1. Business Understanding

Customer churn refers to the phenomenon where customers or subscribers end their relationship with a company or service provider. The primary objective of any business is to reduce customer churn. By identifying potential churners in advance, SyriaTel can take precaution measures to retain these customers.

Such measures may include:

- Improved customer service.
- Addressing the issues that may lead to churn.
- Making targeted offers and,
- Targeted market campaigns among others.

Churn can have great significant financial impact on the business as high churn leads to loss of recurring revenue, damage of brand's reputation among many other effects.

This project aims to predict customer churn by developing an algorithm to predict the churn rate based on customer usage pattern.

Objectives

- To determine attributes that contribute to customer churn.
- To build a classification model that predicts customer churn.
- To achieve a recall score for the model of at least 70%
- To make valid reccomendations to SyriaTel on ways they can reduce customer churn.

Analytical Questions

1. Which factors generally lead to customer churn?
2. Which is the most appropriate evaluation metric for the model I will build?
3. What reccommendations do I have for the stakeholder?

2. Data Understanding

> The dataset has customer usage pattern and whether the customer has churned or not. I will develop an algorithm to predict the churn score based on usage pattern. The predictors provided are as follows:

1. "state", string. 2-letter code of the US state of customer residence
2. "account_length", Number of months the customer has been with current telco provider
3. "area_code", a string 3 digit area code.
4. "international_plan", (yes/no). The customer has international plan.
5. "voice_mail_plan", (yes/no). The customer has voice mail plan.
6. "number_vmail_messages", numerical. Number of voice-mail messages.
7. "total_day_minutes", numerical. Total minutes of day calls.
8. "total_day_calls", numerical. Total minutes of day calls.
9. "total_day_charge", numerical. Total charge of day calls.
10. "total_eve_minutes", numerical. Total minutes of evening calls.

11. "total_eve_calls", numerical. Total number of evening calls.
12. "total_eve_charge", numerical. Total charge of evening calls.
13. "total_night_minutes", numerical. Total minutes of night calls.
14. "total_night_calls", numerical. Total number of night calls.
15. "total_night_charge", numerical. Total charge of night calls.
16. "total_intl_minutes", numerical. Total minutes of international calls.
17. "total_intl_calls", numerical. Total number of international calls.
18. "total_intl_charge", numerical. Total charge of international calls
19. "number_customer_service_calls", numerical. Number of calls to customer service

Target Variable is:

- Churn: if the customer has churned (1=yes; 0 = no)

Data Exploration and Cleaning

In [1]:

```
# Importing libraries.
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import  accuracy_score, recall_score, precision_score, f1_sc
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
```

2.1 Loading the dataset

In [2]:

```
# Creating a dataframe and viewing the first 5 columns.
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df.head()
```

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

2.2 Statistical Analysis.

In [3]:

```
# A function to analyze the shape, number of columns, and information of the data
def analyze_dataset(filename):
    """
    This function  outputs information about the shape,
    columns, and information of the dataset using the Pandas library.
    """
    # Output the shape of the dataset
    print("Shape of dataset:", df.shape)
    print('\n-----------------------------------------------------------')

    # Output the column names of the dataset
    print("Column names:", list(df.columns))
    print('\n-----------------------------------------------------------')

    # Output information about the dataset
    print(df.info())
    print('\n-----------------------------------------------------------')

    # output descriptive statistics about the dataset
    print(df.describe())
    print('\n-----------------------------------------------------------')

    # output if the dataset has duplicates
    print("Number of duplicates: ",df.duplicated().sum())
```

In [4]:

```
# Applying the function to analyze our dataset
analyze_dataset('bigml_59c28831336c6604c800002a.csv')
```

```
Shape of dataset: (3333, 21)

-----------------------------------------------------------
Column names: ['state', 'account length', 'area code', 'phone numbe
r', 'international plan', 'voice mail plan', 'number vmail message
s', 'total day minutes', 'total day calls', 'total day charge', 'tot
al eve minutes', 'total eve calls', 'total eve charge', 'total night
minutes', 'total night calls', 'total night charge', 'total intl min
utes', 'total intl calls', 'total intl charge', 'customer service ca
lls', 'churn']

-----------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
None

-----------------------------------------------------------
       account length    area code  number vmail messages  total day
minutes  \
count     3333.000000  3333.000000            3333.000000        333
3.000000
mean       101.064806   437.182418               8.099010         17
9.775098
std         39.822106    42.371290              13.688365          5
4.467389
min          1.000000   408.000000               0.000000
0.000000
25%         74.000000   408.000000               0.000000         14
3.700000
50%        101.000000   415.000000               0.000000         17
9.400000
75%        127.000000   510.000000              20.000000         21
6.400000
max        243.000000   510.000000              51.000000         35
0.800000
```

```
       total day calls  total day charge  total eve minutes  total e
ve calls  \
count      3333.000000       3333.000000        3333.000000      333
3.000000
mean        100.435644         30.562307         200.980348       10
0.114311
std          20.069084          9.259435          50.713844        1
9.922625
min           0.000000          0.000000           0.000000
0.000000
25%          87.000000         24.430000         166.600000        8
7.000000
50%         101.000000         30.500000         201.400000       10
0.000000
75%         114.000000         36.790000         235.300000       11
4.000000
max         165.000000         59.640000         363.700000       17
0.000000

       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
75%             3.270000                2.000000
max             5.400000                9.000000

-----------------------------------------------------------
Number of duplicates:  0
```

From the analysis of our dataset:

- There are 3333 rows and, 21 columns.
- There are no missing values.
- There are both categorical and numeric features.
- The numeric features are not all in the same scale.
- There are no duplicates in the dataset.

In [5]:

```
# Creating a dataframe to display datatypes and, the unique values.
desc = []
for i in df.columns:
    desc.append([
        i,
        df[i].dtypes,
        df[i].nunique(),
    ])

pd.DataFrame(data = desc, columns=['Feature','Dtypes','Sample_Unique'])
```

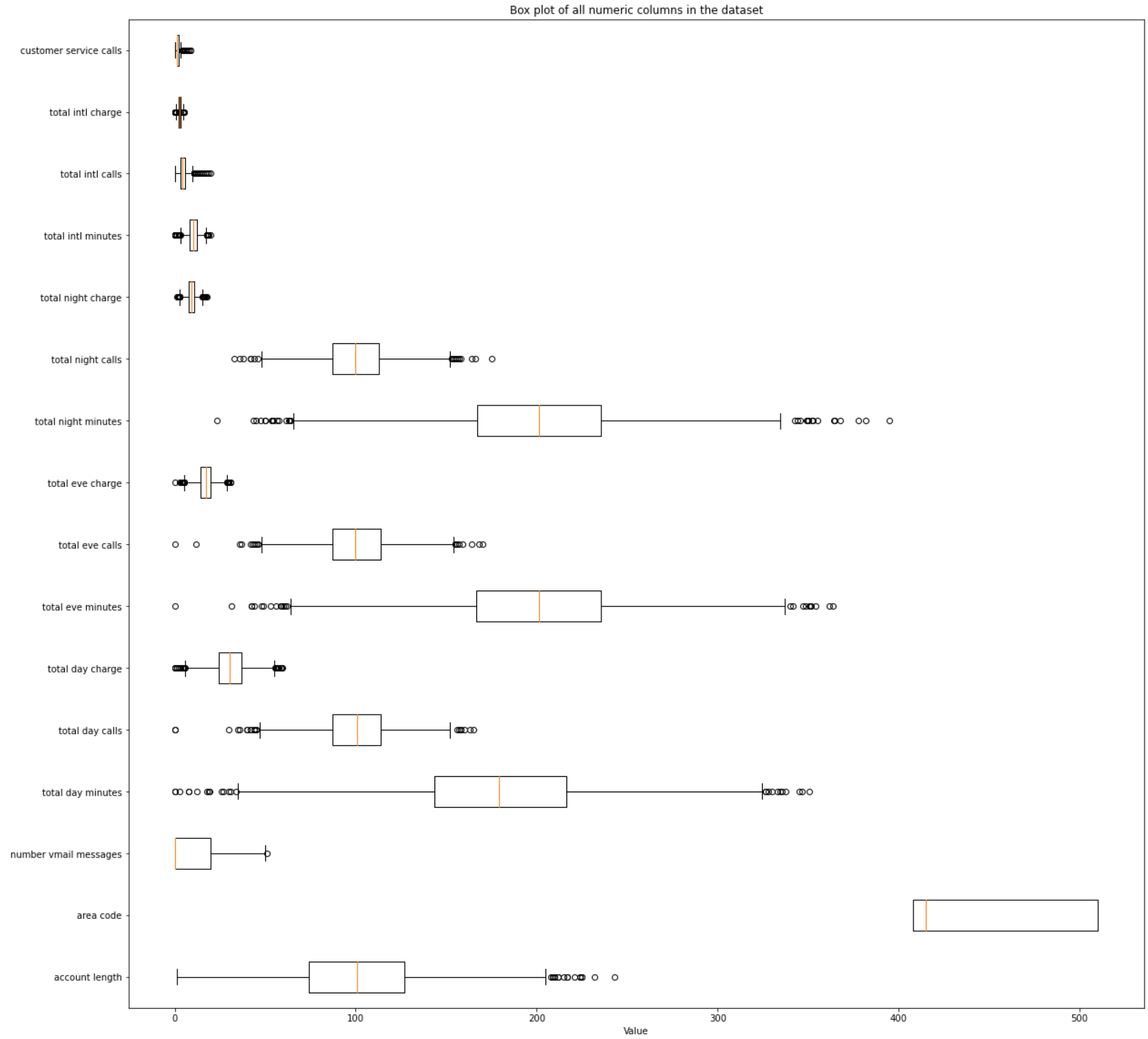
Out[5]:

	Feature	Dtypes	Sample_Unique
0	state	object	51
1	account length	int64	212
2	area code	int64	3
3	phone number	object	3333
4	international plan	object	2
5	voice mail plan	object	2
6	number vmail messages	int64	46
7	total day minutes	float64	1667
8	total day calls	int64	119
9	total day charge	float64	1667
10	total eve minutes	float64	1611
11	total eve calls	int64	123
12	total eve charge	float64	1440
13	total night minutes	float64	1591
14	total night calls	int64	120
15	total night charge	float64	933
16	total intl minutes	float64	162
17	total intl calls	int64	21
18	total intl charge	float64	162
19	customer service calls	int64	10
20	churn	bool	2

Checking for Outliers.

In [6]:

```
# plotting a boxplot of each column to view outliers.

numeric_columns = df.select_dtypes(include=['int64', 'float64'])
plt.figure(figsize=(20, 20))
plt.boxplot(numeric_columns.values, vert=False)  # Show boxplots horizontally
plt.title("Box plot of all numeric columns in the dataset")
plt.xlabel("Value")
plt.yticks(range(1, len(numeric_columns.columns) + 1), numeric_columns.columns)
plt.show()
```



Outliers can affect the predictive power of our models,therefore removing them will increase our model performance.

In [7]:

```
# Removing outliers
for col in numeric_columns:
    q1 = df[col].quantile(0.20)
    q3 = df[col].quantile(0.80)
    iqr = q3 - q1
    range_low = q1 - 1.5 * iqr
    range_high = q3 + 1.5 * iqr
    # Filtering the dataset in-place
    df = df.loc[(df[col] > range_low) & (df[col] < range_high)]

data = df
data.shape
```

Out[7]:

```
(3218, 21)
```

3. Univariate analysis

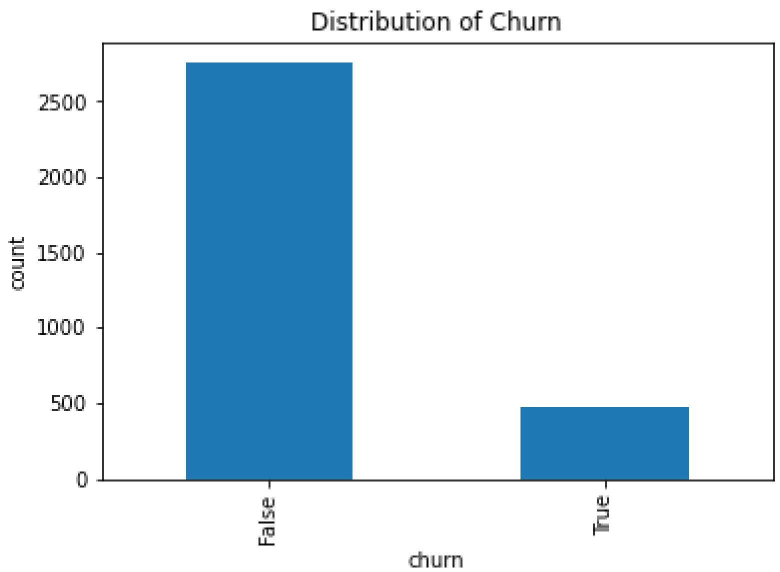
Exploring some of the variables in the data set looking for patterns of response to the variable.

In [8]:

```
# plot a bar graph to show the distribution of churn
data['churn'].value_counts().plot(kind='bar')
plt.xlabel('churn')
plt.ylabel('count')
plt.title('Distribution of Churn')
data['churn'].value_counts(normalize=True)
```

Out[8]:

```
False    0.85519
True     0.14481
Name: churn, dtype: float64
```



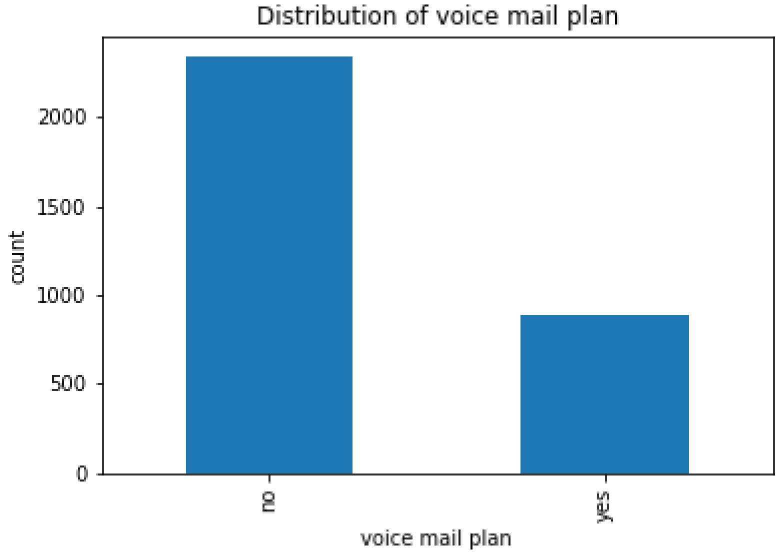
The bar graph above shows the distribution of churn where we see 14% of the customers churned.

In [9]:

```
# Distribution of voice mail plan

data['voice mail plan'].value_counts().plot(kind='bar')
plt.xlabel('voice mail plan')
plt.ylabel('count')
plt.title('Distribution of voice mail plan')
data['voice mail plan'].value_counts(normalize=True)
```

Out[9]:

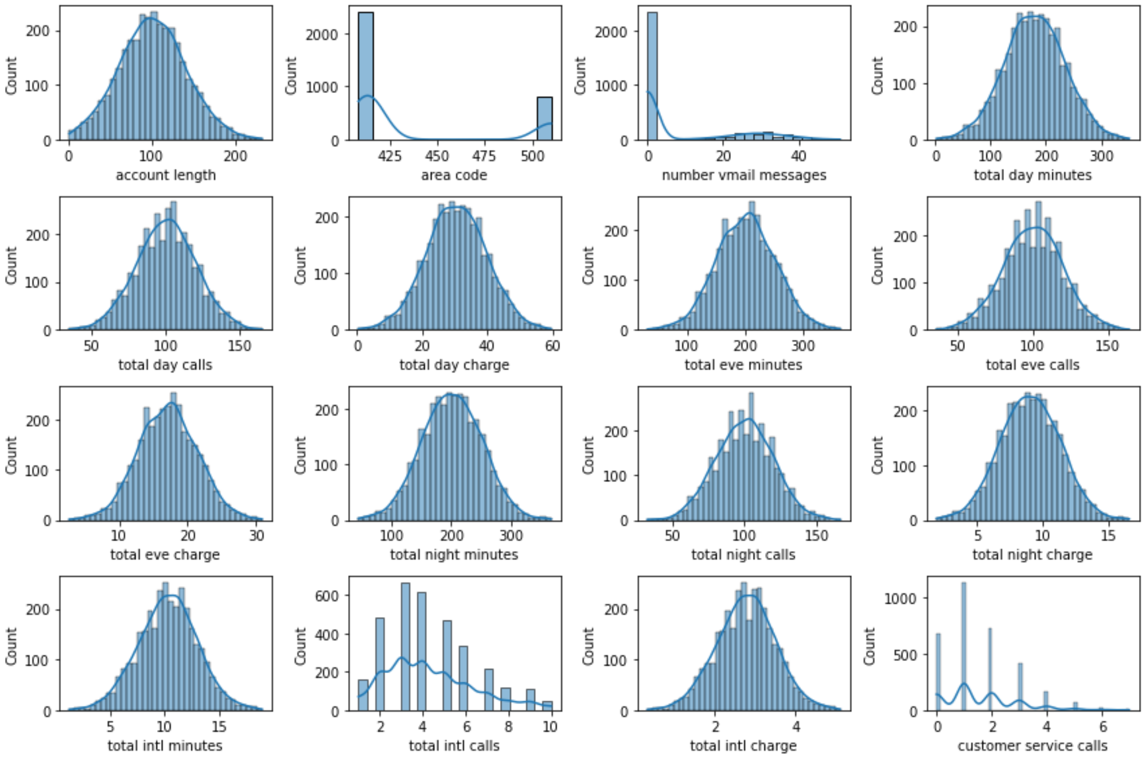
```
no     0.725606
yes    0.274394
Name: voice mail plan, dtype: float64
```



From the plot above we see that most customers (73%) did not have a voice mail plan.

In [10]:

```
# Checking the distribution of columns.
plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(4, 4, i)
    sns.histplot(df[col], kde=True)
    plt.xlabel(col)
plt.tight_layout()
plt.show();
```



As seen from the distributions above most of the numeric features are normally distributed and the features will be good for modelling.

3.2 Bivariate analysis.

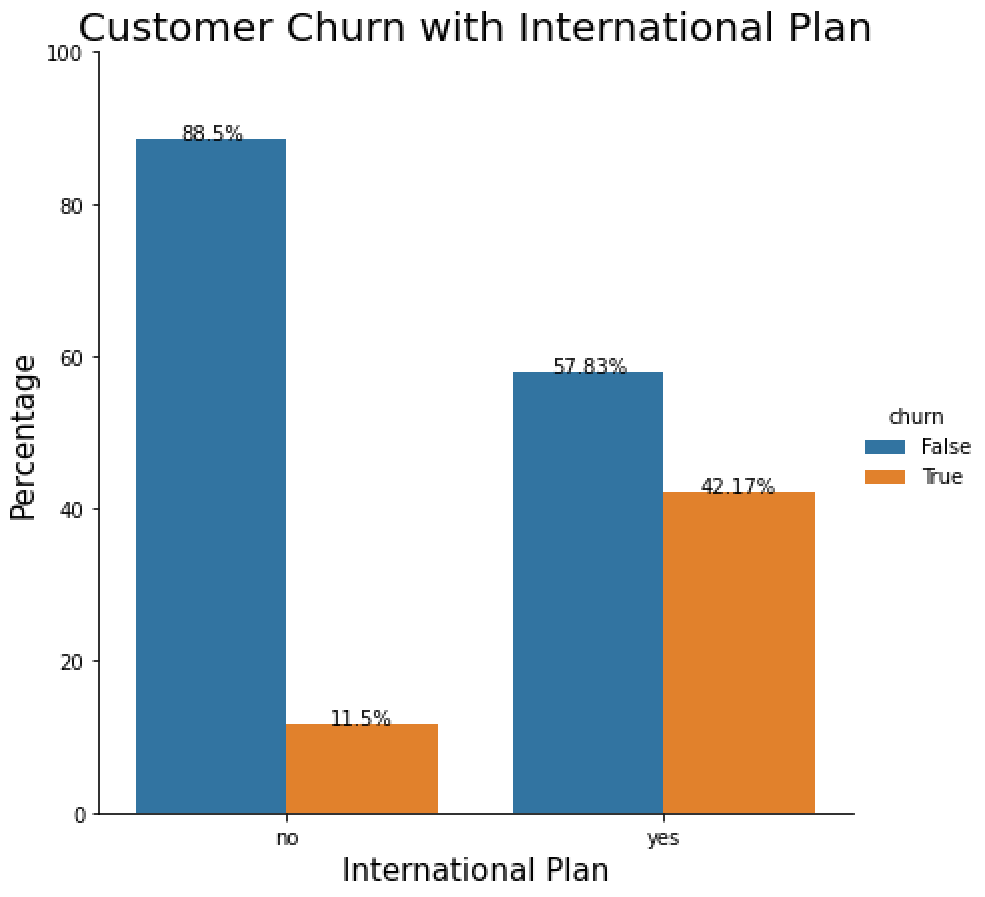
In this analysis, we will be examining the bivariate relationships between the features we have in relation to churn.

In [11]:

```
# A function that plots a feature in relation to churn.
def plot_churn_with_international_plan(data, x, y):
    df1 = data.groupby(x)['churn'].value_counts(normalize=True)
    df1 = df1.mul(100)
    df1 = df1.rename('Percentage').reset_index()

    ax = sns.catplot(x=x, y='Percentage', hue=y, kind='bar', data=df1, height=6)
    ax.set(ylim=(0, 100))

    for p in ax.ax.patches:
        txt = str(p.get_height().round(2)) + '%'
        txt_x = p.get_x() + p.get_width() / 2
        txt_y = p.get_height()
        ax.ax.text(txt_x, txt_y, txt, ha='center')
```

In [12]:

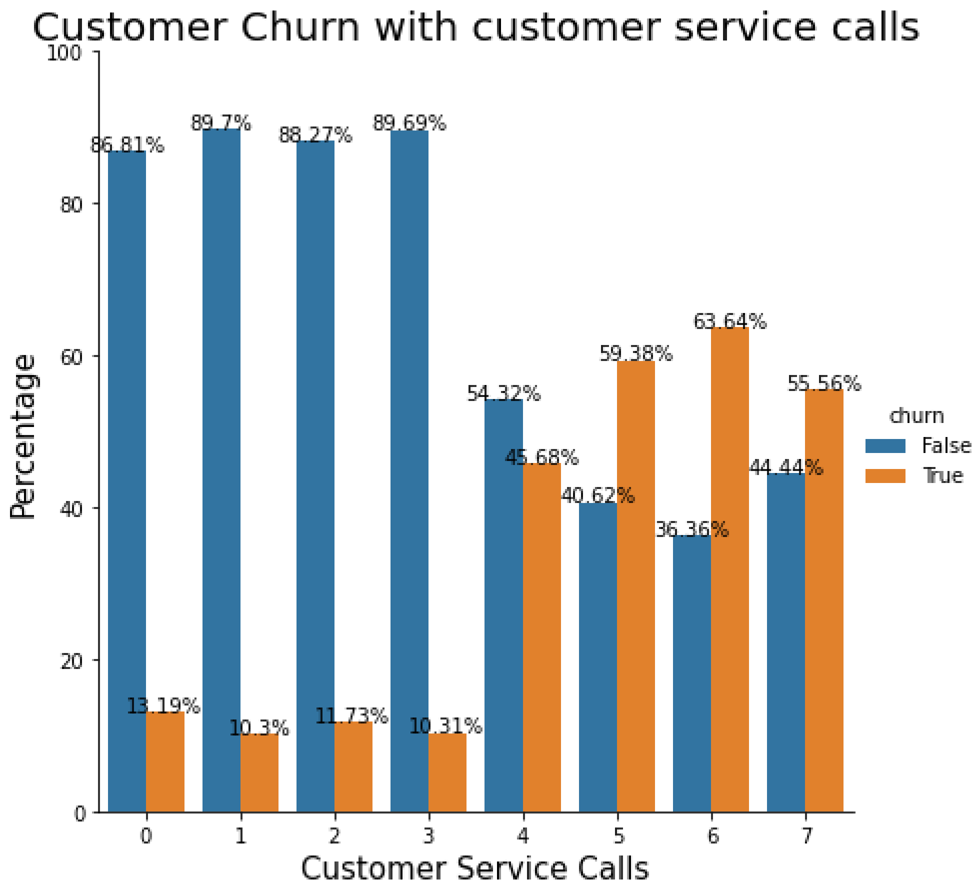
```
# A bar plot of international plan with churn
plot_churn_with_international_plan(data, 'international plan', 'churn')
plt.title('Customer Churn with International Plan', fontsize=20)
plt.xlabel('International Plan', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.show()
```



From the plot above the higher percentage of the customers who churned had an international plan.

In [13]:

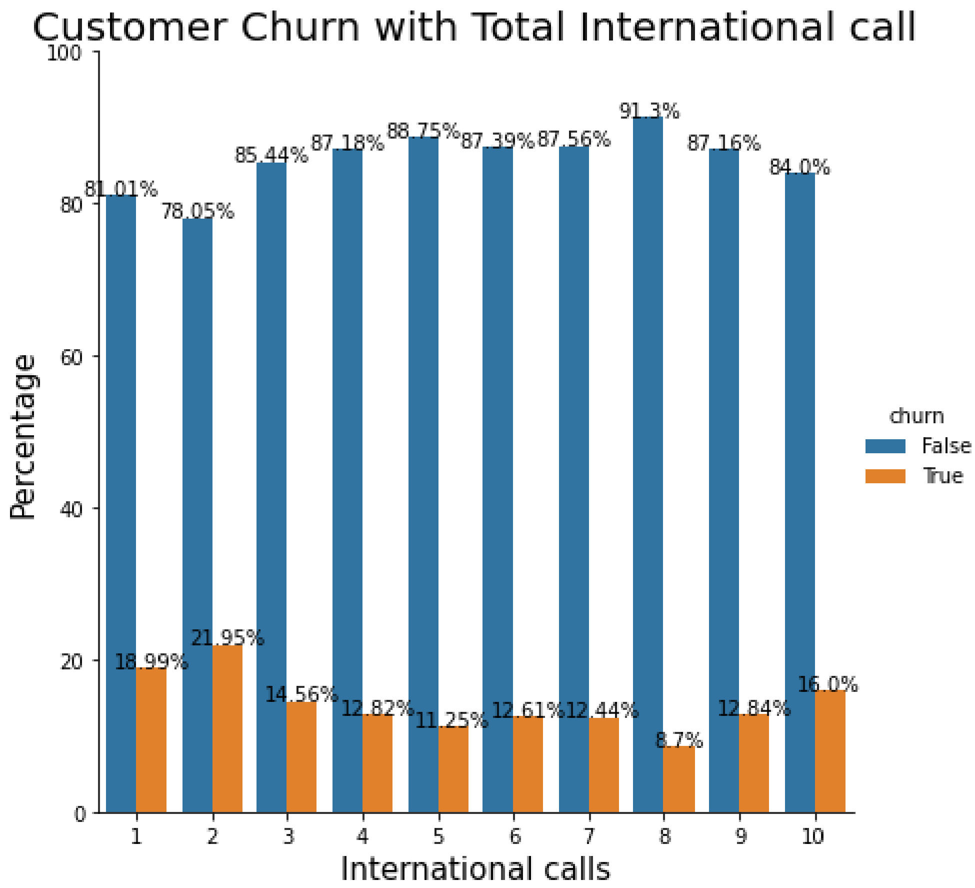
```
# A bar plot of total day minute with churn
plot_churn_with_international_plan(data, 'customer service calls', 'churn')
plt.title('Customer Churn with customer service calls', fontsize=20)
plt.xlabel('Customer Service Calls', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.show()
```



From the plot above, a customer who had more customer service call are most likely to churn. This implies the costomers who did not have a customer service call are most likely satisfied from the services offered by the company.

In [14]:

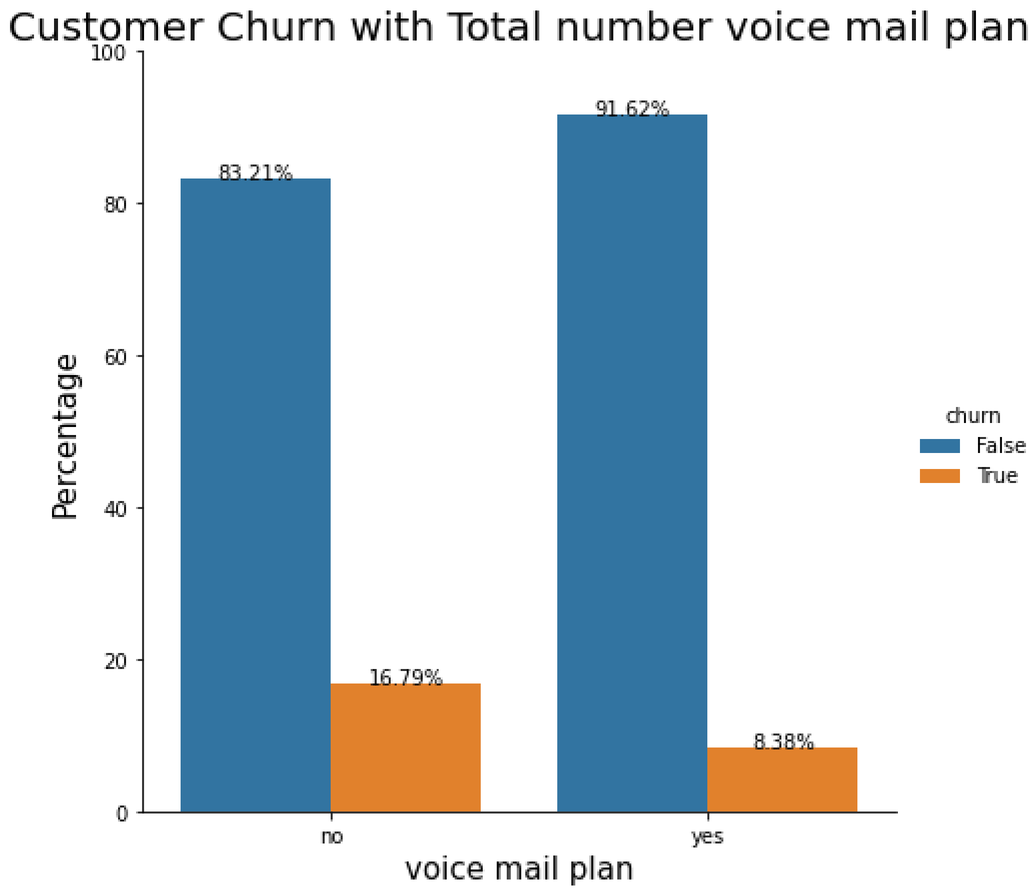
```
# A bar plot of international calls with churn
plot_churn_with_international_plan(data, 'total intl calls', 'churn')
plt.title('Customer Churn with Total International call', fontsize=20)
plt.xlabel('International calls', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.show()
```



From the plot high number of international calls contributed to low churn rate.

In [15]:

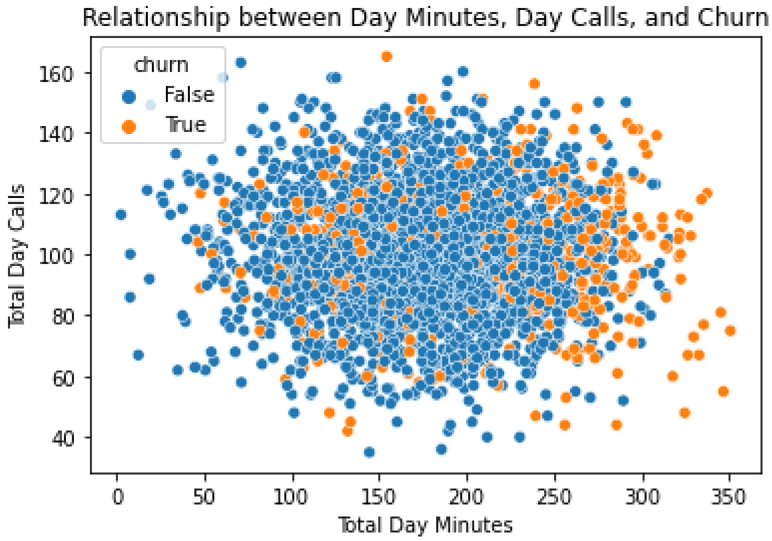
```
# A bar plot of number vmail messages with churn
plot_churn_with_international_plan(data, 'voice mail plan', 'churn')
plt.title('Customer Churn with Total number voice mail plan', fontsize=20)
plt.xlabel('voice mail plan', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.show()
```



From the plot above customers with a voice mail plan have a low churn rate.

In [16]:

```
# A scatter plot to show relationship between day minute and day calls with churn
sns.scatterplot(data=data, x='total day minutes', y='total day calls', hue='churn
plt.xlabel('Total Day Minutes')
plt.ylabel('Total Day Calls')
plt.title('Relationship between Day Minutes, Day Calls, and Churn')
plt.show()
```



From the scatter plot above high total day minutes potentially leads to high churn rate.

4. Data Preprocessing

The steps for our data preprocessing include:

- Remove unnecessary columns ('Phone number')
- Check for correlation using heatmaps.
- Convert object binary columns into binary encoding of 0's and 1's.
- One-hot encode categorical columns.
- Store the target column, 'churn', in a separate variable and remove it from the dataframe
- Split the data into training and test sets.

we can first drop the features that will not be useful for the model, Phone number is a unique identifier of a customer and it is not requred in our model.

In [17]:

```
# selecting the necessary columns for the model.
data = data.drop('phone number', axis=1)
```

In [18]:

```
# converting into binary columns.
data['international plan'] = df['international plan'].map({'yes':1, 'no':0})
data['voice mail plan'] = df['voice mail plan'].map({'yes':1, 'no':0})
data.head()
```

Out[18]:

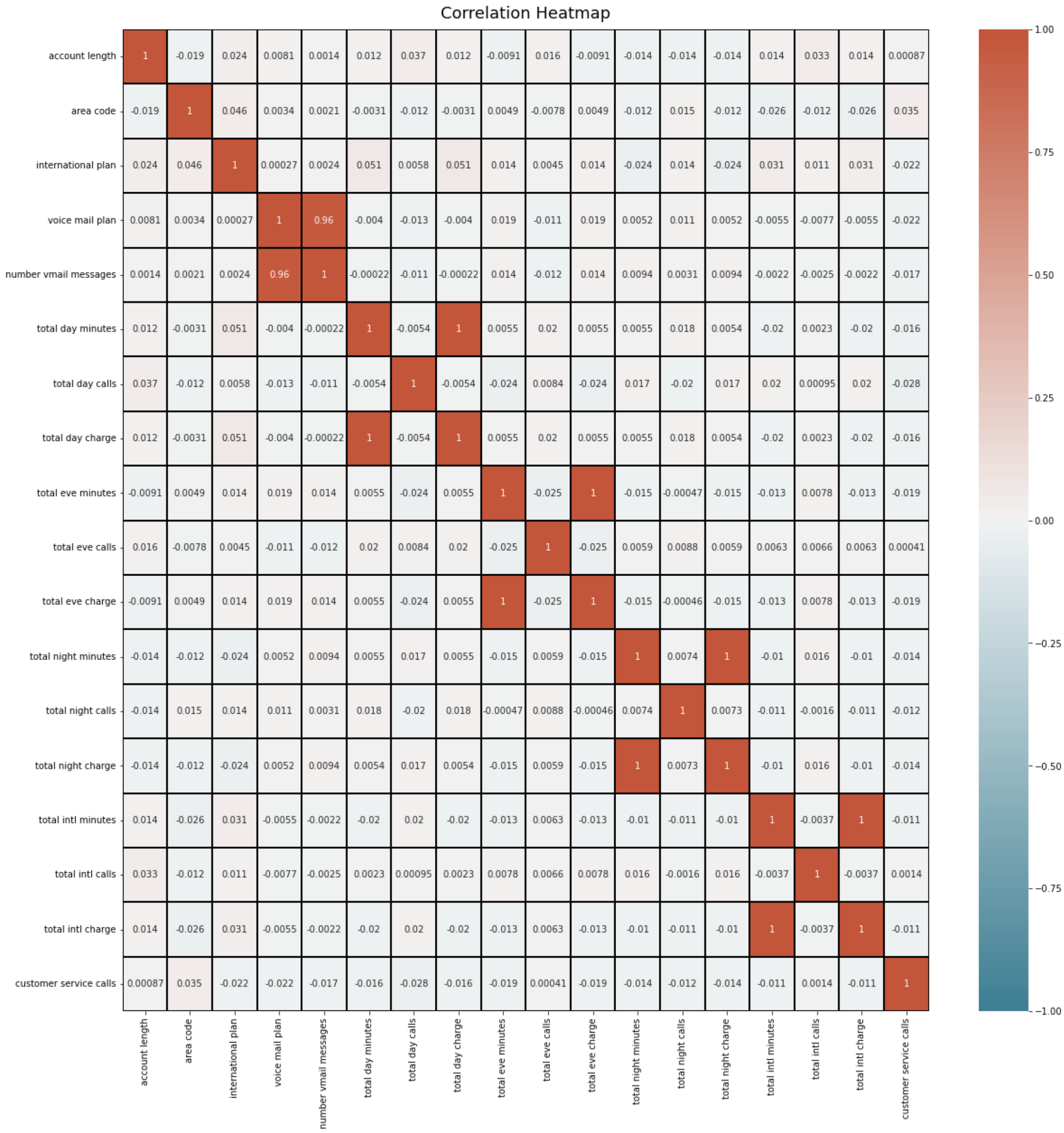
	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	tot e cal
0	KS	128	415	0	1	25	265.1	110	45.07	197.4	!
1	OH	107	415	0	1	26	161.6	123	27.47	195.5	1(
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2	1:
3	OH	84	408	1	0	0	299.4	71	50.90	61.9	{
4	OK	75	415	1	0	0	166.7	113	28.34	148.3	1:

In [19]:

```
# Assign the 'churn' column to labels
labels = data.churn

# Drop the 'churn' column from the dataframe.
predictors = data.drop('churn',axis=1)
```

In [20]:

```
# a heatmap that shows correlation.
plt.figure(figsize=(20, 20))
cmap = sns.diverging_palette(220,20,n=200)
heatmap = sns.heatmap(
    predictors.corr(),vmin=-1, vmax=1,center = 0,
    annot=True,cmap=cmap,linewidths=2, linecolor='black')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12);
```



In [21]:

```
# correlations with churn
data.corr().churn.sort_values(ascending = False)
```

Out[21]:

```
churn                     1.000000
international plan        0.258294
total day minutes         0.206320
total day charge          0.206318
customer service calls    0.201696
total eve minutes         0.093162
total eve charge          0.093151
total intl charge         0.056676
total intl minutes        0.056659
total night charge        0.037257
total night minutes       0.037243
account length            0.016708
total day calls           0.015308
area code                 0.010032
total night calls         0.007787
total eve calls           0.004030
total intl calls         -0.070657
number vmail messages    -0.094225
voice mail plan          -0.106604
Name: churn, dtype: float64
```

Here we see that all minutes and charge features are perfectly correlated (r = 1). This implies charge is usually based on minutes.

In [22]:

```
# importing  OneHotEncoder
from sklearn.preprocessing import OneHotEncoder

# One-hot encode categorical columns.
ohe = OneHotEncoder(sparse = False, handle_unknown = "ignore")

# fit ohe on small train data
ohe.fit(predictors[['state']])

# access the column names of the states
col_names = ohe.categories_[0]

# make a df with encoded states
train_state_encoded = pd.DataFrame(ohe.transform(predictors[["state"]]),
                               index = predictors.index,
                               columns = col_names)

# combine encoded features
predictors = pd.concat([predictors.drop("state", axis = 1), train_state_encoded],
```

In [23]:

```
# Split the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(predictors,labels, test_size 
```

In [24]:

```
X_train.head()
```

Out[24]:

	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
2401	126	415	1	0	0	239.7	87	40.75	281.7	92
991	50	415	0	1	35	192.6	97	32.74	135.2	101
2725	51	408	0	0	0	169.3	111	28.78	139.5	69
778	115	415	0	1	26	170.5	107	28.99	217.2	77
235	139	510	0	0	0	134.4	106	22.85	211.3	98

5 rows × 69 columns

since now all the columns are numeric we can proceed and create a logistic baseline model

5. Modeling

At this step I will build three models to and evaluate their performance. I will then choose the model that has the best perfomance and tune it to aiming to get better performance on the predictions made.

The models that I will try out are:

1.Logistic Regression

2.Decision Trees

3.Random Forest

4.Final Tuned model.

5.1 Logistic Regression

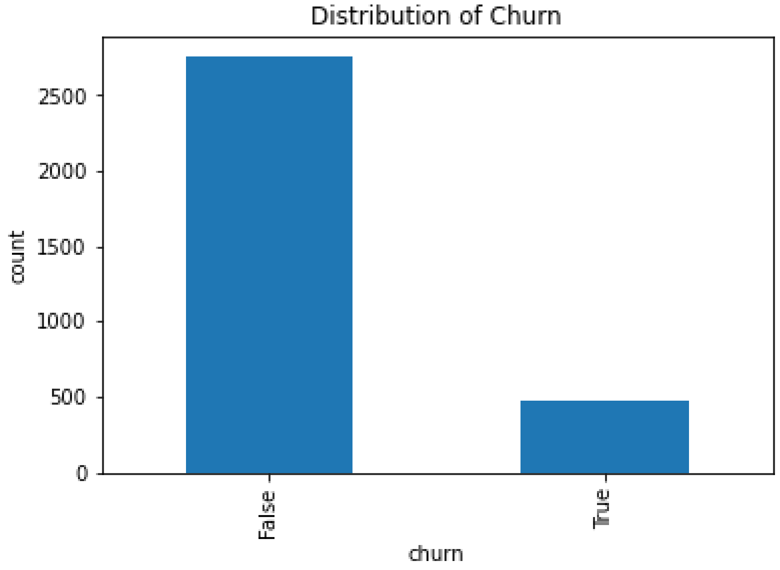
From the analysis it was evident that there was a class imbalance. Because of class the imbalance, we should add some kind of resampling step. Specifically we'll use SMOTE from imblearn.

In [25]:

```
# checking for imbalance
data['churn'].value_counts().plot(kind='bar')
plt.xlabel('churn')
plt.ylabel('count')
plt.title('Distribution of Churn')
data['churn'].value_counts(normalize=True)
```

Out[25]:

```
False    0.85519
True     0.14481
Name: churn, dtype: float64
```



Since the features are on different scale we can first scale and then fit then use a resampling method on the class imbalance.

In [26]:

```
# Scale X_train and X_test using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Ensure X_train and X_test are scaled DataFrames
X_train1 = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test1 = pd.DataFrame(X_test_scaled, columns=X_train.columns)

# Fit SMOTE to training data
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train_scaled, y_tra
```

In [27]:

```
# Fit a model
logreg = LogisticRegression(fit_intercept=False, solver='liblinear')
model_log = logreg.fit(X_train_resampled, y_train_resampled)
```

Evaluation.

For evaluation we use Recall as it measures the proportion of correctly predicted positive instances out of all actual positive instances. Recall focuses on the model's ability to correctly identify positive instances and is useful when the cost of false negatives is high.

In [28]:

```
# Evaluating Logistic regression model.
print(f'Train Accuracy: {accuracy_score(y_train, model_log.predict(X_train1))}')
print(f'Test Accuracy:  {accuracy_score(y_test, model_log.predict(X_test1))}')
print('-----------------------------------')
print(f'Train Recall: {recall_score(y_train, model_log.predict(X_train1))}')
print(f'Test Recall:  {recall_score(y_test, model_log.predict(X_test1))}')
```

```
Train Accuracy: 0.641110650642354
Test Accuracy:  0.6285714285714286
------------------------------------
Train Recall: 0.8854748603351955
Test Recall:  0.8055555555555556
```

From the above evaluation metrics the accuracy score is not satisfactory but the recall score is fairly much better. We can try to use another model, see if it improves in metrics.

5.2 Decision Trees

In [29]:

```
# Instantiate and fit a DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth = 5, random_state=4
tree_clf.fit(X_train,y_train)
```

Out[29]:

```
DecisionTreeClassifier(max_depth=5, random_state=42)
```

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

In [30]:

```
# Evaluation of model accuracy and recall.
print(f'Train Accuracy: {accuracy_score(y_train, tree_clf.predict(X_train))}')
print(f'Test Accuracy:  {accuracy_score(y_test, tree_clf.predict(X_test))}')
print('---------------------------------')

print(f'Train Recall: {recall_score(y_train, tree_clf.predict(X_train))}')
print(f'Test Recall:  {recall_score(y_test, tree_clf.predict(X_test))}')
```

```
Train Accuracy: 0.9481972648155823
Test Accuracy:  0.9341614906832298
--------------------------------
Train Recall: 0.7458100558659218
Test Recall:  0.7222222222222222
```

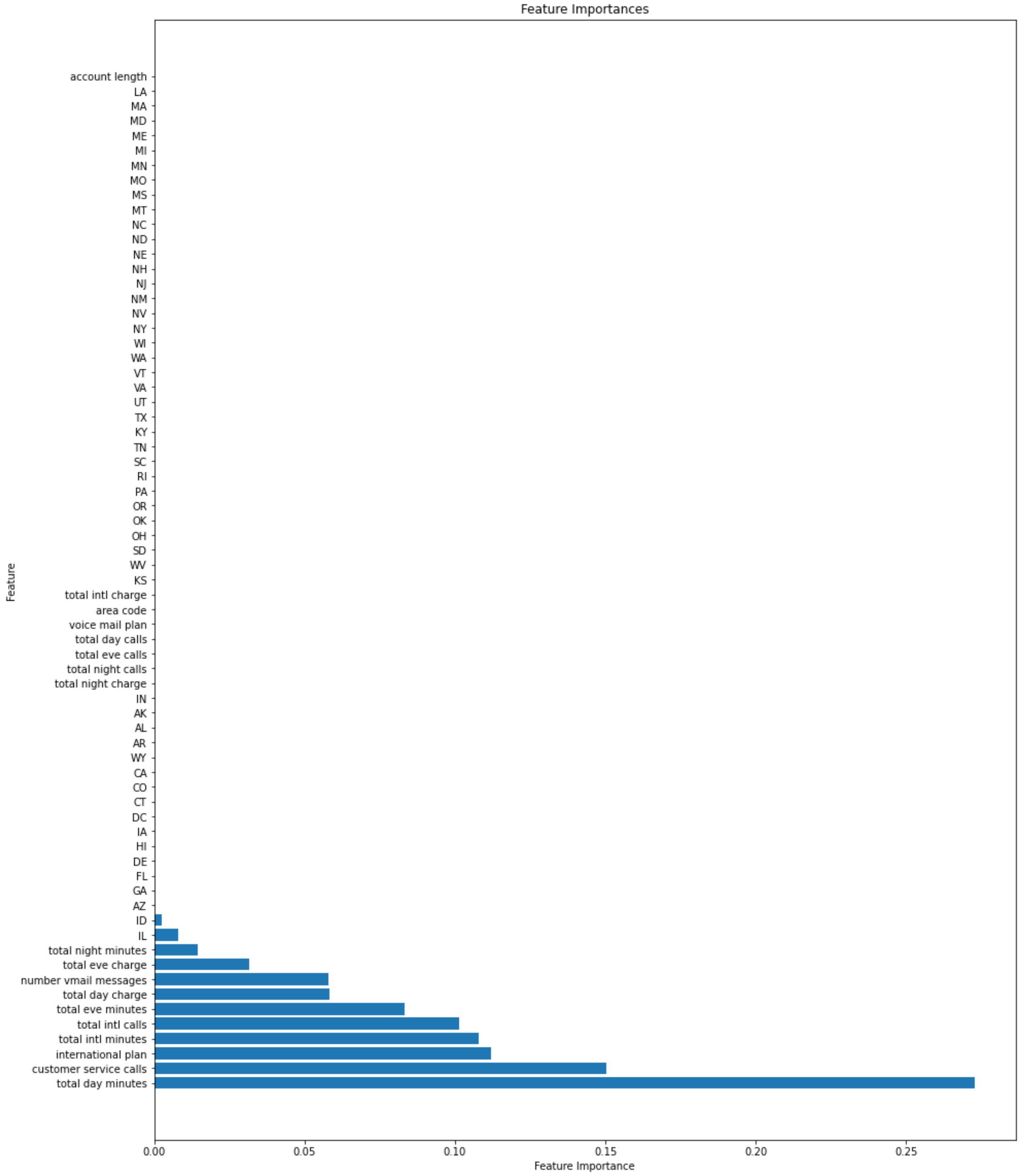
From the evaluation of the Decision tree the accuracy score is very good and Recall is fairly good we can fit yet another model to improve recall score.

Feature importance.

we can examine to see which features were important in our decision tree model

In [31]:

```
# Defining a function that plots the most important features.
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    sorted_idx = np.argsort(model.feature_importances_)[::-1]  # Sort feature imp
    plt.figure(figsize=(15, 20))
    plt.barh(range(n_features), model.feature_importances_[sorted_idx], align='ce
    plt.yticks(np.arange(n_features), X_train.columns.values[sorted_idx])
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importances')
    plt.show()

plot_feature_importances(tree_clf)
```



Here as we can see from our model the 5 most important features are:

1. Total day minutes.

2. Customer service calls.
3. Total day charge.
4. Total evening charge.
5. Total international calls.

5.3 Random Forest.

In [32]:

```
# Instantiate and fit a RandomForestClassifier
forest = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
forest.fit(X_train,y_train)
```

Out[32]:

```
RandomForestClassifier(max_depth=5, random_state=42)
```

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

In [33]:

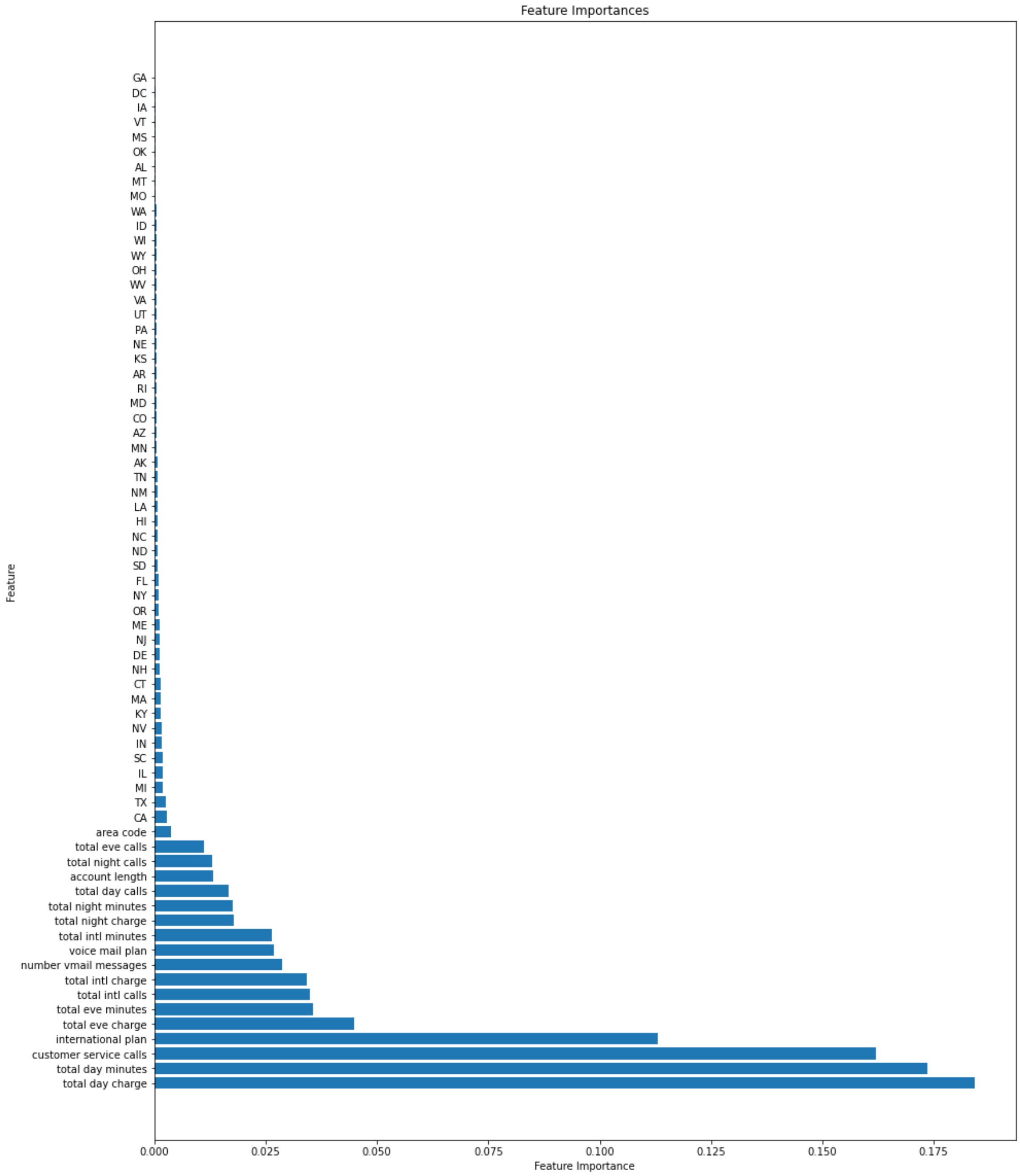
```
# Evaluation.
print(f'Train Accuracy: {accuracy_score(y_train, forest.predict(X_train))}')
print(f'Test Accuracy:  {accuracy_score(y_test, forest.predict(X_test))}')
print('--------------------------------')

print(f'Train Recall: {recall_score(y_train, forest.predict(X_train))}')
print(f'Test Recall:  {recall_score(y_test, forest.predict(X_test))}')
```

```
Train Accuracy: 0.886033982594281
Test Accuracy:  0.8832298136645963
--------------------------------
Train Recall: 0.23184357541899442
Test Recall:  0.12962962962962962
```

The model performed very poorly on recall and the accuracy is a bit low as compared to decision tree, Tuning this model can potentially improve the metrics. so we are going to try hyperparameter tuning using grid search.

In [34]:

```
plot_feature_importances(forest)
```



from our Random forest model the 5 most important features are:

1. Total day minutes.
2. Total day charge.
3. Customer service calls.
4. Internatinal plan
5. Total evening charge.

Hyperparameter Tuning

we can perform a grid search on the forest to identify the best parameters to use for our model

In [35]:

```
# Defining a parameter grid
dt_param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth": [None, 2, 3, 4, 5, 6],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 3, 4, 5, 6],
}
```

In [36]:

```
from sklearn.model_selection import GridSearchCV
# Instantiate GridSearchCV
dt_grid_search = GridSearchCV(tree_clf, dt_param_grid, cv=3, return_train_score=T

# Fit to the data
dt_grid_search.fit(X_train, y_train)
```

Out[36]:

```
GridSearchCV(cv=3,
             estimator=DecisionTreeClassifier(max_depth=5, random_st
ate=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 2, 3, 4, 5, 6],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6],
                         'min_samples_split': [2, 5, 10]},
             return_train_score=True)
```

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

In [37]:

```
# Mean training score
dt_gs_training_score = np.mean(dt_grid_search.cv_results_["mean_train_score"])

# Mean test score
dt_gs_testing_score = dt_grid_search.score(X_test, y_test)

print(f"Mean Training Score: {dt_gs_training_score :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score :.2%}")
print("Best Parameter Combination Found During Grid Search:")
print('-------------------------')

print(f'Train Recall: {recall_score(y_train, dt_grid_search.predict(X_train))}')
print(f'Test Recall:  {recall_score(y_test, dt_grid_search.predict(X_test))}')

dt_grid_search.best_params_
```

```
Mean Training Score: 93.54%
Mean Test Score: 95.03%
Best Parameter Combination Found During Grid Search:
-------------------------
Train Recall: 0.7849162011173184
Test Recall:  0.7222222222222222
```

Out[37]:

```
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_leaf': 3,
 'min_samples_split': 10}
```

Model Summary

I used three classification models to predict churn and a more tuned model to improve the evaluation metrics i.e:

- Logistic regression which had a: -Train Accuracy: 65% -Test Accuracy: 63% -Train Recall: 89% -Test Recall: 80%
- Decision Tree which had a: -Train Accuracy: 95% -Test Accuracy: 93% -Train Recall: 75% -Test Recall: 72%
- Random Forest with a: -Train Accuracy: 89% -Test Accuracy: 88% -Train Recall: 23% -Test Recall: 12%
- Tuned Random Forest wth a: -Train Accuracy: 94% -Test Accuracy: 95% -Train Recall: 78% -Test Recall: 72% The best paramters for random forest are criterion 'entropy', Maximum depth of 6, minimum sample leaf of 3 and a minimum sample split of 10

Conclusion

The final model that will be used to predict customer churn is Random Forest with the tuned hyperparameters (criterion 'entropy', Maximum depth of 6, minimum sample leaf of 3 and a minimum sample split of 10) as it has the least number of false negatives with a better recall score. The most most important features that lead to customer churn are:

- Total day minutes.
- Total day charge.
- Customer service calls.

- Internatinal plan.
- Total evening charge.

Recommendations.

From the findings

- A customer who had more customer service call are most likely to churn. This implies the costomers who did not have a customer service call are most likely satisfied from the services offered by the company.
- A higher percentage of the customers who churned had an international plan.
- High total day minutes spent potentially leads to high churn rate, this implies that a customer was charged based on the total number of minutes spent.
- Total evening charge and night minutes also increased the churn rate. Therefore from the findings I would reccomend SyriaTel to:

1. Improve the quality of customer service provided
2. Review the international plan.
3. Review on the prices how they charge customers based on the total number of minutes.

Limitations.

- The dataset used has limited number of features while there could be other factors that contribute to customer churn.
- The classification models used in this analysis make certain assumptions, such as independence, and normality. Violations of these assumptions could impact the accuracy and reliability of the results.

In [ ]: