# Project Title : Customer Churn Analytics

## Abstract

In the competitive landscape of telecommunications, customer churn poses a significant challenge, impacting revenue and market share. This project aims to analyze customer churn patterns within a telecom company to understand the underlying factors contributing to customer attrition. The study leverages historical customer data, including demographic information, service usage metrics, and customer interaction logs. Through exploratory data analysis and predictive modeling techniques, key drivers of churn will be identified, such as service dissatisfaction, pricing strategies, contract terms, and customer support effectiveness.

## 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 chum 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 problen 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,RandomForestClassifer,AddaboostClassifier,GradientBo

## 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)

# Target Variable

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

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

In [1]:
```
# EDA
import numpy as np
import pandas as pd

# data visualisations
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import os
os.getcwd()
os.chdir(r"C:\Users\Prachi\Documents\VS Code Files\ML CAPSTONE PROJECT\Customer Chu
```

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

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

Out[4]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines |
|---|---|---|---|---|---|---|---|---|
| **0** | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service |
| **1** | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No |
| **2** | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No |
| **3** | 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service |
| **4** | 9237-HQITU | Female | 0 | No | No | 2 | Yes | No |

5 rows × 21 columns

In [5]:
```
telco_base_data.shape
```

Out[5]:
```
(7043, 21)
```

In [6]:
```
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 [7]:

```
for col in telco_base_data.columns:
    print("columns:{} - Unique Values: {}".format(col,telco_base_data[col].unique()
```

```
columns:customerID - Unique Values: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4
801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
columns:gender - Unique Values: ['Female' 'Male']
columns:SeniorCitizen - Unique Values: [0 1]
columns:Partner - Unique Values: ['Yes' 'No']
columns:Dependents - Unique Values: ['No' 'Yes']
columns:tenure - Unique Values: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 7
1 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]
columns:PhoneService - Unique Values: ['No' 'Yes']
columns:MultipleLines - Unique Values: ['No phone service' 'No' 'Yes']
columns:InternetService - Unique Values: ['DSL' 'Fiber optic' 'No']
columns:OnlineSecurity - Unique Values: ['No' 'Yes' 'No internet service']
columns:OnlineBackup - Unique Values: ['Yes' 'No' 'No internet service']
columns:DeviceProtection - Unique Values: ['No' 'Yes' 'No internet service']
columns:TechSupport - Unique Values: ['No' 'Yes' 'No internet service']
columns:StreamingTV - Unique Values: ['No' 'Yes' 'No internet service']
columns:StreamingMovies - Unique Values: ['No' 'Yes' 'No internet service']
columns:Contract - Unique Values: ['Month-to-month' 'One year' 'Two year']
columns:PaperlessBilling - Unique Values: ['Yes' 'No']
columns:PaymentMethod - Unique Values: ['Electronic check' 'Mailed check' 'Bank tr
ansfer (automatic)'
 'Credit card (automatic)']
columns:MonthlyCharges - Unique Values: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
columns:TotalCharges - Unique Values: ['29.85' '1889.5' '108.15' ... '346.45' '30
6.6' '6844.5']
columns:Churn - Unique Values: ['No' 'Yes']
```

In [8]:
```
telco_base_data.columns.values
```

Out[8]:
```
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 [9]:
```
telco_base_data.TotalCharges = pd.to_numeric(telco_base_data.TotalCharges, errors='
```

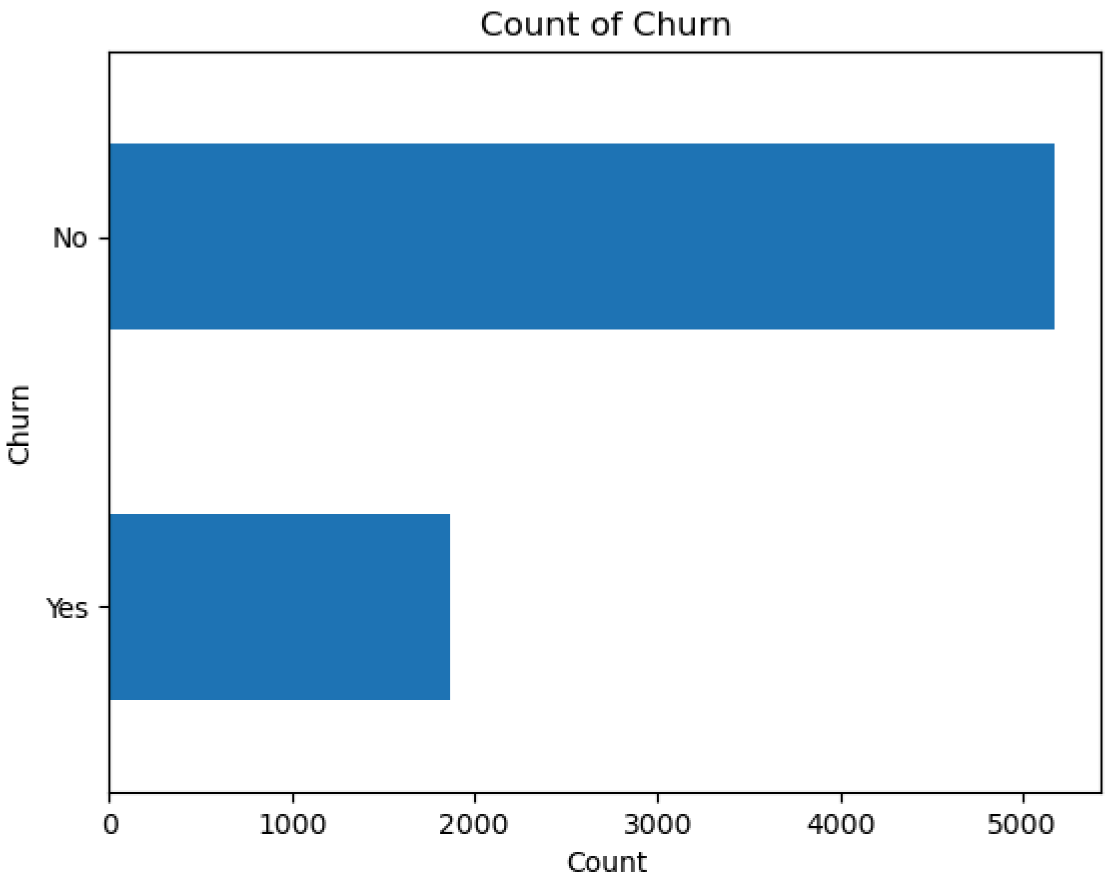
In [10]:
```
telco_base_data.describe()
```

Out[10]:

| | 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 |

In [11]:
```
telco_base_data['Churn'].value_counts().plot(kind='barh')
plt.xlabel("Count")
plt.ylabel("Churn")
```

```
plt.title("Count of Churn")
plt.gca().invert_yaxis()  # Invert y-axis to have 'No Churn' on top
plt.show()
```




In [12]:
```
telco_base_data['Churn'].value_counts()/len(telco_base_data)
```

Out[12]:
```
Churn
No     0.73463
Yes    0.26537
Name: count, dtype: float64
```

In [13]:
```
telco_base_data['Churn'].value_counts()
```

Out[13]:
```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

In [14]:
```
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 [15]:
```
telco_data = telco_base_data.copy()
```

In [16]:
```
telco_data.isna().sum()
```

Out[16]:
```
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 [17]:
```
telco_data.loc[telco_data['TotalCharges'].isna()==True]
```

Out[17]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLin |
|---|---|---|---|---|---|---|---|---|
| **488** | 4472-LVYGI | Female | 0 | Yes | Yes | 0 | No | No phoi servi |
| **753** | 3115-CZMZD | Male | 0 | No | Yes | 0 | Yes | N |
| **936** | 5709-LVOEQ | Female | 0 | Yes | Yes | 0 | Yes | N |
| **1082** | 4367-NUYAO | Male | 0 | Yes | Yes | 0 | Yes | Y |
| **1340** | 1371-DWPAZ | Female | 0 | Yes | Yes | 0 | No | No phoi servi |
| **3331** | 7644-OMVMY | Male | 0 | Yes | Yes | 0 | Yes | N |
| **3826** | 3213-VVOLG | Male | 0 | Yes | Yes | 0 | Yes | Y |
| **4380** | 2520-SGTTA | Female | 0 | Yes | Yes | 0 | Yes | N |
| **5218** | 2923-ARZLG | Male | 0 | Yes | Yes | 0 | Yes | N |
| **6670** | 4075-WKNIU | Female | 0 | Yes | Yes | 0 | Yes | Y |
| **6754** | 2775-SEFEE | Male | 0 | No | Yes | 0 | Yes | Y |

11 rows × 21 columns

In [18]:

```
telco_data.dtypes
```

Out[18]:

```
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 [19]:

```
telco_data.isna().sum()/len(telco_data)
```

Out[19]:
```
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
```

# Missing Value Treatment

In [20]:
```
# Removing missing values
telco_data.dropna(how = 'any', inplace = True)
```

In [21]:
```
# Get the max tenure
print(telco_data['tenure'].max())
```

```
72
```

In [22]:
```
# Define the bins and labels
bins = [0, 12, 24, 36, 48, 60, 72]
labels = ['1 - 12', '13 - 24', '25 - 36', '37 - 48', '49 - 60', '61 - 72']

# Create the tenure_group column
telco_data['tenure_group'] = pd.cut(telco_data['tenure'], bins=bins, labels=labels,
```

In [23]:
```
telco_data['tenure_group']
```

Out[23]:
```
0        1 - 12
1       25 - 36
2        1 - 12
3       37 - 48
4        1 - 12
         ...
7038    25 - 36
7039        NaN
7040     1 - 12
7041     1 - 12
7042    61 - 72
Name: tenure_group, Length: 7032, dtype: category
Categories (6, object): ['1 - 12' < '13 - 24' < '25 - 36' < '37 - 48' < '49 - 60'
< '61 - 72']
```

In [24]:
```
 telco_data['tenure_group'].value_counts()
```

Out[24]:
```
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 [25]:
```
telco_data['tenure_group'].value_counts()/len(telco_data)
```

Out[25]:
```
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
```

Remove columns not reqiured for processing

In [26]:
```
#drop column customerID and tenure
telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telco_data.head()
```

Out[26]:

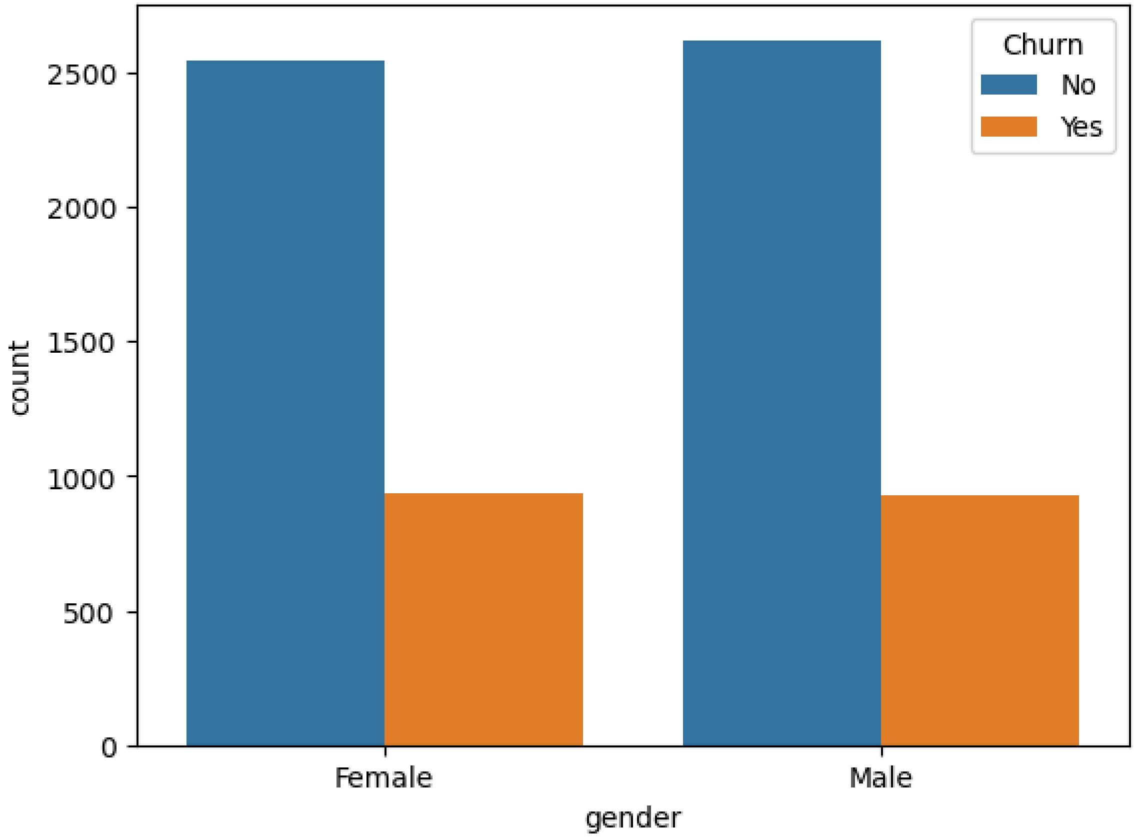
| | gender | SeniorCitizen | Partner | Dependents | PhoneService | MultipleLines | InternetService | Onli |
|---|---|---|---|---|---|---|---|---|
| **0** | Female | 0 | Yes | No | No | No phone service | DSL | |
| **1** | Male | 0 | No | No | Yes | No | DSL | |
| **2** | Male | 0 | No | No | Yes | No | DSL | |
| **3** | Male | 0 | No | No | No | No phone service | DSL | |
| **4** | Female | 0 | No | No | Yes | No | Fiber optic | |

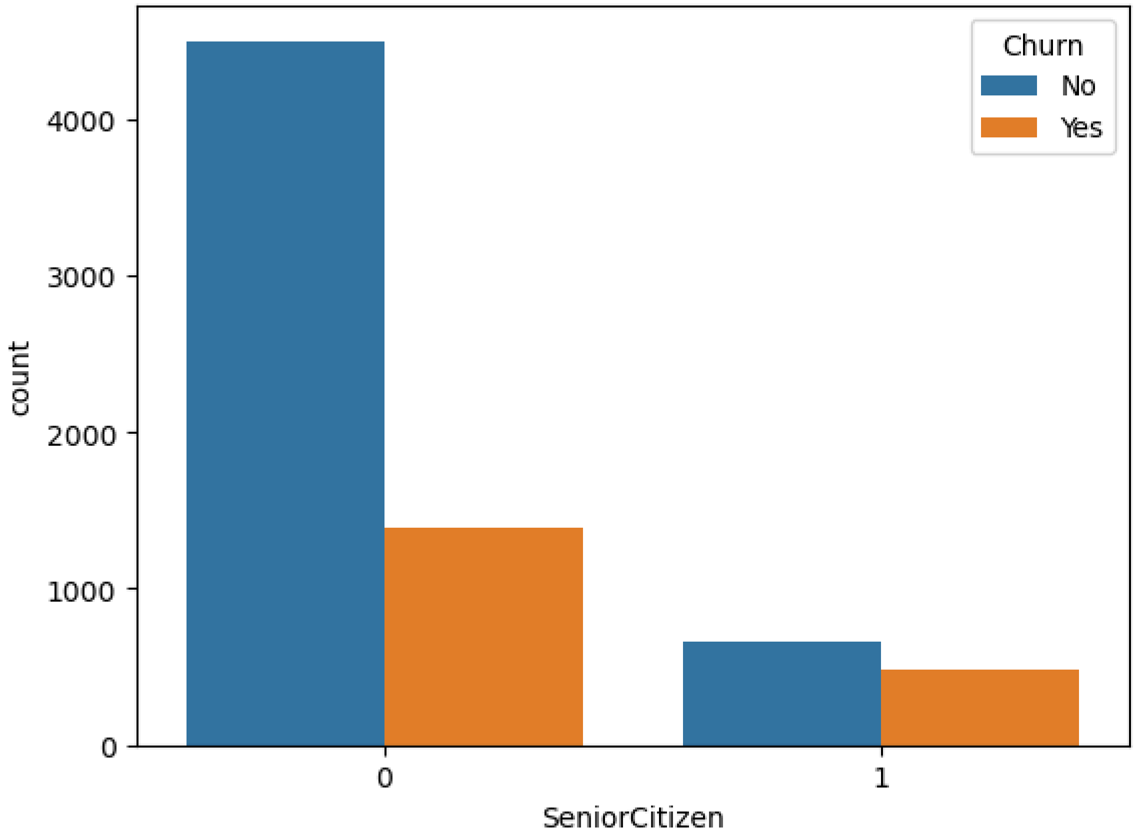
Data Exploration 1. Plot distribution predictors by churn

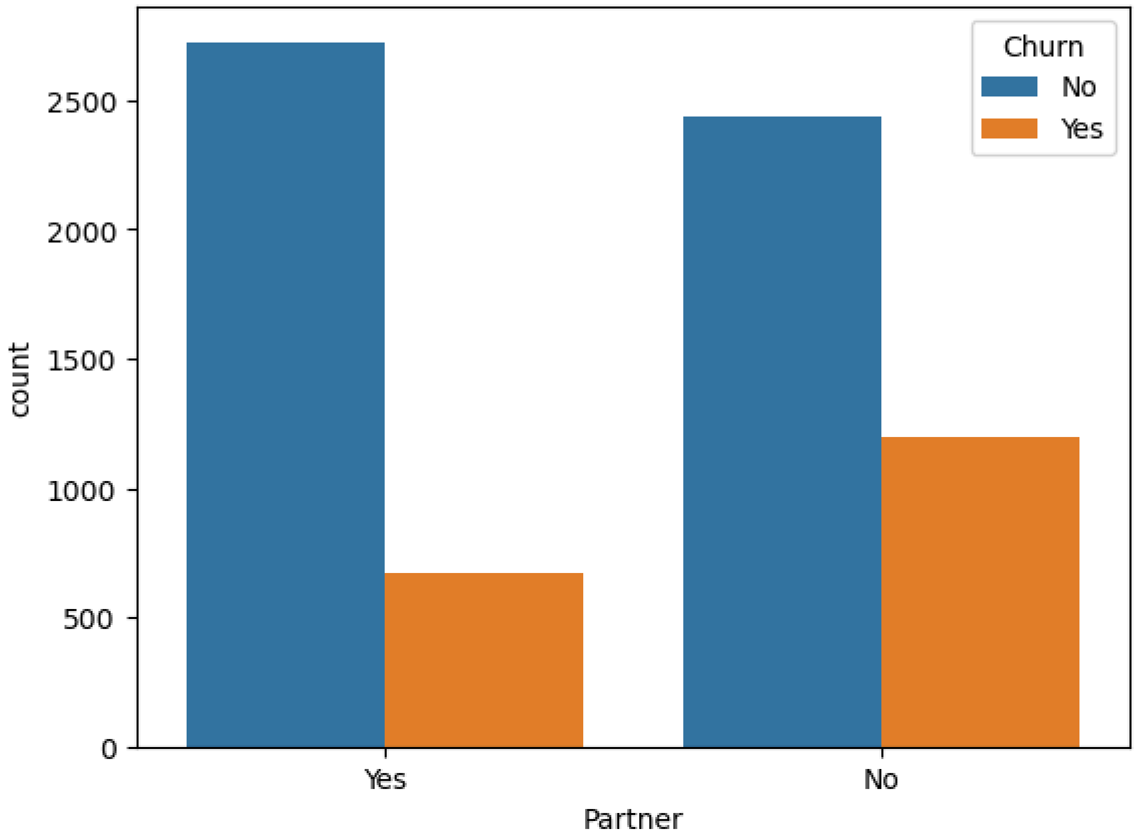
Univariate Analysis

In [27]:
```
for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges', 'Mc
    plt.figure(i)
    sns.countplot(data=telco_data, x=predictor, hue='Churn')
```

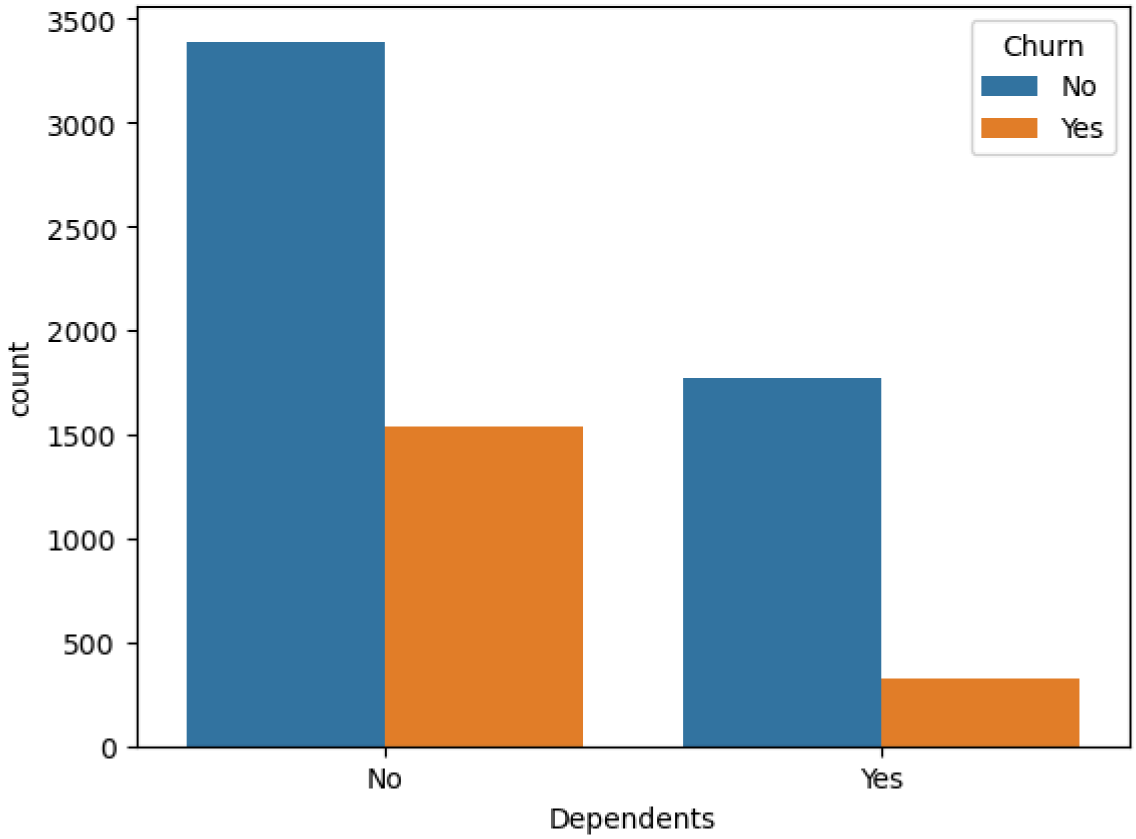
```
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\categorical.py:641: FutureWarn
ing: The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or observ
ed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\categorical.py:641: FutureWarn
ing: The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or observ
ed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```



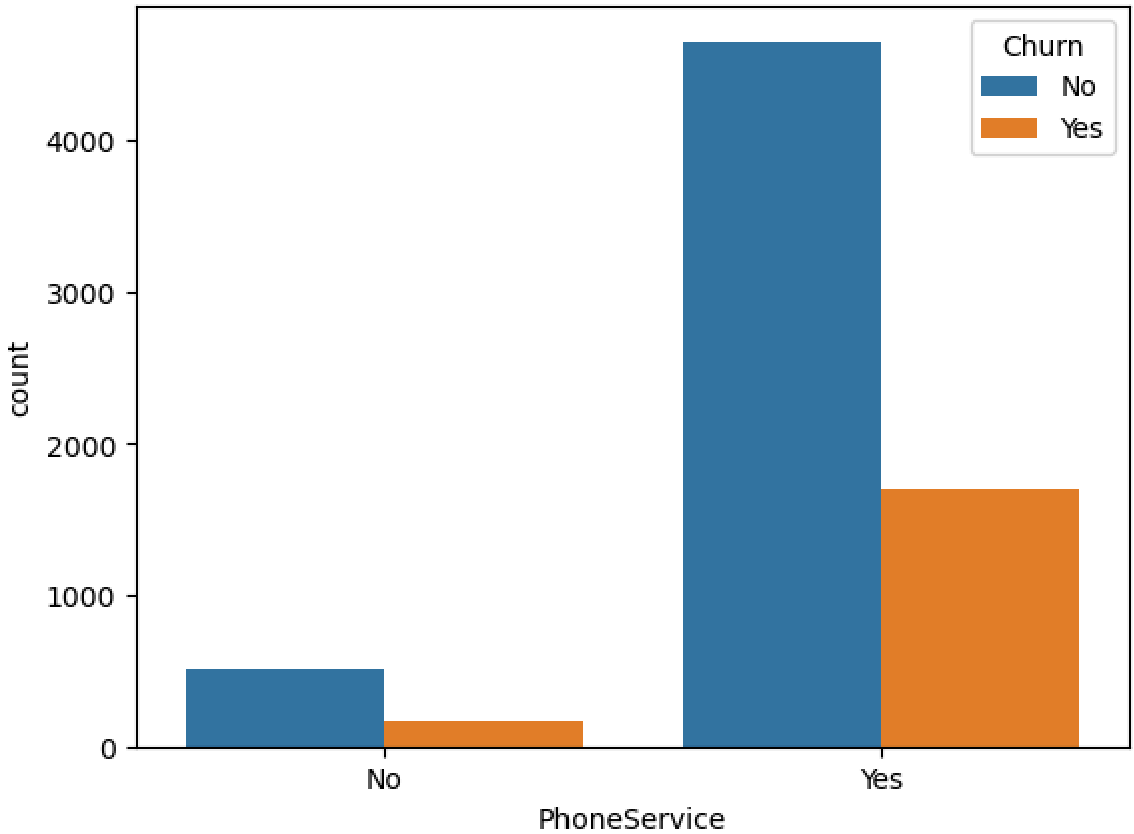





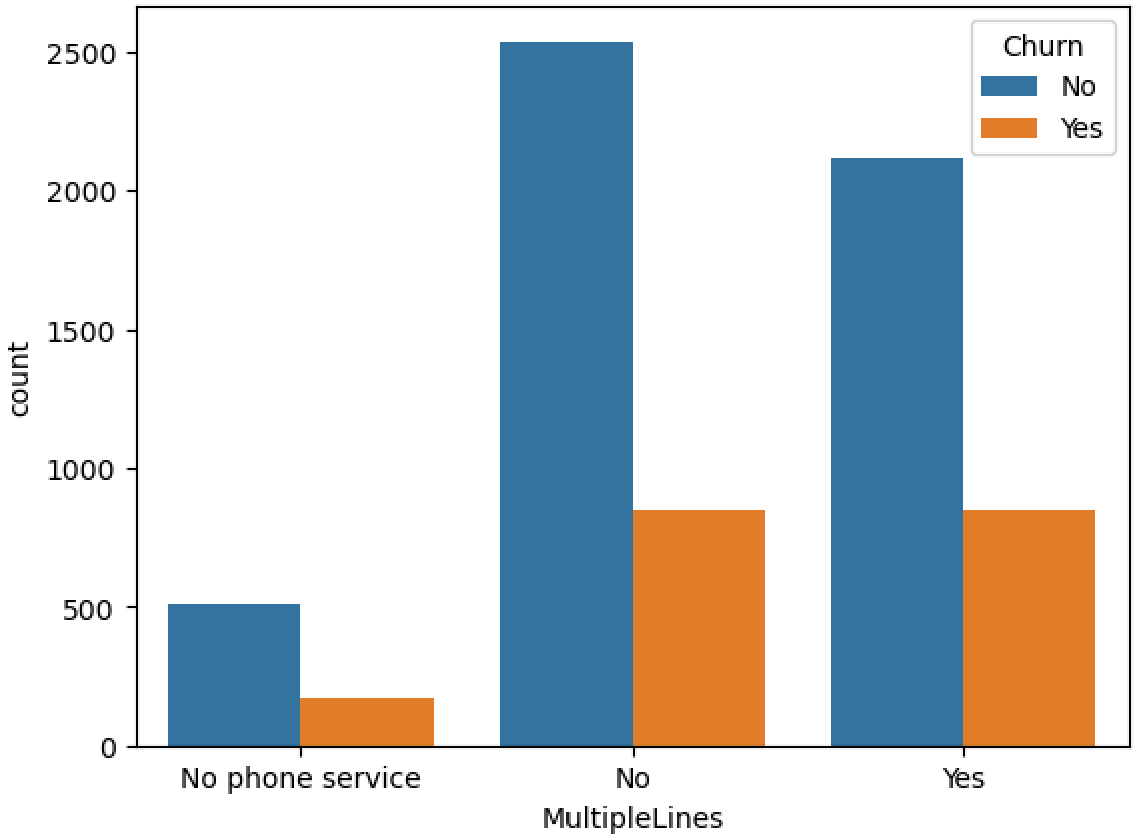
Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
Partner

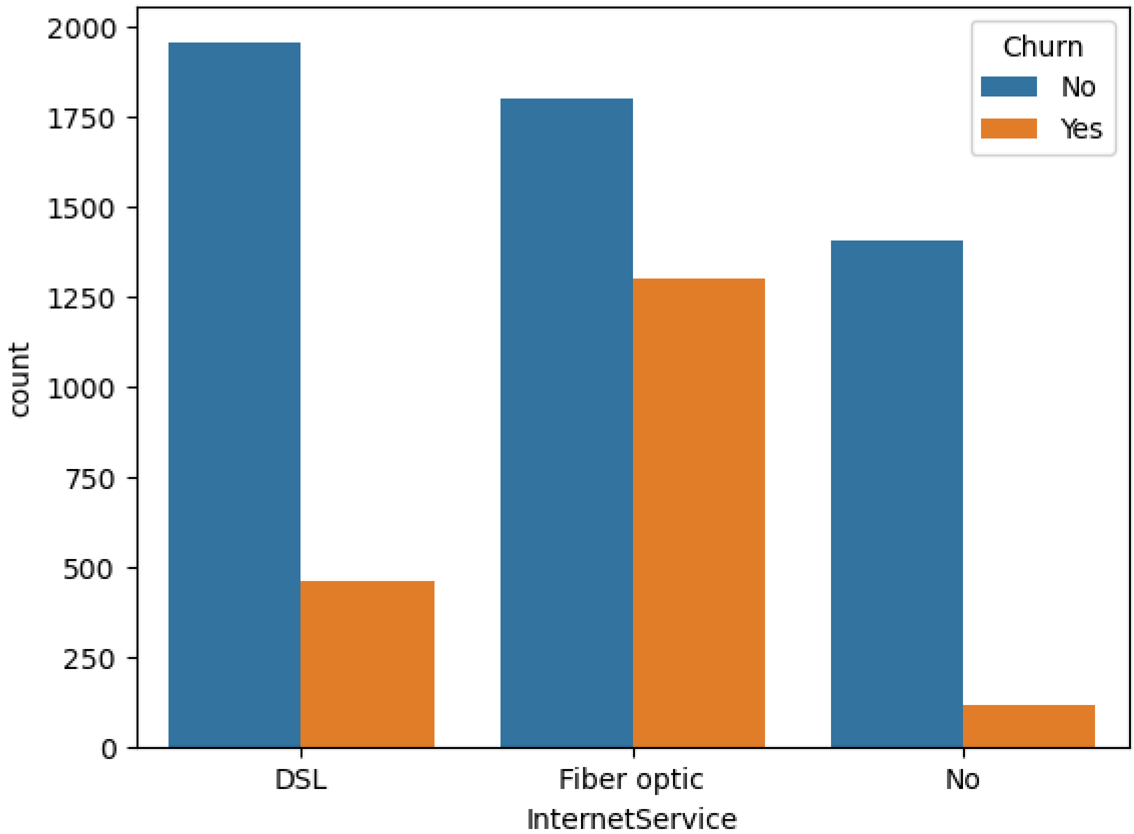
Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents

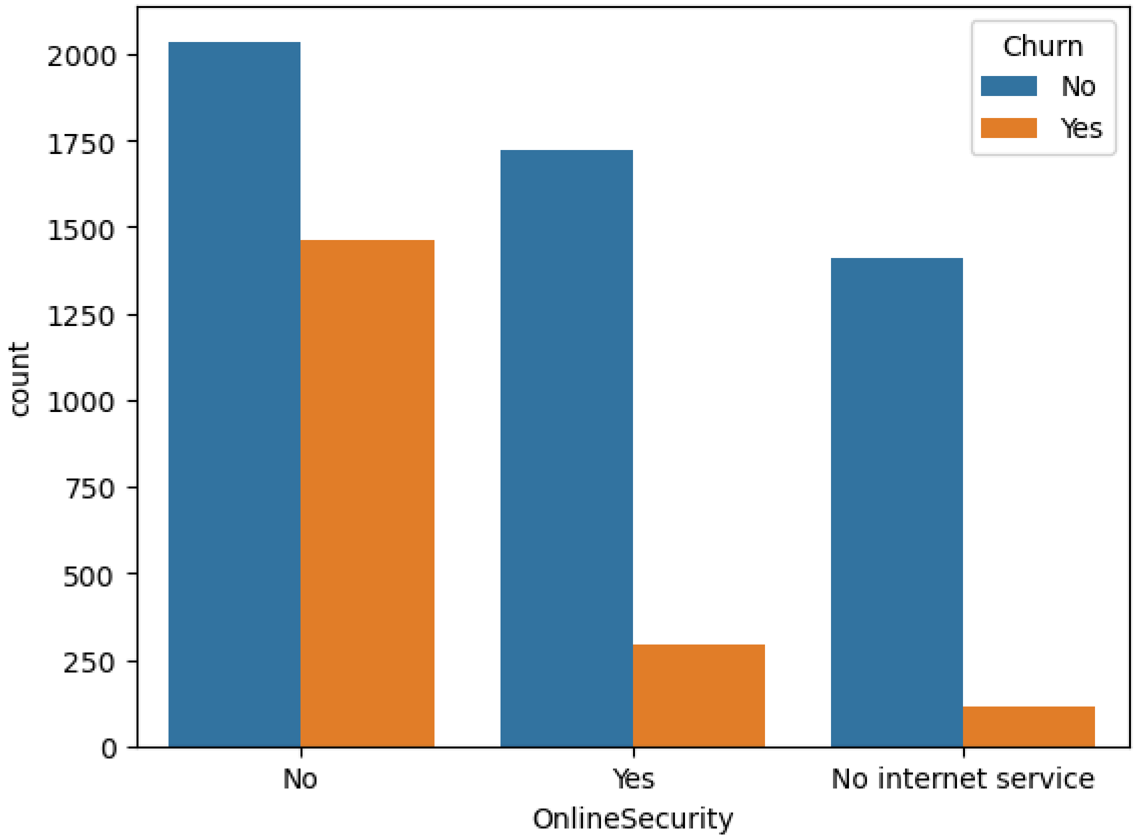


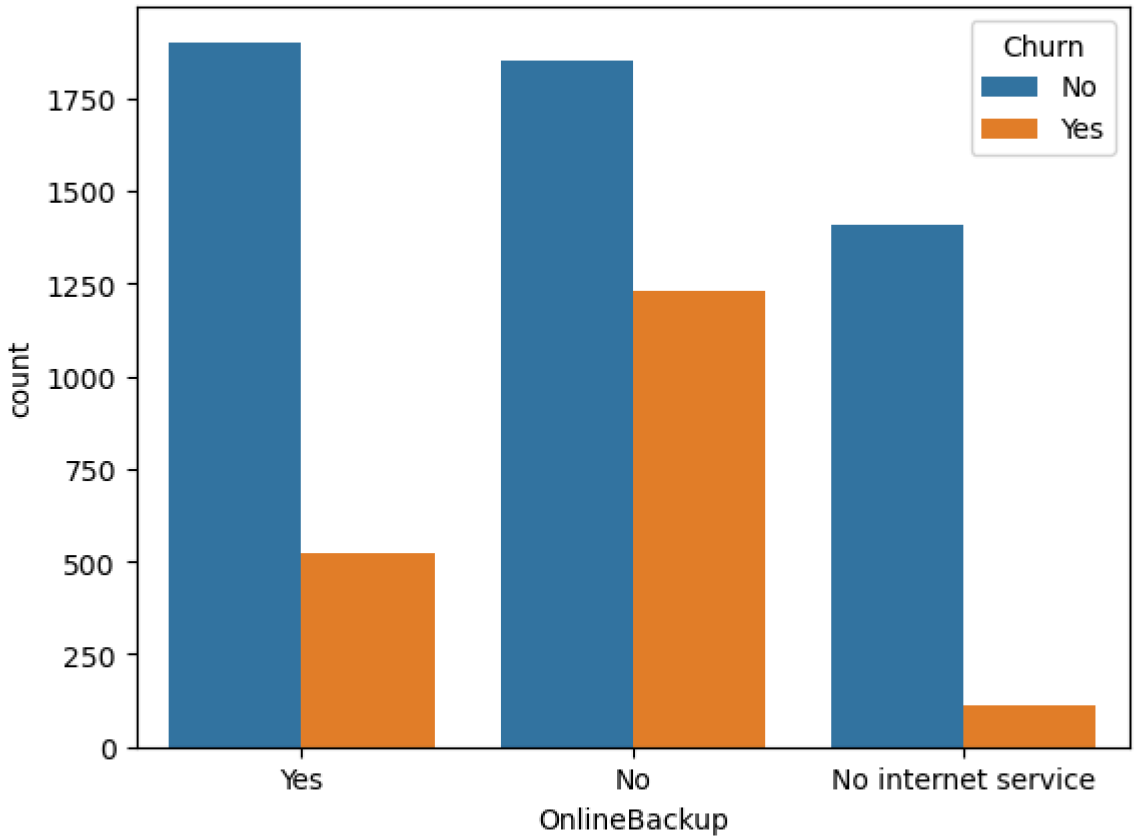
Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

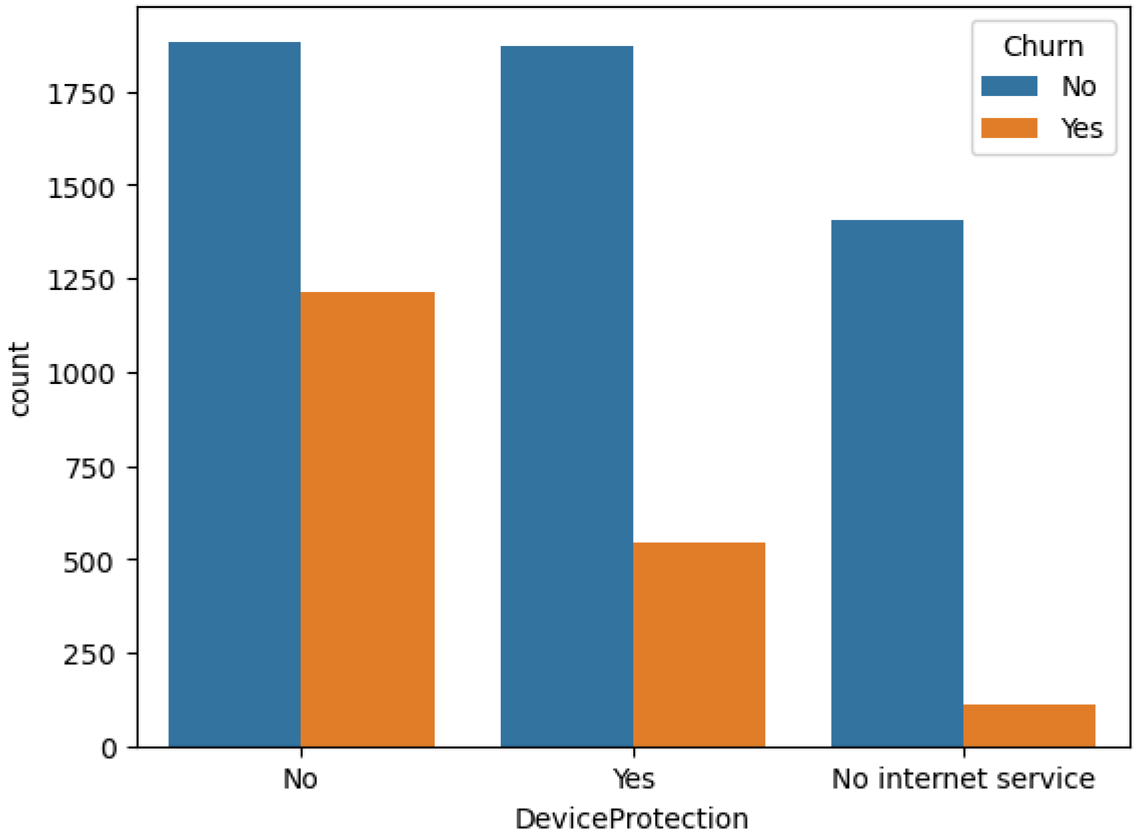
Churn
No
Yes
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines


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

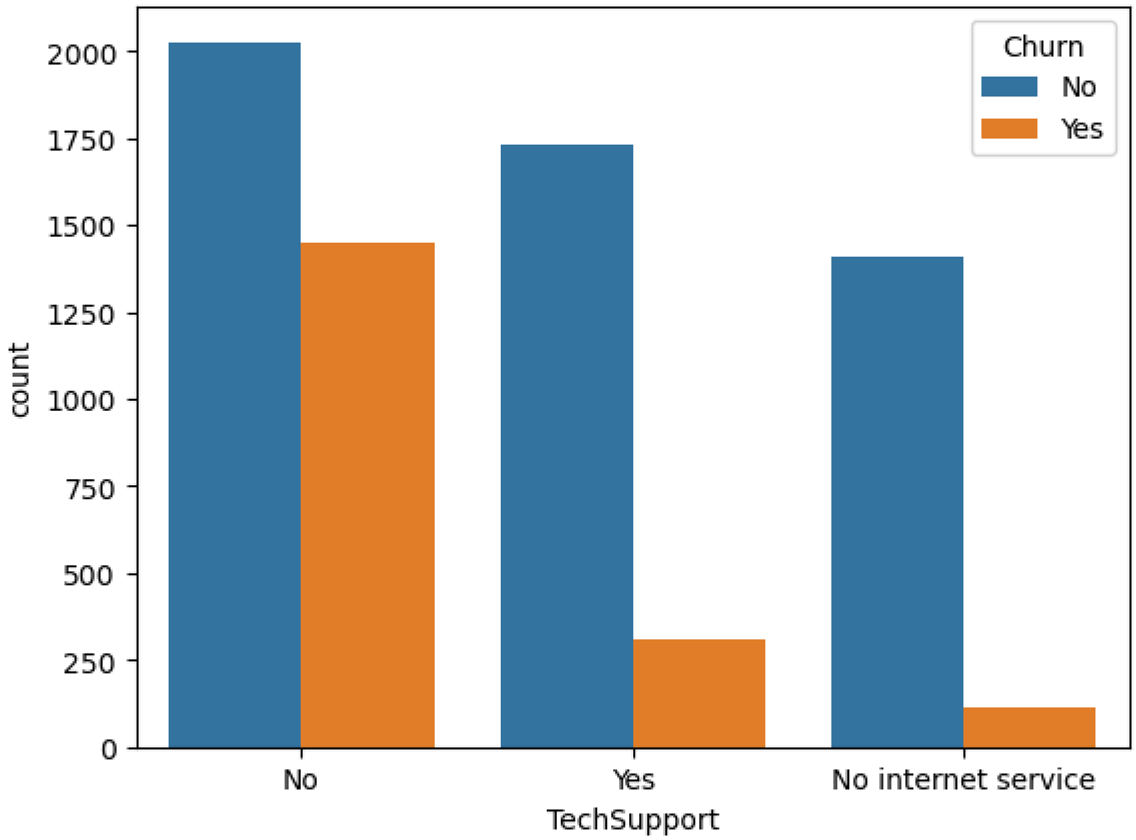

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

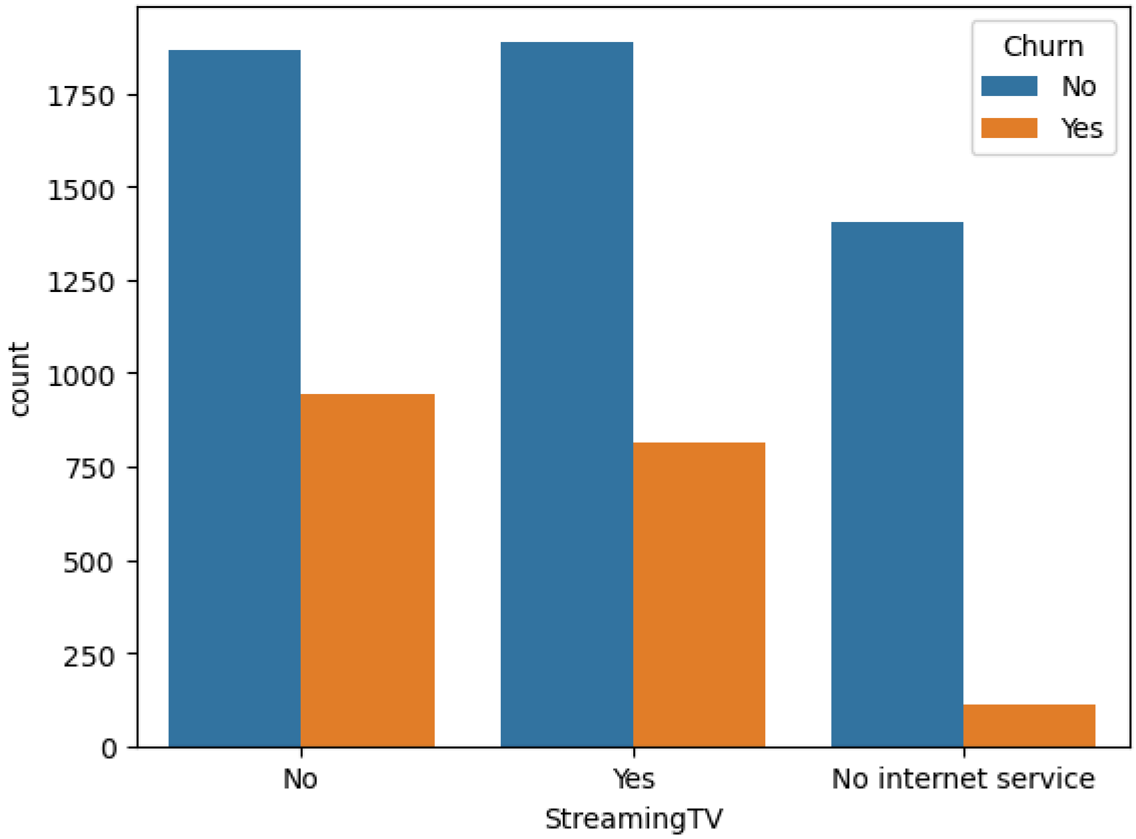


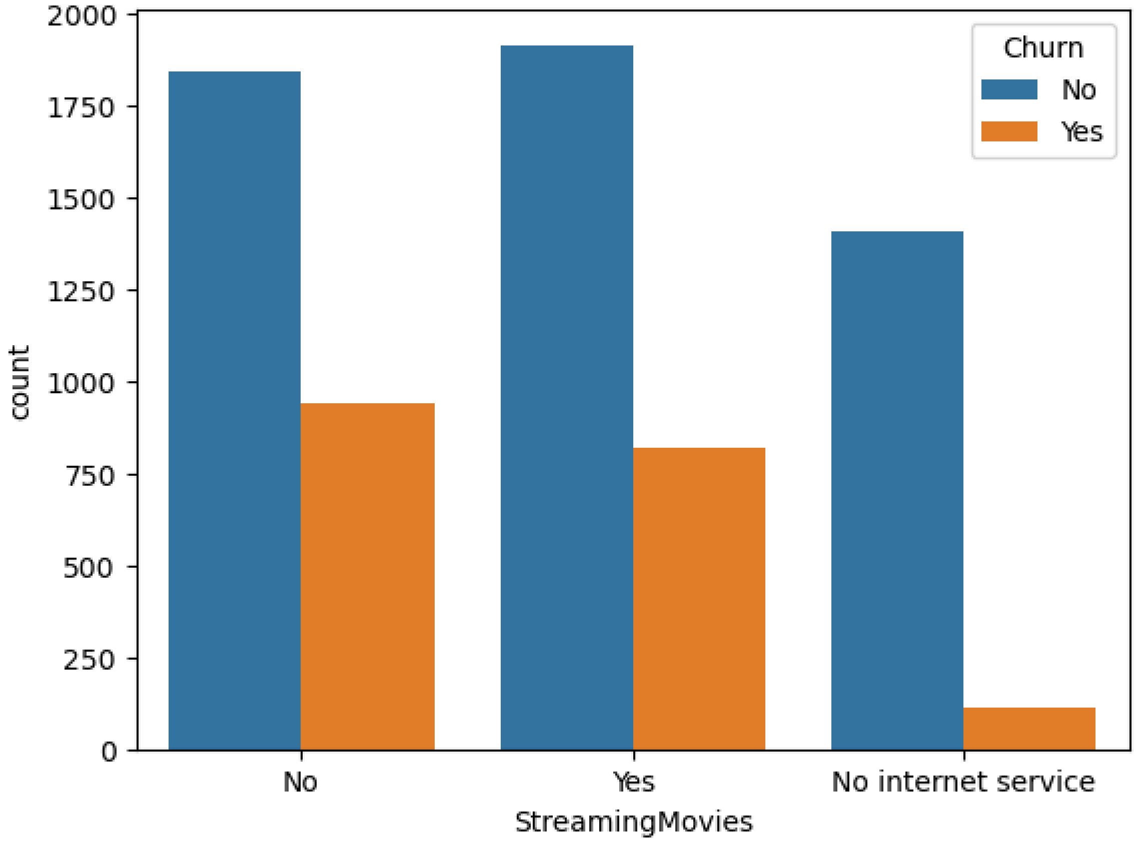
Churn
No
Yes
count
0
250
500
750
1000
1250
1500
1750
Yes
No
No internet service
OnlineBackup

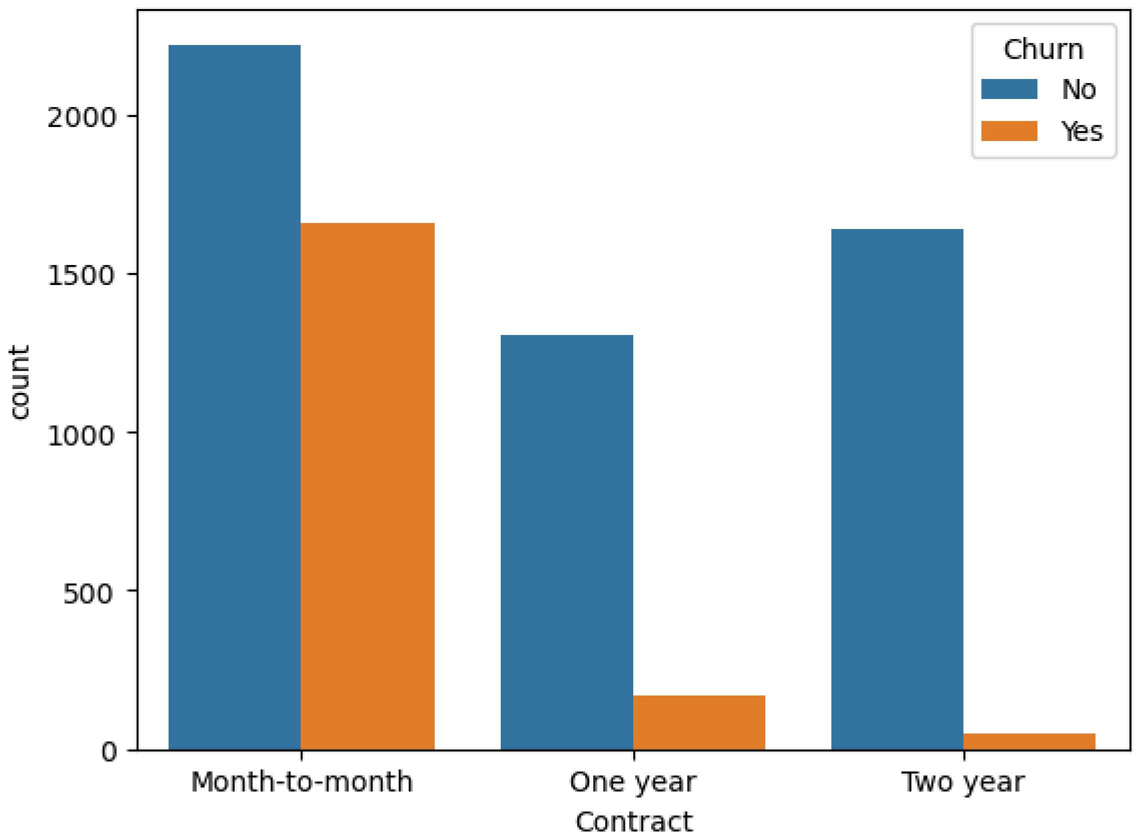



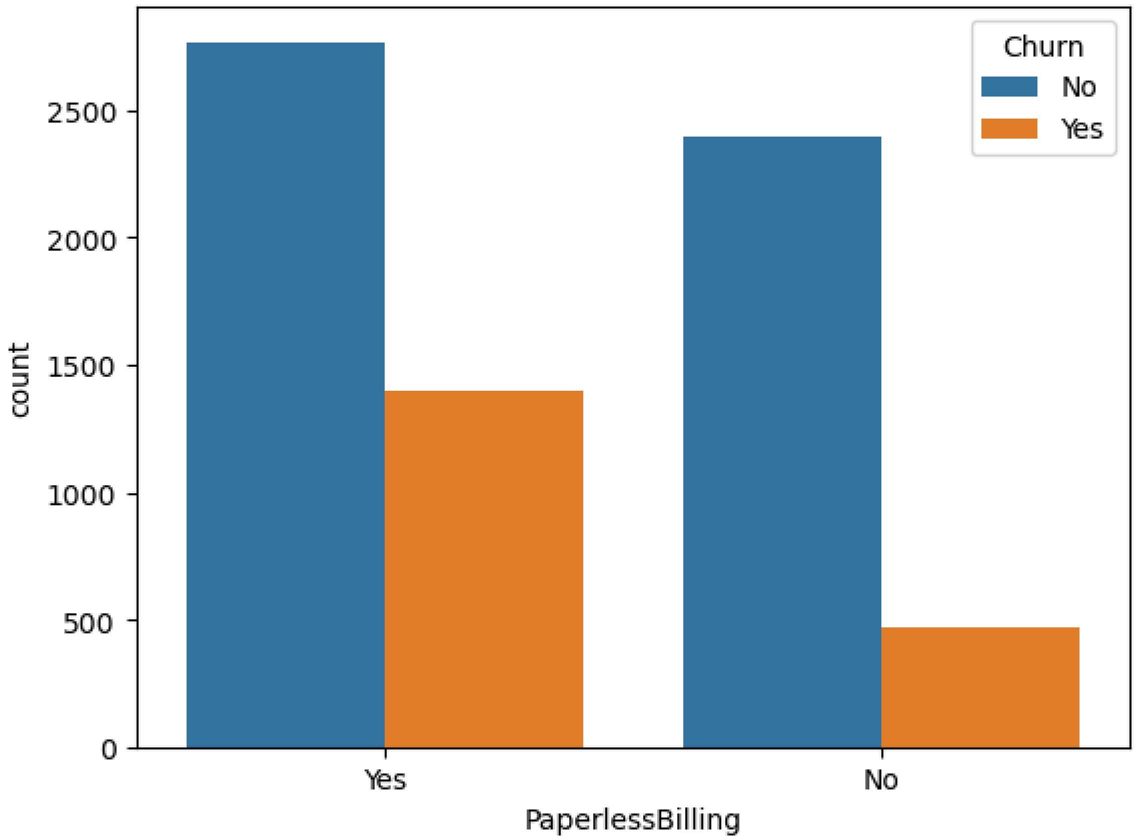
Churn
No
Yes
count
0
250
500
750
1000
1250
1500
1750
No
Yes
No internet service
DeviceProtection

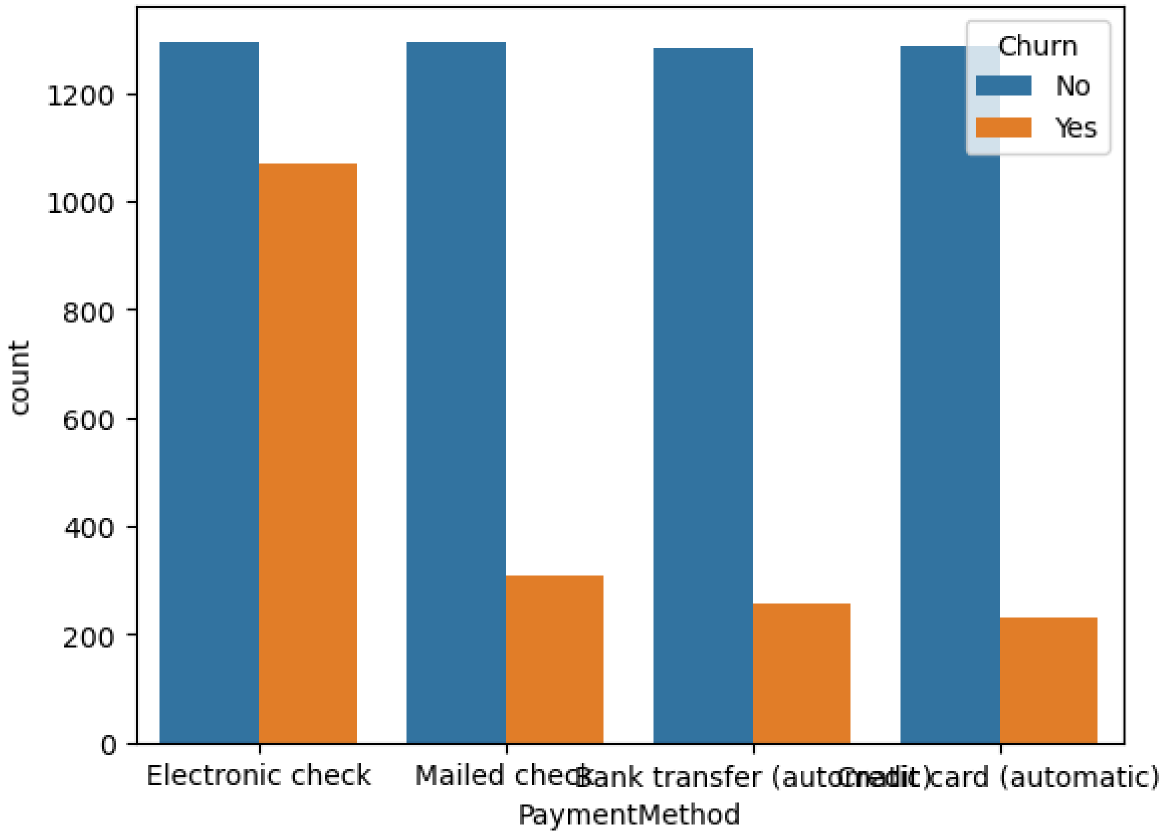

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
TechSupport

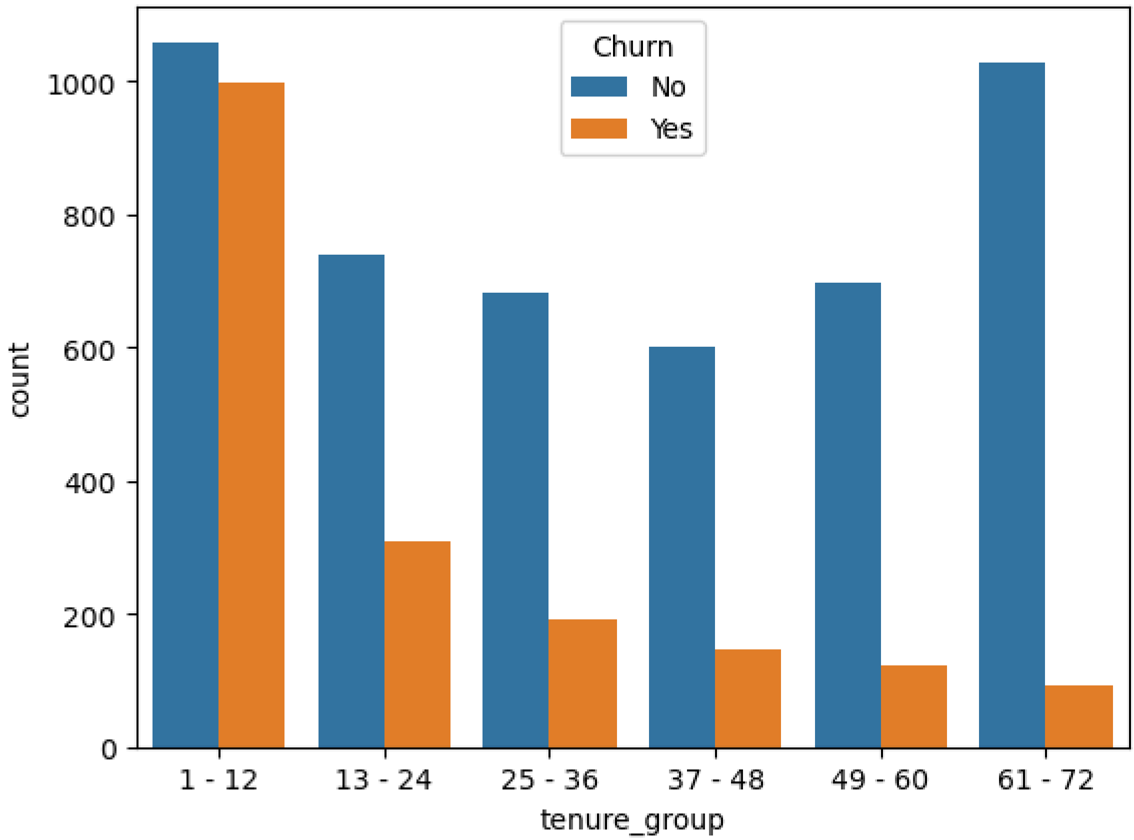

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
StreamingTV


2000
1750
1500
1250
1000
750
500
250
0
count
Churn
No
Yes
No
Yes
No internet service
StreamingMovies


2000
1500
1000
500
0
count
Churn
No
Yes
Month-to-month
One year
Two year
Contract


Churn
No
Yes
count
0
500
1000
1500
2000
2500
Yes
No
PaperlessBilling


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




1. Convert the target variable 'Churn' in a binary numeric variable i.e Yes=1, no=0

In [28]:
```
telco_data['Churn'] = np.where(telco_data.Churn == 'Yes',1,0)
```

In [29]:
```
telco_data.sample(3)
```

Out[29]:

| | gender | SeniorCitizen | Partner | Dependents | PhoneService | MultipleLines | InternetService | C |
|---|---|---|---|---|---|---|---|---|
| **3405** | Female | 0 | Yes | Yes | Yes | Yes | Fiber optic | |
| **1794** | Female | 0 | Yes | Yes | Yes | Yes | DSL | |
| **2161** | Female | 0 | No | No | Yes | No | Fiber optic | |

In [30]:
```
telco_data.dtypes
```

Out[30]:
```
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

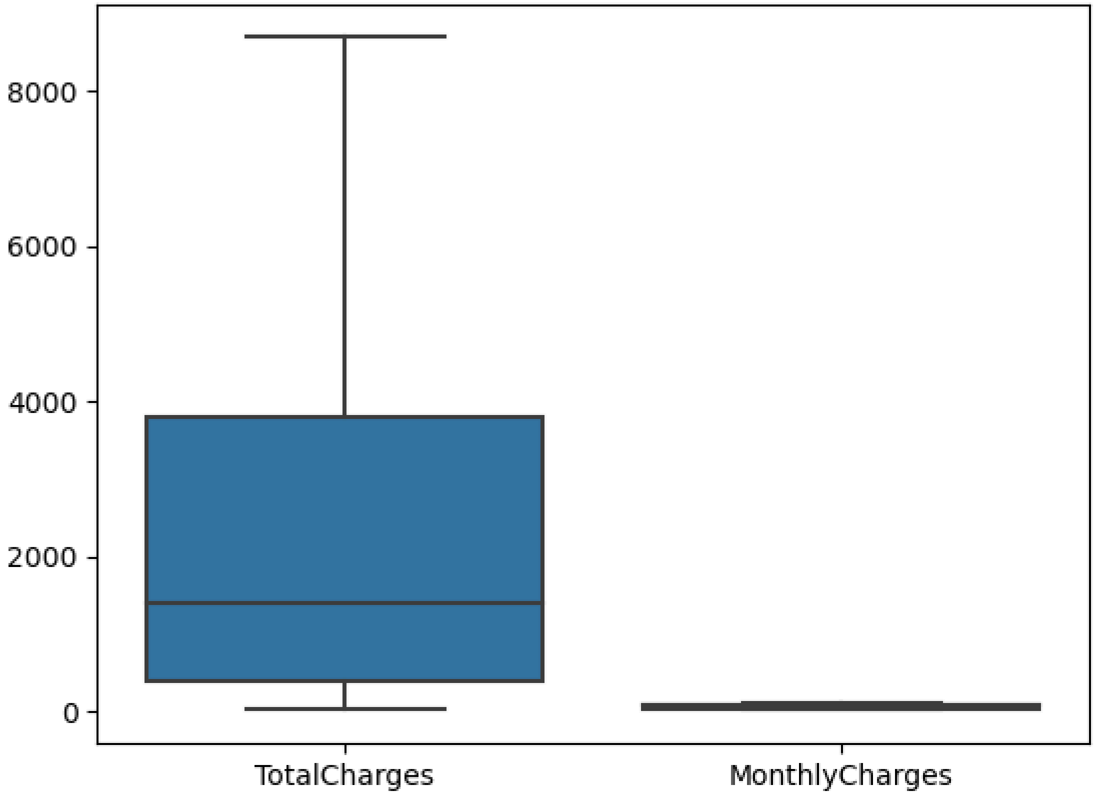
In [31]:
```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

le
```

Out[31]:
```
▾ LabelEncoder
▶ Parameters
```

In [32]:
```
categ=['gender','SeniorCitizen', 'tenure_group' ,'Partner', 'Dependents', 'PhoneSer
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod',  'Churn',]

telco_data[categ] = telco_data[categ].apply(le.fit_transform)
```

In [33]:
```
sns.boxplot(data=telco_data[['TotalCharges', 'MonthlyCharges']])
```

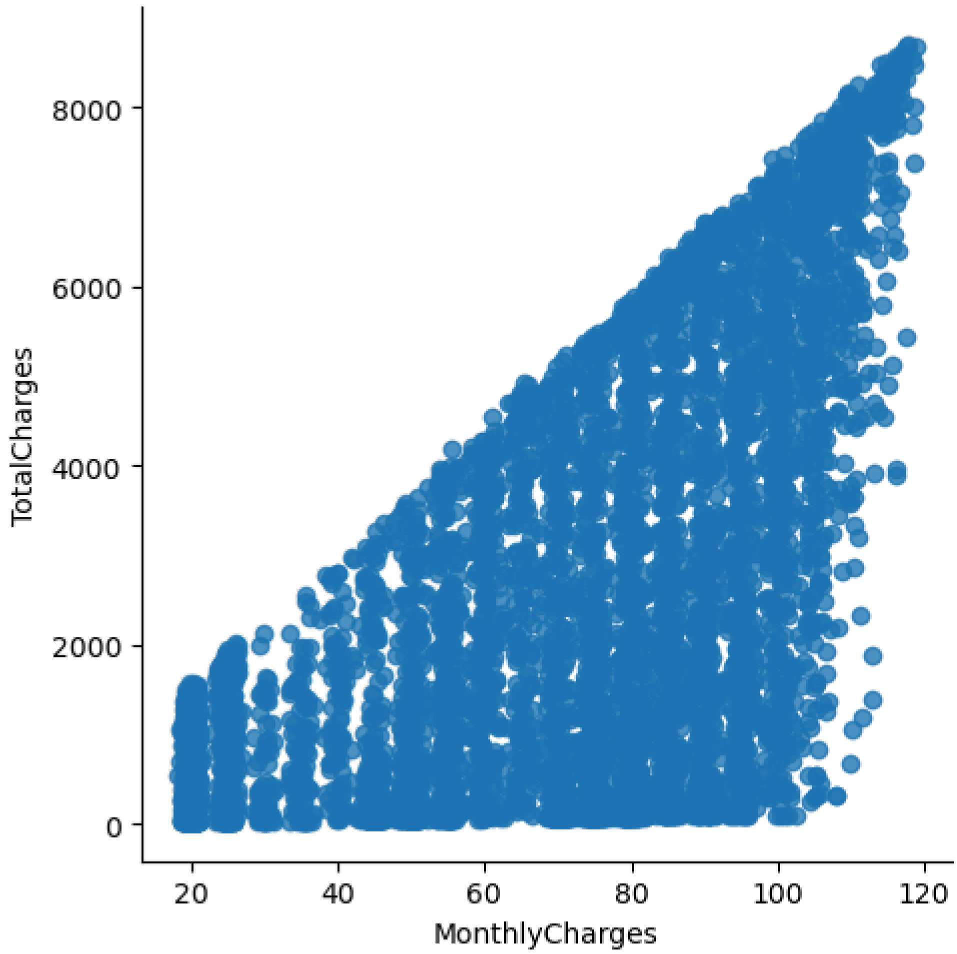
Out[33]:
```
<Axes: >
```




In [34]:
```
sns.lmplot(data=telco_data, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```

```
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[34]:
```
<seaborn.axisgrid.FacetGrid at 0x24fc60118d0>
```

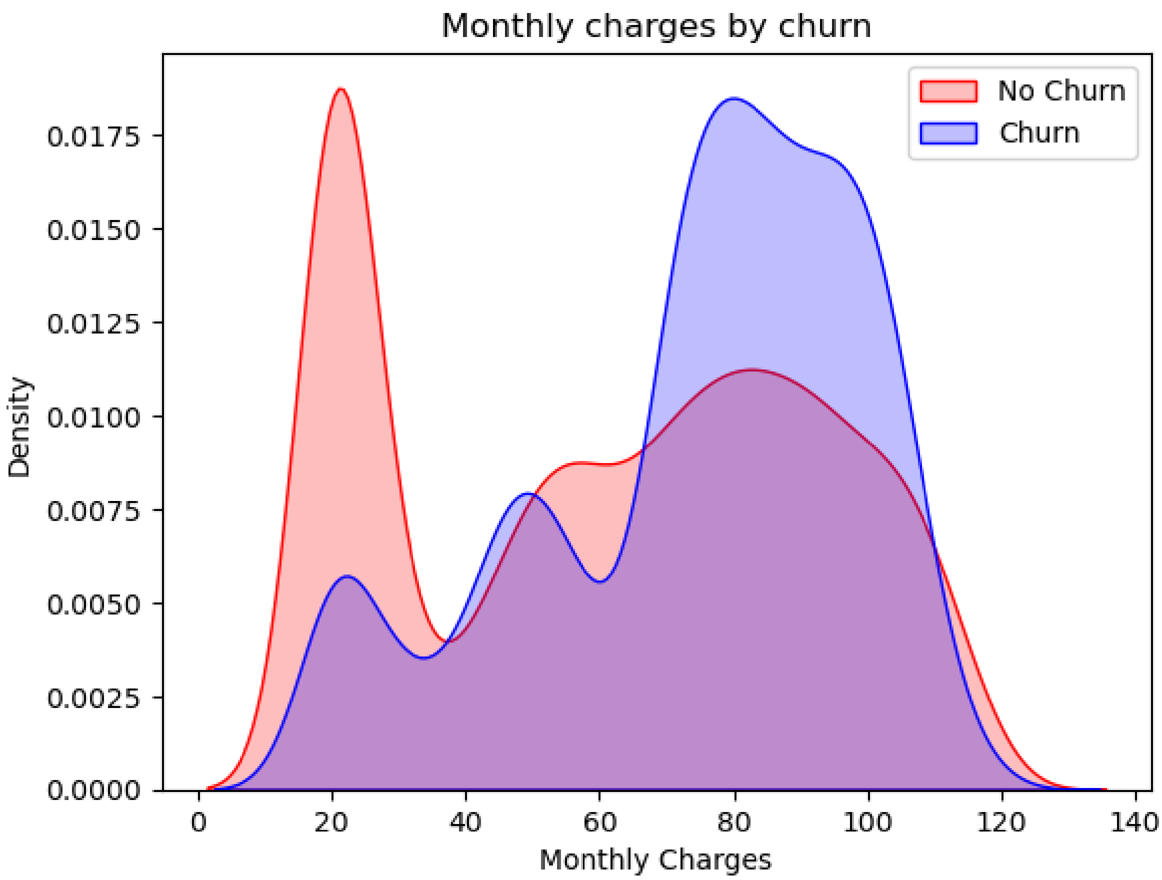



In [35]:

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

```
C:\Users\Prachi\AppData\Local\Temp\ipykernel_16244\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\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarnin
g: use_inf_as_na option is deprecated and will be removed in a future version. Con
vert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Prachi\AppData\Local\Temp\ipykernel_16244\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) ],
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarnin
g: use_inf_as_na option is deprecated and will be removed in a future version. Con
vert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[35]: Text(0.5, 1.0, 'Monthly charges by churn')

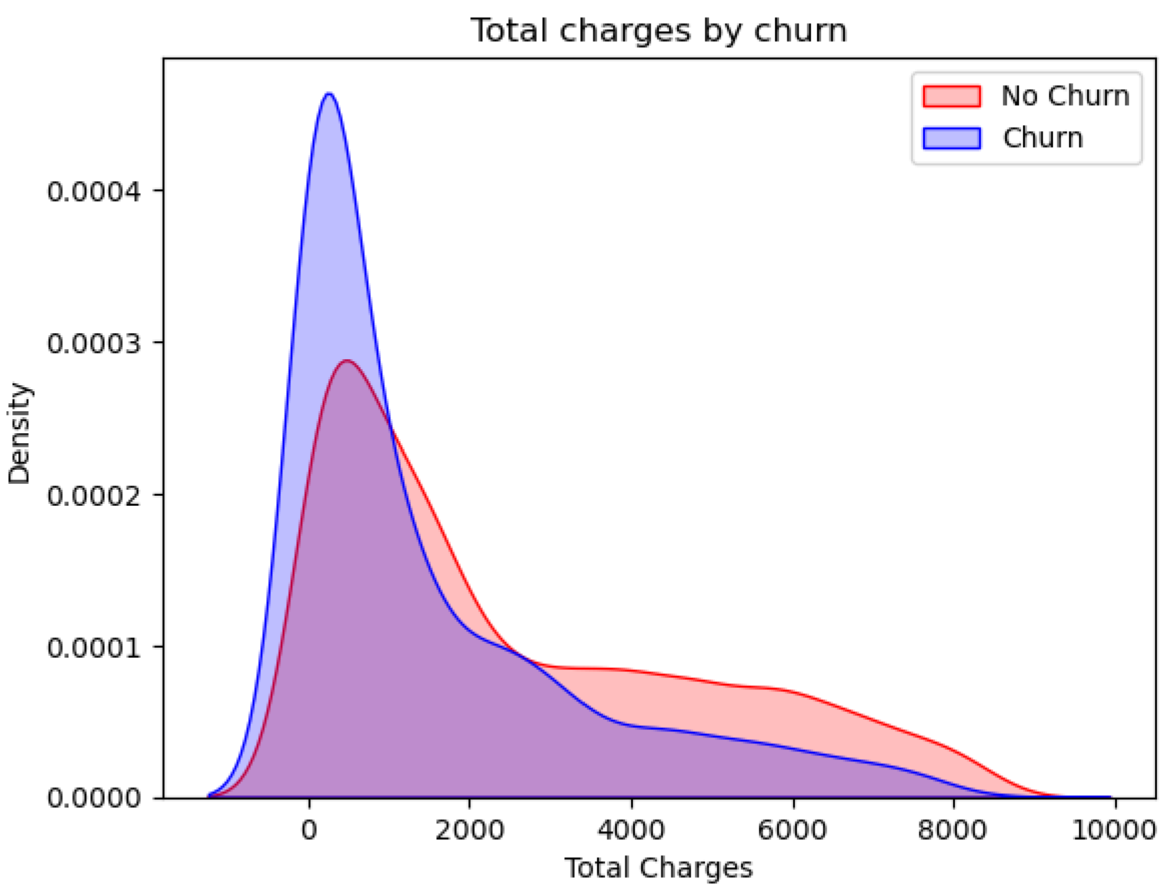



# Churn is High when Monthly charges are high

In [36]:
```
Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"]==0)],color="Red", sh
Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 1) ],
                ax =Tot, color="Blue", shade= True)
Tot.legend(["No Churn","Churn"],loc='upper right')
Tot.set_ylabel('Density')
Tot.set_xlabel('Total Charges')
Tot.set_title('Total charges by churn')
```

```
C:\Users\Prachi\AppData\Local\Temp\ipykernel_16244\2506797266.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)],color="Red",
shade=True)
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarnin
g: use_inf_as_na option is deprecated and will be removed in a future version. Con
vert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Prachi\AppData\Local\Temp\ipykernel_16244\2506797266.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.

  Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 1) ],
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarnin
g: use_inf_as_na option is deprecated and will be removed in a future version. Con
vert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[36]: Text(0.5, 1.0, 'Total charges by churn')

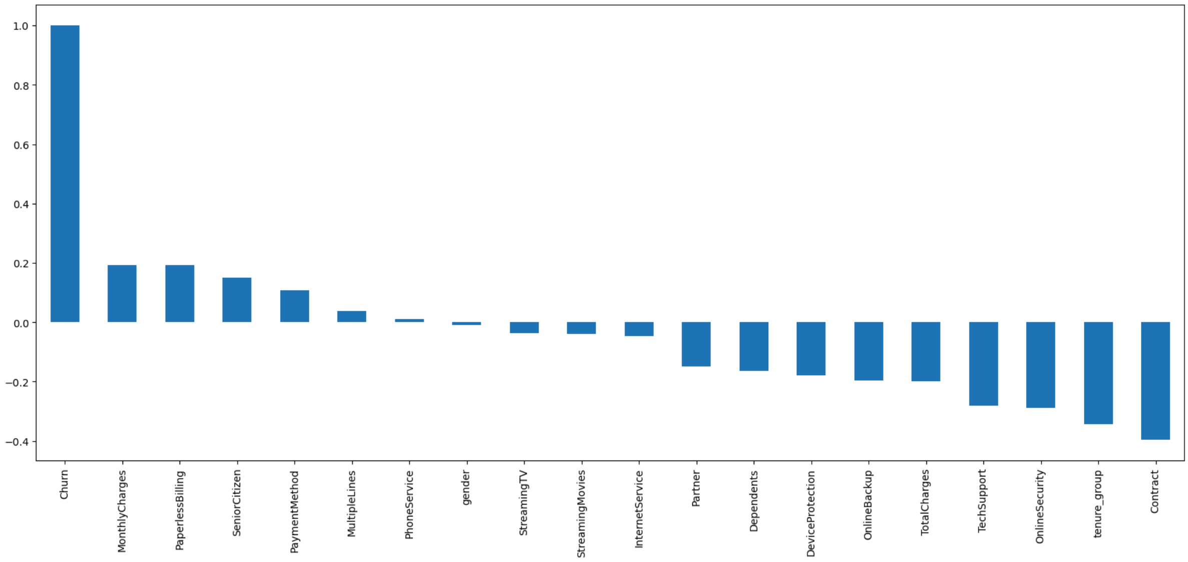



# Build a corelation of all predictors with 'Churn'

In [37]:

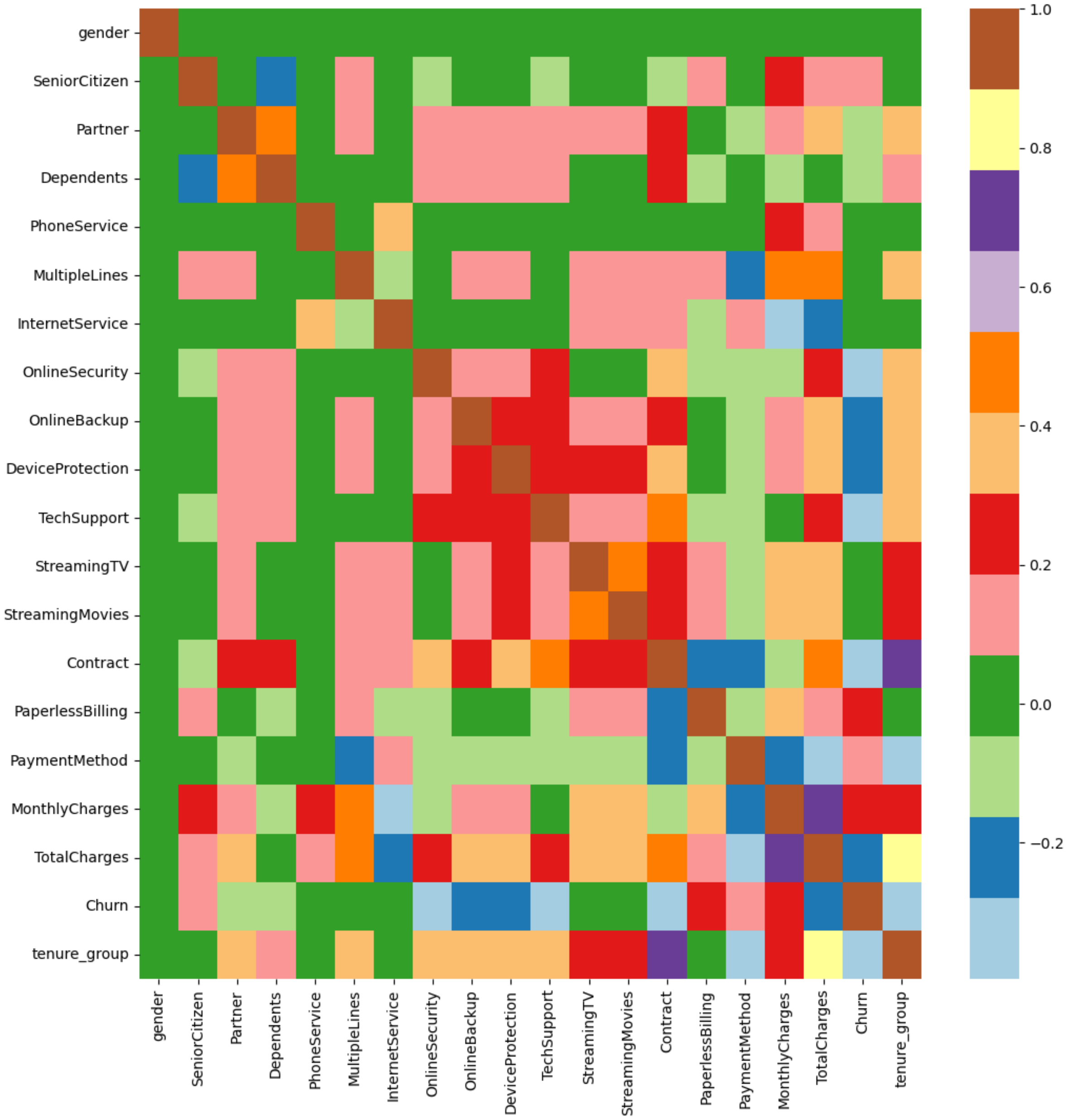
```
plt.figure(figsize=(20,8))
telco_data.corr()['Churn'].sort_values(ascending=False).plot(kind='bar')
```

Out[37]: <Axes: >




In [38]:
```
plt.figure(figsize=(12,12))
sns.heatmap(telco_data.corr(), cmap="Paired")
```

Out[38]: <Axes: >

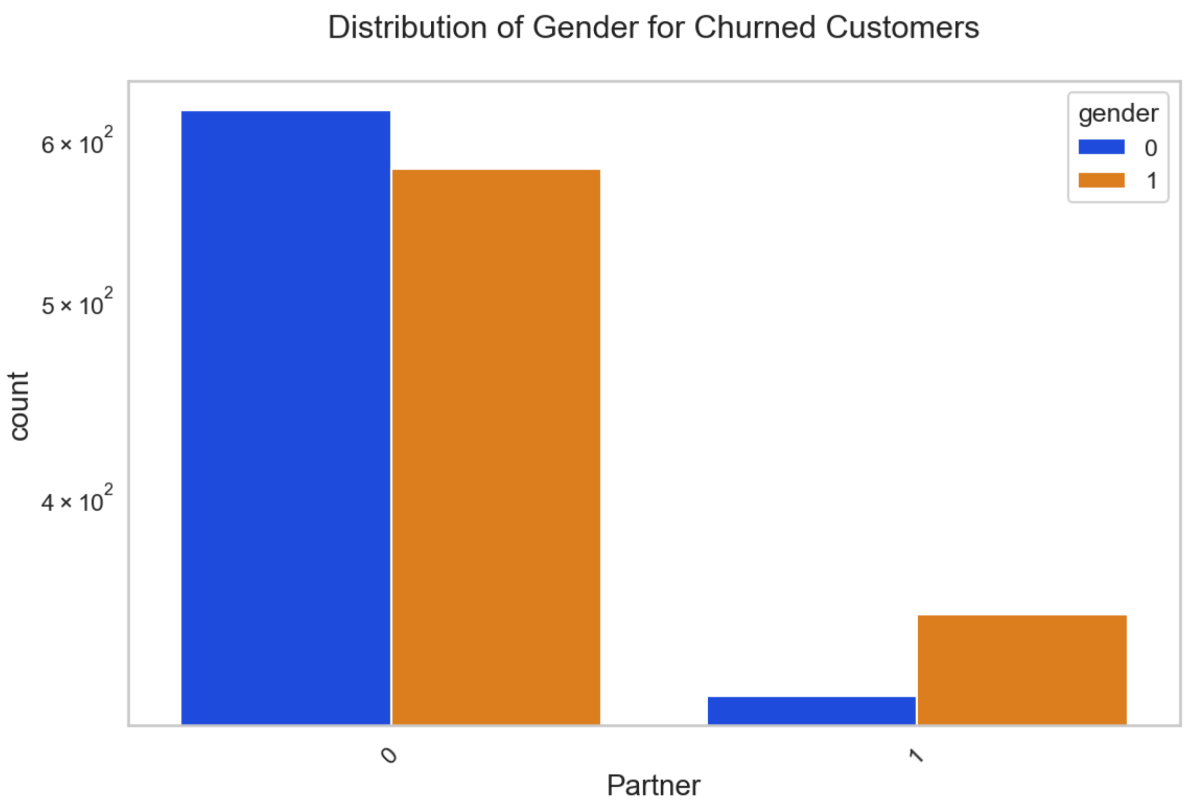


# Bivariate Analysis

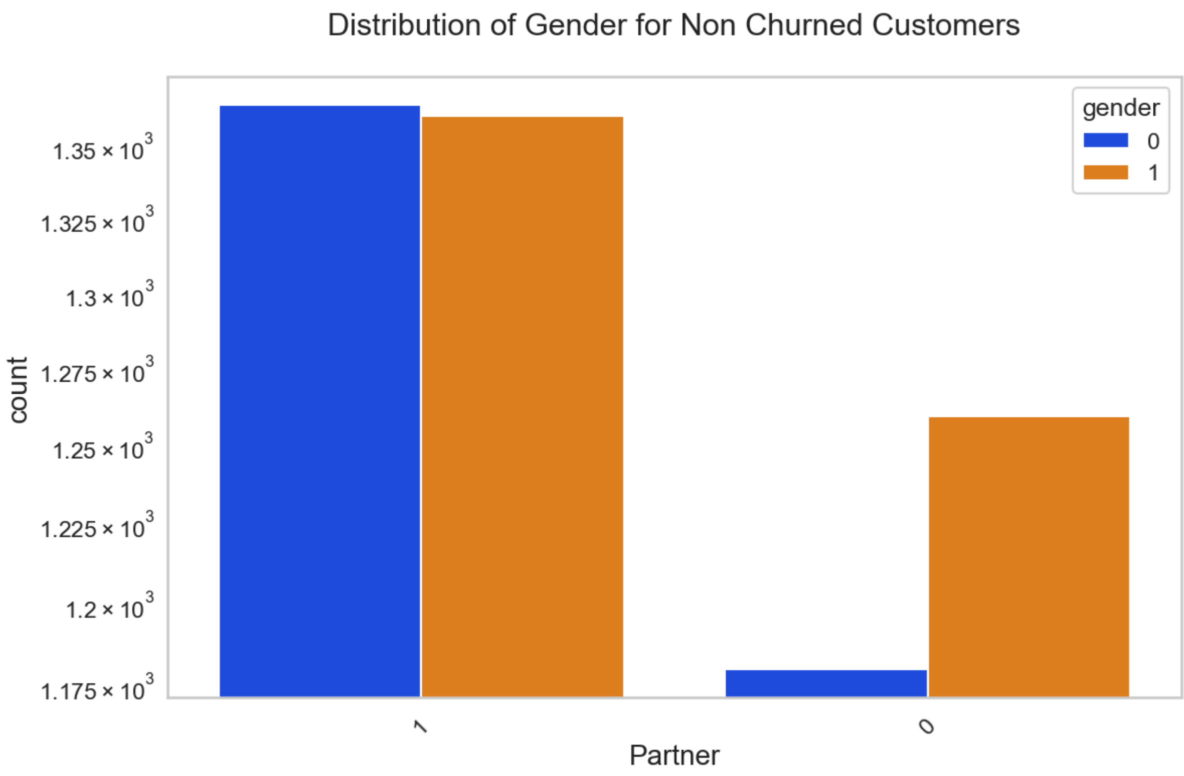
In [39]:
```
new_df1_target0 = telco_data.loc[telco_data["Churn"]==0]
new_df1_target1 = telco_data.loc[telco_data["Churn"]==1]
```

In [40]:
```
def uniplot(df,col,title,hue =None):

    sns.set_style('whitegrid')
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 20
    plt.rcParams['axes.titlesize'] = 22
    plt.rcParams['axes.titlepad'] = 30


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(df[col].unique()) + 7 + 4*len(temp.unique())
    fig.set_size_inches(width , 8)
    plt.xticks(rotation=45)
    plt.yscale('log')
    plt.title(title)
    ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue =

    plt.show()
```

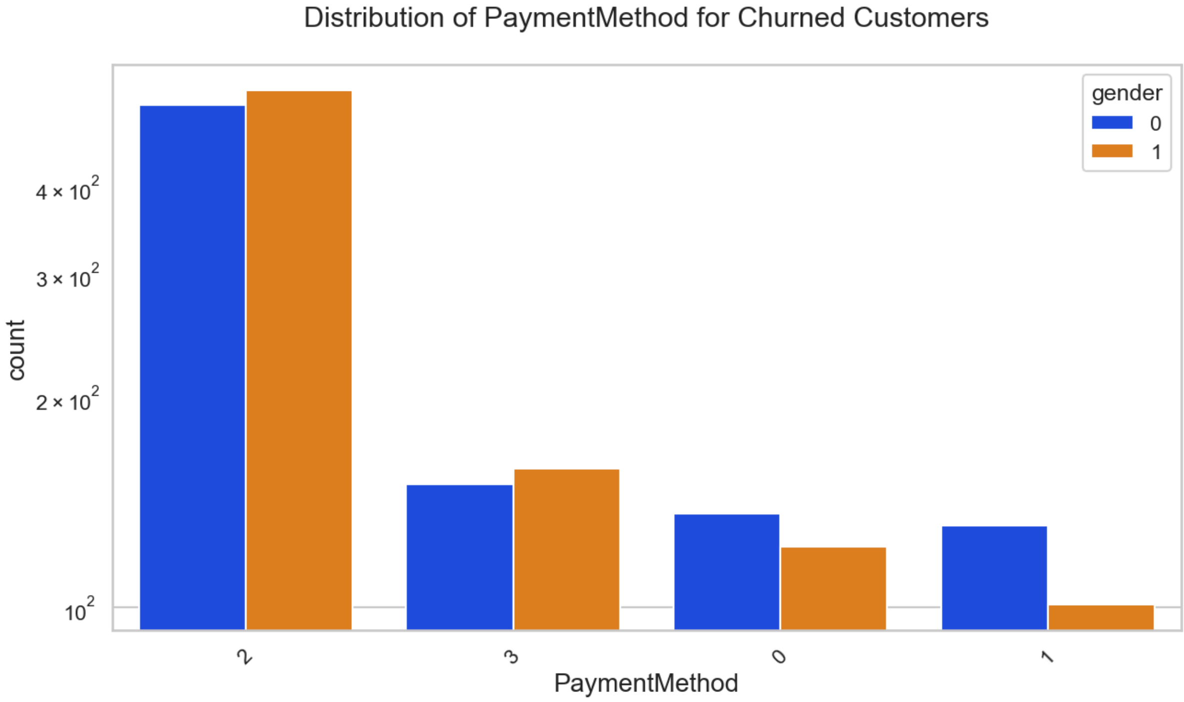
In [41]:
```
uniplot(new_df1_target1,col='Partner',title='Distribution of Gender for Churned Cus
```



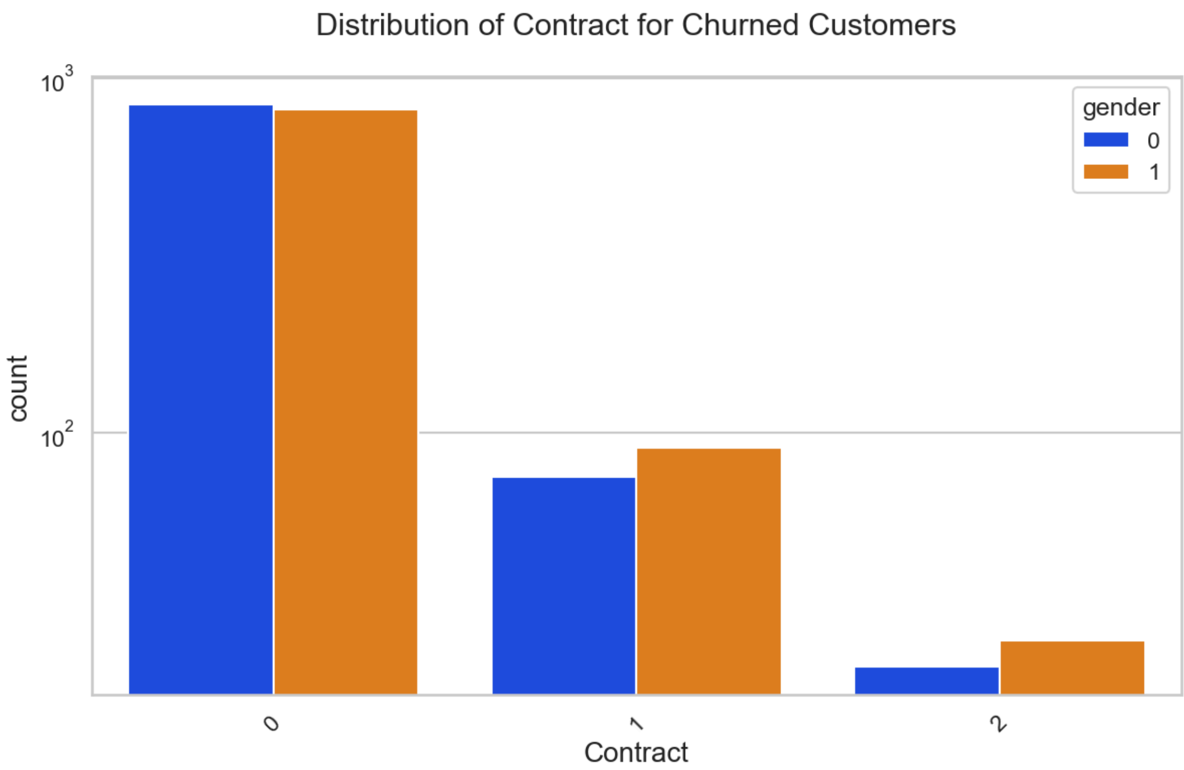

In [42]:
```
uniplot(new_df1_target0,col='Partner',title='Distribution of Gender for Non Churned
```



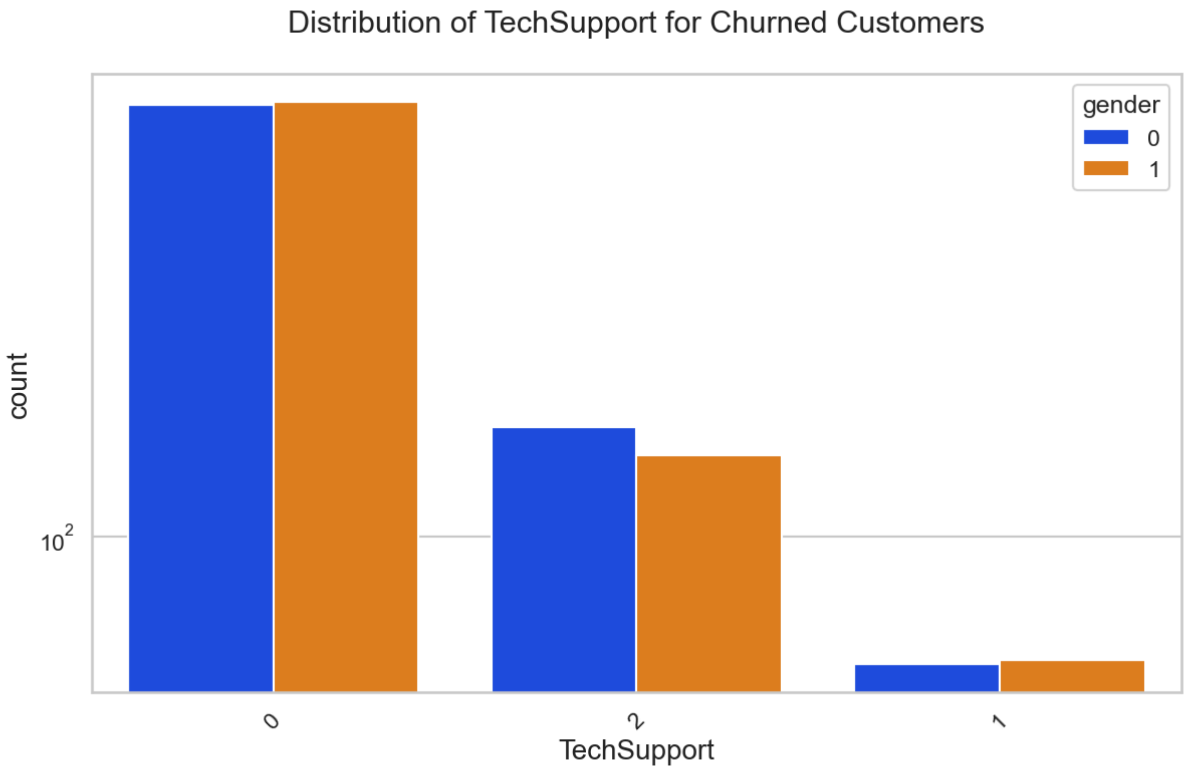

In [43]:
```
uniplot(new_df1_target1,col='PaymentMethod',title='Distribution of PaymentMethod fo
```



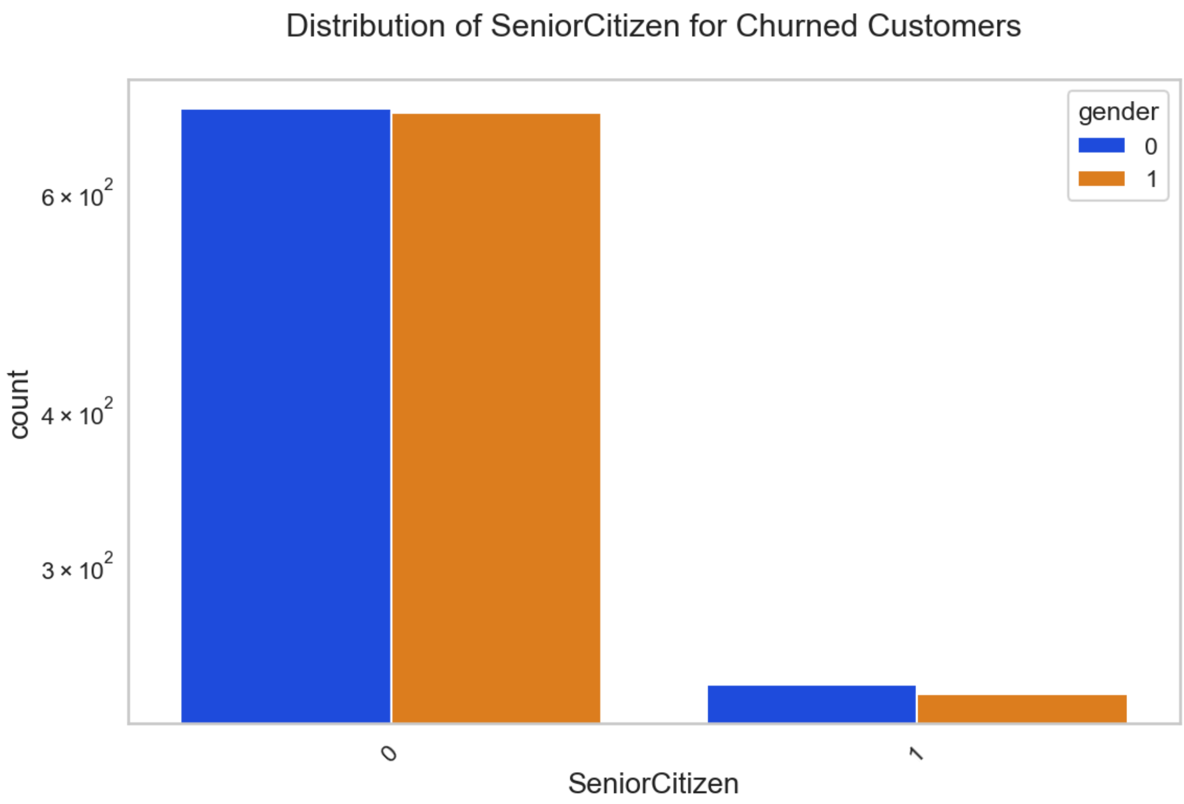

In [44]:
```
uniplot(new_df1_target1,col='Contract',title='Distribution of Contract for Churned
```




In [45]:
```
uniplot(new_df1_target1,col='TechSupport',title='Distribution of TechSupport for Ch
```




In [46]:
```
uniplot(new_df1_target1,col='SeniorCitizen',title='Distribution of SeniorCitizen fo
```




In [47]:
```python
X = telco_data.drop('Churn',axis=1)
y = telco_data['Churn']
```

In [48]:
```python
X
```

Out[48]:

| | gender | SeniorCitizen | Partner | Dependents | PhoneService | MultipleLines | InternetService | C |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| **1** | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| **2** | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| **3** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | |
| **4** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | 1 | 0 | 1 | 1 | 1 | 2 | 0 | |
| **7039** | 0 | 0 | 1 | 1 | 1 | 2 | 1 | |
| **7040** | 0 | 0 | 1 | 1 | 0 | 1 | 0 | |
| **7041** | 1 | 1 | 1 | 0 | 1 | 2 | 1 | |
| **7042** | 1 | 0 | 0 | 0 | 1 | 0 | 1 | |

7032 rows × 19 columns

In [49]:
```python
telco_data['Churn'].value_counts()/len(telco_data) #data is highly imbalancing
```

Out[49]:
```
Churn
0    0.734215
1    0.265785
Name: count, dtype: float64
```

# Train Test Split

In [50]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_stat
```

In [51]:
```
print('Training data shape')

print(X_train.shape)
print(y_train.shape)

print('Testing Data shape')

print(X_test.shape)
print(y_test.shape)
```

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

In [52]:
```
print(y_test.value_counts())

print(y_train.value_counts())
```

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

In [53]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [54]:
```
model_dtc=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6,
```

In [55]:
```
model_dtc.fit(X_train,y_train)
```

Out[55]:

▾ DecisionTreeClassifier

▶ Parameters

In [56]:
```
model_dtc.score(X_test,y_test)
```

Out[56]:
```
0.7619047619047619
```

In [57]:
```
y_pred = model_dtc.predict(X_test)
y_pred[:10]
```

Out[57]:
```
array([0, 0, 1, 0, 0, 1, 0, 1, 0, 0], dtype=int64)
```

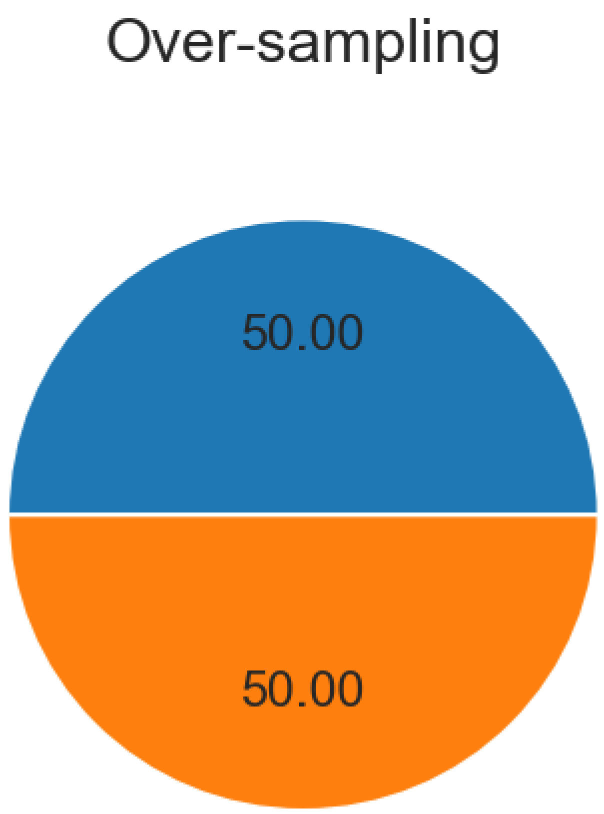
In [58]:
```
print(y_test[:10])
```

```
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 [59]:
```
from sklearn.metrics import classification_report
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
```

In [60]:
```
from imblearn.over_sampling import SMOTE

smote=SMOTE()

X_ovs,y_ovs=smote.fit_resample(X,y)



fig, oversp = plt.subplots()
oversp.pie( y_ovs.value_counts(), autopct='%.2f')
oversp.set_title("Over-sampling")
plt.show()
```




In [61]:
```
Xr_train , Xr_test, yr_train, yr_test, = train_test_split(X_ovs, y_ovs, test_size=0
```

In [64]:
```
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression(max_iter=1000)
```

In [65]:
```
model_lr.fit(Xr_train, yr_train)
```

```
c:\Users\Prachi\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:473:
ConvergenceWarning: lbfgs failed to converge after 1000 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[65]:

▾ LogisticRegression

▶ Parameters

In [66]:
```
y_pred=model_lr.predict(Xr_test)
y_pred[:10]
```

Out[66]:
```
array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [67]:
```
model_lr.score(Xr_test,yr_test)
```

Out[67]:
```
0.8049370764762827
```

In [69]:
```python
from sklearn.metrics import accuracy_score, classification_report

report = classification_report(y_pred, yr_test, labels=[0,1])

print(report)
```

```
              precision    recall  f1-score   support

           0       0.78      0.83      0.80       974
           1       0.83      0.79      0.81      1092

    accuracy                           0.80      2066
   macro avg       0.81      0.81      0.80      2066
weighted avg       0.81      0.80      0.81      2066
```

In [71]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(yr_test,y_pred)
```

Out[71]:
```
array([[804, 233],
       [170, 859]], dtype=int64)
```

# Decision Tree Classifier

In [72]:
```python
from sklearn.tree import DecisionTreeClassifier

model_dtc = DecisionTreeClassifier(criterion = "gini", random_state=100, max_depth=
```

In [73]:
```python
model_dtc.fit(Xr_train, yr_train)
```

Out[73]:
```
▾ DecisionTreeClassifier

▶ Parameters
```

In [74]:
```python
y_pred = model_dtc.predict(Xr_test)
y_pred[:10]
```

Out[74]:
```
array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [75]:
```python
yr_test[:10]
```

Out[75]:
```
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 [76]:
```python
model_dtc.score(Xr_test,yr_test)
```

Out[76]:
```
0.7879961277831559
```

In [77]:
```python
print(classification_report(yr_test,y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.82      0.74      0.78      1037
           1       0.76      0.84      0.80      1029

    accuracy                           0.79      2066
   macro avg       0.79      0.79      0.79      2066
weighted avg       0.79      0.79      0.79      2066
```

In [79]:
```
confusion_matrix(yr_test,y_pred)
```

Out[79]:
```
array([[768, 269],
       [169, 860]], dtype=int64)
```

# Random Forest Classifier

In [80]:
```
from sklearn.ensemble import RandomForestClassifier

model_rfc = RandomForestClassifier(n_estimators=100, random_state=100, max_depth=6,
```

In [81]:
```
model_rfc.fit(Xr_train,yr_train)
```

Out[81]:
```
▾ RandomForestClassifier

▶ Parameters
```

In [82]:
```
y_pred=model_rfc.predict(Xr_test)
y_pred[:10]
```

Out[82]:
```
array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [83]:
```
yr_test[:10]
```

Out[83]:
```
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 [84]:
```
model_rfc.score(Xr_test, yr_test)
```

Out[84]:
```
0.8160696999031946
```

In [85]:
```
report_rfc = classification_report(y_pred, yr_test)
print(report_rfc)
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       947
           1       0.86      0.79      0.82      1119

    accuracy                           0.82      2066
   macro avg       0.82      0.82      0.82      2066
weighted avg       0.82      0.82      0.82      2066
```

In [86]:
```
confusion_matrix(yr_test,y_pred)
```

Out[86]:
```
array([[802, 235],
       [145, 884]], dtype=int64)
```

# AdaBoost

In [87]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [88]:
```
model_abc=AdaBoostClassifier(n_estimators=100)
```

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

Out[89]:
```
▾ AdaBoostClassifier
▶ Parameters
```

In [90]:
```
y_pred = model_abc.predict(Xr_test)
```

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

```
              precision    recall  f1-score   support

           0       0.76      0.86      0.81       924
           1       0.87      0.79      0.83      1142

    accuracy                           0.82      2066
   macro avg       0.82      0.82      0.82      2066
weighted avg       0.82      0.82      0.82      2066
```

In [92]:
```
confusion_matrix(yr_test,y_pred)
```

Out[92]:
```
array([[793, 244],
       [131, 898]], dtype=int64)
```

# GradientBoostingClassifier

In [93]:
```
from sklearn.ensemble import GradientBoostingClassifier
model_gbc = GradientBoostingClassifier()
model_gbc
```

Out[93]:
```
▾ GradientBoostingClassifier
▶ Parameters
```

In [94]:
```
model_gbc.fit(Xr_train, yr_train)
```

Out[94]:
```
▾ GradientBoostingClassifier
▶ Parameters
```

In [95]:
```
y_pred_gbc = model_gbc.predict(Xr_test)
y_pred_gbc[:10]
```

Out[95]:
```
array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [96]:
```
yr_test[:10]
```

Out[96]:
```
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 [97]:
```
print(classification_report(y_pred_gbc,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       973
           1       0.86      0.81      0.83      1093

    accuracy                           0.83      2066
   macro avg       0.83      0.83      0.83      2066
weighted avg       0.83      0.83      0.83      2066
```

In [98]:
```
confusion_matrix(yr_test, y_pred)
```

Out[98]:
```
array([[793, 244],
       [131, 898]], dtype=int64)
```

# Xgboost

In [99]:
```
from xgboost import XGBClassifier

model_xgb = XGBClassifier(class_weight={0:1, 1:2})

model_xgb
```

Out[99]:
```
▾ XGBClassifier
▶ Parameters
```

In [100…
```
model_xgb.fit(Xr_train,yr_train)
```

Out[100]:

```
▾ XGBClassifier
▶ Parameters
```

In [102…

```
y_pred = model_xgb.predict(Xr_test)
y_pred[:10]
```

Out[102]:

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

In [103…

```
yr_test[:10]
```

Out[103]:

```
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 [104…

```
print(classification_report(y_pred,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      1015
           1       0.85      0.83      0.84      1051

    accuracy                           0.84      2066
   macro avg       0.84      0.84      0.84      2066
weighted avg       0.84      0.84      0.84      2066
```

In [105…

```
from sklearn.metrics import confusion_matrix

# Assuming y_pred and y_test are your predicted and true labels respectively

cm = confusion_matrix(yr_test, y_pred)

print("confusion_matrix:")
print(cm)
```

```
confusion_matrix:
[[858 179]
 [157 872]]
```

In [106…

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier
import time

# Define your GradientBoostingClassifier and param_dist
model = GradientBoostingClassifier()
param_dist = {
    'learning_rate': [0.1, 0.5, 1.0],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],  # Example: Adding max_depth parameter
    'min_samples_split': [2, 5, 10]  # Example: Adding min_samples_split parameter
}

# Create RandomizedSearchCV object with fewer iterations
```

```python
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist,

# Start the timer
start_time = time.time()

# Fit the RandomizedSearchCV object
random_search.fit(Xr_train, yr_train)

# Stop the timer
end_time = time.time()

# Calculate the total time taken
total_time = end_time - start_time

print("RandomizedSearchCV took {:.2f} seconds to complete.".format(total_time))

# Get the best parameters
best_params = random_search.best_params_
print("Best Parameters:", best_params)
```

```
RandomizedSearchCV took 72.19 seconds to complete.
Best Parameters: {'n_estimators': 100, 'min_samples_split': 5, 'max_depth': 7, 'le
arning_rate': 0.1}
```

# final model

In [107…

```python
from sklearn.ensemble import GradientBoostingClassifier

# Define the best hyperparameters obtained from GridSearchCV
best_params = {
    'n_estimators': 100, 'min_samples_split':5 , 'max_depth': 7, 'learning_rate': 0.



}

# Create Gradient Boosting Classifier with the best hyperparameters
final_gb_classifier = GradientBoostingClassifier(**best_params)

# Train the final model on the entire training data
final_gb_classifier.fit(Xr_train, yr_train)
```

Out[107]:

```
▾ GradientBoostingClassifier

▶ Parameters
```

In [108…

```python
from sklearn.model_selection import cross_val_score

# trained model with tuned hyperparameters
# X_train and y_train are your training data
# cv=10 indicates 10-fold cross-validation
cv_scores = cross_val_score(final_gb_classifier, Xr_train, yr_train, cv=10, scoring

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV score:", cv_scores.mean())
```

```
Cross-validation scores: [0.84261501 0.86440678 0.83898305 0.86077482 0.8559322
0.8559322
 0.83656174 0.84140436 0.8377724  0.86924939]
Mean CV score: 0.8503631961259079
```

In [109…
```
y_pred=final_gb_classifier.predict(Xr_test)
y_pred[:10]
```

Out[109]:
```
array([1, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [110…
```
yr_test[:10]
```

Out[110]:
```
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 [111…
```
print(classification_report(y_pred,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83      1016
           1       0.84      0.83      0.84      1050

    accuracy                           0.83      2066
   macro avg       0.83      0.84      0.83      2066
weighted avg       0.84      0.83      0.83      2066
```

In [112…
```
confusion_matrix(y_pred,yr_test)
```

Out[112]:
```
array([[856, 160],
       [181, 869]], dtype=int64)
```

In [113…
```
import os
import pickle
from sklearn.ensemble import GradientBoostingClassifier

# Change directory if needed
os.chdir(r"C:\Users\Prachi\Documents\VS Code Files\ML CAPSTONE PROJECT\Customer Chu
# Assuming final_gb_classifier is your trained model
# Define and train Gradient Boosting Classifier
best_params = {
    'n_estimators': 100,
    'min_samples_split': 5,
    'max_depth': 7,
    'learning_rate': 0.1
}

final_gb_classifier = GradientBoostingClassifier(**best_params)

# Train the final model on the entire training data (assuming Xr_train and yr_trair
final_gb_classifier.fit(X_train, y_train)

# Dumping the model to a file
with open('final_gb_classifier.pkl', 'wb') as file:
    pickle.dump(final_gb_classifier, file)
```

```python
# Load the saved model
with open('final_gb_classifier.pkl', 'rb') as file:
    loaded_model = pickle.load(file)
```

# checking accuracy with our features

In [114…

```python
import pickle
import pandas as pd

# Load the saved model from the pickle file
with open('final_gb_classifier.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

# Prepare your own data for testing
# Create a DataFrame with your feature data
your_features = pd.DataFrame({
    'gender': [1, 0, 0, 0, 0],
    'SeniorCitizen': [0, 0, 0, 0, 0],
    'Partner': [0, 0, 0, 1, 1],
    'Dependents': [0, 0, 0, 0, 1],
    'PhoneService': [1, 0, 1, 1, 1],
    'MultipleLines': [0, 0, 0, 2, 2],
    'InternetService': [1, 0, 1, 1, 0],
    'OnlineSecurity': [0, 0, 0, 2, 2],
    'OnlineBackup': [0, 0, 1, 2, 2],
    'DeviceProtection': [0, 0, 0, 0, 2],
    'TechSupport': [0, 0, 0, 2, 2],
    'StreamingTV': [0, 1, 0, 0, 0],
    'StreamingMovies': [0, 1, 0, 0, 0],
    'Contract': [2, 0, 0, 1, 2],
    'PaperlessBilling': [0, 1, 0, 0, 0],
    'PaymentMethod': [1, 1, 1, 0, 0],
    'MonthlyCharges': [90.407734, 58.273891, 74.379767, 108.55, 64.35],
    'TotalCharges': [707.535237, 3264.466697, 1146.937795, 5610.7, 1558.65],
    'tenure_group': [0, 4, 1, 4, 2]
})

# Make predictions using the loaded model on your own data
predictions = loaded_model.predict(your_features)

# Print the predictions
print("Predictions:", predictions)
```

```
Predictions: [0 0 0 0 0]
```