



Telecom Customer Churn Prediction

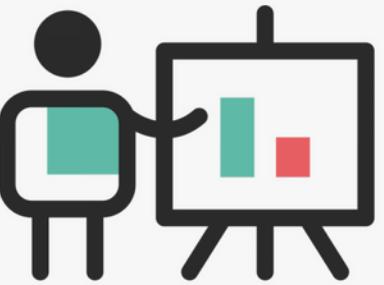
P - 191 - Group(3) - Working Team



- Ankita Kiran Sutar
- Lakshmi Priya
- Madasu Manjusha
- Muktagucha Manisha
- Nikita Anil Kesarkar
- Suraj Santosh Temkar
- Triveni Gunwant Chapekar
- Vaishnavi Tandle



INTRODUCTION



 Churn prediction is one of the most popular big data set that are used cases in business. It consists of detecting customers who are likely to leave the company. Churn is a problem for telecom companies because it is more expensive to acquire a new customer than to keep your existing one from leaving the company.

 Most of the telecom companies suffer from voluntary churn. Churn rate does have a strong impact on the life time value of the customer because it affects the length of service and also the future revenue for the company.

 In this situation the company tries to identify in advance about the customers who are going to churn. Then the company especially for those customers will provide with some special offers. But this approach can bring the company a huge loss if it is not predicted accurately.



DATASET DETAILS

The datafile 'Churn.csv' contains a total of 20 features for 5000 customers. Each row corresponds to a client of a telecommunications company for whom it has collected information about the type of plan they have contracted, the minutes they have talked, or the charge they pay every month.

The data set includes the following variables:

- state: Categorical, for the 51 states and the District of Columbia.
- Area.code
- account.length: how long the account has been active.
- voice.plan: yes or no, voicemail plan.
- voice.messages: number of voicemail messages.
- intl.plan: yes or no, international plan.
- intl.mins: minutes customer used service to make international calls.
- intl.calls: total number of international calls.
- intl.charge: total international charge.
- day.mins: minutes customer used service during the day.
- day.calls: total number of calls during the day.
- day.charge: total charge during the day.
- eve.mins: minutes customer used service during the evening.
- eve.calls: total number of calls during the evening.
- eve.charge: total charge during the evening.
- night.mins: minutes customer used service during the night.
- night.calls: total number of calls during the night.
- night.charge: total charge during the night.
- customer.calls: number of calls to customer service.
- churn: Categorical, yes or no. Indicator of whether the customer has left the company (yes or no).



Business Objective



01

Customer churn is a big problem for telecommunications companies. Indeed, their annual churn rates are usually higher than 10%. For that reason, they develop strategies to keep as many clients as possible.

02

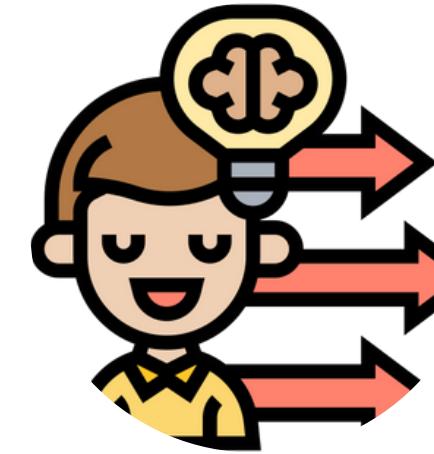
This is a classification project since the variable to be predicted is binary (churn or loyal customer). The goal here is to model churn probability, conditioned on the customer features.



Project Architecture



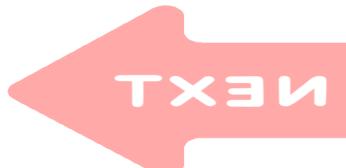
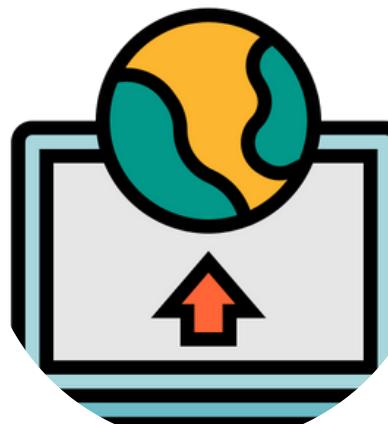
Collection of data



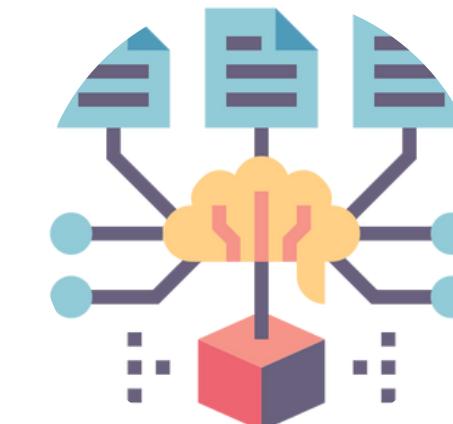
**Business
understanding**



EDA



Deployment



Model Building

EDA

EXPLORATORY DATA ANALYSIS



- Data collection
- Finding all variables & understanding them
- Cleaning the dataset
- Identify correlated variables
- Choosing the right statistical methods
- Visualizing & analyzing results

Processing Data



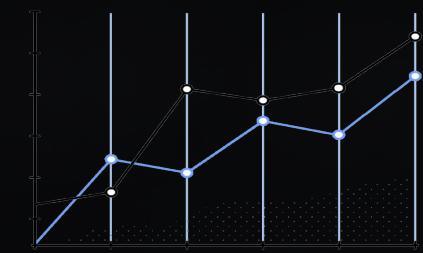
```
[26] df.head()
```

	Unnamed: 0	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	...
0	1	KS	area_code_415	128	yes	25	no	10.0	
1	2	OH	area_code_415	107	yes	26	no	13.7	
2	3	NJ	area_code_415	137	no	0	no	12.2	
3	4	OH	area_code_408	84	no	0	yes	6.6	
4	5	OK	area_code_415	75	no	0	yes	10.1	

5 rows × 21 columns

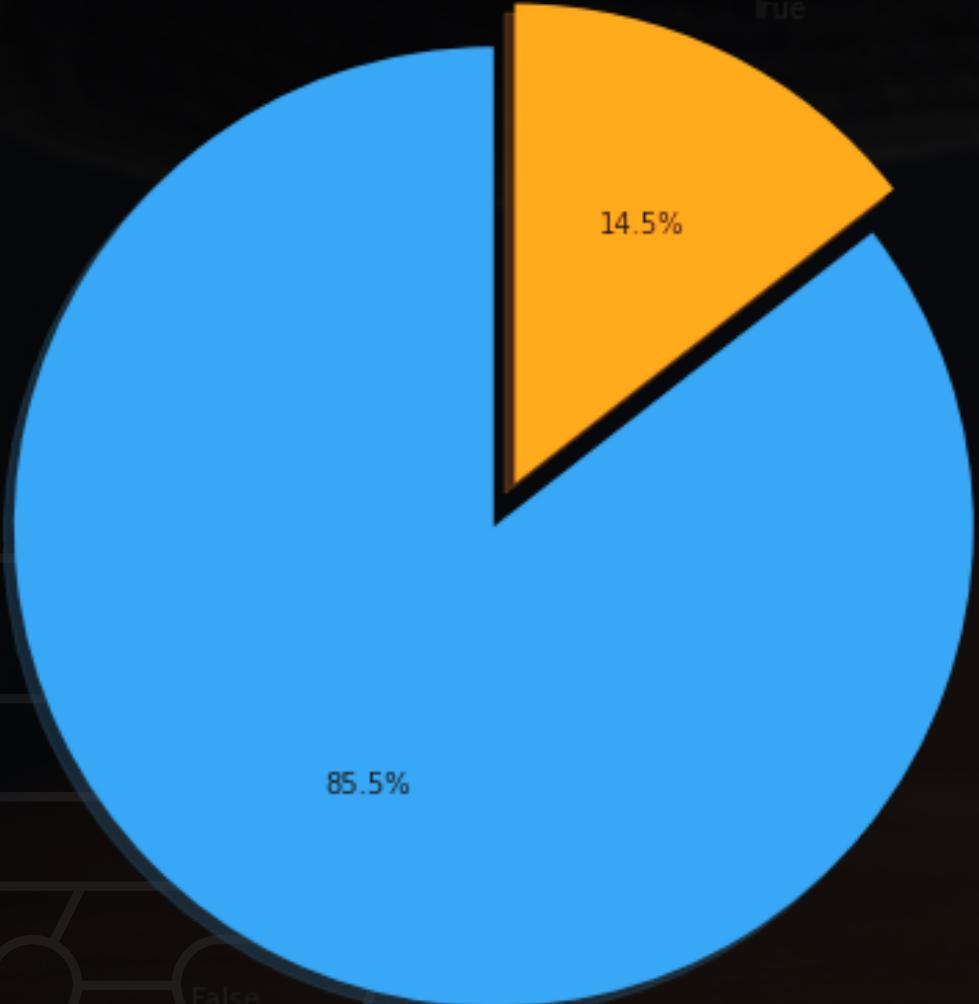
- No duplicate values (0)
- Found missing values (31)-removed the values
- Found wrong datatypes('eve.min','day.charge')-changed to float
- Number of numerical variables (15)
- Discrete Variables Count (2)
- Continuous Feature Count (13)
- categorical variables (5)
- Data is correlated

Visualizations



Analyzing Target Variable (Churn)

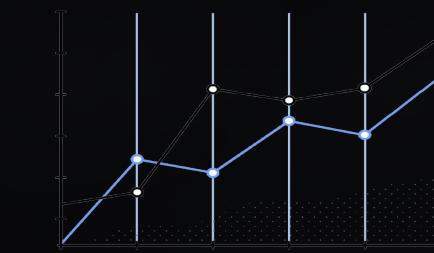
Pie Chart for Churn



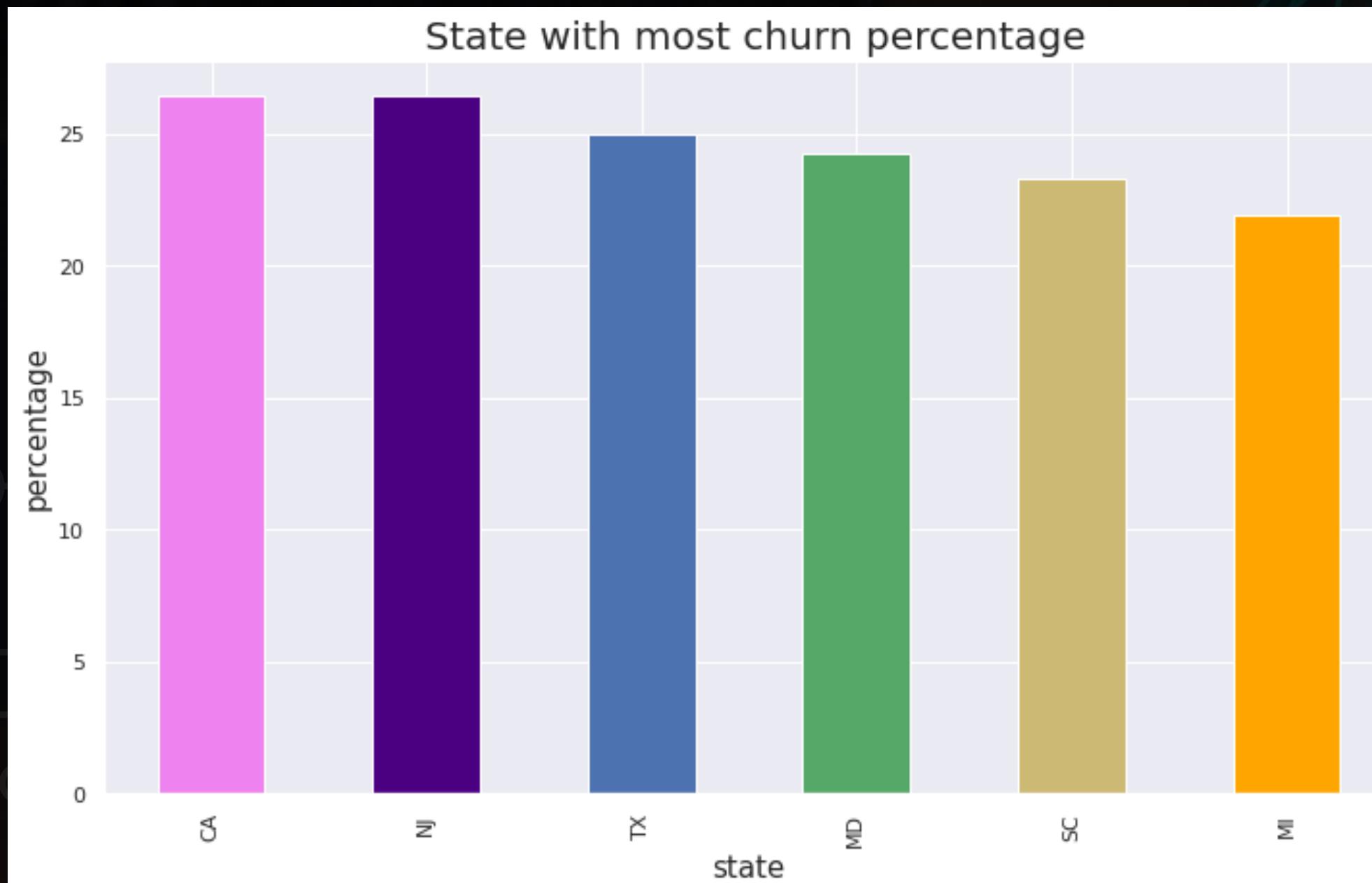
Report :

After analyzing the results we can see only 15% customers are churned. Let's check the other features and relation between them correlated with churn.

Visualizations



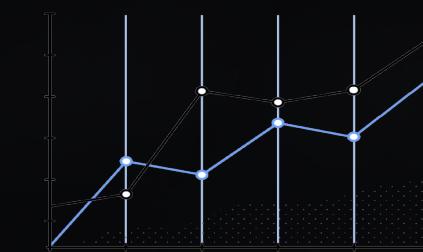
Analyzing State vs Churn



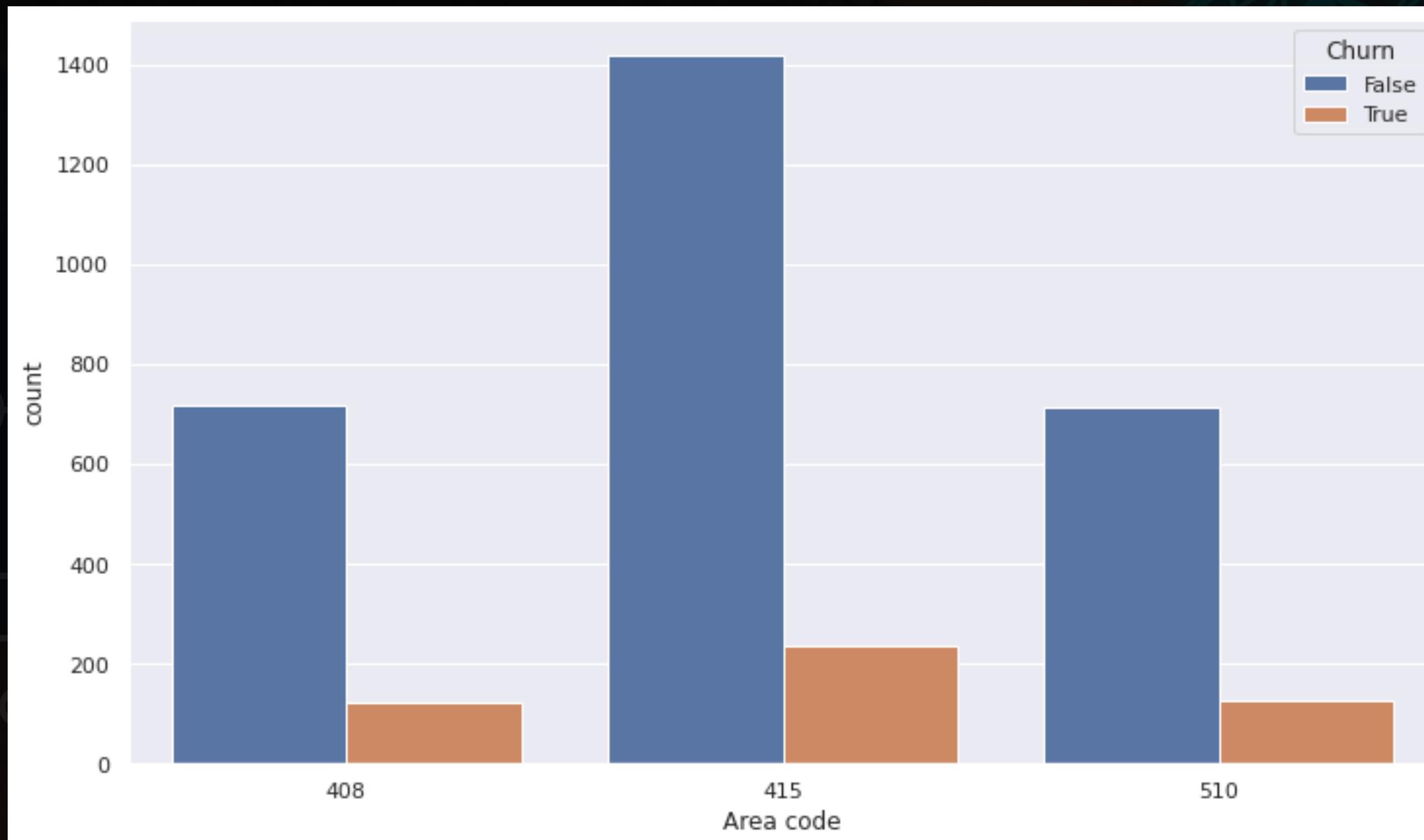
Report :

There are 51 unique states present that have different churn rate.
From the analysis CA, NJ, TX, MD, SC, MI are the ones who have a higher churn rate of more than 21

Visualizations



Analyzing Area.code vs Churn

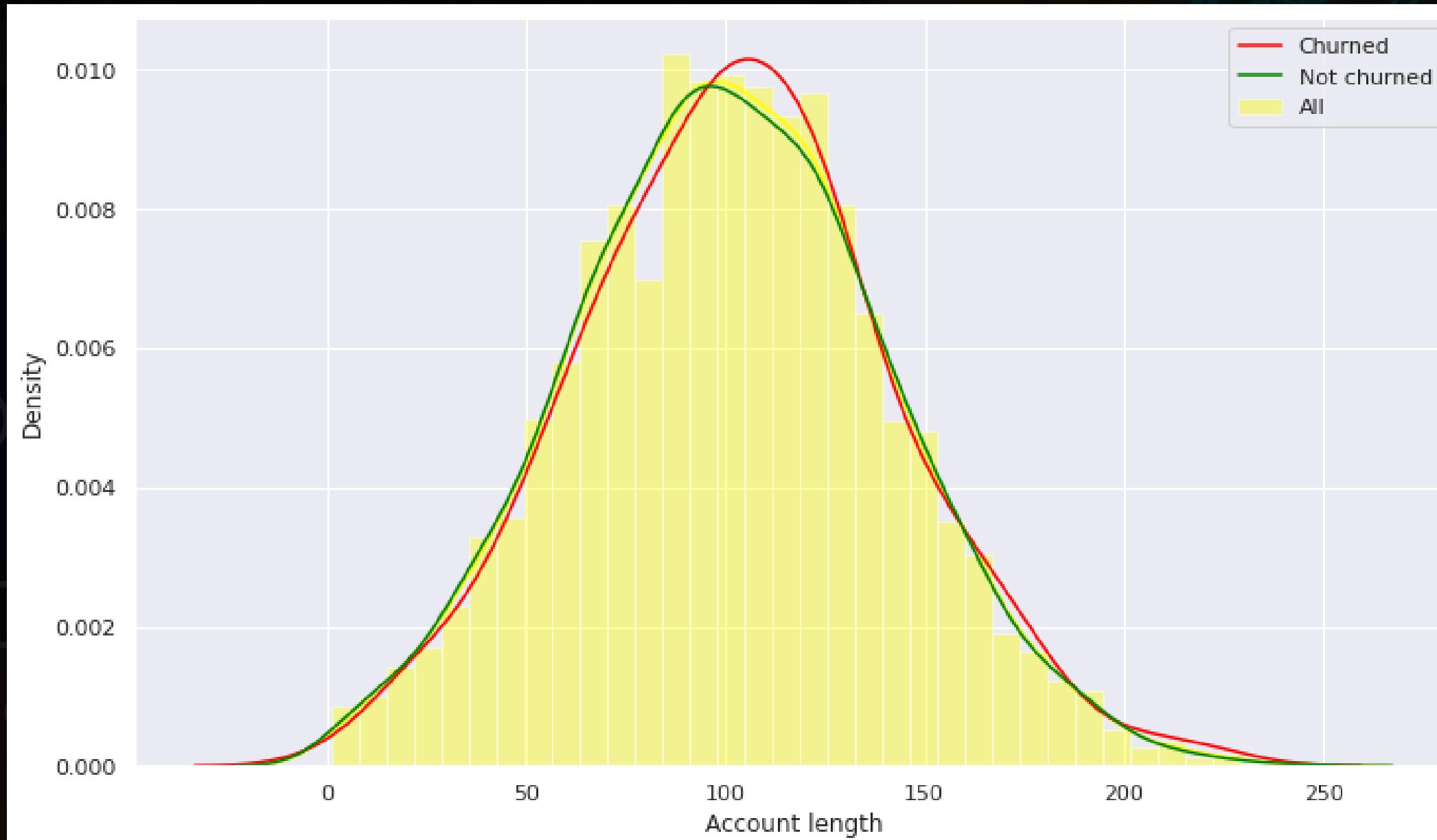


Report :

There are 3 unique values in area.code. 415 has most churn rate compared to other two. The churn rate has no much difference almost same. There is no relationship between area.code and churn due to which the customer leaves the operator.

Visualizations

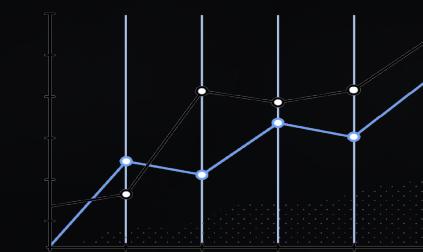
Analyzing Acc.len vs Churn



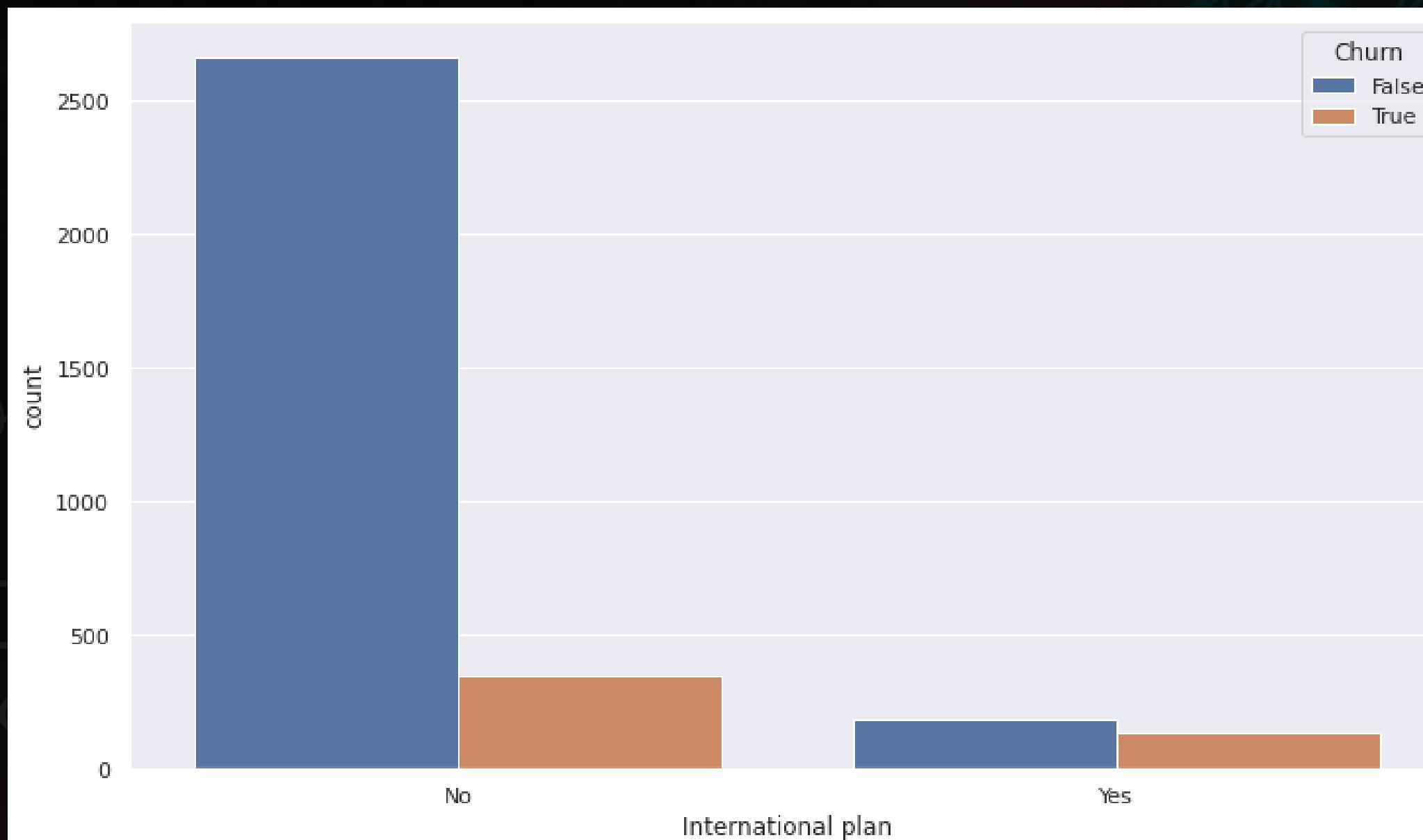
Report :

In this feature also we didn't find any relation between acc.len and churn as there is no much difference between them.

Visualizations



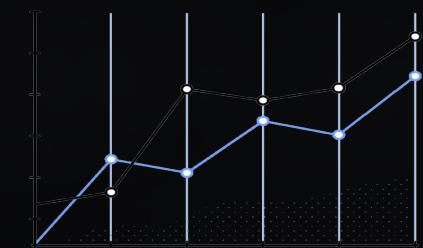
Analyzing Intl.plan vs Churn



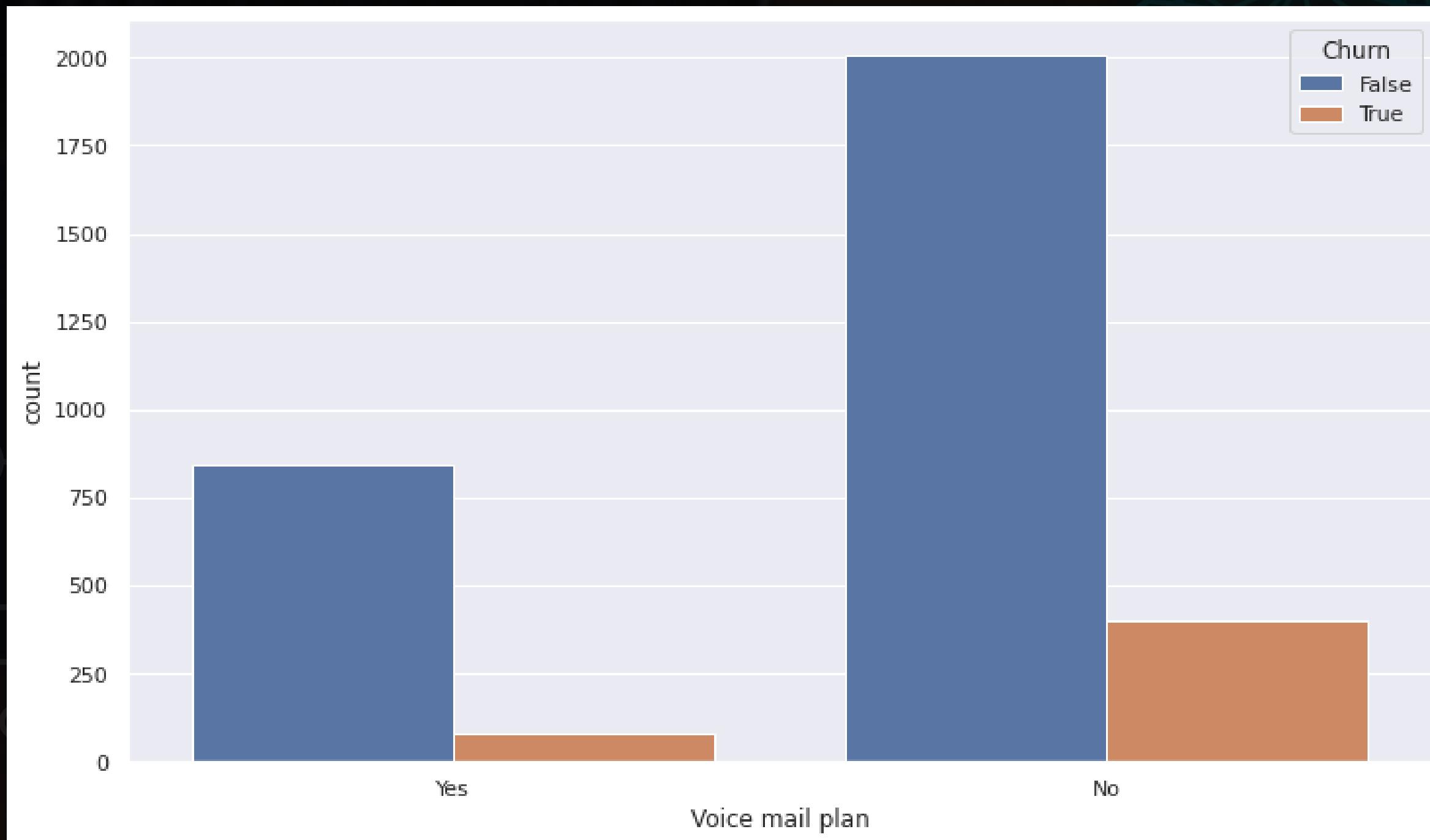
Report :

There are 3010 customers who don't have an international plan.
There are 323 customers who have an international plan.
Among those who have an international plan 42.4 % people churn.
Whereas among those who don't have a international plan only 11.4 % people churn.
So basically the people who bought International plans are churning in big numbers.

Visualizations



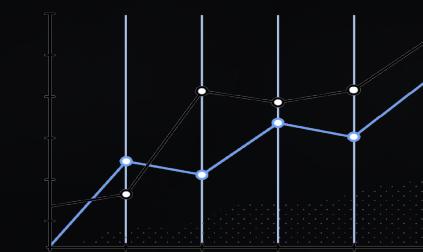
Analyzing Voice.plan vs Churn



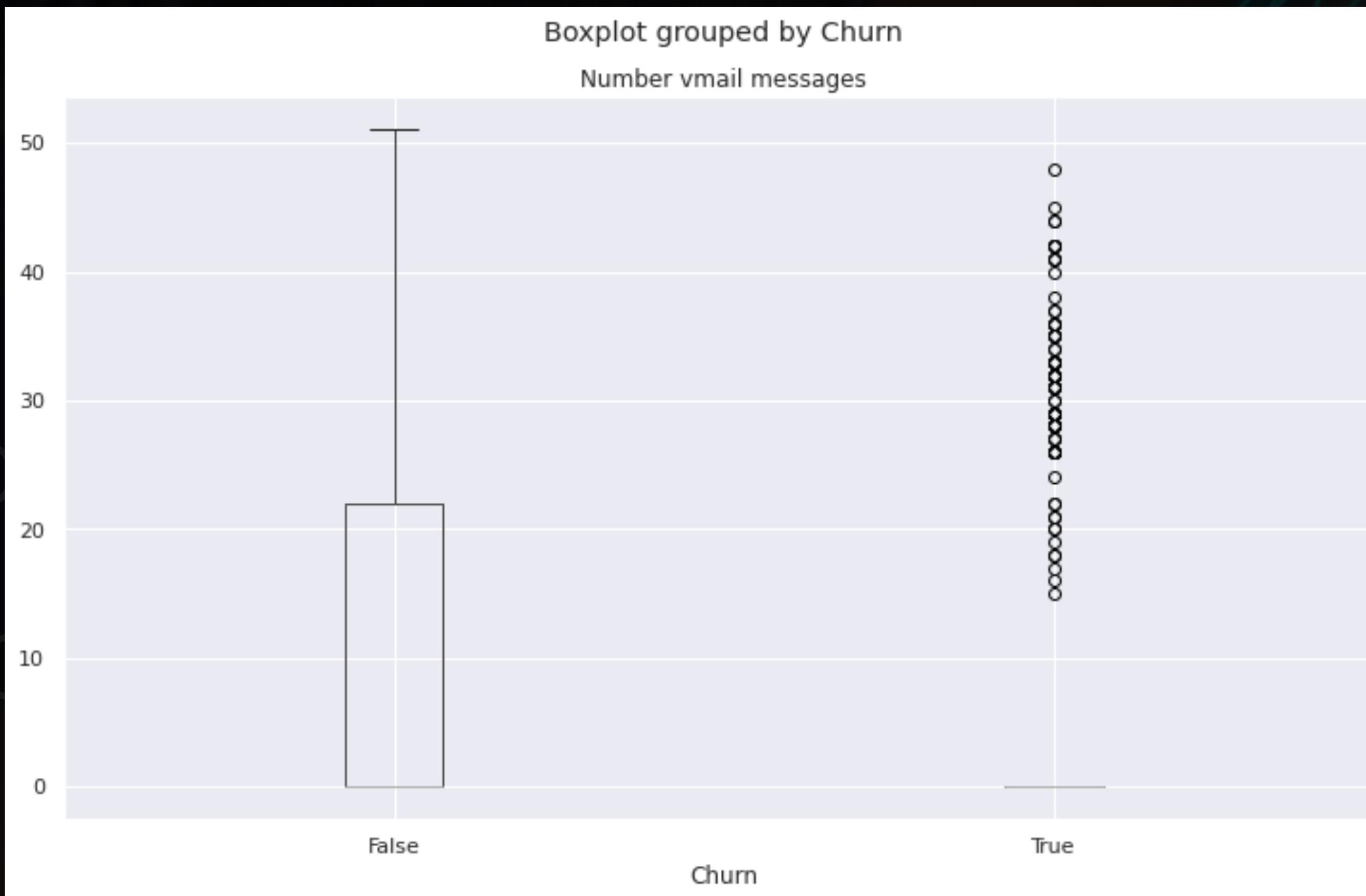
Report :

As we can see there is no clear relation between voice.plan and churn. There is no much difference.

Visualizations



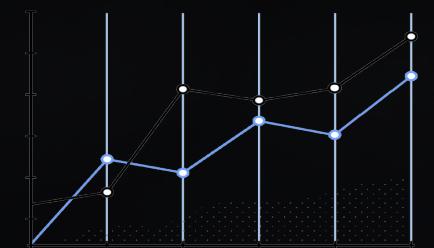
Analyzing Voice.msgs vs Churn



Report :

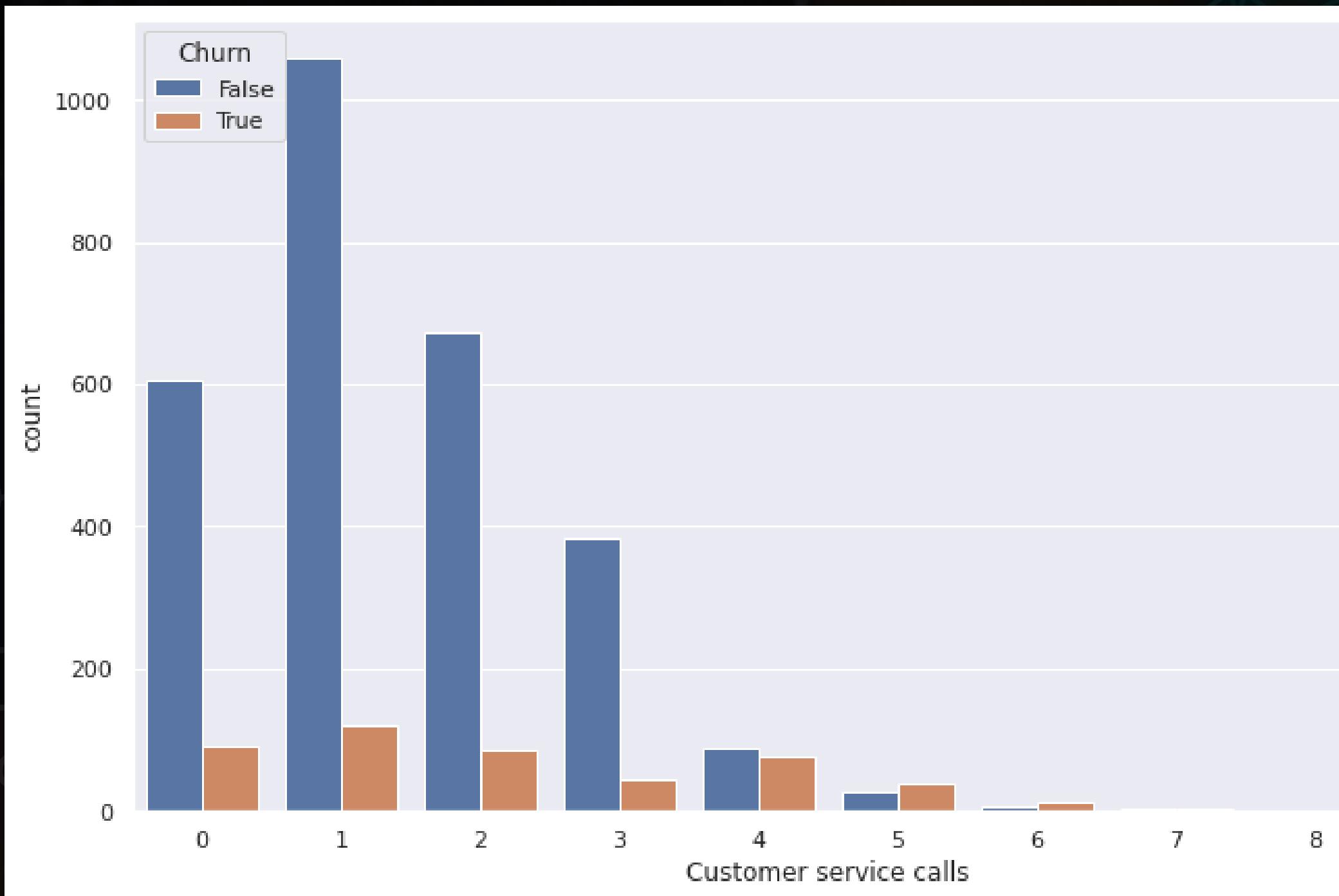
After analyzing the voice mail feature data we get an insight that when there are more than 20 voice-mail messages then there is a churn.

Visualizations



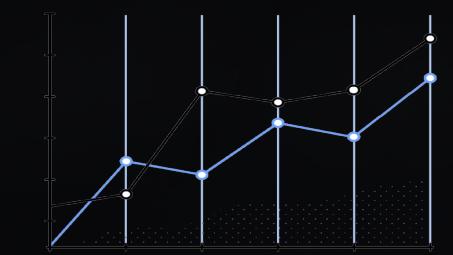
Analyzing Cust.calls vs Churn

Report :

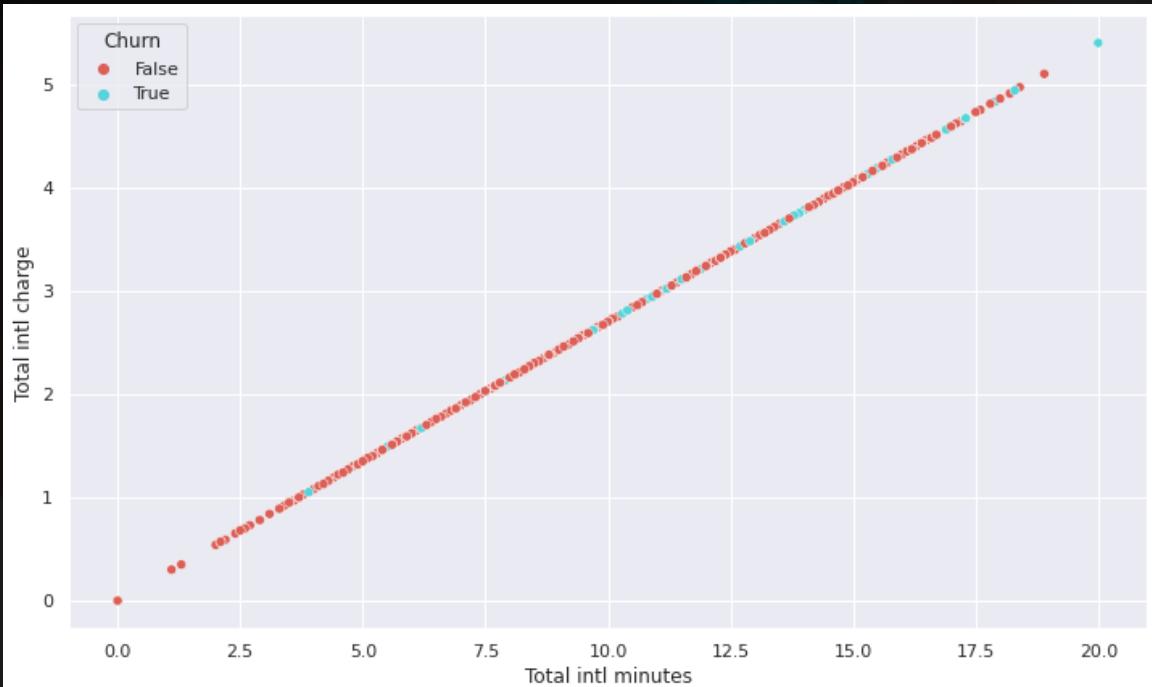
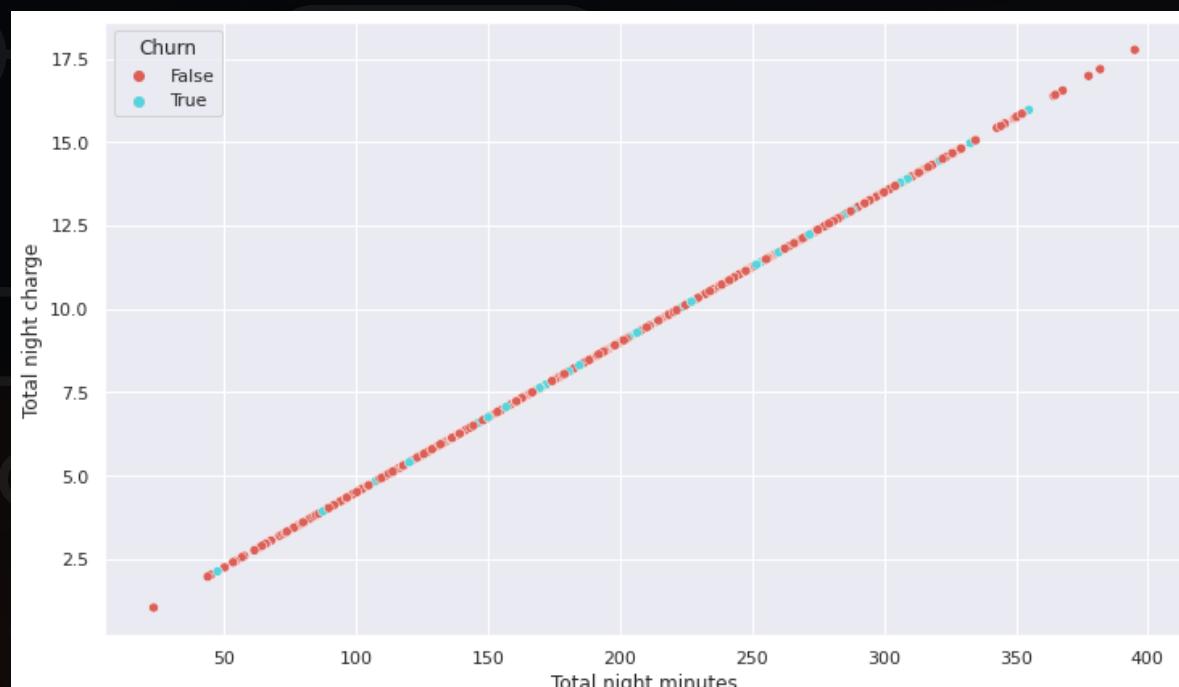
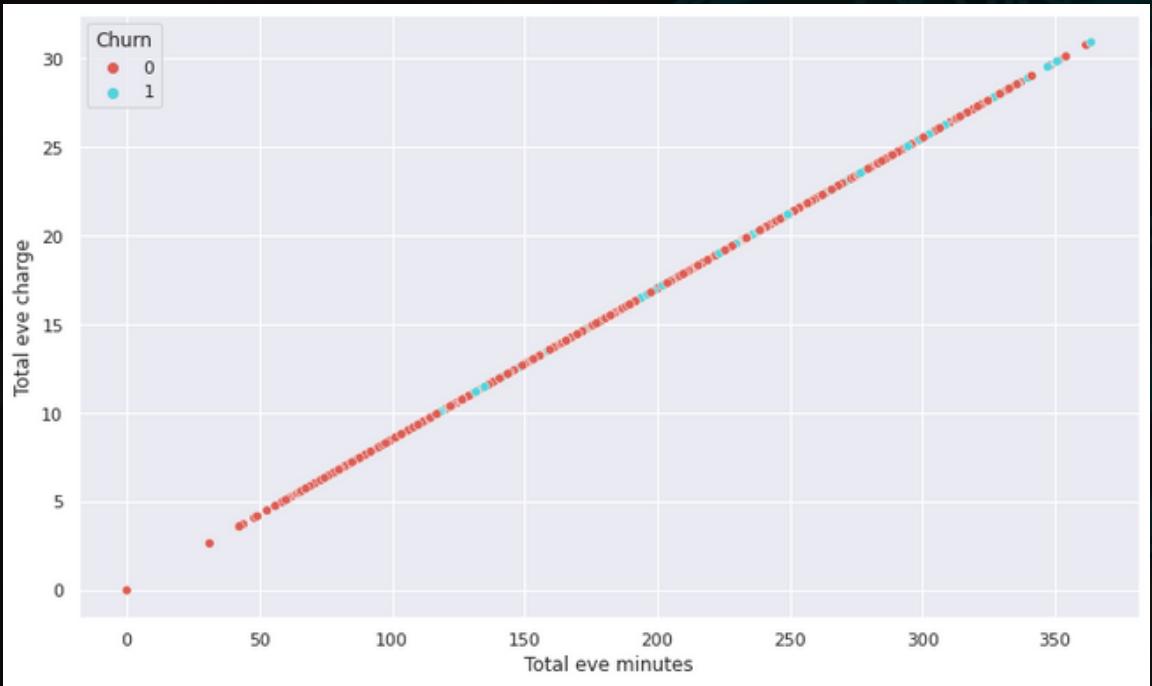
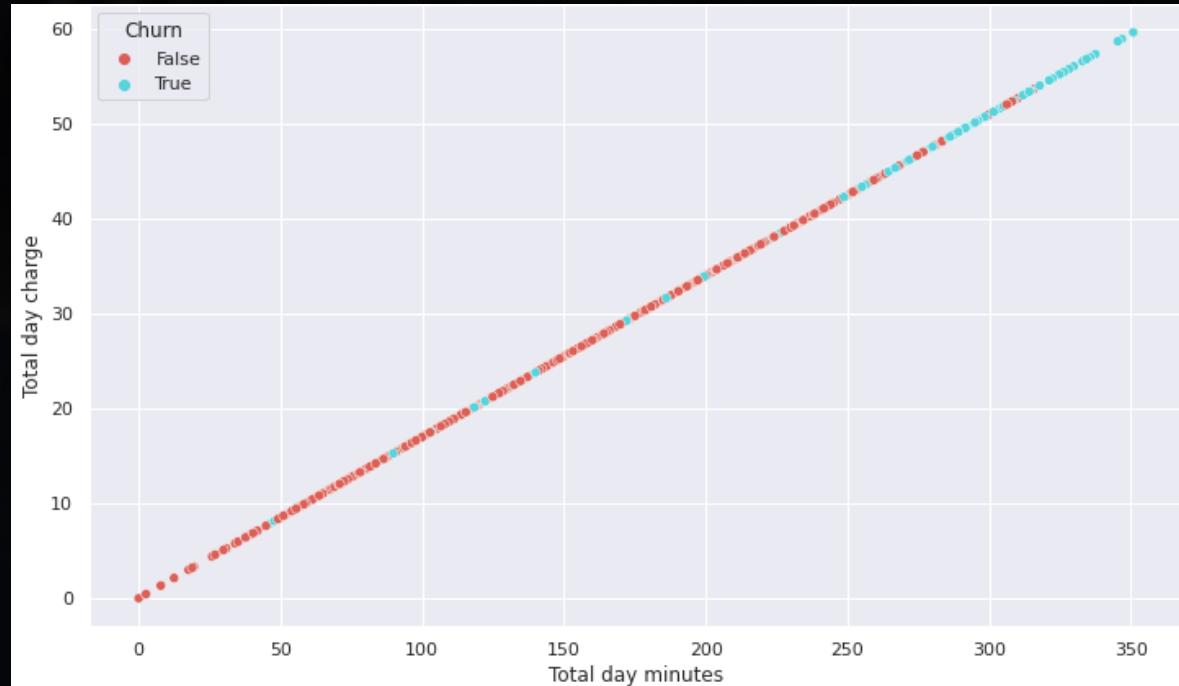


By analysis, mostly because of bad customer service, people tend to leave the operator. The data indicating that those customers who called the service center 5 times or above those customer churn percentage is higher than 60% and customers who have called once also have a high churn rate indicating their issue was not solved in the first attempt. So operator should work to improve the service call.

Visualizations



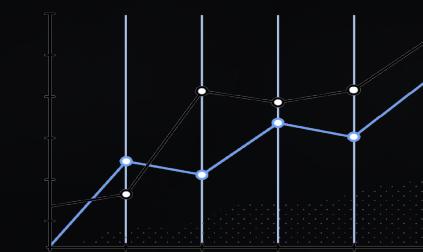
Analyzing all Day.Eve.Night.Intl. (calls,mins,charge) vs Churn



Report :

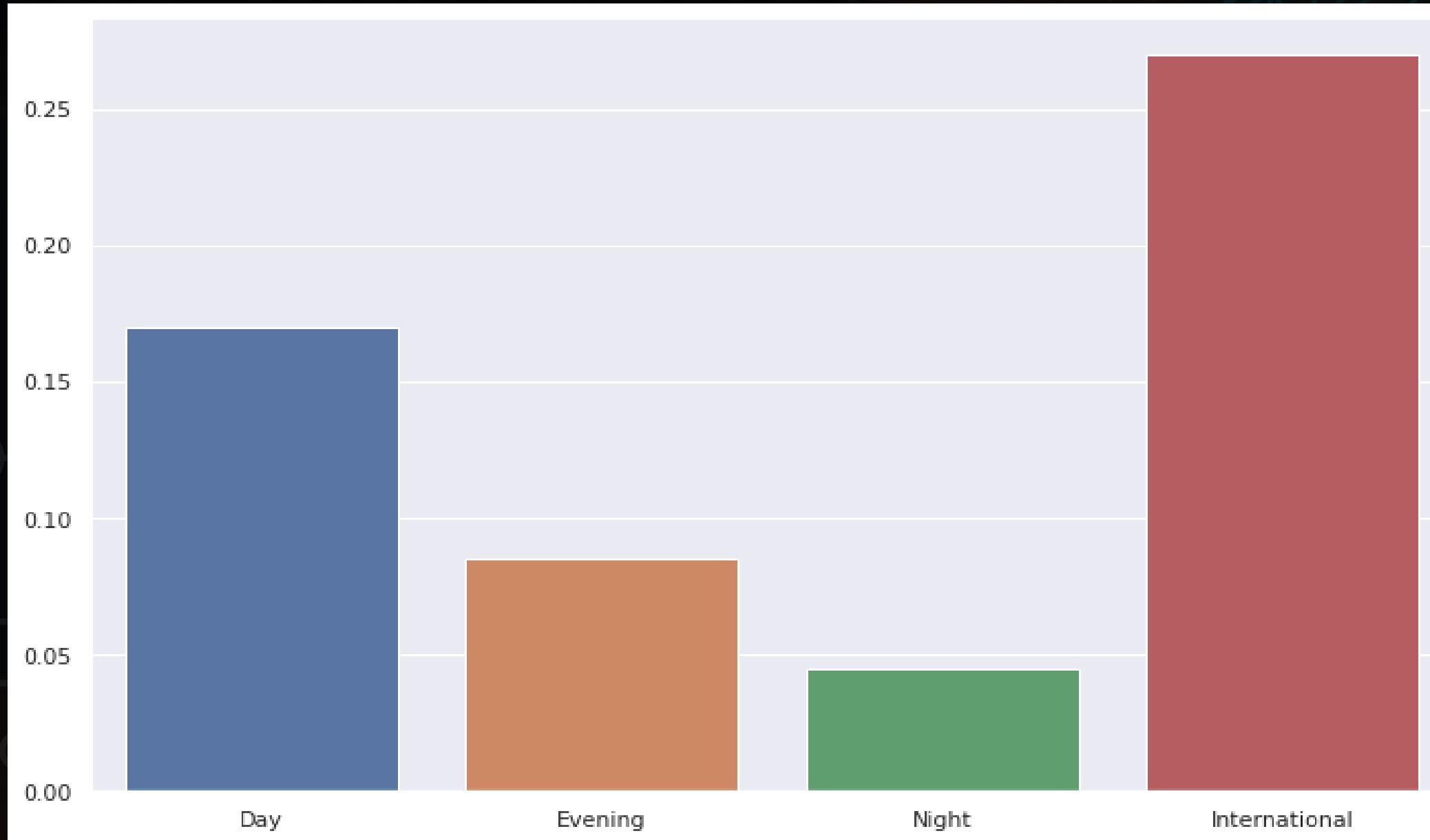
As these data sets are numerical data type, so for analyzing with the 'churn' which is a categorical data set, we are using mean, median.

Visualizations



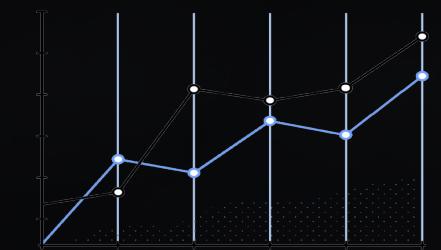
Analyzing relation between overall call charge
and overall call minutes

Report :



After analyzing the dataset we have noticed that total day/night/eve .mins/call/charges are not put any kind of cause for churn rate. But international call charges are high as compare to others it's an obvious thing but that may be a cause for international plan customers to churn out.

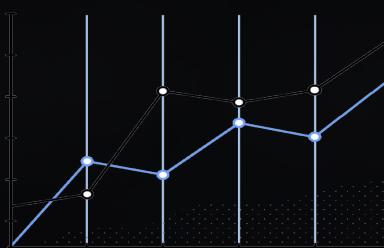
Visualizations



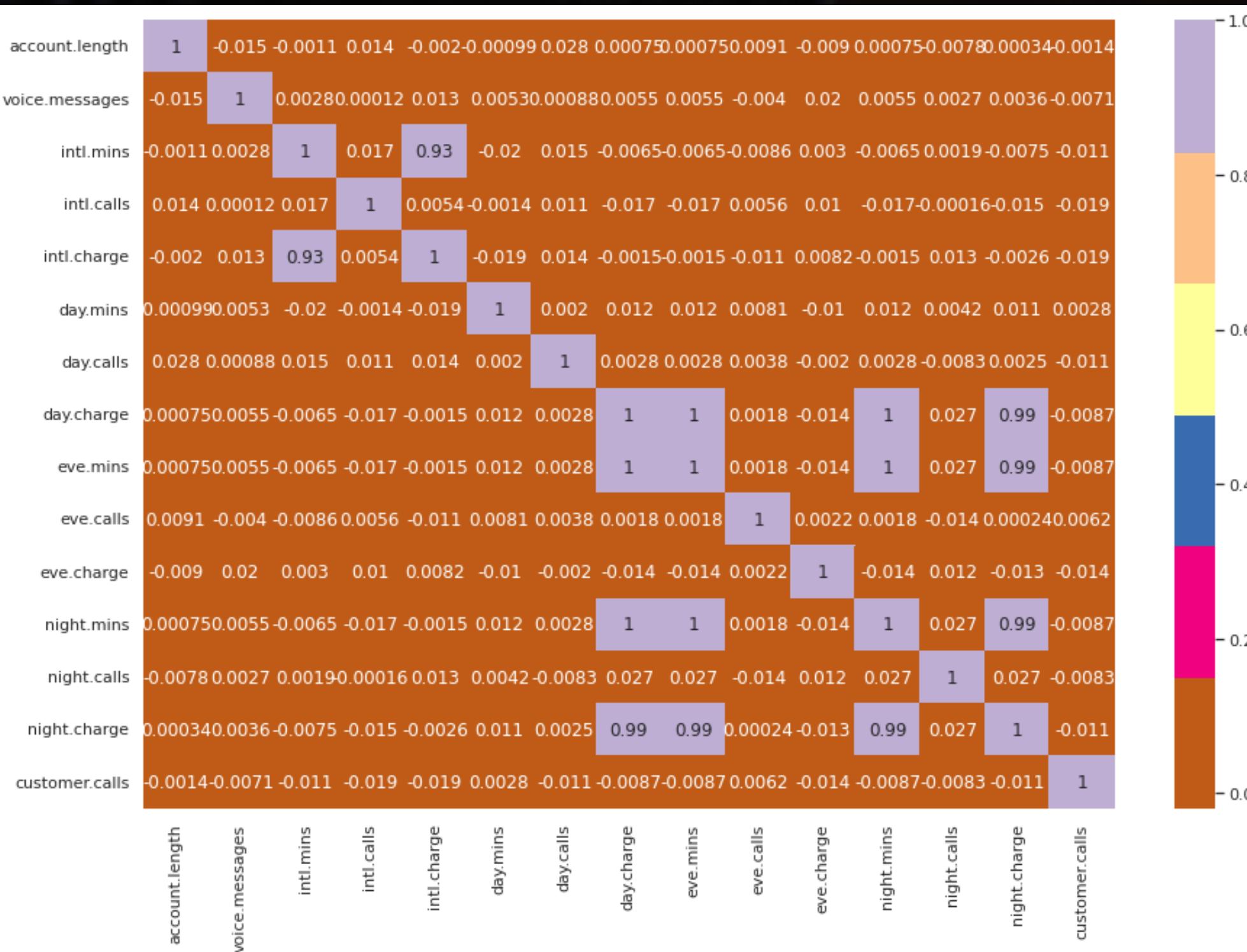
Graphical Analysis

- 1. Univariate Analysis :** we analyze data over a single column from the numerical dataset, for this we use 3 types of plot which are box plot, strip plot, dist plot
- 2. Bivariate Analysis :** We analyze data by taking two columns into consideration from a dataset, here we only take numerical data type column, for this visualization we use Box plot, scatter plot
- 3. Multivariate Analysis :** We analyze data by taking more than two columns into consideration from a dataset, for this we using correlation plot, correlation matrix, correlation heatmap, pair plot

Visualizations



Analyzing Correlated Variables (HeatMap)



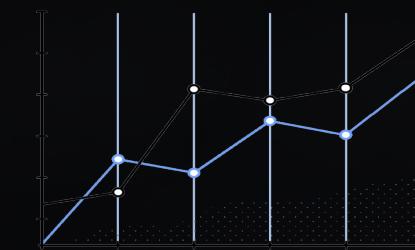
Report:

$|r| = [>0.8]$ = Strong Correlation

$|r| = [0.5-0.8]$ = Moderate Correlation

$|r| = [<0.5]$ = Weak Correlation

Visualizations



Analyzing summary from all the visualizations

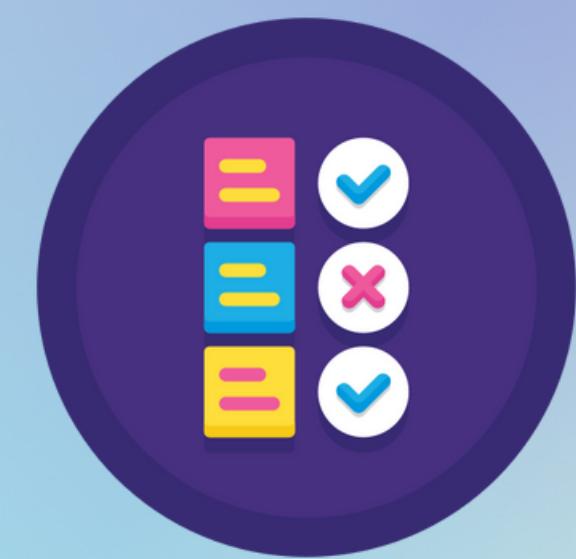
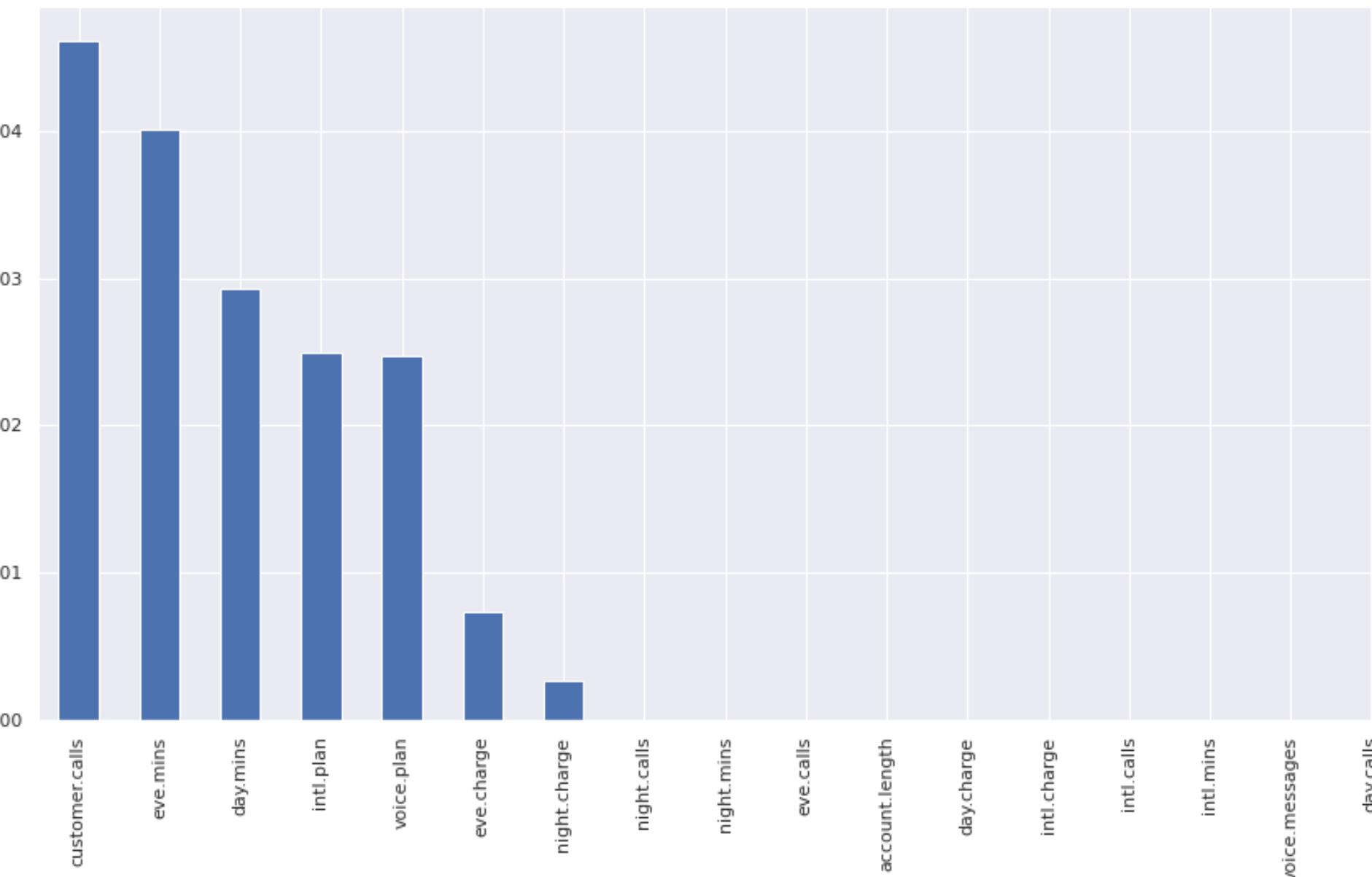
Report :

- Improve network coverage churned state
- In international plan provide some discount plan to the customer
- Improve the voice mail quality or take feedback from the customer
- Improve the service of call center and take frequently feedback from the customer regarding their issue and try to solve it as soon as possible

Feature Selection

1) Performing mutual info regression

```
[ ] from sklearn.feature_selection import mutual_info_regression  
# determine the mutual information  
mutual_info=mutual_info_regression(x_train,y_train)  
mutual_info
```



Report:

By the plot we can observe
"cust.calls,eve.min,day.min,intl.plan,voice.plan,nit.pcharge,eve.charge are Important"

Feature Selection

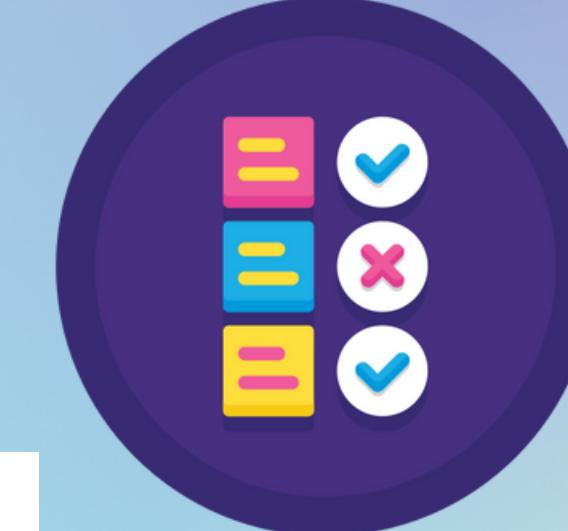
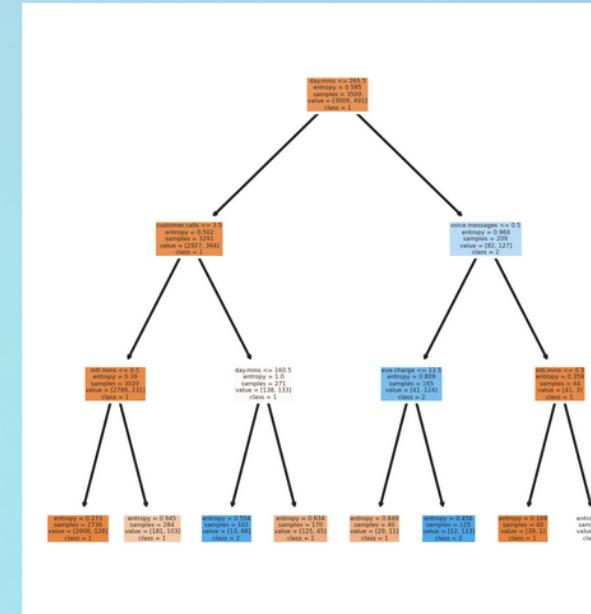
2) Performing Decision Tree Classifier Using Entropy Criteria (C5.0)

```
[53] from sklearn.tree import DecisionTreeClassifier  
from sklearn import tree  
from sklearn.metrics import classification_report  
from sklearn import preprocessing
```

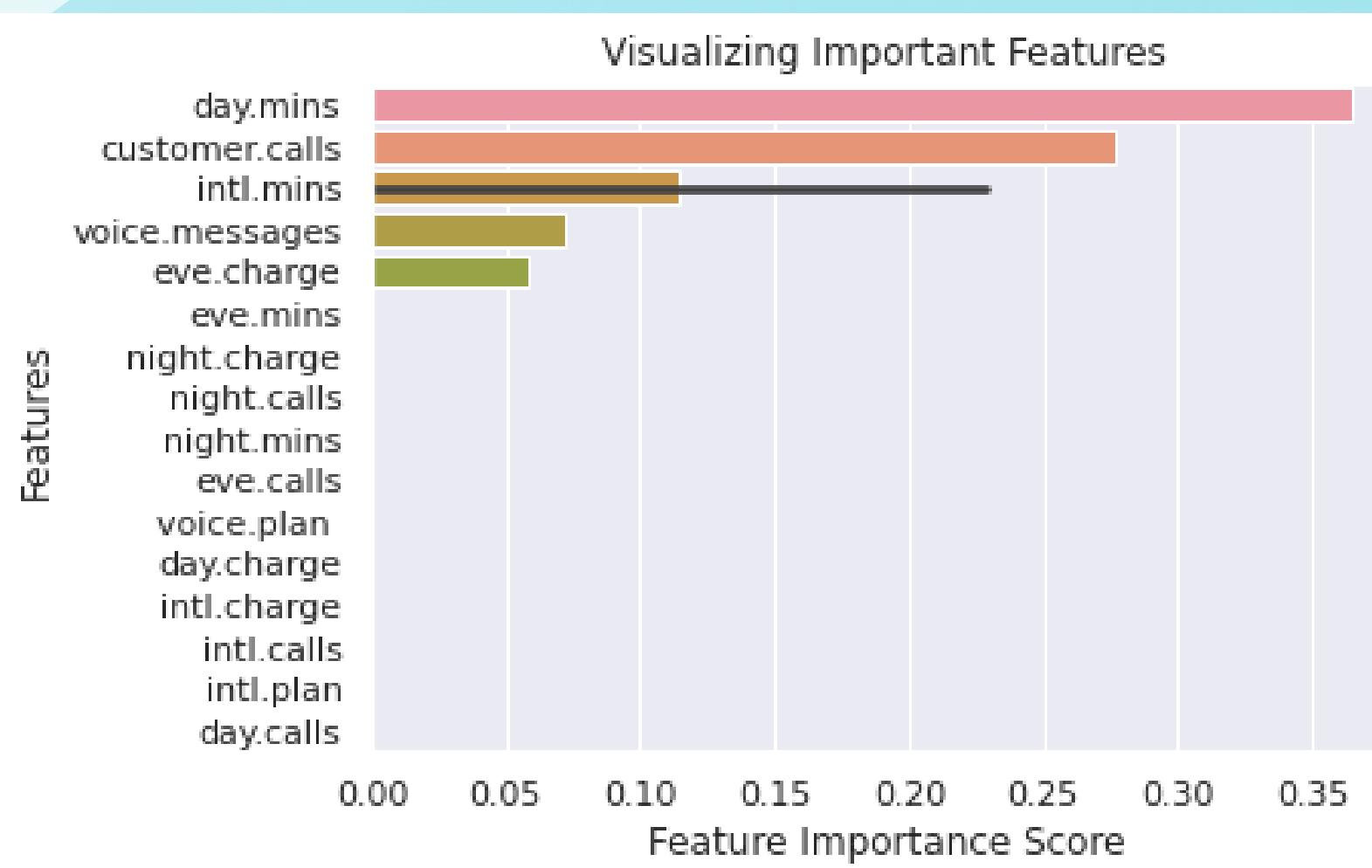
```
[54] # Pruning the Decision Tree  
model = DecisionTreeClassifier(criterion = 'entropy',max_depth=3)  
model.fit(x_train,y_train)  
  
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

```
[55] # Plot the decision tree  
from sklearn import tree  
tree.plot_tree(model);
```

Tree



```
# Accuracy  
np.mean(preds==y_test)
```


0.9046666666666666

Report:

By the plot we can observe
"cust.calls,intl.min,day.min,voi
ce.msgs,charge,eve.charge are
Important"



Categorical to Numerical

```
# Encoding categorical columns  
df3=pd.get_dummies(df2,columns=['state','area.code'])  
df3
```

	account.length	voice.plan	voice.messages	intl.plan	intl.mins	intl.calls	intl.charge	day.mins
0	128	1	25	0	10	3	2	265
1	107	1	26	0	13	3	3	161
2	137	0	0	0	12	5	3	243
3	84	0	0	1	6	7	1	299
4	75	0	0	1	10	3	2	166

Smote for Imbalanced data

Resampling data is one of the most commonly preferred approaches to deal with an imbalanced dataset. There are broadly two types of methods for this i) Under sampling ii) Oversampling. In most cases, oversampling is preferred over under sampling techniques. The reason being, in under sampling we tend to remove instances from data that may be carrying some important information.

```
# Before balancing  
y.value_counts()  
  
churn  
0      4264  
1      785  
dtype: int64
```



```
# After balancing  
from imblearn.over_sampling import SMOTE  
smote= SMOTE(sampling_strategy ='minority')  
x_sm,y_sm=smote.fit_resample(x,y)  
y_sm.value_counts()  
  
churn  
0      4264  
1      4264  
dtype: int64
```

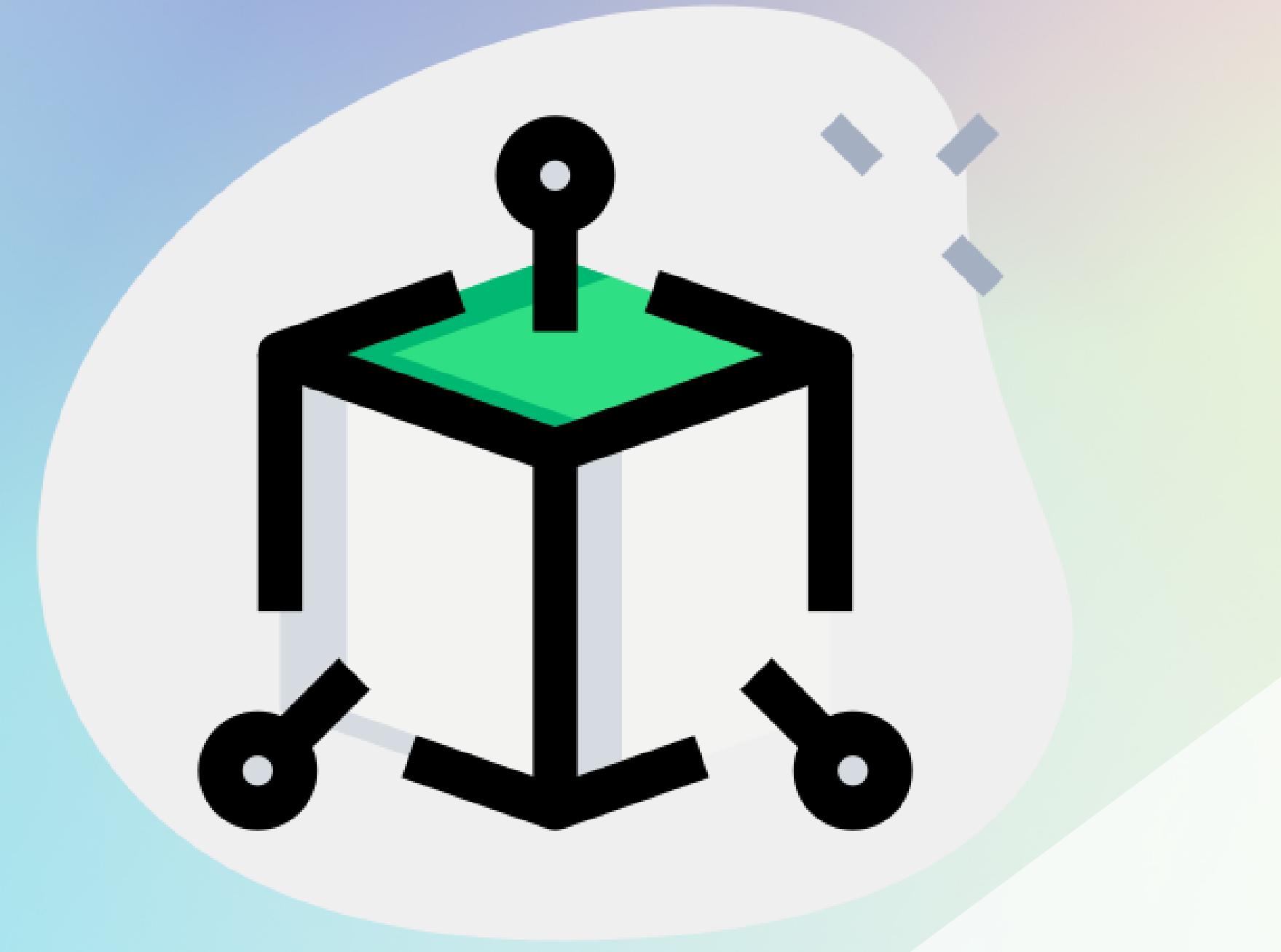
Scaling down the data to keep the values in same range

```
# standardizing the data
# standardizing can help features arrive in more digestible form for these algorithms
scaler=preprocessing.StandardScaler()
a=scaler.fit_transform(x)
a

array([[ 0.69894149,  1.66712012,  1.27314513, ..., -0.58012157,
       1.002002 , -0.57611859],
       [ 0.16984882,  1.66712012,  1.3469729 , ..., -0.58012157,
       1.002002 , -0.57611859],
       [ 0.92569549, -0.5998368 , -0.57254912, ..., -0.58012157,
       1.002002 , -0.57611859],
       ...,
       [-0.98911606, -0.5998368 , -0.57254912, ..., -0.58012157,
       1.002002 , -0.57611859],
       [ 0.2202386 , -0.5998368 , -0.57254912, ..., -0.58012157,
       -0.998002 ,  1.73575375],
       [-0.35924384,  1.66712012,  1.93759506, ..., -0.58012157,
       1.002002 , -0.57611859]])
```

Model Building

- Gradient Boost Classifier
- Support Vector Machines
- Random Forest Classifier
- K Nearest Neighbor Classifier
- Decision Tree Classifier
- Logistic Regression



Gradient Boost Classifier (GBC)

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
skfolds=StratifiedKFold(n_splits=10,random_state=42,shuffle=True)

from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier(random_state=42)

for train_index, test_index in skfolds.split(x,y):
    clone_clf=clone(gbc)
    x_train_folds=a[train_index]
    y_train_folds=(y[train_index])
    x_test_fold=a[test_index]
    y_test_fold=(y[test_index])
    clone_clf.fit(x_train_folds,y_train_folds)
    y_pred=clone_clf.predict(x_test_fold)
    n_correct=sum(y_pred==y_test_fold)
    print("Results for GB",n_correct/len(y_pred))
```

Results :

Results for GB 0.95
Results for GB 0.962
Results for GB 0.956
Results for GB 0.954
Results for GB 0.96
Results for GB 0.946
Results for GB 0.948
Results for GB 0.932
Results for GB 0.948
Results for GB 0.948

Support Vector Machine (SVM)

```
from sklearn.svm import SVC  
svc_clf=SVC(random_state=42)  
  
for train_index, test_index in skfolds.split(x,y):  
    clone_clf=clone(svc_clf)  
    x_train_folds=a[train_index]  
    y_train_folds=(y[train_index])  
    x_test_fold=a[test_index]  
    y_test_fold=(y[test_index])  
    clone_clf.fit(x_train_folds,y_train_folds)  
    y_pred=clone_clf.predict(x_test_fold)  
    n_correct=sum(y_pred==y_test_fold)  
    print("Results for SVM",n_correct/len(y_pred))
```

Results :

Results for SVM 0.868
Results for SVM 0.866
Results for SVM 0.87
Results for SVM 0.868
Results for SVM 0.868
Results for SVM 0.87
Results for SVM 0.872
Results for SVM 0.868
Results for SVM 0.86
Results for SVM 0.864

Random Forest Classifier (RF)

```
from sklearn.ensemble import RandomForestClassifier  
rf_clf=RandomForestClassifier(random_state=42)  
  
for train_index, test_index in skfolds.split(x,y):  
    clone_clf=clone(rf_clf)  
    x_train_folds=a[train_index]  
    y_train_folds=(y[train_index])  
    x_test_fold=a[test_index]  
    y_test_fold=(y[test_index])  
    clone_clf.fit(x_train_folds,y_train_folds)  
    y_pred=clone_clf.predict(x_test_fold)  
    n_correct=sum(y_pred==y_test_fold)  
    print("Results for RFC",n_correct/len(y_pred))
```

Results :

Results for RFC 0.928
Results for RFC 0.946
Results for RFC 0.926
Results for RFC 0.934
Results for RFC 0.946
Results for RFC 0.936
Results for RFC 0.94
Results for RFC 0.916
Results for RFC 0.908
Results for RFC 0.934

K Nearest Neighbor Classifier (KNN)

```
from sklearn.neighbors import KNeighborsClassifier  
knn_clf=KNeighborsClassifier()  
  
for train_index, test_index in skfolds.split(x,y):  
    clone_clf=clone(gbc)  
    x_train_folds=a[train_index]  
    y_train_folds=(y[train_index])  
    x_test_fold=a[test_index]  
    y_test_fold=(y[test_index])  
    clone_clf.fit(x_train_folds,y_train_folds)  
    y_pred=clone_clf.predict(x_test_fold)  
    n_correct=sum(y_pred==y_test_fold)  
    print("Results for KNN",n_correct/len(y_pred))
```

Results :

Results for KNN 0.95
Results for KNN 0.962
Results for KNN 0.956
Results for KNN 0.954
Results for KNN 0.96
Results for KNN 0.946
Results for KNN 0.948
Results for KNN 0.932
Results for KNN 0.948
Results for KNN 0.948

Decision Tree Classifier (DT)

```
from sklearn.tree import DecisionTreeClassifier  
dt_clf=DecisionTreeClassifier(random_state=42)  
  
for train_index, test_index in skfolds.split(x,y):  
    clone_clf=clone(gbc)  
    x_train_folds=a[train_index]  
    y_train_folds=(y[train_index])  
    x_test_fold=a[test_index]  
    y_test_fold=(y[test_index])  
    clone_clf.fit(x_train_folds,y_train_folds)  
    y_pred=clone_clf.predict(x_test_fold)  
    n_correct=sum(y_pred==y_test_fold)  
    print("Results for DT",n_correct/len(y_pred))
```

Results :

Results for DT 0.95
Results for DT 0.962
Results for DT 0.956
Results for DT 0.954
Results for DT 0.96
Results for DT 0.946
Results for DT 0.948
Results for DT 0.932
Results for DT 0.948
Results for DT 0.948

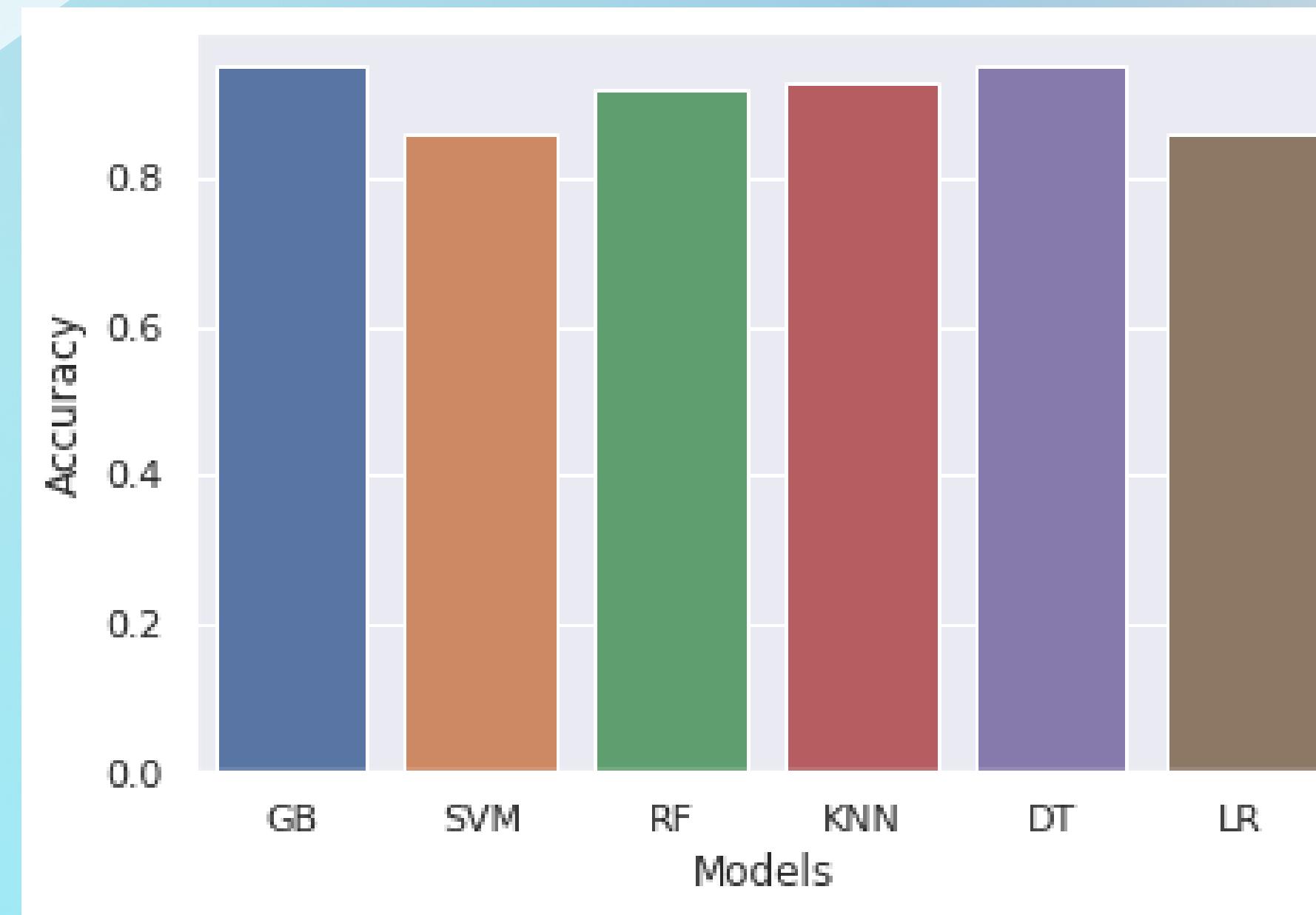
Logistic Regression (LR)

```
from sklearn.linear_model import LogisticRegression  
lr_clf=LogisticRegression(random_state=42)  
  
for train_index, test_index in skfolds.split(x,y):  
    clone_clf=clone(lr_clf)  
    x_train_folds=a[train_index]  
    y_train_folds=(y[train_index])  
    x_test_fold=a[test_index]  
    y_test_fold=(y[test_index])  
    clone_clf.fit(x_train_folds,y_train_folds)  
    y_pred=clone_clf.predict(x_test_fold)  
    n_correct=sum(y_pred==y_test_fold)  
    print("Results for LR",n_correct/len(y_pred))
```

Results :

Results for LR 0.868
Results for LR 0.872
Results for LR 0.874
Results for LR 0.858
Results for LR 0.858
Results for LR 0.872
Results for LR 0.878
Results for LR 0.856
Results for LR 0.854
Results for LR 0.862

Analyzing Best Model



Gradient Boost Classifier (GBC)

Cross Validation



K-Fold :

```
from sklearn.model_selection import KFold
model=GradientBoostingClassifier(random_state=42)
kfold_validation=KFold(10)
import numpy as np
from sklearn.model_selection import cross_val_score
results=cross_val_score(model,x,y,cv=kfold_validation)
print(results)
print(np.mean(results))

[0.96 0.948 0.958 0.948 0.952 0.932 0.964 0.934 0.948 0.974]
0.9518000000000001
```

K-Folds cross validation is one method that attempts to maximize the use of the available data for training and then testing a model. It is particularly useful for assessing model performance, as it provides a range of accuracy scores across (somewhat) different data sets.

Cross Validation



Stratified K-Fold :

```
from sklearn.model_selection import StratifiedKFold  
skfold=StratifiedKFold(n_splits=5)  
model=GradientBoostingClassifier(random_state=42)  
scores=cross_val_score(model,x,y,cv=skfold)  
print(np.mean(scores))  
  
0.9517999999999999
```

Implementing the concept of stratified sampling in cross-validation ensures the training and test sets have the same proportion of the feature of interest as in the original dataset. Doing this with the target variable ensures that the cross-validation result is a close approximation of generalization error.

Model Deployment



Model deployment is the process of putting machine learning models into production. This makes the model's predictions available to users, developers or systems, so they can make business decisions based on data, interact with their application (like recognize a face in an image) and so on.

- **Step 1: Create a new virtual environment using Pickle library.**
- **Step 2: Install necessary libraries.**
- **Step 3: Build the best machine learning model and Save it.**
- **Step 4: Test the loaded model.**
- **Step 5: Create main.py file.**
- **Step 6: Open local path in prompt and deploy to predict.**

Model Deployment



Saving the final model :

```
In [142]: 1 pipe=Pipeline(steps=[('step1',GradientBoostingClassifier())])
           2 pipe.fit(x_train,y_train)

Out[142]: Pipeline(steps=[('step1', GradientBoostingClassifier())])

In [143]: 1 y_pred=pipe.predict(x_test)

In [144]: 1 pipe.predict_proba(x_test)[10]

Out[144]: array([0.96621062, 0.03378938])

In [145]: 1 pipe1=Pipeline(steps=[('step1',RandomForestClassifier())])
           2 pipe1.fit(x_train,y_train)

Out[145]: Pipeline(steps=[('step1', RandomForestClassifier())])

In [146]: 1 pipe1.predict_proba(x_test)[10]

Out[146]: array([0.88, 0.12])

In [147]: 1 import pickle
           2 pickle.dump(pipe,open('pipe.pkl','wb'))

In [ ]: 1
```

Model Deployment



Creating Main
app.py file :

The screenshot shows a code editor window with a dark theme. On the left is a sidebar with various icons for file operations like copy, paste, search, and refresh. The main area displays a Python script named 'app.py' with line numbers from 1 to 37. The code uses Streamlit's API to build a user interface for a 'Customer Churn Predictor'. It imports Streamlit, sklearn, pandas, pickle, and warnings. It loads a trained model from 'pipe.pkl' and sets the title of the app. The script then creates a 2x2 grid of input fields for different customer features: 'Voice Plan', 'Voice Messages', 'International plan', 'International minutes', 'International calls', and 'International charge'. Finally, it creates another 2x2 grid of columns. The status bar at the bottom indicates the script has 76 lines, 64 columns, 4 spaces, and is in UTF-8 encoding, using Python 3.10.5 64-bit. The taskbar at the bottom shows icons for File Explorer, Edge browser, Task View, and other system tools.

```
C:\>Churn Project > app.py > ...
1 import streamlit as st
2 import sklearn
3 import pandas as pd
4 import pickle
5 import warnings
6 warnings.filterwarnings('ignore')
7
8 pipe=pickle.load(open('pipe.pkl','rb'))
9
10 st.title('Customer Churn Predictor')
11 col1,col2=st.columns(2)
12
13 with col1:
14     Voice_Plan=st.number_input('Select Voice plan is active or not')
15
16 with col2:
17     Voice_Messages=st.number_input('Number of voicemail messages')
18
19 col3,col4=st.columns(2)
20
21 with col3:
22     International_plan=st.number_input('International plan is active or not')
23
24 with col4:
25     International_min=st.number_input('How many minutes customer used service to make international calls')
26
27 col5,col6=st.columns(2)
28
29 with col5:
30     International_calls=st.number_input('Total number of international calls')
31
32 with col6:
33     International_charge=st.number_input('Total international charge')
34
35
36 col7,col8=st.columns(2)
37
```

Model Deployment



Cmd prompt :

```
C:\Windows\System32\cmd.exe - streamlit run app.py
Microsoft Windows [Version 10.0.22000.1455]
(c) Microsoft Corporation. All rights reserved.

C:\Churn Project>python -m venv .

C:\Churn Project>cd scripts

C:\Churn Project\Scripts>activate

(Churn Project) C:\Churn Project\Scripts>cd..

(Churn Project) C:\Churn Project>pip install streamlit
Requirement already satisfied: streamlit in c:\churn project\lib\site-packages (1.18.0)
Requirement already satisfied: pandas>=0.25 in c:\churn project\lib\site-packages (from streamlit) (1.5.3)
Requirement already satisfied: altair>=3.2.0 in c:\churn project\lib\site-packages (from streamlit) (4.2.2)
Requirement already satisfied: cachetools>=4.0 in c:\churn project\lib\site-packages (from streamlit) (5.3.0)
Requirement already satisfied: protobuf<4,>=3.12 in c:\churn project\lib\site-packages (from streamlit) (3.20.3)
Requirement already satisfied: semver in c:\churn project\lib\site-packages (from streamlit) (2.13.0)
Requirement already satisfied: pyyaml>=0.9 in c:\churn project\lib\site-packages (from streamlit) (1.0.1)
Requirement already satisfied: typing-extensions>=3.10.0.0 in c:\churn project\lib\site-packages (from streamlit) (4.4.0)
Requirement already satisfied: tzlocal>=1.1 in c:\churn project\lib\site-packages (from streamlit) (4.2)
Requirement already satisfied: watchdog in c:\churn project\lib\site-packages (from streamlit) (2.2.1)
Requirement already satisfied: click>=7.0 in c:\churn project\lib\site-packages (from streamlit) (8.1.3)
Requirement already satisfied: blinker>=1.0.0 in c:\churn project\lib\site-packages (from streamlit) (1.5)
Requirement already satisfied: numpy in c:\churn project\lib\site-packages (from streamlit) (1.24.2)
Requirement already satisfied: importlib-metadata>=1.4 in c:\churn project\lib\site-packages (from streamlit) (6.0.0)
Requirement already satisfied: gitpython!=3.1.19 in c:\churn project\lib\site-packages (from streamlit) (3.1.30)
Requirement already satisfied: packaging>=14.1 in c:\churn project\lib\site-packages (from streamlit) (23.0)
Requirement already satisfied: tornado>=6.0.3 in c:\churn project\lib\site-packages (from streamlit) (6.2)
Requirement already satisfied: rich>=10.11.0 in c:\churn project\lib\site-packages (from streamlit) (13.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\churn project\lib\site-packages (from streamlit) (9.4.0)
Requirement already satisfied: validators>=0.2 in c:\churn project\lib\site-packages (from streamlit) (0.20.0)
Requirement already satisfied: pydeck>=0.1.dev5 in c:\churn project\lib\site-packages (from streamlit) (0.8.0)
Requirement already satisfied: toml in c:\churn project\lib\site-packages (from streamlit) (0.10.2)
Requirement already satisfied: pyarrow>=4.0 in c:\churn project\lib\site-packages (from streamlit) (11.0.0)
Requirement already satisfied: python-dateutil in c:\churn project\lib\site-packages (from streamlit) (2.8.2)
Requirement already satisfied: requests>=2.4 in c:\churn project\lib\site-packages (from streamlit) (2.28.2)
Requirement already satisfied: jinja2 in c:\churn project\lib\site-packages (from altair>=3.2.0->streamlit) (3.1.2)
Requirement already satisfied: toolz in c:\churn project\lib\site-packages (from altair>=3.2.0->streamlit) (0.12.0)
Requirement already satisfied: jsonschema>=3.0 in c:\churn project\lib\site-packages (from altair>=3.2.0->streamlit) (4.17.3)
Requirement already satisfied: entrypoints in c:\churn project\lib\site-packages (from altair>=3.2.0->streamlit) (0.4)
Requirement already satisfied: colorama in c:\churn project\lib\site-packages (from click>=7.0->streamlit) (0.4.6)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\churn project\lib\site-packages (from gitpython!=3.1.19->streamlit) (4.0.10)
Requirement already satisfied: zipp>=0.5 in c:\churn project\lib\site-packages (from importlib-metadata>=1.4->streamlit) (3.13.0)
Requirement already satisfied: pytz>=2020.1 in c:\churn project\lib\site-packages (from pandas>=0.25->streamlit) (2022.7.1)
Requirement already satisfied: six>=1.5 in c:\churn project\lib\site-packages (from python-dateutil->streamlit) (1.16.0)
Requirement already satisfied: idna<4,>=2.5 in c:\churn project\lib\site-packages (from requests>=2.4->streamlit) (3.4)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\churn project\lib\site-packages (from requests>=2.4->streamlit) (3.0.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\churn project\lib\site-packages (from requests>=2.4->streamlit) (2022.12.7)
```

Model Deployment



Churn Predictor
Webpage :

Customer Churn Predictor

Select Voice plan is active or not : 1.00 Number of voicemail messages : 25.00

International plan is active or not : 0.00 How many minutes customer used service to make international calls : 10.00

Total number of international calls : 3.00 Total international charge : 2.70

Total number of calls to customer service : 1.00 Total number of minutes customer used service during the day : 707.20

Total charge for calls : 72.86 Total number of Call in a Day : 300.00

Predict Probability

Chances of customer Staying:-96%