Business Understanding

In the current modern world and competitive landscape, understanding and effectively managing customer churn in paramount for business aiming to sustain growth, customer retention and profitability.A churn predictive model serves as a valuable tool in the endeavour by providing insights into customer behaviour and predicting which customers are likely to discontinue their relationship with the company. Some of the key variable features that will be used to creative a business churn predictive model include: factors that contribute to customer churn, allocation of resources, intervetion needed and lifetime of a customer in the business.

Factors contributing to Churn: This can be achived by analyzing historical business data and interactions with the customer to identify the key factor that result to customer churn.These include, customer patterns, demographics,customer satisfaction, metrics of engagement, history of purchases and interaction with the business. The understanding of this factors will give guideline to businesses on the strategies to put in place to address customer preferences and needs.

Allocation of Resources: A churn model will enable the business to focus more resources on specific arears which in result leads to optimization and more priority is giving to high risk clients. These resources could be time, budget, human capital among others.

Lifetime value: Understanding customer behaviour and churn dynamics will assist the business to predict and maximize on customer retention and identify new opportunities for purchases while maximizing on profits. Additionally, by identifying customers with the likelihood of churning, the business will created targeted efforts to mitigate such occurences through enhancing customer support, or introducing new services or products to improve on customer loyalty and satisfaction.

Problem Statement

Customer churn remains a challenge for most business in different sectors that contributes not only to decline in customer base but also in revenue loss.Therefore, to enable business mitigate and take control of their customers, it is crucial to create a predictive churn model that gives businesses a heads up and allows them to intervene before losing a customer. The objective of this project is to give insights for strategizing and forecasting for growth of business through retention and acquisition of customers.

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

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression,
LogisticRegressionCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import
GradientBoostingClassifier,RandomForestClassifier,AdaBoostClassifier,
BaseEnsemble
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, make_scorer,
recall_score, ConfusionMatrixDisplay, precision_score,
accuracy_score,f1_score,roc_auc_score,roc_curve
from sklearn.metrics import mean_squared_error
import warnings
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
```

Exploratory Data Analysis

```
df = pd.read_csv("churndataset.csv")
df

     state  account length  area code phone number international plan  
\
0       KS             128        415     382-4657                 no

1       OH             107        415     371-7191                 no

2       NJ             137        415     358-1921                 no

3       OH              84        408     375-9999                yes

4       OK              75        415     330-6626                yes

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

3328    AZ             192        415     414-4276                 no

3329    WV              68        415     370-3271                 no

3330    RI              28        510     328-8230                 no

3331    CT             184        510     364-6381                yes

3332    TN              74        415     400-4344                 no


     voice mail plan  number vmail messages  total day minutes  \
0                yes                     25              265.1
```

```
1                  yes                     26              161.6
2                   no                      0              243.4
3                   no                      0              299.4
4                   no                      0              166.7
...                ...                    ...                ...
3328               yes                     36              156.2
3329                no                      0              231.1
3330                no                      0              180.8
3331                no                      0              213.8
3332               yes                     25              234.4

      total day calls  total day charge  ...  total eve calls  \
0                 110             45.07  ...               99
1                 123             27.47  ...              103
2                 114             41.38  ...              110
3                  71             50.90  ...               88
4                 113             28.34  ...              122
...               ...               ...  ...              ...
3328               77             26.55  ...              126
3329               57             39.29  ...               55
3330              109             30.74  ...               58
3331              105             36.35  ...               84
3332              113             39.85  ...               82

      total eve charge  total night minutes  total night calls  \
0                16.78                244.7                 91
1                16.62                254.4                103
2                10.30                162.6                104
3                 5.26                196.9                 89
4                12.61                186.9                121
...                ...                  ...                ...
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
0                  11.01                10.0                 3
1                  11.45                13.7                 3
2                   7.32                12.2                 5
3                   8.86                 6.6                 7
4                   8.41                10.1                 3
...                  ...                 ...               ...
3328               12.56                 9.9                 6
3329                8.61                 9.6                 4
3330                8.64                14.1                 6
3331                6.26                 5.0                10
3332               10.86                13.7                 4
```

```
      total intl charge  customer service calls  churn
0                  2.70                       1  False
1                  3.70                       1  False
2                  3.29                       0  False
3                  1.78                       2  False
4                  2.73                       3  False
...                 ...                     ...    ...
3328               2.67                       2  False
3329               2.59                       3  False
3330               3.81                       2  False
3331               1.35                       2  False
3332               3.70                       0  False

[3333 rows x 21 columns]
```

```
df.shape
```

```
(3333, 21)
```

```
df.describe().T
```

```
                         count        mean        std     min     25%
50%  \
account length          3333.0  101.064806  39.822106    1.00   74.00
101.00
area code               3333.0  437.182418  42.371290  408.00  408.00
415.00
number vmail messages   3333.0    8.099010  13.688365    0.00    0.00
0.00
total day minutes       3333.0  179.775098  54.467389    0.00  143.70
179.40
total day calls         3333.0  100.435644  20.069084    0.00   87.00
101.00
total day charge        3333.0   30.562307   9.259435    0.00   24.43
30.50
total eve minutes       3333.0  200.980348  50.713844    0.00  166.60
201.40
total eve calls         3333.0  100.114311  19.922625    0.00   87.00
100.00
total eve charge        3333.0   17.083540   4.310668    0.00   14.16
17.12
total night minutes     3333.0  200.872037  50.573847   23.20  167.00
201.20
total night calls       3333.0  100.107711  19.568609   33.00   87.00
100.00
total night charge      3333.0    9.039325   2.275873    1.04    7.52
9.05
total intl minutes      3333.0   10.237294   2.791840    0.00    8.50
10.30
total intl calls        3333.0    4.479448   2.461214    0.00    3.00
```

```
4.00
total intl charge       3333.0    2.764581   0.753773    0.00    2.30
2.78
customer service calls  3333.0    1.562856   1.315491    0.00    1.00
1.00

                            75%     max
account length           127.00  243.00
area code                510.00  510.00
number vmail messages     20.00   51.00
total day minutes        216.40  350.80
total day calls          114.00  165.00
total day charge          36.79   59.64
total eve minutes        235.30  363.70
total eve calls          114.00  170.00
total eve charge          20.00   30.91
total night minutes      235.30  395.00
total night calls        113.00  175.00
total night charge        10.59   17.77
total intl minutes        12.10   20.00
total intl calls           6.00   20.00
total intl charge          3.27    5.40
customer service calls     2.00    9.00
```

```
df.isna().sum()
```

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

The data set does not have null values

```
df.duplicated().sum()
```

```
0
```

```
df.info()
```

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

The target variable is 'churn' which is categorical. Most features in our data are numerical excluding 'state', 'international plan', 'voice_mail_plan' and 'phone number' that are strings. Dropping the phone number column which is not required in our analysis.

```
df.drop(['phone number'], axis = 1, inplace = True)
```

```
df.mean()
```

```
account length             101.064806
area code                  437.182418
number vmail messages        8.099010
total day minutes          179.775098
total day calls            100.435644
total day charge            30.562307
total eve minutes          200.980348
```

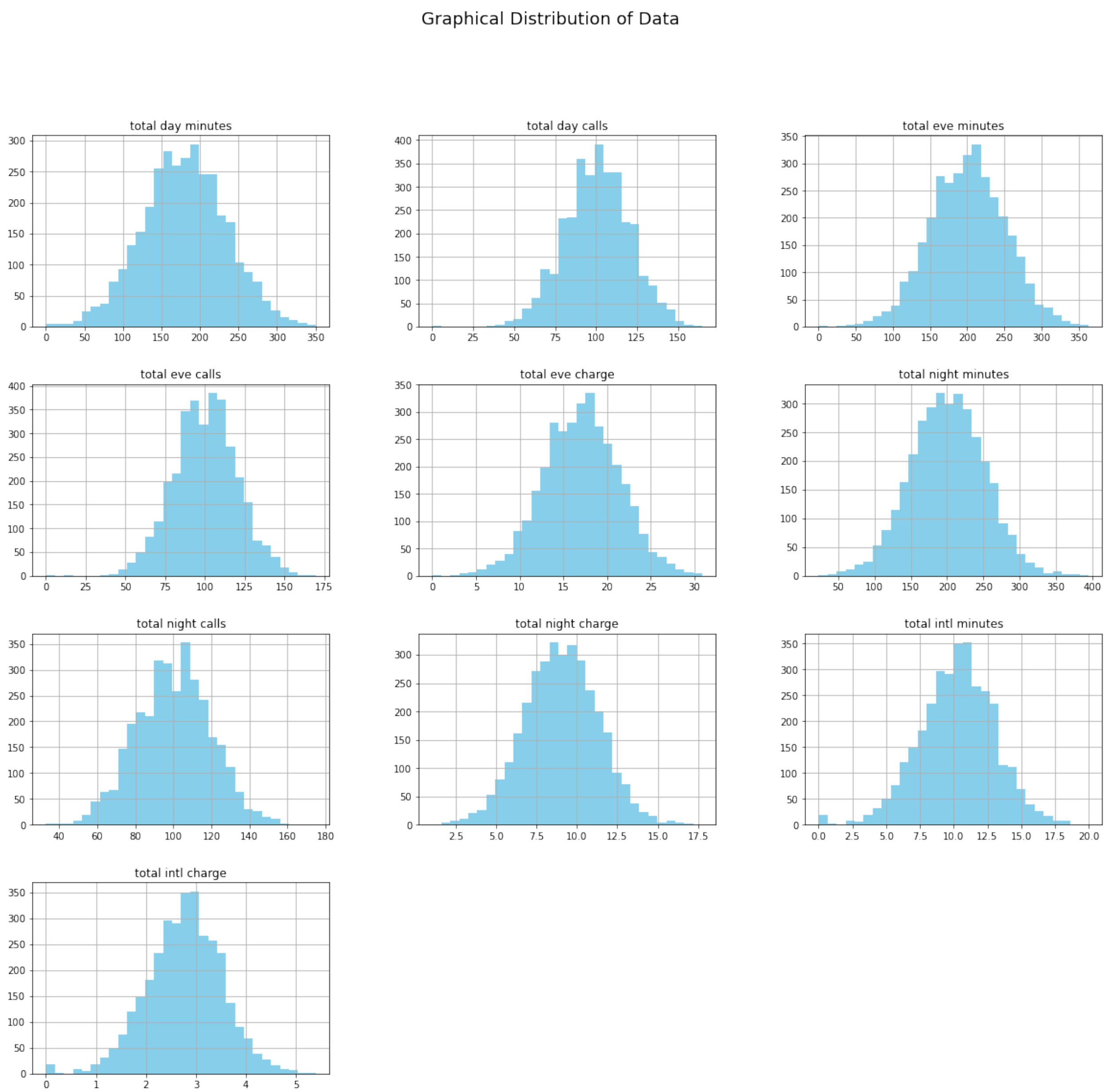
```
total eve calls              100.114311
total eve charge              17.083540
total night minutes          200.872037
total night calls            100.107711
total night charge             9.039325
total intl minutes            10.237294
total intl calls               4.479448
total intl charge              2.764581
customer service calls         1.562856
churn                          0.144914
dtype: float64
```

```
df.median()
```

```
account length               101.00
area code                    415.00
number vmail messages          0.00
total day minutes            179.40
total day calls              101.00
total day charge              30.50
total eve minutes            201.40
total eve calls              100.00
total eve charge              17.12
total night minutes          201.20
total night calls            100.00
total night charge             9.05
total intl minutes            10.30
total intl calls               4.00
total intl charge              2.78
customer service calls         1.00
churn                          0.00
dtype: float64
```

The mean and the median of our data is almost the same, meaning that our data is normally distributed.

```
# Checking the graphical presentation to see if it is normally
distributed.
columns_to_check = ['total day minutes', 'total day calls',
                    'total eve minutes','total eve calls',
                    'total eve charge','total night minutes',
                    'total night calls','total night charge',
                    'total intl minutes','total intl charge']

#Creating subplots
df[columns_to_check].hist(figsize=(20,18), color='skyblue', bins=30)
plt.suptitle('Graphical Distribution of Data', fontsize=18)
plt.show()
```



Most of our data is normally distributed meaning that most data is clustered near the mean.

Exploratory Data Analysis

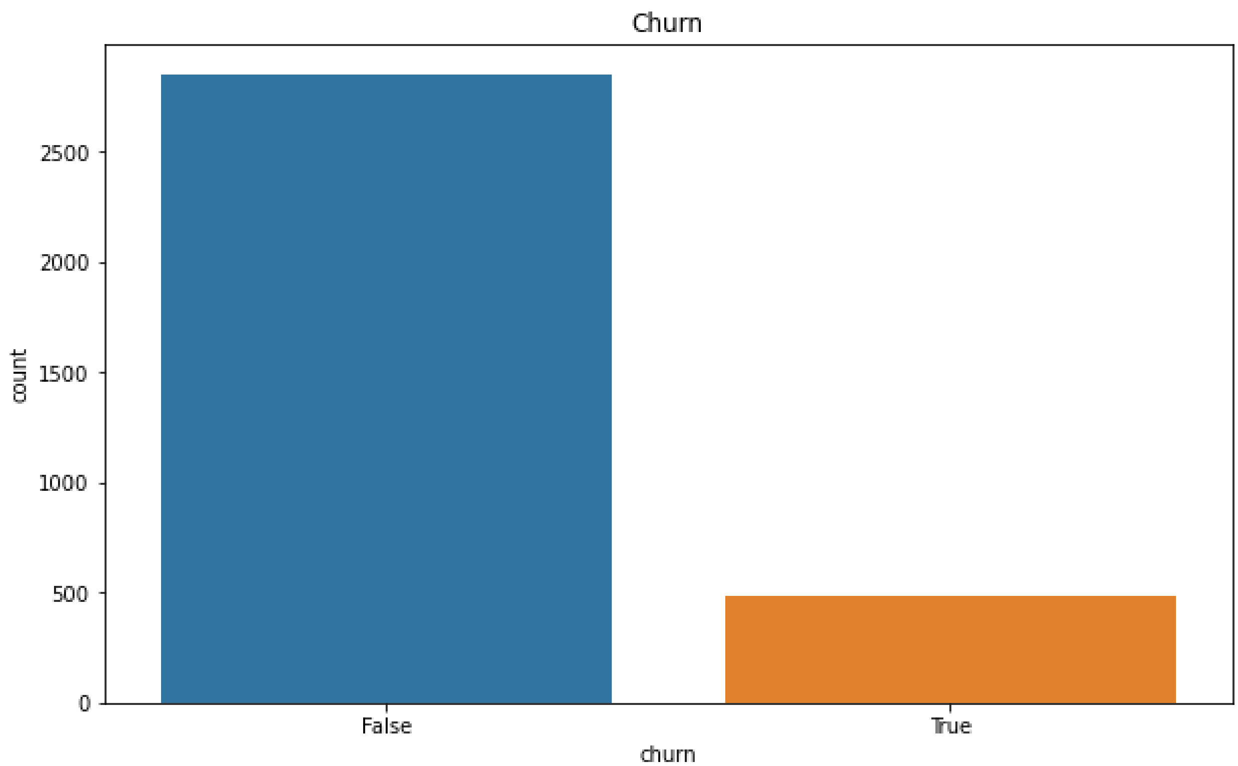
Univeriate Analysis

I will be doing a univariate analysis on churn to see the distribution of customers. Other variable are: state, area code, international plan, voice mail plan, customer service call, total day charge, total night charge, customer service call, and account legth. This will enable us to establish which areas and states of high customer churn and what the key factor contributors to this, is it the charges that are high, is it the global access plan in place, minimum required customer service call and how long do the businness enjoy customer loyalty before they end their relationship.

```
df['churn'].value_counts()
```

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

```
#Creating a count plot for churn to see the distribution
plt.figure(figsize=(10,6))
sns.countplot( data=df, x='churn')
plt.title('Churn');
```



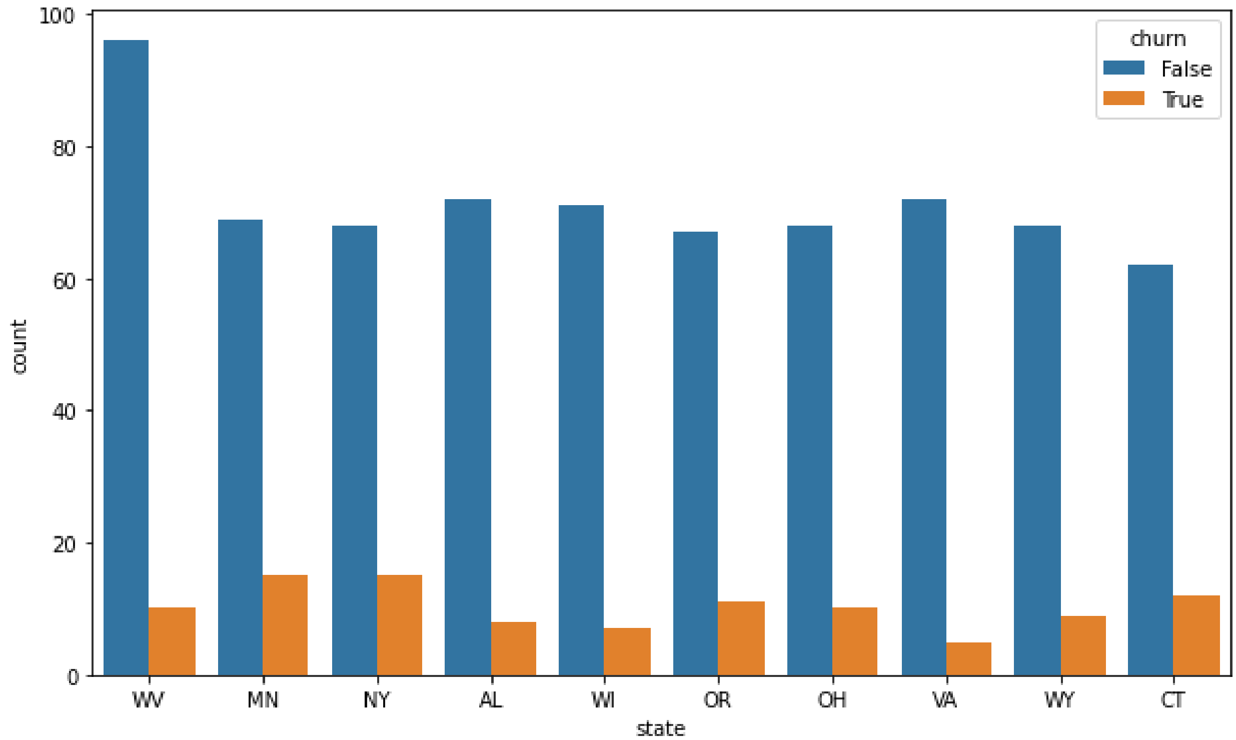

```
df['area code'].value_counts()
```

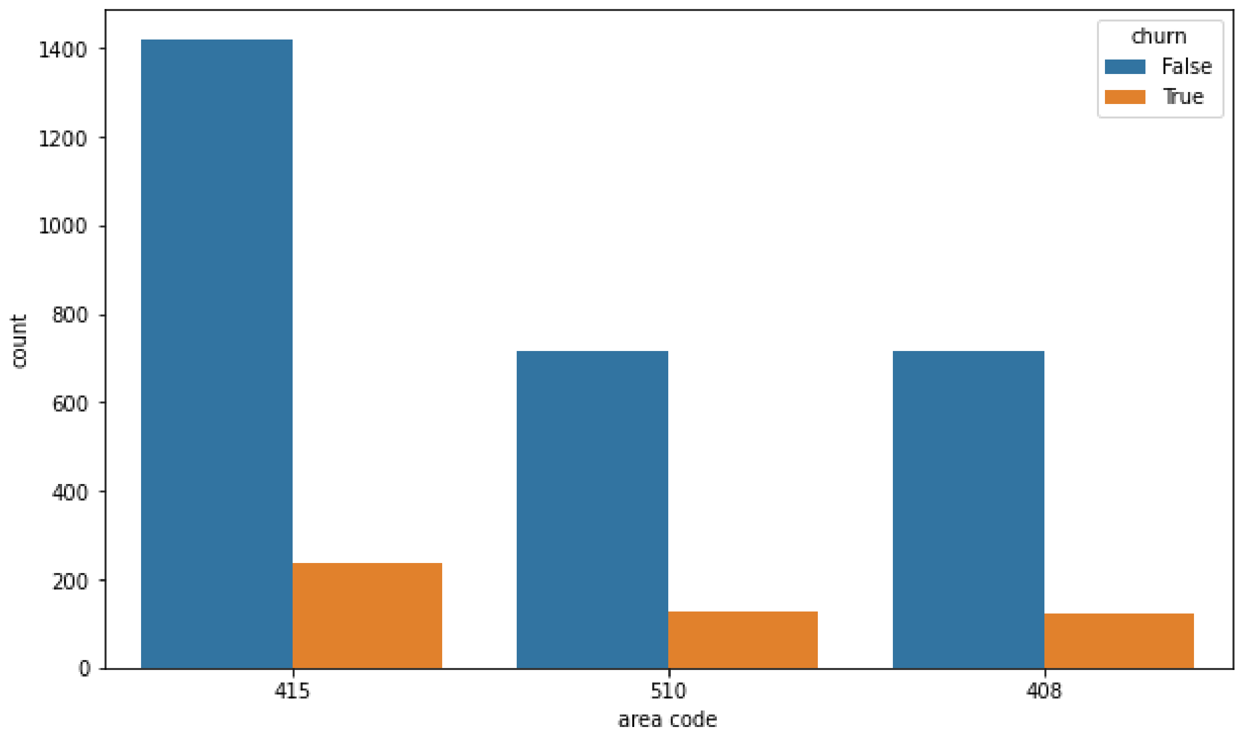
```
415    1655
510     840
408     838
Name: area code, dtype: int64
```

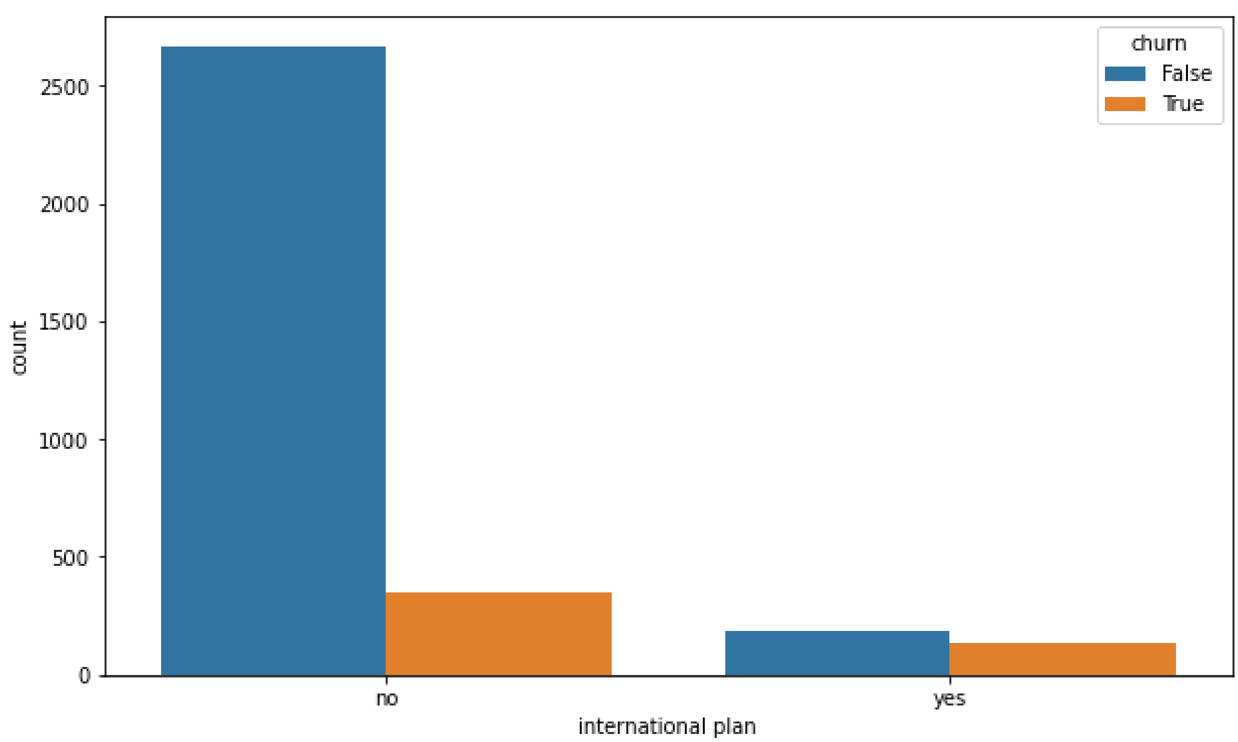
```
churn_dist_percentage = df.groupby('area code')['churn'].mean()*100
churn_dist_percentage
```

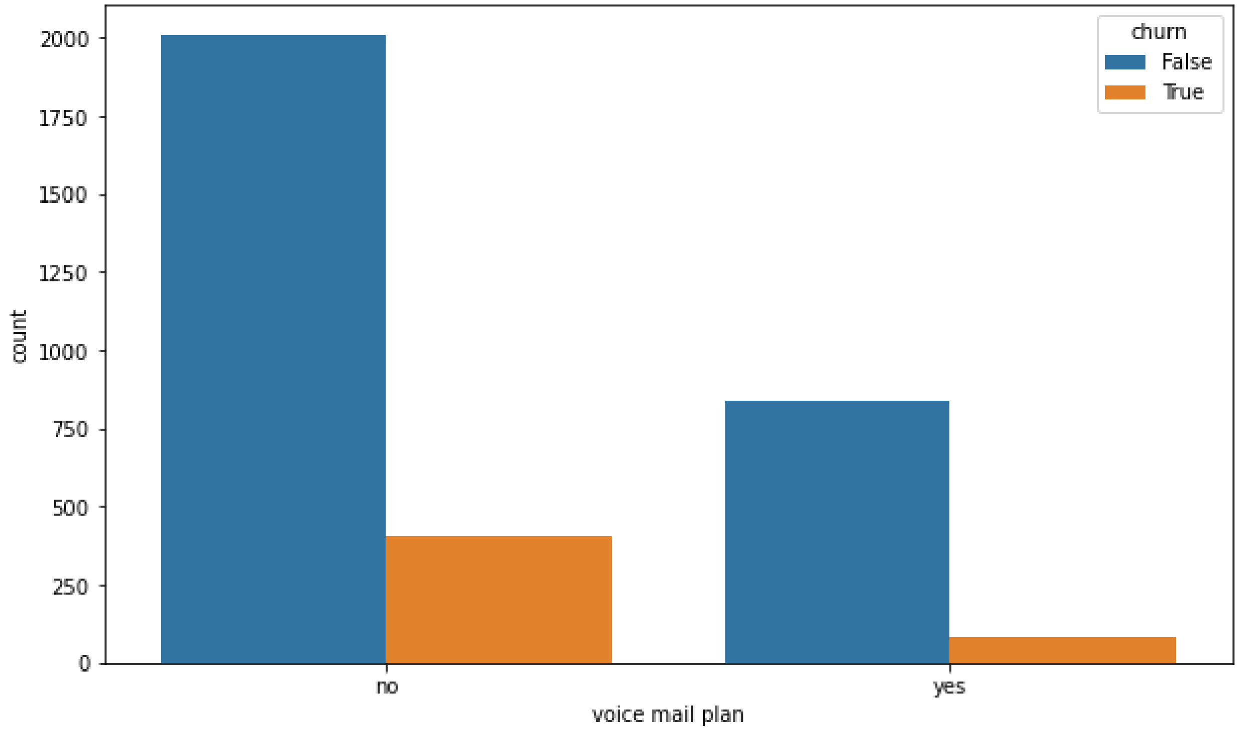
```
area code
408    14.558473
415    14.259819
510    14.880952
Name: churn, dtype: float64
```

```
#Categorical and charge columns churn distribution
categorical_cols = ['state','area code','international plan','voice
mail plan','customer service calls', 'total day charge', 'total night
charge','account length']
for column in categorical_cols:
    plt.figure(figsize=(10,6))
    sns.countplot(data=df, x= column, hue ='churn',
order=df[column].value_counts().iloc[:10].index)
    plt.show();
```

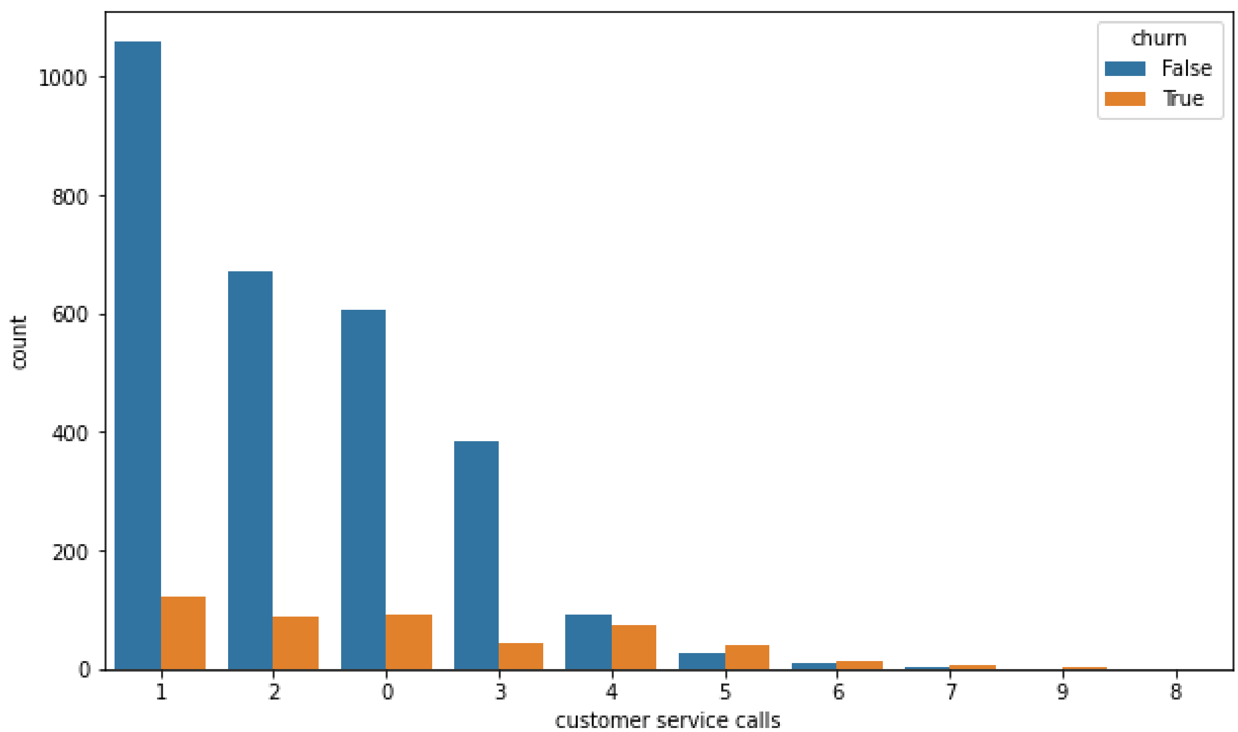



churn
False
True
1400
1200
1000
800
600
400
200
0
count
415
510
408
area code

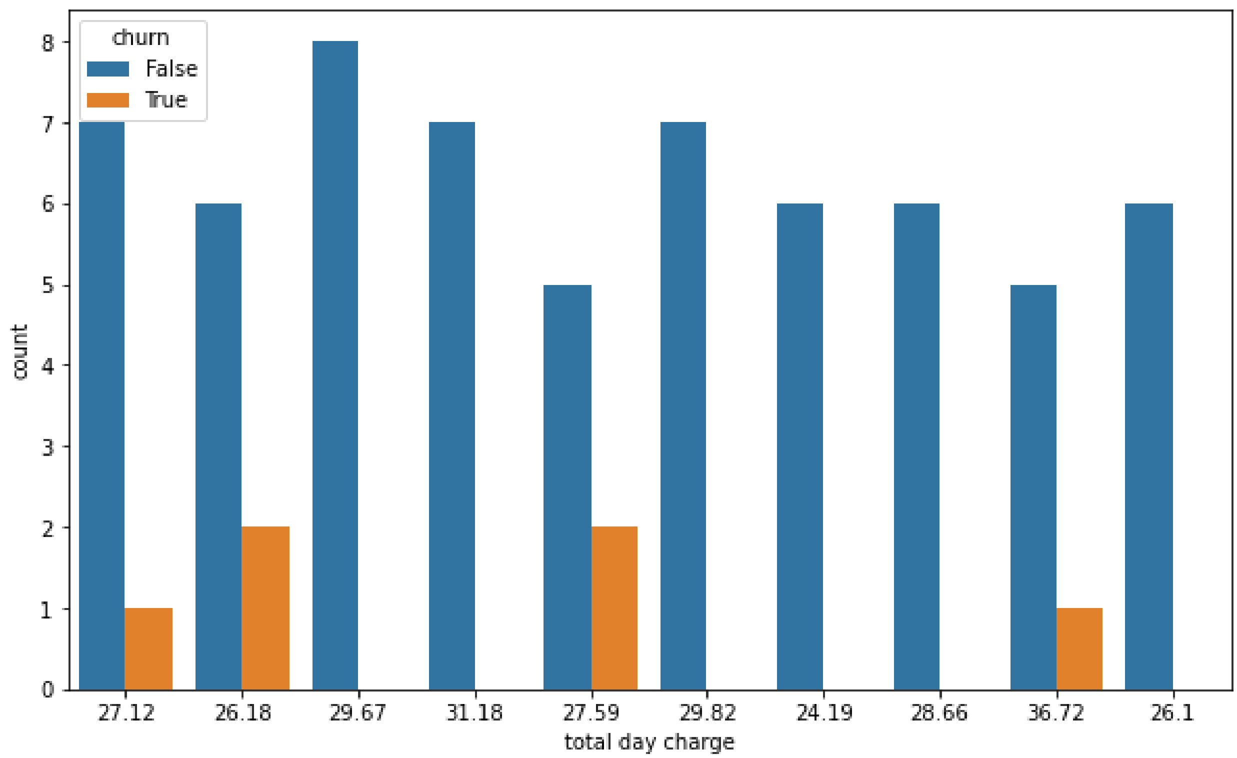

churn
False
True
2500
2000
1500
1000
500
0
count
no
yes
international plan

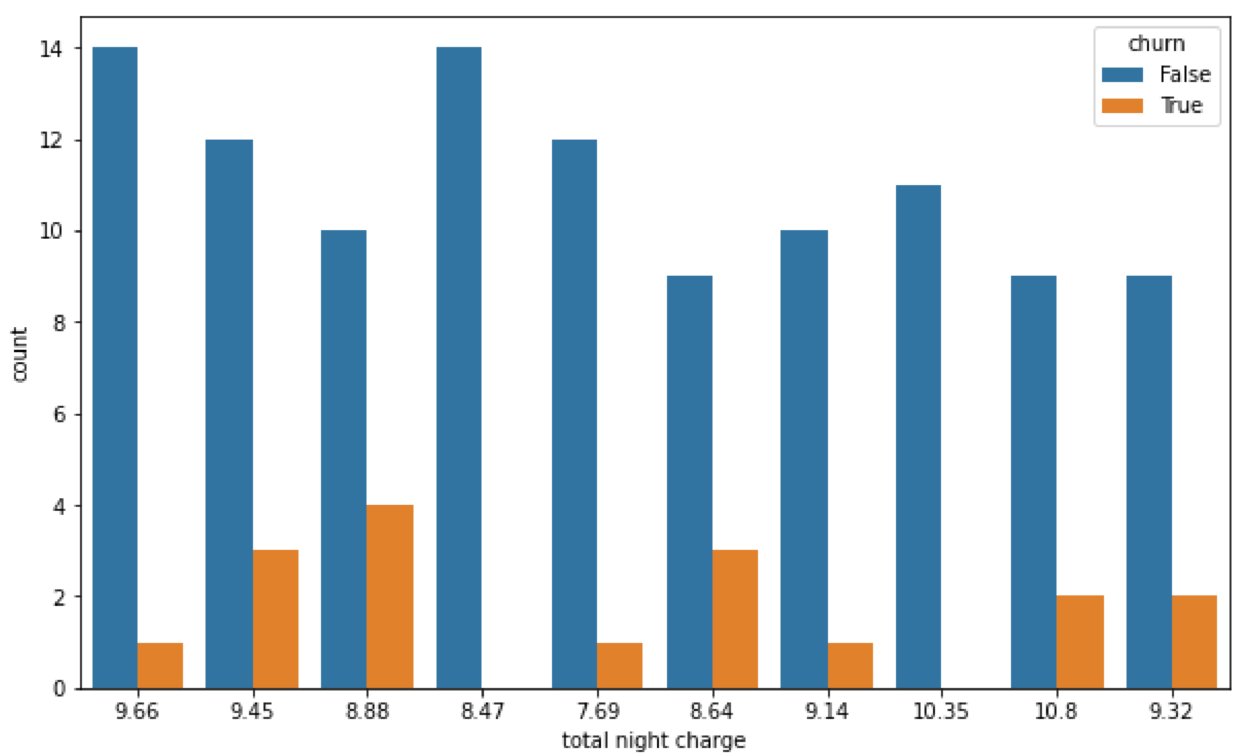


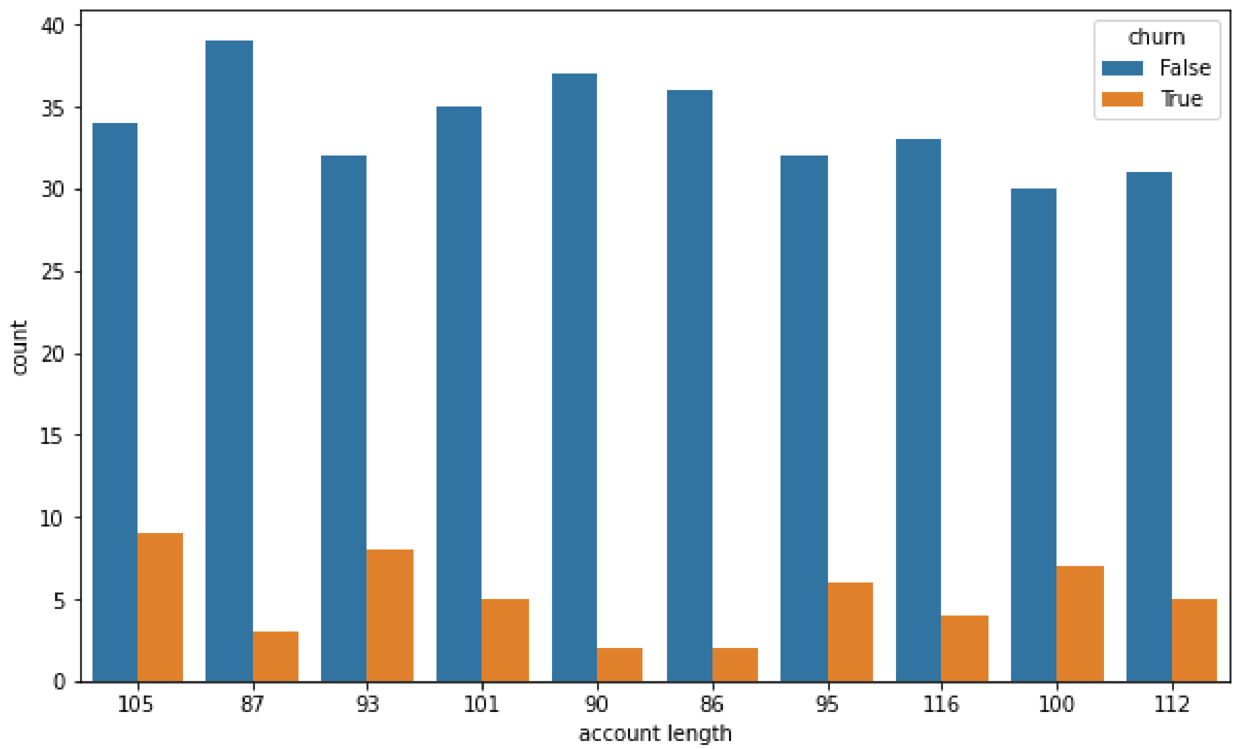
count
voice mail plan
no
yes
churn
False
True
0
250
500
750
1000
1250
1500
1750
2000



count
customer service calls
1
2
0
3
4
5
6
7
9
8
churn
False
True
0
200
400
600
800
1000

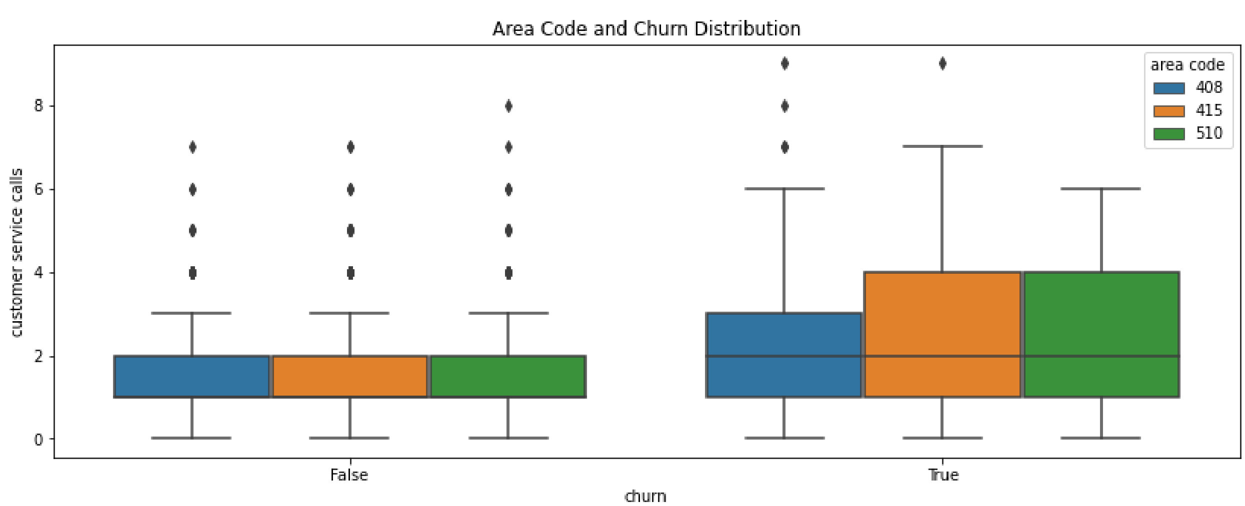

churn
False
True
count
27.12
26.18
29.67
31.18
27.59
29.82
24.19
28.66
36.72
26.1
total day charge


churn
False
True
count
9.66
9.45
8.88
8.47
7.69
8.64
9.14
10.35
10.8
9.32
total night charge



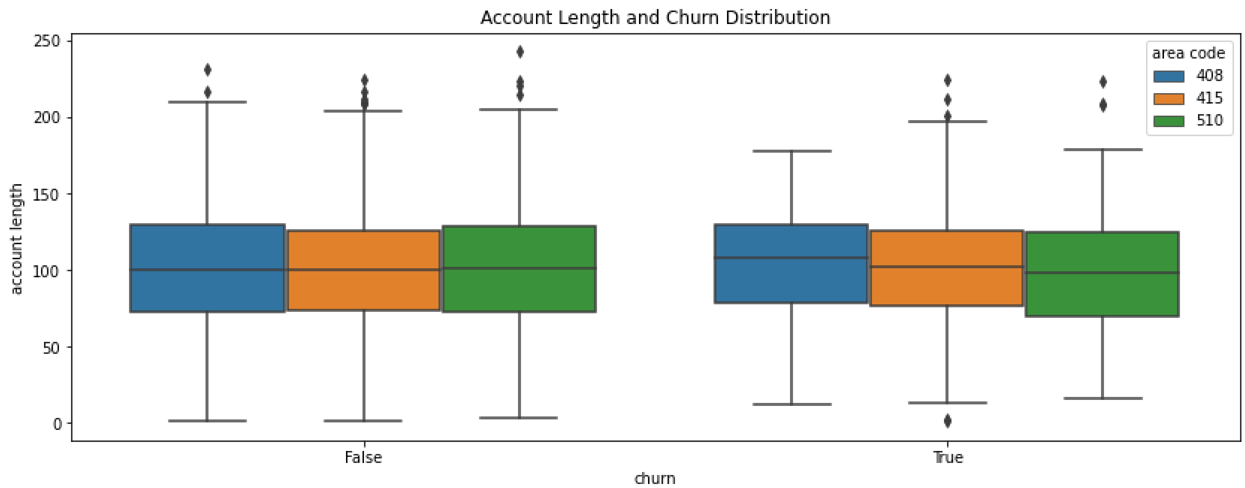
Bivariate Analysis

```
palette = sns.color_palette('tab10',3)
plt.figure(figsize=(14,5))
sns.boxplot(data=df, x='churn', y='customer service calls', hue= 'area
code', palette=palette)
plt.title('Area Code and Churn Distribution');
```

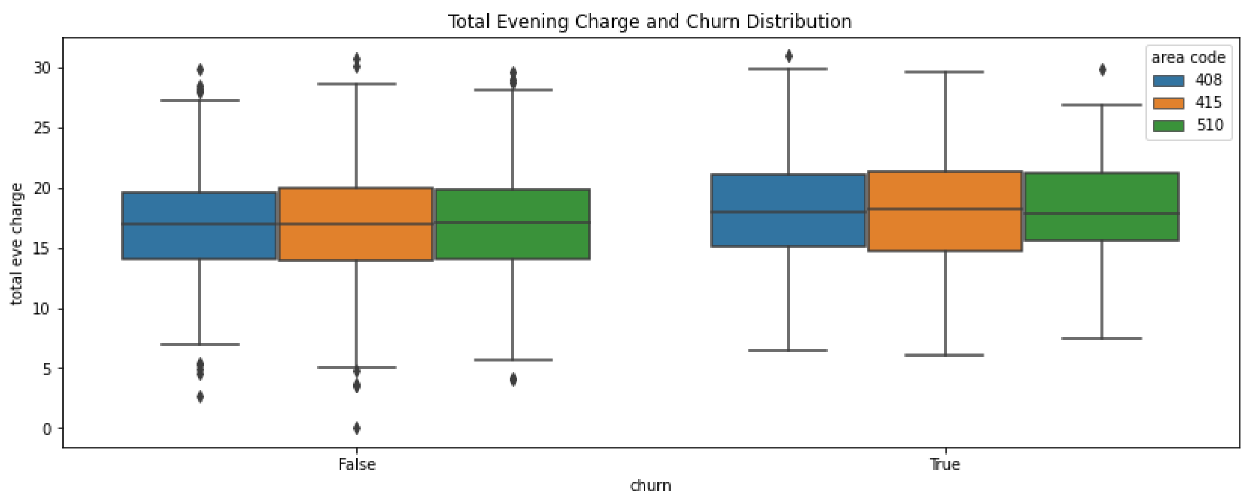



```
palette = sns.color_palette('tab10',3)
plt.figure(figsize=(14,5))
sns.boxplot(data=df, x='churn', y='account length', hue= 'area code',
```

```
palette=palette)
plt.title('Account Length and Churn Distribution');
```



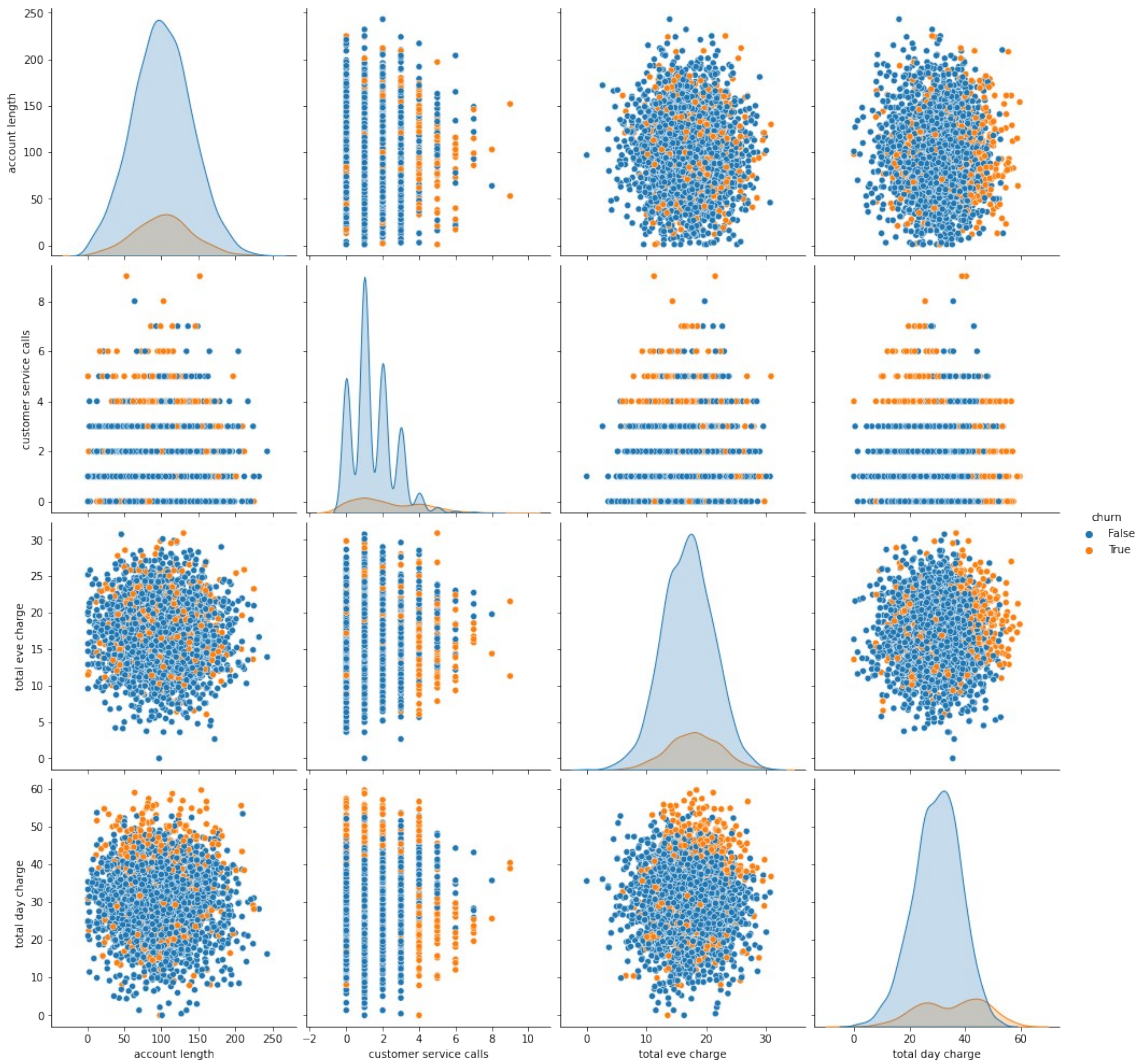

```
palette = sns.color_palette('tab10',3)
plt.figure(figsize=(14,5))
sns.boxplot(data=df, x='churn', y='total eve charge', hue= 'area
code', palette=palette)
plt.title('Total Evening Charge and Churn Distribution');
```



The bivariate analysis shows the relationship between churn and other features in the different area codes. This helps to investigate areas that we highly affected and why.The ratio of customer retention to total customer per area code is almost the same, however we note their are outliers in the retained customer meaning that customers could have left due to lack of equality in service provision by the business. The most affected area is code 510, it has most customer service call outliers and high churn rate regardless.

Multivariate Analysis

```
#Creating a pair plot to show correlation in respect to chur
selected_cols = df[['account length','customer service calls','total eve charge','total day charge','churn']]
sns.pairplot(selected_cols, hue='churn', height=3.5)
plt.show()
```



From the above analysis, most customer churn is from the forth call onwards. We also note that the number of calls is inconstent and has outliers, however, we will keep the data because it is relevant for our study.

```
# Dropping the area code column since the churn distribution ranges from 14.25 to 14.88 which may not be very insightful for our data.
df.drop(['area code'], axis=1, inplace = True)
```

```
df.columns
```

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

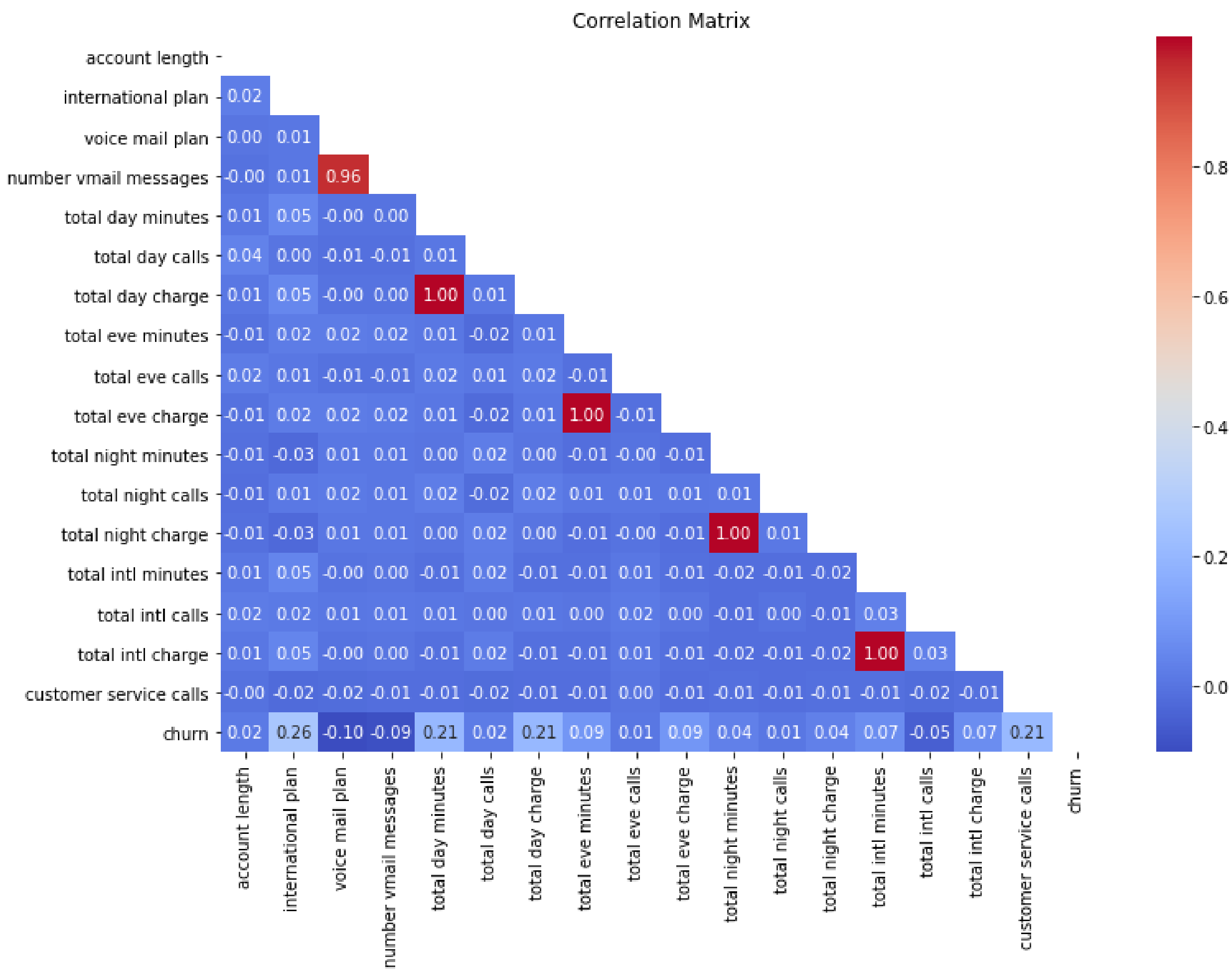
Data preprocessing

```
#Coverting the caterical variables to binary.
#Converting target churn, intenational plan and voice mail plan
variable to binary using label encoding or binary mapping
df['churn'] = df['churn'].map({False:0,True:1})
df['international plan'] = df['international plan'].map({'yes':1,
'no':0})
df['voice mail plan'] = df['voice mail plan'].map({'yes':1, 'no':0})
```

```
df.head()
```

```
  state  account length  international plan  voice mail plan  \
0    KS             128                   0                1
1    OH             107                   0                1
2    NJ             137                   0                0
3    OH              84                   1                0
4    OK              75                   1                0

   number vmail messages  total day minutes  total day calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   total day charge  total eve minutes  total eve calls  total eve
charge  \
0             45.07              197.4               99
16.78
1             27.47              195.5              103
16.62
2             41.38              121.2              110
10.30
3             50.90               61.9               88
5.26
4             28.34              148.3              122
12.61
```

```
   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1      0
1                       1      0
2                       0      0
3                       2      0
4                       3      0

#Creating a correlation matrix to show the relationship of different
variables

#Exluding non_numerical for the calculation
numeric_df = df.select_dtypes(include='number')

#Calculation correlation matrix
corr_matrix = numeric_df.corr()

#Creating a mask for lower triangle display

mask = np.triu(np.ones_like(corr_matrix, dtype = bool))

#Plotting

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='coolwarm',
fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



There is a strong corelation between total minutes and total charges which is expected because one depends on the other and the relationship is linear. Our target churn, has the highest correlation of 0.26 with international plan variable followed by total day minutes, total day charge, and customer service calls which is at 0.21.

Modelling

```
#Assing the y and X variables for splitting and training
y = df['churn']
X = df.drop(columns=['churn','state'], axis = 1 )

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 30)

X_train.head()
```

```
      account length  international plan  voice mail plan  \
2154             126                   1                0
2839             112                   0                0
1564             137                   0                0
1015             122                   0                0
```

```
874              103                     0                0

      number vmail messages  total day minutes  total day calls  \
2154                      0              197.6              126
2839                      0              266.0               97
1564                      0               97.5               95
1015                      0              232.5               96
874                       0              204.9              107

      total day charge  total eve minutes  total eve calls  total eve
charge  \
2154             33.59              246.5              112
20.95
2839             45.22              214.6               94
18.24
1564             16.58              195.8               82
16.64
1015             39.53              205.5              120
17.47
874              34.83              135.2              102
11.49

      total night minutes  total night calls  total night charge  \
2154                285.3                104               12.84
2839                306.2                100               13.78
1564                288.8                 78               13.00
1015                213.7                 91                9.62
874                 208.2                106                9.37

      total intl minutes  total intl calls  total intl charge  \
2154                12.5                 8               3.38
2839                14.2                 2               3.83
1564                 0.0                 0               0.00
1015                11.9                 2               3.21
874                 10.4                 3               2.81

      customer service calls
2154                       2
2839                       2
1564                       1
1015                       0
874                        5

# Scaling data.Choosing StandardScaler because most variable in our
data assume a Gaussian Distribution
#Initializing the scaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled =  scaler.transform(X_test)
```

```python
#Converting scales array to DataFrame

X_train_scaled_df = pd.DataFrame(X_train_scaled, columns =
X_train.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns =
X_test.columns)

#Initializing SMOTE
smote = SMOTE(random_state = 30)
X_train_scaled_df, y_train = smote.fit_resample(X_train_scaled_df,
y_train)
```

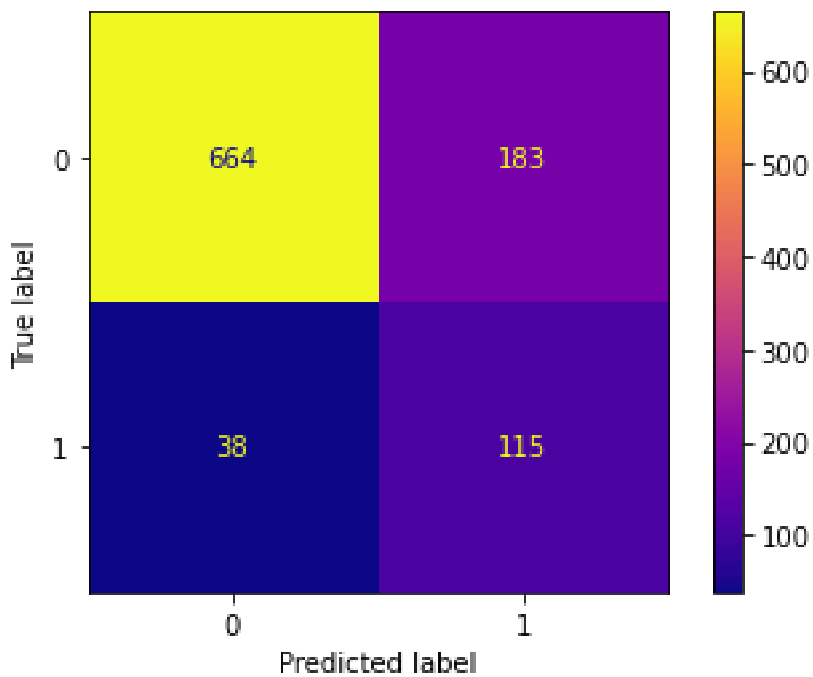
Model 1: Logistic Regression

```python
loreg = LogisticRegression()
loreg.fit(X_train_scaled_df, y_train)

y_pred_train = loreg.predict(X_train_scaled_df)
y_pred_test = loreg.predict(X_test_scaled_df)

#Calculating Recall Scores

train_recall = recall_score(y_train, y_pred_train)
test_recall = recall_score(y_test, y_pred_test)

print(f'Training Recall:',train_recall)
print(f'Test Recall:',test_recall)
```

```
Training Recall: 0.7713429855217174
Test Recall: 0.7516339869281046
```

From the above recall metric on the logistic model we can see that the model correctly identifies approximately 77.13% of the positive instances in the training data and 75.16% on the test data.This indicates that the model generalizes well from the training data to the test data.

```python
#Confusion matrix metric on the Logistic model
conf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels=loreg.classes_)
disp.plot(cmap='plasma')
plt.show()
```




```python
# Defining a metric function
def mod_metrics(labels, preds):
    print('Recall Score:{}'.format(recall_score(labels, preds)))
    print('Precision Score:{}'.format(precision_score(labels, preds)))
    print('Accuracy Score:{}'.format(accuracy_score(labels, preds)))
    print('F1 Score:{}'.format(f1_score(labels, preds)))
    print('ROC AUC Score:{}'.format(roc_auc_score(labels, preds)))


mod_metrics(y_test, y_pred_test)
```

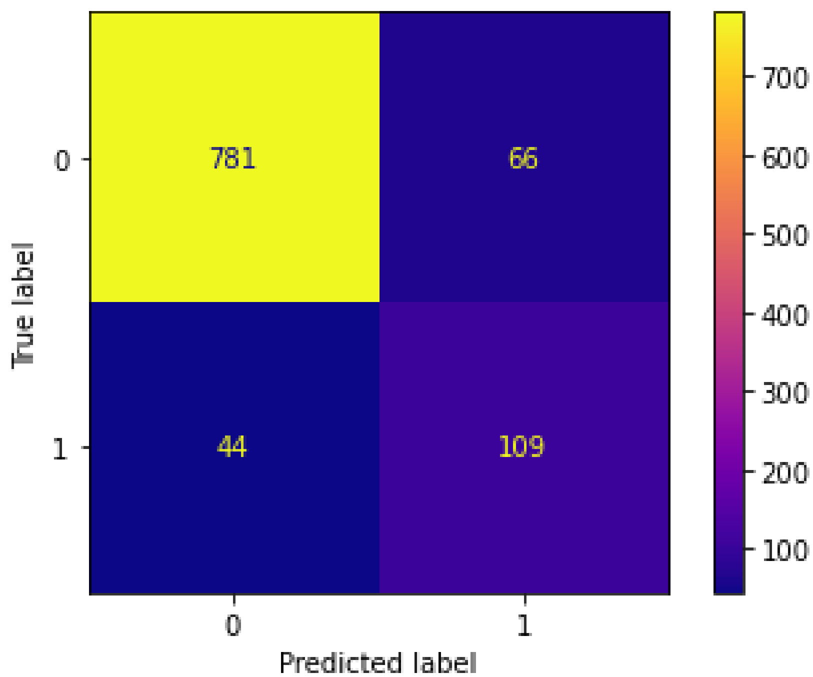
```
Recall Score:0.7516339869281046
Precision Score:0.3859060402684564
Accuracy Score:0.779
F1 Score:0.5099778270509978
ROC AUC Score:0.7677886581629897
```

The model correctly identified that: 1.Approximately 75.16% of the actual positive instances. 2.38.59% of the positive predictions made by the model were correct. 3.Model has an overall accuracy of 77.9%. 4.Model is performing at approximately 51.0% in reference to recall and precision. 5.Has approximately 76.78% ability to discriminate between negative and positive instances

```python
dtc =DecisionTreeClassifier()
dtc.fit(X_train_scaled_df, y_train)

#Making predictions
y_pred_train = dtc.predict(X_train_scaled_df)
```

```python
y_pred_test = dtc.predict(X_test_scaled_df)

#Model performance metrics

print('Model:Decision Tree Classifier')
print('Training Metrics:')
mod_metrics(y_train, y_pred_train)
print('\n---------------------------\n')

print('Model:Decision Tree Classifier')
print('Testing Metrics:')
mod_metrics(y_test, y_pred_test)

#Plotting confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels = dtc.classes_)
disp.plot(cmap='plasma')
plt.show()
```

```
Model:Decision Tree Classifier
Training Metrics:
Recall Score:1.0
Precision Score:1.0
Accuracy Score:1.0
F1 Score:1.0
ROC AUC Score:1.0

---------------------------

Model:Decision Tree Classifier
Testing Metrics:
Recall Score:0.7124183006535948
Precision Score:0.6228571428571429
Accuracy Score:0.89
F1 Score:0.6646341463414634
ROC AUC Score:0.8172481113657585
```

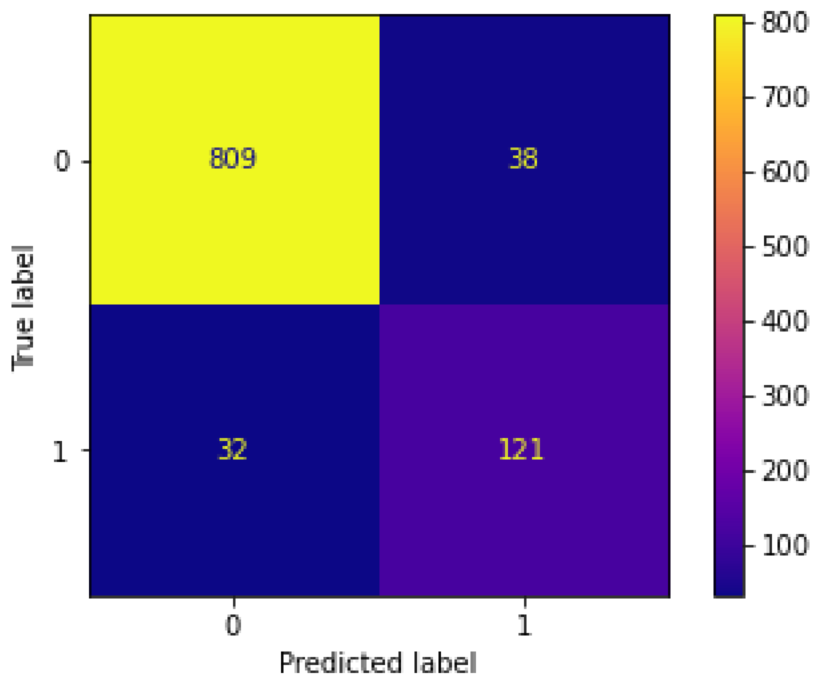


Decision Tree Model: It show perfect scores on the training all training metrics which could mean that the model had memorized all training data which indicates overfitting. Testing metrics are slightly lower which is a sign of overfitting.Generally, the model shows good performance and good scores.

```
gbc = GradientBoostingClassifier()
gbc.fit(X_train_scaled_df, y_train)

#Making predictions
y_pred_train = gbc.predict(X_train_scaled_df)
y_pred_test = gbc.predict(X_test_scaled_df)

#Model performance metrics

print('Model:Gradient Boosting Classifier')
print('Training Metrics:')
mod_metrics(y_train, y_pred_train)
print('\n---------------------------\n')

print('Model:Gradient Boosting Classifier')
print('Testing Metrics:')
mod_metrics(y_test, y_pred_test)

#Plotting confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels = gbc.classes_)
disp.plot(cmap='plasma')
```

```
plt.show()
```

```
Model:Gradient Boosting Classifier
Training Metrics:
Recall Score:0.9550673989016475
Precision Score:0.9780163599182005
Accuracy Score:0.9667998002995507
F1 Score:0.966405657994443
ROC AUC Score:0.9667998002995506

----------------------------

Model:Gradient Boosting Classifier
Testing Metrics:
Recall Score:0.7908496732026143
Precision Score:0.7610062893081762
Accuracy Score:0.93
F1 Score:0.7756410256410258
ROC AUC Score:0.8729927232601029
```

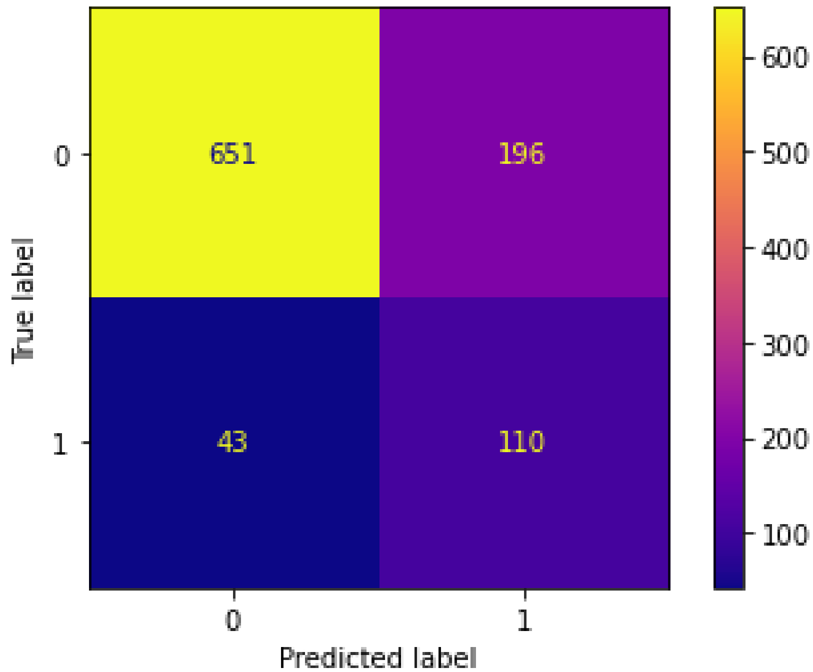


Gradient Boosting Classifier: It has high scores indicating good performance and generalization ability

```
gnb = GaussianNB()
gnb.fit(X_train_scaled_df, y_train)

#Making predictions
```

```
y_pred_train = gnb.predict(X_train_scaled_df)
y_pred_test = gnb.predict(X_test_scaled_df)

#Model performance metrics

print('Model:Gaussian NB')
print('Training Metrics:')
mod_metrics(y_train, y_pred_train)
print('\n---------------------------\n')

print('Model:Gaussian NB')
print('Testing Metrics:')
mod_metrics(y_test, y_pred_test)

#Plotting confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels = gnb.classes_)
disp.plot(cmap='plasma')
plt.show()
```

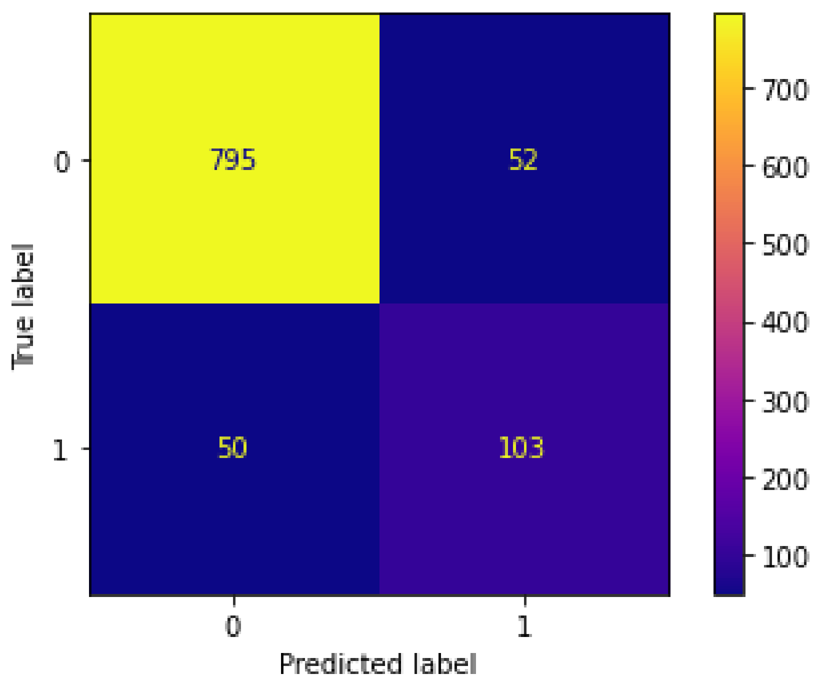
```
Model:Gaussian NB
Training Metrics:
Recall Score:0.8092860708936596
Precision Score:0.7782045127220355
Accuracy Score:0.7893160259610584
F1 Score:0.7934410181106216
ROC AUC Score:0.7893160259610584

---------------------------

Model:Gaussian NB
Testing Metrics:
Recall Score:0.7189542483660131
Precision Score:0.35947712418300654
Accuracy Score:0.761
F1 Score:0.4793028322440087
ROC AUC Score:0.7437746448441636
```



Gaussian NB: The model shows the lowest scores so far. Testing metrics are lower which shows the possibility of limitation to complexity of the data and overfitting

```
svc = SVC()
svc.fit(X_train_scaled_df, y_train)

#Making predictions
y_pred_train = svc.predict(X_train_scaled_df)
y_pred_test = svc.predict(X_test_scaled_df)

#Model performance metrics

print('Model:SVC')
print('Training Metrics:')
mod_metrics(y_train, y_pred_train)
print('\n---------------------------\n')

print('Model:SVC')
print('Testing Metrics:')
mod_metrics(y_test, y_pred_test)

#Plotting confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels = svc.classes_)
disp.plot(cmap='plasma')
plt.show()
```

```
Model:SVC
Training Metrics:
Recall Score:0.9181228157763355
Precision Score:0.953838174273859
Accuracy Score:0.9368447329006491
F1 Score:0.9356397863139151
ROC AUC Score:0.9368447329006491

---------------------------

Model:SVC
Testing Metrics:
Recall Score:0.673202614379085
Precision Score:0.6645161290322581
Accuracy Score:0.898
F1 Score:0.6688311688311689
ROC AUC Score:0.8059047310384209
```



Support Vector Classifier: The model shows good performance by the high scores even with the lower test metrics

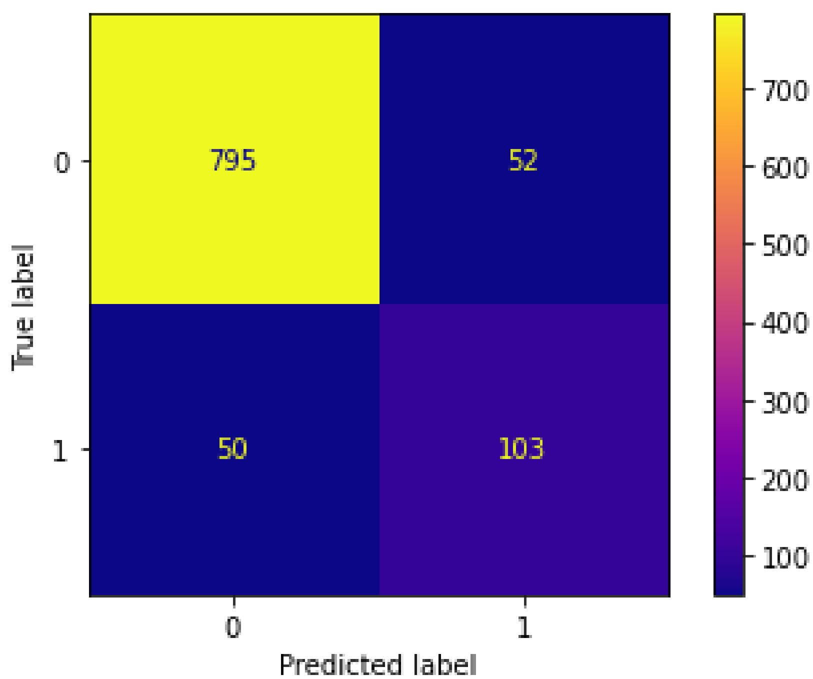
Generally, the Gradient Boosting classifier has the best performance and seems a better choice among all models because high metrics for both training and testing. It shows a good generalizing ability, robustness and reliability. The other model that shows good performance is the SVC.

Model Tuning

```
# Initializing the Classifier
gbc_classifier = GradientBoostingClassifier()

#Defininf parameter grid

param_grid = {'n_estimators': [50, 100, 200],
            'learning_rate':[0.01, 0.05, 0.1],
            'max_depth':[3, 4, 5]}

#Perfoming grid search with cross validation
grid_search = GridSearchCV(estimator=gbc_classifier,
param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train_scaled_df, y_train)

#Getting the best parameters

best_params = grid_search.best_params_

print('Best Hyperparameters:', best_params)
print('\n------------------------------\n')

#Using the best estimator to give predictions

best_gbc_classifier = grid_search.best_estimator_

y_pred_train_tuned = best_gbc_classifier.predict(X_train_scaled_df)
y_pred_test_tuned = best_gbc_classifier.predict(X_test_scaled_df)

#Printing Results

print('Training Metrics for Tuned model:')
mod_metrics(y_train, y_pred_train_tuned)
print('\n---------------------------\n')
print('Testing Metrics for Tuned model:')
mod_metrics(y_test, y_pred_test_tuned)

#Plotting a confusion Matrix

conf_matrix_tuned = confusion_matrix(y_test, y_pred_test)
disp_tuned = ConfusionMatrixDisplay(confusion_matrix =
conf_matrix_tuned, display_labels = best_gbc_classifier.classes_)
disp_tuned.plot(cmap='plasma')
plt.show()
```

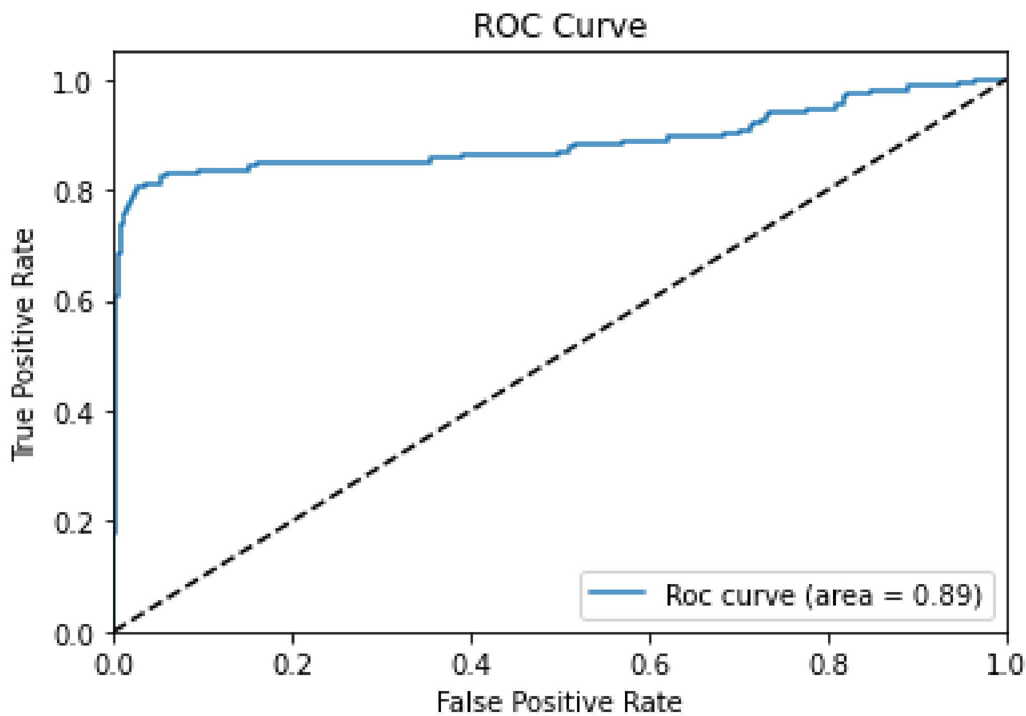
```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5,
'n_estimators': 200}

------------------------------
```

```
Training Metrics for Tuned model:
Recall Score:1.0
Precision Score:1.0
Accuracy Score:1.0
F1 Score:1.0
ROC AUC Score:1.0

----------------------------

Testing Metrics for Tuned model:
Recall Score:0.7973856209150327
Precision Score:0.8652482269503546
Accuracy Score:0.95
F1 Score:0.8299319727891157
ROC AUC Score:0.8874767537869143
```




```
#Plotting the ROC Curve

#Predicted probabilities from the positive class
y_pred_proba = best_gbc_classifier.predict_proba(X_test_scaled_df)
[:,1]

#Computing fpr,tpr, thresholds for the ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

#Calcualting Area Under the Curve
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

```
#Plotting the curve
plt.figure()
plt.plot(fpr, tpr, label = 'Roc curve (area =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



Model Performance: From the ROC Curve, we can conclusively say that the model has a strong distinguishing power between positive and negative classes. The model performs very well on the training data; it correctly identifies all instances and correctly predicts all instances.However, on the testing data, the model metrics are still strong but lower than those of the training data.

Model Limitation: The model is easily overfitting to training data even after tuning the model. This is due to lack of enough data to enable the data generalize well and more accurately

Recommendations

1. Collection of more data to reduce the overfitting by enabling the model perform better

2. Investigate and understand the ability of customer care staff on handling customer complains. This will enable the business to establish their contribution to customer churn and be able to mitigate the same through education and training of staff.
3. Include more data from competitors to help the business improve on internal process and marketing strategies.
4. Study on competitors charge rate, customer incentives, products and services offered to enable the business get a competitive edge.
5. Investigate on signals and ability to communicate effectively in areas where there is high customer churn