Group 8 collaborators:

```
Students Names:
    1. Nazra Nyangwara
    2. Kenneth Kimani
    3. Hellen Mwangi
    4. Denis Kipkorir
    5. Jonathan Okwaro
    6. Janet Khainza - Team Lead
```

SYRIATEL CUSTOMER CHURN PROJECT

1.0 Business Understanding

1.1 Background

Acquiring customers in the telecommunications industry is a challenging task due to intense competition, evolving technology, high acquisition costs and customer retention concerns. Telecommunication companies need to stand out by offering specialized services, stay updated with technology and making sure they don't spend too much on customer acquisition. Among the three main revenue-generating strategies (acquiring new customers, upselling to existing customers, and increasing customer retention), increasing retention has proven to be less costly and the most profitable. To achieve this, reducing customer churn, the movement from one provider to another, becomes a critical focus in the highly competitive telecommunications sector. By prioritizing customer retention and addressing churn, telecom companies can maximize profitability and long-term success.Therefore, finding those factors that increase customer churn is important to take necessary actions to reduce this churn. The main goal of our project is to develop an understanding of the cause of customer churn which assists telecom operators to predict customers who are most likely subject to churn, and what to do to retain the most valuable customer

1.2 Problem Statement

SyriaTel Telecommunications has experienced a substantial increase in customer churn rates in American states during the last financial period, resulting in a significant number of customers switching to competitors. Recognizing the urgency of understanding the underlying factors contributing to this trend, the marketing department at SyriaTel has taken proactive measures. They have engaged a consortium of scientists to develop a predictive model capable of identifying customers who are likely to churn, as well as analyzing their behaviors. This initiative aims to address the pressing challenge of customer attrition, which poses a threat to SyriaTel's bottom line and overall revenue growth. By leveraging the insights provided by the predictive model, SyriaTel intends to implement targeted strategies that will enhance customer retention, safeguard their bottom line, and foster new avenues for revenue growth

1.3 Objectives

- To understand which factors or variables contribute the most to customer churn.
- To identify different customer segments based on churn behaviour
- To develop a model that can accurately predict customer churn.
- To obtain valuable insights that help generate the best recommendations to protect Syriatel's revenue.

2.0 Data Understanding

The dataset contains various features related to telecom customer behavior, service usage, and account information. It includes details such as the customer's state, account length, area code, phone number, international plan, voice mail plan, number of voicemail messages, and the total duration and charges for calls made during the day, evening, and night. It also includes information on international calls, customer service calls, and whether or not the customer churned (terminated their contract). The data will be suitable to build a classifier to predict whether a customer will ("soon") stop doing business with SyriaTel telecommunications company.

Data Source : https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset) (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset))

Summary of Features in the Dataset

- **state:** the state the customer lives in - seems like American states
- **account length:** the number of days the customer has had an account
- **area code:** the area code of the customer
- **phone number:** the phone number of the customer
- **international plan:** true if the customer has the international plan, otherwise false
- **voice mail plan:** true if the customer has the voice mail plan, otherwise false
- **number vmail messages:** the number of voicemails the customer has sent
- **total day minutes:** total number of minutes the customer has been in calls during the day
- **total day calls:** total number of calls the user has done during the day
- **total day charge:** total amount of money the customer was charged by the Telecom company for calls during the day
- **total eve minutes:** total number of minutes the customer has been in calls during the evening
- **total eve calls:** total number of calls the customer has done during the evening
- **total eve charge:** total amount of money the customer was charged by the Telecom company for calls during the evening
- **total night minutes:** total number of minutes the customer has been in calls during the night
- **total night calls:** total number of calls the customer has done during the night
- **total night charge:** total amount of money the customer was charged by the Telecom company for calls during the night
- **total intl minutes:** total number of minutes the user has been in international calls
- **total intl calls:** total number of international calls the customer has done
- **total intl charge:** total amount of money the customer was charged by the Telecom company for international calls
- **customer service calls:** number of calls the customer has made to customer service
- **churn:** target variable which is true if the customer terminated their contract, otherwise false

In [1]:
```
#import all the libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, explained_variance_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, precision_recall_curve, recall_
from xgboost import XGBClassifier, plot_importance
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```
#load the dataset and view the first 5 columns

churn_df = pd.read_csv("syriatel_data.csv")
churn_df.head(5)
```

Out[2]:

	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 [3]:
```
# view the last 5 columns to check for any differences
churn_df.tail(5)
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...

5 rows × 21 columns

In [4]:
```
#checking for column features

churn_df.columns
```

Out[4]:
```
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 [5]:
```
#concise summary of the dataset

churn_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
```

In [6]:
```
#shape of the dataset

churn_df.shape
```

Out[6]:
```
(3333, 21)
```

In [7]:
```
# summary statistics for numerical columns

churn_df.describe()
```

Out[7]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total minu
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700

Observations

- Our data consists of 3333 rows and 21 columns
- Data has both continuous and categorical features comprising of the following data types; objects, integers, float and booleans
- Churn which is our target variable is of data type boolean.
- We can also see statistical summary of the numerical records based on their count, median, mean, standard deviation, percentiles, minimum and maximum values.

3.0 Data Preparation

3.1 Data Cleaning

This section prepares the data for EDA and modeling. The dataset will be checked for:

- duplicated rows
- missing values
- In our analysis, we will drop phone numbers as they do not provide any relevant insights.
- Create two variables for our numerical and categorical data types respectively

In [8]:
```
# check for duplicate records

churn_df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
# check missing values
churn_df.isnull().sum()
```

Out[9]:
```
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 [10]:
```
# Dropping phone number because it lacks information about customer behavior.
churn_df = churn_df.drop('phone number', axis=1)

# confirming we have dropped phone number
churn_df.columns
```

Out[10]:
```
Index(['state', 'account length', 'area code', 'international plan',
       'voice mail plan', 'number vmail messages', 'total day minutes',
       'total day calls', 'total day charge', 'total eve minutes',
       'total eve calls', 'total eve charge', 'total night minutes',
       'total night calls', 'total night charge', 'total intl minutes',
       'total intl calls', 'total intl charge', 'customer service calls',
       'churn'],
      dtype='object')
```

In [11]:
```
# Categorical and numerical variables
cat_vars = []
num_vars = []

for col in churn_df.columns:
    if churn_df[col].dtype == 'object':
        cat_vars.append(col)
    else:
        num_vars.append(col)
num_vars.pop(-1)
print("-------------------------------------------------------------------")
print('Categorical variables:', cat_vars)
print("--------------------------------------------------------------------")
print('Numerical variables:', num_vars)
print("--------------------------------------------------------------------")
```

```
------------------------------------------------------------------
Categorical variables: ['state', 'international plan', 'voice mail plan']
-------------------------------------------------------------------
Numerical variables: ['account length', 'area code', 'number vmail message
s', 'total day minutes', 'total day calls', 'total day charge', 'total eve m
inutes', 'total eve calls', 'total eve charge', 'total night minutes', 'tota
l night calls', 'total night charge', 'total intl minutes', 'total intl call
s', 'total intl charge', 'customer service calls']
-------------------------------------------------------------------
```

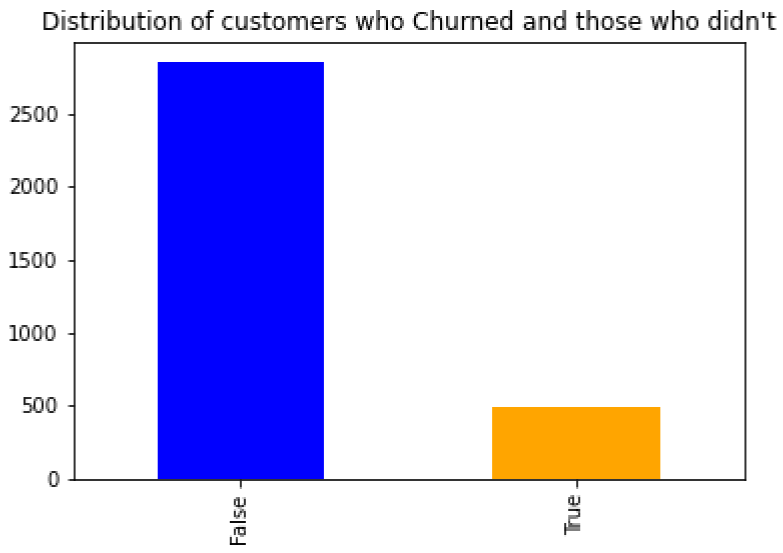
3.2 Univariate Analysis

a. Analysis of the target variable "Churn"

In [12]:
```
#Value_count of target variable

churn_df["churn"].value_counts()
```

Out[12]:
```
False    2850
True      483
Name: churn, dtype: int64
```

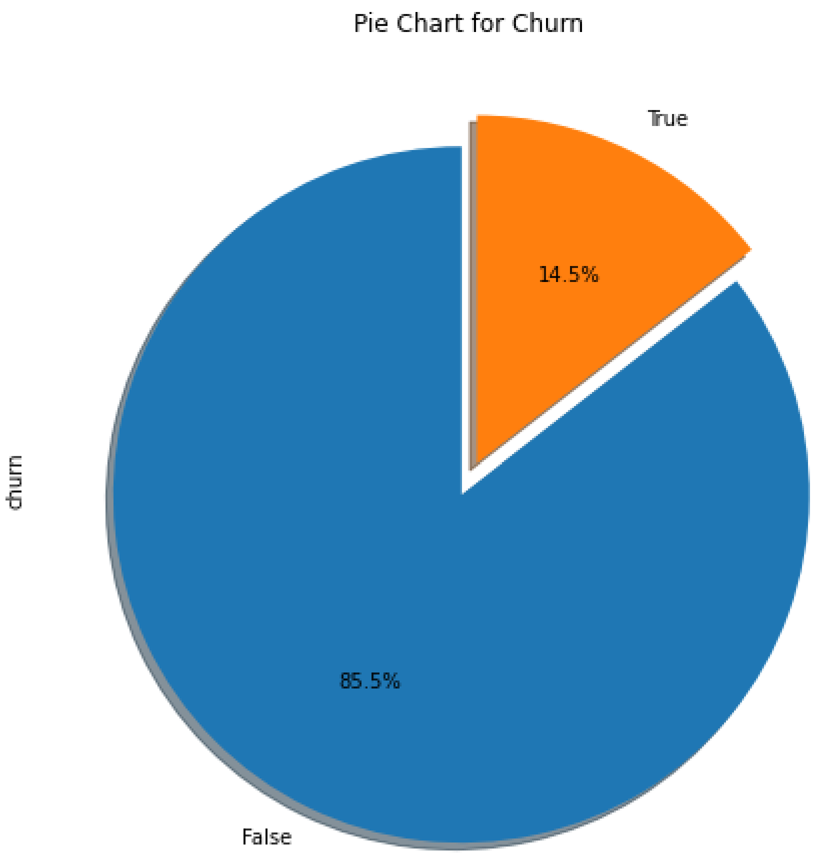
In [13]:
```
#count visualization
churn_df['churn'].value_counts().plot(
    kind='bar',color=['blue', 'orange']).set_title(
    "Distribution of customers who Churned and those who didn't");
```



In [14]:
```
#Check percent of current customers that have churned (True) and those that d

churn_df["churn"].value_counts(normalize=True) * 100
```

Out[14]:
```
False    85.508551
True     14.491449
Name: churn, dtype: float64
```

In [15]:
```
#To get the pie chart to analyze churn
churn_df ['churn'].value_counts().plot.pie(explode=[0.05,0.05], autopct='%1.1
plt.title('Pie Chart for Churn')
plt.show()
```



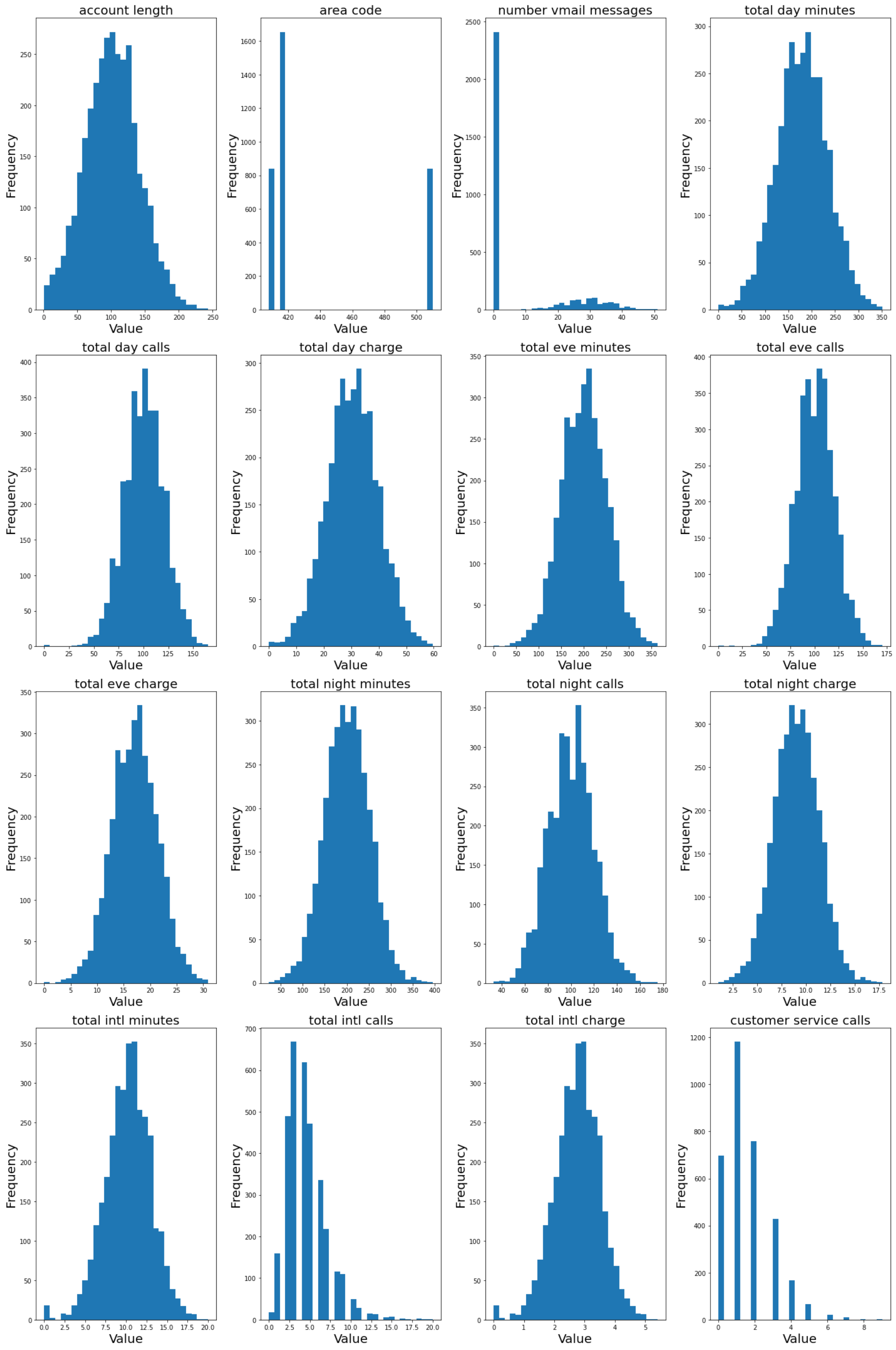
The above visualization shows customers with active contracts and those that have terminated.

When we check our target variable "churn" it indicates that the majority, which is 85% of the customers in the churn_df dataset are active and the rest, which is about 14.5% are inactive.

This means that the dataset primarily consists of active customers, with a relatively smaller portion of inactive customers presenting a case of class imbalance which will require appropriate strategies to handle before modeling as an imbalanced class can cause the model to make false predictions.

b. Univariate analysis for Numerical Variables

In [16]:
```
# Create subplots for each numerical variable
num_plots = len(num_vars)
num_rows = 4
num_cols = 4
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 30))

for i, var in enumerate(num_vars):
    row = i // num_cols
    col = i % num_cols
    axes[row, col].hist(churn_df[var], bins=30)
    axes[row, col].set_title(var,fontsize=20)
    axes[row, col].set_xlabel('Value',fontsize=20)
    axes[row, col].set_ylabel('Frequency',fontsize=20)

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

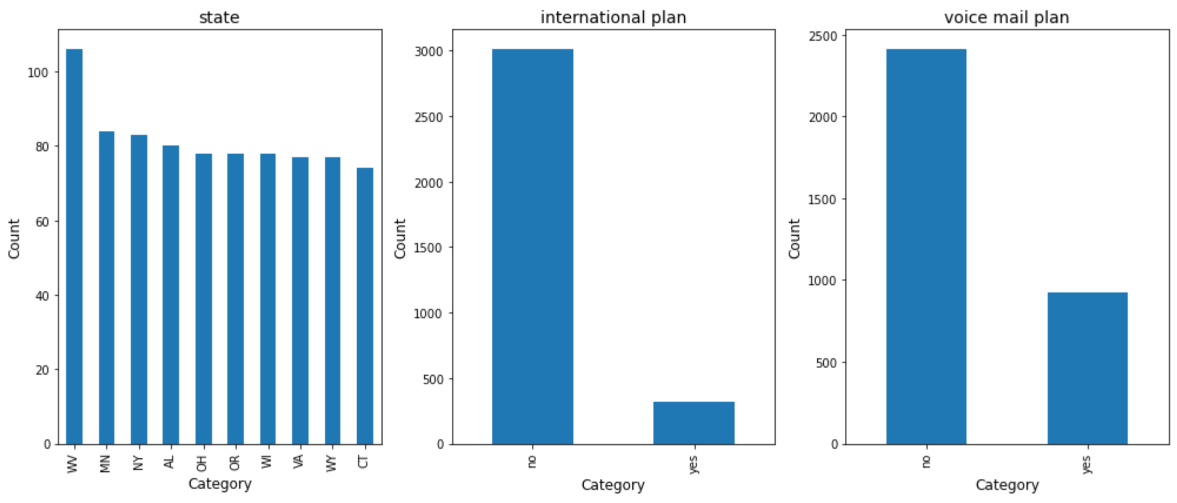


- The majority of the features in the data exhibit a normal distribution. This characteristic implies that the data points within these features tend to cluster around the mean, with relatively fewer occurrences of extreme values.

- Majority of customers in the dataset have made one customer service call.
- The highest number of calls made to customer service is 9 calls.
- The total international calls and customer service calls are skewed to the right

c. Univariate analysis for Categorical Variables

In [17]:
```
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(14, 6))

for i, cat_var in enumerate(cat_vars):
    top_ten_cats = churn_df[cat_var].value_counts().nlargest(10)
    top_ten_cats.plot(kind='bar', ax=ax[i])
    ax[i].set_title(cat_var, fontsize=14)
    ax[i].set_xlabel('Category', fontsize=12)
    ax[i].set_ylabel('Count', fontsize=12)

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



- The top five American states that syriatel operates in are West Virginia, Minnesota, New York, Alabama and Oregon respectively.
- The majority of customers in the dataset do not have an international plan or a voice mail plan

3.3 Bivariate Analysis

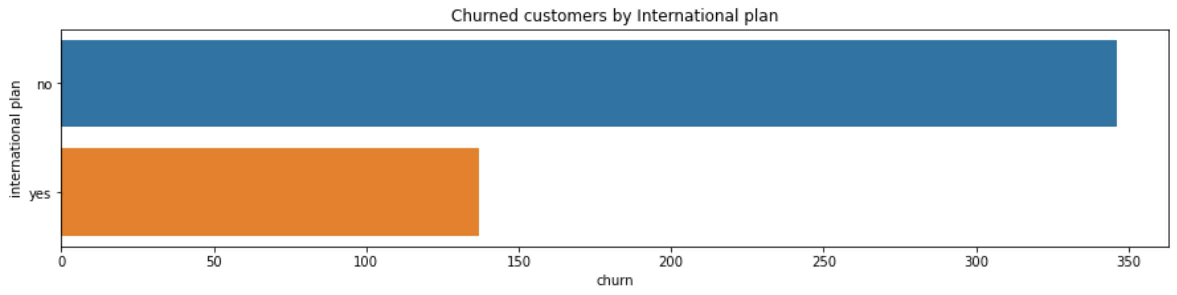
a. Analysis of churned Customers based on International Plan

In [18]:
```
#Churned customers by international_plan

churn_international_plan = churn_df.groupby("international plan")["churn"].su
churn_international_plan
```

Out[18]:

	international plan	churn
0	no	346
1	yes	137

In [19]:
```
# Lets visualize customers who have terminated their contracts based on inter

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 3))
sns.barplot(x = "churn", y = "international plan", data = churn_international
axes.set_title("Churned customers by International plan");
```



- Out of the 483 customers who terminated their contracts 346 had no international plan and 137 had international plan

In [20]:
```
#Calculate the International Plan vs Churn percentage

International_plan_data = pd.crosstab(churn_df["international plan"],churn_df
International_plan_data['Percentage Churn'] = International_plan_data.apply(l
print(International_plan_data)
```

```
churn               False  True  Percentage Churn
international plan
no                   2664   346         11.495017
yes                   186   137         42.414861
```

The above comparative analysis shows that:

- Out of the 3010 customers who do not have an international plan, 11.4% of customers have churned.
- Out of the 323 customers who have an international plan, 42.4% of them have terminated their accounts.

b. Analysis of churned customers based on Area Code

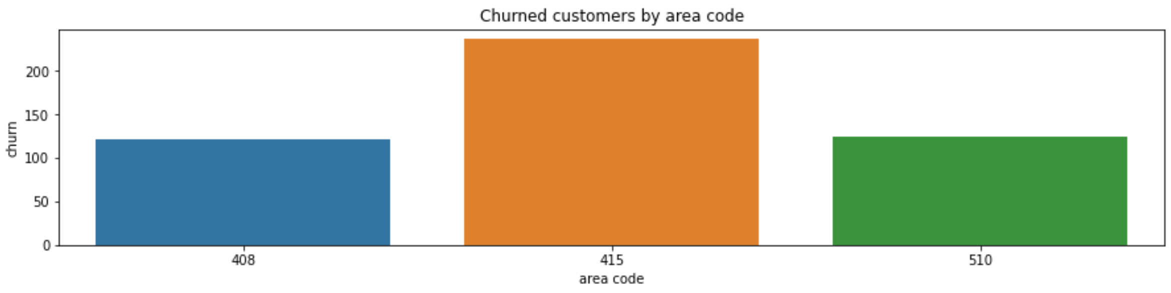
In [21]:
```
# We shall look at the distribution of inactive customers based on their area

churn_area_code = churn_df.groupby("area code")["churn"].sum().reset_index()
churn_area_code
```

Out[21]:

	area code	churn
0	408	122
1	415	236
2	510	125

In [22]:
```
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 3))
sns.barplot(x = "area code", y = "churn", data = churn_area_code, ax=axes)
axes.set_title("Churned customers by area code")
```

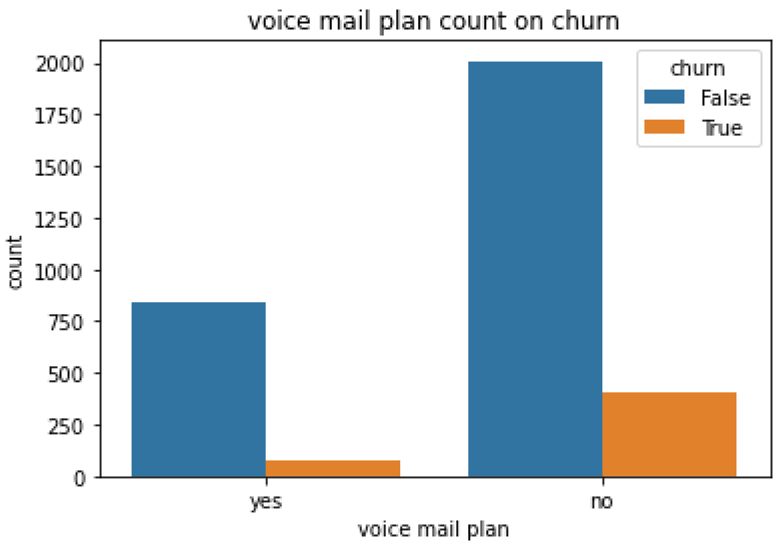
Out[22]:
```
Text(0.5, 1.0, 'Churned customers by area code')
```



- The area code 415 had the most customers who terminated their contract while 408 area code had the least

c. Analysis of churn based on Voice Mail Plan

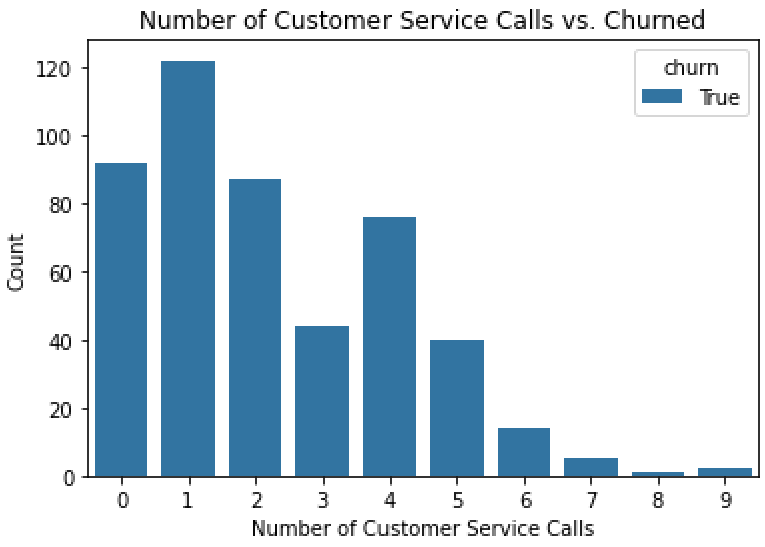
In [23]:
```
sns.countplot(x='voice mail plan',
              data=churn_df, hue='churn').set(title='voice mail plan count on
```



- Majority of the customers that have terminated their contract do not have voicemail plan. It could indicate that the voicemail plan might not be a highly desired or valued service among customers.

d. Analysis of churned based on Customer Service Calls

```
In [24]: # Create the countplot
         sns.countplot(x='customer service calls', hue='churn', data=churn_df[churn_df

         # Set the title and labels
         plt.title("Number of Customer Service Calls vs. Churned")
         plt.xlabel("Number of Customer Service Calls")
         plt.ylabel("Count")

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



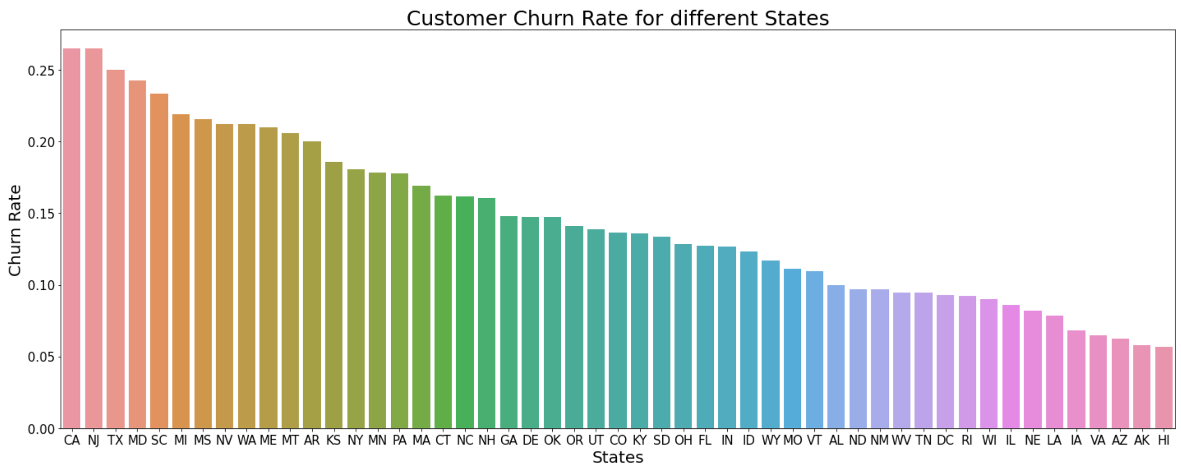
- The above visualization shows that the majority of churned customers made 1 call to customer service. This could indicates that a significant number of customers who decided to leave the service had limited engagement with customer service, possibly suggesting that their issues or concerns were not adequately addressed.

e. Analysis of churn rates based on the different states

In [25]:
```
# Does different states have different churn rates?
churn_rate_state = pd.DataFrame(churn_df.groupby(["state"])['churn'].mean().s
print(churn_rate_state)
```

```
          churn
state          
CA     0.264706
NJ     0.264706
TX     0.250000
MD     0.242857
SC     0.233333
MI     0.219178
MS     0.215385
NV     0.212121
WA     0.212121
ME     0.209677
MT     0.205882
AR     0.200000
KS     0.185714
NY     0.180723
MN     0.178571
PA     0.177778
MA     0.169231
CT     0.162162
NC     0.161765
NH     0.160714
GA     0.148148
DE     0.147541
OK     0.147541
OR     0.141026
UT     0.138889
CO     0.136364
KY     0.135593
SD     0.133333
OH     0.128205
FL     0.126984
IN     0.126761
ID     0.123288
WY     0.116883
MO     0.111111
VT     0.109589
AL     0.100000
ND     0.096774
NM     0.096774
WV     0.094340
TN     0.094340
DC     0.092593
RI     0.092308
WI     0.089744
IL     0.086207
NE     0.081967
LA     0.078431
IA     0.068182
VA     0.064935
AZ     0.062500
AK     0.057692
HI     0.056604
```

In [26]:
```python
# visualization of the churn rates for states

fig, ax = plt.subplots(figsize=(20,8))
sns.barplot(x = np.linspace(0, len(churn_rate_state)-1, len(churn_rate_state)
            y = 'churn', data = churn_rate_state , ax = ax)
plt.title('Customer Churn Rate for different States', fontsize = 25)
ax.tick_params(axis = 'both', labelsize = 15)
plt.xlabel('States', fontsize = 20)
plt.ylabel('Churn Rate', fontsize = 20)
ax.set_xticklabels(churn_rate_state.index)
plt.tight_layout()
```

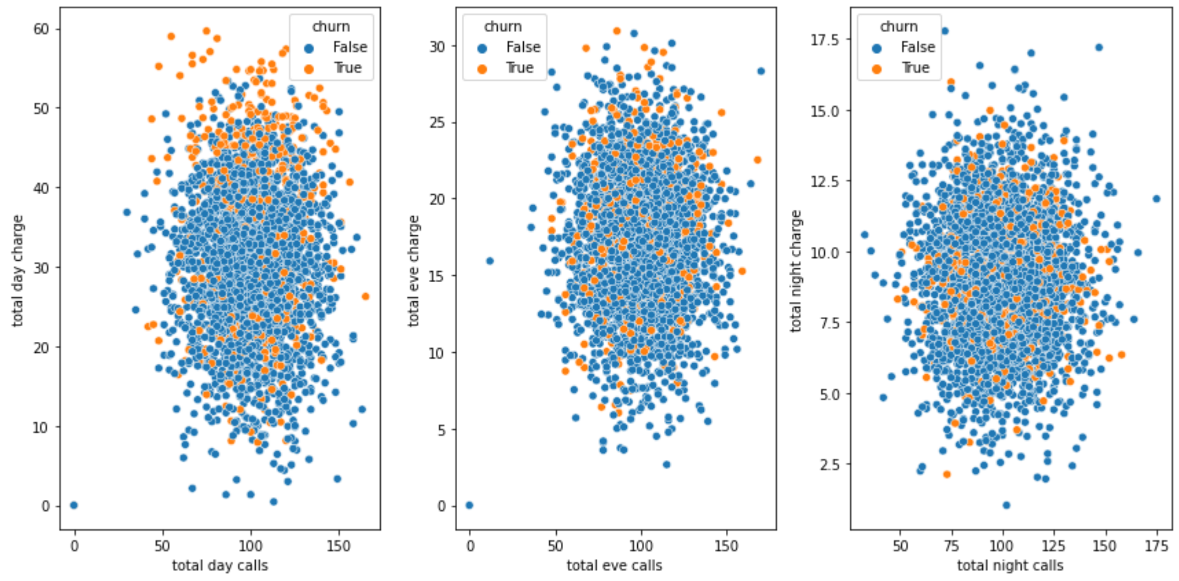


The vizualization aboves shows that different states have different churn rates. California and New Jersey are the two highest churn rate states greater than 25%, while Alaska and Hawaii are the two lowest churn rate states with less than 6%.

3.4 Multivariate Analysis

a. Churn analysis - total calls vs. total charges by time period

In [27]:
```
# lets visualize the performance of calls

features = [
    ('total day calls', 'total day charge'),
    ('total eve calls', 'total eve charge'),
    ('total night calls', 'total night charge')
]

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 6))

for i, (x, y) in enumerate(features):
    ax = axes[i]  # Access the corresponding axis from the 1x3 grid
    sns.scatterplot(x=x, y=y, data=churn_df, hue='churn', ax=ax)
    ax.set_xlabel(x)
    ax.set_ylabel(y)

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

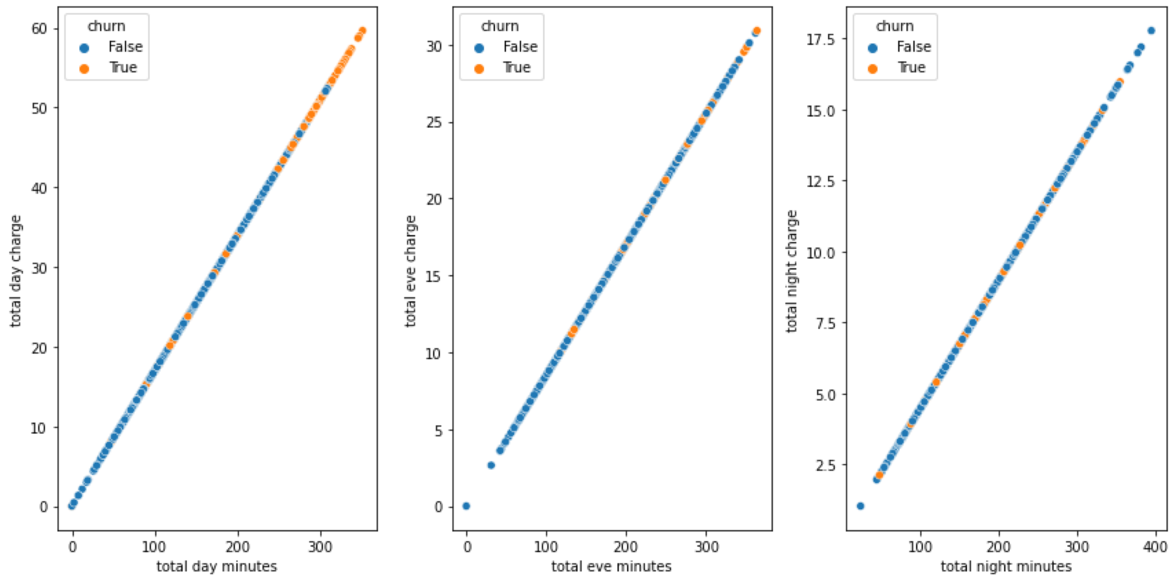


Based on the visualization above, we can draw the following observations:

- Among all the time periods, daytime calls are significantly charged higher compared to evening and nighttime calls.
- The above observations may indicate that daytime is considered a peak hour leading to higher charges
- The call charges for daytime, evening, and nighttime are higher even with fewer calls made. This may indicate that calls are also charged on duration and not necessarily the number of calls.

b. Churn analysis - total minutes vs. total charges by time period

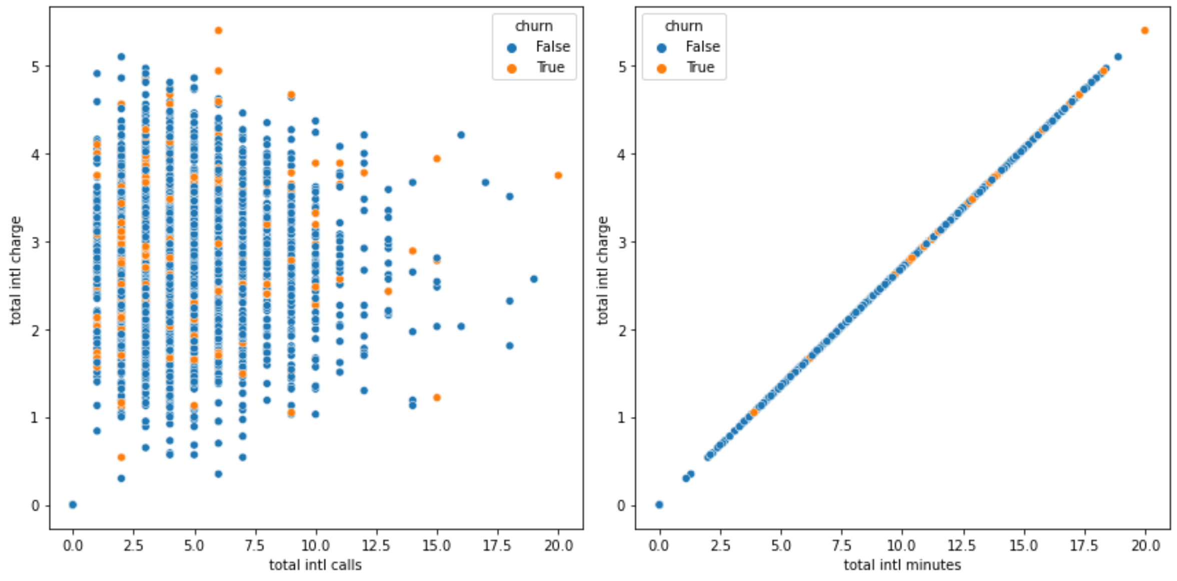
In [28]:
```
# lets visualize minutes performance

features = [
    ('total day minutes', 'total day charge'),
    ('total eve minutes', 'total eve charge'),
    ('total night minutes', 'total night charge')
]

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 6))

for i, (x, y) in enumerate(features):
    ax = axes[i]  # Access the corresponding axis from the 1x3 grid
    sns.scatterplot(x=x, y=y, data=churn_df, hue='churn', ax=ax)
    ax.set_xlabel(x)
    ax.set_ylabel(y)

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



- Among all the time periods, daytime minutes are significantly charged higher compared to evening and nighttime minutes.
- The above observations may indicate that daytime is considered a peak hour leading to higher charges.
- There is a linear relationship between the total minutes of daytime, evening, nighttime and the corresponding total charges. This indicates that the higher the subscription minutes the higher the charges.
- On average, customers who have terminated their accounts appear to have subscribed to more day minutes, leading to higher charges.

c. Churn analysis - total international calls and minutes vs. total international charges

In [29]:
```
# Lets visualize performance of international services

features = [
    ('total intl calls', 'total intl charge'),
    ('total intl minutes', 'total intl charge')
]

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))  # 1 row, 2 colum

for i, (x, y) in enumerate(features):
    ax = axes[i]  # Access the corresponding axis from the 1x2 grid
    sns.scatterplot(x=x, y=y, data=churn_df, hue='churn', ax=ax)
    ax.set_xlabel(x)
    ax.set_ylabel(y)

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

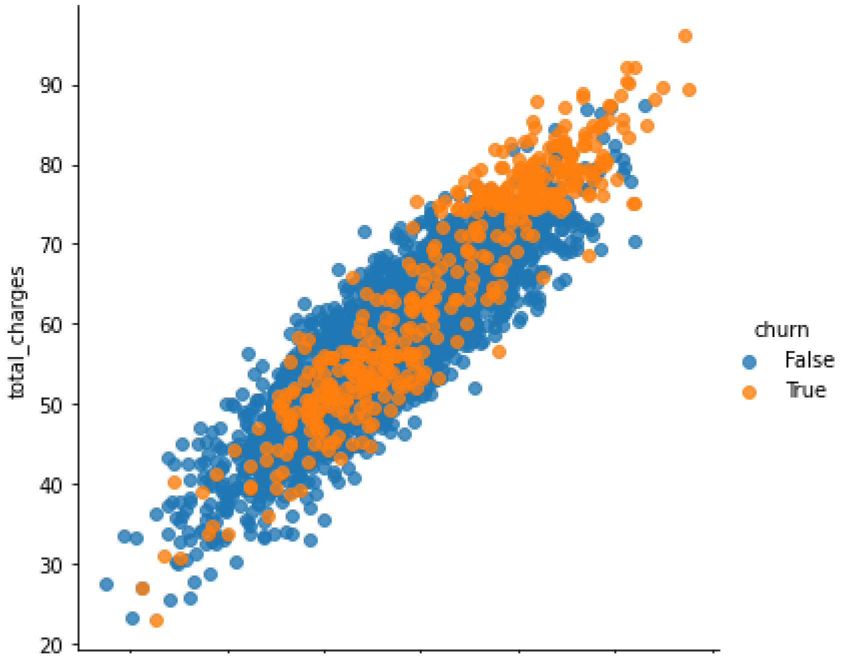


- There is a linear relationship between the total international minutes and the corresponding total charges. This indicates that the higher the subscription minutes the higher the charges.
- The call charges seem to be higher even with fewer calls made.This may indicate that international calls may also be charged on duration and not necessarily the number of calls.

d. Churn analysis - total minutes and total charges

In [30]:
```
#total minutes during the period
churn_df['total_calls'] = (churn_df['total day calls'] + churn_df['total eve 
# total charge during the period
churn_df['total_charges']= (churn_df['total day charge'] + churn_df['total ev
                 churn_df['total night charge'] + churn_df['total intl charge
#total minutes during the period
churn_df['total_minutes']= (churn_df['total day minutes'] +churn_df['total ev
             churn_df['total night minutes'])
```

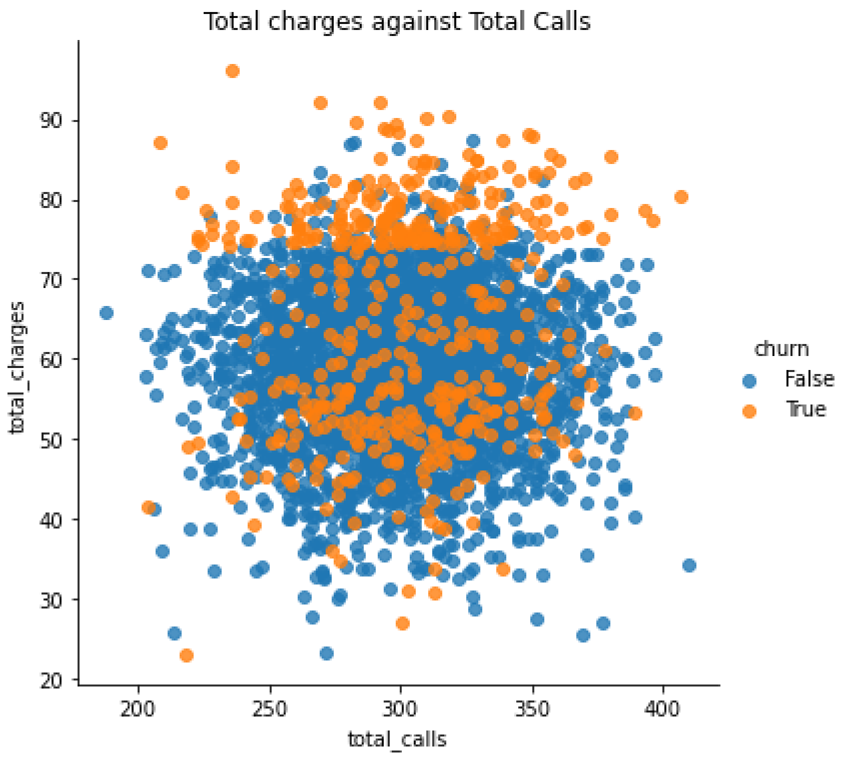
In [31]:
```
# visualization of churn performance for total minutes and charges

# Plot the lmplot
sns.lmplot(x='total_minutes', y='total_charges', data=churn_df, hue='churn', 
```



- Total minutes have a linear relationship with the total charge, indicating that as the number of minutes a customer subscribes to increases, the charge also increases.
- We can also observe that customers who have terminated their accounts tend to subscribe to higher minutes, resulting in a higher charge.

e. Churn analysis - total calls and total charges

In [32]:
```
#visualizing this performance

sns.lmplot(x='total_calls',
           y='total_charges',
           data=churn_df, hue='churn', fit_reg=False).set(title='Total charge
```

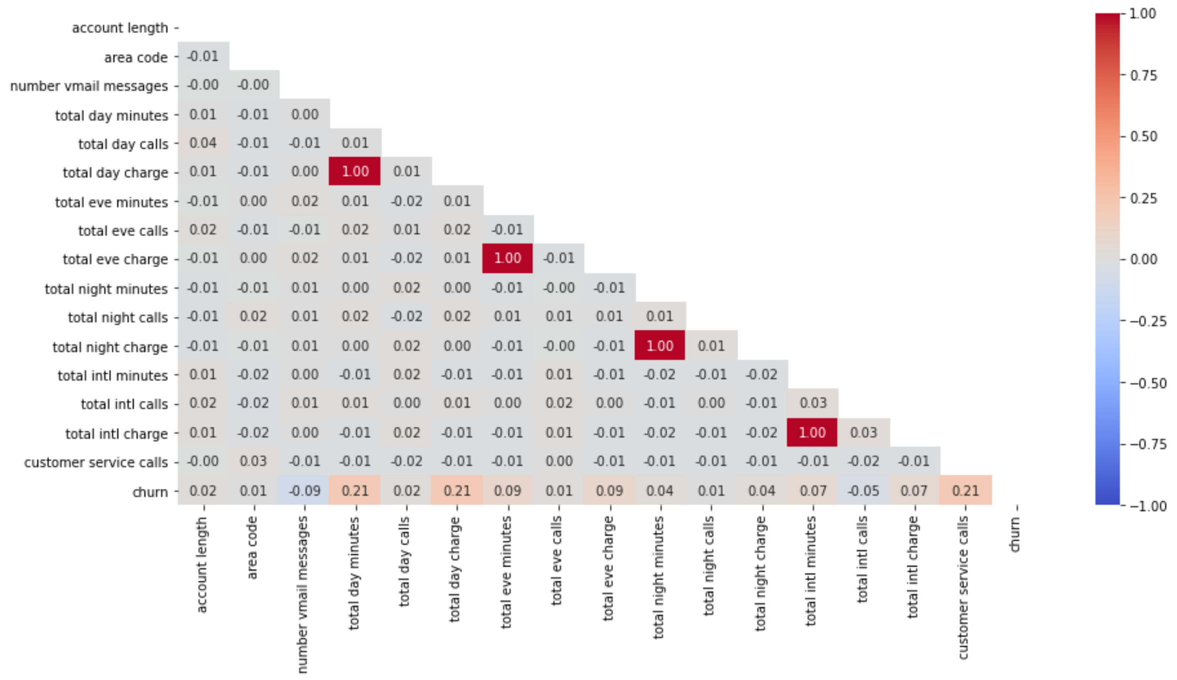


It is quite surprising that customers with a lower total number of calls tend to have higher charges, and a significant number of these high charges are associated with customers who have terminated their accounts.

In [33]:
```
#drop the comparsion columns as they will not be included in our model
churn_df = churn_df.drop(columns = ['total_calls','total_charges','total_minu
```

f. Visualization of Correlation Heatmap

In [34]:
```
# Calculate correlation matrix
corr_matrix = churn_df.corr()

# Generate a mask to hide the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

plt.figure(figsize=(15, 7))
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="coolwa
plt.show()
```



In [35]:
```
#checking the correlation between target variable and other features

churn_df.corr()['churn'].sort_values(ascending=False)
```

Out[35]:
```
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
account length            0.016541
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 correlation heatmap, we can see high multicollinearity of total day charge & total day minute, total evening charge & total evening minute, total night charge & total night minute with a value of 1.
- Customer service call is positively correlated with only area code among the features and negatively correlated with rest of the variables.
- We can also see that from numerical values, the top 5 highly correlated features with churn are customer service calls, total day minutes and charge, total eve minutes and charge, total international minutes and charge and total night minutes and charge.

3.5 Preprocessing

In [36]:
```
#Get a copy of the churn dataset and view
churn_df_copy = churn_df.copy()
churn_df_copy
```

Out[36]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3
...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	no	yes	36	156.2	77	26.55	215.5
3329	WV	68	415	no	no	0	231.1	57	39.29	153.4
3330	RI	28	510	no	no	0	180.8	109	30.74	288.8
3331	CT	184	510	yes	no	0	213.8	105	36.35	159.6
3332	TN	74	415	no	yes	25	234.4	113	39.85	265.9

3333 rows × 20 columns

In [37]:
```
# Converting churn column from boolean to integer
churn_df_copy['churn'] = churn_df_copy['churn'].astype(int)
```

In [38]:
```
# Dropping states column as it will not impact our modelling part

churn_df_copy = churn_df_copy.drop('state', axis=1)
```

In [39]:
```
#creating dummy variables
churn_df_copy= pd.get_dummies(churn_df_copy, drop_first=True)
churn_df_copy.head()
```

Out[39]:

	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	c
0	128	415	25	265.1	110	45.07	197.4	99	16.78	244.7	91	
1	107	415	26	161.6	123	27.47	195.5	103	16.62	254.4	103	
2	137	415	0	243.4	114	41.38	121.2	110	10.30	162.6	104	
3	84	408	0	299.4	71	50.90	61.9	88	5.26	196.9	89	
4	75	415	0	166.7	113	28.34	148.3	122	12.61	186.9	121	

a. Defining the predictor and target variables

In [40]:
```
# define our X and y variables
X = churn_df_copy.drop (columns = ['churn'], axis=1)
y = churn_df_copy['churn']
```

In [41]:
```
#for consistency of results set a random seed
np.random.seed(123)

# Performing a train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran
```

In [42]:
```
#scale the data
#initialize the scaler
scaler = StandardScaler()

#fit the data on the scaler
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

b. Fixing the class imbalance

In [43]:
```
# Previous original class distribution
print(y_train.value_counts())
```

```
0    2141
1     358
Name: churn, dtype: int64
```

In [44]:
```
# Use Smote to resample and fix the class imbalance problem
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_t
```

We used SMOTE class in order to improve the model's performance on the minority class.

In [45]:
```
# Preview synthetic sample class distribution

print(pd.Series(y_train_resampled).value_counts())
```

```
1    2141
0    2141
Name: churn, dtype: int64
```

The imbalance on the target variable is now resolved.

4.0 Modeling

We will now build a model that can predict the customer churn based on the features in our dataset using the following algorithms:

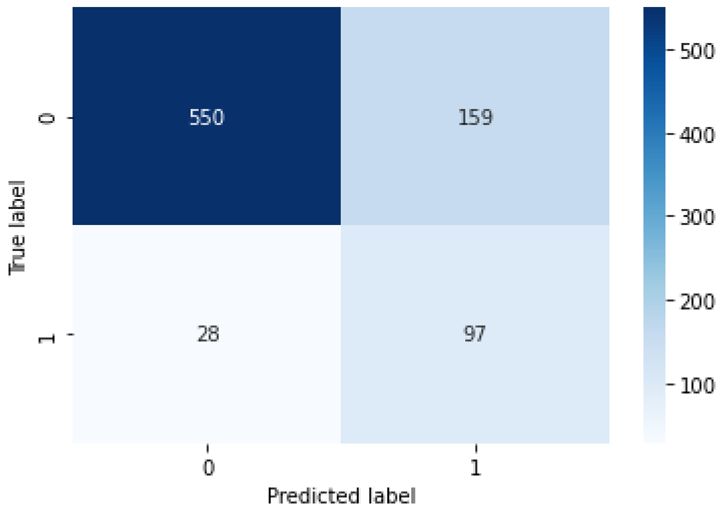
- Logistic Regression
- Decision Tree
- Random Forest
- XG Boost

Model 1 : Logistics Regression Classifier

In [46]:
```
# Instantiate the model
logreg = LogisticRegression(random_state =42)

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

#predicting on the test
y_pred_log = logreg.predict(X_test_scaled)
```

In [47]:
```
def plot_confusion_matrix(y_true, y_pred, classes):
    """
    Plots a confusion matrix.
    """
    cm = confusion_matrix(y_true, y_pred)
    plt.figure()
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, y
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()
```

In [48]:
```
# visualizing confusion matrix
plot_confusion_matrix(y_test, y_pred_log, [0,1])
```



In [49]:
```
# displaying scores
print(classification_report(y_test,y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.95      0.78      0.85       709
           1       0.38      0.78      0.51       125

    accuracy                           0.78       834
   macro avg       0.67      0.78      0.68       834
weighted avg       0.87      0.78      0.80       834
```

Logistics Regression observations

Recall measures the ability of the model to correctly identify customers who are likely to churn (positive instances) out of all the customers who actually churned.

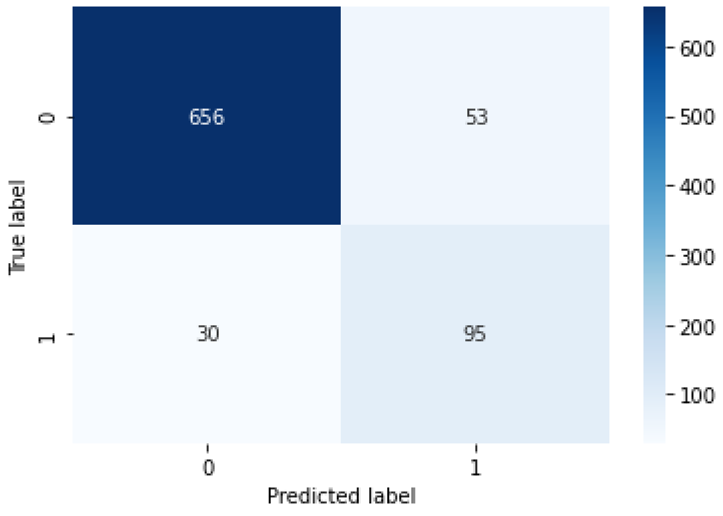
- For class 0, which represents customers who did not churn, the recall is 0.78. This means that the model correctly identified 78% of the customers who did not churn out of the total number of customers who actually did not churn.

- Similarly, for class 1, which represents customers who churned, the recall is 0.78, indicating that the model correctly identified 78% of the customers who churned out of the total number of customers who actually churned.

Model 2: Decision Tree Classifier

In [50]:
```
# Instantiate a DT classifier
clf = DecisionTreeClassifier(random_state=42)

# fit DT classifier
clf.fit(X_train_resampled, y_train_resampled)

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

In [51]:
```
# plotting a confusin matrix
plot_confusion_matrix(y_test, y_pred_clf, [0,1])
```



In [52]:
```
print(classification_report(y_test,y_pred_clf))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       709
           1       0.64      0.76      0.70       125

    accuracy                           0.90       834
   macro avg       0.80      0.84      0.82       834
weighted avg       0.91      0.90      0.90       834
```

Decision Tree Classifier Observations

Recall

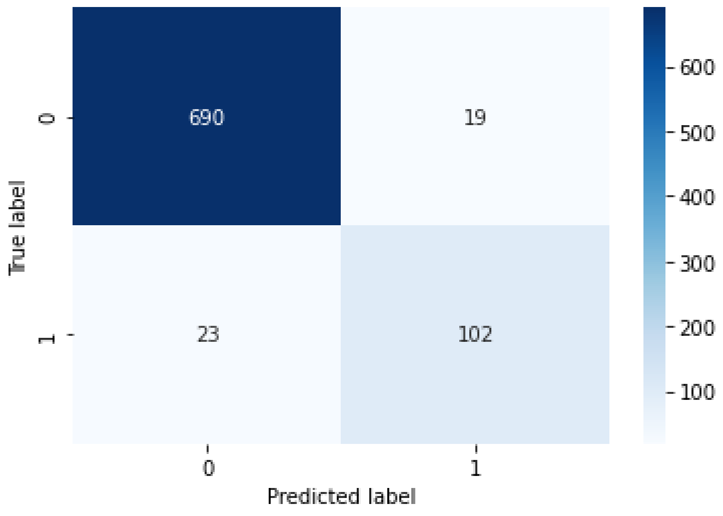
- For class 0, which represents customers who did not churn, the recall is 0.93. This means that the model correctly identified 93% of the customers who did not churn out of the total number of customers who actually did not churn.
- Similarly, for class 1, which represents customers who churned, the recall is 0.76, indicating that the model correctly identified 76% of the customers who churned out of the total number of customers who actually churned.

Accuracy: 0.90 means that 90% of the total number of customers was correctly classified. The

Model 3: Random Forest Classifier

In [53]:
```
# Instanstiate a DT classifier
rfc = RandomForestClassifier(random_state=42)

# fit RFCclassifier
rfc.fit(X_train_resampled, y_train_resampled)

# Make predictions for test data
y_pred_rfc = rfc.predict(X_test_scaled)
```

In [54]:
```
plot_confusion_matrix(y_test, y_pred_rfc, [0,1])
```



In [55]:
```
print(classification_report(y_test,y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       709
           1       0.84      0.82      0.83       125

    accuracy                           0.95       834
   macro avg       0.91      0.89      0.90       834
weighted avg       0.95      0.95      0.95       834
```

Random Forest Classifier Observations

Recall

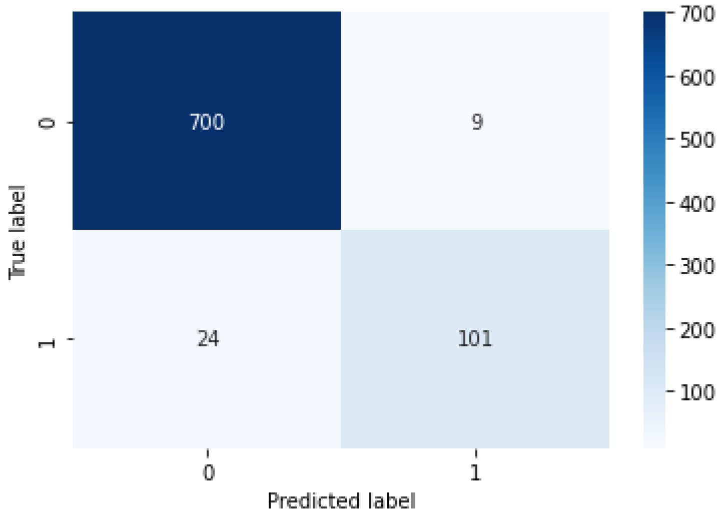
- For class 0, which represents customers who did not churn, the recall is 0.97. This means that the model correctly identified 97% of the customers who did not churn out of the total number of customers who actually did not churn.
- Similarly, for class 1, which represents customers who churned, the recall is 0.82, indicating that the model correctly identified 82% of the customers who churned out of the total number of customers who actually churned.

Accuracy: 0.95 means that 95% of the total number of customers was correctly classified. The model performs better than Decision Tree Classifier model.

Model 4:XGBoost

```
In [56]: # Instanstiate the model
x_gb = XGBClassifier(random_state=42)

# fit XGB classifier
x_gb.fit(X_train_resampled, y_train_resampled)

# Make predictions for test data
y_pred_xgb = x_gb.predict(X_test_scaled)
```

```
In [57]: plot_confusion_matrix(y_test, y_pred_xgb, [0,1])
```



```
In [58]: print(classification_report(y_test,y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       709
           1       0.92      0.81      0.86       125

    accuracy                           0.96       834
   macro avg       0.94      0.90      0.92       834
weighted avg       0.96      0.96      0.96       834
```

XGBoost Classifier Observations

Recall

- For class 0, which represents customers who did not churn, the recall is 0.99. This means that the model correctly identified 99% of the customers who did not churn out of the total number of customers who actually did not churn.
- Similarly, for class 1, which represents customers who churned, the recall is 0.81, indicating that the model correctly identified 81% of the customers who churned out of the total number of customers who actually churned.

Accuracy: 0.96 means that 96% of the total number of customers was correctly classified. The model seems to perform as well as the Random Forest Classifier model.

5.0 Model Evaluation

5.1 Model comparison

In [59]:
```
classifiers = [LogisticRegression(),
               RandomForestClassifier(),
               DecisionTreeClassifier(),
               XGBClassifier()]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'accuracy', 'recall'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train_resampled, y_train_resampled)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)  # Calculate precision score
    result_table = result_table.append({'classifiers': cls.__class__.__name__
                                        'accuracy': accuracy, 'recall': recal

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

result_table
```

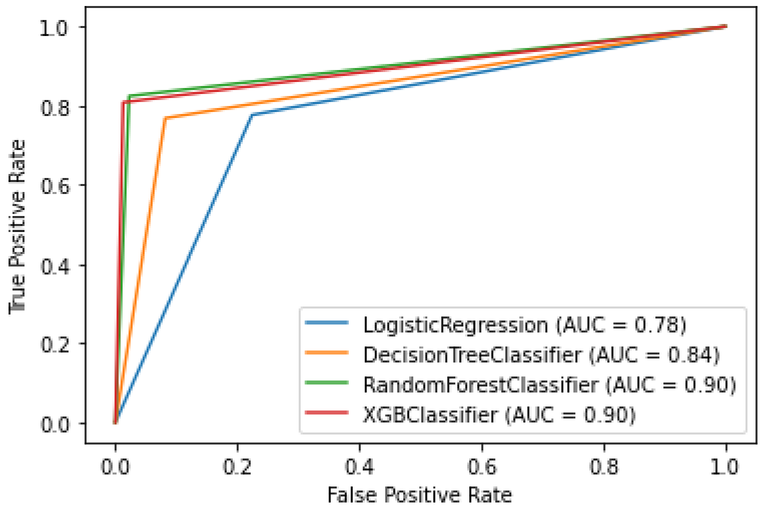
Out[59]:

	accuracy	recall
classifiers		
LogisticRegression	0.775779	0.776
RandomForestClassifier	0.954436	0.832
DecisionTreeClassifier	0.894484	0.744
XGBClassifier	0.960432	0.808

- All the models are able to predict well, however, Random Forest Classifier and XGBoost Classier have the highest accuracy and recall scores.
- We shall proceed and tune Random Forest Classifier and XGBost classifier hyperparameters and compare the results.

ROC

In [60]:
```
# Get the ROC curves for all classifiers
classifiers = ["LogisticRegression", "DecisionTreeClassifier", "RandomForestC
roc_curves = []

for classifier_name in classifiers:
    if classifier_name == "LogisticRegression":
        classifier = LogisticRegression()
    elif classifier_name == "DecisionTreeClassifier":
        classifier = DecisionTreeClassifier()
    elif classifier_name == "RandomForestClassifier":
        classifier = RandomForestClassifier()
    elif classifier_name == "XGBClassifier":
        classifier = XGBClassifier()

    classifier.fit(X_train_resampled, y_train_resampled)
    y_pred = classifier.predict(X_test_scaled)
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    roc_curves.append((fpr, tpr, roc_auc, classifier_name))

# Plot the ROC curves and print AUC values
plt.figure()
for fpr, tpr, roc_auc, classifier_name in roc_curves:
    plt.plot(fpr, tpr, label=f'{classifier_name} (AUC = {roc_auc:.2f})')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



- XGB Classifier and RandomForestClassifier are producing better results in model 4 and model 3 respectively.
- The AUC value for model 3:RandomForest is 0.90 and Model 4: XGBoost is 0.90
- Lets perform hyperparameter tuning to improve them.

5.2 Hyperparameter tuning for our best models

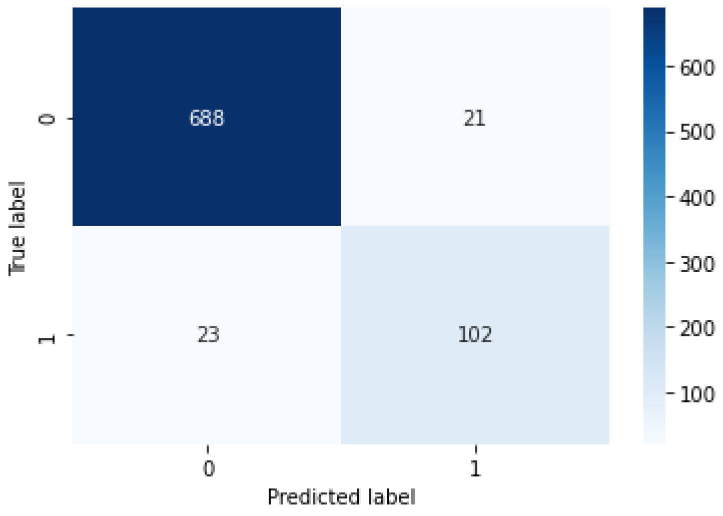
1. Tuned RandomForestClassifier`

In [61]:
```
# Create a parameter grid with reduced values
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

# Create a grid search object
rfc = RandomForestClassifier()
grid_search = GridSearchCV(rfc, param_grid, cv=3, scoring='accuracy', n_jobs=

# Fit the grid search object
grid_search.fit(X_train_resampled, y_train_resampled)

# Print the best parameters
print(grid_search.best_params_)
```

```
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimato
rs': 100}
```

In [62]:
```
# Instantiate a  RandomForest classifier
rfc_tune = RandomForestClassifier(max_depth=10,
                                  min_samples_leaf=1,
                                  min_samples_split=2,
                                  n_estimators=200,
                                  random_state=42)

# fit RFCclassifier
rfc_tune.fit(X_train_resampled, y_train_resampled)

# Make predictions for test data
y_pred_rfc_tune = rfc_tune.predict(X_test_scaled)
```

```
In [63]: plot_confusion_matrix(y_test, y_pred_rfc_tune, [0,1])
```



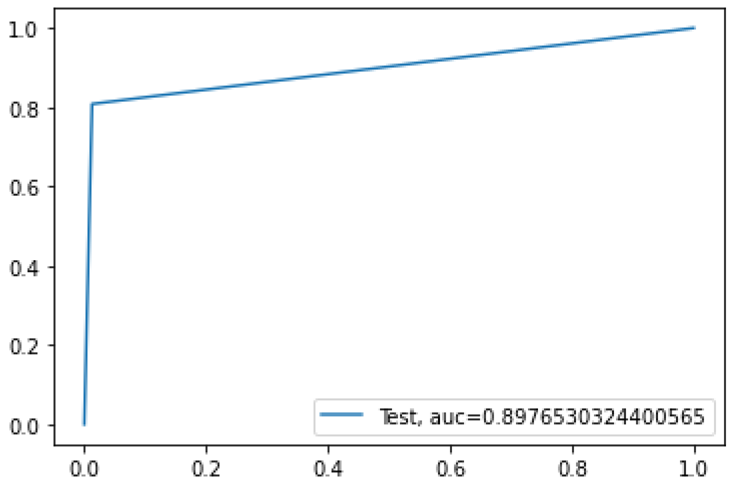
```
In [64]: print(classification_report(y_test,y_pred_rfc_tune))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       709
           1       0.83      0.82      0.82       125

    accuracy                           0.95       834
   macro avg       0.90      0.89      0.90       834
weighted avg       0.95      0.95      0.95       834
```

ROC curve

In [65]:
```
# Make predictions
y_prob = x_gb.predict(X_test_scaled)
y_pred = (y_prob > 0.5).astype(int)



# Calculate evaluation metrics
roc_value = roc_auc_score(y_test, y_prob)
print("RNN roc_value: {0}" .format(roc_value))
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
threshold = thresholds[np.argmax(tpr-fpr)]



roc_auc = auc(fpr, tpr)
print("ROC for the test dataset",'{:.1%}'.format(roc_auc))
plt.plot(fpr, tpr, label="Test, auc="+str(roc_auc))
plt.legend(loc=4)
plt.show()
```

```
RNN roc_value: 0.8976530324400565
ROC for the test dataset 89.8%
```



Checking for Overfiting

In [66]:
```
# Make predictions for test data
y_train_pred_rfc = rfc_tune.predict(X_train_resampled)
y_test_pred_rfc = rfc_tune.predict(X_test_scaled)


# Calculate accuracy on the training and test data
train_accuracy = accuracy_score(y_train_resampled, y_train_pred_rfc)
test_accuracy = accuracy_score(y_test, y_test_pred_rfc)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

```
Train Accuracy: 0.9673049976646427
Test Accuracy: 0.947242206235012
```

Tuned Random Forest Classifier Observations

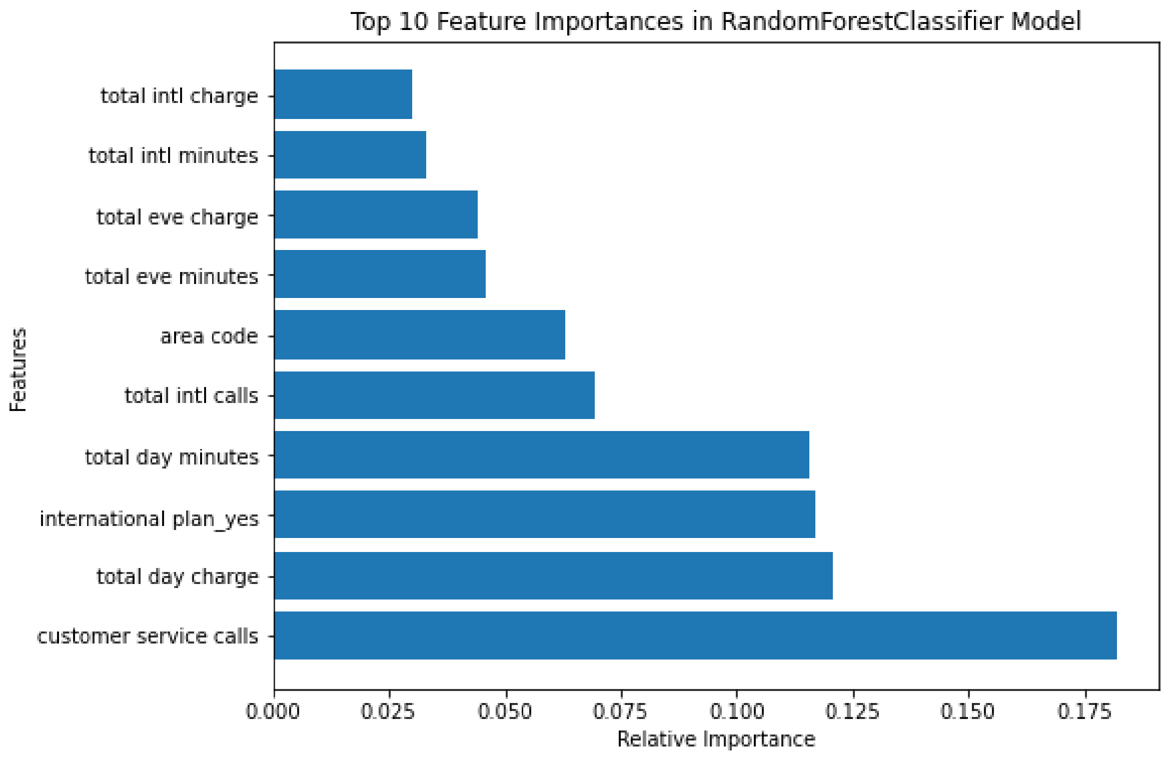
Recall

- For class 0, which represents customers who did not churn, the recall is 0.97. This means that the model correctly identified 97% of the customers who did not churn out of the total number of customers who actually did not churn.
- Similarly, for class 1, which represents customers who churned, the recall is 0.82, indicating that the model correctly identified 82% of the customers who churned out of the total number of customers who actually churned.

Accuracy: 0.95 means that 95% of the total number of customers was correctly classified. The model performs better than Decision Tree Classifier model.

Important Features for RandomForest Model

In [67]:
```
# Assuming 'churn' is the target column, and you want to remove it from churn_
# You can create a new DataFrame without the 'churn' column
churn_df_copy_without_churn = churn_df_copy.drop('churn', axis=1)

# Get the feature importances from the XGBoost model
importances = rfc_tune.feature_importances_

# Get the indices to sort the features in descending order of importance
indices = np.argsort(importances)[::-1]

# Get the feature names and importances for the top 10 features
top_n = 10
top_feature_names = churn_df_copy_without_churn.columns[indices[:top_n]]
top_importances = importances[indices][:top_n]

# Plot the top 10 feature importances as a horizontal bar plot
plt.figure(figsize=(8, 6))
plt.barh(range(top_n), top_importances, align='center')
plt.yticks(range(top_n), top_feature_names)
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.title('Top 10 Feature Importances in RandomForestClassifier Model')
plt.show()
```



According to the Random Forest Model, customer service calls, total day charge and international plan yes are the top 3 most important features contributing to customer churn.

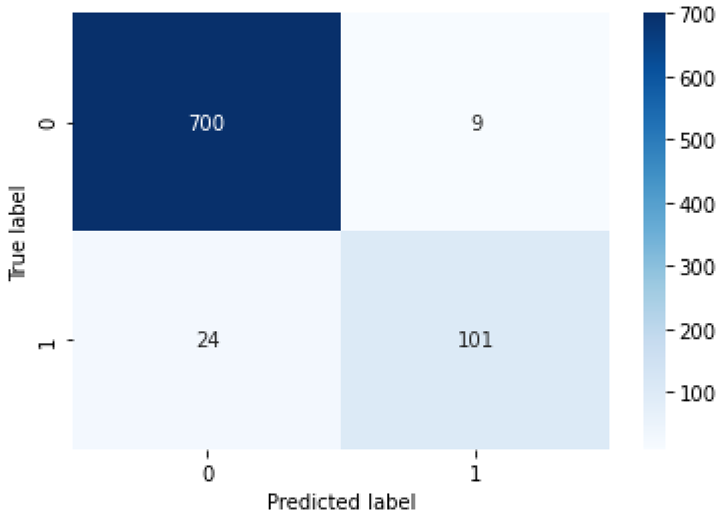
2. Tuned XGBoost Classifier

In [68]:
```
parameters = {
'max_depth':range(3,10,2),
'min_child_weight':range(1,6,2),
'gamma':[i/10.0 for i in range(0,5)],
'learning_rate' : [i/10.0 for i in range(0,5)],
'n_estimators': range(10,150,10)

}
random_search=RandomizedSearchCV(estimator = XGBClassifier(base_score=0.5, bo
                                                           learning_rate=0.1,
                                                           nthread=None, obje
                                                           silent=None, subsa
random_search.fit(X_train_resampled, y_train_resampled)
random_search.best_params_
```

Out[68]:
```
{'n_estimators': 120,
 'min_child_weight': 5,
 'max_depth': 9,
 'learning_rate': 0.3,
 'gamma': 0.0}
```

In [75]:
```
# Instanstiate the model
x_gb_tune = XGBClassifier(learning_rate=0.3, max_depth=9,
                          n_estimators=120, min_child_weight = 5, gamma = 0.0


# fit XGB classifier
x_gb_tune.fit(X_train_resampled, y_train_resampled)

# Make predictions for test data
y_pred_xgb_tune = x_gb.predict(X_test_scaled)
```

In [76]:
```
# Plotting confusion matrix
plot_confusion_matrix(y_test, y_pred_xgb_tune, [0,1])
```



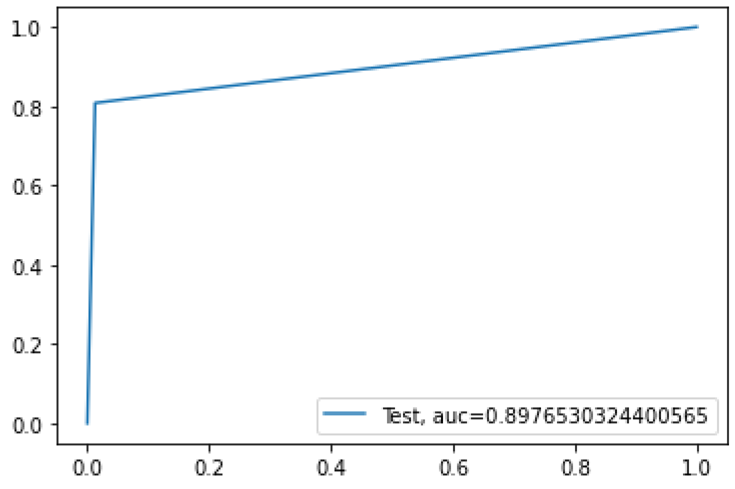
In [77]:
```
# display scores
print(classification_report(y_test,y_pred_xgb_tune))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       709
           1       0.92      0.81      0.86       125

    accuracy                           0.96       834
   macro avg       0.94      0.90      0.92       834
weighted avg       0.96      0.96      0.96       834
```

ROC

In [78]:
```
# Make predictions
y_prob = x_gb.predict(X_test_scaled)
y_pred = (y_prob > 0.5).astype(int)



# Calculate evaluation metrics
roc_value = roc_auc_score(y_test, y_prob)
print("RNN roc_value: {0}" .format(roc_value))
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
threshold = thresholds[np.argmax(tpr-fpr)]



roc_auc = auc(fpr, tpr)
print("ROC for the test dataset",'{:.1%}'.format(roc_auc))
plt.plot(fpr, tpr, label="Test, auc="+str(roc_auc))
plt.legend(loc=4)
plt.show()
```

```
RNN roc_value: 0.8976530324400565
ROC for the test dataset 89.8%
```



Checking for Overfitting

```
In [79]: # Make predictions for test data
y_train_pred_xgb = x_gb.predict(X_train_resampled)
y_test_pred_xgb = x_gb.predict(X_test_scaled)


# Calculate accuracy on the training and test data
train_accuracy = accuracy_score(y_train_resampled, y_train_pred_xgb)
test_accuracy = accuracy_score(y_test, y_test_pred_xgb)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

```
Train Accuracy: 1.0
Test Accuracy: 0.960431654676259
```

Tuned XGBoost Classifier Observations

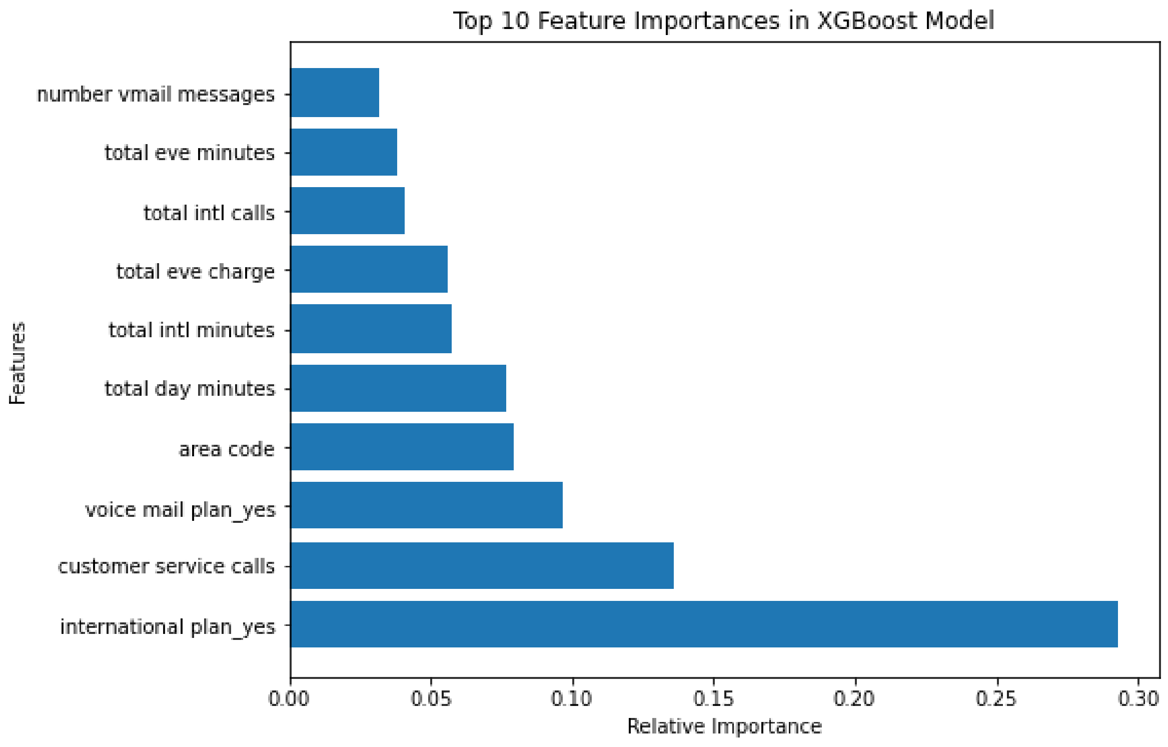
Recall

- For class 0, which represents customers who did not churn, the recall is 0.99. This means that the model correctly identified 99% of the customers who did not churn out of the total number of customers who actually did not churn.
- Similarly, for class 1, which represents customers who churned, the recall is 0.81, indicating that the model correctly identified 81% of the customers who churned out of the total number of customers who actually churned.

Accuracy: 0.96 means that 96% of the total number of customers was correctly classified. The model performs better than Decision Tree Classifier model.

Important Features for tuned XGBoost Model

In [80]:
```
# Assuming 'churn' is the target column, and you want to remove it from churn_
# You can create a new DataFrame without the 'churn' column
churn_df_copy_without_churn = churn_df_copy.drop('churn', axis=1)

# Get the feature importances from the XGBoost model
importances = x_gb_tune.feature_importances_

# Get the indices to sort the features in descending order of importance
indices = np.argsort(importances)[::-1]

# Get the feature names and importances for the top 10 features
top_n = 10
top_feature_names = churn_df_copy_without_churn.columns[indices[:top_n]]
top_importances = importances[indices][:top_n]

# Plot the top 10 feature importances as a horizontal bar plot
plt.figure(figsize=(8, 6))
plt.barh(range(top_n), top_importances, align='center')
plt.yticks(range(top_n), top_feature_names)
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.title('Top 10 Feature Importances in XGBoost Model')
plt.show()
```



According to the XGBoost Model, international plan yes, customer service calls and voice mail plan_yes are the top 3 most important features contributing to customer churn.

Conclusion

- **RandomForestClasssifier**
    - Tuned RandomClassifier model with an AUC of 0.89 suggesting that the model has a strong ability to distinguish between positive (churned) and negative (not churned) instances.
    - This indicates that the model has a good balance between sensitivity (recall) and specificity, capturing a high proportion of both churned and non-churned customers accurately.
    - The recall values varied slightly, with the Tuned Random Forest Classifier performing slightly better in identifying customers who churned at 82%.
    - Tuned Random Forest Classifier had an accuracy of 95% of the total number of customers that were correctly classified
- **XGBoost Classifier**
    - Tuned XGBoost Classifier had an AUC of 0.89, it had a recall for class 1 at 81% and an accuracy of 96%

- **Picking the best model**
    - After carefully analyzing the performance metrics of both models, the Tuned Random Forest Classifier emerges as the better choice for our specific objective of correctly identifying churn customers. With a recall of 82%, the model accurately identifies 82% of the customers who churned out of the total number of customers who actually churned.
    - While the Tuned XGBoost Classifier exhibits a slightly higher accuracy (96%) and recall for non-churn customers (99%), our primary focus lies in correctly identifying churn customers to effectively target retention strategies. The Tuned Random Forest Classifier's recall of 82% for churn customers is commendable and aligns better with our priority.
    - Therefore, we confidently select the Tuned Random Forest Classifier as our best model for predicting customer churn and enabling us to take proactive measures to retain valuable customers, thereby enhancing overall business performance.

- Based on the analysis using our best model (Random Forest Classifier), we can confidently conclude that the three most important factors influencing churn are the number of customer service calls made, the total day charge incurred, and the presence of an international plan

Summary Findings

- Majority of customers who terminated their contracts did not have a voicemail plan.
- California and New Jersey have the highest churn rates, both exceeding 25%
- Customers who terminated their accounts appeared to have subscribed to more day minutes, resulting in higher charges.
- Charges for total daytime calls and minutes were significantly higher compared to evening and nighttime calls and minutes.

- There is a lack of proportionality between the total number of international calls made and the corresponding charges, meaning that charges are higher even with fewer total calls.
- The customers with international plan have the higher churn rate compared to those with

Recommendation

- Ensure fairness in charging, establish a proportional charge for daytime, evening,nighttime and international calls.
- Enhance the voice mail plan service to be more appealing to customers.
- Bring down cost of daytime calls and minutes charges
- Focus more on customer service