In [49]:
```
# First, let's start by importing the necessary libraries and loading the dataset:
```

In [50]:
```
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
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier

# load dataset
df = pd.read_csv("telco.csv")
```

In [51]:
```
# Next, let's take a look at the data to get a better understanding of its structure and format:
# view first five rows of the dataset
df.head()

# view summary statistics of the dataset
df.describe()

# view column data types and null values
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 [52]:
```
# Based on the above analysis, we can see that the dataset contains both numerical and categorical variables,
# and there are no missing values. Now, let's preprocess the data by encoding categorical variables and
# scaling numerical variables:
```

In [53]:
```
# The TotalCharges column in the DataFrame contains missing values represented as empty strings, which cannot b
# processed by the StandardScaler function. To fix this, we need to drop the TotalCharges column.
```

In [54]:
```
df = df.drop("TotalCharges", axis=1)
```

In [55]:
```
df.head()
```

Out[55]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBacku
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Ye
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Ye
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	N
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	N

In [56]:
```
# encode categorical variables
cat_cols = ["gender", "Partner", "Dependents", "PhoneService", "MultipleLines", "InternetService",
            "OnlineSecurity", "OnlineBackup", "DeviceProtection", "TechSupport", "StreamingTV",
            "StreamingMovies", "Contract", "PaperlessBilling", "PaymentMethod"]
```

```
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# scale numerical variables
num_cols = ["tenure", "MonthlyCharges"]
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])

# create a new column 'Churn_Yes' based on the values in the 'Churn' column
df['Churn_Yes'] = (df['Churn'] == 'Yes').astype(int)
```

In [57]:
```
df.head()
```

Out[57]:

	customerID	SeniorCitizen	tenure	MonthlyCharges	Churn	gender_Male	Partner_Yes	Dependents_Yes	PhoneService_Yes	MultipleLine phone se
0	7590-VHVEG	0	-1.277445	-1.160323	No	0	1	0	0	
1	5575-GNVDE	0	0.066327	-0.259629	No	1	0	0	1	
2	3668-QPYBK	0	-1.236724	-0.362660	Yes	1	0	0	1	
3	7795-CFOCW	0	0.514251	-0.746535	No	1	0	0	0	
4	9237-HQITU	0	-1.236724	0.197365	Yes	0	0	0	1	

5 rows × 32 columns

In [58]:
```
# Now that the data is preprocessed, we can split it into training and testing sets:
from sklearn.preprocessing import OneHotEncoder
```

In [59]:
```
# split data into X (features) and y (target)
X = df.drop(["Churn_Yes"], axis=1)
y = df["Churn_Yes"]

# convert categorical features to numeric using one-hot encoding
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
encoder = OneHotEncoder(drop='first', sparse=False)
encoded_cols = encoder.fit_transform(X[categorical_cols])
encoded_cols_df = pd.DataFrame(encoded_cols, columns=encoder.get_feature_names_out(categorical_cols))
X.drop(categorical_cols, axis=1, inplace=True)
X = pd.concat([X, encoded_cols_df], axis=1)

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
C:\Users\Gidama1\AppData\Local\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:828: FutureWarnin
g: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignore
d unless you leave `sparse` to its default value.
  warnings.warn(
```

In [60]:
```
# Next, let's train a Random Forest classifier on the training set:
```

In [61]:
```
# train Random Forest classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)
rfc.fit(X_train, y_train)
```

Out[61]:
```
RandomForestClassifier(random_state=42)
```

In [62]:
```
# Finally, let's evaluate the performance of the classifier on the testing set:
```

In [63]:
```
# predict churn on testing set
y_pred = rfc.predict(X_test)

# evaluate classifier performance
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification report:\n", classification_report(y_test, y_pred))
```

```
Accuracy score: 0.9950319375443577
Confusion matrix:
 [[1036    0]
 [   7  366]]
Classification report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1036
           1       1.00      0.98      0.99       373

    accuracy                           1.00      1409
   macro avg       1.00      0.99      0.99      1409
weighted avg       1.00      1.00      1.00      1409
```

In [64]:
```
# The code above evaluates the performance of a random forest classifier on the testing set.
# The accuracy score is 0.995, which means that the classifier predicted the correct churn
# status for 99.5% of the customers in the testing set. The confusion matrix shows the number of true positives
# false positives (FP), false negatives (FN), and true negatives (TN) predictions. In this case,
# the confusion matrix shows that the classifier correctly predicted 1036 true negatives and 366 true positives
# with 7 false negatives and 0 false positives.
```

In [65]:
```
results_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results_df.head(10))
```

```
      Actual  Predicted
185        1          1
2715       0          0
3825       0          0
1807       1          1
132        0          0
1263       1          1
3732       0          0
1672       0          0
811        1          1
2526       1          1
```

In [ ]: