Customer Churn Reduction

Author-

Pooja Yadwani

Contents

Chapter 1

Introduction

Customer churn refers to a situation when a customer (player, subscriber, user, etc.) ceases her/is relationship with a company. Businesses typically treat a customer as churned once a specific amount of time has elapsed since the customer's last interaction with the site or service. Total cost of customer churn includes both lost revenue and the marketing costs involved in replacing those customers with the new ones. Reduction of customer churn is important because cost of acquiring a new customer is higher than retaining the existing one. Reducing customer churn is a key business goal of every business. This case is related to telecom industry where organizations want to know that for given certain parameters whether a person will churn or not.

1.1 Problem Statement

We have been provided with a train and test dataset of a telecom company. The data set consists of 20 variables describing various services and charges associated with them, duration and any service calls made by customer. The dataset has geographic location of customers in the form of state and area code. '**Churn'** is the target variable, which defines weather the customer has churned or not.

1.2 Data

Two different (files as Test.csv and Train.csv) datasets were provided as train data and test data. Data contains 20 predictor variables and 1 target variables.

Variables		**Description**
State	:	State to which customer belongs
Account length	:	Service usage period
Area code	:	Telephone area code
Phone number	:	Customer's phone number
International plan	:	'yes' if customer opted for international plan else 'no'
Voice mail plan	:	'yes' if customer opted for voice mail plan else 'no'
Number vmail messages	:	Number of voice messages stored or received by customer
Total day minutes	:	Total minutes in day time usage
Total day calls	:	Total calls made in day time
Total day charge	:	Charges for services used during day time
Total eve minutes	:	Total minutes in evening time usage
Total eve calls	:	Total calls made in evening time
Total eve charge	:	Charges for services used during evening time
Total night minutes	:	Total minutes in night time usage
Total night calls	:	Total calls made in night time
Total night charge	:	Charges for services used during night time
Total intl minutes	:	Total international minutes used
Total intl calls	:	Total international calls made
Total intl charge	:	Charges for international calls
Number customer service calls	:	Services call made by customer
Churn	:	**Target -** 'True' If customer churned else 'False'

<u>Size of Dataset Provided: -</u>
Train.csv = 3333 rows, 21 Columns
Test.csv = 1667 rows, 21 Columns

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
KS	128	415	382-4657	no	yes	25	265.1	110	45.07
OH	107	415	371-7191	no	yes	26	161.6	123	27.47
NJ	137	415	358-1921	no	no	0	243.4	114	41.38
OH	84	408	375-9999	yes	no	0	299.4	71	50.9
OK	75	415	330-6626	yes	no	0	166.7	113	28.34

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	number customer service calls	Churn
197.4	99	16.78	244.7	91	11.01	10	3	2.7	1	False.
195.5	103	16.62	254.4	103	11.45	13.7	3	3.7	1	False.
121.2	110	10.3	162.6	104	7.32	12.2	5	3.29	0	False.
61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False.
148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False.

***Churn is Our Target Variable here.**

Chapter 2

Methodology

Customer churn reduction is a business scenario in which a company is trying to retain customers which are more likely to leave the services. For reducing churn rate, we need to identify which customers are most likely to churn and which are not. Also, we have some data to train our model which makes our problem as Supervised Classification problem.

- **Exploratory Data Analysis(EDA)-** It includes following steps
    - Looking into the data and analyzing various variables,
    - Visualization,
    - Missing value analysis,
    - Correlation analysis,
    - Chi-square test,
    - Feature Scaling
    - Feature Sampling.

- **Basic Modeling-** Trying different models over preprocessed data
    - Random forest,
    - Logistic regression,
    - KNN,
    - Naïve Bayes

- **Model Evaluation & Optimization-** Evaluating model performances and then selecting the best model fit for our data, optimizing hyper parameters tuning and cost effectiveness of model. This step is optional. We may or may not involve it. It is basically done to avoid a scenario where the selected approach works very well with training data but fails to support out test data in similar way.

- **Implementation model on Final test data and saving the results**

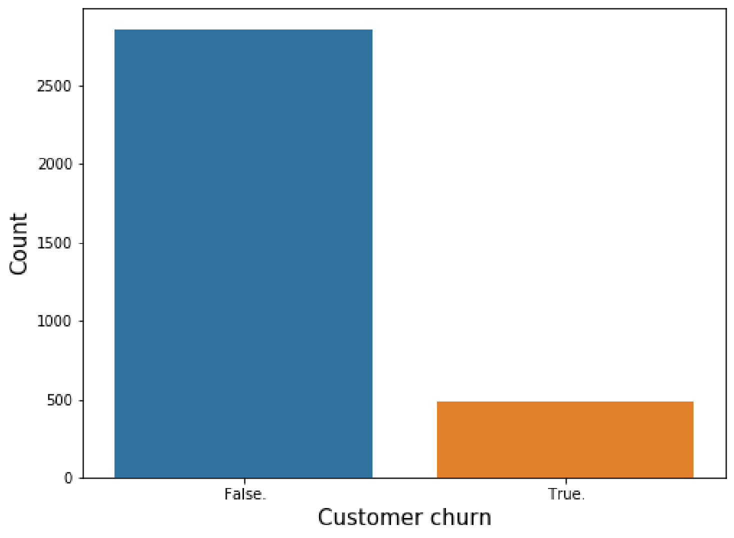
2.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis refers to the critical process of performing initial investigations on data to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. It is a good practice to understand the data first and try to gather as many insights from it. EDA is all about making sense of data in hand, before getting them dirty with it.

2.1.1 Target Variable - Churn

Our target variable has two categories which include True and False values: -

- **True** = Customer will churn out.
- **False** = Customer won’t churn



We can clearly see that our data is highly imbalanced. The occurrence of false is much higher than True. There are 2850 (85.51%) customers who churn out and 483 (14.49%) customers to be retained.

2.1.2 Uniqueness in Variable

Let's have a look at count of unique values for each variable-

Variable	Unique Counts
State	51
account length	212
area code	3
phone number	3333
international plan	2
voice mail plan	2
number vmail messages	46
total day minutes	1667
total day calls	119
total day charge	1667
total eve minutes	1611
total eve calls	123
total eve charge	1440
total night minutes	1591
total night calls	120
total night charge	933
total intl minutes	162
total intl calls	21

total intl charge	162
number customer service calls	10
Churn	2

Conclusion:

- **area code** has only 3 values, So, we will convert it to categorical variable.
- **phone number** has 3333, which makes it a full unique variable. This can be removed as it doesn't contain any information that is relevant for our analysis.

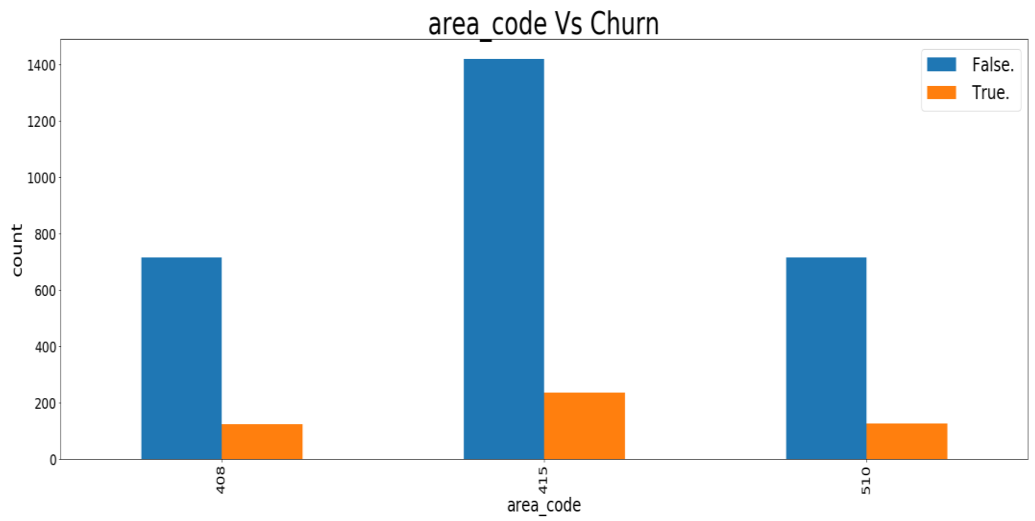
2.1.3 Missing Value Analysis

Variable	Values
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
number customer service calls	0
Churn	0

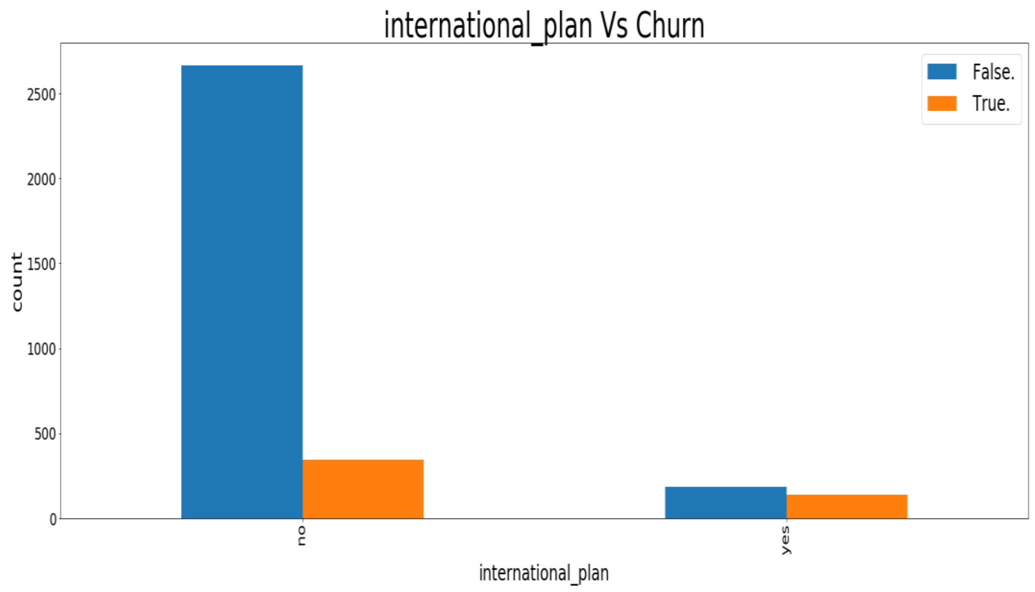
No missing values were present in the training and test dataset.

2.1.4 Churning of Customers according to different variables

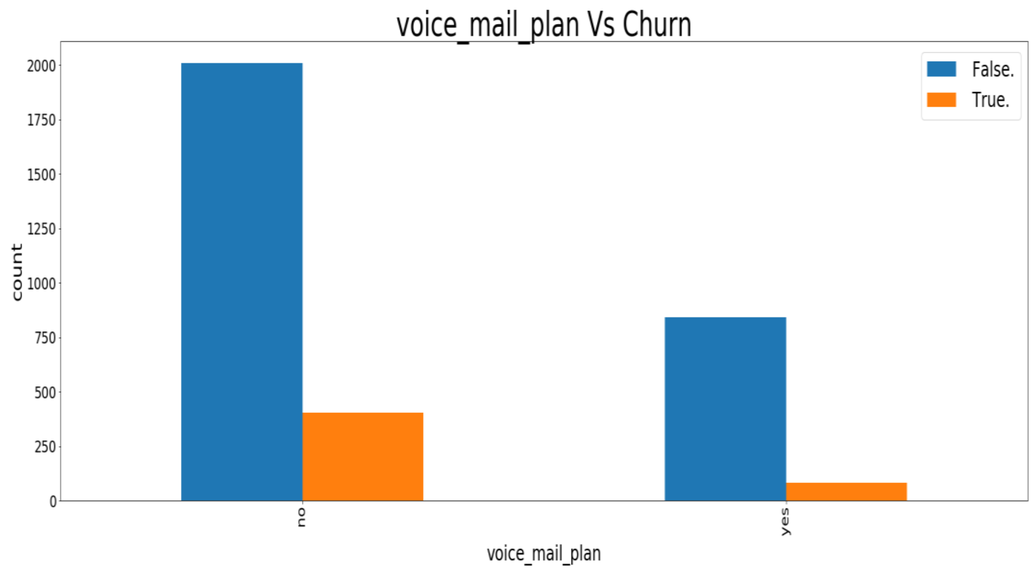
- Area code- Most of the churned customers are from 415 area



- International plan- Churn rate is more with customer using international plan. As only 323 customer using International plan and 137 churning out of them.



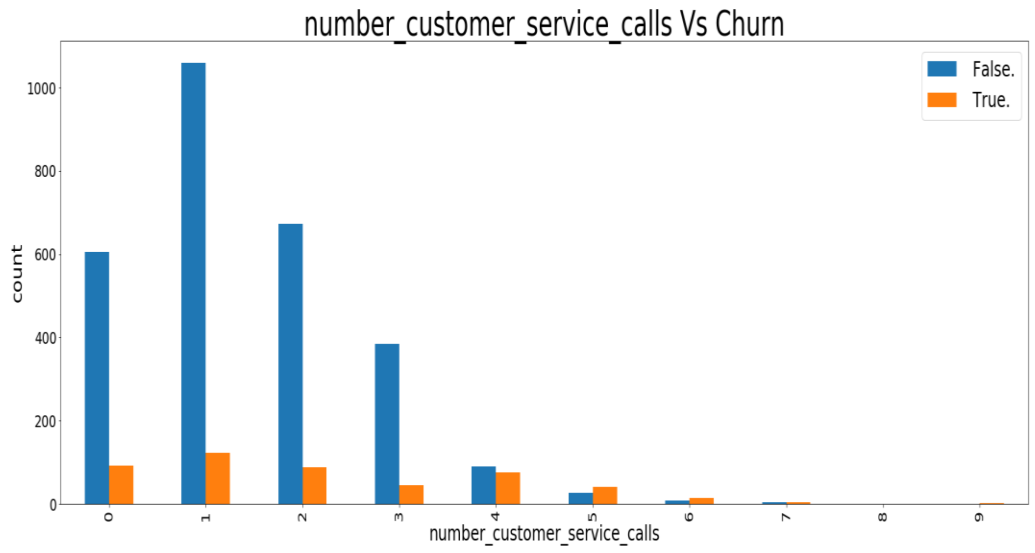
- Voice Mail Plan- 922 customer using voice mail plan and 80 out of them are churning.



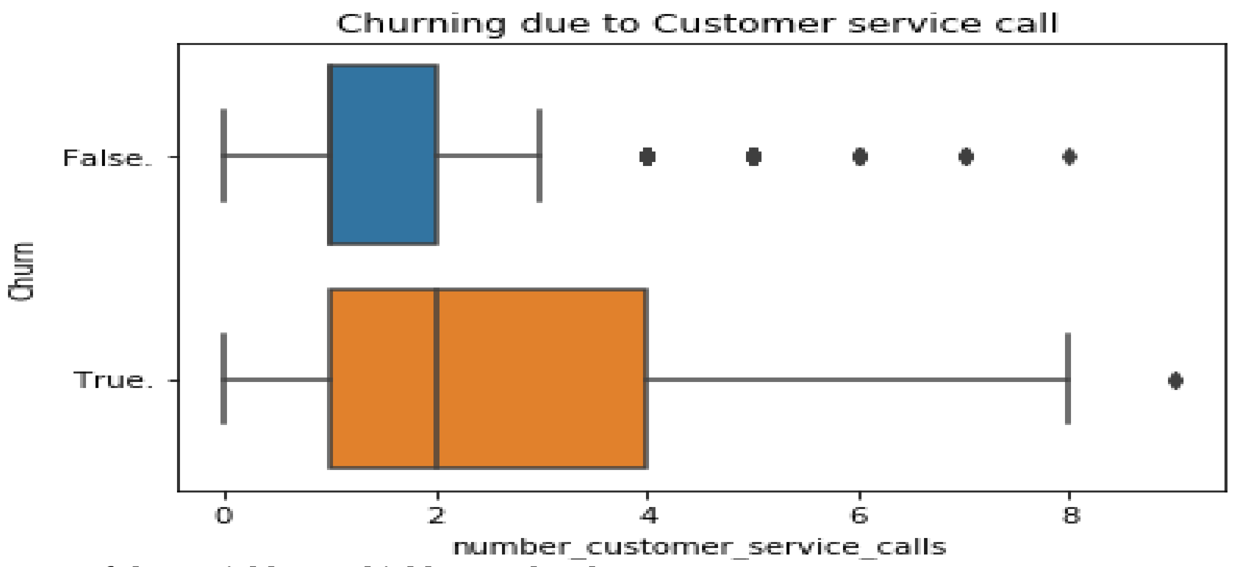
- Churned Customers who has or not (voice mail plan and international plan)
  - Churn rate for Customer neither having voice mail plan nor international plan is 9.06%.
  - Churning rate for customer having International plan but don't have voice mail plan is 3.03% out of 6.93% customers.
  - Churning of customer having both voice mail plan & international plan is 1.08% out of 2.76%

.

- Customer Service Calls impact to Churn- 20.68% customer Churn due to customer service calls

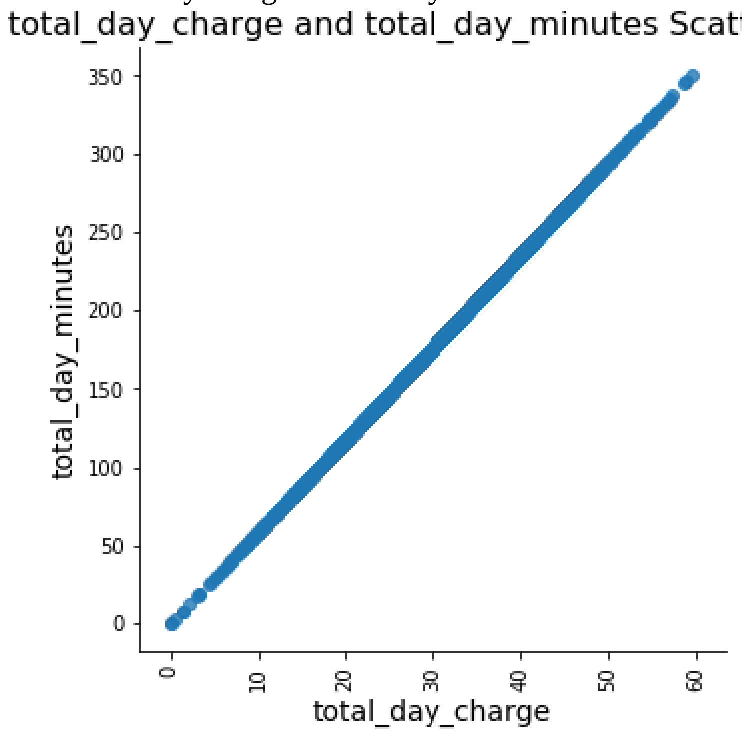


We can note that Churn rate is increasing with increase in customer service call frequency.

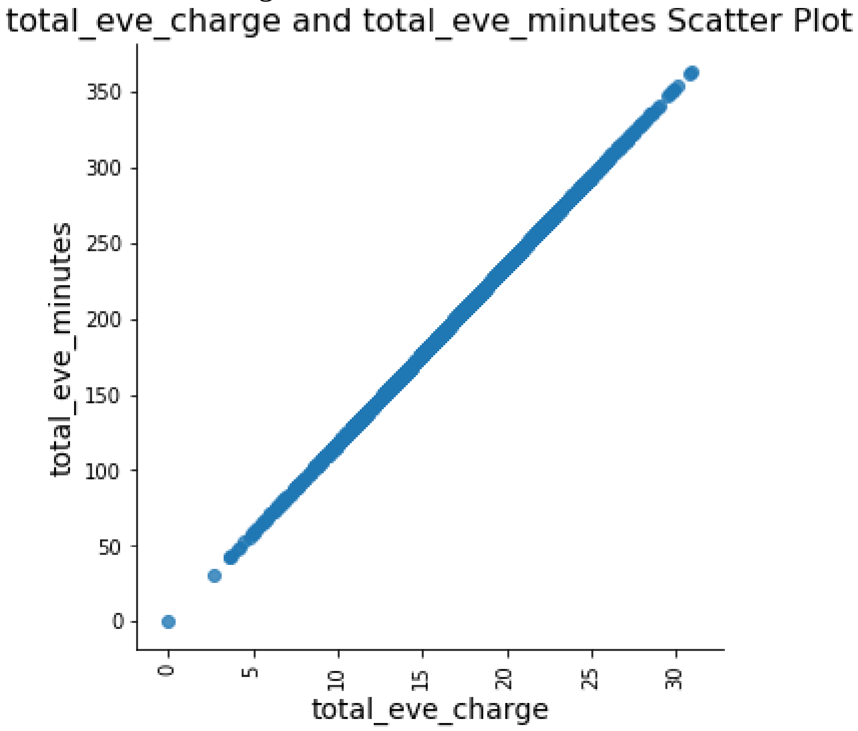


Some of the Variables are highly correlated: -

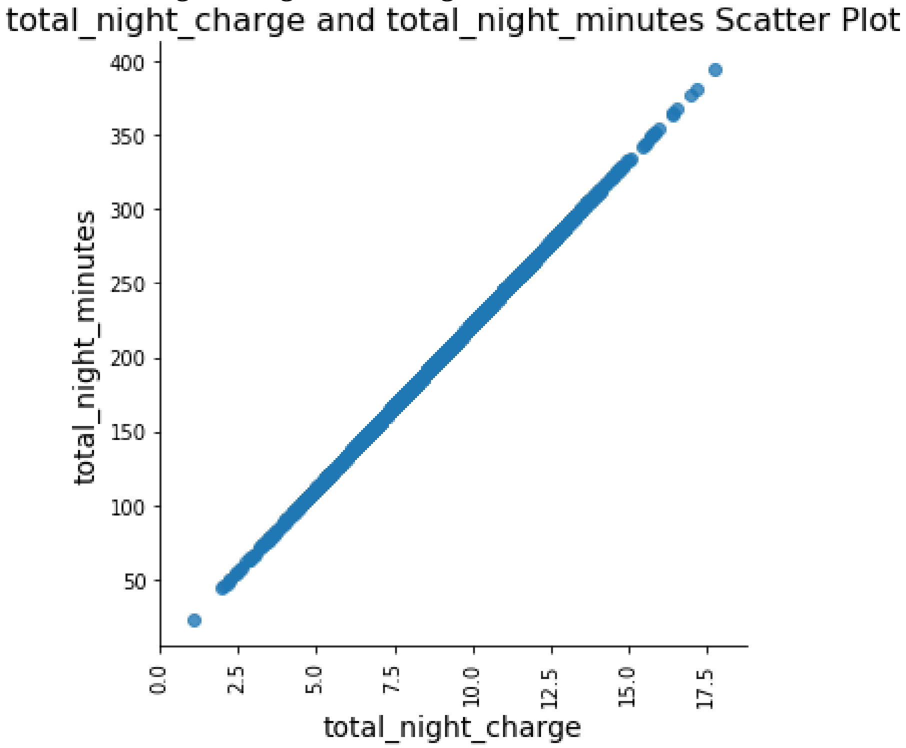
- total day charge & total day minute



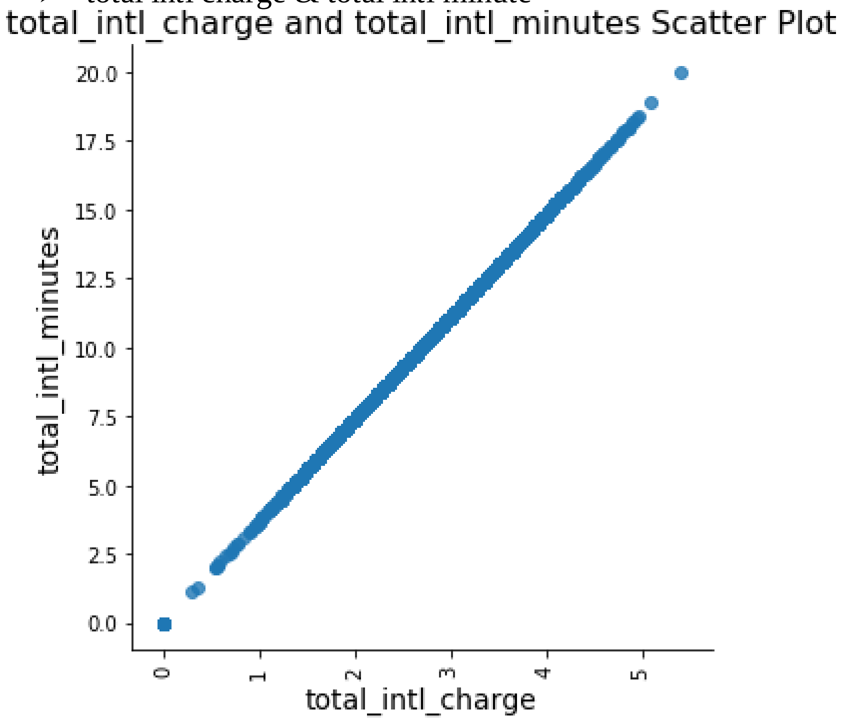
- total eve charge & total eve minute



- total night charge & total night minute



- total intl charge & total intl minute



2.1.5 Feature Selection

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance.
Feature selection and Data cleaning should be the first and most important step of your model designing. Feature selection techniques are used for four reasons:

1. Simplification of models to make them easier to interpret by researchers.
2. Shorter training times.
3. to avoid the curse of dimensionality,
4. Enhanced generalization by reducing over fitting (reduction of variance).

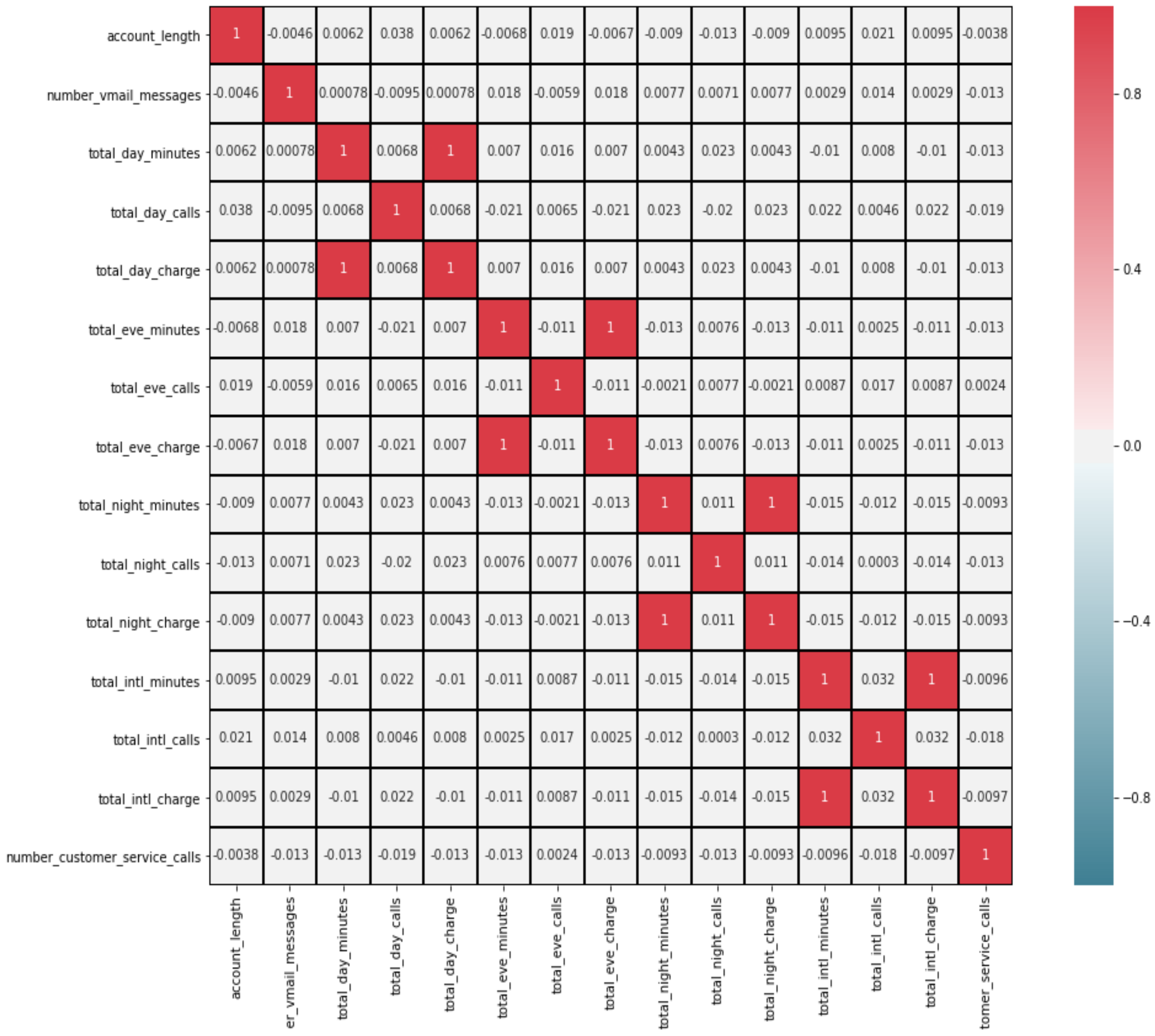
For Continuous variable we use Correlation Matrix whereas for categorical variable we use Chi Square test.

- **Correlation Analysis**

Correlation analysis is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continuous variables (e.g. height and weight). This particular type of analysis is useful when a researcher wants to establish if there are possible connections between variables. We can easily analyze it creating a **heatmap** for our dataset.

Variables that are highly correlated are highlighted with red color with their corresponding score. From this correlation plot we can infer that –

- ‘Total day minutes’ and ‘total day charges’ are highly correlated
- ‘Total eve minutes’ and ‘total eve charges’ are highly correlated
- ‘Total night minutes’ and ‘total night charges’ are highly correlated
- ‘Total intl minutes’ and ‘total intl charges’ are highly correlated



➢ **Chi Square test – Categorical Variables**

Chi-square test is one of the important nonparametric tests that is used to compare more than two variables for a randomly selected data. The expected frequencies are calculated based on the conditions of null hypothesis. The rejection of null hypothesis is based on the differences of actual value and expected value. Chi-square will give us a p-value and if p value is less than 0.05 we will remove the variable because if p-value is less than 0.05 means that variable is independent and not contributing much information in explaining to our target variable.

Variables and p-values-

1. **State** : 0.002296221552011188
2. **area_code** : 0.9150556960243712
3. **international_plan** : 2.4931077033159556e-50
4. **voice_mail_plan** : 5.15063965903898e-09

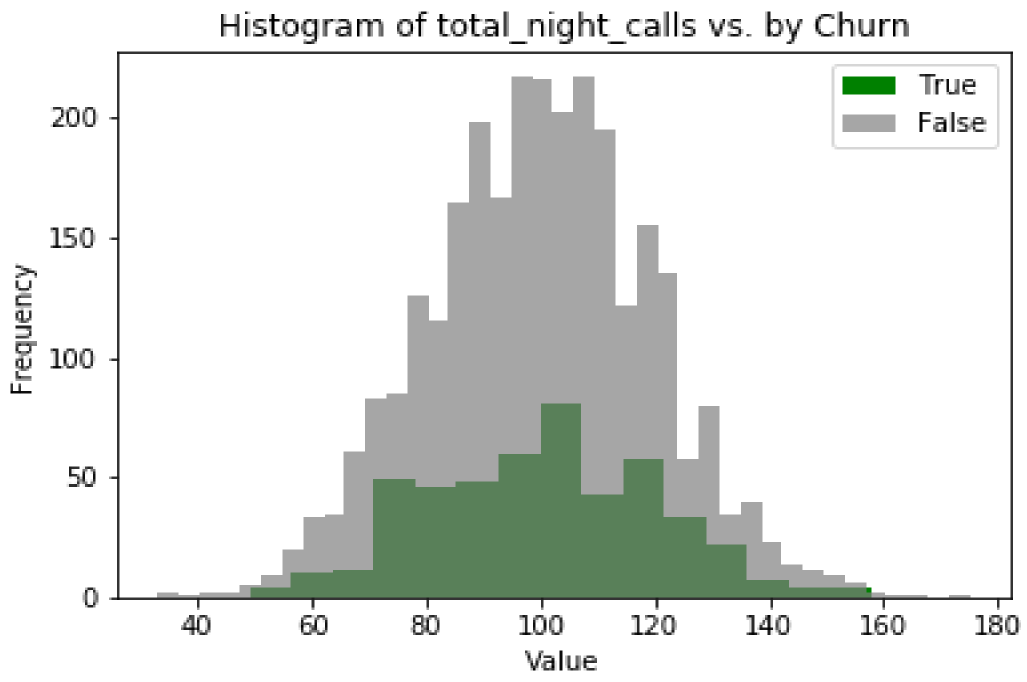
Removing all the redundant variables: -

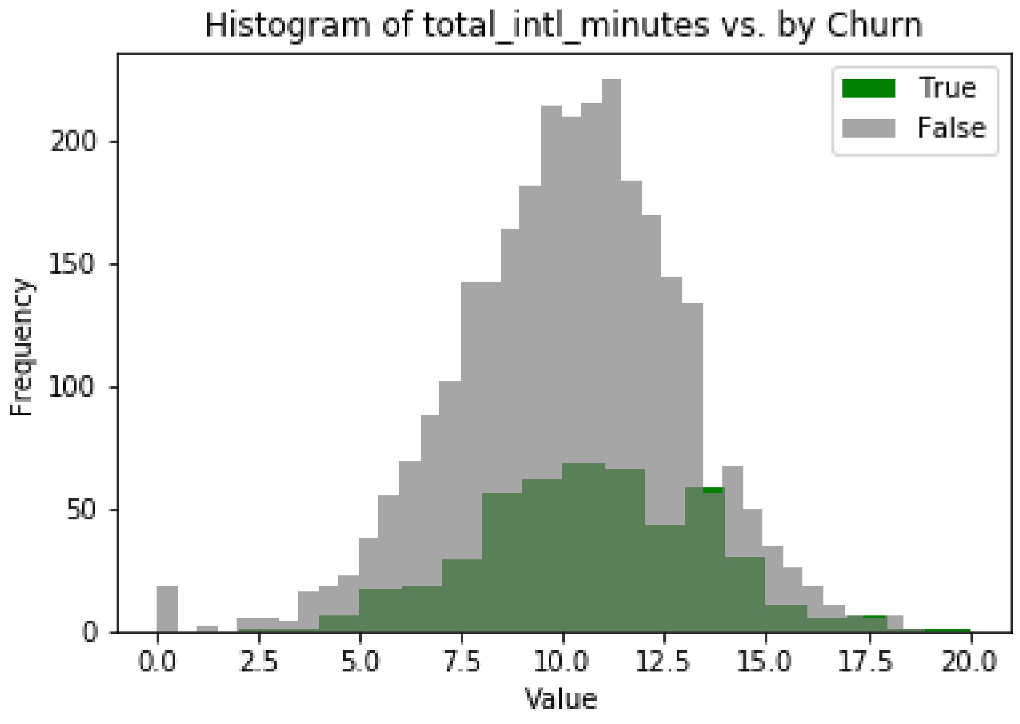
- 'state','total_day_charge'
- 'total_eve_charge'
- 'total_night_charge'
- 'total_intl_charge'

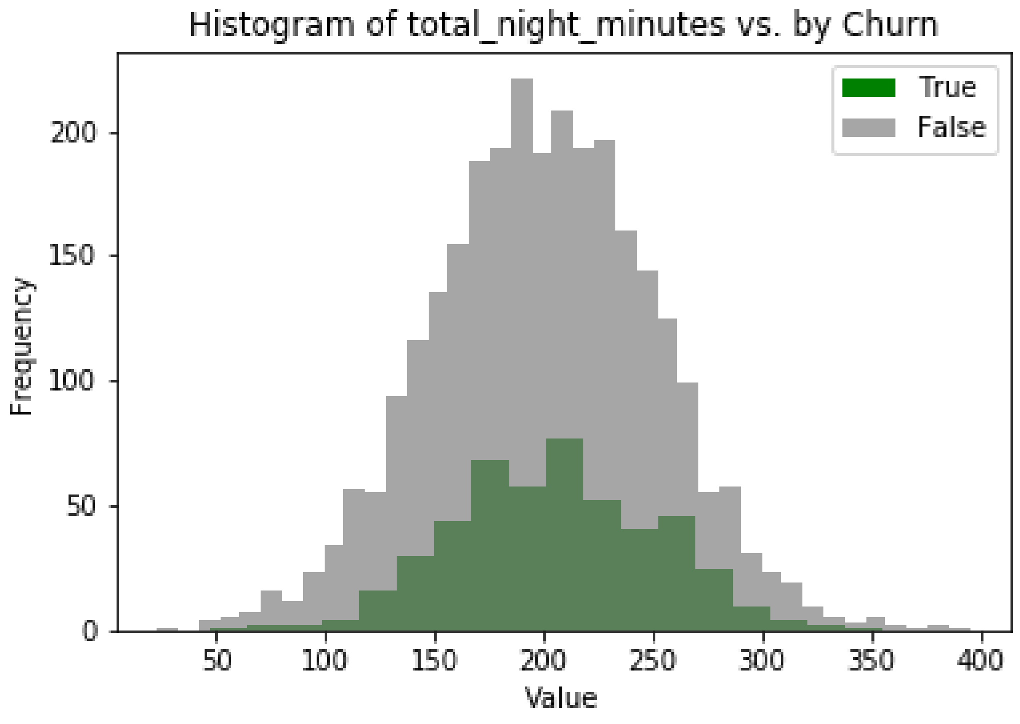
2.1.6 Feature Scaling

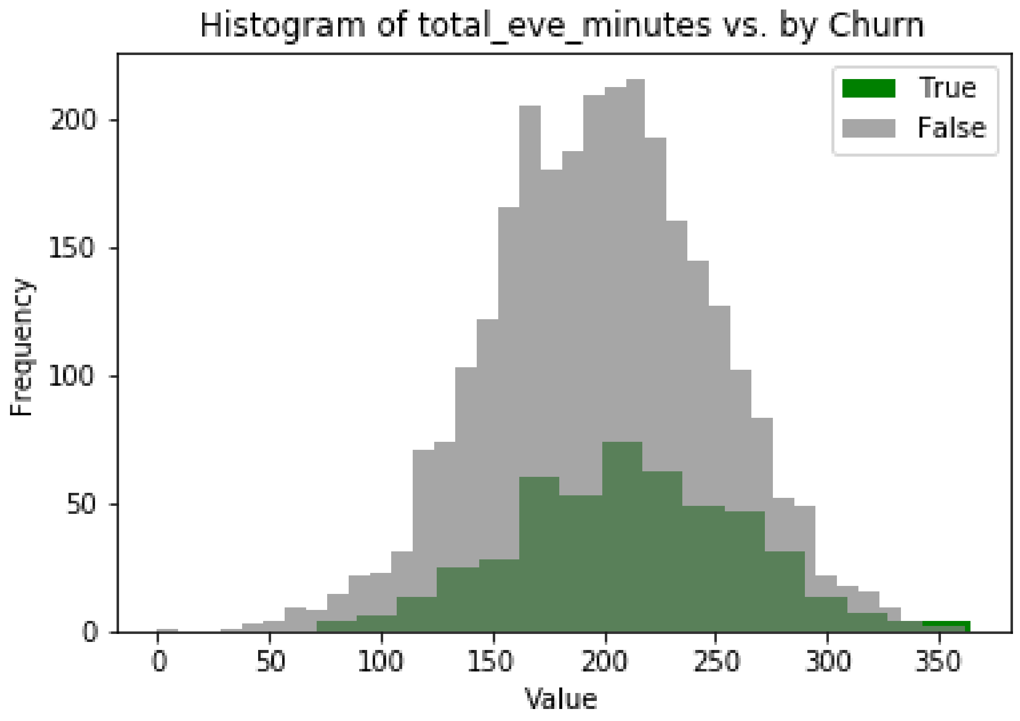
Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.

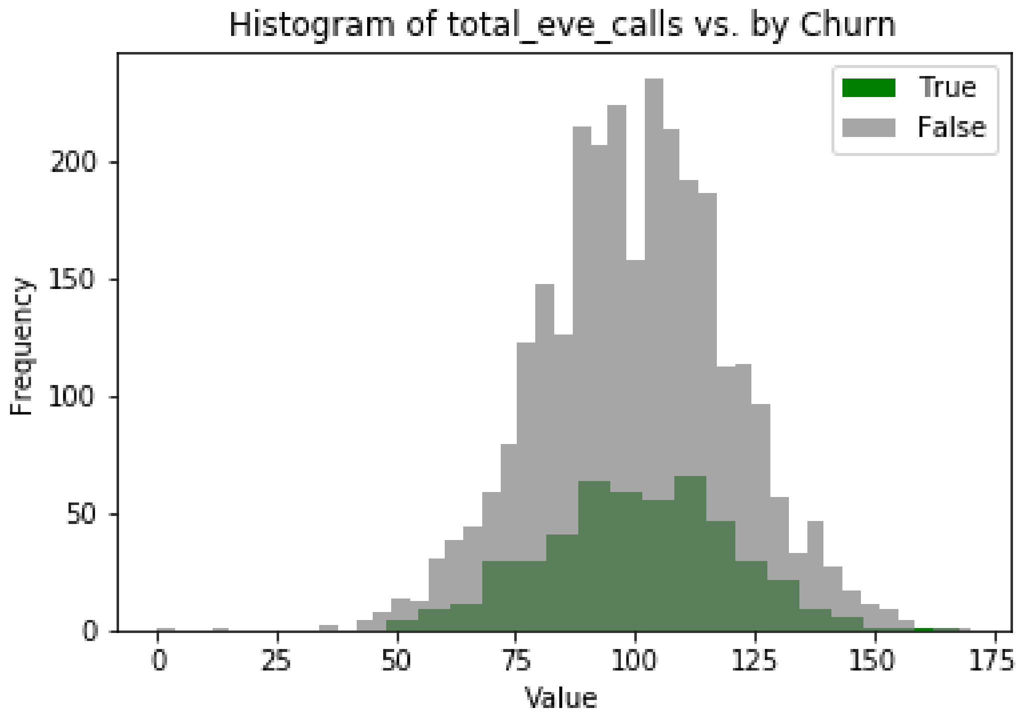
Let us visualize the distribution of continuous variables: -

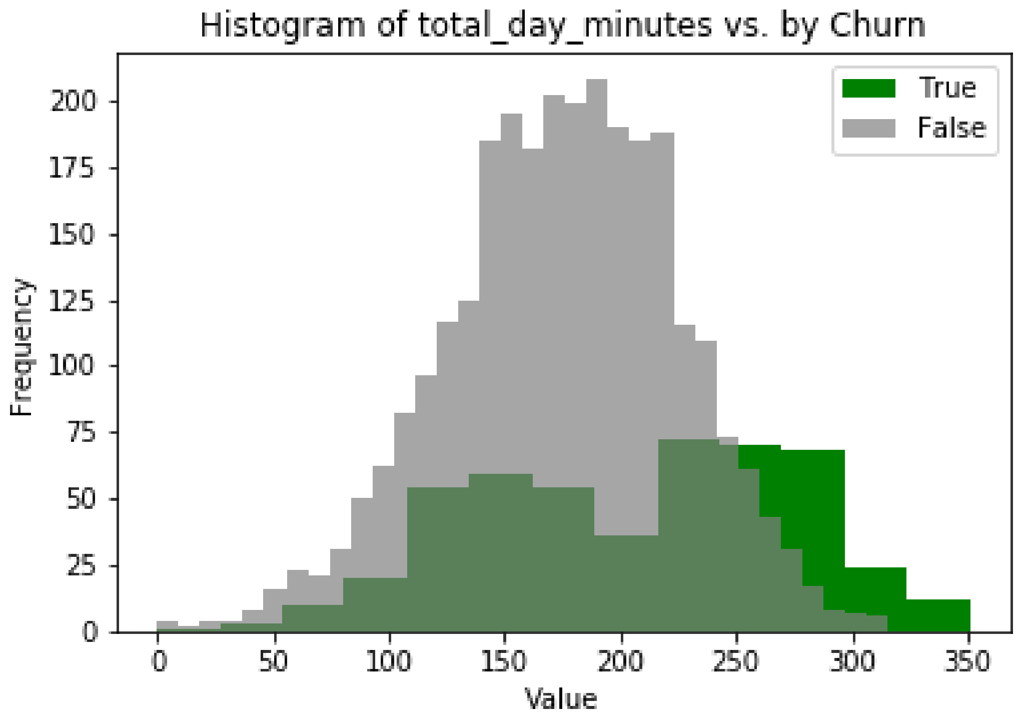


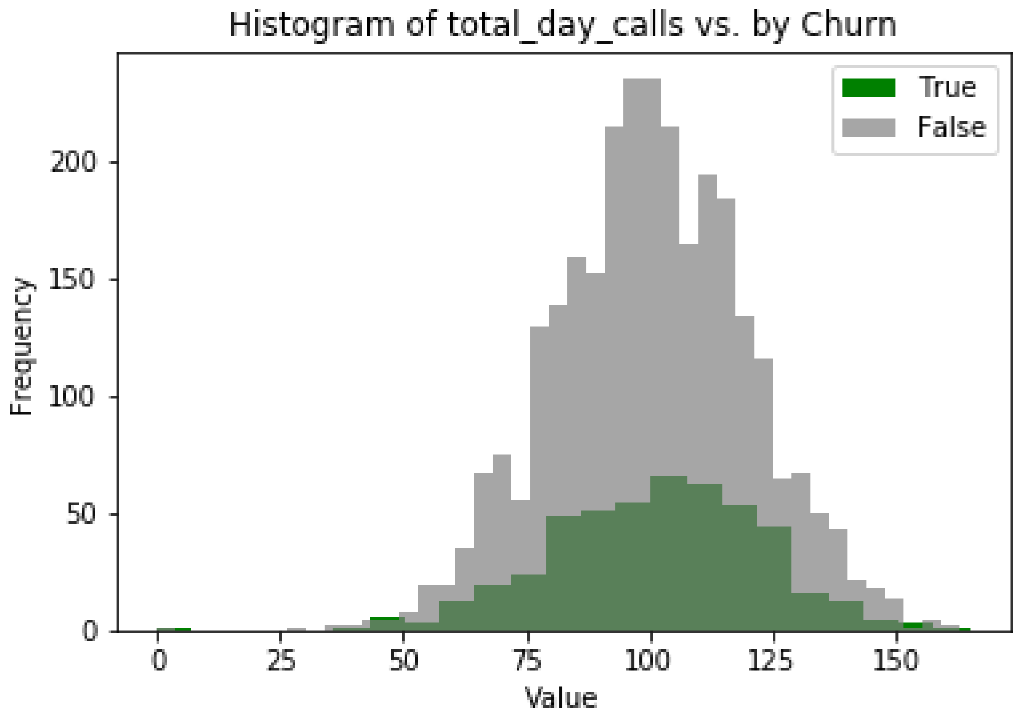

Histogram of total_intl_minutes vs. by Churn
True
False
Frequency
Value
0
50
100
150
200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

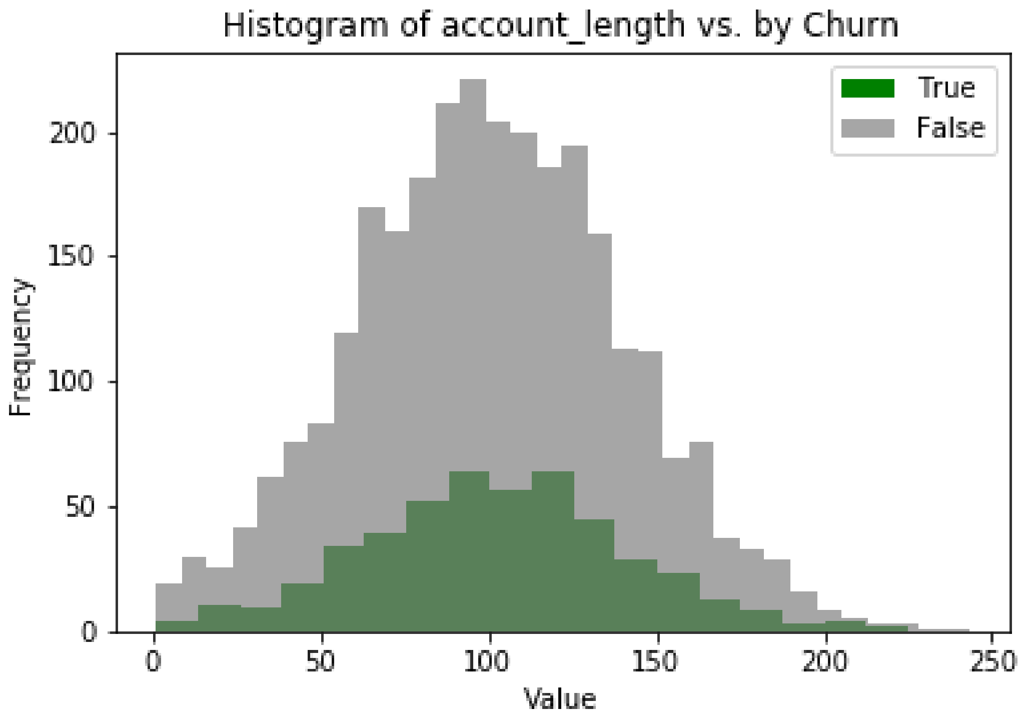

Histogram of total_night_minutes vs. by Churn
True
False
Frequency
Value
0
50
100
150
200
50
100
150
200
250
300
350
400


Histogram of total_eve_minutes vs. by Churn
True
False
Frequency
Value
0
50
100
150
200
250
300
350


Histogram of total_eve_calls vs. by Churn
True
False
Frequency
Value
0
25
50
75
100
125
150
175


Histogram of total_day_minutes vs. by Churn
True
False
Frequency
Value
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350


Histogram of total_day_calls vs. by Churn
True
False
Frequency
Value
0
50
100
150
200
0
25
50
75
100
125
150


Histogram of account_length vs. by Churn
True
False
Frequency
Value
0
50
100
150
200
250

We can see that most of our continuous data distribution is uniformly. We will use Standardization \ Z - Score here.

- **Standardization**: - It will convert mean or Average of each variable to Zero and the each value of variable will convert to unique standard Deviation.

- **Data after Scaling**

account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_eve_minutes	total_eve_calls
0.676388	1	0	1	1.234697	1.566532	0.476572	-0.0706	-0.05593
0.149043	1	0	1	1.307752	-0.33369	1.124334	-0.10806	0.144845
0.902393	1	0	0	-0.59167	1.168128	0.675883	-1.57315	0.496204
-0.42853	0	1	0	-0.59167	2.196267	-1.46672	-2.74245	-0.60807
-0.65453	1	1	0	-0.59167	-0.24005	0.626055	-1.03878	1.098534

total_night_minutes	total_night_calls	total_intl_minutes	total_intl_calls	number_customer_service_calls	Churn
0.866613	-0.46543	-0.085	-0.60111	-0.427868	0
1.058412	0.147802	1.240296	-0.60111	-0.427868	0
-0.75676	0.198905	0.703015	0.211502	-1.18804	0
-0.07854	-0.56763	-1.30283	1.024109	0.332305	0
-0.27627	1.067643	-0.04918	-0.60111	1.092477	0

2.1.7 Feature Sampling (Train = Train + Validation)

High dimensional data with thousands of features present a big challenge to current clustering algorithms. Sparsity, noise and correlation of features are common characteristics of such data. Another common phenomenon is that clusters in such high dimensional data often exist in different subspaces. Ensemble clustering is emerging as a prominent technique for improving robustness, stability and accuracy of high dimensional data clustering. Under sampling we will divide train data we have into train test split.

- In Python we have used train_test_split() for sampling the train.csv data into train and validation data.
- In R we use createDataPartition() for randomly chosen values from each class.

Both methods use stratified sampling technique to cut the data into train and validation set. Our target variable class is imbalanced and after split of data in our train set we get

```
# False True
# 1881 319
```

If we train our model in this data, then our model training will get biased and will accurately predict target class False more than True. To overcome the imbalanced data problem, we will go for over sampling of training data. There are multiple approaches to do over sampling but here we will use synthetic over sampling. We have used SMOTE in Python and ROSE in R.

2.1.7.1 SMOTE Oversampling in Python:-

SMOTE is an over-sampling method. What it does is, it creates synthetic (not duplicate) samples of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighboring records. It synthesizes new minority instances between existing real minority instances.

- ➢ **Before**: - False = 1895 // True = 338
- ➢ **After Smote:** - False = 1895 // True = 1895

2.1.7.2 ROSE Oversampling in R:-

IN R we have used ROSE sampling technique which is like SMOTE. It is also generating the synthetic data points and it will under sample some random points from majority class.

- ➢ **Before**: - False = 1881 // True =319
- ➢ **After ROSE:** - False = 1101 // True=1019

Finally, our data is ready to feed to the machine learning model.

Chapter 3

Modeling

Customer churn reduction is a binary classification problem. Here we must build a model which can classify if a customer will churn out or not. Hence, to deal with particular problem we will use multiple Classification Models here.

1. Random Forest
2. Logistic Regression
3. K- Nearest Neighbors
4. Naïve Bayes

We will implement all four models on our preprocessed data in both Python and R and later on decide the final model that best suits our data set.

3.1 Random Forest

Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

Summary of Random forest model: -

Python

Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	925	30
TRUE	28	117

AUC = 0.80

Classification paradox	**Value**
Accuracy	94.73
True Negative Rate	96.86
True Positive Rate / Recall	80.69
False Negative Rate	19.31
False Positive Rate	3.14

	precision	**recall**	**f1-score**	**support**
FALSE	0.97	0.97	0.97	955
TRUE	0.80	0.81	0.83	145
Avg/total	0.95	0.95	0.95	1100

R

Parameter	Value
Accuracy	86.23124448
FNR	17.68292683
FPR	13.10629515
precision	51.52671756
recall//TPR	51.52671756
Sensitivity	82.31707317
Specificity	86.89370485

3.2 Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine the outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a logit transformation of the probability of presence of the characteristic of interest.

Summary of Logistic Regression model: -

Python

Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	752	203
TRUE	37	108

AUC = 0.81

Classification paradox	Value
Accuracy	78.18
True Negative Rate	78.74
True Positive Rate / Recall	74.48
False Negative Rate	25.52
False Positive Rate	21.26

	precision	recall	f1-score	support
FALSE	0.95	0.79	0.86	955
TRUE	0.35	0.74	0.47	145
Avg/total	0.87	0.78	0.81	1100

R

Parameter	Value
Accuracy	78.19947043
FNR	28.65853659
FPR	20.63983488
precision	36.90851735
recall//TPR	36.90851735
Sensitivity	71.34146341
Specificity	79.36016512

3.3 K- Nearest Neighbor

K-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- **In k-NN classification,** the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- **In k-NN regression**, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Summary of KNN model: -

Python

Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	759	196
TRUE	45	100

AUC = 0.80

Classification paradox	Value
Accuracy	78.09
True Negative Rate	79.48
True Positive Rate / Recall	68.97
False Negative Rate	31.03
False Positive Rate	20.52

	precision	recall	f1-score	support
FALSE	0.94	0.79	0.86	955
TRUE	0.34	0.69	0.45	145
Avg/total	0.86	0.78	0.81	1100

R

Parameter	Value
Accuracy	78.81729921
FNR	46.34146341
FPR	16.9246646
precision	34.92063492
recall//TPR	34.92063492
Sensitivity	53.65853659
Specificity	83.0753354

3.4 Naïve Bayesian

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods

Algorithm: -Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assumes that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

Summary of Naïve Bayesian model: -

Python

Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	754	201
TRUE	34	111

AUC = 0.82

Classification paradox	Value
Accuracy	78.64
True Negative Rate	78.95
True Positive Rate / Recall	76.55
False Negative Rate	23.45
False Positive Rate	21.05

	precision	recall	f1-score	support
FALSE	0.96	0.79	0.87	955
TRUE	0.36	0.77	0.49	145
Avg/total	0.88	0.79	0.82	1100

R

Parameter	Value
Accuracy	83.14210062
FNR	29.26829268
FPR	14.75748194
precision	44.78764479
recall//TPR	44.78764479
Sensitivity	70.73170732
Specificity	85.24251806

Chapter 4

4.1 Model Evaluation

Model evaluation is the process of choosing between models, different model types, tuning parameters, and features. Better evaluation processes lead to better and more accurate models in analysis process.

In previous chapter we have built: -

- ➢ Random Forest
- ➢ Logistic Regression
- ➢ KNN
- ➢ Naïve Bayesian

We have chosen Classification Accuracy Matrix for our evaluation. It includes following parameters:

- ✓ **Accuracy**: the proportion of the total number of predictions that was correct.
- ✓ **Positive Predictive Value or Precision:** the proportion of positive cases that were correctly identified.
- ✓ **Negative Predictive Value**: the proportion of negative cases that were correctly identified.
- ✓ **Sensitivity or Recall**: the proportion of actual positive cases which are correctly identified.
- ✓ **Specificity**: the proportion of actual negative cases which are correctly identified.
- ✓ **False Positive Rate (Type –I error):** False positive, commonly called a "false alarm", is a result that indicates a given condition exists, when it does not.
- ✓ **False Negative Rate (Type – II error):** false negative, is a test result that indicates that a condition does not hold, while in fact it does.

Let’s understand False Negative and False Positive according to our problem statement-

- ➢ If in real any customer is not churning, and our model predicts that he /she will churn. Then it’s okay as we can deal with it. Since, we will start putting more effort on the specific customer so that he/she stays and won’t churn out.
- ➢ But in other case if our model predicts that a particular customer won’t churn out and in actual he will churn out, then there might be a big problem, because in this scenario our client will lose some important customers.

For both cases it’s important to make a good trade off, predict more accurate and have low false negative rate.

Let’s check and compare the results of R and Python-

Models		**Random Forest**	**Logistic Regression**	**KNN**	**Naïve Bayes**
Classification paradox					
Python Results	Accuracy	94.73%	78.18%	78.09%	78.64%
	False Negative Rate	19.31%	25.52%	31.03%	23.45%
	False Positive Rate	3.14%	21.16%	20.52%	21.05%
R Results	Accuracy	86.23%	78.20%	78.82%	83.14%
	False Negative Rate	17.68%	28.65%	46.34%	26.27%
	False Positive Rate	13.11%	20.63%	16.92%	14.76%

Random Forest has the best accuracy and lowest false negative rate and false positive rate when used in R and Python both.

This was the basic approach so as to find which model works best with our preprocessed data. We can make it better by cross validating and tuning the parameters by applying **Hyperparameter** Tuning of the Random Forest. The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal.

4.2 Final Test Data Prediction

Above analysis and models have shown that Random Forest is the best fit model for our train dataset. Let's check the prediction accuracy on our final test data using similar approach:

We have 1667 observations and 21 columns in test.csv

Python Result:-

Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	1331	112
TRUE	37	187

AUC = 0.80

Classification paradox	**Value**
Accuracy	91.06%
True Negative Rate	92.24%
True Positive Rate / Recall	83.48%
False Negative Rate	16.52
False Positive Rate	7.76%

	precision	**Recall**	**f1-score**	**support**
FALSE	0.97	0.92	0.95	1443
TRUE	0.63	0.83	0.72	224
Avg/total	0.93	0.91	0.92	1667

R Result:-

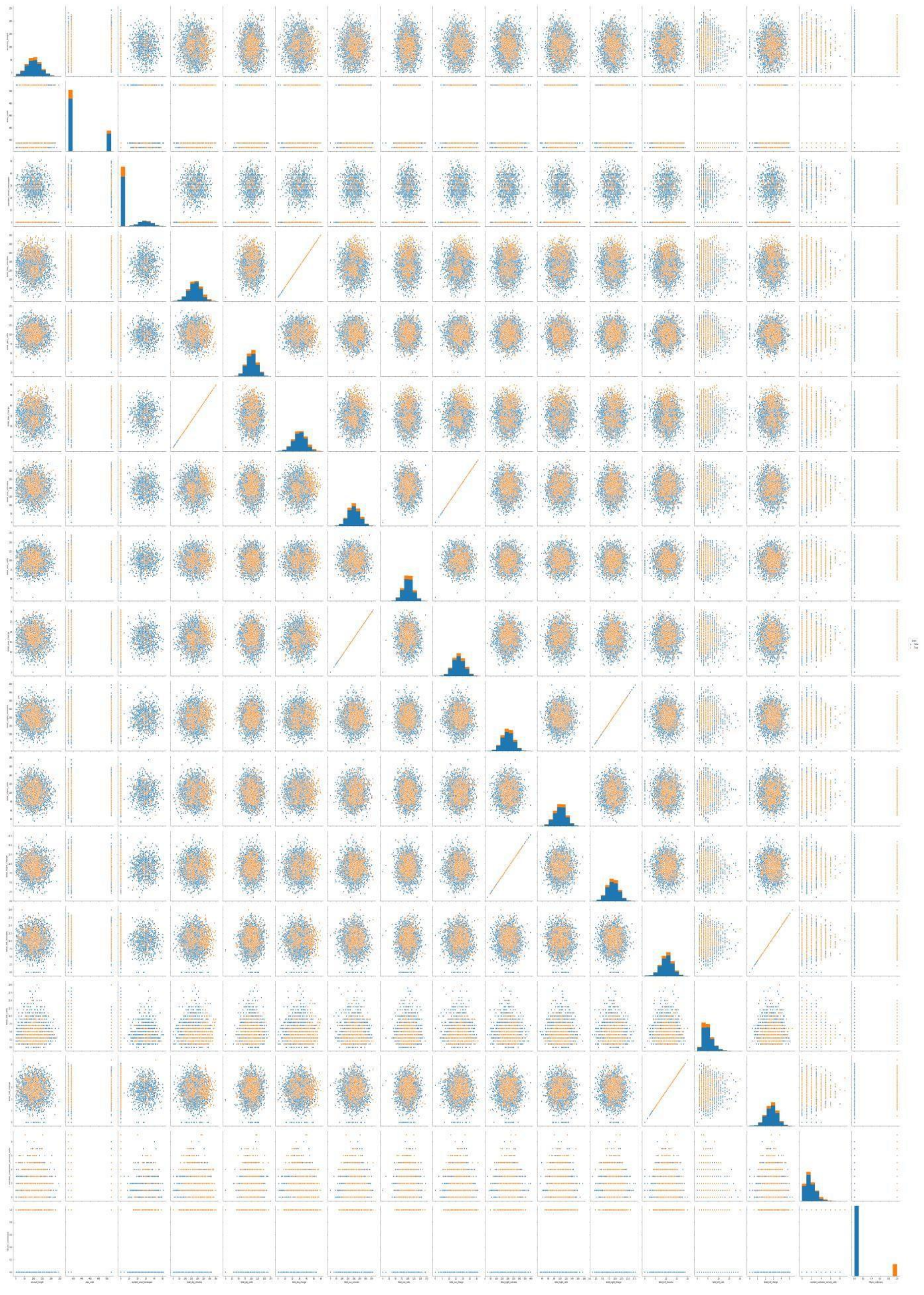
Confusion Matrix:---->>		
	FALSE	TRUE
FALSE	1241	202
TRUE	33	191

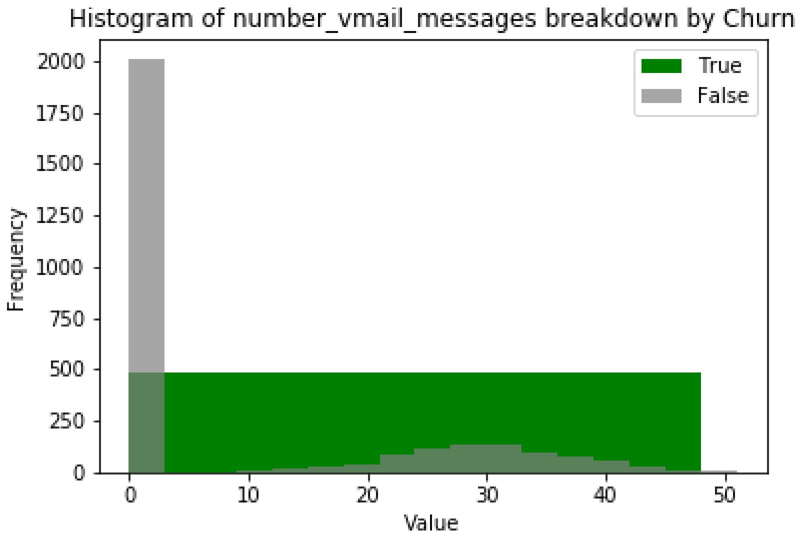
Parameter and value predicted
Accuracy : 0.859
95% CI : (0.8414, 0.8754)
No Information Rate : 0.7642
P-Value [Acc > NIR] : < 2.2e-16
Kappa : 0.5405
Mcnemar's Test P-Value : < 2.2e-16
Sensitivity : 0.9741
Specificity : 0.4860

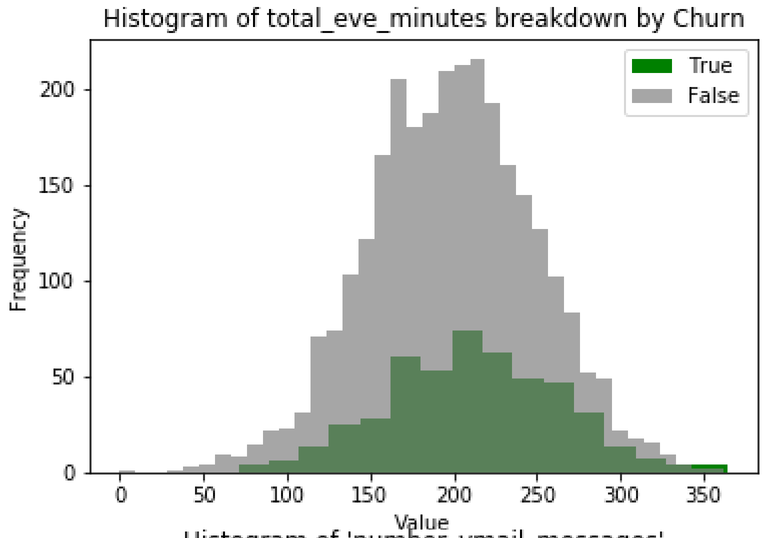
Pos Pred Value : 0.8600
Neg Pred Value : 0.8527
Prevalence : 0.7642
Detection Rate : 0.7445
Detection Prevalence : 0.8656
Balanced Accuracy : 0.7301
'Positive' Class : False.

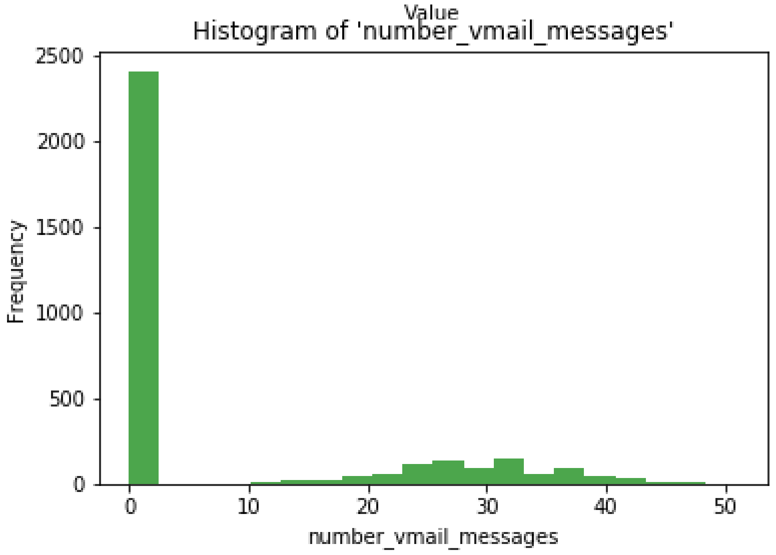
Appendix A: Extra Figures

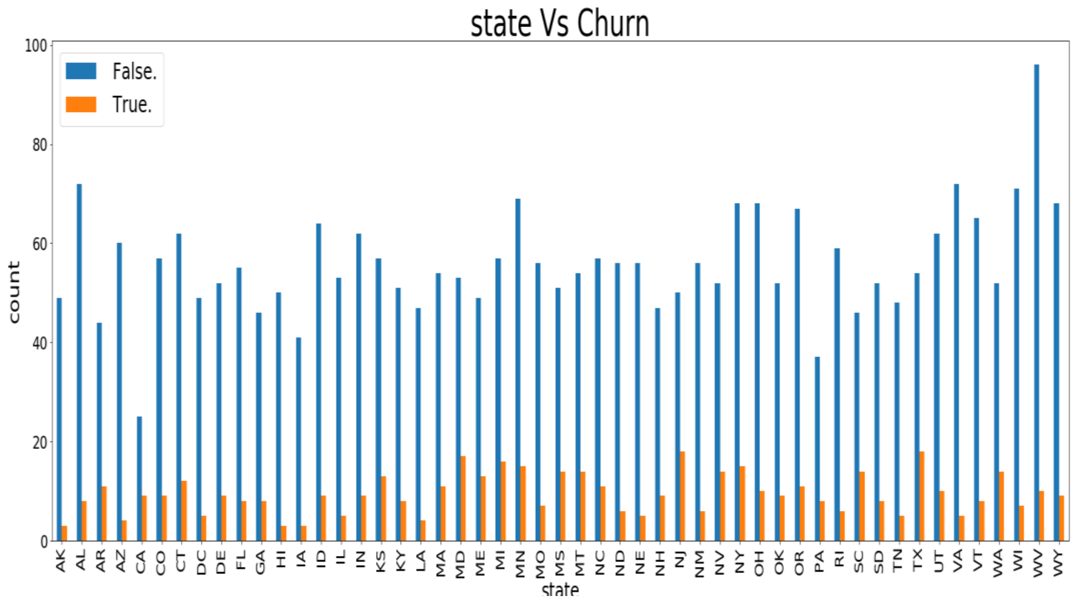
Pair Plot of Entire Test Data set

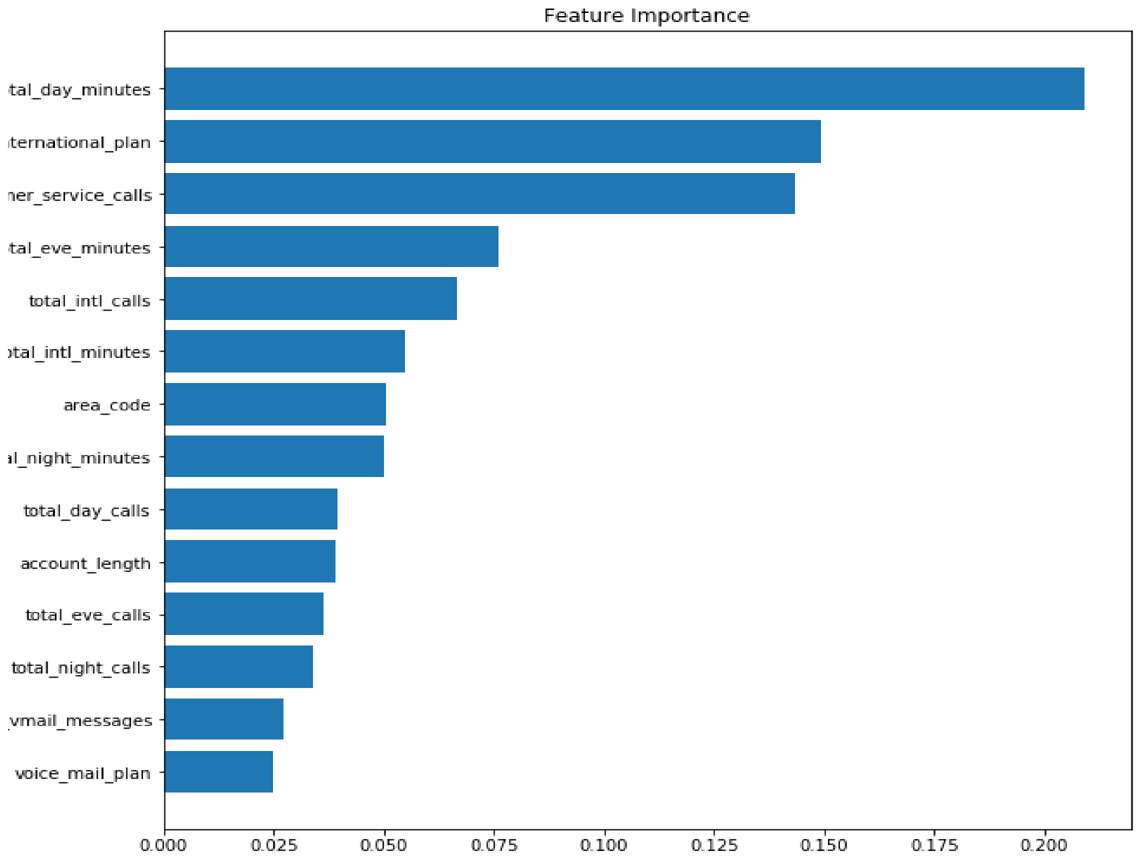


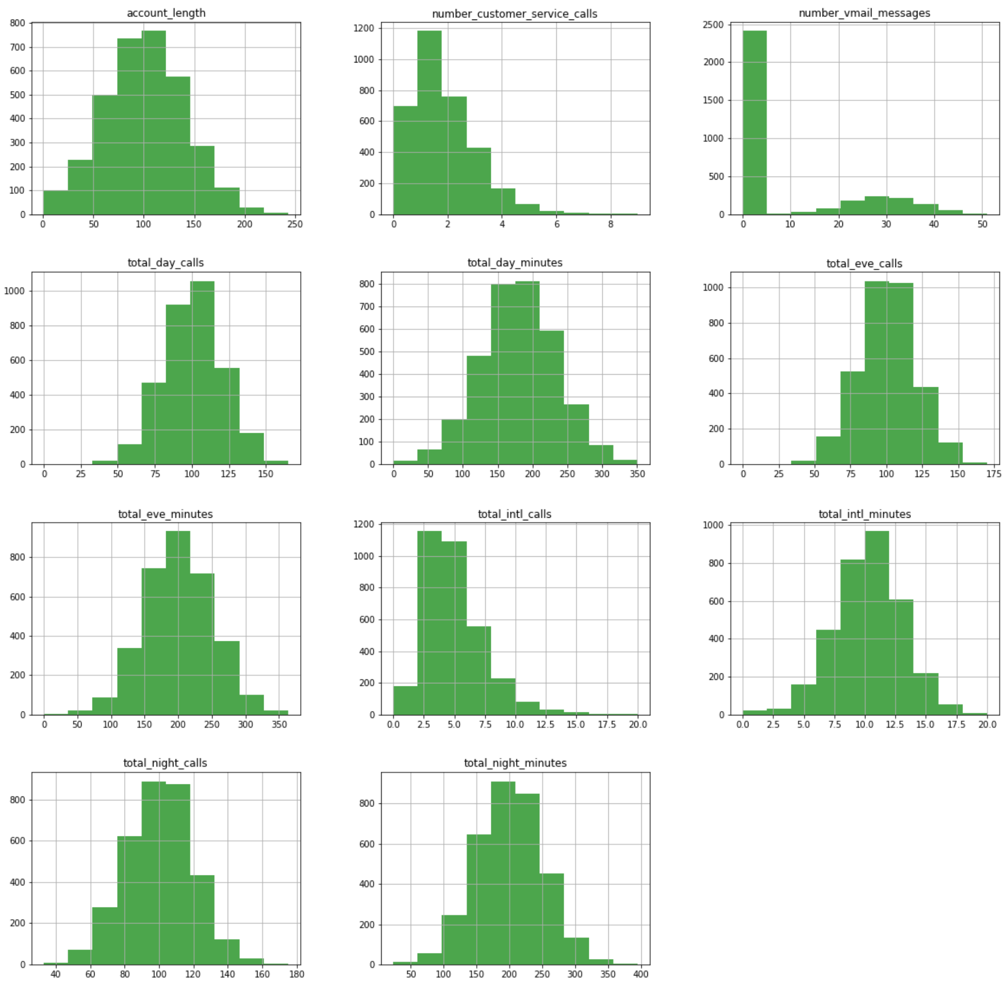

Histogram of number_vmail_messages breakdown by Churn
True
False
Frequency
Value
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50


Histogram of total_eve_minutes breakdown by Churn
True
False
Frequency
Value
0
50
100
150
200
0
50
100
150
200
250
300
350


Histogram of 'number_vmail_messages'
Frequency
number_vmail_messages
0
500
1000
1500
2000
2500
0
10
20
30
40
50


state Vs Churn
False.
True.
count
state
AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO MS MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY


Feature Importance
tal_day_minutes
ternational_plan
ner_service_calls
tal_eve_minutes
total_intl_calls
otal_intl_minutes
area_code
al_night_minutes
total_day_calls
account_length
total_eve_calls
total_night_calls
vmail_messages
voice_mail_plan


account_length
number_customer_service_calls
number_vmail_messages
total_day_calls
total_day_minutes
total_eve_calls
total_eve_minutes
total_intl_calls
total_intl_minutes
total_night_calls
total_night_minutes

Appendix B: Python Code

```
# coding: utf-8

# # Project Name: - Churn Reduction

# ### Project Description -

# Churn (loss of customers to competition) is a problem for companies because it is more expensive to acquire a new customer than to keep existing one from leaving. This problem statement is targeted at enabling churn reduction using analytics concepts. Our objective is to predict customer behavior. We are provided with a public dataset that has customer usage pattern and if the customer has moved or not. We are expected to develop an algorithm to predict the churn score based on usage pattern.

#

# ### Loading Libraries & Data

# In[1]:

# Importing Libraries

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split,RandomizedSearchCV

from imblearn.over_sampling import SMOTE

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

from sklearn.metrics import roc_curve,auc,roc_auc_score

get_ipython().run_line_magic('matplotlib', 'inline')

# In[2]:

#Setting working directory
```

```
os.chdir("C:/Users/Click/Desktop/project1_custchurn/")
print(os.getcwd())
#Loading Dataset
train_original = pd.read_csv("Train_data.csv")
test_original = pd.read_csv("Test_data.csv")
# In[3]:
#Creating Duplicate instances of data for Preprocessing and exploration
train = train_original.copy()
test = test_original.copy()
# ### Exploring Data
# In[4]:
train.head(5)
# In[5]:
#Checking info of data -> data types and rows n cols
train.info()
# In[6]:
train.describe()
# In[7]:
#calculating number of unique values for all df columns
train.nunique()
# In[8]:
#Replacing spaces from columns name with underscore
train.columns = train.columns.str.replace(" ","_")
test.columns = test.columns.str.replace(" ","_")
# In[9]:
#Veriying the change in column names
train.head()
# In[10]:
#Changing area_code type to categorical in both test and train data set
train['area_code'] = train['area_code'].astype('object')
```

```
test['area_code'] = test['area_code'].astype('object')

# In[11]:

#Droping phone_number column as it will not be of any help in our analysis

train = train.drop('phone_number',axis=1)

test = test.drop('phone_number',axis=1)

# In[12]:

#All continous var list

cname = train.columns[(train.dtypes=="float64")|(train.dtypes=="int64")].tolist()

print(cname)

#All categorical var after removing target var

cat_names = train.select_dtypes(exclude=np.number).columns.tolist()

cat_names.remove('Churn')

cat_names

# ### Checking Missing Values in Data

# In[13]:

#Checking missing values in train dataset

print(train.isnull().sum())

#result shows there are no missing values in the dataset

# In[14]:

#Checking missing values in test data set

print(test.isnull().sum())

#no missing value present in the test data

#  No missing values found in both train and test data sets

# ### Visualizing data

# In[15]:

#Target Variable data distribution

plt.figure(figsize=(8,6))

sns.countplot(x='Churn', data= train)

plt.xlabel('Customer churn', fontsize= 15)

plt.ylabel('Count', fontsize= 15)
plt.savefig("Churn_Vs_Count.png")
plt.title("Customer Churn Statistics",fontsize= 20)
# ###### We can see that there is a target class imbalance problem
# In[16]:
#Relational bar graph for checking data distribution with respect to target variable
def diff_bar(x,y):
    train.groupby([x,y]).size().unstack(level=-1).plot(kind='bar', figsize=(30,10))
    plt.xlabel(x,fontsize= 25)
    plt.ylabel('count',fontsize= 25)
    plt.legend(loc=0,fontsize= 25)
    plt.xticks(fontsize=20, rotation=90)
    plt.yticks(fontsize=20)
    plt.title("{X} Vs {Y}".format(X=x,Y=y),fontsize = 40)
    plt.savefig("{X}_Vs_{Y}.png".format(X=x,Y=y))
    plt.show()
# In[17]:
#State Wise Churning of customer
diff_bar('state','Churn')
# In[18]:
#area_code Wise Churning of customer
diff_bar('area_code','Churn')
# In[19]:
#International_Plan Wise Churning of customer
diff_bar('international_plan','Churn')
# In[20]:
#Number of Customer_Service Call Wise Churning of customer
diff_bar('number_customer_service_calls','Churn')
# In[21]:
#No. of Customer Churning and had a Voice mail plan
diff_bar('voice_mail_plan','Churn')
```

```
# In[22]:

sns.pairplot(train,hue='Churn',size=4.5)

# In[24]:

plt.savefig("pair_Plot.png".format())

# In[29]:

### Changing Categorical colum values to numeric codes

# In[25]:

#function for converting cat to num codes

def cat_to_num(df):

    for i in range(0, df.shape[1]):

        #print(i)

        if(df.iloc[:,i].dtypes == 'object'):

            df.iloc[:,i] = pd.Categorical(df.iloc[:,i])

            df.iloc[:,i] = df.iloc[:,i].cat.codes

            df.iloc[:,i] = df.iloc[:,i].astype('object')

    return df

# In[26]:

train = cat_to_num(train)

test = cat_to_num(test)


# ### Anomaly Detections or Outlier Analysis

#

# We  would be skipping outlier analysis as there is already a class imbalance impact over data.<br />

# Also, we will be using Random forest which can deal with outliers.

# ### Feature Selections

# In[27]:

#Setting up the pane or matrix size

f, ax = plt.subplots(figsize=(18,12))  #Width,height


#Generating Corelation Matrix
```

```
corr = train[cname].corr()

#Plot using Seaborn library

sns.heatmap(corr,mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220,10,
as_cmap=True),        square=True, ax=ax,annot=True,linewidths=1 , linecolor= 'black',vmin = -1, vmax =
1)

plt.show()

f.savefig('heatmap.png')


# #### Chi-Square for Categorical variables

# In[28]:

#checking Relation b/w categorical variables with respect to target var

from scipy.stats import chi2_contingency

for i in cat_names:

    print(i)

    #As we know imput to chi square is always a contiguency table so we generating it using crostab
function present in pd

    chi2, p, dof, ex =chi2_contingency(pd.crosstab(train['Churn'],train[i]))

    #as above pd.crosstab(dependent variable , independent variable)

    print(p)


#chi2 = Actual chi square test value

#p = pvalue

#dof = degree of freedom

#ex = expected value


#if p value is less than 0.05 then we will reject null hypothesis

#Null = both the variables are independent

#Alternate = Both the variables are not independent

# In[29]:

#Removing correlated variable & the variable which doesn't contain any meaningfull info

rmev = ['state','total_day_charge','total_eve_charge','total_night_charge','total_intl_charge']
```

```
train = train.drop(rmev,axis=1)

test = test.drop(rmev,axis=1)

# In[30]:

#Updating values after removal of var

cname = ['account_length', 'number_vmail_messages', 'total_day_minutes', 'total_day_calls',
'total_eve_minutes',

        'total_eve_calls', 'total_night_minutes', 'total_night_calls', 'total_intl_minutes', 'total_intl_calls',

        'number_customer_service_calls']


#All categorical var and removing target var

cat_names = ['area_code', 'international_plan', 'voice_mail_plan']


print('cname :- {}'.format(cname))

print()

print('cat_name :- {}'.format(cat_names))


# ###  Feature Scaling

# ##### Checking Distribution of data

# In[31]:

#Checking distribution of data via pandas visualization

train[cname].hist(figsize=(20,20),color='g',alpha = 0.7)

plt.savefig('distribution.png')

plt.show()

# In[32]:

# #Histogram breaks down by target variable

def plot_hist_y(x,y):

    plt.hist(list(x[y == 1]),color='green',label='True',bins='auto')

    plt.hist(list(x[y == 0]),color='grey', alpha = 0.7, label='False',bins='auto')

    plt.title("Histogram of {var} vs. by {Y}".format(var = x.name,Y=y.name))

    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.legend(loc="upper right")
    plt.savefig("Histogram of {var} vs. by {Y}.png".format(var = x.name,Y=y.name))
    plt.show()

# In[33]:
for i in cname:
    #print(i)
    plot_hist_y(train[i],train.Churn)

# ###### Most of the data is uniformally distributed , Hence Using data Standardization/Z-Score here
# ### Scalling
# In[34]:
def scale_standard(df):
    for i in cname:
        df[i] = (df[i] - df[i].mean())/df[i].std()
    return df

# In[35]:
#Standardizing Scale
train = scale_standard(train)
test = scale_standard(test)
# In[36]:
train.head()
# ### Sampling Data For Train and Test
# #### Stratified Sampling
# In[37]:
len(train)
# In[38]:
#Using train test split functionality for creating sampling
```

```
X = train.iloc[:,:14]
y = train.iloc[:,14]
y=y.astype('int')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=101)

# In[39]:
(X_train.shape),(y_train.shape)
# In[62]:
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))
# ### Using SMOTE   ( SMOTE: Synthetic Minority Over-sampling Technique)
# <br \>
# Due to target variable imbalance, it's good to over sample the minority class .

# In[40]:
# from imblearn.over_sampling import SMOTE
Smo = SMOTE(random_state=101)
X_train_res, y_train_res = Smo.fit_sample(X_train,y_train)

# In[41]:
(X_train_res.shape,y_train_res.shape)

# In[63]:
print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))
# In[61]:
train['Churn'].value_counts()
# ### Prediction function
# In[42]:
#Predicting & Stats Function
```

```
def pred(model_object,predictors,compare):
    """1.model_object = model name
       2.predictors = data to be predicted
       3.compare = y_train"""
    predicted = model_object.predict(predictors)
    # Determine the false positive and true positive rates
    fpr, tpr, _ = roc_curve(compare, model_object.predict_proba(predictors)[:,1])
    cm = pd.crosstab(compare,predicted)
    TN = cm.iloc[0,0]
    FN = cm.iloc[1,0]
    TP = cm.iloc[1,1]
    FP = cm.iloc[0,1]
    print("CONFUSION MATRIX ------->> ")
    print(cm)
    print()

    ##check accuracy of model
    print('Classification paradox :------->>')
    print('Accuracy :- ', round(((TP+TN)*100)/(TP+TN+FP+FN),2))
    print()
    print('True Negative Rate :- ',round((TN*100)/(TN+FP),2))
    print()
    print('True Positive Rate / Recall :- ',round((TP*100)/(FN+TP),2))
    print()
    print('False Negative Rate :- ',round((FN*100)/(FN+TP),2))
    print()
    print('False Positive Rate :- ',round((FP*100)/(FP+TN),2))
    print()
    print(classification_report(compare,predicted))
    print()
    # Calculate the AUC
    print ('AUC -: %0.2f' % auc(fpr, tpr))


#
#
# ### Model Level Approach
#
# #### Determinng which models fits good without optimization
#

# ### RandomForest

# In[114]:
#Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100,random_state=101).fit(X_train_res,y_train_res)
#Model Score on Valdation Data Set
pred(rf_model,X_test,y_test)

# Accuracy :-  94.73
# Specificity //  True Negative Rate :-  96.86
# Sensivity // True Positive Rate // Recall :-  80.69
# False Negative Rate :-  19.31
# False Positive Rate :-  3.14
# AUC -: 0.91

#
# ### Logistic Regression

# In[43]:
#logistic without binaries
```

```
logit_model = LogisticRegression(random_state=101).fit(X_train_res,y_train_res)

#Model Score on Valdation Data Set
pred(logit_model,X_test,y_test)

# Classification paradox :------->>
# Accuracy :-  78.18
# True Negative Rate :-  78.74
# True Positive Rate / Recall :-  74.48
# False Negative Rate :-  25.52
# False Positive Rate :-  21.26
#AUC -: 0.81

# ### KNN

# In[45]:
#KNN Model Development
KNN_Model = KNeighborsClassifier(n_neighbors=5).fit(X_train_res,y_train_res)

#Model Score on Valdation Data Set
pred(KNN_Model,X_test,y_test)

# Classification paradox :------->>
# Accuracy :-  78.09
# True Negative Rate :-  79.48
# True Positive Rate / Recall :-  68.97
# False Negative Rate :-  31.03
# False Positive Rate :-  20.52
# AUC = 0.80
# ### Navie Bayes


# In[44]:

#Navie Model Development

Naive_model = GaussianNB().fit(X_train_res,y_train_res)


#Model Score on Valdation Data Set

pred(Naive_model,X_test,y_test)


# Classification paradox :------->>

# Accuracy :-  78.64

# True Negative Rate :-  78.95

# True Positive Rate / Recall :-  76.55

# False Negative Rate :-  23.45

# False Positive Rate :-  21.05

# AUC = 0.82


# ### Final Model :- Random Forest

# Results show that random forest fits best for our dataset out of all the tested models


# In[46]:

# Training Final Model With Optimum Parameters

final_Model = RandomForestClassifier(random_state=101, n_estimators = 500,n_jobs=-1)

final_Model.fit(X_train_res,y_train_res)


# In[47]:

#Validating Predictions

pred(final_Model,X_test,y_test)

# ##### Features Importance
# In[48]:
#Calculating feature importances
importances = final_Model.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances
names = [train.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(10,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(X.shape[1]),importances[indices],align = 'center')
plt.yticks(range(X.shape[1]), names)
plt.show()
fig.savefig('feature_importance.png')

# #### AUC & ROC Curve
# In[49]:
#from sklearn.metrics import roc_curve,auc,roc_auc_score
# Determine the false positive and true positive rates
fpr, tpr, _ = roc_curve(y_test, final_Model.predict_proba(X_test)[:,1])
# Calculate the AUC
roc_auc = auc(fpr, tpr)
print ('ROC AUC: %0.2f' % roc_auc)

# Plot of a ROC curve for a specific class
```

```
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()


# ### Final Test Data Predictions
# In[50]:
# #Test Data Spliting parts target and Predictors
XX = test.iloc[:,:14].values  #predictors
yy = test.iloc[:,14].values   #target
yy=yy.astype('int')
# In[51]:
#Predicting test data
#pred(model_object=final_Model,predictors=XX,compare=yy)
Churn_Prediction = final_Model.predict(XX)
cm = pd.crosstab(yy,Churn_Prediction)
TN = cm.iloc[0,0]
FN = cm.iloc[1,0]
TP = cm.iloc[1,1]
FP = cm.iloc[0,1]
print("CONFUSION MATRIX ----->> ")
print(cm)
print()
##check accuracy of model
```

```
print('Accuracy :- ', round(((TP+TN)*100)/(TP+TN+FP+FN),2))
print('False Negative Rate :- ',round((FN*100)/(FN+TP),2))
print('False Positive Rate :- ',round((FP*100)/(FP+TN),2))

# In[52]:
print(classification_report(yy,Churn_Prediction))
# #### AUC & ROC over Test Data
# In[53]:
from sklearn.metrics import roc_curve,auc,roc_auc_score
# Determine the false positive and true positive rates
fpr, tpr, _ = roc_curve(yy, final_Model.predict_proba(XX)[:,1])
# Calculate the AUC
roc_auc = auc(fpr, tpr)
print ('ROC AUC: %0.2f' % roc_auc)

# Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
# #### Saving the OutPut
# In[54]:
#output
test_original['Churn Prediction'] = Churn_Prediction
```

```
test_original['Churn Prediction'] = test_original['Churn Prediction'].map({1 : 'True', 0 : 'False'})

#Predicted _Output

prob_output = pd.DataFrame(data=final_Model.predict_proba(XX),columns=("False_Probability","True_Probability"))

prob_output.head()

output = test_original[['state','area code','phone number','international plan','voice mail plan','Churn Prediction']]

# In[55]:

#Saving Result with Class

output.to_csv('Sample_Output.csv',sep='\t', encoding='utf-8')

#Saving with Class and Probabilities

output.join(prob_output).to_csv('Sample_Probable_Churn_output.csv',sep='\t', encoding='utf-8')

#del(output)

# In[56]:

#End of code
```

Appendix C: R Code-

```
rm(list = ls())

setwd("C:/Users/Click/Desktop/project1_custchurn")

getwd()

# #loading Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "e1071",

      "DataCombine", "pROC", "doSNOW", "class", "readxl","ROSE","dplyr", "plyr",
"reshape","xlsx","pbapply", "caret")


# #install.packages if not

lapply(x, install.packages)


# #load libraries

lapply(x, require, character.only = TRUE)

rm(x)



#Input Train & Test Data Source

train_Original = read.csv('Train_data.csv',header = T,na.strings = c(""," ","NA"))

test_Original = read.csv('Test_data.csv',header = T,na.strings = c(""," ","NA"))

#Creating backup of orginal data

train = train_Original

test = test_Original


###############################################################################

#                   EXPLORING DATA
               #

###############################################################################


#viewing the data

head(train,4)
```

```
dim(train)

#structure of data or data types
str(train)

#Summary of data
summary(train)

#unique value of each count
apply(train, 2,function(x) length(table(x)))

#Replacing the dot b/w collumn name to underscore for easy to use
names(train) <- gsub('\\.','_',names(train))
names(test) <- gsub('\\.','_',names(test))

#Converting area code as factor
train$area_code <- as.factor(train$area_code)
test$area_code <- as.factor(test$area_code)

#Removing phone number
train$phone_number <- NULL
test$phone_number <- NULL

#Let's see the percentage of our target variable
round(prop.table(table(train$Churn))*100,2)

# False.   True.
# 85.51   14.49
#Our target Class is suffering from target imbalance
```

```
#########################################################################
#         Checking Missing data
          #
#########################################################################
apply(train, 2, function(x) {sum(is.na(x))}) # in R, 1 = Row & 2 = Col
apply(test, 2, function(x) {sum(is.na(x))})

#No missing data found

#########################################################################
#                   Visualizing the data                          #
#########################################################################

#library(ggplot2)
#Target class distribution
ggplot(train,aes(Churn))+
  geom_bar(colour="black", fill = "blue") +  labs(y='Churn Count', title = 'Customer Churn Statistics')

# Churning of customer Vs. State
ggplot(train, aes(fill=Churn, x=state)) +
  geom_bar(position="dodge") + labs(title="Churning ~ State")

# Churning of customer Vs. Voice Mail Plan
ggplot(train, aes(fill=Churn, x=voice_mail_plan)) +
  geom_bar(position="dodge") + labs(title="Churning ~ Voice Mail Plan")

# Churning of customer Vs. international_plan
ggplot(train, aes(fill=Churn, x=international_plan)) +
  geom_bar(position="dodge") + labs(title="Churning ~ international_plan")
```

```
# Churning of customer Vs. area_code
ggplot(train, aes(fill=Churn, x=area_code)) +
  geom_bar(position="dodge") + labs(title="Churning ~ Area Code")

# Apply the factor function for converting Categorical to level -> factors
#train <- factor(train)
#test <- factor(test)



# Verifying that conversion was successful
print(is.factor(train))
print(is.factor(test))

#all numeric var
num_index = sapply(train, is.numeric)
num_data = train[,num_index]
num_col = colnames(num_data) #storing all the column name

#Checking for categorical features
cat_index = sapply(train,is.factor) #Fetching all the categorical index & later data
cat_data = train[,cat_index]
cat_col = colnames(cat_data)[-5]  #Removing target var

################################################################
#              Outlier Analysis                                #
################################################################

#  #We are skipping outliers analysis becoz we already have a Class Imbalance issue.

################################################################
```

```
#              Feature Selection                          #
################################################################

#Here we will use corrgram to find corelation

##Correlation plot
#library('corrgram')

corrgram(train[,num_index],
         order = F,  #we don't want to reorder
         upper.panel=panel.pie,
         lower.panel=panel.shade,
         text.panel=panel.txt,
         main = 'CORRELATION PLOT')

#We can see that the highly corr related vars in plot are marked in dark blue.
#Dark blue color means highly positive correlation

##------------------Chi Square Test--------------------------##

for(i in cat_col){
  print(names(cat_data[i]))
  print((chisq.test(table(cat_data$Churn,cat_data[,i])))[3])  #printing only pvalue
}

#-----------------Removing Highly Corelated and Independent var----------------------
train = subset(train,select= -c(state,total_day_charge,total_eve_charge,
                                total_night_charge,total_intl_charge))

test = subset(test,select= -c(state,total_day_charge,total_eve_charge,
                              total_night_charge,total_intl_charge))

##################################################################
#              Feature Scaling                                                          #
##################################################################

#all numeric var
num_index = sapply(train, is.numeric)
num_data = train[,num_index]
num_col = colnames(num_data) #storing all the column name



#Checking Data for Continuous Variables

################  Histogram   ##################
hist(train$total_day_calls)
hist(train$total_day_minutes)
hist(train$account_length)

#Most of the data is uniformly distributed
#Hence we shall be using data Standardization/Z-Score here

for(i in num_col){
  print(i)
  train[,i] = (train[,i] - mean(train[,i]))/sd(train[,i])
  test[,i] = (test[,i] - mean(test[,i]))/sd(test[,i])
}

##################################################################
#                              Sampling of Data                                          #
```

```
################################################################

# #Divide data into train and test using stratified sampling method

#install.packages('caret')
#library(caret)
set.seed(101)
split_index = createDataPartition(train$Churn, p = 0.66, list = FALSE)
trainset = train[split_index,]
validation_set  = train[-split_index,]

#Checking Train Set Target Class
table(trainset$Churn)
# 1    2
# 1881  319



# #Clearly our data has target class imbalance issue
# Synthetic Over Sampling the minority class & Under Sampling Majority Class can be applied to have a good Training Set
# #library(ROSE)  #---> Lib for Over and Under Sampling

trainset <- ROSE(Churn~.,data = trainset,p = 0.5,seed = 101)$data
table(trainset$Churn)
# 1 = 1101  2 = 1099

######################################################################################
########################################################################
# #           Basic approach for ML - Models
# # We will first get a basic idea of how different models perform on our preprocesed data and then select the best model and make it more efficient for our Dataset
```

```
##########################################################################################
############################################################################


# #Calculating FNR,FPR,Accuracy
calc <- function(cm){
  TN = cm[1,1]
  FP = cm[1,2]
  FN = cm[2,1]
  TP = cm[2,2]
  # #calculations
  print(paste0('Accuracy :- ',((TN+TP)/(TN+TP+FN+FP))*100))
  print(paste0('FNR :- ',((FN)/(TP+FN))*100))
  print(paste0('FPR :- ',((FP)/(TN+FP))*100))
  print(paste0('FPR :- ',((FP)/(TN+FP))*100))
  print(paste0('precision :-  ',((TP)/(TP+FP))*100))
  print(paste0('recall//TPR :-  ',((TP)/(TP+FP))*100))
  print(paste0('Sensitivity :-  ',((TP)/(TP+FN))*100))
  print(paste0('Specificity :-  ',((TN)/(TN+FP))*100))
  plot(cm)
}



### ##----------------------- Random Forest ----------------------- ## ###


#install.packages('randomForest')
#library('randomForest')
set.seed(101)
RF_model = randomForest(Churn ~ ., trainset,ntree= 500,importance=T,type='class')
plot(RF_model)
#Predict test data using random forest model
```

```
RF_Predictions = predict(RF_model, validation_set[,-15])

##Evaluate the performance of classification model
cm_RF = table(validation_set$Churn,RF_Predictions)
confusionMatrix(cm_RF)
calc(cm_RF)
plot(RF_model)

# Result of validaton-
# [1] "Accuracy :- 86.2312444836717"
# [1] "FNR :- 17.6829268292683"
# [1] "FPR :- 13.1062951496388"
# [1] "precision :-  51.5267175572519"
# [1] "recall//TPR :-  51.5267175572519"s
# [1] "Sensitivity :-  82.3170731707317"
# [1] "Specificity :-  86.8937048503612"

### ##----------------------- LOGISTIC REGRESSION ----------------------- ## ###
set.seed(101)
logit_model = glm(Churn ~., data = trainset, family =binomial(link="logit"))
summary(logit_model)
#Prediction
logit_pred = predict(logit_model,newdata = validation_set[,-15],type = 'response')

#Converting Prob to number or class
logit_pred = ifelse(logit_pred > 0.5, 2,1)
#logit_pred = as.factor(logit_pred)
##Evaluate the performance of classification model
cm_logit = table(validation_set$Churn, logit_pred)
confusionMatrix(cm_logit)
```

```
calc(cm_logit)

plot(logit_model)

#roc(validation_set$Churn~logit_pred)


# Result on validaton set

# [1] "Accuracy :- 78.1994704324801"

# [1] "FNR :- 28.6585365853659"

# [1] "FPR :- 20.6398348813209"

# [1] "precision :-  36.9085173501577"

# [1] "recall//TPR :-  36.9085173501577"

# [1] "Sensitivity :-  71.3414634146341"

# [1] "Specificity :-  79.360165118679"

# ROC = 0.8017


### ##------------------------ KNN ----------------------- ## ###

######I have commented the code as I was facing some issue while final runnning of the file. Approach to be followed is the one used in below code######


#set.seed(101)

#library(class)


##Predicting Test data

#knn_Pred = knn(train = trainset[,1:14],test = validation_set[,1:14],cl = trainset$Churn, k = 5,prob = T)


#Confusion matrix

#cm_knn = table(validation_set$Churn,knn_Pred)

#confusionMatrix(cm_knn)

#calc(cm_knn)


# Result on validaton set
# [1] "Accuracy :- 78.8172992056487"
# [1] "FNR :- 46.3414634146341"
# [1] "FPR :- 16.9246646026832"
# [1] "precision :-  34.9206349206349"
# [1] "recall//TPR :-  34.9206349206349"
# [1] "Sensitivity :-  53.6585365853659"
# [1] "Specificity :-  83.0753353973168"




### ##----------------------- Naive Bayes ----------------------- ## ###

# library(e1071) #lib for Naive bayes
set.seed(101)
#Model Development and Training
naive_model = naiveBayes(Churn ~., data = trainset, type = 'class')
#prediction
naive_pred = predict(naive_model,validation_set[,1:14])

#Confusion matrix
cm_naive = table(validation_set[,15],naive_pred)
confusionMatrix(cm_naive)
calc(cm_naive)

# Result on validaton set
# [1] "Accuracy :- 83.1421006178288"
# [1] "FNR :- 29.2682926829268"
# [1] "FPR :- 14.7574819401445"
# [1] "precision :-  44.7876447876448"
# [1] "recall//TPR :-  44.7876447876448"
# [1] "Sensitivity :-  70.7317073170732"

# [1] "Specificity :-  85.2425180598555"



##########################################################################################
######

#               Final Random Forest Model with tuning parameters
#

##########################################################################################
######


set.seed(101)

final_model = randomForest(Churn~.,data = trainset,ntree=800,mtry=4,importance=TRUE,type = 'class')

final_validation_pred = predict(final_model,validation_set[,-15])

cm_final_valid = table(validation_set[,15],final_validation_pred)

confusionMatrix(cm_final_valid)

calc(cm_final_valid)

#Result on validation set after parameter tuning

# [1] "Accuracy :- 86.4960282436011"

# [1] "FNR :- 17.0731707317073"

# [1] "FPR :- 12.8998968008256"

# [1] "FPR :- 12.8998968008256"

# [1] "precision :-  52.1072796934866"

# [1] "recall//TPR :-  52.1072796934866"

# [1] "Sensitivity :-  82.9268292682927"

# [1] "Specificity :-  87.1001031991744"



#Variable Importance

importance(final_model) #builtin function in Random forest lib

varImpPlot(final_model) #builtin func
```

```
#Plotting ROC curve and Calculate AUC metric

# library(pROC)

PredictionwithProb <-predict(final_model,validation_set[,-15],type = 'prob')

auc <- auc(validation_set$Churn,PredictionwithProb[,2])

auc

# # AUC = 89.47

plot(roc(validation_set$Churn,PredictionwithProb[,2]))


################################################################################

#        Final Prediction On test Data set
                                   #

################################################################################


#rmExcept(c("final_model","train","test","train_Original","test_Original","calc"))


set.seed(101)

final_test_pred = predict(final_model,test[,-15])

cm_final_test = table(test[,15],final_test_pred)

confusionMatrix(cm_final_test)

calc(cm_final_test)


# #Final Test Prediction

# [1] "Accuracy :- 85.9028194361128"

# [1] "FNR :- 15.625"

# [1] "FPR :- 13.8600138600139"

# [1] "precision :-  48.586118251928"

# [1] "recall//TPR :-  48.586118251928"

# [1] "Sensitivity :-  84.375"

# [1] "Specificity :-  86.1399861399861"
```

```
#Plotting ROC curve and Calculate AUC metric

# library(pROC)

finalPredictionwithProb <-predict(final_model,test[,-15],type = 'prob')

auc <- auc(test$Churn,finalPredictionwithProb[,2])

auc

# # AUC = 91.74

plot(roc(test$Churn,finalPredictionwithProb[,2]))


##########################################################################################
###

#                                               Saving output to file
                                       #

##########################################################################################
###


test_Original$predicted_output <- final_test_pred

test_Original$predicted_output <- gsub(1,"False",test_Original$predicted_output)

test_Original$predicted_output <- gsub(2,"True",test_Original$predicted_output)


#Entire Comparison

write.csv(test_Original,'C:/Users/Click/Desktop/project1_custchurn/Output_R.csv',row.names = F)


#Phonenumber and Churning class and probab

submit <- data.frame(test_Original$state,

                     test_Original$area.code,

                     test_Original$international.plan,

                     test_Original$voice.mail.plan,

                     test_Original$phone.number,

                     test_Original$predicted_output,

                     finalPredictionwithProb[,1],
                    finalPredictionwithProb[,2])

colnames(submit) <- c("State","Area Code","International Plan","Voice Mail Plan","Phone_Number",
                     "Predicted_Output","Probability_of_False","Probability_of_True")

write.csv(submit,file = 'C:/Users/Click/Desktop/project1_custchurn/FinalChurn_R.csv',row.names = F)
rm(list = ls())
```

References

- https://www.tutorialspoint.com/r/r_random_forest.htm
- http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/
- http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html
- http://www.cs.utexas.edu/~cannata/dataVis/Class%20Notes/Beautiful%20plotting%20in%20R_%20A%20ggplot2%20cheatsheet%20_%20Technical%20Tidbits%20From%20Spatial%20Analysis%20&%20Data%20Science.pdf
- https://www.rdocumentation.org/packages/base/versions/3.5.2/topics/prop.table
- https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e
- https://en.wikipedia.org/wiki/Hyperparameter_optimization
- https://www.hindawi.com/journals/complexity/2018/2520706/
- https://en.wikipedia.org/wiki/Feature_scaling
- https://ieeexplore.ieee.org/document/8126867/
- https://www.pro-face.com/otasuke/files/manual/gpproex/new/refer/mergedProjects/sampling/sampling_ov_sampling.htm
- https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/learn/v4/content