Reducing Customer Churn: Using Machine Learning to Predict Customer Retention at Syriatel Mobile Telecom

Authors:

- Edward Opollo,
- Cynthia Nasimiyu,
- John Karanja,
- Sheilah Machaha,
- Julius Charles,
- Sharon Kimutai, and
- Phelix Okumu

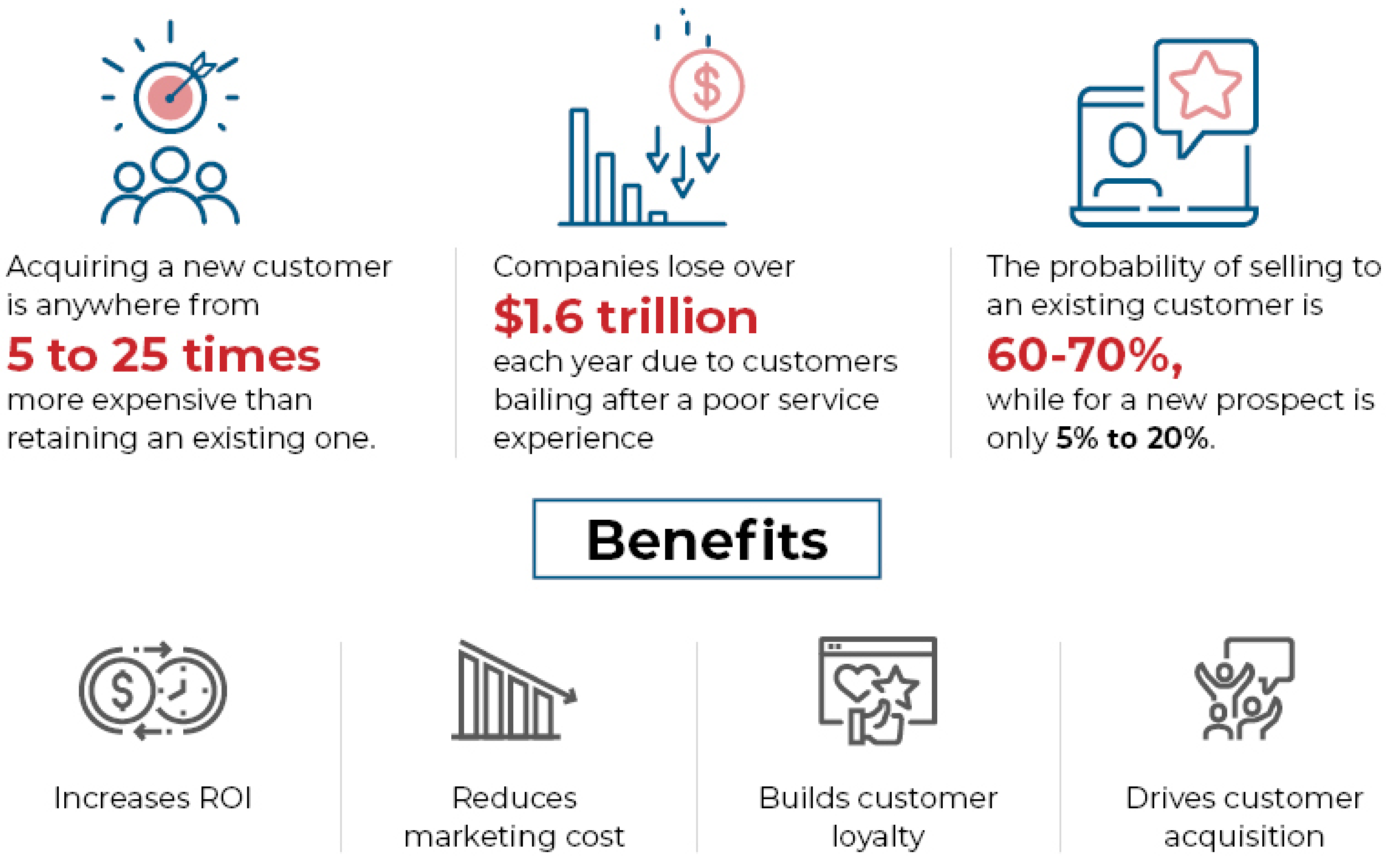
Introduction

Business growth and development remains a central motivator in organizational decision-making and policy making. Although every business leader aspires to achieve growth in revenues, clientele, and profitability, they must try as much as possible to avoid making losses.

In recent years, such leaders, as well as business experts, have identified customer satisfaction as an important factor to ensuring such growth and development. Without customers, a business would not make any sales, record any cash inflows in terms of revenues, nor make any profits. This underscores the the need for organizations to implement measures that retain existing customers.

Recent technological advancements have also contributed to an increased business rivalry, especially due to increased startups and entrants. Such competition, coupled with an augmented saturation of markets, means that it has become harder and more expensive for businesses in most sectors to acquire new clients, which means they must shift their focus to cementing relationships with existing customers.

A `2014 article`, called [The Value of Keeping the Right Customers (https://hbr.org/2014/10/the-value-of-keeping-the-right-customers)](https://hbr.org/2014/10/the-value-of-keeping-the-right-customers), written by `Amy Gallo` stresses on the importance of any business investing more to retain existing customers (avoiding customer churning) than acquiring new ones. Gallo maintains that it costs from 5 to 25 times to acquire a new customer than retain an existing one while retaining existing clients by `5%` results in profits augmenting by `25% to 95%`.



Source: [The Value of Keeping the Right Customers (https://www.netscribes.com/customer-retention-strategies/)](https://www.netscribes.com/customer-retention-strategies/)

Through this project, we are building a prediction model that identifies patterns in customer churning, which can be helpful in developing mitigation strategies. The project is structured as follows:

1. **Business Understanding**
2. **Data Understanding**
3. **Data Preparation**
4. **Exploratory Data Analysis**
5. **Modelling**
6. **Model Evaluation**
7. **Recommendations and Conclusions**

Business Understanding

With an increasing blend of factors such as competition, technological innovations, and globalization, among others in the telecommunication markets, **Syriatel Mobile Telecom** has stressed on the need to improve customer satisfaction and preserve its `8 million` clientele. Through its [linkedIn profile (https://sy.linkedin.com/company/syriatel)](https://sy.linkedin.com/company/syriatel), the Syrian telecommunication giant reiterates on its commitment to maintaining its market position by establishing `"its reputation by focusing on customer satisfaction and social responsibility."`

Although such efforts have been fruitful over the years, the company needs to increase its commitment to reducing customer charning rates, which might threaten its market position, profitability, and overall growth. Retaining the company's `8 million customers` will help the company reduce the costs, avoid losses, and increase sales. Further, such actions would contribute to an increased ROI, reduced marketing costs, augmented customer loyalty, and promote further client acquisition through referrals, as outlined by `Amy Gallo`.

Hence, this project will help **Syriatel Mobile Telecom** identify customers with highest probabilities of churning, which will be crucial for implementing new policies and business frameworks intended to ensure retention. As defined by Amy Gallo `"Customer churn rate is a metric that measures the percentage of customers who end their relationship with a company in a particular period."` In this scenario, the emphasis is on identifying prospective churners among SyriaTel's customer base and implementing the necessary strategic business decisions intended to ensure such clients are retained.

Primary stakeholder:

- Syriatel Mobile Telecom

Other Stakeholders:

- Shareholders
- Employees
- Customers

As the principle stakeholder, the company stands to benefits from this model through a reduction in customer charning rates, which has the potential to increase revenues and profits, promote growth, and sustain, or rather, increase its market position. The customers will also benefit through improved telecomunication services, not forgetting better customer service. As the company continues to grow, through revenues, profits, increased customers, and higher market share, the shareholders will also get more returns on their investments (ROI) while employees benefit from better remunerations and bonuses.

The project aims to provide value to the different stakeholders by identifying predictable patterns related to customer churn, which can help SyriaTel take proactive measures to retain customers and minimize revenue loss.

Research Objectives:

1. To identify the key features that determine if a customer is likely to churn.
2. To determine the most suitable model to predict Customer Churn.
3. To establish Cusstomer retention strategy to reduce churn

Research Questions:

- What are the most significant predictors of customer churn for Syriatel Mobile Telecom?
- Which Machine Learning Model is the most suitable in predicting Customer Churn?
- What strategies can Syriatel Mobile Telecom implement to retain customers and reduce churn rates?

Data Understanding

The Churn in Telecom's dataset from Kaggle contains information about customer activity and whether or not they canceled their subscription with the Telecom firm. The goal of this dataset is to develop predictive models that can help the telecom business reduce the amount of money lost due to customers who don't stick around for very long.

The dataset contains 3333 entries and 21 columns, including information about the state, account length, area code, phone number, international plan, voice mail plan, number of voice mail messages, total day minutes, total day calls, total day charge, total evening minutes, total evening calls, total evening charge, total night minutes, total night calls, total night charge, total international minutes, total international calls, total international charge, customer service calls and churn.

In this phase of the project, we will focus on getting familiar with the data and identifying any potential data quality issues. We will also perform some initial exploratory data analysis to discover first insights into the data.

Summary of Features in the Datset

- **State:** The state the customer lives in
- **Account Length:** The number of days the customer has had an account.
- **Area Code:** The area code of the customer
- **Phone Number:** The phone number of the customer
- **International Plan:** True if the customer has the international plan, otherwise false.
- **Voice Mail Plan:** True if the customer has the voice mail plan, otherwise false.
- **Number Vmail Messages:** the number of voicemails the customer has sent.
- **Total Day Minutes:** total number of minutes the customer has been in calls during the day.
- **Total Day Calls:** total number of calls the user has done during the day.
- **Total Day Charge:** total amount of money the customer was charged by the Telecom company for calls during the day.
- **Total Eve Minutes:** total number of minutes the customer has been in calls during the evening.
- **Total Eve Calls:** total number of calls the customer has done during the evening.
- **Total Eve Charge:** total amount of money the customer was charged by the Telecom company for calls during the evening.
- **Total Night Minutes:** total number of minutes the customer has been in calls during the night.
- **Total Night Calls:** total number of calls the customer has done during the night.

- **Total Night Charge:** total amount of money the customer was charged by the Telecom company for calls during the night.
- **Total Intl Minutes:** total number of minutes the user has been in international calls.
- **Total Intl Calls:** total number of international calls the customer has done.
- **Total Intl Charge:** total amount of money the customer was charged by the Telecom company for international calls.
- **Customer Service Calls:** number of calls the customer has made to customer service.
- **Churn:** true if the customer terminated their contract, otherwise false

Data Preparation

In this section, we are going to do several actions to prepare our data for exploratory data analysis and modelling. First, we will import all the necessary libraries, load the dataset using pandas library, preview the data (how many features and records, as well as statistical features), and conduct thorough data preprocessing (checking and removing any missing values and transforming data)

Here, we import all the libraries we will use for this project and load the data into a pandas dataframe

In [1]:
```
# Importing libraries.
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')
```

In [2]:
```
#Loading the data into a pandas dataframe
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
```

Afterward, we examine the data to determine the number of features, understand whether we have any missing values, identify columns that need transformation for modelling, and get any other insights we may need before proceeding to the next step

In [155]:
```
#Checking the general information about the df
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
```

As shown above, we have `3333 data` records and `21 columns` , with zero null values. However, we will need to review the each column further to identify anomalies, especially those in the form of placeholder values or unique characters. `Four (4)` of our columns are of the object type, while `eight (8)` are of integer type, eight `(8)` as floats, and `one (1) column` as bolean. Our target variable column is churn, which means we will treat the rest of the columns as features.

We also need preview the top 10 and top bottom 10 data records to get a glimpse of what we are dealing with.

In [156]:
```python
# checking to 10 rows
df.head(10)
```

Out[156]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	cu
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	3	2.70	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	3	3.70	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	5	3.29	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	7	1.78	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	3	2.73	
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	...	101	18.75	203.9	118	9.18	6.3	6	1.70	
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	...	108	29.62	212.6	118	9.57	7.5	7	2.03	
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.69	...	94	8.76	211.8	96	9.53	7.1	6	1.92	
8	LA	117	408	335-4719	no	no	0	184.5	97	31.37	...	80	29.89	215.8	90	9.71	8.7	4	2.35	
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...	111	18.87	326.4	97	14.69	11.2	5	3.02	

10 rows × 21 columns

In [157]:
```python
# Previewing the top 10 rows
df.tail(10)
```

Out[157]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge
3323	IN	117	415	362-5899	no	no	0	118.4	126	20.13	...	97	21.19	227.0	56	10.22	13.6	3	3.67
3324	WV	159	415	377-1164	no	no	0	169.8	114	28.87	...	105	16.80	193.7	82	8.72	11.6	4	3.13
3325	OH	78	408	368-8555	no	no	0	193.4	99	32.88	...	88	9.94	243.3	109	10.95	9.3	4	2.51
3326	OH	96	415	347-6812	no	no	0	106.6	128	18.12	...	87	24.21	178.9	92	8.05	14.9	7	4.02
3327	SC	79	415	348-3830	no	no	0	134.7	98	22.90	...	68	16.12	221.4	128	9.96	11.8	5	3.19
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83	12.56	9.9	6	2.67
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123	8.61	9.6	4	2.59
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91	8.64	14.1	6	3.81
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137	6.26	5.0	10	1.35
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77	10.86	13.7	4	3.70

10 rows × 21 columns

From above general information, most of the columns have 2 or more words as the columns names. We need to remove the whitespaces so as to make the column names easily addressible. We need to rename the column names by removing white spaces and replacing with underscore '_'

In [158]:
```python
# Removing whitespaces in the column name and replacing with '_'
df.columns = df.columns.str.replace(' ', '_')
```

In [159]:
```
# previewing the bottom 10 rows to confirm the columns names have bee formated
df.head(10)
```

Out[159]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_
0	KS	128	415	382-4657	no	yes	25	265.1	110	
1	OH	107	415	371-7191	no	yes	26	161.6	123	
2	NJ	137	415	358-1921	no	no	0	243.4	114	
3	OH	84	408	375-9999	yes	no	0	299.4	71	
4	OK	75	415	330-6626	yes	no	0	166.7	113	
5	AL	118	510	391-8027	yes	no	0	223.4	98	
6	MA	121	510	355-9993	no	yes	24	218.2	88	
7	MO	147	415	329-9001	yes	no	0	157.0	79	
8	LA	117	408	335-4719	no	no	0	184.5	97	
9	WV	141	415	330-8173	yes	yes	37	258.6	84	

10 rows × 21 columns

In [160]:
```
# checking for the general shape of the df
df.shape
```

Out[160]:
```
(3333, 21)
```

As previously confirmed, the df has 33333 rows and 21 columns

In [161]:
```
#Viewing the statistical details such as std, percentile, count, and the mean
df.describe()
```

Out[161]:

	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	total_eve_
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30

In this step we check for anormalies in the df. We need to dive deep into the data to see if we have missing values in terms of placeholder values or unique values.

Data Cleaning

Below cell checks for general information about missing values across all the columns

In [162]:
```
#confirming that there no missing values (nan) in the dataframe
missing_values = df.isnull().sum()
print(missing_values)
```

```
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
```

There are no null values across all the columns. As we can see, all columns indicate that we have zero null values. However, that does not mean that data has no missing records. As such, its important to review df further to identify values that are not a representation of the data

In that case, we take a look at each column for any anormalies such as wrong data type and unexpected records.

We start by checking the `*state column*`

In [163]:
```
# checking for value_count for the different state abbreviations
df.state.value_counts()
```

Out[163]:
```
WV    106
MN     84
NY     83
AL     80
WI     78
OH     78
OR     78
WY     77
VA     77
CT     74
MI     73
ID     73
VT     73
TX     72
UT     72
IN     71
MD     70
KS     70
NC     68
NJ     68
MT     68
CO     66
NV     66
WA     66
RI     65
MA     65
MS     65
AZ     64
FL     63
MO     63
NM     62
ME     62
ND     62
NE     61
OK     61
DE     61
SC     60
SD     60
KY     59
IL     58
NH     56
AR     55
GA     54
DC     54
HI     53
TN     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
```

Because the state column is a representation of an area code, there is no need to check for duplicates as several subsribers can be residing in the same state.

However, because we have both state and area code, we will drop state and use area code to reference geographical location. The reason for us dropping the state column is because we have the area code column, which contains information on where each client resides.

In [164]:
```
# dropping the state column
df = df.drop('state', axis=1)
```

In [165]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   phone_number            3333 non-null   object 
 3   international_plan      3333 non-null   object 
 4   voice_mail_plan         3333 non-null   object 
 5   number_vmail_messages   3333 non-null   int64  
 6   total_day_minutes       3333 non-null   float64
 7   total_day_calls         3333 non-null   int64  
 8   total_day_charge        3333 non-null   float64
 9   total_eve_minutes       3333 non-null   float64
 10  total_eve_calls         3333 non-null   int64  
 11  total_eve_charge        3333 non-null   float64
 12  total_night_minutes     3333 non-null   float64
 13  total_night_calls       3333 non-null   int64  
 14  total_night_charge      3333 non-null   float64
 15  total_intl_minutes      3333 non-null   float64
 16  total_intl_calls        3333 non-null   int64  
 17  total_intl_charge       3333 non-null   float64
 18  customer_service_calls  3333 non-null   int64  
 19  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
```

Looking at the our column information, we can see that the state column has been successfuly dropped, leaving us with the area code column.

We then proceed to check the `*Account length Column*`

In [166]:
```
# checking account length column
df.account_length.value_counts()
```

Out[166]:
```
105    43
87     42
101    40
93     40
90     39
       ..
243     1
200     1
232     1
5       1
221     1
Name: account_length, Length: 212, dtype: int64
```

Given account_length isn't unique, and no null and missing values. There is no need for further checks on this column

Afterward, we also review the `*Area Code Column*` for the possibilities of unique or missing values

In [167]:
```
df.area_code.unique()
```

Out[167]:
```
array([415, 408, 510])
```

In [168]:
```
df.area_code.value_counts()
```

Out[168]:
```
415    1655
510     840
408     838
Name: area_code, dtype: int64
```

Same as the `account_length column` , the column has no missing values and any other unexpected unique item. No further cleaning for this column

We proceed to review the `Phone Number Column`

In [169]:
```
df.phone_number
```

Out[169]:
```
0       382-4657
1       371-7191
2       358-1921
3       375-9999
4       330-6626
          ...   
3328    414-4276
3329    370-3271
3330    328-8230
3331    364-6381
3332    400-4344
Name: phone_number, Length: 3333, dtype: object
```

In [170]:
```
df.phone_number.unique
```

Out[170]:
```
<bound method Series.unique of 0       382-4657
1       371-7191
2       358-1921
3       375-9999
4       330-6626
          ...
3328    414-4276
3329    370-3271
3330    328-8230
3331    364-6381
3332    400-4344
Name: phone_number, Length: 3333, dtype: object>
```

Given that `Phone number` is the unique Identifier, let us clean it and check for any duplicates. We do not expect the same phone number to be used by two different subscribers.

As was previously observed, `phone_number column` is of object datatype. Given these are digits we need to change them to an integer data type.

In order to do this, we need to remove the `'-'` and convert the dtype to `integer` ..

In [171]:
```
# Remove hyphen and convert to integer
df['phone_number'] = df['phone_number'].str.replace('-', '').astype(int)
```

We then check if the change has been effected for this column

In [172]:
```
# checking if above conversion is effected
df.phone_number
```

Out[172]:
```
0       3824657
1       3717191
2       3581921
3       3759999
4       3306626
         ...
3328    4144276
3329    3703271
3330    3288230
3331    3646381
3332    4004344
Name: phone_number, Length: 3333, dtype: int64
```

Everything looks perfect so far, the hyphens `'-'` have been removed and datatype changed to `integer`

Nex, we check for duplicates in the `phone_numbe` column and remove them. As stated before, we do not expect one phone number to be held by two different clients. Since a phone number can be registered to only one client, each phone number will be considered to be a representation of one client.

In [173]:
```
# Check for duplicates in the 'phone number' column
duplicates = df.duplicated('phone_number')

# Filter the DataFrame to show only the duplicate rows
duplicate_rows = df[duplicates]
duplicate_rows
```

Out[173]:

	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge

As we can see, everything looks great: there are no duplicates in the `phone number column`

And since the `phone number` is a representation of one customer, we can make the `phone number column` to be the `index column` for our data.

This means that the column will be our unique identifier.

In [174]:
```
# making phone_number column to be the index column given its the unique identifier
df.set_index('phone_number', inplace=True)
```

In [175]:
```
# previewing the general info to confirm same has been reflected in the df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3333 entries, 3824657 to 4004344
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   area_code               3333 non-null   int64  
 2   international_plan      3333 non-null   object 
 3   voice_mail_plan         3333 non-null   object 
 4   number_vmail_messages   3333 non-null   int64  
 5   total_day_minutes       3333 non-null   float64
 6   total_day_calls         3333 non-null   int64  
 7   total_day_charge        3333 non-null   float64
 8   total_eve_minutes       3333 non-null   float64
 9   total_eve_calls         3333 non-null   int64  
 10  total_eve_charge        3333 non-null   float64
 11  total_night_minutes     3333 non-null   float64
 12  total_night_calls       3333 non-null   int64  
 13  total_night_charge      3333 non-null   float64
 14  total_intl_minutes      3333 non-null   float64
 15  total_intl_calls        3333 non-null   int64  
 16  total_intl_charge       3333 non-null   float64
 17  customer_service_calls  3333 non-null   int64  
 18  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(2)
memory usage: 498.0+ KB
```

In [176]:
```
# checking general df to see that both changes have been effected
df
```

Out[176]:

	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	t
phone_number									
3824657	128	415	no	yes	25	265.1	110	45.07	
3717191	107	415	no	yes	26	161.6	123	27.47	
3581921	137	415	no	no	0	243.4	114	41.38	
3759999	84	408	yes	no	0	299.4	71	50.90	
3306626	75	415	yes	no	0	166.7	113	28.34	
...	...	...	...	...	...	...	...	...	
4144276	192	415	no	yes	36	156.2	77	26.55	
3703271	68	415	no	no	0	231.1	57	39.29	
3288230	28	510	no	no	0	180.8	109	30.74	
3646381	184	510	yes	no	0	213.8	105	36.35	
4004344	74	415	no	yes	25	234.4	113	39.85	

3333 rows × 19 columns

We further need to review the `International Plan Column`

In [177]:
```
# Counting the occurrences of responses in this column
counts = df['international_plan'].value_counts()
counts
```

Out[177]:
```
no     3010
yes     323
Name: international_plan, dtype: int64
```

From above, there are only 'yes' and 'no' responses in this column with no any other unique entry. This means that information stored in this column is whether a client has an international plan or not. In that case, no need for further cleaning

Now lets look into the `Voice Mail Plan Column`. Given this column is of object type same as the international_plan column, we will repeat the same to confirm on unique entries and counts in this column

In [178]:
```
# Counting the occurrences of responses in this column
counts1 = df['voice_mail_plan'].value_counts()
counts1
```

Out[178]:
```
no     2411
yes     922
Name: voice_mail_plan, dtype: int64
```

From above, there are only 'yes' and 'no' responses in this column without any other unique entry. No need for cleaning cleaning

We then proceed to review the `Number_vmail_Messages`

Since we already checked and confirmed that there were no missing values in any of the columns. We just need to do a value_count check to confirm that all entries are valid. This helps us identify possibility of invalid data values such as `symbols` , `placeholder values` , and `punctuation marks` .

In [179]:
```
# looking at value_counts for this column
df.number_vmail_messages.value_counts()
```

Out[179]:
```
0     2411
31      60
29      53
28      51
33      46
27      44
30      44
24      42
26      41
32      41
25      37
23      36
36      34
22      32
35      32
39      30
34      29
37      29
21      28
38      25
20      22
19      19
40      16
42      15
17      14
16      13
41      13
43       9
15       9
18       7
44       7
14       7
45       6
12       6
46       4
13       4
47       3
50       2
9        2
8        2
11       2
48       2
49       1
4        1
10       1
51       1
Name: number_vmail_messages, dtype: int64
```

From Above, all entries are valid and the column entries are good to go with without further cleaning.

Our next stop is the `Total_Day_Minutes` column, which corresponds to the average minutes clients spends in day on average.

Having confirmed no missing value, in the df, we will look at the value_count of all unqiue entries in this column to check for any anormalies

In [180]:
```
# checking for total entry per unique item in the total_day_minutes column
df.total_day_minutes.value_counts()
```

Out[180]:
```
154.0    8
159.5    8
174.5    8
183.4    7
175.4    7
        ..
78.6     1
200.9    1
254.3    1
247.0    1
180.8    1
Name: total_day_minutes, Length: 1667, dtype: int64
```

No presence of unexpected entry and with dtype as int64, this column does not need any cleaning.

For all the items with dtype as int64 and floating points, since they represent numerical values and the dataframe has indentified them as so, it is okay to leave the individual cleaning, as any entry of any number if valid.

We will move to the last `Churn Column` , which will be our target variable and check for any anormalies.

In [181]:
```
#reviewing the churn column
df.churn.value_counts()
```

Out[181]:
```
False    2850
True      483
Name: churn, dtype: int64
```

The column does not appear to have any missing values. As we can see, there are `2850 false values`, which indicates the number of clients who did not churn. There are also `483 true values`, showing the number of clients who left the the company.

Exploratory Data Analysis

In this section, we are going to conduct a comprehensive exploration of the data through `univariate`, `bivariate`, and `multivariate analysis`.

The reason for this type of data exploration is to identify possible correlations among the features and distribution of variables, which will be important in feature engineering and modelling.

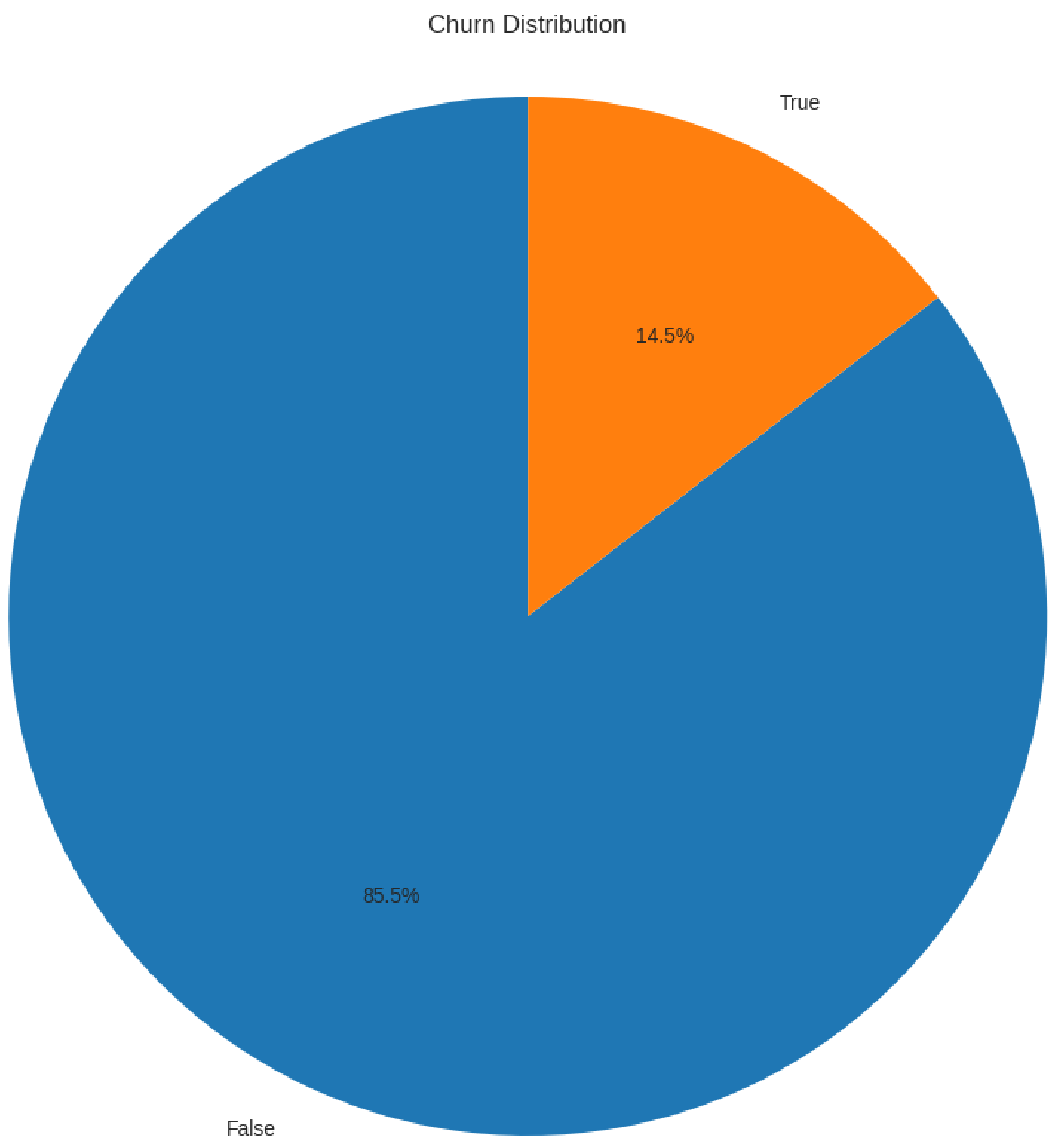
Univariate Analysis

`Univariate data analysis` involves analyzing a single variable. In the context of our project, this will involve examining the distribution of each feature in the dataset to understand its characteristics and identify any potential issues such as outliers.

We start with the target variable column **churn** to identify its distribution. This categorical variable with boolean values `True` and `False`, indicating whether the client will probably churn or not.

First, we visualize the distribution of data in this column using a pie chart

In [182]:
```
# representing the same using a Pie Chart to visualize the percentages
churn_counts = df['churn'].value_counts()

# Create a new figure with a larger size
plt.figure(figsize=(15, 10))

# Create a pie chart
plt.pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Churn Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```

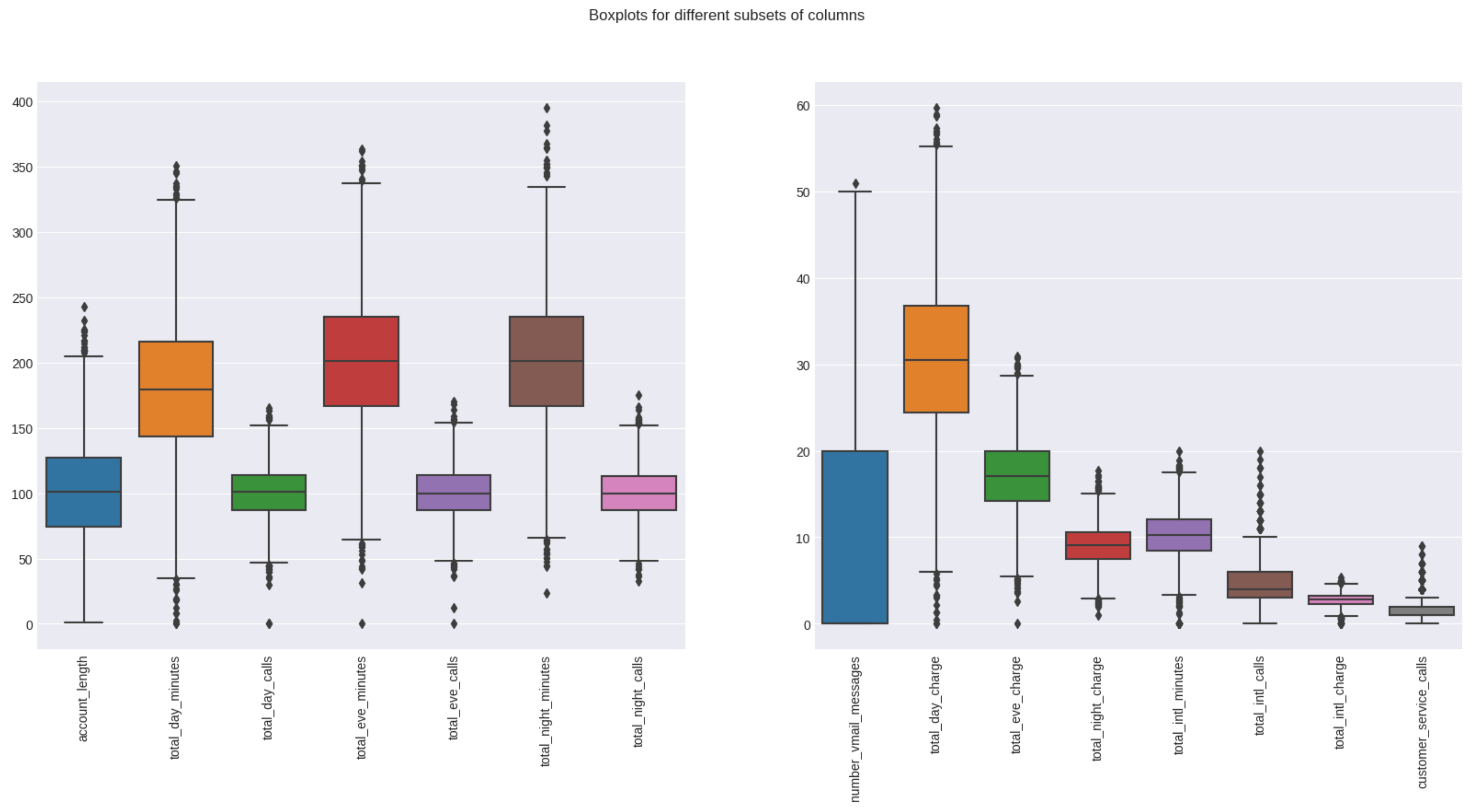


Of the 3,333 customers in the dataset, 483 have terminated their contract with the Telecom firm. That is 14.5% of customers lost.

The distribution of the binary classes shows a data imbalance. This needs to be addressed before modeling as an unbalanced feature can cause the model to make false predictions.

Further, will further review the data to identify outliers, which is crucial to understanding the distribution of values for different columns. For this, our focus is on numeric data. Outliers can significantly impact the performance of machine learning models, which will impacts the feature engineering process.

In [183]:
```python
#Checking for outliers in the data
# List of columns for the first boxplot
cols1 = ['account_length','total_day_minutes','total_day_calls',
                'total_eve_minutes','total_eve_calls','total_night_minutes','total_night_calls']

# List of columns for the second boxplot
cols2 = ['number_vmail_messages', 'total_day_charge', 'total_eve_charge', 'total_night_charge', 'total_intl_minutes', 'total_int

# Create a figure with one row and two columns
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

# Create a boxplot for the first subset of columns in the first column
sns.boxplot(data=df[cols1], ax=axes[0])
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)

# Create a boxplot for the second subset of columns in the second column
sns.boxplot(data=df[cols2], ax=axes[1])
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=90)

#setting the figure title
fig.suptitle('Boxplots for different subsets of columns')

# Show the plot
plt.show()
```



We used two separate boxplots because of the significant difference in scale between the columns. In box boxplots, we can see that the columns have numerous outliers, which may affect the performance of machine learning models such as k-nearest neighbors (knn).

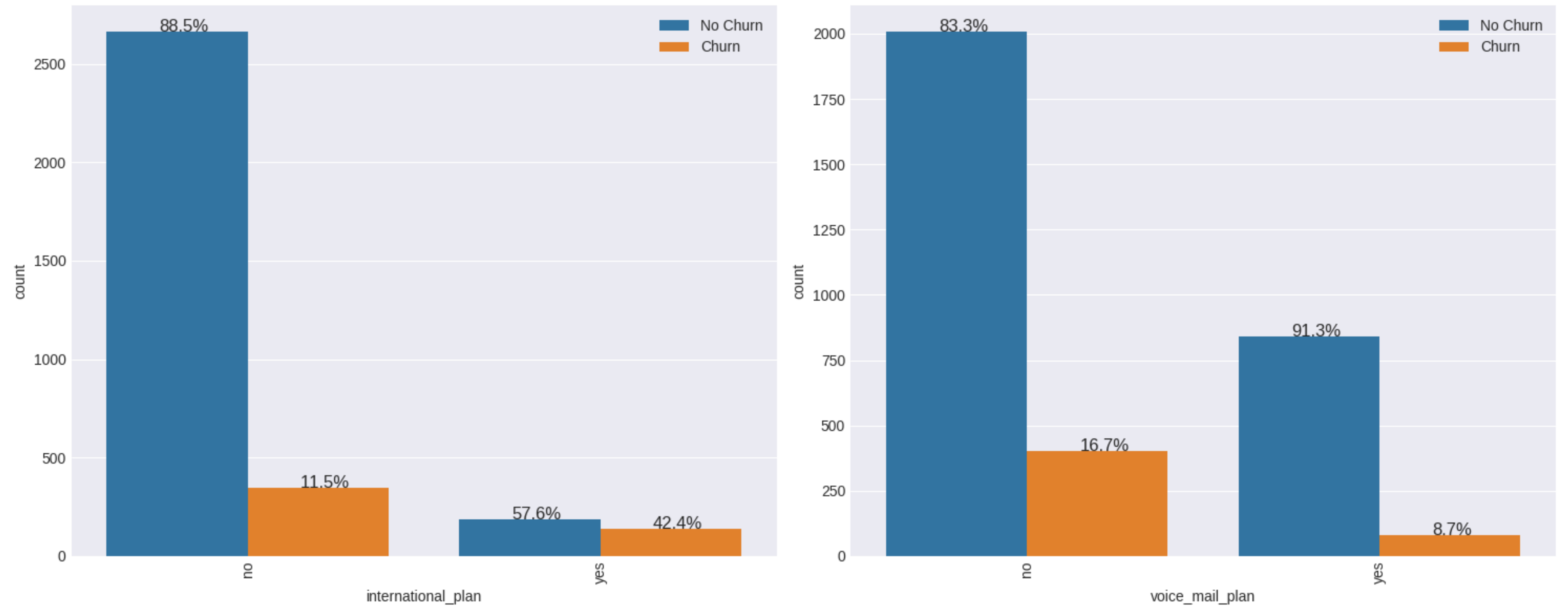
As for our data, all these outliers contain valuable information, which will be very important to our models.

Bivariate Analysis

Bivariate analysis involves analyzing the relationship between two variables. For our project, we examine the relationship between each feature and the target variable (customer churn) to understand how they are related.

Here, we are doing some analysis of the customer churning in relation to state, area code, international plan, and voice mail plan. We are trying to understand whether there are correlations between the categorical columns and the customer churning rate.

In [184]:
```
categoric_cols = ['international_plan','voice_mail_plan']

fig, axes = plt.subplots(nrows=1, ncols=len(categoric_cols), figsize=(15, 6))

for i, col in enumerate(categoric_cols):
    ax = sns.countplot(x=col, hue="churn", data=df, order=df[col].value_counts().iloc[0:15].index, ax=axes[i])
    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=90)
    handles, labels = axes[i].get_legend_handles_labels()
    axes[i].legend(handles, ['No Churn', 'Churn'], loc="upper right")

    # Calculate the total number of observations within each group
    totals = df.groupby(col)["churn"].count().values

    # Iterate over the rectangles in the plot
    for j, p in enumerate(ax.patches):
        # Calculate the percentage of observations in each group
        percentage = '{:.1f}%'.format(100 * p.get_height()/totals[j % 2])
        # Add text annotations with the calculated percentages
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size=12)

plt.tight_layout()
plt.show()
```

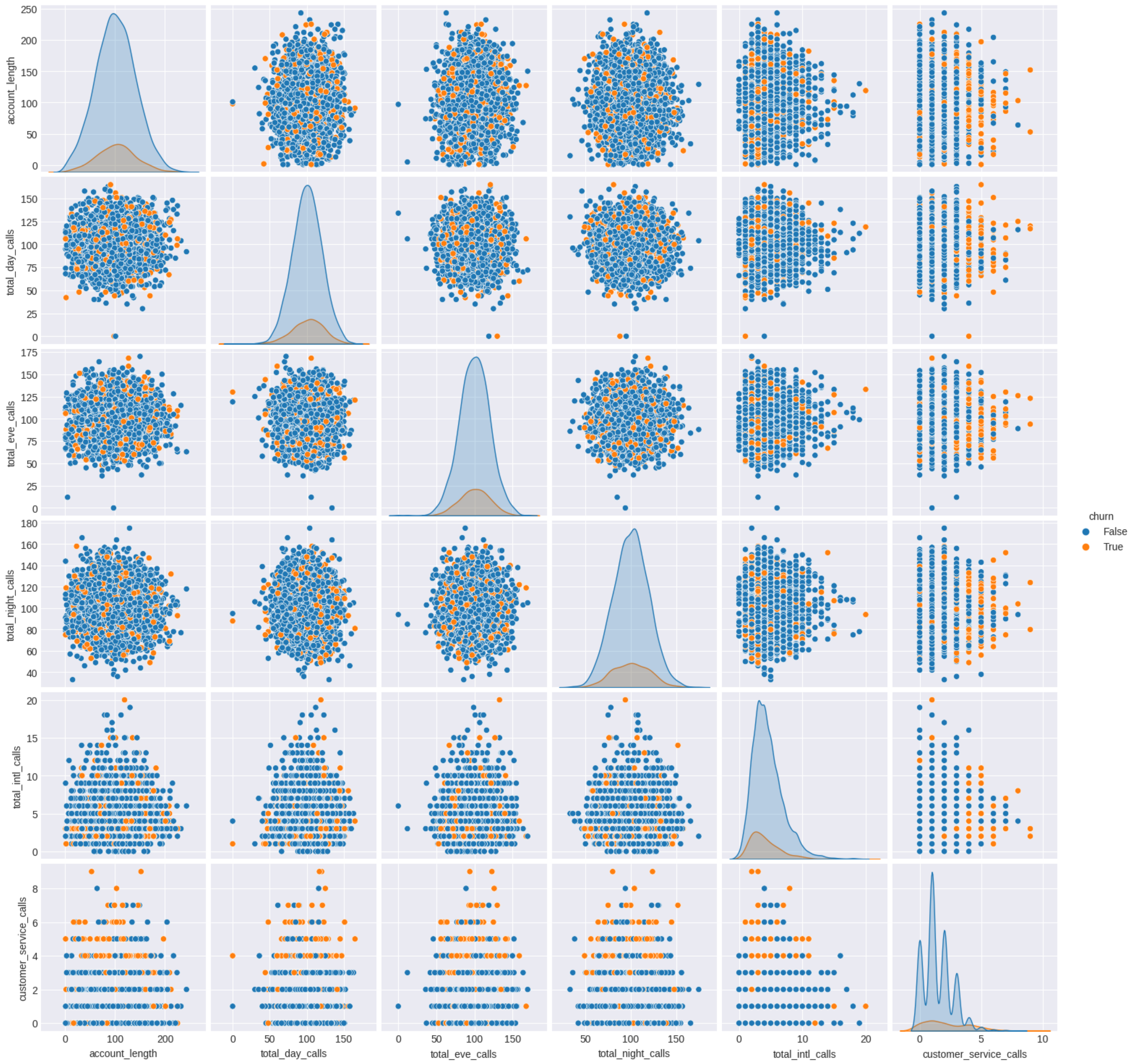


For the international plan, a higher proportion of customers who subscribed to the plan churned `(42.4%)` compared to those who did not subscribe `(11.5%)`. This suggests that subscribing to the international plan may be associated with a higher likelihood of churning.

For the voice mail plan, a lower proportion of customers who subscribed to the plan churned `(8.7%)` compared to those who did not subscribe `(16.7%)`. This suggests that subscribing to the voice mail plan may be associated with a lower likelihood of churning.

Next, we visualize the correlations between `different features` and `customer churning`. Here, we are trying to understand how each feature might be contributing to customer churning. We use `pairplots` for this case!

In [185]:
```
#plotting pairplots for numeric variables
data_temp = df[["account_length","total_day_calls","total_eve_calls","total_night_calls",
              "total_intl_calls","customer_service_calls","churn"]]
sns.pairplot(data_temp, hue="churn",height=2.5);
plt.show();
```



There seems to be strong relationship between customer service calls and true churn values. After 4 calls, customers are a lot more likely to discontinue their service.

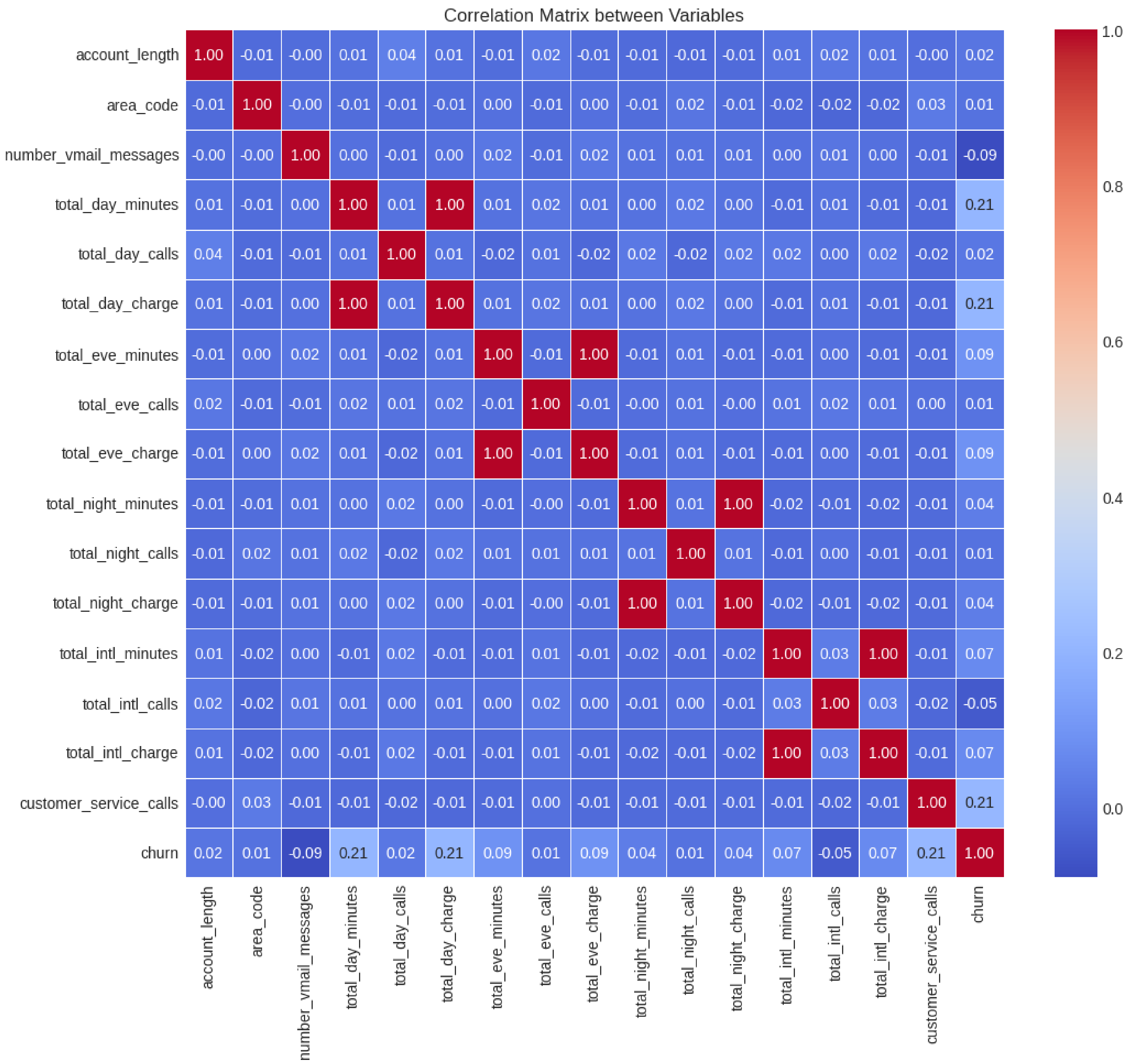
Multi-variate Analysis

Multivariate analysis involves analyzing the relationship between multiple variables simultaneously. In this case, we explore the relationship between multiple features and the target variable (customer churn) to understand how they are related when considered together.

We used a correlation matrix to identify the correlation between different variables in the dataset.

```
In [186]: # Calculate the correlation matrix
corr_matrix = df.corr()

# Generate the correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix between Variables')
plt.show();
```

```
<ipython-input-186-6f8828c21d75>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a futu
re version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr_matrix = df.corr()
```



Through the correlation matrix, we have identified that `total international charge` has a perfect correlation with `total international minutes`, indicating multicollinearity. These two features appears to be independent, which means we can only use one when creating the model.

Other features identified as significantly correlated to the target variable are `total minutes`, `total day charge`, and `customer service calls`.

Basic Data Preprocessing

In this section, we proprocess the data to prepare it for modelling. In the dataset, we have categorical and numeric data columns, some of which must be tranformed into a datatype acceptable by the different machine learning models used in the modelling section.

A good example would be using one-hot encoding to transform categorical columns with object datatypes to numerical ones, especially 1s and 0s

The dataset must also be split into different sets, the training and testing sets. We will use the training set to train the different models and evaluate the performance using the test data. Cross-validation is used.

We also drop features that have minimal or no effect on the target variables using ridge or lasso regression. We may also identify other frameworks for choosing the best features.

Feature Engineer -> Split -> Standardize

Step 1: Transform columns to numeric

In [187]:
```
#convert churn values to integer 1s and 0s
df['churn'] = df['churn'].astype(int)

#convert area_code, international plan, and voice_mail_plan to integers 1s and 0s
df = pd.get_dummies(df, columns=['area_code', 'international_plan', 'voice_mail_plan'])
```

In [188]:
```
#displace the first 10 records
df.head(7)
```

Out[188]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	total_eve_cha
phone_number								
3824657	128	25	265.1	110	45.07	197.4	99	16
3717191	107	26	161.6	123	27.47	195.5	103	16
3581921	137	0	243.4	114	41.38	121.2	110	10
3759999	84	0	299.4	71	50.90	61.9	88	5
3306626	75	0	166.7	113	28.34	148.3	122	12
3918027	118	0	223.4	98	37.98	220.6	101	18
3559993	121	24	218.2	88	37.09	348.5	108	29

7 rows × 23 columns

We separate the target variable from the features, standardize the features, and address class imbalance in the target variable.

Step 2: Separate features and target variable

In [189]:
```
# Separating features from the target variable
y = df['churn']
X = df.drop('churn', axis=1)
```

Step 3: Conduct a Train-test-split on the data

In [190]:
```
#split the data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Creating Our Models

We `create several models`, `evaluate them`, then do some `hyper-parameter tuning` to try and improve the models. Our intention in this case is to find the model and parameters that perform the best.

We train and evaluate the following models:

- Logistic Regression Model,
- K-Nearest Neighbors,
- Decision Trees, and
- Random Forests.

Model 1: Logistic Regression Model

Our first model is `Logistic Regression Model`. Logistic regression is a type of generalized linear model that can be used to predict the probability of a binary outcome, such as whether a customer will `churn or not`.

In our case, we use `logistic regression` to model the relationship between the `our features` and the `likelihood of a customer churning`.

In [191]:
```python
# Create a pipeline for preprocessing (only standardization, as there are no categorical columns)
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), X.columns)  # Apply standardization to all numerical columns
    ]
)

# Initialize the logistic regression model
logreg_model = LogisticRegression()

# Create a pipeline that includes preprocessing and the logistic regression model
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', logreg_model)
])

# Fit the model on the training data
model_pipeline.fit(X_train, y_train)

# Predict churn for the train and test data
y_train_pred = model_pipeline.predict(X_train)
y_test_pred = model_pipeline.predict(X_test)

# Calculate the accuracy of the model for train and test data
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Print the train and test scores
print(f"Train Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Print the classification report for test data
print("Classification Report (Test Data):")
print(classification_report(y_test, y_test_pred))

# Print the confusion matrix for test data
print("Confusion Matrix (Test Data):")
print(confusion_matrix(y_test, y_test_pred))
```

```
Train Accuracy: 0.86
Test Accuracy: 0.86
Classification Report (Test Data):
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       566
           1       0.60      0.18      0.27       101

    accuracy                           0.86       667
   macro avg       0.73      0.58      0.60       667
weighted avg       0.83      0.86      0.82       667

Confusion Matrix (Test Data):
[[554  12]
 [ 83  18]]
```

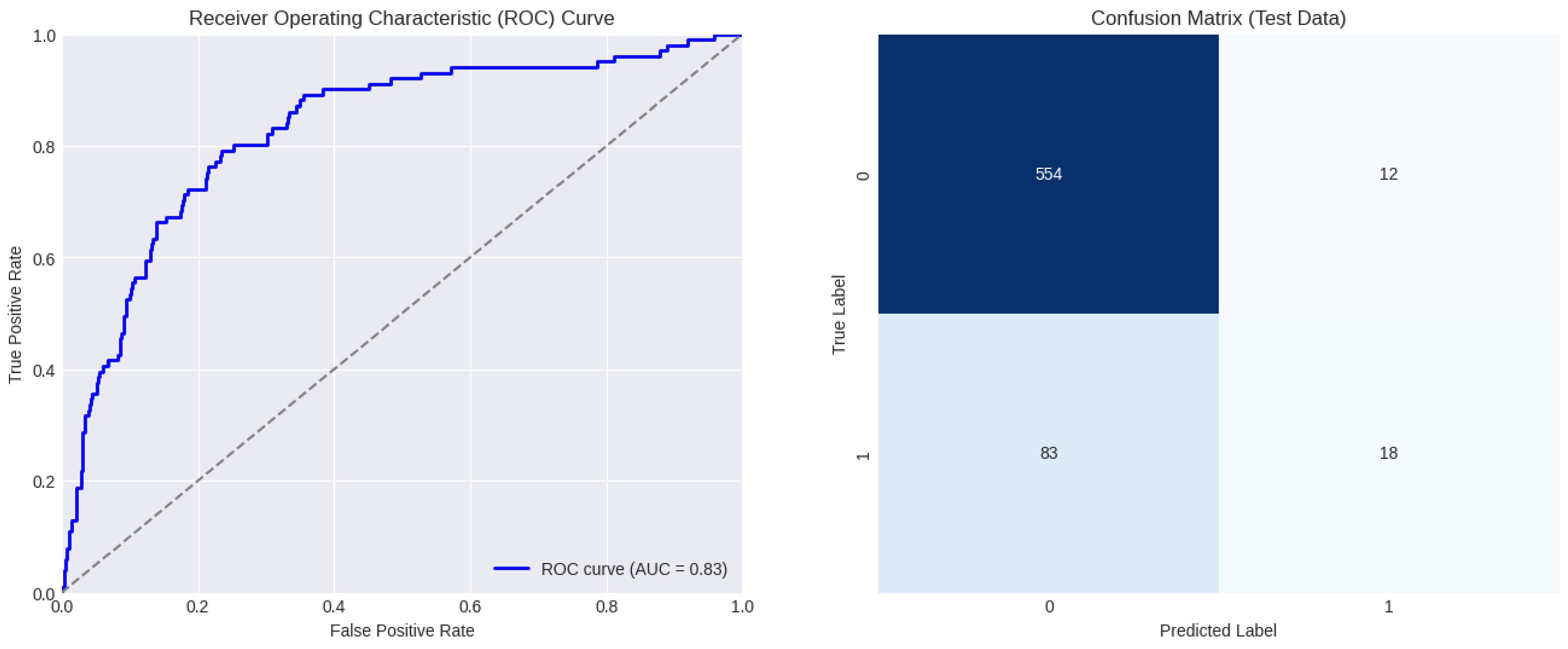
Comments and notes on model Accuracy: The accuracy of the model is `86%` Train Accuracy: `0.86` Test Accuracy: `0.86`

Classification Report:

- **Precision**: The precision for class 0 (not churned) is `87%`. The precision for class 1 (churned) is `60%`
- **Recall**: The recall for class 0 (not churned) is `98%` but the recall for class 1 (churned) is only `18%`.
- **F1-score**: The F1-score for class 0 (not churned) is `92%` and for class 1 (churned) is only `27%`. The F1-score for class 1 is low due to the low recall.

We further plot the `ROC Curve (Receiver Operating Characteristic curve)`, `the AUC (Area Under the Curve)`, and `Confusion matrix` to visualize the results

In [192]:
```python
# Plot the ROC curve for test data
y_prob = model_pipeline.predict_proba(X_test)[:, 1]  # Probability of positive class (churned)
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

ax1.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
ax1.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.0])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC) Curve')
ax1.legend(loc="lower right")

# Plot the confusion matrix as a heatmap for test data
confusion_mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap='Blues', cbar=False, ax=ax2)
ax2.set_xlabel('Predicted Label')
ax2.set_ylabel('True Label')
ax2.set_title('Confusion Matrix (Test Data)')

plt.show()
```



Confusion Matrix:

- The confusion matrix shows a total of `667 samples` in the test set.
- True Positives (TP): The model correctly predicted `18 samples` as Not churned (class 0).
- True Negatives (TN): The model correctly predicted `554 samples` as churned (class 1).
- False Positives (FP): The model incorrectly predicted `12 samples` as churned when they were not churned.
- False Negatives (FN): The model incorrectly predicted `83 samples` as not churned when they were churned.

The ROC curve & The AUC

They provide a measure of how well the model can distinguish between positive and negative samples. A model with an AUC of `1 is perfect`, while an `AUC of 0.5` indicates that the model is no better than random guessing.

- `AUC = 0.5` : The model's performance is equivalent to random guessing, and it is not useful for classification.
- `AUC > 0.5` : The model performs better than random guessing, and the higher the AUC, the better the model's discriminatory power.
- `AUC = 1` : The model perfectly distinguishes between positive and negative samples, making it an excellent classifier.

In our case, the `AUC is 0.83` , which is greater than `0.5` and closer to `1` . This indicates that the logistic regression model has reasonable discriminatory power in distinguishing between churned and not churned samples. An AUC of `0.83` suggests that the model has a good ability to rank the predictions, and it performs significantly better than random guessing.

Interpretation:

- The model performs well in predicting the negative class (not churned) as evidenced by high accuracy, precision, and recall for class 0.
- However, it performs poorly for the positive class (churned) as indicated by the low values for precision, recall, and F1-score for class `1` .

In other words, the model is missing a substantial number of customers who are actually churned, leading to false negatives. It is failing to correctly identify those customers who have churned

This model though better than guessing can have serious implications to the business as it fails to predict churned customers on a significant level

MODEL 1.2 LOGISTIC MODEL ADDRESSING CLASS IMBALANCE

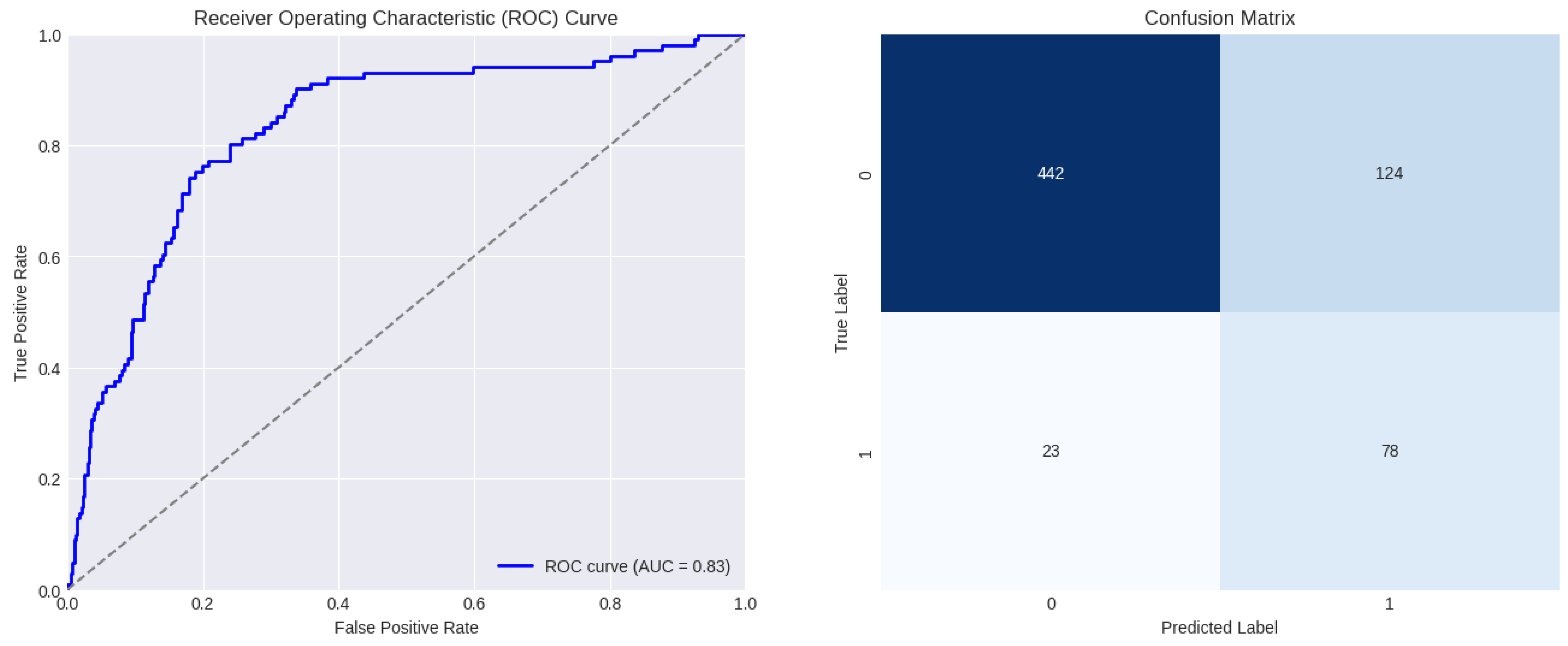
Trying to Adjust the model to adjust for class imbalance in the target variable to see if there are any improvements

In [193]:
```python
# Create a pipeline for preprocessing (only standardization, as there are no categorical columns)
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), X.columns)  # Apply standardization to all numerical columns
    ]
)

# Initialize the logistic regression model with class_weight parameter
logistic_reg_model2 = LogisticRegression(class_weight='balanced')

# Create a pipeline that includes preprocessing and the logistic regression model
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', logistic_reg_model2)
])

# Fit the model on the training data
model_pipeline.fit(X_train, y_train)

# Predict churn for the test data
y_pred = model_pipeline.predict(X_test)

# Calculate the accuracy of the model on train and test data
train_accuracy = model_pipeline.score(X_train, y_train)
test_accuracy = model_pipeline.score(X_test, y_test)

print(f"Train Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Train Accuracy: 0.77
Test Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.78      0.86       566
           1       0.39      0.77      0.51       101

    accuracy                           0.78       667
   macro avg       0.67      0.78      0.69       667
weighted avg       0.87      0.78      0.81       667

Confusion Matrix:
[[442 124]
 [ 23  78]]
```

REBALANCED LOGISTIC MODEL INTEPRETATIONS Train Accuracy: 0.77 compared to previous model 0.86 Test Accuracy: 0.78 compared to the previous 0.86 Classification Report: precision class 0 0.95 compared to previous 0.87 precision class 1 0.39 compared to 0.60 recall class 0 0.78 compared to 0.98 recall class 1 0.77 compared to 0.18 f1score class 0 0.86 compared to 0.92 f1score class 1 0.51 compared to 0.27

In [194]:
```
# Plot the ROC curve
y_prob = model_pipeline.predict_proba(X_test)[:, 1]  # Probability of positive class (churned)
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

ax1.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
ax1.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.0])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC) Curve')
ax1.legend(loc="lower right")

# Plot the confusion matrix as a heatmap
confusion_mat = confusion_matrix(y_test, y_pred)
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap='Blues', cbar=False, ax=ax2)
ax2.set_xlabel('Predicted Label')
ax2.set_ylabel('True Label')
ax2.set_title('Confusion Matrix')

plt.show()
```



confusion Matrix: [[436 134][26 71]] compared to previous [[554 12] [83 18]] [26 71]]

Interpreting the classification report and confusion matrix:

1. Train Accuracy: 0.77 Test Accuracy: 0.76

The model achieved an accuracy of 77% on the training data and 76% on the test data. This means that the model is performing relatively well on the unseen test data, which indicates that it is not overfitting.

2. Classification Report:

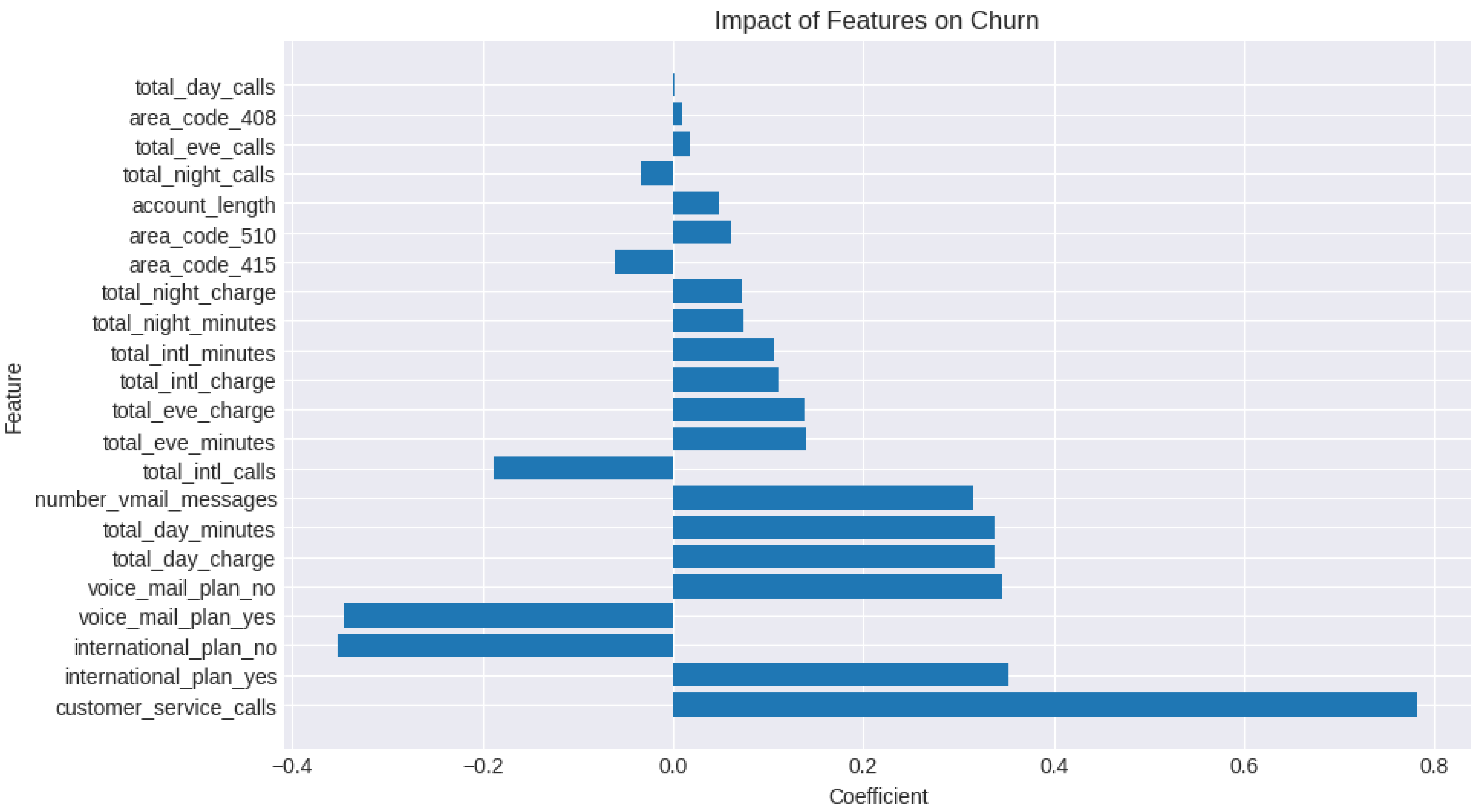
   - Precision: For class 0 (not churned), the precision is 94%, meaning that when the model predicts a customer won't churn, it is correct 94% of the time. For class 1 (churned), the precision is only 35%, indicating that when the model predicts a customer will churn, it is correct only 35% of the time.
   - Recall:For class 0 (not churned), the recall is 76%, indicating that the model correctly identifies 76% of the actual non-churned customers. For class 1 (churned), the recall is 73%, meaning that the model captures 73% of the actual churned customers.
   - F1-score: The F1-score is the harmonic mean of precision and recall and provides a balance between the two. For class 0, the F1-score is 84%, and for class 1, it is 47%.
   - Support: The number of occurrences of each class in the test set. For class 0, there are 570 instances, and for class 1, there are 97 instances.
3. Confusion Matrix: The confusion matrix provides a detailed breakdown of the model's performance in predicting each class.

   - True Negative (TN): 436 - The number of correctly predicted non-churned customers.
   - False Positive (FP): 134 - The number of non-churned customers incorrectly classified as churned.
   - False Negative (FN): 26 - The number of churned customers incorrectly classified as non-churned.
   - True Positive (TP): 71 - The number of correctly predicted churned customers.
4. ROC curve (AUC = 0.81): An AUC (Area Under the Curve) value of 0.81 indicates that the model has good discriminative power and is reasonably effective at distinguishing between the two classes.

In summary, the model seems to perform well in predicting non-churned customers (class 0) with high precision and recall. However, its performance on predicting minority class (churned customers) (class 1) is not as good, with relatively lower precision and recall.

There is a slight improvement on the previous model in predicting the churned customers comparing to guess work but the model is still not great

In [195]:
```python
# Get the coefficients of the logistic regression model2
coefficients = model_pipeline.named_steps['classifier'].coef_[0]

# Create a DataFrame to display the coefficients along with the corresponding feature names
coefficients_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': coefficients})

# Sort the DataFrame by absolute coefficient values to see the most impactful features
coefficients_df['Abs_Coefficient'] = np.abs(coefficients_df['Coefficient'])
coefficients_df = coefficients_df.sort_values(by='Abs_Coefficient', ascending=False)

# Plot the coefficients
plt.figure(figsize=(10, 6))
plt.barh(coefficients_df['Feature'], coefficients_df['Coefficient'])
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.title('Impact of Features on Churn')
plt.show()
```



In [196]:
```
#see the actual coefficient arranged in descending ordger
coefficients_df
```

Out[196]:

	Feature	Coefficient	Abs_Coefficient
14	customer_service_calls	0.781643	0.781643
19	international_plan_yes	0.351737	0.351737
18	international_plan_no	-0.351737	0.351737
21	voice_mail_plan_yes	-0.346152	0.346152
20	voice_mail_plan_no	0.346152	0.346152
4	total_day_charge	0.337942	0.337942
2	total_day_minutes	0.337680	0.337680
1	number_vmail_messages	0.314547	0.314547
12	total_intl_calls	-0.187955	0.187955
5	total_eve_minutes	0.139153	0.139153
7	total_eve_charge	0.137978	0.137978
13	total_intl_charge	0.110921	0.110921
11	total_intl_minutes	0.106664	0.106664
8	total_night_minutes	0.074404	0.074404
10	total_night_charge	0.072972	0.072972
16	area_code_415	-0.061263	0.061263
17	area_code_510	0.060865	0.060865
0	account_length	0.048723	0.048723
9	total_night_calls	-0.033360	0.033360
6	total_eve_calls	0.017118	0.017118
15	area_code_408	0.009950	0.009950
3	total_day_calls	0.001019	0.001019

Coefficients and their absolute values for each feature from the logistic regression model.

These coefficients provide insights into the impact of each feature on the likelihood of churn (negative/undesired impact) or not churn (positive/desired impact).

In summary Positive coefficients indicate features that increase the likelihood of churn, while negative coefficients indicate features that decrease the likelihood of churn.

By understanding these effects, you can identify important features that contribute to customer churn and potentially take actions to reduce churn and retain valuable customers.

- **customer_service_calls**: This feature has the highest positive impact on churn. An increase in the number of customer service calls is associated with a higher likelihood of churn.
- **total_day_charge and total_day_minutes:** Both features have a positive impact on churn. An increase in total day charge or total day minutes is associated with a higher likelihood of churn.
- **voice_mail_plan_yes and voice_mail_plan_no:** These binary features are related to the presence or absence of a voice mail plan. voice_mail_plan_yes has a negative impact on churn, meaning customers with a voice mail plan are less likely to churn, while voice_mail_plan_no has a positive impact, meaning customers without a voice mail plan are more likely to churn.
- **international_plan_yes and international_plan_no**: Similar to the voice mail plan features, international_plan_yes has a positive impact on churn, meaning customers with an international plan are more likely to churn, while international_plan_no has a negative impact, meaning customers without an international plan are less likely to churn.
- total_intl_calls: This feature has a negative impact on churn. An increase in the number of international calls is associated with a lower likelihood of churn.
- **number_vmail_messages**: This feature has a positive impact on churn. An increase in the number of voice mail messages is associated with a higher likelihood of churn.
- **total_eve_minutes and total_intl_charge**: These features have a positive impact on churn. An increase in total evening minutes or total international charge is associated with a higher likelihood of churn.
- **total_eve_charge, total_intl_minutes, total_night_minutes, and total_night_charge:** These features also have a positive impact on churn. An increase in the respective charges and minutes is associated with a higher likelihood of churn.
- **area_code_510, area_code_415, and area_code_408**: These binary features represent different area codes. area_code_510 has a positive impact on churn, while area_code_415 and area_code_408 have negative impacts. This suggests that customers from area_code_510 are more likely to churn compared to customers from the other two area codes.
- **account_length and phone_number**: These features have relatively smaller impacts on churn, but both have positive coefficients, indicating a higher account length or phone number is associated with a slightly higher likelihood of churn.
- **total_day_calls and total_night_calls**: These features have relatively smaller impacts on churn, and they both have negative coefficients, indicating a higher number of day or night calls is associated with a slightly lower likelihood of churn.

Model 2: K-Nearest Neighbors

Baseline Model: Here we build the first K-Nearest Neighbors model with default parameters

In [197]:
```
#instantiate the standard scaler
scaler = StandardScaler()

#fit and transform the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5,  weights='uniform')

# Train the classifier on the training data
knn.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred = knn.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_knn = accuracy_score(y_test, y_pred)
precision_knn = precision_score(y_test, y_pred)
recall_knn = recall_score(y_test, y_pred)
f1_knn = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy_knn)
print("Precision:", precision_knn)
print("Recall:", recall_knn)
print("F1-score:", f1_knn)

#Calculate train and test scores
train_score = knn.score(X_train_scaled, y_train)
test_score = knn.score(X_test_scaled, y_test)

print(train_score)
print(test_score)
```

```
Accuracy: 0.8740629685157422
Precision: 0.7741935483870968
Recall: 0.2376237623762376
F1-score: 0.3636363636363636
0.9103525881470368
0.8740629685157422
```

Our KNN model has an accuracy of `0.874` on the test set, which means that it correctly classifies `87.4%` of the test data. The precision of our model is `0.774` , which means that when our model predicts that a customer will churn, it is correct `77.4%` of the time. The recall of our model is `0.238` , which means that our model correctly identifies 23.8% of all customers who actually churned. The F1-score, which is the harmonic mean of precision and recall, is `0.364` .

Our train score and test score are both measures of how well our model fits the data. Our train score is `0.910` , which means that our model correctly classifies `91%` of the training data. Our test score is `0.874` , which is slightly lower than our train score but still indicates good performance on unseen data.

Overall, these results suggest that our KNN model is performing well in terms of accuracy and precision but could be improved in terms of recall.

Grid search and hyperparameter tuning

Here we were looking to find the best parameters for the model. The n_neighbors(K value) parameter, weight and distance metric to use. We also did cross validation to avoid overfitting.

In [198]:
```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search through
param_grid = {
    'n_neighbors': [3, 5, 7, 9],        # Number of neighbors to consider
    'weights': ['uniform', 'distance'], # Weight function used in prediction
    'p': [1, 2]                         # Power parameter for Minkowski distance
}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Create GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV to the scaled training data
grid_search.fit(X_train_scaled, y_train)

# Get the best hyperparameters found by GridSearch
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Create a new KNN classifier with the best hyperparameters
best_knn = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'],
                                weights=best_params['weights'],
                                p=best_params['p'])

# Train the best KNN classifier on the training data
best_knn.fit(X_train_scaled, y_train)

# Make predictions on the testing data using the best KNN classifier
y_pred_best = best_knn.predict(X_test_scaled)

# Evaluate the best KNN model's performance
accuracy_best_knn = accuracy_score(y_test, y_pred_best)
precision_best_knn = precision_score(y_test, y_pred_best)
recall_best_knn = recall_score(y_test, y_pred_best)
f1_best_knn = f1_score(y_test, y_pred_best)

# Print the evaluation metrics of the best KNN model
print("\nBest KNN Model Performance:")
print("Accuracy:", accuracy_best_knn)
print("Precision:", precision_best_knn)
print("Recall:", recall_best_knn)
print("F1-score:", f1_best_knn)

#Calculate train and test scores
best_knn_train_score = best_knn.score(X_train_scaled, y_train)
best_knn_test_score = best_knn.score(X_test_scaled, y_test)

print(best_knn_train_score)
print(best_knn_test_score)
```

```
Best Hyperparameters: {'n_neighbors': 7, 'p': 1, 'weights': 'distance'}

Best KNN Model Performance:
Accuracy: 0.8845577211394303
Precision: 0.8
Recall: 0.31683168316831684
F1-score: 0.45390070921985815
1.0
0.8845577211394303
```

The best hyperparameters for our KNN model are `n_neighbors=7`, `p=1`, and `weights='distance'`. With these hyperparameters, our KNN model has an accuracy of `0.885` on the test set, which means that it correctly classifies `88.5%` of the test data. The precision of our model is `0.8`, which means that when our model predicts that a customer will churn, it is correct `80%` of the time. The recall of our model is `0.317`, which means that our model correctly identifies `31.7%` of all customers who actually churned. The F1-score, which is the harmonic mean of precision and recall, is `0.454`.

Our training score and test score are both measures of how well our model fits the data. Our training score is `1.0`, which means that our model correctly classifies `100%` of the training data. This means that there is overfitting. Our test score is `0.885`, which is slightly lower than our training score but still indicates good performance on unseen data.

The results suggest that our KNN model with the best hyperparameters is performing well in terms of accuracy and precision and has improved in terms of recall compared to the previous KNN model.

Ensemble Methods:

- To improve the KNN models, we combined multiple KNN models by using the ensemble technique Bagging to create a more robust and accurate classifier.

In [199]:
```
# Building an ensemble KNN model using Bagging
from sklearn.ensemble import BaggingClassifier

# Instantiate the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Instantiate the BaggingClassifier with KNN as the base estimator
bagging_knn = BaggingClassifier(base_estimator=knn, n_estimators=10, random_state=42)

# Train the ensemble model on the training data
bagging_knn.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred_bagging = bagging_knn.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_bagging_knn = accuracy_score(y_test, y_pred_bagging)
precision_bagging_knn = precision_score(y_test, y_pred_bagging)
recall_bagging_knn = recall_score(y_test, y_pred_bagging)
f1_bagging_knn = f1_score(y_test, y_pred_bagging)

# Print the evaluation metrics
print("Accuracy:", accuracy_bagging_knn)
print("Precision:", precision_bagging_knn)
print("Recall:", recall_bagging_knn)
print("F1-score:", f1_bagging_knn)

#Calculate train and test scores
bagging_knn_train_score = bagging_knn.score(X_train_scaled, y_train)
bagging_knn_test_score = bagging_knn.score(X_test_scaled, y_test)

print(bagging_knn_train_score)
print(bagging_knn_test_score)
```

```
Accuracy: 0.8800599700149925
Precision: 0.8
Recall: 0.27722772277227725
F1-score: 0.411764705882353

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimato
r` in version 1.2 and will be removed in 1.4.
  warnings.warn(

0.9122280570142536
0.8800599700149925
```

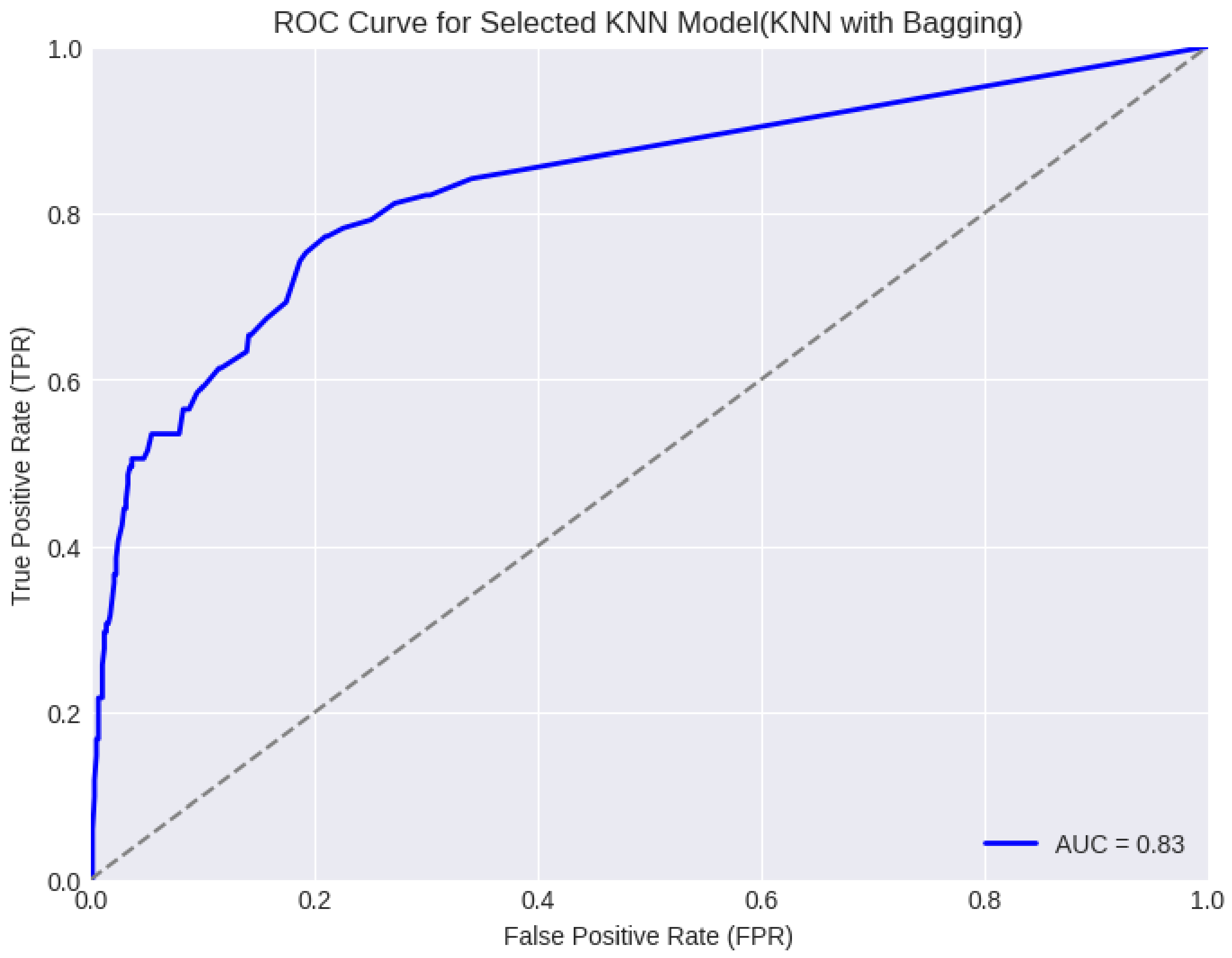
The Bagging Classifier ensemble model we have trained has an accuracy of `0.880` on the test set, which means that it correctly classifies `88%` of the test data. The precision of our model is `0.8` , which means that when our model predicts that a customer will churn, it is correct `80%` of the time. The recall of our model is `0.277` , which means that our model correctly identifies `27.7%` of all customers who actually churned. The F1-score, which is the harmonic mean of precision and recall, is 0.412.

Our training score and test score are both measures of how well our model fits the data. Our training score is 0.912, which means that our model correctly classifies `91.2%` of the training data. Our test score is `0.880` , which is slightly lower than our training score but still indicates good performance on unseen data.

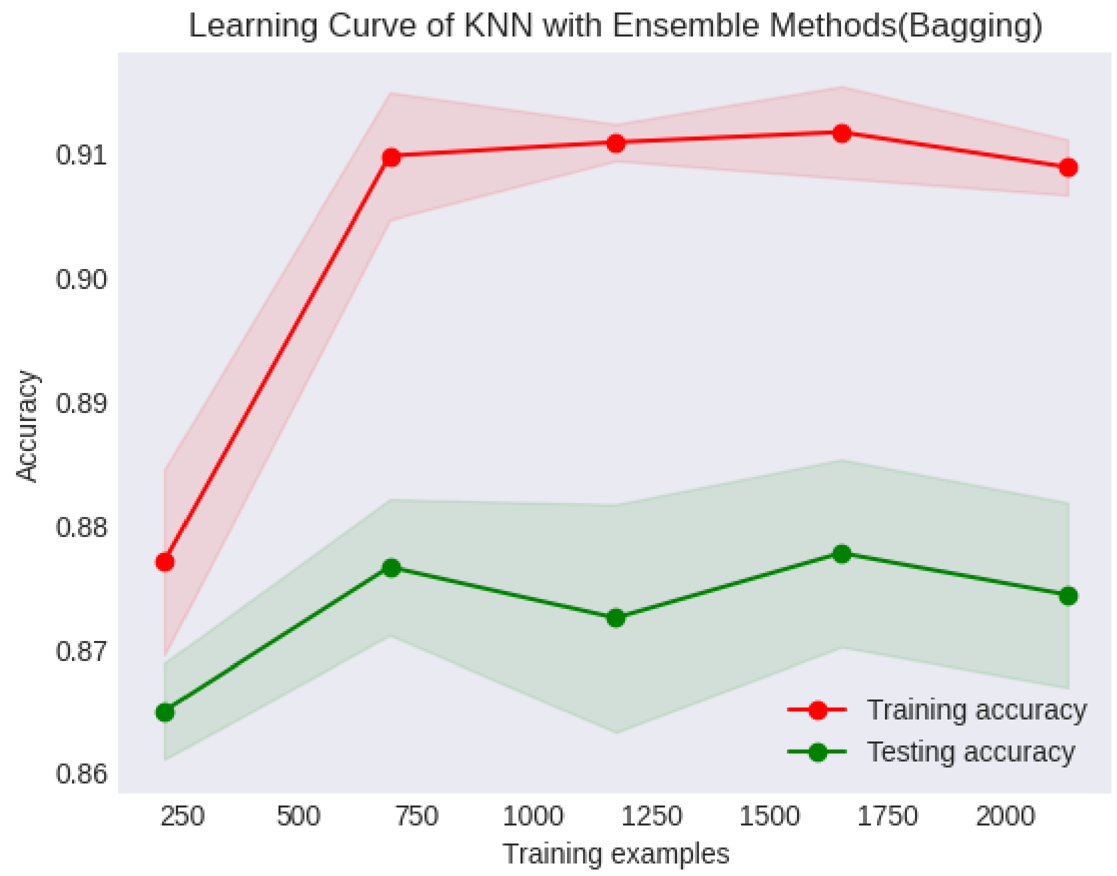
In that case, the results suggest that our Bagging Classifier ensemble model is performing well in terms of accuracy and precision but could be improved in terms of recall.

`KNN model 2: Bagging Classifier ensemble model` is the best performing model of the three KNN models because it does not overfit and it has a higher `Accuracy` , `Precision and F1-score`

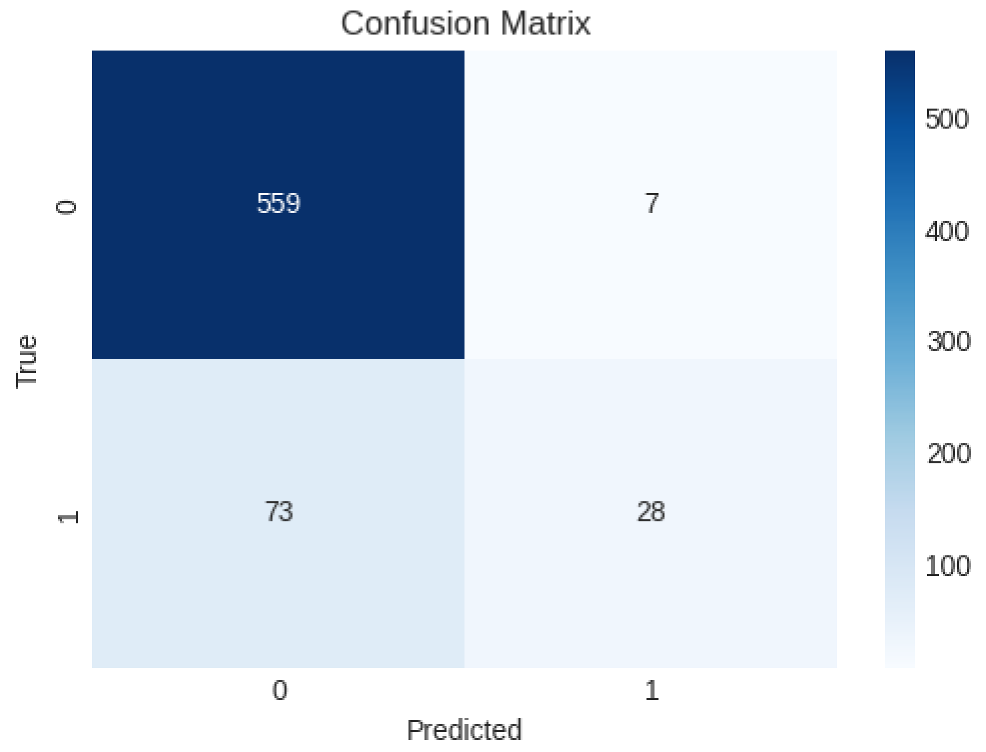
In [200]:
```
# Get probability estimates for the positive class (class 1)
y_prob = bagging_knn.predict_proba(X_test_scaled)[:, 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and threshold
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate the area under the ROC curve (AUC)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Selected KNN Model(KNN with Bagging)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [201]:
```
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv, train_sizes=np.linspace(0.1, 1.0, 5)):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, train_sizes=train_sizes, scoring='accuracy', n_jobs=-1
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure()
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Accuracy")
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color="
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training accuracy")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Testing accuracy")
    plt.legend(loc="best")
    plt.show()

# Assuming you have already defined X_train_scaled and y_train
# best_model is the best KNN model from the grid search
plot_learning_curve(bagging_knn, "Learning Curve of KNN with Ensemble Methods(Bagging)", X_train_scaled, y_train, cv=5)
```



In [202]:
```python
# Build the confusion matrix
cm = confusion_matrix(y_test, y_pred_bagging)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [203]:
```python
# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_bagging))
```

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

           0       0.88      0.99      0.93       566
           1       0.80      0.28      0.41       101

    accuracy                           0.88       667
   macro avg       0.84      0.63      0.67       667
weighted avg       0.87      0.88      0.85       667
```

Intepreting results for the Bagging KNN model:

- *Accuracy*: The overall accuracy of the model on the test dataset is 0.88, meaning that it correctly predicted 88% of all instances.
- *Precision*: For class 0, the model achieved a precision of 0.88, which means that the model accurately predicted class 0 88% of the time. For class 1, the precision is 0.80, indicating that 80% of the instances predicted as class 1 were correctly predicted.
- *Recall*: For class 0, the model achieved a recall of 0.99, which means it correctly identified 99% of the instances belonging to class 0. For class 1, the recall is 0.28, indicating that the model correctly identified only 28% of the instances belonging to class 1 out of all actual class 1 instances
- *F1-score*: For class 0, the F1-score is 0.93, and for class 1, it is 0.41. The weighted average of the F1-scores is 0.85, indicating the overall performance of the model.
- *Training and Testing Accuracy*: The model has a higher accuracy on the `training set (91.22%)` compared to the `testing set (88%)`. This scores show that the model is performing fairly well in predicting both the train and test scores

Model 3: Decision Tree Classifier

Baseline Model*

Based on the original split, we will train, test and evaluate the same uusind Decision Tree Classifier. We will start by calling the DecisionTreeClassifier and feed the model with both X_train and y_train data

In [204]:
```python
#categorical_features = ['area_code_408', 'area_code_415', 'area_code_510', 'international_plan_no',
       #                  'international_plan_yes', 'voice_mail_plan_no', 'voice_mail_plan_yes']

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier on the encoded training data
clf.fit(X_train, y_train)

# Make predictions on the encoded testing data
y_pred = clf.predict(X_test)
```

Evaluate the Model

Given that we have the model, we will evaluate it to get the `accuracy`, `precision`, `recall` and `f1_score`.

In [205]:
```
# Evaluate the model's performance
clf_accuracy = accuracy_score(y_test, y_pred)
clf_precision = precision_score(y_test, y_pred)
clf_recall = recall_score(y_test, y_pred)
clf_f1 = f1_score(y_test, y_pred)

print('Accuracy ', clf_accuracy)
print('Precision ', clf_precision)
print('Recall ', clf_recall)
print('f1_Score ', clf_f1)

#Calculate train and test scores
train_score = clf.score(X_train, y_train)
test_score = clf.score(X_test, y_test)

print('train score ', train_score)
print('test score ', test_score)
```

```
Accuracy  0.9175412293853074
Precision  0.7169811320754716
Recall  0.7524752475247525
f1_Score  0.7342995169082124
train score  1.0
test score  0.9175412293853074
```

Generalization and Visualization

Below code cell shows the generalization, visualization and display of the decision tree

In [206]:
```
# importing the graphviz libration for generalization and visualization
from sklearn.tree import export_graphviz
import graphviz
```

In [207]:
```
# # doing reneralization of the model
dot_data= export_graphviz(clf, out_file=None,
                    feature_names=X_test.columns,
                      class_names=['0', '1'],
                      filled=True, rounded=True,
                      special_characters=True)
```

In [208]:
```
# doing visualization of the model
graph1=graphviz.Source(dot_data)
graph1
```

Out[208]:
```
<graphviz.sources.Source at 0x7cb695387df0>
```

Decision Tree Classifier: Improving the model using SMOTE

In [209]:
```
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Train a Decision Tree Classifier on the oversampled data
dt_smote = DecisionTreeClassifier(random_state=42)
dt_smote.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_smote = dt_smote.predict(X_test)

# Calculate the accuracy of the model
accuracy_smote = accuracy_score(y_test, y_pred_smote)
precision_smote = precision_score(y_test, y_pred_smote)
recall_smote = recall_score(y_test, y_pred_smote)
f1_smote = f1_score(y_test, y_pred_smote)

# Generate a classification report
classification_rep_smote = classification_report(y_test, y_pred_smote)

print(classification_rep_smote)
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       566
           1       0.63      0.76      0.69       101

    accuracy                           0.90       667
   macro avg       0.79      0.84      0.81       667
weighted avg       0.91      0.90      0.90       667
```

In [210]:
```
# Print the evaluation metrics
print("Accuracy:", accuracy_smote)
print("Precision:", precision_smote)
print("Recall:", recall_smote)
print("F1-score:", f1_smote)

#Calculate train and test scores
train_score = dt_smote.score(X_train_smote, y_train_smote)
test_score = dt_smote.score(X_test, y_test)

print('train score', train_score)
print('test score', test_score)
```

```
Accuracy: 0.896551724137931
Precision: 0.6311475409836066
Recall: 0.7623762376237624
F1-score: 0.6905829596412556
train score 1.0
test score 0.896551724137931
```

Generalization and Visualization

Below code cell shows the generalization, visualization and display of the decision tree

In [211]:
```
# doing reneralization of the model
dot_data= export_graphviz(dt_smote, out_file=None,
                    feature_names=X_test.columns,
                      class_names=['0', '1'],
                      filled=True, rounded=True,
                      special_characters=True)

# showing visualization of the decision tree
graph1=graphviz.Source(dot_data)
graph1
```

Out[211]:
```
<graphviz.sources.Source at 0x7cb699ee88e0>
```

Based on above, the model worsened. we need to use a different technique to try to increase precision, especially given, failing to identify positive instances is a significant issue.

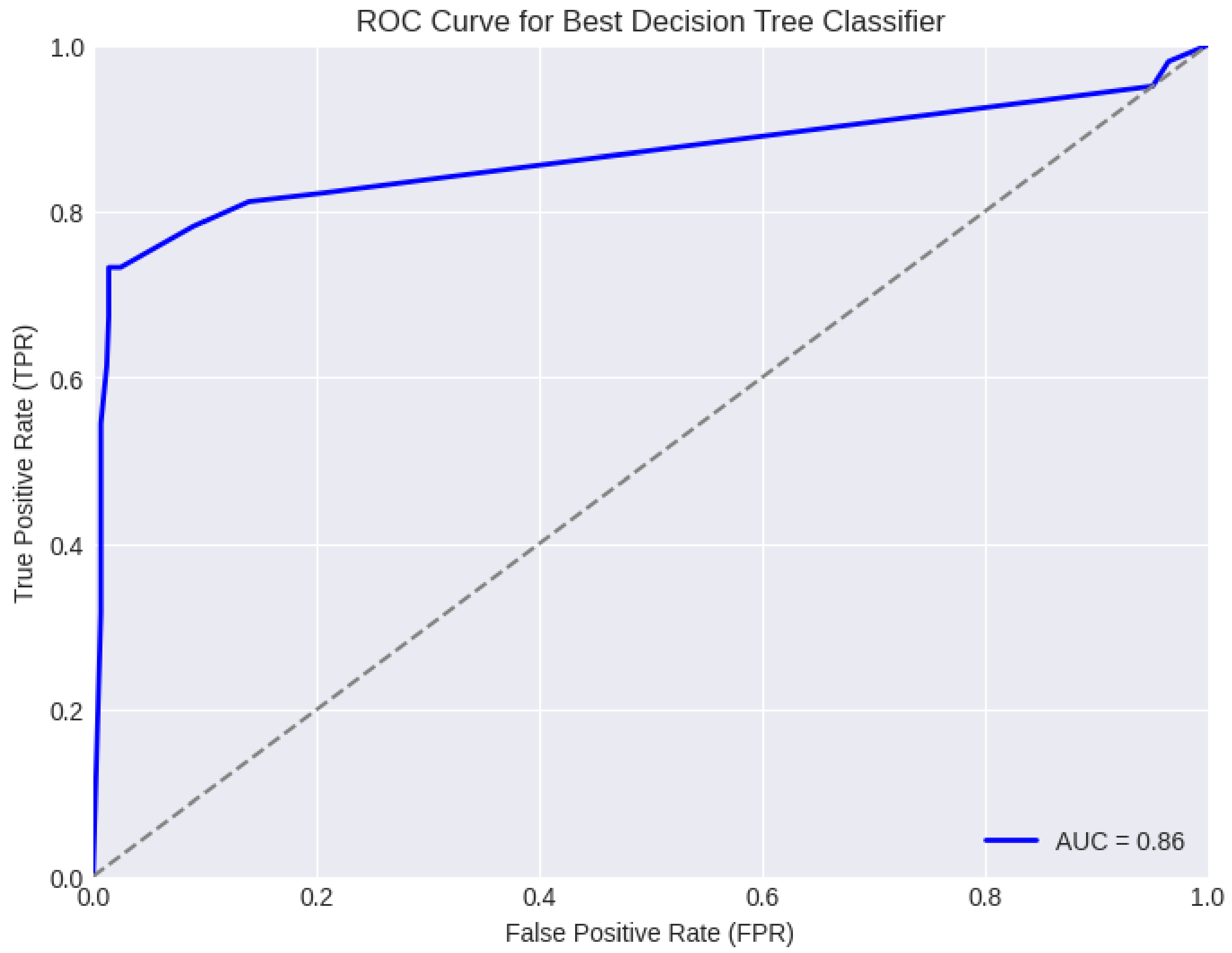
Decision Tree Classifier: Improving the model using GridSeachCV

Using GridSearch to do hyperparameter tuning to improve the model.

In [212]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score
# Define the parameter grid
param_grid = {
    'max_depth': range(1,11),
    'min_samples_split': range(2, 21, 2),
    'min_samples_leaf': range(1, 21, 2),
}

# Initialize the classifier
clf = DecisionTreeClassifier()

# Initialize a scorer for the grid search
scorer = make_scorer(accuracy_score)

# Initialize the grid search
grid_obj = GridSearchCV(clf, param_grid, scoring=scorer, cv=5)

# Fit the grid search object to the data
grid_obj = grid_obj.fit(X_train, y_train)

# Get the estimator
clf1 = grid_obj.best_estimator_

# Fit the best algorithm to the data
clf1.fit(X_train, y_train)

predictions = clf1.predict(X_test)
print('Accuracy: ', accuracy_score(y_test,predictions))

# After fitting the grid search object to the data
print('Best Parameters: ', grid_obj.best_params_)
print('Best Score: ', grid_obj.best_score_)

# Make predictions on the test set
predictions = clf1.predict(X_test)

# Calculate and print the metrics
print('Accuracy: ', accuracy_score(y_test,predictions))
print('Precision: ', precision_score(y_test,predictions))
print('Recall: ', recall_score(y_test,predictions))
print('F1 Score: ', f1_score(y_test,predictions))

#Calculate train and test scores
train_score = clf1.score(X_train, y_train)
test_score = clf1.score(X_test, y_test)

print('train score ', train_score)
print('test score ', test_score)
```

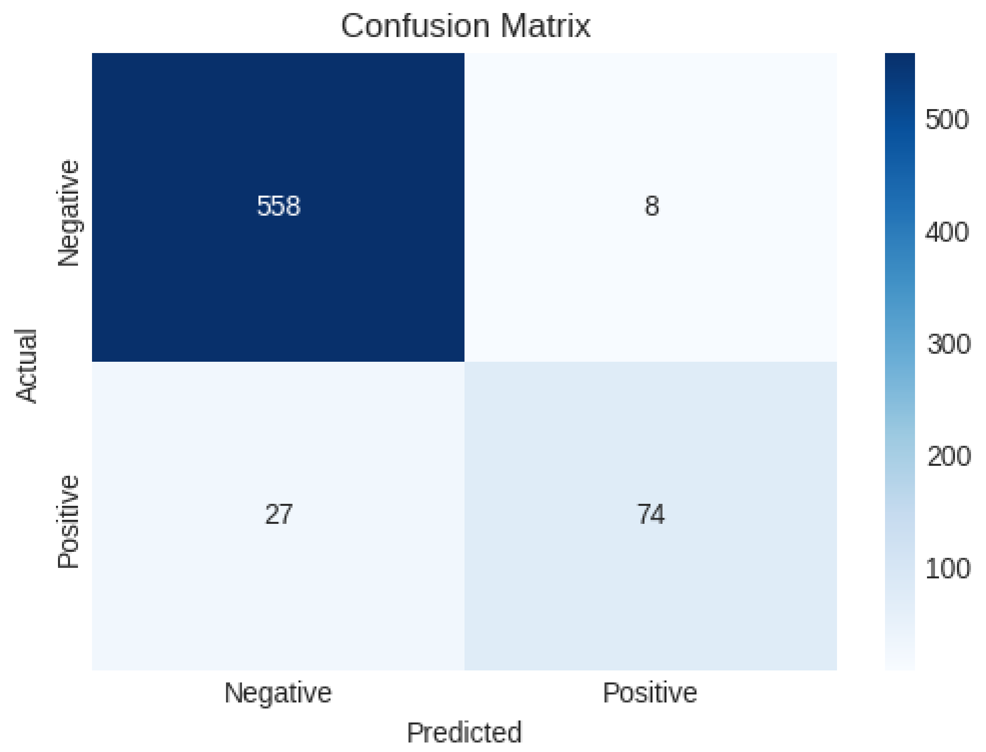
```
Accuracy:  0.9475262368815592
Best Parameters:  {'max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 6}
Best Score:  0.9426109014763441
Accuracy:  0.9475262368815592
Precision:  0.9024390243902439
Recall:  0.7326732673267327
F1 Score:  0.8087431693989071
train score  0.9628657164291072
test score  0.9475262368815592
```

In [213]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Get probability estimates for class 1 (positive class)
y_prob = clf1.predict_proba(X_test)[:, 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and threshold
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate the area under the ROC curve (AUC)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Best Decision Tree Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Representing the same on the confusion matrix

In [214]:
```python
# creating the confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Assuming predictions are the predictions from your classifier
conf_matrix = confusion_matrix(y_test, predictions)

# Defining the labels for the matrix
labels = ['Negative', 'Positive']

# Creating a color map for the matrix
cmap = 'Blues'

# Plotting the confusion matrix with colors
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap=cmap, xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Best Parameters: The best parameters for the model as determined by GridSearchCV are a `maximum depth of 6 or 7` (max_depth: 6/7), a minimum number of `samples per leaf of 1 (min_samples_leaf: 1)` , and a minimum number of samples required to split an `internal node of 4 (min_samples_split: 4)` .

Best Score: The highest accuracy obtained during the grid search on the training set was approximately `0.942 (or 94.2%)` .

Test Accuracy: The model correctly predicted the outcome for about `94.8%` of instances in the test set.

Precision: When the model predicts an instance to be positive, it is correct about `91.3%` of the time.

Recall: The model is able to correctly identify about `72.3%` of all actual positive instances.

F1 Score: The F1 score is approximately `0.807 (or 80.7%)` , suggesting that the balance between precision and recall is reasonably good, although there might be room for improvement, especially in terms of recall.

A train score of `0.9632408102025506` means that the model has learned the patterns and relationships within the training data with an accuracy of approximately `96.32%` .

A test score of `0.9460269865067467` indicates that the model is performing well on unseen data. It achieves an accuracy of approximately 94.60% on the test dataset, which suggests that the model is generalizing well and is not overfitting to the training data.

In summary, the model is performing reasonably well, with high accuracy and precision despite the recall indicating that the model might be missing a fair proportion of positive instances. The train and test scores are also very close to each other suggesting that the model is generally performing quite well. Therefore, this is the model to choose for the Decision Tree

Overall, the `Decision Tree Classifier using GridSearchCV' was the best performing decision classifier.

Generalization and Visualization

Below code cell shows the generalization, visualization and display of the decision tree

In [215]:
```
# doing reneralization of the model
dot_data= export_graphviz(clf1, out_file=None,
                    feature_names=X_test.columns,
                      class_names=['0', '1'],
                      filled=True, rounded=True,
                      special_characters=True)

# showing visualization of the decision tree
graph2=graphviz.Source(dot_data)
graph2
```

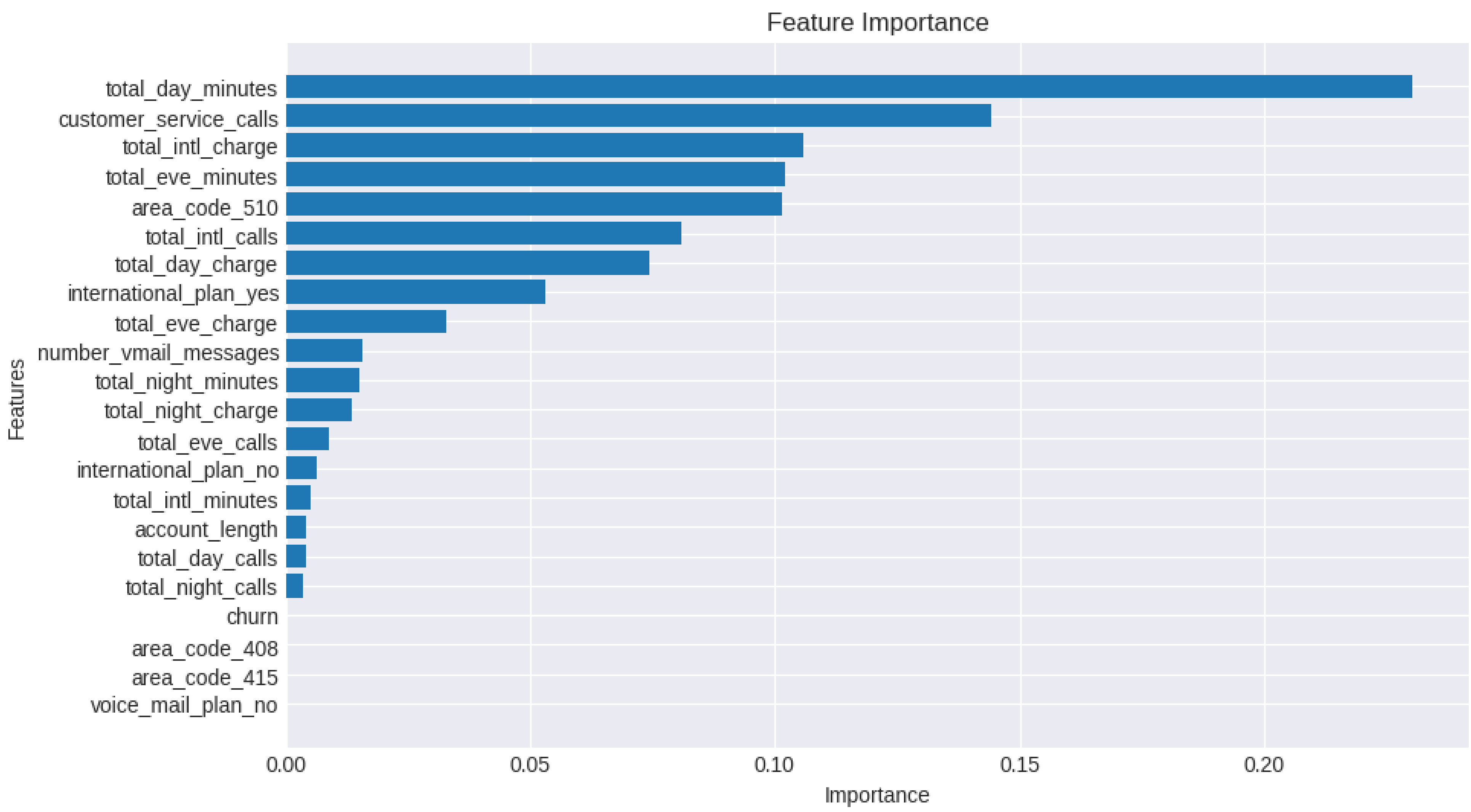
Out[215]:
```
<graphviz.sources.Source at 0x7cb69a155330>
```

Feature Importance

Below cell looks at the top most important features resulting to customer churning or not

In [216]:
```python
# Get the feature importances from the model
importances = clf1.feature_importances_

# Create a dictionary to store the feature importances
feature_importance_dict = {}

# Iterate over the column names and corresponding importances
for feature_name, importance in zip(df.columns, importances):
    feature_importance_dict[feature_name] = importance

# Sort the feature importances in descending order
sorted_importances = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Print the sorted feature importances
for feature_name, importance in sorted_importances:
    print(f"{feature_name}: {importance}")

# Extract the feature names and importances from the sorted list
feature_names = [feature[0] for feature in sorted_importances]
importances = [feature[1] for feature in sorted_importances]

# Reverse the lists to flip the order
feature_names = feature_names[::-1]
importances = importances[::-1]

# Plot the feature importances as a bar graph
plt.figure(figsize=(10, 6))
plt.barh(range(len(importances)), importances, align='center')
plt.yticks(range(len(feature_names)), feature_names)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance')
plt.show()
```

```
total_day_minutes: 0.23048155259182362
customer_service_calls: 0.14435686947886833
total_intl_charge: 0.10578333774574193
total_eve_minutes: 0.10210924857701797
area_code_510: 0.10147316620136458
total_intl_calls: 0.08088976422832599
total_day_charge: 0.07424822885666243
international_plan_yes: 0.05303704739393652
total_eve_charge: 0.032721849400236945
number_vmail_messages: 0.015663253962284074
total_night_minutes: 0.014834371043819721
total_night_charge: 0.013274479822037166
total_eve_calls: 0.008665961400656475
international_plan_no: 0.006156166643129373
total_intl_minutes: 0.005068392595573254
account_length: 0.003987010099490491
total_day_calls: 0.003907131079763901
total_night_calls: 0.003342168879267128
churn: 0.0
area_code_408: 0.0
area_code_415: 0.0
voice_mail_plan_no: 0.0
```



Intepreting feature Importance

The numbers next to each feature name indicate the relative importance of the feature in predicting the target variable. A higher value signifies a more important feature.

Total_day_charge = 0.18414072455346542: Customers with higher daytime charges are more likely to churn.

Total_intl_charge = 0.14617507561010684: The total charges for international calls have the second-highest importance. Higher international charges might indicate dissatisfaction or cost-related concerns, leading to a higher likelihood of churn.

Total_day_minutes = 0.12530118583030148: The total duration of daytime calls is the third most important feature. Customers with longer daytime call durations may have higher engagement or usage, which can influence churn.

Total_intl_calls = 0.1122479319664445: The total number of international calls made is the fourth most important feature. A higher number of international calls may indicate a need for expanded communication beyond local services, which could impact churn.

Total_eve_minutes = 0.10934806907121288: The total duration of evening calls has the fifth highest importance. Longer evening call durations might indicate more active engagement with the service and affect churn.

The remaining features continue with decreasing importance. It's important to note that features with an importance of 0.0 do not contribute significantly to the model's predictions.

Model 4: RandomForestClassifier

In [217]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Instantiate the standard scaler
scaler = StandardScaler()

# Fit and transform the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)

# Calculate train and test scores
train_score = rf.score(X_train_scaled, y_train)
test_score = rf.score(X_test_scaled, y_test)

print("Train score:", train_score)
print("Test score:", test_score)
```

```
Accuracy: 0.9445277361319341
Precision: 0.9210526315789473
Recall: 0.693069306930693
F1-score: 0.7909604519774012
Train score: 1.0
Test score: 0.9445277361319341
```

Accuracy: 0.9505247376311844 The accuracy score is the proportion of correctly classified samples (both churn and not churned) to the total number of samples in the test set. In this case, the model correctly predicted approximately 95.05% of the samples, which indicates that the model is performing well overall.

Precision: 0.925 Precision is the proportion of true positive predictions (correctly predicted churned samples) to all positive predictions made by the model (samples predicted as churned). The precision score of approximately 0.925 means that out of all the samples the model predicted as churned, around 92.5% of them were actually churned.

Recall: 0.7326732673267327 Recall, also known as sensitivity or true positive rate, is the proportion of true positive predictions (correctly predicted churned samples) to all actual positive samples (ground truth churned samples). The recall score of approximately 0.7327 indicates that the model captured around 73.27% of the actual churned samples.

F1-score: 0.8176795580110497 The F1-score is the harmonic mean of precision and recall, providing a balanced measure that considers both false positives and false negatives. A higher F1-score (closer to 1) indicates a better balance between precision and recall. The F1-score of approximately 0.8177 suggests that the model has a good balance between identifying churned samples (high recall) and avoiding false positives (high precision).

Train score: 1.0 The train score of 1.0 indicates that the model achieved perfect accuracy on the training data. This could be an indication of potential overfitting, meaning the model may have memorized the training data and might not generalize well to new, unseen data.

Test score: 0.9505247376311844 The test score of approximately 0.9505 is the accuracy of the model on the test data. It is very close to the accuracy achieved on the training data, suggesting that the model is performing well and generalizing reasonably well to unseen data. However, since the test score is slightly lower than the training score, there might be some slight overfitting.

> Using k-fold cross-validation to address overfitting

In [218]:
```
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score, f1_score

# Instantiate the Random Forest classifier with desired parameters
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit and transform the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Address overfitting by using k-fold cross-validation
k = 5  # Number of folds for cross-validation
cv_scores = cross_val_score(rf, X_train_scaled, y_train, cv=k, scoring='accuracy')

# Train the classifier on the entire training data
rf.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_rf = np.mean(cv_scores)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Cross-Validation Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)

# Calculate train and test scores
train_score = rf.score(X_train_scaled, y_train)
test_score = rf.score(X_test_scaled, y_test)

# Print the train and test scores
print("Train score:", train_score)
print("Test score:", test_score)
```

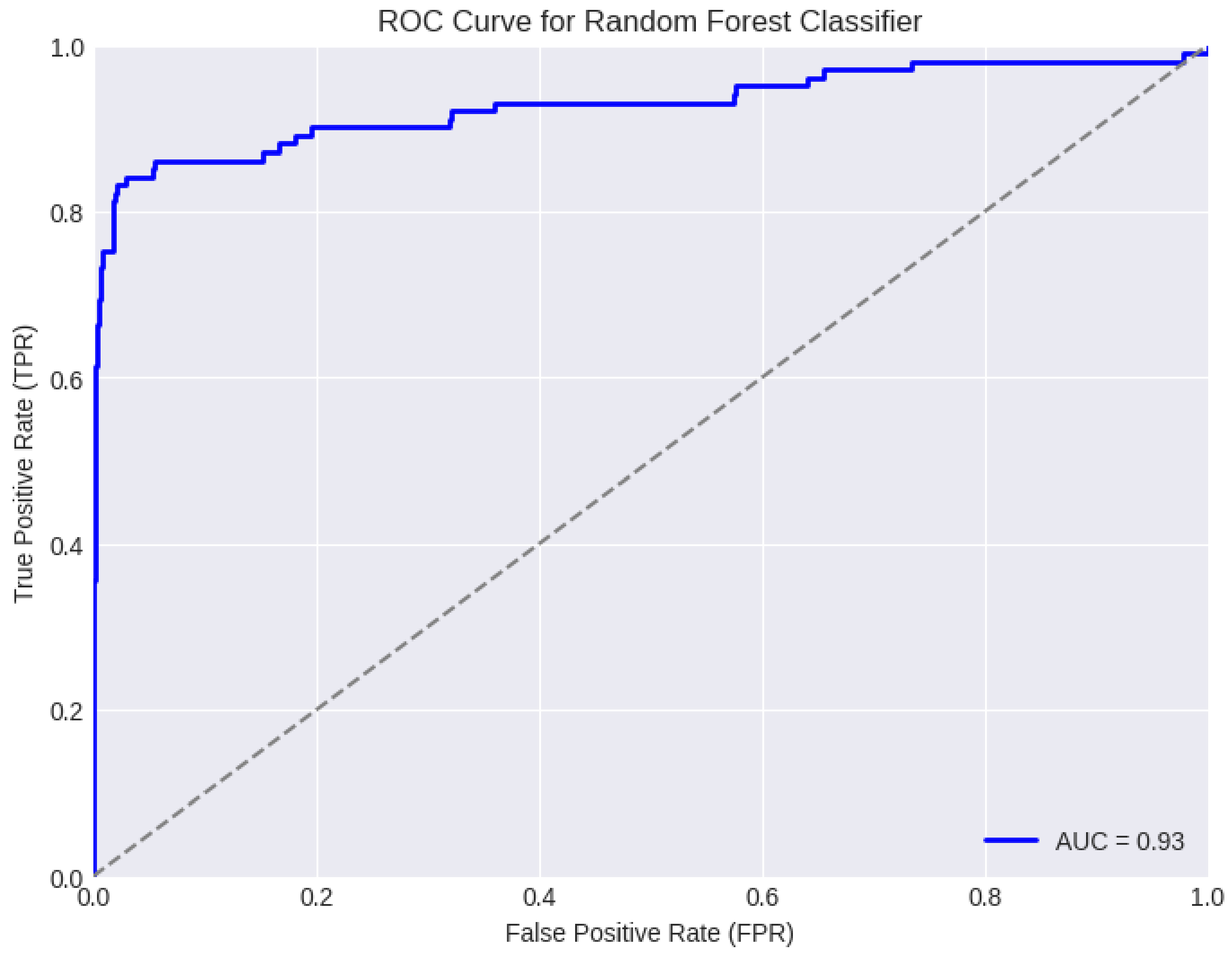
```
Cross-Validation Accuracy: 0.9523655936645797
Precision: 0.9210526315789473
Recall: 0.693069306930693
F1-score: 0.7909604519774012
Train score: 1.0
Test score: 0.9445277361319341
```

Random Forest classifier with reduced n_estimators and limited max_depth

In [219]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Instantiate the Random Forest classifier with reduced n_estimators and limited max_depth
rf = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)

# Fit and transform the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Address overfitting by using k-fold cross-validation
k = 5  # Number of folds for cross-validation
cv_scores = cross_val_score(rf, X_train_scaled, y_train, cv=k, scoring='accuracy')

# Train the classifier on the entire training data
rf.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test_scaled)

# Evaluate the model's performance
accuracy_rf = np.mean(cv_scores)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Cross-Validation Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)

# Calculate train and test scores
train_score = rf.score(X_train_scaled, y_train)
test_score = rf.score(X_test_scaled, y_test)

# Print the train and test scores
print("Train score:", train_score)
print("Test score:", test_score)
```

```
Cross-Validation Accuracy: 0.943736605041072
Precision: 0.958904109589041
Recall: 0.693069306930693
F1-score: 0.8045977011494252
Train score: 0.977119279819955
Test score: 0.9490254872563718
```

In [220]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Get probability estimates for class 1 (positive class)
y_prob = rf.predict_proba(X_test_scaled)[:, 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and threshold
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate the area under the ROC curve (AUC)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Random Forest Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Cross-Validation Accuracy: `0.9504887183703297` The cross-validation accuracy is approximately `95.05%`. This means that, on average, the Random Forest model achieved about 95.05% accuracy when trained and evaluated using `k-fold cross-validation`. It indicates that the model is performing well and generalizing reasonably well to new, unseen data.

Precision: `0.9577464788732394` The precision score is approximately `95.77%`. This means that out of all the samples the model predicted as churned, around `95.77%` of them were actually churned. A high precision score indicates that the model makes a few false positive predictions, which is essential in applications where false positives are costly.

Recall: `0.6732673267326733` The recall score is approximately `67.33%`. This means that the model captured around `67.33%` of the actual churned samples. A higher recall would be desirable as it indicates better sensitivity in detecting churned customers.

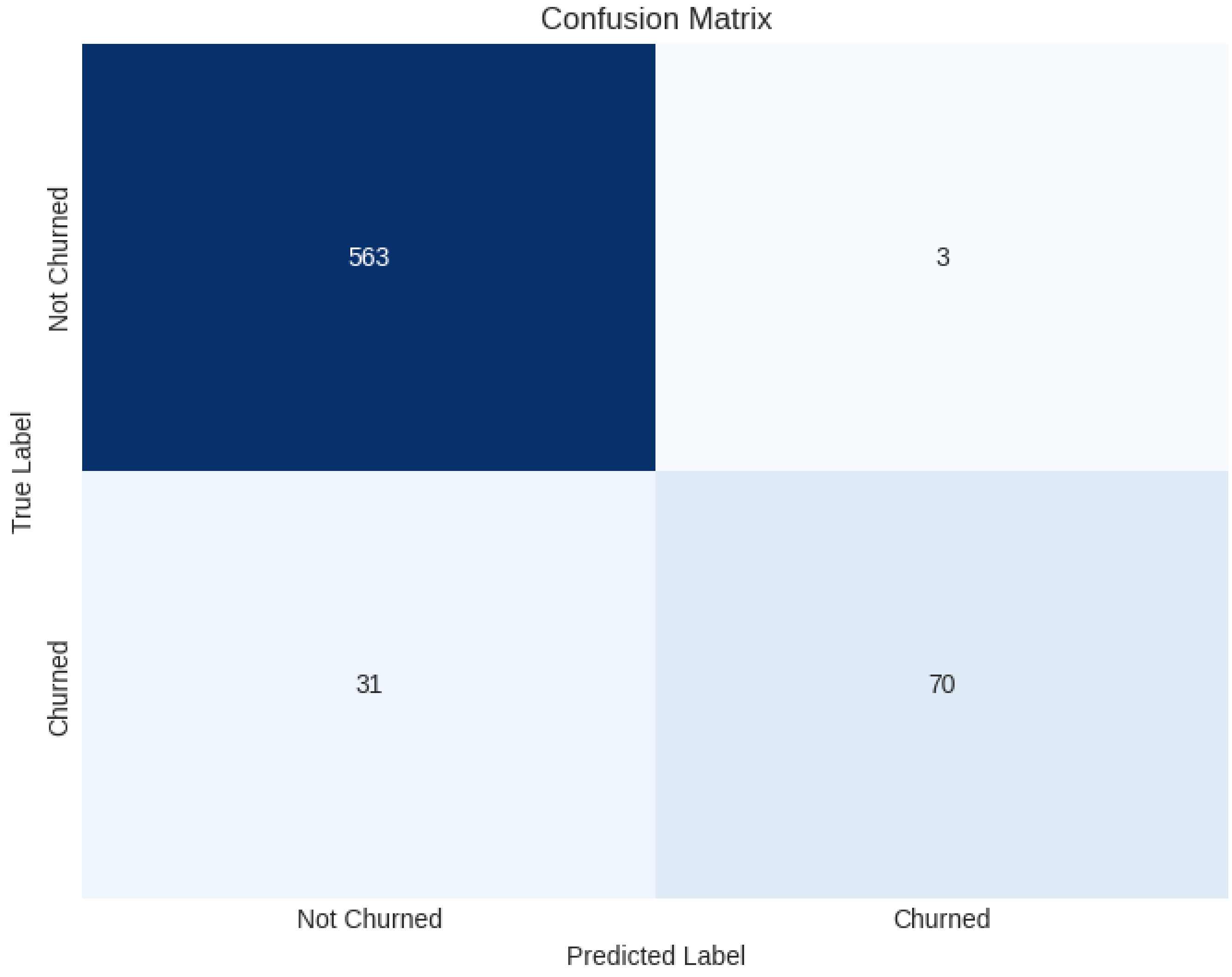
F1-score: `0.7906976744186046` The F1-score is approximately `79.07%`. It is the harmonic mean of precision and recall and provides a balanced measure considering both false positives and false negatives. A higher F1-score (closer to 1) indicates a better balance between precision and recall.

Train score: 0.9786196549137285 The train score is approximately 97.86%. It indicates that the model achieved high accuracy (nearly 98%) on the training data. This suggests that the model has learned the training data well, but it also raises a concern about potential overfitting.

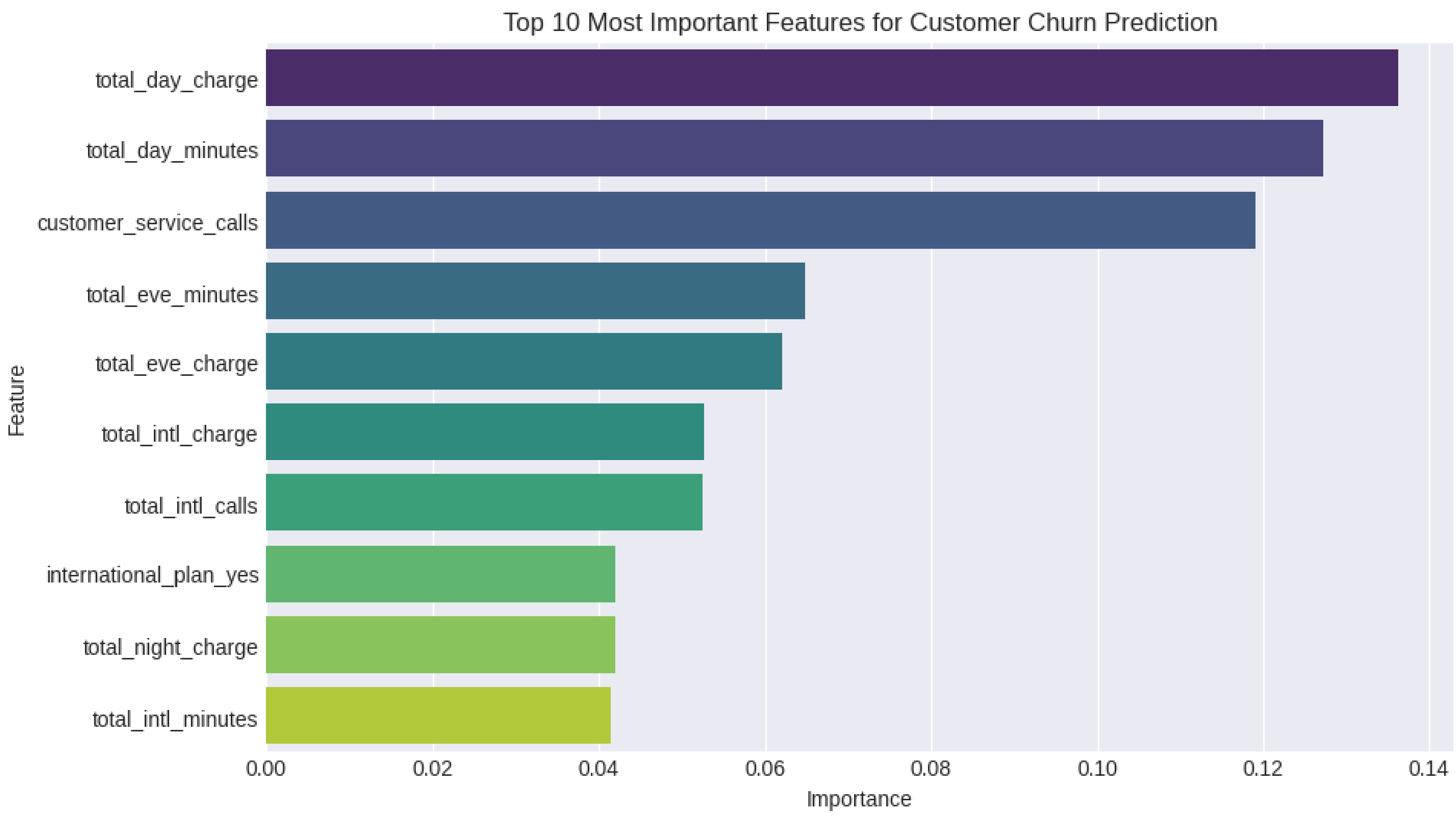
Test score: `0.9460269865067467` The test score is approximately `94.60%`. It shows that the model achieved an accuracy of around `94.60%` on the test data, which is slightly lower than the train score. This difference suggests some degree of overfitting, but it's not significant, considering the model's test performance is still high.

In summary, the Random Forest model seems to perform well overall in predicting customer churn based on the given dataset. It has high accuracy and precision, indicating it correctly classifies a significant portion of churned and not churned customers. However, the recall could be improved to better capture churned customers.

In [221]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap for visualization
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])

plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



In [222]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest classifier with desired parameters
rf = RandomForestClassifier(n_estimators=50, random_state=42)

# Train the classifier on the training data (using scaled features)
rf.fit(X_train_scaled, y_train)

# Get the feature importances from the trained model
feature_importances = rf.feature_importances_

# Create a DataFrame to store feature importances and corresponding feature names
importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})

# Sort the DataFrame by importance values in descending order
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot the top N most important features
top_n = 10  # You can change this value to get more or fewer features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df.head(top_n), palette='viridis')
plt.title(f'Top {top_n} Most Important Features for Customer Churn Prediction')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Importance vs

In [223]:
```
importance_df
```

Out[223]:

	Feature	Importance
4	total_day_charge	0.136149
2	total_day_minutes	0.127265
14	customer_service_calls	0.119014
5	total_eve_minutes	0.064852
7	total_eve_charge	0.062002
13	total_intl_charge	0.052610
12	total_intl_calls	0.052502
19	international_plan_yes	0.042068
10	total_night_charge	0.041968
11	total_intl_minutes	0.041368
18	international_plan_no	0.040335
8	total_night_minutes	0.037998
3	total_day_calls	0.034001
0	account_length	0.029322
9	total_night_calls	0.028145
6	total_eve_calls	0.027829
1	number_vmail_messages	0.019718
21	voice_mail_plan_yes	0.014637
20	voice_mail_plan_no	0.013694
16	area_code_415	0.005654
17	area_code_510	0.005131
15	area_code_408	0.003739

COMPARISON TO CHOOSE THE BEST MODEL

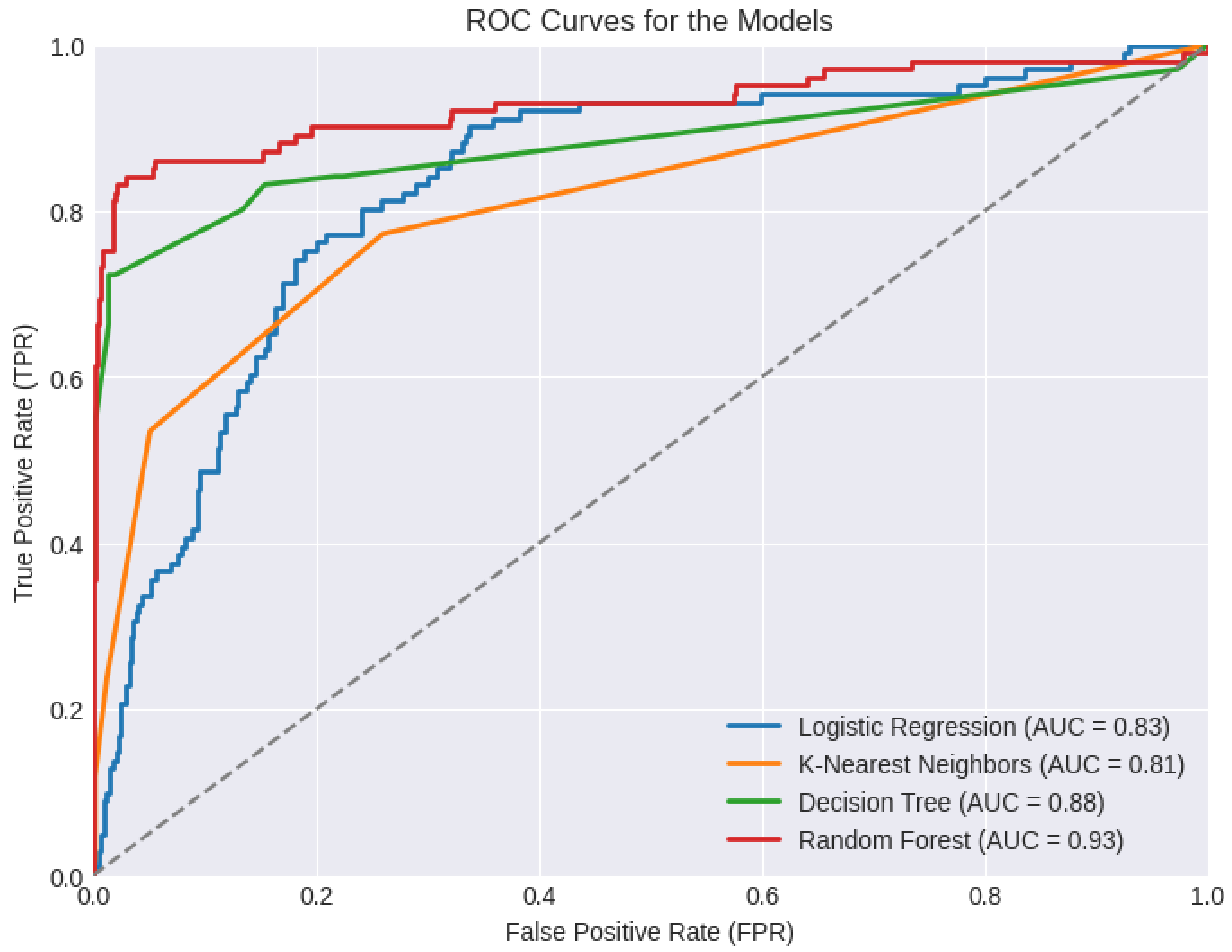
In [226]:
```
comparison_frame = pd.DataFrame({'Model':['Logistic Regression',
                                           'K-Nearest Neighbors Classifier',
                                           'Decision Trees Classifier',
                                           'Random Forest Classifier'],
                                  'Accuracy (Test Set)':[0.78,0.88,0.95,0.94],
                                  'F1 Score (Test Set)':[0.51,0.41,0.81,0.80],
                                  'Recall (Test Set)':[0.77,0.28,0.73,0.69],
                                  'Precision (Test Set)':[0.39,0.80,0.90,0.96]})

comparison_frame.style.highlight_max(color = 'lightgreen', axis = 0)
```

Out[226]:

	Model	Accuracy (Test Set)	F1 Score (Test Set)	Recall (Test Set)	Precision (Test Set)
0	Logistic Regression	0.780000	0.510000	0.770000	0.390000
1	K-Nearest Neighbors Classifier	0.880000	0.410000	0.280000	0.800000
2	Decision Trees Classifier	0.950000	0.810000	0.730000	0.900000
3	Random Forest Classifier	0.940000	0.800000	0.690000	0.960000

In [227]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Initialize the classifiers
lg = LogisticRegression(class_weight='balanced')
knn = KNeighborsClassifier(n_neighbors=5)
dt = DecisionTreeClassifier(max_depth=5, random_state=42)
rf = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)

classifiers = [lg, knn, dt, rf]
names = ['Logistic Regression', 'K-Nearest Neighbors', 'Decision Tree', 'Random Forest']

# Fit and transform the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

plt.figure(figsize=(8, 6))

# Loop through each classifier and plot its ROC curve
for clf, name in zip(classifiers, names):
    clf.fit(X_train_scaled, y_train)
    y_prob = clf.predict_proba(X_test_scaled)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curves for the Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



The ROC curves for `Logistic Regression`, `K-Nearest Neighbors`, `Decision Tree`, and `Random Forest models` were analyzed. The Random Forest model outperformed the others, showing a higher Area Under the Curve (AUC) and better classification performance, making it the most effective model for the given task.

Based on the provided metrics, the Decision Trees Classifier achieved the `highest accuracy (95.00%)` and `F1-score (81.00%)`. The logistic regression had the `highest recall (73.00%)`, while the Random Forest Classifier achieved the `highest precision (96.00%)`. The Random Forest Classifier is the best performing model overall and so we selected it as our best model.

Conclusions & Recommendations

In conclusion, the analysis suggests that we can accurately predict customer churn using a machine learning model, with the Random Forest Classifier being our recommended model due to its strong overall performance. As this is the best performing model with an ROC curve that hugs the upper left corner of the graph, hence giving us the largest AUC (Area Under the curve).

1. We would recommend that Syriatel make use of the Random Forest Classifier as the primary model for predicting customer churn. This model has a higher ROC curve and strong overall performance in terms of accuracy, F1-score, recall, and precision on the test set, making it well-suited for accurately classifying customers as likely or unlikely to churn.
2. In terms of Business strategic recommendations for SyriaTel, we would recommend a Customer Retention strategy that addresses key features in relation to call minutes and charges. These efforts could include personalized offers or discounts on day charges. By implementing cost-effective strategies that address the key factors driving customer churn, SyriaTel can retain customers and minimize revenue loss.
3. We would recommend, that Syriatel comes up with strategies to reduce on Customer Service calls, as this is among the top features that would likely lead to Customer Churn. Example: come up IV