Customer Churn Prediction Project

Introduction

In this project, we analyze a customer dataset to predict whether a customer will churn (leave) or not using machine learning models. We use logistic regression and random forest classifiers to build and evaluate our predictive model.

Exploratory Data Analysis (EDA)

We start by loading the dataset into a DataFrame and displaying the first 5 rows to understand its structure.

In [25]:
```
import pandas as pd

# Read the dataset
data = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')

# Display the first 5 rows of the dataset
data.head()
```

Out[25]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceP
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

Checking Data Information

We check the basic information about the dataset, including the number of rows, columns, and the data types of each column.

In [26]:

```
# Display dataset information
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      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Summary Statistics

We review summary statistics for numerical columns to get a sense of the data distribution and any potential anomalies.

In [27]:

```
# Display summary statistics for numerical columns
data.describe()
```

Out[27]:

	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

Identifying Categorical and Numerical Variables

We identify the categorical and numerical variables in the dataset. This will help us in preprocessing steps.

In [29]:

```
# Identify categorical and numerical columns
categorical_columns = data.select_dtypes(include=['object']).columns
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns

print("Categorical columns:", categorical_columns)
print("Numerical columns:", numerical_columns)
```

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

Data Preprocessing

We preprocess the data by encoding categorical variables into numerical format. This is necessary for machine learning models which require numerical input.

In [30]:
```python
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to categorical columns
data_encoded = data.copy()
categorical_columns = data_encoded.select_dtypes(include=['object']).columns

for col in categorical_columns:
    data_encoded[col] = le.fit_transform(data_encoded[col])

# Display the first few rows of the encoded dataset
data_encoded.head()
```

Out[30]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceP
0	5375	0	0	1	0	1	0	1	0	0	...	
1	3962	1	0	0	0	34	1	0	0	2	...	
2	2564	1	0	0	0	2	1	0	0	2	...	
3	5535	1	0	0	0	45	0	1	0	2	...	
4	6511	0	0	0	0	2	1	0	1	0	...	

5 rows × 21 columns

Defining Features and Target Variable

We separate the features (X) and the target variable (y). The target variable is 'Churn', which we aim to predict.

In [31]:
```python
# Define features and target variable
X = data_encoded.drop('Churn', axis=1)
y = data_encoded['Churn']
```

Splitting Data into Training and Testing Sets

We split the dataset into training and testing sets to evaluate the performance of our models. This helps us assess how well the models generalize to new, unseen data.

In [32]:
```
from sklearn.model_selection import train_test_split

# Split the 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)
```

Model Building

We apply different machine learning models to predict customer churn and evaluate their performance.

Logistic Regression

Logistic Regression is used to model the probability of a binary outcome based on one or more predictor variables. Here, we evaluate its performance on the test data.

In [35]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Initialize and train the Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)

# Predict on the test set
y_pred = lr_model.predict(X_test)

# Evaluate the Logistic Regression model
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Logistic Regression Confusion Matrix:
[[938  98]
 [167 206]]

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1036
           1       0.68      0.55      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

Random Forest is an ensemble learning method that combines multiple decision trees to improve performance. We compare its results with Logistic Regression.

In [36]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Predict on the test set
y_rf_pred = rf_model.predict(X_test)

# Evaluate the Random Forest model
print("Random Forest Confusion Matrix:")
print(confusion_matrix(y_test, y_rf_pred))
print("\nRandom Forest Classification Report:")
print(classification_report(y_test, y_rf_pred))
```

```
Random Forest Confusion Matrix:
[[942  94]
 [182 191]]

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1036
           1       0.67      0.51      0.58       373

    accuracy                           0.80      1409
   macro avg       0.75      0.71      0.73      1409
weighted avg       0.79      0.80      0.80      1409
```

Model Evaluation

We evaluate the performance of both models using metrics such as precision, recall, and F1-score. We also present confusion matrices to visualize the classification performance.

Visualization of Confusion Matrices

We visualize the confusion matrices for both Logistic Regression and Random Forest models to better understand their classification performance.

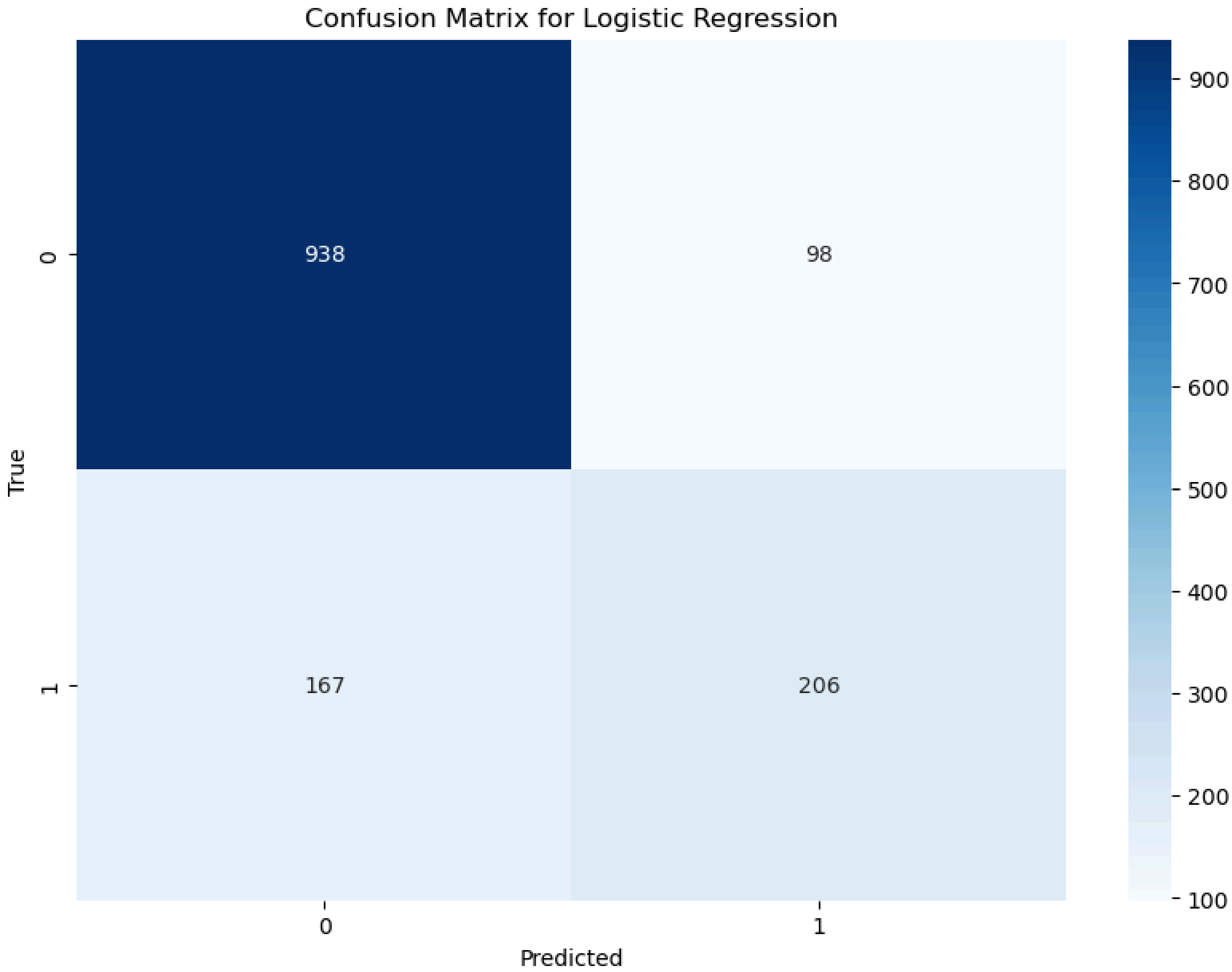
In [38]:

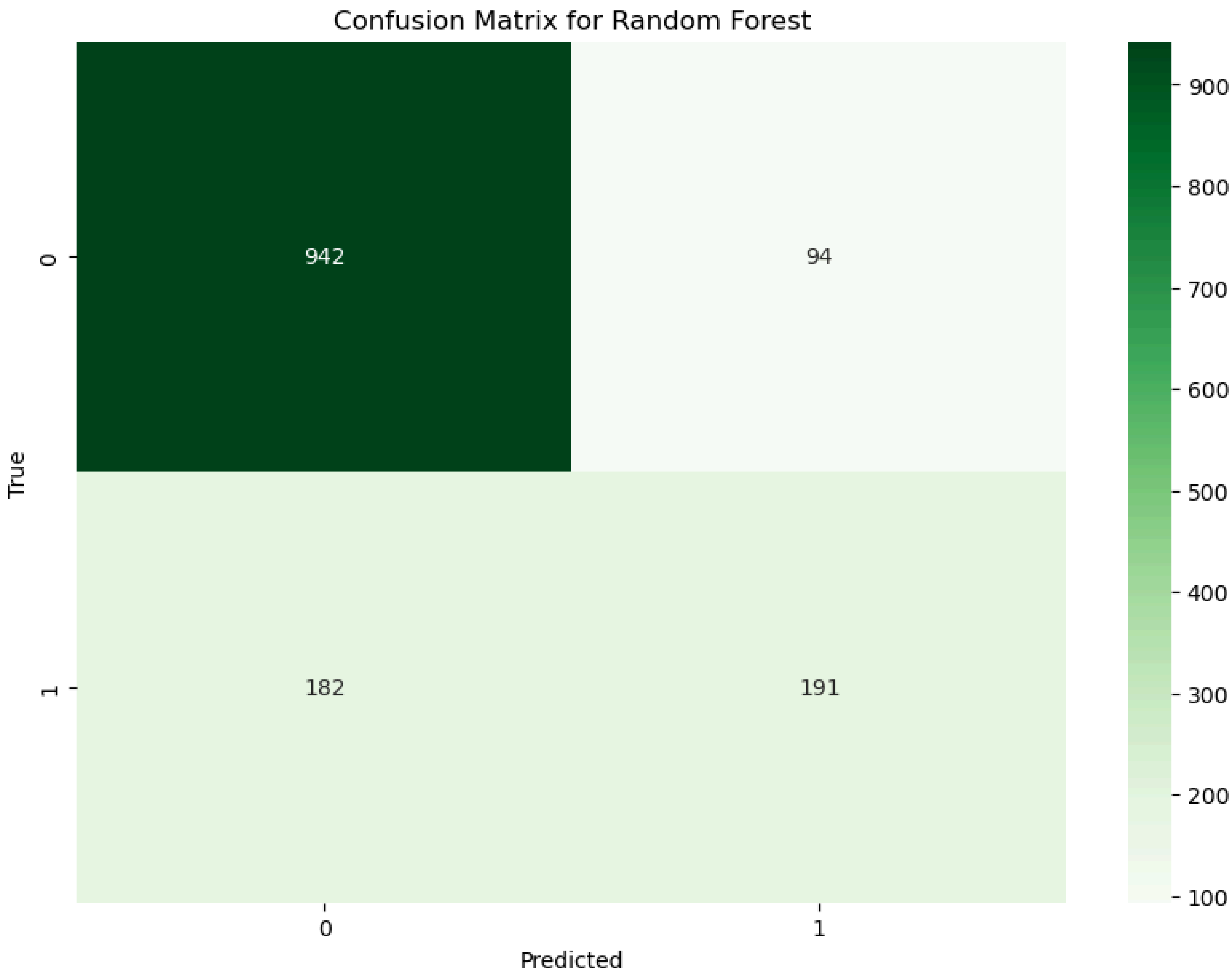
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plot confusion matrix for Logistic Regression
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Plot confusion matrix for Random Forest
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(y_test, y_rf_pred), annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix for Random Forest')
plt.xlabel('Predicted')
```

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





Recommendations

Based on the analysis, we provide recommendations for choosing the best model and potential strategies for improving predictions.

Choosing the Best Model

Based on the performance metrics and confusion matrices, we can determine which model performs better and why. Consider factors like precision, recall, and F1-score for each class.

Potential Strategies for Improvement

1. **Feature Engineering:** Explore additional features or create new features that may improve model performance.
2. **Hyperparameter Tuning:** Adjust model parameters to find the best configuration.
3. **Cross-Validation:** Use cross-validation to ensure the model generalizes well to different data subsets.
4. **Additional Models:** Consider experimenting with other machine learning algorithms to see if they offer better performance.

In [ ]:

```

```