SyriaTel Customer Churn Analysis

Natalia Edelson

Flatiron School, Project_3

Self-Paced

Importing Libraries

In [418…

```
import pandas as pd
import numpy as np
np.random.seed(0)
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc
from sklearn.metrics import plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import auc, roc_curve, roc_auc_score, precision_recall_curv
from sklearn.model_selection import cross_val_score,KFold
from sklearn.metrics import f1_score,precision_score,recall_score,plot_confusion
global model_auc, model_ll, model_roc_auc
import warnings
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings('ignore')
import time
import seaborn as sns
%matplotlib inline
from xgboost import XGBClassifier
from sklearn.inspection import permutation_importance
```

In [420…

```
# Import the data and look into the different columns.
Customer_Churn = pd.read_csv('Churn.csv')

Customer_Churn.head()
```

Out[420…

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tot e cal
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	...	1(
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	1'
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	ε

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tot ev cal
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	1:

5 rows × 21 columns

Exploring and cleaning the data

- Check data types
- Check for null values
- Check for duplicates
- Check for imbalance of churn True vs False
- Check for outlier

In [421…

```
# Column Names: Replacing space with underscore
new_columns = [i.replace(' ', '_') for i in Customer_Churn.columns]
Customer_Churn.columns = new_columns
```

In [422…

```
# Check for null values
null_counts = Customer_Churn.isnull().sum()
print("Number of null values in each column:\n{}".format(null_counts))
```

```
Number of null values in each column:
state                     0
account_length            0
area_code                 0
phone_number              0
international_plan        0
voice_mail_plan           0
number_vmail_messages     0
total_day_minutes         0
total_day_calls           0
total_day_charge          0
total_eve_minutes         0
total_eve_calls           0
total_eve_charge          0
total_night_minutes       0
total_night_calls         0
total_night_charge        0
total_intl_minutes        0
total_intl_calls          0
total_intl_charge         0
customer_service_calls    0
churn                     0
dtype: int64
```

In [423…

```
Customer_Churn.shape
```

Out[423…

```
(3333, 21)
```

In [424…

```
#Distribution of data type
```

```
print("Data types and their frequency\n{}".format(Customer_Churn.dtypes.value_co
```

```
Data types and their frequency
float64    8
int64      8
object     4
bool       1
dtype: int64
```

In [425…

```
# Check for duplicates
Customer_Churn.duplicated().sum()
```

Out[425… 0

In [426…

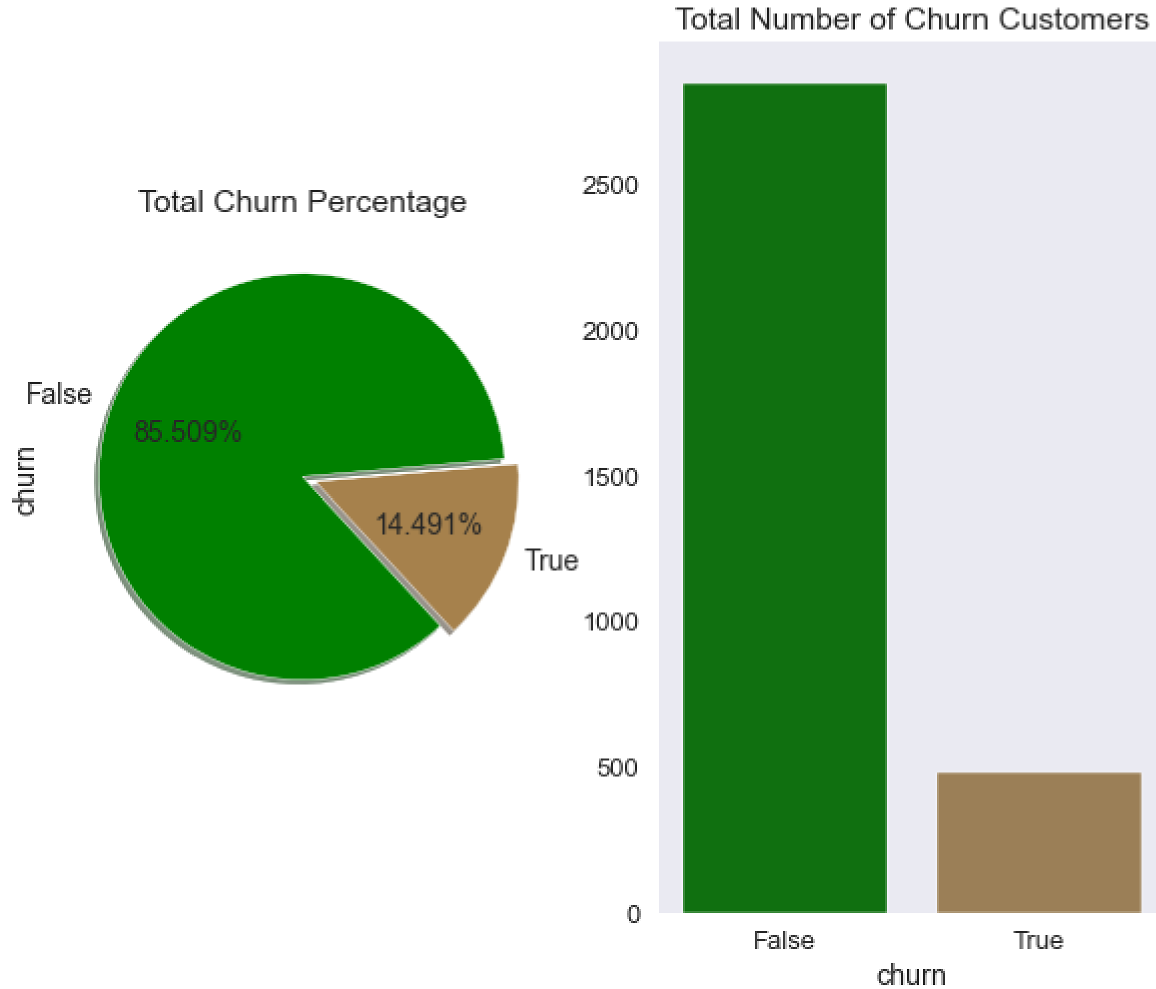
```
# Check balance of target data
Customer_Churn['churn'].value_counts()
```

Out[426…

```
False    2850
True      483
Name: churn, dtype: int64
```

In [427…

```
# Data to plot
plt.style.use(['seaborn-dark','seaborn-talk'])

fig, ax = plt.subplots(1,2,figsize=(10,8))

Customer_Churn['churn'].value_counts().plot.pie(explode=[0,0.07], ax=ax[0], auto
                                    fontsize=14, startangle=5, colors=["#008000"
ax[0].set_title('Total Churn Percentage')

sns.countplot('churn', data=Customer_Churn, ax=ax[1], palette=["#008000", "#a681
ax[1].set_title('Total Number of Churn Customers')
ax[1].set_ylabel('      ')

plt.show()
```



We can see that the data is not balanced: 85% of people are not churning. We need to keep it in mind when building our models and see whether we need to use any tools to balance the data.

In [428…

```
# Explore the dataset's stats and check for outliers
display(Customer_Churn.describe())
```

	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_calls
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644
std	39.822106	42.371290	13.688365	54.467389	20.069084
min	1.000000	408.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000
75%	127.000000	510.000000	20.000000	216.400000	114.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000

There are no outliers.

Converting Categorical variables to dummy ones

1. state - object
2. international_plan - object
3. voice_mail_plan - object
4. churn - bool

In [429…

```
conditions = [
    Customer_Churn['international_plan'] == 'no',
    Customer_Churn['international_plan'] == 'yes'
]

choices = [
    0,
    1,
]

Customer_Churn['international_plan'] = np.select(conditions, choices)


conditions = [
    Customer_Churn['voice_mail_plan'] == 'no',
    Customer_Churn['voice_mail_plan'] == 'yes'
]

choices = [
    0,
    1,
]

Customer_Churn['voice_mail_plan'] = np.select(conditions, choices)




conditions = [
    Customer_Churn.churn == True,
    Customer_Churn.churn == False
]

choices = [
    1,
    0,
]

Customer_Churn.churn = np.select(conditions, choices)

Customer_Churn.head()
```

Out[429…

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number
0	KS	128	415	382-4657	0	1	
1	OH	107	415	371-7191	0	1	
2	NJ	137	415	358-1921	0	0	
3	OH	84	408	375-9999	1	0	
4	OK	75	415	330-6626	1	0	

5 rows × 21 columns

In [430…

```
# Cleaning and exploring relationships
Customer_Churn.phone_number
```

Out[430…

```
0       382-4657
1       371-7191
2       358-1921
3       375-9999
4       330-6626
          ...
3328    414-4276
3329    370-3271
3330    328-8230
3331    364-6381
3332    400-4344
Name: phone_number, Length: 3333, dtype: object
```

In [431…

```
Customer_Churn.area_code.unique()
```

Out[431…

```
array([415, 408, 510])
```

Dropping columns that are not useful.

There are only three types of area codes which clearly does not represent the real population given the distribution of costumer's states. Also, the six digits of the phone number will not give us any useful information to predict churning. Therefore, we will drop both the "area_codes" and "phone_number" columns.

In [432…

```
Customer_Churn.drop('area_code', axis = 1, inplace = True)
Customer_Churn.drop('phone_number', axis = 1, inplace = True)
```

In [433…

```
# Dheck that the drop function worked

Customer_Churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account_length          3333 non-null   int64
 2   international_plan      3333 non-null   int64
 3   voice_mail_plan         3333 non-null   int64
 4   number_vmail_messages   3333 non-null   int64
 5   total_day_minutes       3333 non-null   float64
 6   total_day_calls         3333 non-null   int64
 7   total_day_charge        3333 non-null   float64
 8   total_eve_minutes       3333 non-null   float64
 9   total_eve_calls         3333 non-null   int64
 10  total_eve_charge        3333 non-null   float64
 11  total_night_minutes     3333 non-null   float64
 12  total_night_calls       3333 non-null   int64
 13  total_night_charge      3333 non-null   float64
```

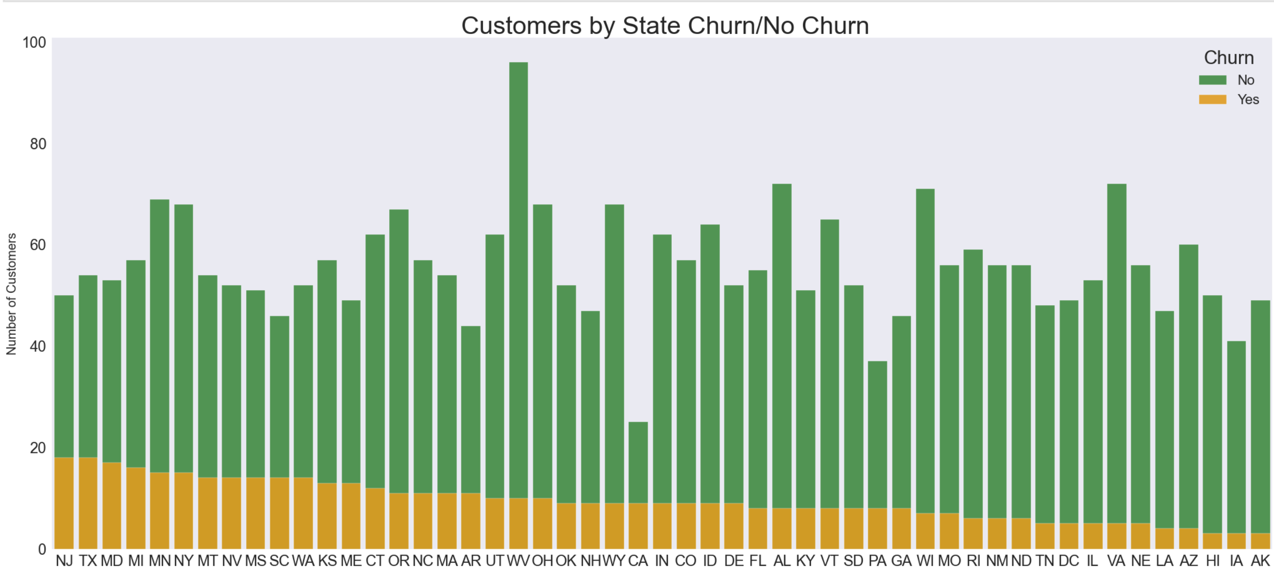
```
 14   total_intl_minutes      3333 non-null   float64
 15   total_intl_calls        3333 non-null   int64
 16   total_intl_charge       3333 non-null   float64
 17   customer_service_calls  3333 non-null   int64
 18   churn                   3333 non-null   int64
dtypes: float64(8), int64(10), object(1)
memory usage: 494.9+ KB
```

Exploring States

We investigate the states churning data. We want to see whether there are any states that show more churning than others.

In [434…

```
states_plot = Customer_Churn.pivot_table(index='state', columns='churn', values=
states_plot.columns = ['state', 'false', 'true']
states_plot = states_plot.sort_values('true', ascending=False).reset_index(drop=

# Plot
plt.figure(figsize=(35,15))

# Two bar charts
s1 = sns.barplot(x = 'state', y = 'false', data = states_plot, color = 'green',
s2 = sns.barplot(x = 'state', y = 'true', data = states_plot, color = 'orange',

# Title, labels and legend
plt.savefig('output.png')
plt.title(' Customers by State Churn/No Churn', size=40)
plt.ylabel('Number of Customers', size=20)
plt.xlabel(None)
plt.legend(title='Churn', prop={'size': 22}, title_fontsize=30)
plt.xticks(fontsize=24)
plt.yticks(fontsize=24)
plt.show();
```



In [435…

```
top_states =states_plot.head(6)
top_states
```

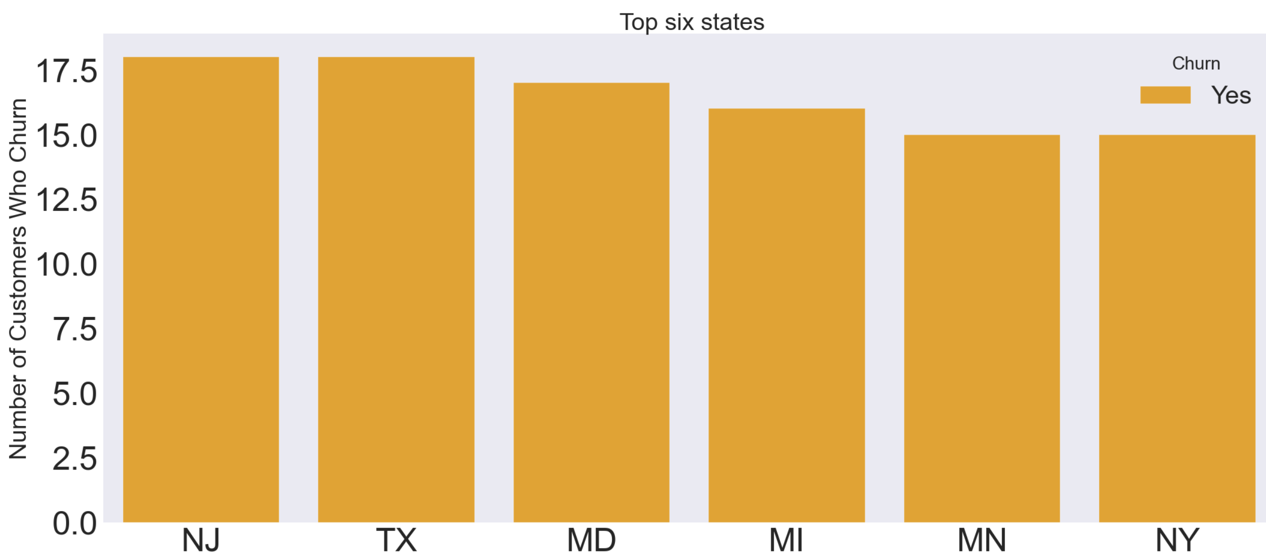
Out[435…

state	false	true

	state	false	true
0	NJ	50	18
1	TX	54	18
2	MD	53	17
3	MI	57	16
4	MN	69	15
5	NY	68	15

In [436…

```
plt.figure(figsize=(35,15))

# Two bar charts
S1 = sns.barplot(x = 'state', y = 'true', data = top_states, color = 'orange', a

# Title, labels and legend
plt.savefig('output.png')
plt.title(' Top six states', size=40)
plt.ylabel('Number of Customers Who Churn', size=40)
plt.xlabel(None)
plt.legend(title='Churn', prop={'size': 42}, title_fontsize=30)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.show()
plt.savefig('Top 6 States.pdf')
```




```
<Figure size 748.8x514.8 with 0 Axes>
```

Check the average churn of the highest states versus the average of all the states.

In [437…

```
states_plot.true.mean()
```

Out[437…

```
9.470588235294118
```

In [438…

```
top_states.true.mean()
```

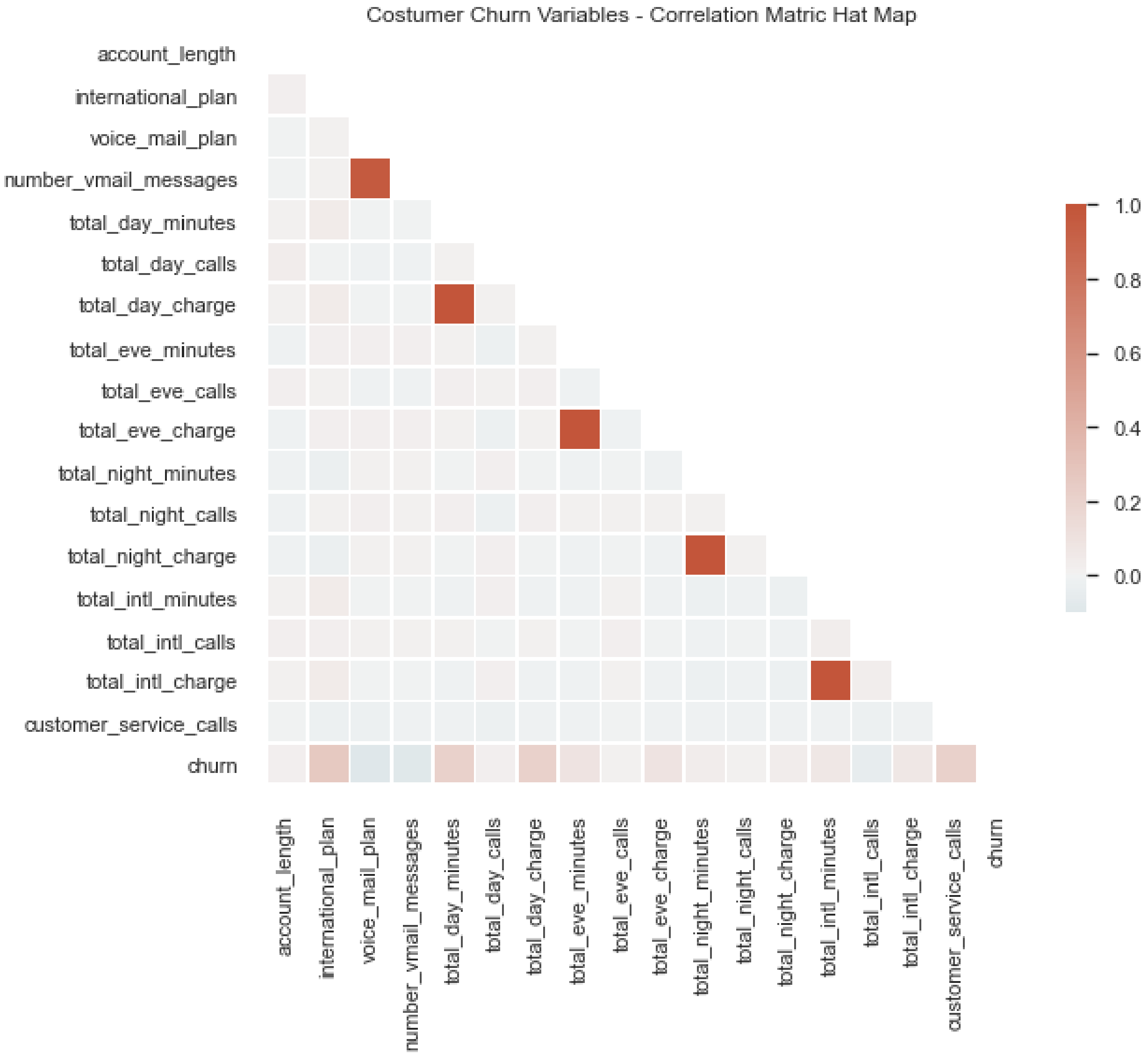
Out[438…
```
16.5
```

The states above were the six highest. It will be useful for the company to have these states flagged in order decide whether there are cost effective methods to refrain them from leaving.

In [439…
```
#Below we are checking the correlation between variables and we will examine the
```

In [440…
```
sns.set(style="white")

corr = Customer_Churn.corr().round(2)

mask = np.triu(np.ones_like(corr, dtype=bool))
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(9, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.title(' Costumer Churn Variables - Correlation Matric Hat Map ')
```

Out[440…
```
Text(0.5, 1.0, ' Costumer Churn Variables - Correlation Matric Hat Map ')
```



In [441…

```
(abs(Customer_Churn.corr()) > 0.85).sum()
```

Out[441…

```
account_length            1
international_plan        1
voice_mail_plan           2
number_vmail_messages     2
total_day_minutes         2
total_day_calls           1
total_day_charge          2
total_eve_minutes         2
total_eve_calls           1
total_eve_charge          2
total_night_minutes       2
total_night_calls         1
total_night_charge        2
total_intl_minutes        2
total_intl_calls          1
total_intl_charge         2
customer_service_calls    1
churn                     1
dtype: int64
```

The variables below are naturally perfectly correlated and therefore provide redundant information. We can exclude the minutes and will leave the charges.

total_day_minutes & total_day_charge

total_eve_minutes & total_eve_charge

In [442…

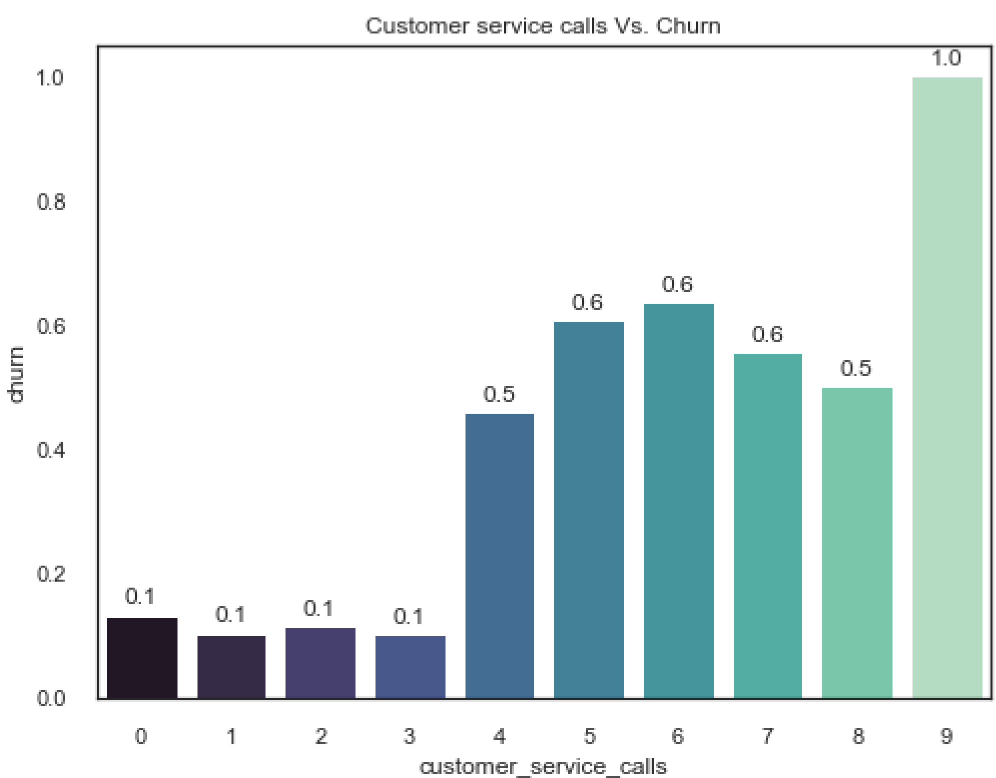
```
Customer_Churn.drop('total_day_minutes', axis = 1, inplace = True)
Customer_Churn.drop('total_eve_minutes', axis = 1, inplace = True)
```

We will look closely into the variables that show a higher correlation to churn.

- total_day_charge
- total_eve_charge
- customer_service_calls

In [443…

```
plt.figure(figsize=(8, 6))
splot = sns.barplot(x='customer_service_calls', y='churn',
                    data=Customer_Churn, palette='mako', ci=None)



for p in splot.patches:

    splot.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
plt.title('Customer service calls Vs. Churn')
plt.show()
```

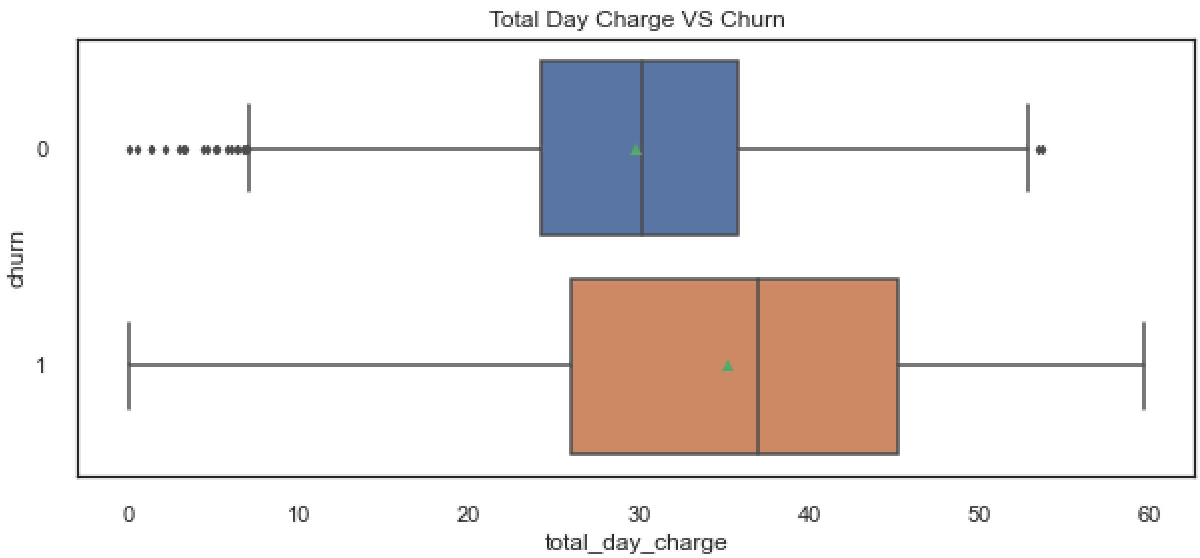


We can clearly see that the higher customer service call will likely lead to a customer leaving. Especially after 3 calls we saw an increase in churning. We would recommend investing in the costumer service assistance.

We visualize the data to clearly see where the data rests. Let's use boxplot to visualize the evening and day charge. Using a box and whisker plot, allows us to see how the data is distributed and whether it is concentrated in a certain area.

In [444…

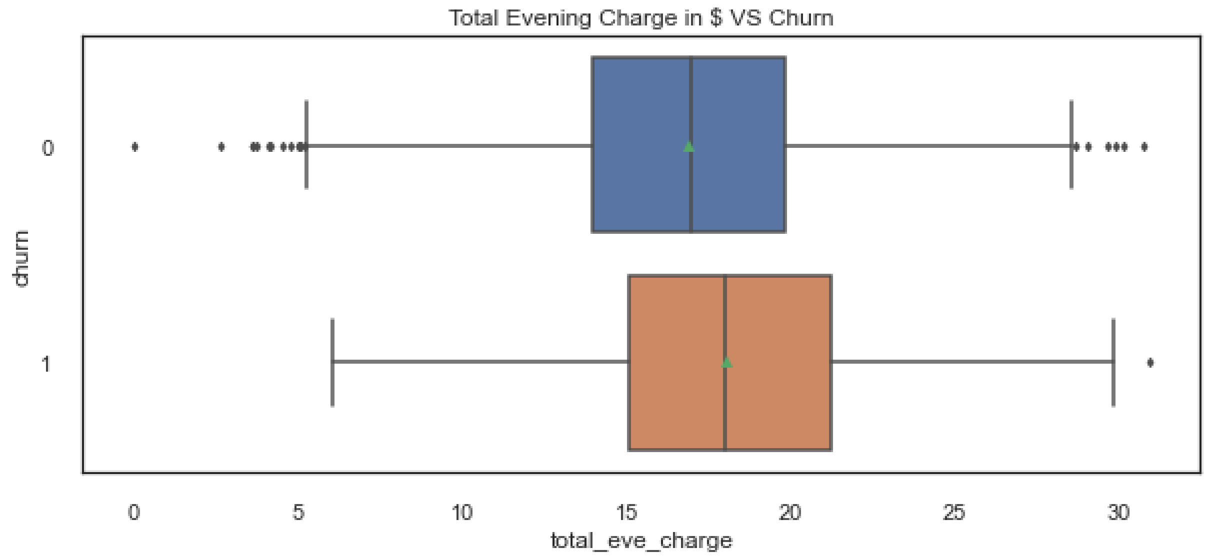
```
# We would need to visualize the data to clearly see where the data rests. Let's
from scipy import stats, linalg

fig, ax = plt.subplots(figsize=(10,4))

sns.boxplot(y = Customer_Churn['churn'], x = Customer_Churn['total_day_charge'],
plt.title('Total Day Charge VS Churn')
plt.show()

# Calculate the correlation coefficient
r, p = stats.pointbiserialr(Customer_Churn['churn'], Customer_Churn['total_day_c
print ('point biserial correlation r is %s with p = %s' %(r,p))
```




```
point biserial correlation r is 0.20515074317015197 with p = 5.300605952407281e-
33
```

In [445…

```
fig, ax = plt.subplots(figsize=(10,4))

sns.boxplot(y = Customer_Churn['churn'], x = Customer_Churn['total_eve_charge'],
plt.title('Total Evening Charge in $ VS Churn')
plt.show()

# Calculate the correlation coefficient
r, p = stats.pointbiserialr(Customer_Churn['churn'], Customer_Churn['total_eve_c
print ('point biserial correlation r is %s with p = %s' %(r,p))
```



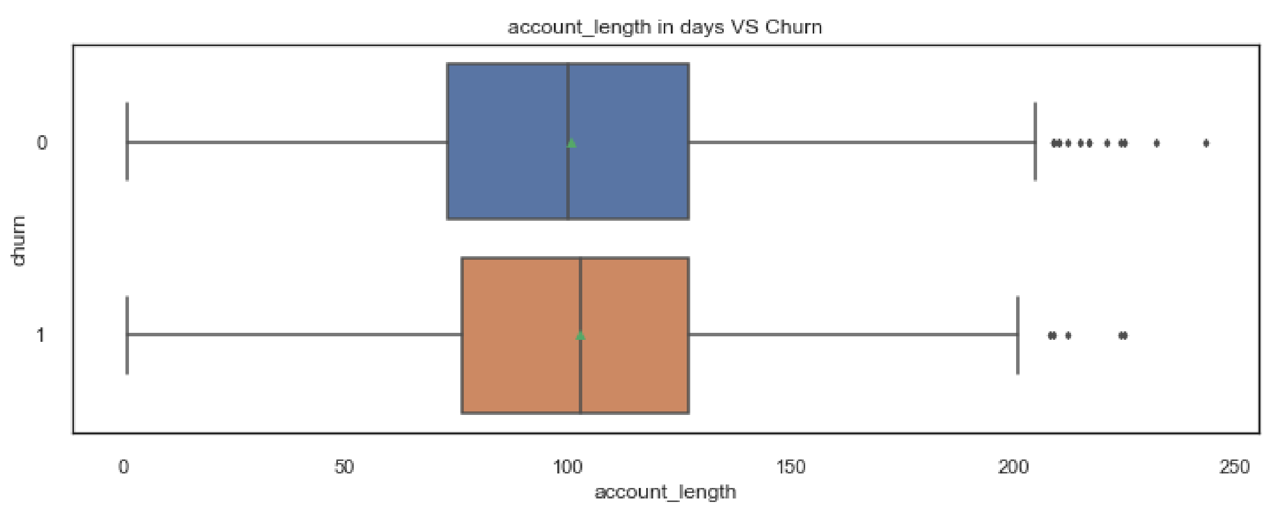

```
point biserial correlation r is 0.09278603942871282 with p = 8.036524227764227e-
08
```

Account Length

We check whether costumers leave in a certain time and whether staying longer had any relationship to churning.

In [446…

```
fig, ax = plt.subplots(figsize=(12,4))

sns.boxplot(y = Customer_Churn['churn'], x = Customer_Churn['account_length'],wi
plt.title('account_length in days VS Churn')
plt.show()

# Calculate the correlation coefficient
r, p = stats.pointbiserialr(Customer_Churn['churn'], Customer_Churn['account_len
print ('point biserial correlation r is %s with p = %s' %(r,p))
```




```
point biserial correlation r is 0.016540742243673988 with p = 0.3397600070559311
```

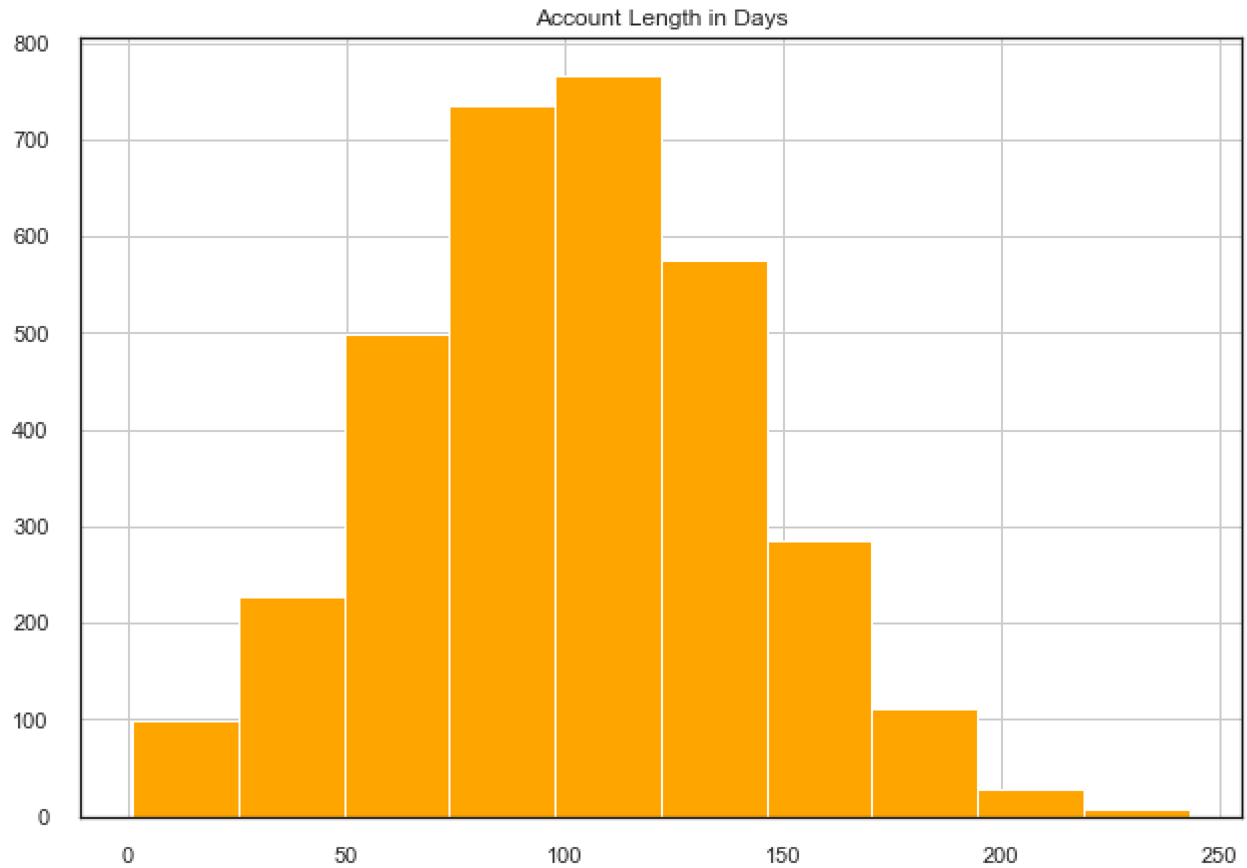
The account length data is normally distributed and the majority of the account stays from 60 days to 150.

In [447…

```
import seaborn as sns
```

```
AL_Plot= Customer_Churn.account_length.hist(color="orange", label="accountlength
plt.title('Account Length in Days')
```

Out[447… Text(0.5, 1.0, 'Account Length in Days')



The length of the accounts is normally distributed.

In [448…

```
states_dummies = pd.get_dummies(Customer_Churn["state"], prefix="STATES")
Customer_Churn = pd.concat([Customer_Churn, states_dummies], axis = 1)
Customer_Churn.head()
```

Out[448…

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_
0	KS	128	0	1	25	
1	OH	107	0	1	26	
2	NJ	137	0	0	0	
3	OH	84	1	0	0	
4	OK	75	1	0	0	

5 rows × 68 columns

Converting states into dummy variables

We convert the states into dummy variables in order for the model to be able work with the states data.

In [449…

```
Customer_Churn.drop('state', axis =1, inplace=True)
```

In [450…

```
Customer_Churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 67 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account_length          3333 non-null   int64  
 1   international_plan      3333 non-null   int64  
 2   voice_mail_plan         3333 non-null   int64  
 3   number_vmail_messages   3333 non-null   int64  
 4   total_day_calls         3333 non-null   int64  
 5   total_day_charge        3333 non-null   float64
 6   total_eve_calls         3333 non-null   int64  
 7   total_eve_charge        3333 non-null   float64
 8   total_night_minutes     3333 non-null   float64
 9   total_night_calls       3333 non-null   int64  
 10  total_night_charge      3333 non-null   float64
 11  total_intl_minutes      3333 non-null   float64
 12  total_intl_calls        3333 non-null   int64  
 13  total_intl_charge       3333 non-null   float64
 14  customer_service_calls  3333 non-null   int64  
 15  churn                   3333 non-null   int64  
 16  STATES_AK               3333 non-null   uint8  
 17  STATES_AL               3333 non-null   uint8  
 18  STATES_AR               3333 non-null   uint8  
 19  STATES_AZ               3333 non-null   uint8  
 20  STATES_CA               3333 non-null   uint8  
 21  STATES_CO               3333 non-null   uint8  
 22  STATES_CT               3333 non-null   uint8  
 23  STATES_DC               3333 non-null   uint8  
 24  STATES_DE               3333 non-null   uint8  
 25  STATES_FL               3333 non-null   uint8  
 26  STATES_GA               3333 non-null   uint8  
 27  STATES_HI               3333 non-null   uint8  
 28  STATES_IA               3333 non-null   uint8  
 29  STATES_ID               3333 non-null   uint8  
 30  STATES_IL               3333 non-null   uint8  
 31  STATES_IN               3333 non-null   uint8  
 32  STATES_KS               3333 non-null   uint8  
 33  STATES_KY               3333 non-null   uint8  
 34  STATES_LA               3333 non-null   uint8  
 35  STATES_MA               3333 non-null   uint8  
 36  STATES_MD               3333 non-null   uint8  
 37  STATES_ME               3333 non-null   uint8  
 38  STATES_MI               3333 non-null   uint8  
 39  STATES_MN               3333 non-null   uint8  
 40  STATES_MO               3333 non-null   uint8  
 41  STATES_MS               3333 non-null   uint8  
 42  STATES_MT               3333 non-null   uint8  
 43  STATES_NC               3333 non-null   uint8  
 44  STATES_ND               3333 non-null   uint8  
 45  STATES_NE               3333 non-null   uint8  
 46  STATES_NH               3333 non-null   uint8  
 47  STATES_NJ               3333 non-null   uint8  
 48  STATES_NM               3333 non-null   uint8  
 49  STATES_NV               3333 non-null   uint8  
 50  STATES_NY               3333 non-null   uint8  
 51  STATES_OH               3333 non-null   uint8  
 52  STATES_OK               3333 non-null   uint8  
```

```
 53   STATES_OR                 3333 non-null   uint8
 54   STATES_PA                 3333 non-null   uint8
 55   STATES_RI                 3333 non-null   uint8
 56   STATES_SC                 3333 non-null   uint8
 57   STATES_SD                 3333 non-null   uint8
 58   STATES_TN                 3333 non-null   uint8
 59   STATES_TX                 3333 non-null   uint8
 60   STATES_UT                 3333 non-null   uint8
 61   STATES_VA                 3333 non-null   uint8
 62   STATES_VT                 3333 non-null   uint8
 63   STATES_WA                 3333 non-null   uint8
 64   STATES_WI                 3333 non-null   uint8
 65   STATES_WV                 3333 non-null   uint8
 66   STATES_WY                 3333 non-null   uint8
dtypes: float64(6), int64(10), uint8(51)
memory usage: 582.7 KB
```

In [451…

```
Customer_Churn.head()
```

Out[451…

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_calls
0	128	0	1	25	110
1	107	0	1	26	123
2	137	0	0	0	114
3	84	1	0	0	71
4	75	1	0	0	113

5 rows × 67 columns

Labeling

Creating features, labels, training, and test data

Creating features, labels, training, and test data

In [452…

```
#  We create X and y by selecting 'churn' from the dataset and then we create a
X = Customer_Churn.drop('churn', axis=1)
y = Customer_Churn['churn']
```

Train and Test Splits

We create X and y by selecting 'churn' from the dataset and then we create an 80/20 split on the dataset for training/test. We use random_state=10 to achieve reproducible results.

In [453…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random
```

In [454…

```
#check that the split worked.
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[454…

```
((2666, 66), (2666,), (667, 66), (667,))
```

Scaling

We are scaling the data using the Standard Scaler method. Standardize the data by making the mean of the distribution zero and the majority of the data will be between -1 and 1.

In [455…

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
```

Scaling all the data to perform Cross Validation Score

In [456…

```
# We are going to scale the original X and Y data in order to use the cross vali


X_scaler = StandardScaler()

X_scaler.fit(X)

X_scaled = pd.DataFrame(X_scaler.transform(X), columns=X.columns)
```

Buiding Models

We create a function that would later look through the model classifiers and calculate the various scores to evaluate each model.

We chose to run the following classifiers for our data:

Logistic Regression K-Nearest Neighbor Decision Tree model XGboost

In [457…

```
def Train_Test_Scores(model):

    model.fit(X_train_scaled,y_train)

    print('Test_Accuracy:',  model.score(X_train_scaled,y_train))
    print('Test_Accuracy:', model.score(X_test_scaled,y_test))
    print('Recall:', precision_score(y_test,y_preds))
    print('Precision:', f1_score(y_test,y_preds))
    plot_confusion_matrix(model, X_test_scaled, y_test, cmap="Blues")
```

Logisic Regression Model

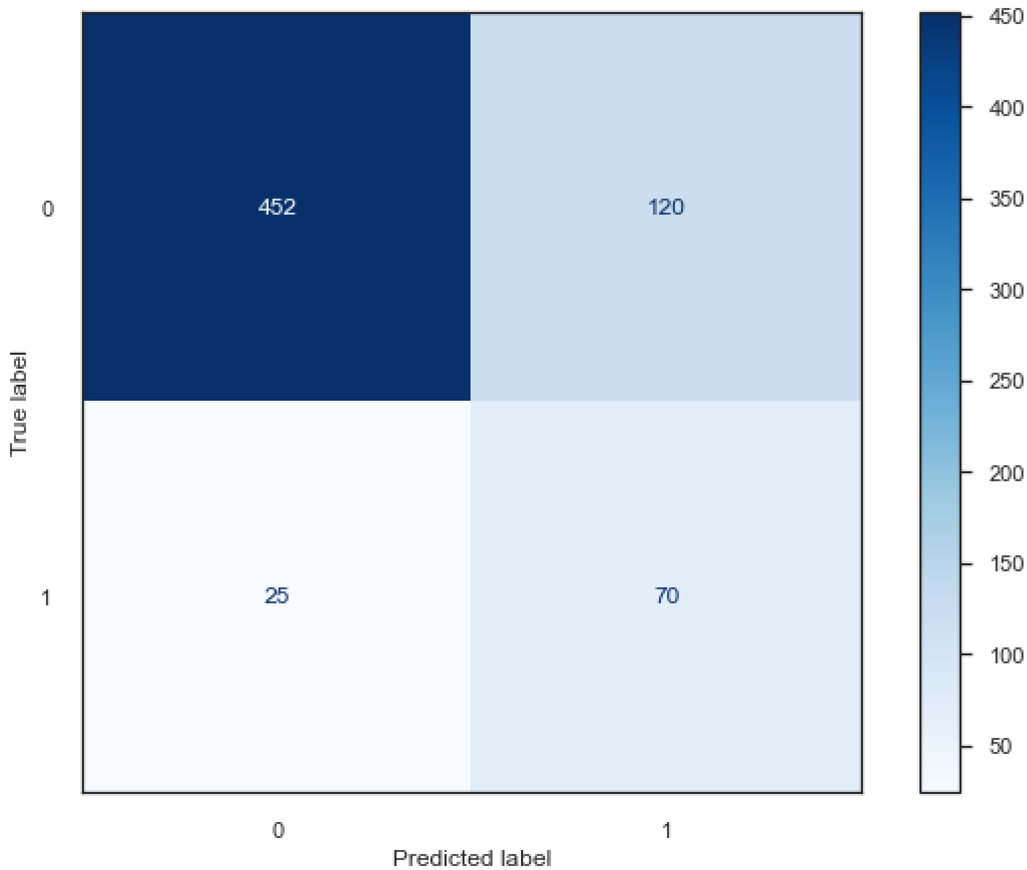
We will run the model regularly and then tune logistic regression as well.

Given the unbalance data of SyriaTel, we use class_weight='balanced') method for our Logistic Regression model. This will give weight to both the majority and minority variables.

In [458…

```
LogReg =(LogisticRegression(solver='lbfgs', class_weight='balanced'))
Train_Test_Scores(LogReg)
```

```
Test_Accuracy: 0.7790697674418605
Test_Accuracy: 0.782608695652174
Recall: 0.9487179487179487
Precision: 0.8554913294797688
```



KNN Model with SMOTE

We will use SMOTE for as our data as unbalanced.

In [459…

```
# Previous original class distribution
from imblearn.over_sampling import SMOTE
print(y_train.value_counts())

# Fit SMOTE to training data
X_train_scaled_resampled, y_train_resampled = SMOTE().fit_resample(X_train_scale

# Preview synthetic sample class distribution
print('\n')

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

```
0    2278
1     388
Name: churn, dtype: int64


0    2278
1    2278
Name: churn, dtype: int64
```

In [460…

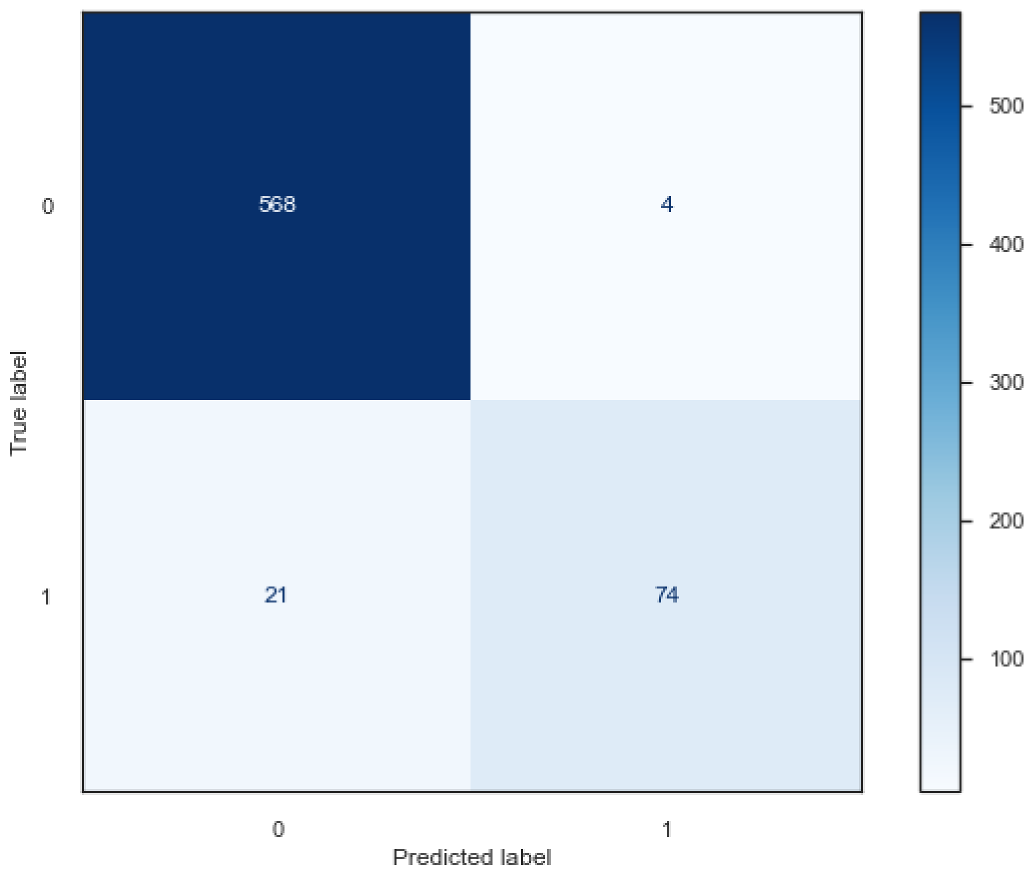
```
knn = KNeighborsClassifier()
```

```
knn.fit(X_train_scaled_resampled, y_train_resampled)

print('Train_Accuracy:', knn.score(X_train_scaled_resampled,y_train_resampled))
print('Test_Accuracy:', knn.score(X_test_scaled,y_test))
print('Recall:', recall_score(y_test,knn.predict(X_test_scaled)))
print('Precision:', precision_score(y_test,knn.predict(X_test_scaled)))
print('F1_Score:',f1_score(y_test,knn.predict(X_test_scaled)))
print('mean_CV_recall:', np.mean(cross_val_score(knn, X_scaled, y, scoring="reca

plot_confusion_matrix(model, X_test_scaled, y_test, cmap="Blues")
```

```
Train_Accuracy: 0.9102282704126426
Test_Accuracy: 0.6851574212893553
Recall: 0.5157894736842106
Precision: 0.2300469483568075
F1_Score: 0.3181818181818182
mean_CV_recall: 0.0413659793814433
```

Out[460…

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe4c111bd00
>
```



KNN Model and GridSearchCV, Hyperparameters:

We tune the KNN model using GridSearchCV and it finds the optimal parameters.

In [461…

```
param_grid = {
     'n_neighbors': list(range(1, 20, 2)),
     'weights': ['uniform','distance'],
     'metric': ['euclidean', 'manhattan'],
 }
```

```
gs_knn = GridSearchCV(knn, param_grid=param_grid, cv=5)
gs_knn.fit(X_train_scaled, y_train)

gs_knn.best_params_
```

Out[461…

```
{'metric': 'euclidean', 'n_neighbors': 11, 'weights': 'uniform'}
```

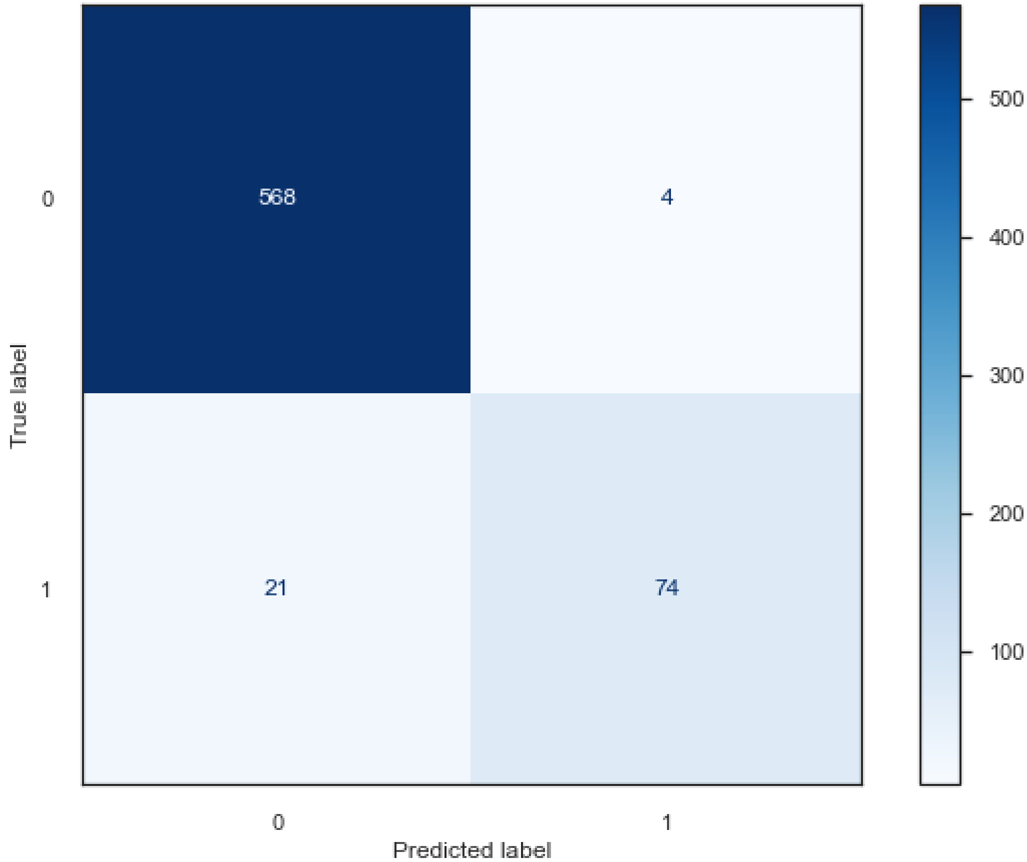
In [462…

```
best_knn = KNeighborsClassifier(n_neighbors = 13,
                                metric = 'euclidean',
                                weights = 'uniform')

best_knn.fit(X_train_scaled_resampled,y_train_resampled)

print('Train_Accuracy:', best_knn.score(X_train_scaled_resampled,y_train_resampl
print('Test_Accuracy:', best_knn.score(X_test_scaled,y_test))
print('Recall:', recall_score(y_test,best_knn.predict(X_test_scaled)))
print('Precision:',precision_score(y_test,best_knn.predict(X_test_scaled)))
print('F1_Score:',f1_score(y_test,best_knn.predict(X_test_scaled)))
('mean_CV_recall:', np.mean(cross_val_score(best_knn, X_scaled, y, scoring="reca

steps = [('scaler', StandardScaler()), ('predictor', best_knn)]
pipeline = Pipeline(steps) # define the pipeline object.

mean_cv_recall = np.mean(cross_val_score(pipeline, X_scaled, y, scoring="recall"

mean_cv = np.mean(cross_val_score(pipeline, X_scaled, y, cv = 5))

plot_confusion_matrix(model, X_test_scaled, y_test, cmap="Blues")
```

```
Train_Accuracy: 0.8430640913081651
Test_Accuracy: 0.6506746626686657
Recall: 0.5789473684210527
Precision: 0.2217741935483871
F1_Score: 0.3206997084548105
```

Out[462…

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe4b1d53c70
>
```



In [463…

```
y_test.value_counts(normalize=True)
```

Out[463…

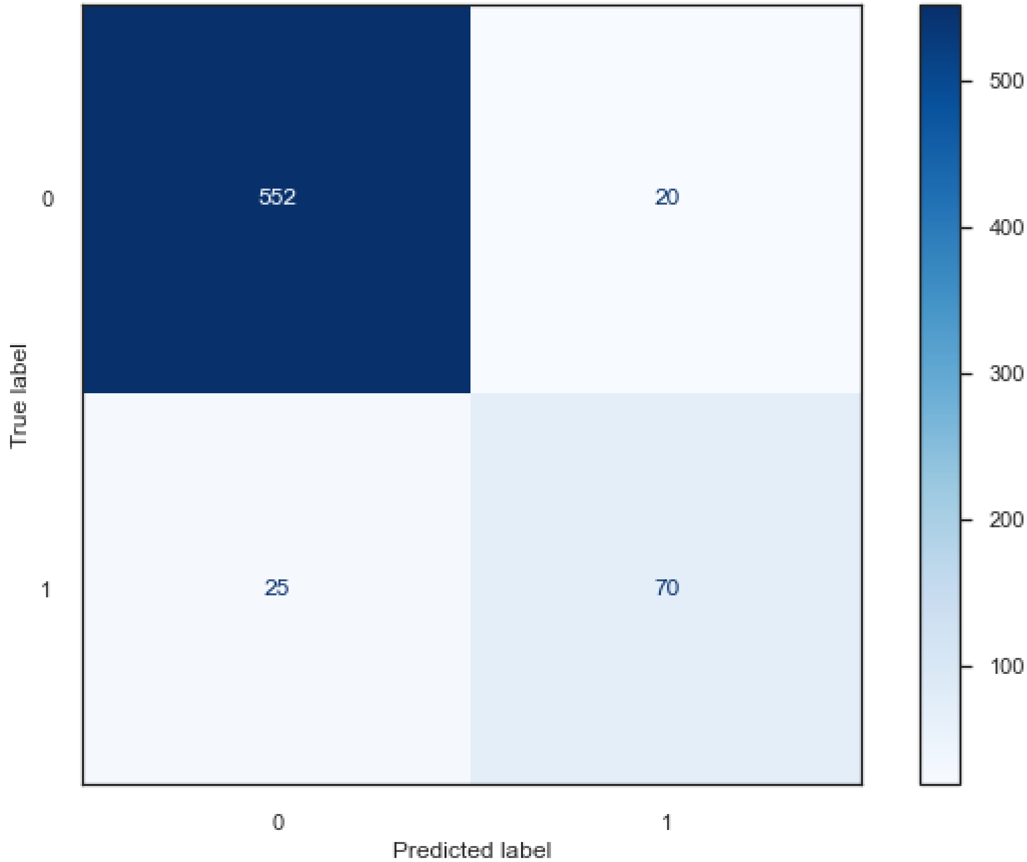
```
0    0.857571
1    0.142429
Name: churn, dtype: float64
```

Decision Tree Model

In [464…

```
DT_clf = DecisionTreeClassifier()
Train_Test_Scores(DT_clf)
```

```
Test_Accuracy: 1.0
Test_Accuracy: 0.9325337331334332
Recall: 0.9487179487179487
Precision: 0.8554913294797688
```



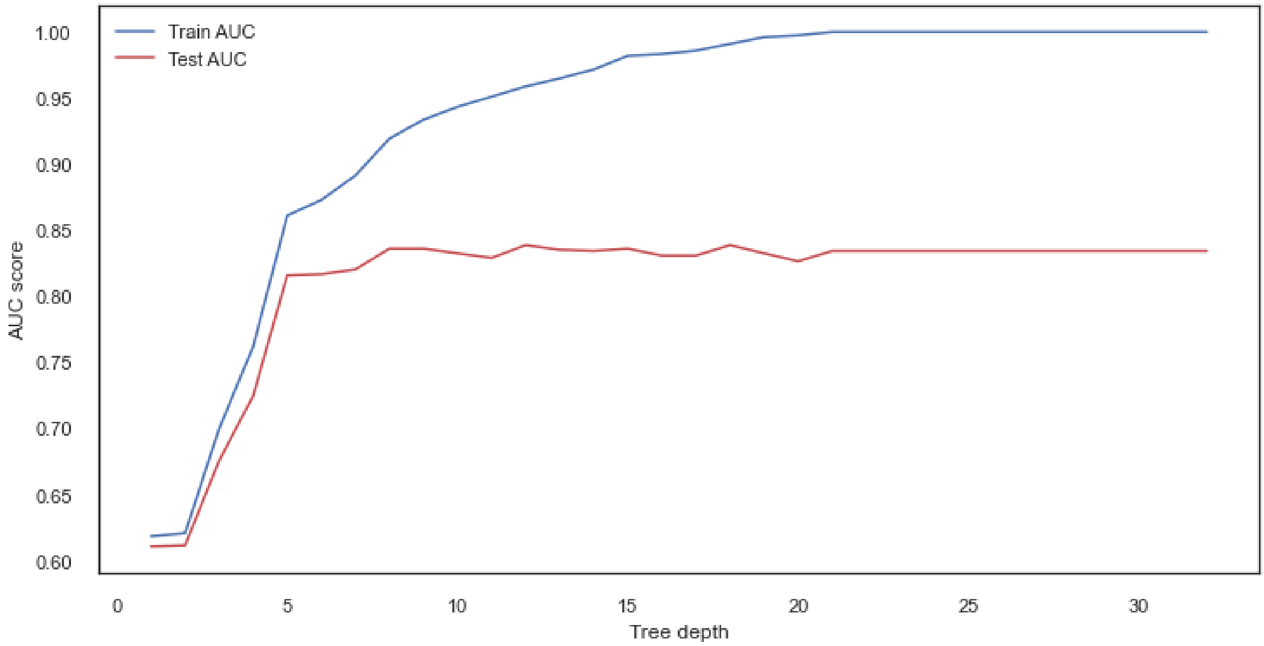
Hyperparameter Tuning and Pruning in Decision Tree

We will tune our Decision Tree classifier in order to avoid overfitting and hopefully achieve better results with our prediction.

In [465…

```
#Maximum Tree Depth

max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []
for max_depth in max_depths:
   DT = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth, random_
   DT.fit(X_train, y_train)
   train_pred = DT.predict(X_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, trai
   roc_auc = auc(false_positive_rate, true_positive_rate)


   # Adding AUC score to previous train results
   train_results.append(roc_auc)
   y_pred = DT.predict(X_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pre
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_depths, train_results, 'b', label='Train AUC')
plt.plot(max_depths, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
```

```
plt.xlabel('Tree depth')
plt.legend()
plt.show()
```

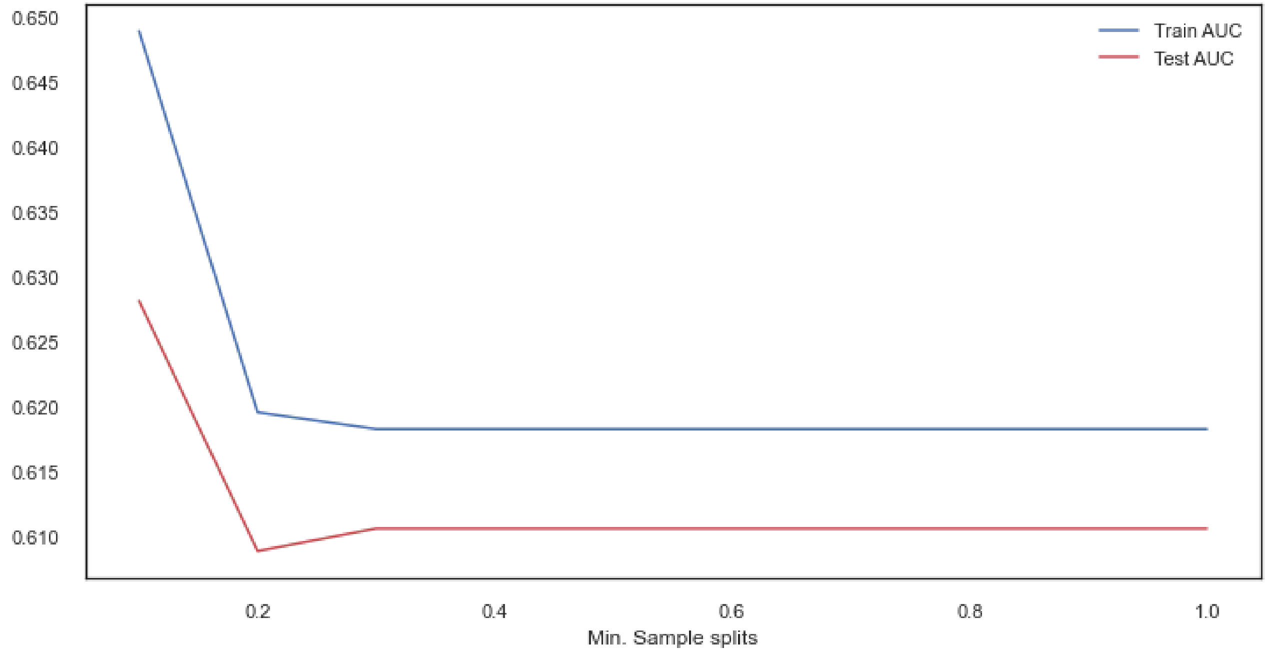


In [466…

```
# Optimum Value of max_debth is 3  - Training and test errors rise rapidly betwe
```

In [467…

```
# Minimun Split

min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
train_results = []
test_results = []
for min_samples_split in min_samples_splits:
   DT = DecisionTreeClassifier(criterion='entropy', min_samples_split=min_sample
   DT.fit(X_train, y_train)
   train_pred = DT.predict(X_train)
   false_positive_rate, true_positive_rate, thresholds =    roc_curve(y_train, t
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = DT.predict(X_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pre
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(min_samples_splits, train_results, 'b', label='Train AUC')
plt.plot(min_samples_splits, test_results, 'r', label='Test AUC')
plt.xlabel('Min. Sample splits')
plt.legend()
plt.show()
```

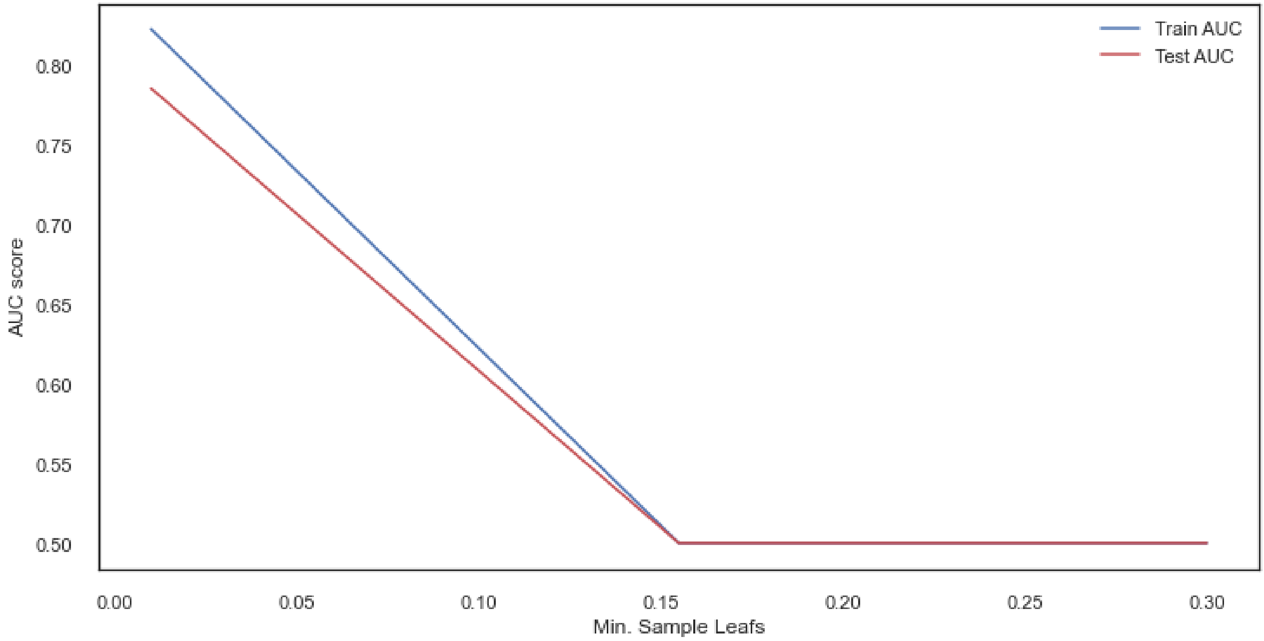


In [468…

```
# AUC for both test and train data plateaued  at 0.2
# Further increase in minimum sample split does not improve learning
```

In [469…

```
#Minimum Sample Leafs

min_samples_leafs = np.linspace(0.01, 0.3, 3, endpoint=True)
train_results = []
test_results = []
for min_samples_leaf in min_samples_leafs:
   DT = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=min_samples
   DT.fit(X_train, y_train)
   train_pred = DT.predict(X_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, trai
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = DT.predict(X_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pre
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(min_samples_leafs, train_results, 'b', label='Train AUC')
plt.plot(min_samples_leafs, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Min. Sample Leafs')
plt.legend()
plt.show()
```



In [470…

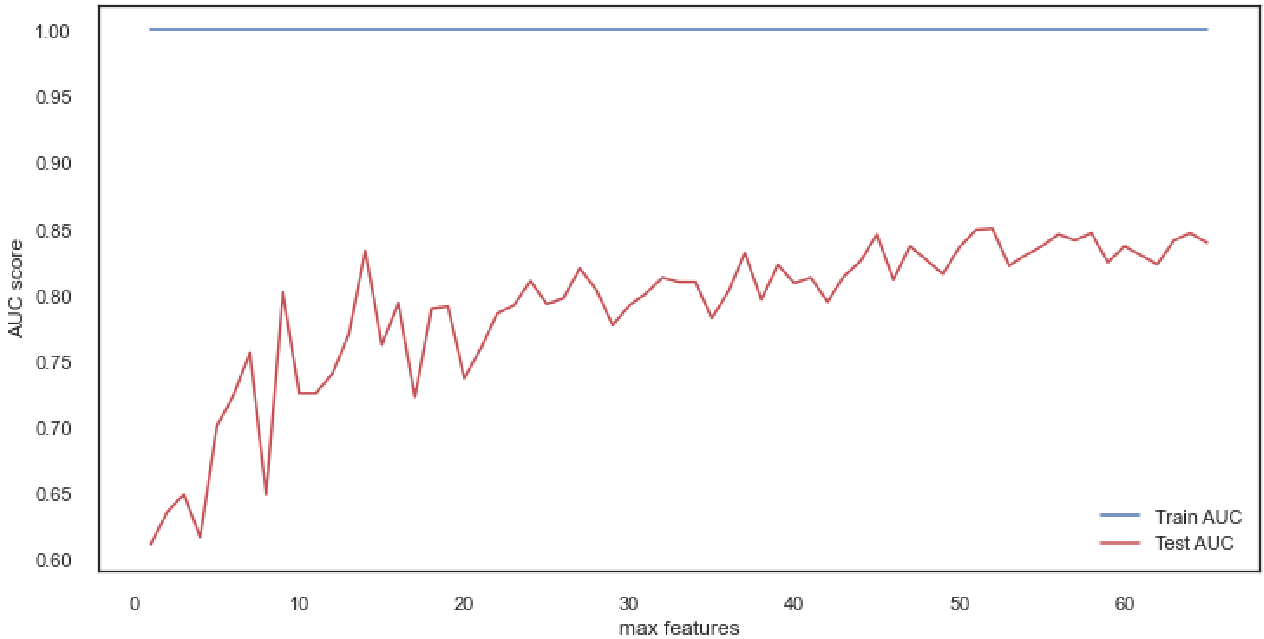
```
# AUC gives best value between 0.01
```

In [471…

```
# Find the best value for optimal maximum feature size
max_features = list(range(1, X_train.shape[1]))
train_results = []
test_results = []
for max_feature in max_features:
   DT = DecisionTreeClassifier(criterion='entropy', max_features=max_feature, ra
   DT.fit(X_train, y_train)
   train_pred = DT.predict(X_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, trai
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = DT.predict(X_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pre
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_features, train_results, 'b', label='Train AUC')
plt.plot(max_features, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('max features')
plt.legend()
```

Out[471…

```
<matplotlib.legend.Legend at 0x7fe4a50d9a90>
```



In [472…

```
# No effect on the training dataset - flat AUC
# Highest AUC value seen at 0.86
```

In [473…

```
# Train a classifier with optimal values we identified
dt_h_tuning = DecisionTreeClassifier(criterion='entropy',
                           max_features=0.86,
                           max_depth=10,
                           min_samples_split=0.001,
                           min_samples_leaf=0.0001,
                           random_state=1)
dt_h_tuning.fit(X_train_scaled, y_train)
y_pred = dt_h_tuning.predict(X_test_scaled)
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Out[473…

```
0.8421420684578579
```

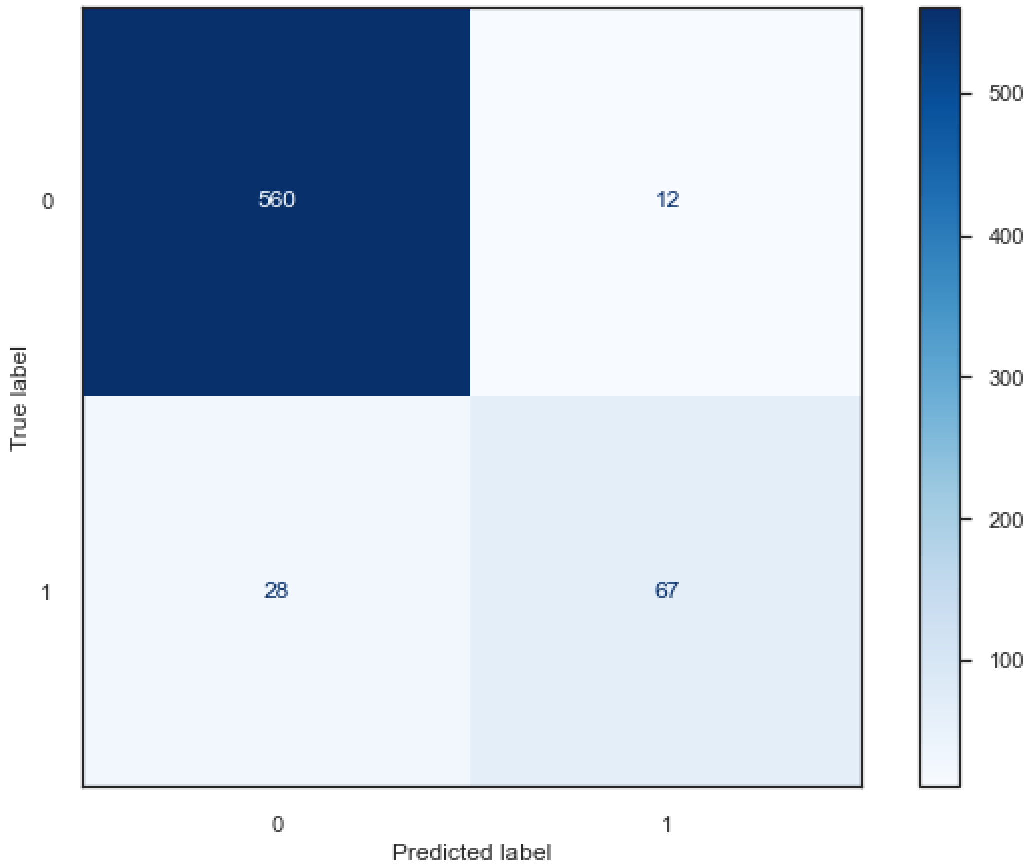
In [474…

```
# The improvement of the hyper parameter was not significant
```

In [475…

```
Train_Test_Scores(dt_h_tuning)
```

```
Test_Accuracy: 0.9831207801950488
Test_Accuracy: 0.9400299850074962
Recall: 0.9487179487179487
Precision: 0.8554913294797688
```



In [476…

```
dt_h_tuning.score(X_train_scaled, y_train)
```

Out[476… 0.9831207801950488

XGboost Model

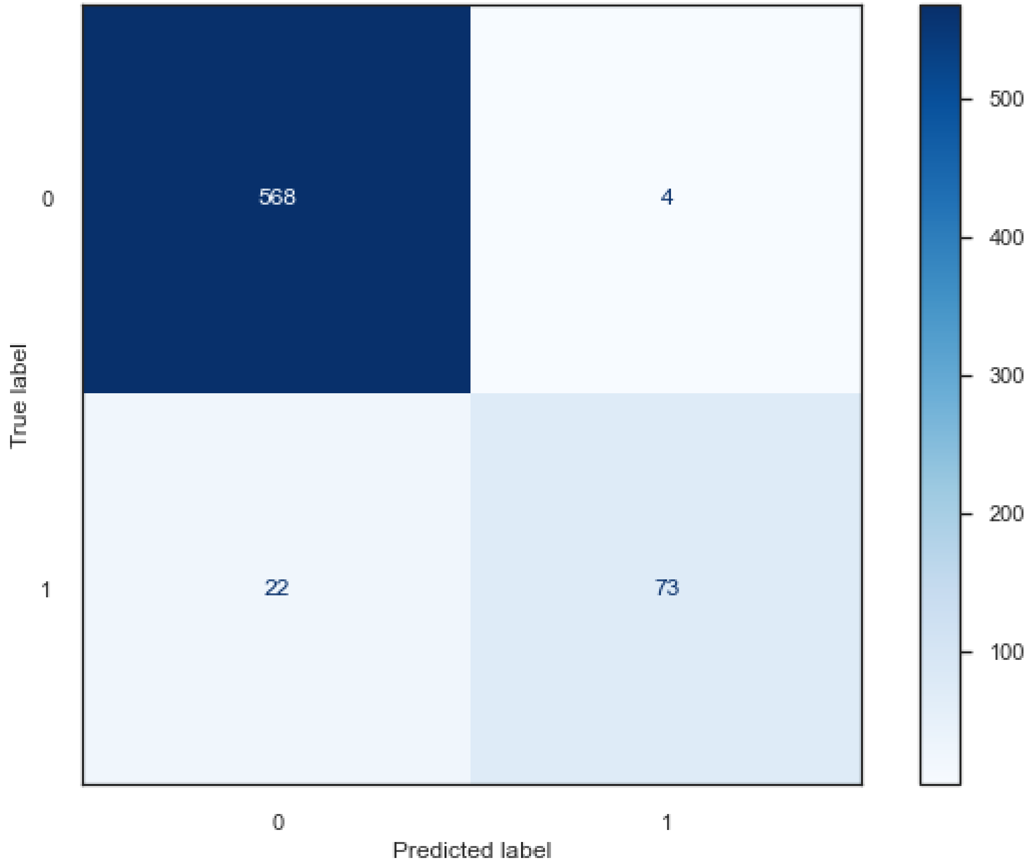
In [479…

```
XGboost_model= XGBClassifier(eval_metric='mlogloss')
```

In [480…

```
Train_Test_Scores(XGboost_model)
```

```
Test_Accuracy: 1.0
Test_Accuracy: 0.9610194902548725
Recall: 0.9487179487179487
Precision: 0.8554913294797688
```



Tuning XGBoost

We will also tune our XGboost model by applying GrisSearchCV to obtain ultimate values and we will set up restrictions for the search using “param_grid” for the purpose of time efficiency.

In [481…

```
param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
    'verbosity': [0]

}
```

In [482…

```
grid_clf = GridSearchCV(XGboost_model, param_grid, scoring='accuracy', cv=None,


best_parameters = grid_clf.param_grid

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s`: %r' % (param_name, best_parameters[param_name]))
```

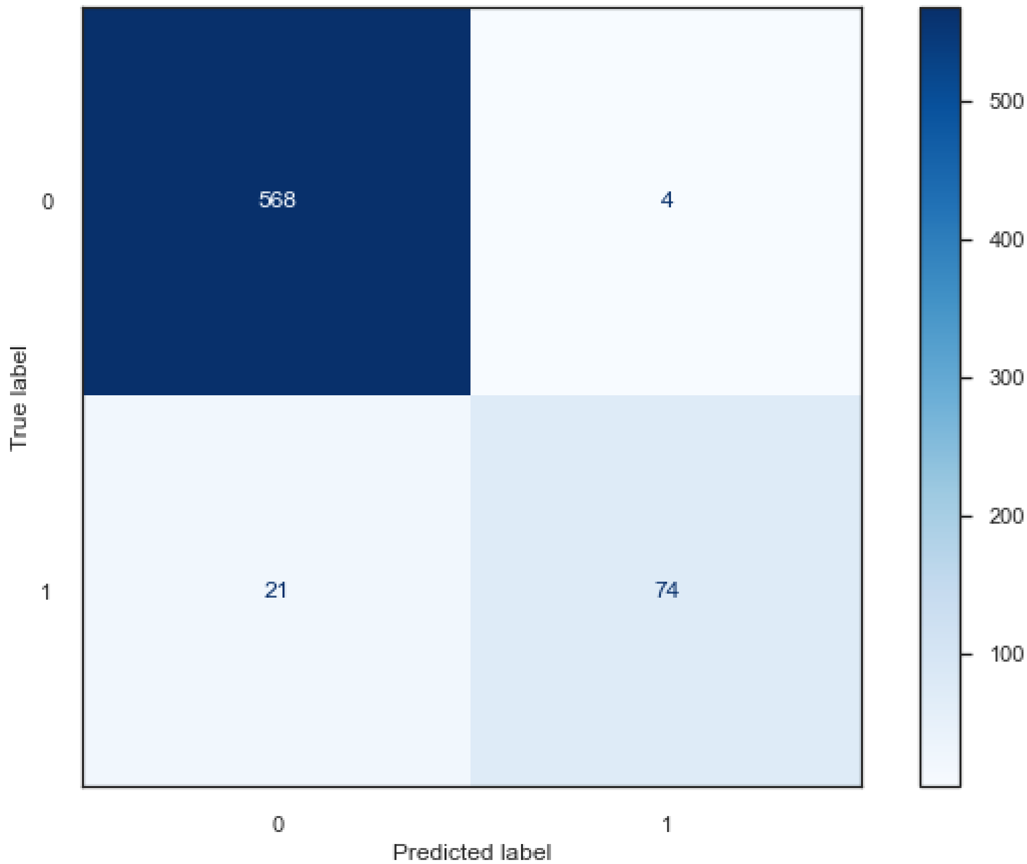
```
Grid Search found the following optimal parameters:
learning_rate`: [0.1, 0.2]
max_depth`: [6]
min_child_weight`: [1, 2]
n_estimators`: [100]
```

```
subsample`: [0.5, 0.7]
verbosity`: [0]
```

In [483…

```
Train_Test_Scores(grid_clf)
```

```
Test_Accuracy: 0.9827456864216054
Test_Accuracy: 0.9625187406296851
Recall: 0.9487179487179487
Precision: 0.8554913294797688
```



Evaluating Models

We calculate the scores for the the following model: Logistic Regression, Decision Tree and XGboost and their respective tuned classifiers. The code will loop through the models and generate data-frame to compare them again each other. We omitted kNN given the low Recall score.

In [484…

```
ALLmodels = [LogReg,DT_clf,dt_h_tuning,XGboost_model,grid_clf]

model_names = 'Logistic_reg Decision_Tree Decision_Tree_tuned XGboost XGboost_Tu

models_DataFrame = pd.DataFrame(columns=['Model','Train_Accuracy','Test_Accuracy

for (model,model_names) in zip(ALLmodels,model_names):

    print(model_names)

    model.fit(X_train_scaled, y_train)
```

```
    y_preds = model.predict(X_test_scaled)

    # Calculating the scores

    Train_Accuracy = model.score(X_train_scaled,y_train)
    Test_Accuracy = model.score(X_test_scaled,y_test)
    Precision = precision_score(y_test,y_preds)
    Recall = recall_score(y_test,y_preds)
    f1_Score = f1_score(y_test,y_preds)


    report = classification_report(y_test, y_preds)


    # Pipeline: We  calculate cross-validation score a pipeline so that data tha

    steps = [('scaler', StandardScaler()), ('predictor', model)]

    pipeline = Pipeline(steps) # define the pipeline object.

    mean_cv_recall = np.mean(cross_val_score(pipeline, X_scaled, y, scoring="rec

    mean_cv = np.mean(cross_val_score(pipeline, X_scaled, y, cv = 5))

    print(report)

    models_DataFrame  = models_DataFrame.append({'Model':model_names,'Train_Accu
```

```
Logistic_reg
              precision    recall  f1-score   support

           0       0.95      0.79      0.86       572
           1       0.37      0.74      0.49        95

    accuracy                           0.78       667
   macro avg       0.66      0.76      0.68       667
weighted avg       0.87      0.78      0.81       667

Decision_Tree
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       572
           1       0.74      0.74      0.74        95

    accuracy                           0.93       667
   macro avg       0.85      0.85      0.85       667
weighted avg       0.93      0.93      0.93       667

Decision_Tree_tuned
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       572
           1       0.85      0.71      0.77        95

    accuracy                           0.94       667
   macro avg       0.90      0.84      0.87       667
weighted avg       0.94      0.94      0.94       667
```

```
XGboost
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       572
           1       0.95      0.77      0.85        95

    accuracy                           0.96       667
   macro avg       0.96      0.88      0.91       667
weighted avg       0.96      0.96      0.96       667

XGboost_Tuned
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       572
           1       0.95      0.78      0.86        95

    accuracy                           0.96       667
   macro avg       0.96      0.89      0.92       667
weighted avg       0.96      0.96      0.96       667
```

Comparing Models

We compare the scores below for all the models. Our main focus will be to look into Recall because we will not want to miss a false negative. If the costumer left and we miss that data, it can be very costly for SyriaTel. We have more tolerance for precision because in the worst case scenario, we would offer a costumer whom we think left but didn't leave, some incentive which will hopefully increase his likelihood to stay with the company.

XGboost has showed to have the highest Ave Cross validation score. We will dig into the feature importance to get further details.

In [485…

```
models_DataFrame
```

Out[485…

	Model	Train_Accuracy	Test_Accuracy	Precision	Recall	F1_score	Mean_CV
0	Logistic_reg	0.779070	0.782609	0.368421	0.736842	0.491228	0.759079
1	Decision_Tree	1.000000	0.925037	0.736842	0.736842	0.736842	0.921392
2	Decision_Tree_tuned	0.983121	0.940030	0.848101	0.705263	0.770115	0.932493
3	XGboost	1.000000	0.961019	0.948052	0.768421	0.848837	0.954996
4	XGboost_Tuned	0.982746	0.962519	0.948718	0.778947	0.855491	0.955898

XGBoost has the highest Mean CV Score

In [487…

```
# Checking which model received the highest score - focsuing on recall.
AvgCrossValRecall = models_DataFrame["Mean_CV_Recall"]
max_value = AvgCrossValRecall.max()
print(max_value)
```

```
0.7597723367697593
```

In [488…

```
# XGboost received the highest score.
```

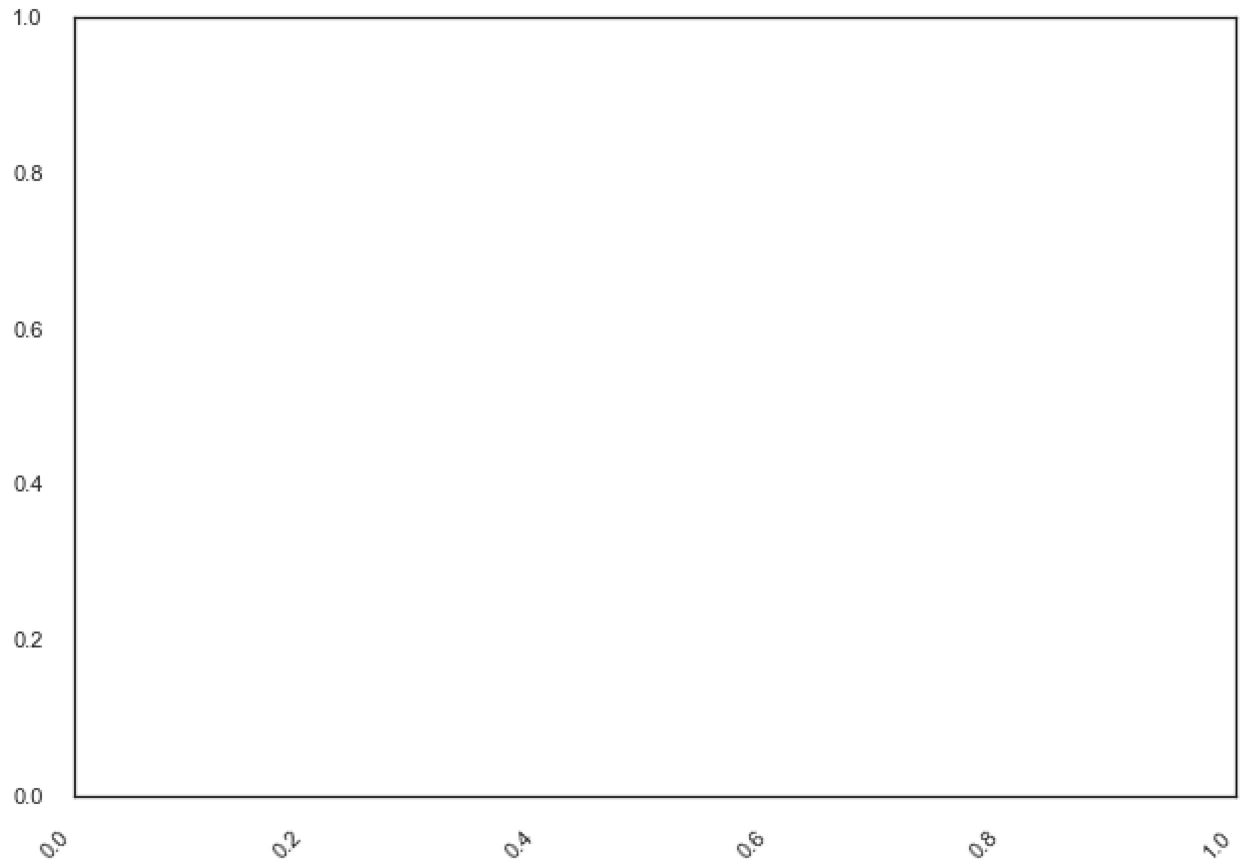
Plotting the results

In [489…

```
x_plot = models_DataFrame["Model"]
y_plot = models_DataFrame["Mean_CV_Recall"]
```

In [490…

```
fig = plt.figure()
ax.bar(x_plot,y_plot, color ='g')
plt.xticks(rotation=45, ha="right")
ax.set_ylabel('Average Cross Validation Score')
ax.set_title('Models')
plt.show()
```



Receiver Operating Characteristic ("ROC")

ROC Curve presents the trade-off among the true positive rate and false positive rate for the XGBoost model using different probability thresholds.

In [491…

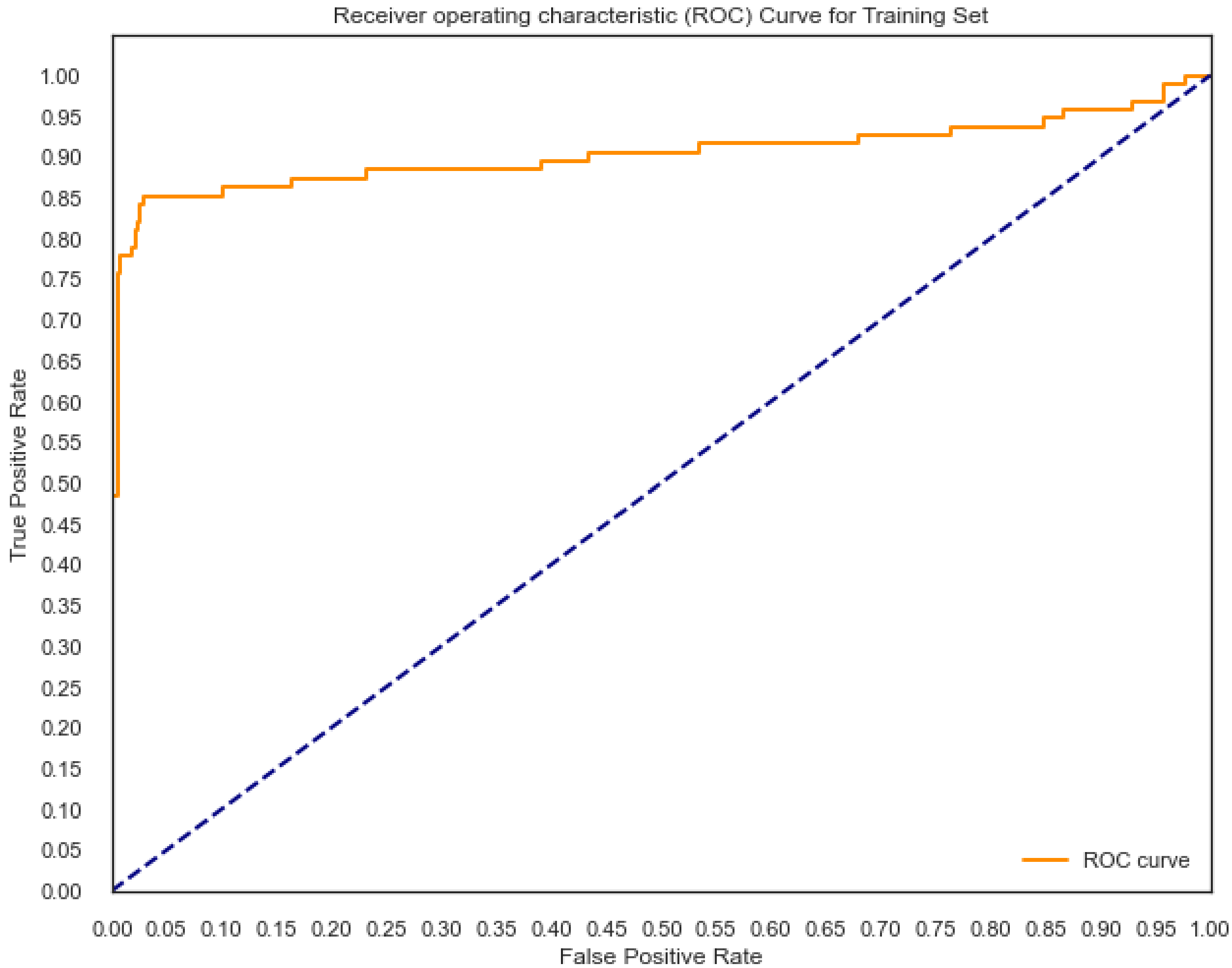
```
# Generate a no skill prediction (majority class)
from matplotlib import pyplot
import matplotlib.pyplot as plt
%matplotlib inline

ns_probs = [0 for _ in range(len(y_test))]

#for XG Boost

XGboost_model= XGBClassifier()

XGboost_model.fit(X_train_scaled, y_train)
```

```
# predict probabilities
XG_probs = XGboost_model.predict_proba(X_test_scaled)


# keep probabilities for the positive outcome only
XG_probs = XG_probs[:, 1]

# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
XG_auc = roc_auc_score(y_test, XG_probs)


# summarize scores


print('XGBoost:  AUC=%.3f' % (XG_auc))



# calculate roc curves
XG_fpr, XG_tpr, _ = roc_curve(y_test, XG_probs)


# ROC curve for training set
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(XG_fpr, XG_tpr, color='darkorange',
         lw=lw, label='ROC curve')

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])


plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve for Training Set')
plt.legend(loc='lower right')
```

```
XGBoost:  AUC=0.904
```

Out[491…

```
<matplotlib.legend.Legend at 0x7fe488951730>
```

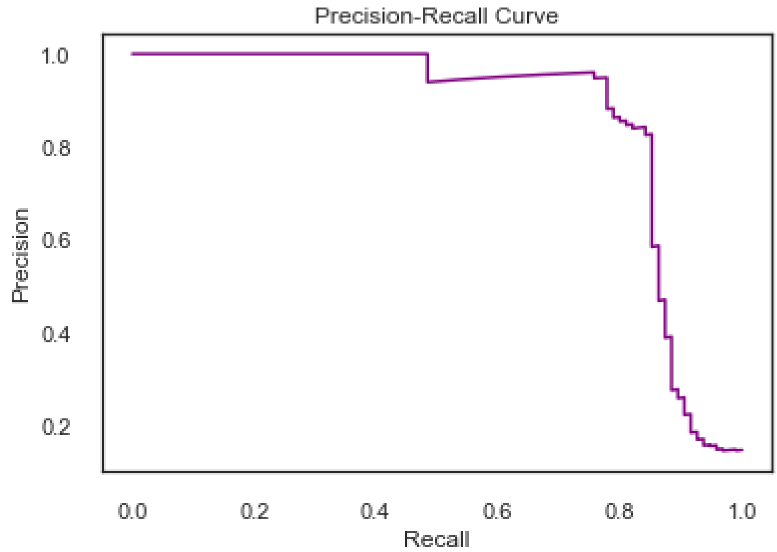


Precision-Recall Curve ("PRC")

We also show the precision-recall curve as it is more appropriate for imbalanced data that we are dealing with. We can see in the graph the trade-off among the true positive and the predictive positive value for our XGboost model using various probability thresholds. As we mentioned above, we are more focused on having a higher recall without giving up too much on precision. Roughly around 90%, precision is a little north of 80% and this is a feasible trade-off for our model.

In [492…

```
precision, recall, thresholds = precision_recall_curve(y_test, XG_probs)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```



Permutation of Importance

Permutation of importance randomly shuffle each feature and compute the perfromace of the model. The features which impact the performance with the highest score are the ones we SyriaTel should focus on.

In [493…

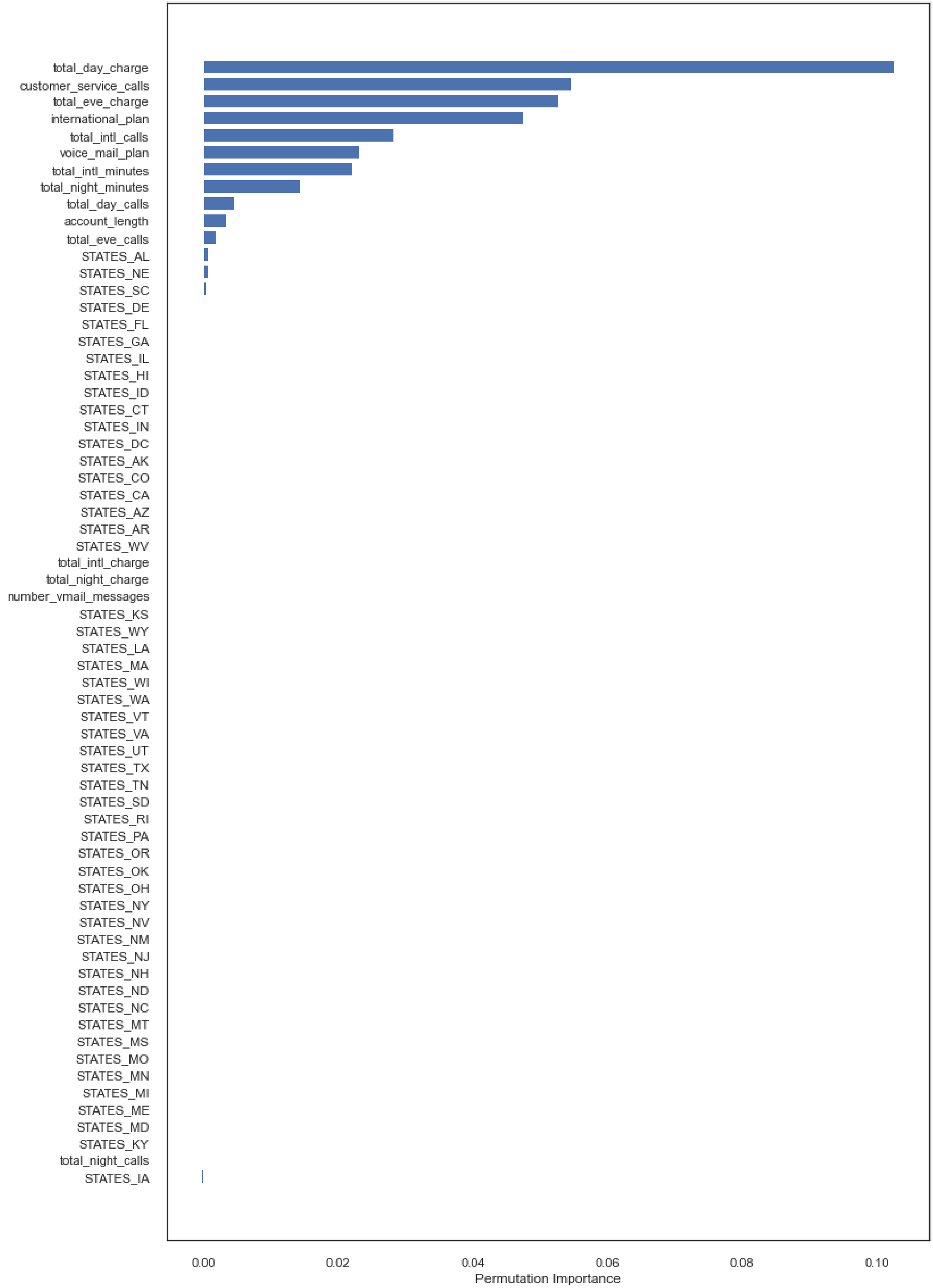
```
perm_importance = permutation_importance(XGboost_model, X_test_scaled, y_test)
```

In [494…

```
sorted_idx = perm_importance.importances_mean.argsort()
plt.figure(figsize=(12,20))
plt.barh(feature_names[sorted_idx], perm_importance.importances_mean[sorted_idx]
plt.xlabel("Permutation Importance")
```

Out[494…

```
Text(0.5, 0, 'Permutation Importance')
```



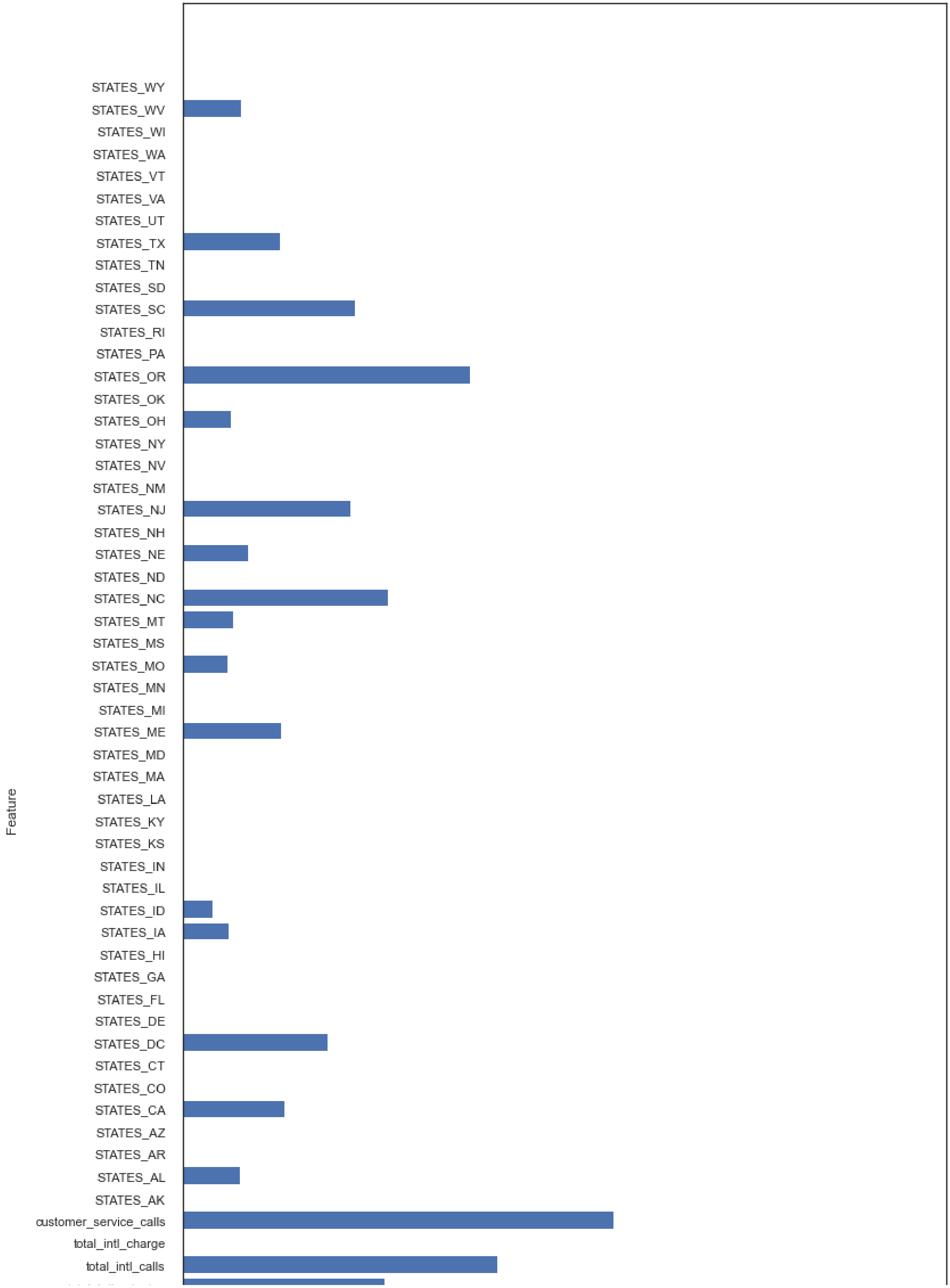
In [495…

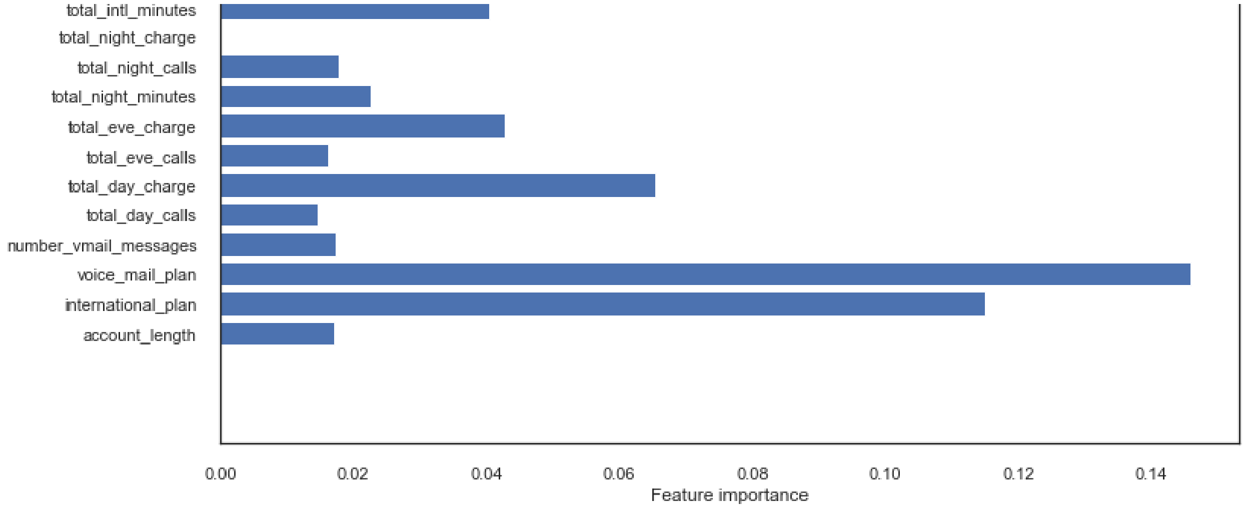
```
def plot_feature_importances(model):
    n_features = X_train_scaled.shape[1]
    plt.figure(figsize=(12,26))
    plt.barh(range(n_features), model.feature_importances_, align='center')
```

```
    plt.yticks(np.arange(n_features), X_train_scaled.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

In [496…

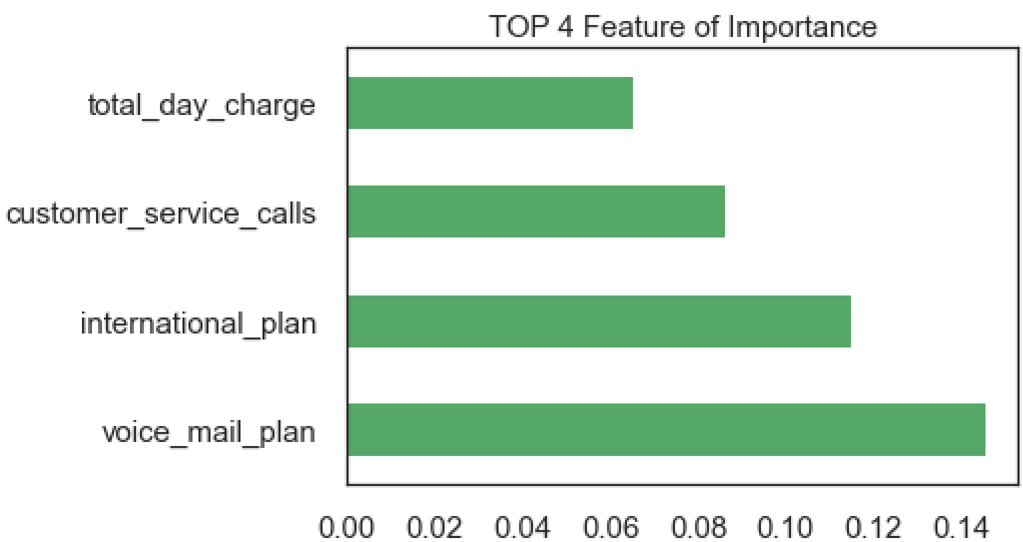
```
plot_feature_importances(XGboost_model)
```





In [497…

```
model_1 = XGboost_model

(pd.Series(model_1.feature_importances_, index=X.columns)
   .nlargest(4)
   .plot(kind='barh',color='g'))
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('TOP 4 Feature of Importance',fontsize=15)
fig.savefig('TOP_4.jpg')
```

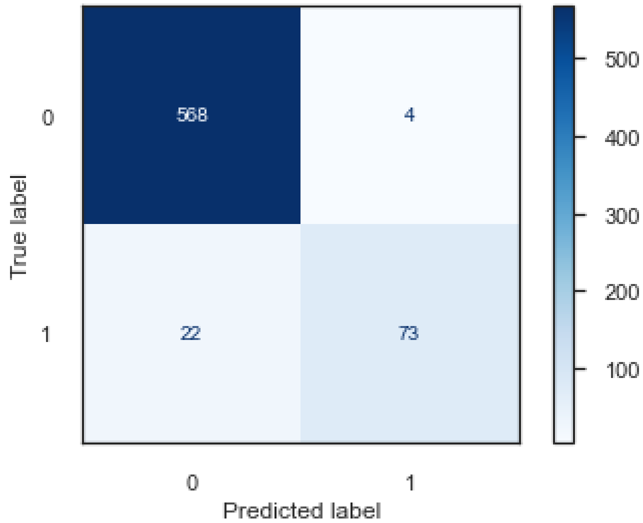


In [498…

```
plot_confusion_matrix(XGboost_model, X_test_scaled, y_test, cmap="Blues")
```

Out[498…

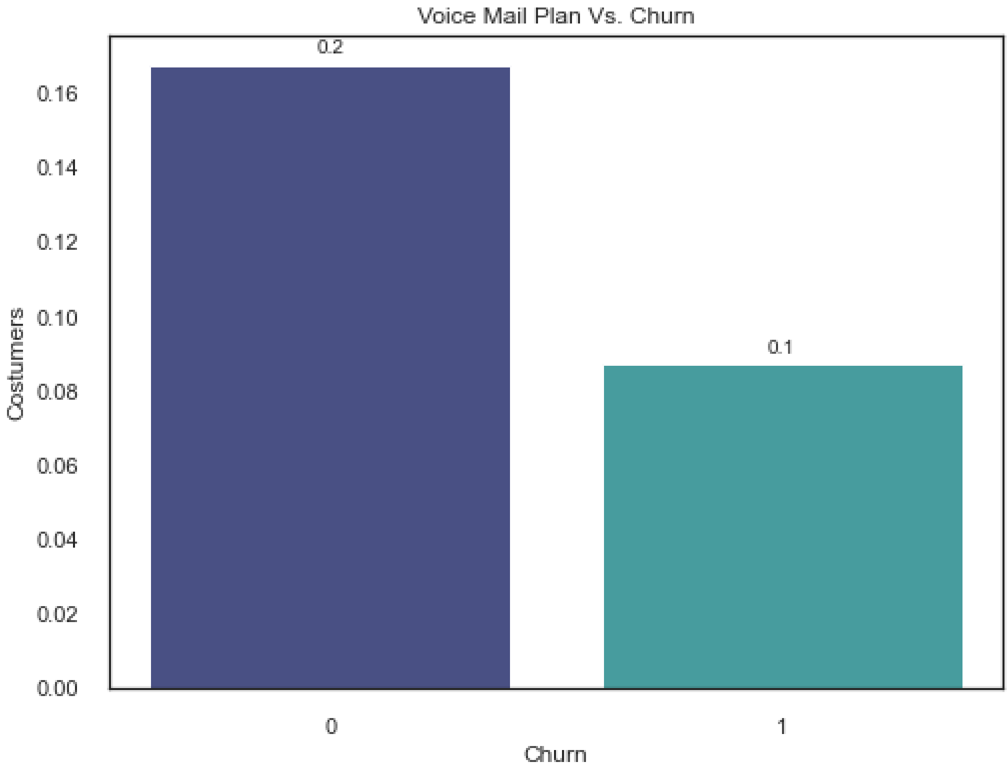
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe4c1d0fd90
>
```



Voicemail Plan

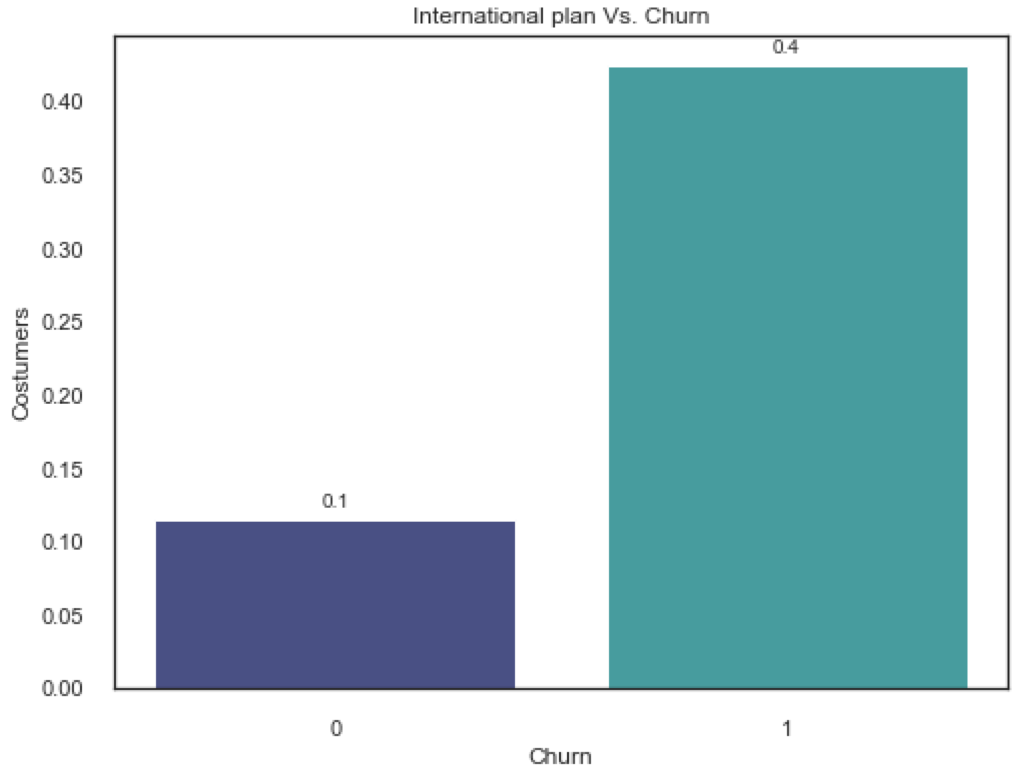
In [501…

```
plt.figure(figsize=(8, 6))
splot = sns.barplot(x='voice_mail_plan', y='churn',
                    data=Customer_Churn, palette='mako', ci=None)


for p in splot.patches:

    splot.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
plt.title('Voice Mail Plan Vs. Churn')

#plt.legend(title='Churn', prop={'size': 12}, title_fontsize=30)


#plt.figure(figsize=(12,6))

plt.plot()
plt.plot()

plt.ylabel('Costumers')
plt.xlabel('Churn')

plt.show()
```



In [502…

```
plt.figure(figsize=(8, 6))
splot = sns.barplot(x='international_plan', y='churn',
                    data=Customer_Churn, palette='mako', ci=None)


for p in splot.patches:

    splot.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
plt.title('International plan Vs. Churn')

#plt.legend(title='Churn', prop={'size': 12}, title_fontsize=30)


#plt.figure(figsize=(12,6))

plt.plot()
plt.plot()

plt.ylabel('Costumers')
plt.xlabel('Churn')

plt.show()
```



Features of Importance

Voicemail Plan

We found a voicemail plan stood out as one of the most important features. As seen in the graph, people with a voicemail plan are twice as less likely to churn.

Therefore, we recommend offering voicemail plans to customers who do not have them as part of the incentives used to retain customers. Perhaps when a customer calls the second or third time, SyrianTel can offer them a voicemail plan as a promotion if they don't currently have one.

International Plan

An international plan was also an important feature. Customers who had an international plan were four times as likely to churn. This is an element SyriaTel should focus on. Perhaps they could consider eliminating this specific plan and offer one reoccurring plan for all.

As expected, Customer Service calls were shown to be an important feature. As we see above, customers are five times more likely to churn after the third call. This supports our suggestion of offering an incentive to stay after the second and third call. SyriaTel can offer three weeks free of charge before subscribing for a year or as mentioned above, gift a customer a voicemail plan for three weeks as well.

States

In addition to the states mentioned above, Oregon (OR) should be flagged, as it came out to be an important feature. Customer Service should be aware of the states that customers are calling from. We recommend exploring the possibility of partnering with other companies. For instance

– if a customer from Oregon calls the second time and already has a voicemail plan, one incentive could be to offer a gift from another vendor such as Uber EATS – e.g. a $10 credit to order food which might incentivize the client to stay.

Next Step

- We would like to gather more data on the specific dates of churning. Ideally, we would be able to look at an individual account and learn the dates of a company subscribing and subsequently leaving.
- Allowing us to look closely into customer satisfaction could be useful by offering a survey once a Customer Service call is complete. Perhaps also closely examining how long a customer waited before his request was satisfied would be beneficial.
- We will examine whether a flat fee per month would be more cost-effective than a reoccurring monthly charge with a certain number of minutes.
- Additionally, we would like to consider using a different vendor or temporally partnering to offer incentives and promotions when a customer seems dissatisfied may increase satisfaction and reduce churning.
- Ultimately, we will implement the new features to see whether churning was reduced and calculate the cost of retaining the customers.

Thank you 😀