Customer Churn Analysis Model

INTRODUCTION

In this model, we are going TO predict churn of customer.

Used customer churn dataset which contains feature columns of customer and at the end, one column of target for which we will train model.

This is an classification column, as we have found target variable as categorical variable

OVERVIEW OF DATASET:

- This dataset is having 7043 rows and 21 columns.
- Dataset is having total 21 columns in which 18 columns are of object type, 1 column is of float type and 2 column are of int type

PROBLEM STATEMENT:

Customer churn is when a company's customers stop doing business with that company. Business are very keen on measuring churn because keeping on existing customer is far less expensive then acquiring a new customers. Now 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 priorities 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.

We will examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

FEATURES OF DATASET:

- **customerID**: Name of the customer
- **gender**: gender of customer
- **SeniorCitizen**: Senior Citizen yes or no
- **Partner**: Having partner Yes or No
- **Dependents**: Having dependent Yes or No.
- **tenure**: Tenure of customer
- **PhoneService**: Having Phone services Yes or No.
- **MultipleLines**: Having Multiple Lines service
- **InternetService**: Which internet services one is using.

- **OnlineSecurity**: Having online security Yes or No
- **OnlineBackup**: Having online Backup Yes or No
- **DeviceProtection:** Having device protection Yes or No
- **TechSupport:** Getting Technical Support or not
- **StreamingTV:** Getting streaming TV Yes or not
- **StreamingMovies:** Getting Streaming Movies Yes or not
- **Contract:** Type of contract
- **PaperlessBilling:** Getting paperless billing Yes or Not
- **PaymentMethod:** Mode of payment one is choosing
- **MonthlyCharges:** Monthly Charges one is paying
- **TotalCharges:** Total Charges one is paying
- **Churn : Customer Churn Analysis**

MODEL BUILDIING:

Used Libraries:

```
# Importing important Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from imblearn.over_sampling import SMOTE
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
```

LOADING DATASET

```
url = 'https://raw.githubusercontent.com/Bhushan0130/Datasets/main/Telecom_customer_churn.csv'
df = pd.read_csv(url)
df.shape
# (7043, 21)
```

```
(7043, 21)
```

```
pd.set_option('display.max_rows', None)  # to maximize the rows
pd.set_option('display.max_columns', None)  # to maximize the columns
```

```
df.head()  # top 5 rows
```

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

```
df.head()  # top 5 rows
```

ce	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
SL	No	Yes	No	No	No	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
SL	Yes	No	Yes	No	No	No	One year	No	Mailed check	56.95	1889.5	No
SL	Yes	Yes	No	No	No	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
SL	Yes	No	Yes	Yes	No	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
ic	No	No	No	No	No	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

INFORMATION ABOUT DATASET:

```
df.info()

# dataset is haviing 7043 rows and 21 columns
# 18 columns are of object type and 3 are of int type
# As non_null values are same for each column, means null values are not present in the dataset
```

```
<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
```

```
df.dtypes

# # #   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
```

PREPROCESSING PART:

***Note**: Here one hack found that, Total Charges column containing value of int type but its column type is object, need to convert this column into numeric type*

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors = 'coerce')
# Coverting object column into numeric type
```

```
df['TotalCharges'].dtypes  # TotalCharges column have been converted into float64 type

# dtype('float64')
```

```
dtype('float64')
```

NULL VALUE CHECKING:

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

# null values are present TotalCharges column
# TotalCharges        11
```

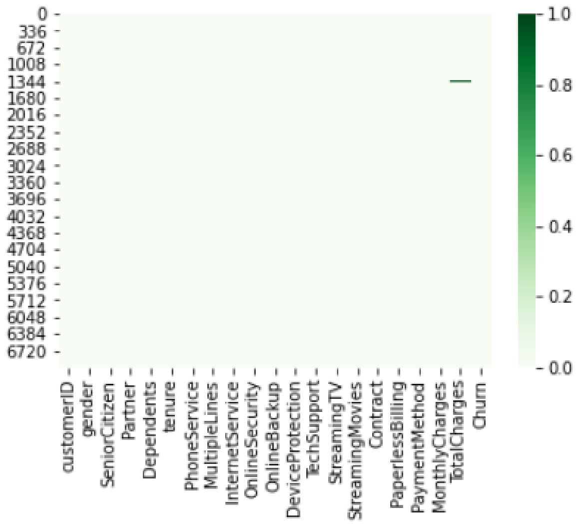
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

```
df.isnull().sum().sum()
# Total 11 null values, we are having,
# this count is so small for our dataset,
# we can remove this rows and it will not impact much to our training dataset

# 11
```

```
11
```

```python
sns.heatmap(df.isnull(), cmap = 'Greens')
```

```
<AxesSubplot:>
```




```python
# HeatMap is not clear, Few null values are present in the dataset
```

```python
df.shape
# (7043, 21)
```

```
(7043, 21)
```

```python
df.dropna(inplace = True)   # Remvoing null values
```

```python
df.shape   # Not major difference found after remove null values
# (7032, 21)
```

```
(7032, 21)
```

Few null values were found in the dataset and count of that null value was to small, therefore we have drop that rows instead performing mean, mode imputation

DELETING UN-NECESSARY COLUMNS:

```
# customerID column: is not needed for building ML model, because it is just a unique number for each customer
# therefore better to remove it from the dataset
```

```
# Found these columns as un usuable for building Ml model
df.drop(columns = ['customerID'], inplace = True)
```

```
df.shape
# (7032, 20)
```

```
(7032, 20)
```

CONVERTING TENURE COLUMN INTO CLASSES:

```
# Tenure column can be converted into some class range
print('Minimum value of tenure column: ', df['tenure'].min())
print('Maxium value of tenure column: ', df['tenure'].max())

# Minimum value of tenure column:  1
# Maxium value of tenure column:  72
```

```
Minimum value of tenure column:  1
Maxium value of tenure column:  72
```

```
df['tenure'].unique() # Unique value of tenure column

# array([ 1, 34,  2, 45,  8, 22, 10, 28, 62, 13, 16, 58, 49, 25, 69, 52, 71,
#        21, 12, 30, 47, 72, 17, 27,  5, 46, 11, 70, 63, 43, 15, 60, 18, 66,
#         9,  3, 31, 50, 64, 56,  7, 42, 35, 48, 29, 65, 38, 68, 32, 55, 37,
#        36, 41,  6,  4, 33, 67, 23, 57, 61, 14, 20, 53, 40, 59, 24, 44, 19,
#        54, 51, 26, 39], dtype=int64)
```

```
array([ 1, 34,  2, 45,  8, 22, 10, 28, 62, 13, 16, 58, 49, 25, 69, 52, 71,
       21, 12, 30, 47, 72, 17, 27,  5, 46, 11, 70, 63, 43, 15, 60, 18, 66,
        9,  3, 31, 50, 64, 56,  7, 42, 35, 48, 29, 65, 38, 68, 32, 55, 37,
       36, 41,  6,  4, 33, 67, 23, 57, 61, 14, 20, 53, 40, 59, 24, 44, 19,
       54, 51, 26, 39], dtype=int64)
```

```
# Loop for creating class of tenure class
tenure_class = []
for i in df['tenure']:
    if i in range (0, 16):
        tenure_class.append('0-15 yrs')
    elif i in range (16, 31):
        tenure_class.append('16-30 yrs')
    elif i in range(31,46):
        tenure_class.append('31-45 yrs')
    elif i in range(46, 61):
        tenure_class.append('46-60 yrs')
    elif i in range(61, 75):
        tenure_class.append('60+ yrs')
len(tenure_class)

# 7032
```

```
7032
```

```
df['Tenure'] = tenure_class  # Created new column for tenure column
```

```
# As we converted tenure column into Tenure column, therefore now we can remove tenure coloum from dataset
df.drop(columns = ['tenure'], inplace = True)
```

VALUE COUNT FOR OBJECT TYPE COLUMNS:

```
object_col = df.select_dtypes(include = 'object').keys()
object_col

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

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

```
for i in object_col:
    print(i, 'column')
    print(df[i].value_counts(), '\n')
```

```
gender column
Male      3549
Female    3483
Name: gender, dtype: int64

Partner column
No     3639
Yes    3393
Name: Partner, dtype: int64

Dependents column
No     4933
Yes    2099
Name: Dependents, dtype: int64

PhoneService column
Yes    6352
No      680
Name: PhoneService, dtype: int64

MultipleLines column
No                  3385
Yes                 2967
No phone service     680
Name: MultipleLines, dtype: int64
```

```
InternetService column
Fiber optic    3096
DSL            2416
No             1520
Name: InternetService, dtype: int64

OnlineSecurity column
No                     3497
Yes                    2015
No internet service    1520
Name: OnlineSecurity, dtype: int64

OnlineBackup column
No                     3087
Yes                    2425
No internet service    1520
Name: OnlineBackup, dtype: int64

DeviceProtection column
No                     3094
Yes                    2418
No internet service    1520
Name: DeviceProtection, dtype: int64

TechSupport column
No                     3472
Yes                    2040
No internet service    1520
Name: TechSupport, dtype: int64

StreamingTV column
No                     2809
Yes                    2703
No internet service    1520
Name: StreamingTV, dtype: int64
```

```
StreamingMovies column
No                     2781
Yes                    2731
No internet service    1520
Name: StreamingMovies, dtype: int64

Contract column
Month-to-month    3875
Two year          1685
One year          1472
Name: Contract, dtype: int64

PaperlessBilling column
Yes    4168
No     2864
Name: PaperlessBilling, dtype: int64

PaymentMethod column
Electronic check             2365
Mailed check                 1604
Bank transfer (automatic)    1542
Credit card (automatic)      1521
Name: PaymentMethod, dtype: int64

Churn column
No     5163
Yes    1869
Name: Churn, dtype: int64

Tenure column
0-15 yrs     2459
60+ yrs      1407
16-30 yrs    1171
46-60 yrs    1038
31-45 yrs     957
Name: Tenure, dtype: int64
```

VISUALIZATION OF CATEGORICAL COLUMNS:

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

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

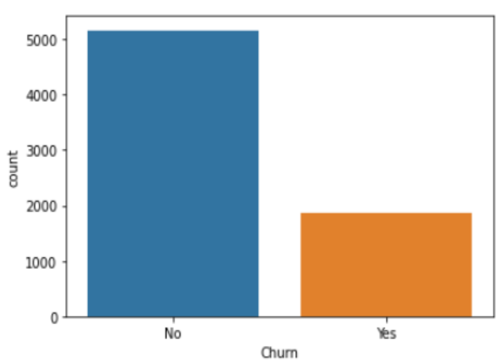
```
df.Churn.value_counts()
```

```
No     5163
Yes    1869
Name: Churn, dtype: int64
```

COUNT PLOT FOR TARGET COLUMN:

```
sns.countplot(df.Churn)
```

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

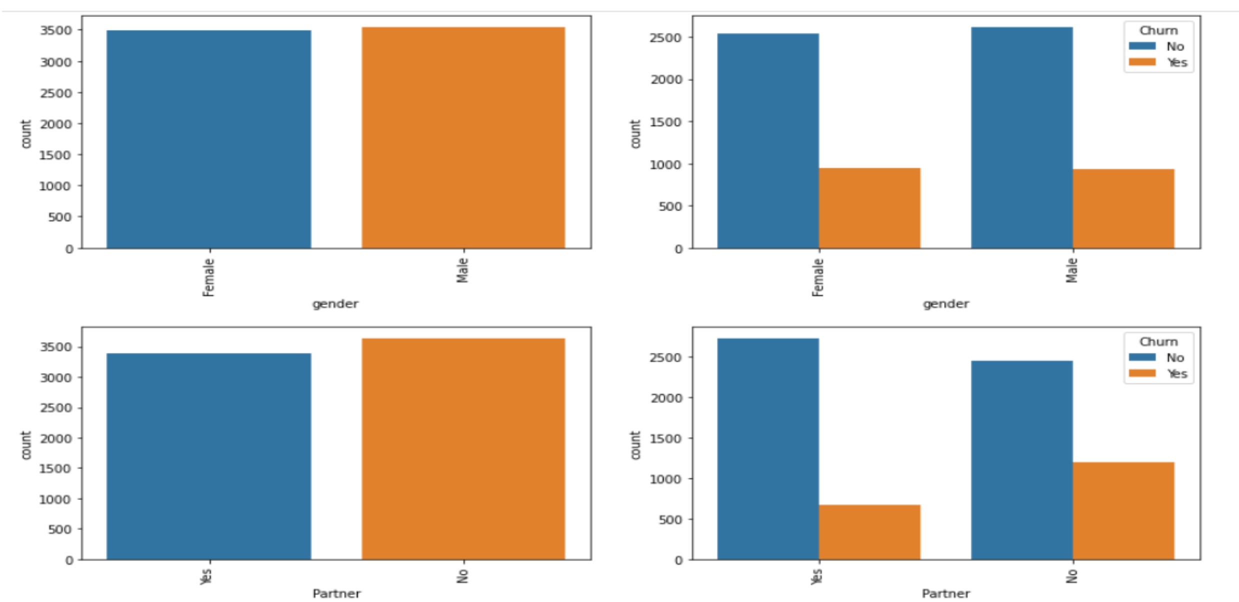


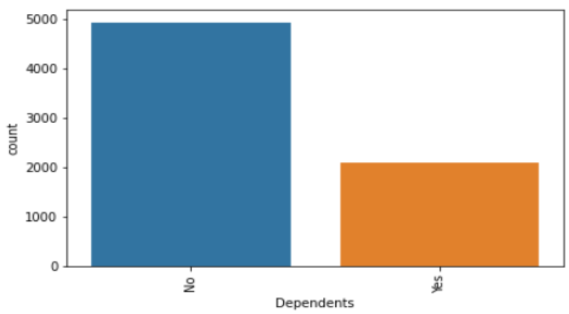

```
# As we can see our target variable value_count is not same for both class, need to balance this my applying SMOTE technique
```

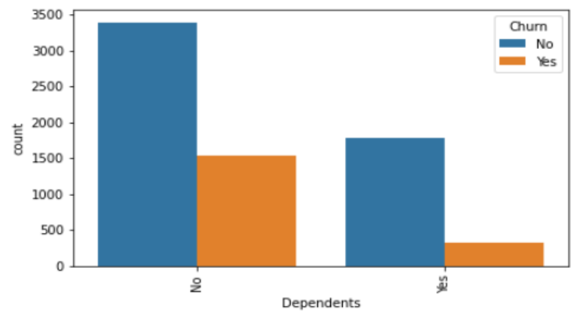
COUNT PLOT FOR FEATURE COLUMNS

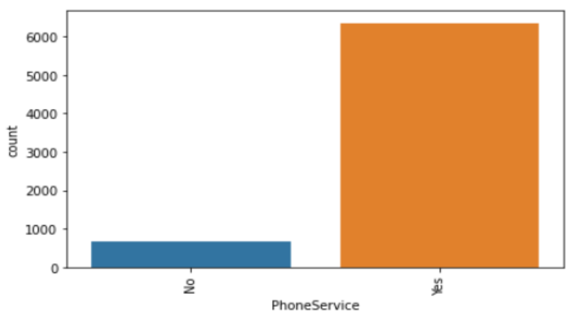
- Code for creating multiple count plots for categorical column of dataset

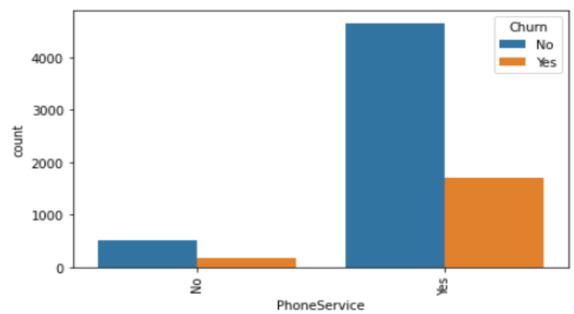
```
# countplot for object type columns
for i in object_col:
    plt.figure(figsize= (15, 4))
    l = list(df[i].unique())
    plt.subplot(1,2, 1)
    bar = sns.countplot(df[i])
    bar.set_xticklabels(labels = l, rotation = 90)

    plt.subplot(1,2, 2)
    bar1 = sns.countplot(df[i], hue = df['Churn'])
    bar1.set_xticklabels(labels = l, rotation = 90)

    plt.show()
```

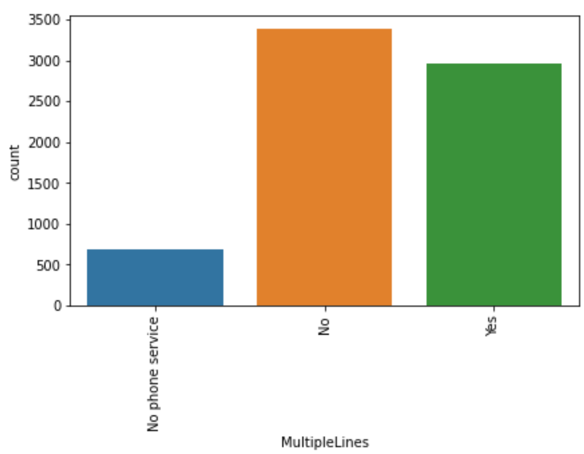


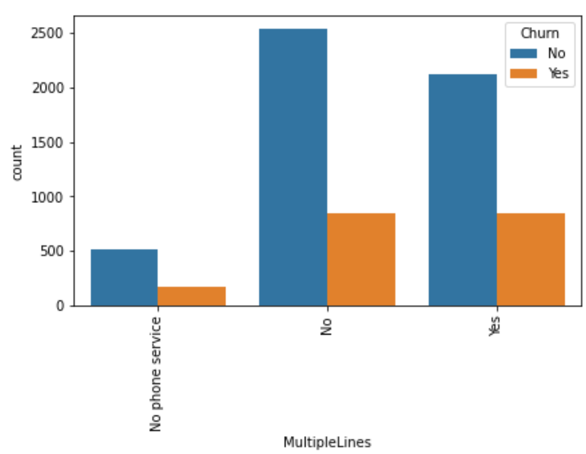

count
0
1000
2000
3000
4000
5000
No
Yes
Dependents

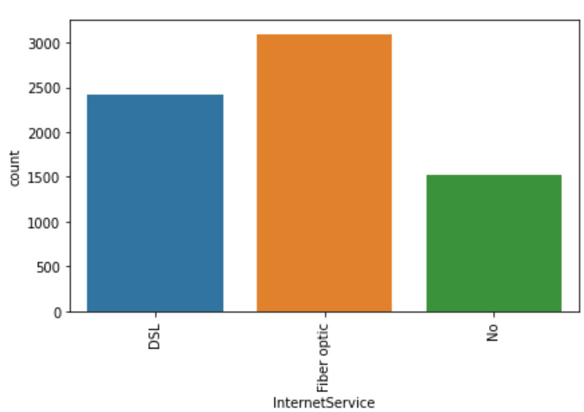

count
0
500
1000
1500
2000
2500
3000
3500
Churn
No
Yes
No
Yes
Dependents

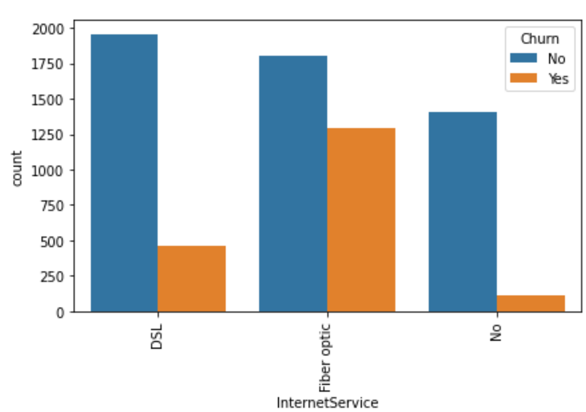

count
0
1000
2000
3000
4000
5000
6000
No
Yes
PhoneService

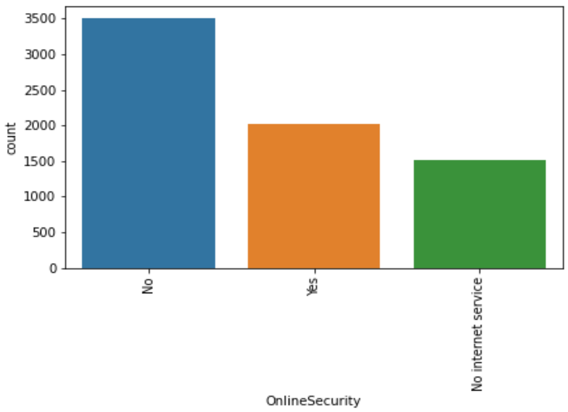

count
0
1000
2000
3000
4000
Churn
No
Yes
No
Yes
PhoneService


count
0
500
1000
1500
2000
2500
3000
3500
No phone service
No
Yes
MultipleLines

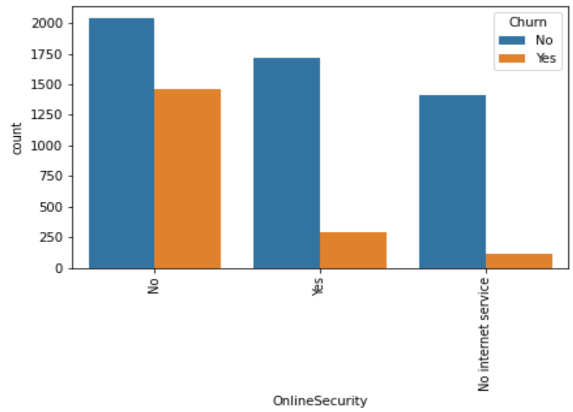

count
0
500
1000
1500
2000
2500
Churn
No
Yes
No phone service
No
Yes
MultipleLines


count
0
500
1000
1500
2000
2500
3000
DSL
Fiber optic
No
InternetService

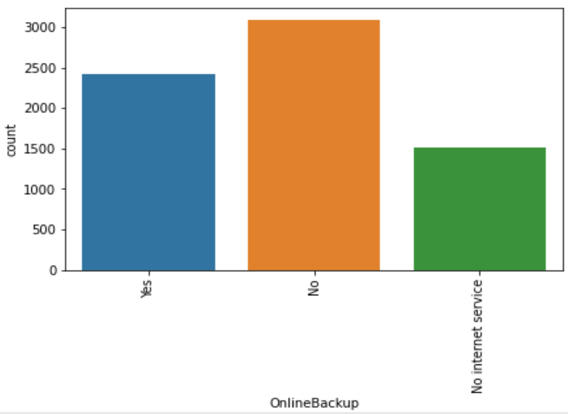

count
0
250
500
750
1000
1250
1500
1750
2000
Churn
No
Yes
DSL
Fiber optic
No
InternetService



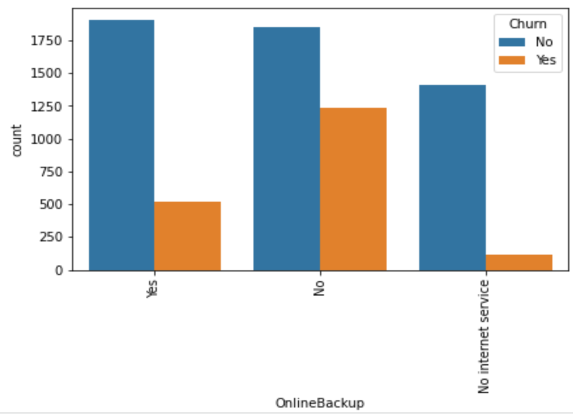
count
No
Yes
No internet service
OnlineSecurity



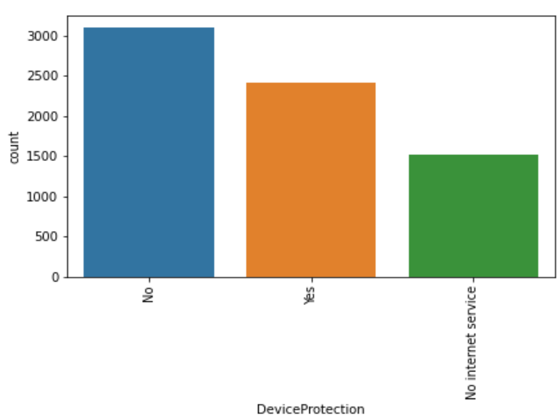
Churn
No
Yes
count
No
Yes
No internet service
OnlineSecurity



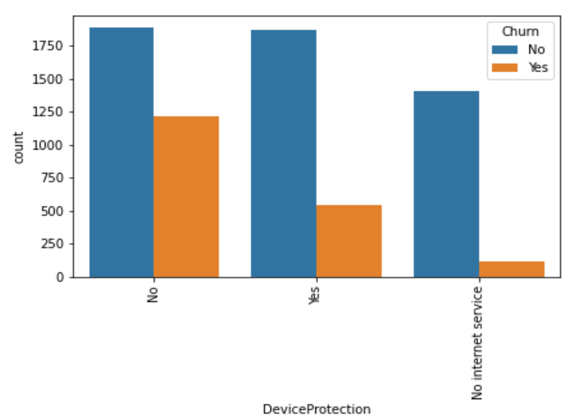
count
Yes
No
No internet service
OnlineBackup



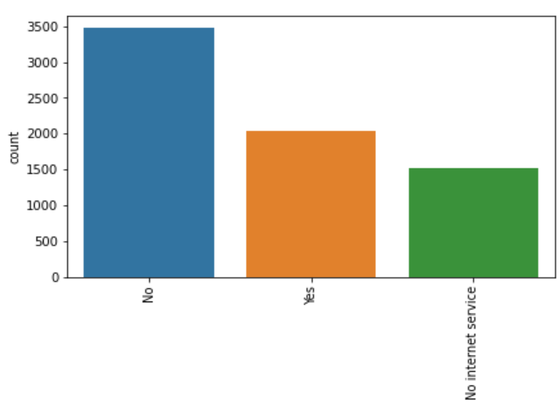
Churn
No
Yes
count
Yes
No
No internet service
OnlineBackup



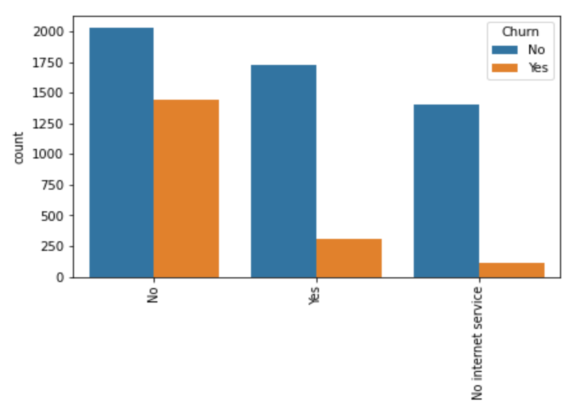
count
No
Yes
No internet service
DeviceProtection



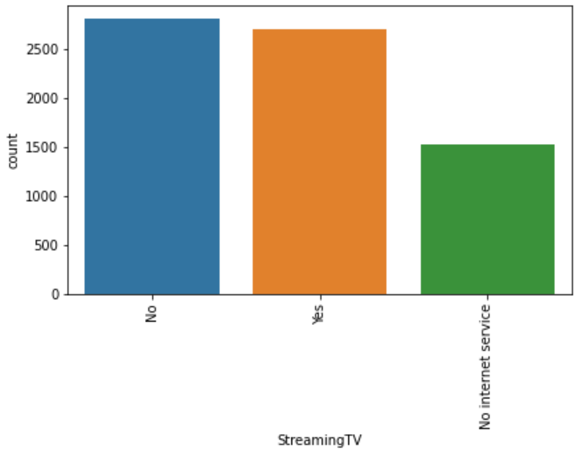
Churn
No
Yes
count
No
Yes
No internet service
DeviceProtection

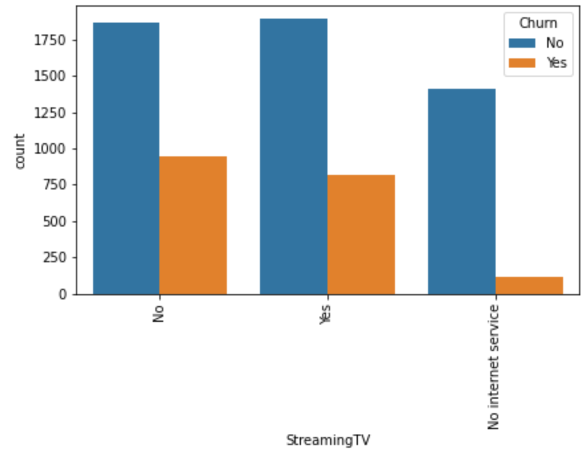


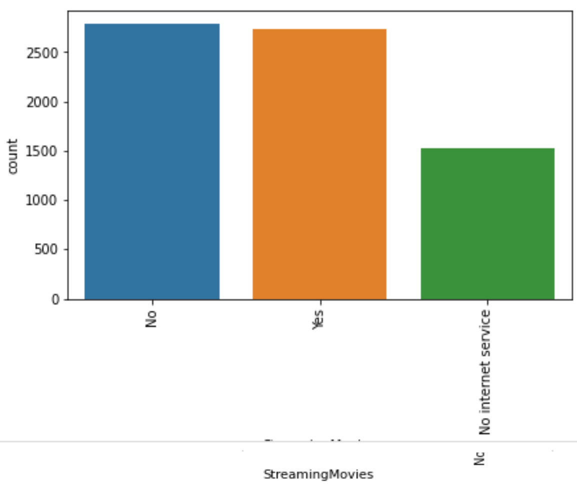
count
No
Yes
No internet service
TechSupport

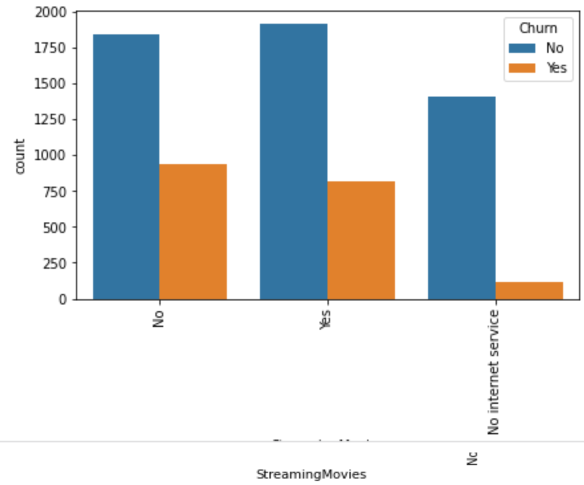


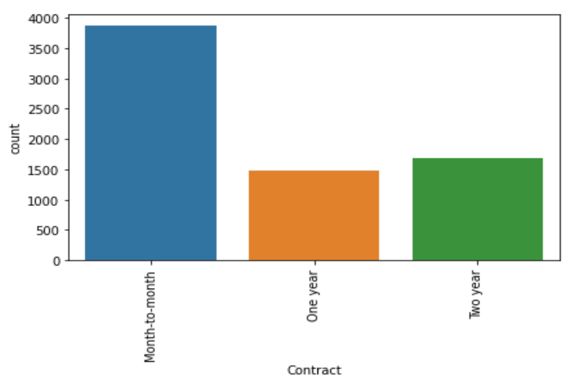
Churn
No
Yes
count
No
Yes
No internet service
TechSupport

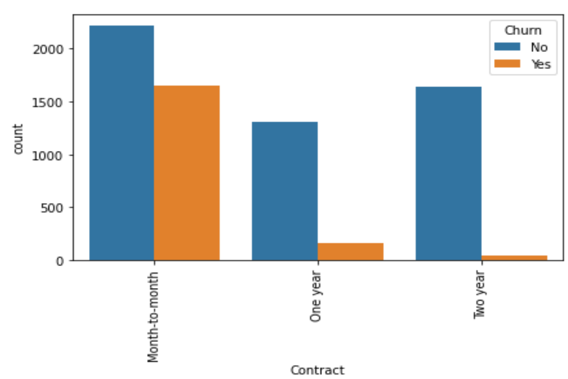

count
No
Yes
No internet service
StreamingTV

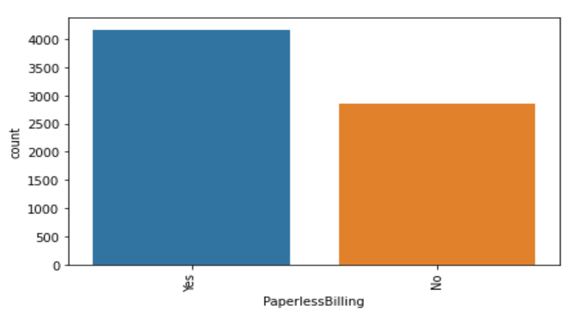

Churn
No
Yes
count
No
Yes
No internet service
StreamingTV

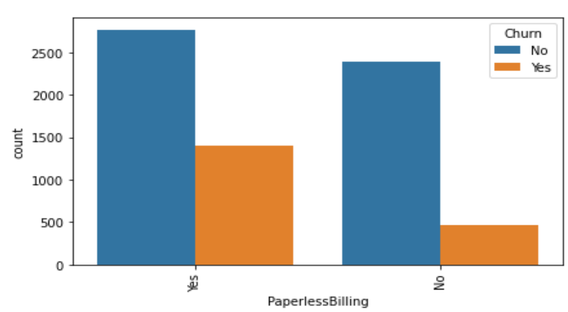

count
No
Yes
No internet service
StreamingMovies

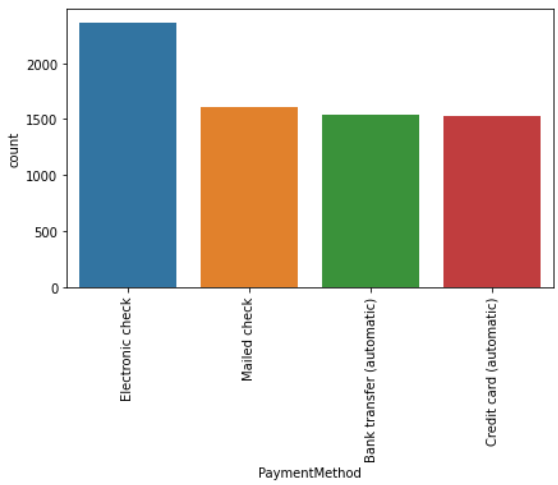

Churn
No
Yes
count
No
Yes
No internet service
StreamingMovies

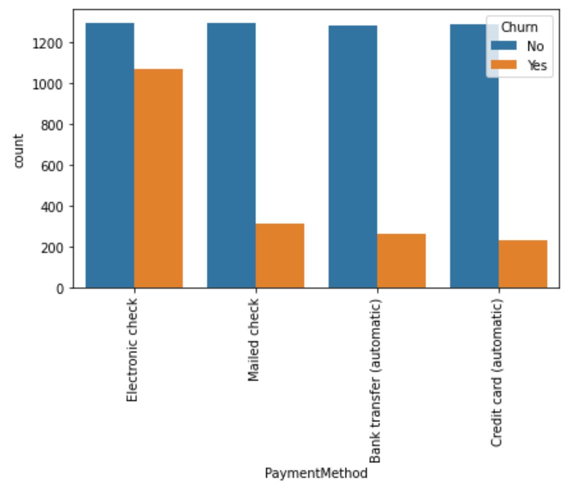

count
Month-to-month
One year
Two year
Contract

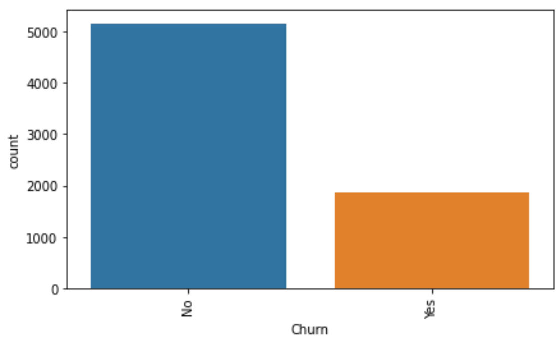

Churn
No
Yes
count
Month-to-month
One year
Two year
Contract

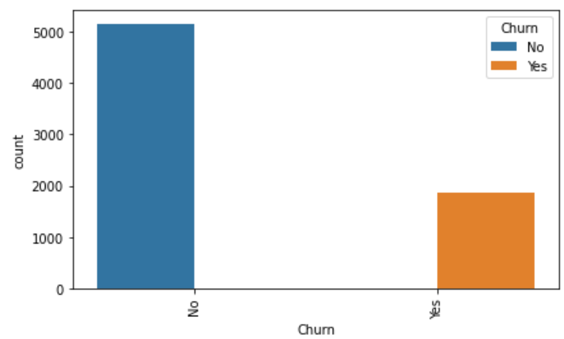

count
Yes
No
PaperlessBilling

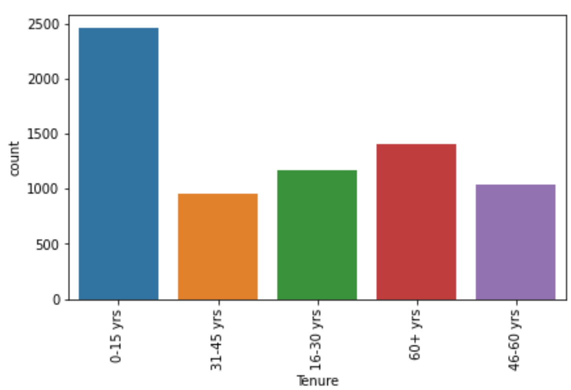

Churn
No
Yes
count
Yes
No
PaperlessBilling

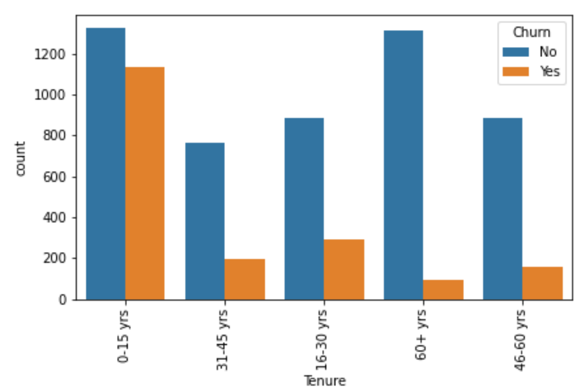

count
0
500
1000
1500
2000
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod

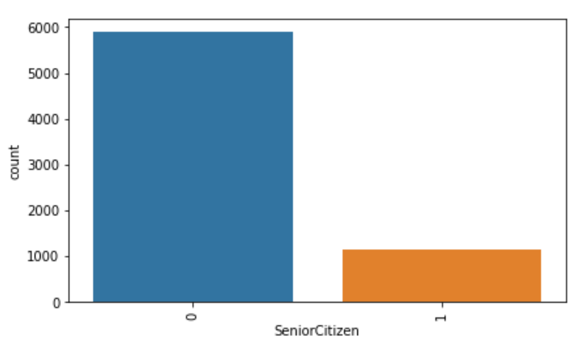

count
0
200
400
600
800
1000
1200
Churn
No
Yes
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod

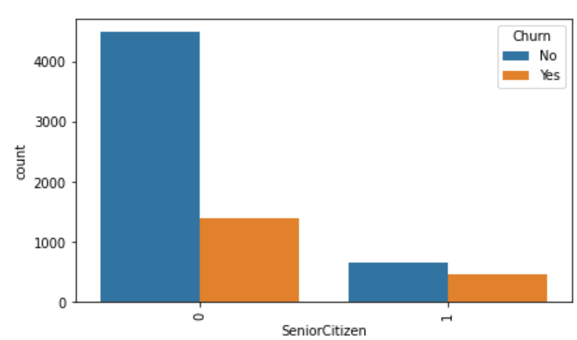

count
0
1000
2000
3000
4000
5000
No
Yes
Churn


count
0
1000
2000
3000
4000
5000
Churn
No
Yes
No
Yes
Churn


count
0
500
1000
1500
2000
2500
0-15 yrs
31-45 yrs
16-30 yrs
60+ yrs
46-60 yrs
Tenure


count
0
200
400
600
800
1000
1200
Churn
No
Yes
0-15 yrs
31-45 yrs
16-30 yrs
60+ yrs
46-60 yrs
Tenure


count
0
1000
2000
3000
4000
5000
6000
0
1
SeniorCitizen


count
0
1000
2000
3000
4000
Churn
No
Yes
0
1
SeniorCitizen

KEY OBSERVATION FOR FEATURE COUNT PLOT:

Gender:
 # for both gender, value seems almost same, therefore we can say, gender not impacting much to churn

OnlineBackup:
 # Due online Backup issue many customer have churn

Device protection
 # may be due to mobile protection, people churn the telecome, therefore company needs to work on it

Patner column:
 # We can say if person is not having partner, then he is churn

Dependent column
 # majority for population do not churn if he is not having dependents

Phone service
 # Majority of population who churn, basically having phone services, need to improve phone service

MultipleLines column
 # Very less people churn if person is not having phone service

Internet Service
 # Telcom compnay need to work highly on fiber optice, by seeing it we can say, due fiber optice problem many person have churn

Online Security
 # Majority of customer have churn due to internet security factor

TechSupport
 # As we can see many of custerm have churn due to No tech support, need to work on it.

Streaming TV and Streamiing Movies:

this is also leading for churn

Contract column:
Majority of customer have churn, who are having month to month contract, need to find what problem they facing ,
and how can be fixed that

Paper Less billing *is also leading for churn*

ElectronicMethod *is main reason out of remaining, for churn the telecom*
Churn column, data is not balanced, need to balance it

Tenure column:
customer having age more than 60 they are more loyal to the company as its churn rate is least as compare to other
age group class

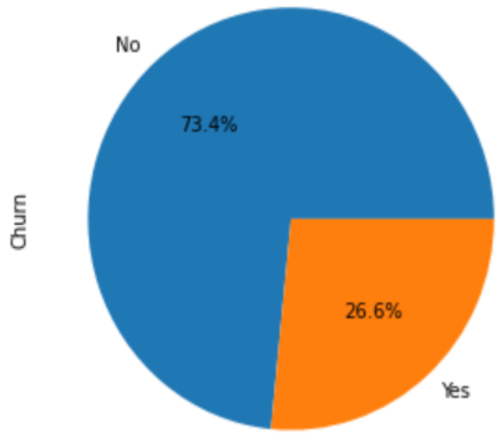
SeniorCitizen*:*
When seniorCitizen is 0 then its churn rate less as compare to 1

CODE FOR CREATING PIE CHART:

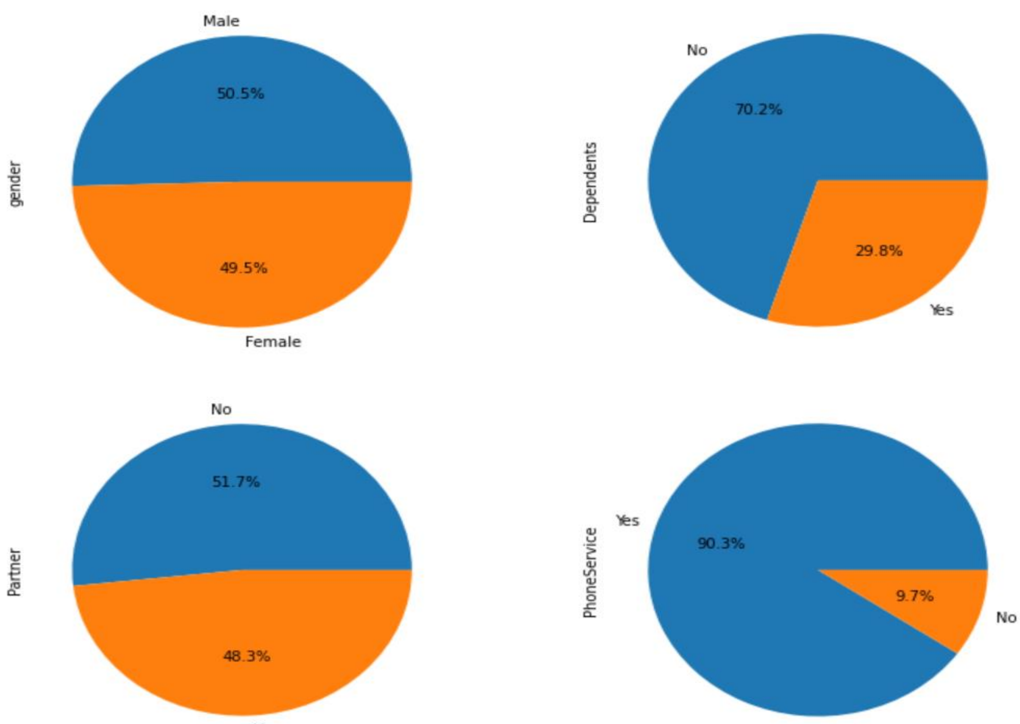
```
# We can see percentage of value counts object type columns
for i in object_col:
    plt.figure(figsize = (5, 5))
    df[i].value_counts().plot(kind = 'pie', autopct = '%.1f%%')
```

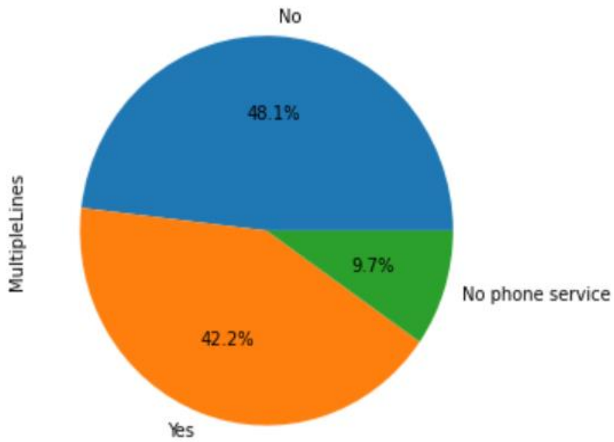
PERCENTAGE OF VALUES OF COLUMN:

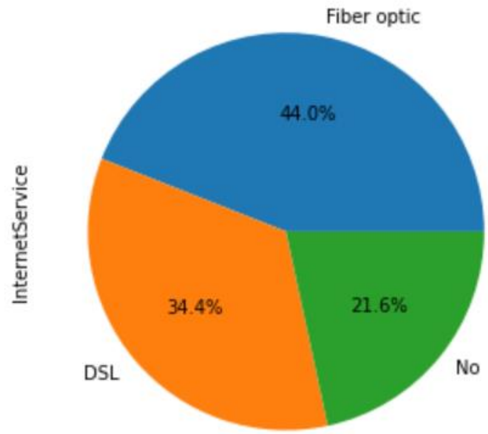
For Target Variable:

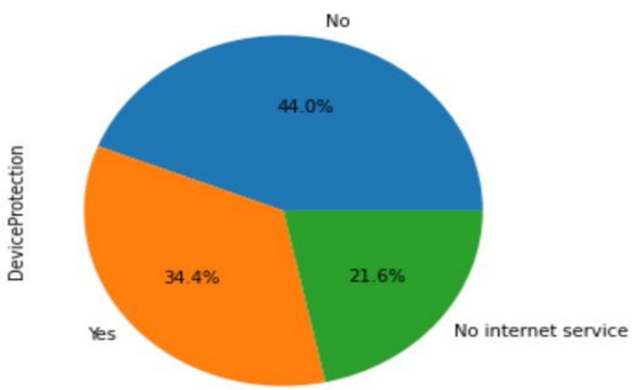


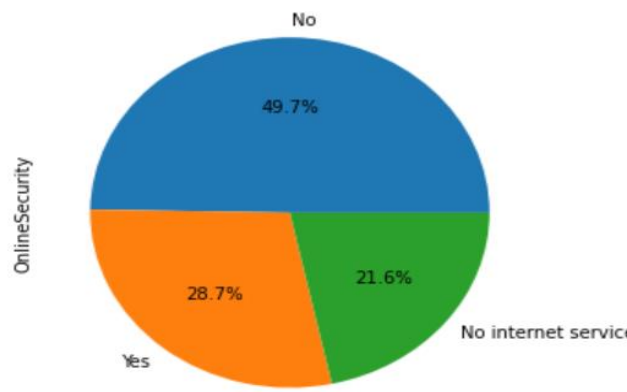
For feature categorical columns:

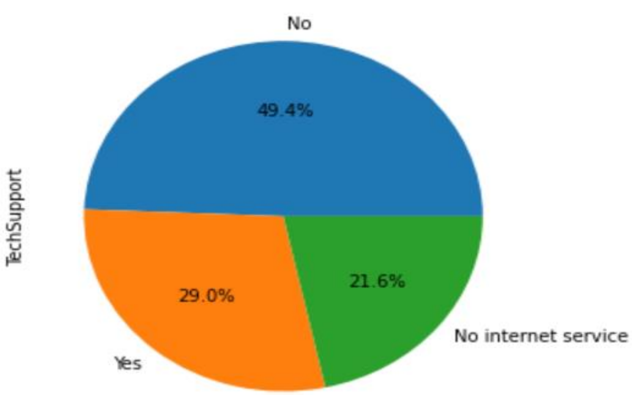


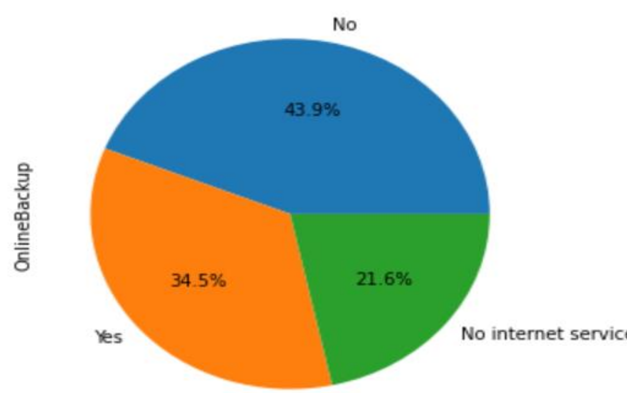

No
48.1%
MultipleLines
9.7%
No phone service
42.2%
Yes

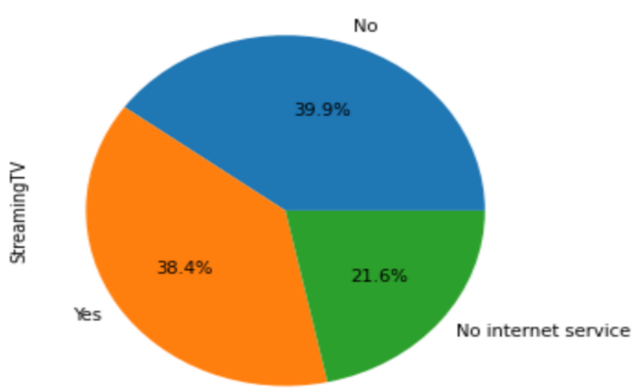

Fiber optic
44.0%
InternetService
34.4%
21.6%
DSL
No

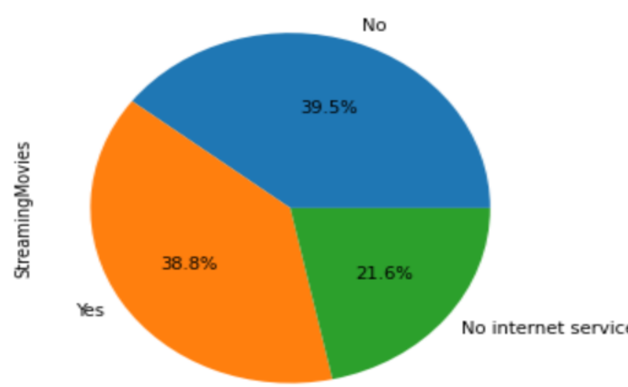

No
44.0%
DeviceProtection
34.4%
21.6%
Yes
No internet service

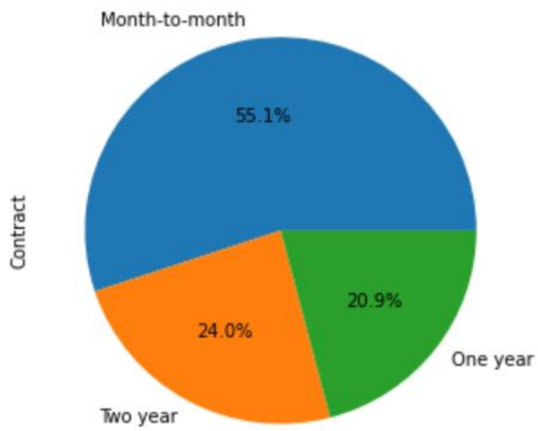

No
49.7%
OnlineSecurity
28.7%
21.6%
Yes
No internet service

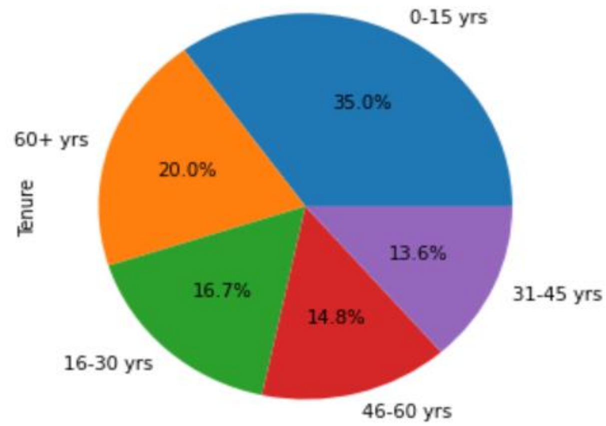

No
49.4%
TechSupport
29.0%
21.6%
Yes
No internet service

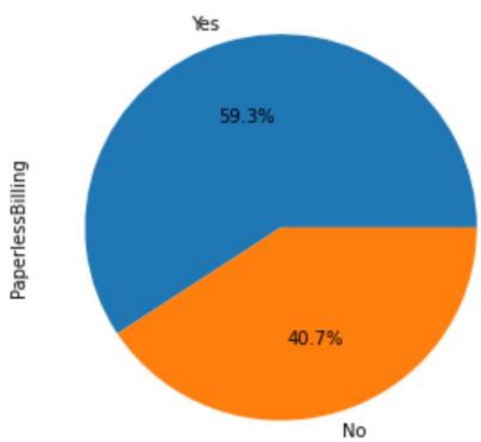

No
43.9%
OnlineBackup
34.5%
21.6%
Yes
No internet service

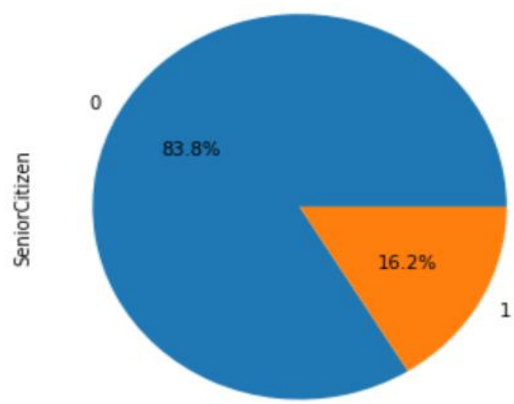

No
39.9%
StreamingTV
38.4%
21.6%
Yes
No internet service

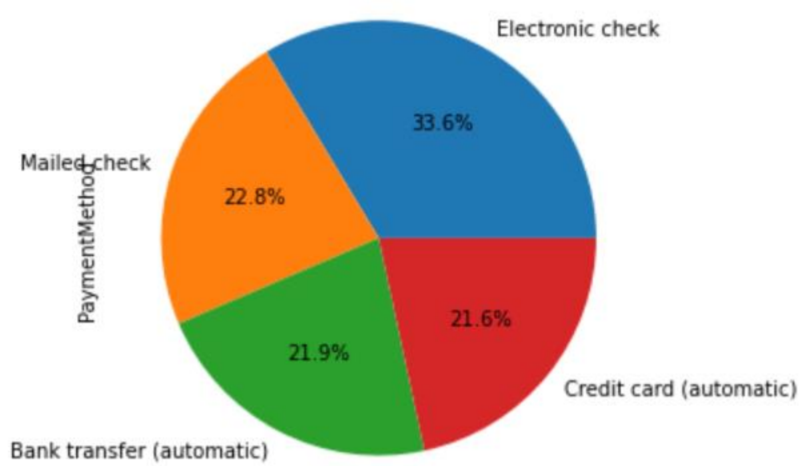

No
39.5%
StreamingMovies
38.8%
21.6%
Yes
No internet service


Month-to-month
55.1%
Contract
24.0%
20.9%
One year
Two year


0-15 yrs
35.0%
60+ yrs
20.0%
Tenure
13.6%
31-45 yrs
16.7%
14.8%
16-30 yrs
46-60 yrs

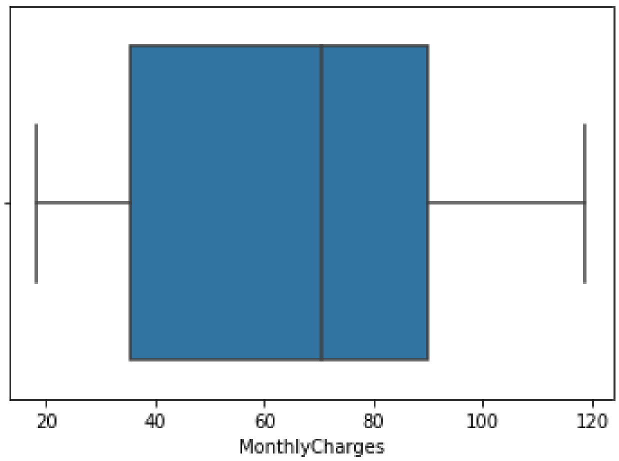

Yes
59.3%
PaperlessBilling
40.7%
No

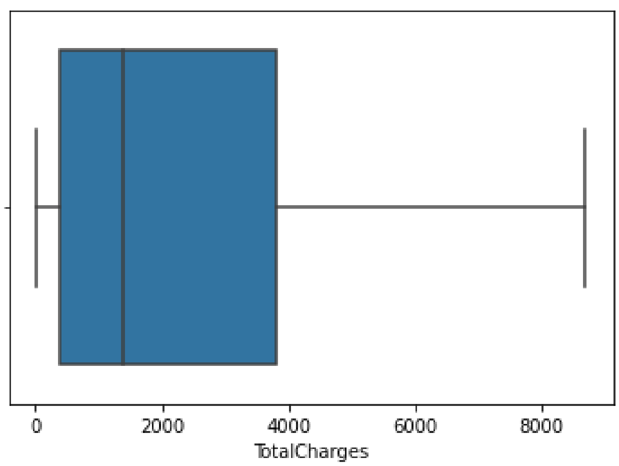

0
83.8%
SeniorCitizen
16.2%
1


Electronic check
33.6%
Mailed check
22.8%
PaymentMethod
21.6%
21.9%
Credit card (automatic)
Bank transfer (automatic)

CHECKING OUTLIERS:

```
for i in numeric_col:
    plt.figure()
    sns.boxplot(df[i])  # Checking outlier for numeric columns
```

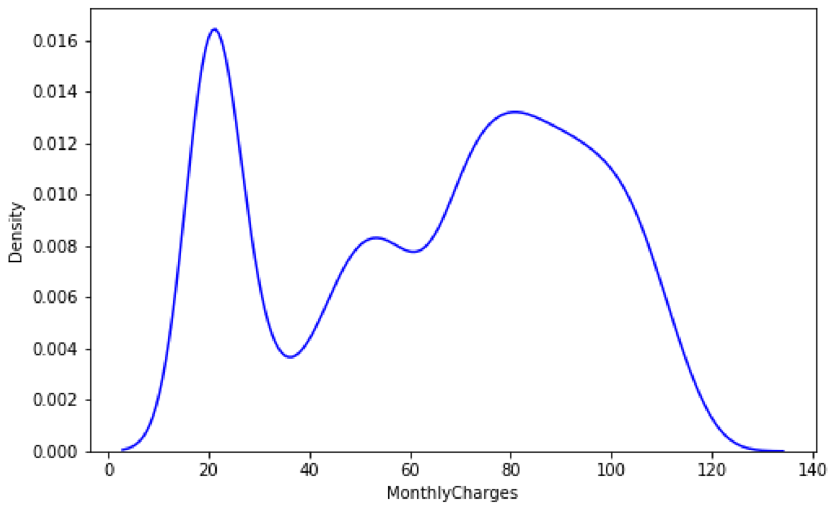


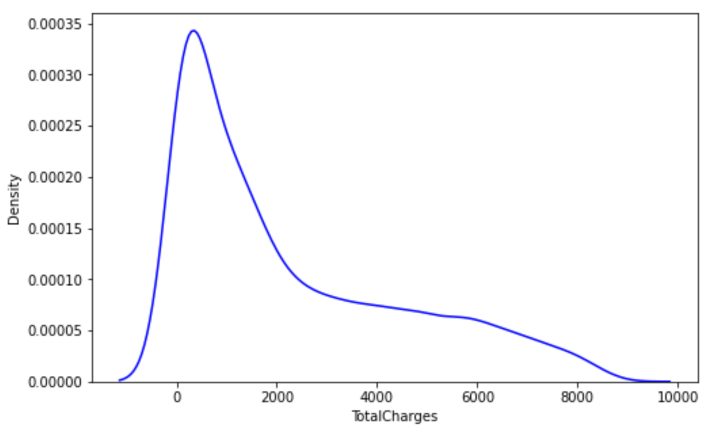



```
# MonthlyCharges and Total Charges columns not having any outliers,
# It seems good that no outlier present
```

CHECKING SKEWNESS OF THE COLUMNS

```
for i in numeric_col:
    plt.figure(figsize = (8,5))
    sns.distplot(df[i], color = 'b', hist = False)   # checking skewness of numeric columns
```

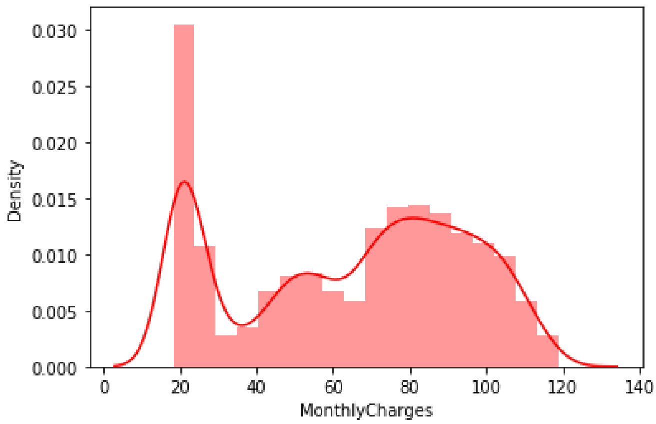


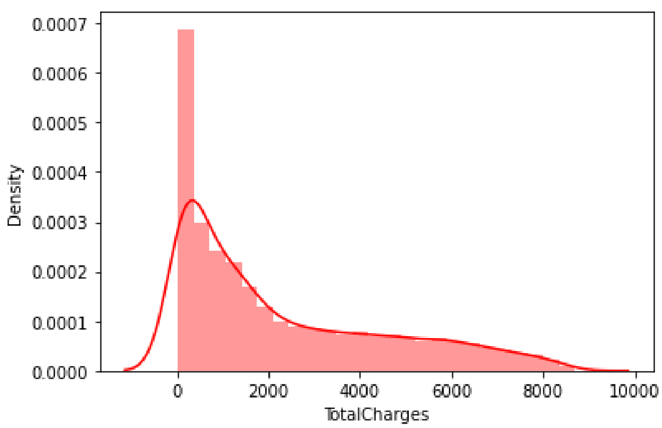



```
# Both columns are showing skewness, need to check its skewness value and need to apply operations accordingly
# TotalCharges is right skewed
# MonthlyCharges is left skewed
```

CHECKING DISTRIBUTION OF THE COLUMNS

```
for i in numeric_col:
    plt.figure()
    sns.distplot(df[i], kde = True, color = 'r')  # Checking distribution of numeric columns
```

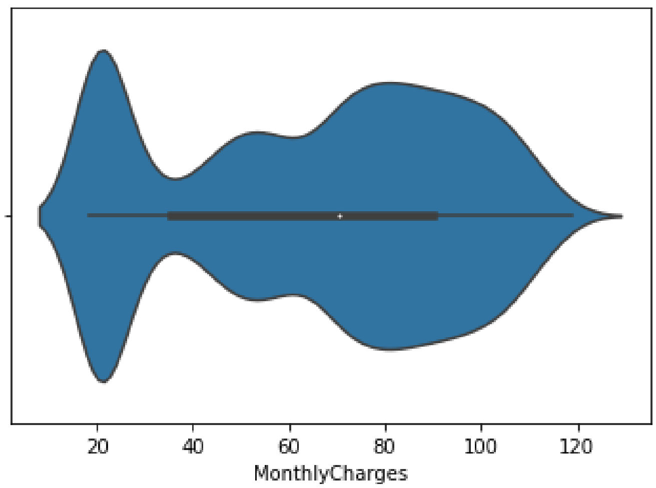


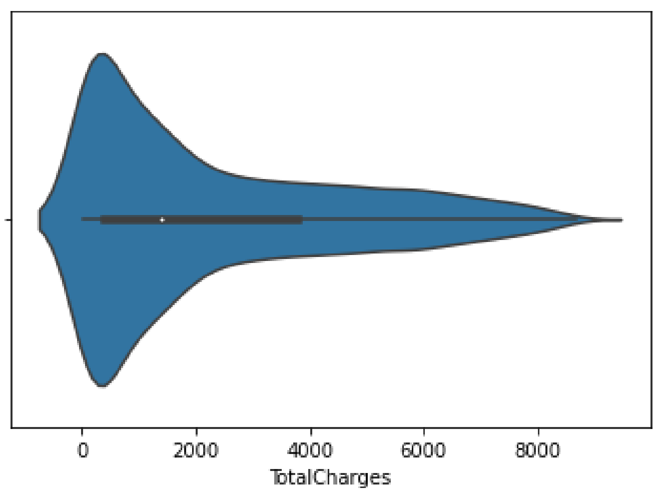



```
# MonthlyCharges columnm having minimum value at 20 and maxium value at 120 approx
# and TotalCharges column having minimum value at 0 and maxium value at near 8000
# Need to perform skewness remvoing techniques to change these columns into normally distributed columns
```

SPREAD OF COLUMNS:

```
for i in numeric_col:
    plt.figure()
    sns.violinplot(df[i], orient = 'vertical')  # Checking spread of numeric columns
```

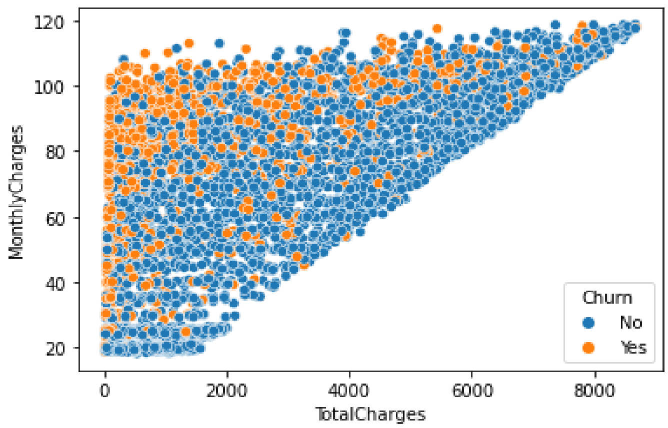




KNOWING THE PATTERN OF DATA:

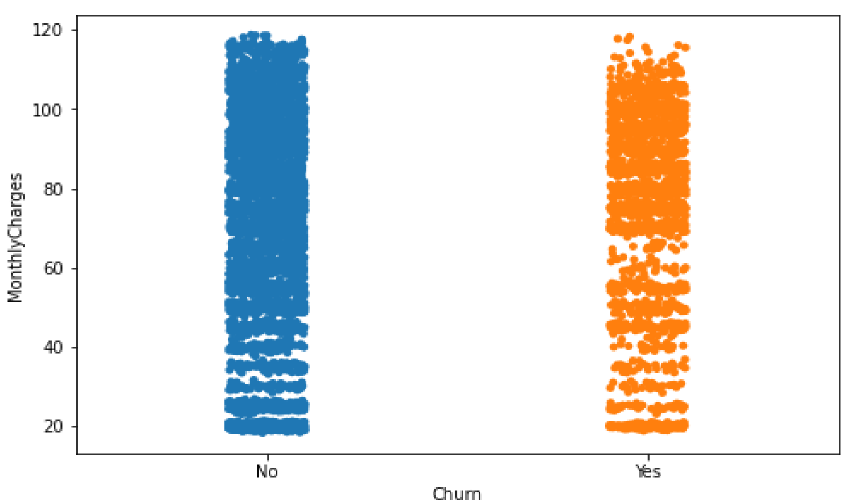
```
plt.figure()
sns.scatterplot(df['TotalCharges'], df['MonthlyCharges'], hue = df['Churn'])
```

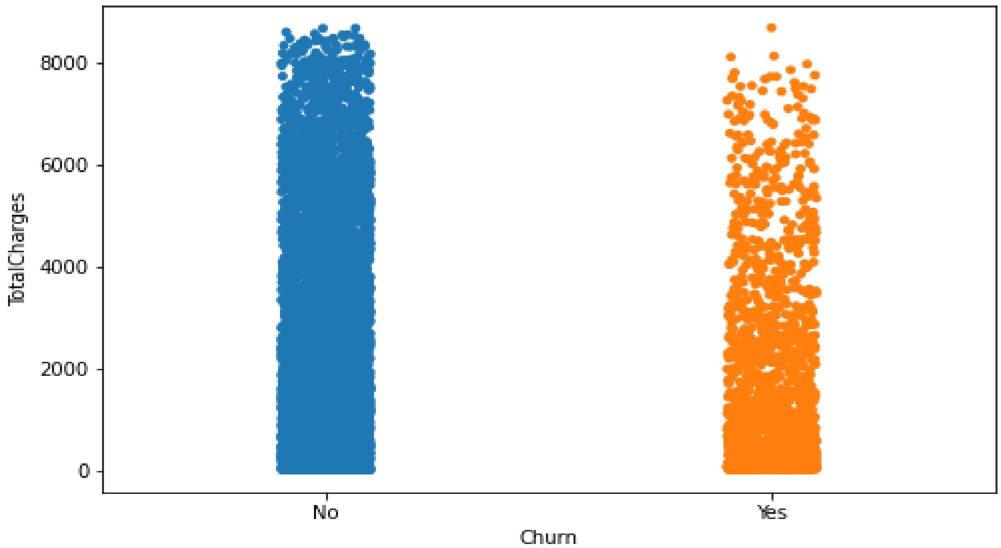
<AxesSubplot:xlabel='TotalCharges', ylabel='MonthlyCharges'>




```
# Monthly charges are increasing as totalcharges are increasing
```

```
for i in numeric_col:
    plt.figure(figsize = (8, 5))
    ax = sns.stripplot(df['Churn'], df[i])
    ax.set(xlabel = 'Churn', ylabel = i)
```

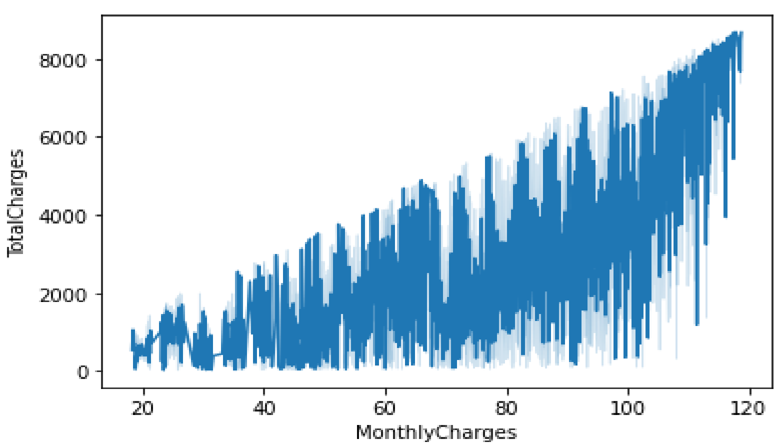





```
# MonthlyCharges with Churn:
# When monthly charges increase then customer churn take place in high rate

# TotalCharges
# we can see churn is decreasing when TotalCharges is decreasing
```

```
plt.figure()
sns.lineplot(x='MonthlyCharges', y='TotalCharges', data=df)
```

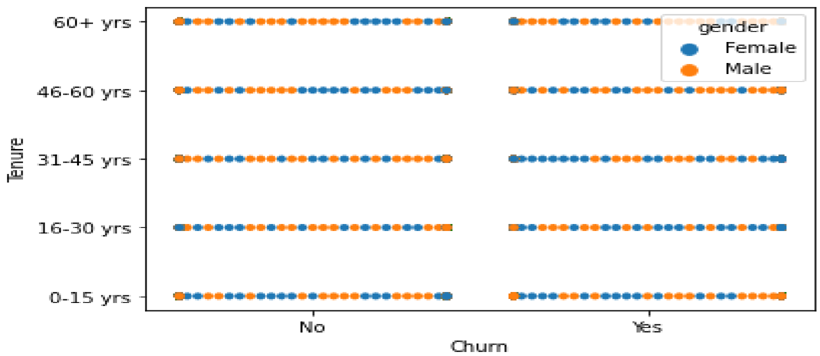
```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='TotalCharges'>
```




```
# totalcharges increases with monthlycharges
```

```
plt.figure()
sns.swarmplot(df['Churn'], df['Tenure'], hue = df['gender'])
```

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



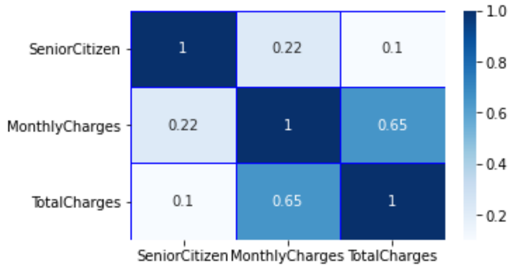
CORRELATION OF COLUMNS:

```
df.corr()
```

	SeniorCitizen	MonthlyCharges	TotalCharges
SeniorCitizen	1.000000	0.219874	0.102411
MonthlyCharges	0.219874	1.000000	0.651065
TotalCharges	0.102411	0.651065	1.000000

```
plt.figure(figsize = (5, 3))
sns.heatmap(df.corr(), annot = True, cmap = 'Blues', linewidth = 0.5, linecolor = 'blue' )    # Heatmap with number,
```

<AxesSubplot:>




```
# MonthlyCharges and TotalCharges columns are correlation
```

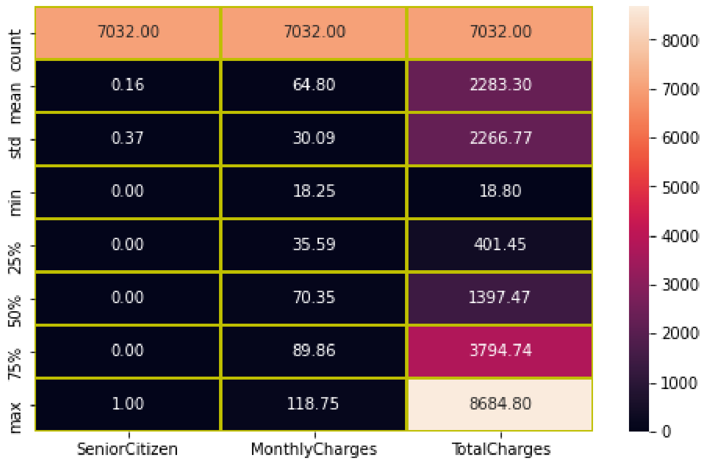
DESCRIBE DATASET:

```
df.describe()
```

	SeniorCitizen	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000
mean	0.162400	64.798208	2283.300441
std	0.368844	30.085974	2266.771362
min	0.000000	18.250000	18.800000
25%	0.000000	35.587500	401.450000
50%	0.000000	70.350000	1397.475000
75%	0.000000	89.862500	3794.737500
max	1.000000	118.750000	8684.800000

```python
plt.figure(figsize = (8, 5))
sns.heatmap(df.describe(), annot = True, linewidth = 0.05, linecolor = 'y', fmt = "0.2f")
```

<AxesSubplot:>




```python
# As we can see in MonthlyCharges columns we are getting some difference between mean and 50% percentile
# TotalColumn is also showing difference between mean and 50 percentile
```

ENCODING OPERATION:

```python
df.Churn.value_counts()
```

```
No     5163
Yes    1869
Name: Churn, dtype: int64
```

```python
for i in object_col:
    print(i,'column having ',df[i].nunique(),'values')
```

```
gender column having  2 values
Partner column having  2 values
Dependents column having  2 values
PhoneService column having  2 values
MultipleLines column having  3 values
InternetService column having  3 values
OnlineSecurity column having  3 values
OnlineBackup column having  3 values
DeviceProtection column having  3 values
TechSupport column having  3 values
StreamingTV column having  3 values
StreamingMovies column having  3 values
Contract column having  3 values
PaperlessBilling column having  2 values
PaymentMethod column having  4 values
Churn column having  2 values
Tenure column having  5 values
SeniorCitizen column having  2 values
```

LABEL ENCODING TO THE TARGET VARIABLE:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
encoded_df = df.copy()
```

```
encoded_df['Churn'] = le.fit_transform(encoded_df['Churn'])
# Applying encoding to the target variable
```

```
encoded_df.Churn.unique()
# array([0, 1])
# No: 0
# Yes: 1
```

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

ONE HOT ENCODING FOR FEATURE COLUMNS:

```
# print(object_col)
encode_list = ['SeniorCitizen', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService',
               'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
                'Contract', 'PaperlessBilling', 'PaymentMethod', 'Tenure', ]
# encode_list
```

Applying OneHotEncoding to categorical features

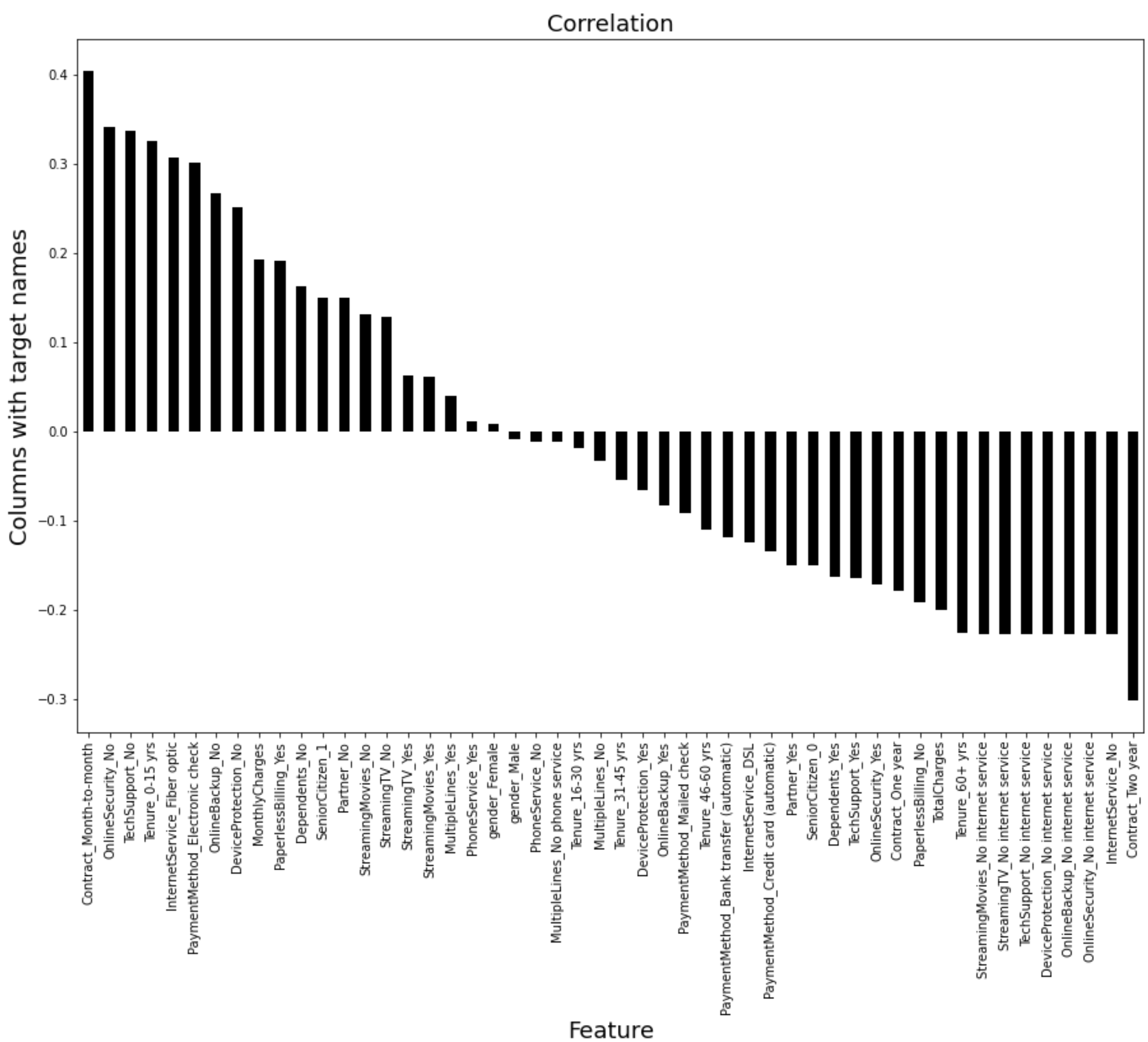
```
encoded_df = pd.get_dummies(encoded_df, columns =  encode_list)
# Applied OneHotEncoding to the Object type variable
```

```
encoded_df.shape
```

```
(7032, 51)
```

IMPACT OF FEATURES ON TARGET:

```
plt.figure(figsize = (15, 10))
encoded_df.corr()['Churn'].sort_values(ascending = False).drop(['Churn']).plot(kind = 'bar', color = 'black')
plt.xlabel( 'Feature', fontsize = 18)
plt.ylabel( 'Columns with target names', fontsize = 18)
plt.title ('Correlation', fontsize = 18)
plt.show()
```



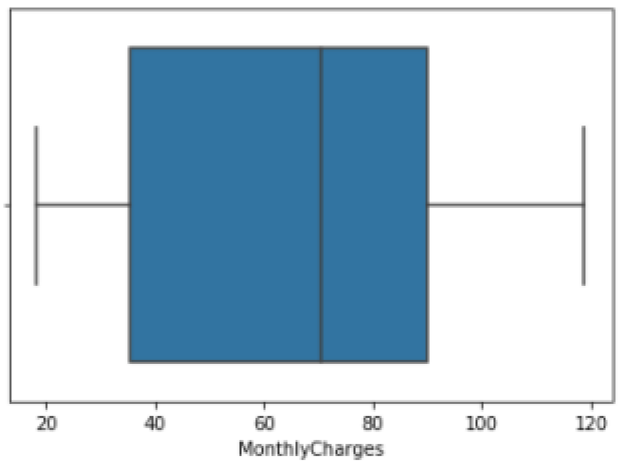
REMOVINF OUTLIERS:

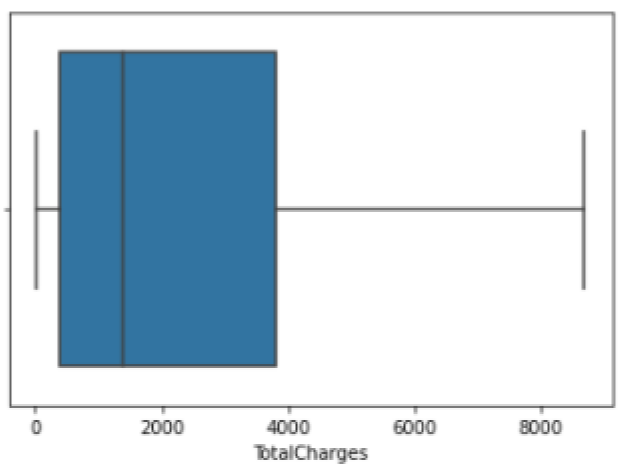
using zscore technique

```
from scipy.stats import zscore
```

```
# No need to apply outlier removing technique as numeric column having no outliers
```

```
for i in numeric_col:
    plt.figure()
    sns.boxplot(encoded_df[i])  # Checking outlier for numeric columns
```





SEOERATING DATASET INTO X AND Y FORM:

```
x = encoded_df.drop(columns = ['Churn'])
y = encoded_df['Churn']

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

# (7032, 50)
# (7032,)
```

```
(7032, 50)
(7032,)
```

```
y.unique()

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

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

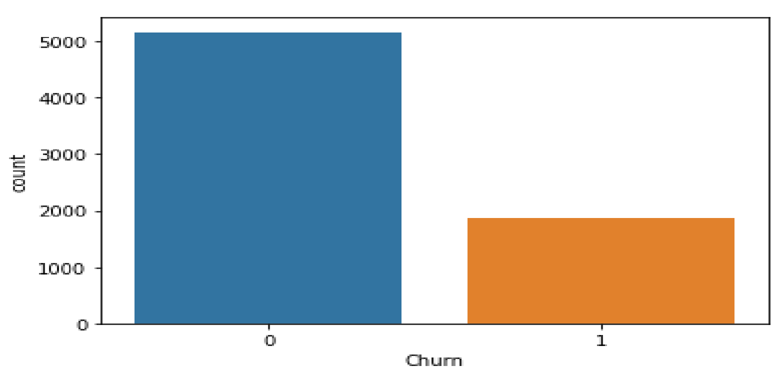
SMOTE Technique to balance to dataset:

```
y.value_counts()

# 0     5163
# 1     1869
```

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

```
sns.countplot(y)
```

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



```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

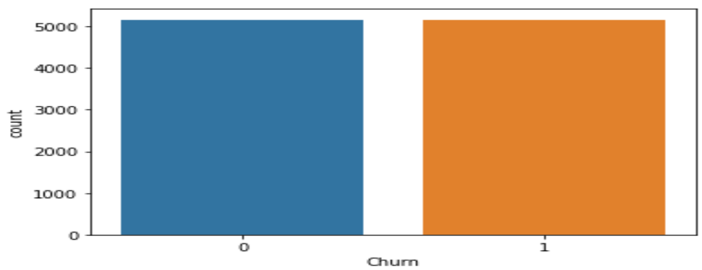
```
x, y = smote.fit_resample(x,y)
```

```
y.value_counts()

# 0     5163
# 1     5163
```

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

```
sns.countplot(y)
```

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




```
# We apply up sample because if we apply down sampling , then it may leads to remove important record and will impact on
# model performace
```

REMOVE SKEWNESS:

```
numeric_col

# ['MonthlyCharges', 'TotalCharges']
```

```
['MonthlyCharges', 'TotalCharges']
```

```
x[numeric_col].skew()

# MonthlyCharges    -0.422146
# TotalCharges       1.095919
```

```
MonthlyCharges    -0.422146
TotalCharges       1.095919
dtype: float64
```

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
```

```
x[numeric_col] = pt.fit_transform(x[numeric_col])
# applying skewness removing techniques to skewed columns of dataset
```

```
x[numeric_col].skew()

# MonthlyCharges    -0.308320
# TotalCharges      -0.126454
```

```
MonthlyCharges    -0.308320
TotalCharges      -0.126454
dtype: float64
```

```
# Both numeric columns is showing skewness value which is in acceptable range, Now, we can move ahead
```

REMOVING MULTICOLLINEARITY:

Using VIF Technique

```
# using VIF Technique
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# function to calculate VIF
def cal_vif(data):
    vif = pd.DataFrame()
    vif['Columns Name'] = data.columns
    vif['VIF'] = [variance_inflation_factor(data.values, i) for i in range(data.shape[1])]
    return (vif)
```

```
cal_vif(x[numeric_col])

# Columns Name  VIF
# 0     MonthlyCharges  1.516051
# 1     TotalCharges    1.516051

# Both columns vif value is in acceptable range
```

	Columns Name	VIF
0	MonthlyCharges	1.516051
1	TotalCharges	1.516051

STANDARD SCALING:

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()   # Instance of Standard Scaler
```

```
x[numeric_col] = ss.fit_transform(x[numeric_col])  # applying standard scaling only to the numeric_col
```

```
x.head()  # Top 5 rows of train dataset
```

	MonthlyCharges	TotalCharges	SeniorCitizen_0	SeniorCitizen_1	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Dependents_Yes	PhoneService_No	Phon
0	-1.326217	-1.741608	1	0	1	0	0	1	1	0	1	
1	-0.442628	0.390081	1	0	0	1	1	0	1	0	0	
2	-0.549135	-1.267488	1	0	0	1	1	0	1	0	0	
3	-0.934503	0.370030	1	0	0	1	1	0	1	0	1	
4	0.043515	-1.119369	1	0	1	0	1	0	1	0	0	

```
y.unique()

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

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

TRAINING MODEL:

1. LogisticRegression:

```
train(LogisticRegression, x, y, 62)
```

```
Training accuracy is :  0.8616491422246818
Testing accuracy is : 0.8615235635894125
_____________________________________________________________________
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.89      0.86      1531
           1       0.88      0.84      0.86      1567

    accuracy                           0.86      3098
   macro avg       0.86      0.86      0.86      3098
weighted avg       0.86      0.86      0.86      3098

Confusion Matrix:
 [[1359  172]
 [ 257 1310]]
______________________________________________________________________
Cross value score
cv score 0.7850087158628705 at 2 cross fold
cv score 0.8022467557621539 at 3 cross fold
cv score 0.8290899263551107 at 4 cross fold
cv score 0.8349953706850683 at 5 cross fold
cv score 0.8403060236296725 at 6 cross fold
cv score 0.84293535961994 at 7 cross fold
cv score 0.8441004059109278 at 8 cross fold
```

2. *AdaBoostClassifier:*

```
train(AdaBoostClassifier, x, y, 52)
```

```
Training accuracy is :  0.8562534587714444
Testing accuracy is : 0.856036152356359
__________________________________________________________________
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.84      0.85      1542
           1       0.84      0.88      0.86      1556

    accuracy                           0.86      3098
   macro avg       0.86      0.86      0.86      3098
weighted avg       0.86      0.86      0.86      3098

Confusion Matrix:
 [[1289  253]
 [ 193 1363]]
___________________________________________________________________
Cross value score
cv score 0.768739105171412 at 2 cross fold
cv score 0.7941119504164246 at 3 cross fold
cv score 0.8313115927001555 at 4 cross fold
cv score 0.8267606749658368 at 5 cross fold
cv score 0.8356575634321132 at 6 cross fold
cv score 0.8367335314999638 at 7 cross fold
cv score 0.8405128078107831 at 8 cross fold
```

3. *GradientBoostingClassifier:*

```
train(GradientBoostingClassifier, x, y, 99)
```

```
Training accuracy is :  0.871333702268954
Testing accuracy is : 0.8705616526791479
__________________________________________________________________
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.87      0.87      1541
           1       0.87      0.87      0.87      1557

    accuracy                           0.87      3098
   macro avg       0.87      0.87      0.87      3098
weighted avg       0.87      0.87      0.87      3098

Confusion Matrix:
 [[1345  196]
 [ 205 1352]]
___________________________________________________________________
Cross value score
cv score 0.7865582025953903 at 2 cross fold
cv score 0.8030214991284138 at 3 cross fold
cv score 0.8320895623172495 at 4 cross fold
cv score 0.8383830447531697 at 5 cross fold
cv score 0.8431144683323649 at 6 cross fold
cv score 0.847388137561763 at 7 cross fold
cv score 0.8483596184677462 at 8 cross fold
```

4. *RandomForestClassifier*:

```
train(RandomForestClassifier, x, y, 99)
```

```
Training accuracy is :  0.9988931931377975
Testing accuracy is : 0.8570045190445449
_________________________________________________________________
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1541
           1       0.87      0.84      0.85      1557

    accuracy                           0.86      3098
   macro avg       0.86      0.86      0.86      3098
weighted avg       0.86      0.86      0.86      3098

Confusion Matrix:
 [[1352  189]
 [ 254 1303]]
__________________________________________________________________
Cross value score
cv score 0.7959519659112919 at 2 cross fold
cv score 0.8121247336819678 at 3 cross fold
cv score 0.8341246705127232 at 4 cross fold
cv score 0.8374163031580132 at 5 cross fold
cv score 0.842727096649235 at 6 cross fold
cv score 0.843903751386182 at 7 cross fold
cv score 0.847006031620221 at 8 cross fold
```

OBSERVATION FOUND OF MACHINE LEARNING MODELS:

```
# # Observation

# Model                         Train Accuracy      Test Accuracy         CV                  Difference
# LogisticRegression                0.86164914          0.86152356            0.8441004           0.01742316
# AdaBoostClassifier                0.85625345          0.85603615            0.8405128           0.01552335
# GradientBoostingClassifier        0.8713337           0.8705616             0.8483596           0.022202
# KNeighborsClassifier              0.99889319          0.857004519         0.847006031         0.009998488
```

```
# In Above observation:
# LogisticRegression model is giving very close cv value to accuracy of model and also not giving overfitted or underfitted
# model

# AdaBoostClassifier is also performing good, as it is giving least difference between cv and accuracy of model

# GradientBoosting is also givng good model which its cv and accuracy difference is greater than Logistic and AdaBoost

# KNeighbors is showing least difference of cv and accuracy but giving overfitted model

# As per observation: AdaBoosting is givng least cv difference and also not giving overfitted model or underfitted model
# hence we will apply ensemble technique to this model for hyper parameter tunning
```

Hyper Parameter Tuning for AdaBoostClassifier:

```
from sklearn.model_selection import GridSearchCV
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, random_state = 52)
```

```
parameter = {'algorithm' : ['SAMME', 'SAMME.R'],
              'learning_rate' : [0.1, 0.01, 1.0],
             'n_estimators': [100, 50, 10],
             'base_estimator' : [LogisticRegression(), None]}
```

```
gcv = GridSearchCV(estimator = AdaBoostClassifier(), param_grid = parameter, cv = 8)
gcv.fit(x_train, y_train)
```

```
GridSearchCV(cv=8, estimator=AdaBoostClassifier(),
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'base_estimator': [LogisticRegression(), None],
                         'learning_rate': [0.1, 0.01, 1.0],
                         'n_estimators': [100, 50, 10]})
```

```
gcv.best_params_
```

```
{'algorithm': 'SAMME.R',
 'base_estimator': LogisticRegression(),
 'learning_rate': 1.0,
 'n_estimators': 100}
```

Final Model of AdaBoostClassifier:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, random_state = 129)
final_model = AdaBoostClassifier(algorithm= 'SAMME.R', base_estimator= LogisticRegression(), learning_rate= 1.0, n_estimators= 100)
final_model.fit(x_train, y_train)
predict_train = final_model.predict(x_train)
predict_test = final_model.predict(x_test)

training = accuracy_score(predict_train, y_train)
testing = accuracy_score(predict_test, y_test)

print('At random state', i, 'the training accuracy is :-', training)
print('At random state', i, 'the testing accuracy is :-', testing)
print('________________________________________________________________')
print('Classification Report: \n', classification_report(y_test, predict_test, ) )
print('Confusion Matrix: \n', confusion_matrix(y_test, predict_test) )
print('_________________________________________________________________')
```

```
At random state 6 the training accuracy is :- 0.8537631433314886
At random state 6 the testing accuracy is :- 0.8741123305358296
________________________________________________________________
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.89      0.88      1569
           1       0.88      0.86      0.87      1529

    accuracy                           0.87      3098
   macro avg       0.87      0.87      0.87      3098
weighted avg       0.87      0.87      0.87      3098

Confusion Matrix:
 [[1391  178]
 [ 212 1317]]
_________________________________________________________________
```

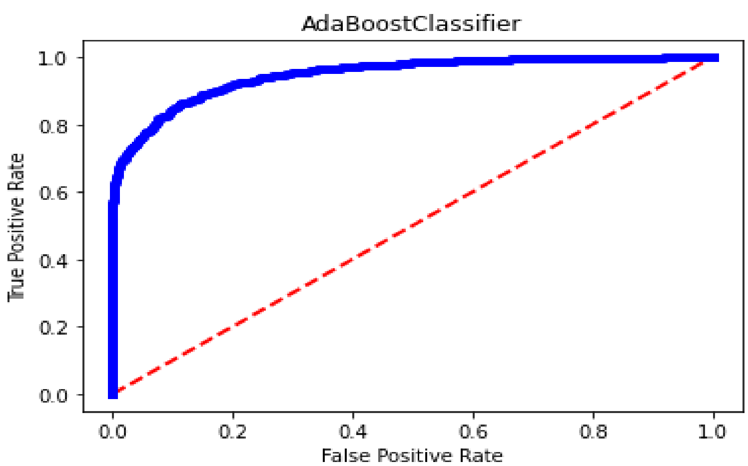
AOC – ROC Curve (Churn Status: Yes):

```
final_pred_prob = final_model.predict_proba( x_test)[:, 1]  #  probability of getting 1

# Yes (1) : Yes Churn
# No (0) : No Churn
```

```
fpr, tpr, threshols = roc_curve(y_test, final_pred_prob)
#  By the use of fpr and tpr we create AUC ROC curve
```

```
# fpr
# tpr
# threshols
```

```
plt.plot([0, 1], [0, 1], 'k--', color = 'red', lw = 2)
plt.plot( fpr, tpr, color = 'b', lw = 5, label = 'ROC Curve')   # graph for AOC ROC curve
plt.xlabel('False Positive Rate')  # x axis
plt.ylabel('True Positive Rate')   # y axis
plt.title('AdaBoostClassifier')  # Title
plt.show()
```




```
# As our model is giving accuracy of 87 % , therefore, curve is not sharp
```

```
auc_score = roc_auc_score(y_test, final_model.predict(x_test))
auc_score

# 0.873949614860519
```

```
0.873949614860519
```

DEPLOY MODEL:

```
import pickle
filename = 'churn_predictor.pkl'                    # model name
pickle.dump(final_model, open(filename, 'wb'))         # operation to deploy model
```

LOADING MODEL:

```
load_model =  pickle.load(open('churn_predictor.pkl', 'rb'))    # Loading deployed model
result = load_model.score(x_test, y_test)
print(result)

# 0.8741123305358296
```

```
0.8741123305358296
```

CONCLUSION:

```
original = np.array(y_test)
predicted = np.array(load_model.predict(x_test))
# convert columns in to np.array
```

```
conclusion = pd.DataFrame({'Actual Churn': original, 'Predicted Churn': predicted}, index = range(len(original)))
# Dataframe creation
```

```
pd.set_option('display.max_rows', None)  # To maximize the rows
conclusion.head()
```

	Actual Churn	Predicted Churn
0	0	0
1	0	0
2	1	1
3	0	0
4	0	0

In this way you can create your machine learning model for predicting churn of a customer,

By using this model one can growth his business, by focusing to overcome from the main reason of churn

Thank for giving time on this page, I hope it helped you!

Submitted by:

Bhushan Kumar Sharma

(Batch 1834)