Telco Customer Churn Prediction

Group 3

Pramoth Guhan

Haritha Anand

Loading libraries and data:

In [1]:
```
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classificatio
```

In [3]:
```
df = pd.read_csv(r'/Users/harithaanand/Downloads/WA_Fn-UseC_-Telco-Customer-Churn.csv')
df
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	Tech
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	No	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	No	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	Yes	

7043 rows × 21 columns

In [4]:
```
df.columns.values
```

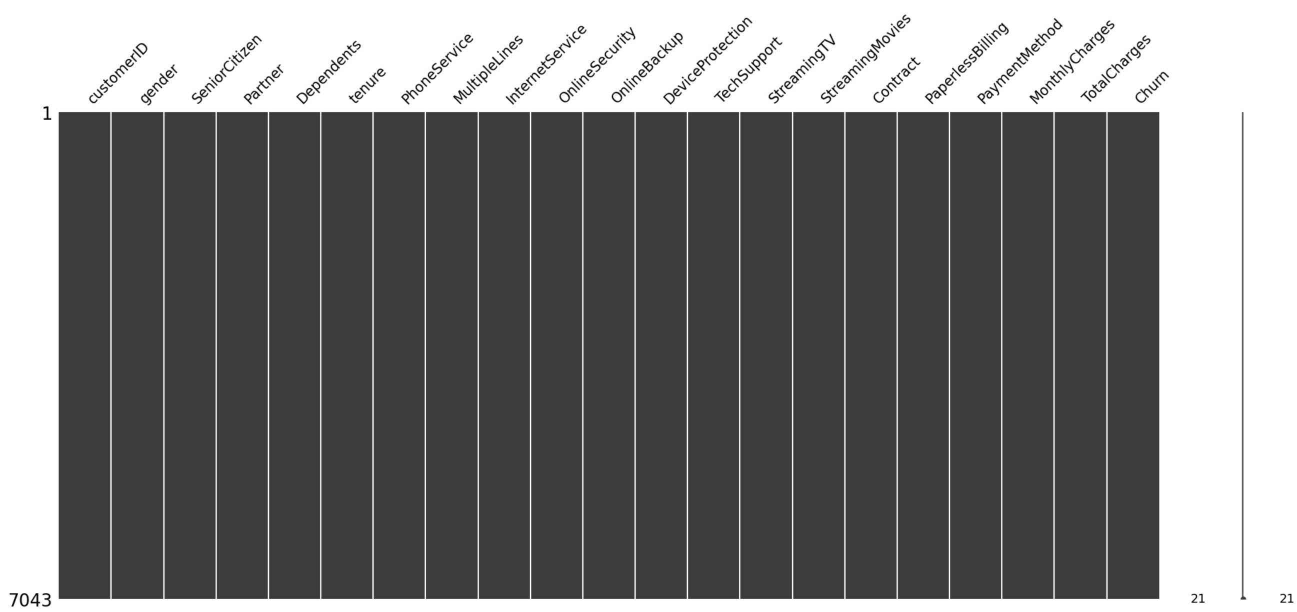
Out[4]:
```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

In [5]:
```
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]:
```
# Visualize missing values as a matrix
msno.matrix(df)
```

Out[6]:
```
<Axes: >
```



Data Manuplation:

In [7]:
```
# Drop Unnecessary columns
df = df.drop(['customerID'], axis=1)
df.head(3)
```

Out[7]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupp
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	

In [8]:
```
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
df.isnull().sum()
```

Out[8]:
```
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
```

Its found there are some whitespaces in TotalCharges

In [9]:
```
# check num in Total Charges column
df[np.isnan(df['TotalCharges'])]
```

Out[9]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechS
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No i
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No i
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No i
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No i
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No i
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No i
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	

- Some client tenure is 0 while it has monthly charges.
- Checking if there are other 0 values in tenure columns.

In [10]:
```
df[df['tenure']==0].index
```

Out[10]:
```
Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

- No other missing values were found in tenure except the above
- Deleting the above missing values in tenuure with charges

In [11]:
```
df.drop(labels=df[df['tenure']==0].index, axis=0, inplace=True) # row delelation
df[df['tenure']==0].index # now no missing values
```

Out[11]:
```
Index([], dtype='int64')
```

In [12]:
```
df['TotalCharges'].isnull().sum()
```

Out[12]:
```
0
```

In [13]:
```
df.isnull().sum()
```

Out[13]:
```
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
```

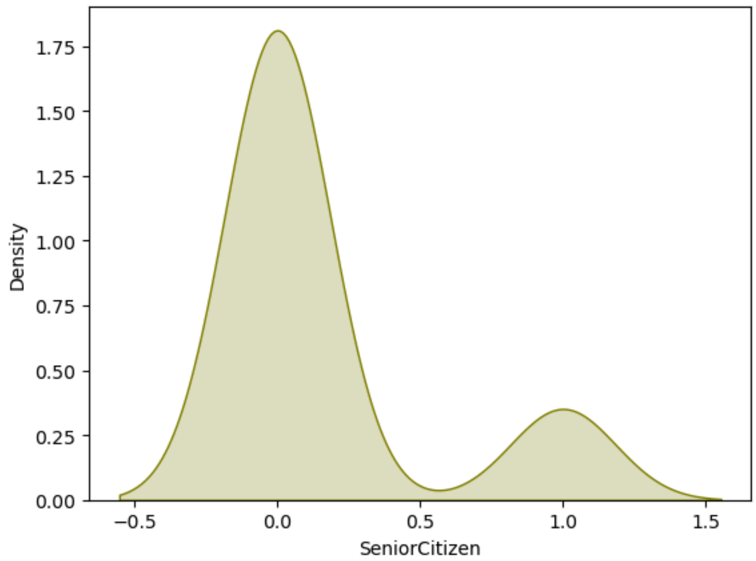
Exploratory Data Analysis:

In [14]:
```
df['SeniorCitizen'].value_counts()
```

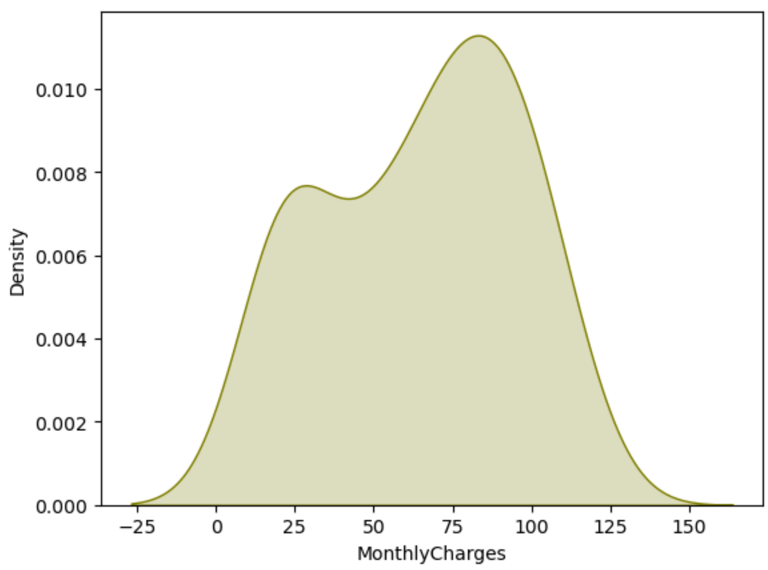
Out[14]:
```
SeniorCitizen
0    5890
1    1142
Name: count, dtype: int64
```

Probability Distribution

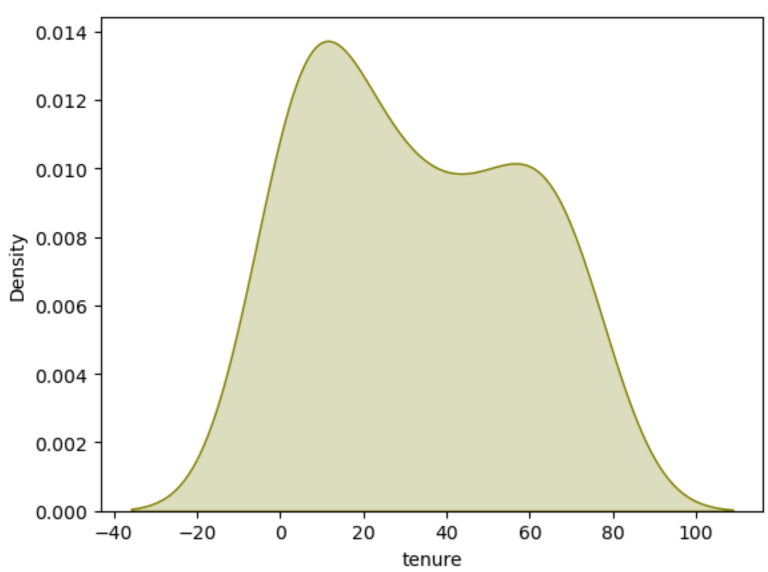
In [15]:
```
sns.kdeplot(df['SeniorCitizen'], shade=True, bw=0.5, color="olive")
plt.show()
```



In [16]:
```
sns.kdeplot(df['MonthlyCharges'], shade=True, bw=0.5, color="olive")
plt.show()
```



In [17]:
```
sns.kdeplot(df['tenure'], shade=True, bw=0.5, color="olive")
plt.show()
```



In [18]:
```
df['SeniorCitizen'] = df['SeniorCitizen'].map({0:'No', 1:'Yes'})
df.head()
```

Out[18]:

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

Checking Column Types:

In [19]:
```
df.select_dtypes('float64')
```

Out[19]:

	MonthlyCharges	TotalCharges
0	29.85	29.85
1	56.95	1889.50
2	53.85	108.15
3	42.30	1840.75
4	70.70	151.65
...	...	...
7038	84.80	1990.50
7039	103.20	7362.90
7040	29.60	346.45
7041	74.40	306.60
7042	105.65	6844.50

7032 rows × 2 columns

In [20]:
```
df.select_dtypes('int64')
```

Out[20]:

	tenure
0	1
1	34
2	2
3	45
4	2
...	...
7038	24
7039	72
7040	11
7041	4
7042	66

7032 rows × 1 columns

In [21]:
```
df.select_dtypes('object').columns
```

Out[21]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn'],
      dtype='object')
```

In [22]:
```
# Continuous Features
numerical_cols = ['tenure','MonthlyCharges','TotalCharges',]
df[numerical_cols].describe()
```

Out[22]:

	tenure	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000
mean	32.421786	64.798208	2283.300441
std	24.545260	30.085974	2266.771362
min	1.000000	18.250000	18.800000
25%	9.000000	35.587500	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.862500	3794.737500
max	72.000000	118.750000	8684.800000

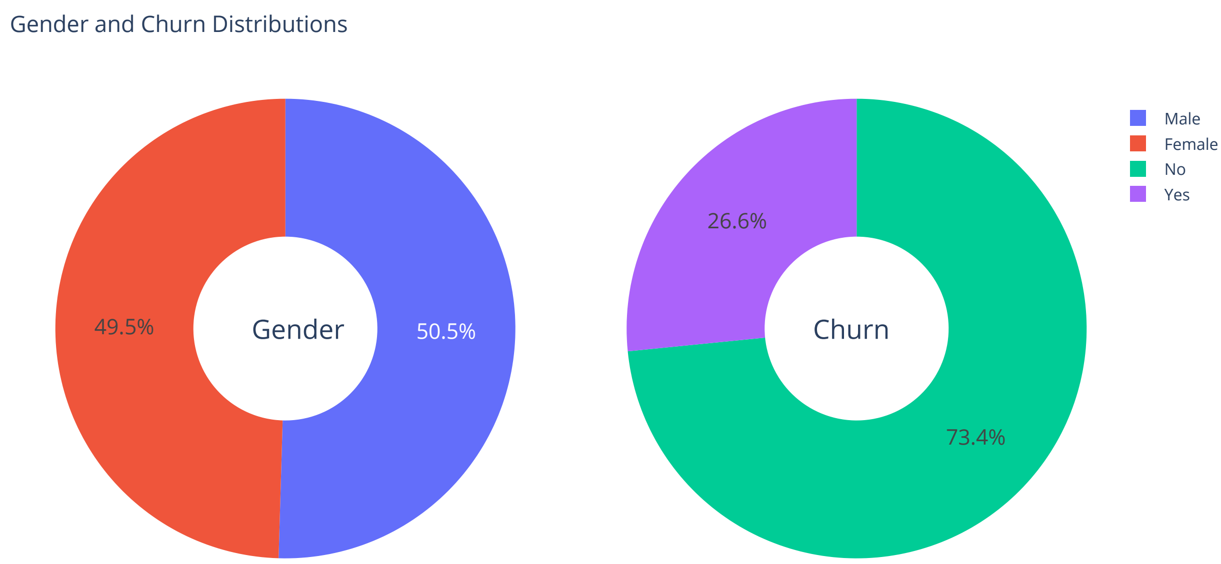
In [23]:
```
df["InternetService"].describe(include=['object','bool'])
```

Out[23]:
```
count            7032
unique              3
top       Fiber optic
freq             3096
Name: InternetService, dtype: object
```

Data Visualization:

1. Gender and Churn Distribution:

In [24]:
```
g_labels = ['Male', 'Female']
c_labels = ['No', 'Yes']
# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=g_labels, values=df['gender'].value_counts(), name="Gender"),
              1, 1)
fig.add_trace(go.Pie(labels=c_labels, values=df['Churn'].value_counts(), name="Churn"),
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Gender and Churn Distributions",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='Gender', x=0.19, y=0.5, font_size=20, showarrow=False),
                 dict(text='Churn', x=0.81, y=0.5, font_size=20, showarrow=False)])
fig.show()
```



- There is a balanced ratio in gender
- Over 26% customer churned or discontinue from the company services

2. Customer Churn Ratio:

In [25]:
```
# How many customer still taking service / not churn by gender- Regular Customer
df["Churn"][df["Churn"]=="No"].groupby(by=df["gender"]).count()
```

Out[25]:
```
gender
Female    2544
Male      2619
Name: Churn, dtype: int64
```

In [26]:
```
churn_no = 2544+2619
churn_no
```

Out[26]:
```
5163
```

In [27]:
```
# How many customer already churned by gender- Discontinuation
df["Churn"][df["Churn"]=="Yes"].groupby(by=df["gender"]).count()
```

Out[27]:
```
gender
Female    939
Male      930
Name: Churn, dtype: int64
```

In [28]:
```
churn_yes = 939+930
churn_yes
```

Out[28]:
```
1869
```

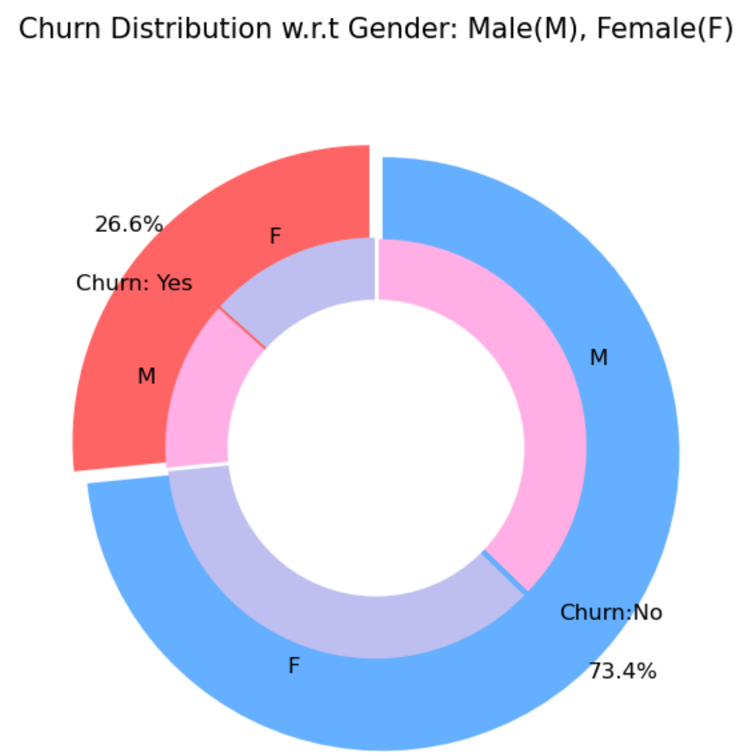
In [29]:
```
total_cus = df['gender'].count()
total_cus
```

Out[29]:
```
7032
```

In [30]:
```
churn_rate = churn_yes/total_cus*100
churn_rate
```

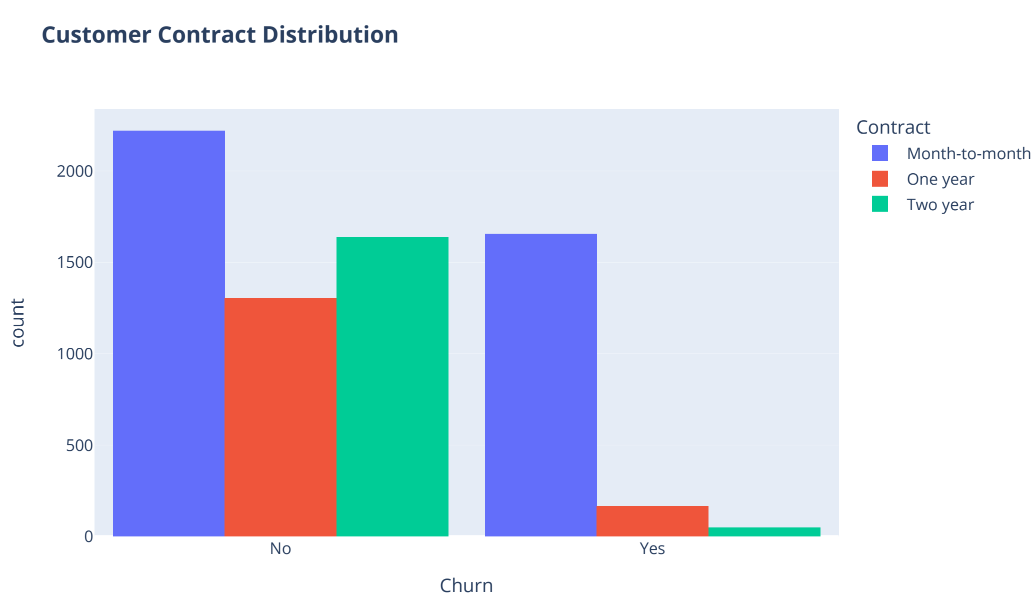
Out[30]:
```
26.578498293515356
```

In [31]:
```
plt.figure(figsize=(6, 6))
labels =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":12}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.09, labeldistance=0.8,colors=colors, startangle=90,fra
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textp
#Draw circle
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Churn Distribution w.r.t Gender: Male(M), Female(F)', fontsize=15, y=1.1)

# show plot

plt.axis('equal')
plt.tight_layout()
plt.show()
```



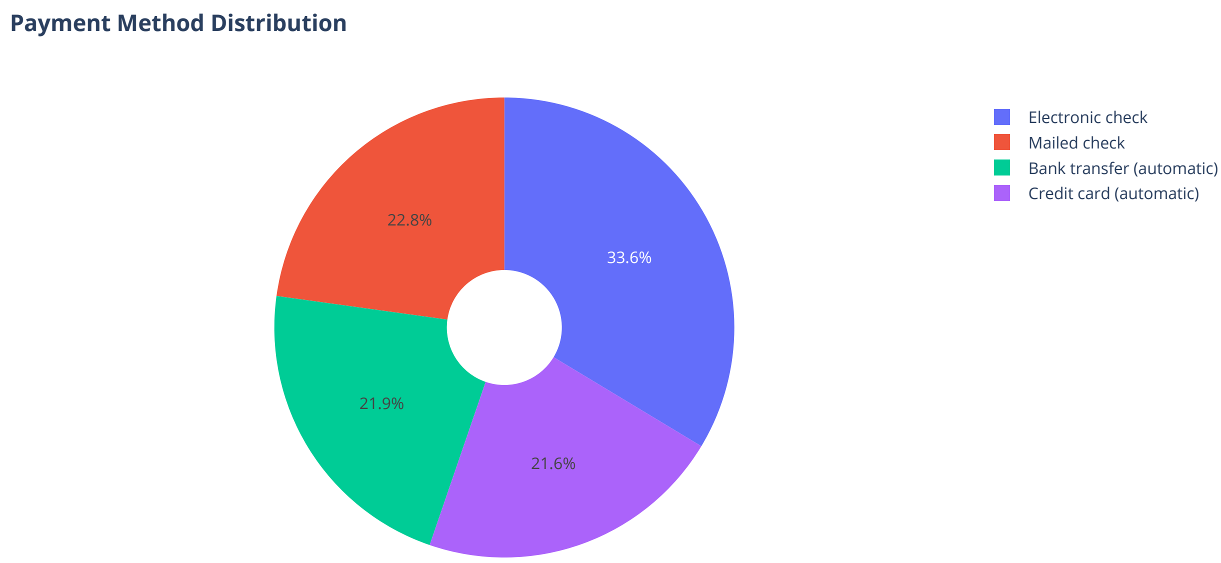
3. Customer ontract Distribution

In [32]:
```
fig = px.histogram(df, x="Churn", color="Contract", barmode='group', title="<b>Customer Contract Distribution</b>")
fig.update_layout(width=800, height=500, bargap=0.1)
fig.show()
```



4. Payment Method Distribution

In [33]:
```
labels = df['PaymentMethod'].unique()
values = df['PaymentMethod'].value_counts()
colors = ['#FF9999', '#66B2FF', '#99FF99', '#FFCC99']
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.25,)])
fig.update_layout(title_text="<b>Payment Method Distribution</b>")
fig.show()
```



- Customers using electronic payment methods exhibit a higher likelihood of churning.
- Conversely, customers utilizing credit card (automatic) or bank transfer methods are less likely to churn.

5. Internet Service & Churn Status by Gender

In [34]:
```
df['InternetService'].value_counts()
```

Out[34]:
```
InternetService
Fiber optic    3096
DSL            2416
No             1520
Name: count, dtype: int64
```

In [35]:
```
df['InternetService'].unique()
```

Out[35]:
```
array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

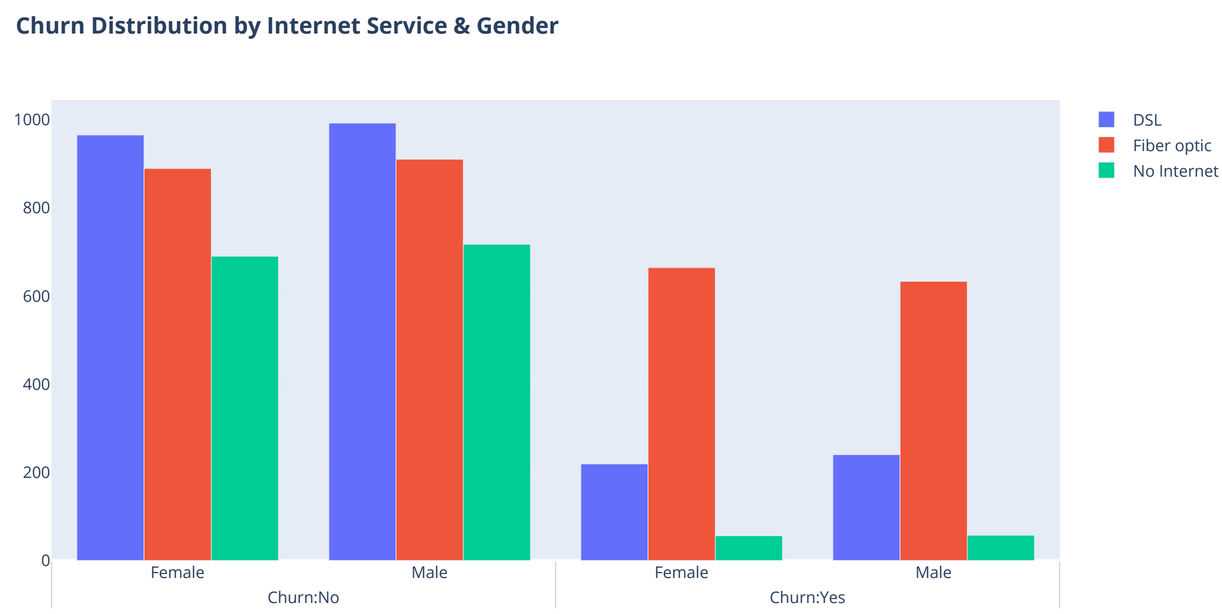
In [36]:
```
# Filtering by Male & Churn Status
df[df['gender']=="Male"][['InternetService','Churn']].value_counts()
```

Out[36]:
```
InternetService  Churn
DSL              No       992
Fiber optic      No       910
No               No       717
Fiber optic      Yes      633
DSL              Yes      240
No               Yes       57
Name: count, dtype: int64
```

In [37]:
```
# Filtering by Female & Churn Status
df[df['gender']=="Female"][['InternetService','Churn']].value_counts()
```

Out[37]:
```
InternetService  Churn
DSL              No       965
Fiber optic      No       889
No               No       690
Fiber optic      Yes      664
DSL              Yes      219
No               Yes       56
Name: count, dtype: int64
```

In [38]:
```
fig = go.Figure()

fig.add_trace(go.Bar(
    x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
        ['Female','Male','Female','Male']],
    y = [965, 992, 219,240],
    name = "DSL",
))

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [889, 910, 664, 633],
  name = 'Fiber optic',
))

fig.add_trace(go.Bar(
    x = [['Churn:No','Churn:No','Churn:Yes','Churn:Yes'],
        ['Female','Male','Female','Male']],
    y = [690,717,56,57],
    name = 'No Internet'
))

## Ploting
fig.update_layout(title_text='<b> Churn Distribution by Internet Service & Gender')
fig.show()
```



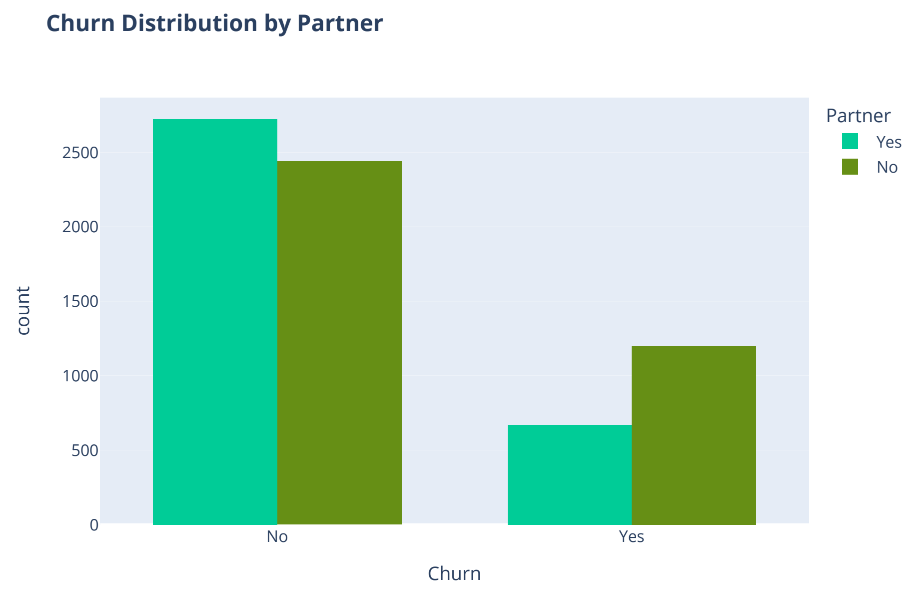
- It is found that there are significant higher churn rate at Fiber Optic, suggesting improving Fiber optic service
- DSL has less churn rate compared to the Fiber Optic

6. Churn Distribution by Partner

In [39]:
```
df['Partner'].value_counts()
```

Out[39]:
```
Partner
No     3639
Yes    3393
Name: count, dtype: int64
```

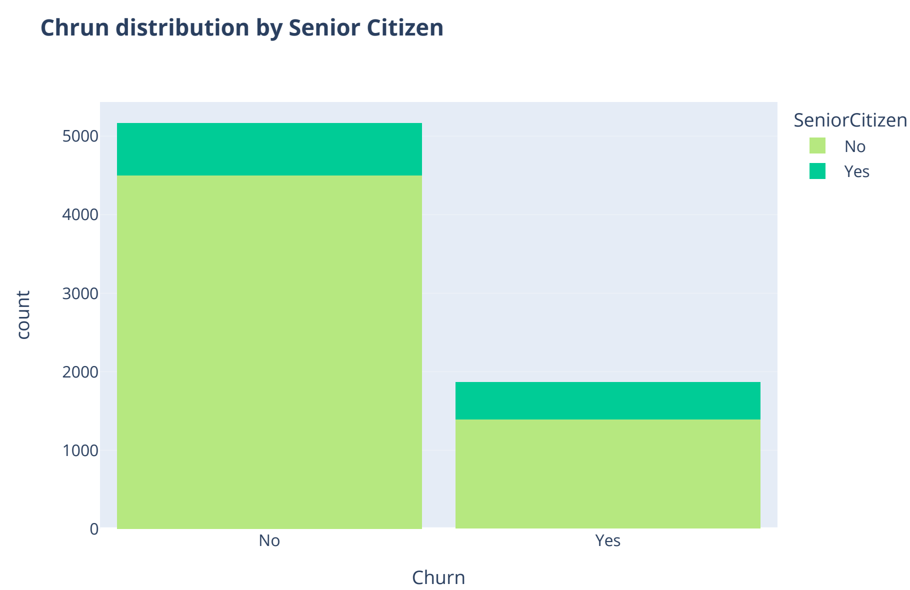
In [40]:
```
color_map = {"Yes": "#00CC97", "No":"#668F15"}
fig = px.histogram(df, x='Churn', color='Partner', barmode='group',
                   title="<b> Churn Distribution by Partner</b>",
                   color_discrete_map=color_map,
                  )
## Size & Display
fig.update_layout(width=700, height=500, bargap=0.3)
fig.show()
```



- Customer without partner are more likely to churn

7. Chrun distribution by Senior Citizen

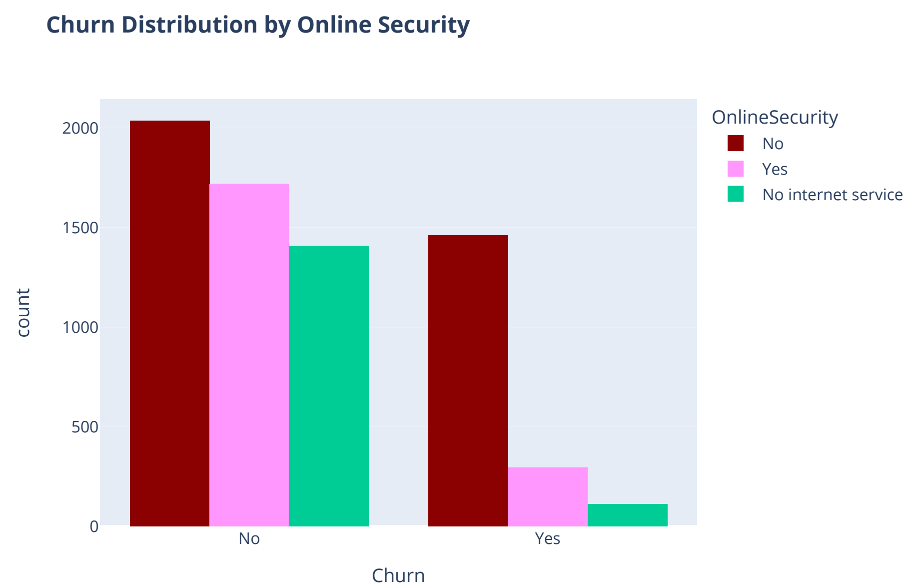
In [41]:
```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution by Senior Citizen</b>", color_(
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



- More number of churn is found in non-senior citizens.

8. Churn Distribution by Online Security

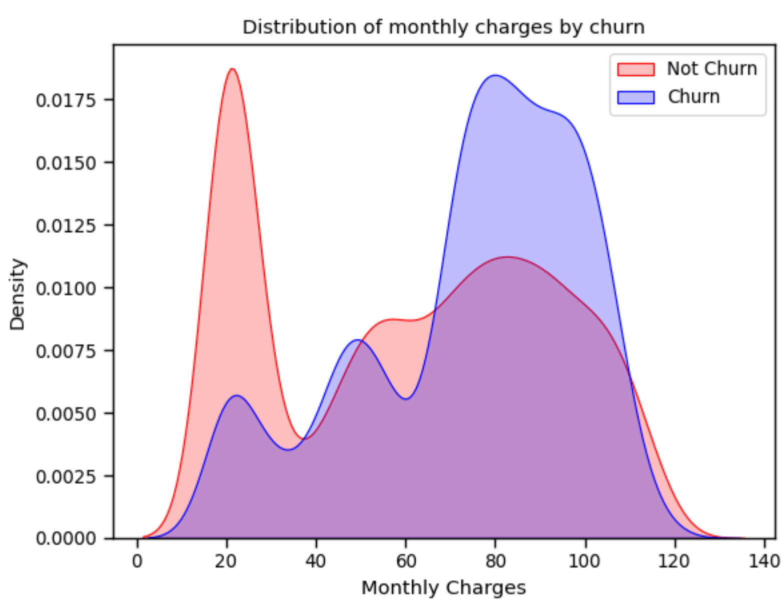
In [42]:
```
color_map = {"Yes": "#FF97FF", "No": "#8B0000"}
fig = px.histogram(df, x='Churn', color='OnlineSecurity',
                  barmode='group',title="<b> Churn Distribution by Online Security</a>",
                   color_discrete_map=color_map
                  )
fig.update_layout(width=700, height=500, bargap=0.2)
fig.show()
```



- It is obvious that more churn has occured when there is no online security.

9. Distribution of monthly charges by churn

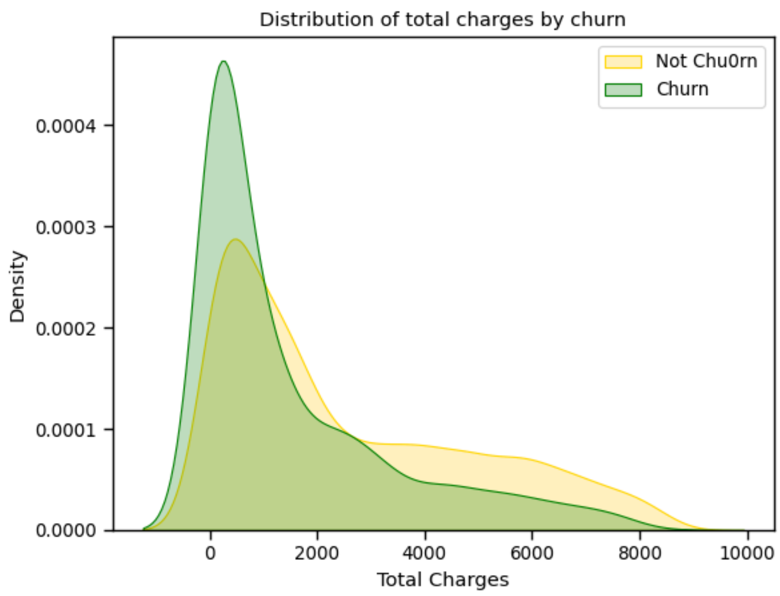
In [43]:
```
sns.set_context("paper",font_scale=1.1)
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'No') ],
                color="Red", shade = True);
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'Yes') ],
                ax =ax, color="Blue", shade= True);
ax.legend(["Not Churn","Churn"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Monthly Charges');
ax.set_title('Distribution of monthly charges by churn');
```



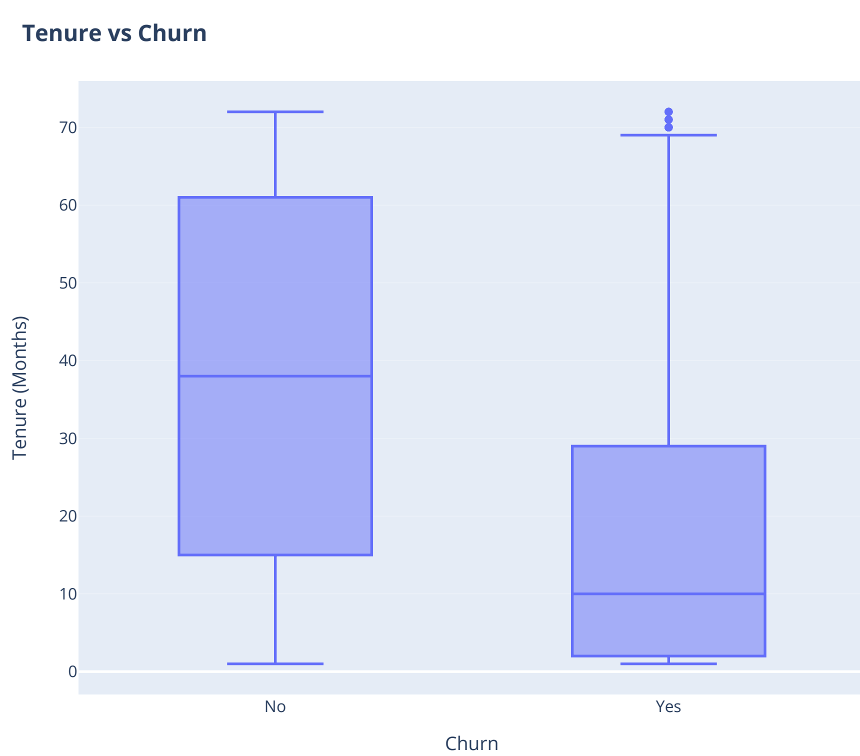
- The distribution for customers who churned (blue) is a bit more complex with two peaks: one around the 70-80 monthly charges mark and another smaller one at approximately 100. This suggests that higher monthly charges are associated with a higher rate of customer churn

10. Distribution of total charges by churn

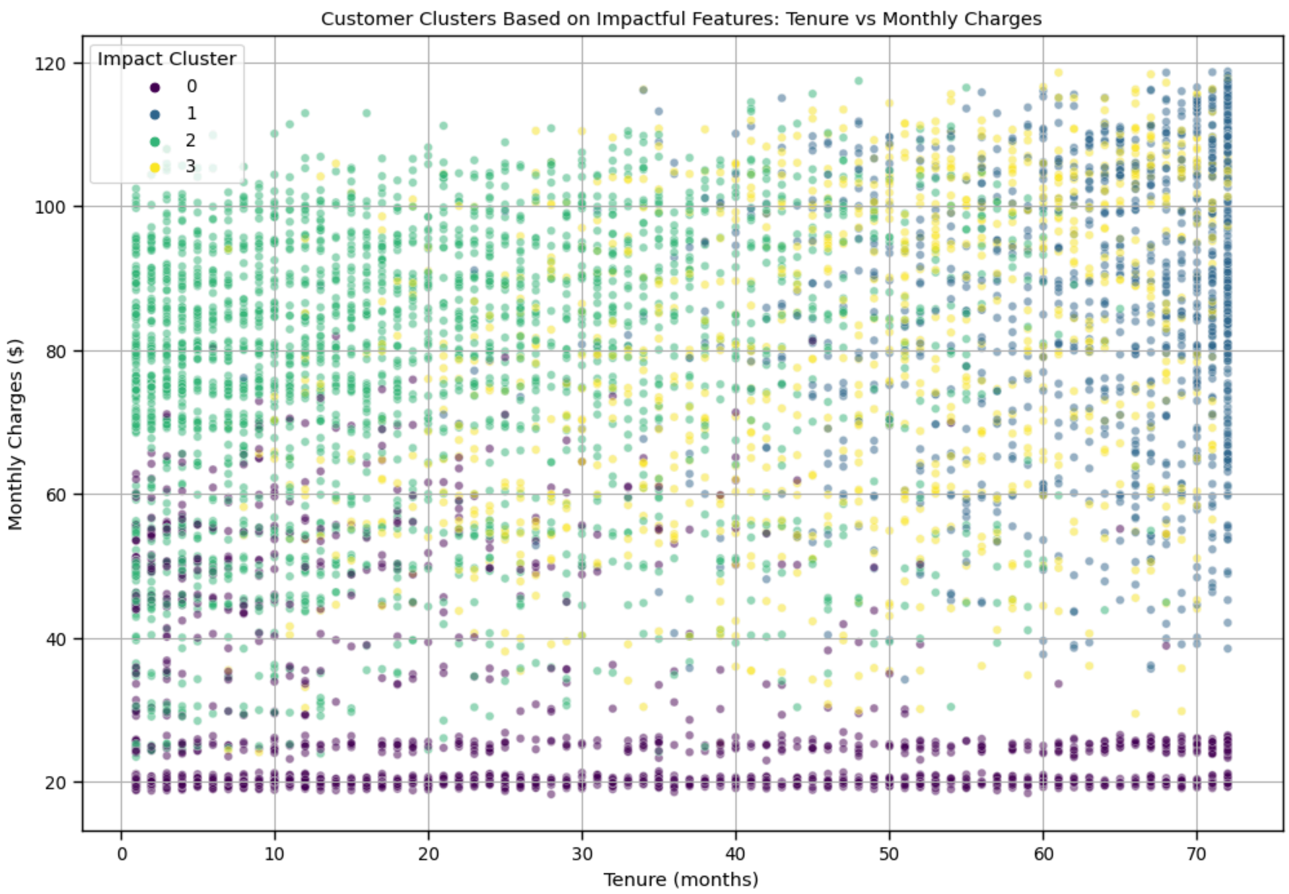
In [44]:
```
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'No') ],
                color="Gold", shade = True);
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'Yes') ],
                ax =ax, color="Green", shade= True);
ax.legend(["Not Chu0rn","Churn"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Total Charges');
ax.set_title('Distribution of total charges by churn');
```



In [45]:
```
fig = px.box(df, x='Churn', y = 'tenure')

# Update yaxis properties
fig.update_yaxes(title_text='Tenure (Months)', row=1, col=1)
# Update xaxis properties
fig.update_xaxes(title_text='Churn', row=1, col=1)

# Update size and title
fig.update_layout(autosize=True, width=750, height=600,
    title_font=dict(size=25, family='Courier'),
    title='<b>Tenure vs Churn</b>',
)

fig.show()
```



In [46]:
```
from sklearn.cluster import KMeans
# One-hot encoding the categorical columns
categorical_cols = ['InternetService', 'Contract', 'PaymentMethod']
data_encoded = pd.get_dummies(df[categorical_cols], drop_first=True)

# Including the numerical columns
impactful_features = df[['tenure', 'MonthlyCharges', 'TotalCharges']].join(data_encoded)

# Standardizing the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(impactful_features)

# Applying K-means clustering
kmeans_impact = KMeans(n_clusters=4, random_state=42)
clusters_impact = kmeans_impact.fit_predict(features_scaled)

# Adding the cluster labels to the original data
df['Impact_Cluster'] = clusters_impact

# Plotting the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(x='tenure', y='MonthlyCharges', hue='Impact_Cluster', palette='viridis', data=df, alpha=0.5)
plt.title('Customer Clusters Based on Impactful Features: Tenure vs Monthly Charges')
plt.xlabel('Tenure (months)')
plt.ylabel('Monthly Charges ($)')
plt.legend(title='Impact Cluster')
plt.grid(True)
plt.show()
```



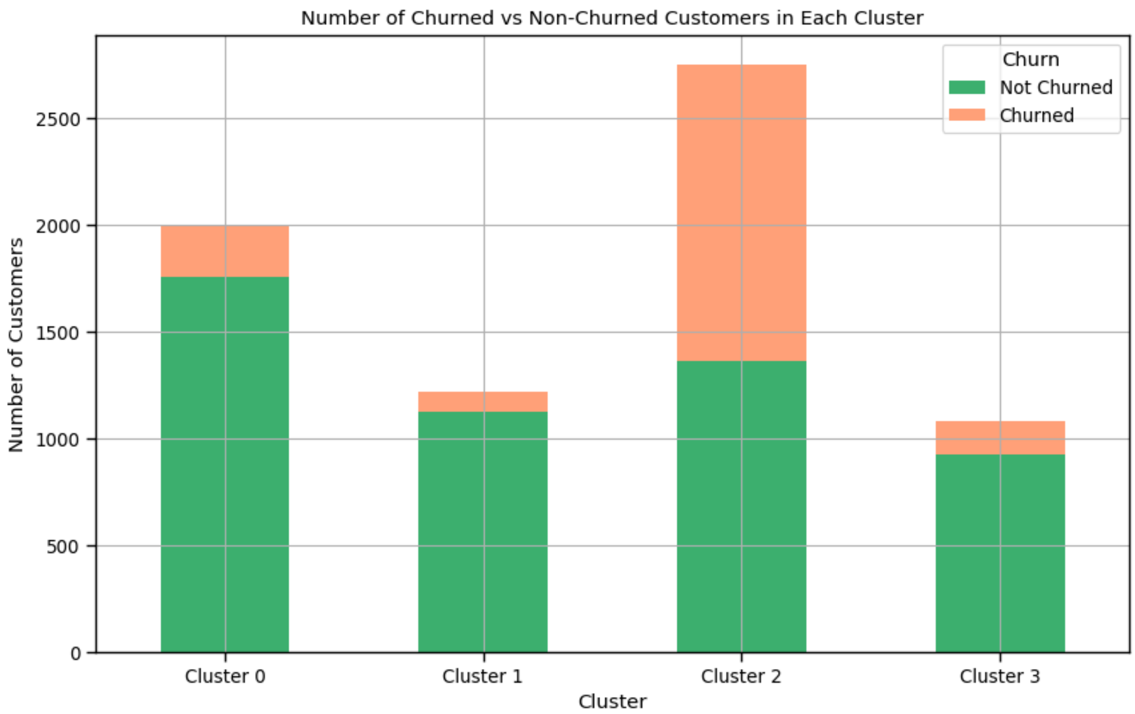
Cluster 0 (light green): customers with medium to high monthly charges and a range of tenure lengths. This could be a segment that uses more services or higher-tier services.

Cluster 1 (purple): This cluster have higher monthly charges and longer tenure, possibly representing loyal customers with high service usage or premium services.

Cluster 2 (yellow): Customers in this cluster have lower monthly charges and shorter tenure, indicating newer or more price-sensitive customers.

Cluster 3 (blue): Encompasses customers with lower to medium monthly charges and a broad range of tenure, possibly indicating a mixed segment with stable, yet budget-conscious customers.

In [47]:
```
#Counting churned and non-churned customers in each cluster
churn_counts = df.groupby(['Impact_Cluster', 'Churn']).size().unstack(fill_value=0)

# Plotting the churn counts
churn_counts.plot(kind='bar', stacked=True, color=['mediumseagreen', 'lightsalmon'], figsize=(10, 6))
plt.title('Number of Churned vs Non-Churned Customers in Each Cluster')
plt.xlabel('Cluster')
plt.ylabel('Number of Customers')
plt.xticks(ticks=range(4), labels=['Cluster 0', 'Cluster 1', 'Cluster 2', 'Cluster 3'], rotation=0)
plt.legend(title='Churn', labels=['Not Churned', 'Churned'])
plt.grid(True)
plt.show()
```



Data Preprocessing:

In [48]:
```
def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series
```

In [49]:
```
df = df.apply(lambda x: object_to_int(x))
df.head()
```

Out[49]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	...	TechSupport	Streaming
0	0	0	1	0	1	0	1	0	0	2	...	0	
1	1	0	0	0	34	1	0	0	2	0	...	0	
2	1	0	0	0	2	1	0	0	2	2	...	0	
3	1	0	0	0	45	0	1	0	2	0	...	2	
4	0	0	0	0	2	1	0	1	0	0	...	0	

5 rows × 21 columns

Correlation Checking:

- Mothly Charges & Paperless billing having comperative max correlation

Splitting Data and Training:

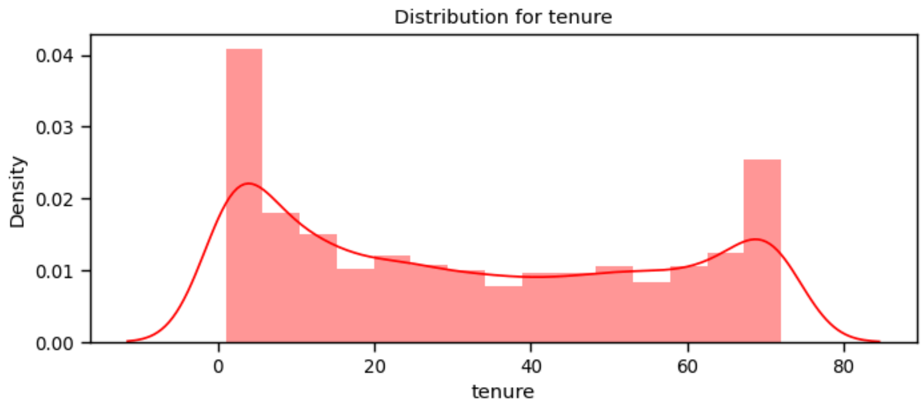
In [50]:
```
X = df.drop(columns=['Churn']) # Features
y = df['Churn'].values         # Target
```

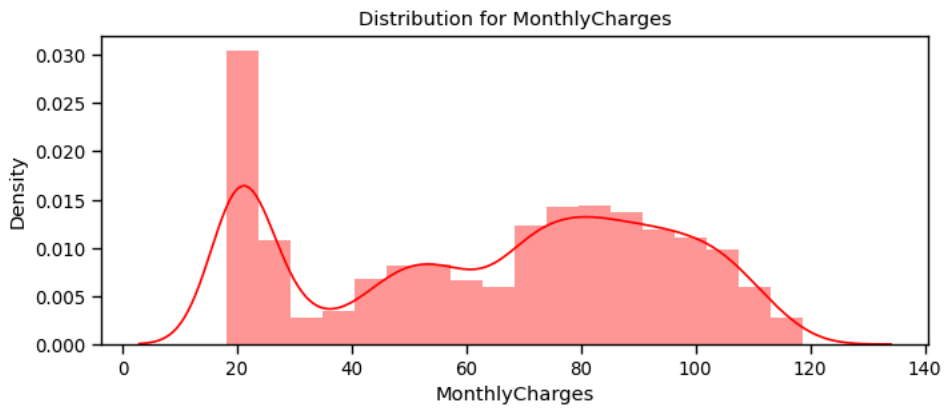
In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40, stratify=y)
```

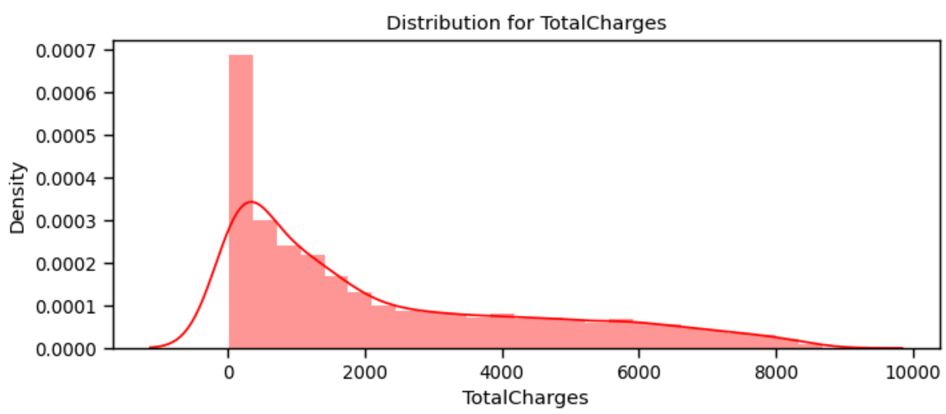
In [52]:
```
print("Shape of x_train is: {0}".format(X_train.shape))
print("Shape of x_test is: {0}".format(X_test.shape))
print("Shape of y_train is: {0}".format(y_train.shape))
print("Shape of y_test is:{0}".format(y_test.shape))
```

```
Shape of x_train is: (5625, 20)
Shape of x_test is: (1407, 20)
Shape of y_train is: (5625,)
Shape of y_test is:(1407,)
```

In [53]:
```
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,3))
    plt.title("Distribution for {}".format(feature))
    ax = sns.distplot(frame[feature], color= color)

num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
for feat in num_cols: distplot(feat, df)
```

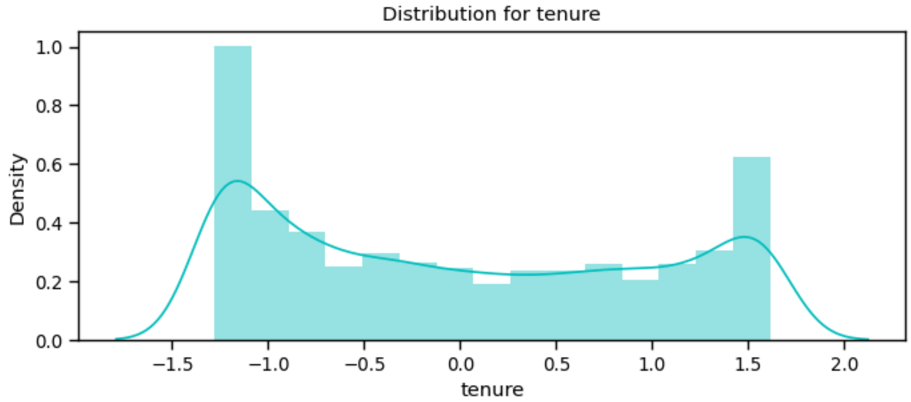


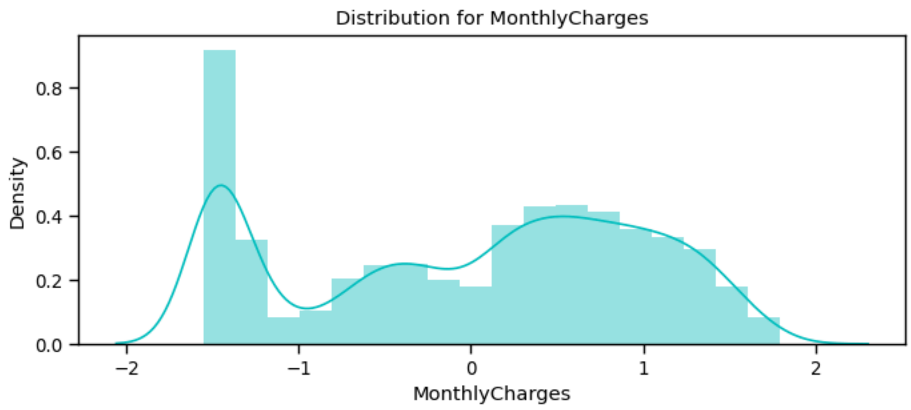


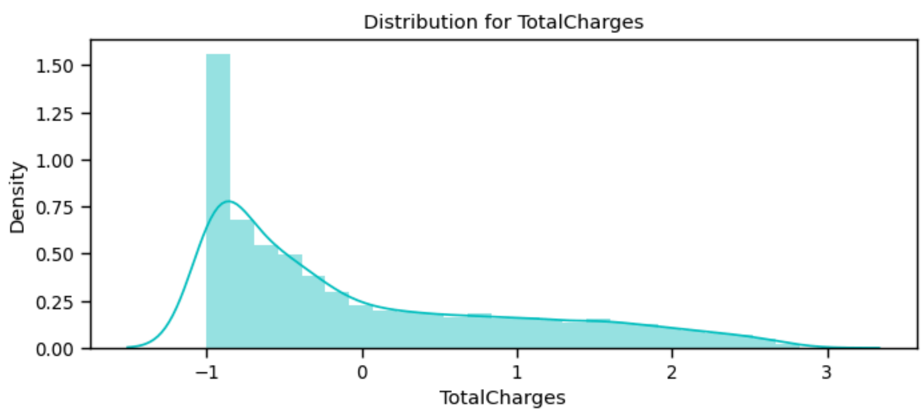


Since the numerical features are distributed over different value ranges, standard scalar is used to scale them down to the same range.

In [54]:
```
df_std = pd.DataFrame(StandardScaler().fit_transform(df[num_cols].astype('float64')),
                      columns = num_cols
                      )
for feat in numerical_cols: distplot(feat, df_std, color='c')   # numerical_cols defined earlier
```







Columns Categorization for different purpose (Standardization, Label & One Hot Encoding)

In [55]:
```
at_cols_ohe = ['PaymentMethod','Contract','InternetService']             # For those one-hot encoding considered to b
at_cols_le = list(set(X_train.columns)- set(num_cols)-set(cat_cols_ohe)) # For those Label encoding considered to be
```

In [56]:
```
scaler= StandardScaler()

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

In [57]:
```
from sklearn.model_selection import cross_val_score

# List of models
models = [
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    MLPClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    ExtraTreesClassifier(),
    LogisticRegression(),
    XGBClassifier(),
    CatBoostClassifier(verbose=0)  # Turn off progress printing for CatBoost
]

# Define the function to evaluate the models using cross-validation
def evaluate_models(X, y, models, cv=5):
    model_results = {}
    for model in models:
        model_name = type(model).__name__
        scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy',)
        model_results[model_name] = scores
        print(f"{model_name}: {scores.mean():.2f}")
    return model_results

# Evaluate all models
model_results = evaluate_models(X, y, models)
```

```
DecisionTreeClassifier: 0.72
RandomForestClassifier: 0.79
MLPClassifier: 0.75
AdaBoostClassifier: 0.80
GradientBoostingClassifier: 0.80
ExtraTreesClassifier: 0.78
LogisticRegression: 0.80
XGBClassifier: 0.78
CatBoostClassifier: 0.79
```

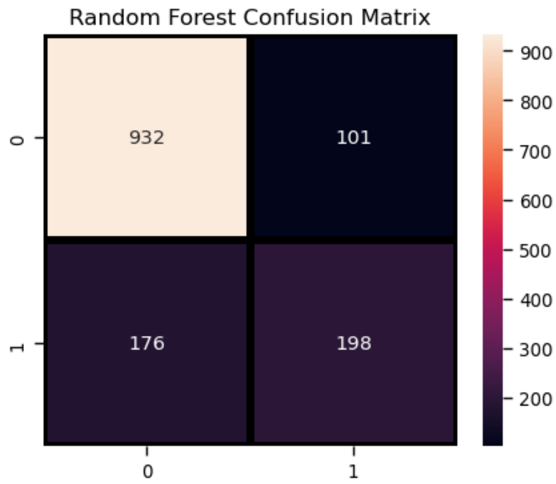
Model Evaluation

1. Random Forest

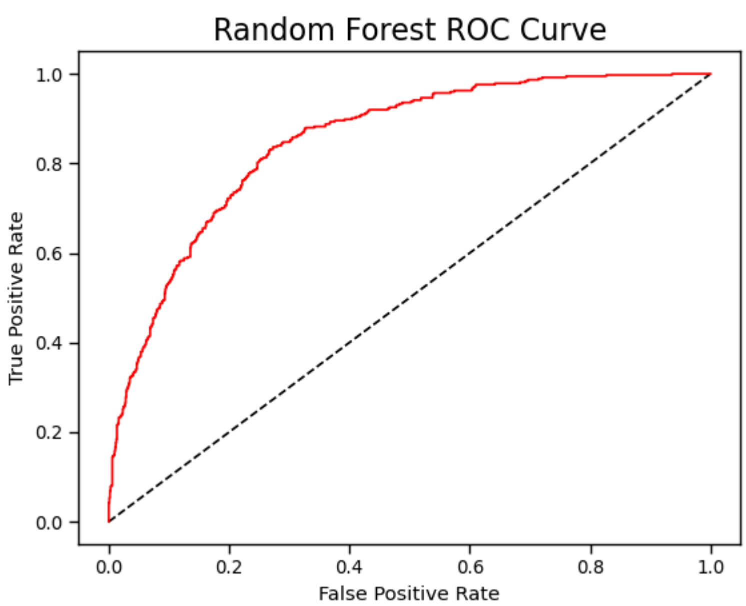
In [58]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Initialize the RandomForestClassifier
random_forest = RandomForestClassifier(random_state=0)

# Define the parameter grid for RandomForestClassifier
param_grid_rf = {
    'n_estimators': [10, 50, 100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Set up the grid search
grid_search_rf = GridSearchCV(random_forest, param_grid_rf, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_rf.fit(X_train, y_train)

# Get the best parameters and model
best_params_rf = grid_search_rf.best_params_
print(f"Best Parameters for Random Forest Model are: {best_params_rf}")
best_rf = RandomForestClassifier(**best_params_rf, random_state=0)
best_rf.fit(X_train, y_train)

# Predictions
y_train_pred_rf = best_rf.predict(X_train)
y_test_pred_rf = best_rf.predict(X_test)

# Evaluation
print("Random Forest Training Accuracy:", round(accuracy_score(y_train, y_train_pred_rf) * 100, 2), "%")
print("Random Forest Testing Accuracy:", round(accuracy_score(y_test, y_test_pred_rf) * 100, 2), "%")
print("Random Forest Model F1 Score:", f1_score(y_test, y_test_pred_rf, average="micro"))
print("Random Forest Model Recall:", recall_score(y_test, y_test_pred_rf, average="micro"))
print("Random Forest Model Precision Score:", precision_score(y_test, y_test_pred_rf, average="micro"))
```

```
Best Parameters for Random Forest Model are: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_
samples_split': 2, 'n_estimators': 50}
Random Forest Training Accuracy: 88.44 %
Random Forest Testing Accuracy: 80.31 %
Random Forest Model F1 Score: 0.8031272210376686
Random Forest Model Recall: 0.8031272210376688
Random Forest Model Precision Score: 0.8031272210376688
```

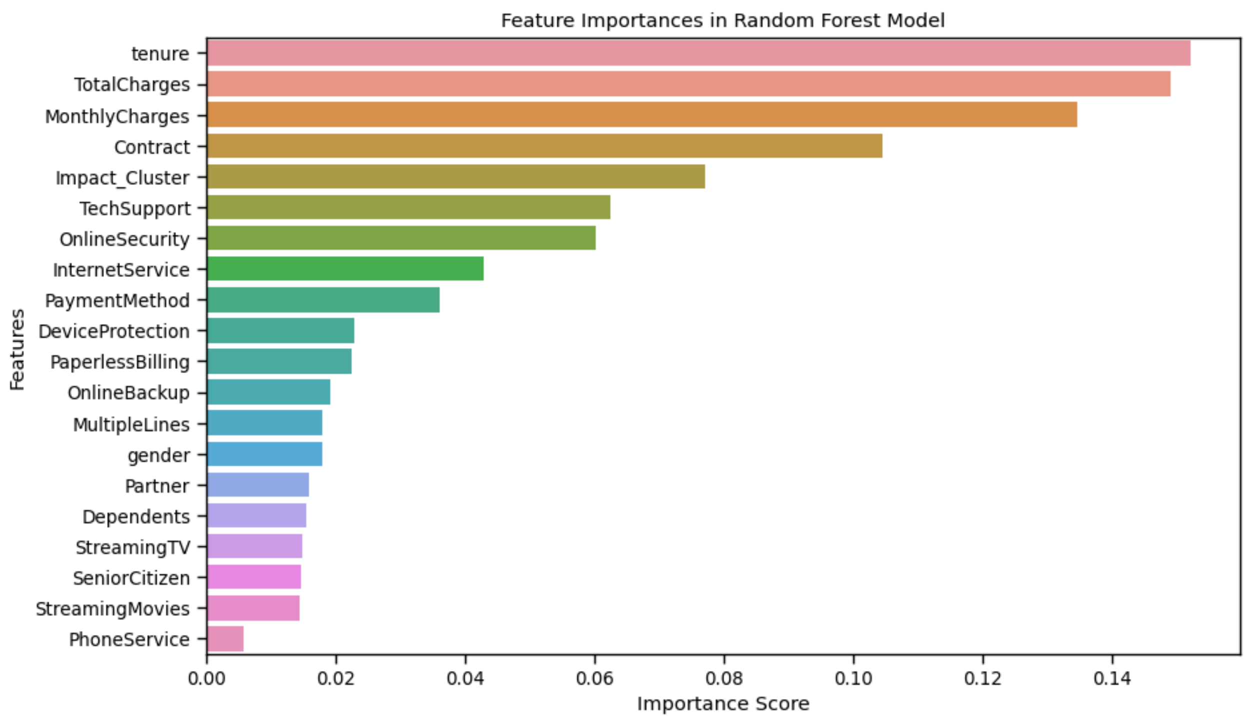
In [59]:
```
# Confusion Matrix
plt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_test, y_test_pred_rf),
           annot=True, fmt="d", linecolor="k", linewidths=3
           )
plt.title("Random Forest Confusion Matrix", fontsize=12)
plt.show()
```



In [60]:
```
y_rfpred_prob = best_rf.predict_proba(X_test)[:,1]
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_rfpred_prob)
plt.plot([0, 1], [0, 1], 'k--' )
plt.plot(fpr_rf, tpr_rf, label='Random Forest',color = "r")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve',fontsize=16)
plt.show();
```



In [61]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assume 'best_rf' is your trained Random Forest model from the GridSearchCV
feature_importances = best_rf.feature_importances_

# 'X_train' should be your feature DataFrame, ensure it is used in the fitting process
feature_names = X_train.columns  # Ensure X_train has column names

# Create a DataFrame to hold feature names and their importance scores
importances = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
})

# Sort the features by importance
importances = importances.sort_values(by='Importance', ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importances)
plt.title('Feature Importances in Random Forest Model')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



In [62]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Initialize the LogisticRegression
logreg = LogisticRegression(random_state=0)

# Define the parameter grid for LogisticRegression
param_grid_logreg = {
    'C': np.logspace(-4, 4, 20),  # Regularization strength
    'penalty': ['l2'],            # Norm used in penalization
    'solver': ['lbfgs']           # Algorithm to use in optimization
}

# Set up the grid search
grid_search_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_logreg.fit(X_train, y_train)  # Ensure your data is appropriately defined as X_train and y_train
```

Out[62]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [63]:
```python
# Get the best parameters and instantiate a new model
best_params_logreg = grid_search_logreg.best_params_
print(f"Best Parameters for Logistic Regression Model are: {best_params_logreg}")
best_logreg = LogisticRegression(**best_params_logreg, random_state=0)
best_logreg.fit(X_train, y_train)
```

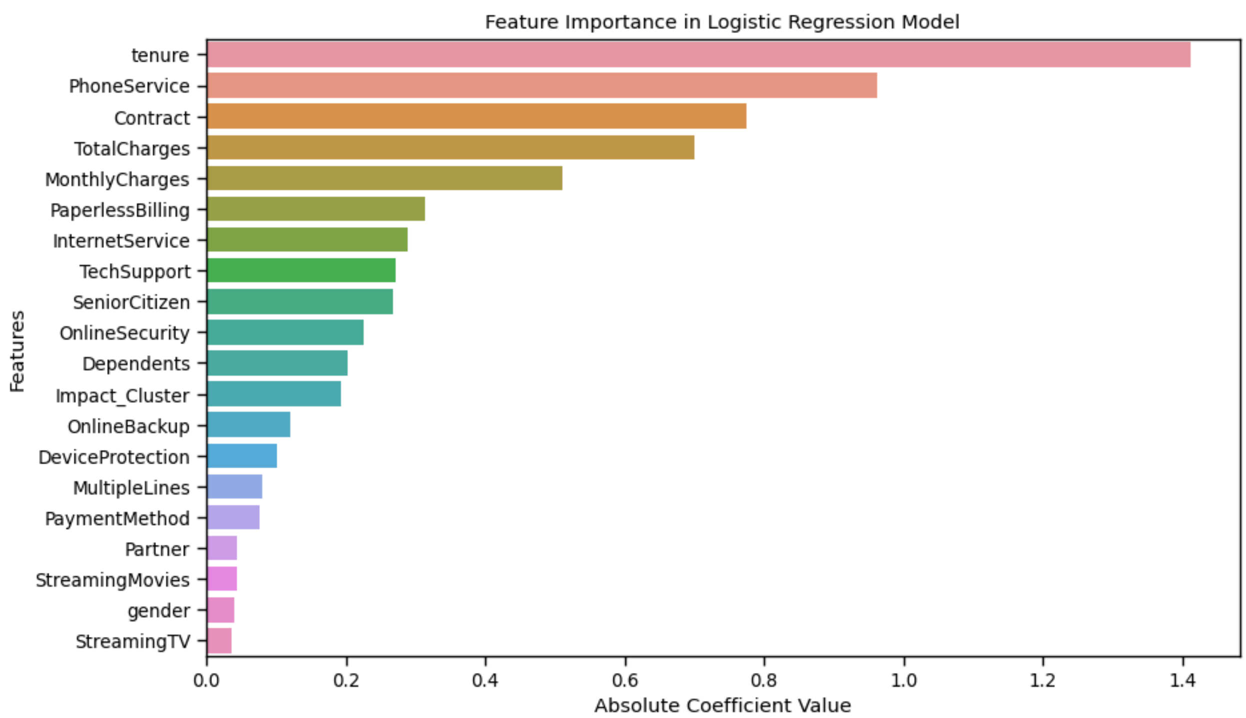
```
Best Parameters for Logistic Regression Model are: {'C': 29.763514416313132, 'penalty': 'l2', 'solver': 'lbfgs'}
```

Out[63]:
```
▾ LogisticRegression
LogisticRegression(C=29.763514416313132, random_state=0)
```

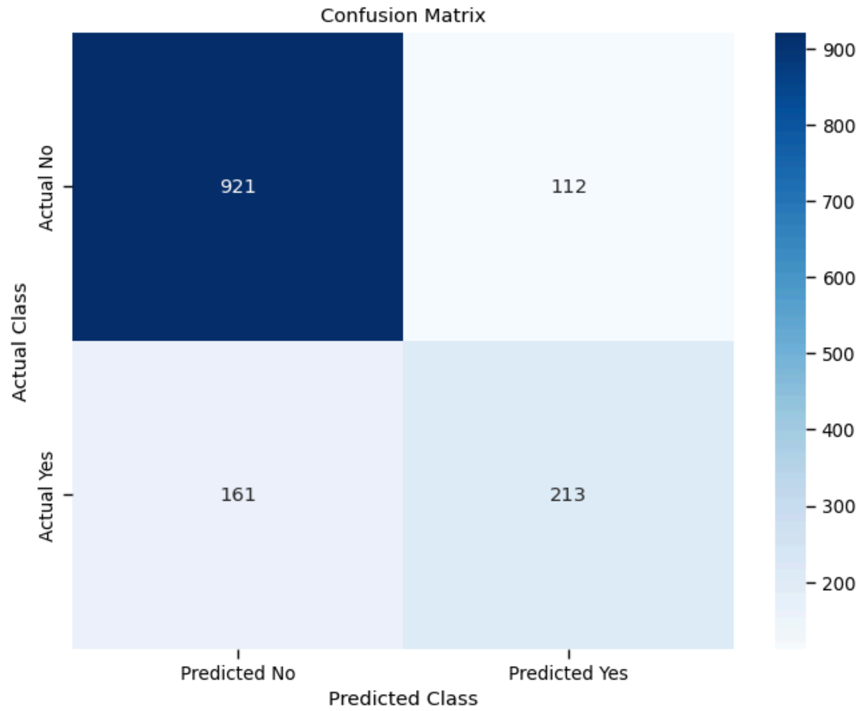
In [64]:
```
# Predictions
y_train_pred = best_logreg.predict(X_train)
y_test_pred = best_logreg.predict(X_test)  # Ensure X_test is defined

# Evaluation
print("Logistic Regression Training Accuracy:", round(accuracy_score(y_train, y_train_pred) * 100, 2), "%")
print("Logistic Regression Testing Accuracy:", round(accuracy_score(y_test, y_test_pred) * 100, 2), "%")
print("Logistic Regression Model F1 Score:", f1_score(y_test, y_test_pred, average="micro"))
print("Logistic Regression Model Recall:", recall_score(y_test, y_test_pred, average="micro"))
print("Logistic Regression Model Precision Score:", precision_score(y_test, y_test_pred, average="micro"))
```

```
Logistic Regression Training Accuracy: 80.39 %
Logistic Regression Testing Accuracy: 80.6 %
Logistic Regression Model F1 Score: 0.8059701492537313
Logistic Regression Model Recall: 0.8059701492537313
Logistic Regression Model Precision Score: 0.8059701492537313
```

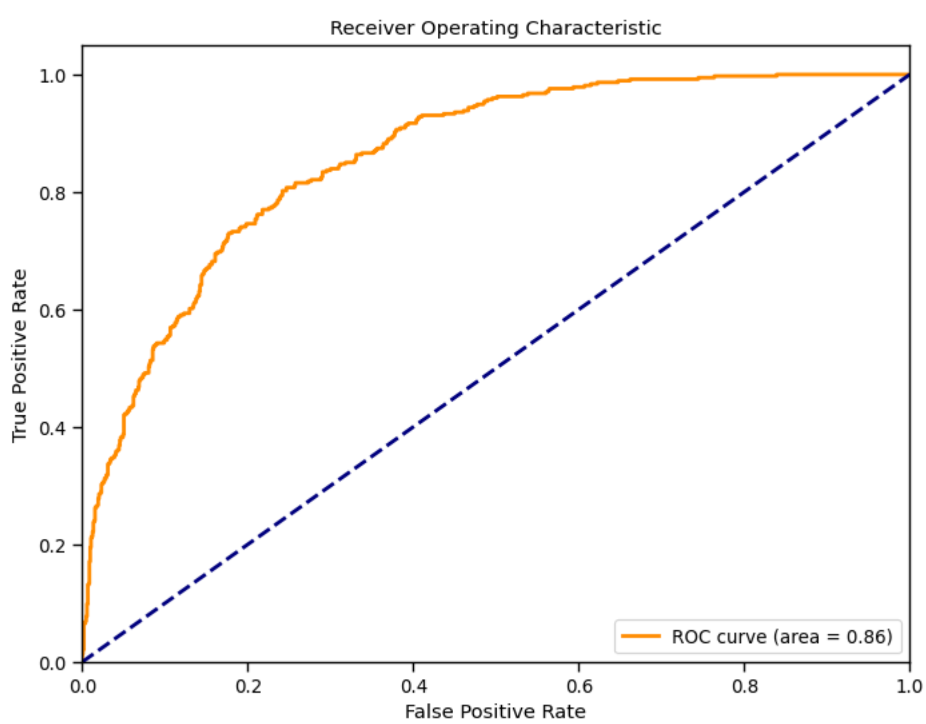
In [65]:
```
# Feature Importance from Coefficients
coefficients = best_logreg.coef_[0]  # Extract coefficients
features = X_train.columns  # Assuming your dataframe has column names

# Creating DataFrame for feature importance
feature_importance = pd.DataFrame(features, columns=["Feature"])
feature_importance['Importance'] = np.abs(coefficients)
feature_importance = feature_importance.sort_values(by="Importance", ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance)
plt.title('Feature Importance in Logistic Regression Model')
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Features')
plt.show()
```



In [66]:
```
m sklearn.metrics import confusion_matrix
port seaborn as sns
port matplotlib.pyplot as plt

ssuming y_test and y_test_pred are already defined from your previous model predictions
alculate the confusion matrix
f_matrix = confusion_matrix(y_test, y_test_pred)

isualizing the confusion matrix using Seaborn's heatmap
.figure(figsize=(8, 6))
.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Predicted No", "Predicted Yes"], yticklabels=
.title('Confusion Matrix')
.ylabel('Actual Class')
.xlabel('Predicted Class')
.show()
```



In [67]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Ensure your model is fitted and you have the test set ready
# best_logreg is assumed to be the fitted Logistic Regression model
y_test_prob = best_logreg.predict_proba(X_test)[:, 1]  # Get probabilities for the positive class

# Calculate FPR (False Positive Rate), TPR (True Positive Rate), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_test_prob)

# Calculate the AUC (Area Under Curve)
roc_auc = auc(fpr, tpr)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



3. Decision Tree Classifier

In [68]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Initialize the DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(random_state=0)

# Define the parameter grid for DecisionTreeClassifier
param_grid_dt = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Set up the grid search
grid_search_dt = GridSearchCV(decision_tree, param_grid_dt, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_dt.fit(X_train, y_train)  # Make sure your X_train and y_train are properly defined
```

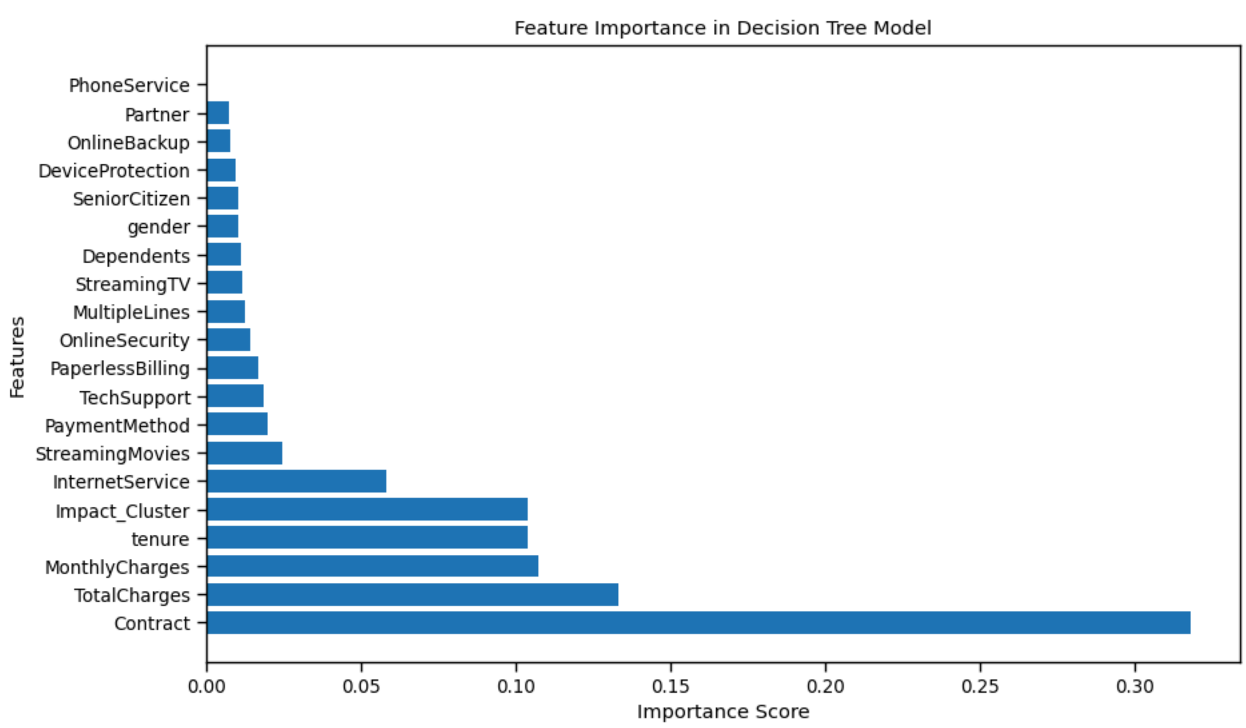
Out[68]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [69]:
```
# Get the best parameters and instantiate a new model with these parameters
best_params_dt = grid_search_dt.best_params_
print(f"Best Parameters for Decision Tree Model are: {best_params_dt}")
best_dt = DecisionTreeClassifier(**best_params_dt, random_state=0)
best_dt.fit(X_train, y_train)
```

```
Best Parameters for Decision Tree Model are: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
```

Out[69]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, min_samples_leaf=4, min_samples_split=10,
                       random_state=0)
```

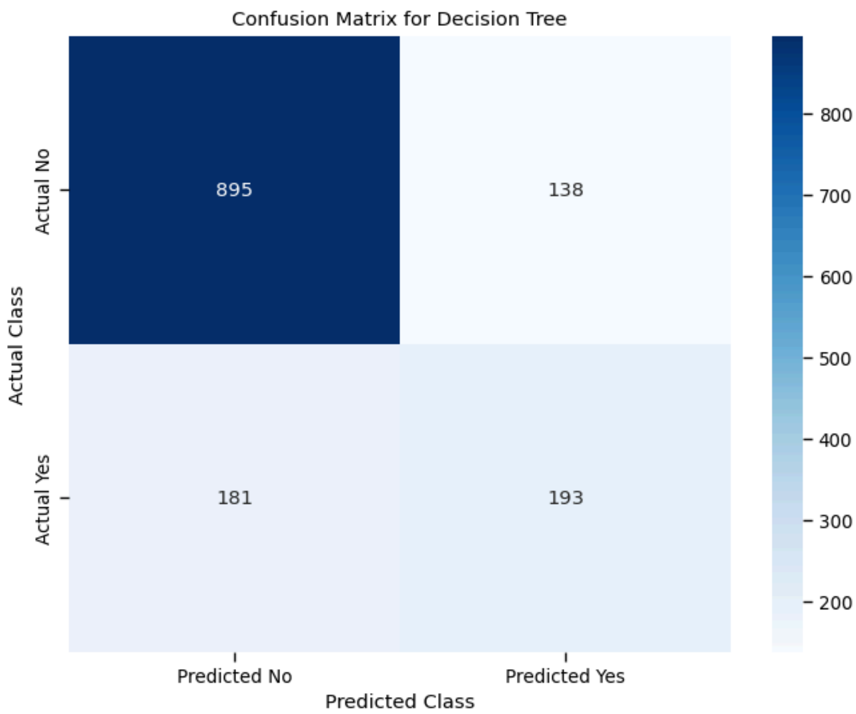
In [70]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Extracting feature importances
feature_importances_dt = best_dt.feature_importances_
features = X_train.columns

# Creating DataFrame for feature importance
feature_importance_dt = pd.DataFrame({'Feature': features, 'Importance': feature_importances_dt})
feature_importance_dt = feature_importance_dt.sort_values(by="Importance", ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_dt['Feature'], feature_importance_dt['Importance'])
plt.title('Feature Importance in Decision Tree Model')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



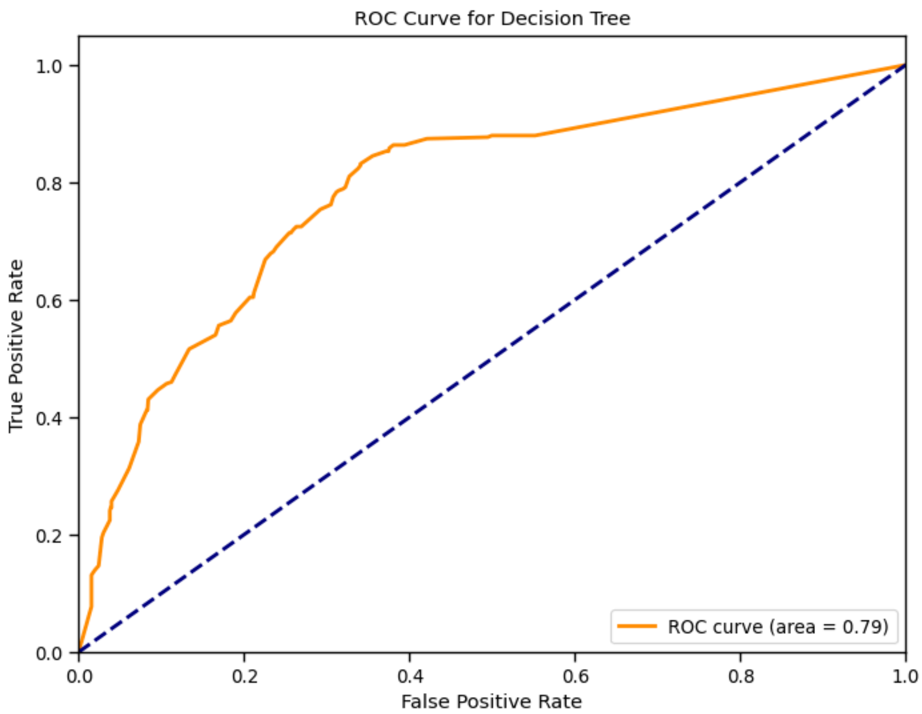
In [71]:
```
# Predictions for training and testing sets
y_train_pred_dt = best_dt.predict(X_train)
y_test_pred_dt = best_dt.predict(X_test)  # Ensure X_test is defined

# Evaluation for Decision Tree
print("Decision Tree Training Accuracy:", round(accuracy_score(y_train, y_train_pred_dt) * 100, 2), "%")
print("Decision Tree Testing Accuracy:", round(accuracy_score(y_test, y_test_pred_dt) * 100, 2), "%")
print("Decision Tree Model F1 Score:", f1_score(y_test, y_test_pred_dt, average="micro"))
print("Decision Tree Model Recall:", recall_score(y_test, y_test_pred_dt, average="micro"))
print("Decision Tree Model Precision Score:", precision_score(y_test, y_test_pred_dt, average="micro"))
```

```
Decision Tree Training Accuracy: 84.69 %
Decision Tree Testing Accuracy: 77.33 %
Decision Tree Model F1 Score: 0.7732764747690121
Decision Tree Model Recall: 0.7732764747690121
Decision Tree Model Precision Score: 0.7732764747690121
```

In [72]:
```
# Calculate the confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_test_pred_dt)

# Visualizing the confusion matrix using Seaborn's heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, fmt="d", cmap="Blues", xticklabels=["Predicted No", "Predicted Yes"], ytickl
plt.title('Confusion Matrix for Decision Tree')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()
```



In [73]:
```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Ensure your model supports probability estimates
y_test_prob_dt = best_dt.predict_proba(X_test)[:, 1]  # Probabilities for the positive class

# Calculate FPR, TPR, and thresholds
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_test_prob_dt)

# Calculate the AUC
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_dt)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree')
plt.legend(loc="lower right")
plt.show()
```



4. Adaboost Classifier

In [74]:
```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Initialize the AdaBoostClassifier with a base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)  # Commonly a shallow tree
adaboost = AdaBoostClassifier(base_estimator=base_estimator, random_state=0)

# Define the parameter grid for AdaBoostClassifier
param_grid_ada = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.1, 1]
}

# Set up the grid search
grid_search_ada = GridSearchCV(adaboost, param_grid_ada, cv=5, scoring='accuracy', n_jobs=-1)
grid_search_ada.fit(X_train, y_train)  # Ensure X_train and y_train are defined
```

```
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/Users/harithaanand/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_esti
mator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

Out[74]:
```
▸ GridSearchCV
▸ estimator: AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
▸ DecisionTreeClassifier
```

In [75]:
```
# Get the best parameters and instantiate a new model
best_params_ada = grid_search_ada.best_params_
print(f"Best Parameters for AdaBoost Model are: {best_params_ada}")
best_ada = AdaBoostClassifier(**best_params_ada, random_state=0)
best_ada.fit(X_train, y_train)
```

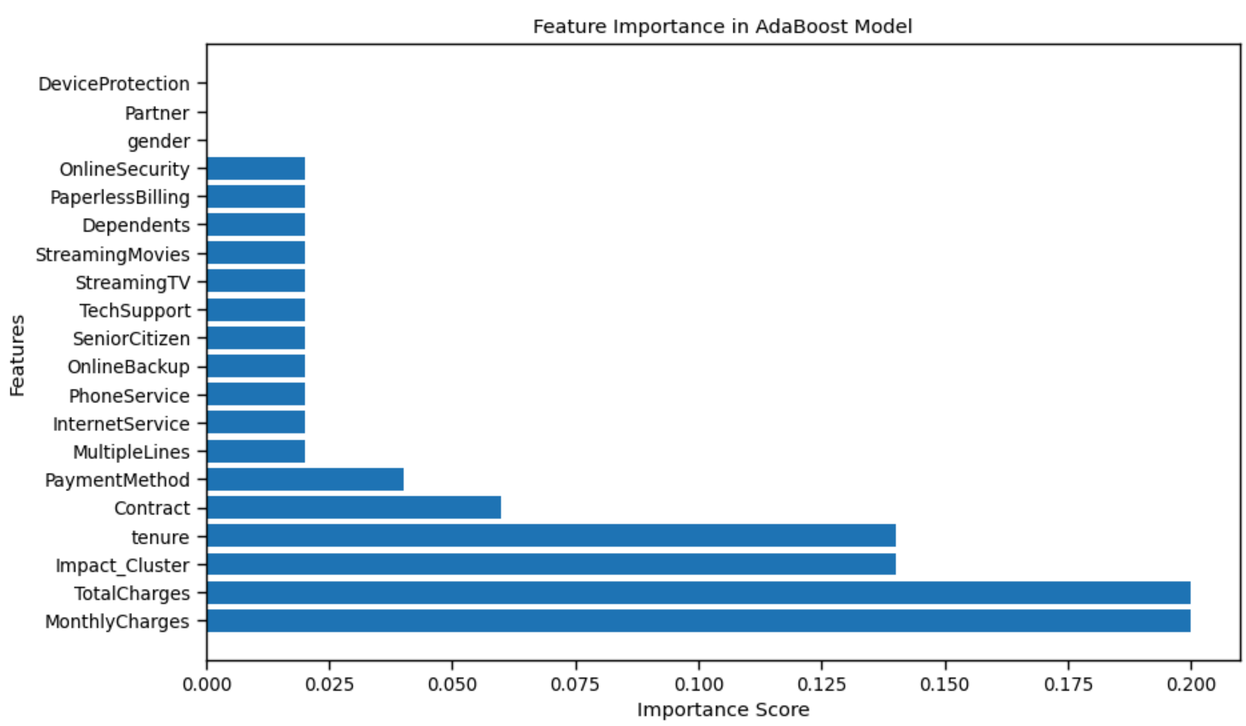
```
Best Parameters for AdaBoost Model are: {'learning_rate': 1, 'n_estimators': 50}
```

Out[75]:
```
AdaBoostClassifier
AdaBoostClassifier(learning_rate=1, random_state=0)
```

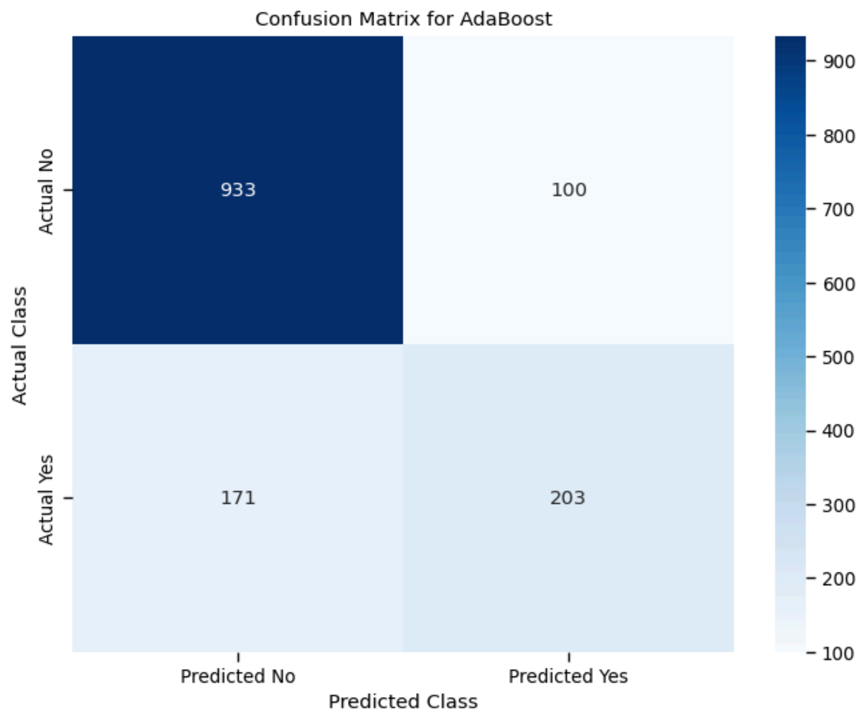
In [76]:
```
# Predictions for training and testing sets
y_train_pred_ada = best_ada.predict(X_train)
y_test_pred_ada = best_ada.predict(X_test)  # Ensure X_test is defined

# Evaluation for AdaBoost
print("AdaBoost Training Accuracy:", round(accuracy_score(y_train, y_train_pred_ada) * 100, 2), "%")
print("AdaBoost Testing Accuracy:", round(accuracy_score(y_test, y_test_pred_ada) * 100, 2), "%")
print("AdaBoost Model F1 Score:", f1_score(y_test, y_test_pred_ada, average="micro"))
print("AdaBoost Model Recall:", recall_score(y_test, y_test_pred_ada, average="micro"))
print("AdaBoost Model Precision Score:", precision_score(y_test, y_test_pred_ada, average="micro"))
```

```
AdaBoost Training Accuracy: 80.64 %
AdaBoost Testing Accuracy: 80.74 %
AdaBoost Model F1 Score: 0.8073916133617627
AdaBoost Model Recall: 0.8073916133617626
AdaBoost Model Precision Score: 0.8073916133617626
```

In [77]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Extracting feature importances (only works if the base estimator supports this, e.g., trees)
feature_importances_ada = best_ada.feature_importances_
features = X_train.columns

# Creating DataFrame for feature importance
feature_importance_ada = pd.DataFrame({'Feature': features, 'Importance': feature_importances_ada})
feature_importance_ada = feature_importance_ada.sort_values(by="Importance", ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_ada['Feature'], feature_importance_ada['Importance'])
plt.title('Feature Importance in AdaBoost Model')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```

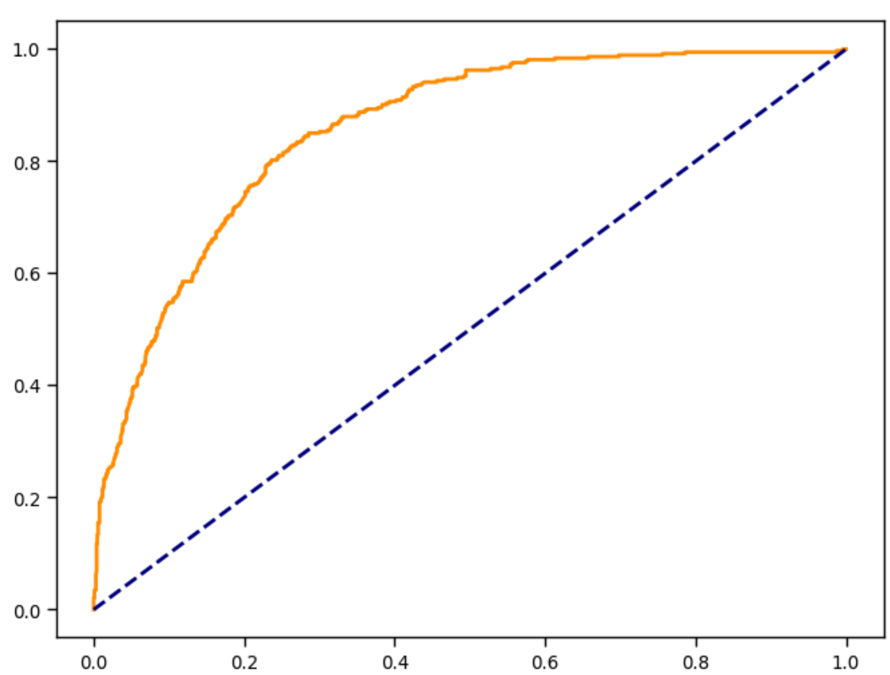


In [78]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Calculate the confusion matrix
conf_matrix_ada = confusion_matrix(y_test, y_test_pred_ada)

# Visualizing the confusion matrix using Seaborn's heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_ada, annot=True, fmt="d", cmap="Blues", xticklabels=["Predicted No", "Predicted Yes"], ytick
plt.title('Confusion Matrix for AdaBoost')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()
```

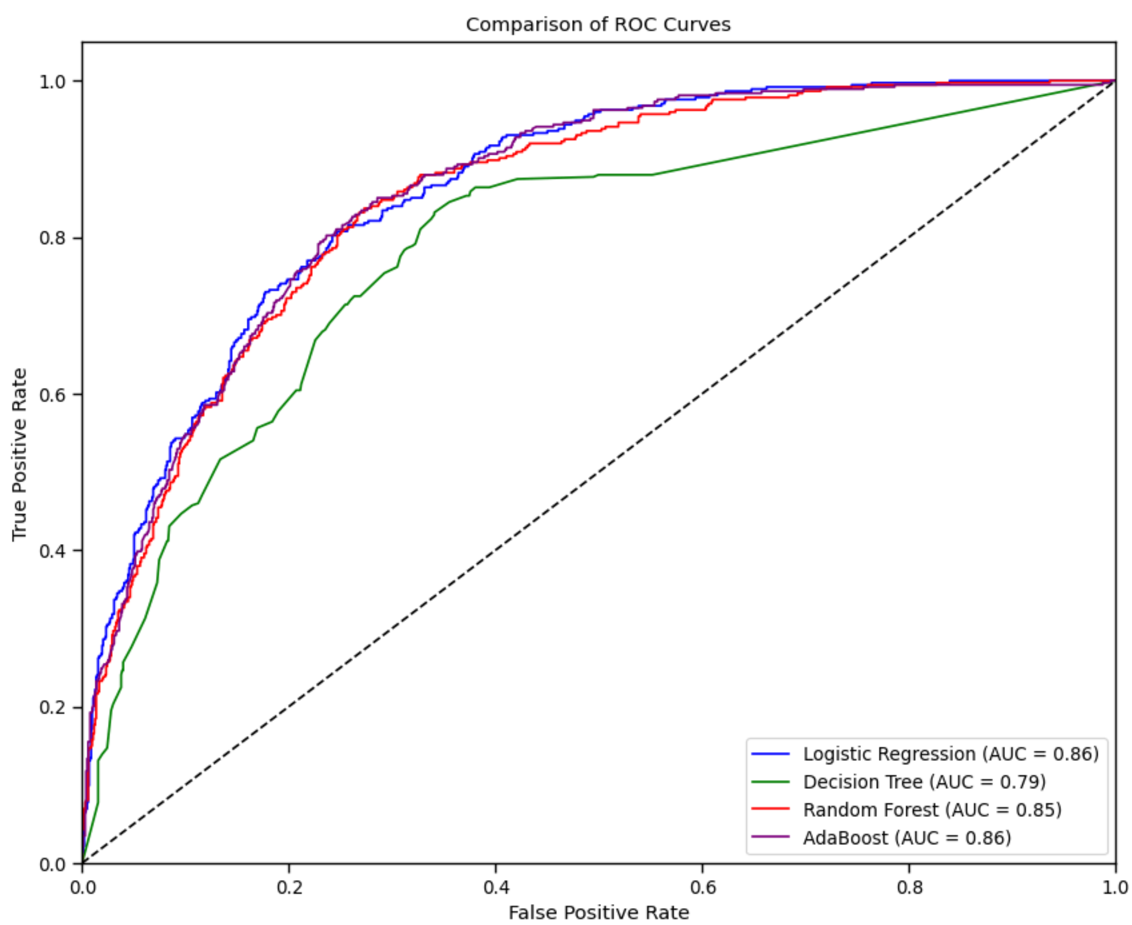


In [79]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for the positive class
y_test_prob_ada = best_ada.predict_proba(X_test)[:, 1]

# Calculate FPR, TPR, and thresholds
fpr_ada, tpr_ada, thresholds_ada = roc_curve(y_test, y_test_prob_ada)

# Calculate the AUC
roc_auc_ada = auc(fpr_ada, tpr_ada)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(fpr_ada, tpr_ada, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_ada)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```

Out[79]: [<matplotlib.lines.Line2D at 0x15f706c10>]



In [80]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Ensure these models have been fitted and you have the test set ready:
# best_logreg, best_dt, best_rf, best_ada

# Predict probabilities for the positive class for each model
y_test_prob_logreg = best_logreg.predict_proba(X_test)[:, 1]
y_test_prob_dt = best_dt.predict_proba(X_test)[:, 1]
y_test_prob_rf = best_rf.predict_proba(X_test)[:, 1]
y_test_prob_ada = best_ada.predict_proba(X_test)[:, 1]

# Calculate FPR, TPR, and thresholds for each model
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, y_test_prob_logreg)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_test_prob_dt)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_test_prob_rf)
fpr_ada, tpr_ada, _ = roc_curve(y_test, y_test_prob_ada)

# Calculate the AUC for each model
roc_auc_logreg = auc(fpr_logreg, tpr_logreg)
roc_auc_dt = auc(fpr_dt, tpr_dt)
roc_auc_rf = auc(fpr_rf, tpr_rf)
roc_auc_ada = auc(fpr_ada, tpr_ada)

# Plotting all ROC curves
plt.figure(figsize=(10, 8))
plt.plot(fpr_logreg, tpr_logreg, label=f'Logistic Regression (AUC = {roc_auc_logreg:.2f})', color='blue')
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {roc_auc_dt:.2f})', color='green')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})', color='red')
plt.plot(fpr_ada, tpr_ada, label=f'AdaBoost (AUC = {roc_auc_ada:.2f})', color='purple')

plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparison of ROC Curves')
plt.legend(loc="lower right")
plt.show()
```



In [ ]: