Customer Churn Prediction

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

In [2]:
```
# Reading the telecom customer churn data from a CSV file
telecom_churn_data = pd.read_csv('Telecom-Customer-Churn.csv')
```

In [3]:
```
# Displaying the first 5 rows of the telecom customer churn data
telecom_churn_data.head(5)
```

Out[3]:

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

5 rows × 21 columns

In [4]:
```
# Getting the dimensions of the telecom customer churn data
telecom_churn_data.shape
```

Out[4]:
```
(7043, 21)
```

In [5]:
```
# Getting the column names of the telecom customer churn data
telecom_churn_data.columns.values
```

Out[5]:
```
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 [6]:
```
# Getting the data types of the columns in the telecom customer churn data
telecom_churn_data.dtypes
```

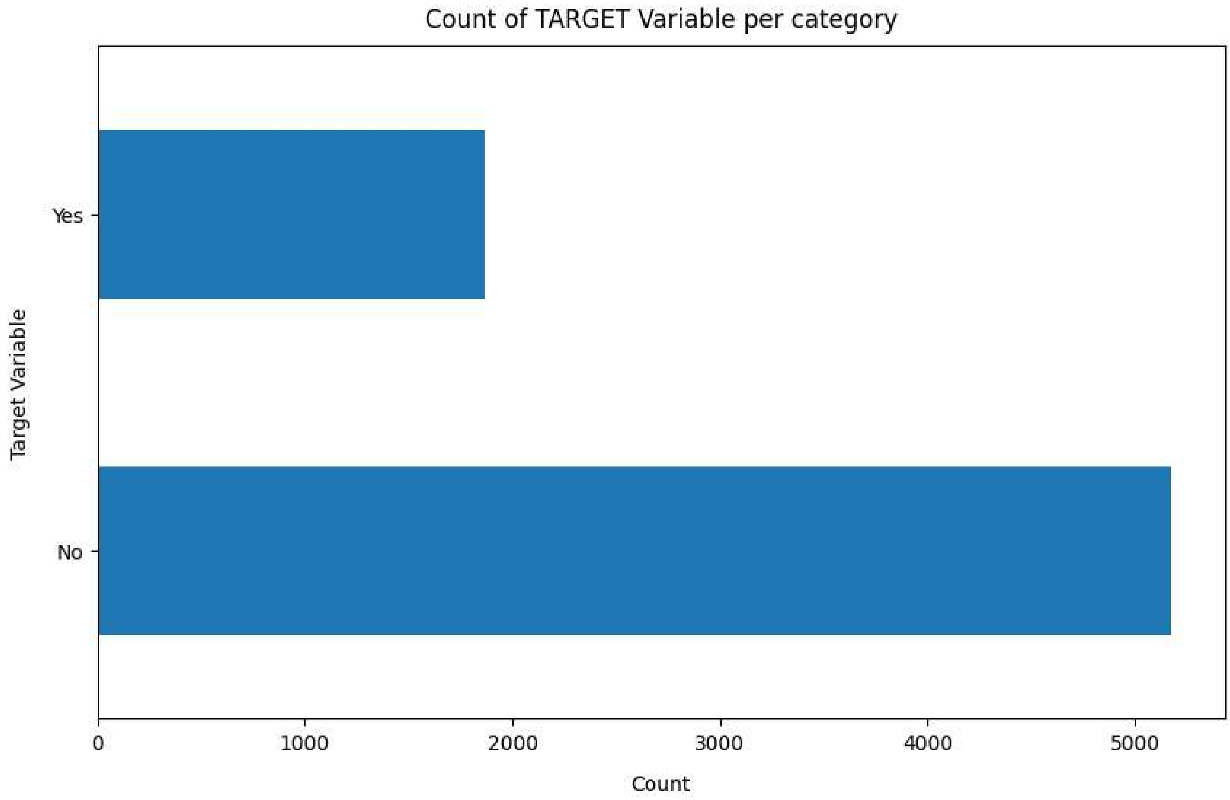
Out[6]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

In [7]:
```
# Generating descriptive statistics for the numeric columns in the telecom customer churn data
telecom_churn_data.describe()
```

Out[7]:

	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 [8]:
```
# Plotting the count of the 'Churn' variable
telecom_churn_data['Churn'].value_counts().plot(kind='barh', figsize=(10, 6))
plt.xlabel("Count", labelpad=10)
plt.ylabel("Target Variable", labelpad=10)
plt.title("Count of TARGET Variable per category", y=1.01);
```



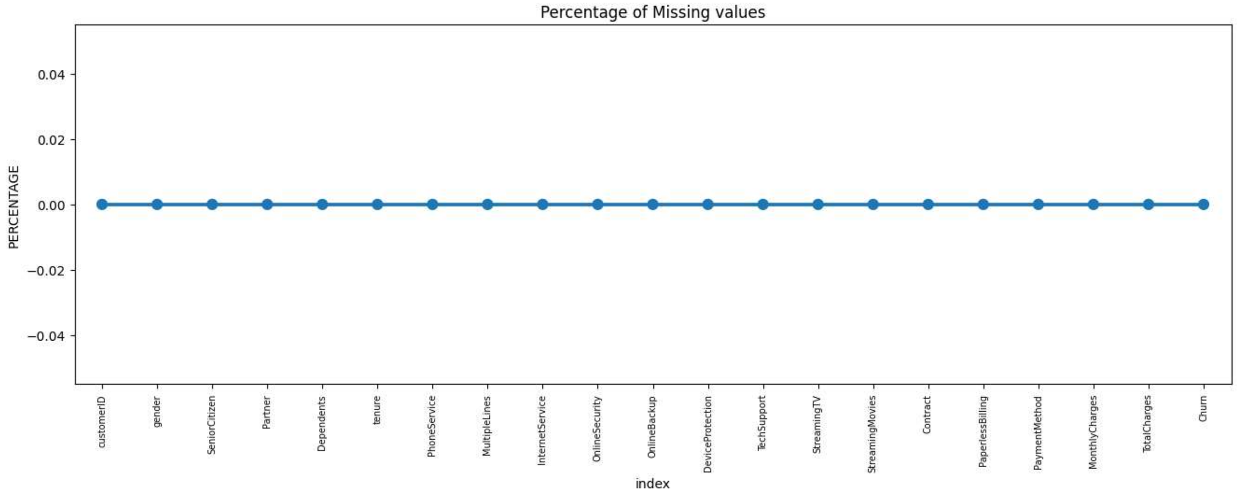
In [9]:
```
# Counting the occurrences of each category in the 'Churn' variable
telecom_churn_data['Churn'].value_counts()
```

Out[9]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [10]:
```
# Printing the summary of the telecom customer churn data
telecom_churn_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [11]:
```
# Calculating the percentage of missing values in each column
missing = pd.DataFrame((telecom_churn_data.isnull().sum()) * 100 / telecom_churn_data.shape[0]).reset_in
plt.figure(figsize=(16, 5))
ax = sns.pointplot(x='index', y=0, data=missing)  # Specifying 'x' and 'y' for clarity
plt.xticks(rotation=90, fontsize=7)
plt.title("Percentage of Missing values")
plt.ylabel("PERCENTAGE")
plt.show()
```



Data Exploration and Preprocessing

In [12]:
```
# Making a copy of data
telecom_data=telecom_churn_data.copy()
```

In [13]:
```
# Converting the data type of the Total Charges column to numeric
telecom_data.TotalCharges=pd.to_numeric(telecom_data.TotalCharges, errors='coerce')

# Checking for missing values after the conversion
telecom_data.isnull().sum()
```

Out[13]:
```
customerID           0
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

In [14]:
```
# Checking the null value records
telecom_data.loc[telecom_data['TotalCharges'].isnull()==True]
```

Out[14]:

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

11 rows × 21 columns

In [15]:
```
# Removing the missing records
telecom_data.dropna(how='any',inplace=True)
```

In [16]:
```
# Printing the maximum value in the 'tenure' column
print(telecom_data['tenure'].max())
```

```
72
```

In [17]:
```
# Grouping the tenure in classes of 12 months
labels = ["{0} - {1}".format(i, i + 11) for i in range(1, 72, 12)]

telecom_data['tenure_group'] = pd.cut(telecom_data.tenure, range(1, 80, 12), right=False, labels=labels)
```

In [18]:
```
# Counting the frequency of each unique value in the 'tenure_group' column
telecom_data['tenure_group'].value_counts()
```

Out[18]:
```
1 - 12     2175
61 - 72    1407
13 - 24    1024
25 - 36     832
49 - 60     832
37 - 48     762
Name: tenure_group, dtype: int64
```

In [19]:
```
# Drop columns customerID and tenure which are not required for the study
telecom_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telecom_data.head()
```

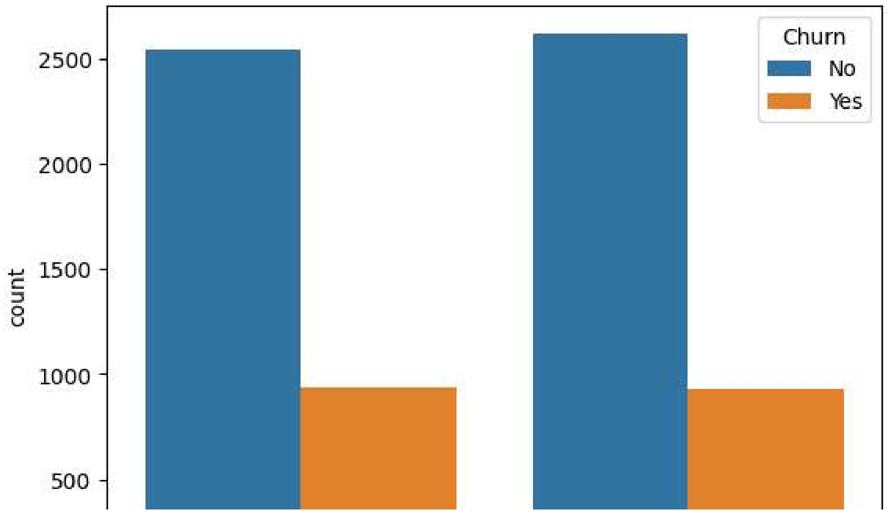
Out[19]:

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

Data Analysis & Feature Engineering

Univariate Analysis

In [20]:
```
# Iterating over the predictor variables and plotting distibution of individual predictors by churn
for i, predictor in enumerate(telecom_data.drop(columns=['Churn', 'TotalCharges', 'MonthlyCharges'])):
    plt.figure(i)
    sns.countplot(data=telecom_data, x=predictor, hue='Churn')
```



In [21]:
```
#Converting the target variable 'Churn' in a binary numeric variable such as Yes=1 ; No=0
telecom_data['Churn'] = np.where(telecom_data.Churn == 'Yes',1,0)
```

In [22]:
```
# Displaying the first 5 rows of the 'telecom_data' dataset
telecom_data.head()
```

Out[22]:

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

In [23]:
```
#Converting all the categorical variables into dummy variables
telecom_data_dummies = pd.get_dummies(telecom_data)

# Displaying the first 5 rows
telecom_data_dummies.head()
```

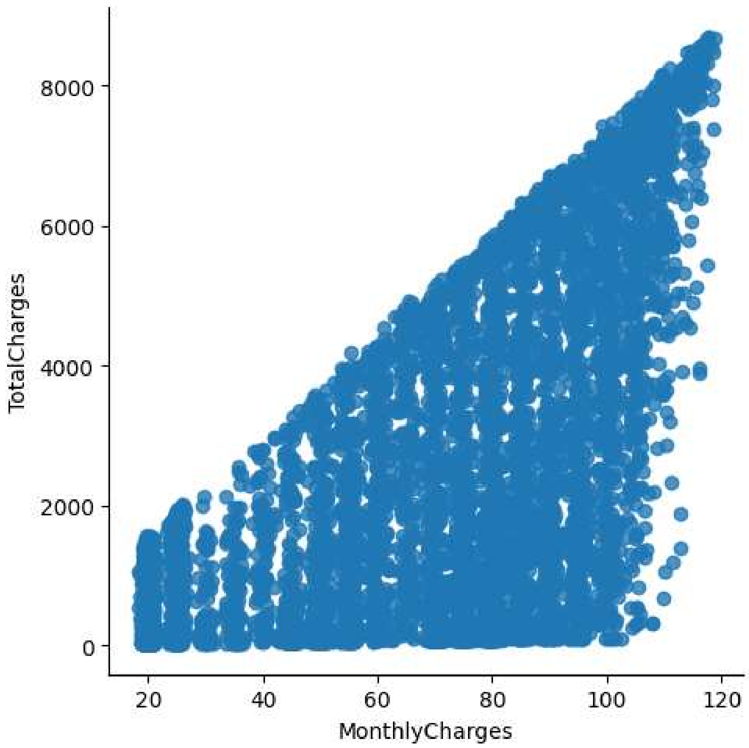
Out[23]:

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

5 rows × 51 columns

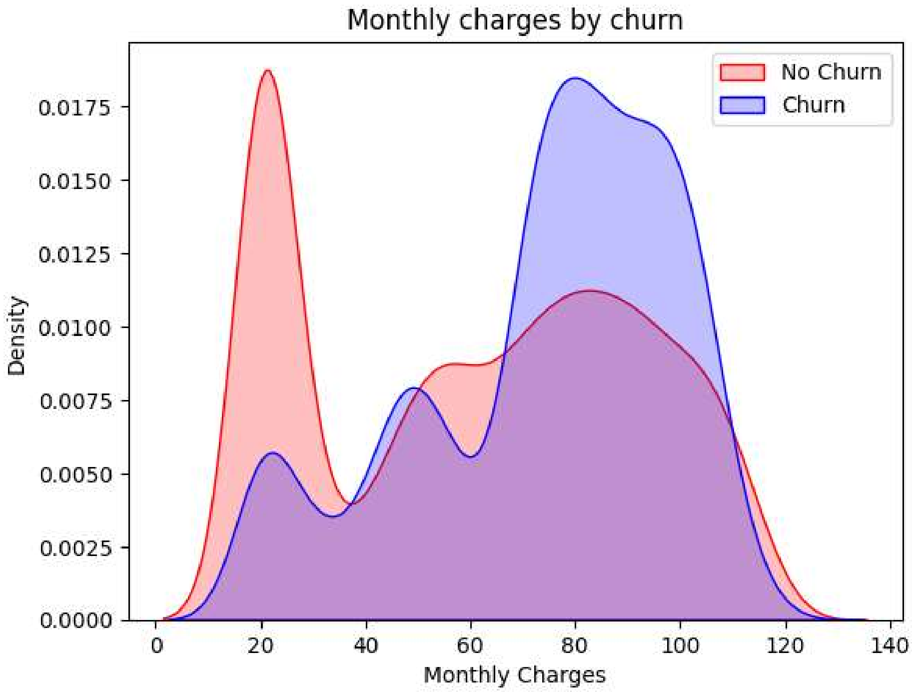
In [24]:
```
# Creating the relationship between Monthly Charges and Total Charges in the telecom_data_dummies datase
sns.lmplot(data=telecom_data_dummies, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```

Out[24]: <seaborn.axisgrid.FacetGrid at 0x2c74ef92f88>



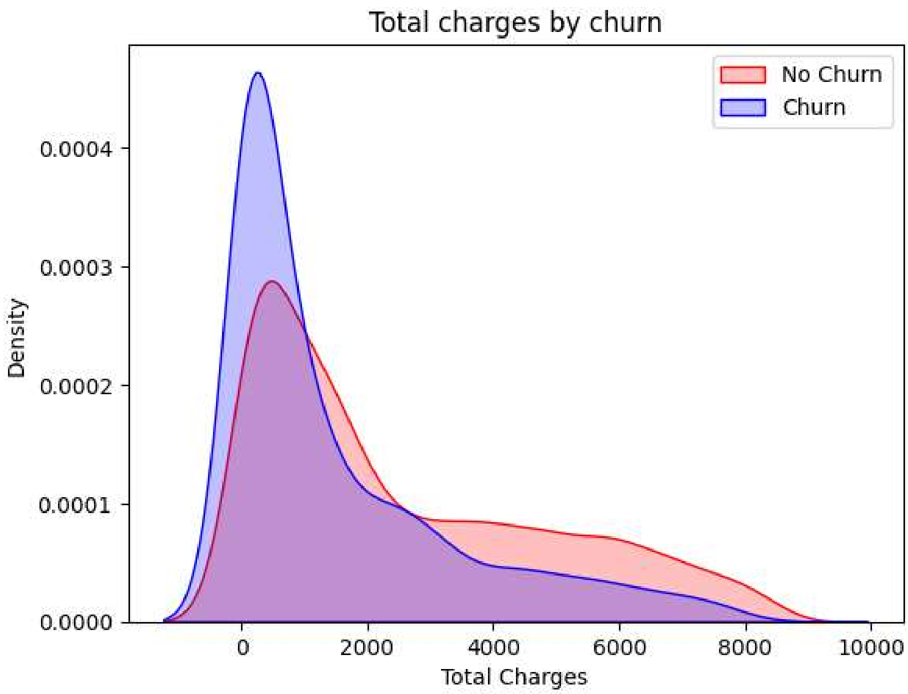
In [25]:
```
# Creating a KDE plot to compare the distribution of monthly charges for churned and non-churned custome
Mth = sns.kdeplot(telecom_data_dummies.MonthlyCharges[(telecom_data_dummies["Churn"] == 0) ],
                color="Red", fill = True)
Mth = sns.kdeplot(telecom_data_dummies.MonthlyCharges[(telecom_data_dummies["Churn"] == 1) ],
                ax =Mth, color="Blue", fill= True)
Mth.legend(["No Churn","Churn"],loc='upper right')
Mth.set_ylabel('Density')
Mth.set_xlabel('Monthly Charges')
Mth.set_title('Monthly charges by churn')
```

Out[25]:
```
Text(0.5, 1.0, 'Monthly charges by churn')
```



In [26]:
```
# Creating a KDE plot to compare the distribution of total charges for churned and non-churned customers
Tot = sns.kdeplot(telecom_data_dummies.TotalCharges[(telecom_data_dummies["Churn"] == 0) ],
                color="Red", fill = True)
Tot = sns.kdeplot(telecom_data_dummies.TotalCharges[(telecom_data_dummies["Churn"] == 1) ],
                ax =Tot, color="Blue", fill= 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[26]:
```
Text(0.5, 1.0, 'Total charges by churn')
```

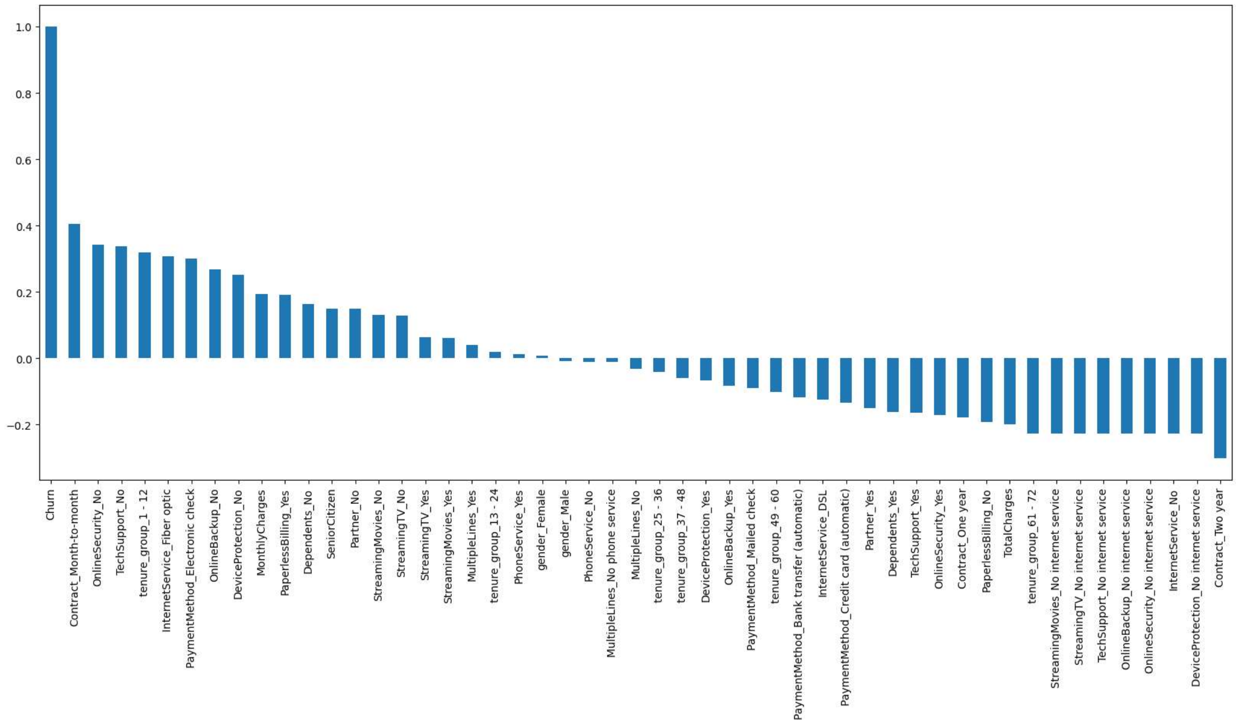


Some Insights:

1. Total Charges increase as Monthly Charges increase
2. Churn is high when Monthly Charges are high
3. Higher Churn at lower Total Charges

In [27]:
```python
# Creating a bar plot to display the correlation values between the 'Churn' variable and other variables
plt.figure(figsize=(20, 8))
telecom_data_dummies.corr()['Churn'].sort_values(ascending=False).plot(kind='bar')
```

Out[27]: <AxesSubplot:>



Some Insights:

The analysis suggests that the following factors are associated with high churn rates:

1. Month-to-month contracts
2. Lack of online security
3. Lack of tech support
4. Customers in their first year of subscription
5. Customers with fiber optics internet

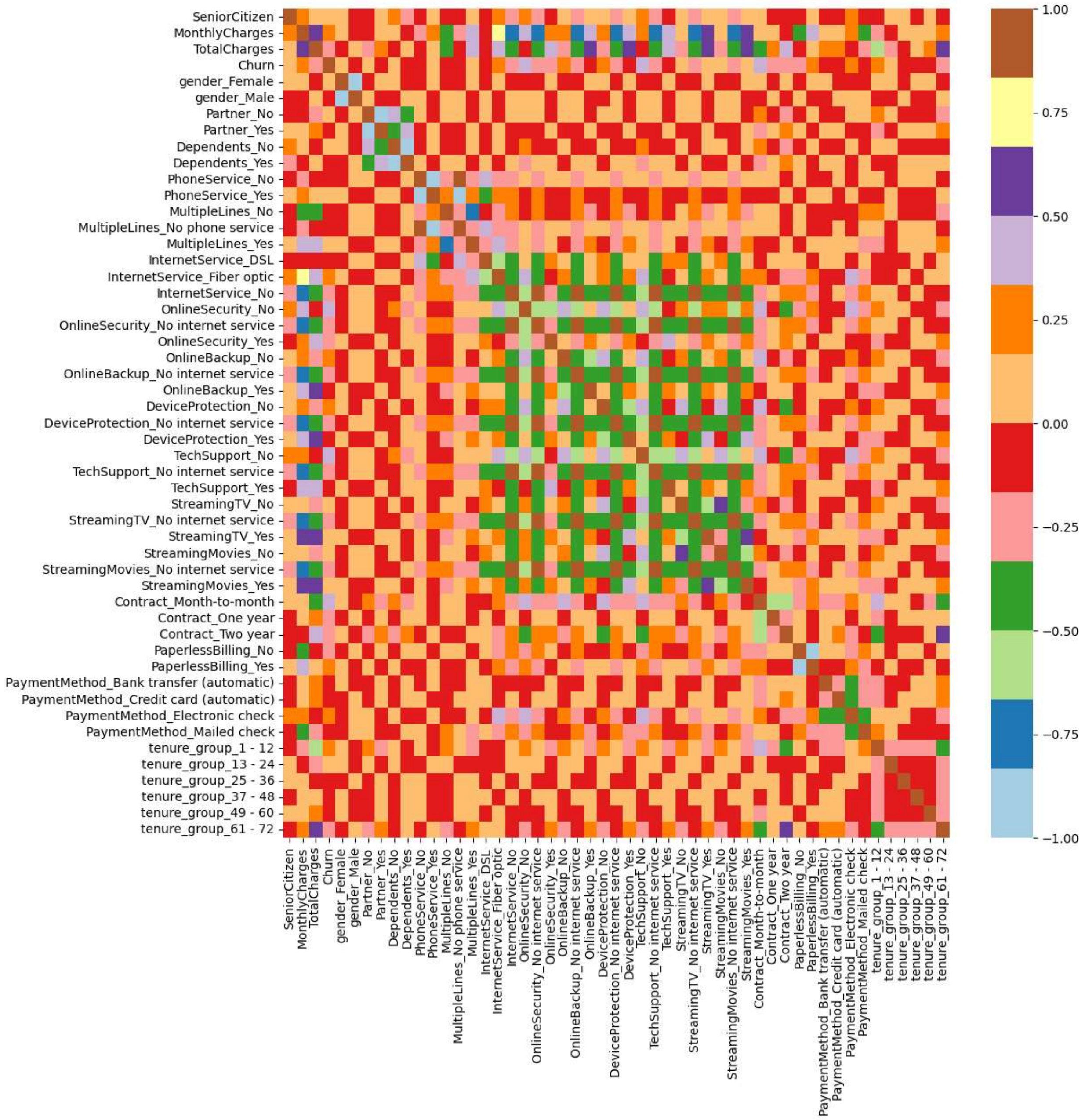
On the other hand, the following factors are associated with low churn rates:

1. Long-term contracts
2. Subscriptions without internet service
3. Customers who have been engaged for 5 or more years

Factors such as gender, availability of phone service, and the number of multiple lines have almost no impact on churn.

In [28]:
```python
# Creating a heatmap to visualize the correlation matrix of variables in the telecom_data_dummies datase
plt.figure(figsize=(12,12))
sns.heatmap(telecom_data_dummies.corr(), cmap="Paired")
```

Out[28]: <AxesSubplot:>

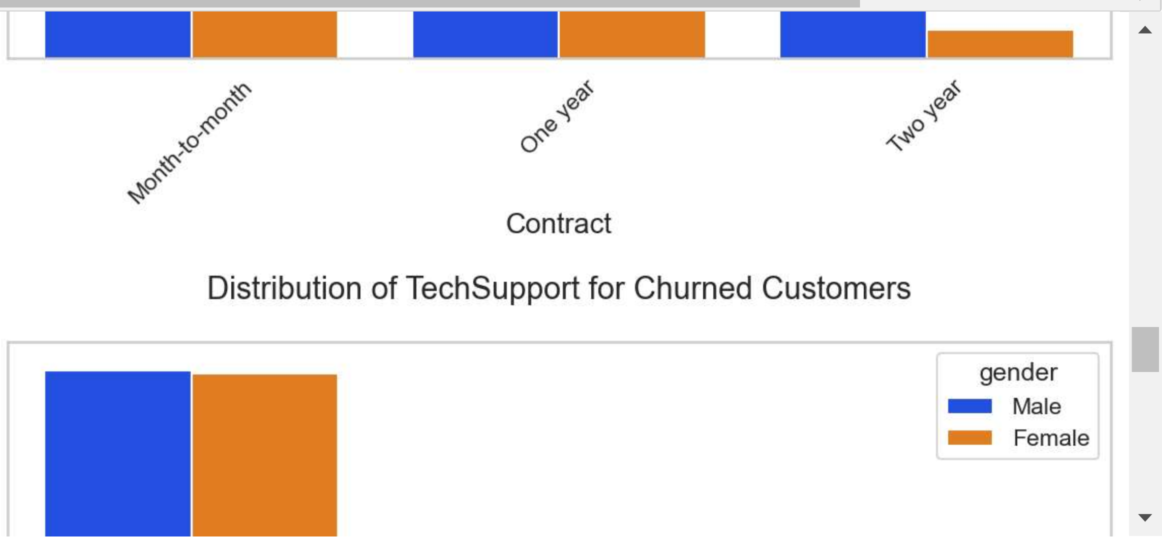


Bivariate Analysis

In [29]:
```python
# Creating a new DataFrame, new_df1_target0, by filtering telecom_data where Churn column is 0
new_df1_target0=telecom_data.loc[telecom_data["Churn"]==0]

# Creating a new DataFrame, new_df1_target1, by filtering telecom_data where Churn column is 1
new_df1_target1=telecom_data.loc[telecom_data["Churn"]==1]
```

In [30]:
```
#  Defining a function to create a customized countplot using seaborn
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,palette='bright')

    plt.show()
```

In [31]:
```
# Creating a countplot to visualize the distribution of the 'Partner' variable for churned customers, di
uniplot(new_df1_target1,col='Partner',title='Distribution of Partner for Churned Customers',hue='gender'

# Creating a countplot to visualize the distribution of the 'Partner' variable for non-churned customers
uniplot(new_df1_target0,col='Partner',title='Distribution of Gender for Non Churned Customers',hue='gend

# Creating a countplot to visualize the distribution of the 'PaymentMethod' variable for churned custome
uniplot(new_df1_target1,col='PaymentMethod',title='Distribution of PaymentMethod for Churned Customers',

# Creating a countplot to visualize the distribution of the 'Contract' variable for churned customers, d
uniplot(new_df1_target1,col='Contract',title='Distribution of Contract for Churned Customers',hue='gende

# Creating a countplot to visualize the distribution of the 'TechSupport' variable for churned customers
uniplot(new_df1_target1,col='TechSupport',title='Distribution of TechSupport for Churned Customers',hue=

# Creating a countplot to visualize the distribution of the 'SeniorCitizen' variable for churned custome
uniplot(new_df1_target1,col='SeniorCitizen',title='Distribution of SeniorCitizen for Churned Customers',
```



Some key insights from the above plots:

1. Customers who do not have a partner are more likely to churn.
2. Customers who use the electronic check payment method have the highest churn rate.
3. Customers with monthly contracts are more likely to churn since they have no contract terms and can freely switch providers.
4. Customers without online security and tech support are more likely to churn.
5. Non-senior citizens have a higher churn rate compared to senior citizens.

In [32]:
```
# Saving the telecom_data_dummies DataFrame as a CSV file named "telecom_churn.csv" without the index co
telecom_data_dummies.to_csv('telecom_churn.csv',index=False)
```

Model Development & Evaluation

In [34]:
```
# Importing the required libraries for development of the model
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.ensemble import RandomForestClassifier
from imblearn.combine import SMOTEENN
```

In [35]:
```
# Loading the telecom_churn.csv file into a DataFrame named df and Displaying the first few rows
df=pd.read_csv("telecom_churn.csv")
df.head()
```

Out[35]:

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

5 rows × 51 columns

In [36]:
```
# Splitting the dataset into feature variables and target variable
x=df.drop('Churn',axis=1)
y=df['Churn']
```

In [37]:
```
# Splitting the data into training and testing sets with 80% for training and 20% for testing
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

Random Forest Classifier

In [38]:
```
# Creating a Random Forest Classifier model
model_rf=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_
```

In [39]:
```
# Fitting the training data to the Random Forest Classifier model
model_rf.fit(x_train,y_train)
```

Out[39]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [40]:
```
# Predicting the target variable for the test data
y_pred=model_rf.predict(x_test)
```

In [41]:
```
# Calculating the accuracy score of the Random Forest Classifier model on the test data
model_rf.score(x_test,y_test)
```

Out[41]:
```
0.7938877043354655
```

In [42]:
```
# Printing the classification report
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1037
           1       0.64      0.48      0.55       370

    accuracy                           0.79      1407
   macro avg       0.74      0.69      0.71      1407
weighted avg       0.78      0.79      0.78      1407
```

In [43]:
```
# Creating an instance of the SMOTEENN algorithm
sm = SMOTEENN()

# Resampling the data using SMOTEENN
X_resampled1, y_resampled1 = sm.fit_resample(x,y)
```

In [44]:
```
# Splitting the resampled data into training and testing sets
xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test_size=0.20)
```

In [45]:
```
# Creating a Random Forest Classifier model with SMOTE-ENN resampled data
model_rf_smote=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6
```

In [46]:
```
# Fitting the training data to the Random Forest Classifier model with SMOTE-ENN resampled data
model_rf_smote.fit(xr_train1,yr_train1)
```

Out[46]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [47]:
```
# Predicting the target variable for the resampled test data
yr_predict1 = model_rf_smote.predict(xr_test1)
```

In [48]:
```
# Calculating the accuracy score of the Random Forest Classifier model with SMOTE-ENN on the resampled t
model_score_r1 = model_rf_smote.score(xr_test1, yr_test1)
```

In [49]:
```
# Printing the accuracy score of the Random Forest Classifier model with SMOTE-ENN
print(model_score_r1)

# Printing the classification report for the resampled test data
print(metrics.classification_report(yr_test1, yr_predict1))
```

```
0.942390369733448
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       533
           1       0.93      0.97      0.95       630

    accuracy                           0.94      1163
   macro avg       0.94      0.94      0.94      1163
weighted avg       0.94      0.94      0.94      1163
```

The rationale behind the chosen model, which is a Random Forest Classifier, is that it is a powerful and versatile machine learning algorithm that can handle both numerical and categorical data. It is an ensemble model that combines multiple decision trees to make predictions.The chosen model is a Random Forest Classifier, which is effective for handling the given dataset. The code also incorporates the SMOTEENN algorithm to address class imbalance, resulting in improved performance in terms of accuracy and other evaluation metrics.