
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = '/content/ML Datasets.csv'  # Update the file path if
necessary
dataset = pd.read_csv(file_path)
```

Phase 1: Data Exploration and Preprocessing

```
#Convert TotalCharges to numeric (coercing errors to NaN)
dataset['TotalCharges'] = pd.to_numeric(dataset['TotalCharges'],
errors='coerce')

#Check for missing values
missing_data = dataset.isnull().sum()
print("Missing Data:", missing_data)
```

```
Missing Data: 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
```

```
# Drop rows with missing TotalCharges
dataset.dropna(subset=['TotalCharges'], inplace=True)

#Encode categorical variables
binary_columns = ['Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']
for col in binary_columns:
    dataset[col] = LabelEncoder().fit_transform(dataset[col])

# One-hot encode categorical columns
dataset = pd.get_dummies(dataset, columns=['MultipleLines',
'InternetService', 'OnlineSecurity',
                                           'OnlineBackup',
'DeviceProtection', 'TechSupport',
                                           'StreamingTV',
'StreamingMovies', 'Contract',
                                           'PaymentMethod'],
drop_first=True)

# Replace spaces with NaN in 'TotalCharges' column
dataset['TotalCharges'] = pd.to_numeric(dataset['TotalCharges'],
errors='coerce')

# Drop rows with missing TotalCharges
dataset.dropna(subset=['TotalCharges'], inplace=True)

#Scale numerical columns
scaler = StandardScaler()
dataset[['tenure', 'MonthlyCharges', 'TotalCharges']] =
scaler.fit_transform(
    dataset[['tenure', 'MonthlyCharges', 'TotalCharges']]
)
```

```python
# Display the preprocessed dataset info
print(dataset.info())
print(dataset.head())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7042
Data columns (total 32 columns):
 #   Column                                   Non-Null Count  Dtype  
---  ------                                   --------------  -----  
 0   customerID                               7032 non-null   object 
 1   gender                                   7032 non-null   object 
 2   SeniorCitizen                            7032 non-null   int64  
 3   Partner                                  7032 non-null   int64  
 4   Dependents                               7032 non-null   int64  
 5   tenure                                   7032 non-null   float64
 6   PhoneService                             7032 non-null   int64  
 7   PaperlessBilling                         7032 non-null   int64  
 8   MonthlyCharges                           7032 non-null   float64
 9   TotalCharges                             7032 non-null   float64
 10  Churn                                    7032 non-null   int64  
 11  MultipleLines_No phone service           7032 non-null   bool   
 12  MultipleLines_Yes                        7032 non-null   bool   
 13  InternetService_Fiber optic              7032 non-null   bool   
 14  InternetService_No                       7032 non-null   bool   
 15  OnlineSecurity_No internet service       7032 non-null   bool   
 16  OnlineSecurity_Yes                       7032 non-null   bool   
 17  OnlineBackup_No internet service         7032 non-null   bool   
 18  OnlineBackup_Yes                         7032 non-null   bool   
 19  DeviceProtection_No internet service     7032 non-null   bool   
 20  DeviceProtection_Yes                     7032 non-null   bool   
 21  TechSupport_No internet service          7032 non-null   bool   
 22  TechSupport_Yes                          7032 non-null   bool   
 23  StreamingTV_No internet service          7032 non-null   bool   
 24  StreamingTV_Yes                          7032 non-null   bool   
 25  StreamingMovies_No internet service      7032 non-null   bool   
 26  StreamingMovies_Yes                      7032 non-null   bool   
 27  Contract_One year                        7032 non-null   bool   
 28  Contract_Two year                        7032 non-null   bool   
 29  PaymentMethod_Credit card (automatic)    7032 non-null   bool   
 30  PaymentMethod_Electronic check           7032 non-null   bool   
 31  PaymentMethod_Mailed check               7032 non-null   bool   
dtypes: bool(21), float64(3), int64(6), object(2)
memory usage: 803.5+ KB
None
   customerID  gender  SeniorCitizen  Partner  Dependents    tenure  \
0  7590-VHVEG  Female              0        1           0 -1.280248   
1  5575-GNVDE    Male              0        0           0  0.064303   
2  3668-QPYBK    Male              0        0           0 -1.239504   
3  7795-CFOCW    Male              0        0           0  0.512486   
4  9237-HQITU  Female              0        0           0 -1.239504   
```

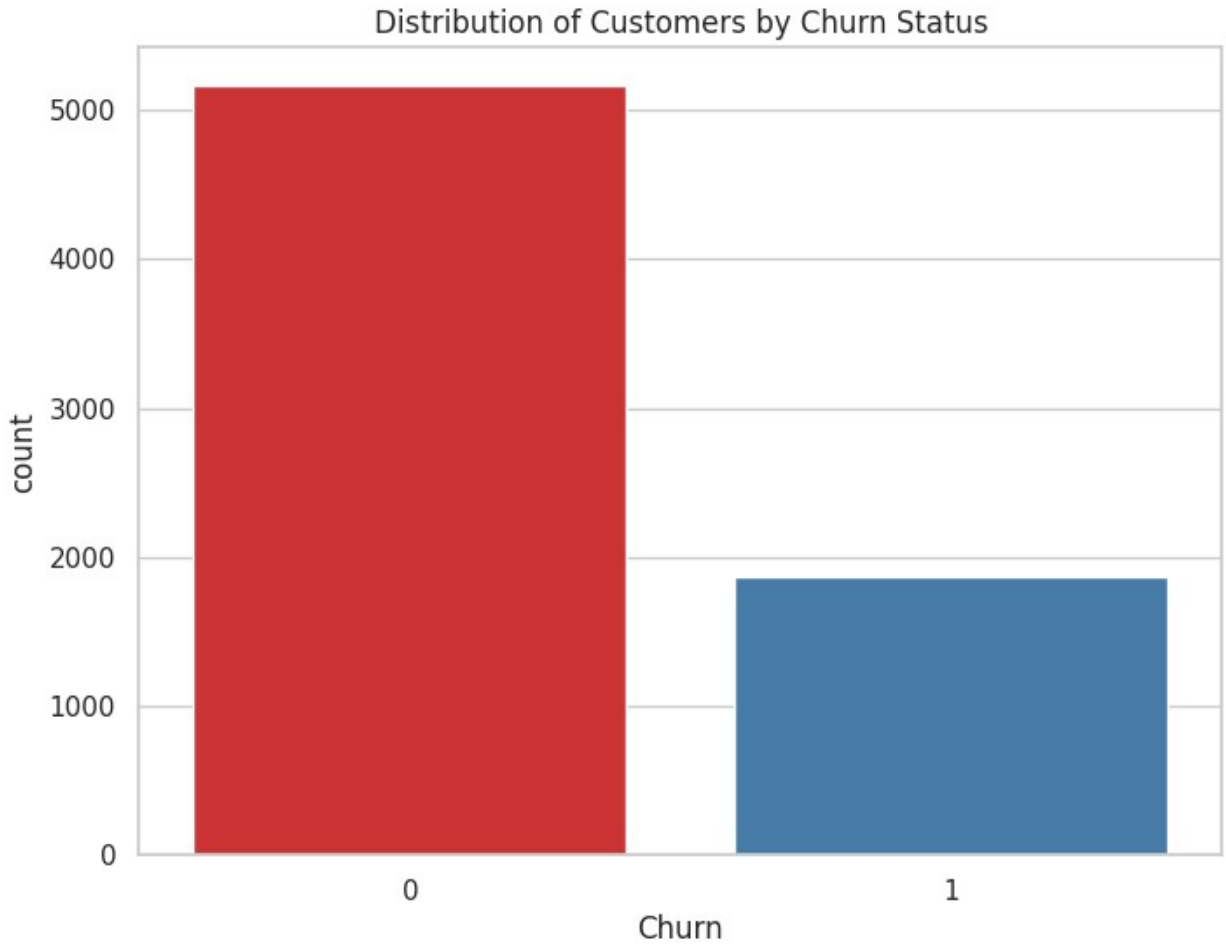
```
   PhoneService  PaperlessBilling  MonthlyCharges
TotalCharges  ...  \
0             0                 1       -1.161694     -0.994194  ...

1             1                 0       -0.260878     -0.173740  ...

2             1                 1       -0.363923     -0.959649  ...

3             0                 0       -0.747850     -0.195248  ...

4             1                 1        0.196178     -0.940457  ...


   TechSupport_Yes  StreamingTV_No internet service
StreamingTV_Yes  \
0            False                            False             False

1            False                            False             False

2            False                            False             False

3             True                            False             False

4            False                            False             False


   StreamingMovies_No internet service  StreamingMovies_Yes  \
0                                False                False
1                                False                False
2                                False                False
3                                False                False
4                                False                False

   Contract_One year  Contract_Two year  \
0              False              False
1               True              False
2              False              False
3               True              False
4              False              False

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic
check  \
0                                  False
True
1                                  False
False
2                                  False
False
3                                  False
False
```

```
4                                    False
True

   PaymentMethod_Mailed check
0                      False
1                       True
2                       True
3                      False
4                      False

[5 rows x 32 columns]
```

Phase 2: Exploratory Data Analysis (EDA)

```
# Set plot style
sns.set(style="whitegrid")

#Visualizing the distribution of customers who churned vs. those who
did not
plt.figure(figsize=(8, 6))
sns.countplot(x='Churn', data=dataset, palette='Set1')
plt.title('Distribution of Customers by Churn Status')
plt.show()
```

```
<ipython-input-34-8d131e4bf0ba>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Churn', data=dataset, palette='Set1')
```




```
#Analyzing relationship between tenure, monthly charges, and churn
plt.figure(figsize=(14, 6))
```

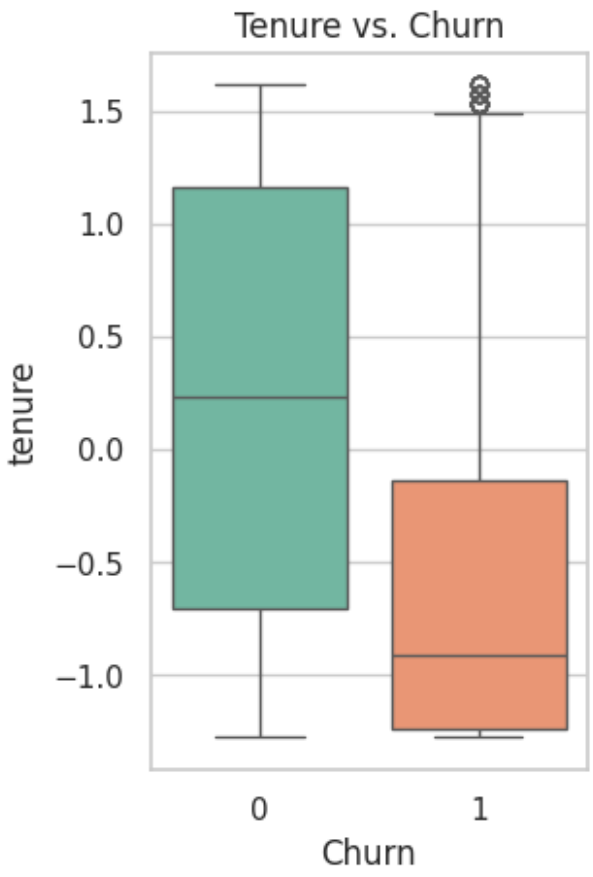
```
<Figure size 1400x600 with 0 Axes>
```

```
<Figure size 1400x600 with 0 Axes>
```

```
# Tenure vs. Churn
plt.subplot(1, 2, 1)
sns.boxplot(x='Churn', y='tenure', data=dataset, palette='Set2')
plt.title('Tenure vs. Churn')
```

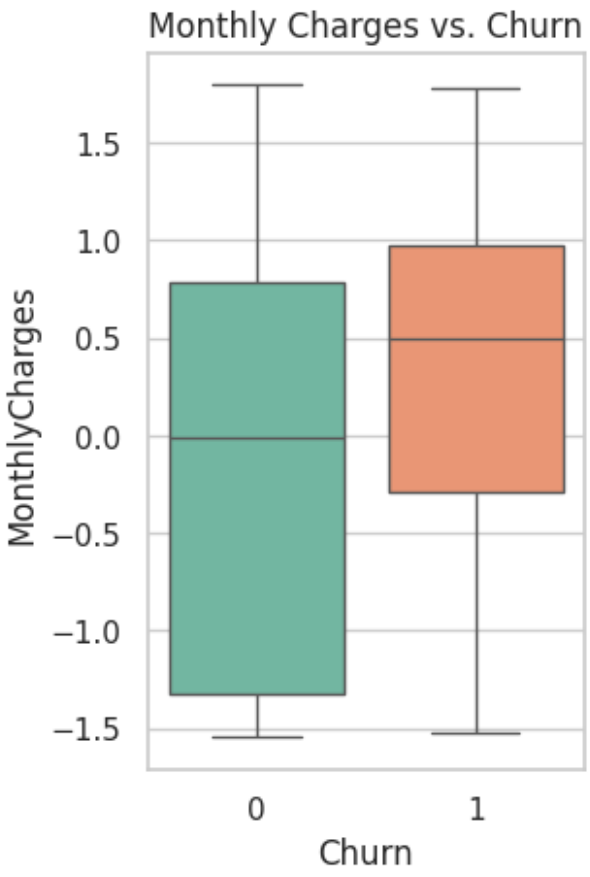
```
<ipython-input-36-d4d76e3c870f>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Churn', y='tenure', data=dataset, palette='Set2')
```

```
Text(0.5, 1.0, 'Tenure vs. Churn')
```

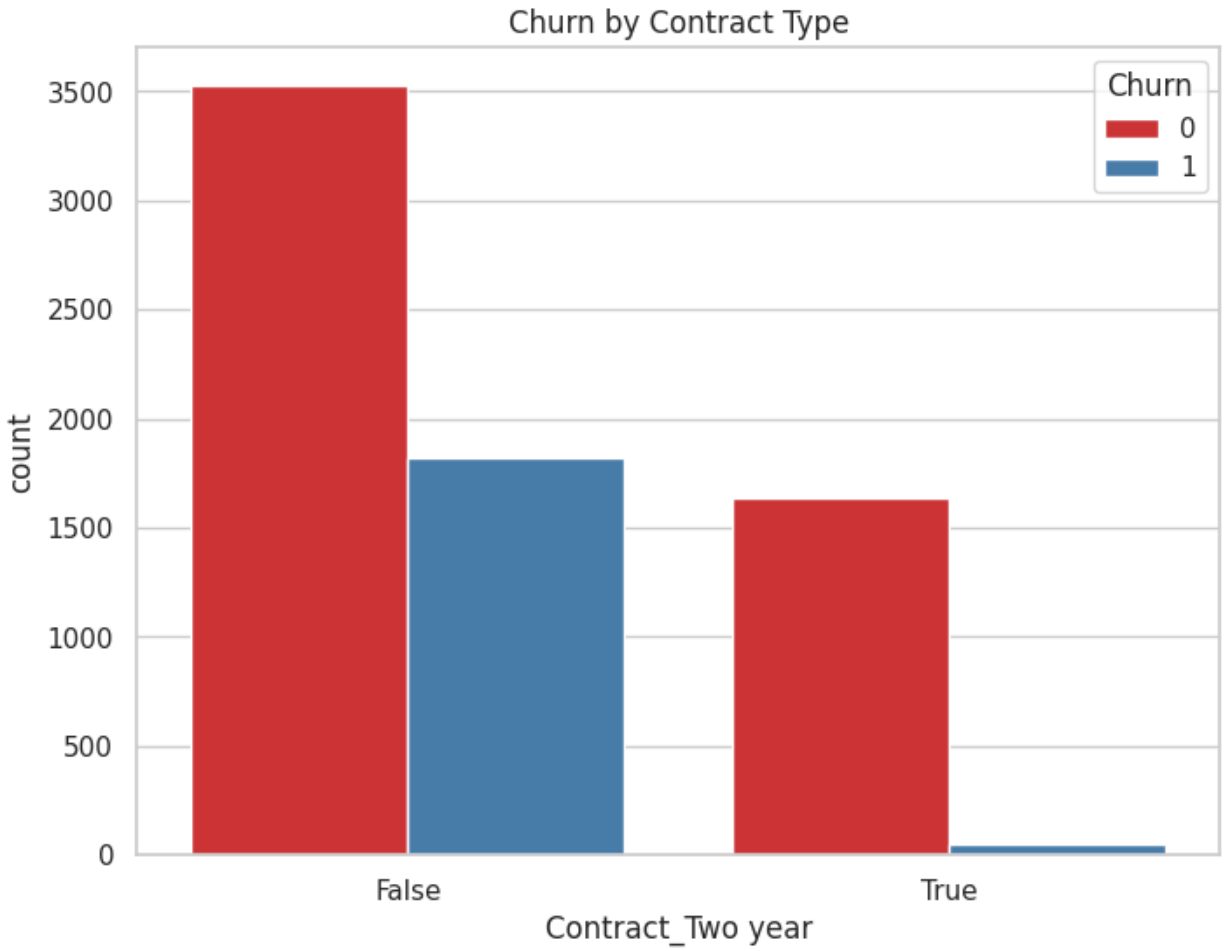



```
# Monthly Charges vs. Churn
plt.subplot(1, 2, 2)
sns.boxplot(x='Churn', y='MonthlyCharges', data=dataset,
palette='Set2')
plt.title('Monthly Charges vs. Churn')
plt.show()
```

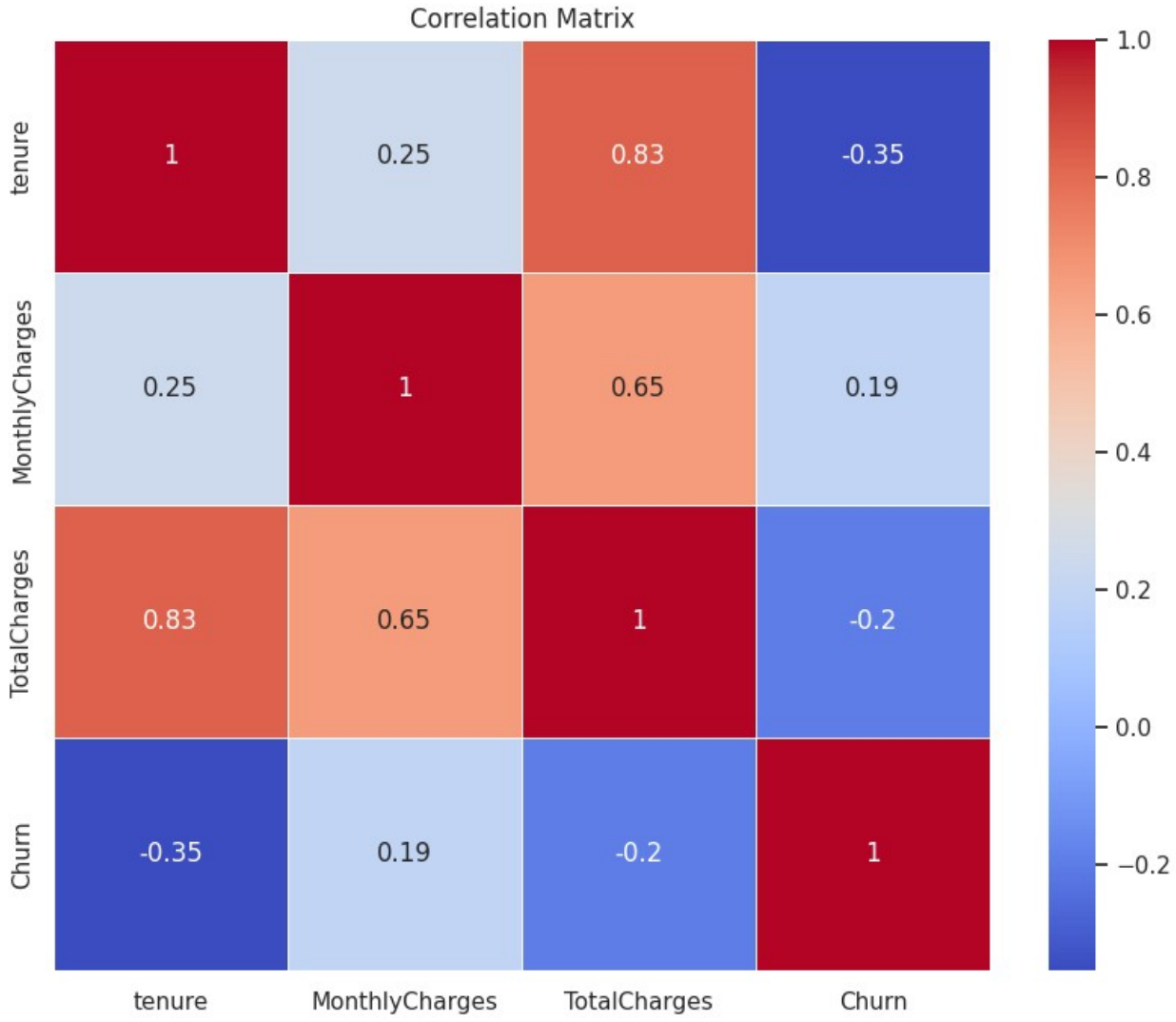
```
<ipython-input-37-854522a3c161>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Churn', y='MonthlyCharges', data=dataset,
palette='Set2')
```




```
#Analyzing the relationship between contract type and churn
plt.figure(figsize=(8, 6))
sns.countplot(x='Contract_Two year', hue='Churn', data=dataset,
palette='Set1')
plt.title('Churn by Contract Type')
plt.show()
```




```
#Correlation heatmap (numerical variables)
plt.figure(figsize=(10, 8))
corr = dataset[['tenure', 'MonthlyCharges', 'TotalCharges',
'Churn']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



Phase 3: Model Development

```
# Recheck the actual column names first (inspect the print output)
print(dataset.columns)

# One-hot encode the correct categorical columns based on the actual
dataset
# Adjust column names if they differ slightly
categorical_columns = ['MultipleLines', 'InternetService',
'OnlineSecurity',
                       'OnlineBackup', 'DeviceProtection',
'TechSupport',
                       'StreamingTV', 'StreamingMovies', 'Contract',
                       'PaymentMethod']

Index(['customerID', 'gender', 'SeniorCitizen', 'Partner',
'Dependents',
```

```
       'tenure', 'PhoneService', 'PaperlessBilling', 'MonthlyCharges',
       'TotalCharges', 'Churn', 'MultipleLines_No phone service',
       'MultipleLines_Yes', 'InternetService_Fiber optic',
       'InternetService_No', 'OnlineSecurity_No internet service',
       'OnlineSecurity_Yes', 'OnlineBackup_No internet service',
       'OnlineBackup_Yes', 'DeviceProtection_No internet service',
       'DeviceProtection_Yes', 'TechSupport_No internet service',
       'TechSupport_Yes', 'StreamingTV_No internet service',
'StreamingTV_Yes',
       'StreamingMovies_No internet service', 'StreamingMovies_Yes',
       'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed
check'],
      dtype='object')
```

```
# Only apply one-hot encoding to columns that exist in the dataset
available_columns = [col for col in categorical_columns if col in
dataset.columns]
dataset = pd.get_dummies(dataset, columns=available_columns,
drop_first=True)

# Now proceed with splitting the data and training the model as
before:
X = dataset.drop(['customerID', 'Churn'], axis=1)  # Drop unnecessary
columns
y = dataset['Churn']  # Target variable

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

# Train Logistic Regression Model
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train)

# Train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
# Logistic Regression Evaluation
log_predictions = log_model.predict(X_test)
print("Logistic Regression Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, log_predictions)}")
print(f"ROC-AUC Score: {roc_auc_score(y_test, log_predictions)}")
print(classification_report(y_test, log_predictions))
```

```
Logistic Regression Model Performance:
Accuracy: 0.7839374555792467
ROC-AUC Score: 0.6865458583327726
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1033
           1       0.62      0.48      0.54       374

    accuracy                           0.78      1407
   macro avg       0.72      0.69      0.70      1407
weighted avg       0.77      0.78      0.77      1407
```

```
# Random Forest Evaluation
rf_predictions = rf_model.predict(X_test)
print("\nRandom Forest Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, rf_predictions)}")
print(f"ROC-AUC Score: {roc_auc_score(y_test, rf_predictions)}")
print(classification_report(y_test, rf_predictions))
```

```
Random Forest Model Performance:
Accuracy: 0.7917555081734187
ROC-AUC Score: 0.6927230277836736
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1033
           1       0.65      0.48      0.55       374

    accuracy                           0.79      1407
   macro avg       0.74      0.69      0.71      1407
weighted avg       0.78      0.79      0.78      1407
```

Phase 4: Code for Model Evaluation

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, classification_report,
roc_curve, auc
import matplotlib.pyplot as plt

# Logistic Regression Evaluation
log_predictions = log_model.predict(X_test)
log_probabilities = log_model.predict_proba(X_test)[:, 1]  # For ROC-
AUC

print("Logistic Regression Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, log_predictions)}")
print(f"Precision: {precision_score(y_test, log_predictions)}")
print(f"Recall: {recall_score(y_test, log_predictions)}")
print(f"F1 Score: {f1_score(y_test, log_predictions)}")
print(f"ROC-AUC Score: {roc_auc_score(y_test, log_probabilities)}")
print(classification_report(y_test, log_predictions))
```

```
Logistic Regression Model Performance:
Accuracy: 0.7839374555792467
Precision: 0.6215277777777778
Recall: 0.4786096256684492
F1 Score: 0.540785498489426
ROC-AUC Score: 0.8307535810240667
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1033
           1       0.62      0.48      0.54       374

    accuracy                           0.78      1407
   macro avg       0.72      0.69      0.70      1407
weighted avg       0.77      0.78      0.77      1407


# Random Forest Evaluation
rf_predictions = rf_model.predict(X_test)
rf_probabilities = rf_model.predict_proba(X_test)[:, 1]  # For ROC-AUC

print("\nRandom Forest Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, rf_predictions)}")
print(f"Precision: {precision_score(y_test, rf_predictions)}")
print(f"Recall: {recall_score(y_test, rf_predictions)}")
print(f"F1 Score: {f1_score(y_test, rf_predictions)}")
print(f"ROC-AUC Score: {roc_auc_score(y_test, rf_probabilities)}")
print(classification_report(y_test, rf_predictions))


Random Forest Model Performance:
Accuracy: 0.7917555081734187
Precision: 0.6451612903225806
Recall: 0.48128342245989303
F1 Score: 0.5513016845329249
ROC-AUC Score: 0.8182607637792422
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1033
           1       0.65      0.48      0.55       374

    accuracy                           0.79      1407
   macro avg       0.74      0.69      0.71      1407
weighted avg       0.78      0.79      0.78      1407


# Plot ROC Curves for both models
log_fpr, log_tpr, _ = roc_curve(y_test, log_probabilities)
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probabilities)

plt.figure(figsize=(10, 6))
plt.plot(log_fpr, log_tpr, label='Logistic Regression (AUC = %0.2f)' %
```

```
auc(log_fpr, log_tpr))
plt.plot(rf_fpr, rf_tpr, label='Random Forest (AUC = %0.2f)' %
auc(rf_fpr, rf_tpr))
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```