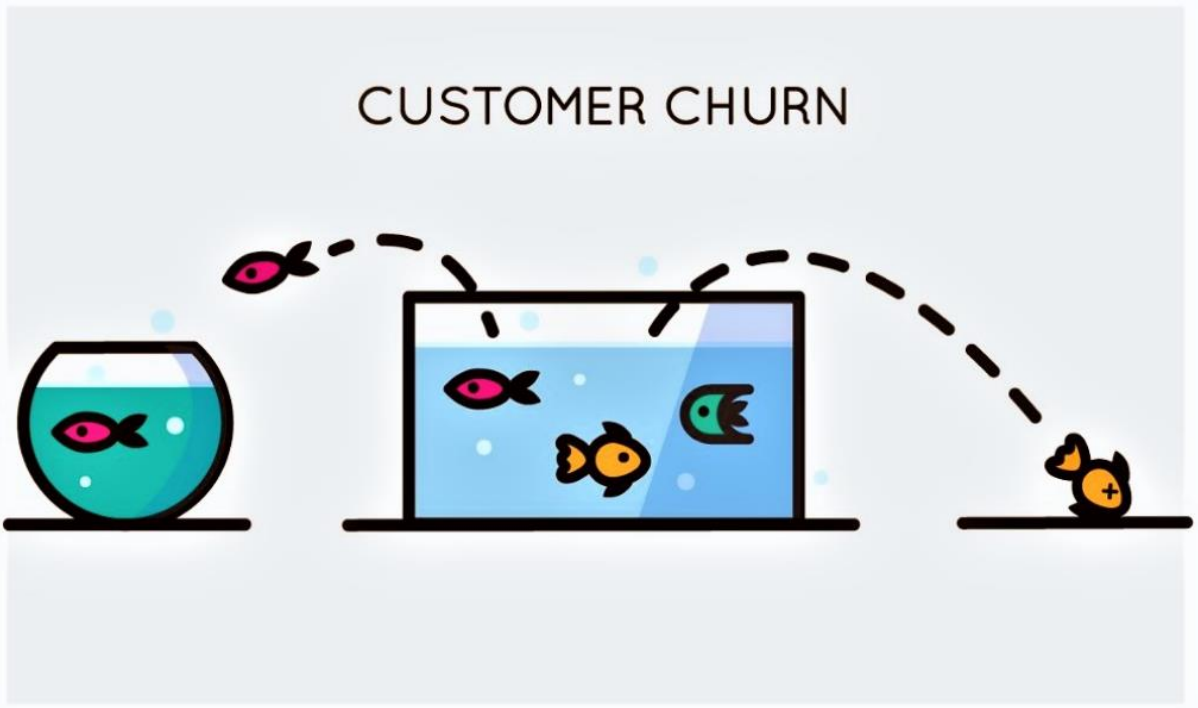
Project Report on Customer Churn Reduction

Submitted by

J Dheeraj Kumar



Submitted to

edWisor

Table of Contents

Chapter 3

CHAPTER 1

Introduction

1.1 Given Problem 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 your existing one from leaving. This problem statement is targeted at enabling churn reduction using analytics concepts.

1.2 Addressing The Problem Statement

The given problem description Domain belongs to 'Telecom Industry', as the term Churn describes that whether a particular subscriber/customer stays or moves out from that particular subscribed telecom services company. When a customer moves out from that subscribed company it is not only loss to the company but also difficult to retain the same or existing customer as the company spends lots of money on marketing and promotions for acquiring the present and future subscribing customers. So the churn rate in telecom industry services companies is usually high so here we are expected to develop a churn predictive model to retain the existing customer and future subscribing customers in advance. Which also helps to reduce the attrition rate of churners in the telecom industry.

1.3 Problem Level Approach

Thus the objective of the statement here addresses that we need to predict whether a customer has churned or not using both Python and R programming languages.

Here we need to develop the predictive model by the given customer usage pattern data by applying the necessary steps to develop the Data Mining / Machine Learning algorithms to predict the patterns of the customer usage and finalize which model development algorithm is going to be suitable for this problem description.

1.4 Understanding The Data

We are provided with public dataset which contains customer usage pattern data. The dataset assigned are **Train.csv** and **Test.csv** (Both are CSV files **Comma Separated Values**) dataset. The **Train data** has **3333** observations and **21** predictors/features and **Test data** has **1667** observations and **21** predictors/ features. The **Total** number of observations are **5000** and **21** features/predictors.

Table 1.1 – Description of Data

Predictors/Features	Description	Data-Type	Statistical Type
State	Ks = Kansas	String	Categorical
Account Length	Clients active status with the company	Numerical	Quantitative
Area Code	Phones area code	Numerical	Categorical
Phone Number	Customer's phone number	Numerical	Categorical
International Plan	Customer has opted or not ('yes/no')	String	Categorical/Binary
Voice Mail Plan	Customer has opted or not ('yes/no')	String	Categorical/Binary
Number v-mail messages	Number of voice mail messages	Numerical	Quantitative
Total day minutes	Total duration of daytime calls	Numerical	Quantitative
Total day calls	Total number of daytime calls	Numerical	Quantitative
Total day charge	Total charge for daytime services	Numerical	Quantitative
Total eve minutes	Total duration of evening calls	Numerical	Quantitative
Total eve calls	Total number of evening calls	Numerical	Quantitative
Total eve charge	Total charge for evening services	Numerical	Quantitative
Total night minutes	Total duration of night calls	Numerical	Quantitative
Total night calls	Total number of night calls	Numerical	Quantitative
Total night charge	Total charge for night services	Numerical	Quantitative
Total intl minutes	Total duration of international calls	Numerical	Quantitative
Total intl calls	Total number of international calls	Numerical	Quantitative
Total intl charge	Total charge for international services	Numerical	Quantitative
Number customer services calls	Number of calls customer service has made	Numerical	Quantitative
***Churn**	Target-If customer has moved then ('True./1 = Yes', 'False./0 = No')	String	Categorical/Binary

As per the given above data we are expected to develop an algorithm to predict the churn score based on the usage pattern provided. The last column here is **Churn** it has two labels ('False.' And 'True.') which is our **'Target variable'**. It is of **binary/ categorical** that is ***'True.'*** indicates that the customer has moved out from the subscribed company and ***'False.'*** Indicates that the customer was retained did not churn out.

Table: 1.2 – All Columns with top 5 observations of the dataset

state	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes
KS	415	382-4657	no	yes	25	265.1
OH	415	371-7191	no	yes	26	265.1
NJ	415	358-1921	no	no	0	243.4
OH	408	375-9999	yes	no	0	299.4
OK	415	330-6626	yes	no	0	166.7

total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls
110	45.07	197.4	99	16.78	244.7	91
123	27.47	195.5	103	16.62	254.4	103
114	41.38	121.2	110	10.30	162.6	104
71	50.90	61.9	88	5.26	196.9	89
113	28.34	148.3	122	12.61	186.9	121

total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn
11.01	10.0	3	2.70	1	False.
11.45	13.7	3	3.70	1	False.
7.32	12.2	5	3.29	0	False.
8.86	6.6	7	1.78	2	False.
8.41	10.1	3	2.73	3	False.

The Problem Category Falls Under

- Here the problem statement addresses that the problem category belongs to Supervised Learning and it belongs to Classification.
- Because we are trying to classify a customer or any dataset and here the given dataset has a Target Variable: Churn.
- So whenever you are trying to **"Predict the Categories"** or you are trying to **classify** into the **categories** that **"Use-Case"** comes under the **"Classification Problem".**

CHAPTER 2

Methodology

The methodology followed for the approach of the of developing the algorithm is as follows.

Step 1: - **(EDA) Exploratory Data Analysis:** Exploring the dataset firstly, analyzing for the missing values in the dataset, analyzing the data through visualization, Boxplot analysis for outliers check in the dataset, Feature Selection for continuous and categorical variables, Analyzing Chi2-Square test of Independence for categorical variables, Performing Dimension Reduction, Applying Feature Scaling for the dataset each of the approach applied will be explained in more detailed form in the further report.

Step 2: - **Model Development:** As the problem category falls under the classification models so all the classification algorithms like Decision Tree, Random Forest, Logistic Regression, KNN and Naïve Bayes have been developed.

Step 3: - **Optimization and Final Model Selection:** Hyper-parameter tuning and optimization of the outperformed model has been selected and finalized here.

2.1 Data Pre-processing (Exploratory Data Analysis)

Data pre-processing is the important steps while model development approach it is the phase where most of the time is spent nearly 70 % of the project space is occupied by data-pre-processing steps so each and every method applied here reflects on our model development stage as we all know there is a famous quote saying that **'garbage in is garbage out'** in the data science field.

Information about the dataset

```
Information about Training DataFrame including the Index data-type and
Column data-types
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
state                           3333 non-null object
account length                  3333 non-null int64
area code                       3333 non-null int64
phone number                    3333 non-null object
international plan              3333 non-null object
voice mail plan                 3333 non-null object
number vmail messages           3333 non-null int64
total day minutes               3333 non-null float64
total day calls                 3333 non-null int64
total day charge                3333 non-null float64
total eve minutes               3333 non-null float64
total eve calls                 3333 non-null int64
total eve charge                3333 non-null float64
total night minutes             3333 non-null float64
total night calls               3333 non-null int64
total night charge              3333 non-null float64
total intl minutes              3333 non-null float64
total intl calls                3333 non-null int64
total intl charge               3333 non-null float64
number customer service calls   3333 non-null int64
Churn                           3333 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
None
```

As we can observe that there here we have five categorical variables of the object data-type and the rest of them are numerical values we have 20 independent variables and 1 dependent variable.

Let's See the Descriptive Statistics of the Dataset

Table 2.1: Descriptive Statistics of columns from (1-5)

	account length	area code	Number vmail messages	total day minutes	total day calls
count	3333	3333	3333	3333	3333
mean	101.064806	437.182418	8.099010	179.775098	100.435644
std	39.822106	42.371290	13.688365	54.467389	20.069084
min	1.000000	408.000000	0.000000	0.0000000	0.0000000
25 %	74.000000	408.000000	0.000000	143.700000	87.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000
75%	127.000000	510.000000	20.000000	216.400000	114.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000

Table 2.2: Descriptive Statistics of columns from (6-10)

	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
count	3333	3333	3333	3333	3333
mean	30.562307	200.980348	100.114311	17.083540	200.872037
std	9.259435	50.713844	19.92265	4.310668	50.573847
min	0.000000	0.000000	0.000000	0.000000	23.200000
25 %	24.230000	166.600000	87.000000	14.160000	167.000000
50%	30.500000	201.400000	100.000000	17.120000	201.200000
75%	36.790000	235.300000	114.000000	20.000000	235.000000
max	59.640000	363.700000	170.00000	30.910000	395.000000

Table 2.3: Descriptive Statistics of columns from (10-15)

	total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls
count	3333	3333	3333	3333	3333
mean	9.039325	10.237294	4.479448	2.764581	1.562856
std	2.275873	2.791840	2.461214	0.753773	1.315491
min	1.040000	0.000000	0.000000	0.000000	0.000000
25 %	7.520000	8.500000	3.000000	2.300000	1.000000
50%	9.050000	10.300000	4.000000	2.780000	1.000000
75%	10.590000	12.100000	6.000000	3.270000	2.000000
max	17.770000	20.000000	20.000000	5.400000	9.000000

Let's analyze the data as we can see there are 15 numerical / continuous variables there are lots of zeros in the minimum it describes that most of our data is uniformly distributed.

2.1.1 Missing Values Analysis

The steps involved in data pre-processing is firstly we explore the data and its data-types and its basic information, checking the descriptive statistics of the dataset then next we do the missing values check in the dataset. Missing values in the data may occur due to numerous ways and the steps to be taken towards it is either we can ignore or impute missing values in the dataset.

We impute the variables missing values whose data is less than (< 30%) only or else drop the variables whose percent is greater than 30 %, if we try to impute the data which has greater than thirty percent may lead us to **'false predictions'** in the model development. The best way to understand each value is missing is by plotting a bar graph. The methods involved in imputing missing values are {'Fill in with Central Statistics': (Mean, Median, Mode), 'Distance based or Data Mining Method': (KNN Imputation Method), 'Prediction Method'}. We are lucky here because our dataset does contain any missing values and we don't need to apply these steps. This approach is same in both Python and R.

Table: 2.4 – Missing Values Data Check in Train & Test dataset

Predictors/Features	Missing Value
State	0
Account Length	0
Area Code	0
Phone number	0
International Plan	0
Voice Mail Plan	0
Number v-mail 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 services calls	0

Let's check the unique values in our dataset

By using a for loop for checking the unique values in the data

Table: 2.5 – Unique values and its count in every variable

Predictors/Features	Unique Values
State	51
Account Length	212
Area Code	3
Phone Number	3333
International Plan	2
Voice Mail Plan	2
Number v-mail 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 services calls	10
Churn	2

As we can observe that most the variables have different count of unique variables except phone-number it has 3333 unique values and state has 51 unique values and is categorical variable if we convert and assign it to dummy variables the columns will increase and will lead us to curse of dimensionality so it is better to drop both state and as well as phone number in dimension reduction stage.

Let's check the churn count and its percentage rate in the dataset

```
False.    2850
 True.     483
Name: Churn, dtype: int64

**************************************
 False.    0.855086
 True.     0.144914
Name: Churn, dtype: float64

**************************************
```

The target class of churn variable is imbalanced there is less percent of observations on the positive class that is 14.49 % when compared with the negative class is 85.50 % so we need to address the target class imbalance by using necessary methods and which will prevent us from wrong predictions of our predictive model.

2.1.2 Analyzing Data through Visualization

- Let's see the churning of customers through bar-plot.

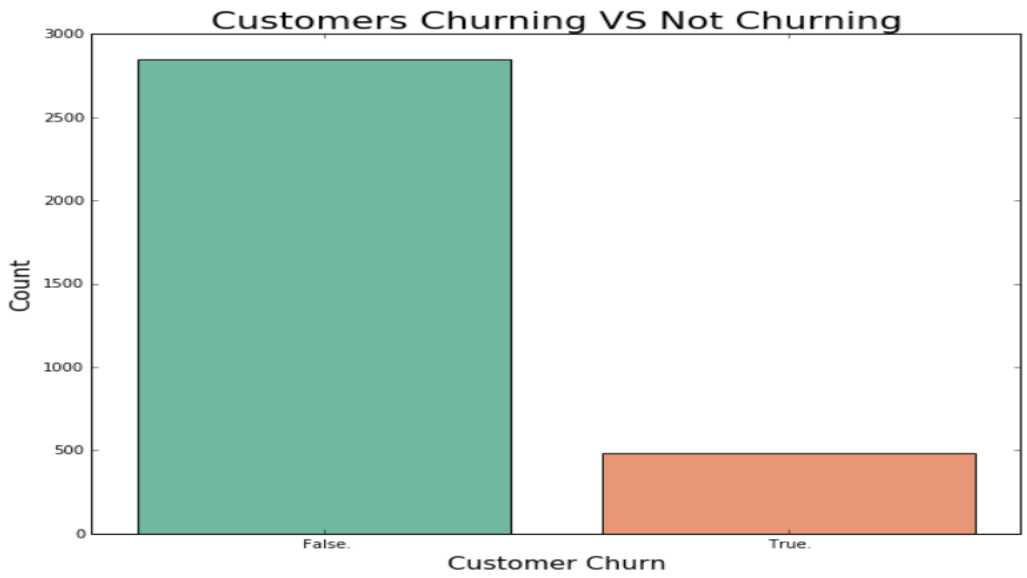


Fig 1.1 – Churning of customers with the target variable 'Churn'

The churning of customers in the training data on the **negative class** that is **'False'** is about **2850 count** which is nearly **85.50 %** of customer's not churning VS **483** on the **positive class** that is **'True'** case scenario is nearly **14.49 %** of churn rate of customers positively churning which is where there is a huge target class imbalance here and we need to address it.

- Let's see the customers churning according to area code wise through bar plot.

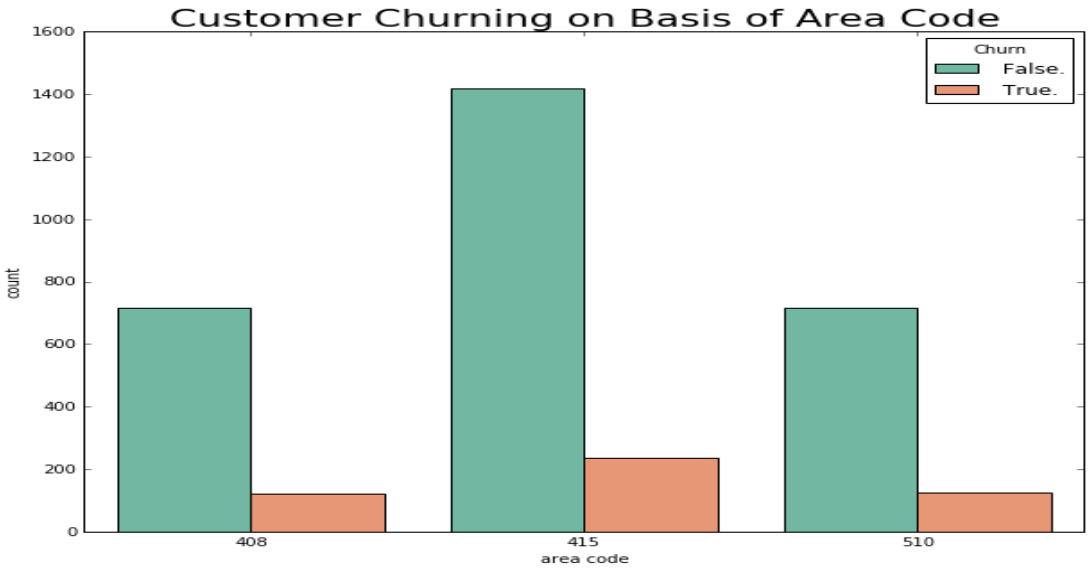


Fig 1.2 – Churning of customer's area code wise

There is a clear indication that most of the customers belonging from **'area-code 415'** have churned out. The count of churn in area-code's {**'408'** is 838, **'415'** is 1655, **'510'** is 840}.

- Let’s see the customers churning according to number customer service calls.

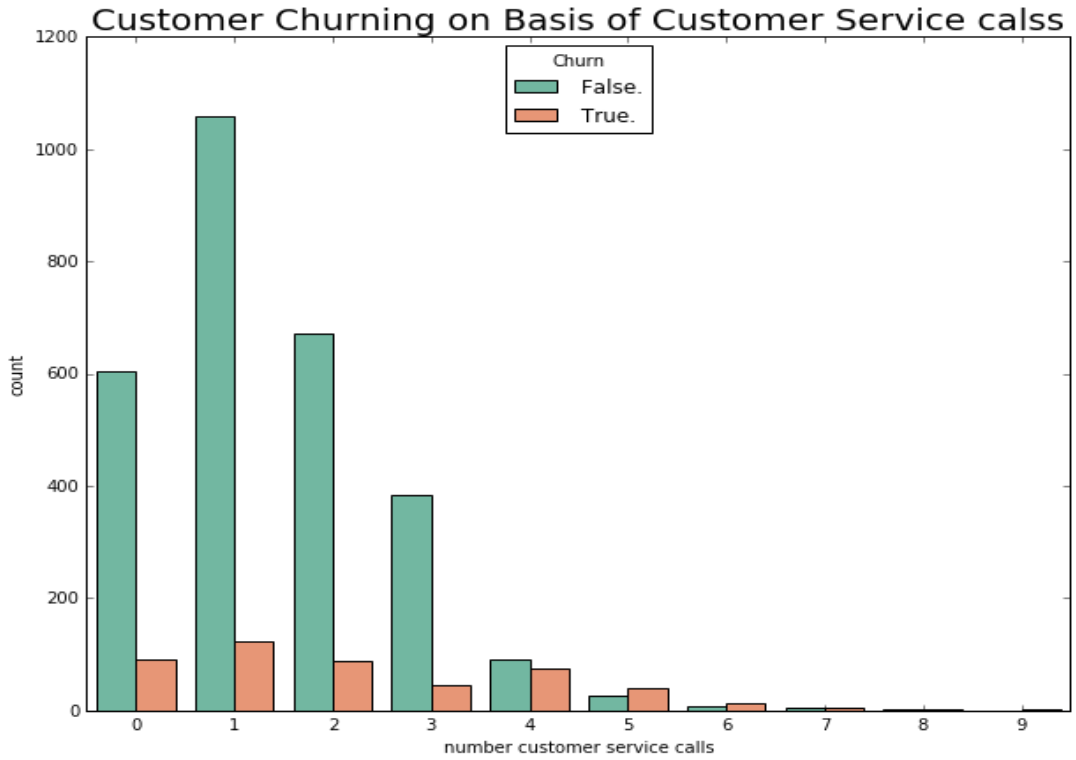


Fig 1.3 – Churning of customers on basis of customer service calls

The churn count due to number of customer service calls is nearly **20.68 %** approximately.

- Let’s see the churning of customers through international and voice mail plan wise.

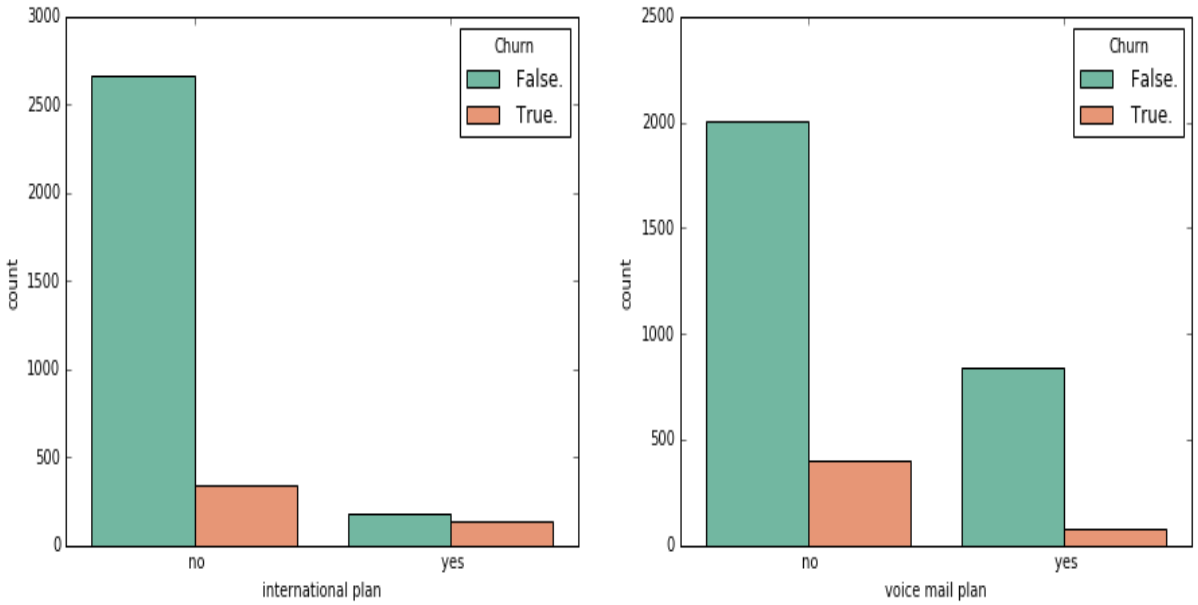


Fig 1.4 – Churning of customers on basis of international & voice mail plan

Customers Subscribing to International and Voice mail plan Figures

The customers count is **3010** who have **not subscribed** to **international plan** and its percentage is **90.39 %** and the count **323** of customers who have **subscribed** to the **international plan** and its percentage is **9.69 %**

The customers count is **2411** who have **not subscribed** to **voice mail plan** and its percentage is **72.33 %** and the count of customers who have **subscribed** to **the voice mail plan** is **922** and its percentage is **27.66 %** of customers who have **subscribed** to voice mail plan.

Customers Churn Percent accordingly when they have Subscribed or they have not Subscribed to International and Voice mail plan Figures

Analysis of international plan wise:

The customers not having subscribed to international plan and its count is **2664** customers have **not subscribed** to the international plan and its percentage is **88.50 %** and out of **2664** customers **346** customers have **churned** even though they have **not subscribed** to international plan and its percentage is **11.49 %.**

The customers having subscribed to international plan and its count is **186** customers have **subscribed** to the international plan and its percentage is **57.58 %** and out of **186** customers **137** customers have **churned** after **subscribing** to international plan and its percentage is **42.41 %.**

Analysis of voice mail plan wise:

The customers not having subscribed to voice mail plan and its count is **2008** customers have **not subscribed** to the voice mail plan and its percentage is **83.28 %** and out of **2008** customers **403** customers have **churned** even though they have **not subscribed** to voice mail plan and its percentage is **16.71 %.**

The customers having subscribed to voice mail plan and its count is **842** customers have **subscribed** to the voice mail plan and its percentage is **91.32 %** and out of **842** customers **80** customers have **churned** after **subscribing** to voice mail plan and its percentage is **8.67 %.**

- Let’s see the linearity relationship between total- international and day according to minutes and charge.

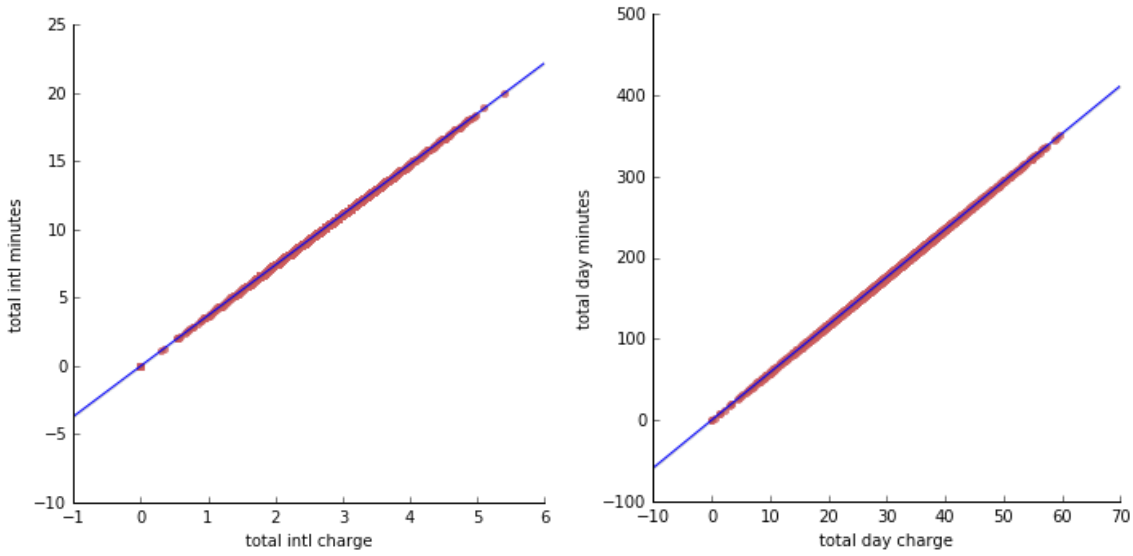


Fig 1.5 – Linearity relationship and Highly correlation between features

- Let’s see the linearity relationship between total- evening and night according to minutes and charge.

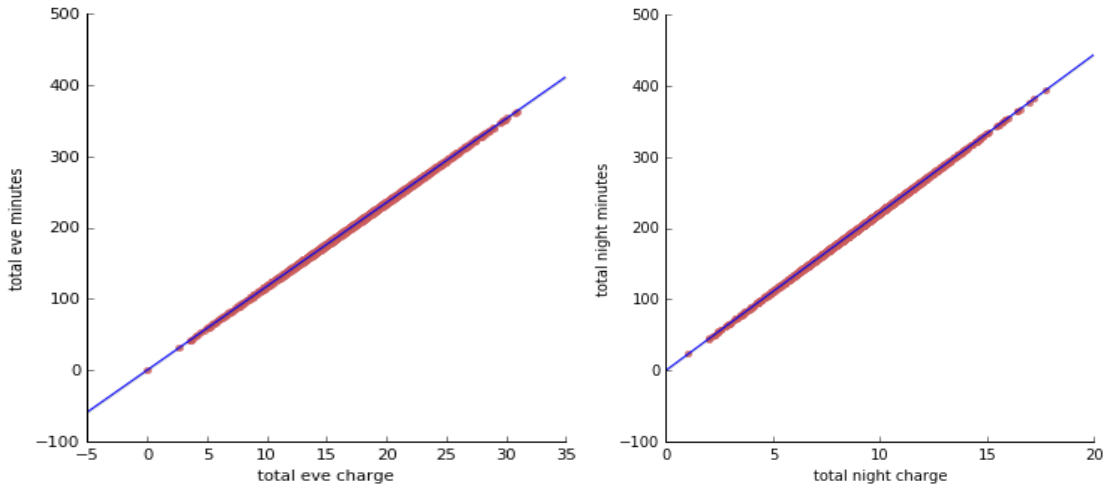


Fig 1.6 – Linearity relationship and Highly correlation between features

Total international, day, evening and night- minutes variables are highly correlated with Total international, day, evening and night- charge variables so multi-collinearity exists here and we need to drop either of the four variables in the dimension reduction stage.

2.1.3 Outlier Analysis

Using Boxplot Analysis for Outliers check

An outlier is nothing but which is **“inconsistent”** with the rest of the dataset. Observations inconsistent with the rest of the global outlier, in simple terms any value which is falling away from a bunch of values is nothing but an outlier. Outlier will be available only in continuous variable. This approach is same in both Python and R.

- Let’s see the Boxplot visualization of outlier’s detection which shows in our dataset.

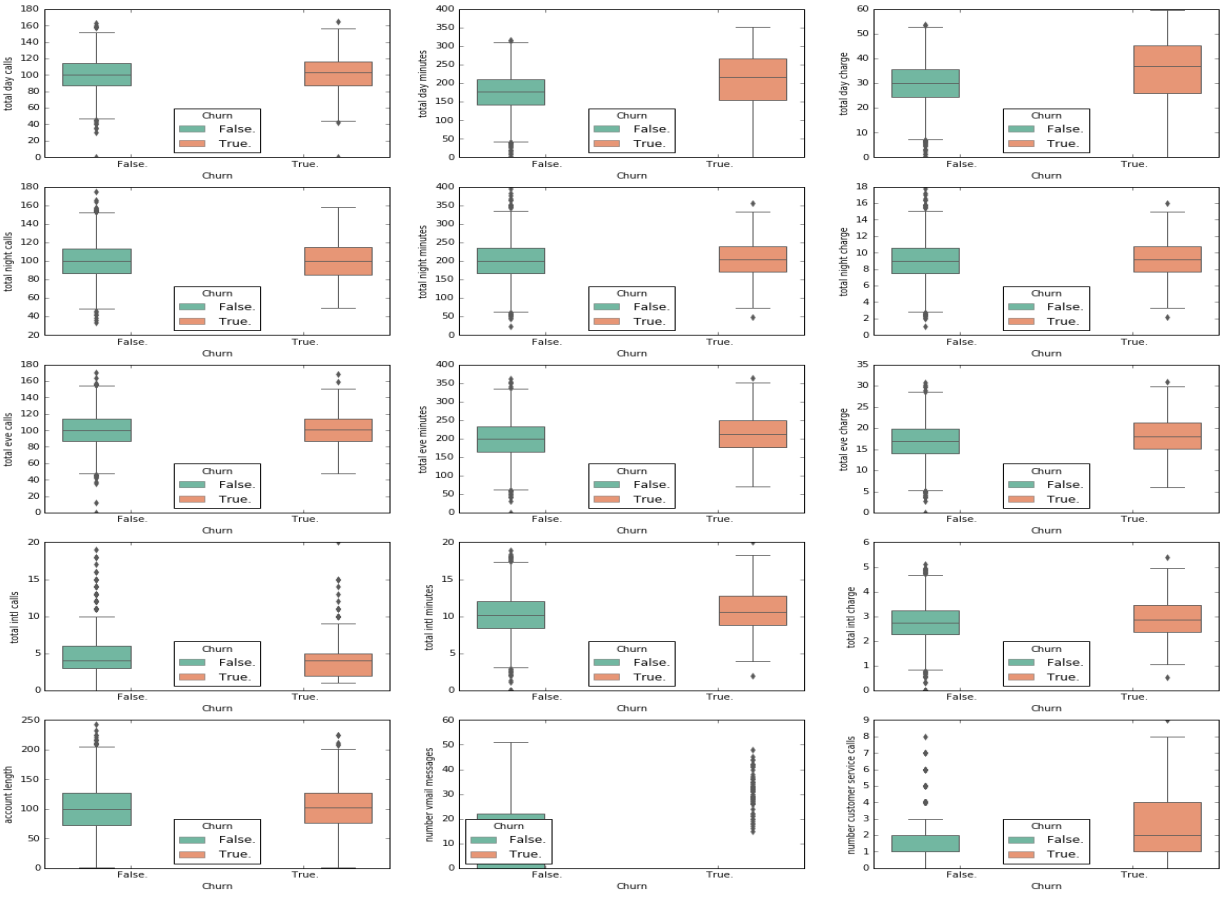


Fig 1.7 – Boxplot for outlier detection check

As we can see that almost all features/predictors contain outliers and we will try to remove the outliers from the combined **‘data’** dataset and not disturbing the train dataset as we are anyhow going to not apply the boxplot step for train data as we may lose some of the important information but here we will remove the outliers from the combined dataset that is **'data'** dataset which we have combined both train and test dataset so that if we remove the outliers and then do KNN imputation still the data will be reduced and already we have only 5000 records data for analysis which is small so here we will just apply Outlier Analysis but will not use this dataset for further predictions.

- Let’s see the Boxplot visualization after dropping outliers from the dataset.

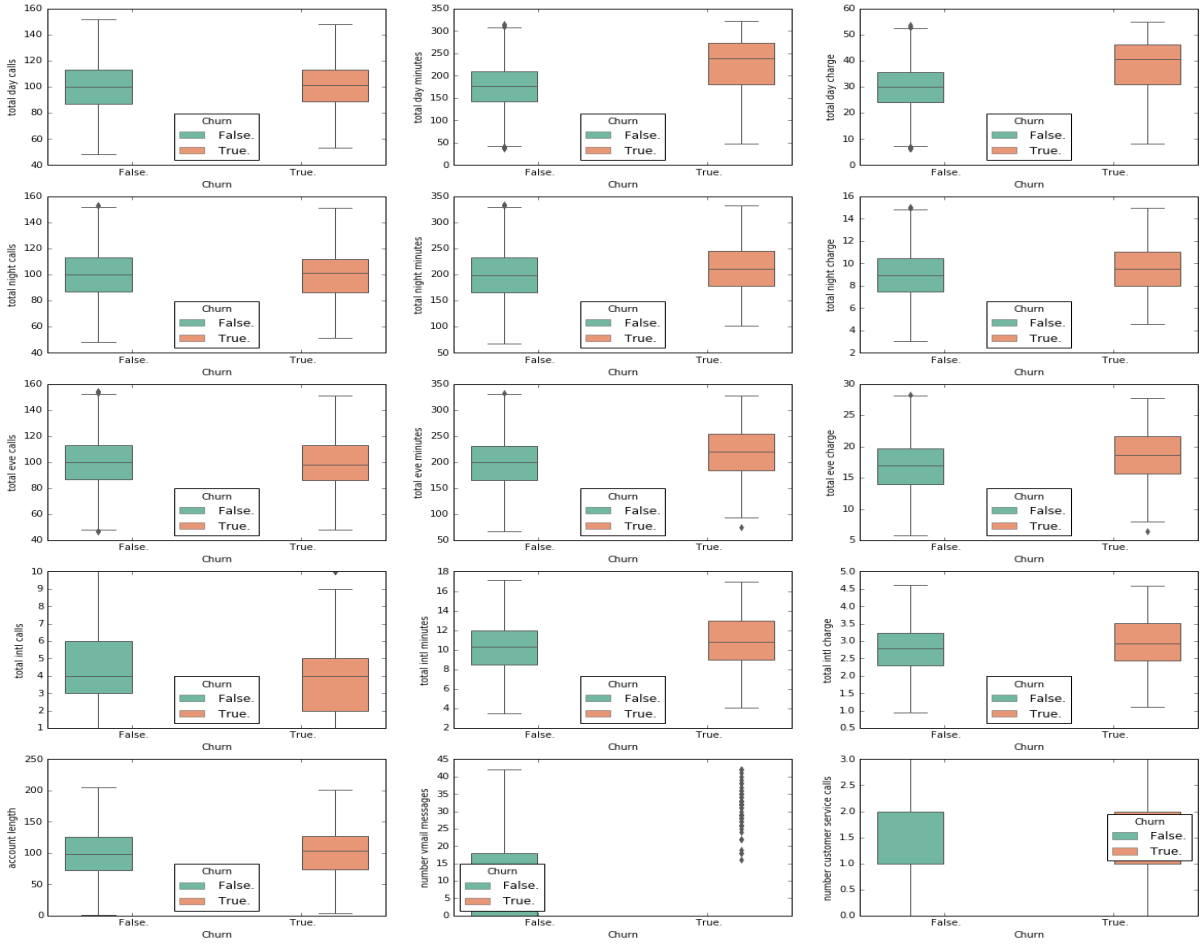


Fig 1.8 – Boxplot after dropping outliers

Now we can observe that most of the outliers have been dropped except from the number vmail messages as there may be many number voice mail messages and is not considered as an outlier so this way we can understand that boxplot is dropping some of the necessary and important information se here we are skipping the boxplot analysis step.

2.1.4 Feature Selection

- **Correlation analysis for continuous features**
- **Chi-2 square test of independence for categorical features**

Correlation analysis

Extracting a relevant and meaningful features out of a dataset also known as **“feature engineering or feature extraction”** feature selection means you are selecting a subset of features out of all variables present in the dataset.

So the agenda of feature selection is to identify and to remove irrelevant features from the dataset which do not contribute much information to explain the variance of the target variable.

Because the fewer the features are desirable it reduces the complexity of the model and simpler model is simpler to understand and explain. This approach is same in both Python and R.

- Let's see the Correlation matrix of the dataset.

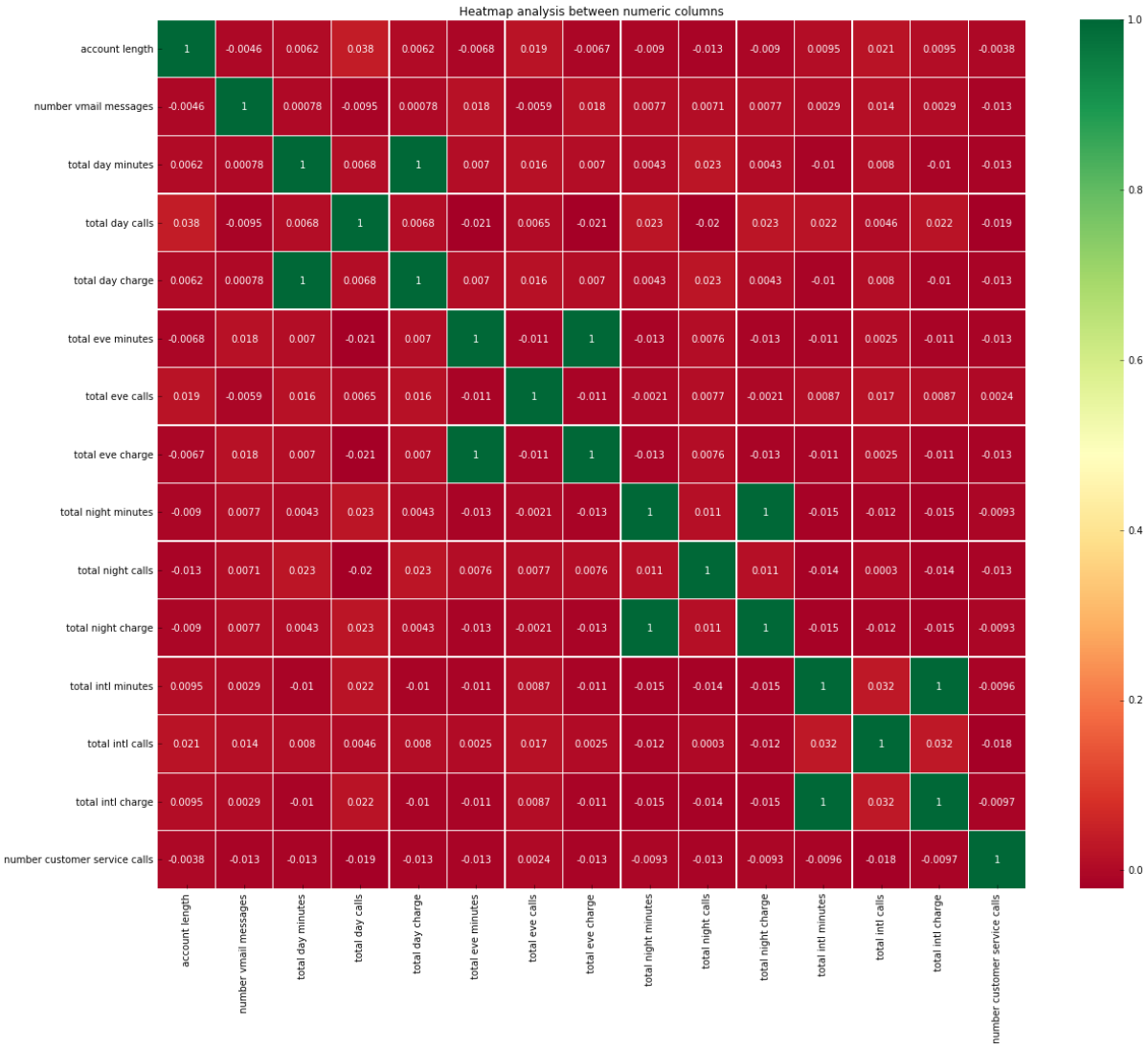


Fig 1.9 – Correlation analysis to check highly positively correlated features

Correlation is only applicable in numerical/continuous variables. The correlation matrix clearly describes that these features total international, day, evening and night- minutes variables are highly positively correlated with total international, day, evening and night-charge variables so multi-collinearity exists here and we need to drop either of the four variables in the dimension reduction stage. Correlation analysis helps us to get rid of irrelevant and redundant features in our dataset

Chi-2 Test of Independence

If we have two categorical features in that case chi-2 square test is the best suitable statistical test to measure the dependency between two categorical features.

Here the dependency between the independent and dependent features should be high and if you are measuring the dependency between the independent features then it should be very low. This approach is same in both Python and R.

Before running chi-2 square test of independence we set the **"Hypothesis"** that is: -

Null Hypothesis: - Two features are independent with each other.

Alternate Hypothesis: - Two features are not independent with each other.

Which means that there is some dependency between the two features.

Using the chi-2 square **p-value = 0.05** we will decide to keep which features and drop the features if the p- values of it is greater than **> 0.05.**

- Let's see the results after we ran chi-2 square test on the dataset.

```
Chi2-Square Test of Independence

******************************
state
0.00229622155201
----------------------------
area code
0.915055696024 ------------------→ area code's values is 0.91 > 0.05
----------------------------
international plan
2.49310770332e-50
----------------------------
voice mail plan
5.15063965904e-09
----------------------------
```

As we can see that area code value is greater than the chi-2 square p – values so we are going to drop area code feature in dimension reduction stage.

Dimension Reduction

Dimension reduction stage we are going to drop the features which are irrelevant or redundant or not contributing important information or carrying to explain our target label. The features which we are going to drop here are ['state', 'area code', 'phone number', 'total day charge', 'total eve charge', 'total night charge', 'total intl charge'] we have dropped the features in both train and test datasets. This approach is same in both Python and R.

Assigning levels to categorical features

In this step we are changing the categorical column features to numerical features by assigning levels such as 0,1,2 for the columns because the models accept integer / float data-types they cannot interpret the categorical, string and object data types that why we are assigning levels to them in this step.

We are assigning levels to categorical features for [‘international plan’, ‘voice mail plan’, ‘Churn’] feature columns. This approach is same in both Python and R.

2.1.5 Feature Scaling

It involves in dealing with the parameters of different units and scale. Feature scaling is a method to **“limit the range of variables”**, so that they can be compared on the common ground and it is performed only on the **“continuous variables”**.

We are using “Standardization/Z-score” method for scaling of the data here, the standardization scaling transforms the data to **‘0’ – mean** and the **‘unit variance’**.

It will convert each data point to a unit of standard deviation. The data may range from negative to positive on the scale. But the output of the z-score will always tell you that how many standard deviations away the data point is from the mean.

- Let’s see the distribution of data with histogram and density plot combined.

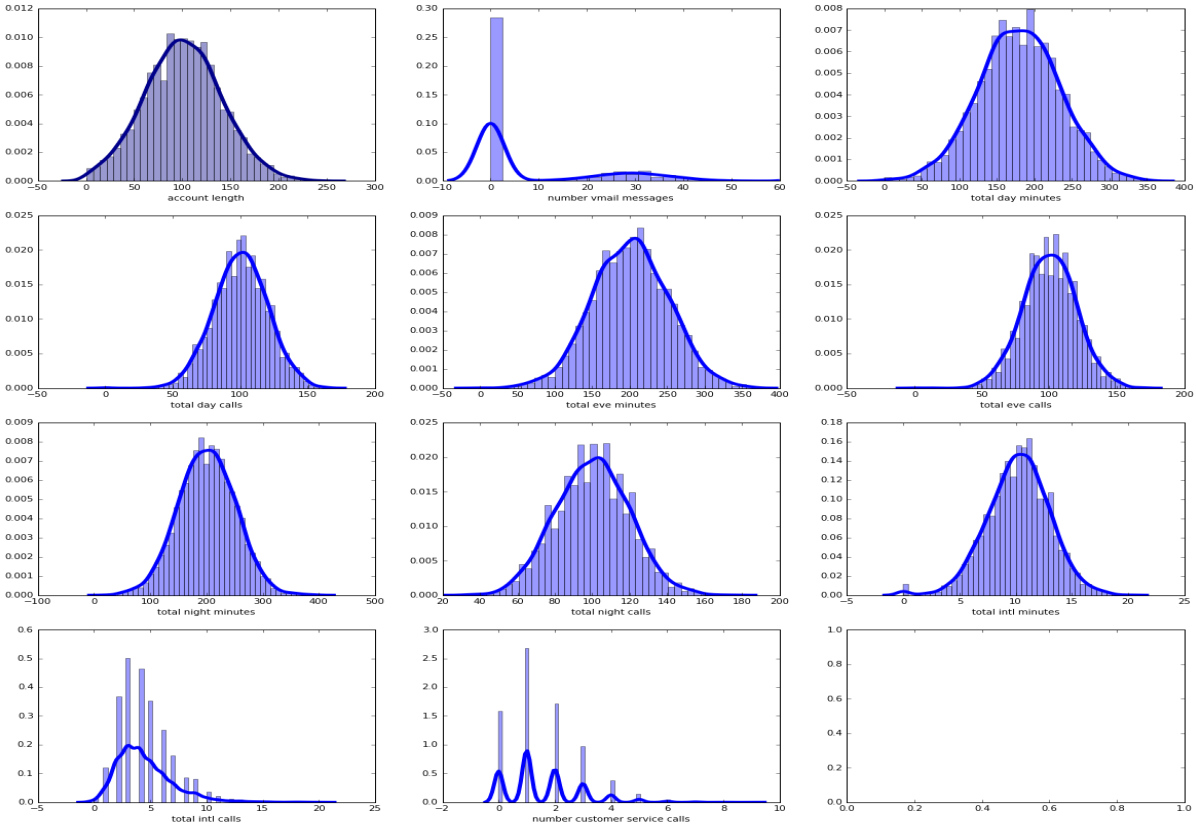


Fig 2 – Normality check using histogram and density plot

We can clearly observe that most of the data is uniformly distributed so we can go for standardization/ z-score method for feature scaling of the data. This approach is same in both Python and R.

Data After Applying Standardization / Z- Score Method

	account length	international plan	voice mail plan	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	Churn
0	0.676388	0	1	1.234697	1.566532	0.476572	-0.070599	-0.055932	0.866613	-0.465425	-0.084995	-0.601105	-0.427868	0
1	0.149043	0	1	1.307752	-0.333688	1.124334	-0.108064	0.144845	1.058412	0.147802	1.240296	-0.601105	-0.427868	0
2	0.902393	0	0	-0.591671	1.168128	0.675883	-1.573147	0.496204	-0.756756	0.198905	0.703015	0.211502	-1.188040	0
3	-0.428526	1	0	-0.591671	2.196267	-1.466716	-2.742453	-0.608068	-0.078539	-0.567629	-1.302831	1.024109	0.332305	0
4	-0.654531	1	0	-0.591671	-0.240054	0.626055	-1.038776	1.098534	-0.276270	1.067643	-0.049177	-0.601105	1.092477	0

Fig 2.1 – Data after applying feature scaling

2.1.5 Sampling Data

Here I had **split** the **data** directly into the **train** and **test sets** before itself in **python** where as in **R** using **stratified** sampling method has been used for splitting the data. Using the CreateDataPartition () is a function to create stratified samples and here taking the split ratio of 70 % into train and 30 % into test has been done in R.

Data after splitting in python

```
X_train values------> (3333, 13)
y_train values------> (3333,)
X_test values-------> (1667, 13)
y_test values-------> (1667,)
```

let's see the target value churn in python before smote

```
0    2850
1     483
Name: Churn, dtype: int64
```

let's see the target value churn in R before rose

```
# 0   1

# 1995 339
```

Here **Churn False. – 0 and True. – 1,** we have a target class imbalance problem here so we need to balance the target class or else we will get false predictions which will be biased towards the negative class of churn that is False class.

So in **python** we are using **SMOTE** method for **over sampling** the **minority class** and in **R** we are using **ROSE package** for balancing the target class.

2.1.6 Balancing Target Class (SMOTE & ROSE)

we do synthetic data only on the train **SMOTE** creates synthetic observations of the minority class (churn) by Finding the k-nearest-neighbors for minority class observations (finding similar observations) Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observation.
In simple words we can say that instead of creating replica of the data by adding observations from the minority class, it overcomes this imbalance problem by generating artificial data and it is also one of the over sampling technique.

With the help of ROSE and DMwr libraries it helps us to perform sampling of the imbalanced data in **R**, ROSE (Random Over Sampling Examples) which helps us generate artificial data using sampling method.

Data after applying smote in python here nearly 2367 artificial data has been generated

```
0    2850
1    2850
Name: Churn, dtype: int64
```

Data after applying ROSE in R here nearly 339 artificial data has been generated

```
# 0   1

# 1218 1116
```

We can observe that the target class is balanced now after applying SMOTE in python and ROSE in R.

2.2 Model Development

The problem category falls under the Supervised learning algorithms that is Classification Models and as we also have the target label that is Churn here we need to develop the classification algorithms to predict the target label that is if True whether the customer has churned or False the customer has not churned.

Which can be achieved by using the most traditional Data Mining / Machine Learning Algorithms which are Decision Tree, Random Forest, Logistic Regression, K-Nearest Neighbor and Naïve Bayes Classification Models are being developed for the customer churn reduction by developing the predictive models which are going to predict the customer usage pattern data by feeding the data to the models we develop the classification models and evaluate their performance based on the error metrics and we are going to finalize the model on basis

of K-Fold Cross Validation best accuracy of the model will be considered and by using Hyper-parameter Tuning method that is GridSearchCV we are going to find the best parameters like which 'criterion', 'max_depth', 'n_estimators' and 'scoring' among these which are optimum for tuning and increasing the performance of the final model selection is based on the K-Fold CV, Hyper-Parameter Tuning in model selection stage.

So here the target class is of binary in this case we try to predict the values or categories which we call it as classification.

In **python** we are defining two functions the first function is to fit the model on the balanced data and make predictions by passing the models to the predict function we are going to predict the new test cases here and it will result out the K-Fold-CV accuracy and the metrics such as precision, recall, f1-score and support are generated from classification report.

The second function defined here is for evaluation of the performance of the models that is by analyzing the Error Metrics. Such as Confusion Matrix // Contingency Table, although not an error metric but all error metrics are calculated from the confusion matrix itself. False Negative Rate (FNR), False Positive Rate (FPR), Sensitivity// True Positive Rate // Recall, and Specificity// True Negative Rate (TNR) are evaluated on all the classification models.

In **R** also we are defining the error metrics function for the evaluation of the models performance the function results out Such as Confusion Matrix // Contingency Table, although not an error metric but all error metrics are calculated from the confusion matrix itself. Accuracy, False Negative Rate, False Positive Rate, Sensitivity// True Positive Rate // Recall, and Specificity// True Negative Rate (TNR), and Precision for the evaluation of the classification models performance.

Here the main moto is to reduce the False Negative Rate (FNR) and to select the outperformed on basis of the best accuracy and low False Negative Rate as the Customer/ Client may be interested in many metrics but False Negative Rate is more concentrated here to reduce it as much as possible by developing necessary classification models.

Description of Error Metrics used: -

Confusion Matrix: To extract the error, accuracy, the value of recall, precision first we need to develop a confusion matrix is also an error metric in simple terms we say it as contingency table.

Accuracy: Number of correct predictions made divide by the total number of predictions made.

Specificity//True Negative Rate: The proportion of actual negative cases which are correctly identified.

Sensitivity// True Positive Rate // Recall: The proportion of actual positive cases which are correctly identified.

Precision: The proportion of positive predictive class which are identified.

f1- Score: It is the balance between the precision and recall.

Support: It is the number of actual occurrence in the specified dataset.

False Negative Rate: Deals with the proportion of number of wrongly positive classified with negative outcomes.

False Positive Rate: Deals with the proportion of number of wrongly negative classified with positive outcomes.

Miss Classification Error: Is nothing but 1- Accuracy, 'Error' is nothing but your misclassified 'records'.

Error = Classifying a record which belongs to one class when it belongs to another class, which means that if your class belongs to **'positive'**, but our model has predicted it as a **'negative'** which is my **'Error'**.

2.2.1 Developing Classification Models

Decision Tree Classifier Model Performance Results in Python

```
K Fold Crossvalidation Accuracy-------> 0.932280701754

************Classification Report*************

              precision    recall  f1-score   support

           0       0.96      0.88      0.92      1443
           1       0.49      0.76      0.60       224

   micro avg       0.86      0.86      0.86      1667
   macro avg       0.73      0.82      0.76      1667
weighted avg       0.90      0.86      0.87      1667

************Confusion Matrix******************

[[1267  176]
 [  54  170]]
```

Decision tree model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **93.22 %** here **54** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Decision Tree Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 24.1071428571

False Positive Rate--------------> 12.1968121968

Sensitivity/TPR/Recall-----------> 75.8928571429

Specificity/TNR------------------> 87.8031878032
```

False Negative Rate is **24.10 %** here.

AUC-ROC Curve of Decision Tree Model in Python

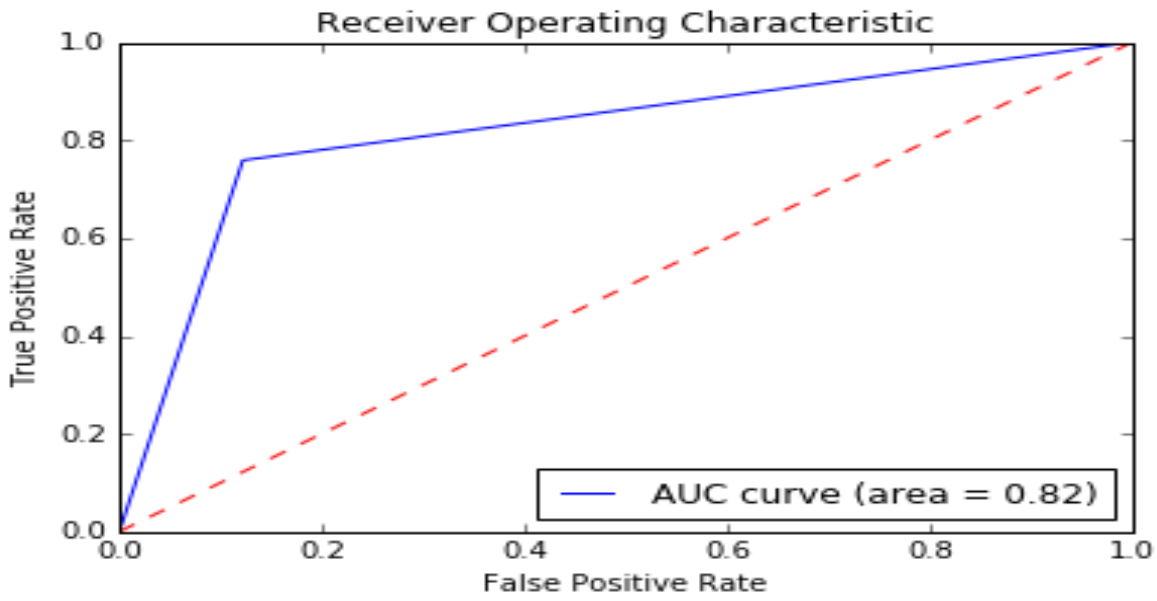


Fig 2.2 – Decision Tree AUC-ROC Curve

Decision Tree Classifier Performance results in R

```
ERROR METRICS OF DECISION TREE MODEL

Accuracy: ----------------------  88.3357041251778

False Negative Rate: ----------  15.4545454545455

False Positive Rate: ------------- 10.9612141652614

Sensitivity//TPR//Recall: ------ 84.5454545454545

Specificity//TNR: ---------------- 89.0387858347386

Precision: ------------------------ 58.8607594936709
```

Accuracy of the decision tree model in R is **88.33 %** false negative rate is **15.45 %**

AUC-ROC Curve of Decision Tree Model in R

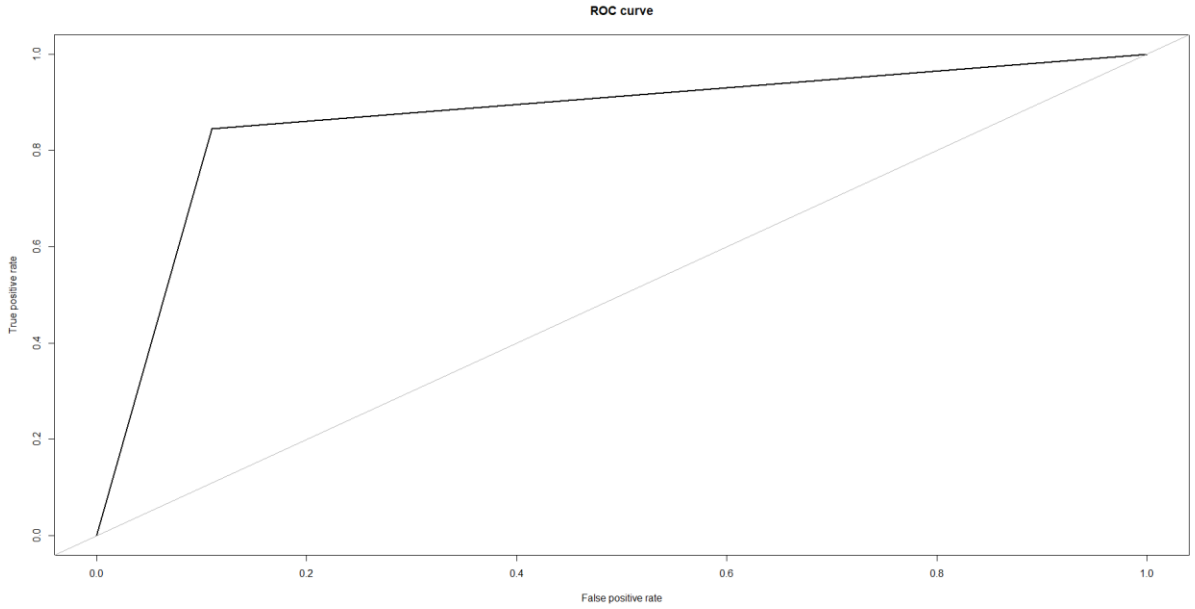


Fig 2.3 – Decision Tree AUC-ROC Curve with AUC - 86.8

Random Forest Classifier Model Performance results in Python

```
K Fold Crossvalidation Accuracy-------> 0.96649122807

************Classification Report*************

              precision    recall  f1-score   support

           0       0.97      0.91      0.94      1443
           1       0.58      0.83      0.69       224

   micro avg       0.90      0.90      0.90      1667
   macro avg       0.78      0.87      0.81      1667
weighted avg       0.92      0.90      0.90      1667

************Confusion Matrix******************

[[1310  133]
 [  38  186]]
```

Random Forest model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **96.64 %** here **38** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Random Forest Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 16.9642857143

False Positive Rate--------------> 9.21690921691

Sensitivity/TPR/Recall-----------> 83.0357142857

Specificity/TNR------------------> 90.7830907831
```

False Negative Rate is **16.96 %** less when compared to Decision tree model.

AUC-ROC Curve of Random Forest Model in Python

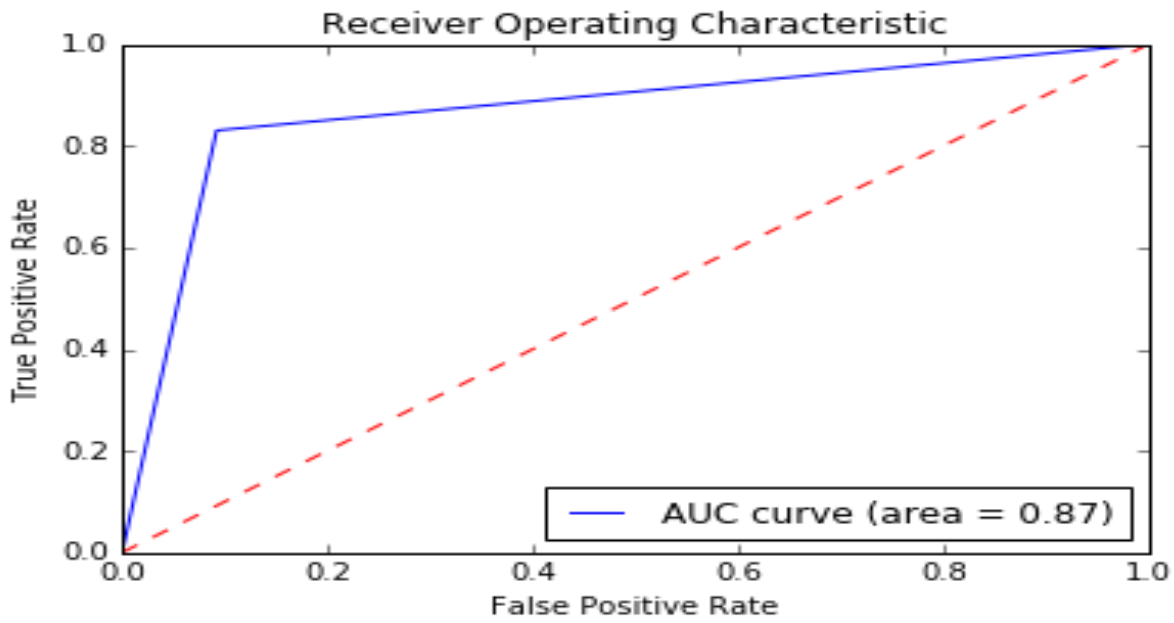


Fig 2.4 – Random Forest AUC-ROC Curve

Random Forest Classifier Performance results in R

```
ERROR METRICS OF RANDOM FOREST MODEL

Accuracy: ---------- 89.7581792318634

False Negative Rate: ------- 11.8181818181818

False Positive Rate: --------- 9.94940978077572

Sensitivity//TPR//Recall: --- 88.1818181818182

Specificity//TNR: ----------- 90.0505902192243

Precision: ---------- 62.1794871794872
```

Accuracy of the Random forest model is **89.75 %** and false negative rate is **11.81 %**

AUC-ROC Curve of Random Forest Model in R

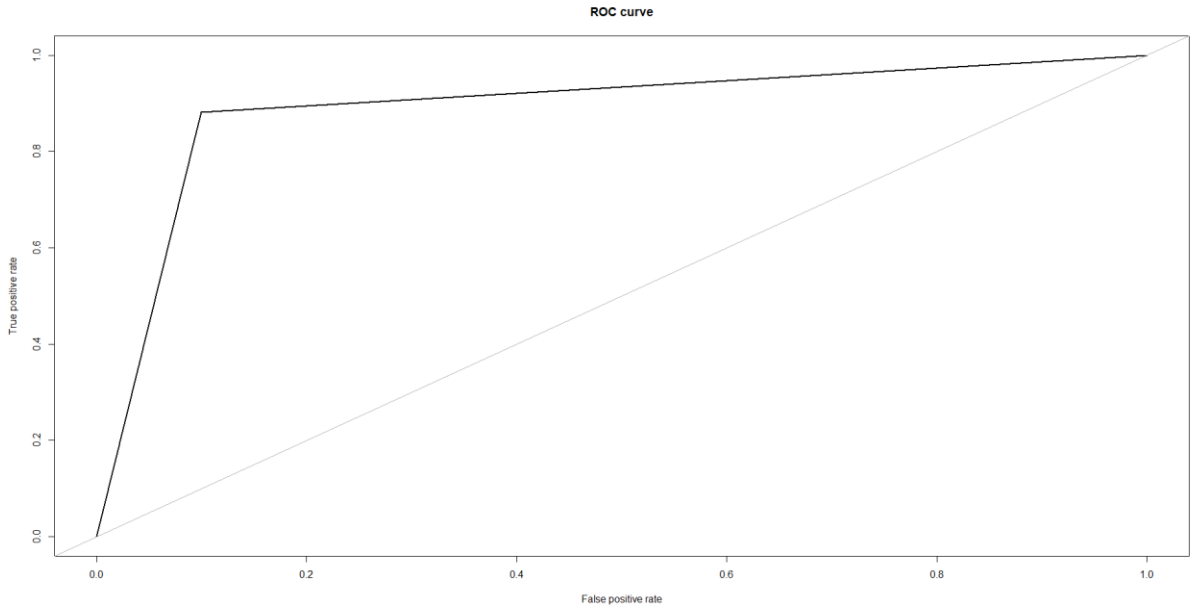


Fig 2.5 – Random Forest AUC-ROC Curve with AUC - 89.1

Logistic Regression Classifier Model Performance results in Python

```
K Fold Crossvalidation Accuracy-------> 0.777368421053

************Classification Report*************

              precision    recall  f1-score   support

           0       0.96      0.77      0.85      1443
           1       0.35      0.81      0.49       224

   micro avg       0.77      0.77      0.77      1667
   macro avg       0.66      0.79      0.67      1667
weighted avg       0.88      0.77      0.80      1667

************Confusion Matrix******************

[[1105  338]
 [  42  182]]
```

Logistic Regression model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **77.73 %** here **42** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Logistic Regression Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 18.75

False Positive Rate--------------> 23.4234234234

Sensitivity/TPR/Recall-----------> 81.25

Specificity/TNR------------------> 76.5765765766
```

False negative rate here is **18.75 %**

AUC-ROC Curve of Logistic Regression Model in Python

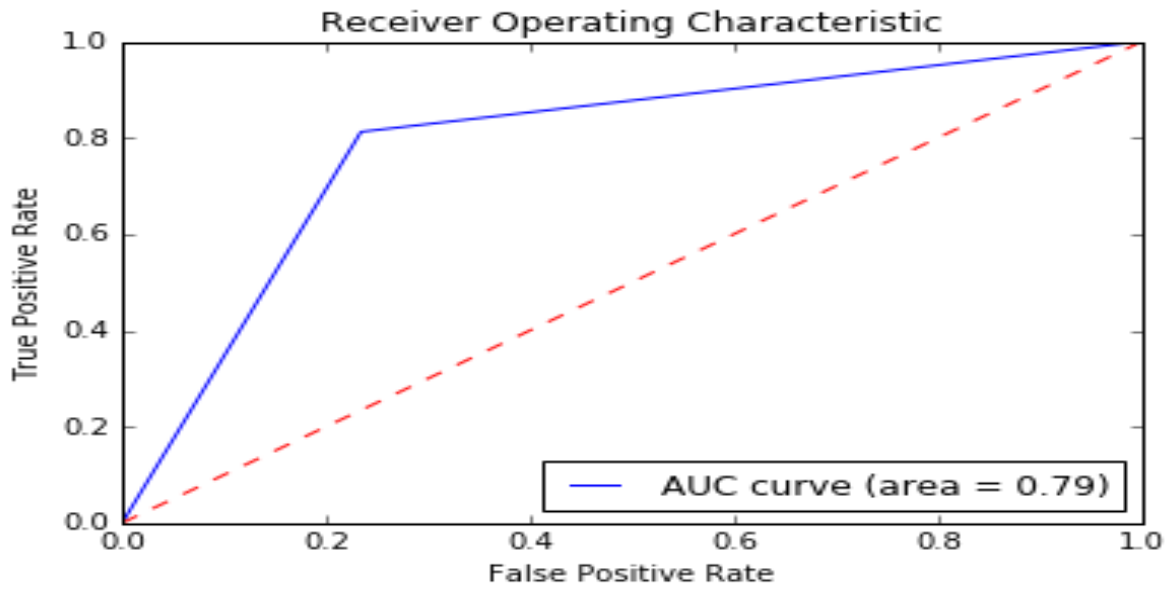


Fig 2.6 – Logistic Regression AUC-ROC Curve

Logistic Regression Classifier Performance results in R

```
ERROR METRICS OF LOGISTIC REGRESSION MODEL

Accuracy: ---------- 80.2275960170697

False Negative Rate: ------- 27.2727272727273

False Positive Rate: -------- 18.3811129848229

Sensitivity//TPR//Recall: -- 72.7272727272727

Specificity//TNR: ---------- 81.6188870151771

Precision: ---------- 42.3280423280423
```

Accuracy of the Logistic Regression model is **80.22 %** and false negative rate is **27.27 %**

AUC-ROC Curve of Logistic Regression Model in R

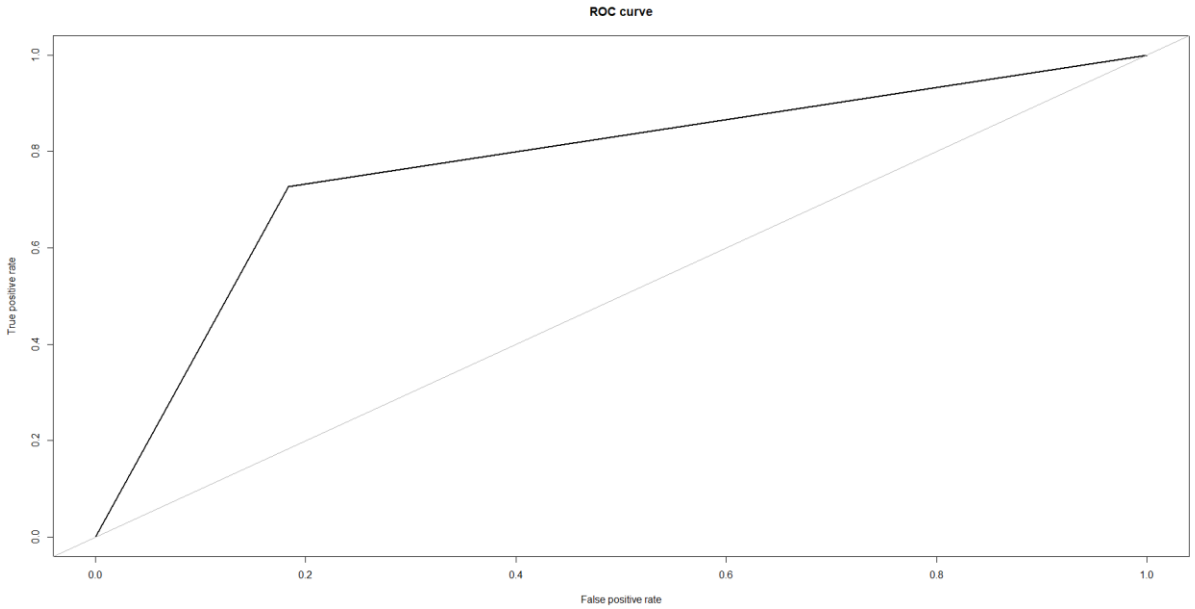


Fig 2.7 – Logistic Regression AUC-ROC Curve with AUC - 77.2

K Nearest Neighbor Classifier Model Performance results in Python

```
K Fold Crossvalidation Accuracy-------> 0.902807017544

************Classification Report*************

              precision    recall  f1-score   support

           0       0.94      0.82      0.87      1443
           1       0.36      0.66      0.46       224

   micro avg       0.80      0.80      0.80      1667
   macro avg       0.65      0.74      0.67      1667
weighted avg       0.86      0.80      0.82      1667

************Confusion Matrix******************

[[1179  264]
 [  77  147]]
```

K Nearest Neighbor model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **90.28 %** here **77** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

K Nearest Neighbor Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 34.375

False Positive Rate--------------> 18.2952182952

Sensitivity/TPR/Recall-----------> 65.625

Specificity/TNR------------------> 81.7047817048
```

False Negative Rate is **34.37 %**

AUC-ROC Curve of K Nearest Neighbor Model in Python

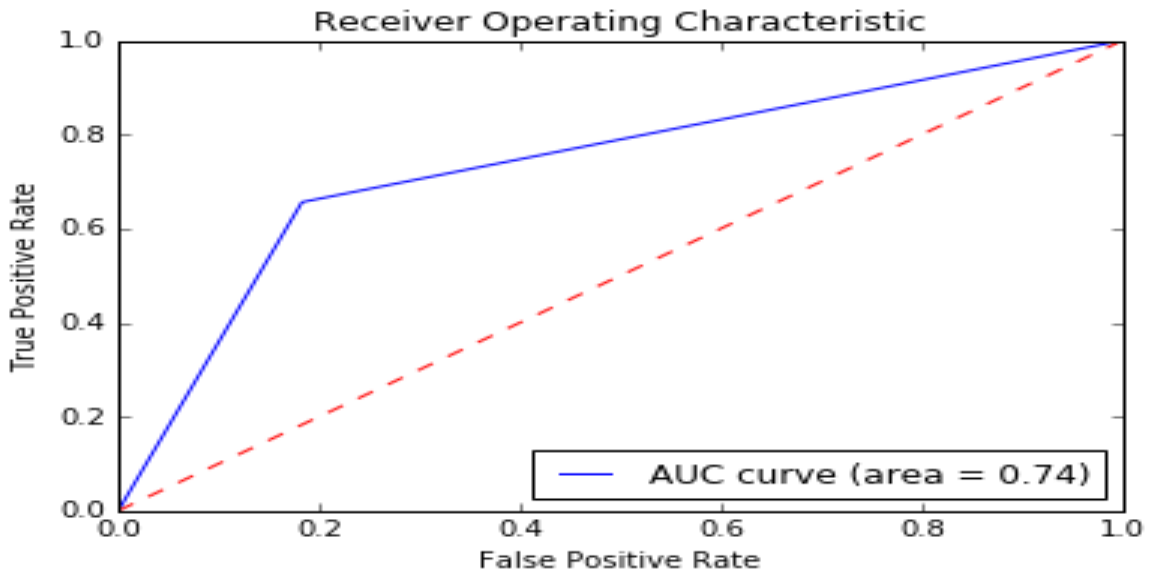


Fig 2.8 – K Nearest Neighbor AUC-ROC Curve

K Nearest Neighbor Classifier Performance results in R

```
ERROR METRICS OF K NEAREST NEIGHBOR MODEL

 Accuracy: ----------------------  86.4864864864865

 False Negative Rate: -------   45.2830188679245

 False Positive Rate: ---------- 4.22794117647059

 Sensitivity//TPR//Recall: --- 54.7169811320755

 Specificity//TNR: ------------- 95.7720588235294

 Precision: ---------------------- 79.0909090909091
```

Accuracy of the K Nearest Neighbor model is **86.48 %** and false negative rate is **45.28 %**

AUC-ROC Curve of K Nearest Neighbor Model in R

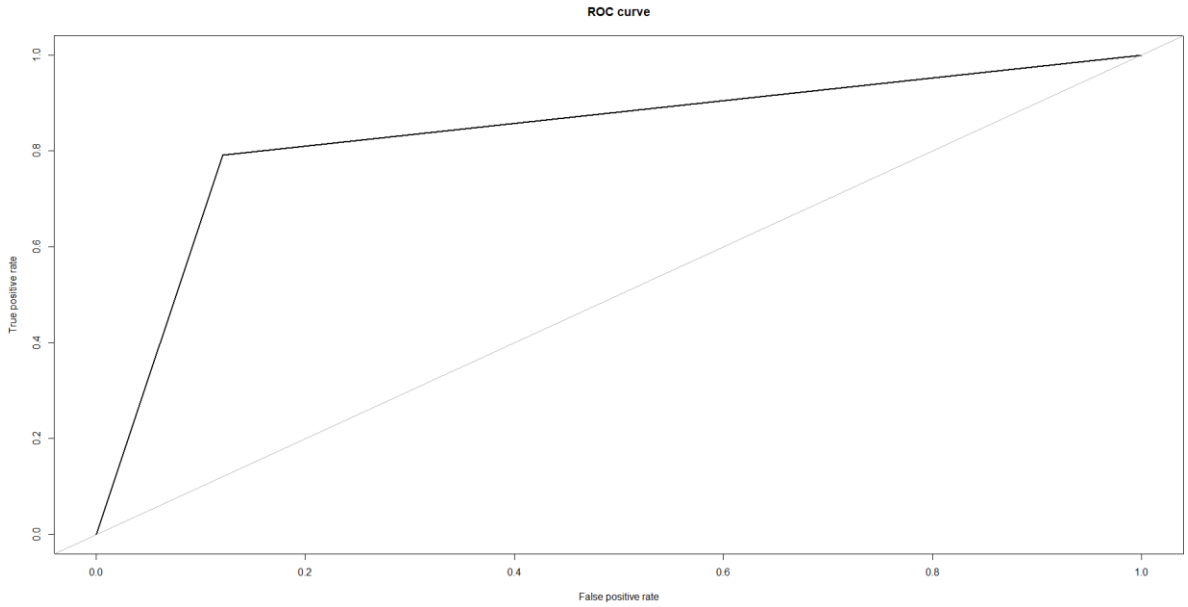


Fig 2.9 – K Nearest Neighbor AUC-ROC Curve with AUC - 83.5

Naïve Bayes Classifier Model Performance results in Python

```
K Fold Crossvalidation Accuracy-------> 0.80649122807

************Classification Report*************

              precision    recall  f1-score   support

           0       0.97      0.78      0.87      1443
           1       0.37      0.83      0.51       224

   micro avg       0.79      0.79      0.79      1667
   macro avg       0.67      0.81      0.69      1667
weighted avg       0.89      0.79      0.82      1667

************Confusion Matrix******************

[[1132  311]
 [  39  185]]
```

Naïve Bayes model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **80.64 %** here **39** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Naïve Bayes Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 17.4107142857

False Positive Rate--------------> 21.5523215523

Sensitivity/TPR/Recall-----------> 82.5892857143

Specificity/TNR------------------> 78.4476784477
```

False Negative Rate here is **17.41 %**

AUC-ROC Curve of Naïve Bayes Model in Python

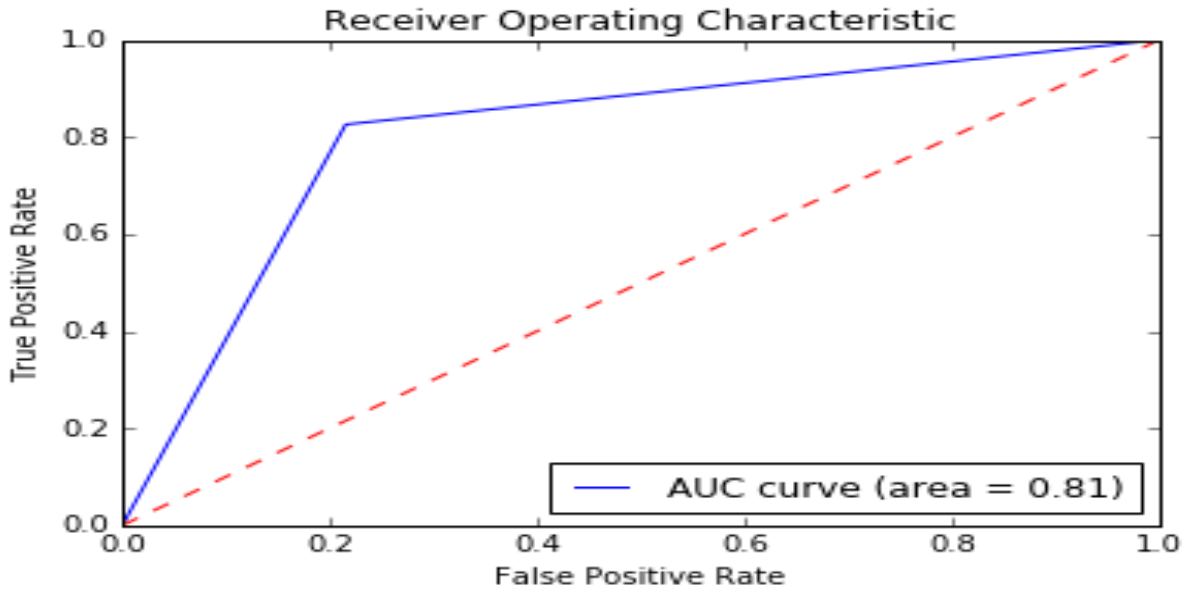


Fig 3 – Naïve Bayes AUC-ROC Curve

Naïve Bayes Classifier Performance results in R

```
ERROR METRICS OF NAIVE BAYES MODEL

Accuracy: ---------------------- 86.9132290184922

False Negative Rate: -------- 43.6619718309859

False Positive Rate: ---------- 5.3475935828877

Sensitivity//TPR//Recall: --- 56.3380281690141

Specificity//TNR: ------------- 94.6524064171123

Precision: ---------------------- 72.7272727272727
```

Accuracy of the Naïve Bayes model in R is **86.91 %** and False Negative Rate is **43.66 %**

AUC-ROC Curve of Naïve Bayes Model in R

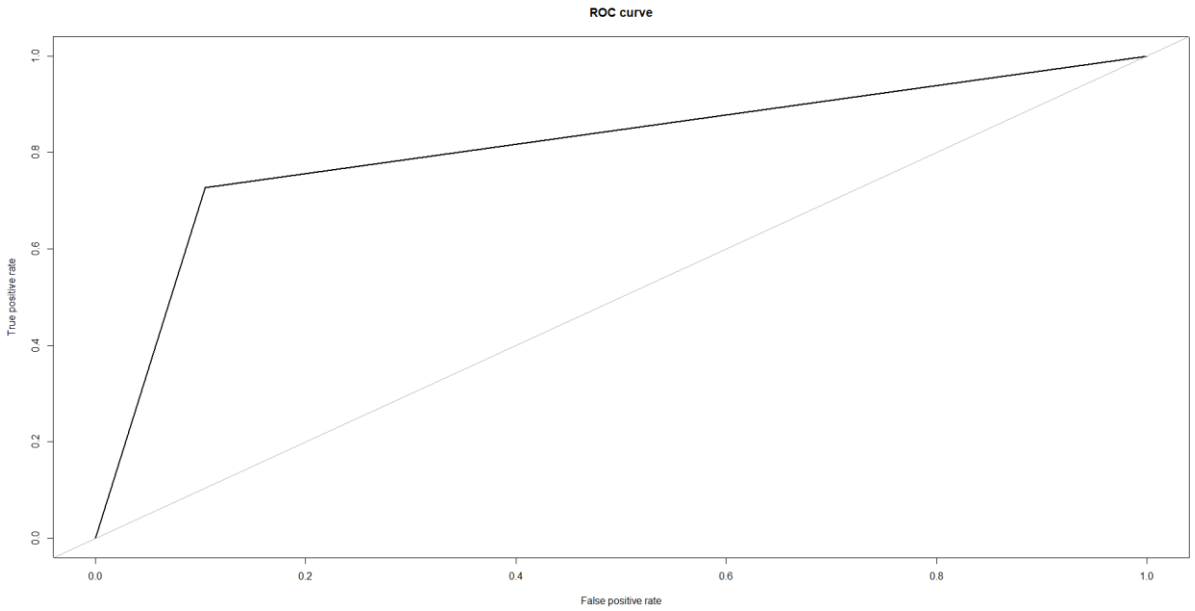


Fig 3.1 – Naïve Bayes AUC-ROC Curve with AUC - 81.1

All the classification models have been developed here and compared on the ground on basis of low False Negative Rate and Good AUC-ROC Curve is considered for Hyper-Parameter Tuning and optimization in python.

Out of all the models in python Random Forest Model outperformed here with good K-Fold CV accuracy and low False Negative Rate when compared with other models. In R Also Random Forest model outperformed here when compared with other models.

2.2.2 Model Selection

Comparing which models out performed in Python

Final Model Selection	Decision Tree	Random Forest	Logistic Regression	KNN	Naïve Bayes
K-Fold-CV Accuracy	93.22 %	96.64 %	77.36 %	90.28 %	80.64 %
False Negative Rate	24. 10 %	16.96 %	18.75 %	34.37 %	17.41 %
False Positive Rate	12.19 %	9.21 %	23.42 %	18.29 %	21.55 %
AUC-ROC Curve	82 %	87 %	79 %	74 %	81 %

Comparing which models out performed in R

Final Model Selection	Decision Tree	Random Forest	Logistic Regression	KNN	Naïve Bayes
K-Fold-CV Accuracy	88.33 %	89.75 %	80.22 %	86.48 %	86.91 %
False Negative Rate	15. 45 %	11.81 %	27.27 %	45.28 %	43.66 %
False Positive Rate	10.96 %	9.94 %	18.38 %	4.22 %	5.34 %
AUC-ROC Curve	86 %	89 %	77 %	83 %	81 %

As we can clearly observe that in both Python and R Random Forest Model outperforms rest of the models.

2.2.3 Hyper Parameter Tuning and Grid Search CV

Here we are applying hyper parameter tuning in python itself it is not carried out in R and the final model selection will be based on comparison with decision tree and random forest model which will perform well after tuning both the models that model will be considered as our final model.

K Fold Cross Validation will be our final accuracy for consideration k Fold cross validation randomly splits the entire data into 'K-Folds' first then for each and every K-fold of the dataset, will build the model on k-1 folds on the dataset and will test the model on K'th fold.

Hyper parameter tuning is done for finding the optimum parameters for the particular model Such as 'n_estimators', 'max_depth', 'criterion' are selected for getting the best performance on the tuned model.

Grid Search CV is a way of selecting the best model from the family of models, by passing the parameters which differ from each other parameters passed previously to the models.

Hyper parameter tuning of Decision Tree Model Performance results in Python

```
Best_accuracy---- 0.944912280702
{'criterion': 'gini', 'max_depth': 100}
```

```
K Fold Crossvalidation Accuracy-------> 0.928947368421

************Classification Report*************

              precision    recall  f1-score   support

           0       0.96      0.86      0.91      1443
           1       0.47      0.79      0.59       224

   micro avg       0.85      0.85      0.85      1667
   macro avg       0.72      0.83      0.75      1667
weighted avg       0.90      0.85      0.87      1667

************Confusion Matrix******************

[[1243  200]
 [ 47   177]]
```

Hyper Parameter Tuned Decision Tree model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **92.89 %** here **47** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Hyper Parameter Tuned Decision Tree Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 20.9821428571

False Positive Rate--------------> 13.86001386

Sensitivity/TPR/Recall-----------> 79.0178571429

Specificity/TNR------------------> 86.13998614
```

As compared to previous decision tree model and after tuning of decision tree model False Negative Rate Has Reduced **20.98 %**

AUC-ROC Curve of Tuned Decision Tree Model in Python

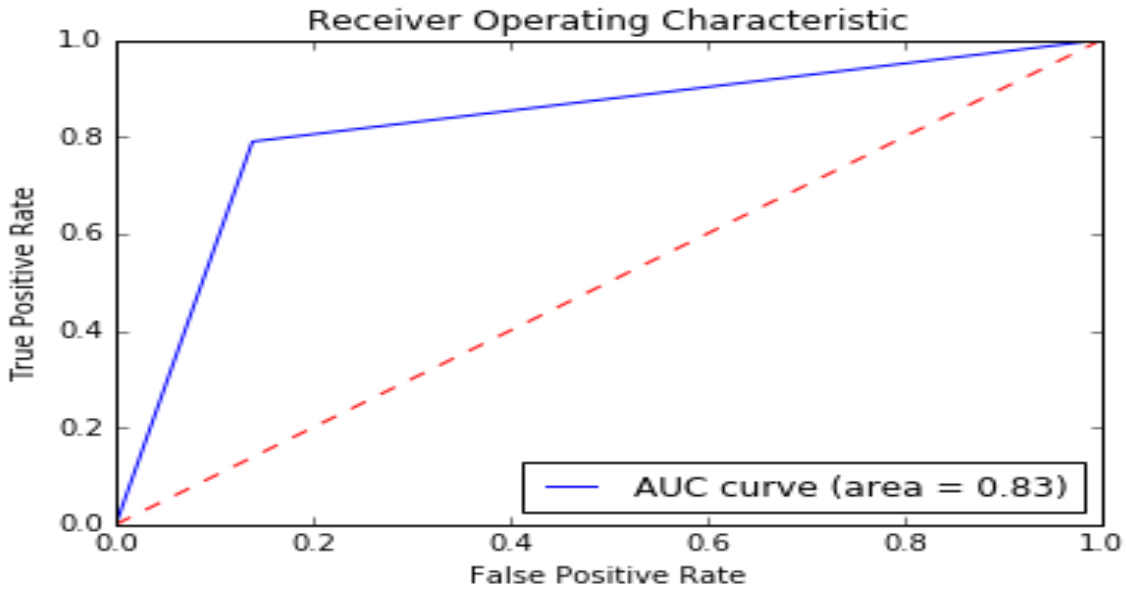


Fig 3.2 – Tuned Decision Tree AUC-ROC Curve

Hyper parameter tuning of Random Forest Model Performance results in Python

```
Best_accuracy 0.964912280702
{'criterion': 'entropy', 'max_depth': 50, 'n_estimators': 300}
```

```
K Fold Crossvalidation Accuracy-------> 0.966666666667

************Classification Report*************

              precision    recall   f1-score   support

           0       0.97      0.91      0.94      1443
           1       0.59      0.84      0.69       224

   micro avg       0.90      0.90      0.90      1667
   macro avg       0.78      0.88      0.82      1667
weighted avg       0.92      0.90      0.91      1667

************Confusion Matrix******************

[[1310  133]
 [  35  189]]
```

Hyper Parameter Tuned Random Forest model with K-Fold Cross validation accuracy on the test dataset is evaluated here with **96.66 %** here **35** observations are predicted as churning that is **false** but in actual case the customers are churning and it is **True.**

Hyper Parameter Tuned Random Forest Error Metrics in Python

```
<---------------ERROR METRICS-------------->

False Negative Rate--------------> 15.625

False Positive Rate--------------> 9.21690921691

Sensitivity/TPR/Recall-----------> 84.375

Specificity/TNR------------------> 90.7830907831
```

As compared to previous random forest model and after tuning of random forest tree model False Negative Rate Has Reduced **15.62 %**

AUC-ROC Curve of Tuned Random Forest Model in Python

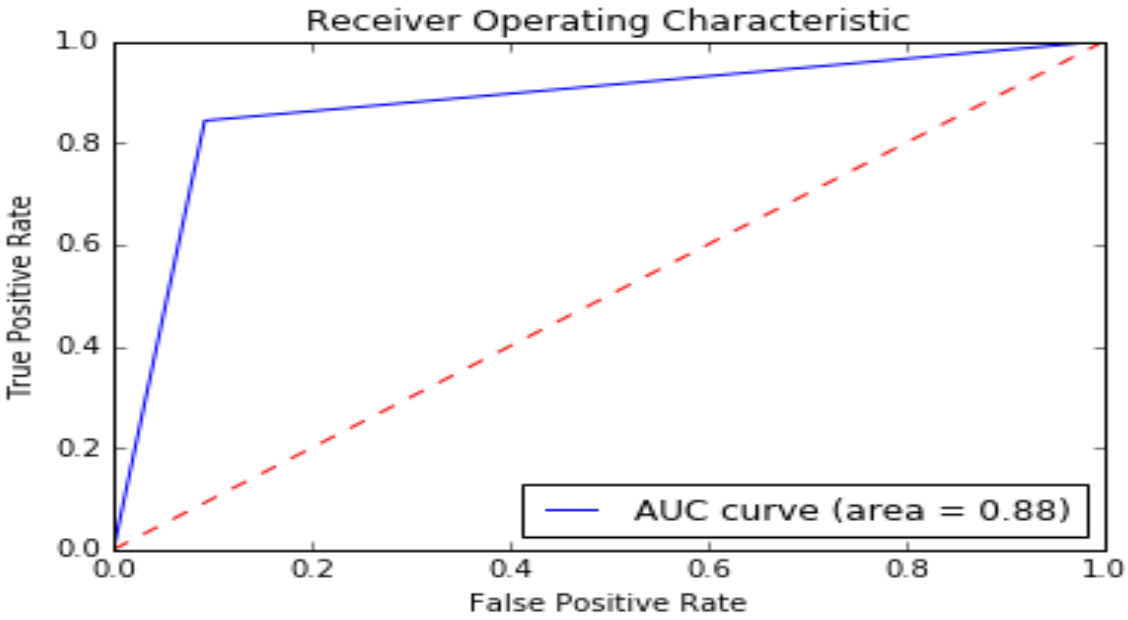


Fig 3.3 – Tuned Random Forest AUC-ROC Curve

2.2.4 Final Model Selection

Comparing Hyper Parameter Tuned Models Decision Tree & Random Forest in Python

Final Model Selection	Decision Tree	Random Forest
K-Fold-CV Accuracy	92.89 %	96.66 %
False Negative Rate	20. 98 %	15.62 %
False Positive Rate	13.86 %	9.21 %
AUC-ROC Curve	83 %	88 %

Random Forest Model outperforms when compared with Decision Tree Model we can also see that the false negative rate has been reduced after applying hyper parameter tuning and the K Fold CV Accuracy has also improved.

2.2.5 Best Parameters / Importance of Features

Let's check out the important features which have contributed important information to explain our target label in both Python and R. The importance of features is plotted on the tuned Random Forest Model in Python.

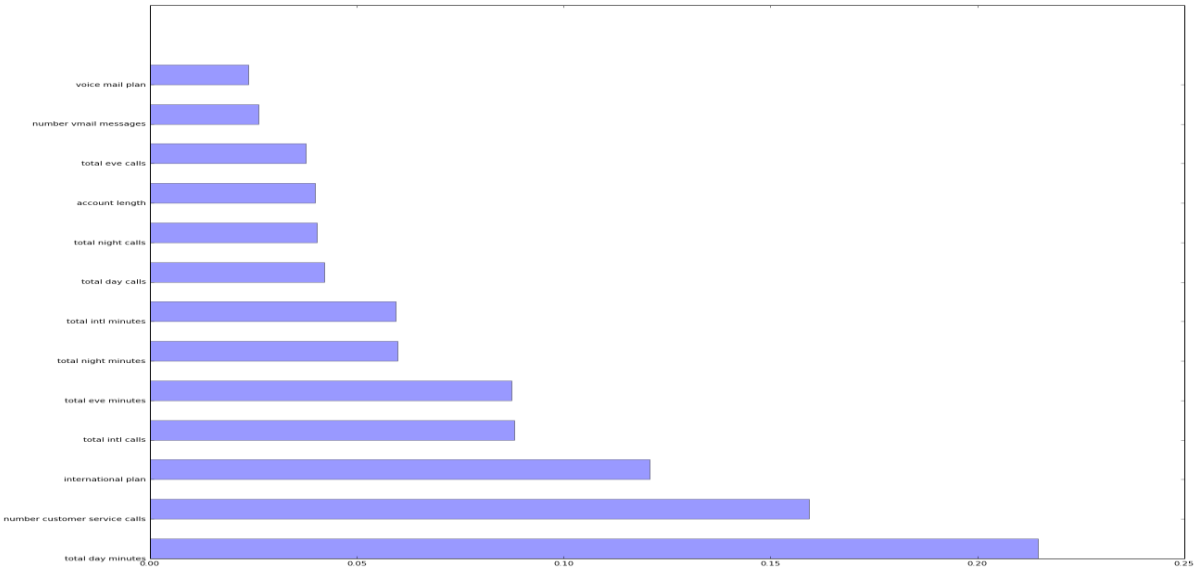


Fig 3.4 – Feature Importance in Python

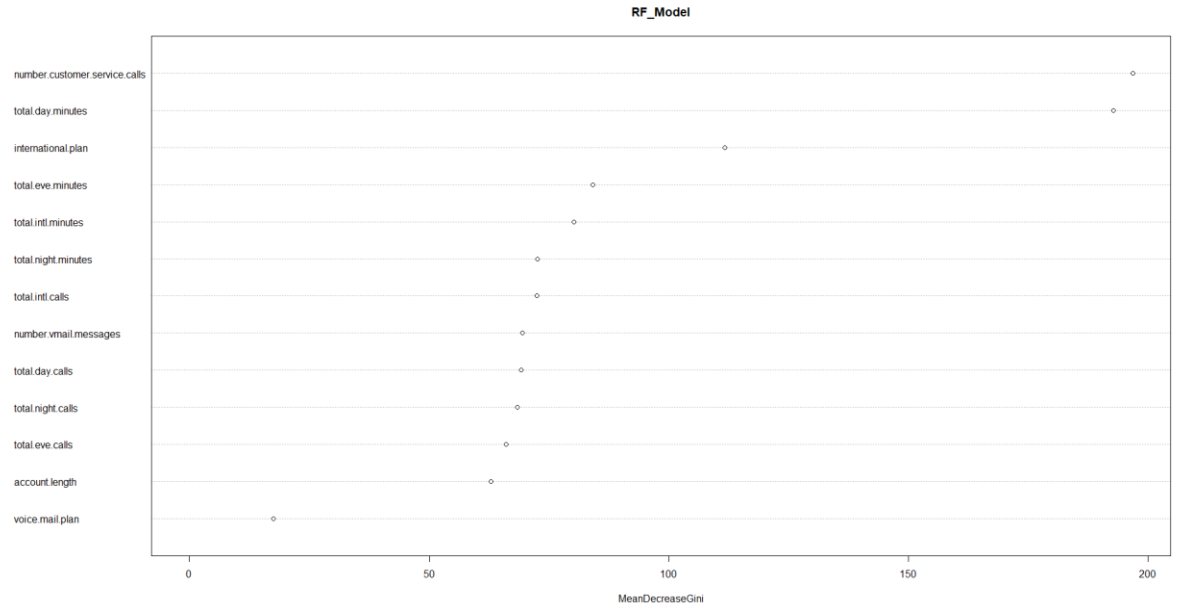


Fig 3.5 – Feature Importance in R

The Features / Predictors in python total_day_minutes, number_customer_service_calls, international_plan, are the top three features contributing much information.

The Features / Predictors in R contributing major information are total_day_minutes, number_customer_service_calls, international_plan, are the top three features contributing much information as we can see that in both Python and R for Random Forest model feature importance is same.

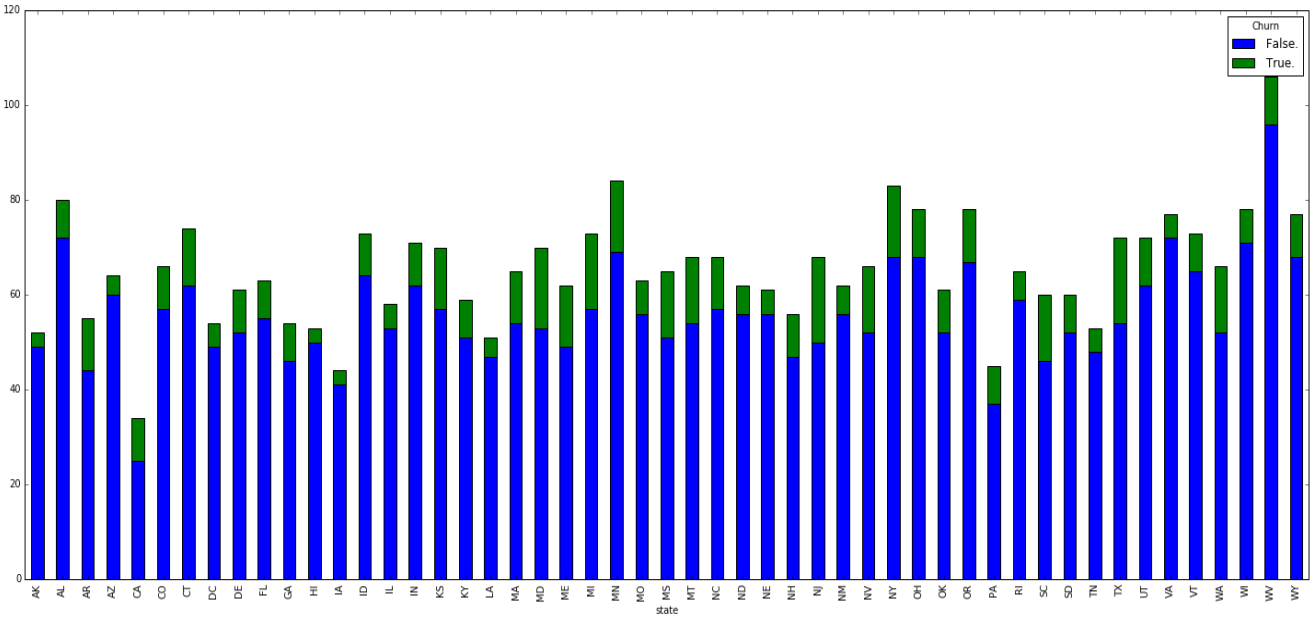
CHAPTER 3

3 Conclusion

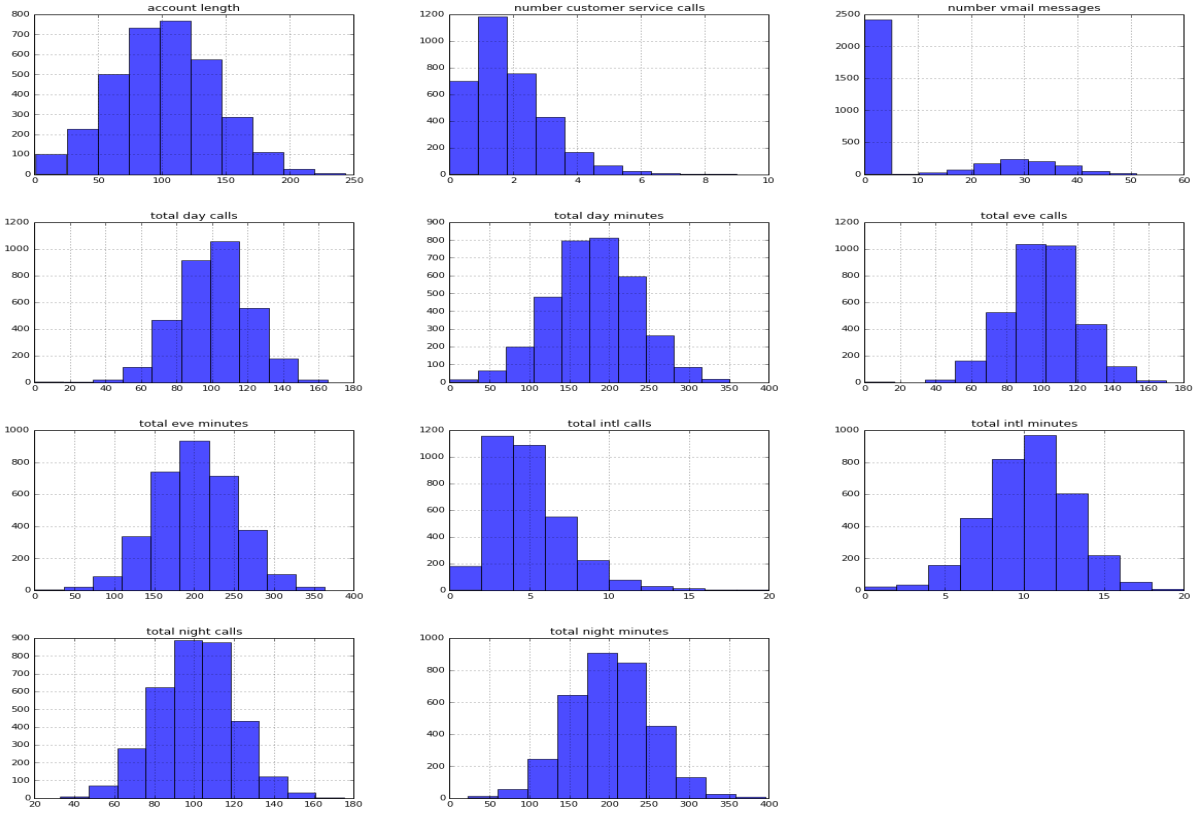
After developing all the classification models in both Python and R we have evaluated all the models based on their accuracy performance and importantly false negative rate taking into consideration by analyzing the error metrics of all models. Here we have come firstly came to a conclusion that random Forest has outperformed every other model in both Python and R as well.

That's why we have opted for Hyper parameter tuning of the best model which is performing well on basis of K-Fold Cross Validation, False Positive Rate, False Negative Rate and AUC-ROC Curve also we can see from the section 2.2.4 model selection section.

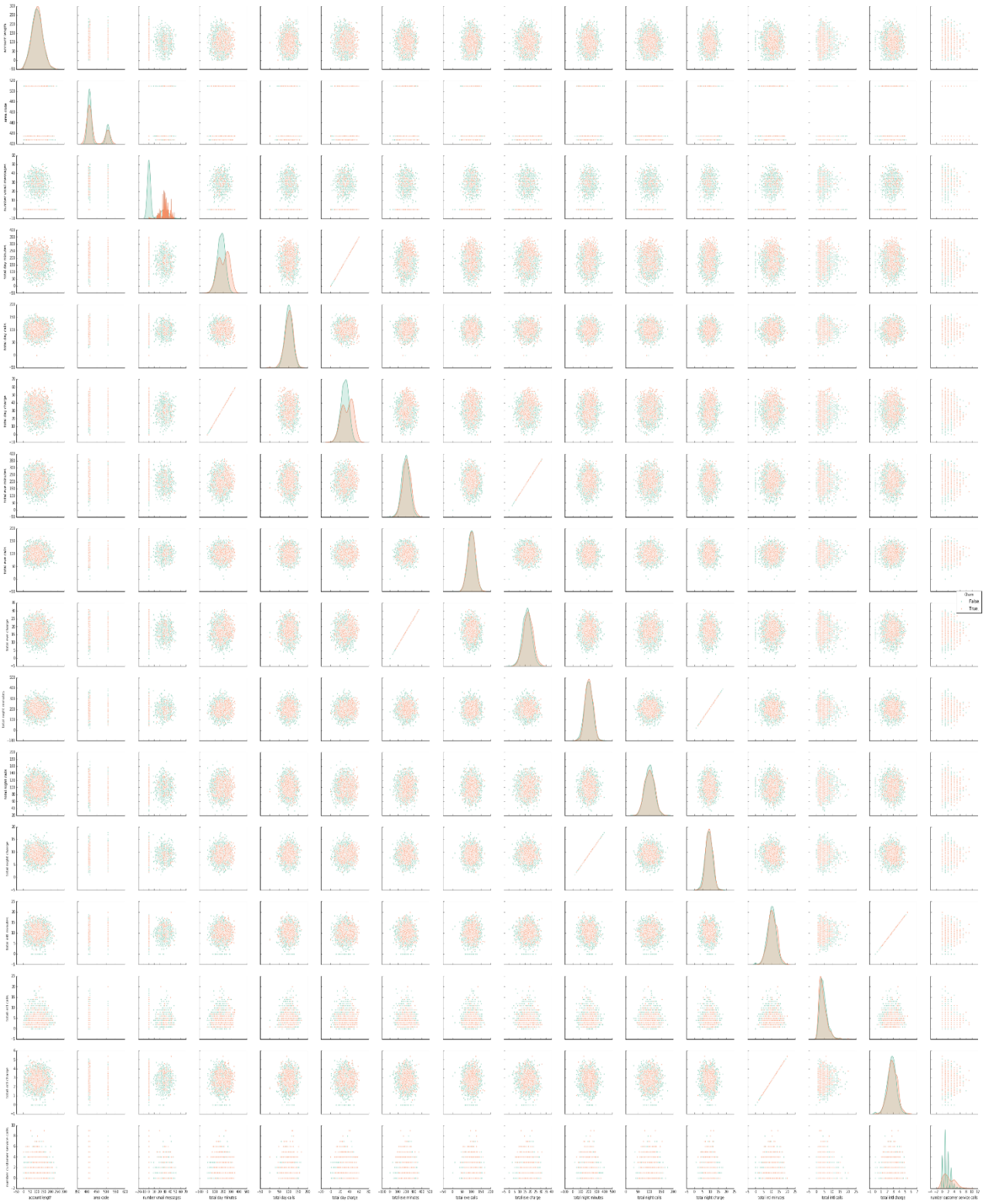
Appendix A: Extra Figures of Python and R



Customers Churning State Wise

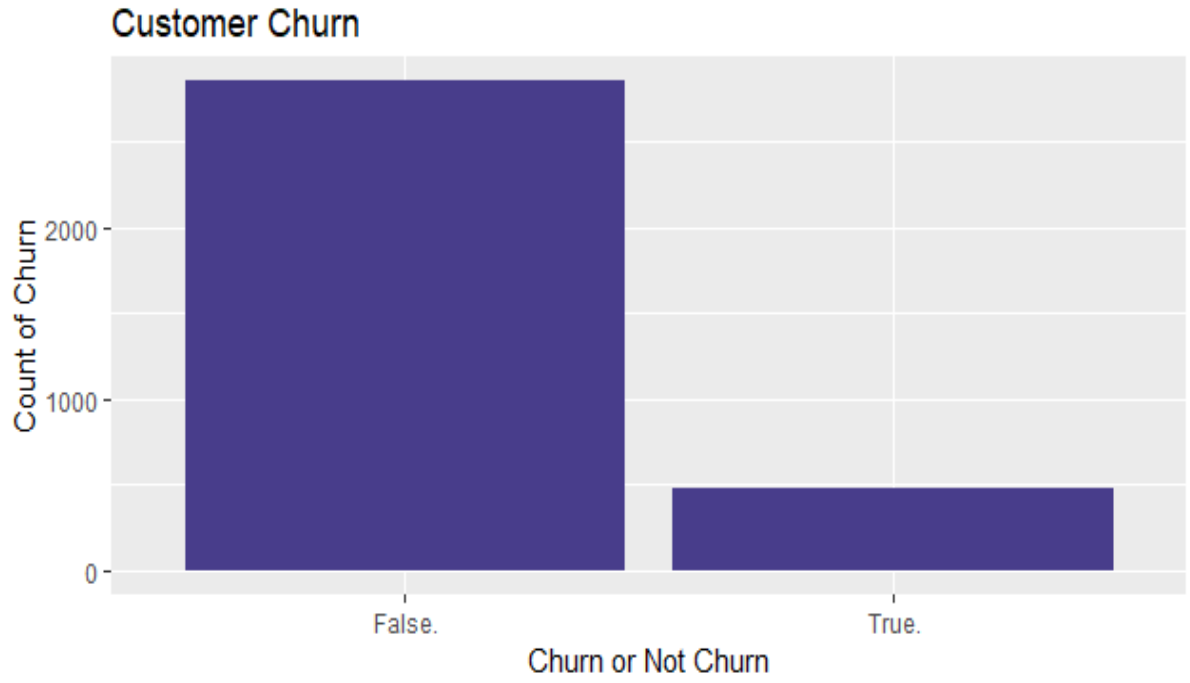


Pandas built-in histogram plot for checking the distribution of continuous variables

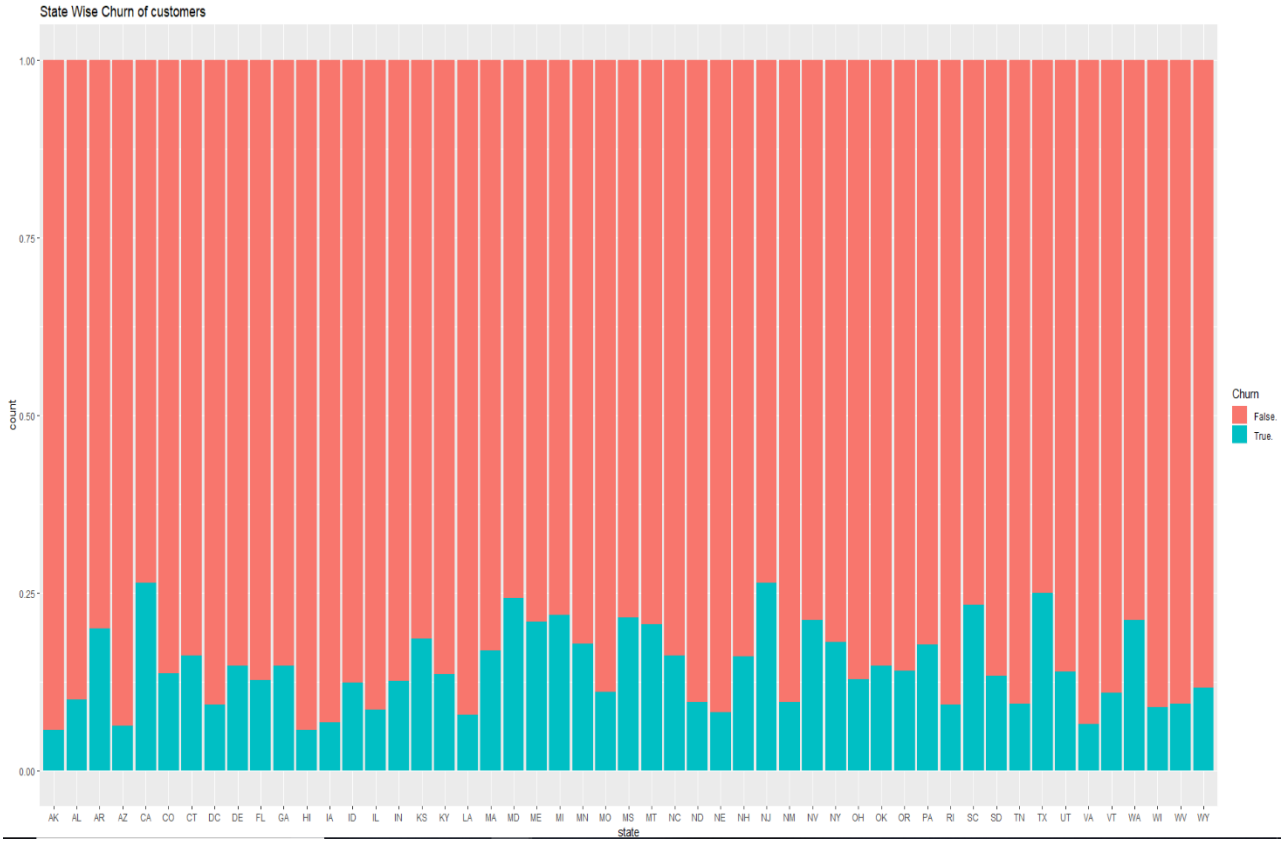


Pair Plot for Correlation Analysis and for checking highly correlated variables

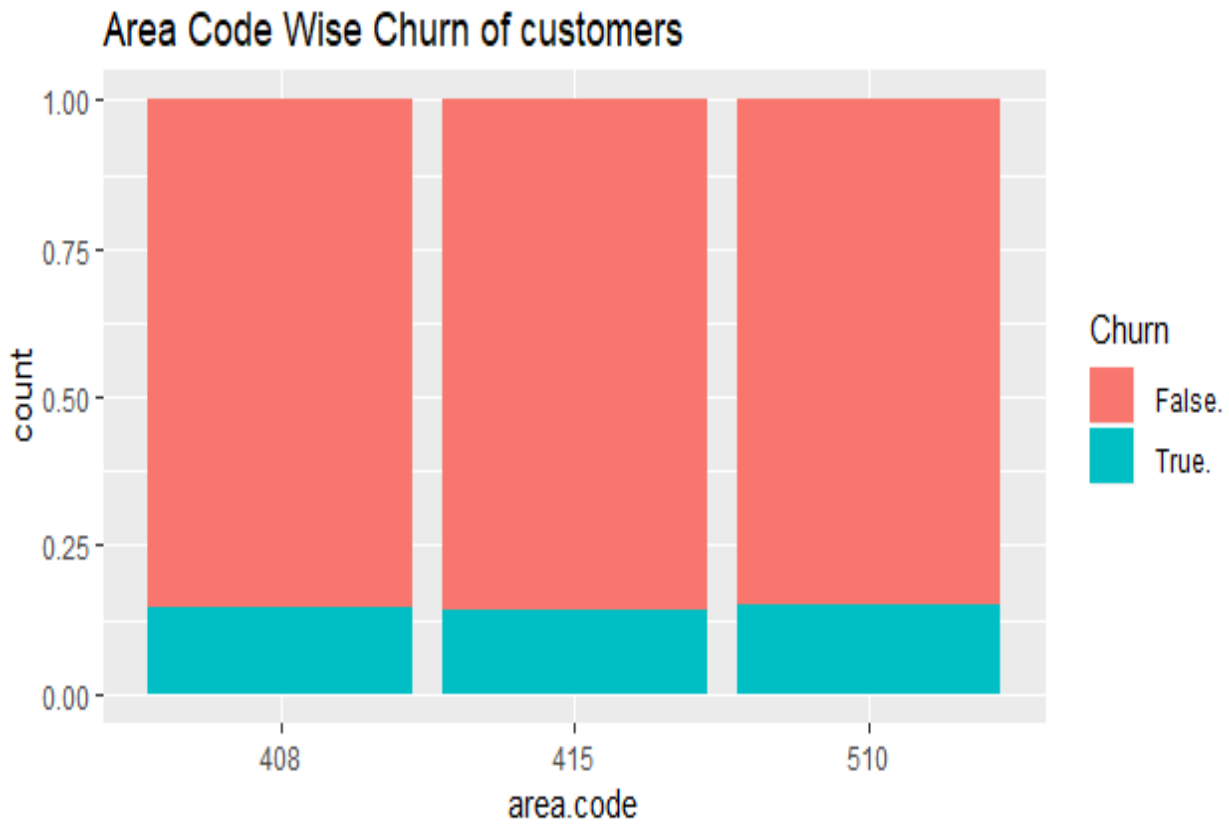
Extra Figures of R



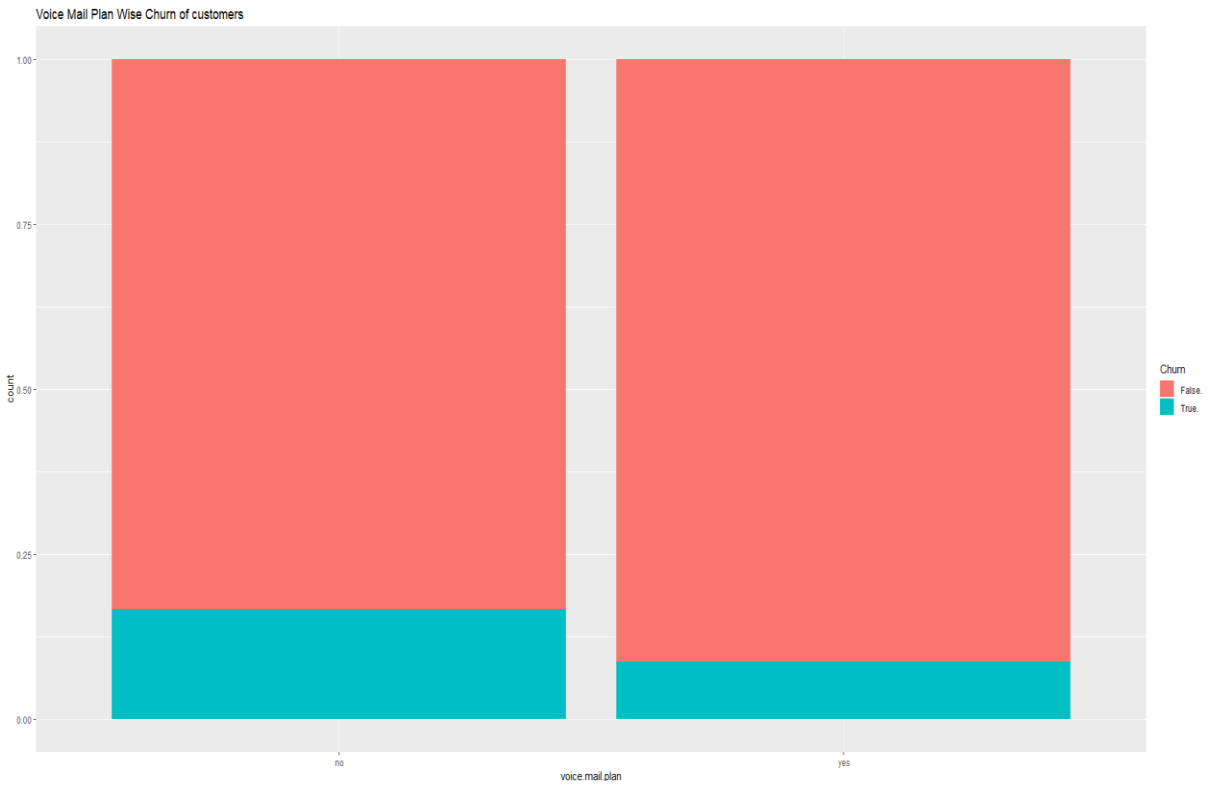
Count Churn Bar plot analysis



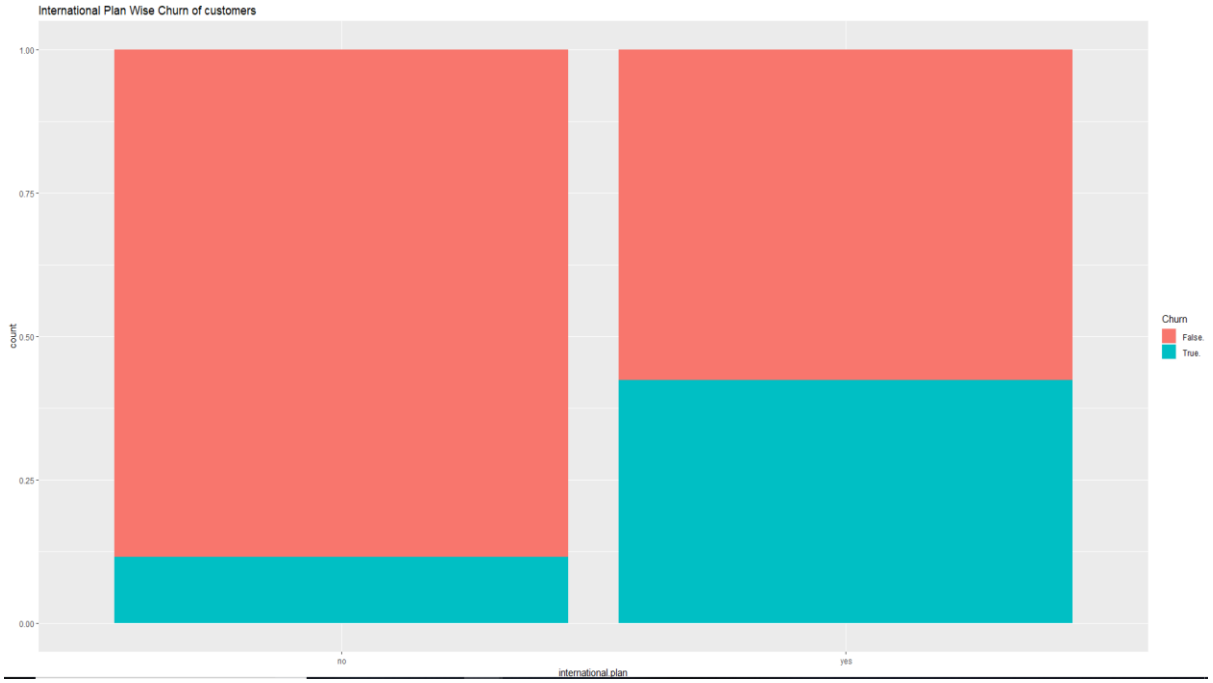
Churning of Customers State wise



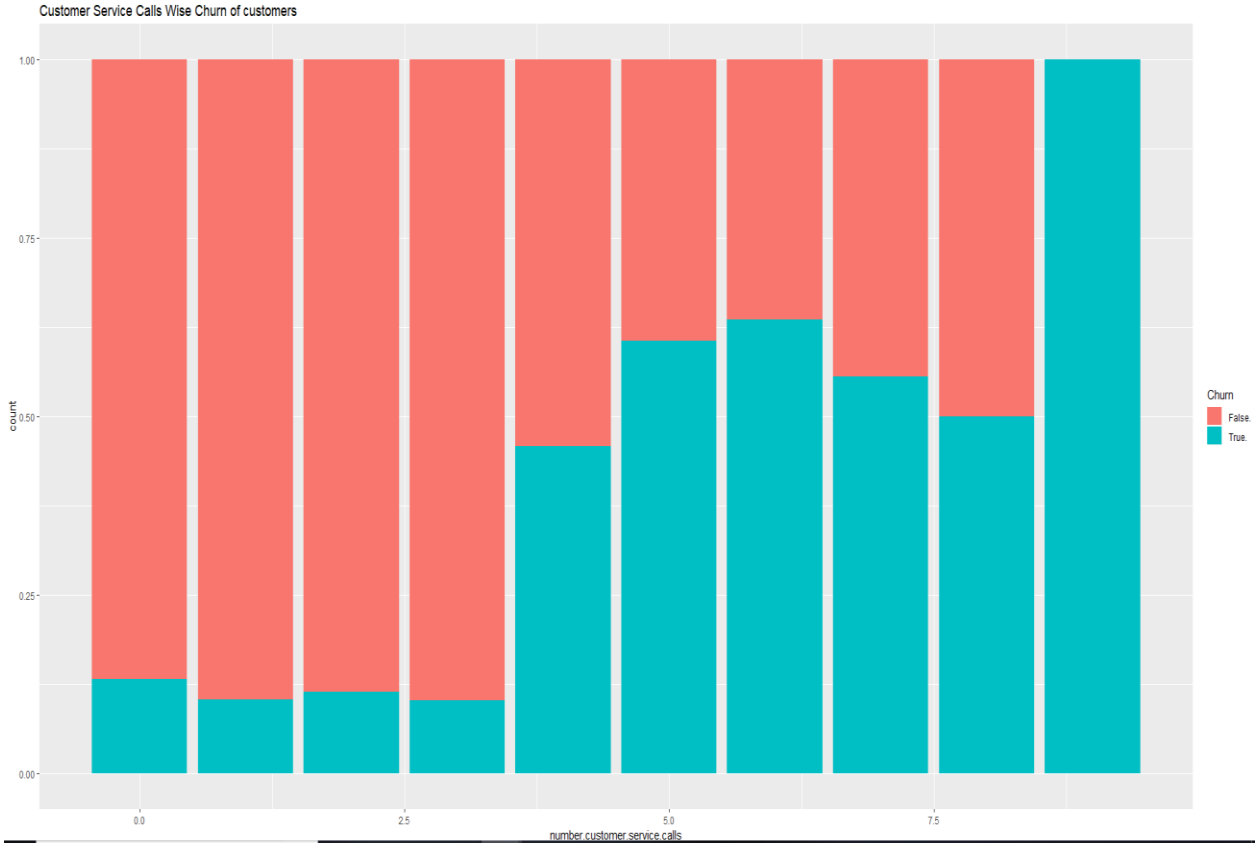
Customers Churn area code wise



Customers Churn Voice mail plan wise

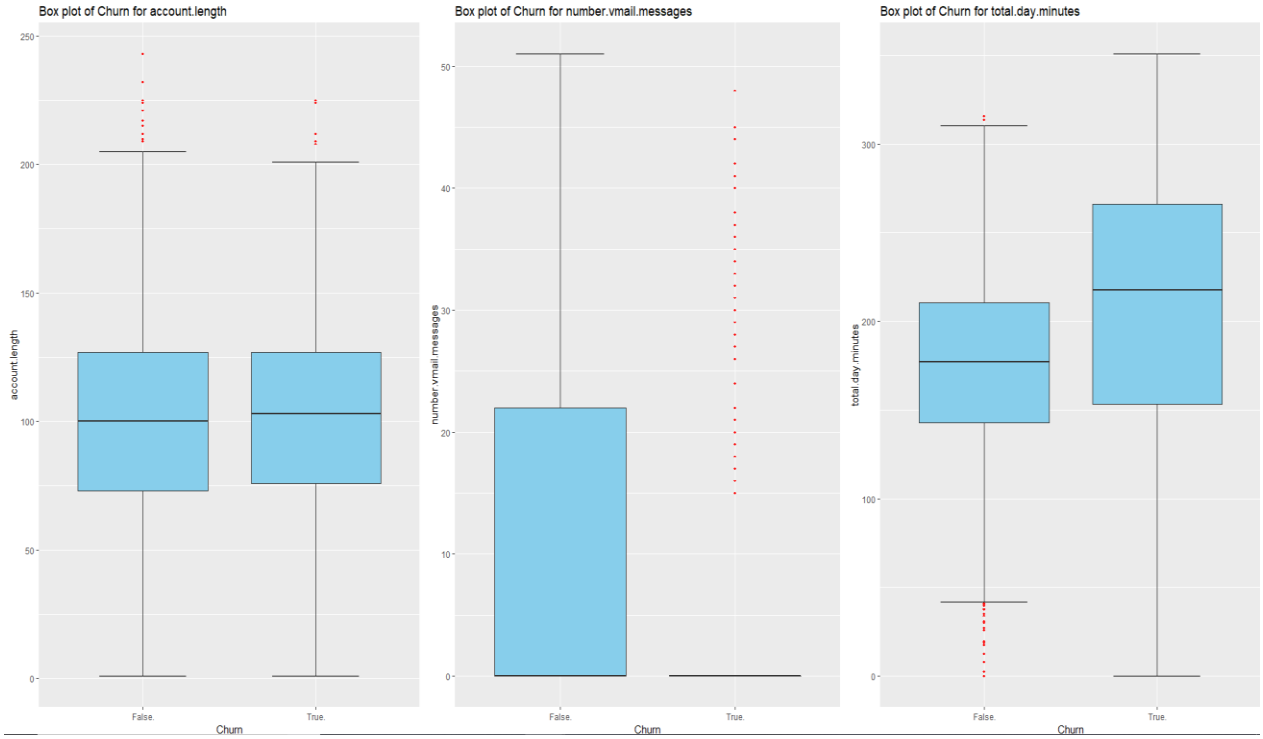


Customers Churn International plan wise

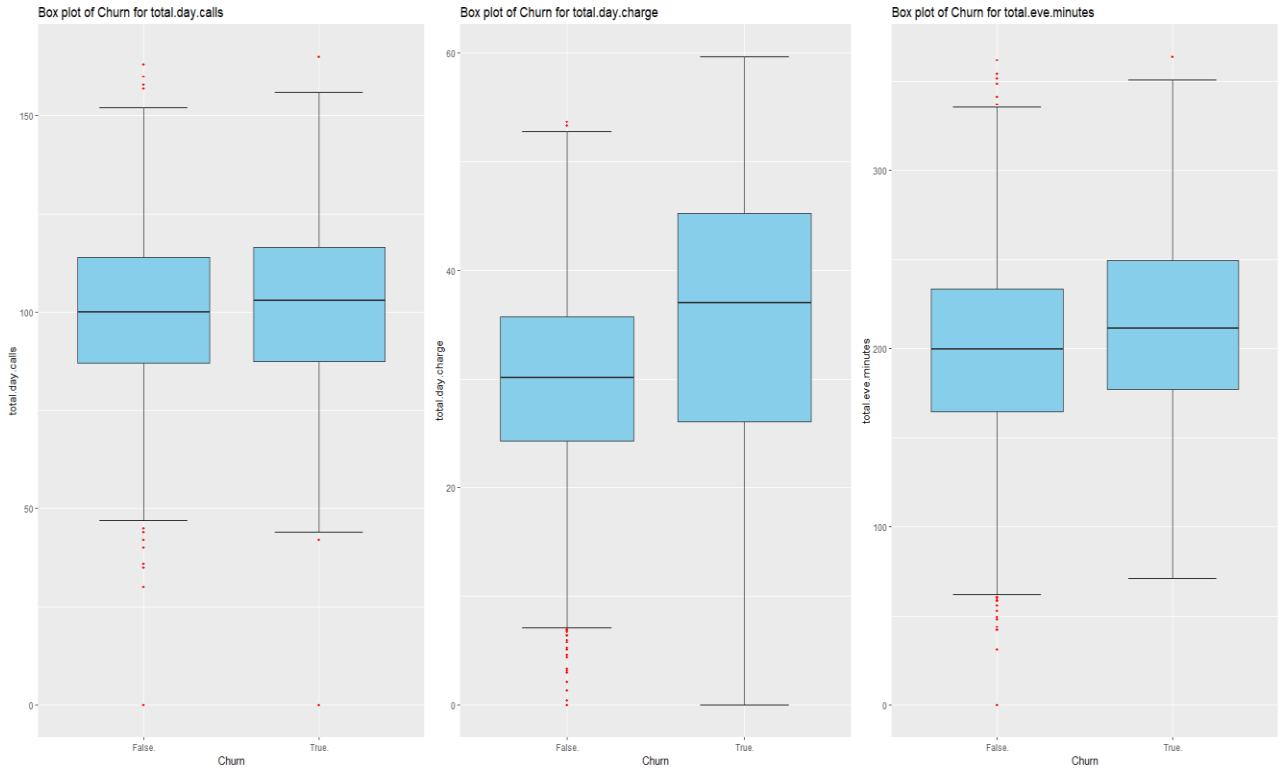


Customers Churn through number of customer service calls wise

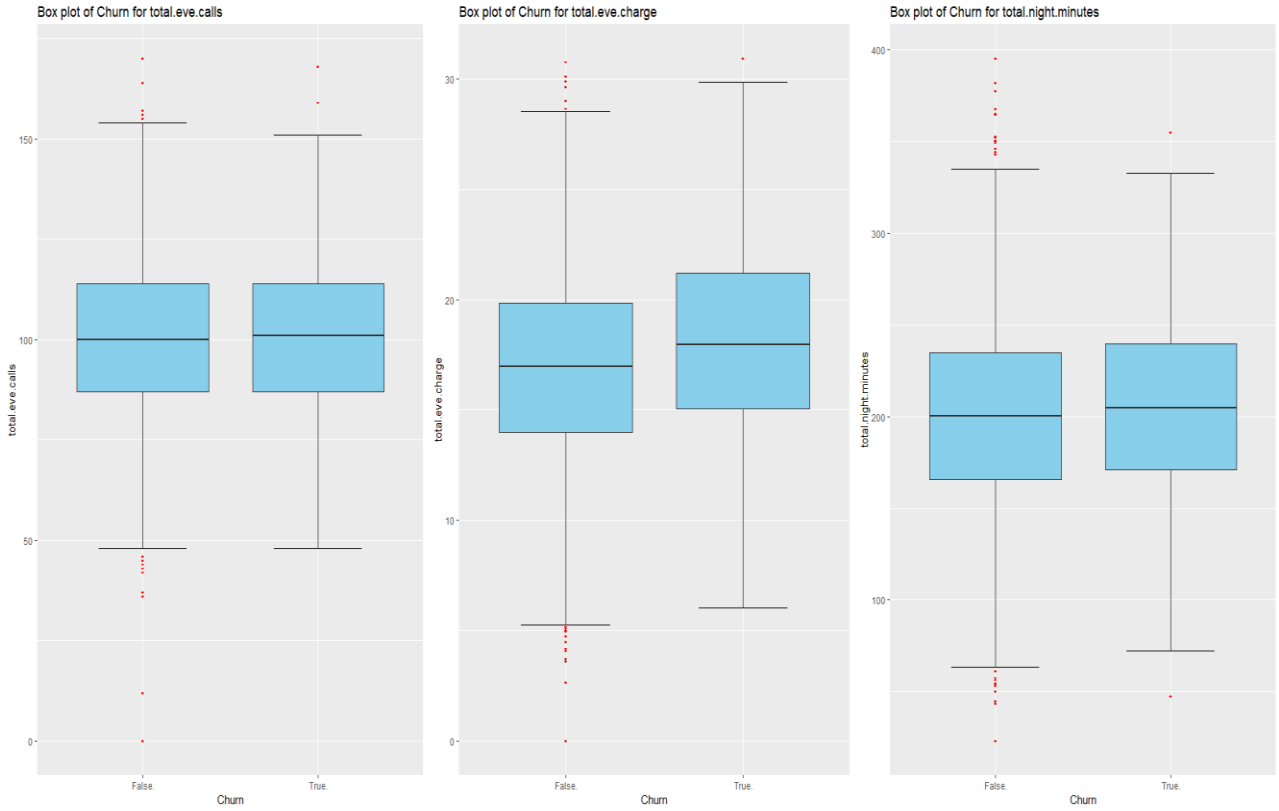
Boxplot analysis for checking outliers in R



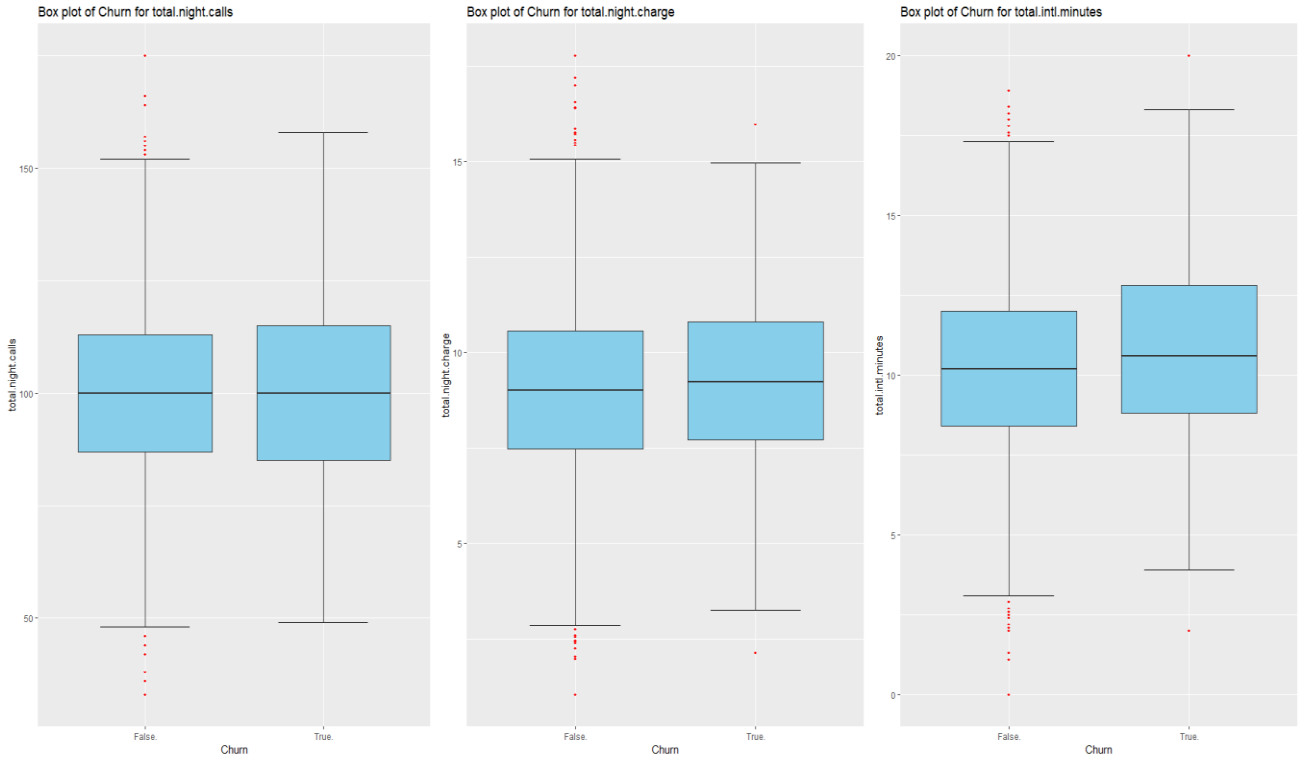
account-length, number-vmail-messages, total-day-minutes



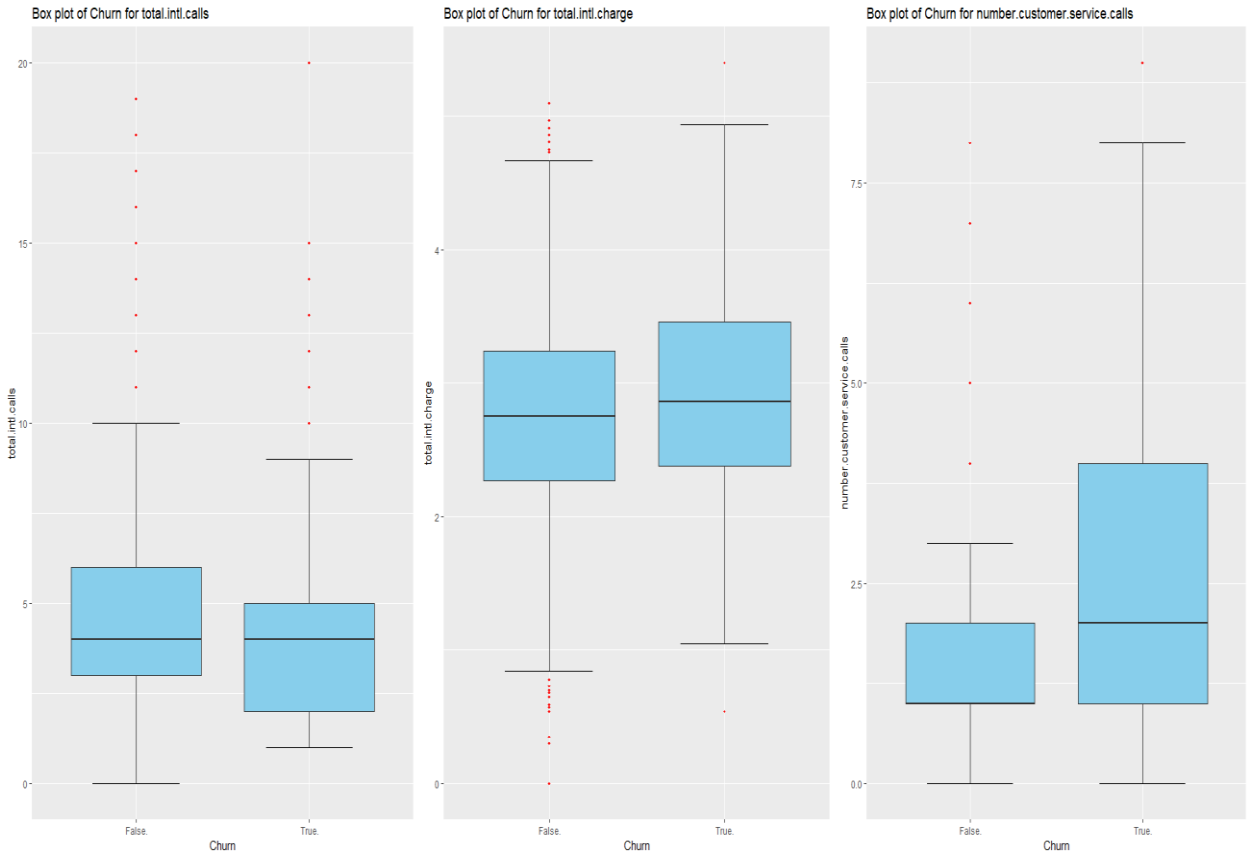
total-day-calls, total-day-charge, total-eve-minutes



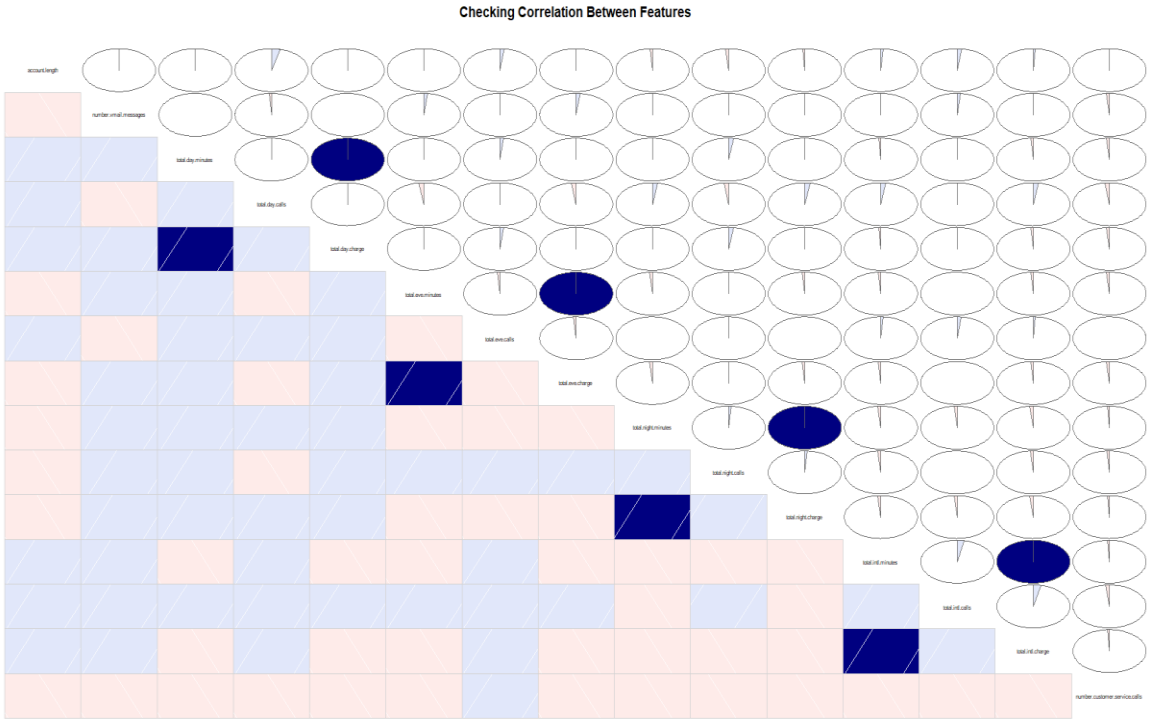
total-eve-calls, total-eve-charge, total-night-minutes



total-night-calls, total-night-charge, total-intl-minutes

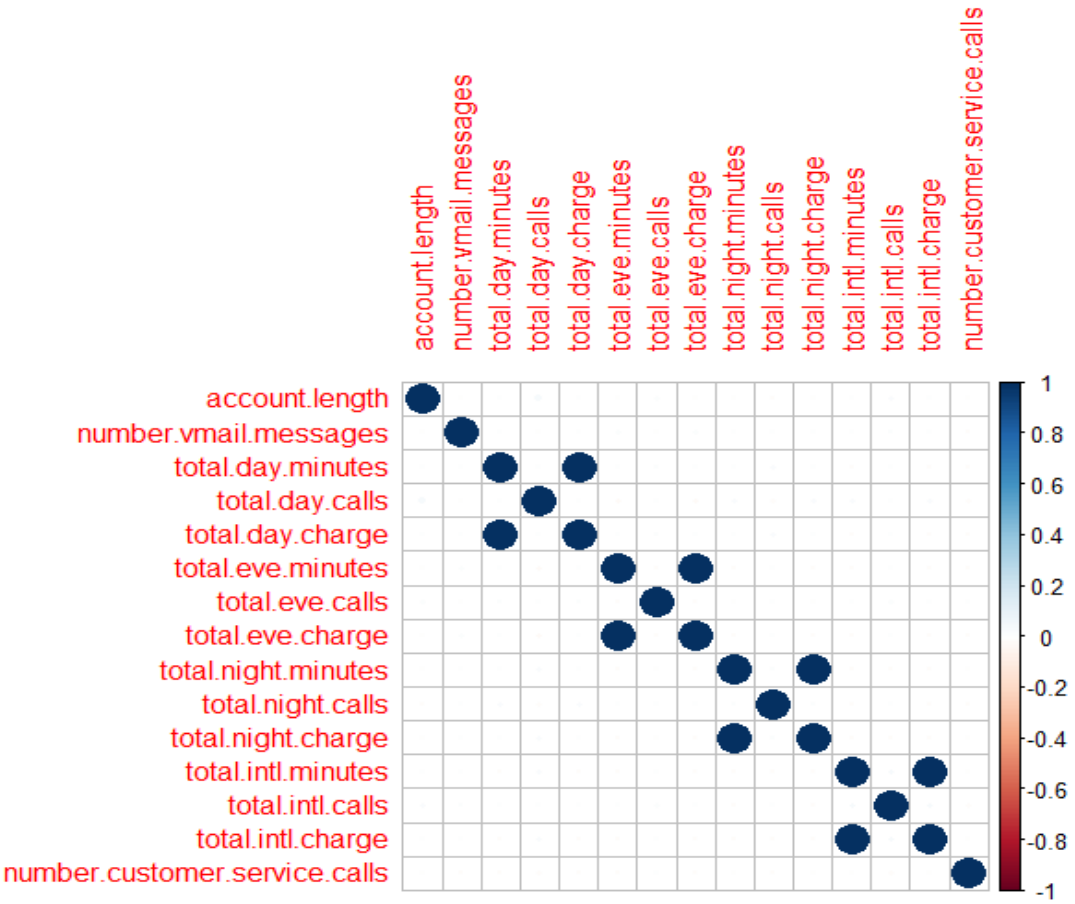


total-intl-calls, total-intl-charge, number-customer-service-calls



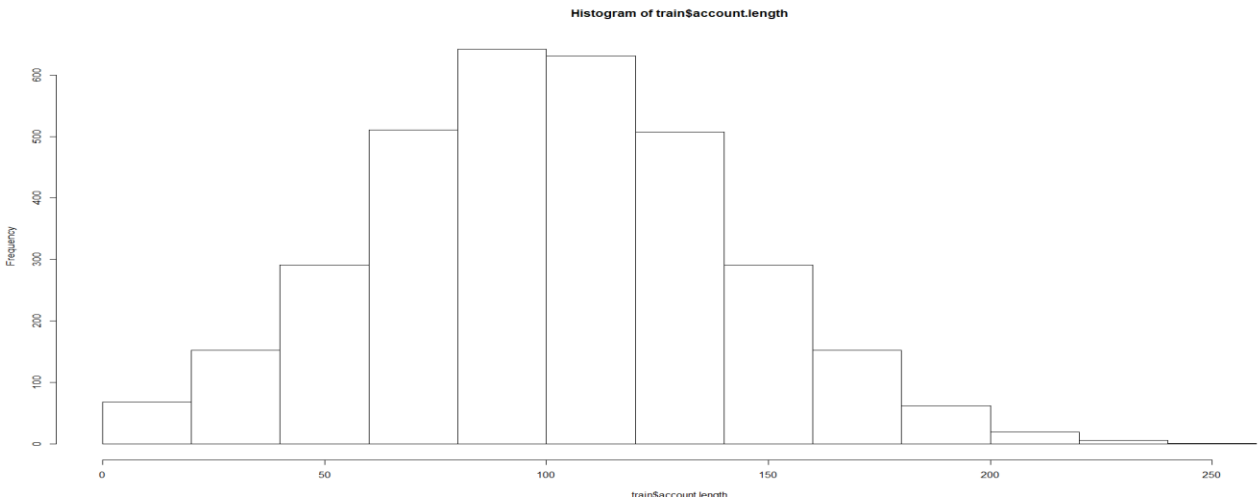
Corrgram plot for correlation analysis

In this Correlation Plot we can identify easily the highly correlated variables in R

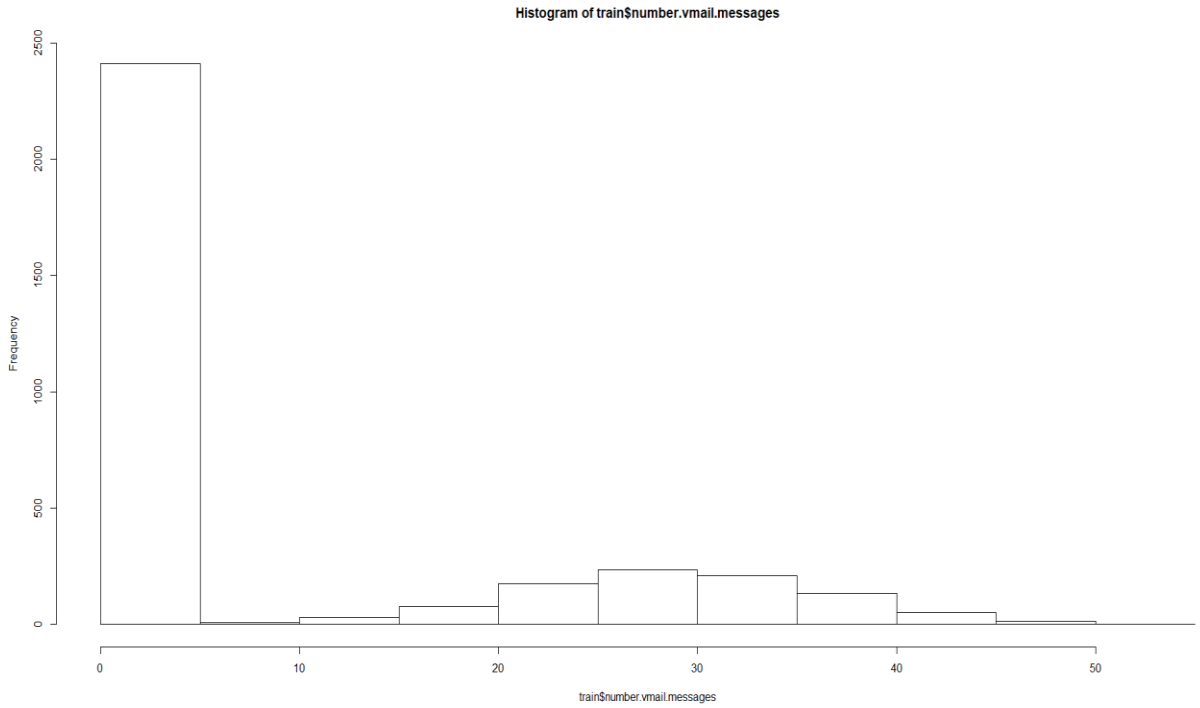


Checking for multi-collinearity existence using Correlation Plot

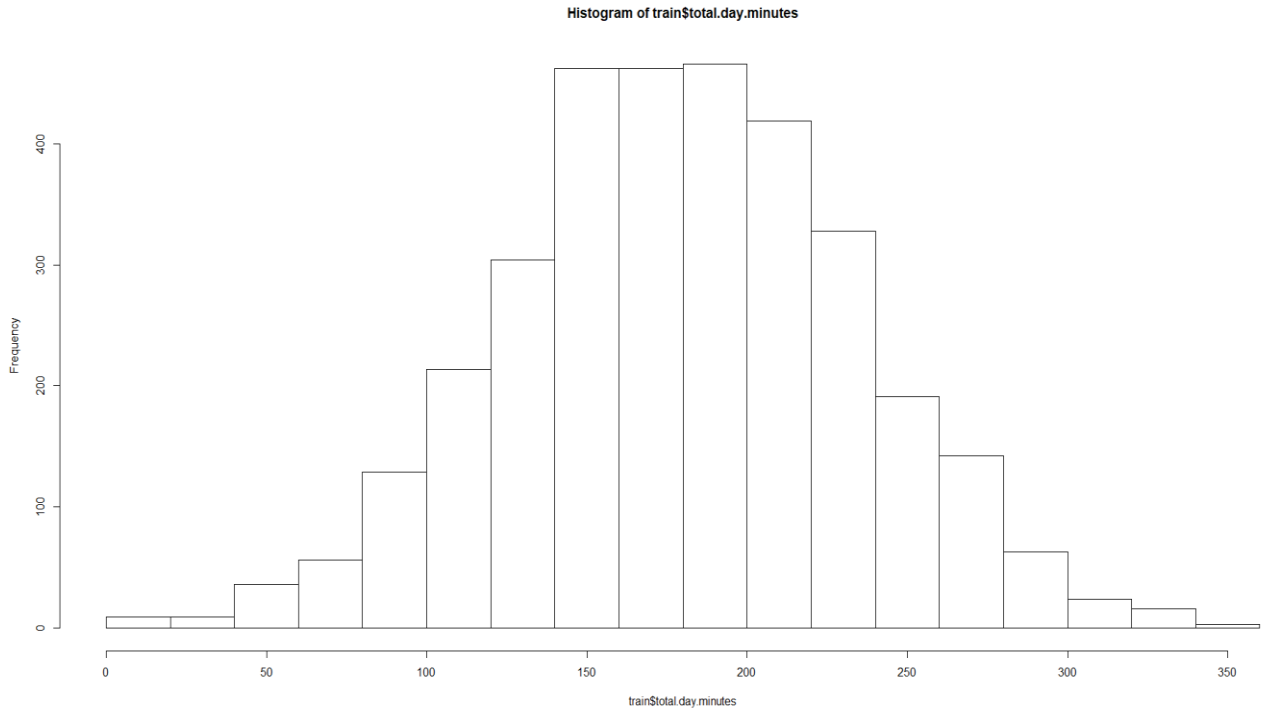
Distribution of continuous variables in R



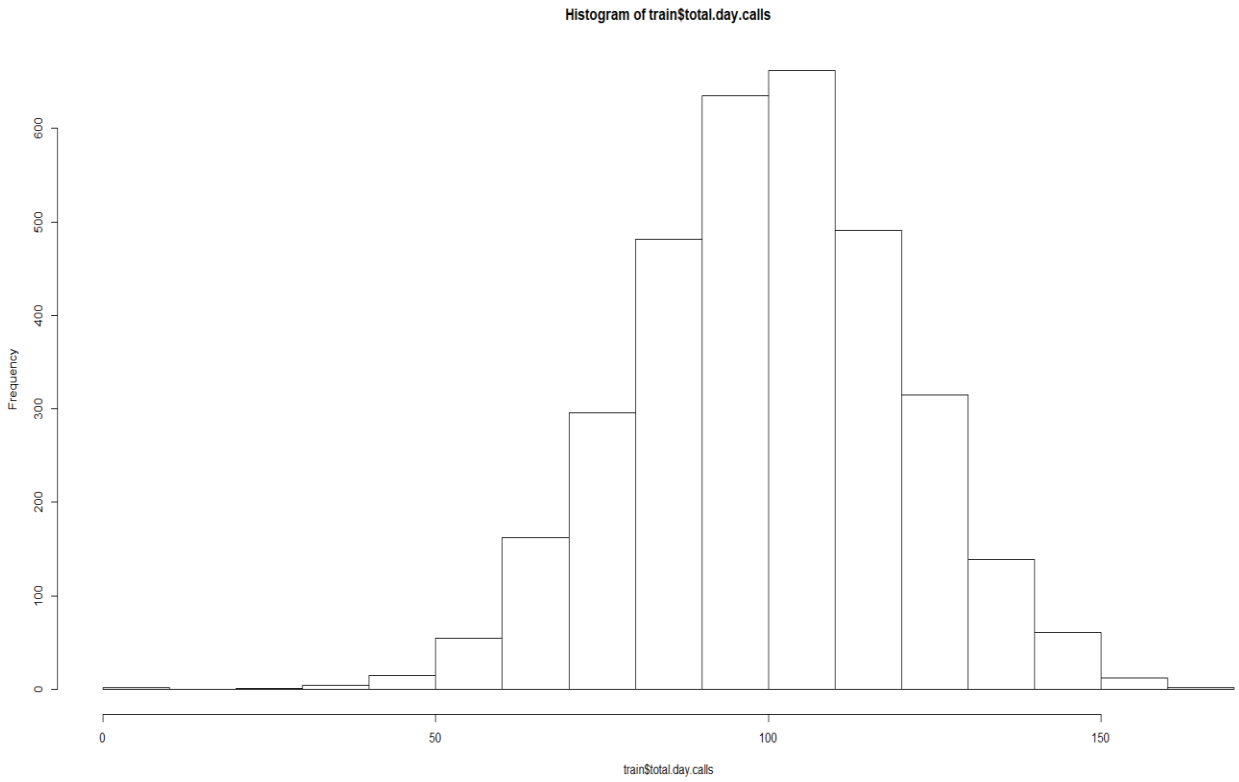
account length variable distribution of data



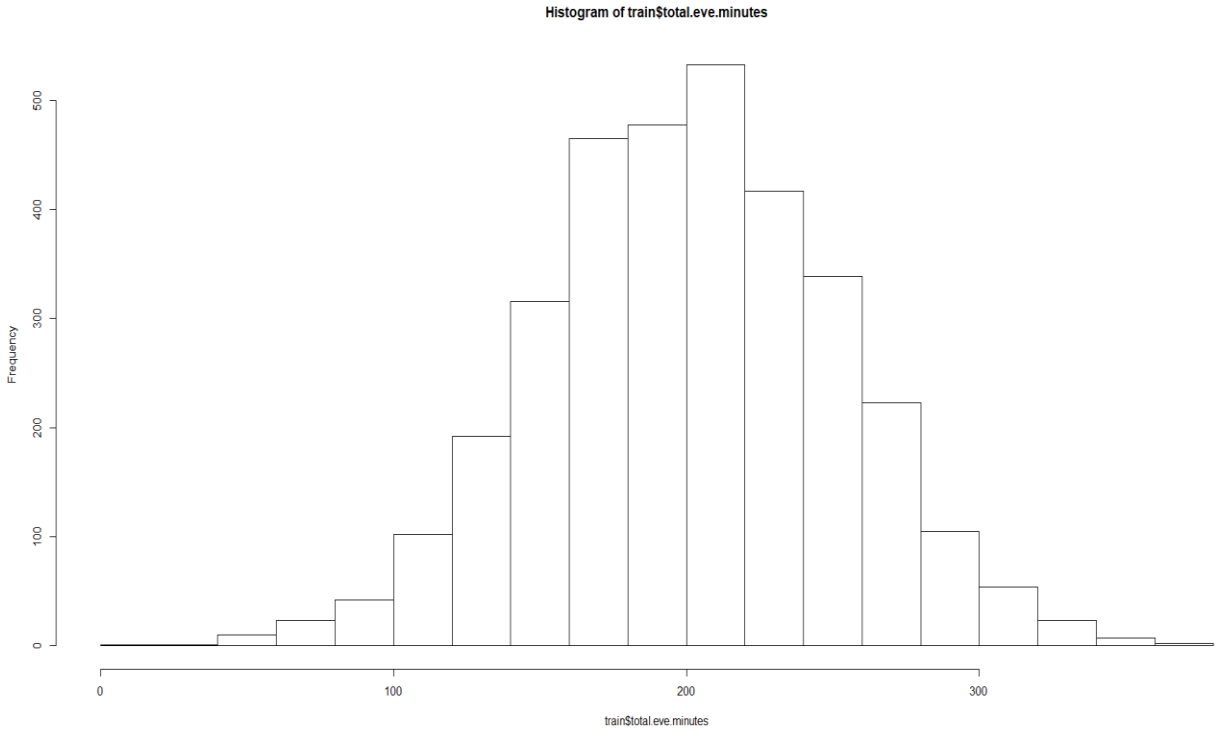
number vmail messages variable distribution of data



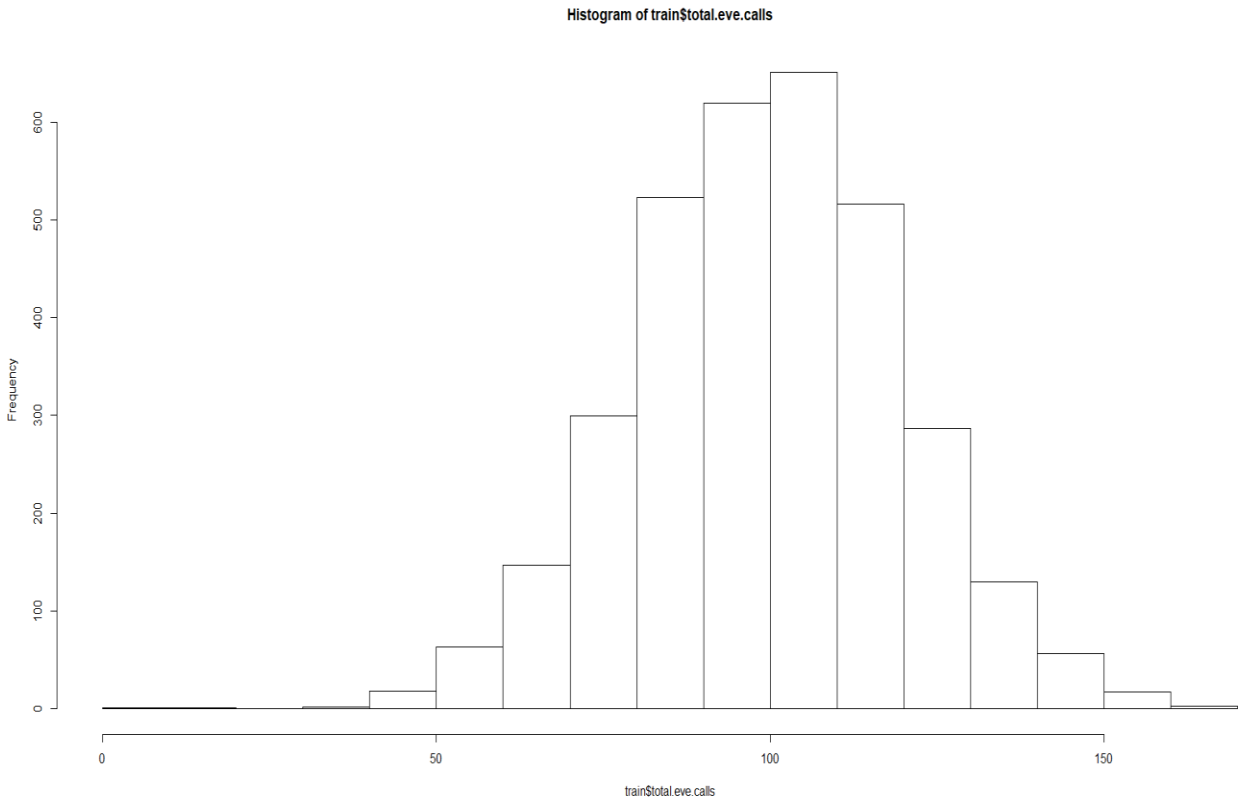
Total day minutes' variable distribution of data



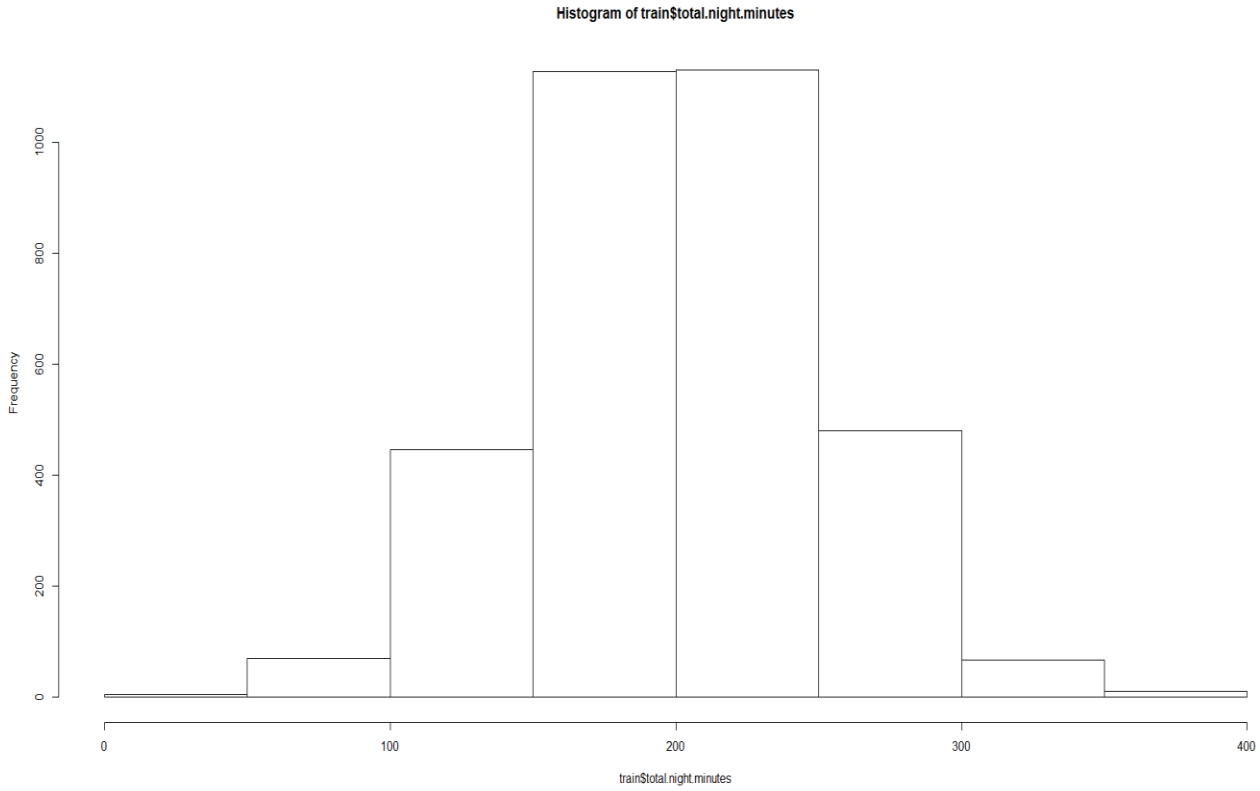
Total day calls variable distribution of data



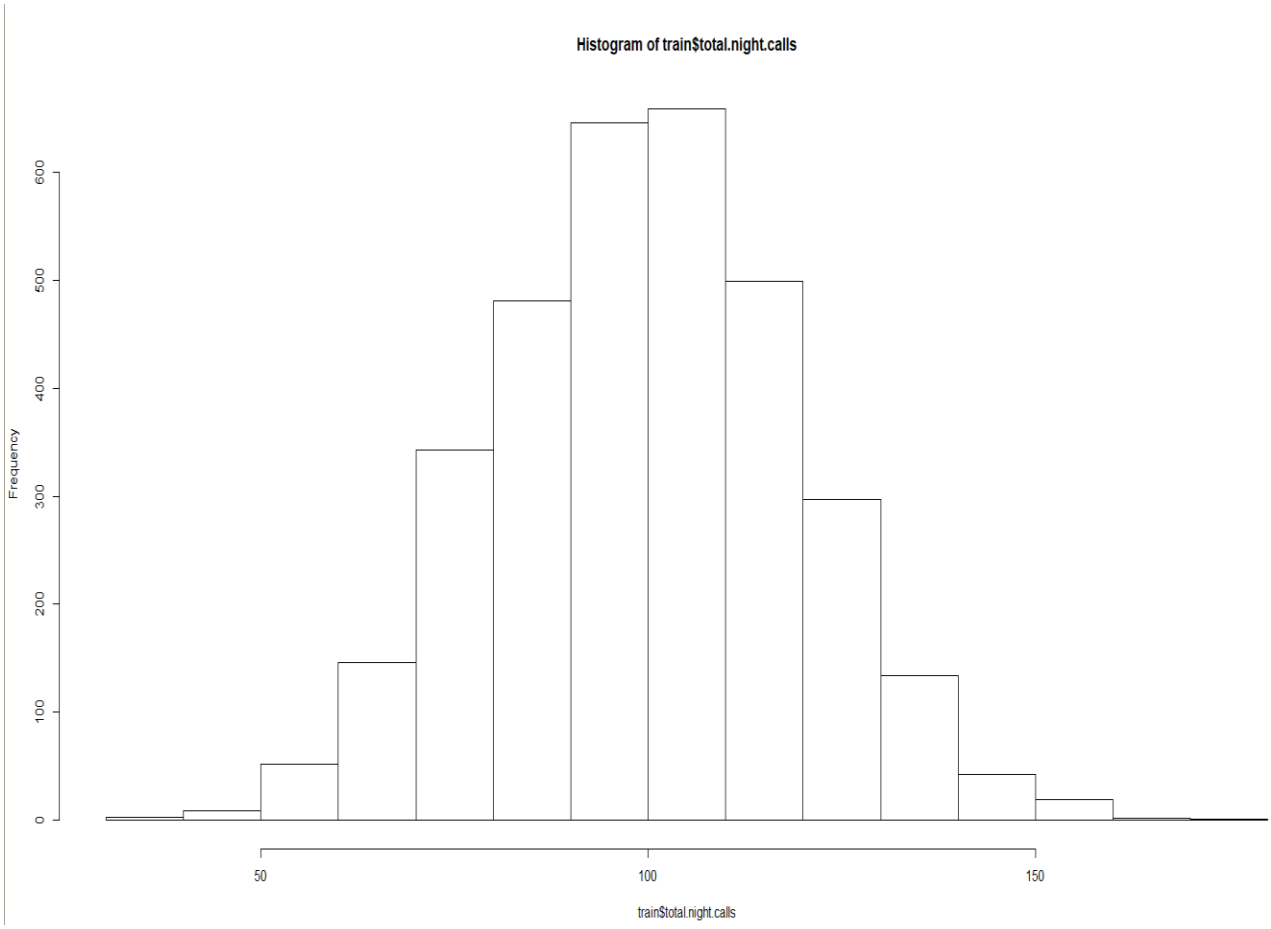
Total eve minutes' variable distribution of data



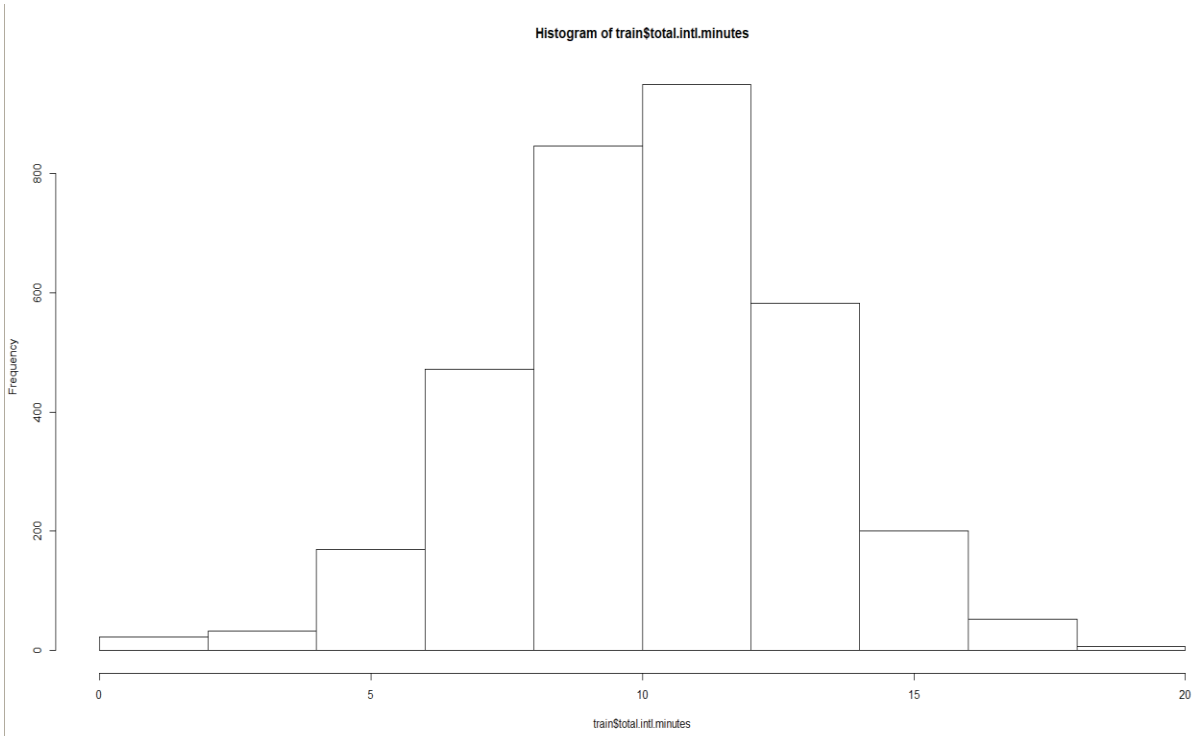
Total eve calls variable distribution of data



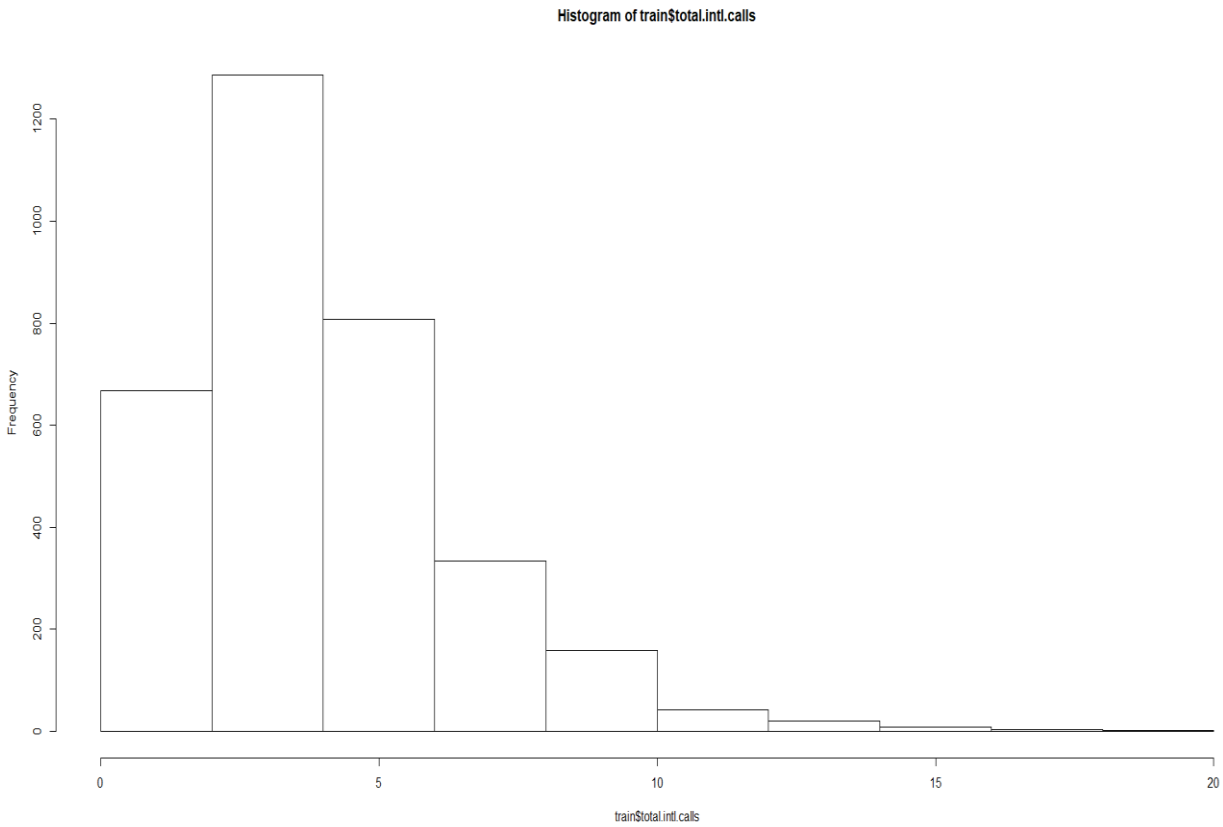
Total night minutes' variable distribution of data



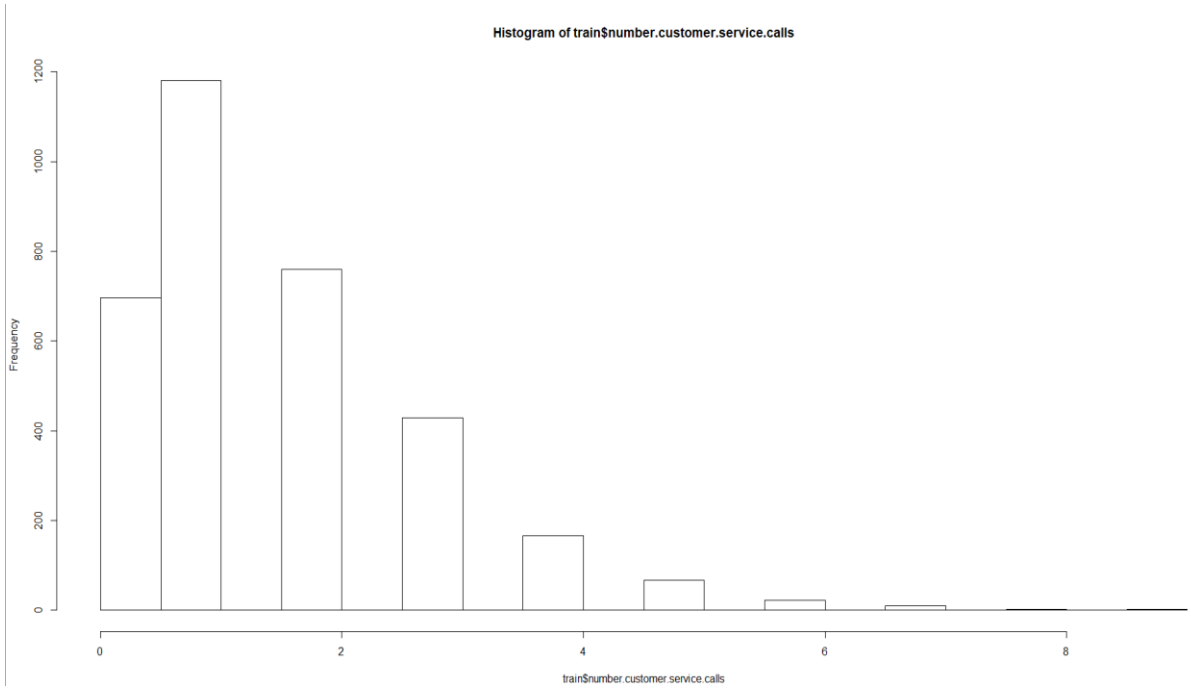
Total night calls variable distribution of data



Total international minutes' variable distribution of data



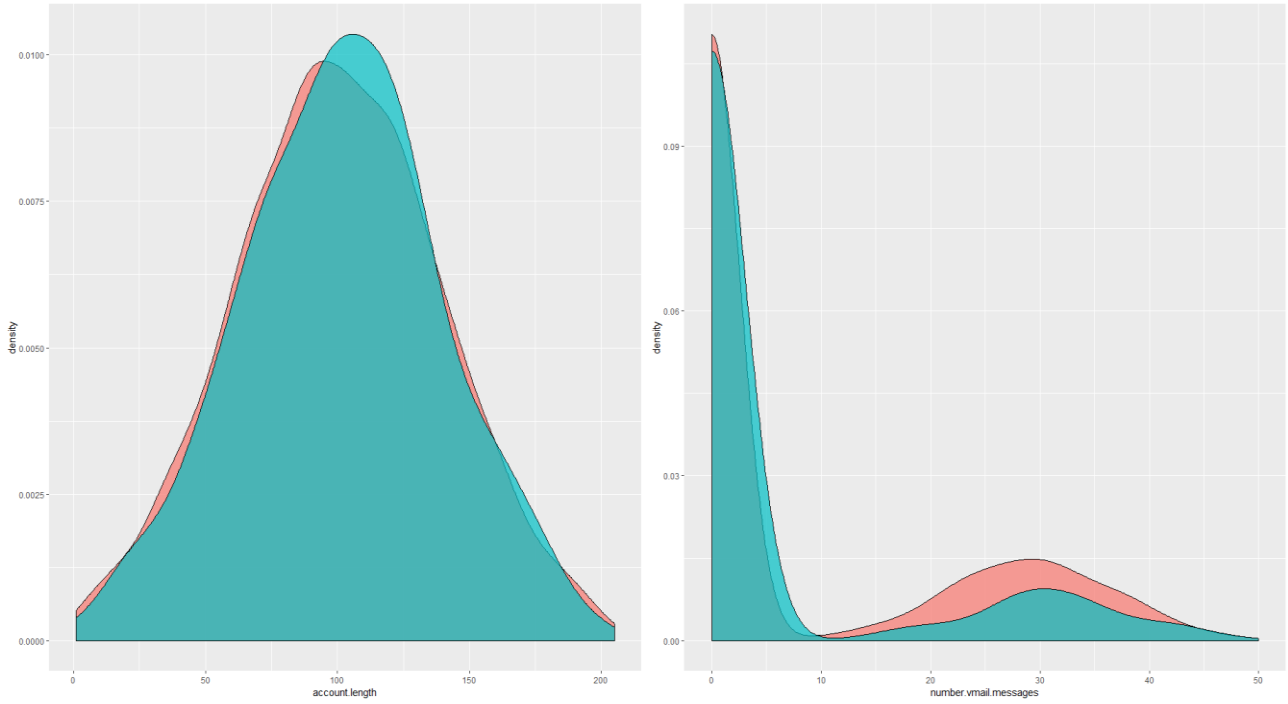
Total international calls variable distribution of data



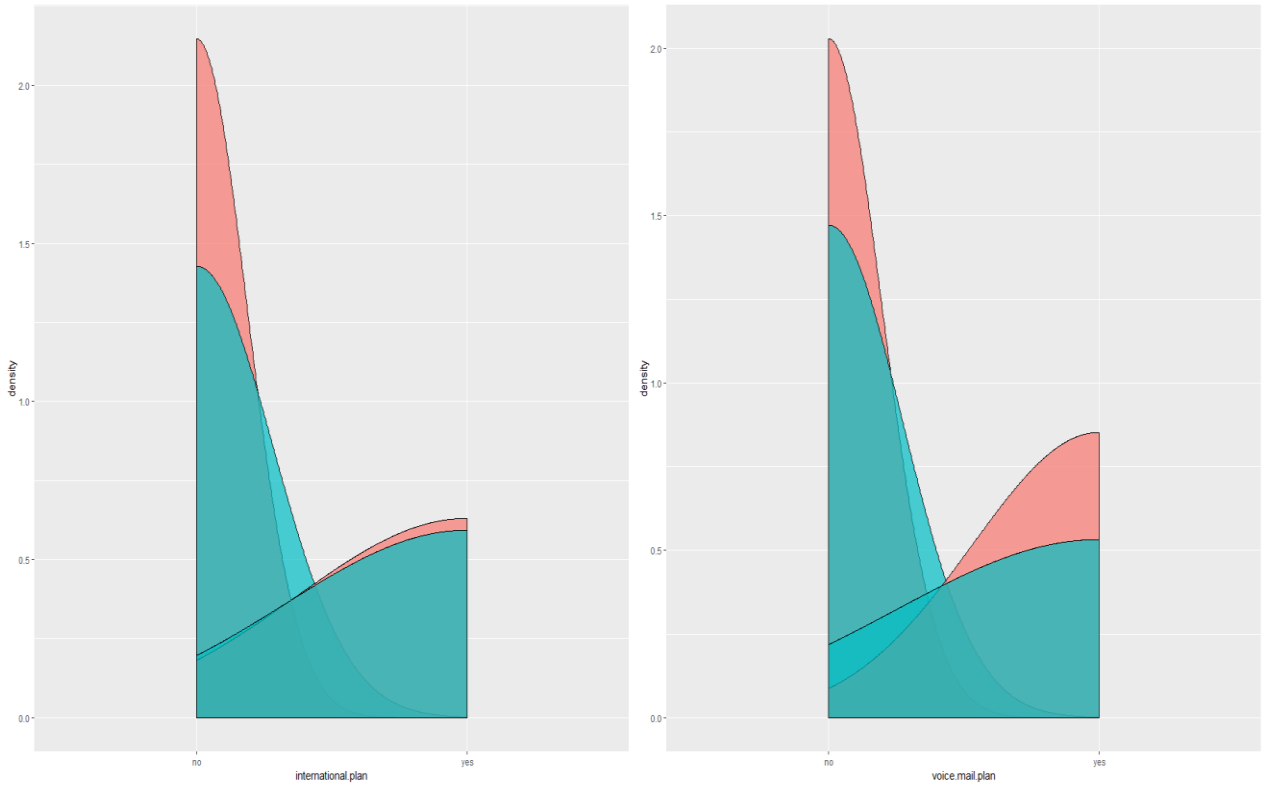
Number customer service calls variable distribution of data

- As we can observe that most of the data was uniformly distributed.

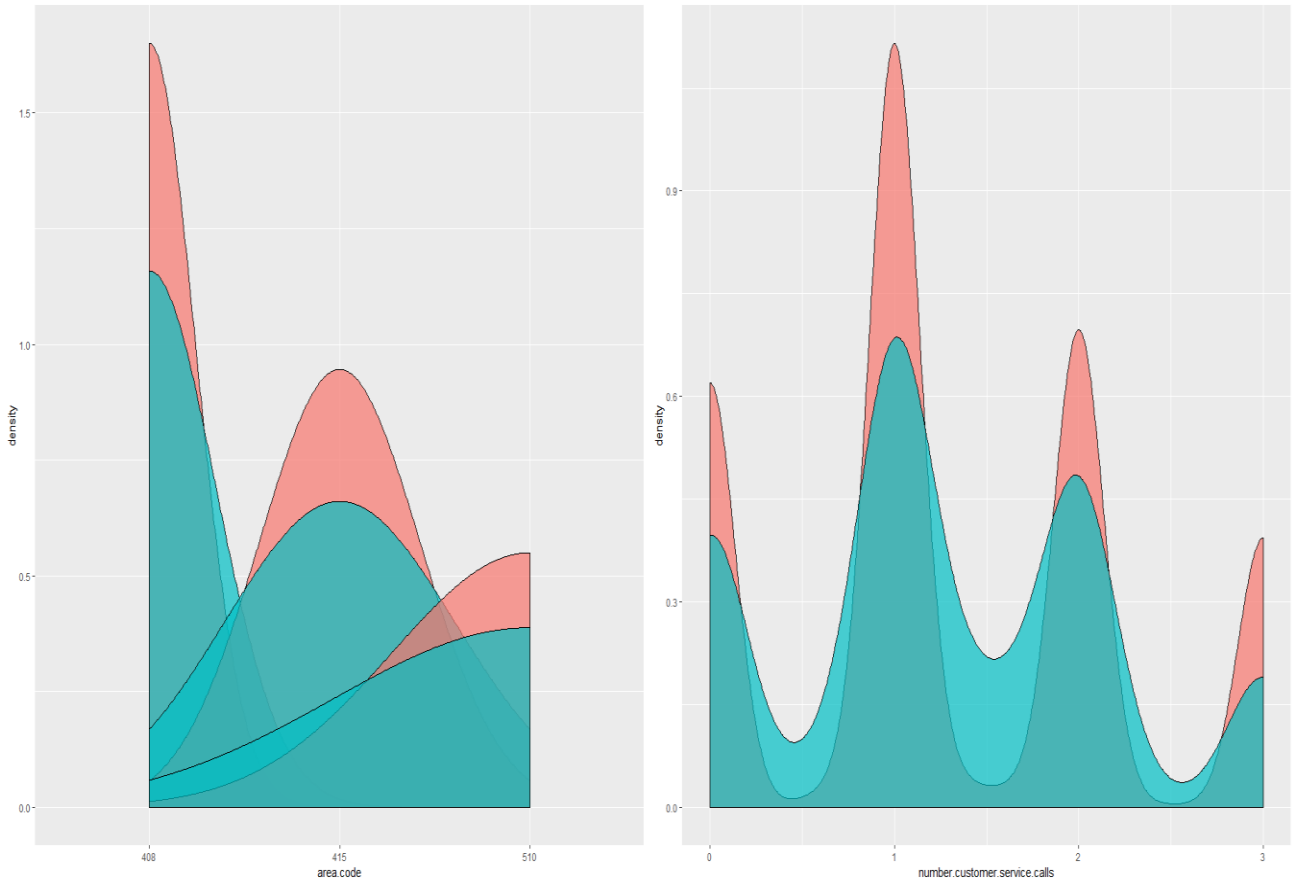
Let's Visualize some of the variables with density plots



Account length and number of voice mail messages of customers for Churn



International and voice mail plan of customers for churn



Area code and number customer service calls wise customer churn

Appendix B: Complete Python Code

Importing & Loading Standard Libraries

```
#Load the Standard Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import chi2_contingency
import seaborn as sns
from fancyimpute import KNN

# Importing the sci-kit learn package modules for model development, ev
aluation & also optimization
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, classification_report, ro
c_curve, auc


import warnings
warnings.filterwarnings("ignore")
```

```
# Setting up the working directory
os.chdir("E:\DataScienceEdwisor\PythonScripts")

#Checking the current working directory
os.getcwd()
# Loading the dataset which is in '.CSV' format i.e; (Comma-Seperated-V
alues)
train_actual = pd.read_csv("Train_data.csv")
test_actual = pd.read_csv("Test_data.csv")
# Before we make any manipulation let's create an alternate copy of our
actual train and test data sets
train_data = train_actual.copy()
test_data = test_actual.copy()
# lets Exploring few observations of the Train-dataset
train_data.head()
# Lets also Exploring few observations of the Test-dataset
test_data.head()
```

Exploratory Data Analysis

```
# Checking the Dimensions of the dataset
print("Dimensions of the training Dataset", train_data.shape)

print("\n**************************************************************
****************")

print("Dimensions of the test Dataset", test_data.shape)

print("\n**************************************************************
****************")

# Checking the total observations by combining both train & test data

data = train_data.append(test_data)

print("Total No. Observations of the combine Dataset", data.shape)

print("\n**************************************************************
****************")
# Checking the Information about the dataframe

print("Information about  Training DataFrame including the Index data-t
ype and Column data-types")

print(train_data.info())
# Checking the descriptive statistics of the dataset

print("Generates descriptive statistics for each feature")

train_data.describe()
# Column names of the dataset
train_data.columns
#Extracting Unique values and Count using a for loop on the whole data

for i in train_data.columns:
    print(i, '***************', len(train_data[i].value_counts()))
# Now lest check the churn count and precentage rate in out dataset
print(train_data['Churn'].value_counts())
```

```
print('\n**************************************')

print(train_data['Churn'].value_counts(normalize=True))

print('\n**************************************')
```

Missing Values Data Check in train & test and its Percentage

```
# Missing value analysis check for Train Dataset
total = train_data.isnull().sum().sort_values(ascending=True)
percnt = (train_data.isnull().sum()/train_data.isnull().count()*100).so
rt_values(ascending=False)
miss_train_data = pd.concat([total,percnt], axis = 1, keys=['Total_miss
_val_train','Percentage_train'])
miss_train_data
# Missing value analysis check for Test Dataset
total = test_data.isnull().sum().sort_values(ascending=True)
percnt = (test_data.isnull().sum()/test_data.isnull().count()*100).sort
_values(ascending=False)
miss_test_data = pd.concat([total,percnt], axis = 1, keys=['Total_miss_
val_test','Percentage_test'])
miss_test_data
# Before we proceed furthur let's extract categorical variables
cat_names = train_data.select_dtypes(exclude=np.number).columns.tolist(
)
cat_names.append('area code')
cat_names
# Lets Change the train and test columns to Categorical data types
train_data[cat_names] = train_data[cat_names].apply(pd.Categorical)
test_data[cat_names] = test_data[cat_names].apply(pd.Categorical)
```

Analyzing Data Through Visualization

```
# Lets analyze the target variable Churn
plt.figure(figsize=(10,8))
sns.countplot(x = train_data.Churn, palette='Set2')
plt.title('Customers Churning VS Not Churning', fontsize=22)
plt.xlabel('Customer Churn',fontsize=16)
plt.ylabel('Count',fontsize=16)
# Lets analyze the Churn of Customer's on basis of customer service cal
ls
plt.figure(figsize=(10,8))
plt.title('Customer Churning on Basis of Customer Service calss', fonts
ize=20)
sns.countplot(x='number customer service calls', hue='Churn', data=trai
n_data, palette="Set2")
#Lets also analyze through area-code wise the customer's Churn
plt.figure(figsize=(10,8))
plt.title('Customer Churning on Basis of Area Code', fontsize=22)
sns.countplot(x='area code', hue='Churn', data=train_data, palette="Set
2")
# Lets analyze the customer Churn on basis of International Plan & Voic
e mail Plan by BAR PLot Analysis
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,5))
sns.countplot(x='international plan', hue='Churn', data=train_data, ax=
ax[0], palette="Set2")
```

```
sns.countplot(x='voice mail plan', hue='Churn', data=train_data, ax=ax[
1], palette="Set2")
# Lets analyze the Cutomer Churn by State here we are using Groupby fun
ction to unstack the Categories
train_data.groupby(['state','Churn']).size().unstack().plot(kind='bar',
stacked=True, figsize=(25,10))
#Lets analyze the linearity between Total-International-Charge & Minute
s
sns.lmplot(x='total intl charge',y='total intl minutes', data=train_dat
a,
           scatter_kws={'marker':'o','color':'indianred'},
           line_kws={'linewidth':1,'color':'blue'})
#Lets analyze the linearity between the Total-Day-Charge & Minutes
sns.lmplot(x='total day charge',y='total day minutes', data=train_data,
           scatter_kws={'marker':'o','color':'indianred'},
           line_kws={'linewidth':1,'color':'blue'})
#Lets analyze the linearity between the Total-Evening-Charge & Minutes
sns.lmplot(x='total eve charge',y='total eve minutes', data=train_data,
           scatter_kws={'marker':'o','color':'indianred'},
           line_kws={'linewidth':1,'color':'blue'})
#Lets analyze the linearity between the Total-Night-Charge & Minutes
sns.lmplot(x='total night charge',y='total night minutes', data=train_d
ata,
           scatter_kws={'marker':'o','color':'indianred'},
           line_kws={'linewidth':1,'color':'blue'})
```

Boxplot Analysis for Outliers check

```
# Boxplot for checking Outliers in our dataset
f, axes = plt.subplots(5, 3, figsize=(20, 20))
# total_day_calls
sns.boxplot(x='Churn', y='total day calls', data=train_data, hue='Churn
', palette="Set2", ax=axes[0, 0])
# total_day_minutes
sns.boxplot(x='Churn', y='total day minutes', data=train_data, hue='Chu
rn', palette="Set2", ax=axes[0, 1])
# total_day_charge
sns.boxplot(x='Churn', y='total day charge', data=train_data, hue='Chur
n', palette="Set2", ax=axes[0, 2])
# total_night_calls
sns.boxplot(x='Churn', y='total night calls', data=train_data, hue='Chu
rn', palette="Set2", ax=axes[1, 0])
# total_night_minutes
sns.boxplot(x='Churn', y='total night minutes', data=train_data, hue='C
hurn', palette="Set2", ax=axes[1, 1])
# total_night_charge
sns.boxplot(x='Churn', y='total night charge', data=train_data, hue='Ch
urn', palette="Set2", ax=axes[1, 2])
# total_eve_calls
sns.boxplot(x='Churn', y='total eve calls', data=train_data, hue='Churn
', palette="Set2", ax=axes[2, 0])
# total_eve_minutes
sns.boxplot(x='Churn', y='total eve minutes', data=train_data, hue='Chu
rn', palette="Set2", ax=axes[2, 1])
# total_eve_charge
sns.boxplot(x='Churn', y='total eve charge', data=train_data, hue='Chur
n', palette="Set2", ax=axes[2, 2])
# total_intl_calls
```

```
sns.boxplot(x='Churn', y='total intl calls', data=train_data, hue='Churn', palette="Set2", ax=axes[3, 0])
# total_intl_minutes
sns.boxplot(x='Churn', y='total intl minutes', data=train_data, hue='Churn', palette="Set2", ax=axes[3, 1])
# total_intl_charge
sns.boxplot(x='Churn', y='total intl charge', data=train_data, hue='Churn', palette="Set2", ax=axes[3, 2])
# account_length
sns.boxplot(x='Churn', y='account length', data=train_data, hue='Churn', palette="Set2", ax=axes[4, 0])
# number_vmail_messages
sns.boxplot(x='Churn', y='number vmail messages', data=train_data, hue='Churn', palette="Set2", ax=axes[4, 1])
# number_customer_service_calls
sns.boxplot(x='Churn', y='number customer service calls', data=train_data, hue='Churn', palette="Set2", ax=axes[4, 2])
```

Outlier Analysis

```
# Lets seperate the numeric values becoz outlier analysis is applicable only on 'Numeric/Continous Values'
# Lets exclude the category vriables and only numeric columns will be selected her

cnames = train_data.columns[(train_data.dtypes=="float64")|(train_data.dtypes=="int64")].tolist()
print(cnames)
# Lets detect and delete outliers from data-set
# Outliers which fall above the upper fence which is 1.5*IQR and below fence 1.5*IQR will be dropped
for i in cnames:
    print(i)
    q75, q25 = np.percentile(data.loc[:,i], [75, 25])

    # iqr-Inter Quartile Range
    iqr = q75 - q25
    min = q25 - (iqr * 1.5)
    max = q75 + (iqr * 1.5)

    print(iqr)
    print(min)
    print(max)

# Replace the values with np.nan
    data = data.drop(data[data.loc[:,i] < min].index)
    data = data.drop(data[data.loc[:,i] > max].index)
# Lets check the dimension of the data dataset

print("Total No. Observations of the  Dataset", data.shape)

print("\n*************************************************************")

# As we can see that before we applied the Outlier Analysis the dimension of the dataset was
# Toatal No. Observations of the combine Dataset (5000, 21)
# After outlier Analysis the dimension of the dataset
```

```
# Total No. Observations of the  Dataset (3666, 21)
# As we can see that data is gradually reducing so here we are going to
skip outlier analysis on our train_data & test_data
# dataset which we are going to use for model development & predictions
.
```

Feature Selection

```
# Generate the Correlation matrix
corr = train_data[cnames].corr()

# Ploting using seaborn library
sns.heatmap(train_data.corr(), annot=True, cmap='RdYlGn', linewidths=0.
2)
fig=plt.gcf()
fig.set_size_inches(20,16)
plt.title("Heatmap analysis between numeric columns")
plt.show()
# Lets analyze through pair plot
sns.pairplot(train_data,hue='Churn',palette ='Set2',size=2.7,diag_kind=
'kde',diag_kws=dict(shade=True),plot_kws=dict(s=10))
plt.tight_layout()
plt.show()
```

Let's Analyze Chi2-Square Test of Independence for Categorical Variables

```
# Chi2 square test of independence for checking relation between Catego
rical variables and target varoable
# Lets save all categorical column names
cat_names = ['state', 'area code', 'international plan', 'voice mail pl
an']

print("Chi2-Square Test of Independence")
print("\n*********************************")

# loop for chi2 square test of independence
for i in cat_names:
    print(i)
    # here Chi2-Square test compares two variables in contigency table
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(train_data['Churn']
, train_data[i]))
    print(p)
    print("------------------------------")

# Here Chi2 - is the observed/actual value
# Here P -  is the pearson correlation value i.e: (p<0.05)
# Here DOF( Degrees of Freedom) = (no. of rows-1) - (no. of columns-1)
# Here EX - is the expected value
# Here Conclusion is that if the P-Value is less than (p<0.05) we rejec
t Null Hypothesis saying that 2 values depend
# on each other else we accept alternate hypothesis saying that these 2
values are independent of each other
```

Dimension Reduction

```
# Dropping the correlated variables total-day, evening, night and inter
national charge, as well state area code & phone number
# which are not carrying usefull information to explain the variance of
target variable lets drop from both train&test dataset
train_data = train_data.drop(columns=['state','phone number','area code
','total day charge',
```

```
                                          'total eve charge', 'total night
charge', 'total intl charge'])
test_data = test_data.drop(columns=['state','phone number','area code',
'total day charge',
                                        'total eve charge', 'total night ch
arge', 'total intl charge'])
```

Feature Selection and confirming our final Continuous & Categorical Features

```
# Lets re-check the numeric column variables and ctegorical columns var
iables
# Lets update them as we have dropped some of the features
cnames = ['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']
print(cnames)
print('*********************************************************')
# Lets recheck the categorical names now
cat_names = ['international plan', 'voice mail plan','Churn']

print(cat_names)
```

Assigning levels to Categorical Columns

```
#Lets Handle Categorical Columns now by assigning levels (0 & 1)
cat_names = train_data.columns[train_data.dtypes == 'category']
for i in cat_names:
    #print(i)
    train_data[i] = train_data[i].cat.codes
    test_data[i] = test_data[i].cat.codes
```

Let's Do Some More Exploration of our data

```
# Lets apply groupby with International-Plan
intl_plan=train_data.groupby("international plan").size()
intl_plan
# Lets now analyze how many customers have subscribed and not subscribe
d to International plan in percent
print("Subscribed to International-Plan in percent:\t{}".format((intl_p
lan[0]/3333)*100))
print("Not Subscribed to International-Plan in percent:\t{}".format((in
tl_plan[1]/3333)*100))
# Lets apply the groupby with Voice mail plan now
vmail_plan = train_data.groupby('voice mail plan').size()
vmail_plan
# Lets now analyze how many customers have subscribed and not subscribe
d to Voice Mail Plan in percent
print("Subscribed to International-Plan in percent:\t{}".format((vmail_
plan[1]/3333)*100))
print("Not Subscribed to International-Plan in percent:\t{}".format((vm
ail_plan[0]/3333)*100))
# Lets groupby customer service calls now
custmr_calls = data.groupby('number customer service calls').size()
custmr_calls
# Lets analyze the count of customer service calls
train_data['number customer service calls'].hist(bins=500,figsize=(10,8
))
plt.title("Number of Customer Service Calls")
plt.xlabel("Customer Service Calls")
plt.ylabel("Count of Customer Service Calls")
```

```
plt.tight_layout()
plt.show()
# Lets see Account Length of customers here
Account_Length = train_data.groupby(['account length']).size()
#Account_Length # has length of 212
# Lets plot Histogram to analyze the account length
train_data['account length'].hist(bins=500,figsize=(10,8))
plt.title("Account length of customers ")
plt.xlabel("Customer Account length")
plt.ylabel("Count of Customer Account Length")
plt.tight_layout()
plt.show()
# Now Lets check the CHURN of Customers in Percent
Churn=train_data.groupby(['Churn']).size()
Churn # Here 0- Means 'False.'- > 'No', & Here 1- Means 'True.'- > 'Yes
'
# Lets see the Percentage of Churn
print (" Negative Chrun in percent:{}".format((Churn[0]/3333)*100))
print (" Positive Chrun in percent:{}".format((Churn[1]/3333)*100))
# Lets see churn by international plan
Intl_Churn = train_data.groupby(['international plan','Churn']).size()
Intl_Churn
# Lets see churn by voice mail plan
Vmail_Churn = train_data.groupby(['voice mail plan', 'Churn']).size()
Vmail_Churn
# Lets see churn by customer service calls
Custserv_Chrun=data.groupby(['number customer service calls','Churn']).
size()
Custserv_Chrun
# Lets Plot to analyze the customer service calls as per Churn
Custserv_Chrun.plot(kind= 'bar', figsize=(10,8))
plt.title('Churn By Customer Service calls Breakdown')
plt.xlabel('Customer Serv calls wise Churn', fontsize=18)
plt.ylabel('Count of Customer Serv calls wise Churn', fontsize=18)
plt.show()
```

Checking Distribution of Variables/ Normality Check

```
# Lets Check the Data Distribution of Continous variables
train_data[cnames].hist(figsize=(20,20), alpha=0.7)
plt.show()
# Lets analyze better by also plotting density plot over histogram plot
# histogram and Density Plot togeather for checking distribution of our
variables
f, axes = plt.subplots(4, 3, figsize=(20, 20))
#account length
sns.distplot(train_data['account length'], hist=True, bins='auto',kde=T
rue,color = 'darkblue', ax=axes[0, 0],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#number vmail messages
sns.distplot(train_data['number vmail messages'], hist=True, bins='auto
',ax=axes[0, 1],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total day minutes
sns.distplot(train_data['total day minutes'], hist=True, bins='auto',ax
=axes[0, 2],
             hist_kws={'edgecolor':'black'},
```

```
             kde_kws={'linewidth': 4})
#total day calls
sns.distplot(train_data['total day calls'], hist=True, bins='auto',ax=a
xes[1, 0],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total eve minutes
sns.distplot(train_data['total eve minutes'], hist=True, bins='auto',ax
=axes[1, 1],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total eve calls
sns.distplot(train_data['total eve calls'], hist=True, bins='auto',ax=a
xes[1, 2],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total night minutes
sns.distplot(train_data['total night minutes'], hist=True, bins='auto',
ax=axes[2, 0],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total night calls
sns.distplot(train_data['total night calls'], hist=True, bins='auto',ax
=axes[2, 1],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total international minutes
sns.distplot(train_data['total intl minutes'], hist=True, bins='auto',a
x=axes[2, 2],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#total international calls
sns.distplot(train_data['total intl calls'], hist=True, bins='auto',ax=
axes[3, 0],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
#number customer service calls
sns.distplot(train_data['number customer service calls'], hist=True, bi
ns='auto',ax=axes[3, 1],
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

Feature Scaling and applying Standardization/Z-Score Method

```
# Standardization/ Z-Score Method
stand_zs = ['account length', 'number vmail messages', 'total day minut
es', 'total day calls',
               'total eve minutes', 'total eve calls', 'total night min
utes', 'total night calls',
               'total intl minutes', 'total intl calls', 'number custom
er service calls']

for i in stand_zs:
    print(i)
    train_data[i] = (train_data[i] - train_data[i].mean())/train_data[i
].std()
# Standardization/ Z-Score Method
stand_zs = ['account length', 'number vmail messages', 'total day minut
es', 'total day calls',
```

```
                'total eve minutes', 'total eve calls', 'total night min
utes', 'total night calls',
                'total intl minutes', 'total intl calls', 'number custom
er service calls']

for i in stand_zs:
    print(i)
    test_data[i] = (test_data[i] - test_data[i].mean())/test_data[i].st
d()
# Splitting the data into Train and Test Sets
X_train = train_data.drop('Churn', axis = 1)
y_train = train_data['Churn']
X_test = test_data.drop('Churn', axis = 1)
y_test = test_data['Churn']
# Lets print out the X_train, y_train, X_test and y_test
print('X_train values------>', X_train.shape)
print('y_train values------>', y_train.shape)
print('X_test values------->', X_test.shape)
print('y_test values------->', y_test.shape)
# As we can see there is target class imbalance problem, training value
s before applying Smote
y_train.value_counts()
```

SMOTE (Synthetic Minority Over-Sampling Technique)

```
# Using SMOTE
from imblearn.over_sampling import SMOTE
SMT = SMOTE()
X_train_balanced ,y_train_balanced = SMT.fit_sample(X_train, y_train)
print('X_training data set after SMOTE--->', X_train_balanced.shape)
print('y_training data set after SMOTE--->', y_train_balanced.shape)
```

MODEL DEVELOPMENT

```
# Lets define our  Prediction Function to fit and make predictions
def predct_func(classification_models, features, comparison):
    ''' Here we are going to fit our train data by passing
        to the funtion predct_func and predict our X_test
        data which is our new test cases here and it will result
        out classification-report and all the error metrics of the
        particular model used here.
    '''
    # Lets fit the model first
    classification_models.fit(features, comparison)
    # Predict new test cases
    predicted_vals = classification_models.predict(X_test)
    # Lets apply K-Fold Croos Validatiion CV=10
    KVC = cross_val_score(estimator=classification_models, X=features,
y=comparison,cv=10)
    KFoldCross_Accuracies = KVC.mean()
    print('K Fold Crossvalidation Accuracy------->', KFoldCross_Accurac
ies)
    print()
    # Generates the classification report of the model
    print("************Classification Report************")
    print()
    class_report = classification_report(y_test,predicted_vals)
    print(class_report)
    # Generate the Confusion Matrix of the Model
    print("************Confusion Matrix******************")
    print()
```

```
    CM = confusion_matrix(y_test, predicted_vals)
    print(CM)
```

```
# Lets Define another function for Evaluation of out models
def eval_model(actual_vals, prediction_vals):
    ''' Function for evaluation of error metrics
        generates confusion matrix and results out
        False Positive Rate, False Negative Rate,
        Sensitivity/TruePositiveRate/Recall &
        specificity/TrueNegativeRate of models
    '''

    CM = pd.crosstab(actual_vals, prediction_vals)
    TN = CM.iloc[0,0]
    FN = CM.iloc[1,0]
    TP = CM.iloc[1,1]
    FP = CM.iloc[0,1]
    print()

    # Lets evaluate Error Metrics of the model algorithms
    print("<---------------ERROR METRICS-------------->")
    print()
    # False Negative Rate
    print("False Negative Rate-------------->",  (FN*100)/(FN+TP))
    print()
    # False Positive Rate
    print("False Positive Rate-------------->",  (FP*100)/(FP+TN))
    print()
    # Sensitivity
    print("Sensitivity/TPR/Recall----------->",  (TP*100)/(TP+FN))
    print()
    # Specificity
    print("Specificity/TNR------------------>",  (TN*100)/(TN+FP))
```

```
# Lets Develop Decision Tree Model
DT_Model = DecisionTreeClassifier(criterion='entropy',random_state=100)
predct_func(DT_Model, X_train_balanced, y_train_balanced)
# Lets predict new test cases
DT_Predictions = DT_Model.predict(X_test)
# Now Lets evaluate Error Metrics for Decision Tree model
eval_model(y_test, DT_Predictions)
#ROC curve for false positive rate-fpr, true positive rate-tpr
# The ROC-Curve is the plot between sensitivity and (1-specificity) is
also known as False positive rate
# and Sensitivity is also known as True Positive rate
# ROC-AUC Curve for Decision Tree MOdel
fpr, tpr, thresholds_DT = roc_curve(y_test, DT_Predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
# Lets Develop Random Forest Model
RF_Model = RandomForestClassifier(n_estimators=20, criterion='entropy',
random_state=100)
predct_func(RF_Model, X_train_balanced, y_train_balanced)
# Lets Predict new test cases
RF_Predictions = RF_Model.predict(X_test)
# Lets Evaluate Error Metrics for Random Forest Model
eval_model(y_test, RF_Predictions)
# ROC-AUC Curve for Random Forest Model
fpr, tpr, thresholds_RF = roc_curve(y_test, RF_Predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# Lets Develop Logistic Regression Model
LR_Model = LogisticRegression()
predct_func(LR_Model, X_train_balanced, y_train_balanced)
# Lets Predict New Test Cases
LR_Predictions = LR_Model.predict(X_test)
# Now lets evaluate Error Metrics For Logistic Regression Model
eval_model(y_test, LR_Predictions)
# ROC-AUC Curve for Logistic Regression Model
fpr, tpr, thresholds_RF = roc_curve(y_test, LR_Predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# Lets Develop KNN Model Now
KNN_Model = KNeighborsClassifier(n_neighbors=5)
predct_func(KNN_Model, X_train_balanced, y_train_balanced)
# Lets Predict New Test Cases
KNN_Predictions = KNN_Model.predict(X_test)
# Now lets evaluate Error Metrics For K-Nearest-Neighbor Model
eval_model(y_test, KNN_Predictions)
# ROC-AUC Curve for K-Nearest-Neighbor Model
fpr, tpr, thresholds_RF = roc_curve(y_test, KNN_Predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
# Lets Develop Naive Bayes Model
NB_Model  = GaussianNB()
predct_func(NB_Model, X_train_balanced, y_train_balanced)
# Lets Predict new test cases
NB_Predictions = NB_Model.predict(X_test)
# Now lets evaluate Error Metrics For Naive Bayes Model
eval_model(y_test, NB_Predictions)
# ROC-AUC Curve for Naive Bayes Model
fpr, tpr, thresholds_RF = roc_curve(y_test, NB_Predictions)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Hyperparameter tuning using GridSearchCV

```
#Hyper Parameter Tuning for Decision Tree Model
from sklearn.model_selection import GridSearchCV

# Finding Best Parameters for Random Forest Model
#dt_model = DecisionTreeClassifier()

#params_grid = [{'criterion': ['entropy','gini'],
#                'max_depth':[50,100,150,200,250,300,350]}]

#grid_search = GridSearchCV(estimator=dt_model, param_grid=params_grid,
#                           scoring='recall', cv=10, n_jobs=-1)

#grid_search = grid_search.fit(X_train_balanced, y_train_balanced)
#best_accuracy = grid_search.best_score_
#print(best_accuracy)
#best_parameters = grid_search.best_params_
#print(best_parameters)

#accuracy_score = 0.944912280702
#best_parameters = {'criterion': 'gini', 'max_depth': 100}
# Now lets apply the tuned parameters
#dt_model = DecisionTreeClassifier(criterion='gini',max_depth=100,random_st
ate=100)
#predct_func(dt_model, X_train_balanced, y_train_balanced)
# lets predict predict new test cases on tuned decision tree model
#dt_Predictions  = dt_model.predict(X_test)
# Lets evaluate error metrics on our tuned decision tree model
#eval_model(y_test, dt_Predictions)
```

```
#<---------------ERROR METRICS-------------->

#False Negative Rate--------------> 20.9821428571

#False Positive Rate--------------> 13.86001386

#Sensitivity/TPR/Recall-----------> 79.0178571429

#Specificity/TNR------------------> 86.13998614

# ROC-AUC Curve for tuned - decision tree
#fpr, tpr, thresholds_RF = roc_curve(y_test, dt_Predictions)
#roc_auc = auc(fpr, tpr)
#plt.title('Receiver Operating Characteristic')
#plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
#plt.legend(loc='lower right')
#plt.plot([0,1],[0,1],'r--')
#plt.xlim([0.0,1.0])
#plt.ylim([0.0,1.0])
#plt.ylabel('True Positive Rate')
#plt.xlabel('False Positive Rate')
#plt.show()
```

```
#Hyper Parameter Tuning for Random Forest Model
#from sklearn.model_selection import GridSearchCV

# Finding Best Parameters for Random Forest Model
#rf_model = RandomForestClassifier()

#params_grid = [{'n_estimators':[100,200,300,400,500,800,1000],
#                'criterion': ['entropy','gini'],
#                'max_depth':[50,100,150,200,250,300,350]}]

#grid_search = GridSearchCV(estimator=rf_model, param_grid=params_grid,
#                           scoring='recall', cv=10, n_jobs=-1)

#grid_search = grid_search.fit(X_train_balanced, y_train_balanced)
#best_accuracy = grid_search.best_score_
#print(best_accuracy)
#best_parameters = grid_search.best_params_
#print(best_parameters)

#accuracy_score = 0.964912280702
#best_parameters = {'criterion': 'entropy', 'max_depth': 50, 'n_estimators'
: 300}
# lets run Tuned Random Forest Model on our prediction function
```

```
#rf_model = RandomForestClassifier(criterion='entropy', n_estimators=300, m
ax_depth =500, random_state=100)

#predct_func(rf_model, X_train_balanced, y_train_balanced)

## K Fold Crossvalidation Accuracy-------> 0.966666666667
# Now lets Predict new test cases for Tuned Random Forest Final  Prediction
s
#rf_Predictions = rf_model.predict(X_test)
# Now Lets Evaluate Error Metrics for Tuned Randoom Forest Model
#eval_model(y_test, rf_Predictions)

# Final Model Error metrics of FINAL MODEL Random Forest

#<---------------ERROR METRICS-------------->

#False Negative Rate--------------> 15.625

#False Positive Rate--------------> 9.21690921691

#Sensitivity/TPR/Recall-----------> 84.375

#Specificity/TNR------------------> 90.7830907831

# we can observe that the False Negative Rate has Reduced compare to previo
us RF_Model and Tuned rf_model is working well.
# ROC-AUC Curve for Tuned - Random Forest Model
#fpr, tpr, thresholds_RF = roc_curve(y_test, rf_Predictions)
#roc_auc = auc(fpr, tpr)
#plt.title('Receiver Operating Characteristic')
#plt.plot(fpr, tpr, 'b',label='AUC curve (area = %0.2f)'% roc_auc)
#plt.legend(loc='lower right')
#plt.plot([0,1],[0,1],'r--')
#plt.xlim([0.0,1.0])
#plt.ylim([0.0,1.0])
#plt.ylabel('True Positive Rate')
#plt.xlabel('False Positive Rate')
#plt.show()
# Lets Plot Feature importances
#features = train_data.drop(["Churn"], axis=1).columns
#fig = plt.figure(figsize=(20, 18))
#ax = fig.add_subplot(111)
#FI = pd.DataFrame(rf_model.feature_importances_, columns=["importance"])
#FI["labels"] = features
#FI.sort_values("importance", inplace=True, ascending=False)
#display(FI.head(5))
#index = np.arange(len(rf_model.feature_importances_))
```

```
#bar_width = 0.5
#rects = plt.barh(index , FI["importance"], bar_width, alpha=0.4, color='b'
, label='Main')
#plt.yticks(index, FI["labels"])
#plt.show()
# Setting up the working directory
os.chdir("E:\DataScienceEdwisor\PROJECT-1\Python")

# Lets Save the final results back to hard disk
# Writing a csv (output) Training & Test Data-set

train_data.to_csv("train_df_final_data.csv", index = False)
test_data.to_csv("test_df_final_data.csv", index = False)
```

Appendix C: Complete R Code

Lets Clear the Environment First

rm(list = ls(all=T))

Setting up the Working Directory

setwd("E:/DataScienceEdwisor/Rscripts")

Lets Check Our Present working directory

getwd()

Lets Load Required Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'DataCombine','sampling','pROC','ROCR')

Lets Load the Packages

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

rm(x)

Lets Read The Datasets Train & Test

train_actual = read.csv("E:/DataScienceEdwisor/Rscripts/Train_data.csv", header = T, na.strings = c(" "," ", "NA"))

test_actual = read.csv("E:/DataScienceEdwisor/Rscripts/Test_data.csv",header = T, na.strings = c(" ", " ","NA"))

Lets Create another instances copy of our Train&Test datasets on which we are going to work

train = train_actual

test = test_actual

```
# Lets Combine the train&test datasets to check the total observations of the combined dataset

data = rbind(train, test)

#**********************EXPLORATORY DATA ANALYSIS*******************#

# Lets View the Combined 'data' dataset and its total observations

View(data) # Consists 5000 observations & 21 Features || Here Last Column var is CHURN-> False. & True.

# Lets Check the Dimension of the dataset

dim(data)


# Lets View the data

View(train) # Consists 3333 observations & 21 Features || Here Last Column var is CHURN-> False. & True.

# Dimensions of the train data

dim(train)

# Lets Check the Structure of the train data

str(train)

# Lets CHeck the summary of thetrain data

summary(train) # Gives us back a brief summary stats of all numeric cols

# Lets Check the column names of train data

colnames(train) # As we can see that in excel there was space in between col names but in r synatx has provided dot(.) in between

# Lets check the class of train dataset

class(train)

# Now lets Check the Unique Values of each count

apply(train, 2,function(x) length(table(x)))

# lets drop phone number as we see it is showing 3333 entries and their are no unique values in it

# Lets drop in both train & test datasets

train$phone.number = NULL

test$phone.number = NULL

# Lets first change factor column variables to category in train dataset
```

```
train$state = as.factor(train$state)

train$area.code = as.factor(train$area.code)

train$international.plan = as.factor(train$international.plan)

train$voice.mail.plan = as.factor(train$voice.mail.plan)

train$Churn = as.factor(train$Churn)

# Lets also do the same for our test dataset also

test$state = as.factor(test$state)

test$area.code = as.factor(test$area.code)

test$international.plan = as.factor(test$international.plan)

test$voice.mail.plan = as.factor(test$voice.mail.plan)

test$Churn = as.factor(test$Churn)

# Lets the rate of churn count & in precent

table(train$Churn)

# False.   True.

# 2850     483

prop.table(table(train$Churn))

# False.     True.

# 85.50  14.49

#*******************MISSING VALUE ANALYSIS **********************#

# Lets Check missing values in our Data dataset so that we dont need to check seperately in both train&test

# here 2 indicates that is is column & 1 means row

apply(data, 2, function(x) {sum(is.na(x))}) # As we can see that there are no missing values in our dataset

#**************ANALYZING DATA THROUGH VISUALIZATION **************#

#********************** BAR PLOT ANALYSIS ******************************#

# Lets Plot Bar plots using ggplot2 library

# Count of Churn in False and True

ggplot(data=train, aes(Churn, fill=Churn)) + geom_bar(stat='count',fill='DarkSlateBlue') +

  labs(x='Churn or Not Churn', y='Count of Churn') + ggtitle("Customer Churn")

# Lets see State wise churn of customers
```

```
ggplot(train, aes(state, fill = Churn)) + geom_bar(position = "fill") +
  labs(title = "State Wise Churn of customers")
# Lets see Area code wise churn of customers
ggplot(train, aes(area.code, fill = Churn)) + geom_bar(position = "fill") +
  labs(title = "Area Code Wise Churn of customers")


# Lets see Voice Mail Plan wise churn of customers
ggplot(train, aes(voice.mail.plan, fill = Churn)) + geom_bar(position = "fill") +
  labs(title = "Voice Mail Plan Wise Churn of customers")
# Lets see Voice Mail Plan wise churn of customers
ggplot(train, aes(international.plan, fill = Churn)) + geom_bar(position = "fill") +
  labs(title = "International Plan Wise Churn of customers")
# Lets see Voice Mail Plan wise churn of customers
ggplot(train, aes(number.customer.service.calls, fill = Churn)) + geom_bar(position = "fill")
+
  labs(title = "Customer Service Calls Wise Churn of customers")
#********************** DENSITY PLOT ANALYSIS ************************#
# Lets see the density plot for account length & number.voice.mail messages For Checking
Churn Predicatability from these features
accntlngth <- ggplot(train, aes(account.length, fill = Churn)) + geom_density(alpha = 0.7) +
  theme(legend.position = "null")
nvmailmsgs <- ggplot(train, aes(number.vmail.messages, fill = Churn)) +
geom_density(alpha = 0.7) +
  theme(legend.position = "null")
gridExtra:: grid.arrange(accntlngth, nvmailmsgs, ncol = 2, nrow = 1)
# Lets see the density plot for internationalPlan & Voice Mail Plan For Checking Churn
Predicatability from these features
intlplan  <- ggplot(train, aes(international.plan, fill = Churn)) + geom_density(alpha = 0.7) +
  theme(legend.position = "null")
vmailplan <- ggplot(train, aes(voice.mail.plan, fill = Churn)) + geom_density(alpha = 0.7) +
  theme(legend.position = "null")
gridExtra:: grid.arrange(intlplan, vmailplan, ncol = 2, nrow = 1)
```

```
# Lets see the density plot for arecode & customerservicecalls For Checking Churn Predicatability from these features

areacode     <- ggplot(train, aes(area.code, fill = Churn)) + geom_density(alpha = 0.7) +

  theme(legend.position = "null")

custmrsrvcls <- ggplot(train, aes(number.customer.service.calls, fill = Churn)) + geom_density(alpha = 0.7) +

  theme(legend.position = "null")

gridExtra:: grid.arrange(areacode, custmrsrvcls, ncol = 2, nrow = 1)

#************************ OUTLIER ANALYSIS ***********************#

# Before we do outlier analysis lets save all the continous variables seperately

# As boxplot analysis is only applicable on continous variable

numeric_index = sapply(train, is.numeric) # selects only numeric

numeric_data = train[,numeric_index]

cnames = colnames(numeric_data)

# Loop For Detecting Outliers

#for (i in 1:length(cnames)) {

#assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "Churn"), data = subset(train))+

#       stat_boxplot(geom = "errorbar", width = 0.5) +

#       geom_boxplot(outlier.colour="red", fill = "skyblue" ,outlier.shape=18,

#                    outlier.size=1, notch=FALSE) +

#       labs(y=cnames[i],x="Churn")+

#       ggtitle(paste("Box plot of Churn for",cnames[i])))

#}

## Now Lets Plot, plots togeather

#gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)

#gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)

#gridExtra::grid.arrange(gn7,gn8,gn9,ncol=3)

#gridExtra::grid.arrange(gn10,gn11,gn12,ncol=3)

#gridExtra::grid.arrange(gn13,gn14,gn15,ncol=3)

## Lets Remove Outliers & then Replace all outliers with NA and impute with KNN Method
```

```
# Loop to remove outliers from the whole train data
#for(i in cnames){
#   print(i)
#   oa = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
#   print(length(oa))
#   train[,i][train[,i] %in% oa] = NA
#}
# Lets Check the count of missing values in our data now
#sum(is.na(train)) # 682 missing values after dropping outliers
# Now lets impute values with KNN method
#train = knnImputation(train, k = 3)
## Now Lets Plot again to see, Again as we have dropped outliers
#gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)
#gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)
#gridExtra::grid.arrange(gn7,gn8,gn9,ncol=3)
#gridExtra::grid.arrange(gn10,gn11,gn12,ncol=3)
#gridExtra::grid.arrange(gn13,gn14,gn15,ncol=3)
# Againg after plotting the grids here we can observe that most of the outliers have been removed except from
# number.vmail.messges may be not considering for removing as outliers.
# So as there is already target class imbalance so i am going to skip Outlier Analysis for further model development
#**********************FEATURE SELECTION ****************************#
# Lets do correlation analysis using corrgram library
corrgram(train[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt,
         main = "Checking Correlation Between Features")
# Dark blue color indicated that these variables are highly correlated with each
# heatmap plot for numerical features using corrplot library for checking collinearity
library(corrplot)
corrplot(cor(train[sapply(train, is.numeric)]))
```

```
# From Correlation plot here we can clearly observe that total-day,evening,night&international-charge are

#highly correlated with total-day,evening,night&international-minutes, so here i am going to drop charges variables

# Lets save categorical columns seperately

cat_names = c('state','area.code','international.plan','voice.mail.plan')

# Lets Do Chi2-Square Test of Independence by using for loop

for (i in cat_names)

{

  print(i)

  print(chisq.test(table(train$Churn,train[,i])))

}

# As we can see that "area.code" value i grater than p =0.05, i.e; p-value = 0.9151

#************************** DIMENSION REDUCTION ******************#

# Lets drop the fetaures highly correlated and the features which are not contributing much information

# From both train & test datasets

train = subset(train,select= -c(state,area.code,total.day.charge,total.eve.charge,

                                total.night.charge,total.intl.charge))

test = subset(test,select= -c(state,area.code,total.day.charge,total.eve.charge,

                              total.night.charge,total.intl.charge))

# Lets assign levels to categorical columns for bot train&test datasets

# After assigning the levels here 'No'-> 0 & 'Yes'-> 1

# Same as above 'False.' -> 0 & 'True.'-> 1

cat_names = c('international.plan','voice.mail.plan','Churn')

for (i in cat_names){

  levels(train[,i]) <- c(0,1)

  levels(test[,i]) <- c(0,1)

}


# Lets Update our continous variables as we have dropped some of the columns
```

```
numeric_index = sapply(train, is.numeric) # selects only numeric
numeric_data = train[,numeric_index]
cnames = colnames(numeric_data)
# Lets Do Normality Check First Using Histogram plot for all the continous variablecolumns
hist(train$account.length)
hist(train$number.vmail.messages)
hist(train$total.day.minutes)
hist(train$total.day.calls)
hist(train$total.eve.minutes)
hist(train$total.eve.calls)
hist(train$total.night.minutes)
hist(train$total.night.calls)
hist(train$total.intl.minutes)
hist(train$total.intl.calls)
hist(train$number.customer.service.calls)
# As we can observe that most of the data distibution here is uniformly distributed is it is better to go for
# Standardization/Z-Score Method for Feature Scaling of features/Predictors
#********************** FEATURE SCALING **********************#

#******************STANDARDIZTION/ Z-SCORE METHOD*******************#
# Lets aplly for both train & test datasets at a time itself
for (i in cnames) {
  print(i)
  train[,i] = (train[,i] - mean(train[,i]))/sd(train[,i])
  test[,i] = (test[,i] - mean(test[,i]))/sd(test[,i])
}

# So here after applying standardization we can observe that it has transformed the data to have 0->mean & the unit variance
```

```
# That is it will take the data point range from negative to positive as we can see here. further explained in report.

# Lets Clean-up the environment before we proceed further for model development

rmExcept(c("train_actual","test_actual","train","test"))

#********SAMPLING OF DATA USING STRATIFIED SAMPLING METHOD*******#

#Divide data into train and test using stratified sampling method

set.seed(1234)

train.index = createDataPartition(train$Churn, p = .70, list = FALSE)

train = train[ train.index,]

test  = train[-train.index,]

# Lets Check the target value churn count

table(train$Churn)

# As we can observe that there is target class imbalance problem here

# 0    1

# 1995  339

#****************************ROSE*****************************************#

# ROSE(Random Over Sampling Examples) package which helps us to generate synthetic artificial data

train_res = ROSE(Churn ~ ., data = train, p = 0.5, seed = 1)$data

# Lets CHeck the Churn Count of values in our train dataset

table(train_res$Churn)

# 0    1

# 1218 1116



#************************* MODEL DEVELOPMENT ************************#

# Lets Define a Function for Evaluating Error Metrics so that we dont need to eavluate for each model seperately

ErrorMetrics <- function(CM){

  TN=CM$table[1,1]
```

```
  FP=CM$table[1,2]
  FN=CM$table[2,1]
  TP=CM$table[2,2]
  # Accuracy
  print(paste0('Accuracy:- ' ,((TP+TN)*100/(TN+FN+TP+FP))))
  # False Negative Rate
  print(paste0('False Negative Rate:-  ' ,((FN*100)/(FN+TP))))
  # False Positive Rate
  print(paste0('False Positive Rate:-  ' ,((FP*100)/(FP+TN))))
  # Sensitivity-TruePositiveRate-Recall
  print(paste0('Sensitivity//TPR//Recall:-  ' ,((TP*100)/(TP+FN))))
  # Specificity-TrueNegativeRate
  print(paste0('Specificity//TNR:-  ' ,((TN*100)/(TN+FP))))
  # Precision
  print(paste0('Precision:-  ' ,((TP*100)/(TP+FP))))
}
########################DECISION TREE MODEL#######################
# Lets Develop Decision Tree Model on training data
DT_Model = C5.0(Churn ~., data = train_res, trials = 100, rules = TRUE)
# Lets see the summary of the model
summary(DT_Model)
# Lets write the results back into our hard-disk for further analysis
#write(capture.output(summary(DT_Model)), "DTRules.txt")

# Lets Predit for new test cases
DT_Predictions = predict(DT_Model, test[,-14], type = 'class')
DT_ConfMatrix = table(actualCases = test$Churn, predictedCases = DT_Predictions)
# Lets Evaluate Decision Tree model now by building Confusion Matrix
DT_CM = confusionMatrix(DT_ConfMatrix)
print(DT_CM)
```

```
# Lets Evaluate Decision Tree Error Metrics by passing it to the ErrorMetrics Function
ErrorMetrics(DT_CM)
# ERROR METRICS OF DECISION TREE MODEL
# [1] "Accuracy:------------------> 88.3357041251778"
# [1] "False Negative Rate:-------> 15.4545454545455"
# [1] "False Positive Rate:-------> 10.9612141652614"
# [1] "Sensitivity//TPR//Recall:->  84.5454545454545"
# [1] "Specificity//TNR:--------->  89.0387858347386"
# [1] "Precision:---------------->  58.8607594936709"
# Lets Plot AUC-ROC Curve for Decision Tree Model
roc.curve(test$Churn, DT_Predictions) # Area under the curve (AUC): 86.8
##################### RANDOM FOREST MODEL ##########################
RF_Model = randomForest(Churn ~ ., train_res, importance = TRUE, ntree = 500)
# Lets Extract Rules From RandomForest model Now
#transform rf object to an inTrees' format
library(inTrees)
treeList = RF2List(RF_Model)
# Lets Extract the rules now
extract = extractRules(treeList, train_res[,-14])
# 4498 rules (length<=6) were extracted from the first 100 trees.
# Lets Visualize some rules
extract[1:2,]

# Lets make rules more readable form
readableRules = presentRules(extract, colnames(train_res))
readableRules[1:2,]
# Lets Get few rule metrics
ruleMetrics = getRuleMetric(extract, train_res[,-14], train_res$Churn)
# Lets Evaluate few rule metrics
ruleMetrics[1:2,]
```

```
# Now lets predict new test cases

RF_Predictions = predict(RF_Model, test[,-14])

RF_ConfMatrix = table(actualCases = test$Churn, predictedCases = RF_Predictions)

# Lets build confusion matrix for random forest model

RF_CM = confusionMatrix(RF_ConfMatrix)

print(RF_CM)

# Lets Evaluate Random Forest Model Error Metrics by passing it to the ErrorMetrics
Function

ErrorMetrics(RF_CM)

# ERROR METRICS OF RANDOM FOREST MODEL

# [1] "Accuracy:----------> 89.7581792318634"

# [1] "False Negative Rate:-------> 11.8181818181818"

# [1] "False Positive Rate:-------> 9.94940978077572"

# [1] "Sensitivity//TPR//Recall:->  88.1818181818182"

# [1] "Specificity//TNR:--------->  90.0505902192243"

# [1] "Precision:---------->  62.1794871794872"

# Lets Plot AUC-ROC Curve for Random Forest Model

roc.curve(test$Churn, RF_Predictions) # Area under the curve (AUC): 89.1

######################LOGISTIC REGRESSION MODEL ########################

# Lets Develop Logistic Regression Model

LR_Model = glm(Churn ~ ., train_res, family = 'binomial')


# Lets Check the summary

summary(LR_Model)

# Output:- international.plan1, voice.mail.plan1, total.day.minutes, total.eve.minutes,
total.night.minutes

# total.intl.minutes, number.customer.service.calls are very much significant

# Lets Predict new test cases

LR_Predictions = predict(LR_Model, newdata = test, type = 'response')

# Lets Convert them to probabilities

LR_Predictions = ifelse(LR_Predictions > 0.5, 1, 0)
```

```
# Lets evaluate the performance of the model
LR_ConfMatrix = table(actualCases = test$Churn, predictedCases = LR_Predictions)
# Lets build confusion matrix for Logistic Regression model
LR_CM = confusionMatrix(LR_ConfMatrix)
print(LR_CM)
# Lets Evaluate Logistic Regression Model Error Metrics by passing it to the ErrorMetrics Function
ErrorMetrics(LR_CM)
# ERROR METRICS OF LOGISTIC REGRESSION MODEL
# [1] "Accuracy:------------------> 80.2275960170697"
# [1] "False Negative Rate:-------> 27.2727272727273"
# [1] "False Positive Rate:-------> 18.3811129848229"
# [1] "Sensitivity//TPR//Recall:->  72.7272727272727"
# [1] "Specificity//TNR:--------->  81.6188870151771"
# [1] "Precision:----------------->  42.3280423280423"
# Lets Plot AUC-ROC Curve for Logistic Regression Model
roc.curve(test$Churn, LR_Predictions) # Area under the curve (AUC): 77.2

################ K- NEAREST NEIGHBOR MODEL ##############
# Lets Develop KNN  MODEL
library(class)
# Lets Predict the test data
KNN_Predictions = knn(train_res[, 1:13], test[, 1:13], train_res$Churn, k = 5)
# Lets Build Confusion Matrix for KNN MODEL
KNN_ConfMatrix = table(KNN_Predictions, test$Churn)
KNN_CM = confusionMatrix(KNN_ConfMatrix)
print(KNN_CM)
# Lets Evaluate K Nearest Neighbor Error Metrics by passing it to the ErrorMetrics Function
ErrorMetrics(KNN_CM)
# ERROR METRICS OF K NEAREST NEIGHBOR MODEL
```

```
# [1] "Accuracy:------------------> 86.4864864864865"
# [1] "False Negative Rate:-------> 45.2830188679245"
# [1] "False Positive Rate:-------> 4.22794117647059"
# [1] "Sensitivity//TPR//Recall:->  54.7169811320755"
# [1] "Specificity//TNR:--------->  95.7720588235294"
# [1] "Precision:----------------->  79.0909090909091"
# Lets Plot AUC-ROC Curve for KNN Model
roc.curve(test$Churn, KNN_Predictions) # Area under the curve (AUC): 83.5
###############NAIVE BAYES MODEL ############
# Lets Develop Naive Bayes MOdel
NB_Model = naiveBayes(Churn ~ ., data = train_res)
# Lets Predict new test cases
NB_Predictions = predict(NB_Model, test[,1:13], type = 'class')
# Lets Build Confusion Matrix for Naive Bayes Model
NB_ConfMatrix = table(NB_Predictions, test$Churn)
NB_CM = confusionMatrix(NB_ConfMatrix)
print(NB_CM)
# Lets Evaluate Naive Bayes Model Error Metrics by passing it to the ErrorMetrics Function
ErrorMetrics(NB_CM)
# ERROR METRICS OF NAIVE BAYES MODEL
# [1] "Accuracy:------------------> 86.9132290184922"
# [1] "False Negative Rate:-------> 43.6619718309859"
# [1] "False Positive Rate:-------> 5.3475935828877"
# [1] "Sensitivity//TPR//Recall:->  56.3380281690141"
# [1] "Specificity//TNR:--------->  94.6524064171123"
# [1] "Precision:----------------->  72.7272727272727"
# Lets Plot AUC-ROC Curve for KNN Model
roc.curve(test$Churn, NB_Predictions) # Area under the curve (AUC): 81.1
# Variable importance
library(caret)
```

```
varImp(RF_Model)
#Lets Plot the Variable Importance
varImpPlot(RF_Model, type = 2)
# Lets save the results back to hard disk
## Setting up the Working Directory
setwd("E:/DataScienceEdwisor/PROJECT-1/R")
write.csv(train_res, "train_data.csv", row.names = F)
write.csv(test, "test_data.csv", row.names = F)
```

References

https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/

https://imbalancedlearn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

https://www.rdocumentation.org/packages/ROSE/versions/0.0-3/topics/ROSE