Telecom Churn Prediction and Analysis

Objective:

The goal of this notebook is to explore and analyze the Telecom Churn dataset to understand factors contributing to customer churn and to develop a predictive model that can forecast customer churn with high accuracy.

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import pandas as pd
import numpy as np
```

Dataset Description:

The Telecom Churn dataset comprises customer data from a telecom company. Key features include customer account information, demographic data, service usage, and churn status (Yes or No).

Data Fields:

- CustomerID: Unique identifier for the customer
- Gender: Customer gender (Male, Female)
- Age: Customer age
- Tenure: Number of months the customer has stayed with the company
- ServiceCalls: Number of customer service calls made
- MonthlyCharges: The amount charged to the customer monthly
- TotalCharges: The total amount charged to the customer
- Churn: Customer churn status (Yes or No)

```
data = pd.read_csv("/content/Telecom_Customer_Churn_Dataset.csv")
data.head(10)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
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	
5	9305-CDSKC	Female	0	No	No	8	Yes	
6	1452-KIOVK	Male	0	No	Yes	22	Yes	
7	6713-OKOMC	Female	0	No	No	10	No	
8	7892-POOKP	Female	0	Yes	No	28	Yes	
9	6388-TABGU	Male	0	No	Yes	62	Yes	

10 rows × 21 columns

Data Preprocessing

```
data.shape
```

```
(7043, 21)
```

```
data.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      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

```
print(data['SeniorCitizen'].nunique())
data['SeniorCitizen'].unique()
```

```
2
array([0, 1])
```

```
print(data['Contract'].nunique())
data['Contract'].unique()
```

```
3
array(['Month-to-month', 'One year', 'Two year'], dtype=object)
```

```
print(data['PaymentMethod'].nunique())
data['PaymentMethod'].unique()
```

```
4
array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
       'Credit card (automatic)'], dtype=object)
```

```
print(data['Partner'].nunique())
data['Partner'].unique()
```

```
2
array(['Yes', 'No'], dtype=object)
```

```
print(data['DeviceProtection'].nunique())
data['DeviceProtection'].unique()
```

```
3
array(['No', 'Yes', 'No internet service'], dtype=object)
```

```
print(data['PhoneService'].nunique())
data['PhoneService'].unique()
```

```
2
array(['No', 'Yes'], dtype=object)
```

```
print(data['Churn'].nunique())
data['Churn'].unique()
```

```
2
array(['No', 'Yes'], dtype=object)
```

```
data.columns
```

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

```
data.isna().sum()
```

```
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        11
Churn                0
dtype: int64
```

There are 11 null values in `TotalCharges` column which can be replaced with mean or median of the data column.

```
data.describe()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7032.000000
mean	0.162147	32.371149	64.761692	2283.300441
std	0.368612	24.559481	30.090047	2266.771362
min	0.000000	0.000000	18.250000	18.800000
25%	0.000000	9.000000	35.500000	401.450000
50%	0.000000	29.000000	70.350000	1397.475000
75%	0.000000	55.000000	89.850000	3794.737500
max	1.000000	72.000000	118.750000	8684.800000

```
# Calculate the mean of the column
mid = data['TotalCharges'].median()

# Replace missing values with the mean
data['TotalCharges'].fillna(mid, inplace=True)
```

```
data.isna().sum()
```

```
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
```

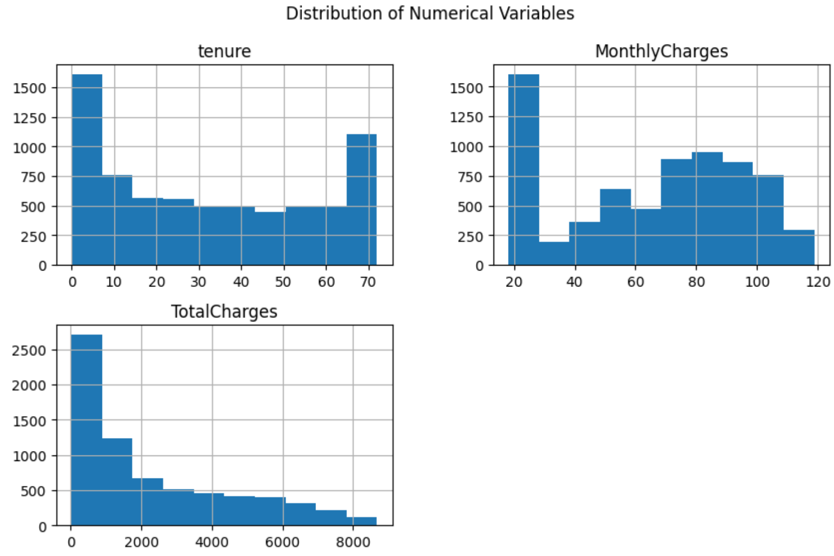
```
data.duplicated().sum()
```

```
0
```

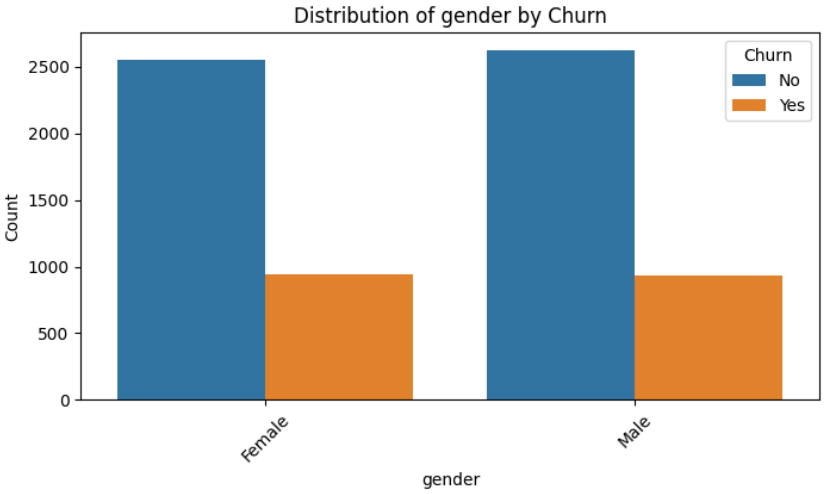
The dataset has no dublicates.

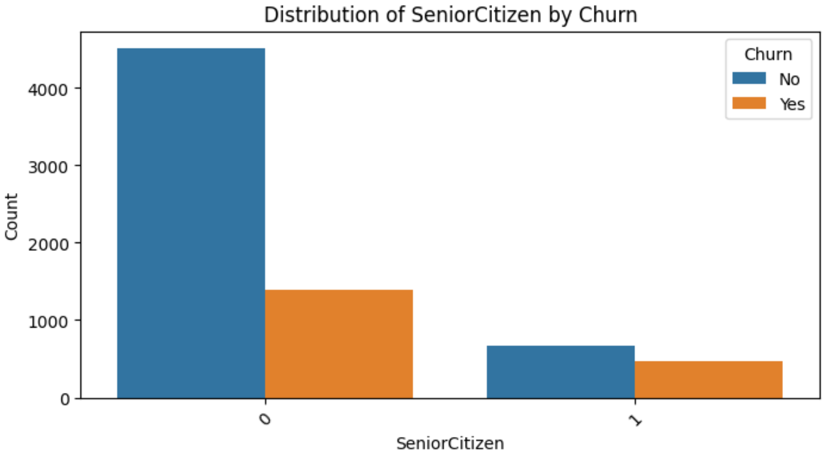
Exploratory Data Analysis

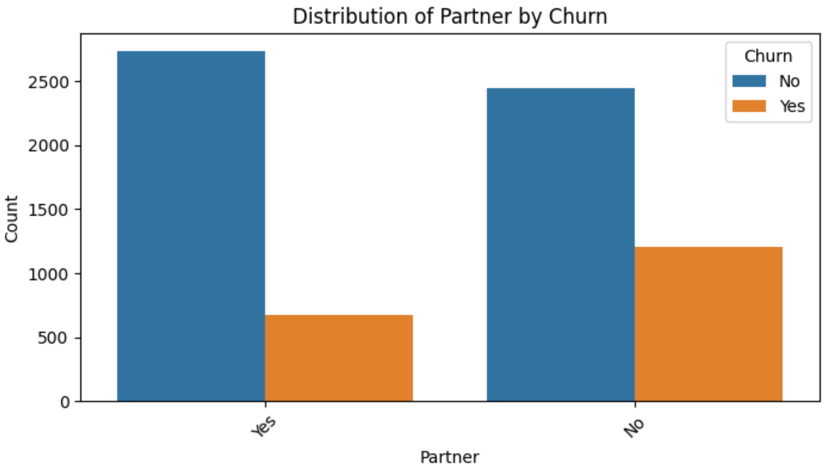
```
# Distribution of key variables
# Numerical variables
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
data[num_cols].hist(figsize=(10, 6))
plt.suptitle('Distribution of Numerical Variables')
plt.show()
```

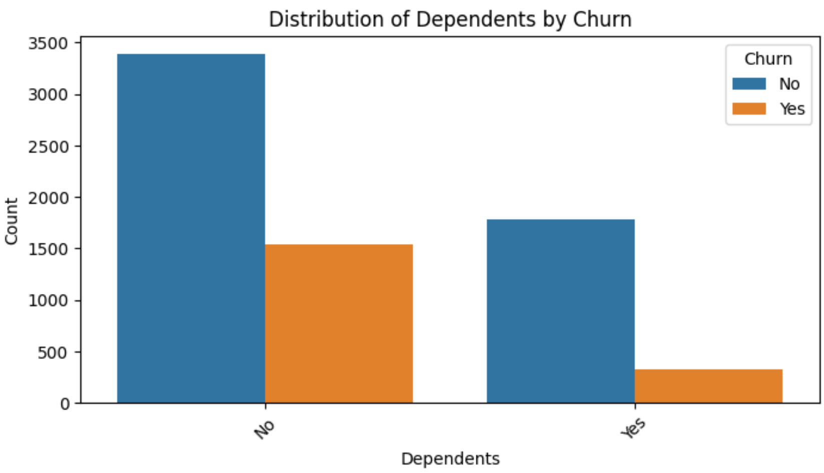


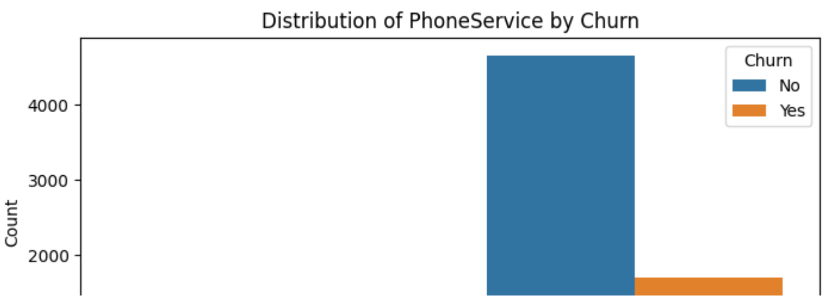

```
# Categorical variables
cat_cols = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
            'InternetService', 'OnlineSecurity', 'DeviceProtection', 'TechSupport', 'StreamingTV',
            'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn']

for col in cat_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=data, x=col, hue='Churn')
    plt.title(f'Distribution of {col} by Churn')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.legend(title='Churn', loc='upper right')
    plt.show()
```

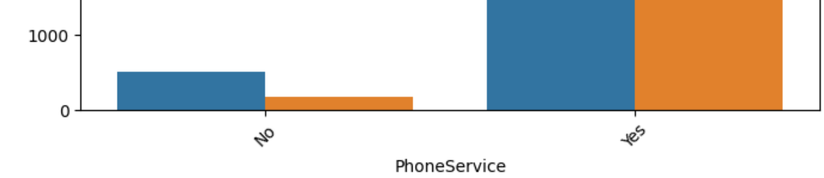

Distribution of gender by Churn
Churn
No
Yes
Count
2500
2000
1500
1000
500
0
Female
Male
gender

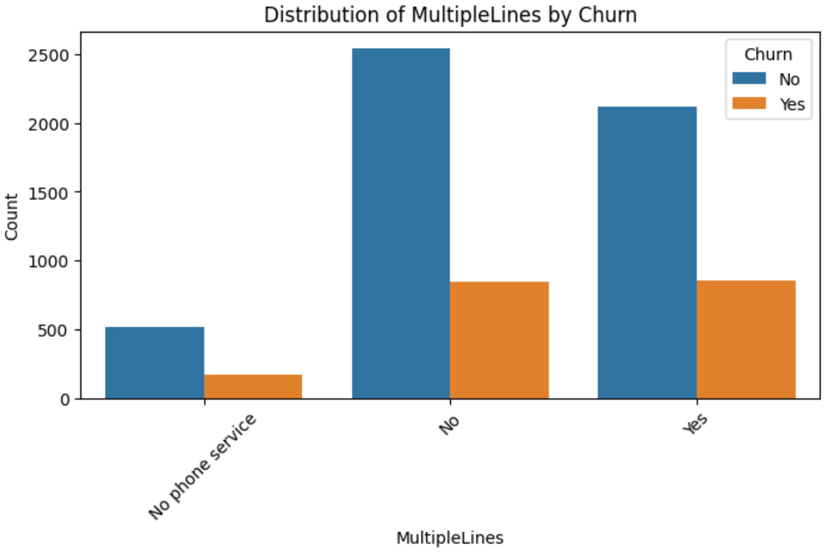

Distribution of SeniorCitizen by Churn
Churn
No
Yes
Count
4000
3000
2000
1000
0
0
1
SeniorCitizen

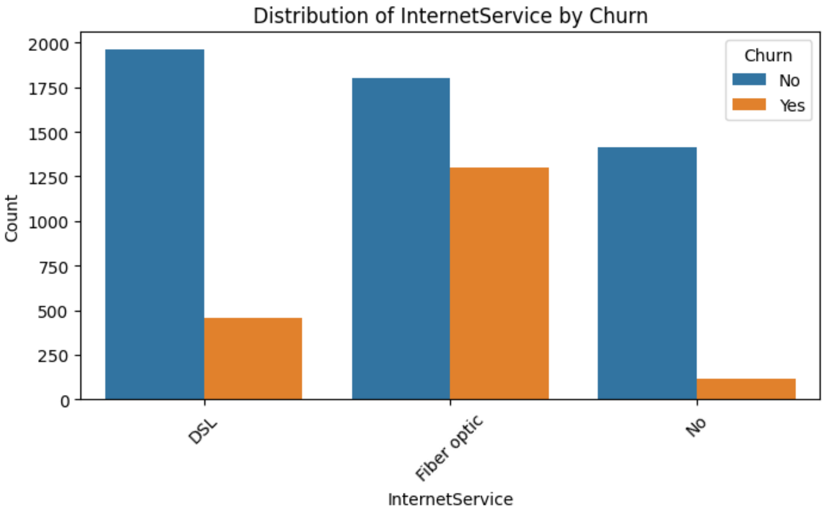

Distribution of Partner by Churn
Churn
No
Yes
Count
2500
2000
1500
1000
500
0
Yes
No
Partner

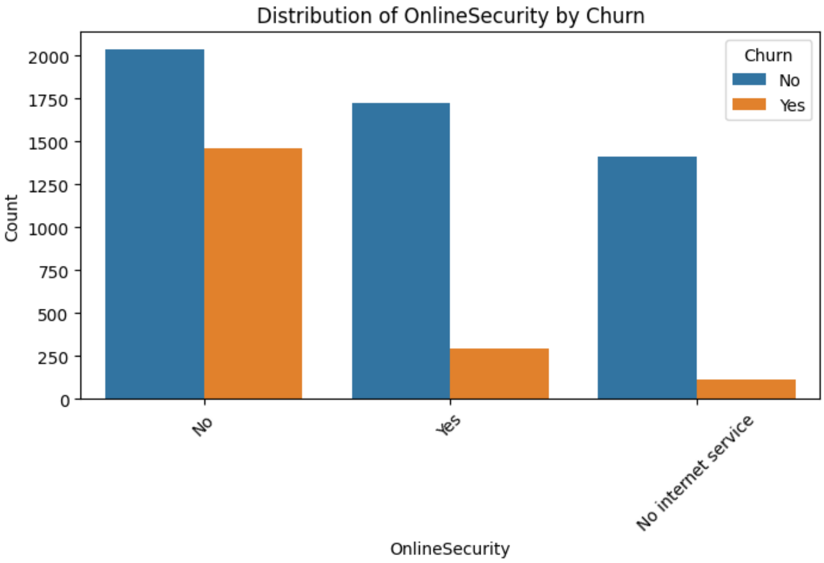

Distribution of Dependents by Churn
Churn
No
Yes
Count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents

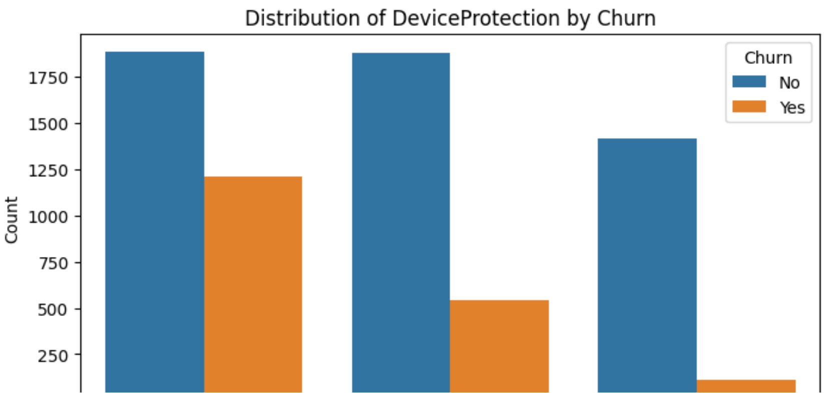

Distribution of PhoneService by Churn
Churn
No
Yes
Count
4000
3000
2000

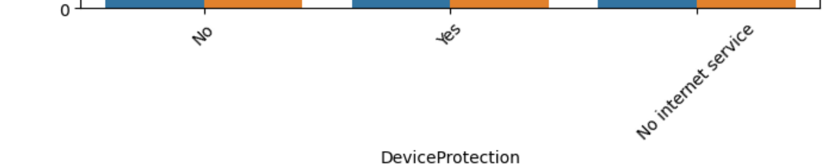

1000
0
No
Yes
PhoneService

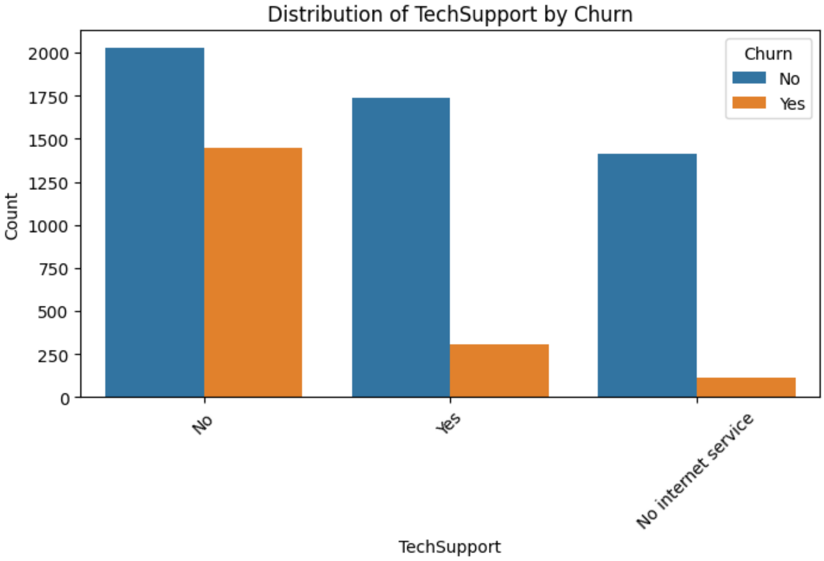

Distribution of MultipleLines by Churn
Churn
No
Yes
Count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

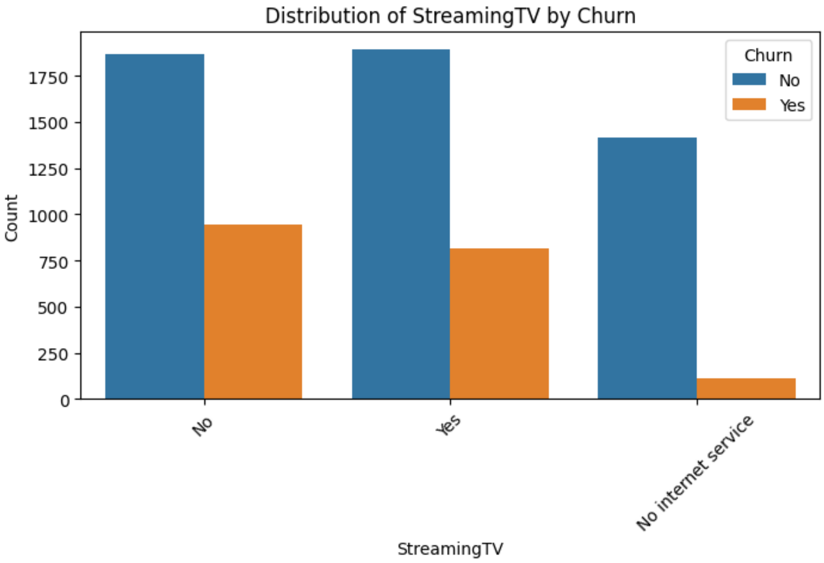

Distribution of InternetService by Churn
Churn
No
Yes
Count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

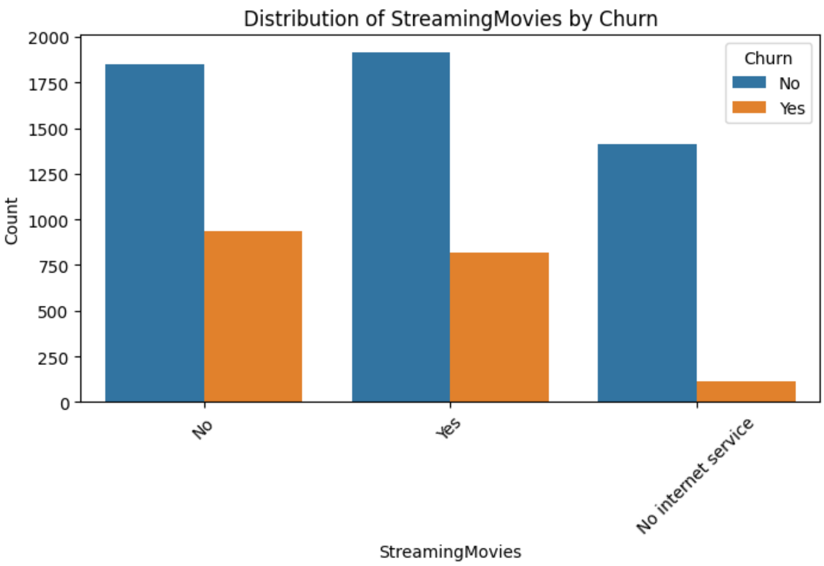

Distribution of OnlineSecurity by Churn
Churn
No
Yes
Count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

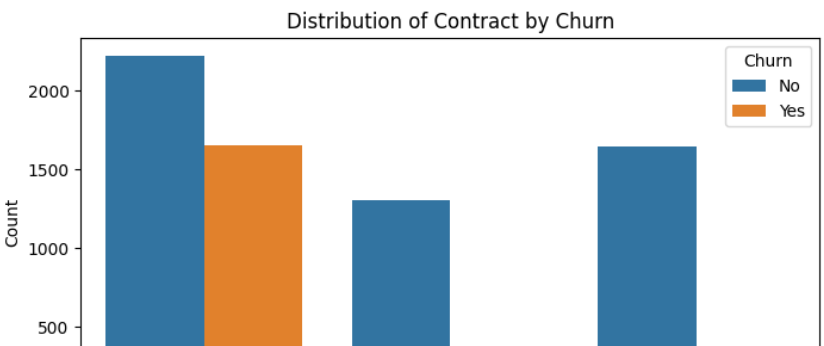

Distribution of DeviceProtection by Churn
Churn
No
Yes
Count
1750
1500
1250
1000
750
500
250

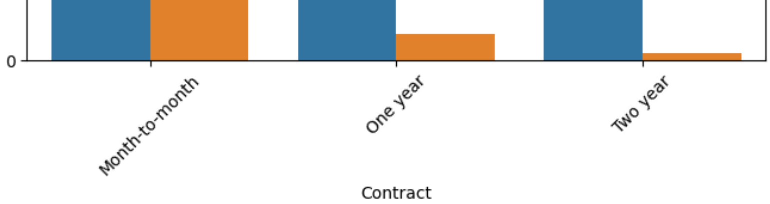

0
No
Yes
No internet service
DeviceProtection

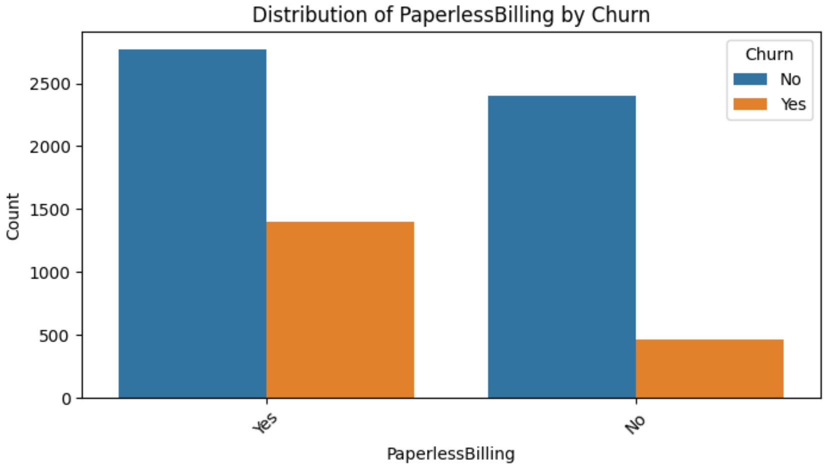

Distribution of TechSupport by Churn
Count
0
250
500
750
1000
1250
1500
1750
2000
Churn
No
Yes
No
Yes
No internet service
TechSupport

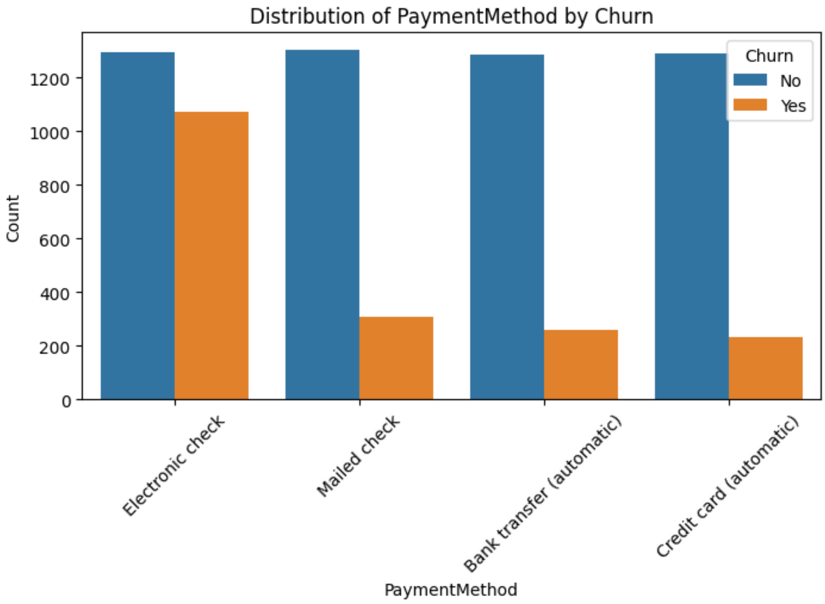

Distribution of StreamingTV by Churn
Count
0
250
500
750
1000
1250
1500
1750
Churn
No
Yes
No
Yes
No internet service
StreamingTV


Distribution of StreamingMovies by Churn
Count
0
250
500
750
1000
1250
1500
1750
2000
Churn
No
Yes
No
Yes
No internet service
StreamingMovies

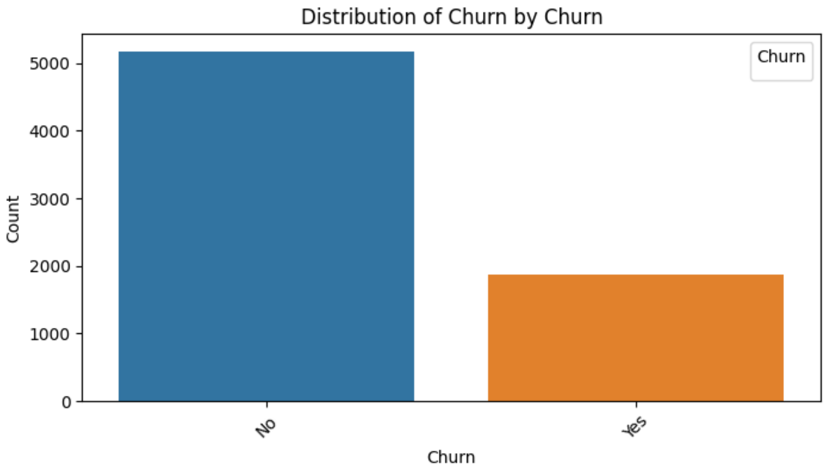

Distribution of Contract by Churn
Count
500
1000
1500
2000
Churn
No
Yes





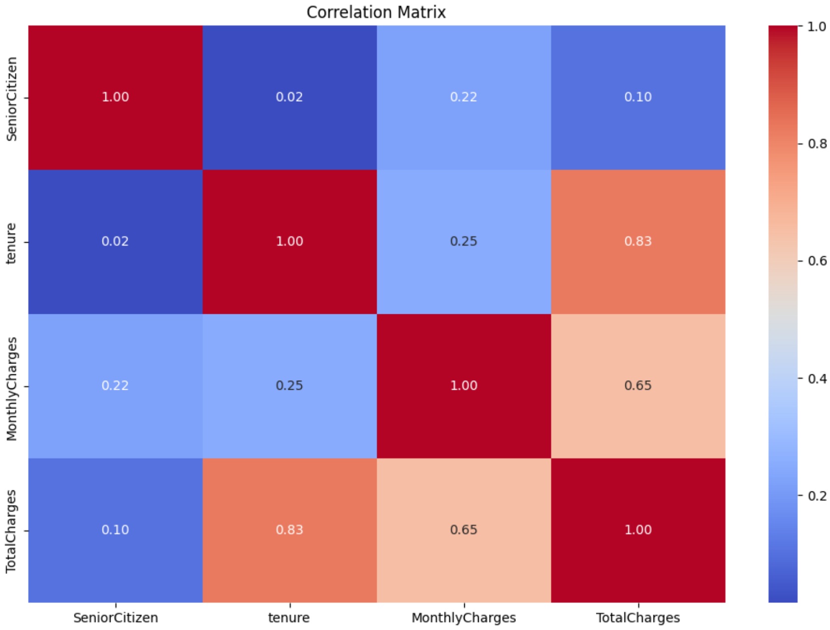



```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
```



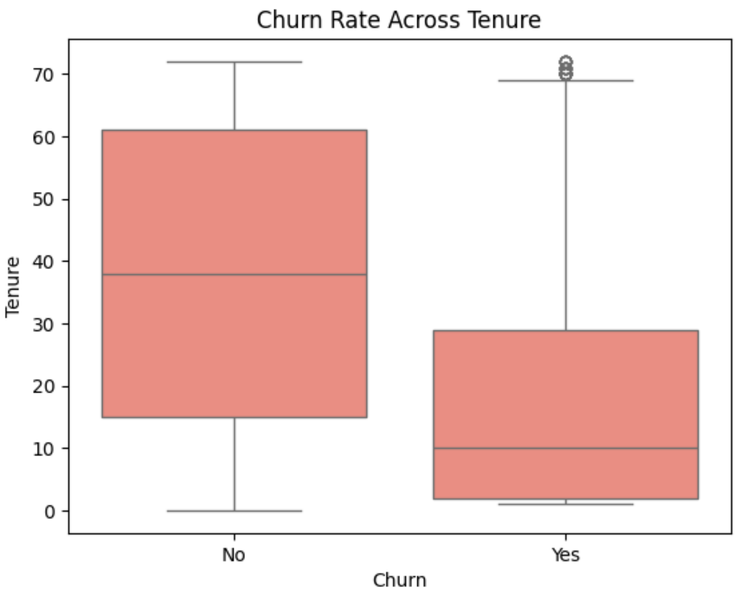
```
# Investigate relationships between features
# Select only numeric columns
numeric_df = data.select_dtypes(include=['number'])

# Calculate correlation matrix
correlation_matrix = numeric_df.corr()
# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
# Correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
Correlation Matrix:
                SeniorCitizen    tenure  MonthlyCharges  TotalCharges
SeniorCitizen        1.000000  0.016567        0.220173      0.102652
tenure               0.016567  1.000000        0.247900      0.825464
MonthlyCharges       0.220173  0.247900        1.000000      0.650864
TotalCharges         0.102652  0.825464        0.650864      1.000000
```



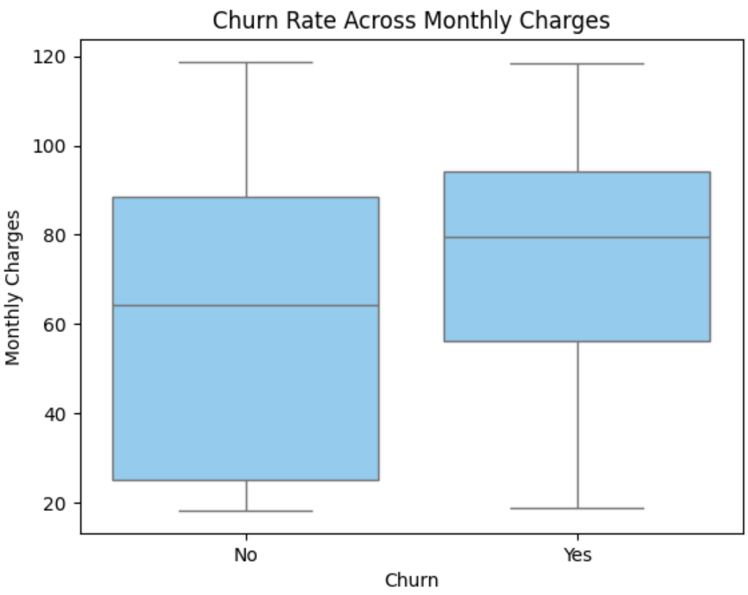

```
# Churn rate across tenure
sns.boxplot(x='Churn', y='tenure', data=data, color='salmon')
plt.title('Churn Rate Across Tenure')
plt.xlabel('Churn')
```

```
plt.ylabel('Tenure')
plt.show()
```



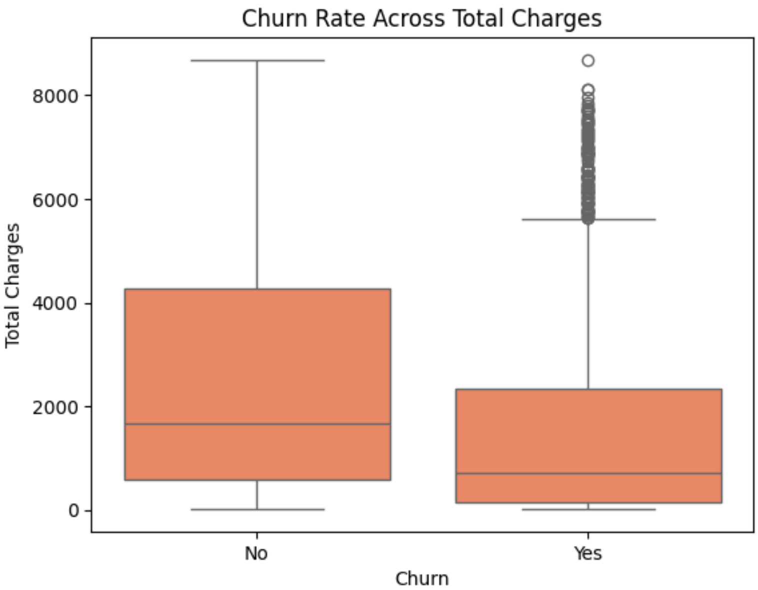
Long-term clients tend to stay while new clients having a short tenure of 1-2 years tend to churn.

```
# Churn rate across MonthlyCharges
sns.boxplot(x='Churn', y='MonthlyCharges', data=data, color='lightskyblue')
plt.title('Churn Rate Across Monthly Charges')
plt.xlabel('Churn')
plt.ylabel('Monthly Charges')
plt.show()
```



Increase in monthly charges is also seen to be a factor for higher churn rates.

```
# Churn rate across TotalCharges
sns.boxplot(x='Churn', y='TotalCharges', data=data, color='coral')
plt.title('Churn Rate Across Total Charges')
plt.xlabel('Churn')
plt.ylabel('Total Charges')
plt.show()
```

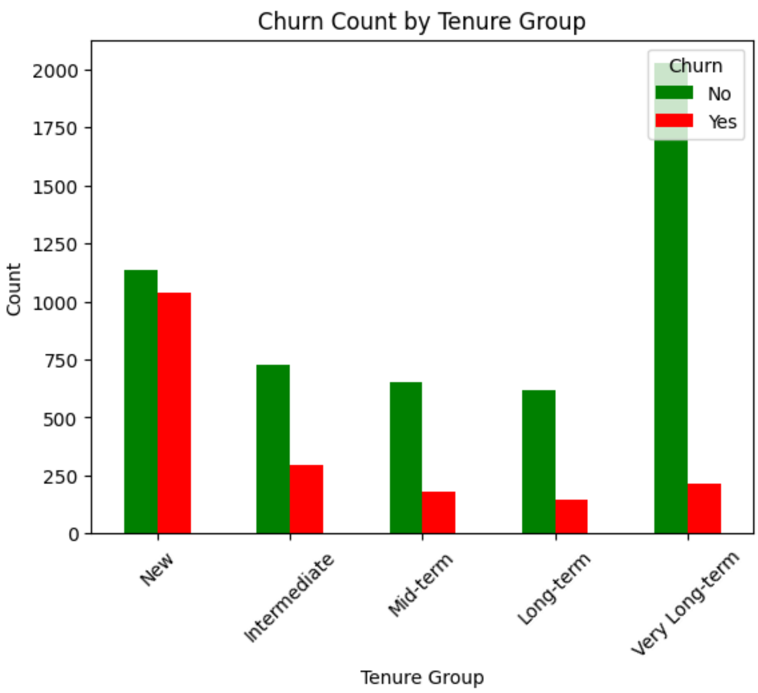


Total charges does not seem to be a sigificant factor for higher churn rate.

Feature Engineering

```
# Create new features
data['TenureGroup'] = pd.cut(data['tenure'], bins=[0, 12, 24, 36, 48, data['tenure'].max()], labels=['New', 'Intermediate', 'Mid-term', 'Long-term',
data['FamilyStatus'] = (data['Partner'] == 'Yes') | (data['Dependents'] == 'Yes')
data['NumServices'] = data[['PhoneService', 'InternetService', 'OnlineSecurity', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']
```

```
# Calculate churn counts within each tenure group
grouped = data.groupby(['TenureGroup', 'Churn']).size().unstack()

# Plot grouped bar plot
grouped.plot(kind='bar', stacked=False, color=['green', 'red'])
plt.title('Churn Count by Tenure Group')
plt.xlabel('Tenure Group')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Churn', loc='upper right')
plt.show()
```




```
data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
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	

5 rows × 24 columns

```
# Interaction features
data['InternetAndPhoneService'] = (data['PhoneService'] == 'Yes') & (data['InternetService'] != 'No')
data['SecurityAndSupportBundle'] = (data['OnlineSecurity'] == 'Yes') & (data['TechSupport'] == 'Yes')
```

```
data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
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	

5 rows × 26 columns

Feature Encoding

```
# Encode categorical features
data = pd.get_dummies(data, columns=['Contract', 'PaymentMethod','MultipleLines','InternetService','PaymentMethod','TenureGroup'])
```

```
data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	O
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	

5 rows × 43 columns

```
from sklearn.preprocessing import LabelEncoder
# Select categorical columns
categorical_columns = ['gender', 'Partner', 'Dependents', 'PhoneService', 'OnlineSecurity', 'PaperlessBilling','DeviceProtection','OnlineBackup', 'Te

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in categorical_columns:
    data[col] = label_encoder.fit_transform(data[col])
```

```
data.columns
```

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'FamilyStatus', 'NumServices', 'InternetAndPhoneService',
       'SecurityAndSupportBundle', 'Contract_Month-to-month',
       'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check',
       'MultipleLines_No', 'MultipleLines_No phone service',
       'MultipleLines_Yes', 'InternetService_DSL',
       'InternetService_Fiber optic', 'InternetService_No',
       'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check',
       'TenureGroup_New', 'TenureGroup_Intermediate', 'TenureGroup_Mid-term',
       'TenureGroup_Long-term', 'TenureGroup_Very Long-term'],
      dtype='object')
```

```
data.drop(columns=['customerID'], inplace=True) # is not required for analysis
```

```
data['Churn'] = data['Churn'].replace({'Yes': 1, 'No': 0})
```

```
data.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	OnlineSecurit
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	

5 rows × 42 columns

```
# Feature scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit_transform(data[['tenure', 'MonthlyCharges', 'TotalCharges']])
```

```
data.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	OnlineSecur:
0	0	0	1	0	0.013889	0	
1	1	0	0	0	0.472222	1	
2	1	0	0	0	0.027778	1	
3	1	0	0	0	0.625000	0	
4	0	0	0	0	0.027778	1	

5 rows × 42 columns

Feature Selection

```
# Feature selection
# Example: Recursive Feature Elimination (RFE) with a Random Forest Classifier
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

X = data.drop(columns=['Churn'])
y = data['Churn']

rf_model = RandomForestClassifier()
rfe = RFE(rf_model, n_features_to_select=10)  # Select top 10 features
rfe.fit(X, y)

selected_features = X.columns[rfe.support_]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['gender', 'tenure', 'OnlineSecurity', 'TechSupport', 'MonthlyCharges',
       'TotalCharges', 'NumServices', 'Contract_Month-to-month',
       'InternetService_Fiber optic', 'PaymentMethod_Electronic check'],
      dtype='object')
```

We can reduce the dimensionality of the data by selecting the most important features. This will resullt in an optimum network and improved results.

Predictive Modeling

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
# Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
# Random Forest Classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

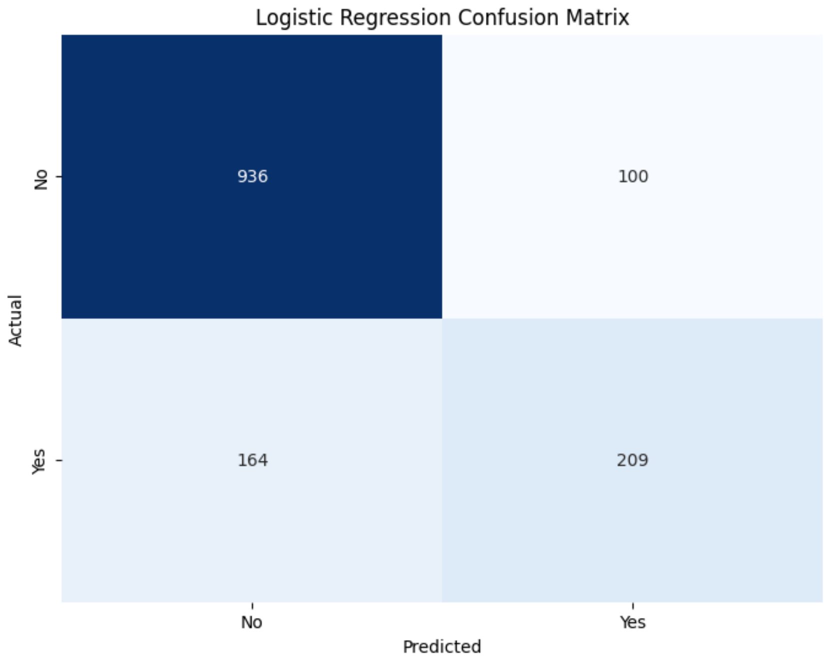
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Predictions
logistic_preds = logistic_model.predict(X_test)
rf_preds = rf_model.predict(X_test)


# Evaluation metrics
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, logistic_preds))
print("Precision:", precision_score(y_test, logistic_preds))
print("Recall:", recall_score(y_test, logistic_preds))
print("F1 Score:", f1_score(y_test, logistic_preds))
print("ROC-AUC Score:", roc_auc_score(y_test, logistic_preds))

print("\nRandom Forest Classifier:")
print("Accuracy:", accuracy_score(y_test, rf_preds))
print("Precision:", precision_score(y_test, rf_preds))
print("Recall:", recall_score(y_test, rf_preds))
print("F1 Score:", f1_score(y_test, rf_preds))
print("ROC-AUC Score:", roc_auc_score(y_test, rf_preds))
```

```
Logistic Regression:
Accuracy: 0.8126330731014905
Precision: 0.6763754045307443
Recall: 0.5603217158176944
F1 Score: 0.6129032258064516
ROC-AUC Score: 0.731898309646299

Random Forest Classifier:
Accuracy: 0.7899219304471257
Precision: 0.6452830188679245
Recall: 0.4584450402144772
F1 Score: 0.5360501567398118
ROC-AUC Score: 0.6838557247404432
```

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Set custom labels for confusion matrix
labels = ['No', 'Yes']

# Generate classification report and confusion matrix for Logistic Regression
logistic_report = classification_report(y_test, logistic_preds)
```

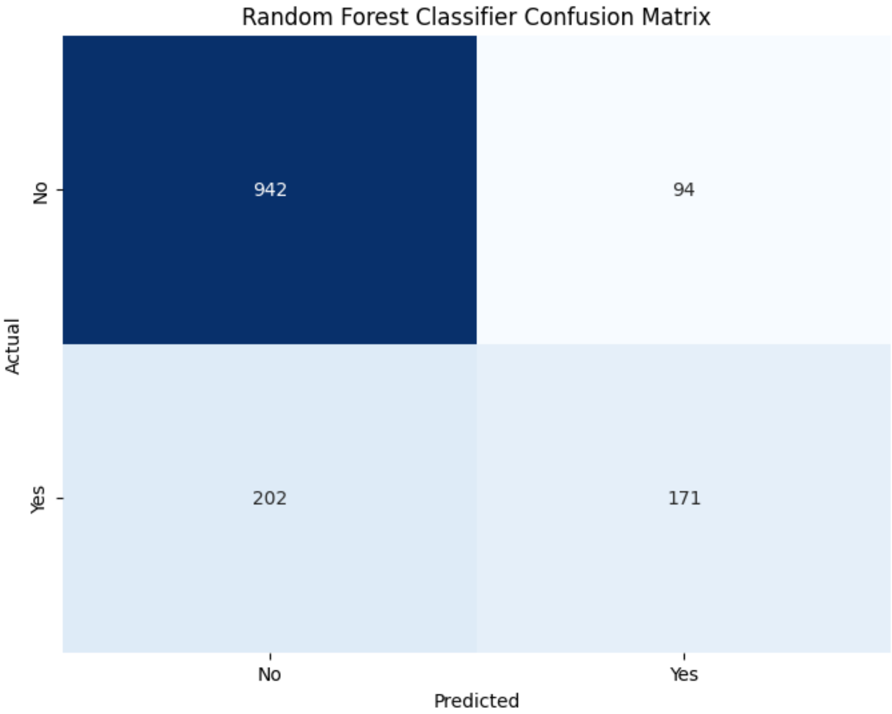
```
logistic_report = classification_report(y_test, logistic_preds)
logistic_conf_matrix = confusion_matrix(y_test, logistic_preds)

print("Logistic Regression Classification Report:")
print(logistic_report)

plt.figure(figsize=(8, 6))
sns.heatmap(logistic_conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False, xticklabels=labels, yticklabels=labels)
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1036
           1       0.68      0.56      0.61       373

    accuracy                           0.81      1409
   macro avg       0.76      0.73      0.74      1409
weighted avg       0.80      0.81      0.81      1409
```




```
# Generate classification report and confusion matrix for Random Forest Classifier
rf_report = classification_report(y_test, rf_preds)
rf_conf_matrix = confusion_matrix(y_test, rf_preds)

print("\nRandom Forest Classifier Classification Report:")
print(rf_report)

plt.figure(figsize=(8, 6))
sns.heatmap(rf_conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False, xticklabels=labels, yticklabels=labels)
plt.title('Random Forest Classifier Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Random Forest Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1036
           1       0.65      0.46      0.54       373

    accuracy                           0.79      1409
   macro avg       0.73      0.68      0.70      1409
weighted avg       0.78      0.79      0.78      1409
```



Model Tuning

Parameters used for tuning a logistic regression model:

1. `C`:
   - It controls the regularization strength in logistic regression. Smaller values indicate stronger regularization, preventing overfitting by penalizing large parameter values.
   - In the parameter grid, different values of `C` ranging from very small (0.001) to very large (100) are specified. This allows the grid search to explore a wide range of regularization strengths.
2. `penalty`:
   - This parameter determines the type of regularization used in logistic regression.
   - `'l1'` penalty refers to L1 regularization, which adds the absolute values of the coefficients to the loss function. It can lead to sparse solutions by pushing less informative features' coefficients to zero.
   - `'l2'` penalty refers to L2 regularization, which adds the squared magnitudes of the coefficients to the loss function. It tends to shrink the coefficients towards zero without necessarily setting them to zero.
   - By including both penalties in the parameter grid, the grid search will explore the effects of different types of regularization.

Overall, these parameters allow the grid search to systematically evaluate the logistic regression model's performance across various regularization strengths and types, helping to find the combination that optimizes the model's predictive performance.

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid for logistic regression
logistic_param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

# Grid search with cross-validation for logistic regression
logistic_grid_search = GridSearchCV(LogisticRegression(), logistic_param_grid, cv=5, scoring='accuracy')
logistic_grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

Certainly! Here's an explanation of the parameters in the `rf_param_grid` dictionary used for tuning a Random Forest Classifier:

1. **`n_estimators`**:
   - This parameter determines the number of trees in the random forest. Each tree in the forest is built using a different random subset of the training data.
   - In the parameter grid, `[50, 100, 200]` are the candidate values for `n_estimators`. These values represent different numbers of trees to be included in the random forest.
2. **`max_depth`**:
   - The `max_depth` parameter controls the maximum depth of each decision tree in the random forest. A deeper tree can capture more complex relationships in the data, but it also increases the risk of overfitting.
   - In the parameter grid, `[None, 10, 20]` are the candidate values for `max_depth`. Using `None` means that there is no maximum depth limit, allowing the trees to grow until all leaves are pure or until they contain less than `min_samples_split` samples.
3. **`min_samples_split`**:
   - This parameter specifies the minimum number of samples required to split an internal node in a decision tree. It helps control the tree's complexity and prevents overfitting.
   - In the parameter grid, `[2, 5, 10]` are the candidate values for `min_samples_split`. These values represent different thresholds for splitting nodes based on the number of samples.
4. **`min_samples_leaf`**:
   - The `min_samples_leaf` parameter determines the minimum number of samples required to be at a leaf node. It helps prevent overfitting by controlling the minimum size of the leaves.
   - In the parameter grid, `[1, 2, 4]` are the candidate values for `min_samples_leaf`. These values represent different thresholds for the minimum number of samples required to form a leaf node.

```
# Define parameter grid for random forest classifier
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
# Grid search with cross-validation for random forest classifier
rf_grid_search = GridSearchCV(RandomForestClassifier(), rf_param_grid, cv=5, scoring='accuracy')
rf_grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
# Get best parameters and best scores
print("Best parameters for Logistic Regression:", logistic_grid_search.best_params_)
print("Best score for Logistic Regression:", logistic_grid_search.best_score_)

print("\nBest parameters for Random Forest Classifier:", rf_grid_search.best_params_)
print("Best score for Random Forest Classifier:", rf_grid_search.best_score_)
```

```
Best parameters for Logistic Regression: {'C': 1, 'penalty': 'l2'}
Best score for Logistic Regression: 0.8026256853811106

Best parameters for Random Forest Classifier: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}
Best score for Random Forest Classifier: 0.8033355345381645
```

```
# Evaluate tuned models
tuned_logistic_model = logistic_grid_search.best_estimator_
tuned_rf_model = rf_grid_search.best_estimator_

tuned_logistic_preds = tuned_logistic_model.predict(X_test)
tuned_rf_preds = tuned_rf_model.predict(X_test)

# Compare performance before and after tuning
print("Logistic Regression Performance (Before Tuning):")
print(classification_report(y_test, logistic_preds))

print("\nLogistic Regression Performance (After Tuning):")
print(classification_report(y_test, tuned_logistic_preds))

print("\nRandom Forest Classifier Performance (Before Tuning):")
print(classification_report(y_test, rf_preds))

print("\nRandom Forest Classifier Performance (After Tuning):")
print(classification_report(y_test, tuned_rf_preds))
```

```
Logistic Regression Performance (Before Tuning):
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1036
           1       0.68      0.56      0.61       373

    accuracy                           0.81      1409
   macro avg       0.76      0.73      0.74      1409
weighted avg       0.80      0.81      0.81      1409


Logistic Regression Performance (After Tuning):
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1036
           1       0.68      0.56      0.61       373

    accuracy                           0.81      1409
   macro avg       0.76      0.73      0.74      1409
weighted avg       0.80      0.81      0.81      1409


Random Forest Classifier Performance (Before Tuning):
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1036
           1       0.65      0.46      0.54       373

    accuracy                           0.79      1409
   macro avg       0.73      0.68      0.70      1409
weighted avg       0.78      0.79      0.78      1409


Random Forest Classifier Performance (After Tuning):
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1036
           1       0.68      0.51      0.58       373

    accuracy                           0.81      1409
   macro avg       0.76      0.71      0.73      1409
weighted avg       0.79      0.81      0.80      1409
```

Interpretation and Conclusion

```
# 1. Interpretation of Model Results
# For Logistic Regression:
logistic_coefficients = logistic_model.coef_[0]
feature_names = X.columns

logistic_feature_importance = pd.DataFrame({'Feature': feature_names, 'Coefficient': logistic_coefficients})
logistic_feature_importance = logistic_feature_importance.sort_values(by='Coefficient', ascending=False)

print("Logistic Regression Coefficients:")
print(logistic_feature_importance)

# For Random Forest Classifier:
rf_feature_importance = pd.DataFrame({'Feature': feature_names, 'Importance': rf_model.feature_importances_})
rf_feature_importance = rf_feature_importance.sort_values(by='Importance', ascending=False)

print("\nRandom Forest Classifier Feature Importances:")
print(rf_feature_importance)
```

```
38                  TenureGroup_Mid-term    -0.132405
3                             Dependents    -0.137057
13                        MonthlyCharges    -0.190163
5                           PhoneService    -0.276744
26                      MultipleLines_No    -0.300611
9                            TechSupport    -0.325419
29                   InternetService_DSL    -0.332883
31                    InternetService_No    -0.337472
37              TenureGroup_Intermediate    -0.353014
6                         OnlineSecurity    -0.373056
21                     Contract_Two year    -0.850881
4                                 tenure    -3.420536

Random Forest Classifier Feature Importances:
                                 Feature  Importance
14                          TotalCharges    0.153012
13                        MonthlyCharges    0.142501
4                                 tenure    0.121467
19               Contract_Month-to-month    0.049192
6                         OnlineSecurity    0.036864
9                            TechSupport    0.035325
16                           NumServices    0.031878
36                       TenureGroup_New    0.030064
0                                 gender    0.026892
12                      PaperlessBilling    0.024989
30           InternetService_Fiber optic    0.024271
7                           OnlineBackup    0.022645
1                          SeniorCitizen    0.020404
8                       DeviceProtection    0.017300
2                                Partner    0.017131
3                             Dependents    0.016304
11                       StreamingMovies    0.016110
15                          FamilyStatus    0.015867
10                           StreamingTV    0.015132
24        PaymentMethod_Electronic check    0.014850
21                     Contract_Two year    0.014599
34        PaymentMethod_Electronic check    0.013669
26                      MultipleLines_No    0.013007
28                     MultipleLines_Yes    0.012994
29                   InternetService_DSL    0.011142
40             TenureGroup_Very Long-term    0.009736
20                     Contract_One year    0.008098
37              TenureGroup_Intermediate    0.007332
38                  TenureGroup_Mid-term    0.007155
32  PaymentMethod_Bank transfer (automatic)    0.007073
```