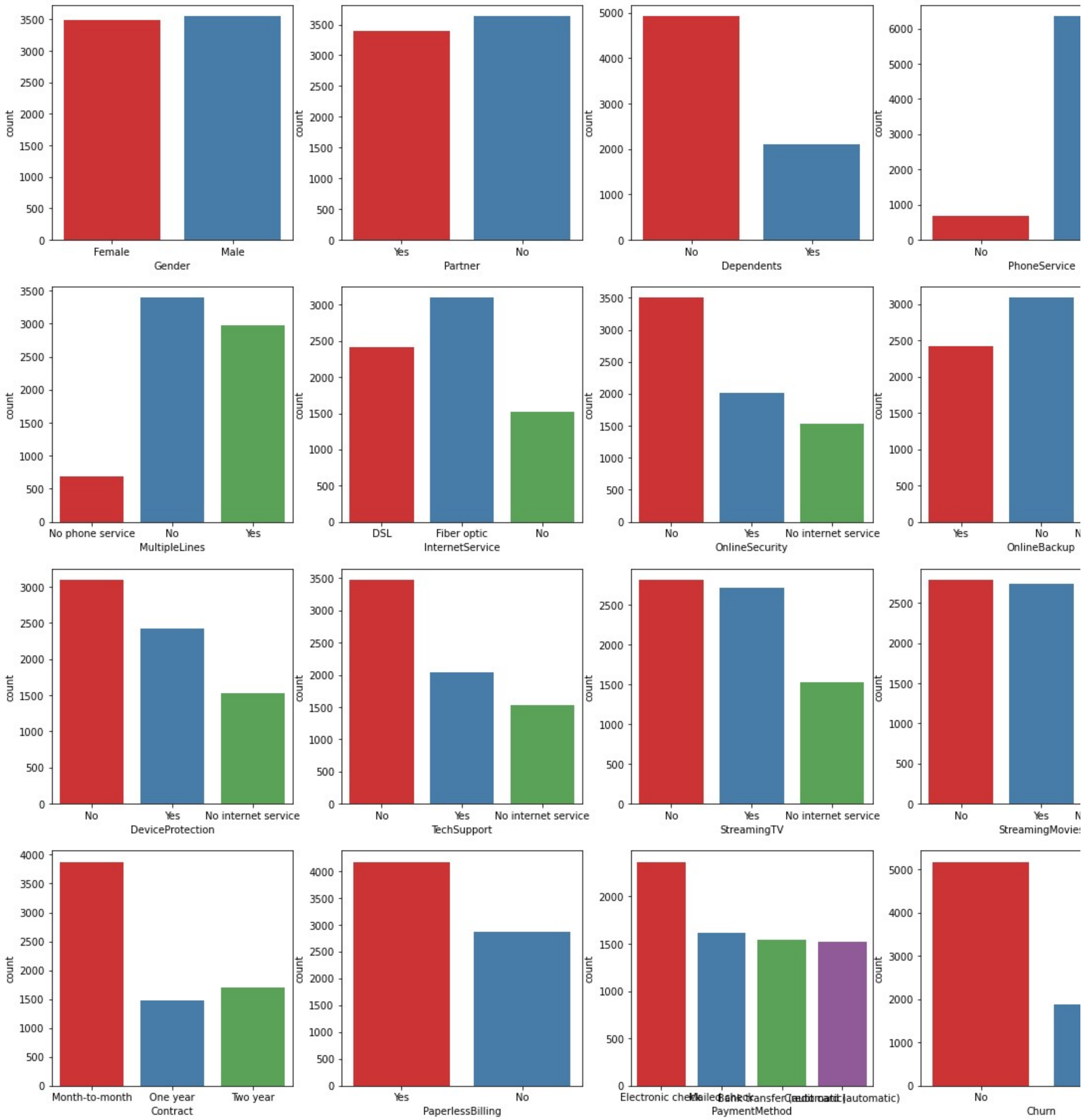
```
In [39]:
    ...: """
    ...: Created on Thu Dec 10 18:05:51 2020
    ...:
    ...: @author: Karthii78
    ...: """
    ...: import numpy as np
    ...: import pandas as pd
    ...:
    ...: #Data Vizualization Libraries
    ...: import matplotlib.pyplot as plt
    ...: import seaborn as sns
    ...:
    ...: #Reading the file
    ...: datset_churn=pd.read_csv('tele_churn.csv')
    ...:
    ...:
    ...: # Machine Learning Library
    ...: from sklearn.preprocessing import LabelEncoder # Encode Categorical Variable to
Numerical Variable
    ...: from sklearn.impute import SimpleImputer # Imputer Class to replace missing
values
    ...: from sklearn.metrics import confusion_matrix # Library for model evaluation
    ...: from sklearn.metrics import accuracy_score # Library for model evaluation
    ...: from sklearn.model_selection import train_test_split # Library to split datset
into test and train
    ...: import warnings
    ...: warnings.filterwarnings('ignore')

In [40]:
    ...: datset_churn_copy = datset_churn.copy()
    ...: datset_churn.shape
    ...: datset_churn.columns.values
    ...:
    ...: ## Renaming columns
    ...: datset_churn = datset_churn.rename(columns={'customerID' : 'CustomerID' ,
'gender': 'Gender', 'tenure':'Tenure'})
    ...: print(datset_churn.columns.values)
['CustomerID' 'Gender' 'SeniorCitizen' 'Partner' 'Dependents' 'Tenure'
 'PhoneService' 'MultipleLines' 'InternetService' 'OnlineSecurity'
 'OnlineBackup' 'DeviceProtection' 'TechSupport' 'StreamingTV'
 'StreamingMovies' 'Contract' 'PaperlessBilling' 'PaymentMethod'
 'MonthlyCharges' 'TotalCharges' 'Churn']

In [41]:
    ...:
    ...: datset_churn['TotalChargesNum']=pd.to_numeric(datset_churn['TotalCharges'])
Traceback (most recent call last):

  File "pandas\_libs\lib.pyx", line 1926, in pandas._libs.lib.maybe_convert_numeric

ValueError: Unable to parse string " "


During handling of the above exception, another exception occurred:
```

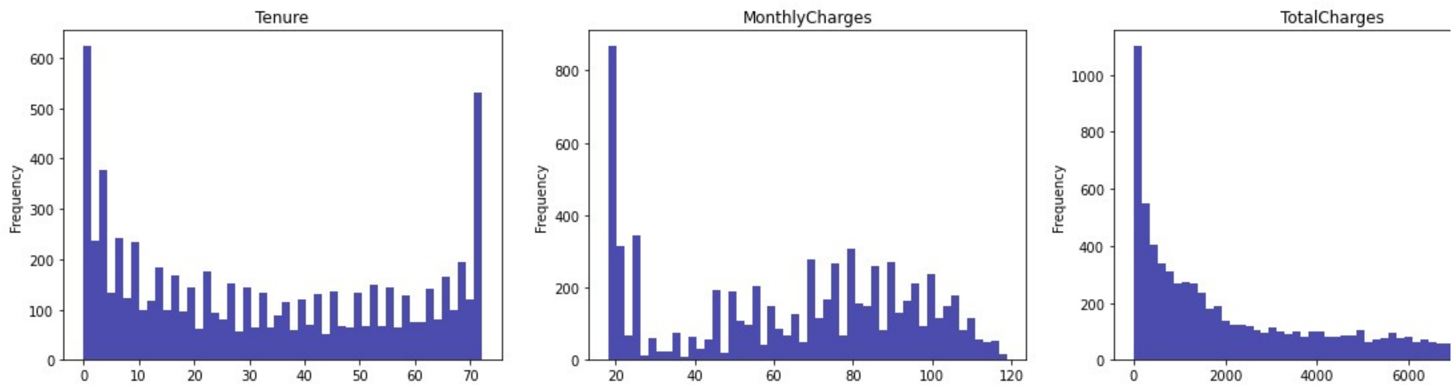
```
Traceback (most recent call last):

  File "<ipython-input-41-357eb8b9ad7c>", line 1, in <module>
    datset_churn['TotalChargesNum']=pd.to_numeric(datset_churn['TotalCharges'])

  File "C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\tools\numeric.py", line
149, in to_numeric
    values = lib.maybe_convert_numeric(

  File "pandas\_libs\lib.pyx", line 1963, in pandas._libs.lib.maybe_convert_numeric

ValueError: Unable to parse string " " at position 488


In [42]:
    ...: missing_value_row = list(datset_churn[datset_churn['TotalCharges'] == "
"].index)
    ...:
    ...: print('Missing Value Rows-->', missing_value_row , '\nTotal rows-->',
len(missing_value_row))
    ...:
    ...: ## replacing msiiing values with zero
    ...: for missing_row in missing_value_row :
    ...:     datset_churn['TotalCharges'][missing_row] = 0
    ...: datset_churn['TotalCharges']=pd.to_numeric(datset_churn['TotalCharges'])
    ...: datset_churn.describe(include=['O'])
    ...:
    ...: datset_churn_column = list(datset_churn.columns)
Missing Value Rows--> [488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754]
Total rows--> 11

In [43]:
    ...: datset_churn_column.remove('CustomerID')
    ...: datset_churn_column.remove('SeniorCitizen')
    ...: datset_churn_column.remove('Tenure')
    ...: datset_churn_column.remove('MonthlyCharges')
    ...: datset_churn_column.remove('TotalCharges')
    ...:
    ...: # Printing Unique values in each categorical column
    ...: for col in datset_churn_column:
    ...:     print(col, "-", datset_churn[col].unique())
Gender - ['Female' 'Male']
Partner - ['Yes' 'No']
Dependents - ['No' 'Yes']
PhoneService - ['No' 'Yes']
MultipleLines - ['No phone service' 'No' 'Yes']
InternetService - ['DSL' 'Fiber optic' 'No']
OnlineSecurity - ['No' 'Yes' 'No internet service']
OnlineBackup - ['Yes' 'No' 'No internet service']
DeviceProtection - ['No' 'Yes' 'No internet service']
TechSupport - ['No' 'Yes' 'No internet service']
StreamingTV - ['No' 'Yes' 'No internet service']
StreamingMovies - ['No' 'Yes' 'No internet service']
Contract - ['Month-to-month' 'One year' 'Two year']
PaperlessBilling - ['Yes' 'No']
PaymentMethod - ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn - ['No' 'Yes']
```

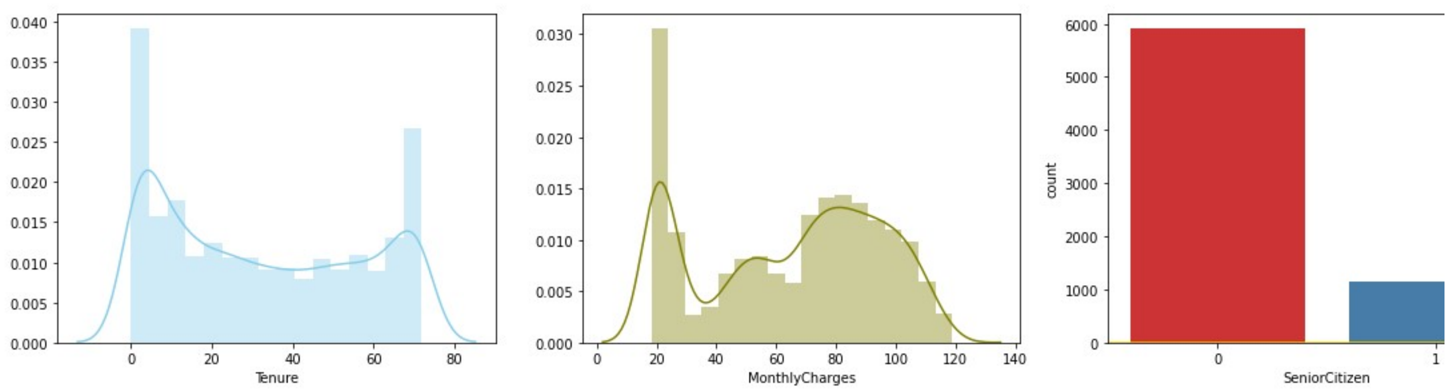
```
In [44]:
    ...: datset_churn.describe()
    ...: total = datset_churn.isnull().sum().sort_values(ascending=False)
    ...:
    ...: ##Prinyting the percentage of missing data in the columns
    ...: percent = (datset_churn.isnull().sum()/
datset_churn.isnull().count()).sort_values(ascending=False)
    ...: missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    ...: print(missing_data)
                  Total  Percent
Churn                 0      0.0
OnlineSecurity        0      0.0
Gender                0      0.0
SeniorCitizen         0      0.0
Partner               0      0.0
Dependents            0      0.0
Tenure                0      0.0
PhoneService          0      0.0
MultipleLines         0      0.0
InternetService       0      0.0
OnlineBackup          0      0.0
TotalCharges          0      0.0
DeviceProtection      0      0.0
TechSupport           0      0.0
StreamingTV           0      0.0
StreamingMovies       0      0.0
Contract              0      0.0
PaperlessBilling      0      0.0
PaymentMethod         0      0.0
MonthlyCharges        0      0.0
CustomerID            0      0.0

In [45]: datset_churn[['MonthlyCharges','Tenure','TotalCharges']].head()
    ...: zero_value_row = list(datset_churn[datset_churn['TotalCharges'] == 0].index)
    ...: print('0 Value Rows-->', missing_value_row , '\nTotal rows-->',
len(missing_value_row))
    ...:
    ...: # Replacing the spaces considered with zero by the value obtained by
multiplying monthly charge and tenure
    ...: for zero_row in zero_value_row :
    ...:     datset_churn['TotalCharges'][zero_row] = datset_churn['Tenure'][zero_row] *
datset_churn['MonthlyCharges'][zero_row]
    ...:
    ...:
    ...: for zero_row in zero_value_row :
    ...:     print( datset_churn['MonthlyCharges'][zero_row],datset_churn['Tenure']
[zero_row],datset_churn['TotalCharges'][zero_row])
0 Value Rows--> [488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754]
Total rows--> 11
52.55 0 0.0
20.25 0 0.0
80.85 0 0.0
25.75 0 0.0
56.05 0 0.0
19.85 0 0.0
25.35 0 0.0
20.0 0 0.0
```

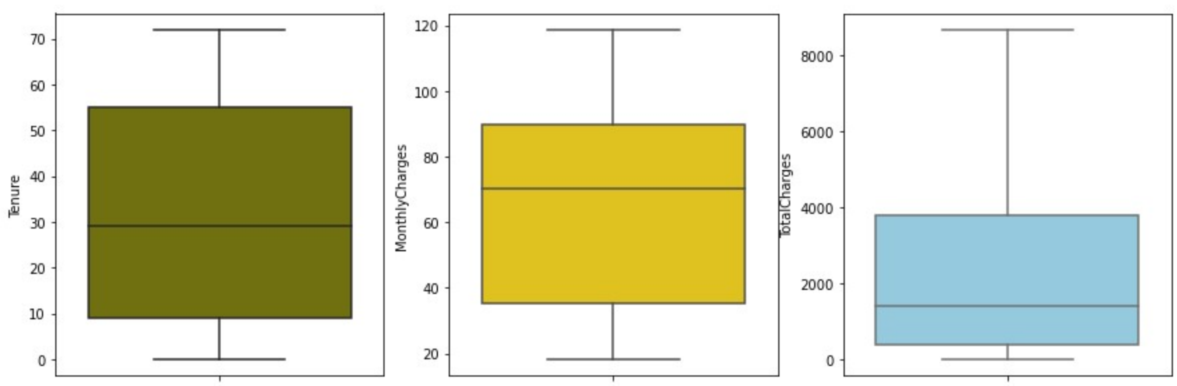
```
19.7 0 0.0
73.35 0 0.0
61.9 0 0.0

In [46]:
    ...: columns_hist = list(datset_churn.columns)
    ...:
    ...: #Removing the Numerical Variables
    ...: columns_hist.remove('CustomerID')
    ...: columns_hist.remove('SeniorCitizen')
    ...: columns_hist.remove('Tenure')
    ...: columns_hist.remove('MonthlyCharges')
    ...: columns_hist.remove('TotalCharges')
    ...:
    ...: #Creating Column into 4X4 matrix to display 16 bar charts in 4X4 form:
    ...: columns_hist_nparray = np.array(columns_hist)
    ...: columns_hist_nparray = np.reshape(columns_hist_nparray, (4,4)) # reshaping the
columns into 4X4 matrix
    ...:
    ...: ## Univariate Analysis of each categorical Variables

In [47]: rows = 4 ; columns = 4
    ...: f, axes = plt.subplots(rows, columns, figsize=(20, 20))
    ...: print('Univariate Analysis of each categorical Variables')
    ...: for row in range(rows):
    ...:     for column in range(columns):
    ...:         sns.countplot(datset_churn[columns_hist_nparray[row][column]], palette
= "Set1", ax = axes[row, column])
    ...:
    ...: print('Univariate Analysis of each numerical Variables')
    ...: f, axes = plt.subplots(2, 3, figsize=(20,10))
    ...: #Charting the histogram
    ...: datset_churn["Tenure"].plot.hist(color='DarkBlue', alpha=0.7, bins=50,
title='Tenure',ax=axes[0, 0])
    ...: datset_churn["MonthlyCharges"].plot.hist(color='DarkBlue', alpha=0.7, bins=50,
title='MonthlyCharges',ax=axes[0, 1])
    ...: datset_churn["TotalCharges"].plot.hist(color='DarkBlue', alpha=0.7, bins=50,
title='TotalCharges',ax=axes[0, 2])
    ...:
    ...: #Charting the density plot
    ...: sns.distplot( datset_churn["Tenure"] , kde=True, rug=False, color="skyblue",
ax=axes[1, 0])
    ...: sns.distplot( datset_churn["MonthlyCharges"] , kde=True, rug=False,
color="olive", ax=axes[1, 1])
    ...: sns.distplot( datset_churn["TotalCharges"] , kde=True, rug=False, color="gold",
ax=axes[1, 2])
    ...:
    ...:
    ...: sns.countplot(datset_churn['SeniorCitizen'], palette = "Set1")
    ...:
    ...: f, axes = plt.subplots(1, 3, figsize=(15,5))
    ...: sns.boxplot(x=datset_churn["Tenure"], orient="v", color="olive",ax=axes[0])
    ...: sns.boxplot(x=datset_churn["MonthlyCharges"], orient="v",
color="gold",ax=axes[1])
    ...: sns.boxplot(x=datset_churn["TotalCharges"] , orient="v",
color="skyblue",ax=axes[2])
    ...:
    ...:
```

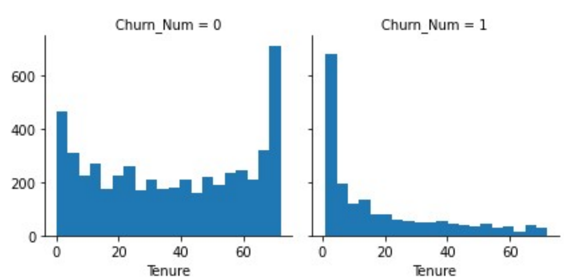
```
    ...: datset_churn['Churn_Num'] = datset_churn['Churn'].map( {'Yes': 1, 'No': 0}
).astype(int)
    ...: datset_churn[['Churn','Churn_Num']].head()
    ...:
    ...: # Plotting Tenure Column with Churn
    ...: # Churn_num indicates customer who left the company. 0 indicates customer who
stayed.
    ...: fighist = sns.FacetGrid(datset_churn, col='Churn_Num')
    ...: fighist.map(plt.hist, 'Tenure', bins=20)
    ...:
    ...:
    ...: # Plotting MonthlyCharges Column with Churn
    ...: # Churn_num indicates customer who left the company. 0 indicates customer who
stayed.
    ...: fighist = sns.FacetGrid(datset_churn, col='Churn_Num')
    ...: fighist.map(plt.hist, 'MonthlyCharges', bins=20)
    ...:
    ...: # Plotting TotalCharges Column with Churn
    ...: # Churn_num indicates customer who left the company. 0 indicates customer who
stayed.
    ...: fighist = sns.FacetGrid(datset_churn, col='Churn_Num')
    ...: fighist.map(plt.hist, 'TotalCharges', bins=20)
Univariate Analysis of each categorical Variables
Univariate Analysis of each numerical Variables
Out[47]: <seaborn.axisgrid.FacetGrid at 0x181829c2670>
```

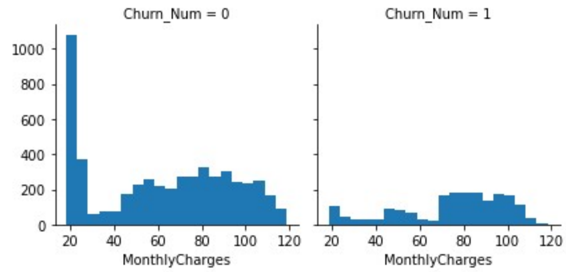

count
Female
Male
Gender
Yes
No
Partner
No
Yes
Dependents
No
PhoneService
No phone service
No
Yes
MultipleLines
DSL
Fiber optic
No
InternetService
No
Yes
No internet service
OnlineSecurity
Yes
No
OnlineBackup
No
Yes
No internet service
DeviceProtection
No
Yes
No internet service
TechSupport
No
Yes
No internet service
StreamingTV
No
Yes
Month-to-month
One year
Two year
Contract
Yes
No
PaperlessBilling
PaymentMethod
No
Churn

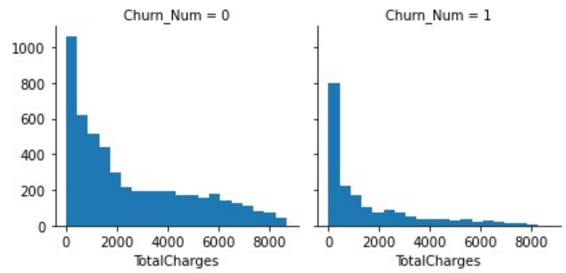

Tenure
MonthlyCharges
TotalCharges
Frequency


Tenure
MonthlyCharges
SeniorCitizen
count


Tenure
MonthlyCharges
TotalCharges

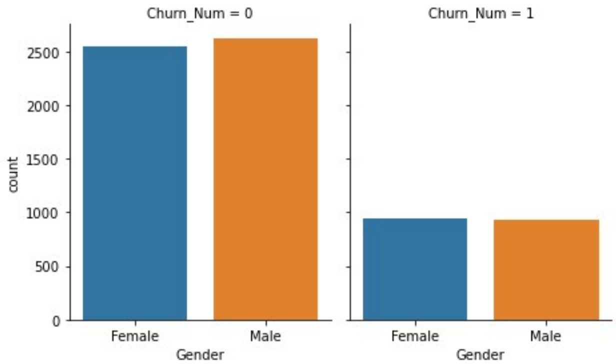

Churn_Num = 0
Churn_Num = 1
Tenure

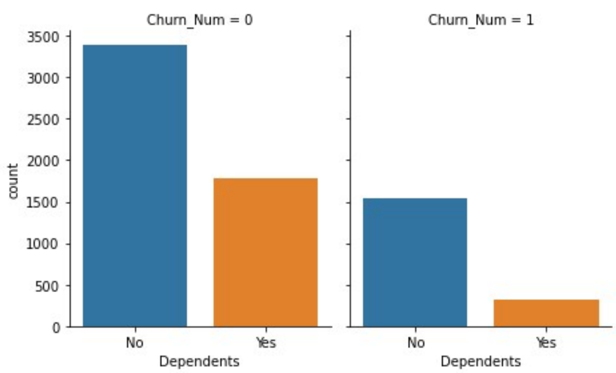


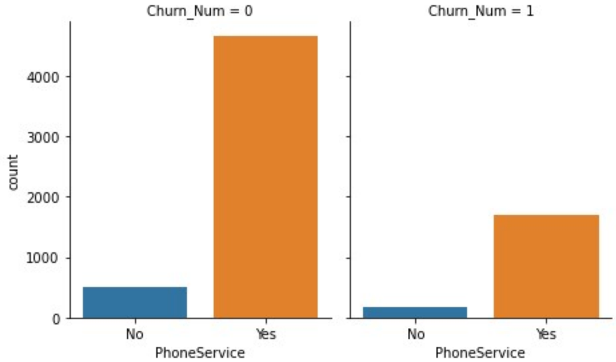


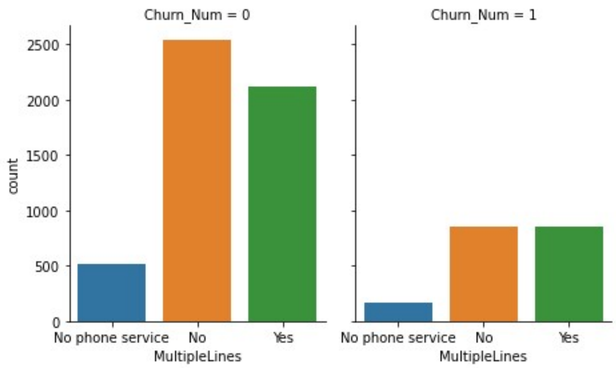

```
In [48]:
    ...:
    ...: col_list = columns_hist
    ...: col_list.remove('Churn')
    ...: for col in col_list:
    ...:     if col == 'PaymentMethod':
    ...:         aspect_ratio = 2.0
    ...:     else:
    ...:         aspect_ratio = 0.8
    ...:
    ...:     plot_cat_data = sns.catplot(x=col, col='Churn_Num', data = datset_churn,
kind='count', height=4, aspect=aspect_ratio)
    ...:
    ...:
    ...: # Creating tenure band and co-relation with Churn
    ...: datset_churn['TenureRange'] = pd.cut(datset_churn['Tenure'], 5)
    ...: datset_churn[['TenureRange', 'Churn_Num']].groupby(['TenureRange'],
as_index=False).mean().sort_values(by='TenureRange', ascending=True)
    ...:
    ...: # Replacing Age band with ordinals based on these bands
    ...: datset_churn.loc[ datset_churn['Tenure'] <= 8, 'TenureCat'] = 0
    ...: datset_churn.loc[(datset_churn['Tenure'] > 8) & (datset_churn['Tenure'] <= 15),
'TenureCat'] = 1
    ...: datset_churn.loc[(datset_churn['Tenure'] > 15) & (datset_churn['Tenure'] <=
30), 'TenureCat'] = 2
    ...: datset_churn.loc[(datset_churn['Tenure'] > 30) & (datset_churn['Tenure'] <= 45
), 'TenureCat'] = 3
    ...: datset_churn.loc[(datset_churn['Tenure'] > 45) & (datset_churn['Tenure'] <= 60
), 'TenureCat'] = 4
    ...: datset_churn.loc[ datset_churn['Tenure'] > 60, 'TenureCat'] = 5
    ...:
    ...: datset_churn[['Tenure','TenureRange','TenureCat']].head(10)
Out[48]:
   Tenure      TenureRange  TenureCat
0       1  (-0.072, 14.4]        0.0
1      34    (28.8, 43.2]        3.0
2       2  (-0.072, 14.4]        0.0
```

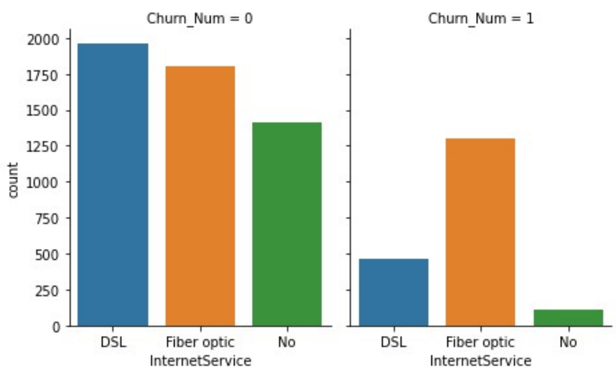
```
3       45    (43.2, 57.6]         3.0
4        2  (-0.072, 14.4]         0.0
5        8  (-0.072, 14.4]         0.0
6       22    (14.4, 28.8]         2.0
7       10  (-0.072, 14.4]         1.0
8       28    (14.4, 28.8]         2.0
9       62    (57.6, 72.0]         5.0
```

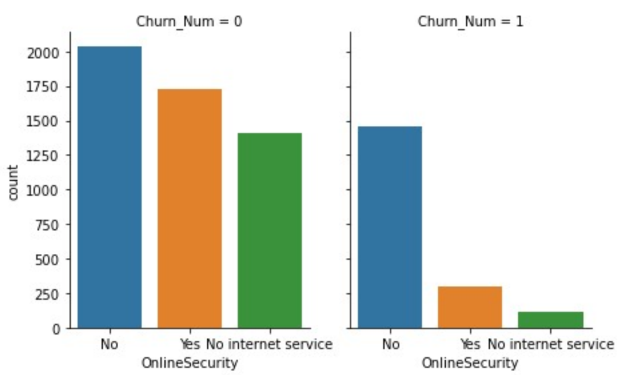


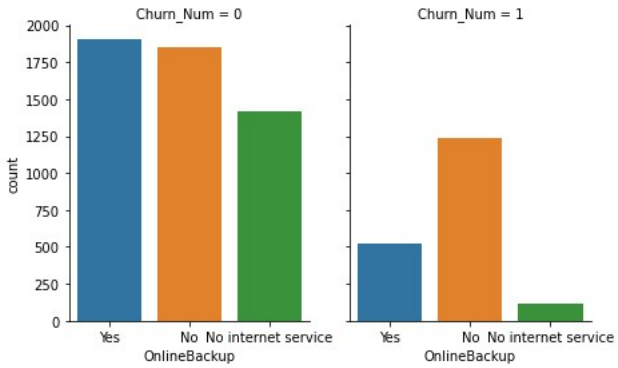


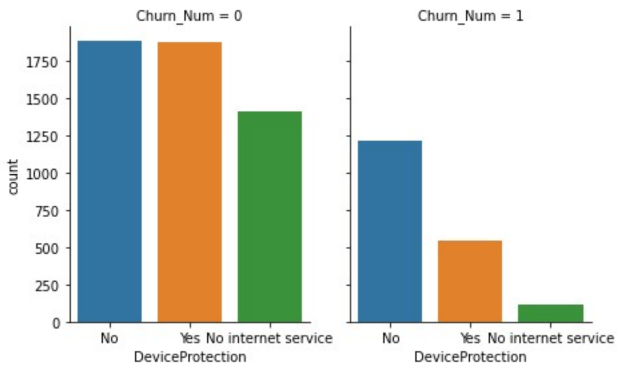

Churn_Num = 0
Churn_Num = 1
count
4000
3000
2000
1000
0
No
Yes
PhoneService

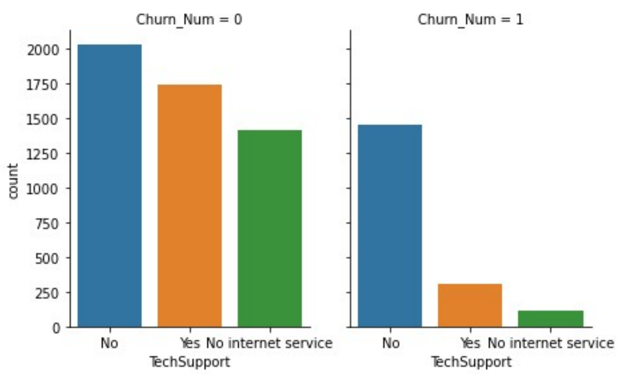

Churn_Num = 0
Churn_Num = 1
count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

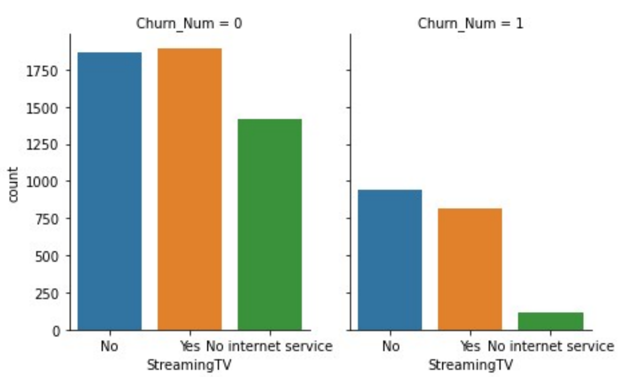

Churn_Num = 0
Churn_Num = 1
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

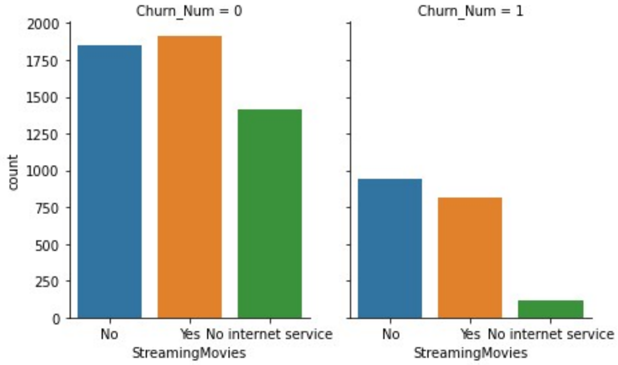

Churn_Num = 0
Churn_Num = 1
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

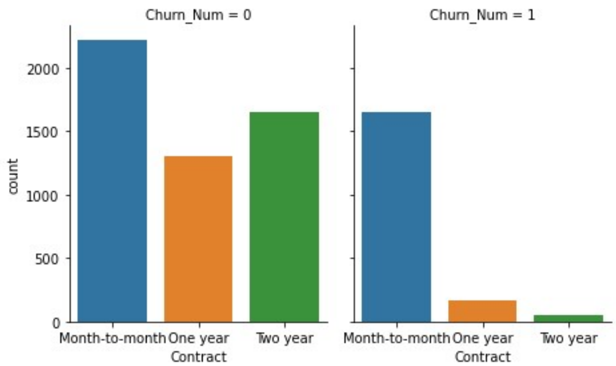

Churn_Num = 0
Churn_Num = 1
count
2000
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

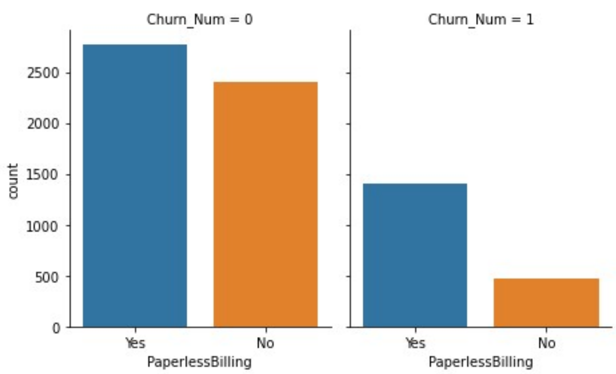

Churn_Num = 0
Churn_Num = 1
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection

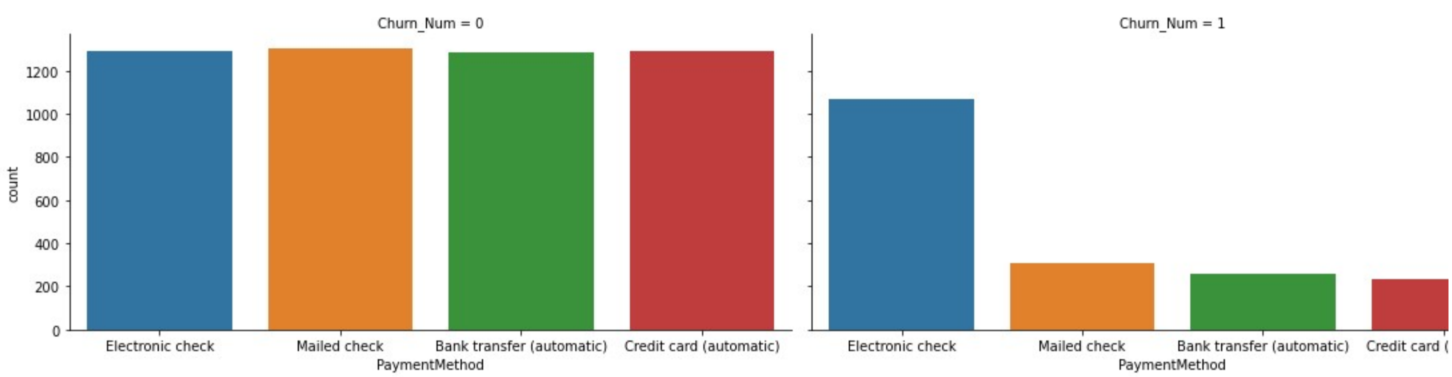

Churn_Num = 0
Churn_Num = 1
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
TechSupport


Churn_Num = 0
Churn_Num = 1
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingTV







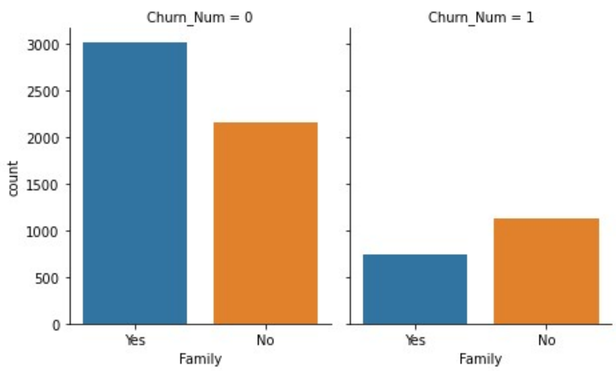


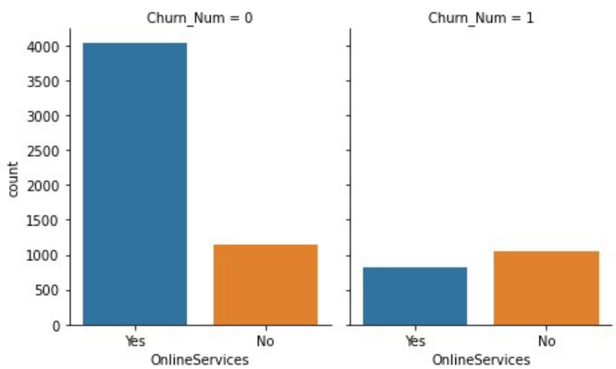

```
In [49]:
    ...:
    ...: datset_churn['MonthlyChargesRange'] = pd.cut(datset_churn['MonthlyCharges'], 5)
    ...: datset_churn[['MonthlyChargesRange',
'Churn_Num']].groupby(['MonthlyChargesRange'],
```

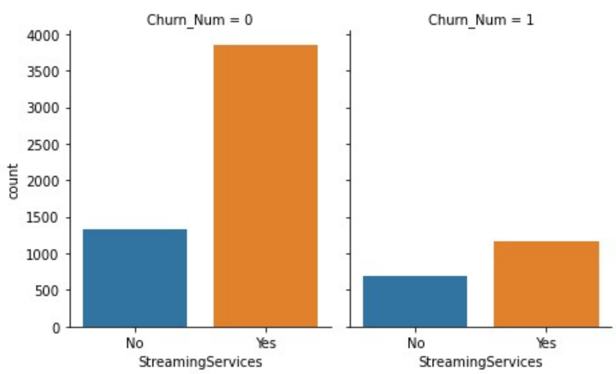
```
as_index=False).mean().sort_values(by='MonthlyChargesRange', ascending=True)
    ...:
    ...: # Replacing Age band with ordinals based on these bands
    ...: datset_churn.loc[ datset_churn['MonthlyCharges'] <= 20, 'MonthlyChargesCat'] =
0
    ...: datset_churn.loc[(datset_churn['MonthlyCharges'] > 20) &
(datset_churn['MonthlyCharges'] <= 40), 'MonthlyChargesCat'] = 1
    ...: datset_churn.loc[(datset_churn['MonthlyCharges'] > 40) &
(datset_churn['MonthlyCharges'] <= 60), 'MonthlyChargesCat'] = 2
    ...: datset_churn.loc[(datset_churn['MonthlyCharges'] > 60) &
(datset_churn['MonthlyCharges'] <= 80 ), 'MonthlyChargesCat'] = 3
    ...: datset_churn.loc[(datset_churn['MonthlyCharges'] > 80) &
(datset_churn['MonthlyCharges'] <= 100 ), 'MonthlyChargesCat'] = 4
    ...: datset_churn.loc[ datset_churn['MonthlyCharges'] > 100, 'MonthlyChargesCat'] =
5
    ...:
    ...: #Checking the categories
    ...:
datset_churn[['MonthlyCharges','MonthlyChargesRange','MonthlyChargesCat']].head(10)
    ...:
    ...: ##Considering  dependents and partners resprespnt same, creating family column
out of it and using it
    ...:
    ...: list_family = []
    ...: for rows in range(len(datset_churn['Partner'])):
    ...:     if ((datset_churn['Partner'][rows] == 'No') and (datset_churn['Dependents']
[rows] == 'No')):
    ...:         list_family.append('No')
    ...:     else:
    ...:         list_family.append('Yes')
    ...: datset_churn['Family'] = list_family
    ...: print(datset_churn[['Partner', 'Dependents', 'Family' ]].head(10))
    ...:
    ...: #Creating a new column for Online Services (Online Security & Online Backup) .
If a customer has Online Security or Online Backup services
    ...: #then , I am considering it as "Yes" else "No"
    ...: list_online_services = []
    ...: for rows_os in range(len(datset_churn['OnlineSecurity'])):
    ...:     if ((datset_churn['OnlineSecurity'][rows_os] == 'No') and
(datset_churn['OnlineBackup'][rows_os] == 'No')):
    ...:         list_online_services.append('No')
    ...:     else:
    ...:         list_online_services.append('Yes')
    ...: datset_churn['OnlineServices'] = list_online_services
    ...:
    ...: #print(datset_churn[['OnlineSecurity', 'OnlineBackup', 'OnlineServices'
]].head(10))
    ...:
    ...: #Creating a new column for Streaming Services (StreamingTV & StreamingMovies) .
If a customer has StreamingTV or StreamingMovies
    ...: #then , I am considering it as "Yes" else "No"
    ...: list_streaming_services = []
    ...: for rows_stv in range(len(datset_churn['StreamingTV'])):
    ...:     if ((datset_churn['StreamingTV'][rows_stv] == 'No') and
(datset_churn['StreamingMovies'][rows_stv] == 'No')):
    ...:         list_streaming_services.append('No')
    ...:     else:
    ...:         list_streaming_services.append('Yes')
```

```
    ...: datset_churn['StreamingServices'] = list_streaming_services
    ...:
    ...: #print(datset_churn[['StreamingTV', 'StreamingMovies', 'StreamingServices'
]].head(10))
    ...:
    ...: plot_cat_data = sns.catplot(x='Family', col='Churn_Num', data = datset_churn,
kind='count', height=4, aspect=0.8)
    ...: plot_cat_data = sns.catplot(x='OnlineServices', col='Churn_Num', data =
datset_churn, kind='count', height=4, aspect=0.8)
    ...: plot_cat_data = sns.catplot(x='StreamingServices', col='Churn_Num', data =
datset_churn, kind='count', height=4, aspect=0.8)
    ...:
    ...:
    ...: #Converting Gender column to numeric value
    ...: datset_churn['Gender'].unique() # Print unique values in the column
    ...: datset_churn['Gender_Num'] = datset_churn['Gender'].map( {'Female': 1, 'Male':
0} ).astype(int) #Map Categorical to Numerical Values
    ...: datset_churn[['Gender','Gender_Num']].head(2) # Test the mapping
    ...:
    ...:
    ...:
    ...:
    ...: # For Partner & Dependant , we created Family Column . Converting Family column
to numeric value
    ...: #datset_churn['Family'].unique() # Print unique values in the column
    ...: datset_churn['Family_Num'] = datset_churn['Family'].map( {'Yes': 1, 'No': 0}
).astype(int) #Map Categorical to Numerical Values
    ...: datset_churn[['Family','Family_Num']].head(2) # Test the mapping
    ...:
    ...: datset_churn['PhoneService_Num'] = datset_churn['PhoneService'].map( {'Yes': 1,
'No': 0} ).astype(int)
    ...: datset_churn['MultipleLines_Num'] = datset_churn['MultipleLines'].map( {'No':
0, 'Yes': 1, 'No phone service':2} ).astype(int)
    ...: datset_churn['InternetService_Num'] = datset_churn['InternetService'].map(
{'DSL': 0, 'Fiber optic': 1, 'No':2} ).astype(int)
    ...: datset_churn['OnlineServices_Num'] = datset_churn['OnlineServices'].map(
{'Yes': 1, 'No': 0} ).astype(int)
    ...:
    ...: datset_churn['DeviceProtection_Num'] = datset_churn['DeviceProtection'].map(
{'No': 0, 'Yes': 1, 'No internet service':2} ).astype(int)
    ...: datset_churn['StreamingServices_Num'] = datset_churn['StreamingServices'].map(
{'Yes': 1, 'No': 0} ).astype(int)
    ...: datset_churn['TechSupport_Num'] = datset_churn['TechSupport'].map( {'No': 0,
'Yes': 1, 'No internet service':2} ).astype(int)
    ...: datset_churn['Contract_Num'] = datset_churn['Contract'].map( {'Month-to-month':
0, 'One year': 1, 'Two year': 2} ).astype(int)
    ...: datset_churn['PaperlessBilling_Num'] = datset_churn['PaperlessBilling'].map(
{'Yes': 1, 'No': 0} ).astype(int)
    ...: datset_churn['PaymentMethod_Num'] = datset_churn['PaymentMethod'].map(
{'Electronic check': 0, 'Mailed check': 1, 'Bank transfer (automatic)': 2 , 'Credit card
(automatic)' : 3} ).astype(int)
    ...:
    ...:
    ...: # Take a copy of dataset
    ...: datset_churn_copy = datset_churn.copy()
  Partner Dependents Family
0     Yes         No    Yes
1      No         No     No
```

```
2       No          No      No
3       No          No      No
4       No          No      No
5       No          No      No
6       No         Yes     Yes
7       No          No      No
8      Yes          No     Yes
9       No         Yes     Yes
```








```
In [50]:
     ...: datset_churn['Gender'].unique() # Print unique values in the column
     ...: datset_churn['Gender_Num'] = datset_churn['Gender'].map( {'Female': 1, 'Male':
0} ).astype(int) #Map Categorical to Numerical Values
     ...: datset_churn[['Gender','Gender_Num']].head(2) # Test the mapping
     ...:
     ...:
     ...:
     ...:
```

```
    ...: # For Partner & Dependant , we created Family Column . Converting Family column to numeric value
    ...: #datset_churn['Family'].unique() # Print unique values in the column
    ...: datset_churn['Family_Num'] = datset_churn['Family'].map( {'Yes': 1, 'No': 0} ).astype(int) #Map Categorical to Numerical Values
    ...: datset_churn[['Family','Family_Num']].head(2) # Test the mapping
    ...:
    ...: datset_churn['PhoneService_Num'] = datset_churn['PhoneService'].map( {'Yes': 1, 'No': 0} ).astype(int)
    ...: datset_churn['MultipleLines_Num'] = datset_churn['MultipleLines'].map( {'No': 0, 'Yes': 1, 'No phone service':2} ).astype(int)
    ...: datset_churn['InternetService_Num'] = datset_churn['InternetService'].map( {'DSL': 0, 'Fiber optic': 1, 'No':2} ).astype(int)
    ...: datset_churn['OnlineServices_Num'] = datset_churn['OnlineServices'].map( {'Yes': 1, 'No': 0} ).astype(int)
    ...:
    ...: datset_churn['DeviceProtection_Num'] = datset_churn['DeviceProtection'].map( {'No': 0, 'Yes': 1, 'No internet service':2} ).astype(int)
    ...: datset_churn['StreamingServices_Num'] = datset_churn['StreamingServices'].map( {'Yes': 1, 'No': 0} ).astype(int)
    ...: datset_churn['TechSupport_Num'] = datset_churn['TechSupport'].map( {'No': 0, 'Yes': 1, 'No internet service':2} ).astype(int)
    ...: datset_churn['Contract_Num'] = datset_churn['Contract'].map( {'Month-to-month': 0, 'One year': 1, 'Two year': 2} ).astype(int)
    ...: datset_churn['PaperlessBilling_Num'] = datset_churn['PaperlessBilling'].map( {'Yes': 1, 'No': 0} ).astype(int)
    ...: datset_churn['PaymentMethod_Num'] = datset_churn['PaymentMethod'].map( {'Electronic check': 0, 'Mailed check': 1, 'Bank transfer (automatic)': 2 , 'Credit card (automatic)' : 3} ).astype(int)
    ...:
    ...:
    ...: # Take a copy of dataset
    ...: datset_churn_copy = datset_churn.copy()
    ...:
    ...:
    ...:
    ...: #Dropping the Categorical columns and keeping their equivalent numeric column
    ...: columns_to_drop = ['Gender', 'Partner', 'Dependents', 'Tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'TotalCharges', 'Churn', 'Family', 'OnlineServices', 'StreamingServices']
    ...: datset_churn = datset_churn.drop(columns_to_drop, axis=1)
    ...:
    ...: #Re-arranging the columns as per origial dataset
    ...: datset_churn = datset_churn[['CustomerID', 'Gender_Num', 'SeniorCitizen', 'Family_Num', 'TenureCat', 'PhoneService_Num', 'MultipleLines_Num', 'InternetService_Num', 'OnlineServices_Num', 'DeviceProtection_Num', 'TechSupport_Num', 'StreamingServices_Num', 'Contract_Num', 'PaperlessBilling_Num', 'PaymentMethod_Num', 'MonthlyChargesCat', 'Churn_Num']]
    ...: datset_churn = datset_churn.rename(columns={'Gender_Num' : 'Gender',
    ...:                                 'Family_Num' : 'Family',
    ...:                                 'PhoneService_Num' : 'PhoneService',
    ...:                                 'MultipleLines_Num': 'MultipleLines',
    ...:                                 'InternetService_Num' : 'InternetService',
    ...:                                 'OnlineServices_Num' : 'OnlineServices',
    ...:                                 'DeviceProtection_Num' : 'DeviceProtection',
    ...:                                 'TechSupport_Num' : 'TechSupport',
```

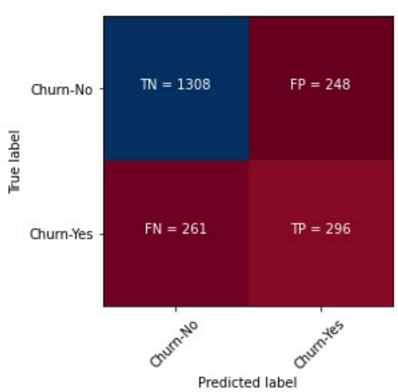
```
    ...:                                  'StreamingServices_Num' : 'StreamingServices',
    ...:                                  'Contract_Num' : 'Contract',
    ...:                                  'PaperlessBilling_Num' : 'PaperlessBilling',
    ...:                                  'PaymentMethod_Num' : 'PaymentMethod',
    ...:                                  'MonthlyCharges' : 'MonthlyCharges',
    ...:                                  'Churn_Num' :  'Churn' })
    ...: datset_churn.columns
    ...:
    ...:
    ...: ##Finally data set after modifying the data set and keeping numerical colums
and removiong actual categorical columns is found in " datset_churn"
    ...: ##The copy of dataset with numeric and categorical columns is found in "
datset_churn_copy"
    ...: ###Correlation
Out[50]:
Index(['CustomerID', 'Gender', 'SeniorCitizen', 'Family', 'TenureCat',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineServices',
       'DeviceProtection', 'TechSupport', 'StreamingServices', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyChargesCat', 'Churn'],
      dtype='object')

In [51]:
```

```
In [33]:
    ...:
    ...:
    ...: from sklearn.metrics import confusion_matrix # Library for model evaluation
    ...: from sklearn.metrics import accuracy_score # Library for model evaluation
    ...: from sklearn.model_selection import train_test_split # Library to split datset
into test and train
    ...:
    ...: from sklearn.linear_model  import LogisticRegression # Logistic Regression
Classifier
    ...: from sklearn.linear_model import SGDClassifier # Stochastic Gradient Descent
Classifier
    ...: from sklearn.tree import DecisionTreeClassifier # Decision Tree Classifier
    ...: from sklearn.ensemble  import RandomForestClassifier # Random Forest Classifier
    ...: from sklearn.neighbors import KNeighborsClassifier # K Nearest neighbors
Classifier
    ...: from sklearn.naive_bayes import GaussianNB #Naive Bayes Classifier
    ...: from sklearn.svm import SVC #Support vector Machine Classifier
    ...: from sklearn.ensemble import AdaBoostClassifier # Ada Boost Classifier
    ...: from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, classification_report, confusion_matrix
    ...: from sklearn.model_selection import cross_val_score
    ...: from sklearn.metrics import precision_recall_curve
    ...: from sklearn.metrics import average_precision_score
    ...:
    ...: X = datset_churn.iloc[:,1:16].values # Feature Variable
    ...: y = datset_churn.iloc[:,16].values # Target Variable

In [34]:
    ...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
    ...: print('There are {} samples in the training set and {} samples in the test
set'.format(X_train.shape[0], X_test.shape[0]))
    ...:
    ...: #Creating function for Confusion Matrix , Precsion, Recall and F1 Score
    ...: def plot_confusion_matrix(classifier, y_test, y_pred_test):
    ...:     cm = confusion_matrix(y_test, y_pred_test)
    ...:
    ...:
    ...:
    ...:     print("\n",classifier,"\n")
    ...:     plt.clf()
    ...:     plt.imshow(cm, interpolation='nearest', cmap='RdBu')
    ...:     classNames = ['Churn-No','Churn-Yes']
    ...:     plt.ylabel('True label')
    ...:     plt.xlabel('Predicted label')
    ...:     tick_marks = np.arange(len(classNames))
    ...:     plt.xticks(tick_marks, classNames, rotation=45)
    ...:     plt.yticks(tick_marks, classNames)
    ...:     s = [['TN','FP'], ['FN', 'TP']]
    ...:
    ...:     for i in range(2):
    ...:         for j in range(2):
    ...:             plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]),
    ...:                      horizontalalignment='center', color='White')
    ...:
    ...:     plt.show()
```

```
   ...:
   ...:     tn, fp, fn, tp = cm.ravel()
   ...:
   ...:     recall = tp / (tp + fn)
   ...:     precision = tp / (tp + fp)
   ...:     F1 = 2*recall*precision/(recall+precision)
   ...:     print('\n\n')
   ...:     print("Classification Report")
   ...:     print(classification_report(y_test,y_pred_test))
   ...:
   ...:     print('\n THe overall performance factors:\n')
   ...:     print("Number of mislabeled points out of a total %d points : %d" %
(len(y_test), (y_test != y_pred_test).sum()))
   ...:
print('Recall={0:0.3f}'.format(recall),'\nPrecision={0:0.3f}'.format(precision))
   ...:     print('F1={0:0.3f}'.format(F1))
   ...:
   ...:     print(f"Error rate : {(1.-sum(np.diag(cm))/cm.sum()):5.3}")
   ...:     return;
There are 4930 samples in the training set and 2113 samples in the test set

In [35]: def plot_prec_rec_curve(classifier, y_test, y_pred_score):
   ...:     precision, recall, _ = precision_recall_curve(y_test, y_pred_score)
   ...:     average_precision = average_precision_score(y_test, y_pred_score)
   ...:
   ...:     print('Average precision-recall score: {0:0.3f}'.format(
   ...:           average_precision))
   ...:
   ...:     plt.plot(recall, precision, label='area = %0.3f' % average_precision,
color="green")
   ...:     plt.xlim([0.0, 1.0])
   ...:     plt.ylim([0.0, 1.05])
   ...:     plt.xlabel('Recall')
   ...:     plt.ylabel('Precision')
   ...:     plt.title('Precision Recall Curve')
   ...:     plt.legend(loc="best")
   ...:     plt.show()
   ...:
   ...: classifier_model = [KNeighborsClassifier(),RandomForestClassifier()]
   ...: import matplotlib.pyplot as plt

In [36]:
   ...: classifier_model_list= []
   ...: classifier_accuracy_test = []
   ...: classifier_accuracy_train = []
   ...: f1score = []
   ...: precisionscore = []
   ...: recallscore = []
   ...: avg_pre_rec_score = []
   ...: cv_score = []

In [37]: for classifier_list in classifier_model:
   ...:     classifier = classifier_list
   ...:
   ...:     # Fitting the training set into classification model
   ...:     classifier.fit(X_train,y_train)
   ...:
   ...:     # Predicting the output on test datset
```

```
   ...:     y_pred_test = classifier.predict(X_test)
   ...:     score_test = accuracy_score(y_test, y_pred_test)
   ...:
   ...:     # Predicting the output on training datset
   ...:     y_pred_train = classifier.predict(X_train)
   ...:     score_train = accuracy_score(y_train, y_pred_train)
   ...:
   ...:     # Cross Validation Score on training test
   ...:     scores = cross_val_score(classifier, X_train,y_train, cv=10)
   ...:     cv_score.append(scores.mean())
   ...:
   ...:       #Keeping the model and accuracy score into a list
   ...:     classifier_model_list.append(classifier_list.__class__.__name__)
   ...:     classifier_accuracy_test.append(round(score_test,4))
   ...:     classifier_accuracy_train.append(round(score_train,4))
   ...:
   ...:     #Precision, Recall and F1 score
   ...:     f1score.append(f1_score(y_test, y_pred_test))
   ...:     precisionscore.append(precision_score(y_test, y_pred_test))
   ...:     recallscore.append(recall_score(y_test, y_pred_test))
   ...:
   ...:     #Calculating Average Precision Recall Score
   ...:     try:
   ...:         y_pred_score = classifier.decision_function(X_test)
   ...:     except:
   ...:         y_pred_score = classifier.predict_proba(X_test)[:,1]
   ...:
   ...:     from sklearn.metrics import average_precision_score
   ...:     average_precision = average_precision_score(y_test, y_pred_score)
   ...:     avg_pre_rec_score.append(average_precision)
   ...:
   ...:
   ...:     #Confusion Matrix
   ...:     plot_confusion_matrix(classifier_list.__class__.__name__, y_test,
y_pred_test)
   ...:     plot_prec_rec_curve(classifier_list.__class__.__name__, y_test,
y_pred_score)
```

KNeighborsClassifier

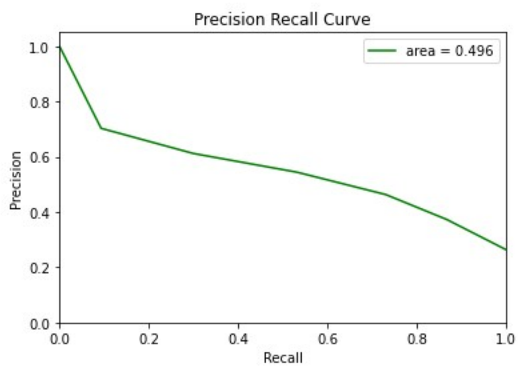


Classification Report

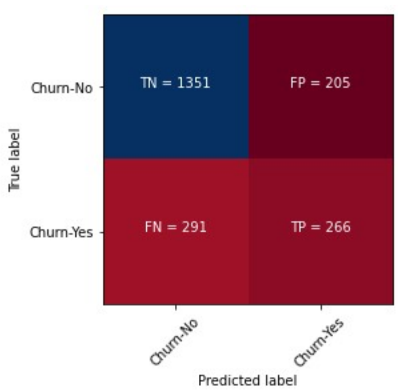
	precision	recall	f1-score	support
0	0.83	0.84	0.84	1556
1	0.54	0.53	0.54	557
accuracy			0.76	2113
macro avg	0.69	0.69	0.69	2113
weighted avg	0.76	0.76	0.76	2113

THe overall performance factors:

Number of mislabeled points out of a total 2113 points : 509
Recall=0.531
Precision=0.544
F1=0.538
Error rate : 0.241
Average precision-recall score: 0.496



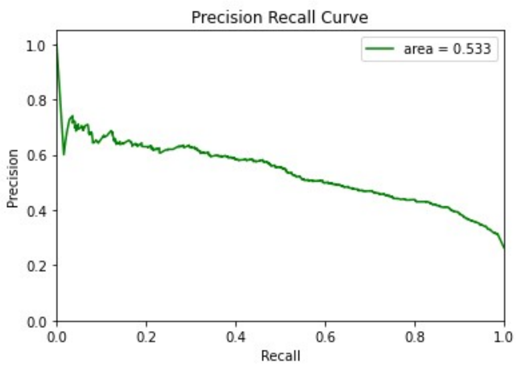
RandomForestClassifier



Classification Report

	precision	recall	f1-score	support
0	0.82	0.87	0.84	1556
1	0.56	0.48	0.52	557
accuracy			0.77	2113

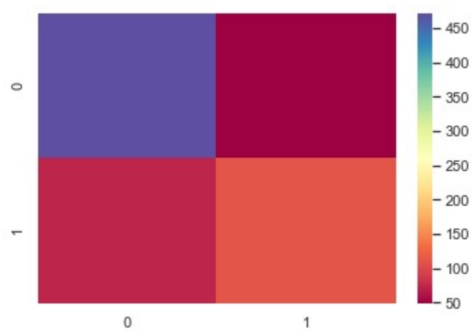
```
   macro avg       0.69      0.67      0.68      2113
weighted avg       0.75      0.77      0.76      2113


 THe overall performance factors:

Number of mislabeled points out of a total 2113 points : 496
Recall=0.478
Precision=0.565
F1=0.518
Error rate : 0.235
Average precision-recall score: 0.533
```



In [38]:

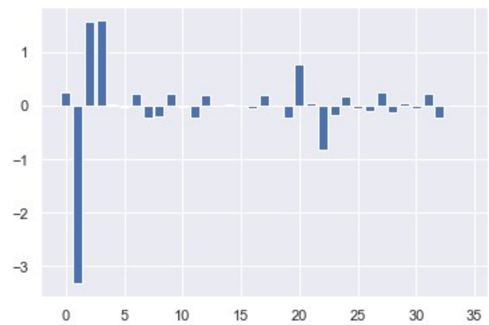
```
In [9]: drop_LIST =
["CustomerID",'Partner','Dependents','OnlineSecurity','OnlineBackup',\
   ...:
'StreamingTV','StreamingMovies','Churn_Num','TenureRange','TenureCat',\
   ...:                     'MonthlyChargesRange','MonthlyChargesCat']
   ...: churn_ = datset_churn.drop(drop_LIST, axis=1)
   ...:
   ...: from sklearn.metrics import confusion_matrix # Library for model evaluation
   ...: from sklearn.metrics import accuracy_score # Library for model evaluation
   ...: from sklearn.model_selection import train_test_split
   ...: import matplotlib.pyplot as plt
   ...: from sklearn.metrics import classification_report, confusion_matrix
   ...: import seaborn as sns
   ...: import numpy as np
   ...: from sklearn.linear_model import LogisticRegression
   ...: from sklearn.preprocessing import LabelEncoder # Encode Categorical Variable to
Numerical Variable
   ...:
   ...: X = churn_.drop('Churn', axis=1)
   ...: y = churn_['Churn']

In [10]:
    ...:
    ...:
    ...:
    ...: model = LogisticRegression()
    ...:
    ...: from sklearn import preprocessing
    ...: colName = ['MonthlyCharges', 'TotalCharges', 'Tenure']
    ...: xo = X[colName].values
    ...: x_scaled = preprocessing.MinMaxScaler().fit_transform(xo)
    ...: df_temp = pd.DataFrame(x_scaled)#, columns=colName, index = xo[colName].index)
    ...: X[colName] = df_temp
    ...:
    ...: #X  = X.apply(LabelEncoder().fit_transform) # we can use label encoder to get
numeric encoded X instead of inserting dummy columns
    ...:
    ...: X_trainX, X_testX, y_train, y_test = train_test_split(X, y, test_size=.1,
random_state=20)#, random_state=None)
    ...:
    ...: # get dummies with panda, we can also use One Hot Encoding, or label encoder
    ...: X_train = pd.get_dummies(X_trainX)
    ...: X_test = pd.get_dummies(X_testX)
    ...:
    ...: # For y-values we will use LabelEncoder
    ...: label_enc  = LabelEncoder()
    ...: y_train = label_enc.fit_transform(y_train)
    ...: y_test = label_enc.fit_transform(y_test)
    ...:
    ...: LogRegModel=model.fit(X_train, y_train)
    ...: y_pred=LogRegModel.predict(X_test)
    ...:
    ...: print("Number of mislabeled points out of a total %d points : %d" %
(X_test.shape[0], (y_test != y_pred).sum()))
    ...: C=confusion_matrix(y_test, y_pred)
    ...: print(classification_report(y_test,y_pred))
```

```
    ...: print("The confusion matrix:\n", C)
    ...: sns.heatmap(C, cmap="Spectral")
    ...: plt.show()
    ...:
    ...: print("Error rate: %5.3f"%   (1.-sum(np.diag(C))/C.sum()))
    ...: predProb=LogRegModel.predict_proba(X_test)
    ...:
    ...: #------------------------------------------------------------
    ...: importance = LogRegModel.coef_[0]
    ...: # summarize feature importance
    ...: for i,v in enumerate(importance):
    ...:    print('Feature: %0d, Score: %.5f' % (i,v))
    ...:
    ...: # plot feature importance
    ...: plt.bar([x for x in range(len(importance))], importance)
    ...: plt.show()
Number of mislabeled points out of a total 705 points : 121
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       520
           1       0.70      0.61      0.65       185

    accuracy                           0.83       705
   macro avg       0.78      0.76      0.77       705
weighted avg       0.82      0.83      0.82       705

The confusion matrix:
 [[472  48]
 [ 73 112]]
```

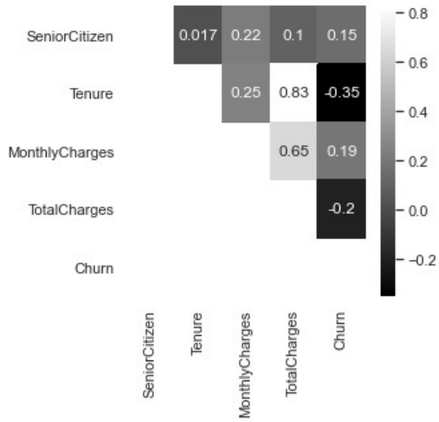



```
Error rate: 0.172
Feature: 0, Score: 0.24460
Feature: 1, Score: -3.32275
Feature: 2, Score: 1.57264
Feature: 3, Score: 1.60352
Feature: 4, Score: 0.02195
Feature: 5, Score: -0.02423
Feature: 6, Score: 0.21856
Feature: 7, Score: -0.22083
Feature: 8, Score: -0.20481
Feature: 9, Score: 0.21856
Feature: 10, Score: -0.01602
Feature: 11, Score: -0.21393
Feature: 12, Score: 0.19949
Feature: 13, Score: 0.01216
Feature: 14, Score: 0.02081
Feature: 15, Score: 0.01216
```

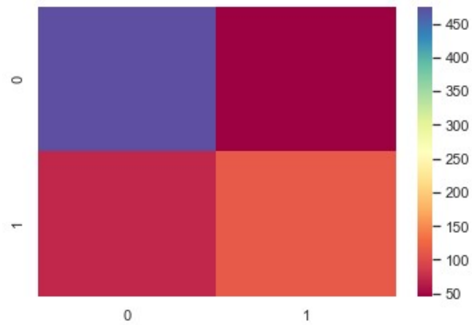
```
Feature: 16, Score: -0.03524
Feature: 17, Score: 0.21009
Feature: 18, Score: 0.01216
Feature: 19, Score: -0.22452
Feature: 20, Score: 0.76266
Feature: 21, Score: 0.05484
Feature: 22, Score: -0.81978
Feature: 23, Score: -0.18003
Feature: 24, Score: 0.17776
Feature: 25, Score: -0.03888
Feature: 26, Score: -0.09140
Feature: 27, Score: 0.24295
Feature: 28, Score: -0.11494
Feature: 29, Score: 0.03994
Feature: 30, Score: -0.04222
Feature: 31, Score: 0.22181
Feature: 32, Score: -0.22409
Feature: 33, Score: -0.00797
Feature: 34, Score: 0.00569
```




```
In [11]:
    ...:
    ...: y_ = y.map({'Yes' : 1, 'No' : 0})
    ...: X_ = X.join(y_)
    ...:
    ...: plt.figure(figsize=(4,4))
    ...: sns.set(font_scale=1)
    ...: mask = np.zeros_like(X_.corr())
    ...: mask[np.tril_indices_from(mask)] = True
    ...: with sns.axes_style("white"):
    ...:     sns.heatmap(X_.corr(), mask=mask, annot=True, cmap="gist_gray")
    ...: plt.show()
```

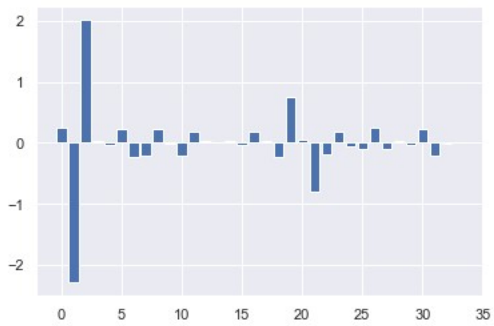



```
In [12]:
    ...:
    ...: XX = X.drop('TotalCharges', axis=1)
    ...: XX_trainX, XX_testX, yy_train, yy_test = train_test_split(XX, y, test_size=.1,
random_state=20)#, random_state=None)
    ...:
    ...: # One Hot Encoding
    ...: XX_train = pd.get_dummies(XX_trainX)
    ...: XX_test = pd.get_dummies(XX_testX)
    ...: yy_train = label_enc.fit_transform(yy_train)
    ...: yy_test = label_enc.fit_transform(yy_test)
    ...:
    ...: LogRegModel2=model.fit(XX_train, yy_train)
    ...: yy_pred=LogRegModel2.predict(XX_test)
    ...:
    ...: print("Number of mislabeled points out of a total %d points : %d" %
(XX_test.shape[0], (yy_test != yy_pred).sum()))
    ...: C=confusion_matrix(yy_test, yy_pred)
    ...: print(classification_report(yy_test,yy_pred))
    ...: print("The confusion matrix:\n", C)
    ...: sns.heatmap(C, cmap="Spectral")
    ...: plt.show()
    ...:
    ...: print("Error rate: %5.3f"%   (1.-sum(np.diag(C))/C.sum()))
    ...: predProb=LogRegModel2.predict_proba(XX_test)
Number of mislabeled points out of a total 705 points : 117
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       520
           1       0.72      0.61      0.66       185

    accuracy                           0.83       705
   macro avg       0.79      0.76      0.77       705
weighted avg       0.83      0.83      0.83       705

The confusion matrix:
 [[475  45]
 [ 72 113]]
```



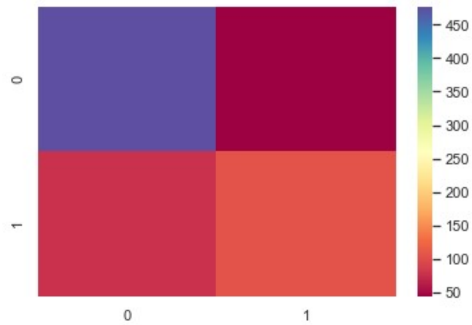

```
Error rate: 0.166

In [13]:
    ...: importance2 = LogRegModel2.coef_[0]
    ...: for i,v in enumerate(importance2):
    ...:    print('Feature: %0d, Score: %.5f' % (i,v))
    ...: plt.bar([x for x in range(len(importance2))], importance2)
    ...: plt.show()
    ...:
    ...: #==============================================================================
    ...: from sklearn.model_selection import cross_val_score
    ...: from sklearn.model_selection import ShuffleSplit
    ...: n_samples = XX.shape[0]
    ...: cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=20)
    ...:
    ...: XX_  = XX.apply(LabelEncoder().fit_transform)
    ...: #y_ = label_enc.fit_transform(yy)
    ...: scores = cross_val_score(LogRegModel2, XX_, y, cv=cv)
    ...: print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
Feature: 0, Score: 0.24774
Feature: 1, Score: -2.29535
Feature: 2, Score: 2.02900
Feature: 3, Score: 0.02270
Feature: 4, Score: -0.02212
Feature: 5, Score: 0.22239
Feature: 6, Score: -0.22181
Feature: 7, Score: -0.21212
Feature: 8, Score: 0.22239
Feature: 9, Score: -0.00969
Feature: 10, Score: -0.21623
Feature: 11, Score: 0.18355
Feature: 12, Score: 0.03326
Feature: 13, Score: 0.00525
Feature: 14, Score: 0.03326
Feature: 15, Score: -0.03793
Feature: 16, Score: 0.19643
Feature: 17, Score: 0.03326
Feature: 18, Score: -0.22911
Feature: 19, Score: 0.75492
Feature: 20, Score: 0.05271
Feature: 21, Score: -0.80705
Feature: 22, Score: -0.17795
Feature: 23, Score: 0.17853
Feature: 24, Score: -0.04515
Feature: 25, Score: -0.09859
Feature: 26, Score: 0.24081
Feature: 27, Score: -0.09649
```

```
Feature: 28, Score: 0.04183
Feature: 29, Score: -0.04125
Feature: 30, Score: 0.21898
Feature: 31, Score: -0.21840
Feature: 32, Score: -0.00073
Feature: 33, Score: 0.00130
```

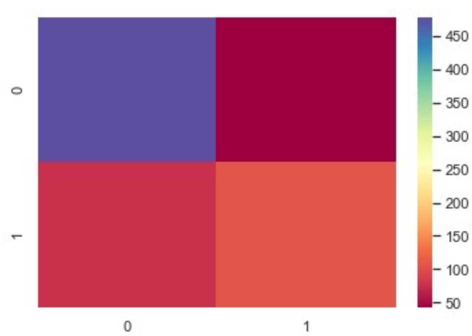


```
Accuracy: 0.81 (+/- 0.01)
```

In [14]:
```
    ...:
    ...:
    ...:
    ...:
    ...: from sklearn.linear_model import RidgeClassifier
    ...: RidgeClassModel2=RidgeClassifier().fit(XX_train, yy_train)
    ...: yy_pred2=RidgeClassModel2.predict(XX_test)
    ...:
    ...: print("Number of mislabeled points out of a total %d points : %d" %
(XX_test.shape[0], (yy_test != yy_pred2).sum()))
    ...: C=confusion_matrix(yy_test, yy_pred2)
    ...: print(classification_report(yy_test,yy_pred2))
    ...: print("The confusion matrix:\n", C)
    ...: sns.heatmap(C, cmap="Spectral")
    ...: plt.show()
    ...: print("Error rate: %5.3f"%   (1.-sum(np.diag(C))/C.sum()))
    ...:
    ...: scores = cross_val_score(RidgeClassModel2, XX_, y, cv=cv)
    ...: print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
Number of mislabeled points out of a total 705 points : 121
              precision    recall  f1-score   support

           0       0.86      0.92      0.89       520
           1       0.71      0.58      0.64       185

    accuracy                           0.83       705
   macro avg       0.79      0.75      0.76       705
weighted avg       0.82      0.83      0.82       705

The confusion matrix:
 [[477  43]
 [ 78 107]]
```

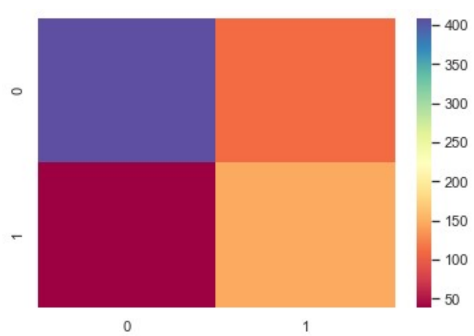



```
Error rate: 0.172
Accuracy: 0.80 (+/- 0.01)

In [15]:
    ...: RidgeClassModel1=RidgeClassifier().fit(X_train, y_train)
    ...: y_pred2=RidgeClassModel1.predict(X_test)
    ...: print("Number of mislabeled points out of a total %d points : %d" %
(X_test.shape[0], (y_test != y_pred2).sum()))
    ...: C=confusion_matrix(y_test, y_pred2)
    ...: print(classification_report(y_test,y_pred2))
    ...: print("The confusion matrix:\n", C)
    ...: sns.heatmap(C, cmap="Spectral")
    ...: plt.show()
    ...: print("Error rate: %5.3f"%   (1.-sum(np.diag(C))/C.sum()))
    ...: scores = cross_val_score(RidgeClassModel1,
X.apply(LabelEncoder().fit_transform), y, cv=cv)
    ...: print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
Number of mislabeled points out of a total 705 points : 119
              precision    recall  f1-score   support

           0       0.86      0.92      0.89       520
           1       0.72      0.58      0.64       185

    accuracy                           0.83       705
   macro avg       0.79      0.75      0.77       705
weighted avg       0.82      0.83      0.83       705

The confusion matrix:
 [[478  42]
 [ 77 108]]
```




```
Error rate: 0.169
Accuracy: 0.81 (+/- 0.02)

In [16]:
    ...:
```

```
    ...:
    ...: import sklearn.naive_bayes as skb
    ...: model2 = skb.GaussianNB()
    ...:
    ...: X00 = X.apply(LabelEncoder().fit_transform) # we can use label encoder to get
numeric encoded X instead of inserting dummy columns
    ...: # X00 = XX.apply(LabelEncoder().fit_transform)
    ...:
    ...: X00_train, X00_test, y00_train, y00_test =
train_test_split(X00,y,test_size=0.1,random_state=20)
    ...: GaussModel=model2.fit(X00_train, y00_train)
    ...: y00_pred=GaussModel.predict(X00_test)
    ...:
    ...: print("Number of mislabeled points out of a total %d points : %d" %
(X00_test.shape[0], (y00_test != y00_pred).sum()))
    ...: C=confusion_matrix(y00_test, y00_pred)
    ...: print(classification_report(y00_test,y00_pred))
    ...: print("The confusion matrix:\n", C)
    ...: sns.heatmap(C, cmap="Spectral")
    ...: plt.show()
    ...:
    ...: print("Error rate: %5.3f"%   (1.-sum(np.diag(C))/C.sum()))
    ...: predProb=GaussModel.predict_proba(X00_test)
    ...:
    ...: #-------------------------------------------------------------
    ...: n_samples = X00.shape[0]
    ...: cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=20)
    ...: scores = cross_val_score(LogRegModel2, X00, y, cv=cv)
    ...: print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
    ...:
    ...: #==========================================================================
    ...: ####################_____________ END ______________#########################
Number of mislabeled points out of a total 705 points : 149
              precision    recall  f1-score   support

          No       0.91      0.79      0.85       520
         Yes       0.57      0.79      0.66       185

    accuracy                           0.79       705
   macro avg       0.74      0.79      0.75       705
weighted avg       0.82      0.79      0.80       705

The confusion matrix:
 [[409 111]
 [ 38 147]]
```




```
Error rate: 0.211
```

```
Accuracy: 0.81 (+/- 0.02)
```

In [**17**]: