Customer Churn Analytics

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
telco_base_data=pd.read_csv(r"C:\Users\JANHAVI\Desktop\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
```

In [7]:
```
for col in telco_base_data.columns:
    print("column:{} -Unique Values: {}".format(col,telco_base_data[col].unique()))
```

```
column:customerID -Unique Values: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '480
1-JZAZL' '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 tran
sfer (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 [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.dtypes
```

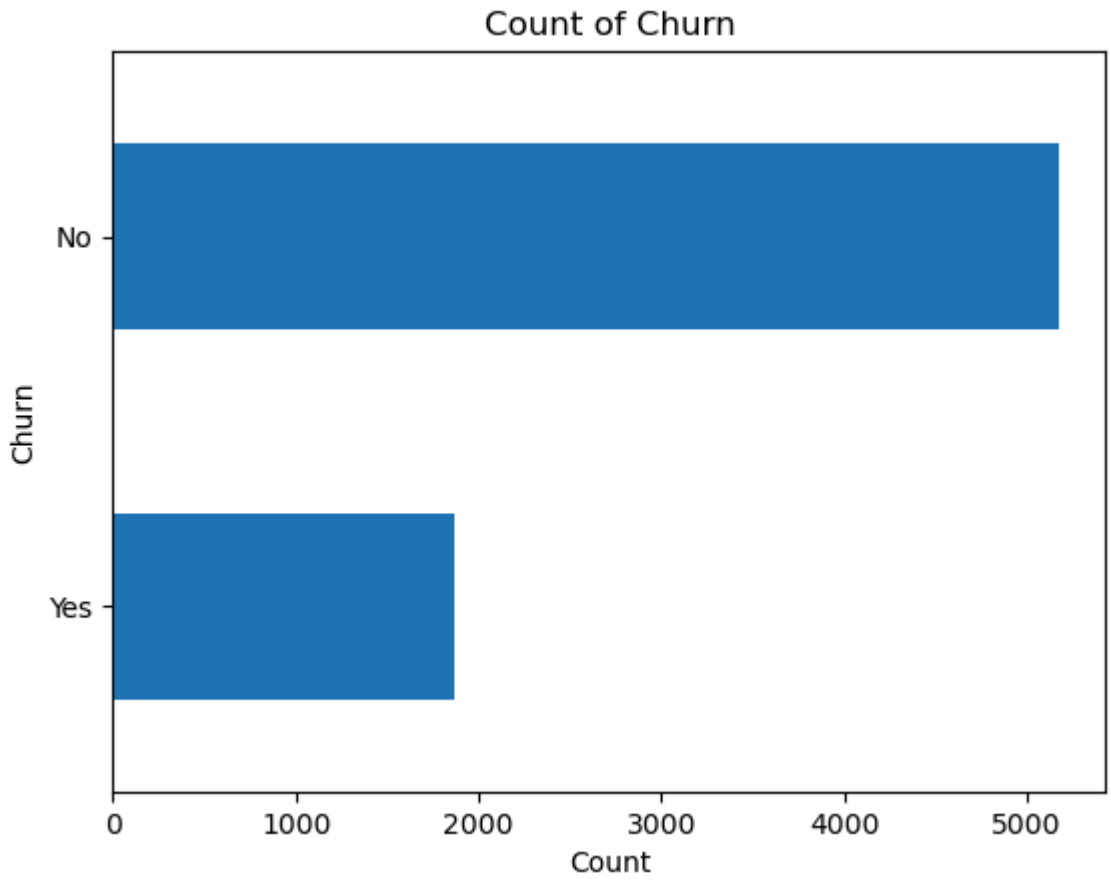
Out[10]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                object
dtype: object
```

In [11]:
```
telco_base_data.describe()
```

Out[11]:

	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 [12]:
```
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 [13]:
```
telco_base_data['Churn'].value_counts()/len(telco_base_data)
```

Out[13]:
```
Churn
No     0.73463
Yes    0.26537
Name: count, dtype: float64
```

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

Out[14]:
```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

In [15]:
```
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 [16]:
```
telco_data=telco_base_data.copy()
```

In [17]:
```
telco_data.isna().sum()
```

Out[17]:
```
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 [18]:
```
telco_data.loc[telco_data['TotalCharges'].isna()==True]
```

Out[18]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phor 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 phor 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 [19]:
```
telco_data.dtypes
```

Out[19]:
```
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 [20]:
```
telco_data.isna().sum()/len(telco_data)
```

Out[20]:
```
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

In [21]:
```
#Removing missing values
telco_data.dropna(how = 'any', inplace = True)
```

In [22]:
```
# Get the max tenure
print(telco_data['tenure'].max()) #72
```

```
72
```

In [23]:
```
# 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 [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)
```

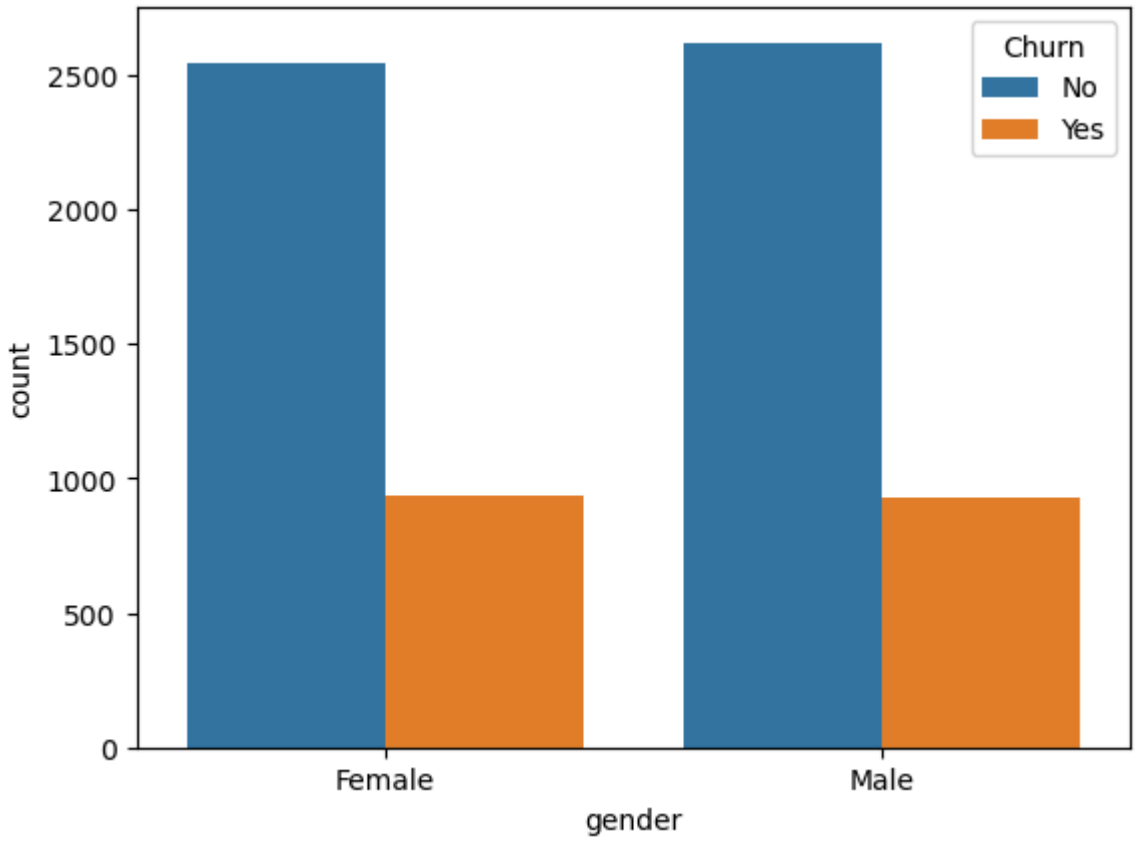
Out[25]:
```
tenure_group
1 - 12     0.292662
61 - 72    0.159414
13 - 24    0.148891
25 - 36    0.124573
49 - 60    0.116610
37 - 48    0.106371
Name: count, dtype: float64
```

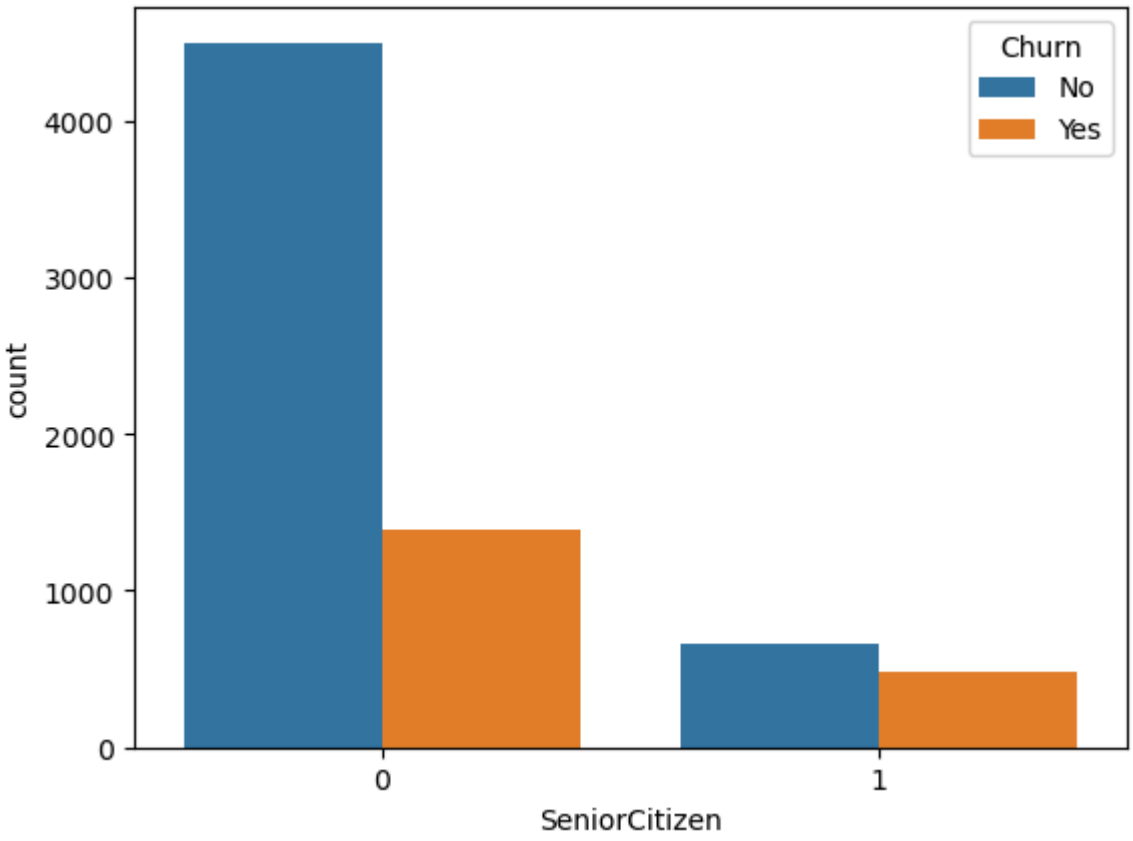
In [26]:
```
#drop column customerID and tenure
telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telco_data.head()
```

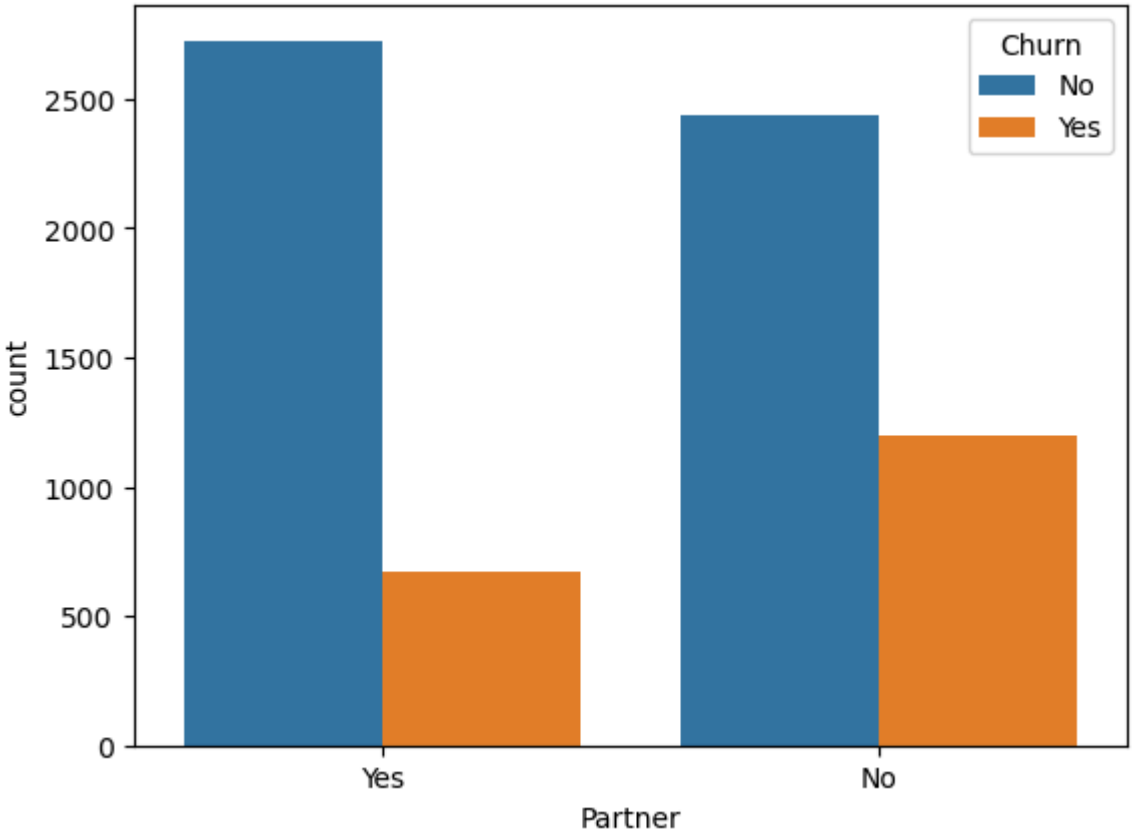
Out[26]:

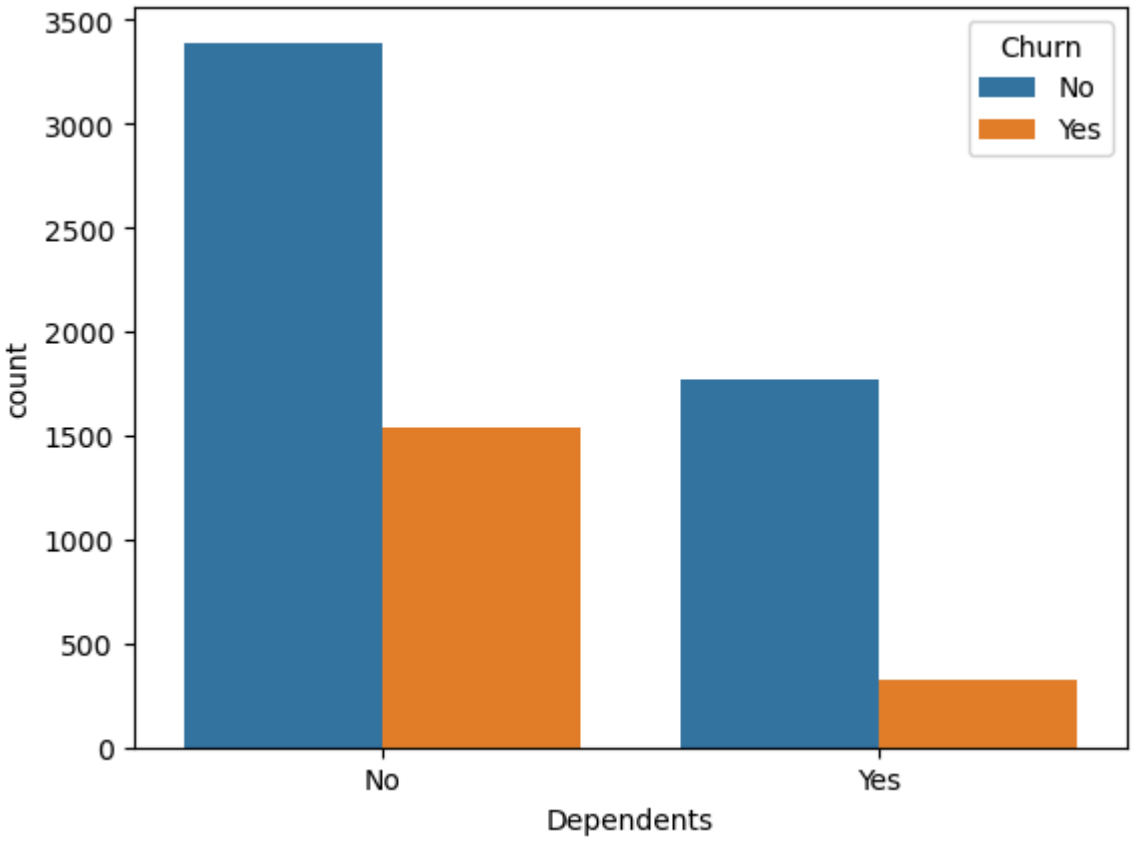
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	Onli
0	Female	0	Yes	No	No	No phone service	DSL	
1	Male	0	No	No	Yes	No	DSL	
2	Male	0	No	No	Yes	No	DSL	
3	Male	0	No	No	No	No phone service	DSL	
4	Female	0	No	No	Yes	No	Fiber optic	

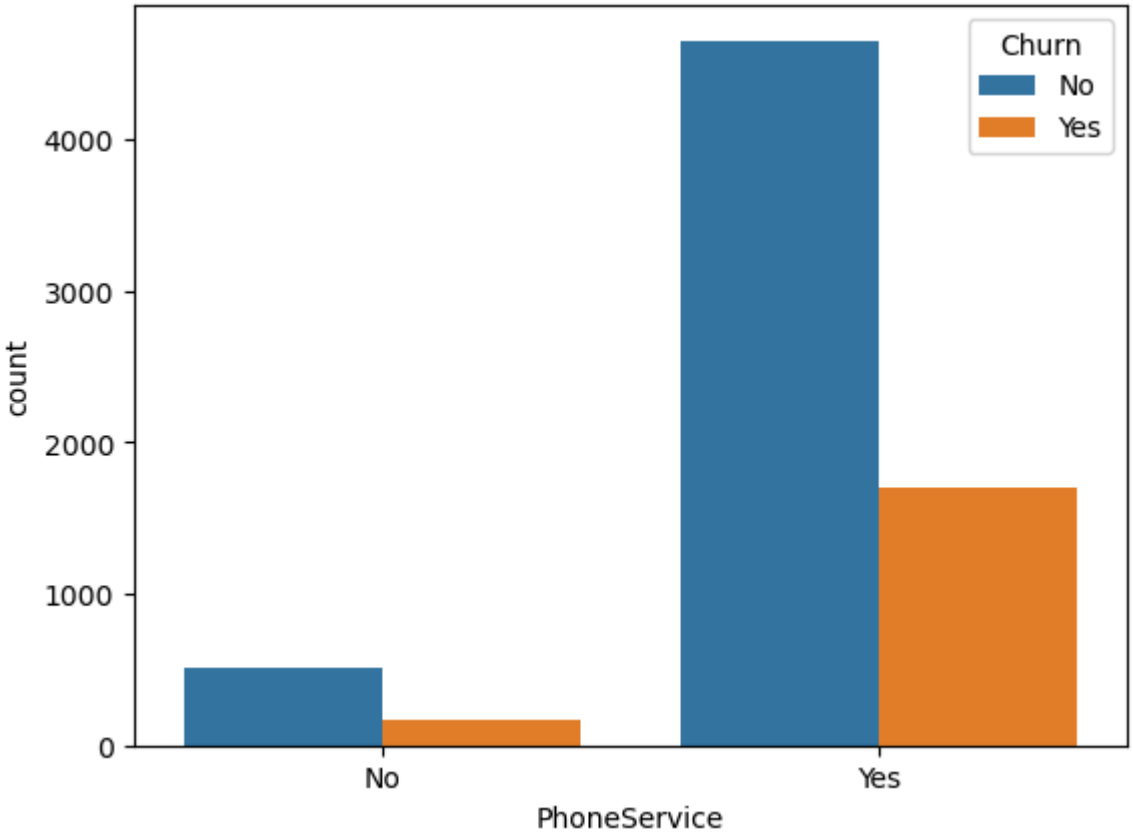
In [27]:
```
# plot distribution of individual predetion by churn
for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges', 'Mo
    plt.figure(i)
    sns.countplot(data=telco_data, x=predictor, hue='Churn')
```

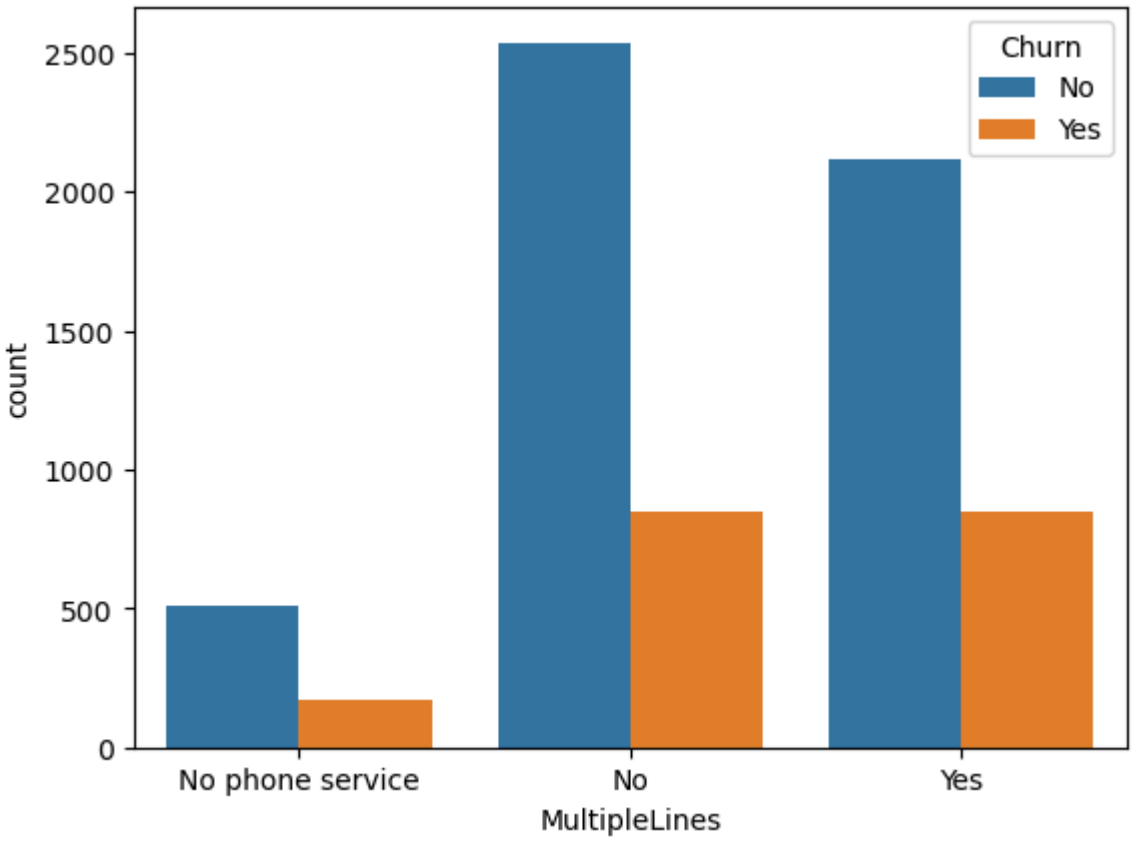


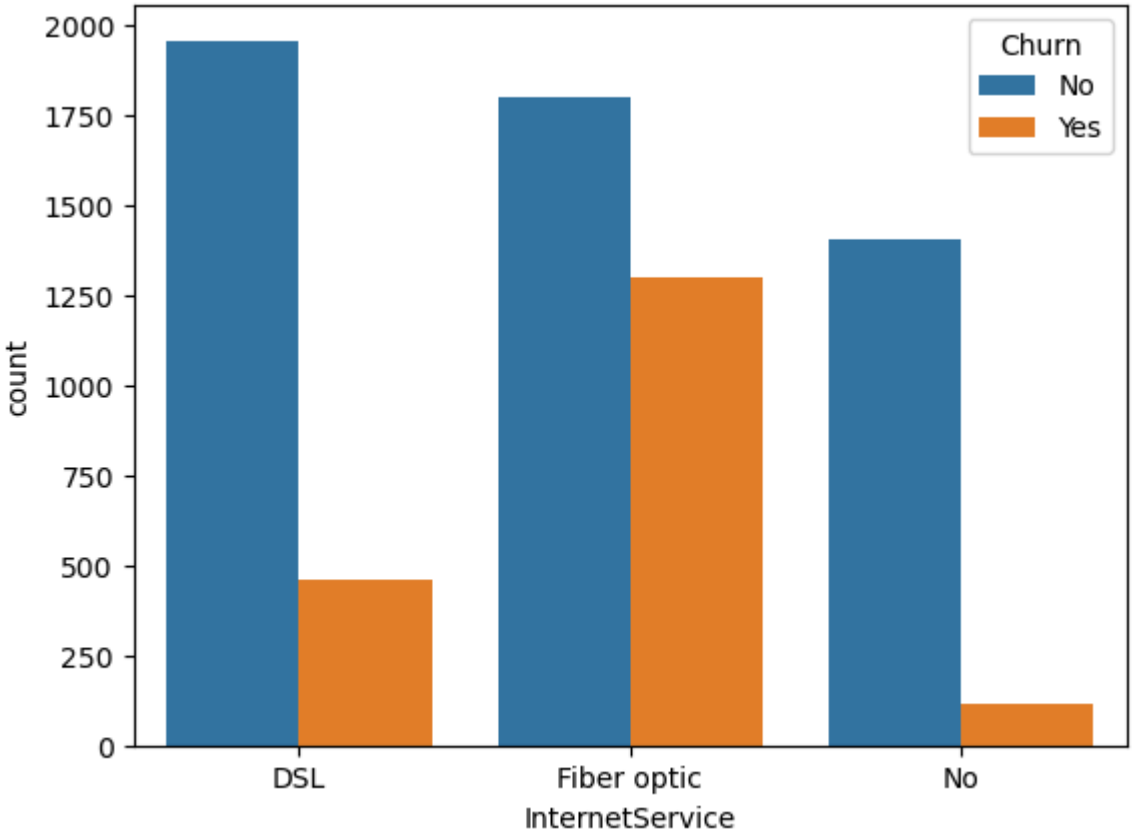

Churn
No
Yes
4000
3000
2000
1000
0
count
0
1
SeniorCitizen

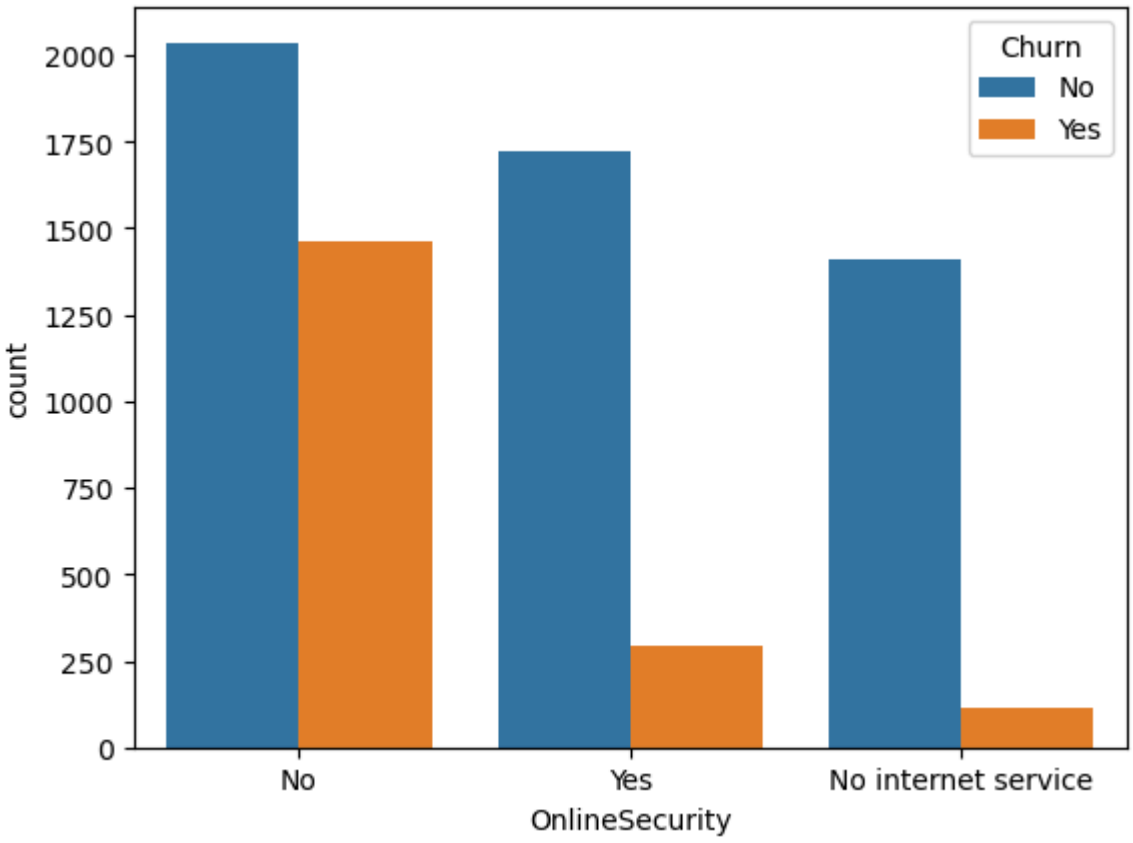

Churn
No
Yes
2500
2000
1500
1000
500
0
count
Yes
No
Partner

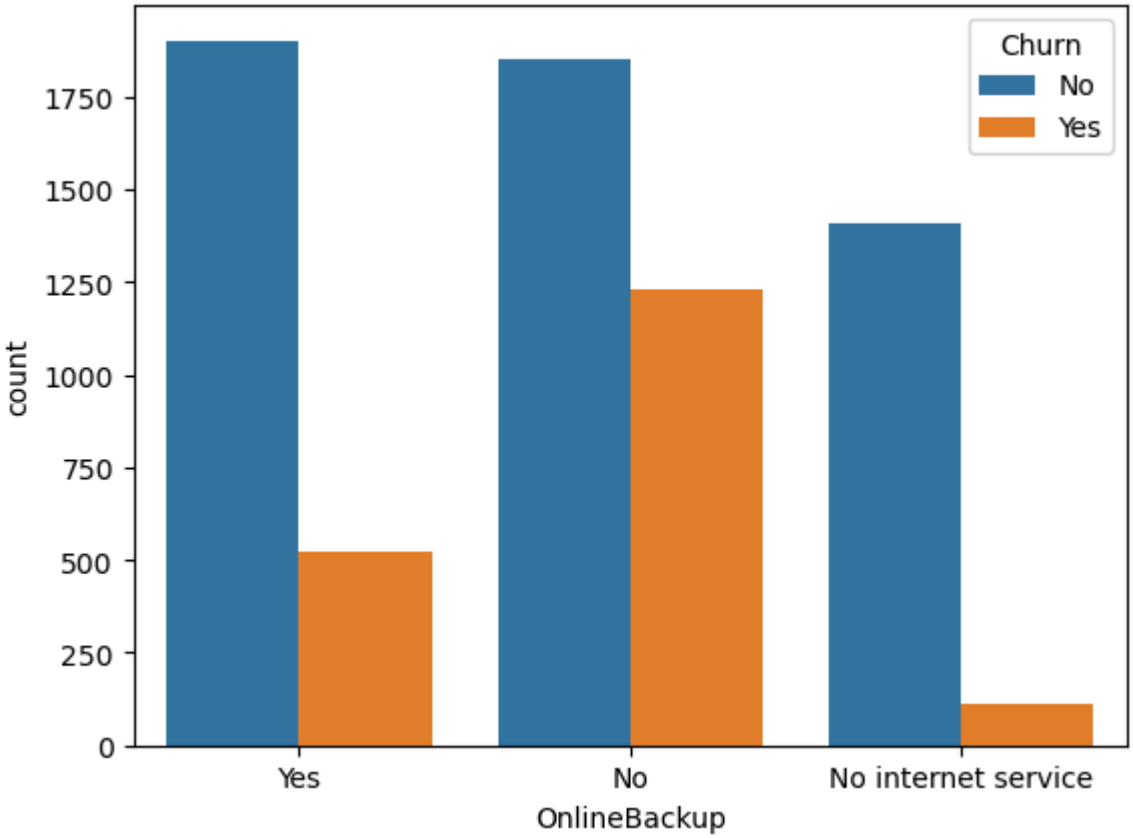

Churn
No
Yes
3500
3000
2500
2000
1500
1000
500
0
count
No
Yes
Dependents

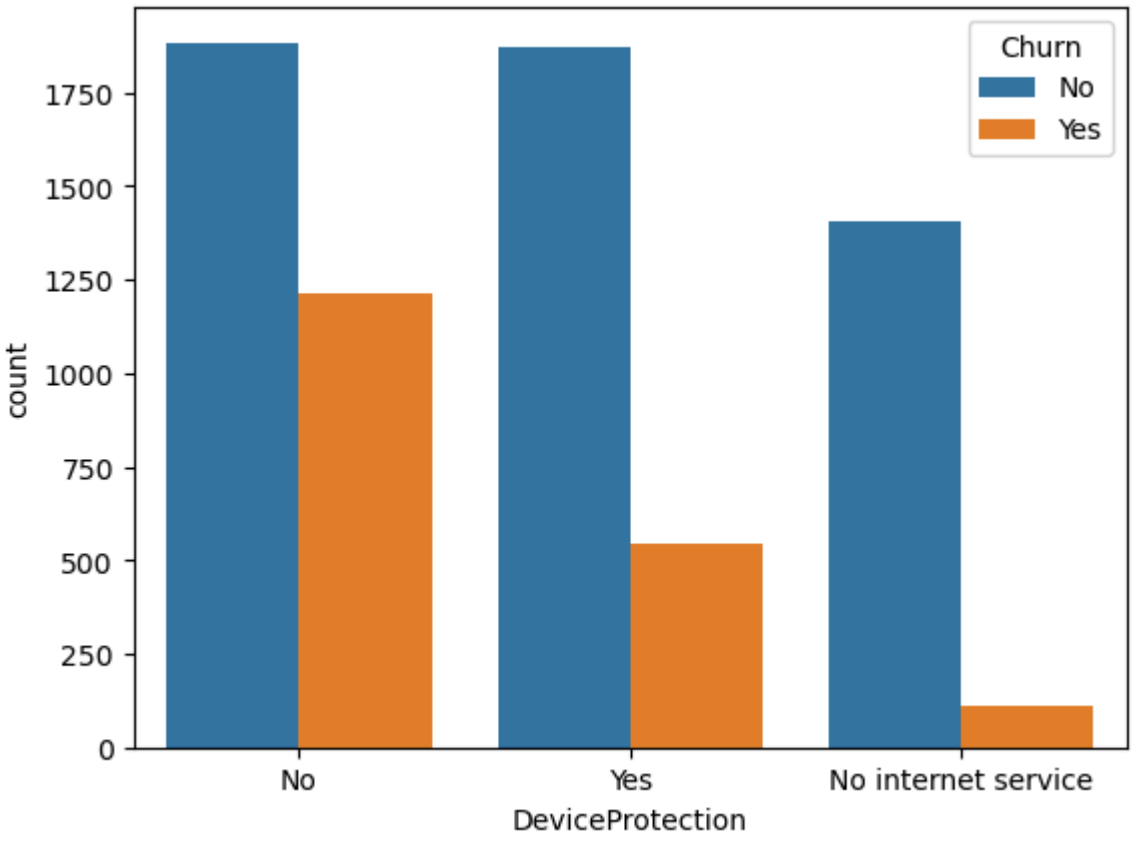

Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

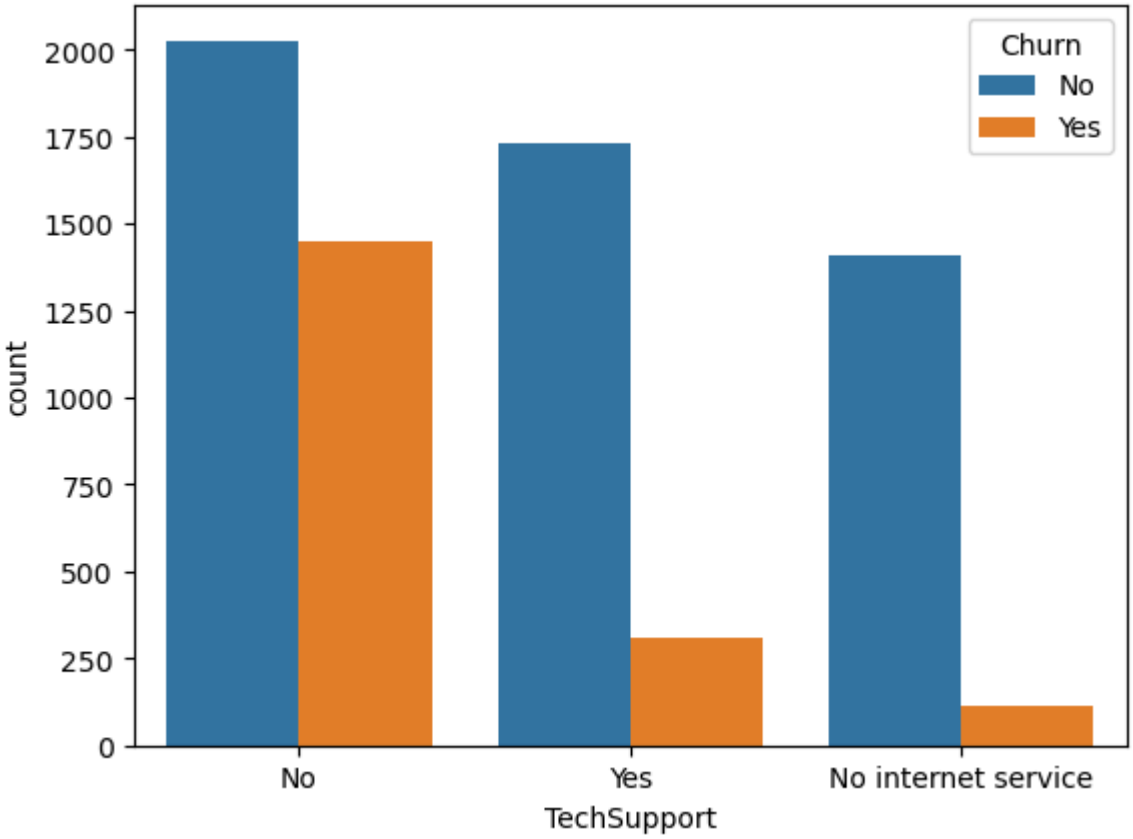

Churn
No
Yes
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

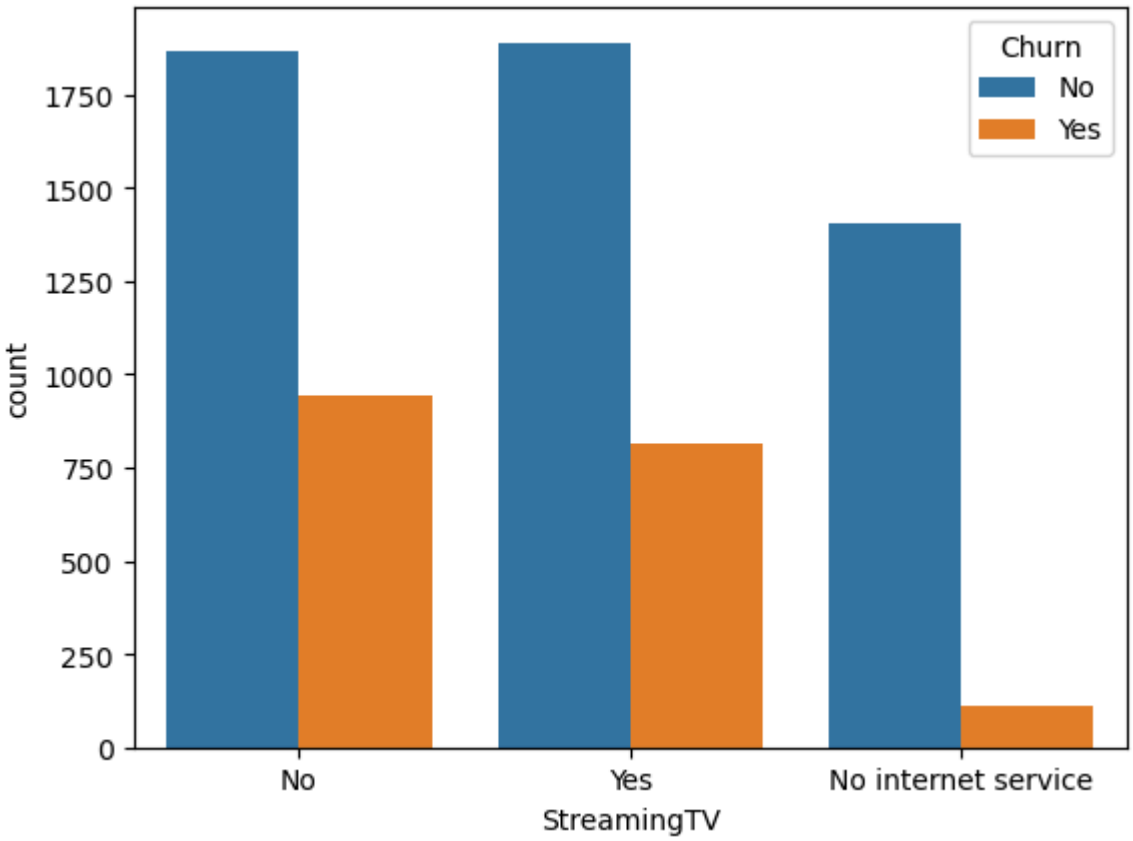

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
DSL
Fiber optic
No
InternetService

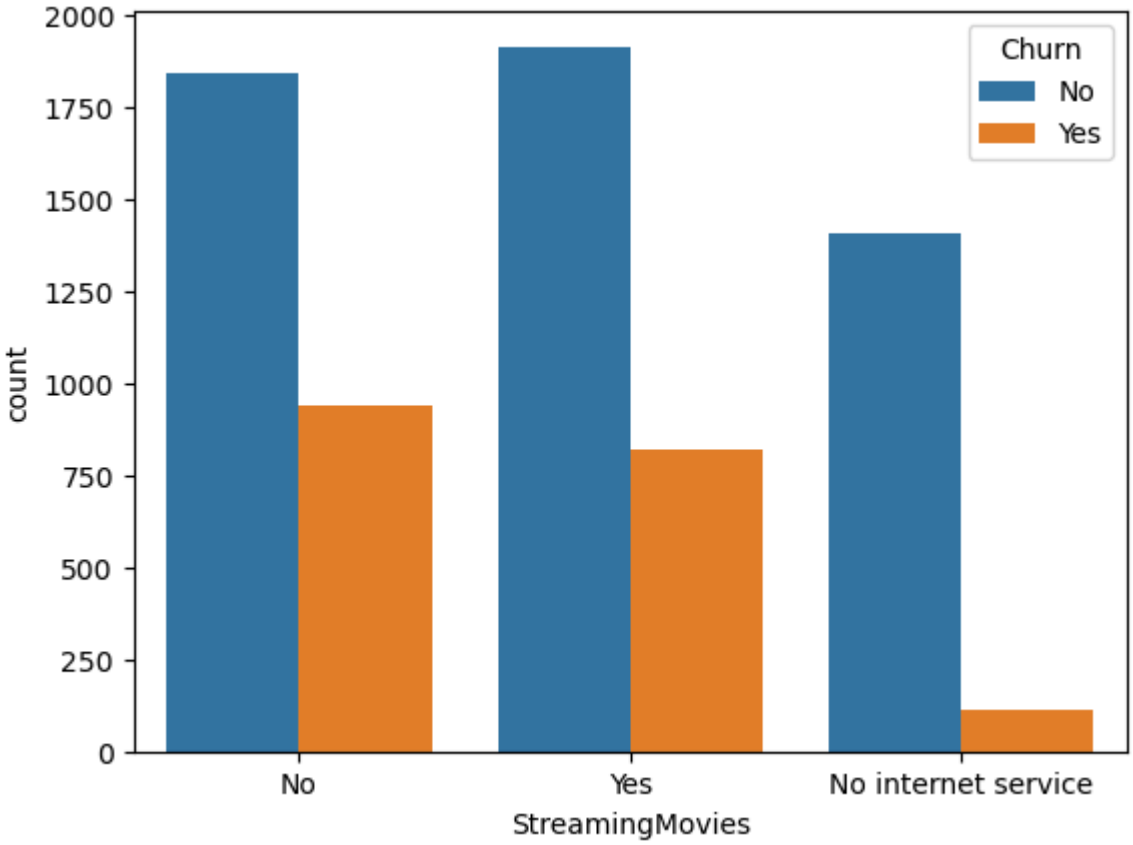

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

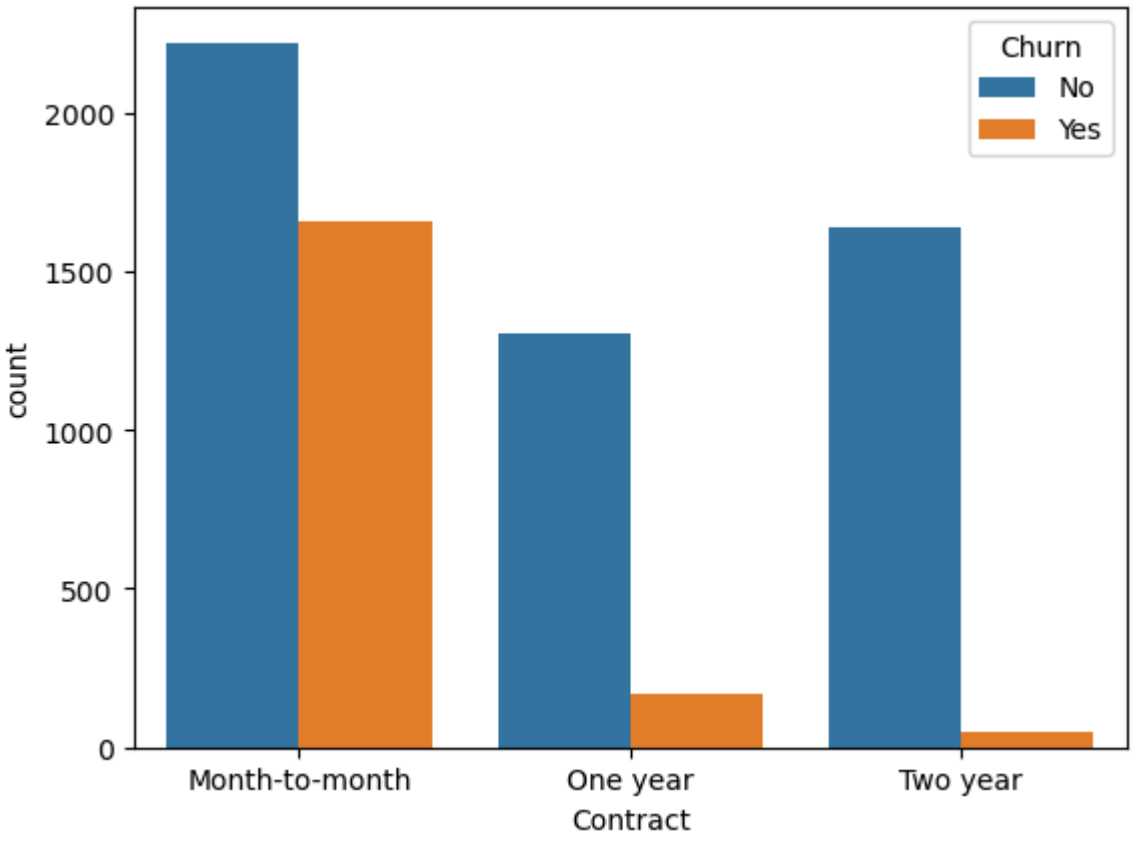

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

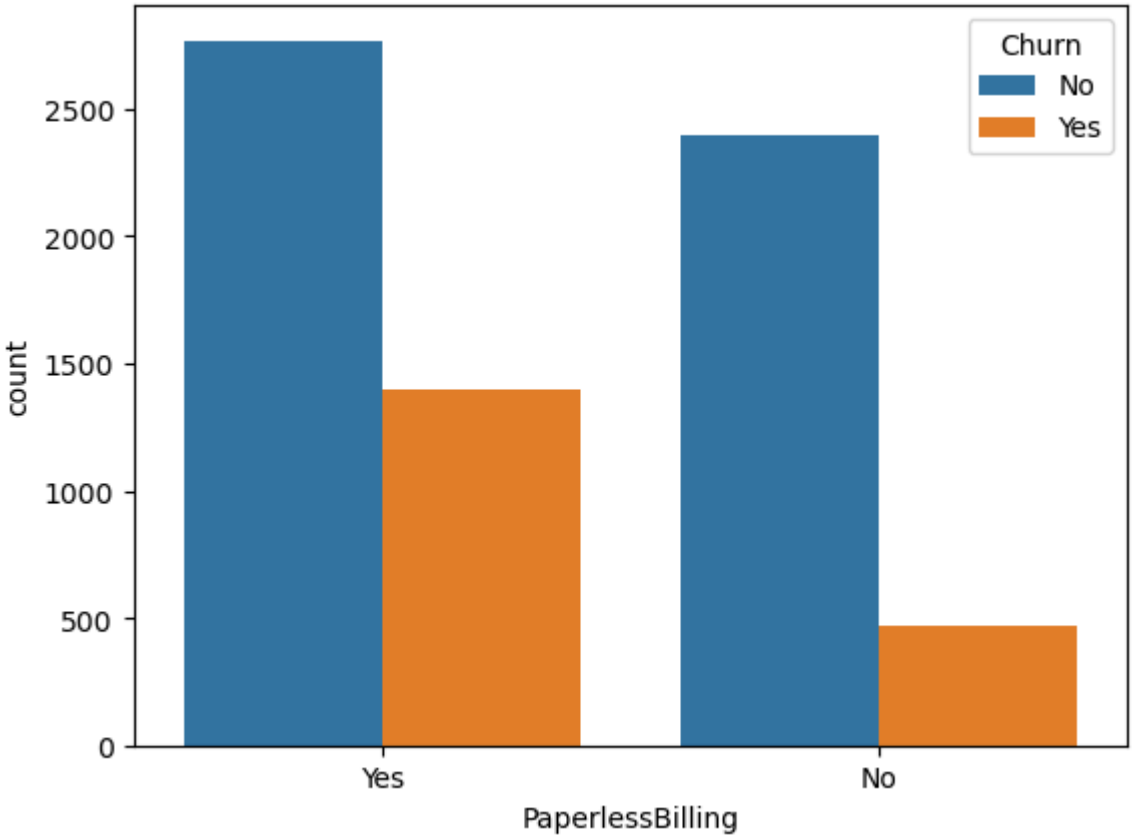

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
DeviceProtection

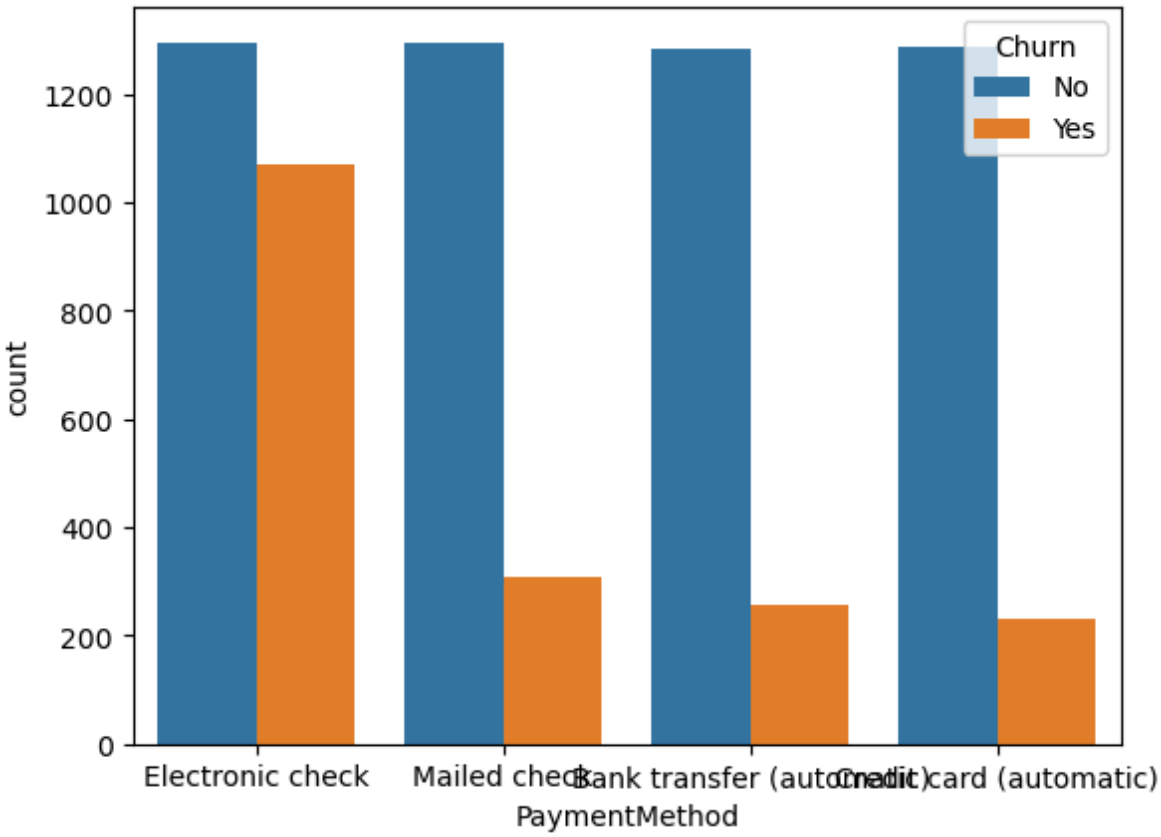

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
TechSupport

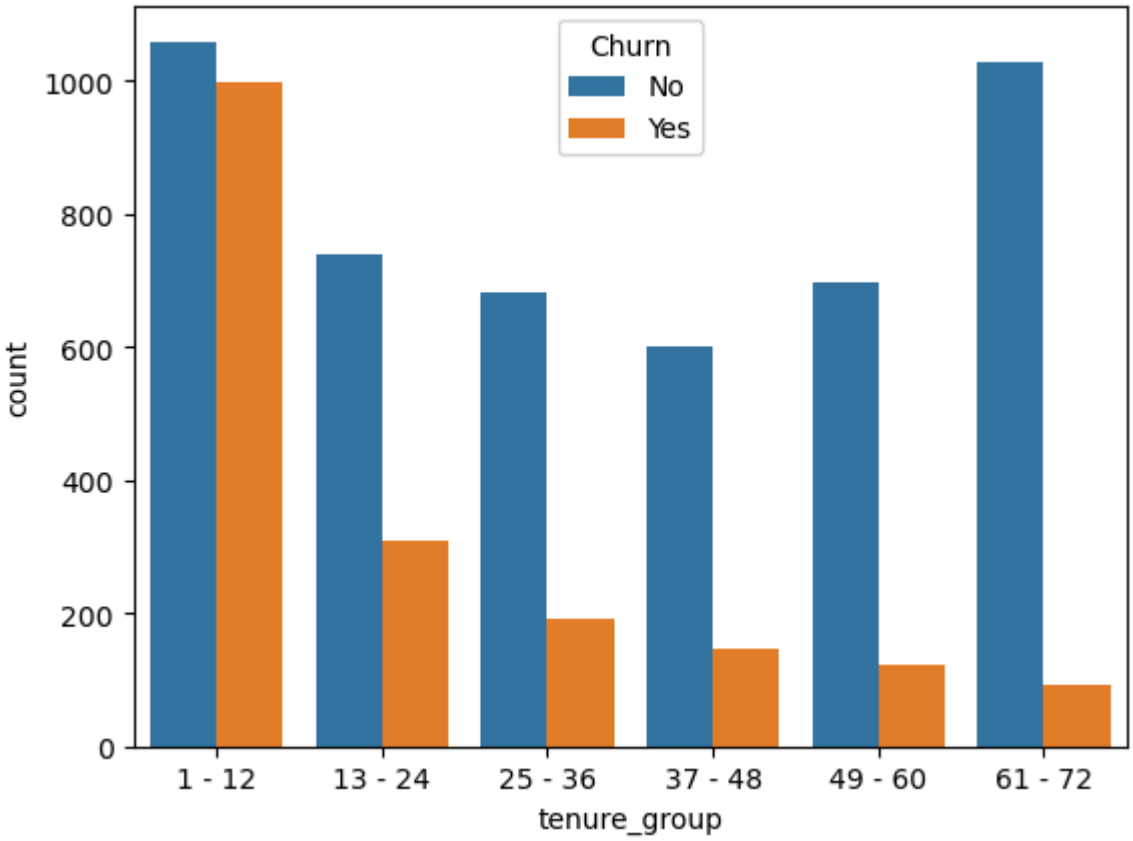

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
StreamingTV


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


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


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





In [28]:
```
# convert the target variable churn in a binary numeric variable i.e yes=1, no=0
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
1652	Male	0	Yes	No	No	No phone service	DSL	
2305	Male	0	Yes	Yes	Yes	Yes	Fiber optic	
442	Female	0	Yes	No	Yes	Yes	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
```

In [31]:
```
# Convert all the categorical variable into dummy variable
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)
```

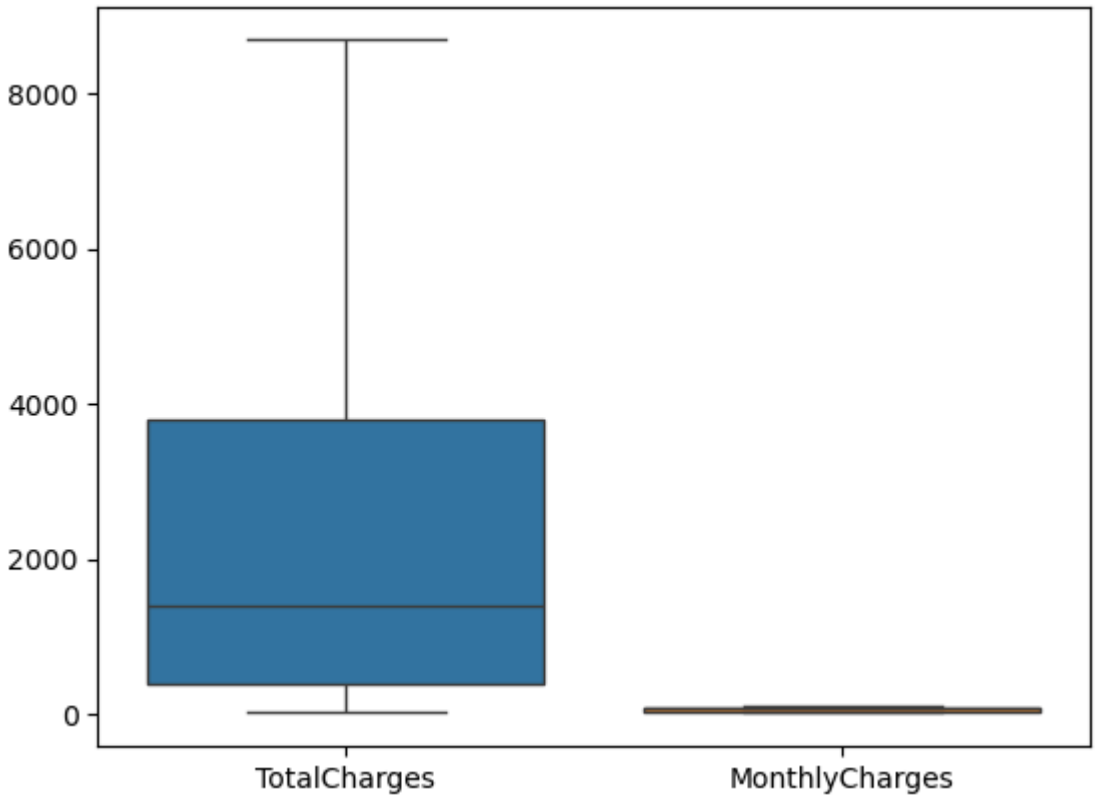
In [33]:
```
telco_data.sample(3)
```

Out[33]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	C
6189	1	0	1	1	1	2	0	
2599	0	0	0	0	1	0	0	
1554	1	0	0	0	1	0	2	

In [34]:
```
sns.boxplot(data=telco_data[['TotalCharges', 'MonthlyCharges']])
```

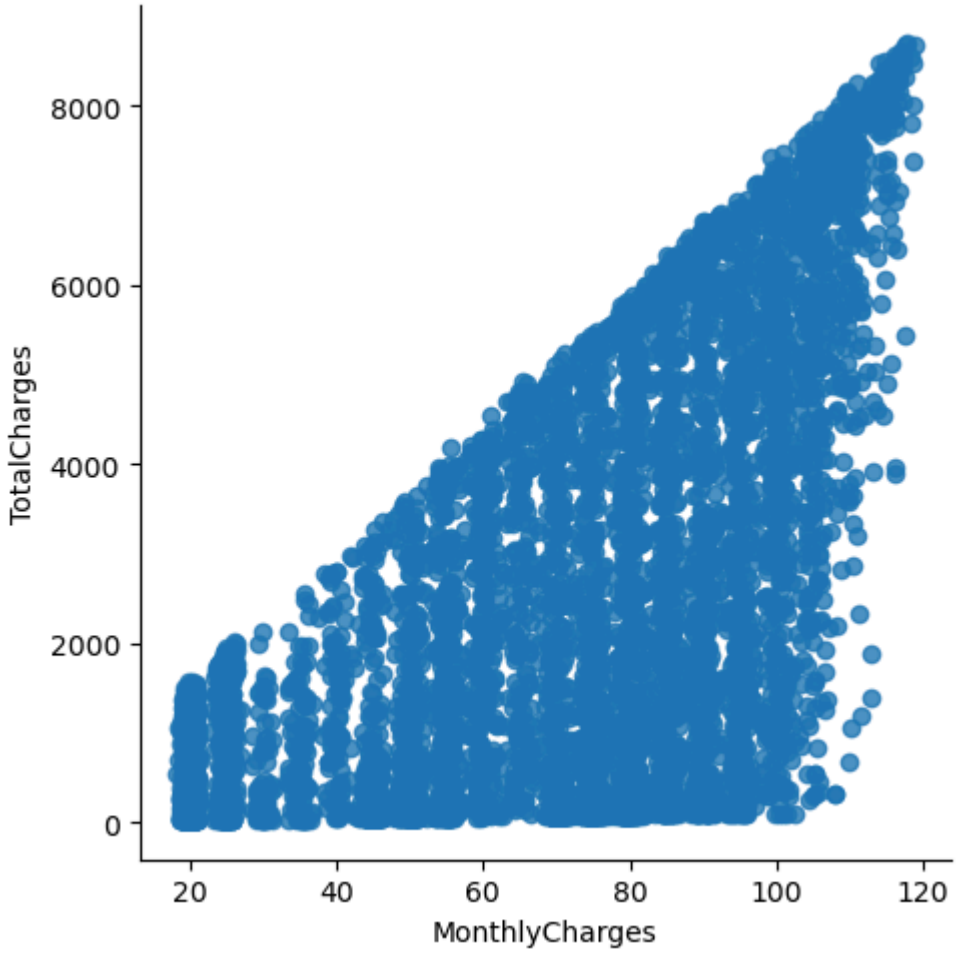
Out[34]:
```
<Axes: >
```



Realtionship Between Monthly Charges and Total Charges

In [35]:
```
sns.lmplot(data=telco_data, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```

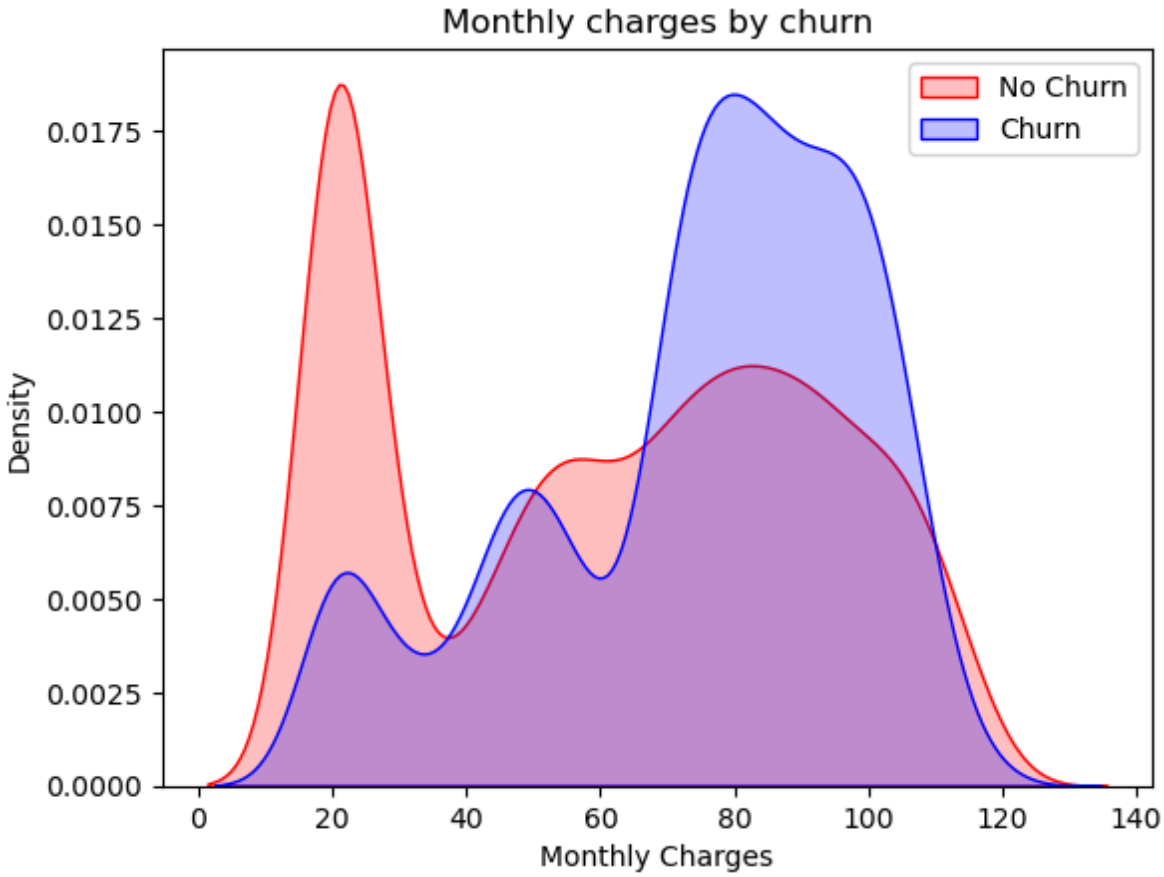
Out[35]:
```
<seaborn.axisgrid.FacetGrid at 0x1cfb8e51ed0>
```



Churn By Monthly Charges and Total Charges

In [36]:
```
# 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')
```

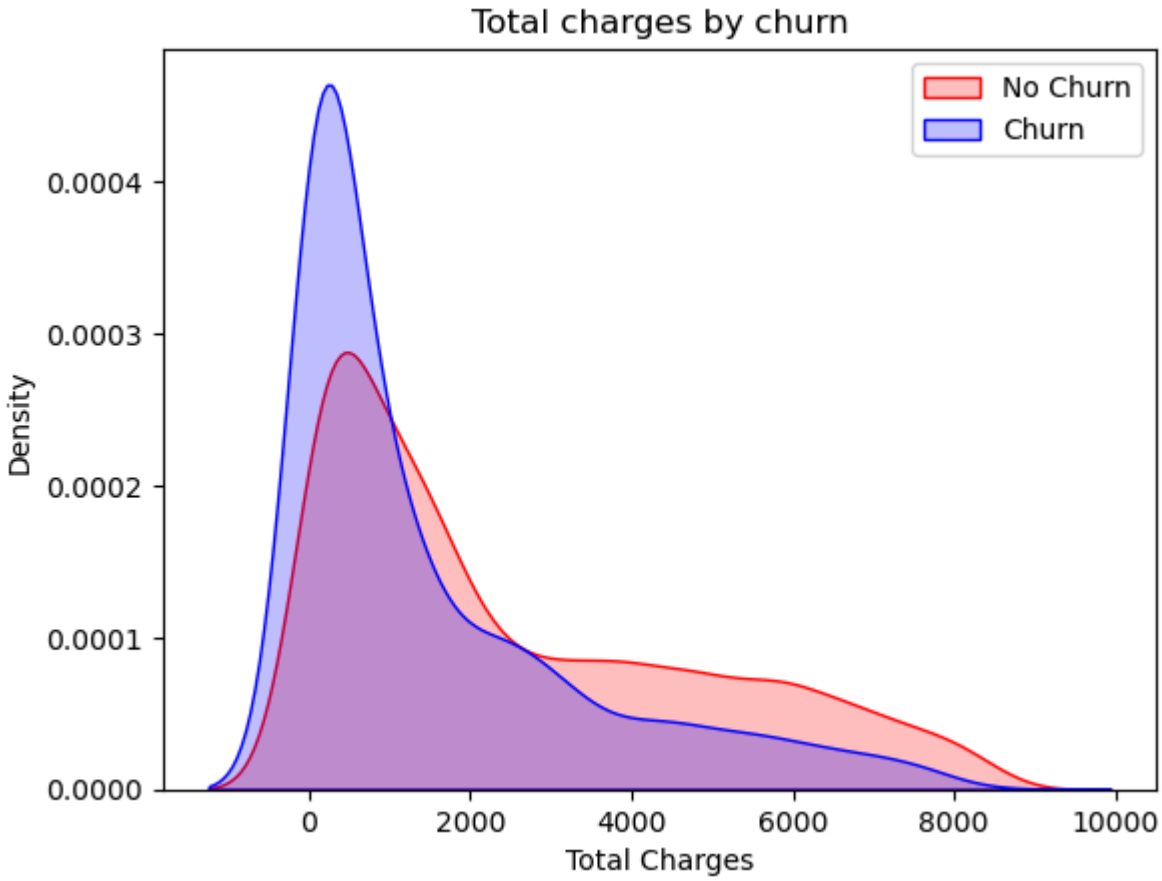
Out[36]:
```
Text(0.5, 1.0, 'Monthly charges by churn')
```



Insight: Churn is high when monthly charges are high

In [37]:
```
Tot = sns.kdeplot(telco_data.TotalCharges[(telco_data["Churn"] == 0) ],
                  color="Red", shade = True)
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')
```

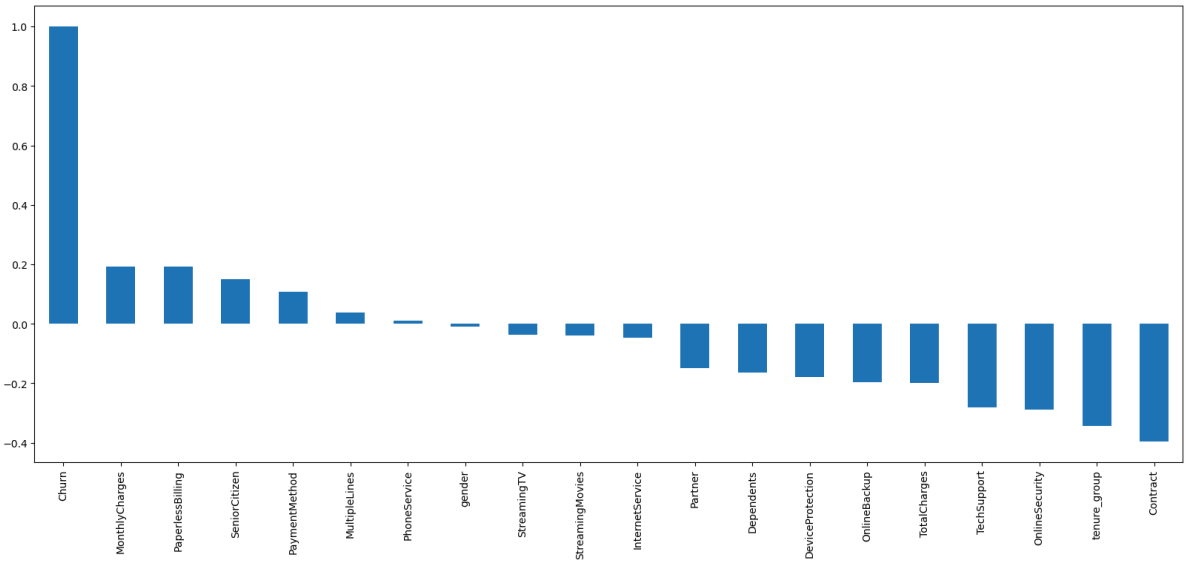
Out[37]:
```
Text(0.5, 1.0, 'Total charges by churn')
```



Build a corelation of all predictors with Churn

In [38]:
```
plt.figure(figsize=(20,8))
telco_data.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

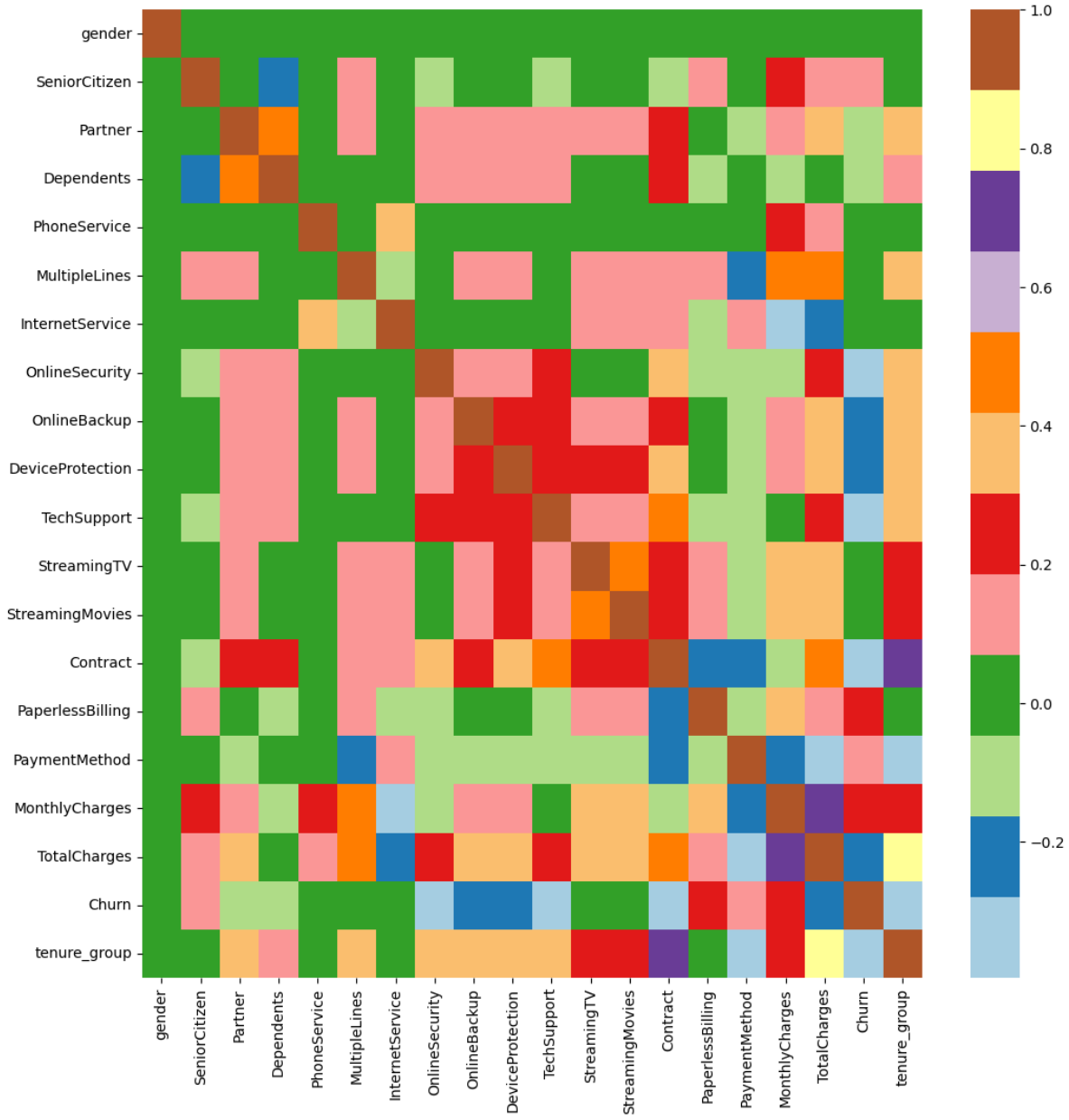
Out[38]:
```
<Axes: >
```



Derived Insight

In [39]:
```
plt.figure(figsize=(12,12))
sns.heatmap(telco_data.corr(), cmap="Paired")
```

Out[39]: <Axes: >



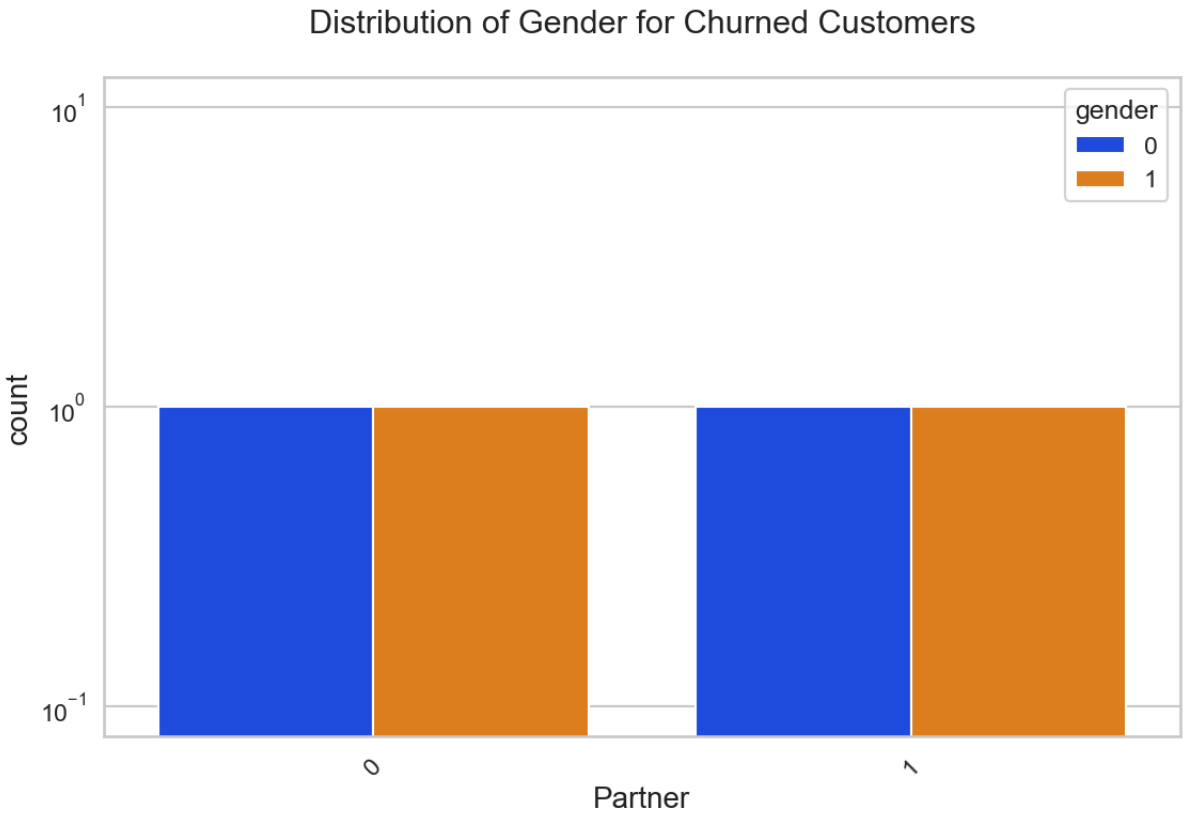
Bivariate Analysis

In [40]:
```
new_df1_target0=telco_data.loc[telco_data["Churn"]==0]
new_df1_target1=telco_data.loc[telco_data["Churn"]==1]
```

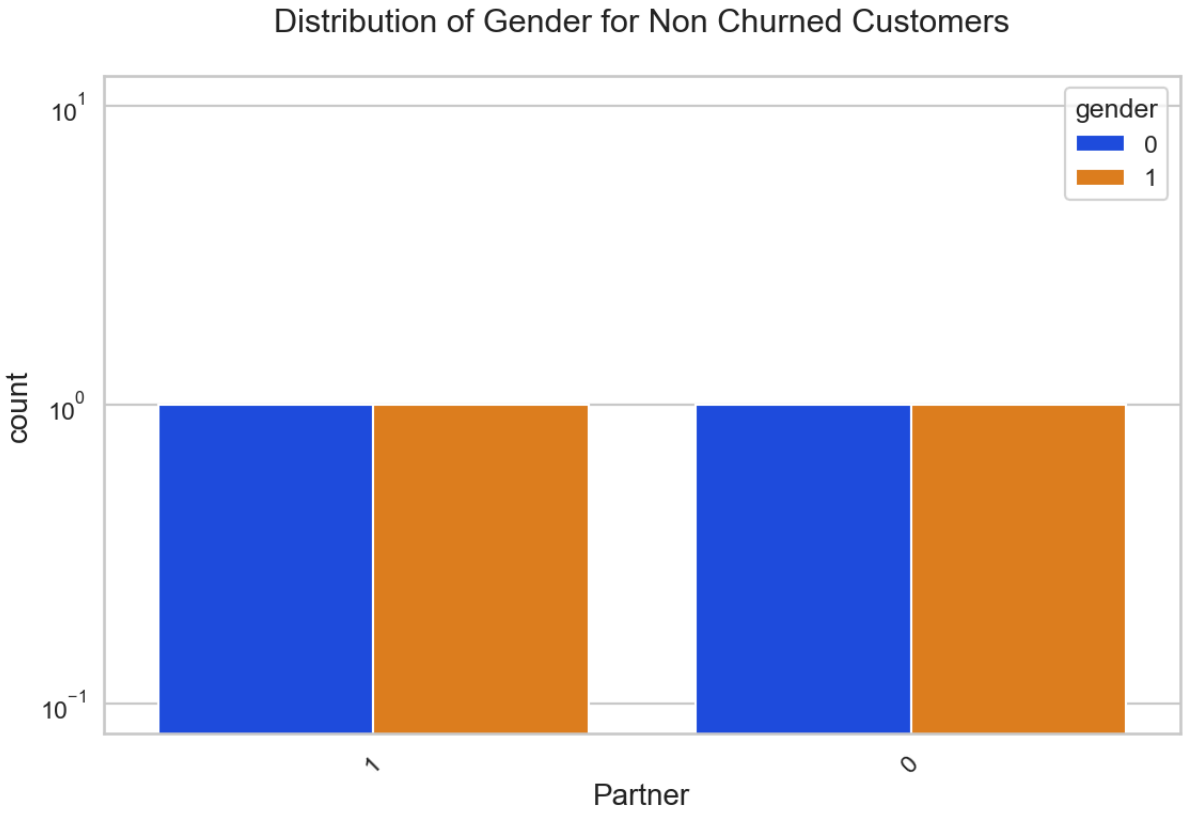
In [41]:
```
def uniplot(df,col,title,hue =None):

    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()
```

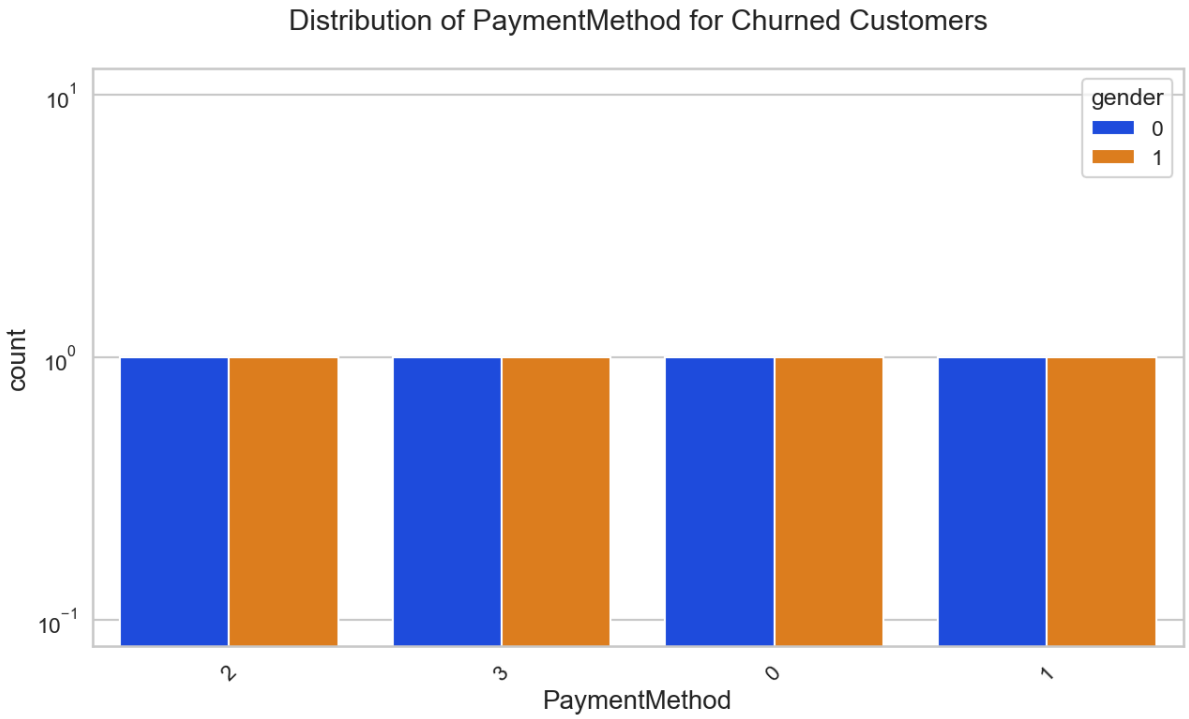
In [42]:
```
uniplot(new_df1_target1,col='Partner',title='Distribution of Gender for Churned Cus
```



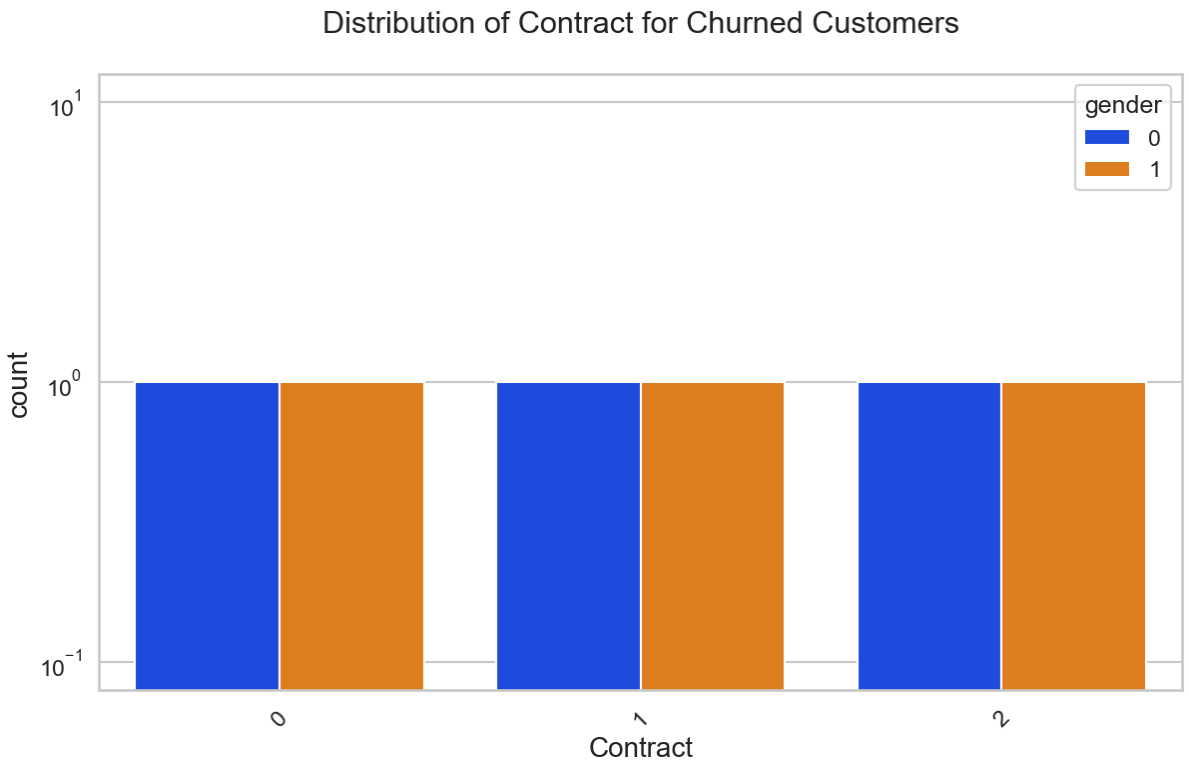
In [43]:
```
uniplot(new_df1_target0,col='Partner',title='Distribution of Gender for Non Churned
```



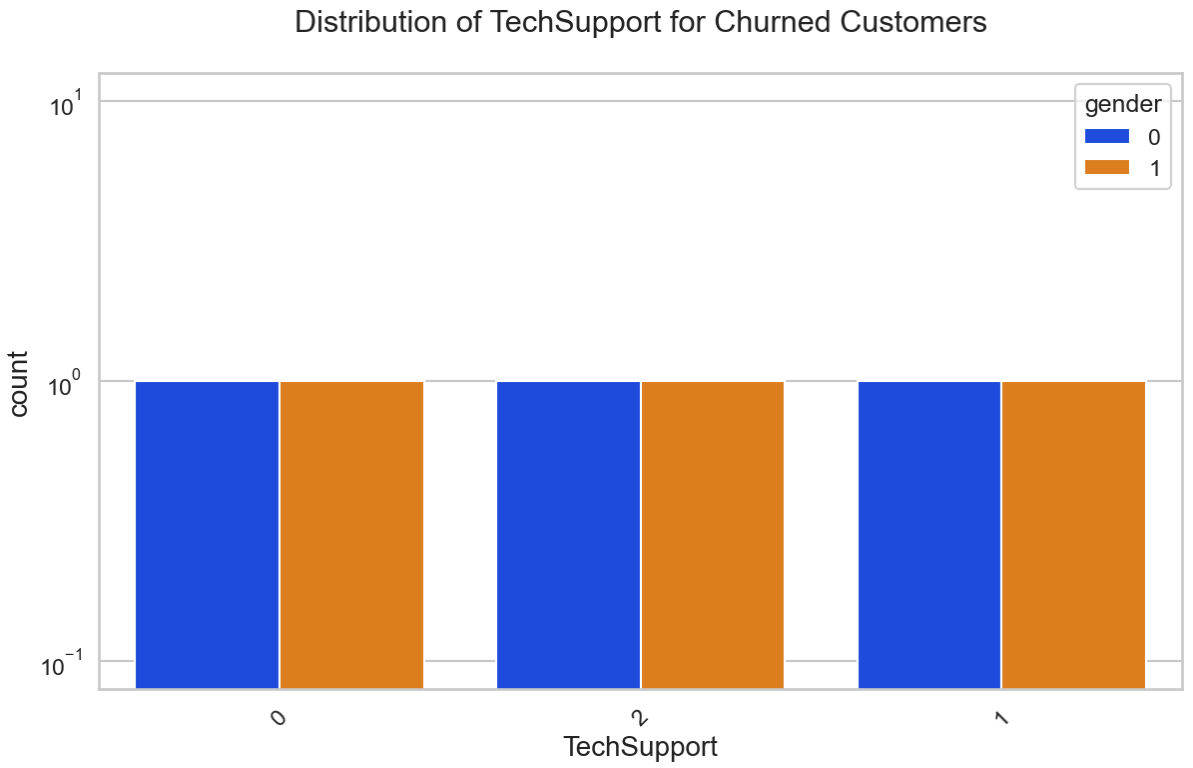
In [44]:
```
uniplot(new_df1_target1,col='PaymentMethod',title='Distribution of PaymentMethod fo
```



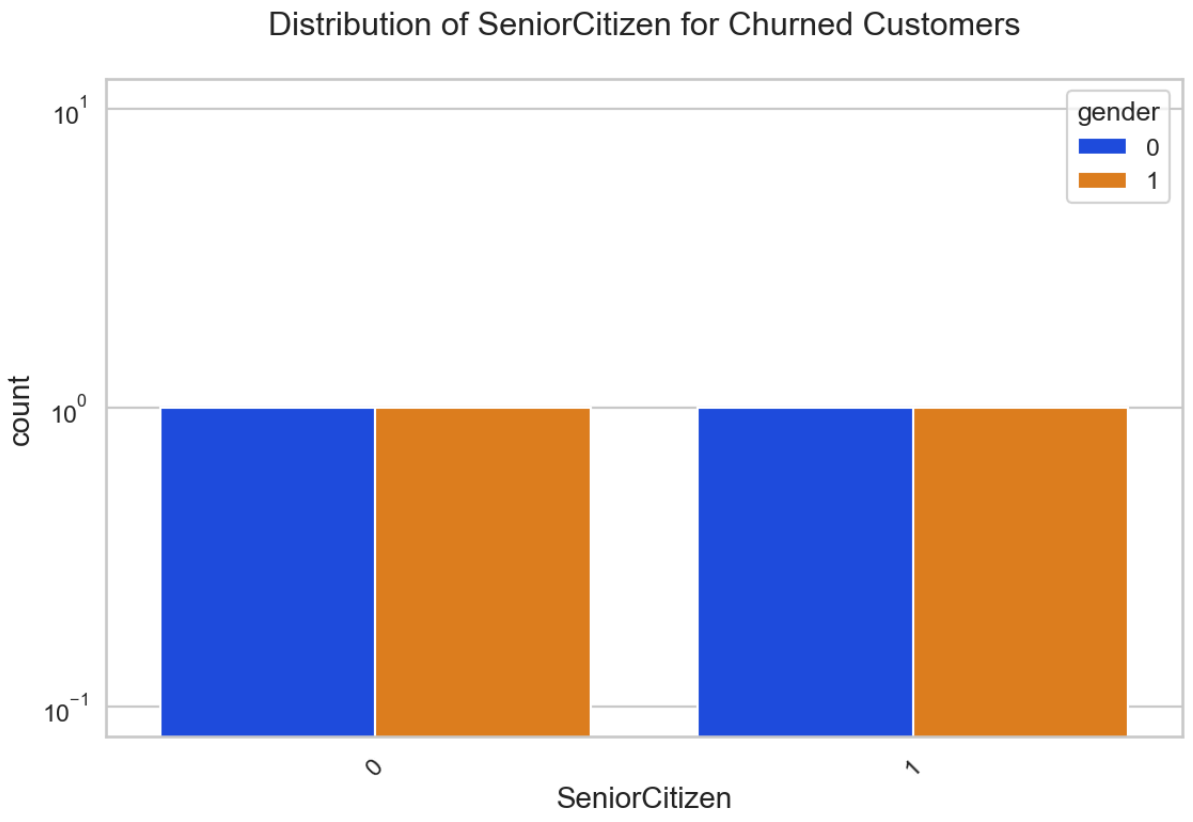
In [45]:
```
uniplot(new_df1_target1,col='Contract',title='Distribution of Contract for Churned
```



In [46]:
```
uniplot(new_df1_target1,col='TechSupport',title='Distribution of TechSupport for Ch
```



In [47]:
```
uniplot(new_df1_target1,col='SeniorCitizen',title='Distribution of SeniorCitizen fo
```



In [48]:
```
X=telco_data.drop('Churn',axis=1)
y=telco_data['Churn']
```

In [49]:
```
X
```

Out[49]:

	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 [50]:
```
telco_data['Churn'].value_counts()/len(telco_data) #data is highly imbalancing
```

Out[50]:
```
Churn
0    0.734215
1    0.265785
Name: count, dtype: float64
```

Train Test Split

In [51]:
```
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_state=42)
```

In [52]:
```
print('Traing data shape')
print(X_train.shape)
print(y_train.shape)
print('Testing Data shape')
print(X_test.shape)
print(y_test.shape)
```

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

In [53]:
```
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 [54]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [55]:
```
model_dtc=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6,
```

In [56]:
```
model_dtc.fit(X_train,y_train)
```

Out[56]:

▾ DecisionTreeClassifier

▶ Parameters

In [57]:
```
model_dtc.score(X_test,y_test)
```

Out[57]:
```
0.7619047619047619
```

In [58]:
```
y_pred=model_dtc.predict(X_test)
y_pred[:10]
```

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

In [59]:
```
print(y_test[:10])
```

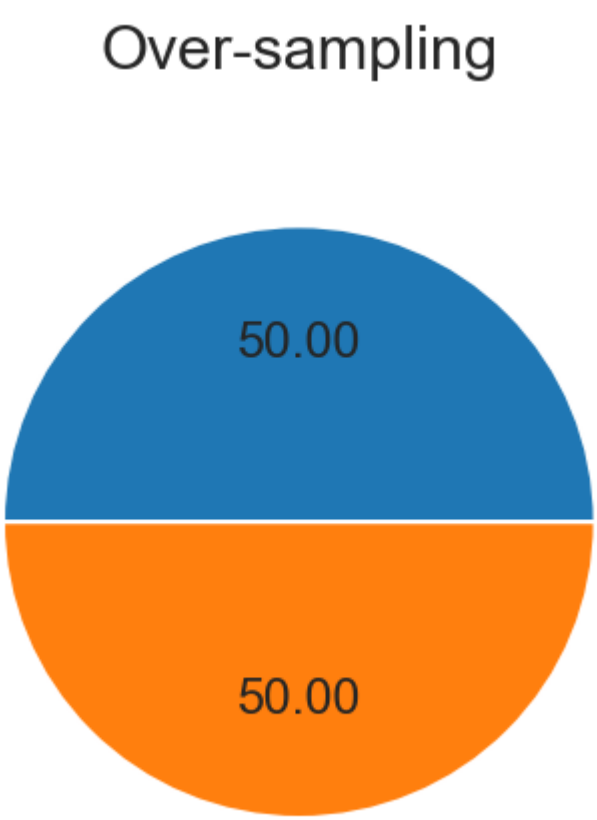
```
2481    0
6784    0
6125    1
3052    0
4099    0
3223    0
3774    0
3469    0
3420    0
1196    0
Name: Churn, dtype: int64
```

In [60]:
```
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 [61]:
```
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 [62]:
```
Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_ovs, y_ovs,test_size=0.2,randc
```

In [63]:
```
from sklearn.linear_model  import LogisticRegression
model_lr=LogisticRegression(max_iter=1000)
```

In [64]:
```
model_lr.fit(Xr_train,yr_train)
```

Out[64]:
```
▾ LogisticRegression
▶ Parameters
```

In [65]:
```
y_pred=model_lr.predict(Xr_test)
y_pred[:10]
```

Out[65]:
```
array([1, 0, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

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

Out[66]:
```
0.8025169409486931
```

In [67]:
```
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      0.82      0.80       983
           1       0.83      0.79      0.81      1083

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

In [68]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(yr_test,y_pred)
```

Out[68]:
```
array([[806, 231],
       [177, 852]], dtype=int64)
```

Decision Tree Classifier

In [69]:
```
from sklearn.tree import DecisionTreeClassifier
model_dtc=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6,
```

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

Out[70]:
```
▾ DecisionTreeClassifier
▶ Parameters
```

In [71]:
```
y_pred=model_dtc.predict(Xr_test)
y_pred[:10]
```

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

In [72]:
```
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]:
```
model_dtc.score(Xr_test,yr_test)
```

Out[73]:
```
0.8049370764762827
```

In [74]:
```
print(classification_report(yr_test, y_pred, labels=[0,1]))
```

```
                precision    recall  f1-score   support

           0       0.82      0.79      0.80      1037
           1       0.79      0.82      0.81      1029

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

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

Out[75]:
```
array([[817, 220],
       [183, 846]], dtype=int64)
```

Random Forest Classifier

In [76]:
```
from sklearn.ensemble import RandomForestClassifier
model_rfc=RandomForestClassifier(n_estimators=100, random_state = 100,max_depth=6,
```

In [77]:
```
model_rfc.fit(Xr_train,yr_train)
```

Out[77]:
```
▾ RandomForestClassifier
▶ Parameters
```

In [78]:
```
y_pred=model_rfc.predict(Xr_test)
y_pred[:10]
```

Out[78]:
```
array([1, 0, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [79]:
```
yr_test[:10]
```

Out[79]:
```
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 [80]:
```
model_rfc.score(Xr_test,yr_test)
```

Out[80]:
```
0.818973862536302
```

In [81]:
```
report_rfc=classification_report(y_pred,yr_test)
print(report_rfc)
```

```
                precision    recall  f1-score   support

           0       0.78      0.84      0.81       965
           1       0.85      0.80      0.82      1101

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

In [82]:
```
confusion_matrix(yr_test,y_pred)
```

Out[82]:
```
array([[814, 223],
       [151, 878]], dtype=int64)
```

AdaBoost

In [83]:
```
from sklearn.ensemble import  AdaBoostClassifier
```

In [84]:
```
model_abc=AdaBoostClassifier(n_estimators=100)
```

In [85]:
```
model_abc.fit(Xr_train,yr_train)
```

Out[85]:
```
▾ AdaBoostClassifier
▶ Parameters
```

In [86]:
```
y_pred=model_abc.predict(Xr_test)
```

In [87]:
```
print(classification_report(y_pred,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       946
           1       0.86      0.79      0.82      1120

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

In [88]:
```
confusion_matrix(yr_test,y_pred)
```

Out[88]:
```
array([[802, 235],
       [144, 885]], dtype=int64)
```

Gradient Boosting Classifier

In [89]:
```
from sklearn.ensemble import GradientBoostingClassifier
model_gbc=GradientBoostingClassifier()
model_gbc
```

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

In [90]:
```
model_gbc.fit(Xr_train,yr_train)
```

Out[90]:
GradientBoostingClassifier

▶ Parameters

In [91]:
```
y_pred_gbc=model_gbc.predict(Xr_test)
y_pred_gbc[:10]
```

Out[91]:
```
array([1, 0, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

In [92]:
```
yr_test[:10]
```

Out[92]:
```
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 [93]:
```
print(classification_report(y_pred_gbc,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       961
           1       0.87      0.81      0.84      1105

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

In [94]:
```
confusion_matrix(yr_test,y_pred)
```

Out[94]:
```
array([[802, 235],
       [144, 885]], dtype=int64)
```

Xgboost

In [95]:
```
from xgboost import XGBClassifier

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

model_xgb
```

Out[95]:
XGBClassifier

▶ Parameters

In [96]:
```
model_xgb.fit(Xr_train,yr_train)
```

Out[96]:
```
XGBClassifier
▶ Parameters
```

In [97]:
```
y_pred=model_xgb.predict(Xr_test)
y_pred[:10]
```

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

In [98]:
```
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]:
```
print(classification_report(y_pred,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.84      1021
           1       0.84      0.83      0.84      1045

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

In [100…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(yr_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[860 177]
 [161 868]]
```

Hyperparameter

In [101…
```
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
```

```
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)
```

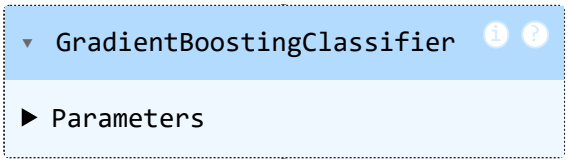
```
RandomizedSearchCV took 62.31 seconds to complete.
Best Parameters: {'n_estimators': 100, 'min_samples_split': 5, 'max_depth': 7, 'le
arning_rate': 0.1}
```

Final Model

In [102…

```
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[102]:



In [103…

```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(final_gb_classifier, Xr_train, yr_train, cv=10, scoring
print("Cross-validation scores:", cv_scores)
print("Mean CV score:", cv_scores.mean())
```

```
Cross-validation scores: [0.8377724  0.86440678 0.83656174 0.8535109  0.84987893
0.83898305
 0.83050847 0.8401937  0.83050847 0.87046005]
Mean CV score: 0.845278450363196
```

In [104…

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

Out[104]:

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

In [105…

```
yr_test[:10]
```

Out[105]:
```
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 [106…
```
print(classification_report(y_pred,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83      1019
           1       0.84      0.83      0.83      1047

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

In [107…
```
confusion_matrix(y_pred,yr_test)
```

Out[107]:
```
array([[856, 163],
       [181, 866]], dtype=int64)
```

Pickle File

In [108…
```
X = pd.get_dummies(X, drop_first=True)
```

In [109…
```
import os
import pickle
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier

# Load dataset
csv_path = r"C:\Users\JANHAVI\Desktop\Telco-Customer-Churn.csv"
df = pd.read_csv(csv_path)

# Explore data types and head
print(df.info())
print(df.head())

# Drop irrelevant columns like 'customerID'
df = df.drop(['customerID'], axis=1)

# Convert target 'Churn' from Yes/No to 1/0
df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})

# Separate features and target
X = df.drop('Churn', axis=1)
y = df['Churn']

# One-hot encode categorical variables
X = pd.get_dummies(X, drop_first=True)

# Now you can train the model
best_params = {
    'n_estimators': 100,
```

```
    'min_samples_split': 5,
    'max_depth': 7,
    'learning_rate': 0.1
}

final_gb_classifier = GradientBoostingClassifier(**best_params)
final_gb_classifier.fit(X, y)

# Save the model
model_path = r"C:\Users\JANHAVI\Desktop\final_gb_classifier.pkl"
with open(model_path, 'wb') as file:
    pickle.dump(final_gb_classifier, file)

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

```
<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
None
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No   
1  5575-GNVDE    Male              0      No         No      34          Yes   
2  3668-QPYBK    Male              0      No         No       2          Yes   
3  7795-CFOCW    Male              0      No         No      45           No   
4  9237-HQITU  Female              0      No         No       2          Yes   

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No   
1                No             DSL            Yes  ...              Yes   
2                No             DSL            Yes  ...               No   
3  No phone service             DSL            Yes  ...              Yes   
4                No     Fiber optic             No  ...               No   

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes   
1          No          No              No        One year               No   
2          No          No              No  Month-to-month              Yes   
3         Yes          No              No        One year               No   
4          No          No              No  Month-to-month              Yes   

              PaymentMethod MonthlyCharges  TotalCharges Churn  
0          Electronic check          29.85         29.85    No  
1              Mailed check          56.95        1889.5    No  
2              Mailed check          53.85        108.15   Yes  
3  Bank transfer (automatic)          42.30       1840.75    No  
4          Electronic check          70.70        151.65   Yes  

[5 rows x 21 columns]
```

Accuracy Features

In [ ]:
```
import pickle
import pandas as pd

# Use the correct full path where you saved the model
model_path = r"C:\Users\JANHAVI\Desktop\final_gb_classifier.pkl"

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

# Prepare your own data for testing
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]
})
```

In [ ]:

In [ ]: