SYRIATEL CUSTOMER CHURN ANALYSIS

Author : Prudence Coredo

DSF-FT06P3 - Full-time.

Overview

Syriatel is one of the major telecommunications companies in Syria, providing a range of services including mobile and fixed-line telephony, internet, and other related services. Conducting a churn analysis for Syriatel involves examining customer behaviors and patterns to predict which customers are likely to discontinue their services with the company.

This project uses the syriatel dataset to conduct a comprehensive analysis of churn predictions using various models such as logistic regression , Decison trees classifier and random forest classifier as it is a binary classification problem.

We seek to gather insights that can help syriatel predict which customers are most likely to leave and stay and what factors cause or instigate this decision.

Business Understanding

Our main objective is to build a model that can accurately predict when a customer is about to leave by analysisng the behaviors and trends that precede churn. Identify indicators or triggers indicating a higher likelihood of churn.

To identify which features are correlated to churn.

How can we keep our customer base to reduce churn while also attracting new customers. This analysis should provid some insights on strategies we can take to keep our customers instead of spending alot of money to attract new ones.

To the stakeholder which is SyriaTel. We are interested in reducing how much money is lost because of customers who don't stick around very long and take precautions.

Data Understanding

The dataset contains information about syriaTel customers and is in a csv file format. Some features included in the data are state of a client , minutes clients have used, how many day/evening/international calls they have made, the number of times a customer has called

customer service and if they have churned. Our target variable is Churn. The dataset contains

Importing the libraries

In [278]:
```
# Import the modules & packages

# Data manipulation
import pandas as pd
import numpy as np

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.colors as colors
import plotly.graph_objs as go
from plotly.offline import iplot
from plotly.subplots import make_subplots

# Modeling
#splitting the dataset into test-train
from sklearn.model_selection import train_test_split,cross_val_score,GridSearc

from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score,f1_score,recall_score,precision_sco
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from scipy import stats

# Feature Selection, Feature Importance
#from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.inspection import permutation_importance
#from mlxtend.plotting import plot_sequential_feature_selection
from sklearn.feature_selection import RFE

# Algorithms for supervised learning methods
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Filtering future warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [279]:
```
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df
```

Out[279]:

	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	...
...	...	...	...	...	...	...	...	...	...	...	...
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	...

3333 rows × 21 columns

Summary of Features in the Datset

state: the state the customer lives in

account length: the number of days the customer has had an account

area code: the area code of the customer

phone number: the phone number of the customer

international plan: true if the customer has the international plan, otherwise false

voice mail plan: true if the customer has the voice mail plan, otherwise false

number vmail messages: the number of voicemails the customer has sent

total day minutes: total number of minutes the customer has been in calls during the day

total day calls: total number of calls the user has done during the day

total day charge: total amount of money the customer was charged by the Telecom company for calls during the day

total eve minutes: total number of minutes the customer has been in calls during the evening

total eve calls: total number of calls the customer has done during the evening

total eve charge: total amount of money the customer was charged by the Telecom company for calls during the evening

total night minutes: total number of minutes the customer has been in calls during the night

total night calls: total number of calls the customer has done during the night

total night charge: total amount of money the customer was charged by the Telecom company for calls during the night

total intl minutes: total number of minutes the user has been in international calls

total intl calls: total number of international calls the customer has done

total intl charge: total amount of money the customer was charged by the Telecom company for international calls

customer service calls: number of calls the customer has made to customer service

churn: true if the customer terminated their contract, otherwise false

Data exploration

In [280]:
```
# Checking the shape of our dataset
df.shape
```

Out[280]:
```
(3333, 21)
```

statistical summary description of the numeric features

In [281]:
```
df.describe().transpose()
```

Out[281]:

	count	mean	std	min	25%	50%	75%	max
account length	3333.0	101.064806	39.822106	1.00	74.00	101.00	127.00	243.00
area code	3333.0	437.182418	42.371290	408.00	408.00	415.00	510.00	510.00
number vmail messages	3333.0	8.099010	13.688365	0.00	0.00	0.00	20.00	51.00
total day minutes	3333.0	179.775098	54.467389	0.00	143.70	179.40	216.40	350.80
total day calls	3333.0	100.435644	20.069084	0.00	87.00	101.00	114.00	165.00
total day charge	3333.0	30.562307	9.259435	0.00	24.43	30.50	36.79	59.64
total eve minutes	3333.0	200.980348	50.713844	0.00	166.60	201.40	235.30	363.70
total eve calls	3333.0	100.114311	19.922625	0.00	87.00	100.00	114.00	170.00
total eve charge	3333.0	17.083540	4.310668	0.00	14.16	17.12	20.00	30.91
total night minutes	3333.0	200.872037	50.573847	23.20	167.00	201.20	235.30	395.00
total night calls	3333.0	100.107711	19.568609	33.00	87.00	100.00	113.00	175.00
total night charge	3333.0	9.039325	2.275873	1.04	7.52	9.05	10.59	17.77
total intl minutes	3333.0	10.237294	2.791840	0.00	8.50	10.30	12.10	20.00
total intl calls	3333.0	4.479448	2.461214	0.00	3.00	4.00	6.00	20.00
total intl charge	3333.0	2.764581	0.753773	0.00	2.30	2.78	3.27	5.40
customer service calls	3333.0	1.562856	1.315491	0.00	1.00	1.00	2.00	9.00

Based on this Area code should be changed to Categorical data

In [282]:
```
# Information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

Data preparation

This process prepares the data for EDA and modeling. It entails checking for missing values, duplicates and removing features that are not necessary in our analysis.

In [283]:
```
# Checking for duplicated values
df.duplicated().sum()
```

Out[283]:
```
0
```

In [284]:
```
# Checking for missing values.
df.isnull().sum()
```

Out[284]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

The dataset has no missing values or duplicates to deal with

In [285]:
```
# Checking for the number of unique values in all the features.
df.nunique()
```

Out[285]:
```
state                      51
account length            212
area code                   3
phone number             3333
international plan          2
voice mail plan             2
number vmail messages      46
total day minutes        1667
total day calls           119
total day charge         1667
total eve minutes        1611
total eve calls           123
total eve charge         1440
total night minutes      1591
total night calls         120
total night charge        933
total intl minutes        162
total intl calls           21
total intl charge         162
customer service calls     10
churn                       2
dtype: int64
```

In [286]:
```
# We drop the phone number feature as it contains the customer's contact infor
df.drop(['phone number'],axis=1,inplace=True)
df.head()
```

Out[286]:

	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
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	110
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	88
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	122

Creating numeric and categorical list

In [287]:
```
numeric_cols = ['account length','number vmail messages','total day minutes','
                'total eve minutes','total eve calls','total eve charge','tota
                'total night charge','total intl minutes','total intl calls','

categoric_cols = ['state','area code','international plan','voice mail plan']
```

Separating the numeric and categorical features will help in the analysis and when modelling.

Categorical Features:

state

area code

international plan

voicemail plan

Continuous Features:

These columns have numeric values.

account length

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 charge

customer service calls

Changing Data Types

In [288]:
```
# Changing the datatype fro int to str so that area code can be a categorical
df["area code"] = df["area code"].astype("str")
print(df["area code"].dtype)
```

```
object
```

In [289]:
```
# Transforming "Churn" Feature's Rows into 0s and 1s
df['churn'] = df['churn'].map({True: 1, False: 0}).astype('int')
df.head()
```

Out[289]:

	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
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	110
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	88
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	122

Exploratory Data Analysis

This is the exploration and assessment of the dataset. Including they're distribution and relation with our target feature churn.

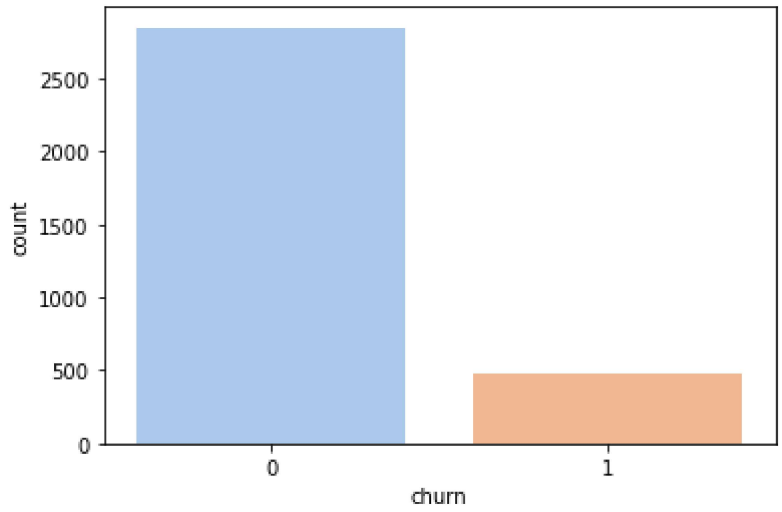
Churn Feature

Churn is our dependent (target) variables.

Churn indicates if a customer has terminated their contract with SyriaTel. True [1] indicates they have terminated and false [0] indicates they have not and have and have an existing account.

In [290]:
```
print(df['churn'].value_counts())
sns.set_palette('pastel')
sns.countplot(data=df, x='churn');
```

```
0    2850
1     483
Name: churn, dtype: int64
```



483 out of 3,333 have terminated their contract with SyriaTel and 2850 still have an account.

This means that 14.5% of the customers are lost.

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

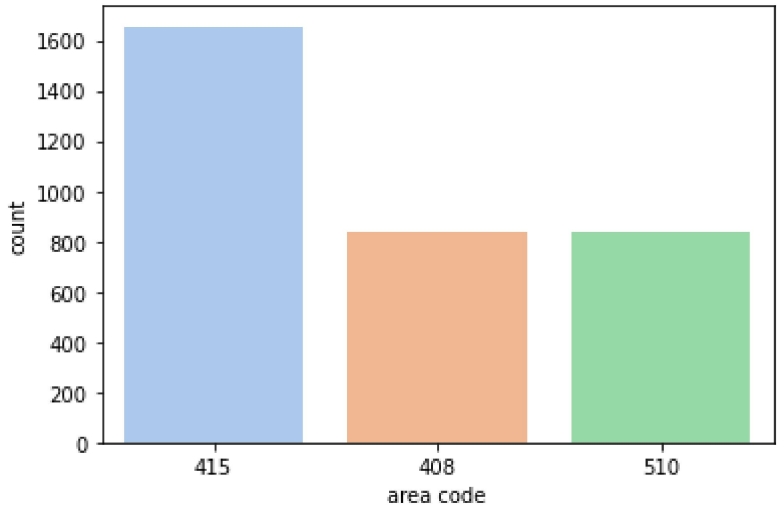
Area Code

Pie chart of area code feature

In [291]:
```
area = df['area code'].value_counts()
print(area)
sns.countplot(data=df, x='area code')
```

```
415    1655
510     840
408     838
Name: area code, dtype: int64
```

Out[291]: <AxesSubplot:xlabel='area code', ylabel='count'>



The area code 415 is used by half the customers(1655) , 840 uses the code 510 and 838 uses 408. The difference between the 2 is very minimal.

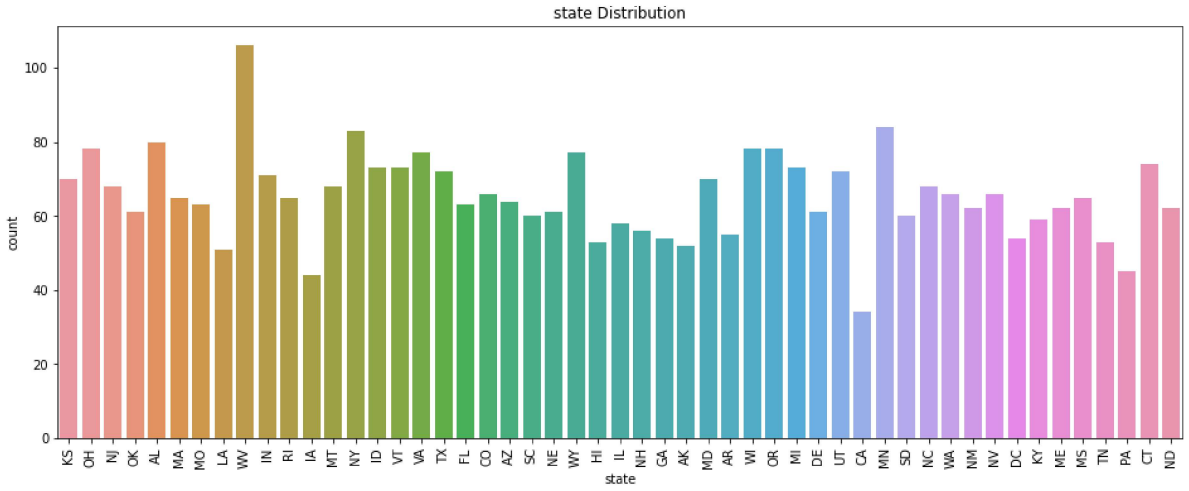
The area code 415 covers parts of the San Francisco Bay Area.

State Feature

In [292]:
```
# Checking how many customers we have per state.
df['state'].value_counts()
```

Out[292]:
```
WV    106
MN     84
NY     83
AL     80
OR     78
OH     78
WI     78
VA     77
WY     77
CT     74
ID     73
MI     73
VT     73
TX     72
UT     72
IN     71
KS     70
MD     70
NJ     68
NC     68
MT     68
NV     66
CO     66
WA     66
MS     65
RI     65
MA     65
AZ     64
FL     63
MO     63
NM     62
ME     62
ND     62
DE     61
OK     61
NE     61
SC     60
SD     60
KY     59
IL     58
NH     56
AR     55
DC     54
GA     54
HI     53
TN     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
```

In [293]:
```
# Calculate value counts and plot bar plots for categorical variables

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



The state with the highest count is West Virginia (WV) with 106 occurrences, indicating it is the most frequent state in the dataset.

Minnesota (MN) follows closely with 84 occurrences, making it the second most common state.

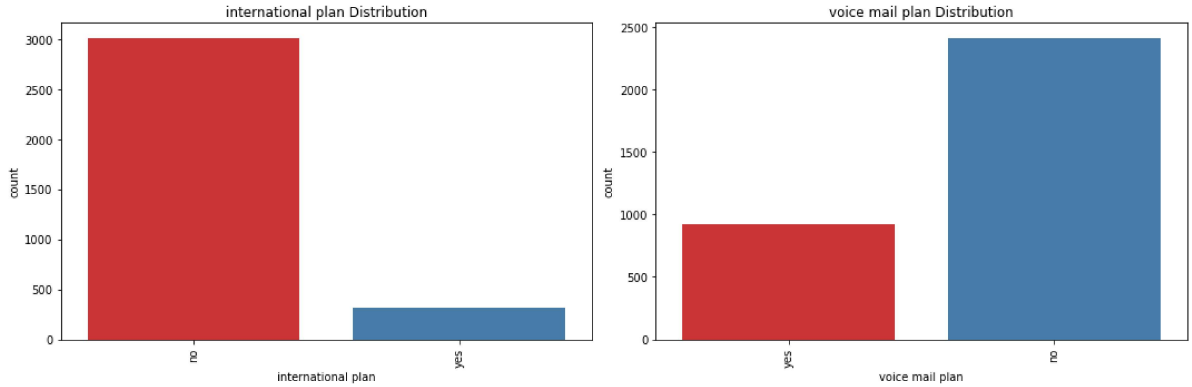
New York (NY) comes next with 83 occurrences, showing a similar frequency to Minnesota.

Alabama (AL), Wisconsin (WI), Oregon (OR), and Ohio (OH) all have 78 occurrences, placing them among the top states in terms of frequency.

The state with the lowest count is California (CA) with only 34 occurrences.

International Plan and Voice Mail Plan

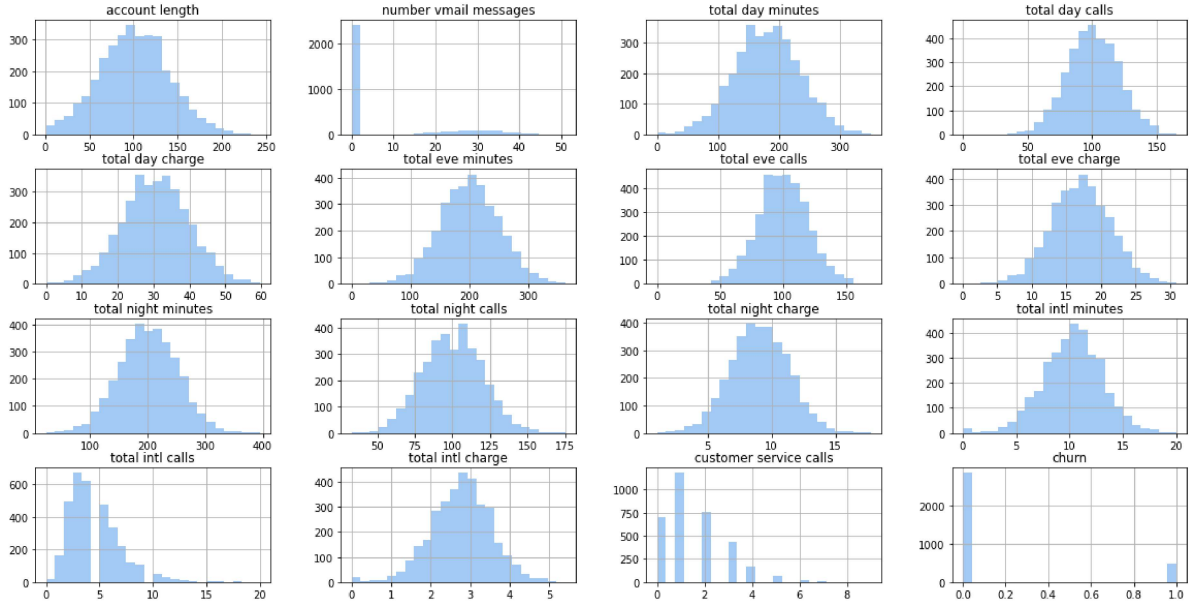
In [294]:
```
# Set up the figure and axes for subplots
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

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


for i, col in enumerate(categorical_cols):
    sns.countplot(x=col, data=df, ax=axs[i], palette = 'Set1')
    axs[i].set_title(f"{col} Distribution")
    axs[i].tick_params(axis='x', rotation=90)

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



Majority of the customers don't have an international plan and voice mail plan.

Distribution plot for the numeric values

In [295]:
```
df.hist(figsize = (20, 10), bins = 24)
plt.show()
```



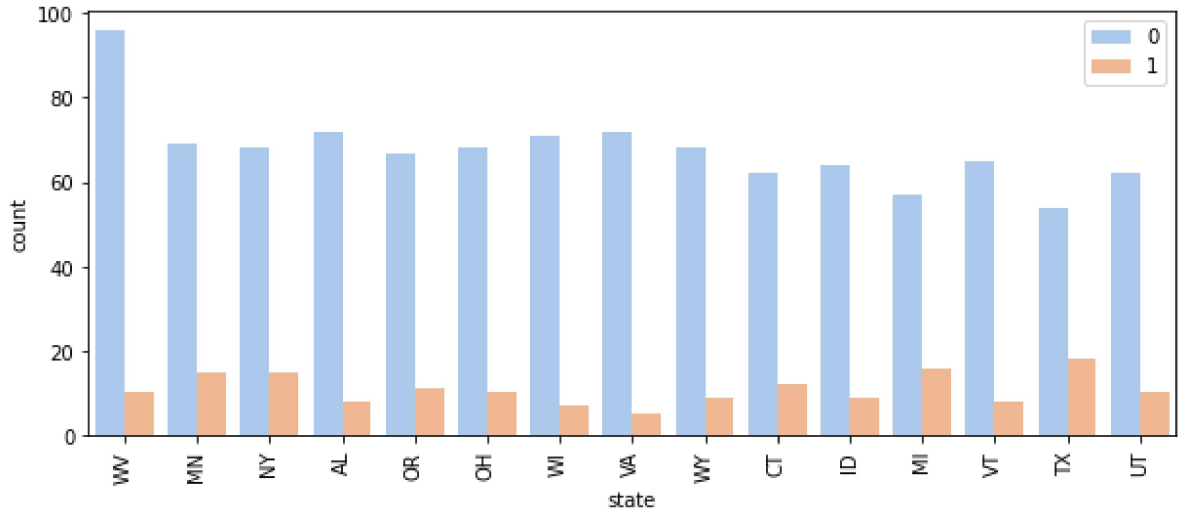
Many of the features have a normal distribution.

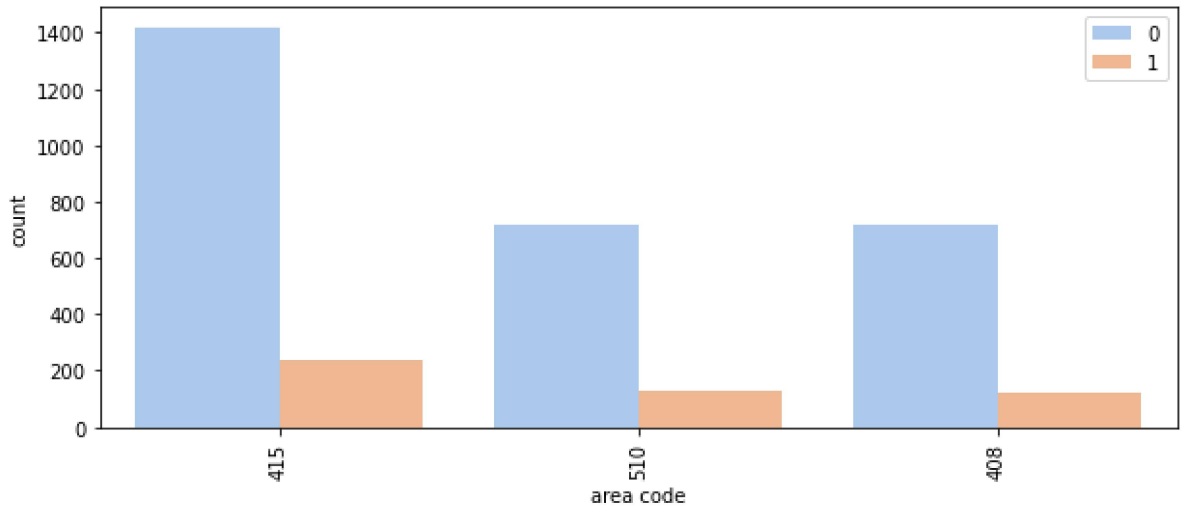
Bivariate Analysis

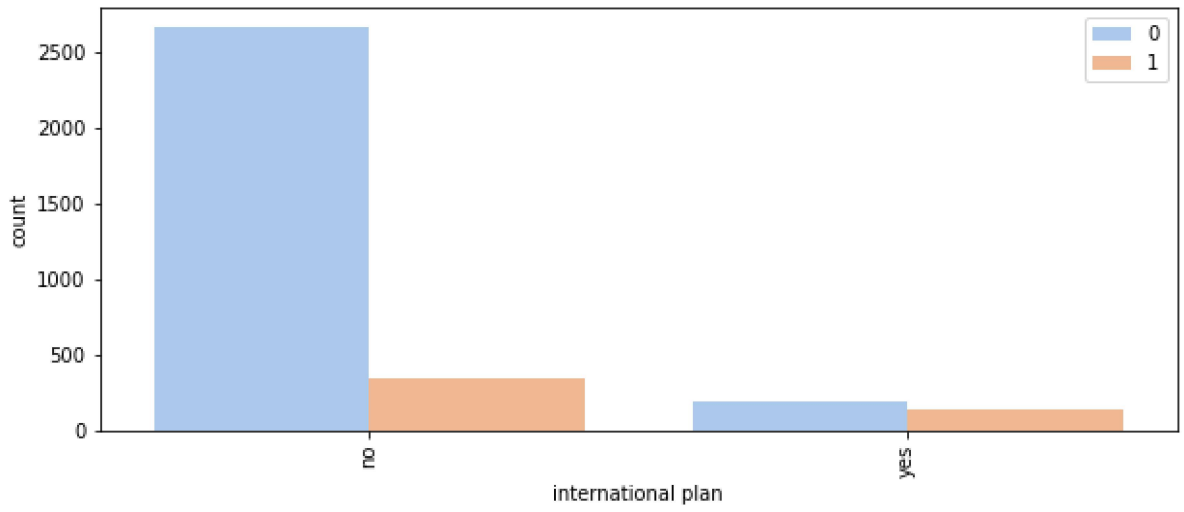
How our features are distributed with our target feature.

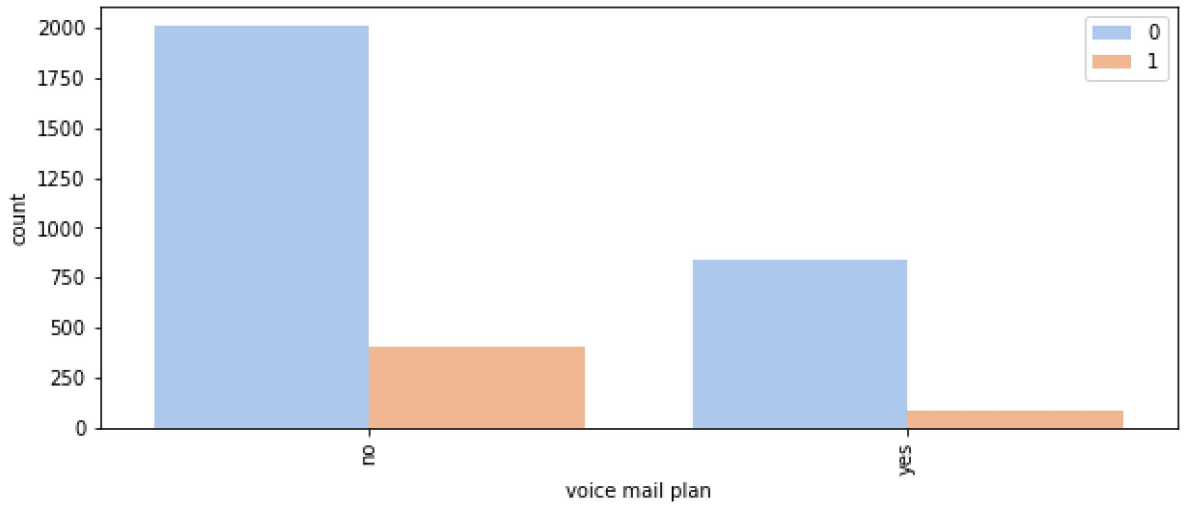
Categorical features

In [296]:
```
# For loop function to plot all the distribution of our categorical features i
for i in categoric_cols:
    plt.figure(figsize=(10,4))
    sns.countplot(x=i, hue="churn", data=df,order= df[i].value_counts().iloc[0
    plt.xticks(rotation=90)
    plt.legend(loc="upper right")
    plt.show()
```









From the graphs, very few customers have stopped working with syriatel based on these features, this suggests that the reason for customers leaving might be due to them.

Churn rates vary between the different area codes, with area code 415 having the highest churn rate and area code 408 having the lowest churn rate.

Customers without a voice mail plan and those without an international plan had a higher churn rate compared to customers with a voice mail plan and an international plan.

The proportion of churn for customers with an international plan is significantly higher compared to those without. This indicates that the presence of an international plan may be a significant factor in customer churn.

Numeric features

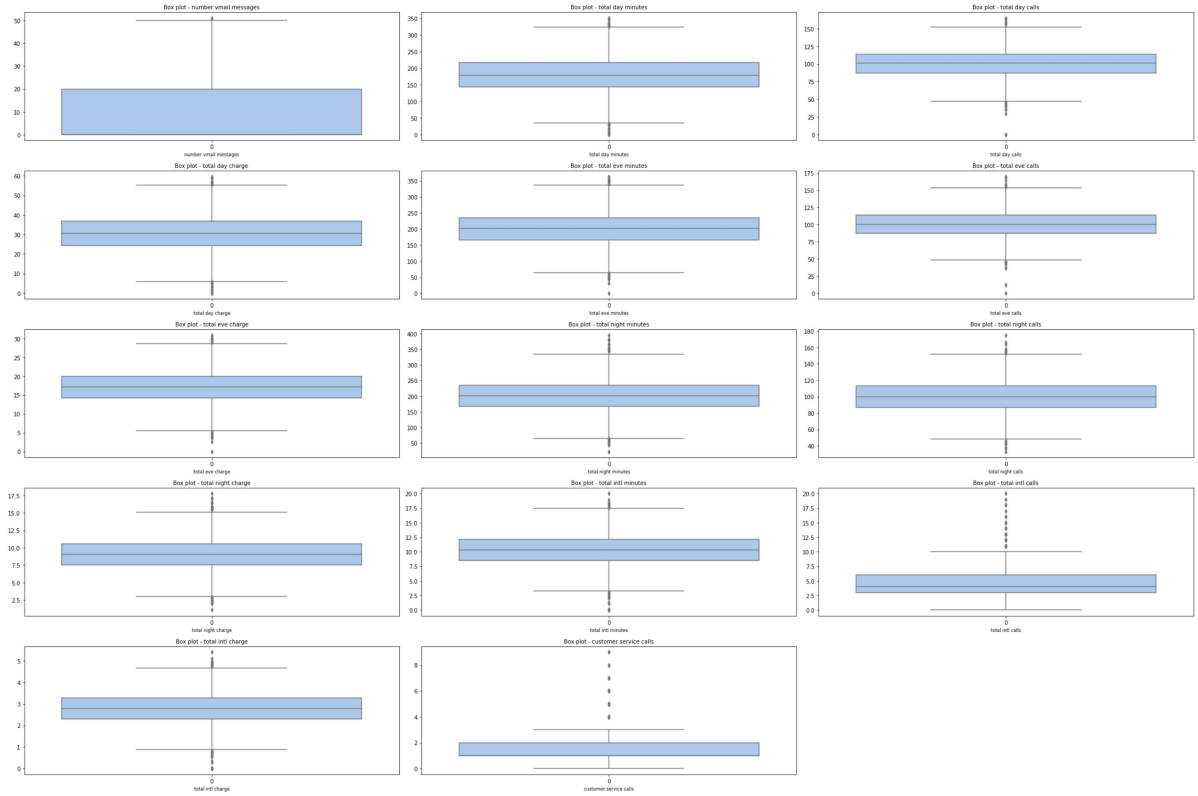
Distribution plot for all the numberic features with their relation to churn

In [297]:
```
#sing Plotly to display histograms for different features in the dataset, compa

churn = df[df["churn"] == 1]
no_churn = df[df["churn"] == 0]
colors = ['rgb(31, 119, 180)', 'rgb(255, 127, 14)']  # Churn: blue, No Churn: 

def create_churn_trace(col, visible=False):
    return go.Histogram(
        x=churn[col],
        name='Churn',
        marker=dict(color=colors[0]),
        visible=visible
    )

def create_no_churn_trace(col, visible=False):
    return go.Histogram(
        x=no_churn[col],
        name='No Churn',
        marker=dict(color=colors[1]),
        visible=visible
    )

features_not_for_hist = ["state", "churn", 'area code', 'international plan',
features_for_hist = [x for x in df.columns if x not in features_not_for_hist]

n_features = len(features_for_hist)
rows = int(n_features / 2) + n_features % 2
cols = 2
fig = make_subplots(rows=rows, cols=cols, subplot_titles=features_for_hist)

for i, feature in enumerate(features_for_hist):
    row = (i // cols) + 1
    col = (i % cols) + 1

    fig.add_trace(create_churn_trace(feature, visible=True), row=row, col=col)
    fig.add_trace(create_no_churn_trace(feature, visible=True), row=row, col=c

    fig.update_xaxes(title_text=feature, row=row, col=col)
    fig.update_yaxes(title_text="# Samples", row=row, col=col)

    fig.update_layout(
        showlegend=False,
        height=rows * 300,
        width=900,
        title="Feature Distribution: Churn vs No Churn"
    )

fig.show()
```

The distributions of minutes, charges, and number of calls in the evening and night do not show significant differences between customers with churn and those without churn.

Checking for outliers

In [298]:
```
columns = ['number vmail messages', 'total day minutes', 'total day calls', 't
           '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']

# Calculate the required number of rows and columns for subplots
num_rows = (len(columns) - 1) // 3 + 1
num_cols = min(len(columns), 3)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10*num_cols, 4*num_rows))

# Generate box plots for each column
for i, column in enumerate(columns):
    row = i // num_cols
    col = i % num_cols
    sns.boxplot(data=df[column], ax=axes[row, col])
    axes[row, col].set_title(f'Box plot - {column}', fontsize=10)
    axes[row, col].set_xlabel(column, fontsize=8)

# Remove any empty subplots
if i < (num_rows * num_cols) - 1:
    for j in range(i + 1, num_rows * num_cols):
        fig.delaxes(axes.flatten()[j])

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



Dealing with outliers before modelling.

In [299]:
```
print("Before dropping numerical outliers, length of the dataframe is: ",len(d
def drop_numerical_outliers(df, z_thresh=3):
    constrains = df.select_dtypes(include=[np.number]).apply(lambda x: np.abs(
        .all(axis=1)
    df.drop(df.index[~constrains], inplace=True)

drop_numerical_outliers(df)
print("After dropping numerical outliers, length of the dataframe is: ",len(df
print(df.shape)
```

```
Before dropping numerical outliers, length of the dataframe is:  3333
After dropping numerical outliers, length of the dataframe is:  3169
(3169, 20)
```

Modelling

Data Processing

Checking highly correlated features

In [300]:
```
## Defining a function to check highly correlated features
def check_collinearity(df, threshold=0.8):
    corr_matrix = df.select_dtypes(include=np.number).corr().abs()
    correlated_pairs = set()
    for col in corr_matrix:
        correlated_cols = corr_matrix.index[corr_matrix[col] > threshold]
        correlated_pairs.update([(min(col, correlated_col), max(col, correlate
    for pair in correlated_pairs:
        print(f"{pair[0]} --- {pair[1]}")
    return set(df.columns) & set(col for pair in correlated_pairs for col in p

# Call the function to check multicollinearity
multicollinear_features = check_collinearity(df)
```

```
total day charge --- total day minutes
total eve charge --- total eve minutes
total intl charge --- total intl minutes
total night charge --- total night minutes
```

These features are highly correlated with each other hence we need to drop one of the pair.

In [301]:
```
# Drop some columns in order to deal with multicollinearity
features= ['total day charge', 'total eve charge','total night charge', 'total
df_new =df.drop(features,axis=1)
df_new.head()
```

Out[301]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes
0	KS	128	415	no	yes	25	265.1	110	197.4	99	244.7
1	OH	107	415	no	yes	26	161.6	123	195.5	103	254.4
2	NJ	137	415	no	no	0	243.4	114	121.2	110	162.6
3	OH	84	408	yes	no	0	299.4	71	61.9	88	196.9
4	OK	75	415	yes	no	0	166.7	113	148.3	122	186.9

In [302]:
```
# Correlation between churn and all features.
df.corr()['churn']
```

Out[302]:
```
account length            0.016993
number vmail messages    -0.094053
total day minutes         0.220531
total day calls           0.021057
total day charge          0.220529
total eve minutes         0.097185
total eve calls           0.006586
total eve charge          0.097172
total night minutes       0.039937
total night calls         0.004197
total night charge        0.039947
total intl minutes        0.065636
total intl calls         -0.066175
total intl charge         0.065662
customer service calls    0.174661
churn                     1.000000
Name: churn, dtype: float64
```

Feature Engineering

One-Hot Encoding categorical features

Changing the categorical features to numeric in preparation for modelling.

In [303]:
```
# Create dummy variables for the "area code" feature
dummy_df_area_code = pd.get_dummies(df_new["area code"], dtype=np.int64, prefi

# Create dummy variables for the "international plan" feature and drop the fir
dummy_df_international_plan = pd.get_dummies(df_new["international plan"], dty

# Create dummy variables for the "voice mail plan" feature and drop the first
dummy_df_voice_mail_plan = pd.get_dummies(df_new["voice mail plan"], dtype=np.

# Concatenate the dummy variables with the original dataframe
df_new = pd.concat([df_new, dummy_df_area_code, dummy_df_international_plan, d

# Remove duplicate columns, if any
df_new = df_new.loc[:, ~df_new.columns.duplicated()]

# Drop the original "area code", "international plan", and "voice mail plan" c
df_new = df_new.drop(['area code', 'international plan', 'voice mail plan'], a
```

label encoding the state categorical feature

We use this method for state because it has many unique values. Each state is given a number.

In [304]:
```
label = LabelEncoder()
label.fit(df_new['state'])
df_new['state'] = label.transform(df_new['state'])
df_new.head()
```

Out[304]:

	state	account length	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	cus s
0	16	128	25	265.1	110	197.4	99	244.7	91	10.0	3	
1	35	107	26	161.6	123	195.5	103	254.4	103	13.7	3	
2	31	137	0	243.4	114	121.2	110	162.6	104	12.2	5	
3	35	84	0	299.4	71	61.9	88	196.9	89	6.6	7	
4	36	75	0	166.7	113	148.3	122	186.9	121	10.1	3	

Scaling using Min-Max Normalization

MinMaxScaler is used to reduce the effects of outliers in the dataset.

In [305]:
```
#Min-Max Normalization
transformer = MinMaxScaler()

def scaling(columns):
    return transformer.fit_transform(df_new[columns].values.reshape(-1,1))

for i in df_new.select_dtypes(include=[np.number]).columns:
    df_new[i] = scaling(i)

df_new.head()
```

Out[305]:

	state	account length	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total minu
0	0.32	0.587963	0.510204	0.773921	0.576271	0.490079	0.487179	0.643519	0.422414	0.487
1	0.70	0.490741	0.530612	0.450281	0.686441	0.483796	0.521368	0.675595	0.525862	0.713
2	0.62	0.629630	0.000000	0.706066	0.610169	0.238095	0.581197	0.372024	0.534483	0.621
3	0.70	0.384259	0.000000	0.881176	0.245763	0.041997	0.393162	0.485450	0.405172	0.280
4	0.72	0.342593	0.000000	0.466229	0.601695	0.327712	0.683761	0.452381	0.681034	0.493

Train-Test Split

Split the data into 80/20 : 80% training , 20% testing

In [306]:
```
# define x and y , y being you're target feature and x the independent variabl
X = df_new.drop(['churn'], axis=1)
y = df_new['churn']

# Split the data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[306]:
```
((2535, 17), (634, 17), (2535,), (634,))
```

Applying SMOTE to handle class imbalances

In [307]:
```
df_new.churn.value_counts()
```

Out[307]:
```
0.0    2727
1.0     442
Name: churn, dtype: int64
```

In [308]:
```
# Create an instance of SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training set
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print(X_train_resampled.shape,  y_train_resampled.shape)
```

```
(4382, 17) (4382,)
```

In [309]:
```
# Checking the class imbalance.
y_train_resampled.value_counts()
```

Out[309]:
```
1.0    2191
0.0    2191
Name: churn, dtype: int64
```

Models

1. Logistic Regression

Logistic Regression is used when the target variable is categorical with two possible outcomes in binarry format 0 and 1

In [310]:
```
# Object creation, fitting the data & getting predictions
log_reg= LogisticRegression()
log_reg.fit(X_train_resampled,y_train_resampled)
y_pred_log_reg = log_reg.predict(X_test)
```

Classifiation report

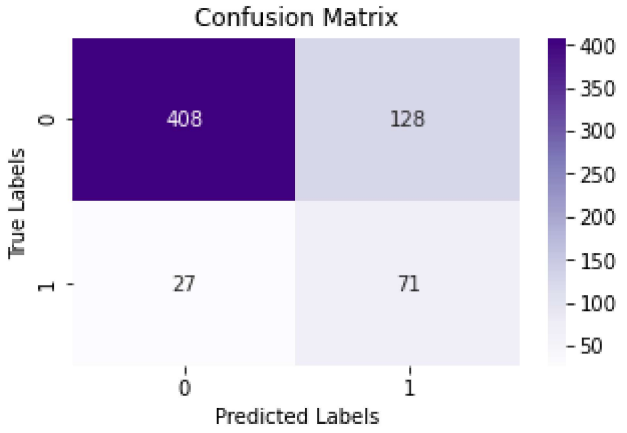
In [311]:
```
print(classification_report(y_test, y_pred_log_reg, target_names=['0', '1']))
```

```
              precision    recall  f1-score   support

           0       0.94      0.76      0.84       536
           1       0.36      0.72      0.48        98

    accuracy                           0.76       634
   macro avg       0.65      0.74      0.66       634
weighted avg       0.85      0.76      0.78       634
```

Classification Matrix

In [312]:
```
print('-----LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS----')
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_lo
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_log_reg),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_log_re
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_
cm_lr = confusion_matrix(y_test, y_pred_log_reg)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_lr, annot=True, cmap='Purples', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
-----LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS----
Accuracy score for testing set:  0.75552
F1 score for testing set:  0.47811
Recall score for testing set:  0.72449
Precision score for testing set:  0.35678
```



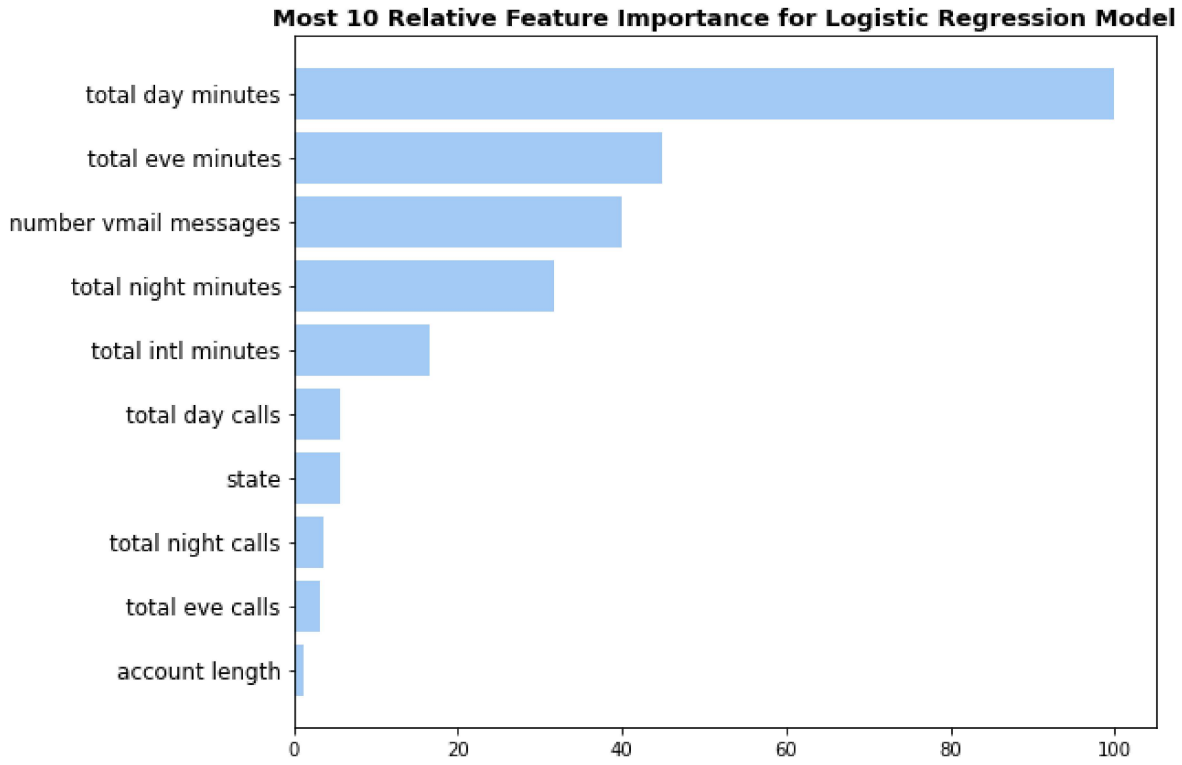
The accuracy on the testing set is approximately 76%. It indicates the overall correctness of predictions made by the model.

The F1 score of 0.47811 represents a balanced evaluation of the model's accuracy in predicting both churn and non-churn customers.

The recall score on the testing set is approximately 0.72449, indicating that the model correctly identified around 72% of the actual positive cases.

The precision score for the testing set is roughly 0.35678, implying that out of all the predicted positive cases, around 35.68% were actually positive.

In [313]:
```
# Feature Importances
feature_importance = abs(log_reg.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())[0
sorted_idx = np.argsort(feature_importance)[0:10]
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure(figsize=(9, 6))
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
plt.title('Most 10 Relative Feature Importance for Logistic Regression Model',
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=12)

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



Based on the logistic regression classifier model, total day minutes, total evening minutes and number of voice mail messsages are the top three important features that can be used to predict churn.

2. Decision Tress

Decision tree classifier is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In [314]:
```
# Create an instance of the decision tree classifier and fit the model on the
decision = DecisionTreeClassifier(random_state=42)
#Fit the model on the training data
decision.fit(X_train_resampled, y_train_resampled)
#Make predictions on the testing data
y_pred = decision.predict(X_test)
# Model evaluation
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93       536
           1       0.61      0.78      0.68        98

    accuracy                           0.89       634
   macro avg       0.78      0.84      0.81       634
weighted avg       0.90      0.89      0.89       634
```

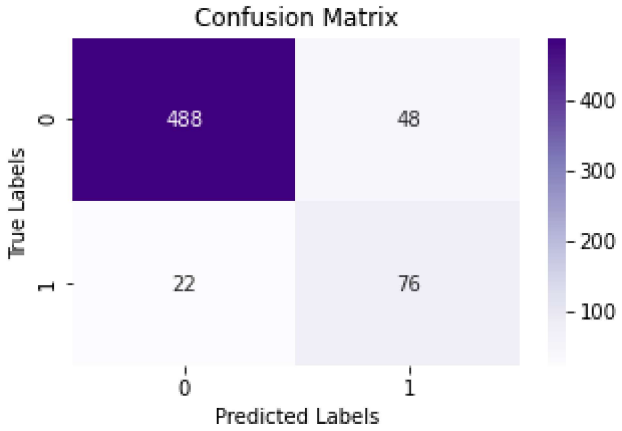
Better perfomance than the logistic model.

The precision is 96%. It means that among all the instances predicted as class 0, 96% were actually in class 0. however class 1 has a precision score of 61%. This indicates that among all the instances predicted as class 1, only 61% were actually in class 1. The lower precision for class 1 suggests that there might be some false positives in the predictions.

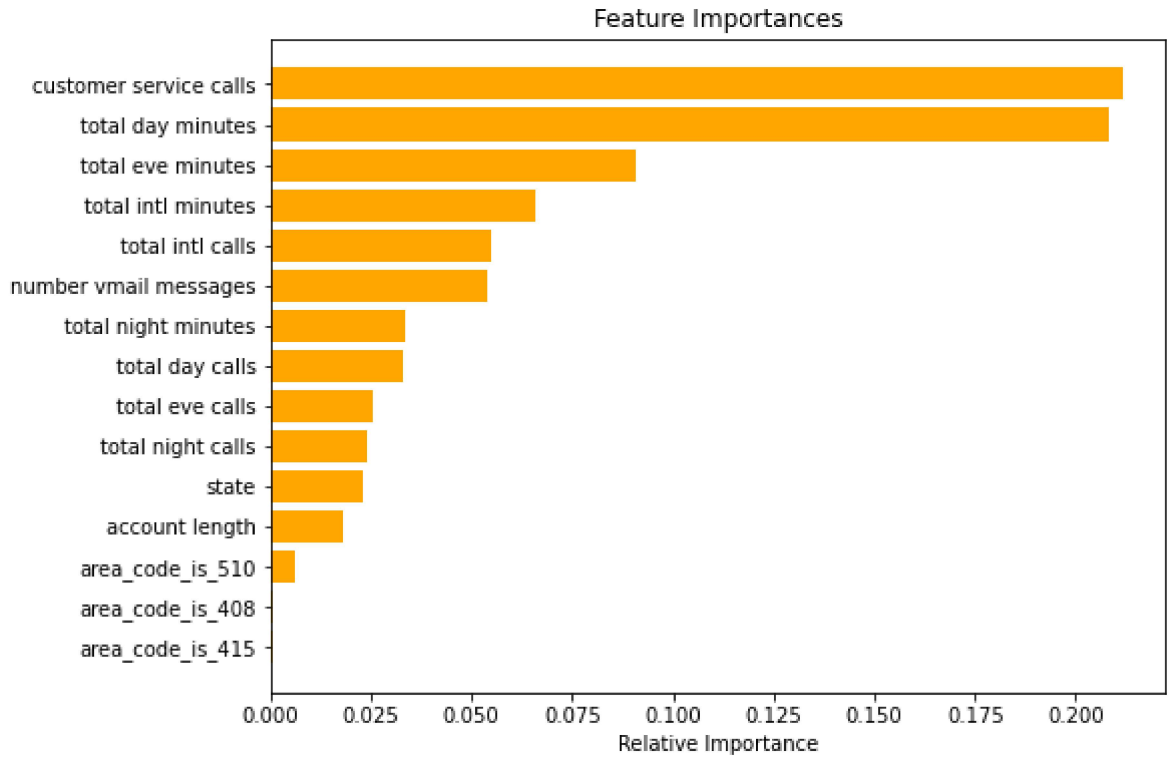
In summary, while the model shows strong performance in correctly predicting class 0 with high precision and recall, there's room for improvement in precision for class 1.

In [315]:
```
print('-----DECISION CLASSIFIER MODEL RESULTS----')
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred),5
print('F1 score for testing set: ',round(f1_score(y_test,y_pred),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred)
cm_decision = confusion_matrix(y_test, y_pred)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_decision, annot=True, cmap='Purples', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
-----DECISION CLASSIFIER MODEL RESULTS----
Accuracy score for testing set:  0.88959
F1 score for testing set:  0.68468
Recall score for testing set:  0.77551
Precision score for testing set:  0.6129
```



In [316]:
```
feature_names = list(X_train_resampled.columns)
importances = decision.feature_importances_[0:15]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='orange', align='cen
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



3. Random Forests

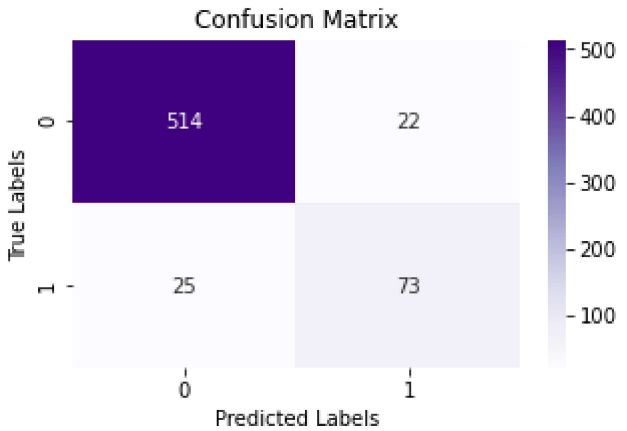
Random Forests is a versatile and powerful ensemble learning method primarily used for classification problems. It operates by constructing multiple decision trees and merging their predictions to obtain more accurate and stable results.

In [317]:
```
#Instantiate the classifier
rf_clf= RandomForestClassifier(random_state=42)

#Fit on the training data
rf_clf.fit(X_train_resampled,y_train_resampled)
RandomForestClassifier(random_state=42)
#predict on the test data
y_pred_rf = rf_clf.predict(X_test)
print(classification_report(y_test,y_pred_rf))
#plot_confusion_matrix(y_test, y_pred_rf, [0,1])
```

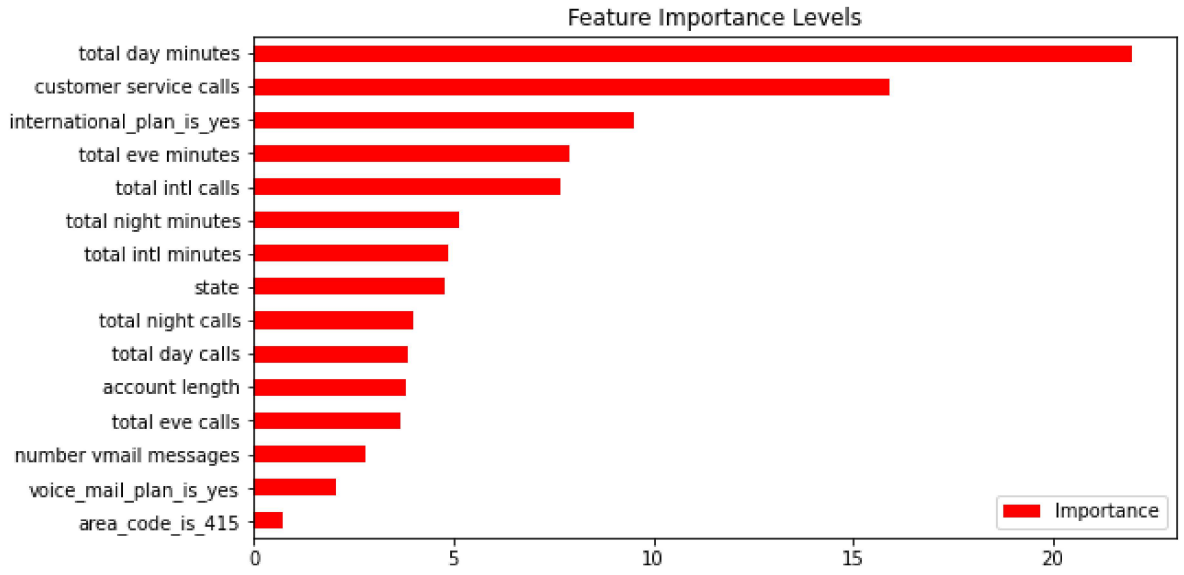
```
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96       536
         1.0       0.77      0.74      0.76        98

    accuracy                           0.93       634
   macro avg       0.86      0.85      0.86       634
weighted avg       0.92      0.93      0.93       634
```

In [318]:
```
print('-----RANDOM FORESTS CLASSIFIER MODEL RESULTS----')
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_rf
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_rf),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_rf),5)
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_
cm_decision = confusion_matrix(y_test, y_pred_rf)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_decision, annot=True, cmap='Purples', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
-----RANDOM FORESTS CLASSIFIER MODEL RESULTS----
Accuracy score for testing set:  0.92587
F1 score for testing set:  0.75648
Recall score for testing set:  0.7449
Precision score for testing set:  0.76842
```



In [319]:
```
Importance =pd.DataFrame({"Importance": rf_clf.feature_importances_*100},index
Importance.sort_values(by = "Importance", axis = 0, ascending = True).tail(15)
plt.title("Feature Importance Levels");
plt.show()
```



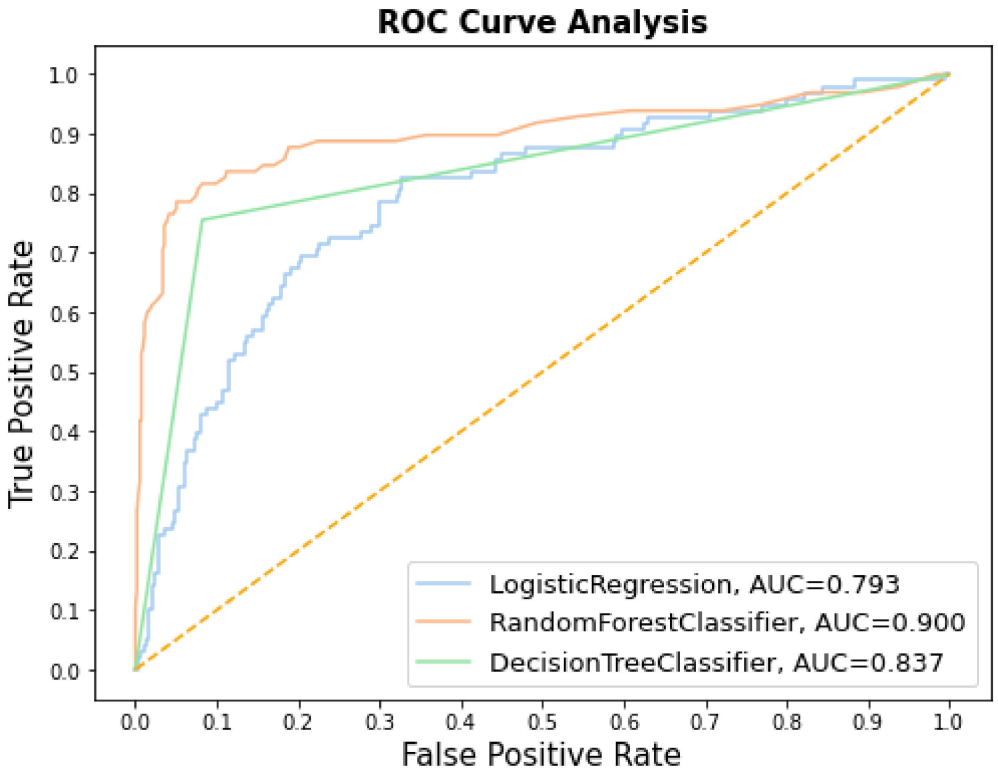
Total day minutes , customer serive calls and those with an internation plan are top three features that can predict churn.

Across all 3 models total day minutes is always top 2 along wiht customer service calls. it is safe to say they are the most important features. An international plan and total evening minutes follow next.

Model Comparision.

ROC SCORE

In [320]:
```
classifiers = [LogisticRegression(),
               RandomForestClassifier(),
               DecisionTreeClassifier()]


# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train_resampled, y_train_resampled)
    yproba = model.predict_proba(X_test)[::,1]

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

    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# 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', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```



The ROC curve illustrates the true positive rate against the false positive rate of our classifier. The best performing models will have a curve that hugs the upper left of the graph, which is the the random forest classifier in this case.

We can say that random forest classifier perfomed best

Model Evaluation

In [321]:
```
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
    "Decision Trees": DecisionTreeClassifier(),
}
for model_name, model in models.items():
    # Train the model on the resampled data
    model.fit(X_train_resampled, y_train_resampled)
```

In [322]:
```
def calculate_metrics(y_true, y_pred):
    """
    Calculate model performance metrics: accuracy, precision, recall, and F1-s
    :param y_true: True labels.
    :param y_pred: Predicted labels.
    :return: Dictionary of metrics.
    """
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    # Return as a dictionary
    return {"Accuracy": accuracy, "Precision": precision, "Recall": recall, "F

# Dictionary to hold the results
results = {}

# For each model
for model_name, model in models.items():
    # Make predictions on the test set
    y_pred_test = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    # Calculate metrics
    metrics_test = calculate_metrics(y_test, y_pred_test)
    metrics_train = calculate_metrics(y_train, y_pred_train)

    # Store the results
    results[(model_name, 'Test')] = metrics_test
    results[(model_name, 'Train')] = metrics_train
    # Convert the results dictionary to a DataFrame
results_df = pd.DataFrame(results).T

results_df
```

Out[322]:

		Accuracy	Precision	Recall	F1-score
Logistic Regression	**Test**	0.755521	0.356784	0.724490	0.478114
	Train	0.769231	0.345315	0.781977	0.479074
Random Forest	**Test**	0.927445	0.776596	0.744898	0.760417
	Train	1.000000	1.000000	1.000000	1.000000
Decision Trees	**Test**	0.883281	0.593750	0.775510	0.672566
	Train	1.000000	1.000000	1.000000	1.000000

Hyperparameter Tuning of Decision Trees Classifier

In [323]:
```
dt_params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"],
    'max_features': ["sqrt"],
    'min_samples_split': [6, 10, 14]}
```

In [324]:
```
decision_t_model2 = DecisionTreeClassifier()
decision_t_cv_model = GridSearchCV(decision_t_model2, dt_params, cv=3, n_jobs=
decision_t_cv_model.fit(X_train_resampled,y_train_resampled)
print("Best parameters:"+str(decision_t_cv_model.best_params_))
```

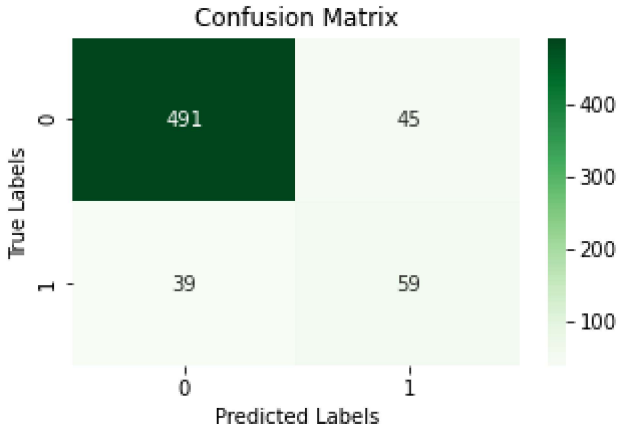
```
Best parameters:{'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sq
rt', 'min_samples_leaf': 5, 'min_samples_split': 6}
```

In [325]:
```
decision_t_model = DecisionTreeClassifier(criterion='entropy', max_depth=20, m
decision_t_model.fit(X_train_resampled,y_train_resampled)
y_pred_decision_t_model = decision_t_model.predict(X_test)
print(classification_report(y_test, y_pred_decision_t_model, target_names=['0'
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       536
           1       0.57      0.60      0.58        98

    accuracy                           0.87       634
   macro avg       0.75      0.76      0.75       634
weighted avg       0.87      0.87      0.87       634
```

In [326]:
```
print("HYPERPARAMETER TUNED DECISION TREE MODEL RESULTS")
print('Accuracy score for testing set: ',round(accuracy_score(y_test, y_pred_d
print('F1 score for testing set: ',round(f1_score(y_test, y_pred_decision_t_mo
print('Recall score for testing set: ',round(recall_score(y_test, y_pred_decis
print('Precision score for testing set: ',round(precision_score(y_test, y_pred
cm_rf = confusion_matrix(y_test, y_pred_decision_t_model)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Greens', fmt='g', ax=ax);
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
HYPERPARAMETER TUNED DECISION TREE MODEL RESULTS
Accuracy score for testing set:  0.86751
F1 score for testing set:  0.58416
Recall score for testing set:  0.60204
Precision score for testing set:  0.56731
```



Hyperparameter Tuning of Random Forest Classifier

We will use the cross validated GridSearchCV hyperparameter tuning technique

In [327]:
```
# Parameter grid
random_f_params = {"max_depth": [5,8,10],
              "max_features":[5,8,10],
              "n_estimators":[100,500],
              "min_samples_split":[5,10]}
```

Random Forest Model with GridSearchCV Applied

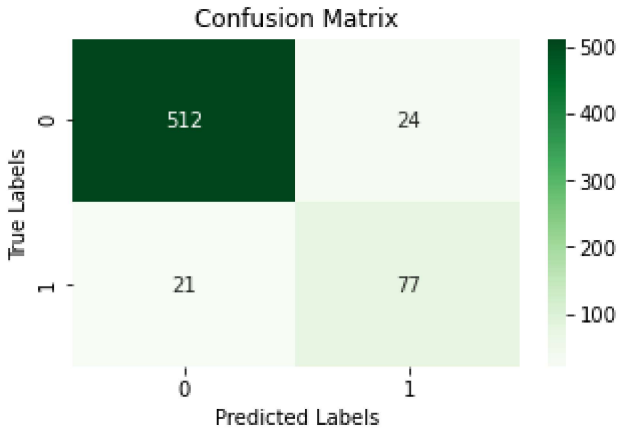
In [328]:
```
random_f_model_final = RandomForestClassifier(max_depth=10,max_features=10,min
random_f_model_final.fit(X_train_resampled,y_train_resampled)
y_pred_final = random_f_model_final.predict(X_test)
print(classification_report(y_test, y_pred_final, target_names=['0', '1']))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       536
           1       0.76      0.79      0.77        98

    accuracy                           0.93       634
   macro avg       0.86      0.87      0.87       634
weighted avg       0.93      0.93      0.93       634
```

Classification model summary

In [329]:
```
print("**************** HYPERPARAMETER TUNED RANDOM FOREST MODEL RESULTS *****
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_fi
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_final),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_final)
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_
cm_rf = confusion_matrix(y_test, y_pred_final)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Greens', fmt='g', ax=ax);
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
**************** HYPERPARAMETER TUNED RANDOM FOREST MODEL RESULTS **********
*****
Accuracy score for testing set:  0.92902
F1 score for testing set:  0.77387
Recall score for testing set:  0.78571
Precision score for testing set:  0.76238
```

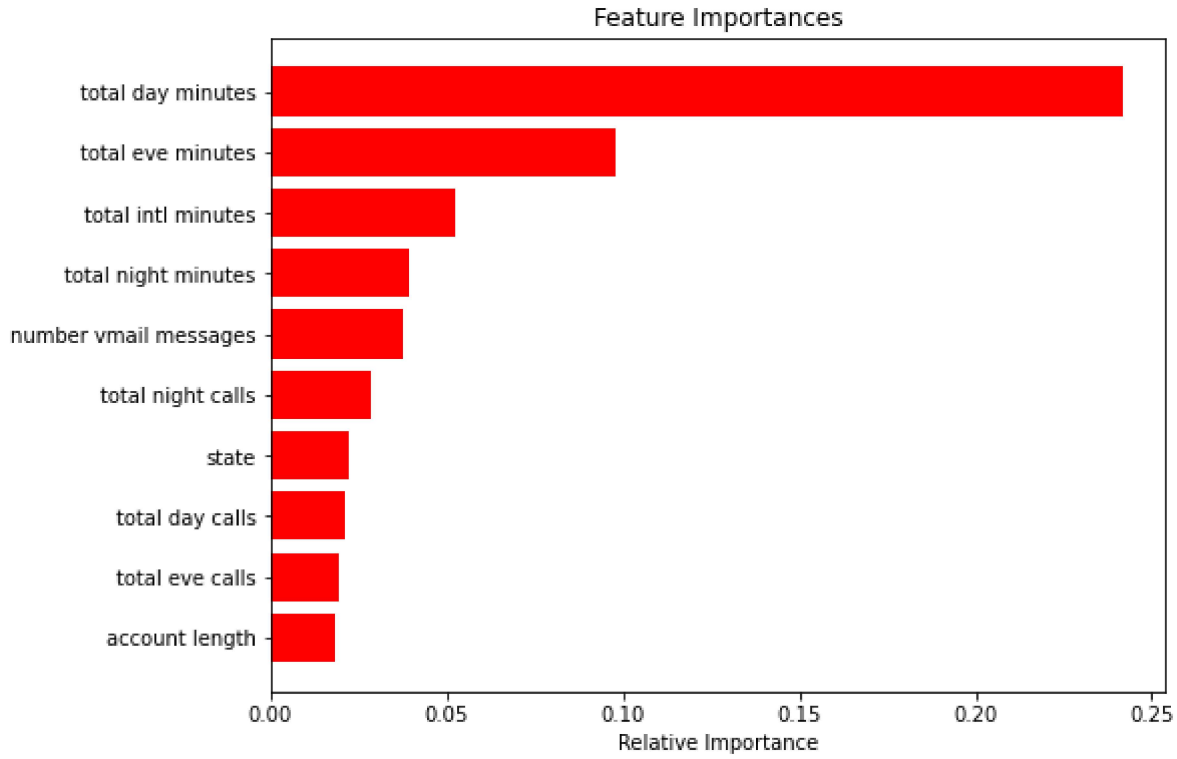


The tuned model has better perfomance with an accuracy of 93% from 92 , f1-score of 78.57% from 75.26% , recall of 78.57% from 74.49% and precision of 78%

Features

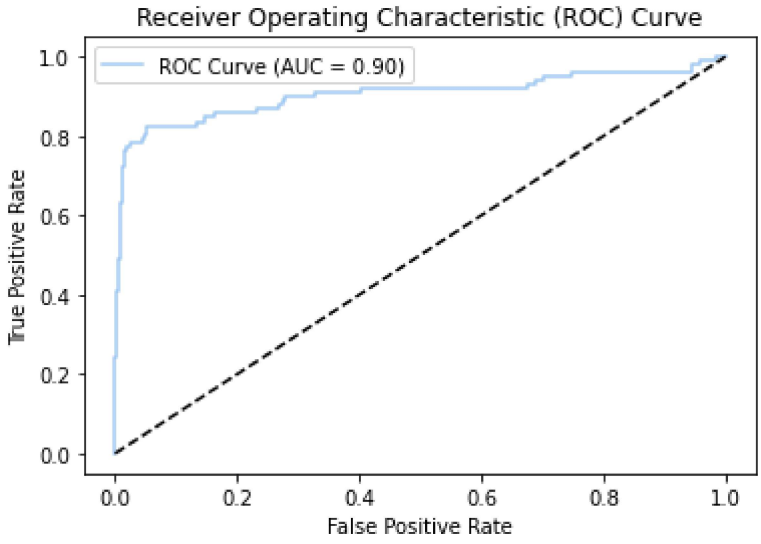
The chart below shows the top 10 features and their importance levels determined by the hyperparameter tuned Random Forest model. The importance values indicate the relative significance of each feature in predicting customer churn.

In [330]:
```
feature_names = list(X_train_resampled.columns)
importances = random_f_model_final.feature_importances_[0:10]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='red', align='center
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Tuned Models Evaluation

In [331]:
```
# Get the predicted probabilities for the positive class
y_proba = random_f_model_final.predict_proba(X_test)[:, 1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Compute the AUC score
auc_score = roc_auc_score(y_test, y_proba)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC Curve (AUC = {:.2f})'.format(auc_score ),)
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



CONCLUSION

The random forest classier performed the best with a recall of 79%, indicating that the model correctly identified around 79% of the actual positive cases, which isn't bad. And an acccuracy score of 93%.

A high f1-score of 78% shows a balanced evaluation of the model's accuracy in predicting both churn and non-churn customers.

Texas has the highest churn rate.

Based on the graph Total day minutes, total eveing minutes and total international minutes are the most important features when predicting churn.

RECOMMENDATIONS

Implement a strategy that could include giving discounts to international plans and voice mail subscriptions as we've seen those without these plans tend to churn. Many of them may see these plans as too expensive to purchase hence opting not to . This could help attract these niche customer base.

People with large total call minutes must be charged large amounts of money and may be dissatisfied with the charges hence decide to leave. To counteract this, we could introduce package deals to those who call for long periods of time.

Loyalty points / cards and deals for customers who have had their account for more than 3 yrs or more for example.

Improve Customer feedback - This could include conducting surveys of customers who have churned to understand why they left and the reasoning behind it. Thsi could also include improving customer service calls as they play a role in churn. Find out the most common issues that customers call about.

NEXT STEPS AND LIMITATIONS

Further investigate on what trends might be causing high churn in certain states such as texas and how we can prevent this.

There could be othr features that affect churn other than the ones privided in our dataset.

Explore other model classifiers that may provide better results than random forest classifier.