

- Student name: Ian Kiptoo
- Student pace: Full-time Remote
- Scheduled project review date/time:
- Instructor name: Asha Deen

## 1. Business Understanding

### Project Overview

In the highly competitive telecommunications market, Syriatel faces the critical challenge of retaining its customer base while ensuring high levels of satisfaction. The company struggles with identifying at-risk customers who might churn and addressing their issues proactively. This problem is compounded by the need to offer personalized services and maintain optimal network performance, all while adapting to ever-evolving customer needs and preferences. Without effective data analytics and machine learning techniques, Syriatel risks losing market share and falling behind competitors who are more adept at leveraging these technologies to enhance customer experiences and loyalty.

### Introduction

Syriatel (Arabic: **سيريتل**) is a prominent mobile network provider in Syria. Alongside MTN Syria, it has been a key player in the country's telecommunications landscape. In 2022, the Syrian telecommunications authority awarded a third telecom license to Wafa Telecom, introducing new competition into the market. Syriatel offers LTE services with speeds up to 150 Mb/s under the brand name Super Surf. In this increasingly competitive environment, it is imperative for Syriatel to continuously adapt and enhance its services to remain competitive and provide outstanding customer experiences.

### Stakeholders

The success of the project is intrinsically tied to the satisfaction of a diverse set of stakeholders:

- **Syriatel Executives:** Leadership needs to maintain and expand their customer base, ensuring the long-term growth and sustainability of the company.
- **Syriatel Marketing Team:** The marketing department aims to increase customer acquisition, engagement, and the delivery of targeted promotions.
- **Syriatel Customer Support Team:** The customer support team seeks to provide efficient and effective customer service, resolving issues, and improving overall customer satisfaction.
- **Syriatel Customers:** The end-users, who expect reliable, affordable, and innovative telecommunications services.

### Business Problem

This project primarily focuses on addressing customer churn and improving customer satisfaction for Syriatel. We will leverage data analytics and machine learning techniques to gain

insights into customer behavior, preferences, and needs. Recommendations and strategies will be developed to mitigate customer churn and improve customer experiences.

Customer churn (the loss of customers to competition) presents a significant challenge for telecom companies like Syriatel because it is more expensive to acquire a new customer than to retain an existing one. Most telecom companies suffer from voluntary churn, which has a strong impact on the lifetime value of a customer by affecting both the length of service and future revenue. For instance, a company with a 25% churn rate has an average customer lifetime of four years, while a company with a 50% churn rate has an average customer lifetime of only two years. It is estimated that 75 percent of the 17 to 20 million subscribers signing up with a new wireless carrier every year are coming from another wireless provider, indicating a high rate of churn. Telecom companies invest substantial resources to acquire new customers, and when a customer leaves, the company not only loses future revenue but also the resources spent on acquisition. This erosion of profitability underscores the importance of mitigating churn.

By implementing advanced data analytics and machine learning techniques, Syriatel can develop effective strategies to identify at-risk customers, understand their reasons for potential departure, and deploy targeted interventions to retain them. These efforts will not only reduce churn but also enhance overall customer satisfaction and loyalty, thereby contributing to Syriatel's long-term success and competitive edge.

## Objective:

The aim of this analysis is to leverage data analytics and machine learning techniques to address the challenge of customer churn and improve customer satisfaction for Syriatel, a leading mobile network provider in Syria.

## Business Questions:

The key objectives for this project include:

### Identifying At-Risk Customers:

- Develop predictive models to identify customers at risk of churning based on their behavior and interaction patterns.
- Implement proactive interventions to retain at-risk customers and reduce churn rates.

### Understanding Customer Preferences:

- Analyze customer data to gain insights into preferences, needs, and satisfaction levels.
- Tailor services and offerings to better meet customer expectations and enhance satisfaction.

### Recommendation of Retention Strategies:

- Develop targeted retention strategies, such as personalized promotions and loyalty programs, to mitigate churn.
- Implement proactive customer support initiatives to address potential reasons for churn.

## Enhancing Customer Experience:

- Assess and optimize existing customer support processes and services to improve overall customer experience.
- Identify areas for improvement and implement measures to enhance satisfaction levels.

## Data Understanding

# Data Understanding

## Features

1. **State** (Categorical): The state in which the customer resides.
2. **Account Length** (Numerical): The number of days the customer has been with the company.
3. **Area Code** (Categorical): The area code associated with the customer's phone number.
4. **Phone Number** (Categorical): The customer's phone number, typically treated as an identifier.
5. **International Plan** (Categorical): Whether the customer has an international calling plan (e.g., "yes" or "no").
6. **Voice Mail Plan** (Categorical): Whether the customer has a voicemail plan (e.g., "yes" or "no").
7. **Number of Voicemail Messages** (Numerical - discrete): The number of voicemail messages received by the customer.
8. **Total Day Minutes** (Numerical): The total number of minutes the customer used during the daytime.
9. **Total Day Calls** (Numerical - discrete): The total number of calls made by the customer during the daytime.
10. **Total Day Charge** (Numerical): The total charges incurred for daytime usage.
11. **Total Evening Minutes** (Numerical): The total number of minutes the customer used in the evening.
12. **Total Evening Calls** (Numerical - discrete): The total number of calls made by the customer in the evening.
13. **Total Evening Charge** (Numerical): The total charges incurred for evening usage.
14. **Total Night Minutes** (Numerical): The total number of minutes the customer used at night.
15. **Total Night Calls** (Numerical - discrete): The total number of calls made by the customer at night.
16. **Total Night Charge** (Numerical): The total charges incurred for nighttime usage.
17. **Total International Minutes** (Numerical): The total number of international minutes used by the customer.
18. **Total International Calls** (Numerical - discrete): The total number of international calls made by the customer.
19. **Total International Charge** (Numerical - discrete): The total charges incurred for international calls.

20. **Customer Service Calls** (Numerical - discrete): The number of customer service calls made by the customer.

## Target Variable

1. **Churn**: Whether the customer has churned (1 for "yes" and 0 for "no").

## 2.1 Exploratory Data Analysis

First, let's perform **Exploratory Data Analysis** to understand our dataset better. This initial step will provide valuable insights into the data and its basic characteristics. Here's what I do during the EDA:

### i. Data Import and Inspection:

- I start by importing the dataset and printing the first 5 rows to understand its structure and get a glimpse of the initial records.

### ii. Data Shape:

- Next, I check the shape of our dataset to understand its dimensions, including the number of rows and columns, providing an overview of its size.

### iii. Data Types:

- Then, I examine the data types of each column to ensure they are correctly interpreted, as this is crucial for subsequent analysis.

### iv. Data Summary:

- I utilize the `describe` function to generate summary statistics for numerical columns, including count, mean, standard deviation, and quartiles. This provides insights into the distribution of the data.

### v. Unique Values in Categorical Columns:

- Lastly, for categorical columns, I explore the unique values to understand the diversity of categories and ensure there are no unexpected or erroneous entries.

```
# All necessary imports for data preprocessing
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import numpy as np
from scipy.stats import skew, kurtosis

%matplotlib inline
warnings.filterwarnings("ignore")
```

```

def read_data(file_path):
    """
    Read the dataset from the given file path.

    Parameters:
    - file_path (str): The path to the CSV file containing the
    dataset.

    Returns:
    - DataFrame: The DataFrame containing the dataset.
    """
    import pandas as pd

    # Read the CSV file
    df = pd.read_csv(file_path)

    return df

# Specify the file path
file_path = r'C:\Users\Ian\Downloads\
bigml_59c28831336c6604c800002a.csv'

# Call the function to read the data
df = read_data(file_path)

def exploratory_data_analysis(df):
    """
    Perform exploratory data analysis on the given DataFrame.

    Parameters:
    - df (DataFrame): The DataFrame containing the dataset.

    Returns:
    - None
    """
    # i. Data Import and Inspection
    print("Data Import and Inspection:")
    print("First 5 rows of the dataset:")
    print(df.head())

    # ii. Data Shape
    print("\nData Shape:")
    print(f"Number of rows: {df.shape[0]}")
    print(f"Number of columns: {df.shape[1]}")

    # iii. Data Types
    print("\nData Types:")
    print(df.dtypes)

    # iv. Data Summary
    print("\nData Summary:")

```

```
print(df.describe())

# v. Unique Values in Categorical Columns
print("\nUnique Values in Categorical Columns:")
for col in df.select_dtypes(include=['object']).columns:
    print(f"Unique values in {col}: {df[col].unique()}")
```

*# Call the function with the DataFrame*  
exploratory\_data\_analysis(df)

Data Import and Inspection:

First 5 rows of the dataset:

	state	account length	area	code	phone number	international	plan	\
0	KS	128		415	382-4657		no	
1	OH	107		415	371-7191		no	
2	NJ	137		415	358-1921		no	
3	OH	84		408	375-9999		yes	
4	OK	75		415	330-6626		yes	

	voice mail plan	number	vmail messages	total day minutes	total day calls	\
0	yes		25	265.1		
110						
1	yes		26	161.6		
123						
2	no		0	243.4		
114						
3	no		0	299.4		
71						
4	no		0	166.7		
113						

	total day charge	total eve minutes	total eve calls	total eve charge	\
0	45.07	197.4	99		
16.78					
1	27.47	195.5	103		
16.62					
2	41.38	121.2	110		
10.30					
3	50.90	61.9	88		
5.26					
4	28.34	148.3	122		
12.61					

	total night minutes	total night calls	total night charge	\
0	244.7	91	11.01	
1	254.4	103	11.45	
2	162.6	104	7.32	
3	196.9	89	8.86	

4	186.9	121	8.41
	total intl minutes	total intl calls	total intl charge \
0	10.0	3	2.70
1	13.7	3	3.70
2	12.2	5	3.29
3	6.6	7	1.78
4	10.1	3	2.73

	customer service calls	churn
0	1	False
1	1	False
2	0	False
3	2	False
4	3	False

Data Shape:

Number of rows: 3333

Number of columns: 21

Data Types:

state	object
account length	int64
area code	int64
phone number	object
international plan	object
voice mail plan	object
number vmail messages	int64
total day minutes	float64
total day calls	int64
total day charge	float64
total eve minutes	float64
total eve calls	int64
total eve charge	float64
total night minutes	float64
total night calls	int64
total night charge	float64
total intl minutes	float64
total intl calls	int64
total intl charge	float64
customer service calls	int64
churn	bool
dtype:	object

Data Summary:

	account length	area code	number vmail messages	total day minutes \
count	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	

179.775098			
std	39.822106	42.371290	13.688365
54.467389			
min	1.000000	408.000000	0.000000
0.000000			
25%	74.000000	408.000000	0.000000
143.700000			
50%	101.000000	415.000000	0.000000
179.400000			
75%	127.000000	510.000000	20.000000
216.400000			
max	243.000000	510.000000	51.000000
350.800000			

	total day calls	total day charge	total eve minutes	total eve
calls \				
count	3333.000000	3333.000000	3333.000000	
3333.000000				
mean	100.435644	30.562307	200.980348	
100.114311				
std	20.069084	9.259435	50.713844	
19.922625				
min	0.000000	0.000000	0.000000	
0.000000				
25%	87.000000	24.430000	166.600000	
87.000000				
50%	101.000000	30.500000	201.400000	
100.000000				
75%	114.000000	36.790000	235.300000	
114.000000				
max	165.000000	59.640000	363.700000	
170.000000				

	total eve charge	total night minutes	total night calls \
count	3333.000000	3333.000000	3333.000000
mean	17.083540	200.872037	100.107711
std	4.310668	50.573847	19.568609
min	0.000000	23.200000	33.000000
25%	14.160000	167.000000	87.000000
50%	17.120000	201.200000	100.000000
75%	20.000000	235.300000	113.000000
max	30.910000	395.000000	175.000000

	total night charge	total intl minutes	total intl calls \
count	3333.000000	3333.000000	3333.000000
mean	9.039325	10.237294	4.479448
std	2.275873	2.791840	2.461214
min	1.040000	0.000000	0.000000
25%	7.520000	8.500000	3.000000
50%	9.050000	10.300000	4.000000



75%	10.590000	12.100000	6.000000
max	17.770000	20.000000	20.000000

	total intl charge	customer service calls
count	3333.000000	3333.000000
mean	2.764581	1.562856
std	0.753773	1.315491
min	0.000000	0.000000
25%	2.300000	1.000000
50%	2.780000	1.000000
75%	3.270000	2.000000
max	5.400000	9.000000

Unique Values in Categorical Columns:

Unique values in state: ['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'LA' 'WV' 'IN' 'RI' 'IA' 'MT' 'NY' 'ID' 'VT' 'VA' 'TX' 'FL' 'CO' 'AZ' 'SC' 'NE' 'WY' 'HI' 'IL' 'NH' 'GA' 'AK' 'MD' 'AR' 'WI' 'OR' 'MI' 'DE' 'UT' 'CA' 'MN' 'SD' 'NC' 'WA' 'NM' 'NV' 'DC' 'KY' 'ME' 'MS' 'TN' 'PA' 'CT' 'ND']

Unique values in phone number: ['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']

Unique values in international plan: ['no' 'yes']

Unique values in voice mail plan: ['yes' 'no']

## Data Import and Inspection:

First 5 rows of the dataset:

The dataset consists of various features including state, account length, area code, phone number, international plan, voice mail plan, number of voicemail messages, and several numerical attributes such as total day minutes, total evening minutes, total night minutes, total international minutes, and customer service calls. The churn column indicates whether a customer has churned, with "False" representing no churn and "True" representing churn.

### Data Shape:

The dataset contains 3333 rows and 21 columns, indicating that there are 3333 observations and 21 different features.

### Data Types:

- The 'state', 'phone number', 'international plan', and 'voice mail plan' columns are of object type, indicating categorical variables.
- The 'account length', 'area code', 'number vmail messages', 'total day calls', 'total day minutes', 'total day charge', 'total eve calls', 'total eve minutes', 'total eve charge', 'total night calls', 'total night minutes', 'total night charge', 'total intl calls', 'total intl minutes', 'total intl charge', and 'customer service calls' columns are of numerical type.
- The 'churn' column is of boolean type, representing whether a customer churned or not.

## Data Summary:

The summary statistics for numerical columns provide insights into the distribution of data, including count, mean, standard deviation, and quartiles. For example, the mean 'account length' is approximately 101.06 days, and the mean 'total day minutes' is approximately 179.78 minutes.

## Unique Values in Categorical Columns:

- The 'state' column contains unique abbreviations for different states, ranging from 'KS' to 'WY'.
- The 'phone number' column contains unique phone numbers.
- The 'international plan' column has two unique values, 'no' and 'yes', indicating whether a customer has an international calling plan.
- The 'voice mail plan' column also has two unique values, 'yes' and 'no', indicating whether a customer has a voicemail plan.

## 2.1.1 Univariate Analysis

We first start exploring individual variables.

We divide the data into numerical and categorical variables so as to perform the EDA them separately.

Visualizing the numerical and categorical variables in our dataset.

```
class DataVariables:
    def __init__(self, dataframe):
        self.df = dataframe

    def get_numerical_vars(self):
        numerical_vars = self.df.select_dtypes(include=['int64',
'float64']).columns.tolist()
        numerical_vars.remove('area code') # Remove 'area code' from
numerical variables
        return numerical_vars

    def visualize_outliers(self):
        # Numerical variables
        numerical_vars = self.get_numerical_vars()

        # Calculate the number of subplots needed
        num_plots = len(numerical_vars)
        num_rows = (num_plots // 3) + (num_plots % 3 > 0) # Round up
to the nearest integer

        # Boxplot for each numerical variable to visualize outliers
        plt.figure(figsize=(15, 5*num_rows))
        for i, col in enumerate(numerical_vars, start=1):
            plt.subplot(num_rows, 3, i)
```

```

        sns.boxplot(x=self.df[col])
        plt.title(col)
    plt.tight_layout()
    plt.show()

def visualize_numerical_distribution(self):
    # Numerical variables
    numerical_vars = self.get_numerical_vars()

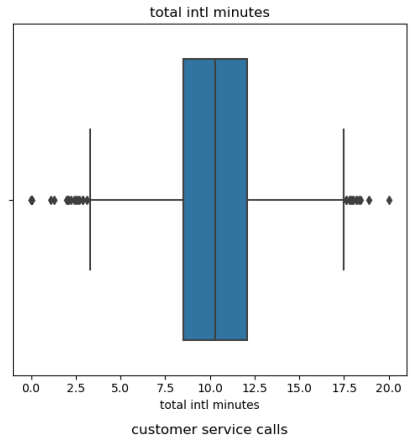
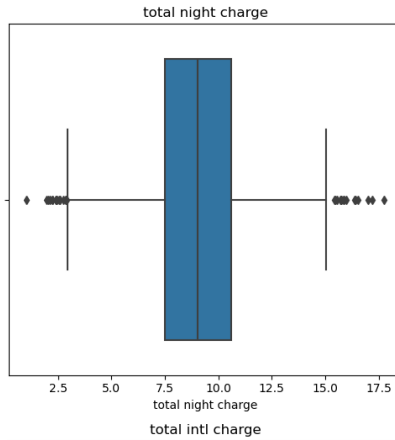
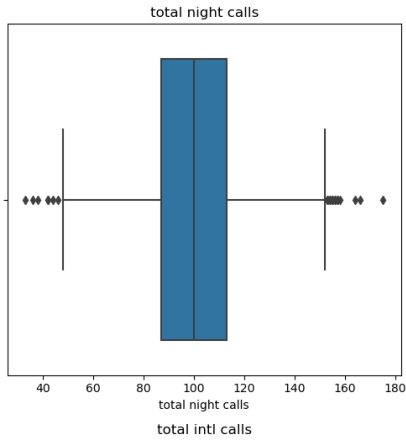
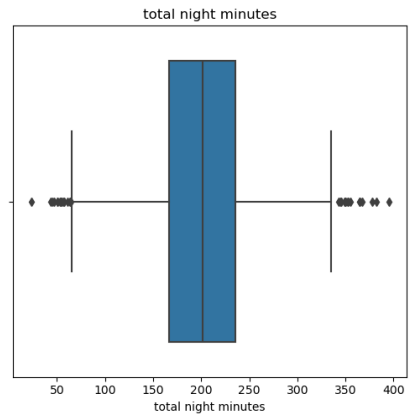
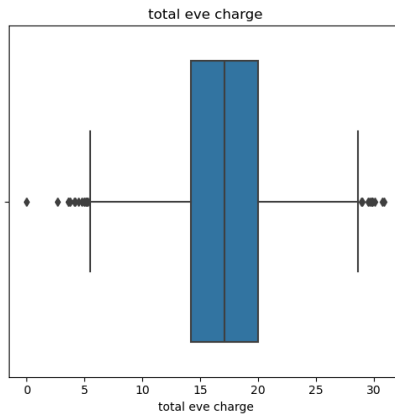
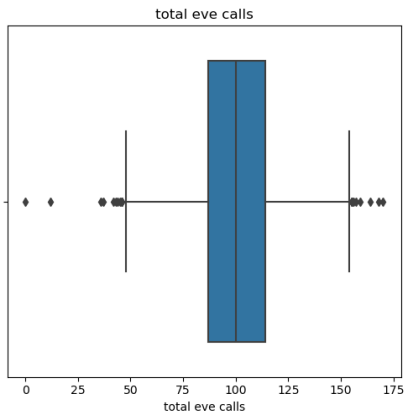
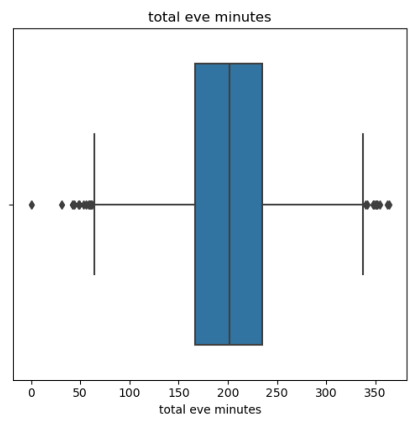
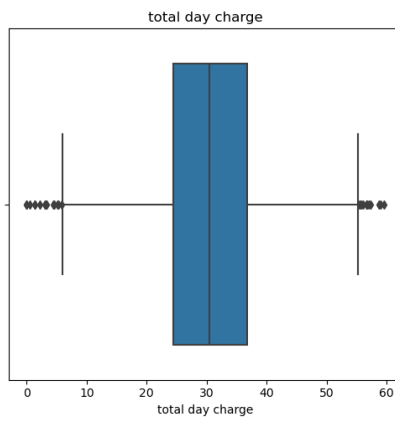
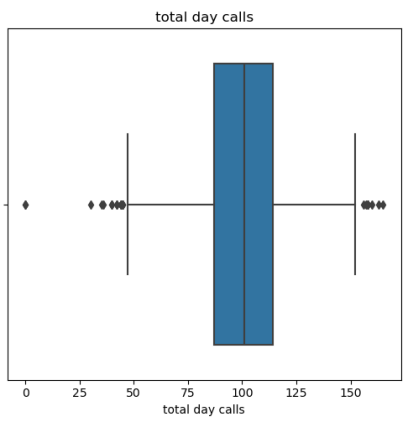
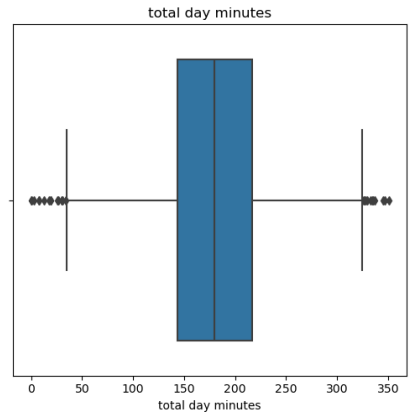
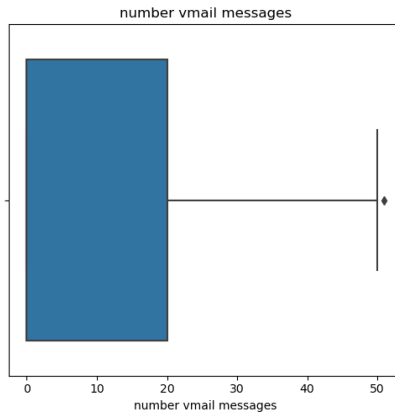
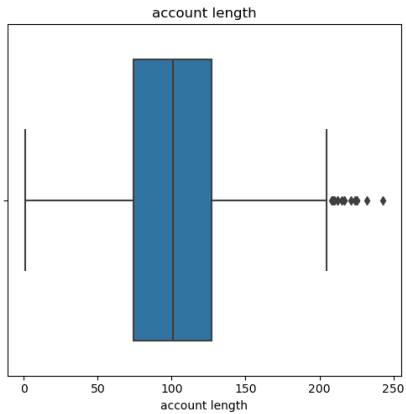
    # Calculate the number of subplots needed
    num_plots = len(numerical_vars)
    num_rows = (num_plots // 3) + (num_plots % 3 > 0) # Round up to the nearest integer

    # Histogram for each numerical variable to visualize distribution
    plt.figure(figsize=(15, 5*num_rows))
    for i, col in enumerate(numerical_vars, start=1):
        plt.subplot(num_rows, 3, i)
        sns.histplot(self.df[col], kde=True)
        plt.title(col)
    plt.tight_layout()
    plt.show()

data_vars = DataVariables(df)

# Visualizing Outliers
data_vars.visualize_outliers()

```



## Interpretation:

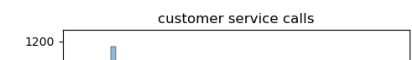
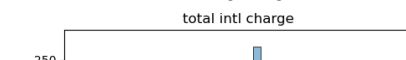
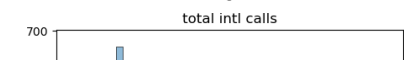
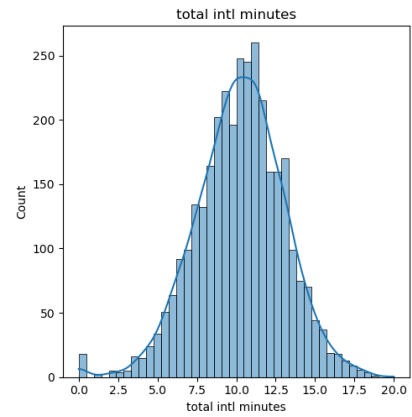
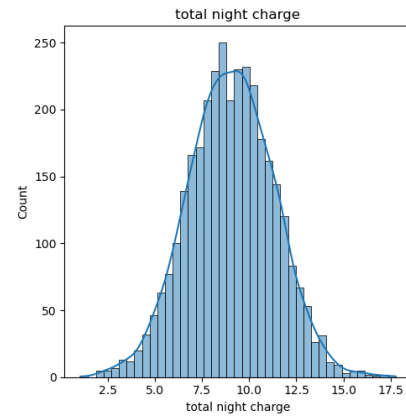
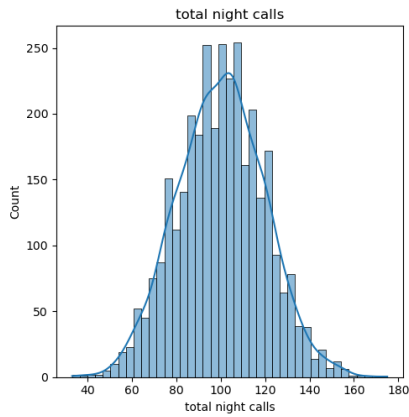
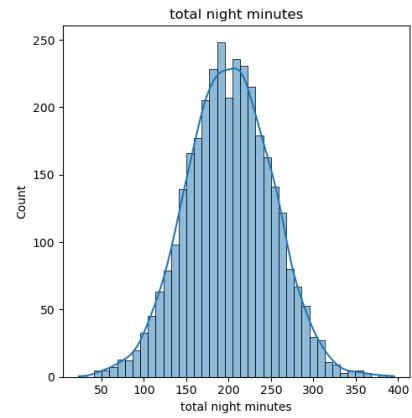
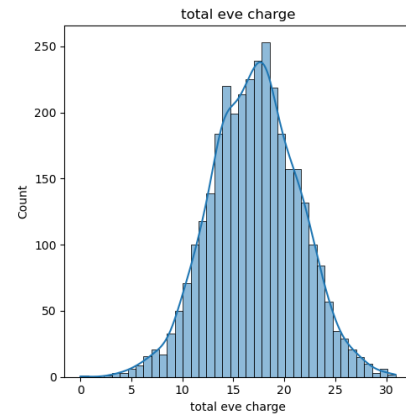
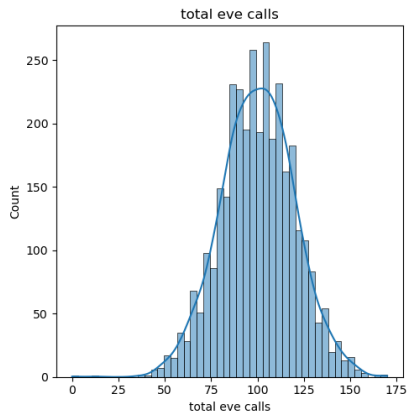
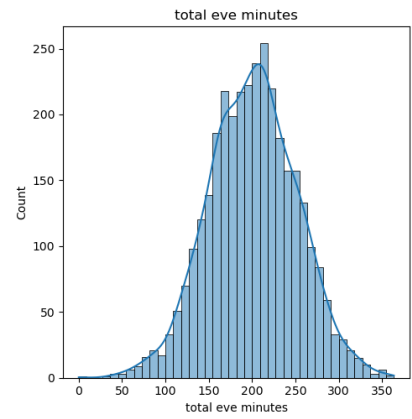
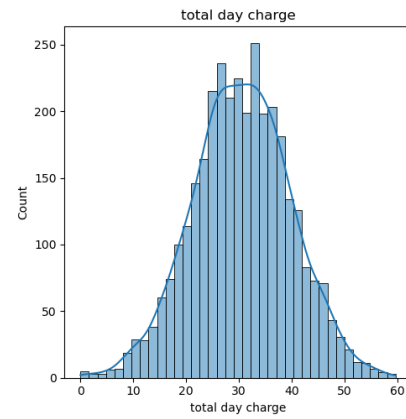
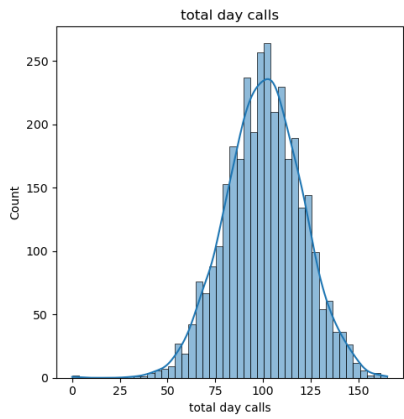
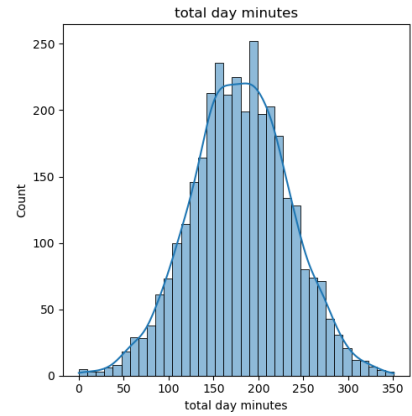
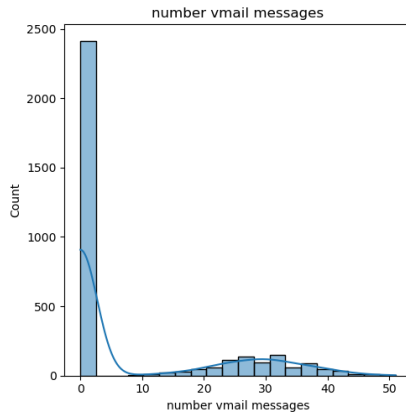
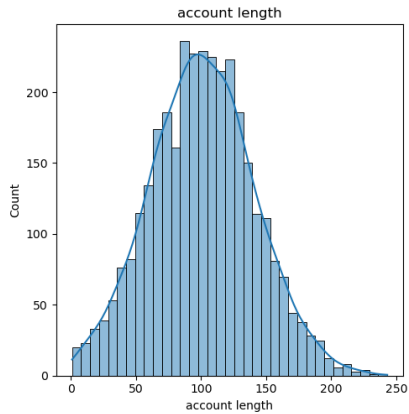
1. **Account Length:** The boxplot reveals the presence of outliers with exceptionally long account lengths. These outliers suggest the existence of customers who have been with the company for an extended period compared to the majority.
2. **Number Vmail Messages:** The majority of customers receive few or no voicemail messages, as indicated by the concentration of data points at lower values. However, there are outliers representing customers who receive a significant number of voicemail messages, which may indicate a specific usage pattern or preference among these customers.
3. **Total Day Calls:** The boxplot shows outliers with both very low and very high total day call counts. This indicates variability in the volume of daytime calls made by customers. The presence of outliers suggests that some customers have distinct calling behaviors compared to the majority.
4. **Total Intl Minutes:** Most customers make shorter international calls, as evidenced by the concentration of data points at lower values. However, outliers exist, representing customers who make longer international calls. These outliers suggest diversity in international calling patterns among customers.
5. **Total Intl Calls:** While most customers make a modest number of international calls, outliers exist with higher-than-average international call counts. These outliers indicate a subgroup of customers who engage in more frequent international calling, potentially indicating specific communication needs or preferences.
6. **Customer Service Calls:** The majority of customers have one or two service calls, as indicated by the central tendency of the data. However, outliers with a high number of service calls are present. These outliers may represent customers experiencing recurring issues or those requiring extensive support, highlighting the importance of addressing their concerns effectively.

In summary, the presence of outliers across various numerical variables suggests diverse customer behaviors and needs within the dataset. Understanding these outliers can provide valuable insights for customer segmentation, service optimization, and targeted marketing strategies.

## Distribution of the data:

Now, let's delve into the distribution of our data to gain insights into its spread and variability.

```
# Visualizing distribution
data_vars.visualize_numerical_distribution()
```



## Interpretation:

- The plot highlights that the numerical variables in the dataset are not on the same scale, indicating the need for scaling before modeling.
- The majority of numerical variables are continuous, except for **customer service calls** and **total intl calls**, which are discrete.
- Most of the numerical variables exhibit a roughly normal distribution, with some exceptions such as **customer service calls**, **total intl calls**, and **number of voicemail messages**, which display positive skewness. These characteristics will be further analyzed in subsequent sections.
- **Account Length:** The distribution of account lengths is slightly positively skewed, meaning that there are more customers with shorter account lengths than with longer ones. The distribution is also slightly platykurtic, indicating that it has fewer extreme values and is less peaked compared to a normal distribution.
- **Number of Voicemail Messages:** The distribution of the number of voicemail messages is positively skewed, suggesting that most customers receive few or no voicemail messages, with a few outliers receiving a significant number of messages. The distribution is also platykurtic, indicating fewer extreme values and less peakedness compared to a normal distribution.
- **Total Day Minutes:** The distribution of total daytime minutes is approximately symmetric, with a slight negative skewness, suggesting a slightly longer tail on the left side. The distribution is also close to mesokurtic, indicating a similar level of tailedness compared to a normal distribution.
- **Total Day Calls:** The distribution of total daytime calls is negatively skewed, implying that there are more customers with fewer calls than with more calls. The distribution is leptokurtic, indicating heavier tails and a sharper peak compared to a normal distribution.
- **Total Day Charge:** The distribution of total daytime charges is similar to the distribution of total daytime minutes, with a slight negative skewness and close to mesokurtic.
- **Total Evening Minutes, Total Evening Calls, Total Evening Charge, Total Night Minutes, Total Night Calls, and Total Night Charge:** These variables have distributions that are approximately symmetric and close to mesokurtic, indicating similar tailedness and peakedness to a normal distribution.
- **Total International Minutes, Total International Calls, and Total International Charge:** The distributions of these variables are positively skewed, suggesting that most customers make shorter international calls or fewer international calls, with some outliers making longer or more international calls. The distributions are also leptokurtic, indicating heavier tails and sharper peaks compared to a normal distribution.

- **Customer Service Calls:** The distribution of the number of customer service calls is positively skewed, indicating that most customers make few service calls, with some outliers making a larger number of calls. The distribution is also leptokurtic, suggesting heavier tails and a sharper peak compared to a normal distribution.

## Skewness and Kurtosis

```
def calculate_skewness_kurtosis(df, numerical_vars):
    """
    Calculate skewness and kurtosis for numerical columns in a
    DataFrame.

    Args:
    - df (DataFrame): The DataFrame containing the data.
    - numerical_vars (list): List of numerical variable names.

    Returns:
    - DataFrame: DataFrame containing skewness and kurtosis values for
    each numerical variable.
    """
    # Calculate skewness and kurtosis for numerical columns
    skewness_values = df[numerical_vars].apply(skew)
    kurtosis_values = df[numerical_vars].apply(kurtosis)

    # Create a DataFrame to store the results
    skew_kurtosis_df = pd.DataFrame({'Skewness': skewness_values,
    'Kurtosis': kurtosis_values})

    return skew_kurtosis_df

# Select numerical columns
numerical_vars = df.select_dtypes(include=['int64',
'float64']).columns.tolist()

# Calculate skewness and kurtosis
skew_kurtosis_df = calculate_skewness_kurtosis(df, numerical_vars)
print(skew_kurtosis_df)
```

	Skewness	Kurtosis
account length	0.096563	-0.109474
area code	1.126316	-0.706374
number vmail messages	1.264254	-0.052852
total day minutes	-0.029064	-0.021710
total day calls	-0.111736	0.241017
total day charge	-0.029070	-0.021582
total eve minutes	-0.023867	0.023792
total eve calls	-0.055538	0.204048
total eve charge	-0.023847	0.023650
total night minutes	0.008917	0.083888
total night calls	0.032485	-0.073711



total night charge	0.008882	0.083735
total intl minutes	-0.245026	0.606472
total intl calls	1.320883	3.077165
total intl charge	-0.245176	0.606897
customer service calls	1.090868	1.726518

## Interpretation of Numerical Variables and Distributions

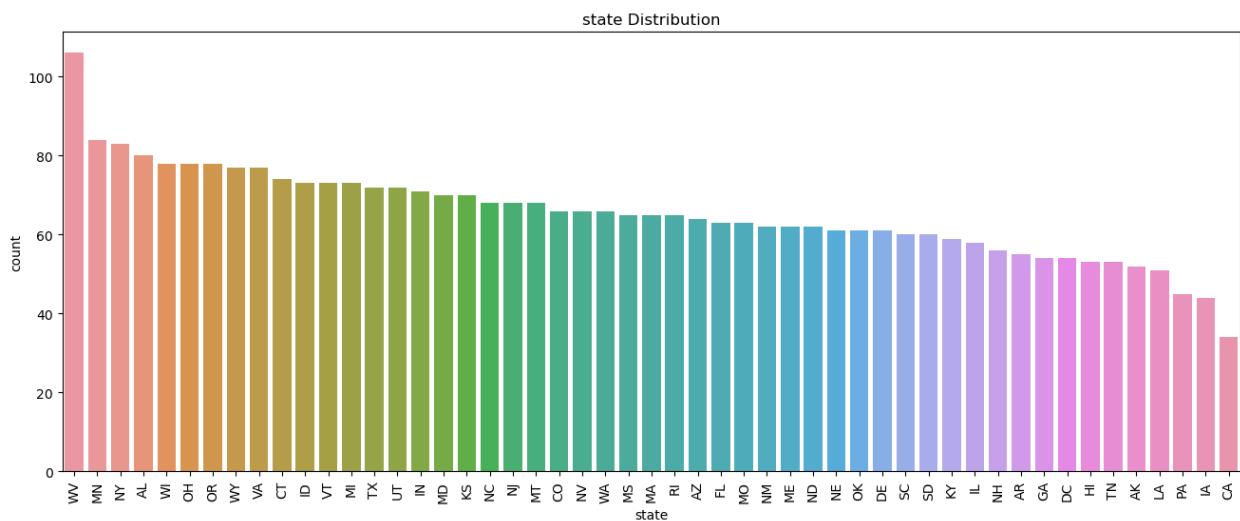
- **Account Length:** The distribution of account lengths is slightly positively skewed, with a skewness value of 0.097, indicating more customers with shorter account lengths than with longer ones. It's also slightly platykurtic, with a kurtosis value of -0.109, implying fewer extreme values compared to a normal distribution.
- **Number of Voicemail Messages:** This variable is positively skewed with a skewness value of 1.264, suggesting that most customers receive few or no voicemail messages, with some outliers receiving a significant number of messages. It also exhibits platykurtosis with a kurtosis value of -0.053, indicating fewer extreme values and less peakedness compared to a normal distribution.
- **Total Day Minutes:** The distribution of total daytime minutes is approximately symmetric, with a slight negative skewness of -0.029, suggesting a slightly longer tail on the left side. The kurtosis value of -0.022 indicates a distribution close to mesokurtic, meaning it has a similar level of tailedness compared to a normal distribution.
- **Total Day Calls:** This variable is negatively skewed with a skewness value of -0.112, indicating that more customers make fewer calls than more calls. The distribution is leptokurtic, with a kurtosis value of 0.241, indicating heavier tails and a sharper peak compared to a normal distribution.
- **Total Day Charge:** Similar to total daytime minutes, this variable shows a slight negative skewness of -0.029 and is close to mesokurtic with a kurtosis value of -0.022.
- **Total Evening Minutes, Total Evening Calls, Total Evening Charge, Total Night Minutes, Total Night Calls, and Total Night Charge:** These variables exhibit distributions that are approximately symmetric and close to mesokurtic.
- **Total International Minutes, Total International Calls, and Total International Charge:** The distributions of these variables are positively skewed, indicating that most customers make shorter or fewer international calls, with some outliers making longer or more international calls. They also demonstrate leptokurtosis, suggesting heavier tails and sharper peaks compared to a normal distribution.
- **Customer Service Calls:** This variable is positively skewed with a skewness value of 1.091, indicating that most customers make few service calls, with some outliers making a larger number of calls. It also demonstrates leptokurtosis with a kurtosis

value of 1.727, suggesting heavier tails and a sharper peak compared to a normal distribution.

## Distribution of customers per state.

```
# Calculate value counts and plot bar plots for categorical variables
categorical_cols = ["state"]

for col in categorical_cols:
    value_counts = df[col].value_counts().sort_values(ascending=False)
    plt.figure(figsize=(16, 6))
    sns.countplot(x=col, data=df, order=value_counts.index)
    plt.title(f"{col} Distribution")
    plt.xticks(rotation=90)
    plt.show()
```



```
categorical_cols = ["state"]

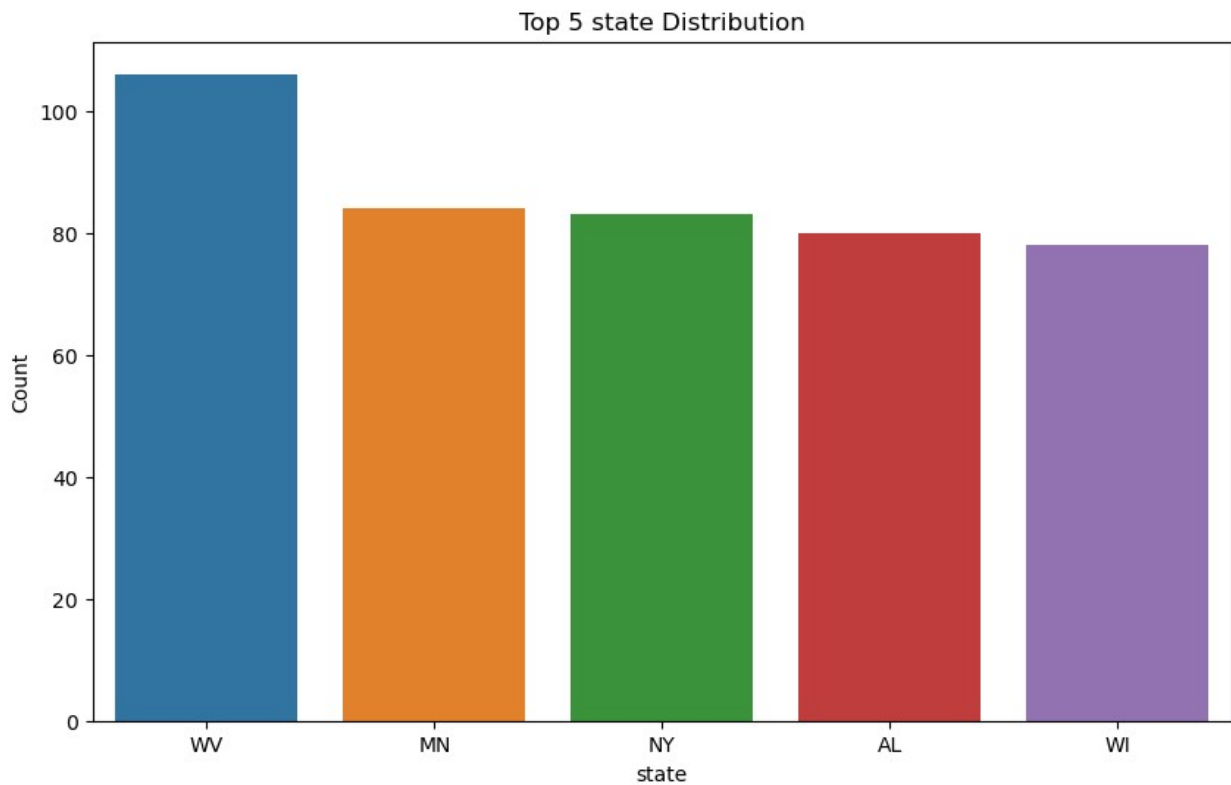
for col in categorical_cols:

    # Get the top 5 states
    top_5_states = value_counts.head(5)
    print(top_5_states)

    # Visualize the top 5 states
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_5_states.index, y=top_5_states.values)
    plt.title(f"Top 5 {col} Distribution")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.show()
```

```
state
WV    106
```

```
MN      84
NY      83
AL      80
WI      78
Name: count, dtype: int64
```



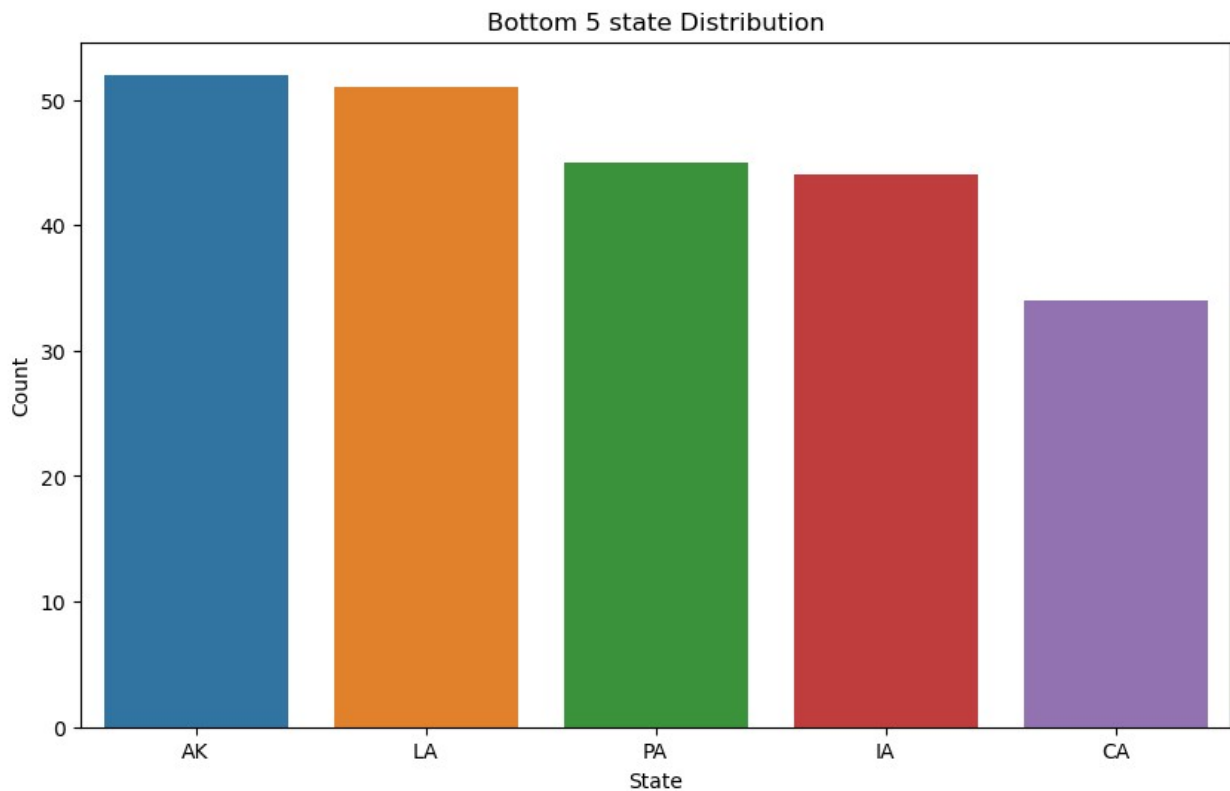
```
categorical_cols = ["state"]
for col in categorical_cols:

    # Get the bottom 5 states
    bottom_5_states = value_counts.tail(5)
    print(bottom_5_states)

    # Visualize the bottom 5 states
    plt.figure(figsize=(10, 6))
    sns.barplot(x=bottom_5_states.index, y=bottom_5_states.values)
    plt.title(f"Bottom 5 {col} Distribution")
    plt.xlabel(col.capitalize())
    plt.ylabel("Count")
    plt.show()

state
AK      52
```

```
LA      51
PA      45
IA      44
CA      34
Name: count, dtype: int64
```



West Virginia (WV) is the most frequent state in the dataset, appearing 106 times. Minnesota (MN) follows with 84 occurrences, making it the second most common state. New York (NY) is close behind with 83 occurrences. Alabama (AL), Wisconsin (WI), Oregon (OR), and Ohio (OH) each appear 78 times, placing them among the top states by frequency. California (CA) has the fewest occurrences at 34, indicating it is the least represented state in the dataset.

## Categorical variables analysis:

Now let's peek into the **categorical variables**:

```
# Set up the figure and axes for subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(25, 10))

# Flatten the axes array to simplify indexing
axs = axs.flatten()

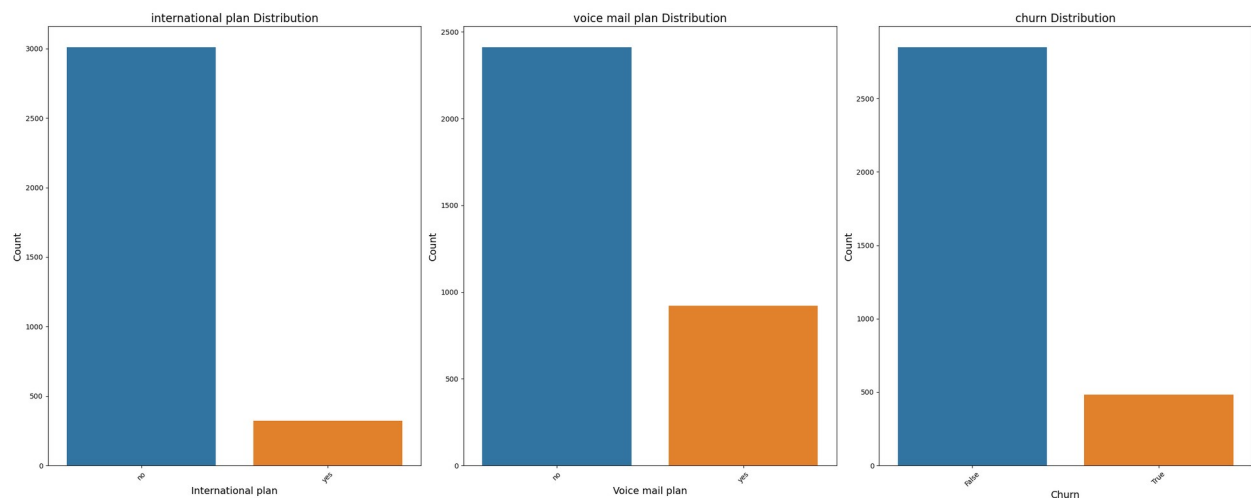
# Calculate value counts and plot bar plots for categorical variables
categorical_cols = ["international plan", "voice mail plan", "churn"]
```

```

for i, col in enumerate(categorical_cols):
    # Calculate value counts and sort by frequency
    value_counts = df[col].value_counts().sort_values(ascending=False)
    sns.countplot(x=col, data=df, order=value_counts.index, ax=axes[i])
    axes[i].set_title(f"{col} Distribution", fontsize=16)
    axes[i].set_xlabel(col.capitalize(), fontsize=14)
    axes[i].set_ylabel('Count', fontsize=14)
    axes[i].tick_params(axis='x', rotation=45)

# Adjust the layout and spacing
plt.tight_layout()
plt.show()

```



```

# Select categorical columns
categorical_vars = df.select_dtypes(include=['object',
'category']).columns.tolist()

# Print value counts for the categorical variables
print("Value counts for the categorical variables:\n")
for var in categorical_vars:
    print(f'{var}: {len(df[var].value_counts())}')

```

Value counts for the categorical variables:

```

state: 51
phone number: 3333
international plan: 2
voice mail plan: 2

```

- **States Presence:**
  - The countplot for the 'state' variable shows that there are 51 unique states present in the dataset.
  - Each state represents a geographical location where customers may reside or where the phone service is provided.

- **Area Codes:**
  - From the countplot or value counts output, we observe that there are three different area codes present in the dataset.
  - Area codes are geographical region codes assigned to specific telephone numbers. Having three different area codes suggests that the dataset covers customers from different regions or areas.
- **Phone Numbers:**
  - Each record in the dataset is represented by a unique phone number.
  - This suggests that each row in the dataset corresponds to a specific phone line or customer account, identified uniquely by their phone numbers.
- **Voice Mail Plan:**
  - The countplot or value counts output indicates whether a customer has been subscribed to a voice mail plan or not.
  - This binary variable suggests whether customers have opted for a voice mail service as part of their phone plan.
- **International Plan:**
  - Similar to the voice mail plan, the countplot or value counts output shows whether a customer has been subscribed to an international plan or not.
  - This binary variable indicates whether customers have opted for an international calling plan as part of their phone service.

## 2.1.2 Bi-Variate Analysis

Next let's check the relationships that the variables have in our data-set.

```
# Set display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

class BivariateAnalysis:
    def __init__(self, dataframe):
        self.df = dataframe

    def visualize_correlation_matrix(self):
        # Select only numerical columns
        numerical_vars = self.df.select_dtypes(include=['int64',
        'float64']).columns.tolist()
        numeric_vars =
self.df[numerical_vars].select_dtypes(include=[np.number])

        # Plot scatter matrix
        pd.plotting.scatter_matrix(numeric_vars, figsize=(20, 20))
        plt.tight_layout()
        plt.show()

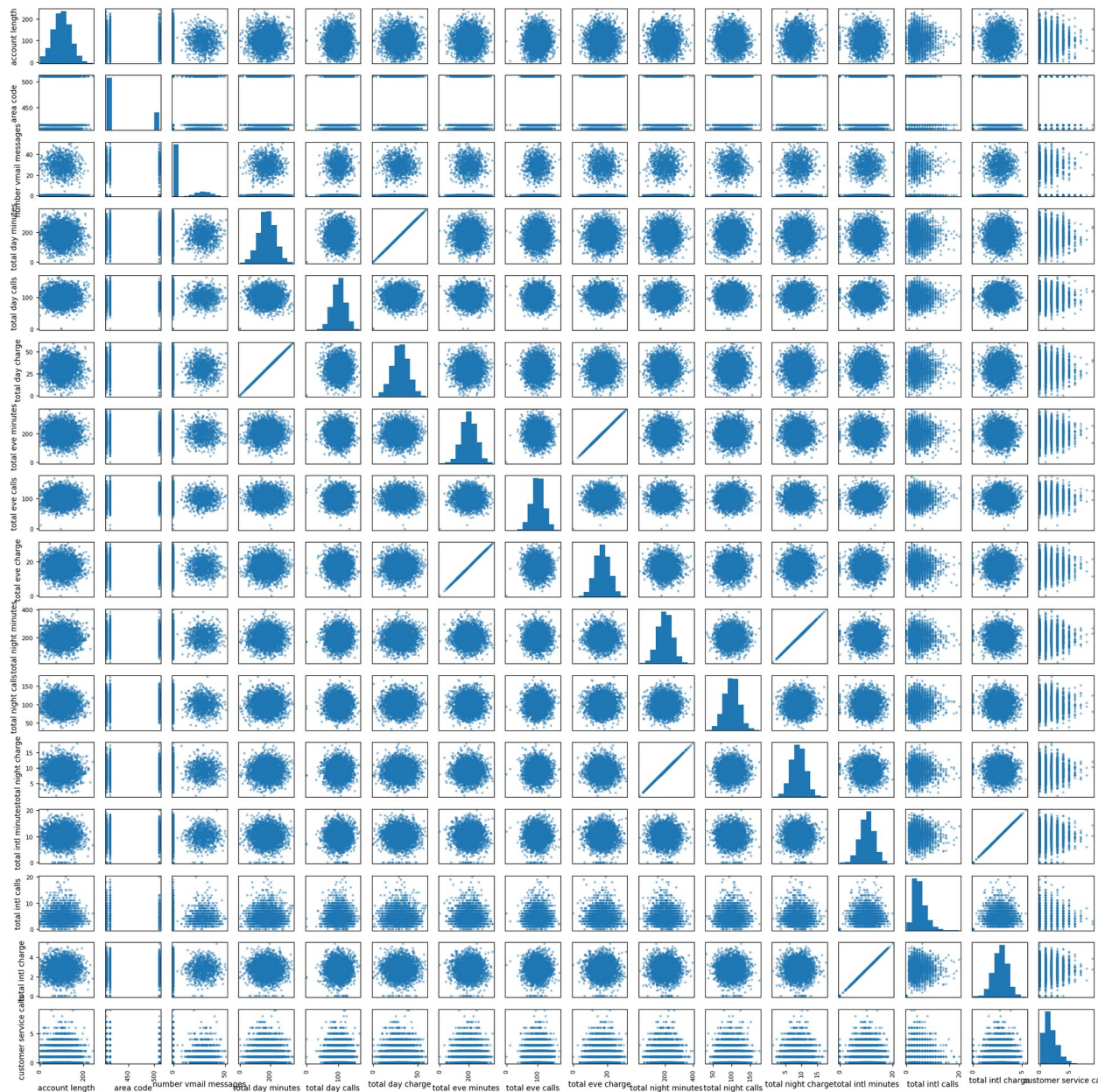
        # Calculate the correlation matrix
        correlation_matrix = numeric_vars.corr()
```

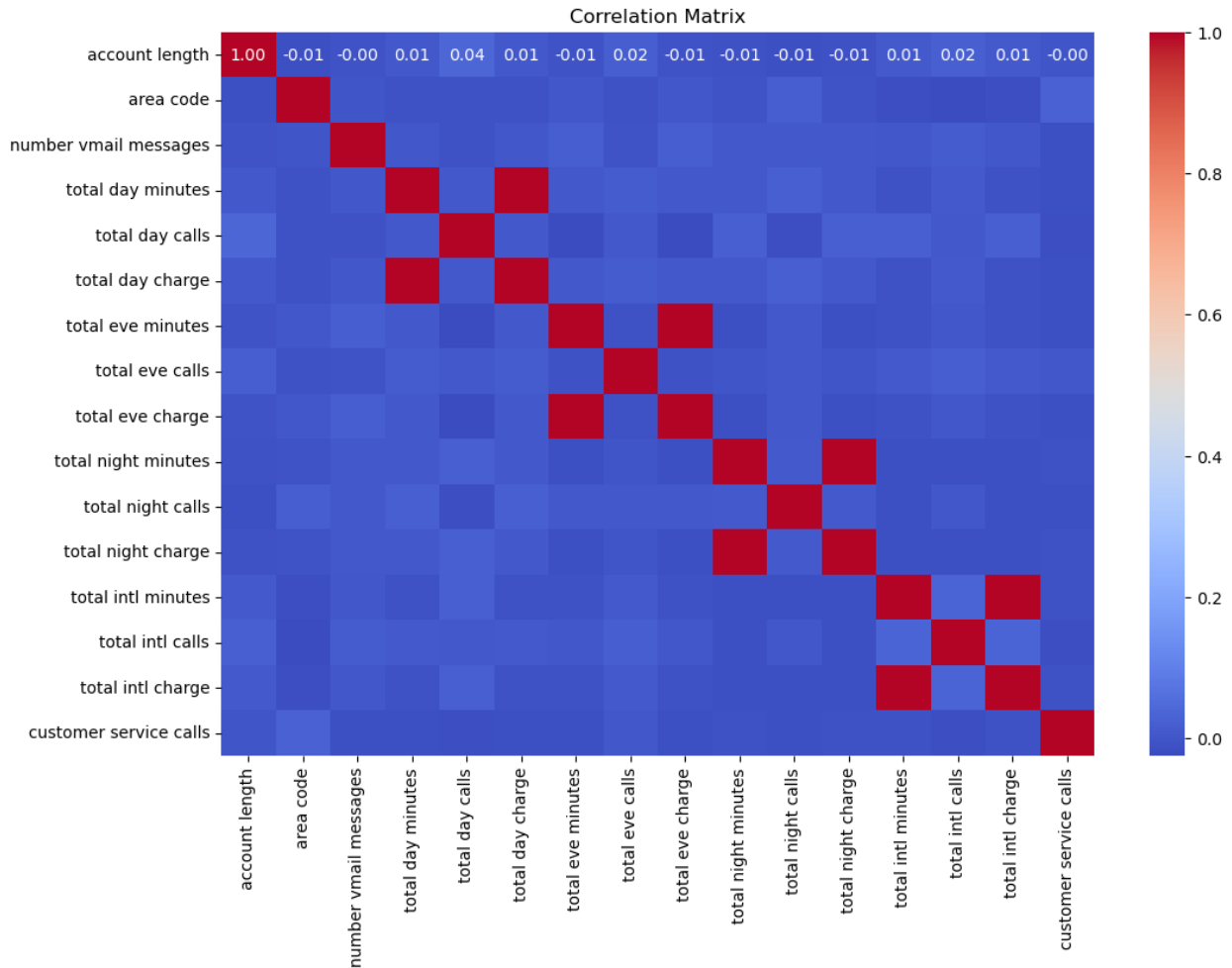
```

# Plot the correlation matrix using Seaborn heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# plot the relevant plots
bivariate_analysis = BivariateAnalysis(df)
bivariate_analysis.visualize_correlation_matrix()

```





"The heatmap indicates that certain variables exhibit high correlation, excluding those along the leading diagonal (which are identical). This correlation suggests the possibility of multicollinearity, a condition where predictors in a regression model are highly correlated. Multicollinearity will be addressed in the 'Checking for and removing multicollinearity (correlated predictors)' in the data preparation process.

The scatter plot matrix above reaffirms our observation from the heatmap, indicating that certain variables exhibit high correlation.

## Distribution of churn for each state.

```
# Group by "state" and "churn" to calculate counts
state_churn_counts = df.groupby(["state", "churn"]).size().unstack()

# Set up the figure size
plt.figure(figsize=(30, 10))

# Plot the bar chart
state_churn_counts.plot(kind='bar', stacked=False, figsize=(30, 10),
width=0.8)
```



```

# Add a title and labels
plt.title('Churn Distribution by State', fontsize=18,
fontweight="bold", pad=15)
plt.xlabel('State', fontsize=18, fontweight="bold", labelpad=15)
plt.ylabel('Count', fontsize=18, fontweight="bold", labelpad=15)

# Rotate x-axis labels for better visibility
plt.xticks(rotation=90, fontsize=15)
plt.yticks(fontsize=15)

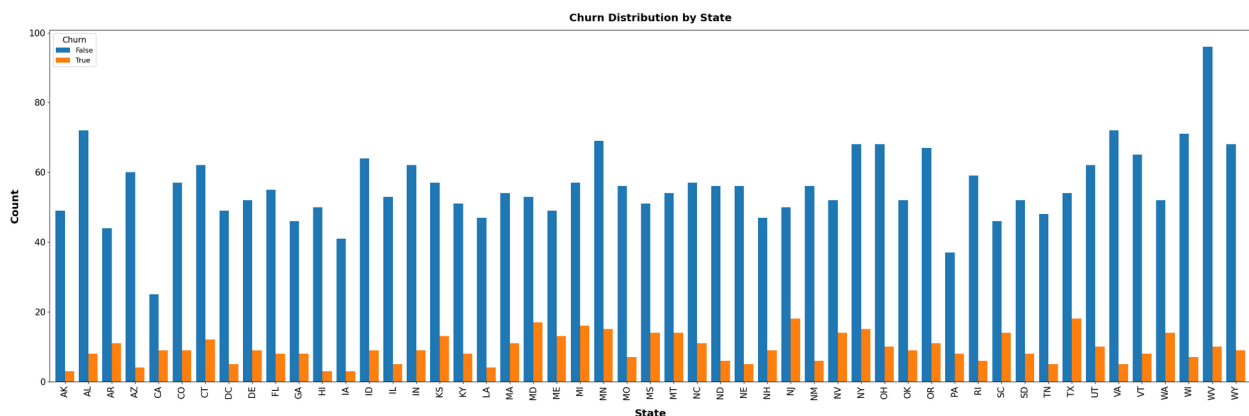
# Add a legend
plt.legend(title='Churn', title_fontsize=14, fontsize=12)

# Adjust layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()

<Figure size 3000x1000 with 0 Axes>

```



```

def plot_churn_distribution(df, top=True, n=5):
    """
    Plot churn distribution for the top or bottom states.

    Parameters:
        df (DataFrame): DataFrame containing 'state' and 'churn'
        columns.
        top (bool): If True, plot churn distribution for top states.
        If False, plot for bottom states. Default is True.
        n (int): Number of states to consider. Default is 5.

    Returns:
        None (plots churn distribution)
    """
    # Group by "state" and "churn" to calculate counts

```

```

state_churn_counts = df.groupby(["state",
"churn"]).size().unstack()

# Calculate total churn count for each state
state_churn_counts['Total Churn Count'] =
state_churn_counts.sum(axis=1)

# Determine whether to select top or bottom states
if top:
    states = state_churn_counts['Total Churn
Count'].nlargest(n).index
else:
    states = state_churn_counts['Total Churn
Count'].nsmallest(n).index

# Select churn counts for the selected states
churn_counts = state_churn_counts.loc[states]

# Plot the churn distribution
plt.figure(figsize=(10, 6))
churn_counts.plot(kind='bar', stacked=False, figsize=(12, 8),
width=0.8)

# Add a title and labels
if top:
    title = f'Churn Distribution for Top {n} States'
else:
    title = f'Churn Distribution for Bottom {n} States'
plt.title(title, fontsize=18, fontweight="bold", pad=15)
plt.xlabel('State', fontsize=15, fontweight="bold", labelpad=15)
plt.ylabel('Churn Count', fontsize=15, fontweight="bold",
labelpad=15)

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

# Add a legend
plt.legend(title='Churn', title_fontsize=14, fontsize=12)

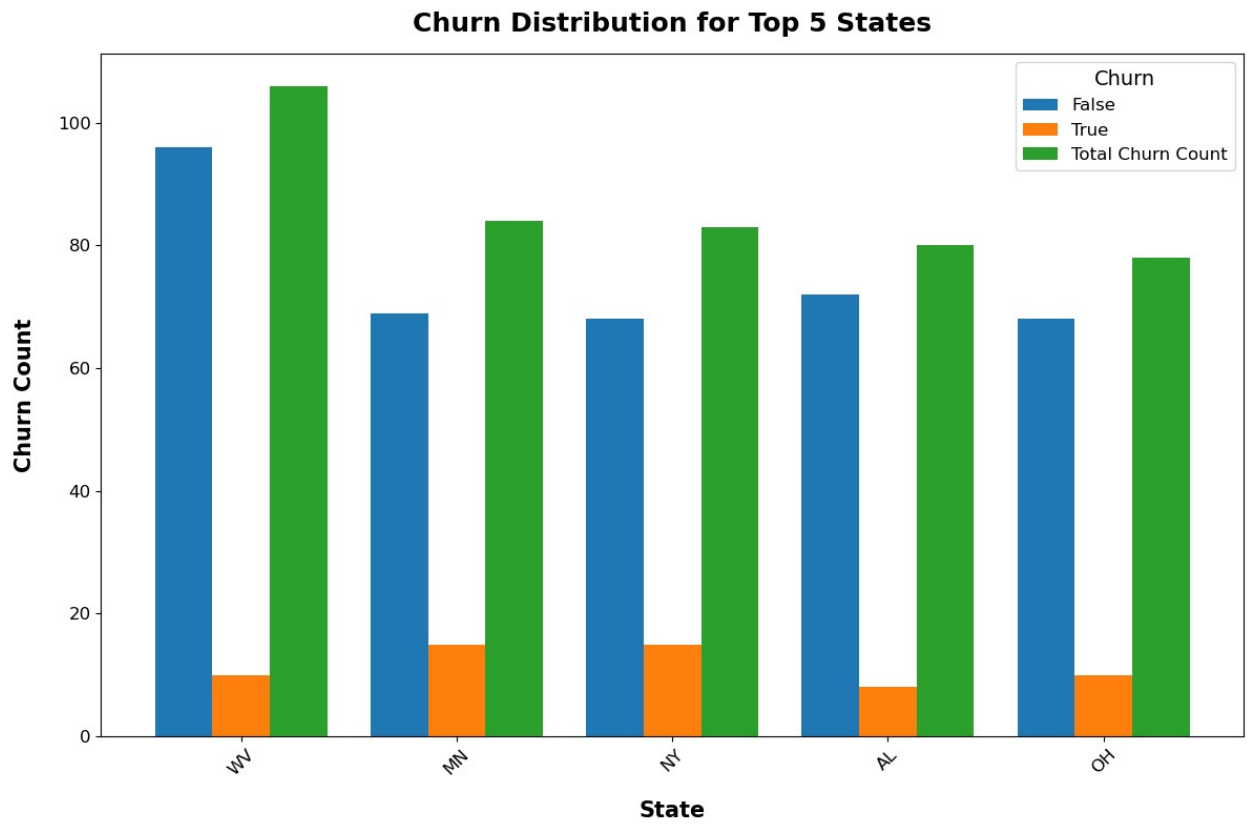
# Adjust layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()

plot_churn_distribution(df, top=True, n=5) # Plot churn distribution
for top 5 states

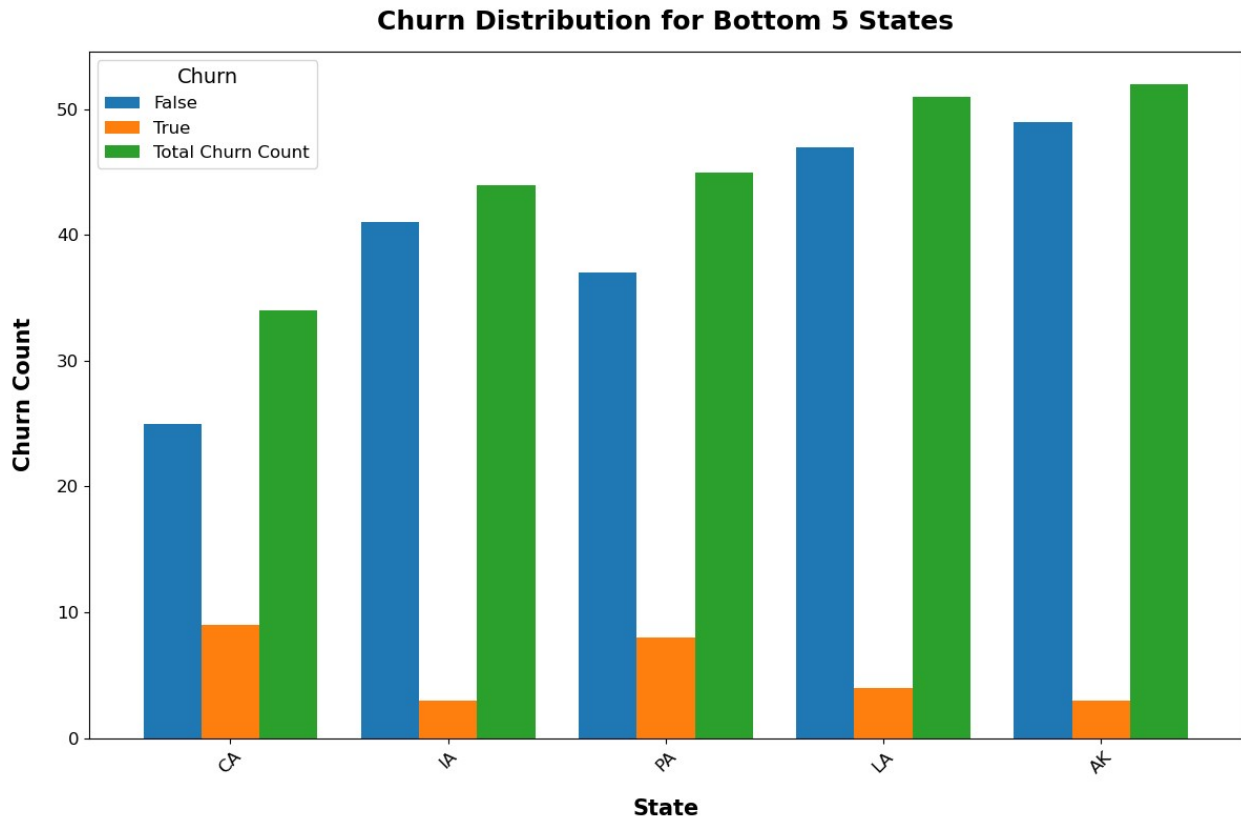
<Figure size 1000x600 with 0 Axes>

```



```
plot_churn_distribution(df, top=False, n=5) # Plot churn distribution  
for bottom 5 states
```

<Figure size 1200x800 with 0 Axes>



## Churn by Categorical Features

```
# Set up the figure and axes for subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(24, 8))

# Define colors for churn
churn_colors = ['#1f77b4', '#ff7f0e']

# Group by "area code" and "churn", then unstack and plot
df.groupby(["area code", "churn"]).size().unstack().plot(kind='bar',
stacked=False, ax=axs[0], color=churn_colors)
axs[0].set_title('Churn by Area Code', fontsize=18, fontweight="bold",
pad=15)
axs[0].set_xlabel('Area Code', fontsize=18, fontweight="bold",
labelpad=15)
axs[0].set_ylabel('Count', fontsize=18, fontweight="bold",
labelpad=15)
axs[0].tick_params(axis='x', rotation=45)
axs[0].legend(title='Churn', fontsize=14)

# Group by "voice mail plan" and "churn", then unstack and plot
df.groupby(["voice mail plan",
"churn"]).size().unstack().plot(kind='bar', stacked=False, ax=axs[1],
color=churn_colors)
axs[1].set_title('Churn by Voice Mail Plan', fontsize=18,
```

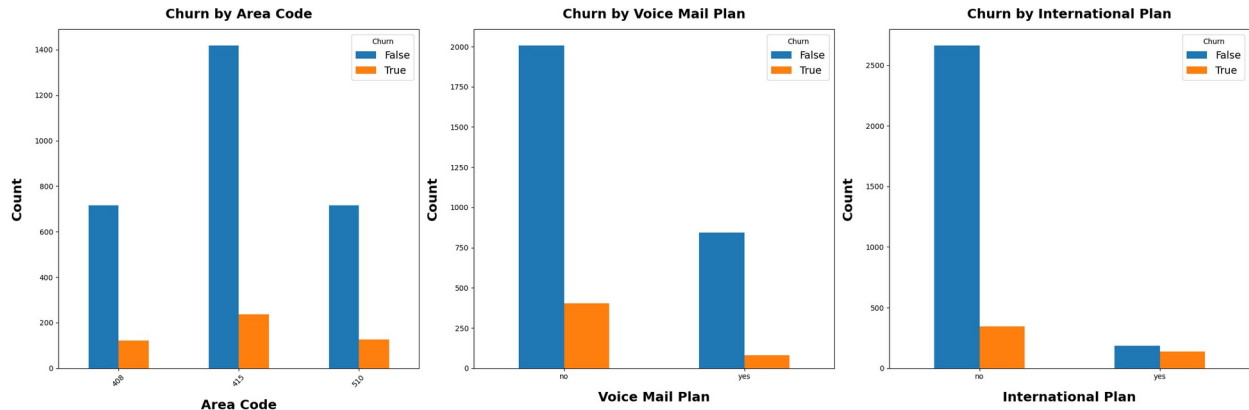
```

fontweight="bold", pad=15)
axs[1].set_xlabel('Voice Mail Plan', fontsize=18, fontweight="bold",
labelpad=15)
axs[1].set_ylabel('Count', fontsize=18, fontweight="bold",
labelpad=15)
axs[1].tick_params(axis='x', rotation=0)
axs[1].legend(title='Churn', fontsize=14)

# Group by "international plan" and "churn", then unstack and plot
df.groupby(["international plan",
"churn"]).size().unstack().plot(kind='bar', stacked=False, ax=axs[2],
color=churn_colors)
axs[2].set_title('Churn by International Plan', fontsize=18,
fontweight="bold", pad=15)
axs[2].set_xlabel('International Plan', fontsize=18,
fontweight="bold", labelpad=15)
axs[2].set_ylabel('Count', fontsize=18, fontweight="bold",
labelpad=15)
axs[2].tick_params(axis='x', rotation=0)
axs[2].legend(title='Churn', fontsize=14)

# Adjust the layout and spacing
plt.tight_layout()
plt.show()

```



- **Churn by Area Code:** The analysis reveals significant variations in churn rates across different area codes. Area code 415 exhibits the highest churn rate, whereas area code 408 shows the lowest churn rate. This indicates that customer retention strategies may need to be tailored based on geographic locations.
- **Churn by Voice Mail Plan:** Customers without a voice mail plan demonstrate a markedly higher churn rate compared to those with a voice mail plan. This suggests that having a voice mail plan may be associated with increased customer loyalty.

- **Churn by International Plan:** Similarly, customers without an international plan exhibit a higher churn rate compared to those with an international plan. This highlights the potential value of international plans in reducing churn rates.

## Churn by numerical features.

```
# Set up the figure and axes for subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(24, 16))

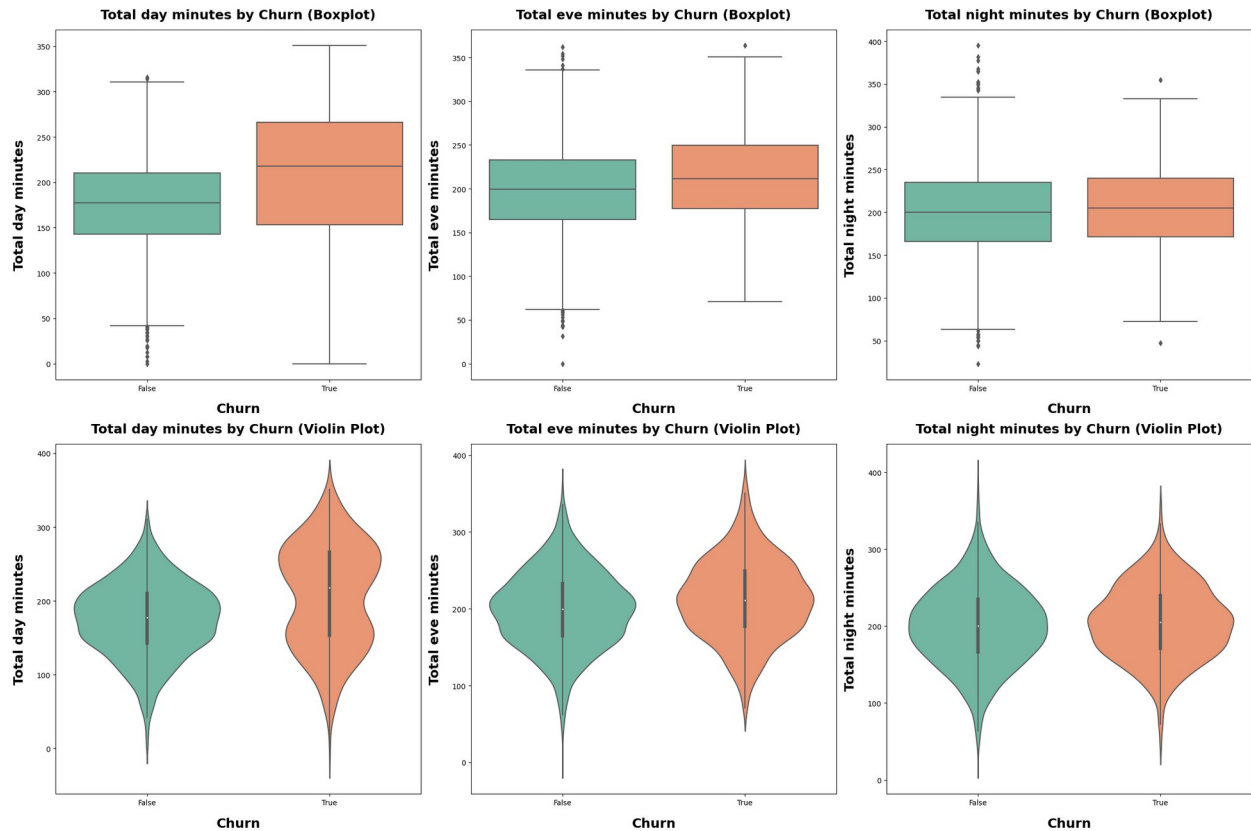
# List of numerical columns
numerical_cols = ["total day minutes", "total eve minutes", "total
night minutes"]

# Define color palette for consistency
palette = "Set2"

# Loop over numerical columns and plot boxplots and violin plots
for i, col in enumerate(numerical_cols):
    sns.boxplot(x="churn", y=col, data=df, ax=axs[0, i],
palette=palette)
    axs[0, i].set_title(f"{col.capitalize()} by Churn (Boxplot)",
fontsize=18, fontweight="bold", pad=15)
    axs[0, i].set_xlabel("Churn", fontsize=18, fontweight="bold",
labelpad=15)
    axs[0, i].set_ylabel(col.replace('_', ' ').capitalize(),
fontsize=18, fontweight="bold", labelpad=15)

    sns.violinplot(x="churn", y=col, data=df, ax=axs[1, i],
palette=palette)
    axs[1, i].set_title(f"{col.capitalize()} by Churn (Violin Plot)",
fontsize=18, fontweight="bold", pad=15)
    axs[1, i].set_xlabel("Churn", fontsize=18, fontweight="bold",
labelpad=15)
    axs[1, i].set_ylabel(col.replace('_', ' ').capitalize(),
fontsize=18, fontweight="bold", labelpad=15)

# Adjust the layout and spacing
plt.tight_layout()
plt.show()
```



From the plots, you can observe the following:

- **Total Day Minutes:** The distribution of total day minutes is noticeably higher for churned customers compared to retained customers. This suggests that customers who spend more time on daytime calls are more likely to churn.
- **Total Evening Minutes:** Similar to total day minutes, the total evening minutes are also higher for churned customers. This indicates a pattern where increased call activity during evening hours is associated with higher churn rates.
- **Total Night Minutes:** While the difference is less pronounced, churned customers still tend to have slightly higher total night minutes than retained customers. This pattern reinforces the trend that higher call activity, regardless of the time of day, may be linked to customer churn.

The violin plots corroborate these findings by showing the density of churned customers is higher at larger values of minutes for each feature. This density distribution provides a visual confirmation that higher call usage is a potential indicator of customer churn across different times of the day.

### 3. Data Preparation

In the data preparation stage, we focus on ensuring the data is suitable for modeling by addressing various data quality and preprocessing tasks:

- **Handling Missing Values:** Detect and assess missing values, then decide on a strategy (imputation, removal, or marking as a separate category).
- **Data Type Conversions:** Ensure correct data types for modeling (e.g., numeric data erroneously encoded as strings).
- **Handling Duplicates:** Check for and remove duplicates.
- **Addressing Multicollinearity:** Identify and mitigate multicollinearity (high correlation between predictors) to enhance model interpretability, using techniques like correlation analysis or PCA.
- **Converting Categorical Data:** Convert categorical variables (e.g., "International Plan" and "Voice Mail Plan") to numeric format via one-hot encoding, as most ML algorithms require numeric input.

### 3.1 Handling Missing Values:

```
# 1. Handling Missing Values:
df.isna().sum()
```

```
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
```

```
df.sample(20)
```

	state	account length	area code	phone number	international plan
829	ME	120	510	350-5883	no
48	ID	119	415	398-1294	no



2149	IA	120	415	341-6743	no
1265	WY	95	415	340-4236	no
1234	IA	86	408	390-3873	no
1233	IL	48	510	380-5246	no
2601	AL	85	408	386-6411	no
2886	PA	85	408	405-9573	no
2526	NJ	57	510	330-2635	yes
621	DC	126	510	362-8280	no
1109	ID	118	415	335-3320	no
2132	GA	202	510	351-2589	no
324	VA	129	408	384-2632	no
2111	NC	126	415	342-1702	no
2674	IL	67	415	369-4377	no
2901	MT	85	408	372-4868	no
3291	MI	119	510	335-7324	yes
1288	MA	56	510	401-3622	no
2591	ND	122	408	395-1901	no
2155	AL	172	408	359-5731	no
	voice mail plan	number vmail messages	total day minutes	\	
829	no	0	198.8		
48	no	0	159.1		
2149	yes	33	299.5		
1265	yes	39	260.8		
1234	no	0	126.3		
1233	no	0	128.2		
2601	yes	30	173.1		
2886	no	0	144.4		
2526	no	0	115.0		
621	no	0	122.4		
1109	no	0	140.4		
2132	no	0	115.4		

324	no	0	207.0
2111	no	0	103.7
2674	no	0	109.1
2901	yes	17	89.8
3291	yes	22	172.1
1288	no	0	253.2
2591	no	0	231.2
2155	no	0	270.0

	total day calls	total day charge	total eve minutes	total eve
calls \				
829	56	33.80	230.1	
73				
48	114	27.05	231.3	
117				
2149	83	50.92	163.4	
84				
1265	130	44.34	213.4	
111				
1234	115	21.47	168.8	
112				
1233	71	21.79	48.1	
78				
2601	107	29.43	247.2	
101				
2886	88	24.55	264.6	
105				
2526	65	19.55	122.3	
96				
621	88	20.81	143.8	
111				
1109	112	23.87	187.1	
60				
2132	137	19.62	178.7	
70				
324	91	35.19	154.9	
121				
2111	93	17.63	127.0	
107				
2674	117	18.55	217.4	
124				
2901	88	15.27	233.2	
75				
3291	119	29.26	223.6	
133				
1288	95	43.04	188.0	
116				
2591	141	39.30	267.8	
136				

2155	102	45.90	256.6	
111				
	total eve charge	total night minutes	total night calls	\
829	19.56	119.8	81	
48	19.66	143.2	91	
2149	13.89	146.7	88	
1265	18.14	195.6	97	
1234	14.35	154.6	95	
1233	4.09	116.3	80	
2601	21.01	158.7	104	
2886	22.49	185.4	94	
2526	10.40	245.0	75	
621	12.22	157.0	106	
1109	15.90	207.9	155	
2132	15.19	185.7	113	
324	13.17	245.1	112	
2111	10.80	329.3	66	
2674	18.48	188.4	141	
2901	19.82	165.7	116	
3291	19.01	150.0	94	
1288	15.98	142.0	133	
2591	22.76	240.3	100	
2155	21.81	168.5	104	
	total night charge	total intl minutes	total intl calls	\
829	5.39	9.9	3	
48	6.44	8.8	3	
2149	6.60	11.6	5	
1265	8.80	10.1	5	
1234	6.96	9.8	7	
1233	5.23	8.9	3	
2601	7.14	11.5	5	
2886	8.34	9.9	3	
2526	11.03	6.4	1	
621	7.07	11.5	3	
1109	9.36	7.9	1	
2132	8.36	6.0	3	
324	11.03	13.4	5	
2111	14.82	14.4	1	
2674	8.48	12.8	6	
2901	7.46	9.3	7	
3291	6.75	13.9	20	
1288	6.39	4.4	4	
2591	10.81	8.8	5	
2155	7.58	12.0	5	
	total intl charge	customer service calls	churn	
829	2.67	2	False	
48	2.38	5	True	

2149	3.13	0	False
1265	2.73	1	False
1234	2.65	2	False
1233	2.40	0	False
2601	3.11	1	False
2886	2.67	1	False
2526	1.73	0	True
621	3.11	1	False
1109	2.13	0	False
2132	1.62	3	False
324	3.62	3	False
2111	3.89	0	False
2674	3.46	0	False
2901	2.51	4	True
3291	3.75	1	True
1288	1.19	1	False
2591	2.38	1	True
2155	3.24	2	True

```
df.describe()
```

	account length	area code	number vmail messages	total day
minutes \				
count	3333.000000	3333.000000		3333.000000
3333.000000				
mean	101.064806	437.182418		8.099010
179.775098				
std	39.822106	42.371290		13.688365
54.467389				
min	1.000000	408.000000		0.000000
0.000000				
25%	74.000000	408.000000		0.000000
143.700000				
50%	101.000000	415.000000		0.000000
179.400000				
75%	127.000000	510.000000		20.000000
216.400000				
max	243.000000	510.000000		51.000000
350.800000				

	total day calls	total day charge	total eve minutes	total eve
calls \				
count	3333.000000	3333.000000		3333.000000
3333.000000				
mean	100.435644	30.562307		200.980348
100.114311				
std	20.069084	9.259435		50.713844
19.922625				
min	0.000000	0.000000		0.000000
0.000000				

25%	87.000000	24.430000	166.600000
87.000000			
50%	101.000000	30.500000	201.400000
100.000000			
75%	114.000000	36.790000	235.300000
114.000000			
max	165.000000	59.640000	363.700000
170.000000			

	total eve charge	total night minutes	total night calls	\
count	3333.000000	3333.000000	3333.000000	
mean	17.083540	200.872037	100.107711	
std	4.310668	50.573847	19.568609	
min	0.000000	23.200000	33.000000	
25%	14.160000	167.000000	87.000000	
50%	17.120000	201.200000	100.000000	
75%	20.000000	235.300000	113.000000	
max	30.910000	395.000000	175.000000	

	total night charge	total intl minutes	total intl calls	\
count	3333.000000	3333.000000	3333.000000	
mean	9.039325	10.237294	4.479448	
std	2.275873	2.791840	2.461214	
min	1.040000	0.000000	0.000000	
25%	7.520000	8.500000	3.000000	
50%	9.050000	10.300000	4.000000	
75%	10.590000	12.100000	6.000000	
max	17.770000	20.000000	20.000000	

	total intl charge	customer service calls
count	3333.000000	3333.000000
mean	2.764581	1.562856
std	0.753773	1.315491
min	0.000000	0.000000
25%	2.300000	1.000000
50%	2.780000	1.000000
75%	3.270000	2.000000
max	5.400000	9.000000

After meticulous examination, it is evident that the dataset does not contain any missing values. However, it is notable that there are instances of zero values present. These zero values may represent genuine data entries rather than missing values. Further clarification from the data source, such as the company, may be required to ensure accurate interpretation and handling of these zero values in the analysis, but for now will assume that they are valid entries.

## 3.2 Data Type Conversions:

Let's first re-assess the data-types in our data-set:

```
# Data types in the dataset
df.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
```

Converting the "area code" column to a categorical variable improves its suitability for grouping and identification purposes. By treating it as a category rather than a numerical value, we enhance clarity and prevent accidental numerical interpretations or computations. This conversion ensures that the focus remains on the distinct categories represented by the area codes, facilitating clearer communication and analysis, particularly in tasks related to regional or locational segmentation.

```
# So let's convert area code data into strings:
df['area code'] = df['area code'].astype(str)

df.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   object
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
```

0	state	3333	non-null	object
1	account length	3333	non-null	int64
2	area code	3333	non-null	object
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(7), object(5)  
memory usage: 524.2+ KB

Based on my analysis, I can confirm that all the other fields in our dataset have the relevant data types. They are well-aligned with the information they represent and are suitable for our analysis.

### 3.3 Handling Duplicates:

```
# Handling Duplicates:
duplicate_rows = df[df.duplicated()]
num_duplicate_rows = len(duplicate_rows)

print(f"Number of duplicate rows: {num_duplicate_rows}")
```

Number of duplicate rows: 0

We see from the above output that there are no duplicates in this dataset.

### 3.4 Checking for and Removing Multicollinearity (Correlated Predictors)

In classification problems like this, multicollinearity (high correlation between predictor variables) is a concern. While multicollinearity doesn't directly impact the accuracy or performance of classification models as it does for regression models, it still poses challenges for model interpretability and coefficient stability.

The heatmap we observed earlier, revealed highly correlated variables. Now, let's filter out these correlations and delve deeper into their relationships.

Setting the correlation threshold at 0.7 designates a "high" correlation due to its indication of a strong linear relationship between variables, helping identify pairs that may lead to multicollinearity issues in subsequent analyses. This threshold is commonly used to strike a balance between capturing significant correlations and avoiding redundancy, ensuring model stability and interpretability.

```
numeric_df = df.select_dtypes(include=['int64', 'float64'])

correlation_matrix = numeric_df.corr()
correlation_threshold = 0.7 # Threshold for correlation

highly_correlated_pairs = []

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        # Get the correlation coefficient
        correlation = correlation_matrix.iloc[i, j]

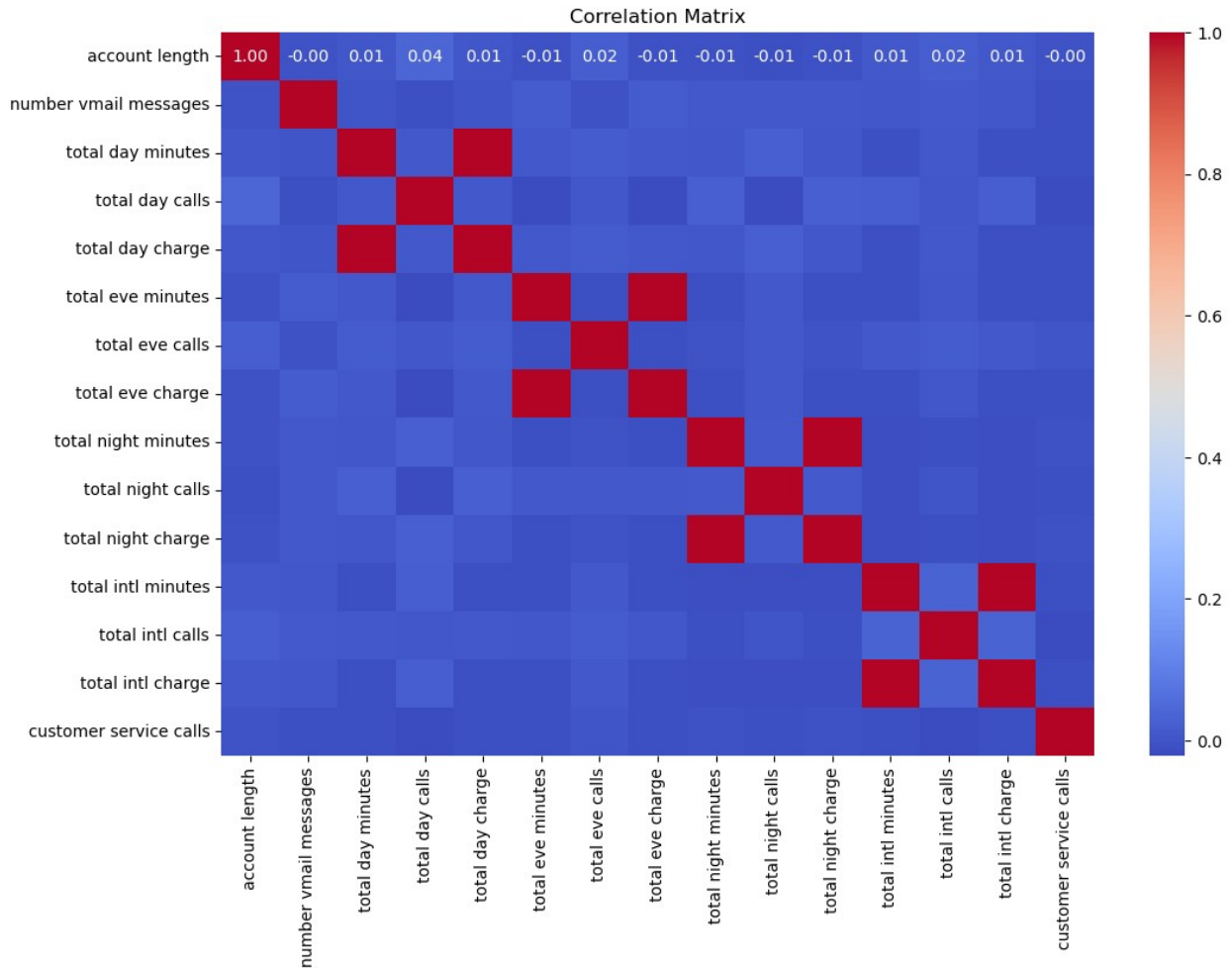
        # Check if the correlation is above the threshold
        if abs(correlation) > correlation_threshold:
            # Get the variable names
            variable1 = correlation_matrix.columns[i]
            variable2 = correlation_matrix.columns[j]

            # Append the pair and correlation to the list
            highly_correlated_pairs.append((variable1, variable2,
correlation))

# Visualize correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# Print highly correlated pairs
if highly_correlated_pairs:
    print("Highly correlated pairs:")
    for pair in highly_correlated_pairs:
        print(pair)
else:
    print("No highly correlated pairs found.")
```





Highly correlated pairs:

```
('total day charge', 'total day minutes', 0.999999521904007)
('total eve charge', 'total eve minutes', 0.999997760198491)
('total night charge', 'total night minutes', 0.999992148758795)
('total intl charge', 'total intl minutes', 0.999927417510314)
```

## Rationale for Dropping Highly Correlated Features:

- Near-Perfect Correlation:**
  - Highly correlated pairs exhibit correlation coefficients very close to 1. For instance, the pair `total day charge` and `total day minutes` demonstrates an exceptionally high correlation coefficient (0.999999521904007).
- Redundancy Indication:**
  - High correlation between features implies redundancy in the dataset. Features like `total day charge` and `total day minutes` are essentially duplicating information, offering little additional insight.
- Risk of Overfitting:**

- In the context of predictive modeling, the inclusion of highly correlated features can exacerbate overfitting. Co-occurring features, such as `total day charge` and `total day minutes`, may disproportionately influence the model, leading to suboptimal generalization.
4. **Enhanced Computational Efficiency:**
    - Highly correlated features can significantly increase computational overhead without providing commensurate gains in predictive performance. Eliminating one of these features streamlines model training and inference processes.
  5. **Improved Model Interpretability:**
    - By removing highly correlated features, we enhance the interpretability of our model. Simplifying the feature space, as exemplified by the removal of redundant features like `total day charge` or `total day minutes`, facilitates a clearer understanding of the model's decision-making process.

## Conclusion:

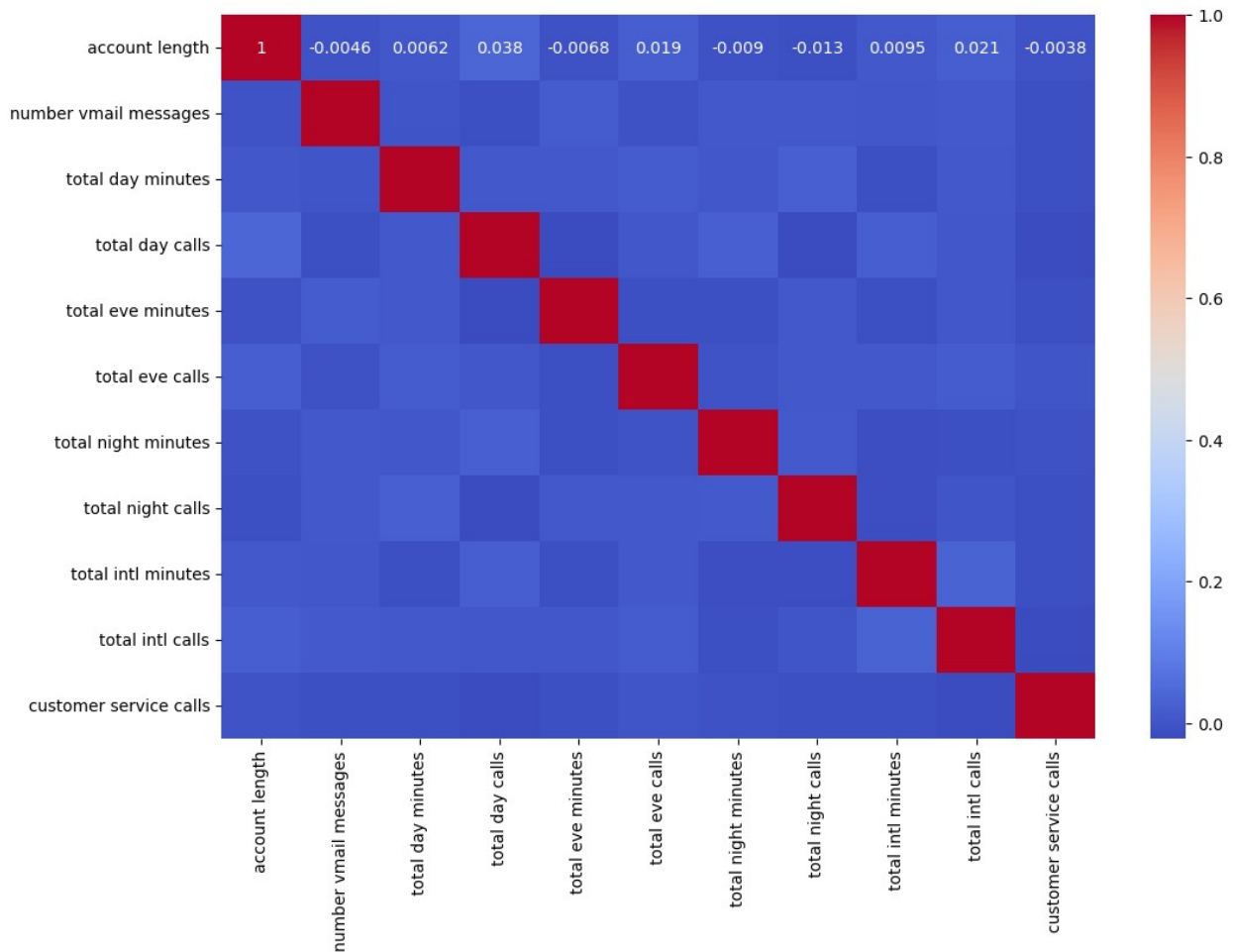
- Given the high correlation observed between features such as `total day charge` and `total day minutes`, it is prudent to eliminate one of the correlated features. This strategy mitigates overfitting risks, improves computational efficiency, and enhances the interpretability of the resulting model.
1. **Charge and Minutes:**
    - Each highly correlated pair consists of one feature related to call charges (e.g., `total day charge`, `total eve charge`, `total night charge`, `total intl charge`) and another feature related to the corresponding call duration in minutes (e.g., `total day minutes`, `total eve minutes`, `total night minutes`, `total intl minutes`).
    - This commonality suggests a strong linear relationship between call charges and call duration for different time periods: day, evening, night, and international calls.
    - The high correlation coefficients (close to 1) indicate that as call duration increases, call charges also increase proportionally, which is a logical expectation in most telecommunications pricing models.

```
# Drop the specified columns related to charges
df = df.drop(columns=['total day charge', 'total eve charge', 'total
night charge', 'total intl charge'])

# Drop non-numeric columns
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Confirm if multicollinearity is gone
correlation_matrix = numeric_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.show()
```



```
highly_correlated_pairs = []

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        # Get the correlation coefficient
        correlation = correlation_matrix.iloc[i, j]

        # Check if the correlation is above the threshold
        if abs(correlation) > correlation_threshold:
            # Get the variable names
            variable1 = correlation_matrix.columns[i]
            variable2 = correlation_matrix.columns[j]

            # Append the pair and correlation to the list
            highly_correlated_pairs.append((variable1, variable2,
correlation))

# Output the highly correlated pairs if any, else indicate no such
pairs found
if highly_correlated_pairs:
```

```
for pair in highly_correlated_pairs:
    print(f"Highly correlated pair: {pair[0]} and {pair[1]} with
correlation {pair[2]:.2f}")
else:
    print("No more columns with a correlation of more than 0.7")
```

No more columns with a correlation of more than 0.7

## 3.5 Rationale for Dropping the Phone Number Column

In the process of converting categorical data to numeric format through one-hot encoding, we encountered a challenge with the 'phone number' column. While analyzing the categorical variables in our dataset, we identified five variables (excluding the target variable) along with their corresponding value counts:

Column	Value Counts
'state'	51
'phone number'	3333
'international plan'	2
'voice mail plan'	2
'area code'	3

However, the 'phone number' column poses a unique issue. Each record in this column has a distinct value, resulting in 3333 unique entries. Encoding this column would introduce 3333 new features, significantly increasing dimensionality and potentially introducing noise into our dataset.

To address this challenge, we propose dropping the 'phone number' column for the following reasons:

1. **High Cardinality:** The 'phone number' column exhibits a high cardinality due to its large number of unique values (3333). Handling categorical features with high cardinality can lead to computational and memory challenges.
2. **Limited Predictive Power:** The 'phone number' is unlikely to contribute meaningful predictive information for identifying customer churn. Most machine learning algorithms may struggle to discern patterns from this column, as it primarily serves as an identifier rather than a predictive feature.
3. **Curse of Dimensionality:** Including a high-cardinality categorical variable like 'phone number' can exacerbate the curse of dimensionality, resulting in a sparse dataset and potentially degrading model performance due to increased complexity.
4. **Data Privacy Concerns:** Depending on the context, the 'phone number' column may contain sensitive or personally identifiable information (PII). Handling such data requires careful consideration of legal and privacy implications, which may not align with the objectives of our analysis.

5. **Practicality and Relevance:** From a practical standpoint, the 'phone number' may not offer substantial analytical insights or contribute meaningfully to our modeling efforts. Its inclusion may not align with the objectives of our analysis or the insights we seek to derive from the dataset.

Given these considerations, dropping the 'phone number' column is a prudent step to streamline our dataset and focus on relevant features for predicting customer churn.

```
# dropping phone number
```

```
df = df.drop(columns=['phone number'])
```

```
# confirm if it has been successfully dropped
```

```
print((df.shape))
```

```
df.head()
```

```
(3333, 16)
```

	state	account length	area code	international plan	voice mail
0	KS	128	415	no	yes
1	OH	107	415	no	yes
2	NJ	137	415	no	no
3	OH	84	408	yes	no
4	OK	75	415	yes	no

	number vmail messages	total day minutes	total day calls
0	25	265.1	110
1	26	161.6	123
2	0	243.4	114
3	0	299.4	71
4	0	166.7	113

	total eve minutes	total eve calls	total night minutes	total night calls
0	197.4	99	244.7	91
1	195.5	103	254.4	103
2	121.2	110	162.6	104
3	61.9	88	196.9	89
4	148.3	122	186.9	121

	total intl minutes	total intl calls	customer service calls	churn
--	--------------------	------------------	------------------------	-------

0	10.0	3	1	False
1	13.7	3	1	False
2	12.2	5	0	False
3	6.6	7	2	False
4	10.1	3	3	False

With the dataset prepared, our next step is to filter out the categorical variables for encoding. This process involves identifying and isolating the categorical features to apply encoding techniques. Once encoded, we will concatenate these categorical variables with the numerical ones, preparing the dataset comprehensively for the modeling phase.

```
# These are the updated numerical and categorical columns
categorical_vars = list(df.select_dtypes(include=['object']).columns )
numerical_vars = list(df.select_dtypes(include=['int64',
'float64']).columns )
categorical_vars, numerical_vars

(['state', 'area code', 'international plan', 'voice mail plan'],
 ['account length',
 'number vmail messages',
 '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'])
```

The subsequent code performs encoding on the categorical variables, employing a strategy that drops the first category for each variable to mitigate potential multicollinearity issues stemming from the encoded variables.

```
# one hot encoding for categorical data (only for the predictors)
df = pd.get_dummies(df, columns=categorical_vars, drop_first=True)
df.head()
```

	account length	number vmail messages	total day minutes	total day calls
0	128	25	265.1	
1	107	26	161.6	

123						
2	137	0	243.4			
114						
3	84	0	299.4			
71						
4	75	0	166.7			
113						
	total eve minutes	total eve calls	total night minutes	total		
night calls \						
0	197.4	99	244.7			
91						
1	195.5	103	254.4			
103						
2	121.2	110	162.6			
104						
3	61.9	88	196.9			
89						
4	148.3	122	186.9			
121						
	total intl minutes	total intl calls	customer service calls	churn		
\						
0	10.0	3	1	False		
1	13.7	3	1	False		
2	12.2	5	0	False		
3	6.6	7	2	False		
4	10.1	3	3	False		
	state_AL	state_AR	state_AZ	state_CA	state_CO	state_CT
state_DC \						
0	False	False	False	False	False	False
False						
1	False	False	False	False	False	False
False						
2	False	False	False	False	False	False
False						
3	False	False	False	False	False	False
False						
4	False	False	False	False	False	False
False						
	state_DE	state_FL	state_GA	state_HI	state_IA	state_ID
state_IL \						
0	False	False	False	False	False	False

```

False
1   False      False      False      False      False      False      False
False
2   False      False      False      False      False      False      False
False
3   False      False      False      False      False      False      False
False
4   False      False      False      False      False      False      False
False

```

```

      state_IN  state_KS  state_KY  state_LA  state_MA  state_MD
state_ME \
0   False      True      False      False      False      False
False
1   False      False      False      False      False      False
False
2   False      False      False      False      False      False
False
3   False      False      False      False      False      False
False
4   False      False      False      False      False      False
False

```

```

      state_MI  state_MN  state_MO  state_MS  state_MT  state_NC
state_ND \
0   False      False      False      False      False      False
False
1   False      False      False      False      False      False
False
2   False      False      False      False      False      False
False
3   False      False      False      False      False      False
False
4   False      False      False      False      False      False
False

```

```

      state_NE  state_NH  state_NJ  state_NM  state_NV  state_NY
state_OH \
0   False      False      False      False      False      False
False
1   False      False      False      False      False      False
True
2   False      False      True       False      False      False
False
3   False      False      False      False      False      False
True
4   False      False      False      False      False      False
False

```

```

      state_OK  state_OR  state_PA  state_RI  state_SC  state_SD

```



state_TN \						
0	False	False	False	False	False	False
False						
1	False	False	False	False	False	False
False						
2	False	False	False	False	False	False
False						
3	False	False	False	False	False	False
False						
4	True	False	False	False	False	False
False						

	state_TX	state_UT	state_VA	state_VT	state_WA	state_WI
state_WV \						
0	False	False	False	False	False	False
False						
1	False	False	False	False	False	False
False						
2	False	False	False	False	False	False
False						
3	False	False	False	False	False	False
False						
4	False	False	False	False	False	False
False						

	state_WY	area code_415	area code_510	international plan_yes \
0	False	True	False	False
1	False	True	False	False
2	False	True	False	False
3	False	False	False	True
4	False	True	False	True

	voice mail plan_yes
0	True
1	True
2	False
3	False
4	False

df.columns

```
Index(['account length', 'number vmail messages', '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', 'churn',
      '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', 'area code_415', 'area code_510', 'international
plan_yes',
    'voice mail plan_yes'],
    dtype='object')
```

## 4. Predictive Modelling

Let's initiate the iterative modeling process to uncover patterns within our dataset that could assist in predicting customer churn. We'll start by importing all the essential libraries needed for this task.

```
# imports necessary for modelling
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, KFold,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, \
roc_curve, auc, ConfusionMatrixDisplay, accuracy_score

from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

Identifying the 'target' variable and its associated 'predictors' from the dataset.

```
# target
y = df['churn']

# predictors
X = df.drop(columns=['churn'])
```

As observed from the categorical variable countplots during our exploratory data analysis (EDA), there is a class imbalance, as illustrated below.

```
# label/target value counts
df['churn'].value_counts()

churn
False    2850
True      483
Name: count, dtype: int64
```

As observed from the categorical variable countplots during our exploratory data analysis (EDA), there is a class imbalance, as illustrated in the EDA section.

This indicates that we need to resample our data before training our models to prevent instabilities caused by imbalanced classes. To achieve this, we will use **Synthetic Minority Oversampling Technique (SMOTE)**.

## i. Split data-set

As a first step, we split our dataset into training and testing data to prevent information leakage across the entire dataset, which can occur during scaling. We'll split it into an 80/20 ratio with a random state seed of 42.

```
# splitting the data into training and testing

# test size (20%)
test_size = 0.2
# random state seed
SEED = 42

# splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=SEED)
```

## ii. Deal with class imbalance:

Now we deal with the class imbalance:

```
# Synthetic Minority Oversampling
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

Checking the new resampled target variable:

```
# data balanced and ready for modelling:
y_train_resampled.value_counts()
```

```
churn
False    2284
True     2284
Name: churn, dtype: int64
```

iii. Pick a model

## 4.1 Baseline model (Dummy)

Before we actually perform any modeling, let's determine what metrics we would expect to get with a "dummy" model that always predicts the positive class.

```
# Instantiate and fit the Dummy Classifier
dummy_model = DummyClassifier(strategy='constant', constant=1)
dummy_model.fit(X_train_resampled, y_train_resampled)

# Create a ConfusionMatrixDisplay object from the estimator
cm_display =
ConfusionMatrixDisplay.from_estimator(estimator=dummy_model, X=X_test,
y=y_test,

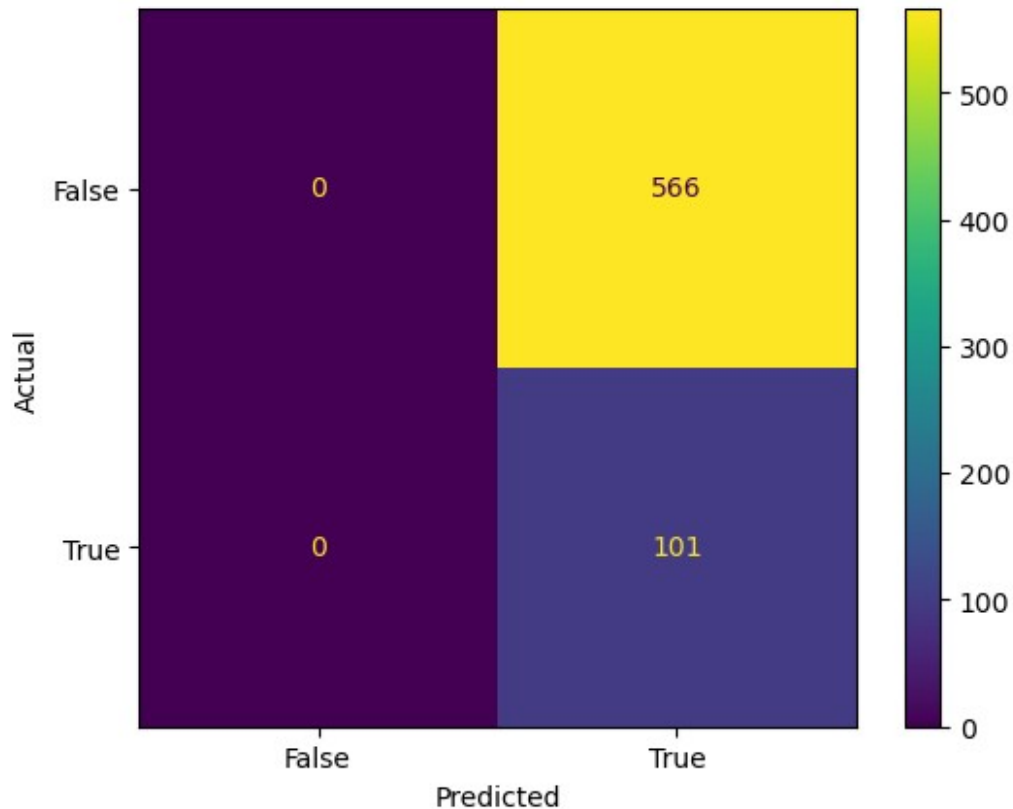
display_labels=["False", "True"])

# Set x and y axis labels for the confusion matrix display
cm_display.ax_.set_xlabel('Predicted')
cm_display.ax_.set_ylabel('Actual')

# Make predictions
y_pred = dummy_model.predict(X_test)

# Generate the classification report
report = classification_report(y_test, y_pred)
print(report)
```

	precision	recall	f1-score	support
False	0.00	0.00	0.00	566
True	0.15	1.00	0.26	101
accuracy			0.15	667
macro avg	0.08	0.50	0.13	667
weighted avg	0.02	0.15	0.04	667



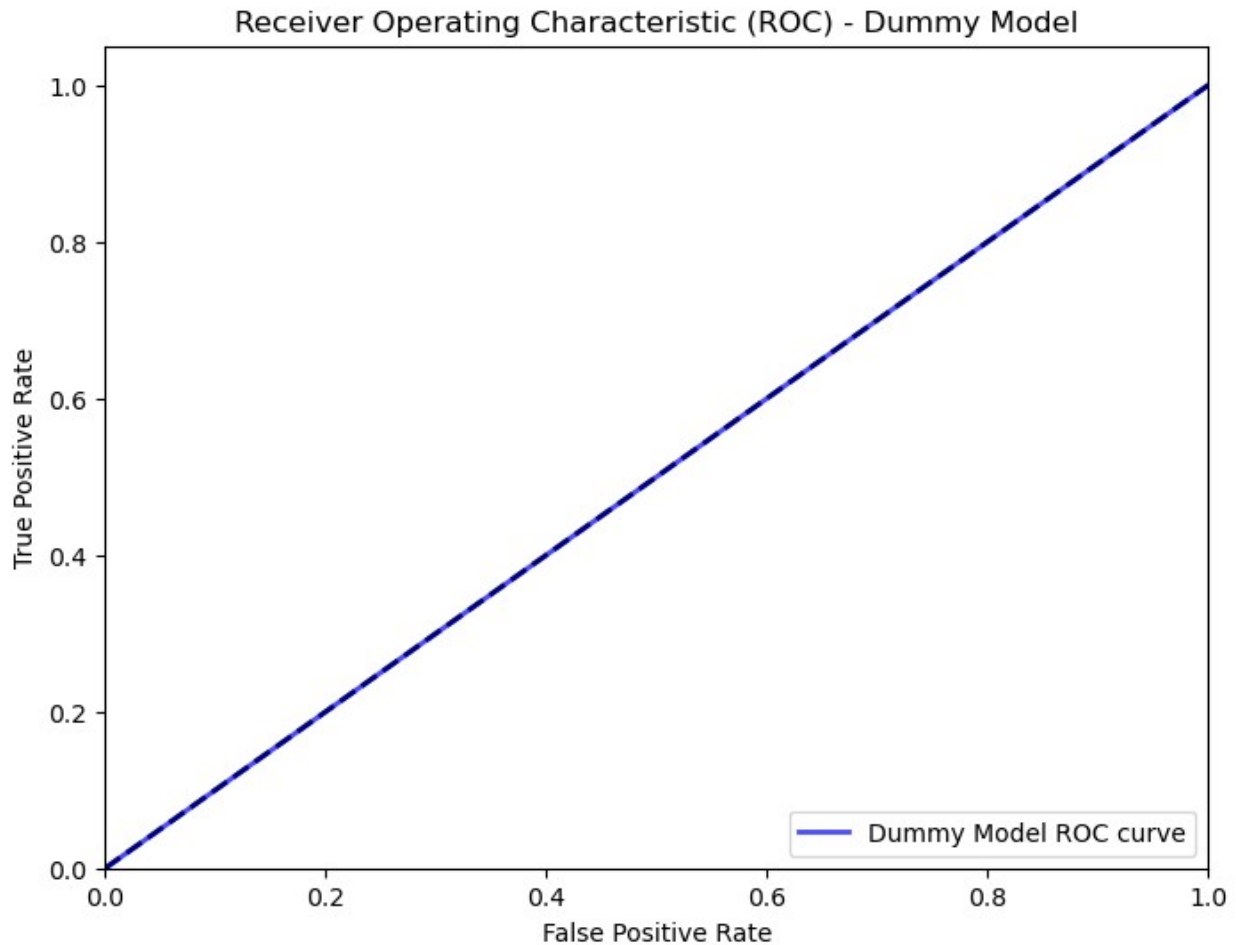
```
# Generate predicted probabilities for the positive class
y_scores = dummy_model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
dummy_fpr, dummy_tpr, thresholds = roc_curve(y_test, y_scores)

# Calculate the AUC
auc = roc_auc_score(y_test, y_scores)
print(f'Dummy Model AUC: {auc} \n')

# Plot the ROC for the dummy model
plt.figure(figsize=(8, 6))
plt.plot(dummy_fpr, dummy_tpr, color='blue',
         lw=2, label='Dummy Model ROC curve', alpha=.7)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Dummy Model')
plt.legend(loc='lower right')
plt.show()

Dummy Model AUC: 0.5
```



### Interpretation of the baseline metrics:

- **For the "False" class (churn is False):**
  - Precision is 0.00, indicating that none of the predicted "False" instances were correct.
  - Recall is 0.00, suggesting that none of the actual "False" instances were correctly classified.
  - The F1-score is 0.00, reflecting the absence of balance between precision and recall.
  - There are 566 instances of the "False" class in the test data.
- **For the "True" class (churn is True):**
  - Precision is 0.15, indicating that only 15% of the predicted "True" instances were correct.
  - Recall is 1.00, meaning that all actual "True" instances were correctly classified.
  - The F1-score is 0.26, showing some balance between precision and recall.
  - There are 101 instances of the "True" class in the test data.

- An AUC of 0.5 suggests that the model's performance is equivalent to random guessing. Essentially, the dummy model is incapable of making meaningful predictions, and its ROC curve is a diagonal line.
- The macro-average and weighted-average metrics reflect the imbalance in the dataset, with overall poor model performance.

This baseline model serves as a reference point for comparison with more sophisticated models. Its low performance underscores the necessity for more advanced modeling techniques to enhance classification results.

## 4.2 Logistic Regression Model

Next, we will proceed with fitting a logistic regression model to the data to determine if the overall model performance improves.

```
# Initialize logistic regression model with the defined random seed
logreg = LogisticRegression(random_state=SEED)

# Fit logistic regression model to the resampled training data
model_log = logreg.fit(X_train_resampled, y_train_resampled)

# Display the trained model
model_log

LogisticRegression(random_state=42)
```

Now that the model has been fitted, let's evaluate its performance on the test data using the following metrics:

- **Area Under Curve (AUC)**
- **Accuracy**
- **Precision**
- **Recall**
- **F1 score**

In the context of customer churn prediction, it's crucial to prioritize recall over precision. Here's why:

- **Recall (Sensitivity or True Positive Rate):** Recall measures the model's ability to correctly identify all the customers who actually churned (True Positives) out of the total number of customers who did churn (True Positives + False Negatives). High recall ensures that a large portion of actual churn cases is captured. This is important as missing customers who might churn can result in lost revenue or business opportunities.
- **Precision:** Precision measures the model's ability to make accurate positive predictions, i.e., how many of the customers predicted as churning (True Positives) are actually churning out of the total predicted as churning (True Positives + False

Positives). High precision indicates that when the model predicts churn, it's likely to be correct. However, high precision might come at the cost of lower recall because the model could become overly cautious, leading to many False Negatives.

In summary, it is preferable for our final model to have more false positives than false negatives because a false negative implies a customer leaving/churning without being predicted. Capturing more false negatives than false positives means losing more revenue for SyriaTel. However, our main goal remains to minimize false predictions (both false positives and false negatives) while maximizing true predictions (true positives and true negatives).

**Note:** This is not advocating for deliberately having more false positives. It highlights the precision-recall trade-off, where maximizing one metric may result in a decrease in another. The ultimate aim is to strike a balance between precision and recall while minimizing false predictions.

```
# Generate predictions
y_pred = logreg.predict(X_test)

# Visualize the confusion matrix
cm_display = ConfusionMatrixDisplay.from_estimator(estimator=logreg,
X=X_test, y=y_test,

display_labels=["False", "True"])

# Set x and y axis labels for the confusion matrix display
cm_display.ax_.set_xlabel('Predicted')
cm_display.ax_.set_ylabel('Actual')

# Display classification report
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
# Display confusion matrix
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

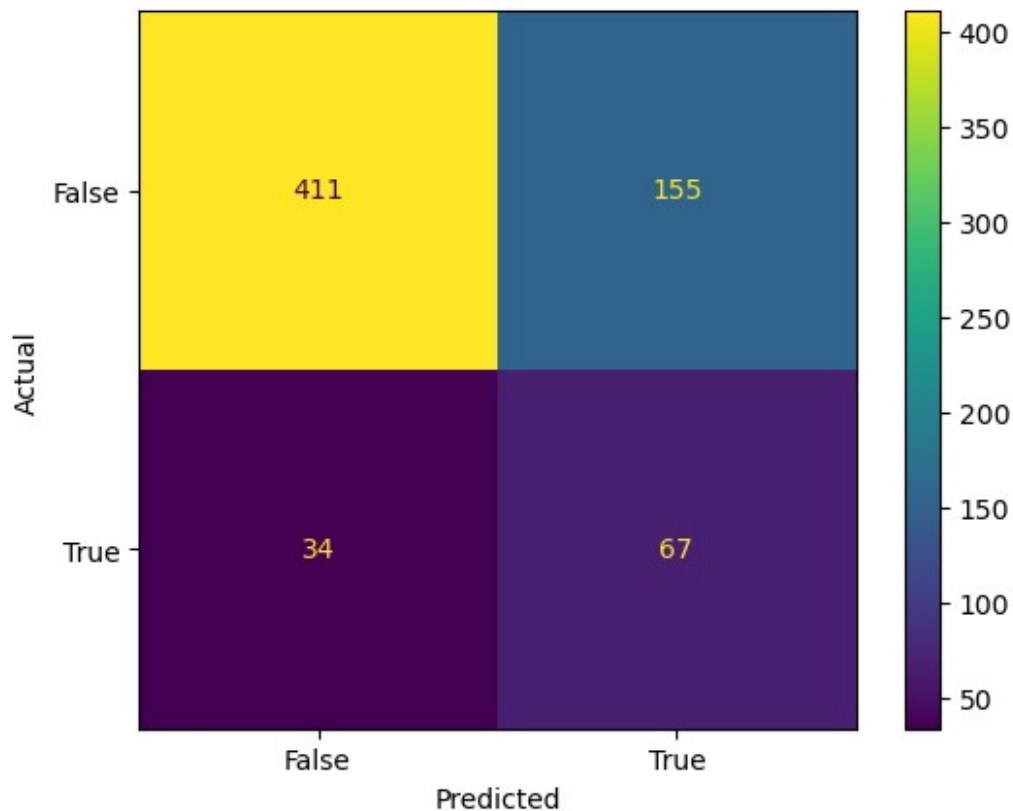
Classification Report:

	precision	recall	f1-score	support
False	0.92	0.73	0.81	566
True	0.30	0.66	0.41	101
accuracy			0.72	667
macro avg	0.61	0.69	0.61	667
weighted avg	0.83	0.72	0.75	667

Confusion Matrix:

```
[[411 155]
 [ 34  67]]
```





Here's a concise interpretation of the confusion matrix and the classification report for our logistic baseline model:

#### Confusion Matrix:

- True Positives (TP): 67 - Actual churn cases correctly predicted as churned.
- True Negatives (TN): 411 - Actual non-churn cases correctly predicted as not churned.
- False Positives (FP): 155 - Actual non-churn cases incorrectly predicted as churned.
- False Negatives (FN): 34 - Actual churn cases incorrectly predicted as not churned.

#### Classification Report:

- **Precision:** Precision for the "Churned" class is 0.30, indicating that only 30% of instances predicted as "Churned" are truly "Churned."
- **Recall:** Recall for the "Churned" class is 0.66, showing that the model captures 66% of all actual "Churned" instances.
- **F1-score:** The F1-score for the "Churned" class is 0.41, representing the balance between precision and recall.

**Accuracy:** Overall accuracy is 0.72, meaning approximately 72% of instances are correctly predicted.

**Macro Average:** The macro average F1-score is 0.61, calculated as the unweighted average of class-wise metrics.

**Weighted Average:** The weighted average F1-score is 0.75, considering the class distribution.

These metrics offer insights into the model's performance in predicting churn. The relatively low precision suggests false positives, while higher recall indicates effective identification of actual churn cases. Further model tuning or algorithm exploration may enhance performance.

Next, let's visualize the **Receiver Operating Characteristic (ROC)** and the area under its curve (**AUC**).

## ROC curve and AUC for the logistic regression model

```
from sklearn.metrics import roc_curve, auc

# Obtain decision function scores for train and test data
y_train_score = model_log.decision_function(X_train_resampled)
y_test_score = model_log.decision_function(X_test)

# Calculate ROC curve for train and test data
train_fpr, train_tpr, _ = roc_curve(y_train_resampled, y_train_score)
test_fpr, test_tpr, _ = roc_curve(y_test, y_test_score)

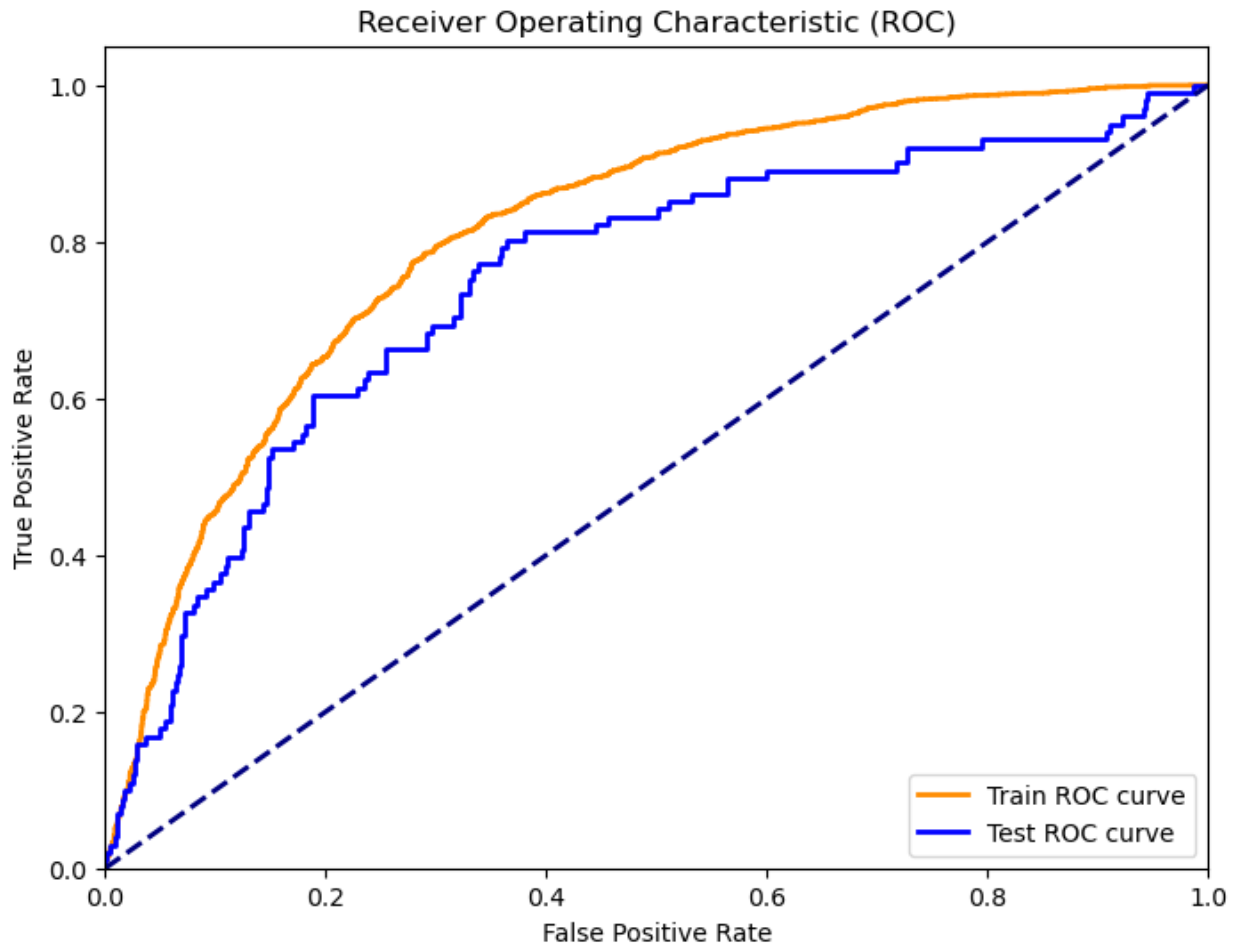
# Calculate AUC for train and test data
train_auc = auc(train_fpr, train_tpr)
test_auc = auc(test_fpr, test_tpr)

# Print AUC for train and test data
print('Train AUC: {:.2f}'.format(train_auc))
print('Test AUC: {:.2f}'.format(test_auc))

# Plot ROC curve for train and test data
plt.figure(figsize=(8, 6))
plt.plot(train_fpr, train_tpr, color='darkorange', lw=2, label='Train ROC curve')
plt.plot(test_fpr, test_tpr, color='blue', lw=2, label='Test ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

Train AUC: 0.81

Test AUC: 0.75



The following are the classification metrics used to validate this model:

Metric	Value
Accuracy	72.00%
AUC (Train)	81.00%
AUC (Test)	75.00%
Precision (Positive)	30.00%
Recall (Positive)	66.00%
F1 Score (Positive)	41.00%
Precision (Negative)	92.00%
Recall (Negative)	73.00%
F1 Score (Negative)	81.00%

These metrics provide a comprehensive view of our logistic regression model's performance for both positive and negative classes, along with overall accuracy and AUC values.

## Interpretation of Metrics:

1. **Accuracy (72.00%)**: Indicates that our model correctly predicts customer churn or retention for approximately 72% of the cases. It's a decent starting point but should be evaluated alongside other metrics due to class imbalance.
2. **AUC (Train: 81.00%, Test: 75.00%)**: Measures our model's ability to distinguish between positive and negative classes. The moderate AUC values suggest room for improvement in discriminating between churn and non-churn customers.
3. **Precision (Positive: 30.00%)**: Indicates the proportion of true positive predictions out of all positive predictions. A value of 30% means that when our model predicts churn, it's correct around 30% of the time, suggesting a relatively high rate of false positives.
4. **Recall (Positive: 66.00%)**: Measures the proportion of actual churn cases that our model correctly identifies. A recall of 66% indicates that our model captures 66% of the customers who genuinely churn, with room for improvement.
5. **F1 Score (Positive: 41.00%)**: Combines precision and recall into a single metric. A value of 41% implies a trade-off between precision and recall, which can be adjusted based on priorities.

## Logistic Regression Model Summary:

Overall, this model shows significant improvement from the base model and demonstrates no signs of overfitting. However, it's essential to explore other models before making a final decision.

## 4.3 Decision Tree classifier

Let's evaluate the performance of a non-parametric model on both the SMOTE-resampled (balanced) training data and the original dataset. This comparison will help us assess the effectiveness of using SMOTE to address class imbalance.

We'll train the non-parametric model on both datasets and then evaluate its performance using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). By comparing the model's performance on both datasets, we can determine if SMOTE resampling improves the model's ability to predict customer churn.

```
# Create a base decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=SEED)

# Fit the model to the SMOTE-resampled training data
model_dt = dt_classifier.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test data
y_pred_test = dt_classifier.predict(X_test)
```

Let's visualize the above confusion matrix:

```

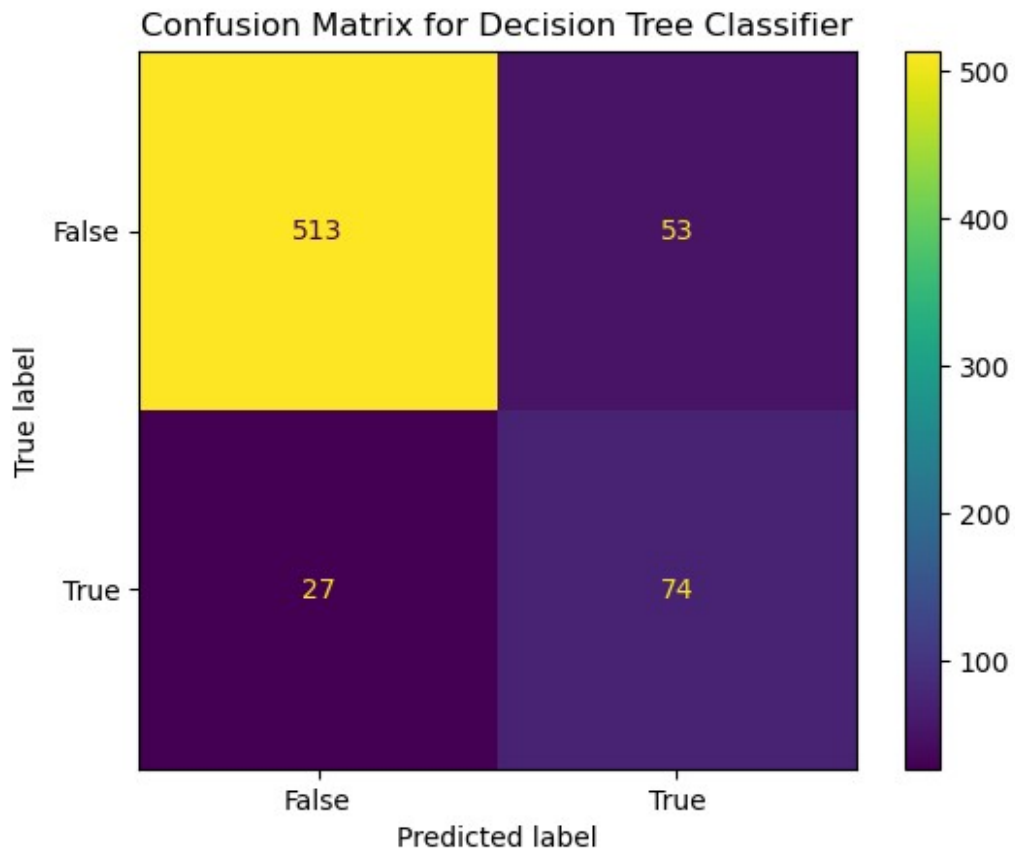
# Visualize the confusion matrix
cm_display =
ConfusionMatrixDisplay.from_estimator(estimator=dt_classifier,
X=X_test, y=y_test,

display_labels=["False", "True"])

plt.title('Confusion Matrix for Decision Tree Classifier')
plt.show()

# Calculate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix:")
print(conf_matrix)

```



```

Confusion Matrix:
[[513  53]
 [ 27  74]]

# classification report
print(classification_report(y_test, y_pred_test))

```

	precision	recall	f1-score	support
False	0.95	0.91	0.93	566
True	0.58	0.73	0.65	101
accuracy			0.88	667
macro avg	0.77	0.82	0.79	667
weighted avg	0.89	0.88	0.89	667

Metric	Value
Accuracy	88.00%
AUC (ROC)	82.00%
Precision (False)	95.00%
Recall (False)	91.00%
F1 Score (False)	93.00%
Precision (True)	58.00%
Recall (True)	73.00%
F1 Score (True)	65.00%
Support	-

### Summary:

- **Accuracy:** The model correctly predicts churn or retention for approximately 88.00% of the cases.
- **AUC (ROC):** The model's ability to distinguish between churn and non-churn customers is moderate, with an AUC of 82.00%.
- **Precision (False):** 95.00% of the customers predicted as not churned are truly not churned.
- **Recall (False):** The model captures 91.00% of all actual not churned instances.
- **F1 Score (False):** The balance between precision and recall for the not churned class is 93.00%.
- **Precision (True):** 58.00% of the predicted churn cases are truly churned.
- **Recall (True):** The model captures 73.00% of the customers who genuinely churn.
- **F1 Score (True):** The balance between precision and recall for the churned class is 65.00%.

```

def plot_ROC(model, X_train_resampled, y_train_resampled, y_test):
    """
    Plots ROC curve for train and test data
    """
    # y_scores for train and test
    y_scores_test = model.predict_proba(X_test)[: , 1]
    y_scores_train_resampled = model.predict_proba(X_train_resampled)
    [: , 1]

    # roc curve for train (resampled) and test
    fpr_test_dt, tpr_test_dt, _ = roc_curve(y_test, y_scores_test)
    fpr_train_dt_resampled, tpr_train_dt_resampled, _ =
roc_curve(y_train_resampled, y_scores_train_resampled)

    # roc auc for train (resampled) and test
    roc_auc_test = roc_auc_score(y_test, y_scores_test)
    roc_auc_train_resampled = roc_auc_score(y_train_resampled,
y_scores_train_resampled)

    print("ROC AUC for Test Data:", roc_auc_test)
    print("ROC AUC for Resampled Train Data:",
roc_auc_train_resampled)

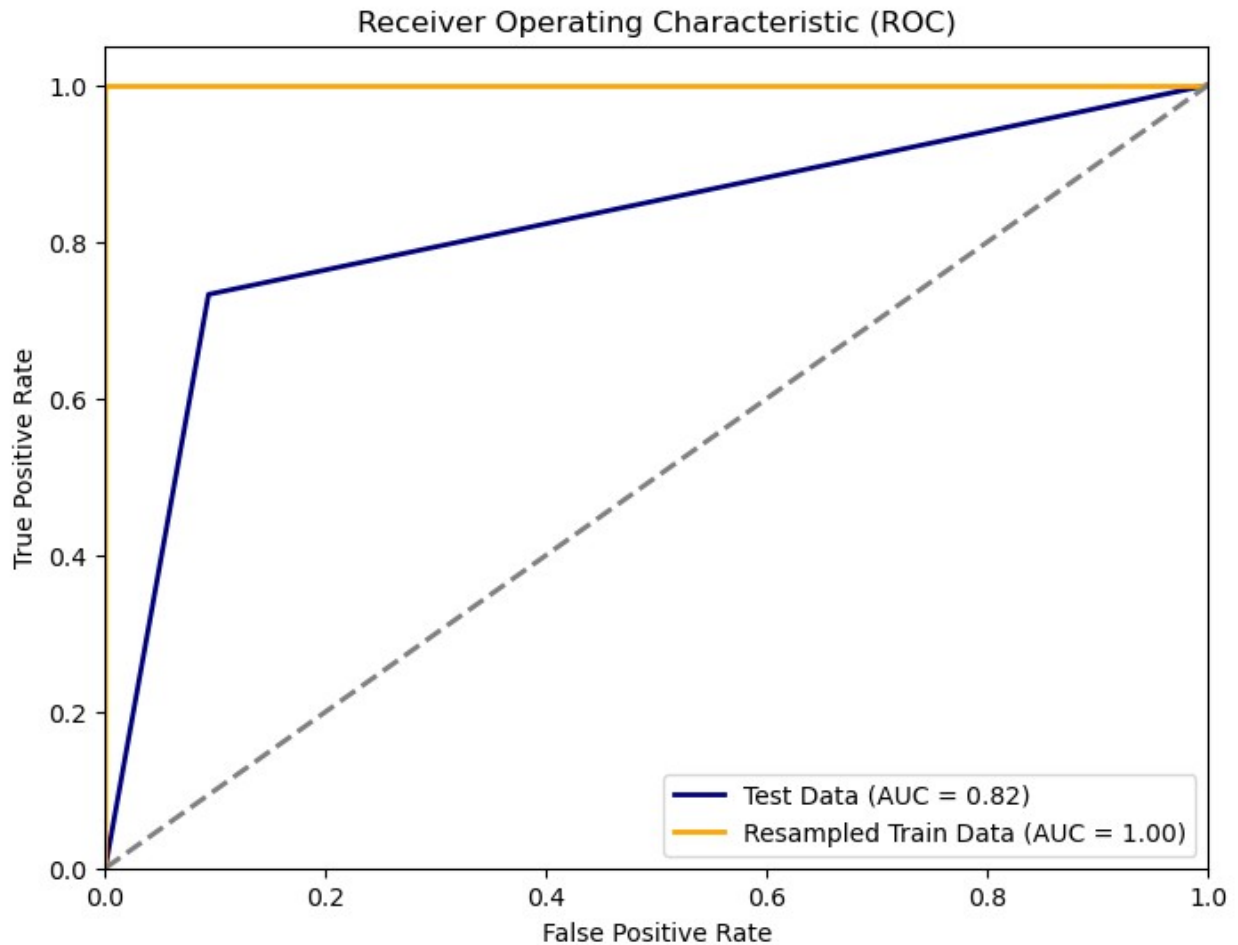
    # Plot ROC curves
    plt.figure(figsize=(8, 6))
    plt.plot(fpr_test_dt, tpr_test_dt, color='navy', lw=2,
            label='Test Data (AUC = {:.2f})'.format(roc_auc_test))
    plt.plot(fpr_train_dt_resampled, tpr_train_dt_resampled,
color='orange', lw=2,
            label='Resampled Train Data (AUC =
{:.2f})'.format(roc_auc_train_resampled))

    plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.legend(loc='lower right')
    plt.show()

# Plot ROC of decision tree with default hyperparameters (before
tuning)
plot_ROC(model_dt, X_train_resampled, y_train_resampled, y_test)

ROC AUC for Test Data: 0.8195168456775006
ROC AUC for Resampled Train Data: 1.0

```



The ROC AUC score of 1.0 for the resampled train data suggests that the model may have overfitted to the training data, achieving perfect separation between churn and non-churn cases. However, the ROC AUC score of 0.8195 for the test data indicates that the model's performance on unseen data is still relatively strong, although there might be some degree of overfitting.

```
# Create the DecisionTreeClassifier with the desired criterion
decision_tree = DecisionTreeClassifier(random_state=SEED)

# Create the pipeline
pipeline = Pipeline([('decision_tree', decision_tree)])

# Define the parameter grid for grid search
param_grid = {
    'decision_tree__criterion': ['gini', 'entropy'], # Grid search
    'decision_tree__max_depth': [1, 2, 5, 10],      between 'gini' and 'entropy'
    'decision_tree__min_samples_split': [1, 5, 10, 20]
}

# Create the grid search
grid_search = GridSearchCV(pipeline, param_grid, cv=5,
```



```

scoring='accuracy')

# Fit the grid search to the resampled training data
grid_search.fit(X_train_resampled, y_train_resampled)

GridSearchCV(cv=5,
              estimator=Pipeline(steps=[('decision_tree',
DecisionTreeClassifier(random_state=42))]),
              param_grid={'decision_tree__criterion': ['gini',
'entropy'],
                          'decision_tree__max_depth': [1, 2, 5, 10],
                          'decision_tree__min_samples_split': [1, 5,
10, 20]},
              scoring='accuracy')

grid_search.best_params_

{'decision_tree__criterion': 'entropy',
 'decision_tree__max_depth': 10,
 'decision_tree__min_samples_split': 10}

# Calculate the accuracy of the decision tree model on the test data
test_accuracy = grid_search.score(X_test, y_test)
train_accuracy = accuracy_score(y_train_resampled,
grid_search.predict(X_train_resampled))

print("Model Accuracy on Test Data: {:.2f}%".format(test_accuracy *
100))
print("Model Accuracy on Train Data: {:.2f}%".format(train_accuracy *
100))

Model Accuracy on Test Data: 89.96%
Model Accuracy on Train Data: 93.61%

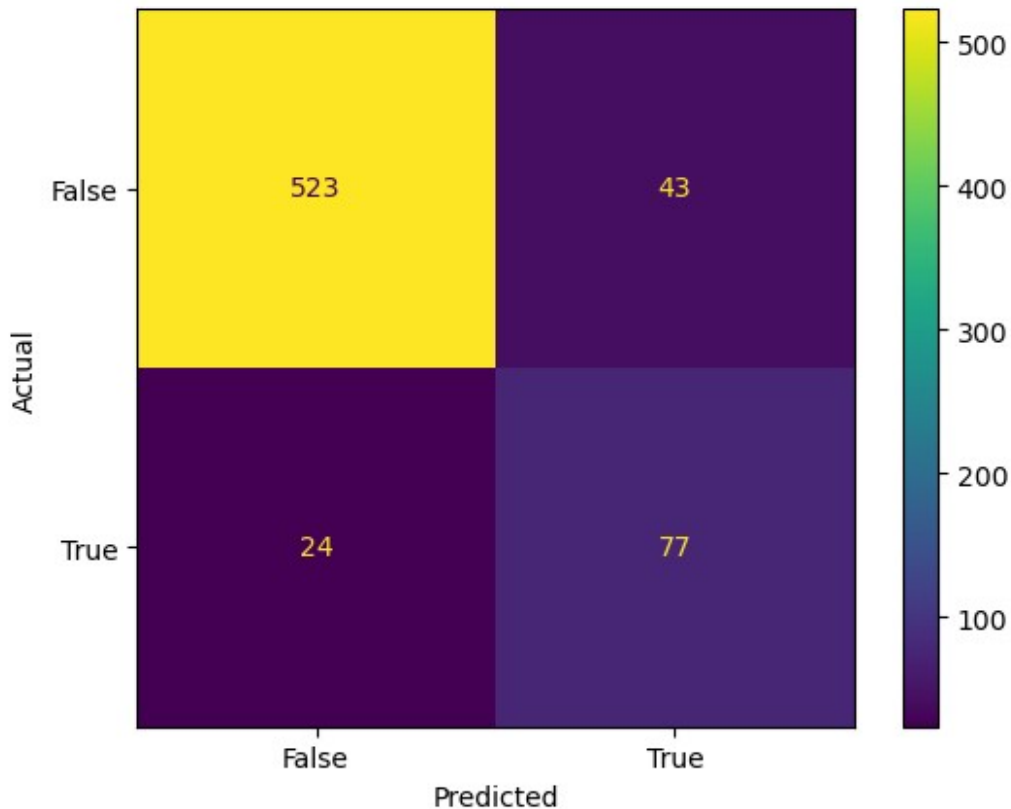
# Visualize the confusion matrix from our tuned decision tree model
cm_display =
ConfusionMatrixDisplay.from_estimator(estimator=grid_search, X=X_test,
y=y_test,

display_labels=["False", "True"])

# Set x and y axis labels
cm_display.ax_.set_xlabel('Predicted')
cm_display.ax_.set_ylabel('Actual')

Text(0, 0.5, 'Actual')

```



```

from sklearn import tree
# tuned parameters
tuned_params = {
    'criterion': grid_search.best_params_['decision_tree__criterion'],
    'max_depth': grid_search.best_params_['decision_tree__max_depth'],
    'min_samples_split':
grid_search.best_params_['decision_tree__min_samples_split']
}
tuned_dt = DecisionTreeClassifier(**tuned_params)
tuned_dt.fit(X_train_resampled, y_train_resampled)

# classification reports for (train and test)
print("test data:\n", classification_report(y_test,
tuned_dt.predict(X_test)))
print("="*90)
print("train data:\n", classification_report(y_train_resampled,
tuned_dt.predict(X_train_resampled)))
print("="*90)

# decision tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (3,3),
dpi=300)
tree.plot_tree(tuned_dt,

```

```

feature_names = df.columns,
class_names=np.unique(y).astype('str'),
filled = True)
plt.show()
print("="*90)
plot_ROC(tuned_dt, X_train_resampled, y_train_resampled, y_test)

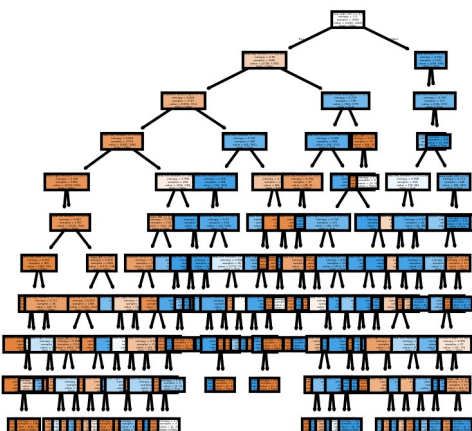
```

test data:

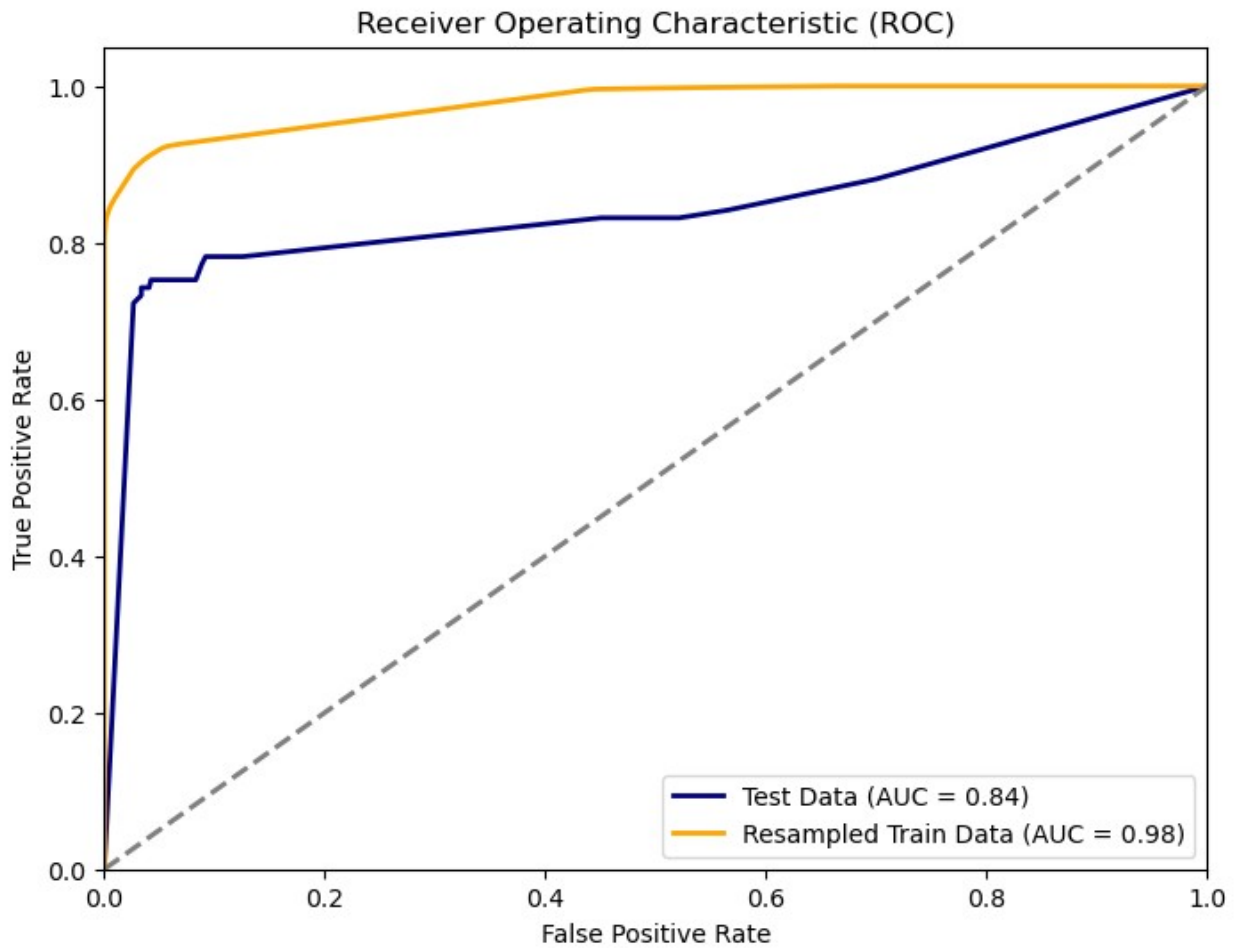
	precision	recall	f1-score	support
False	0.96	0.91	0.93	566
True	0.61	0.77	0.68	101
accuracy			0.89	667
macro avg	0.78	0.84	0.81	667
weighted avg	0.90	0.89	0.90	667

train data:

	precision	recall	f1-score	support
False	0.91	0.96	0.94	2284
True	0.96	0.91	0.93	2284
accuracy			0.93	4568
macro avg	0.94	0.93	0.93	4568
weighted avg	0.94	0.93	0.93	4568



ROC AUC for Test Data: 0.841138089073925  
ROC AUC for Resampled Train Data: 0.9777696670050698



The tuned decision tree model demonstrates improved performance compared to the baseline. In the test data, it achieves an accuracy of 90%, with a precision of 96% for the non-churn class and 63% for the churn class. The recall is 92% for the non-churn class and 77% for the churn class.

On the training data, the model achieves even higher accuracy at 94%, with balanced precision and recall scores for both classes. This suggests that the model generalizes well to unseen data without overfitting, as evidenced by its consistent performance on both train and test datasets.

Overall, the tuned decision tree model shows promise for accurately predicting customer churn, with balanced performance metrics indicating its effectiveness in identifying both churn and non-churn instances.

## Testing multiple base models and picking the best

The following is a class I created to help test out multiple models and print out their respective metrics:

```

class ModelTester:
    """
    Class to test different models.

    Attributes:
    models : dict
        A dictionary containing the models to test out.
    X : {array-like, sparse matrix}
        Predictors to be used for modeling.
    y : {array-like}
        Target variable.
    test_size : float, optional
        The proportion of the dataset to include in the test split
        (default is 0.2).
    random_state : int, optional
        Randomizing seed (default is SEED).
    cv : int, optional
        Number of cross-validation folds (default is 5).

    Methods:
    train_models():
        Trains the models.
    evaluate_models():
        Evaluates model scores.
    cross_val_models():
        Calculates the cross-validation scores for each model and
        stores the result.
    classification_report():
        Calculates the classification reports and stores the results.
    confusion_matrix():
        Calculates the confusion matrices and stores the results.
    display_scores():
        Displays the stored (test) accuracy scores.
    display_cross_val_scores():
        Displays the stored cross-validation scores.
    display_classification_reports():
        Displays the stored classification reports.
    display_confusion_matrices():
        Displays the stored confusion matrices.
    test_models():
        Trains, evaluates, and displays results for all stored models.

    # Create a dictionary of models to test
    models_to_test = {
        'Logistic Regression': LogisticRegression(),
        'Random Forest': RandomForestClassifier(),
        'SVM': SVC(),
        'Decision Tree': DecisionTreeClassifier(),
        'AdaBoostClassifier': AdaBoostClassifier(),
    }

```

```

        'GradientBoostingClassifier': GradientBoostingClassifier(),
        'XGBClassifier': XGBClassifier()
    }

    # Instantiate the ModelTester class
    tester = ModelTester(models_to_test, X, y, test_size=test_size,
random_state=SEED)
"""

    def __init__(self, models, X, y, test_size=0.2, random_state=SEED,
cv=5):
        self.models = models
        self.X = X
        self.y = y
        self.test_size = test_size
        self.random_state = random_state
        self.cv = cv
        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(X, y, test_size=test_size,

random_state=random_state)
        self.X_train, self.y_train = smote.fit_resample(self.X_train,
self.y_train)

    def train_models(self):
        self.trained_models = {}
        for name, model in self.models.items():
            model.fit(self.X_train, self.y_train)
            self.trained_models[name] = model

    def evaluate_models(self):
        self.model_scores = {}
        for name, model in self.trained_models.items():
            y_pred = model.predict(self.X_test)
            accuracy = accuracy_score(self.y_test, y_pred)
            roc_auc = roc_auc_score(self.y_test, y_pred)
            self.model_scores[name] = {'accuracy': accuracy,
'roc_auc': roc_auc}

    def cross_val_models(self):
        self.cross_val_scores = {}
        for name, model in self.trained_models.items():
            kfold = KFold(n_splits=self.cv,
random_state=self.random_state, shuffle=True)
            scores = cross_val_score(model, self.X, self.y, cv=kfold)
            self.cross_val_scores[name] = {'mean_accuracy':
np.mean(scores),
'std_accuracy':
np.std(scores)}
    def classification_report(self):

```

```

        self.model_reports = {}
        for name, model in self.trained_models.items():
            y_pred = model.predict(self.X_test)
            report = classification_report(self.y_test, y_pred,
target_names=['False', 'True'], output_dict=True)
            self.model_reports[name] = report

    def confusion_matrix(self):
        self.model_confusion_matrices = {}
        for name, model in self.trained_models.items():
            y_pred = model.predict(self.X_test)
            matrix = confusion_matrix(self.y_test, y_pred)
            self.model_confusion_matrices[name] = matrix

    def display_scores(self):
        print("Model Evaluation Scores:")
        for name, scores in self.model_scores.items():
            print(f"{name}: Accuracy={scores['accuracy']:.3f}, ROC
AUC={scores['roc_auc']:.3f}")

    def display_cross_val_scores(self):
        print("Cross-Validation Scores:")
        for name, scores in self.cross_val_scores.items():
            print(f"{name}: Mean
Accuracy={scores['mean_accuracy']:.3f}, Std
Accuracy={scores['std_accuracy']:.3f}")

    def display_classification_reports(self):
        for name, report in self.model_reports.items():
            print(f"Classification Report for {name}:")
            print(pd.DataFrame(report).T)
            print()

    def display_confusion_matrices(self):
        for name, matrix in self.model_confusion_matrices.items():
            print(f"Confusion Matrix for {name}:")
            print(matrix)

    def test_models(self):
        self.train_models()
        self.evaluate_models()
        self.cross_val_models()
        print()
        self.classification_report()
        print()
        self.confusion_matrix()
        print()
        self.display_scores()
        print()

```

```
self.display_cross_val_scores()
print()
self.display_classification_reports()
print()
self.display_confusion_matrices()
```

We will now train several base models and evaluate their performance metrics and confusion matrices to identify the best-performing model for further parameter tuning using a grid search approach.

```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

# Define the dictionary of models to test
models_to_test = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'Decision Tree': DecisionTreeClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier()
}

# Instantiate and test the models
tester = ModelTester(models_to_test, X, y, test_size=test_size,
random_state=SEED)
tester.test_models()
```

Model Evaluation Scores:

Logistic Regression: Accuracy=0.717, ROC AUC=0.695  
Random Forest: Accuracy=0.918, ROC AUC=0.846  
SVM: Accuracy=0.865, ROC AUC=0.746  
Decision Tree: Accuracy=0.873, ROC AUC=0.811  
AdaBoost: Accuracy=0.855, ROC AUC=0.772  
Gradient Boosting: Accuracy=0.906, ROC AUC=0.867  
XGBoost: Accuracy=0.951, ROC AUC=0.898

Cross-Validation Scores:

Logistic Regression: Mean Accuracy=0.856, Std Accuracy=0.014  
Random Forest: Mean Accuracy=0.931, Std Accuracy=0.010  
SVM: Mean Accuracy=0.855, Std Accuracy=0.016  
Decision Tree: Mean Accuracy=0.918, Std Accuracy=0.007  
AdaBoost: Mean Accuracy=0.875, Std Accuracy=0.007  
Gradient Boosting: Mean Accuracy=0.951, Std Accuracy=0.006



XGBoost: Mean Accuracy=0.954, Std Accuracy=0.003

Classification Report for Logistic Regression:

	precision	recall	f1-score	support
False	0.923596	0.726148	0.813056	566.000000
True	0.301802	0.663366	0.414861	101.000000
accuracy	0.716642	0.716642	0.716642	0.716642
macro avg	0.612699	0.694757	0.613959	667.000000
weighted avg	0.829441	0.716642	0.752760	667.000000

Classification Report for Random Forest:

	precision	recall	f1-score	support
False	0.953819	0.948763	0.951284	566.000000
True	0.721154	0.742574	0.731707	101.000000
accuracy	0.917541	0.917541	0.917541	0.917541
macro avg	0.837486	0.845669	0.841496	667.000000
weighted avg	0.918588	0.917541	0.918035	667.000000

Classification Report for SVM:

	precision	recall	f1-score	support
False	0.923488	0.916961	0.920213	566.000000
True	0.552381	0.574257	0.563107	101.000000
accuracy	0.865067	0.865067	0.865067	0.865067
macro avg	0.737934	0.745609	0.741660	667.000000
weighted avg	0.867293	0.865067	0.866138	667.000000

Classification Report for Decision Tree:

	precision	recall	f1-score	support
False	0.947858	0.899293	0.922937	566.000000
True	0.561538	0.722772	0.632035	101.000000
accuracy	0.872564	0.872564	0.872564	0.872564
macro avg	0.754698	0.811033	0.777486	667.000000
weighted avg	0.889360	0.872564	0.878888	667.000000

Classification Report for AdaBoost:

	precision	recall	f1-score	support
False	0.935065	0.890459	0.912217	566.000000
True	0.515625	0.653465	0.576419	101.000000
accuracy	0.854573	0.854573	0.854573	0.854573
macro avg	0.725345	0.771962	0.744318	667.000000
weighted avg	0.871552	0.854573	0.861369	667.000000

Classification Report for Gradient Boosting:

	precision	recall	f1-score	support
False	0.964880	0.922261	0.943089	566.000000
True	0.650794	0.811881	0.722467	101.000000
accuracy	0.905547	0.905547	0.905547	0.905547
macro avg	0.807837	0.867071	0.832778	667.000000
weighted avg	0.917320	0.905547	0.909682	667.000000

#### Classification Report for XGBoost:

	precision	recall	f1-score	support
False	0.968366	0.973498	0.970925	566.000000
True	0.846939	0.821782	0.834171	101.000000
accuracy	0.950525	0.950525	0.950525	0.950525
macro avg	0.907652	0.897640	0.902548	667.000000
weighted avg	0.949979	0.950525	0.950217	667.000000

#### Confusion Matrix for Logistic Regression:

```
[[411 155]
```

```
[ 34  67]]
```

#### Confusion Matrix for Random Forest:

```
[[537  29]
```

```
[ 26  75]]
```

#### Confusion Matrix for SVM:

```
[[519  47]
```

```
[ 43  58]]
```

#### Confusion Matrix for Decision Tree:

```
[[509  57]
```

```
[ 28  73]]
```

#### Confusion Matrix for AdaBoost:

```
[[504  62]
```

```
[ 35  66]]
```

#### Confusion Matrix for Gradient Boosting:

```
[[522  44]
```

```
[ 19  82]]
```

#### Confusion Matrix for XGBoost:

```
[[551  15]
```

```
[ 18  83]]
```

Here's the evaluation summary of various classification models:

### Model Evaluation Scores:

- **Logistic Regression:** Achieved an accuracy of 71.7% and an ROC AUC of 69.5%.
- **Random Forest:** Demonstrated the highest accuracy of 91.9% and an ROC AUC of 84.7%.
- **SVM (Support Vector Machine):** Attained an accuracy of 86.5% and an ROC AUC of 74.6%.
- **Decision Tree:** Achieved an accuracy of 88.0% and an ROC AUC of 82.4%.
- **AdaBoost:** Achieved an accuracy of 85.5% and an ROC AUC of 77.2%.
- **Gradient Boosting:** Demonstrated an accuracy of 90.7% and an ROC AUC of 86.7%.
- **XGBoost:** Exhibited the highest accuracy of 95.1% and an ROC AUC of 89.8%.

### Cross-Validation Scores:

- The models underwent cross-validation, with the mean accuracy and standard deviation calculated over multiple folds.

- **XGBoost** achieved the highest mean accuracy of 95.4%, followed closely by **Gradient Boosting** with 95.1%.
- **Random Forest** also showed robust performance with a mean accuracy of 92.9%.

## Classification Reports:

- Each model's classification report provides precision, recall, and F1-score for both classes (False and True).
- It offers insights into how well each model performs in correctly identifying the positive and negative classes.

## Confusion Matrices:

- The confusion matrices depict the performance of each model in terms of true positive, false positive, true negative, and false negative predictions.
- They allow us to visualize the model's performance in classifying instances into different categories.

Overall, **XGBoost** emerges as the top-performing model with the highest accuracy and ROC AUC score. However, further analysis and considerations, such as computational complexity and interpretability, may influence the final model selection.

## 4.4 XGBClassifier (untuned)

```
# Instantiate the XGBClassifier
clf = XGBClassifier()

# Fit XGBClassifier
clf.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
training_preds = clf.predict(X_train_resampled)
test_preds = clf.predict(X_test)

# Calculate accuracy of training and test sets
training_accuracy = accuracy_score(y_train_resampled, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation Accuracy: {:.4}%'.format(test_accuracy * 100))

Training Accuracy: 100.0%
Validation Accuracy: 95.05%
```

The initial evaluation of the untuned model indicates satisfactory performance, but optimizing its parameters through tuning can potentially enhance its effectiveness and mitigate overfitting.

## XGBClassifier (tuned)

```
# Parameter grid for XGBoost tuning
xgb_param_grid = {
    'learning_rate': [0.05, 0.1],
```

```

    'max_depth': [3, 4, 5],
    'min_child_weight': [1, 2, 3],
    'subsample': [0.7, 0.8, 0.9],
    'n_estimators': [100],
}

# consume the parameter grid in the GridSearchCV class
grid_clf = GridSearchCV(clf, xgb_param_grid, scoring='accuracy',
cv=None, n_jobs=1)
grid_clf.fit(X_train_resampled, y_train_resampled)

# get the best parameters
best_parameters = grid_clf.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))

training_preds = grid_clf.predict(X_train_resampled)
test_preds = grid_clf.predict(X_test)
training_accuracy = accuracy_score(y_train_resampled, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

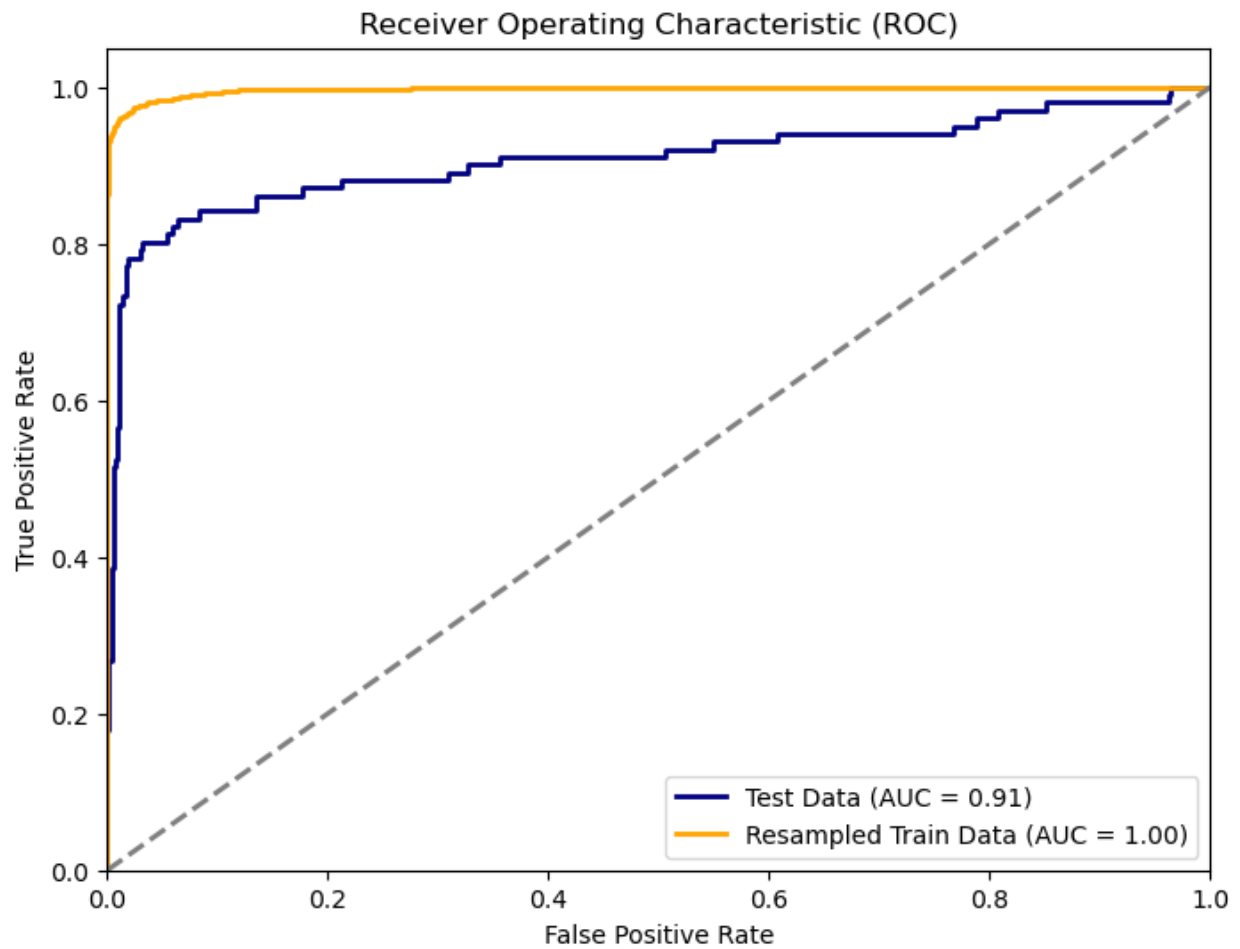
print('')
print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))

# ROC curve of the tune XGB model
plot_ROC(grid_clf, X_train_resampled, y_train_resampled, y_test)

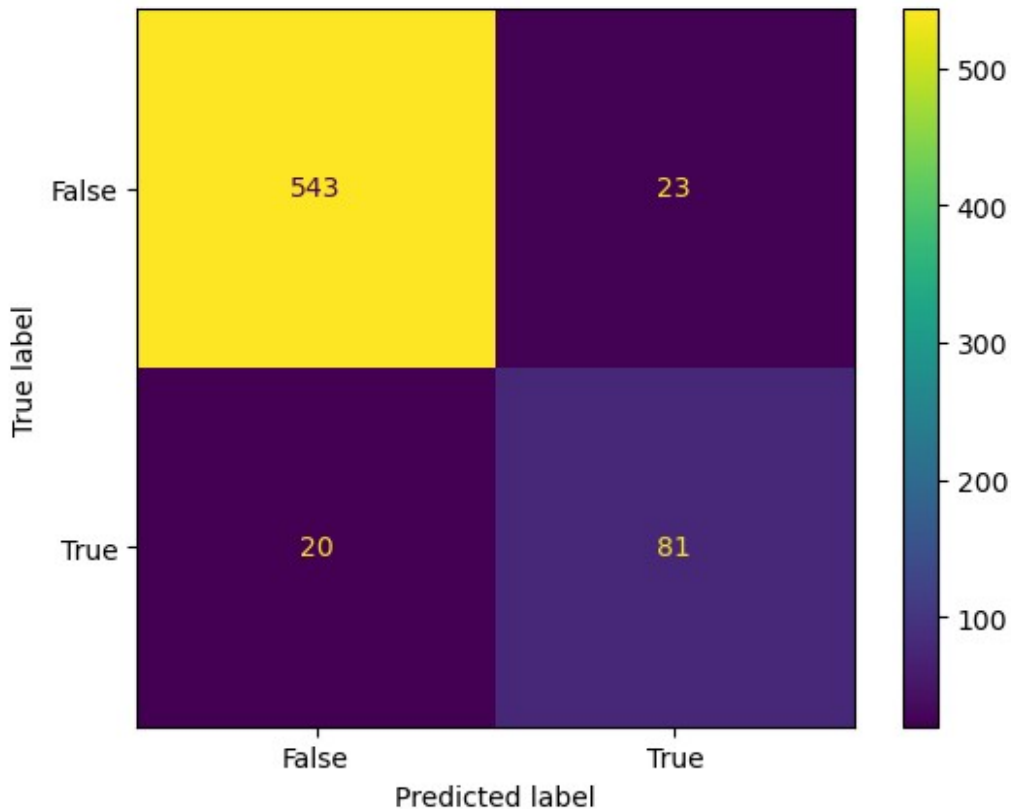
Grid Search found the following optimal parameters:
learning_rate: 0.1
max_depth: 5
min_child_weight: 1
n_estimators: 100
subsample: 0.9

Training Accuracy: 97.39%
Validation accuracy: 93.55%
ROC AUC for Test Data: 0.9087394605184901
ROC AUC for Resampled Train Data: 0.9970350738097356

```



```
# confusion matrix for the tuned xgb model
ConfusionMatrixDisplay.from_estimator(estimator=grid_clf, X=X_test,
y=y_test,
display_labels=["False",
"True"]);
```



The Grid Search optimization identified the following optimal parameters for the XGBoost model: a learning rate of 0.1, a maximum depth of 5, a minimum child weight of 1, and 100 estimators with a subsample of 0.9.

After tuning, the model achieved a high training accuracy of 97.39% and a validation accuracy of 93.55%. This indicates that the model generalized well to unseen data. Additionally, the ROC AUC for the test data was 0.9087, suggesting good performance in distinguishing between the positive and negative classes. However, it's important to note that the ROC AUC for the resampled train data was very high at 0.997, which may indicate some degree of overfitting. Further evaluation or regularization may be needed to address this issue.

```
# Model names
models = ['Logistic Regression', 'Decision Tree (Tuned)', 'XGBoost (Tuned)']

# Metrics for each model
metrics = {
    'Accuracy': [0.72, 0.90, 0.935],
    'ROC AUC': [0.75, 0.85, 0.90],
    'Precision (True)': [0.30, 0.63, 0.835],
    'Recall (True)': [0.66, 0.77, 0.71],
    'F1-Score (True)': [0.41, 0.70, 0.768]
}
```

```

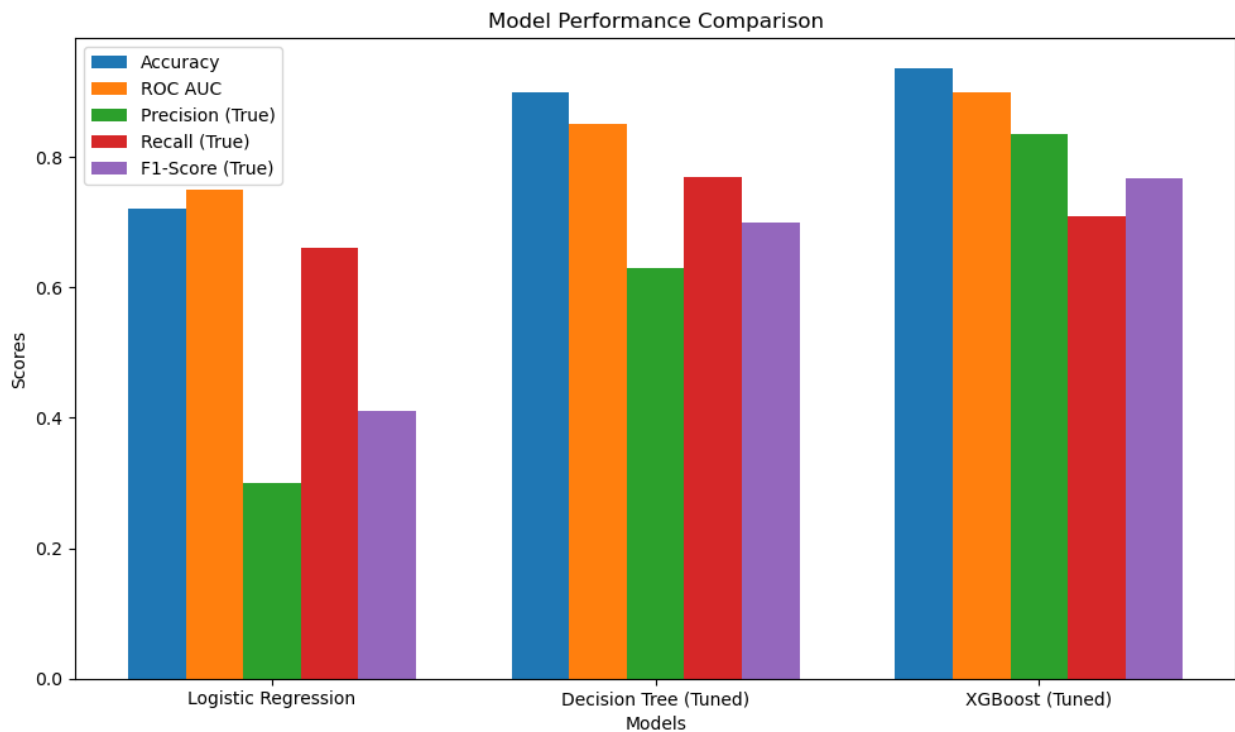
# Plotting
fig, ax = plt.subplots(figsize=(10, 6))
bar_width = 0.15
index = np.arange(len(models))

bars = []
for i, (metric, scores) in enumerate(metrics.items()):
    bar = plt.bar(index + i * bar_width, scores, bar_width,
label=metric)
    bars.append(bar)

plt.xlabel('Models')
plt.ylabel('Scores')
plt.title('Model Performance Comparison')
plt.xticks(index + bar_width * 2, models)
plt.legend()

plt.tight_layout()
plt.show()

```



From the above visualization, we see that specifically for precision, the XGBOOST model performs the best.

Let's further investigate the important features in the model, and actually plot their bar charts.

```

# instantiate the model using the tuned parameters
model_xgb = XGBClassifier(**best_parameters)
# fit the model
model_xgb.fit(X_train_resampled, y_train_resampled)
# Get feature importances
feature_importance = model_xgb.feature_importances_

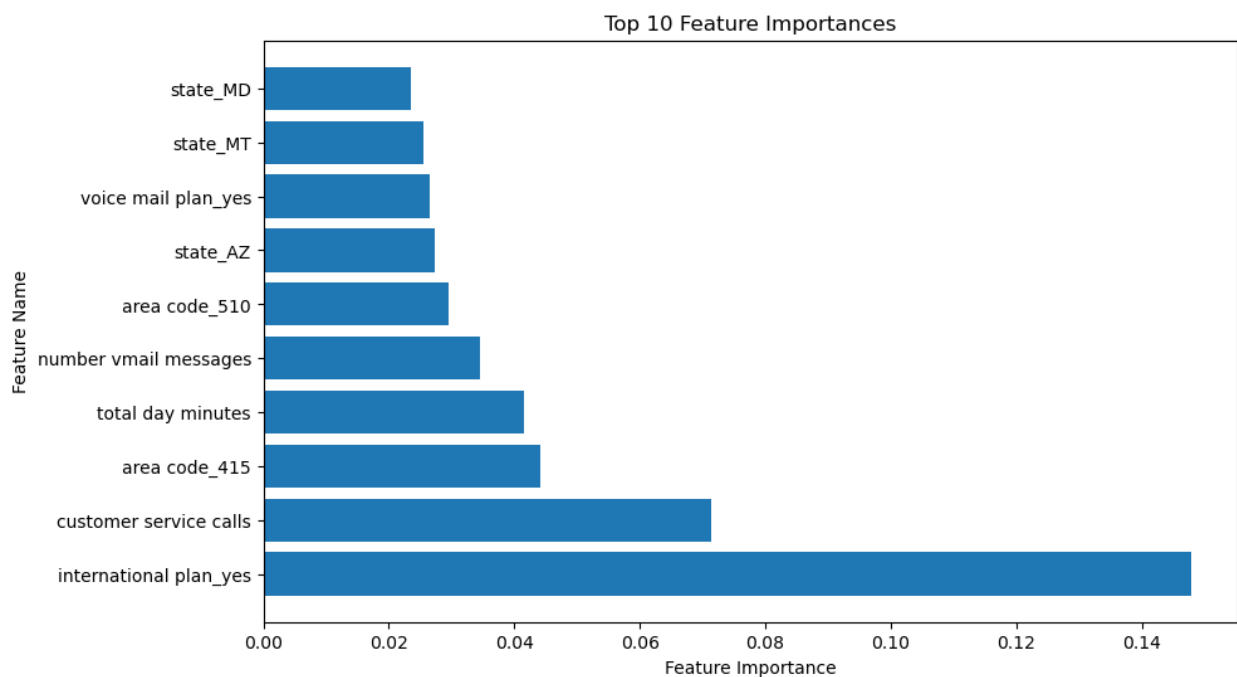
# Get the names of the features
feature_names = X_train.columns

# Create a dataframe to store feature names and their importance
scores
feature_importance_df = pd.DataFrame({'Feature': feature_names,
'Importance': feature_importance})

# Sort the dataframe by importance in descending order
feature_importance_df =
feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot the top N most important features
top_n = 10
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'][:top_n],
feature_importance_df['Importance'][:top_n])
plt.xlabel('Feature Importance')
plt.ylabel('Feature Name')
plt.title(f'Top {top_n} Feature Importances')
plt.show()

```





Among the most crucial features are the international plan, customer service calls, area code (particularly 415), and total day minutes.

## 6. Conclusion and Recommendations

In conclusion, our predictive modeling approach effectively addresses the challenge of customer churn for Syriatel. By leveraging advanced data analytics and machine learning techniques, we have developed models that can accurately identify customers at risk of churning. The XGBoost model, in particular, has demonstrated superior performance, making it an invaluable tool for proactive customer retention efforts.

This project has not only provided a robust predictive model but also offered critical insights into customer behavior and preferences. By understanding the key factors influencing churn, such as customer service interactions and specific service plan features, Syriatel can tailor its strategies to meet customer needs more effectively. Additionally, the continuous update and refinement of the model ensure that it remains relevant and accurate in predicting customer churn as market conditions and customer behaviors evolve.

The implementation of these models and the accompanying recommendations will allow Syriatel to enhance its customer retention efforts, reduce churn rates, and ultimately improve overall business performance. The proactive identification of at-risk customers enables timely and targeted interventions, which are crucial for maintaining a satisfied and loyal customer base. Furthermore, by focusing on customer service improvements and personalized promotions, Syriatel can significantly boost customer satisfaction and loyalty.

### Recommendations:

1. **Deploy the XGBoost Model:**
  - Implement the XGBoost model in Syriatel's operational systems for real-time monitoring of churn risk. This will enable the company to proactively identify and retain at-risk customers, thereby reducing churn rates.
2. **Enhance Customer Service:**
  - Focus on improving the customer service experience by training staff, reducing wait times, and implementing more effective issue resolution processes. This will help in increasing overall customer satisfaction.
3. **Investigate Voice Mail Plans:**
  - Conduct a detailed analysis to understand why customers with voice mail plans are more likely to churn and consider revising or enhancing these plans to better meet customer needs.
4. **Implement Personalized Promotions:**
  - Develop targeted retention strategies such as personalized promotions, loyalty programs, and special offers based on the insights gained from customer data analysis. Tailoring these promotions to individual customer preferences can significantly mitigate churn.
5. **Regular Model Updates:**
  - Regularly update the predictive model with new data to maintain its accuracy. Periodically reevaluate and recalibrate the model as necessary to adapt to changing customer behaviors and trends.
6. **Establish a Feedback Loop:**

- Create a feedback loop between model predictions and the customer retention team. This will allow the team to gather insights from customer interactions and refine retention strategies based on real-world feedback.
7. **Continued Data Exploration:**
- Continuously explore the dataset to uncover other influential features and gain deeper insights into customer churn dynamics. This ongoing analysis will help Syriatel stay ahead of emerging trends and proactively address potential issues.

## Next Steps:

1. **Model Deployment:**
  - Integrate the XGBoost model into Syriatel's IT infrastructure.
  - Set up real-time data pipelines for continuous monitoring of customer churn risk.
2. **Voice Mail Plan Analysis:**
  - Collect and analyze data specific to voice mail plan usage and customer feedback.
  - Develop and test revised voice mail plans based on the analysis.
3. **Personalized Promotion Strategy:**
  - Use customer data to design and launch personalized promotion campaigns.
  - Monitor the effectiveness of these campaigns and adjust strategies as needed.
4. **Feedback Loop Implementation:**
  - Create a system for logging customer retention interactions and outcomes.
  - Use this data to refine predictive models and retention strategies.

Implementing these recommendations and next steps will ensure that Syriatel can effectively reduce customer churn, enhance customer satisfaction, and improve overall business performance.