In [ ]:
```
# Customer churn is the phenomenon in which a client stops doing business with an entity.
# Users can stop using a company’s product or service for a variety of reasons, such as:
# affordability, dissatisfaction with the offering, and bad customer service.
# We will use the Telco Customer Churn dataset from Kaggle for this analysis.
```

In [ ]:
```
# STEP 1: Pre-Requisites for Building a Churn Prediction Model-----------!
```

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

In [2]:
```
# Step 2: Reviewing the Dataset:
# First, let’s load the dataframe into Python with the pandas library and take a look at its head.
```

In [3]:
```
df=pd.read_csv('Telco-Customer-Churn.csv')
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Dev
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]:

```
# Checking Number of Rows and Columns in Our Dataset:
df.shape
```

Out[4]:

```
(7043, 21)
```

In [5]:

```
# Checking Basic Information Of the Dataset
df.info()
```

```
<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 [6]:

```
# Here we can see each user is identified through a unique customer ID. There are 19 independent variables used to predict the
# In this dataset, customer churn is defined as users who have left within the last month.
```

In [7]:

```
# Checking Missing Values in the Dataset.
df.isnull().sum()
```

Out[7]:
```
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        0
Churn               0
dtype: int64
```

In [8]:
```
# Checking duplicates in the Dataset
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
# Checking all Columns names
df.columns
```

Out[9]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [10]:
```
# Stastical Analysis of numeric features
df.describe()
```

Out[10]:

	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 [11]:

```
# Let’s count the number of customers in the dataset who have churned:
df["Churn"].value_counts()
```

Out[11]:

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

In [21]:

```
# We can see Only around 27% of the customers in the dataset have churned. This means that we are dealing with an imbalanced c
# We will need to perform some feature engineering to create a balanced training dataset before building the predictive model.
```

In [25]:

```
# Step 3: Exploratory Data Analysis for Customer Churn Prediction---------------!
```

In [27]:

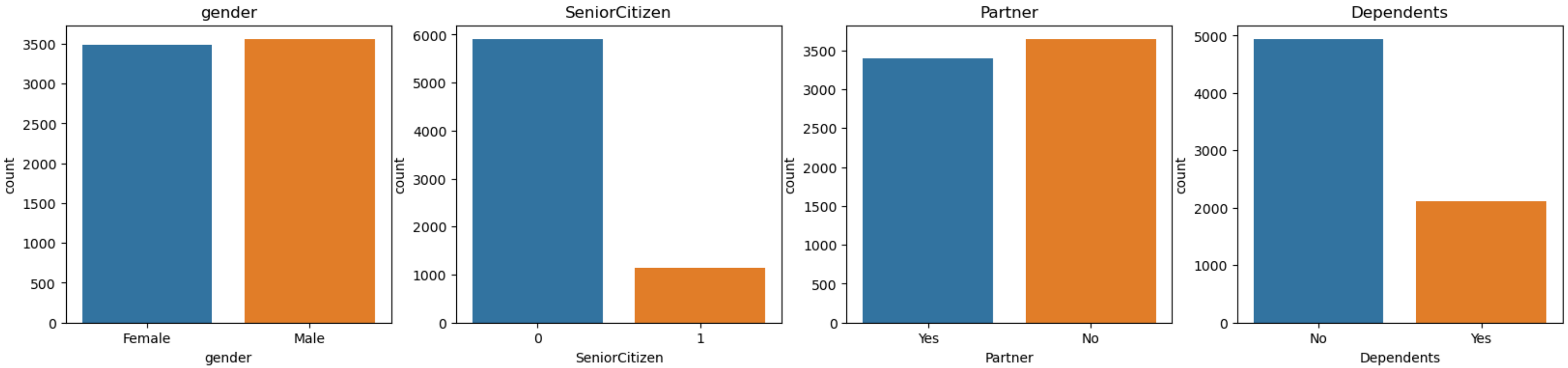
```
# Now, let’s perform some exploratory data analysis to gain a better understanding of the independent variables in the dataset
#and their relationship with customer churn.
```

In [29]:

```
# We will start by analyzing the demographic data points:
cols = ['gender','SeniorCitizen',"Partner","Dependents"]
num = cols

plt.figure(figsize=(20,4))
for i, col in enumerate(num):
```

```python
    ax = plt.subplot(1, len(num), i+1)
    sns.countplot(x=str(col), data=df)
    ax.set_title(f"{col}")
```

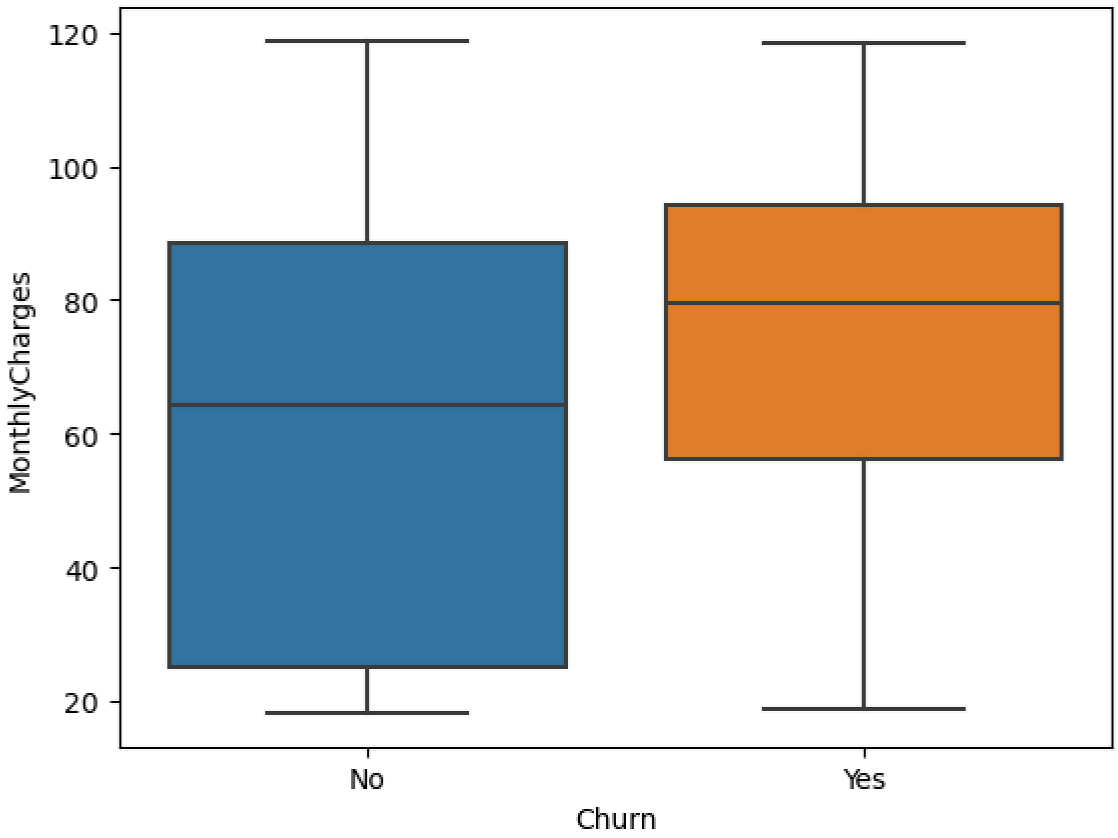


In [30]:
```python
# From here we see most customers in the dataset are younger individuals without a dependent.
# There is an equal distribution of user gender and marital status.
```

In [31]:
```python
# Now, let’s look into the relationship between cost and customer churn.
# In the real world, users tend to unsubscribe to their mobile service provider and switch to a different brand,
# if they find the monthly subscription cost too high. Let’s check if that behavior is reflected in our dataset:
```

In [35]:
```python
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
```

Out[35]:
```
<Axes: xlabel='Churn', ylabel='MonthlyCharges'>
```



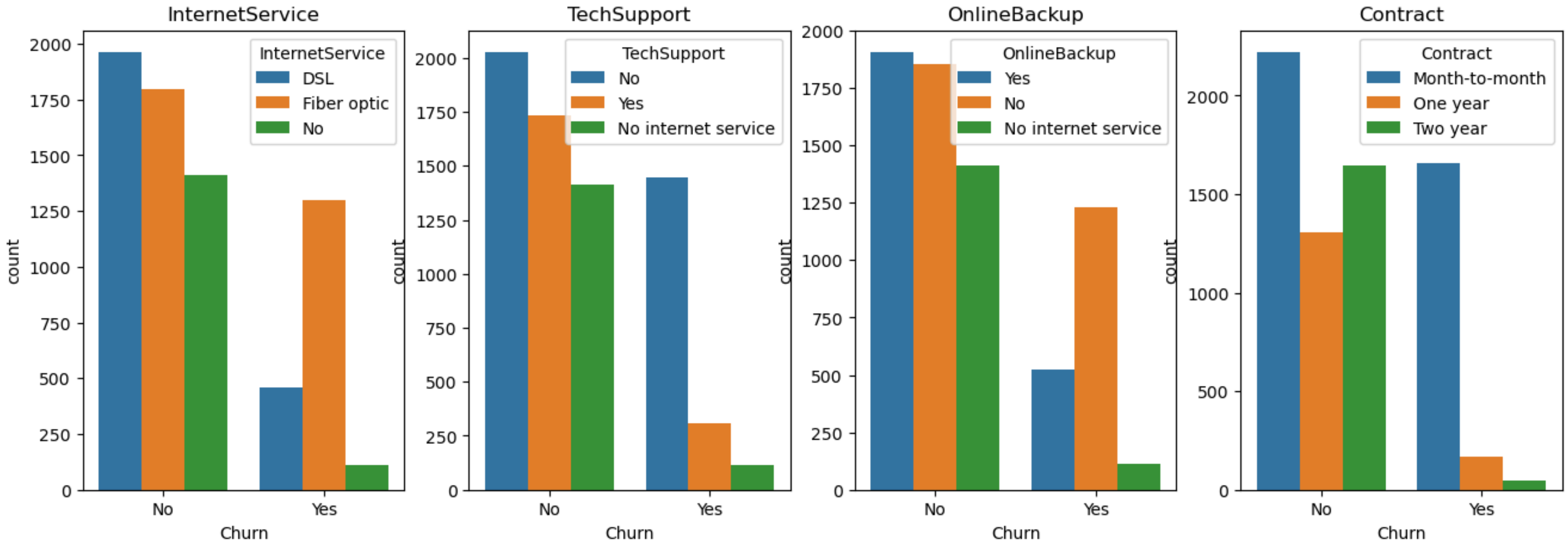
In [37]:
```
# From here we can see customers who churned have a higher median monthly charge than customers who renewed their subscription
```

In [39]:
```
# Finally, let’s analyze the relationship between customer churn and a few other categorical variables captured in the dataset
```

In [41]:
```
cols = ['InternetService',"TechSupport","OnlineBackup","Contract"]

plt.figure(figsize=(16,5))

for i, col in enumerate(cols):
    ax = plt.subplot(1, len(cols), i+1)
    sns.countplot(x ="Churn", hue = str(col), data = df)
    ax.set_title(f"{col}")
```



In [42]:
```
# Analysis of Above Visualization:
# 1.InternetService: It is clear from the visual above that customers who use fiber optic Internet churn more often than other
# This might be because fiber Internet is a more expensive service, or this provider doesn’t have good coverage.

# 2.TechSupport: Many users who churned did not sign up for tech support.
# This might mean that these customers did not receive any guidance on fixing technical issues and decided to stop using the s

# 3.OnlineBackup: Many customers who had churned did not sign up for an online backup service for data storage.

# 4.Contract: Users who churned were almost always on a monthly contract.
# This makes sense, since these customers pay for the service on a monthly basis and can easily cancel their subscription befo
```

In [43]:
```
# For instance,if the company realizes that most of their users who churn have not signed up for tech support,
# they can include this as a complimentary service in some of their future product offerings to prevent other customers from l
```

In [47]:
```
# Step 4: Preprocessing Data for Customer Churn--------------!
```

In [49]:
```
# Converting total_charge column into numeric:
df['TotalCharges']=pd.to_numeric(df['TotalCharges'],errors='coerce')
```

In [51]:
```
# Step 5: Encoding of Categorical Columns------------>
```

In [53]:
```
# The categorical variables in the dataset need to be converted into a numeric format before we can feed them into the machine
# We will perform the encoding using Scikit-Learn’s label encoder.
```

In [55]:
```
# First, let’s take a look at the categorical features in the dataset:

cat_features = df.drop(['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure'],axis=1)
cat_features.head()
```

Out[55]:

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSuppo
0	Female	Yes	No	No	No phone service	DSL	No	Yes	No	I
1	Male	No	No	Yes	No	DSL	Yes	No	Yes	I
2	Male	No	No	Yes	No	DSL	Yes	Yes	No	I
3	Male	No	No	No	No phone service	DSL	Yes	No	Yes	Y
4	Female	No	No	Yes	No	Fiber optic	No	No	No	I

In [57]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
df_cat = cat_features.apply(le.fit_transform)
df_cat.head()
```

Out[57]:

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSuppo
0	0	1	0	0	1	0	0	2	0	
1	1	0	0	1	0	0	2	0	2	
2	1	0	0	1	0	0	2	2	0	
3	1	0	0	0	1	0	2	0	2	
4	0	0	0	1	0	1	0	0	0	

In [58]:

```
# Notice that all the categorical values in the dataset have now been replaced with numbers.
#Finally, run the following lines of code to merge the dataframe we just created with the previous one:
```

In [61]:

```
num_features = df[['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure']]
finaldf = pd.merge(num_features, df_cat, left_index=True, right_index=True)
```

In [63]:

```
finaldf.head()
```

Out[63]:

	customerID	TotalCharges	MonthlyCharges	SeniorCitizen	tenure	gender	Partner	Dependents	PhoneService	MultipleLines	...	Onli
0	7590-VHVEG	29.85	29.85	0	1	0	1	0	0	1	...	
1	5575-GNVDE	1889.50	56.95	0	34	1	0	0	1	0	...	
2	3668-QPYBK	108.15	53.85	0	2	1	0	0	1	0	...	
3	7795-CFOCW	1840.75	42.30	0	45	1	0	0	0	1	...	
4	9237-HQITU	151.65	70.70	0	2	0	0	0	1	0	...	

5 rows × 21 columns

In [79]:

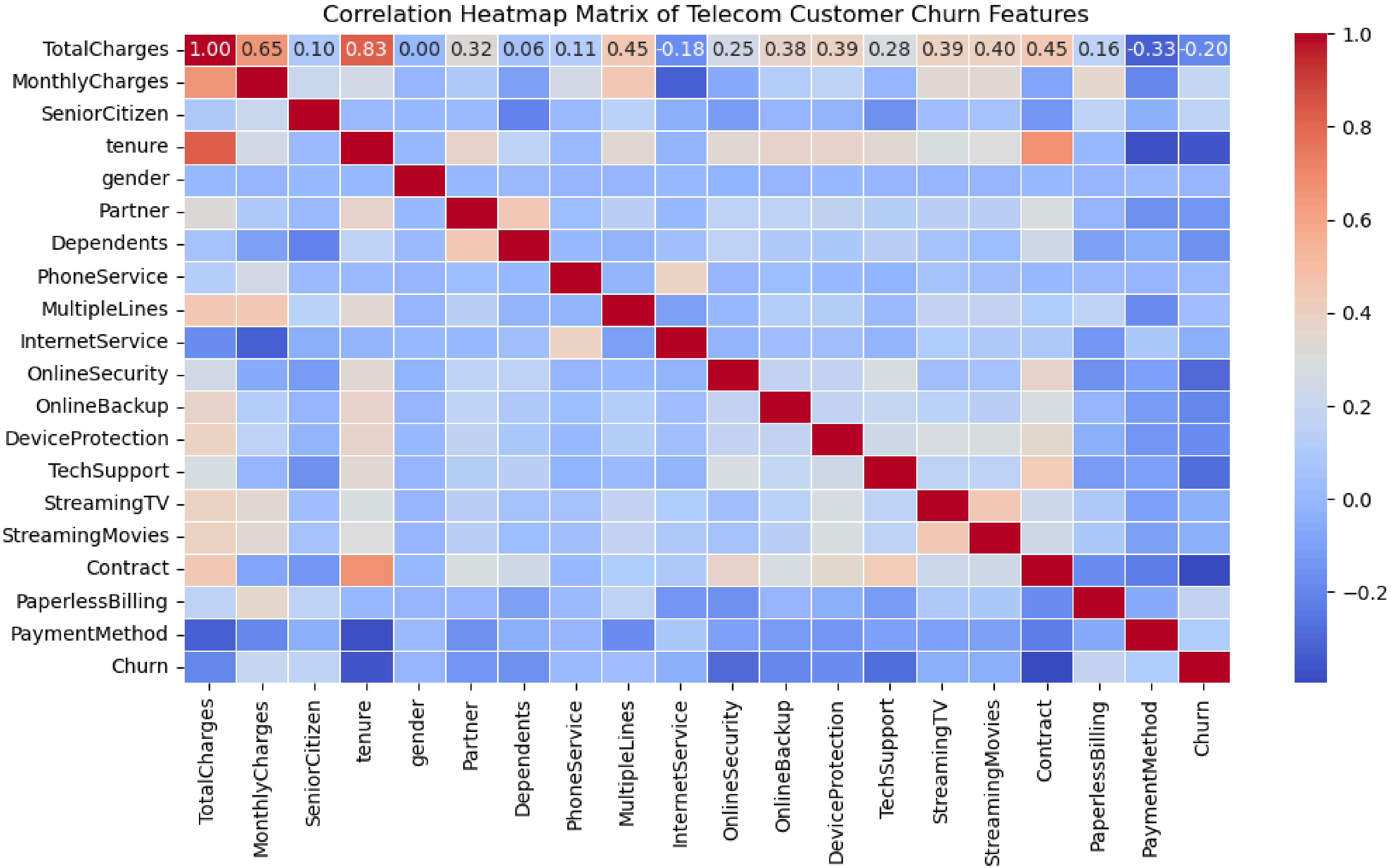
```
# Checking Correlation between Features:
corr_matrix=finaldf.corr()
corr_matrix
```

Out[79]:

	TotalCharges	MonthlyCharges	SeniorCitizen	tenure	gender	Partner	Dependents	PhoneService	MultipleLines
TotalCharges	1.000000	0.651065	0.102411	0.825880	0.000048	0.319072	0.064653	0.113008	0.453202
MonthlyCharges	0.651065	1.000000	0.219874	0.246862	-0.013779	0.097825	-0.112343	0.248033	0.433905
SeniorCitizen	0.102411	0.219874	1.000000	0.015683	-0.001819	0.016957	-0.210550	0.008392	0.146287
tenure	0.825880	0.246862	0.015683	1.000000	0.005285	0.381912	0.163386	0.007877	0.343673
gender	0.000048	-0.013779	-0.001819	0.005285	1.000000	-0.001379	0.010349	-0.007515	-0.006908
Partner	0.319072	0.097825	0.016957	0.381912	-0.001379	1.000000	0.452269	0.018397	0.142717
Dependents	0.064653	-0.112343	-0.210550	0.163386	0.010349	0.452269	1.000000	-0.001078	-0.024975
PhoneService	0.113008	0.248033	0.008392	0.007877	-0.007515	0.018397	-0.001078	1.000000	-0.020504
MultipleLines	0.453202	0.433905	0.146287	0.343673	-0.006908	0.142717	-0.024975	-0.020504	1.000000
InternetService	-0.175691	-0.322173	-0.032160	-0.029835	-0.002236	0.000513	0.044030	0.387266	-0.108849
OnlineSecurity	0.254473	-0.053576	-0.127937	0.327283	-0.014899	0.150610	0.151198	-0.014163	0.007306
OnlineBackup	0.375556	0.119943	-0.013355	0.372434	-0.011920	0.153045	0.090231	0.024040	0.117276
DeviceProtection	0.389066	0.163984	-0.021124	0.372669	0.001348	0.165614	0.079723	0.004718	0.122614
TechSupport	0.276890	-0.008237	-0.151007	0.324729	-0.006695	0.126488	0.132530	-0.018136	0.010941
StreamingTV	0.392472	0.337156	0.031019	0.290572	-0.005624	0.136679	0.046214	0.056393	0.175403
StreamingMovies	0.398088	0.335761	0.047088	0.296785	-0.008920	0.129907	0.022088	0.043025	0.181705
Contract	0.450306	-0.072739	-0.141820	0.676734	0.000095	0.294094	0.240556	0.003019	0.111029
PaperlessBilling	0.157830	0.351930	0.156258	0.004823	-0.011902	-0.013957	-0.110131	0.016696	0.165306
PaymentMethod	-0.330594	-0.192500	-0.038158	-0.370087	0.016942	-0.156232	-0.041989	-0.005499	-0.176598
Churn	-0.199484	0.192858	0.150541	-0.354049	-0.008545	-0.149982	-0.163128	0.011691	0.038043

In [81]:

```
# Plot Heatmap:
plt.figure(figsize=(12,6))
sns.heatmap(corr_matrix,annot=True,fmt=".2f",cmap="coolwarm",linewidths=0.5)
plt.title('Correlation Heatmap Matrix of Telecom Customer Churn Features',size=12)
plt.show()
```



In [83]:
```python
# Sort the Correleation of Churn with Other Features:
churn_corr=corr_matrix['Churn'].sort_values(ascending=False)
churn_corr
```

Out[83]:
```
Churn               1.000000
MonthlyCharges      0.192858
PaperlessBilling    0.191454
SeniorCitizen       0.150541
PaymentMethod       0.107852
MultipleLines       0.038043
PhoneService        0.011691
gender             -0.008545
StreamingTV        -0.036303
StreamingMovies    -0.038802
InternetService    -0.047097
Partner            -0.149982
Dependents         -0.163128
DeviceProtection   -0.177883
OnlineBackup       -0.195290
TotalCharges       -0.199484
TechSupport        -0.282232
OnlineSecurity     -0.289050
tenure             -0.354049
Contract           -0.396150
Name: Churn, dtype: float64
```

In [ ]:
```python
# From the correlation analysis, here are some key observations:
# Strong Positive Correlations:
# Internet Service (Fiber optic): Indicates customers with fiber optic internet are more likely to churn compared to other int
# Payment Method : Customers who pay via electronic check have a higher likelihood of churning.
# Monthly Charges (0.193356): Higher monthly charges are moderately associated with a higher churn rate.
```

In [ ]:
```python
# Strong Negative Correlations:
# Tenure Months (-0.354049): Customers with longer tenure are less likely to churn, indicating loyalty.
# Contract (Two year) (-0.392253): Longer contracts (e.g., two-year contracts) are associated with lower churn, likely due to
# Dependents (-0.163142): Customers with dependents are less likely to churn.
# Internet Service (No Internet Service) (-0.047890): Not having internet service is associated with lower churn,
# possibly because these customers are less reliant on services where churn is relevant.
```

In [ ]:
```
# Step 6: Training the Dataset--------->
```

In [ ]:
```
# Oversampling:
# As we see above,the dataset is imbalanced, which means that a majority of values in the target variable belong to a single c
# Most customers in the dataset did not churn - only 27% of them did.
# This class imbalance problem can lead to an underperforming machine learning model.Some algorithms that train on an imbalanc
# predicting the majority class. In our case, for instance, the model may predict that none of the customers churned.
# While a model like this will be highly accurate (in this case it will be correct 73% of the time),
# it is of no value to us since it is always predicting a single outcome.
```

In [ ]:
```
# We are going to oversample the minority class until the number of data points are equal to that of the majority class.
# Before we oversample, let’s do a train-test split. We will oversample solely on the training dataset,
# as the test dataset must be representative of the true population:
```

In [67]:
```
from sklearn.model_selection import train_test_split

finaldf = finaldf.dropna()
finaldf = finaldf.drop(['customerID'],axis=1)

x= finaldf.drop(['Churn'],axis=1)
y = finaldf['Churn']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

In [69]:
```
# Now, let’s oversample the training dataset:

from imblearn.over_sampling import SMOTE

oversample = SMOTE(k_neighbors=5)
x_smote, y_smote = oversample.fit_resample(x_train, y_train)
x_train, y_train = x_smote, y_smote
```

In [70]:
```
# Let’s check the number of samples in each class to ensure that they are equal:

y_train.value_counts() #There should be 3614  values in each class, which means that the training dataset is now balanced.
```

Out[70]:
```
Churn
1    3614
0    3614
Name: count, dtype: int64
```

In [71]:
```python
# Step 7: Building the Customer Churn Prediction Model------------!
# 1. We will now build a random forest classifier to predict customer churn:

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=46)
rf.fit(x_train,y_train)
```

Out[71]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=46)
```

In [72]:
```python
# Step 8: Customer Churn Prediction Model Evaluation-----------!
```

In [97]:
```python
# Let's evaluate the model predictions on the test dataset:

from sklearn.metrics import accuracy_score
preds = rf.predict(x_test)
print(confusion_matrix(y_test, preds))
print(classification_report(y_test, preds))
```

```
[[1308  241]
 [ 238  323]]
              precision    recall  f1-score   support

           0       0.85      0.84      0.85      1549
           1       0.57      0.58      0.57       561

    accuracy                           0.77      2110
   macro avg       0.71      0.71      0.71      2110
weighted avg       0.77      0.77      0.77      2110
```

In [ ]:
```python
# Our Random Classifier model is performing well, with an accuracy of approximately 0.78 on the test dataset.
```

In [87]:
```
# 2.Building LogisticRegression Model:
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [89]:
```
model1 = LogisticRegression(max_iter=1000)
model1.fit(x_train, y_train)

y_pred = model1.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[1174  375]
 [ 150  411]]
              precision    recall  f1-score   support

           0       0.89      0.76      0.82      1549
           1       0.52      0.73      0.61       561

    accuracy                           0.75      2110
   macro avg       0.70      0.75      0.71      2110
weighted avg       0.79      0.75      0.76      2110
```

In [93]:
```
model1.score(x_test,y_test)
```

Out[93]:
```
0.7511848341232228
```

In [23]:
```
# Hence Our Logistic Regression model is also performing well, with an accuracy of approximately 0.76 on the test dataset.
```