

Project Title :Telecom Customer Churn Analytics:

Problem Statement:

The objective of this project is to analyze customer churn in a telecom company. Customer churn refers to the phenomenon where customers switch from one service provider to another or cancel their subscription altogether. By analyzing customer churn patterns, we aim to identify the factors that contribute to churn and develop strategies to mitigate it.

Project Description:

In this project, we will work with a dataset from a telecom company that includes information about their customers, such as demographics, customer Accounting information, Service information. The dataset will also include a churn indicator that specifies nether a customer has churned or not.

Desired problem come(Objective or goal)The main objective is to find out the reasons for call drops and voice connectivity Built a classification predictive model to predict call drop

DesiredOutcome:

our main goal is to bulid a computer program that can predict when a customer might leave the company

Algorithms:

LogisticRegression,DecisionTreeClassifier,RandomForestClassifier,AddaboostClassifier,GradientBoostClassifier

About Data

Data is divided into 3 Types

Demographic information:

- 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)
- Dependents : Whether the customer has dependents or not (Yes, No)

Customer Accounting Information:

- Contract: The contract term of the customer (Month-to-month, One year, Two year)
- PaperlessBilling : Whether the customer has paperless billing or not (Yes, No)
- MonthlyCharges: The amount charged to the customer monthly
- TotalCharges: The total amount charged to the customer
- tenure: Number of months the customer has stayed with the company
- PaymentMethod: The customer's payment method (Electronic check, Mailed check, Bank transfer (au card (automatic)))
- customerID: Customer ID

Service information

PhoneService: Whether the customer has a phone service or not (yes, No)

- MultipleLines: 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 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)
- Streaming TV: 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)

Traget variable

- Churn: Whether the customer churn or not (yes or No)*

1. Data Preparation - (EDA & Feature Engineering -Data Analytics)

In [1]:

```
1 #EDA
2 import numpy as np
3 import pandas as pd
4
5 #data visualations
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 %matplotlib inline
```

```
In [2]: 1 import os
        2 os.getcwd()
        3 #os.chdir(r"C:\Users\RAGHAVENDER GOUD\dataminds project")
```

```
Out[2]: 'C:\\Users\\sumit\\Data Science\\Live Project\\Projects end to end\\Customer'
```

```
In [3]: 1 telco_base_data = pd.read_csv('Telco-Customer-Churn.csv')
```

```
In [4]: 1 telco_base_data.head()
```

```
Out[4]:
```

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

5 rows × 21 columns



```
In [5]: 1 telco_base_data['InternetService'].unique()
```

```
Out[5]: array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

```
In [6]: 1 telco_base_data.shape
```

```
Out[6]: (7043, 21)
```

In [7]: 1 telco_base_data.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
```

Knowing the unique values

```
In [8]: 1 for col in telco_base_data.columns:
        2     print("column: {} - Unique Values: {}\n".format(col,telco_base_data[col].unique()))
```

```
column: customerID - Unique Values: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JJAZL' '8361-LTMKD'
'3186-AJIEK']

column: gender - Unique Values: ['Female' 'Male']

column: SeniorCitizen - Unique Values: [0 1]

column: Partner - Unique Values: ['Yes' 'No']

column: Dependents - Unique Values: ['No' 'Yes']

column: tenure - Unique Values: [ 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  0
 39]

column: PhoneService - Unique Values: ['No' 'Yes']

column: MultipleLines - Unique Values: ['No phone service' 'No' 'Yes']

column: InternetService - Unique Values: ['DSL' 'Fiber optic' 'No']

column: OnlineSecurity - Unique Values: ['No' 'Yes' 'No internet service']

column: OnlineBackup - Unique Values: ['Yes' 'No' 'No internet service']

column: DeviceProtection - Unique Values: ['No' 'Yes' 'No internet service']

column: TechSupport - Unique Values: ['No' 'Yes' 'No internet service']

column: StreamingTV - Unique Values: ['No' 'Yes' 'No internet service']

column: StreamingMovies - Unique Values: ['No' 'Yes' 'No internet service']

column: Contract - Unique Values: ['Month-to-month' 'One year' 'Two year']

column: PaperlessBilling - Unique Values: ['Yes' 'No']

column: PaymentMethod - Unique Values: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
'Credit card (automatic)']
```

column: MonthlyCharges - Unique Values: [29.85 56.95 53.85 ... 63.1 44.2 78.7]

column: TotalCharges - Unique Values: ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']

column: Churn - Unique Values: ['No' 'Yes']

```
In [9]: 1 telco_base_data.columns.values
```

```
Out[9]: array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',  
            'tenure', 'PhoneService', 'MultipleLines', 'InternetService',  
            'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',  
            'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',  
            'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',  
            'TotalCharges', 'Churn'], dtype=object)
```

```
In [10]: 1 telco_base_data.TotalCharges = pd.to_numeric(telco_base_data.TotalCharges, errors='coerce')
```



```
In [11]: 1 telco_base_data.dtypes
```

```
Out[11]: customerID      object
gender      object
SeniorCitizen  int64
Partner      object
Dependents    object
tenure      int64
PhoneService  object
MultipleLines object
InternetService object
OnlineSecurity object
OnlineBackup  object
DeviceProtection object
TechSupport   object
StreamingTV   object
StreamingMovies object
Contract      object
PaperlessBilling object
PaymentMethod object
MonthlyCharges float64
TotalCharges  float64
Churn         object
dtype: object
```

```
In [12]: 1 telco_base_data.describe()
```

```
Out[12]:
```

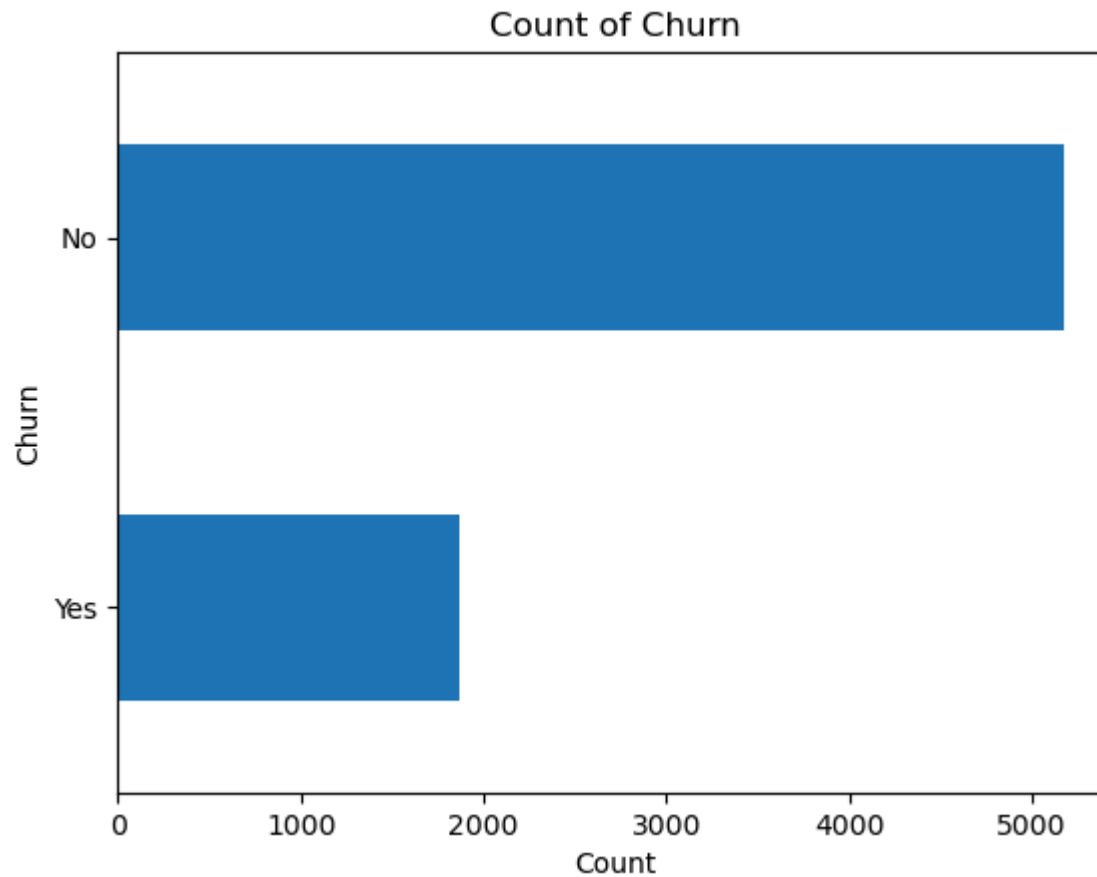
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7032.000000
mean	0.162147	32.371149	64.761692	2283.300441
std	0.368612	24.559481	30.090047	2266.771362
min	0.000000	0.000000	18.250000	18.800000
25%	0.000000	9.000000	35.500000	401.450000
50%	0.000000	29.000000	70.350000	1397.475000
75%	0.000000	55.000000	89.850000	3794.737500
max	1.000000	72.000000	118.750000	8684.800000

SeniorCitizen is actually a categorical hence the 25%-50%-75% distribution is not proper

75% customers have tenure less than 55 months

Average Monthly charges are USD 64.76 whereas 25% customers pay more than USD 89.85 per month

```
In [13]: 1 telco_base_data['Churn'].value_counts().plot(kind='barh')
2 plt.xlabel("Count")
3 plt.ylabel("Churn")
4 plt.title("Count of Churn")
5 plt.gca().invert_yaxis() # Invert y-axis to have 'No Churn' on top
6 plt.show()
7
```



```
In [14]: 1 telco_base_data['Churn'].value_counts()/len(telco_base_data)
```

```
Out[14]: Churn  
No      0.73463  
Yes     0.26537  
Name: count, dtype: float64
```

```
In [15]: 1 telco_base_data['Churn'].value_counts()
```

```
Out[15]: Churn  
No      5174  
Yes     1869  
Name: count, dtype: int64
```

```
In [16]: 1 telco_base_data.info(verbose=True)
```

```
<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          7032 non-null   float64
20  Churn                 7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

```
In [17]: 1 telco_data=telco_base_data.copy()
```

```
In [18]: 1 telco_data.isna().sum()
```

```
Out[18]: 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
```

In [19]: 1 telco_data.loc[telco_data['TotalCharges'].isna()==True]

Out[19]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProt
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No internet service	...	No i ;
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	...	
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No i ;
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No i ;
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No i ;
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No internet service	...	No i ;
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No i ;
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	...	
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	...	

11 rows × 21 columns



```
In [20]: 1 telco_data.dtypes
```

```
Out[20]: customerID      object
gender      object
SeniorCitizen  int64
Partner      object
Dependents    object
tenure      int64
PhoneService  object
MultipleLines object
InternetService object
OnlineSecurity object
OnlineBackup  object
DeviceProtection object
TechSupport   object
StreamingTV   object
StreamingMovies object
Contract      object
PaperlessBilling object
PaymentMethod object
MonthlyCharges float64
TotalCharges  float64
Churn         object
dtype: object
```



```
In [21]: 1 telco_data.isna().sum()/len(telco_data)
```

```
Out[21]: customerID      0.000000  
gender      0.000000  
SeniorCitizen  0.000000  
Partner      0.000000  
Dependents    0.000000  
tenure      0.000000  
PhoneService  0.000000  
MultipleLines  0.000000  
InternetService  0.000000  
OnlineSecurity  0.000000  
OnlineBackup  0.000000  
DeviceProtection  0.000000  
TechSupport  0.000000  
StreamingTV  0.000000  
StreamingMovies  0.000000  
Contract      0.000000  
PaperlessBilling  0.000000  
PaymentMethod  0.000000  
MonthlyCharges  0.000000  
TotalCharges  0.001562  
Churn         0.000000  
dtype: float64
```

4. Missing Value Treatement

Since the % of these records compared to total dataset is very low ie 0.0015%, it is safe to ignore them from further processing.

```
In [22]: 1 #Removing missing values  
2 telco_data.dropna(how = 'any', inplace = True)
```

5. Divide customers into bins based on tenure e.g. for tenure < 12 months: assign a tenure group if 1-12, for tenure between 1 to 2 Yrs, tenure group of 13-24; so on...

```
In [23]: 1 # Get the max tenure
          2 print(telco_data['tenure'].max()) #72
```

72

```
In [24]: 1
          2 # Define the bins and labels
          3 bins = [0, 12, 24, 36, 48, 60, 72]
          4 labels = ['1 - 12', '13 - 24', '25 - 36', '37 - 48', '49 - 60', '61 - 72']
          5
          6 # Create the tenure_group column
          7 telco_data['tenure_group'] = pd.cut(telco_data['tenure'], bins=bins, labels=labels, right=False)
          8
```

```
In [25]: 1 telco_data['tenure_group'].value_counts()
```

```
Out[25]: tenure_group
1 - 12      2058
61 - 72     1121
13 - 24     1047
25 - 36      876
49 - 60      820
37 - 48      748
Name: count, dtype: int64
```

```
In [26]: 1 telco_data['tenure_group'].value_counts()/len(telco_data)
```

```
Out[26]: tenure_group
1 - 12      0.292662
61 - 72     0.159414
13 - 24     0.148891
25 - 36     0.124573
49 - 60     0.116610
37 - 48     0.106371
Name: count, dtype: float64
```

6. Remove columns not required for processing

```
In [27]: 1 #drop column customerID and tenure
2 telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
3 telco_data.head()
```

```
Out[27]:
```

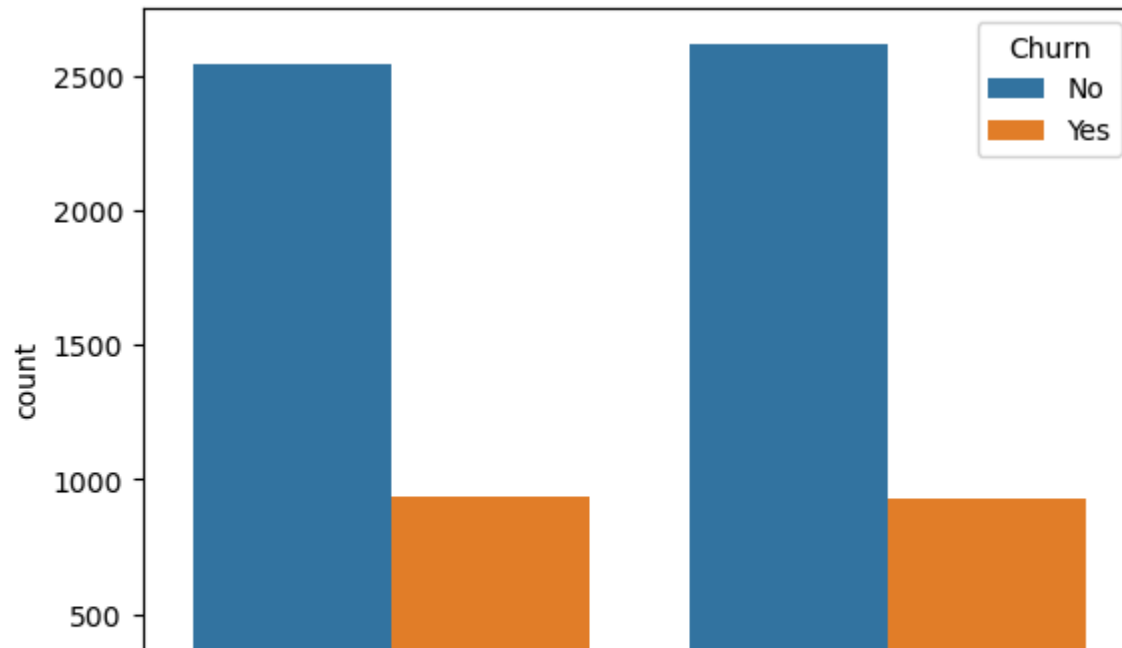
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	Tech
0	Female	0	Yes	No	No	No phone service	DSL	No	Yes	No	
1	Male	0	No	No	Yes	No	DSL	Yes	No	Yes	
2	Male	0	No	No	Yes	No	DSL	Yes	Yes	No	
3	Male	0	No	No	No	No phone service	DSL	Yes	No	Yes	
4	Female	0	No	No	Yes	No	Fiber optic	No	No	No	

Data Exploration

***1. * Plot distribution of individual predictors by churn**

Univariate Analysis

```
In [28]: 1 for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges', 'MonthlyCharges'])):  
2         plt.figure(i)  
3         sns.countplot(data=telco_data, x=predictor, hue='Churn')
```



2. Convert the target variable 'Churn' in a binary numeric variable i.e. Yes=1 ; No = 0

```
In [29]: 1 telco_data['Churn'] = np.where(telco_data.Churn == 'Yes',1,0)
```

In [30]: 1 telco_data.sample(3)

Out[30]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	T
2659	Female	0	No	No	Yes	Yes	DSL	Yes	No	Yes	
4878	Male	0	Yes	No	Yes	Yes	DSL	No	No	No	
1156	Female	0	No	No	Yes	No	No	No internet service	No internet service	No internet service	

In [31]: 1 telco_data.dtypes

Out[31]:

gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	float64
Churn	int32
tenure_group	category
dtype:	object

3. Convert all the categorical variables into dummy variables

```
In [32]: 1 from sklearn.preprocessing import LabelEncoder
        2 le=LabelEncoder()
        3 le
```

Out[32]: LabelEncoder()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [33]: 1 categ=['gender','SeniorCitizen', 'tenure_group' , 'Partner', 'Dependents', 'PhoneService',
        2         'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
        3         'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
        4         'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn',]
        5
        6 telco_data[categ] = telco_data[categ].apply(le.fit_transform)
```

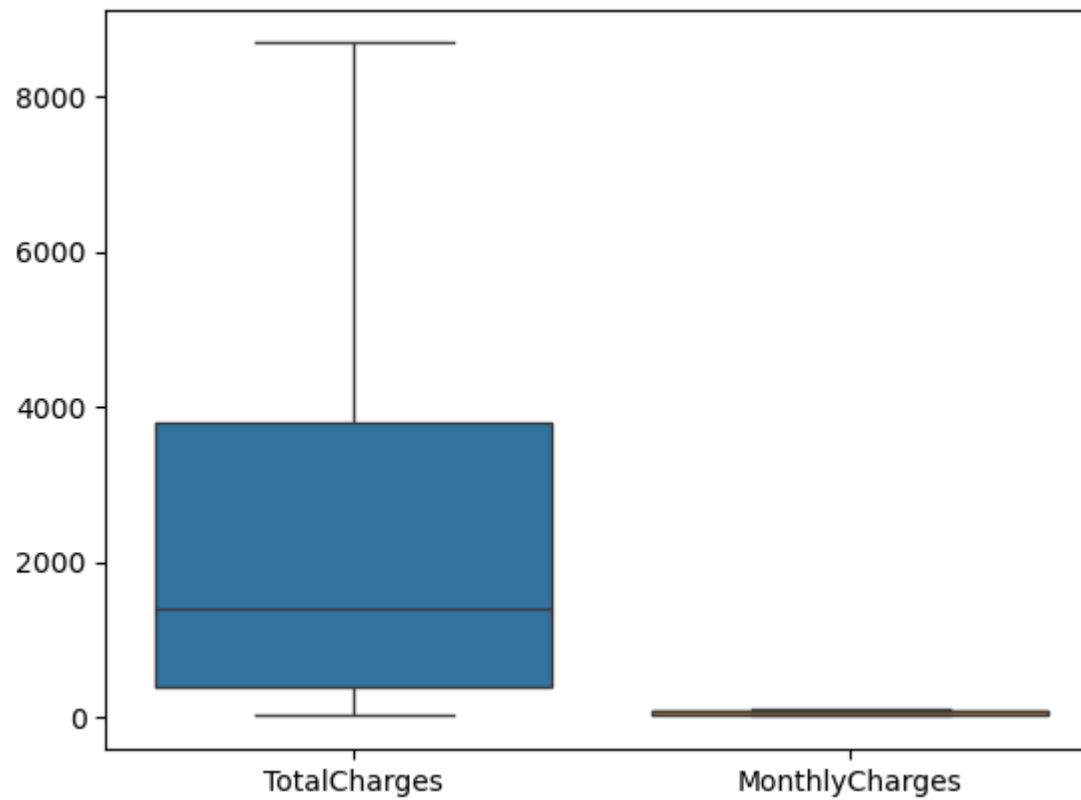
```
In [34]: 1 telco_data.sample(3)
```

Out[34]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	T
533	1	1	1	1	1	2	1	0	0	2	
2959	1	0	1	1	1	2	1	2	2	0	
4710	0	0	1	1	0	1	0	0	2	2	

```
In [35]: 1 sns.boxplot(data = telco_data[['TotalCharges', 'MonthlyCharges']])
```

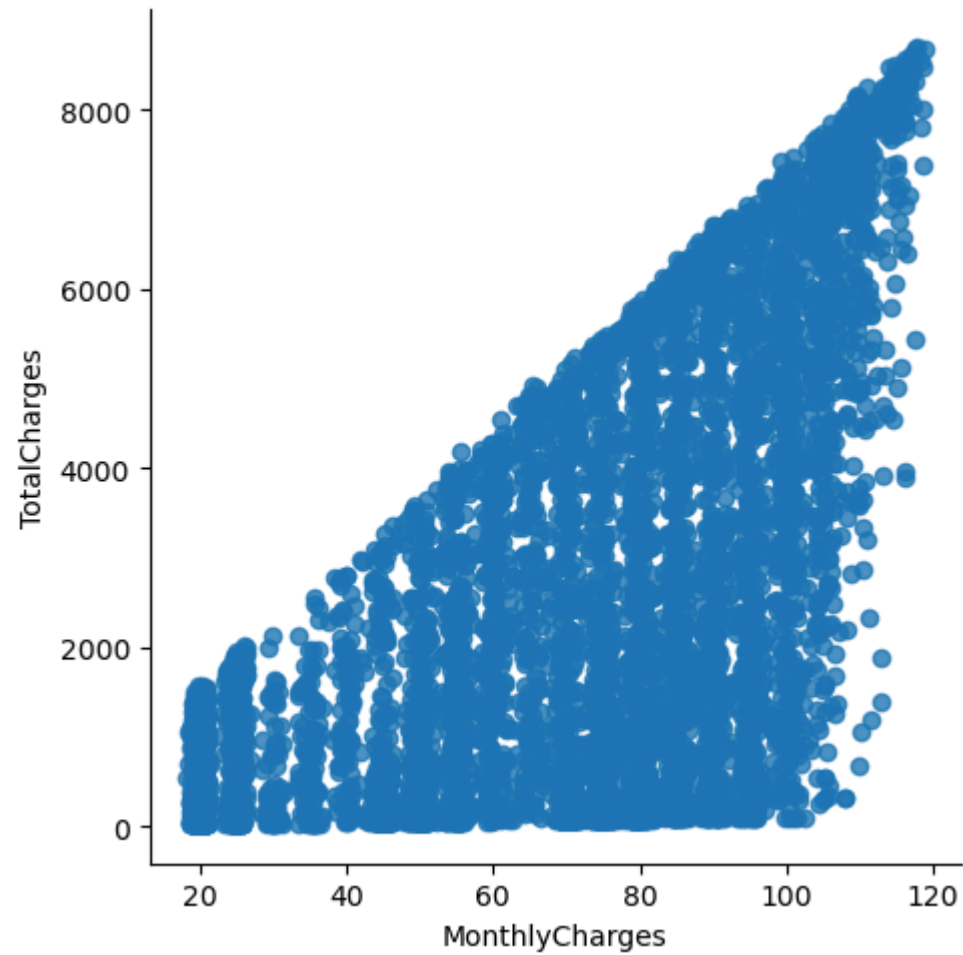
```
Out[35]: <Axes: >
```



***4. * Relationship between Monthly Charges and Total Charges**

```
In [36]: 1 sns.lmplot(data=telco_data, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```

```
Out[36]: <seaborn.axisgrid.FacetGrid at 0x2d4e2c55a50>
```



***5. * Churn by Monthly Charges and Total Charges**


```
In [37]: 1 # kernel density estimate (KDE) plot.
2 Mth = sns.kdeplot(telco_data.MonthlyCharges[(telco_data["Churn"] == 0) ],
3                 color="Red", shade = True)
4 Mth = sns.kdeplot(telco_data.MonthlyCharges[(telco_data["Churn"] == 1) ],
5                 ax =Mth, color="Blue", shade= True)
6 Mth.legend(["No Churn", "Churn"],loc='upper right')
7 Mth.set_ylabel('Density')
8 Mth.set_xlabel('Monthly Charges')
9 Mth.set_title('Monthly charges by churn')
```

C:\Users\sumit\AppData\Local\Temp\ipykernel_10728\1021104028.py:2: FutureWarning:

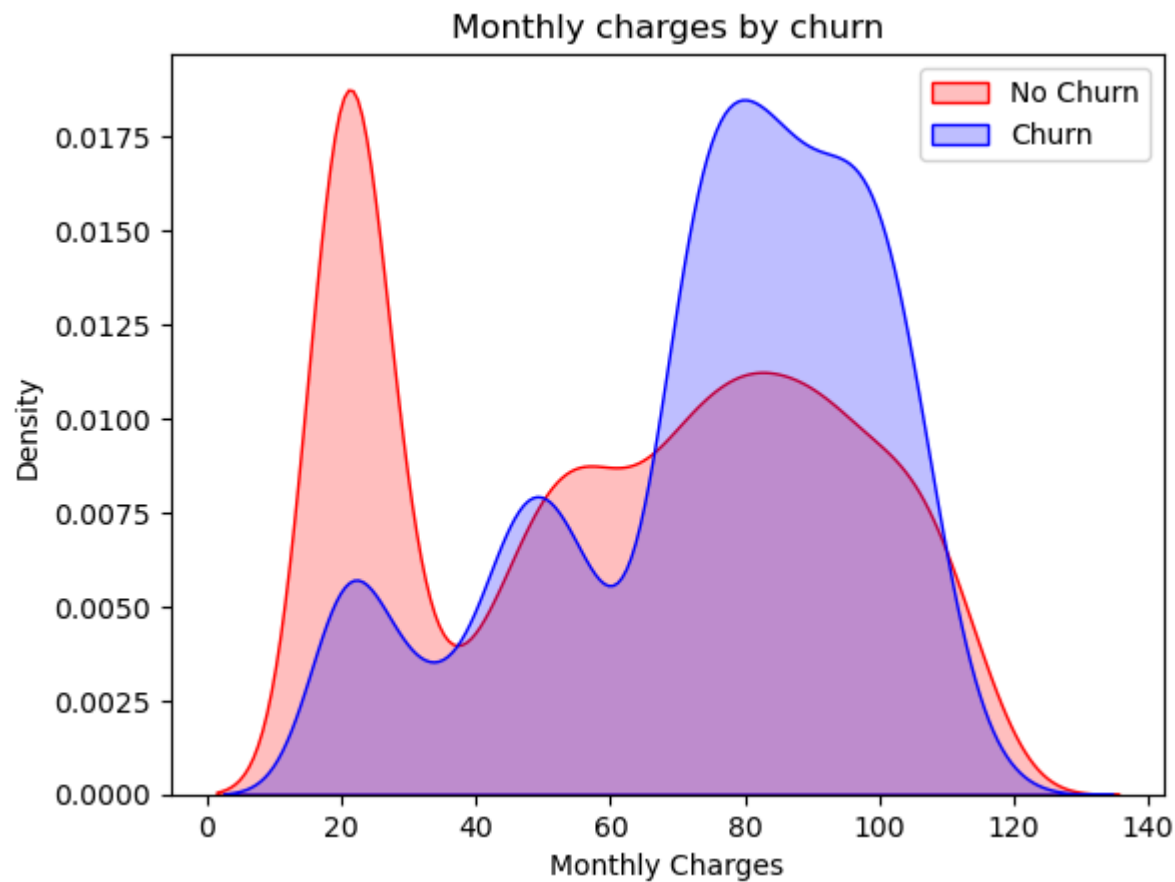
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

```
Mth = sns.kdeplot(telco_data.MonthlyCharges[(telco_data["Churn"] == 0) ],
C:\Users\sumit\AppData\Local\Temp\ipykernel_10728\1021104028.py:4: FutureWarning:
```

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

```
Mth = sns.kdeplot(telco_data.MonthlyCharges[(telco_data["Churn"] == 1) ],
```

Out[37]: Text(0.5, 1.0, 'Monthly charges by churn')



Insight: Churn is high when Monthly Charges are high

```
In [38]: 1 Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 0) ],
2                 color="Red", shade = True)
3 Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 1) ],
4                 ax =Tot, color="Blue", shade= True)
5 Tot.legend(["No Churn", "Churn"],loc='upper right')
6 Tot.set_ylabel('Density')
7 Tot.set_xlabel('Total Charges')
8 Tot.set_title('Total charges by churn')
```

C:\Users\sumit\AppData\Local\Temp\ipykernel_10728\2039743036.py:1: FutureWarning:

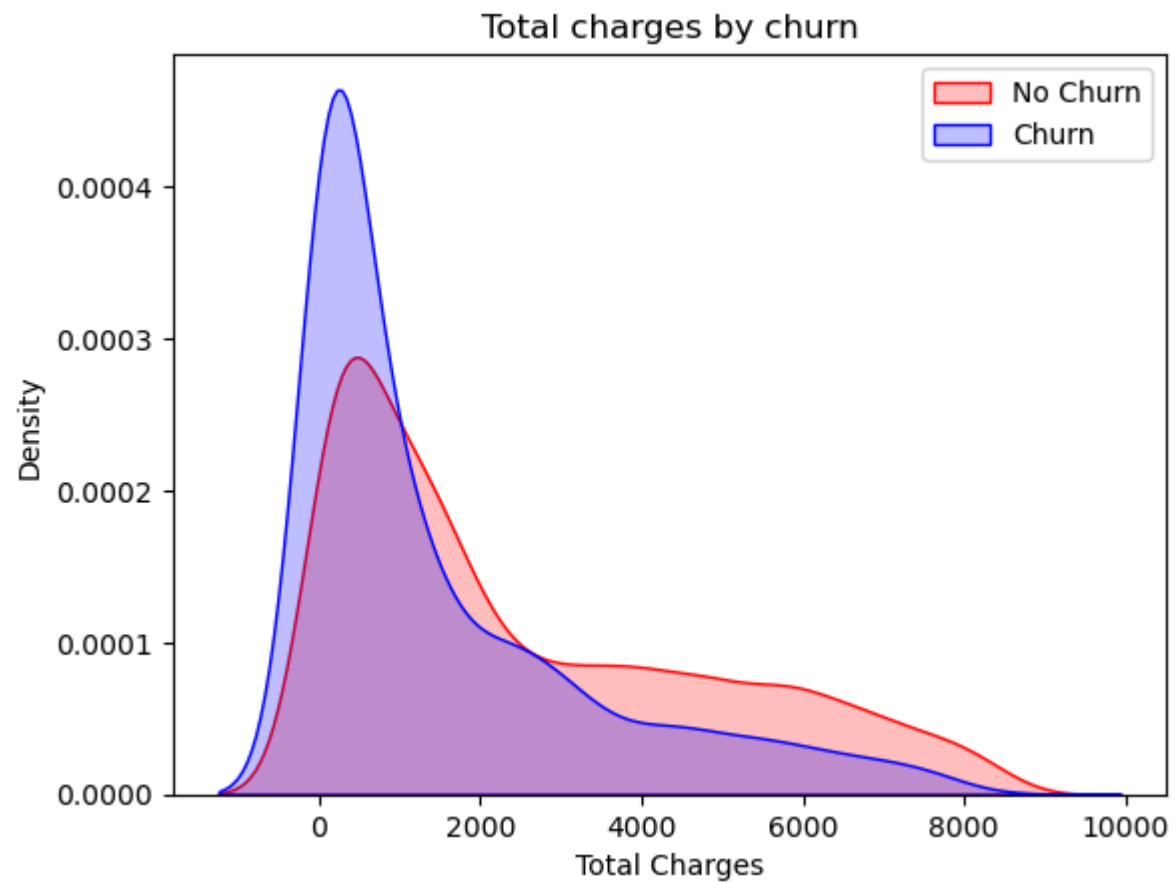
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

```
Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 0) ],
C:\Users\sumit\AppData\Local\Temp\ipykernel_10728\2039743036.py:3: FutureWarning:
```

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

```
Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 1) ],
```

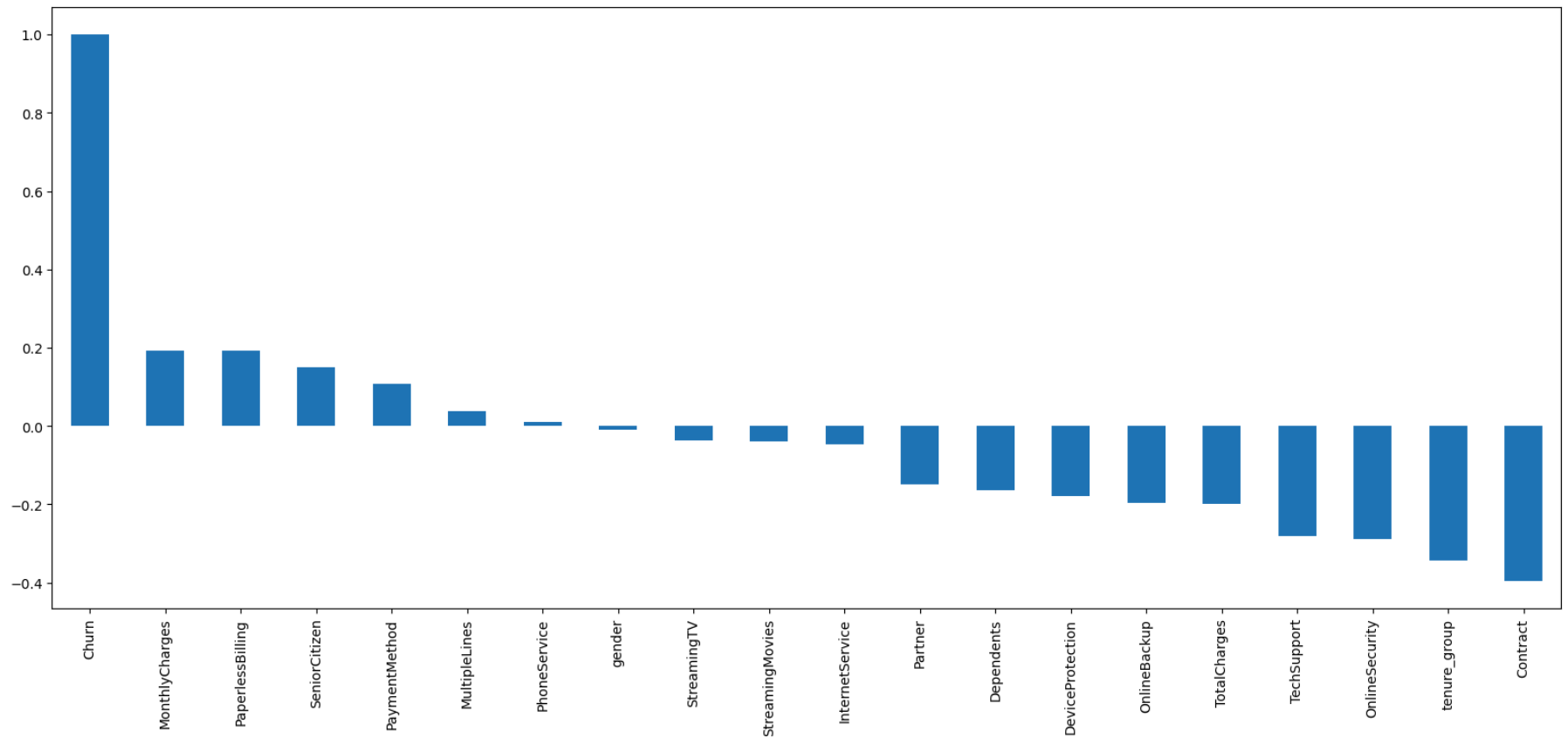
Out[38]: Text(0.5, 1.0, 'Total charges by churn')



***6. Build a corelation of all predictors with 'Churn' ***

```
In [39]: 1 plt.figure(figsize=(20,8))  
2 telco_data.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

Out[39]: <Axes: >



****Derived Insight: ****

HIGH Churn seen in case of Month to month contracts, No online security, No Tech support, First year of subscription and Fibre Optics Internet

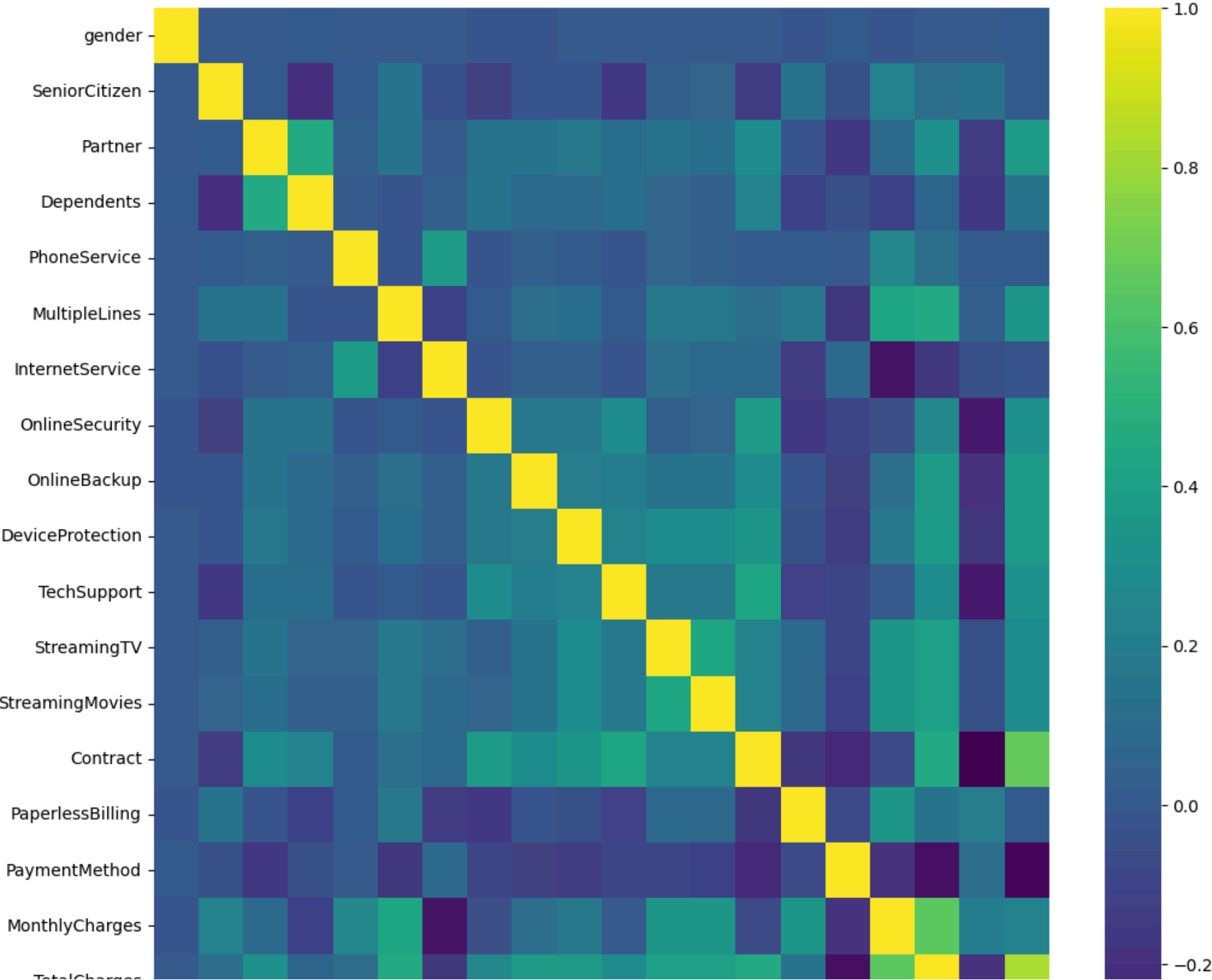
LOW Churn is seen in case of Long term contracts, Subscriptions without internet service and The customers engaged for 5+ years

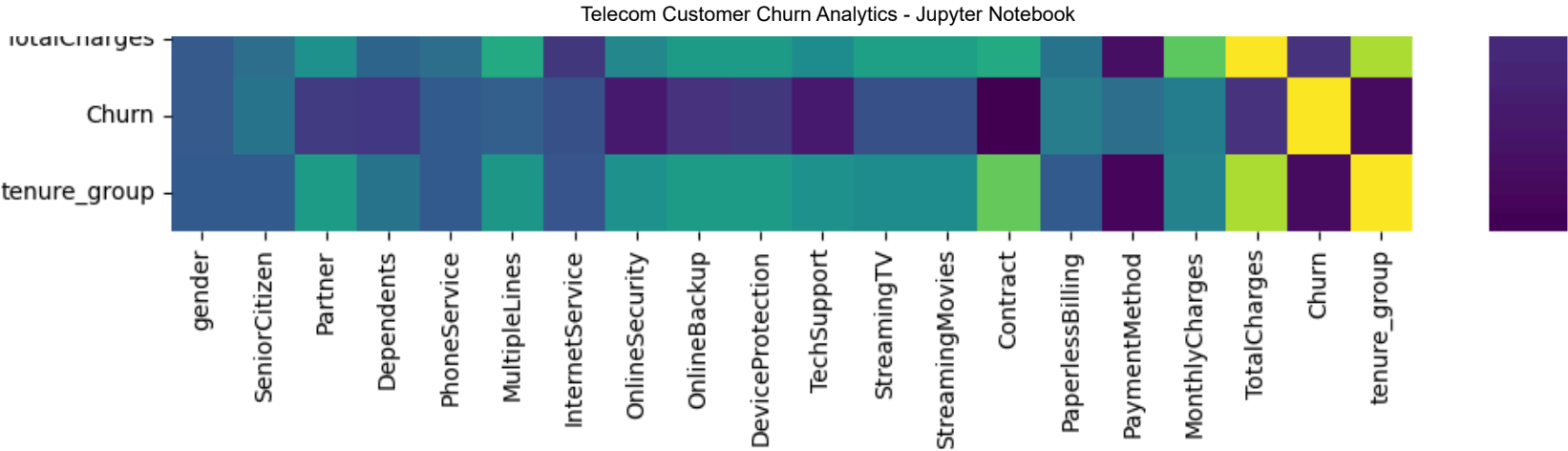
Factors like Gender, Availability of PhoneService and # of multiple lines have almost NO impact on Churn

This is also evident from the Heatmap below

```
In [40]: 1 plt.figure(figsize=(12,12))
          2 sns.heatmap(telco_data.corr(), cmap="viridis")
```

```
Out[40]: <Axes: >
```



```
In [41]: 1 X=telco_data.drop('Churn',axis=1)
          2 y=telco_data['Churn']
```

In [42]:

1 X

Out[42]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	T
0	0	0	1	0	0	1	0	0	2	0	
1	1	0	0	0	1	0	0	2	0	0	2
2	1	0	0	0	1	0	0	2	2	0	0
3	1	0	0	0	0	1	0	2	0	0	2
4	0	0	0	0	1	0	1	0	0	0	0
...
7038	1	0	1	1	1	2	0	2	0	0	2
7039	0	0	1	1	1	2	1	0	2	0	2
7040	0	0	1	1	0	1	0	2	0	0	0
7041	1	1	1	0	1	2	1	0	0	0	0
7042	1	0	0	0	1	0	1	2	0	0	2

7032 rows × 19 columns



In [43]:

1 telco_data['Churn'].value_counts()/len(telco_data) *#data is highly imbalancing*

Out[43]:

Churn

0	0.734215
1	0.265785

Name: count, dtype: float64

Train Test Split

```
In [44]: 1 from sklearn.model_selection import train_test_split
2
3 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
In [45]: 1 print('Traing data shape')
2
3 print(X_train.shape)
4 print(y_train.shape)
5
6 print('Testing Data shape')
7
8 print(X_test.shape)
9 print(y_test.shape)
```

```
Traing data shape
(5625, 19)
(5625,)
Testing Data shape
(1407, 19)
(1407,)
```

```
In [46]: 1 print(y_test.value_counts())
2
3 print(y_train.value_counts())
```

```
Churn
0    1033
1     374
Name: count, dtype: int64
Churn
0    4130
1    1495
Name: count, dtype: int64
```

Decision Tree

```
In [47]: 1 from sklearn.tree import DecisionTreeClassifier
```

```
In [48]: 1 model_dtc=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
In [49]: 1 model_dtc.fit(X_train,y_train)
```

```
Out[49]: DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [50]: 1 model_dtc.score(X_test,y_test)
```

```
Out[50]: 0.7619047619047619
```

```
In [51]: 1 y_pred=model_dtc.predict(X_test)
2 y_pred[:10]
```

```
Out[51]: array([0, 0, 1, 0, 0, 1, 0, 1, 0, 0], dtype=int64)
```

```
In [52]: 1 print(y_test[:10])
```

```
2481    0
6784    0
6125    1
3052    0
4099    0
3223    0
3774    0
3469    0
3420    0
1196    0
Name: Churn, dtype: int64
```

```
In [53]: 1 from sklearn.metrics import classification_report
          2 print(classification_report(y_test, y_pred, labels=[0,1]))
```

	precision	recall	f1-score	support
0	0.84	0.83	0.84	1033
1	0.55	0.56	0.56	374
accuracy			0.76	1407
macro avg	0.70	0.70	0.70	1407
weighted avg	0.76	0.76	0.76	1407

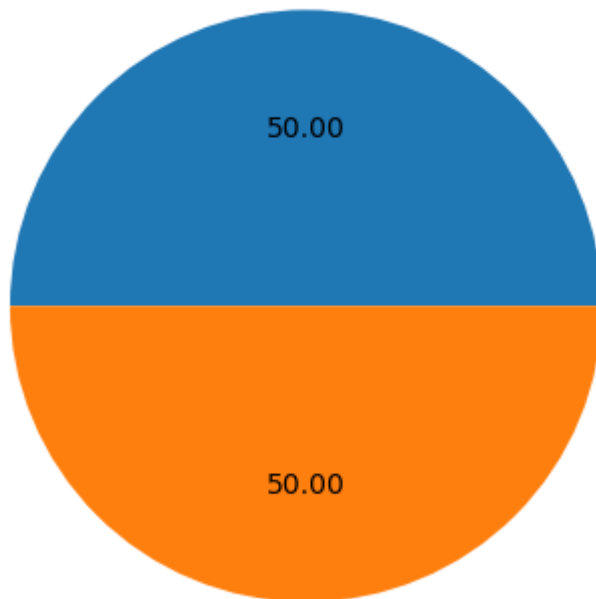
" As you can see that the accuracy is quite low, and as it's an imbalanced dataset, we shouldn't consider Accuracy as our metrics to measure the model, as Accuracy is cursed in imbalanced datasets. Hence, we need to check recall, precision & f1 score for the minority class, and it's quite evident that the precision, recall & f1 score is too low for Class 1, i.e. churned customers. Hence, moving ahead to call SMOTEENN (UpSampling + ENN)"

"main advantage of using SMOTEENN is that it addresses both overfitting and underfitting issues that can arise from class imbalance. By generating synthetic samples and removing noisy ones"

Balancing the Datasets

```
In [54]: 1 from imblearn.over_sampling import SMOTE
2
3 smote=SMOTE()
4
5 X_ovs,y_ovs=smote.fit_resample(X,y)
6
7
8
9 fig, oversp = plt.subplots()
10 oversp.pie( y_ovs.value_counts(), autopct='%.2f')
11 oversp.set_title("Over-sampling")
12 plt.show()
13
```

Over-sampling



```
In [55]: 1 Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_ovs, y_ovs,test_size=0.2,random_state=42)
```

Logistic Regression

```
In [56]: 1 from sklearn.linear_model import LogisticRegression
2
3 model_lr=LogisticRegression(max_iter=1000)
```

```
In [57]: 1 model_lr.fit(Xr_train,yr_train)
```

```
Out[57]: LogisticRegression(max_iter=1000)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [58]: 1 y_pred=model_lr.predict(Xr_test)
2 y_pred[:10]
```

```
Out[58]: array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

```
In [59]: 1 model_lr.score(Xr_test,yr_test)
```

```
Out[59]: 0.8059051306873185
```



```
In [60]: 1 from sklearn.metrics import accuracy_score, classification_report
2
3 report = classification_report(y_pred, yr_test, labels=[0, 1])
4
5 print(report)
```

	precision	recall	f1-score	support
0	0.77	0.83	0.80	968
1	0.84	0.79	0.81	1098
accuracy			0.81	2066
macro avg	0.81	0.81	0.81	2066
weighted avg	0.81	0.81	0.81	2066

```
In [61]: 1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(yr_test,y_pred)
```

```
Out[61]: array([[802, 235],
               [166, 863]], dtype=int64)
```

Decision Tree classifier

```
In [62]: 1 from sklearn.tree import DecisionTreeClassifier
2
3 model_dtc=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
In [63]: 1 model_dtc.fit(Xr_train,yr_train)
```

```
Out[63]: DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [64]: 1 y_pred=model_dtc.predict(Xr_test)
          2 y_pred[:10]
```

```
Out[64]: array([1, 0, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

```
In [65]: 1 yr_test[:10]
```

```
Out[65]: 4139    1
          1692    0
          2692    0
          7704    1
          321    0
          9752    1
           39    1
          3813    0
          7396    1
          2613    0
          Name: Churn, dtype: int64
```

```
In [66]: 1 model_dtc.score(Xr_test,yr_test)
```

```
Out[66]: 0.8073572120038722
```

```
In [67]: 1 print(classification_report(yr_test, y_pred, labels=[0,1]))
```

	precision	recall	f1-score	support
0	0.83	0.77	0.80	1037
1	0.78	0.85	0.81	1029
accuracy			0.81	2066
macro avg	0.81	0.81	0.81	2066
weighted avg	0.81	0.81	0.81	2066

```
In [68]: 1 confusion_matrix(yr_test,y_pred)
```

```
Out[68]: array([[797, 240],  
               [158, 871]], dtype=int64)
```

Random Forest Classifier

```
In [69]: 1 from sklearn.ensemble import RandomForestClassifier  
        2  
        3 model_rfc=RandomForestClassifier(n_estimators=100, random_state = 100,max_depth=6, min_samples_leaf=8,class_weight='balanced')
```

```
In [70]: 1 model_rfc.fit(Xr_train,yr_train)
```

```
Out[70]: RandomForestClassifier(class_weight='balanced', max_depth=6, min_samples_leaf=8,  
                                random_state=100)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [71]: 1 y_pred=model_rfc.predict(Xr_test)  
        2 y_pred[:10]
```

```
Out[71]: array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

```
In [72]: 1 yr_test[:10]
```

```
Out[72]: 4139    1
          1692    0
          2692    0
          7704    1
          321     0
          9752    1
           39     1
          3813    0
          7396    1
          2613    0
          Name: Churn, dtype: int64
```

```
In [73]: 1 model_rfc.score(Xr_test,yr_test)
```

```
Out[73]: 0.81945788964182
```

```
In [74]: 1 report_rfc=classification_report(y_pred,yr_test)
          2 print(report_rfc)
```

	precision	recall	f1-score	support
0	0.77	0.85	0.81	938
1	0.87	0.79	0.83	1128
accuracy			0.82	2066
macro avg	0.82	0.82	0.82	2066
weighted avg	0.82	0.82	0.82	2066

```
In [75]: 1 confusion_matrix(yr_test,y_pred)
```

```
Out[75]: array([[801, 236],
                [137, 892]], dtype=int64)
```

AdaBoost

```
In [76]: 1 from sklearn.ensemble import AdaBoostClassifier
```

```
In [77]: 1 model_abc=AdaBoostClassifier(n_estimators=100)
```

```
In [78]: 1 model_abc.fit(Xr_train,yr_train)
```

```
Out[78]: AdaBoostClassifier(n_estimators=100)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [79]: 1 y_pred=model_abc.predict(Xr_test)
```

```
In [80]: 1 print(classification_report(y_pred,yr_test))
```

	precision	recall	f1-score	support
0	0.78	0.86	0.82	948
1	0.87	0.80	0.83	1118
accuracy			0.82	2066
macro avg	0.82	0.83	0.82	2066
weighted avg	0.83	0.82	0.82	2066

```
In [81]: 1 confusion_matrix(yr_test,y_pred)
```

```
Out[81]: array([[811, 226],
                [137, 892]], dtype=int64)
```

GradientBoostingClassifier

```
In [82]: 1 from sklearn.ensemble import GradientBoostingClassifier
        2 model_gbc=GradientBoostingClassifier()
        3 model_gbc
```

Out[82]: GradientBoostingClassifier()

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [83]: 1 model_gbc.fit(Xr_train,yr_train)
```

Out[83]: GradientBoostingClassifier()

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [84]: 1 y_pred_gbc=model_gbc.predict(Xr_test)
        2 y_pred_gbc[:10]
```

Out[84]: array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)

```
In [85]: 1 yr_test[:10]
```

Out[85]: 4139 1
1692 0
2692 0
7704 1
321 0
9752 1
39 1
3813 0
7396 1
2613 0
Name: Churn, dtype: int64

```
In [86]: 1 print(classification_report(y_pred_gbc,yr_test))
```

	precision	recall	f1-score	support
0	0.80	0.85	0.82	978
1	0.86	0.81	0.83	1088
accuracy			0.83	2066
macro avg	0.83	0.83	0.83	2066
weighted avg	0.83	0.83	0.83	2066

```
In [87]: 1 confusion_matrix(yr_test,y_pred)
```

```
Out[87]: array([[811, 226],  
               [137, 892]], dtype=int64)
```

Xgboost

```
In [88]: 1 from xgboost import XGBClassifier
          2
          3 model_xgb=XGBClassifier(class_weight={0:1, 1:2})
          4
          5 model_xgb
```

```
Out[88]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                        class_weight={0: 1, 1: 2}, colsample_bylevel=None,
                        colsample_bynode=None, colsample_bytree=None, device=None,
                        early_stopping_rounds=None, enable_categorical=False,
                        eval_metric=None, feature_types=None, gamma=None,
                        grow_policy=None, importance_type=None,
                        interaction_constraints=None, learning_rate=None, max_bin=None,
                        max_cat_threshold=None, max_cat_to_onehot=None,
                        max_delta_step=None, max_depth=None, max_leaves=None,
                        min_child_weight=None, missing=nan, monotone_constraints=None,
                        multi_strategy=None, n_estimators=None, n_jobs=None,
                        num_parallel_tree=None, ...)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**


```
In [89]: 1 model_xgb.fit(Xr_train,yr_train)
```

```
C:\Users\sumit\anaconda3\Lib\site-packages\xgboost\core.py:160: UserWarning: [16:19:14] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\learner.cc:742: Parameters: { "class_weight" } are not used.
```

```
warnings.warn(smsg, UserWarning)
```

```
Out[89]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                      class_weight={0: 1, 1: 2}, colsample_bylevel=None,
                      colsample_bynode=None, colsample_bytree=None, device=None,
                      early_stopping_rounds=None, enable_categorical=False,
                      eval_metric=None, feature_types=None, gamma=None,
                      grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=None, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=None, max_leaves=None,
                      min_child_weight=None, missing=nan, monotone_constraints=None,
                      multi_strategy=None, n_estimators=None, n_jobs=None,
                      num_parallel_tree=None, ...)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [90]: 1 y_pred=model_xgb.predict(Xr_test)
         2 y_pred[:10]
```

```
Out[90]: array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0])
```

```
In [91]: 1 yr_test[:10]
```

```
Out[91]: 4139    1
          1692    0
          2692    0
          7704    1
          321     0
          9752    1
           39     1
          3813    0
          7396    1
          2613    0
          Name: Churn, dtype: int64
```

```
In [92]: 1 print(classification_report(y_pred,yr_test))
```

	precision	recall	f1-score	support
0	0.84	0.83	0.84	1042
1	0.83	0.84	0.83	1024
accuracy			0.84	2066
macro avg	0.84	0.84	0.84	2066
weighted avg	0.84	0.84	0.84	2066

```
In [93]: 1 from sklearn.metrics import confusion_matrix
          2
          3 # Assuming y_pred and y_test are your predicted and true labels respectively
          4 cm = confusion_matrix(yr_test, y_pred)
          5
          6 print("Confusion Matrix:")
          7 print(cm)
```

```
Confusion Matrix:
[[870 167]
 [172 857]]
```

GradientBoostingClassifier and adaboost has accuracy i go with Gradientboostingclassifier /finding the best hyperparameter

```
In [94]: 1 from sklearn.model_selection import RandomizedSearchCV
2 from sklearn.ensemble import GradientBoostingClassifier
3 import time
4
5 # Define your GradientBoostingClassifier and param_dist
6 model = GradientBoostingClassifier()
7 param_dist = {
8     'learning_rate': [0.1, 0.5, 1.0],
9     'n_estimators': [50, 100, 200],
10    'max_depth': [3, 5, 7], # Example: Adding max_depth parameter
11    'min_samples_split': [2, 5, 10] # Example: Adding min_samples_split parameter
12 }
13
14 # Create RandomizedSearchCV object with fewer iterations
15 random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=5, cv=10, scoring='acc
16
17 # Start the timer
18 start_time = time.time()
19
20 # Fit the RandomizedSearchCV object
21 random_search.fit(Xr_train, yr_train)
22
23 # Stop the timer
24 end_time = time.time()
25
26 # Calculate the total time taken
27 total_time = end_time - start_time
28
29 print("RandomizedSearchCV took {:.2f} seconds to complete.".format(total_time))
30
31 # Get the best parameters
32 best_params = random_search.best_params_
33 print("Best Parameters:", best_params)
```

RandomizedSearchCV took 219.31 seconds to complete.

Best Parameters: {'n_estimators': 100, 'min_samples_split': 5, 'max_depth': 7, 'learning_rate': 0.1}

final model

```
In [95]: 1 from sklearn.ensemble import GradientBoostingClassifier
2
3 # Define the best hyperparameters obtained from GridSearchCV
4 best_params = {
5     'n_estimators': 100, 'min_samples_split':5 , 'max_depth': 7, 'learning_rate': 0.1
6 }
7
8 # Create Gradient Boosting Classifier with the best hyperparameters
9 final_gb_classifier = GradientBoostingClassifier(**best_params)
10
11 # Train the final model on the entire training data
12 final_gb_classifier.fit(Xr_train, yr_train)
```

Out[95]: GradientBoostingClassifier(max_depth=7, min_samples_split=5)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [96]: 1 from sklearn.model_selection import cross_val_score
2
3 # trained model with tuned hyperparameters
4 # X_train and y_train are your training data
5 # cv=10 indicates 10-fold cross-validation
6 cv_scores = cross_val_score(final_gb_classifier, Xr_train, yr_train, cv=10, scoring='accuracy')
7
8 # Print the cross-validation scores
9 print("Cross-validation scores:", cv_scores)
10 print("Mean CV score:", cv_scores.mean())
11
```

Cross-validation scores: [0.83171913 0.86924939 0.8401937 0.83898305 0.85230024 0.84866828
0.83171913 0.83656174 0.83898305 0.86319613]
Mean CV score: 0.8451573849878935

```
In [97]: 1 y_pred=final_gb_classifier.predict(Xr_test)
        2 y_pred[:10]
```

```
Out[97]: array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

```
In [98]: 1 yr_test[:10]
```

```
Out[98]: 4139    1
        1692    0
        2692    0
        7704    1
        321    0
        9752    1
         39     1
        3813    0
        7396    1
        2613    0
        Name: Churn, dtype: int64
```

```
In [99]: 1 print(classification_report(y_pred,yr_test))
```

	precision	recall	f1-score	support
0	0.83	0.84	0.84	1020
1	0.84	0.83	0.84	1046
accuracy			0.84	2066
macro avg	0.84	0.84	0.84	2066
weighted avg	0.84	0.84	0.84	2066

```
In [100]: 1 confusion_matrix(y_pred,yr_test)
```

```
Out[100]: array([[860, 160],
                [177, 869]], dtype=int64)
```

Electronic check medium are the highest churners

Contract Type - Monthly customers are more likely to churn because of no contract terms, as they are free to go customers.

No Online security, No Tech Support category are high churners

Non senior Citizens are high churners

Pickle file

```
In [101]: 1 import os
          2 import pickle
          3 from sklearn.ensemble import GradientBoostingClassifier
          4
          5 # Change directory if needed
          6 #os.chdir('D:\\Datasets')
          7
          8 # Assuming final_gb_classifier is your trained model
          9 # Define and train Gradient Boosting Classifier
         10 best_params = {
         11     'n_estimators': 100,
         12     'min_samples_split': 5,
         13     'max_depth': 7,
         14     'learning_rate': 0.1
         15 }
         16
         17 final_gb_classifier = GradientBoostingClassifier(**best_params)
         18
         19 # Train the final model on the entire training data (assuming Xr_train and yr_train are defined)
         20 final_gb_classifier.fit(X_train, y_train)
         21
         22 # Dumping the model to a file
         23 with open('final_gb_classifier.pkl', 'wb') as file:
         24     pickle.dump(final_gb_classifier, file)
         25
         26 # Load the saved model
         27 #with open('final_gb_classifier.pkl', 'rb') as file:
         28     # loaded_model = pickle.load(file)
```

Checking accuravy with our features

In [102]:

```
1 import pickle
2 import pandas as pd
3
4 # Load the saved model from the pickle file
5 with open('final_gb_classifier.pkl', 'rb') as file:
6     loaded_model = pickle.load(file)
7
8 # Prepare your own data for testing
9 # Create a DataFrame with your feature data
10 your_features = pd.DataFrame({
11     'gender': [1, 0, 0, 0, 0],
12     'SeniorCitizen': [0, 0, 0, 0, 0],
13     'Partner': [0, 0, 0, 1, 1],
14     'Dependents': [0, 0, 0, 0, 1],
15     'PhoneService': [1, 0, 1, 1, 1],
16     'MultipleLines': [0, 0, 0, 2, 2],
17     'InternetService': [1, 0, 1, 1, 0],
18     'OnlineSecurity': [0, 0, 0, 2, 2],
19     'OnlineBackup': [0, 0, 1, 2, 2],
20     'DeviceProtection': [0, 0, 0, 0, 2],
21     'TechSupport': [0, 0, 0, 2, 2],
22     'StreamingTV': [0, 1, 0, 0, 0],
23     'StreamingMovies': [0, 1, 0, 0, 0],
24     'Contract': [2, 0, 0, 1, 2],
25     'PaperlessBilling': [0, 1, 0, 0, 0],
26     'PaymentMethod': [1, 1, 1, 0, 0],
27     'MonthlyCharges': [90.407734, 58.273891, 74.379767, 108.55, 64.35],
28     'TotalCharges': [707.535237, 3264.466697, 1146.937795, 5610.7, 1558.65],
29     'tenure_group': [0, 4, 1, 4, 2]
30 })
31 # Make predictions using the loaded model on your own data
32 predictions = loaded_model.predict(your_features)
33
34 # Print the predictions
35 print("Predictions:", predictions)
```

Predictions: [0 0 0 0 0]

