Title: Predicting Customer Churn in a Telecommunications Company Objective: Develop a predictive model to accurately identify telecom customers who are likely to churn, enabling the company to take proactive measures to retain them.

Business Context: Customer churn is a significant issue for telecommunications companies, leading to substantial revenue loss. Understanding and predicting customer churn is critical for developing effective retention strategies. By analyzing customer data, we aim to identify the key factors contributing to churn and build a model that can predict at-risk customers.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# importing the data.
data=pd.read_csv(r"C:\Users\medam\Downloads\archive (8)\WA_Fn-UseC_-Telco-Customer-Churn
# Head gives the top 5 records.
data.head()
```

Out[2]:

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

5 rows × 21 columns

In [3]:
```
# checking the dimensions of the data.
data.shape
```

Out[3]:
```
(7043, 21)
```

In [4]:
```
# checking the types of columns present in the data.
data.columns
```

Out[4]:
```
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 [5]:
```
# checking the data type of each column.
data.dtypes
```

Loading [MathJax]/extensions/Safe.js

Out[5]:
```
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 [6]:
```
# checking the descriptive statistics of numerical variables.
data.describe()
```

Out[6]:

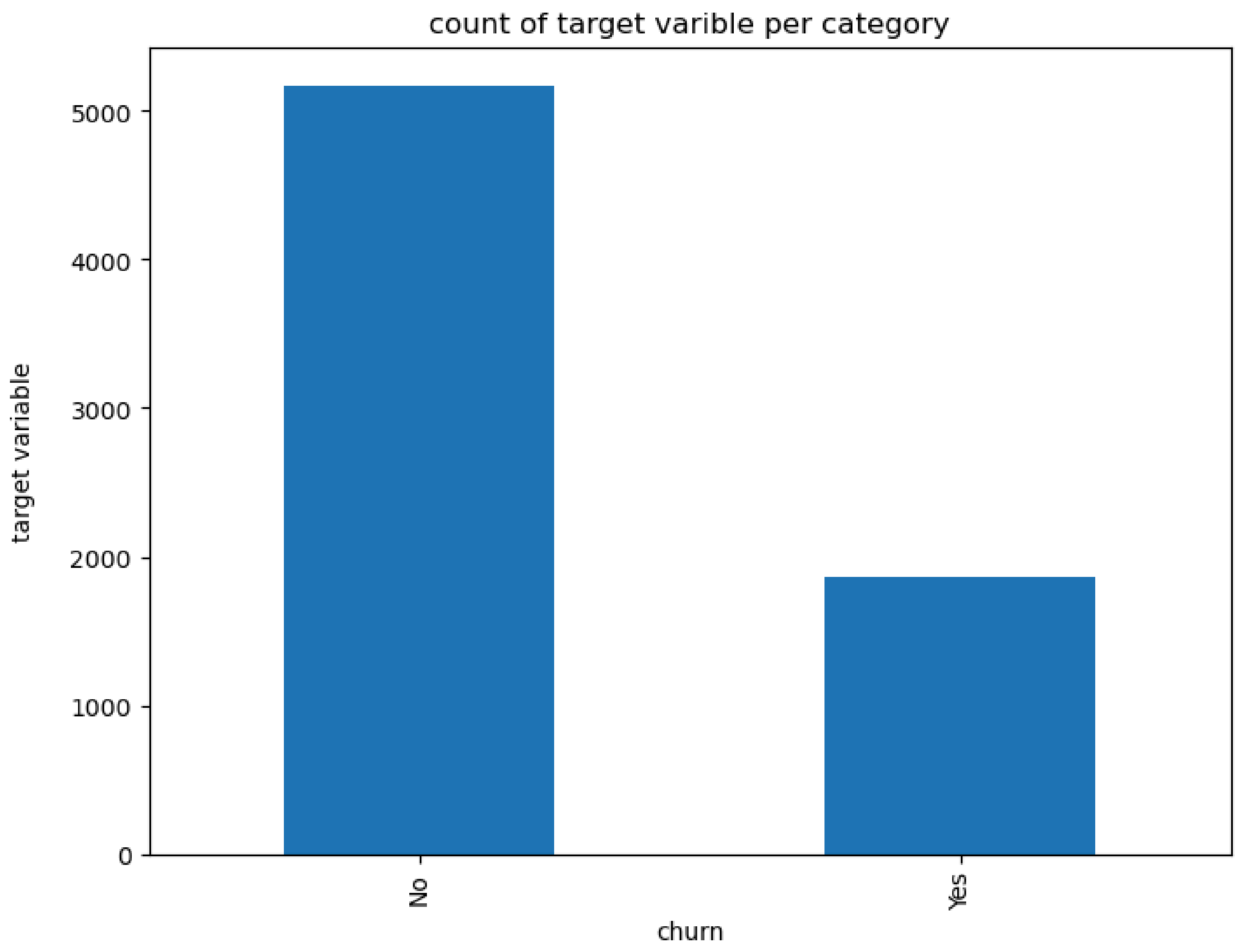
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

- senior citizen supposed to be a categorical variable that is why the 25%-50%-75% distribution is not proper.
- 75% customers have tennure less than 55 months.
- The average monthly charges are USD65 but the customers are paying USD89.

In [7]:
```
data["Churn"].value_counts().plot(kind="bar",figsize=(8,6))
plt.xlabel("churn")
plt.ylabel("target variable",labelpad=14)
plt.title("count of target varible per category")
```

Out[7]:
```
Text(0.5, 1.0, 'count of target varible per category')
```

Loading [MathJax]/extensions/Safe.js



```
In [8]: data["Churn"].value_counts()
```

```
Out[8]: No     5174
        Yes    1869
        Name: Churn, dtype: int64
```

```
In [9]: # checking the percentage of distribution in churn.
        data["Churn"].value_counts(normalize=True)*100
```

```
Out[9]: No     73.463013
        Yes    26.536987
        Name: Churn, dtype: float64
```

- Here we can see the data is imbalanced(73:27) ratio.
- so we analyze the data with other features while taking the target values separately to get some insights.

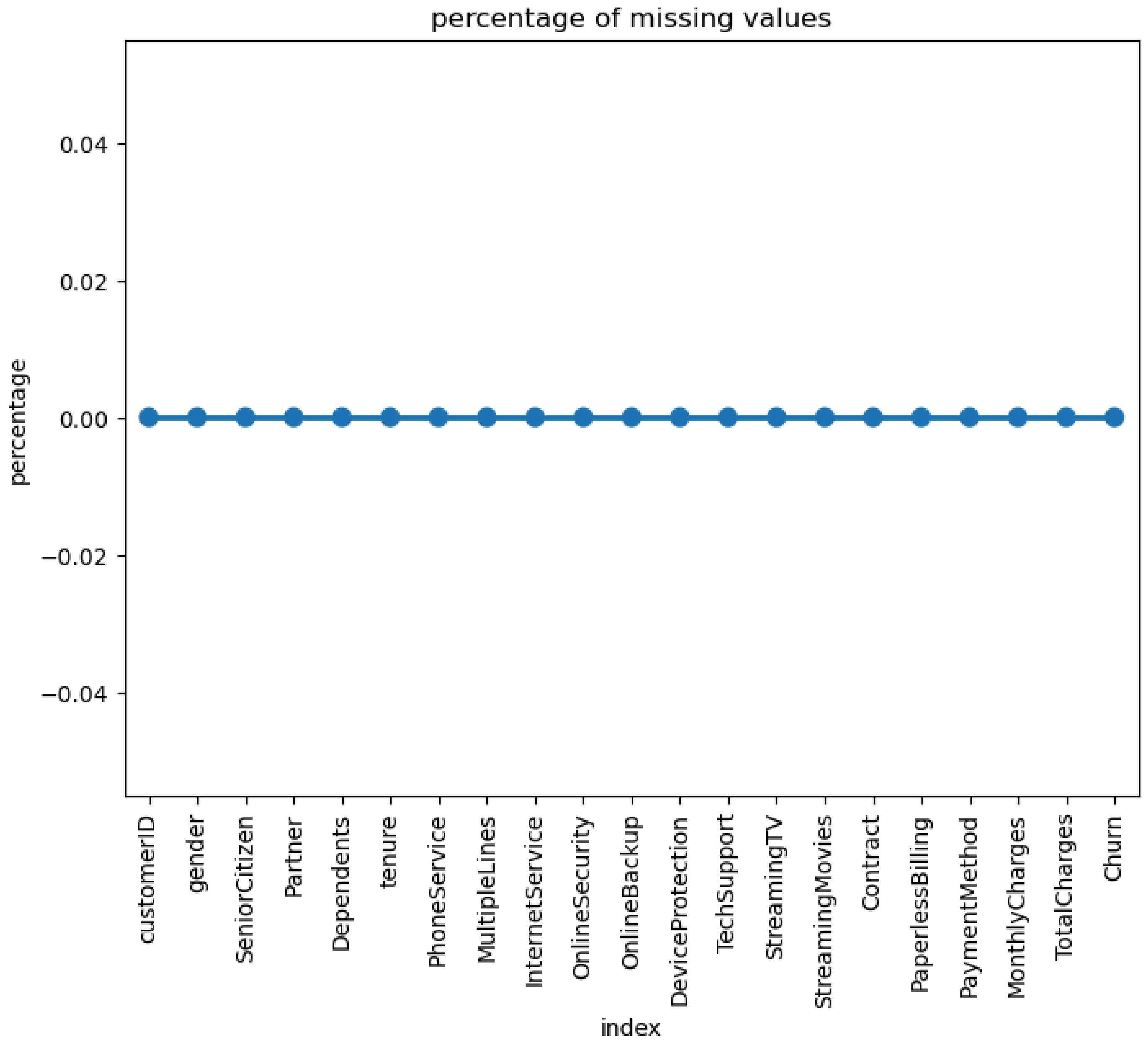
```
In [10]: # Here we are using verbose is True because we have many columns.
         data.info(verbose=True)
```

```
<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 [11]:

```
# To identify missing the percentage of values.
missing=pd.DataFrame((data.isnull().sum())*100/data.shape[0]).reset_index()
plt.figure(figsize=(8,6),dpi=100)
ax=sns.pointplot(x="index",y=0,data=missing)
plt.xticks(rotation=90)
plt.ylabel("percentage")
plt.title("percentage of missing values")
plt.show()
```

Loading [MathJax]/extensions/Safe.js



Missing data- initial intuition.

- Here we can see there is no missing data in this dataset. ### General thumb rules.
- If the variable has lower no.of missing values then we can use mean/median/mode(it depends on type of the variable).If the variable is numerical we can use mean/median whereas if the data is categorical we can use mode.
- If the variable has higher no.of missing values(60-70%) then undoubtedly we can drop that varible.

Data cleaning.

- creating a copy of base data for manipulation and processing.

In [12]:
```
data=data.copy()
```

- Total charges supposed to be numerical.so we will convert that into numerical datatype.

Loading [MathJax]/extensions/Safe.js

In [13]:
```python
data.TotalCharges=pd.to_numeric(data.TotalCharges,errors="coerce")
data.isnull().sum()
```

Out[13]:
```
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
```

- we can see some of the missing values in total charges.so,let's see.

In [14]:
```python
data.loc[data["TotalCharges"].isnull()==True]
```

Out[14]:

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

11 rows × 21 columns

Missing value treatment.

- since, the missing records are very low compared to total dataset is very low.so,it is safe to ignorr them from further processing.

In [15]:
```
data.dropna(how="any",inplace=True)
```

In [16]:
```
data.shape
```

Out[16]:
```
(7032, 21)
```

- Divide customers into bins based on tennure.

In [17]:
```
data["tenure"].max()
```

Out[17]:
```
72
```

In [18]:
```
data["tenure"].min()
```

Out[18]:
```
1
```

In [19]:
```
# Group the tennure in bins of 12 months.
labels=["{0} - {1}".format(i,i+11) for i in range(1,72,12)]
data["tenure_group"]=pd.cut(data["tenure"],range(1,80,12),right=False,labels=labels)
data["tenure_group"].value_counts()
```

Out[19]:
```
1 - 12     2175
61 - 72    1407
13 - 24    1024
25 - 36     832
49 - 60     832
37 - 48     762
Name: tenure_group, dtype: int64
```

- Remove the columns which are not required.

In [20]:
```
data.drop(columns=["customerID","tenure"],inplace=True)
```

In [21]:
```
data.head()
```

Out[21]:

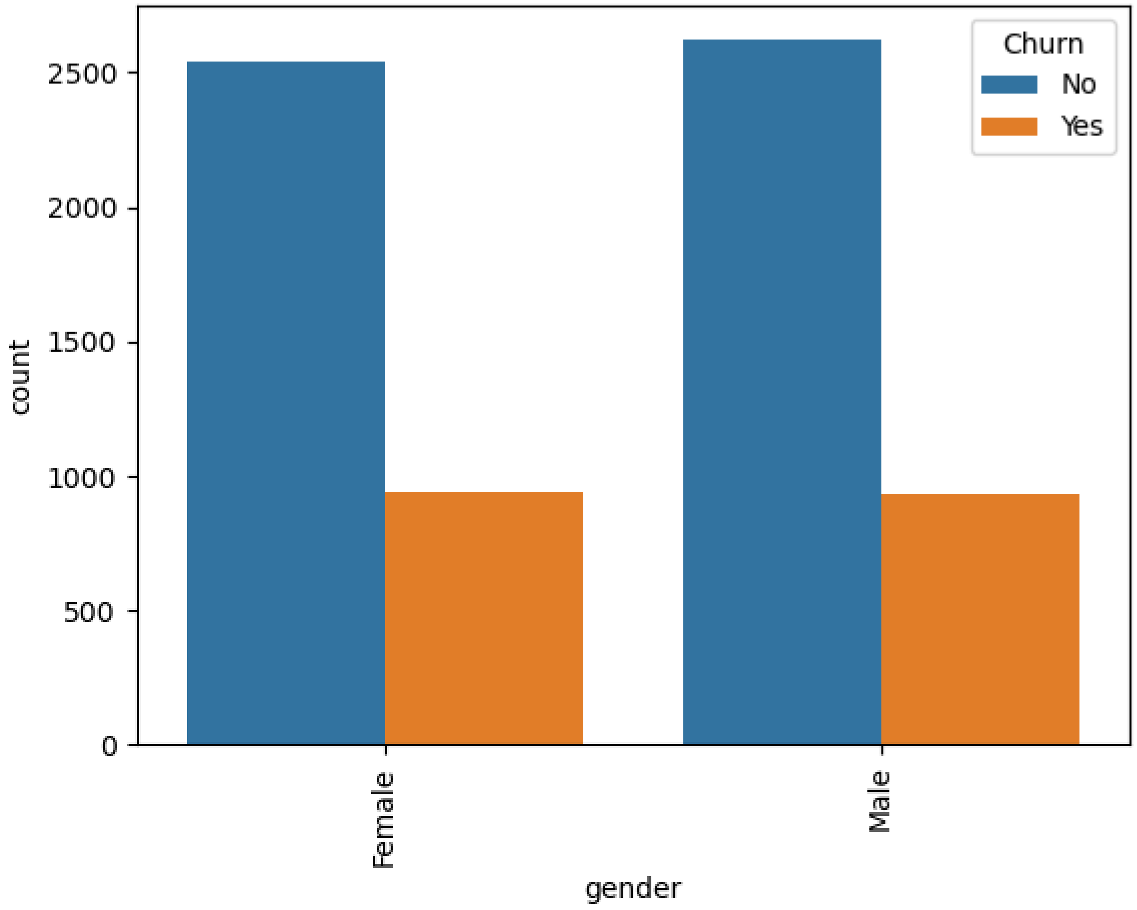
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	Onl
0	Female	0	Yes	No	No	No phone service	DSL	No	
1	Male	0	No	No	Yes	No	DSL	Yes	
2	Male	0	No	No	Yes	No	DSL	Yes	
3	Male	0	No	No	No	No phone service	DSL	Yes	
4	Female	0	No	No	Yes	No	Fiber optic	No	

Data Exploration.

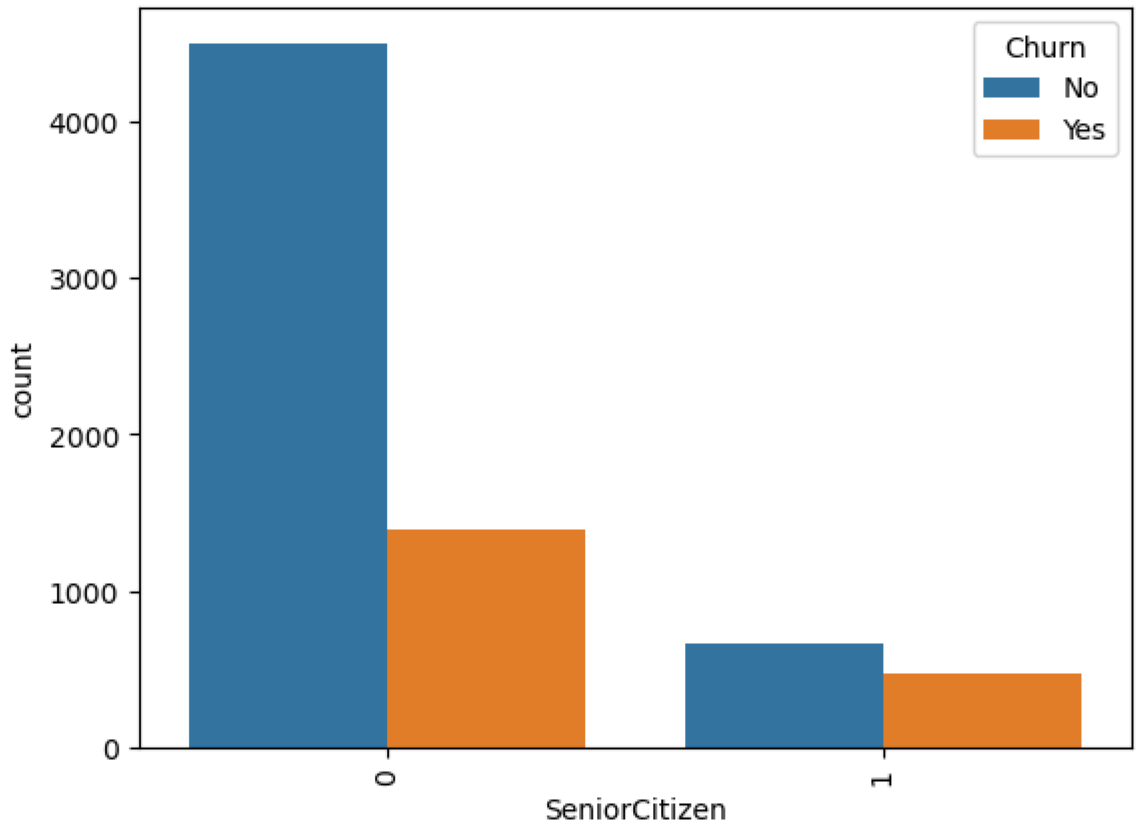
- plot distribution of individual predictors by churn.

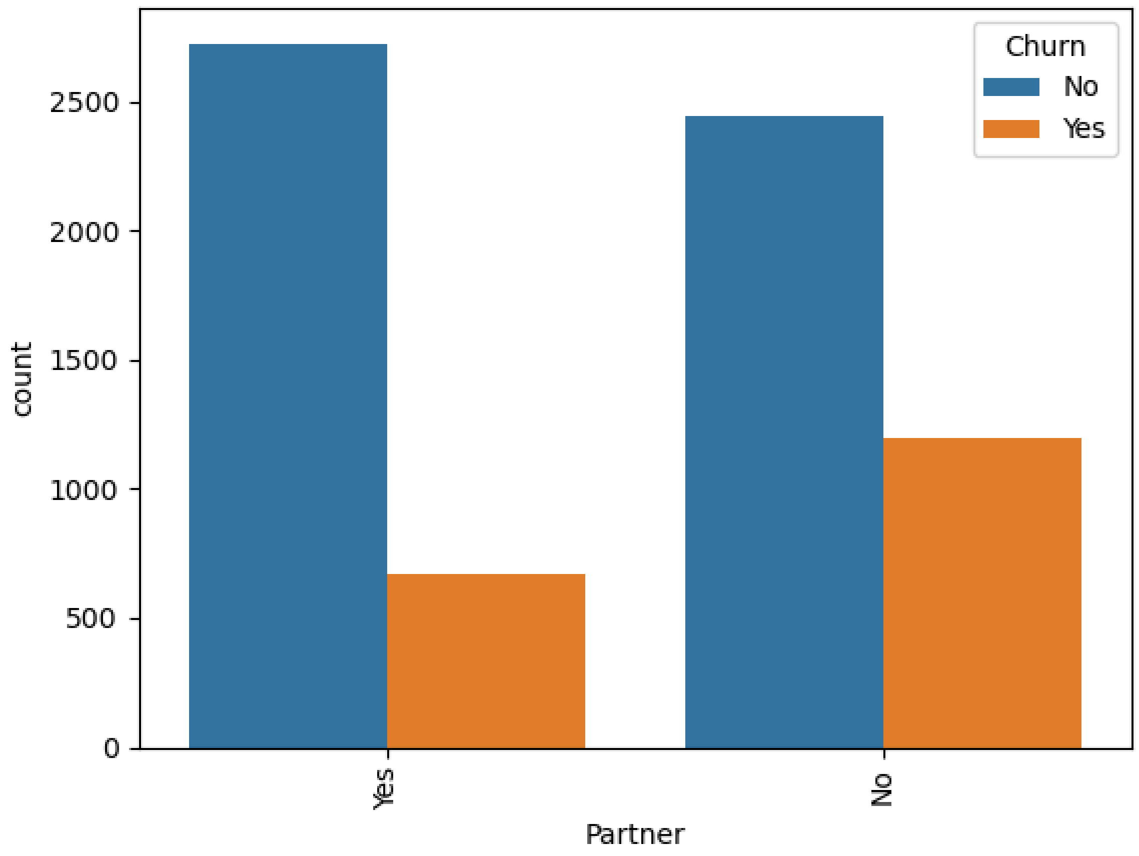
Univariate Analysis.

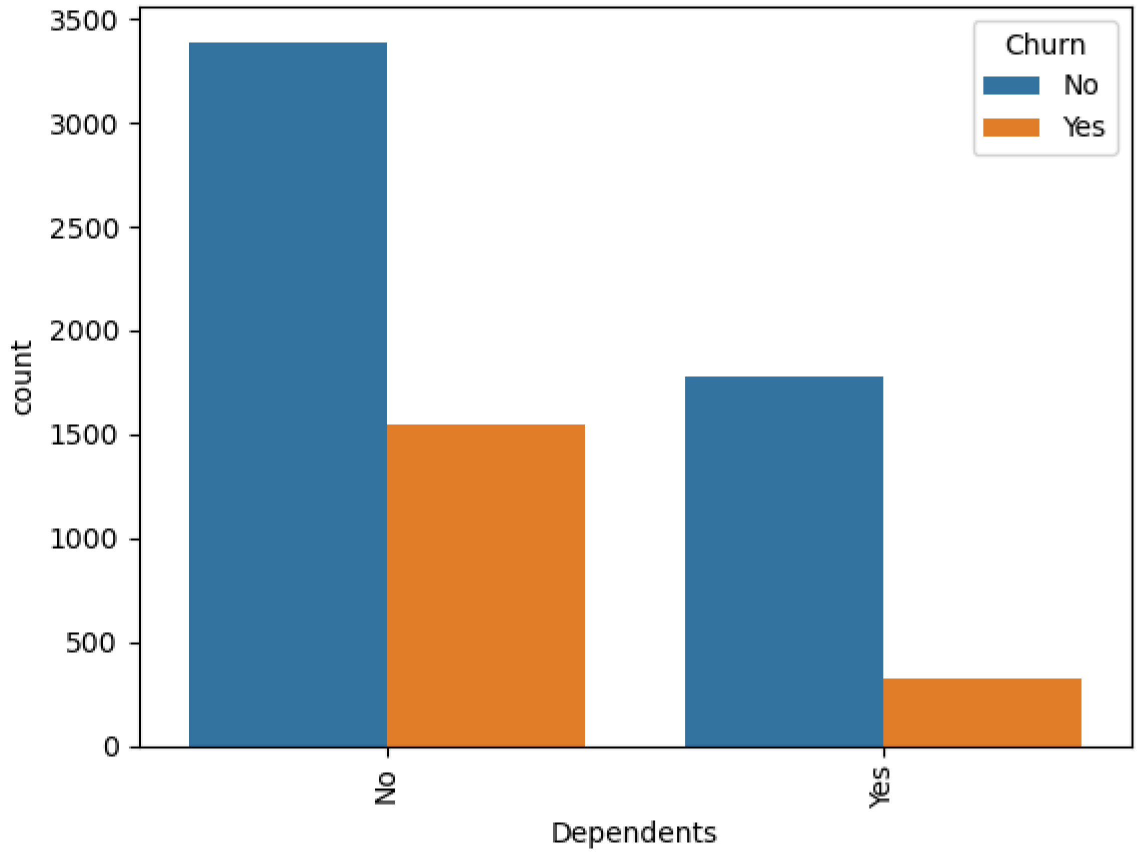
```
In [22]: for i,predictor in enumerate(data.drop(columns=["MonthlyCharges","TotalCharges","Churn"]
             plt.figure(i)
             sns.countplot(data=data,x=predictor,hue="Churn")
             plt.xticks(rotation=90)
```



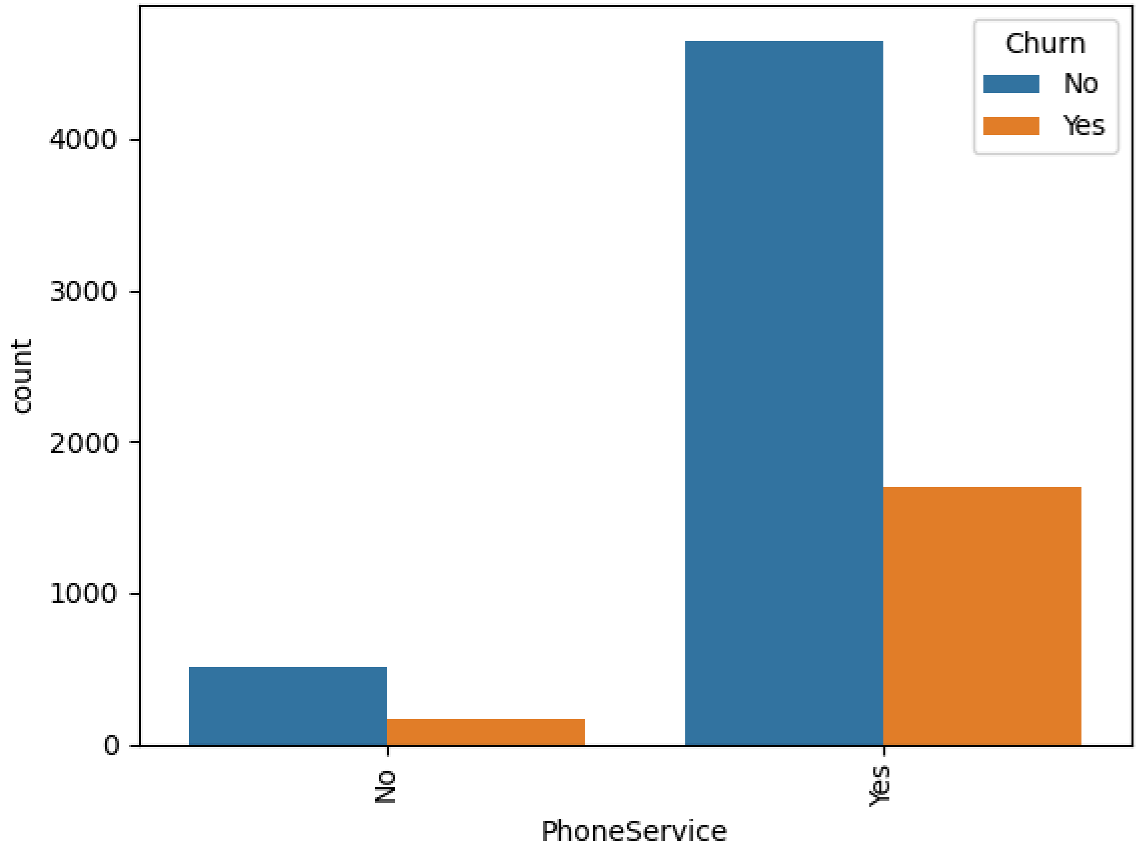
Loading [MathJax]/extensions/Safe.js


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

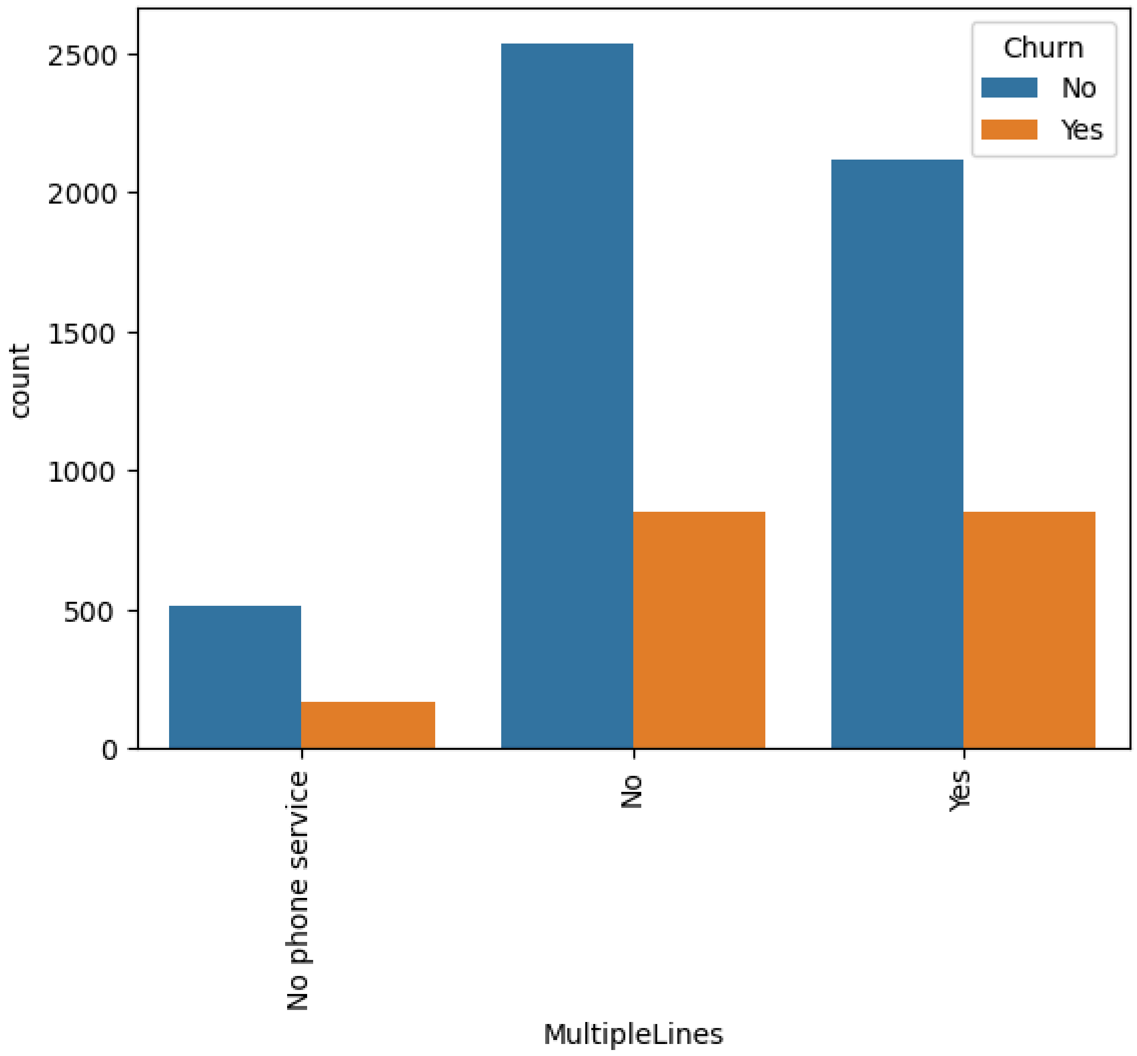

Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
Partner

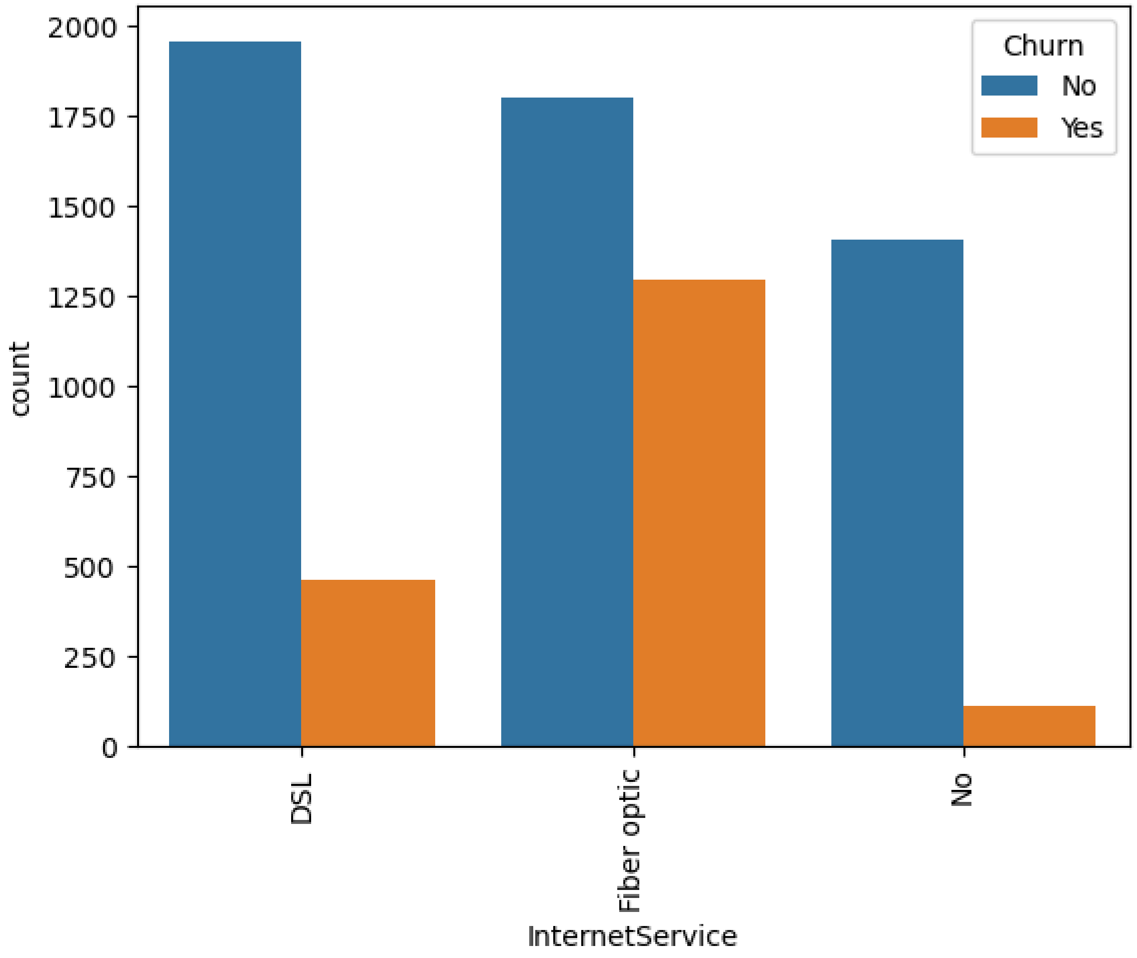


Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents

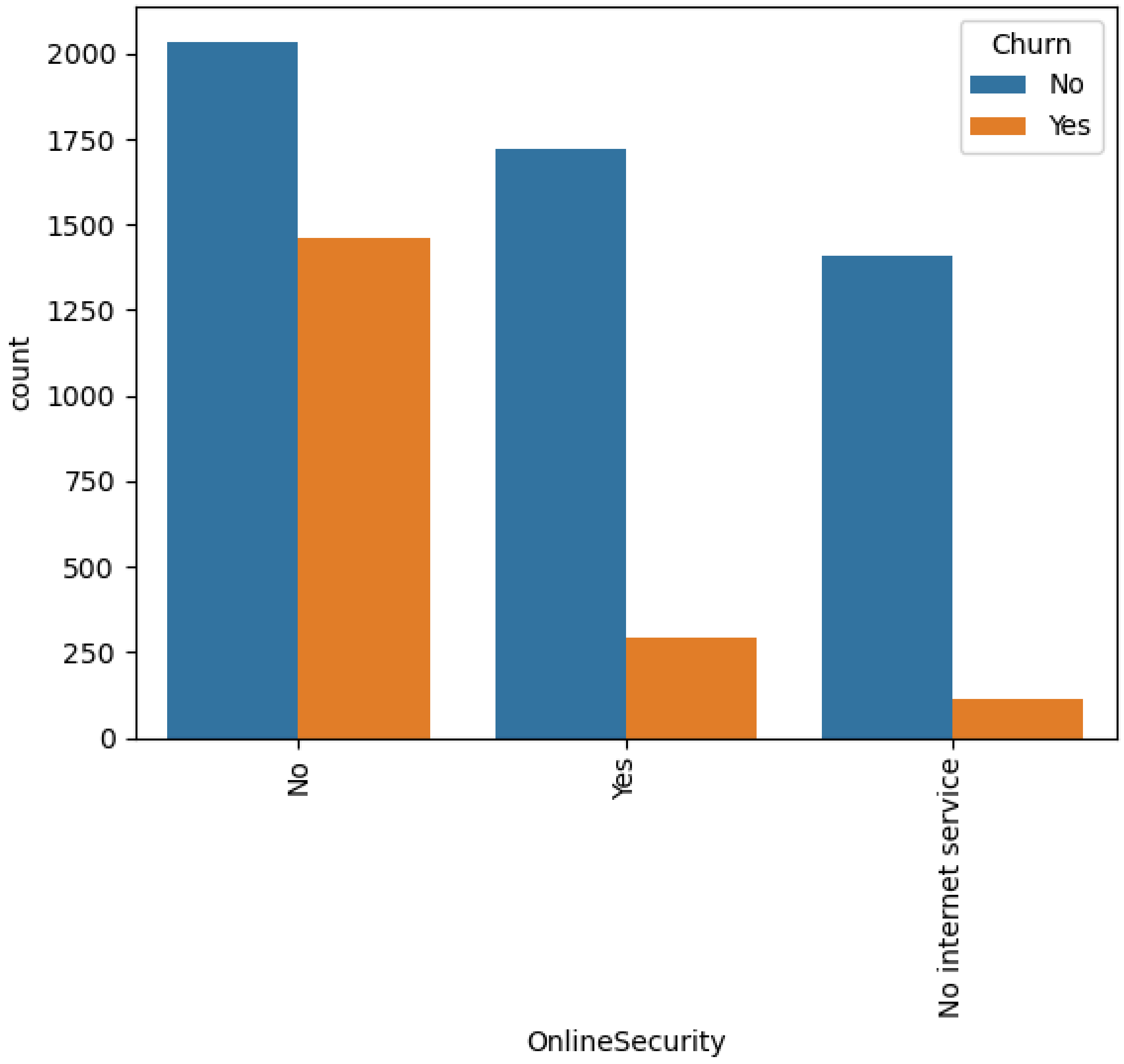


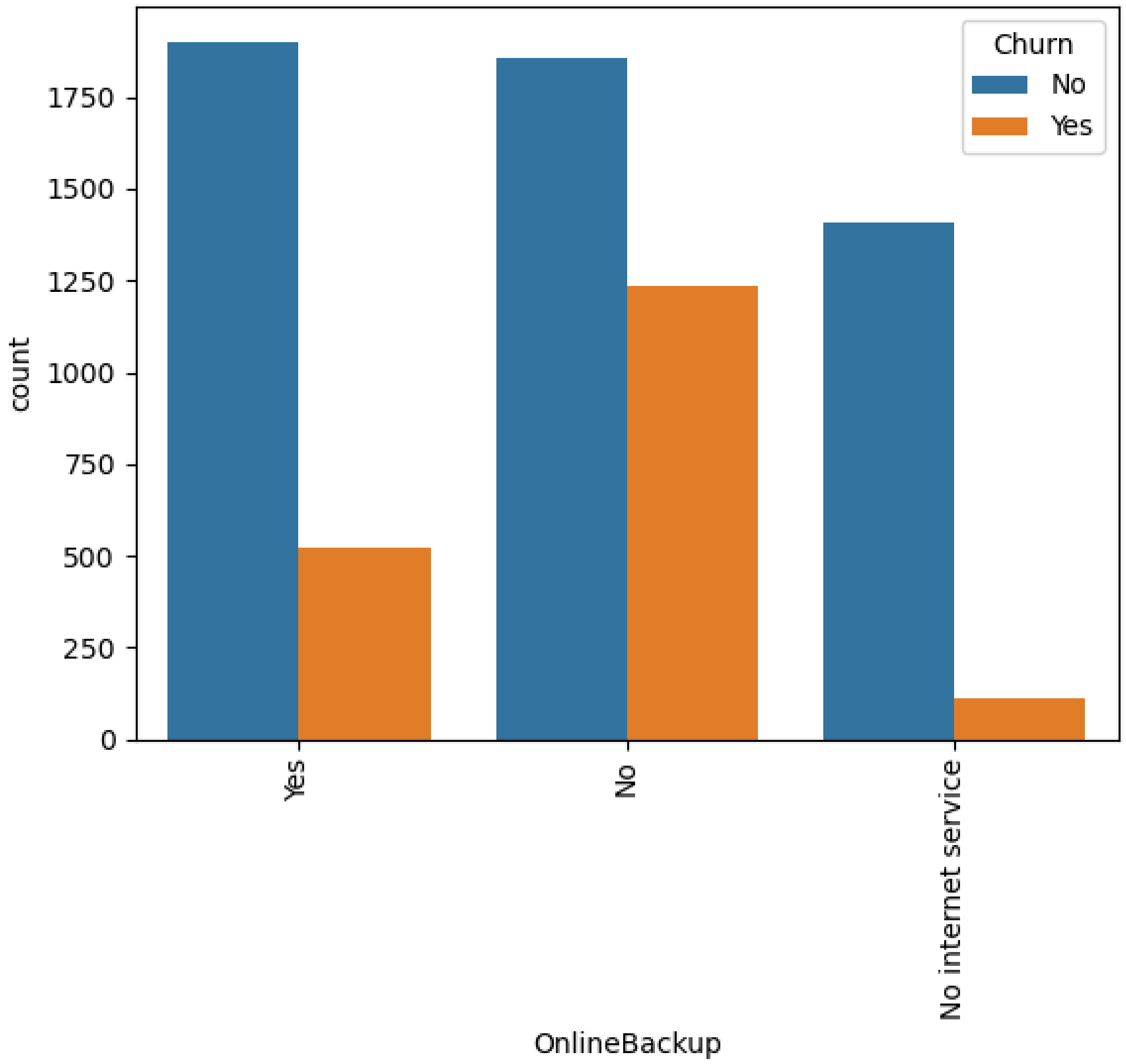
Churn
No
Yes
count
4000
3000
2000
1000
0
No
Yes
PhoneService



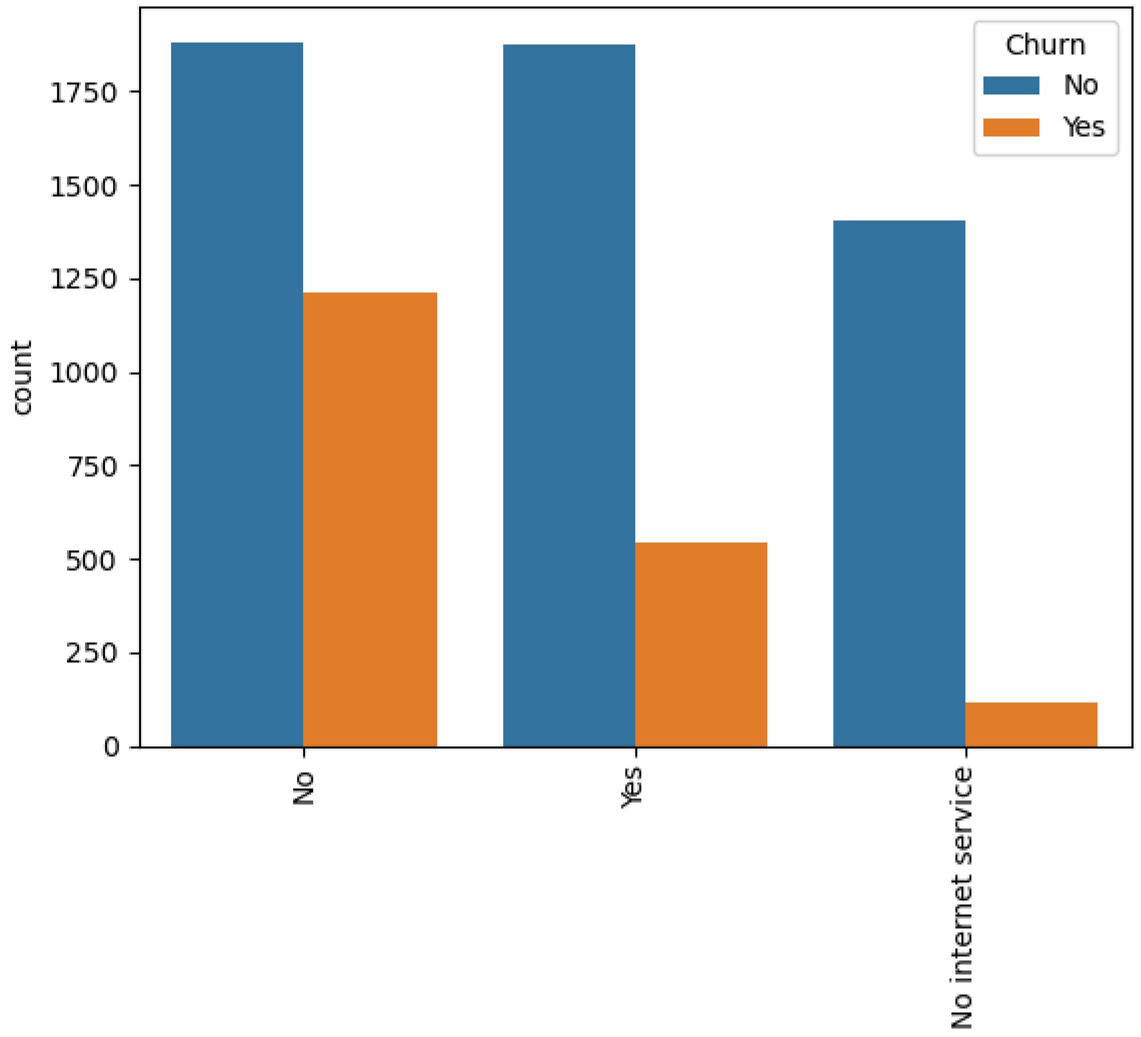


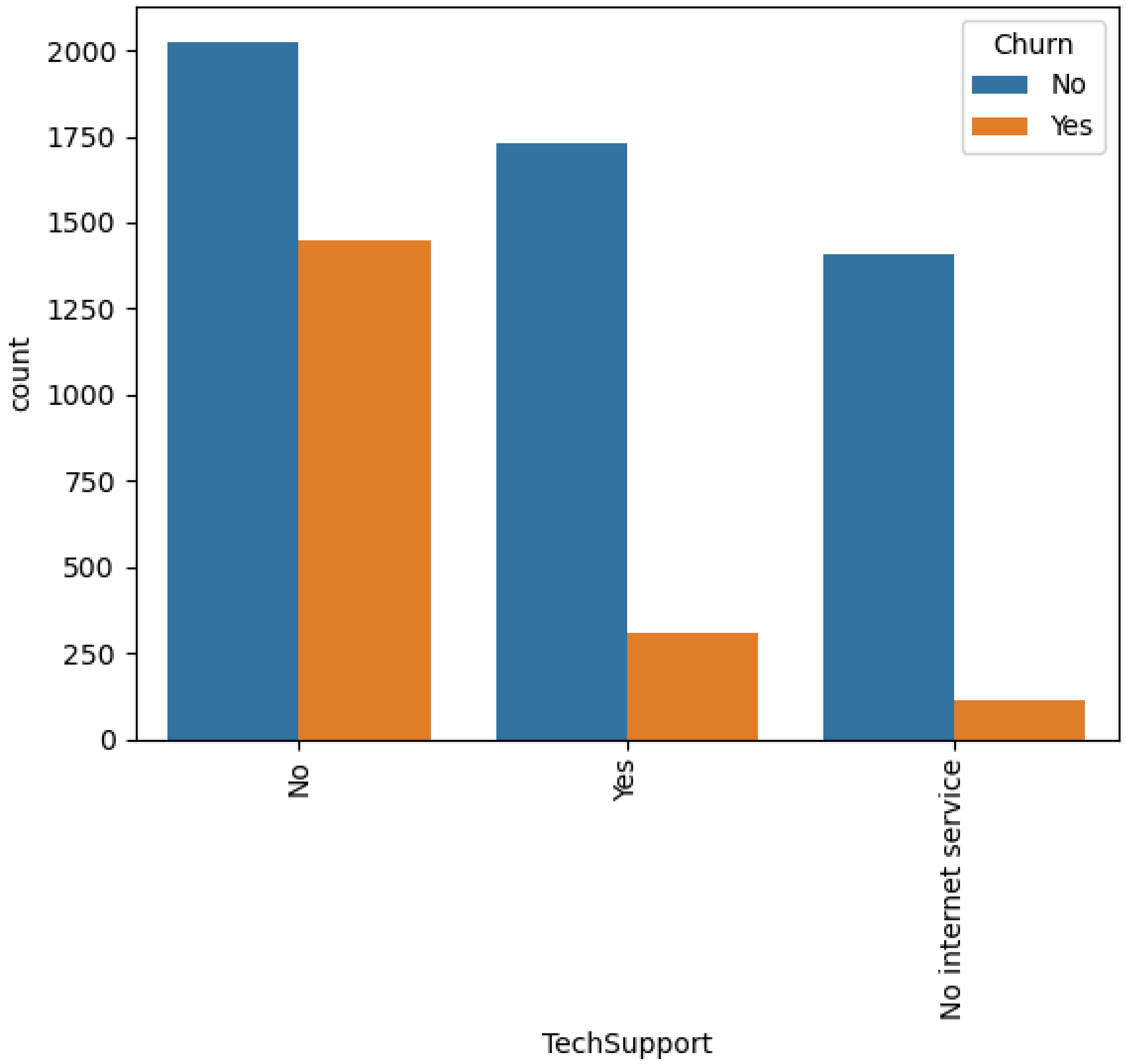
Loading [MathJax]/extensions/Safe.js


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

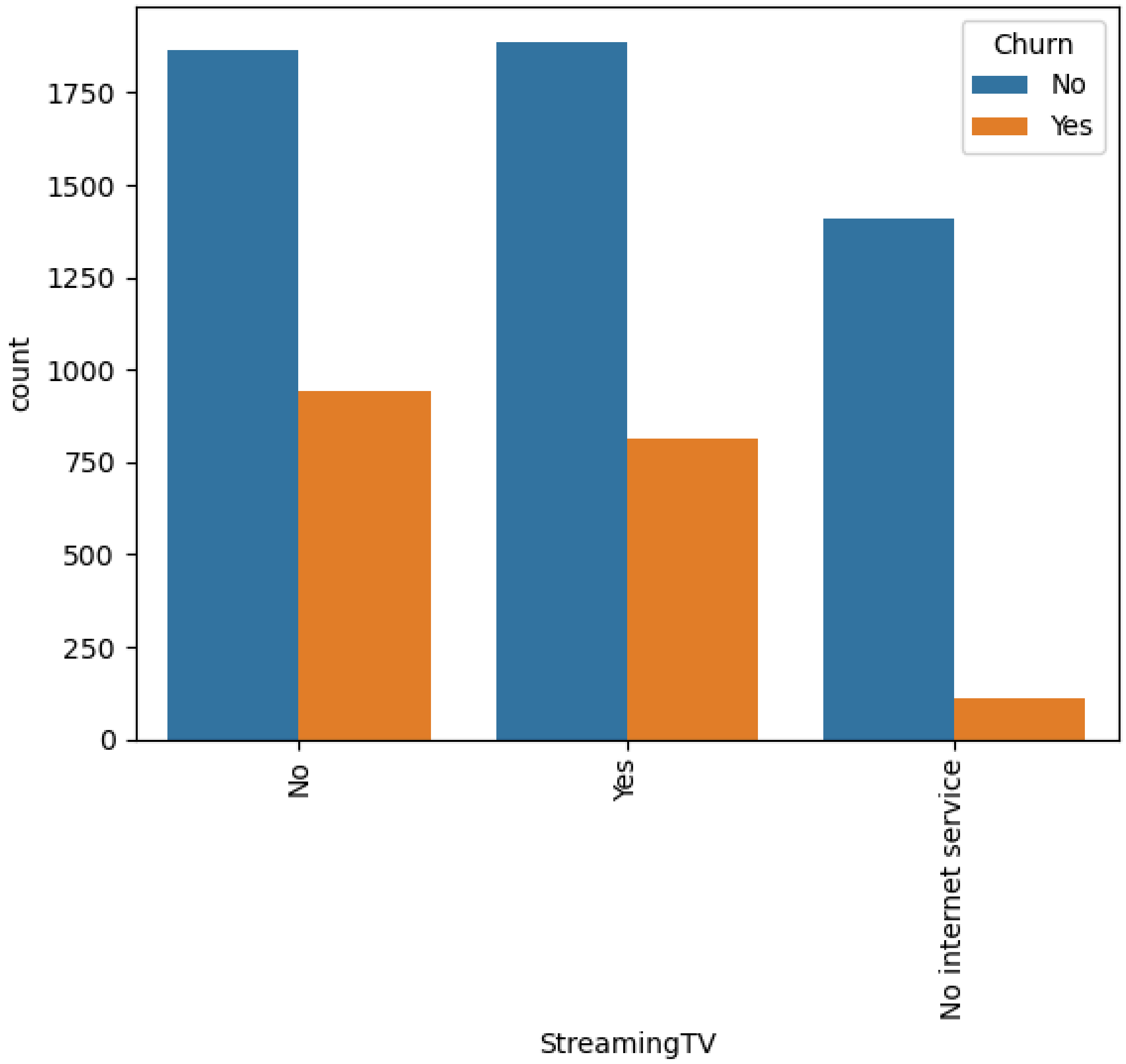


Loading [MathJax]/extensions/Safe.js

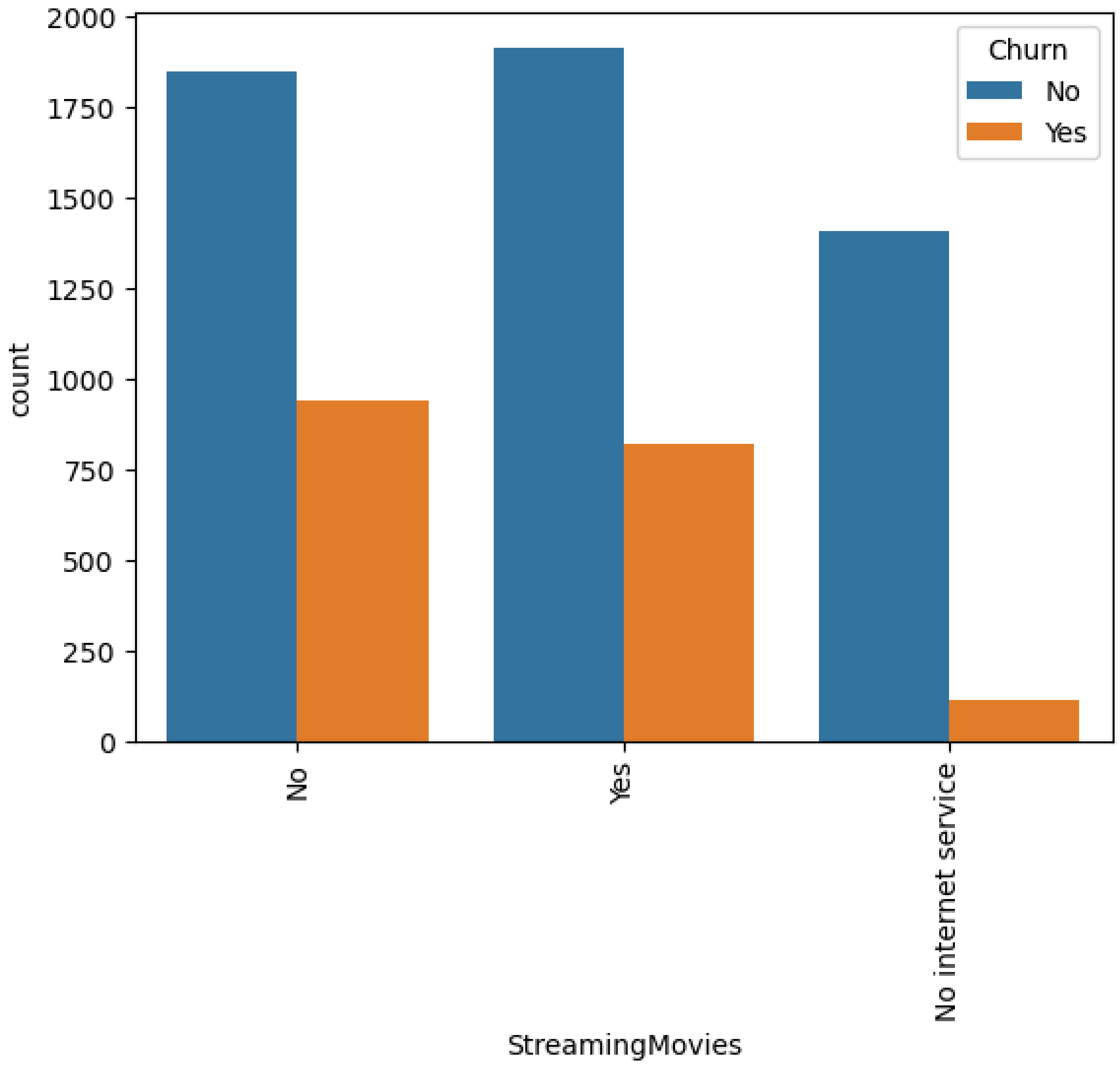

Churn
No
Yes
count
0
250
500
750
1000
1250
1500
1750
No
Yes
No internet service
DeviceProtection



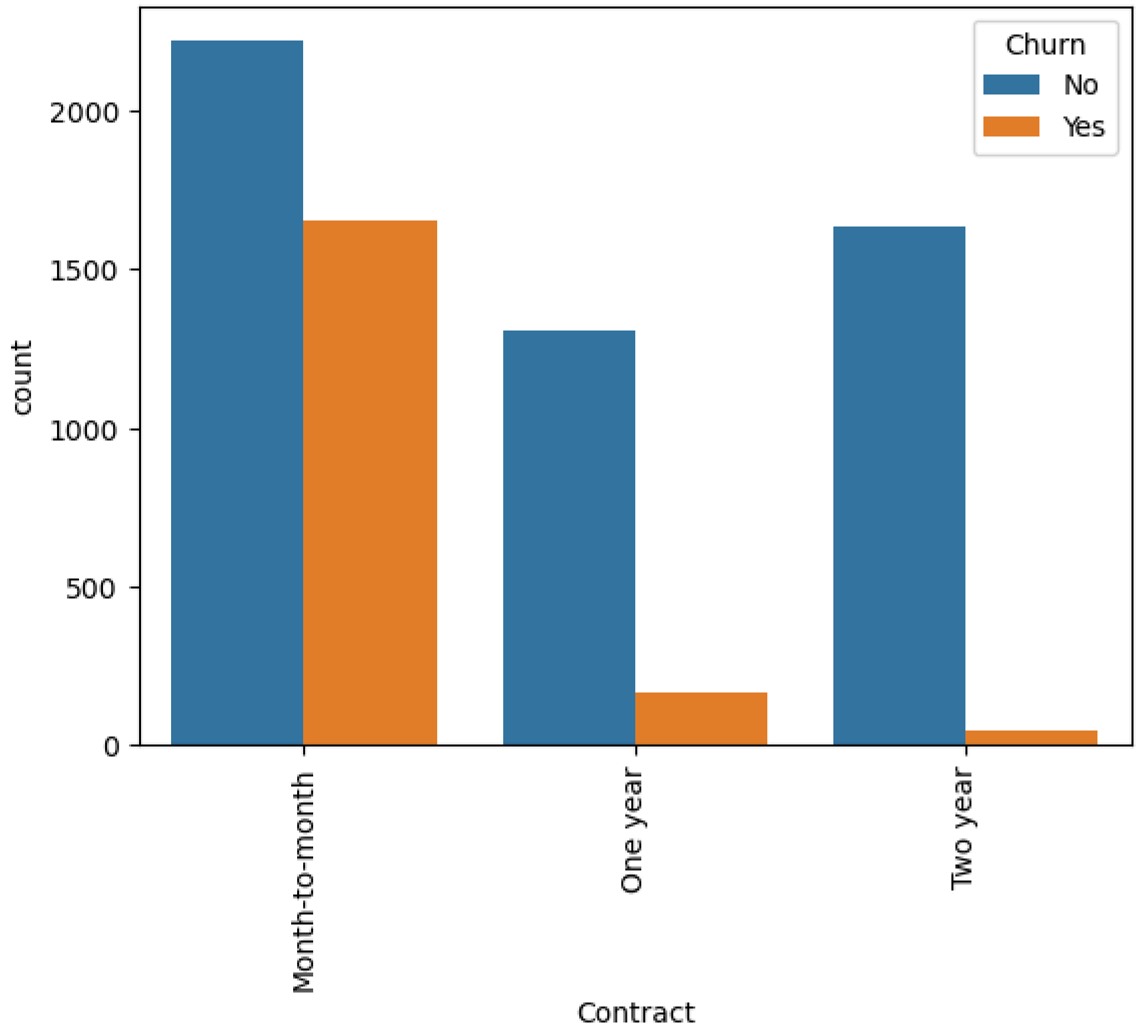
Loading [MathJax]/extensions/Safe.js

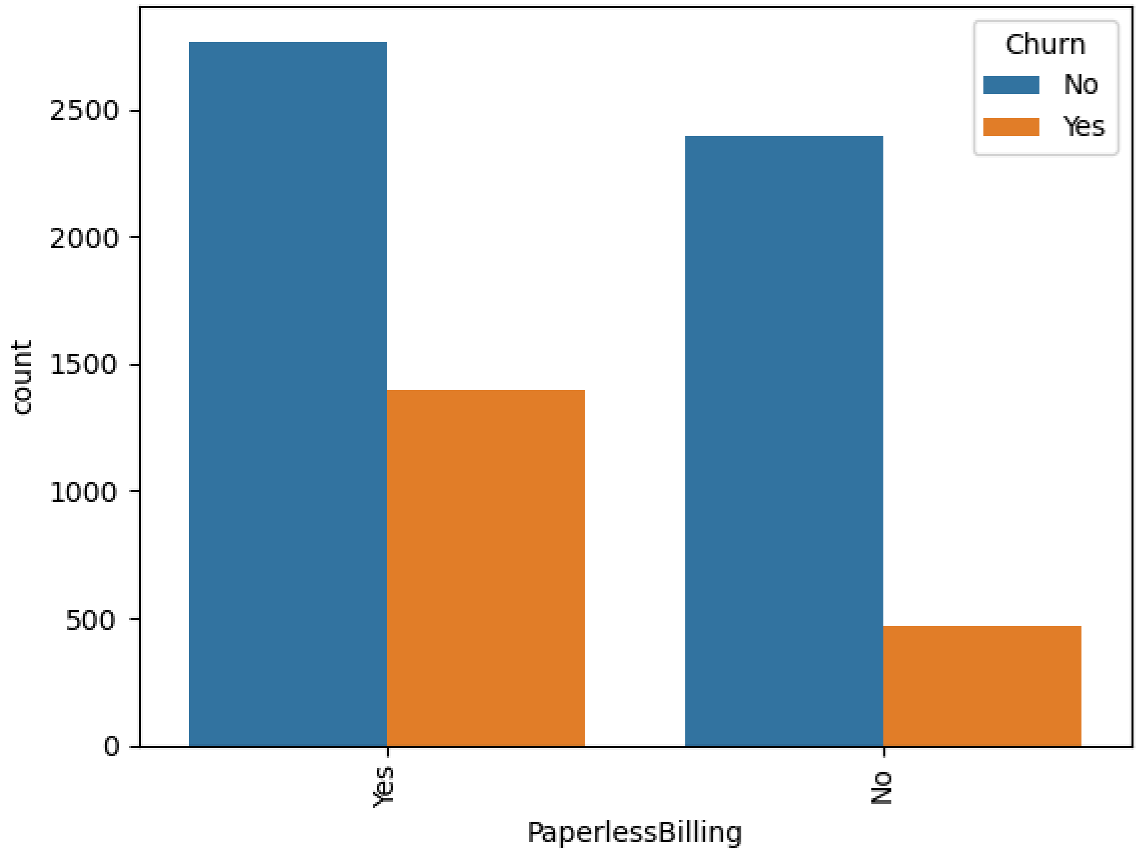


Loading [MathJax]/extensions/Safe.js

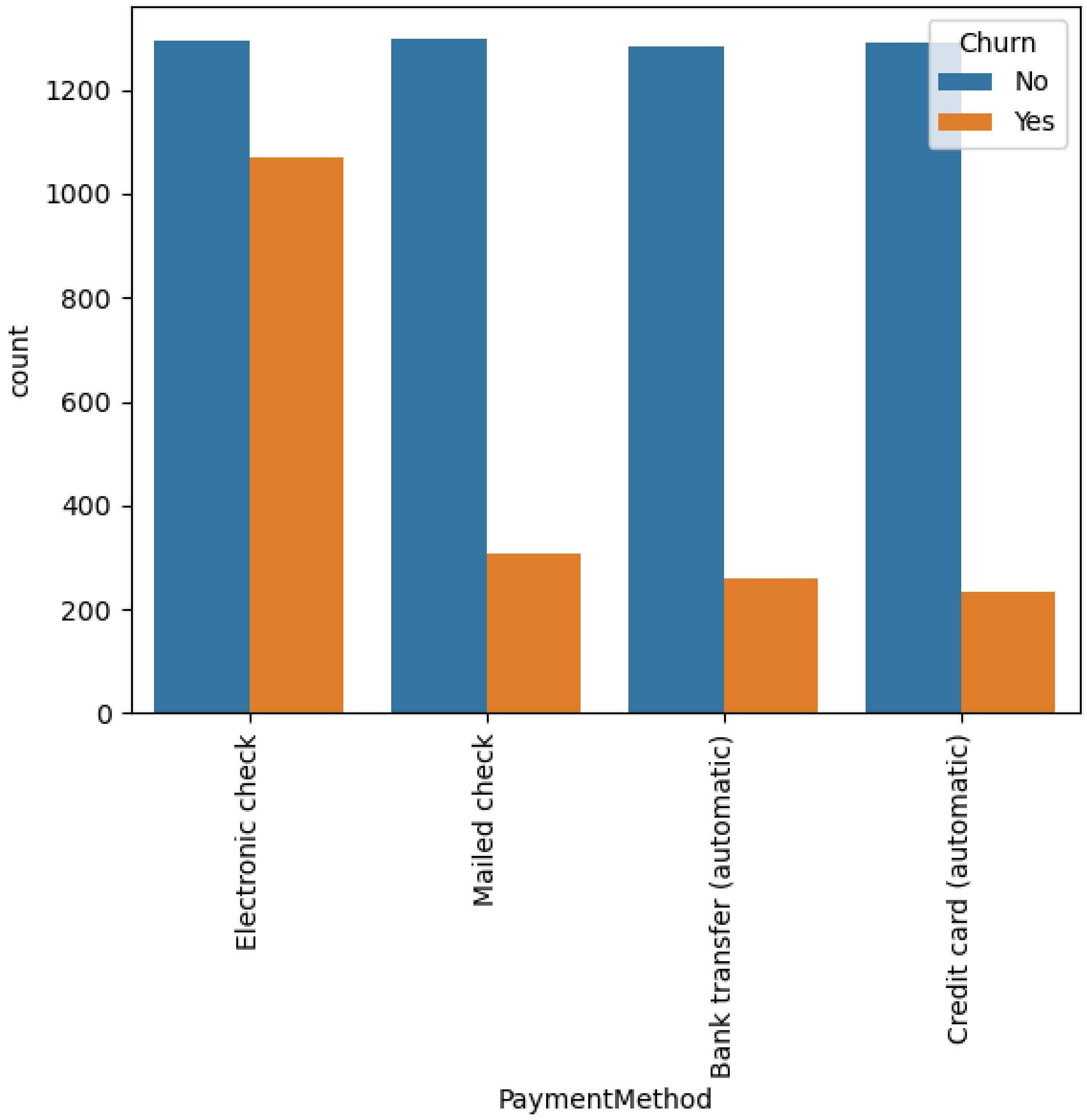


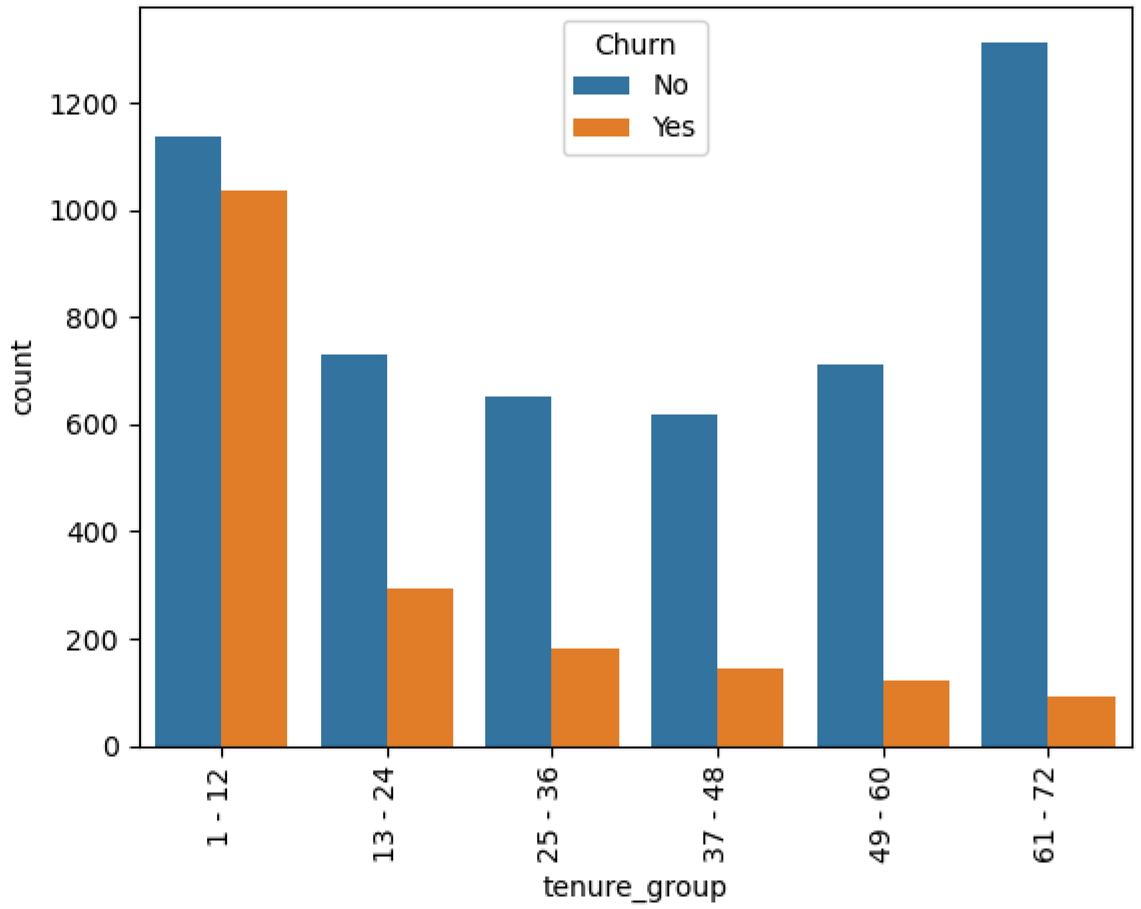
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingMovies





Loading [MathJax]/extensions/Safe.js


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



*convert the target variable into No=0 and Yes=1.

In [23]:
```
data["Churn"]=np.where(data.Churn=="Yes",1,0)
```

In [24]:
```
data.head()
```

Out[24]:

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

In [25]:
```
# converting all categorical variabels into dummy variabels.
data_dummies=pd.get_dummies(data)
data_dummies.head()
```

Loading [MathJax]/extensions/Safe.js

```
Out[25]:
```

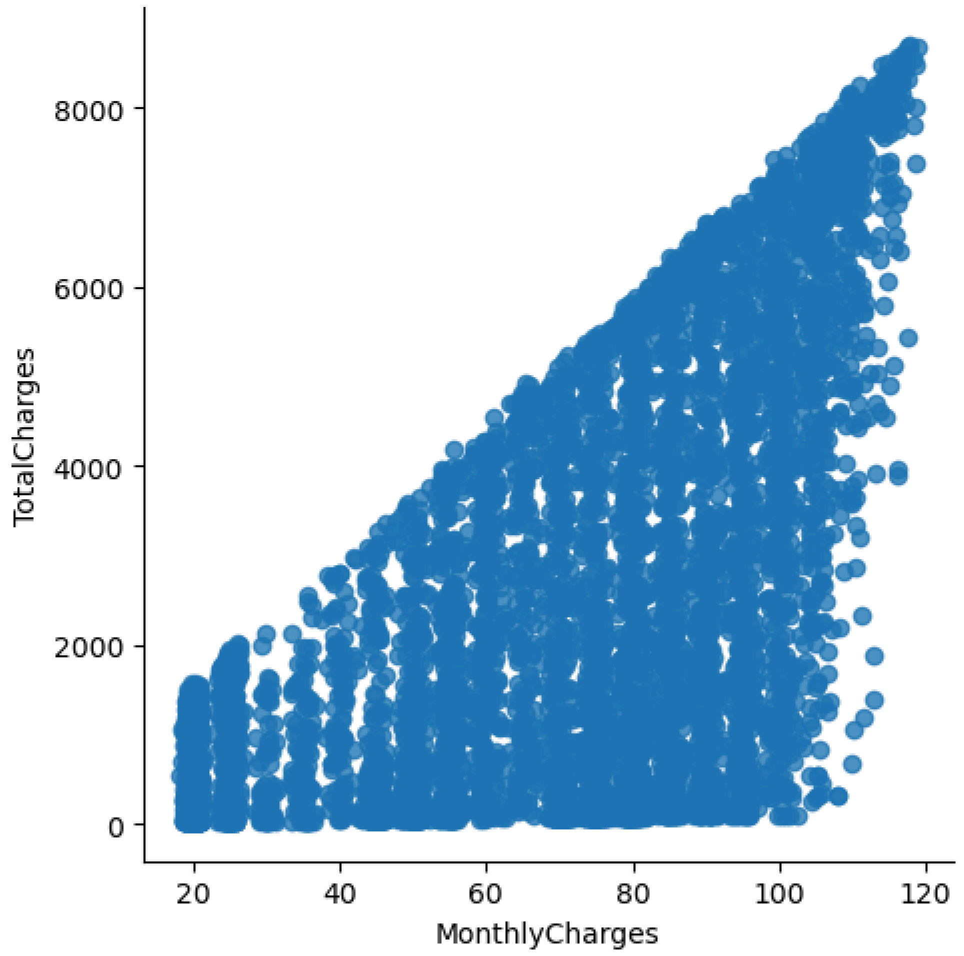
	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes
0	0	29.85	29.85	0	1	0	0	1
1	0	56.95	1889.50	0	0	1	1	0
2	0	53.85	108.15	1	0	1	1	0
3	0	42.30	1840.75	0	0	1	1	0
4	0	70.70	151.65	1	1	0	1	0

5 rows × 51 columns

```
In [26]: # Relationship between monthly charges and total charges.
```

```
In [27]: sns.lmplot(data=data_dummies,x="MonthlyCharges",y="TotalCharges",fit_reg=False)
```

```
Out[27]: <seaborn.axisgrid.FacetGrid at 0x2abd5e35f10>
```



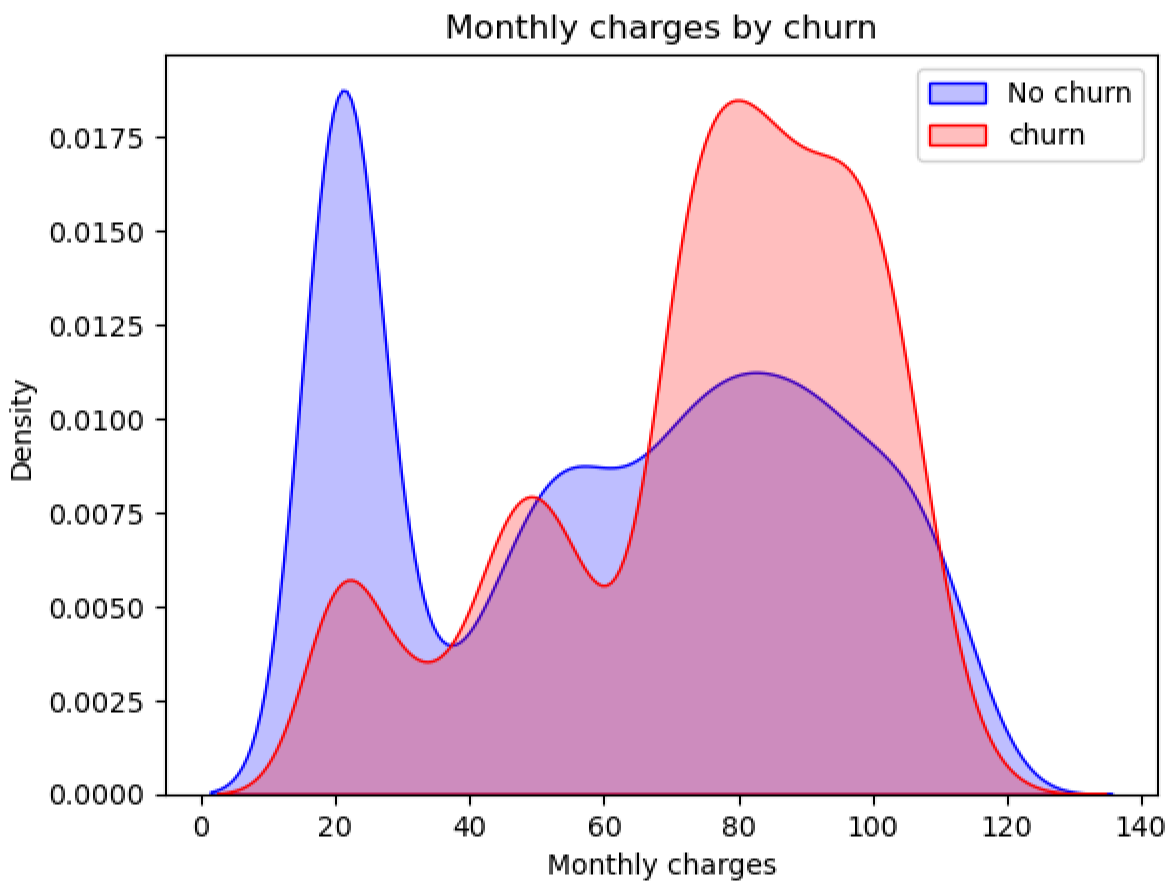
- Total charges increase as Monthly charges increases.

- churn by monthly charges and total charges.

```
In [28]: Mth=sns.kdeplot(data_dummies.MonthlyCharges[(data_dummies["Churn"]==0)],color="blue",sha
         Mth=sns.kdeplot(data_dummies.MonthlyCharges[(data_dummies["Churn"]==1)],ax=Mth,color="re
         Mth.legend(["No churn","churn"],loc="upper right")
         Mth.set_ylabel("Density")
         Mth.set_xlabel("Monthly charges")
         Mth.set_title("Monthly charges by churn")
```

Loading [MathJax]/extensions/Safe.js

```
Out[28]: Text(0.5, 1.0, 'Monthly charges by churn')
```

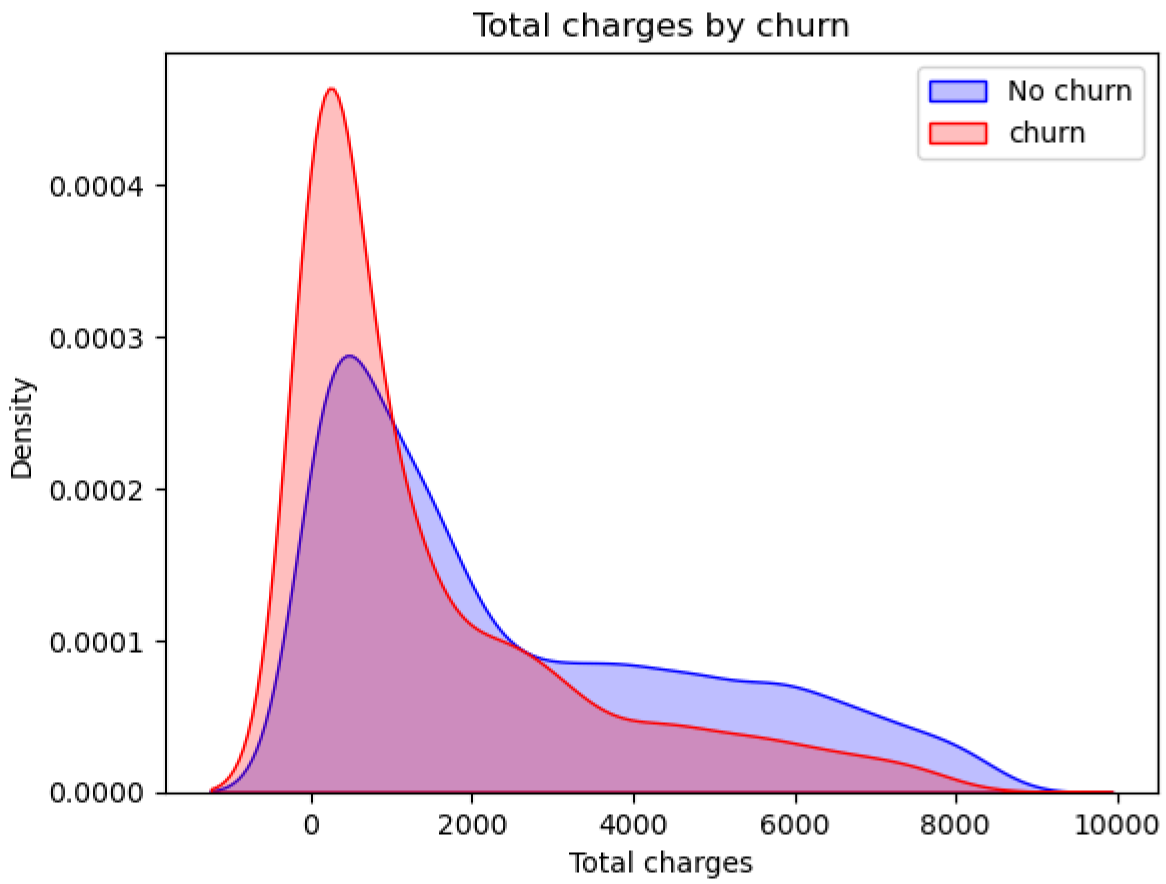


- churn is high when monthly charges are high.

```
In [29]: Tot=sns.kdeplot(data_dummies.TotalCharges[(data_dummies["Churn"]==0)],color="blue",shade
         Tot=sns.kdeplot(data_dummies.TotalCharges[(data_dummies["Churn"]==1)],ax=Tot,color="red"
         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[29]: Text(0.5, 1.0, 'Total charges by churn')
```

Loading [MathJax]/extensions/Safe.js

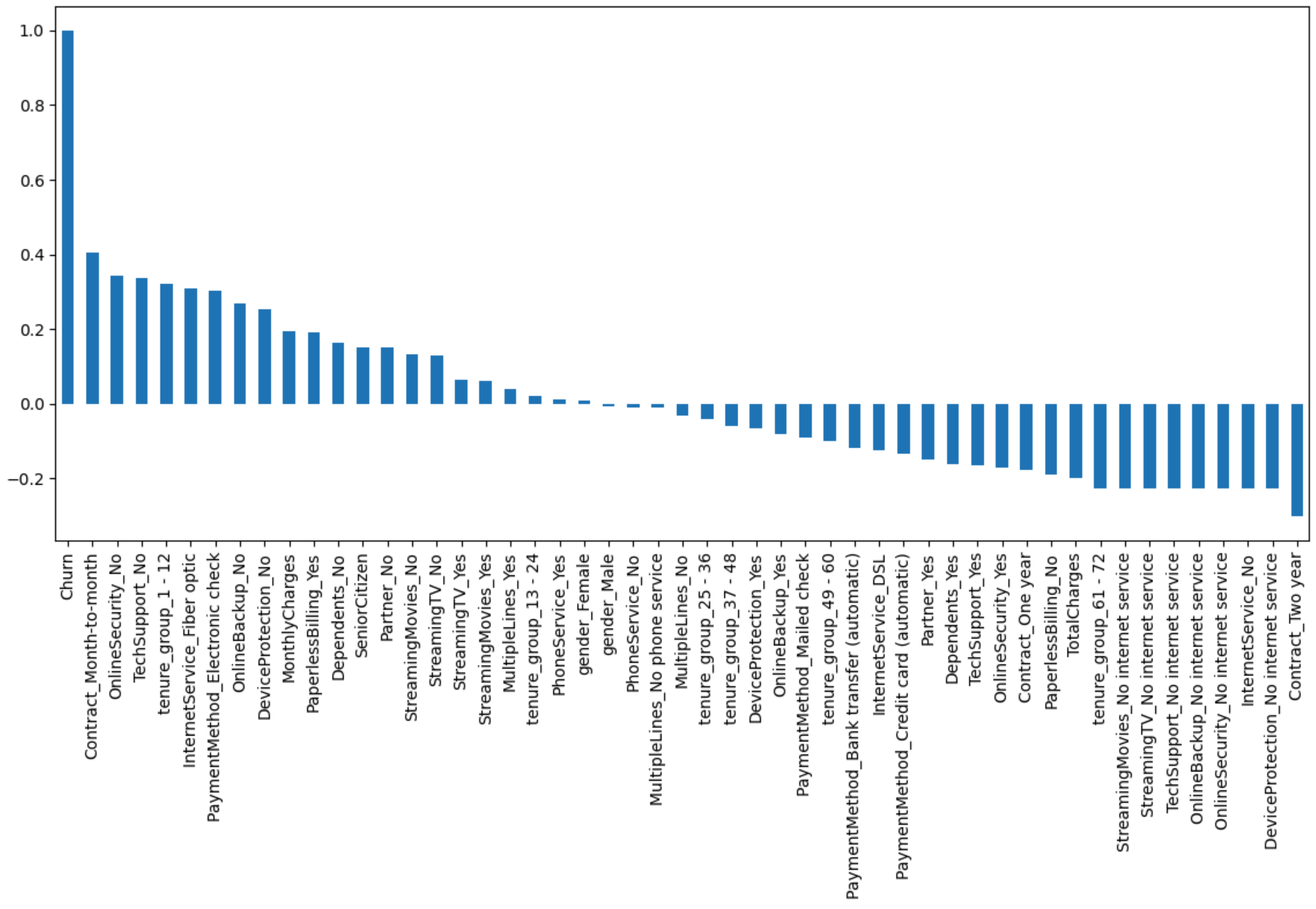


- High churn at lower total charges.

In [30]:
```
# Build a correlation of all predictors with churn.
plt.figure(figsize=(14,6))
data_dummies.corr()["Churn"].sort_values(ascending=False).plot(kind="bar")
```

Out[30]:
```
<Axes: >
```

Loading [MathJax]/extensions/Safe.js



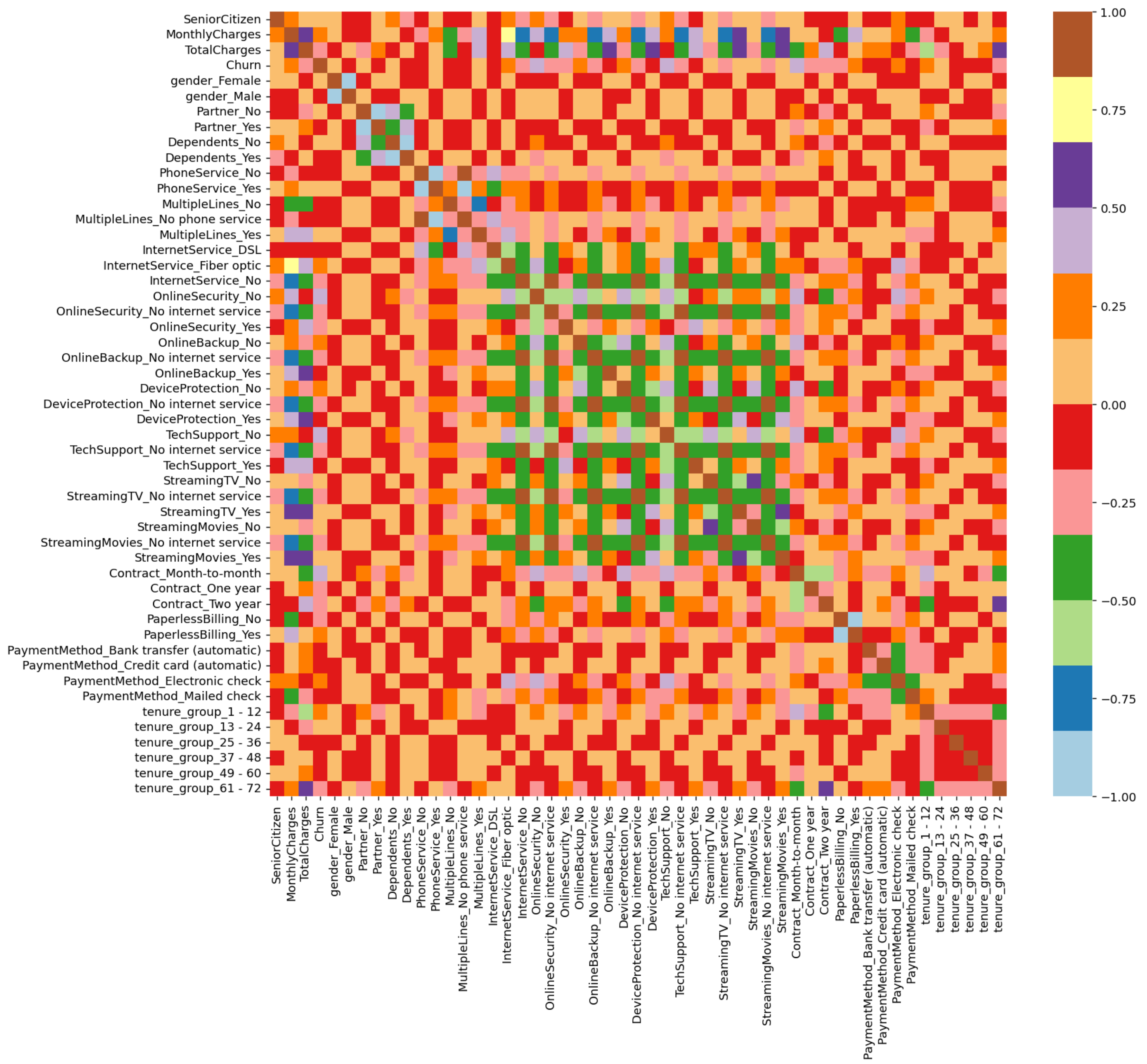
Insights.

- Higher rate of churn can be seen in month_to_month contract,No_onlinesecurity,No_technical support,Fibre_optic internerservice.
- Lower rate of churn can be seen in Two year contract,subscription without internet service and the customers engaged for 5+years.
- Gender and phone service has no impact on churn.

In [31]:
```
# This is also evident from Heatmap.
plt.figure(figsize=(14,12),dpi=130)
sns.heatmap(data_dummies.corr(),cmap="Paired")
```

Out[31]:
```
<Axes: >
```

Loading [MathJax]/extensions/Safe.js



Bivariate Analysis.

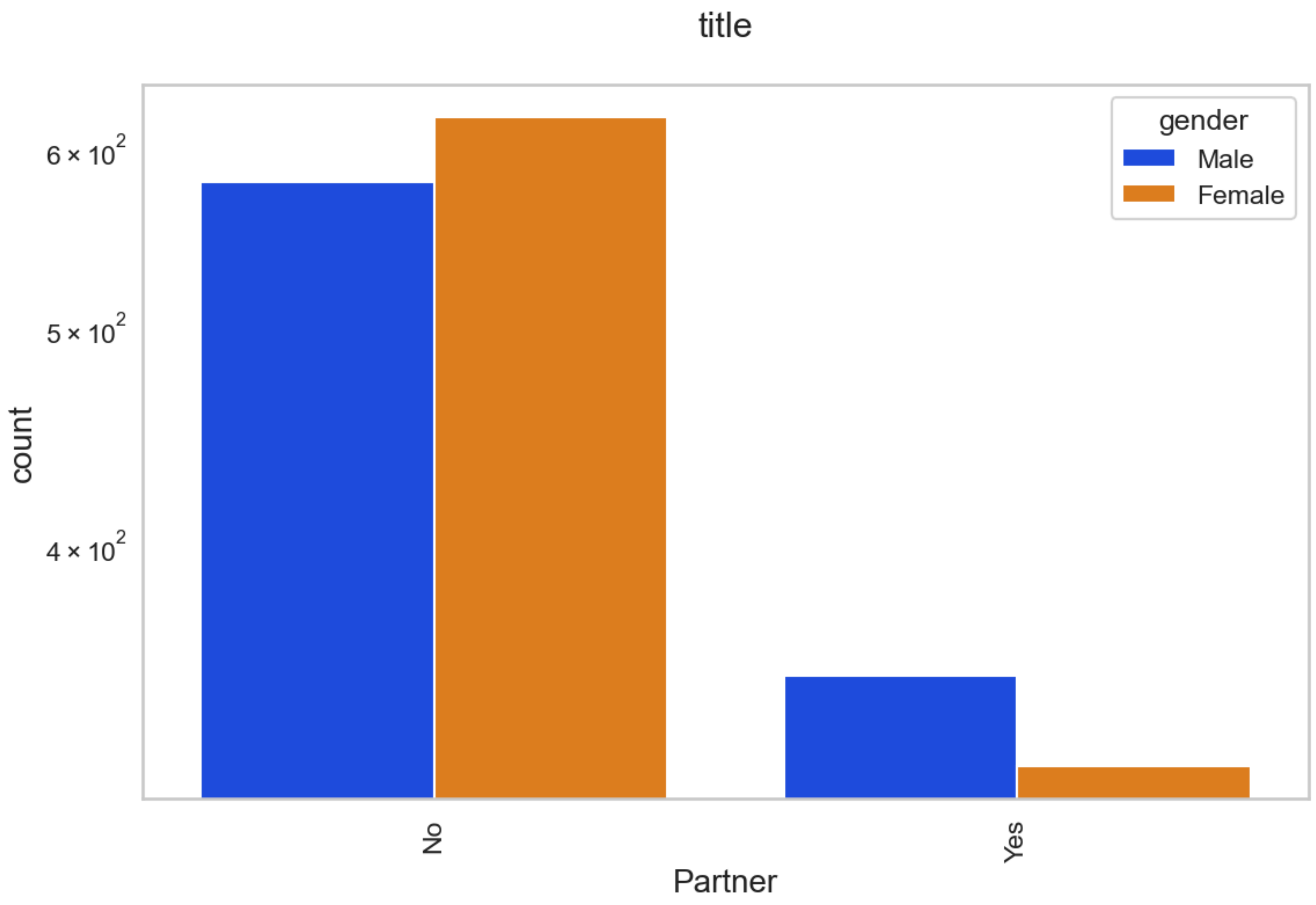
In [32]:
```
new_data_target0=data.loc[data["Churn"]==0]
new_data_target1=data.loc[data["Churn"]==1]
```

In [33]:
```
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=90)
    plt.yscale("log")
    plt.title("title")
    ax=sns.countplot(data=df,x=col,order=df[col].value_counts().index,hue=hue,palette="b
    plt.show
```

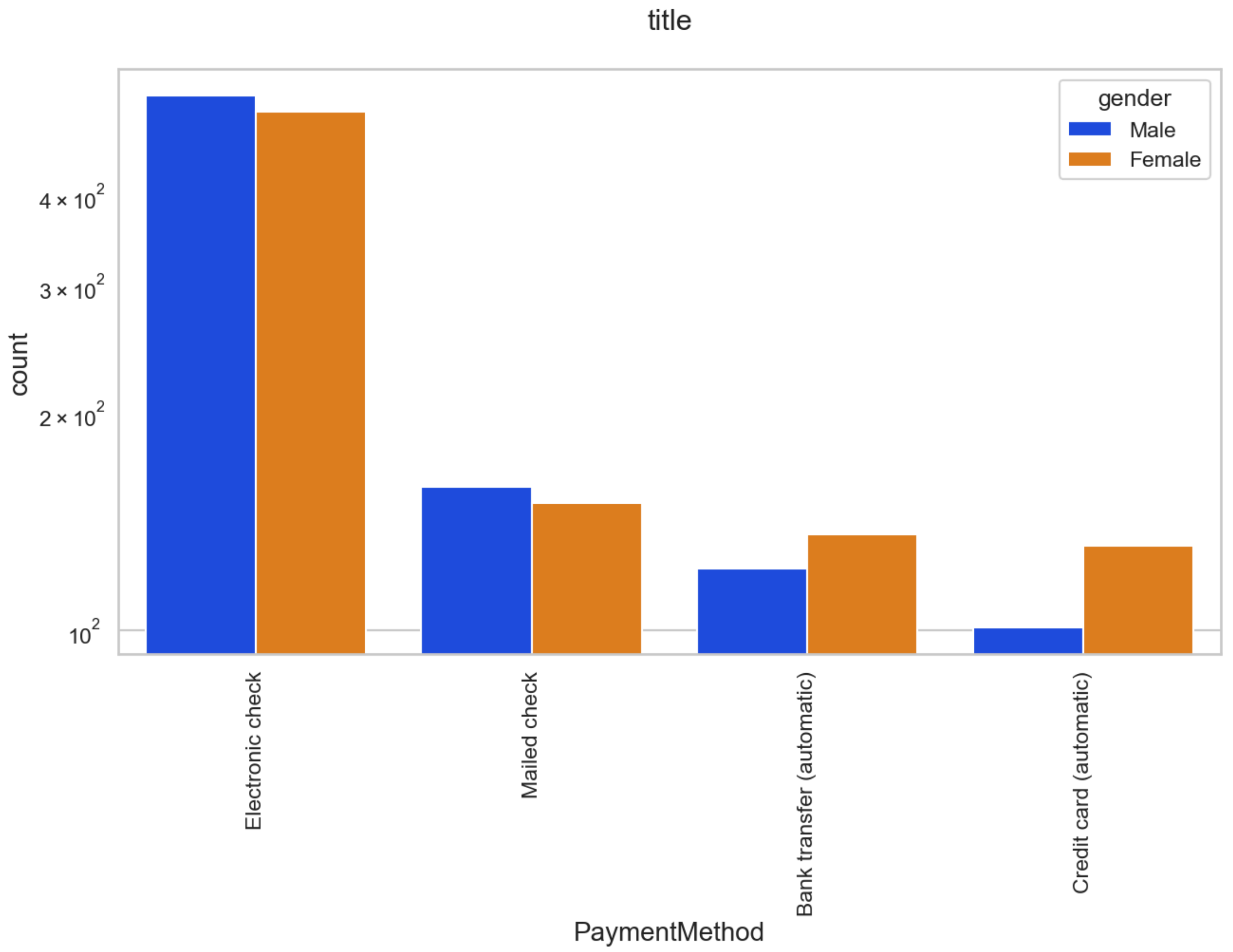
Loading [MathJax]/extensions/Safe.js

```
In [34]: uniplot(new_data_target1,col="Partner",title="distribution of gender for churned custome
```



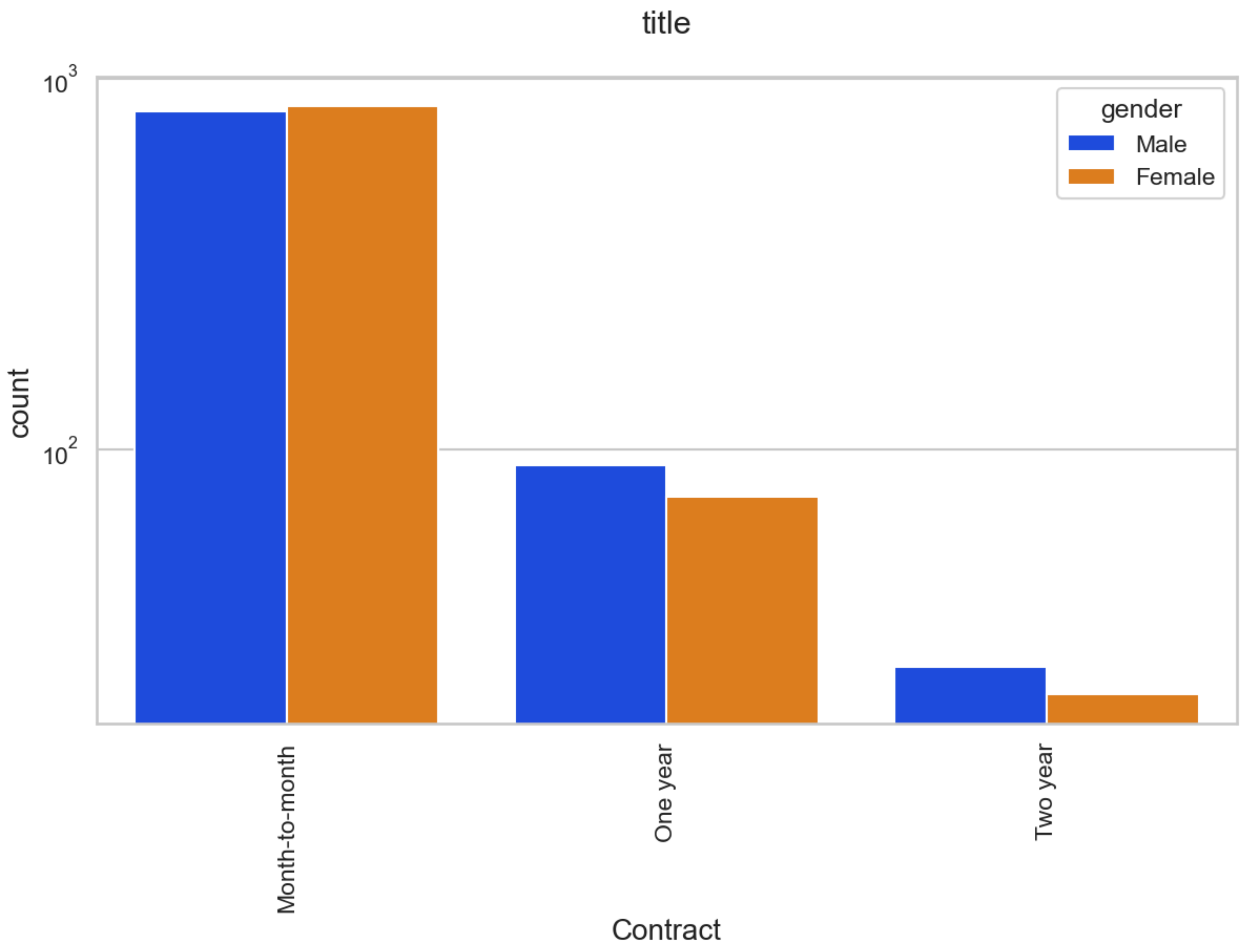
```
In [35]: uniplot(new_data_target1,col="PaymentMethod",title="distribution of gender for churned c
```

Loading [MathJax]/extensions/Safe.js



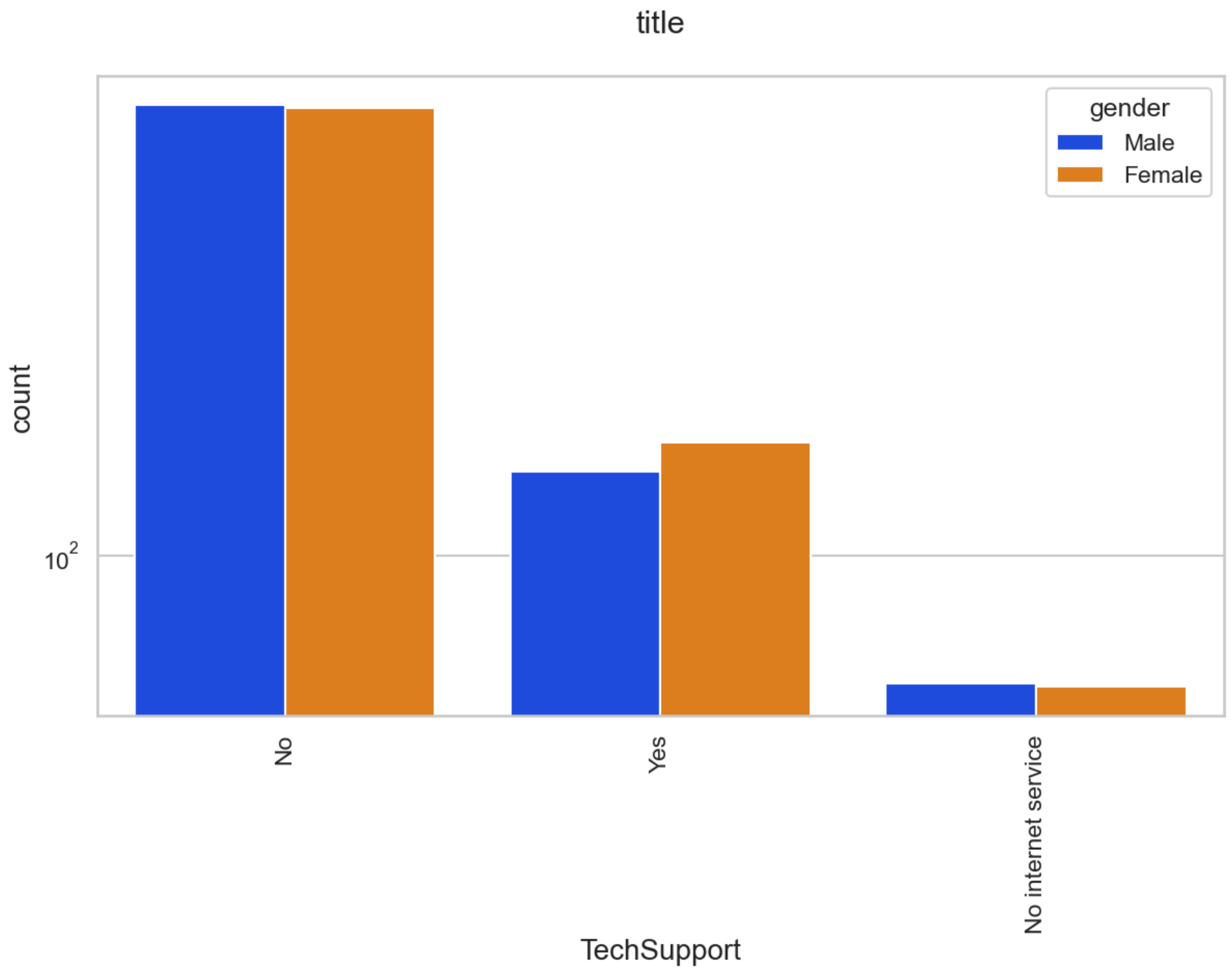
In [36]:
```
uniplot(new_data_target1,col="Contract",title="distribution of gender for churned custom
```

Loading [MathJax]/extensions/Safe.js



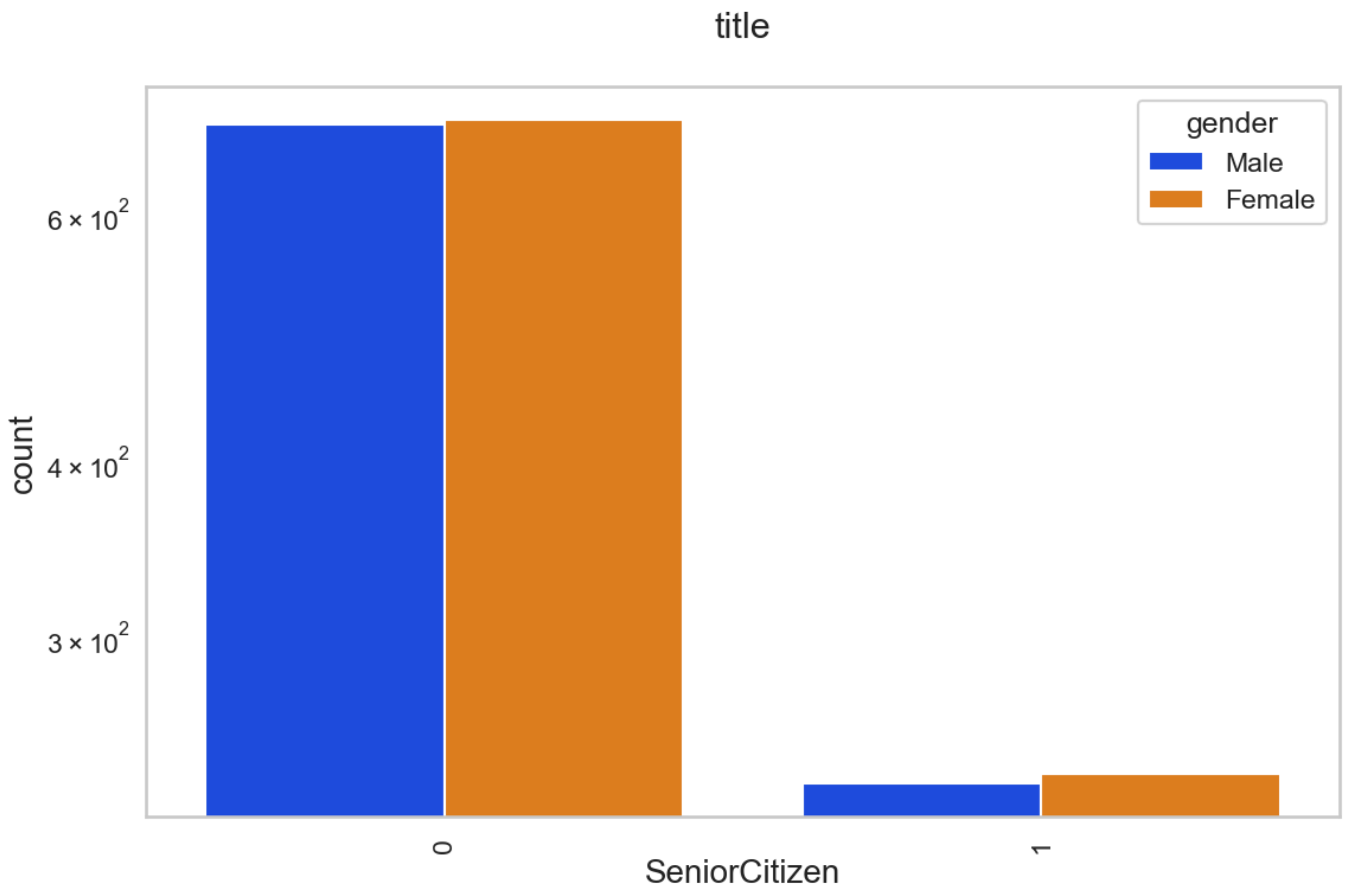
```
In [37]: uniplot(new_data_target1,col="TechSupport",title="distribution of gender for churned cus
```

Loading [MathJax]/extensions/Safe.js



```
In [38]: uniplot(new_data_target1,col="SeniorCitizen",title="distribution of gender for churned c
```

Loading [MathJax]/extensions/Safe.js



Conclusions

- Electronic check medium has the highest number of churn customers.
- Monthly customers are more likely to churn as there is no contract.
- Non senior citizens are high in number in terms of churn.
- No online security,No tech support categoty are high churners.

In [39]:
```
data_dummies.to_csv("data.csv")
```

In [40]:
```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
#from imblearn.combine import SMOTEENN
```

In [41]:
```
df=pd.read_csv("data.csv")
```

In [42]:
```
df.head()
```

Loading [MathJax]/extensions/Safe.js

Out[42]:

	Unnamed: 0	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	P
0	0	0	29.85	29.85	0	1	0	0	
1	1	0	56.95	1889.50	0	0	1	1	
2	2	0	53.85	108.15	1	0	1	1	
3	3	0	42.30	1840.75	0	0	1	1	
4	4	0	70.70	151.65	1	1	0	1	

5 rows × 52 columns

In [43]:
```
df=df.drop("Unnamed: 0",axis=1)
```

In [44]:
```
df.head()
```

Out[44]:

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

5 rows × 51 columns

In [45]:
```
# creating x and y variables.
x=df.drop("Churn",axis=1)
y=df["Churn"]
```

In [46]:
```
x
```

Out[46]:

	SeniorCitizen	MonthlyCharges	TotalCharges	gender_Female	gender_Male	Partner_No	Partner_Yes	Dep
0	0	29.85	29.85	1	0	0	1	
1	0	56.95	1889.50	0	1	1	0	
2	0	53.85	108.15	0	1	1	0	
3	0	42.30	1840.75	0	1	1	0	
4	0	70.70	151.65	1	0	1	0	
...	...	...	...	...	...	...	...	
7027	0	84.80	1990.50	0	1	0	1	
7028	0	103.20	7362.90	1	0	0	1	
7029	0	29.60	346.45	1	0	0	1	
7030	1	74.40	306.60	0	1	0	1	
7031	0	105.65	6844.50	0	1	1	0	

7032 rows × 50 columns

In [47]:
```
y
```

```
Out[47]: 0       0
         1       0
         2       1
         3       0
         4       1
                ..
         7027    0
         7028    0
         7029    0
         7030    1
         7031    0
         Name: Churn, Length: 7032, dtype: int64
```

```
In [48]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Decision Tree Classifier.

```
In [49]: model_dt=DecisionTreeClassifier(criterion="gini",random_state=42,max_depth=6,min_samples
```

```
In [50]: model_dt.fit(x_train,y_train)
```

```
Out[50]: ▼                DecisionTreeClassifier
         DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=42)
```

```
In [51]: y_pred=model_dt.predict(x_test)
```

```
In [52]: y_pred
```

```
Out[52]: array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
In [53]: print(classification_report(y_test,y_pred,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1528
           1       0.66      0.56      0.61       582

    accuracy                           0.80      2110
   macro avg       0.75      0.72      0.74      2110
weighted avg       0.79      0.80      0.79      2110
```

```
In [59]: print(confusion_matrix(y_test,y_pred))
```

```
[[1190  359]
 [ 428  133]]
```

```
In [60]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report

         # Assuming you have your data stored in X (features) and y (target)
         # Split the data into training and testing sets
         x_train, y_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42

         # Initialize the logistic regression model
         model = LogisticRegression()

         # Train the model on the training data
Loading [MathJax]/extensions/Safe.js rain, y_train)
```

```python
# Make predictions on the test data
predictions = model.predict(x_test)

# Evaluate the model
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.73      0.78      0.76      1549
           1       0.26      0.21      0.23       561

    accuracy                           0.63      2110
   macro avg       0.50      0.50      0.49      2110
weighted avg       0.61      0.63      0.62      2110
```

```
In [ ]:
```

Loading [MathJax]/extensions/Safe.js