Introduction

Customer churn, the phenomenon where customers stop using a company's products or services, is a significant challenge for many businesses. Understanding and predicting churn can help companies take proactive steps to retain customers, thereby reducing loss of revenue and improving long-term profitability. This project aims to predict customer churn by analyzing behavioral and demographic data using machine learning techniques. By leveraging a comprehensive dataset, the project identifies key factors contributing to churn and develops models to accurately predict which customers are at risk of leaving.

Data Source

The primary dataset was sourced from a telecommunications company, encompassing various customer attributes and their churn status. The dataset includes 7,043 rows and 21 columns, with a binary target variable indicating whether the customer has churned.

Data Source: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

Necessary Libraries

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

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
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 sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classifica
```

In [3]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
```

Data Exploration

In [4]:
```
df = pd.read_csv("Telco-Customer-Churn.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	De
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 [6]:

```
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 [7]:
```
df.describe()
```

Out[7]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
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 [9]:
```
df.shape
```
Out[9]:
```
(7043, 21)
```
In [10]:
```
# Iterate over all columns in the DataFrame
for column in df.columns:
    print(f"Value counts for column: {column}")
    print(df[column].value_counts())
    print("\n" + "-"*50 + "\n")
```
```
Value counts for column: customerID
customerID
7590-VHVEG    1
3791-LGQCY    1
6008-NAIXK    1
5956-YHHRX    1
5365-LLFYV    1
             ..
9796-MVYXX    1
2637-FKFSY    1
1552-AAGRX    1
4304-TSPVK    1
3186-AJIEK    1
Name: count, Length: 7043, dtype: int64

--------------------------------------------------

Value counts for column: gender
gender
Male      3555
Female    3488
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: SeniorCitizen
SeniorCitizen
0    5901
1    1142
Name: count, dtype: int64
```

```
--------------------------------------------------

Value counts for column: Partner
Partner
No     3641
Yes    3402
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: Dependents
Dependents
No     4933
Yes    2110
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: tenure
tenure
1     613
72    362
2     238
3     200
4     176
     ...
28     57
39     56
44     51
36     50
0      11
Name: count, Length: 73, dtype: int64

--------------------------------------------------

Value counts for column: PhoneService
PhoneService
Yes    6361
No      682
Name: count, dtype: int64
```

```
--------------------------------------------------

Value counts for column: MultipleLines
MultipleLines
No                  3390
Yes                 2971
No phone service     682
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: InternetService
InternetService
Fiber optic    3096
DSL            2421
No             1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: OnlineSecurity
OnlineSecurity
No                     3498
Yes                    2019
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: OnlineBackup
OnlineBackup
No                     3088
Yes                    2429
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: DeviceProtection
DeviceProtection
```

```
No                     3095
Yes                    2422
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: TechSupport
TechSupport
No                     3473
Yes                    2044
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: StreamingTV
StreamingTV
No                     2810
Yes                    2707
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: StreamingMovies
StreamingMovies
No                     2785
Yes                    2732
No internet service    1526
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: Contract
Contract
Month-to-month    3875
Two year          1695
One year          1473
Name: count, dtype: int64
```

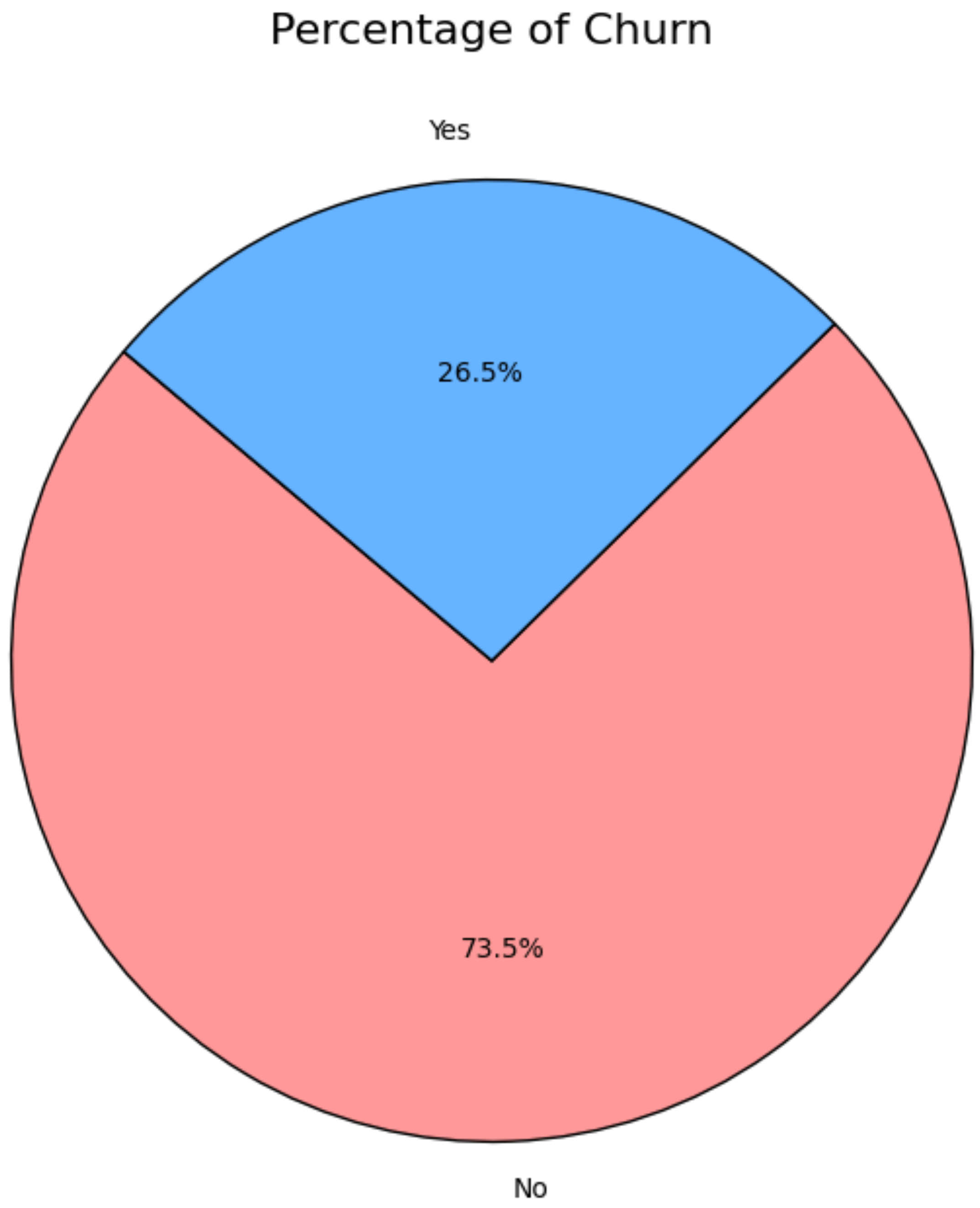
```
--------------------------------------------------

Value counts for column: PaperlessBilling
PaperlessBilling
Yes    4171
No     2872
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: PaymentMethod
PaymentMethod
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: count, dtype: int64

--------------------------------------------------

Value counts for column: MonthlyCharges
MonthlyCharges
20.05     61
19.85     45
19.95     44
19.90     44
20.00     43
          ..
23.65      1
114.70     1
43.65      1
87.80      1
78.70      1
Name: count, Length: 1585, dtype: int64

--------------------------------------------------

Value counts for column: TotalCharges
TotalCharges
          11
20.2      11
```

```
19.75       9
20.05       8
19.9        8
           ..
6849.4      1
692.35      1
130.15      1
3211.9      1
6844.5      1
Name: count, Length: 6531, dtype: int64

--------------------------------------------------

Value counts for column: Churn
Churn
No     5174
Yes    1869
Name: count, dtype: int64

--------------------------------------------------
```

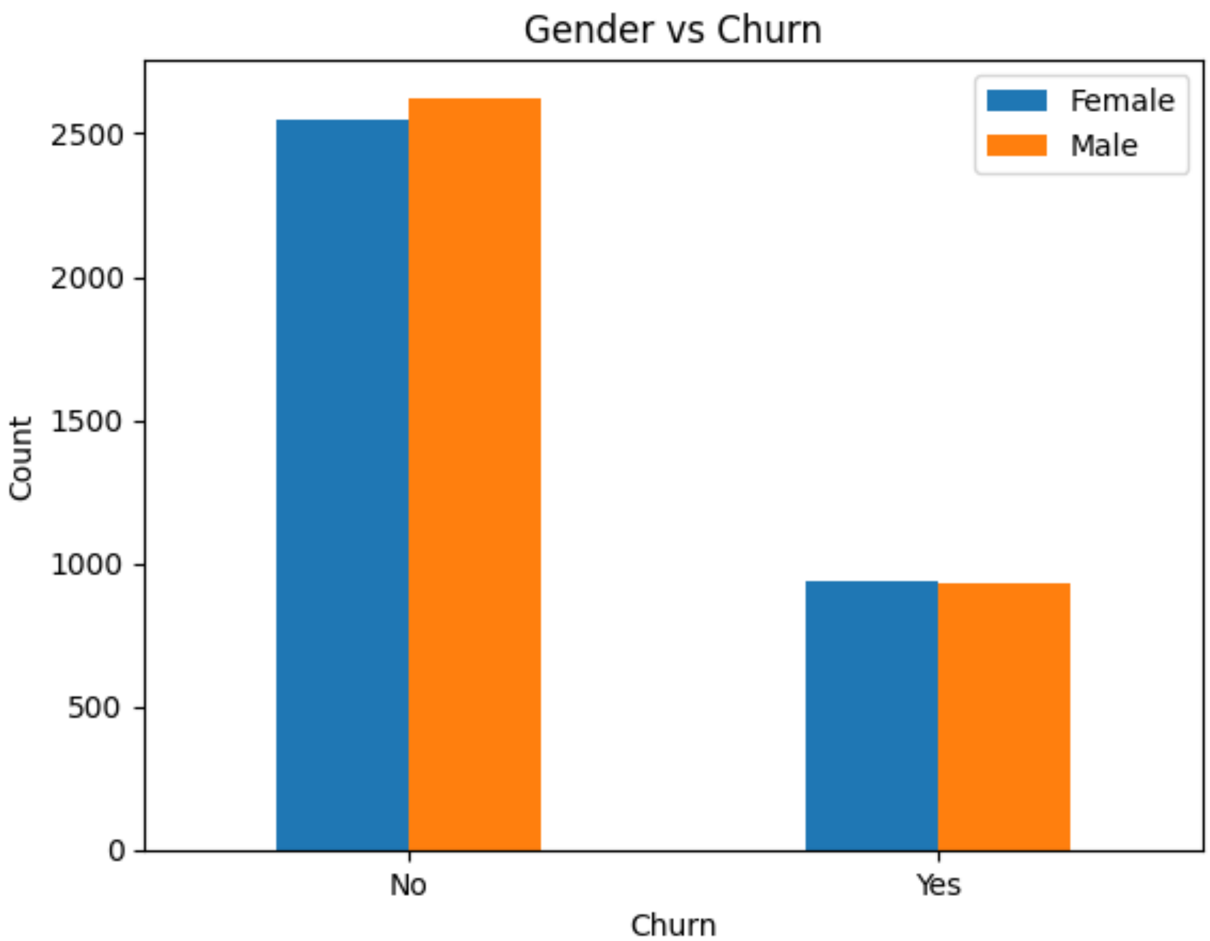
Data Visualization

In [11]:

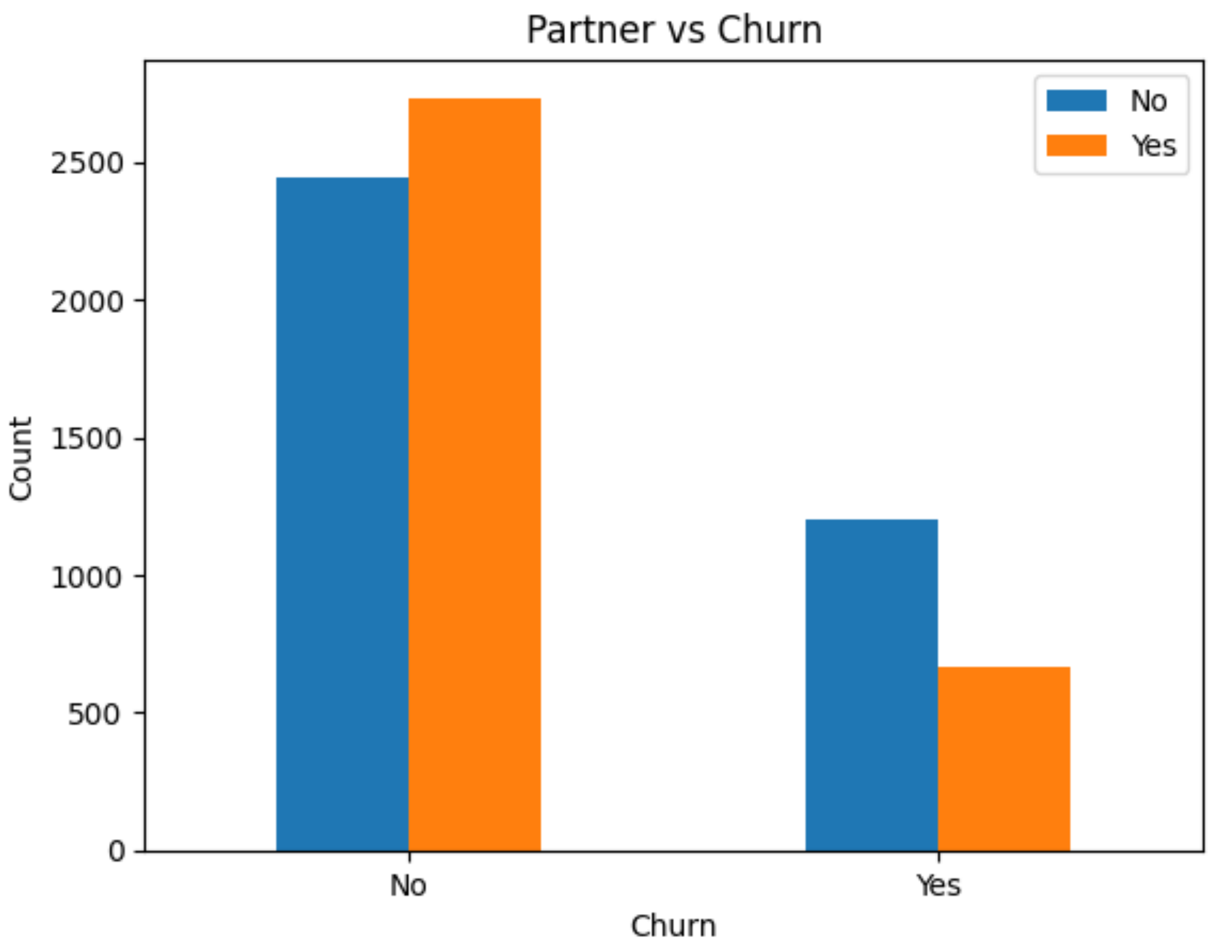
```
# Data preparation
churn_counts = df['Churn'].value_counts()
labels = churn_counts.index
sizes = churn_counts.values
colors = ['#ff9999','#66b3ff']

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140, wedgeprops={'edgecolor': 'black'}
plt.title('Percentage of Churn', fontsize=16)
plt.show()
```


Percentage of Churn
Yes
26.5%
73.5%
No

In [12]:
```
pd.crosstab(df['Churn'], df['gender']).plot(kind='bar')
plt.title('Gender vs Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.legend(['Female', 'Male'])
plt.xticks(rotation=0)
plt.show()
```

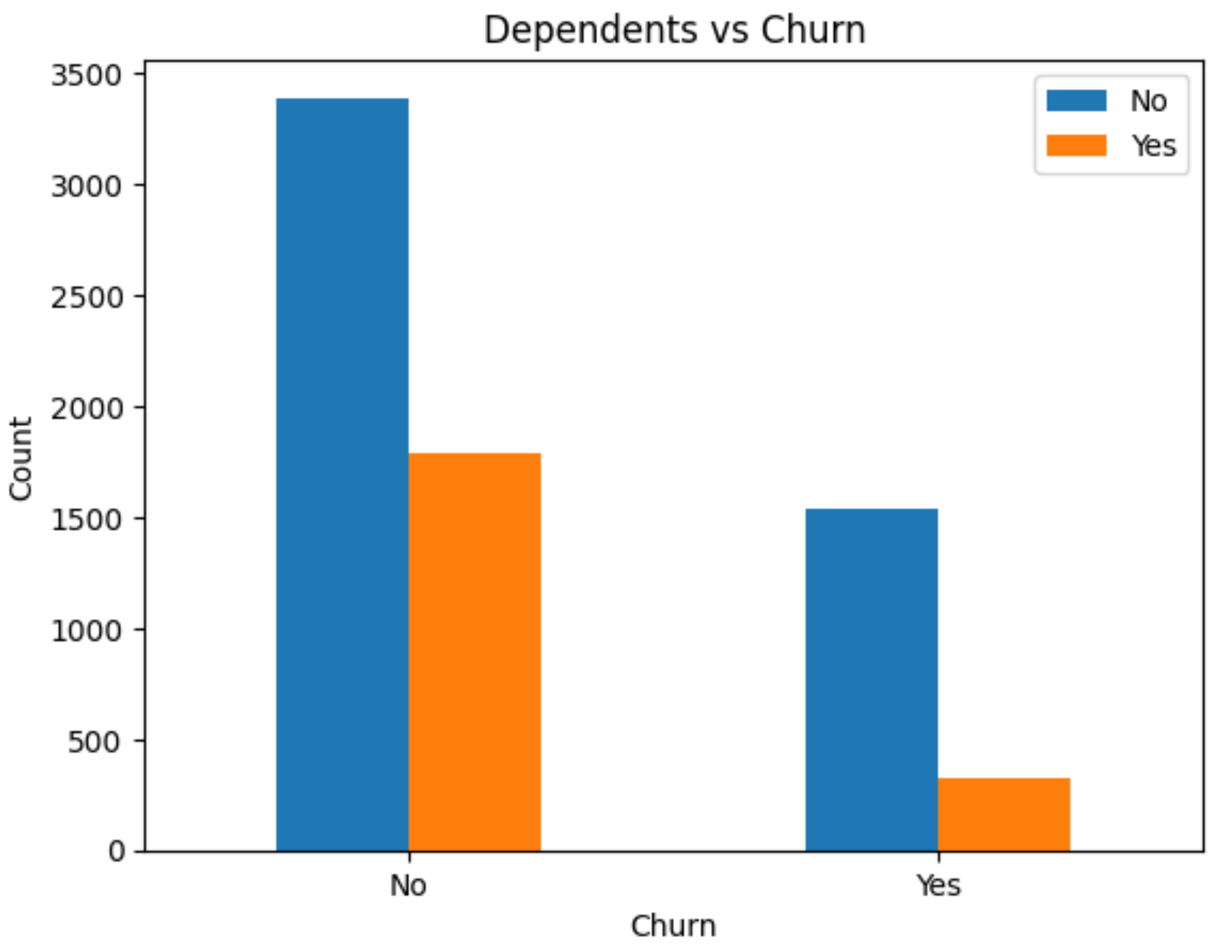


In [13]:
```
pd.crosstab(df['Churn'], df['Partner']).plot(kind='bar')
plt.title('Partner vs Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.legend(['No', 'Yes'])
plt.xticks(rotation=0)
plt.show()
```

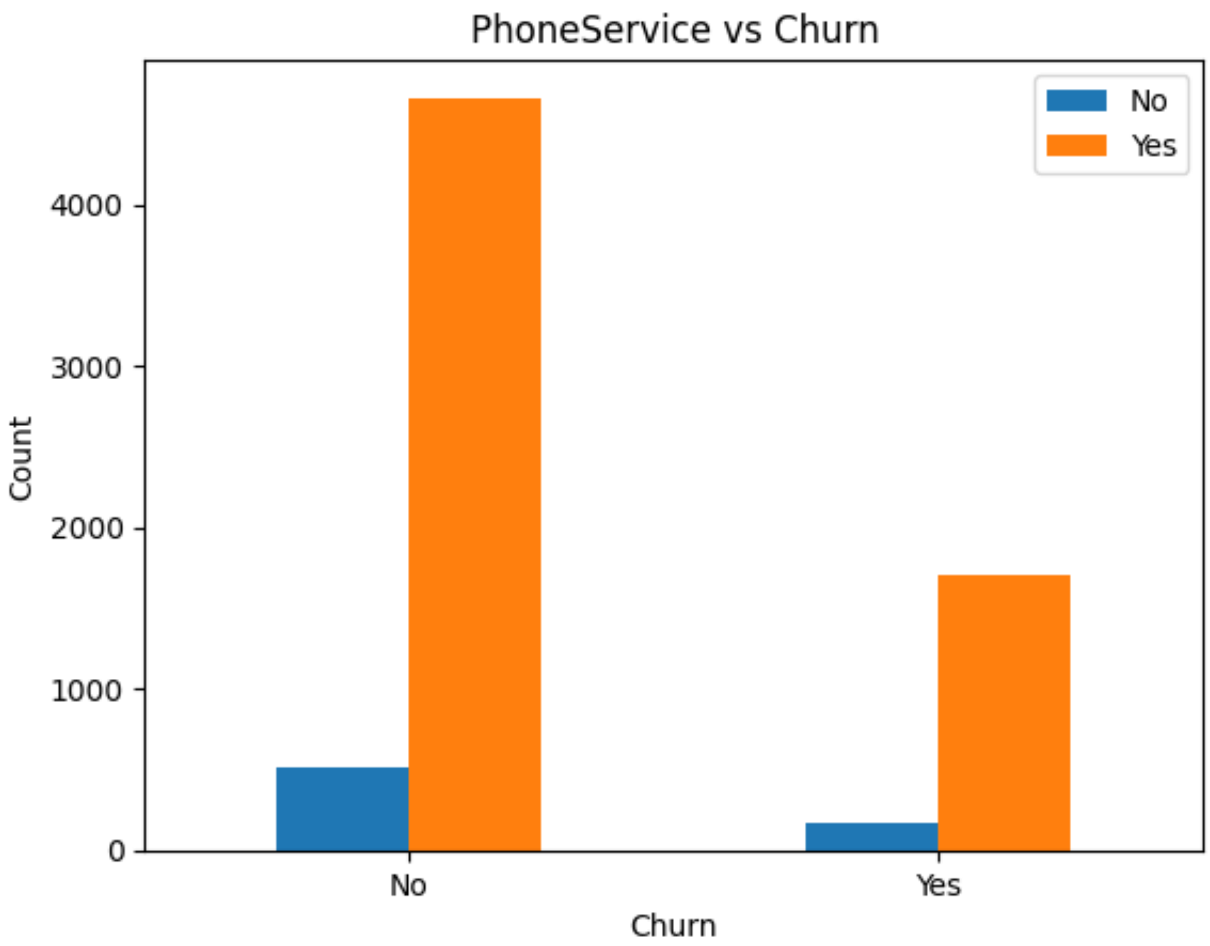


In [14]:

```
pd.crosstab(df['Churn'], df['Dependents']).plot(kind='bar')
plt.title('Dependents vs Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.legend(['No', 'Yes'])
plt.xticks(rotation=0)
plt.show()
```

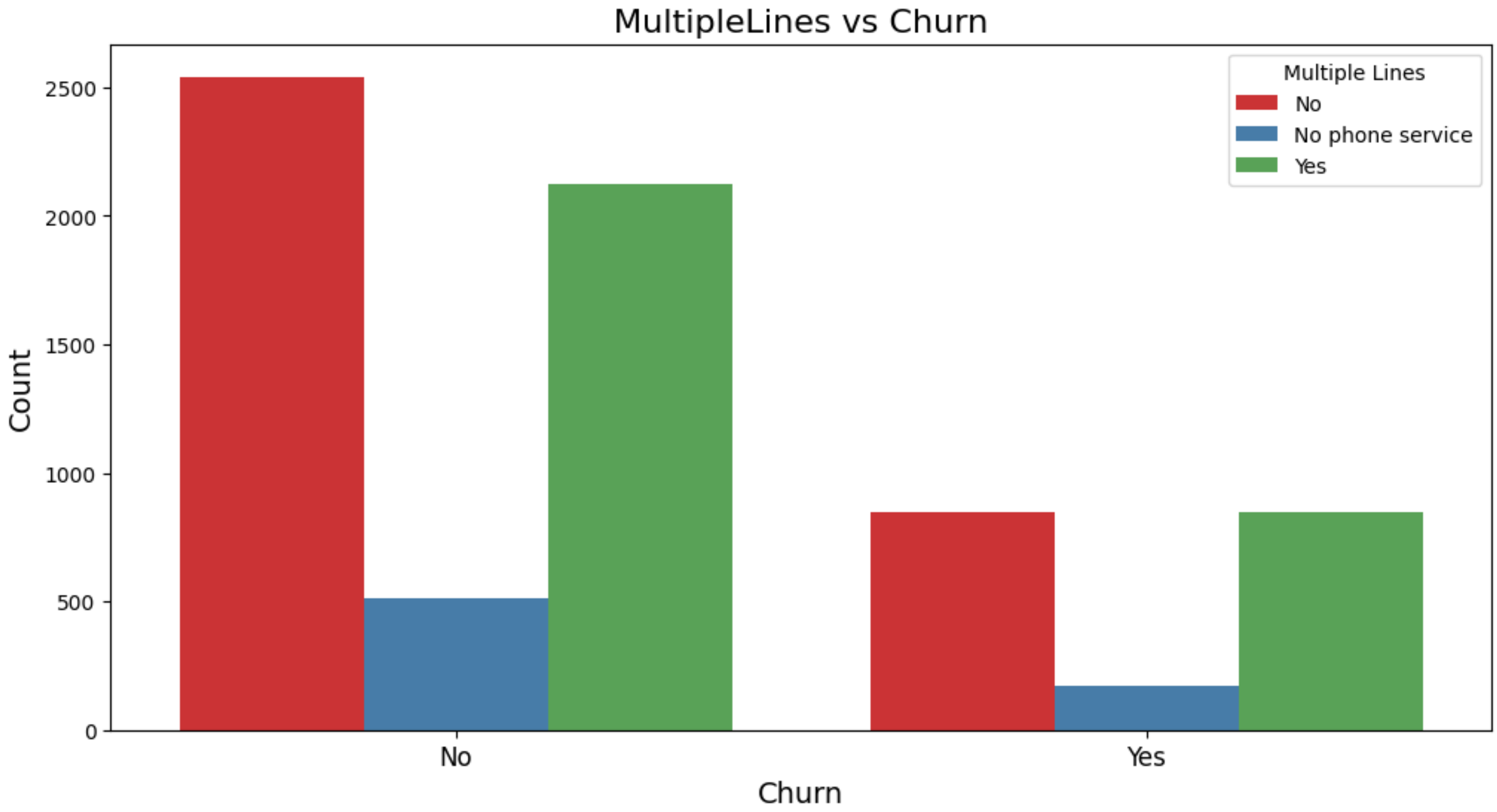


In [15]:
```
pd.crosstab(df['Churn'], df['PhoneService']).plot(kind='bar')
plt.title('PhoneService vs Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.legend(['No', 'Yes'])
plt.xticks(rotation=0)
plt.show()
```

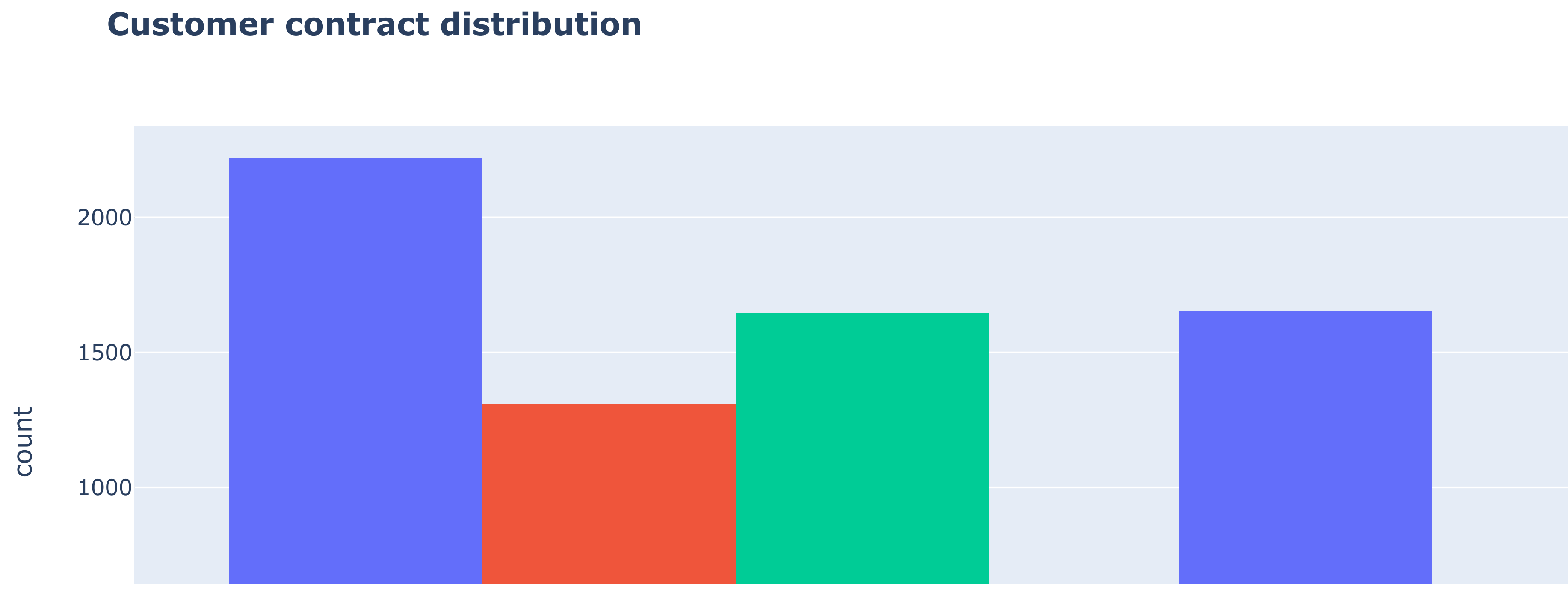


In [16]:

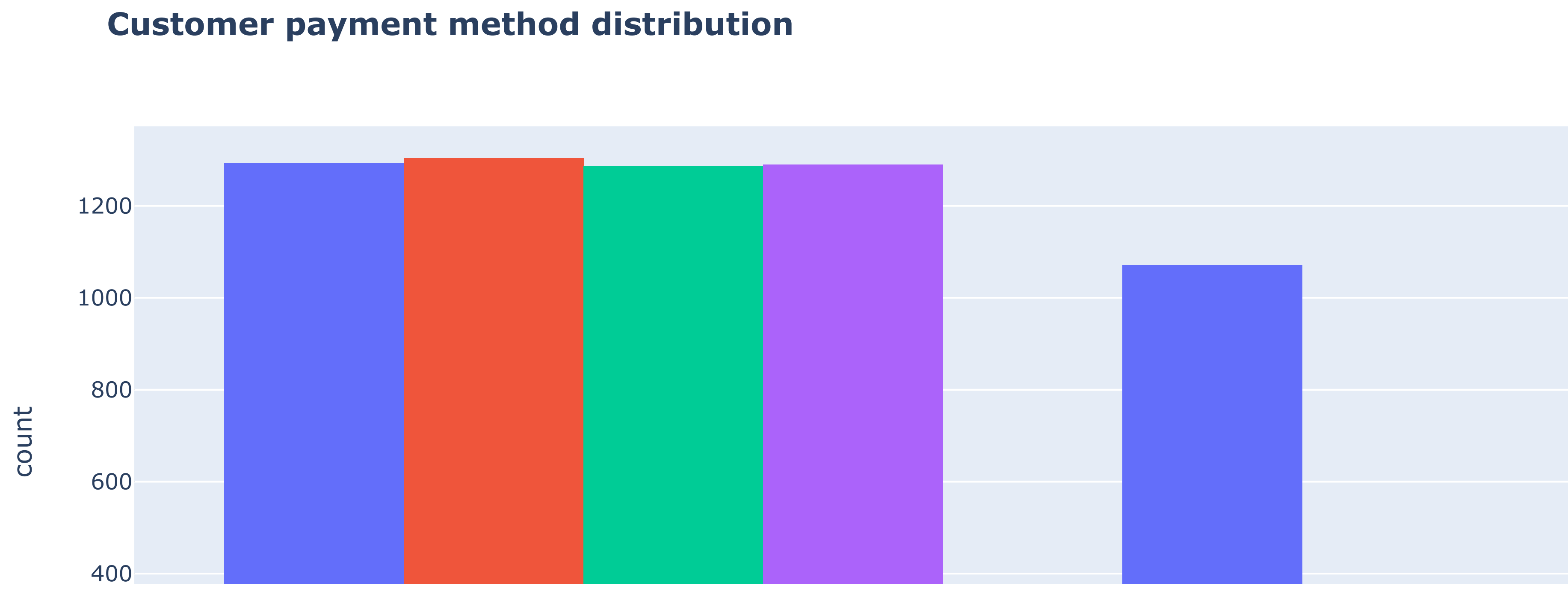
```
# Create a crosstab of Churn and MultipleLines
churn_multiplelines = pd.crosstab(df['Churn'], df['MultipleLines'])

# Reset the index for easier plotting
churn_multiplelines = churn_multiplelines.reset_index()

# Melt the DataFrame for Seaborn compatibility
churn_multiplelines_melted = churn_multiplelines.melt(id_vars='Churn', value_vars=churn_multiplelines.columns[1:]

# Plot using Seaborn
plt.figure(figsize=(12, 6))
sns.barplot(x='Churn', y='Count', hue='MultipleLines', data=churn_multiplelines_melted, palette='Set1')

# Add title and labels
plt.title('MultipleLines vs Churn', fontsize=16)
plt.xlabel('Churn', fontsize=14)
plt.ylabel('Count', fontsize=14)

# Customize legend
plt.legend(title='Multiple Lines')

# Adjust x-ticks for better readability
plt.xticks(rotation=0, fontsize=12)

# Display the plot
plt.show()
```



In [17]:
```
fig = px.histogram(df, x="Churn", color="Contract", barmode="group", title="<b>Customer contract distribution<b>'
fig.show()
```

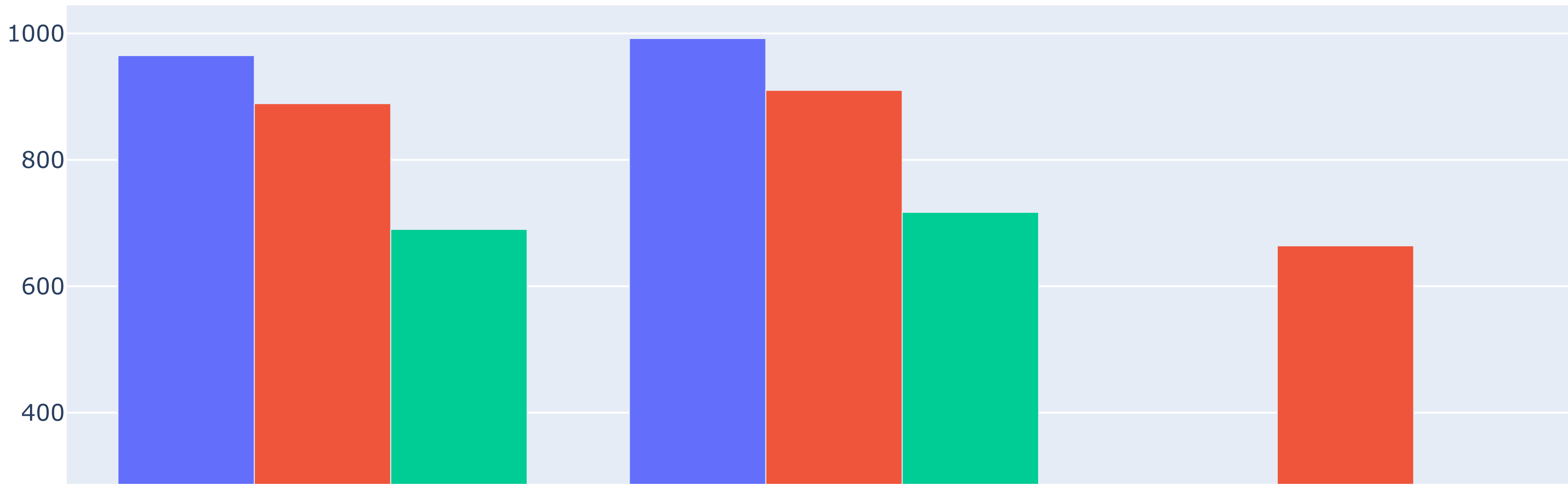


In [18]:
```
fig = px.histogram(df, x="Churn", color="PaymentMethod", barmode="group", title="<b>Customer payment method distr
fig.show()
```

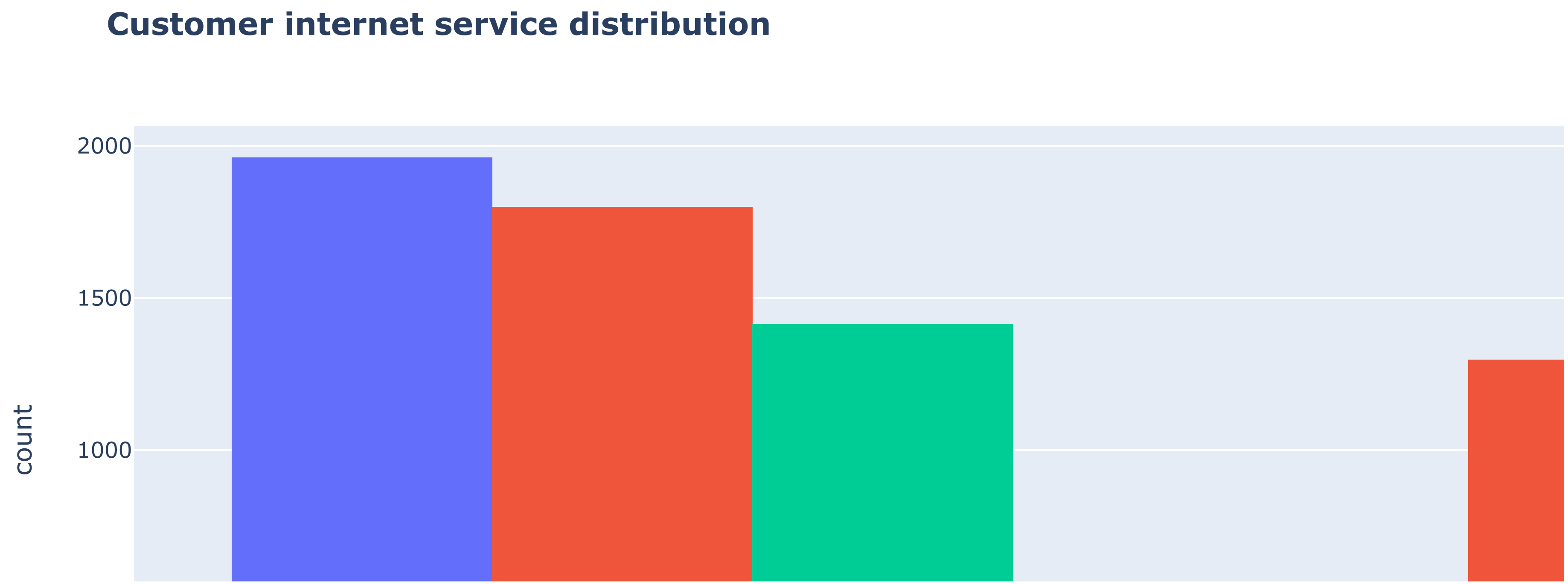

Customer payment method distribution
count
1200
1000
800
600
400

In [19]:
```
import plotly.graph_objects as go
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',
))

fig.update_layout(title_text="<b>Churn Distribution w.r.t. Internet Service and Gender</b>")

fig.show()
```

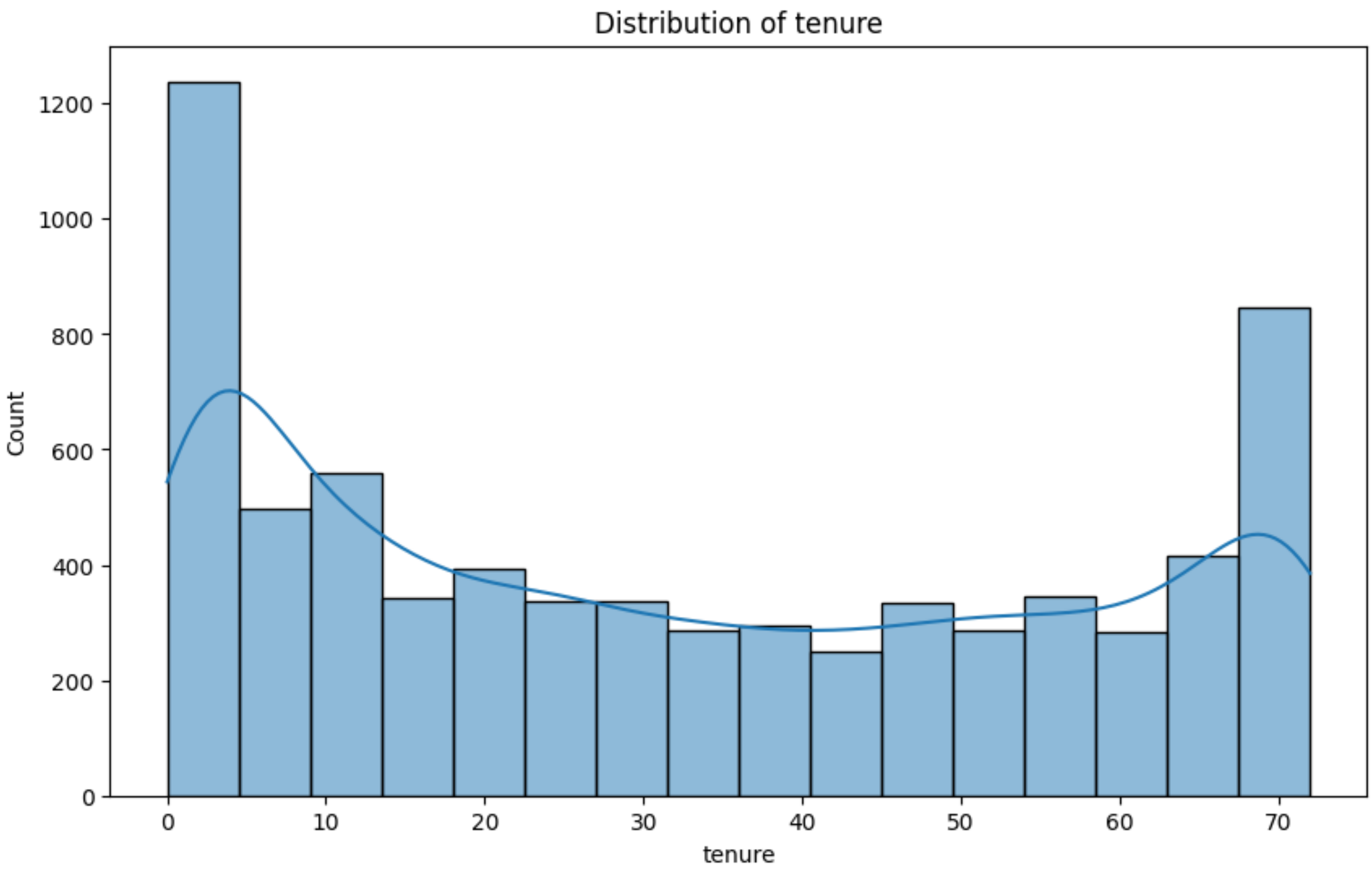
Churn Distribution w.r.t. Internet Service and Gender

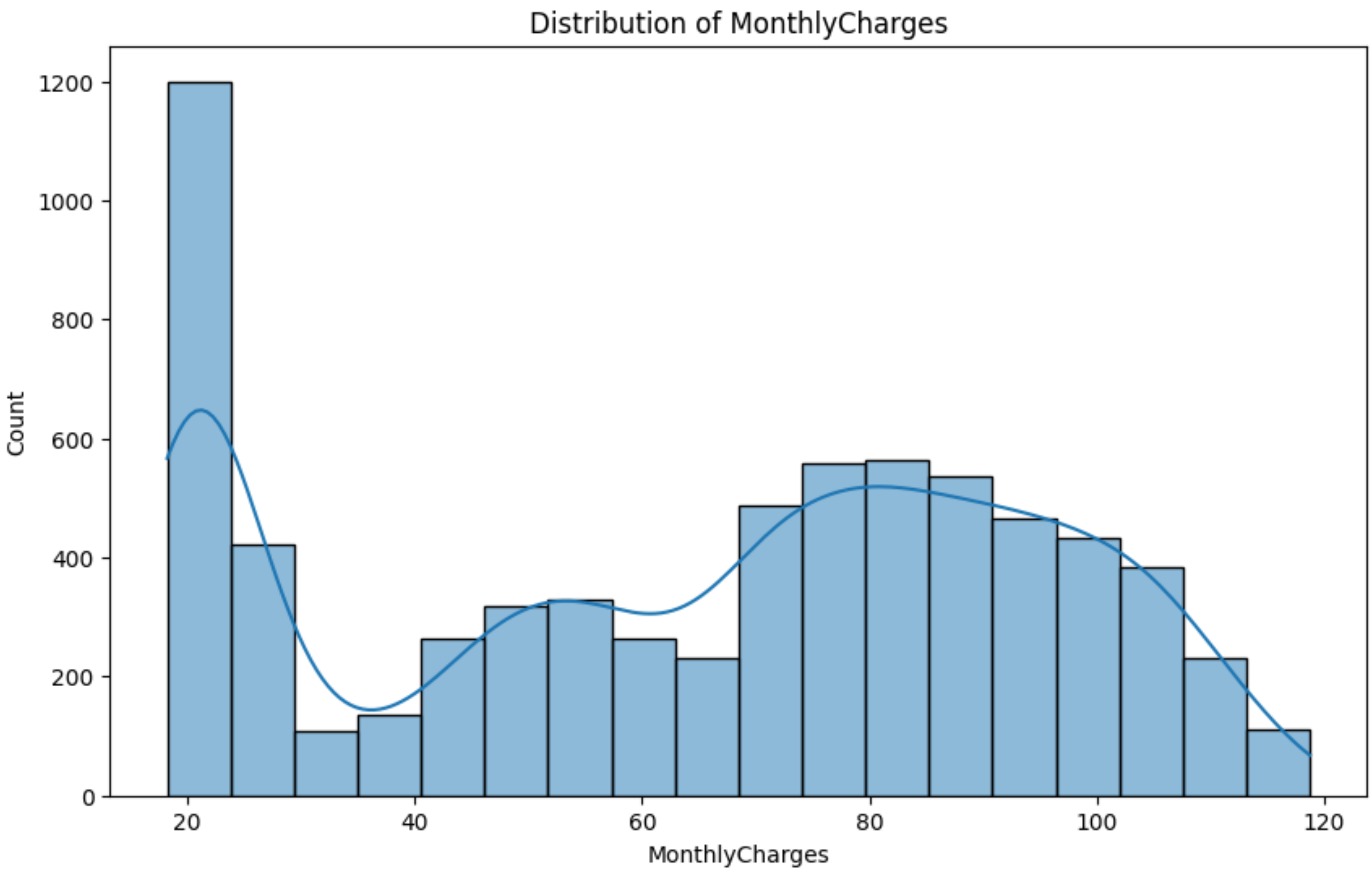


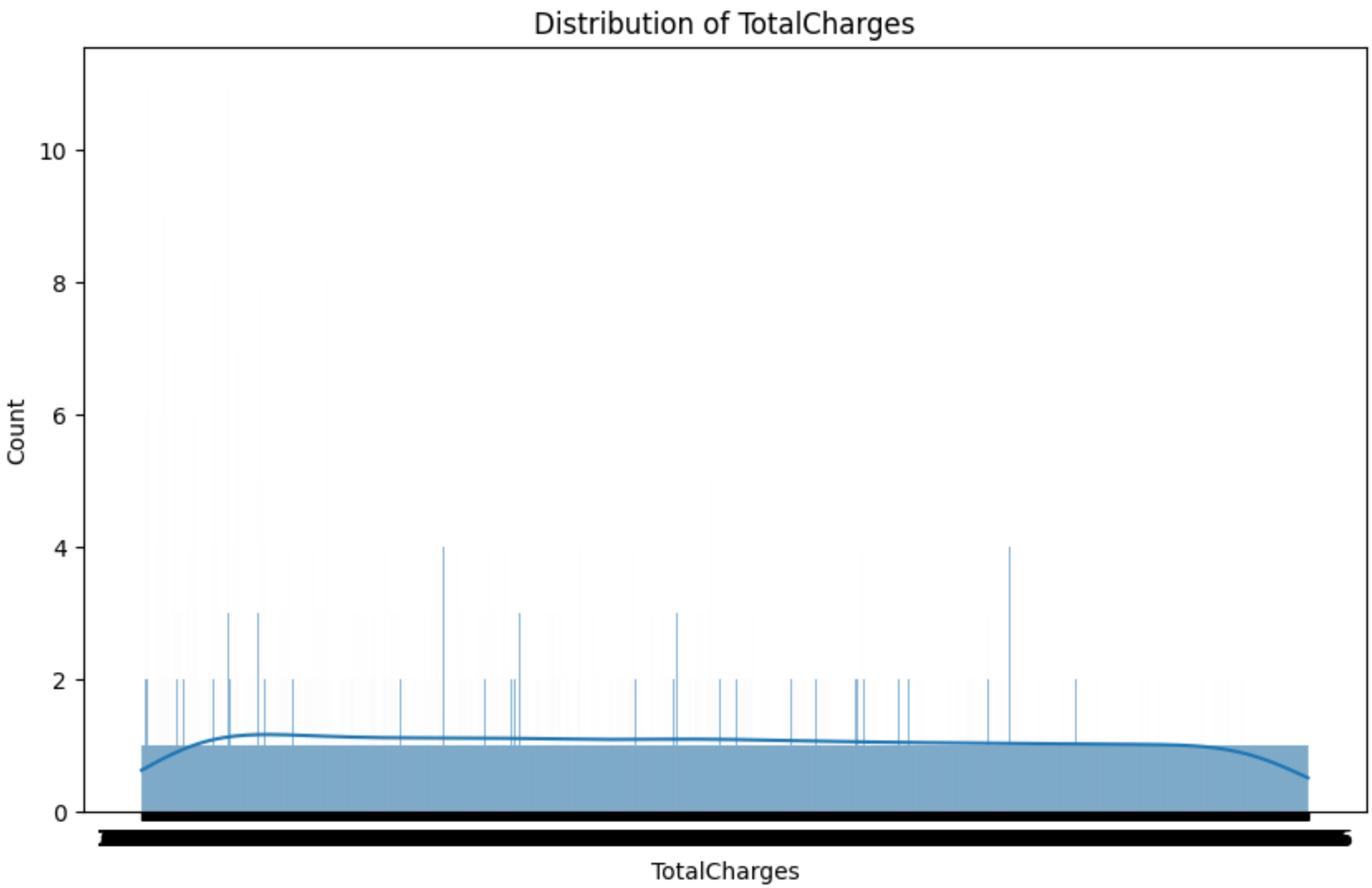
In [20]:
```
fig = px.histogram(df, x="Churn", color="InternetService", barmode="group", title="<b>Customer internet service c
fig.show()
```




```
In [21]: num_cols = ["tenure", "MonthlyCharges", "TotalCharges"]
         for col in num_cols:
             plt.figure(figsize=(10, 6))
             sns.histplot(df[col], kde=True)
             plt.title(f'Distribution of {col}')
             plt.show()
```


Distribution of tenure
Count
tenure
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60
70


Distribution of MonthlyCharges
Count
MonthlyCharges
0
200
400
600
800
1000
1200
20
40
60
80
100
120



Data Handling

In [22]:
```
df.drop('customerID', axis=1, inplace=True)
df.head()
```

Out[22]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	Devi
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 [23]:

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

Out[23]:

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

In [24]:
```
df.fillna(df["TotalCharges"].mean(),inplace=True)
df.isnull().sum()
```

Out[24]:
```
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 [25]:
```
df["SeniorCitizen"]= df["SeniorCitizen"].map({0: "No", 1: "Yes"})
df.head()
```

Out[25]:

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

In [26]:

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

Out[26]:

```
count             7043
unique               3
top        Fiber optic
freq              3096
Name: InternetService, dtype: object
```

In [27]:

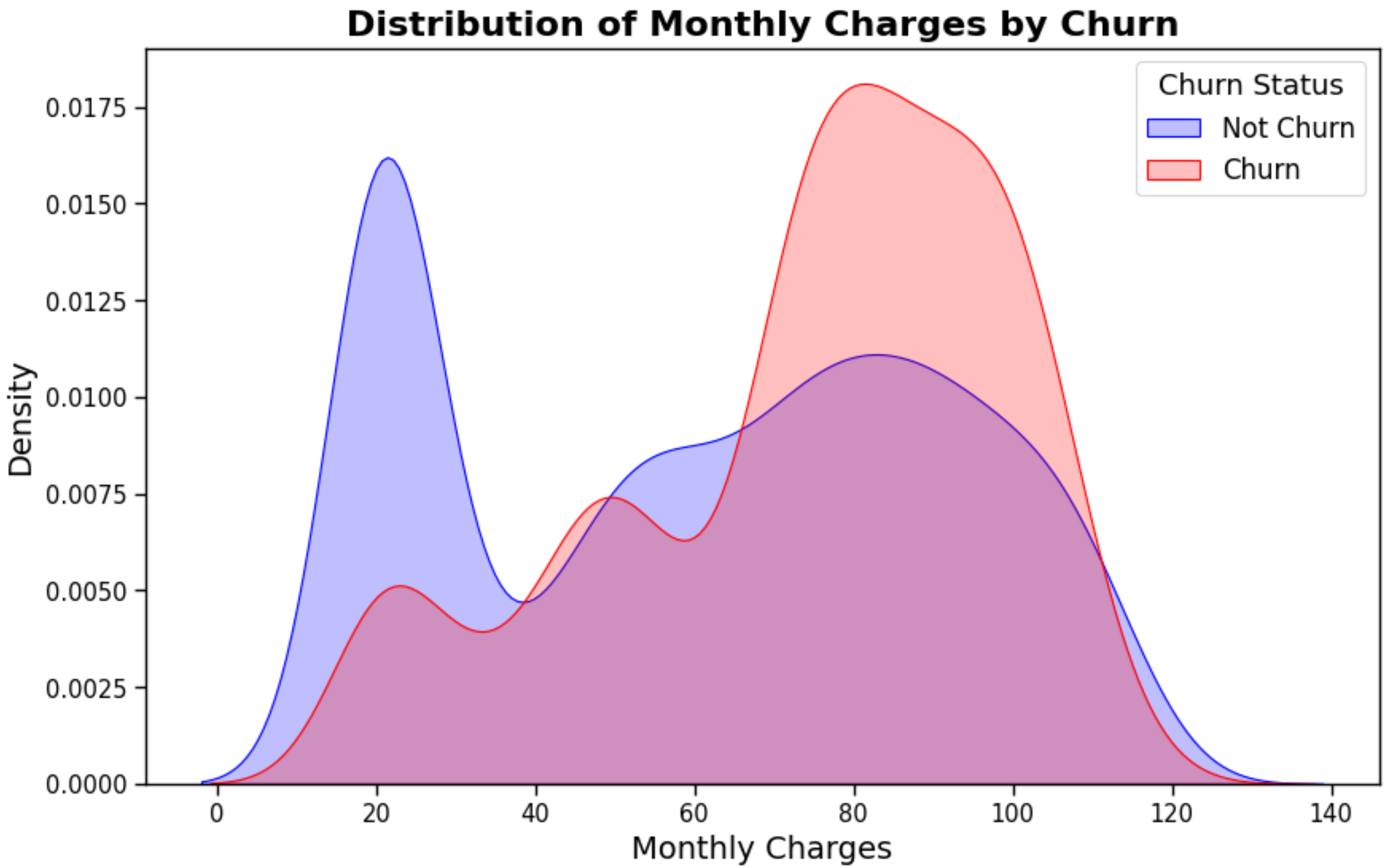
```
df.describe(exclude='object')
```

Out[27]:

	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	2283.300441
std	24.559481	30.090047	2265.000258
min	0.000000	18.250000	18.800000
25%	9.000000	35.500000	402.225000
50%	29.000000	70.350000	1400.550000
75%	55.000000	89.850000	3786.600000
max	72.000000	118.750000	8684.800000

In [28]:

```
# Set the context for the plot
sns.set_context("paper", font_scale=1.2)

# Set the figure size
plt.figure(figsize=(10, 6))

# Plot KDE for "Not Churn"
sns.kdeplot(df.MonthlyCharges[df["Churn"] == 'No'], shade=True, color="blue", label="Not Churn", bw_adjust=1.2)

# Plot KDE for "Churn"
sns.kdeplot(df.MonthlyCharges[df["Churn"] == 'Yes'], shade=True, color="red", label="Churn", bw_adjust=1.2)

# Customize the title and labels
plt.title('Distribution of Monthly Charges by Churn', fontsize=16, weight='bold')
plt.xlabel('Monthly Charges', fontsize=14)
plt.ylabel('Density', fontsize=14)

# Customize the legend
plt.legend(title='Churn Status', loc='upper right', fontsize=12, title_fontsize='13')

# Display the plot
plt.show()
```



Data Preprocessing

In [29]:
```
def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series
```

In [30]:
```
df = df.apply(lambda x: object_to_int(x))
df.head()
```

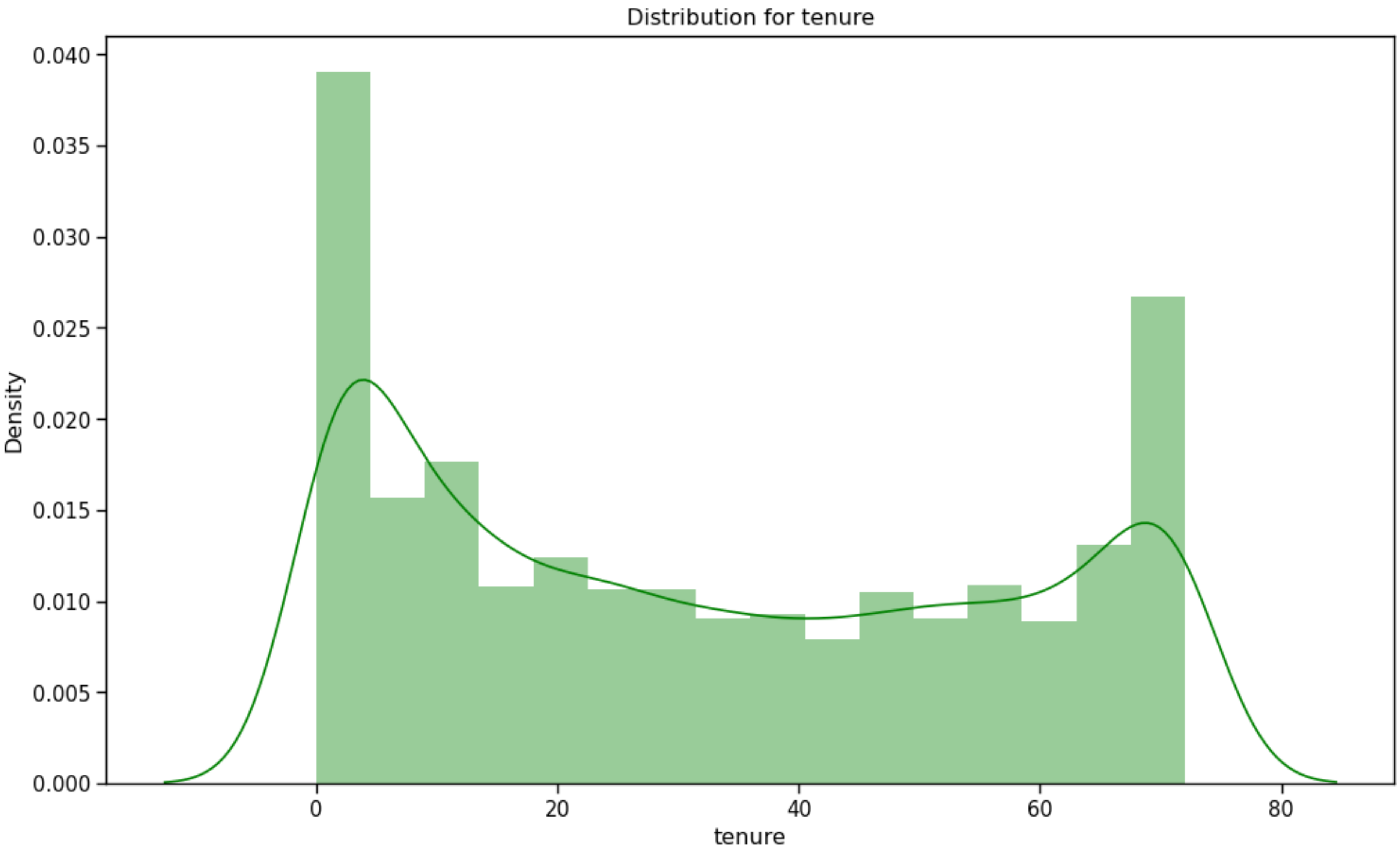
Out[30]:

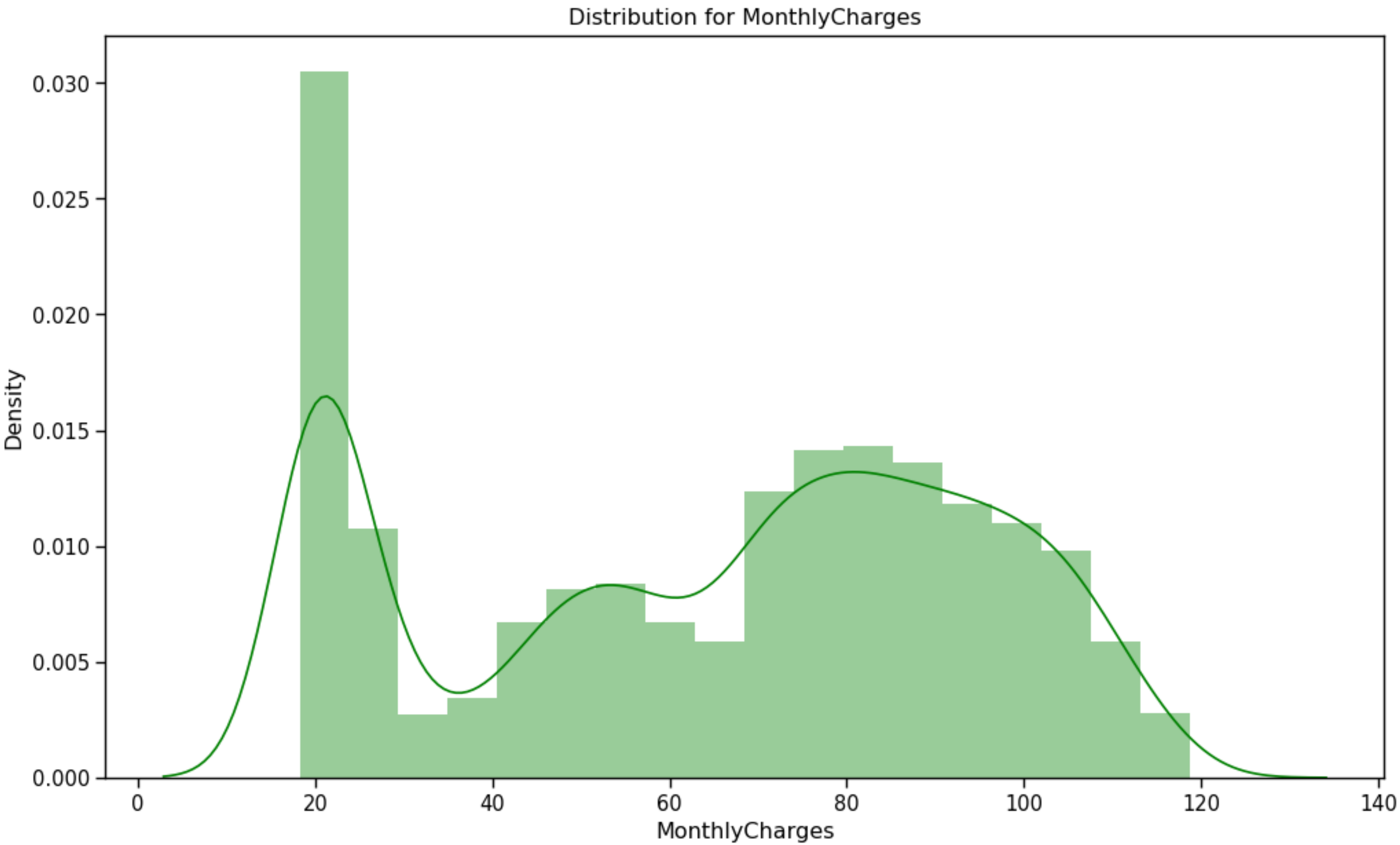
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	Devi
0	0	0	1	0	1	0	1	0	0	2	
1	1	0	0	0	34	1	0	0	2	0	
2	1	0	0	0	2	1	0	0	2	2	
3	1	0	0	0	45	0	1	0	2	0	
4	0	0	0	0	2	1	0	1	0	0	

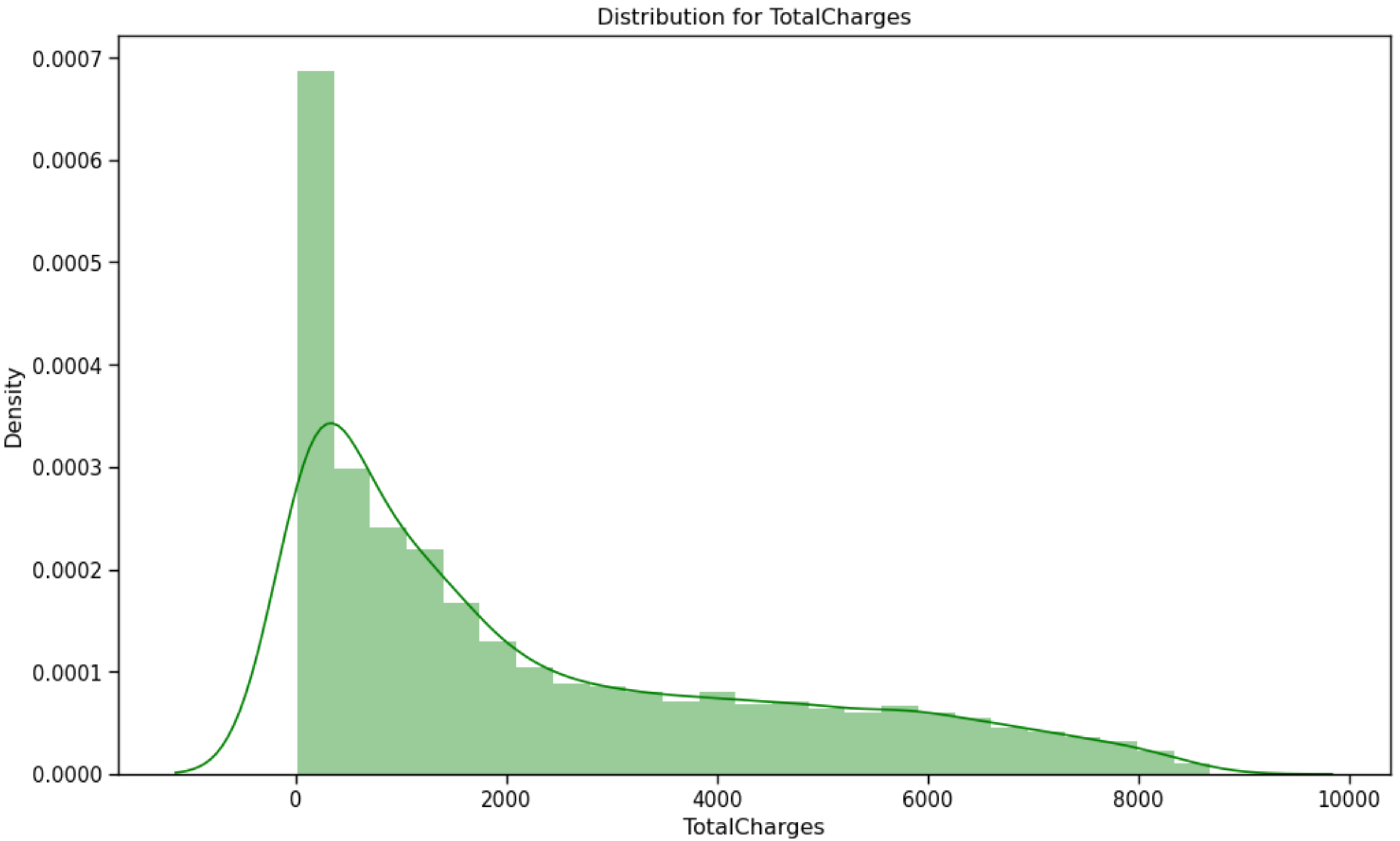
In [31]:
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [32]:
```
def distplot(feature, frame, color='g'):
    plt.figure(figsize=(12, 7))
    plt.title("Distribution for {}".format(feature))
    ax = sns.distplot(frame[feature], color= color)
```

In [33]:
```
num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
for feat in num_cols: distplot(feat, df)
```


Distribution for tenure
Density
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
tenure


Distribution for MonthlyCharges
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0
20
40
60
80
100
120
140
MonthlyCharges



In [34]:
```
# StandardScaler
scaler= StandardScaler()

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

Machine Learning Models

KNN

In [35]:
```
knn_model = KNeighborsClassifier(n_neighbors=2)
knn_model.fit(X_train, y_train)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
print("KNN accuracy:",accuracy_knn)
```

```
KNN accuracy: 0.7700496806245565
```

In [36]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.79      0.94      0.86      1036
           1       0.64      0.31      0.41       373

    accuracy                           0.77      1409
   macro avg       0.71      0.62      0.64      1409
weighted avg       0.75      0.77      0.74      1409
```

Random Forest

In [37]:
```
model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))
```

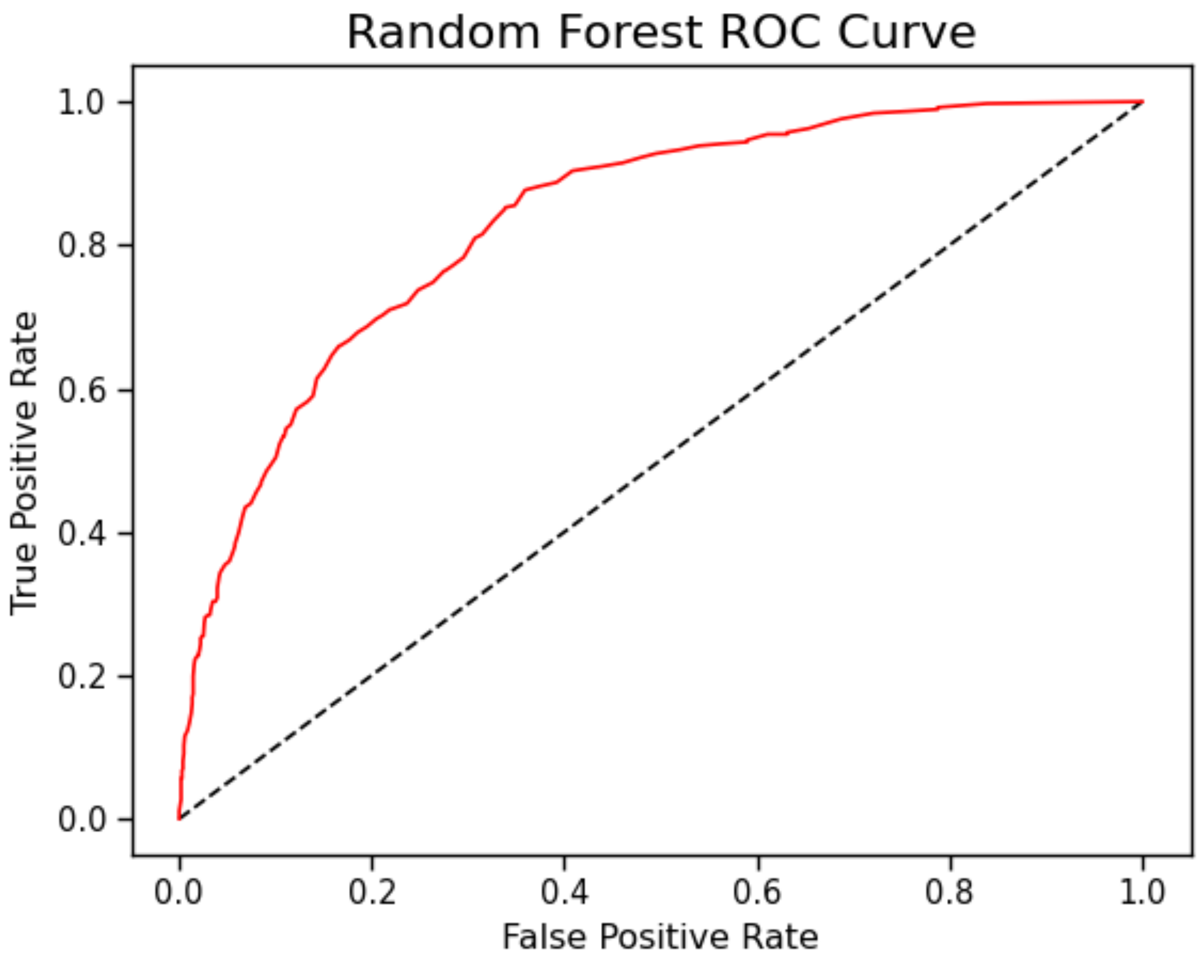
```
0.7970191625266146
```

In [38]:
```
print(classification_report(y_test, prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1036
           1       0.66      0.49      0.56       373

    accuracy                           0.80      1409
   macro avg       0.74      0.70      0.71      1409
weighted avg       0.79      0.80      0.79      1409
```

In [39]:

```
y_rfpred_prob = model_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();
```



Logistic Regression

In [40]:
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

```
Logistic Regression accuracy is : 0.8168914123491838
```

In [41]:
```
lr_pred= lr_model.predict(X_test)
print(classification_report(y_test,lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1036
           1       0.68      0.58      0.63       373

    accuracy                           0.82      1409
   macro avg       0.77      0.74      0.75      1409
weighted avg       0.81      0.82      0.81      1409
```

XGBOOST

In [42]:
```
xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train)
accuracy_xgb = xgb_model.score(X_test,y_test)
print("XGBOOST accuracy is :",accuracy_xgb)
```

```
XGBOOST accuracy is : 0.7821149751596878
```

In [43]:
```
print(classification_report(y_test, xgb_model.predict(X_test)))
```

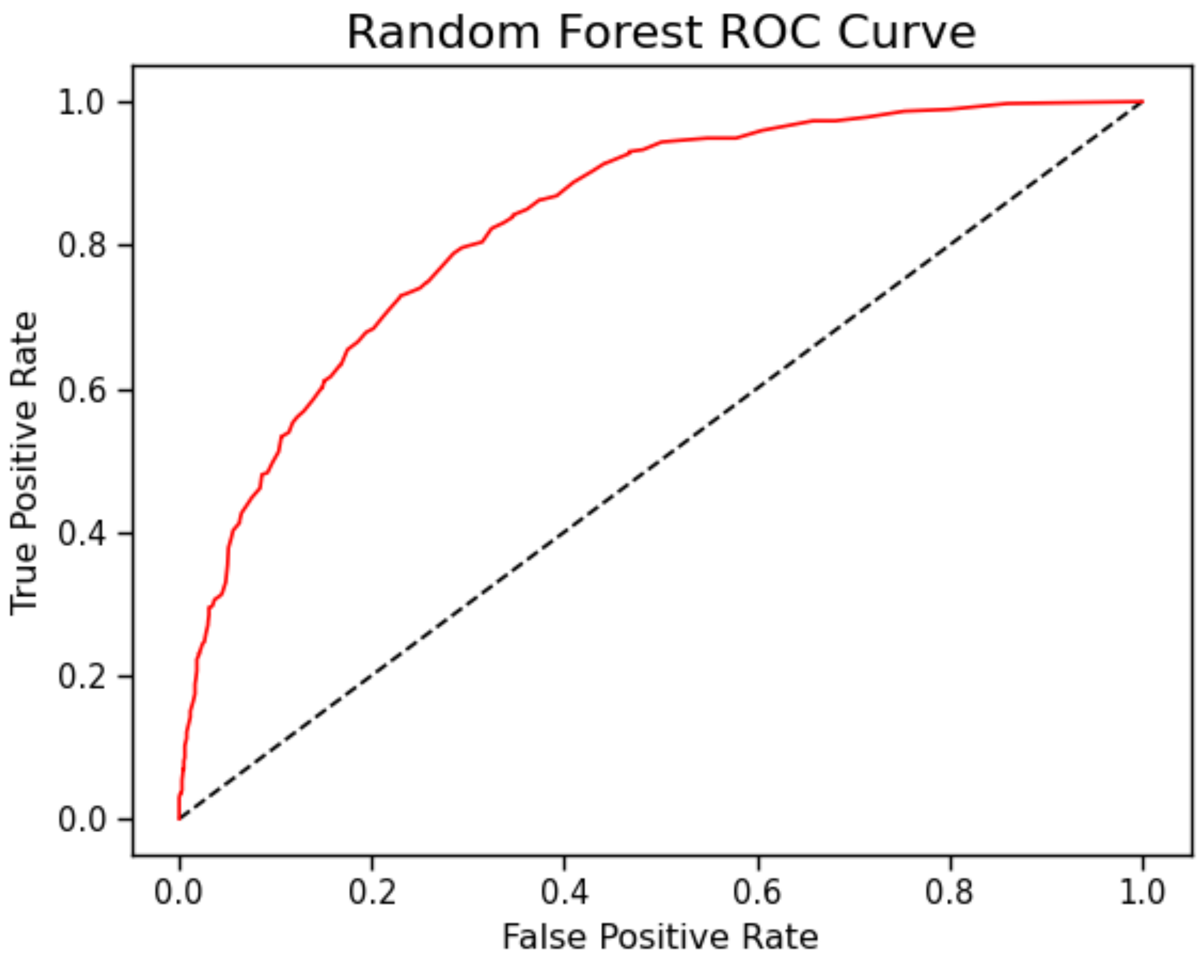
```
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      1036
           1       0.61      0.50      0.55       373

    accuracy                           0.78      1409
   macro avg       0.72      0.69      0.70      1409
weighted avg       0.77      0.78      0.77      1409
```

In [44]:
```
# KNN

knn_model = KNeighborsClassifier(n_neighbors=2)
knn_model.fit(X_train, y_train)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
```

```python
print("KNN accuracy:",accuracy_knn)

print(classification_report(y_test, predicted_y))

# Random Forest

model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))

print(classification_report(y_test, prediction_test))

y_rfpred_prob = model_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();

# Logistic Regression

lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)

lr_pred= lr_model.predict(X_test)
print(classification_report(y_test,lr_pred))

# XGBOOST

xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train)
accuracy_xgb = xgb_model.score(X_test,y_test)
print("XGBOOST accuracy is :",accuracy_xgb)
```

```python
print(classification_report(y_test, xgb_model.predict(X_test)))
```

```
KNN accuracy: 0.7700496806245565
              precision    recall  f1-score   support

           0       0.79      0.94      0.86      1036
           1       0.64      0.31      0.41       373

    accuracy                           0.77      1409
   macro avg       0.71      0.62      0.64      1409
weighted avg       0.75      0.77      0.74      1409

0.7955997161107168
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1036
           1       0.65      0.48      0.56       373

    accuracy                           0.80      1409
   macro avg       0.74      0.70      0.71      1409
weighted avg       0.78      0.80      0.78      1409
```



Random Forest ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```
Logistic Regression accuracy is : 0.8168914123491838
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1036
           1       0.68      0.58      0.63       373

    accuracy                           0.82      1409
   macro avg       0.77      0.74      0.75      1409
weighted avg       0.81      0.82      0.81      1409

XGBOOST accuracy is : 0.7821149751596878
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      1036
           1       0.61      0.50      0.55       373

    accuracy                           0.78      1409
   macro avg       0.72      0.69      0.70      1409
weighted avg       0.77      0.78      0.77      1409
```

Deep Learning Model

In [45]:
```python
model = Sequential()

# Input layer with L2 regularization
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu', kernel_regularizer=l2(0.001)))

# Additional hidden layers with L2 regularization and Dropout
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.0001)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.0001)))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.0001)))

# Output layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 1/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5850 - loss: 0.7115 - val_accuracy: 0.7353 - val_loss: 0.59
35
Epoch 2/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7241 - loss: 0.6051 - val_accuracy: 0.7353 - val_loss: 0.52
96
Epoch 3/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7309 - loss: 0.5609 - val_accuracy: 0.7402 - val_loss: 0.49
82
Epoch 4/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7414 - loss: 0.5287 - val_accuracy: 0.7743 - val_loss: 0.48
20
Epoch 5/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7611 - loss: 0.5169 - val_accuracy: 0.8006 - val_loss: 0.47
29
Epoch 6/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7665 - loss: 0.5098 - val_accuracy: 0.8077 - val_loss: 0.46
65
Epoch 7/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7836 - loss: 0.4964 - val_accuracy: 0.8133 - val_loss: 0.46
```

```
40
Epoch 8/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7695 - loss: 0.4986 - val_accuracy: 0.8126 - val_loss: 0.46
07
Epoch 9/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7843 - loss: 0.4990 - val_accuracy: 0.8148 - val_loss: 0.45
81
Epoch 10/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7812 - loss: 0.5031 - val_accuracy: 0.8105 - val_loss: 0.45
65
Epoch 11/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7775 - loss: 0.4906 - val_accuracy: 0.8169 - val_loss: 0.45
48
Epoch 12/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7782 - loss: 0.5024 - val_accuracy: 0.8155 - val_loss: 0.45
48
Epoch 13/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7846 - loss: 0.4954 - val_accuracy: 0.8133 - val_loss: 0.45
29
Epoch 14/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7845 - loss: 0.4858 - val_accuracy: 0.8148 - val_loss: 0.45
17
Epoch 15/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7895 - loss: 0.4824 - val_accuracy: 0.8119 - val_loss: 0.45
10
Epoch 16/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7891 - loss: 0.4737 - val_accuracy: 0.8133 - val_loss: 0.44
91
Epoch 17/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7932 - loss: 0.4808 - val_accuracy: 0.8112 - val_loss: 0.44
91
Epoch 18/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 986us/step - accuracy: 0.7881 - loss: 0.4838 - val_accuracy: 0.8112 - val_loss: 0.
4486
Epoch 19/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7861 - loss: 0.4838 - val_accuracy: 0.8148 - val_loss: 0.44
79
Epoch 20/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8024 - loss: 0.4637 - val_accuracy: 0.8148 - val_loss: 0.44
77
Epoch 21/100
```

```
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7903 - loss: 0.4877 - val_accuracy: 0.8162 - val_loss: 0.44
60
Epoch 22/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7888 - loss: 0.4773 - val_accuracy: 0.8141 - val_loss: 0.44
59
Epoch 23/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7988 - loss: 0.4648 - val_accuracy: 0.8155 - val_loss: 0.44
62
Epoch 24/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7957 - loss: 0.4668 - val_accuracy: 0.8162 - val_loss: 0.44
45
Epoch 25/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7876 - loss: 0.4791 - val_accuracy: 0.8141 - val_loss: 0.44
43
Epoch 26/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 985us/step - accuracy: 0.7970 - loss: 0.4691 - val_accuracy: 0.8176 - val_loss: 0.
4433
Epoch 27/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8046 - loss: 0.4621 - val_accuracy: 0.8169 - val_loss: 0.44
39
Epoch 28/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7976 - loss: 0.4777 - val_accuracy: 0.8190 - val_loss: 0.44
24
Epoch 29/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7972 - loss: 0.4698 - val_accuracy: 0.8183 - val_loss: 0.44
20
Epoch 30/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7904 - loss: 0.4647 - val_accuracy: 0.8190 - val_loss: 0.44
14
Epoch 31/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7976 - loss: 0.4623 - val_accuracy: 0.8197 - val_loss: 0.44
23
Epoch 32/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7986 - loss: 0.4636 - val_accuracy: 0.8204 - val_loss: 0.44
09
Epoch 33/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7995 - loss: 0.4641 - val_accuracy: 0.8190 - val_loss: 0.43
97
Epoch 34/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7975 - loss: 0.4625 - val_accuracy: 0.8204 - val_loss: 0.43
96
```

```
Epoch 35/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8106 - loss: 0.4524 - val_accuracy: 0.8190 - val_loss: 0.44
02
Epoch 36/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7991 - loss: 0.4669 - val_accuracy: 0.8190 - val_loss: 0.43
92
Epoch 37/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8037 - loss: 0.4589 - val_accuracy: 0.8211 - val_loss: 0.43
94
Epoch 38/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8024 - loss: 0.4626 - val_accuracy: 0.8197 - val_loss: 0.43
98
Epoch 39/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8060 - loss: 0.4640 - val_accuracy: 0.8169 - val_loss: 0.43
84
Epoch 40/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8002 - loss: 0.4515 - val_accuracy: 0.8204 - val_loss: 0.43
89
Epoch 41/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7950 - loss: 0.4568 - val_accuracy: 0.8197 - val_loss: 0.43
83
Epoch 42/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7998 - loss: 0.4536 - val_accuracy: 0.8197 - val_loss: 0.43
78
Epoch 43/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8025 - loss: 0.4537 - val_accuracy: 0.8183 - val_loss: 0.43
90
Epoch 44/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7985 - loss: 0.4641 - val_accuracy: 0.8190 - val_loss: 0.43
77
Epoch 45/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8007 - loss: 0.4584 - val_accuracy: 0.8219 - val_loss: 0.43
78
Epoch 46/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8042 - loss: 0.4524 - val_accuracy: 0.8233 - val_loss: 0.43
78
Epoch 47/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8024 - loss: 0.4536 - val_accuracy: 0.8219 - val_loss: 0.43
79
Epoch 48/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8025 - loss: 0.4536 - val_accuracy: 0.8219 - val_loss: 0.43
```

```
75
Epoch 49/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8031 - loss: 0.4508 - val_accuracy: 0.8197 - val_loss: 0.43
77
Epoch 50/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8066 - loss: 0.4498 - val_accuracy: 0.8211 - val_loss: 0.43
77
Epoch 51/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8047 - loss: 0.4575 - val_accuracy: 0.8226 - val_loss: 0.43
67
Epoch 52/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8010 - loss: 0.4634 - val_accuracy: 0.8219 - val_loss: 0.43
74
Epoch 53/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 994us/step - accuracy: 0.8070 - loss: 0.4485 - val_accuracy: 0.8204 - val_loss: 0.
4359
Epoch 54/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7984 - loss: 0.4560 - val_accuracy: 0.8219 - val_loss: 0.43
65
Epoch 55/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8047 - loss: 0.4507 - val_accuracy: 0.8226 - val_loss: 0.43
72
Epoch 56/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - accuracy: 0.8063 - loss: 0.4521 - val_accuracy: 0.8211 - val_loss: 0.
4359
Epoch 57/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 993us/step - accuracy: 0.8044 - loss: 0.4466 - val_accuracy: 0.8211 - val_loss: 0.
4358
Epoch 58/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 995us/step - accuracy: 0.8089 - loss: 0.4552 - val_accuracy: 0.8240 - val_loss: 0.
4352
Epoch 59/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - accuracy: 0.7983 - loss: 0.4631 - val_accuracy: 0.8254 - val_loss: 0.
4355
Epoch 60/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8105 - loss: 0.4298 - val_accuracy: 0.8240 - val_loss: 0.43
59
Epoch 61/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8019 - loss: 0.4418 - val_accuracy: 0.8261 - val_loss: 0.43
54
Epoch 62/100
```

```
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8128 - loss: 0.4406 - val_accuracy: 0.8247 - val_loss: 0.43
51
Epoch 63/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8072 - loss: 0.4427 - val_accuracy: 0.8247 - val_loss: 0.43
55
Epoch 64/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8037 - loss: 0.4447 - val_accuracy: 0.8247 - val_loss: 0.43
67
Epoch 65/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8168 - loss: 0.4406 - val_accuracy: 0.8247 - val_loss: 0.43
61
Epoch 66/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 992us/step - accuracy: 0.8103 - loss: 0.4388 - val_accuracy: 0.8240 - val_loss: 0.
4351
Epoch 67/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 991us/step - accuracy: 0.8050 - loss: 0.4431 - val_accuracy: 0.8240 - val_loss: 0.
4341
Epoch 68/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8085 - loss: 0.4469 - val_accuracy: 0.8204 - val_loss: 0.43
33
Epoch 69/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8109 - loss: 0.4320 - val_accuracy: 0.8226 - val_loss: 0.43
36
Epoch 70/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8124 - loss: 0.4383 - val_accuracy: 0.8219 - val_loss: 0.43
33
Epoch 71/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8043 - loss: 0.4430 - val_accuracy: 0.8233 - val_loss: 0.43
29
Epoch 72/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8174 - loss: 0.4318 - val_accuracy: 0.8233 - val_loss: 0.43
27
Epoch 73/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8142 - loss: 0.4384 - val_accuracy: 0.8226 - val_loss: 0.43
28
Epoch 74/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8141 - loss: 0.4405 - val_accuracy: 0.8211 - val_loss: 0.43
30
Epoch 75/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8239 - loss: 0.4285 - val_accuracy: 0.8240 - val_loss: 0.43
36
```

```
Epoch 76/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8140 - loss: 0.4365 - val_accuracy: 0.8219 - val_loss: 0.43
29
Epoch 77/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8028 - loss: 0.4402 - val_accuracy: 0.8233 - val_loss: 0.43
30
Epoch 78/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8103 - loss: 0.4445 - val_accuracy: 0.8211 - val_loss: 0.43
26
Epoch 79/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8178 - loss: 0.4313 - val_accuracy: 0.8211 - val_loss: 0.43
34
Epoch 80/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8109 - loss: 0.4406 - val_accuracy: 0.8226 - val_loss: 0.43
39
Epoch 81/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8093 - loss: 0.4413 - val_accuracy: 0.8211 - val_loss: 0.43
31
Epoch 82/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8144 - loss: 0.4321 - val_accuracy: 0.8197 - val_loss: 0.43
33
Epoch 83/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8180 - loss: 0.4370 - val_accuracy: 0.8197 - val_loss: 0.43
36
Epoch 84/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8110 - loss: 0.4452 - val_accuracy: 0.8211 - val_loss: 0.43
34
Epoch 85/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8209 - loss: 0.4312 - val_accuracy: 0.8133 - val_loss: 0.43
41
Epoch 86/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8153 - loss: 0.4333 - val_accuracy: 0.8204 - val_loss: 0.43
35
Epoch 87/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8108 - loss: 0.4419 - val_accuracy: 0.8233 - val_loss: 0.43
28
Epoch 88/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8115 - loss: 0.4305 - val_accuracy: 0.8233 - val_loss: 0.43
31
Epoch 89/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8137 - loss: 0.4371 - val_accuracy: 0.8219 - val_loss: 0.43
```

```
32
Epoch 90/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8034 - loss: 0.4469 - val_accuracy: 0.8190 - val_loss: 0.43
32
Epoch 91/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8225 - loss: 0.4202 - val_accuracy: 0.8197 - val_loss: 0.43
42
Epoch 92/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8103 - loss: 0.4371 - val_accuracy: 0.8204 - val_loss: 0.43
36
Epoch 93/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8165 - loss: 0.4259 - val_accuracy: 0.8211 - val_loss: 0.43
36
Epoch 94/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8158 - loss: 0.4313 - val_accuracy: 0.8190 - val_loss: 0.43
37
Epoch 95/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8204 - loss: 0.4150 - val_accuracy: 0.8190 - val_loss: 0.43
35
Epoch 96/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8189 - loss: 0.4381 - val_accuracy: 0.8183 - val_loss: 0.43
40
Epoch 97/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8028 - loss: 0.4439 - val_accuracy: 0.8183 - val_loss: 0.43
35
Epoch 98/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8246 - loss: 0.4132 - val_accuracy: 0.8204 - val_loss: 0.43
37
Epoch 99/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8289 - loss: 0.4131 - val_accuracy: 0.8176 - val_loss: 0.43
46
Epoch 100/100
89/89 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8200 - loss: 0.4261 - val_accuracy: 0.8169 - val_loss: 0.43
41
```

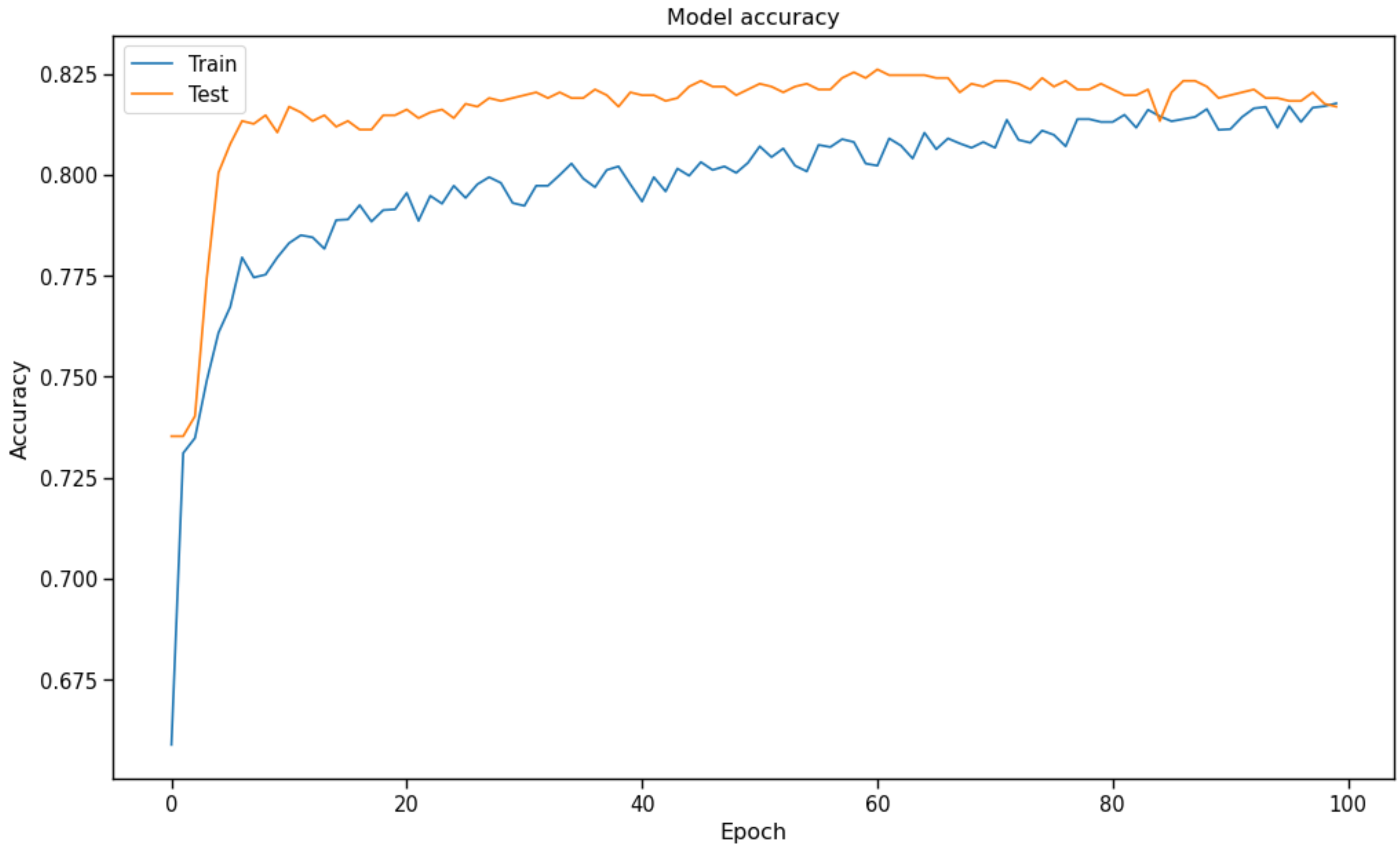
In [46]:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

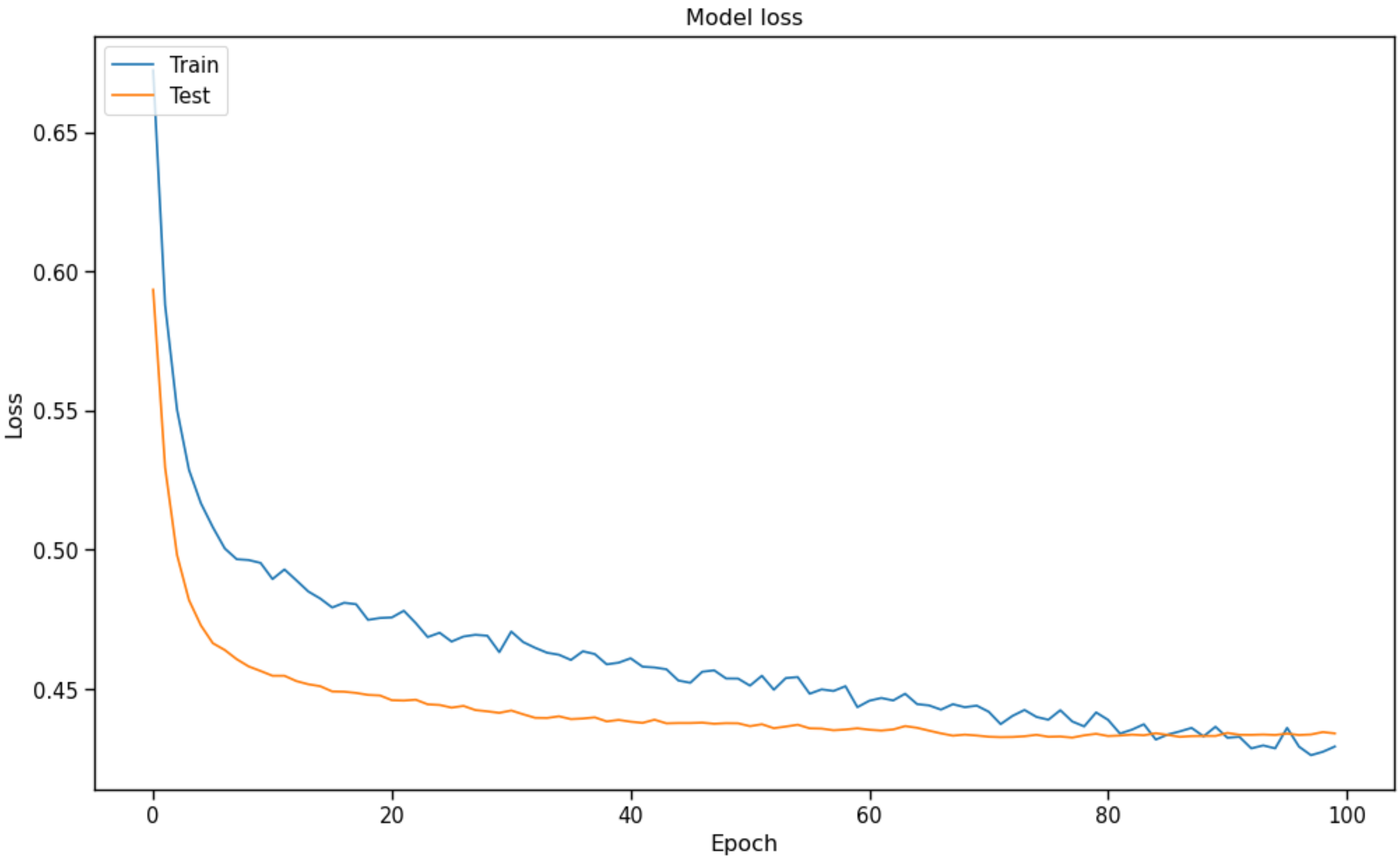
```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 417us/step - accuracy: 0.8044 - loss: 0.4370
Test Accuracy: 0.8169
```

In [47]:

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 7))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(12, 7))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Model accuracy
Train
Test
0.825
0.800
0.775
0.750
0.725
0.700
0.675
Accuracy
0
20
40
60
80
100
Epoch


Model loss
Train
Test
0.65
0.60
0.55
0.50
0.45
Loss
0
20
40
60
80
100
Epoch

In [48]:
```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 986us/step
[[934 102]
 [156 217]]
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1036
           1       0.68      0.58      0.63       373

    accuracy                           0.82      1409
   macro avg       0.77      0.74      0.75      1409
weighted avg       0.81      0.82      0.81      1409
```

Key Findings

Summary of Findings

1.High Monthly Charges: Customers with higher monthly charges are more likely to churn. This suggests a need for value reassessment or targeted discounts for high-paying customers.

2.Contract Type: Customers on short-term or month-to-month contracts are more likely to churn. Incentivizing longer-term contracts could improve retention.

3.Service Usage: Customers who do not use additional services (e.g., MultipleLines, OnlineSecurity) may be more prone to churn. Offering bundled services at a discount might help retain these customers.

4.Proactive Interventions: By predicting churn, the company can proactively reach out to at-risk customers before they decide to leave, potentially reducing overall churn rates.

5.Tenure and Loyalty:Customers with shorter tenure (e.g., less than a year) are often at higher risk of churn. This suggests that early engagement strategies are crucial. Companies could implement loyalty programs, onboarding processes, or personalized communication to foster stronger relationships with new customers and reduce the likelihood of early churn.

6.Payment Method:The model might reveal that customers using certain payment methods (e.g., month-to-month billing with manual payments) have a higher churn rate. Encouraging customers to switch to automated payments or offering discounts for upfront payments could reduce churn by increasing convenience and commitment.

7.Customer Support Interaction:Frequent interactions with customer support, especially those involving complaints, might indicate dissatisfaction, leading to a higher risk of churn. Improving customer support, resolving issues promptly, and offering compensation for negative experiences could enhance customer satisfaction and retention.

8.Internet Service Type:If certain internet service types (e.g., DSL) are associated with higher churn rates compared to others (e.g., fiber optic), this could indicate a need to upgrade infrastructure or offer better service plans to customers in areas with inferior service quality.

9.Promotion and Discount Utilization:Customers who initially signed up during promotional periods or with discounts might have higher churn rates once those promotions expire. To retain these customers, the company could offer extended promotions, loyalty discounts, or alternative value-added services when the initial offer ends.

10.Geographical Location:If the model indicates that churn rates are higher in specific geographical areas, it may suggest local competition, service quality issues, or demographic factors. Targeted marketing campaigns, infrastructure improvements, or localized customer engagement efforts could help address these issues.

11.Service Upgrades and Downgrades:Customers who frequently change their service plans (e.g., downgrading to a lower tier) may be at higher risk of churn. This could suggest dissatisfaction with the value received. Offering more flexible plans or value-added features that encourage customers to stay could mitigate this risk.

Strategic Recommendations

1. Early Engagement Programs: Implement initiatives such as personalized welcome packages, early-stage loyalty rewards, or tailored communication strategies to improve retention among new customers.
2. Automated Payment Incentives: Offer discounts or additional benefits to customers who switch to automated payment methods to reduce the risk of churn due to payment convenience.
3. Customer Feedback Loops: Establish regular feedback loops, particularly for customers who frequently contact support, to address pain points and enhance satisfaction.
4. Service Upgrade Incentives: Create targeted campaigns that encourage customers to upgrade to higher-tier services, offering additional features or benefits to improve perceived value.
5. Localized Marketing: Develop geographically targeted marketing efforts to address specific regional churn trends, such as improved infrastructure or localized service offerings.