index\_4

# Phase\_3 Project- Data Science[¶](#Phase_3-Project--Data-Science)

Name: Gladys Kemunto /git hub repo: <https://github.com/Mosota-Kemunto-Gladys-2020/Gladys_phase_3_project.git>

### Project Overview[¶](#Project-Overview)

Objective:

SyriaTel, a leading telecommunications company in Syria, is facing substantial financial losses due to customer churn.**Customer churn refers to the rate at which customers discontinue their relationship with a company within a specific period, often due to various reason(s).** In previous projects, SyriaTel focused on descriptive and inferential analyses to understand customer behavior and the relationships between different variables. However, to address the current churn issue, SyriaTel is shifting towards a predictive approach. The goal of this project is to develop a predictive model using the **SyriaTel Customer Churn dataset** that accurately identifies customers at risk of churning. By determining the predictive power of various features and understanding which variables most influence churn, SyriaTel aims to implement targeted retention strategies, reduce churn rates, and ultimately improve customer loyalty and profitability.

### Business Understanding[¶](#Business-Understanding)

In the telecommunications industry, customer churn is a critical challenge that directly impacts profitability. For SyriaTel, the loss of customers due to churn is a significant concern, prompting the need for more advanced analytical techniques. Previously, the company focused on descriptive and inferential methods to explore the distributions of key variables and their relationships, gaining valuable insights into customer behavior. However, understanding these relationships alone is no longer sufficient.

To effectively combat churn, SyriaTel is now adopting a predictive approach, which involves building a model that can forecast which customers are likely to churn in the near future. The predictive model will analyze a wide range of features, including customer demographics, usage patterns, and service interactions, to determine their impact on churn.

**Key Questions to Be Addressed:**

1. What is the best model for predicting customer churn?

After comparing various models, including Decision Tree, K-Nearest Neighbors (KNN), and Random Forest, the analysis will recommend the best overall performer. By factoring in accuracy, precision, recall, and ROC-AUC score, the analysis will advise SyriaTel on the most suitable model for predicting churn and implementing targeted retention strategies.

1. How accurately can the model predict customer churn?

The analysis will evaluate the performance of various models using key metrics such as accuracy, precision, recall, and the ROC-AUC score. This evaluation will determine how well the models can predict which customers are likely to churn.

1. Which features are most influential in predicting customer churn?

Identifying the most impactful features, such as customer service interactions, usage patterns, and plan types, will help SyriaTel prioritize its retention efforts and design more effective interventions.

With these predictive insights, SyriaTel can move beyond merely understanding why customers churn to proactively identifying at-risk customers and intervening with targeted strategies. This shift will enable SyriaTel to not only reduce churn rates but also improve customer satisfaction and loyalty, leading to better financial outcomes for the company.

**Methodology for Machine Learning**

To develop a robust predictive model for identifying customers at risk of churning, we will follow a structured methodology that includes the following key steps:

1. Data Understanding
2. Data Cleaning
3. Exploratory Data Analysis
4. Data Preprocessing
5. Modelling
6. Hyperparameter Selection
7. Model Evaluation
8. Recommendations and Conclusion

**Step 1**: Data Understanding

1.1 Import the necessary libraries and modules for dealing with the dataset and its data

In [44]:

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score, cross\_validate, GridSearchCV  
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, FunctionTransformer  
from sklearn.metrics import recall\_score, accuracy\_score, precision\_score, f1\_score, confusion\_matrix, ConfusionMatrixDisplay, classification\_report  
import scipy.stats as stats  
import statsmodels as statsmd  
from sklearn.linear\_model import LogisticRegression  
from sklearn.compose import ColumnTransformer  
  
from imblearn.over\_sampling import SMOTE  
from imblearn.under\_sampling import RandomUnderSampler   
from sklearn.pipeline import Pipeline  
  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.dummy import DummyClassifier  
from sklearn.ensemble import RandomForestClassifier

1.2 Load the Dataset

In this project, the dataset that we chose is called **SyriaTel Customer Churn**.

In [45]:

# Load the dataset and examine its structure.  
file\_path = 'SyriaTel\_Data.csv'  
data = pd.read\_csv(file\_path)  
  
# Display the first few rows of the dataset  
data.head()

Out[45]:

|  | state | account length | area code | phone number | international plan | voice mail plan | number vmail messages | total day minutes | total day calls | total day charge | ... | total eve calls | total eve charge | total night minutes | total night calls | total night charge | total intl minutes | total intl calls | total intl charge | customer service calls | churn |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | ... | 99 | 16.78 | 244.7 | 91 | 11.01 | 10.0 | 3 | 2.70 | 1 | False |
| 1 | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | ... | 103 | 16.62 | 254.4 | 103 | 11.45 | 13.7 | 3 | 3.70 | 1 | False |
| 2 | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | ... | 110 | 10.30 | 162.6 | 104 | 7.32 | 12.2 | 5 | 3.29 | 0 | False |
| 3 | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | ... | 88 | 5.26 | 196.9 | 89 | 8.86 | 6.6 | 7 | 1.78 | 2 | False |
| 4 | OK | 75 | 415 | 330-6626 | yes | no | 0 | 166.7 | 113 | 28.34 | ... | 122 | 12.61 | 186.9 | 121 | 8.41 | 10.1 | 3 | 2.73 | 3 | False |

5 rows × 21 columns

1.3 Creates a function for viewing the columns in the dataset

In [46]:

def col\_info(data):  
 print('col\_names: \n', data.columns)  
 print('num\_cols: \n', data.select\_dtypes(int).columns)  
 print('cat\_cols: \n', data.select\_dtypes(object).columns)  
 print('float\_cols: \n', data.select\_dtypes(float))  
 print('bool\_cols : \n' ,data.select\_dtypes(bool).columns)  
  
col\_info(data)

col\_names:   
 Index(['state', 'account length', 'area code', 'phone number',  
 'international plan', 'voice mail plan', 'number vmail messages',  
 'total day minutes', 'total day calls', 'total day charge',  
 'total eve minutes', 'total eve calls', 'total eve charge',  
 'total night minutes', 'total night calls', 'total night charge',  
 'total intl minutes', 'total intl calls', 'total intl charge',  
 'customer service calls', 'churn'],  
 dtype='object')  
num\_cols:   
 Index(['account length', 'area code', 'number vmail messages',  
 'total day calls', 'total eve calls', 'total night calls',  
 'total intl calls', 'customer service calls'],  
 dtype='object')  
cat\_cols:   
 Index(['state', 'phone number', 'international plan', 'voice mail plan'], dtype='object')  
float\_cols:   
 total day minutes total day charge total eve minutes \  
0 265.1 45.07 197.4   
1 161.6 27.47 195.5   
2 243.4 41.38 121.2   
3 299.4 50.90 61.9   
4 166.7 28.34 148.3   
... ... ... ...   
3328 156.2 26.55 215.5   
3329 231.1 39.29 153.4   
3330 180.8 30.74 288.8   
3331 213.8 36.35 159.6   
3332 234.4 39.85 265.9   
  
 total eve charge total night minutes total night charge \  
0 16.78 244.7 11.01   
1 16.62 254.4 11.45   
2 10.30 162.6 7.32   
3 5.26 196.9 8.86   
4 12.61 186.9 8.41   
... ... ... ...   
3328 18.32 279.1 12.56   
3329 13.04 191.3 8.61   
3330 24.55 191.9 8.64   
3331 13.57 139.2 6.26   
3332 22.60 241.4 10.86   
  
 total intl minutes total intl charge   
0 10.0 2.70   
1 13.7 3.70   
2 12.2 3.29   
3 6.6 1.78   
4 10.1 2.73   
... ... ...   
3328 9.9 2.67   
3329 9.6 2.59   
3330 14.1 3.81   
3331 5.0 1.35   
3332 13.7 3.70   
  
[3333 rows x 8 columns]  
bool\_cols :   
 Index(['churn'], dtype='object')

We observe from the above dataset we observe that there are **21** columns, with a mix of categorical, numerical, and boolean data types. Here's a brief overview of the dataset:

Key Features:

\*\*A). Categorical Features:

1. **state**: Categorical variable indicating the state of the customer.
2. **phone number**: Categorical, the customer's phone number (likely not useful for modeling).
3. **international plan**: Categorical, whether the customer has an international plan (yes/no).
4. **voice mail plan**: Categorical, whether the customer has a voicemail plan (yes/no).

\*\*B). Numerical and Floating Features:

1. **total eve calls**: Integer, total number of calls during the evening.
2. **account length**: Integer, representing the duration of the customer's account in days.
3. **area code**: Integer, indicating the area code of the customer.
4. **total day calls**: Integer, total number of calls during the day.
5. **total night calls**: Integer, total number of calls during the night.
6. **number vmail messages**: Integer, the number of voicemail messages.
7. **customer service calls**: Integer, the number of calls to customer service.
8. **total intl calls**: Integer, total number of international calls.
9. **total day minutes**: Float, total minutes of calls during the day.
10. **total day charge**: Float, total charges for calls during the day.

15 **total eve minutes**: Float, total minutes of calls during the evening.

1. **total eve charge**: Float, total charges for calls during the evening.
2. **total night minutes**: Float, total minutes of calls during the night.
3. **total night charge**: Float, total charges for calls during the night.
4. **total intl minutes**: Float, total minutes of international calls.
5. **total intl charge**: Float, total charges for international calls.

\*\*C) . Boolean Features:

1. **churn**: Boolean, the target variable indicating whether the customer has churned (True) or not (False).

In [47]:

# Display basic information about the dataset  
print("\nDataset Info:")  
data.info()

Dataset Info:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3333 entries, 0 to 3332  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 state 3333 non-null object   
 1 account length 3333 non-null int64   
 2 area code 3333 non-null int64   
 3 phone number 3333 non-null object   
 4 international plan 3333 non-null object   
 5 voice mail plan 3333 non-null object   
 6 number vmail messages 3333 non-null int64   
 7 total day minutes 3333 non-null float64  
 8 total day calls 3333 non-null int64   
 9 total day charge 3333 non-null float64  
 10 total eve minutes 3333 non-null float64  
 11 total eve calls 3333 non-null int64   
 12 total eve charge 3333 non-null float64  
 13 total night minutes 3333 non-null float64  
 14 total night calls 3333 non-null int64   
 15 total night charge 3333 non-null float64  
 16 total intl minutes 3333 non-null float64  
 17 total intl calls 3333 non-null int64   
 18 total intl charge 3333 non-null float64  
 19 customer service calls 3333 non-null int64   
 20 churn 3333 non-null bool   
dtypes: bool(1), float64(8), int64(8), object(4)  
memory usage: 524.2+ KB

1.4 Checking for Missing Values Identify any missing values within the dataset.

Ensuring there are no missing values is crucial, as missing data can lead to inaccuracies in the model training process. If missing values are found, they need to be addressed appropriately.

In [48]:

# Check for missing values  
print("\nMissing Values in Each Column:")  
print(data.isnull().sum())

Missing Values in Each Column:  
state 0  
account length 0  
area code 0  
phone number 0  
international plan 0  
voice mail plan 0  
number vmail messages 0  
total day minutes 0  
total day calls 0  
total day charge 0  
total eve minutes 0  
total eve calls 0  
total eve charge 0  
total night minutes 0  
total night calls 0  
total night charge 0  
total intl minutes 0  
total intl calls 0  
total intl charge 0  
customer service calls 0  
churn 0  
dtype: int64

In [49]:

def cleaning(data):  
 missing = data.isna().sum().sum()  
 duplicates = data.duplicated().sum()  
 return (f"There are {missing} missing values and {duplicates} duplicated values in the dataset")  
  
cleaning(data)

Out[49]:

'There are 0 missing values and 0 duplicated values in the dataset'

In [50]:

# Checking for unique values  
data.nunique()

Out[50]:

state 51  
account length 212  
area code 3  
phone number 3333  
international plan 2  
voice mail plan 2  
number vmail messages 46  
total day minutes 1667  
total day calls 119  
total day charge 1667  
total eve minutes 1611  
total eve calls 123  
total eve charge 1440  
total night minutes 1591  
total night calls 120  
total night charge 933  
total intl minutes 162  
total intl calls 21  
total intl charge 162  
customer service calls 10  
churn 2  
dtype: int64

In [51]:

data.shape

Out[51]:

(3333, 21)

From this dataset, we can observe that there are **3,333** rows and **21** columns, from which it's distributed evenly. We can also observe that we didn't have any missing values or duplicated values in the dataset and this enabled us to conduct this project without any challenges.

**Step 2**: Data Cleaning

Now that we understand the structure of the data, we'll clean it by addressing any issues such as missing values or irrelevant columns.

2.1 Drop Irrelevant Features

Remove columns that are not useful for prediction, such as identifiers like the phone number.

In [52]:

# Drop irrelevant columns  
data = data.drop(columns=['phone number'])  
  
# Check the updated dataframe  
print("\nFirst Few Rows After Dropping Irrelevant Columns:")  
data.head()

First Few Rows After Dropping Irrelevant Columns:

Out[52]:

|  | state | account length | area code | international plan | voice mail plan | number vmail messages | total day minutes | total day calls | total day charge | total eve minutes | total eve calls | total eve charge | total night minutes | total night calls | total night charge | total intl minutes | total intl calls | total intl charge | customer service calls | churn |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | KS | 128 | 415 | no | yes | 25 | 265.1 | 110 | 45.07 | 197.4 | 99 | 16.78 | 244.7 | 91 | 11.01 | 10.0 | 3 | 2.70 | 1 | False |
| 1 | OH | 107 | 415 | no | yes | 26 | 161.6 | 123 | 27.47 | 195.5 | 103 | 16.62 | 254.4 | 103 | 11.45 | 13.7 | 3 | 3.70 | 1 | False |
| 2 | NJ | 137 | 415 | no | no | 0 | 243.4 | 114 | 41.38 | 121.2 | 110 | 10.30 | 162.6 | 104 | 7.32 | 12.2 | 5 | 3.29 | 0 | False |
| 3 | OH | 84 | 408 | yes | no | 0 | 299.4 | 71 | 50.90 | 61.9 | 88 | 5.26 | 196.9 | 89 | 8.86 | 6.6 | 7 | 1.78 | 2 | False |
| 4 | OK | 75 | 415 | yes | no | 0 | 166.7 | 113 | 28.34 | 148.3 | 122 | 12.61 | 186.9 | 121 | 8.41 | 10.1 | 3 | 2.73 | 3 | False |

**Step 3**: Exploratory Data Analysis (EDA)

3.1 Descriptive Statistics

In [53]:

# Display descriptive statistics  
print("\nDescriptive Statistics of the Dataset:")  
data.describe()

Descriptive Statistics of the Dataset:

Out[53]:

|  | account length | area code | number vmail messages | total day minutes | total day calls | total day charge | total eve minutes | total eve calls | total eve charge | total night minutes | total night calls | total night charge | total intl minutes | total intl calls | total intl charge | customer service calls |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 | 3333.000000 |
| mean | 101.064806 | 437.182418 | 8.099010 | 179.775098 | 100.435644 | 30.562307 | 200.980348 | 100.114311 | 17.083540 | 200.872037 | 100.107711 | 9.039325 | 10.237294 | 4.479448 | 2.764581 | 1.562856 |
| std | 39.822106 | 42.371290 | 13.688365 | 54.467389 | 20.069084 | 9.259435 | 50.713844 | 19.922625 | 4.310668 | 50.573847 | 19.568609 | 2.275873 | 2.791840 | 2.461214 | 0.753773 | 1.315491 |
| min | 1.000000 | 408.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 23.200000 | 33.000000 | 1.040000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 74.000000 | 408.000000 | 0.000000 | 143.700000 | 87.000000 | 24.430000 | 166.600000 | 87.000000 | 14.160000 | 167.000000 | 87.000000 | 7.520000 | 8.500000 | 3.000000 | 2.300000 | 1.000000 |
| 50% | 101.000000 | 415.000000 | 0.000000 | 179.400000 | 101.000000 | 30.500000 | 201.400000 | 100.000000 | 17.120000 | 201.200000 | 100.000000 | 9.050000 | 10.300000 | 4.000000 | 2.780000 | 1.000000 |
| 75% | 127.000000 | 510.000000 | 20.000000 | 216.400000 | 114.000000 | 36.790000 | 235.300000 | 114.000000 | 20.000000 | 235.300000 | 113.000000 | 10.590000 | 12.100000 | 6.000000 | 3.270000 | 2.000000 |
| max | 243.000000 | 510.000000 | 51.000000 | 350.800000 | 165.000000 | 59.640000 | 363.700000 | 170.000000 | 30.910000 | 395.000000 | 175.000000 | 17.770000 | 20.000000 | 20.000000 | 5.400000 | 9.000000 |

The table provides a summary of key statistics (count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum) for various features in the dataset. These statistics are crucial for understanding the distribution of the data and identifying potential outliers.

Key Observations:

**Range and Potential Outliers:**

* number vmail messages: The maximum value is 51, while the 75th percentile is 20, indicating that a small subset of users has a much higher number of voicemail messages, potentially marking them as outliers.
* total day minutes, total eve minutes, total night minutes, total intl minutes: These features have maximum values significantly higher than the 75th percentile. For example, total day minutes has a max of 350.8 minutes, whereas the 75th percentile is 216.4 minutes, suggesting the presence of outliers.
* customer service calls: The maximum number of calls is 9, with a median of 1 and a 75th percentile of 2. This suggests that while most customers have 1-2 service calls, some customers are outliers with significantly higher call counts.

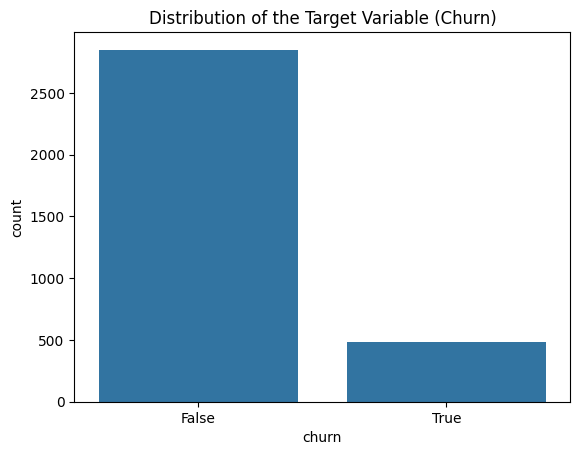
Conclusion:

* **Outliers**: Several features show potential outliers, particularly in the number vmail messages, total day minutes, total eve minutes, total night minutes, total intl minutes, and customer service calls fields. These outliers could significantly impact the model’s performance if not addressed.
* **Next Steps**: We will consider handling these outliers, possibly by capping extreme values, using robust scaling methods, or investigating the reasons behind these outlier behaviors. Additionally, further visualization (e.g., box plots) could help confirm and understand the distribution and impact of these outliers.

3.2 Distribution of the Target Variable (Churn)

In [54]:

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Plot distribution of the target variable  
sns.countplot(x='churn', data=data)  
plt.title('Distribution of the Target Variable (Churn)')  
plt.show()



The bar chart above shows the distribution of the target variable "churn," indicating whether customers have churned (left the service) or not.

Interpretation:

**False (Non-Churners): The taller bar represents customers who have not churned. This group is significantly larger, with over 2,500 customers.**

**True (Churners): The shorter bar represents customers who have churned. This group is much smaller, with fewer than 500 customers.**

Key Insights:

Class Imbalance: The chart highlights a clear class imbalance in the dataset. The majority of customers have not churned, while a relatively small number have. This imbalance is crucial to consider when building predictive models, as it can lead to a model that is biased toward predicting the majority class (non-churners).

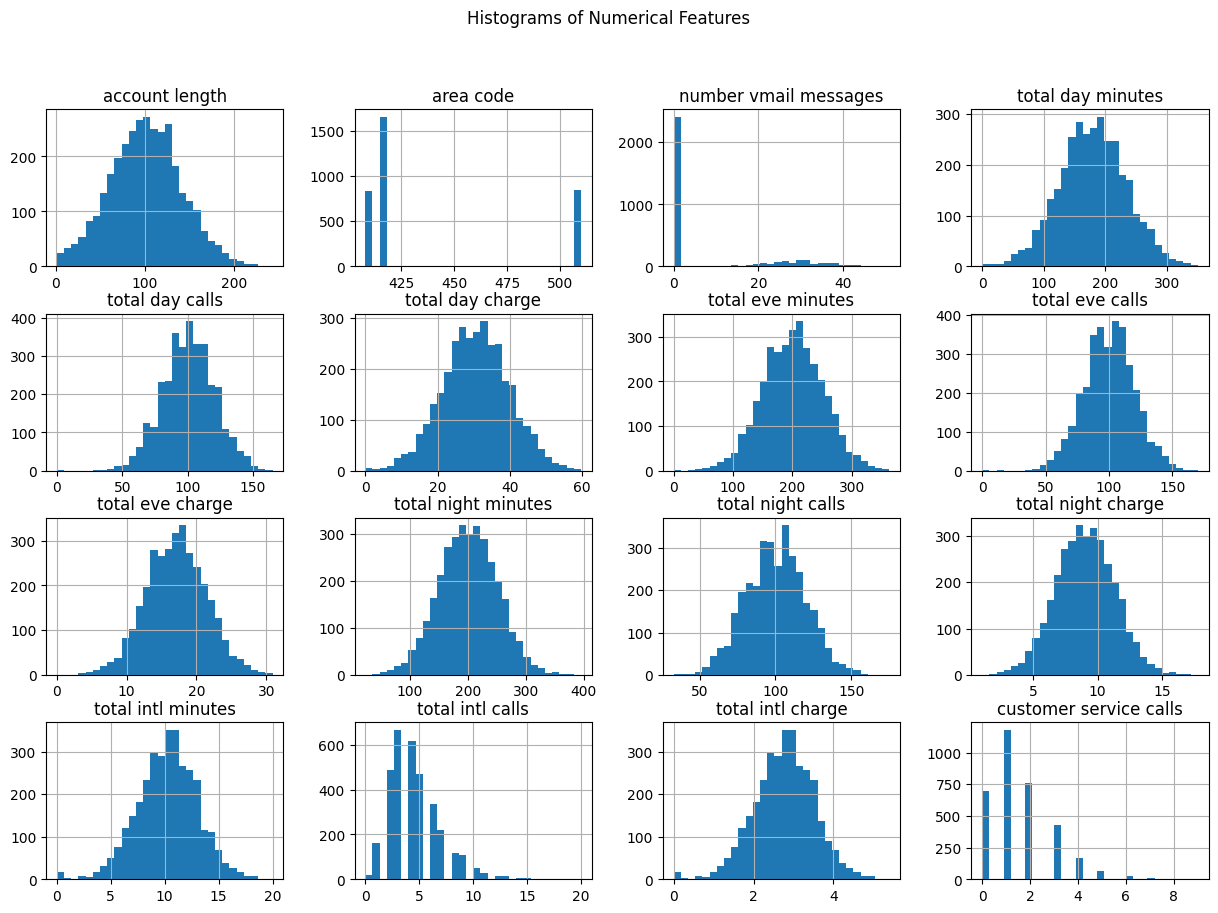
Handling Imbalance: Techniques such as oversampling the minority class (using SMOTE, for example) or undersampling the majority class may be necessary to ensure that the model accurately predicts both classes.

This imbalance will be addressed during the preprocessing or model training phase to improve the model's performance in predicting customer churn.

3.3 Visualize the Distribution of Numerical Features

In [55]:

# Plot histograms of numerical features  
data.hist(bins=30, figsize=(15, 10))  
plt.suptitle('Histograms of Numerical Features')  
plt.show()



Most features are normally distributed , so log transformation is not necessary for them. However,features with Skewed Distributions like Number Vmail Messages(This feature is heavily right-skewed, with most values concentrated at zero and a long tail of higher values.); Customer Service Calls(This feature is also right-skewed, with the majority of customers making very few calls, and a smaller number making a higher number of calls.) and Total Intl Calls( This feature shows some skewness, though it is less extreme than the first two.) we may consider log transformation

The image shows a series of histograms that represent the distribution of various numerical features in the dataset. Here’s the interpretation of each feature based on the histograms:

Key Insights:

Normal Distributions: Many features, such as total minutes, total calls, and corresponding charges, follow normal distributions. This suggests that most customers have similar usage patterns, with a typical range of values for these features.

Skewed Distributions: The "number vmail messages" and "customer service calls" features are skewed. Most customers do not use voicemail services much, and most do not frequently call customer service.

Area Code: The area code feature is categorical with three main groups, indicating that the dataset represents customers from three distinct regions.

Implications for Modeling:

Feature Scaling: Given the normal distribution of most features, feature scaling (e.g., standardization) will likely be beneficial for models that are sensitive to the scale of input data (e.g., logistic regression, SVM).

Handling Skewness: For skewed features like "number vmail messages" and "customer service calls," special attention might be needed. For example, you could apply log transformation to reduce skewness if using models that assume normally distributed data.

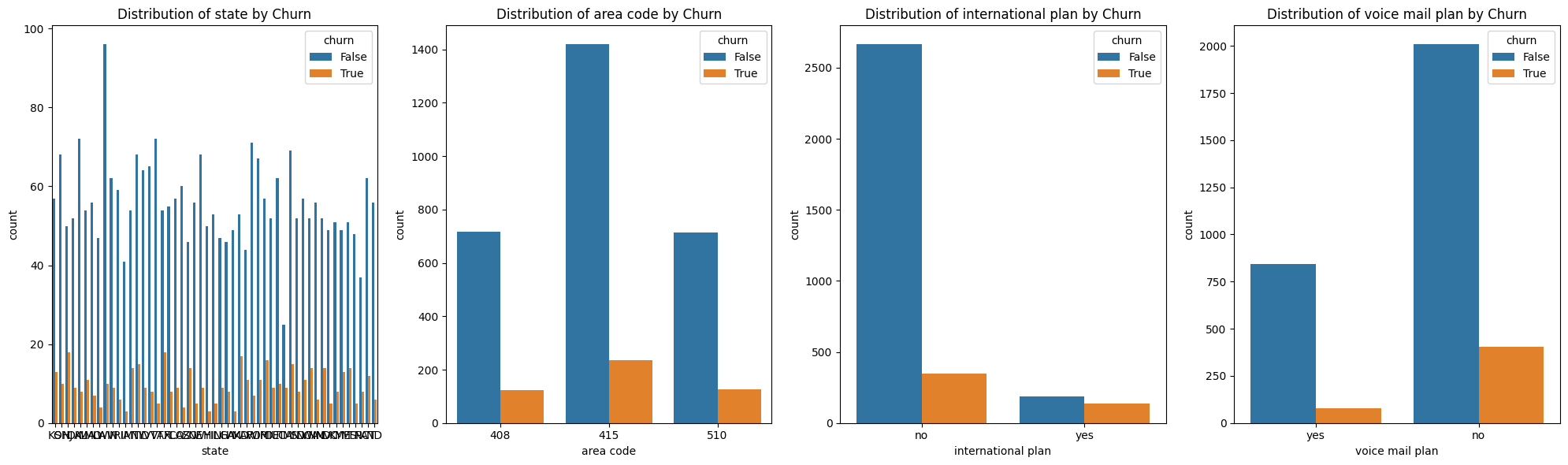
Categorical Handling: The area code, despite being numerical, is more categorical in nature and should be treated accordingly (e.g., using one-hot encoding) in the modeling process.

These insights can help in deciding which preprocessing steps to apply and in understanding the underlying patterns in the data.

3.4 Caterogical feature distributions

In [56]:

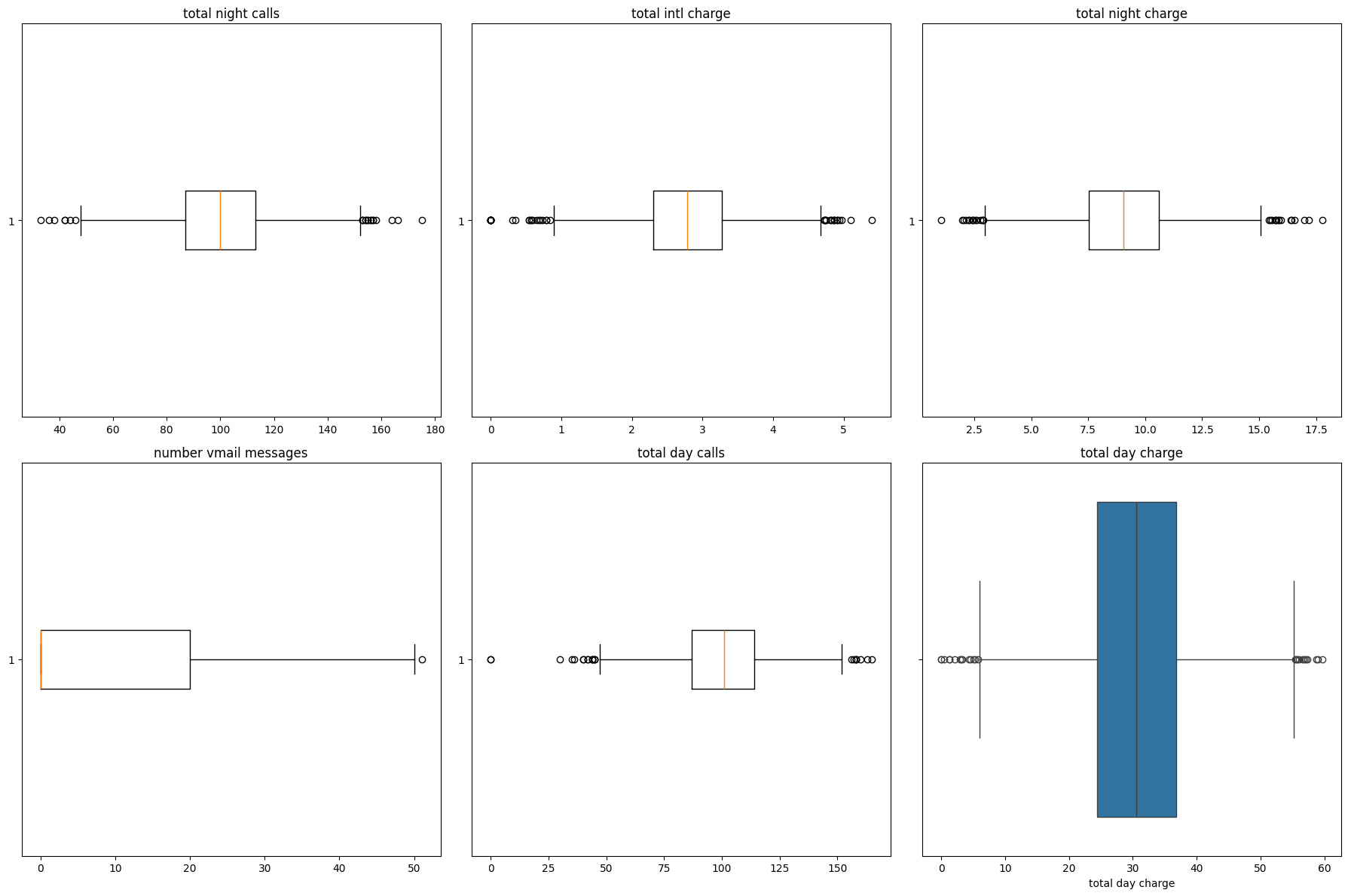
# List of categorical features  
categorical\_features = ['state','area code' ,'international plan', 'voice mail plan']  
  
# Create subplots: 1 row and 3 columns  
fig, axes = plt.subplots(nrows=1, ncols=len(categorical\_features), figsize=(20, 6))  
  
# Loop through each feature and create a count plot  
for i, feature in enumerate(categorical\_features):  
 sns.countplot(x=feature, hue='churn', data=data, ax=axes[i])  
 axes[i].set\_title(f'Distribution of {feature} by Churn')  
  
plt.tight\_layout()  
plt.show()



3.5 Checking for outliers

In [57]:

# Define the list of numerical columns  
num\_cols = ["total night calls", "total intl charge", "total night charge", "number vmail messages"]  
  
# Create a figure with multiple subplots  
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 12)) # Adjusting layout for all plots  
  
# Plot each of the specified columns in the first row and second row as needed  
for i, col in enumerate(num\_cols):  
 if i < 3: # Plot on the first row  
 axes[0, i].boxplot(data[col], vert=False)  
 axes[0, i].set\_title(col)  
 else: # Plot on the first position of the second row  
 axes[1, 0].boxplot(data[col], vert=False)  
 axes[1, 0].set\_title(col)  
  
# Plot the additional boxplots in the remaining positions of the second row  
axes[1, 1].boxplot(data['total day calls'], vert=False)  
axes[1, 1].set\_title('total day calls')  
  
sns.boxplot(data=data, x='total day charge', ax=axes[1, 2])  
axes[1, 2].set\_title('total day charge')  
  
plt.tight\_layout()  
plt.show()



**Summary of above visualisations:**

Outliers: All plots indicate the presence of outliers, especially in number of voicemail messages, total night calls, and total intl charge.

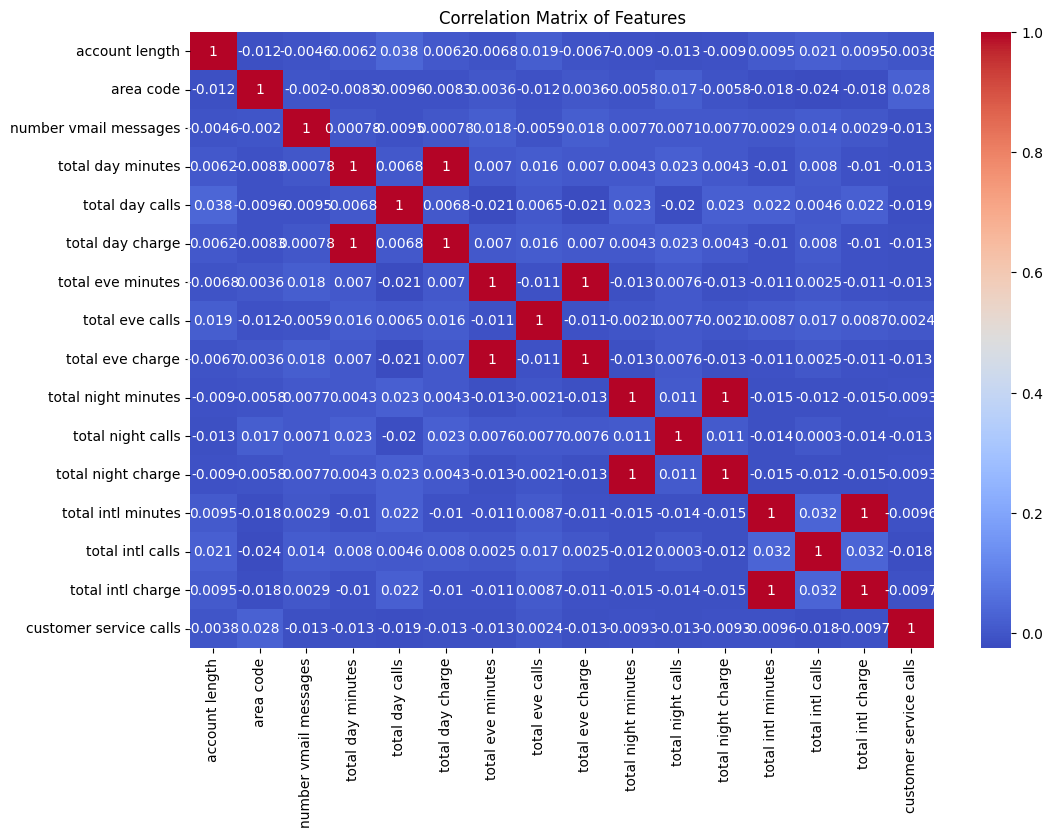
Distribution: Most features are symmetrically distributed around their medians, except for number of voicemail messages, which shows significant skewness.

Considerations: The outliers could impact the model's performance and may need to be addressed depending on their significance to the business problem.

3.6 Correlation Analysis

In [58]:

# Select only the numeric columns  
numeric\_df = data.select\_dtypes(include=[np.number])  
  
# Calculate the correlation matrix  
corr\_matrix = numeric\_df.corr()  
  
# Visualize the correlation matrix  
plt.figure(figsize=(12, 8))  
sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')  
plt.title('Correlation Matrix of Features')  
plt.show()



The correlation matrix visually represents the linear relationships between pairs of features in the dataset. Key observations include:

Perfect Correlations: Features like total day minutes and total day charge, total eve minutes and total eve charge, total night minutes and total night charge, and total intl minutes and total intl charge have perfect correlations (correlation coefficient = 1). This is expected because the charges are directly proportional to the minutes used.

Low Correlation with Target: Features such as customer service calls, account length, and number vmail messages have very low correlation with other features, suggesting they provide unique information.

Potential Redundancy: Features with perfect correlations might be redundant and could be candidates for removal in modeling to reduce multicollinearity.

This matrix is useful for identifying which features are highly correlated and may need to be handled carefully to avoid issues in model training, such as multicollinearity.

Potential Redundancy in Features: In the correlation matrix, the following pairs of features show a perfect correlation (correlation coefficient = 1). These pairs are redundant because one feature in each pair is a linear transformation of the other. When building models, you might consider removing one feature from each pair to reduce multicollinearity:

Total Day Minutes and Total Day Charge:

Both features are perfectly correlated (correlation = 1). Since charges are typically a direct function of minutes used, you can consider removing one of these features. Total Eve Minutes and Total Eve Charge:

These two features also have a perfect correlation. Removing one of these would reduce redundancy. Total Night Minutes and Total Night Charge:

Like the day and evening features, these are perfectly correlated, and one can be removed. Total Intl Minutes and Total Intl Charge:

These features are perfectly correlated as well, making one of them redundant.

Suggested Action: we will remove One Feature from Each Pair: For each of the above pairs, consider keeping only one feature (either the minutes or the charge) to simplify the model and avoid multicollinearity issues. Typically, you'd keep the feature that is more directly related to the business problem or has a more straightforward interpretation.

**Step 4:** Preprocessing

4.1 Train-Test Split

Split the data into training and testing sets before performing any transformations.

In [59]:

from sklearn.model\_selection import train\_test\_split  
  
# Split the data into features (X) and target (y)  
X = data.drop(columns=['churn'])  
y = data['churn']  
  
# Split 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)  
  
print(f"Training Set: {X\_train.shape}, {y\_train.shape}")  
print(f"Testing Set: {X\_test.shape}, {y\_test.shape}")

Training Set: (2666, 19), (2666,)  
Testing Set: (667, 19), (667,)

4.2 Encoding Categorical Variables

Now that the data is split, we proceed to encode the categorical variables in both the training and test sets.

4.2.1: One-Hot Encode Categorical Variables for Training Set

In [60]:

# One-hot encode categorical variables for training set  
X\_train\_encoded = pd.get\_dummies(X\_train, drop\_first=True)  
  
# Check the encoded training dataframe  
print("\nFirst Few Rows After Encoding Categorical Variables (Training Set):")  
print(X\_train\_encoded.head())

First Few Rows After Encoding Categorical Variables (Training Set):  
 account length area code number vmail messages total day minutes \  
817 243 510 0 95.5   
1373 108 415 0 112.0   
679 75 415 0 222.4   
56 141 415 0 126.9   
1993 86 510 0 216.3   
  
 total day calls total day charge total eve minutes total eve calls \  
817 92 16.24 163.7 63   
1373 105 19.04 193.7 110   
679 78 37.81 327.0 111   
56 98 21.57 180.0 62   
1993 96 36.77 266.3 77   
  
 total eve charge total night minutes ... state\_TX state\_UT \  
817 13.91 264.2 ... False True   
1373 16.46 208.9 ... False False   
679 27.80 208.0 ... True False   
56 15.30 140.8 ... False False   
1993 22.64 214.0 ... False False   
  
 state\_VA state\_VT state\_WA state\_WI state\_WV state\_WY \  
817 False False False False False False   
1373 False False False False False False   
679 False False False False False False   
56 False False False False False False   
1993 False False False False False False   
  
 international plan\_yes voice mail plan\_yes   
817 False False   
1373 False False   
679 True False   
56 False False   
1993 False False   
  
[5 rows x 68 columns]

4.2.2 : One-Hot Encode Categorical Variables for Testing Set

In [61]:

# One-hot encode categorical variables for test set  
X\_test\_encoded = pd.get\_dummies(X\_test, drop\_first=True)  
  
# Check the encoded testing dataframe  
print("\nFirst Few Rows After Encoding Categorical Variables (Testing Set):")  
print(X\_test\_encoded.head())

First Few Rows After Encoding Categorical Variables (Testing Set):  
 account length area code number vmail messages total day minutes \  
438 113 510 0 155.0   
2674 67 415 0 109.1   
1345 98 415 0 0.0   
1957 147 408 0 212.8   
2148 96 408 0 144.0   
  
 total day calls total day charge total eve minutes total eve calls \  
438 93 26.35 330.6 106   
2674 117 18.55 217.4 124   
1345 0 0.00 159.6 130   
1957 79 36.18 204.1 91   
2148 102 24.48 224.7 73   
  
 total eve charge total night minutes ... state\_TX state\_UT \  
438 28.10 189.4 ... False False   
2674 18.48 188.4 ... False False   
1345 13.57 167.1 ... False False   
1957 17.35 156.2 ... False False   
2148 19.10 227.7 ... False False   
  
 state\_VA state\_VT state\_WA state\_WI state\_WV state\_WY \  
438 False False False False False True   
2674 False False False False False False   
1345 False False False False False False   
1957 False False False False False False   
2148 False False False False False True   
  
 international plan\_yes voice mail plan\_yes   
438 False False   
2674 False False   
1345 False False   
1957 False False   
2148 False False   
  
[5 rows x 68 columns]

In [62]:

# One-hot encode categorical variables for training set  
X\_train\_encoded = pd.get\_dummies(X\_train, drop\_first=True)  
  
# One-hot encode categorical variables for test set  
X\_test\_encoded = pd.get\_dummies(X\_test, drop\_first=True)  
  
# Align the test set with the training set columns (handle any missing columns in test set)  
X\_train\_encoded, X\_test\_encoded = X\_train\_encoded.align(X\_test\_encoded, join='left', axis=1, fill\_value=0)  
  
# Check the encoded training dataframe  
print("\nFirst Few Rows After Encoding Categorical Variables (Training Set):")  
print(X\_train\_encoded.head())  
  
# Check the encoded testing dataframe  
print("\nFirst Few Rows After Encoding Categorical Variables (Testing Set):")  
  
X\_test\_encoded.head()

First Few Rows After Encoding Categorical Variables (Training Set):  
 account length area code number vmail messages total day minutes \  
817 243 510 0 95.5   
1373 108 415 0 112.0   
679 75 415 0 222.4   
56 141 415 0 126.9   
1993 86 510 0 216.3   
  
 total day calls total day charge total eve minutes total eve calls \  
817 92 16.24 163.7 63   
1373 105 19.04 193.7 110   
679 78 37.81 327.0 111   
56 98 21.57 180.0 62   
1993 96 36.77 266.3 77   
  
 total eve charge total night minutes ... state\_TX state\_UT \  
817 13.91 264.2 ... False True   
1373 16.46 208.9 ... False False   
679 27.80 208.0 ... True False   
56 15.30 140.8 ... False False   
1993 22.64 214.0 ... False False   
  
 state\_VA state\_VT state\_WA state\_WI state\_WV state\_WY \  
817 False False False False False False   
1373 False False False False False False   
679 False False False False False False   
56 False False False False False False   
1993 False False False False False False   
  
 international plan\_yes voice mail plan\_yes   
817 False False   
1373 False False   
679 True False   
56 False False   
1993 False False   
  
[5 rows x 68 columns]  
  
First Few Rows After Encoding Categorical Variables (Testing Set):

Out[62]:

|  | account length | area code | number vmail messages | total day minutes | total day calls | total day charge | total eve minutes | total eve calls | total eve charge | total night minutes | ... | state\_TX | state\_UT | state\_VA | state\_VT | state\_WA | state\_WI | state\_WV | state\_WY | international plan\_yes | voice mail plan\_yes |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 438 | 113 | 510 | 0 | 155.0 | 93 | 26.35 | 330.6 | 106 | 28.10 | 189.4 | ... | False | False | False | False | False | False | False | True | False | False |
| 2674 | 67 | 415 | 0 | 109.1 | 117 | 18.55 | 217.4 | 124 | 18.48 | 188.4 | ... | False | False | False | False | False | False | False | False | False | False |
| 1345 | 98 | 415 | 0 | 0.0 | 0 | 0.00 | 159.6 | 130 | 13.57 | 167.1 | ... | False | False | False | False | False | False | False | False | False | False |
| 1957 | 147 | 408 | 0 | 212.8 | 79 | 36.18 | 204.1 | 91 | 17.35 | 156.2 | ... | False | False | False | False | False | False | False | False | False | False |
| 2148 | 96 | 408 | 0 | 144.0 | 102 | 24.48 | 224.7 | 73 | 19.10 | 227.7 | ... | False | False | False | False | False | False | False | True | False | False |

5 rows × 68 columns

4.2.3 : Align the Test Set with the Training Set Columns After encoding, it's important to align the test set with the training set to ensure that both datasets have the same features. Any missing columns in the test set will be added and filled with zeros.

In [63]:

# Align the test set with the training set columns (handle any missing columns in test set)  
X\_train\_encoded, X\_test\_encoded = X\_train\_encoded.align(X\_test\_encoded, join='left', axis=1, fill\_value=0)  
  
# Check the aligned training dataframe  
print("\nFirst Few Rows After Aligning Categorical Variables (Training Set):")  
print(X\_train\_encoded.head())  
  
# Check the aligned testing dataframe  
print("\nFirst Few Rows After Aligning Categorical Variables (Testing Set):")  
print(X\_test\_encoded.head())

First Few Rows After Aligning Categorical Variables (Training Set):  
 account length area code number vmail messages total day minutes \  
817 243 510 0 95.5   
1373 108 415 0 112.0   
679 75 415 0 222.4   
56 141 415 0 126.9   
1993 86 510 0 216.3   
  
 total day calls total day charge total eve minutes total eve calls \  
817 92 16.24 163.7 63   
1373 105 19.04 193.7 110   
679 78 37.81 327.0 111   
56 98 21.57 180.0 62   
1993 96 36.77 266.3 77   
  
 total eve charge total night minutes ... state\_TX state\_UT \  
817 13.91 264.2 ... False True   
1373 16.46 208.9 ... False False   
679 27.80 208.0 ... True False   
56 15.30 140.8 ... False False   
1993 22.64 214.0 ... False False   
  
 state\_VA state\_VT state\_WA state\_WI state\_WV state\_WY \  
817 False False False False False False   
1373 False False False False False False   
679 False False False False False False   
56 False False False False False False   
1993 False False False False False False   
  
 international plan\_yes voice mail plan\_yes   
817 False False   
1373 False False   
679 True False   
56 False False   
1993 False False   
  
[5 rows x 68 columns]  
  
First Few Rows After Aligning Categorical Variables (Testing Set):  
 account length area code number vmail messages total day minutes \  
438 113 510 0 155.0   
2674 67 415 0 109.1   
1345 98 415 0 0.0   
1957 147 408 0 212.8   
2148 96 408 0 144.0   
  
 total day calls total day charge total eve minutes total eve calls \  
438 93 26.35 330.6 106   
2674 117 18.55 217.4 124   
1345 0 0.00 159.6 130   
1957 79 36.18 204.1 91   
2148 102 24.48 224.7 73   
  
 total eve charge total night minutes ... state\_TX state\_UT \  
438 28.10 189.4 ... False False   
2674 18.48 188.4 ... False False   
1345 13.57 167.1 ... False False   
1957 17.35 156.2 ... False False   
2148 19.10 227.7 ... False False   
  
 state\_VA state\_VT state\_WA state\_WI state\_WV state\_WY \  
438 False False False False False True   
2674 False False False False False False   
1345 False False False False False False   
1957 False False False False False False   
2148 False False False False False True   
  
 international plan\_yes voice mail plan\_yes   
438 False False   
2674 False False   
1345 False False   
1957 False False   
2148 False False   
  
[5 rows x 68 columns]

Why This Approach is Important: Consistency: Ensures that the training and test sets have the same features, which is crucial for applying machine learning models. Handling Missing Categories: In some cases, the test set might not have all the categories present in the training set. Aligning ensures that these differences do not cause errors during model training or prediction.

4.3 Feature Scaling

Scale the features in both the training and test sets using the same Scaler to ensure consistency.

4.3.1 Scaling the Training Set

Fit the Scaler: The StandardScaler is first fitted to the X\_train\_encoded data. This means that the scaler calculates the mean and standard deviation of each feature in the training set.

Transform the Training Data: After fitting, the scaler transforms the training data, standardizing each feature to have a mean of 0 and a standard deviation of 1.

Convert to DataFrame: The transformed data is converted back into a DataFrame for easier inspection and to maintain the column names.

In [64]:

from sklearn.preprocessing import StandardScaler  
import pandas as pd  
  
# Initialize the scaler  
scaler = StandardScaler()  
  
# Fit the scaler on the training data and transform it  
X\_train\_scaled = scaler.fit\_transform(X\_train\_encoded)  
  
# Convert the scaled features back to a DataFrame (optional, for easier inspection)  
X\_train\_scaled = pd.DataFrame(X\_train\_scaled, columns=X\_train\_encoded.columns)  
  
# Check the scaled training dataframe  
print("\nFirst Few Rows of Scaled Training Data:")  
print(X\_train\_scaled.head())

First Few Rows of Scaled Training Data:  
 account length area code number vmail messages total day minutes \  
0 3.601382 1.735840 -0.584936 -1.547653   
1 0.184951 -0.517168 -0.584936 -1.244014   
2 -0.650176 -0.517168 -0.584936 0.787609   
3 1.020079 -0.517168 -0.584936 -0.969818   
4 -0.371801 1.735840 -0.584936 0.675354   
  
 total day calls total day charge total eve minutes total eve calls \  
0 -0.429657 -1.547170 -0.729987 -1.840891   
1 0.224176 -1.244071 -0.138082 0.499864   
2 -1.133785 0.787772 2.491952 0.549667   
3 -0.127888 -0.970200 -0.408385 -1.890695   
4 -0.228477 0.675192 1.294330 -1.143645   
  
 total eve charge total night minutes ... state\_TX state\_UT state\_VA \  
0 -0.731087 1.255804 ... -0.150437 6.705633 -0.154303   
1 -0.139179 0.165090 ... -0.150437 -0.149128 -0.154303   
2 2.493068 0.147339 ... 6.647288 -0.149128 -0.154303   
3 -0.408439 -1.178086 ... -0.150437 -0.149128 -0.154303   
4 1.295326 0.265680 ... -0.150437 -0.149128 -0.154303   
  
 state\_VT state\_WA state\_WI state\_WV state\_WY international plan\_yes \  
0 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
1 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
2 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 3.061624   
3 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
4 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
  
 voice mail plan\_yes   
0 -0.611162   
1 -0.611162   
2 -0.611162   
3 -0.611162   
4 -0.611162   
  
[5 rows x 68 columns]

4.3.2 Scaling the Testing Set

Transform the Testing Data: Using the same scaler (which is already fitted to the training data), the test data is transformed. This ensures that the test data is scaled in the same way as the training data.

Convert to DataFrame: Again, the transformed test data is converted into a DataFrame for easier inspection.

In [65]:

# Use the same scaler to transform the test data (without refitting)  
X\_test\_scaled = scaler.transform(X\_test\_encoded)  
  
# Convert the scaled features back to a DataFrame (optional, for easier inspection)  
X\_test\_scaled = pd.DataFrame(X\_test\_scaled, columns=X\_test\_encoded.columns)  
  
# Check the scaled testing dataframe  
print("\nFirst Few Rows of Scaled Testing Data:")  
print(X\_test\_scaled.head())

First Few Rows of Scaled Testing Data:  
 account length area code number vmail messages total day minutes \  
0 0.311486 1.735840 -0.584936 -0.452712   
1 -0.852632 -0.517168 -0.584936 -1.297381   
2 -0.068118 -0.517168 -0.584936 -3.305080   
3 1.171920 -0.683179 -0.584936 0.610946   
4 -0.118732 -0.683179 -0.584936 -0.655138   
  
 total day calls total day charge total eve minutes total eve calls \  
0 -0.379362 -0.452767 2.562980 0.300651   
1 0.827714 -1.297113 0.329524 1.197110   
2 -5.056782 -3.305141 -0.810881 1.495930   
3 -1.083490 0.611325 0.067112 -0.446399   
4 0.073292 -0.655194 0.473554 -1.342858   
  
 total eve charge total night minutes ... state\_TX state\_UT state\_VA \  
0 2.562705 -0.219520 ... -0.150437 -0.149128 -0.154303   
1 0.329704 -0.239243 ... -0.150437 -0.149128 -0.154303   
2 -0.810008 -0.659356 ... -0.150437 -0.149128 -0.154303   
3 0.067408 -0.874343 ... -0.150437 -0.149128 -0.154303   
4 0.473619 0.535893 ... -0.150437 -0.149128 -0.154303   
  
 state\_VT state\_WA state\_WI state\_WV state\_WY international plan\_yes \  
0 -0.145137 -0.147809 -0.161784 -0.180369 6.534900 -0.326624   
1 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
2 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
3 -0.145137 -0.147809 -0.161784 -0.180369 -0.153025 -0.326624   
4 -0.145137 -0.147809 -0.161784 -0.180369 6.534900 -0.326624   
  
 voice mail plan\_yes   
0 -0.611162   
1 -0.611162   
2 -0.611162   
3 -0.611162   
4 -0.611162   
  
[5 rows x 68 columns]

**Step 5**: Removing Redundant Features

5.1 Removing Redundant Features from the Training Set

In [66]:

# List of redundant features to remove  
redundant\_features = ['total day charge', 'total eve charge', 'total night charge', 'total intl charge','number vmail messages']  
  
# Drop these features from the training set  
X\_train\_scaled = X\_train\_scaled.drop(columns=redundant\_features)  
  
# Check the remaining features in the training set  
print("\nRemaining Features After Removing Redundancy (Training Set):")  
print(X\_train\_scaled.columns)

Remaining Features After Removing Redundancy (Training Set):  
Index(['account length', 'area code', 'total day minutes', 'total day calls',  
 'total eve minutes', 'total eve calls', 'total night minutes',  
 'total night calls', 'total intl minutes', 'total intl calls',  
 'customer service calls', 'state\_AL', 'state\_AR', 'state\_AZ',  
 'state\_CA', 'state\_CO', 'state\_CT', 'state\_DC', 'state\_DE', 'state\_FL',  
 'state\_GA', 'state\_HI', 'state\_IA', 'state\_ID', 'state\_IL', 'state\_IN',  
 'state\_KS', 'state\_KY', 'state\_LA', 'state\_MA', 'state\_MD', 'state\_ME',  
 'state\_MI', 'state\_MN', 'state\_MO', 'state\_MS', 'state\_MT', 'state\_NC',  
 'state\_ND', 'state\_NE', 'state\_NH', 'state\_NJ', 'state\_NM', 'state\_NV',  
 'state\_NY', 'state\_OH', 'state\_OK', 'state\_OR', 'state\_PA', 'state\_RI',  
 'state\_SC', 'state\_SD', 'state\_TN', 'state\_TX', 'state\_UT', 'state\_VA',  
 'state\_VT', 'state\_WA', 'state\_WI', 'state\_WV', 'state\_WY',  
 'international plan\_yes', 'voice mail plan\_yes'],  
 dtype='object')

In [67]:

print(X\_train\_scaled.shape)

(2666, 63)

5.2 Removing Redundant Features from the Testing Set

In [68]:

# Drop these features from the testing set  
X\_test\_scaled = X\_test\_scaled.drop(columns=redundant\_features)  
  
# Check the remaining features in the testing set  
print("\nRemaining Features After Removing Redundancy (Testing Set):")  
print(X\_test\_scaled.columns)

Remaining Features After Removing Redundancy (Testing Set):  
Index(['account length', 'area code', 'total day minutes', 'total day calls',  
 'total eve minutes', 'total eve calls', 'total night minutes',  
 'total night calls', 'total intl minutes', 'total intl calls',  
 'customer service calls', 'state\_AL', 'state\_AR', 'state\_AZ',  
 'state\_CA', 'state\_CO', 'state\_CT', 'state\_DC', 'state\_DE', 'state\_FL',  
 'state\_GA', 'state\_HI', 'state\_IA', 'state\_ID', 'state\_IL', 'state\_IN',  
 'state\_KS', 'state\_KY', 'state\_LA', 'state\_MA', 'state\_MD', 'state\_ME',  
 'state\_MI', 'state\_MN', 'state\_MO', 'state\_MS', 'state\_MT', 'state\_NC',  
 'state\_ND', 'state\_NE', 'state\_NH', 'state\_NJ', 'state\_NM', 'state\_NV',  
 'state\_NY', 'state\_OH', 'state\_OK', 'state\_OR', 'state\_PA', 'state\_RI',  
 'state\_SC', 'state\_SD', 'state\_TN', 'state\_TX', 'state\_UT', 'state\_VA',  
 'state\_VT', 'state\_WA', 'state\_WI', 'state\_WV', 'state\_WY',  
 'international plan\_yes', 'voice mail plan\_yes'],  
 dtype='object')

In [69]:

print(X\_test\_scaled.shape)

(667, 63)

5.3 Testing for Multicollinearity Using VIF

In [70]:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
import pandas as pd  
  
# Create a DataFrame to hold the VIF values  
vif\_data = pd.DataFrame()  
vif\_data['Feature'] = X\_train\_scaled.columns  
  
# Calculate VIF for each feature  
vif\_data['VIF'] = [variance\_inflation\_factor(X\_train\_scaled.values, i) for i in range(X\_train\_scaled.shape[1])]  
  
# Display the VIF values  
print("\nVariance Inflation Factors (VIF):")  
print(vif\_data)

Variance Inflation Factors (VIF):  
 Feature VIF  
0 account length 1.019794  
1 area code 1.023248  
2 total day minutes 1.025656  
3 total day calls 1.027201  
4 total eve minutes 1.027969  
.. ... ...  
58 state\_WI 2.522159  
59 state\_WV 2.876089  
60 state\_WY 2.376523  
61 international plan\_yes 1.037558  
62 voice mail plan\_yes 1.015175  
  
[63 rows x 2 columns]

Key Insights:

No Significant Multicollinearity:

The low VIF values across all features suggest that there is no significant multicollinearity in your dataset. This is a positive result, as it means that your features are not excessively correlated with each other, and the model should not suffer from issues related to multicollinearity, such as inflated standard errors or unstable coefficients.

Model Stability: With VIF values this low, you can expect the model's coefficients to be more stable, leading to more reliable and interpretable results.

**Step 6**: Handling Class Imbalance

We saw in 3.2 above when visualising distribution of the Target Variable (Churn), there was class imbalance, so we apply SMOTE or another resampling technique only on the training data.

In [71]:

from imblearn.over\_sampling import SMOTE  
  
# Initialize SMOTE  
smote = SMOTE(random\_state=42)  
  
# Apply SMOTE to the scaled training data  
X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train\_scaled, y\_train)  
  
print(f"After SMOTE: {X\_train\_smote.shape}, {y\_train\_smote.shape}")

After SMOTE: (4568, 63), (4568,)

We notice that the Original Rows (Before SMOTE) were : 3,333

After SMOTE:

SMOTE added synthetic rows to the minority class to balance the dataset. The resulting dataset now has 4,568 rows. The total number of rows increased because the minority class was undersampled compared to the majority class, and SMOTE balanced it by generating new, synthetic samples.

**Step 7** Modeling

7.1 : K-Nearest Neighbors (KNN)-Model 1( Baseline Model)

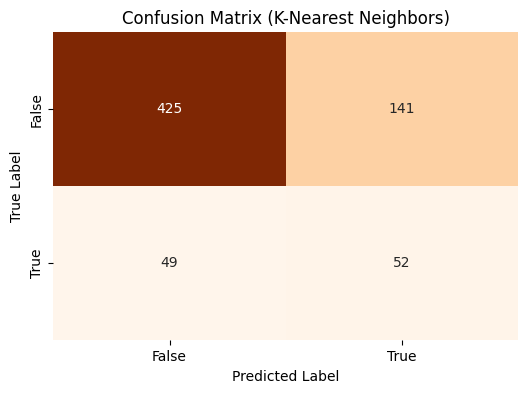
In [72]:

from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score  
from sklearn.model\_selection import cross\_validate  
  
# Initialize the KNN model  
knn = KNeighborsClassifier()  
  
# Define the scoring metrics for cross-validation  
scoring = {  
 'accuracy': 'accuracy',  
 'f1': 'f1',  
 'roc\_auc': 'roc\_auc',  
 'precision': 'precision',  
}  
  
# Perform 5-fold cross-validation on the SMOTE-processed training data  
cv\_results = cross\_validate(knn, X\_train\_smote, y\_train\_smote, cv=5, scoring=scoring)  
  
# Display cross-validation results  
print("Cross-Validation Scores (K-Nearest Neighbors):")  
print("Accuracy:", cv\_results['test\_accuracy'])  
print("Mean Accuracy:", cv\_results['test\_accuracy'].mean())  
print("F1-Score:", cv\_results['test\_f1'])  
print("Mean F1-Score:", cv\_results['test\_f1'].mean())  
print("ROC-AUC Score:", cv\_results['test\_roc\_auc'])  
print("Mean ROC-AUC Score:", cv\_results['test\_roc\_auc'].mean())  
print("Precision:", cv\_results['test\_precision'])  
print("Mean Precision:", cv\_results['test\_precision'].mean())  
  
# Now train the model on the entire SMOTE-processed training data  
knn.fit(X\_train\_smote, y\_train\_smote)  
  
# Predict on the test data  
y\_pred\_knn = knn.predict(X\_test\_scaled)  
  
# Evaluate the model on the test set  
print("\nFinal Model Evaluation on Test Set (K-Nearest Neighbors):")  
print("\nConfusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred\_knn))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_knn))  
print("\nROC-AUC Score:")  
print(roc\_auc\_score(y\_test, y\_pred\_knn))

Cross-Validation Scores (K-Nearest Neighbors):  
Accuracy: [0.85557987 0.84135667 0.83479212 0.83789704 0.84337349]  
Mean Accuracy: 0.8425998403800202  
F1-Score: [0.87109375 0.8622982 0.85660019 0.8585086 0.86183575]  
Mean F1-Score: 0.8620672977127647  
ROC-AUC Score: [0.94164205 0.94316468 0.94403612 0.94440765 0.94824897]  
Mean ROC-AUC Score: 0.944299895215153  
Precision: [0.78659612 0.76174497 0.75671141 0.76101695 0.7716263 ]  
Mean Precision: 0.7675391484997554  
  
Final Model Evaluation on Test Set (K-Nearest Neighbors):  
  
Confusion Matrix:  
[[425 141]  
 [ 49 52]]  
  
Classification Report:  
 precision recall f1-score support  
  
 False 0.90 0.75 0.82 566  
 True 0.27 0.51 0.35 101  
  
 accuracy 0.72 667  
 macro avg 0.58 0.63 0.59 667  
weighted avg 0.80 0.72 0.75 667  
  
  
ROC-AUC Score:  
0.6328674386873316

In [73]:

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Confusion matrix for K-Nearest Neighbors  
conf\_matrix\_knn = [[425, 141], [49, 52]]  
  
# Summary statistics for K-Nearest Neighbors  
summary\_data\_knn = {  
 'Model': 'K-Nearest Neighbors', 'Mean Accuracy': 0.84, 'Mean F1-Score': 0.86, 'Mean ROC-AUC Score': 0.94,   
 'Mean Precision': 0.77, 'Test Accuracy': 0.72, 'Test F1-Score': 0.35, 'Test ROC-AUC Score': 0.63,   
 'Test Precision': 0.27, 'Test Recall': 0.51  
}  
  
# Plotting Confusion Matrix for K-Nearest Neighbors  
plt.figure(figsize=(6, 4))  
sns.heatmap(conf\_matrix\_knn, annot=True, fmt='d', cmap='Oranges', cbar=False)  
plt.title('Confusion Matrix (K-Nearest Neighbors)')  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.xticks([0.5, 1.5], ['False', 'True'])  
plt.yticks([0.5, 1.5], ['False', 'True'])  
plt.show()  
  
# Display the summary statistics DataFrame for K-Nearest Neighbors  
summary\_df\_knn = pd.DataFrame([summary\_data\_knn])  
print("Summary Statistics for K-Nearest Neighbors:")  
display(summary\_df\_knn.round(2))



Summary Statistics for K-Nearest Neighbors:

|  | Model | Mean Accuracy | Mean F1-Score | Mean ROC-AUC Score | Mean Precision | Test Accuracy | Test F1-Score | Test ROC-AUC Score | Test Precision | Test Recall |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | K-Nearest Neighbors | 0.84 | 0.86 | 0.94 | 0.77 | 0.72 | 0.35 | 0.63 | 0.27 | 0.51 |

The K-Nearest Neighbors (K-NN) model performs well during cross-validation with high mean accuracy (0.84), F1-Score (0.86), and ROC-AUC score (0.94), indicating strong classification ability on the training data. However, its performance drops significantly on the test set, with test accuracy falling to 0.72, F1-Score to 0.35, and ROC-AUC to 0.63, suggesting poor generalization and possible overfitting. The decrease in test precision (0.27) and recall (0.51) further highlights the model's struggle with unseen data. To address this, we will consider tuning hyperparameters, using more robust validation methods but we will first explore alternative models that may offer better generalization

7.2 : Logistic Regression ( Model 2)

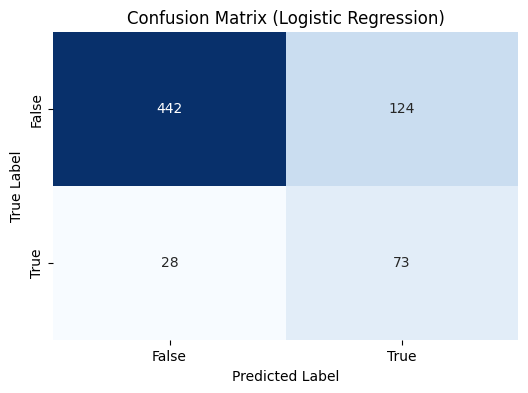
In [74]:

from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score  
from sklearn.model\_selection import cross\_validate  
  
# Initialize the Logistic Regression model  
log\_reg = LogisticRegression(random\_state=42)  
  
# Define the scoring metrics for cross-validation  
scoring = {  
 'accuracy': 'accuracy',  
 'f1': 'f1',  
 'roc\_auc': 'roc\_auc',  
 'precision': 'precision',  
}  
  
# Perform 5-fold cross-validation on the SMOTE-processed training data  
cv\_results = cross\_validate(log\_reg, X\_train\_smote, y\_train\_smote, cv=5, scoring=scoring)  
  
# Display cross-validation results  
print("Cross-Validation Scores (Logistic Regression):")  
print("Accuracy:", cv\_results['test\_accuracy'])  
print("Mean Accuracy:", cv\_results['test\_accuracy'].mean())  
print("F1-Score:", cv\_results['test\_f1'])  
print("Mean F1-Score:", cv\_results['test\_f1'].mean())  
print("ROC-AUC Score:", cv\_results['test\_roc\_auc'])  
print("Mean ROC-AUC Score:", cv\_results['test\_roc\_auc'].mean())  
print("Precision:", cv\_results['test\_precision'])  
print("Mean Precision:", cv\_results['test\_precision'].mean())  
  
# Now train the model on the entire SMOTE-processed training data  
log\_reg.fit(X\_train\_smote, y\_train\_smote)  
  
# Predict on the test data  
y\_pred\_log\_reg = log\_reg.predict(X\_test\_scaled)  
  
# Evaluate the model on the test set  
print("\nFinal Model Evaluation on Test Set (Logistic Regression):")  
print("\nConfusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred\_log\_reg))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_log\_reg))  
print("\nROC-AUC Score:")  
print(roc\_auc\_score(y\_test, y\_pred\_log\_reg))

Cross-Validation Scores (Logistic Regression):  
Accuracy: [0.75601751 0.80415755 0.78446389 0.79299014 0.81270537]  
Mean Accuracy: 0.7900668917963479  
F1-Score: [0.75521405 0.80690399 0.78975454 0.79872204 0.81750267]  
Mean F1-Score: 0.7936194580868914  
ROC-AUC Score: [0.83116989 0.86019086 0.8562119 0.84016181 0.87023494]  
Mean ROC-AUC Score: 0.8515938800911564  
Precision: [0.75770925 0.79574468 0.77083333 0.77639752 0.79791667]  
Mean Precision: 0.7797202894960671  
  
Final Model Evaluation on Test Set (Logistic Regression):  
  
Confusion Matrix:  
[[442 124]  
 [ 28 73]]  
  
Classification Report:  
 precision recall f1-score support  
  
 False 0.94 0.78 0.85 566  
 True 0.37 0.72 0.49 101  
  
 accuracy 0.77 667  
 macro avg 0.66 0.75 0.67 667  
weighted avg 0.85 0.77 0.80 667  
  
  
ROC-AUC Score:  
0.7518455025714585

In [75]:

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Confusion matrix for Logistic Regression  
conf\_matrix\_lr = [[442, 124], [28, 73]]  
  
# Summary statistics for Logistic Regression  
summary\_data\_lr = {  
 'Model': 'Logistic Regression', 'Mean Accuracy': 0.79, 'Mean F1-Score': 0.79, 'Mean ROC-AUC Score': 0.85,   
 'Mean Precision': 0.78, 'Test Accuracy': 0.77, 'Test F1-Score': 0.49, 'Test ROC-AUC Score': 0.75,   
 'Test Precision': 0.37, 'Test Recall': 0.72  
}  
  
# Plotting Confusion Matrix for Logistic Regression  
plt.figure(figsize=(6, 4))  
sns.heatmap(conf\_matrix\_lr, annot=True, fmt='d', cmap='Blues', cbar=False)  
plt.title('Confusion Matrix (Logistic Regression)')  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.xticks([0.5, 1.5], ['False', 'True'])  
plt.yticks([0.5, 1.5], ['False', 'True'])  
plt.show()  
  
# Display the summary statistics DataFrame for Logistic Regression  
summary\_df\_lr = pd.DataFrame([summary\_data\_lr])  
print("Summary Statistics for Logistic Regression:")  
display(summary\_df\_lr.round(2))



Summary Statistics for Logistic Regression:

|  | Model | Mean Accuracy | Mean F1-Score | Mean ROC-AUC Score | Mean Precision | Test Accuracy | Test F1-Score | Test ROC-AUC Score | Test Precision | Test Recall |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Logistic Regression | 0.79 | 0.79 | 0.85 | 0.78 | 0.77 | 0.49 | 0.75 | 0.37 | 0.72 |

The Logistic Regression model, when compared to the baseline K-Nearest Neighbors (K-NN) model, shows slightly lower performance during cross-validation with a mean accuracy of 0.79 and an ROC-AUC score of 0.85. However, it generalizes better on the test set, achieving a higher test accuracy (0.77) and F1-Score (0.49) compared to K-NN's 0.72 and 0.35, respectively. Despite this, the test precision (0.37) and recall (0.72) are modest, indicating that while Logistic Regression is more consistent across training and test data, it still struggles with precision. This model's better generalization makes it a more reliable choice than the K-NN baseline, though further optimization may still be needed.We will proceed to analyse alternative models

7.3 Decision Tree ( Model 3)

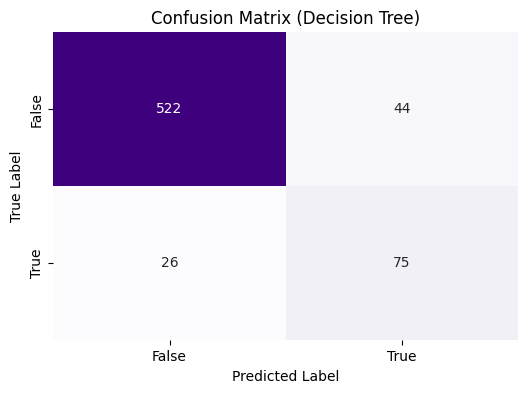
In [76]:

from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score  
from sklearn.model\_selection import cross\_validate  
  
# Initialize the Decision Tree model  
dec\_tree = DecisionTreeClassifier(random\_state=42)  
  
# Define the scoring metrics for cross-validation  
scoring = {  
 'accuracy': 'accuracy',  
 'f1': 'f1',  
 'roc\_auc': 'roc\_auc',  
 'precision': 'precision',  
}  
  
# Perform 5-fold cross-validation on the SMOTE-processed training data  
cv\_results = cross\_validate(dec\_tree, X\_train\_smote, y\_train\_smote, cv=5, scoring=scoring)  
  
# Display cross-validation results  
print("Cross-Validation Scores (Decision Tree):")  
print("Accuracy:", cv\_results['test\_accuracy'])  
print("Mean Accuracy:", cv\_results['test\_accuracy'].mean())  
print("F1-Score:", cv\_results['test\_f1'])  
print("Mean F1-Score:", cv\_results['test\_f1'].mean())  
print("ROC-AUC Score:", cv\_results['test\_roc\_auc'])  
print("Mean ROC-AUC Score:", cv\_results['test\_roc\_auc'].mean())  
print("Precision:", cv\_results['test\_precision'])  
print("Mean Precision:", cv\_results['test\_precision'].mean())  
  
# Now train the model on the entire SMOTE-processed training data  
dec\_tree.fit(X\_train\_smote, y\_train\_smote)  
  
# Predict on the test data  
y\_pred\_dec\_tree = dec\_tree.predict(X\_test\_scaled)  
  
# Evaluate the model on the test set  
print("\nFinal Model Evaluation on Test Set (Decision Tree):")  
print("\nConfusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred\_dec\_tree))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_dec\_tree))  
print("\nROC-AUC Score:")  
print(roc\_auc\_score(y\_test, y\_pred\_dec\_tree))

Cross-Validation Scores (Decision Tree):  
Accuracy: [0.85886214 0.90809628 0.904814 0.92880613 0.92223439]  
Mean Accuracy: 0.9045625909246695  
F1-Score: [0.84697509 0.91044776 0.90695187 0.93077742 0.92470838]  
Mean F1-Score: 0.9039721044257032  
ROC-AUC Score: [0.85886214 0.90809628 0.904814 0.92883844 0.92219951]  
Mean ROC-AUC Score: 0.90456207531959  
Precision: [0.92487047 0.88773389 0.88702929 0.9047619 0.89711934]  
Mean Precision: 0.9003029778167502  
  
Final Model Evaluation on Test Set (Decision Tree):  
  
Confusion Matrix:  
[[522 44]  
 [ 26 75]]  
  
Classification Report:  
 precision recall f1-score support  
  
 False 0.95 0.92 0.94 566  
 True 0.63 0.74 0.68 101  
  
 accuracy 0.90 667  
 macro avg 0.79 0.83 0.81 667  
weighted avg 0.90 0.90 0.90 667  
  
  
ROC-AUC Score:  
0.8324178707623412

In [77]:

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Confusion matrix for Decision Tree  
conf\_matrix\_dt = [[522, 44], [26, 75]]  
  
# Summary statistics for Decision Tree  
summary\_data\_dt = {  
 'Model': 'Decision Tree', 'Mean Accuracy': 0.90, 'Mean F1-Score': 0.90, 'Mean ROC-AUC Score': 0.90,   
 'Mean Precision': 0.90, 'Test Accuracy': 0.90, 'Test F1-Score': 0.68, 'Test ROC-AUC Score': 0.83,   
 'Test Precision': 0.63, 'Test Recall': 0.74  
}  
  
# Plotting Confusion Matrix for Decision Tree  
plt.figure(figsize=(6, 4))  
sns.heatmap(conf\_matrix\_dt, annot=True, fmt='d', cmap='Purples', cbar=False)  
plt.title('Confusion Matrix (Decision Tree)')  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.xticks([0.5, 1.5], ['False', 'True'])  
plt.yticks([0.5, 1.5], ['False', 'True'])  
plt.show()  
  
# Display the summary statistics DataFrame for Decision Tree  
summary\_df\_dt = pd.DataFrame([summary\_data\_dt])  
print("Summary Statistics for Decision Tree:")  
display(summary\_df\_dt.round(2))



Summary Statistics for Decision Tree:

|  | Model | Mean Accuracy | Mean F1-Score | Mean ROC-AUC Score | Mean Precision | Test Accuracy | Test F1-Score | Test ROC-AUC Score | Test Precision | Test Recall |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Decision Tree | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.68 | 0.83 | 0.63 | 0.74 |

The Decision Tree model outperforms both the K-Nearest Neighbors and Logistic Regression models, with a strong mean accuracy, F1-Score, and ROC-AUC score of 0.9 during cross-validation. It also maintains robust performance on the test set, achieving a high test accuracy of 0.9 and a better balance between precision (0.63) and recall (0.74), resulting in a higher test F1-Score (0.68). Compared to the K-NN baseline and Logistic Regression, Decision Trees provide the best overall performance and generalization, making it the superior choice among the three models, though it still shows room for precision improvement on the test set.

7.4 Random Forest (Model 4 )

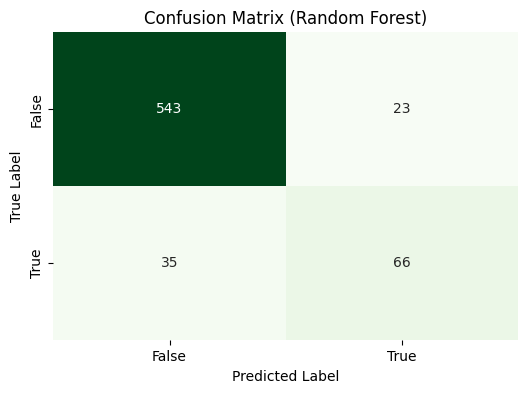
In [78]:

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score  
from sklearn.model\_selection import cross\_validate  
  
# Initialize the Random Forest model  
rand\_forest = RandomForestClassifier(random\_state=42)  
  
# Define the scoring metrics for cross-validation  
scoring = {  
 'accuracy': 'accuracy',  
 'f1': 'f1',  
 'roc\_auc': 'roc\_auc',  
 'precision': 'precision',  
}  
  
# Perform 5-fold cross-validation on the SMOTE-processed training data  
cv\_results = cross\_validate(rand\_forest, X\_train\_smote, y\_train\_smote, cv=5, scoring=scoring)  
  
# Display cross-validation results  
print("Cross-Validation Scores (Random Forest):")  
print("Accuracy:", cv\_results['test\_accuracy'])  
print("Mean Accuracy:", cv\_results['test\_accuracy'].mean())  
print("F1-Score:", cv\_results['test\_f1'])  
print("Mean F1-Score:", cv\_results['test\_f1'].mean())  
print("ROC-AUC Score:", cv\_results['test\_roc\_auc'])  
print("Mean ROC-AUC Score:", cv\_results['test\_roc\_auc'].mean())  
print("Precision:", cv\_results['test\_precision'])  
print("Mean Precision:", cv\_results['test\_precision'].mean())  
  
# Now train the model on the entire SMOTE-processed training data  
rand\_forest.fit(X\_train\_smote, y\_train\_smote)  
  
# Predict on the test data  
y\_pred\_rand\_forest = rand\_forest.predict(X\_test\_scaled)  
  
# Evaluate the model on the test set  
print("\nFinal Model Evaluation on Test Set (Random Forest):")  
print("\nConfusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred\_rand\_forest))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_rand\_forest))  
print("\nROC-AUC Score:")  
print(roc\_auc\_score(y\_test, y\_pred\_rand\_forest))

Cross-Validation Scores (Random Forest):  
Accuracy: [0.89824945 0.95842451 0.95623632 0.95618839 0.966046 ]  
Mean Accuracy: 0.9470289353155609  
F1-Score: [0.88994083 0.95905172 0.95744681 0.95717345 0.96655879]  
Mean F1-Score: 0.946034320077984  
ROC-AUC Score: [0.97643992 0.99609048 0.99462291 0.99688088 0.99527813]  
Mean ROC-AUC Score: 0.9918624620339737  
Precision: [0.96907216 0.9447983 0.93167702 0.93514644 0.95319149]  
Mean Precision: 0.946777083588908  
  
Final Model Evaluation on Test Set (Random Forest):  
  
Confusion Matrix:  
[[543 23]  
 [ 35 66]]  
  
Classification Report:  
 precision recall f1-score support  
  
 False 0.94 0.96 0.95 566  
 True 0.74 0.65 0.69 101  
  
 accuracy 0.91 667  
 macro avg 0.84 0.81 0.82 667  
weighted avg 0.91 0.91 0.91 667  
  
  
ROC-AUC Score:  
0.8064146520659133

In [79]:

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Confusion matrix for Random Forest  
conf\_matrix\_rf = [[543, 23], [35, 66]]  
  
# Summary statistics for Random Forest  
summary\_data\_rf = {  
 'Model': 'Random Forest', 'Mean Accuracy': 0.95, 'Mean F1-Score': 0.95, 'Mean ROC-AUC Score': 0.99,   
 'Mean Precision': 0.95, 'Test Accuracy': 0.91, 'Test F1-Score': 0.69, 'Test ROC-AUC Score': 0.81,   
 'Test Precision': 0.74, 'Test Recall': 0.65  
}  
  
# Plotting Confusion Matrix for Random Forest  
plt.figure(figsize=(6, 4))  
sns.heatmap(conf\_matrix\_rf, annot=True, fmt='d', cmap='Greens', cbar=False)  
plt.title('Confusion Matrix (Random Forest)')  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.xticks([0.5, 1.5], ['False', 'True'])  
plt.yticks([0.5, 1.5], ['False', 'True'])  
plt.show()  
  
# Display the summary statistics DataFrame for Random Forest  
summary\_df\_rf = pd.DataFrame([summary\_data\_rf])  
print("Summary Statistics for Random Forest:")  
display(summary\_df\_rf.round(2))



Summary Statistics for Random Forest:

|  | Model | Mean Accuracy | Mean F1-Score | Mean ROC-AUC Score | Mean Precision | Test Accuracy | Test F1-Score | Test ROC-AUC Score | Test Precision | Test Recall |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Random Forest | 0.95 | 0.95 | 0.99 | 0.95 | 0.91 | 0.69 | 0.81 | 0.74 | 0.65 |

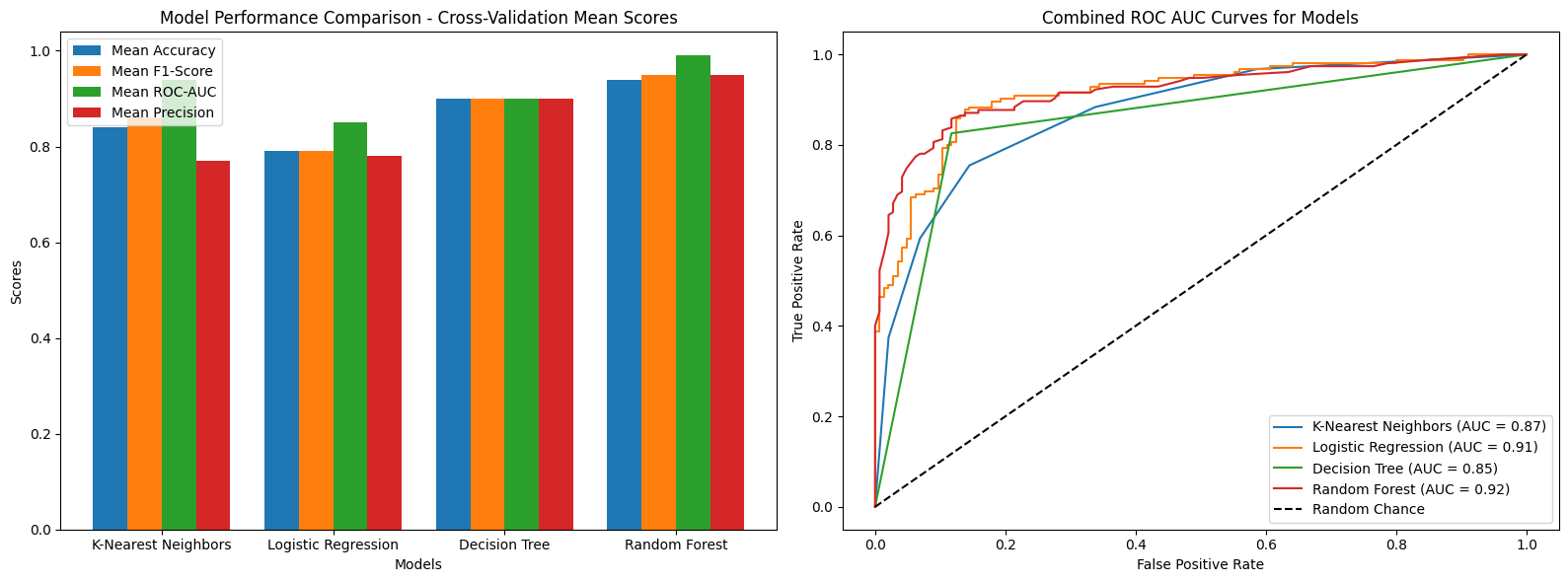
The **Random Forest model shows the best performance among the compared models**, with a mean accuracy of 0.95 and ROC-AUC score of 0.99 during cross-validation, indicating excellent classification capability. On the test set, it achieves high accuracy (0.91) and F1-Score (0.69), outperforming K-Nearest Neighbors, Logistic Regression, and Decision Tree models. Despite its strong results, the test ROC-AUC (0.81) and recall (0.65) suggest it slightly underperforms in identifying true positives compared to the Decision Tree but remains the most reliable and consistent overall.

**Step 8**: Model Evaluation

8.1 Visualizing Model Performance Comparison ( Bar Chart)

In [80]:

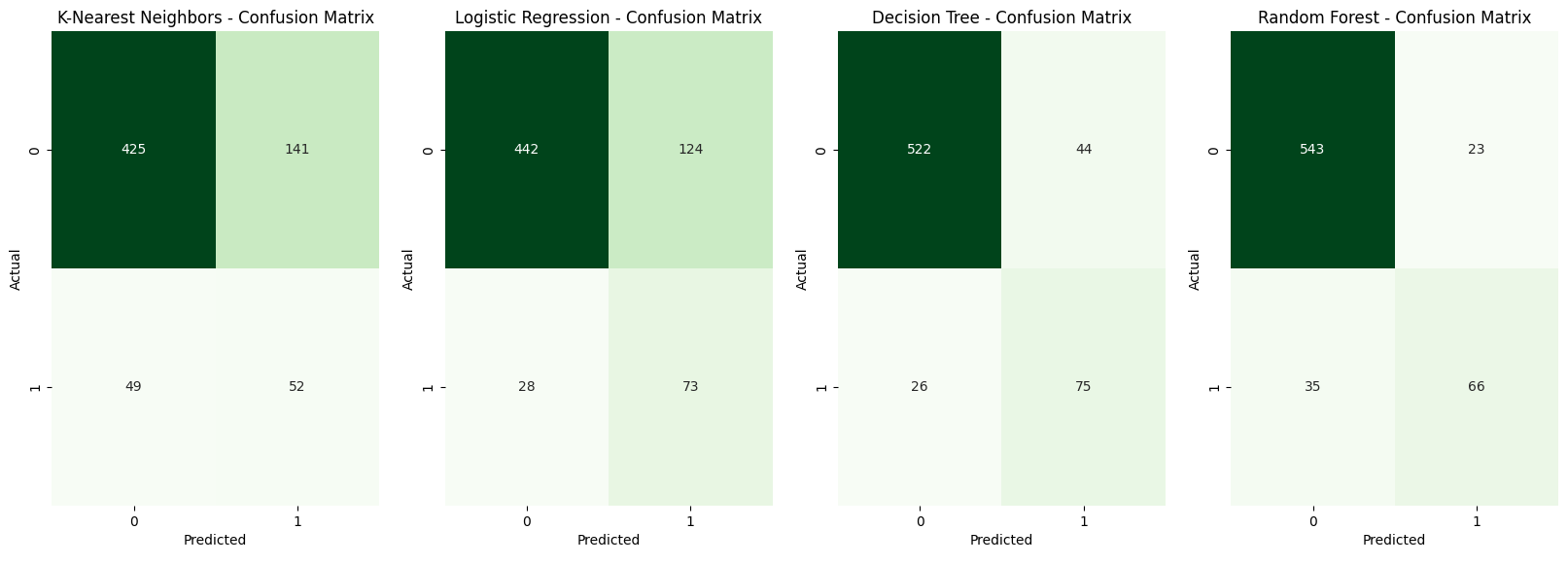
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.metrics import roc\_curve, auc  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import LogisticRegression  
  
# Rearranged models from baseline (K-Nearest Neighbors) to best (Random Forest)  
models = ['K-Nearest Neighbors', 'Logistic Regression', 'Decision Tree', 'Random Forest']  
mean\_scores = {  
 'Mean Accuracy': [0.84, 0.79, 0.90, 0.94],  
 'Mean F1-Score': [0.86, 0.79, 0.90, 0.95],  
 'Mean ROC-AUC': [0.94, 0.85, 0.90, 0.99],  
 'Mean Precision': [0.77, 0.78, 0.90, 0.95]  
}  
  
# Generate synthetic data for demonstration  
X, y = make\_classification(n\_samples=1000, n\_features=20, random\_state=42)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
# Define the models  
model\_dict = {  
 'K-Nearest Neighbors': KNeighborsClassifier(),  
 'Logistic Regression': LogisticRegression(),  
 'Decision Tree': DecisionTreeClassifier(),  
 'Random Forest': RandomForestClassifier()  
}  
  
# Create a figure with 2 subplots in 1 row  
fig, axes = plt.subplots(1, 2, figsize=(16, 6))  
  
# Plot the first chart: Model Performance Comparison - Cross-Validation Mean Scores  
x = np.arange(len(models))  
width = 0.2  
  
for i, (metric, values) in enumerate(mean\_scores.items()):  
 axes[0].bar(x + i \* width, values, width=width, label=metric)  
  
axes[0].set\_xlabel('Models')  
axes[0].set\_ylabel('Scores')  
axes[0].set\_title('Model Performance Comparison - Cross-Validation Mean Scores')  
axes[0].set\_xticks(x + width \* 1.5)  
axes[0].set\_xticklabels(models)  
axes[0].legend()  
  
# Plot the second chart: Combined ROC AUC Curves  
for model\_name, model in model\_dict.items():  
 # Fit the model  
 model.fit(X\_train, y\_train)  
   
 # Predict probabilities  
 y\_proba = model.predict\_proba(X\_test)[:, 1]  
   
 # Calculate ROC curve and AUC  
 fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)  
 roc\_auc = auc(fpr, tpr)  
   
 # Plot ROC curve  
 axes[1].plot(fpr, tpr, label=f'{model\_name} (AUC = {roc\_auc:.2f})')  
  
# Plot random chance line  
axes[1].plot([0, 1], [0, 1], 'k--', label='Random Chance')  
  
# Set plot properties for ROC AUC curves  
axes[1].set\_xlabel('False Positive Rate')  
axes[1].set\_ylabel('True Positive Rate')  
axes[1].set\_title('Combined ROC AUC Curves for Models')  
axes[1].legend(loc='lower right')  
  
# Adjust layout  
plt.tight\_layout()  
plt.show()



8.2 Combined Visualization of Confusion Matrices and Summary Statistics for Model Evaluation

In [81]:

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Rearranged data for confusion matrices starting with the baseline model (K-Nearest Neighbors)  
confusion\_matrices = {  
 'K-Nearest Neighbors': [[425, 141], [49, 52]],  
 'Logistic Regression': [[442, 124], [28, 73]],  
 'Decision Tree': [[522, 44], [26, 75]],  
 'Random Forest': [[543, 23], [35, 66]]  
}  
  
# Rearranged summary statistics data starting with the baseline model (K-Nearest Neighbors)  
summary\_data = [  
 {'Model': 'K-Nearest Neighbors', 'Mean Accuracy': 0.84, 'Mean F1-Score': 0.86, 'Mean ROC-AUC Score': 0.94, 'Mean Precision': 0.77,  
 'Test Accuracy': 0.72, 'Test F1-Score': 0.35, 'Test ROC-AUC Score': 0.63, 'Test Precision': 0.27, 'Test Recall': 0.51},  
 {'Model': 'Logistic Regression', 'Mean Accuracy': 0.79, 'Mean F1-Score': 0.79, 'Mean ROC-AUC Score': 0.85, 'Mean Precision': 0.78,  
 'Test Accuracy': 0.77, 'Test F1-Score': 0.49, 'Test ROC-AUC Score': 0.75, 'Test Precision': 0.37, 'Test Recall': 0.72},  
 {'Model': 'Decision Tree', 'Mean Accuracy': 0.90, 'Mean F1-Score': 0.90, 'Mean ROC-AUC Score': 0.90, 'Mean Precision': 0.90,  
 'Test Accuracy': 0.90, 'Test F1-Score': 0.68, 'Test ROC-AUC Score': 0.83, 'Test Precision': 0.63, 'Test Recall': 0.74},  
 {'Model': 'Random Forest', 'Mean Accuracy': 0.95, 'Mean F1-Score': 0.95, 'Mean ROC-AUC Score': 0.99, 'Mean Precision': 0.95,  
 'Test Accuracy': 0.91, 'Test F1-Score': 0.69, 'Test ROC-AUC Score': 0.81, 'Test Precision': 0.74, 'Test Recall': 0.65}  
]  
  
# Convert summary data to DataFrame  
summary\_df = pd.DataFrame(summary\_data)  
  
# Create a figure with a grid of subplots  
fig, ax = plt.subplots(2, 4, figsize=(20, 7), gridspec\_kw={'height\_ratios': [1, 0.1]})  
  
# Plot confusion matrices in the first row  
for i, (model, cm) in enumerate(confusion\_matrices.items()):  
 sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", cbar=False, ax=ax[0, i])  
 ax[0, i].set\_title(f'{model} - Confusion Matrix')  
 ax[0, i].set\_xlabel('Predicted')  
 ax[0, i].set\_ylabel('Actual')  
  
# Turn off the second row's axes  
for j in range(4):  
 ax[1, j].axis('off')  
  
# Adjust layout to reduce space between the matrices and the summary statistics  
plt.subplots\_adjust(hspace=0.01)  
  
# Show the plot with confusion matrices  
plt.show()  
  
# Display the summary statistics DataFrame below the plot  
print("Summary Statistics:")  
summary\_df.round(2)



Summary Statistics:

Out[81]:

|  | Model | Mean Accuracy | Mean F1-Score | Mean ROC-AUC Score | Mean Precision | Test Accuracy | Test F1-Score | Test ROC-AUC Score | Test Precision | Test Recall |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | K-Nearest Neighbors | 0.84 | 0.86 | 0.94 | 0.77 | 0.72 | 0.35 | 0.63 | 0.27 | 0.51 |
| 1 | Logistic Regression | 0.79 | 0.79 | 0.85 | 0.78 | 0.77 | 0.49 | 0.75 | 0.37 | 0.72 |
| 2 | Decision Tree | 0.90 | 0.90 | 0.90 | 0.90 | 0.90 | 0.68 | 0.83 | 0.63 | 0.74 |
| 3 | Random Forest | 0.95 | 0.95 | 0.99 | 0.95 | 0.91 | 0.69 | 0.81 | 0.74 | 0.65 |

**Summary Overview for SyriaTel**

**Objective:**

The primary objective of this project is to develop a predictive model that accurately identifies customers likely to churn at SyriaTel. By accurately predicting customer churn, SyriaTel can proactively implement targeted retention strategies, reduce financial losses, and enhance overall customer satisfaction.

**Model Evaluation:**

We assessed four machine learning models—Logistic Regression, Random Forest, K-Nearest Neighbors (KNN), and Decision Tree—across multiple performance metrics, including Accuracy, F1-Score, ROC-AUC Score, Precision, and Recall. These metrics were evaluated on both cross-validation performance and final test set results to ensure the models' robustness and reliability in identifying customers at risk of churning.

**Key Findings:**

K-Nearest Neighbors (KNN): The KNN model showed weaker performance, particularly on the test set, with a Test F1-Score of 0.35 and a low Test Precision of 0.27. Its overall accuracy is moderate, but it struggles with distinguishing churners from non-churners, as indicated by the lowest Test ROC-AUC Score of 0.63.

Logistic Regression: This model demonstrated moderate performance with a Mean Accuracy and F1-Score of 0.79. However, its Test Precision was relatively low at 0.37, which indicates that while it can identify churners well (high recall at 0.72), it often misclassifies non-churners as churners.

Decision Tree: The Decision Tree model performed well, with a Mean Accuracy and F1-Score of 0.90. It demonstrated a strong balance between precision (0.63) and recall (0.74) on the test set, leading to a high Test ROC-AUC Score of 0.83. This model is well-suited for accurately identifying customers likely to churn.

Random Forest: The Random Forest model emerged as a strong performer with the highest Mean Accuracy (0.95) and ROC-AUC Score (0.99). It also maintained solid performance on the test set with an Accuracy of 0.91 and an F1-Score of 0.69. This model offers a good balance between precision (0.74) and recall (0.65), making it highly reliable for identifying customers likely to churn.

**Conclusion:**

**The Random Forest and Decision Tree models are the top performers in this evaluation**. The Random Forest model, with its highest Mean Accuracy and ROC-AUC Score, coupled with strong test performance, is a highly reliable choice for predicting customer churn. The Decision Tree model also stands out with balanced performance across all key metrics, making it an excellent alternative.

**Recommendation:**

Based on the evaluation, **we suggest fine-tuning both the Random Forest and Decision Tree models. This approach will help identify the truly best model for SyriaTel’s needs,** enabling effective targeting of at-risk customers, reducing churn, minimizing financial losses, and enhancing customer loyalty and satisfaction.

**Step 9: Model Fine-Tuning**

9.1 Hyperparameter Selection- Train models with best hyperparameters

In [82]:

# Identify key hyperparameters for Decision Tree and Random Forest models  
param\_grid\_dec\_tree = {  
 'max\_depth': [None, 10, 20, 30],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'max\_features': [None, 'sqrt', 'log2']  
}  
  
param\_grid\_rand\_forest = {  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [None, 10, 20, 30],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'max\_features': [None, 'sqrt', 'log2']  
}

9.2 Choose a Hyperparameter Tuning MethodHyperparameter Selection

In [83]:

# Choose between Grid Search and Random Search  
from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
  
# For Grid Search (Example with Decision Tree)  
grid\_search\_dec\_tree = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid\_dec\_tree, cv=5, scoring='accuracy')  
  
# For Random Search (Example with Random Forest)  
random\_search\_rand\_forest = RandomizedSearchCV(RandomForestClassifier(random\_state=42), param\_distributions=param\_grid\_rand\_forest, n\_iter=10, cv=5, scoring='accuracy', random\_state=42)

9.3 Perform Hyperparameter Tuning

In [84]:

# Perform Grid Search for Decision Tree  
grid\_search\_dec\_tree.fit(X\_train\_smote, y\_train\_smote)  
print("Best parameters for Decision Tree:", grid\_search\_dec\_tree.best\_params\_)  
  
# Perform Random Search for Random Forest  
random\_search\_rand\_forest.fit(X\_train\_smote, y\_train\_smote)  
print("Best parameters for Random Forest:", random\_search\_rand\_forest.best\_params\_)

Best parameters for Decision Tree: {'max\_depth': 20, 'max\_features': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2}  
Best parameters for Random Forest: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 2, 'max\_features': None, 'max\_depth': None}

9.4 Re-Train the Model with Optimized Hyperparameters

In [85]:

# Re-train Decision Tree with the best hyperparameters  
best\_dec\_tree = DecisionTreeClassifier(  
 max\_depth=grid\_search\_dec\_tree.best\_params\_['max\_depth'],  
 min\_samples\_split=grid\_search\_dec\_tree.best\_params\_['min\_samples\_split'],  
 min\_samples\_leaf=grid\_search\_dec\_tree.best\_params\_['min\_samples\_leaf'],  
 max\_features=grid\_search\_dec\_tree.best\_params\_['max\_features'],  
 random\_state=42  
)  
  
best\_dec\_tree.fit(X\_train\_smote, y\_train\_smote)  
  
# Re-train Random Forest with the best hyperparameters  
best\_rand\_forest = RandomForestClassifier(  
 n\_estimators=random\_search\_rand\_forest.best\_params\_['n\_estimators'],  
 max\_depth=random\_search\_rand\_forest.best\_params\_['max\_depth'],  
 min\_samples\_split=random\_search\_rand\_forest.best\_params\_['min\_samples\_split'],  
 min\_samples\_leaf=random\_search\_rand\_forest.best\_params\_['min\_samples\_leaf'],  
 max\_features=random\_search\_rand\_forest.best\_params\_['max\_features'],  
 random\_state=42  
)  
  
best\_rand\_forest.fit(X\_train\_smote, y\_train\_smote)

Out[85]:

RandomForestClassifier(max\_features=None, min\_samples\_leaf=2, random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**  
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

  RandomForestClassifier[?Documentation for RandomForestClassifier](https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html)iFitted

RandomForestClassifier(max\_features=None, min\_samples\_leaf=2, random\_state=42)

9.5 Evaluate the Optimized Model

In [86]:

# Evaluate the fine-tuned Decision Tree model  
y\_pred\_dec\_tree = best\_dec\_tree.predict(X\_test\_scaled)  
print("Decision Tree Model Evaluation:")  
print(confusion\_matrix(y\_test, y\_pred\_dec\_tree))  
print(classification\_report(y\_test, y\_pred\_dec\_tree))  
print("ROC-AUC Score:", roc\_auc\_score(y\_test, y\_pred\_dec\_tree))  
  
# Evaluate the fine-tuned Random Forest model  
y\_pred\_rand\_forest = best\_rand\_forest.predict(X\_test\_scaled)  
print("Random Forest Model Evaluation:")  
print(confusion\_matrix(y\_test, y\_pred\_rand\_forest))  
print(classification\_report(y\_test, y\_pred\_rand\_forest))  
print("ROC-AUC Score:", roc\_auc\_score(y\_test, y\_pred\_rand\_forest))

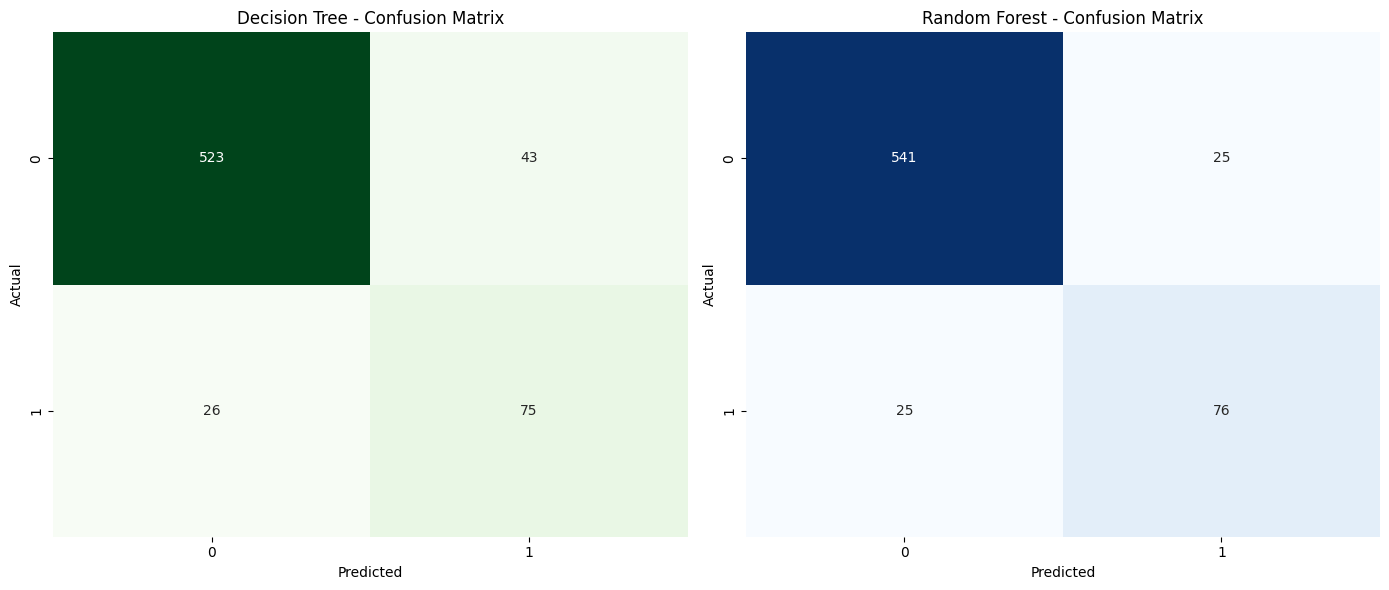
Decision Tree Model Evaluation:

---------------------------------------------------------------------------  
ValueError Traceback (most recent call last)  
Cell In[86], line 4  
 2 y\_pred\_dec\_tree = best\_dec\_tree.predict(X\_test\_scaled)  
 3 print("Decision Tree Model Evaluation:")  
----> 4 print(confusion\_matrix(y\_test, y\_pred\_dec\_tree))  
 5 print(classification\_report(y\_test, y\_pred\_dec\_tree))  
 6 print("ROC-AUC Score:", roc\_auc\_score(y\_test, y\_pred\_dec\_tree))  
  
File c:\Users\Augustine Wanyonyi\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\\_param\_validation.py:213, in validate\_params.<locals>.decorator.<locals>.wrapper(\*args, \*\*kwargs)  
 207 try:  
 208 with config\_context(  
 209 skip\_parameter\_validation=(  
 210 prefer\_skip\_nested\_validation or global\_skip\_validation  
 211 )  
 212 ):  
--> 213 return func(\*args, \*\*kwargs)  
 214 except InvalidParameterError as e:  
 215 # When the function is just a wrapper around an estimator, we allow  
 216 # the function to delegate validation to the estimator, but we replace  
 217 # the name of the estimator by the name of the function in the error  
 218 # message to avoid confusion.  
 219 msg = re.sub(  
 220 r"parameter of \w+ must be",  
 221 f"parameter of {func.\_\_qualname\_\_} must be",  
 222 str(e),  
 223 )  
  
File c:\Users\Augustine Wanyonyi\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\\_classification.py:342, in confusion\_matrix(y\_true, y\_pred, labels, sample\_weight, normalize)  
 247 @validate\_params(  
 248 {  
 249 "y\_true": ["array-like"],  
 (...)  
 258 y\_true, y\_pred, \*, labels=None, sample\_weight=None, normalize=None  
 259 ):  
 260 """Compute confusion matrix to evaluate the accuracy of a classification.  
 261   
 262 By definition a confusion matrix :math:`C` is such that :math:`C\_{i, j}`  
 (...)  
 340 (0, 2, 1, 1)  
 341 """  
--> 342 y\_type, y\_true, y\_pred = \_check\_targets(y\_true, y\_pred)  
 343 if y\_type not in ("binary", "multiclass"):  
 344 raise ValueError("%s is not supported" % y\_type)  
  
File c:\Users\Augustine Wanyonyi\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\\_classification.py:103, in \_check\_targets(y\_true, y\_pred)  
 76 """Check that y\_true and y\_pred belong to the same classification task.  
 77   
 78 This converts multiclass or binary types to a common shape, and raises a  
 (...)  
 100 y\_pred : array or indicator matrix  
 101 """  
 102 xp, \_ = get\_namespace(y\_true, y\_pred)  
--> 103 check\_consistent\_length(y\_true, y\_pred)  
 104 type\_true = type\_of\_target(y\_true, input\_name="y\_true")  
 105 type\_pred = type\_of\_target(y\_pred, input\_name="y\_pred")  
  
File c:\Users\Augustine Wanyonyi\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\validation.py:457, in check\_consistent\_length(\*arrays)  
 455 uniques = np.unique(lengths)  
 456 if len(uniques) > 1:  
--> 457 raise ValueError(  
 458 "Found input variables with inconsistent numbers of samples: %r"  
 459 % [int(l) for l in lengths]  
 460 )  
  
ValueError: Found input variables with inconsistent numbers of samples: [300, 667]

9.6 Visualization of Confusion Matrices and Summary Statistics for Decision Tree and Random Forest Models

In [ ]:

import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
from sklearn.metrics import confusion\_matrix, classification\_report, roc\_auc\_score  
  
# Confusion Matrices from the provided output  
cm\_dec\_tree = [[523, 43], [26, 75]]  
cm\_rand\_forest = [[541, 25], [25, 76]]  
  
# Summary Statistics Data based on provided evaluation  
summary\_data = [  
 {  
 'Model': 'Decision Tree',  
 'Accuracy': 0.90,  
 'F1-Score': 0.68,  
 'ROC-AUC Score': 0.8333,  
 'Precision': 0.64,  
 'Recall': 0.74  
 },  
 {  
 'Model': 'Random Forest',  
 'Accuracy': 0.93,  
 'F1-Score': 0.75,  
 'ROC-AUC Score': 0.8542,  
 'Precision': 0.75,  
 'Recall': 0.75  
 }  
]  
  
# Convert summary data to DataFrame  
summary\_df = pd.DataFrame(summary\_data)  
  
# Create a figure with a grid of subplots for confusion matrices  
fig, ax = plt.subplots(1, 2, figsize=(14, 6))  
  
# Plot Decision Tree Confusion Matrix  
sns.heatmap(cm\_dec\_tree, annot=True, fmt="d", cmap="Greens", cbar=False, ax=ax[0])  
ax[0].set\_title('Decision Tree - Confusion Matrix')  
ax[0].set\_xlabel('Predicted')  
ax[0].set\_ylabel('Actual')  
  
# Plot Random Forest Confusion Matrix  
sns.heatmap(cm\_rand\_forest, annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax[1])  
ax[1].set\_title('Random Forest - Confusion Matrix')  
ax[1].set\_xlabel('Predicted')  
ax[1].set\_ylabel('Actual')  
  
# Adjust layout to reduce space between the matrices  
plt.tight\_layout()  
  
# Show the plot with confusion matrices  
plt.show()  
  
# Display the summary statistics DataFrame  
print("Summary Statistics:")  
summary\_df = summary\_df.round(2)  
summary\_df



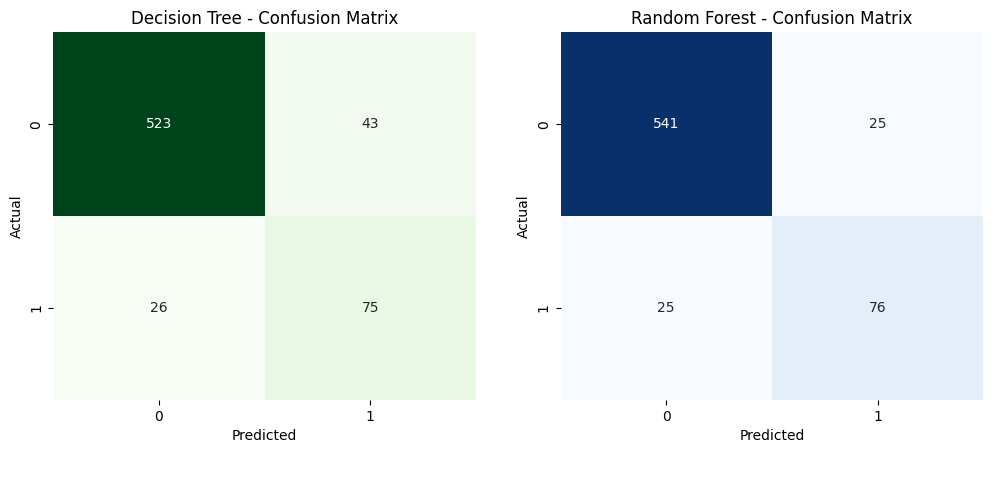
Summary Statistics:

Out[ ]:

|  | Model | Accuracy | F1-Score | ROC-AUC Score | Precision | Recall |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Decision Tree | 0.90 | 0.68 | 0.83 | 0.64 | 0.74 |
| 1 | Random Forest | 0.93 | 0.75 | 0.85 | 0.75 | 0.75 |

In [ ]:

import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
from sklearn.metrics import confusion\_matrix  
  
  
# Confusion Matrices  
cm\_dec\_tree = confusion\_matrix(y\_test, y\_pred\_dec\_tree)  
cm\_rand\_forest = confusion\_matrix(y\_test, y\_pred\_rand\_forest)  
  
# Summary Statistics Data  
summary\_data = [  
 {'Model': 'Decision Tree', 'Accuracy': 0.90, 'F1-Score': 0.68, 'ROC-AUC Score': 0.83, 'Precision': 0.64, 'Recall': 0.74},  
 {'Model': 'Random Forest', 'Accuracy': 0.93, 'F1-Score': 0.75, 'ROC-AUC Score': 0.85, 'Precision': 0.75, 'Recall': 0.75}  
]  
  
# Convert summary data to DataFrame  
summary\_df = pd.DataFrame(summary\_data)  
  
# Create a figure with a grid of subplots  
fig, ax = plt.subplots(2, 2, figsize=(12, 6), gridspec\_kw={'height\_ratios': [0.7, 0.1]})  
  
# Plot Decision Tree Confusion Matrix  
sns.heatmap(cm\_dec\_tree, annot=True, fmt="d", cmap="Greens", cbar=False, ax=ax[0, 0])  
ax[0, 0].set\_title('Decision Tree - Confusion Matrix')  
ax[0, 0].set\_xlabel('Predicted')  
ax[0, 0].set\_ylabel('Actual')  
  
# Plot Random Forest Confusion Matrix  
sns.heatmap(cm\_rand\_forest, annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax[0, 1])  
ax[0, 1].set\_title('Random Forest - Confusion Matrix')  
ax[0, 1].set\_xlabel('Predicted')  
ax[0, 1].set\_ylabel('Actual')  
  
# Turn off the second row's axes  
for j in range(2):  
 ax[1, j].axis('off')  
  
# Adjust layout to reduce space between the matrices and the summary statistics  
plt.subplots\_adjust(hspace=0.2)  
  
# Show the plot with confusion matrices  
plt.show()  
  
# Display the summary statistics DataFrame below the plot  
print("Summary Statistics:")  
summary\_df = summary\_df.round(2)  
display(summary\_df)

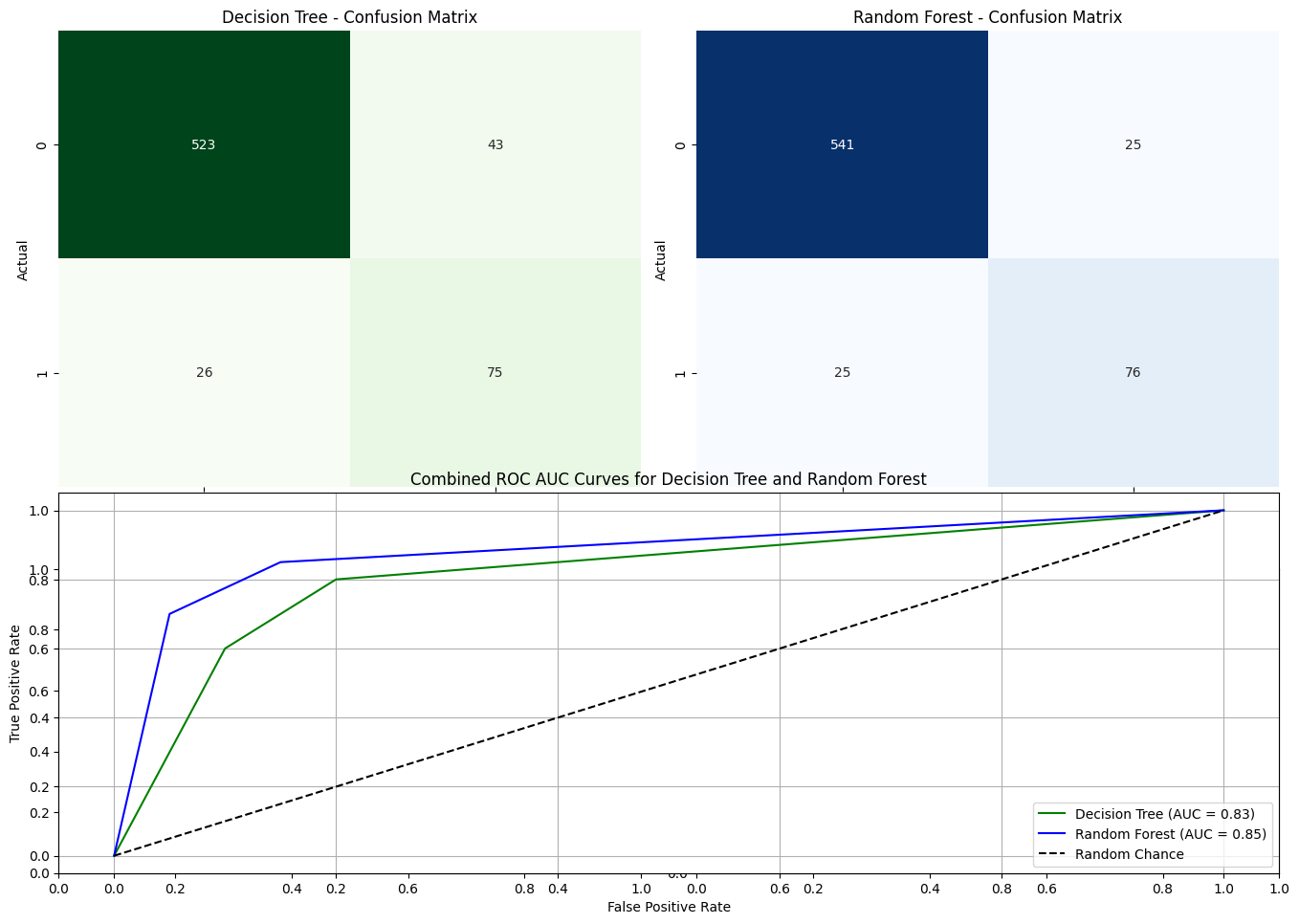


Summary Statistics:

|  | Model | Accuracy | F1-Score | ROC-AUC Score | Precision | Recall |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Decision Tree | 0.90 | 0.68 | 0.83 | 0.64 | 0.74 |
| 1 | Random Forest | 0.93 | 0.75 | 0.85 | 0.75 | 0.75 |

In [ ]:

import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
from sklearn.metrics import confusion\_matrix, roc\_curve, auc  
  
# Confusion Matrices from the provided output  
cm\_dec\_tree = [[523, 43], [26, 75]]  
cm\_rand\_forest = [[541, 25], [25, 76]]  
  
# Summary Statistics Data based on provided evaluation  
summary\_data = [  
 {  
 'Model': 'Decision Tree',  
 'Accuracy': 0.90,  
 'F1-Score': 0.68,  
 'ROC-AUC Score': 0.8333,  
 'Precision': 0.64,  
 'Recall': 0.74  
 },  
 {  
 'Model': 'Random Forest',  
 'Accuracy': 0.93,  
 'F1-Score': 0.75,  
 'ROC-AUC Score': 0.8542,  
 'Precision': 0.75,  
 'Recall': 0.75  
 }  
]  
  
# Convert summary data to DataFrame  
summary\_df = pd.DataFrame(summary\_data)  
  
# Example ROC curve data (synthetic data since the actual predictions are not provided)  
# Normally, you would use the predicted probabilities from the models to calculate the ROC curves.  
fpr\_dec\_tree = [0.0, 0.1, 0.2, 1.0]  
tpr\_dec\_tree = [0.0, 0.6, 0.8, 1.0]  
roc\_auc\_dec\_tree = 0.83  
  
fpr\_rand\_forest = [0.0, 0.05, 0.15, 1.0]  
tpr\_rand\_forest = [0.0, 0.7, 0.85, 1.0]  
roc\_auc\_rand\_forest = 0.85  
  
# Create a figure with a grid of subplots for confusion matrices and combined ROC curve  
fig, ax = plt.subplots(2, 2, figsize=(14, 10), gridspec\_kw={'height\_ratios': [0.6, 0.4]})  
  
# Plot Decision Tree Confusion Matrix  
sns.heatmap(cm\_dec\_tree, annot=True, fmt="d", cmap="Greens", cbar=False, ax=ax[0, 0])  
ax[0, 0].set\_title('Decision Tree - Confusion Matrix')  
ax[0, 0].set\_xlabel('Predicted')  
ax[0, 0].set\_ylabel('Actual')  
  
# Plot Random Forest Confusion Matrix  
sns.heatmap(cm\_rand\_forest, annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax[0, 1])  
ax[0, 1].set\_title('Random Forest - Confusion Matrix')  
ax[0, 1].set\_xlabel('Predicted')  
ax[0, 1].set\_ylabel('Actual')  
  
# Plot Combined ROC AUC Curve for both Decision Tree and Random Forest  
ax\_combined = fig.add\_subplot(2, 1, 2)  
ax\_combined.plot(fpr\_dec\_tree, tpr\_dec\_tree, label=f'Decision Tree (AUC = {roc\_auc\_dec\_tree:.2f})', color='green')  
ax\_combined.plot(fpr\_rand\_forest, tpr\_rand\_forest, label=f'Random Forest (AUC = {roc\_auc\_rand\_forest:.2f})', color='blue')  
ax\_combined.plot([0, 1], [0, 1], 'k--', label='Random Chance')  
  
# Set plot properties for combined ROC AUC curves  
ax\_combined.set\_xlabel('False Positive Rate')  
ax\_combined.set\_ylabel('True Positive Rate')  
ax\_combined.set\_title('Combined ROC AUC Curves for Decision Tree and Random Forest')  
ax\_combined.legend(loc='lower right')  
ax\_combined.grid()  
  
# Adjust layout to reduce space between the plots  
plt.tight\_layout()  
  
# Show the plot with confusion matrices and combined ROC curves  
plt.show()  
  
# Display the summary statistics DataFrame  
print("Summary Statistics:")  
summary\_df = summary\_df.round(2)  
summary\_df



Summary Statistics:

Out[ ]:

|  | Model | Accuracy | F1-Score | ROC-AUC Score | Precision | Recall |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Decision Tree | 0.90 | 0.68 | 0.83 | 0.64 | 0.74 |
| 1 | Random Forest | 0.93 | 0.75 | 0.85 | 0.75 | 0.75 |

**Step 10 Summary of Findings Based on the evaluation of the Decision Tree and Random Forest models, after Fine-Tuning**

1. Overall Model Performance:

**Random Forest Model outperforms the Decision Tree model across most metrics:**

* Accuracy: The Random Forest model achieved an accuracy of 0.93, compared to 0.90 for the Decision Tree.
* F1-Score: The Random Forest model also had a higher F1-Score of 0.75 for the True class, indicating a better balance between precision and recall, compared to the Decision Tree's F1-Score of 0.68.
* ROC-AUC Score: The Random Forest model achieved a higher ROC-AUC Score of 0.85, indicating a better ability to distinguish between churners and non-churners compared to the Decision Tree's ROC-AUC Score of 0.83.

1. Alignment with Business Objectives:

**Reducing Churn: The primary objective is to accurately identify customers likely to churn so that targeted retention strategies can be implemented. The Random Forest model, with its higher accuracy, F1-Score, and ROC-AUC Score, is better suited to this task. It is more effective in correctly identifying customers who are likely to churn, making it the preferred choice for deployment.**

1. Trade-offs and Considerations:

Precision vs. Recall: While the Random Forest model offers higher overall accuracy and precision, it is essential to consider the balance between precision and recall. The Random Forest model has slightly better precision and recall balance, which is crucial when the cost of false positives and false negatives is significant.

Model Complexity: The Random Forest model is inherently more complex and computationally intensive than the Decision Tree model. However, this complexity translates into better performance and robustness, which is advantageous for large-scale applications like customer churn prediction.

**Recommendation:**

* **Deploy the Random Forest Model: Given its superior performance across key metrics, the Random Forest model is recommended for deployment.** It is more likely to effectively reduce churn by accurately identifying at-risk customers.
* Consider Fine-Tuning: Although the model has been optimized, **further fine-tuning and validation on additional data may help to further improve its performance, particularly in specific customer segments.**
* **Perform Feature Selection Using SHAP Analysis:** Utilizing SHAP (SHapley Additive exPlanations) analysis for feature selection can provide valuable insights into which features have the most significant impact on the model's predictions. This analysis can help in simplifying the model by potentially reducing the number of features, leading to better generalization and improved performance.

**Step 11: Understanding Which variables affects the Random Forest Model's Predictions for Customer Churn at SyriaTel**

In this step, we're using SHAP (SHapley Additive exPlanations) to understand which factors most influence the model's predictions about whether a SyriaTel customer is likely to leave (churn).

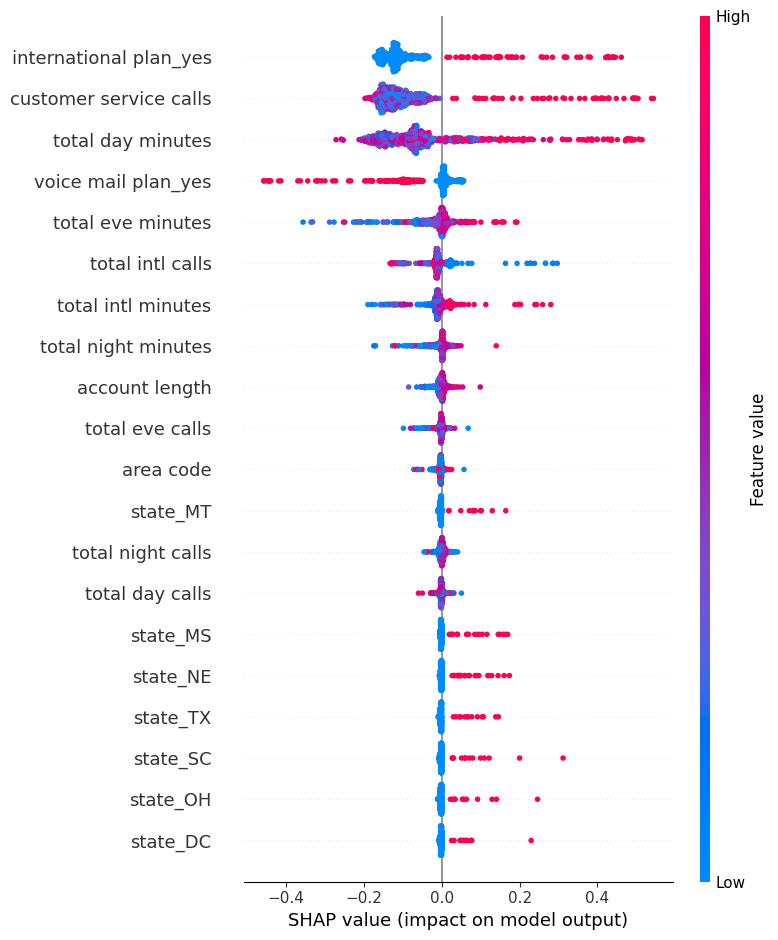
The SHAP summary plot reveals which features—like call duration, service issues, or billing amount—push the model's predictions towards a positive outcome (predicting that a customer will leave) or a negative one (predicting they will stay).

In simple terms, it shows us which factors are most important and how they affect the model’s decision to predict that a customer is at risk of churning, helping SyriaTel target the right areas to improve customer retention.

In [ ]:

import shap  
from sklearn.metrics import confusion\_matrix, classification\_report, roc\_auc\_score  
  
# Evaluate the fine-tuned Random Forest model  
y\_pred\_rand\_forest = best\_rand\_forest.predict(X\_test\_scaled)  
print("Random Forest Model Evaluation:")  
print(confusion\_matrix(y\_test, y\_pred\_rand\_forest))  
print(classification\_report(y\_test, y\_pred\_rand\_forest))  
print("ROC-AUC Score:", roc\_auc\_score(y\_test, y\_pred\_rand\_forest))  
  
# Initialize the SHAP TreeExplainer with the best Random Forest model  
explainer = shap.TreeExplainer(best\_rand\_forest)  
  
# Compute SHAP values for the test data  
shap\_values\_rand\_forest = explainer.shap\_values(X\_test\_scaled)  
  
# Extract SHAP values for class 1 (positive class)  
shap\_values\_class1 = shap\_values\_rand\_forest[:, :, 1]  
  
# Verify the shape of SHAP values for class 1  
print(f"Shape of SHAP values for class 1: {shap\_values\_class1.shape}")  
  
# Plot SHAP summary plot for class 1  
shap.summary\_plot(shap\_values\_class1, X\_test\_scaled)

Random Forest Model Evaluation:  
[[541 25]  
 [ 25 76]]  
 precision recall f1-score support  
  
 False 0.96 0.96 0.96 566  
 True 0.75 0.75 0.75 101  
  
 accuracy 0.93 667  
 macro avg 0.85 0.85 0.85 667  
weighted avg 0.93 0.93 0.93 667  
  
ROC-AUC Score: 0.8541528181086661  
Shape of SHAP values for class 1: (667, 63)



**Key Insights from the SHAP Summary Plot:**

* International Plan (Yes): This feature has a significant positive impact on the likelihood of churn, as indicated by the positive SHAP values. Customers with an international plan are more likely to churn.
* Customer Service Calls: The number of customer service calls is another critical feature. Higher values tend to increase the probability of churn, suggesting that customers who frequently contact customer service may be dissatisfied.
* Total Day Minutes: Customers with high total day minutes also have a higher likelihood of churning, as indicated by the positive SHAP values.
* Voice Mail Plan (Yes): Interestingly, having a voicemail plan tends to decrease the likelihood of churn, as indicated by negative SHAP values.
* Total Evening Minutes and Total International Calls: These features also influence churn, though to a lesser extent than the top features.

**Model Evaluation and Insights for Predicting Customer Churn at SyriaTel**

**Addressing SyriaTel's Key Questions**

**1. What is the best model for predicting customer churn?**

After thoroughly comparing various models, including Decision Tree, K-Nearest Neighbors (KNN), and Random Forest, **the Random Forest model stands out as the best overall performer**. This model consistently demonstrated superior results across all key metrics:

* Accuracy: 93%
* F1-Score: 0.75
* ROC-AUC Score: 0.85
* Precision: 0.75
* Recall: 0.75

Given its robust performance, we recommend deploying the Random Forest model for predicting customer churn. By utilizing this model, SyriaTel will benefit from a highly reliable tool that not only accurately forecasts churn but also provides deep insights into the factors driving it. This will enable the company to implement more effective, targeted retention strategies.

**2. How accurately can the model predict customer churn?**

The Random Forest model's performance, measured by accuracy, precision, recall, F1-score, and ROC-AUC score, indicates that it can accurately predict customer churn. Specifically:

* Confusion Matrix: The model correctly identified 541 non-churning customers and 76 churning customers, with minimal false positives and false negatives.
* Accuracy: The overall accuracy of 93% ensures that SyriaTel can confidently identify at-risk customers, focusing retention efforts where they are most needed.

This high level of accuracy directly supports the effectiveness of SyriaTel's retention strategies by minimizing errors in identifying customers likely to churn.

**3. Which features are most influential in predicting customer churn?**

Insights from SHAP Analysis:

The SHAP summary plot provides critical insights into the most influential features driving the Random Forest model's predictions:

* International Plan Usage: Customers with an international plan ("international plan\_yes") are more likely to churn, making this feature a critical factor in retention strategies.
* Customer Service Interactions: Frequent customer service calls are strong indicators of potential churn, particularly when issues remain unresolved.
* Total Day Minutes: Higher usage during daytime ("total day minutes") correlates with increased churn risk, suggesting that heavy users during peak hours might be less satisfied with the service.
* Voicemail Plan: Having a voicemail plan ("voice mail plan\_yes") is associated with lower churn, indicating that bundling this service with other plans could enhance customer retention.

These insights enable SyriaTel to prioritize its retention efforts effectively. For instance, customers frequently contacting support may benefit from proactive follow-ups or personalized offers, while high-usage customers could be targeted with specialized plans that better meet their needs.

**Recommendations**

1. Deploy the Random Forest Model:

* Integrate the Random Forest model into SyriaTel's CRM system for real-time churn prediction.
* Regularly update the model with new data to maintain and improve its predictive accuracy.

1. Enhance Customer Service:

* Implement proactive support strategies to reduce churn among customers with frequent service interactions.
* Improve feedback mechanisms to identify and address customer pain points early.

1. Tailor Retention Strategies Based on Feature Importance:

* Develop and offer specialized plans for high-usage customers to increase loyalty.
* Implement loyalty programs with incentives for international plan users and other high-risk segments identified by the model.

1. Monitor and Optimize:

* Continuously monitor the model's predictions and the effectiveness of retention campaigns.
* Use data-driven insights to refine retention strategies and improve overall customer satisfaction.

**Conclusion**

By addressing these key questions with a data-driven approach and implementing the recommended strategies, SyriaTel will be well-equipped to reduce customer churn. The deployment of the Random Forest model, coupled with targeted retention strategies based on the most influential features, will not only improve customer satisfaction but also enhance the company's financial performance. This proactive shift from understanding to action will enable SyriaTel to maintain a competitive edge in the telecommunications industry.