SyriaTel Customer Churn

1. Andrew Maina
2. Tasha Kanyiva
3. Emmanuellar Karisa
4. Diana Mbuvi
5. Mercy Ronoh
6. Ashley Simiyu

Table of Contents

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Conclusion & Recommendation

SyriaTel

1. Business Understanding

1.1 Overview

SyriaTel is a telecommunication company which provides state of the art communication services. However, it is grappling with a challenge of customers churning. The business is

services. However, it is grappling with a challenge of customers churning. The business is facing revenue losses as a result of customers terminating their subscription. The primary goal of this project is to analyse the factors fueling customer churn and build a predictive model to identify patterns and indicators of the customers likely to churn soon. The target stakeholders are executives, product managers, marketing teams, finance and account teams, and contact centre team at Syria Tel. The data used can be accessed from here.

1.2 Business Problem

SyriaTel has noticed that customers are terminating their services, which has resulted in the company's loss of revenue and decreased market share. The stakeholders are afraid that this may eventually damage the company's brand. It is imperative to understand the factors driving the rate of customer churn before they can implement strategies to retain subscribers. The problem at hand is to develop a classifier model based on historical customer data to analyse churn patterns and predict the users at risk of unsubscribing. This will enable the company to implement various retention startegies, improve customer satisfaction, and eventually lessen the overall impact of turnover on its operations.

For this challenge, the project will explore various types of classification models like logistic regression, decision trees and random forests. These algorithms are ideal as we are predicting binary outcomes, that is, customer churn or no customer churn.

Project Objectives

1. To identify drivers of customer churn
2. To develop binary classifiers to predict potential unsubscribers
3. To provide actionable suggestions for retaining users

Research questions

1. What is the quality of the customer service calls?
2. Which area codes have a high churn rate?
3. Can frequent customer service contacts predict customers who are likely to leave the company?
4. How much are people using for the plans?
5. Does a customer's geographic location influence their chance of leaving the company?
6. Which machine learning model would be most effective for the business problem?
7. How does the churn differ across different states?
8. Which machine learning algorithms perform best for predicting customer churn?
9. What are the main drivers of customer churn?
10. How effective are the classification models in predicting customer churn and what are the strengths and weaknesses of each model?

2. Data Understanding

We will use historical data from SyriaTel which contain information about customer call data obtained from Kaggle. The dataset has numerical features like customer phone numbers

data obtained from Kaggle. The dataset has numerous features like user phone numbers, total calls, total call durations, state, customers churning e.t.c.distributed across 20 columns with 3333 records.

Below are the columns in the SyriaTel customer dataset:

- state - Residence state of the cutomer
- account length - the number of days the customer has had an account
- area code - Area code of the customer
- phone number - Customer's registered phone number
- international plan - Yes if the customer has an voicemail plan and no if not
- voice mail plan - Yes if the customer has an internation plan and no if not
- number vmail messages - number of customers vocemail messages
- total day minutes - total minutes the customer was in call for during daytime
- total day calls - total calls the customer made at daytime
- total day charge - total amount the customer was charged by SyriaTel for calls made during the day
- total eve minutes - total minutes the customer was in call for during the evening
- total eve calls - total calls the customer made during the evening
- total eve charge - total amount the customer was charged by SyriaTel for calls made during the evening
- total night minutes - total minutes the customer was in call for at night
- total night calls - total calls the customer made at night
- total night charge - total amount the customer was charged by SyriaTel for calls made during at night
- total intl minutes - total minutes the customer was in international calls
- total intl calls - total international calls made by the customer
- total intl charge - total amount the cutomer was charged by SyriaTel on international calls
- customer service calls - total number of calls the customer made to SyriaTel customer service
- churn - true if customer dicontinued services and false otherwise

In [135…

```
# import the necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    ConfusionMatrixDisplay,
    accuracy_score,
    roc_auc_score
)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import (
    cross_val_score,
    GridSearchCV,
    train_test_split
)
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [6]:

```
!ls datasets/
```

telecom_churn.csv

In [7]:
```
# load the data
df = pd.read_csv('./datasets/telecom_churn.csv')
# check the first five rows
df.head()
```

Out[7]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34

5 rows × 21 columns

In [8]:
```
# check the last five rows
df.tail()
```

Out[8]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	t cha
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	2
3329	WV	68	415	370-3271	no	no	0	231.1	57	3
3330	RI	28	510	328-8230	no	no	0	180.8	109	3
3331	CT	184	510	364-6381	yes	no	0	213.8	105	3
3332	TN	74	415	400-4344	no	yes	25	234.4	113	3

5 rows × 21 columns

In [9]:
```
# check ten rows at random
df.sample(10)
```

Out[9]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	t cha

3117	OR	113	415	367-5923	no	no	0	159.8	143	2
681	WV	107	415	361-1581	no	no	0	123.1	100	2
3261	VT	131	415	416-8394	no	no	0	175.1	73	2
1563	MN	161	415	394-8086	no	yes	39	218.5	76	3
2682	DC	55	510	354-5058	yes	no	0	106.1	77	1
2110	CT	102	415	421-6694	no	yes	25	137.4	100	2
2764	RI	123	510	348-8711	no	yes	23	245.0	88	4
884	DE	73	415	355-9541	no	no	0	254.8	85	4
1021	WV	67	408	406-6708	no	no	0	167.8	91	2
1582	WA	86	510	387-6498	no	no	0	150.5	92	2

10 rows × 21 columns

In [10]:
```
# Check data types and column information
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
```

No empty values present in the dataset. We will find out when we look at unique values

In [11]:
```
# Check the statistical summary of the data
df.describe()
```

Out[11]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	33
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	2(
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	1(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	2(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	2
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	3(

Columns are in different scales and others such as *area code* should not be treated as numerical

In [12]:
```
df.corr()
```

Out[12]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	to
account length	1.000000	-0.012463	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.(
area code	-0.012463	1.000000	-0.001994	-0.008264	-0.009646	-0.008264	0.003580	-0.(
number vmail messages	-0.004628	-0.001994	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.(
total day minutes	0.006216	-0.008264	0.000778	1.000000	0.006750	1.000000	0.007043	0.(
total day calls	0.038470	-0.009646	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.(
total day charge	0.006214	-0.008264	0.000776	1.000000	0.006753	1.000000	0.007050	0.(
total eve minutes	-0.006757	0.003580	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.(
total eve calls	0.019260	-0.011886	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.(
total eve charge	-0.006745	0.003607	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.(
total night minutes	-0.008955	-0.005825	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.(

minutes								
total night calls	-0.013176	0.016522	0.007123	0.022972	-0.019557	0.022972	0.007586	0.(
total night charge	-0.008960	-0.005845	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.(
total intl minutes	0.009514	-0.018288	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.(
total intl calls	0.020661	-0.024179	0.013957	0.008033	0.004574	0.008032	0.002541	0.(
total intl charge	0.009546	-0.018395	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.(
customer service calls	-0.003796	0.027572	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.(
churn	0.016541	0.006174	-0.089728	0.205151	0.018459	0.205151	0.092796	0.(

Some features are perfectly positively correlated with a value of 1.0:

1. *total day minutes* & *total day charge*
2. *total day charge* & *total day minutes*
3. *total eve minutes* & *total eve charge*
4. *total eve charge* & *total eve minutes*

In [13]:
```
# Check the shape of the data
df.shape
```

Out[13]:
```
(3333, 21)
```

Investigate Unique Values in Each Column

Investigate unique values in the columns to find any uncaught null values or incorrect data types

In [14]:
```
# Check number of customers in each unique state
df['state'].value_counts()
```

Out[14]:
```
WV    106
MN     84
NY     83
AL     80
OR     78
WI     78
OH     78
VA     77
WY     77
CT     74
VT     73
ID     73
MI     73
TX     72
HI     72
```

```
UT     72
IN     71
MD     70
KS     70
NC     68
NJ     68
MT     68
WA     66
CO     66
NV     66
MS     65
RI     65
MA     65
AZ     64
MO     63
FL     63
ND     62
NM     62
ME     62
DE     61
OK     61
NE     61
SC     60
SD     60
KY     59
IL     58
NH     56
AR     55
GA     54
DC     54
TN     53
HI     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
```

In [15]:
```
# Check the customers'account length in days
df['account length'].value_counts()
```

Out[15]:
```
105    43
87     42
93     40
101    40
90     39
       ..
191     1
199     1
215     1
221     1
2       1
Name: account length, Length: 212, dtype: int64
```

In [16]:
```
# Check the different area codes
df['area code'].value_counts()
```

Out[16]:
```
415    1655
510     840
408     838
Name: area code, dtype: int64
```

In [17]:
```
# Check phone numbers
df['phone number'].value_counts()
```

Out[17]:
```
331-6270    1
397-9148    1
392-5296    1
380-3974    1
376-1713    1
           ..
336-1043    1
342-5906    1
376-4861    1
393-3635    1
365-6756    1
Name: phone number, Length: 3333, dtype: int64
```

In [18]:
```
# Check how many customers are subscribed to an international plan
df['international plan'].value_counts()
```

Out[18]:
```
no     3010
yes     323
Name: international plan, dtype: int64
```

In [19]:
```
# Check how many customers have a voice mail plan
df['voice mail plan'].value_counts()
```

Out[19]:
```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

In [20]:
```
# Check number of voice mail messages
df['number vmail messages'].value_counts()
```

Out[20]:
```
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
35      32
22      32
39      30
37      29
34      29
21      28
38      25
20      22
19      19
40      16
42      15
17      14
41      13
16      13
43       9
15       9
18       7
44       7
14       7
45       6
```

```
12       6
46       4
13       4
47       3
8        2
48       2
50       2
9        2
11       2
49       1
10       1
4        1
51       1
Name: number vmail messages, dtype: int64
```

In [21]:
```
# Check how much time customers spend on calls
df['total day minutes'].value_counts()
```

Out[21]:
```
174.5    8
159.5    8
154.0    8
175.4    7
162.3    7
        ..
199.9    1
105.8    1
125.6    1
179.8    1
270.8    1
Name: total day minutes, Length: 1667, dtype: int64
```

In [22]:
```
# Check the total daily calls customers make
df['total day calls'].value_counts()
```

Out[22]:
```
102    78
105    75
107    69
95     69
104    68
       ..
149     1
157     1
36      1
30      1
165     1
Name: total day calls, Length: 119, dtype: int64
```

In [23]:
```
# Check the total daily charge customers spend
df['total day charge'].value_counts()
```

Out[23]:
```
27.12    8
26.18    8
29.67    8
31.18    7
27.59    7
        ..
19.36    1
16.95    1
34.12    1
48.35    1
13.28    1
Name: total day charge, Length: 1667, dtype: int64
```

In [24]:
```
# Check the total evening minutes customers spend on calls
```

```
# Check the total evening minutes customers spend on calls
df['total eve minutes'].value_counts()
```

Out[24]:
```
169.9    9
230.9    7
209.4    7
201.0    7
220.6    7
        ..
335.0    1
258.9    1
134.7    1
318.8    1
317.2    1
Name: total eve minutes, Length: 1611, dtype: int64
```

In [25]:
```
# Check the total evening calls customers make
df['total eve calls'].value_counts()
```

Out[25]:
```
105    80
94     79
108    71
97     70
102    70
       ..
45      1
49      1
145     1
153     1
0       1
Name: total eve calls, Length: 123, dtype: int64
```

In [26]:
```
# Check the total evening charge customers incur
df['total eve charge'].value_counts()
```

Out[26]:
```
14.25    11
16.12    11
15.90    10
18.62     9
14.44     9
         ..
12.64     1
13.83     1
11.39     1
28.03     1
20.53     1
Name: total eve charge, Length: 1440, dtype: int64
```

In [27]:
```
# Check the total minutes customers spend on calls at night
df['total night minutes'].value_counts()
```

Out[27]:
```
210.0    8
214.6    8
197.4    8
191.4    8
188.2    8
        ..
132.3    1
306.2    1
293.5    1
271.7    1
182.6    1
Name: total night minutes, Length: 1591, dtype: int64
```

In [28]:
```
# Check the total number of calls customers make at night
df['total night calls'].value_counts()
```

Out[28]:
```
105    84
104    78
91     76
102    72
100    69
       ..
164     1
166     1
33      1
149     1
36      1
Name: total night calls, Length: 120, dtype: int64
```

In [29]:
```
# Check the total charges customers incur on calls at night
df['total night charge'].value_counts()
```

Out[29]:
```
9.66     15
9.45     15
8.88     14
8.47     14
7.69     13
         ..
14.65     1
6.46      1
3.94      1
15.74     1
6.14      1
Name: total night charge, Length: 933, dtype: int64
```

In [30]:
```
# Check total minutes customers spend on international calls
df['total intl minutes'].value_counts()
```

Out[30]:
```
10.0    62
11.3    59
9.8     56
10.9    56
10.1    53
        ..
18.9     1
1.3      1
2.7      1
2.6      1
3.1      1
Name: total intl minutes, Length: 162, dtype: int64
```

In [31]:
```
# Check total international calls customers make
df['total intl calls'].value_counts()
```

Out[31]:
```
3     668
4     619
2     489
5     472
6     336
7     218
1     160
8     116
9     109
10     50
11     28
0      18
```

```
12     15
13     14
15      7
14      6
18      3
16      2
19      1
17      1
20      1
Name: total intl calls, dtype: int64
```

In [32]:
```
# Check total charges customers incur on international calls
df['total intl charge'].value_counts()
```

Out[32]:
```
2.70    62
3.05    59
2.65    56
2.94    56
2.73    53
        ..
0.68     1
4.83     1
0.84     1
0.30     1
5.40     1
Name: total intl charge, Length: 162, dtype: int64
```

In [33]:
```
# Check the number of customer service calls made
df['customer service calls'].value_counts()
```

Out[33]:
```
1    1181
2     759
0     697
3     429
4     166
5      66
6      22
7       9
9       2
8       2
Name: customer service calls, dtype: int64
```

In [34]:
```
# Check the customers who churned(stopped using the service) and those who don't
df['churn'].value_counts()
```

Out[34]:
```
False    2850
True      483
Name: churn, dtype: int64
```

3. Data Preparation

In this section, we will prepare our data for modelling. We will check our data for and handle:

- Duplicate rows
- Missing values
- Conversion of categorical data to numeric
- Multicollinearity

In [35]:
```
# check if there are any duplicates
```

```
# check if there are any duplicates
df.duplicated().sum()
```

Out[35]: 0

In [36]:
```
# Find and count null entries
df.isna().sum()
```

Out[36]:
```
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
```

The data set has no missing values

In [37]:
```
# Check thenumber of columns and missing values
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
```

Dropping Unnecessary Columns

In [38]:
```
# Phone number is not important
df = df.drop("phone number", axis=1)
```

Converting Categorical Variables

In [39]:
```
# Converting categorical variables to numeric using One-Hot Encoding
df = pd.get_dummies(df, columns=["voice mail plan", "international plan"], drop_
```

Converting the Boolean to Numeric

In [40]:
```
# converting Churn to numeric format where 1 is True and 0 is False
df["churn"] = df['churn'].astype(int)
```

In [41]:
```
# check that everything is in order
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 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   number vmail messages   3333 non-null   int64  
 4   total day minutes       3333 non-null   float64
 5   total day calls         3333 non-null   int64  
 6   total day charge        3333 non-null   float64
 7   total eve minutes       3333 non-null   float64
 8   total eve calls         3333 non-null   int64  
 9   total eve charge        3333 non-null   float64
 10  total night minutes     3333 non-null   float64
 11  total night calls       3333 non-null   int64  
 12  total night charge      3333 non-null   float64
 13  total intl minutes      3333 non-null   float64
 14  total intl calls        3333 non-null   int64  
 15  total intl charge       3333 non-null   float64
 16  customer service calls  3333 non-null   int64  
 17  churn                   3333 non-null   int64  
 18  voice mail plan_yes     3333 non-null   uint8  
 19  international plan_yes  3333 non-null   uint8  
dtypes: float64(8), int64(9), object(1), uint8(2)
memory usage: 475.3+ KB
```

In [42]:
```
df.head()
```

Out[42]:

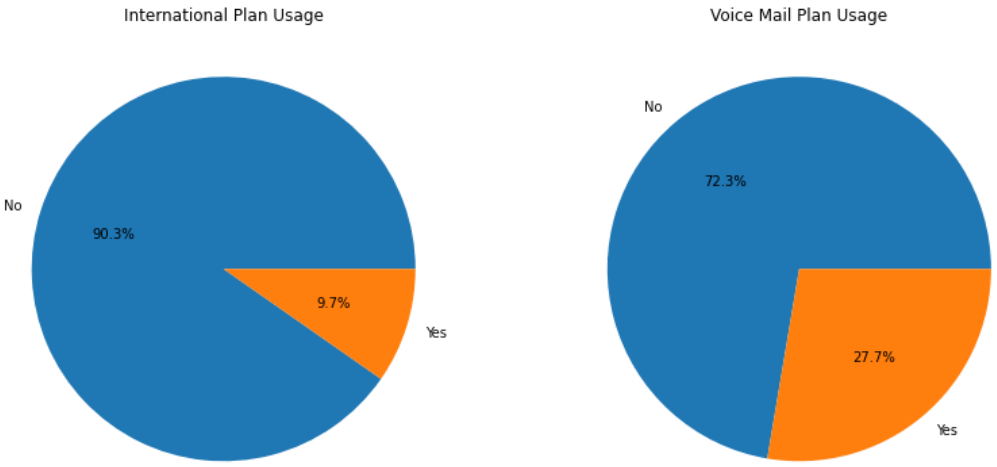
	state	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	t n min
0	KS	128	415	25	265.1	110	45.07	197.4	99	16.78	2
1	OH	107	415	26	161.6	123	27.47	195.5	103	16.62	2
2	NJ	137	415	0	243.4	114	41.38	121.2	110	10.30	1

3	OH	84	408	0	299.4	71	50.90	61.9	88	5.26	1
4	OK	75	415	0	166.7	113	28.34	148.3	122	12.61	1

Explanatory Data Analysis

Here, we will conduct explanatory data analysis to explore and understand the features. We will apply descriptive statistics and visualizations to identify the main patterns of the data and discover any relationships between the features and the target variable "churn".

In [43]:
```
# Assuming df is your DataFrame
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

# Pie chart for 'international plan'
y_international_plan = df['international plan_yes'].value_counts()
axes[0].pie(y_international_plan, labels=['No', 'Yes'], autopct='%1.1f%%')
axes[0].set_title('International Plan Usage')

# Pie chart for 'voice mail plan'
y_voice_mail_plan = df['voice mail plan_yes'].value_counts()
axes[1].pie(y_voice_mail_plan, labels=['No', 'Yes'], autopct='%1.1f%%')
axes[1].set_title('Voice Mail Plan Usage')

plt.show()
```

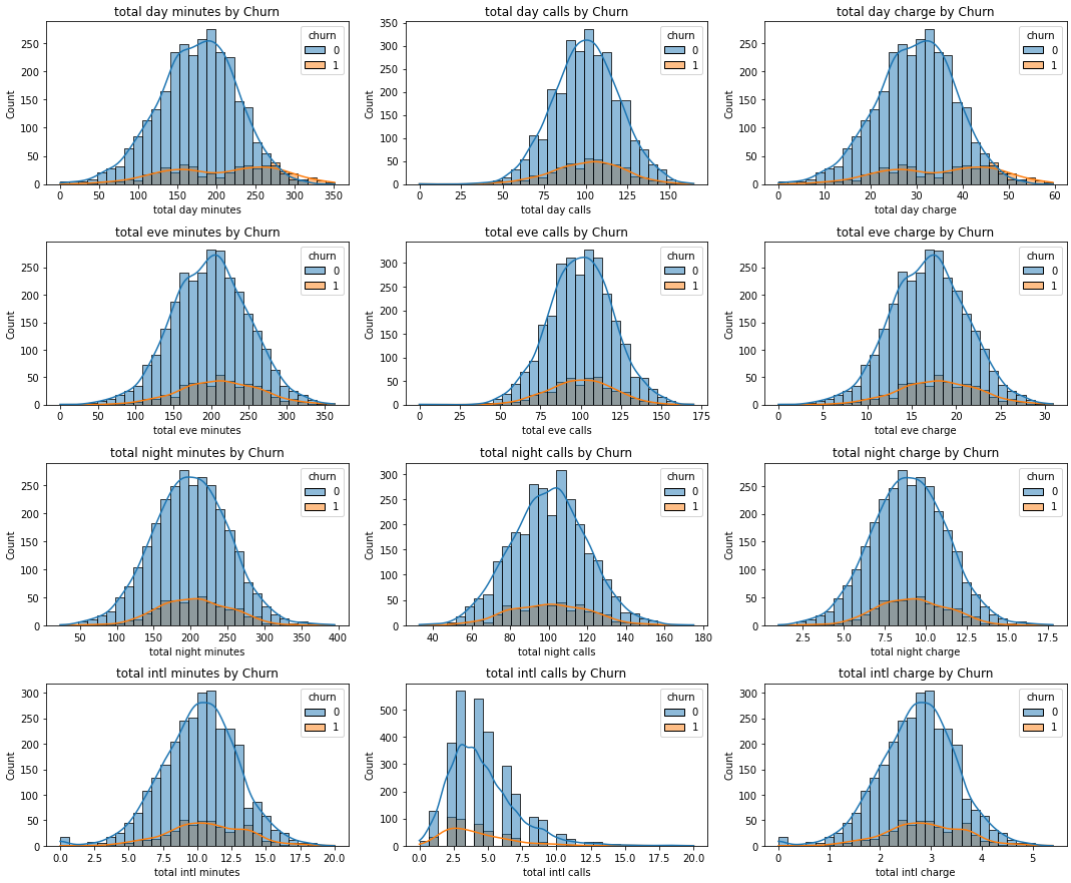


9.7% of the customers do use the international plans while 27.7% use the voice mail plan

To try and the answer the research question: *What are the main drivers of customer churn?*, we can check the relationship between different features and churn

In [44]:
```
# Relationship Between Call Charge Time and Churn

plt.figure(figsize=(15, 15))

# List of numerical features
call_charge_time = [ 'total day minutes', 'total day calls',
                     'total day charge', 'total eve minutes', 'total eve calls'
                     'total night minutes', 'total night calls', 'total night c
                     'total intl calls', 'total intl charge']
```

```
# Plot histograms for each numerical feature against the 'churn' variable
for i, feature in enumerate(call_charge_time, 1):
    plt.subplot(5, 3, i)
    sns.histplot(data=df, x=feature, hue='churn', bins=30, kde=True)
    plt.title(f'{feature} by Churn')

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



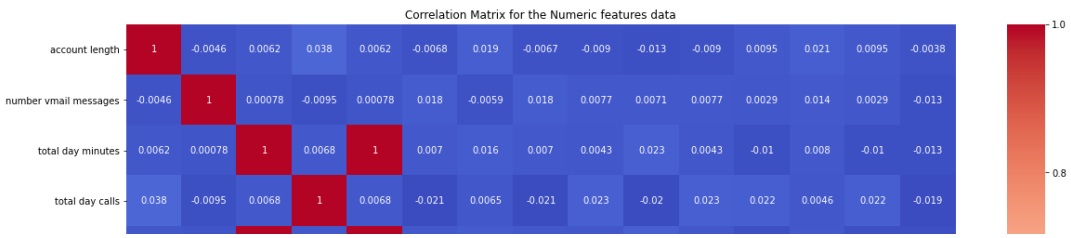
All the call charge time have a normal distribution apart from the International Calls

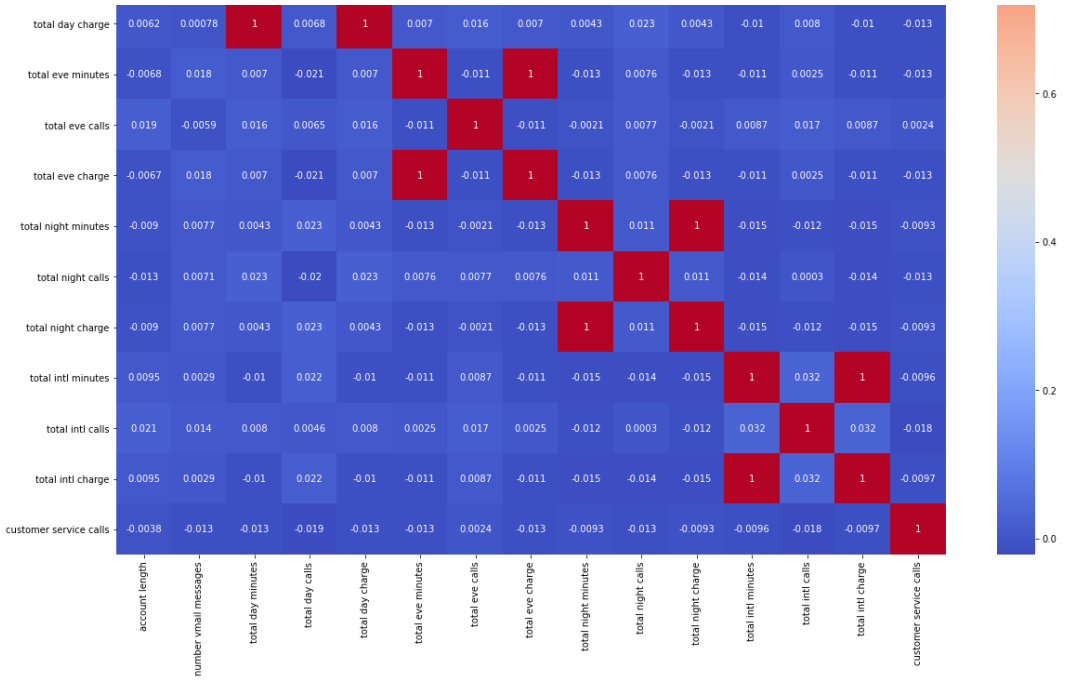
In [45]:

```
#Defining numeric features from the data set
numerical_features = ['account length', 'number vmail messages', 'total day minu
                      'total day charge', 'total eve minutes', 'total eve calls'
                      'total night minutes', 'total night calls', 'total night c
                      'total intl calls', 'total intl charge', 'customer service
#defining categorical features from the data set
categorical_features = ['state', 'area code', 'international plan', 'voice mail
```

In [46]:

```
# Check correlations between features
plt.figure(figsize=(20, 15))
sns.heatmap(df[numerical_features].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix for the Numeric features data')
plt.show()
```





In [47]:

```
# checking for multicolinearity

corr = df[numerical_features].corr().abs()

Multicollinear_Features = []
Multicollinear_Corr = []

# Function to check multicollinearity for a given feature
def check_multicollinearity(feature):
    for idx, correlation in corr[feature].items():
        if correlation >= 0.75 and idx != feature:
            Multicollinear_Features.append([feature, idx])
            Multicollinear_Corr.append(correlation)

# Iterate over features in the correlation matrix
for feature in corr.columns:
    check_multicollinearity(feature)

# Creating a DataFrame with multicollinearity information
MC_df = pd.DataFrame({'Correlations': Multicollinear_Corr, 'Features': Multicoll
MC_df = MC_df.sort_values(by='Correlations', ascending=False)


print(MC_df)
```

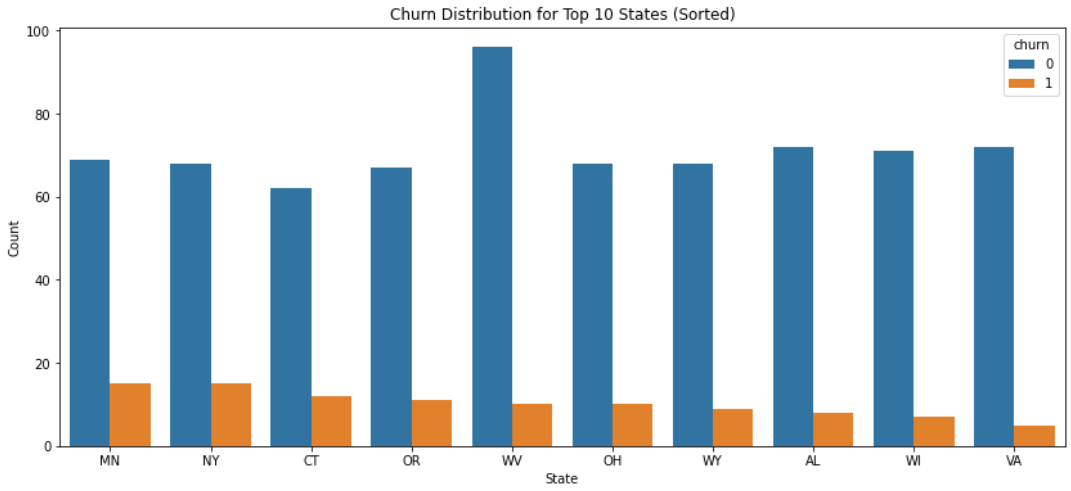
```
   Correlations                                  Features
0      1.000000      [total day minutes, total day charge]
1      1.000000      [total day charge, total day minutes]
2      1.000000      [total eve minutes, total eve charge]
3      1.000000      [total eve charge, total eve minutes]
4      0.999999  [total night minutes, total night charge]
5      0.999999  [total night charge, total night minutes]
6      0.999993    [total intl minutes, total intl charge]
7      0.999993    [total intl charge, total intl minutes]
```

The columns:

- total day minutes & total day charge
- total day charge & total day minutes
- total eve minutes & total eve charge
- total eve charge & total eve minutes

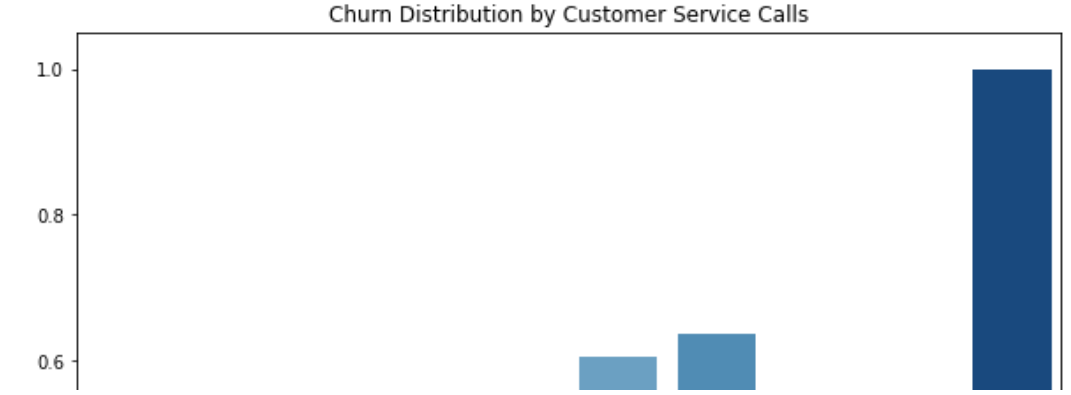
Have a high correlation

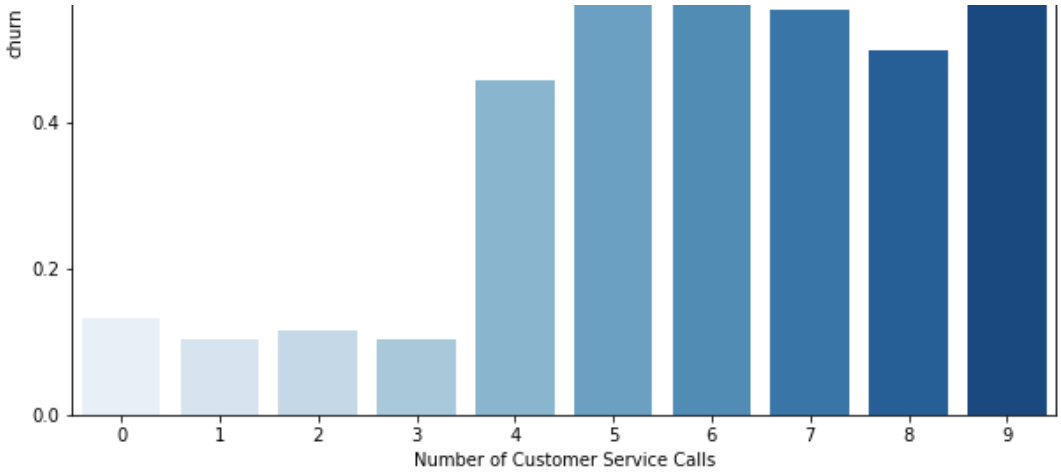
In [48]:
```
# Churn distribution based on the top 10 states
# Identifying the top 10 states based on churn counts and sorting them
top_states = df['state'].value_counts().nlargest(10).index

# Sort the top states in descending order based on churn counts
top_states = sorted(top_states, key=lambda state: df[df['state'] == state]['chur

# Filter the DataFrame for the top 10 states
df_top_states = df[df['state'].isin(top_states)]

# Plot the countplot for the top 10 states
plt.figure(figsize=(14, 6))
sns.countplot(x='state', hue='churn', data=df_top_states, order=top_states)
plt.title('Churn Distribution for Top 10 States (Sorted)')
plt.xlabel('State')
plt.ylabel('Count')
plt.show()
```



The states with the highest churn rates include MN, NY, CT. However, there isn't a definitive explanation for why certain states outperform others in this regard, prompting a shift to analyzing regions instead.

In [49]:
```
# churn distribution based on the customer service calls
plt.figure(figsize=(10, 8))
sns.barplot(x='customer service calls', y='churn', data=df, palette='Blues', ci=
plt.title('Churn Distribution by Customer Service Calls')
plt.xlabel('Number of Customer Service Calls')
plt.show()
```



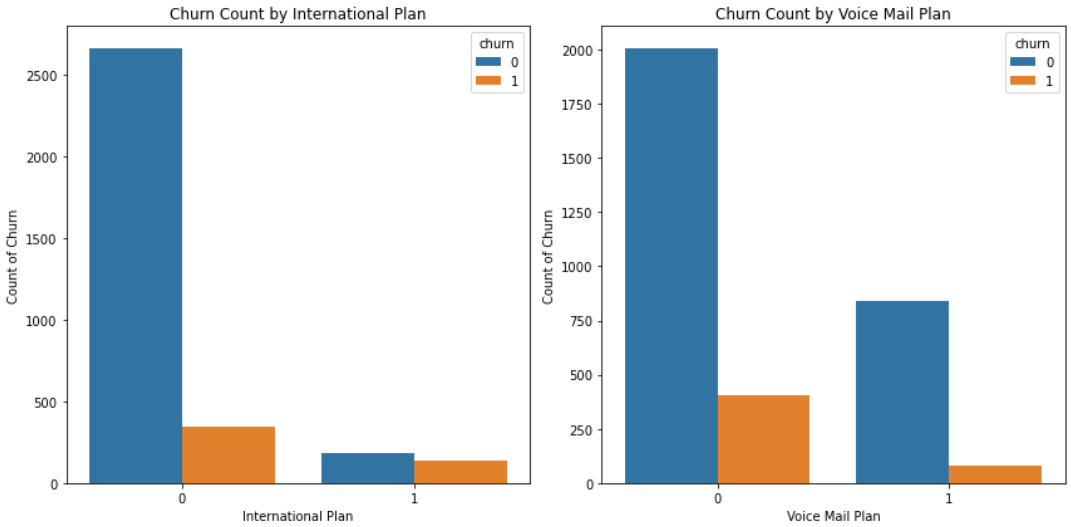


In [ ]:

As the number of customer service calls increases, the likelihood of churning also increases. Above 4 calls, the churn rate increases.

In [50]:

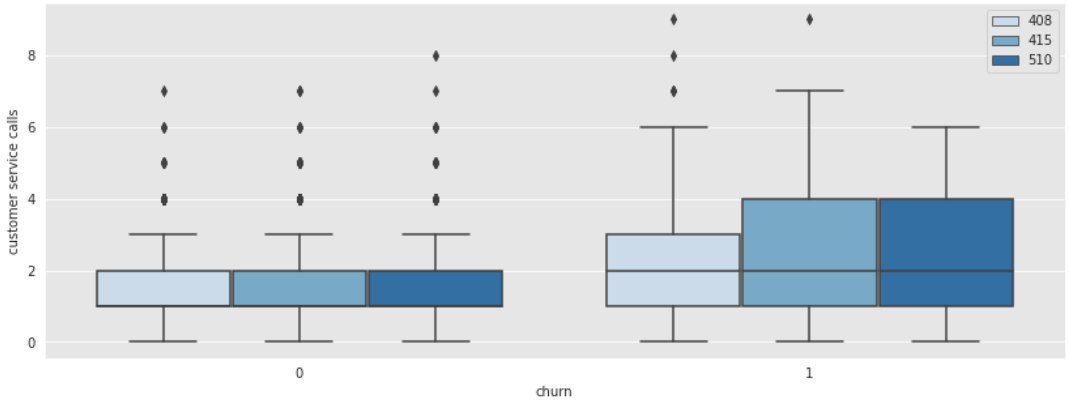
```
# churn count by international plan and voice mail plan
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot Churn Count by International Plan
sns.countplot(x='international plan_yes', data=df, hue='churn', ax=axes[0])
axes[0].set_xlabel('International Plan')
axes[0].set_ylabel('Count of Churn')
axes[0].set_title('Churn Count by International Plan')

# Plot Churn Count by Voice Mail Plan
sns.countplot(x='voice mail plan_yes', data=df, hue='churn', ax=axes[1])
axes[1].set_xlabel('Voice Mail Plan')
axes[1].set_ylabel('Count of Churn')
axes[1].set_title('Churn Count by Voice Mail Plan')

# Adjust layout to prevent overlap
plt.tight_layout()

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



People who do not have an international plan and a voice mail plane have a higher churn

rate

In [61]:
```
# churn rate based on the customer service calls and the area codes
plt.figure(figsize=(14,5))
sns.boxplot(data=df,x='churn',y='customer service calls',hue='area code', palett
plt.legend(loc='upper right');
```



Area Code 415 & 510 have a high customer service calls which in turn have a high churn rate

In [52]:
```
# Average cost per minute
cost_per_minute_day = df['total day charge'] / df['total day minutes']
cost_per_minute_evening = df['total eve charge'] / df['total eve minutes']
cost_per_minute_night = df['total night charge'] / df['total night minutes']
cost_per_minute_international = df['total intl charge'] / df['total intl minutes

average_cost_day = cost_per_minute_day.mean()
average_cost_evening = cost_per_minute_evening.mean()
average_cost_night = cost_per_minute_night.mean()
average_cost_international = cost_per_minute_international.mean()


print(f'Average Cost per Minute (Day): {average_cost_day:.2f}')
print(f'Average Cost per Minute (Evening): {average_cost_evening:.2f}')
print(f'Average Cost per Minute (Night): {average_cost_night:.2f}')
print(f'Average Cost per Minute (International): {average_cost_international:.2f
```

```
Average Cost per Minute (Day): 0.17
Average Cost per Minute (Evening): 0.09
Average Cost per Minute (Night): 0.05
Average Cost per Minute (International): 0.27
```

The averages for making the calls per minute ranges from 0.05(Night time) to 0.27(International Calls)

Based on the features that we have investigated above, we can now train some machine learning models that can predict the likelihood of a customer churning.

4.Modeling

Here we train various models using the dataset provided. The machine learning models explored are:

1. [Logistic regression](4.1 LogisticRegression)

[4.1 LogisticRegression)()

In [53]:
```
#import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

All features except State

Model One

In [54]:
```
# Drop state and the predictor, churn
X= df.drop(columns=["state","churn"], axis =1 )
y = df["churn"]

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

#Standerdize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Instantiate the model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran

# fit the model
model_log = logreg.fit(X_train_scaled, y_train)

# predict the model
y_hat_train = logreg.predict(X_train_scaled)
y_hat_test = logreg.predict(X_test_scaled)
```

In [55]:
```
# Get the precision score, recall score, accuracy score and F1 score for both the
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc

print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')
print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')
print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')
print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))


# Additional: Confusion Matrix
# Import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_hat_test)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Training Precision:  0.2896613190730838
Testing Precision:  0.3079584775086505
```

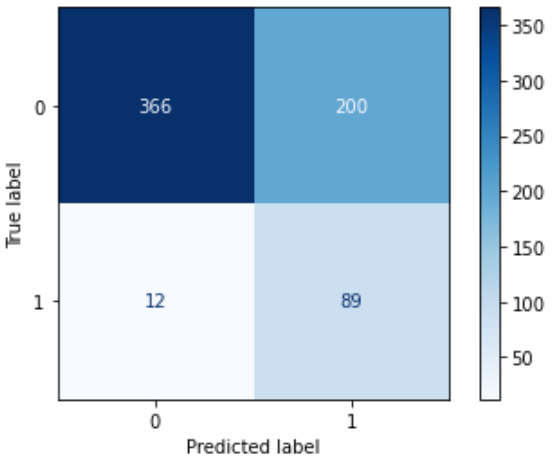
```
Training Recall:  0.8507853403141361
Testing Recall:  0.8811881188118812



Training Accuracy:  0.6796699174793699
Testing Accuracy:  0.6821589205397302



Training F1-Score:  0.43218085106382986
Testing F1-Score:  0.4564102564102564

Confusion Matrix:
 [[366 200]
 [ 12  89]]
```

In [56]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues)
```

Out[56]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f578ec8f910
>
```



In [57]:

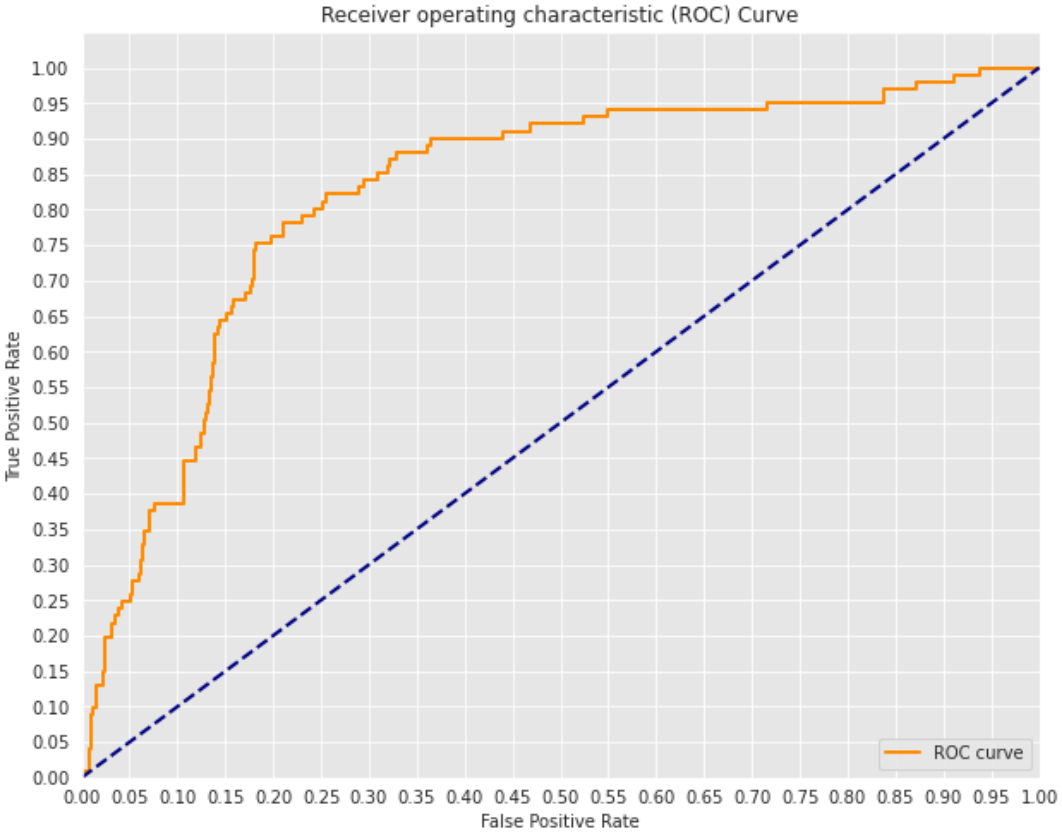
```
from sklearn.metrics import roc_curve, auc

#calculate the probability scores
y_score = logreg.fit(X_train_scaled, y_train).decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

print('AUC: {}'.format(auc(fpr, tpr)))
```

```
AUC: 0.8234090193471644
```

In [58]:

```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
plt.figure(figsize=(10, 8))
lw = 2
logreg_fpr1 = fpr
logreg_tpr1 = tpr
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
```

```
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



In order to deal with the class imbalance- Use SMOTE Method to resample the dataset to make sure there is equal representation of customers who churn wnd those who do not

In [62]:
```
from imblearn.over_sampling import SMOTE

# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

0    2850
1     483
Name: churn, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    2284
0    2284
Name: churn, dtype: int64
```

Model 2- Resampled

In [63]:

```
# Resample the dataset
sm = SMOTE()
X_train_resampled, y_train_resampled = sm.fit_resample(X_train_scaled, y_train)

# Check the counts of the resampled target variable
print(pd.Series(y_train_resampled).value_counts())

# creates an instance of the logistic regression model after balancing
model_log_resampled = LogisticRegression(fit_intercept=False, C=1e12, solver='li

# training the model after balancing the training set
resampled_fit = model_log_resampled.fit(X_train_resampled, y_train_resampled)

# make predictions on the balanced training set
y_hat_train_resampled = model_log_resampled.predict(X_train_resampled)

train_residuals_resampled = np.abs(y_train_resampled - y_hat_train_resampled)
```

```
1    2284
0    2284
Name: churn, dtype: int64
```

In [64]:

```
from imblearn.over_sampling import SMOTE

# Convert to NumPy arrays without feature names
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Apply SMOTE to the train set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai

# Instantiate the model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran

# fit the model
model_log = logreg.fit(X_train_resampled, y_train_resampled)

# Predict on the resampled train & test set
y_hat_train_resampled = logreg.predict(X_train_resampled)
y_hat_test_resampled = logreg.predict(X_test_scaled)

# Calculate precision, accuracy, recall, and F1 score
print('Training Precision: ', precision_score(y_train_resampled, y_hat_train_res
print('Testing Precision: ', precision_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Recall: ', recall_score(y_train_resampled, y_hat_train_resampled
print('Testing Recall: ', recall_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Accuracy: ', accuracy_score(y_train_resampled, y_hat_train_resam
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training F1-Score: ', f1_score(y_train_resampled, y_hat_train_resampled))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test_resampled))


from sklearn.metrics import confusion_matrix
print('\n\n')
# confusion matrix for training set
conf_matrix_resampled = confusion_matrix(y_train_resampled, y_hat_train_resample
print("Confusion Matrix for training set (after SMOTE):\n", conf_matrix_resample
print('\n\n')
# confusion matrix for testing set
conf_matrix_resampled = confusion_matrix(y_test, y_hat_test_resampled)
print("Confusion Matrix for testing set (after SMOTE):\n", conf_matrix_resampled
```

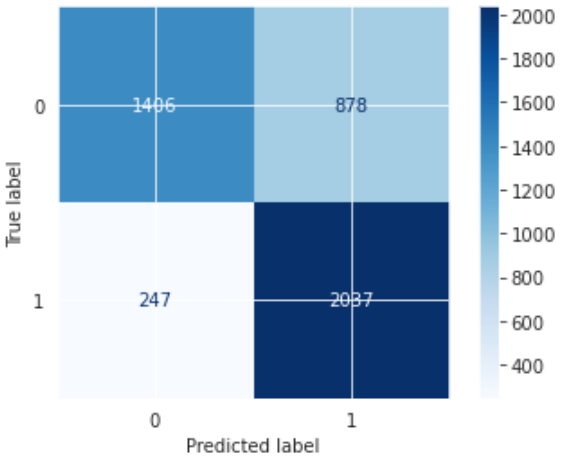
```
Training Precision:  0.6987993138936535
Testing Precision:  0.30132450331125826



Training Recall:  0.8918563922942206
Testing Recall:  0.900990099009901



Training Accuracy:  0.7537215411558669
Testing Accuracy:  0.6686656671664168



Training F1-Score:  0.7836122331217541
Testing F1-Score:  0.4516129032258065



Confusion Matrix for training set (after SMOTE):
 [[1406  878]
 [ 247 2037]]



Confusion Matrix for testing set (after SMOTE):
 [[355 211]
 [ 10  91]]
```

In [65]:

```
# Print confusion matrix for training set
cnf_matrix = confusion_matrix(y_train_resampled, y_hat_train_resampled)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues)
```

Out[65]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f578ec29610
>
```

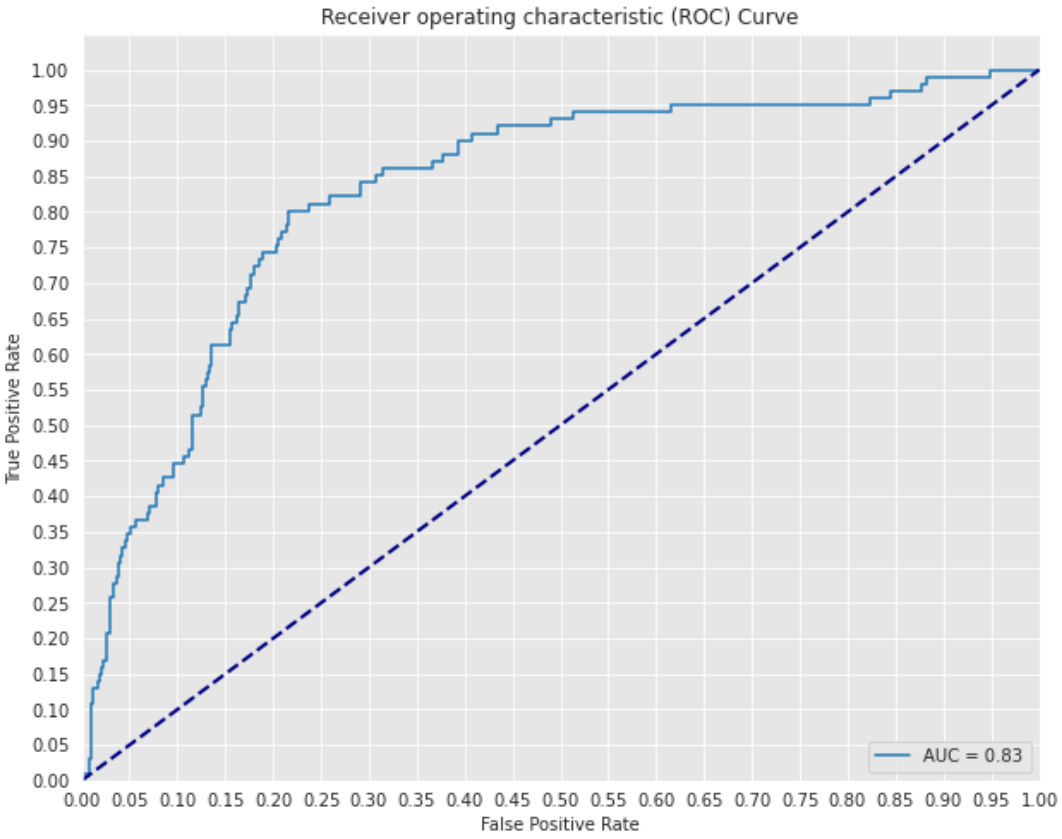


In [66]:

```
# Fit a model
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver ='liblinear', ra
model_log = logreg.fit(X_train_resampled, y_train_resampled)
print(model_log)

# Predict
y_hat_test = logreg.predict(X_test_scaled)
```

```
y_score = logreg.decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

print('AUC: {}'.format(auc(fpr, tpr)))
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(roc_auc))
lw = 2

logreg_fpr1_resampled = fpr
logreg_tpr1_resampled = tpr

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
LogisticRegression(C=1000000000000.0, fit_intercept=False, random_state=42,
                   solver='liblinear')
AUC: 0.8297064688801036
```



- Precision for the training dataset has increased significantly after SMOTE, indicating fewer false positives. However, there's a slight decreased in the precision for the testing dataset.

- Recall for both the training and testing datasets has increased after SMOTE, indicating

better identification of positive cases.

- Accuracy for the training dataset has increased, but the accuracy for the testing dataset has slightly decreased after SMOTE.
- F1-score for both the training and testing datasets has increased after SMOTE, indicating a better balance between precision and recall.

In summary, balancing the data using SMOTE has positively impacted the model's ability to correctly classify instances of the minority class, leading to improvements in precision, recall, and overall model performance.

Dropping Irrelevant columns

In [67]:
```
X= df.drop(['area code', 'state','account length','total day charge','total eve
y = df['churn']

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

#Standerdize the features
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)

# Instantiate the mode
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran

# fit the model
model_log = logreg.fit(X_train_scaled, y_train)
```

In [68]:
```
# Confirm the features
X.columns
```

Out[68]:
```
Index(['number vmail messages', 'total day minutes', 'total day calls',
       'total eve minutes', 'total eve calls', 'total night minutes',
       'total night calls', 'total intl minutes', 'total intl calls',
       'customer service calls', 'voice mail plan_yes',
       'international plan_yes'],
      dtype='object')
```

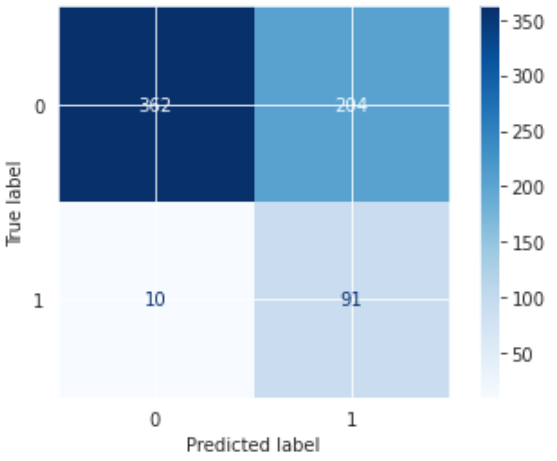
In [69]:
```
# Scale the data
y_hat_train = logreg.predict(X_train_scaled)
y_hat_test = logreg.predict(X_test_scaled)

# calculating the metrics

print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
```

```
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))

# Additional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_hat_test)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Training Precision:  0.2867256637168142
Testing Precision:  0.30847457627118646



Training Recall:  0.8481675392670157
Testing Recall:  0.900990099009901



Training Accuracy:  0.6759189797449362
Testing Accuracy:  0.679160419790105



Training F1-Score:  0.4285714285714286
Testing F1-Score:  0.4595959595959596

Confusion Matrix:
 [[362 204]
 [ 10  91]]
```

In [71]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues);
```



In [72]:

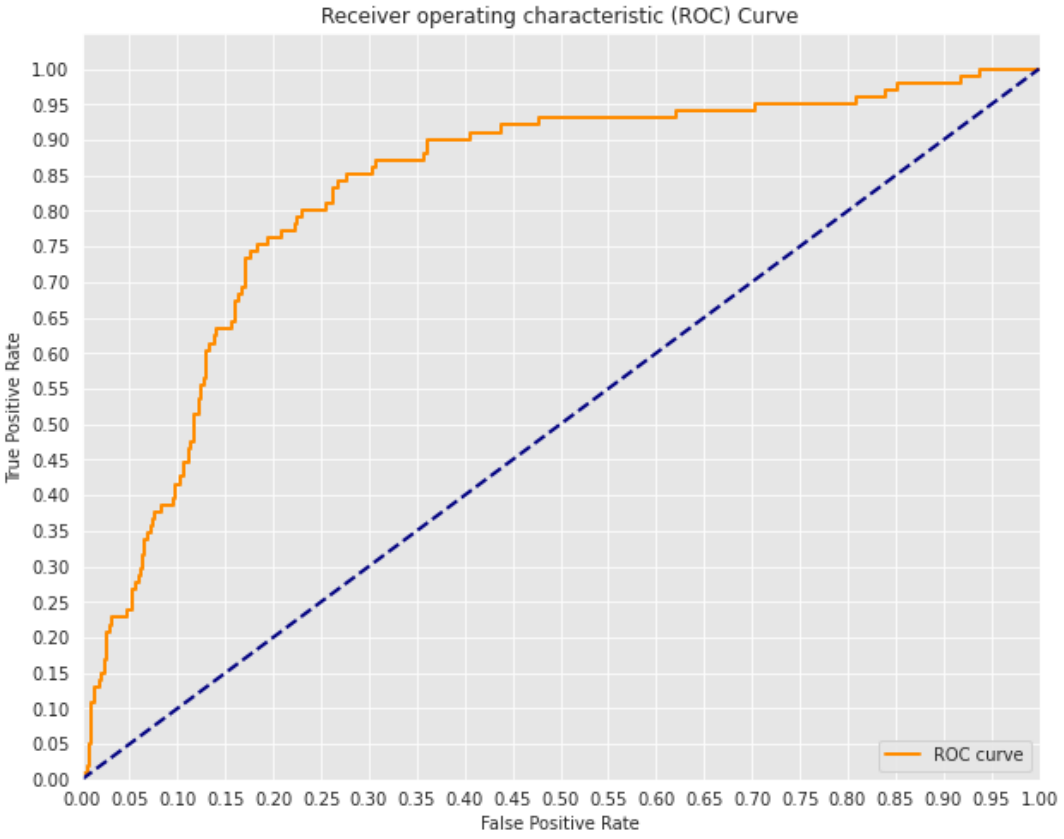
```
#calculate the probability scores of each of the datapoints:
y_score = logreg.fit(X_train_scaled, y_train).decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

print('AUC: {}'.format(auc(fpr, tpr)))
```

```
AUC: 0.8271000244900816
```

In [73]:

```
# Plot the ROC curve
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

# plotting the ROC curve
plt.figure(figsize=(10, 8))
```

```
lw = 2
logreg_fpr2= fpr
logreg_tpr2 = tpr
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



In [74]:

```
# performing SMOTE

# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai
# Preview synthetic sample class distribution
print('------------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

0    2850
1     483
Name: churn, dtype: int64
------------------------------------------
Synthetic sample class distribution:
```

```
1    2284
0    2284
Name: churn, dtype: int64
```

In [75]:

```
# Check the counts of the resampled target variable
print(pd.Series(y_train_resampled).value_counts())

# creates an instance of the logistic regression model after balancing
model_log_resampled = LogisticRegression(fit_intercept=False, C=1e12, solver='li

# training the model after balancing the training set
resampled_fit = model_log_resampled.fit(X_train_resampled, y_train_resampled)

# make predictions on the balanced training set
y_hat_train_resampled = model_log_resampled.predict(X_train_resampled)

train_residuals_resampled = np.abs(y_train_resampled - y_hat_train_resampled)

# Apply SMOTE to the test set
smote = SMOTE(random_state=42)
X_test_resampled, y_test_resampled = smote.fit_resample(X_train_scaled, y_train)

# Predict on the resampled test set
y_hat_test_resampled = logreg.predict(X_test_resampled)

# Calculate precision, accuracy, recall, and F1 score
precision_resampled = precision_score(y_test_resampled, y_hat_test_resampled)
accuracy_resampled = accuracy_score(y_test_resampled, y_hat_test_resampled)
recall_resampled = recall_score(y_test_resampled, y_hat_test_resampled)
f1_resampled = f1_score(y_test_resampled, y_hat_test_resampled)

# Print the results
print("Precision Score (after SMOTE):", precision_resampled)
print("Accuracy Score (after SMOTE):", accuracy_resampled)
print("Recall Score (after SMOTE):", recall_resampled)
print("F1 Score (after SMOTE):", f1_resampled)


from sklearn.metrics import confusion_matrix

conf_matrix_resampled = confusion_matrix(y_test_resampled, y_hat_test_resampled)

# Print the confusion matrix
print("Confusion Matrix (after SMOTE):\n", conf_matrix_resampled)
```

```
1    2284
0    2284
Name: churn, dtype: int64
Precision Score (after SMOTE): 0.7131672597864769
Accuracy Score (after SMOTE): 0.7622591943957968
Recall Score (after SMOTE): 0.8774080560420315
F1 Score (after SMOTE): 0.7868080094228503
Confusion Matrix (after SMOTE):
 [[1478  806]
 [ 280 2004]]
```

In [76]:

```
# Convert to NumPy arrays without feature names
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Apply SMOTE to the train set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai

# Instantiate the model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran
```

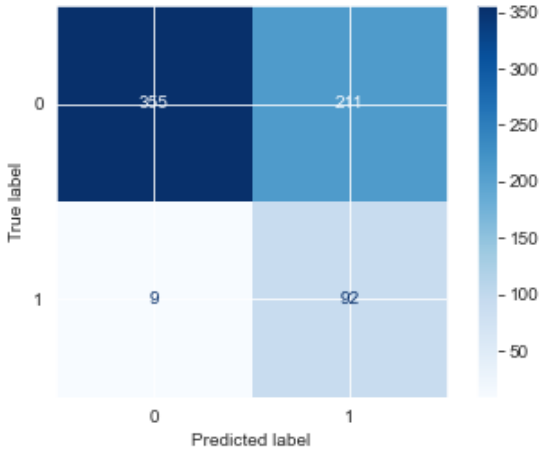
```
# fit the model
model_log = logreg.fit(X_train_resampled, y_train_resampled)

# Predict on the resampled train & test set
y_hat_train_resampled = logreg.predict(X_train_resampled)
y_hat_test_resampled = logreg.predict(X_test_scaled)

# Calculate precision, accuracy, recall, and F1 score
print('Training Precision: ', precision_score(y_train_resampled, y_hat_train_res
print('Testing Precision: ', precision_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Recall: ', recall_score(y_train_resampled, y_hat_train_resampled
print('Testing Recall: ', recall_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Accuracy: ', accuracy_score(y_train_resampled, y_hat_train_resam
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training F1-Score: ', f1_score(y_train_resampled, y_hat_train_resampled))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test_resampled))


from sklearn.metrics import confusion_matrix
print('\n\n')
# confusion matrix for training set
conf_matrix_resampled = confusion_matrix(y_train_resampled, y_hat_train_resample
print("Confusion Matrix for training set (after SMOTE):\n", conf_matrix_resample
print('\n\n')
# confusion matrix for testing set
conf_matrix_resampled = confusion_matrix(y_test, y_hat_test_resampled)
print("Confusion Matrix for testing set (after SMOTE):\n", conf_matrix_resampled
```

```
Training Precision:  0.695280437756498
Testing Precision:  0.30363036303630364



Training Recall:  0.8901050788091068
Testing Recall:  0.9108910891089109



Training Accuracy:  0.75
Testing Accuracy:  0.6701649175412294



Training F1-Score:  0.7807219662058371
Testing F1-Score:  0.45544554455445546



Confusion Matrix for training set (after SMOTE):
 [[1393  891]
 [ 251 2033]]



Confusion Matrix for testing set (after SMOTE):
 [[355 211]
 [  9  92]]
```

In [76]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test_resampled)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues);
```

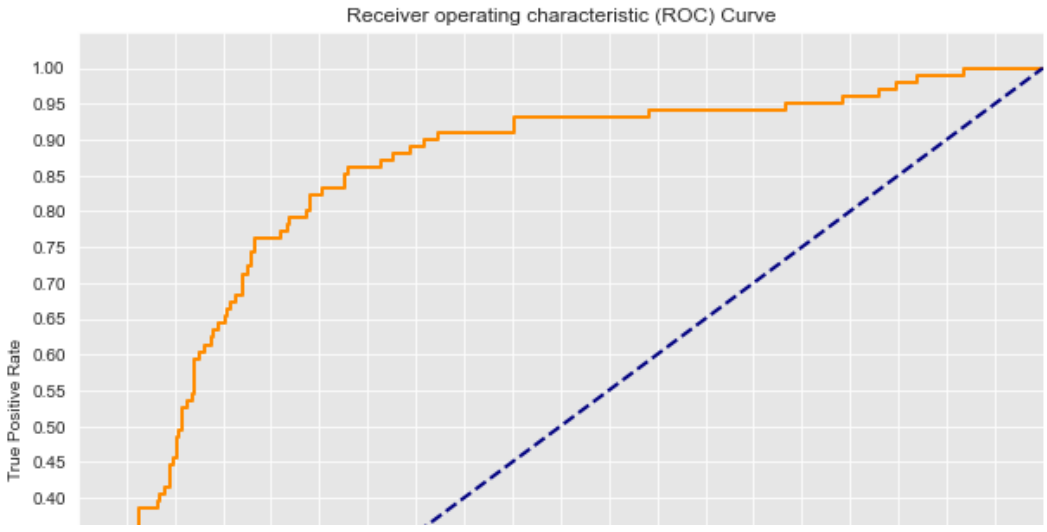
Out[76]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x240e14ecf10>

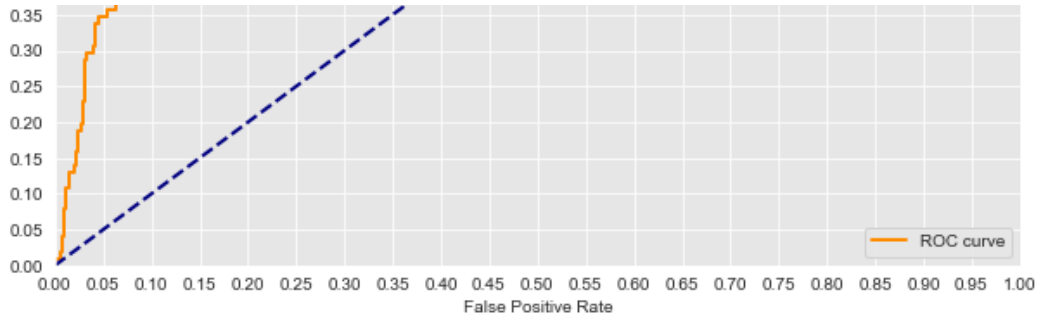


In [77]:

```
# Plotting the ROC curve
y_hat_test = logreg.predict(X_test_scaled)

y_score = logreg.decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
logreg_fpr2_resampled = fpr
logreg_tpr2_resampled = tpr
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

AUC: 0.8355141167827029





The curve is closer to the top-left corner, with a high AUC value indicating the classifier has a high true positive rate and a low false positive rate. This indicates a better performance of the model after balancing.

Multicolinearity features

Model 1

In [78]:
```
X= df.drop(columns=['state','churn', 'total day minutes', 'total eve minutes', '
y = df['churn']

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

#Standerdize the features
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)

# Instantiate the mode
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran

# fit the model
model_log = logreg.fit(X_train_scaled, y_train)
```

In [79]:
```
y_hat_train = logreg.predict(X_train_scaled)
y_hat_test = logreg.predict(X_test_scaled)

from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc

print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))

# Additional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_hat_test)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Training Precision:  0.29103815439219166
Testing Precision:  0.31010452961672474
```

```
Training Recall:  0.8586387434554974
Testing Recall:  0.8811881188118812



Training Accuracy:  0.6800450112528132
Testing Accuracy:  0.6851574212893553



Training F1-Score:  0.43472498343273697
Testing F1-Score:  0.4587628865979381

Confusion Matrix:
 [[368 198]
 [ 12  89]]
```

In [80]:

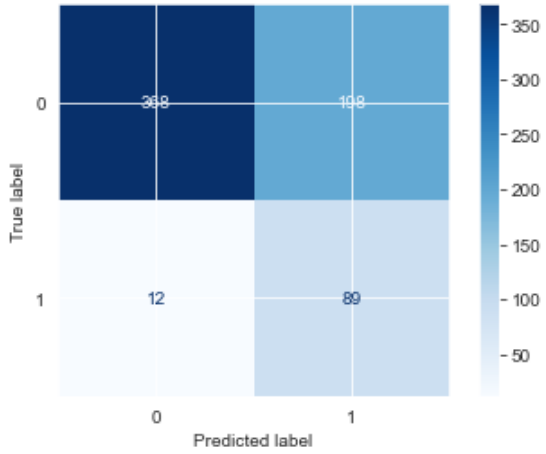
```
# predict the model
y_hat_train = logreg.predict(X_train_scaled)
y_hat_test = logreg.predict(X_test_scaled)

# calculating the metrics
print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))
```

```
Training Precision:  0.29103815439219166
Testing Precision:  0.31010452961672474



Training Recall:  0.8586387434554974
Testing Recall:  0.8811881188118812



Training Accuracy:  0.6800450112528132
Testing Accuracy:  0.6851574212893553



Training F1-Score:  0.43472498343273697
Testing F1-Score:  0.4587628865979381
```

In [81]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues)
```

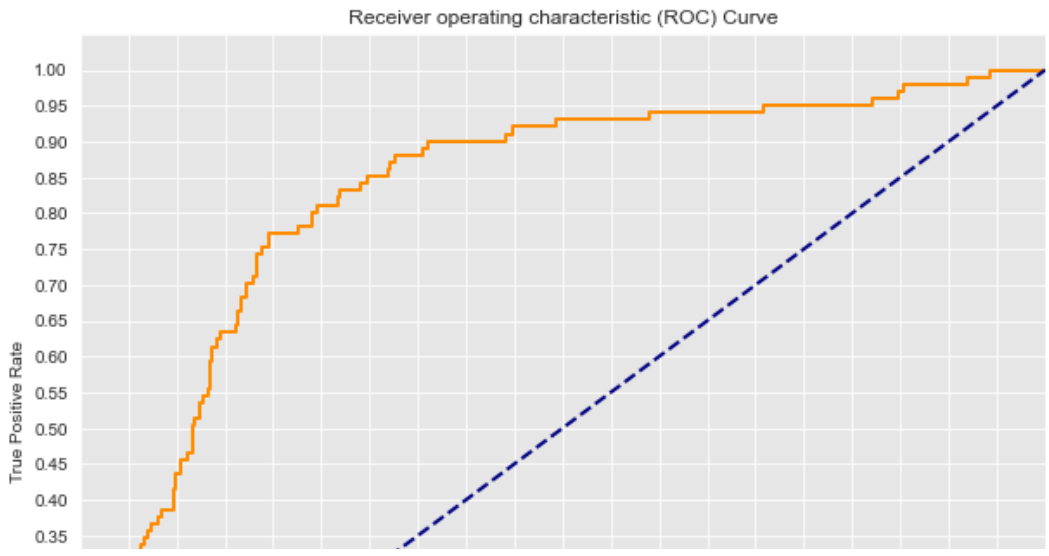
Out[81]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x240e705f070>



In [82]:
```
#calculate the probability scores of each of the datapoints:
y_score = logreg.fit(X_train_scaled, y_train).decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

print('AUC: {}'.format(auc(fpr, tpr)))
```

```
AUC: 0.824843438407445
```

In [83]:
```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

# Plotting the ROC curve
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')

logreg_fpr3 = fpr
logreg_tpr3 = tpr
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



In [77]:
```
# performing SMOTE

# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

0    2850
1     483
Name: churn, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    2284
0    2284
Name: churn, dtype: int64
```

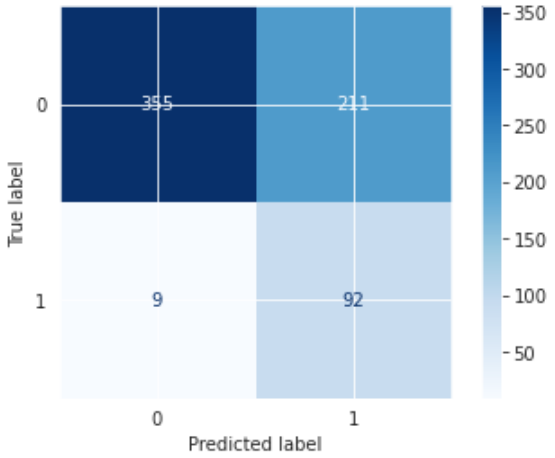
In [78]:
```
# Convert to NumPy arrays without feature names
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Apply SMOTE to the train set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai

# Instantiate the model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear', ran

# fit the model
model_log = logreg.fit(X_train_resampled, y_train_resampled)

# Predict on the resampled train & test set
y_hat_train_resampled = logreg.predict(X_train_resampled)
y_hat_test_resampled = logreg.predict(X_test_scaled)

# Calculate precision, accuracy, recall, and F1 score
print('Training Precision: ', precision_score(y_train_resampled, y_hat_train_res
print('Testing Precision: ', precision_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Recall: ', recall_score(y_train_resampled, y_hat_train_resampled
print('Testing Recall: ', recall_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training Accuracy: ', accuracy_score(y_train_resampled, y_hat_train_resam
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test_resampled))
print('\n\n')
print('Training F1-Score: ', f1_score(y_train_resampled, y_hat_train_resampled))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test_resampled))
```

```
from sklearn.metrics import confusion_matrix
print('\n\n')
# confusion matrix for training set
conf_matrix_resampled = confusion_matrix(y_train_resampled, y_hat_train_resample
print("Confusion Matrix for training set (after SMOTE):\n", conf_matrix_resample
print('\n\n')
# confusion matrix for testing set
conf_matrix_resampled = confusion_matrix(y_test, y_hat_test_resampled)
print("Confusion Matrix for testing set (after SMOTE):\n", conf_matrix_resampled
```

```
Training Precision:  0.695280437756498
Testing Precision:  0.30363036303630364



Training Recall:  0.8901050788091068
Testing Recall:  0.9108910891089109



Training Accuracy:  0.75
Testing Accuracy:  0.6701649175412294



Training F1-Score:  0.7807219662058371
Testing F1-Score:  0.45544554455445546



Confusion Matrix for training set (after SMOTE):
 [[1393  891]
 [ 251 2033]]



Confusion Matrix for testing set (after SMOTE):
 [[355 211]
 [  9  92]]
```

In [79]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test_resampled)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model_
disp.plot(cmap=plt.cm.Blues);
```

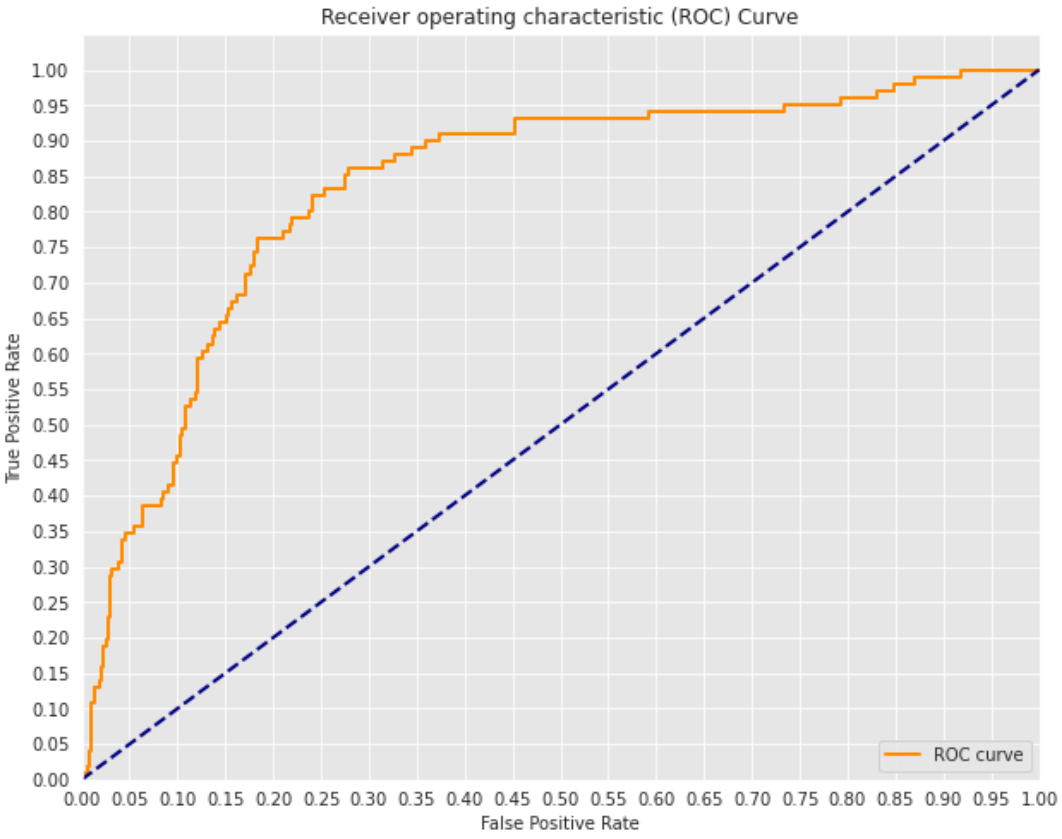


In [80]:

```
# Plotting the ROC curve
```

```
y_hat_test = logreg.predict(X_test_scaled)

y_score = logreg.decision_function(X_test_scaled)

fpr, tpr, thresholds = roc_curve(y_test, y_score)

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')

logreg_fpr3_resampled = fpr
logreg_tpr3_resampled = tpr

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.8355141167827029
```



ii) K-Nearest Neighbors (KNN) Model

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

In [82]:

```
# We defines predictors
X = df.drop(columns=["state","churn"], axis =1 )

# Define target variable
y = df["churn"]

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

#Standerdize the features
scaler = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)

#Convert into a DataFrame
X_train_scaled = pd.DataFrame(X_train_scaled, columns=[X_train.columns])
X_test_scaled = pd.DataFrame(X_test_scaled, columns=[X_train.columns])

# Instantiate the mode
knn = KNeighborsClassifier(n_neighbors=3)

# fit the model
knn.fit(X_train_scaled, y_train)

# Calculate the probability scores of each datapoint for the positive class (cla
y_score = knn.predict_proba(X_test_scaled)[:, 1]
y_train_score = knn.predict_proba(X_train_scaled)[:, 1]

# predict the model
y_hat_train = knn.predict(X_train_scaled)
y_hat_test = knn.predict(X_test_scaled)
```

In [83]:

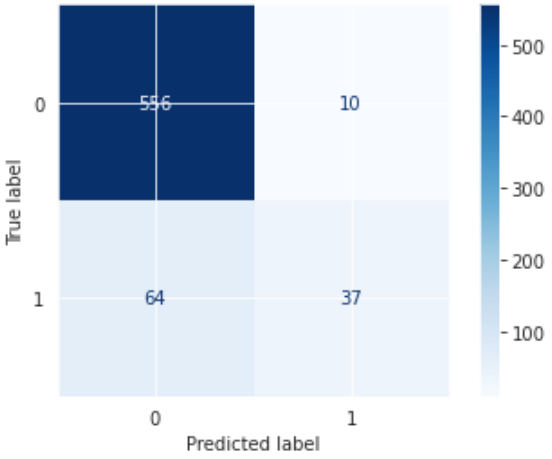
```
# calculating the metrics

print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))

# Additional: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_hat_test)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Training Precision:  0.9016393442622951
Testing Precision:  0.7872340425531915



Training Recall:  0.5759162303664922
Testing Recall:  0.36633663366336633



Training Accuracy:  0.9302325581395349
```

```
Testing Accuracy:  0.889055472263868



Training F1-Score:  0.7028753993610224
Testing F1-Score:  0.5

Confusion Matrix:
 [[556  10]
 [ 64  37]]
```

In [86]:

```
# Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_test)

disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=knn.cl
disp.plot(cmap=plt.cm.Blues);
```



In [88]:

```
# calculating the AUC value
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_score)
fpr, tpr, thresholds = roc_curve(y_test, y_score)

roc_auc_train = auc(fpr_train, tpr_train)
roc_auc = auc(fpr, tpr)
print('Train AUC: {}'.format(auc(fpr_train, tpr_train)))
print('Test AUC: {}'.format(auc(fpr, tpr)))
```

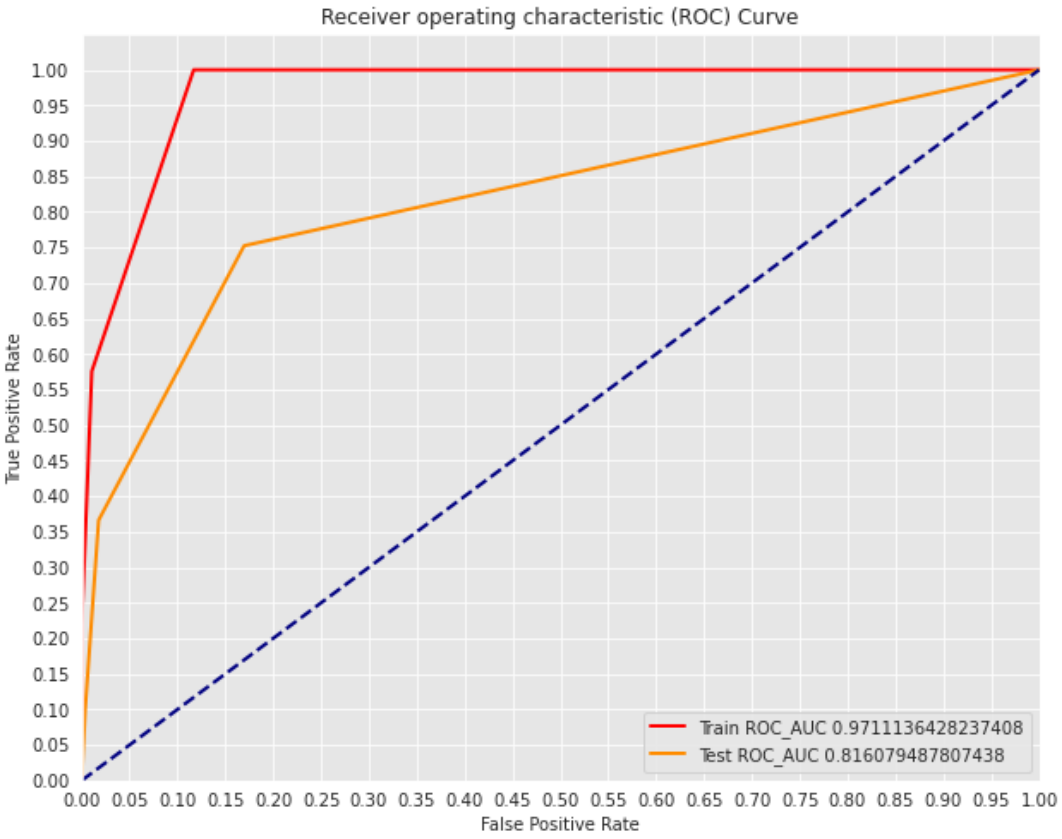
```
Train AUC: 0.9711136428237408
Test AUC: 0.816079487807438
```

Performance looks better on training set than the test set

In [90]:

```
# plotting the ROC curve

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr_train, tpr_train, color='red',
         lw=lw, label=f'Train ROC_AUC {roc_auc_train }')

plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label=f'Test ROC_AUC {roc_auc }')

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Resampling

In [91]:
```
# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

0    2850
1     483
Name: churn, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    2284
0    2284
Name: churn, dtype: int64
```

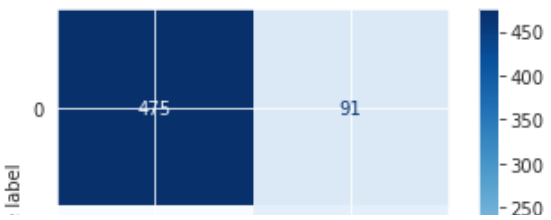
In [93]:
```
# Apply SMOTE to the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_trai

knn_resampled = KNeighborsClassifier(n_neighbors=3)
```

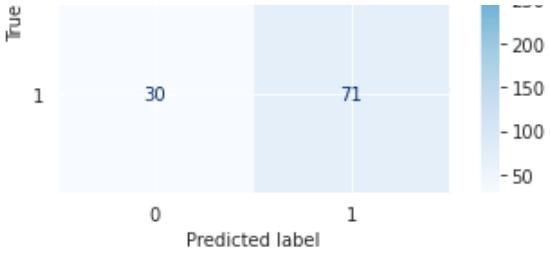
```
knn_resampled.fit(X_train_resampled, y_train_resampled)

# Predict on the resampled test set
y_hat_test_resampled = knn_resampled.predict(X_test_scaled)
y_hat_train_resampled = knn_resampled.predict(X_train_resampled)


print('Training Resampled Precision: ', precision_score(y_train_resampled, y_hat
print('Testing Resampled Precision: ', precision_score(y_test, y_hat_test_resamp
print('\n\n')

print('Training Resampled Recall: ', recall_score(y_train_resampled, y_hat_train
print('Testing Resampled Recall: ', recall_score(y_test, y_hat_test_resampled))
print('\n\n')

print('Training Resampled Accuracy: ', accuracy_score(y_train_resampled, y_hat_t
print('Testing Resampled Accuracy: ', accuracy_score(y_test, y_hat_test_resample
print('\n\n')

print('Training Resampled F1-Score: ', f1_score(y_train_resampled, y_hat_train_r
print('Testing Resampled F1-Score: ', f1_score(y_test, y_hat_test_resampled))

conf_matrix_train_resampled = confusion_matrix(y_train_resampled, y_hat_train_re
conf_matrix_resampled = confusion_matrix(y_test, y_hat_test_resampled)

# Print the confusion matrix
print("Train Confusion Matrix (after SMOTE):\n", conf_matrix_train_resampled)
print()
print("Test Confusion Matrix (after SMOTE):\n", conf_matrix_resampled)
```

```
Training Resampled Precision:  0.9145950280673617
Testing Resampled Precision:  0.4382716049382716



Training Resampled Recall:  0.9986865148861647
Testing Resampled Recall:  0.7029702970297029



Training Resampled Accuracy:  0.9527145359019265
Testing Resampled Accuracy:  0.8185907046476761



Training Resampled F1-Score:  0.9547928003348681
Testing Resampled F1-Score:  0.5399239543726236
Train Confusion Matrix (after SMOTE):
 [[2071  213]
 [   3 2281]]

Test Confusion Matrix (after SMOTE):
 [[475  91]
 [ 30  71]]
```

In [94]:

```
# Print confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_resampled, display_la
disp.plot(cmap=plt.cm.Blues);
```

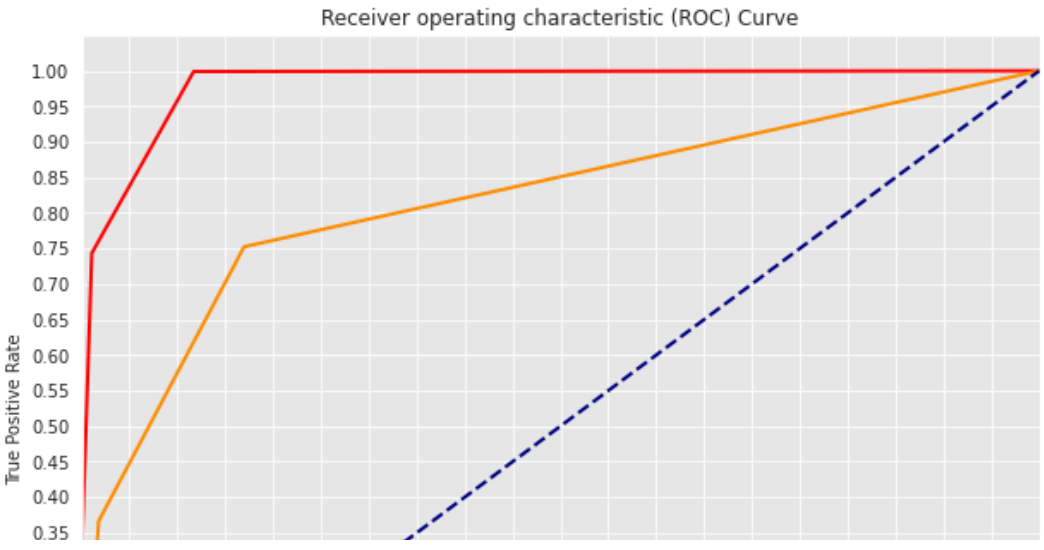




In [95]:

```
# Generate predictions on th train and test sets
y_score = knn.predict_proba(X_test_scaled)[:, 1]
y_train_score_resampled = knn.predict_proba(X_train_resampled)[:, 1]

# calculating the AUC value
fpr_train_resampled, tpr_train_resampled, thresholds_train_resampled = roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_score)

roc_auc_train_resampled = auc(fpr_train_resampled, tpr_train_resampled)
roc_auc = auc(fpr, tpr)
print('Train AUC: {}'.format(auc(fpr_train_resampled, tpr_train_resampled)))
print('Test AUC: {}'.format(auc(fpr, tpr)))

# plotting the ROC curve

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr_train_resampled, tpr_train_resampled, color='red',
         lw=lw, label=f'Train ROC_AUC  {roc_auc_train_resampled }')

plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label=f'Test ROC_AUC {roc_auc }')

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Train AUC: 0.9810002806395515
Test AUC: 0.816079487807438
```



In [96]:

```
# Fit a model
KNN = KNeighborsClassifier(n_neighbors=5)
KNN.fit(X_train_resampled, y_train_resampled)


# Predict
# Calculate the probability scores of each datapoint for the positive class (cla
y_score = KNN.predict_proba(X_test_scaled)[:, 1]
y_train_score = KNN.predict_proba(X_train_resampled)[:, 1]

# predict the model
y_hat_train_resampled = KNN.predict(X_train_resampled)
y_hat_test_resampled = KNN.predict(X_test_scaled)

# calculate the metrics


print('Training Resampled Precision: ', precision_score(y_train_resampled, y_hat
print('Testing Resampled Precision: ', precision_score(y_test, y_hat_test_resamp
print('\n\n')

print('Training Resampled Recall: ', recall_score(y_train_resampled, y_hat_train
print('Testing Resampled Recall: ', recall_score(y_test, y_hat_test_resampled))
print('\n\n')

print('Training Resampled Accuracy: ', accuracy_score(y_train_resampled, y_hat_t
print('Testing Resampled Accuracy: ', accuracy_score(y_test, y_hat_test_resample
print('\n\n')

print('Training Resampled F1-Score: ', f1_score(y_train_resampled, y_hat_train_r
print('Testing Resampled F1-Score: ', f1_score(y_test, y_hat_test_resampled))

conf_matrix_train_resampled = confusion_matrix(y_train_resampled, y_hat_train_re
conf_matrix_resampled = confusion_matrix(y_test, y_hat_test_resampled)

# Print the confusion matrix
print("Train Confusion Matrix (after SMOTE):\n", conf_matrix_train_resampled)
print()
print("Test Confusion Matrix (after SMOTE):\n", conf_matrix_resampled)
```

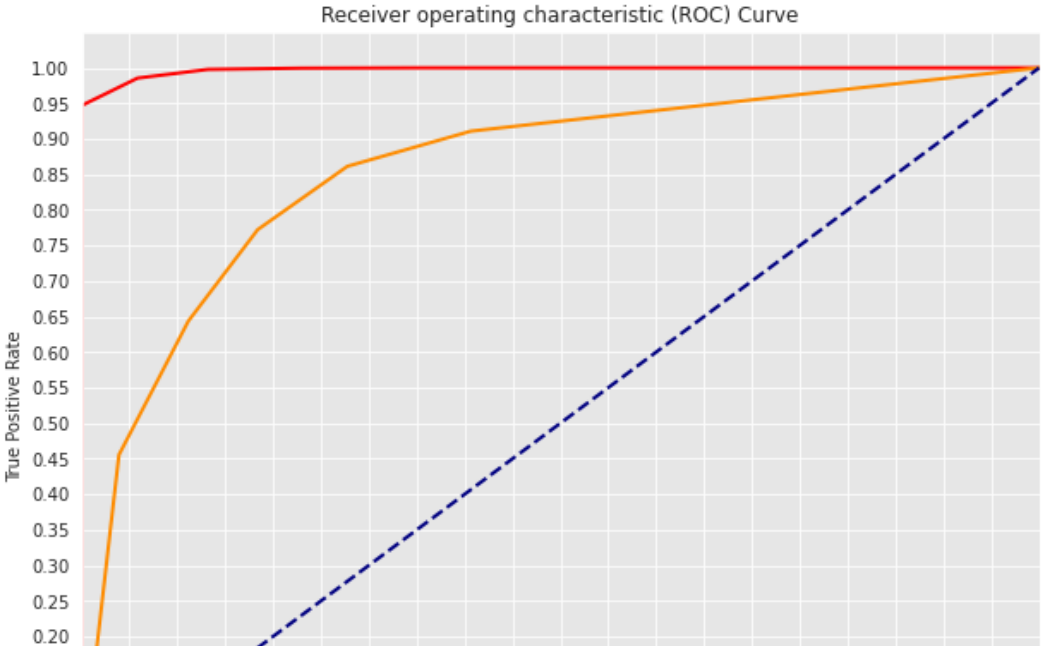
```
Training Resampled Precision:  0.8833333333333333
Testing Resampled Precision:  0.42857142857142855


Training Resampled Recall:  0.9978108581436077
Testing Resampled Recall:  0.7722772277227723


Training Resampled Accuracy:  0.9330122591943958
Testing Resampled Accuracy:  0.8095952023988006
```

```
Training Resampled F1-Score:  0.9370888157894737
Testing Resampled F1-Score:  0.5512367491166077
Train Confusion Matrix (after SMOTE):
 [[1983  301]
 [   5 2279]]

Test Confusion Matrix (after SMOTE):
 [[462 104]
 [ 23  78]]
```

In [97]:

```
# calculating the AUC value
fpr_train_resampled, tpr_train_resampled, thresholds_train_resampled = roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_score)

roc_auc_train_resampled = auc(fpr_train_resampled, tpr_train_resampled)
roc_auc = auc(fpr, tpr)
print('Train AUC: {}'.format(auc(fpr_train_resampled, tpr_train_resampled)))
print('Test AUC: {}'.format(auc(fpr, tpr)))

# plotting the ROC curve

sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr_train_resampled, tpr_train_resampled, color='red',
         lw=lw, label=f'Train ROC_AUC  {roc_auc_train_resampled }')

plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label=f'Test ROC_AUC {roc_auc }')

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Train AUC: 0.9972691317963078
Test AUC: 0.8579050484553756
```



The model has performed better with a K value of 5 as opposed to 3 from the previous model

We thus need to explore further and fine tune the model by using different K values to find the best one

In [103…

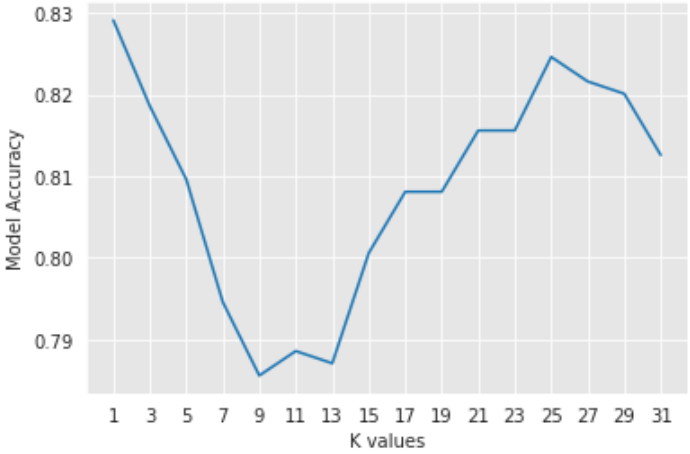
```
def find_best_k(X_train, y_train, X_test, y_test, min_k=1, max_k=31):
    """Finds and prints the best KNN model using the provided k range on a datase

        Args:
            X_train (Dataframe): Feature train set
            y_train (list): Target variable train set
            X_test (Dataframe): Feature test set
            y_test (list): Target variable test set

        Returns:
            None
    """
    best_k = 0
    best_score = 0.0
    precision_ = 0.0
    recall_ = 0.0
    accuracy_ = 0.0
    k_range = range(min_k, max_k+1, 2)
    k_scores_accuracy = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        preds = knn.predict(X_test)
        f1 = f1_score(y_test, preds)
        precision= precision_score(y_test, preds)
        recall = recall_score(y_test, preds)
        accuracy = accuracy_score(y_test, preds)
        k_scores_accuracy.append(accuracy)

        if f1 > best_score:
            best_k = k
            best_score = f1
            precision_ = precision
            recall_ = recall
            accuracy_ = accuracy
    plt.plot(k_range, k_scores_accuracy)
    plt.xticks(k_range)
    plt.xlabel('K values')
    plt.ylabel('Model Accuracy')

    print("Best Value for k: {}".format(best_k))
    print("F1-Score: {}".format(best_score))
    print("Precision: {}".format(precision_))
    print("Recall: {}".format(recall_))
    print("Accuracy: {}".format(accuracy_))
```

In [104…

```
# Find best K value
find_best_k(X_train_resampled, y_train_resampled, X_test_scaled, y_test)
```

Best Value for k: 25

```
Best value for k: 25
F1-Score: 0.5951557093425606
Precision: 0.4574468085106383
Recall: 0.8514851485148515
Accuracy: 0.8245877061469266
```



The f1 score is still very low despite some models having good accuracy and precision. We can explore other methods to improve the model. We can use cross validation to train the model.

In [105…

```
# Using cross validation. K = 3 ,cv = 5
knn_cv = KNeighborsClassifier(n_neighbors=3)
y_scores = cross_val_score(
    knn_cv,
    X_train_resampled,
    y_train_resampled,
    cv=5,
    scoring='accuracy'
)
print(f"Y scores for k = 3 and cv = 5\n{y_scores}")
print()
print(f"Mean accuracy score for k:{y_scores.mean()}")
```

```
Y scores for k = 3 and cv = 5
[0.89934354 0.89934354 0.90809628 0.9101862  0.91894852]

Mean accuracy score for k:0.9071836181008098
```

In [107…

```
# Using best K value during cross validation. K = 25 ,cv = 5
knn_best = KNeighborsClassifier(n_neighbors=25)
y_scores = cross_val_score(
    knn_best,
    X_train_resampled,
    y_train_resampled,
    cv=5,
    scoring='accuracy'
)
print(f"Y scores for k = 25 and cv = 5\n{y_scores}")
print()
print(f"Mean accuracy score for k:{y_scores.mean()}")
```

```
Y scores for k = 25 and cv = 5
[0.84682713 0.86652079 0.87417943 0.86308872 0.8871851 ]

Mean accuracy score for k:0.8675602349721144
```

In [110…

```
# Using best K value during cross validation. K = 25 ,cv = 10
knn_best = KNeighborsClassifier(n_neighbors=25)
y_scores = cross_val_score(
```

```
y_scores = cross_val_score(
    knn_best,
    X_train_resampled,
    y_train_resampled,
    cv=10,
    scoring='f1'
)
print(f"Y scores for k = 25 and cv = 10\n{y_scores}")
print()
print(f"Mean F1 score for k:{y_scores.mean()}")
```

```
Y scores for k = 25 and cv = 10
[0.84583333 0.86792453 0.86597938 0.88270378 0.90612245 0.876
 0.87025948 0.86227545 0.89205703 0.90456432]

Mean F1 score for k:0.8773719741363092
```

The model's generalization capabilities have greatly increased when cross validation is used. The accuracy of the model is 86.82% and the mean f1 score has greatly improved to 87.73%

iii) Decision Trees Model

We train the dataset using a decison tree model in the quest to find a good one that generalizes well on unseen data

In [113…

```
# Define the predictors and target
X= df.drop(columns=['state','churn'], axis =1 )
y = df['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

#Standerdize the features
scaler = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)

# Train a DT classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train_scaled, y_train)

# Make predictions on the test set
predictions= clf.predict(X_test_scaled)

# Make predictions for test data
y_pred = clf.predict(X_test_scaled)
```

In [114…

```
y_hat_train = clf.predict(X_train_scaled)
y_hat_test = clf.predict(X_test_scaled)

# calculating the metrics
print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
```

```
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))
```

```
Training Precision:  1.0
Testing Precision:  0.7403846153846154



Training Recall:  1.0
Testing Recall:  0.7623762376237624



Training Accuracy:  1.0
Testing Accuracy:  0.9235382308845578



Training F1-Score:  1.0
Testing F1-Score:  0.751219512195122
```

In [115…

```
# Calculate accuracy
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy is :{0}'.format(acc))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
```

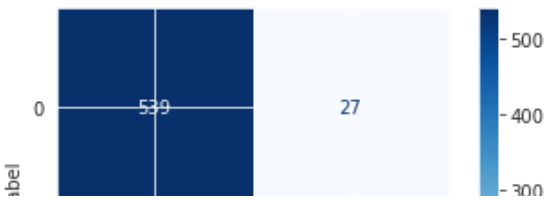
```
Accuracy is :92.35382308845578

AUC is :0.86

Confusion Matrix
----------------
```

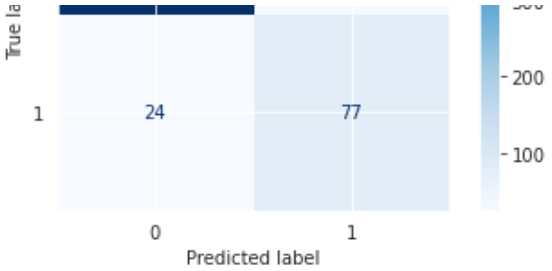
Out[115…

Predicted	0	1	All
True			
0	539	27	566
1	24	77	101
All	563	104	667

In [116…

```
# Plot confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=clf.cl
disp.plot(cmap=plt.cm.Blues);
```





Fine tuning

Finding the best parameter using 'Entropy' criterion

In [121…

```
# Instantiate and fit a DecisionTreeClassifier
classifier_2 = DecisionTreeClassifier(random_state=42, criterion='entropy')
classifier_2.fit(X_train, y_train)
```

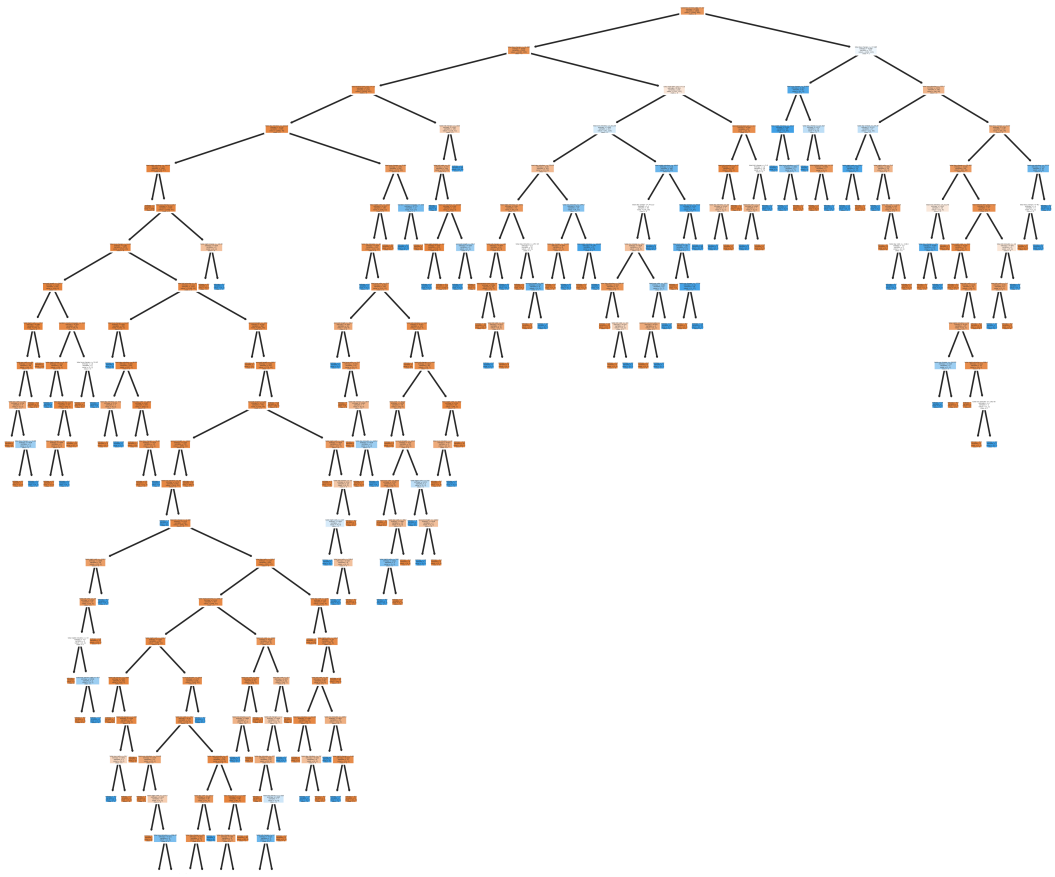
Out[121…

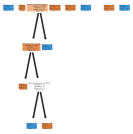
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

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

In [122…

```
# Plot and show decision tree
plt.figure(figsize=(12,12), dpi=500, edgecolor='blue')
tree.plot_tree(classifier_2,
               feature_names=X.columns,
               class_names=np.unique(y).astype('str'),
               filled=True, rounded=True)
plt.show()
```





In [123…

```
# Generate predictoins
y_score = classifier_2.predict_proba(X_test_scaled)[:, 1]
y_train_score = classifier_2.predict_proba(X_train_scaled)[:, 1]
```

```
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py:
465: UserWarning: X does not have valid feature names, but DecisionTreeClassifier
was fitted with feature names
  warnings.warn(
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py:
465: UserWarning: X does not have valid feature names, but DecisionTreeClassifier
was fitted with feature names
  warnings.warn(
```

In [124…

```
y_hat_train = classifier_2.predict(X_train_scaled)
y_hat_test = classifier_2.predict(X_test_scaled)

# calculating the metrics
print('Training Precision: ', precision_score(y_train, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test))
```

```
Training Precision:  0.43656716417910446
Testing Precision:  0.40298507462686567



Training Recall:  0.306282722513089
Testing Recall:  0.26732673267326734



Training Accuracy:  0.8439609902475619
Testing Accuracy:  0.8290854572713643



Training F1-Score:  0.36
Testing F1-Score:  0.3214285714285714
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py:
465: UserWarning: X does not have valid feature names, but DecisionTreeClassifier
was fitted with feature names
  warnings.warn(
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py:
465: UserWarning: X does not have valid feature names, but DecisionTreeClassifier
was fitted with feature names
  warnings.warn(
```

Balancing

Class imbalance using SMOTE

In [125…

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

#Standerdize the features
scaler = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)

# Train a DT classifier
smote1 = SMOTE(sampling_strategy='minority', random_state=420)
X_train_over, y_train_over = smote1.fit_resample(X_train_scaled, y_train)

# Summarize class distribution
#print('Counter'(y_train_over))

clf3 = DecisionTreeClassifier(random_state=420)

clf3.fit(X_train_over, y_train_over)
```

Out[125…

```
DecisionTreeClassifier(random_state=420)
```

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

In [126…

```
# Make predictions on the test set
y_pred_test= clf3.predict(X_test_scaled)

# # Make predictions for test data
y_pred_train = clf3.predict(X_train_over)
```

In [127…

```
# calculating the metrics
print('Training Precision: ', precision_score(y_train_over, y_pred_train))
print('Testing Precision: ', precision_score(y_test, y_pred_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train_over, y_pred_train))
print('Testing Recall: ', recall_score(y_test, y_pred_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train_over, y_pred_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_pred_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train_over, y_pred_train))
print('Testing F1-Score: ', f1_score(y_test, y_pred_test))
```

```
Training Precision:  1.0
Testing Precision:  0.6090225563909775



Training Recall:  1.0
Testing Recall:  0.801980198019802



Training Accuracy:  1.0
Testing Accuracy:  0.8920539730134932
```

```
Training F1-Score:  1.0
Testing F1-Score:  0.6923076923076924
```

In [119…

```python
# Assuming you have loaded and split your data
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Define the decision tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters and values to search
param_grid = {
    'max_depth': np.arange(1, 10),
    'min_samples_split': np.arange(2, 10),
    'min_samples_leaf': np.arange(1, 5),
    'max_features': ['auto', 'sqrt', 'log2', None]
}

# Create the GridSearchCV object
grid_search = GridSearchCV(dt_model, param_grid, cv=5, scoring='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

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

# Get the best model
best_dt_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_dt_model.predict(X_test)

# Evaluate the performance of the best model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the Best Model:", accuracy)
```

```
Best Hyperparameters: {'max_depth': 6, 'max_features': None, 'min_samples_leaf':
1, 'min_samples_split': 5}
Accuracy of the Best Model: 0.9490254872563718
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/model_se
lection/_validation.py:425: FitFailedWarning:
1440 fits failed out of a total of 5760.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sco
re='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
1440 fits failed with the following error:
Traceback (most recent call last):
  File "/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/
model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/
base.py", line 1145, in wrapper
    estimator._validate_params()
  File "/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/
base.py", line 638, in _validate_params
    validate_parameter_constraints(
  File "/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/
```

```
utils/_param_validation.py", line 96, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramete
r of DecisionTreeClassifier must be an int in the range [1, inf), a float in the r
ange (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/andrew/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/model_se
lection/_search.py:979: UserWarning: One or more of the test scores are non-finit
e: [       nan        nan        nan ... 0.93285551 0.93285551 0.93585738]
  warnings.warn(
```

In [121…

```
# calculating the metrics
#print('Training Precision: ', precision_score(y_train_over, y_pred_train))
print('Testing Precision: ', precision_score(y_test, y_pred))
print('\n\n')

#print('Training Recall: ', recall_score(y_train_over, y_pred_train))
print('Testing Recall: ', recall_score(y_test, y_pred))
print('\n\n')

#print('Training Accuracy: ', accuracy_score(y_train_over, y_pred_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_pred))
print('\n\n')

#print('Training F1-Score: ', f1_score(y_train_over, y_pred_train))
print('Testing F1-Score: ', f1_score(y_test, y_pred))
```

```
Testing Precision:  0.7403846153846154


Testing Recall:  0.7623762376237624


Testing Accuracy:  0.9235382308845578


Testing F1-Score:  0.751219512195122
```

iv) Random Forest Model

We employed a Random Forest model to capture and handle the complex and non-linear patterns within the dataset. The problem required that we build a robust model that could factor intricate relationships that the other models could have overlooked. Iteratively, adjustments were made to enhance the model's performance, balancing the pursuit of accuracy with the need for interpretability.

With Imbalanced data

In [122…

```
# separating features from the target variable
X = df.drop(['churn','state'],axis = 1)
y = df['churn']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_
```

In [129…

```
#importing library and initializing the classifier
r_forest = RandomForestClassifier(n_estimators=100, max_depth =5, max_features=1
r_forest.fit(X_train, y_train)
```

Out[129…

```
RandomForestClassifier(max_depth=5, max_features=15)
```

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

In [130…

```
#generating predictions based on the X_test
y_ran_pred = r_forest.predict(X_test)
print(pd.Series(y_ran_pred).value_counts())
```

```
0    587
1     80
dtype: int64
```

In [131…

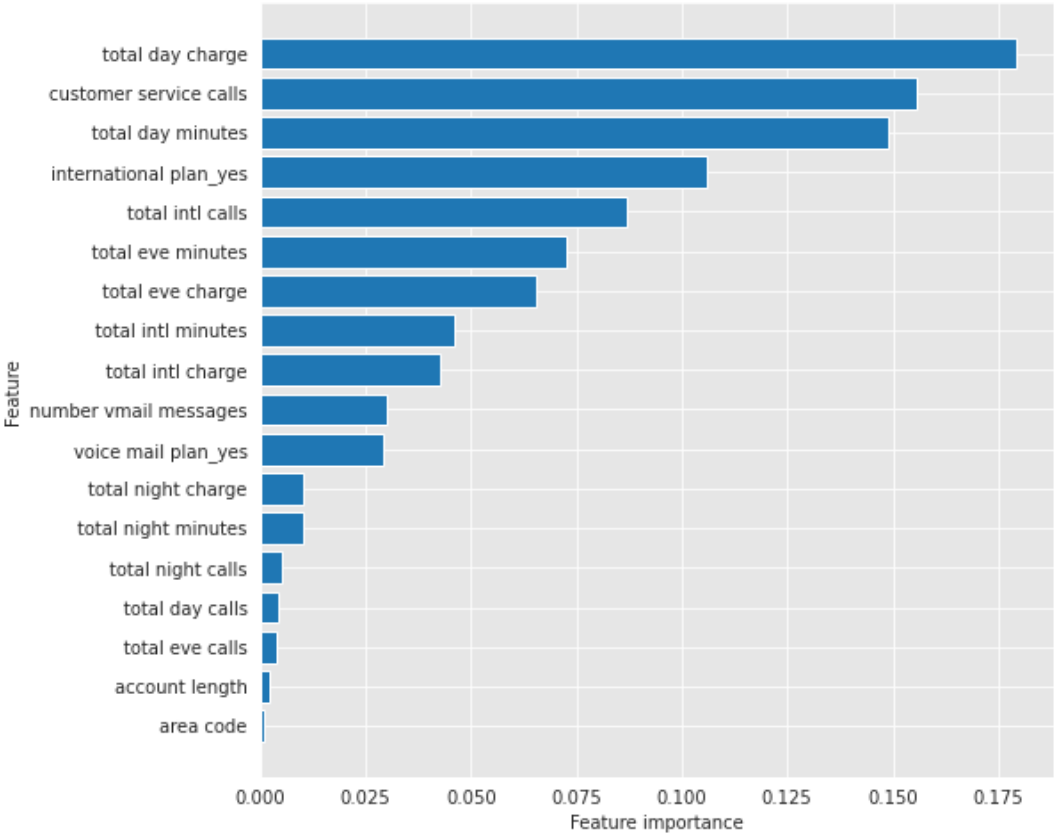
```
#define the scores
recall_ran =recall_score(y_test, y_ran_pred)
precision_ran =precision_score(y_test, y_ran_pred)
f1score_ran =f1_score(y_test, y_ran_pred)
accuracy_ran= accuracy_score(y_test, y_ran_pred)

#print the scores
print(f'Accuracy score: {accuracy_ran}\n')
print(f'Precision score: {precision_ran}\n')
print(f'Recall score: {recall_ran}\n')
print(f'f1 score: {f1score_ran}\n')

#getting the AUC score
from sklearn.metrics import roc_auc_score
y_ran_prob = r_forest.predict_proba(X_test)[:, 1]

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_ran_prob)
print(f'AUC score: {auc_score}')
```

```
Accuracy score: 0.9445277361319341

Precision score: 0.9

Recall score: 0.7128712871287128

f1 score: 0.7955801104972375

AUC score: 0.9302994787111221
```

In [132…

```
#examining how each feature ended up being important in our decison tree.
def plot_feature_importances(model):
    """Plot the feaures and their contribution to the model's performance

        Args:
            model (object): model trained

        Returns:
            None
    """
    n_features = X_train.shape[1]
    feature_names = X_train.columns.values
    feature_importances = model.feature_importances_
    indices = np.argsort(feature_importances)[::+1]
    plt.figure(figsize=(8,8))
    plt.barh(range(n_features),feature_importances[indices], align='center')
    plt.yticks(np.arange(n_features), feature_names[indices])
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

```
#Looking at the feature importance
plot_feature_importances(r_forest)
```



With Balanced Data

In [133…
```
y_train.value_counts()
```

Out[133…
```
0    2284
1     382
Name: churn, dtype: int64
```

In [134…
```
smote = SMOTE(random_state=42)
#resample the train dataset
X_train_over, y_train_over = smote.fit_resample(X_train, y_train)
print(pd.Series(y_train_over).value_counts())
```

```
1    2284
0    2284
Name: churn, dtype: int64
```

In [136…
```
#Initialize another random forest model
r_forest_smote = RandomForestClassifier(n_estimators=100, max_depth=5, max_featu
#fit the model on the oversampled data
r_forest_smote.fit(X_train_over, y_train_over)
#predict the model
y_ran_pred1 = r_forest_smote.predict(X_test)

#evaluate the model using recall, precison, accuracy and f1score
recall_ran1 = recall_score(y_test, y_ran_pred1)
precision_ran1 = precision_score(y_test, y_ran_pred1)
f1score_ran1 = f1_score(y_test, y_ran_pred1)
accuracy_ran1= accuracy_score(y_test, y_ran_pred1)

#print the results
```

```
print(f'Accuracy score: {accuracy_ran1}\n')
print(f'Precision score: {precision_ran1}\n')
print(f'Recall score: {recall_ran1}\n')
print(f'f1 score: {f1score_ran1}\n')

#getting the AUC score
y_ran_prob1 = r_forest_smote.predict_proba(X_test)[:, 1]
# Calculate the AUC score
auc_score1 = roc_auc_score(y_test, y_ran_prob1)
print(f'AUC score: {auc_score1}')
```

```
Accuracy score: 0.9325337331334332

Precision score: 0.7692307692307693

Recall score: 0.7920792079207921

f1 score: 0.7804878048780488

AUC score: 0.8975265017667843
```

Balancing the data led to a trade-off, with improved recall but a decrease in precision.

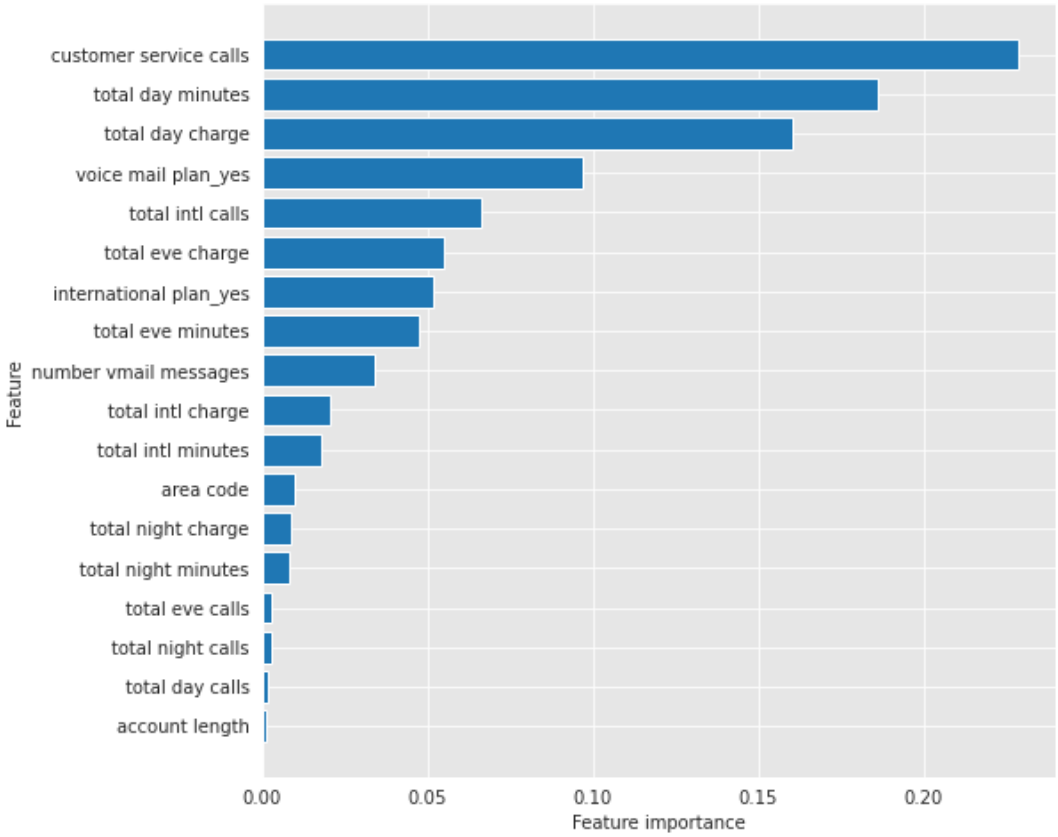
In [137…

```
#checking the distribution for the predicted values
print(pd.Series(y_ran_pred1).value_counts())
```

```
0    563
1    104
dtype: int64
```

In [138…

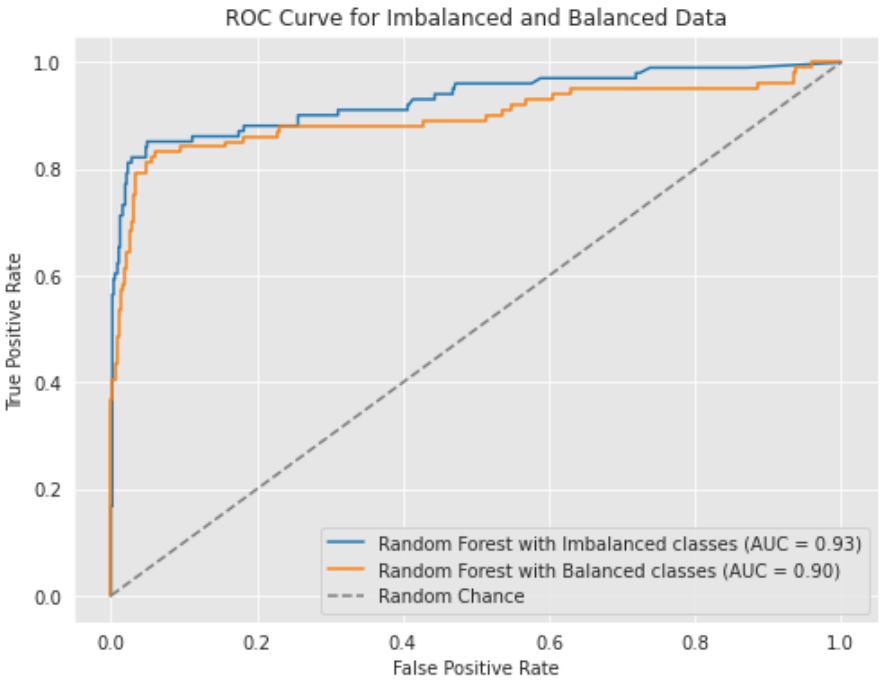
```
#looking at the feature importance
plot_feature_importances(r_forest_smote)
```



In [139…

```
# Plotting an ROC curve to check model performanced for imbalanced and balanced
```

```
# Predict probabilities and calculate the ROC curve for imbalanced data model
fpr, tpr, thresholds = roc_curve(y_test, r_forest.predict_proba(X_test)[:, 1])
# Calculate AUC for the first model
auc_score = roc_auc_score(y_test, y_ran_prob)
# Predict probabilities and calculate the ROC curve for balanced data model
fpr1, tpr1, thresholds1 = roc_curve(y_test, r_forest_smote.predict_proba(X_test)
# Calculate AUC for the second model
auc_score1 = roc_auc_score(y_test, y_ran_prob1)
# Predict probabilities and calculate the ROC curve for tuned model
fpr1, tpr1, thresholds1 = roc_curve(y_test, r_forest_smote.predict_proba(X_test)
# Calculate AUC for the second model
auc_score1 = roc_auc_score(y_test, y_ran_prob1)

# Plot the ROC curves for both models on the same axis
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Random Forest with Imbalanced classes (AUC = {auc_sco
plt.plot(fpr1, tpr1, label=f'Random Forest with Balanced classes (AUC = {auc_sco
# Plot ROC curve for random chance
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Chance')
# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Imbalanced and Balanced Data')
plt.legend()

# Display the plot
plt.show();
```



Hyperparameter Tuning Using Gridsearch

In [140…

```
# Define the parameter grid for tuning
param_grid = {
    'max_depth': range(1, 15),  # Adjust the range based on your preference
}

# # Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=7,  # Using 5-fold cross-validation
                           scoring='accuracy',  # You can use other metrics like
```

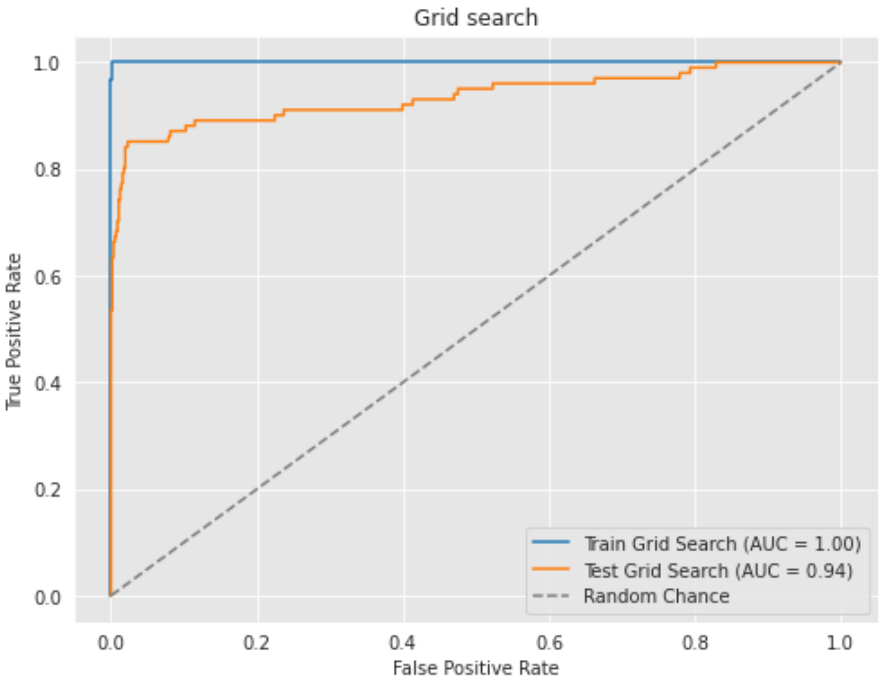
```
                                n_jobs=-1)  # Use all available CPU cores for faster p

# # Perform the grid search on scaled data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# # Get the best model
best_model = grid_search.best_estimator_
#make predictions
y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1_score = f1_score(y_train, y_train_pred)
test_f1_score = f1_score(y_test, y_test_pred)

y_prob = best_model.predict_proba(X_test)[:, 1]
auc_score_tuned = roc_auc_score(y_test, y_prob)
print(f'AUC score: {auc_score_tuned}\n')

print("Train Accuracy:", train_accuracy, '\n')
print("Test Accuracy:", test_accuracy,'\n')
print("Train F1 Score:", train_f1_score,'\n')
print("Test F1 Score:", test_f1_score,'\n')
```

```
Best Hyperparameters: {'max_depth': 14}
AUC score: 0.9359147045446594

Train Accuracy: 0.9879969992498124

Test Accuracy: 0.9490254872563718

Train F1 Score: 0.9562841530054645

Test F1 Score: 0.8111111111111111
```

In [144…

```
# Score the model on the train set
y_prob_train = best_model.predict_proba(X_train)[:, 1]

# Predict probabilities and calculate the ROC curve for imbalanced data model
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_prob_train)
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate AUC for the first model
auc_score = roc_auc_score(y_test, y_prob)
auc_score_train = roc_auc_score(y_train, y_prob_train)

# Plot the ROC curves for both models on the same axis
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, label=f'Train Grid Search (AUC = {auc_score_train
plt.plot(fpr, tpr, label=f'Test Grid Search (AUC = {auc_score:.2f})')

# Plot ROC curve for random chance
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Chance')

# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Grid search')
plt.legend()
```

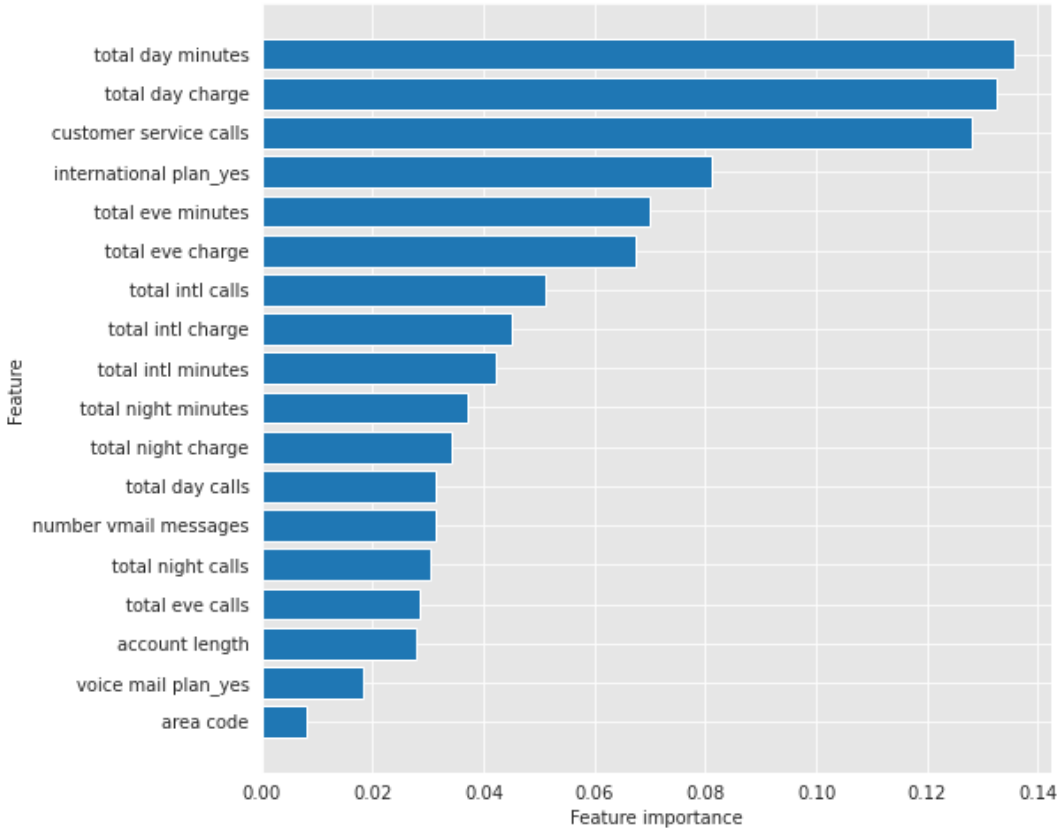
```
# Display the plot
plt.show();
```



The hyperparameter tuned random forest model performs the best with an AUC score of **94%**. Now we can check the importance of the features used in the model to measure how much each contributed to the overall predictive perfomance.

In [142…

```
#Checking the feature importance
plot_feature_importances(best_model)
```



The most significant drivers of customer churn according to this model are:

- The total number of day minutes a customer spend on call.
- The total amount a customer is charged on daytime calls
- The number of customer service calls a customer make
- The presence or absence on an international plan

5. Evaluation

The metrics we used to evaluate our models are precision, recall, accuracy and f1 score. They help us identify which model has the best performance in predicting customer churn.

i) Logistic Regression

Based on these metrics, it seems like the resampled data with SMOTE has slightly better performance compared to the other datasets, especially in terms of recall and F1-score.

- Here are the figures in percentage:Resampled Data with SMOTE balancing :Testing Accuracy: 75.0%,Testing F1-Score: 45.5%.
- Original Data:Testing Accuracy: 68.2%,Testing F1-Score: 45.6%.Resampled Data: Testing Accuracy: 75.4%Testing F1-Score: 45.2%

These results suggest that the resampled data with SMOTE technique slightly outperforms the other datasets in terms of model evaluation metrics.

For logistic we have to consider the limitations of each approach when selecting the best model for your specific hence the need to explore other techniques or algorithms may be necessary to further improve model performance.

ii) KNN

Imbalanced Data: The model performed poorly on the imbalaced data, with an accuracy score of 22%. The precision of 78% shows that the able is able to identify true positives in the prediction. However, the recall score of 36% suggests a poor ability to capture all actual positive cases. The F1 score of 50% balances precision and recall.

Fine Tuning: This was done on a balanced dataset Hyperparameter tuning using cross validation led to a refined model with an AUC score 98%. The accuracy of 90.72% shows a successful generalization to unseen data, decreasing the risk of overfitting

iii) Decision Trees

The decision tree model trained on scaled data exhibited the highest testing accuracy at 92.4%, albeit with potential overfitting, while the models using entropy criterion and SMOTE balancing achieved lower accuracies of 82.9% and 89.2% respectively, indicating the need for further evaluation to balance accuracy and generalization.

iv) Random Forest

Imbalanced Data: The model performed poorly on the imbalaced data, with an accuracy

Imbalanced Data: The model performed poorly on the imbalaced data, with an accuracy score of 39%. The precision of 17% shows that the able is able to identify true positives in the prediction. However, the recall score of 82% suggests a moderate ability to capture all actual positive cases. The F1 score of 77% balances precision and recall.

Hyperparameter Tuning: Hyperparameter tuning using Gridsearch led to a refined model with an AUC score 98%. The difference between train and test scores 98% vs. 94% accuracy shows the successful generalization to unseen data, decreasing the risk of overfitting.

6. Conclusion and Recommendations

Conclusion

In conclusion, this project aimed to predict customer churn in the telecom industry using a machine learning model. The dataset was cleaned. EDA was performed to analyze the distribution of the independent variables and the target variable. 4 models were used to predict customer churn, and the Random Forest model had the highest accuracy.

Recommendations

- Implement targeted promotional offers or discounts for customers in area codes 415 and 510, where the churn rate is notably higher. This strategy aims to incentivize customer loyalty in these specific regions.
- Enhance the overall quality of customer service and actively work towards reducing the number of customer service calls. A seamless and efficient customer service experience can positively impact customer satisfaction and retention.
- Conduct a thorough evaluation of the pricing structure for day, evening, night, and international charges. Ensure that the pricing aligns with customer expectations and market competitiveness.
- Develop and implement robust customer retention strategies, with a special focus on states exhibiting higher churn rates. States like Minnesota, Conneticut, Oregon, New york should be targeted for personalized retention efforts.
- Review and improve the value proposition of the voicemail plan to make it more appealing to customers. This may involve introducing additional features, enhancing user experience, or adjusting pricing to increase the perceived value of the voicemail service.
- By addressing these key areas, Syria Tel can create a comprehensive strategy to reduce churn, enhance customer satisfaction, and foster long-term customer relationships.

Future work

Look for more customer data like feedback, reasons for calling customer service numbers. Employ other robust models like Xboost etc,