In [42]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
%matplotlib inline
```

In [43]:
```
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

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

Out[44]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
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 [45]:
```
df.shape
```

Out[45]:
```
(7043, 21)
```

In [46]:
```
df.columns.values
```

Out[46]:
```
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 [47]:
```
df.dtypes
```

Out[47]:
```
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 [48]:
```
# Checking the decriptive statistics of numeric values
df.describe()
```

Out[48]:

	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

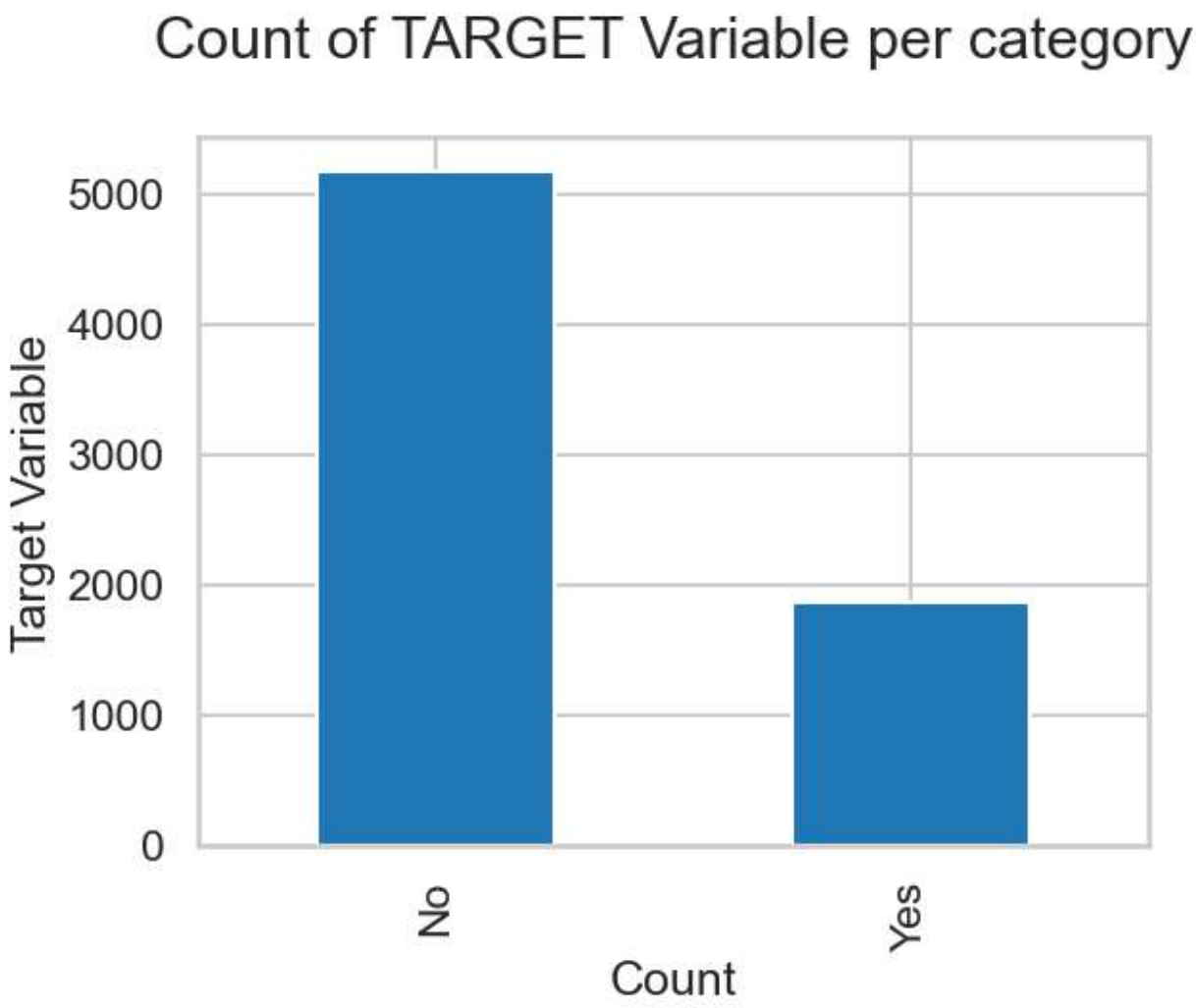
SeniorCitizen is actually a categorical hence the 25%-50%-75% distribution is not propoer

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 [49]:
```
df['Churn'].value_counts().plot(kind='bar')
plt.xlabel("Count")
plt.ylabel("Target Variable")
plt.title("Count of TARGET Variable per category")
```

Out[49]:
```
Text(0.5, 1.0, 'Count of TARGET Variable per category')
```



In [50]:
```
100*df['Churn'].value_counts()/len(df['Churn'])
```

Out[50]:
```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

In [51]:
```
df['Churn'].value_counts()
```

Out[51]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

- Data is highly imbalanced, ratio = 73.27
- So we analyse the data with other features while taking the target variable separately to get some insights.

In [52]:
```
# Concise Summary of the data frame
df.info(verbose=True)
```

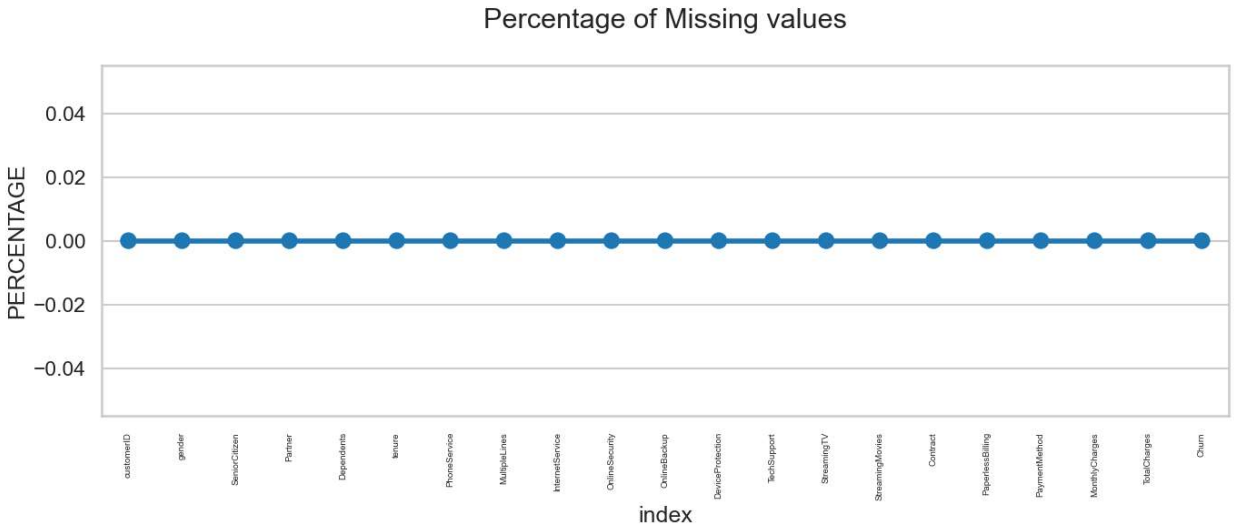
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   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 [53]:
```
missing = pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index()
missing
```

Out[53]:

	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 [54]:
```
plt.figure(figsize=(16,5))
ax = sns.pointplot(data=missing,x='index',y=0)
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

Data Cleaning

In [55]:
```
telco_data = df.copy()
```

In [56]:
```
telco_data.TotalCharges = pd.to_numeric(telco_data.TotalCharges, errors='coerce')
telco_data.isnull().sum()
```

Out[56]:
```
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 [57]:
```
telco_data.dtypes
```

Out[57]:
```
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
```

We can see that there are 11 missing values in the TotalCharges column.

In [58]:
```
telco_data.loc[telco_data ['TotalCharges'].isnull() == True]
```

Out[58]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service
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
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes

11 rows × 21 columns

Removing the missing values

In [59]:
```
telco_data.dropna(how='any', inplace=True)
```

Dividing the customers into bins based on tenure

In [60]:
```
# Get the max tenure
print(telco_data['tenure'].max())
```

```
72
```

In [61]:
```
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=False, labe
```

In [62]:
```
telco_data['tenure_group'].value_counts()
```

Out[62]:
```
1-12     2175
61-72    1407
13-24    1024
25-36     832
49-60     832
37-48     762
Name: tenure_group, dtype: int64
```

Removing columns not required for processing

In [63]:
```
#drop column customerID and tenure
telco_data.drop(columns=['customerID', 'tenure'], axis=1, inplace=True)
telco_data.head()
```

Out[63]:

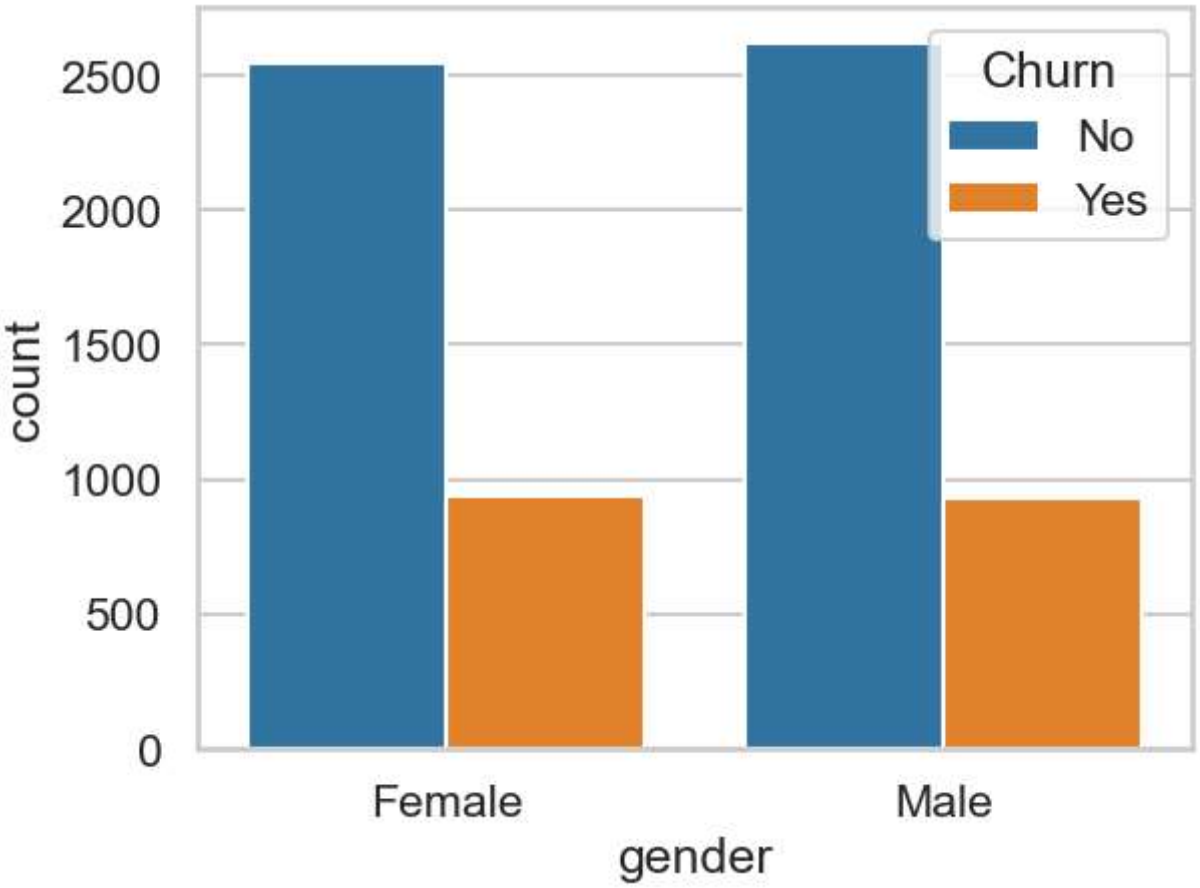
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSe
0	Female	0	Yes	No	No	No phone service	DSL	
1	Male	0	No	No	Yes	No	DSL	
2	Male	0	No	No	Yes	No	DSL	
3	Male	0	No	No	No	No phone service	DSL	
4	Female	0	No	No	Yes	No	Fiber optic	

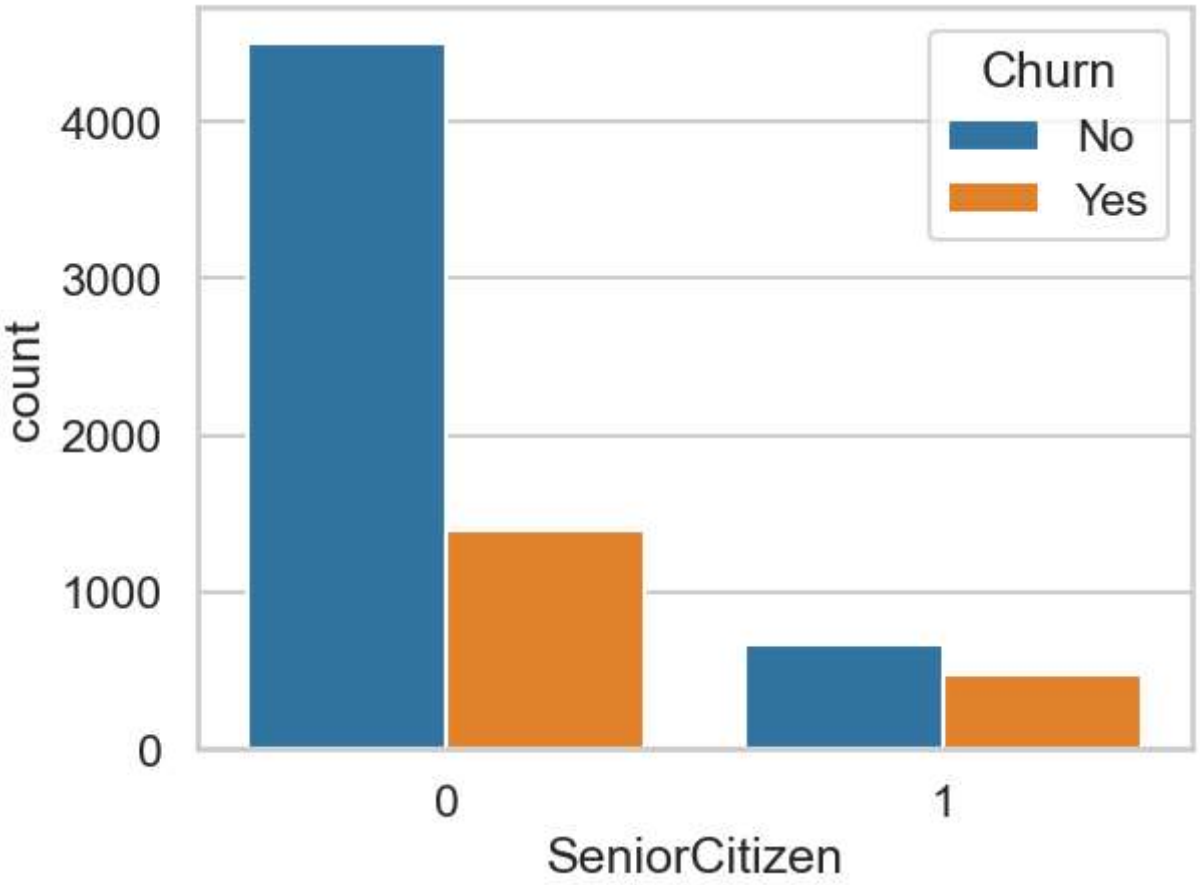
Data Exploration

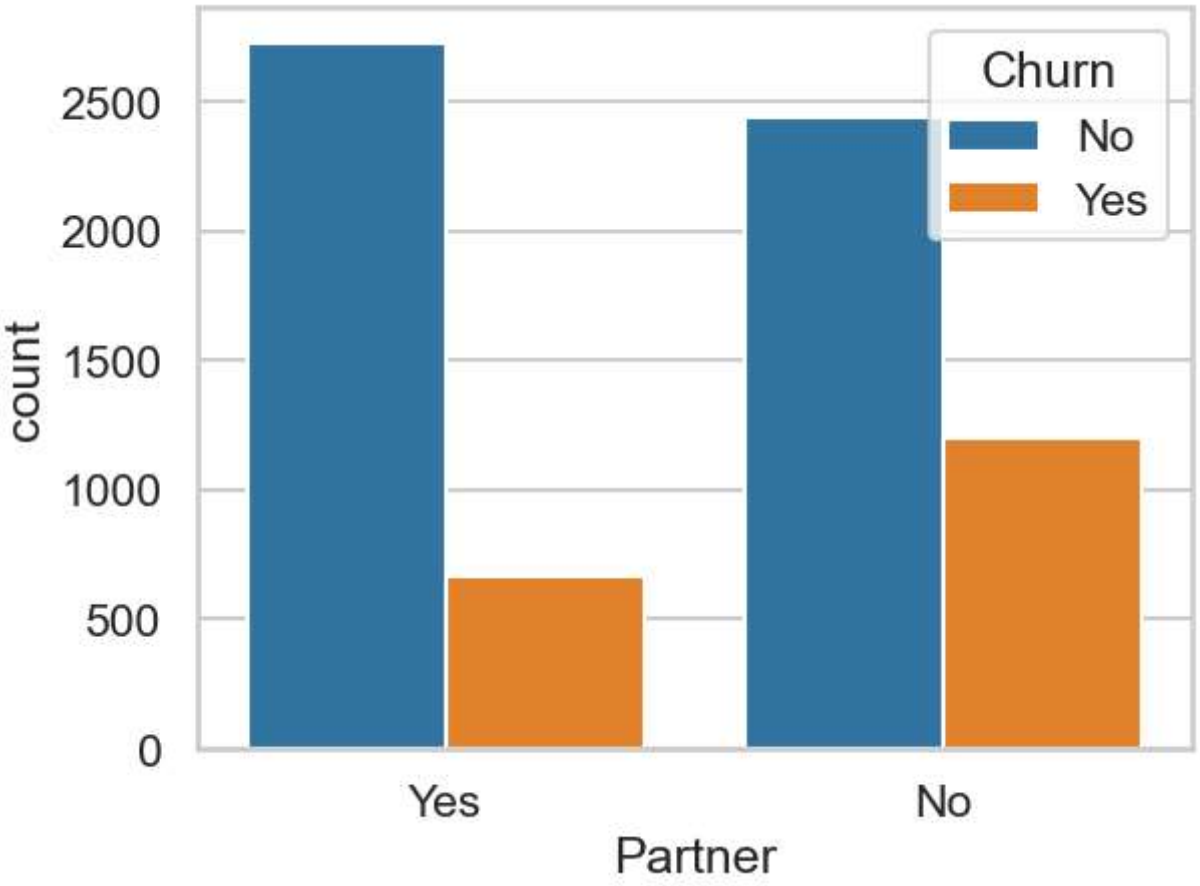
1. Plot distribution of individual predictors by churn

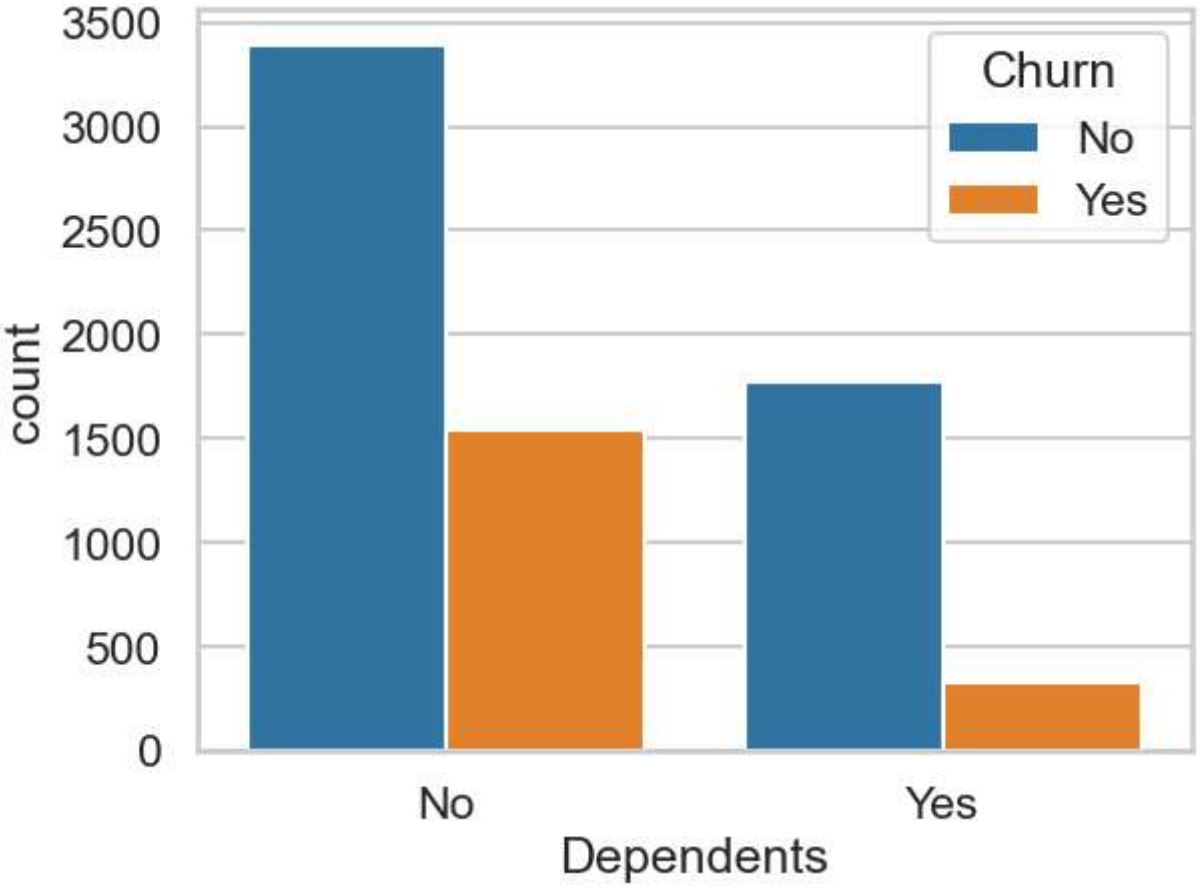
Univariate Analysis

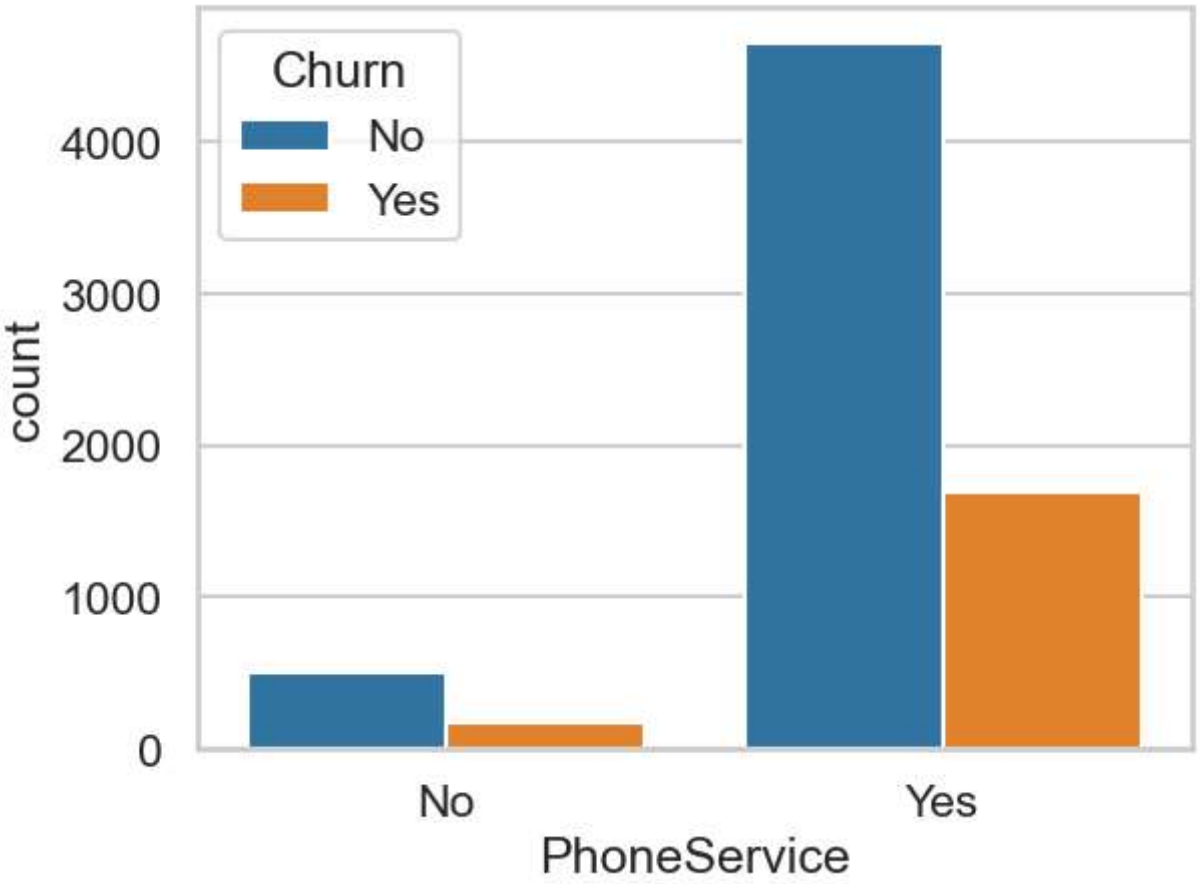
In [64]:
```
for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges','Monthl
    plt.figure(i)
    sns.countplot(data=telco_data, x=predictor, hue='Churn')
```

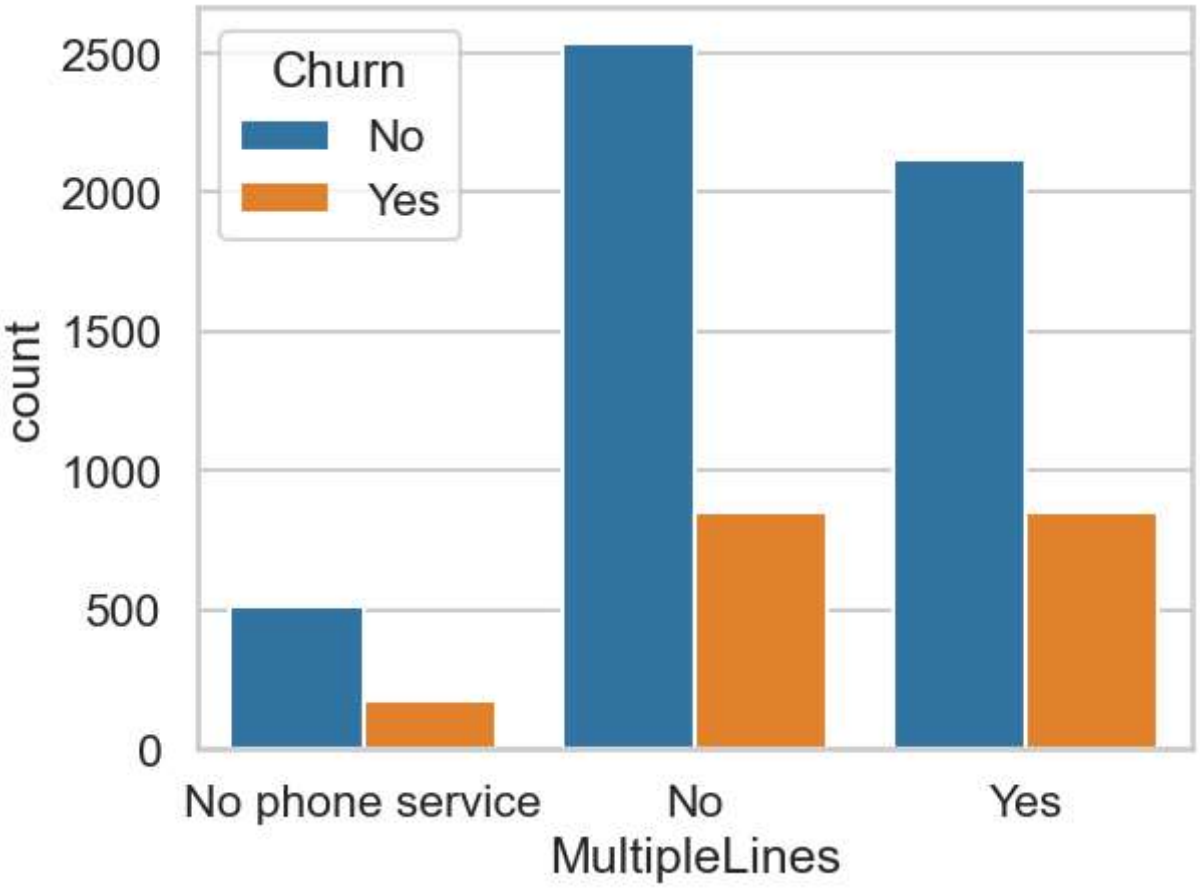

Churn
No
Yes
count
2500
2000
1500
1000
500
0
Female
Male
gender

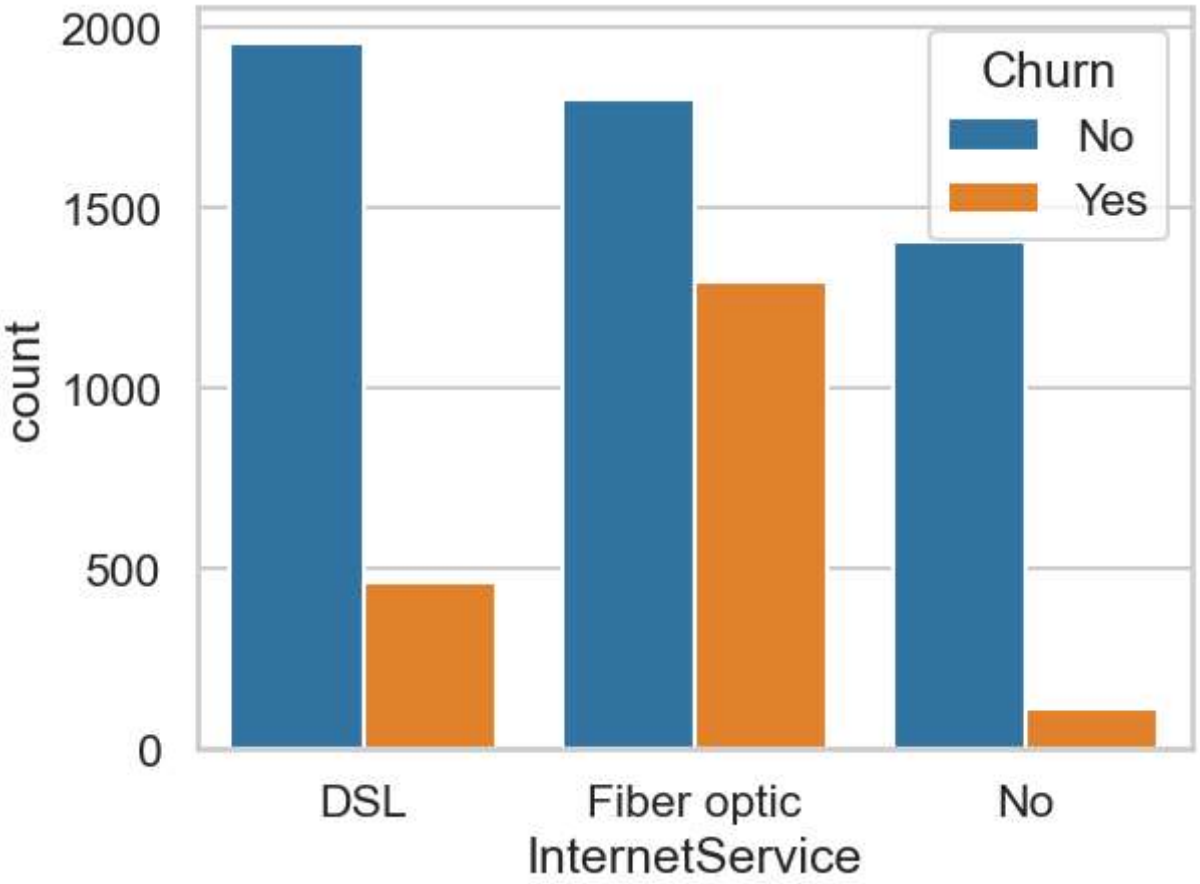

Churn
No
Yes
count
4000
3000
2000
1000
0
0
1
SeniorCitizen

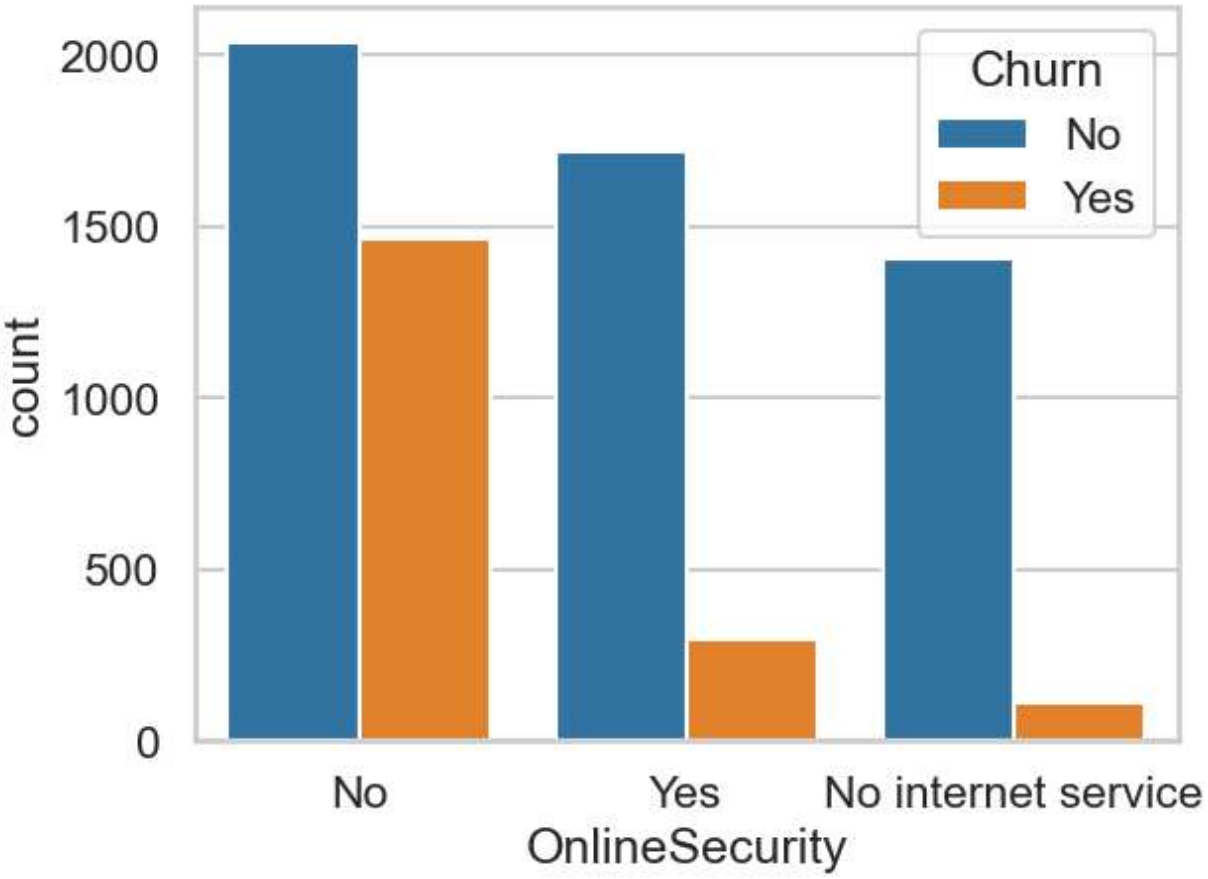

Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
Partner

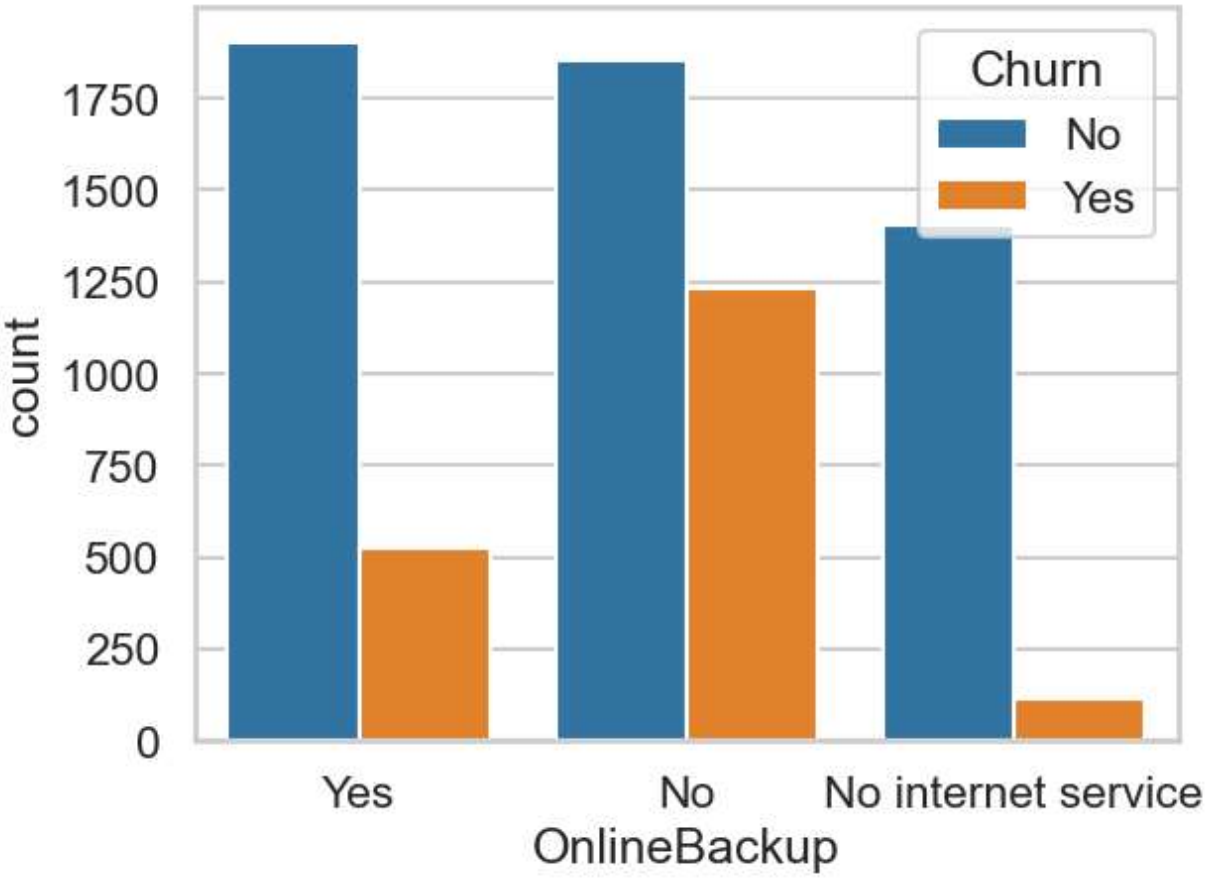

Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents

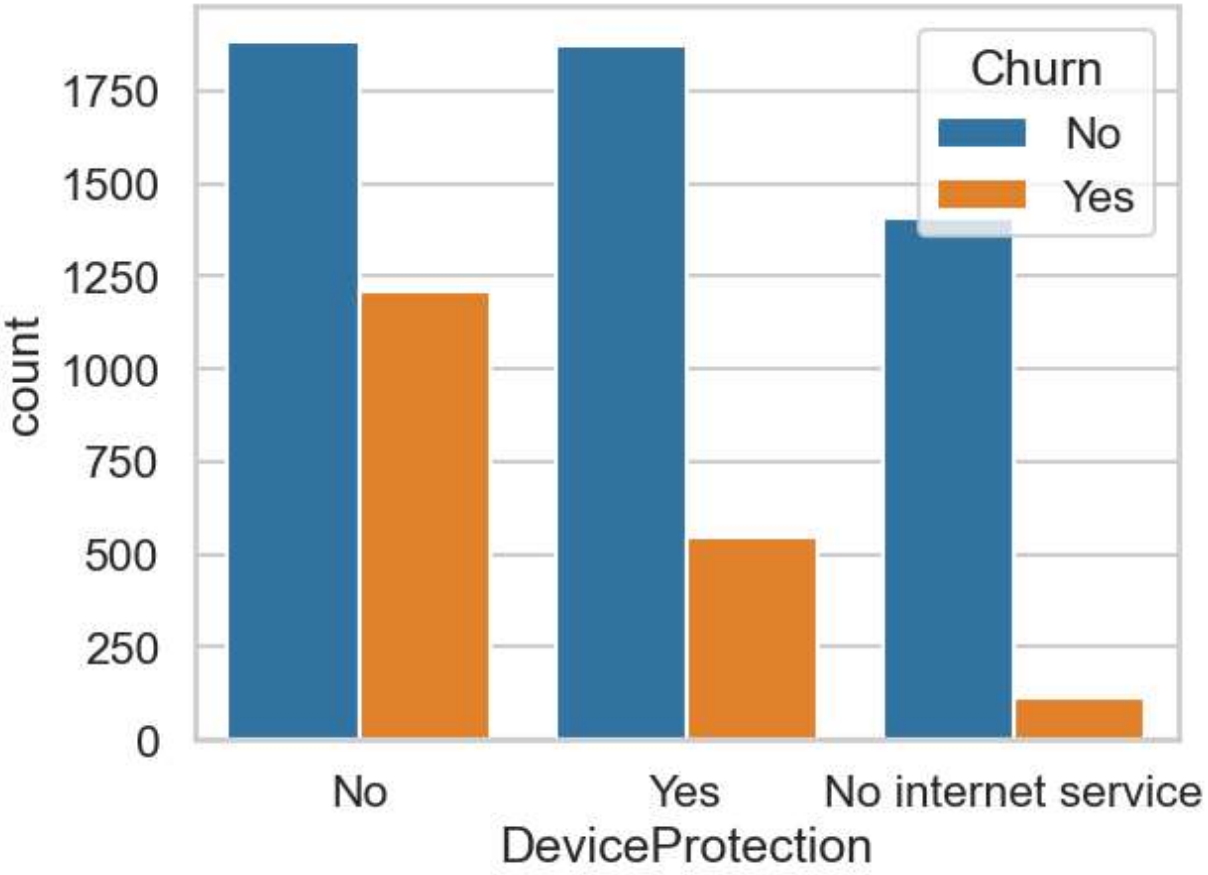

Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

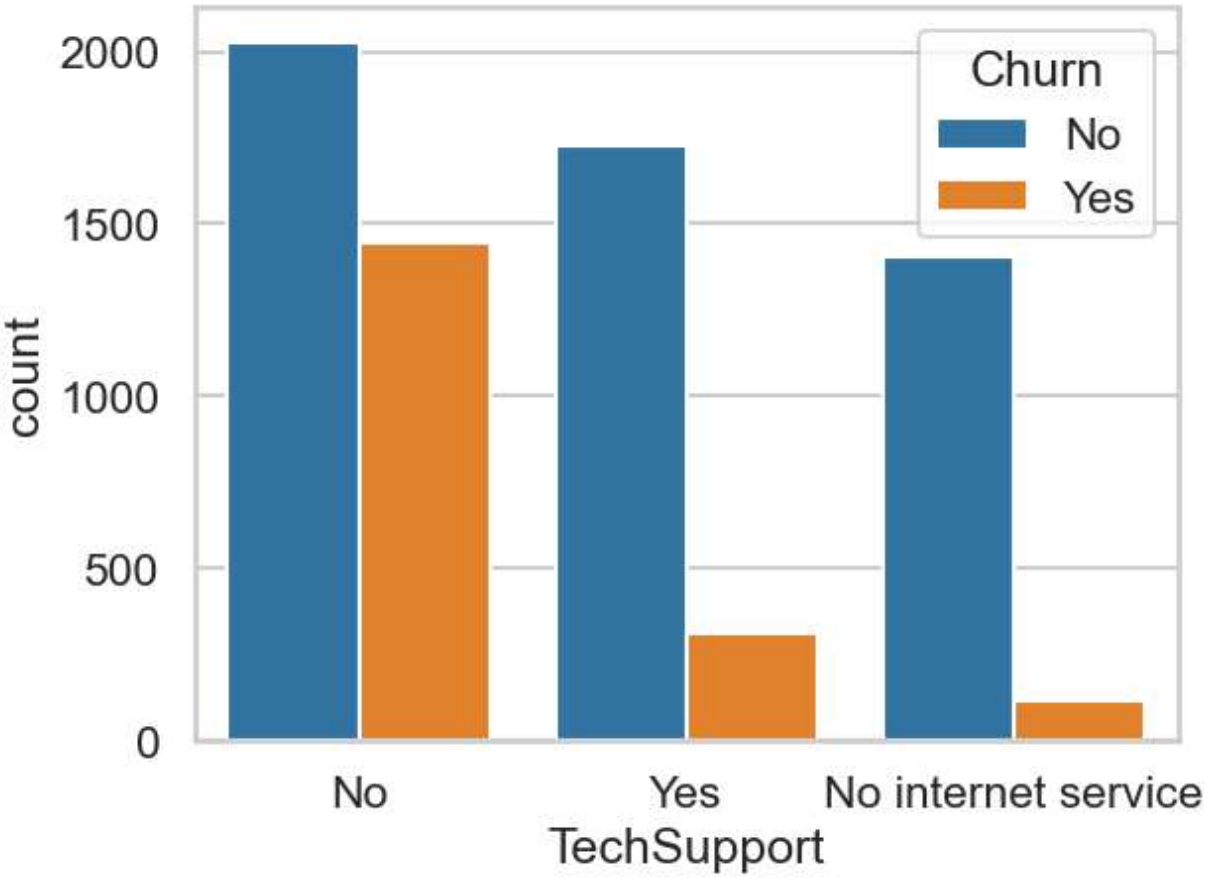

Churn
No
Yes
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

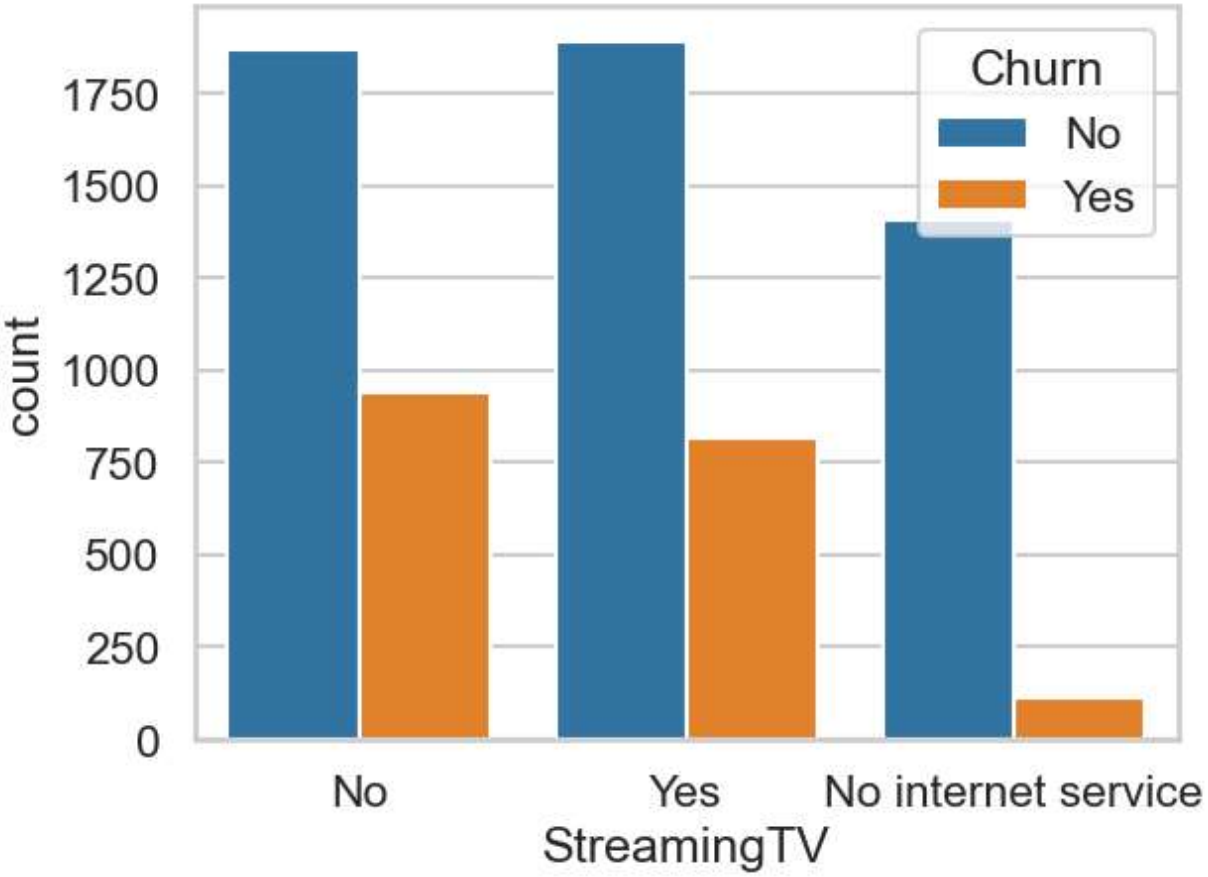

2000
1500
1000
500
0
count
Churn
No
Yes
DSL
Fiber optic
No
InternetService

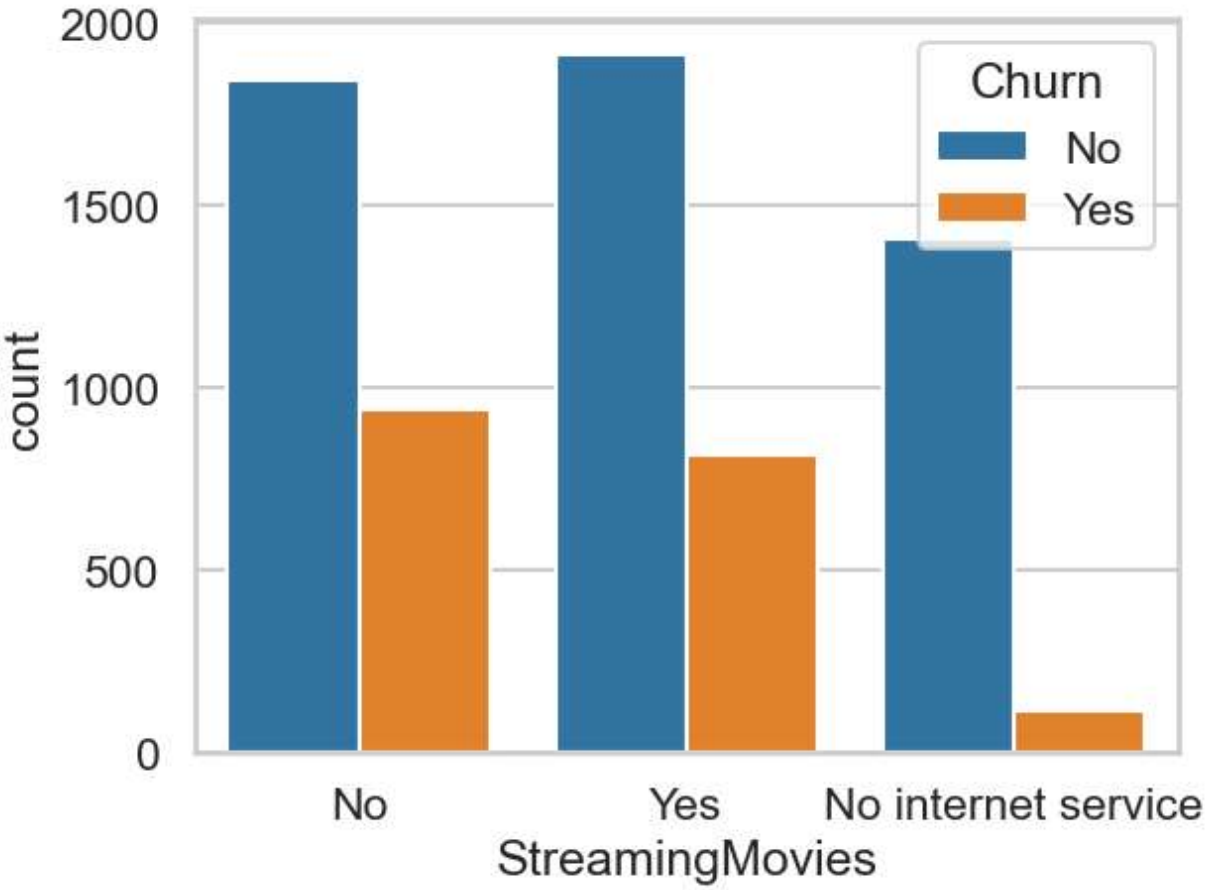

2000
1500
1000
500
0
count
Churn
No
Yes
No
Yes
No internet service
OnlineSecurity

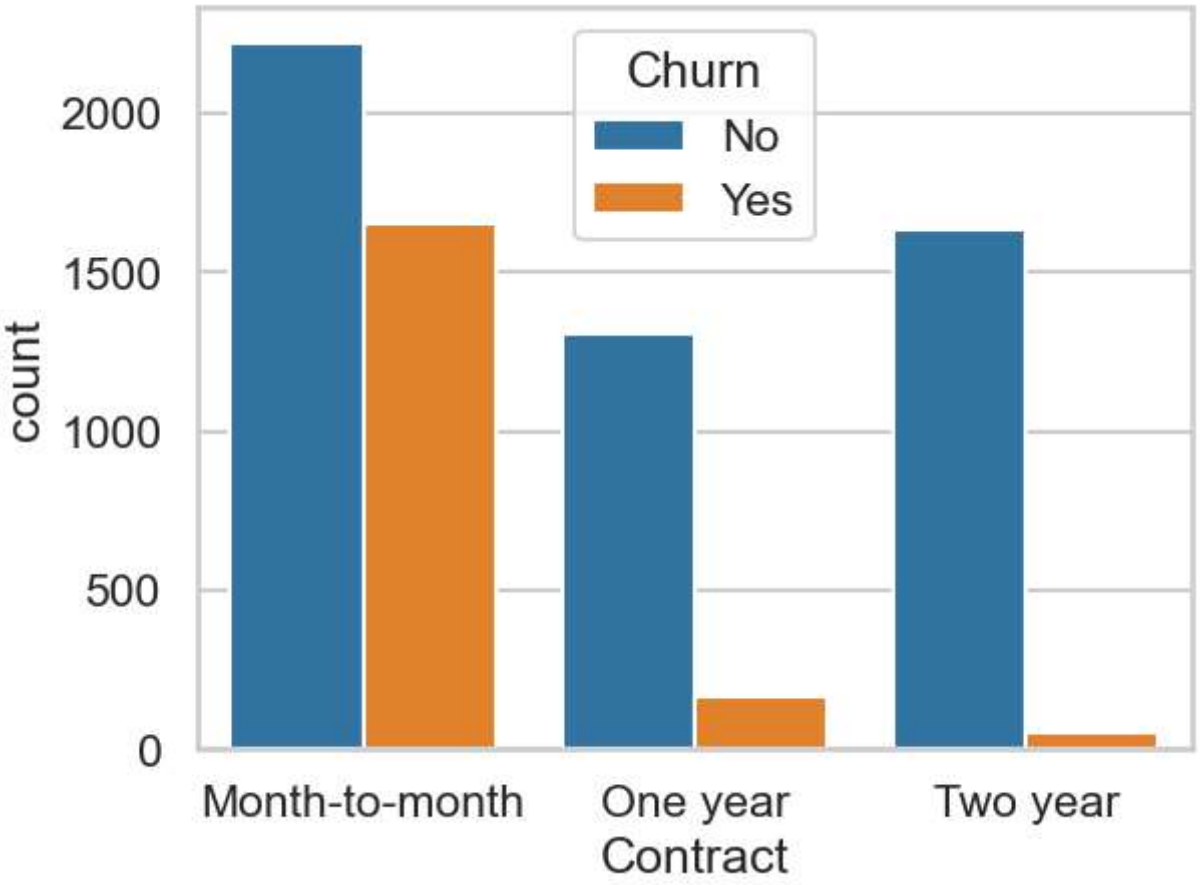

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

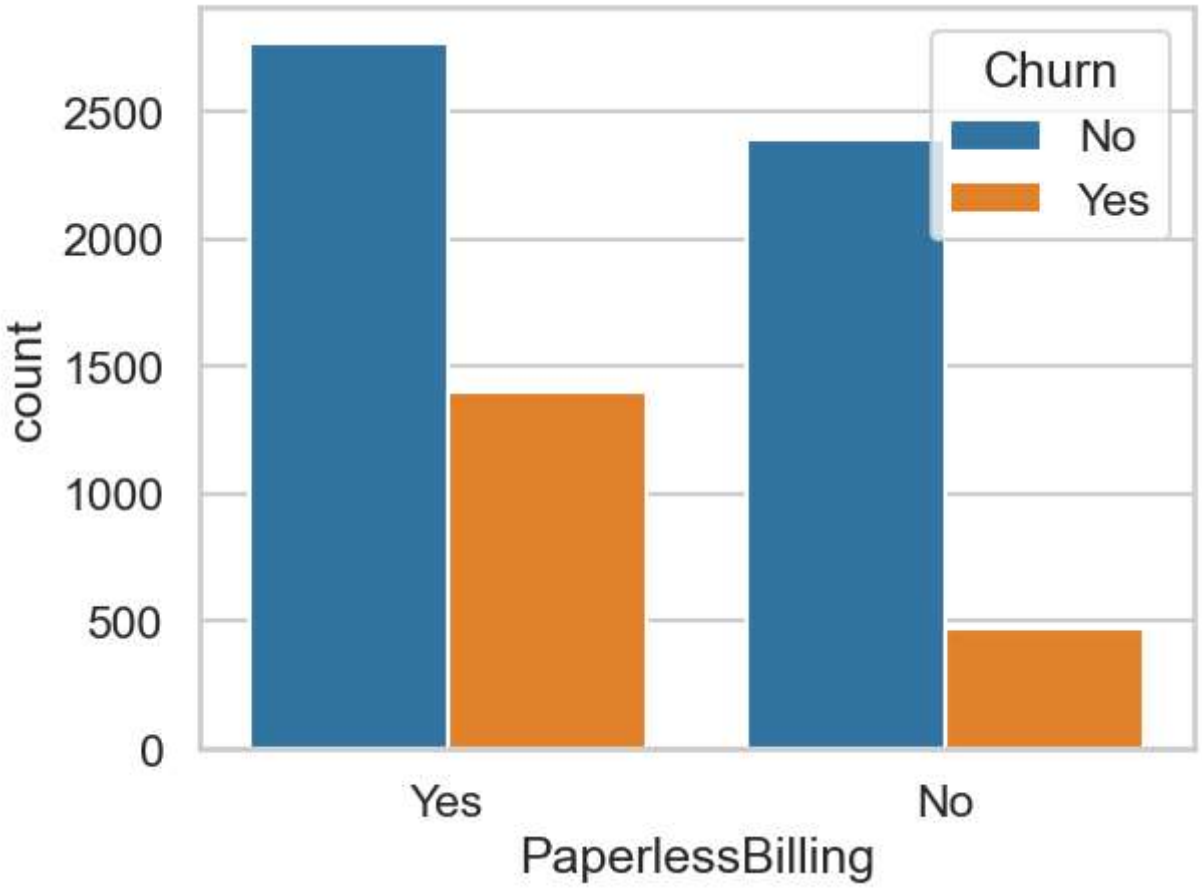

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection

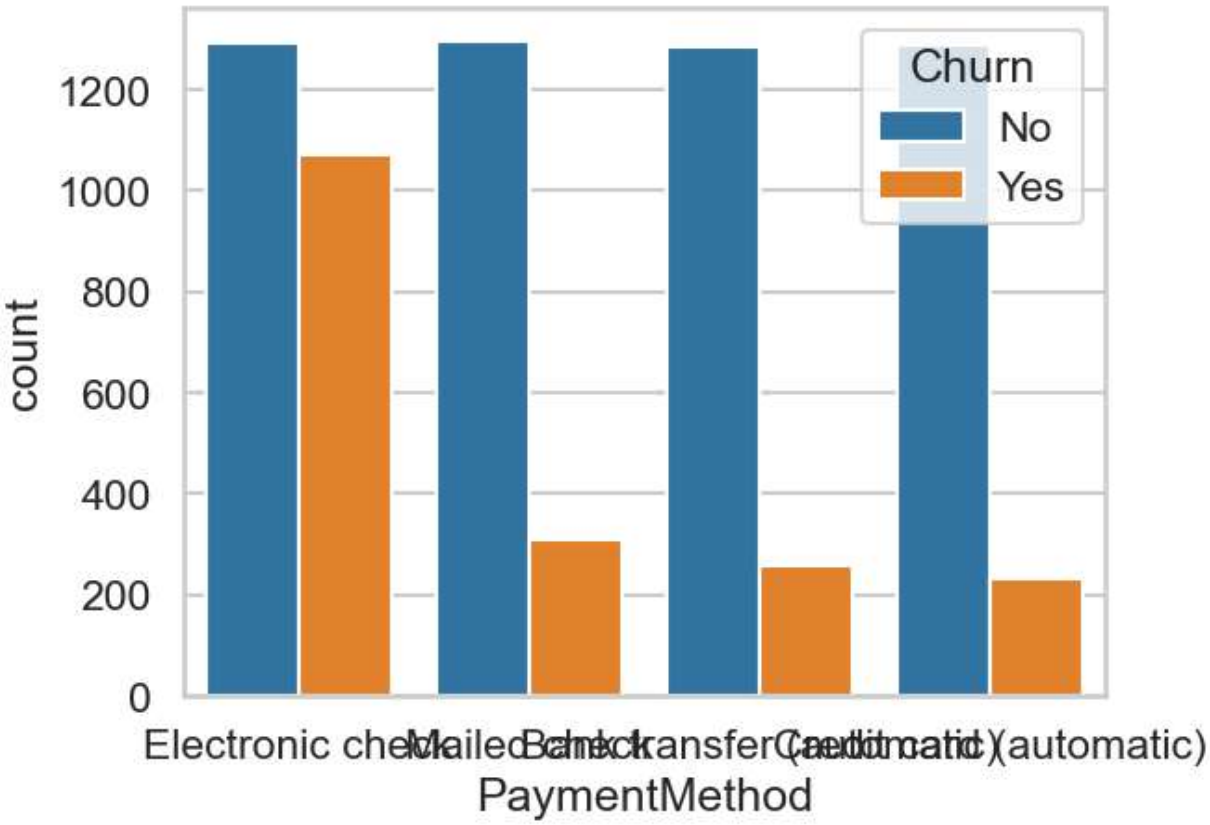

Churn
No
Yes
count
2000
1500
1000
500
0
No
Yes
No internet service
TechSupport

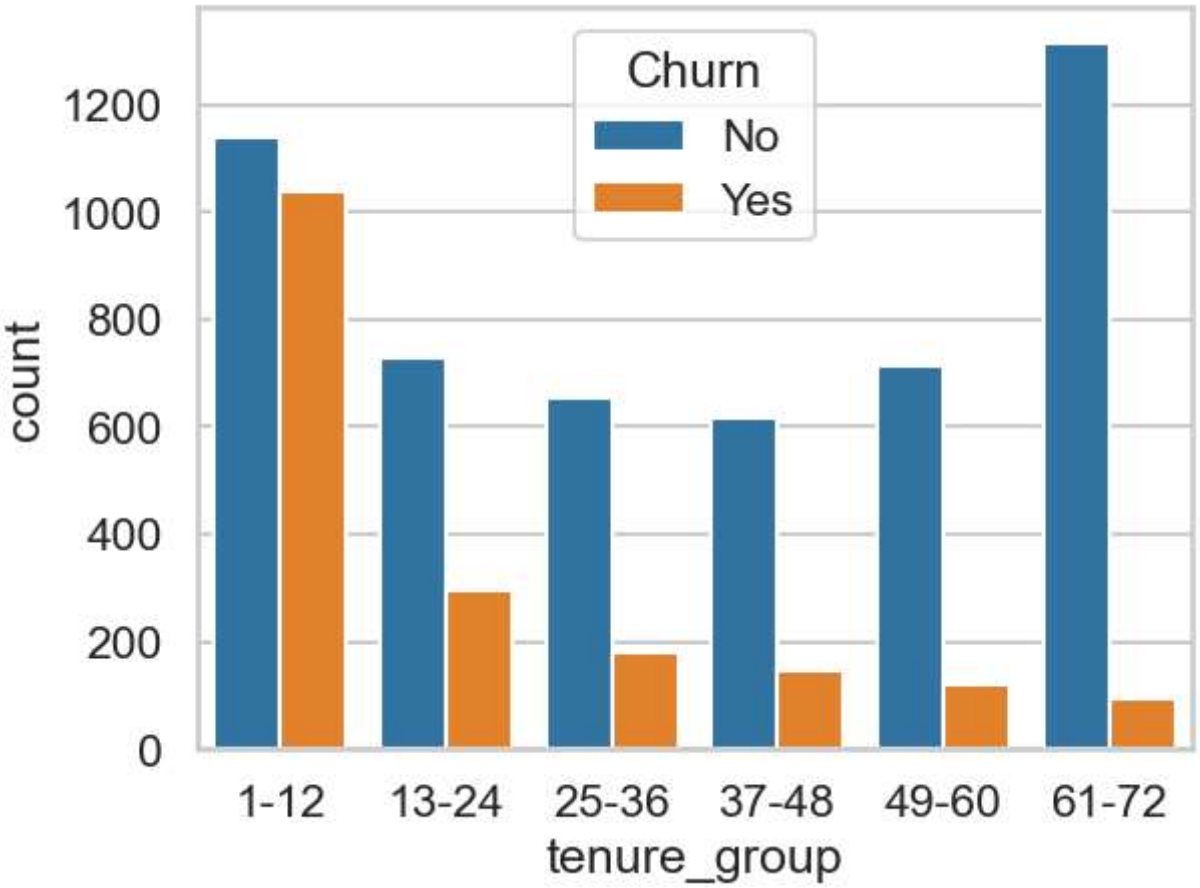

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingTV


2000
1500
1000
500
0
count
Churn
No
Yes
No
Yes
No internet service
StreamingMovies


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


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


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



1. Converting 'Churn' in a binary numeric variable i.e. Yes=1, No=0

In [65]:
```
telco_data['Churn'] = np.where(telco_data.Churn == 'Yes',1,0)
```

In [66]:
```
telco_data.head()
```

Out[66]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineS
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	

1. Convert all categorial variables into dummy variables

In [67]:
```
telco_data_dummies = pd.get_dummies(telco_data)
telco_data_dummies.head()
```

Out[67]:

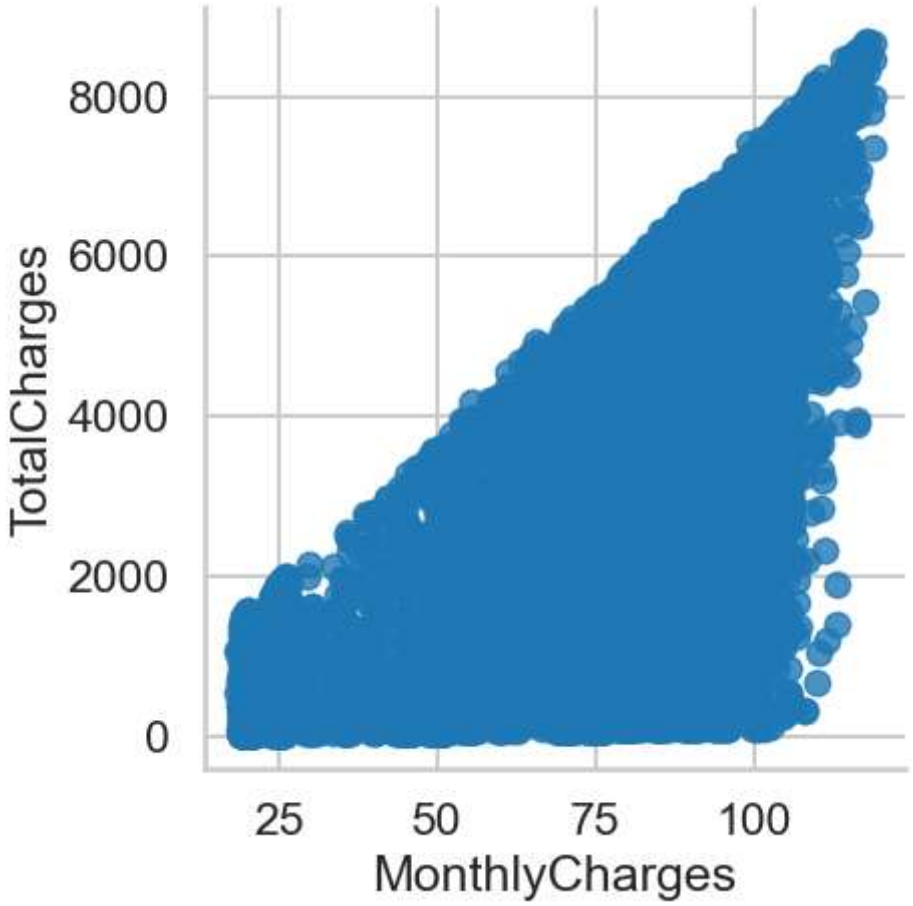
	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Pa
0	0	29.85	29.85	0	1	0	0	
1	0	56.95	1889.50	0	0	1	1	
2	0	53.85	108.15	1	0	1	1	
3	0	42.30	1840.75	0	0	1	1	
4	0	70.70	151.65	1	1	0	1	

5 rows × 51 columns

1. Relationship between Monthly Charges and Total Charges

In [68]:
```
sns.lmplot(data=telco_data_dummies, x='MonthlyCharges', y='TotalCharges', fit_reg=Fals
```

Out[68]:
```
<seaborn.axisgrid.FacetGrid at 0x212634eda50>
```



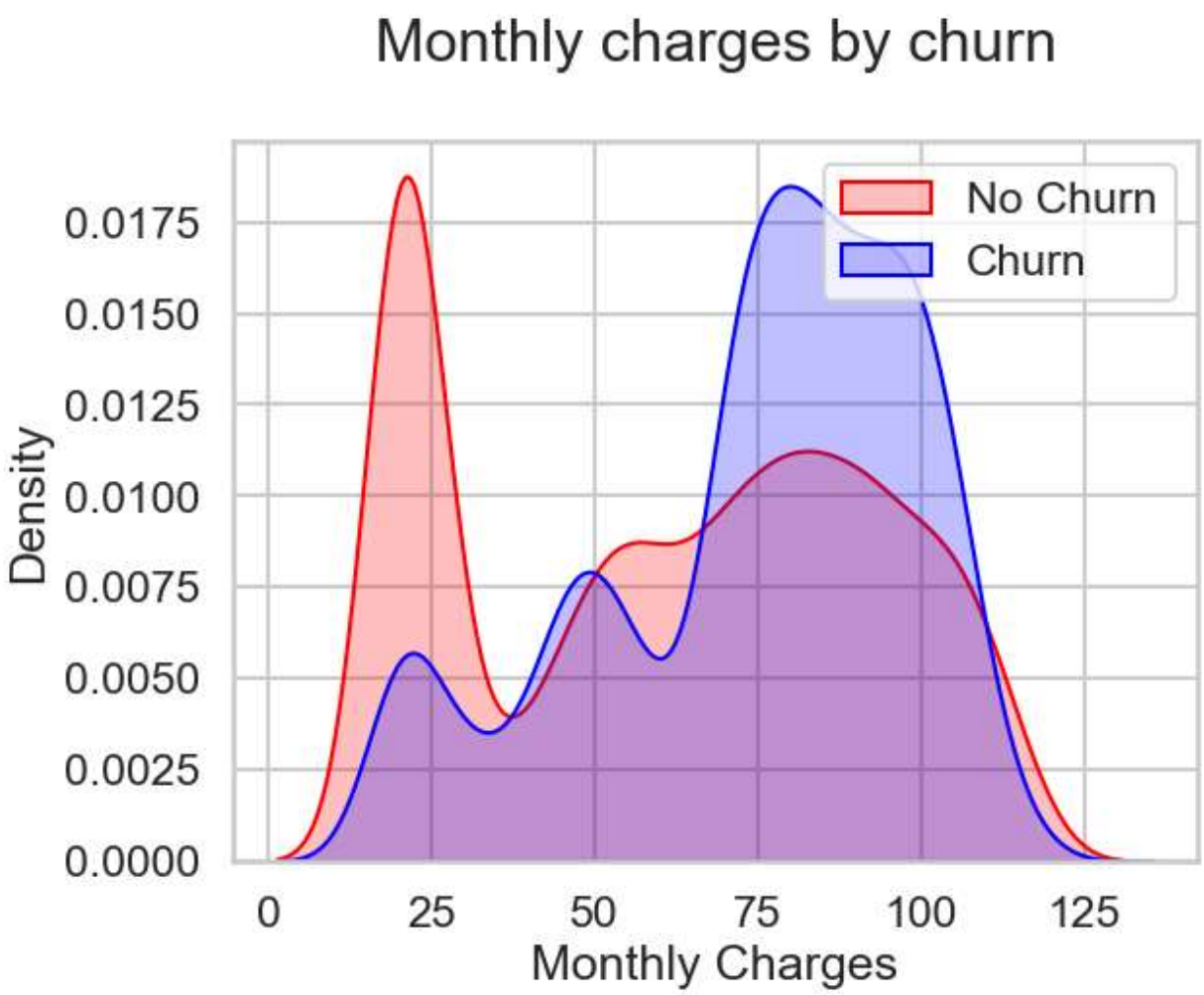
Total charges increase as monthly charges increase.

1. Churn by Monthly charges and Total charges

In [69]:
```
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"] == 0)
                color="Red", shade = True)
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"] == 1)
                ax =Mth, color="Blue", shade= True)
Mth.legend(["No Churn","Churn"],loc='upper right')
Mth.set_ylabel('Density')
Mth.set_xlabel('Monthly Charges')
Mth.set_title('Monthly charges by churn')
```

```
C:\Users\asus\AppData\Local\Temp\ipykernel_4588\722082952.py:1: FutureWarning:

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

  Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"] ==
0) ],
C:\Users\asus\AppData\Local\Temp\ipykernel_4588\722082952.py:3: FutureWarning:

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

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

Out[69]:
```
Text(0.5, 1.0, 'Monthly charges by churn')
```

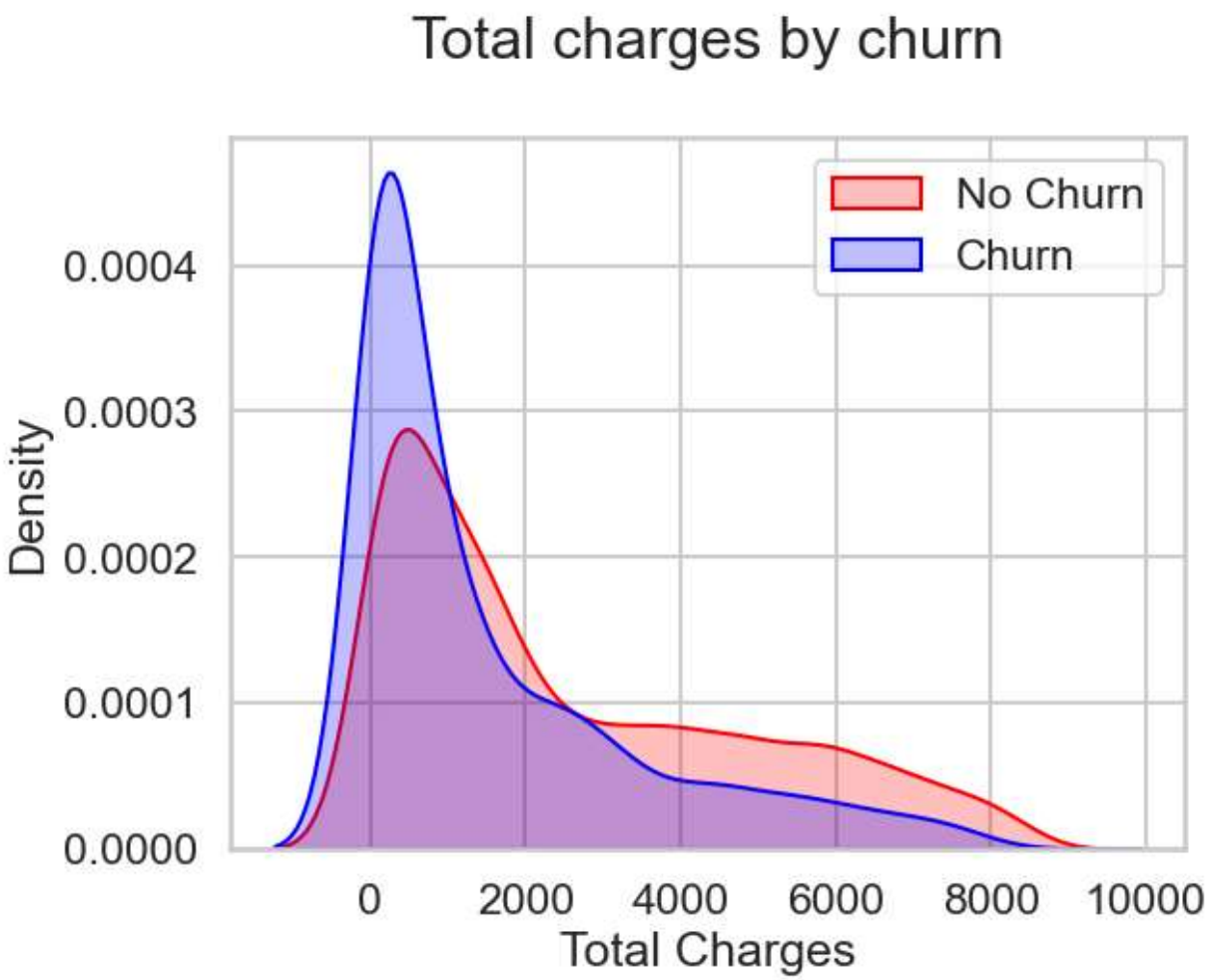


Insight: Churn is high when monthly charges are high.

In [70]:
```
Toc = sns.kdeplot(telco_data_dummies.TotalCharges[telco_data_dummies["Churn"]==0],colo
Toc = sns.kdeplot(telco_data_dummies.TotalCharges[telco_data_dummies["Churn"]==1],ax=
Toc.legend(['No Churn', 'Churn'], loc='upper right')
Toc.set_xlabel('Total Charges')
Toc.set_ylabel('Density')
Toc.set_title('Total charges by churn')
```

```
C:\Users\asus\AppData\Local\Temp\ipykernel_4588\1995776560.py:1: FutureWarning:

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

  Toc = sns.kdeplot(telco_data_dummies.TotalCharges[telco_data_dummies["Churn"]==0],c
olor='Red', shade=True)
C:\Users\asus\AppData\Local\Temp\ipykernel_4588\1995776560.py:2: FutureWarning:

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

  Toc = sns.kdeplot(telco_data_dummies.TotalCharges[telco_data_dummies["Churn"]==1],a
x= Toc, color='Blue', shade=True)
```

Out[70]:
```
Text(0.5, 1.0, 'Total charges by churn')
```

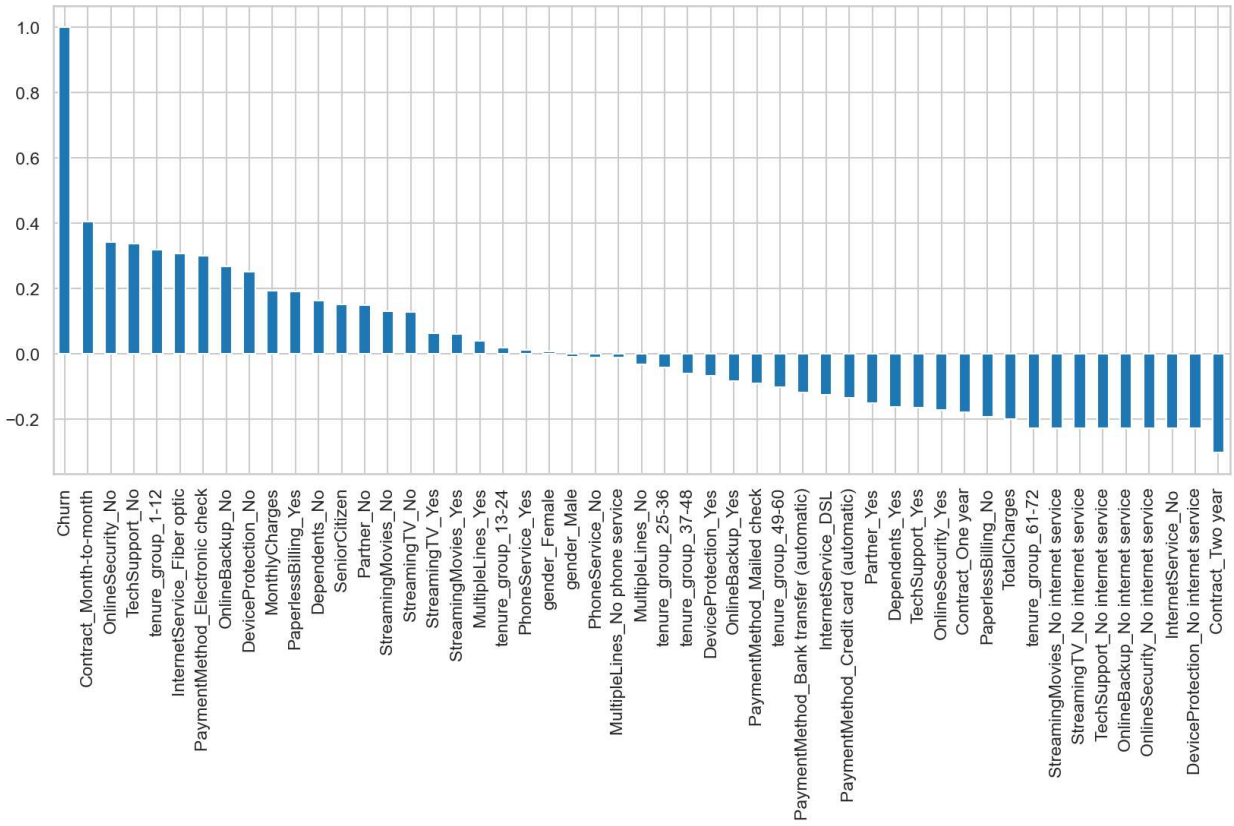


Insight: Higher churn at lower Total Charges

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

In [71]:
```
plt.figure(figsize=(20,8))
telco_data_dummies.corr()['Churn'].sort_values(ascending=False).plot(kind='bar')
```

Out[71]:
```
<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

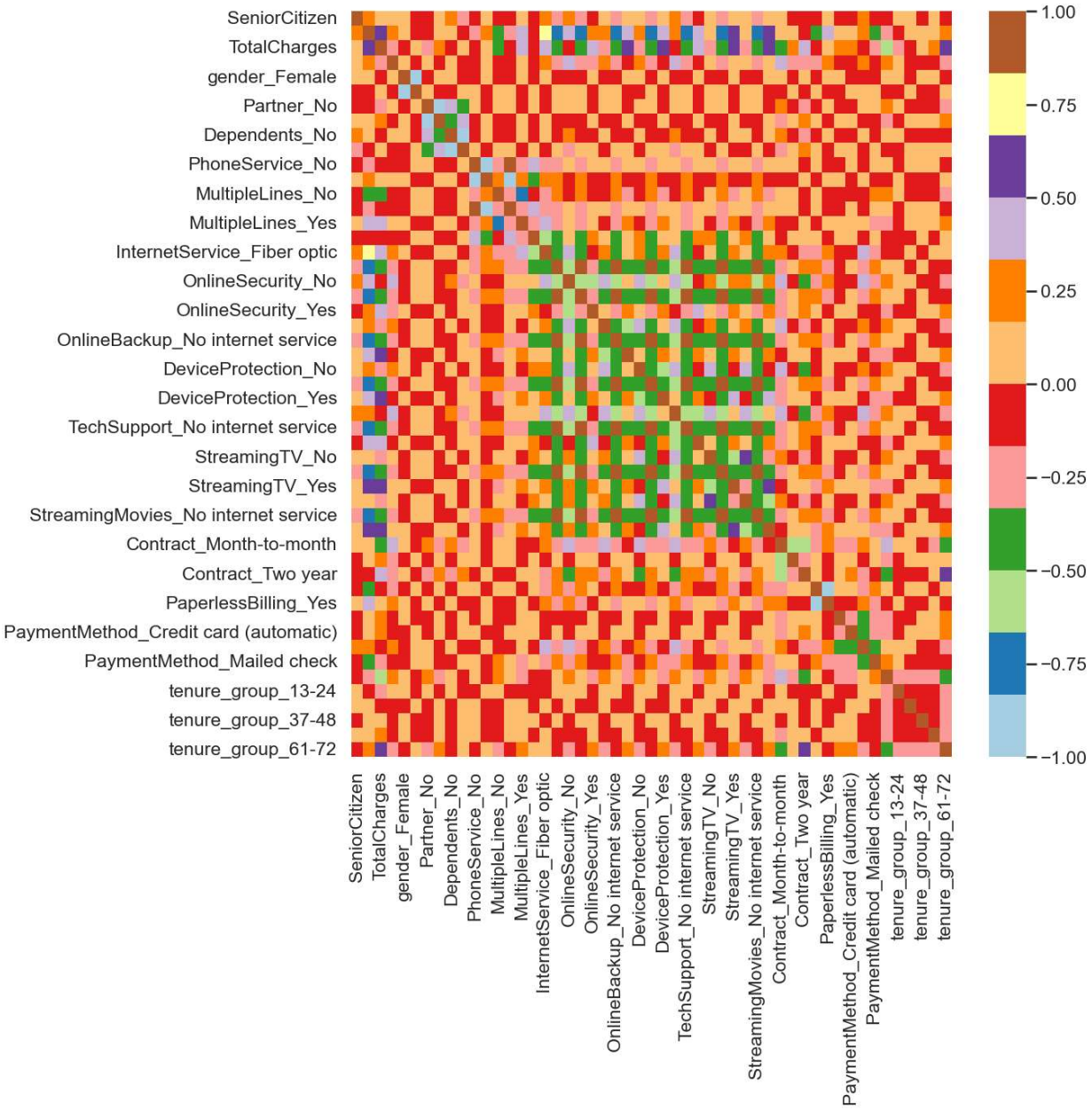
LOW Churn is seens in case of Long term contracts, Subscriptions without internet service and The customers engaged for 5+ years

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 [72]:
```
plt.figure(figsize=(12,12))
sns.heatmap(telco_data_dummies.corr(), cmap="Paired")
```

Out[72]:
```
<Axes: >
```



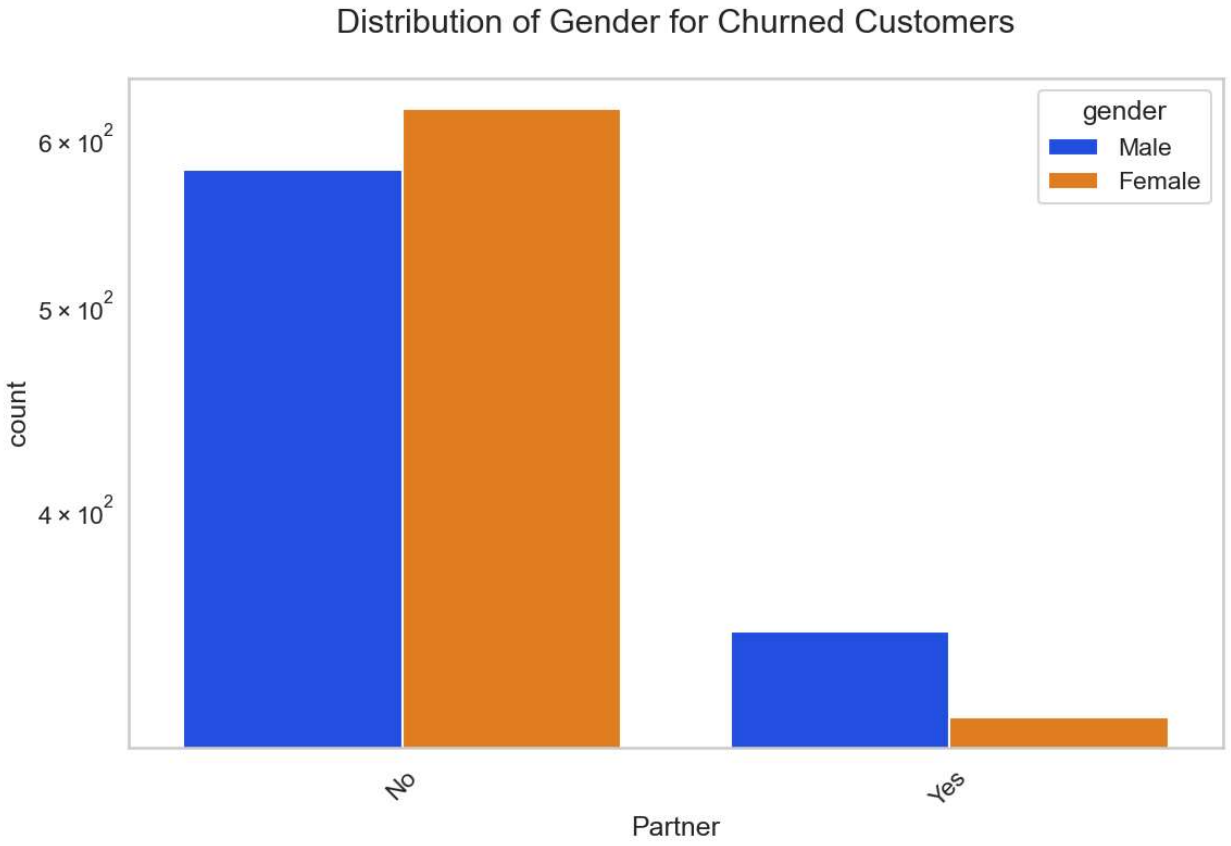
Bivariate Analysis

In [73]:
```
new_df1_target0=telco_data.loc[telco_data["Churn"]==0]
new_df1_target1=telco_data.loc[telco_data["Churn"]==1]
```

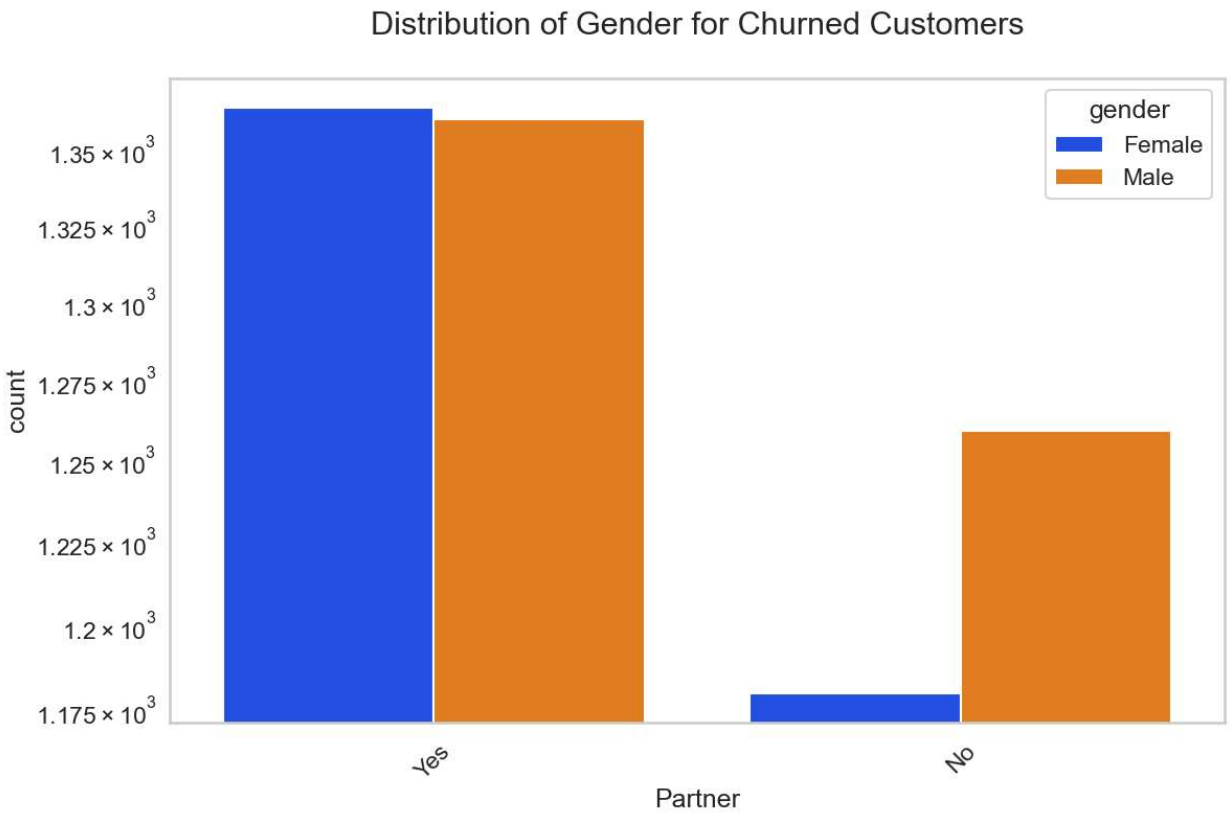
In [74]:
```
def uniplot(df,col,title,hue=None):

    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=hue, p

    plt.show()
```

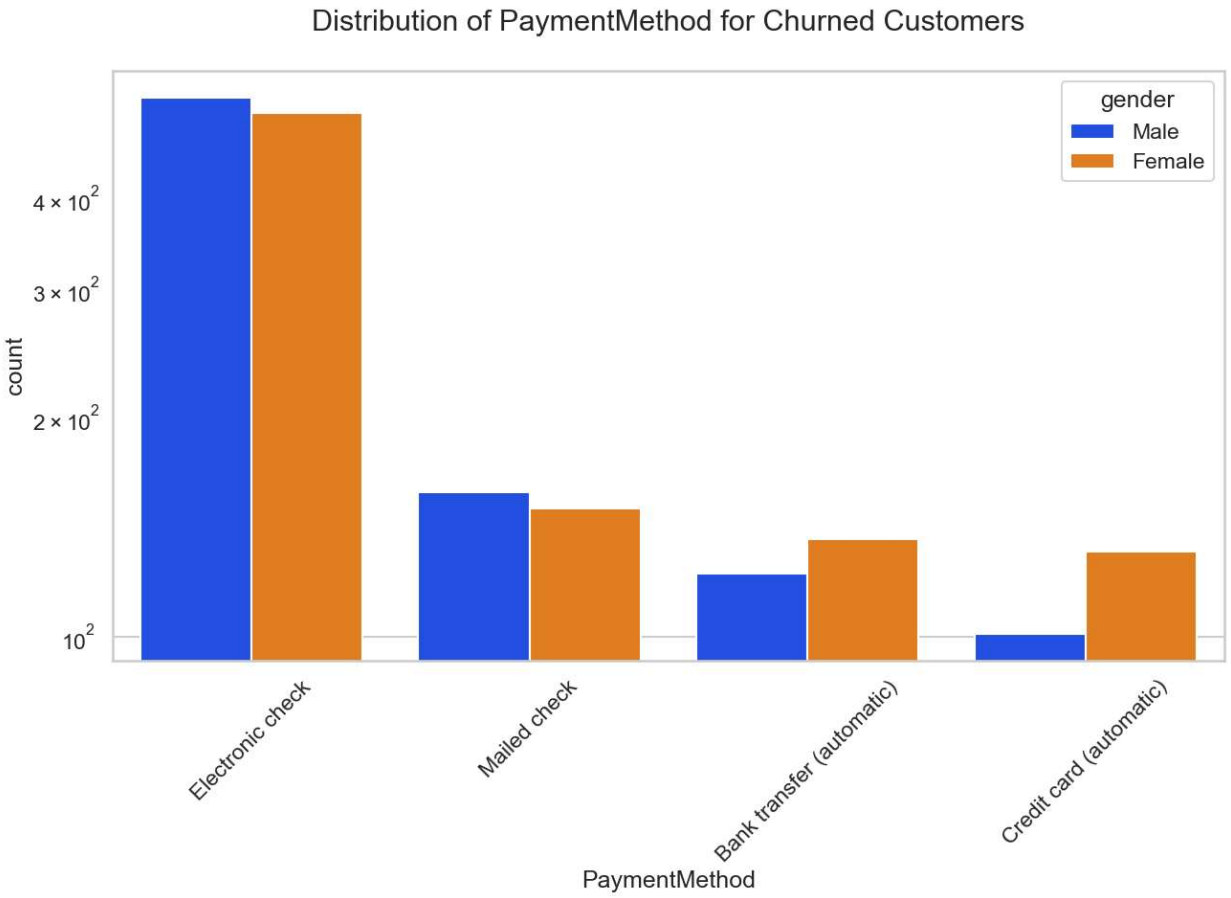
In [75]:
```
uniplot(new_df1_target1,col='Partner',title='Distribution of Gender for Churned Custom
```



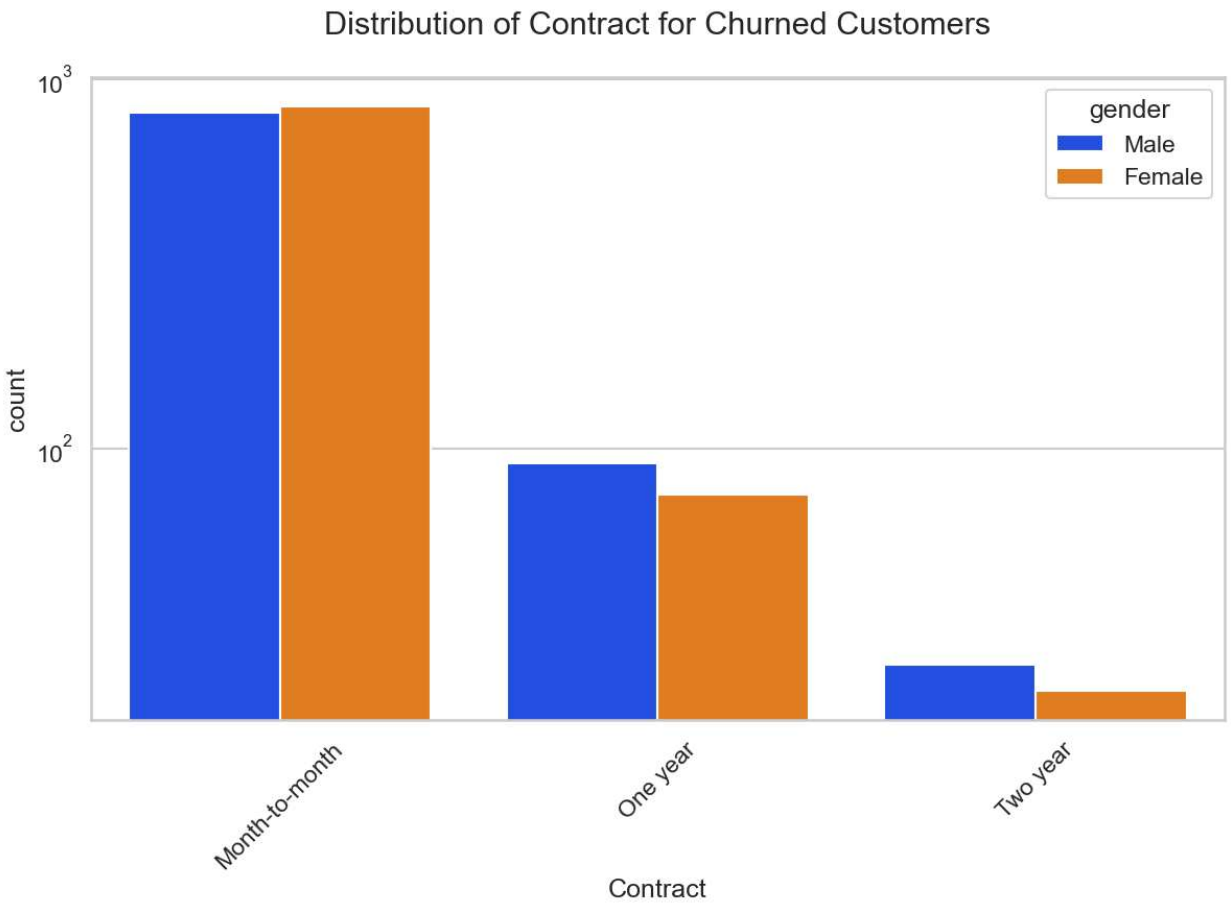
In [76]:
```
uniplot(new_df1_target0,col='Partner',title='Distribution of Gender for Churned Custom
```



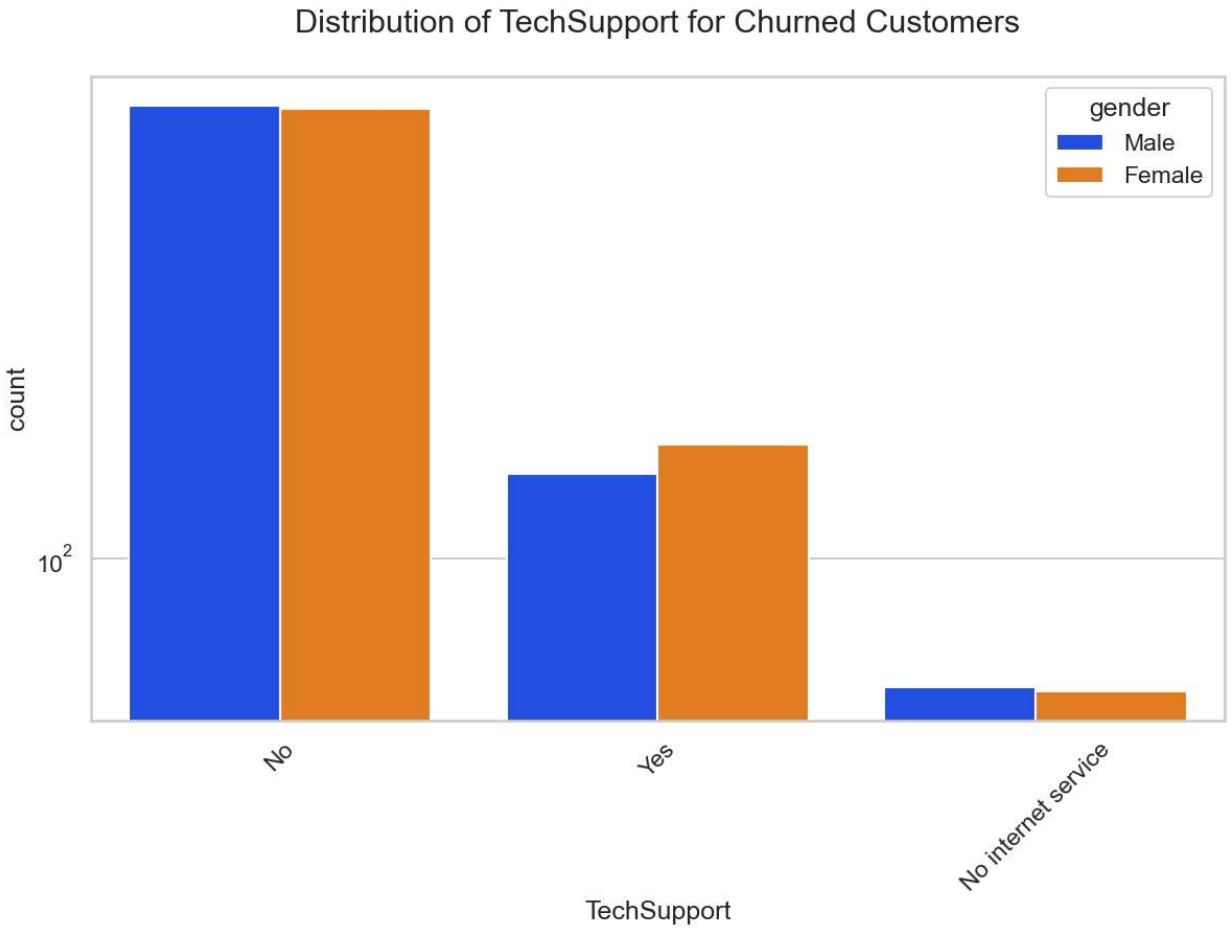
In [77]:
```
uniplot(new_df1_target1,col='PaymentMethod',title='Distribution of PaymentMethod for C
```



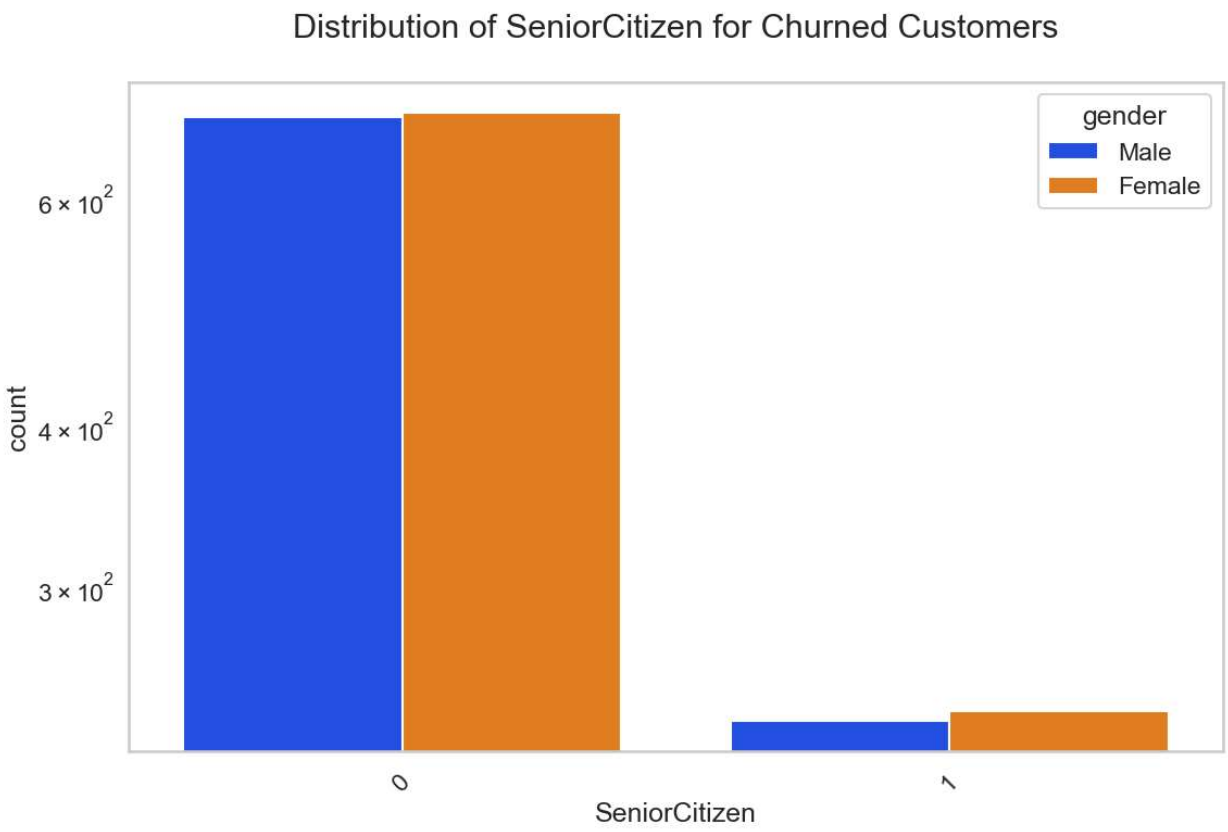
In [78]:
```
uniplot(new_df1_target1,col='Contract',title='Distribution of Contract for Churned Cus
```



In [79]:
```
uniplot(new_df1_target1,col='TechSupport',title='Distribution of TechSupport for Churr
```



In [80]:
```
uniplot(new_df1_target1,col='SeniorCitizen',title='Distribution of SeniorCitizen for C
```



Conclusion

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

In [81]:
```
telco_data_dummies.to_csv('tel_churn.csv')
```

Model Building

In [1]:
```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score, classification_report, confusion_matrix,Con
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
```

In [2]:
```
df=pd.read_csv('tel_churn.csv')
```

In [3]:
```
df.head(5)
```

Out[3]:

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

5 rows × 52 columns

In [4]:
```
df=df.drop('Unnamed: 0',axis=1)
```

In [5]:
```
df.head(5)
```

Out[5]:

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

5 rows × 51 columns

In [6]:
```
# Creating x and y variable
y=df['Churn']
y.head()
```

Out[6]:
```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

In [7]:
```
x=df.drop(['Churn'], axis=1)
x.head()
```

Out[7]:

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

5 rows × 50 columns

Train Test Split

In [8]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

Decision Tree Classifier

In [9]:
```
model_dt=DecisionTreeClassifier(criterion="gini",random_state=100,max_depth=6, min_sar
```

In [10]:
```
model_dt.fit(x_train,y_train)
```

Out[10]:
```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [11]:
```
y_pred=model_dt.predict(x_test)
y_pred
```

Out[11]:
```
array([0, 1, 0, ..., 1, 0, 0], dtype=int64)
```

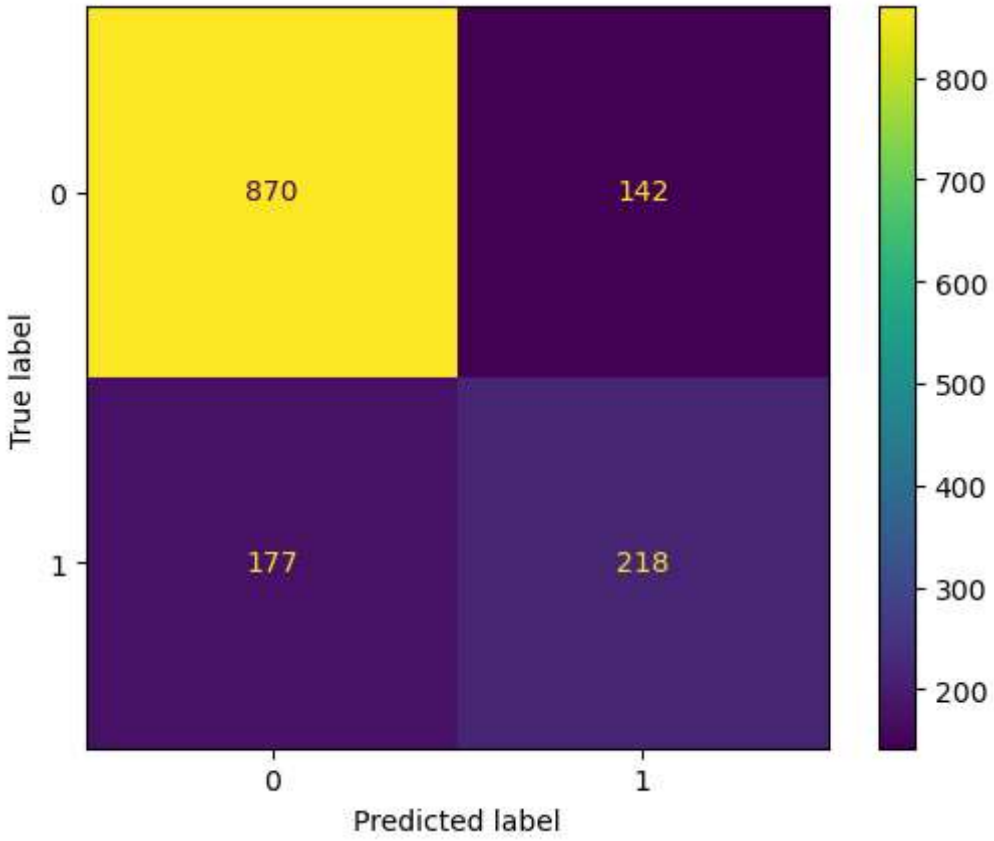
In [12]:
```
model_dt.score(x_test,y_test)
```

Out[12]:
```
0.7732764747690121
```

In [13]:
```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

	precision	recall	f1-score	support
0	0.83	0.86	0.85	1012
1	0.61	0.55	0.58	395
accuracy			0.77	1407
macro avg	0.72	0.71	0.71	1407
weighted avg	0.77	0.77	0.77	1407

In [14]:
```
cm1 = confusion_matrix(y_test, y_pred, labels=model_dt.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm1, display_labels=model_dt.classes_)
disp.plot()
plt.show()
```



As you can see that the accuracy is quite low, and as it's an imbalanced dataset, we shouldn't consider Accuracy as our metrics to measure the model, as Accuracy is cursed in imbalanced datasets.

Hence, we need to check recall, precision & f1 score for the minority class, and it's quite evident that the precision, recall & f1 score is too low for Class 1, i.e. churned customers.

Hence, moving ahead to call SMOTEENN (UpSampling + ENN)

In [15]:
```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(x,y)
```

In [16]:
```
xr_train,xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_size=
```

In [17]:
```
model_dt_smote=DecisionTreeClassifier(criterion="gini", random_state=100, max_depth=6,
```

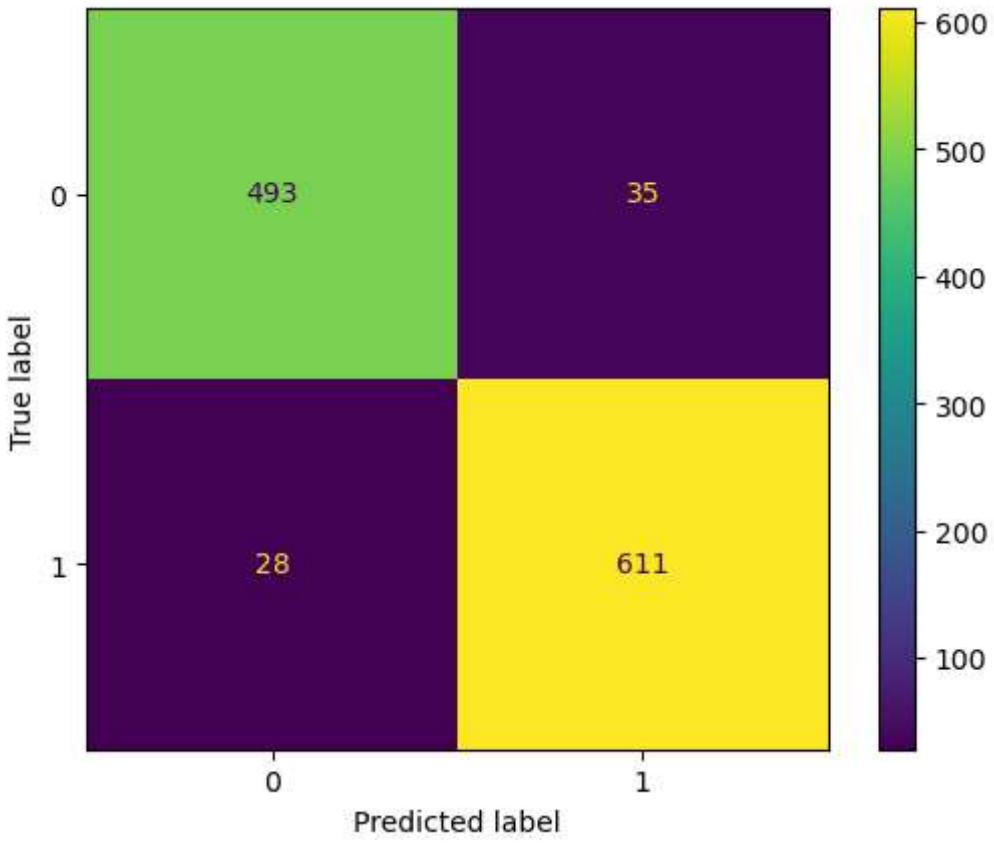
In [18]:
```
model_dt_smote.fit(xr_train,yr_train)
```

Out[18]:
```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [19]:
```
yr_pred=model_dt_smote.predict(xr_test)
```

In [20]:
```
model_score_r = model_dt_smote.score(xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_pred))
cm = confusion_matrix(yr_test, yr_pred, labels=model_dt_smote.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_dt_smote.class
disp.plot()
plt.show()
```

```
0.9460154241645244
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       528
           1       0.95      0.96      0.95       639

    accuracy                           0.95      1167
   macro avg       0.95      0.94      0.95      1167
weighted avg       0.95      0.95      0.95      1167
```



Now we can see quite better results, i.e. Accuracy: 94 %, and a very good recall, precision & f1 score for minority class.

Let's try with some other classifier.

Random Forest Classifier

In [28]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [29]:
```
model_rf=RandomForestClassifier(n_estimators=100, criterion="gini",random_state=100, r
```

In [30]:
```
model_rf.fit(x_train,y_train)
```

Out[30]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [31]:
```
y_pred=model_rf.predict(x_test)
```

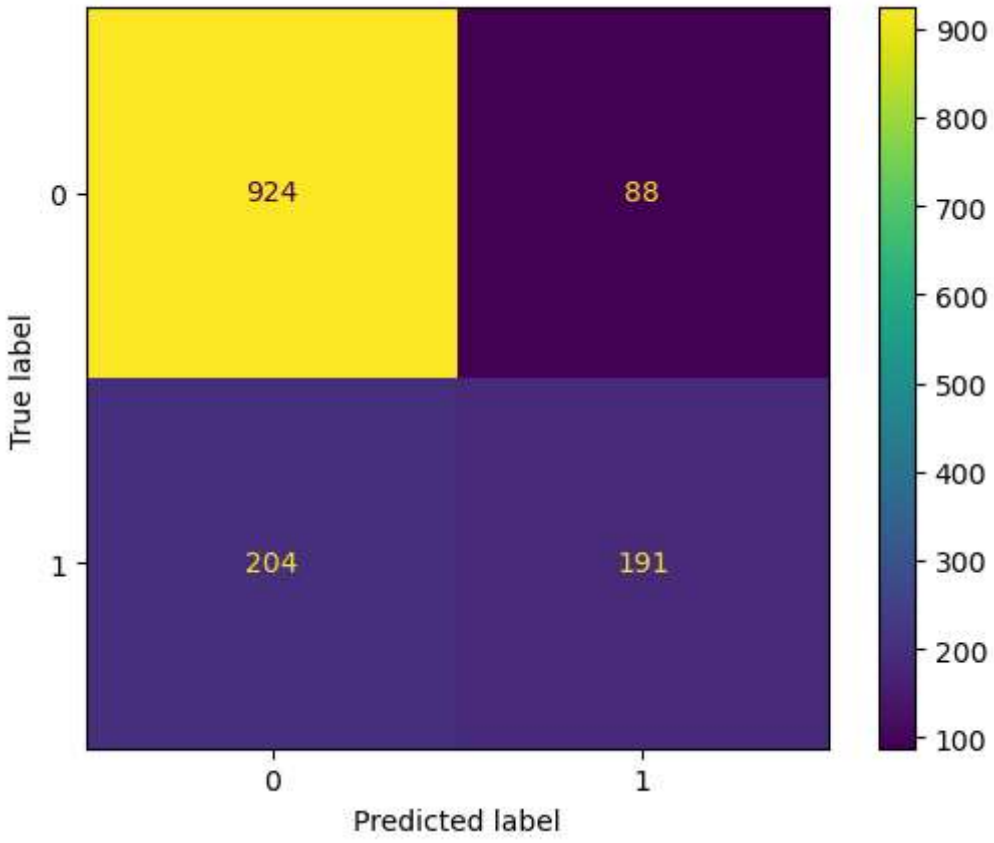
In [44]:
```
model_rf.score(x_test,y_test)
```

Out[44]:
```
0.7924662402274343
```

In [46]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1012
           1       0.68      0.48      0.57       395

    accuracy                           0.79      1407
   macro avg       0.75      0.70      0.72      1407
weighted avg       0.78      0.79      0.78      1407
```

In [47]:
```
cm = confusion_matrix(y_test, y_pred, labels=model_rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_rf.classes_)
disp.plot()
plt.show()
```



In [48]:
```
sm=SMOTEENN()
X_resampled1, y_resampled1 = sm.fit_resample(x,y)
```

In [49]:
```
xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test
```

In [50]:
```
yr_train1.head()
```

Out[50]:
```
3484    1
1710    0
200     0
183     0
2073    0
Name: Churn, dtype: int64
```

In [51]:
```
model_rf_smote=RandomForestClassifier(n_estimators=100,criterion="gini",random_state=1
```

In [52]:
```
model_rf_smote.fit(xr_train,yr_train)
```

Out[52]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [53]:
```
yr_pred1=model_rf_smote.predict(xr_test1)
```

In [54]:
```
model_score_r1=model_rf_smote.score(xr_test1, yr_test1)
```

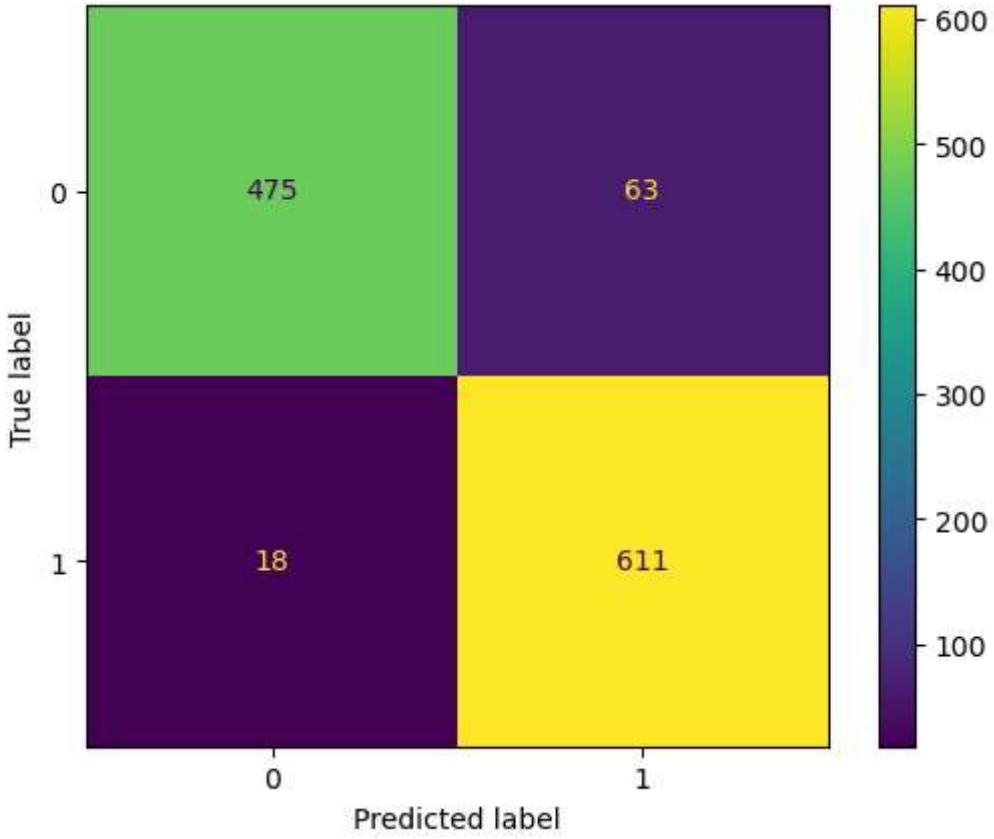
In [55]:
```
print(model_score_r1)
print(metrics.classification_report(yr_test1, yr_pred1, labels=[0,1]))
```

```
0.9305912596401028
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       538
           1       0.91      0.97      0.94       629

    accuracy                           0.93      1167
   macro avg       0.94      0.93      0.93      1167
weighted avg       0.93      0.93      0.93      1167
```

In [56]:
```
print(metrics.confusion_matrix(yr_test1, yr_pred1))
```

```
[[475  63]
 [ 18 611]]
```

In [57]:
```
cm = confusion_matrix(yr_test1, yr_pred1, labels=model_rf_smote.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_rf_smote.class
disp.plot()
plt.show()
```



With RF Classifier, also we are able to get quite good results, infact better than Decision Tree.

PCA

In [58]:
```
from sklearn.decomposition import PCA
pca=PCA(0.9)
xr_train_pca = pca.fit_transform(xr_train1)
xr_test_pca = pca.transform(xr_test1)
explained_variance = pca.explained_variance_ratio_
```

In [59]:
```
model=RandomForestClassifier(n_estimators=100, criterion='gini', random_state =100, ma
```

In [60]:
```
model.fit(xr_train_pca,yr_train1)
```

Out[60]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

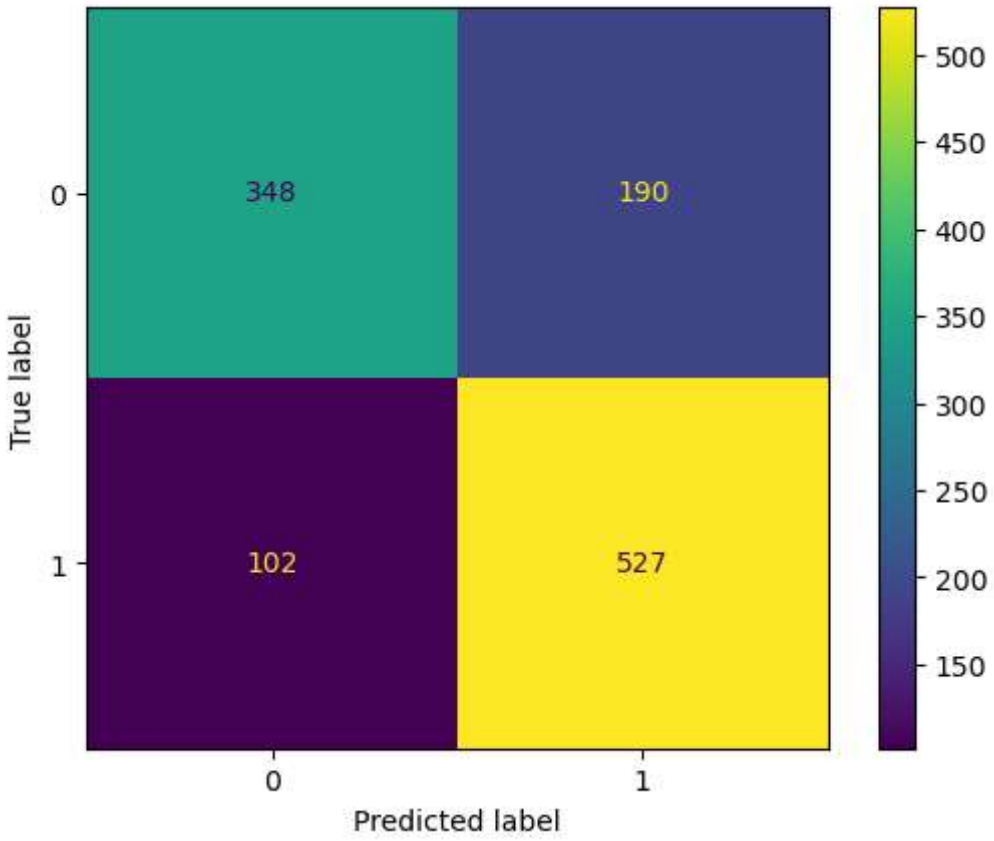
In [61]:
```
yr_predict_pca = model.predict(xr_test_pca)
```

In [62]:
```
model_score_r_pca = model.score(xr_test_pca, yr_test1)
```

In [63]:
```
print(model_score_r_pca)
print(metrics.classification_report(yr_test1, yr_predict_pca))
```

```
0.7497857754927164
              precision    recall  f1-score   support

           0       0.77      0.65      0.70       538
           1       0.74      0.84      0.78       629

    accuracy                           0.75      1167
   macro avg       0.75      0.74      0.74      1167
weighted avg       0.75      0.75      0.75      1167
```

In [64]:
```
cm = confusion_matrix(yr_test1, yr_predict_pca, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot()
plt.show()
```



With PCA, we couldn't see any better results, hence let's finalise the model which was created by RF Classifier, and save the model so that we can use it in a later stage

Pickling the model

In [60]:
```
import pickle
```

In [61]:
```
filename = 'model.sav'
```

In [62]:
```
pickle.dump(model_rf_smote, open(filename, 'wb'))
```

In [64]:
```
load_model= pickle.load(open(filename,'rb'))
```

In [65]:
```
model_score_r1 = load_model.score(xr_test1, yr_test1)
```

In [66]:
```
model_score_r1
```

Out[66]:
```
0.945531914893617
```

Our final model i.e. RF Classifier with SMOTEENN, is now ready and dumped in model.sav, which we will use and prepare API's so that we can access our model from UI.

In [ ]:
```

```