Project Report on Customer Churn

Kiran C

25/02/2019

Contents

1. Introduction

1.1 Problem Statement

The objective of the project is to develop a suitable model that predicts the customer behaviour. The model to be developed should give the churn score that correctly tells whether a customer continues to be with the telecom operator (not churn) or leaves (churn).

1.2 Data

Given data sets comprise of both train and test with reasonably good sampling. Train data contains 3333 observations with 21 variables and test data contains 1667 observations with 21 variables. Out of the 21 variables 20 are predictor variables and one is dependent variable. The below tables – 1, 2, 3 show a glimpse of the train data. Tables – 4 & 5 lists the categorical and continuous variables respectively. The target variable is “Churn”, which is a binary one where “False” represents customer doesn’t churn and “True” represents customer churns.

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages
KS	128	415	382-4657	no	yes	25
OH	107	415	371-7191	no	yes	26
NJ	137	415	358-1921	no	no	0
OH	84	408	375-9999	yes	no	0
OK	75	415	330-6626	yes	no	0
AL	118	510	391-8027	yes	no	0
MA	121	510	355-9993	no	yes	24
MO	147	415	329-9001	yes	no	0
LA	117	408	335-4719	no	no	0

Table 1: First 7 variables of Train data

total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
265.1	110	45.07	197.4	99	16.78	244.7
161.6	123	27.47	195.5	103	16.62	254.4
243.4	114	41.38	121.2	110	10.3	162.6
299.4	71	50.9	61.9	88	5.26	196.9
166.7	113	28.34	148.3	122	12.61	186.9
223.4	98	37.98	220.6	101	18.75	203.9
218.2	88	37.09	348.5	108	29.62	212.6
157	79	26.69	103.1	94	8.76	211.8
184.5	97	31.37	351.6	80	29.89	215.8

Table 2: 8-14 variables of Train data

total night calls	total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn
91	11.01	10	3	2.7	1	False.
103	11.45	13.7	3	3.7	1	False.
104	7.32	12.2	5	3.29	0	False.
89	8.86	6.6	7	1.78	2	False.
121	8.41	10.1	3	2.73	3	False.
118	9.18	6.3	6	1.7	0	False.
118	9.57	7.5	7	2.03	3	False.
96	9.53	7.1	6	1.92	0	False.
90	9.71	8.7	4	2.35	1	False.

Table 3: 15-24 variables of Train data

2. Methodology

2.1 Data Pre-processing

Data pre-processing is the stage where we make our data ready to feed to the model. It includes various steps like – exploratory data analysis, missing values analysis, feature engineering, feature selection, feature scaling and sampling.

2.1.1 Exploratory Data Analysis

As a part of the exploratory data analysis, the data types of the variables are converted into suitable data types, so that it eases further pre-processing techniques. In Table – 4 & Table – 5, categorical and continuous variables are enlisted. In total there are 6 categorical variables and 15 continuous variables. All of these are converted into their data types from the original types. Some of them were already in their correct type. All categorical variables are assigned to Cat_Var and that of continuous variables to Con_Var.
The aim of the project is to predict the churn score of customers, which depends on various factors given under predictor variables list. The variable "Churn" becomes the target (dependent) variable that holds two categories of values – "False" and "True". The remaining variables and their types are self explanatory.
Predictor variables: "state", "area code", "phone number", "international plan", "voice mail plan", "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"
Target variable: "Churn"

S.No.	Categorical Variable
1	state
2	area code
3	phone number
4	international plan
5	voice mail plan
6	Churn

Table 4: List of Categorical variables

S.No.	Continuous Variable
1	account length
2	number vmail messages
3	total day minutes
4	total day calls
5	total day charge
6	total eve minutes
7	total eve calls
8	total eve charge
9	total night minutes
10	total night calls
11	total night charge
12	total intl minutes
13	total intl calls
14	total intl charge
15	number customer service calls

Table 5: List of Continuous variables

2.1.1.1 Univariate Analysis

To check the effect of each predictor variable on the dependent variable, appropriate plots are drawn. All visualisations are included in Appendix. As we are given train and test sets separately, at this stage, it is important to look at the distribution of dependent variable in both the datasets. Fig – 1 below shows the density plot of "Churn" across TrainData and TestData. From the plot, it can be understood that "Churn" is equally distributed w.r.t the number of observations of respective data sets.

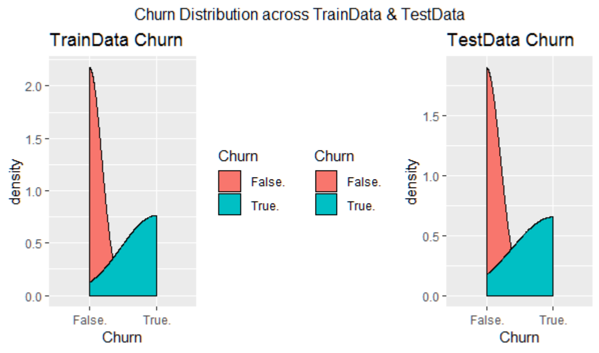


Fig – 1: Churn distribution across TrainData & TestData

2.1.2 Missing Value Analysis

Next step in the data pre-processing stage is to check if there are any missing values in any of the continuous variables, as these values would influence the inferences we draw from the variables. In our present dataset, after investigation, it is found that there are no missing values in any of the "Con_Var".

2.1.3 Outlier Analysis

Outliers are the observations that are distant from other observations. They cause the data analysis biased and incorrect model predictions. If they are not treated well, every effort in data analysis is a mere waste. To trace out the presence of outliers in continuous variables, we plot their box plots against "Churn" and check. Box plots of Con_Var are depicted in grid form as shown in Fig – 2, Fig – 3, Fig – 4 & Fig – 5. Different ways of dealing with outliers are –

i) Removing them from the dataset

ii) Replacing them by NAs and imputing them either by their respective mean or median or KNN.

iii) Outlier capping – calculate inter-quartile range (IQR – which is the difference between 75th percentile and 25th percentile values) of a variable – replace the values, that are less than the difference between 25th percentile & 1.5 times of IQR and more than the summation of 75th percentile & 1.5 times of IQR, with 5th percentile value and 95th percentile value respectively.

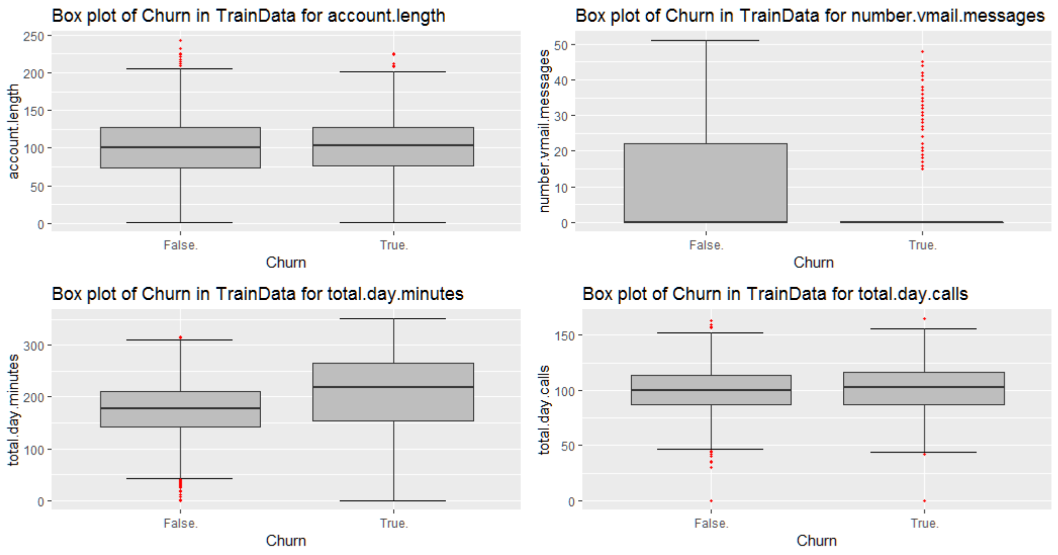
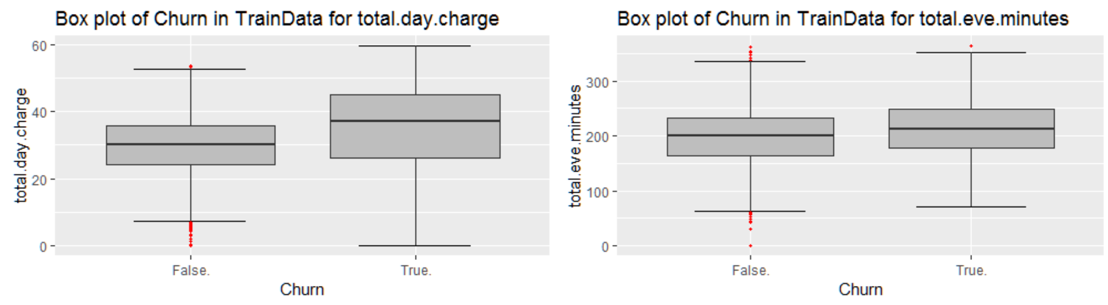


Fig – 2: Box plot grid showing outliers of "account length", "number vmail messages", "total day minutes" & "total day calls" against "Churn" from TrainData



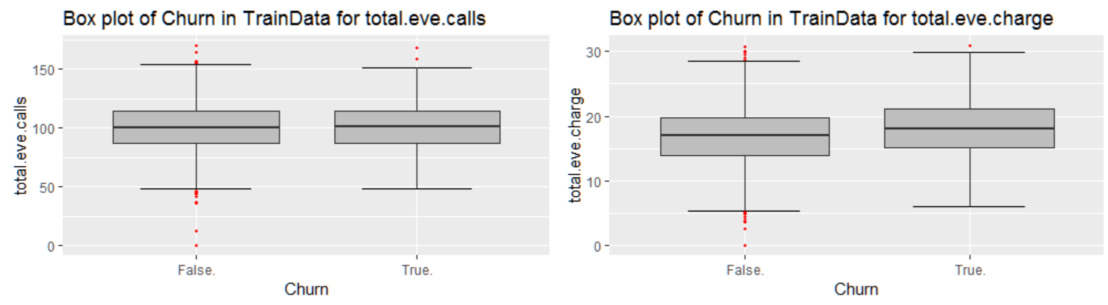
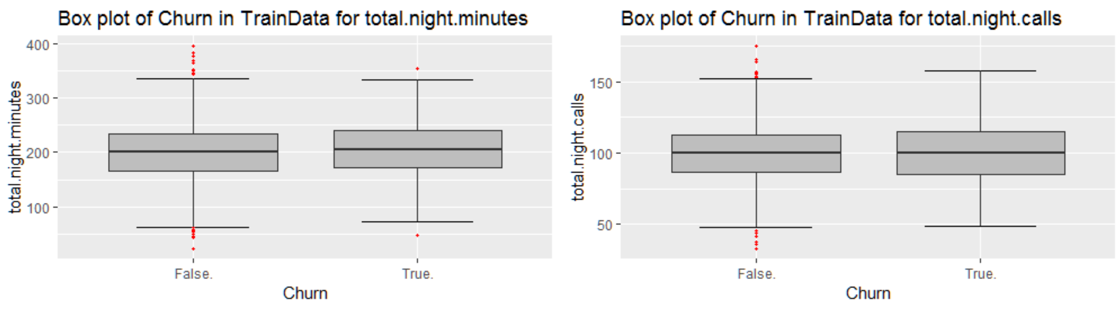


Fig – 3: Box plot grid showing outliers of “total day charge”, “total eve minutes”, “total eve calls” & “total eve charge” against “Churn” from TrainData



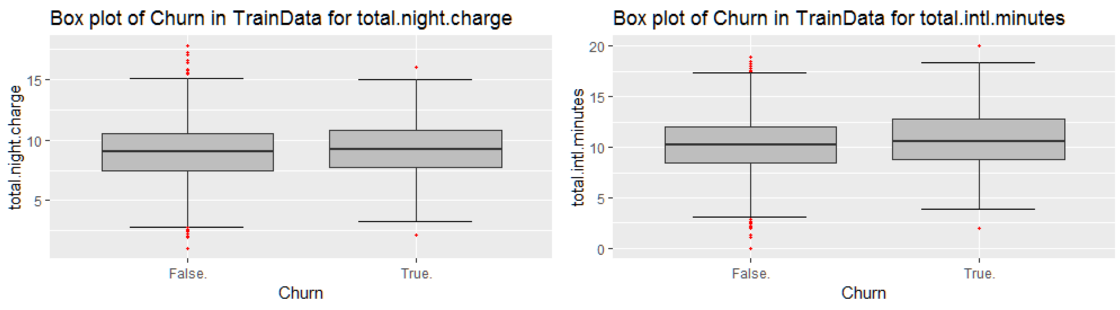


Fig – 4: Box plot grid showing outliers of “total.night.minutes”, “total.night.calls”, “total.night.charge” & “total.intl.minutes” against “Churn” from TrainData

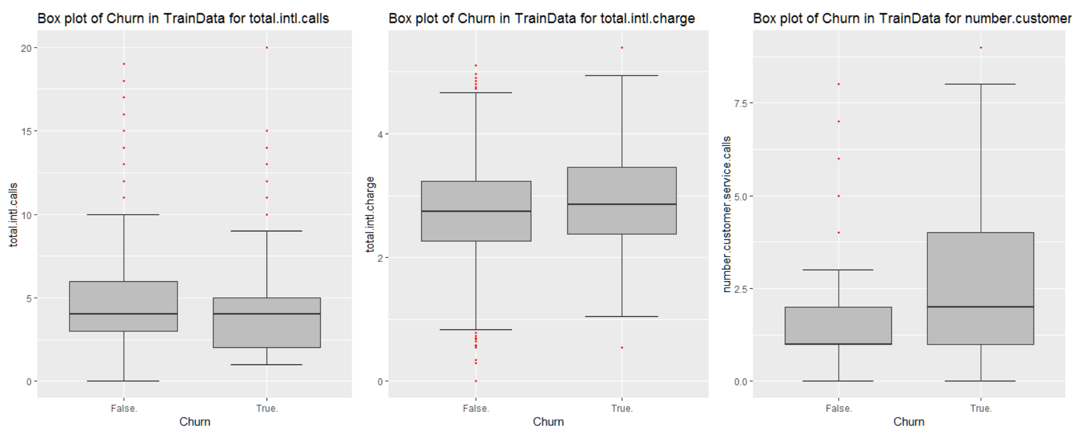
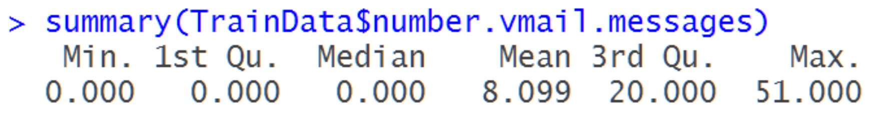


Fig – 5: Box plot grid showing outliers of "total intl calls", "total intl charge" & "number customer service calls" against "Churn" from TrainData

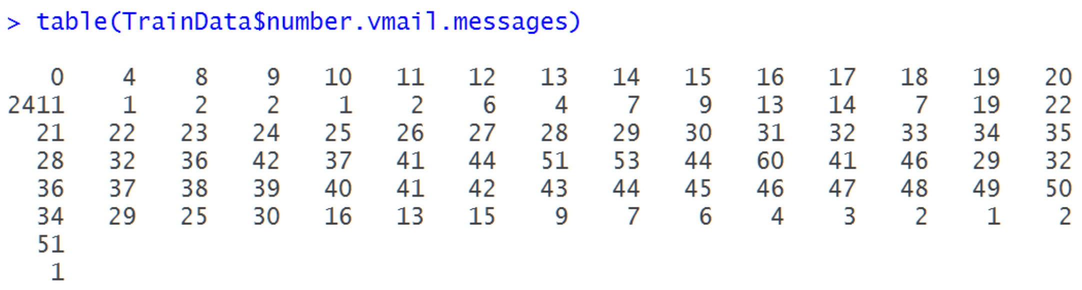
From second grid of in Fig – 2, we can clearly understand,

a) 25th percentile and median of "number vmail messages" in TrainData are coinciding. Summary and tables of values of this variable is given by



```
> summary(TrainData$number.vmail.messages)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   0.000   8.099  20.000  51.000 
```

Fig – 6: summary of "number vmail messages" of TrainData

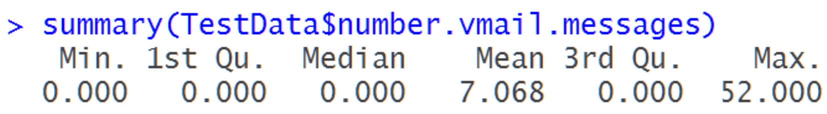


```
> table(TrainData$number.vmail.messages)

   0    4    8    9   10   11   12   13   14   15   16   17   18   19   20 
2411    1    2    2    1    2    6    4    7    9   13   14    7   19   22 
  21   22   23   24   25   26   27   28   29   30   31   32   33   34   35 
  28   32   36   42   37   41   44   51   53   44   60   41   46   29   32 
  36   37   38   39   40   41   42   43   44   45   46   47   48   49   50 
  34   29   25   30   16   13   15    9    7    6    4    3    2    1    2 
  51 
   1 
```

Fig – 7: table of values of "number vmail messages" of TrainData

This variable is heavily inflated by zeros (more than 72% of values are zeros). Any effort to treat the outliers in it, either introduced new outliers or original values are replaced by zeros.

b) In case of TestData, inter-quartile range of "number vmail messages" has completely become zero, which means, 1st quartile; median and 3rd quartile coincided and are equal to zeros. Summary and table of values of it is given by



```
> summary(TestData$number.vmail.messages)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   0.000   7.068   0.000  52.000 
```

Fig – 8: summary of "number vmail messages" of TestData

```
> table(TestData$number.vmail.messages)

   0    6   10   12   14   15   16   17   18   19   20   21   22   23   24   25   26 
1267    2    3    5    2    4    4    7   11    8   10   13   15   16   22   14   17 
  27   28   29   30   31   32   33   34   35   36   37   38   39   40   41   42   43 
  20   16   14   14   23   16   20   12    9   11   14   16   12   12    9    4    7 
  44   45   46   47   48   49   52 
   2    5    4    1    3    2    1 
```

Fig – 9: table of values of "number vmail messages" of TestData

This variable is heavily inflated by zeros (more than 76% of values are zeros). Any effort to treat the outliers in it, either introduced new outliers or original values are replaced by zeros. Therefore, except for zeros, we cannot consider the other values as outliers and process them and it's better to take them as they are. Further there are very few outliers associated with other variables as shown in Table – 6. Number of outliers in each variable, except "number vmail messages" of TestData and "number customer service.calls" of both TrainData & TestData is less than 2% of total number of observations. So they may have little effect on model accuracy. Hence Outlier Analysis is skipped.

Variable name	No.of Outliers in TrainData	No.of Outliers in TestData
account length	18	7
number vmail messages	1	400
total day minutes	25	10
total day calls	23	10
total day charge	25	10
total eve minutes	24	17
total eve calls	20	16
total eve charge	24	17
total night minutes	30	12
total night calls	22	19
total night charge	30	12
total intl minutes	46	23
total intl calls	78	105
total intl charge	49	23
number customer service calls	267	132

Table 6: No.of outliers in TrainData & TestData

2.1.4 Feature Selection

Feature selection aka Variable selection is a very important step in any machine learning model development as we decide how and which features (variables) in the given dataset would affect the predictive model that we are going to adopt. In this section, we will examine, which independent variables are considered to be part of model development.

There are several methods to do feature selection. Here we follow Correlation Analysis to check dependency among continuous variables (features) and Chi-Squared test of independence to check dependency among categorical variables.

2.1.4.1 Correlation Analysis

Correlation analysis is conducted to check if there is any dependency or collinearity between two continuous variables. If there is any collinearity between two variables, it implies that they have similar effect on target variable and it doesn't make sense to have two variables with similar behaviour in the model development. So we should discard one of them. In our present case, Fig – 10 depicts correlation plot of Con_Var of the TrainData. From this plot, it is clearly seen that – "total day minutes" & "total day charge"; "total eve minutes" & "total eve charge"; "total night minutes" & "total night charge" and "total intl minutes" & "total intl charge" are highly correlated. From these sets, we discard "total day charge", "total eve charge", "total intl charge".

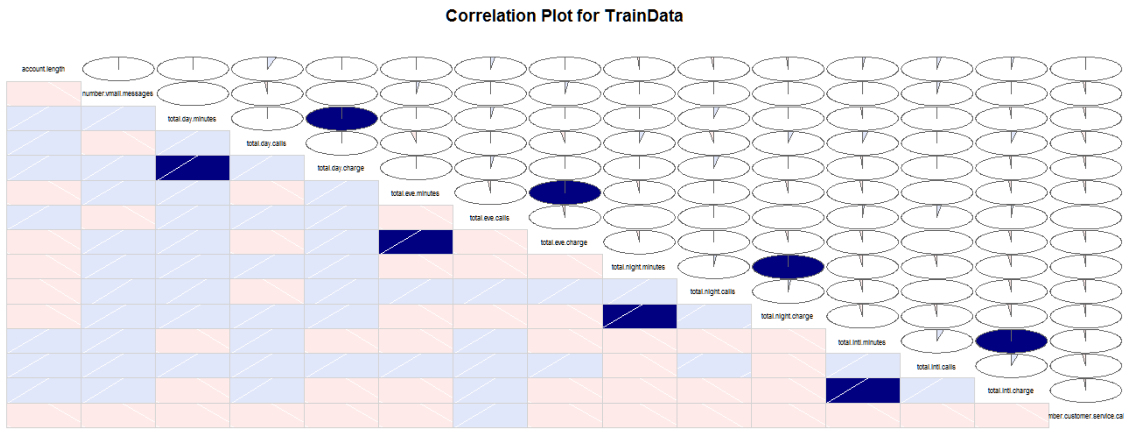


Fig – 10: Correlation plot of TrainData

2.1.4.2 Chi-Squared test of independence

It's a quantitative measure used to determine whether a relationship exists between two categorical variables. Here we run Chi-squared test of independence to check how our target variable, "Churn", is associated with the other categorical variables in the dataset. Firstly, we calculate the Chi-squared value by the formula:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

$$where\ O_i\ is\ the\ original\ data\ point\ and\ E_i\ is\ the\ Expected\ value$$

Now we calculate degrees of freedom (DF) by evaluating the equation:

DF = (number of rows – 1)(number of columns – 1)

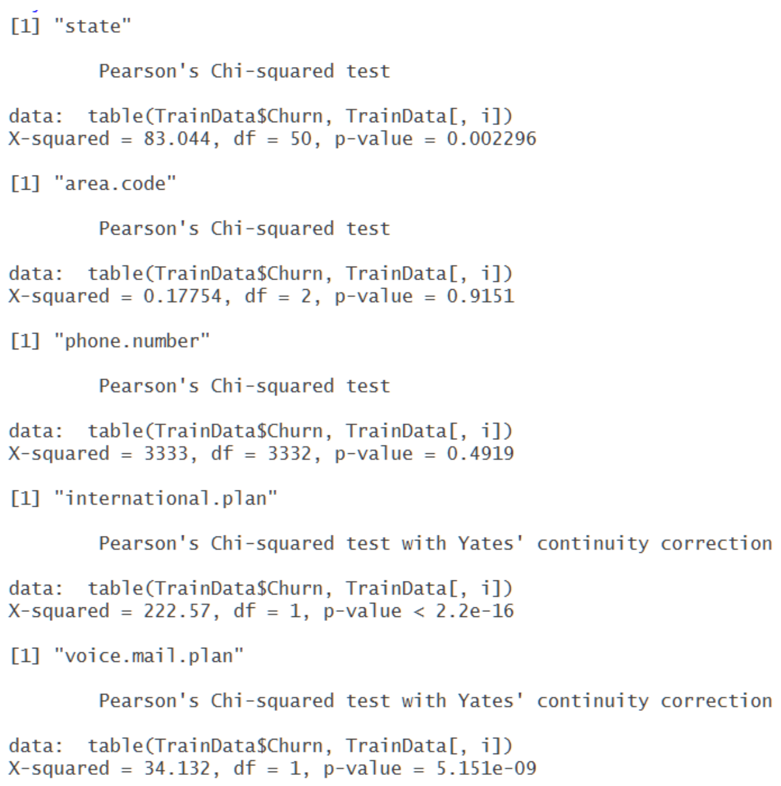
For the number of DF value, we find a critical value for a given confidence interval. We now compare the χ^2 value with this critical value and go for hypothesis testing, which states that –

Null Hypothesis: The two (predictor & target) variables are independent

Alternative Hypothesis: The two (predictor & target) variables are not independent

If χ^2 > Critical value, then we reject the null hypothesis, saying that the two variables are not independent, which means predictor is carrying information in explaining the dependent variable
If χ^2 < Critical value, then we accept the null hypothesis, saying that the two variables are independent, which means predicator is not carrying any information in explaining the dependent variable.
Programmatically, R and Python calculate the Chi-squared value, DF and based on which p-value gets generated. This p-value is the deciding factor whether to accept or reject the null hypothesis.
If p-value > 0.05, we accept the null hypothesis
If p-value < 0.05, we reject the null hypothesis

Fig – 11 shows the Chi-squared test results. It includes, χ^2 value, df and associated p-value. From the results, it is evident that, p-value of “state”, “international plan”, “voice mail plan” is less than 0.05 and that of “area code” and “phone number” is above 0.05. According to hypothesis, we retain three variables with p<0.05 and discard two variables with p>0.05.



```
[1] "state"

        Pearson's Chi-squared test

data:  table(TrainData$Churn, TrainData[, i])
X-squared = 83.044, df = 50, p-value = 0.002296

[1] "area.code"

        Pearson's Chi-squared test

data:  table(TrainData$Churn, TrainData[, i])
X-squared = 0.17754, df = 2, p-value = 0.9151

[1] "phone.number"

        Pearson's Chi-squared test

data:  table(TrainData$Churn, TrainData[, i])
X-squared = 3333, df = 3332, p-value = 0.4919

[1] "international.plan"

        Pearson's Chi-squared test with Yates' continuity correction

data:  table(TrainData$Churn, TrainData[, i])
X-squared = 222.57, df = 1, p-value < 2.2e-16

[1] "voice.mail.plan"

        Pearson's Chi-squared test with Yates' continuity correction

data:  table(TrainData$Churn, TrainData[, i])
X-squared = 34.132, df = 1, p-value = 5.151e-09
```

Fig – 11: Chi-squared test result of categorical variables w.r.t. “Churn”

2.1.5 Dimensionality Reduction

After Correlation and Chi-squared test analysis, the variables we are going to consider in our final dataset to feed to a model are –

Continuous variables – “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”

Categorical variables – “state”, “international plan”, “voice mail plan”, “Churn”

2.1.6 Feature Scaling

This is another important step in making the dataset uniform w.r.t scales and units. The model predictions will go wrong if the variables are of different scales. To get the ranges of independent – continuous variables on to a common scale, so that they can be compared on the common ground, we use feature scaling. In the present case, we use normalisation to scale the features. Normalisation limits the range of a continuous variable between 0 and 1. It reduces the unwanted variation either within or between variables. Normalised values are calculated by the following formula:

$$value_{new} = \frac{value - minValue}{maxValue - minValue}$$

Histograms of continuous variables before and after normalisation are presented in Appendix

2.2 Modelling

Our target variable is a categorical variable, so our predictive model should be a classification model. The following 5 classification methodologies are chosen for investigation so as to decide which is best suitable for our dataset and problem statement –

-> Decision Tree Classification
-> Random Forest Classification
-> KNN Classification
-> Naive Bayes Classification
-> Logistic Regression

We use TrainData to train each of the above models and test and evaluate models’ performance using TestData.

2.2.1 Model Development

2.2.1.1 Decision Tree Classification (DTC) Model

In R, we use “*C5.0*” function from “*C50*” library to build DTR model on training data by choosing “Churn” as the target variable. “*C5.0*” fits classification tree models or rule-based models using *Quinlan’s C5.0* algorithm. An important feature of *C5.0* is its ability to generate classifiers called *rulesets* that consist of unordered collections of (relatively) simple if-then rules. We use these set of rules to predict the test cases. Later we compare the predicted test case with the original values and see how close the values are. The syntax of *C5.0* is given by

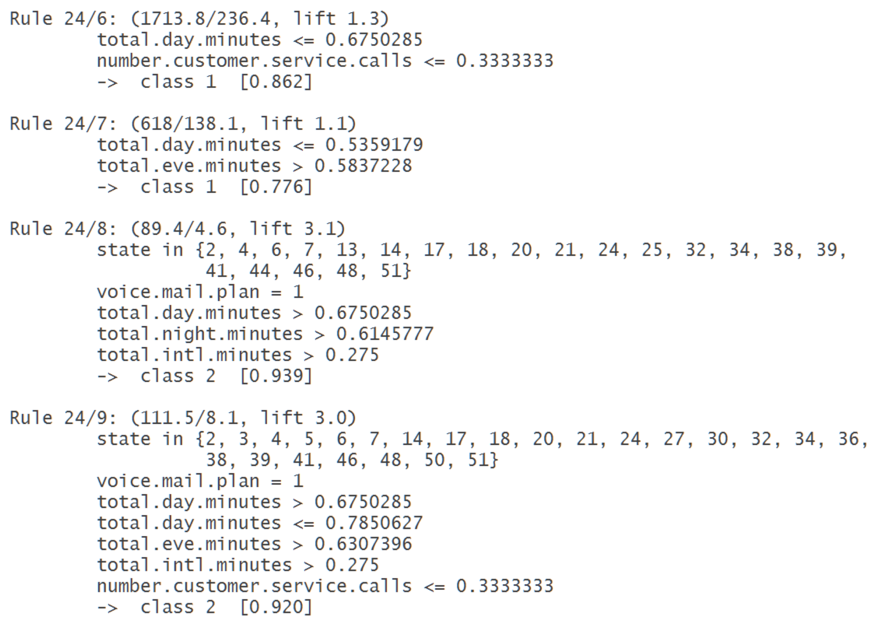
$$C5.0(x, y, trials = 1, rules = FALSE, weights = NULL, Control == C5.0Control(), costs = NULL, \ldots)$$

Where
x – a data frame or matrix of predictors
y – a factor vector with 2 or more levels
trials – an integer specifying the number of boosting iterations which ranges between 1 and 100. A value of one indicates that a single model is used.

rules – a logical that tells should the tree be decomposed into a rule-based model or not
As we are interested in rules based model, we make “rules = TRUE” and “trials” is selected by trial-and-error method by looking at the attribute usage and model accuracy that each trail gives. Here we got best results with 25 trials. Each trial creates a set of rules. Each rule consists of:

- A rule number -- this is quite arbitrary and serves only to identify the rule.
- Statistics (*n*, lift *x*) or (*n*/*m*, lift *x*) that summarize the performance of the rule. Similarly to a leaf, *n* is the number of training cases covered by the rule and *m*, if it appears, shows how many of them do not belong to the class predicted by the rule. The rule's accuracy is estimated by the Laplace ratio *(n-m+1)/(n+2)*. The lift *x* is the result of dividing the rule's estimated accuracy by the relative frequency of the predicted class in the training set.
- One or more conditions that must all be satisfied if the rule is to be applicable.
- A class predicted by the rule.
- A value between 0 and 1 that indicates the confidence with which this prediction is made.

Fig – 12 shows a few of the rules of 25th trial created by *C5.0* function. We have two classes in our target variable viz., “False” and “True”, so the rules are created for both the classes and the associated confidence with which the specific trial has correctly estimated the rule.



```
Rule 24/6: (1713.8/236.4, lift 1.3)
	total.day.minutes <= 0.6750285
	number.customer.service.calls <= 0.3333333
	->  class 1  [0.862]

Rule 24/7: (618/138.1, lift 1.1)
	total.day.minutes <= 0.5359179
	total.eve.minutes > 0.5837228
	->  class 1  [0.776]

Rule 24/8: (89.4/4.6, lift 3.1)
	state in {2, 4, 6, 7, 13, 14, 17, 18, 20, 21, 24, 25, 32, 34, 38, 39,
	          41, 44, 46, 48, 51}
	voice.mail.plan = 1
	total.day.minutes > 0.6750285
	total.night.minutes > 0.6145777
	total.intl.minutes > 0.275
	->  class 2  [0.939]

Rule 24/9: (111.5/8.1, lift 3.0)
	state in {2, 3, 4, 5, 6, 7, 14, 17, 18, 20, 21, 24, 27, 30, 32, 34, 36,
	          38, 39, 41, 46, 48, 50, 51}
	voice.mail.plan = 1
	total.day.minutes > 0.6750285
	total.day.minutes <= 0.7850627
	total.eve.minutes > 0.6307396
	total.intl.minutes > 0.275
	number.customer.service.calls <= 0.3333333
	->  class 2  [0.920]
```

Fig – 12: A few of the rules of 25th trial created by C5.0 function

When a ruleset like this is used to classify a case, it may happen that several of the rules are applicable (that is, all their conditions are satisfied). If the applicable rules predict different classes, there is an implicit conflict that could be resolved in several ways: for instance, we could believe the rule with the highest confidence, or we could attempt to aggregate the rules' predictions to reach a verdict. C5.0 adopts the latter strategy -- each applicable rule votes for its predicted class with a voting weight equal to its confidence value, the votes are totted up, and the class with the highest total vote is chosen as the final prediction. There is also a *default class* (as seen in trial 97 as “*default class: 2”*) that is used when none of the rules apply.

Table – 7 below shows the number of rules created, corresponding number of errors and percentages of erroneous values w.r.t total number of observations in TrainData, while the line labelled boost shows the result of voting all the classifiers.

```
Evaluation on training data (3333 cases):

Trial              Rules
-----        ----------------
        No      Errors

   0    19  146( 4.4%)
   1    14  345(10.4%)
   2    32  312( 9.4%)
   3    36  495(14.9%)
   4    19  407(12.2%)
   5    11  284( 8.5%)
   6    32  482(14.5%)
   7    21  324( 9.7%)
   8    10  226( 6.8%)
   9    17  488(14.6%)
  10    27  314( 9.4%)
  11    24  292( 8.8%)
  12    14  424(12.7%)
  13    29  480(14.4%)
  14    32  190( 5.7%)
  15    20  298( 8.9%)
  16    33  288( 8.6%)
  17    11  401(12.0%)
  18    29  334(10.0%)
  19    17  535(16.1%)
  20    14  306( 9.2%)
  21    28  339(10.2%)
  22    23  263( 7.9%)
  23    23  211( 6.3%)
  24    11  295( 8.9%)
boost                62( 1.9%)   <<
```

Table – 7: 100 Trials with respective number of rules; number of error and corresponding percentage

Fig – 13 below shows confusion matrix and Attribute usage of TrainData in Decision Tree classification modelling. An attribute is used to classify a case if it is referenced by a condition of at least one rule that applies to that case; the order in which attributes appear in a ruleset is not relevant. The percentage associated with each independent variable here indicates the percent of each attribute used in predicting the target variable.

```
  (a)   (b)    <-classified as
 ----  ----
 2849     1    (a): class 1
   61   422    (b): class 2


Attribute usage:

100.00% international.plan
100.00% total.day.minutes
100.00% total.eve.minutes
100.00% total.intl.minutes
100.00% total.intl.calls
100.00% number.customer.service.calls
 99.88% account.length
 99.64% total.night.minutes
 99.04% state
 93.61% total.day.calls
 91.96% number.vmail.messages
 89.62% total.eve.calls
 89.29% total.night.calls
 85.30% voice.mail.plan
```

Fig – 13: Confusion matrix and Attribute usage of TrainData in modelling

In Python, we use *tree.DecisionTreeClassifier* function from *sklearn* library. The syntax is given below:

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
              max_depth=None, max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, presort=False, random_state=None,
              splitter='best')
```

Fig – 14: DecisionTreeClassifier function used

We have used all default argument values except for *class_weight* and *criterion* that has given us best results.

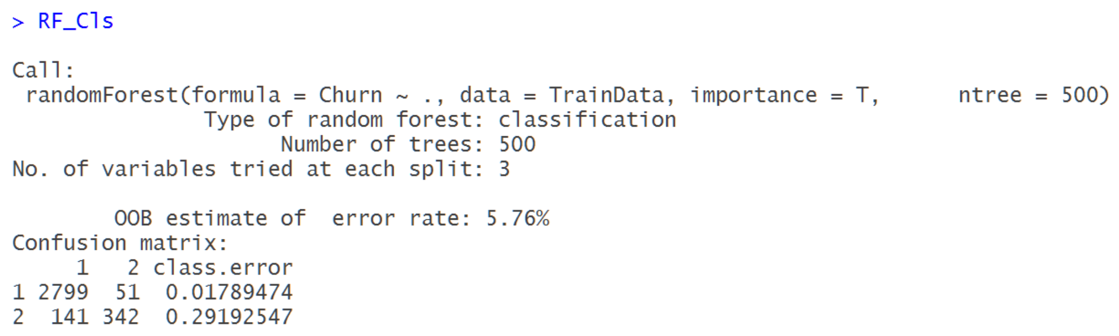
class_weight = 'balanced' mode uses the values of dependent variable to automatically adjust weights inversely proportional to class frequencies in the input data.

criterion = 'entropy', as we are interested in C5.0 model

2.2.1.2 Random Forest Classification (RFC) Model

Random Forest is an ensemble of a number of decision trees and features are selected randomly. Error associated with each decision tree is fed to another decision tree in the next node or level. This continues until the leaf node is reached. This concept is called "bagging" or "bootstrap aggregating". In Breiman's implementation of the random forest algorithm, each tree is trained on about 2/3 of the total training data. As the forest is built, each tree can thus be tested on the samples not used in building that tree. This is the Out-of-Bag (OOB) error estimate - an internal error estimate of a random forest as it is being constructed.

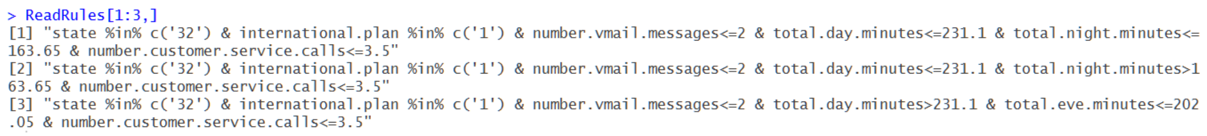
In R, we use *randomForest* function from *randomForest* library to build RFC model on training data by choosing "Churn" as the target variable. With 500 number of tress and other parameters with their default values, has given the best fit of the model. Summary of the "*randomForest*" function generates an object shown in Fig – 15 below. It gives details like No. of variables tried at each split, OOB estimate of error rate and confusion matrix.



```
> RF_Cls

Call:
 randomForest(formula = Churn ~ ., data = TrainData, importance = T,      ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 5.76%
Confusion matrix:
     1   2 class.error
1 2799  51  0.01789474
2  141 342  0.29192547
```

Fig – 15: object created by randomForest function

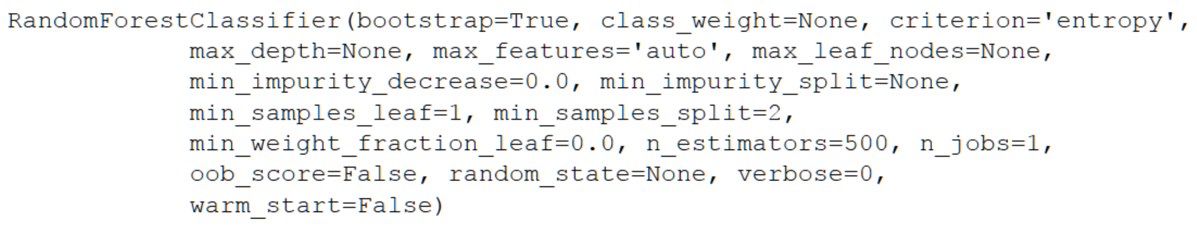
We can extract rules generated by "*randomForest*" function, and through which we can obtain rule metrics. We feed the above object to predict the test cases for RFC. Then we calculate error metrics to see how good our RFC is. Fig – 16 below shows first 3 rules created.



```
> ReadRules[1:3,]
[1] "state %in% c('32') & international.plan %in% c('1') & number.vmail.messages<=2 & total.day.minutes<=231.1 & total.night.minutes<=
163.65 & number.customer.service.calls<=3.5"
[2] "state %in% c('32') & international.plan %in% c('1') & number.vmail.messages<=2 & total.day.minutes<=231.1 & total.night.minutes>1
63.65 & number.customer.service.calls<=3.5"
[3] "state %in% c('32') & international.plan %in% c('1') & number.vmail.messages<=2 & total.day.minutes>231.1 & total.eve.minutes<=202
.05 & number.customer.service.calls<=3.5"
```

Fig – 16: First 3 rules created in random forest classification

In Python, we use *RandomForestClassifier* function imported from *sklearn.ensemble.* We have specified *criterion* = '*entropy*' and by trial-and-error method *n_estimators = 500* has given the best results. The remaining parameters are left default. Fig – 17 below shows the function and its default argument values.



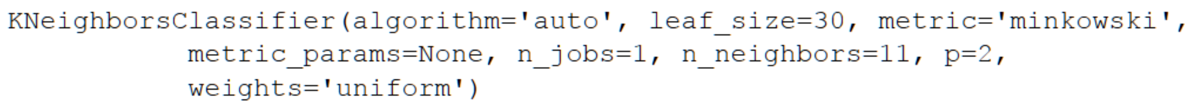
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Fig – 17: RandomForestClassifier

2.2.1.3 KNN Classification (KNN-C) Model

In R, we use "*knn*" function from "*class*" library to build a KNN Classification model. We give TrainData and number of neighbours (k) to be considered as arguments. By trial and error, value of 3 for k has given best predictions. *knn* directly gives us the predicted values too. Based on this we obtain confusion matrix and calculate error metrics.
In Python, we use "*KNeighborsClassifier*" function imported from *"sklearn.neighbors*". We provide *n_neighbors* as argument to this function and remaining are taken as their default and we fit this to the TrainData. With the help outcome of this function, we predict the test values

and compare them with the original test data values to evaluate the model. Fig – 18 below shows the *KNeighborsClassifier* function used.



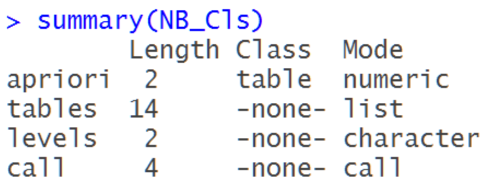
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
```

Fig – 18: *KNeighborsClassifier*

2.2.1.4 Naive Bayes Classification Model

This classification model is based on Bayes' theorem of conditional probability. Accordingly, the predictors are assumed to be independent of each other in predicting the target class. In other words, Naive Bayes' classifier assumes that the prediction of a class by an independent variable is not related to another variable.

In R, we use *naiveBayes* function from *e1074* library to fit the model. This function computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using Bayes' rule. An object of class "*naiveBayes*" includes the following components:

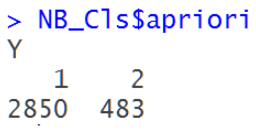


```
> summary(NB_Cls)
        Length Class  Mode
apriori  2     table  numeric
tables  14     -none- list
levels   2     -none- character
call     4     -none- call
```

Fig – 19: Summary of naiveBayes

Where

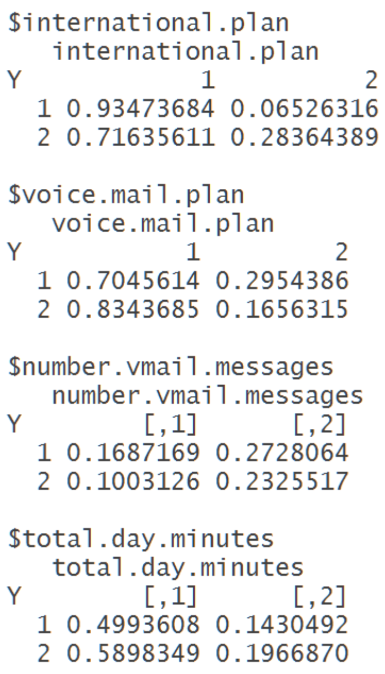
apriori: Class distribution for the dependent variable. And it returns:



```
> NB_Cls$apriori
Y
   1    2
2850  483
```

Fig – 20: apriori of naiveBayes class

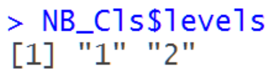
tables: A list of tables, one for each predictor variable. For each categorical variable a table gives, for each attribute level, the conditional probabilities given the target class. For each numeric variable, a table gives, for each target class, mean and standard deviation of the (sub-) variable. Following are the examples of tables created for two categorical and two numeric variables.



```
$international.plan
   international.plan
Y            1          2
  1 0.93473684 0.06526316
  2 0.71635611 0.28364389

$voice.mail.plan
   voice.mail.plan
Y           1         2
  1 0.7045614 0.2954386
  2 0.8343685 0.1656315

$number.vmail.messages
   number.vmail.messages
Y        [,1]      [,2]
  1 0.1687169 0.2728064
  2 0.1003126 0.2325517

$total.day.minutes
   total.day.minutes
Y        [,1]      [,2]
  1 0.4993608 0.1430492
  2 0.5898349 0.1966870
```

Fig – 21: Examples of tables created by naiveBayes function

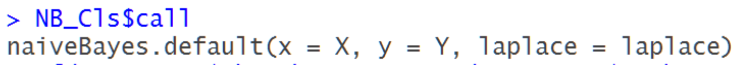
levels: Attribute levels of target class. In our present case, in “Churn”, False is labelled 1 and True is labelled 2. Hence:



```
> NB_Cls$levels
[1] "1" "2"
```

Fig – 22: levels of target class from naiveBayes class

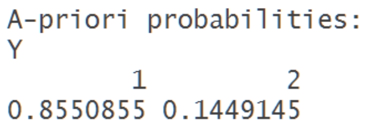
call: Default call of the function *naiveBayes*



```
> NB_Cls$call
naiveBayes.default(x = X, y = Y, laplace = laplace)
```

Fig – 23: call object from naiveBayes function

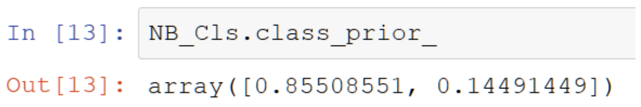
A-priori probabilities (1 – False, 2 – True) generated by naiveBayes are shown below. It represents the likelihood of occurrence of one target class irrespective of occurrence of other target class (of TrainData).



```
A-priori probabilities:
Y
        1         2
0.8550855 0.1449145
```

Fig – 24: A-Priori probabilities of TrainData target classes

In Python, we use *GaussianNB* function from *sklearn.naive_bayes* module. This function hardly takes any parameters. The following figure shows the A-priori probabilities generated by *GuassianNB*. This is similar to the one we obtained in R.



```
In [13]: NB_Cls.class_prior_

Out[13]: array([0.85508551, 0.14491449])
```

Fig – 25: A-priori probabilities generated by *GuassianNB*

2.2.1.5 Logistic Regression (LR) Model

Logistic regression analyses a dataset in which there are one or more independent variables that determine an outcome or a target class. The outcome is measured with a binomial variable in which there are only two possible outcomes like True/False; Yes/No; 1/0; Present/Absent etc. The logistic regression helps to find the best fitting model to describe the relationship between the binomial characteristic of dependent variable and a set of predictor variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit (**log**istic un**it**) transformation* of the probability of presence of the characteristic of target class(p):

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_nX_n$$

The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic\ of\ target\ class}{probability\ of\ absence\ of\ characteristic\ of\ target\ class}$$

&

$$logit(p) = ln(odds) = ln(\frac{p}{1-p})$$

Unlike linear regression, where parameters are estimated using minimum ordinary least squares (OLS) error, logistic regression uses maximum likelihood estimation (MLE) to choose the parameters. The logistic regressions' assumptions on the data to be modelled are:
-> Ratio of cases to variables – using discrete variables requires that there are responses in every given category
-> Absence of multicollinearity
-> No outliers
-> Linearity in the logit – the regression equation should have a linear relationship with the logit form of the target variable (as shown in the equations above); there is no assumption about the independent variables being related to one another
-> Independence of errors – there shouldn't be dependence or some kind of pattern among the errors of the independent variables

In R, we use *glm* (Generalised Linear Models) function from *stats* library. As the target variable consists of two classes, we use *family* parameter as "*binomial*". When we feed the value of this function to predict the test cases, we get probabilities. We label all these values between 0 and 0.5 as zeros and values between 0.5 and 1 as ones. Then we build the confusion matrix with these predicted values and original target classes. Summary of *glm* is shown below. It creates dummies of the categorical variables and gives estimate (coefficients), std.Error, z value and Pr values. Variables are could also be seen in various significance codes.

```
> summary(Log_Cls)

Call:
glm(formula = Churn ~ ., family = "binomial", data = TrainData)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.9298  -0.4991  -0.3117  -0.1654   3.0532  
```

```
Coefficients:
                              Estimate Std. Error z value Pr(>|z|)    
(Intercept)                   -9.66938    0.94427 -10.240  < 2e-16 ***
state2                         0.34920    0.76336   0.457 0.647346    
state3                         0.92004    0.75275   1.222 0.221613    
state4                         0.13349    0.84403   0.158 0.874333    
state5                         1.83285    0.78236   2.343 0.019143 *  
state6                         0.68981    0.76187   0.905 0.365241    
state7                         1.03260    0.72594   1.422 0.154903    
state8                         0.70480    0.80904   0.871 0.383669    
state9                         0.76690    0.74938   1.023 0.306133    
state10                        0.59860    0.76150   0.786 0.431823    
state11                        0.67882    0.77835   0.872 0.383138    
state12                       -0.20184    0.89543  -0.225 0.821661    
state13                        0.23365    0.90280   0.259 0.795783    
state14                        0.88335    0.74755   1.182 0.237340    
state15                       -0.19498    0.83266  -0.234 0.814853    
state16                        0.45304    0.75331   0.601 0.547571    
state17                        1.07942    0.72994   1.479 0.139199    
state18                        0.81988    0.76499   1.072 0.283827    
state19                        0.57319    0.83590   0.686 0.492892    
state20                        1.18498    0.74356   1.594 0.111015    
state21                        1.15327    0.71672   1.609 0.107597    
state22                        1.35556    0.72815   1.862 0.062651 .  
state23                        1.39866    0.71337   1.961 0.049921 *  
state24                        1.17177    0.71589   1.637 0.101669    
state25                        0.60995    0.77418   0.788 0.430772    
state26                        1.36447    0.72781   1.875 0.060824 .  
state27                        1.88038    0.71678   2.623 0.008706 ** 
state28                        0.61927    0.75398   0.821 0.411457    
state29                        0.16454    0.79683   0.206 0.836408    
state30                        0.33520    0.80523   0.416 0.677202    
state31                        1.20244    0.76777   1.566 0.117317    
state32                        1.60019    0.70948   2.255 0.024105 *  
state33                        0.48880    0.78684   0.621 0.534459    
state34                        1.26070    0.72535   1.738 0.082200 .  
state35                        1.17940    0.71987   1.638 0.101350    
state36                        0.70119    0.74634   0.940 0.347471    
state37                        0.88750    0.75427   1.177 0.239343    
state38                        0.78515    0.73611   1.067 0.286145    
state39                        1.16830    0.77955   1.499 0.133952    
state40                       -0.10698    0.82003  -0.130 0.896206    
state41                        1.78415    0.73726   2.420 0.015522 *  
state42                        0.83536    0.76218   1.096 0.273072    
state43                        0.29104    0.82099   0.354 0.722965    
state44                        1.66291    0.70792   2.349 0.018823 *  
state45                        1.05175    0.74418   1.413 0.157569    
state46                       -0.42554    0.82344  -0.517 0.605305    
state47                        0.10969    0.77787   0.141 0.887862    
state48                        1.43567    0.72410   1.983 0.047402 *  
state49                        0.29044    0.78058   0.372 0.709834    
state50                        0.58890    0.73357   0.803 0.422096    
state51                        0.30766    0.75502   0.407 0.683654    
account.length                 0.24165    0.34695   0.696 0.486127    
international.plan2            2.18380    0.15294  14.278  < 2e-16 ***
voice.mail.plan2              -2.10457    0.59314  -3.548 0.000388 ***
number.vmail.messages          1.90681    0.94933   2.009 0.044582 *  
total.day.minutes              4.60327    0.38898  11.834  < 2e-16 ***
total.day.calls                0.66456    0.47147   1.410 0.158669    
total.eve.minutes              2.82449    0.43057   6.560 5.38e-11 ***
total.eve.calls                0.16961    0.49082   0.346 0.729664    
total.night.minutes            1.46247    0.42815   3.416 0.000636 ***
total.night.calls              0.02268    0.41517   0.055 0.956435    
total.intl.minutes             1.67333    0.42178   3.967 7.27e-05 ***
total.intl.calls              -1.80164    0.51376  -3.507 0.000454 ***
number.customer.service.calls  4.82612    0.36818  13.108  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2758.3  on 3332  degrees of freedom
Residual deviance: 2072.2  on 3269  degrees of freedom
AIC: 2200.2

Number of Fisher Scoring iterations: 6
```

In Python, we use replace the target classes by labels - 0 for False and 1 for True. Then we create dummies over the categorical independent variables and join all these with continuous variables to get logit datasets. We feed this logit TrainData to *LogisticRegression* function imported from *sklearn.linear_model*. We predict the test cases and build the confusion matrix.

3. Conclusion

3.1 Model Evaluation

In order to evaluate a classification model, there are various error metrics. Here we consider Accuracy and False Negative Rate (FNR) to evaluate all the models developed above. We have built confusion matrix for every model, both in R and Python, from which we obtain the values of accuracy and FNR. These terms are defined by:

Confusion matrix –

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

TP – True Positive; TN – True Negative; FP – False Positive; FN – False Negative

$$Accuracy = \frac{TP + TN}{Total\ number\ of\ observations} * 100$$

$$FNR = \frac{FN}{FN + TP} * 100$$

3.1.1 Decision Tree error metrics:

In R,

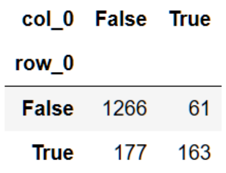
Confusion matrix obtained is –

```
> Conf_Mat_DT
        predicted
observed    1    2
       1 1337  106
       2   70  154
```

Accuracy = 89.44211
FNR = 31.2500

In Python,

Confusion matrix obtained is –



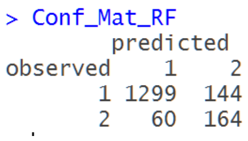
col_0	False	True
row_0		
False	1266	61
True	177	163

Accuracy = 85.72285542891422
FNR = 52.05882352941177

3.1.2 Random Forest error metrics:

In R,

Confusion matrix obtained is –

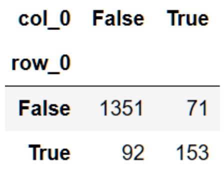


```
> Conf_Mat_RF
        predicted
observed    1    2
       1 1299  144
       2   60  164
```

Accuracy = 87.76245
FNR = 26.78571

In Python,

Confusion matrix obtained is –



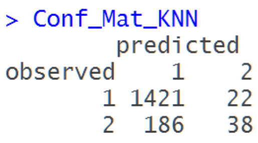
col_0	False	True
row_0		
False	1351	71
True	92	153

Accuracy = 90.22195560887822
FNR = 37.55102040816327

3.1.3 KNN Classifier error metrics:

In R,

Confusion matrix obtained is –

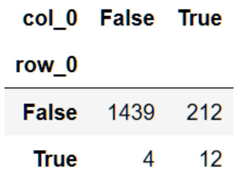


```
> Conf_Mat_KNN
        predicted
observed    1    2
       1 1421   22
       2  186   38
```

Accuracy = 87.52250
FNR = 83.03571

In Python,

Confusion matrix obtained is –



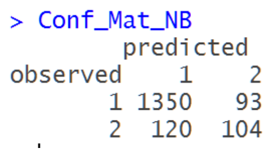
col_0	False	True
row_0		
False	1439	212
True	4	12

Accuracy = 87.04259148170365
FNR = 25.0

3.1.4 Naive Bayes Classifier error metrics:

In R,

Confusion matrix obtained is –



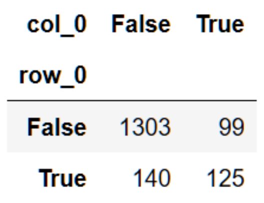
```
> Conf_Mat_NB
        predicted
observed    1    2
       1 1350   93
       2  120  104
```

Accuracy = 87.22256

FNR = 53.57143

In Python,

Confusion matrix obtained is –



col_0	False	True
row_0		
False	1303	99
True	140	125

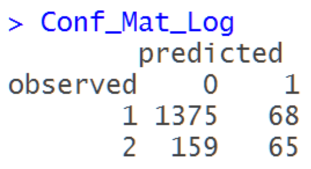
Accuracy = 85.6628674265147

FNR = 52.83018867924528

3.1.5 Logistic Regression error metrics

In R,

Confusion matrix obtained is –



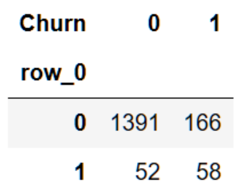
```
> Conf_Mat_Log
        predicted
observed    0    1
       1 1375   68
       2  159   65
```

Accuracy = 86.38272

FNR = 70.98214

In Python,

Confusion matrix obtained is –



Churn	0	1
row_0		
0	1391	166
1	52	58

Accuracy = 86.92261547690462

FNR = 47.27272727272727

3.2 Model Selection

A comparative investigation, w.r.t. both accuracy and FNR, of the above results, Random Forest classifier outperforms all other models. Though FNR of KNN Classifier in Python is the least but Accuracy is compromised with it a little.

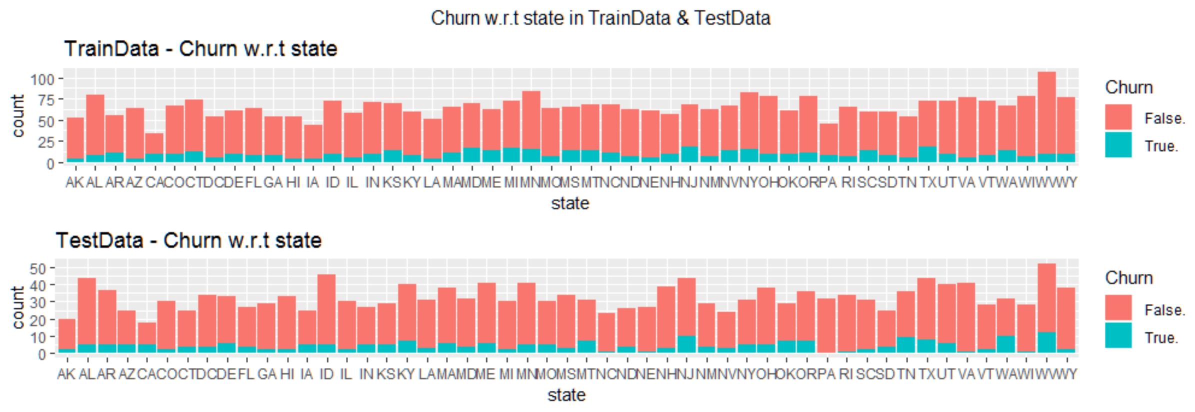
3.3 Points to note

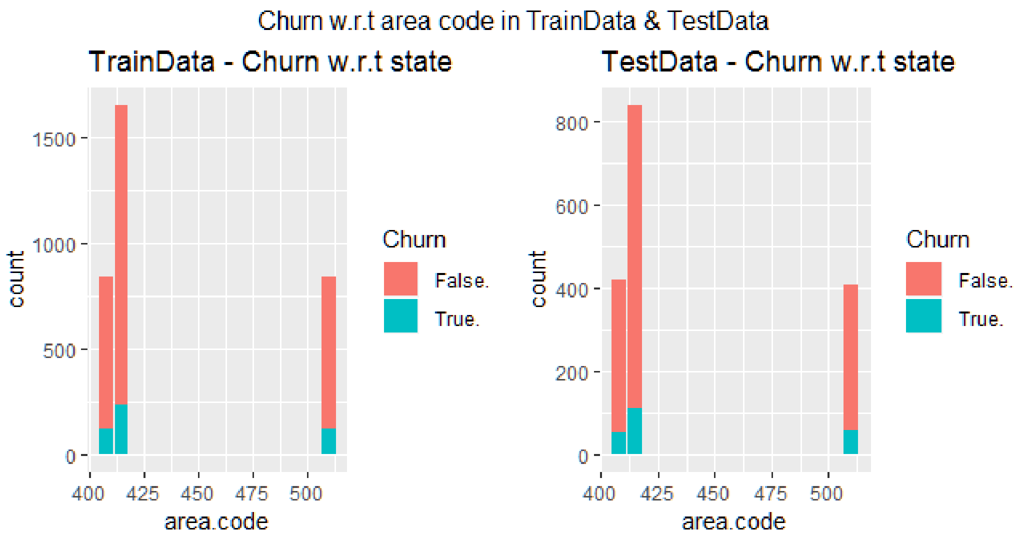
-> Customers from area code 415 are highly churning out. Care needs to be taken with regard to this particular area.
-> Customers with no international plan and no voice mail plan are also churning out more likely. They could be retained with attractive schemes.
-> Customers with zero "number vmails messages" have high churn rate.
-> Customers' dissatisfaction is clearly observed when we look at "number customer service calls" column. When it is between 1 and 3, churn rate is high. Services must be improved to solve customer issues in the very first attempt.

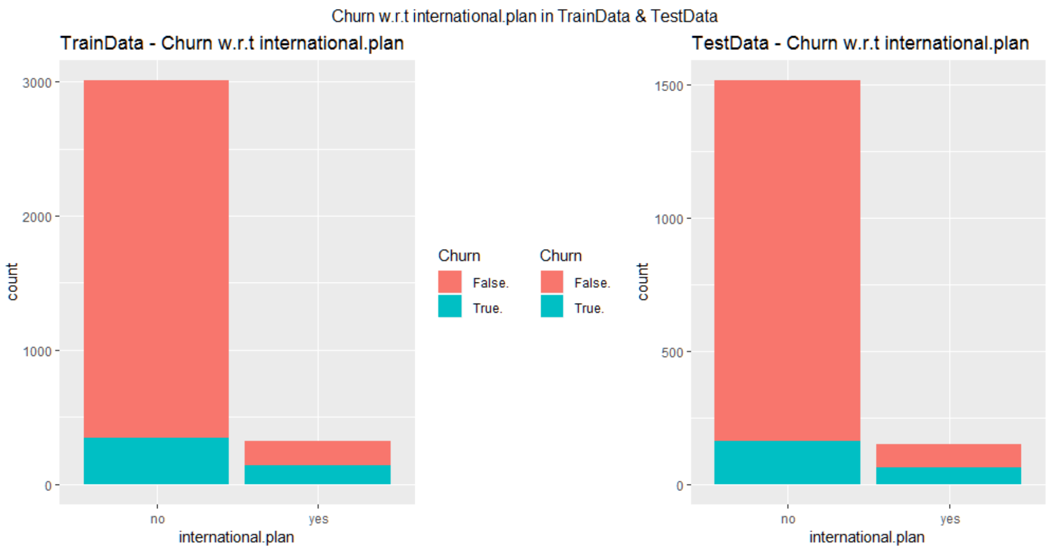
4. Appendix: Extra figures

4.1 From R

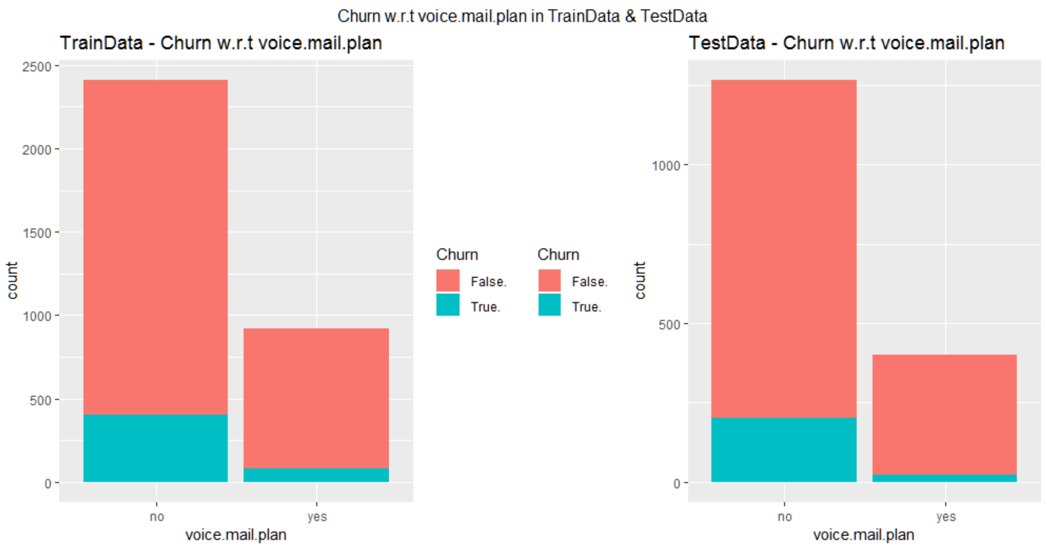
4.1.1 Univariate Analysis graphs



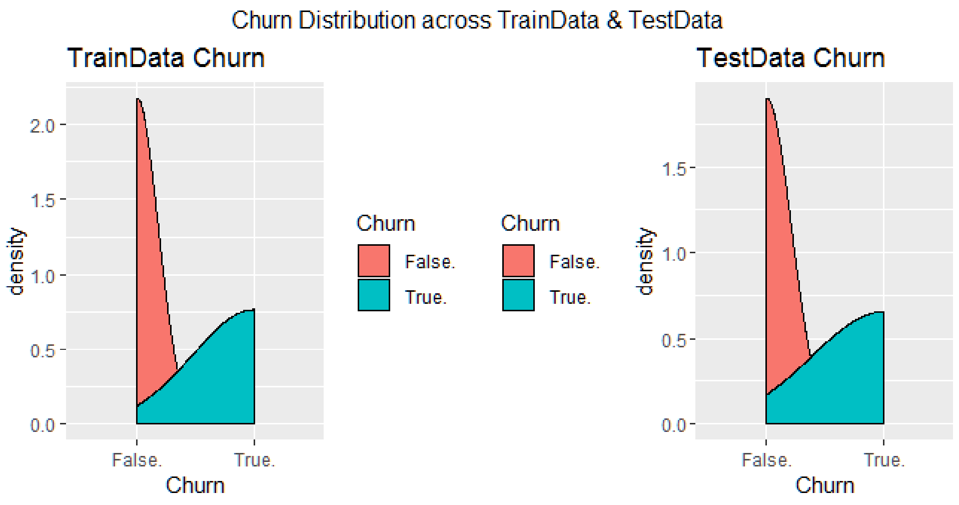




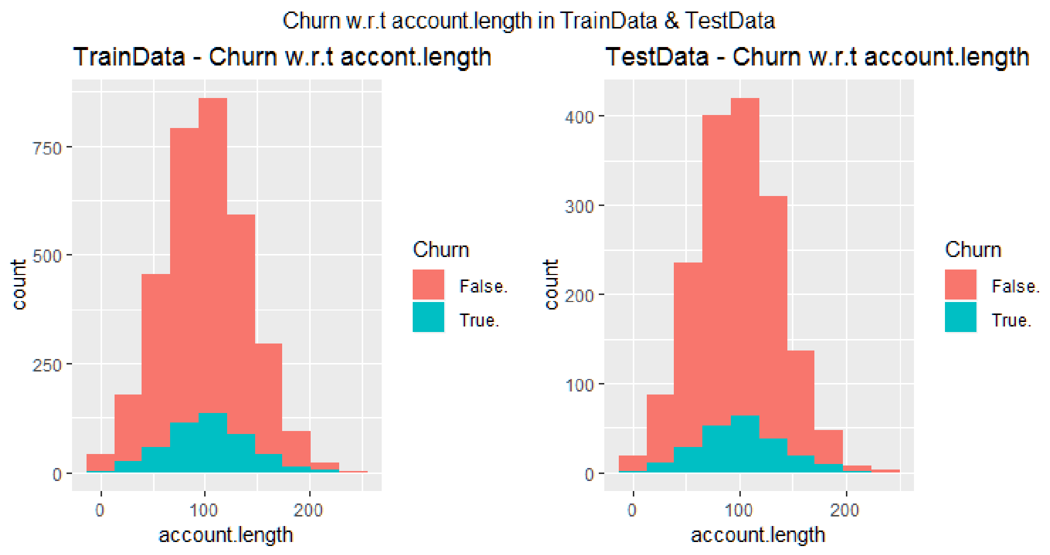
Churn w.r.t international.plan in TrainData & TestData
TrainData - Churn w.r.t international.plan
TestData - Churn w.r.t international.plan
count
3000
2000
1000
0
1500
1000
500
0
no
yes
international.plan
Churn
False.
True.

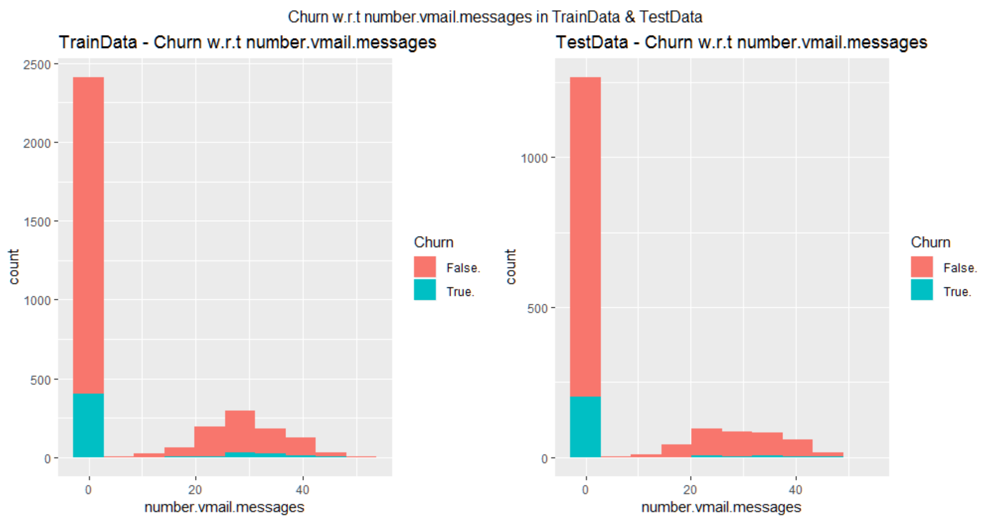


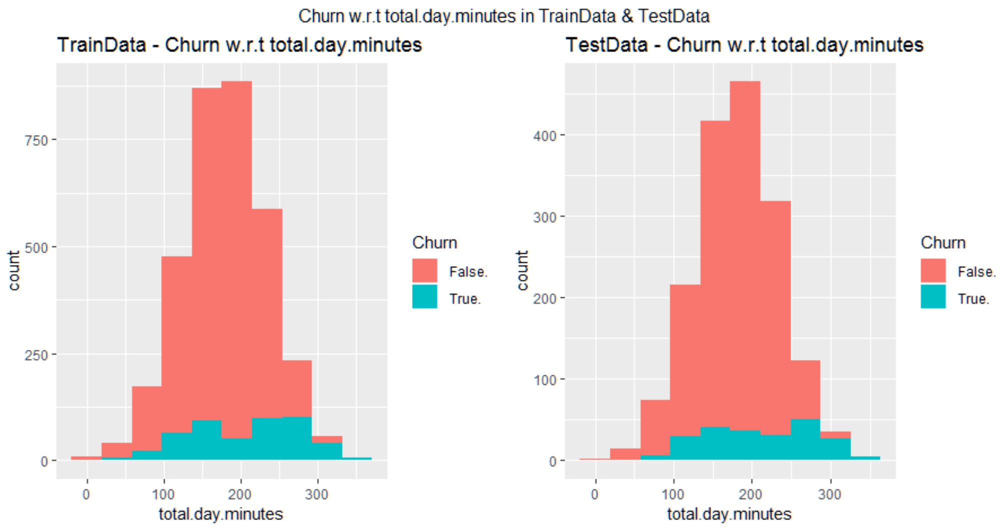
Churn w.r.t voice.mail.plan in TrainData & TestData
TrainData - Churn w.r.t voice.mail.plan
TestData - Churn w.r.t voice.mail.plan
count
2500
2000
1500
1000
500
0
no
yes
voice.mail.plan
Churn
False.
True.

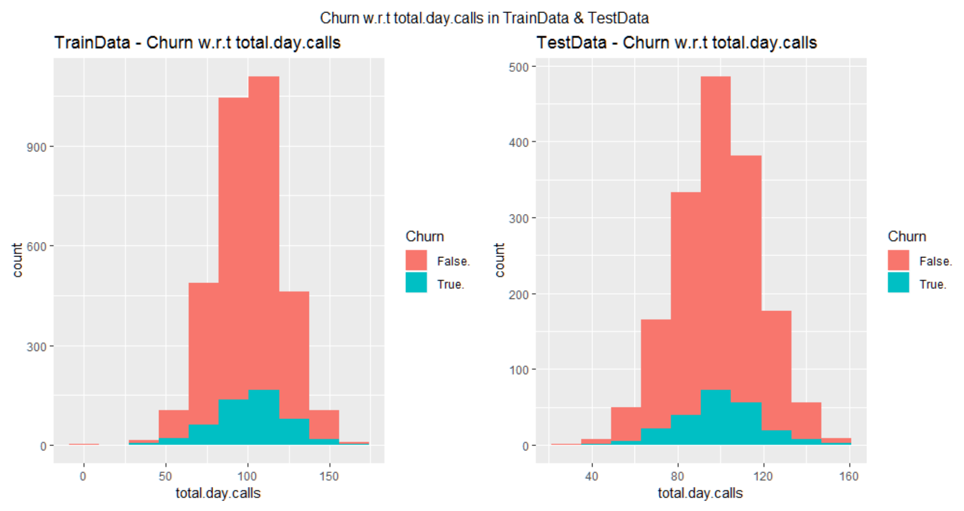


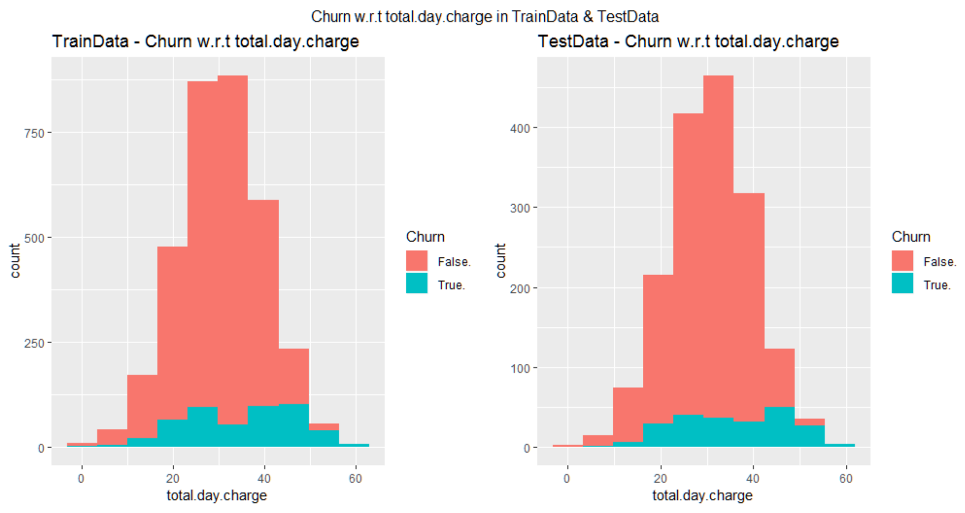
Churn Distribution across TrainData & TestData
TrainData Churn
TestData Churn
density
2.0
1.5
1.0
0.5
0.0
False.
True.
Churn

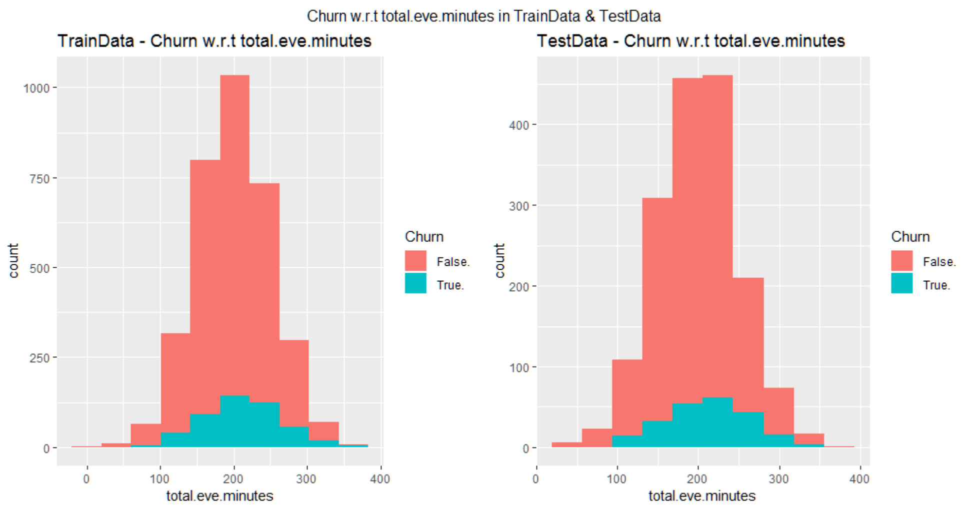

Churn w.r.t account.length in TrainData & TestData
TrainData - Churn w.r.t accont.length
TestData - Churn w.r.t account.length
count
account.length
Churn
False.
True.
0
250
500
750
100
200
300
400

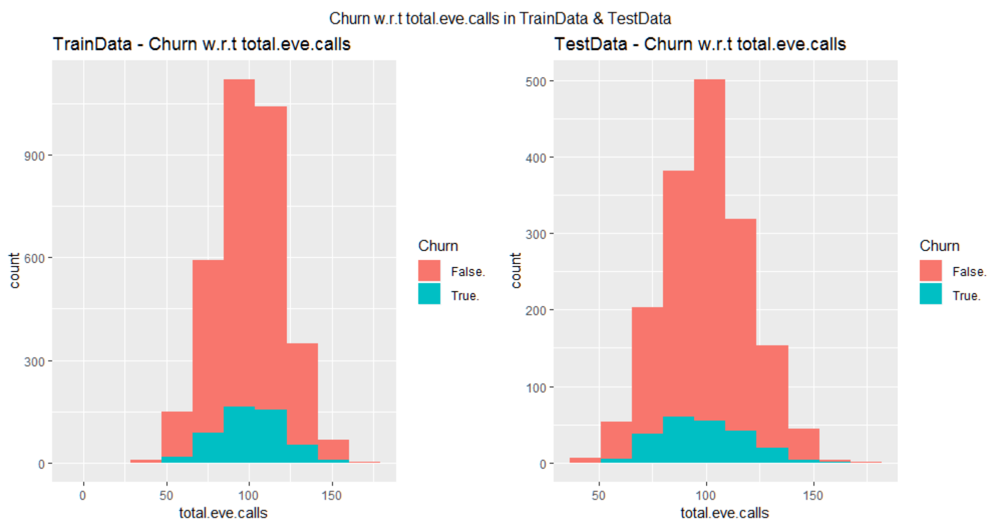

Churn w.r.t number.vmail.messages in TrainData & TestData
TrainData - Churn w.r.t number.vmail.messages
TestData - Churn w.r.t number.vmail.messages
count
number.vmail.messages
Churn
False.
True.
0
500
1000
1500
2000
2500
20
40


Churn w.r.t total.day.minutes in TrainData & TestData
TrainData - Churn w.r.t total.day.minutes
TestData - Churn w.r.t total.day.minutes
count
total.day.minutes
Churn
False.
True.
0
100
200
300
250
500
750
400

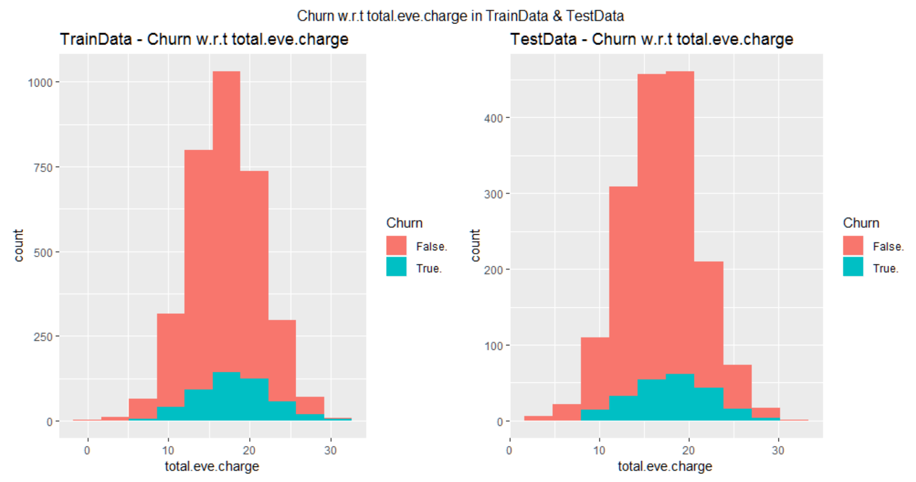

Churn w.r.t total.day.calls in TrainData & TestData
TrainData - Churn w.r.t total.day.calls
TestData - Churn w.r.t total.day.calls
count
total.day.calls
Churn
False.
True.


Churn w.r.t total.day.charge in TrainData & TestData
TrainData - Churn w.r.t total.day.charge
TestData - Churn w.r.t total.day.charge
count
total.day.charge
Churn
False.
True.

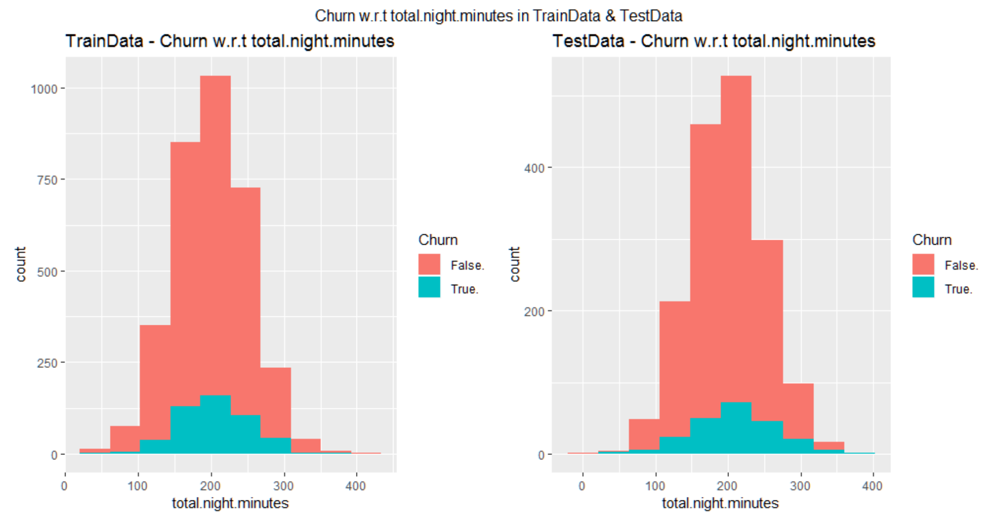

Churn w.r.t total.eve.minutes in TrainData & TestData
TrainData - Churn w.r.t total.eve.minutes
TestData - Churn w.r.t total.eve.minutes
count
total.eve.minutes
Churn
False.
True.



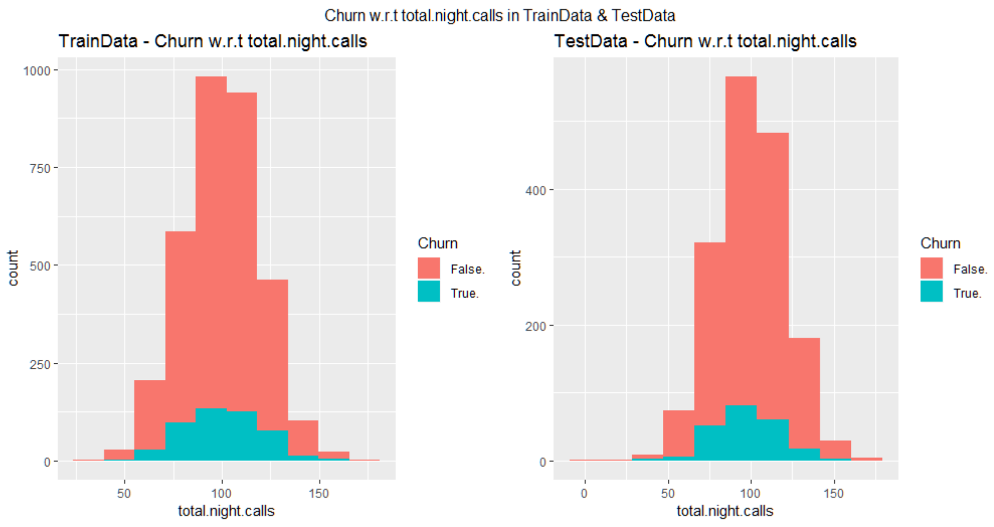
Churn w.r.t total.eve.calls in TrainData & TestData
TrainData - Churn w.r.t total.eve.calls
TestData - Churn w.r.t total.eve.calls
count
total.eve.calls
Churn
False.
True.

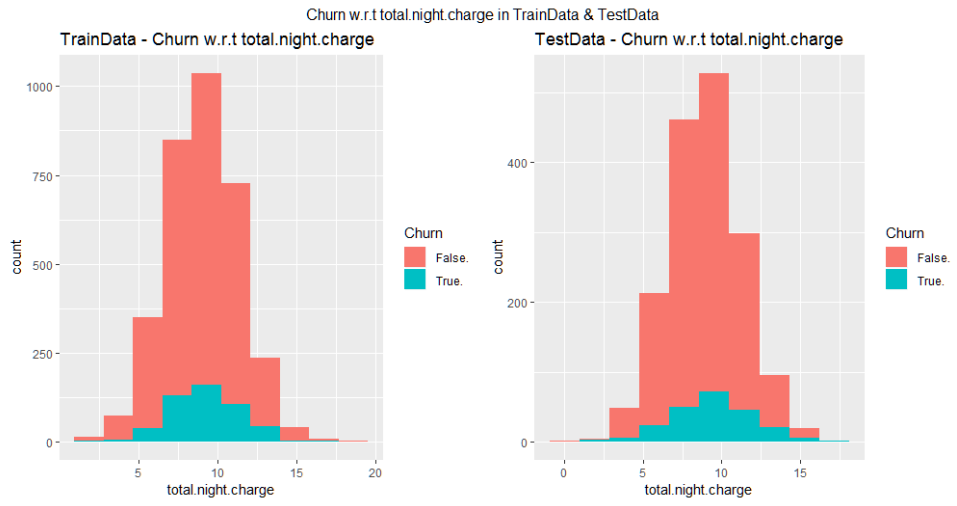


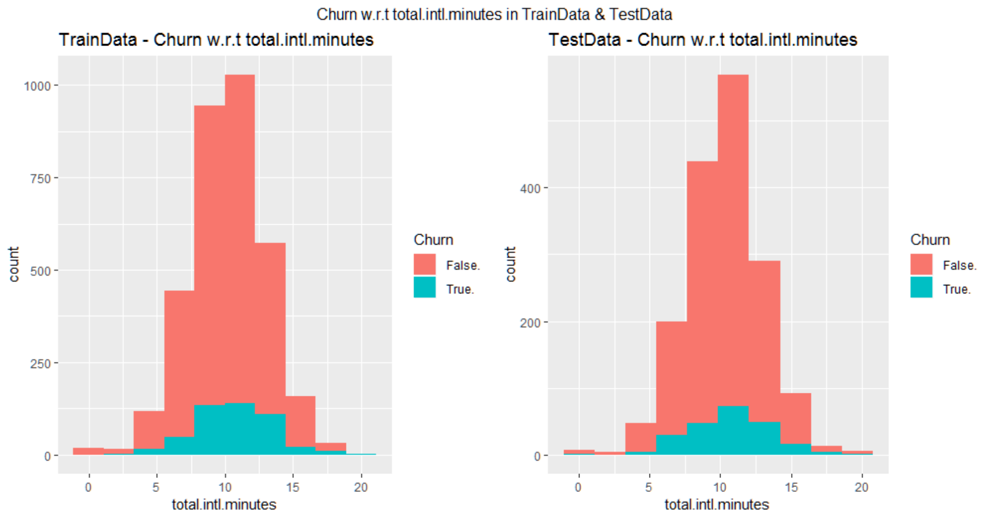
Churn w.r.t total.eve.charge in TrainData & TestData
TrainData - Churn w.r.t total.eve.charge
TestData - Churn w.r.t total.eve.charge
count
total.eve.charge
Churn
False.
True.

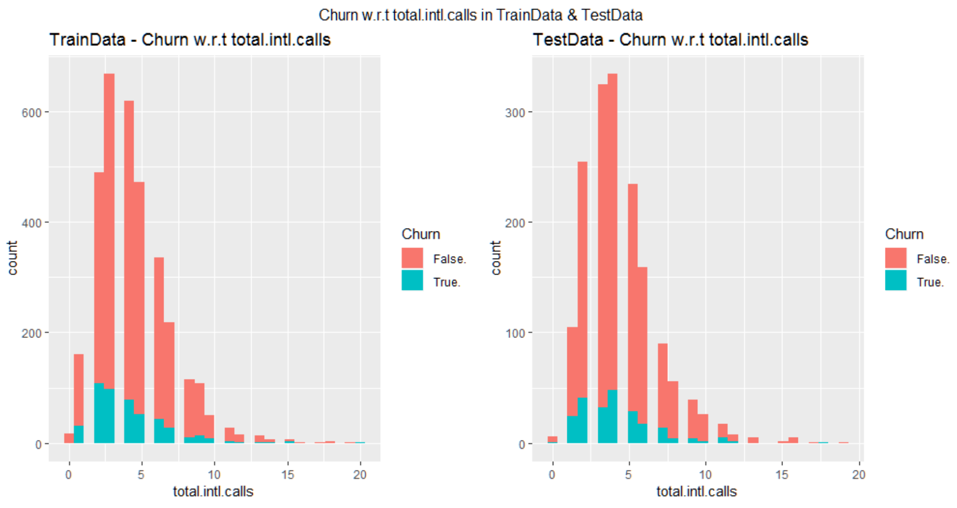


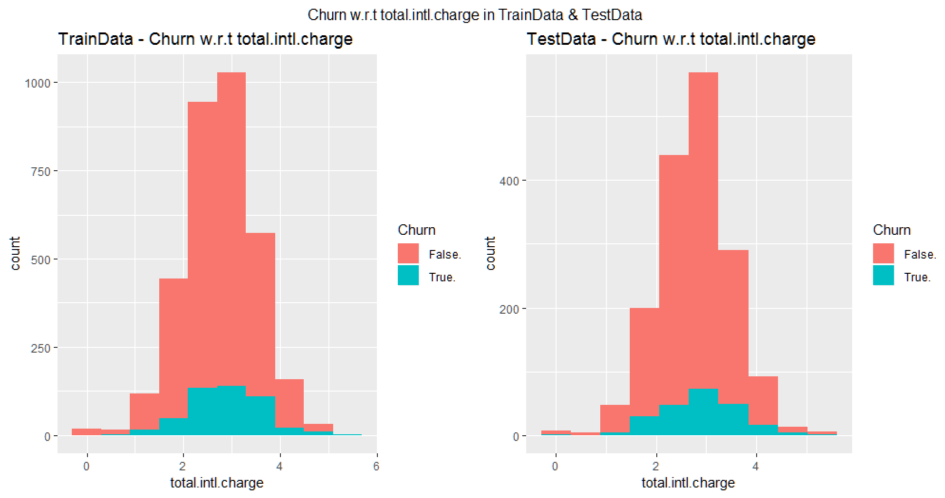
Churn w.r.t total.night.minutes in TrainData & TestData
TrainData - Churn w.r.t total.night.minutes
TestData - Churn w.r.t total.night.minutes
count
total.night.minutes
Churn
False.
True.

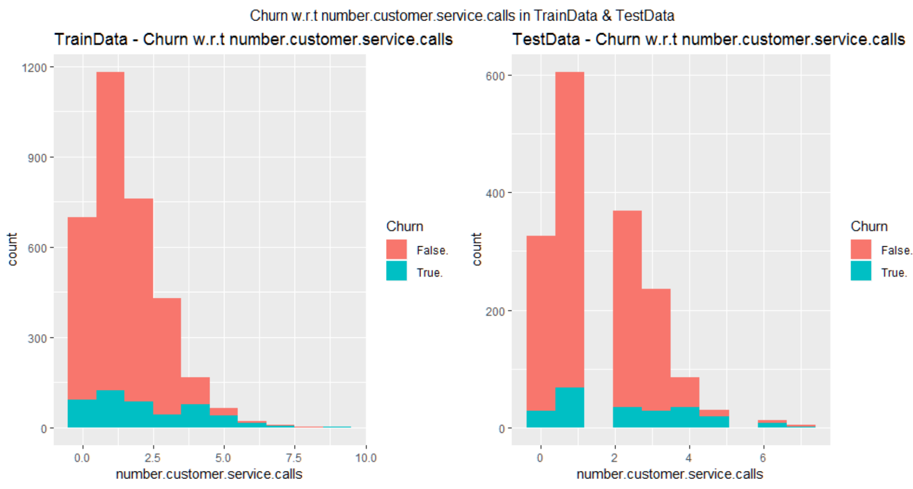

Churn w.r.t total.night.calls in TrainData & TestData
TrainData - Churn w.r.t total.night.calls
TestData - Churn w.r.t total.night.calls
count
total.night.calls
Churn
False.
True.


Churn w.r.t total.night.charge in TrainData & TestData
TrainData - Churn w.r.t total.night.charge
TestData - Churn w.r.t total.night.charge
count
total.night.charge
Churn
False.
True.


Churn w.r.t total.intl.minutes in TrainData & TestData
TrainData - Churn w.r.t total.intl.minutes
TestData - Churn w.r.t total.intl.minutes
count
total.intl.minutes
Churn
False.
True.

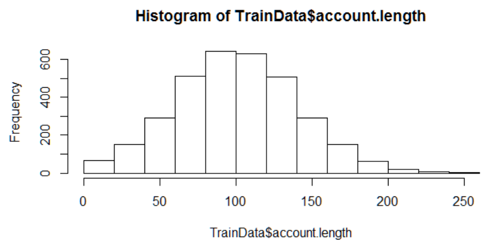

Churn w.r.t total.intl.calls in TrainData & TestData
TrainData - Churn w.r.t total.intl.calls
TestData - Churn w.r.t total.intl.calls
count
total.intl.calls
Churn
False.
True.

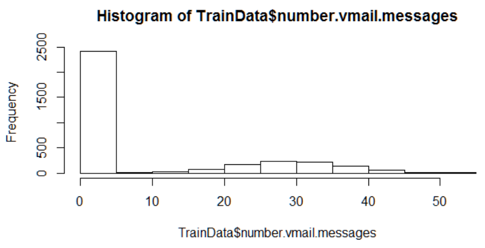

Churn w.r.t total.intl.charge in TrainData & TestData
TrainData - Churn w.r.t total.intl.charge
TestData - Churn w.r.t total.intl.charge
count
total.intl.charge
Churn
False.
True.

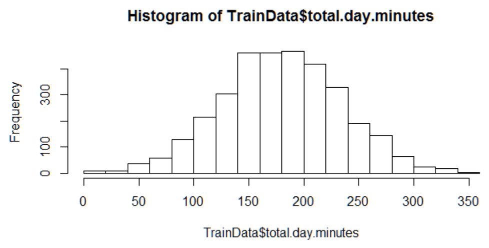

Churn w.r.t number.customer.service.calls in TrainData & TestData
TrainData - Churn w.r.t number.customer.service.calls
TestData - Churn w.r.t number.customer.service.calls
count
number.customer.service.calls
Churn
False.
True.

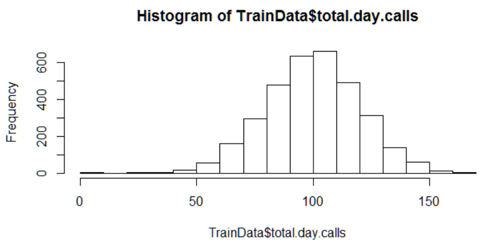
4.1.2 Histograms of continuous variables of TrainData

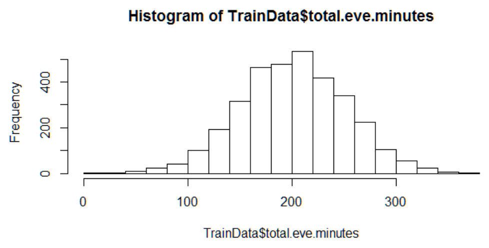
4.1.2.1 before normalisation

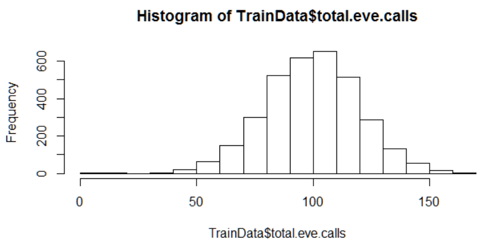


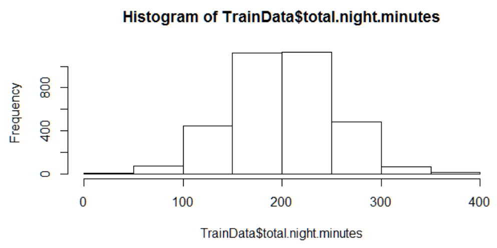


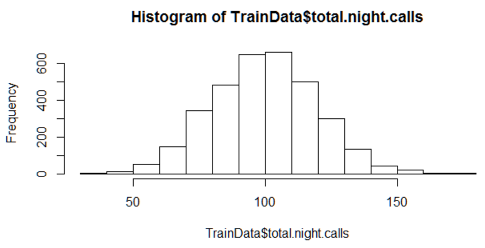


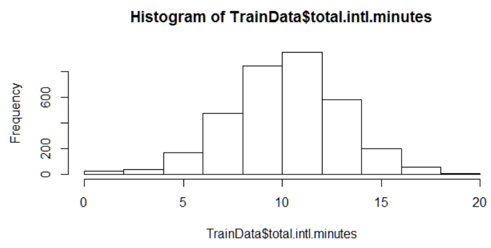


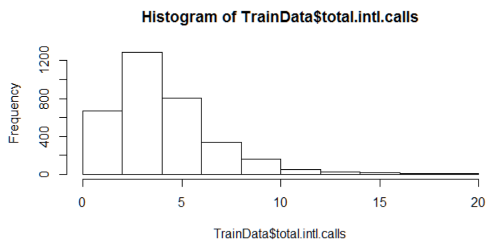


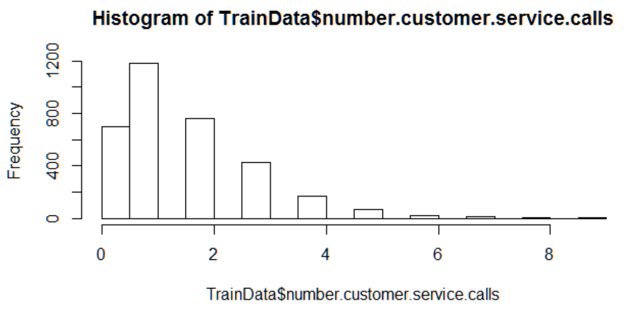




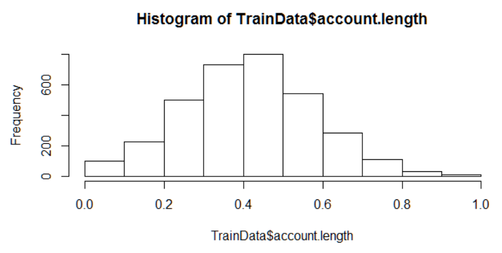


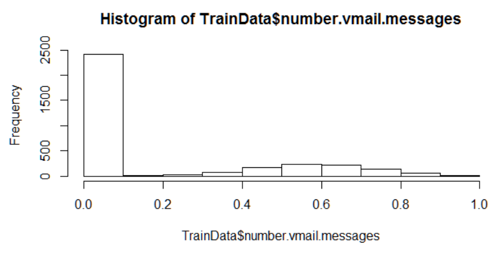


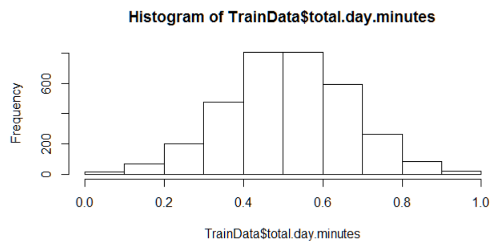


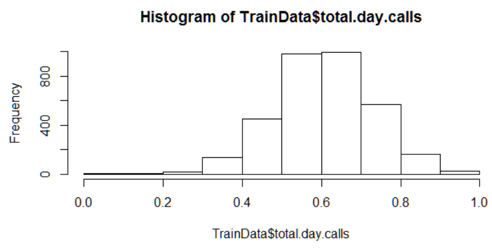


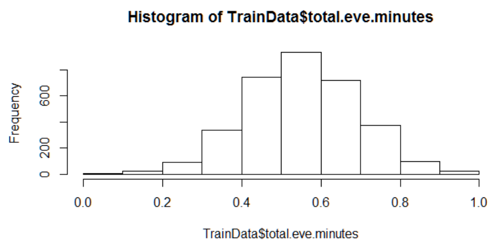
4.1.2.2 After Normalisation

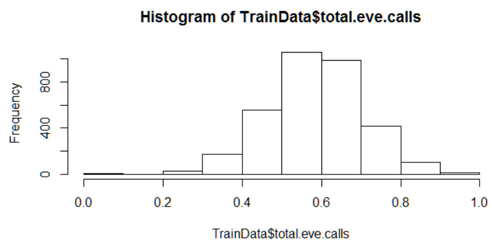


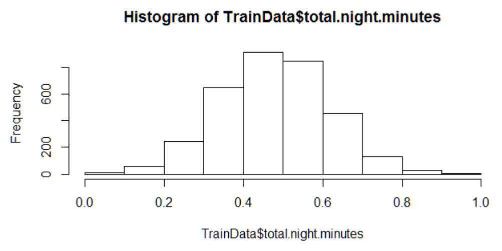


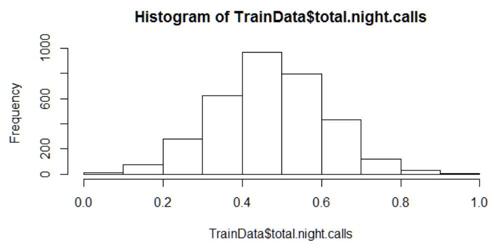


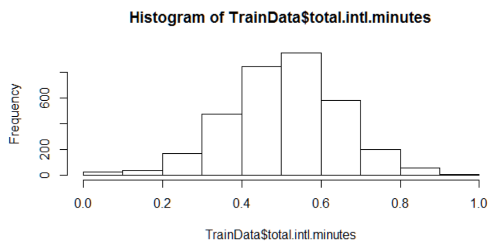


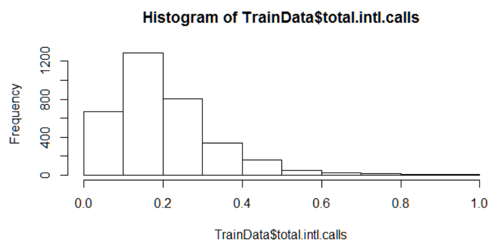


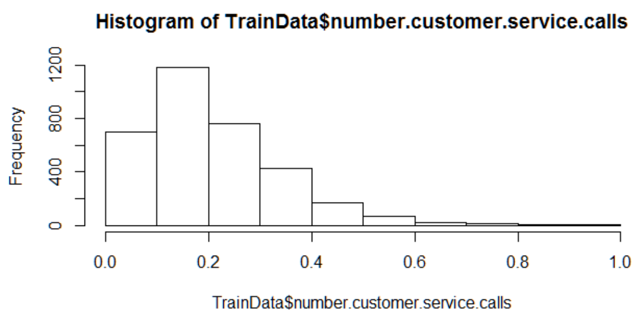






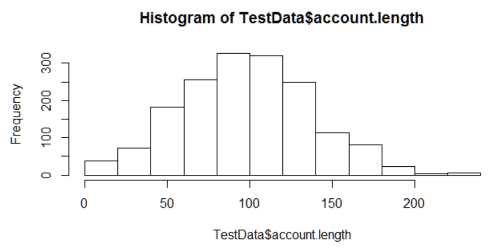


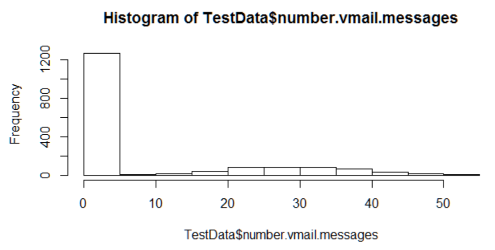


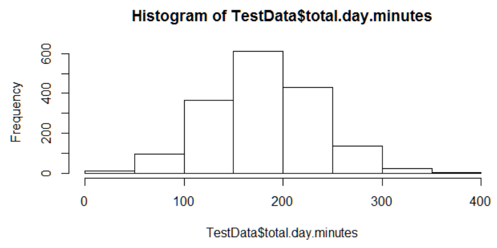


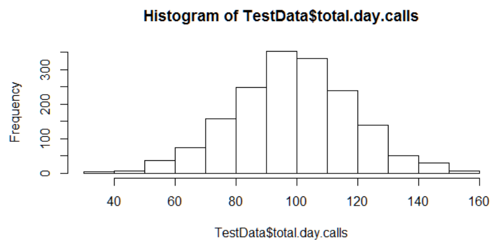
4.1.3 Histograms of continuous variables of TestData

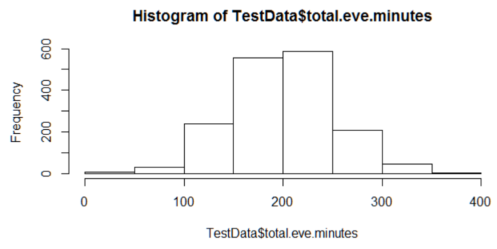
4.1.3.1 Before Normalisation

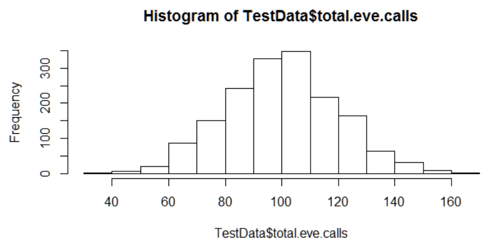


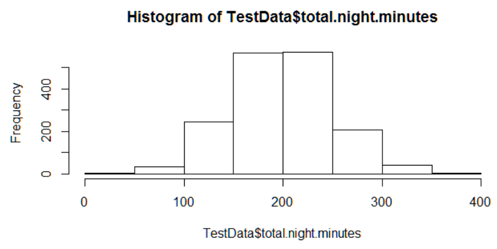


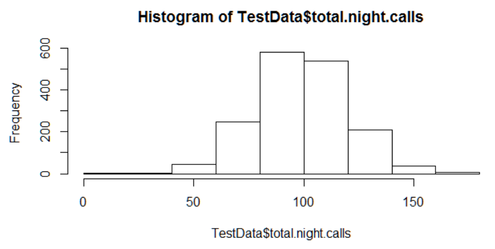


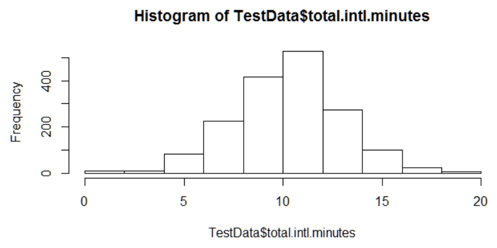


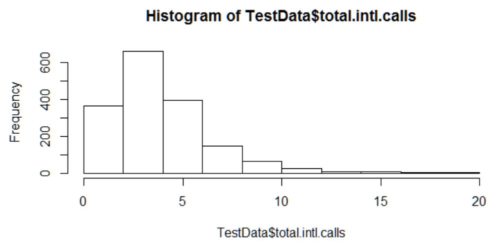


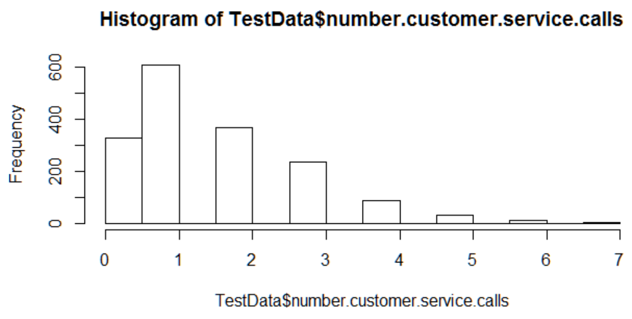




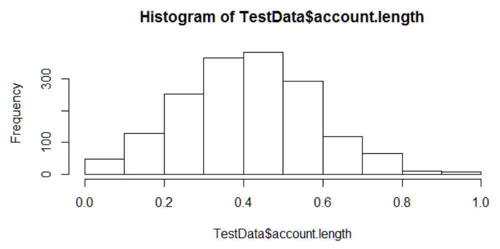


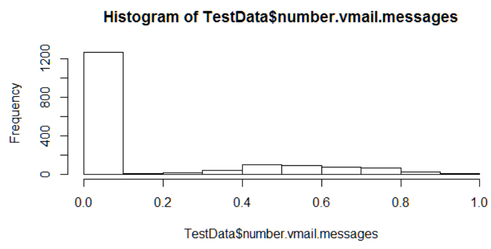


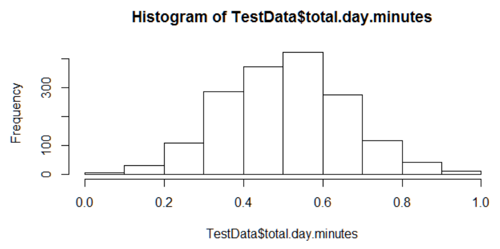


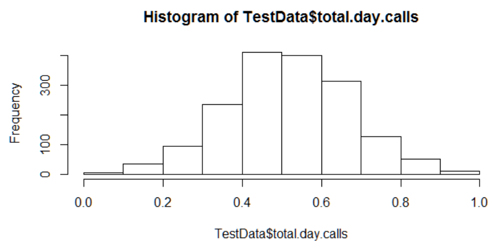


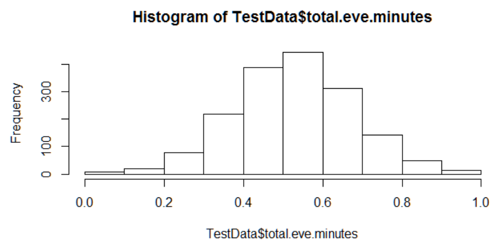
4.1.3.2 After Normalisation

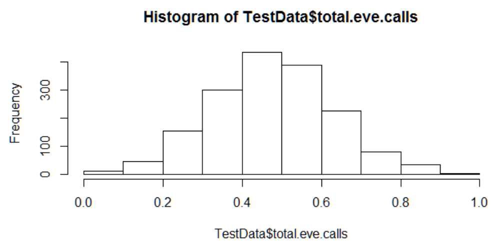


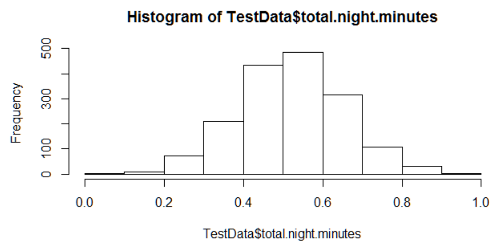


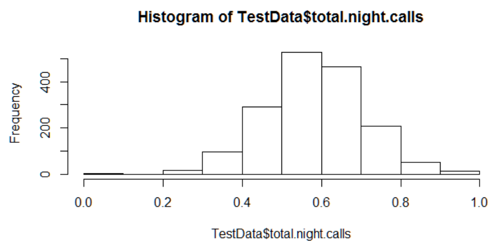


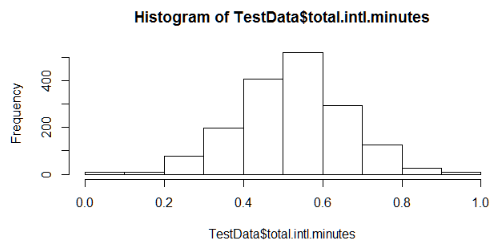


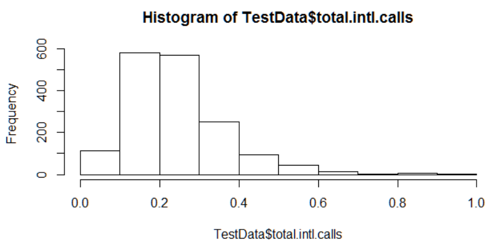


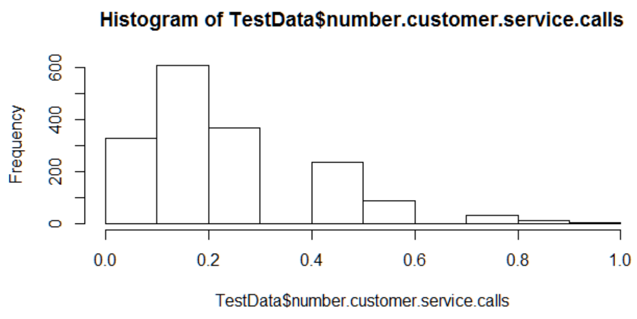






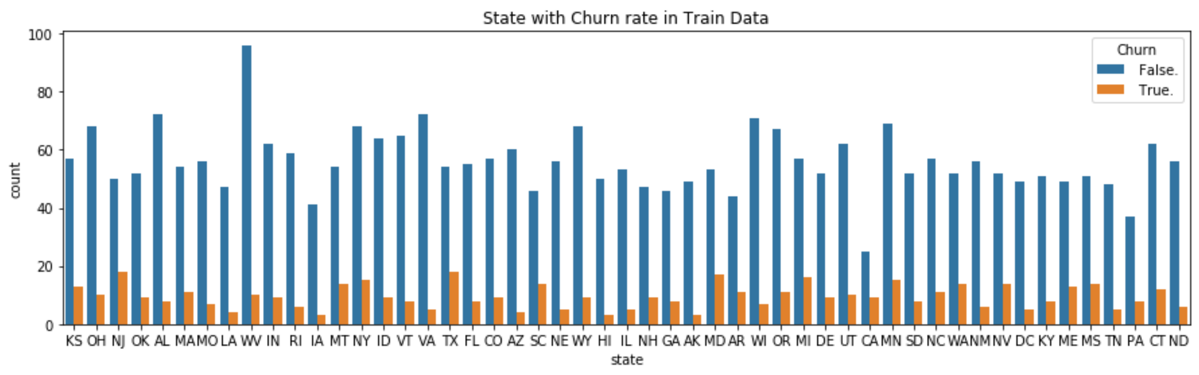


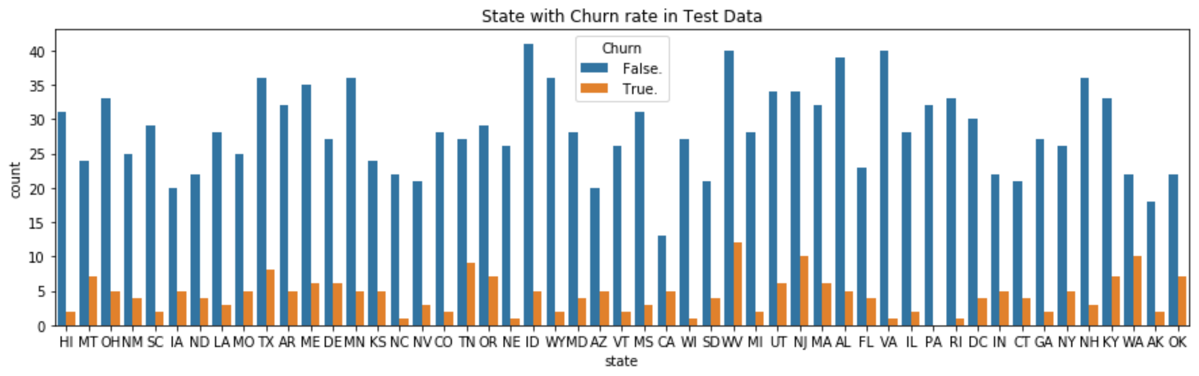


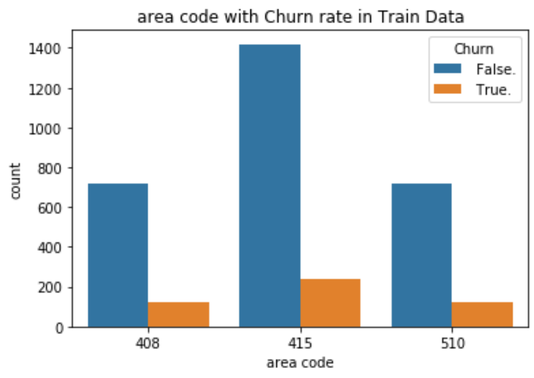


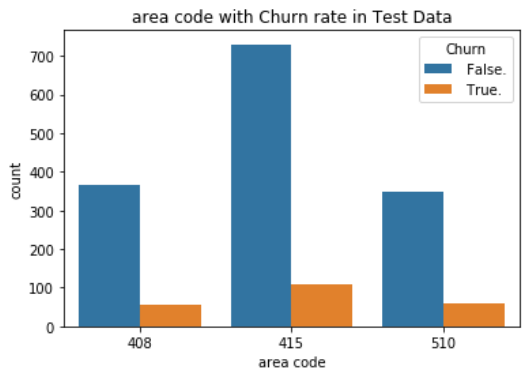
4.2 From Python

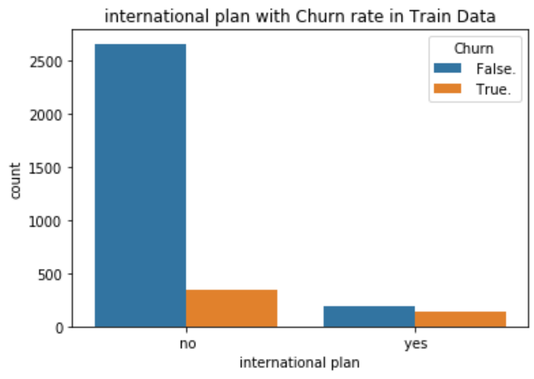
4.2.1 Univariate analysis graphs

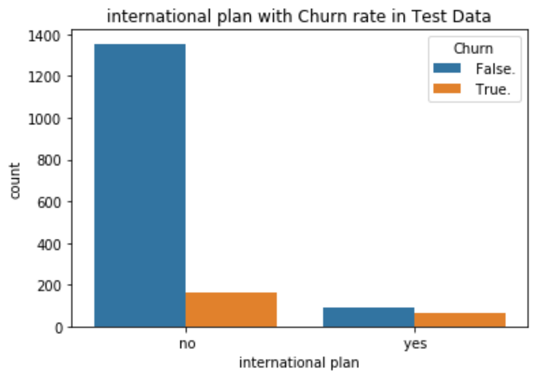


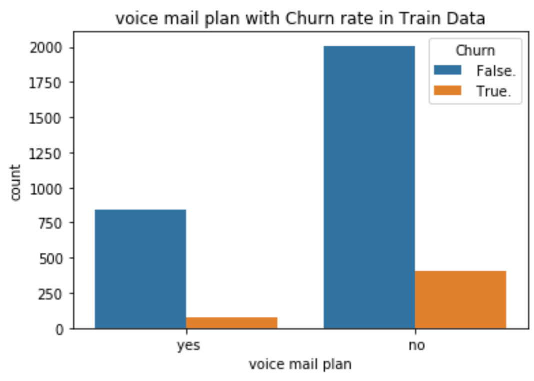


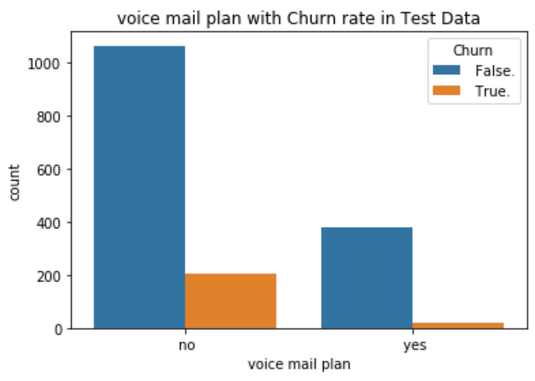


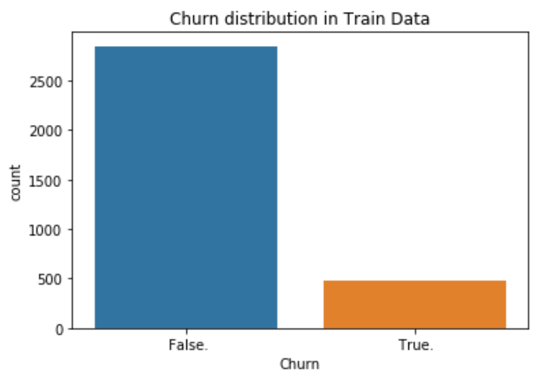


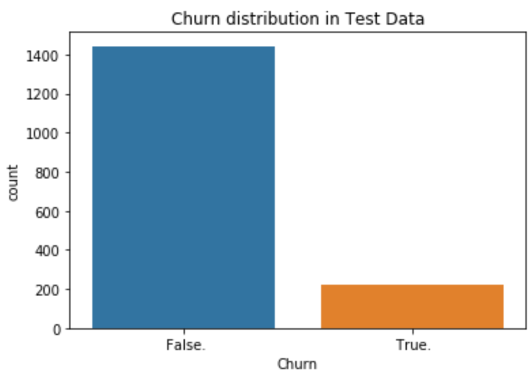


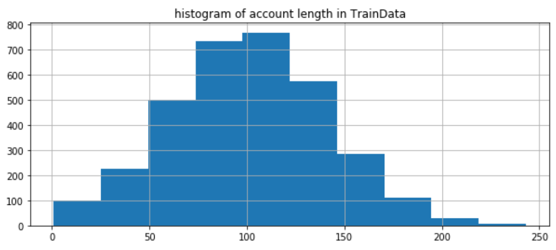


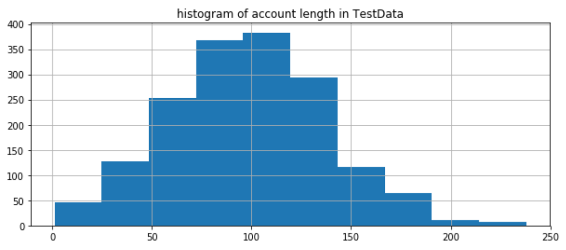

voice mail plan with Churn rate in Train Data
count
Churn
False.
True.
yes
no
voice mail plan

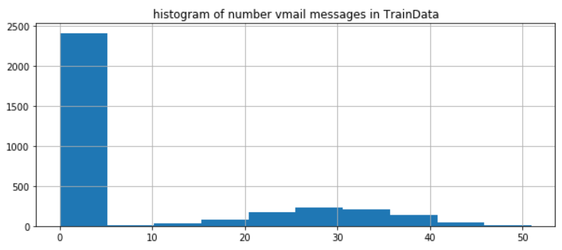

voice mail plan with Churn rate in Test Data
count
Churn
False.
True.
no
yes
voice mail plan

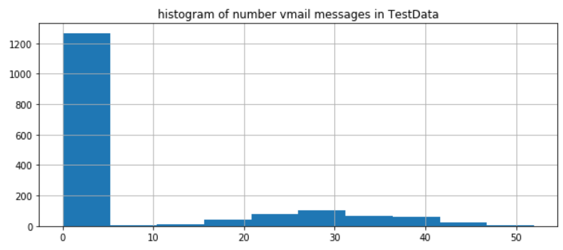

Churn distribution in Train Data
count
False.
True.
Churn

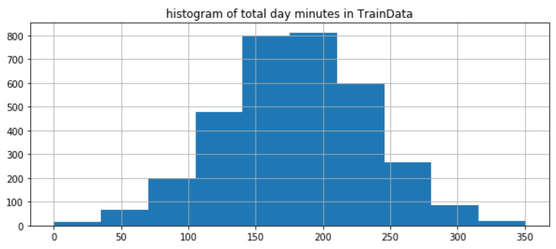

Churn distribution in Test Data
count
False.
True.
Churn

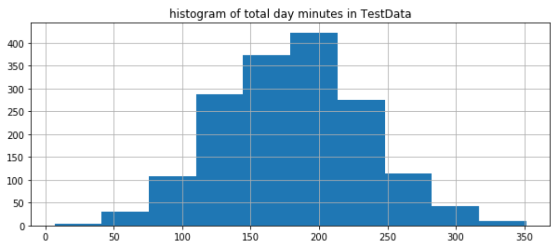

histogram of account length in TrainData

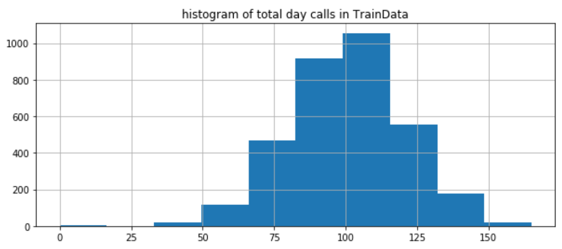

histogram of account length in TestData

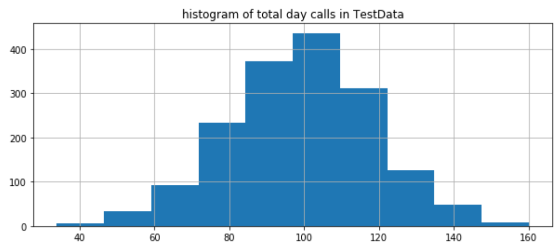

histogram of number vmail messages in TrainData

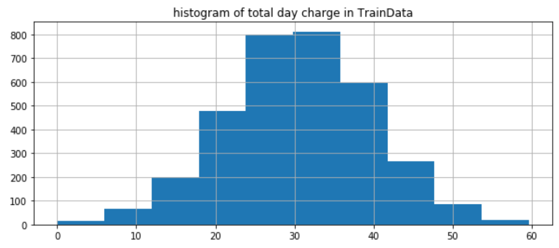
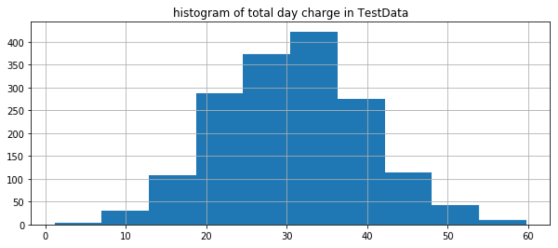
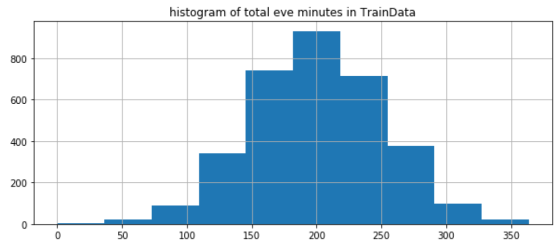
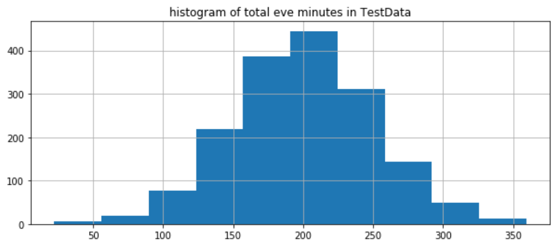
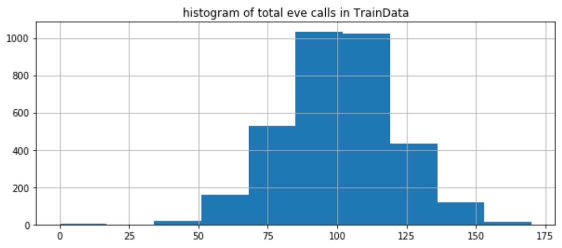
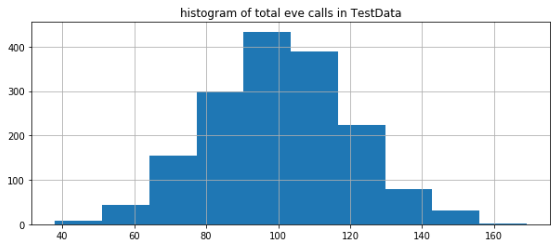
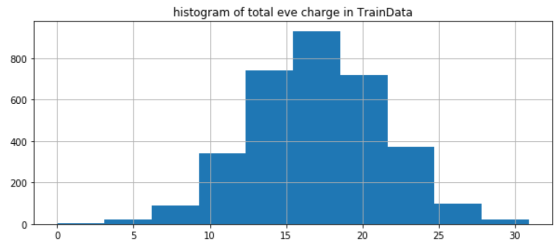
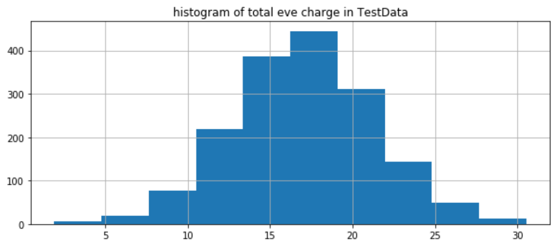
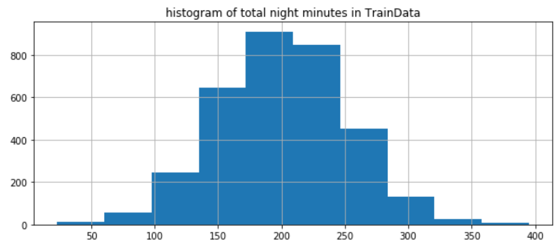
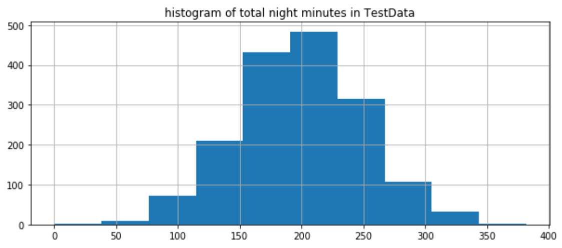
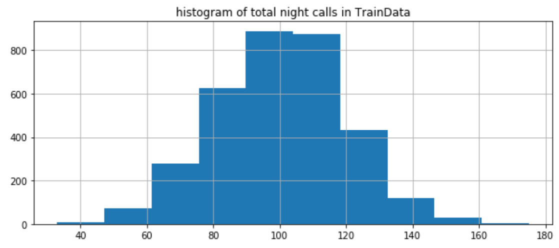
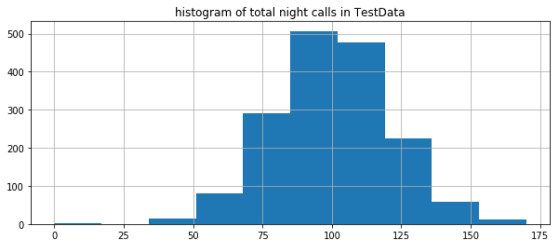
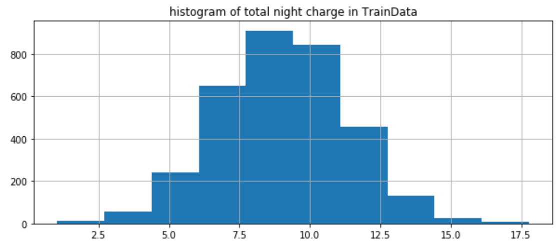
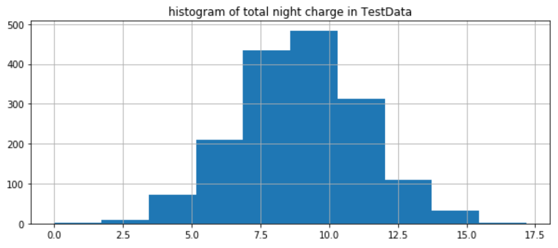

histogram of number vmail messages in TestData

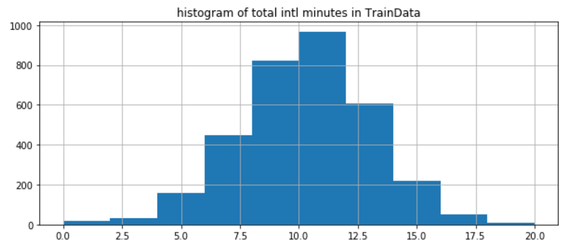

histogram of total day minutes in TrainData

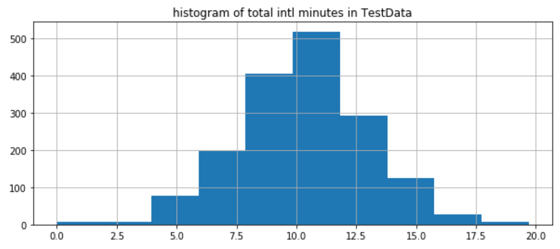

histogram of total day minutes in TestData

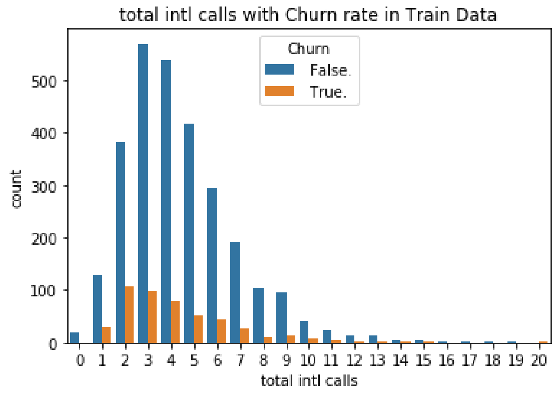

histogram of total day calls in TrainData

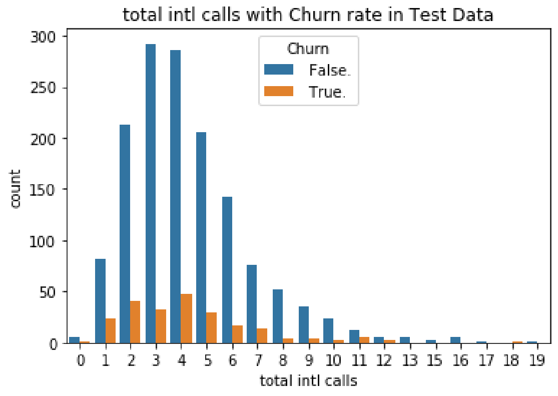

histogram of total day calls in TestData

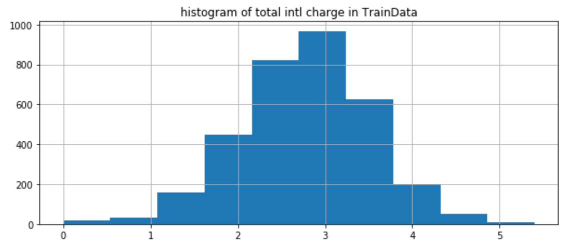

histogram of total day charge in TrainData

histogram of total day charge in TestData

histogram of total eve minutes in TrainData

histogram of total eve minutes in TestData

histogram of total eve calls in TrainData

histogram of total eve calls in TestData

histogram of total eve charge in TrainData

histogram of total eve charge in TestData

histogram of total night minutes in TrainData

histogram of total night minutes in TestData

histogram of total night calls in TrainData

histogram of total night calls in TestData

histogram of total night charge in TrainData

histogram of total night charge in TestData

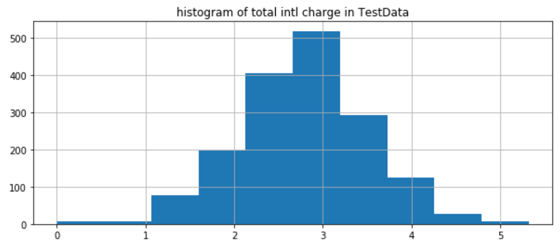


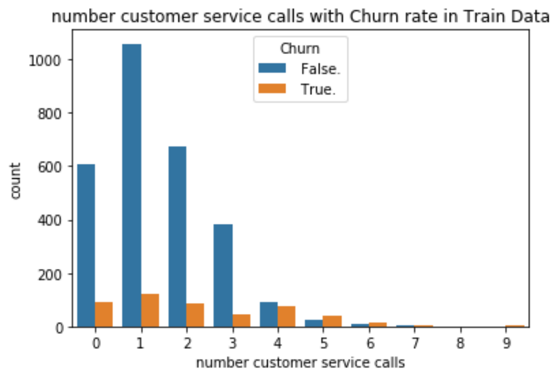


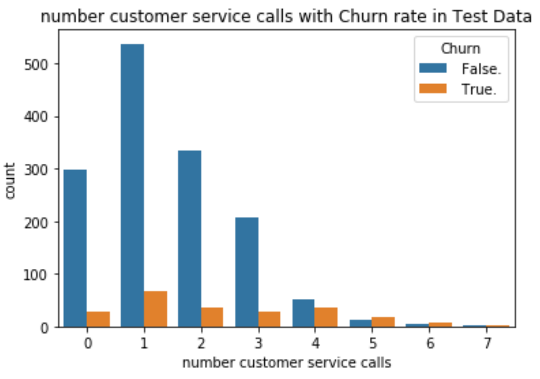




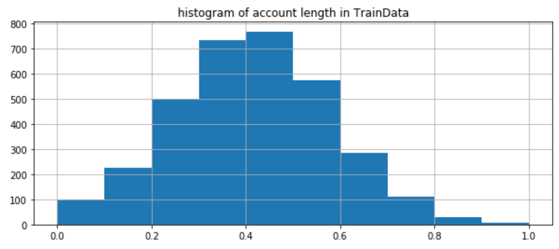


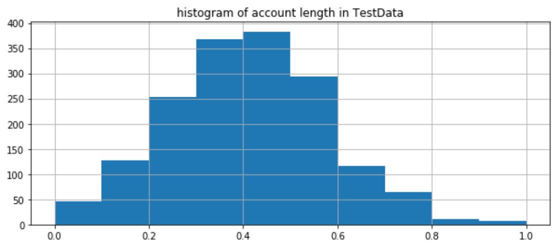


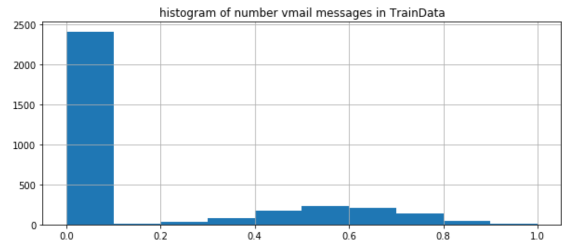


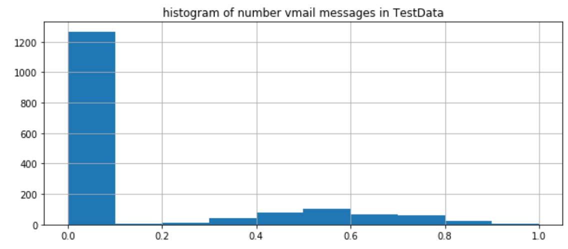


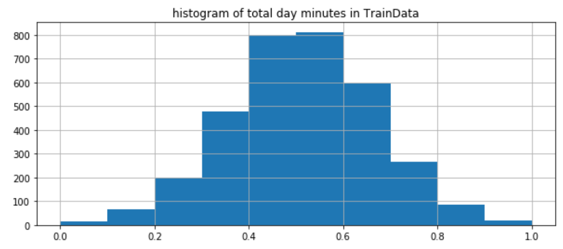
4.2.2 Histograms after normalisation

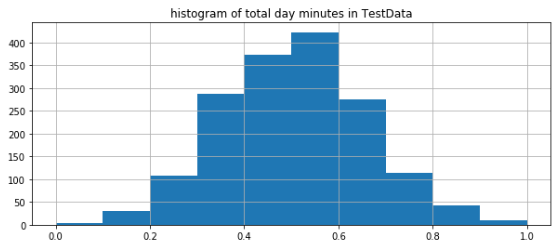


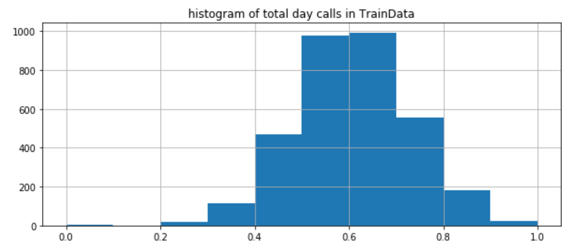


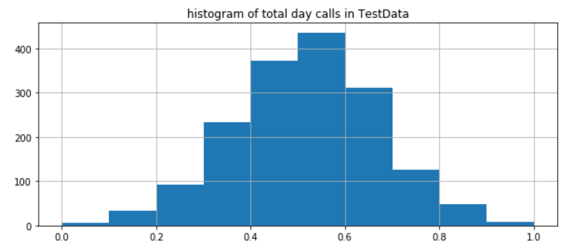

histogram of number vmail messages in TrainData

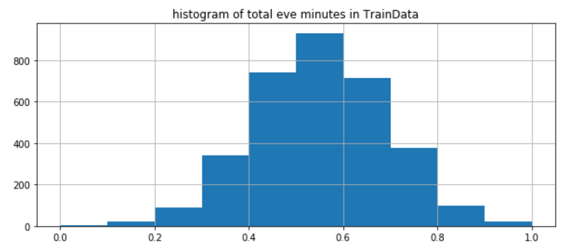

histogram of number vmail messages in TestData

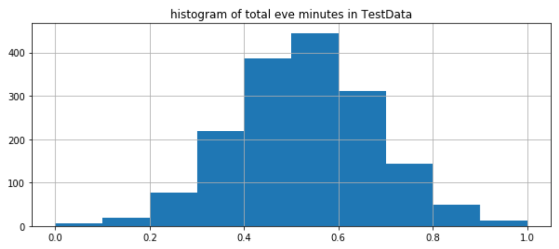

histogram of total day minutes in TrainData

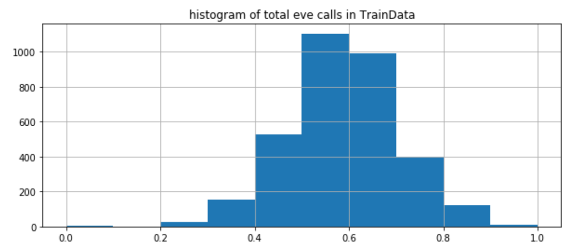

histogram of total day minutes in TestData

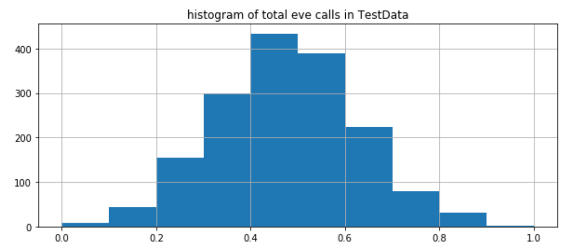

histogram of total day calls in TrainData

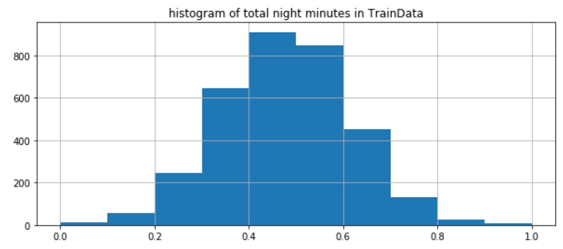

histogram of total day calls in TestData

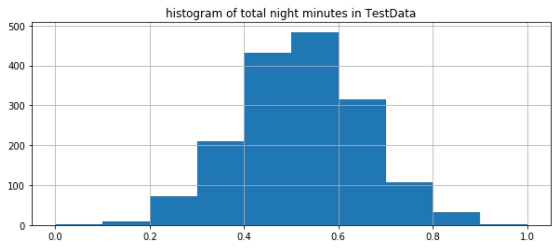

histogram of total eve minutes in TrainData

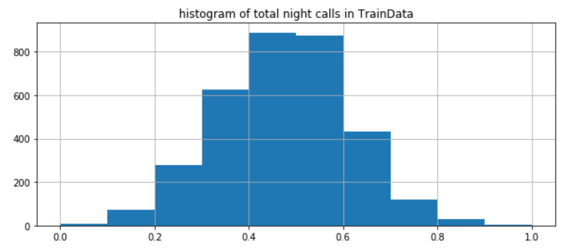

histogram of total eve minutes in TestData

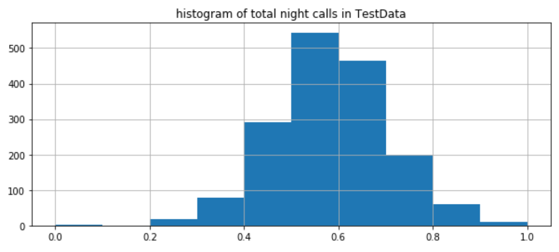

histogram of total eve calls in TrainData

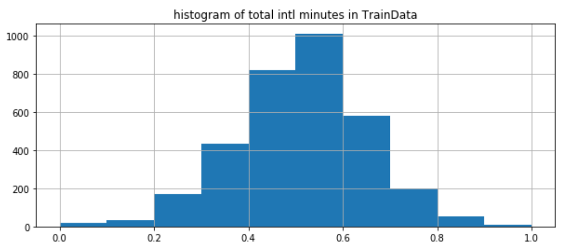

histogram of total eve calls in TestData

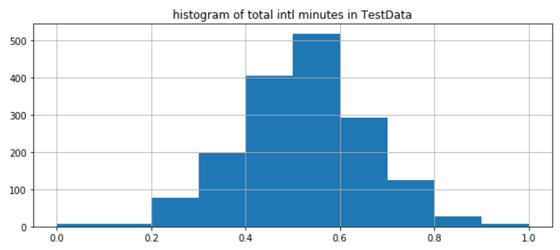

histogram of total night minutes in TrainData

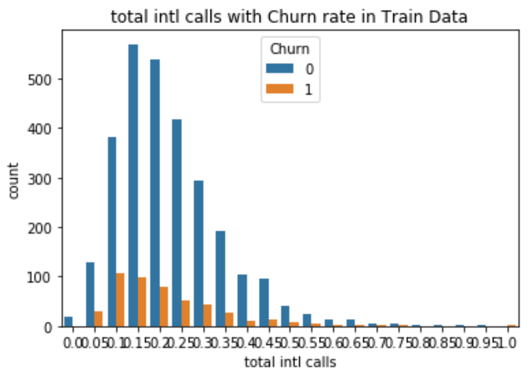

histogram of total night minutes in TestData

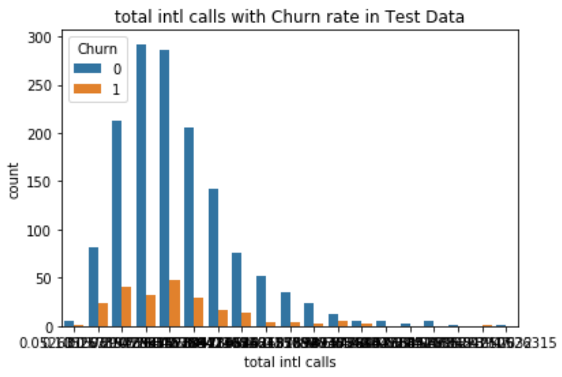

histogram of total night calls in TrainData

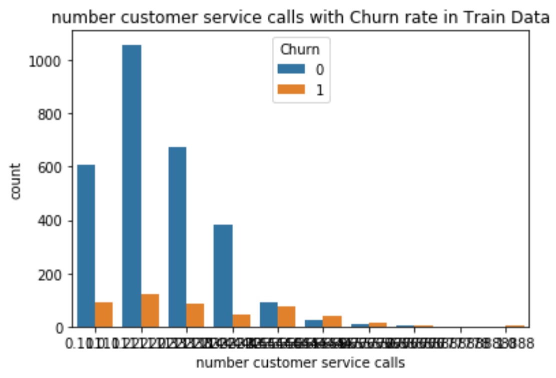

histogram of total night calls in TestData

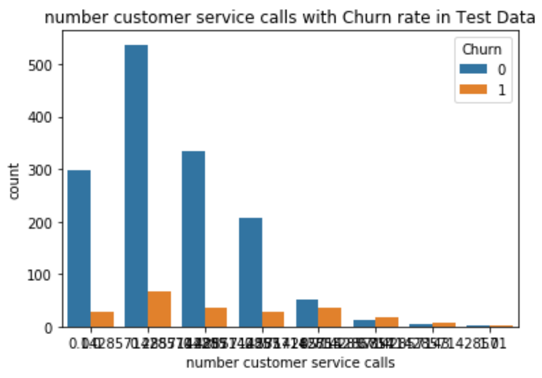




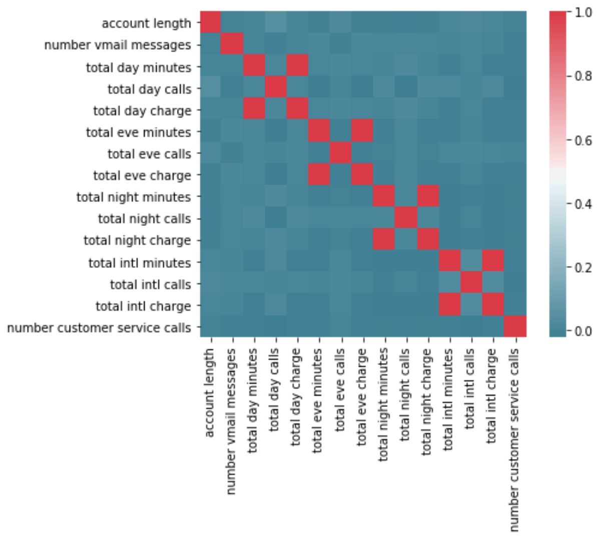








4.2.3 Correlation plot


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