Telecom Churn Analysis

In [1]:
```
#import the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
```

Load the data file

In [2]:
```
telco_base_data = pd.read_csv('Telco-Customer-Churn.csv')
```

Look at the top 5 records of data

In [3]:
```
telco_base_data.columns
```

Out[3]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

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

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
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	

5 rows × 21 columns

Check the various attributes of data like shape (rows and cols), Columns, datatypes

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

Out[5]: (7043, 21)

In [6]:
```
telco_base_data.columns.values
```

Out[6]:
```
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 [7]:
```
# Checking the data types of all the columns
telco_base_data.dtypes
```

Out[7]:
```
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         object
Churn                object
dtype: object
```

In [8]:
```
# Check the descriptive statistics of numeric variables
telco_base_data.describe()
```

Out[8]:

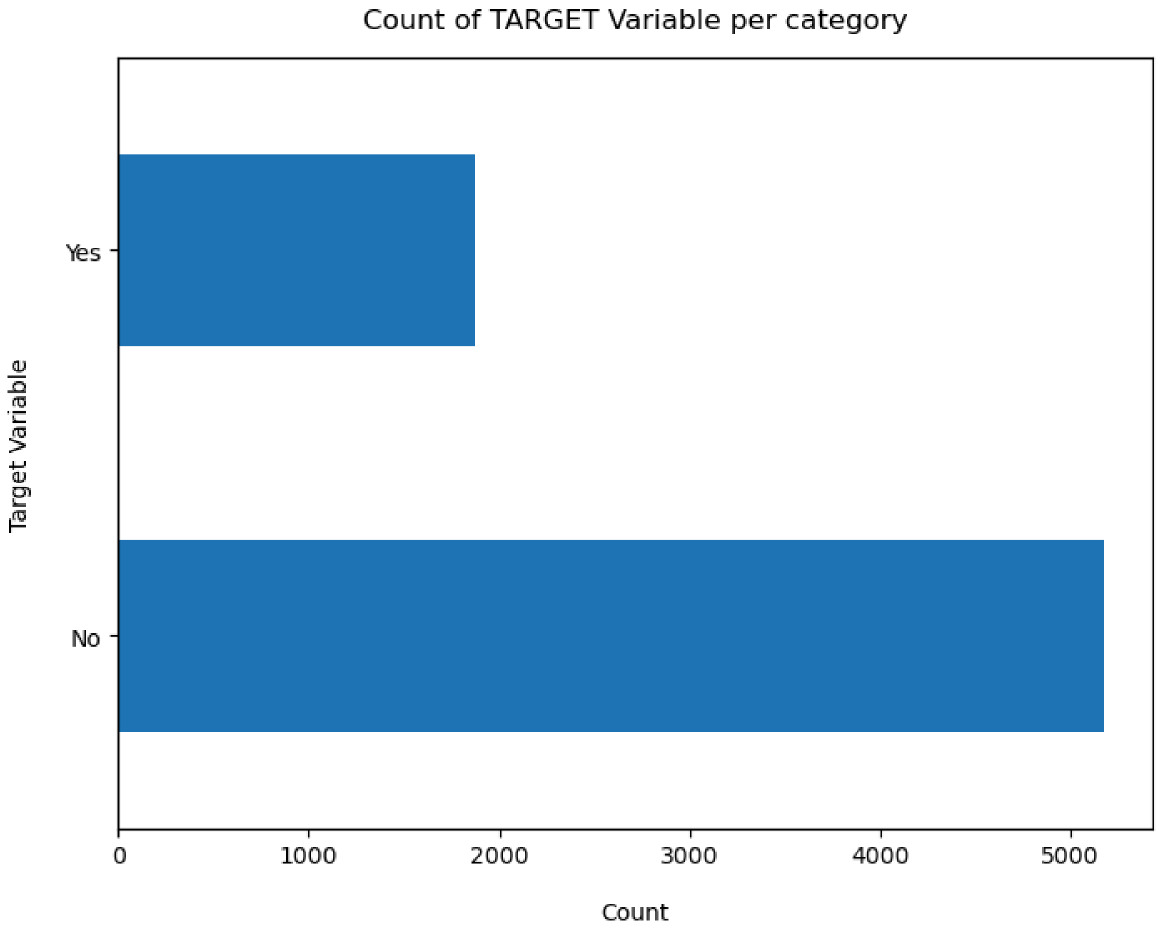
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

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

75% customers have tenure less than 55 months

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

In [9]:
```
telco_base_data['Churn'].value_counts().plot(kind='barh', figsize=(8, 6))
plt.xlabel("Count", labelpad=14)
plt.ylabel("Target Variable", labelpad=14)
plt.title("Count of TARGET Variable per category", y=1.02);
```



In [10]:
```
100*telco_base_data['Churn'].value_counts()/len(telco_base_data['Churn'])
```

Out[10]:
```
Churn
No     73.463013
Yes    26.536987
Name: count, dtype: float64
```

In [11]:
```
telco_base_data['Churn'].value_counts()
```

Out[11]:
```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

- Data is highly imbalanced, ratio = 73:27
- So we analyse the data with other features while taking the target values separately to get some insights.

In [12]:
```python
# Concise Summary of the dataframe, as we have too many columns, we are using th
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      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [13]:
```python
missing = pd.DataFrame((telco_base_data.isnull().sum())*100/telco_base_data.shap
```

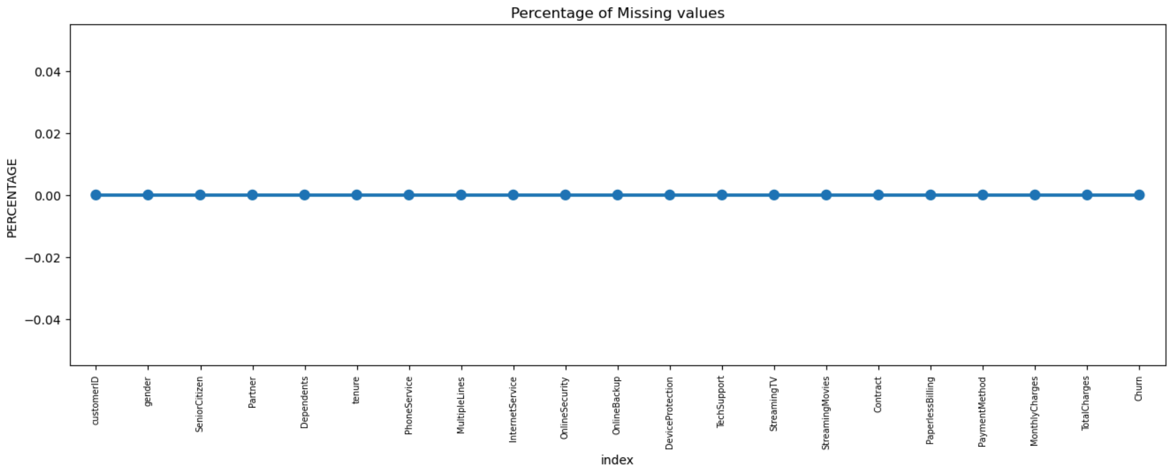
In [14]:
```python
missing
```

Out[14]:

	index	0
0	customerID	0.0
1	gender	0.0
2	SeniorCitizen	0.0
3	Partner	0.0
4	Dependents	0.0
5	tenure	0.0
6	PhoneService	0.0
7	MultipleLines	0.0
8	InternetService	0.0
9	OnlineSecurity	0.0
10	OnlineBackup	0.0
11	DeviceProtection	0.0
12	TechSupport	0.0
13	StreamingTV	0.0
14	StreamingMovies	0.0
15	Contract	0.0
16	PaperlessBilling	0.0
17	PaymentMethod	0.0
18	MonthlyCharges	0.0
19	TotalCharges	0.0
20	Churn	0.0

In [15]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(16, 5))
ax = sns.pointplot(x='index', y=0, data=missing)
plt.xticks(rotation=90, fontsize=7)
plt.title("Percentage of Missing values")
plt.ylabel("PERCENTAGE")
plt.show()
```



Missing Data - Initial Intuition

- Here, we don't have any missing data.

General Thumb Rules:

- For features with less missing values- can use regression to predict the missing values or fill with the mean of the values present, depending on the feature.
- For features with very high number of missing values- it is better to drop those columns as they give very less insight on analysis.
- As there's no thumb rule on what criteria do we delete the columns with high number of missing values, but generally you can delete the columns, if you have more than 30-40% of missing values. But again there's a catch here, for example, Is_Car & Car_Type, People having no cars, will obviously have Car_Type as NaN (null), but that doesn't make this column useless, so decisions has to be taken wisely.

Data Cleaning

1. Create a copy of base data for manupulation & processing

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

2. Total Charges should be numeric amount. Let's convert it to numerical data type

The errors='coerce' parameter is used to handle errors.

When errors is set to 'coerce', it means that if there are any errors encountered while converting data to numeric, those errors will be replaced with NaN (Not a Number) values.

In [17]:
```
telco_data.TotalCharges = pd.to_numeric(telco_data.TotalCharges, errors='coerce'
telco_data.isnull().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
```

3. As we can see there are 11 missing values in TotalCharges column. Let's check these records

In [18]:
```
telco_data.loc[telco_data ['TotalCharges'].isnull() == True]
```

Out[18]:

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

11 rows × 21 columns

4. Missing Value Treatement

Since the % of these records compared to total dataset is very low ie 0.15%, it is safe to ignore them from further processing.

In [19]:

```
#Removing missing values
telco_data.dropna(how = 'any', inplace = True)

#telco_data.fillna(0)
```

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

In [20]:

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

72

In [21]:

```
# Group the tenure in bins of 12 months
labels = ["{0} - {1}".format(i, i + 11) for i in range(1, 72, 12)]
```

```
telco_data['tenure_group'] = pd.cut(telco_data.tenure, range(1, 80, 12), right=F
```

by cutting the 'tenure' column into intervals defined by the range(1, 80, 12) function, which generates intervals starting from 1 and ending at 72 (since the range stops before the end value). The right=False parameter means that the intervals are closed on the left and open on the right.

In [22]:
```
telco_data['tenure_group'].value_counts()
```

Out[22]:
```
tenure_group
1 - 12     2175
61 - 72    1407
13 - 24    1024
25 - 36     832
49 - 60     832
37 - 48     762
Name: count, dtype: int64
```

6. Remove columns not required for processing

In [23]:
```
#drop column customerID and tenure
telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telco_data.head()
```

Out[23]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetSe
0	Female	0	Yes	No	No	No phone service	
1	Male	0	No	No	Yes	No	
2	Male	0	No	No	Yes	No	
3	Male	0	No	No	No	No phone service	
4	Female	0	No	No	Yes	No	Fiber

Data Exploration

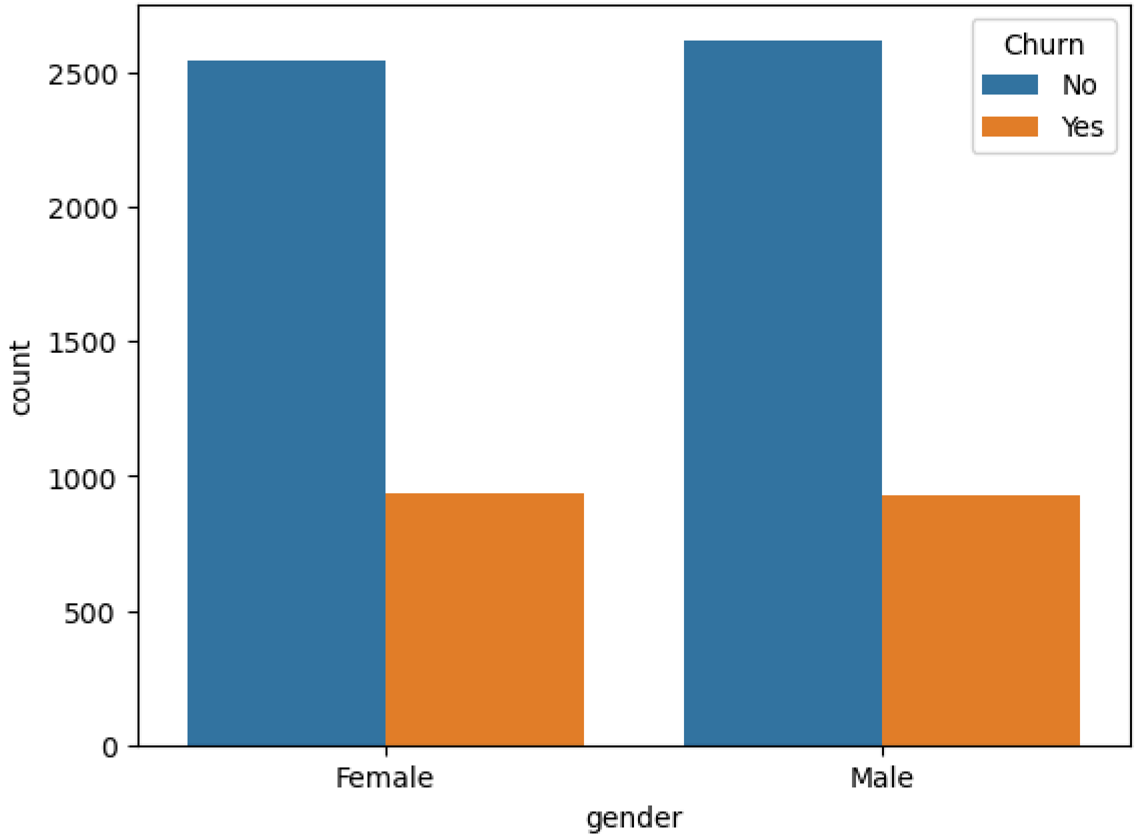
1. Plot distibution of individual predictors by churn

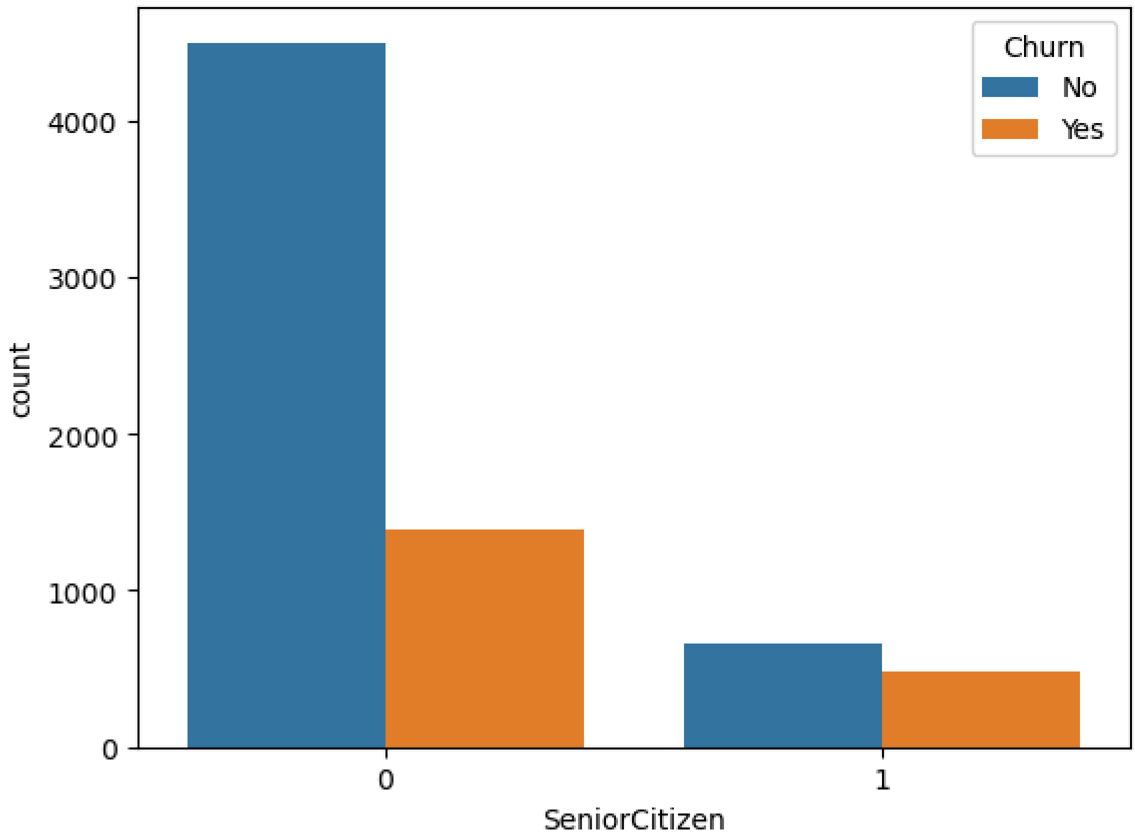
Univariate Analysis

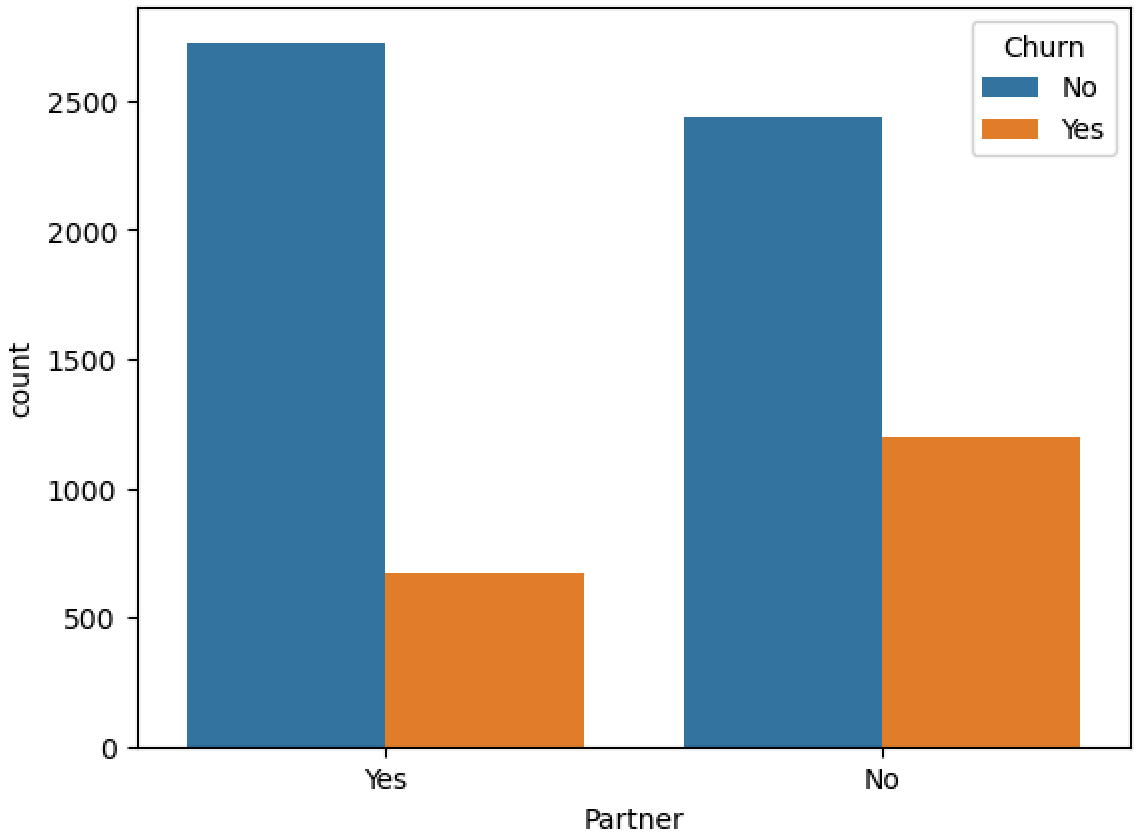
In [24]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
```

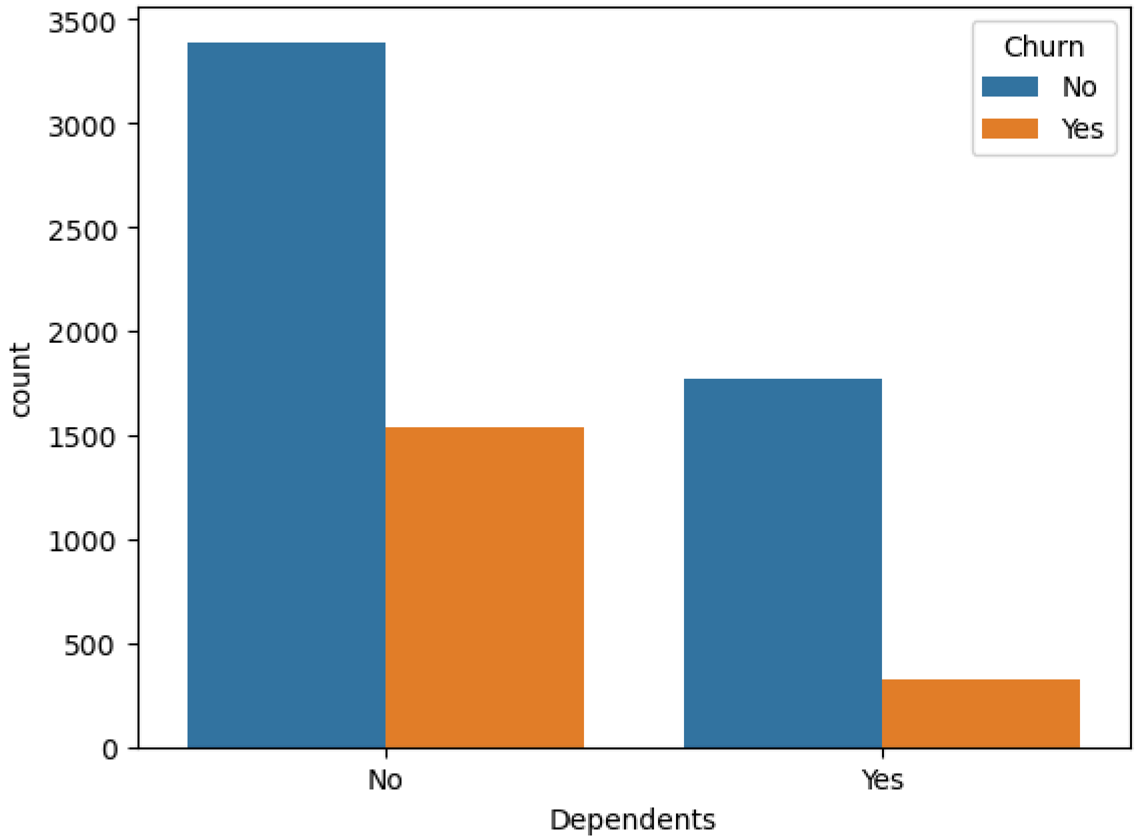
In [25]:
```
for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges',
    plt.figure(i)
```

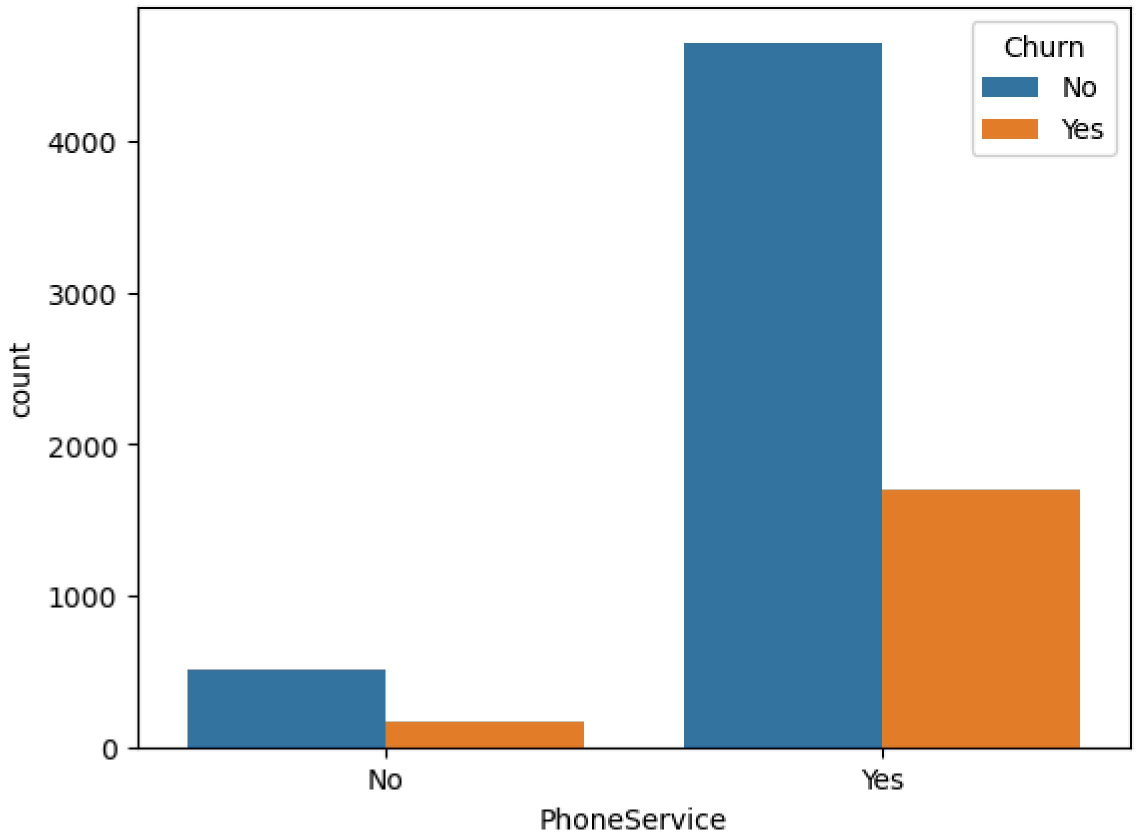
```
sns.countplot(data=telco_data, x=predictor, hue='Churn')
```



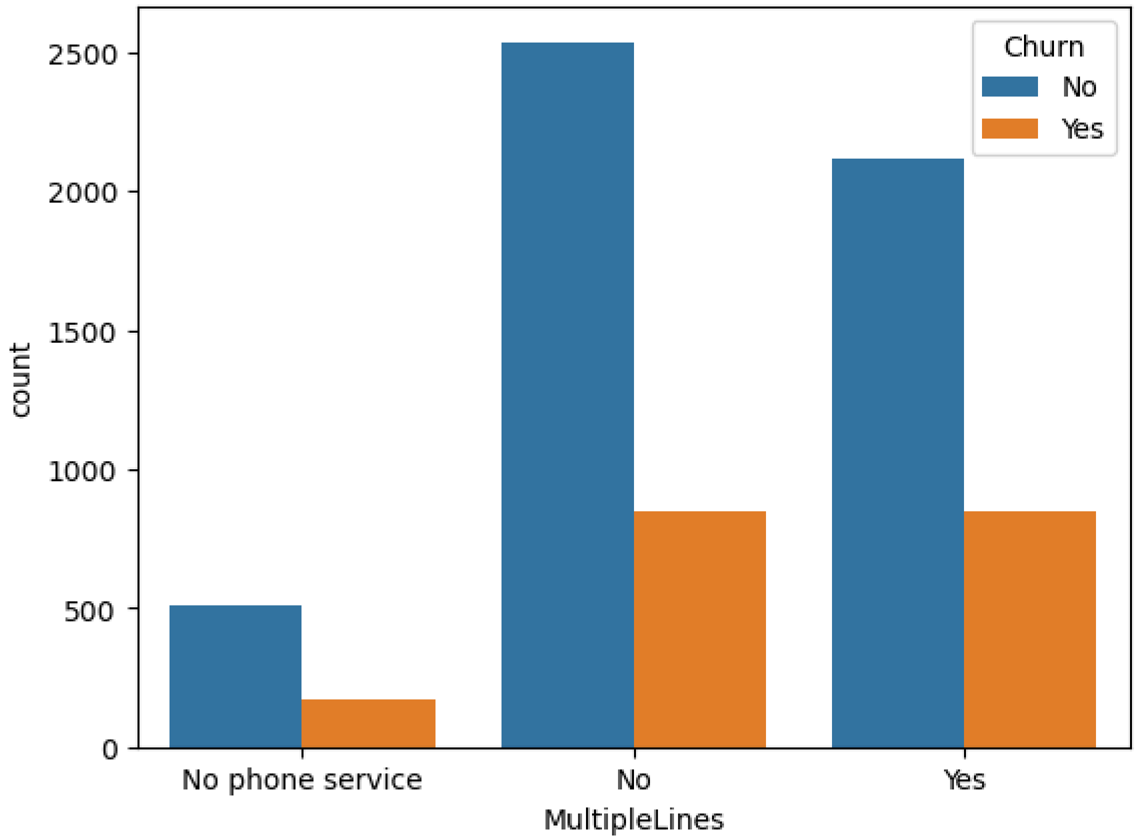



Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
Partner

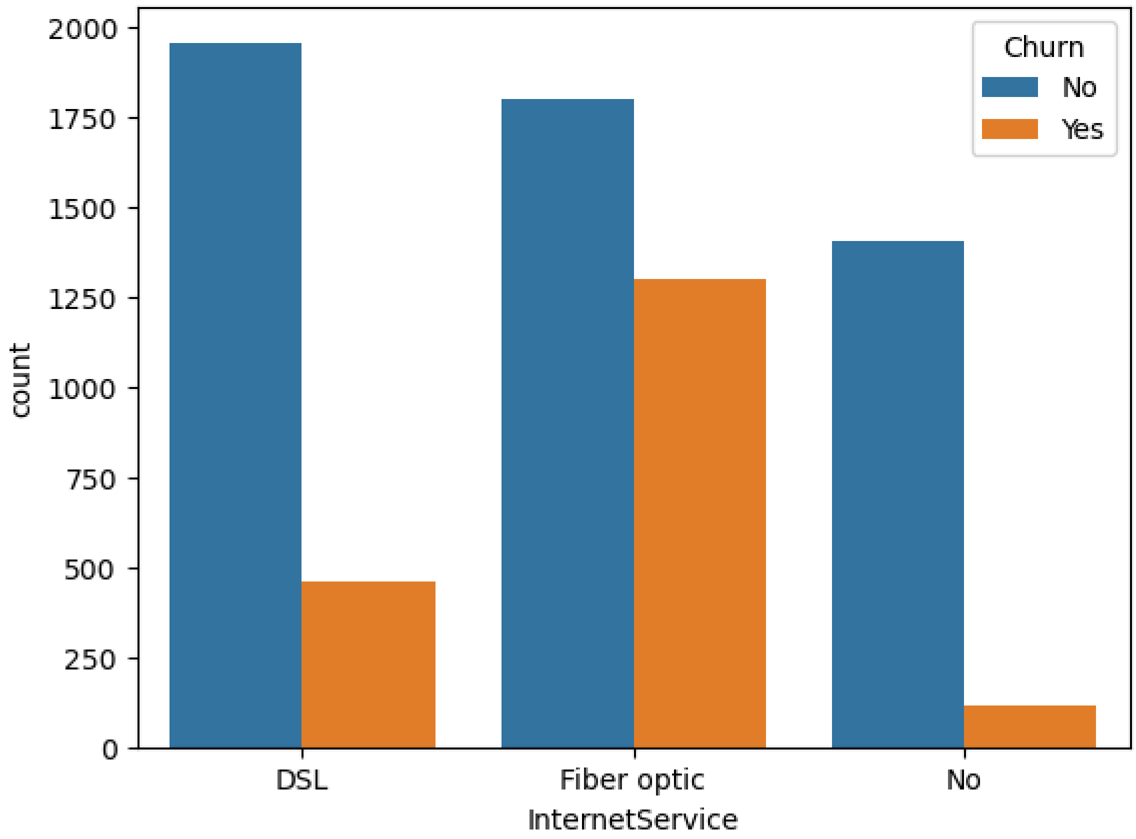

Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents



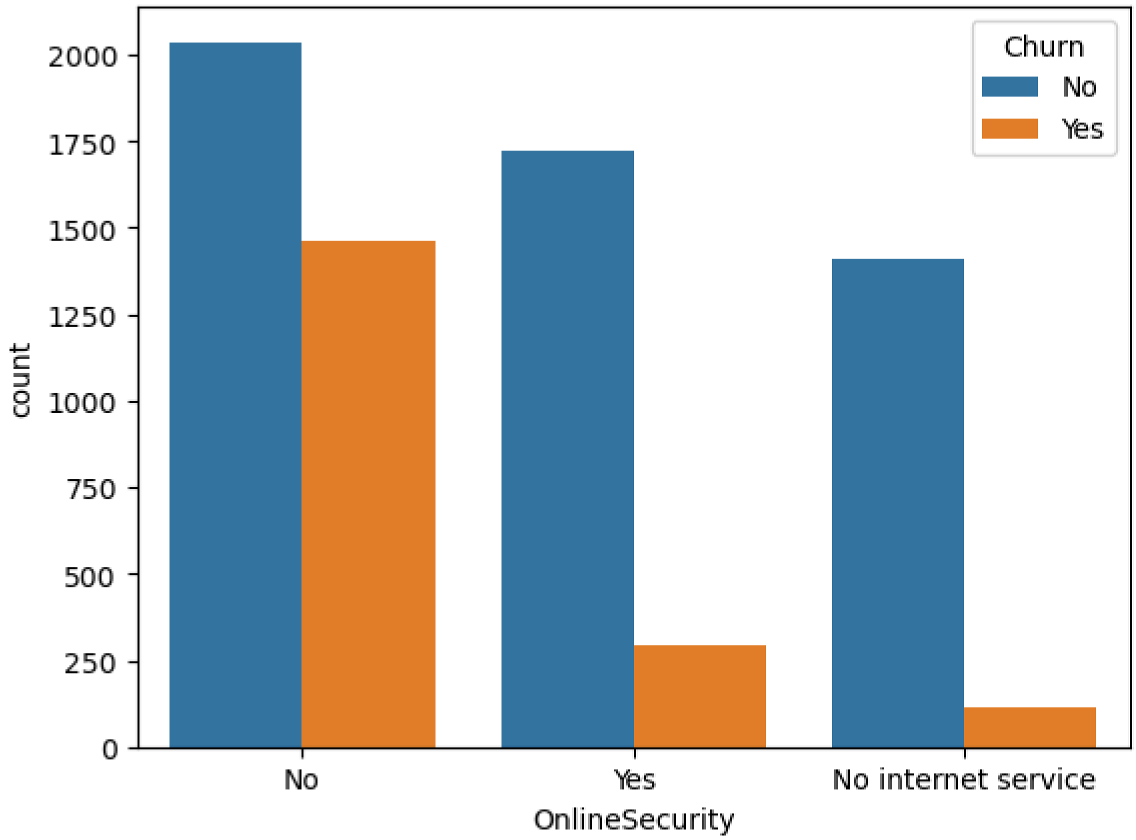
Churn
No
Yes
count
4000
3000
2000
1000
0
No
Yes
PhoneService



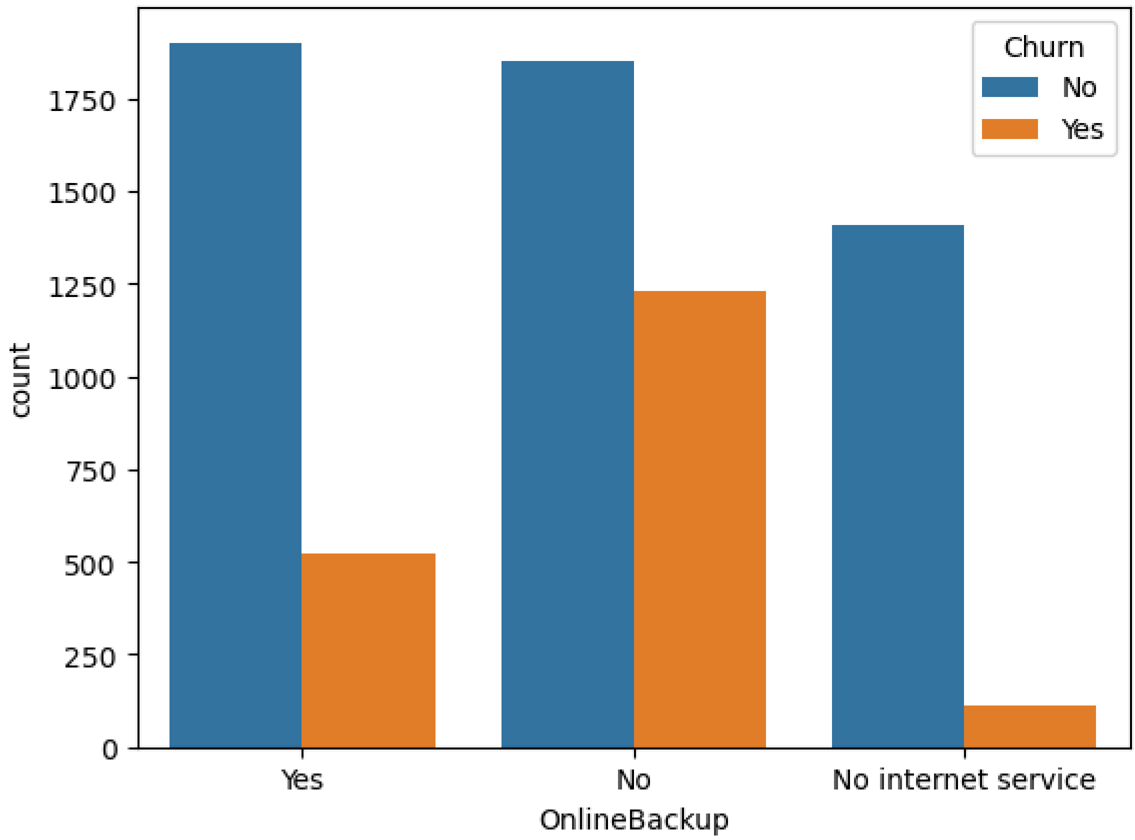
Churn
No
Yes
count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

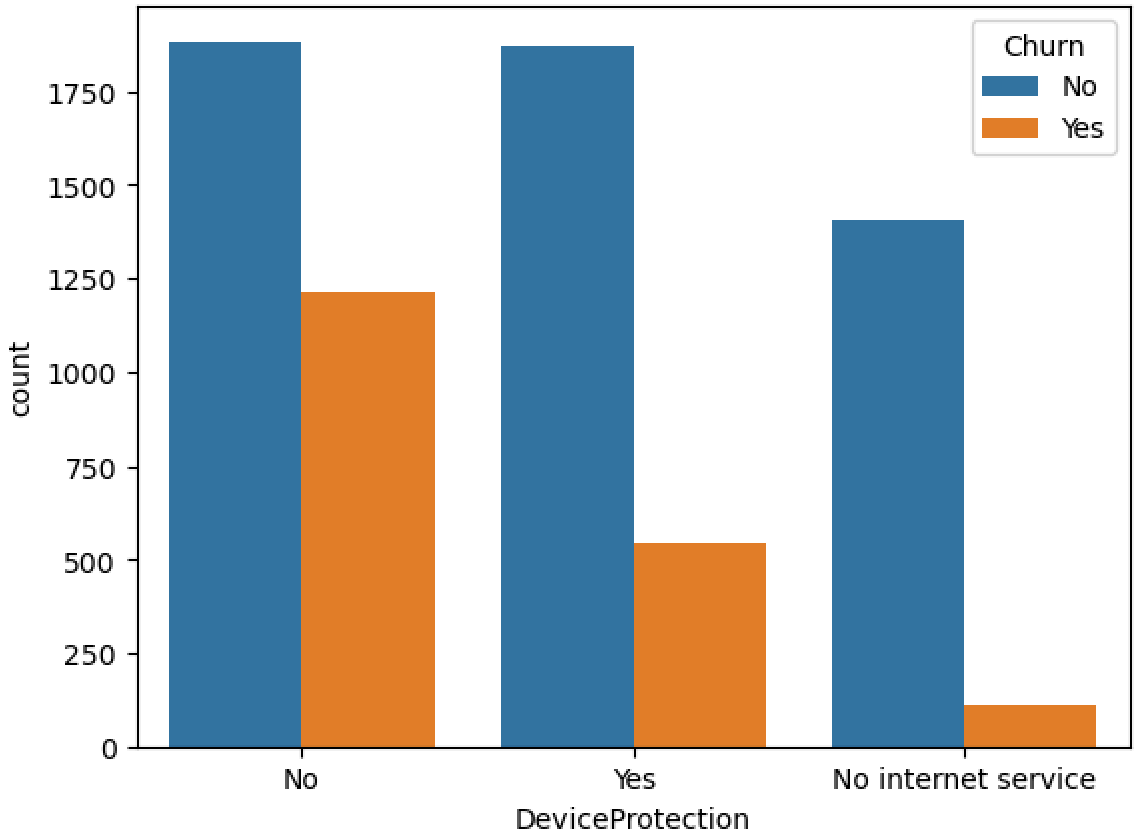


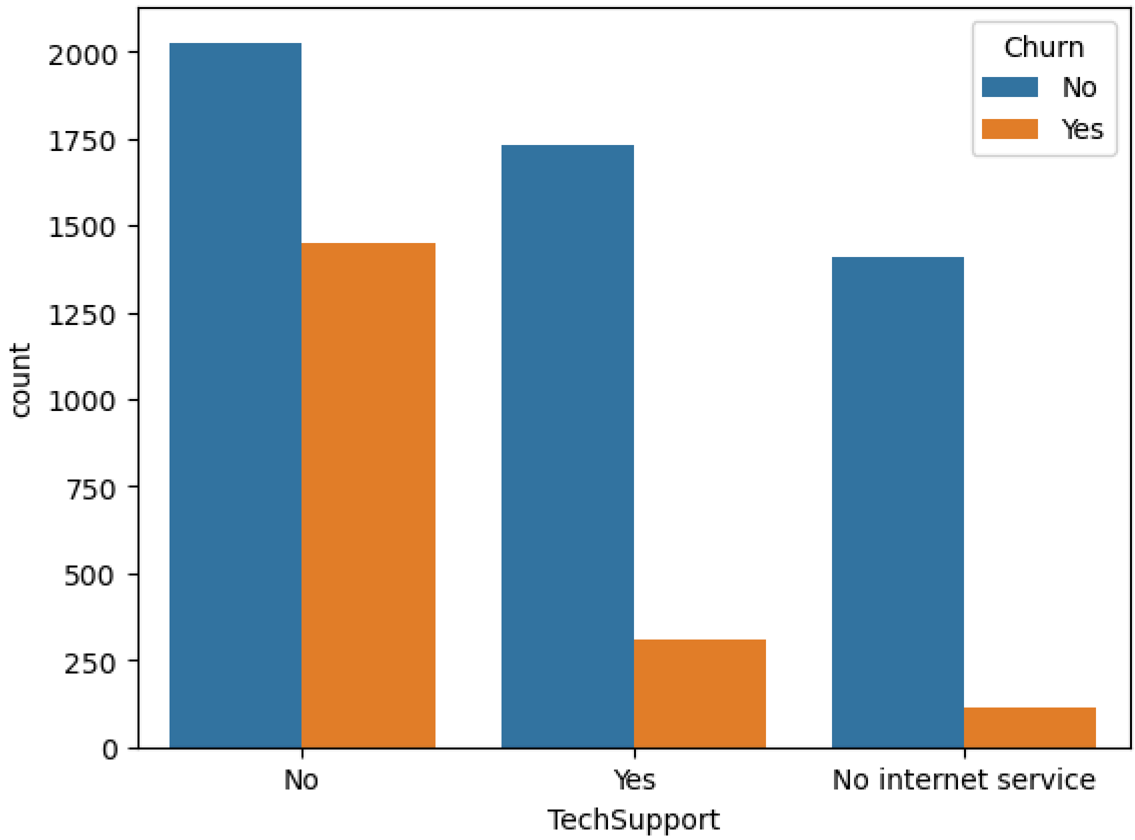
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService



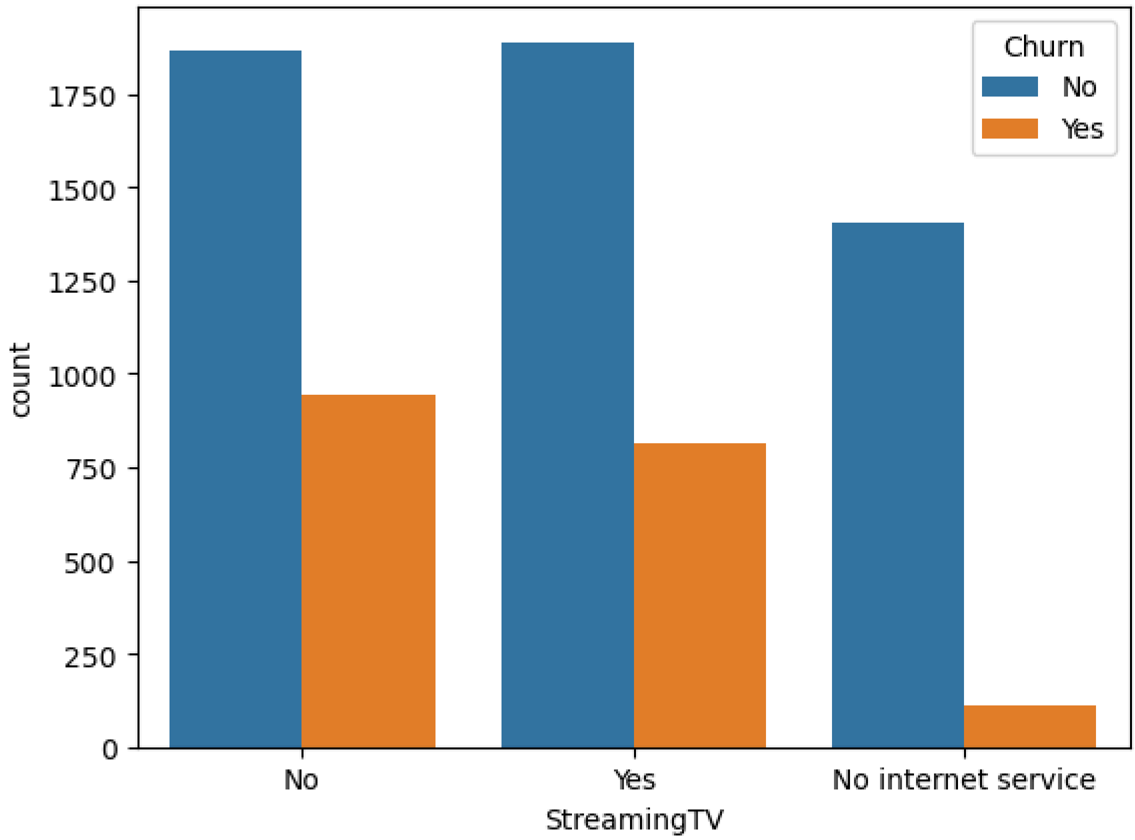
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity


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

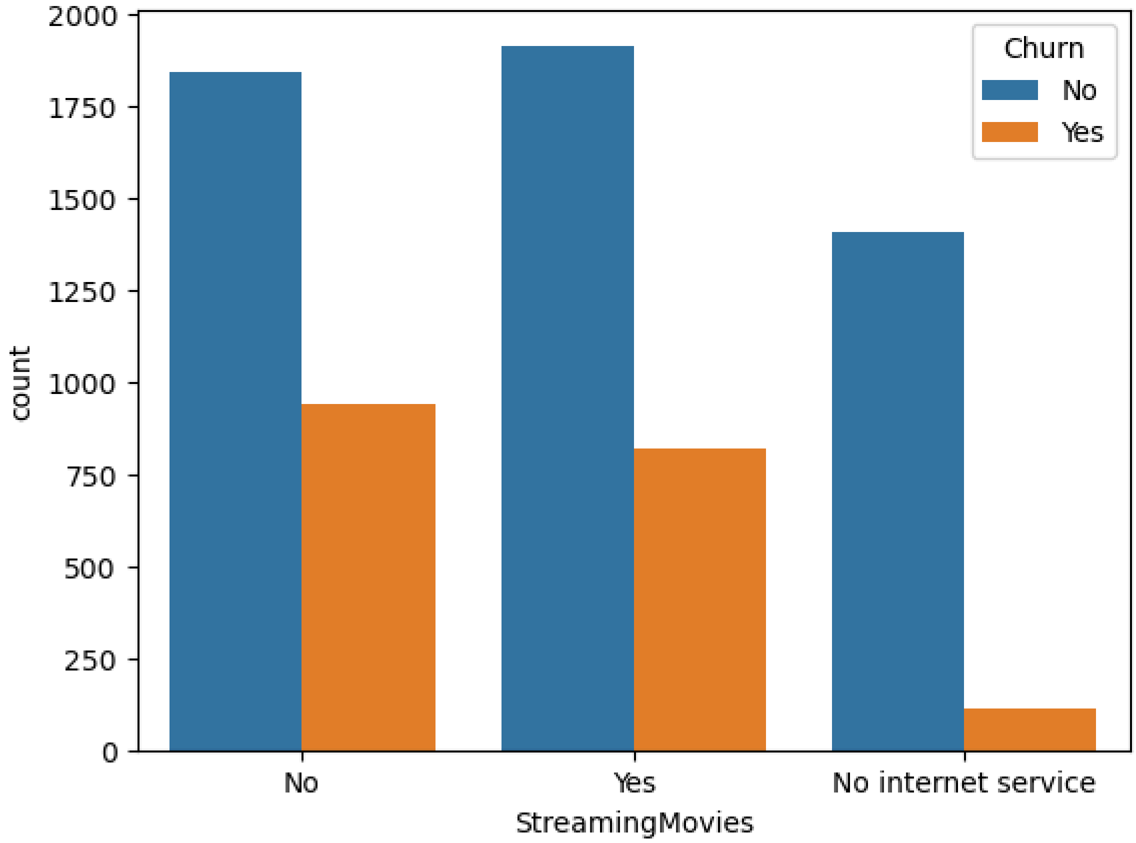

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection



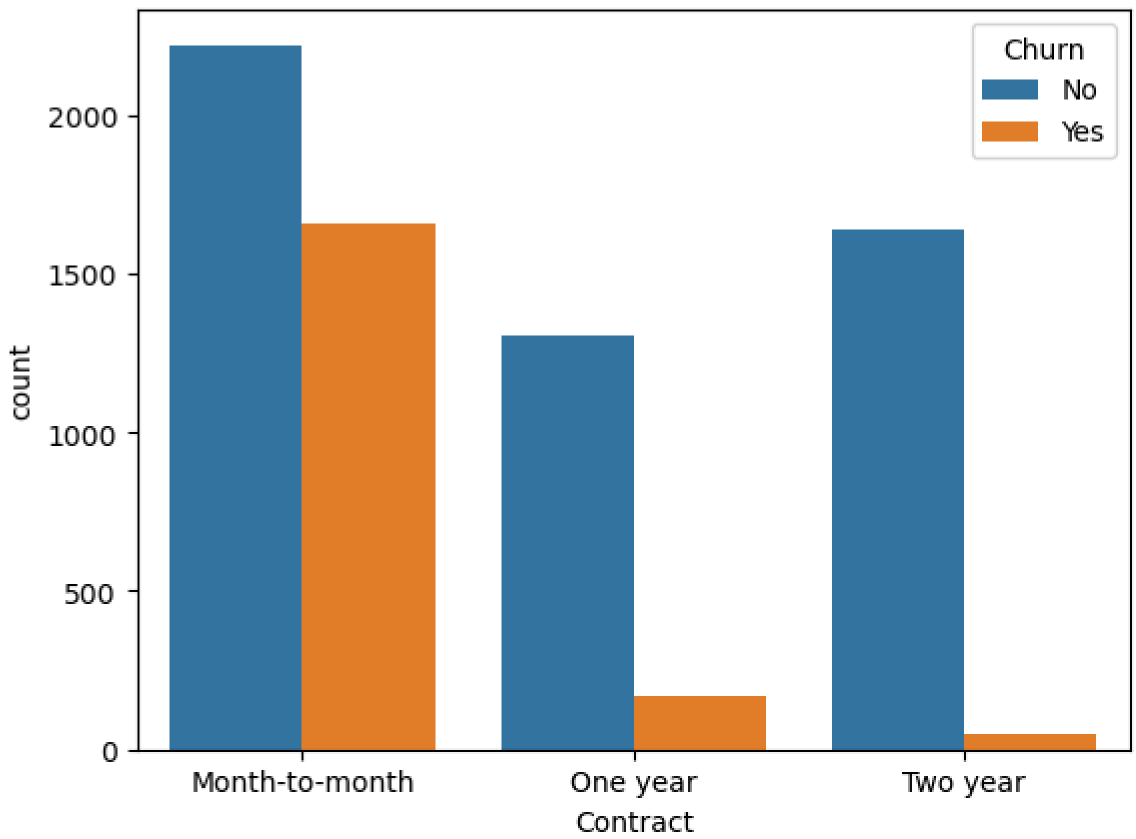
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
TechSupport



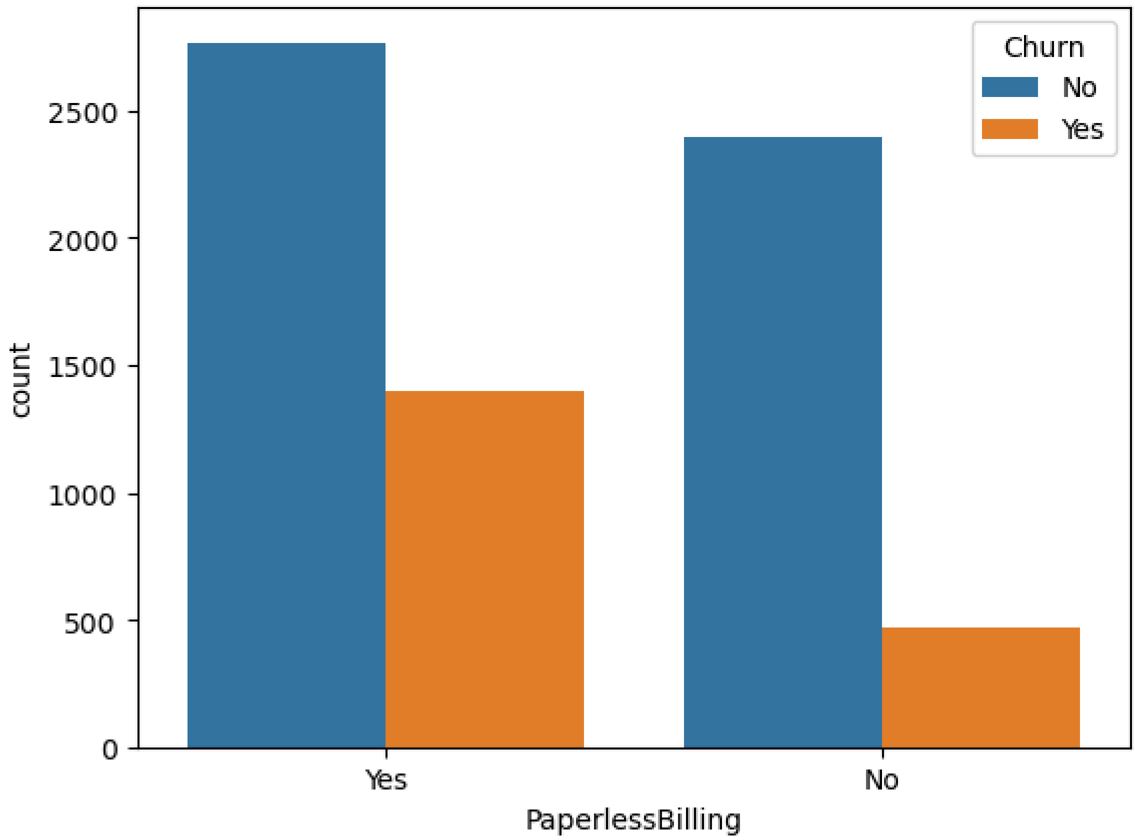
Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingTV

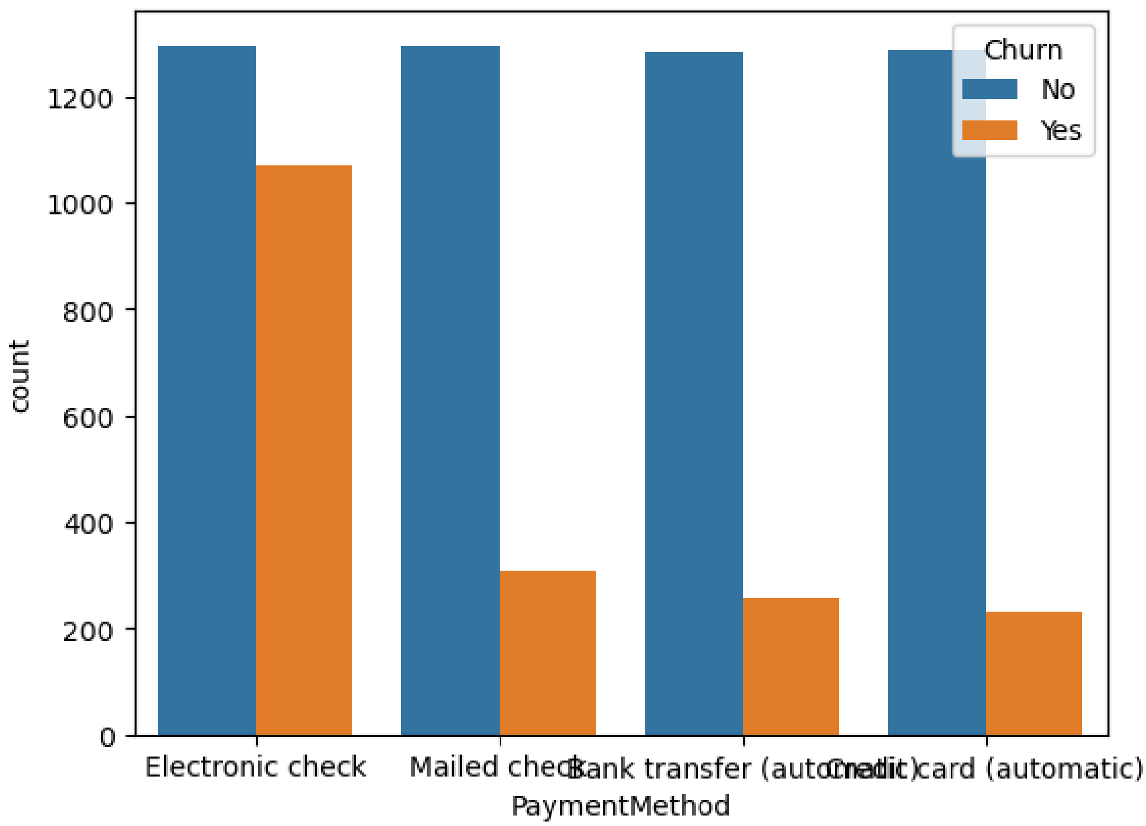


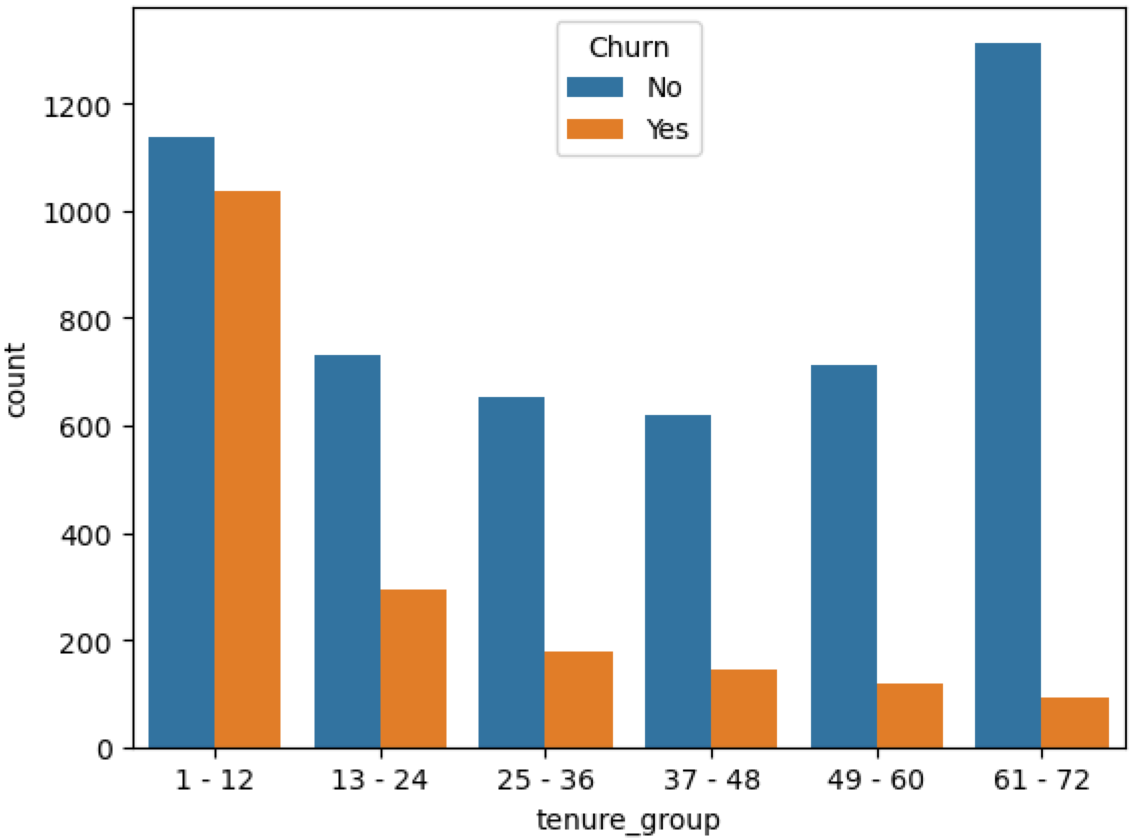
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingMovies



Churn
No
Yes
count
2000
1500
1000
500
0
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



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

In [26]:
```
telco_data['Churn'] = np.where(telco_data.Churn == 'Yes',1,0)
```

In [27]:
```
telco_data.head()
```

Out[27]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetSe
0	Female	0	Yes	No	No	No phone service	
1	Male	0	No	No	Yes	No	
2	Male	0	No	No	Yes	No	
3	Male	0	No	No	No	No phone service	
4	Female	0	No	No	Yes	No	Fiber

3. Convert all the categorical variables into dummy variables

In [28]:
```
telco_data_dummies = pd.get_dummies(telco_data)
telco_data_dummies.head()
```

Out[28]:

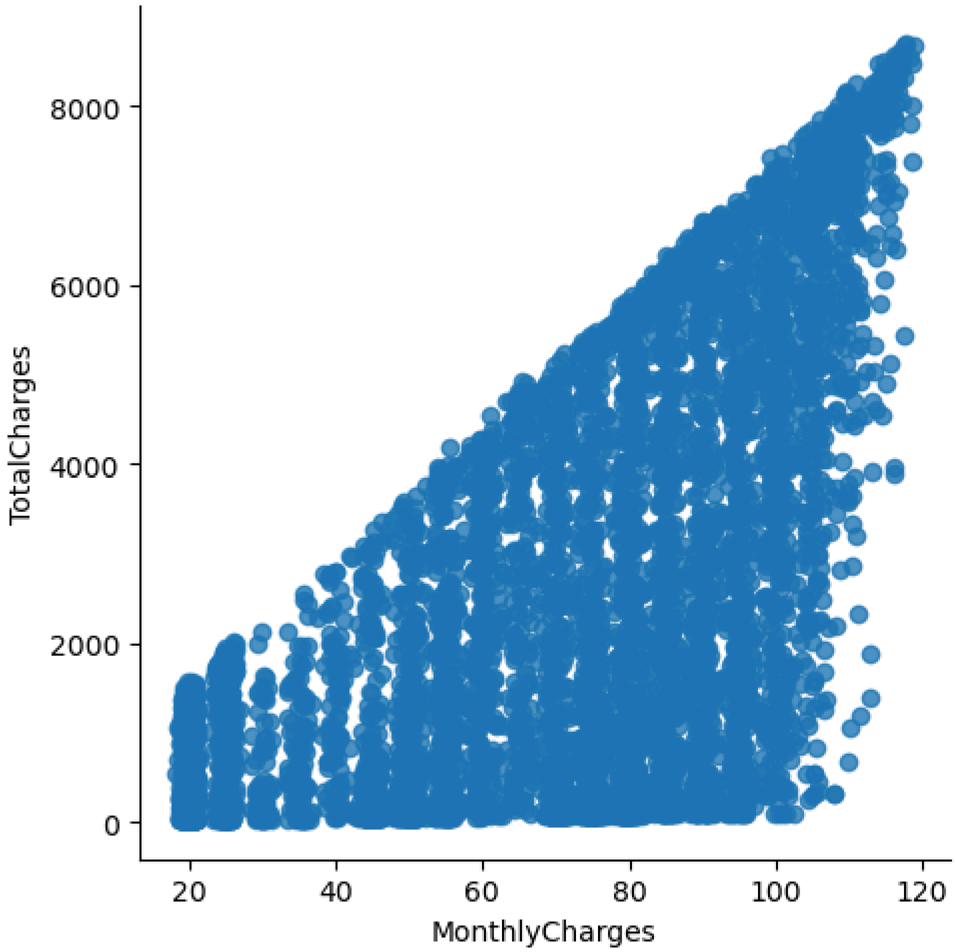
	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	I
0	0	29.85	29.85	0	True	False	
1	0	56.95	1889.50	0	False	True	
2	0	53.85	108.15	1	False	True	
3	0	42.30	1840.75	0	False	True	
4	0	70.70	151.65	1	True	False	

5 rows × 51 columns

9. Relationship between Monthly Charges and Total Charges

In [29]:
```
sns.lmplot(data=telco_data_dummies, x='MonthlyCharges', y='TotalCharges', fit_re
```

Out[29]:
```
<seaborn.axisgrid.FacetGrid at 0x229de97dc90>
```



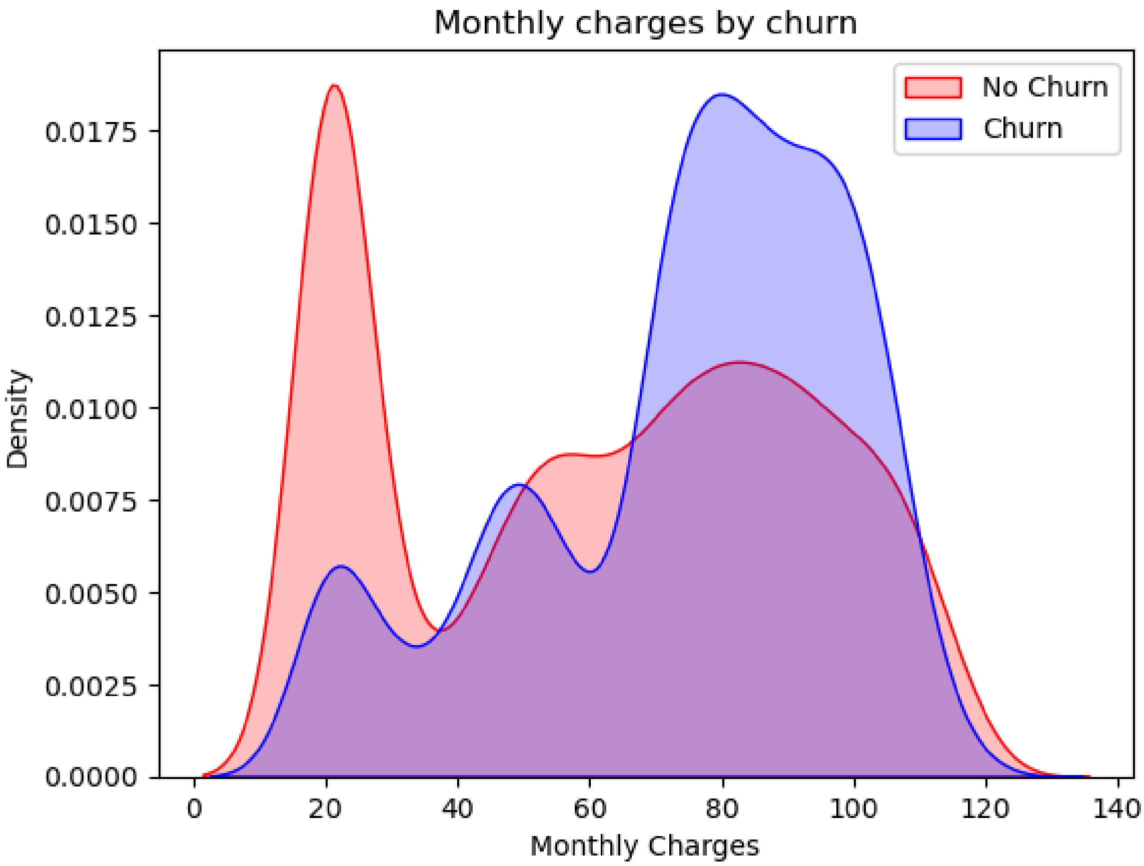
Total Charges increase as Monthly Charges increase - as expected.

10. Churn by Monthly Charges and Total Charges

In [30]:
```
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"]
                color="Red", shade = True)
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"]
```

```
                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')
```

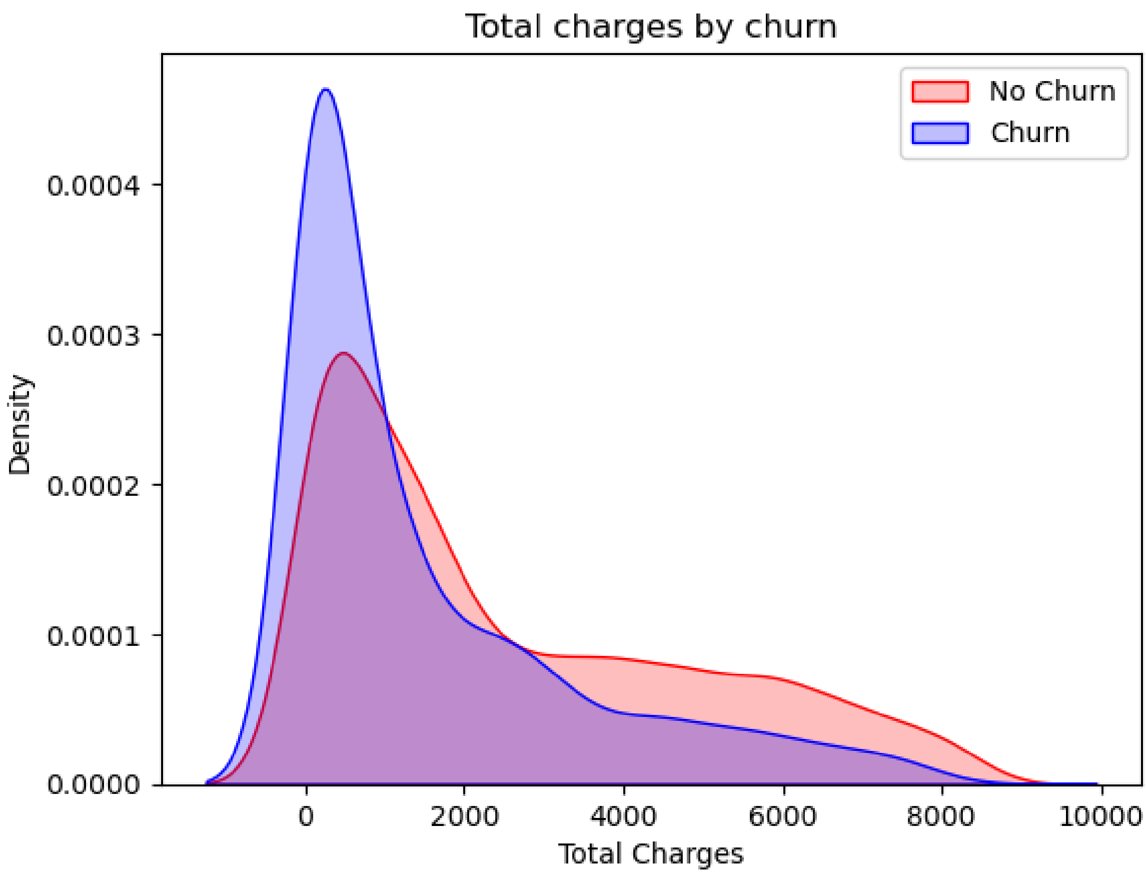
Out[30]: Text(0.5, 1.0, 'Monthly charges by churn')



Insight: Churn is high when Monthly Charges ar high

In [31]:
```
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["Churn"] =
                color="Red", shade = True)
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["Churn"] =
                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[31]: Text(0.5, 1.0, 'Total charges by churn')



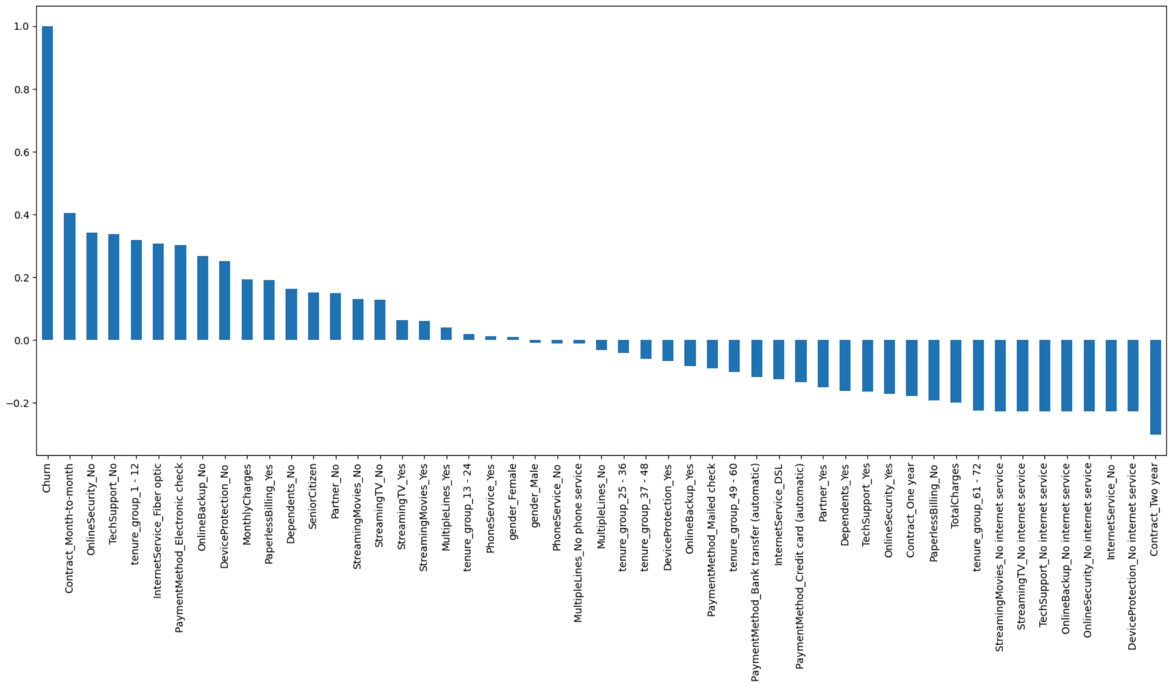
Surprising insight as higher Churn at lower Total Charges

However if we combine the insights of 3 parameters i.e. Tenure, Monthly Charges & Total Charges then the picture is bit clear :- Higher Monthly Charge at lower tenure results into lower Total Charge. Hence, all these 3 factors viz **Higher Monthly Charge**, **Lower tenure** and **Lower Total Charge** are linkd to **High Churn**.

11. Build a corelation of all predictors with 'Churn'

In [32]:
```
plt.figure(figsize=(20,8))
telco_data_dummies.corr()['Churn'].sort_values(ascending = False).plot(kind='bar
```

Out[32]:
```
<Axes: >
```



Derived Insight:

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

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

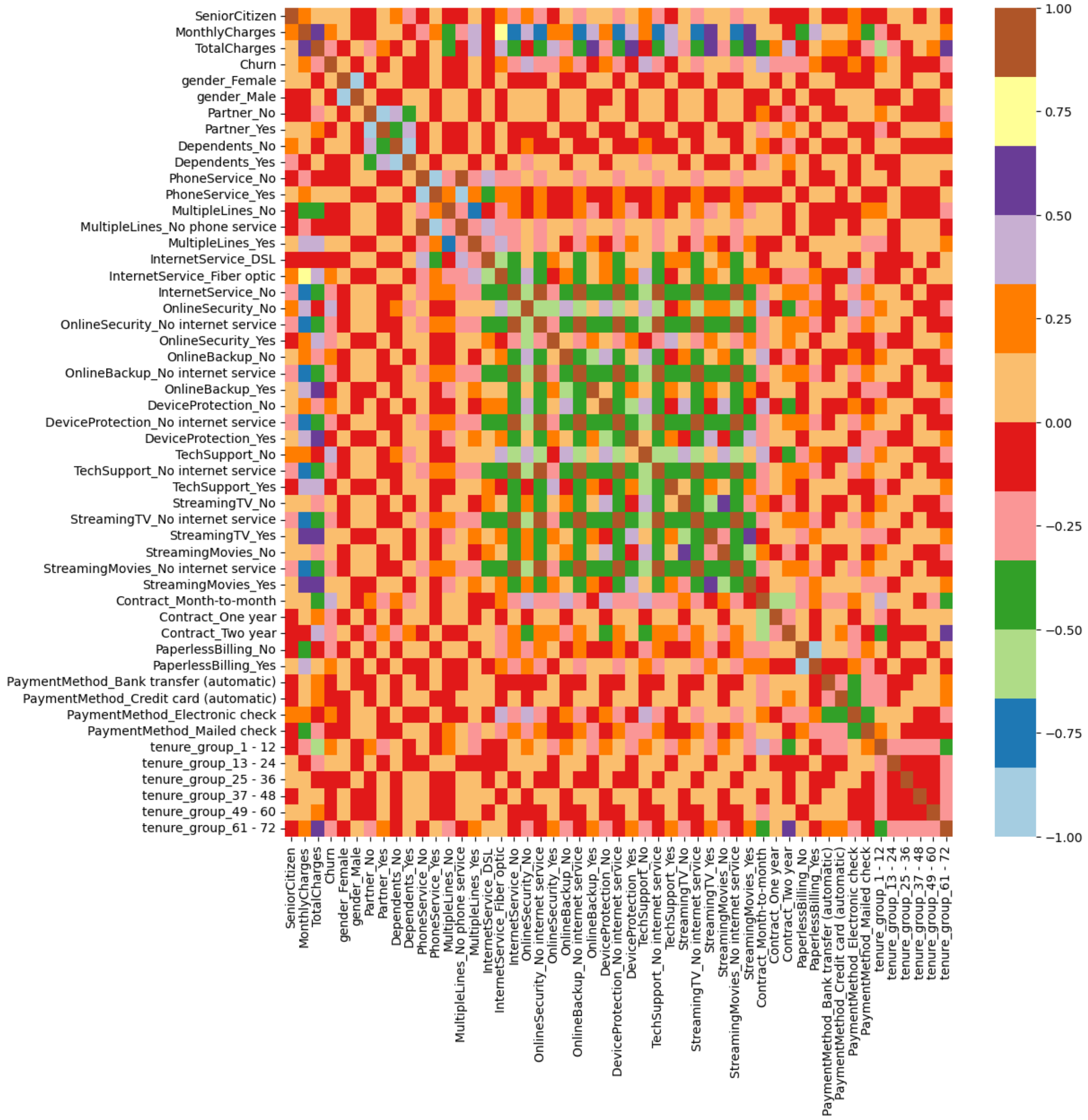
Factors like **Gender**, **Availability of PhoneService** and **# of multiple lines** have alomost **NO** impact on Churn

This is also evident from the **Heatmap** below

In [33]:

```
plt.figure(figsize=(12,12))
sns.heatmap(telco_data_dummies.corr(), cmap="Paired")
```

Out[33]: <Axes: >



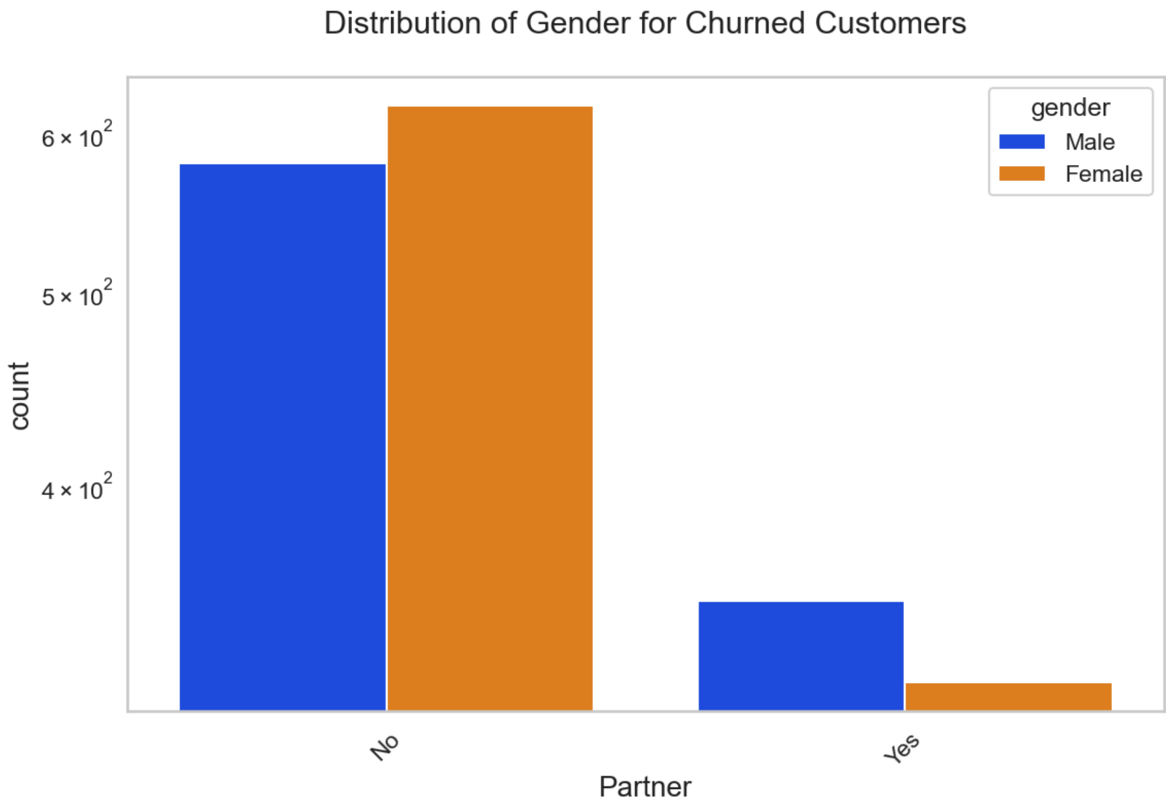
Bivariate Analysis

In [34]:
```
new_df1_target0=telco_data.loc[telco_data["Churn"]==0]
new_df1_target1=telco_data.loc[telco_data["Churn"]==1]
```

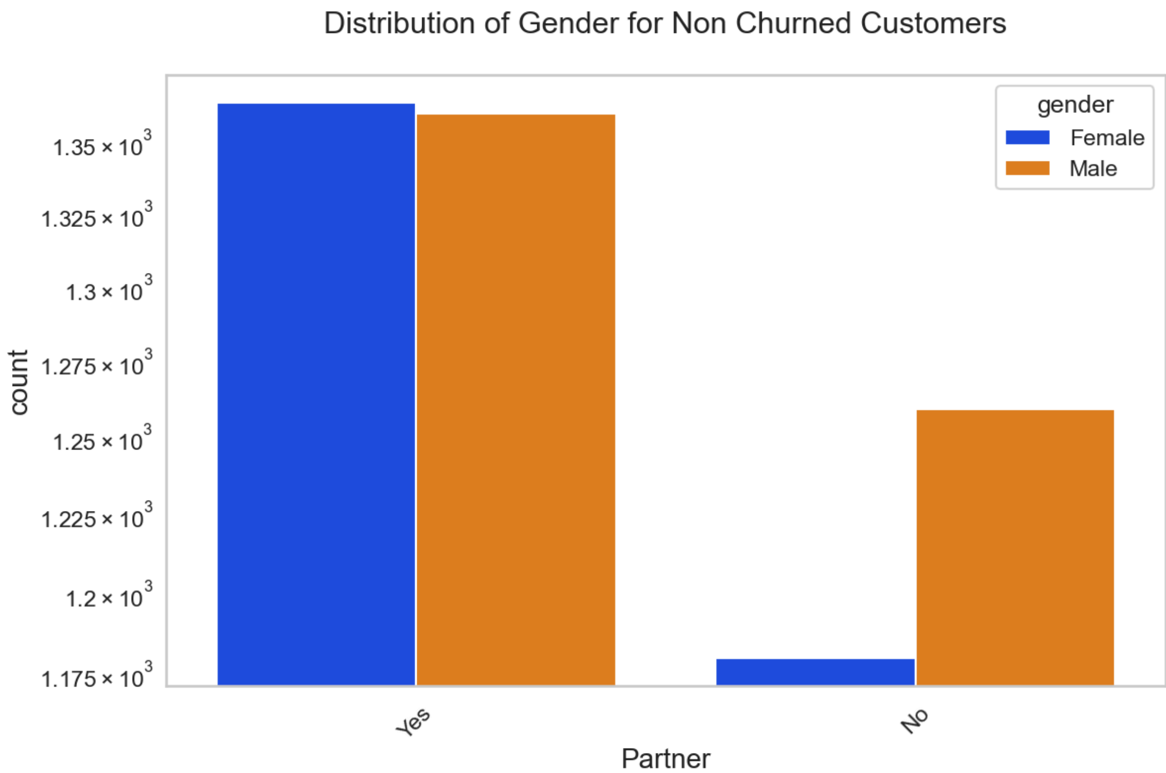
In [35]:
```
def uniplot(df,col,title,hue =None):

    sns.set_style('whitegrid')
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 20
    plt.rcParams['axes.titlesize'] = 22
    plt.rcParams['axes.titlepad'] = 30 #These lines set the size and padding of


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(df[col].unique()) + 7 + 4*len(temp.unique())
    #This line calculates the width of the plot based on the number of unique va
    #in the specified column (col) and the unique values in the hue parameter.
    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
    #This line creates the count plot using seaborn's countplot function. It spe
    #the column (col) to plot on the x-axis, the order of the categories based o
    #the hue parameter for further categorization, and the color palette ('brigh
    plt.show()
```

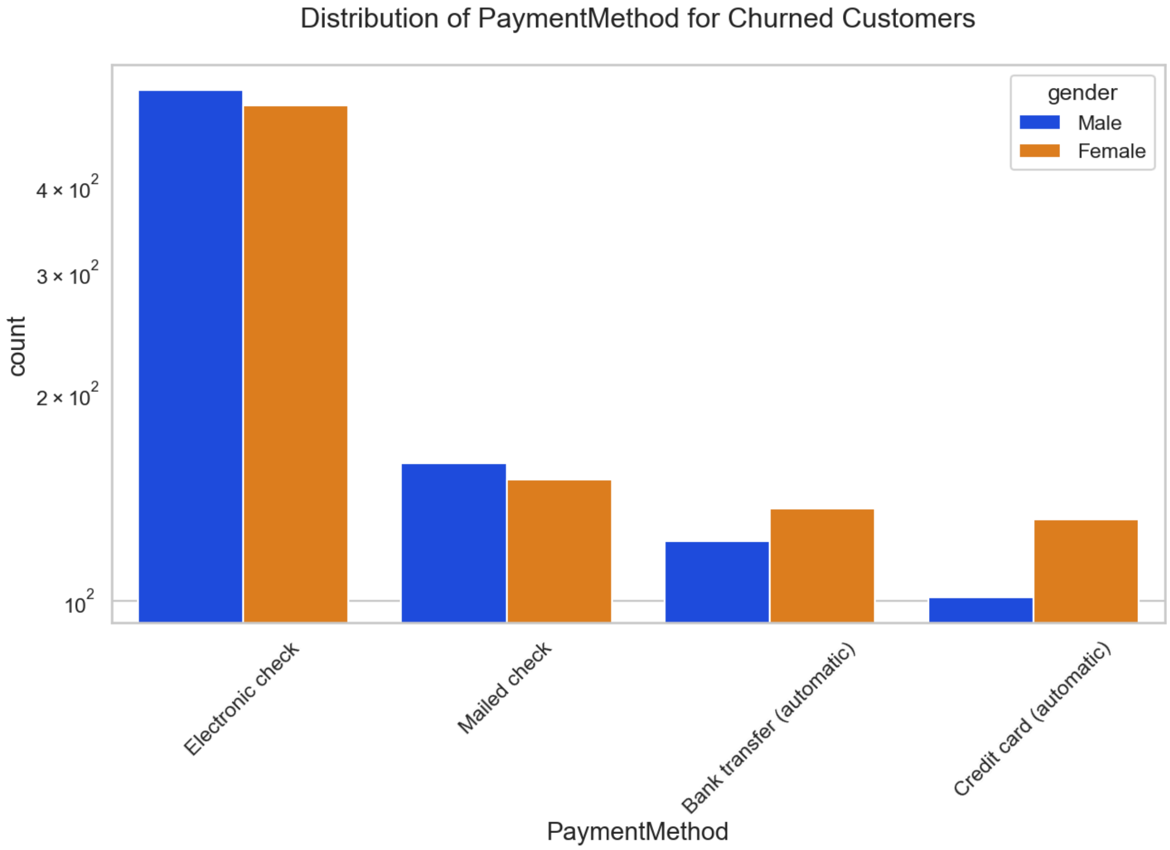
In [36]:
```
#The uniplot() function you've provided is intended to create a count plot using
uniplot(new_df1_target1,col='Partner',title='Distribution of Gender for Churned
```



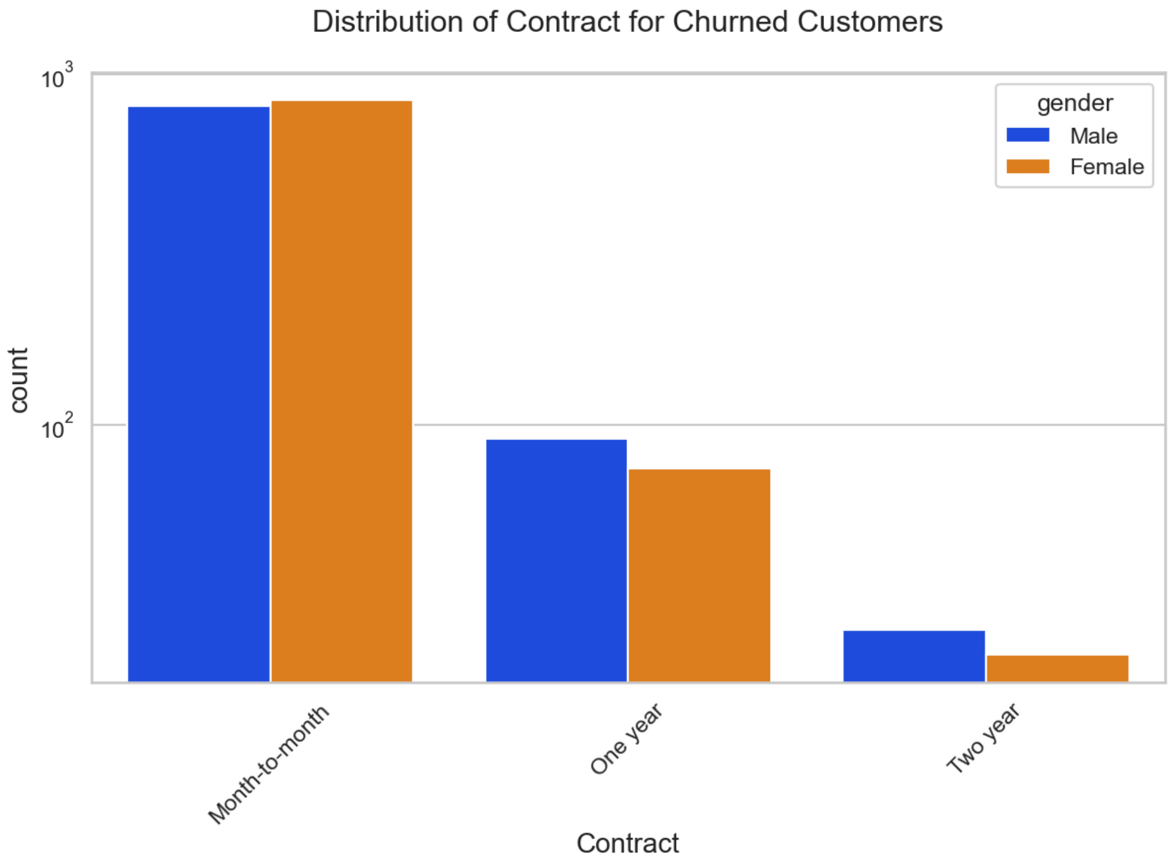
In [37]:
```
uniplot(new_df1_target0,col='Partner',title='Distribution of Gender for Non Chur
```



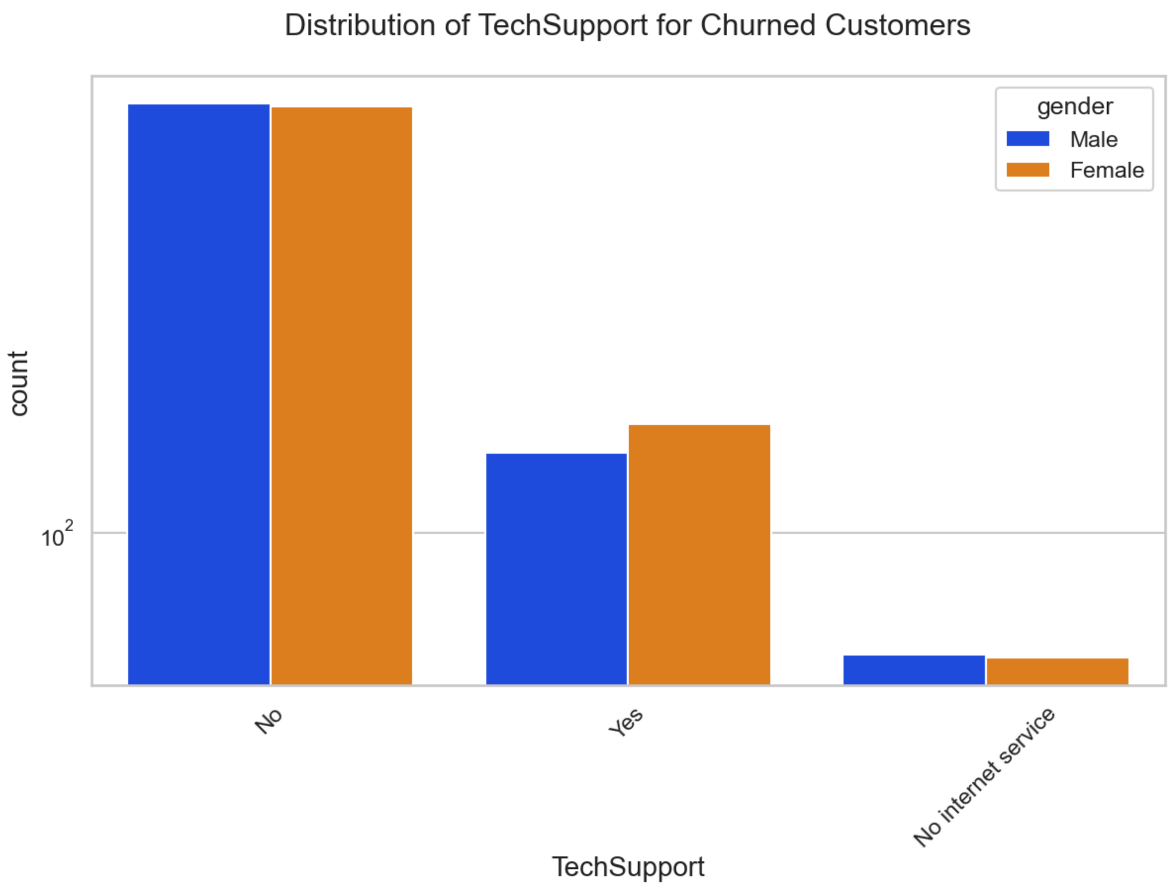
In [38]:
```
uniplot(new_df1_target1,col='PaymentMethod',title='Distribution of PaymentMethod
```



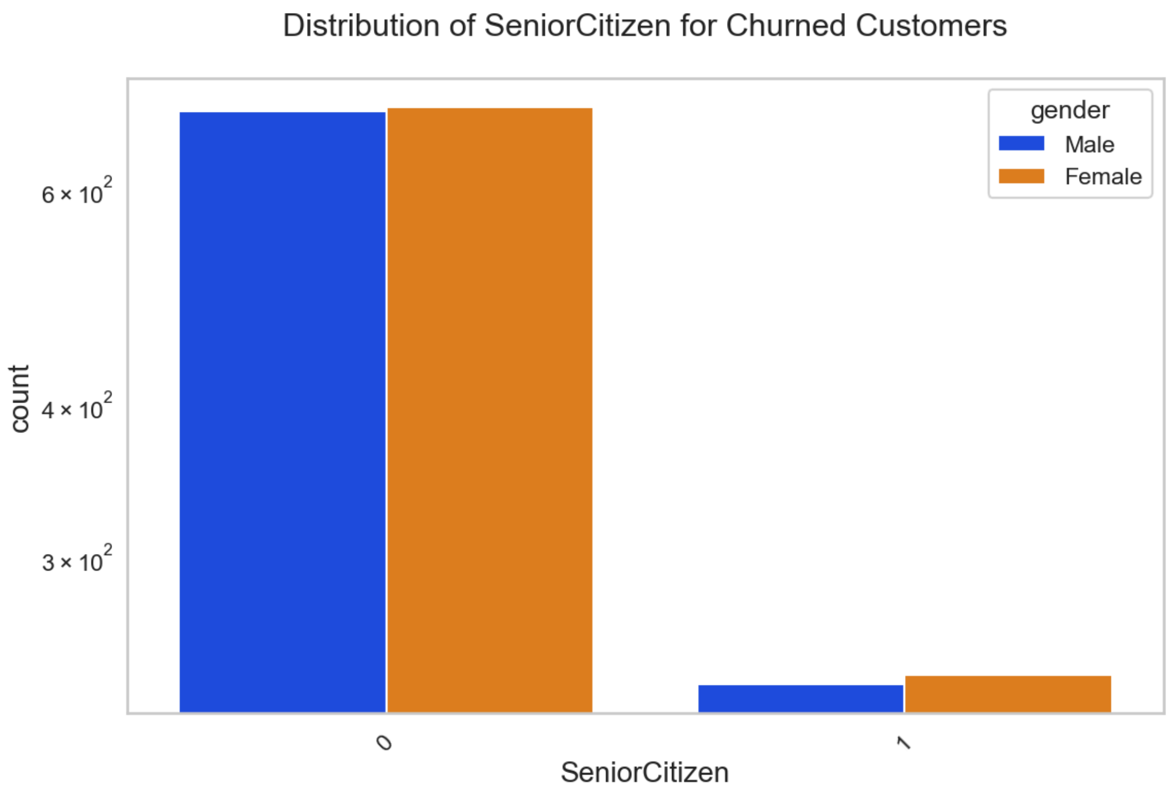
In [39]:
```
uniplot(new_df1_target1,col='Contract',title='Distribution of Contract for Churn
```



In [40]:
```
uniplot(new_df1_target1,col='TechSupport',title='Distribution of TechSupport for
```

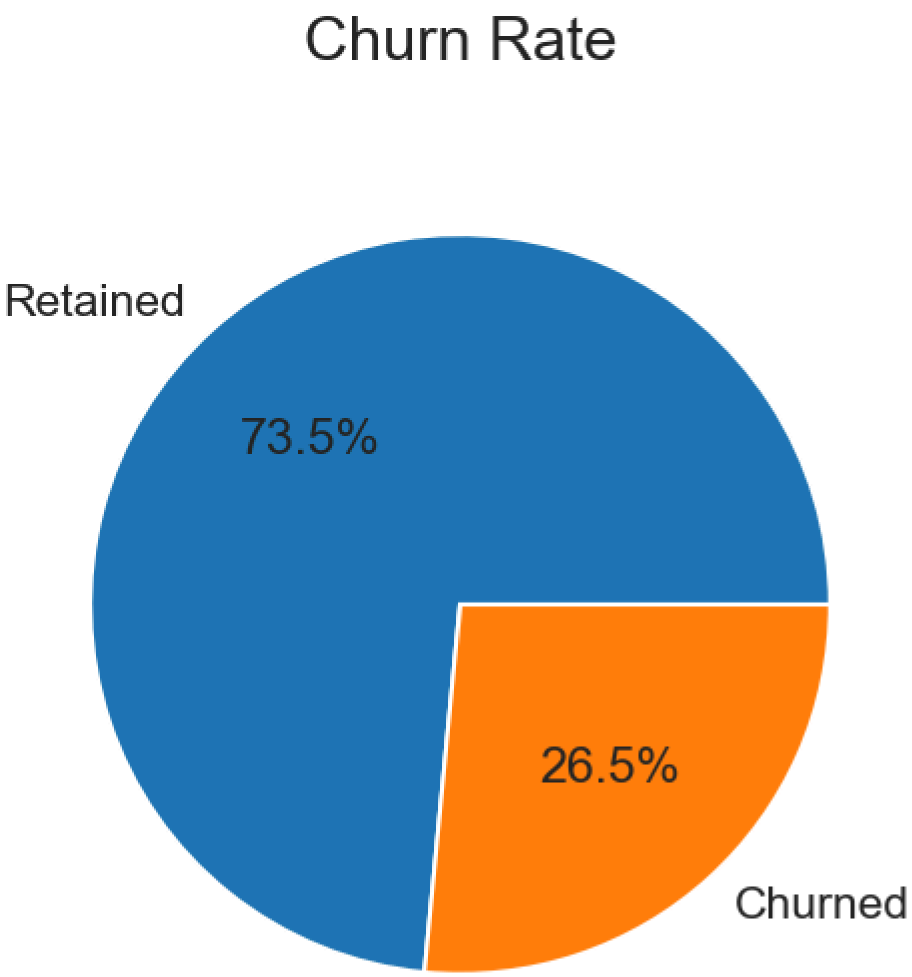


In [41]:
```
uniplot(new_df1_target1,col='SeniorCitizen',title='Distribution of SeniorCitizen
```



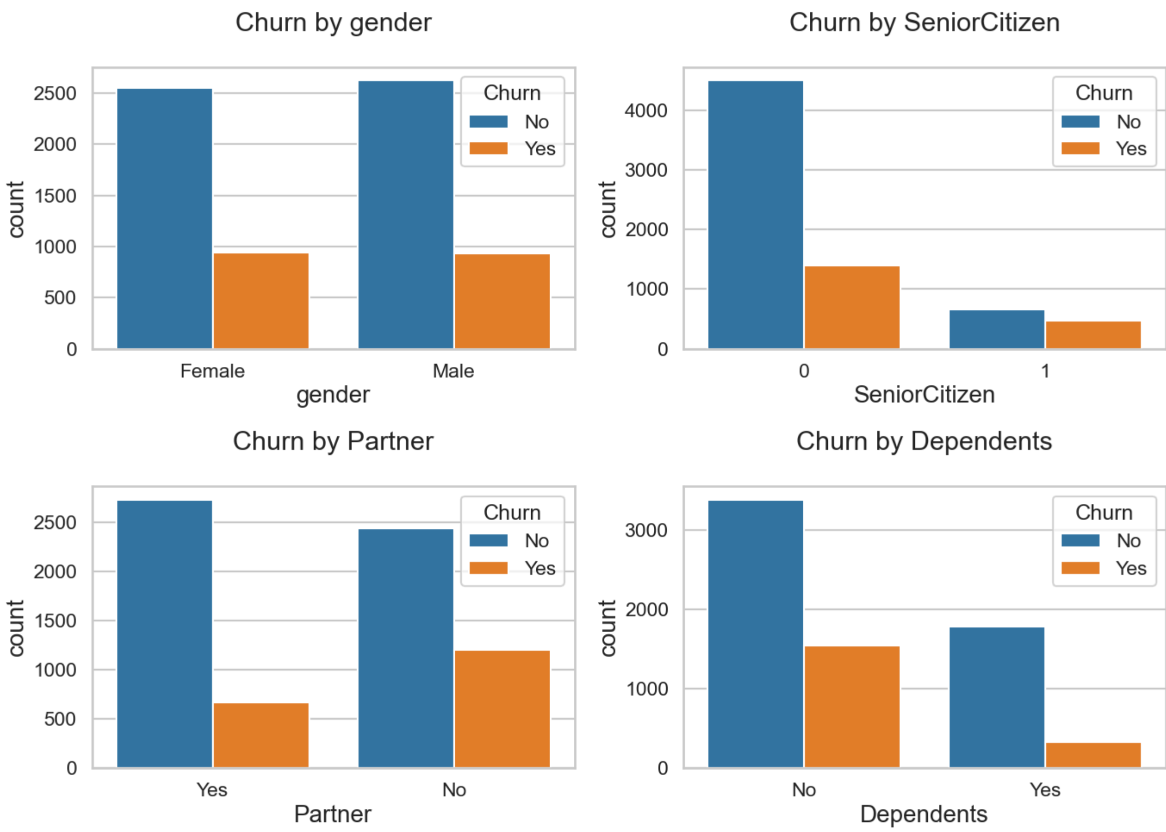
In [42]:
```
# 1. Churn Rate Analysis
churn_rate = telco_base_data['Churn'].value_counts(normalize=True) * 100
plt.figure(figsize=(6, 6))
churn_rate.plot(kind='pie', autopct='%1.1f%%', labels=['Retained', 'Churned'])
plt.title('Churn Rate')
```

```
plt.ylabel('')
plt.show()
```

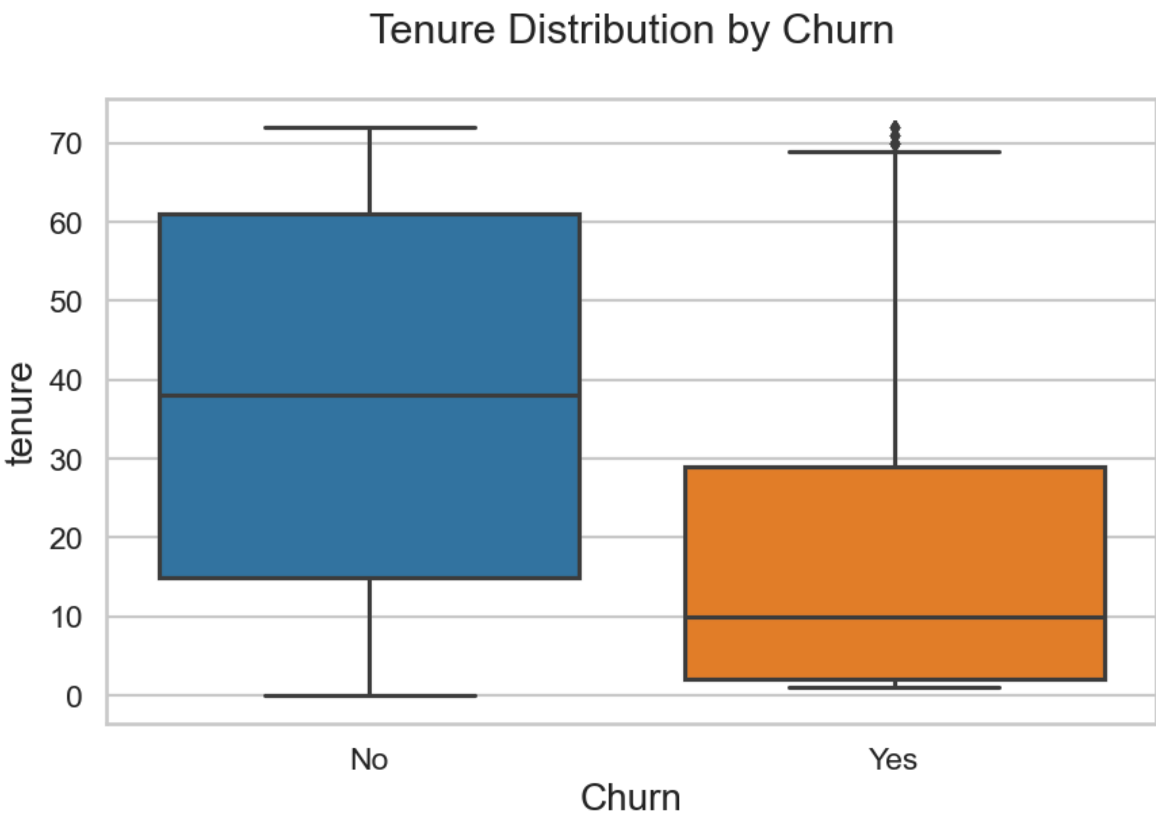


In [43]:

```
# 2. Demographic Analysis
demographics = ['gender', 'SeniorCitizen', 'Partner', 'Dependents']
plt.figure(figsize=(14, 10))
for i, column in enumerate(demographics, 1):
    plt.subplot(2, 2, i)
    sns.countplot(data=telco_base_data, x=column, hue='Churn')
    plt.title(f'Churn by {column}')
plt.tight_layout()
plt.show()
```

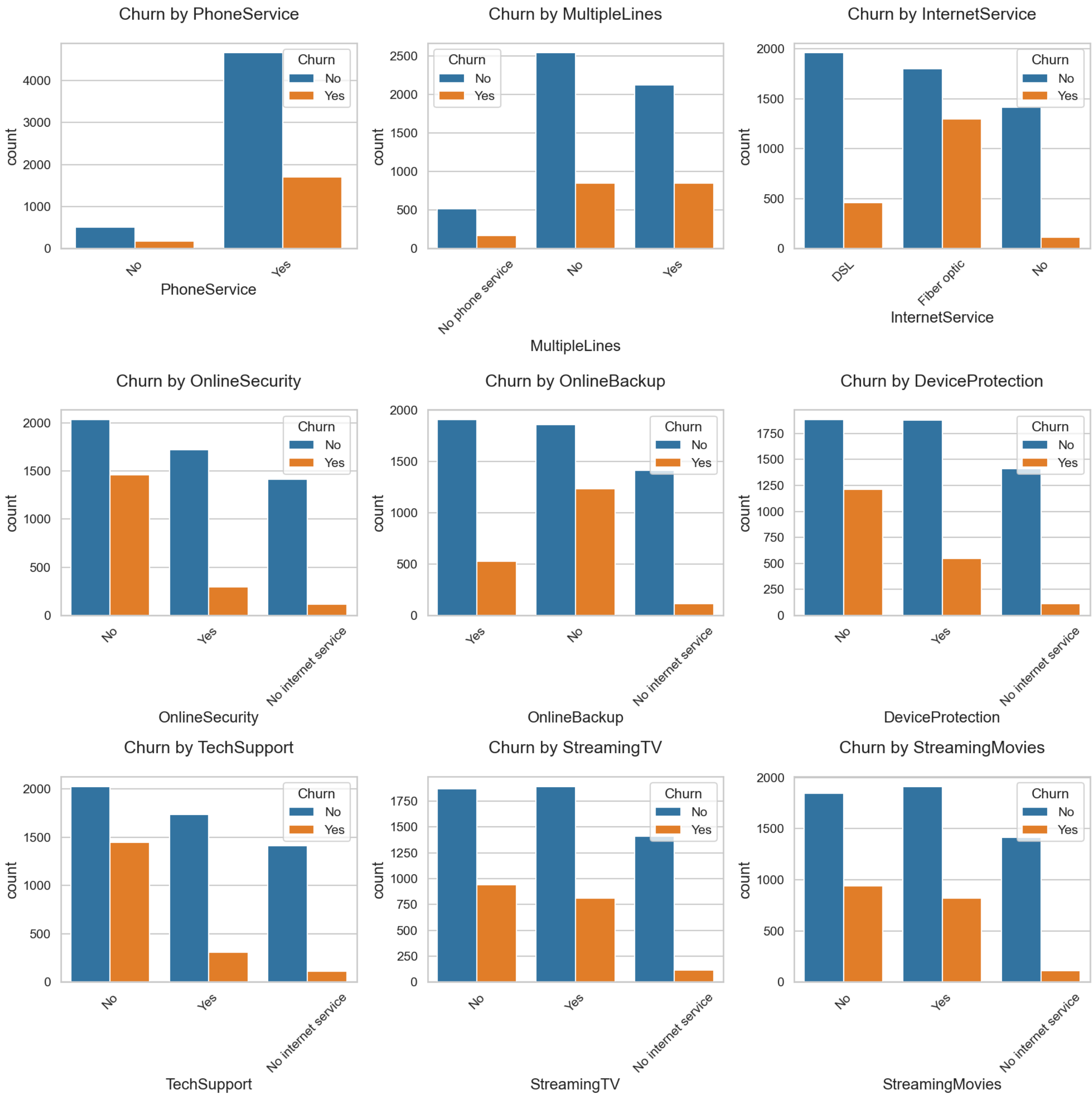


In [44]:
```
# 3. Tenure Analysis
plt.figure(figsize=(10, 6))
sns.boxplot(data=telco_base_data, x='Churn', y='tenure')
plt.title('Tenure Distribution by Churn')
plt.show()
```

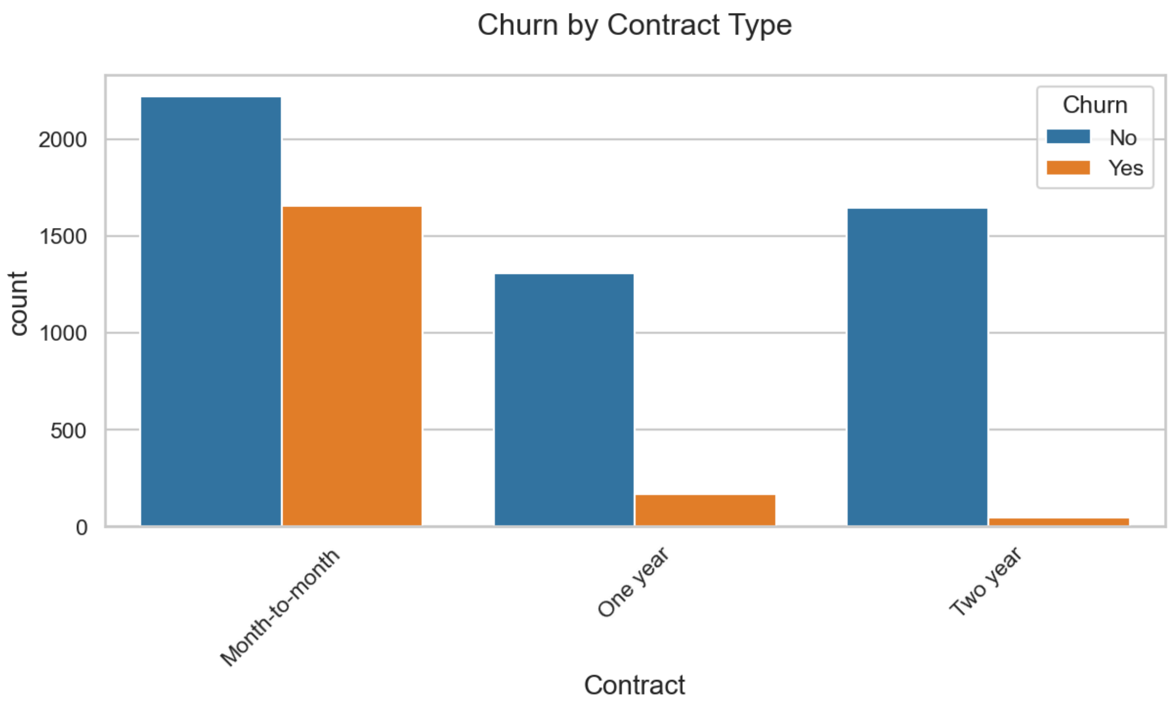


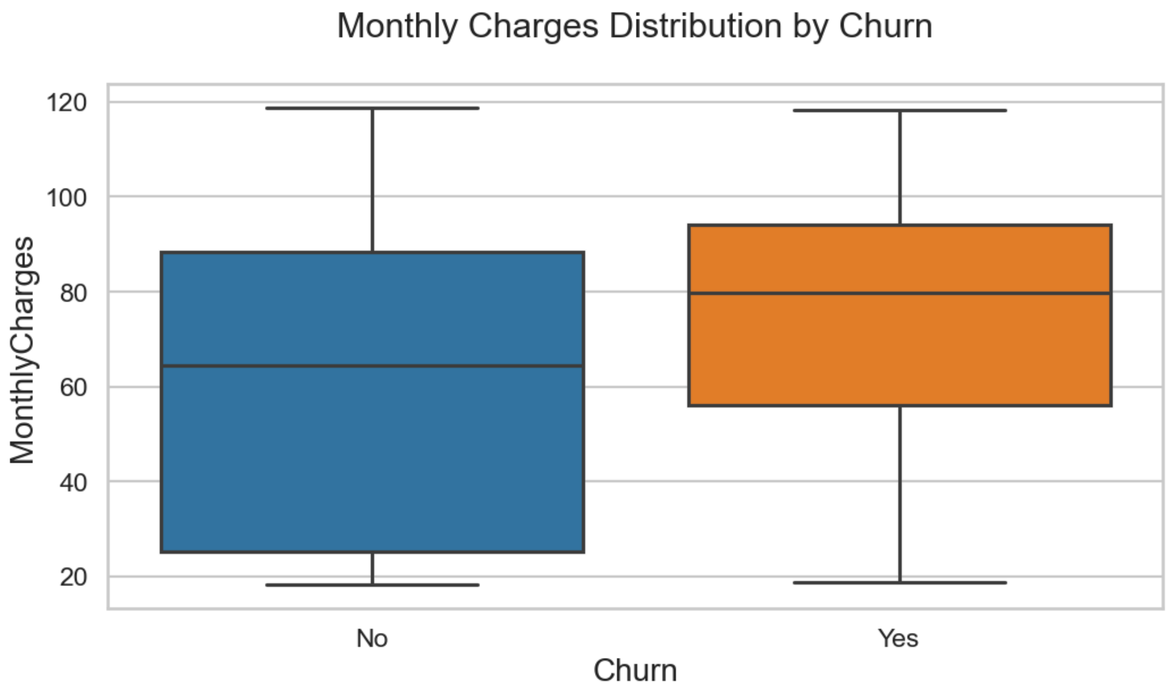
In [45]:
```
# 4. Service Usage Analysis
services = ['PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity'
            'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'S
```

```python
plt.figure(figsize=(20, 20))
for i, service in enumerate(services, 1):
    plt.subplot(3, 3, i)
    sns.countplot(data=telco_base_data, x=service, hue='Churn')
    plt.title(f'Churn by {service}')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [46]:
```python
# 5. Contract and Billing Analysis
plt.figure(figsize=(14, 6))
sns.countplot(data=telco_base_data, x='Contract', hue='Churn')
plt.title('Churn by Contract Type')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(data=telco_base_data, x='Churn', y='MonthlyCharges')
plt.title('Monthly Charges Distribution by Churn')
plt.show()
```





CONCLUSION

1. Electronic check medium are the highest churners
2. Contract Type - Monthly customers are more likely to churn because of no contract terms, as they are free to go customers.
3. No Online security, No Tech Support category are high churners
4. Non senior Citizens are high churners

In [47]:
```
telco_data_dummies.to_csv('tel_churn.csv')
```

In [55]:
```
import pandas as pd
import sqlalchemy as sa
```

In [58]:
```
from sqlalchemy import create_engine

# Define the database connection URL
# Replace 'username', 'password', 'host', and 'database_name' with your MySQL se
db_url = 'mysql+pymysql://root:2002@localhost/Customer_churn'

# Create an engine
engine = create_engine(db_url)

# Test the connection
try:
    engine.connect()
    print("Connection successful!")
except Exception as e:
    print("Connection failed:", e)
```

```
Connection successful!
```

In [63]:
```
# Write DataFrame to SQL table
try:
    telco_base_data.to_sql(name='telco_customer_churn', con=engine, if_exists='f
    print("DataFrame successfully written to SQL table.")
except Exception as e:
    print("DataFrame to SQL table failed:", e)
```

```
DataFrame successfully written to SQL table.
```

In [ ]: