DSF PT6-Phase 3 Project

- *STEVE ABONYO*

Business Understanding

__Overview__
We are looking into the Syria Telecommunications company dataset with a focus n the customer churn

__Problem Statement__
We want to know if a customer will soon stop doing business with Syria Telecommunications company or not and establish of there is a pattern. The goal here is make sure that our telcom business staff know how much money is lost through high churn rate and actually how to reduce this loss

__Stakeholder__
The telecommunication business

__Understanding__
Churn is measure of the rate at which customers stop busines, in our context, the rate at which customers stop doing business with Syria Telcom company

Data Understanding

Below is an explanation of the columns on our dataset

- `state` : The US state in which the customer resides.
- `account length` : The duration (in days) of the customer's account with the company
- `area code` : The area code of the customer's phone number.
- `phone number` : The customer's phone number.
- `international plan` : Whether the customer has an international calling plan (yes or no).
- `voice mail plan` : Whether the customer has a voice mail plan (yes or no).
- `number vmail messages` : The number of voice mail messages the customer has.
- `total day minutes` : The total number of minutes the customer has used during the day.
- `total day calls` : The total number of calls the customer has made during the day.
- `total day charge` : The total charges incurred by the customer for day usage.
- `total eve minutes` : The total number of minutes the customer has used during the evening.

- `total eve calls` : The total number of calls the customer has made during the evening.
- `total eve charge` : The total charges incurred by the customer for evening usage.
- `total night minutes` : The total number of minutes the customer has used during the night.
- `total night calls` : The total number of calls the customer has made during the night.
- `total night charge` : The total charges incurred by the customer for night usage.
- `total intl minutes` : The total number of minutes the customer has used for international calls.
- `total intl calls` : The total number of international calls the customer has made.
- `total intl charge` : The total charges incurred by the customer for international usage.
- `customer service calls` : The number of calls the customer has made to customer service.
- `churn` : Whether the customer has churned (True or False).

Data Preparation

In [1]:
```
###Importing the libraries to use for the project
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from sklearn import datasets
import statsmodels
from statsmodels.formula.api import ols
```

In [2]:
```
##Importing data
data = pd.read_csv('SyriaTel_Customer_Churn.csv')
data.head(3)
```

Out[2]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tot da charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.(
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.3

3 rows × 21 columns

In [3]:
```
# Getting basic information about the datatype
print(data.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
None
```

In [4]:
```
# Descriptive statistics of our dataset
data.describe()
```

Out[4]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	3
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	
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	
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	

In [5]:
```
# Check for missing values
print(data.isnull().sum())
```

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

In [6]:
```
# Sampling 5 random rows of our dataset
data.sample(5)
```

Out[6]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	cl
2470	MS	189	415	411-6501	no	no	0	227.8	124	
1985	MT	101	408	362-2787	no	yes	16	118.9	112	
427	NH	67	415	355-1113	no	yes	40	104.9	65	
2643	MI	74	415	386-4215	no	no	0	124.8	114	
1737	RI	134	415	413-1789	no	no	0	141.7	95	

5 rows × 21 columns

In [7]:
```
## Lets check the shape of our data
data.shape
```

Out[7]:
```
(3333, 21)
```

NB:

- `We have generally previewed our data using to to see the type of information i.e., the column types, the size of the data, and general view of how the data is arranged`

- `We have checked for missing values and luckily enough we do not have missing values in our dataset`

In [8]:
```
## Let me print out the column names in my dataset
print(data.columns)
```

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

In [9]:
```
## Lets replace the spaces in the column names with an underscore
data.columns = data.columns.str.replace(' ', '_')
print(data.columns)
```

```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

In [10]:
```
data.head(2)
```

Out[10]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan
0	KS	128	415	382-4657	no	yes
1	OH	107	415	371-7191	no	yes

2 rows × 21 columns

In [11]:
```
##Lets drop some columns that we may not need going forward
data.drop('phone_number', axis = 1, inplace = True)
```

In [12]:
```
##Now we are good to go
```

EDA

UNIVARIATE ANALYSIS

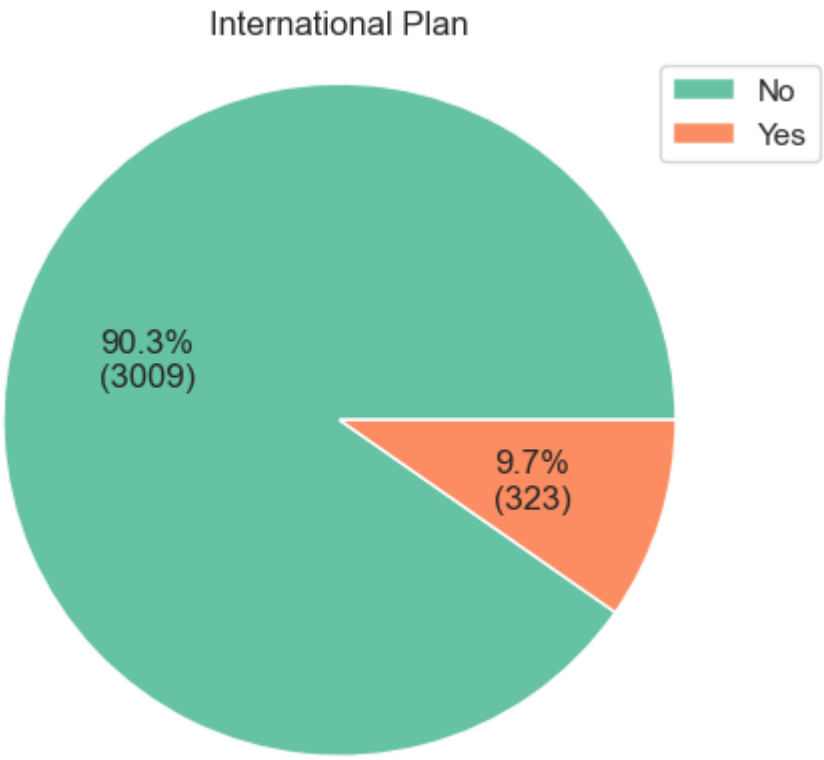
We are looking at the analysis of each relevany single column in our dataset to see its distribution

From observations in the data, we have to columns i.e., Internation plan and Voice mail plan that have the Labels of Yes and No, that can be represented by simple pie charts because the categories are few. We will create a simple function to generate a pie chart for the 2 columns

In [13]:
```
def plot_pie_chart(data, column, title):
    label_map = {'yes' : 'Yes',
         'no' : 'No'}
    data[column] = data[column].replace(label_map)

    # Set style
    sns.set(style="whitegrid")

    # Aggregate the data
    count_response = data[column].value_counts().reset_index()
    count_response.columns = ['Response', 'Counts']

    total_counts = count_response['Counts'].sum()

    # Create pie chart
    fig, ax = plt.subplots()
    ax.pie(
        count_response['Counts'],
        labels=None,
        autopct=lambda pct: f'{pct:.1f}%\n({int(pct*total_counts/100):d})',
        colors=sns.color_palette("Set2")
    )

    ax.axis('equal')  # Ensure that the pie is drawn as a circle

    ax.set_title(title)  # Set graph title

    # Add legend
    ax.legend(title='', loc='upper right', labels=count_response['Response'])

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

International plan

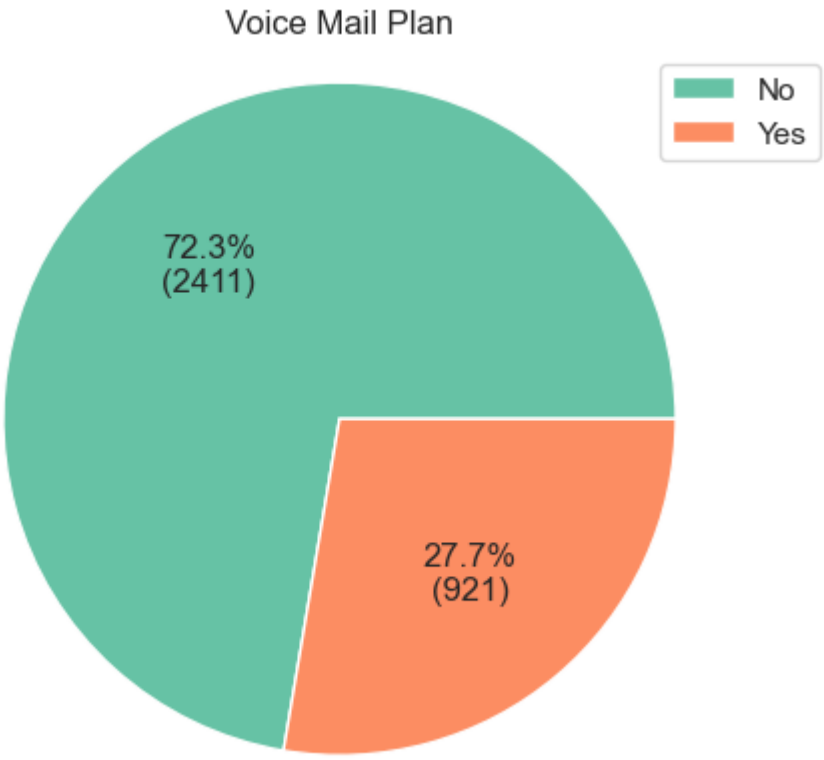
In [14]:
```
plot_pie_chart(data, 'international_plan', "International Plan")
```



We can see that 90.3% of the calls in our dataset are not in the internation plan, where as only 9.7% are in the international plan

Voice mail plan

In [15]:
```
plot_pie_chart(data, 'voice_mail_plan', "Voice Mail Plan")
```



We can see that 72.3% of the calls in our dataset are not in the voice mail, where as only 27.7% are in the international plan

In [16]:
```
## Now let me select all the numeric columns of my dataset first
columns_numeric = data.select_dtypes(include='number').columns.tolist()
columns_numeric
```

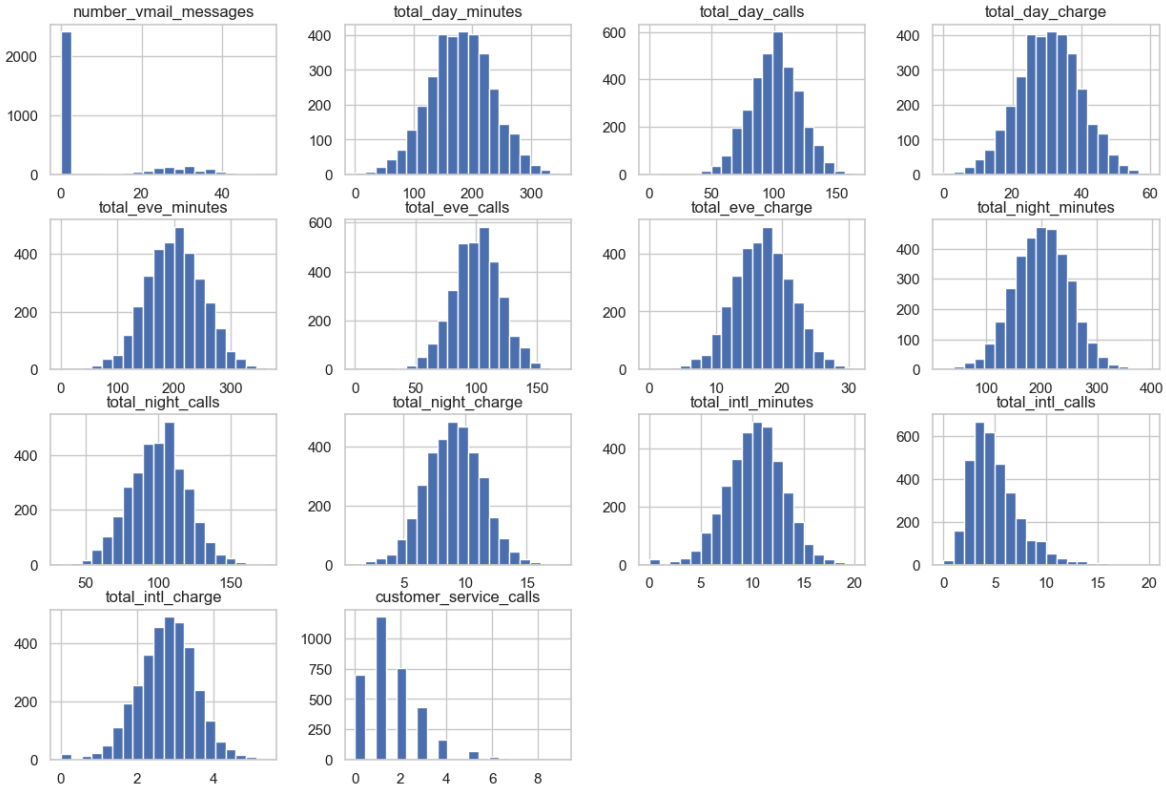
Out[16]:
```
['account_length',
 'area_code',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'customer_service_calls']
```

In [17]:
```
#There are some particular columns that i dont want to be plotted, let me remove
cols_to_drop = ['account_length', 'area_code']

## now dropping the columns
for col in cols_to_drop:
    if col in columns_numeric:
        columns_numeric.remove(col)

columns_numeric ## this are now the columns i need
```

Out[17]:
```
['number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'customer_service_calls']
```

In [18]:
```
##Lets iterate over each column
# for col in columns_numeric:
#     plt.figure()
#     data[col].hist()
#     plt.title(f'Hisgram of {col}')
#     plt.xlabel('Value')
#     plt.ylabel('Frequency')
#     plt.grid(False)
#     plt.show()
```

```
## I want to view all the histograms in one window
data[columns_numeric].hist(bins=20, figsize=(15,10))
plt.show()
```



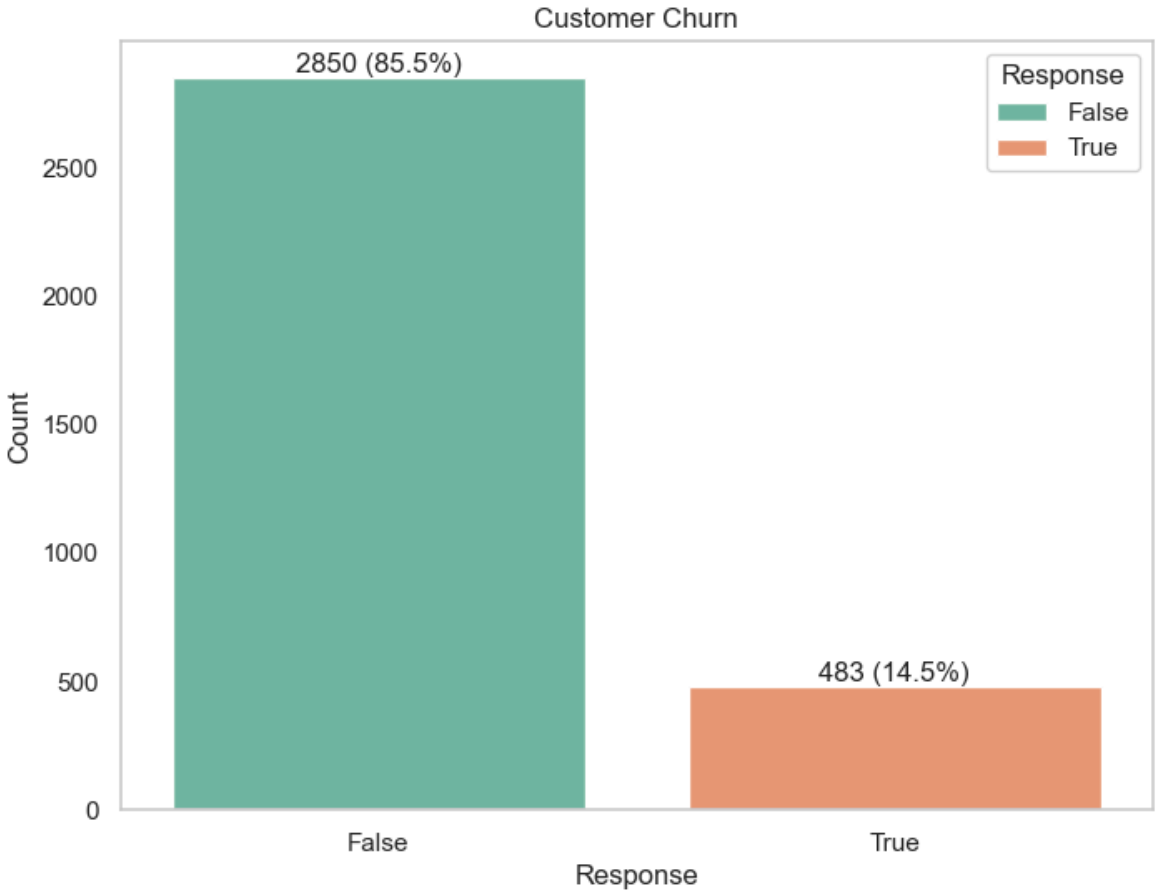
From the plot of histograms for our numeric data, we have seen most of our columns are symmetrical apart from a few, namely:-

- Customer service calls
- Number of voice mail messages
- Total international calls

Churn

In [19]:
```
sns.set(style="whitegrid")

# Aggregate the data
count_response = data['churn'].value_counts().reset_index()
count_response.columns = ['Response', 'Counts']

##Lets calculate the %
total_count = count_response["Counts"].sum()
count_response['Percentage'] = round((count_response['Counts'] / total_count) *

## bar plot
plt.figure(figsize=(8,6)) ##setting pur figure size
sns.barplot(x='Response', y='Counts', hue='Response', data = count_response, pal

# Add labels to the bars
for i in range(len(count_response)):
    plt.text(i, count_response['Counts'][i], f"{count_response['Counts'][i]} ({c
             ha='center', va='bottom')

plt.title("Customer Churn")
plt.xlabel("Response")
```

```
plt.ylabel("Count")
plt.grid(False)
plt.show()
```



According to this dataset 85.5% has their churn to be False (meaning they did not stop their service) and 14.5% is equal to True(meaning they stopped their services)

BIVARIATE ANALYSIS

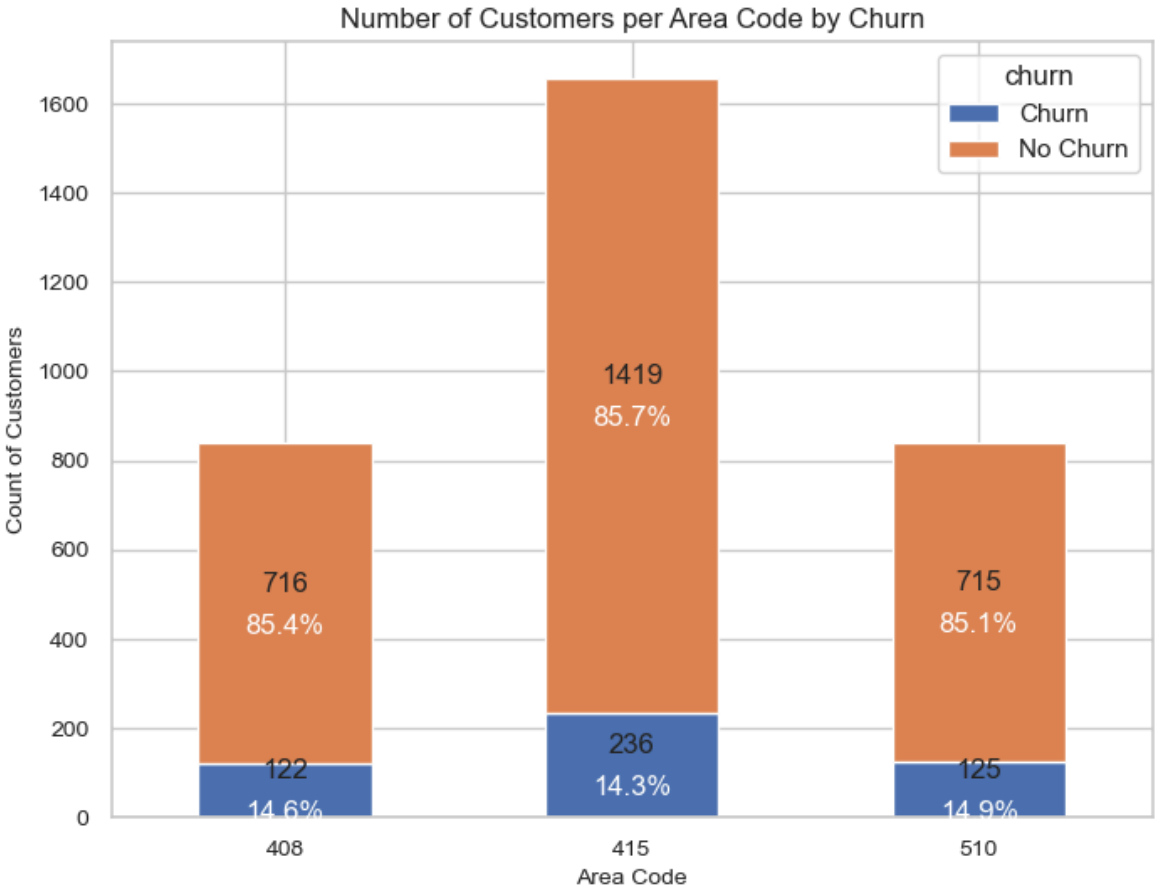
Using our column of interest which is churn, we want compare it against a few other columns which we think may be key to see their relationship

In [20]:
```
##First i want to rename the labels in my churn column
data['churn'] = data['churn'].replace({False: 'No Churn', True: 'Churn'})
```

Churn and Area Code

In [21]:
```
# Plotting bar chart
fig, ax = plt.subplots(figsize=(8, 6))
counts = data.groupby(['area_code', 'churn']).size().unstack()
counts.plot(kind='bar', stacked=True, ax=ax)

# Adding count and percentage labels on top of each bar
for patch in ax.patches:
    # Calculate position for annotation
    height = patch.get_height()
    width = patch.get_width()
    x = patch.get_x()
    y = patch.get_y() + height / 2
```

```
    # Add count label
    ax.annotate(f'{int(height)}', xy=(x + width / 2, y), xytext=(0, 3),
                textcoords='offset points', ha='center', va='bottom')
    # Add percentage label
    total_height = sum([p.get_height() for p in ax.patches if p.get_x() == x])
    percentage = (height / total_height) * 100
    ax.annotate(f'{percentage:.1f}%', xy=(x + width / 2, y), xytext=(0, -15),
                textcoords='offset points', ha='center', va='bottom', color='whi

# Set plot title and labels
plt.title('Number of Customers per Area Code by Churn', fontsize=12)
ax.tick_params(axis='both', labelsize=10, rotation=0)
plt.xlabel('Area Code', fontsize=10)
plt.ylabel('Count of Customers', fontsize=10)

# Show plot
plt.show()
```

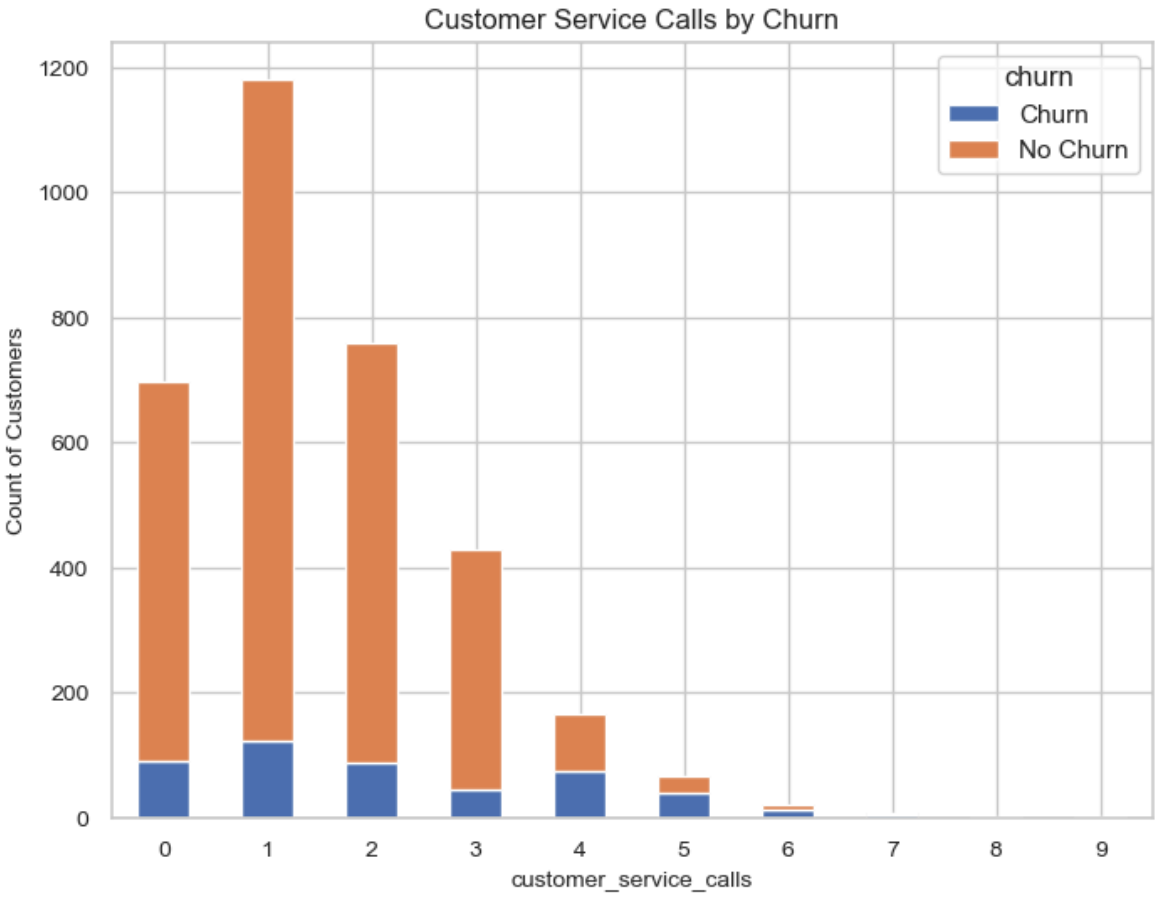


From the above graph, the churn rate is relatively the same accross the three area codes in our dataset, i.e., approximately 14% for all

__Churn and Customer Service Call__

In [22]:
```
# Plotting bar chart
fig, ax = plt.subplots(figsize=(8, 6))
counts = data.groupby(['customer_service_calls', 'churn']).size().unstack()
counts.plot(kind='bar', stacked=True, ax=ax)

# Set plot title and labels
plt.title('Customer Service Calls by Churn', fontsize=12)
ax.tick_params(axis='both', labelsize=10, rotation=0)
```

```
plt.xlabel('customer_service_calls', fontsize=10)
plt.ylabel('Count of Customers', fontsize=10)

# Show plot
plt.show()
```



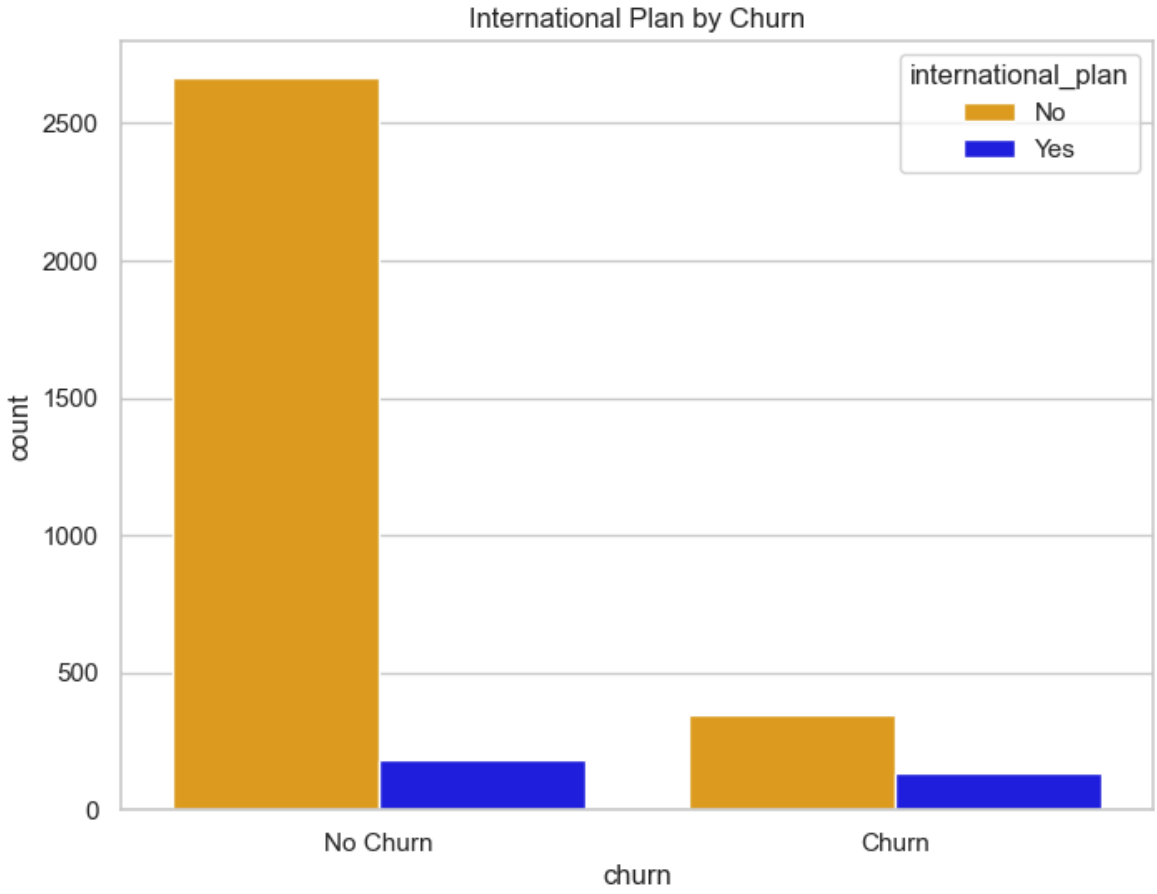
In [23]:
```
## because putting labels i the above graph will clump it, let me do a table ins

# Group data by 'area_code' and 'churn' and calculate counts
churn_summary = data.groupby(['customer_service_calls', 'churn']).size().unstack

# Calculate percentages
total_customers_per_area = churn_summary.sum(axis=1)
churn_percentage_per_area = (churn_summary['Churn'] / total_customers_per_area)

# Combine counts and percentages into a single DataFrame
churn_summary['No Churn Percentage'] = 100 - churn_percentage_per_area
churn_summary['Churn Percentage'] = churn_percentage_per_area

# Rename columns
churn_summary.columns = ['No Churn Count', 'Churn Count', 'No Churn Percentage',

# Display the DataFrame
print(churn_summary.to_string())
```

customer_service_calls	No Churn Count	Churn Count	No Churn Percentage	Churn Percentage
0	92.0	605.0	86.800574	13.199426
1	122.0	1059.0	89.669771	10.330229
2	87.0	672.0	88.537549	11.462451
3	44.0	385.0	89.743590	10.256410
4	76.0	90.0	54.216867	45.783133
5	40.0	26.0	39.393939	60.606061
6	14.0	8.0	36.363636	63.636364
7	5.0	4.0	44.444444	55.555556
8	1.0	1.0	50.000000	50.000000
9	2.0	NaN	0.000000	100.000000

From the above graph and table, we can see that the churn rate varies accross the different number of phone calls that have been made

__Churn and International Plan__

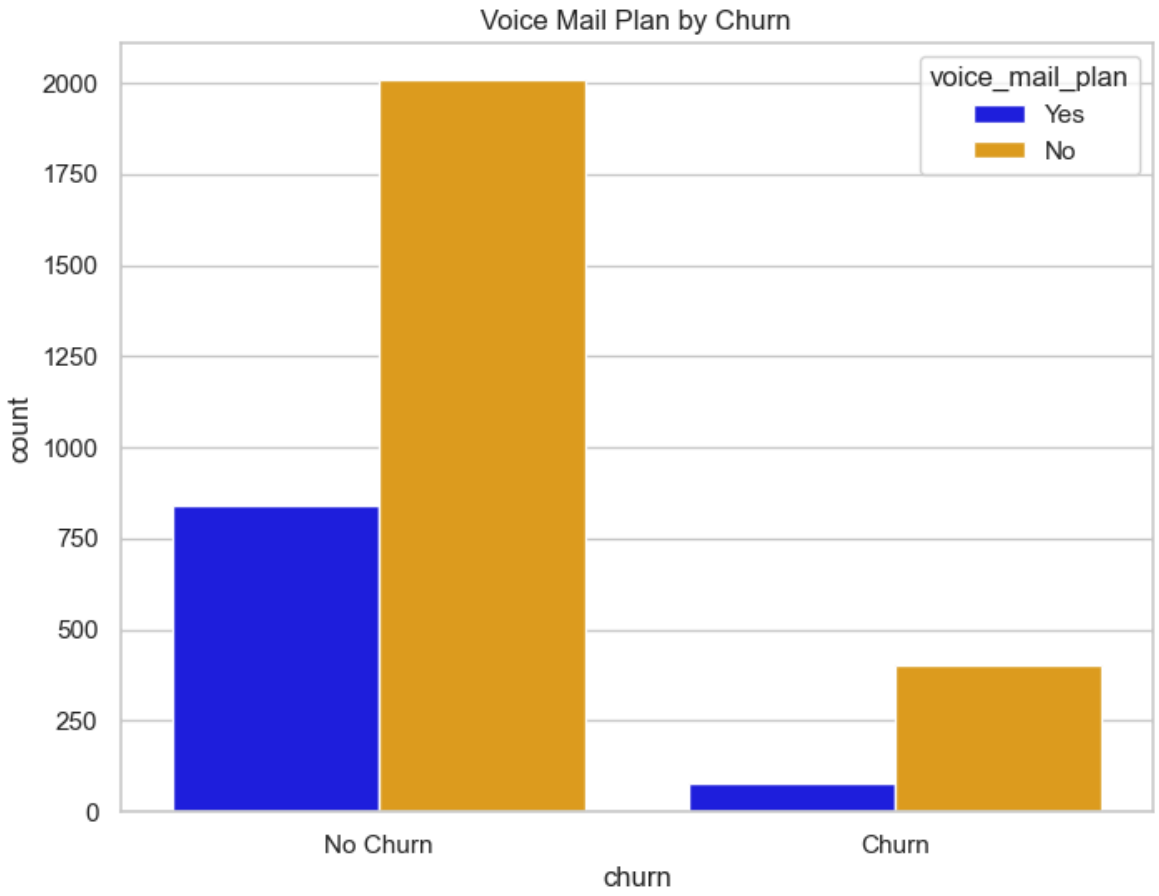
In [24]:
```
# Define colors for each category
colors = ['orange', 'blue']

plt.figure(figsize=(8, 6))
sns.countplot(x='churn', hue='international_plan', data=data, palette=colors)
plt.title('International Plan by Churn')
plt.show()
```



From the above graph those who are not on international plan are the highest out of those who churned and those who did not churn

__Churn and Voice Mail Plan__

In [25]:
```
# Define colors for each category
colors = ['blue', 'orange']

plt.figure(figsize=(8, 6))
sns.countplot(x='churn', hue='voice_mail_plan', data=data, palette=colors)
plt.title('Voice Mail Plan by Churn')
plt.show()
```



From the above graph out of those who did not churn not on voice mail plan, this also applies to those who churned

The churn column wa boolean, lets return it to its initial state then convert it to integer

In [26]:
```
data['churn'] = data['churn'].replace({'No Churn': False, 'Churn': True}) ## Ret
data['churn'] = data['churn'].astype(int) ## converting it to integer
print(data['churn'].dtype) ## checking the type of colmn
data.sample(2) ## previewing the data
```

```
int32
```

Out[26]:

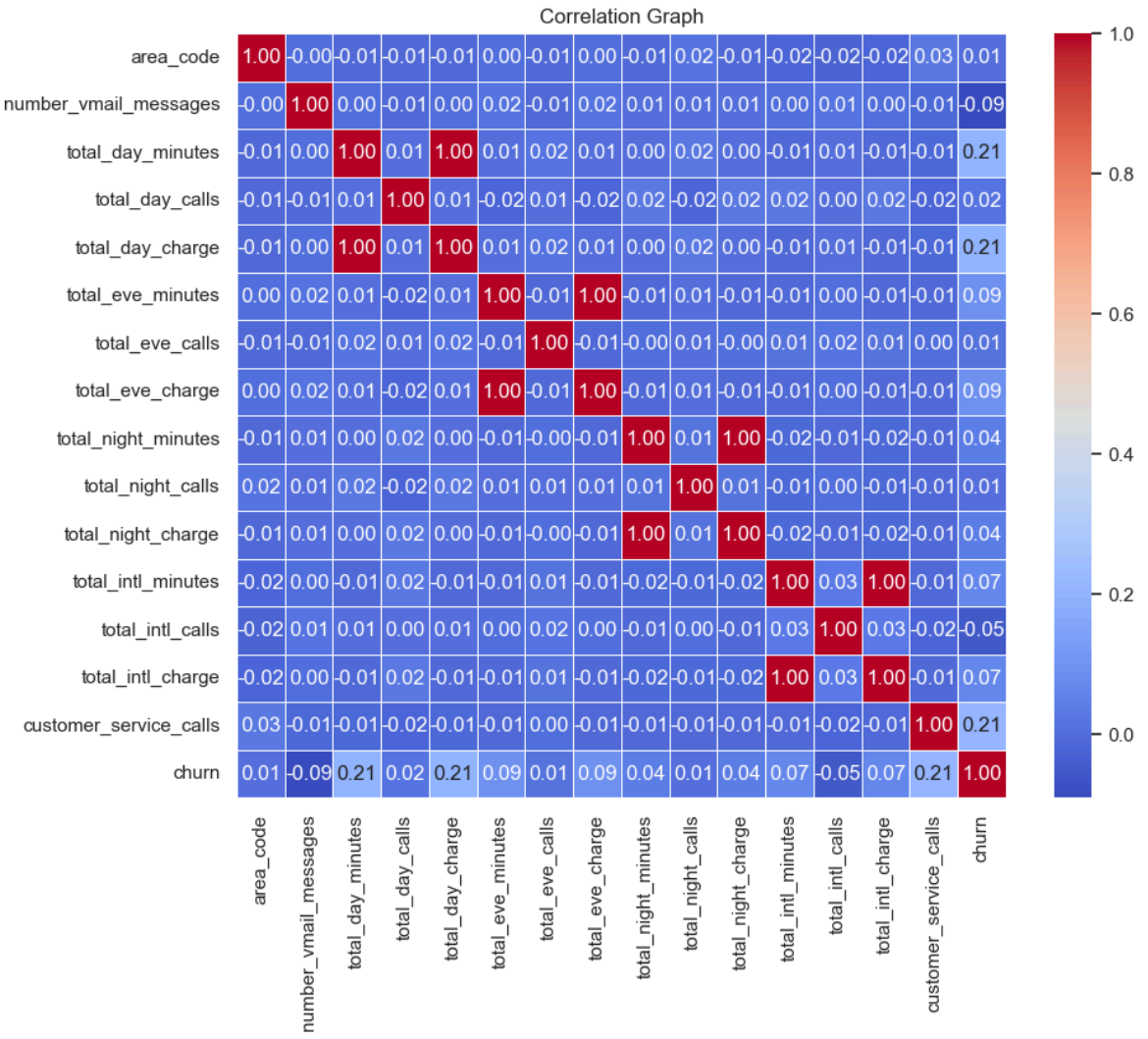
	state	account_length	area_code	international_plan	voice_mail_plan	number_vm
3173	OR	133	415	No	No	
3188	ND	148	415	Yes	No	

In [27]:
```
##Lets first drop some columns that we will not need in our correlation graphs
drop_columns = ['state', 'account_length', 'international_plan', 'voice_mail_pla
data2 = data.drop(columns = drop_columns)
```

__Correlation__

In [28]:
```
corr_matrix = data2.corr()
# Plot correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
```

```
plt.title('Correlation Graph')
plt.show()
```



In [29]:
```
## Lets run this correlatios with respect to churn column
data2.corr()['churn'].sort_values(ascending = False)
```

Out[29]:
```
churn                     1.000000
customer_service_calls    0.208750
total_day_minutes         0.205151
total_day_charge          0.205151
total_eve_minutes         0.092796
total_eve_charge          0.092786
total_intl_charge         0.068259
total_intl_minutes        0.068239
total_night_charge        0.035496
total_night_minutes       0.035493
total_day_calls           0.018459
total_eve_calls           0.009233
area_code                 0.006174
total_night_calls         0.006141
total_intl_calls         -0.052844
number_vmail_messages    -0.089728
Name: churn, dtype: float64
```

From the above graph and table we can see that nearly all the variables have atleast some relationship with our target varibale, apart from total_intl_calls and number_vmail_messages which have weak

`negative correlations to churn, the rest of the variables have weak positive correlation to our target varibale(churn)`

From the correlation matric graph, columns of focus are the following;

- total_day_minutes
- total eve minutes
- total night minutes
- total intl minutes

__Check for multicollinearity__

Will check for multicollenearity on the above mentioned variables using VIF(Variance Inflation Factor). With VIF are measuring how much variance of the estimated regression coefficients is increased because of multicollinearity of its predictors. This is how we are going to interpret the results

- VIF = 1: No multicollinearity
- VIF > 1 and < 5: Moderate multicollinearity
- VIF > 5: High multicollinearity

In [30]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
cols = data2[['total_day_minutes', 'total_eve_minutes', 'total_night_minutes', '

# Getting VIF for each variable
vif_data = pd.DataFrame()
vif_data["feature"] = cols.columns
vif_data["VIF"] = [variance_inflation_factor(cols.values, i) for i in range(len(

vif_data
```

Out[30]:

	feature	VIF
0	total_day_minutes	9.673057
1	total_eve_minutes	12.026619
2	total_night_minutes	12.000415
3	total_intl_minutes	10.844008

`We see that all the columns above have exceeded the VIF=5 mark`

In [31]:
```
data.head(3)
```

Out[31]:

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_
0	KS	128	415	No	Yes	
1	OH	107	415	No	Yes	
2	NJ	137	415	No	No	

`From here now let us drop all the variables with VIF>5` from our original dataset called data

In [32]:
```
cols_to_drop = ['total_day_minutes', 'total_eve_minutes', 'total_night_minutes',
data3 = data.drop(cols_to_drop, axis=1)
print(data3.shape)
data3.head(2)
```

```
(3333, 16)
```

Out[32]:

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_
0	KS	128	415	No	Yes	
1	OH	107	415	No	Yes	

PRE-PROCESSING

In [33]:
```
#Lets check again our variable types
data3.dtypes
```

Out[33]:
```
state                    object
account_length            int64
area_code                 int64
international_plan       object
voice_mail_plan          object
number_vmail_messages     int64
total_day_calls           int64
total_day_charge        float64
total_eve_calls           int64
total_eve_charge        float64
total_night_calls         int64
total_night_charge      float64
total_intl_calls          int64
total_intl_charge       float64
customer_service_calls    int64
churn                     int32
dtype: object
```

`We can see that there some object variables in our data set`

`We are going to use OHE(One Hot Encoding) technique to handle this categorical variables`

In [34]:
```
# Selecting all the categorical variables we have
columns_categorical = data3.select_dtypes(include='object').columns.tolist()
columns_categorical
```

Out[34]:
```
['state', 'international_plan', 'voice_mail_plan']
```

In [35]:
```
df = pd.get_dummies(data, columns=columns_categorical, drop_first = True)
## also notice that now our new dataset is called df
df.head(2)
```

Out[35]:

	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_call
0	128	415	25	265.1	11(
1	107	415	26	161.6	12:

2 rows × 69 columns

__Splitting our dataset into Test and Train set__

In [36]:
```
#importing the necessary library
from sklearn.model_selection import train_test_split
##Setting the target(y) and features variables(X)
X = df.drop('churn',axis = 1)
y = df['churn']

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

__Model Imbalance check__

Here we will basically check the distribution of our target varibale, in this case the churn rate **(NB: We already have a graph for it in the univariate analysis section).***

When the classes are not represented equally then we will know that that there is an imbalance case scenario.

Because we already have graph as up there, lets do a simple table for recap and to check again this imbalance.

This check is important since it will help us select the appropriate modelling techniques and also evaluate our model perfomance

In [37]:
```
# Extract the column you want to display as a table
column_name = 'churn'
column_data = df[column_name]

# Calculate value counts and percentages
value_counts = column_data.value_counts()
value_percentages = (value_counts / len(column_data)) * 100

# Creating a dataframe
result_df = pd.DataFrame({'Count': value_counts, 'Percentage': value_percentages

# Print the DataFrame
print(result_df)
```

```
       Count  Percentage
churn
0       2850   85.508551
1        483   14.491449
```

`Zero represent False i.e. No churn at 85.5%, and 1 represents True i.e., churn at 14.5%.`

We can now conclude that indeed our dataset is imbalanced.

__Handling Imbalance__

We are going to use a resampling technique called SMOTE- Synthetic Minority Over-sampling Technique.

SMOTE just increase the number of instances in the minority class so as to balance the class distribution

In [38]:
```
## Importing the necessary libary
from imblearn.over_sampling import SMOTE

##Applying smote to balance our dataset
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

##Lets see if class imbalance has been solved
pd.Series(y_train_resampled).value_counts()
```

Out[38]:
```
churn
0    2284
1    2284
Name: count, dtype: int64
```

`We are seeing now that our target variable(churn) has been balanced, implying model bias has been reduced`

__Scaling__

Now ensuring that all our features have the same scale/prevents features with high magnitude to dominate the model

In [39]:
```
# importing the necessary library
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() # Create StandardScaler object
X_train_scaled = scaler.fit_transform(X_train_resampled) # Fit and transform the
X_test_scaled = scaler.fit_transform(X_test) # Fit and transform the features
# X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns) # Converting
# X_train_scaled.head(10)
```

MODELLING

__Baseline Model-LOGISTIC REGRESSION__

In [40]:
```
## importing the neccesary library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, precision_sco

model_lr = LogisticRegression() ##Lets instantiate our model
model_lr.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_lr = model_lr.predict(X_test_scaled) # Make predictions on the test data
#print(classification_report(y_test, y_pred)) # Evaluate model performance(this

# Evaluate model performance
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
```

```
print("Accuracy:", accuracy_lr)
print("Precision:", precision_lr)
print("Recall:", recall_lr)
print("F1-score:", f1_lr)
```

```
Accuracy: 0.43478260869565216
Precision: 0.1960352422907489
Recall: 0.8811881188118812
F1-score: 0.3207207207207207
```

Logistic Regression Results

- *Accuracy*

`Our accurabcy level is quite low, at 43%, this could be because of the high imbalanced nature of our target variable` , our model can only predict 43% correctly

- *Precision*

`Our model can only measures 19% correct positive predictions`

- *Recall*

`Our model performs well here, out of all the actual positives, it correctly predicted is 88%`

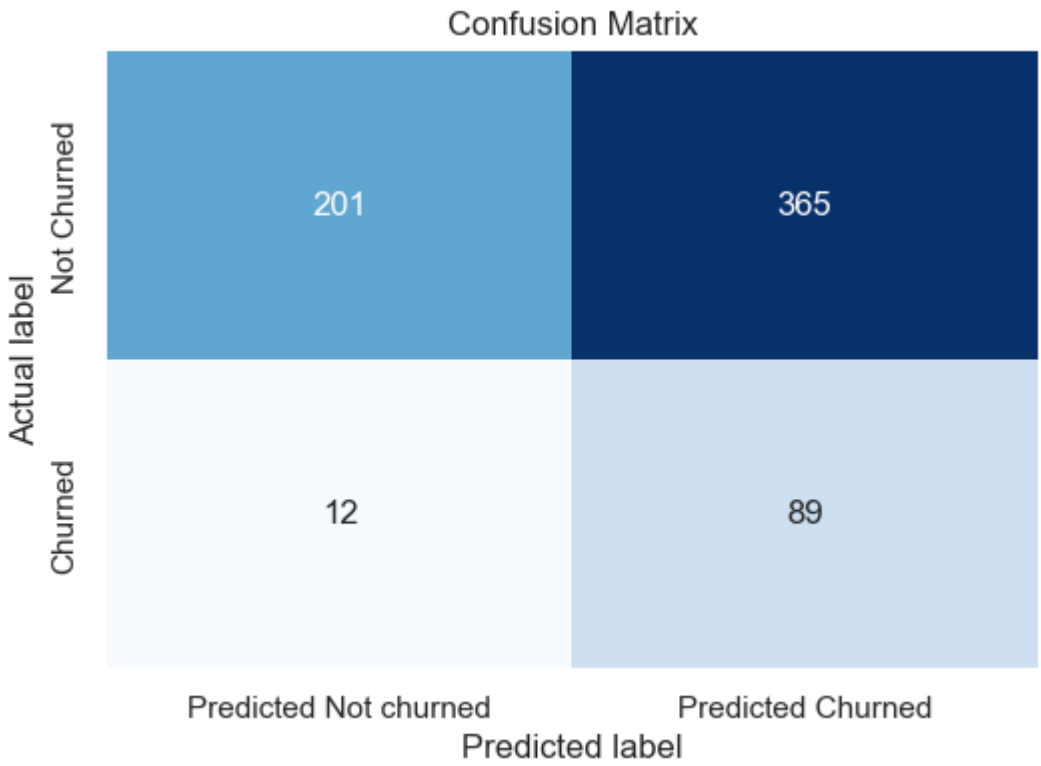
- *F1_Score*

`In our model, the balance of precision and recall is at 32% which is quite low, meaning our overall performance is low`

Now lets proceed to build a confusion matrix to even better understand the performance of our model i.e.,

- Get the actual counts
- See how well how model is perfom in this imbalanced dataset
- We will usee it for comparison with other models

In [41]:
```
cm_lr = confusion_matrix(y_test, y_pred_lr)

class_names1 = ['Predicted Not churned', 'Predicted Churned']
class_names2 = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```



From the matrix we can see that 365 entries were incorrectly predicted

__Model 2 - Decision Tree Classifer Model__

Here the data is split based on feature values and decision is made at each node to classify the data. It is like a tree which keeps forming branches. Each brance is an outcome of a decisoon.

It is an ensemble method.

In [42]:
```
## Import necessary library
from sklearn.tree import DecisionTreeClassifier

model_dt = DecisionTreeClassifier(random_state=42) ##Lets instantiate our model
model_dt.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_dt = model_dt.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)

print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1-score:", f1_dt)
```

```
Accuracy: 0.8620689655172413
Precision: 0.5310344827586206
Recall: 0.7623762376237624
F1-score: 0.6260162601626016
```

Decision Trees Results

- *Accuracy*

`Our accuracy level has significantly improved, at 86%, our model is perfoming well and can correctly predict 86%`

- *Precision*

`Our model can make 53% correct positive predictions which is an improvement from the previous model`

- *Recall*

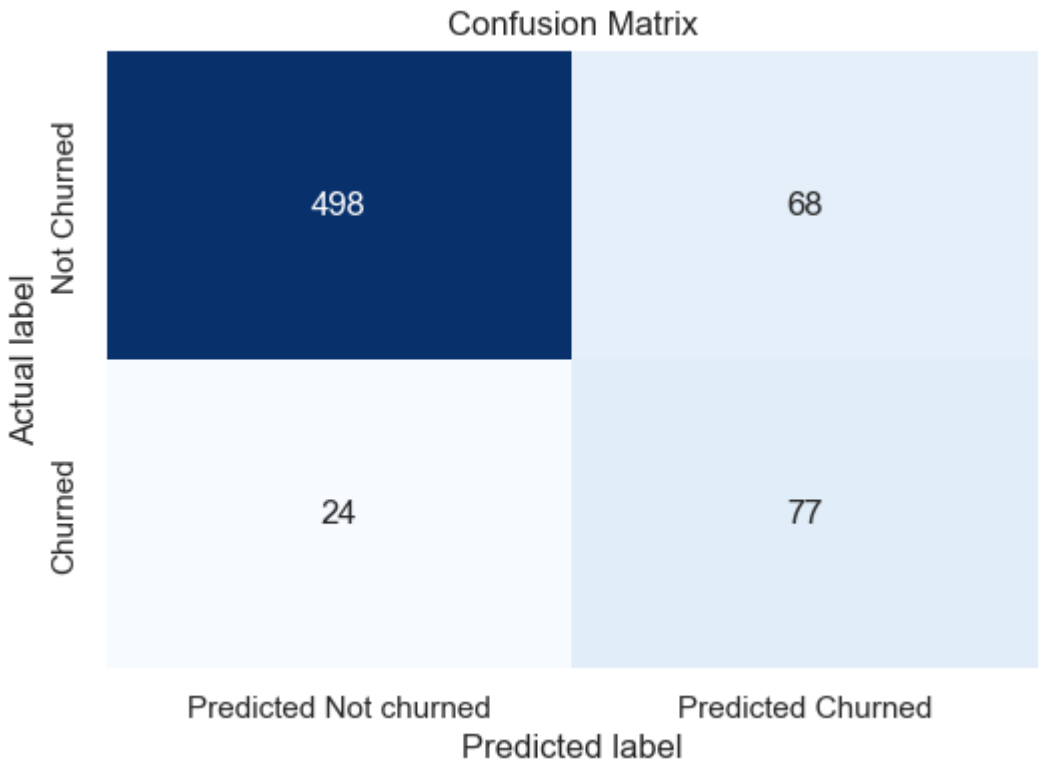
`This has reduced slightly to 76% which is still good enough, this means that out of all the actual positives, our model corretly predicted 76%`

- *F1_Score*

`We have a very significant improvement from our previous model here, the balance between precision and recall is 62% which is above average now`

Now lets proceed to build a confusion matrix to even better understand the performance of our second model better

In [43]:
```
cm_dt = confusion_matrix(y_test, y_pred_dt)

class_names1_dt = ['Predicted Not churned', 'Predicted Churned']
class_names2_dt = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```



From the matrix we can actually confirm our accuracy levels, the model correctly predicted 498 entries as not churned, this is a good perfomance so far compared to the prevous one

__Model 3 - Random Forest Classifier Model__

This is a combination of multiple decision trees created randomly and used to make prediction. Each decison tree in the forest then indipendently makes a prediction on the target and the final pridiction is determined by taking a majority of the vote amongst the decision trees/averaging the predictions

In [44]:
```
## Import the necessary library
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(random_state=42) ##Lets instantiate our model
model_rf.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_rf = model_rf.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)
```

```
Accuracy: 0.904047976011994
Precision: 0.6528925619834711
Recall: 0.7821782178217822
F1-score: 0.7117117117117117
```

Random Forest Results

- *Accuracy*

`Our accuracy level has significantly improved compared to previous models, now at 90%, our model is perfoming well and can correctly predict 90%`

- *Precision*

`Our model can make 65% correct positive predictions which is an improvement from the previous 2 models`

- *Recall*

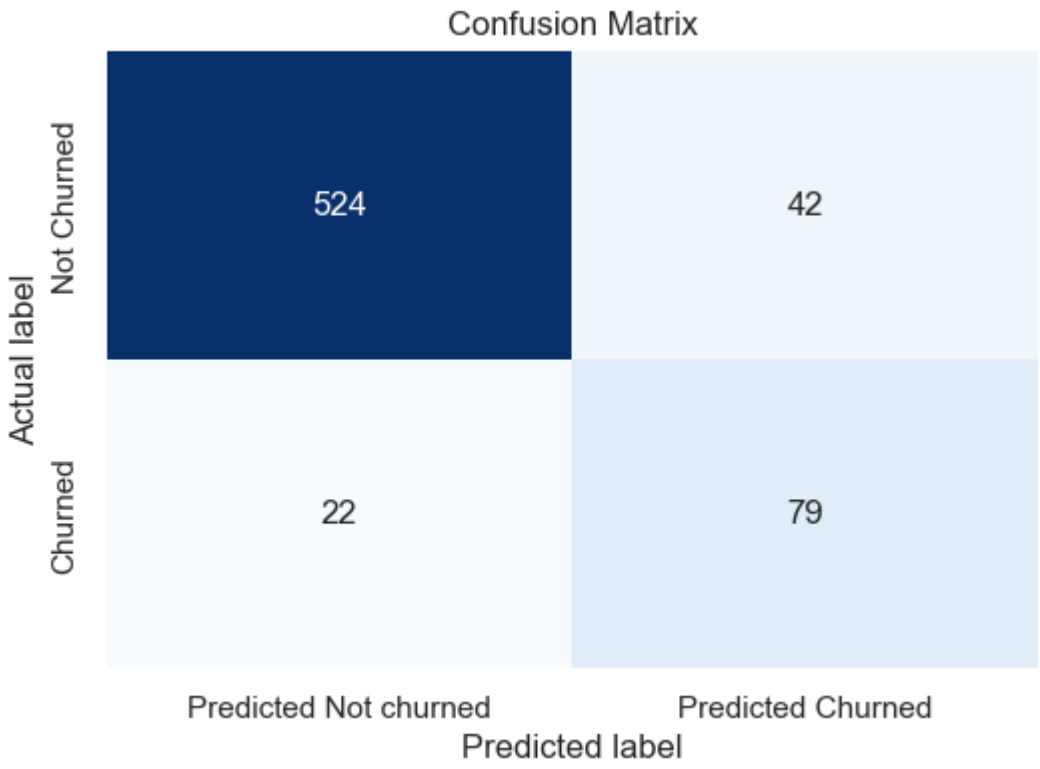
`This has increased slightly to 78% from decision tree model and is still good enough, this means that out of all the actual positives, our model corretly predicted 78%`

- *F1_Score*

`We have a very significant improvement from our previous 2 models, the balance between precision and recall is 71% which is above average now, meaning our model perfoming better than the previous 2 models`

Now lets proceed to build a confusion matrix to even better understand the performance of our third model better

In [45]:
```
cm_rf = confusion_matrix(y_test, y_pred_rf)

class_names1_rf = ['Predicted Not churned', 'Predicted Churned']
class_names2_rf = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```



`This is way better performanc in terms of accuracy, 524 were correctly predicted as not churned`

__Model 4 - Gradient Boosting Classifier Model__

Here, new trees are created whereas each of the new tree created corrects errors made by the previous trees and as a results makes better and accurate predictions i.e., (combine several weak learners to create a strong learner).

It is an ensemble technique

In [46]:
```
## importing the necesary library
from sklearn.ensemble import GradientBoostingClassifier

model_gr = GradientBoostingClassifier(random_state=42) ##Lets instantiate our mo
model_gr.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_gr = model_gr.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_gr = accuracy_score(y_test, y_pred_gr)
precision_gr = precision_score(y_test, y_pred_gr)
recall_gr = recall_score(y_test, y_pred_gr)
f1_gr = f1_score(y_test, y_pred_gr)

print("Accuracy:", accuracy_gr)
print("Precision:", precision_gr)
print("Recall:", recall_gr)
print("F1-score:", f1_gr)
```

```
Accuracy: 0.8665667166416792
Precision: 0.5379746835443038
Recall: 0.8415841584158416
F1-score: 0.6563706563706563
```

Gradient Boosting Results

- *Accuracy*

```
Our accuracy level is still high, now at 87%, our model is perfoming well and can correctly predict 87% of the data
```

- *Precision*

```
Our model can make 53% correct positive predictions which is relatively low
```

- *Recall*

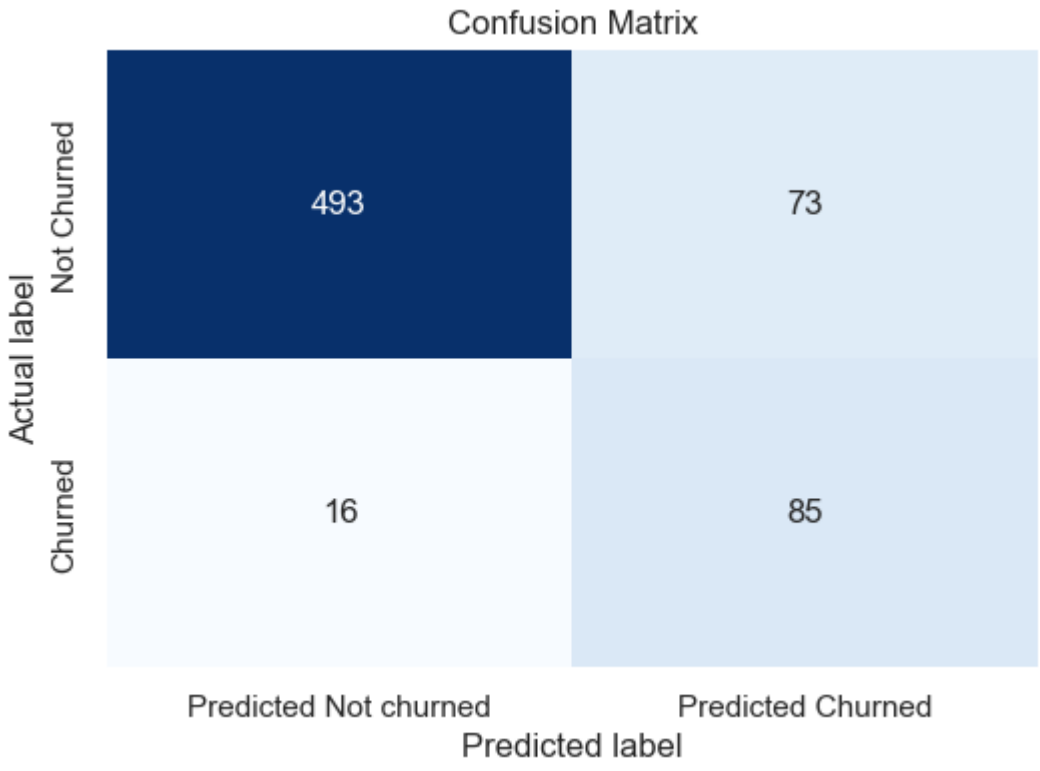
```
Out of all the actual positives, our model corretly predicted 85%
```

- *F1_Score*

```
The balance between precision and recall is 66% which is average
```

Now lets proceed to build a confusion matrix to even better understand the performance of our fourth model better

In [47]:
```
cm_gr = confusion_matrix(y_test, y_pred_gr)

class_names1_gr = ['Predicted Not churned', 'Predicted Churned']
class_names2_gr = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_gr, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```



`Our model correctly predicted 493 record as not churned, this is still a relatively good level of accuracy`

__Model 5 - K-Nearest Neighbors (KNN) Model__

It simply finds the closet datapoints(neighbors) in the training dataset to the new data point that we want to classify.

In [48]:
```
## import the necessary library
from sklearn.neighbors import KNeighborsClassifier

model_kn = KNeighborsClassifier(n_neighbors=5) ##we have set the number of neigh
model_kn.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_kn = model_kn.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_kn = accuracy_score(y_test, y_pred_kn)
precision_kn = precision_score(y_test, y_pred_kn)
recall_kn = recall_score(y_test, y_pred_kn)
f1_kn = f1_score(y_test, y_pred_kn)

print("Accuracy:", accuracy_kn)
print("Precision:", precision_kn)
print("Recall:", recall_kn)
print("F1-score:", f1_kn)
```

```
Accuracy: 0.8365817091454273
Precision: 0.3888888888888889
Recall: 0.13861386138613863
F1-score: 0.20437956204379562
```

KNN Results

The model generally seems to be performing poorer than the previuos models

- *Accuracy*

`Our accuracy level is still high,at 83%, this model is perfoming well and can correctly predict 83% of the data`

- *Precision*

`Our model can make 39% correct positive predictions which is low`

- *Recall*

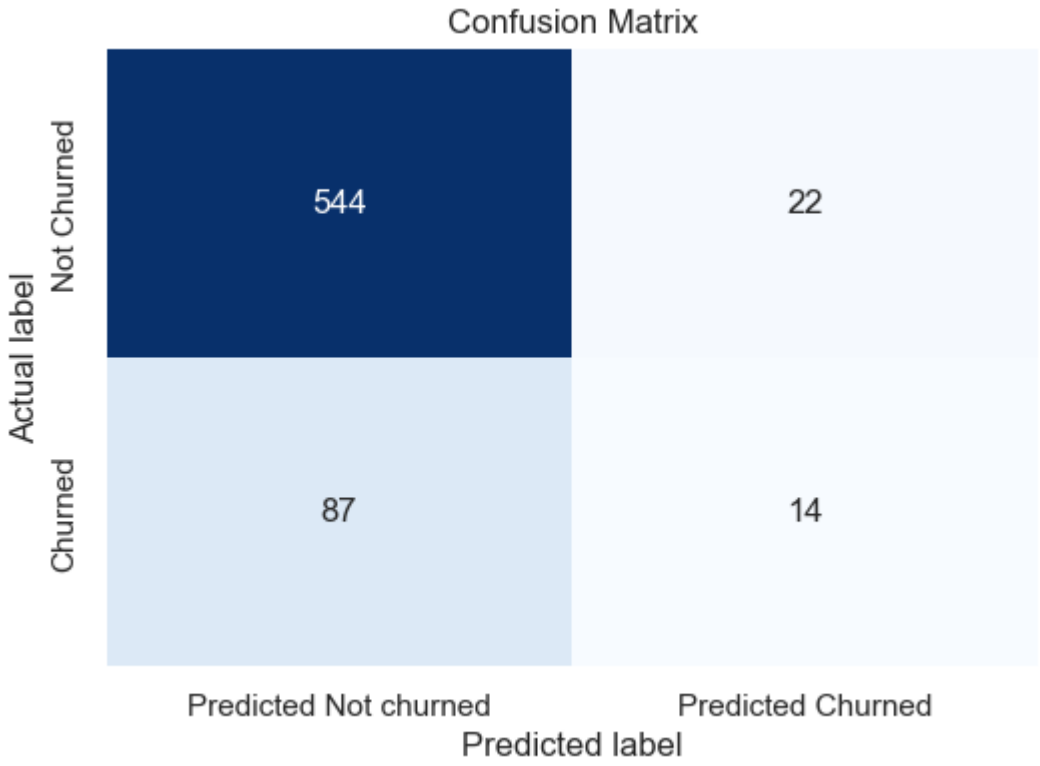
`Out of all the actual positives, our model corretly predicted 14%, this very low`

- *F1_Score*

`The balance between precision and recall is 20%, meaninf our model is performing poorly`

Now lets proceed to build a confusion matrix to even better understand the performance of our fifth model better

In [49]:
```
cm_kn = confusion_matrix(y_test, y_pred_kn)

class_names1_kn = ['Predicted Not churned', 'Predicted Churned']
class_names2_kn = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_kn, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```




```
This model correctly predecited 544 customers as not churned
```

__Model 6 - XGBoost Model__ *(Extreme Gradient Boosting)*

This is an optimized implementation of gradient boosting.

In [50]:
```
#Loading the necessary libarary
import xgboost as xgb
from xgboost import XGBClassifier

model_xg = XGBClassifier(random_state=42) ## initializing this model
model_xg.fit(X_train_scaled, y_train_resampled) # Fit the model to the training
y_pred_xg = model_xg.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_xg = accuracy_score(y_test, y_pred_xg)
precision_xg = precision_score(y_test, y_pred_xg)
recall_xg = recall_score(y_test, y_pred_xg)
f1_xg = f1_score(y_test, y_pred_xg)

print("Accuracy:", accuracy_xg)
print("Precision:", precision_xg)
```

```
print("Recall:", recall_xg)
print("F1-score:", f1_xg)
```

```
Accuracy: 0.8350824587706147
Precision: 0.47513812154696133
Recall: 0.8514851485148515
F1-score: 0.6099290780141844
```

XGBoost Results

- *Accuracy*

`Our accuracy level is still high,at 83%, this model is perfoming well and can correctly predict 83% of the data`

- *Precision*

`Our model can make 43% correct positive predictions which is still low`

- *Recall*

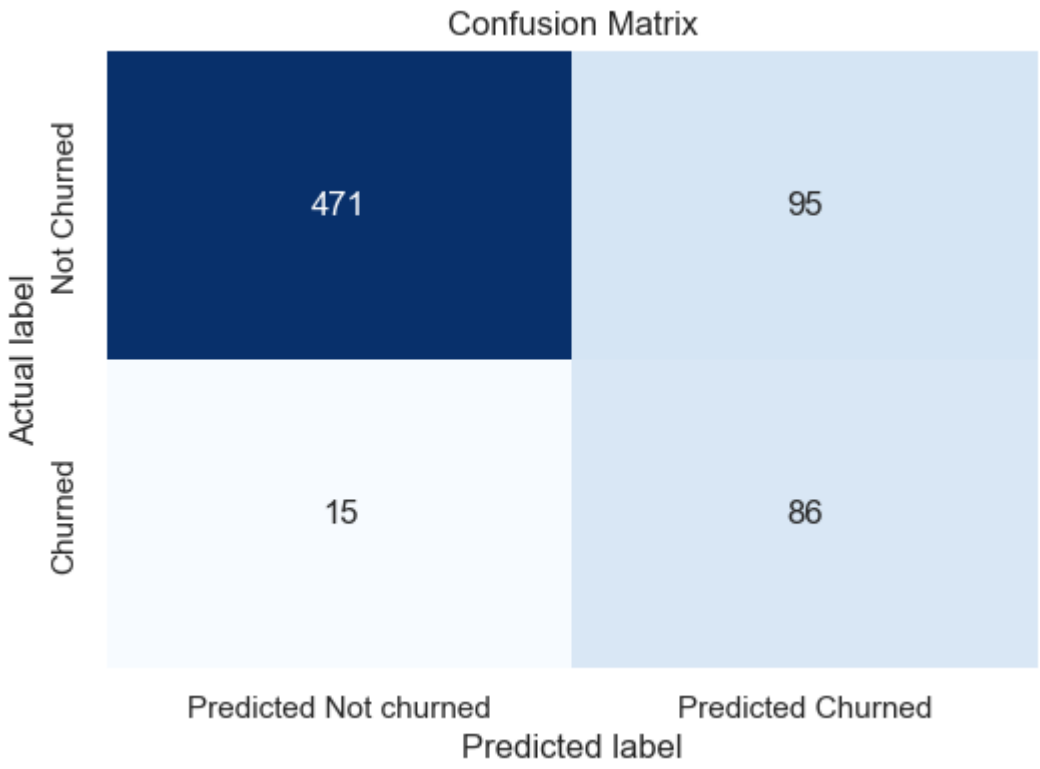
`Out of all the actual positives, our model corretly predicted 85%, this is very good`

- *F1_Score*

`The balance between precision and recall is 61%, which is average performance`

Now lets proceed to build a confusion matrix to even better understand the performance of our sixth model better

In [51]:
```
cm_xg = confusion_matrix(y_test, y_pred_xg)

class_names1_xg = ['Predicted Not churned', 'Predicted Churned']
class_names2_xg = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_xg, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```



`This model correctly predicted 471 customers as not churned which is good performance`

__Model 7 - Cross-Validation__

It splits the data into multiple subsets, then trains the model on some subsets and tests in on other subsets, thus checking the performance on different parts of the dataset. This is done repeatedly then an average performance of the model is derived

In [52]:
```
#Loading the necessary libarary
from sklearn.model_selection import cross_val_score

model_cr = RandomForestClassifier(random_state=42) # Initialize the Random Fores
cv_scores = cross_val_score(model_cr, X_train_scaled, y_train_resampled, cv=5, s

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())


model_cr.fit(X_train_scaled, y_train_resampled) # Train the model on the entire
y_pred_cr = model_cr.predict(X_test_scaled) # Make predictions on the test data

# Evaluate model performance
accuracy_cr = accuracy_score(y_test, y_pred_cr)
precision_cr = precision_score(y_test, y_pred_cr)
recall_cr = recall_score(y_test, y_pred_cr)
f1_cr = f1_score(y_test, y_pred_cr)

print("Accuracy:", accuracy_cr)
print("Precision:", precision_cr)
print("Recall:", recall_cr)
print("F1-score:", f1_cr)
```

```
Cross-Validation Scores: [0.92997812 0.96170678 0.9595186  0.95399781 0.96933187]
Mean CV Accuracy: 0.9549066366919836
Accuracy: 0.904047976011994
Precision: 0.6528925619834711
Recall: 0.7821782178217822
F1-score: 0.7117117117117117
```

Cross Validation Results

- *Accuracy*

`Our accuracy level is very high,at 90%, this model is perfoming well and can correctly predict 90% of the data`

- *Precision*

`Our model can make 65% correct positive predictions which is good`

- *Recall*

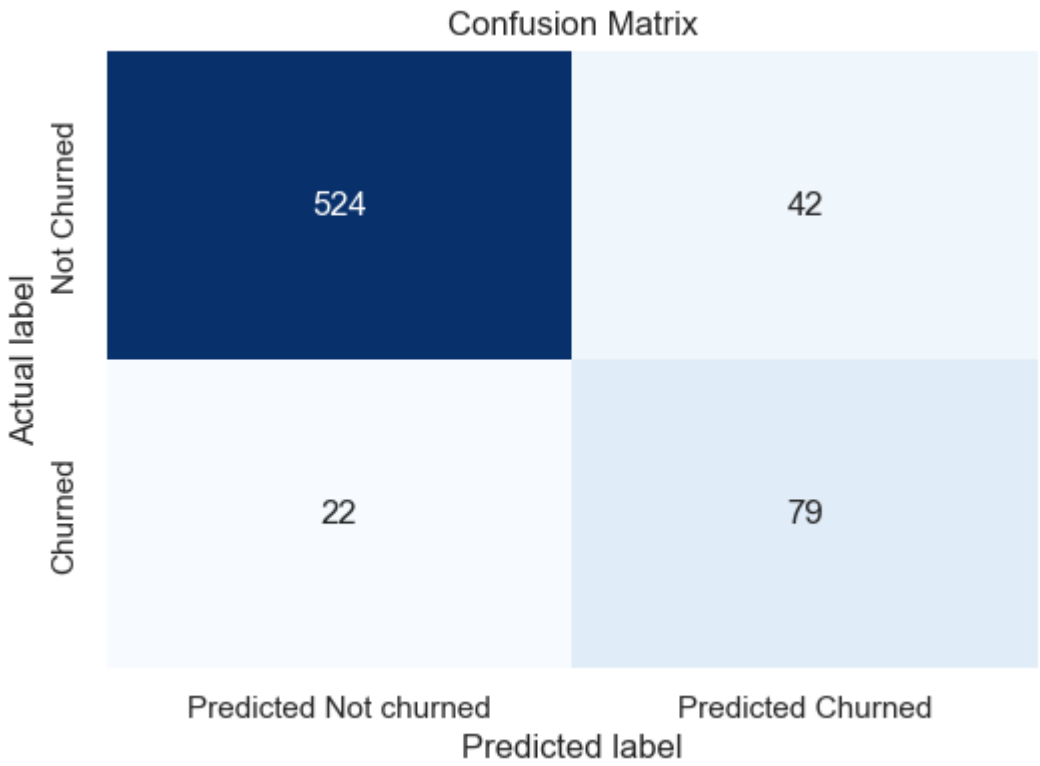
`Out of all the actual positives, our model corretly predicted 78%, this is very good`

- *F1_Score*

`The balance between precision and recall is 71%, which is good performance`

These results are the same as those of the random forest model

In [53]:
```
cm_cr = confusion_matrix(y_test, y_pred_cr)

class_names1_cr = ['Predicted Not churned', 'Predicted Churned']
class_names2_cr = ['Not Churned', 'Churned']

plt.figure(figsize=(6, 4))
sns.heatmap(cm_cr, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=cl
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.title('Confusion Matrix')
plt.show()
```




```
Our model accurately predicted 524 as not churned
```

__Best Model Selection__

We have conducted 7 models for a dataset with the base model being the logistic regression model. We also evaluated the models based on the following;-
* Accuracy * Precision * Recall * F1_Score
Of all the models **Random Forest Model** was the best as it had the best accuracy and besy F1_Score. So weill drop the other models and proceed with this.

__Hyperparameter Tuning__
Now we simple want to optimize the performanc of our chosen model. i.e., maximize the performance of our machine learning model to improve accuracy, generalization and its robustness.
In our case we want to conduct hyperparameter tuning on the Random forest model

In [54]:
```
## import necessary libabry
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


# model_rf = RandomForestClassifier(random_state=42) ##Lets instantiate our mode
# model_rf.fit(X_train_scaled, y_train_resampled) # Fit the model to the trainin
```

```
# y_pred_rf = model_rf.predict(X_test_scaled) # Make predictions on the test dat


model = RandomForestClassifier(random_state=42) # Initialize Random Forest class

# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train_resampled)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Use the best parameters to train the final model
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train_scaled, y_train_resampled)

# Make predictions on the test data
y_pred = best_model.predict(X_test_scaled)
```

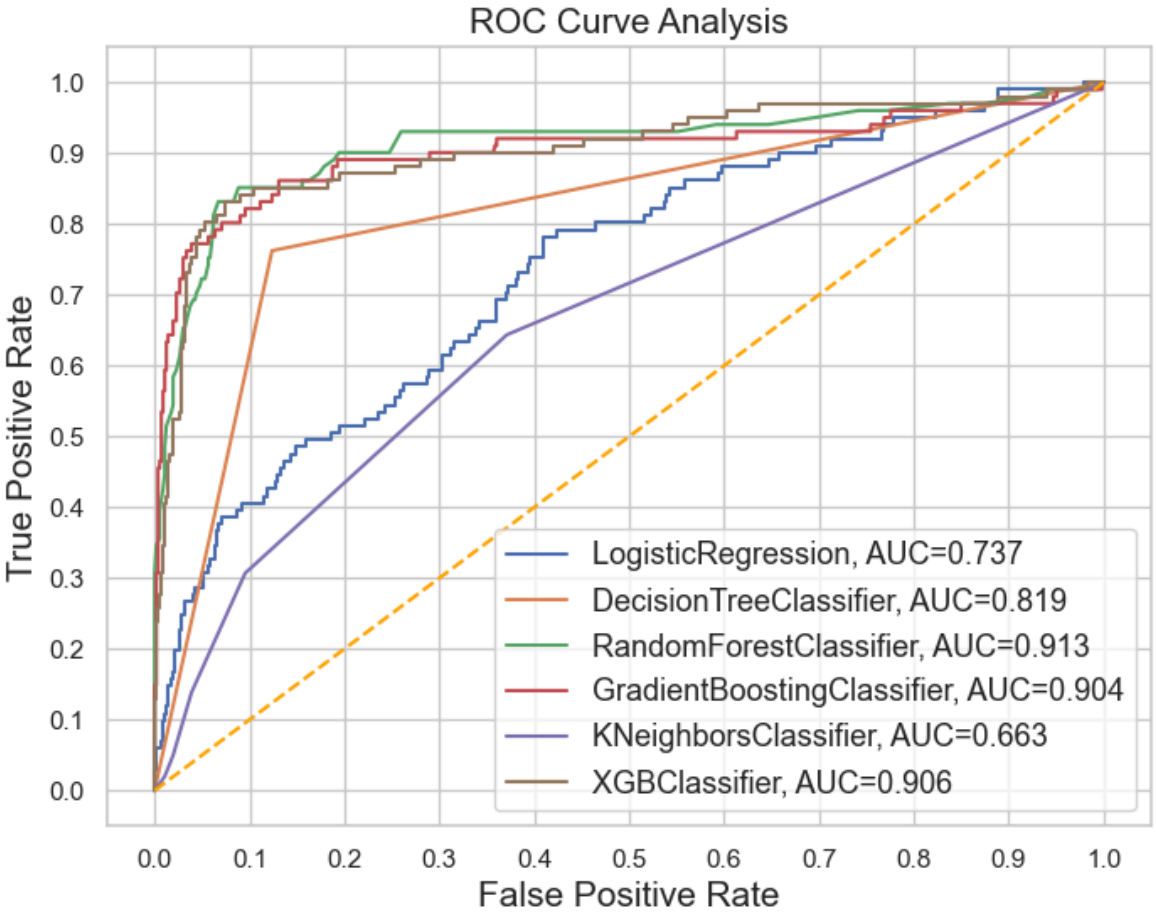
```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 300}
Best Score: 0.9555638108431339
```

Models Comparison with ROC Curve

In [55]:

```
##import necessary libabry
from sklearn.metrics import roc_curve, roc_auc_score

np.random.seed(123)
classifiers = [LogisticRegression(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               GradientBoostingClassifier(),
               KNeighborsClassifier(),
               XGBClassifier()]

# Initialize an empty list to store the results
results = []

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train_scaled, y_train_resampled)
    yproba = model.predict_proba(X_test_scaled)[:,1]  # Use X_test_scaled here i

    fpr, tpr, _ = roc_curve(y_test, yproba)
    auc = roc_auc_score(y_test, yproba)

    # Append the results to the list
    results.append({'classifiers':cls.__class__.__name__,
                    'fpr':fpr,
                    'tpr':tpr,
                    'auc':auc})

# Convert the list of dictionaries to a DataFrame
result_table = pd.DataFrame(results)
```

```
# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

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

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```

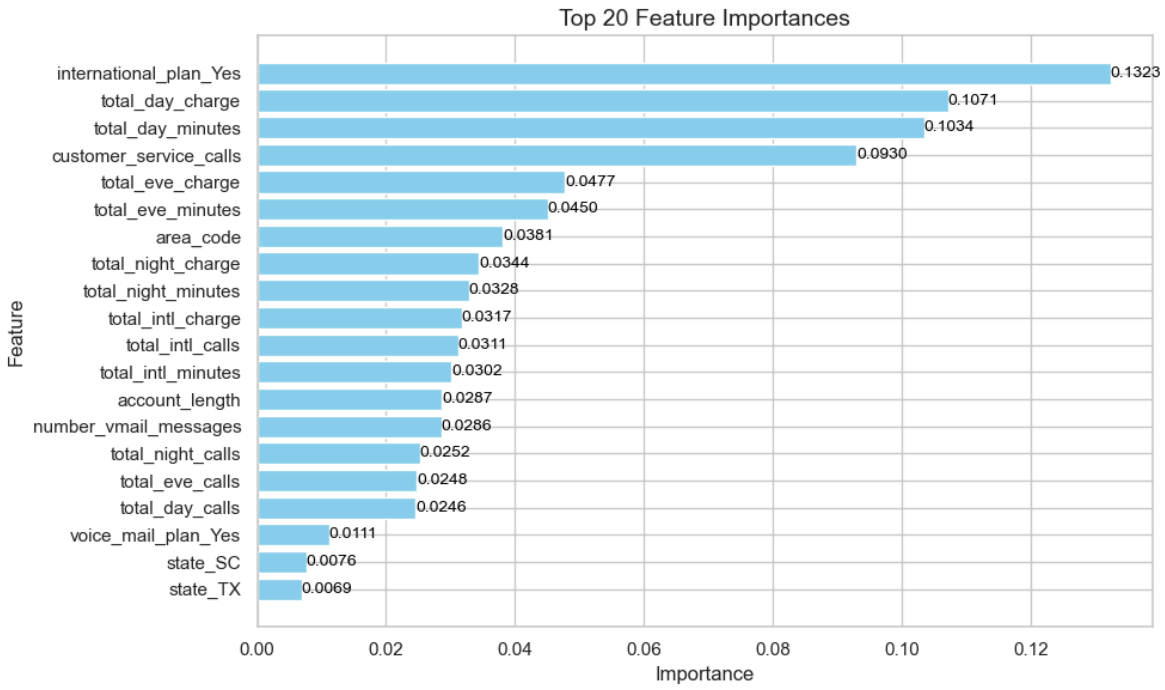


`According to this Random forest has the best performance(0.913) followed by XGB and Gradient Boosting. The least performing is KNN at 0.66.`

A higher AUC score indicates that the classifier is better at distinguishing between positive and negative instances.

Top Predictor

In [56]:
```
# Get feature importances from the trained Random Forest model
feature_importances = model_rf.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': featur

# Sort the DataFrame by feature importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascen

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'][:20], feature_importance_df['Importanc
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Top 20 Feature Importances', fontsize=14)
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at

# Add values on bars
for i, v in enumerate(feature_importance_df['Importance'][:20]):
    plt.text(v, i, f'{v:.4f}', color='black', va='center', fontsize=10)

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



CONCLUSION

The key items that determine a customer's churn are:-

- Internationa plan
- Total day charge and minutes
- Customer service calls

Handling these top 3 major factors will definitely reduce the customer churn.

RECOMMENDATION

Below are the major recommendations for SyriaTel company to consider:

- Better their customer service techniques so as to reduce this churn rate
- Provide credit offers and minute offers to customers on their communication during daytime
- Increase their international coverage/provide better options for their customers to make international calls

NEXT STEPS

The SyriaTel company should do the following:

- Do more research on the which other Telecommunication company are shifting to and why so as to better their services.
- After every specified duration, they should run an analysis to monitor customer churn rate so to mitigate losses do to high

In [ ]: