Goal of the project:

Predict whether a customer will cancel their subscription next month and provide insights on why they churn.

In [1]:
```
import numpy as np
import pandas as pd
customer_data_original=pd.read_csv("Documents/My_projetcs/project_1/WA_Fn-UseC_-Telco-Customer-Churn.csv")
customer_data_original.head()
```

Out[1]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	D
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 [2]:
```
customer_data_duplicate=customer_data_original.copy()
customer_data_duplicate.shape
```

Out[2]: (7043, 21)

In [3]:
```
customer_data_duplicate.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	D
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]:
```
customer_data_duplicate.isnull().sum()
```

Out[4]:
```
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 [5]:
```
customer_data_duplicate.duplicated().sum()
```

Out[5]:
```
0
```

In [6]:
```
customer_data_duplicate.describe()
```

Out[6]:

	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 [7]:
```
customer_data_duplicate.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 [8]:
```
customer_data_duplicate.dtypes
```

Out[8]:
```
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 [9]:
```
customer_data_duplicate.nunique()
```

Out[9]:
```
customerID          7043
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

In [10]:
```
customer_data_duplicate.columns
```

Out[10]:
```
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 [11]:
```
customer_data_duplicate.head()
```

Out[11]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	D
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

EDA

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
customer_data_duplicate
```

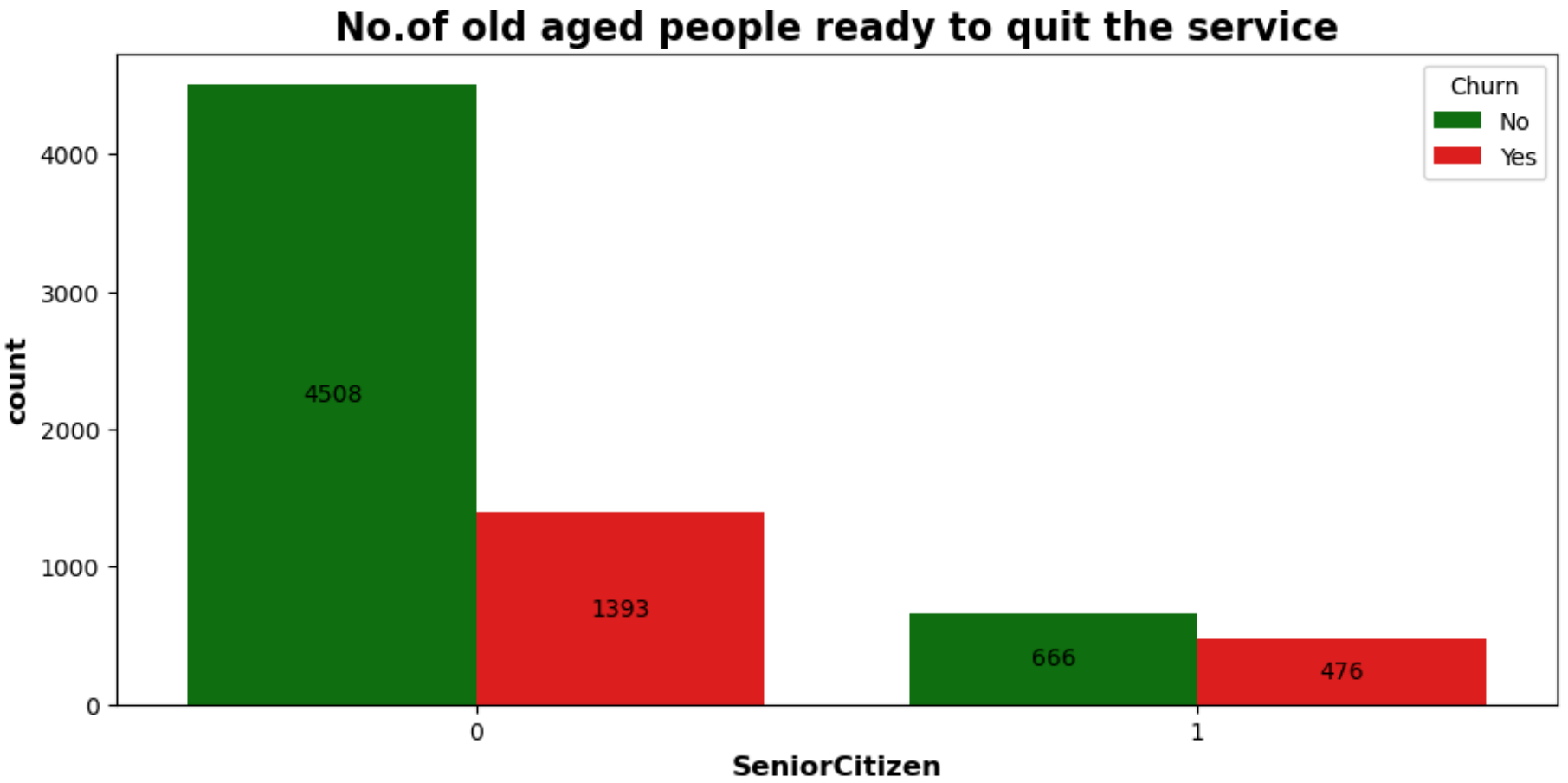
Out[12]:

	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	.
...	...	...	...	...	...	...	...	...	...	...	.
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	.
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	.
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	.
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	.
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	.

7043 rows × 21 columns

Adding Datalabels to bargraph so that the chart looks more understandable, **`annotate()`** is used to add labels to the bars,from the graph we can tell that the male are more who dont wants to quit than women and 939 females are ready to quit than male.

In [67]:
```
def datalabels(ax, fontsize=10, color='black', inside=False): # ax means axis
    for p in ax.patches:
        height = p.get_height()
        if height > 0:
            if inside:
                ax.annotate(
                    f'{height:.0f}',
                    (p.get_x() + p.get_width() / 2, height / 2),
                    ha='center', va='center',
                    fontsize=fontsize, color=color
                )
            else:
                ax.annotate(
                    f'{height:.0f}',
                    (p.get_x() + p.get_width() / 2, height),
                    ha='center', va='bottom',
                    fontsize=fontsize, color=color
                )


plt.figure(figsize=(11,5))
datalabels(
    sns.countplot(data=customer_data_duplicate,
                          x='SeniorCitizen',
                          hue='Churn',
                          palette=['Green','Red']),inside=True)
plt.title('No.of old aged people ready to quit the service',fontsize=16,fontweight='bold')
plt.xlabel('SeniorCitizen',fontsize=12,fontweight='bold')
plt.ylabel('count',fontsize=12,fontweight='bold')
plt.show()
```

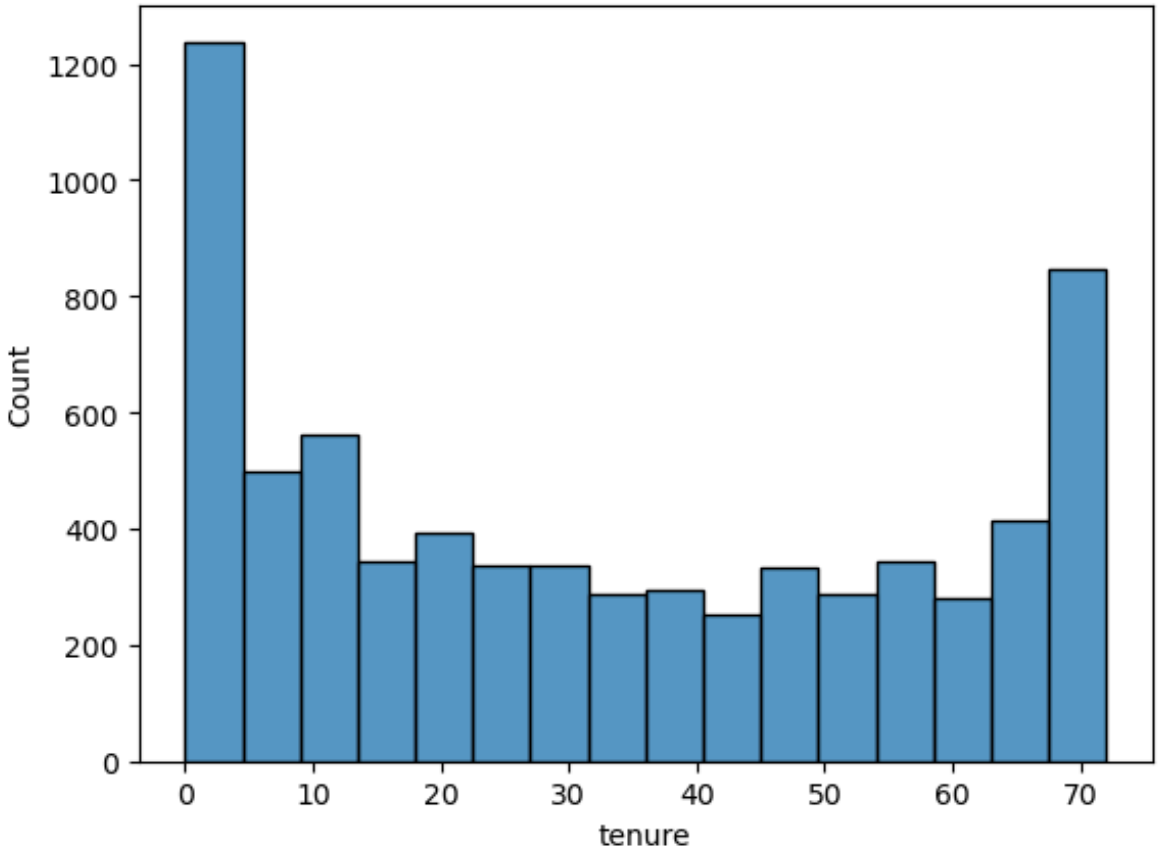


0 indicates the adults(means not senior citizen) and 1 indicates the senior citizen,from overall senior citizens nearly 41% people are ready to quit the service and from adults nearly 24% people are ready to quit the service,so from overall 26% people are ready to quit the service

In [15]:
```
customer_data_duplicate['tenure'].describe()
```

Out[15]:
```
count    7043.000000
mean       32.371149
std        24.559481
min         0.000000
25%         9.000000
50%        29.000000
75%        55.000000
max        72.000000
Name: tenure, dtype: float64
```

In [16]:
```
sns.histplot(data=customer_data_duplicate,x='tenure')
plt.show()
```



In [17]:
```
bins = [0,10,20,30,40,50,60,72, np.inf]
labels = ["0-9", "10-19", "20-29", "30-39", "40-49", "50-59", "60-69",'70-72']

tenure_groups = pd.cut(customer_data_duplicate["tenure"], bins=bins, right=False, labels=labels)
counts = tenure_groups.value_counts().sort_index()

print(counts)
```

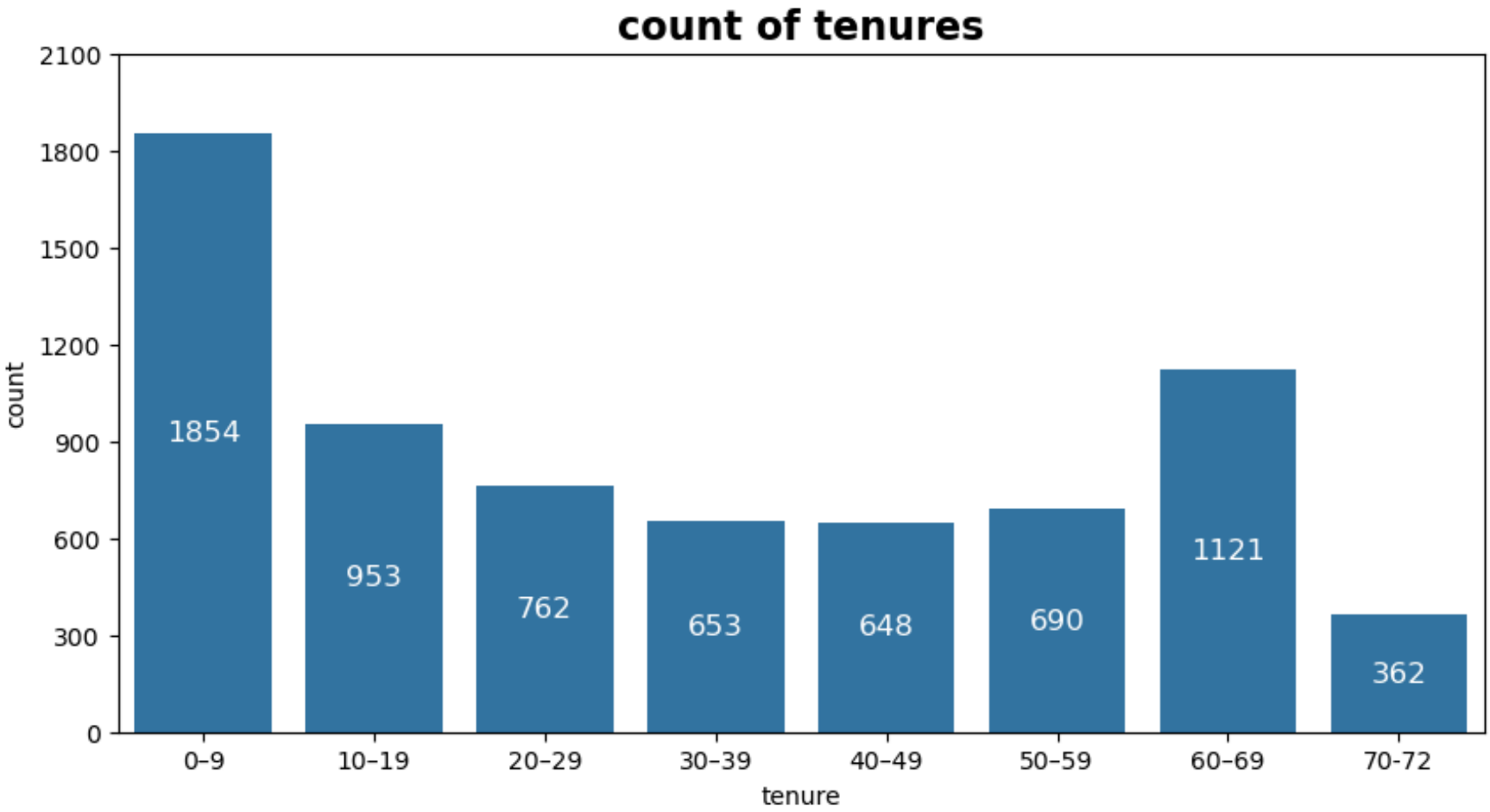
```
tenure
0-9      1854
10-19     953
20-29     762
30-39     653
40-49     648
50-59     690
60-69    1121
70-72     362
Name: count, dtype: int64
```

In [18]:
```
tenure_range_df=pd.DataFrame(counts).reset_index()
tenure_range_df.columns=["tenure","count"]
tenure_range_df
```

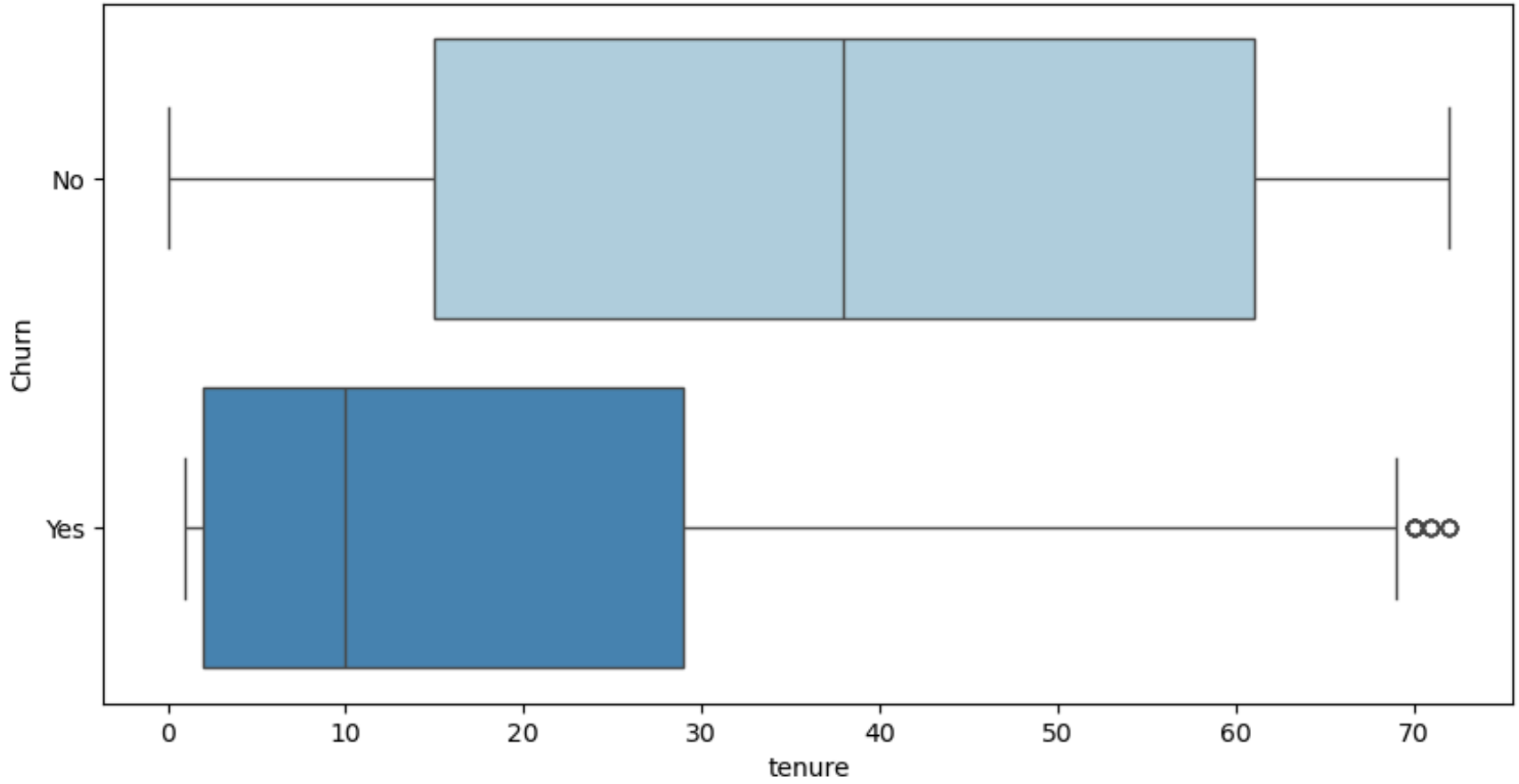
Out[18]:

	tenure	count
0	0–9	1854
1	10–19	953
2	20–29	762
3	30–39	653
4	40–49	648
5	50–59	690
6	60–69	1121
7	70-72	362

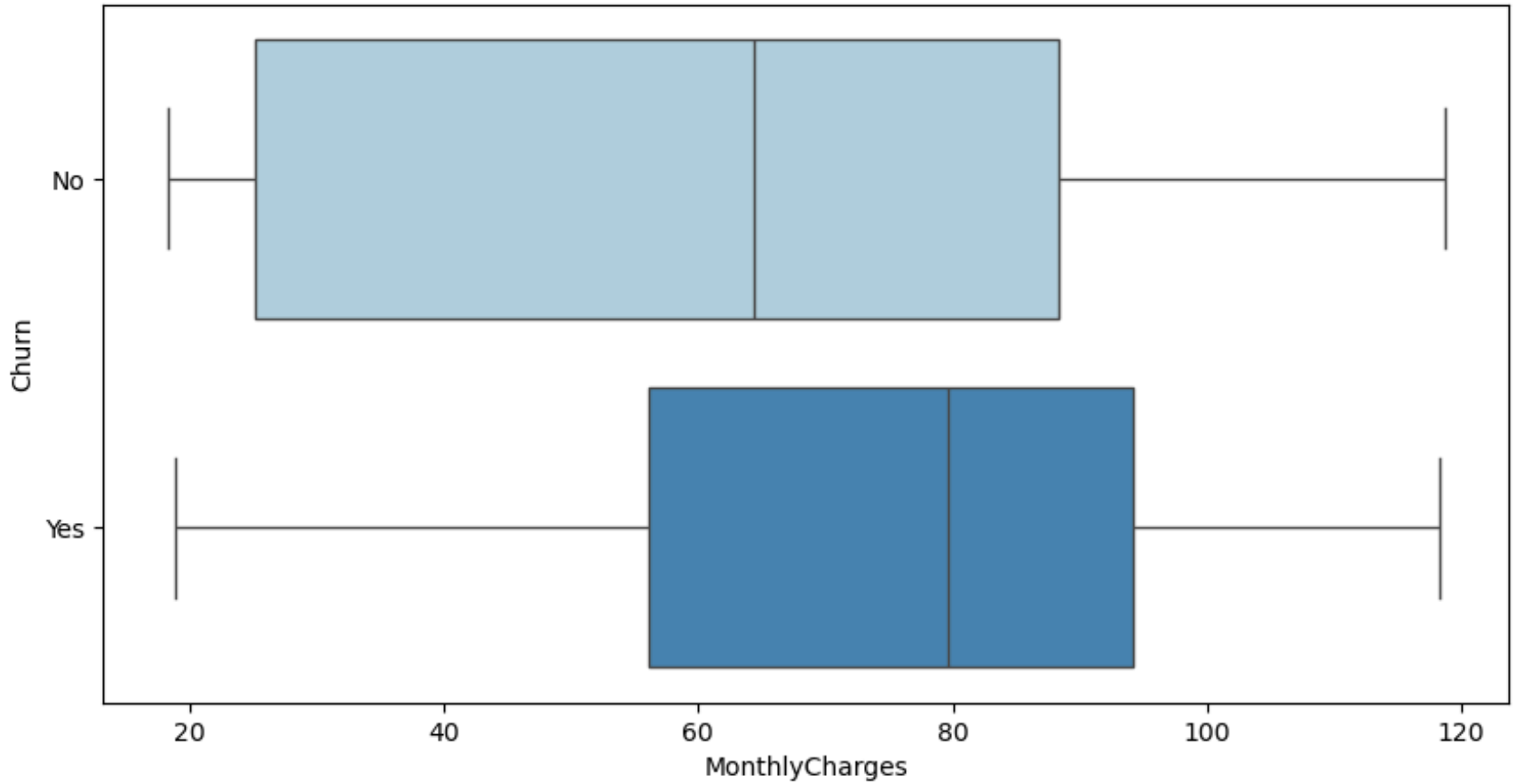
In [19]:
```
plt.figure(figsize=(10,5))
ax=sns.barplot(data=tenure_range_df,x='tenure',y='count')
datalabels(ax,fontsize=12,color='white',inside=True)
plt.title('count of tenures',fontsize=16,fontweight='bold')
plt.yticks(np.arange(0,2400,300))
plt.show()
```



In [20]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data=customer_data_duplicate,x='tenure',y='Churn',hue='Churn',palette='Blues')
plt.show()
```

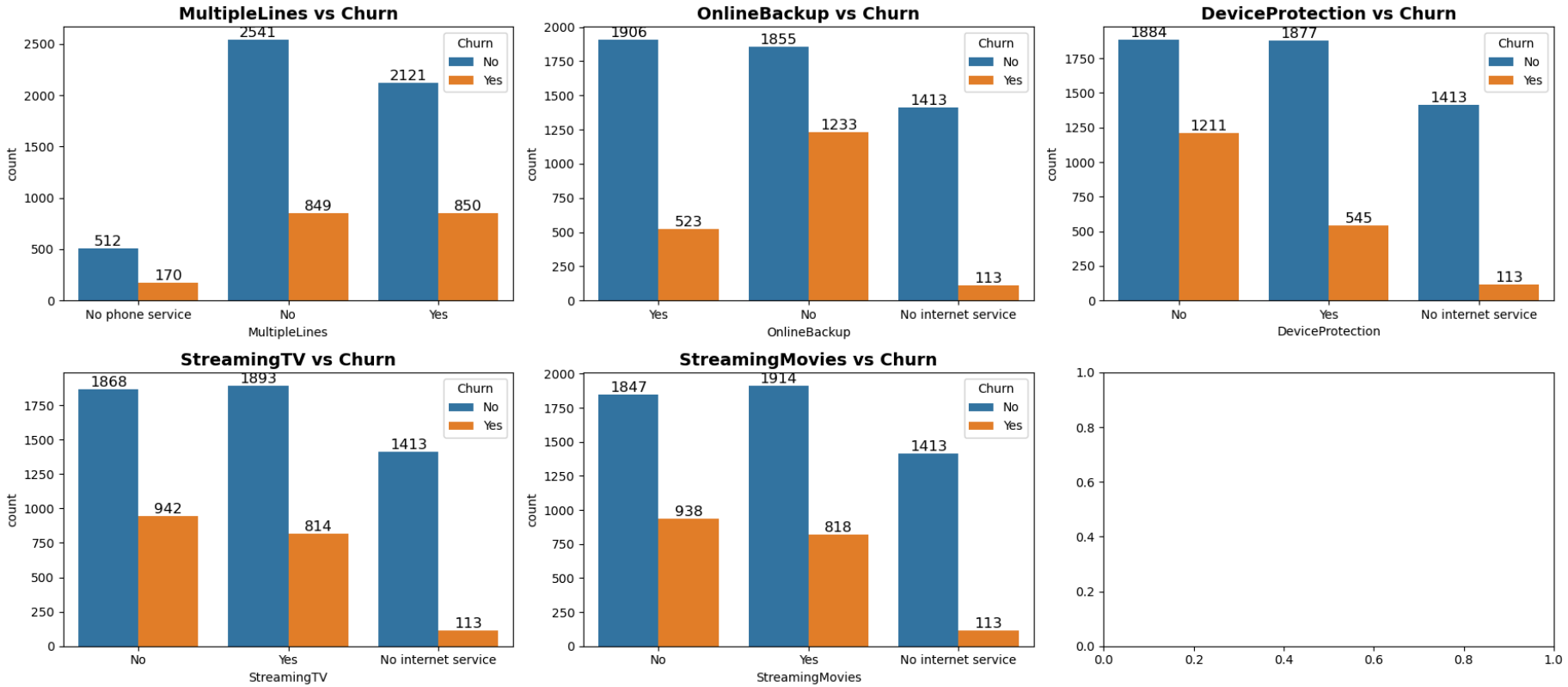


In [21]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data=customer_data_duplicate,x='MonthlyCharges',y='Churn',hue='Churn',palette='Blues')
plt.show()
```



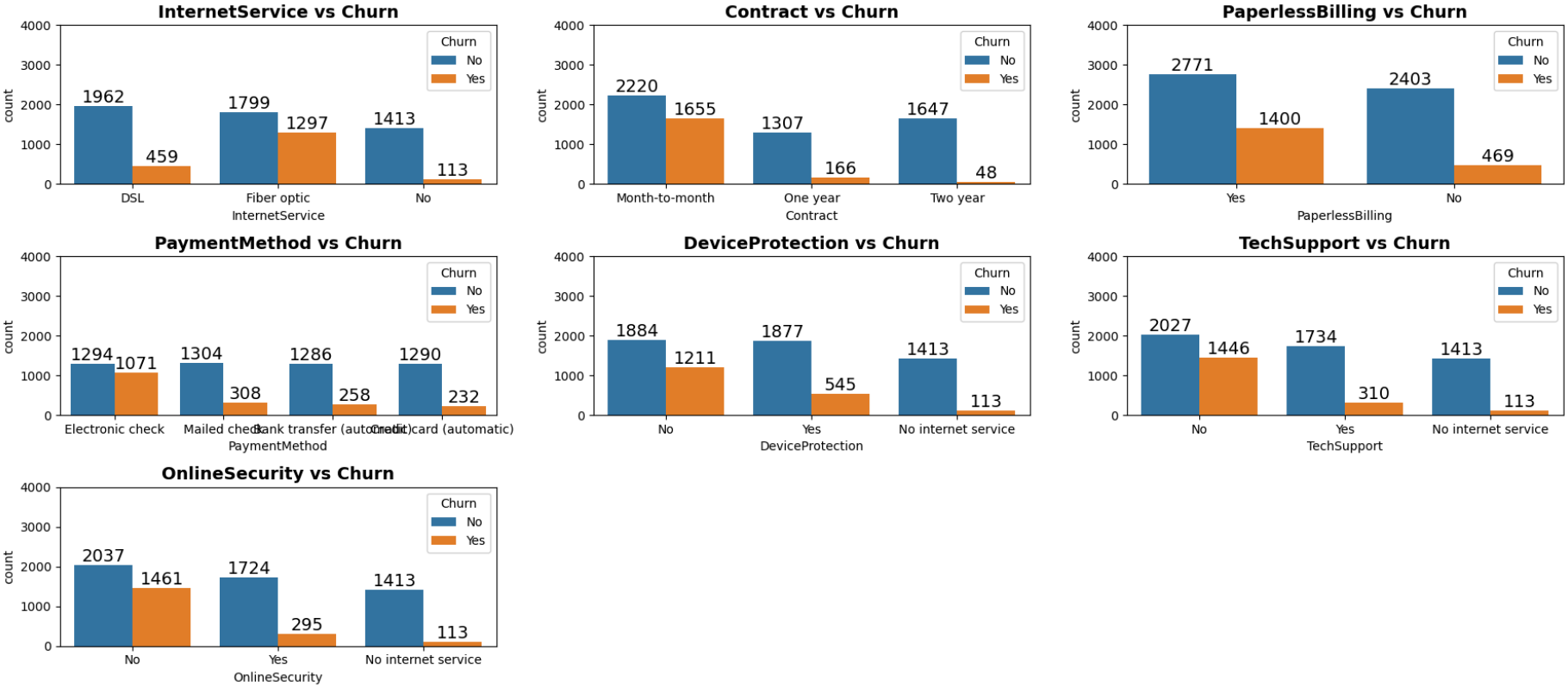
In [22]:
```
cols = [ 'MultipleLines', 'OnlineBackup',
        'DeviceProtection', 'StreamingTV', 'StreamingMovies']

fig, axes = plt.subplots(2, 3, figsize=(18, 8))
axes = axes.flatten()

for i, col in enumerate(cols):
    ax = sns.countplot(data=customer_data_duplicate, x=col, hue='Churn', ax=axes[i])
    datalabels(ax, fontsize=12, color='black', inside=False)
    ax.set_title(f'{col} vs Churn', fontsize=14, fontweight='bold')
   # ax.set_yticks(np.arange(0, 6000, 1000))
```

```
plt.tight_layout()
plt.show()
```



In [23]:
```
cols = ['InternetService', 'Contract', 'PaperlessBilling',
        'PaymentMethod', 'DeviceProtection', 'TechSupport','OnlineSecurity',]

fig, axes = plt.subplots(3, 3, figsize=(18, 8))
axes = axes.flatten()

for i, col in enumerate(cols):
    ax = sns.countplot(data=customer_data_duplicate, x=col, hue='Churn', ax=axes[i])
    datalabels(ax, fontsize=14, color='black', inside=False)
    ax.set_title(f'{col} vs Churn', fontsize=14, fontweight='bold')
    ax.set_yticks(np.arange(0, 5000, 1000))

for ax in axes[7:]:
```

```
    ax.set_visible(False)


plt.tight_layout()
plt.show()
```



Statistical Check

In [24]:
```
customer_data_duplicate=customer_data_duplicate.drop('customerID',axis=1)
customer_data_duplicate.head()
```

Out[24]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	De
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	

In [25]:
```
customer_data_duplicate.head()
```

Out[25]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	De
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	

In [26]:
```
customer_data_duplicate.info()
```

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

In [27]:
```
customer_data_duplicate['TotalCharges'].nunique()
```

Out[27]:
```
6531
```

In [28]:
```
customer_data_duplicate['TotalCharges'].unique()
```

Out[28]:
```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

In [31]:
```
customer_data_duplicate['TotalCharges']=pd.to_numeric(customer_data_duplicate['TotalCharges'], errors='coerce')
```

```
customer_data_duplicate['TotalCharges'].dtypes
```

Out[31]:
```
dtype('float64')
```

In [34]:
```
mean_value = customer_data_duplicate['TotalCharges'].mean()
print("Mean of the column:", mean_value)
```

```
Mean of the column: 2283.3004408418656
```

In [35]:
```
customer_data_duplicate['TotalCharges'].fillna(mean_value, inplace=True)
```

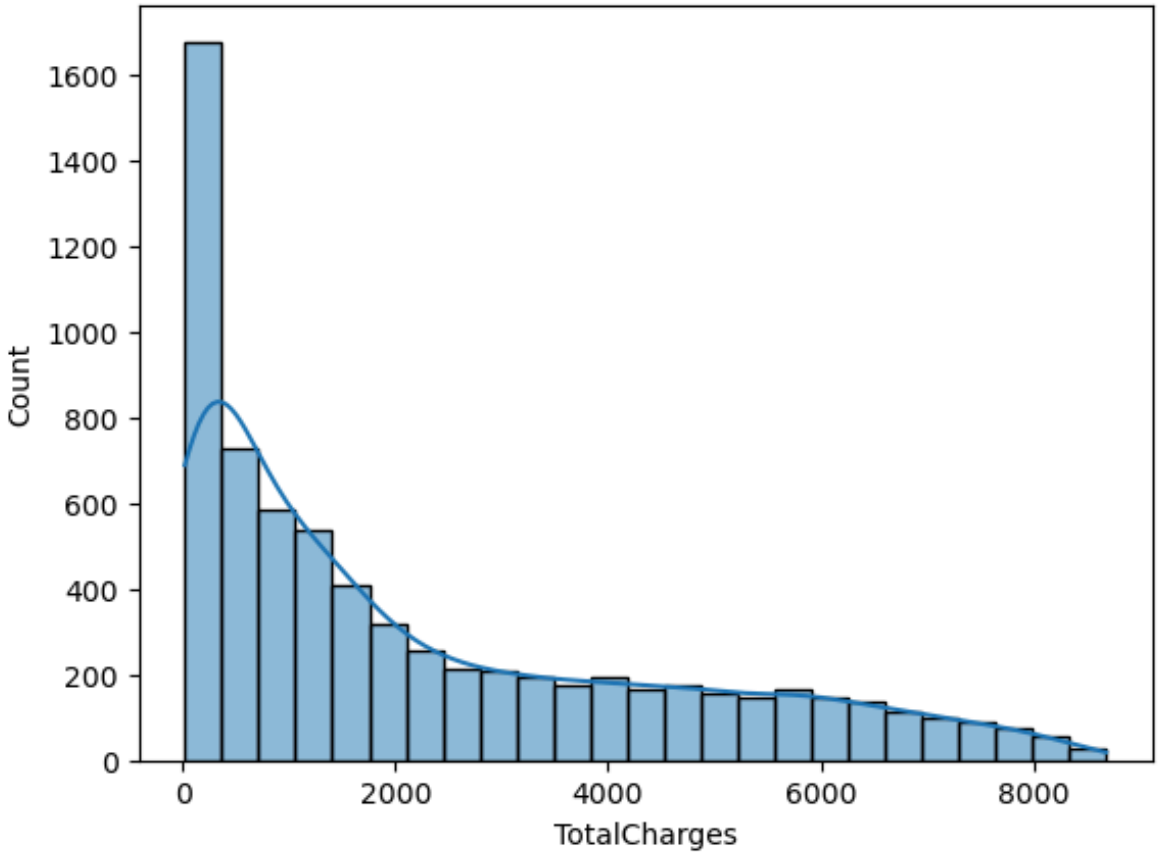
In [36]:
```
customer_data_duplicate['TotalCharges'].isnull().sum()
```

Out[36]:
```
0
```

In [37]:
```
customer_data_duplicate.info()
```

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

In [38]:
```
sns.histplot(data=customer_data_duplicate,x='TotalCharges',kde=True)
plt.show()
```

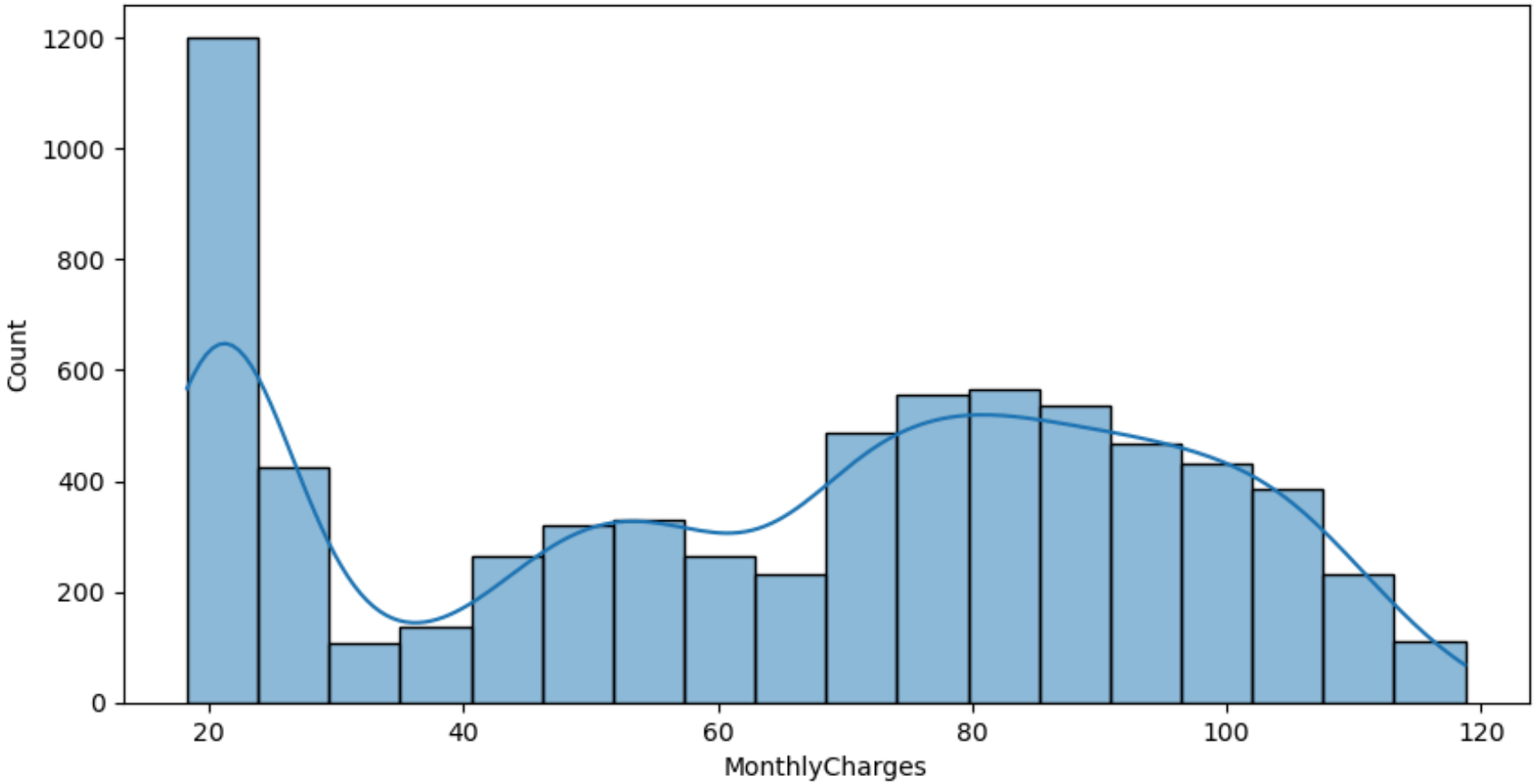


In [39]:
```
from scipy.stats import mannwhitneyu

# Split the groups
group_yes = customer_data_duplicate[customer_data_duplicate['Churn']=='Yes']['TotalCharges']
group_no = customer_data_duplicate[customer_data_duplicate['Churn']=='No']['TotalCharges']

# Mann-Whitney U test
stat, p = mannwhitneyu(group_yes, group_no)
print('U statistic = %.3f, p-value = %.3f' % (stat, p))
```

```
U statistic = 3365890.000, p-value = 0.000
```

In [40]:
```
plt.figure(figsize=(10,5))
sns.histplot(data=customer_data_duplicate,x='MonthlyCharges',kde=True)
```

```
plt.show()
```



`'TotalCharges'` and `'MonthlyCharges'` are related to `'Churn'`

In [41]:
```
group_yes = customer_data_duplicate[customer_data_duplicate['Churn']=='Yes']['MonthlyCharges']
group_no = customer_data_duplicate[customer_data_duplicate['Churn']=='No']['MonthlyCharges']

# Mann-Whitney U test
stat, p = mannwhitneyu(group_yes, group_no)
print('U statistic = %.3f, p-value = %.3f' % (stat, p))
```

```
U statistic = 6003125.500, p-value = 0.000
```

In [42]:
```
group_yes = customer_data_duplicate[customer_data_duplicate['Churn']=='Yes']['SeniorCitizen']
group_no = customer_data_duplicate[customer_data_duplicate['Churn']=='No']['SeniorCitizen']

# Mann-Whitney U test
stat, p = mannwhitneyu(group_yes, group_no)
print('U statistic = %.3f, p-value = %.3f' % (stat, p))
```

```
U statistic = 5444138.000, p-value = 0.000
```

`'SeniorCitizen'` and `'Churn'` are also

In [43]:
```
group_yes = customer_data_duplicate[customer_data_duplicate['Churn']=='Yes']['tenure']
group_no = customer_data_duplicate[customer_data_duplicate['Churn']=='No']['tenure']

# Mann-Whitney U test
stat, p = mannwhitneyu(group_yes, group_no)
print('U statistic = %.3f, p-value = %.3f' % (stat, p))
```

```
U statistic = 2515538.000, p-value = 0.000
```

`'tenure'` is also related to `'churn'`

In [44]:
```
customer_data_duplicate.select_dtypes(include=object).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   gender            7043 non-null   object
 1   Partner           7043 non-null   object
 2   Dependents        7043 non-null   object
 3   PhoneService      7043 non-null   object
 4   MultipleLines     7043 non-null   object
 5   InternetService   7043 non-null   object
 6   OnlineSecurity    7043 non-null   object
 7   OnlineBackup      7043 non-null   object
 8   DeviceProtection  7043 non-null   object
 9   TechSupport       7043 non-null   object
 10  StreamingTV       7043 non-null   object
 11  StreamingMovies   7043 non-null   object
 12  Contract          7043 non-null   object
 13  PaperlessBilling  7043 non-null   object
 14  PaymentMethod     7043 non-null   object
 15  Churn             7043 non-null   object
dtypes: object(16)
memory usage: 880.5+ KB
```

In [45]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['gender'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[45]:
```
(0.4840828822091383,
 0.48657873605618596,
 1,
 array([[2562.38989067,  925.61010933],
        [2611.61010933,  943.38989067]]))
```

In [46]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['Partner'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[46]:
```
(158.7333820309922,
 2.1399113440759935e-36,
 1,
 array([[2674.78830044,  966.21169956],
        [2499.21169956,  902.78830044]]))
```

partner and churn are dependent

In [47]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['Dependents'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[47]:
```
(189.12924940423474,
 4.9249216612154196e-43,
 1,
 array([[3623.93042737, 1309.06957263],
        [1550.06957263,  559.93042737]]))
```

dependents and churn are related

In [48]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['PhoneService'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[48]:
```
(0.9150329892546948,
 0.3387825358066928,
 1,
 array([[ 501.01774812,  180.98225188],
        [4672.98225188, 1688.01774812]]))
```

In [49]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['MultipleLines'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[49]:
```
(11.33044148319756,
 0.0034643829548773,
 2,
 array([[2490.39613801,  899.60386199],
        [ 501.01774812,  180.98225188],
        [2182.58611387,  788.41388613]]))
```

multiplelines and churn are related

In [50]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['InternetService'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[50]:
```
(732.309589667794,
 9.571788222840544e-160,
 2,
 array([[1778.53954281,  642.46045719],
        [2274.41488002,  821.58511998],
        [1121.04557717,  404.95442283]]))
```

internetservice and churn are related

In [51]:
```
from scipy.stats import  chi2_contingency
ct = pd.crosstab(customer_data_duplicate['OnlineSecurity'], customer_data_duplicate['Churn'])
chi2,p,dof,expected=chi2_contingency(ct)
chi2,p,dof,expected
```

Out[51]:
```
(849.9989679615965,
 2.661149635176552e-185,
 2,
 array([[2569.73619196,  928.26380804],
        [1121.04557717,  404.95442283],
        [1483.21823087,  535.78176913]]))
```

onlinesecurity and churn are related

In [52]:
```
import pandas as pd
from scipy.stats import chi2_contingency
```

```
cat_columns = [
   'OnlineBackup',
    'DeviceProtection',
    'TechSupport',
   'StreamingTV',
   'StreamingMovies',
   'Contract',
   'PaperlessBilling',
   'PaymentMethod'
]

results = []

for col in cat_columns:
    ct = pd.crosstab(customer_data_duplicate[col], customer_data_duplicate['Churn'])
    chi2, p, dof, expected = chi2_contingency(ct)
    results.append({
        'Column': col,
        'Chi2': chi2,
        'p-value': p,
        'dof': dof
    })


chi2_results = pd.DataFrame(results)
chi2_results
```

Out[52]:

	Column	Chi2	p-value	dof
0	OnlineBackup	601.812790	2.079759e-131	2
1	DeviceProtection	558.419369	5.505219e-122	2
2	TechSupport	828.197068	1.443084e-180	2
3	StreamingTV	374.203943	5.528994e-82	2
4	StreamingMovies	375.661479	2.667757e-82	2
5	Contract	1184.596572	5.863038e-258	2
6	PaperlessBilling	258.277649	4.073355e-58	1
7	PaymentMethod	648.142327	3.682355e-140	3

`'OnlineBackup'`, `'DeviceProtection'`, `'TechSupport'`, `'StreamingTV'`, `'StreamingMovies'`, `'Contract'`, `'PaperlessBilling'` this are statistically related to `'Churn'`

In [53]:
```
customer_data_duplicate.info()
```

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

except `'gender'` and `'phoneservice'` remaining all are related statistically to `'churn'`

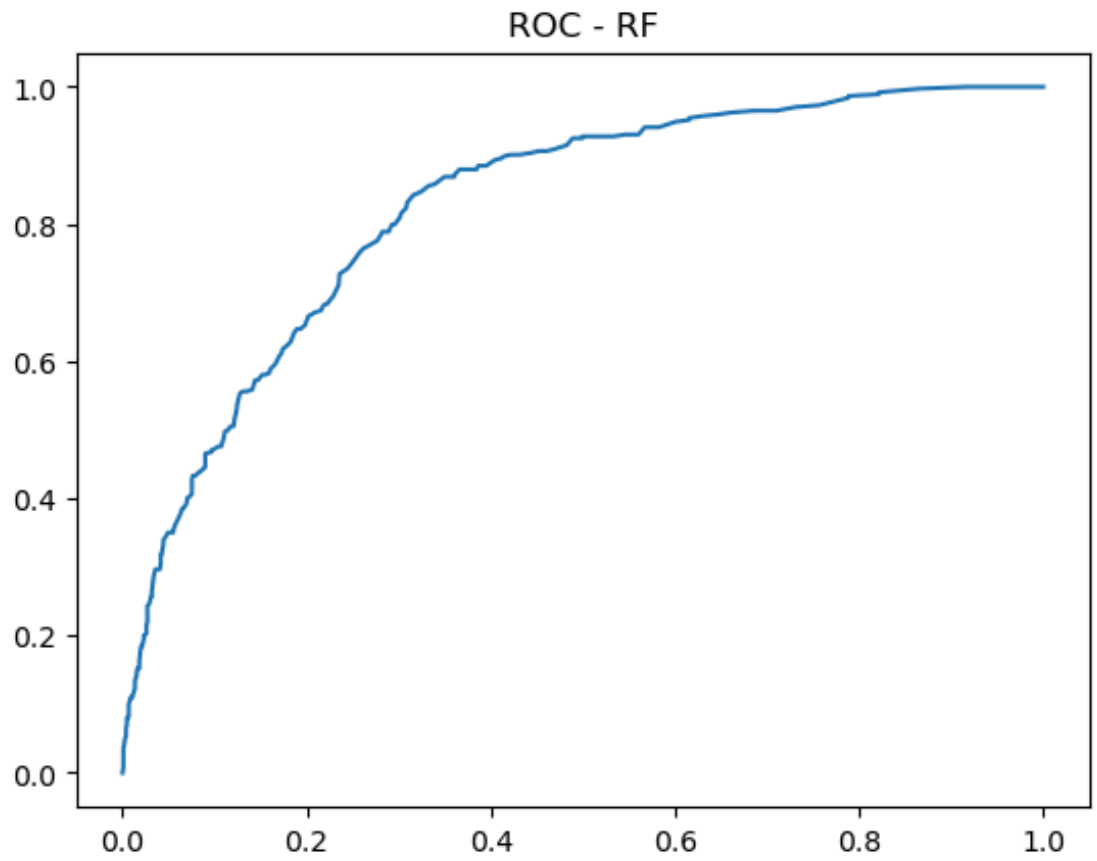
Modeling

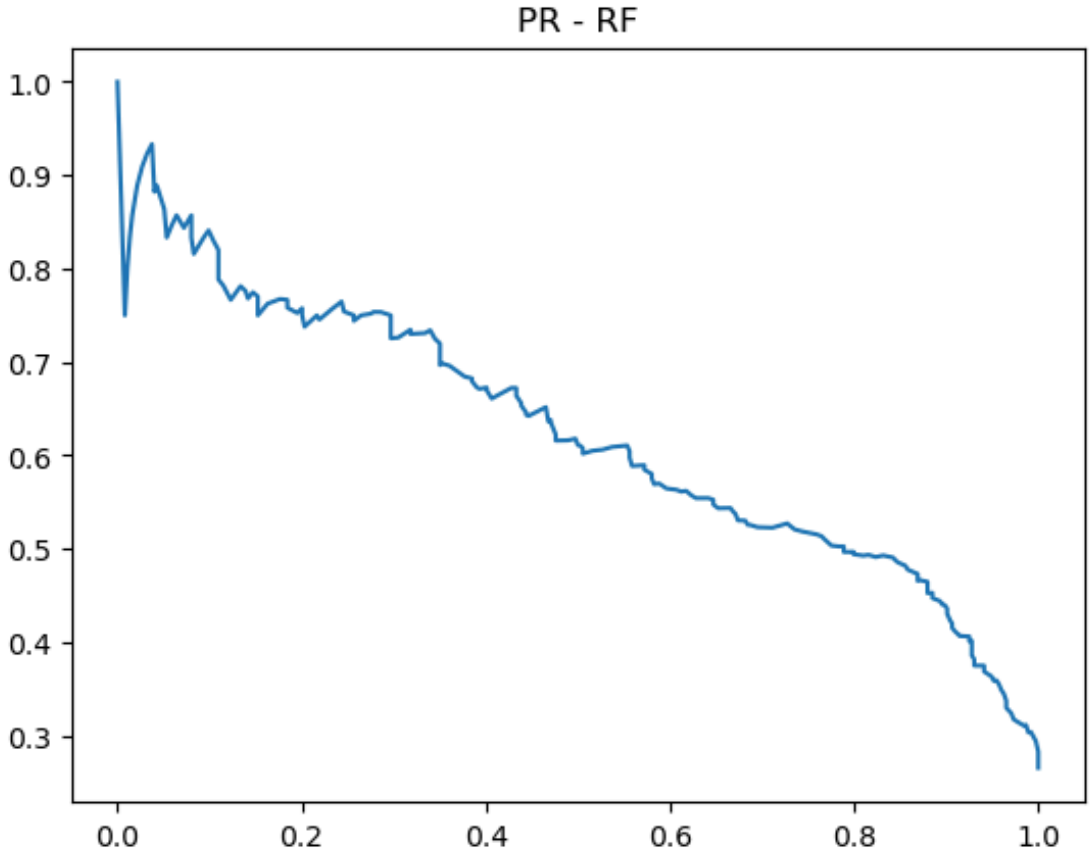
In [69]:
```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, precision_re
from imblearn.over_sampling import SMOTE

customer_data_duplicate['TotalCharges'] = pd.to_numeric(customer_data_duplicate['TotalCharges'], errors='coerce')
customer_data_duplicate['TotalCharges'] = customer_data_duplicate['TotalCharges'].fillna(customer_data_duplicate['TotalChar


customer_data_duplicate['AvgMonthlySpend'] = customer_data_duplicate['TotalCharges'] / (customer_data_duplicate['tenure']+1
customer_data_duplicate['tenure_group'] = pd.cut(customer_data_duplicate['tenure'], bins=[0,6,12,24,48,72,np.inf],
                            labels=['0-5','6-11','12-23','24-47','48-71','72+'])
customer_data_duplicate['Payment_Electronic'] = customer_data_duplicate['PaymentMethod'].apply(lambda x: 1 if 'electronic'
y = customer_data_duplicate['Churn'].map({'No':0,'Yes':1})
X = pd.get_dummies(customer_data_duplicate.drop(columns=['Churn','customerID']), drop_first=True)

# Split & scale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
num_cols = ['tenure','MonthlyCharges','TotalCharges','AvgMonthlySpend']
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols]  = scaler.transform(X_test[num_cols])

# Defining models
models = {
    "Logistic": LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42),
    "RandomForest": RandomForestClassifier(n_estimators=200, class_weight='balanced_subsample', random_state=42),
    "GradientBoosting": GradientBoostingClassifier(n_estimators=200, random_state=42)
}

# Training & evaluation
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_prob = model.predict_proba(X_test)[:,1]
    y_pred = (y_prob >= 0.5).astype(int)
    results.append([name,
        accuracy_score(y_test,y_pred),
        recall_score(y_test,y_pred),
        precision_score(y_test,y_pred),
        f1_score(y_test,y_pred),
        roc_auc_score(y_test,y_prob)])
```

```
results_df = pd.DataFrame(results, columns=["Model","Acc","Recall","Precision","F1","ROC_AUC"])
print(results_df)

X_res, y_res = SMOTE(random_state=42).fit_resample(X_train, y_train)
rf_sm = RandomForestClassifier(n_estimators=250, max_depth=12, min_samples_leaf=4,
                               class_weight='balanced', random_state=42)
rf_sm.fit(X_res, y_res)
y_prob_sm = rf_sm.predict_proba(X_test)[:,1]
y_pred_sm = (y_prob_sm >= 0.5).astype(int)
print("RF+SMOTE:", accuracy_score(y_test,y_pred_sm),
      recall_score(y_test,y_pred_sm), precision_score(y_test,y_pred_sm),
      f1_score(y_test,y_pred_sm), roc_auc_score(y_test,y_prob_sm))

# Threshold tuning
y_prob_rf = models["RandomForest"].predict_proba(X_test)[:,1]
for t in [0.3,0.4,0.5]:
    y_pred_t = (y_prob_rf >= t).astype(int)
    print(f"Thresh={t}: Recall={recall_score(y_test,y_pred_t):.3f} | "
          f"Acc={accuracy_score(y_test,y_pred_t):.3f} | "
          f"F1={f1_score(y_test,y_pred_t):.3f}")

#  Plots (example ROC + PR for RF)
fpr,tpr,_ = roc_curve(y_test,y_prob_rf)
precision,recall,_ = precision_recall_curve(y_test,y_prob_rf)
plt.figure(); plt.plot(fpr,tpr); plt.title("ROC - RF")
plt.figure(); plt.plot(recall,precision); plt.title("PR - RF"); plt.show()
```

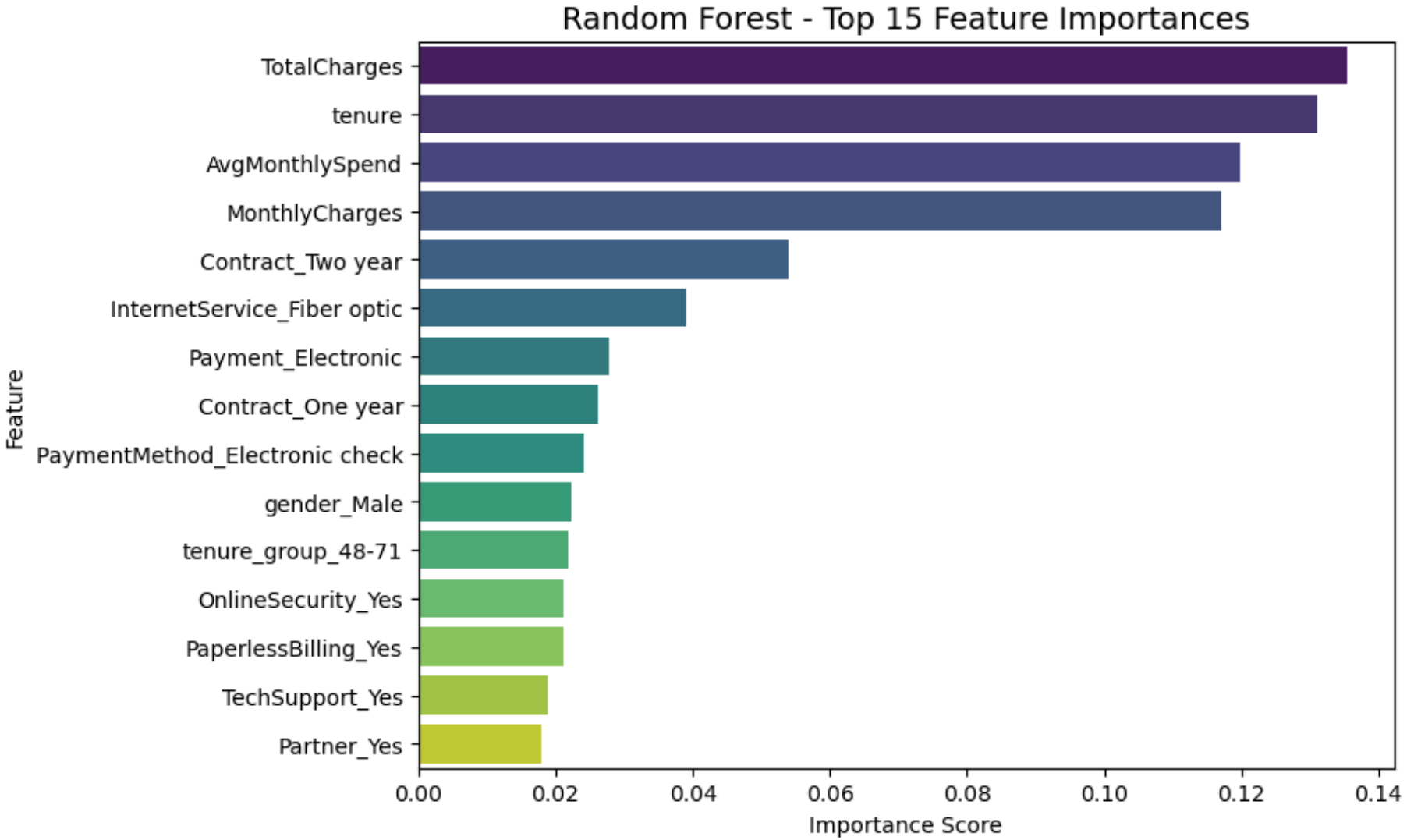
```
              Model       Acc    Recall  Precision        F1   ROC_AUC
0          Logistic  0.740241  0.796791   0.506803  0.619543  0.846891
1      RandomForest  0.784954  0.497326   0.617940  0.551111  0.825061
2  GradientBoosting  0.794180  0.494652   0.646853  0.560606  0.837893
RF+SMOTE: 0.7629524485450674 0.7459893048128342 0.5386100386100386 0.625560538116592 0.8362034668940039
Thresh=0.3: Recall=0.730 | Acc=0.753 | F1=0.611
Thresh=0.4: Recall=0.607 | Acc=0.771 | F1=0.584
Thresh=0.5: Recall=0.497 | Acc=0.785 | F1=0.551
```


ROC - RF
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

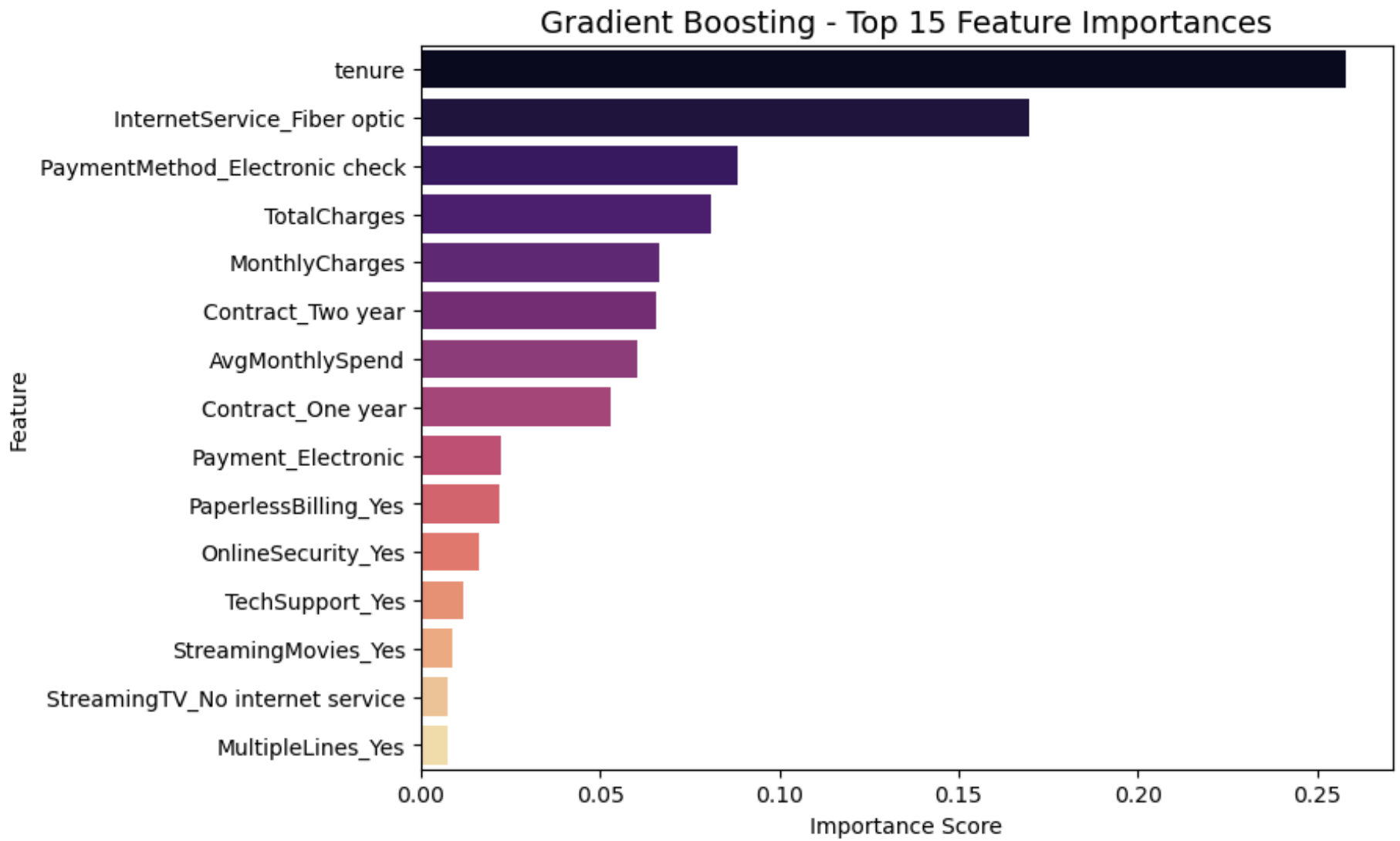


In [57]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Get feature importance from RF model
importances = models["RandomForest"].feature_importances_
feat_importances = pd.Series(importances, index=X_train.columns)

# Top 15 features
top_feats = feat_importances.sort_values(ascending=False)[:15]

# Plot
```

```
plt.figure(figsize=(8,6))
sns.barplot(x=top_feats.values, y=top_feats.index,hue=top_feats.index, palette="viridis")
plt.title("Random Forest - Top 15 Feature Importances", fontsize=14)
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```



In [58]:
```
# Get feature importance from GB model
gb_importances = models["GradientBoosting"].feature_importances_
gb_feats = pd.Series(gb_importances, index=X_train.columns)

# Top 15 features
top_gb = gb_feats.sort_values(ascending=False)[:15]

# Plot
plt.figure(figsize=(8,6))
sns.barplot(x=top_gb.values, y=top_gb.index, hue=top_gb.index,palette="magma")
plt.title("Gradient Boosting - Top 15 Feature Importances", fontsize=14)
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```



Conclusion

1. Data preprocessing and feature engineering improved model readiness, including handling missing TotalCharges and creating AvgMonthlySpend and tenure_group features.

2. Random Forest and Gradient Boosting performed well, with ROC AUC > 0.85. SMOTE helped improve recall, which is crucial for predicting churn.

3. Feature importance analysis showed key drivers of churn:

    - Contract type, tenure, MonthlyCharges, and PaymentMethod are top factors.

4. Threshold tuning allows adjusting the balance between recall and precision depending on business priorities (e.g., catching more churners vs avoiding false positives).

Overall, this workflow provides actionable insights for customer retention strategies.

In [ ]: