2/2/2021

BUSINESS INTELLIGENCE AND ANALYTICS – PROJECT REPORT

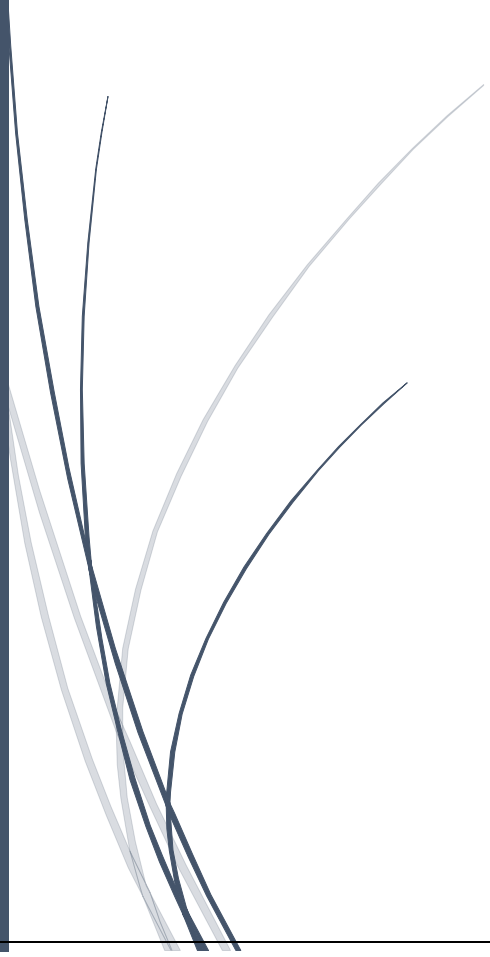


Table of Contents

1. INTRODUCTION:

- This project aims to predict the customer churn value (decision where a customer decides to terminate the service with the current telecom provider) using predictive models
- Various predictive model algorithms were developed and their respective accuracy was compared with one another. The model with the highest accuracy was selected as the optimum model and hence the churn values were predicted using the selected model
- The project followed the following steps in sequence:
    - Data Importing and Pre Processing
    - Dummy variable creation and Data Partitioning
    - Model Creation: Logistic Regression, Random Forest and kNN Classification
    - Model Selection
    - Calculating the Top Decile Lift and Gini Coefficient for the model
    - Predicting the Churn values for the Current score and Future score datasets

2. DATA IMPORTING AND PRE-PROCESSING:

Data Import:

- Calibration dataset is imported into R using ***read.table*** function with separator as ',' as the file is of type .csv. The dataset has blank values in many observations across many columns. Hence these blank values are converted as NA while importing the data. Imported dataset is named as **data**
- The calibration dataset contains 1,00,000 observations and 173 columns (variables)

```
> # 1. Importing Data
> data <- read.table("calibration.csv",sep = ',',header = TRUE,na.strings = c("",NA)) # Replacing blanks with NA
> nrow(data)
[1] 100000
> ncol(data)
[1] 173
```

Variable selection based on judgment:

- From the given calibration dataset, we can see than the 173 variables are not completely unique and many of the variables have similar information.
- Hence based on judgement, we have selected 85 variables out of the given 173. A new csv file 'SelCols.csv' is imported using read table function which contains the list of the selected 85 variables. A new DataFrame **data_v1** is created from the original DataFrame data with the selected 85 variables

```
> # 2. Preprocessing Data
> selCols <- read.table("selCols.csv",sep=',',header = TRUE) # Selecting columns based on judgment and analysis o
f dataset
> nrow(selCols)
[1] 85
> View(selCols)
> data_v1 <- data[names(data) %in% selCols$Col.Names] # DataFrame with selected columns
> nrow(data_v1)
[1] 100000
> ncol(data_v1)
[1] 85
```

Imputing Missing values:

- From the function **colSums(is.na())** we can see that out of the 85 variables, 33 have missing values in the form of NA. These missing values need to be handled

```
> colSums(is.na(data_v1[,colSums(is.na(data_v1)) > 0]))
       mou_Mean      totmrc_Mean          da_Mean      vceovr_Mean      datovr_Mean       mou_Range
            357              357              357              357              357              357
   totmrc_Range         da_Range     vceovr_Range     datovr_Range      change_mou          avg6mou
            357              357              357              357              891             2839
        REF_QTY          tot_ret         tot_acpt prizm_social_one             area         dualband
          95545            96017            96017             7388               40                1
     refurb_new        hnd_price           phones           models       hnd_webcap         dwlltype
              1              847                1                1            10189            31909
        marital             age1           adults           income        forgntvl          ethnic
           1732             1732            23019            25436             1732             1732
      creditcd          retdays          eqpdays
           1732            96017                1
```

- We are using a logic that only those rows and columns with less than 10% missing values are selected. The **which()** function helps to identify the row and column numbers with the condition and a new DataFrame data_**v2** is created accordingly.

```
> rowsToSelect <- which(rowMeans(is.na(data_v1))<0.1) # Selecting rows with < 10% missing values
> colsToSelect <- which(colMeans(is.na(data_v1))<0.1) # Selecting cols with < 10% missing values
> data_v2 <- data_v1[rowsToSelect,colsToSelect]
> nrow(data_v2)
[1] 97557
> ncol(data_v2)
[1] 77
```

- On running the **colSums(is.na())** function again, we can see that we still have 10 variables with missing values. The observations in these variables need to be imputed before proceeding with model creation.

```
> colSums(is.na(data_v2[,colSums(is.na(data_v2)) > 0]))
     change_mou          avg6mou prizm_social_one             area        hnd_price          marital
            497             2557             6323               37              795                2
           age1         forgntvl          ethnic         creditcd
              2                2                2                2
```

- The 10 variables are saved in a vector named **colToImp.** This vector is passed to a **for loop** function where the observations are imputed based on their class.
    - For missing observations of class ‘numeric’ the values are imputed using the mean of the other observations in that particular variables
    - For missing observations of class ‘character’ or ‘factor’ the values are imputed using the mode of the variable (the value that occurs the most in that particular variable)

```
> colToImp <- names(data_v2[colSums(is.na(data_v2))>0]) # Column names to impute
> colToImp
 [1] "change_mou"       "avg6mou"          "prizm_social_one" "area"             "hnd_price"
 [6] "marital"          "age1"             "forgntvl"         "ethnic"           "creditcd"
> # Imputing missing values
> for (i in colToImp){
+     if (class(data_v2[,i]) == 'character' || class(data_v2[,i]) ==  'factor'){
+         data_v2[,i][is.na(data_v2[,i])] = names(which.max(summary(as.factor(data_v2[,i]))))
+     }
+     else{
+         mean_val = mean(data_v2[,i],na.rm = TRUE)
+         data_v2[,i][is.na(data_v2[,i])] = mean_val
+     }
+ }
```

- In one particular variable age1, we can see that more than 27000 observations have value as 0. These are not missing values but a value of 0 does not make sense for the variable age. As these observations make up nearly one third of the total observations in the dataset, we are dropping this variable.

```
> data_v3=data_v2[!names(data_v2) %in% "age1"] # age1 has 28% records with value as 0. Hence removing
> colSums(is.na(data_v3[,colSums(is.na(data_v3)) > 0]))
numeric(0)
```

- The resulting Data Frame **data_v3** does not have any variable with missing values

3. DUMMY VARIABLE CREATION AND DATA PARTITIONING:

- Library **caret** is used to create dummy variables in the dataset and create data partitions into training and validation
- **dummyVars()** functions with fullRank is used for creating the dummy variables. A new Data Frame **data_dummy** is created. After dummy creation, the dataset contains 117 columns in total
- For data partitioning a seed value of 42 is used. 70% of the records are maintained in training partition whereas the remaining 30% are allocated to validation.

```
> # 3. Creating Dummies and Data Partition using `caret` package
> library(caret)
Loading required package: lattice
Loading required package: ggplot2
> dummy <- dummyVars("~.",data_v3,fullRank = T)
> data_dummy <- data.frame(predict(dummy,newdata=data_v3))
> ncol(data_dummy)
[1] 117
> data_dummy$churn = as.factor(data_dummy$churn)
> set.seed(42) # Roll No.
> row = createDataPartition(data_dummy$churn,p=0.7,list=F) #70% records in Training, 30% in Validation
> dtrain_lr = data_dummy[row,]
> dval_lr = data_dummy[-row,]
```

4. MODEL CREATION – LOGISTIC REGRESSION:

- The first model created is based on Logistic Regression algorithm. Before proceeding with the model, the dataset is checked for Multi Collinearity using library **car** and **vif()** function
- Out of the 117 variables, we can see that 42 variables have the Variance Inflation Factor more than 5. This signifies that the variables and the dataset has moderate to high Multi Collinearity. Hence Principal Component Analysis (PCA) needs to be performed.

```
> vif_value <- vif(lm(churn~.,data=data_v3))
> vif_value[vif_value > 5]
 [1]  142.303577    5.133829    9.527584    5.289588    9.721155    5.160946    9.659444    9.636524
 [9]   67.680332  673.425435    6.020469   67.195913  596.490254    9.421372    5.147321    8.383673
[17]    7.373582    7.726190   18.627220  168.713276    6.031062   19.355722  151.125697 2152.762779
[25] 2134.447656    9.383657  150.540807   14.997254    5.168602    5.764514   18.000000   16.000000
[33]   11.929106    8.226806   25.950442    8.197311   24.423150   12.988967   12.293319   46.397875
[41]   46.200083   12.269507
```

- From the rotation matrix we can see that PCA has resulted in 76 principal components. These are further used to create the logistic regression model.

```
> # Principal Component Analysis using caret package. Data centered and scaled
> rot = preProcess(dtrain_lr[,!names(dtrain_lr) %in% 'churn'],method=c('scale','center','pca'))
```

```
> ncol(rot$rotation)
[1] 76
```

- Logistic Regression model is created using glm() function with family specified as binomial. The results of the model are stored in the variable lr_v1.
- From the regression summary we can see that only 47 variables are significant as given by the p-value of Wald's Z Test (p-value of less than 0.05)
- Next iteration of the logistic regression model is created by removing these variables from the Data Frame and running the model again.
- Second iteration of the Logistic Regression model is created using glm() function with family specified as binomial. The results of the model are stored in the variable lr_v2

```
Call:
glm(formula = churn ~ ., family = "binomial", data = dtrain_pca_v2)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5731  -1.1451  -0.7541   1.1491   2.3561

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.021634   0.007795  -2.775 0.005515 **
PC1         -0.020136   0.001954 -10.304  < 2e-16 ***
PC4          0.067491   0.005086  13.269  < 2e-16 ***
PC5         -0.062224   0.005192 -11.984  < 2e-16 ***
PC6         -0.068514   0.005564 -12.314  < 2e-16 ***
PC7         -0.021369   0.005794  -3.688 0.000226 ***
PC8          0.060343   0.005816  10.376  < 2e-16 ***
PC10        -0.092379   0.005945 -15.539  < 2e-16 ***
PC11         0.074836   0.006238  11.996  < 2e-16 ***
PC12         0.072948   0.006266  11.642  < 2e-16 ***
PC14         0.045793   0.006235   7.345 2.06e-13 ***
PC15         0.059154   0.006365   9.293  < 2e-16 ***
PC16        -0.034769   0.006592  -5.274 1.33e-07 ***
PC17        -0.049610   0.006610  -7.505 6.13e-14 ***
PC18         0.013541   0.006586   2.056 0.039780 *
PC19        -0.035316   0.006728  -5.249 1.53e-07 ***
PC21         0.065087   0.006938   9.381  < 2e-16 ***
PC24         0.031107   0.007186   4.329 1.50e-05 ***
PC25        -0.014450   0.007104  -2.034 0.041951 *
PC26         0.075467   0.007233  10.434  < 2e-16 ***
PC27        -0.018945   0.007314  -2.590 0.009590 **
PC29         0.031844   0.007409   4.298 1.72e-05 ***
PC30         0.041463   0.007352   5.640 1.70e-08 ***
PC31         0.047510   0.007490   6.343 2.25e-10 ***
PC33        -0.042356   0.007471  -5.670 1.43e-08 ***
PC34        -0.028134   0.007520  -3.741 0.000183 ***
PC35         0.016118   0.007673   2.101 0.035671 *
PC40         0.061233   0.007678   7.975 1.52e-15 ***
PC41        -0.019829   0.007704  -2.574 0.010059 *
PC42        -0.018582   0.007712  -2.409 0.015977 *
PC43         0.036441   0.007751   4.701 2.59e-06 ***
PC44         0.019528   0.007753   2.519 0.011778 *
PC47        -0.022190   0.007928  -2.799 0.005127 **
PC52         0.020659   0.008150   2.535 0.011245 *
PC55         0.053281   0.007986   6.672 2.52e-11 ***
PC57        -0.081108   0.008111 -10.000  < 2e-16 ***
PC58         0.016898   0.008163   2.070 0.038440 *
PC60         0.020151   0.008836   2.280 0.022581 *
PC61         0.021775   0.008611   2.529 0.011450 *
PC64         0.068112   0.008835   7.709 1.27e-14 ***
PC66        -0.024880   0.009859  -2.524 0.011614 *
PC67         0.043792   0.009788   4.474 7.68e-06 ***
PC68         0.037896   0.010337   3.666 0.000246 ***
PC69         0.024883   0.010592   2.349 0.018805 *
PC71        -0.121138   0.010980 -11.032  < 2e-16 ***
PC72        -0.063269   0.011020  -5.741 9.41e-09 ***
PC73        -0.072113   0.011293  -6.385 1.71e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 94664  on 68290  degrees of freedom
Residual deviance: 92207  on 68244  degrees of freedom
AIC: 92301

Number of Fisher Scoring iterations: 4
```

- From the regression summary, we can see that model is capable of explaining only 2457 of the variance. Residual deviance is very high with a value of 92207. This suggests that the predictive power of the model is low

```
    Null deviance: 94664  on 68290  degrees of freedom
Residual deviance: 92207  on 68244  degrees of freedom
AIC: 92301
```

Predicting the Outcome variable in Validation Partition by fitting the Logistic Regression Model:

- The validation partition is transformed for getting the principal components using the rotation matrix. The second iteration of logistic regression had training dataset with 46 predictor variables. Hence the validation dataset also need to be changed accordingly to contain only these 46 variables.

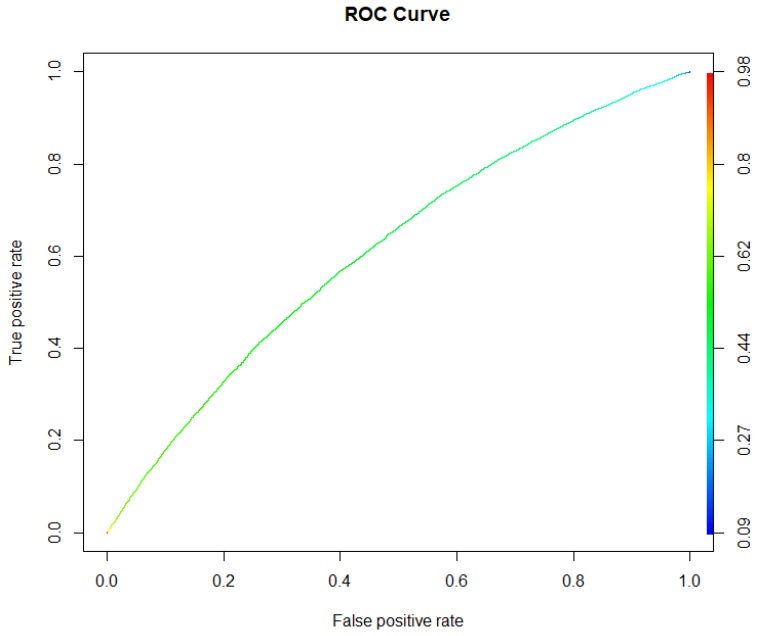
```
> # Predicting Outcome in Validation dataset
> # PCA transformation of Validation dataset using Rotation matrix
> dval_pca = predict(rot,dval_lr[,!names(dval_lr) %in% 'churn'])
> dval_pca <- dval_pca[names(dval_pca) %in% lr_v1_col$Selected.PC]
> names(dval_pca)
 [1] "PC1"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "PC10" "PC11" "PC12" "PC14" "PC15" "PC16" "PC17" "PC18" "PC19"
[16] "PC21" "PC24" "PC25" "PC26" "PC27" "PC29" "PC30" "PC31" "PC33" "PC34" "PC35" "PC40" "PC41" "PC42" "PC43"
[31] "PC44" "PC47" "PC52" "PC55" "PC57" "PC58" "PC60" "PC61" "PC64" "PC66" "PC67" "PC68" "PC69" "PC71" "PC72"
[46] "PC73"
```

- Churn values are predicted in the validation partition using the second iteration of the logistic regression model. The resulting churn outcomes are stored in **lr_val** variable

```
> lr_val <- predict(lr_v2,newdata=dval_pca, type="response") # Prediction on validation Dataset
> head(lr_val)
        3         5        11        21        22        23
0.6456853 0.3553538 0.3966472 0.4319228 0.7073136 0.6281623
```

Generating ROC Curves for determining the Cut-Off probabilities

- Library **ROCR** is used to generate ROC curves and to determine the cut off probabilities.



- Based on the ROC curve, cut off probability is selected as 0.5. Confusion matrix is generated at this cut off probability.

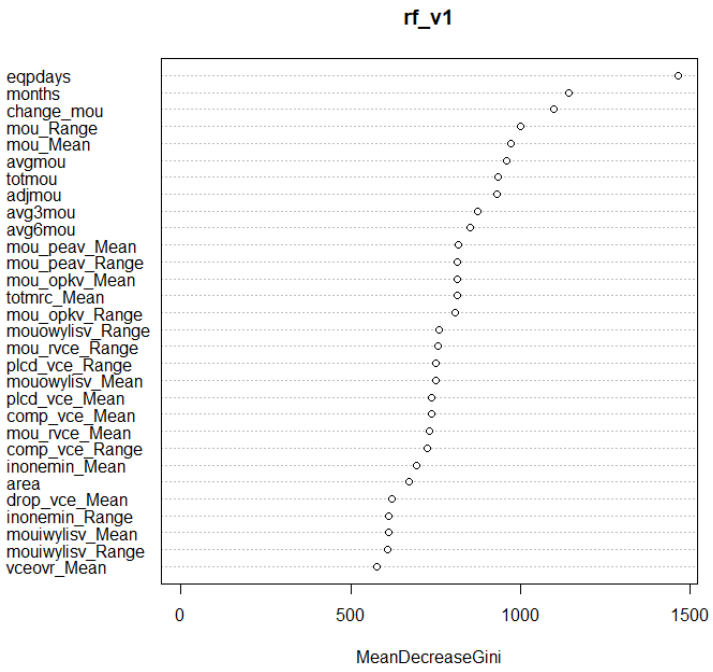
```
> predcton<-ifelse(lr_val>0.5,1,0) #Fixing cutoff probability as 0.5 based on ROC curve
> print("At cutoff=0.50")
[1] "At cutoff=0.50"
> tab_v1 <- table(dval_lr$churn,predcton)
> tab_v1
   predcton
       0    1
  0 8700 6081
  1 6131 8354
> acc_v1=(tab_v1[1,1]+tab_v1[2,2])/nrow(dval_lr) # Accuracy of Model: TP + TN / Total rows
> print('Accuracy at 0.5 Cutoff probability');acc_v1  # 58%
[1] "Accuracy at 0.5 Cutoff probability"
[1] 0.582724
```

- Accuracy is calculated from the Confusion Matrix and we can see that the model is capable of providing an accuracy of 58.27%
- At other cut off probabilities (more than 0.5 and less than 0.5), the model is providing an accuracy of less than 58.27%. Hence the maximum accuracy of the model is achieved at a cut off probability of 0.5

5. MODEL CREATION – RANDOM FOREST:

- Training and Validation partition are created without the dummies (seed value of 42 and 70% observations in training set)
- Random Forest model is created using library **randomForest** and function **randomForest()**
- The model is saved to the variable **rf_v1.**

```
RANDOM FOREST MODEL
rf_v1 <- randomForest(churn~.,dtrain_rf)
```



Predicting the Outcome variable in Validation Partition by fitting the Random Forest Model:

- The Random Forest model is fitted to the validation partition and Churn values are predicted. The output gives binary values of 0 or 1 determining the churn value

```
Predicting Outcome in Validation dataset
rf_val <- predict(rf_v1,newdata=dval_rf)
>   head(rf_val)
 3  5 11 21 22 23
 1  1  0  0  0  0
Levels: 0 1
```

- Confusion matrix is created based on the actual churn values and churn values predicted from the Random Forest model.
- Accuracy is calculated from the Confusion Matrix and we can see that the model is capable of providing an accuracy of 63.11%

```
> rf_tab = table(dval_rf$churn,rf_val)
> rf_tab
   rf_val
       0    1
  0 9201 5580
  1 5214 9271
> rf_acc = (rf_tab[1,1]+rf_tab[2,2])/nrow(dval_rf)
> print('Accuracy of model through Random Forest Algorithm'); rf_acc # 62%
[1] "Accuracy of model through Random Forest Algorithm"
[1] 0.6311761
```

- Hence we can conclude that at 63.11%, Random Forest algorithm provides an accuracy which is greater than that of Logistic Regression model.

6. MODEL CREATION – k NEAREST NEIGHBOURS CLASSIFICATION:

- Training and Validation partition are created with the dummy variables (seed value of 42 and 70% observations in training set)
- K Nearest Neighbors Classification model is created using library **class** and function **knn().** Different values of k are given in order to find the model with maximum accuracy.
- Initial model is created with k value of 101
    - The model is saved to the variable **knn_v2.**

```
> # kNN MODEL
> library(class)
> print("k=101")
[1] "k=101"
> print(table(dval_knn[,names(dval_knn) %in% 'churn'],knn_v2))
   knn_v2
       0    1
  0 8795 5986
  1 6879 7606
> tab_kNN_2=table(dval_knn[,names(dval_knn) %in% 'churn'],knn_v2)
> acc_knn_2=(tab_kNN_2[1,1]+tab_kNN_2[2,2])/nrow(dval_knn)
> acc_knn_2 # 56%
[1] 0.5604114
```

    - This model gives an accuracy of 56.04%
- Next iteration of the model is created with k value of 251
    - The model is saved to the variable **knn_v3.**

```
> knn_v3 <- knn(dtrain_knn_scaled,dval_knn_scaled,dtrain_knn[,names(dtrain_knn) %in% 'churn'],k=251)
> print("k=251")
[1] "k=251"
> print(table(dval_knn[,names(dval_knn) %in% 'churn'],knn_v3))
   knn_v3
      0    1
  0 9008 5773
  1 6877 7608
> tab_kNN_3=table(dval_knn[,names(dval_knn) %in% 'churn'],knn_v3)
> acc_knn_3=(tab_kNN_3[1,1]+tab_kNN_3[2,2])/nrow(dval_knn)
> acc_knn_3
[1] 0.5677578
```

- This model with k value of 251 gives an accuracy of 56.77%

7. SELECTING THE MODEL – BASED ON ACCURACY:

- From the 3 models, we can see that Random Forest model provides the highest accuracy of 63.11 %
- Hence the Random Forest model is chosen for predicting the churn values in Current and Future score datasets

8. CALCULATING TOP DECILE LIFT AND GINI COEFFICIENT FOR THE MODEL:

Since the churn was available only for the calibration dataset, the Top Decile Lift and Gini Coefficient calculations were performed only for the dataset.

- Top Decile Lift:
  Library 'lift' was employed for the calculation and the method used was TopDecileLift.
  Top Decile Lift of 0.99 was obtained for the random forest model

```
> ## Calculating the TopDecileLift and Gini Coefficient
> library(lift)
> TopDecileLift(rf_val,dval_rf['churn'])
[1] 0.99
```

- Gini Coefficient
  Library 'MLmetrics' was used for the calculation of the Gini Coefficient and the method Gini was used. Gini coefficient of 0.25 was obtained for the random forest model. Implying that there is a separation by the methods lift curve from the random, hence there is an improvement in the prediction.

```
> library(MLmetrics)

Attaching package: 'MLmetrics'

The following objects are masked from 'package:caret':

    MAE, RMSE

The following object is masked from 'package:base':

    Recall

Warning message:
package 'MLmetrics' was built under R version 3.6.3
> Gini(y_pred= rf_val,y_true =dval_rf[,'churn'])
[1] 0.256742
```

9. PREDICTING CHURN VALUE – CURRENT SCORE, FUTURE SCORE DATA:

- Based on the models and the accuracy we have selected Random Forest as the optimum model for predicting the churn values. Hence the Random Forest model is fitted to the Current and Future score datasets to predict Churn values
- Importing Data:
    - Current and Future score datasets are imported using the function **read.table**
    - The datasets are stores in Data Frame variable **data_current and data_future** respectively
- Pre Processing Data:
    - The columns in the new Data Frame are edited to have only those variables present in the Random Forest model. The unwanted columns are removed and the new Data Frames are stored in variables **data_c1 and data_f1**
    - **colSums(is.na())** function is used to determine the columns with missing values and hence to impute them subsequently.
        - The result shows that the data frames **data_c1 and data_f1** have 19 and 18 missing values respectively. Hence these needs to be imputed.

```
> colSums(is.na(data_c1[,colSums(is.na(data_c1)) > 0]))
       mou_Mean      totmrc_Mean          da_Mean      vceovr_Mean      datovr_Mean       mou_Range
             41               41               41               41               41              41
   totmrc_Range         da_Range     vceovr_Range     datovr_Range       change_mou          avg6mou
             41               41               41               41              158            1744
prizm_social_one             area       hnd_price          marital         forgntvl          ethnic
           3547               16              522              951              951             951
       creditcd
            951
> colSums(is.na(data_f1[,colSums(is.na(data_f1)) > 0]))
       mou_Mean      totmrc_Mean          da_Mean      vceovr_Mean      datovr_Mean       mou_Range
            114              114              114              114              114             114
   totmrc_Range         da_Range     vceovr_Range     datovr_Range       change_mou          avg6mou
            114              114              114              114              490            5674
prizm_social_one       hnd_price          marital         forgntvl          ethnic        creditcd
           7662              436             1905             1905             1905            1905
```

- The variables to be imputed in data_c1 Data Frame are saved in vector named **colToImp_c**. Similarly, variables to be imputed in data_f1 Data Frame are saved in vector named **colToImp_f**. Both these vectors are passed to a **for loop** function where the observations are imputed based on their class.
  - For missing observations of class 'numeric' the values are imputed using the mean of the other observations in that particular variables
  - For missing observations of class 'character' or 'factor' the values are imputed using the mode of the variable (the value that occurs the most in that particular variable)

```
> colSums(is.na(data_c1[,colSums(is.na(data_c1)) > 0]))
numeric(0)
> colSums(is.na(data_f1[,colSums(is.na(data_f1)) > 0]))
numeric(0)
```

- Again, these datasets also have the age1 variable with more than 28% records with value as 0. Hence this variable is dropped from the datasets.

```
> nrow(data_c2) # Rows in Current Score dataset
[1] 51306
> ncol(data_c2) # Cols in Current Score dataset
[1] 75
> nrow(data_f2) # Rows in Future Score dataset
[1] 100462
> ncol(data_f2) # Cols in Future Score dataset
[1] 75
```

- Predicting the Churn value (outcome) based on the Random Forest Algorithm
  - The imputed data from the current score dataset was fed into the random forest model created, to predict the churn outcome

```
> # Predicting Outcome for both  dataset using Random Forest Model
> rf_val_c <- predict(rf_v1,newdata=data_c2)
> rf_val_c
```

  - Similarly, the future score dataset was fed into the model for prediction. Note: The predict code returned an error that the type of predictors does not match, although the predictors and the type were similar to the current score. Hence, a workaround was used where the first row from the training dataset was added and then removed to resolve the error (solution obtained from stack overflow)

```
> rf_val_f <- predict(rf_v1,newdata=data_f2)
Error in predict.randomForest(rf_v1, newdata = data_f2) :
  Type of predictors in new data do not match that of the training data.
> #Getting Error as type of predictors donot match. Hence running a workaround obtained from stackoverflow
> xtest <- rbind(dtrain_rf[1,!names(dtrain_rf)=='churn'] , data_f2)
> xtest <- xtest[-1,]
> rf_val_f <- predict(rf_v1,newdata=xtest)
> rf_val_f
```

- The predicted churn values in current and future score datasets are written to a new csv file.
  - Using the **cbind()** function, the original datasets (current score and future score) are combined with the predicted churn values by fitting the Random Forest model
  - **write.csv()** function is used to write the churn values into the csv file

10. CONCLUSION:

- Thus, using the Random Forest model we were able to predict the Churn values for the current score and future score datasets
- The accuracy of the Random Forest model was 63.11%
- The predicted churn values were written into new csv files with which the telecom company can make targeted marketing efforts towards the customers who are expected to churn.