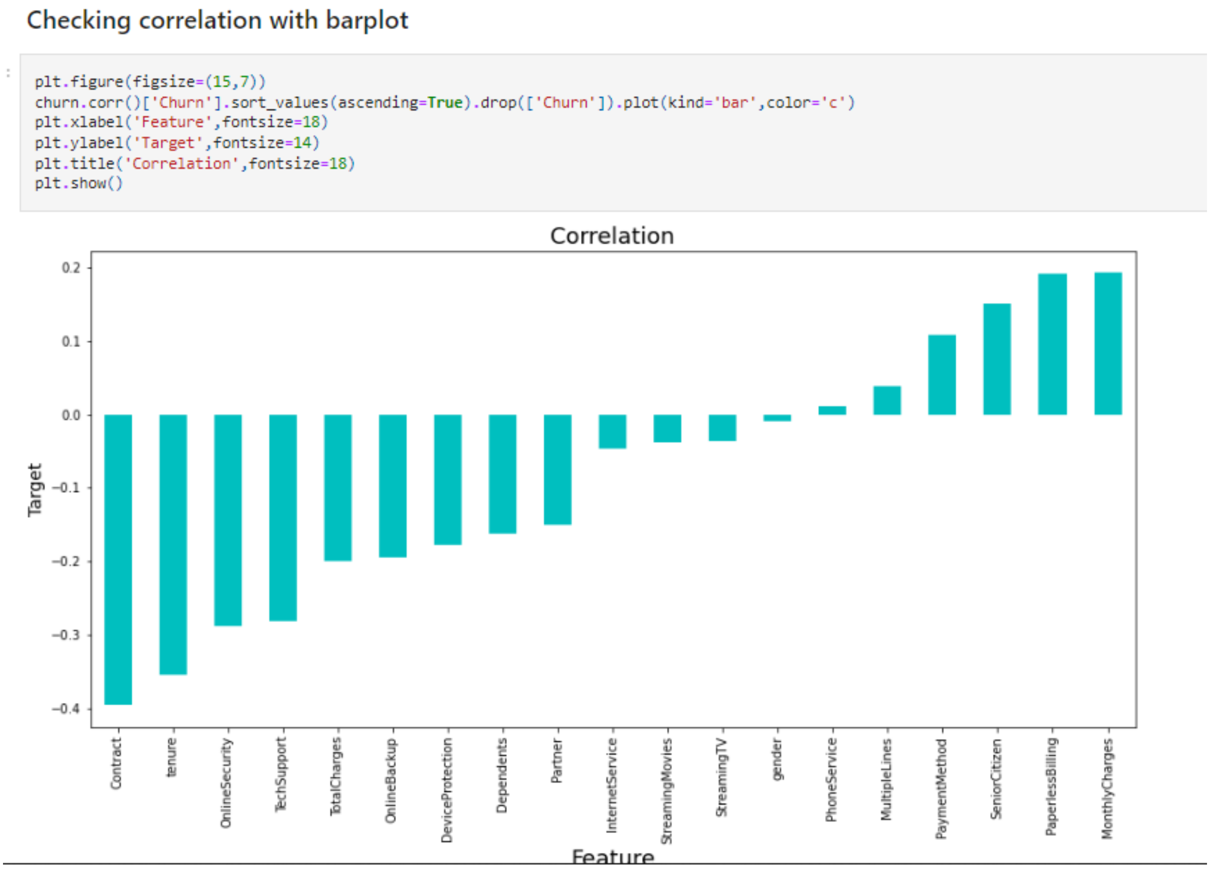


Checking correlation with heatmap

```
plt.figure(figsize=(15,7))
sns.heatmap(churn.corr(),annot=True, linewidth=0.5, linecolor='white', fmt='.2f')
```

```
<AxesSubplot:>
```

gender 1.00 -0.00 -0.00 0.01 0.01 -0.01 -0.01 -0.00 -0.01 -0.01 0.00 -0.01 -0.01 -0.01 0.00 -0.01 0.02 -0.01 0.00 -0.01
SeniorCitizen -0.00 1.00 0.02 -0.21 0.02 0.01 0.15 -0.03 -0.13 -0.01 -0.02 -0.15 0.03 0.05 -0.14 0.16 -0.04 0.22 0.10 0.15
Partner -0.00 0.02 1.00 0.45 0.38 0.02 0.14 0.00 0.15 0.15 0.17 0.13 0.14 0.13 0.29 -0.01 -0.16 0.10 0.32 -0.15
Dependents 0.01 -0.21 0.45 1.00 0.16 -0.00 -0.02 0.04 0.15 0.09 0.08 0.13 0.05 0.02 0.24 -0.11 -0.04 -0.11 0.06 -0.16
tenure 0.01 0.02 0.38 0.16 1.00 0.01 0.34 -0.03 0.33 0.37 0.37 0.32 0.29 0.30 0.68 0.00 -0.37 0.25 0.83 -0.35
PhoneService -0.01 0.01 0.02 -0.00 0.01 1.00 -0.02 0.39 -0.01 0.02 0.00 -0.02 0.06 0.04 0.00 0.02 -0.01 0.25 0.11 0.01
MultipleLines -0.01 0.15 0.14 -0.02 0.34 -0.02 1.00 -0.11 0.01 0.12 0.12 0.01 0.18 0.18 0.11 0.17 -0.18 0.43 0.45 0.04
InternetService -0.00 -0.03 0.00 0.04 -0.03 0.39 -0.11 1.00 -0.03 0.04 0.05 -0.03 0.11 0.10 0.10 -0.14 0.08 -0.32 -0.18 -0.05
OnlineSecurity -0.01 -0.13 0.15 0.15 0.33 -0.01 0.01 -0.03 1.00 0.18 0.18 0.28 0.04 0.06 0.37 -0.16 -0.10 -0.05 0.25 -0.29
OnlineBackup -0.01 -0.01 0.15 0.09 0.37 0.02 0.12 0.04 0.18 1.00 0.19 0.20 0.15 0.14 0.28 -0.01 -0.13 0.12 0.38 -0.20
DeviceProtection 0.00 -0.02 0.17 0.08 0.37 0.00 0.12 0.05 0.18 0.19 1.00 0.24 0.28 0.29 0.35 -0.04 -0.14 0.16 0.39 -0.18
TechSupport -0.01 -0.15 0.13 0.13 0.32 -0.02 0.01 -0.03 0.28 0.20 0.24 1.00 0.16 0.16 0.43 -0.11 -0.10 -0.01 0.28 -0.28
StreamingTV -0.01 0.03 0.14 0.05 0.29 0.06 0.18 0.11 0.04 0.15 0.28 0.16 1.00 0.44 0.23 0.10 -0.10 0.34 0.39 -0.04
StreamingMovies -0.01 0.05 0.13 0.02 0.30 0.04 0.18 0.10 0.06 0.14 0.29 0.16 0.44 1.00 0.23 0.08 -0.11 0.34 0.40 -0.04
Contract 0.00 -0.14 0.29 0.24 0.68 0.00 0.11 0.10 0.37 0.28 0.35 0.43 0.23 0.23 1.00 -0.18 -0.23 -0.07 0.45 -0.40
PaperlessBilling -0.01 0.16 -0.01 -0.11 0.00 0.02 0.17 -0.14 -0.16 -0.01 -0.04 -0.11 0.10 0.08 -0.18 1.00 -0.06 0.35 0.16 0.19
PaymentMethod 0.02 -0.04 -0.16 -0.04 -0.37 -0.01 -0.18 0.08 -0.10 -0.13 -0.14 -0.10 -0.10 -0.11 -0.23 -0.06 1.00 -0.19 -0.33 0.11
MonthlyCharges -0.01 0.22 0.10 -0.11 0.25 0.25 0.43 -0.32 -0.05 0.12 0.16 -0.01 0.34 0.34 -0.07 0.35 -0.19 1.00 0.65 0.19
TotalCharges 0.00 0.10 0.32 0.06 0.83 0.11 0.45 -0.18 0.25 0.38 0.39 0.28 0.39 0.40 0.45 0.16 -0.33 0.65 1.00 -0.20
Churn -0.01 0.15 -0.15 -0.16 -0.35 0.01 0.04 -0.05 -0.29 -0.20 -0.18 -0.28 -0.04 -0.04 -0.40 0.19 0.11 0.19 -0.20 1.00
gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract
PaperlessBilling
PaymentMethod
MonthlyCharges
TotalCharges
Churn
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



Checking correlation with barplot

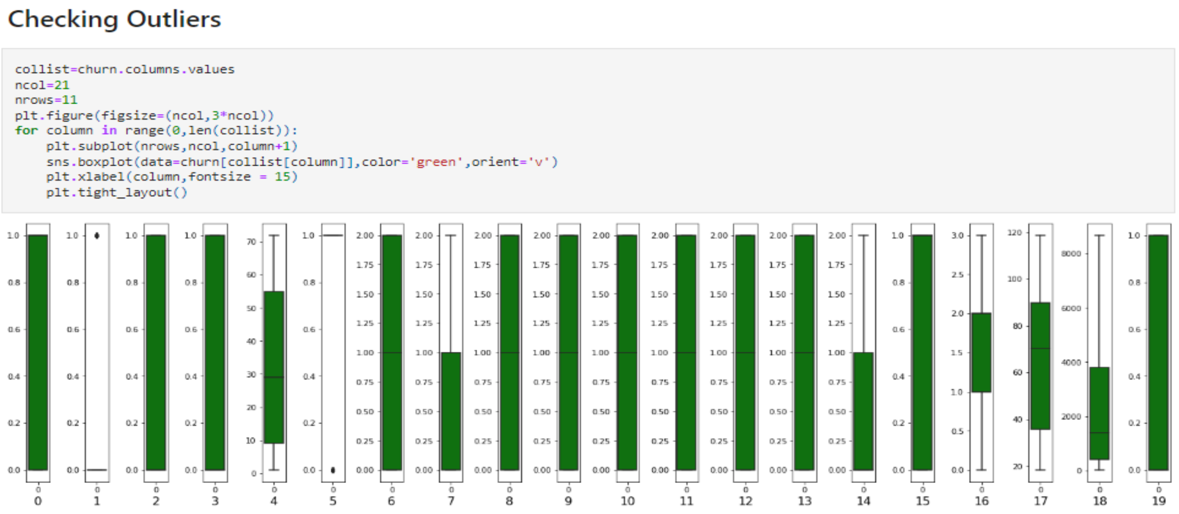
```
plt.figure(figsize=(15,7))
churn.corr()['Churn'].sort_values(ascending=True).drop(['Churn']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```

- **<u>Observation of Correlation:</u>**
    - gender has -1 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - SeniorCitizen has 15 percent correlation with the target column which can be considered as good correlation and positively correlated.
    - Partner has -15 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - Dependents has -16 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - tenure has -35 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - PhoneService has 1 percent correlation with the target column which can be considered as good correlation and positively correlated.
    - MultipleLines has 4 percent correlation with the target column which can be considered as good correlation and positively correlated.
    - InternetService has -5 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - OnlineSecurity has -29 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - OnlineBackup has -20 percent correlation with the target column which can be considered as good correlation and negatively correlated.
    - DeviceProtection has -18 percent correlation with the target column which can be considered as good correlation and negatively correlated.

  - TechSupport has -28 percent correlation with the target column which can be considered as good correlation and negatively correlated.
  - StreamingTV has -4 percent correlation with the target column which can be considered as good correlation and negatively correlated.
  - StreamingMovies has -4 percent correlation with the target column which can be considered as good correlation and negatively correlated.
  - Contract has -40 percent correlation with the target column which can be considered as weak correlation and negatively correlated.
  - PaperlessBilling has 19 percent correlation with the target column which can be considered as good correlation and positively correlated.
  - PaymentMethod has 11 percent correlation with the target column which can be considered as good correlation and positively correlated.
  - MonthlyCharges has 19 percent correlation with the target column which can be considered as strong correlation and positively correlated.
  - TotalCharges has -20 percent correlation with the target column which can be considered as good correlation and negatively correlated.
- **Max correlation** is with MonthlyCharges
- **Min correlation** is with Contract

➢ Finding Outliers:

An outlier is a data point that is noticeably different from the rest. They represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data.



Checking Outliers

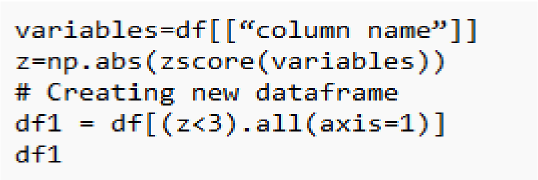
```
collist=churn.columns.values
ncol=21
nrows=11
plt.figure(figsize=(ncol,3*ncol))
for column in range(0,len(collist)):
    plt.subplot(nrows,ncol,column+1)
    sns.boxplot(data=churn[collist[column]],color='green',orient='v')
    plt.xlabel(column,fontsize = 15)
    plt.tight_layout()
```

Observation:

We can see Outliers are present only in 2 columns: "SeniorCitizen" and "PhoneService". But both column are categorical, so we will not remove outliers.

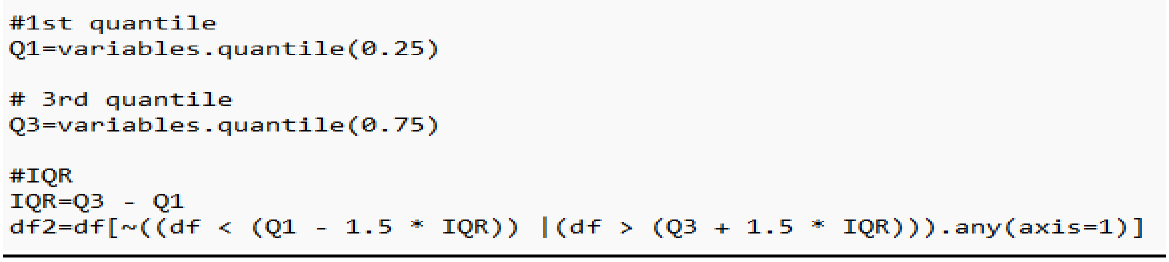
If Outliers were present then we have to remove through Zscore method using Scipy or IQR (Inter Quantile Range) method. Every time we should check both method for removal and then compare between both which method is giving less data loss then use that method for further process.

Formula for Zscore method using Scipy:



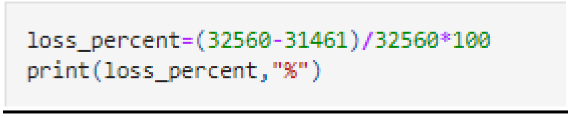
```
variables=df[[“column name”]]
z=np.abs(zscore(variables))
# Creating new dataframe
df1 = df[(z<3).all(axis=1)]
df1
```

Formula for IQR (Inter Quantile Range) method:



```
#1st quantile
Q1=variables.quantile(0.25)

# 3rd quantile
Q3=variables.quantile(0.75)

#IQR
IQR=Q3 - Q1
df2=df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Formula for checking Loss Percent:



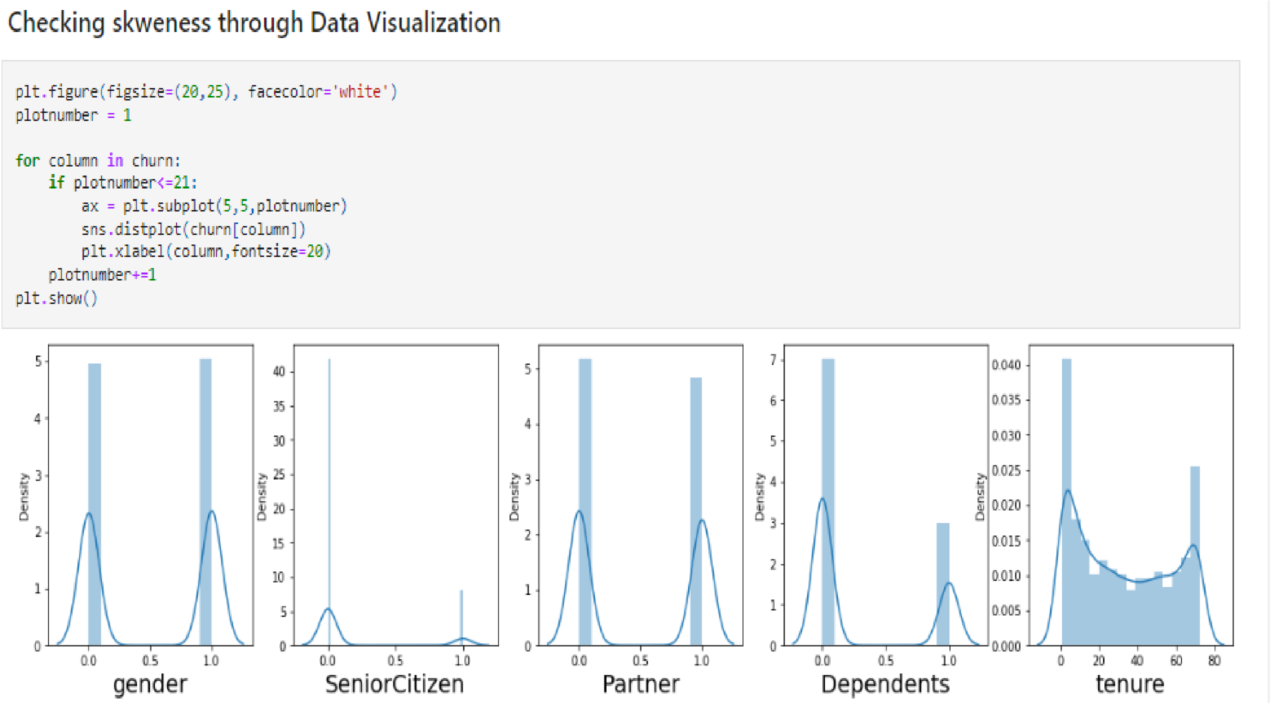
```
loss_percent=(32560-31461)/32560*100
print(loss_percent,"%")
```

Finding Skewness :

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution.

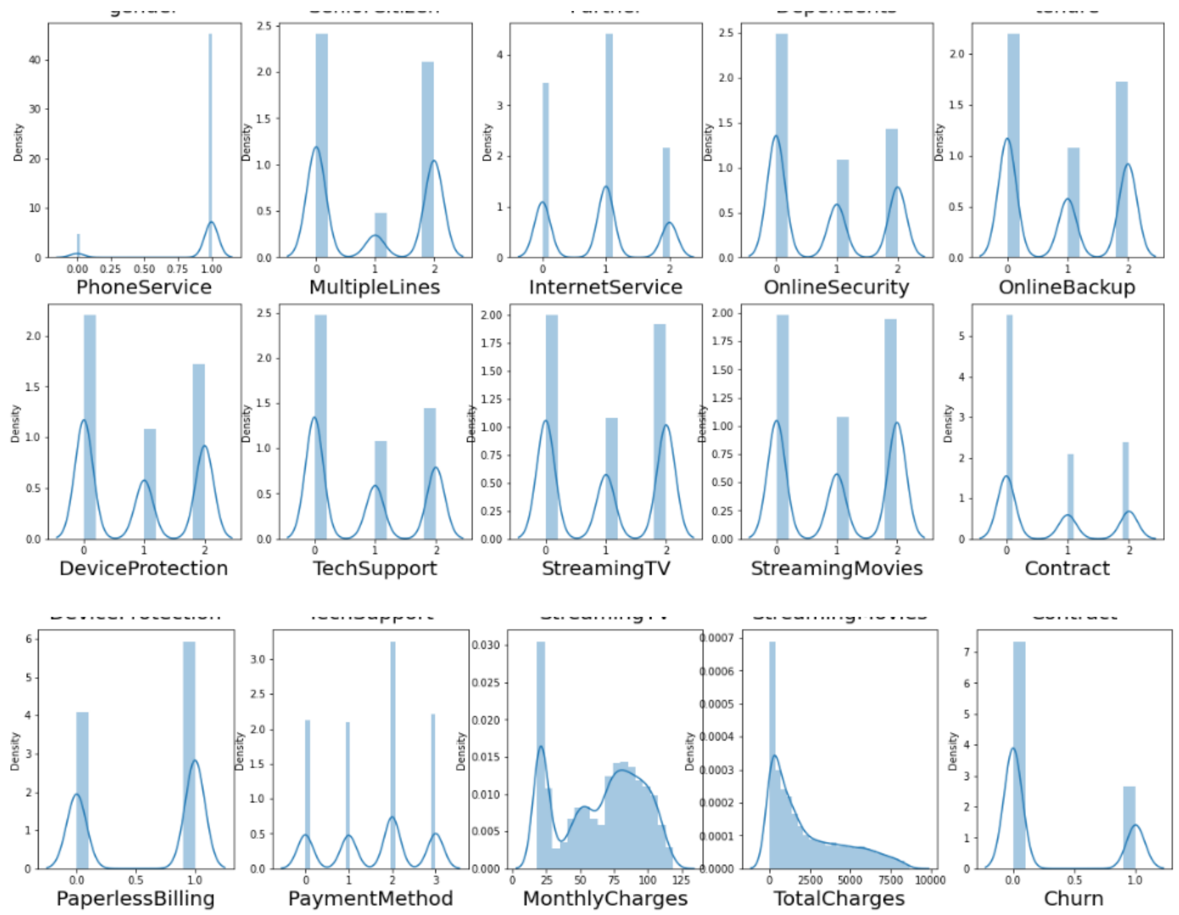
- **The three types of skewness are:**
    - Right skew (also called positive skew). A right-skewed distribution is longer on the right side of its peak than on its left.
    - Left skew (also called negative skew). A left-skewed distribution is longer on the left side of its peak than on its right.
    - Zero skew.

- **Skewness checked through Data Visualization also:**



Checking skweness through Data Visualization

```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in churn:
    if plotnumber<=21:
        ax = plt.subplot(5,5,plotnumber)
        sns.distplot(churn[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



We can see skewed data

Observation:

- Skewness threshold taken is +/-0.25
- Columns which are having skewness: SeniorCitizen, Dependents, PhoneService, OnlineSecurity, TechSupport, Contract, PaperlessBilling and TotalCharges.

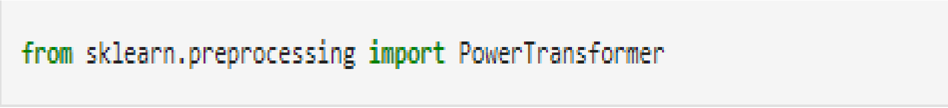
➢ Removal of Skewness

For removing skewness Power Transformer method is used. Power transforms are a technique for transforming numerical input or output variables to have a uniform or a Gaussian probability distribution. A power transform will make the probability distribution of a variable more Gaussian.

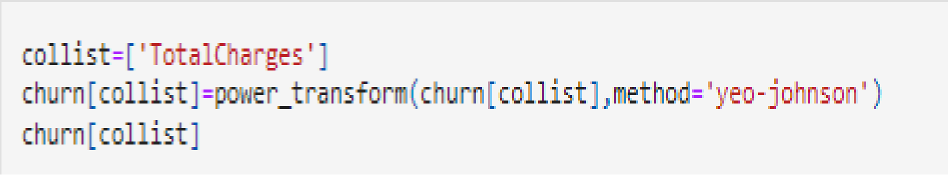
Currently, Power Transformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive and negative data.

So we are using Yeo-Johnson method for removing skewness. Since SeniorCitizen, Dependents, PhoneService, OnlineSecurity, TechSupport, Contract and PaperlessBilling are categorical column so we will not remove skewness from them. Only we will remove skewness from TotalCharges as this column contains continuous data.

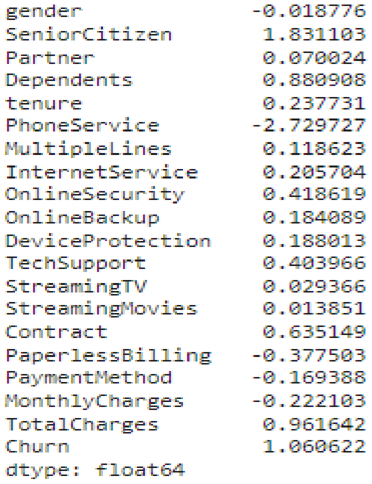
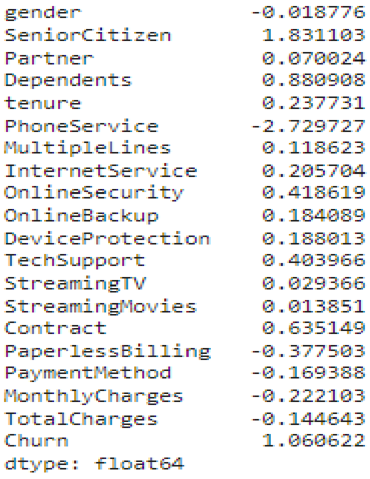


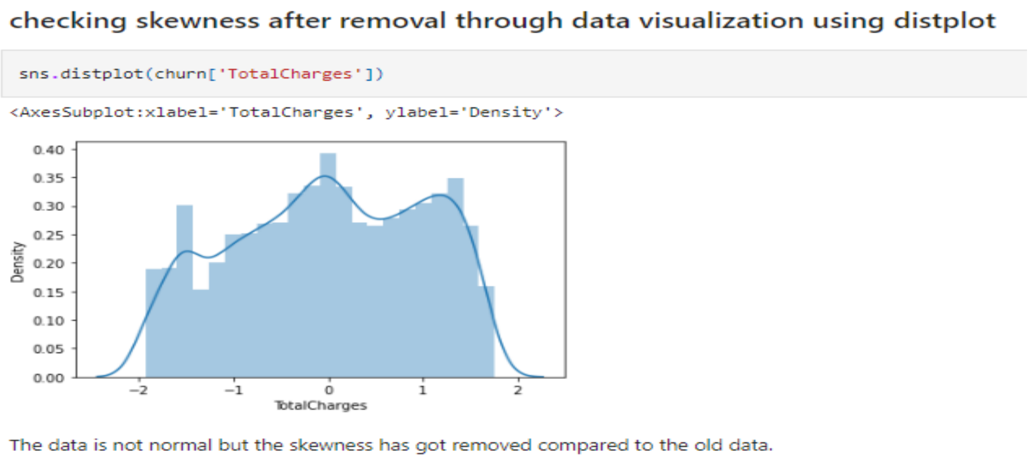
```
from sklearn.preprocessing import PowerTransformer
```



```
collist=['TotalCharges']
churn[collist]=power_transform(churn[collist],method='yeo-johnson')
churn[collist]
```

Comparison between skewed and not skewed data

Data before skewness removal	Data after skewness removal
 gender -0.018776 SeniorCitizen 1.831103 Partner 0.070024 Dependents 0.880908 tenure 0.237731 PhoneService -2.729727 MultipleLines 0.118623 InternetService 0.205704 OnlineSecurity 0.418619 OnlineBackup 0.184089 DeviceProtection 0.188013 TechSupport 0.403966 StreamingTV 0.029366 StreamingMovies 0.013851 Contract 0.635149 PaperlessBilling -0.377503 PaymentMethod -0.169388 MonthlyCharges -0.222103 TotalCharges 0.961642 Churn 1.060622 dtype: float64	 gender -0.018776 SeniorCitizen 1.831103 Partner 0.070024 Dependents 0.880908 tenure 0.237731 PhoneService -2.729727 MultipleLines 0.118623 InternetService 0.205704 OnlineSecurity 0.418619 OnlineBackup 0.184089 DeviceProtection 0.188013 TechSupport 0.403966 StreamingTV 0.029366 StreamingMovies 0.013851 Contract 0.635149 PaperlessBilling -0.377503 PaymentMethod -0.169388 MonthlyCharges -0.222103 TotalCharges -0.144643 Churn 1.060622 dtype: float64



checking skewness after removal through data visualization using distplot

```
sns.distplot(churn['TotalCharges'])
```

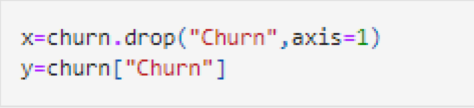
```
<AxesSubplot:xlabel='TotalCharges', ylabel='Density'>
```

The data is not normal but the skewness has got removed compared to the old data.

➢ Spliting data into Target and Features

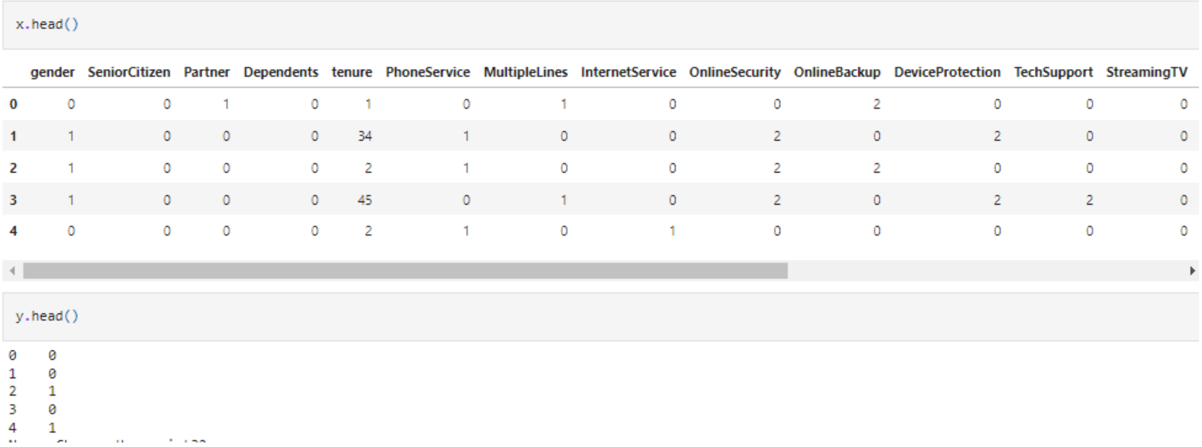
Split the data set into two set: a training set and a testing set. This consists of random sampling without replacement about 75 percent of the rows and putting them into our training set and the remaining 25 percent is put into our test set.

Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.



```
x=churn.drop("Churn",axis=1)
y=churn["Churn"]
```

Data after splitting:



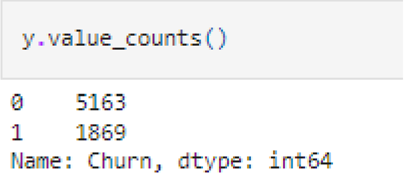
```
x.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV
0	0	0	1	0	1	0	1	0	0	2	0	0	0
1	1	0	0	0	34	1	0	0	2	0	2	0	0
2	1	0	0	0	2	1	0	0	2	2	0	0	0
3	1	0	0	0	45	0	1	0	2	0	2	2	0
4	0	0	0	0	2	1	0	1	0	0	0	0	0

```
y.head()
```

```
0    0
1    0
2    1
3    0
4    1
```

Total value count of y column:



```
y.value_counts()
```

```
0    5163
1    1869
Name: Churn, dtype: int64
```

Here it is observed that data is not balanced. So, we will use oversampling method to balance it.

➢ **Oversampling Method**

Oversampling methods duplicate examples in the minority class or synthesize new examples from the examples in the minority class. Some of the more widely used and implemented oversampling methods include:

- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Borderline-SMOTE
- Borderline Oversampling with SVM
- Adaptive Synthetic Sampling (ADASYN)

Random Oversampling :

The simplest oversampling method involves randomly duplicating examples from the minority class in the training dataset

SMOTE:

The most popular and most successful oversampling method is SMOTE; that is an acronym for Synthetic Minority Oversampling Technique.

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line.

So, to balance the data here using SMOTE method

Borderline-SMOTE:

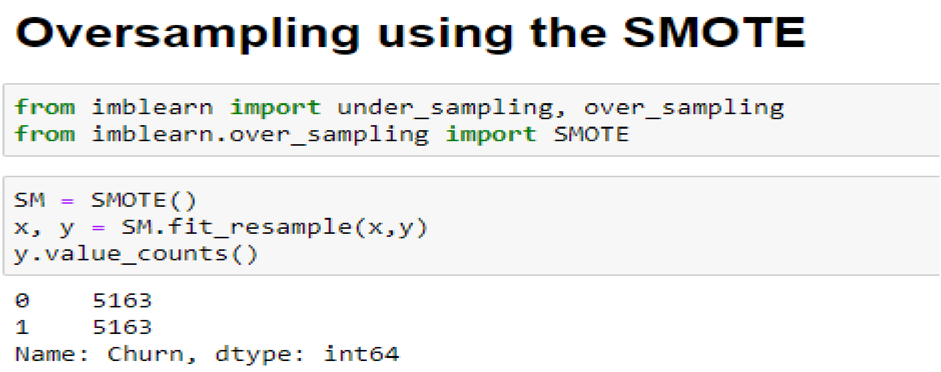
Selecting those instances of the minority class that are misclassified, such as with a k-nearest neighbor classification model, and only generating synthetic samples that are "difficult" to classify is Borderline-SMOTE.

Borderline Oversampling with SVM:

It is an extension to SMOTE that fits an SVM to the dataset and uses the decision boundary as defined by the support vectors as the basis for generating synthetic examples, again based on the idea that the decision boundary is the area where more minority examples are required.

Adaptive Synthetic Sampling (ADASYN):

(ADASYN) is another extension to SMOTE that generates synthetic samples inversely proportional to the density of the examples in the minority class. It is designed to create synthetic examples in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high.



Oversampling using the SMOTE

```
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
```

```
SM = SMOTE()
x, y = SM.fit_resample(x,y)
y.value_counts()
```

```
0     5163
1     5163
Name: Churn, dtype: int64
```

- **Scaling Data:**

It is a step of Data Pre-Processing that is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

Normalization and Standardization are the two main methods for the scaling of the data which are widely used in the algorithms where scaling is required. Both of them can be implemented by the scikit-learn libraries pre-process package.

Normalization is used when we want to bound our values between two numbers, typically, between [0,1] or [-1,1]. It is also known as Min-Max scaling.

Formula for normalization:

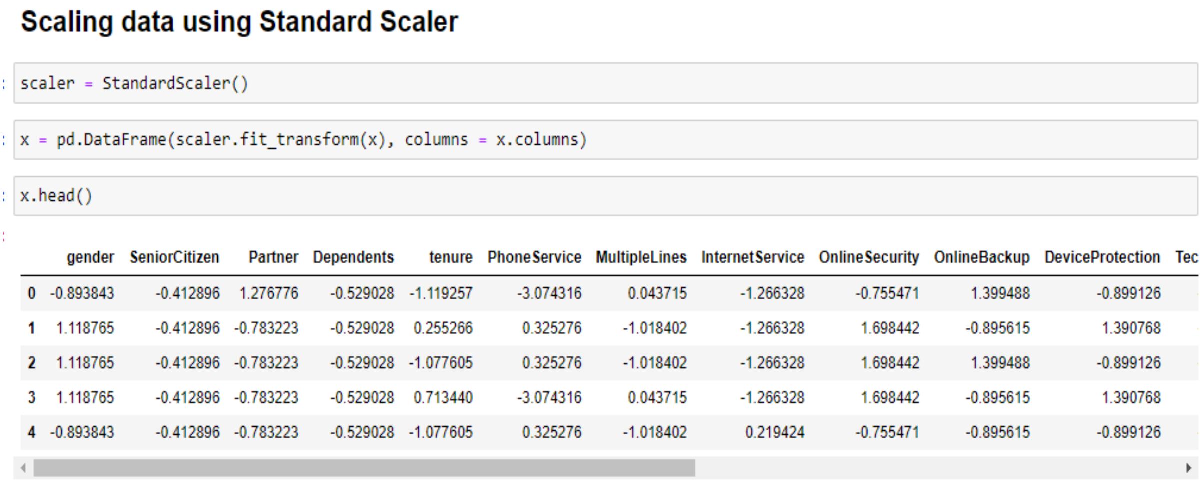
$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

While **Standardization** transforms the data to have zero mean and a variance of 1, they make our data unitless. The values are centred around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Formula for normalization:

$$X' = \frac{X - \mu}{\sigma}$$

Here we are using Standardization to scale the data.



Scaling data using Standard Scaler

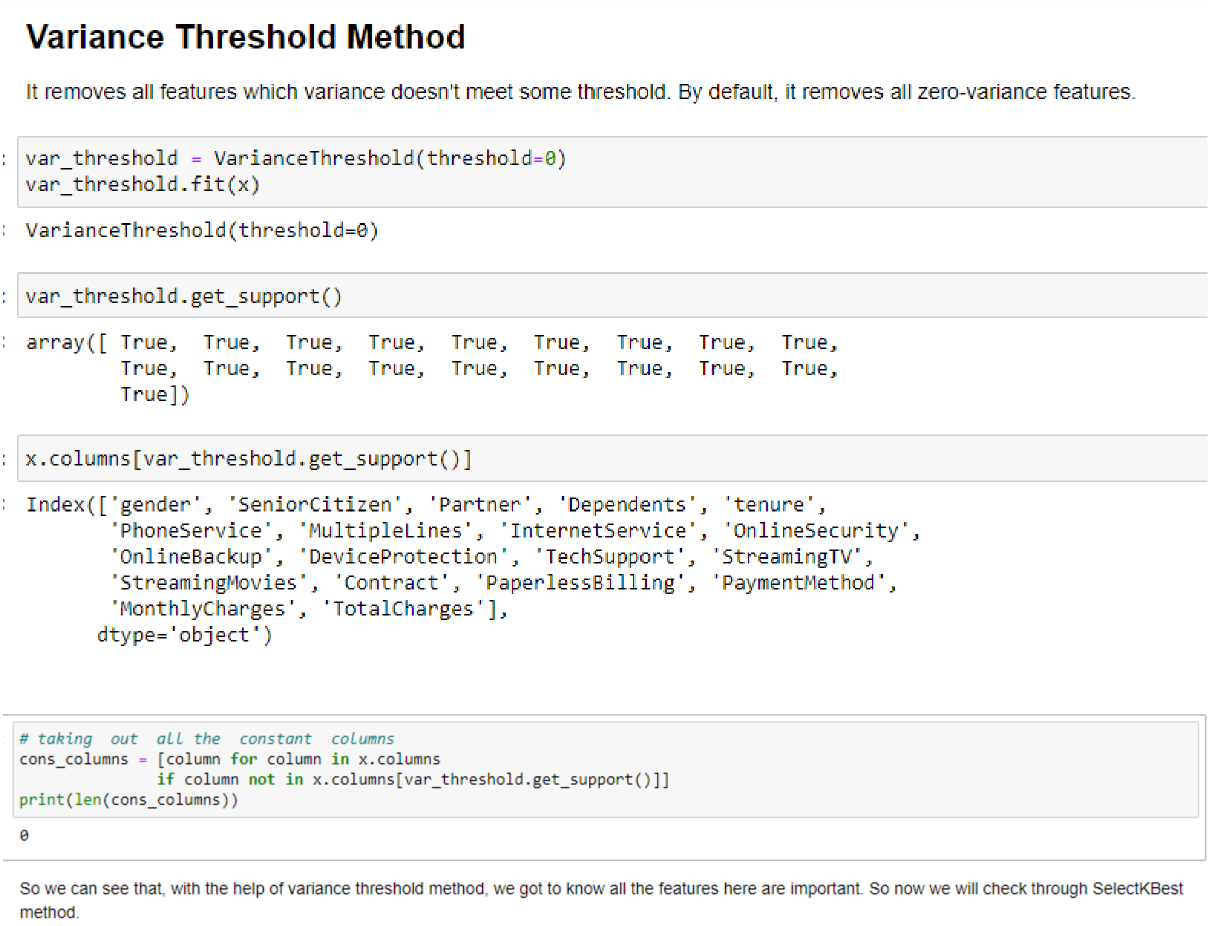
```
scaler = StandardScaler()
```

```
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

```
x.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	Tec
0	-0.893843	-0.412896	1.276776	-0.529028	-1.119257	-3.074316	0.043715	-1.266328	-0.755471	1.399488	-0.899126	
1	1.118765	-0.412896	-0.783223	-0.529028	0.255266	0.325276	-1.018402	-1.266328	1.698442	-0.895615	1.390768	
2	1.118765	-0.412896	-0.783223	-0.529028	-1.077605	0.325276	-1.018402	-1.266328	1.698442	1.399488	-0.899126	
3	1.118765	-0.412896	-0.783223	-0.529028	0.713440	-3.074316	0.043715	-1.266328	1.698442	-0.895615	1.390768	
4	-0.893843	-0.412896	-0.783223	-0.529028	-1.077605	0.325276	-1.018402	0.219424	-0.755471	-0.895615	-0.899126	

Before model creation, we need to know which features are important for prediction and which are not, So we will check using Variance Threshold Method, SelectKBest method and Multicollinearity using Variance Inflation Factor.



Variance Threshold Method

It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features.

```
var_threshold  = VarianceThreshold(threshold=0)
var_threshold.fit(x)
```

```
VarianceThreshold(threshold=0)
```

```
var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

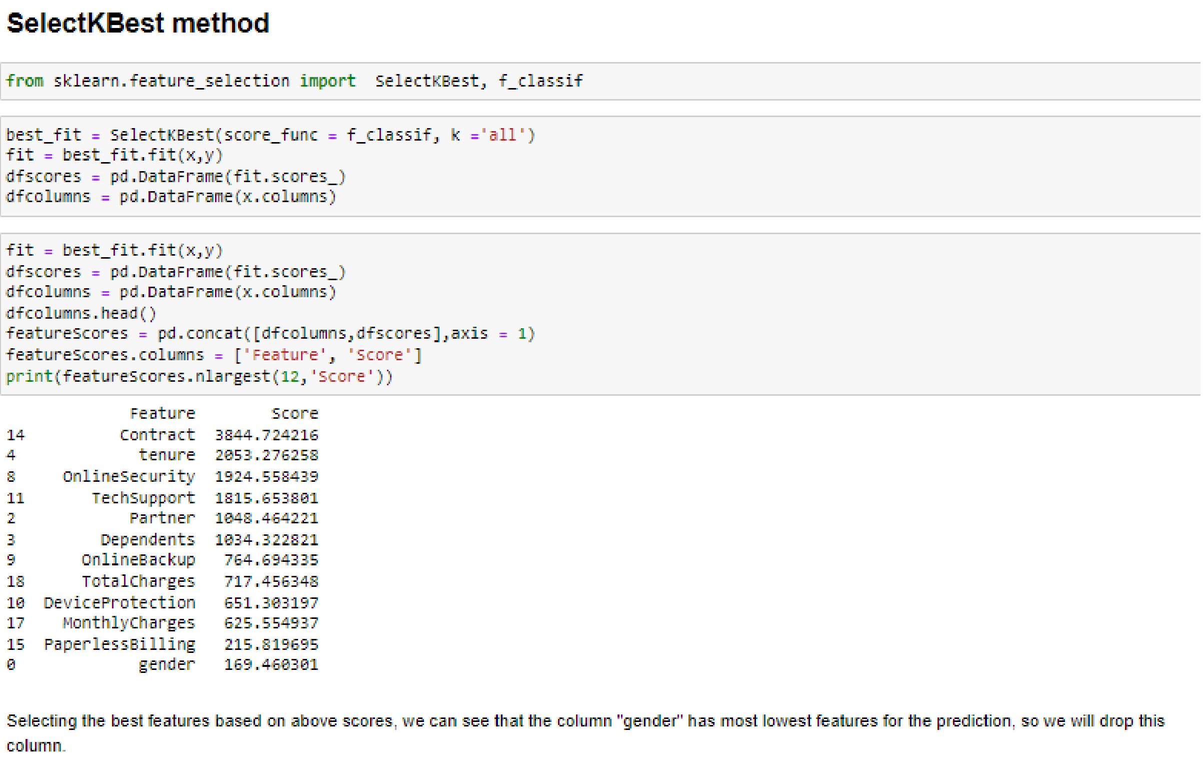
```
x.columns[var_threshold.get_support()]
```

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

```
# taking  out  all the  constant  columns
cons_columns = [column for column in x.columns
                if column not in x.columns[var_threshold.get_support()]]
print(len(cons_columns))
```

```
0
```

So we can see that, with the help of variance threshold method, we got to know all the features here are important. So now we will check through SelectKBest method.



SelectKBest method

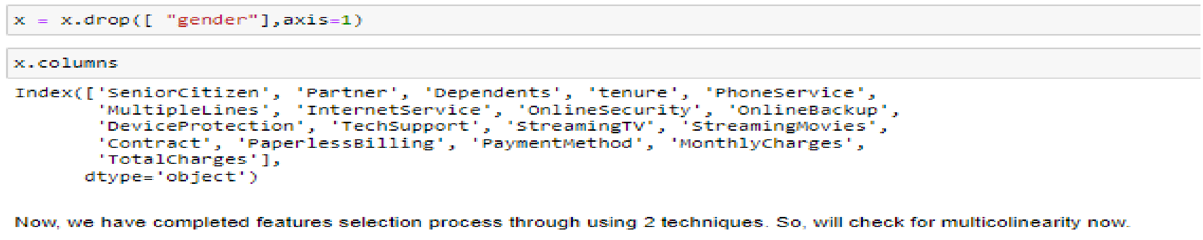
```
from sklearn.feature_selection import  SelectKBest, f_classif
```

```
best_fit = SelectKBest(score_func = f_classif, k ='all')
fit = best_fit.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = best_fit.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(12,'Score'))
```

```
             Feature        Score
14          Contract  3844.724216
4             tenure  2053.276258
8     OnlineSecurity  1924.558439
11       TechSupport  1815.653801
2            Partner  1048.464221
3         Dependents  1034.322821
9       OnlineBackup   764.694335
18      TotalCharges   717.456348
10  DeviceProtection   651.303197
17    MonthlyCharges   625.554937
15  PaperlessBilling   215.819695
0             gender   169.460301
```

Selecting the best features based on above scores, we can see that the column "gender" has most lowest features for the prediction, so we will drop this column.



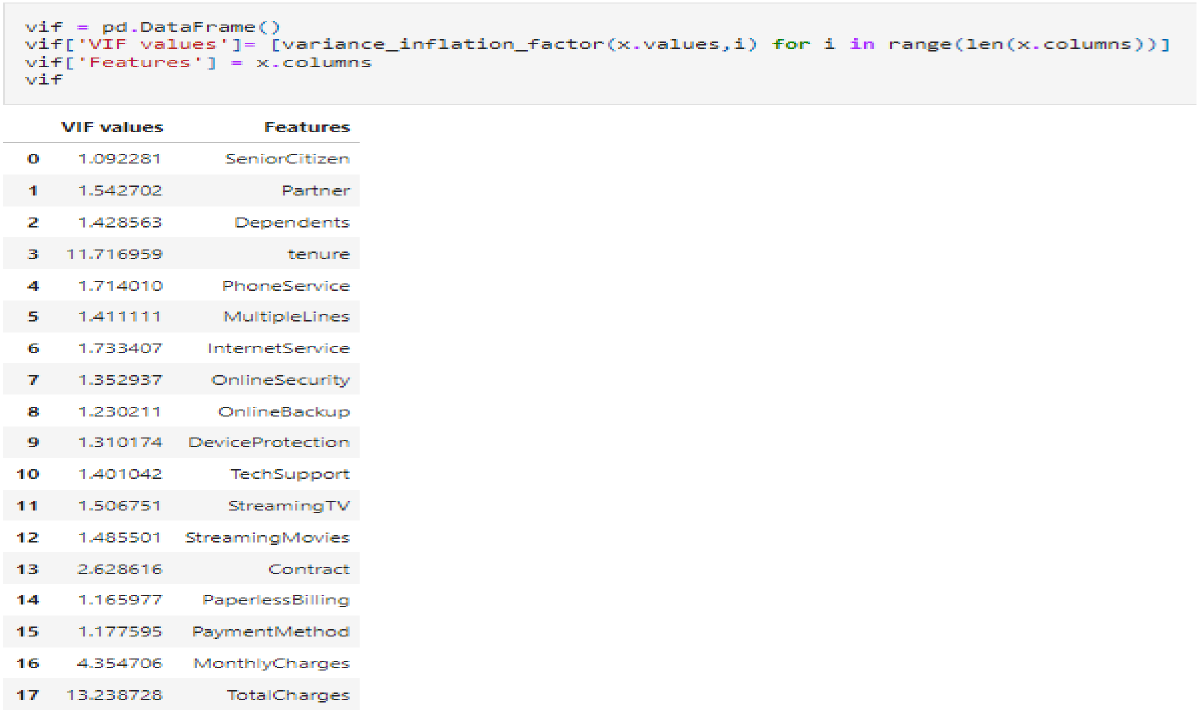
```
x = x.drop([ "gender"],axis=1)
```

```
x.columns
```

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

Now, we have completed features selection process through using 2 techniques. So, will check for multicolinearity now.

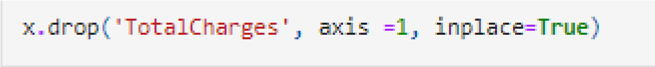
- ➢ **Checking for Multicollinearity using Variance Inflation Factor**
  - ✓ **VIF (Variance Inflation factor)**



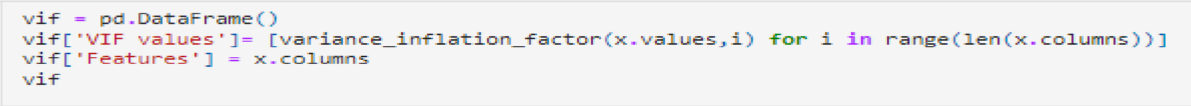
```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.092281	SeniorCitizen
1	1.542702	Partner
2	1.428563	Dependents
3	11.716959	tenure
4	1.714010	PhoneService
5	1.411111	MultipleLines
6	1.733407	InternetService
7	1.352937	OnlineSecurity
8	1.230211	OnlineBackup
9	1.310174	DeviceProtection
10	1.401042	TechSupport
11	1.506751	StreamingTV
12	1.485501	StreamingMovies
13	2.628616	Contract
14	1.165977	PaperlessBilling
15	1.177595	PaymentMethod
16	4.354706	MonthlyCharges
17	13.238728	TotalCharges

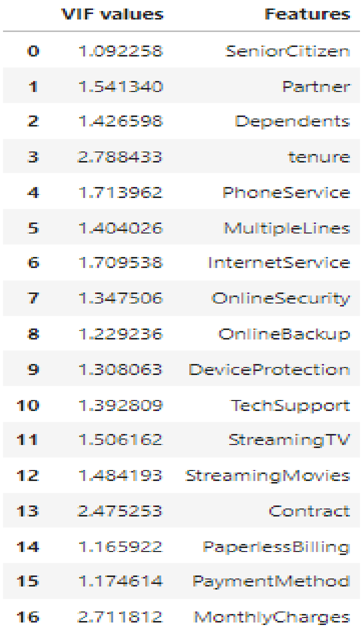
We observe that the VIF value is more than 10 in the columns 'tenure' and 'TotalCharges'. But column 'TotalCharges' is having highest VIF value. So, we will drop column 'TotalCharges' and after dropping we will again check multicollinearity.



```
x.drop('TotalCharges', axis =1, inplace=True)
```



```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```



	VIF values	Features
0	1.092258	SeniorCitizen
1	1.541340	Partner
2	1.426598	Dependents
3	2.788433	tenure
4	1.713962	PhoneService
5	1.404026	MultipleLines
6	1.709538	InternetService
7	1.347506	OnlineSecurity
8	1.229236	OnlineBackup
9	1.308063	DeviceProtection
10	1.392809	TechSupport
11	1.506162	StreamingTV
12	1.484193	StreamingMovies
13	2.475253	Contract
14	1.165922	PaperlessBilling
15	1.174614	PaymentMethod
16	2.711812	MonthlyCharges

Now, we can check Multicollinearity is removed from the columns as VIF value of all columns are less than 10. So, we will create model now.

5. <u>Building Machine Learning Models</u>

Classification Model will be built for Target Variable “Churn” as it contains categorical data. We are going to use models:

- ✓ Logistic Regression
- ✓ Random Forest Classifier
- ✓ Decision Tree Classifier
- ✓ Support Vector Machine Classifier
- ✓ KNeighbors Classifier
- ✓ Gradient Boosting Classifier
- ✓ XGB Classifier

✓ Logistic Regression:

It is a classification algorithm, that is used where the target variable is categorical. Logistic Regression is used to find a relationship between features and probability of particular outcome.

- **Random Forest Classifier:**

Random forest is basically the combination of multiple individual decision trees to act as an ensemble. Ensemble learning can be defined as a paradigm whereby multiple learners are trained to solve the same problem. Ensemble learning actually has been used in several applications such as optical character recognition, medical purpose, etc. In fact, ensemble learning can be used wherever machine learning techniques can be used.

When it comes to classification using Random Forests, the idea is that the combination of outputs of mutually exclusive nodes will outperform any individual models which are then said the predicted output. Combining multiple trees (learner) may be a better choice if the learners are performing well.

- **Decision Tree Classifier:**

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.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

- **Support Vector Machine:**

SVM is a supervised Machine Learning algorithm that is used in many classifications and regression problems. It still presents as one of the most used robust prediction methods that can be applied to many use cases involving classifications.

It works by finding an optimal separation line called a hyperplane to accurately separate 2 or more different classes. The goal is to find the optimal hyperplane separation through training the linearly separable data with the SVM algorithm.

- **KNeighbors Classifier:**

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.The K in the name of this classifier represents the k nearest neighbors, where k is an integer value specified by the user. Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data.

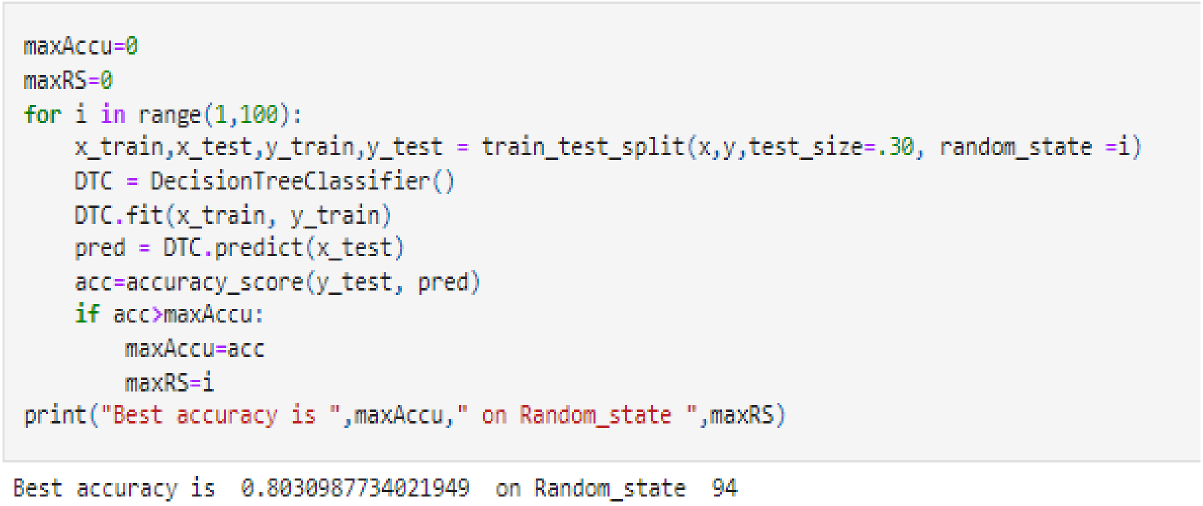
- **Gradient Boosting Classifier**

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting. Gradient boosting models are becoming popular because of their effectiveness at classifying complex datasets.

XG Boost Classifier:

eXtreme Gradient Boosting (XGBoost) is a scalable and improved version of the gradient boosting algorithm (terminology alert) designed for efficacy, computational speed and model performance. It is an open-source library and a part of the Distributed Machine Learning Community. XGBoost is a perfect blend of software and hardware capabilities designed to enhance existing boosting techniques with accuracy in the shortest amount of time.

Finding the best random state among all the models

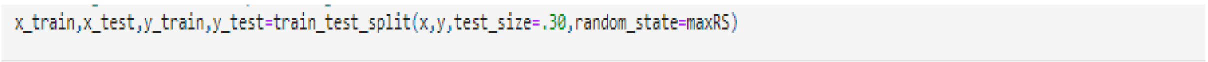


```
maxAccu=0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state =i)
    DTC = DecisionTreeClassifier()
    DTC.fit(x_train, y_train)
    pred = DTC.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

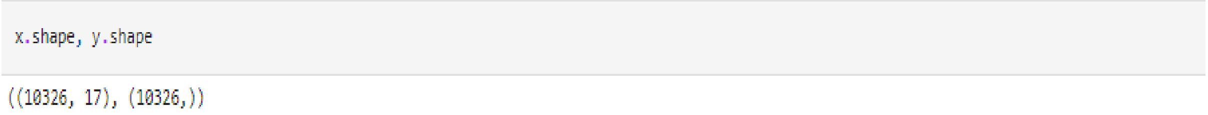
```
Best accuracy is  0.8030987734021949  on Random_state  94
```

At random state 94, we are getting best accuracy score i.e., 83%

Creating new train test split using the random state

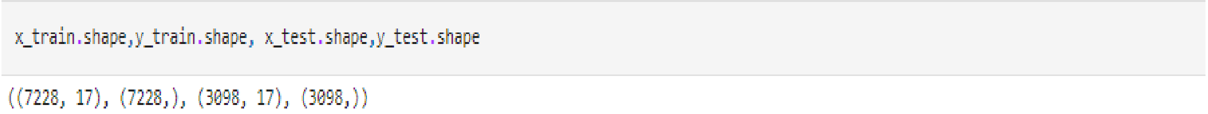


```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```



```
x.shape, y.shape
```

```
((10326, 17), (10326,))
```



```
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

```
((7228, 17), (7228,), (3098, 17), (3098,))
```

We can see the x.shape value is divided into x_train.shape and x_test.shape and like this y.shape is also divided. We will understand this by Classification problem.

Logistic Regression

```
lr=LogisticRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)

print("accuracy_score: ", accuracy_score(y_test, pred_lr))
print("confusion_matrix: \n", confusion_matrix(y_test, pred_lr))
print("classification_report: \n", classification_report(y_test,pred_lr))
```

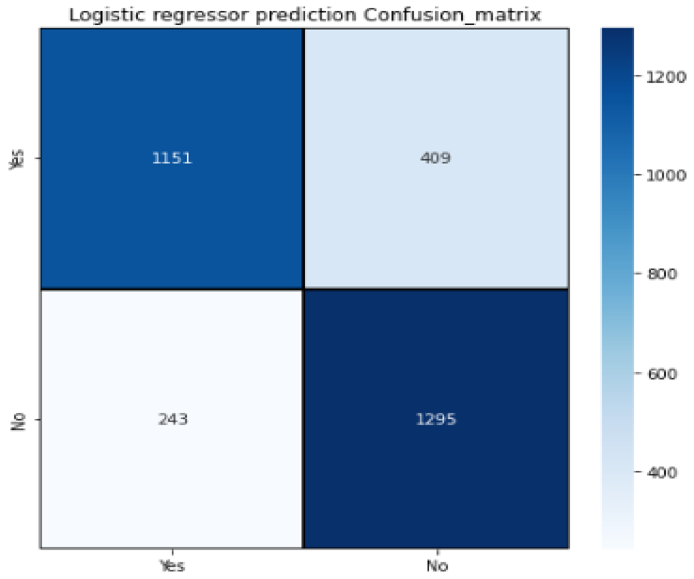
```
accuracy_score:  0.789541639767592
confusion_matrix:
 [[1151  409]
 [ 243 1295]]
classification_report:
               precision    recall  f1-score   support

           0       0.83      0.74      0.78      1560
           1       0.76      0.84      0.80      1538

    accuracy                           0.79      3098
   macro avg       0.79      0.79      0.79      3098
weighted avg       0.79      0.79      0.79      3098
```

Here we are getting 78% accuracy using Logistic Regression.

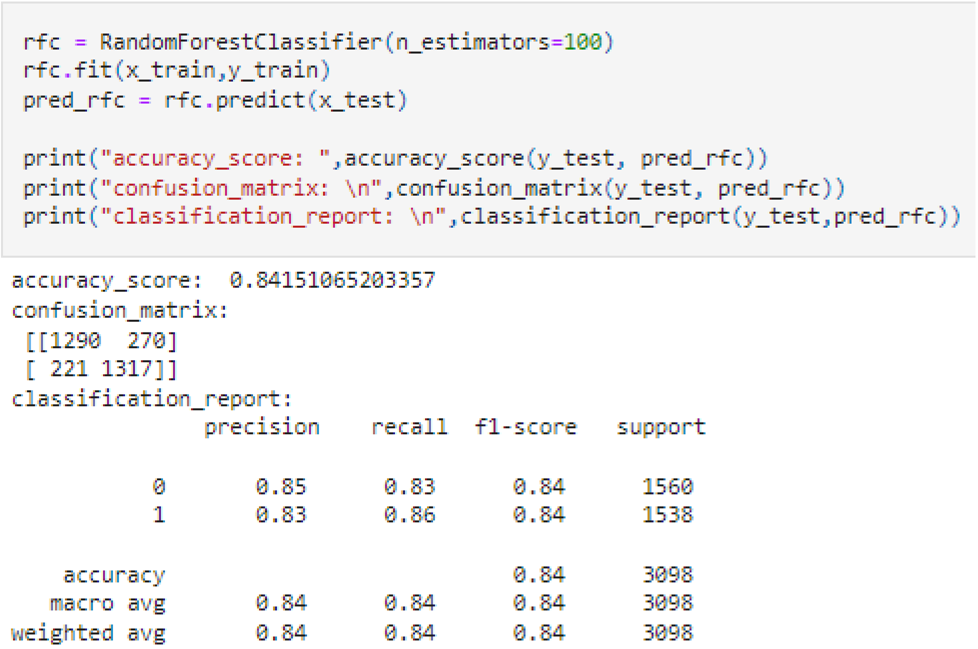
Confusion Matrix for Logistic Regression

```
cm = confusion_matrix(y_test,pred_lr)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("Logistic regressor prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'Logistic regressor prediction Confusion_matrix')
```



Random Forest Classifier

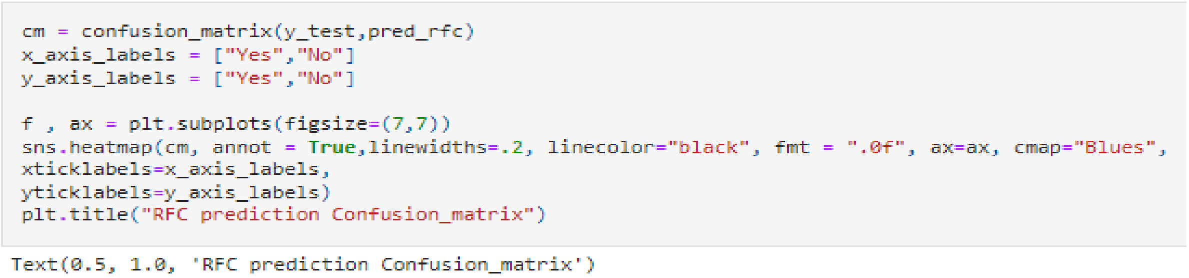


```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(x_train,y_train)
pred_rfc = rfc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_rfc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_rfc))
print("classification_report: \n",classification_report(y_test,pred_rfc))
```

```
accuracy_score:  0.84151065203357
confusion_matrix: 
 [[1290  270]
 [ 221 1317]]
classification_report: 
               precision    recall  f1-score   support

           0       0.85      0.83      0.84      1560
           1       0.83      0.86      0.84      1538

    accuracy                           0.84      3098
   macro avg       0.84      0.84      0.84      3098
weighted avg       0.84      0.84      0.84      3098
```

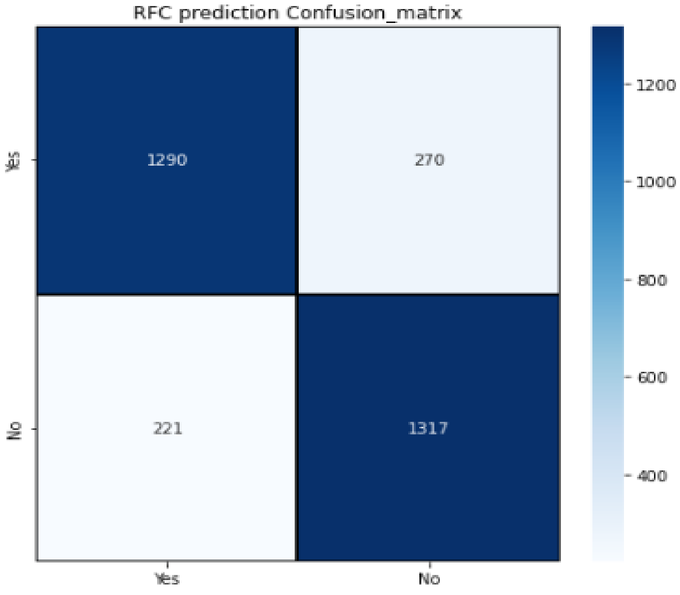
Here we are getting 84% accuracy using Random Forest Classifier.

Confusion Matrix for Random Forest Classifier.



```
cm = confusion_matrix(y_test,pred_rfc)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues", 
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("RFC prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'RFC prediction Confusion_matrix')
```



Decision Tree Classifier

```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc = dtc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_dtc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_dtc))
print("classification_report: \n",classification_report(y_test,pred_dtc))
```

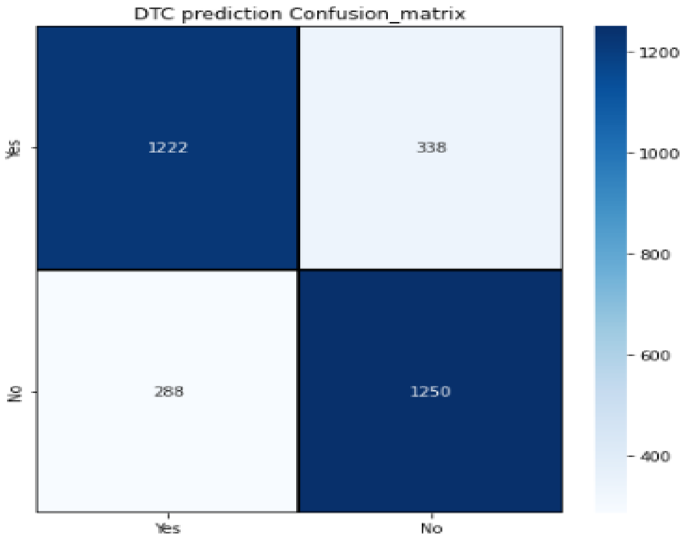
```
accuracy_score:  0.7979341510652034
confusion_matrix:
 [[1222  338]
 [ 288 1250]]
classification_report:
               precision    recall  f1-score   support

           0       0.81      0.78      0.80      1560
           1       0.79      0.81      0.80      1538

    accuracy                           0.80      3098
   macro avg       0.80      0.80      0.80      3098
weighted avg       0.80      0.80      0.80      3098
```

Here we are getting 79% accuracy using Decision Tree Classifier

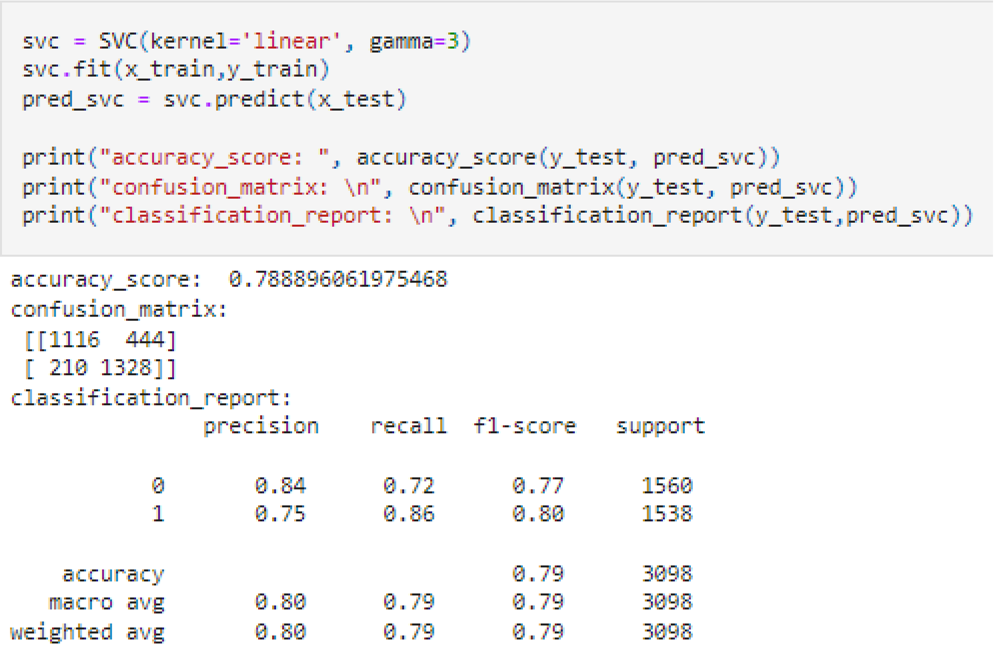
Confusion Matrix for Decision Tree Classifier

```
cm = confusion_matrix(y_test,pred_dtc)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("DTC prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'DTC prediction Confusion_matrix')
```



Support Vector Machine Classifier

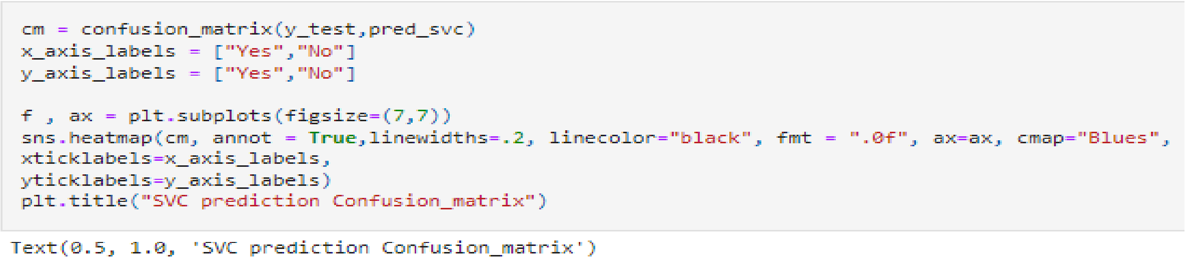


```
svc = SVC(kernel='linear', gamma=3)
svc.fit(x_train,y_train)
pred_svc = svc.predict(x_test)

print("accuracy_score: ", accuracy_score(y_test, pred_svc))
print("confusion_matrix: \n", confusion_matrix(y_test, pred_svc))
print("classification_report: \n", classification_report(y_test,pred_svc))
```

```
accuracy_score:  0.788896061975468
confusion_matrix: 
 [[1116  444]
 [ 210 1328]]
classification_report: 
               precision    recall  f1-score   support

           0       0.84      0.72      0.77      1560
           1       0.75      0.86      0.80      1538

    accuracy                           0.79      3098
   macro avg       0.80      0.79      0.79      3098
weighted avg       0.80      0.79      0.79      3098
```

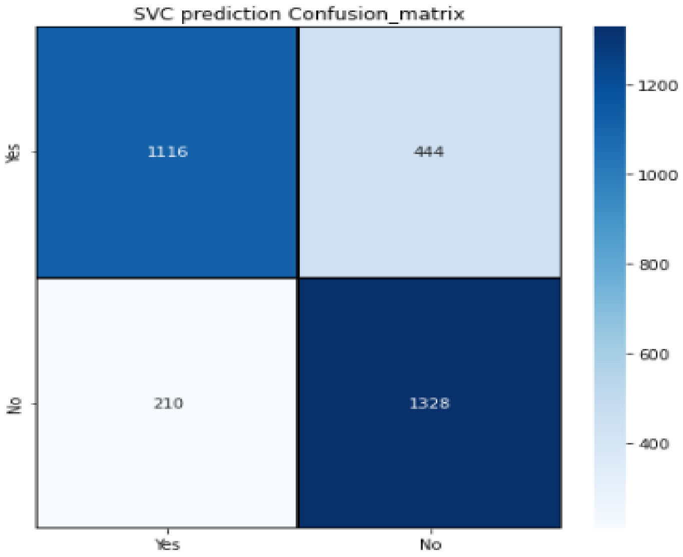
Here we are getting 78% accuracy using Support Vector Machine Classifier

Confusion Matrix for Support Vector Machine Classifier



```
cm = confusion_matrix(y_test,pred_svc)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues", 
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("SVC prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'SVC prediction Confusion_matrix')
```



KNN Classifier

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_knn))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_knn))
print("classification_report: \n",classification_report(y_test,pred_knn))
```

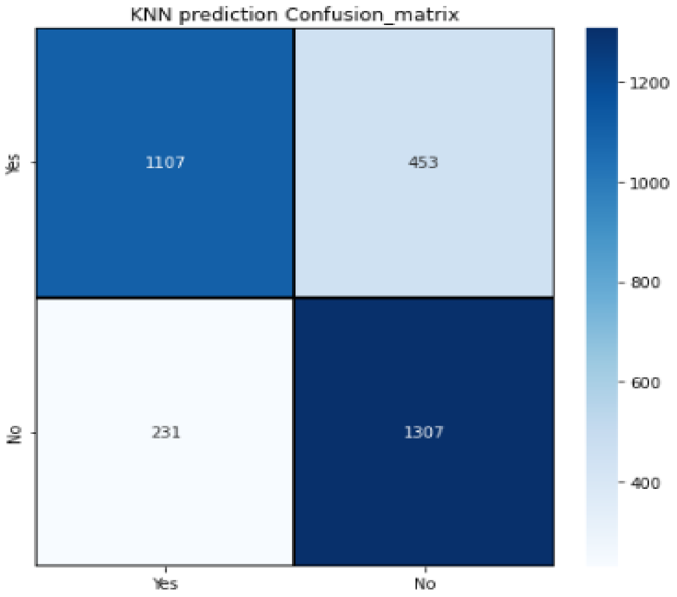
```
accuracy_score:  0.7792123950936087
confusion_matrix: 
 [[1107  453]
 [ 231 1307]]
classification_report: 
               precision    recall  f1-score   support

           0       0.83      0.71      0.76      1560
           1       0.74      0.85      0.79      1538

    accuracy                           0.78      3098
   macro avg       0.78      0.78      0.78      3098
weighted avg       0.79      0.78      0.78      3098
```

Here we are getting 77% accuracy using KNN

Confusion Matrix for KNN

```
cm = confusion_matrix(y_test,pred_knn)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("KNN prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'KNN prediction Confusion_matrix')
```



Gradient Boosting Classifier

```
gb = GradientBoostingClassifier(n_estimators =100,learning_rate=0.1, max_depth=4)
gb.fit(x_train,y_train)
pred_gb = gb.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_gb))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_gb))
print("classification_report: \n",classification_report(y_test,pred_gb))
```

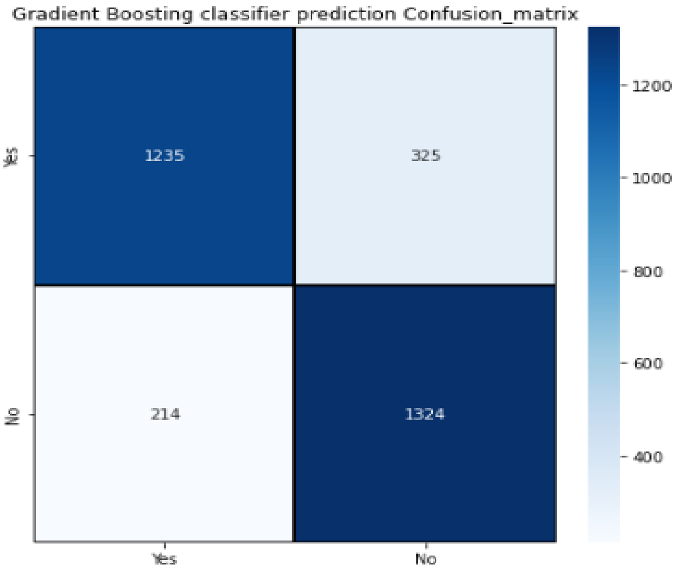
```
accuracy_score:  0.8260167850225952
confusion_matrix:
 [[1235  325]
 [ 214 1324]]
classification_report:
               precision    recall  f1-score   support

           0       0.85      0.79      0.82      1560
           1       0.80      0.86      0.83      1538

    accuracy                           0.83      3098
   macro avg       0.83      0.83      0.83      3098
weighted avg       0.83      0.83      0.83      3098
```

Here we are getting 82% accuracy using Gradient Boosting classifier.

Confusion Matrix for Gradient Boosting classifier

```
cm = confusion_matrix(y_test,pred_gb)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("Gradient Boosting classifier prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'Gradient Boosting classifier prediction Confusion_matrix')
```



XGB Classifier

```
XGBC= XGBClassifier()
XGBC.fit(x_train,y_train)
pred_XGBC = XGBC.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_XGBC))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_XGBC))
print("classification_report: \n",classification_report(y_test,pred_XGBC))
```

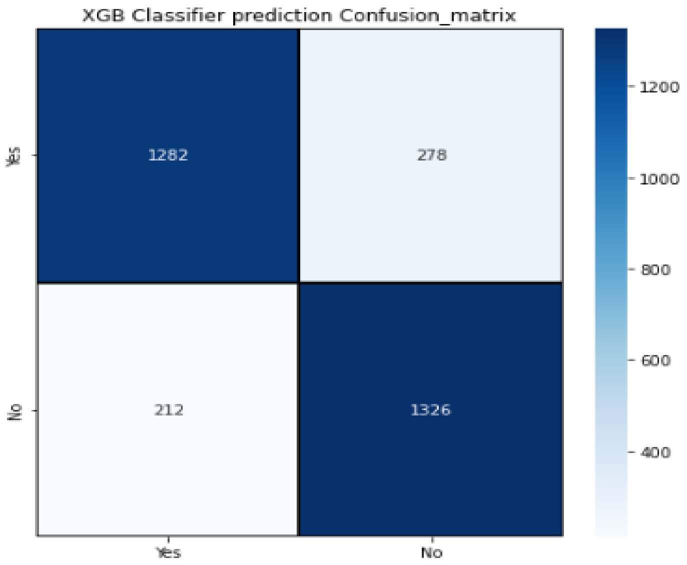
```
accuracy_score:  0.841833440929632
confusion_matrix: 
 [[1282  278]
 [ 212 1326]]
classification_report: 
               precision    recall  f1-score   support

           0       0.86      0.82      0.84      1560
           1       0.83      0.86      0.84      1538

    accuracy                           0.84      3098
   macro avg       0.84      0.84      0.84      3098
weighted avg       0.84      0.84      0.84      3098
```

Here we are getting 84% accuracy using XGB Classifier

Confusion Matrix for XGB Classifier

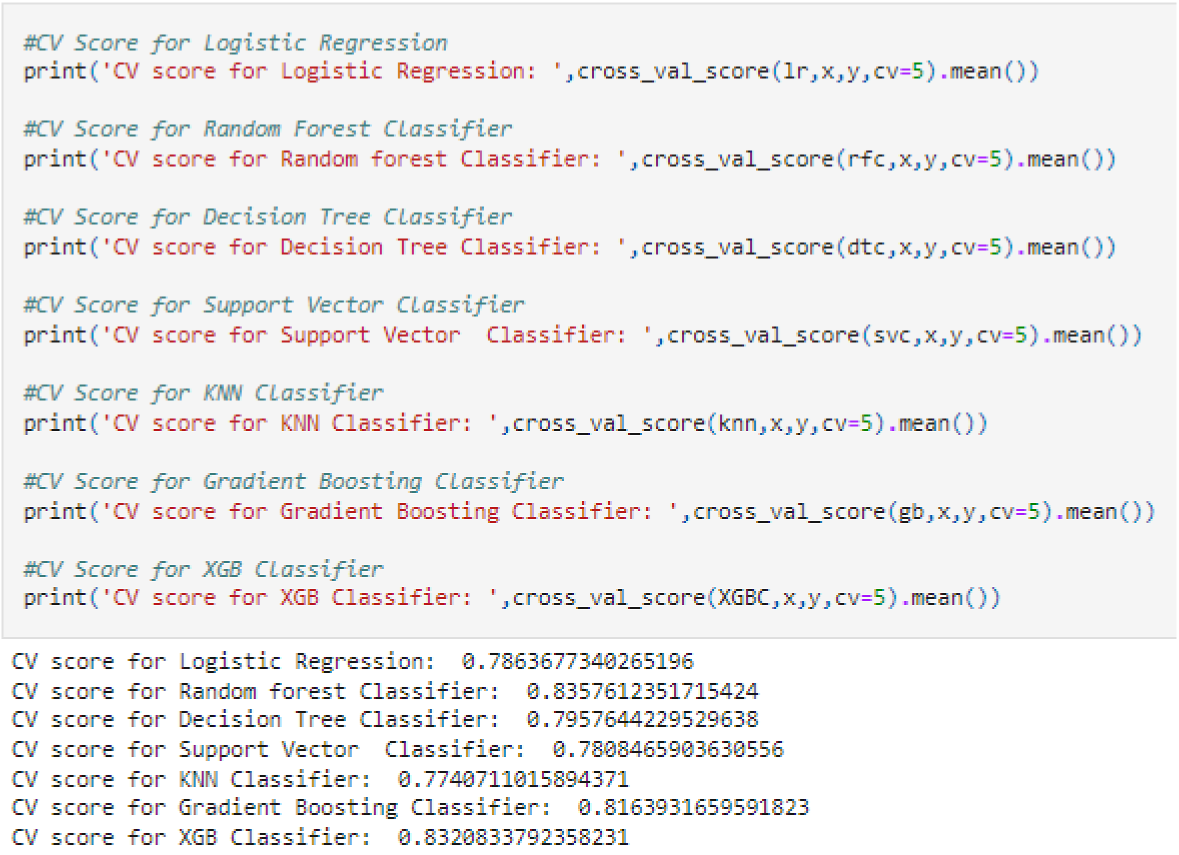
```
cm = confusion_matrix(y_test,pred_XGBC)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("XGB Classifier prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'XGB Classifier prediction Confusion_matrix')
```



➢ Cross Validation Score for all the model

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.



```
#CV Score for Logistic Regression
print('CV score for Logistic Regression: ',cross_val_score(lr,x,y,cv=5).mean())

#CV Score for Random Forest Classifier
print('CV score for Random forest Classifier: ',cross_val_score(rfc,x,y,cv=5).mean())

#CV Score for Decision Tree Classifier
print('CV score for Decision Tree Classifier: ',cross_val_score(dtc,x,y,cv=5).mean())

#CV Score for Support Vector Classifier
print('CV score for Support Vector  Classifier: ',cross_val_score(svc,x,y,cv=5).mean())

#CV Score for KNN Classifier
print('CV score for KNN Classifier: ',cross_val_score(knn,x,y,cv=5).mean())

#CV Score for Gradient Boosting Classifier
print('CV score for Gradient Boosting Classifier: ',cross_val_score(gb,x,y,cv=5).mean())

#CV Score for XGB Classifier
print('CV score for XGB Classifier: ',cross_val_score(XGBC,x,y,cv=5).mean())
```

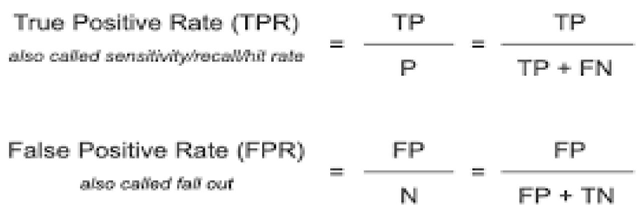
```
CV score for Logistic Regression:  0.7863677340265196
CV score for Random forest Classifier:  0.8357612351715424
CV score for Decision Tree Classifier:  0.7957644229529638
CV score for Support Vector  Classifier:  0.7808465903630556
CV score for KNN Classifier:  0.7740711015894371
CV score for Gradient Boosting Classifier:  0.8163931659591823
CV score for XGB Classifier:  0.8320833792358231
```

From the observation of accuracy and cross validation score and their difference we can predict that XGB Classifier is the best model.

➢ ROC & AUC Curve for all model

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate



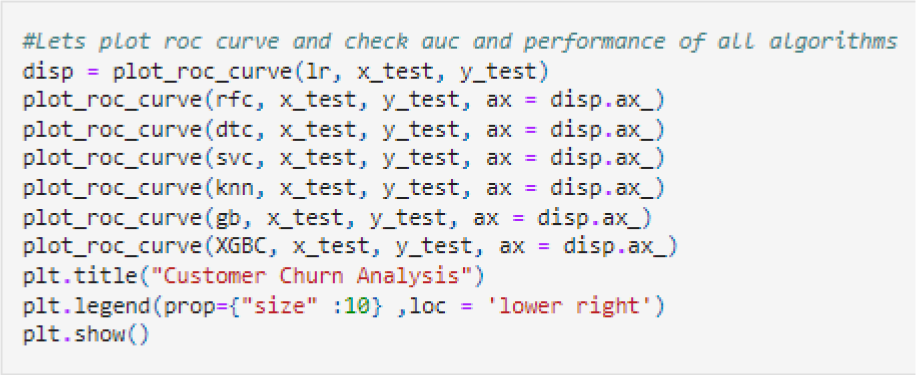
$$\text{True Positive Rate (TPR)} = \frac{TP}{P} = \frac{TP}{TP + FN}$$

also called sensitivity/recall/hit rate

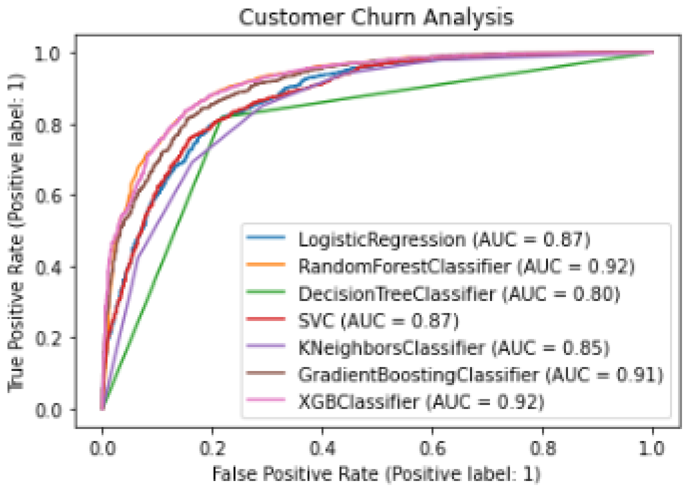
$$\text{False Positive Rate (FPR)} = \frac{FP}{N} = \frac{FP}{FP + TN}$$

also called fall out

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



```
#Lets plot roc curve and check auc and performance of all algorithms
disp = plot_roc_curve(lr, x_test, y_test)
plot_roc_curve(rfc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(dtc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(svc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
plot_roc_curve(gb, x_test, y_test, ax = disp.ax_)
plot_roc_curve(XGBC, x_test, y_test, ax = disp.ax_)
plt.title("Customer Churn Analysis")
plt.legend(prop={"size" :10} ,loc = 'lower right')
plt.show()
```



➢ **Hyper parameter tuning**

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. These parameters express important properties of the model such as its complexity or how fast it should learn. The two best strategies for Hyperparameter tuning are:

- GridSearchCV
- RandomizedSearchCV

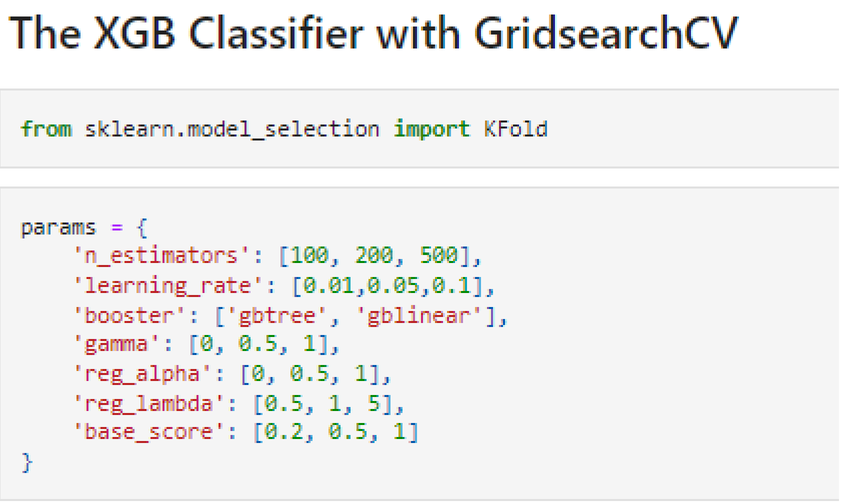
- **GridSearchCV**

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values. We will use this hyperparameters for tuning the data.

Drawback of GridSearchCV: It go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

- **RandomizedSearchCV**

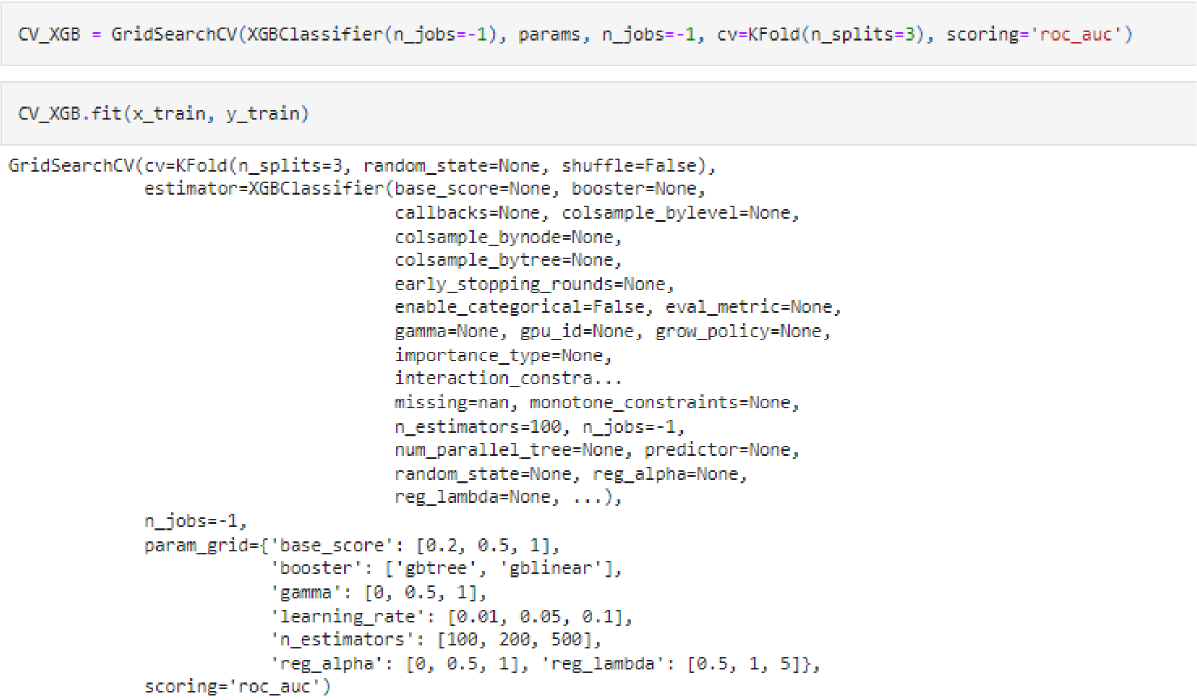
RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.



The XGB Classifier with GridsearchCV

```
from sklearn.model_selection import KFold
```

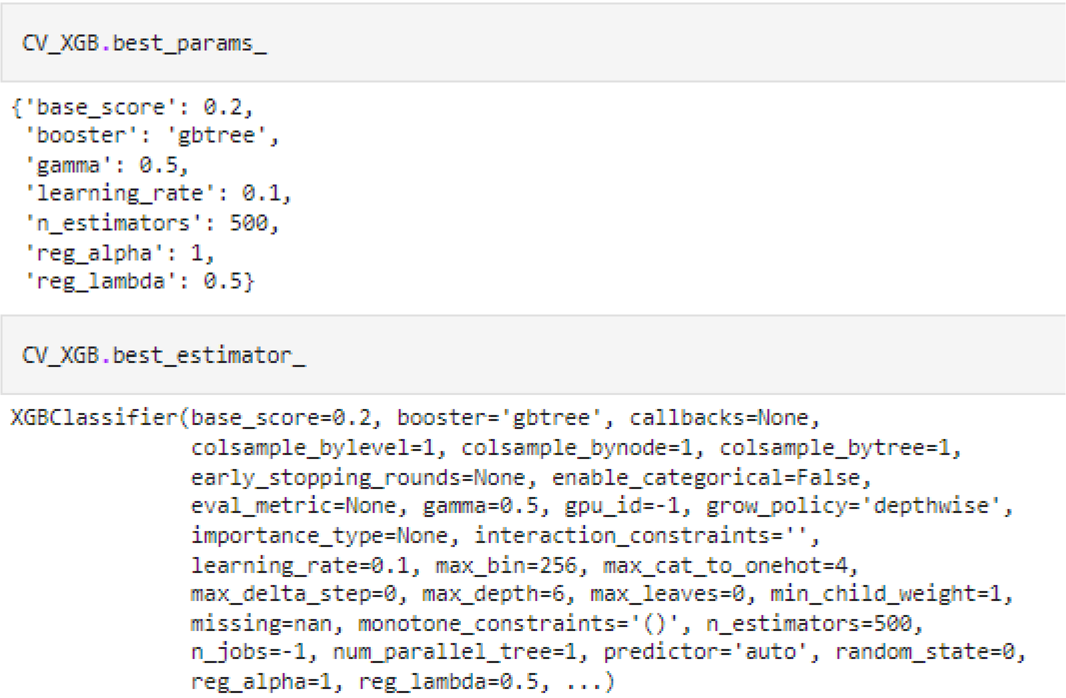
```
params = {
    'n_estimators': [100, 200, 500],
    'learning_rate': [0.01,0.05,0.1],
    'booster': ['gbtree', 'gblinear'],
    'gamma': [0, 0.5, 1],
    'reg_alpha': [0, 0.5, 1],
    'reg_lambda': [0.5, 1, 5],
    'base_score': [0.2, 0.5, 1]
}
```



```
CV_XGB = GridSearchCV(XGBClassifier(n_jobs=-1), params, n_jobs=-1, cv=KFold(n_splits=3), scoring='roc_auc')
```

```
CV_XGB.fit(x_train, y_train)
```

```
GridSearchCV(cv=KFold(n_splits=3, random_state=None, shuffle=False),
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constra...
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=-1,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, ...),
             n_jobs=-1,
             param_grid={'base_score': [0.2, 0.5, 1],
                         'booster': ['gbtree', 'gblinear'],
                         'gamma': [0, 0.5, 1],
                         'learning_rate': [0.01, 0.05, 0.1],
                         'n_estimators': [100, 200, 500],
                         'reg_alpha': [0, 0.5, 1], 'reg_lambda': [0.5, 1, 5]},
             scoring='roc_auc')
```

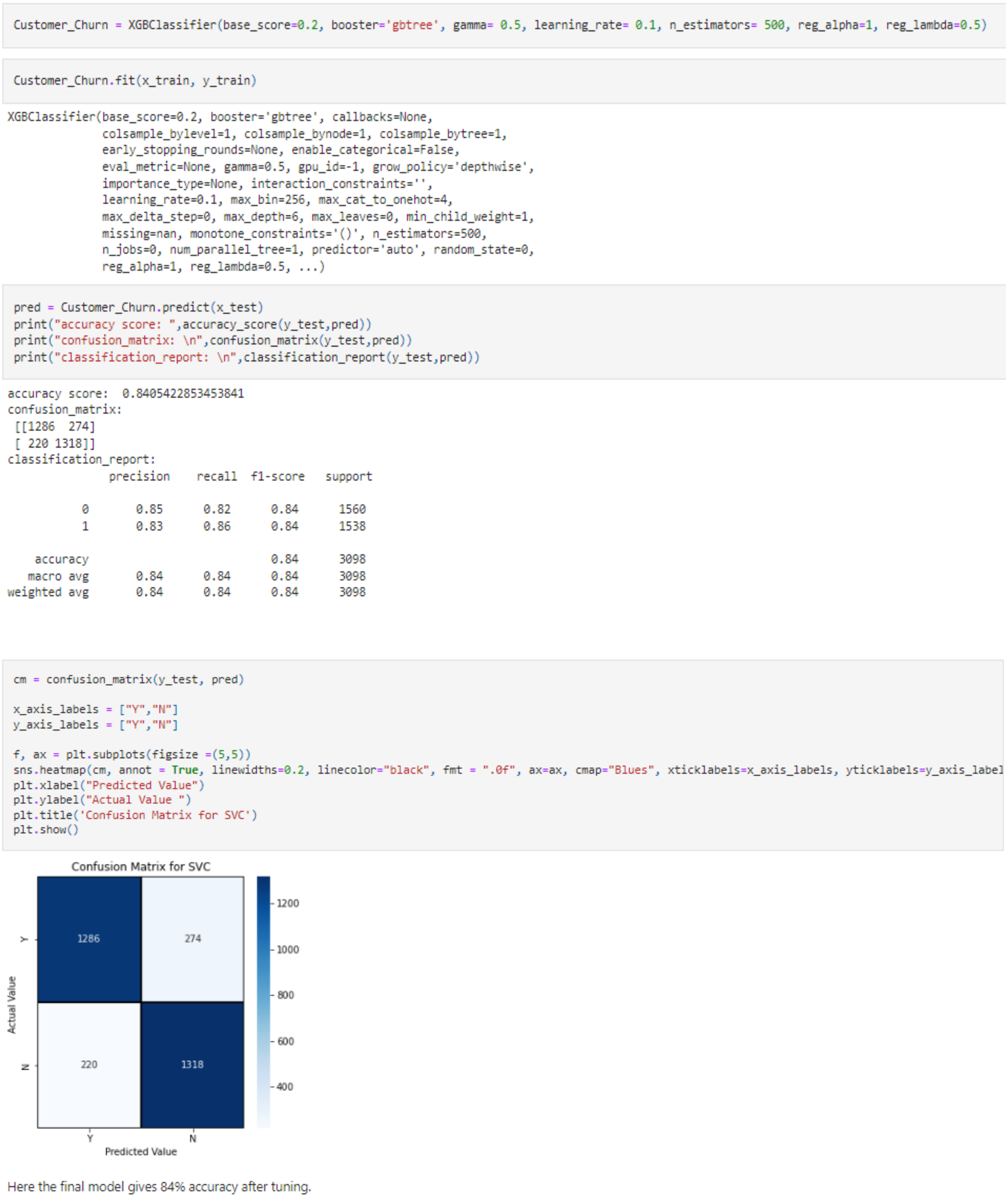


```
CV_XGB.best_params_
```

```
{'base_score': 0.2,
 'booster': 'gbtree',
 'gamma': 0.5,
 'learning_rate': 0.1,
 'n_estimators': 500,
 'reg_alpha': 1,
 'reg_lambda': 0.5}
```

```
CV_XGB.best_estimator_
```

```
XGBClassifier(base_score=0.2, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.5, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=500,
              n_jobs=-1, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=1, reg_lambda=0.5, ...)
```



```
Customer_Churn = XGBClassifier(base_score=0.2, booster='gbtree', gamma= 0.5, learning_rate= 0.1, n_estimators= 500, reg_alpha=1, reg_lambda=0.5)
```

```
Customer_Churn.fit(x_train, y_train)
```

```
XGBClassifier(base_score=0.2, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.5, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=500,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=1, reg_lambda=0.5, ...)
```

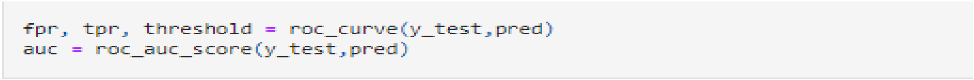
```
pred = Customer_Churn.predict(x_test)
print("accuracy score: ",accuracy_score(y_test,pred))
print("confusion_matrix: \n",confusion_matrix(y_test,pred))
print("classification_report: \n",classification_report(y_test,pred))
```

```
accuracy score:  0.8405422853453841
confusion_matrix: 
 [[1286  274]
 [ 220 1318]]
classification_report: 
               precision    recall  f1-score   support

           0       0.85      0.82      0.84      1560
           1       0.83      0.86      0.84      1538

    accuracy                           0.84      3098
   macro avg       0.84      0.84      0.84      3098
weighted avg       0.84      0.84      0.84      3098
```

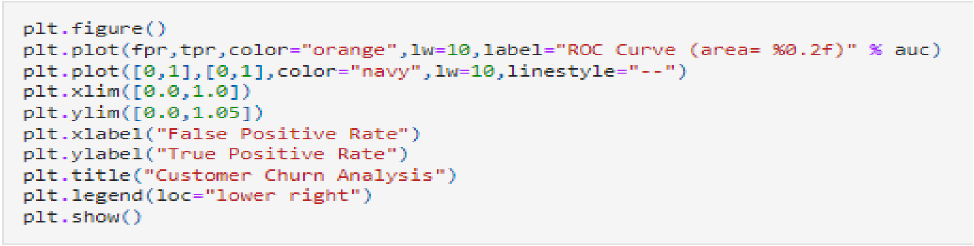
```
cm = confusion_matrix(y_test, pred)

x_axis_labels = ["Y","N"]
y_axis_labels = ["Y","N"]

f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues", xticklabels=x_axis_labels, yticklabels=y_axis_label
plt.xlabel("Predicted Value")
plt.ylabel("Actual Value ")
plt.title('Confusion Matrix for SVC')
plt.show()
```

Here the final model gives 84% accuracy after tuning.

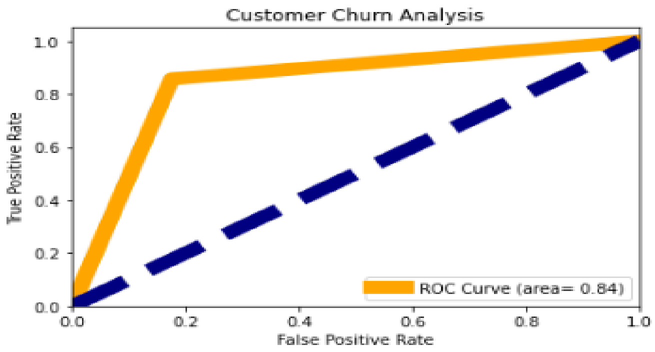
The final model gives 84% accuracy after tuning and will plot ROC-AUC curve for best model.



```
fpr, tpr, threshold = roc_curve(y_test,pred)
auc = roc_auc_score(y_test,pred)
```

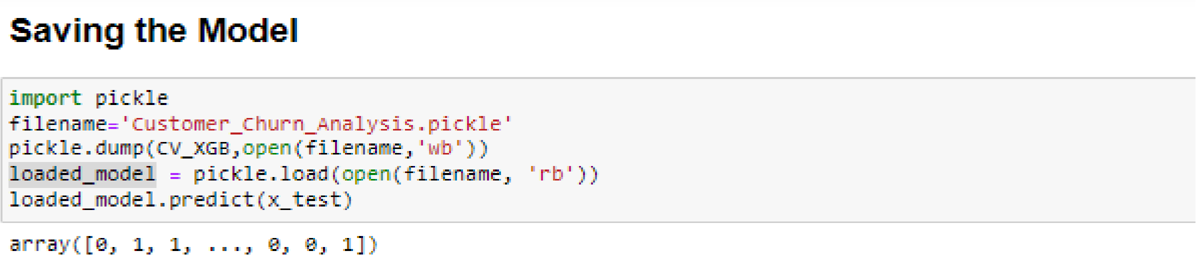


```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label="ROC Curve (area= %0.2f)" % auc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Customer Churn Analysis")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So, the best model has the area under curve as 0.84.

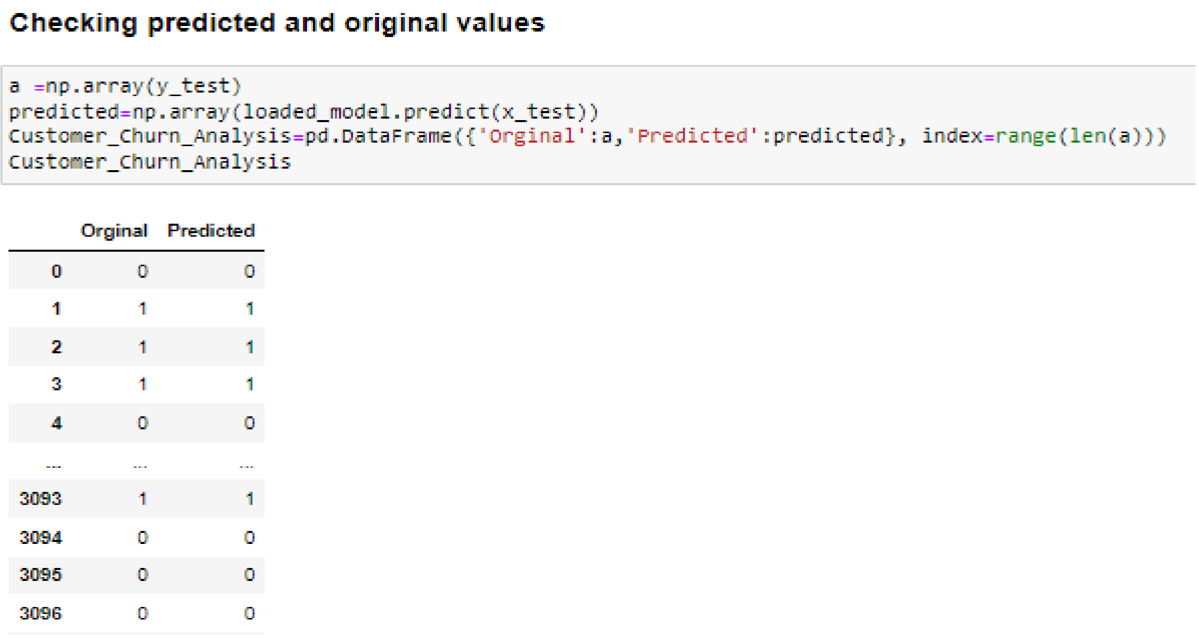
We got our best model so will save this model and will do Prediction.



Saving the Model

```
import pickle
filename='Customer_Churn_Analysis.pickle'
pickle.dump(CV_XGB,open(filename,'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.predict(x_test)
```

```
array([0, 1, 1, ..., 0, 0, 1])
```

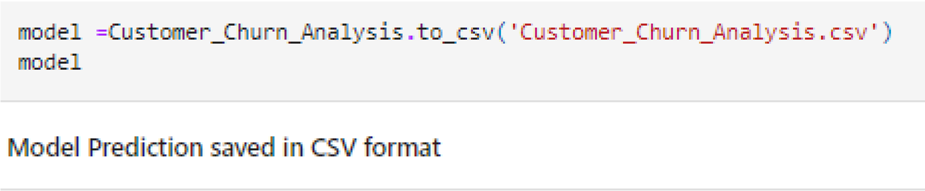


Checking predicted and original values

```
a =np.array(y_test)
predicted=np.array(loaded_model.predict(x_test))
Customer_Churn_Analysis=pd.DataFrame({'Orginal':a,'Predicted':predicted}, index=range(len(a)))
Customer_Churn_Analysis
```

	Orginal	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	0	0
...	...	...
3093	1	1
3094	0	0
3095	0	0
3096	0	0

Finally, we can see, predicted and original values matches approx 100%. So, we will save this model in csv.



```
model =Customer_Churn_Analysis.to_csv('Customer_Churn_Analysis.csv')
model
```

```
Model Prediction saved in CSV format
```

6. Concluding Remarks

Churn rate is an important indicator for companies. So, we need to:

- Analyze churn to improve your customer service team.
- Revamp your onboarding plan for new customers.
- Invest in more training for support and sales reps.
- Ask for feedback at key moments and respond promptly.
- Communicate proactively with customers.
- Offer exclusive perks to existing customers.
- Leverage feedback from free trial customers.
- To Reduce Customer Churn:
    - ✓ Focus your attention on your best customers.
    - ✓ Analyze churn as it occurs.
    - ✓ Show your customers that you care.