In [2]:

```
pwd
```

Out[2]:

```
'C:\\Users\\Acer'
```

In [3]:

```
! pip install xgboost
```

```
Requirement already satisfied: xgboost in d:\python\lib\site-packages (1.5.
2)
Requirement already satisfied: numpy in d:\python\lib\site-packages (from xg
boost) (1.19.5)
Requirement already satisfied: scipy in d:\python\lib\site-packages (from xg
boost) (1.5.2)
```

In [4]:

```
! pip install dtreeviz
```

```
Requirement already satisfied: dtreeviz in d:\python\lib\site-packages (1.3.
3)
Requirement already satisfied: matplotlib in d:\python\lib\site-packages (fr
om dtreeviz) (3.3.2)
Requirement already satisfied: graphviz>=0.9 in d:\python\lib\site-packages
(from dtreeviz) (0.19.1)
Requirement already satisfied: pytest in d:\python\lib\site-packages (from d
treeviz) (0.0.0)
Requirement already satisfied: scikit-learn in d:\python\lib\site-packages
(from dtreeviz) (0.24.2)
Requirement already satisfied: pandas in d:\python\lib\site-packages (from d
treeviz) (1.3.5)
Requirement already satisfied: numpy in d:\python\lib\site-packages (from dt
reeviz) (1.19.5)
Requirement already satisfied: colour in d:\python\lib\site-packages (from d
treeviz) (0.1.5)
Requirement already satisfied: certifi>=2020.06.20 in d:\python\lib\site-pac
kages (from matplotlib->dtreeviz) (2021.5.30)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\acer\appdata
\roaming\python\python38\site-packages (from matplotlib->dtreeviz) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
d:\python\lib\site-packages (from matplotlib->dtreeviz) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\python\lib\site-packa
ges (from matplotlib->dtreeviz) (1.3.0)
Requirement already satisfied: pillow>=6.2.0 in d:\python\lib\site-packages
(from matplotlib->dtreeviz) (8.0.1)
Requirement already satisfied: cycler>=0.10 in d:\python\lib\site-packages
(from matplotlib->dtreeviz) (0.10.0)
Requirement already satisfied: attrs>=17.4.0 in d:\python\lib\site-packages
(from pytest->dtreeviz) (20.3.0)
Requirement already satisfied: iniconfig in d:\python\lib\site-packages (fro
m pytest->dtreeviz) (1.1.1)
Requirement already satisfied: packaging in d:\python\lib\site-packages (fro
m pytest->dtreeviz) (20.4)
Requirement already satisfied: pluggy<1.0,>=0.12 in d:\python\lib\site-packa
ges (from pytest->dtreeviz) (0.13.1)
Requirement already satisfied: py>=1.8.2 in d:\python\lib\site-packages (fro
m pytest->dtreeviz) (1.9.0)
Requirement already satisfied: toml in c:\users\acer\appdata\roaming\python
\python38\site-packages (from pytest->dtreeviz) (0.10.2)
Requirement already satisfied: atomicwrites>=1.0 in d:\python\lib\site-packa
ges (from pytest->dtreeviz) (1.4.0)
Requirement already satisfied: colorama in c:\users\acer\appdata\roaming\pyt
hon\python38\site-packages (from pytest->dtreeviz) (0.4.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\python\lib\site-pa
ckages (from scikit-learn->dtreeviz) (2.1.0)
Requirement already satisfied: scipy>=0.19.1 in d:\python\lib\site-packages
(from scikit-learn->dtreeviz) (1.5.2)
Requirement already satisfied: joblib>=0.11 in d:\python\lib\site-packages
(from scikit-learn->dtreeviz) (0.17.0)
Requirement already satisfied: pytz>=2017.3 in d:\python\lib\site-packages
(from pandas->dtreeviz) (2020.1)
Requirement already satisfied: six>=1.5 in c:\users\acer\appdata\roaming\pyt
hon\python38\site-packages (from python-dateutil>=2.1->matplotlib->dtreeviz)
(1.16.0)
```

In [5]:

```
from dtreeviz.trees import dtreeviz
```

In [6]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
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.over_sampling import SMOTE
from sklearn.metrics import fbeta_score
from numpy import loadtxt
from xgboost import XGBClassifier
```

In [7]:

```
telco_base_data = pd.read_csv('D:/Customer_churn_analysis/WA_Fn-UseC_-Telco-Customer-Churn.
```

In [8]:

```
telco_base_data.head()
```

Out[8]:

	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	Nc
2	3668-QPYBK	Male	0	No	No	2	Yes	Nc
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	Nc

5 rows × 21 columns

In [9]:

```
telco_base_data.shape
```

Out[9]:

```
(7043, 21)
```

In [10]:

```
telco_base_data.columns.values
```

Out[10]:

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

describe is used for numerical data

In [11]:

```
telco_base_data.dtypes
```

Out[11]:

```
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 [12]:

```
telco_base_data.describe()
```

Out[12]:

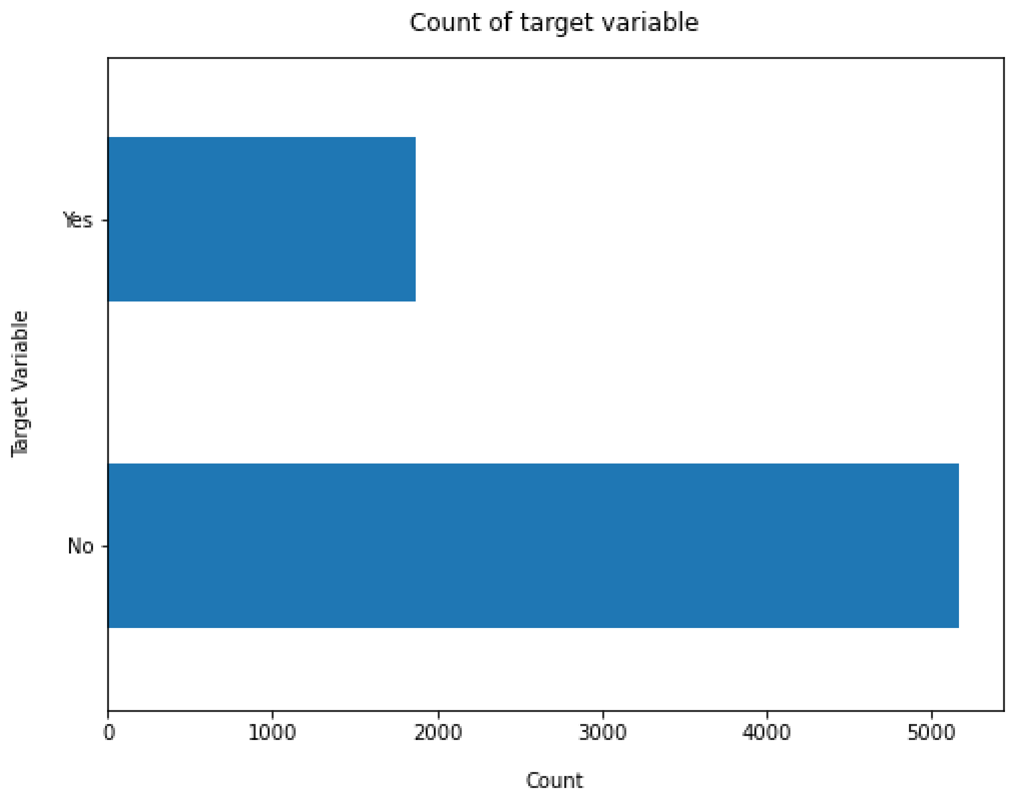
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

In [13]:

```
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",y=1.02)
```

Out[13]:

```
Text(0.5, 1.02, 'Count of target variable')
```



In [14]:

```
telco_base_data['Churn'].value_counts()
```

Out[14]:

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [15]:

```
100*telco_base_data['Churn'].value_counts()/len(telco_base_data['Churn'])
# Basically value counts is used to return unique values. So there are only 2 unique values
```

Out[15]:

```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

imbalanced dataset

In [16]:

```
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 [17]:

```
telco_data =telco_base_data.copy()
```

In [18]:

```
telco_data.TotalCharges = pd.to_numeric(telco_data.TotalCharges, errors='coerce')
telco_data.isnull().sum()
```

Out[18]:

```
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 [19]:

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

Out[19]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No ph ser
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	No ph ser
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

since the nuber 11 is quite negligible as compared to 7413 we can ignore them

In [20]:

```
telco_data.dropna(how = 'any', inplace = True)
# telco_data.fillna(0)
```

In [21]:

```
#creating Bins
```

In [22]:

```
print(telco_data['tenure'].max())
```

```
72
```

In [23]:

```
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, labels=l
```

In [24]:

```
telco_data['tenure_group'].value_counts()
```

Out[24]:

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

In [25]:

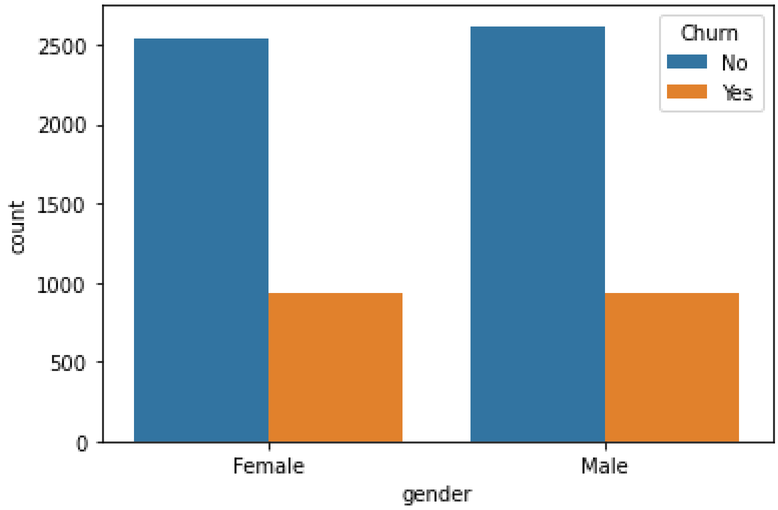
```
telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telco_data.head()
```

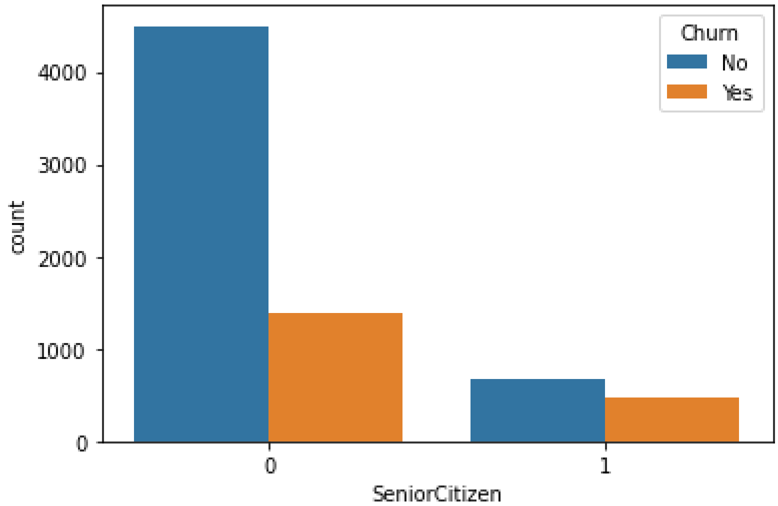
Out[25]:

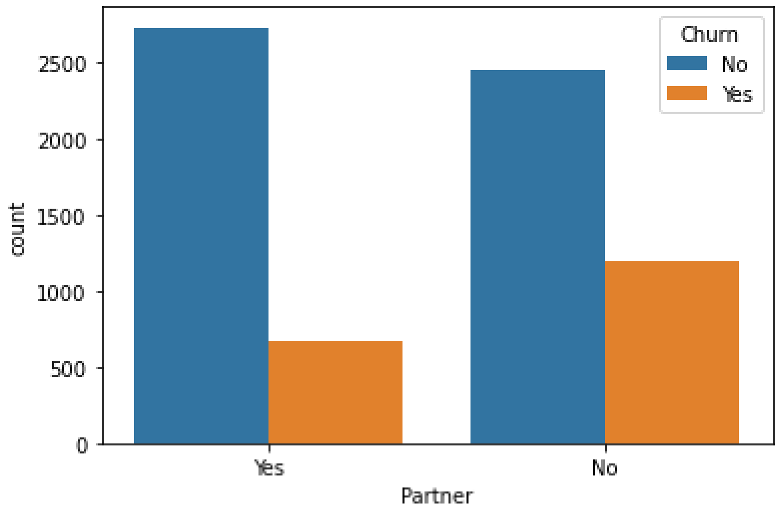
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	Or
0	Female	0	Yes	No	No	No phone service	DSL	
1	Male	0	No	No	Yes	No	DSL	
2	Male	0	No	No	Yes	No	DSL	
3	Male	0	No	No	No	No phone service	DSL	
4	Female	0	No	No	Yes	No	Fiber optic	

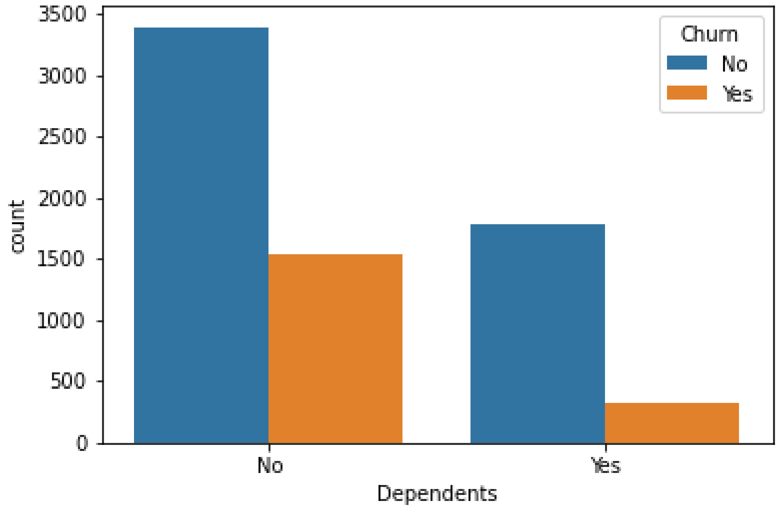
In [26]:

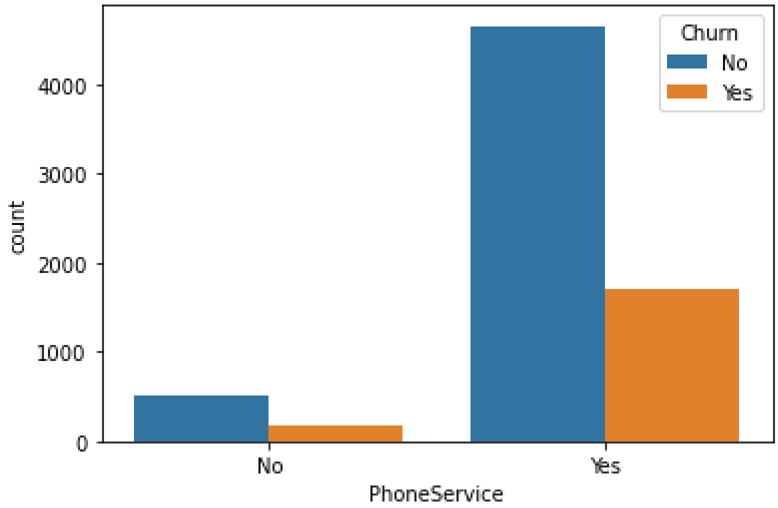
```
for i, predictor in enumerate(telco_data.drop(columns=['Churn','TotalCharges','MonthlyCharg
    plt.figure(i)
    sns.countplot(data=telco_data, x=predictor, hue='Churn')
```

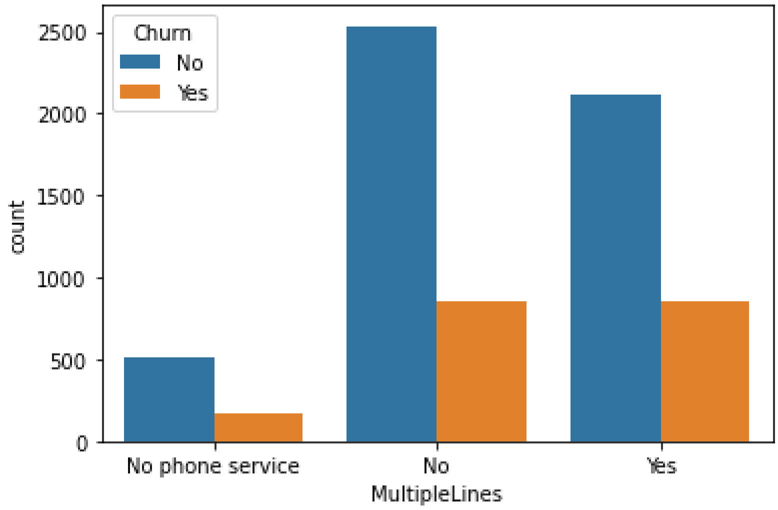


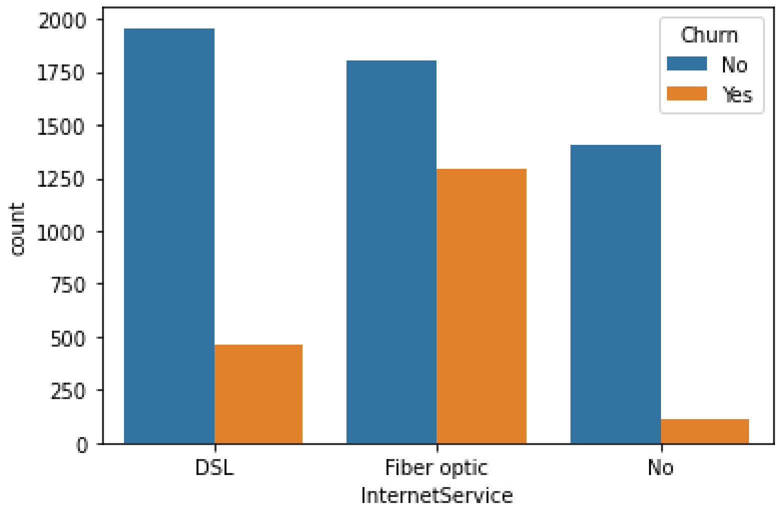


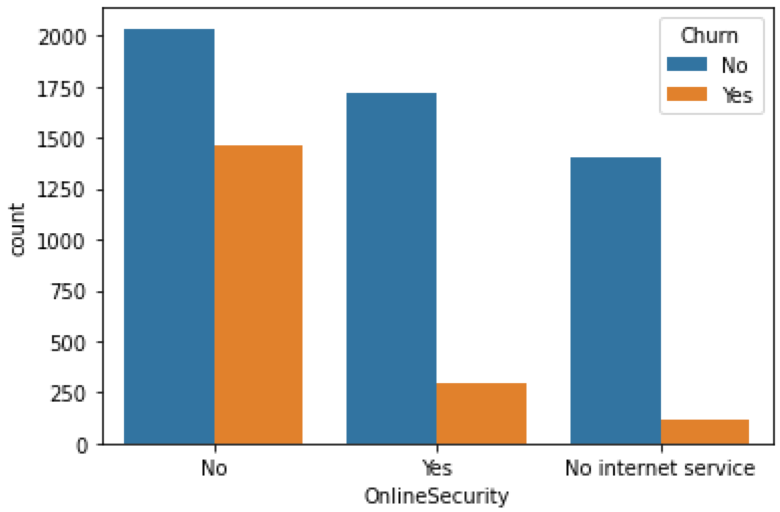


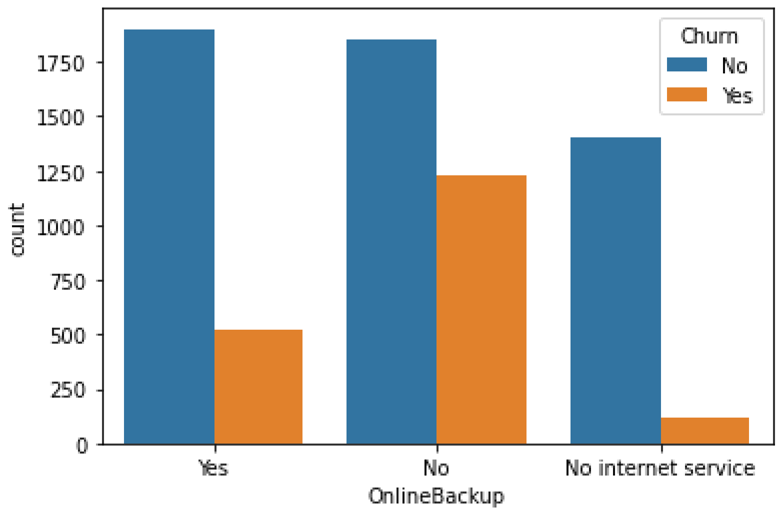

Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents


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

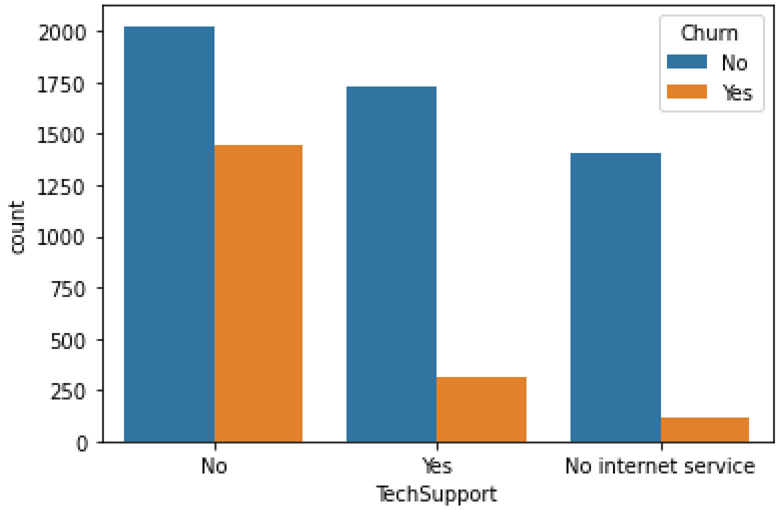

Churn
No
Yes
count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

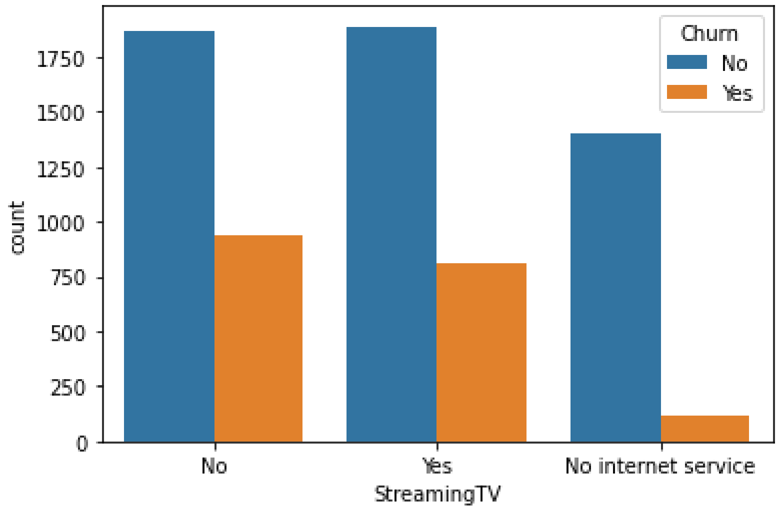

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

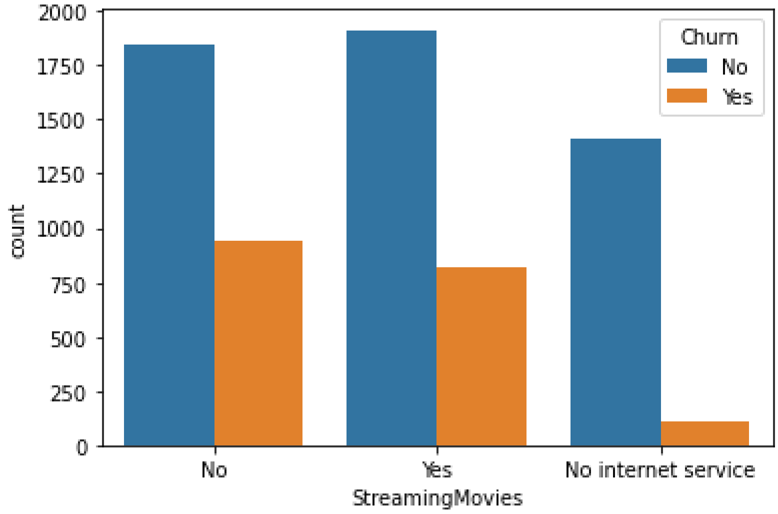

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

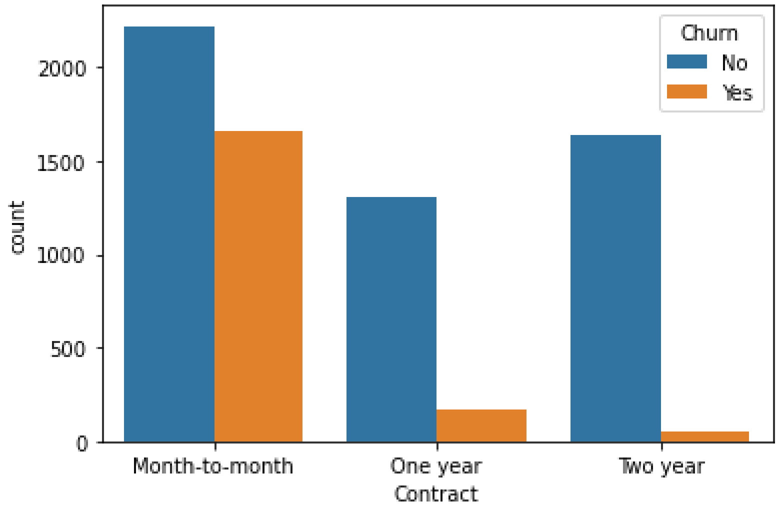

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

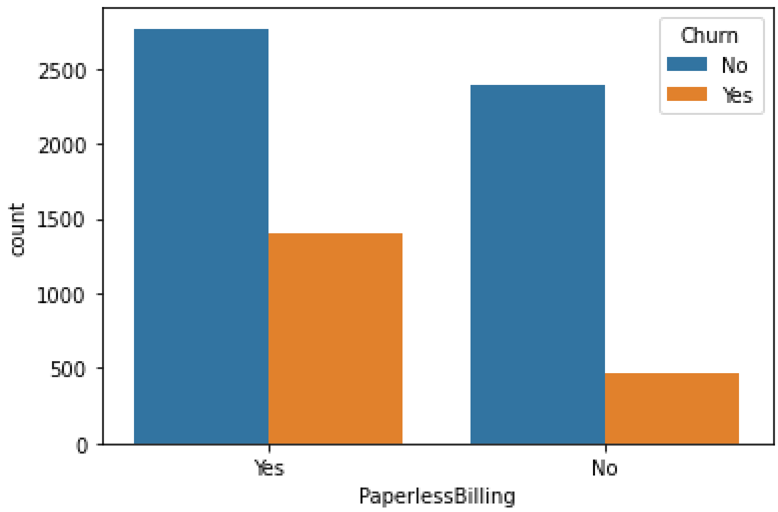

1750
Churn
No

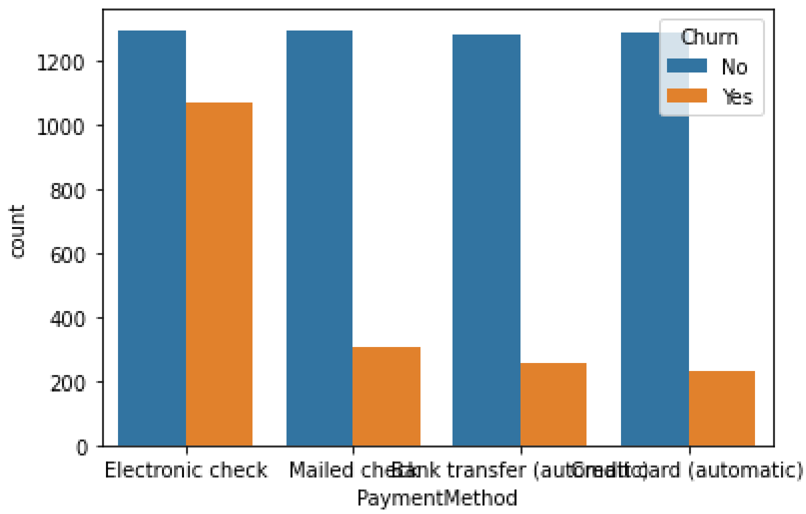

count
0
250
500
750
1000
1250
1500
1750
2000
Churn
No
Yes
No
Yes
No internet service
TechSupport

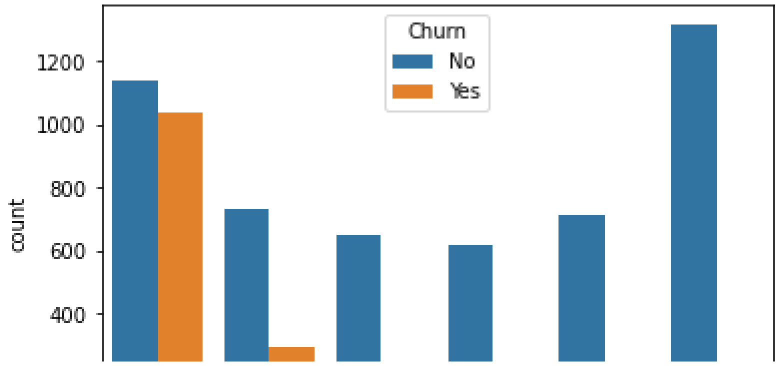

count
0
250
500
750
1000
1250
1500
1750
Churn
No
Yes
No
Yes
No internet service
StreamingTV


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


Churn
No
Yes
count
0
500
1000
1500
2000
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
PaymentMethod



convert The variable Churn in a binary numeric Value: Yes=1, No=0

In [27]:

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

In [28]:

```
telco_data.head()
```

Out[28]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	Or
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	

convert all the categorical vaiables into numerical/dummy variables

In [29]:

```
#one hot Encoding
```

In [30]:

```
#dummy Trap can be used
```

In [31]:

```
telco_data_dummies = pd.get_dummies(telco_data)
telco_data_dummies.head()
```

Out[31]:

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

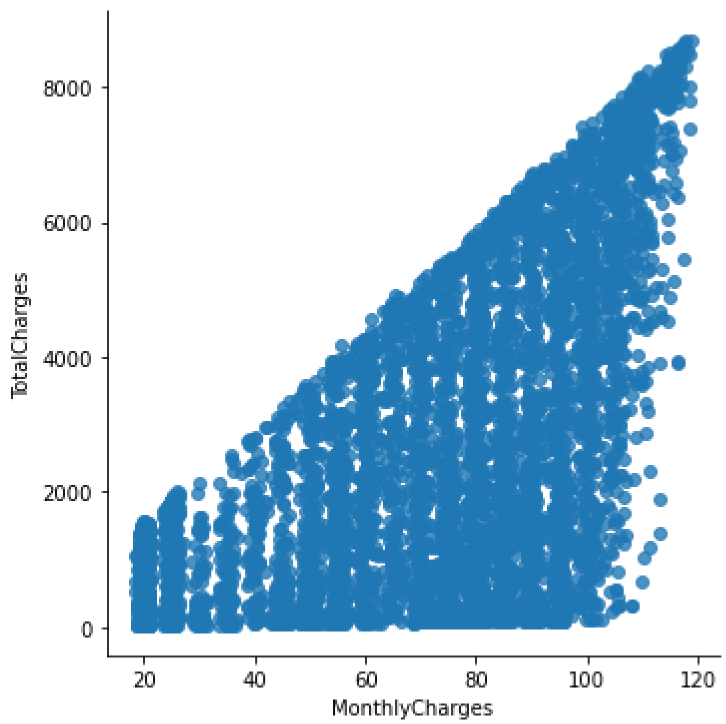
5 rows × 51 columns

In [32]:

```
sns.lmplot(data=telco_data_dummies, x= 'MonthlyCharges', y='TotalCharges', fit_reg=False)
```

Out[32]:

```
<seaborn.axisgrid.FacetGrid at 0x17fc1c2cf40>
```

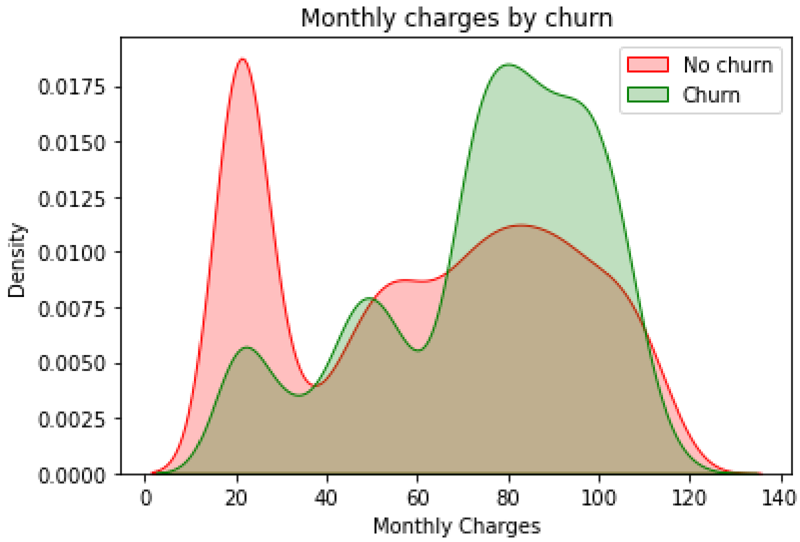


In [33]:

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

Out[33]:

```
Text(0.5, 1.0, 'Monthly charges by churn')
```



In [34]:

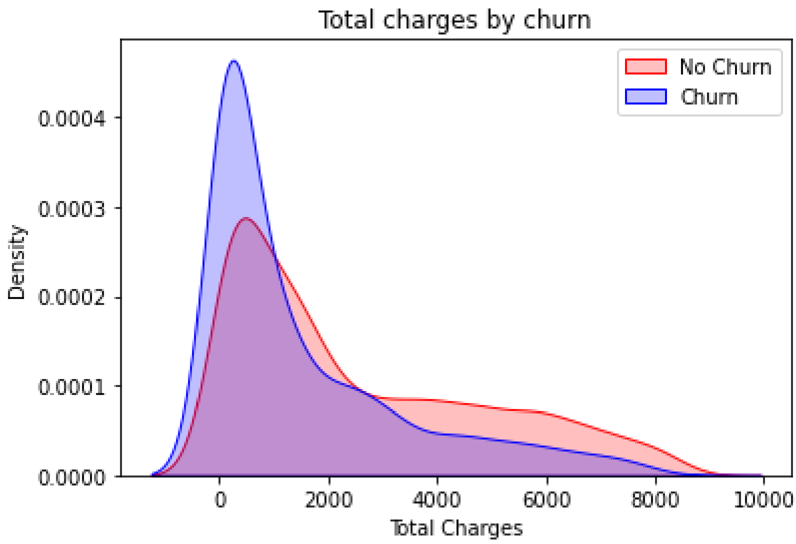
```
# key Insight found As higher the monthly charge, lower the tenure, Lower the Total charge-
```

In [35]:

```
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["Churn"] == 0) ],
                color="Red", shade = True)
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["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[35]:

```
Text(0.5, 1.0, 'Total charges by churn')
```



In [ ]:

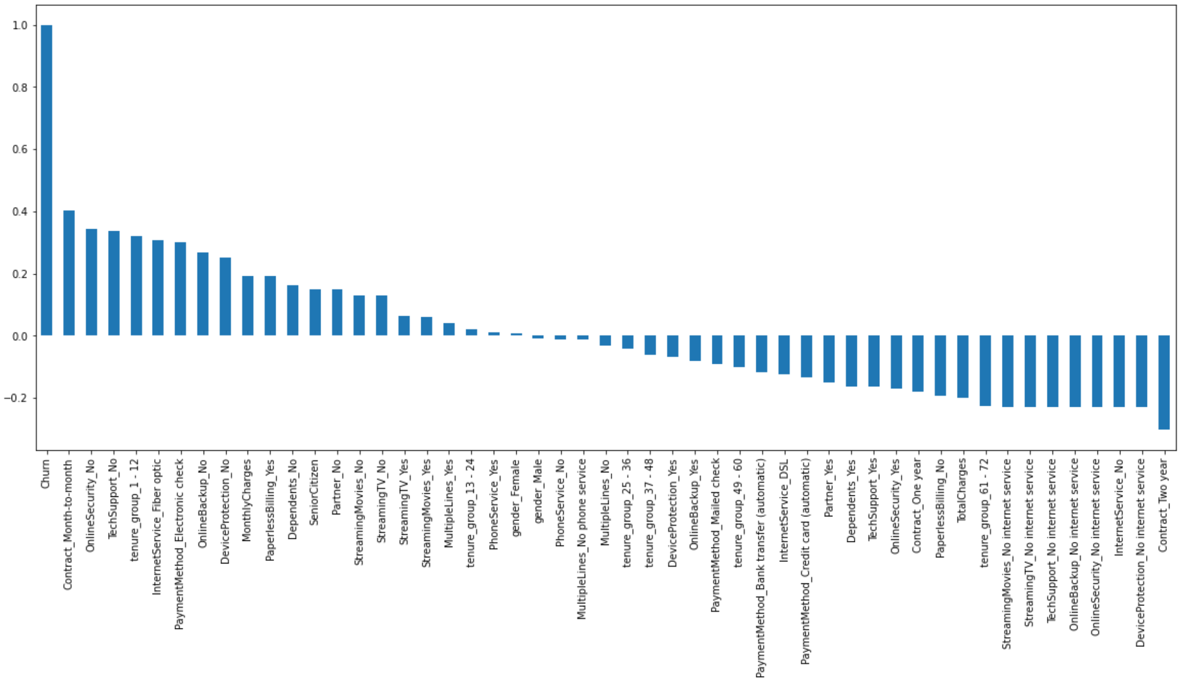
```
# So from here we can Infer that that There is more churning when the total Charges are Low
# understanding because that is quite obvious that if you have not paid a hefty amount to s
# switch.
```

In [36]:

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

Out[36]:

<AxesSubplot:>



In [37]:

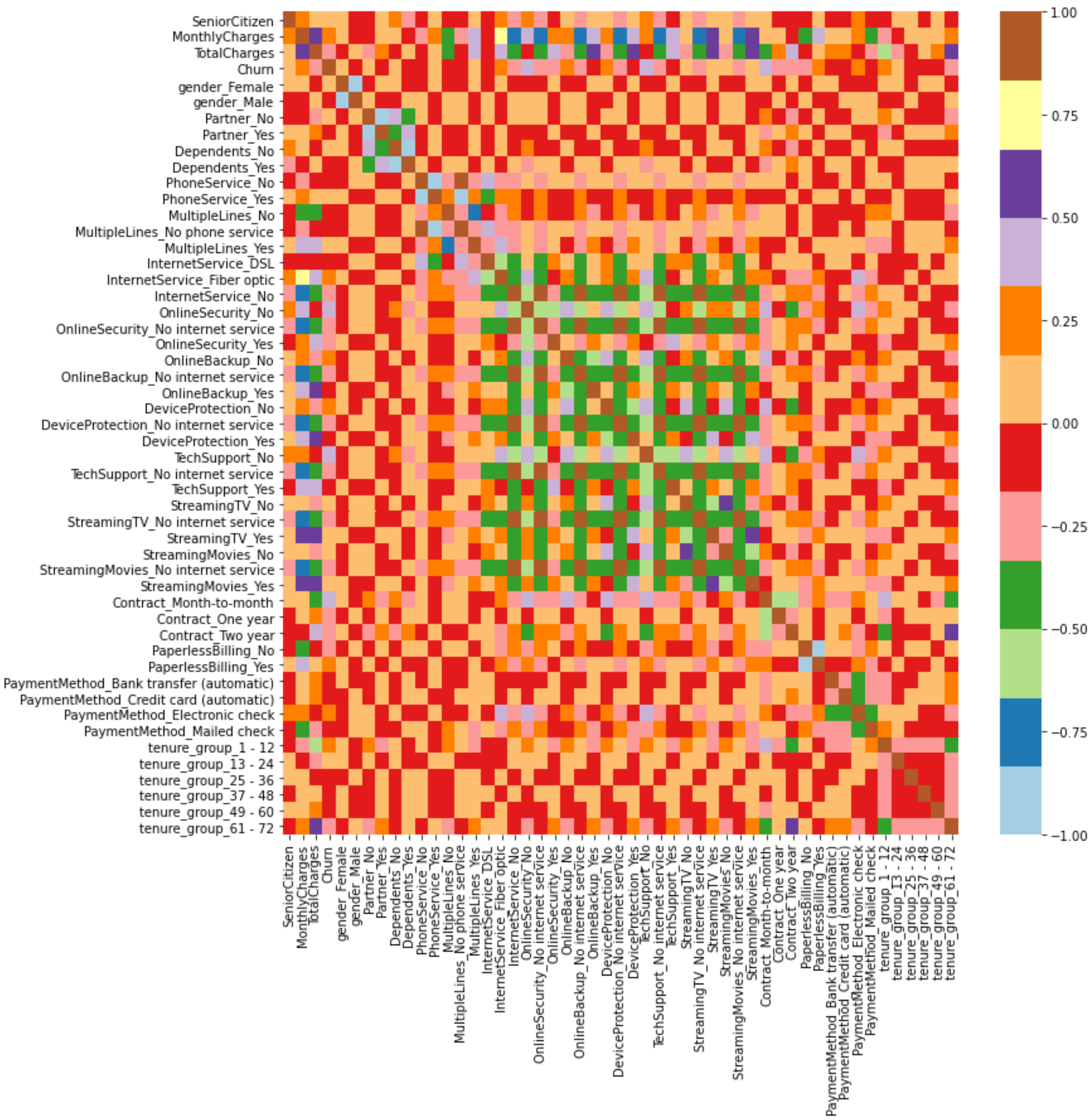
```
# key Insights:-
# High Churn Rate seen in Contract_Month_to_month, No online Security , No Tech Support, Fi
# Low Churn Rate seen in Long term Contracts, Subscriptions without net, Customer Enged for
# Gender, Availablity of phone Service,-----> no impact on churn
```

In [38]:

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

Out[38]:

```
<AxesSubplot:>
```



In [39]:

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

In [40]:

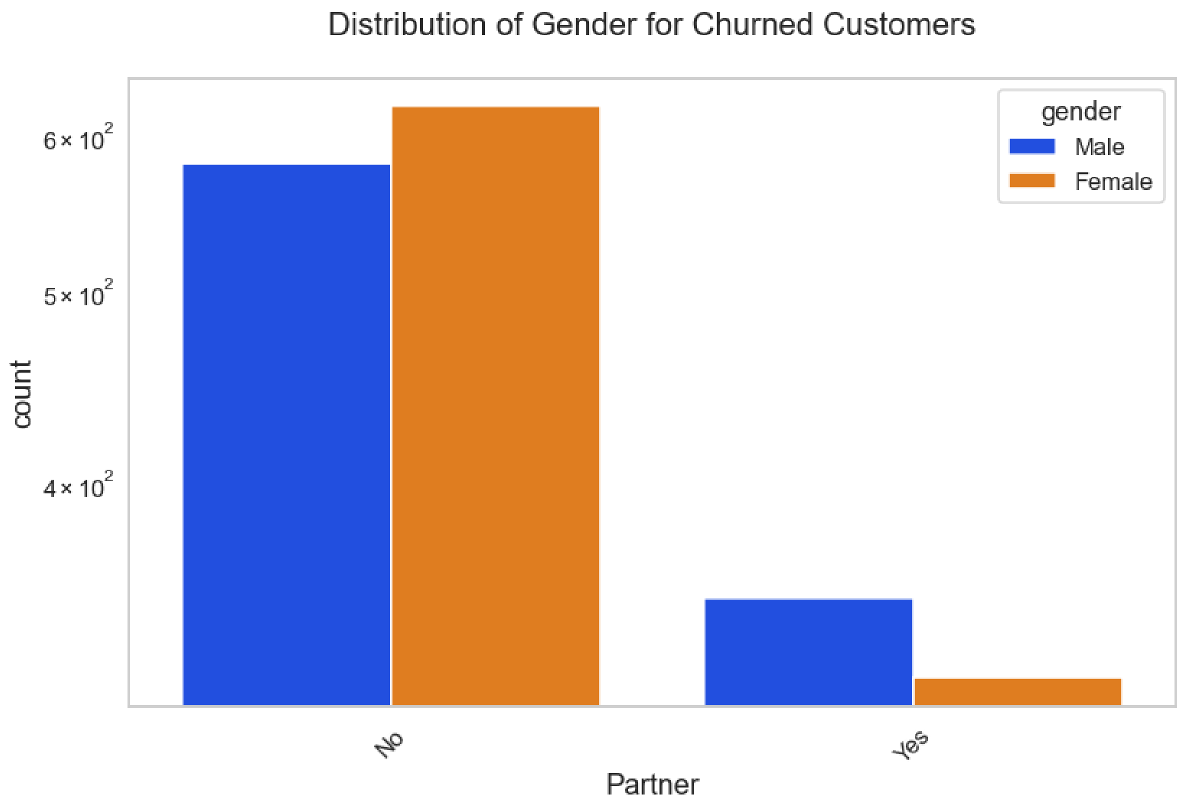
```
def uniplot(df,col,title,hue = None):

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

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

In [41]:

```
uniplot(new_df1_target1,col='Partner',title='Distribution of Gender for Churned Customers',
```

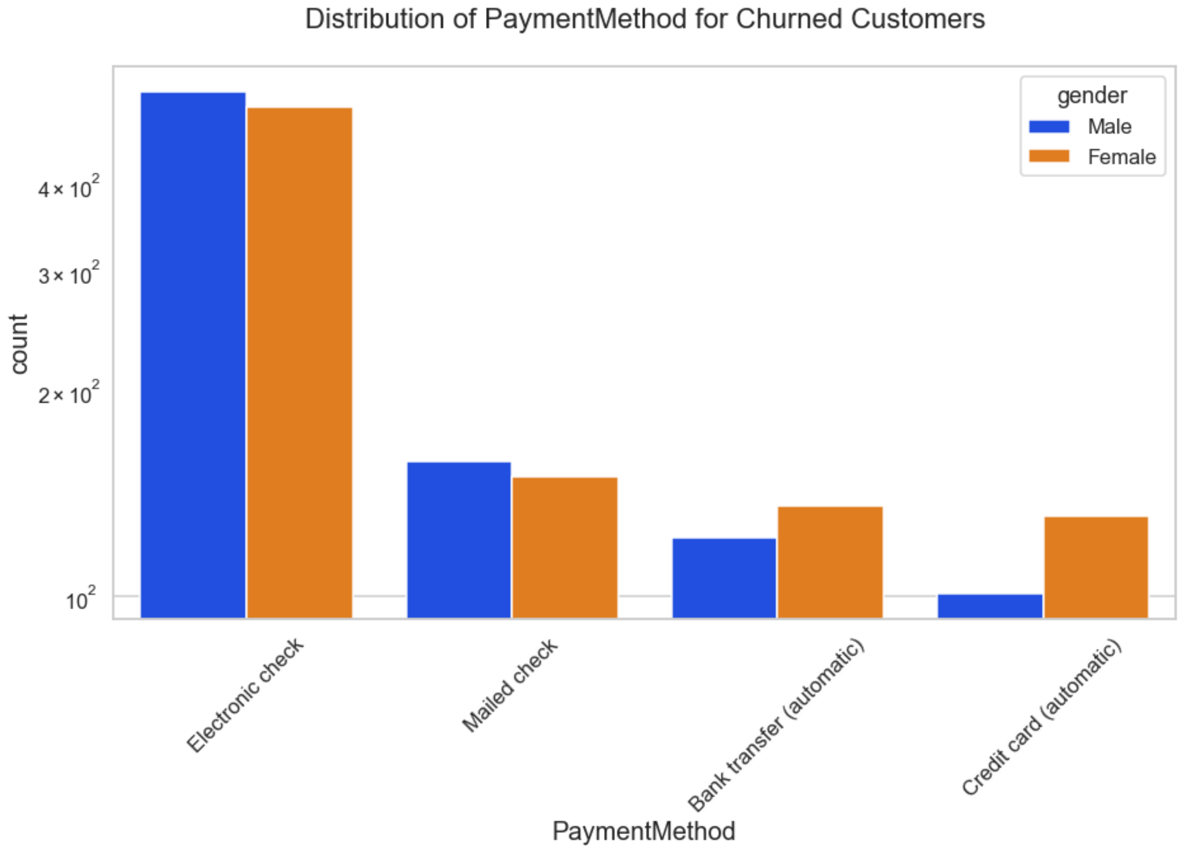


In [42]:

```
# So from here we can see that if there is no partner and and if the selected candidate is
# the Inverse.
```

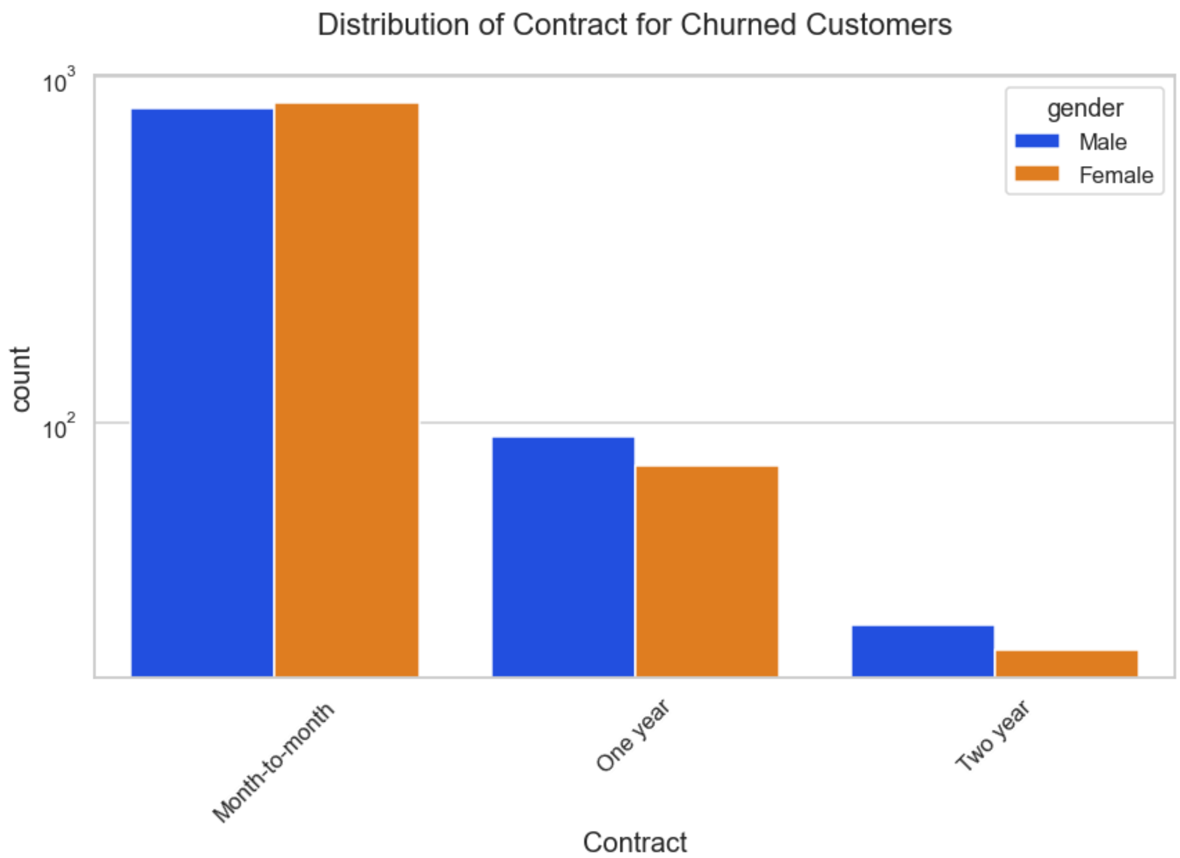
In [43]:

```
uniplot(new_df1_target1,col='PaymentMethod',title= 'Distribution of PaymentMethod for Churn
```



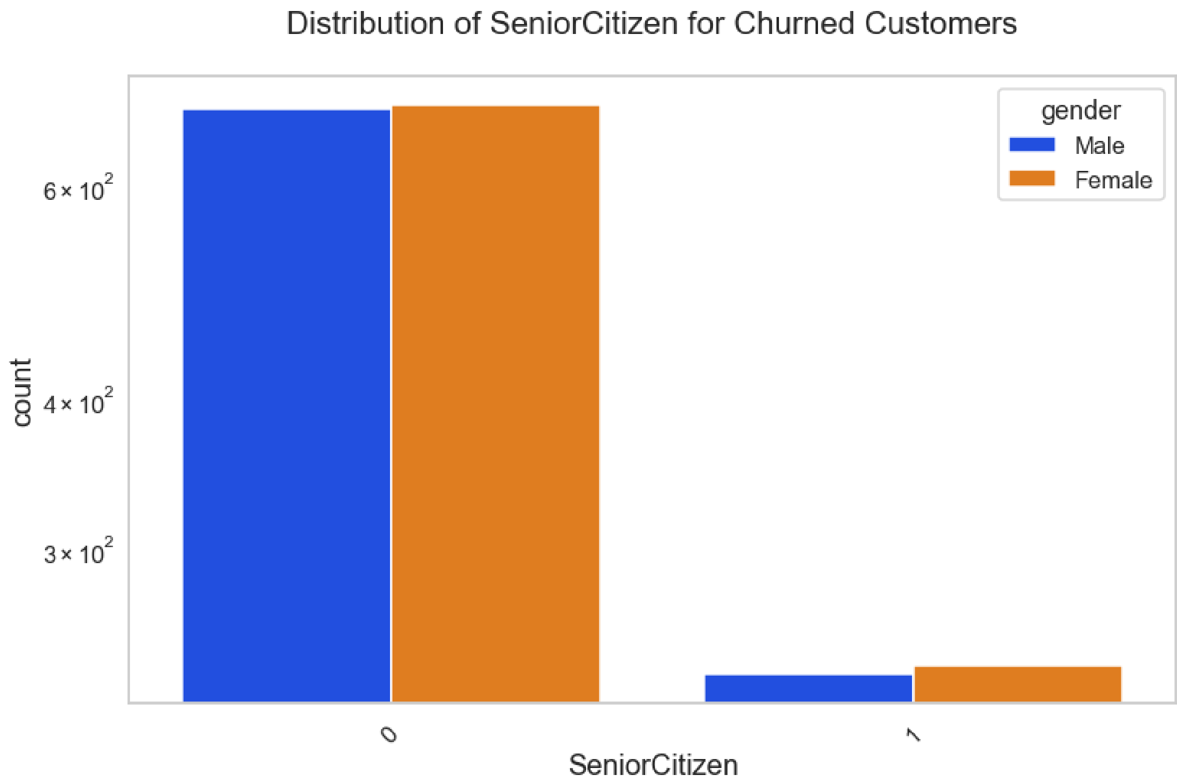
In [44]:

```
uniplot(new_df1_target1,col='Contract',title= 'Distribution of Contract for Churned Custome
```



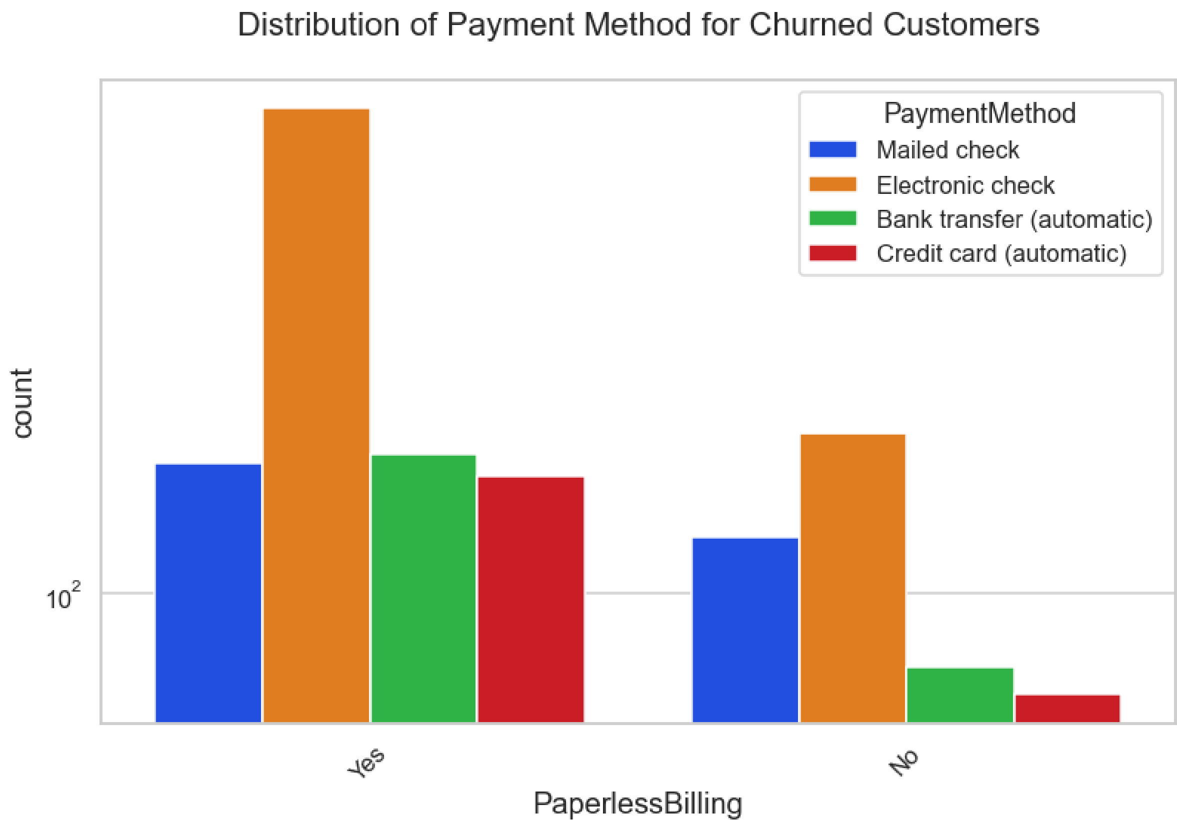
In [45]:

```
uniplot(new_df1_target1,col='SeniorCitizen',title= 'Distribution of SeniorCitizen for Churn
```



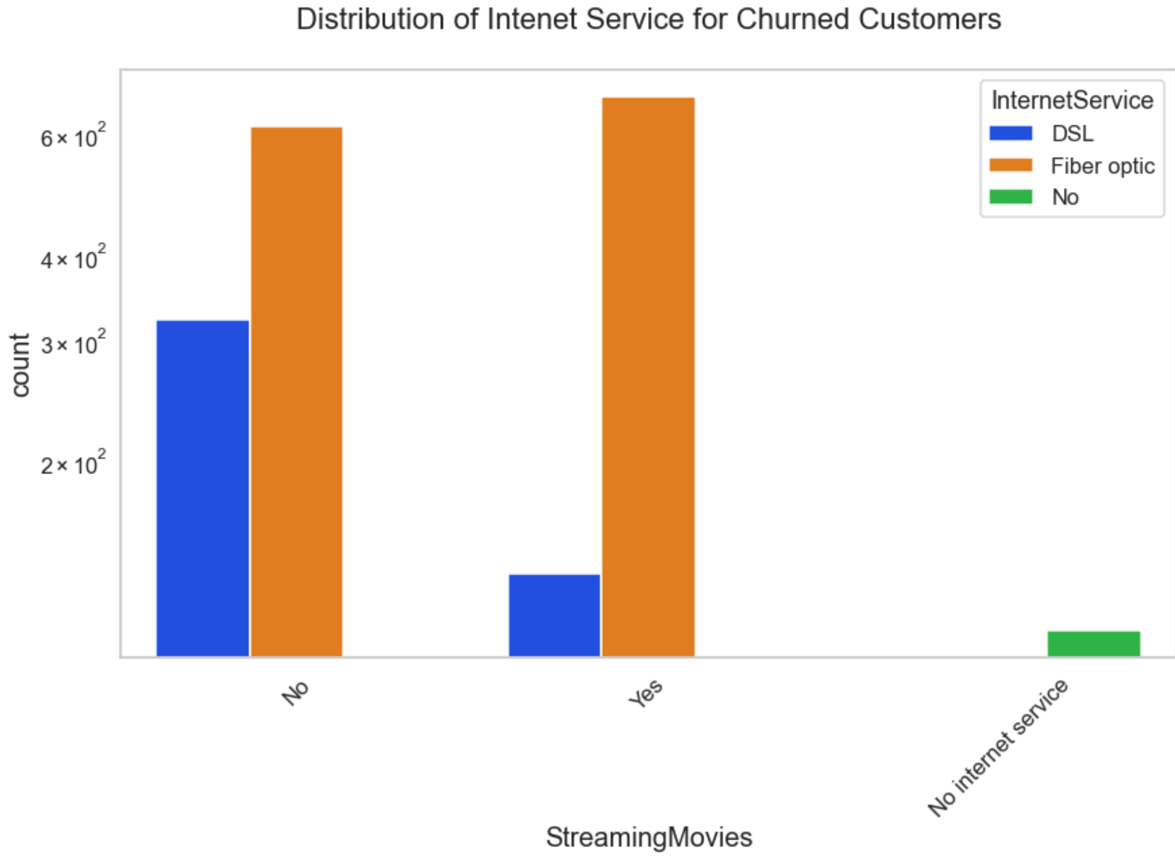
In [46]:

```
uniplot(new_df1_target1,col='PaperlessBilling',title= 'Distribution of Payment Method for C
```



In [47]:

```
uniplot(new_df1_target1,col='StreamingMovies',title= 'Distribution of Intenet Service for C
```



In [48]:

```
# Basic Conclusions from Uniariate and Bivariate analysis
# 1.) Montly Contract People Are most Likely to Churn at higher rate
# 2.) Electronic Check Payment are the People having Higher Churning rate
# 3.) Non- Senior Citizen Citizen are High Churners
# 4.) No Online Security , No Tech Support category are higher Churners
```

In [49]:

```
telco_data_dummies.to_csv('tel_churn.csv')
```

In [50]:

```
df = pd.read_csv('tel_churn.csv')
```

In [51]:

```
df.head()
```

Out[51]:

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

5 rows × 52 columns

In [52]:

```
df = df.drop('Unnamed: 0', axis=1)
```

In [53]:

```
df.head()
```

Out[53]:

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

5 rows × 51 columns

In [54]:

```
# now create variables x and y where x is independent variables and y is dependent variable
```

In [55]:

```
x= df.drop('Churn', axis=1)
x
```

Out[55]:

	SeniorCitizen	MonthlyCharges	TotalCharges	gender_Female	gender_Male	Partner_No
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
...	...	...	...	...	...	...
7027	0	84.80	1990.50	0	1	0
7028	0	103.20	7362.90	1	0	0
7029	0	29.60	346.45	1	0	0
7030	1	74.40	306.60	0	1	0
7031	0	105.65	6844.50	0	1	1

7032 rows × 50 columns

In [56]:

```
y = df['Churn']
y
```

Out[56]:

```
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 [57]:

```
# Evaluation Metrics to be considered
```

In [58]:

```
# Before Moving on to the model processing we must be sure that on what basis are we going
# If we observe our output column is Churn that is a categorical column so we sevral choice
# Recall , F-1/F-2/F-0.5 scores.
```

In [59]:

```
# Lets try to undertand by considering the first evalution metric that is accuracy.
# Accuracy will not be a good evaluation metric the reason being the Imbalanced dataset. If
# model will be more biased towards the majority class that is not churning and it is highl
# minority class. Due to which even though we will have high accuracy but that would not ad
# won't be a deciding parameter.
```

In [60]:

```
# Let us try with the second Evaluation metric that is Precision.
# Precision means out of all the predicted positive values how many of them are actually po
# So if we try to observe the Formula of Precision :- T.P/T.P+F.P
# So we are concerned more with the term False positive :- Here False positive means we pre
# but actually he is not. So if we consider this from a business point of view then there i
# though our model is bit inaccurate.
# so we will not consider Precision in this case.
```

In [61]:

```
# Let us try with the recall :
# Recall means out of actual positive values how many of them are correctly classified.
# So if we look at the formula of Recall:- T.P/T.P+ F.N
# Here we are concerned more with the false negative term :- False negative means we predic
# but actually he is churning. So this could be a major Setback for the organization becaus
# loyal customer where as he is a churner.
# So this is a potential parameter that could be considered as evaluation metric
```

In [62]:

```
# Now we know that Recall could be a Important evaluation metric and in our scenario its mo
# So we are not neglecting the F- score because the number of False positive is not that hi
# So if we are considering the F-score as F.N>F.P(Weightage) So we will be considering F-2,
# balance.
```

In [63]:

```
# Normal train test Split (70-30)
```

In [64]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3)
```

In [ ]:

```
# So as we see that recall score is quite less (less than 50%) so the amount of False negat
# our models recall score is less. Due to which it becomes quite Important to emphasize on
# business perspective of an organization in a wrong direction.
```

In [105]:

```
# Decision Tree Classifier
```

In [106]:

```
model_dt = DecisionTreeClassifier(criterion='entropy',random_state=100,max_depth=6, min_sam
```

In [107]:

```
model_dt.fit(x_train, y_train)
```

Out[107]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_leaf=8,
                       random_state=100)
```

In [108]:

```
y_pred = model_dt.predict(x_test)
```

In [109]:

```
y_pred
```

Out[109]:

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

In [110]:

```
model_dt.score(x_test,y_test)
```

Out[110]:

```
0.7895734597156399
```

In [111]:

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

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1557
           1       0.63      0.49      0.55       553

    accuracy                           0.79      2110
   macro avg       0.73      0.69      0.71      2110
weighted avg       0.78      0.79      0.78      2110
```

In [115]:

```
confusion=confusion_matrix(y_test, y_pred)
print(confusion)
```

```
[[1395  162]
 [ 282  271]]
```

In [117]:

```
recall = confusion[0][0] /(confusion[0][0] + confusion[1][0])
print(recall)
```

```
0.8318425760286225
```

In [ ]:

```
# So we obtained the Recall for the Decision Tree Model
```

In [ ]:

```
# Recall value should Ideally be high because it tells that the there are less false negati
```

In [ ]:

```
# 80% recall means that our model has correctly classified 80% of actual positve data. To o
# Differnt models and then can test the result.
```

In [118]:

```
#RANDOM FOREST CLASSIFIER
```

In [119]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [120]:

```
model_rf = RandomForestClassifier(criterion='gini',random_state=100,max_depth=5, min_sample
model_rf.fit(x_train, y_train)
y_pred_rf = model_rf.predict(x_test)
```

In [121]:

```
model_rf.score(x_test,y_test)
```

Out[121]:

```
0.7862559241706161
```

In [122]:

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

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1557
           1       0.63      0.49      0.55       553

    accuracy                           0.79      2110
   macro avg       0.73      0.69      0.71      2110
weighted avg       0.78      0.79      0.78      2110
```

In [124]:

```
conf_rf= confusion_matrix(y_test, y_pred_rf)
print(conf_rf)
```

```
[[1472   85]
 [ 366  187]]
```

In [126]:

```
recall = conf_rf[0][0] /(conf_rf[0][0] + conf_rf[1][0])
print(recall)
```

0.8008705114254625

In [127]:

```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [128]:

```
params = {
    'max_depth': [2,3,5,6,7,15,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
}
```

In [129]:

```
from sklearn.model_selection import GridSearchCV
```

In [130]:

```
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="recall")
```

In [131]:

```
grid_search.fit(x_train, y_train)
```

Fitting 4 folds for each of 252 candidates, totalling 1008 fits

Out[131]:

```
GridSearchCV(cv=4, estimator=RandomForestClassifier(n_jobs=-1, random_state=
42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 6, 7, 15, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='recall', verbose=1)
```

In [132]:

```
grid_search.best_score_
```

Out[132]:

0.49316109422492405

In [133]:

```
rf_best = grid_search.best_estimator_
rf_best
```

Out[133]:

```
RandomForestClassifier(max_depth=15, min_samples_leaf=10, n_estimators=10,
                       n_jobs=-1, random_state=42)
```

In [134]:

```
# So we have obtained some of the random forest parameters that will be ideal fit according
# Lets try with that and see if any change is possible.
```

In [138]:

```
model_rf_1 = RandomForestClassifier(criterion='entropy',random_state=42,max_depth=15, min_s
model_rf_1.fit(x_train, y_train)
y_pred_rf_1 = model_rf_1.predict(x_test)
```

In [139]:

```
conf_rf_1= confusion_matrix(y_test, y_pred_rf_1)
print(conf_rf_1)
```

```
[[1420  137]
 [ 294  259]]
```

In [140]:

```
recall = conf_rf_1[0][0] /(conf_rf_1[0][0] + conf_rf_1[1][0])
print(recall)
```

```
0.8284714119019837
```

In [ ]:

```
# so we can see by Hyperparameter tunning(Grid_Search_CV) my Recall score Increased.
```

In [ ]:

```
# Trying with XG boost Classifier
```

In [141]:

```
model = XGBClassifier()
```

In [142]:

```
model.fit(x_train, y_train)
```

```
D:\PYTHON\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of
label encoder in XGBClassifier is deprecated and will be removed in a future
release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your lab
els (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[13:05:28] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.
```

Out[142]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=Fal
se,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=na
n,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [145]:

```
y_pred = model.predict(x_test)
```

In [146]:

```
conf_XGB = confusion_matrix(y_test, y_pred)
```

In [147]:

```
print(conf_XGB)
```

```
[[1405  152]
 [ 283  270]]
```

In [148]:

```
recall = conf_XGB[0][0] /(conf_XGB[0][0] + conf_XGB[1][0])
print(recall)
```

```
0.832345971563981
```

In [ ]:

```
# So we can observe that by using XG boost classifier we are getting quite decent Recall.
```

In [149]:

```
# So similarly we can try differnt models and try to understand which model can be a perfec
```

In [ ]:

```
# ------------------------------------------------------------------------------------------
```