BUSINESS UNDERSTANDING

In this project the aim was to predict whether a customer will ("soon") stop doing business with SyriaTel, a telecommunications company. This is a binary classification problem. the first step was to analyze the key factors affecting customers doing business with SyriaTel.I employed six machine learning models namely:

1. Logistic Regression
2. Decison Tree
3. RandomForestClassifer
4. Adaboost
5. k-nearest neighbor
6. Gradient Boosting

BUSINESS PROBLEM

Most naturally, your audience here would be the telecom business itself, interested in reducing how much money is lost because of customers who don't stick around very long. The question you can ask is: are there any predictable patterns here?

OBJECTIVES:

1. predict the patterns that contribute to customer churn, and to which level.

2. Predict the patterns that are highly correlated with churn.

CONTEXT:

0- it standards for customers who havent stopped doing business with SyriaTel.(non-churn)

1- it standards for clients who stopped doing business with SyriaTel(churn).

Evaluation: for the evaluation bit, i used precision ,recall, f1-score and support to characterize churn and non churn from the different models.

2 DATA UNDERSTANDING:

2.1 IMPORTING LIBRARIES

In [ ]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
import seaborn as sns
```

2.2 LOADING DATASET

In [ ]:
```
## Reading in the data and previewing the dataset
df1 = pd.read_csv("bigml_59c28831336c6604c800002a.csv")
df1.head()
```

Out[ ]:

	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

2.3 Checking the Dataset

In [ ]:
```
df1.shape
```

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

In [ ]:
```
df1.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 [ ]:
```
#our dataset contains 3333 rows and 21 columns
df1.groupby('churn')['churn'].agg('count')
```

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

In [ ]:
```
#checking for any missing values.
df1.isnull().sum()
```

Out[ ]:
```
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
```

from our dataset there is no presence of missing values from all the columns

Checking for the statistical analysis of our dataset

In [ ]:
```
df1.describe()
```

Out[ ]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	to r
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363

In [ ]:
```
df1.describe(include=["object","bool"])
```

Out[ ]:

	state	phone number	international plan	voice mail plan	churn
count	3333	3333	3333	3333	3333
unique	51	3333	2	2	2
top	WV	382-4657	no	no	False
freq	106	1	3010	2411	2850

In [ ]:
```
df1 ['total day charge'].unique()
```

Out[ ]:
```
array([45.07, 27.47, 41.38, ..., 54.59, 39.29, 30.74])
```

In [ ]:
```
df1.sort_values(by="total day charge", ascending=False).head()
```

Out[ ]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tot da charg
365	CO	154	415	343-5709	no	no	0	350.8	75	59.(
985	NY	64	415	345-9140	yes	no	0	346.8	55	58.9
2594	OH	115	510	348-1163	yes	no	0	345.3	81	58.7
156	OH	83	415	370-9116	no	no	0	337.4	120	57.3
605	MO	112	415	373-2053	no	no	0	335.5	77	57.(

5 rows × 21 columns

In [ ]:
```
df1.sort_values(by=["churn","total day charge"], ascending=[True, False]).head()
```

Out[ ]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tot da charg
688	MN	13	510	338-7120	no	yes	21	315.6	105	53.6
2259	NC	210	415	363-7802	no	yes	31	313.8	87	53.3
534	LA	67	510	373-6784	no	no	0	310.4	97	52.7
575	SD	114	415	351-7369	no	yes	36	309.9	90	52.6
2858	AL	141	510	388-8583	no	yes	28	308.0	123	52.3

5 rows × 21 columns

In [ ]:
```
#calculating the mean for the target variable
df1["churn"].mean()
```

Out[ ]:
```
0.14491449144914492
```

In [ ]:
```
df1.apply(np.max)
```

Out[ ]:
```
state                        WY
account length              243
area code                   510
phone number           422-9964
international plan          yes
voice mail plan             yes
number vmail messages        51
total day minutes         350.8
total day calls             165
total day charge          59.64
total eve minutes         363.7
total eve calls             170
total eve charge          30.91
total night minutes       395.0
total night calls           175
total night charge        17.77
total intl minutes         20.0
total intl calls             20
total intl charge           5.4
customer service calls        9
churn                      True
dtype: object
```

In [ ]:
```
columns_to_show = ["total day minutes","total eve minutes","total night minutes"]
df1.groupby(["churn"])[columns_to_show].describe(percentiles=[])
```

Out[ ]:

	total day minutes						tc			
	count	mean	std	min	50%	max	count	mean	std	mi
churn										
False	2850.0	175.175754	50.181655	0.0	177.2	315.6	2850.0	199.043298	50.292175	0.
True	483.0	206.914079	68.997792	0.0	217.6	350.8	483.0	212.410145	51.728910	70.

In [ ]:
```
df1["total minutes"] =( df1["total day minutes"]+df1["total eve minutes"] +df1["tot
df1.head()
```

Out[ ]:

	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 × 22 columns

In [ ]:
```
#add a column of the total charge
df1["total charge"] = (df1["total day charge"] +df1["total eve charge"] + df1["tota
df1.head()
```

Out[ ]:

	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 × 23 columns

3. EDA

In [ ]:
```
#convert the target variable to integer
df1['churn'] = df1['churn'].astype(int)
```

In [ ]:
```
# Encode categorical variables
df1['international plan'] = df1['international plan'].map({'yes': 1, 'no': 0})
df1['voice mail plan'] = df1['voice mail plan'].map({'yes': 1, 'no': 0})
```

Data Preparation (dropping less informative columns)

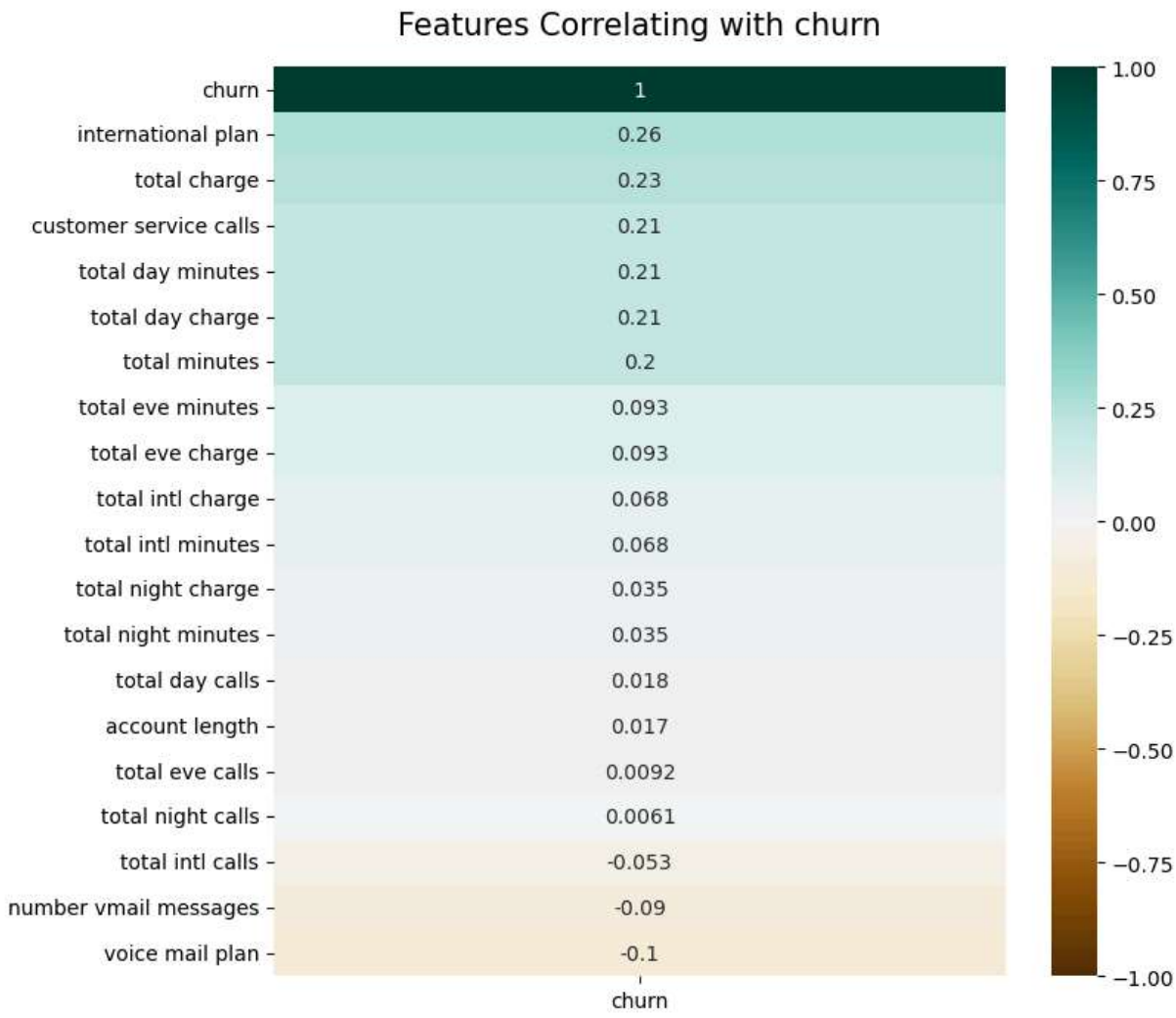
In [ ]:
```
#dropping the less informative columns
df1.drop(columns=['state', 'area code', 'phone number'], inplace=True)
```

Checking for correlation between columns and target variable

In [ ]:
```
#calculating the correlation between numerical features and churn
correlations = df1.corr()['churn'].sort_values(ascending=False)
print(correlations)
```

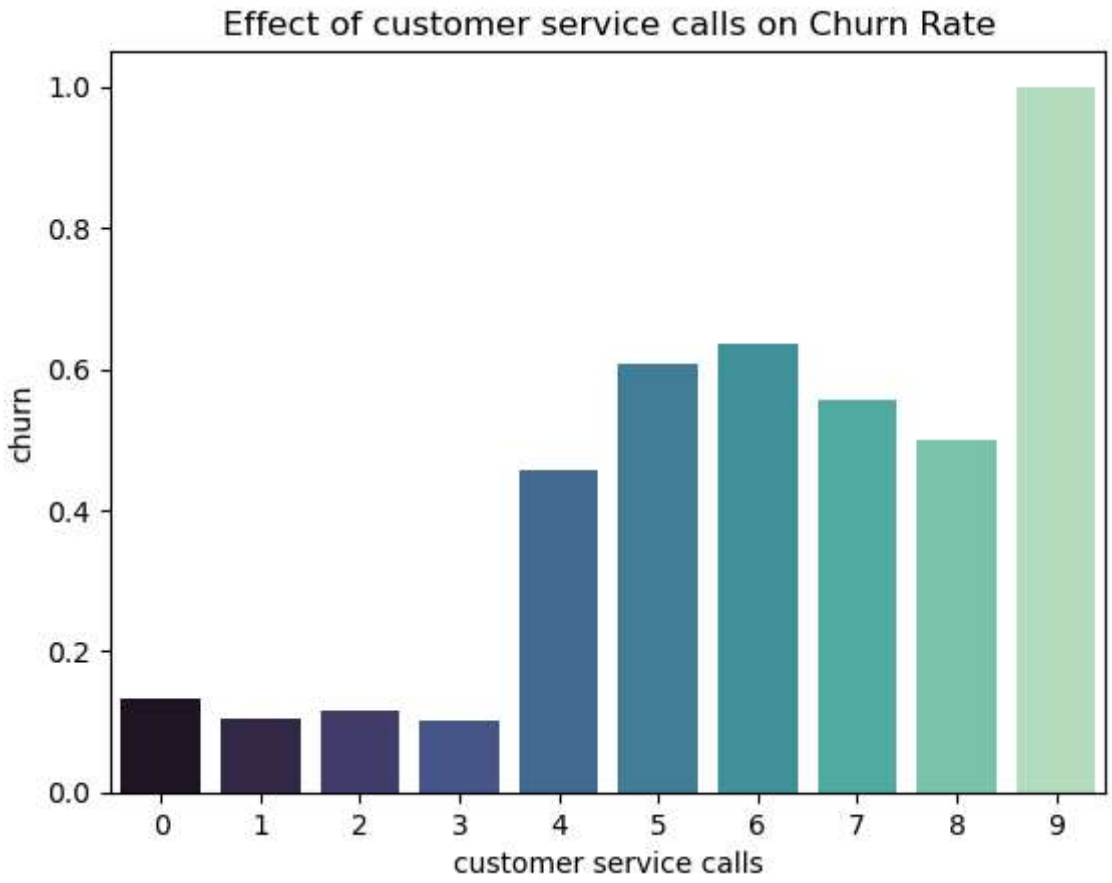
```
churn                     1.000000
international plan        0.259852
total charge              0.231549
customer service calls    0.208750
total day minutes         0.205151
total day charge          0.205151
total minutes             0.198607
total eve minutes         0.092796
total eve charge          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
total night calls         0.006141
total intl calls         -0.052844
number vmail messages    -0.089728
voice mail plan          -0.102148
Name: churn, dtype: float64
```

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(8, 8))
heatmap = sns.heatmap(df1.corr()[['churn']].sort_values(by='churn', ascending=False
heatmap.set_title('Features Correlating with churn', fontdict={'fontsize':15}, pad=
```



In [ ]:
```
sns.barplot(x='customer service calls', y='churn',
                    data=df1, palette='mako', errorbar=None).set_title('Effect of c
```

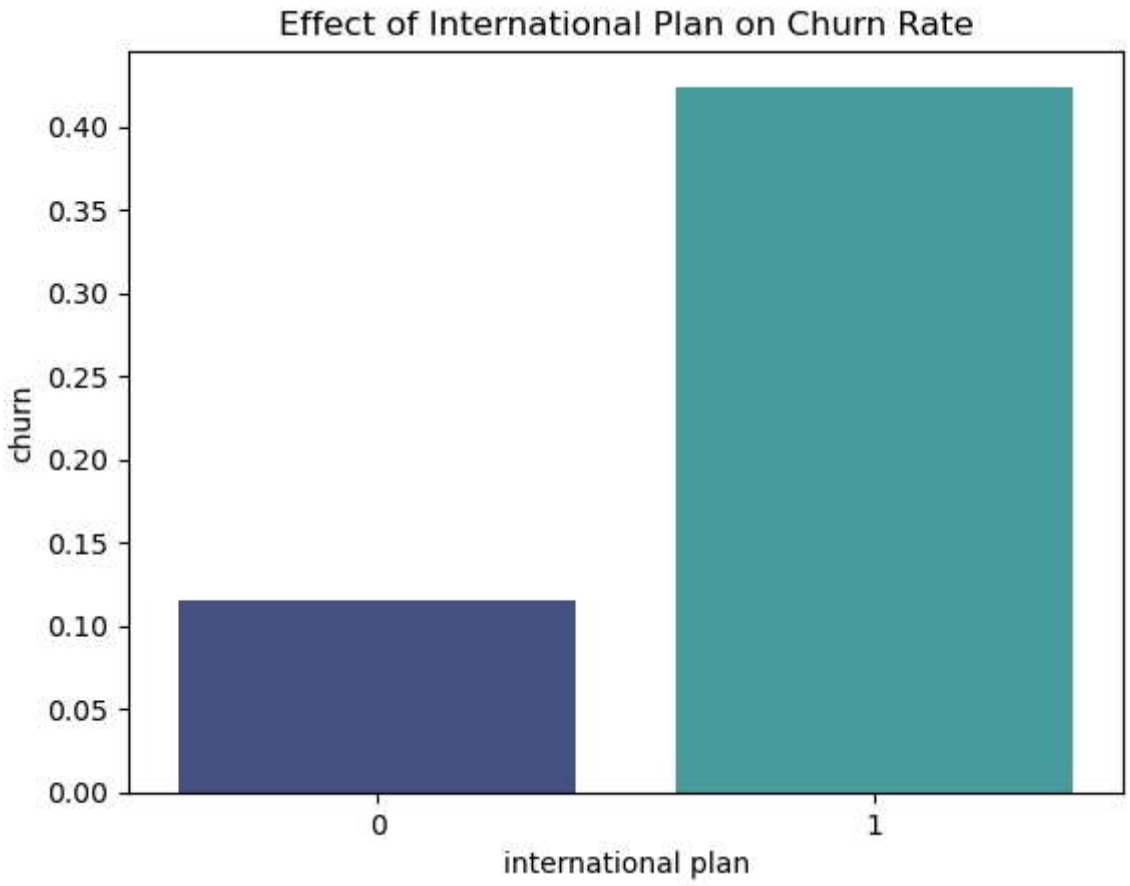
Out[ ]:
```
Text(0.5, 1.0, 'Effect of customer service calls on Churn Rate')
```



From the graph above as the customer service calls increase the more the rate of churn.

In [ ]:
```
sns.barplot(x='international plan', y='churn',
                    data=df1, palette='mako', errorbar=None).set_title('Effect of I
```

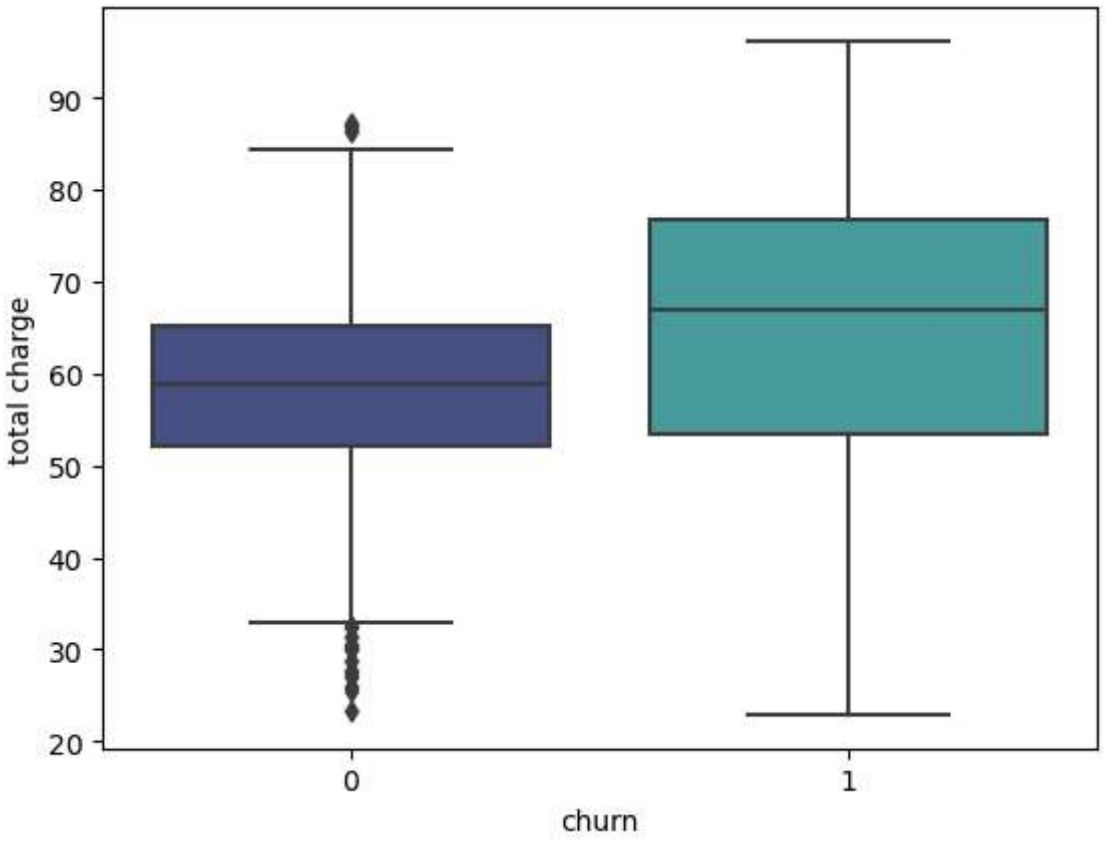
Out[ ]:
```
Text(0.5, 1.0, 'Effect of International Plan on Churn Rate')
```



The churn rate is significantlly high for customer with an international plan compared to those without.

In [ ]:
```
sns.boxplot(x='churn', y='total charge',
                data=df1, palette='mako')
```

Out[ ]:
```
<Axes: xlabel='churn', ylabel='total charge'>
```



Overall, the graph shows that customers who churned generally have higher total charges compared to those who did not churn.

predicting the y and x variable

In [ ]:
```
# defining the appropriate x and y
X =df1.drop(columns=['churn'])
y = df1['churn']
```

splitting the data

In [ ]:
```
# Split the data into training and testing sets

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Scaling the numerical features

In [ ]:
```
# Scale the numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

MODELLING

fit logistic regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Initialize and train the model
log_reg = LogisticRegression()

#model.fit
log_reg.fit(X_train_scaled, y_train)

#predictions
y_pred_log_reg = log_reg.predict(X_test_scaled)

# Evaluation
print("Logistic Regression:")
print(classification_report(y_test, y_pred_log_reg))
print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       566
           1       0.58      0.18      0.27       101

    accuracy                           0.86       667
   macro avg       0.73      0.58      0.60       667
weighted avg       0.83      0.86      0.82       667

Accuracy: 0.856071964017991
```

In [ ]:
```
# Make predictions
y_pred_log_reg = log_reg.predict(X_test_scaled)
```

from the above graph the accuracy rate is at 85.6%, the precision for non churn is 87% and for churn is 58%. recall from non churn is 98% and for churn is 18% f1 score for non churn 92% and for churn 27%. The model has high overall accuracy (85.6%), but this is largely driven by the model's performance on the majority class (class 0).

Fit a Decision Tree model as comparison

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the model
model = DecisionTreeClassifier(random_state=42)

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

#predictions
```

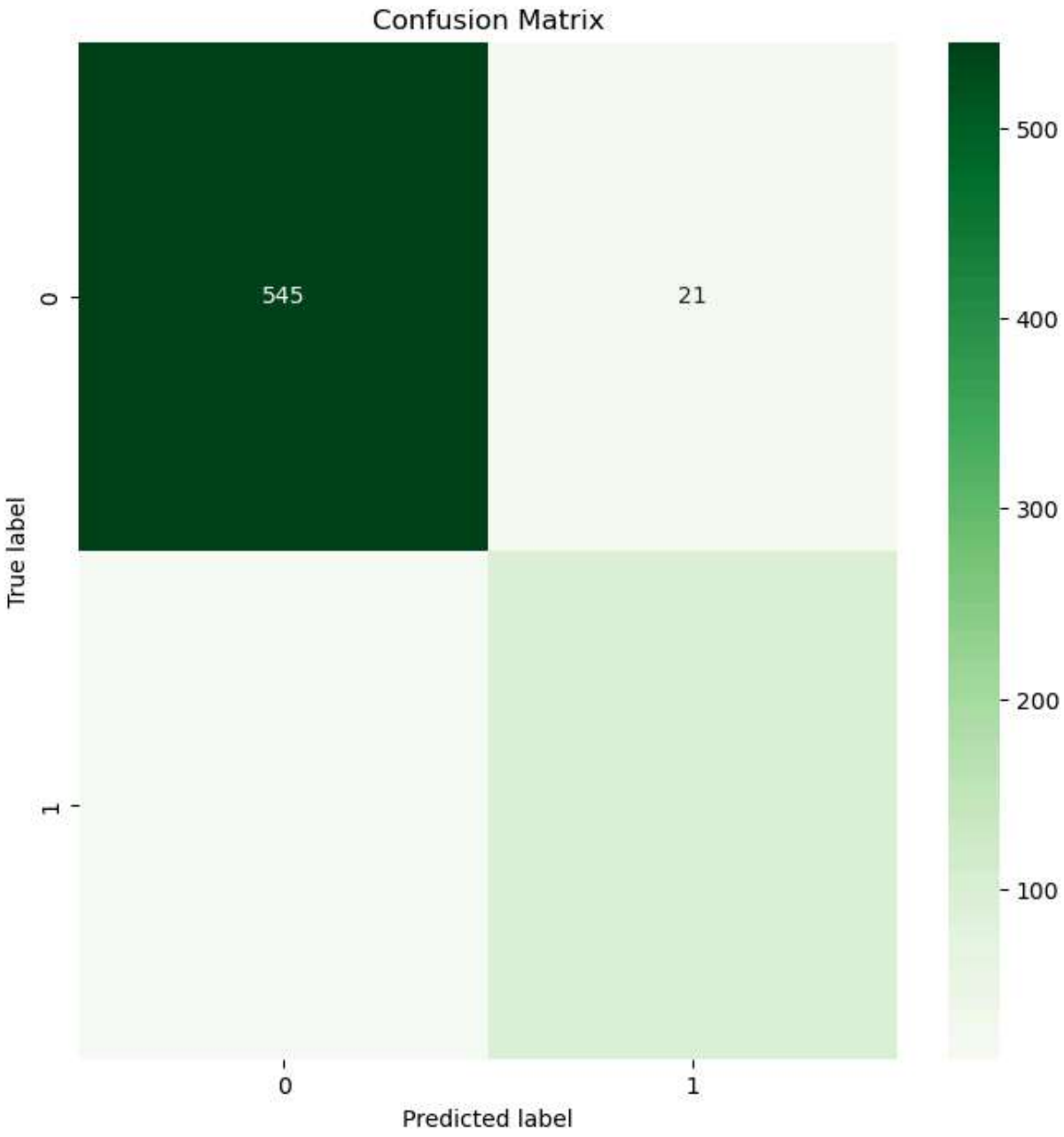
```
y_pred_tree = model.predict(X_test_scaled)

model.score(X_test_scaled, y_pred_tree)



# Evaluation
print("Decision Tree:")
print(classification_report(y_test, y_pred_tree))
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
```

```
Decision Tree:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       566
           1       0.81      0.89      0.85       101

    accuracy                           0.95       667
   macro avg       0.90      0.93      0.91       667
weighted avg       0.95      0.95      0.95       667

Accuracy: 0.952023988005997
```

The model has a high accuracy of 95.2% indicating that it correctly predicts the churn status for a large proportion of the samples.it has a relatively high recall (0.89), indicating that it correctly identifies a significant proportion of actual churn cases. it also performs well in identifying non churn customers from the precision and recall of class(0)

confusion matrix of the decision tree

In [ ]:
```
from sklearn.metrics import confusion_matrix
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred_tree)

# plotting
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



Fit a Random Forest model, " compared to "Decision Tree model, accuracy go up by 3%

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)

#model.fit
rf.fit(X_train_scaled, y_train)

#predictions
y_pred_rf = rf.predict(X_test_scaled)

# Evaluation
print("Random Forest:")
```

```
print(classification_report(y_test, y_pred_rf))
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       566
           1       1.00      0.87      0.93       101

    accuracy                           0.98       667
   macro avg       0.99      0.94      0.96       667
weighted avg       0.98      0.98      0.98       667

Accuracy: 0.9805097451274363
```

Precision is 1.00, for class(1) meaning that 100% of the customers predicted as churners are actually churners. with an accuracy of 98% means that the predictions made by the model are correct. This method is strongly effective in predicting customer churn with a good performances in identifying non churn and churn

Fit AdaBoost model, " compared to "Decision Tree model.

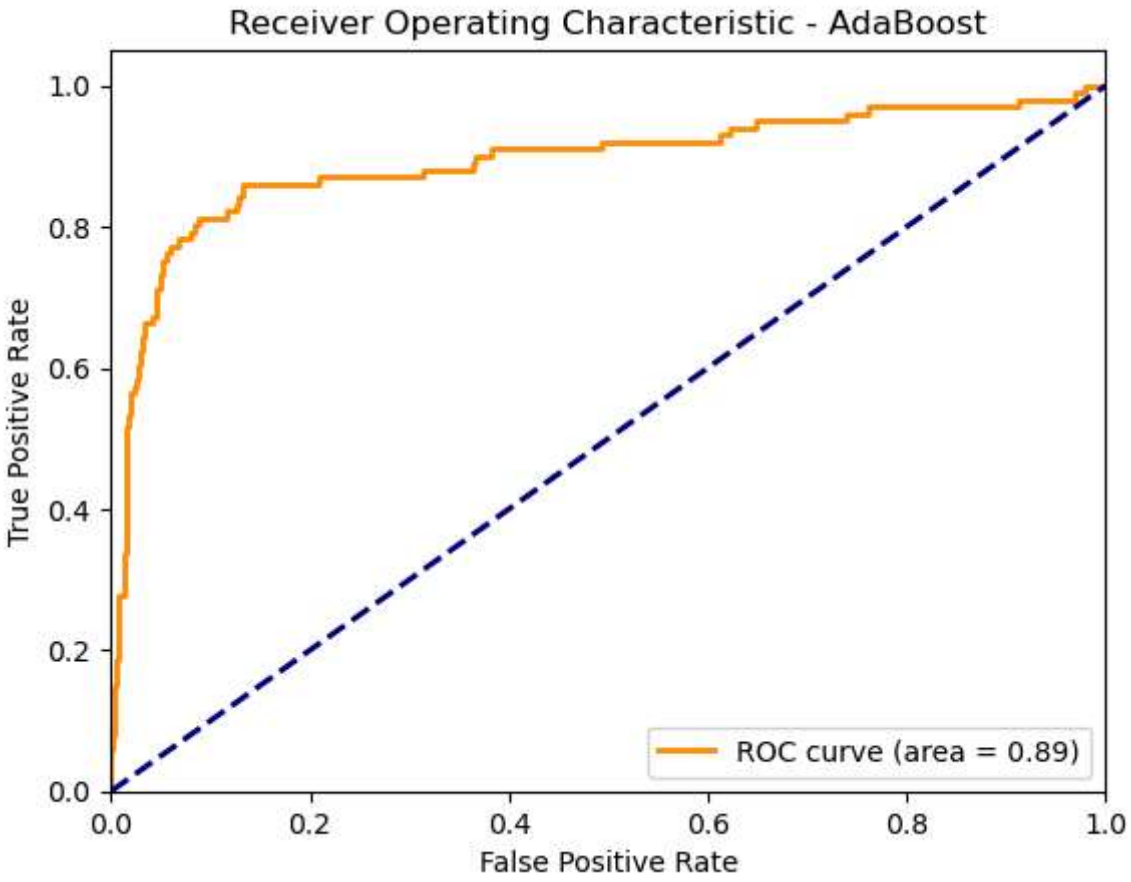
In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc, classification_report


# Initialize and train the AdaBoost model
ada = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1), n_estimator
ada.fit(X_train_scaled, y_train)

#predictions
y_pred_ada = ada.predict(X_test_scaled)

# Confusion Matrix
cm_ada = confusion_matrix(y_test, y_pred_ada)
print("AdaBoost Confusion Matrix:")
print(cm_ada)
```

```
AdaBoost Confusion Matrix:
[[553  13]
 [ 44  57]]
```

In [ ]:
```
# Classification Report
print("AdaBoost Classification Report:")
print(classification_report(y_test, y_pred_ada))

# Accuracy
accuracy_ada = accuracy_score(y_test, y_pred_ada)
print("AdaBoost Accuracy:", accuracy_ada)

# ROC Curve
fpr_ada, tpr_ada, _ = roc_curve(y_test, ada.predict_proba(X_test_scaled)[:, 1])
roc_auc_ada = auc(fpr_ada, tpr_ada)

plt.figure()
```

```
plt.plot(fpr_ada, tpr_ada, color='darkorange', lw=2, label='ROC curve (area = %0.2f
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - AdaBoost')
plt.legend(loc="lower right")
plt.show()
```

```
AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       566
           1       0.81      0.56      0.67       101

    accuracy                           0.91       667
   macro avg       0.87      0.77      0.81       667
weighted avg       0.91      0.91      0.91       667

AdaBoost Accuracy: 0.9145427286356822
```

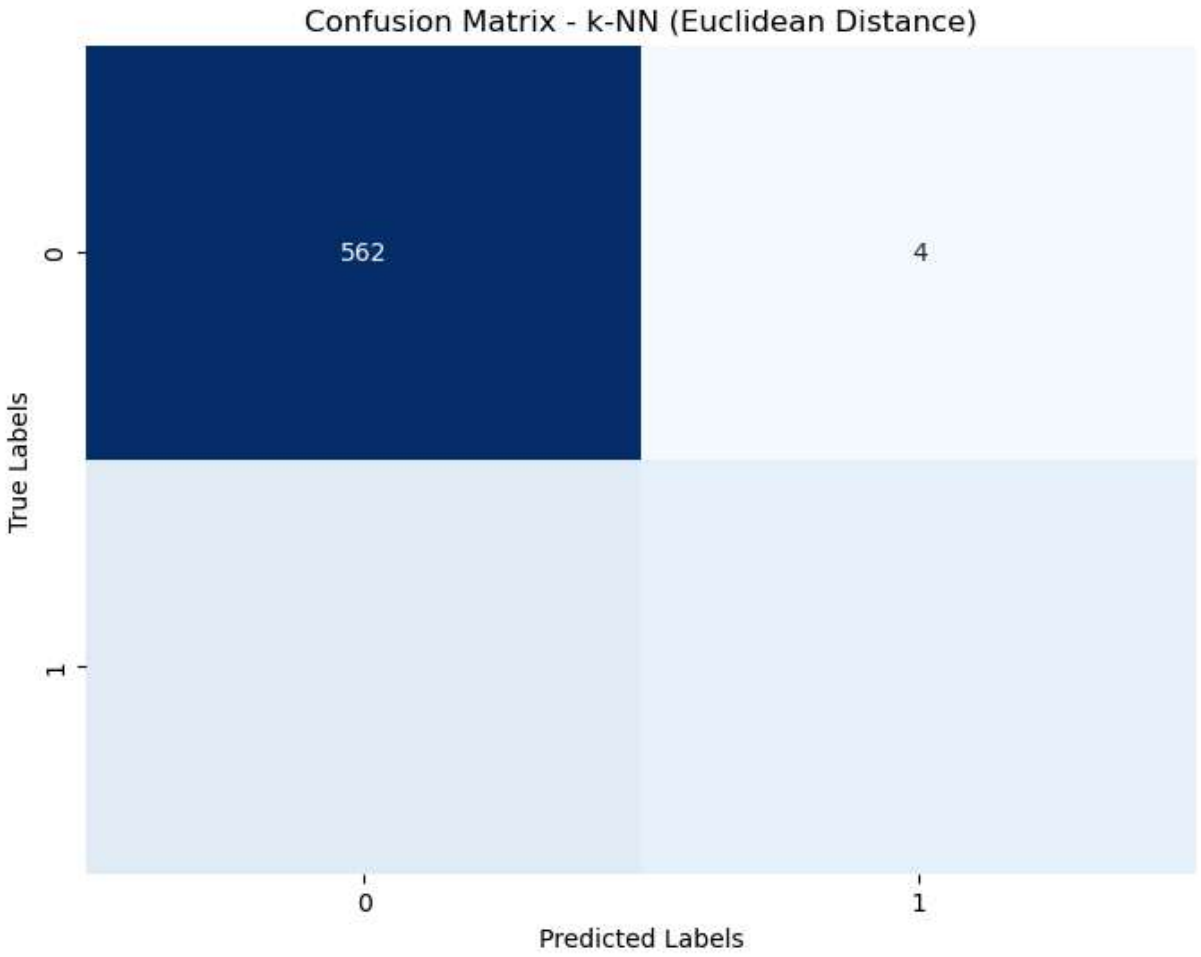


With an AUC of 0.89, the AdaBoost model performs well in distinguishing between churn and non-churn customers. The closer the AUC is to 1, the better the model is at making these distinctions. The ROC curve shows that the AdaBoost model has a high true positive rate (sensitivity) for most thresholds, indicating it correctly identifies a large proportion of churn instances.

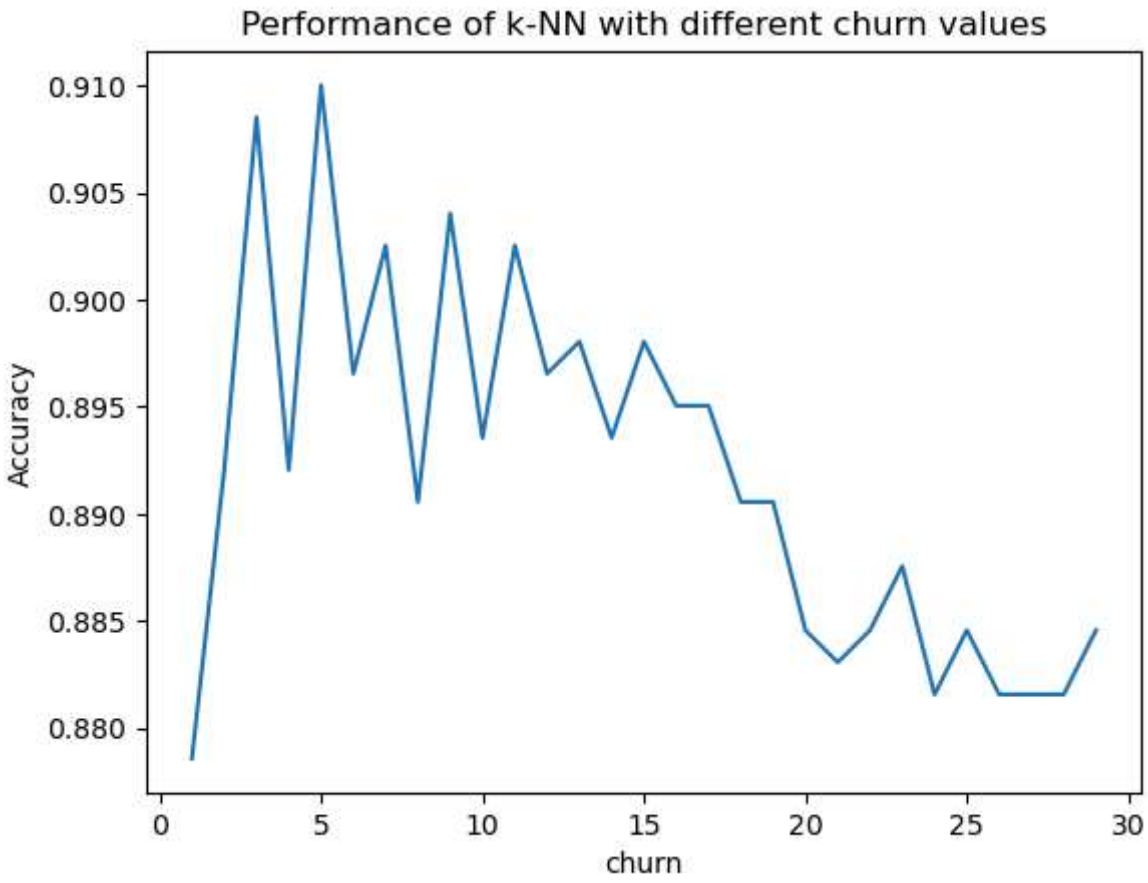
k nearest neighbor model

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

#Train the k-NN Model with Euclidean Distance
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

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

# Predictions
y_pred_knn = knn.predict(X_test_scaled)

# Evaluation
print("k-Nearest Neighbors (Euclidean Distance):")
print(classification_report(y_test, y_pred_knn))
print("Accuracy:", accuracy_score(y_test, y_pred_knn))

#Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_knn)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - k-NN (Euclidean Distance)')
plt.show()
```

```
k-Nearest Neighbors (Euclidean Distance):
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       566
           1       0.92      0.45      0.60       101

    accuracy                           0.91       667
   macro avg       0.91      0.72      0.77       667
weighted avg       0.91      0.91      0.90       667

Accuracy: 0.9100449775112444
```



In [ ]:
```
# Find the best k value
accuracies = []
ks = range(1, 30)

for churn in ks:
    knn = KNeighborsClassifier(n_neighbors=churn, metric='euclidean')
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

# Plot the performance of k-NN for different k values
fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="churn", ylabel="Accuracy", title="Performance of k-NN with different
plt.show()
```



The model has a high precision for both classes, indicating that most of the predicted. The model performs well overall with a high accuracy of 91%,

Gradient Boosting

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier

#Gradient Boosting Classifier
gb = GradientBoostingClassifier(random_state=42)

#model.fit
gb.fit(X_train_scaled, y_train)

#predictions
y_pred_gb = gb.predict(X_test_scaled)

#Evaluation

print("Gradient Boosting:")
print(classification_report(y_test, y_pred_gb))
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
```

```
Gradient Boosting:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       566
           1       0.99      0.87      0.93       101

    accuracy                           0.98       667
   macro avg       0.98      0.93      0.96       667
weighted avg       0.98      0.98      0.98       667

Accuracy: 0.9790104947526237
```

The model performed well with an accuracy of 97.9%, this shows it made correct predictions most of the time.The high f1_score,recall and precision for both classes indicates the model is effective in distinguishing both classes.

CONCLUSIONS:

1. the customer churn rate was mostly affected by the international plan, total charges, customer service calls and voice mail messages.
2. Randomforest classifer was the most effective and robust model for identifying both classes from churn to non churn.
3. from the ROC curve , we get to understand AdaBoost model has a high true positive rate for most thresholds, indicating it correctly identifies a large proportion of churn instances and low positive rate it does not frequently misclassify non-churn instances as churn.

RECCOMENDATIONS:

1. The Syria Tel communication company, needs to make a better international plan, for clients enrolled for the international plan.eg better connectivity, low charge rates
2. They also need to be proactive in offering great customer support and also collecting feedback reports from client so as to reduce their churn rate thus losing money.
3. They should look for methods to engage clients more and also give offers, so as those affected by a high total charge could feel considered.eg loyalty rewards, discounts
4. Implement a system to generate predictive alerts for customers who are likely to churn. This allows the customer support team to intervene early and offer solutions to retain them.