SyriaTelco_Collins_repo- / index.ipynb

Collins-Opiyo adjusted the notebook subheadings 8c31542 · 1 minute ago

7561 lines (7561 loc) · 452 KB

SyriaTel Customer Churn Prediction

Overview

For Telco companies it is key to attract new customers and at the same time avoid contract terminations (=churn) to grow their revenue generating base. Looking at churn, different reasons trigger customers to terminate their contracts, for example better price offers, more interesting packages, bad service experiences or change of customers' personal situations. The churn metric is expressed as the percentage of customers who cancel their contract or subscription within a specific period, typically a month. For example, if SyriaTel had 10 million customers at the beginning of January and 500,000 customers terminated their contracts by the end of January, the monthly churn rate for January would be 5%. This project is geared towards predicting and reducing customer churn for SyriaTel by analyzing customer behavior and applying machine learning models to identify high-risk customers and implement retention strategies.

1. Business UnderStanding

Problem Statement SyriaTel is a prominent telecommunications provider in Syria, offering a range of services including mobile and fixed-line voice communication, data services, and broadband internet. The company aims to expand its market share and enhance customer satisfaction while maintaining a strong and competitive position in the telecom industry. SyriaTel is facing a high churn rate, with many customers discontinuing their services and switching to competitors. The company wants to address this issue by developing a customer churn prediction model. By analyzing the dataset, SyriaTel aims to gain insights into factors associated with churn, with the goal of reducing churn rate, increasing customer retention, and improving overall profitability.

Specific Objectives

1. Identify the factors that are most likely to lead to customer churn.

2. Develop a model that can accurately predict which customers are at risk of churning.

3. Take proactive steps to retain customers who are at risk of churning.

Success Metrics

- Developing a robust churn prediction model with high recall score of 0.8.

- Identifying the key features and factors that significantly contribute to customer churn.

- Providing actionable insights and recommendations to the telecom company for reducing churn and improving customer retention.

- Demonstrating the value of churn prediction models in enabling proactive retention strategies and reducing revenue losses due to customer churn.

Import libraries and packages

In [120…

```
# Data manipulation
import pandas as pd
import numpy as np

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import plotly.express as px
import category_encoders as ce
%matplotlib inline

# Modeling
from sklearn.model_selection import train_test_split,cross_val_score,GridSearc
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE,SMOTENC
from sklearn.metrics import f1_score,recall_score,precision_score,confusion_ma
```

```
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from sklearn import tree

# Algorithms for supervised learning methods
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder

# Filtering future warnings
import warnings
warnings.filterwarnings('ignore')
```

In [121…

```
# Load the dataset
df = pd.read_csv('./data/bigml_59.csv')
df.head()
```

Out[121…

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	ch
0	KS	128	415	382-4657	no	yes	25	265.1	110	
1	OH	107	415	371-7191	no	yes	26	161.6	123	
2	NJ	137	415	358-1921	no	no	0	243.4	114	
3	OH	84	408	375-9999	yes	no	0	299.4	71	
4	OK	75	415	330-6626	yes	no	0	166.7	113	

5 rows × 21 columns

2. Exploratory data analysis

In [122…

```
shape = df.shape
print(f"The DataFrame has {shape[0]} rows and {shape[1]} columns.")
```

```
The DataFrame has 3333 rows and 21 columns.
```

In [123…

```
col_names = df.columns

col_names
```

Out[123…

```
Index(['state', 'account length', 'area code', 'phone number',
       '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')
```

Column Names and Descriptions:

Based on the column descriptions, below are further comments on some of them based on relevance for modelling or predicting house prices.

- **churn:** These columns represents the number of customers who are using and stop using the service. This is the target variable
- **number vmail message:** This column represents the number of voice mail messages sent.
- **total intl charge:** This column represents the amount charge for international calls.
- **total eve calls** This column represents the total evening calls.

In [124...

```
# view summary of dataset

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
```

Types of variables

In this section, I segregate the dataset into categorical and numerical variables. There are a mixture of categorical and numerical variables in the dataset. Categorical variables have data type object or bool. Numerical variables have data type float64 or int64.

First of all, I will find categorical variables.

In [125…

```
# Identify categorical variables
categorical_vars = df.select_dtypes(include=['object', 'bool']).columns

# Print categorical variables
print("Categorical variables:")
print(categorical_vars)
```

```
Categorical variables:
Index(['state', 'phone number', 'international plan', 'voice mail plan',
       'churn'],
      dtype='object')
```

In [126…

```
# view the categorical variables

df[categorical_vars].head()
```

Out[126…

	state	phone number	international plan	voice mail plan	churn
0	KS	382-4657	no	yes	False
1	OH	371-7191	no	yes	False
2	NJ	358-1921	no	no	False
3	OH	375-9999	yes	no	False
4	OK	330-6626	yes	no	False

Summary of categorical variables

- There are 5 categorical variables. These are given by `state`, `phone number`, `international plan`, `voive mail plan`, and `churn`.
- Churn is a binary categorical variables and is the target variable.

Explore problems within categorical variables

First, I will explore the categorical variables.

Missing values in categorical variables

In [127…

```
# check missing values in categorical variables

df[categorical_vars].isnull().sum()
```

Out[127…

```
state                 0
phone number          0
international plan    0
voice mail plan       0
churn                 0
dtype: int64
```

We can see that there no missing values in the categorical variables dataset.

Frequency counts of categorical variables

Now, I will check the frequency counts of categorical variables.

In [128…

```
# view frequency of categorical variables

for var in categorical_vars:

    print(df[var].value_counts())
```

```
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
VT     73
MI     73
ID     73
TX     72
UT     72
IN     71
MD     70
KS     70
MT     68
NC     68
NJ     68
CO     66
NV     66
WA     66
RI     65
MA     65
MS     65
AZ     64
MO     63
FL     63
ME     62
ND     62
NM     62
OK     61
DE     61
NE     61
SD     60
```

```
SC     60
KY     59
IL     58
NH     56
AR     55
GA     54
DC     54
HI     53
TN     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
409-5519    1
421-9144    1
369-8574    1
421-2659    1
334-4438    1
           ..
349-3843    1
388-6441    1
376-4271    1
353-1352    1
345-7117    1
Name: phone number, Length: 3333, dtype: int64
no     3010
yes     323
Name: international plan, dtype: int64
no     2411
yes     922
Name: voice mail plan, dtype: int64
False    2850
True      483
Name: churn, dtype: int64
```

In [129…

```
# View frequency distribution of categorical variables
for var in categorical_vars:
    # Calculate and print the frequency distribution as proportions
    freq_distribution = df[var].value_counts(normalize=True)
    print(f"Frequency distribution for {var}:")
    print(freq_distribution)
    print()
```

```
Frequency distribution for state:
WV    0.031803
MN    0.025203
NY    0.024902
AL    0.024002
OH    0.023402
OR    0.023402
WI    0.023402
VA    0.023102
WY    0.023102
CT    0.022202
VT    0.021902
MI    0.021902
```

```
ID    0.021902
TX    0.021602
UT    0.021602
IN    0.021302
MD    0.021002
KS    0.021002
MT    0.020402
NC    0.020402
NJ    0.020402
CO    0.019802
NV    0.019802
WA    0.019802
RI    0.019502
MA    0.019502
MS    0.019502
AZ    0.019202
MO    0.018902
FL    0.018902
ME    0.018602
ND    0.018602
NM    0.018602
OK    0.018302
DE    0.018302
NE    0.018302
SD    0.018002
SC    0.018002
KY    0.017702
IL    0.017402
NH    0.016802
AR    0.016502
GA    0.016202
DC    0.016202
HI    0.015902
TN    0.015902
AK    0.015602
LA    0.015302
PA    0.013501
IA    0.013201
CA    0.010201
Name: state, dtype: float64

Frequency distribution for phone number:
409-5519    0.0003
421-9144    0.0003
369-8574    0.0003
421-2659    0.0003
334-4438    0.0003
              ...
349-3843    0.0003
388-6441    0.0003
376-4271    0.0003
353-1352    0.0003
345-7117    0.0003
Name: phone number, Length: 3333, dtype: float64

Frequency distribution for international plan:
no     0.90309
yes    0.09691
Name: international plan, dtype: float64
```

```
Frequency distribution for voice mail plan:
no     0.723372
yes    0.276628
Name: voice mail plan, dtype: float64

Frequency distribution for churn:
False    0.855086
True     0.144914
Name: churn, dtype: float64
```

Number of labels: cardinality

The number of labels within a categorical variable is known as **cardinality**. A high number of labels within a variable is known as **high cardinality**. High cardinality may pose some serious problems in the machine learning model. So, I will check for high cardinality.

In [130…

```
for var in categorical_vars:

    print(var, ' contains ', len(df[var].unique()), ' labels')
```

```
state  contains  51  labels
phone number  contains  3333  labels
international plan  contains  2  labels
voice mail plan  contains  2  labels
churn  contains  2  labels
```

We can see that there is a `phone number` variable which needs to be preprocessed. I will do preprocessing in the following section.

All the other variables contain relatively smaller number of labels.

Feature Engineering of phonenumber Variable

In [131…

```
# Extracting phone codes (assuming phone numbers are in a specific format)
df['PhoneCode'] = df['phone number'].str[:3]
df['PhoneCode']
```

Out[131…

```
0       382
1       371
2       358
3       375
4       330
       ...
3328    414
3329    370
3330    328
3331    364
3332    400
Name: PhoneCode, Length: 3333, dtype: object
```

In [132…

```
# Verify the first few rows
print(df[['phone number', 'PhoneCode']].head())
```

```
  phone number PhoneCode
0     382-4657       382
1     371-7191       371
2     358-1921       358
3     375-9999       375
4     330-6626       330
```

In [133…

```
# Check for any unusual values in 'PhoneCode'
print(df['PhoneCode'].value_counts())
```

```
405    53
408    48
406    47
352    47
333    46
       ..
421    24
342    24
412    23
327    19
422    19
Name: PhoneCode, Length: 96, dtype: int64
```

In [134…

```
# Check for missing values in 'PhoneCode'
print(df['PhoneCode'].isnull().sum())
```

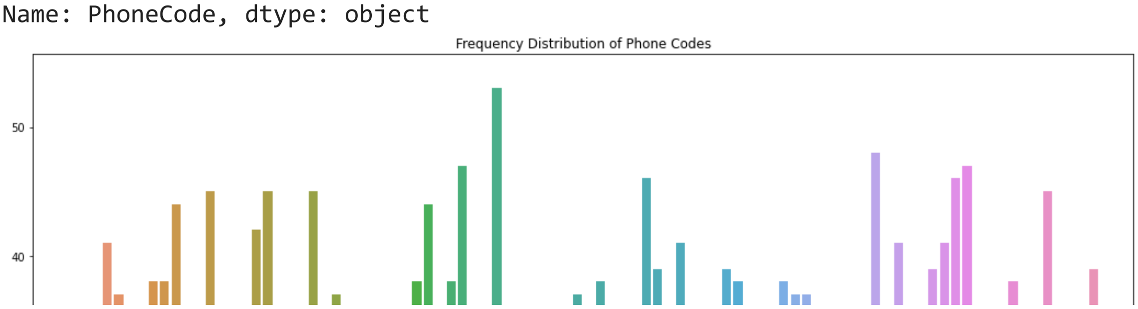
```
0
```

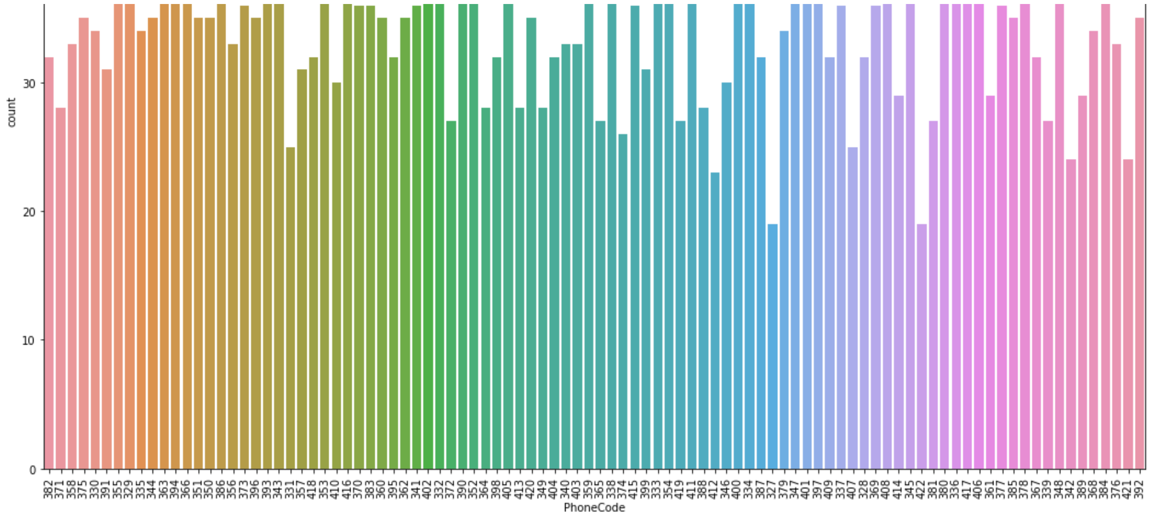
Phone_Code visualization

In [135…

```
# Summary statistics of the 'PhoneCode' column
print(df['PhoneCode'].describe())

# Plot the frequency distribution of phone codes
plt.figure(figsize=(18, 12))
sns.countplot(x='PhoneCode', data=df)
plt.xticks(rotation=90)
plt.title('Frequency Distribution of Phone Codes')
plt.show()
```

```
count     3333
unique      96
top        405
freq        53
Name: PhoneCode, dtype: object
```





In [136…

```
df['phone number'].dtypes
```

Out[136…

```
dtype('O')
```

We can see that the data type of `phone number` variable is object. I will parse the "PhoneCode" as object.

In [137…

```
# Ensure 'PhoneCode' is of object type
df['PhoneCode'] = df['PhoneCode'].astype('object')
```

In [138…

```
# again view the summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 22 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
 21  PhoneCode               3333 non-null   object
dtypes: bool(1), float64(8), int64(8), object(5)
memory usage: 550.2+ KB
```

We can see that there is an additional columns created from `PhoneCode` variable.

Now, I will drop the original `phone number` variable from the dataset.

In [139…

```
# drop the original 'phone number' variable

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

In [140…

```
# preview the dataset again

df.head()
```

Out[140…

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	t minu
0	KS	128	415	no	yes	25	265.1	110	45.07	1
1	OH	107	415	no	yes	26	161.6	123	27.47	1
2	NJ	137	415	no	no	0	243.4	114	41.38	1.
3	OH	84	408	yes	no	0	299.4	71	50.90	
4	OK	75	415	yes	no	0	166.7	113	28.34	1.

5 rows × 21 columns

Now, we can see that the `phone number` variable has been removed from the dataset and 'PhoneCode' has been added

Explore Categorical Variables

Now, I will explore the categorical variables one by one.

In [141…

```
# Identify categorical variables
categorical_vars = df.select_dtypes(include=['object', 'bool']).columns

# Print categorical variables
print('There are {} categorical variables\n'.format(len(categorical_vars)))

print('The categorical variables are :', categorical_vars)
```

```
There are 5 categorical variables

The categorical variables are : Index(['state', 'international plan', 'voice ma
il plan', 'churn', 'PhoneCode'], dtype='object')
```

We can see that there are 5 categorical variables in the dataset. The `phone number` variable has been removed. First, I will check missing values in categorical variables.

In [142…

```
# check for missing values in categorical variables

df[categorical_vars].isnull().sum()
```

Out[142…

```
state                 0
international plan    0
voice mail plan       0
churn                 0
PhoneCode             0
dtype: int64
```

Explore `state` variable

In [143…

```
# print number of labels in state variable

print('state contains', len(df["state"].unique()), 'labels')
```

```
state contains 51 labels
```

In [144…

```
# check labels in state variable

df.state.unique()
```

Out[144…

```
array(['KS', 'OH', 'NJ', 'OK', 'AL', 'MA', 'MO', 'LA', 'WV', 'IN', 'RI',
       'IA', 'MT', 'NY', 'ID', 'VT', 'VA', 'TX', 'FL', 'CO', 'AZ', 'SC',
       'NE', 'WY', 'HI', 'IL', 'NH', 'GA', 'AK', 'MD', 'AR', 'WI', 'OR',
       'MI', 'DE', 'UT', 'CA', 'MN', 'SD', 'NC', 'WA', 'NM', 'NV', 'DC',
       'KY', 'ME', 'MS', 'TN', 'PA', 'CT', 'ND'], dtype=object)
```

In [145…

```
# check frequency distribution of values in state variable

df["state"].value_counts()
```

Out[145…

```
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
VT     73
MI     73
ID     73
TX     72
UT     72
IN     71
```

```
MD      70
KS      70
MT      68
NC      68
NJ      68
CO      66
NV      66
WA      66
RI      65
MA      65
MS      65
AZ      64
MO      63
FL      63
ME      62
ND      62
NM      62
OK      61
DE      61
NE      61
SD      60
SC      60
KY      59
IL      58
NH      56
AR      55
GA      54
DC      54
HI      53
TN      53
AK      52
LA      51
PA      45
IA      44
CA      34
Name: state, dtype: int64
```

In [146…

```
# Let's do One Hot Encoding of state variable
# get k-1 dummy variables after One Hot Encoding
# preview the dataset with head() method

pd.get_dummies(df["state"], drop_first=True).head()
```

Out[146…

	AL	AR	AZ	CA	CO	CT	DC	DE	FL	GA	...	SD	TN	TX	UT	VA	VT	WA	\
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 50 columns

Explore international plan variable

In [147…

```
# print number of labels in international plan variable

print('international plan contains', len(df['international plan'].unique()), '
```

```
international plan contains 2 labels
```

In [148…

```
# check labels in international plan variable

df['international plan'].unique()
```

Out[148…

```
array(['no', 'yes'], dtype=object)
```

In [149…

```
# check frequency distribution of values in international plan variable

df['international plan'].value_counts()
```

Out[149…

```
no     3010
yes     323
Name: international plan, dtype: int64
```

In [150…

```
# let's do One Hot Encoding of international plan variable
# get k-1 dummy variables after One Hot Encoding
# preview the dataset with head() method

pd.get_dummies(df['international plan'], drop_first=True, dummy_na=True,dtype=
```

Out[150…

	yes	NaN
0	0	0
1	0	0
2	0	0
3	1	0
4	1	0

In [151…

```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df['international plan'], drop_first=True, dummy_na=True).sum(a
```

Out[151…

```
yes    323
NaN      0
dtype: int64
```

There are 323 yes values and no missing values in the `international plan` variable. The rest are no values

Explore voice mail plan variable

In [152…

```
# print number of labels in voice mail plan variable

print('voice mail plan contains', len(df['voice mail plan'].unique()), 'labels
```

```
voice mail plan contains 2 labels
```

In [153…

```
# check labels in voice mail plan variable

df['voice mail plan'].unique()
```

Out[153…

```
array(['yes', 'no'], dtype=object)
```

In [154…

```
# check frequency distribution of values in voice mail plan variable

df['voice mail plan'].value_counts()
```

Out[154…

```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

In [155…

```
# let's do One Hot Encoding of voice mail plan variable
# get k-1 dummy variables after One Hot Encoding
# preview the dataset with head() method

pd.get_dummies(df['voice mail plan'], drop_first=True, dummy_na=True,dtype='ir
```

Out[155…

	yes	NaN
0	1	0
1	1	0
2	0	0
3	0	0
4	0	0

In [156…

```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df['voice mail plan'], drop_first=True, dummy_na=True).sum(axis
```

Out[156…

```
yes    922
NaN      0
dtype: int64
```

There are 922 yes values and no missing values in the `voice mail plan` variable. The rest are no values

Explore PhoneCode variable

In [157…

```
# print number of labels in PhoneCode variable

print('PhoneCode contains', len(df['PhoneCode'].unique()), 'labels')
```

```
PhoneCode contains 96 labels
```

In [158…

```
# check labels in PhoneCode variable

df['PhoneCode'].unique()
```

Out[158…

```
array(['382', '371', '358', '375', '330', '391', '355', '329', '335',
       '344', '363', '394', '366', '351', '350', '386', '356', '373',
       '396', '393', '343', '331', '357', '418', '353', '410', '416',
       '370', '383', '360', '395', '362', '341', '402', '332', '372',
       '390', '352', '364', '398', '405', '413', '420', '349', '404',
       '340', '403', '359', '365', '338', '374', '415', '399', '333',
       '354', '419', '411', '388', '412', '346', '400', '334', '387',
       '327', '379', '347', '401', '397', '409', '337', '407', '328',
       '369', '408', '414', '345', '422', '381', '380', '336', '417',
       '406', '361', '377', '385', '378', '367', '339', '348', '342',
       '389', '368', '384', '376', '421', '392'], dtype=object)
```

In [159…

```
# check frequency distribution of values in PhoneCode variable

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

Out[159…

```
405    53
408    48
406    47
352    47
333    46
       ..
421    24
342    24
412    23
327    19
422    19
Name: PhoneCode, Length: 96, dtype: int64
```

In [160…

```
# Let's do One Hot Encoding of PhoneCode variable
# get k-1 dummy variables after One Hot Encoding
# preview the dataset with head() method

pd.get_dummies(df['PhoneCode'], drop_first=True, dummy_na=True,dtype='int').he
```

Out[160…

	328	329	330	331	332	333	334	335	336	337	...	414	415	416	417	418
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	(
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	(
2	0	0	0	0	0	0	0	0	0	0		0	0	0	0	(

2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	(
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	(
4	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	(

5 rows × 96 columns

In [161…

```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df['PhoneCode'], drop_first=True, dummy_na=True).sum(axis=0)
```

Out[161…

```
328    32
329    37
330    34
331    25
332    44
       ..
419    27
420    35
421    24
422    19
NaN     0
Length: 96, dtype: int64
```

There are no missing values

Explore Churn variable

In [162…

```
# print number of labels in Churn variable

print('Churn contains', len(df['churn'].unique()), 'labels')
```

```
Churn contains 2 labels
```

In [163…

```
# check labels in churn variable

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

Out[163…

```
array([False,  True])
```

In [164…

```
# check frequency distribution of values in churn variable

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

Out[164…

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

In [165…

```
# Let's do One Hot Encoding of churn variable
# get k-1 dummy variables after One Hot Encoding
```

```
# preview the dataset with head() method

pd.get_dummies(df['churn'], drop_first=True, dummy_na=True,dtype='int').head(
```

Out[165…

	True	NaN
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

In [166…

```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df['churn'], drop_first=True, dummy_na=True).sum(axis=0)
```

Out[166…

```
True     483
NaN        0
dtype: int64
```

There are 483 True values and no missing values in the `churn` variable. The rest are False values

Explore Numerical Variables

In [167…

```
# find numerical variables

numerical = [var for var in df.columns if df[var].dtype not in ['object', 'boo

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)
```

```
There are 16 numerical variables

The numerical variables are : ['account length', 'area code', 'number vmail mes
sages', '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']
```

In [168…

```
# view the numerical variables

df[numerical].head()
```

Out[168…

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes

0	128	415	25	265.1	110	45.07	197.4	99	16.78	244.7
1	107	415	26	161.6	123	27.47	195.5	103	16.62	254.4
2	137	415	0	243.4	114	41.38	121.2	110	10.30	162.6
3	84	408	0	299.4	71	50.90	61.9	88	5.26	196.9
4	75	415	0	166.7	113	28.34	148.3	122	12.61	186.9

Summary of numerical variables

- There are 16 numerical variables.
- These are given by 'account length', 'area code', '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' and 'customer service calls'
- All of the numerical variables are of continuous type.

Explore problems within numerical variables

Now, I will explore the numerical variables.

Missing values in numerical variables

In [169…

```
# check missing values in numerical variables

df[numerical].isnull().sum()
```

Out[169…

```
account length            0
area code                 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
dtype: int64
```

We can see that all the 16 numerical variables do not contain missing values.

Outliers in numerical variables

In [170…

```
# view summary statistics in numerical variables

print(round(df[numerical].describe()),2)
```

```
       account length  area code  number vmail messages  total day minutes  \
count          3333.0     3333.0                 3333.0             3333.0
mean            101.0      437.0                    8.0              180.0
std              40.0       42.0                   14.0               54.0
min               1.0      408.0                    0.0                0.0
25%              74.0      408.0                    0.0              144.0
50%             101.0      415.0                    0.0              179.0
75%             127.0      510.0                   20.0              216.0
max             243.0      510.0                   51.0              351.0

       total day calls  total day charge  total eve minutes  total eve calls  \
count           3333.0            3333.0             3333.0           3333.0
mean             100.0              31.0              201.0            100.0
std               20.0               9.0               51.0             20.0
min                0.0               0.0                0.0              0.0
25%               87.0              24.0              167.0             87.0
50%              101.0              30.0              201.0            100.0
75%              114.0              37.0              235.0            114.0
max              165.0              60.0              364.0            170.0

       total eve charge  total night minutes  total night calls  \
count            3333.0               3333.0             3333.0
mean               17.0                201.0              100.0
std                 4.0                 51.0               20.0
min                 0.0                 23.0               33.0
25%                14.0                167.0               87.0
50%                17.0                201.0              100.0
75%                20.0                235.0              113.0
max                31.0                395.0              175.0

       total night charge  total intl minutes  total intl calls  \
count              3333.0              3333.0            3333.0
mean                  9.0                10.0               4.0
std                   2.0                 3.0               2.0
min                   1.0                 0.0               0.0
25%                   8.0                 8.0               3.0
50%                   9.0                10.0               4.0
75%                  11.0                12.0               6.0
max                  18.0                20.0              20.0

       total intl charge  customer service calls
count             3333.0                  3333.0
mean                 3.0                     2.0
std                  1.0                     1.0
min                  0.0                     0.0
25%                  2.0                     1.0
50%                  3.0                     1.0
75%                  3.0                     2.0
max                  5.0                     9.0   2
```

On closer inspection, we can see that the `area code`, `number of vmail messages`, `total intl calls` and `customer service calls` columns may contain outliers.

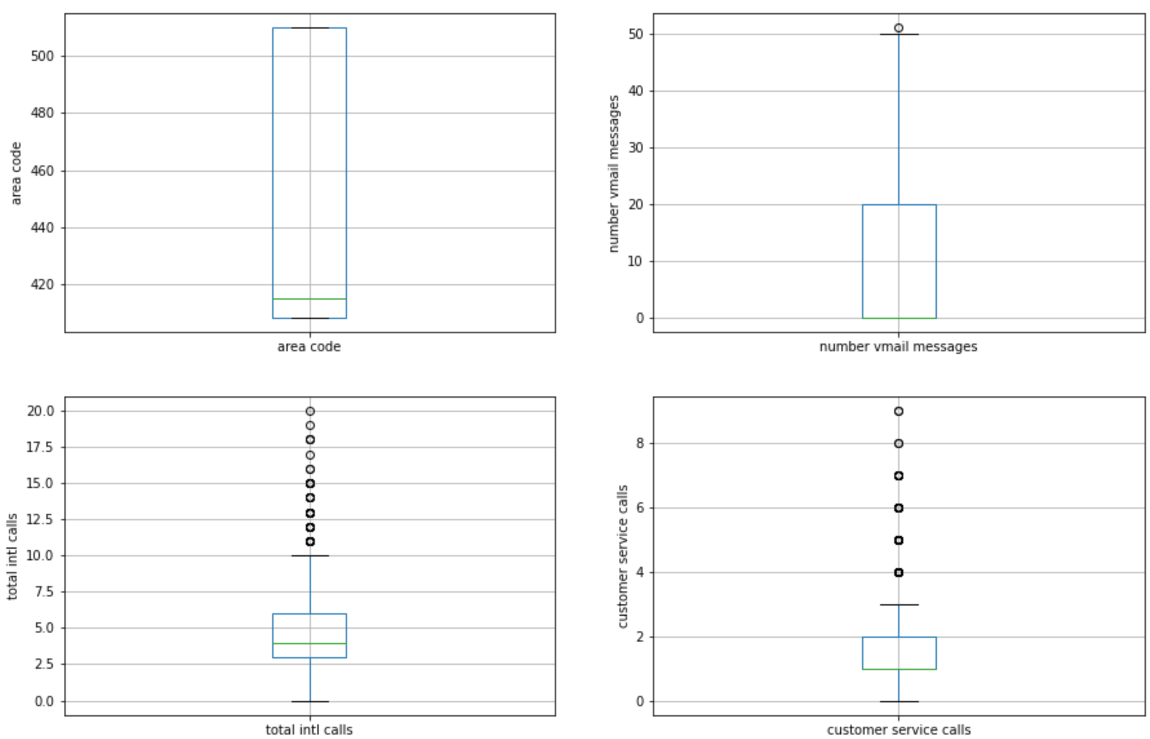
I will draw boxplots to visualise outliers in the above variables.

In [171…

```
# draw boxplots to visualize outliers

plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df.boxplot(column='area code')
fig.set_title('')
fig.set_ylabel('area code')


plt.subplot(2, 2, 2)
fig = df.boxplot(column='number vmail messages')
fig.set_title('')
fig.set_ylabel('number vmail messages')


plt.subplot(2, 2, 3)
fig = df.boxplot(column='total intl calls')
fig.set_title('')
fig.set_ylabel('total intl calls')


plt.subplot(2, 2, 4)
fig = df.boxplot(column='customer service calls')
fig.set_title('')
fig.set_ylabel('customer service calls')
```

Out[171…

```
Text(0, 0.5, 'customer service calls')
```



The above boxplots confirm that there are outliers in these variables except area code.

Check the distribution of variables

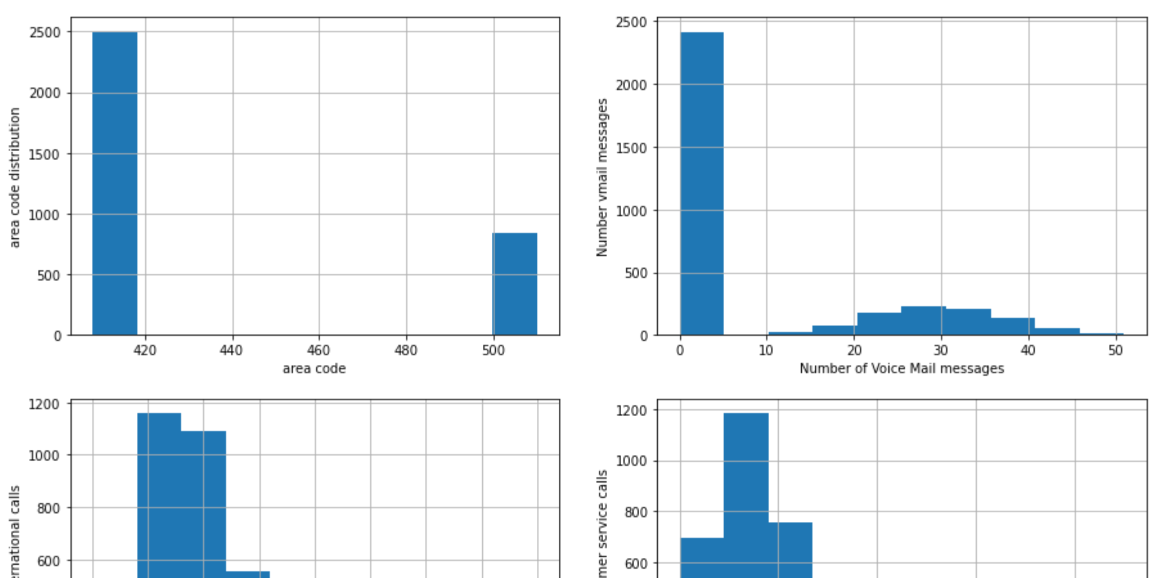
Now, I will plot the histograms to check distributions to find out if they are normal or skewed. If the variable follows normal distribution, then I will do `Extreme Value Analysis` otherwise if they are skewed, I will find IQR (Interquantile range).

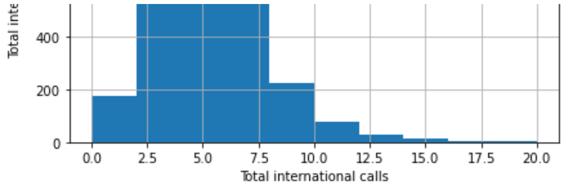
In [172...

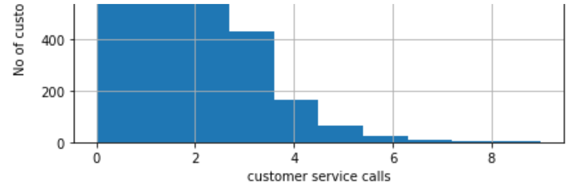
```
# plot histogram to check distribution

plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df["area code"].hist(bins=10)
fig.set_xlabel('area code')
fig.set_ylabel('area code distribution')


plt.subplot(2, 2, 2)
fig = df["number vmail messages"].hist(bins=10)
fig.set_xlabel('Number of Voice Mail messages')
fig.set_ylabel('Number vmail messages')


plt.subplot(2, 2, 3)
fig = df["total intl calls"].hist(bins=10)
fig.set_xlabel('Total international calls')
fig.set_ylabel('Total international calls')


plt.subplot(2, 2, 4)
fig = df["customer service calls"].hist(bins=10)
fig.set_xlabel('customer service calls')
fig.set_ylabel('No of customer service calls')
```

Out[172...

```
Text(0, 0.5, 'No of customer service calls')
```







We can see that all the four variables are skewed. So, I will use interquantile range to find outliers.

In [173…

```
# find outliers for area code variable

IQR = df["area code"].quantile(0.75) - df['area code'].quantile(0.25)
Lower_fence = df["area code"].quantile(0.25) - (IQR * 3)
Upper_fence = df["area code"].quantile(0.75) + (IQR * 3)
print('area code outliers are values < {lowerboundary} or > {upperboundary}'.f
```

```
area code outliers are values < 102.0 or > 816.0
```

For `area code` , the minimum and maximum values are 408 and 510 So, the outliers are values < 102.0 or > 816.0.

In [174…

```
# find outliers for number of voice mail messages variable

IQR = df["number vmail messages"].quantile(0.75) - df['number vmail messages']
Lower_fence = df["number vmail messages"].quantile(0.25) - (IQR * 3)
Upper_fence = df["number vmail messages"].quantile(0.75) + (IQR * 3)
print('Number of voice mail messages outliers are values < {lowerboundary} or
```

```
Number of voice mail messages outliers are values < -60.0 or > 80.0
```

For `voice mail messages` , the minimum and maximum values are 0 and 51 So, the outliers are values > 80.0.

In [175…

```
# find outliers for number of total International calls variable

IQR = df["total intl calls"].quantile(0.75) - df['total intl calls'].quantile(
Lower_fence = df["total intl calls"].quantile(0.25) - (IQR * 3)
Upper_fence = df["total intl calls"].quantile(0.75) + (IQR * 3)
print('Number of total international calls outliers are values < {lowerboundar
```

```
Number of total international calls outliers are values < -6.0 or > 15.0
```

For `total international calls` , the minimum and maximum values are 0 and 5 So, the outliers are values > 15.0.

In [176…

```
# find outliers for number of customer service calls variable

IQR = df["customer service calls"].quantile(0.75) - df['customer service calls
Lower_fence = df["customer service calls"].quantile(0.25) - (IQR * 3)
Upper_fence = df["customer service calls"].quantile(0.75) + (IQR * 3)
print('Number of customer service calls outliers are values < {lowerboundary}
```

```
Number of customer service calls outliers are values < -2.0 or > 5.0
```

```
Number of customer service calls outliers are values < -2.0 or > 5.0
```

For `customer service calls` , the minimum and maximum values are 0 and 9 and the mean is 2, So the outliers are values > 5.0

3. Declare feature vector and target variable

In [177…

```python
X = df.drop(['churn'], axis=1)

y = df['churn']
```

4. Split data into separate training and test set

In [178…

```python
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
```

In [179…

```python
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[179…

```
((2666, 20), (667, 20))
```

5. Feature Engineering

Feature Engineering is the process of transforming raw data into useful features that help us to understand our model better and increase its predictive power. I will carry out feature engineering on different types of variables.

First, I will display the categorical and numerical variables again separately.

In [180…

```python
# check data types in X_train

X_train.dtypes
```

Out[180…

```
state                      object
account length              int64
area code                   int64
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
```

```
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
PhoneCode                  object
dtype: object
```

In [181…

```
# display categorical variables

categorical_vars = X_train.select_dtypes(include=['object', 'bool']).columns.t
categorical_vars
```

Out[181…

```
['state', 'international plan', 'voice mail plan', 'PhoneCode']
```

In [182…

```
# display numerical variables

#numerical = [col for col in X_train.columns if X_train[col].dtypes != ['objec
numerical = X_train.select_dtypes(include=['number']).columns.tolist()
numerical
```

Out[182…

```
['account length',
 'area code',
 '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']
```

Engineering missing values in numerical variables

In [183…

```
# check missing values in numerical variables in X_train

X_test[numerical].isnull().sum()
```

Out[183…

```
account length           0
area code                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
dtype: int64
```

As expected, there is no missing values in the x_test data itself

Assumption

I assume that there is no data no missing x_test values. We therefore we dont need to impute the data to fill in for the missing values.

Engineering missing values in categorical variables

In [184…

```
# print percentage of missing values in the categorical variables in training

X_train[categorical_vars].isnull().mean()
```

Out[184…

```
state                 0.0
international plan    0.0
voice mail plan       0.0
PhoneCode             0.0
dtype: float64
```

Again we see there is no missing values in categorical data

As a final check, I will check for missing values in X_train and X_test.

In [185…

```
# check missing values in X_train

X_train.isnull().sum()
```

Out[185…

```
state                     0
account length            0
area code                 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 minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
PhoneCode                 0
dtype: int64
```

In [186…

```
# check missing values in X_test

X_test.isnull().sum()
```

Out[186…

```
state                     0
account length            0
area code                 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
PhoneCode                 0
dtype: int64
```

We can see that there are no missing values in X_train and X_test.

Engineering outliers in numerical variables

We have seen that the `area code`, `number vmail messages`, `total intl calls` and `customer service calls` columns contain outliers

In [187…

```
X_train.isnull().sum()
```

Out[187…

```
state                     0
account length            0
area code                 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 calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
PhoneCode                 0
dtype: int64
```

In [188…

```
upper_thresholds = {
    'area code': 510,
    'number vmail messages': 51,
    'total intl calls': 5,
    'customer service calls': 9
}

for df3 in [X_train, X_test]:
    for column, top in upper_thresholds.items():
        df3[column] = df3[column].clip(upper=top)
        if column in df.columns:  # Check if the column exists in the DataFran
            df[column] = df[column].clip(upper=top)
```

In [189…

```
#X_train.area code.max(), X_test.area code.max()
max_values_X_train = X_train[upper_thresholds.keys()].max()
max_values_X_test = X_test[upper_thresholds.keys()].max()

print("Max values in X_train after clipping:\n", max_values_X_train)
print("Max values in X_test after clipping:\n", max_values_X_test)
```

```
Max values in X_train after clipping:
 area code                 510
number vmail messages      51
total intl calls            5
customer service calls      9
dtype: int64
Max values in X_test after clipping:
 area code                 510
number vmail messages      50
total intl calls            5
customer service calls      8
dtype: int64
```

In [190…

```
X_train[numerical].describe()
```

Out[190…

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge
count	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000
mean	100.351463	437.351838	7.998500	179.960315	100.424231	30.593792
std	39.902158	42.488511	13.572182	54.233805	20.116856	9.219742
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000
25%	73.000000	408.000000	0.000000	144.650000	87.000000	24.590000

50%	100.000000	415.000000	0.000000	179.400000	100.000000	30.500000
75%	127.000000	510.000000	19.000000	216.000000	114.000000	36.720000
max	232.000000	510.000000	51.000000	350.800000	165.000000	59.640000

We can now see that the outliers in `Area code` , `number of vmail messages` , `total intl calls` and `customer service calls` columns are capped.

In [191…

```
X_train.isnull().sum()
```

Out[191…

```
state                     0
account length            0
area code                 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
PhoneCode                 0
dtype: int64
```

Encode categorical variables

In [192…

```
categorical_vars
```

Out[192…

```
['state', 'international plan', 'voice mail plan', 'PhoneCode']
```

In [193…

```
X_train[categorical_vars].head()
```

Out[193…

	state	international plan	voice mail plan	PhoneCode
1460	MT	no	no	361
2000	PA	no	no	334
666	OR	no	no	368
2962	SD	no	no	393
2773	NJ	no	yes	373

In [194…

```
# Initialize BinaryEncoder with a list of columns
binary_encoder = ce.BinaryEncoder(cols=['international plan', 'voice mail plan

# Fit and transform the training data
X_train_encoded = binary_encoder.fit_transform(X_train)

# Transform the test data
X_test_encoded = binary_encoder.transform(X_test)
```

In [195…

```
X_train.head()
```

Out[195…

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
1460	MT	80	415	no	no	0	198.1	160	33.68
2000	PA	28	415	no	no	0	168.2	87	28.59
666	OR	120	415	no	no	0	252.0	120	42.84
2962	SD	105	415	no	no	0	251.6	88	42.77
2773	NJ	134	510	no	yes	34	247.2	105	42.02

In [196…

```
X_train.isnull().sum()
```

Out[196…

```
state                     0
account length            0
area code                 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
PhoneCode                 0
dtype: int64
```

Now, I will create the `X_train` training set.

In [197…

```
# using ohe Recommended
```

```
ohe = ce.OneHotEncoder(cols=["state","international plan","voice mail plan","F

ohe.fit(X_train)

X_train = ohe.transform(X_train)
X_test = ohe.transform(X_test)

X_train.head()
```

Out[197...

	state_1	state_2	state_3	state_4	state_5	state_6	state_7	state_8	state_9	sta
1460	1	0	0	0	0	0	0	0	0	
2000	0	1	0	0	0	0	0	0	0	
666	0	0	1	0	0	0	0	0	0	
2962	0	0	0	1	0	0	0	0	0	
2773	0	0	0	0	1	0	0	0	0	

5 rows × 167 columns

Similarly, I will create the `X_test` testing set.

In [198...

```
X_test.head()
```

Out[198...

	state_1	state_2	state_3	state_4	state_5	state_6	state_7	state_8	state_9	sta
405	0	0	0	0	0	0	0	0	0	
118	0	0	0	0	0	0	0	0	0	
710	0	0	0	0	0	0	0	0	0	
499	0	0	0	0	0	0	1	0	0	
2594	0	0	0	0	0	0	0	0	0	

5 rows × 167 columns

We now have training and testing set ready for model building. Before that, we should map all the feature variables onto the same scale. It is called `feature scaling`. I will do it as follows.

6. Feature Scaling

In [199...

```
X_train.describe()
```

Out[199...

	state_1	state_2	state_3	state_4	state_5	state_6

count	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000
mean	0.022506	0.013878	0.023256	0.018005	0.019130	0.020255
std	0.148349	0.117009	0.150743	0.132992	0.137007	0.140898
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 167 columns

In [200…

```
scaler = MinMaxScaler()

scaler.fit(X_train)
```

Out[200…

```
MinMaxScaler()
```

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

In [201…

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

In [202…

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

In [203…

```
X_train.describe()
```

Out[203…

	state_1	state_2	state_3	state_4	state_5	state_6
count	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000
mean	0.022506	0.013878	0.023256	0.018005	0.019130	0.020255
std	0.148349	0.117009	0.150743	0.132992	0.137007	0.140898
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 167 columns

We now have `X_train` dataset ready to be fed into the Logistic Regression classifier. I will do it as follows.

7. Model training

In [204…

```
X_train.isnull().sum()
```

Out[204…

```
state_1         0
state_2         0
state_3         0
state_4         0
state_5         0
               ..
PhoneCode_92    0
PhoneCode_93    0
PhoneCode_94    0
PhoneCode_95    0
PhoneCode_96    0
Length: 167, dtype: int64
```

In [205…

```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

Out[205…

```
LogisticRegression(random_state=0, solver='liblinear')
```

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

8. Predict results

In [206…

```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

```
Out[206…  array([False, False, False, False,  True,  True, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                  True, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False,  True, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False,  True, False, False,
                 False, False, False, False, False,  True, False, False, False,
                 False, False, False, False, False,  True, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                  True, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False,  True, False, False, False, False,
                 False, False, False, False,  True, False, False, False, False,
                  True, False, False, False, False, False, False, False, False,
                  True, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False,  True, False, False, False, False,
                 False, False, False,  True, False, False,  True, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False,  True, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False,  True, False, False, False, False, False, False,
                  True, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False,  True,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False,  True,
                 False, False,  True, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False,  True, False,  True, False,  True,
                 False, False, False, False, False, False, False,  True, False,
                 False, False, False, False, False, False, False, False, False,
                 False,  True, False, False, False, False, False, False,  True,
                 False, False, False, False,  True, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False,  True, False, False, False, False,  True,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False,  True, False, False, False, False,  True, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
```

```
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False,  True, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False,  True, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
       False])
```

predict_proba method

predict_proba method gives the probabilities for the target variable(0 and 1) in this case, in array form.

`0 is for not churning and 1 is for churning.`

In [207…

```
# probability of getting output as 0 - no churn

logreg.predict_proba(X_test)[:,0]
```

Out[207…

```
array([0.69492933, 0.96464203, 0.79689587, 0.91123407, 0.1302019 ,
       0.31063709, 0.54002009, 0.90803103, 0.73837134, 0.6051113 ,
       0.97711015, 0.9368615 , 0.79001304, 0.9896553 , 0.79528087,
       0.74516469, 0.90179659, 0.8672895 , 0.9442934 , 0.97617466,
       0.9672907 , 0.97100048, 0.94979693, 0.98502318, 0.93290361,
       0.88821015, 0.90834174, 0.94028818, 0.7438388 , 0.89675339,
       0.92577367, 0.98832052, 0.6986641 , 0.98239899, 0.97209945,
       0.96203685, 0.17726411, 0.95716147, 0.97919816, 0.70039104,
       0.95141513, 0.88593511, 0.74652507, 0.65890416, 0.99056151,
       0.98586719, 0.94368955, 0.93932819, 0.91544105, 0.95136934,
       0.99335282, 0.87038429, 0.8948471 , 0.91035029, 0.9925271 ,
       0.9501804 , 0.96752679, 0.96847162, 0.92182769, 0.32479367,
       0.91383627, 0.94656881, 0.99288901, 0.93549649, 0.89074462,
       0.77114067, 0.88333082, 0.85463879, 0.96046527, 0.93685109,
       0.99448365, 0.98336429, 0.9720325 , 0.51676879, 0.72950824,
       0.60417168, 0.90774092, 0.99567581, 0.4160021 , 0.98221888,
       0.99281446, 0.64974317, 0.93031162, 0.93840404, 0.86232414,
       0.80415295, 0.41438828, 0.97781064, 0.63988017, 0.98586232,
       0.97119972, 0.62573077, 0.91126397, 0.96822888, 0.82091262,
       0.45875601, 0.85963584, 0.9479685 , 0.94065483, 0.96235832,
       0.96007672, 0.98318831, 0.94538657, 0.9834324 , 0.54430365,
       0.9658459 , 0.81199905, 0.70400113, 0.43624322, 0.87430616,
       0.78287493, 0.94404457, 0.93340967, 0.93833487, 0.77418224,
       0.71201401, 0.96103114, 0.78916036, 0.8769919 , 0.88846325,
       0.68875731, 0.94041605, 0.9623649 , 0.95132458, 0.83429559,
       0.81355316, 0.90984247, 0.98370124, 0.96669624, 0.82414027,
       0.97279063, 0.98731796, 0.8745183 , 0.95326188, 0.96604437,
       0.85743813, 0.93564468, 0.92246379, 0.95910902, 0.8636456 ,
```

```
0.96646803, 0.89540352, 0.72122178, 0.93145279, 0.95545246,
0.97901326, 0.98446074, 0.64782252, 0.97844614, 0.90369919,
0.9266015 , 0.91035936, 0.98730827, 0.9234314 , 0.93904337,
0.90440004, 0.96039332, 0.94632357, 0.53467049, 0.87989617,
0.95602875, 0.96598725, 0.85895513, 0.96333609, 0.89094313,
0.94729137, 0.20853944, 0.97399667, 0.72810392, 0.92419922,
0.90746627, 0.88590289, 0.88540643, 0.98767095, 0.95235147,
0.38749025, 0.52387929, 0.56998142, 0.71794239, 0.68740936,
0.39267234, 0.65550157, 0.97316721, 0.98244736, 0.95851578,
0.82452807, 0.96776557, 0.93596221, 0.9463307 , 0.26447468,
0.91810129, 0.99070046, 0.83623021, 0.98337998, 0.95098129,
0.96580116, 0.87565847, 0.97428591, 0.96776568, 0.9334246 ,
0.91025697, 0.75359268, 0.92896959, 0.96071386, 0.98100247,
0.92078595, 0.98652748, 0.90988854, 0.96798237, 0.58990793,
0.92975501, 0.92586413, 0.74951243, 0.99249172, 0.97025516,
0.85562105, 0.97306973, 0.90480524, 0.86823018, 0.96175579,
0.97100399, 0.87539993, 0.85413619, 0.98077095, 0.88900472,
0.94125471, 0.93304794, 0.91320314, 0.87374921, 0.95381335,
0.91933675, 0.97983172, 0.87259625, 0.96520276, 0.9421623 ,
0.74117142, 0.8216904 , 0.9949293 , 0.24935808, 0.90158018,
0.96029395, 0.97217852, 0.97401874, 0.97678731, 0.75747712,
0.83316569, 0.47623831, 0.98018325, 0.80322141, 0.39373367,
0.98120924, 0.89987179, 0.77132735, 0.82809056, 0.97178164,
0.97603486, 0.76718788, 0.76413403, 0.87549527, 0.93118245,
0.94624643, 0.97730354, 0.90232095, 0.8739209 , 0.98292219,
0.95125423, 0.98368206, 0.89631305, 0.89319824, 0.92384975,
0.98619677, 0.97920454, 0.72298151, 0.8843336 , 0.45902143,
0.98435825, 0.97010482, 0.93405336, 0.97745251, 0.97695593,
0.98157342, 0.69140698, 0.87178833, 0.8592899 , 0.99918425,
0.91921865, 0.60862825, 0.97377034, 0.77478411, 0.88572589,
0.93385003, 0.90542617, 0.91325983, 0.88351484, 0.79773652,
0.72733517, 0.8706448 , 0.68379789, 0.96143686, 0.92464325,
0.85126749, 0.57301062, 0.98575284, 0.90328356, 0.96961587,
0.86699416, 0.63432678, 0.89598403, 0.40287238, 0.62040314,
0.94803135, 0.96974434, 0.93623861, 0.72615979, 0.93143978,
0.24192273, 0.85762406, 0.9901284 , 0.98343012, 0.99641893,
0.92060412, 0.89276889, 0.92685695, 0.955538  , 0.73527423,
0.74610205, 0.97685023, 0.94631508, 0.64281374, 0.91109533,
0.96374085, 0.97948201, 0.48488007, 0.80787172, 0.86087777,
0.9949936 , 0.79429317, 0.99049806, 0.98101196, 0.73550822,
0.90316296, 0.92090489, 0.82232984, 0.60204883, 0.85510033,
0.96608737, 0.66105692, 0.99412735, 0.91712202, 0.91226263,
0.34614601, 0.97180441, 0.54873704, 0.16953376, 0.73502962,
0.94939744, 0.97893706, 0.72593458, 0.8020159 , 0.93006719,
0.93383805, 0.8325994 , 0.94205332, 0.80385922, 0.81614855,
0.90966663, 0.77134991, 0.84995655, 0.97254352, 0.62318372,
0.74611422, 0.94239783, 0.60894102, 0.71035725, 0.97280519,
0.91909883, 0.84812229, 0.93271263, 0.97956265, 0.82614054,
0.90012492, 0.55339159, 0.41001253, 0.96124995, 0.43350931,
0.98891763, 0.41316172, 0.95861652, 0.79225547, 0.98311653,
0.6401309 , 0.60890124, 0.65815359, 0.903343  , 0.39644667,
0.7545589 , 0.81887905, 0.81139083, 0.96360134, 0.96978671,
0.76085458, 0.79765179, 0.99392964, 0.97849921, 0.88104291,
0.95373295, 0.28734807, 0.87558862, 0.9370072 , 0.94773771,
0.69452441, 0.97951335, 0.99622368, 0.48596582, 0.6912109 ,
0.94633348, 0.92590498, 0.94197065, 0.49279538, 0.87363941,
0.98913308, 0.8065391 , 0.98183493, 0.79931169, 0.9503064 ,
0.99418752, 0.96899648, 0.97027745, 0.97326168, 0.87471965,
0.96596026, 0.94107061, 0.97972739, 0.97194215, 0.81866132,
```

```
       0.54449028, 0.92779405, 0.9963247 , 0.53045332, 0.93681807,
       0.97960562, 0.87408093, 0.82688013, 0.84290345, 0.48936363,
       0.96497822, 0.61029096, 0.98938914, 0.984698  , 0.17284554,
       0.78427504, 0.96460304, 0.99066781, 0.85910759, 0.75489976,
       0.62450668, 0.98821523, 0.90064957, 0.55561346, 0.95153787,
       0.95144676, 0.94405765, 0.88569955, 0.67024646, 0.96283716,
       0.90077231, 0.9757413 , 0.60634211, 0.95300789, 0.86089413,
       0.92505942, 0.96616939, 0.9841335 , 0.80338418, 0.96988481,
       0.93817448, 0.93854127, 0.52696124, 0.96003691, 0.97558987,
       0.87129382, 0.98626041, 0.93444729, 0.97945041, 0.97811956,
       0.89569868, 0.96052389, 0.98703556, 0.66776245, 0.97797253,
       0.83679744, 0.66505552, 0.80745887, 0.94373889, 0.98331869,
       0.9469281 , 0.96064139, 0.45085486, 0.71162943, 0.82520125,
       0.9485995 , 0.80141824, 0.47737288, 0.80515898, 0.97350478,
       0.93768495, 0.89928015, 0.71062848, 0.74120648, 0.96045375,
       0.92615004, 0.96751013, 0.90906321, 0.92725789, 0.85208138,
       0.93518513, 0.88133952, 0.9879344 , 0.69989904, 0.90530128,
       0.90715134, 0.9669458 , 0.92315919, 0.87005357, 0.81366985,
       0.90493449, 0.97777364, 0.9878933 , 0.50040443, 0.95917095,
       0.93151684, 0.97711099, 0.97093456, 0.98271202, 0.92872109,
       0.983988  , 0.96120593, 0.9375346 , 0.8674268 , 0.76966067,
       0.97957632, 0.81169967, 0.9573034 , 0.98106664, 0.93567516,
       0.9888897 , 0.90634276, 0.74364869, 0.9668205 , 0.48509488,
       0.9392219 , 0.99256772, 0.54164985, 0.84137209, 0.96447219,
       0.98713471, 0.96001998, 0.90962388, 0.88019782, 0.79170606,
       0.82934098, 0.83381791, 0.9688151 , 0.96942115, 0.99236821,
       0.92798065, 0.95773873, 0.4861816 , 0.98739975, 0.81286701,
       0.90111735, 0.69397921, 0.83507166, 0.92459636, 0.84346341,
       0.80747955, 0.90087462, 0.92162249, 0.43062719, 0.97890817,
       0.37427379, 0.9209493 , 0.89764613, 0.87271337, 0.94119784,
       0.81420913, 0.90429867, 0.8593039 , 0.55681284, 0.97737883,
       0.93317889, 0.93572727, 0.93047224, 0.96820752, 0.95469506,
       0.97985938, 0.83906421, 0.91525574, 0.94068683, 0.59247126,
       0.93245423, 0.37547224, 0.97323587, 0.60604669, 0.96622959,
       0.9306123 , 0.36881749, 0.63849758, 0.63189207, 0.84719097,
       0.97705074, 0.13426548, 0.80476968, 0.96014058, 0.63637529,
       0.93612239, 0.96345641, 0.97160394, 0.65149948, 0.90491235,
       0.8108944 , 0.86621769, 0.91197351, 0.97870541, 0.97399397,
       0.89325238, 0.32659806, 0.98836212, 0.7043465 , 0.85962176,
       0.96892164, 0.87672475, 0.98357908, 0.58365414, 0.97680073,
       0.83698305, 0.71789753, 0.92688418, 0.86632457, 0.93720156,
       0.77835916, 0.96298003, 0.84100876, 0.90701871, 0.93910085,
       0.5866718 , 0.96879443, 0.72183038, 0.75742712, 0.97982609,
       0.92901127, 0.93872974, 0.86301786, 0.77516721, 0.96606869,
       0.67366345, 0.85447306, 0.62098592, 0.98709866, 0.96684628,
       0.94585018, 0.97952329, 0.98313476, 0.72134194, 0.89520703,
       0.49241811, 0.81750379])
```

In [208…

```
# probability of getting output as 1 - churn

logreg.predict_proba(X_test)[:,1]
```

Out[208…

```
array([3.05070673e-01, 3.53579652e-02, 2.03104134e-01, 8.87659281e-02,
       8.69798101e-01, 6.89362915e-01, 4.59979910e-01, 9.19689723e-02,
       2.61628660e-01, 3.94888704e-01, 2.28898477e-02, 6.31385027e-02,
       2.09986961e-01, 1.03446958e-02, 2.04719130e-01, 2.54835313e-01,
       9.82034103e-02, 1.32710496e-01, 5.57066029e-02, 2.38253438e-02,
       3.27093023e-02, 2.89995241e-02, 5.02030731e-02, 1.49768177e-02,
```

```
6.70963916e-02, 1.11789849e-01, 9.16582554e-02, 5.97118182e-02,
2.56161196e-01, 1.03246610e-01, 7.42263257e-02, 1.16794819e-02,
3.01335895e-01, 1.76010115e-02, 2.79005501e-02, 3.79631503e-02,
8.22735894e-01, 4.28385280e-02, 2.08018403e-02, 2.99608960e-01,
4.85848735e-02, 1.14064890e-01, 2.53474930e-01, 3.41095835e-01,
9.43848567e-03, 1.41328079e-02, 5.63104512e-02, 6.06718075e-02,
8.45589485e-02, 4.86306583e-02, 6.64718419e-03, 1.29615713e-01,
1.05152904e-01, 8.96497104e-02, 7.47289811e-03, 4.98195962e-02,
3.24732088e-02, 3.15283848e-02, 7.81723067e-02, 6.75206331e-01,
8.61637323e-02, 5.34311937e-02, 7.11099332e-03, 6.45035062e-02,
1.09255384e-01, 2.28859326e-01, 1.16669178e-01, 1.45361211e-01,
3.95347292e-02, 6.31489126e-02, 5.51634981e-03, 1.66357132e-02,
2.79674970e-02, 4.83231214e-01, 2.70491764e-01, 3.95828325e-01,
9.22590761e-02, 4.32418918e-03, 5.83997900e-01, 1.77811163e-02,
7.18554366e-03, 3.50256830e-01, 6.96883849e-02, 6.15959627e-02,
1.37675865e-01, 1.95847050e-01, 5.85611719e-01, 2.21893623e-02,
3.60119828e-01, 1.41376836e-02, 2.88002788e-02, 3.74269230e-01,
8.87360324e-02, 3.17711237e-02, 1.79087376e-01, 5.41243985e-01,
1.40364163e-01, 5.20314951e-02, 5.93451676e-02, 3.76416788e-02,
3.99232833e-02, 1.68116877e-02, 5.46134279e-02, 1.65676004e-02,
4.55696347e-01, 3.41540955e-02, 1.88000948e-01, 2.95998868e-01,
5.63756785e-01, 1.25693843e-01, 2.17125071e-01, 5.59554300e-02,
6.65903259e-02, 6.16651307e-02, 2.25817760e-01, 2.87985986e-01,
3.89688579e-02, 2.10839639e-01, 1.23008097e-01, 1.11536745e-01,
3.11242692e-01, 5.95839500e-02, 3.76350965e-02, 4.86754211e-02,
1.65704405e-01, 1.86446835e-01, 9.01575277e-02, 1.62987603e-02,
3.33037631e-02, 1.75859735e-01, 2.72093689e-02, 1.26820397e-02,
1.25481700e-01, 4.67381186e-02, 3.39556273e-02, 1.42561866e-01,
6.43553244e-02, 7.75362088e-02, 4.08909783e-02, 1.36354400e-01,
3.35319679e-02, 1.04596480e-01, 2.78778219e-01, 6.85472092e-02,
4.45475384e-02, 2.09867356e-02, 1.55392575e-02, 3.52177484e-01,
2.15538586e-02, 9.63008133e-02, 7.33985032e-02, 8.96406435e-02,
1.26917340e-02, 7.65686015e-02, 6.09566271e-02, 9.55999640e-02,
3.96066844e-02, 5.36764304e-02, 4.65329507e-01, 1.20103831e-01,
4.39712491e-02, 3.40127542e-02, 1.41044868e-01, 3.66639132e-02,
1.09056869e-01, 5.27086349e-02, 7.91460559e-01, 2.60033320e-02,
2.71896084e-01, 7.58007779e-02, 9.25337273e-02, 1.14097113e-01,
1.14593567e-01, 1.23290508e-02, 4.76485335e-02, 6.12509751e-01,
4.76120709e-01, 4.30018579e-01, 2.82057609e-01, 3.12590642e-01,
6.07327657e-01, 3.44498434e-01, 2.68327889e-02, 1.75526404e-02,
4.14842228e-02, 1.75471930e-01, 3.22344306e-02, 6.40377949e-02,
5.36692959e-02, 7.35525325e-01, 8.18987104e-02, 9.29954492e-03,
1.63769793e-01, 1.66200214e-02, 4.90187139e-02, 3.41988450e-02,
1.24341534e-01, 2.57140890e-02, 3.22343218e-02, 6.65754002e-02,
8.97430261e-02, 2.46407323e-01, 7.10304076e-02, 3.92861367e-02,
1.89975267e-02, 7.92140520e-02, 1.34725213e-02, 9.01114632e-02,
3.20176275e-02, 4.10092067e-01, 7.02449878e-02, 7.41358727e-02,
2.50487572e-01, 7.50827603e-03, 2.97448440e-02, 1.44378949e-01,
2.69302673e-02, 9.51947632e-02, 1.31769821e-01, 3.82442074e-02,
2.89960124e-02, 1.24600072e-01, 1.45863812e-01, 1.92290481e-02,
1.10995276e-01, 5.87452903e-02, 6.69520579e-02, 8.67968641e-02,
1.26250789e-01, 4.61866471e-02, 8.06632506e-02, 2.01682843e-02,
1.27403755e-01, 3.47972373e-02, 5.78377037e-02, 2.58828577e-01,
1.78309601e-01, 5.07070144e-03, 7.50641922e-01, 9.84198184e-02,
3.97060461e-02, 2.78214759e-02, 2.59812605e-02, 2.32126916e-02,
2.42522881e-01, 1.66834310e-01, 5.23761690e-01, 1.98167549e-02,
1.96778589e-01, 6.06266325e-01, 1.87907637e-02, 1.00128213e-01,
2.28672648e-01, 1.71909440e-01, 2.82183584e-02, 2.39651448e-02,
2.32812125e-01, 2.35865969e-01, 1.24504728e-01, 6.88175494e-02,
```

```
5.37535724e-02, 2.26964586e-02, 9.76790458e-02, 1.26079104e-01,
1.70778116e-02, 4.87457747e-02, 1.63179398e-02, 1.03686948e-01,
1.06801764e-01, 7.61502480e-02, 1.38032274e-02, 2.07954636e-02,
2.77018488e-01, 1.15666396e-01, 5.40978569e-01, 1.56417454e-02,
2.98951830e-02, 6.59466382e-02, 2.25474897e-02, 2.30440718e-02,
1.84265840e-02, 3.08593023e-01, 1.28211668e-01, 1.40710099e-01,
8.15745189e-04, 8.07813530e-02, 3.91371754e-01, 2.62296611e-02,
2.25215894e-01, 1.14274111e-01, 6.61499740e-02, 9.45738323e-02,
8.67401716e-02, 1.16485155e-01, 2.02263480e-01, 2.72664829e-01,
1.29355195e-01, 3.16202107e-01, 3.85631363e-02, 7.53567500e-02,
1.48732505e-01, 4.26989375e-01, 1.42471585e-02, 9.67164396e-02,
3.03841303e-02, 1.33005840e-01, 3.65673223e-01, 1.04015975e-01,
5.97127617e-01, 3.79596862e-01, 5.19686484e-02, 3.02556585e-02,
6.37613906e-02, 2.73840206e-01, 6.85602233e-02, 7.58077265e-01,
1.42375937e-01, 9.87160200e-03, 1.65698815e-02, 3.58107014e-03,
7.93958754e-02, 1.07231115e-01, 7.31430525e-02, 4.44620009e-02,
2.64725771e-01, 2.53897948e-01, 2.31497686e-02, 5.36849246e-02,
3.57186263e-01, 8.89046734e-02, 3.62591543e-02, 2.05179869e-02,
5.15119925e-01, 1.92128275e-01, 1.39122226e-01, 5.00639646e-03,
2.05706834e-01, 9.50194386e-03, 1.89880389e-02, 2.64491779e-01,
9.68370373e-02, 7.90951125e-02, 1.77670159e-01, 3.97951166e-01,
1.44899672e-01, 3.39126256e-02, 3.38943084e-01, 5.87264710e-03,
8.28779826e-02, 8.77373707e-02, 6.53853991e-01, 2.81955859e-02,
4.51262959e-01, 8.30466236e-01, 2.64970381e-01, 5.06025568e-02,
2.10629444e-02, 2.74065423e-01, 1.97984101e-01, 6.99328143e-02,
6.61619465e-02, 1.67400598e-01, 5.79466845e-02, 1.96140779e-01,
1.83851453e-01, 9.03333691e-02, 2.28650090e-01, 1.50043454e-01,
2.74564775e-02, 3.76816275e-01, 2.53885783e-01, 5.76021674e-02,
3.91058977e-01, 2.89642752e-01, 2.71948080e-02, 8.09011658e-02,
1.51877714e-01, 6.72873655e-02, 2.04373468e-02, 1.73859457e-01,
9.98750822e-02, 4.46608413e-01, 5.89987472e-01, 3.87500530e-02,
5.66490692e-01, 1.10823702e-02, 5.86838281e-01, 4.13834751e-02,
2.07744527e-01, 1.68834651e-02, 3.59869098e-01, 3.91098758e-01,
3.41846410e-01, 9.66569985e-02, 6.03553326e-01, 2.45441097e-01,
1.81120948e-01, 1.88609167e-01, 3.63986582e-02, 3.02132879e-02,
2.39145420e-01, 2.02348207e-01, 6.07035886e-03, 2.15007939e-02,
1.18957086e-01, 4.62670504e-02, 7.12651934e-01, 1.24411383e-01,
6.29927987e-02, 5.22622928e-02, 3.05475593e-01, 2.04866543e-02,
3.77631959e-03, 5.14034181e-01, 3.08789097e-01, 5.36665214e-02,
7.40950170e-02, 5.80293457e-02, 5.07204619e-01, 1.26360589e-01,
1.08669232e-02, 1.93460898e-01, 1.81650691e-02, 2.00688305e-01,
4.96936014e-02, 5.81247926e-03, 3.10035151e-02, 2.97225504e-02,
2.67383222e-02, 1.25280346e-01, 3.40397391e-02, 5.89293898e-02,
2.02726116e-02, 2.80578480e-02, 1.81338679e-01, 4.55509718e-01,
7.22059493e-02, 3.67529822e-03, 4.69546682e-01, 6.31819274e-02,
2.03943781e-02, 1.25919074e-01, 1.73119870e-01, 1.57096551e-01,
5.10636365e-01, 3.50217821e-02, 3.89709038e-01, 1.06108629e-02,
1.53019996e-02, 8.27154457e-01, 2.15724956e-01, 3.53969630e-02,
9.33219492e-03, 1.40892405e-01, 2.45100242e-01, 3.75493315e-01,
1.17847678e-02, 9.93504332e-02, 4.44386541e-01, 4.84621294e-02,
4.85532411e-02, 5.59423492e-02, 1.14300453e-01, 3.29753540e-01,
3.71628379e-02, 9.92276886e-02, 2.42586958e-02, 3.93657892e-01,
4.69921063e-02, 1.39105871e-01, 7.49405829e-02, 3.38306083e-02,
1.58664989e-02, 1.96615818e-01, 3.01151863e-02, 6.18255241e-02,
6.14587339e-02, 4.73038763e-01, 3.99630878e-02, 2.44101258e-02,
1.28706181e-01, 1.37395861e-02, 6.55527064e-02, 2.05495878e-02,
2.18804449e-02, 1.04301317e-01, 3.94761050e-02, 1.29644387e-02,
3.32237551e-01, 2.20274711e-02, 1.63202559e-01, 3.34944479e-01,
1.92541131e-01, 5.62611117e-02, 1.66813130e-02, 5.30718969e-02,
```

```
       3.93586111e-02, 5.49145138e-01, 2.88370572e-01, 1.74798748e-01,
       5.14005027e-02, 1.98581761e-01, 5.22627119e-01, 1.94841015e-01,
       2.64952189e-02, 6.23150472e-02, 1.00719847e-01, 2.89371516e-01,
       2.58793519e-01, 3.95462507e-02, 7.38499570e-02, 3.24898689e-02,
       9.09367865e-02, 7.27421098e-02, 1.47918620e-01, 6.48148694e-02,
       1.18660484e-01, 1.20655976e-02, 3.00100959e-01, 9.46987210e-02,
       9.28486577e-02, 3.30542005e-02, 7.68408124e-02, 1.29946428e-01,
       1.86330147e-01, 9.50655070e-02, 2.22263621e-02, 1.21067049e-02,
       4.99595567e-01, 4.08290453e-02, 6.84831585e-02, 2.28890148e-02,
       2.90654380e-02, 1.72879796e-02, 7.12789086e-02, 1.60119976e-02,
       3.87940734e-02, 6.24653986e-02, 1.32573201e-01, 2.30339329e-01,
       2.04236753e-02, 1.88300332e-01, 4.26965989e-02, 1.89333641e-02,
       6.43248390e-02, 1.11102999e-02, 9.36572439e-02, 2.56351313e-01,
       3.31794986e-02, 5.14905118e-01, 6.07780964e-02, 7.43227502e-03,
       4.58350153e-01, 1.58627911e-01, 3.55278146e-02, 1.28652885e-02,
       3.99800221e-02, 9.03761236e-02, 1.19802184e-01, 2.08293937e-01,
       1.70659016e-01, 1.66182093e-01, 3.11848954e-02, 3.05788542e-02,
       7.63179334e-03, 7.20193486e-02, 4.22612744e-02, 5.13818403e-01,
       1.26002507e-02, 1.87132992e-01, 9.88826493e-02, 3.06020793e-01,
       1.64928340e-01, 7.54036361e-02, 1.56536587e-01, 1.92520450e-01,
       9.91253765e-02, 7.83775133e-02, 5.69372805e-01, 2.10918316e-02,
       6.25726213e-01, 7.90507023e-02, 1.02353875e-01, 1.27286628e-01,
       5.88021557e-02, 1.85790870e-01, 9.57013270e-02, 1.40696099e-01,
       4.43187158e-01, 2.26211748e-02, 6.68211145e-02, 6.42727310e-02,
       6.95277631e-02, 3.17924788e-02, 4.53049365e-02, 2.01406247e-02,
       1.60935791e-01, 8.47442623e-02, 5.93131685e-02, 4.07528737e-01,
       6.75457700e-02, 6.24527757e-01, 2.67641280e-02, 3.93953308e-01,
       3.37704062e-02, 6.93876982e-02, 6.31182506e-01, 3.61502420e-01,
       3.68107933e-01, 1.52809029e-01, 2.29492631e-02, 8.65734518e-01,
       1.95230317e-01, 3.98594236e-02, 3.63624713e-01, 6.38776073e-02,
       3.65435856e-02, 2.83960629e-02, 3.48500517e-01, 9.50876528e-02,
       1.89105597e-01, 1.33782309e-01, 8.80264873e-02, 2.12945929e-02,
       2.60060299e-02, 1.06747622e-01, 6.73401942e-01, 1.16378841e-02,
       2.95653502e-01, 1.40378243e-01, 3.10783604e-02, 1.23275245e-01,
       1.64209229e-02, 4.16345864e-01, 2.31992653e-02, 1.63016949e-01,
       2.82102469e-01, 7.31158193e-02, 1.33675432e-01, 6.27984439e-02,
       2.21640839e-01, 3.70199746e-02, 1.58991242e-01, 9.29812885e-02,
       6.08991500e-02, 4.13328197e-01, 3.12055698e-02, 2.78169619e-01,
       2.42572877e-01, 2.01739075e-02, 7.09887305e-02, 6.12702627e-02,
       1.36982139e-01, 2.24832792e-01, 3.39313127e-02, 3.26336547e-01,
       1.45526936e-01, 3.79014082e-01, 1.29013358e-02, 3.31537174e-02,
       5.41498196e-02, 2.04767112e-02, 1.68652395e-02, 2.78658063e-01,
       1.04792965e-01, 5.07581891e-01, 1.82496207e-01])
```

9. Check accuracy score

In [209…

```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_t
```

```
Model accuracy score: 0.8696
```

Here, **y_test** are the true class labels and **y_pred_test** are the predicted class labels in the test-set.

Compare the train-set and test-set accuracy

Now, I will compare the train-set and test-set accuracy to check for overfitting.

In [210...

```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

Out[210...

```
array([False, False, False, ...,  True, False,  True])
```

In [211...

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
```

```
Training-set accuracy score: 0.8811
```

Check for overfitting and underfitting

In [212...

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.8811
Test set score: 0.8696
```

The training-set accuracy score is 0.8811 while the test-set accuracy to be 0.8696. These two values are quite comparable. So, there is no question of overfitting.

In Logistic Regression, we use default value of C = 1. It provides good performance with approximately more than 85% accuracy on both the training and the test set. The model performance on both the training and test set are very comparable. It is likely the case of underfitting.

I will increase C and fit a more flexible model.

In [213...

```
# fit the Logsitic Regression model with C=100

# instantiate the model
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)


# fit the model
logreg100.fit(X_train, y_train)
```

Out[213...

```
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

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

In [214...

```
# print the scores on training and test set
```

```
print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.8837
Test set score: 0.8636
```

We can see that, C=100 results in higher test set accuracy and also a slightly increased training set accuracy. So, we can conclude that a more complex model should perform better.

Now, I will investigate, what happens if we use more regularized model than the default value of C=1, by setting C=0.01.

In [215…

```
# fit the Logsitic Regression model with C=001

# instantiate the model
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)


# fit the model
logreg001.fit(X_train, y_train)
```

Out[215…

```
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

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

In [216…

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg001.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg001.score(X_test, y_test)))
```

```
Training set score: 0.8518
Test set score: 0.8681
```

So, if we use more regularized model by setting C=0.01, then both the training and test set accuracy decrease relative to the default parameters.

Compare model accuracy with null accuracy

So, the model accuracy is 0.8696. But, we cannot say that our model is very good based on the above accuracy. We must compare it with the **null accuracy**. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

So, we should first check the class distribution in the test set.

In [217…

```
# check class distribution in test set

y_test.value_counts()
```

Out[217…

```
False    579
True      88
Name: churn, dtype: int64
```

We can see that the occurences of most frequent class is 579. So, we can calculate null accuracy by dividing 579 by total number of occurences.

In [218…

```
# check null accuracy score

null_accuracy = (579/(579+88))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.8681
```

We can see that our model accuracy score is 0.8696 but null accuracy score is 0.8681. So, we can conclude that our Logistic Regression model needs to be improved for it to do a better job in predicting the class labels.

Now, based on the above analysis we can conclude that our classification model accuracy is good. Our model can be investigated further and further iteration done to improve its performance in terms of predicting the class labels.

It does not give the underlying distribution of values. Also, it does not tell anything about the type of errors our classifer is making.

We have another tool called `Confusion matrix` that comes to our rescue.

10. Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

These four outcomes are summarized in a confusion matrix given below.

In [219…

```
# Print the Confusion Matrix and slice it into four pieces

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[559  20]
 [ 67  21]]

True Positives(TP) =  559

True Negatives(TN) =  21

False Positives(FP) =  20

False Negatives(FN) =  67
```

The confusion matrix shows `559 + 21 = 580 correct predictions` and `20 + 67 = 87 incorrect predictions`.

In this case, we have

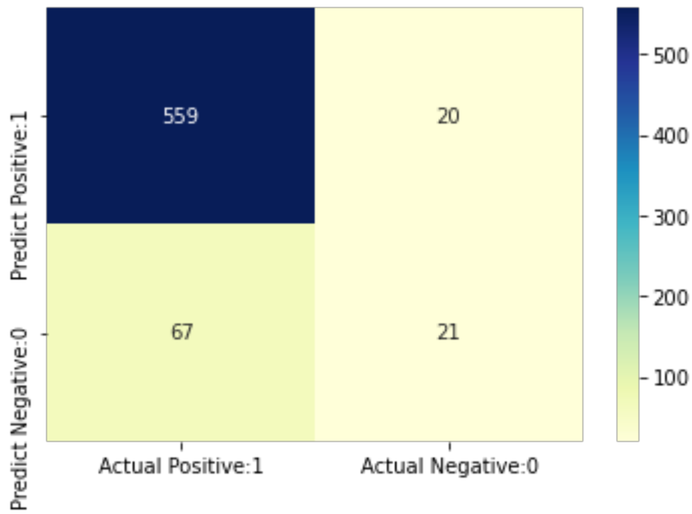
- `True Positives` (Actual Positive:1 and Predict Positive:1) - 559
- `True Negatives` (Actual Negative:0 and Predict Negative:0) - 21
- `False Positives` (Actual Negative:0 but Predict Positive:1) - 20 `(Type I error)`
- `False Negatives` (Actual Positive:1 but Predict Negative:0) - 67 `(Type II error)`

In [220…

```
# visualize confusion matrix with seaborn heatmap
```

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negati
                                  index=['Predict Positive:1', 'Predict Negativ

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[220… <AxesSubplot:>



11. Classification metrices

Classification Report

We can also use **Classification report** to evaluate the classification model performance. It displays the **precision**, **recall**, **f1** and **support** scores for the model. We can print a classification report as follows:-

In [221…

```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

       False       0.89      0.97      0.93       579
        True       0.51      0.24      0.33        88

    accuracy                           0.87       667
   macro avg       0.70      0.60      0.63       667
weighted avg       0.84      0.87      0.85       667
```

Classification accuracy

In [222…

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [223…

```
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.8696
```

Classification error

In [224…

```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.1304
```

Precision

Precision can be defined as the percentage of correctly predicted positive outcomes out of all the predicted positive outcomes. It can be given as the ratio of true positives (TP) to the sum of true and false positives (TP + FP).

So, **Precision** identifies the proportion of correctly predicted positive outcome. It is more concerned with the positive class than the negative class.

Mathematically, precision can be defined as the ratio of `TP to (TP + FP).`

In [225…

```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9655
```

Recall

Recall can be defined as the percentage of correctly predicted positive outcomes out of all the actual positive outcomes. It can be given as the ratio of true positives (TP) to the sum of true positives and false negatives (TP + FN). **Recall** is also called **Sensitivity**.

Recall identifies the proportion of correctly predicted actual positives.

Mathematically, recall can be given as the ratio of `TP to (TP + FN).`

In [226…

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.8930
```

True Positive Rate

True Positive Rate is synonymous with **Recall**.

In [227…

```
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.8930
```

False Positive Rate

In [228…

```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.4878
```

False positive rate is a key component in understanding the trade-offs between sensitivity (true positive rate) and specificity (true negative rate).

Specificity

In [229…

```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.5122
```

A specificity of 0.5122 indicates that the model is only marginally better than random guessing when it comes to identifying negative instances.

12. Adjusting the threshold level

In [230…

```
# print the first 10 predicted probabilities of two classes- 0 and 1

y_pred_prob = logreg.predict_proba(X_test)[0:10]

y_pred_prob
```

Out[230…

```
array([[0.69492933, 0.30507067],
       [0.96464203, 0.03535797],
       [0.79689587, 0.20310413],
       [0.91123407, 0.08876593],
       [0.1302019 , 0.8697981 ],
       [0.31063709, 0.68936291],
       [0.54002009, 0.45997991],
       [0.90803103, 0.09196897],
       [0.73837134, 0.26162866],
       [0.6051113 , 0.3948887 ]])
```

Observations

- In each row, the numbers sum to 1.
- There are 2 columns which correspond to 2 classes - 0 and 1.
    - Class 0 - predicted probability that the customer will churn the SyriaTel.
    - Class 1 - predicted probability that the customer will remain loyal to SyriaTel.
- Importance of predicted probabilities
    - We can rank the observations by probability of churn or does not churn.
- predict_proba process
    - Predicts the probabilities
    - Choose the class with the highest probability
- Classification threshold level
    - There is a classification threshold level of 0.5.
    - Class 1 - probability of rain is predicted if probability > 0.5.
    - Class 0 - probability of no rain is predicted if probability < 0.5.

13. k-Fold Cross Validation

In [231…

```
# Applying 5-Fold Cross Validation

scores = cross_val_score(logreg, X_train, y_train, cv = 5, scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.8670412  0.8630394  0.85928705 0.86866792 0.8424015
]
```

We can summarize the cross-validation accuracy by calculating its mean.

In [232…

```
# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.8601
```

Our, original model score is found to be 0.8696. The average cross-validation score is 0.8601. So, we can conclude that cross-validation does not result in performance improvement.

14. Results and Observation of logistic regression

1. The logistic regression model accuracy score is 0.8696. So, the model does a very good job in predicting whether or not custmoers will churn SyriaTel.
2. Small number of observations predict that the customers will churn SyriaTel. Majority of observations predict that the customers will remain loyal to SyriaTel.
3. The model shows no signs of overfitting.
4. Increasing the value of C results in higher test set accuracy and also a slightly increased training set accuracy. So, we can conclude that a more complex model should perform better.

Lets now compare the logistic regression model with a decisions tree model

15. Decision trees

It is a supervised machine learning algorithm that can be used to classify data. Decision trees work by splitting the data into smaller and smaller subsets until each subset contains only data of a single class. The decision tree then predicts the class of a new data point by following the path down the tree that corresponds to the values of its features. We will make use of CART (Classification and Regression Trees) to create our decision tree since it can handle both classification and regression tasks.

In [233…

```
# data split (already done) specify test size and random state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

In [234…

```
# One-hot encode the training data and show the resulting DataFrame with prope
ohe = OneHotEncoder()

ohe.fit(X_train)
X_train_ohe = ohe.transform(X_train).toarray()

# Creating this DataFrame is not necessary its only to show the result of the
```

```
ohe_df = pd.DataFrame(X_train_ohe, columns=ohe.get_feature_names_out(X_train.c

ohe_df.head()
```

Out[234…

	state_AK	state_AL	state_AR	state_AZ	state_CA	state_CO	state_CT	state_DC	sta
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 8684 columns

In [235…

```
# Create the classifier, fit it on the training data and make predictions on t
clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X_train_ohe, y_train)
```

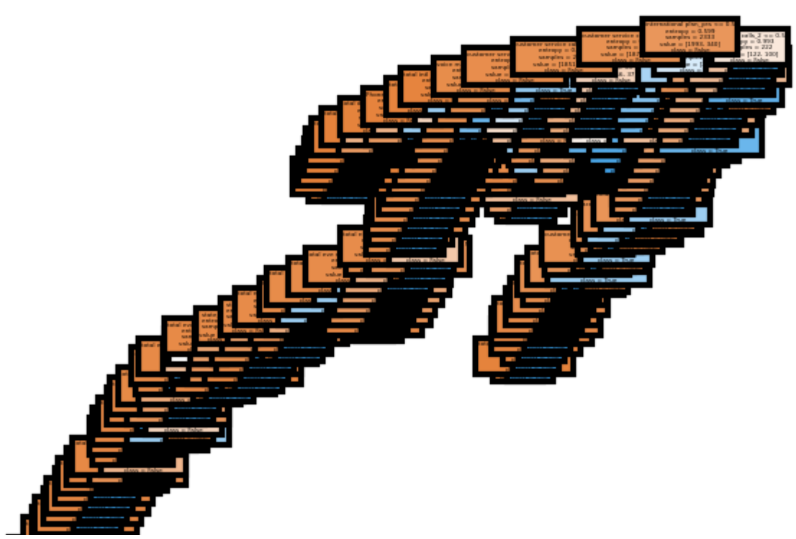
Out[235…

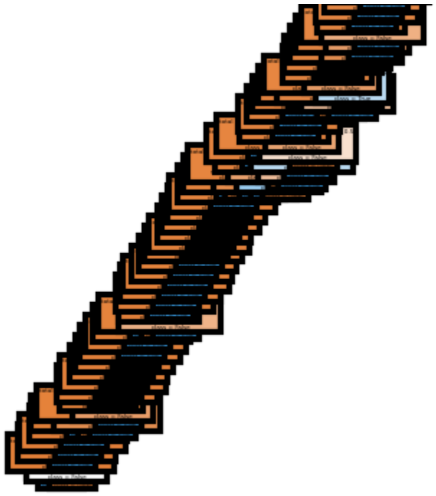
```
DecisionTreeClassifier(criterion='entropy')
```

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

In [236…

```
#plot the tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (3,3), dpi=300)
tree.plot_tree(clf,
               feature_names = ohe_df.columns,
               class_names=np.unique(y).astype('str'),
               filled = True)
plt.show()
```





16. Model Evaluation

In this section, we'll evaluate models based on the classification metrics accuracy in particular. After, we will recommend the best model to implement.

In [237…

```
# Initialize OneHotEncoder with handle_unknown='ignore'
ohe = OneHotEncoder(handle_unknown='ignore')

# Fit and transform the combined training and test data
combined = pd.concat([X_train, X_test])
ohe.fit(combined)

# Transform the training data
X_train_ohe = ohe.transform(X_train)

# Transform the test data
X_test_ohe = ohe.transform(X_test)

# List of classifiers
classifiers = [LogisticRegression(), DecisionTreeClassifier()]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'accuracy', 'recall'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train_ohe, y_train)  # Fit on the encoded training data
    y_pred = model.predict(X_test_ohe)  # Predict on the encoded test data

    accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy score
    recall = recall_score(y_test, y_pred)  # Calculate recall score

    result_table = result_table.append({'classifiers': cls.__class__.__name__,
                                        'accuracy': accuracy,
```