PROBLEM STATEMENT - Churn Prediction for a Telecommunication Brand .

In [53]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
```

In [54]:
```python
# Load the dataset
Telco_df = pd.read_csv('Telco-Customer-Churn.csv')
```

In [55]:
```python
print(Telco_df.head())  # Display the first few rows
```

```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No   
1  5575-GNVDE    Male              0      No         No      34          Yes   
2  3668-QPYBK    Male              0      No         No       2          Yes   
3  7795-CFOCW    Male              0      No         No      45           No   
4  9237-HQITU  Female              0      No         No       2          Yes   

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No   
1                No             DSL            Yes  ...              Yes   
2                No             DSL            Yes  ...               No   
3  No phone service             DSL            Yes  ...              Yes   
4                No     Fiber optic             No  ...               No   

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes   
1          No          No              No        One year               No   
2          No          No              No  Month-to-month              Yes   
3         Yes          No              No        One year               No   
4          No          No              No  Month-to-month              Yes   

               PaymentMethod MonthlyCharges  TotalCharges Churn  
0           Electronic check          29.85         29.85    No  
1               Mailed check          56.95        1889.5    No  
2               Mailed check          53.85        108.15   Yes  
3  Bank transfer (automatic)          42.30       1840.75    No  
4           Electronic check          70.70        151.65   Yes  

[5 rows x 21 columns]
```

In [56]:
```python
print (Telco_df.shape)
Telco_df.isnull().sum()
```

```
(7043, 21)
```

Out[56]:
```
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 [57]:
```python
print(Telco_df.info())  # Summary of the dataset
```

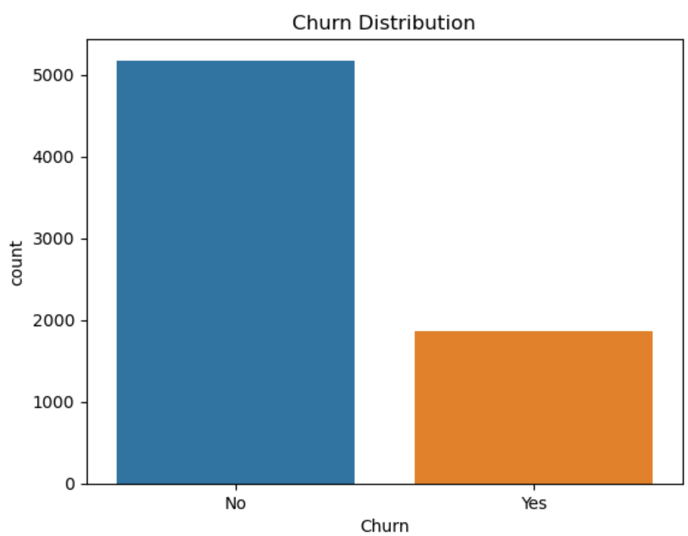
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   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
None
```

In [58]:
```python
# Check for duplicate rows
Telco_df.duplicated().sum()
```

Out[58]: 0

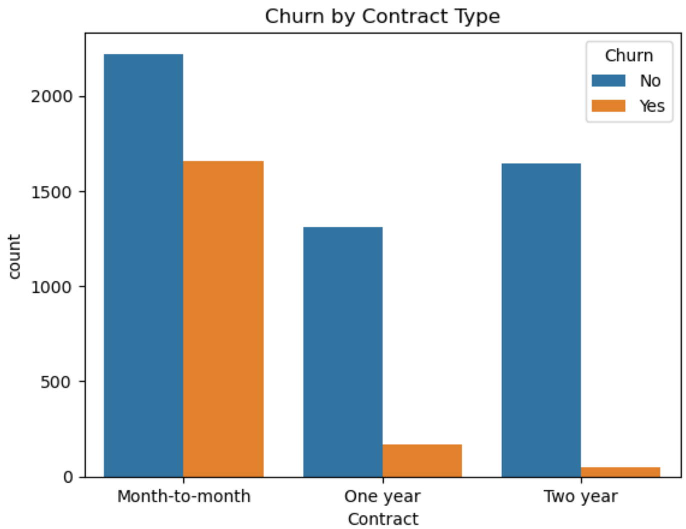
VISUALIZATION

In [59]:
```python
# Visualization 1: Churn distribution
sns.countplot(x='Churn', data=Telco_df)
plt.title('Churn Distribution')
plt.show()
```



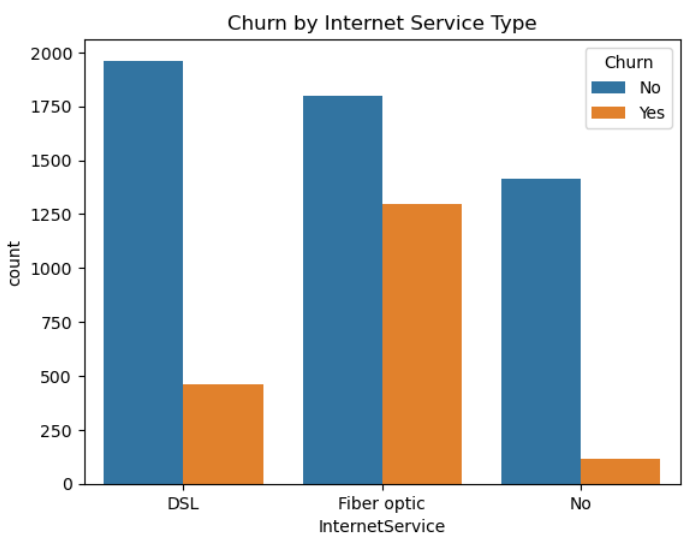
In [60]:
```
# Visualization 2: Churn by Contract type
sns.countplot(x='Contract', hue='Churn', data=Telco_df)
plt.title('Churn by Contract Type')
plt.show()
```

```
C:\Users\amara\anaconda3\lib\site-packages\seaborn\categorical.py:381: DeprecationWarning: distutils Version classes are deprecated. Use pack
aging.version instead.
  if LooseVersion(mpl.__version__) < "3.0":
C:\Users\amara\anaconda3\lib\site-packages\setuptools\_distutils\version.py:346: DeprecationWarning: distutils Version classes are deprecate
d. Use packaging.version instead.
  other = LooseVersion(other)
```



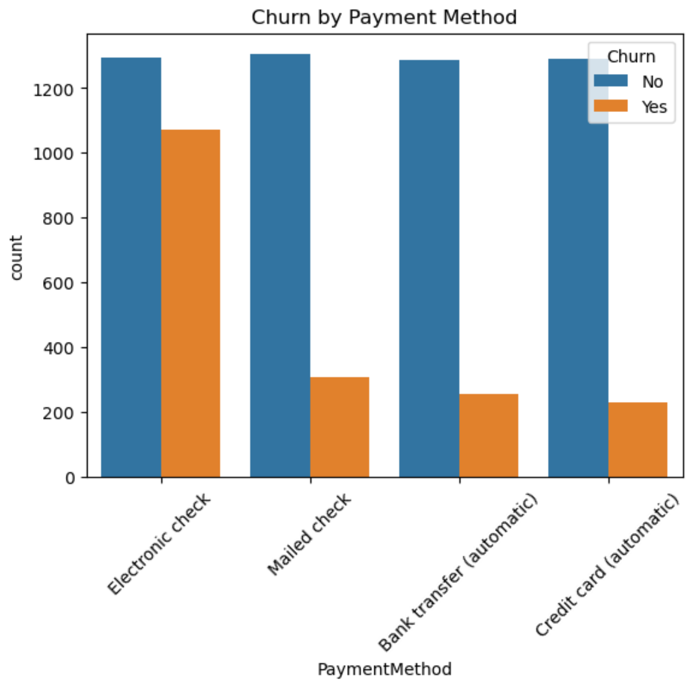
In [61]:
```
# Visualization 3: Churn by Internet Service type
sns.countplot(x='InternetService', hue='Churn', data=Telco_df)
plt.title('Churn by Internet Service Type')
plt.show()
```

```
C:\Users\amara\anaconda3\lib\site-packages\seaborn\categorical.py:381: DeprecationWarning: distutils Version classes are deprecated. Use pack
aging.version instead.
  if LooseVersion(mpl.__version__) < "3.0":
C:\Users\amara\anaconda3\lib\site-packages\setuptools\_distutils\version.py:346: DeprecationWarning: distutils Version classes are deprecate
d. Use packaging.version instead.
  other = LooseVersion(other)
```

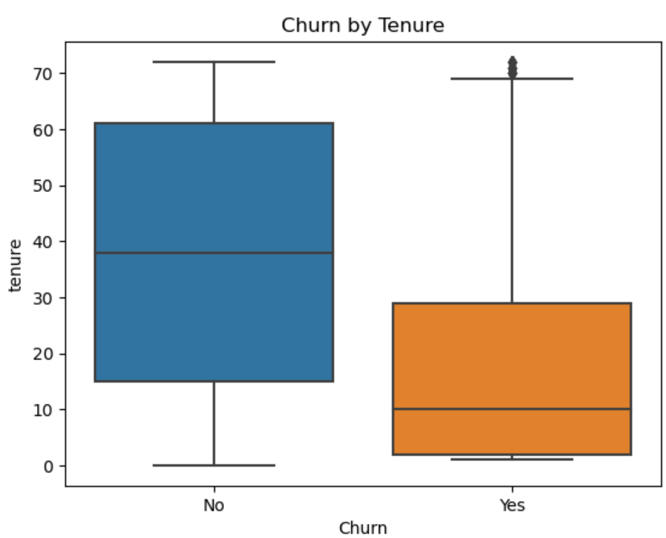


In [62]:
```python
# Visualization 4: Churn by Payment Method
sns.countplot(x='PaymentMethod', hue='Churn', data=Telco_df)
plt.title('Churn by Payment Method')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\amara\anaconda3\lib\site-packages\seaborn\categorical.py:381: DeprecationWarning: distutils Version classes are deprecated. Use pack
aging.version instead.
  if LooseVersion(mpl.__version__) < "3.0":
C:\Users\amara\anaconda3\lib\site-packages\setuptools\_distutils\version.py:346: DeprecationWarning: distutils Version classes are deprecate
d. Use packaging.version instead.
  other = LooseVersion(other)
```

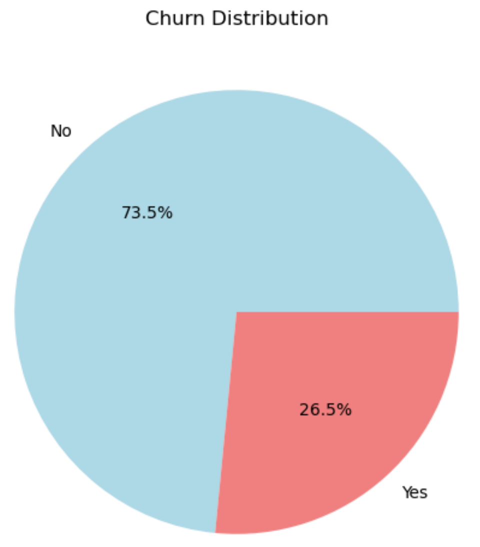


In [63]:
```python
# Visualization 5: Churn by Tenure
sns.boxplot(x='Churn', y='tenure', data=Telco_df)
plt.title('Churn by Tenure')
plt.show()
```

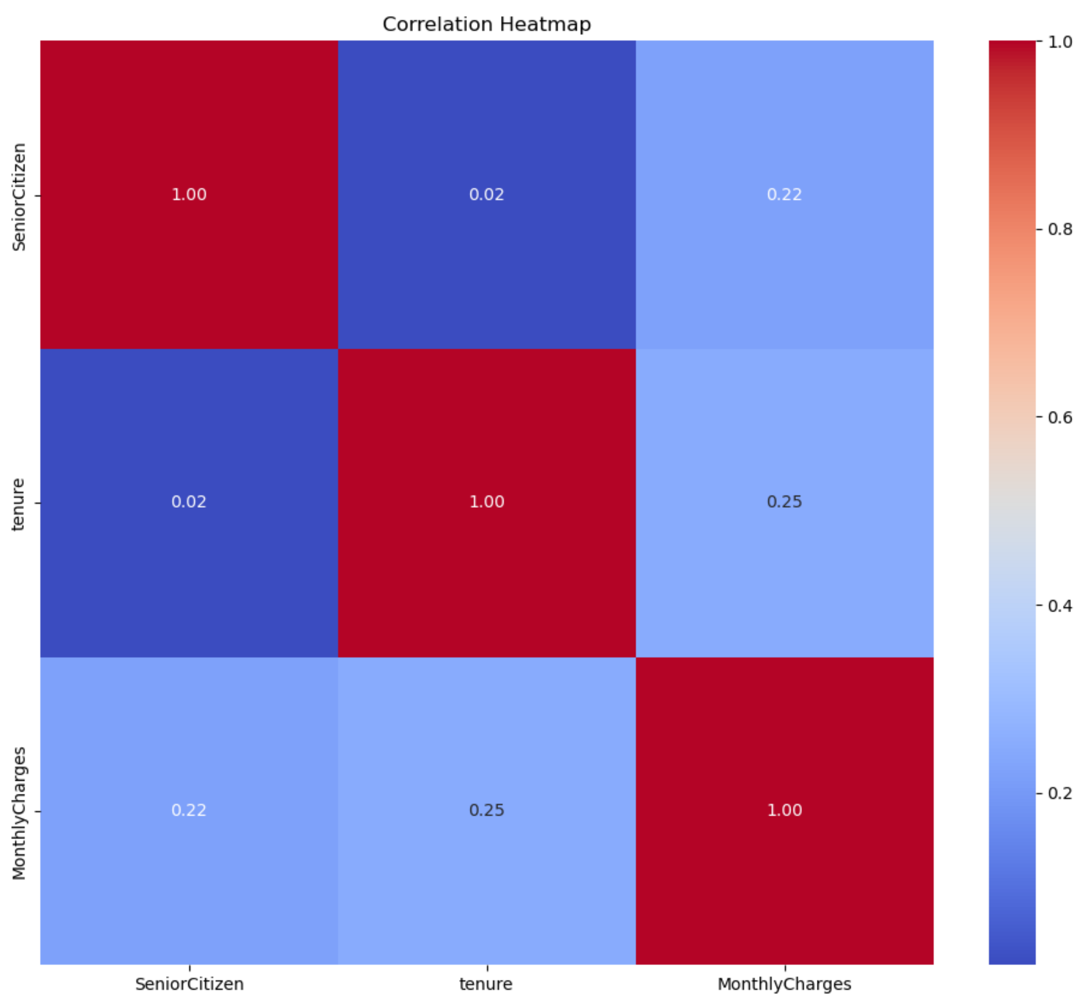


so we are not taking any action to handle the outliers,because a customer can be in the company for many months. Even though the customer had been in the company for a long time, we should consider them as part of our analysis.

In [64]:
```
# 6. Pie chart for churn distribution
plt.figure(figsize=(6, 6))
plt.pie(Telco_df['Churn'].value_counts(), labels=['No', 'Yes'], autopct='%1.1f%%', colors=['lightblue', 'lightcoral'])
plt.title('Churn Distribution')
plt.show()
```



In [65]:
```
# Visualization 7: Correlation Heatmap
correlation_matrix = Telco_df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



DATA CLEANING

In [66]:
```
# Assuming 'No internet service' means the customer does not have that service, we can fill those with 'No'
cols_fillna = ['MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
               'StreamingTV', 'StreamingMovies']
Telco_df[cols_fillna] = Telco_df[cols_fillna].replace('No internet service', 'No')
```

In [67]:
```
# Assuming 'No internet service' means the customer does not have that service, we can fill those with 'No'
cols_fillna = ['MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
               'StreamingTV', 'StreamingMovies']
Telco_df[cols_fillna] = Telco_df[cols_fillna].replace('No phone service', 'No')
```

In [68]:
```
Telco_df
```

Out[68]:

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

7043 rows × 21 columns

In [69]:
```
# Drop irrelevant columns like customerID as it does not contribute to the prediction
Telco_df.drop(columns=['customerID'], inplace=True)
```

Feature Engineering

In [70]:
```
# Convert 'Churn' column to binary values
Telco_df['Churn'] = Telco_df['Churn'].map({'Yes': 1, 'No': 0})
```

In [71]:
```
# Create binary features for 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling'
Telco_df['Partner'] = Telco_df['Partner'].map({'Yes': 1, 'No': 0})
Telco_df['Dependents'] = Telco_df['Dependents'].map({'Yes': 1, 'No': 0})
Telco_df['PhoneService'] = Telco_df['PhoneService'].map({'Yes': 1, 'No': 0})
Telco_df['PaperlessBilling'] = Telco_df['PaperlessBilling'].map({'Yes': 1, 'No': 0})
```

In [72]:
```python
Telco_df
```

Out[72]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Streaming
0	Female	0	1	0	1	0	No	DSL	No	Yes	No	No	
1	Male	0	0	0	34	1	No	DSL	Yes	No	Yes	No	
2	Male	0	0	0	2	1	No	DSL	Yes	Yes	No	No	
3	Male	0	0	0	45	0	No	DSL	Yes	No	Yes	Yes	
4	Female	0	0	0	2	1	No	Fiber optic	No	No	No	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	Male	0	1	1	24	1	Yes	DSL	Yes	No	Yes	Yes	Y
7039	Female	0	1	1	72	1	Yes	Fiber optic	No	Yes	Yes	No	Y
7040	Female	0	1	1	11	0	No	DSL	Yes	No	No	No	
7041	Male	1	1	0	4	1	Yes	Fiber optic	No	No	No	No	
7042	Male	0	0	0	66	1	No	Fiber optic	Yes	No	Yes	Yes	Y

7043 rows × 20 columns

In [73]:
```python
Telco_df
```

Out[73]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Streaming
0	Female	0	1	0	1	0	No	DSL	No	Yes	No	No	
1	Male	0	0	0	34	1	No	DSL	Yes	No	Yes	No	
2	Male	0	0	0	2	1	No	DSL	Yes	Yes	No	No	
3	Male	0	0	0	45	0	No	DSL	Yes	No	Yes	Yes	
4	Female	0	0	0	2	1	No	Fiber optic	No	No	No	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	Male	0	1	1	24	1	Yes	DSL	Yes	No	Yes	Yes	Y
7039	Female	0	1	1	72	1	Yes	Fiber optic	No	Yes	Yes	No	Y
7040	Female	0	1	1	11	0	No	DSL	Yes	No	No	No	
7041	Male	1	1	0	4	1	Yes	Fiber optic	No	No	No	No	
7042	Male	0	0	0	66	1	No	Fiber optic	Yes	No	Yes	Yes	Y

7043 rows × 20 columns

In [74]:
```python
Telco_df = pd.get_dummies(Telco_df, columns=['gender', 'MultipleLines', 'InternetService',
                                             'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
                                             'TechSupport', 'StreamingTV', 'StreamingMovies',
                                             'Contract'], drop_first=True)
```

In [75]:
```
Telco_df
```
Out[75]:

	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn	...	InternetService_Fiber optic	InternetServi
0	0	1	0	1	0	1	Electronic check	29.85	29.85	0	...	0	
1	0	0	0	34	1	0	Mailed check	56.95	1889.5	0	...	0	
2	0	0	0	2	1	1	Mailed check	53.85	108.15	1	...	0	
3	0	0	0	45	0	0	Bank transfer (automatic)	42.30	1840.75	0	...	0	
4	0	0	0	2	1	1	Electronic check	70.70	151.65	1	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	0	1	1	24	1	1	Mailed check	84.80	1990.5	0	...	0	
7039	0	1	1	72	1	1	Credit card (automatic)	103.20	7362.9	0	...	1	
7040	0	1	1	11	0	1	Electronic check	29.60	346.45	0	...	0	
7041	1	1	0	4	1	1	Mailed check	74.40	306.6	1	...	1	
7042	0	0	0	66	1	1	Bank transfer (automatic)	105.65	6844.5	0	...	1	

7043 rows × 22 columns

In [76]:
```
# Convert TotalCharges to numeric
Telco_df['TotalCharges'] = pd.to_numeric(Telco_df['TotalCharges'], errors='coerce')
```
In [77]:
```
#calculate the ratio of MonthlyCharges to TotalCharges to see the average monthly spending:
Telco_df['AvgMonthlySpending'] = Telco_df['TotalCharges'] / Telco_df['tenure']
```
In [78]:
```
Telco_df
```
Out[78]:

	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn	...	InternetService_No	OnlineSecurity_
0	0	1	0	1	0	1	Electronic check	29.85	29.85	0	...	0	
1	0	0	0	34	1	0	Mailed check	56.95	1889.50	0	...	0	
2	0	0	0	2	1	1	Mailed check	53.85	108.15	1	...	0	
3	0	0	0	45	0	0	Bank transfer (automatic)	42.30	1840.75	0	...	0	
4	0	0	0	2	1	1	Electronic check	70.70	151.65	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	0	1	1	24	1	1	Mailed check	84.80	1990.50	0	...	0	
7039	0	1	1	72	1	1	Credit card (automatic)	103.20	7362.90	0	...	0	
7040	0	1	1	11	0	1	Electronic check	29.60	346.45	0	...	0	
7041	1	1	0	4	1	1	Mailed check	74.40	306.60	1	...	0	
7042	0	0	0	66	1	1	Bank transfer (automatic)	105.65	6844.50	0	...	0	

7043 rows × 23 columns

Feature Scaling

In [79]:
```
# Feature Scaling (if required)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
Telco_df[numerical_features] = scaler.fit_transform(Telco_df[numerical_features])
```

In [80]:
```
Telco_df
```

Out[80]:

	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn	...	InternetService_No	OnlineSecur
0	0	1	0	-1.277445	0	1	Electronic check	-1.160323	-0.994194	0	...	0	
1	0	0	0	0.066327	1	0	Mailed check	-0.259629	-0.173740	0	...	0	
2	0	0	0	-1.236724	1	1	Mailed check	-0.362660	-0.959649	1	...	0	
3	0	0	0	0.514251	0	0	Bank transfer (automatic)	-0.746535	-0.195248	0	...	0	
4	0	0	0	-1.236724	1	1	Electronic check	0.197365	-0.940457	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	0	1	1	-0.340876	1	1	Mailed check	0.665992	-0.129180	0	...	0	
7039	0	1	1	1.613701	1	1	Credit card (automatic)	1.277533	2.241056	0	...	0	
7040	0	1	1	-0.870241	0	1	Electronic check	-1.168632	-0.854514	0	...	0	
7041	1	1	0	-1.155283	1	1	Mailed check	0.320338	-0.872095	1	...	0	
7042	0	0	0	1.369379	1	1	Bank transfer (automatic)	1.358961	2.012344	0	...	0	

7043 rows × 23 columns

In [81]:
```
# Check for missing values
Telco_df.isnull().sum()
```

Out[81]:
```
SeniorCitizen                0
Partner                      0
Dependents                   0
tenure                       0
PhoneService                 0
PaperlessBilling             0
PaymentMethod                0
MonthlyCharges               0
TotalCharges                11
Churn                        0
gender_Male                  0
MultipleLines_Yes            0
InternetService_Fiber optic  0
InternetService_No           0
OnlineSecurity_Yes           0
OnlineBackup_Yes             0
DeviceProtection_Yes         0
TechSupport_Yes              0
StreamingTV_Yes              0
StreamingMovies_Yes          0
Contract_One year            0
Contract_Two year            0
AvgMonthlySpending          11
dtype: int64
```

DEALING WITH MISSING VALUES

In [82]:
```
total_charges_mean = Telco_df['TotalCharges'].mean()
#Replace the missing values in the 'TotalCharges' column with the calculated mean
Telco_df['TotalCharges'].fillna(total_charges_mean, inplace=True)
```

In [83]:
```
#dealing with the missing values using the mean
avg_monthly_spending_mean = Telco_df['AvgMonthlySpending'].mean()
#Replace the missing values in the 'AvgMonthlySpending' column with the calculated mean
Telco_df['AvgMonthlySpending'].fillna(avg_monthly_spending_mean, inplace=True)
```

In [84]:
```
Telco_df.isnull().sum()
```

Out[84]:
```
SeniorCitizen                  0
Partner                        0
Dependents                     0
tenure                         0
PhoneService                   0
PaperlessBilling               0
PaymentMethod                  0
MonthlyCharges                 0
TotalCharges                   0
Churn                          0
gender_Male                    0
MultipleLines_Yes              0
InternetService_Fiber optic    0
InternetService_No             0
OnlineSecurity_Yes             0
OnlineBackup_Yes               0
DeviceProtection_Yes           0
TechSupport_Yes                0
StreamingTV_Yes                0
StreamingMovies_Yes            0
Contract_One year              0
Contract_Two year              0
AvgMonthlySpending             0
dtype: int64
```

VALIDATION SPLIT

In [85]:
```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from dmba import classificationSummary
```

In [86]:
```
# Split the data into features (X) and target (y)
X = Telco_df[['SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'gende
y = Telco_df['Churn']
```

In [87]:
```
# Splitting the dataset into training and testing sets
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.2, random_state=42)
```

NEURAL NETWORK

In [88]:
```
clf = MLPClassifier(hidden_layer_sizes=(200, 100), activation='logistic', solver='adam', max_iter=2000, batch_size=256)
```

In [89]:
```
clf.fit(train_X, train_y.values)
```

Out[89]:
```
MLPClassifier(activation='logistic', batch_size=256,
              hidden_layer_sizes=(200, 100), max_iter=2000)
```

In [90]:
```
y_pred_train = clf.predict(train_X)
y_pred_valid = clf.predict(valid_X)
```

In [91]:
```
# Accuracy, Precision, Recall, and F1-score
print('Training Performance:')
print('Accuracy:', accuracy_score(train_y, y_pred_train))
print('Precision:', precision_score(train_y, y_pred_train))
print('Recall:', recall_score(train_y, y_pred_train))
print('F1-score:', f1_score(train_y, y_pred_train))

print('Validation Performance:')
print('Accuracy:', accuracy_score(valid_y, y_pred_valid))
print('Precision:', precision_score(valid_y, y_pred_valid))
print('Recall:', recall_score(valid_y, y_pred_valid))
print('F1-score:', f1_score(valid_y, y_pred_valid))
```

```
Training Performance:
Accuracy: 0.7992545260915868
Precision: 0.6733143399810066
Recall: 0.47393048128342247
F1-score: 0.5562965868968224
Validation Performance:
Accuracy: 0.8133427963094393
Precision: 0.7022058823529411
Recall: 0.5120643431635389
F1-score: 0.5922480620155038
```

RANDOM FOREST

In [92]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [93]:
```
rf = RandomForestClassifier(random_state=1)
rf.fit(train_X, train_y)
```

Out[93]:
```
RandomForestClassifier(random_state=1)
```

In [94]:
```
train_pred = rf.predict(train_X)
valid_pred = rf.predict(valid_X)
```

In [95]:
```
# Calculate Metrics for Training Data
train_cm = confusion_matrix(train_y, train_pred)
train_accuracy = accuracy_score(train_y, train_pred)
train_precision = precision_score(train_y, train_pred)
train_recall = recall_score(train_y, train_pred)
train_f1_score = f1_score(train_y, train_pred)
```

In [96]:
```
# Calculate Metrics for Validation Data
valid_cm = confusion_matrix(valid_y, valid_pred)
valid_accuracy = accuracy_score(valid_y, valid_pred)
valid_precision = precision_score(valid_y, valid_pred)
valid_recall = recall_score(valid_y, valid_pred)
valid_f1_score = f1_score(valid_y, valid_pred)
```

In [97]:
```
# Print the Metrics
print("Random Forest Metrics:")
print("Training Accuracy:", train_accuracy)
print("Training Precision:", train_precision)
print("Training Recall:", train_recall)
print("Training F1 Score:", train_f1_score)
print("\nValidation Accuracy:", valid_accuracy)
print("Validation Precision:", valid_precision)
print("Validation Recall:", valid_recall)
print("Validation F1 Score:", valid_f1_score)
```

```
Random Forest Metrics:
Training Accuracy: 0.9978700745473909
Training Precision: 0.9986559139784946
Training Recall: 0.9933155080213903
Training F1 Score: 0.9959785522788204

Validation Accuracy: 0.7977288857345636
Validation Precision: 0.6617647058823529
Validation Recall: 0.48257372654155495
Validation F1 Score: 0.5581395348837209
```

ASSOCIATION RULE -APRIORI ALGORITHM

In [98]:
```
Telco_df
```

Out[98]:

	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn	...	InternetService_No	OnlineSecur
0	0	1	0	-1.277445	0	1	Electronic check	-1.160323	-0.994194	0	...	0	
1	0	0	0	0.066327	1	0	Mailed check	-0.259629	-0.173740	0	...	0	
2	0	0	0	-1.236724	1	1	Mailed check	-0.362660	-0.959649	1	...	0	
3	0	0	0	0.514251	0	0	Bank transfer (automatic)	-0.746535	-0.195248	0	...	0	
4	0	0	0	-1.236724	1	1	Electronic check	0.197365	-0.940457	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	0	1	1	-0.340876	1	1	Mailed check	0.665992	-0.129180	0	...	0	
7039	0	1	1	1.613701	1	1	Credit card (automatic)	1.277533	2.241056	0	...	0	
7040	0	1	1	-0.870241	0	1	Electronic check	-1.168632	-0.854514	0	...	0	
7041	1	1	0	-1.155283	1	1	Mailed check	0.320338	-0.872095	1	...	0	
7042	0	0	0	1.369379	1	1	Bank transfer (automatic)	1.358961	2.012344	0	...	0	

7043 rows × 23 columns

In [99]:

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Assuming you have already split the data into features (X) and target (y)
X = Telco_df[['SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'gender_Male', 'MultipleLines_Yes', 'InternetServi
y = Telco_df['Churn']

# Encode categorical variables as binary or dummy variables
X = pd.get_dummies(X, drop_first=True)

# Perform association rule mining using Apriori algorithm
frequent_itemsets = apriori(X, min_support=0.05, use_colnames=True)

# Generate association rules with specified metrics
association_results = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Display the association rules
print(association_results)
```

```
C:\Users\amara\anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types resu
lt in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

                        antecedents  \
0                         (Partner)
1                   (SeniorCitizen)
2                    (PhoneService)
3                   (SeniorCitizen)
4                   (SeniorCitizen)
...                             ...
33915           (StreamingMovies_Yes)
33916               (StreamingTV_Yes)
33917                  (PhoneService)
33918  (InternetService_Fiber optic)
33919             (OnlineBackup_Yes)

                                             consequents  antecedent support  \
0                                        (SeniorCitizen)            0.483033
1                                              (Partner)            0.162147
2                                        (SeniorCitizen)            0.903166
3                                         (PhoneService)            0.162147
4                                     (PaperlessBilling)            0.162147
...                                                  ...                 ...
33915  (DeviceProtection_Yes, MultipleLines_Yes, Inte...            0.387903
33916  (DeviceProtection_Yes, MultipleLines_Yes, Stre...            0.384353
33917  (DeviceProtection_Yes, MultipleLines_Yes, Stre...            0.903166
33918  (DeviceProtection_Yes, MultipleLines_Yes, Stre...            0.439585
33919  (DeviceProtection_Yes, MultipleLines_Yes, Stre...            0.344881

       consequent support   support  confidence      lift  leverage  \
0                0.162147  0.081357    0.168430  1.038752  0.003035
1                0.483033  0.081357    0.501751  1.038752  0.003035
2                0.162147  0.147380    0.163182  1.006384  0.000935
3                0.903166  0.147380    0.908932  1.006384  0.000935
4                0.592219  0.124379    0.767075  1.295256  0.028352
...                   ...       ...         ...       ...       ...
33915            0.065029  0.056510    0.145681  2.240240  0.031285
33916            0.064603  0.056510    0.147026  2.275837  0.031680
33917            0.056510  0.056510    0.062569  1.107216  0.005472
33918            0.075678  0.056510    0.128553  1.698684  0.023243
33919            0.089025  0.056510    0.163853  1.840542  0.025807

       conviction  zhangs_metric
0        1.007556       0.072164
1        1.037569       0.044526
2        1.001237       0.065505
3        1.063309       0.007571
4        1.750698       0.272066
...           ...            ...
33915    1.094405       0.904463
33916    1.096630       0.910589
33917    1.006463       1.000000
33918    1.060675       0.733937
33919    1.089493       0.697098

[33920 rows x 10 columns]
```

```
In [100]: association_results
```

Out[100]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Partner)	(SeniorCitizen)	0.483033	0.162147	0.081357	0.168430	1.038752	0.003035	1.007556	0.072164
1	(SeniorCitizen)	(Partner)	0.162147	0.483033	0.081357	0.501751	1.038752	0.003035	1.037569	0.044526
2	(PhoneService)	(SeniorCitizen)	0.903166	0.162147	0.147380	0.163182	1.006384	0.000935	1.001237	0.065505
3	(SeniorCitizen)	(PhoneService)	0.162147	0.903166	0.147380	0.908932	1.006384	0.000935	1.063309	0.007571
4	(SeniorCitizen)	(PaperlessBilling)	0.162147	0.592219	0.124379	0.767075	1.295256	0.028352	1.750698	0.272066
...	...	...	...	...	...	...	...	...	...	...
33915	(StreamingMovies_Yes)	(DeviceProtection_Yes, MultipleLines_Yes, Inte...	0.387903	0.065029	0.056510	0.145681	2.240240	0.031285	1.094405	0.904463
33916	(StreamingTV_Yes)	(DeviceProtection_Yes, MultipleLines_Yes, Stre...	0.384353	0.064603	0.056510	0.147026	2.275837	0.031680	1.096630	0.910589
33917	(PhoneService)	(DeviceProtection_Yes, MultipleLines_Yes, Stre...	0.903166	0.056510	0.056510	0.062569	1.107216	0.005472	1.006463	1.000000
33918	(InternetService_Fiber optic)	(DeviceProtection_Yes, MultipleLines_Yes, Stre...	0.439585	0.075678	0.056510	0.128553	1.698684	0.023243	1.060675	0.733937
33919	(OnlineBackup_Yes)	(DeviceProtection_Yes, MultipleLines_Yes, Stre...	0.344881	0.089025	0.056510	0.163853	1.840542	0.025807	1.089493	0.697098

33920 rows × 10 columns

In [101]:

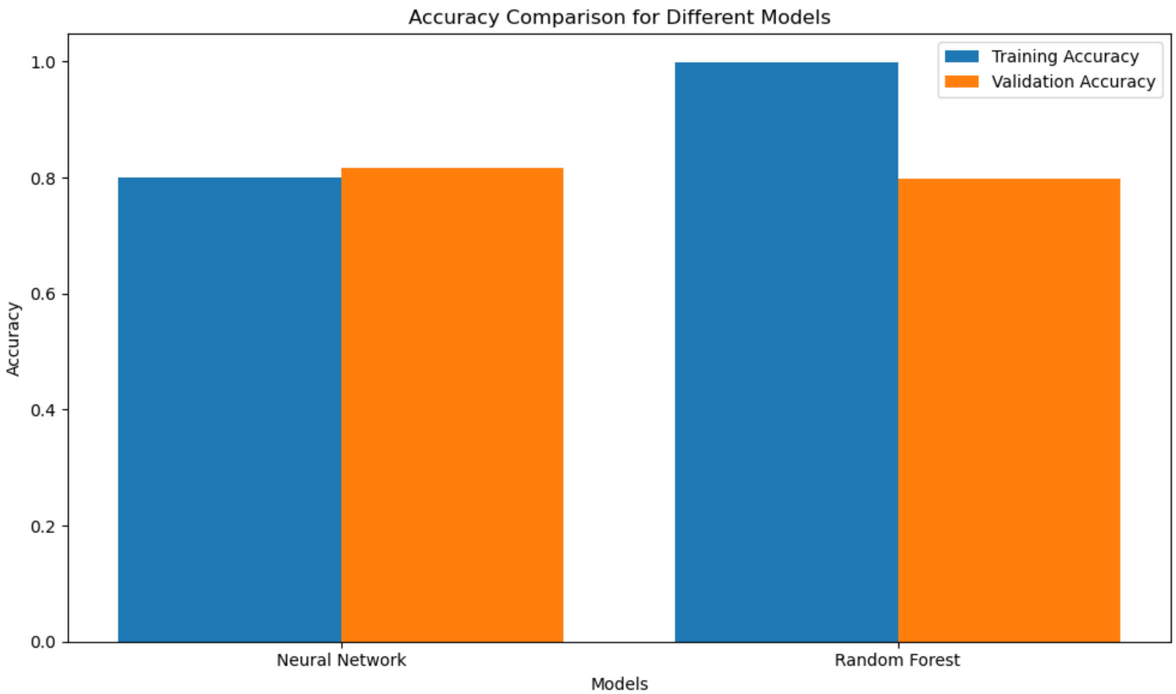
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Data and model initialization for each model
models = ['Neural Network', 'Random Forest']
accuracy_scores_train = []
accuracy_scores_valid = []

# Neural Network
clf = MLPClassifier(hidden_layer_sizes=(200, 100), activation='logistic', solver='adam', max_iter=2000, batch_size=256)
clf.fit(train_X, train_y.values)
y_pred_train_nn = clf.predict(train_X)
y_pred_valid_nn = clf.predict(valid_X)
accuracy_scores_train.append(accuracy_score(train_y, y_pred_train_nn))
accuracy_scores_valid.append(accuracy_score(valid_y, y_pred_valid_nn))

# Random Forest
rf = RandomForestClassifier(random_state=1)
rf.fit(train_X, train_y)
y_pred_train_rf = rf.predict(train_X)
y_pred_valid_rf = rf.predict(valid_X)
accuracy_scores_train.append(accuracy_score(train_y, y_pred_train_rf))
accuracy_scores_valid.append(accuracy_score(valid_y, y_pred_valid_rf))

# Create the bar chart
plt.figure(figsize=(10, 6))
plt.bar(np.arange(len(models)) - 0.2, accuracy_scores_train, width=0.4, label='Training Accuracy')
plt.bar(np.arange(len(models)) + 0.2, accuracy_scores_valid, width=0.4, label='Validation Accuracy')
plt.xticks(np.arange(len(models)), models)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison for Different Models')
plt.legend()
plt.tight_layout()
plt.show()
```



In [ ]: