Mail-Order Company

Devendra Desale (A0134465E), Abhishek Sharma (A0135870A), Ning Chao (A0134563H)

M.Tech KE27 | DMMM Project | March 2015

About the company:

A Mail-Order company, also known as a MOC, in USA has a product they would like to promote. They consider a campaign offering this product for sale, directed at a given customer base. Normally, about 1% of the customer base will be "responders", customers who will purchase the product if it is offered to them. A mailing to a million randomly-chosen customers will therefore generate about ten thousand sales. Business analytics techniques enable more efficient marketting, by identifying which customers are most likely to respond the campaign. If the response can be raised from 1% to, say, 1.5% of the customers contacted, 10,000 sales could be achieved with only 666,667 mailings therby reducing the cost of mailing by one third.

Business Objectives:

- One business objective is to find which customers to promote the product to and hence build a model to predict who would be a good candidate to purchase this.
- Another objective is to explore the data to see if any interesting and useful patterns can be discovered which may help the company deterimine valuable customers.

Data-Set Description:

The file `project.csv` is the dataset with various attributes and related observations. The data was extracted from a much larger set with a response rate of about 1%. In this the objective variable is a response varialbe indicating whether or not a consumer responded to a direct mail campaign for a specific product. "True" or "response" is 1 whereas "False" or "non-response" is 0.All 1079 responders were used, together with 1079 randomly chosen non-responders, for a total of 2158 cases. There are 200 other expanatory variables in the dataset.

`v17,v141 and v200` are indicators for gender "male", "female" or "unkown", respectively.

`v1-v24, v138-v140 and v142-v144` are recency, frequency and monetary type of data for specific accounts.

`v25-v136` are the census variables.

`v145-v199` are demographic "taxfilers" variables.

Along with `project.csv`, following files with short descriptions are also provided:

`Census Variables - Group a.doc`

`Census Variables - Group b.doc`

`Taxfiler.doc`

`Variables.doc`

Some variable descriptions are in `variable.doc`. Some of the product-specific variables have been blinded. "p##" means product, "rcy" means recency, "trans" means number of transactions, "spend" means dollars spending. For example "p01rcy"" means product 1 recency. Note: Zero means the account has never bought the product. The greater the number, the more recent it is. The census and texfiler variables are summary statistics for the enumeration are in which the account holder's address is loaded. They generally give total or average numbers of individuals or families or dollars in the categories indicated. `Txfilers.doc` contains the taxfiler variable descriptions.

Data Mining Process

In view of data mining, the analysis is made from a few iterative steps. The CRISP is a standard process of data mining. DM process includes steps like **Business Understanding** which means understanding the business goals and to plan for the project accordingly; **Data Exploration** which means to gather the data and explore the data by viewing the summary statistics and understanding the structure and its quality, descriptive etc analysis etc; **Data Preparation** phase includes inghts like which data need to select, transformed, and which data to be cleaned; **Modelling** includes steps to do visualizations, making decision trees, neural network ec models as per the project; **Evaluation** stage refers to the evaluation of the models based on the evaluating parameters like ROC, Risk values etc. and to analyse for further iteration of the models with different conditions; **Deployement** is the last stage of the process which lets the model to operate with business data and to analyse the model with operationg conditions.

Data Exploration:

To read `project.csv` file:

```
dmm_data<-read.csv("DMMM_project/data/project.csv")
```

In this phase after analysis the data we will be going further with dataset improvement which will lead us to better prediction.

Renaming The Dataset

For ease of understanding and to make the dataset representable we rename the variables from "vX" to its variable name as mentioned in the provided `variables.doc`. We read the names of variables from `variables.csv` which have been compiled as a short summary of all the provided documents.

```
variables <- read.csv('DMMM_project/data/variables.csv',header = TRUE)
colnames(dmm_data) <-variables[,2][1:201]
```

Data Quality Assesment:

Structure of dataset

To see the underlying structure of the dataset, `str()` is used. Hence, in the dataset, we have 201 variables and 2158 observation.

```
str(dmm_data[1:10])
```

```
## 'data.frame':    2158 obs. of  10 variables:
##  $ Objective : int  1 1 1 1 0 1 1 0 1 1 ...
##  $ p01rcy    : num  0 0.996 0.974 0.97 0 ...
##  $ p02rcy    : num  0 0 0 0 0 ...
##  $ p03rcy    : num  0 0 0.996 0 0.987 ...
##  $ p04rcy    : num  0 0.939 0 0 0 ...
##  $ totalspend: int  4854 5504 11178 2628 3712 5213 5483 7926 6743 6449 ...
##  $ p05spend  : num  0 0 0 0 0 ...
##  $ p05trans  : num  0 0 0 0 0 ...
##  $ p06rcy    : num  0 0.991 0 0 0 ...
##  $ p07rcy    : num  0 0 0 0 0.97 ...
```

We have analysed, almost all data are continuous expect 5 records viz; v137, v139, v140, v141 and v200 (gender1, lowincome, highincome, gender2, gender, respectively) which have binary values. We use `summary()` function to have insight of data statistics as-well.

Are there any Missing Values in the data set?

To check for any missing values, `complete.cases()` when used will return `TRUE` if missing values are found. To get the variables with any rows with missing values:

```
dmm_data[!complete.cases(dmm_data),]
```

```
##   [1] Objective     p01rcy        p02rcy        p03rcy        p04rcy
##   [6] totalspend    p05spend      p05trans      p06rcy        p07rcy
##  [11] p08rcy        p09rcy        p09tenure     p11rcy        p12rcy
##  [16] p13rcy        p14rcy        p14tenure     p15rcy        p16spend
##  [21] p16rcy        p16tenure     p16trans      p17rcy        totaltrans
##  [26] cffempar      cfhuswife     cflonepar     cftotmar      cfwchcom
##  [31] fem40to44     fp1child      fp2child      hh2fam        hh6ppers
##  [36] mtenglish     mtfrench      mtmultlin     mtnengnon     mtsingres
##  [41] mtspanish     mttagalog     nfamrel       cwmpaid       dw46to60
##  [46] dw86to91      dwmaint       dwmajor       dwminor       dwperroom
##  [51] etbritish     etenglish     etfrench      etmulti       etsingle
##  [56] fi20to35      fi50plus      fiinca        fiincm        fiu20
##  [61] fps           fpshealth     fpshuman      fsllabf       fslonepar
##  [66] fslplabf      fsmlabf       hi20to35      hi50plus      hiinca
##  [71] hiincm        hiincs        hiu20         hlenglish     hlfrench
##  [76] hlnonoff      improvres     imuk          incgovp       ineflow
##  [81] ineflowp      inf30plus     inf7to15      infinca       infincm
##  [86] infincs       inhhlow       inhhlowp      inm15to30     inm30plus
##  [91] inm7to15      inmfemina     inminca       inmincm       inmincs
##  [96] knenglish     knfren        lfmaempl      lfmaunemr     lfmtempl
## [101] lfmtunemp     lfmtunemr     lftaempl      lfttempl      lfttunemr
## [106] moy1intep     moy5intep     moy5intrn     moy5mov       moy5non
## [111] mps           mpscomm       mpseng        mpshealth     mpssocial
## [116] ndtallind     ndtbusser     ndtgovser     ocffabric     ocfmanage
## [121] ocfteach      ocmmanage     ocmscieng     pwmcsd        pwmpop
## [126] pwmusual      rlanglic      rlcathol      rlprotest     rlrcathol
## [131] rlunited      sl9to13nc     slg9          slnunivc      slunivdeg
## [136] slunivnc      slunivnd      gender1       first         lowincome
## [141] highincome    gender2       productcount  productcount6 tenure
## [146] tf100         tf101         tf102         tf103         tf107
## [151] tf108         tf122         tf124         tf128         tf129
## [156] tf27          tf28          tf29          tf31          tf32
## [161] tf33          tf34          tf35          tf36          tf37
## [166] tf38          tf39          tf42          tf46          tf47
## [171] tf49          tf50          tf51          tf52          tf55
## [176] tf56          tf57          tf58          tf62          tf65
## [181] tf68          tf70          tf71          tf72          tf73
## [186] tf74          tf75          tf76          tf77          tf80
## [191] tf88          tf89          tf90          tf91          tf92
## [196] tf93          tf94          tf95          tf96          tf99
## [201] gender
## <0 rows> (or 0-length row.names)
```

Running so, we get `0` rows missing. Hence we could conclude that there are no missing values.

Data Profiling:

Data Segmentation is done based on the quality of data. We have created histograms, dodge histogram, boxplots of the availabe data variables with respect to objective and analyse its quality. For instance, `Histgram` of `p02rcy` helps us in inferring that persons who have purchased product #02 recently are likely to respond more. But after seeing its `Dodge-Histogram` we can say that the count of those persons are not so significant that we could consider accepting this fact. Same goes for `p04rcy` variable. But for the `p03rcy` we explore the fact that the people purchasing this product are more responding and are in high count. So this is a significant inefernce. For `totalspend` variable, persons spending below 4000$ are responding to the mails. And this is can be considered significant by analysing its `Dodge-Histogram` in which the count of those persons are high.

```r
require(ggplot2)
```

```
## Loading required package: ggplot2
```

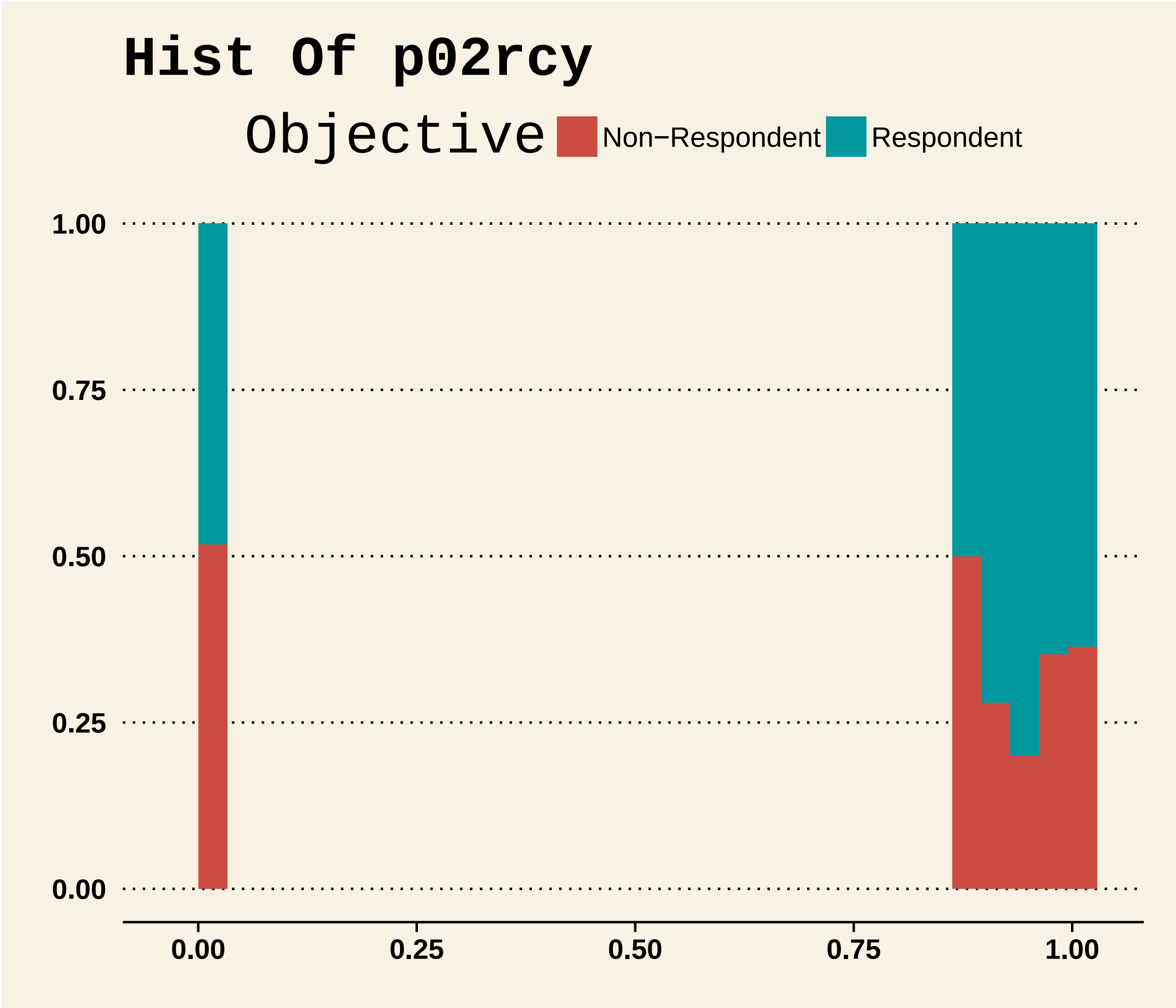
```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

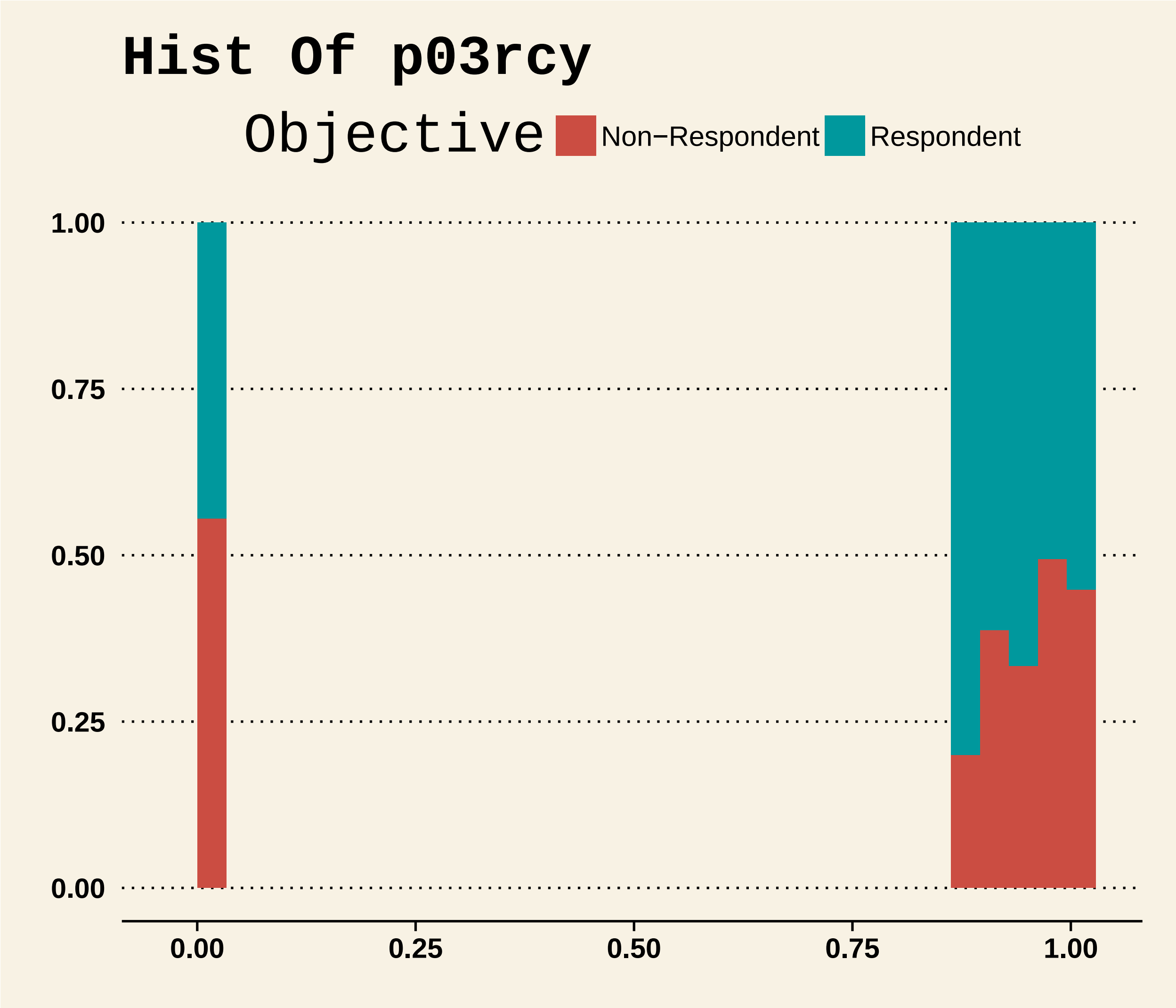
```r
require(ggthemes)
```

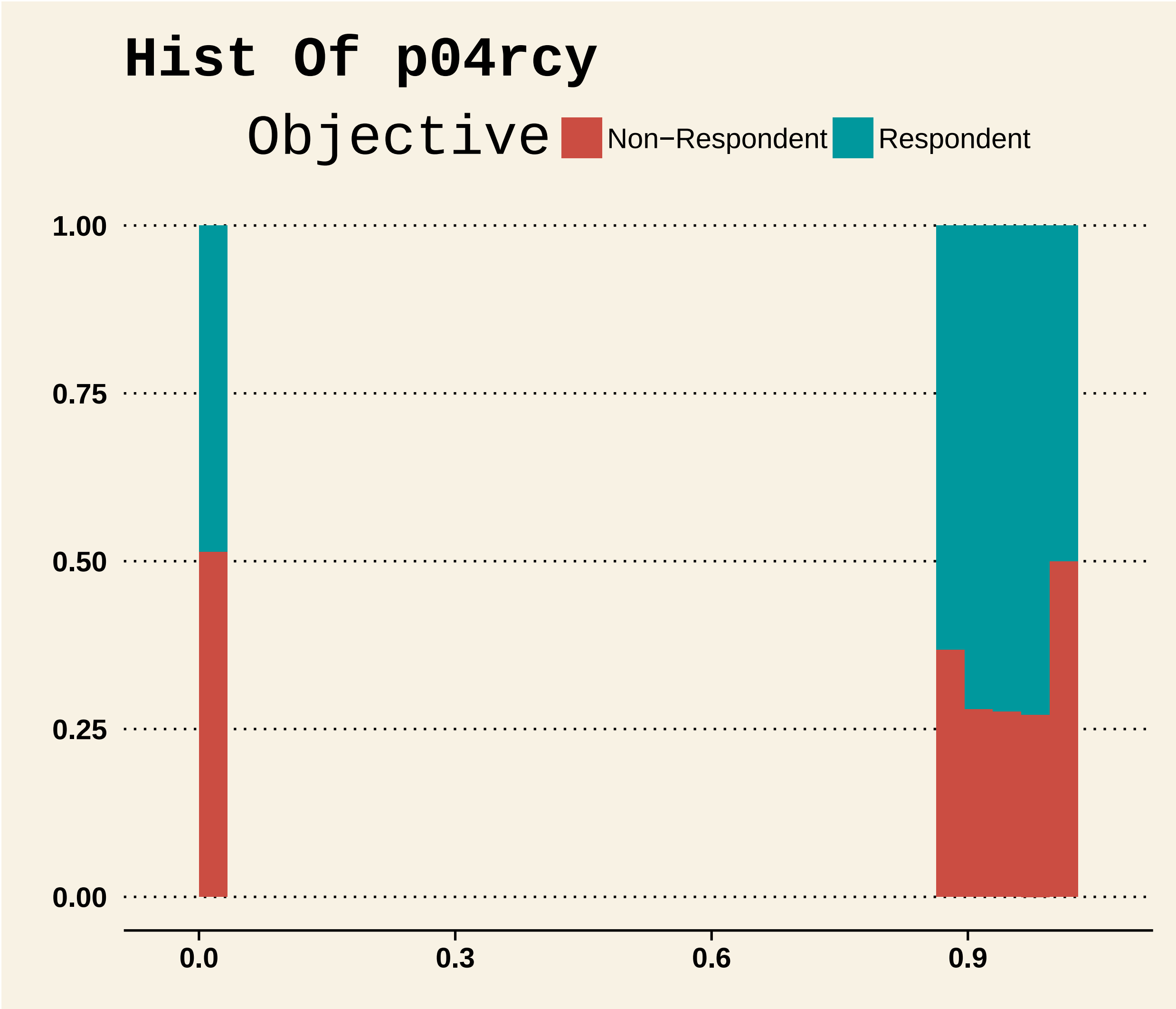
```
## Loading required package: ggthemes
```

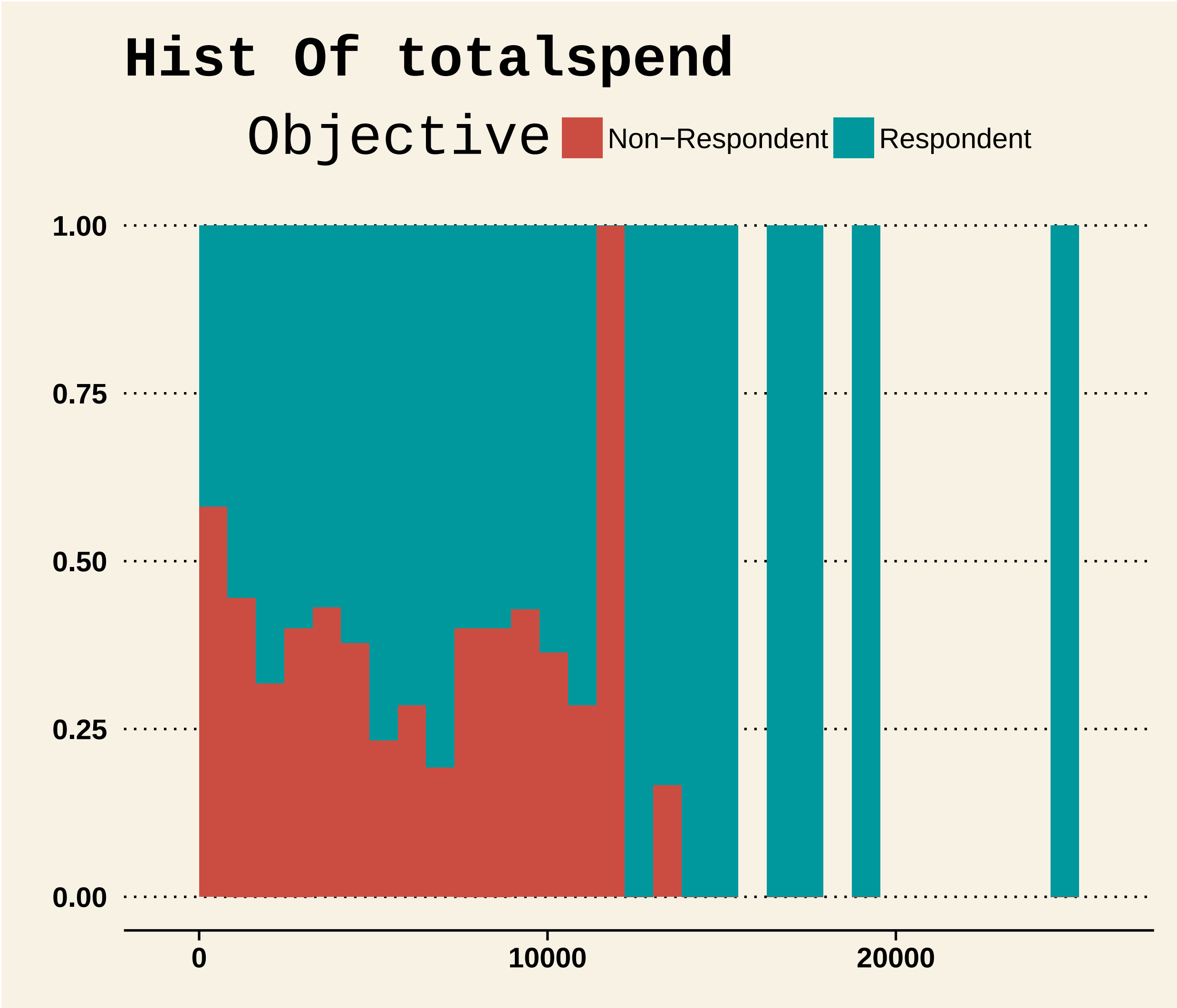
```
## Warning: package 'ggthemes' was built under R version 3.1.3
```

```r
colnames(dmm_data) <-variables[,2][1:201]
var.titles <- variables[,3][1:201]
dmm.cols <- colnames(dmm_data[2:201])

for(i in dmm.cols[2:5]){
  title <- paste('Hist Of',i)
  print(ggplot(dmm_data, aes(x=get(i), fill=factor(Objective))) +
        theme_wsj() +
        geom_histogram(position="fill",binwidth = diff(range(dmm_data[,i]))/30) +
        xlab(i) +
        ylab(paste('Frequency of',i)) +
        ggtitle(title) +
        scale_fill_hue(name="Objective",
                       labels=c("Non-Respondent", "Respondent"),l=50)

        )
}
```

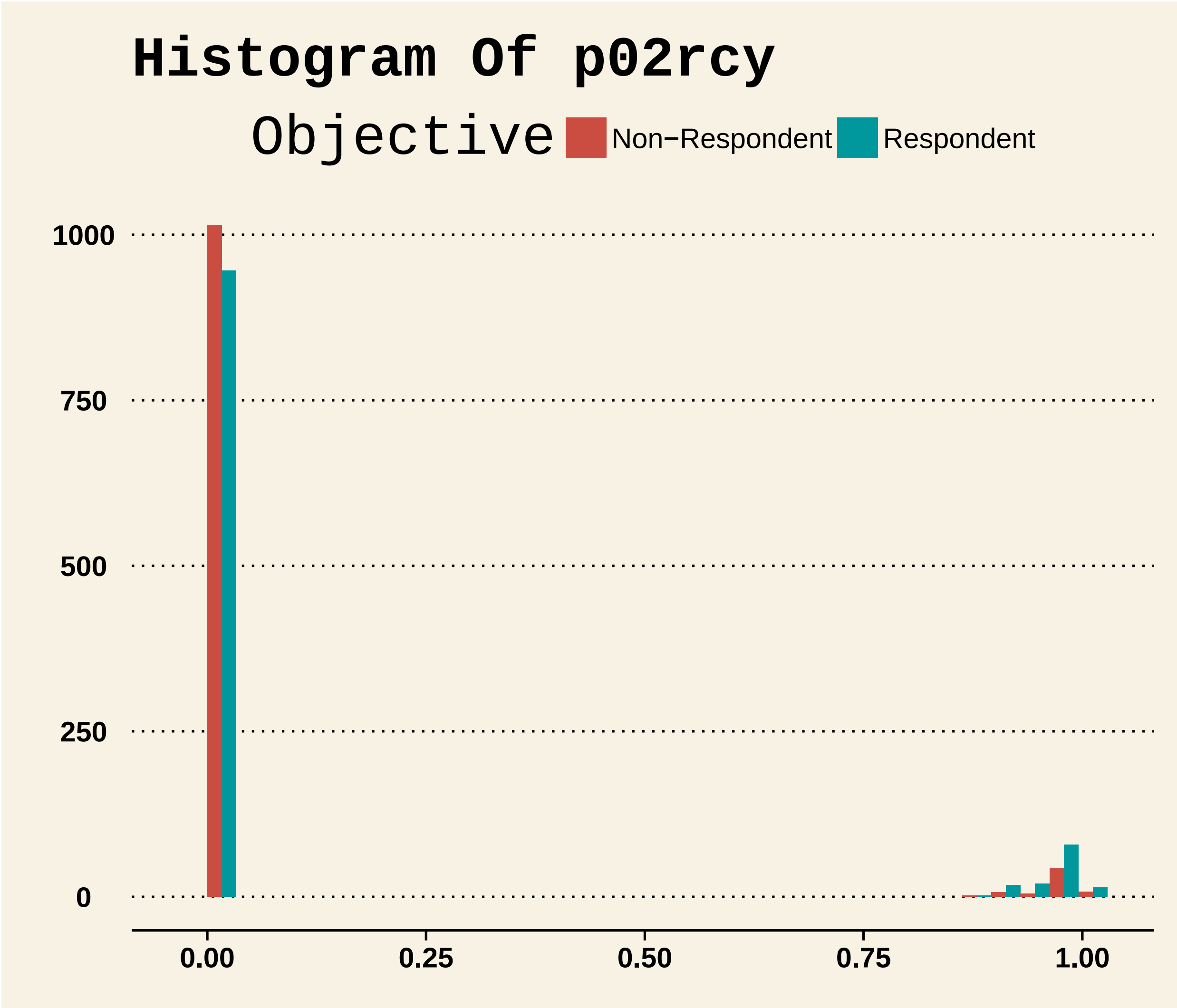

Hist Of p02rcy
Objective
Non−Respondent
Respondent
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00

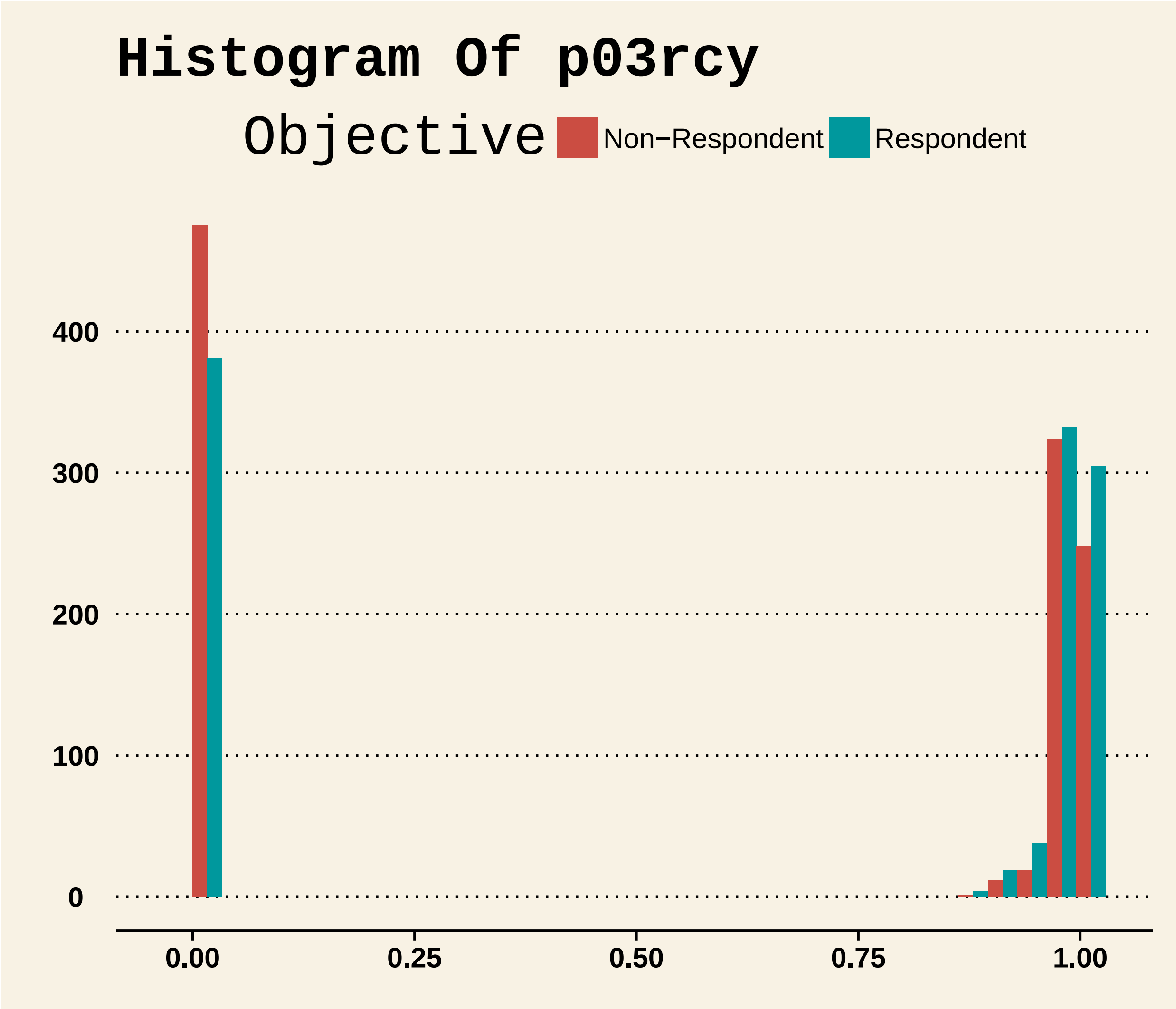

Hist Of p03rcy
Objective
Non−Respondent
Respondent
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00

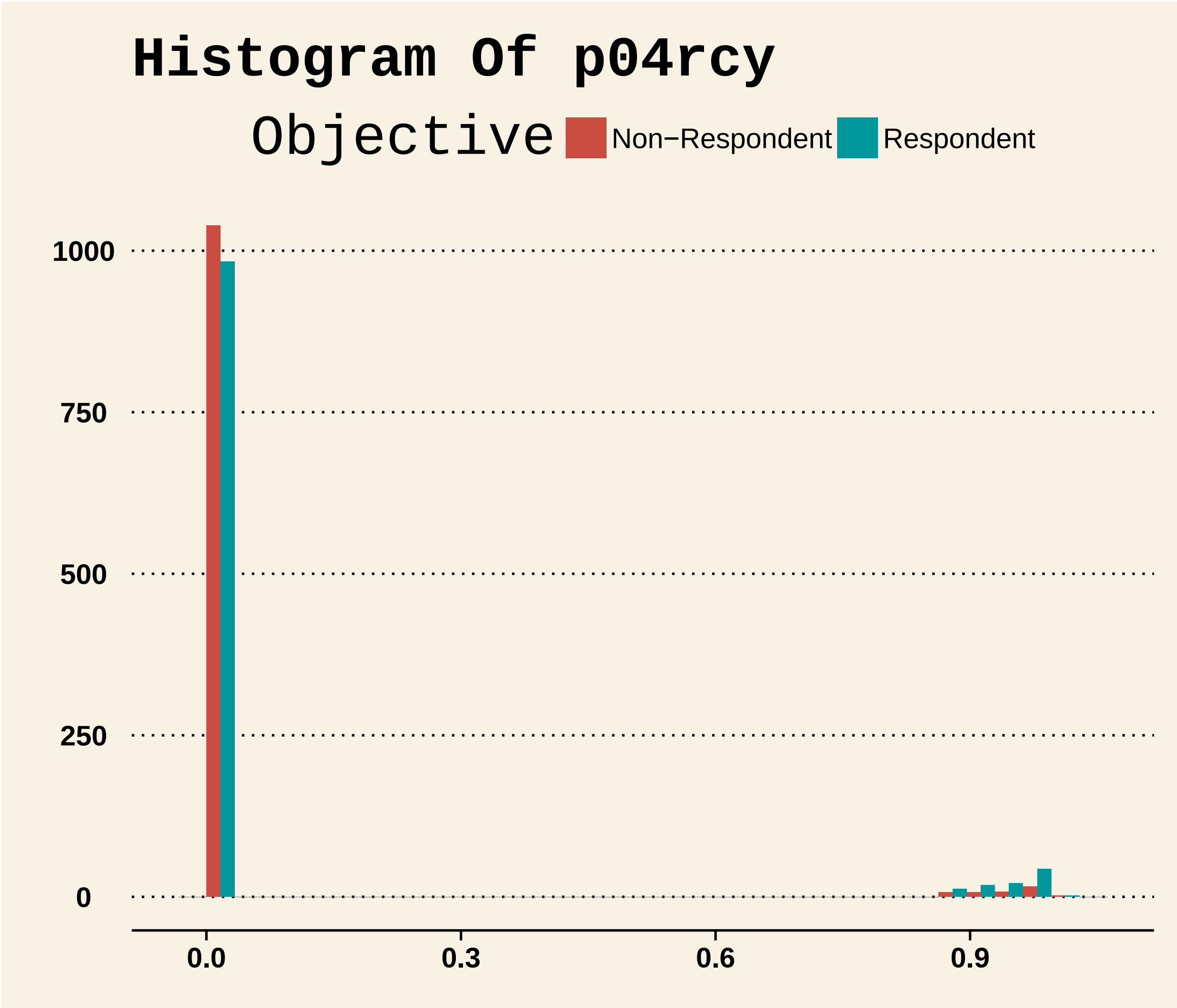

Hist Of p04rcy
Objective
Non−Respondent
Respondent
1.00
0.75
0.50
0.25
0.00
0.0
0.3
0.6
0.9

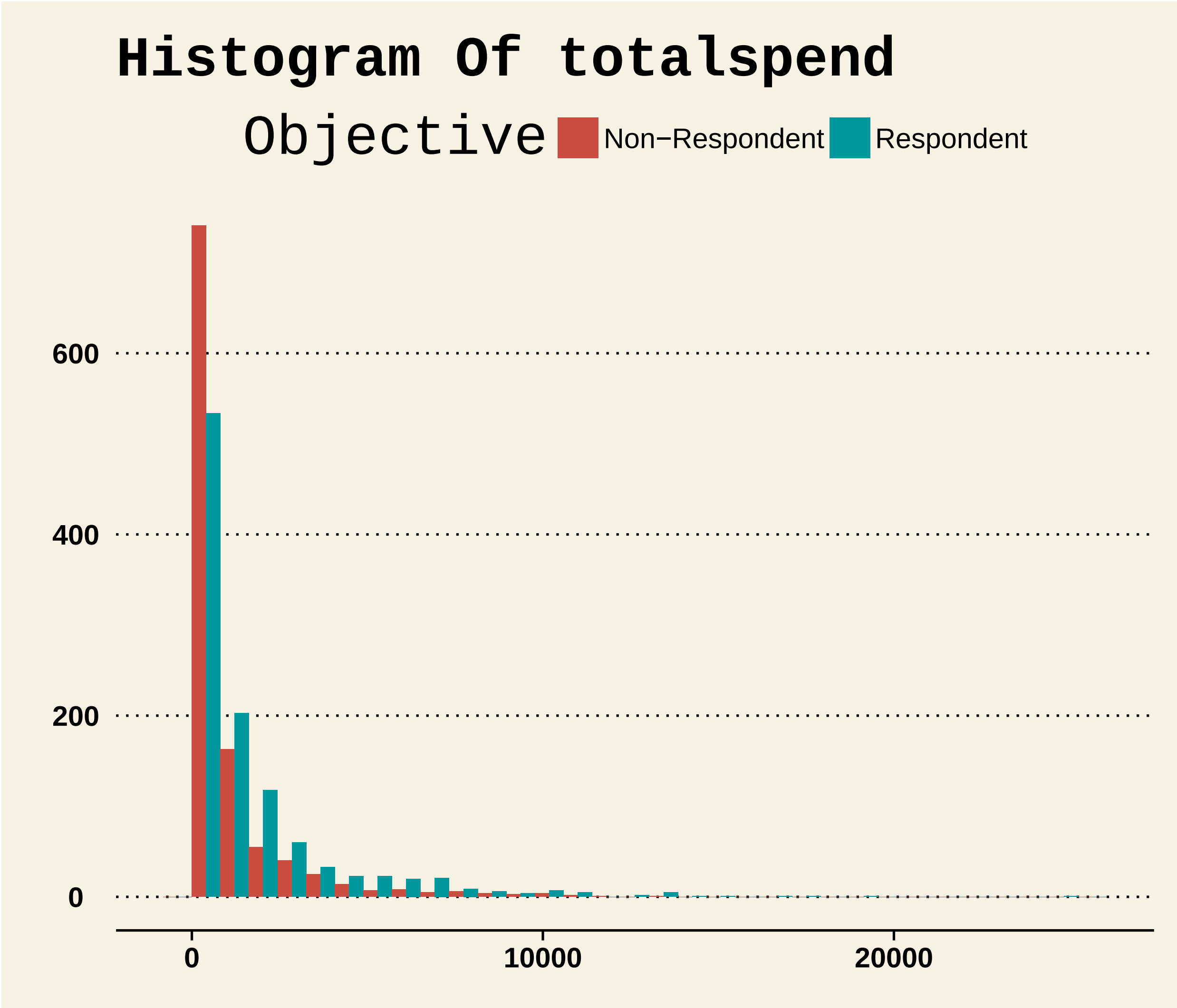



```
require(ggplot2)
require(ggthemes)

colnames(dmm_data) <-variables[,2][1:201]
var.titles <- variables[,3][1:201]
dmm.cols <- colnames(dmm_data[2:201])

for(i in dmm.cols[2:5]){
  print(ggplot(dmm_data, aes(x=get(i), fill=factor(Objective))) +
        theme_wsj() +
        geom_histogram(position="dodge",binwidth = diff(range(dmm_data[,i]))/30) +
        xlab(i) +
        ylab(paste('Frequency of',i)) +
        ggtitle(paste('Histogram Of',i)) +
        scale_fill_hue(name="Objective",
                       labels=c("Non-Respondent", "Respondent"),l=50)
        )
}
```


Histogram Of p02rcy
Objective
Non−Respondent
Respondent
1000
750
500
250
0
0.00
0.25
0.50
0.75
1.00


Histogram Of p03rcy
Objective
Non−Respondent
Respondent
400
300
200
100
0
0.00
0.25
0.50
0.75
1.00


Histogram Of p04rcy
Objective
Non−Respondent
Respondent
1000
750
500
250
0
0.0
0.3
0.6
0.9

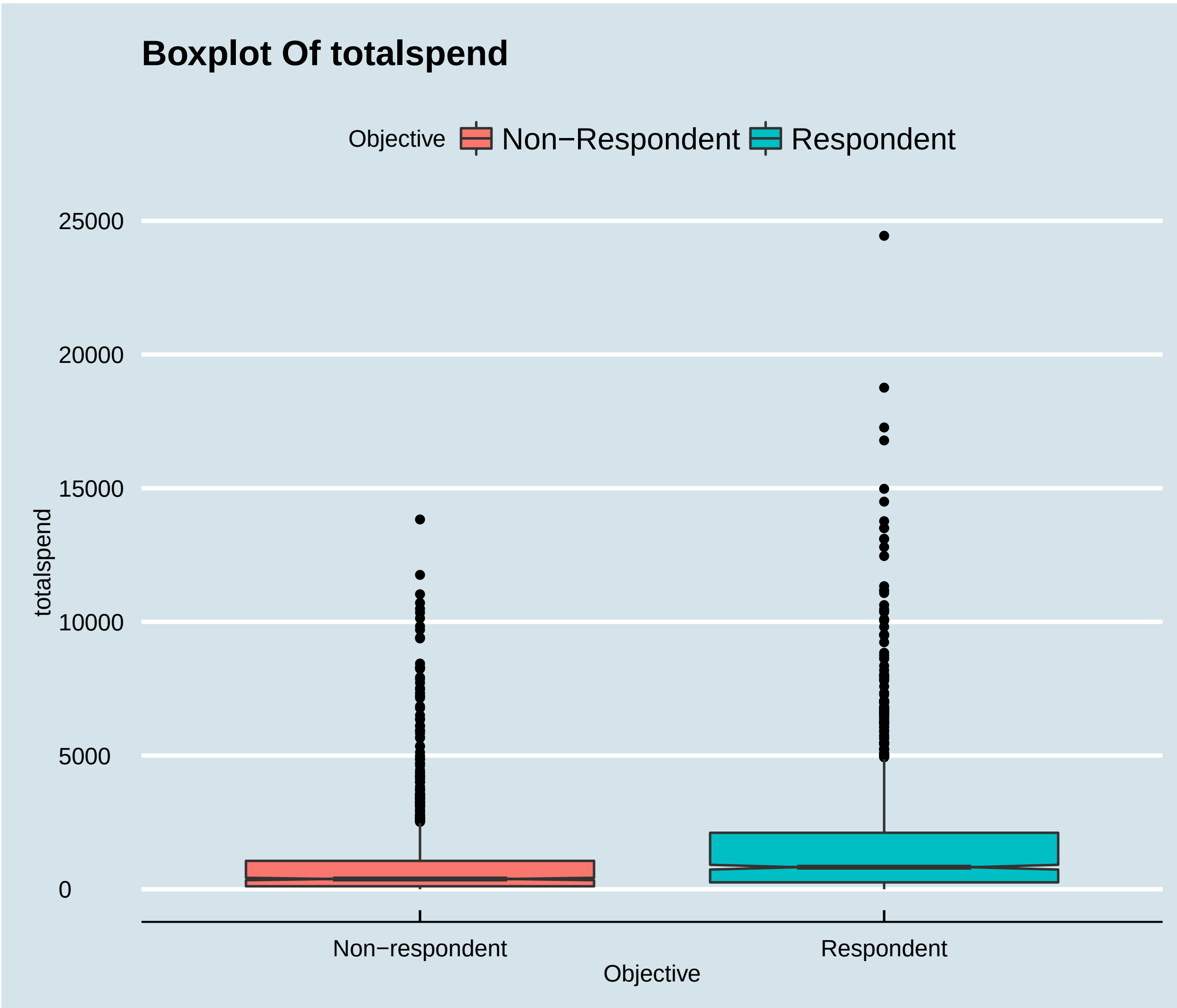


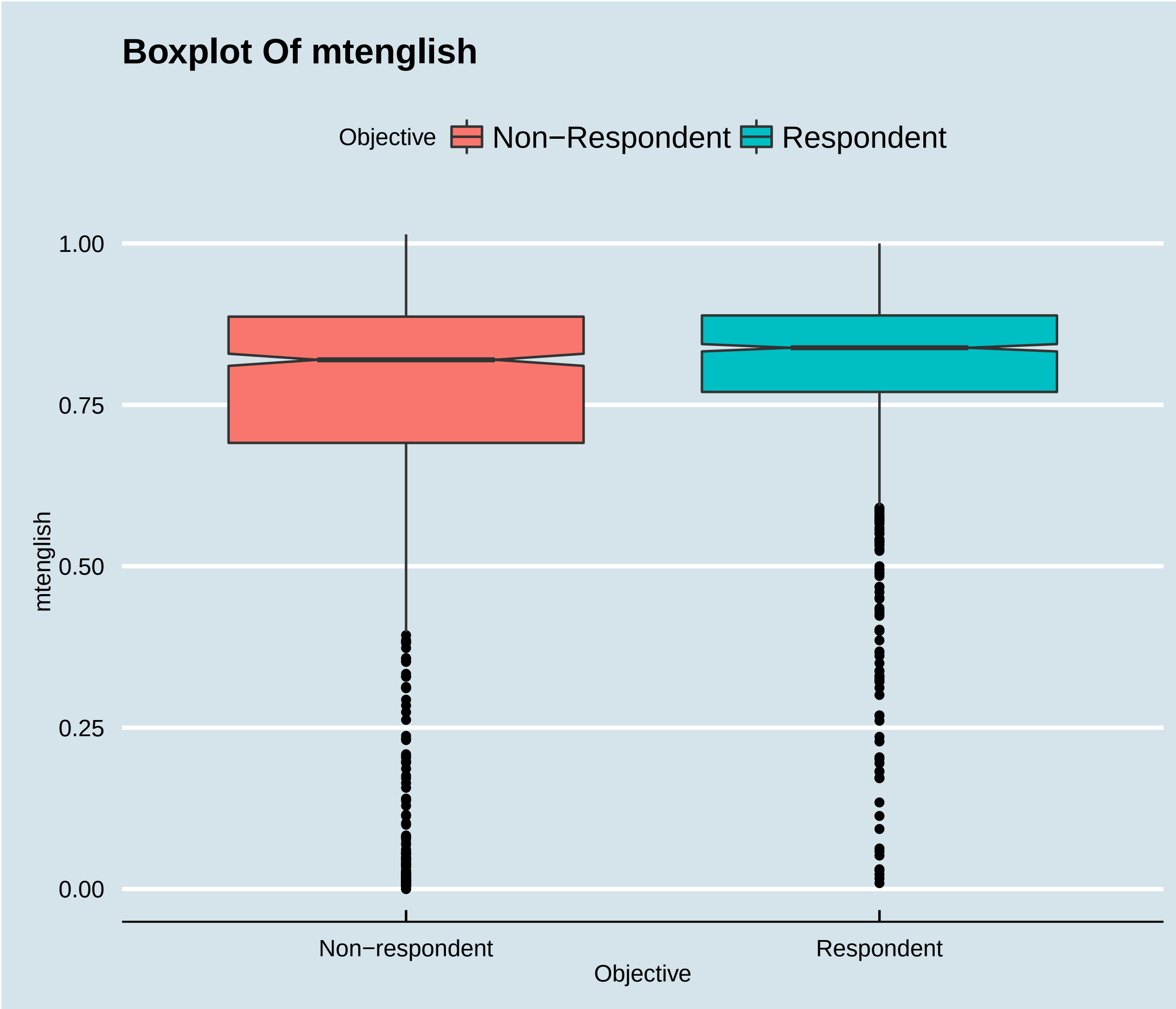
Boxlots:

For doing the data quality analysis, we plot the boxlpots off the data. This gives information about overall pattern of response, range, outliers and other important characteristics. By performing we step we compare the range, mean distributions and outlier details and recognised which data is left or right skewed, which variables need to be scalled.

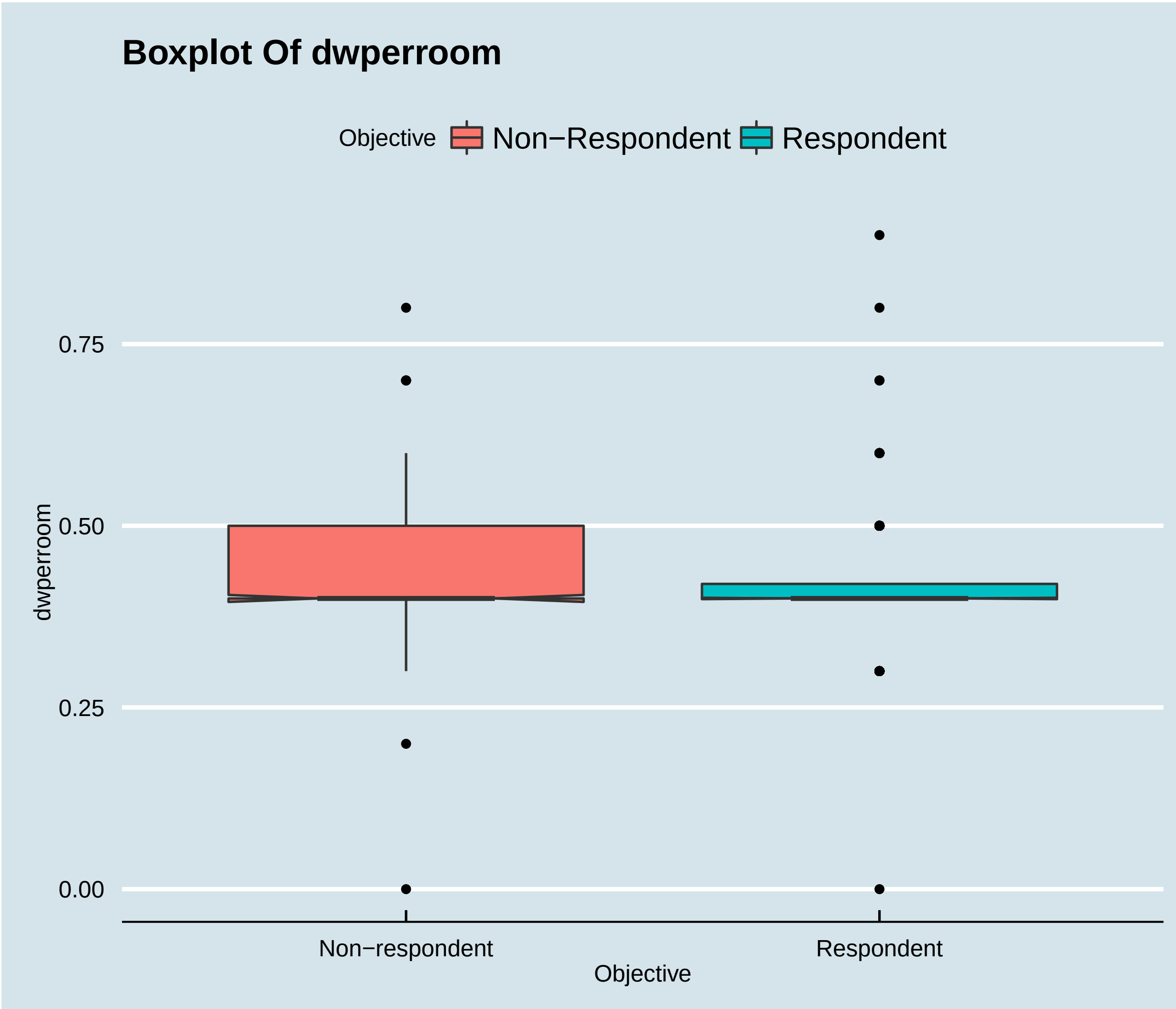
```
require(ggplot2)
require(ggthemes)

colnames(dmm_data) <-variables[,2][1:201]
dmm.cols <- colnames(dmm_data)

to_plots<-c("totalspend","mtenglish","dwperroom")

for(i in to_plots){
  title <- paste('Boxplot Of',i)
  print(ggplot(dmm_data, aes(y=get(i), x = as.factor(Objective), fill=factor(Objective))) +
    geom_boxplot( position="dodge", notch = TRUE) +
```

```
    scale_x_discrete(labels=c("Non-respondent", "Respondent"),name = 'Objective') +
    scale_y_continuous(name=paste(i)) +
    ggtitle(title) +
    scale_fill_hue(name="Objective",
                       labels=c("Non-Respondent", "Respondent")) +
    theme_economist() +
    coord_cartesian(ylim=c(min(dmm_data[,i]) - 0.05 * range(dmm_data[,i]),
         max(dmm_data[,i]) + 0.05 * range(dmm_data[,i])))
  )
}
```






```
## notch went outside hinges. Try setting notch=FALSE.
## notch went outside hinges. Try setting notch=FALSE.
```



Data Partitioning

In order to get consistent results, setting up the seed to a constant.

```
set.seed(1234)
```

Specifying the amount of trainig,validation and test data in ratio of 70:15:15:

```
# shuffle and split the data into three parts
indices <- nrow(dmm_data)
train.indeces <- sample(nrow(dmm_data), 0.7*indices)
validate.indices <- sample(setdiff(seq_len(nrow(dmm_data)), train.indeces), 0.15* indices)
test.indices <- setdiff(setdiff(seq_len(nrow(dmm_data)), train.indeces), validate.indices)
dmm.train <- dmm_data[train.indeces,]
dmm.test <- dmm_data[test.indices,]
dmm.validate <- dmm_data[validate.indices,]
```

Building Few Base Models

In any modelling techique this is the first step, to set some baseline models to check how well you have done by using data modelling techiques. Build model with all 201 variables. In our case we are going to do three widely used modelling techniques for classification: -Nueral Network -Random Forest -Decision Tree

Because of its ease of use and support for widely used modelling techniques we have chosen to use 'caret' library.

```
set.seed(1234)
# set label name and predictors
labelName <- 'Objective'
predictors <- names(dmm.train)[names(dmm.train) != labelName]

library(caret)
require(pROC)
## create a caret control object to control the number of cross-validations performed
# Setting: adptive_cv as method
# Either the number of folds or number of resampling iterations = 3
# Don't save any summary Matrix from resample
var.control <- trainControl(method='cv', number=3, returnResamp='none')

# benchmark model 1 Nueral Network
test_nn <- train(dmm.train[,predictors], dmm.train[,labelName], method='nnet',
                 trControl=var.control)
```

```
## # weights:  203
## initial  value 253.657815
## final  value 251.169835
## converged
## # weights:  607
## initial  value 274.833136
## iter  10 value 250.323609
## final  value 250.323604
## converged
## # weights:  203
## initial  value 255.507654
## iter  10 value 250.847624
## iter  20 value 249.312219
## iter  30 value 244.591675
## iter  40 value 241.535213
## iter  50 value 240.908988
## iter  60 value 240.731366
## iter  70 value 240.470400
## iter  80 value 240.088151
## iter  90 value 239.566441
## iter 100 value 239.333697
## final  value 239.333697
## stopped after 100 iterations
## # weights:  607
## initial  value 266.932114
## iter  10 value 251.576503
## iter  20 value 249.472800
## iter  30 value 245.991978
## iter  40 value 244.275776
## iter  50 value 243.380233
## iter  60 value 243.166441
```

```
## iter  70 value 242.294619
## iter  80 value 240.118194
## iter  90 value 239.489049
## iter 100 value 239.212660
## final  value 239.212660
## stopped after 100 iterations
## # weights:  203
## initial  value 262.687769
## final  value 250.737626
## converged
## # weights:  607
## initial  value 297.296473
## iter  10 value 250.308058
## iter  20 value 249.784916
## iter  30 value 249.756153
## iter  40 value 249.708457
## iter  50 value 249.513415
## iter  60 value 249.176525
## iter  70 value 248.930451
## iter  80 value 248.928072
## iter  90 value 248.910586
## iter 100 value 248.877194
## final  value 248.877194
## stopped after 100 iterations
## # weights:  203
## initial  value 272.850335
## final  value 251.511420
## converged
## # weights:  607
## initial  value 269.502093
## iter  10 value 250.765130
## iter  20 value 249.853868
## iter  30 value 249.837659
## final  value 249.837655
## converged
## # weights:  203
## initial  value 255.081031
## iter  10 value 251.115117
## iter  20 value 249.029396
## iter  30 value 247.272020
## iter  40 value 246.259188
## iter  50 value 245.950352
## iter  60 value 245.425996
## iter  70 value 245.334963
## iter  80 value 245.151756
## iter  90 value 244.835086
## iter 100 value 244.790740
## final  value 244.790740
## stopped after 100 iterations
## # weights:  607
## initial  value 272.506154
## iter  10 value 250.503830
## iter  20 value 249.897633
## iter  30 value 247.543813
```

```
## iter  40 value 245.328786
## iter  50 value 243.689748
## iter  60 value 242.649066
## iter  70 value 242.061659
## iter  80 value 241.659217
## iter  90 value 241.088719
## iter 100 value 240.653348
## final  value 240.653348
## stopped after 100 iterations
## # weights:  203
## initial  value 260.087948
## final  value 251.400918
## converged
## # weights:  607
## initial  value 272.537553
## iter  10 value 251.539320
## iter  20 value 251.105261
## iter  30 value 251.089898
## iter  40 value 249.848775
## iter  50 value 249.477278
## iter  60 value 249.092585
## iter  70 value 248.156902
## iter  80 value 247.727974
## iter  90 value 247.363629
## iter 100 value 247.226200
## final  value 247.226200
## stopped after 100 iterations
## # weights:  203
## initial  value 271.659639
## final  value 250.556530
## converged
## # weights:  607
## initial  value 268.117301
## iter  10 value 246.403661
## final  value 246.401680
## converged
## # weights:  203
## initial  value 284.441951
## iter  10 value 250.159814
## iter  20 value 248.472169
## iter  30 value 247.976170
## iter  40 value 247.718256
## iter  50 value 247.409757
## iter  60 value 246.040151
## iter  70 value 245.766457
## iter  80 value 245.347481
## iter  90 value 244.516662
## iter 100 value 243.618810
## final  value 243.618810
## stopped after 100 iterations
## # weights:  607
## initial  value 266.197239
## iter  10 value 253.179379
## iter  20 value 249.465013
```

```
## iter  30 value 248.994023
## iter  40 value 248.578379
## iter  50 value 247.961764
## iter  60 value 246.801872
## iter  70 value 245.630678
## iter  80 value 245.167768
## iter  90 value 244.719826
## iter 100 value 244.598316
## final  value 244.598316
## stopped after 100 iterations
## # weights:  203
## initial  value 260.719977
## final  value 251.725356
## converged
## # weights:  607
## initial  value 335.179674
## iter  10 value 251.053798
## iter  20 value 250.776576
## iter  30 value 250.763426
## iter  40 value 246.752585
## iter  50 value 246.635860
## iter  60 value 246.410658
## iter  70 value 246.408088
## iter  80 value 246.178340
## iter  90 value 246.164999
## iter 100 value 245.743677
## final  value 245.743677
## stopped after 100 iterations
## # weights:  607
## initial  value 492.232380
## iter  10 value 377.625364
## iter  20 value 374.405631
## iter  30 value 373.845101
## iter  40 value 373.627483
## iter  50 value 370.581776
## iter  60 value 366.967692
## iter  70 value 366.169731
## iter  80 value 363.822778
## iter  90 value 362.577713
## iter 100 value 361.972612
## final  value 361.972612
## stopped after 100 iterations
```

```r
pred_nn <- (ifelse(predict(object=test_nn, dmm.test[,predictors])< 0.5,0,1))
auc_nn <- roc(dmm.test[,labelName], pred_nn)
table(dmm.test$Objective,pred_nn)
```

```
##    pred_nn
##       0   1
##   0  13 146
##   1  15 151
```

```r
acc_nn <- sum(dmm.test$Objective==pred_nn) / nrow(dmm.test) #Accuracy: 0.500
print(auc_nn$auc)  # Area under the curve: 0.5196636
```

```
## Area under the curve: 0.4957
```

```r
# benchmark model 2 Random Forest
test_rf <- train(dmm.train[,predictors], dmm.train[,labelName], method='rf',
                 trControl=var.control)

pred_rf <- (ifelse(predict(object=test_rf, dmm.test[,predictors])< 0.5,0,1))
auc_rf <- roc(dmm.test[,labelName], pred_rf)
table(dmm.test$Objective,pred_rf)
```

```
##    pred_rf
##       0   1
##   0 109  50
##   1  34 132
```

```r
acc_rf <- sum(dmm.test$Objective==pred_rf) / nrow(dmm.test) # Accuracy: 0.7538
print(auc_rf$auc)  # Area under the curve: 0.7518754
```

```
## Area under the curve: 0.7404
```

```r
# benchmark model 3 Decision Tree
test_rpart <- train(dmm.train[,predictors], dmm.train[,labelName],
                    method='rpart', trControl=var.control)

pred_rpart <- (ifelse(predict(object=test_rpart, dmm.test[,predictors])< 0.5,0,1))
auc_rpart <- roc(dmm.test[,labelName], pred_rpart)
table(dmm.test$Objective,pred_rpart)
```

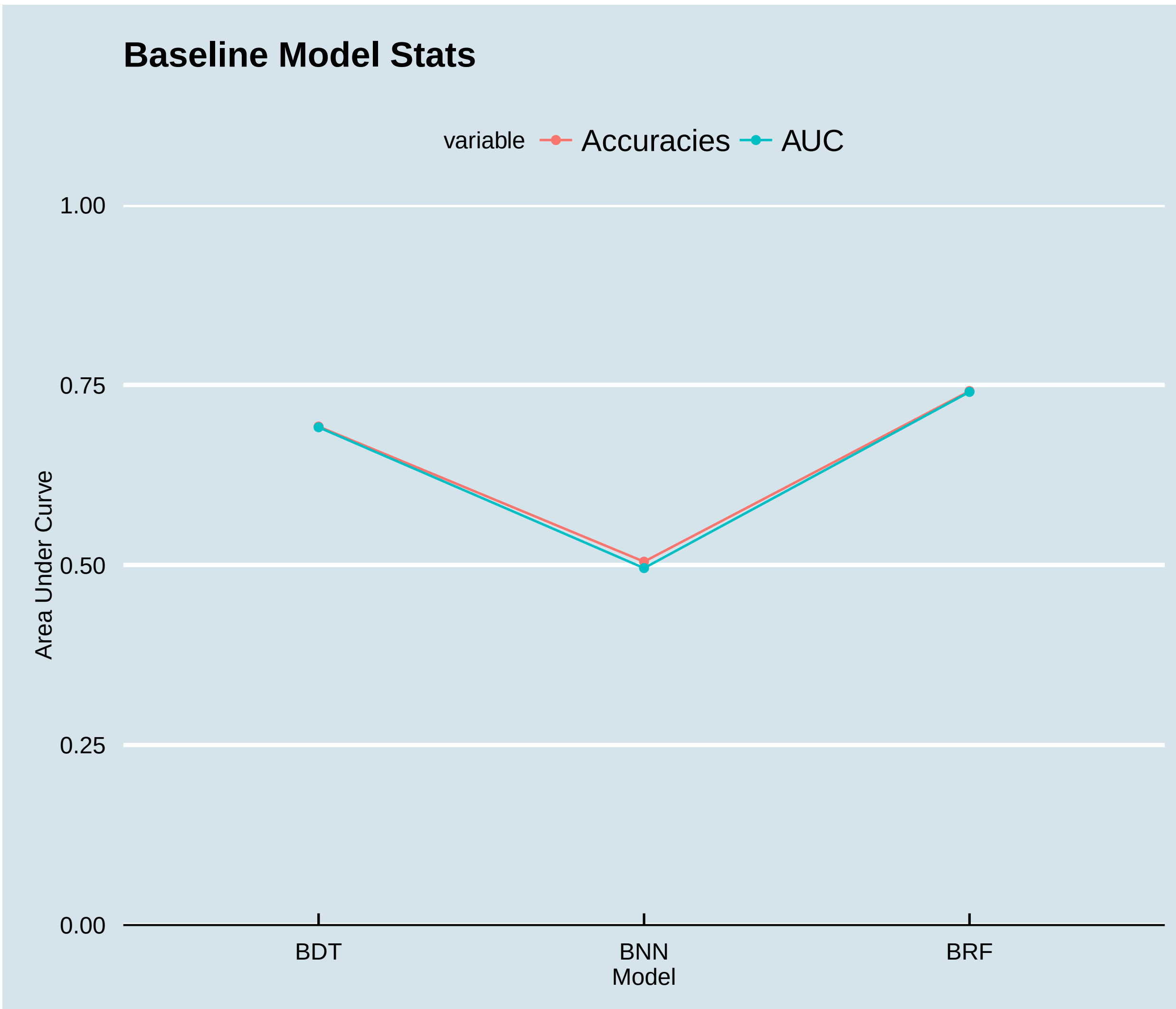
```
##    pred_rpart
##       0   1
##   0 102  57
##   1  43 123
```

```r
acc_rpart <- sum(dmm.test$Objective==pred_rpart) / nrow(dmm.test) #Accuracy: 0.69538
print(auc_rpart$auc)  # Area under the curve: 0.7518754
```

```
## Area under the curve: 0.6912
```

```r
## Combining all the models into one Data Frame
accuracies <- c(acc_nn,acc_rf,acc_rpart)
auc <- c(auc_nn$auc,auc_rf$auc,auc_rpart$auc)
bmodels <- as.data.frame(cbind(accuracies,auc))
colnames(bmodels) <- c('Accuracies','AUC')
rownames(bmodels) <- c('BNN','BRF','BDT')
require(reshape)
bmodels[ "Model" ] <- rownames(bmodels)
model.melt <- melt( bmodels, id.vars="Model", value.name="Accuracies", variable.name="AUC" )
```

```r
require(ggplot2)
require(ggthemes)
ggplot(data=model.melt, aes(x= Model, y=value, group = variable, colour = variable)) +
        theme_economist() +
        geom_line() +
        xlab("Model") +
        ylab("Area Under Curve") +
        ggtitle("Baseline Model Stats") +
        coord_cartesian(ylim= c (0,1)) +
        geom_point()
```



Model-1.pdf

Data Transformation:

By looking on the summary statistics of the dataset by using `summary()` command, we observed 57 records to be non-normalised. The solution of this gives `Max-Min` and `Median-Mean` values. If `mean` is larger than `median` we can say the data would be right-skewed.

These have to be transfomred for ahead processing to get efficient results. We do it by scalng variables to [0,1] the data for making our data standardize.

For tranformation we could use logarithmic transformation but for data with zero we would get unacceptable results, certain observations would be imputed by default.

```r
scale.cols <- c("totalspend","totaltrans","cfhuswife","mtenglish","mtsingres",
                "dw86to91","dwmaint","fiinca","fiincm","fsmlabf","hiinca",
                "hiincm","hiincs","hlenglish","hlfrench","improvres",
                "incgovp","ineflowp","infinca","infincm","infincs",
                "inhhlow","inhhlowp","inmfemina","inminca","inmincm",
                "inmincs","knenglish","lfmaunemr","lfmtunemr","lftaempl",
                "lfttempl","lfttunemr","moy5mov","ndtallind","rlcathol",
                "rlrcathol","first","productcount","productcount6","tenure",
                "tf108","tf129","tf27","tf37","tf38","tf39","tf68","tf73",
                "tf74","tf75","tf88","tf89","tf90","tf94","tf95","tf96")

minVals   <- sapply(dmm.train[,scale.cols],min)
ranges <- sapply(dmm.train[,scale.cols],function(x)diff(range(x)))

train.scaled <- as.data.frame(scale(dmm.train[,scale.cols],center=minVals,scale=ranges))

# Scale using the training parameters for minVals and Range.
validate.scaled <- as.data.frame(scale(dmm.validate[,scale.cols],center=minVals,scale=ranges))
test.scaled <- as.data.frame(scale(dmm.test[,scale.cols],center=minVals,scale=ranges))
#Assignin the values to actual variables
dmm.train[,scale.cols] <- train.scaled[,scale.cols]
dmm.test[,scale.cols] <- test.scaled[,scale.cols]
dmm.validate[,scale.cols] <- validate.scaled[,scale.cols]
```

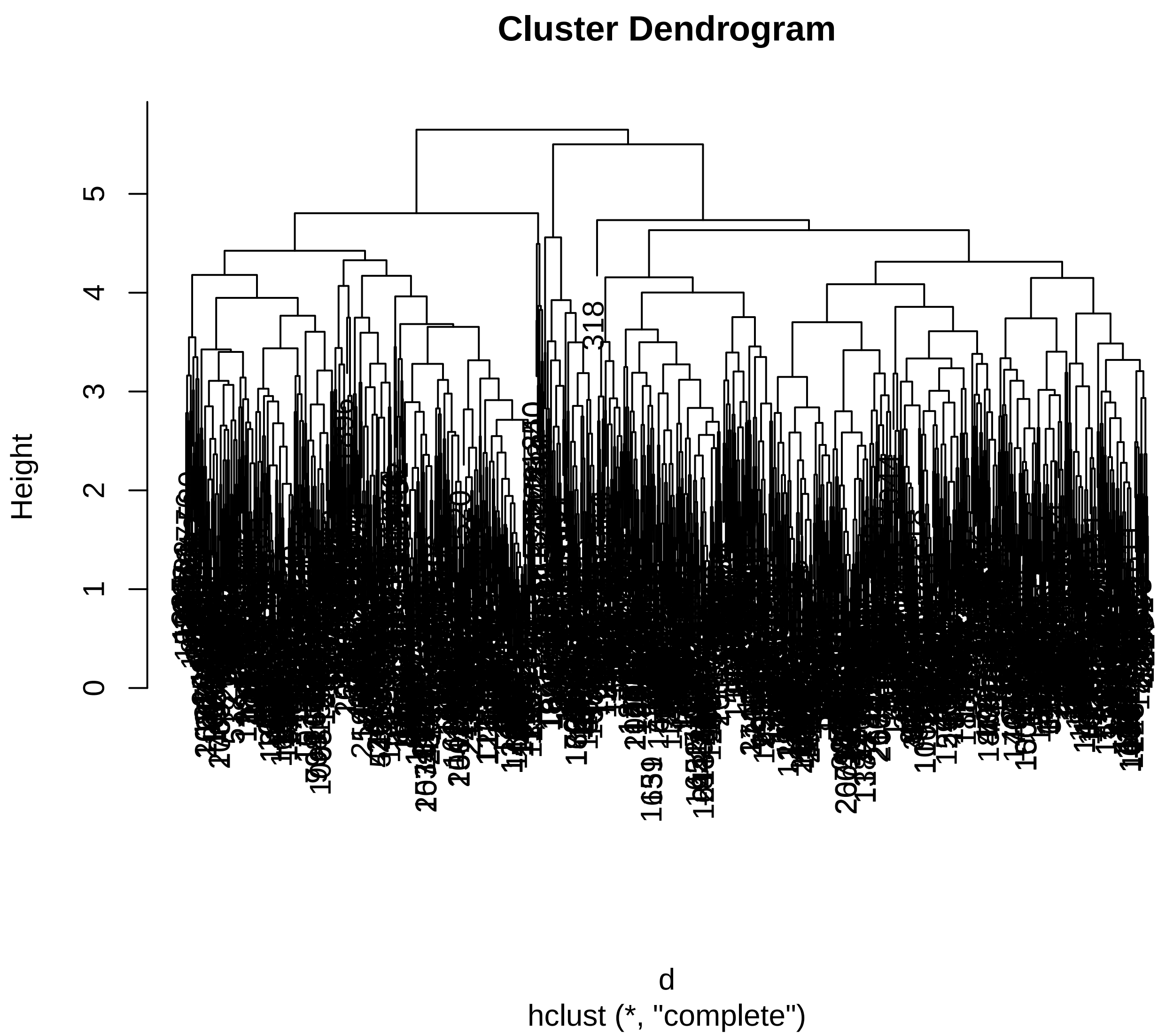
Knowledge Discovery

In this, we will perform descriptive analysis which is a form of unsupervised learning. We will look for some kind of relationship.

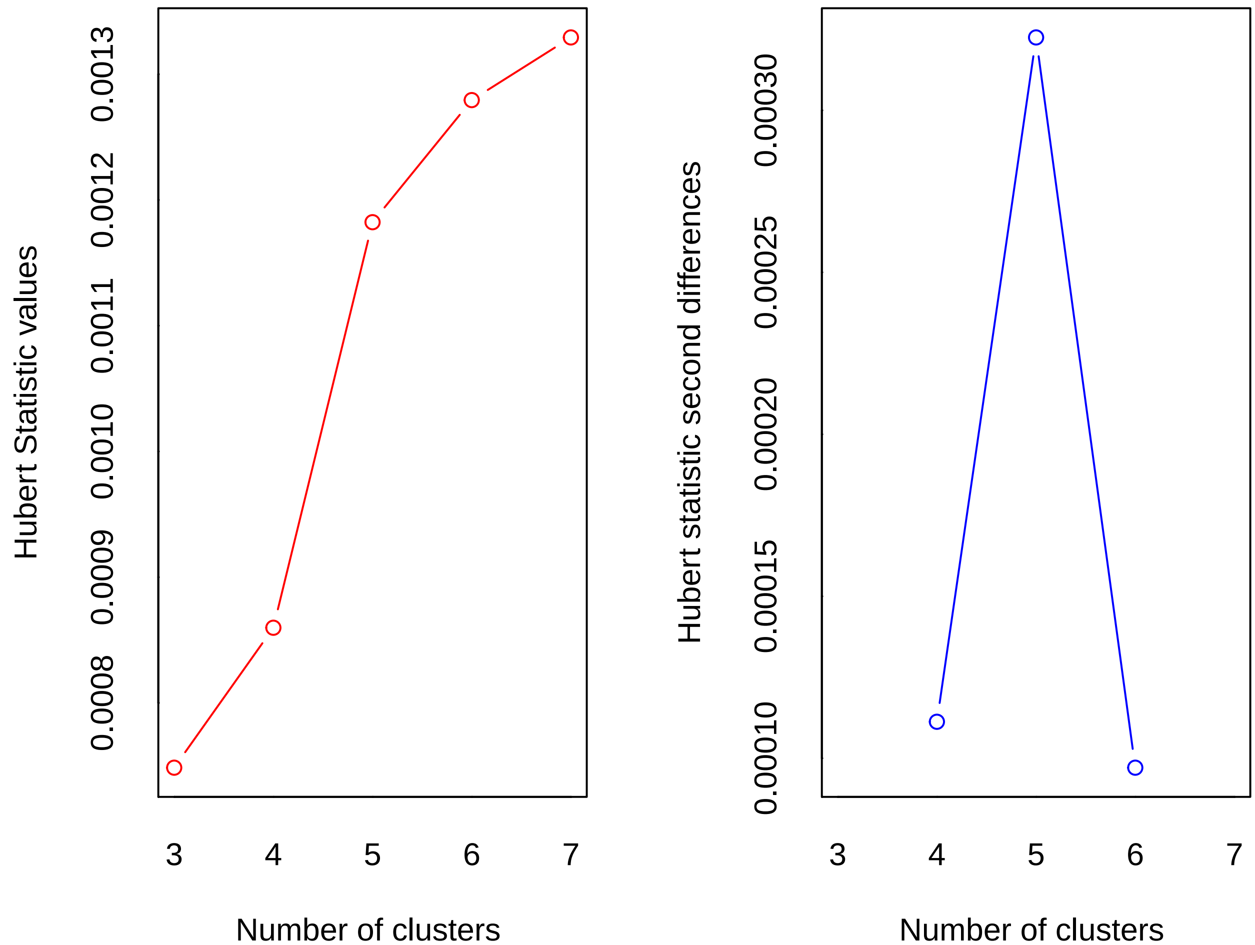
Cluster Analysis

For performing hierarchical clustering, we use various packages. The important one is `NbClust` library which suggest number of clusters to make to have optimised results. By plotting dendogram, we could see various clusters are made but we can see that making less then 5 clusters would be good.

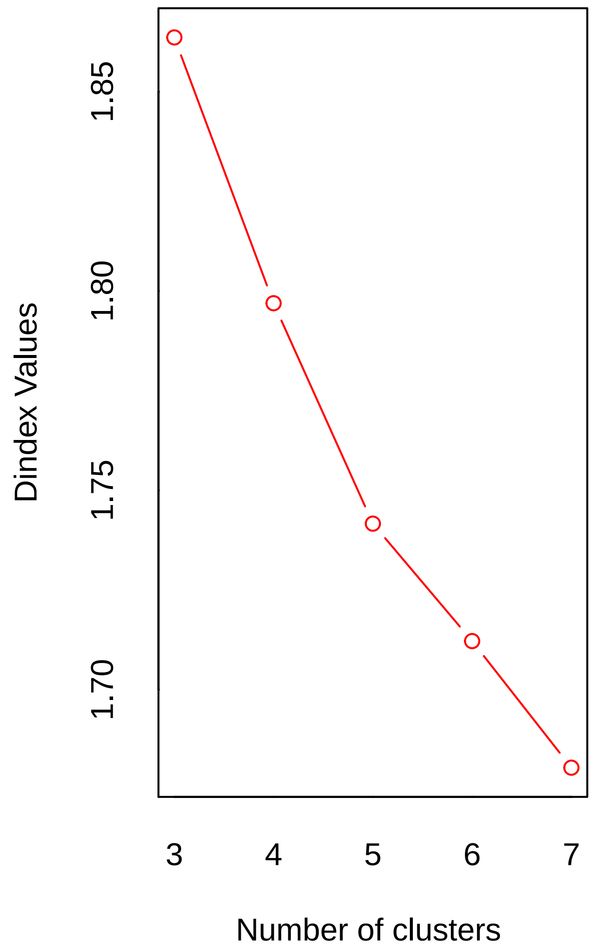
```r
library(MASS)
library(HSAUR)
library(cluster)
library(fpc)

d <- dist(as.matrix(dmm.train))   # find distance matrix
hc <- hclust(d)                  # apply hirarchical clustering
plot(hc)                         # plot the dendrogram
```

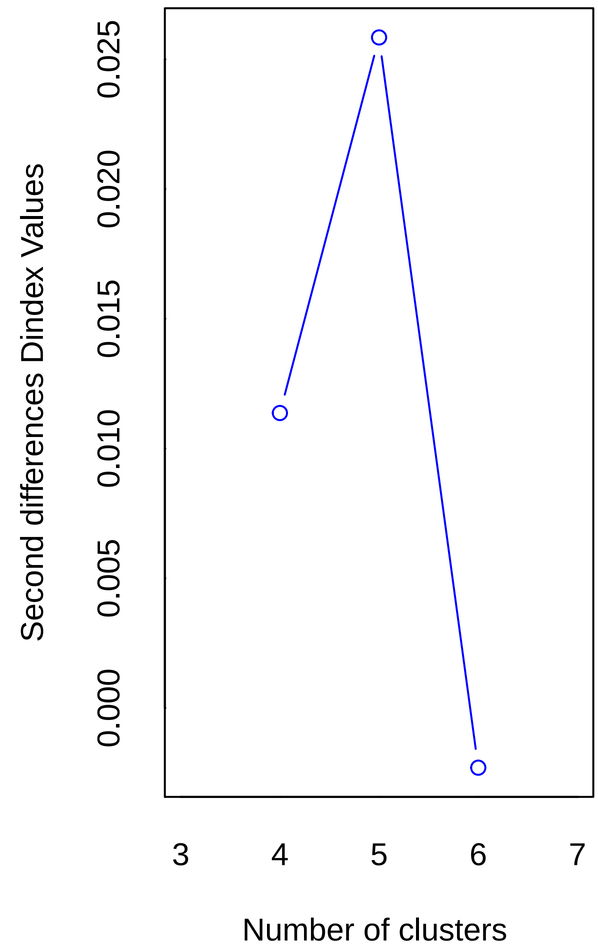



```
kc <- kmeans(dmm.train, 3)

#install.packages('NbClust')
library(NbClust)
# used for better cluster analysis.
#Gives various parameters to analysis and interpret better.

nc <- NbClust(dmm.train, min.nc=3, max.nc=7, method="kmeans")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```






```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## All 1510 observations were used.
##
## *******************************************************************
## * Among all indices:
## * 6 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 7 proposed 5 as the best number of clusters
## * 3 proposed 6 as the best number of clusters
## * 4 proposed 7 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  5
##
##
## *******************************************************************
```

```r
barplot(table(nc$Best.n[1,]),
        xlab="Numer of Clusters", ylab="Number of Criteria",
        main="Number of Clusters Chosen by 26 Criteria")
nc$Best.nc
```

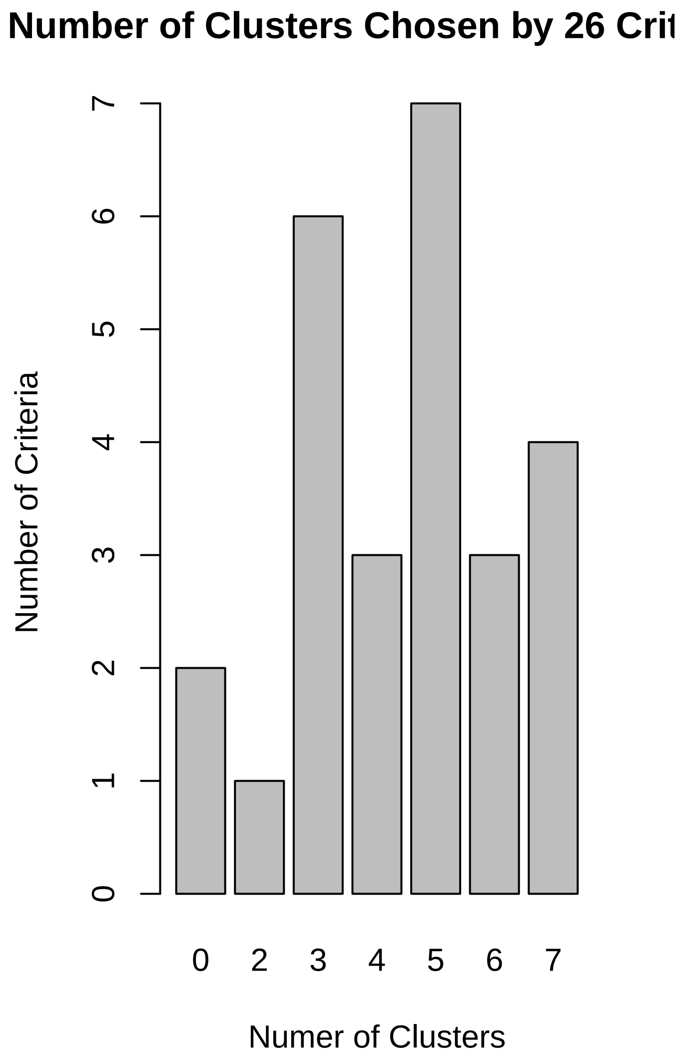
```
##                     KL       CH Hartigan     CCC    Scott       Marriot
## Number_clusters 5.0000   3.0000   5.0000  7.0000    6.000  6.000000e+00
## Value_Index     2.2002 138.1249  56.0465 35.3217 1604.561 -1.265803e+14
##                   TrCovW   TraceW     Friedman   Rubin Cindex     DB
## Number_clusters    5.000   5.0000 7.000000e+00  5.0000 7.0000 5.0000
## Value_Index     1216.279 184.2345 1.440045e+15 -0.1928 0.4825 2.3145
##                 Silhouette   Duda  PseudoT2    Beale Ratkowsky     Ball
## Number_clusters     5.0000 3.0000    3.0000   3.0000    3.0000   4.0000
## Value_Index         0.1164 1.6895 -319.1392 -58.8429    0.1142 545.9187
##                 PtBiserial Frey McClain   Dunn Hubert SDindex Dindex
## Number_clusters     6.0000    2  3.0000 4.0000      0  4.0000      0
## Value_Index         0.4269   NA  1.7144 0.1507      0  2.2706      0
##                   SDbw
## Number_clusters 7.0000
## Value_Index     0.7172
```

```r
## Selecting 4 clusters to do clustering as per suggested by nb cluster library.
set.seed(1234)
fit.km <- kmeans(dmm.train, 5, nstart=25)                          #4
fit.km$size
```

```
## [1] 219  82 446 461 302
```

```r
table(dmm.train$Objective, fit.km$cluster)
```

```
##    
##       1   2   3   4   5
##   0 106  72 369 141  63
##   1 113  10  77 320 239
```



From the above matrix we see clusters 3 and 2 to be interesting.

```
fit.km$centers
```

```
aggregate(dmm.train[-1], by=list(cluster=fit.km$cluster), mean)
```

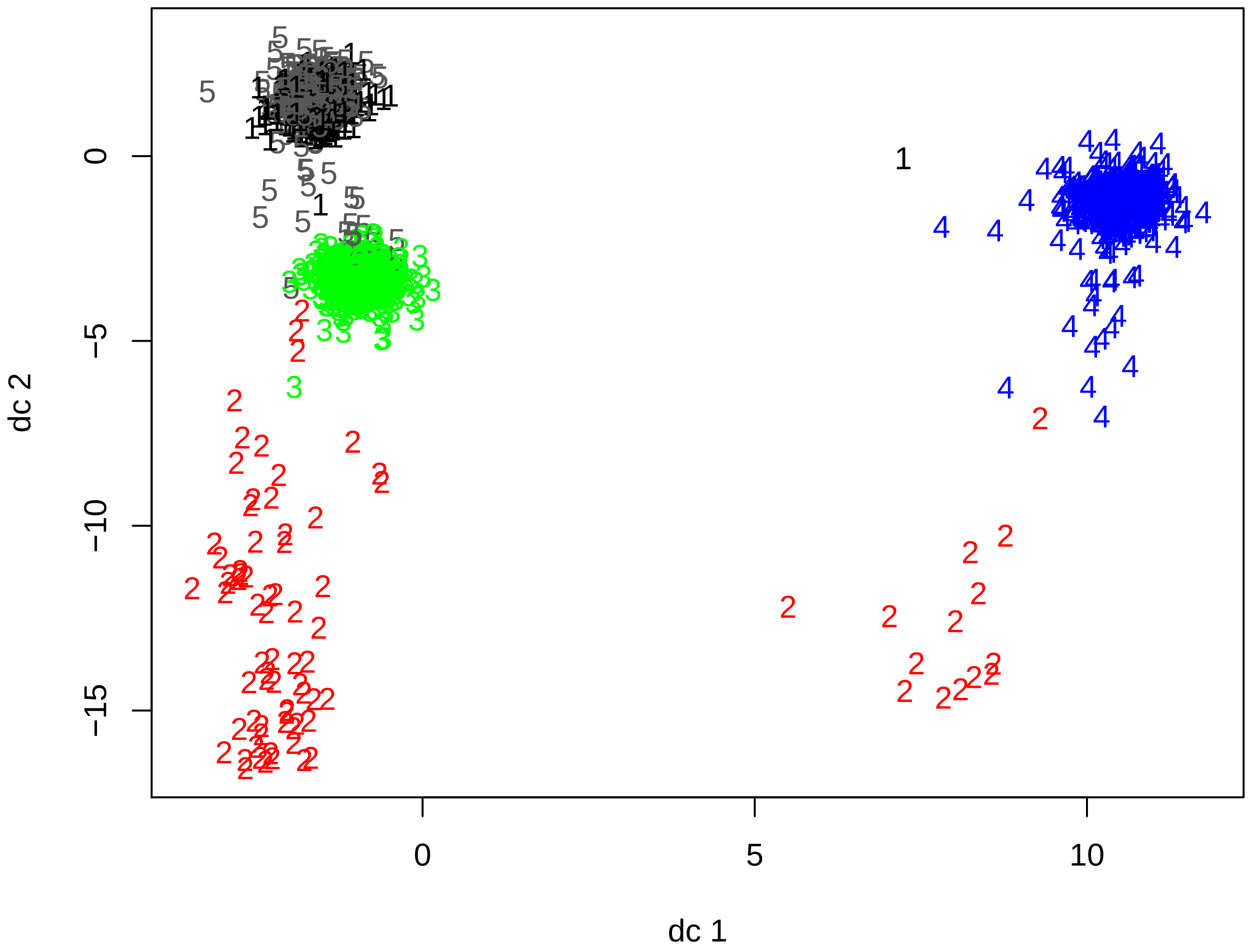
Plotting Parallel cordinates as per the objective in order to understand the relation more.

We will see that some objects will share common attributes and thus will have `similarity` characterization which means the ratio of the number of attributes two objects share in common compared to the total list of attributes between them.

```
table(dmm.train$Objective, fit.km$cluster)
```

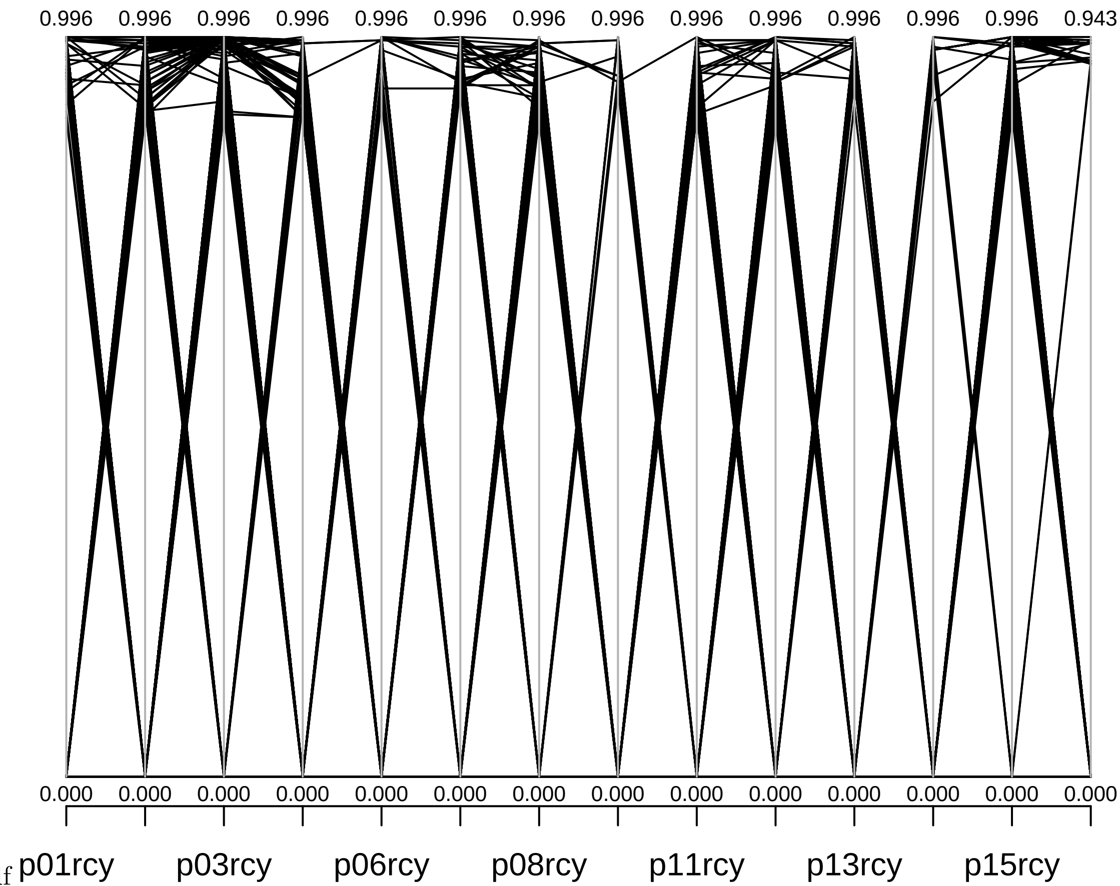
```
##
##       1   2   3   4   5
##   0 106  72 369 141  63
##   1 113  10  77 320 239
```

```
plotcluster(dmm.train, fit.km$cluster)
```



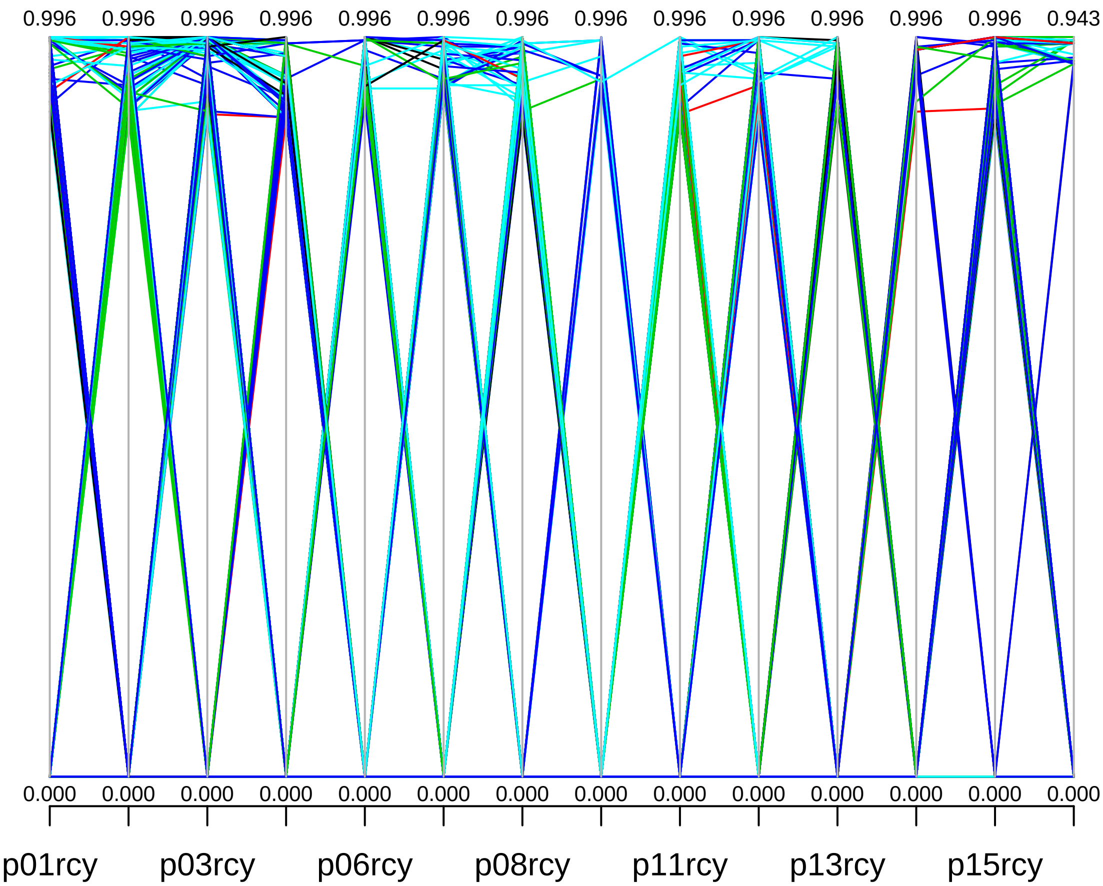
2-1.pdf

```
## Plotting the parallel coordinates.
recent.cols<- c ('p01rcy','p02rcy','p03rcy','p04rcy','p06rcy','p07rcy','p08rcy',
                 'p09rcy','p11rcy','p12rcy','p13rcy','p14rcy','p15rcy','p16rcy')
dmm.recency <- dmm.train[,recent.cols]

##Plot parallel co-ordinates using objective to understand the relation
# Between the recency.
parcoord(dmm.recency, col=dmm.train$Objective,var.label = TRUE)
```



2-2.pdf

```r
parcoord(dmm.recency, col=fit.km$cluster,var.label = TRUE)
```

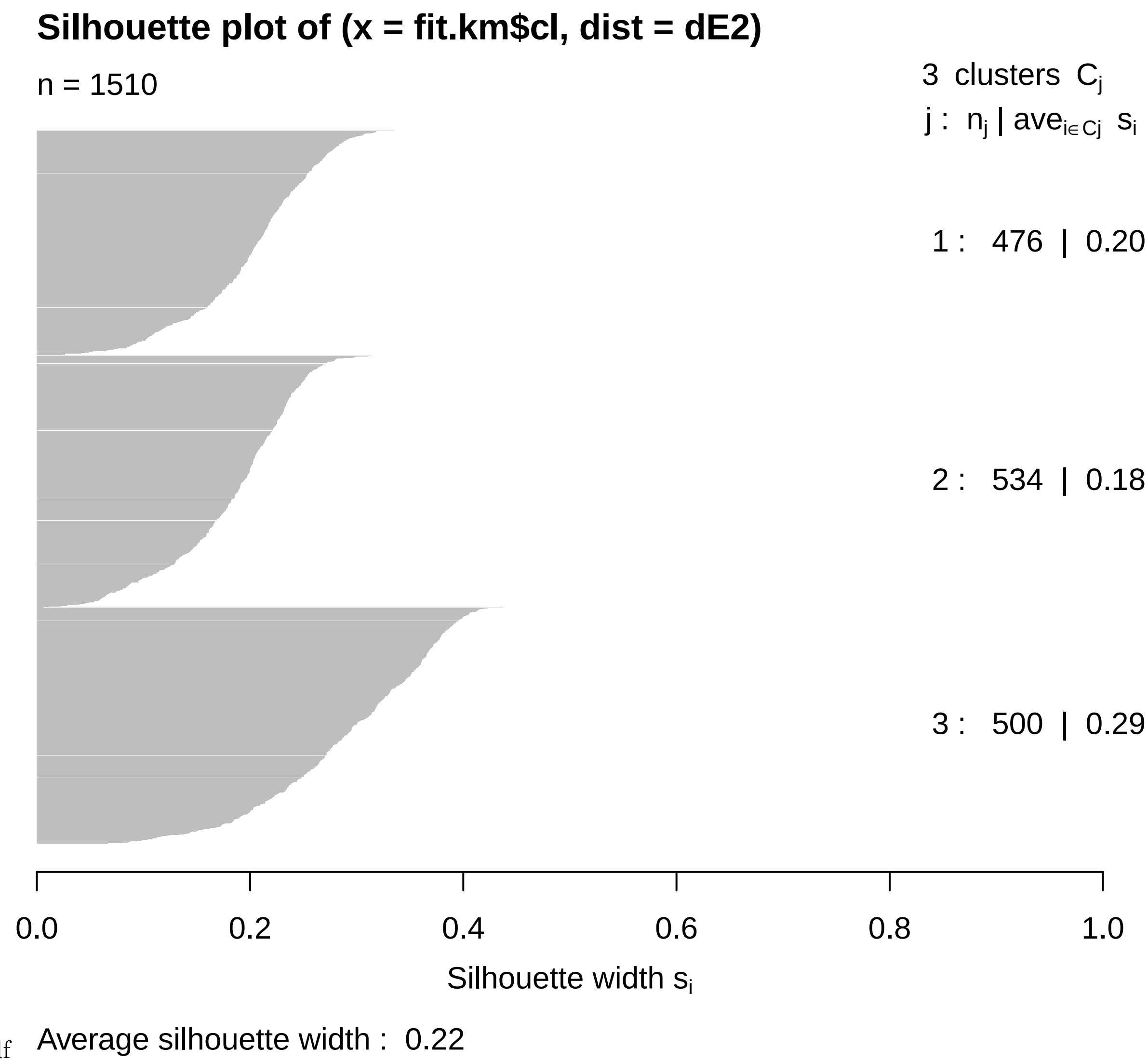


2-3.pdf

```
fit.km <- kmeans(dmm.train, 3, nstart=25)
## Plotting the silhouette
dissE <- daisy(dmm.train)
```

```
## Warning in daisy(dmm.train): binary variable(s) 1, 138, 140, 141, 142, 201
## treated as interval scaled
```

```
dE2   <- dissE^2
sk2   <- silhouette(fit.km$cl, dE2)
plot(sk2)
```



2-4.pdf

```
#3
fit.km$size
```

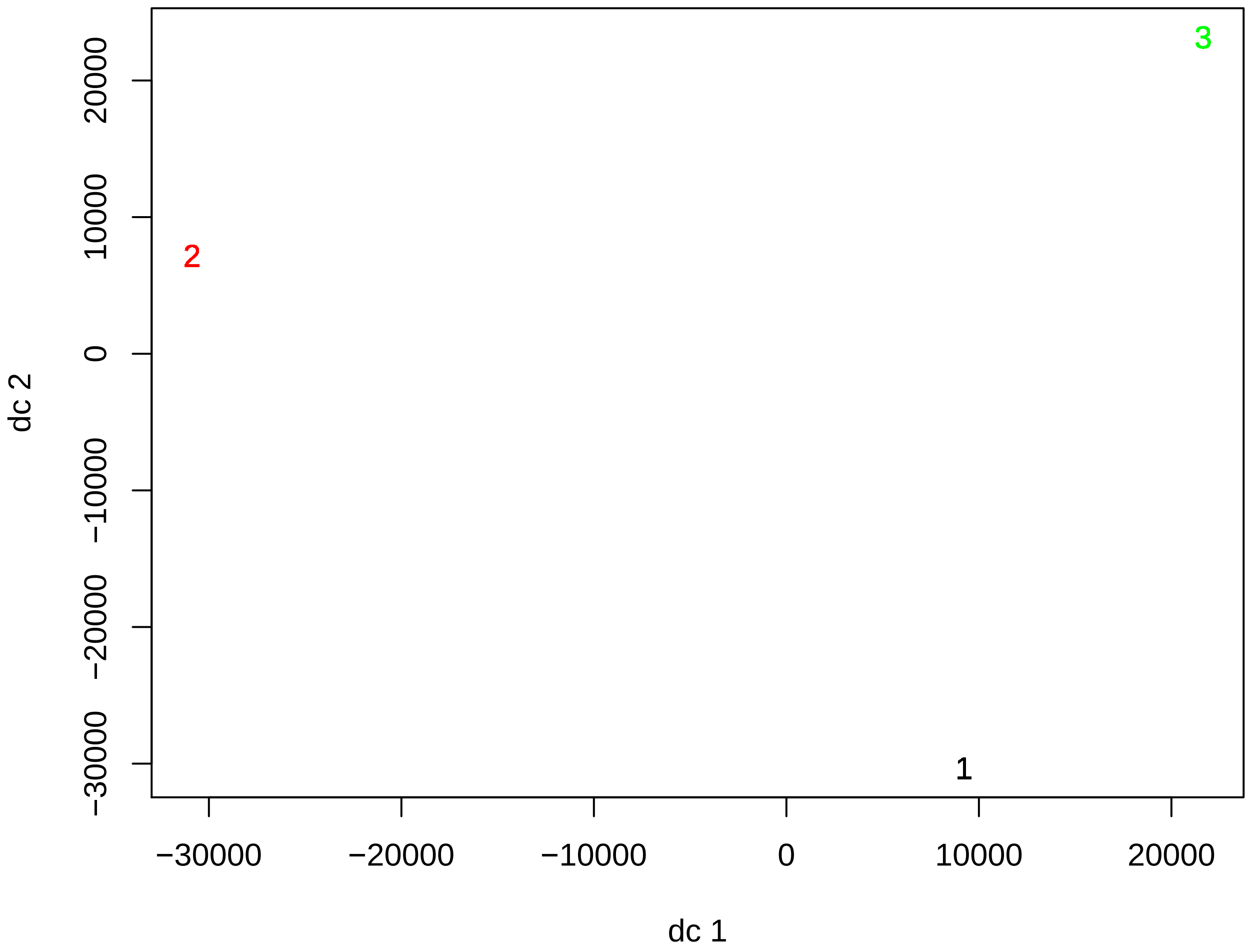
```
## [1] 476 534 500
```

Plots with high positive silhouette width values will have fit well. As visualised clearly, cluster 3 has higher silhouette widht and hence is better.

fit.km$centers

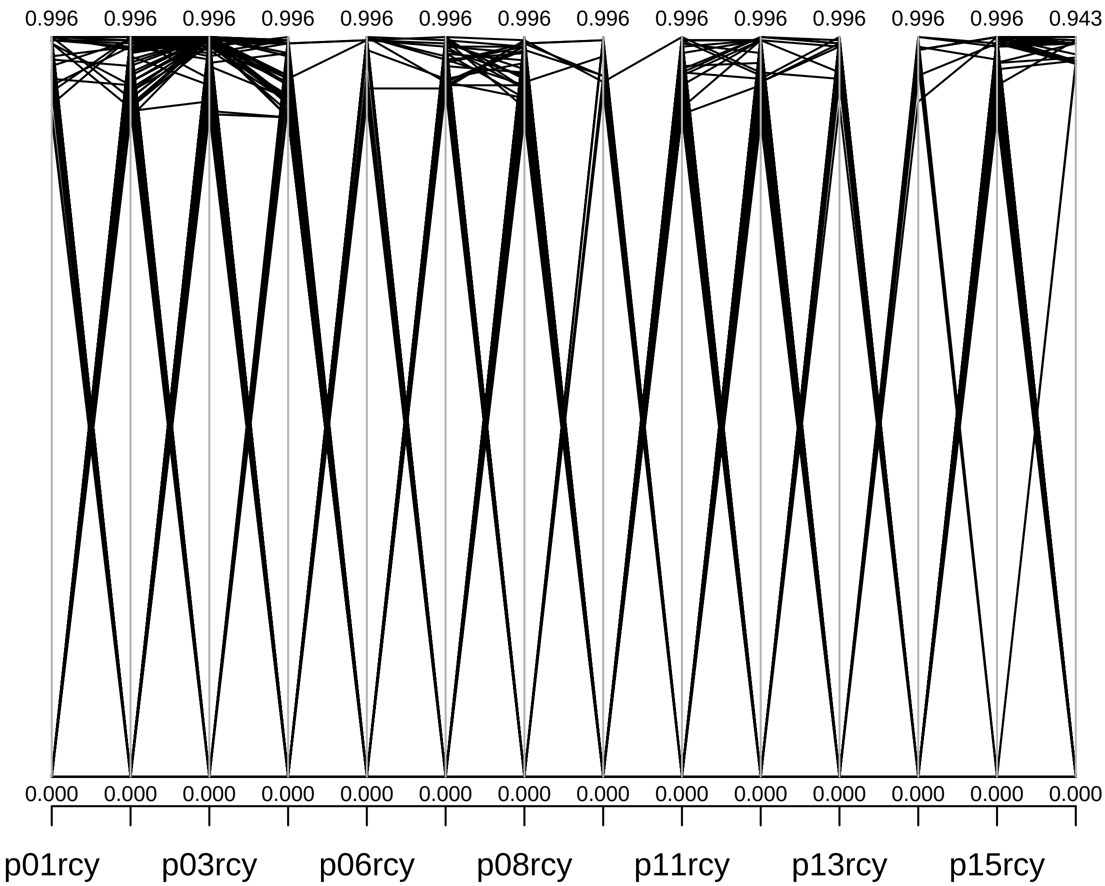
aggregate(dmm.train[-1], by=list(cluster=fit.km$cluster), mean)

```
plotcluster(dmm.train, fit.km$cluster)
```



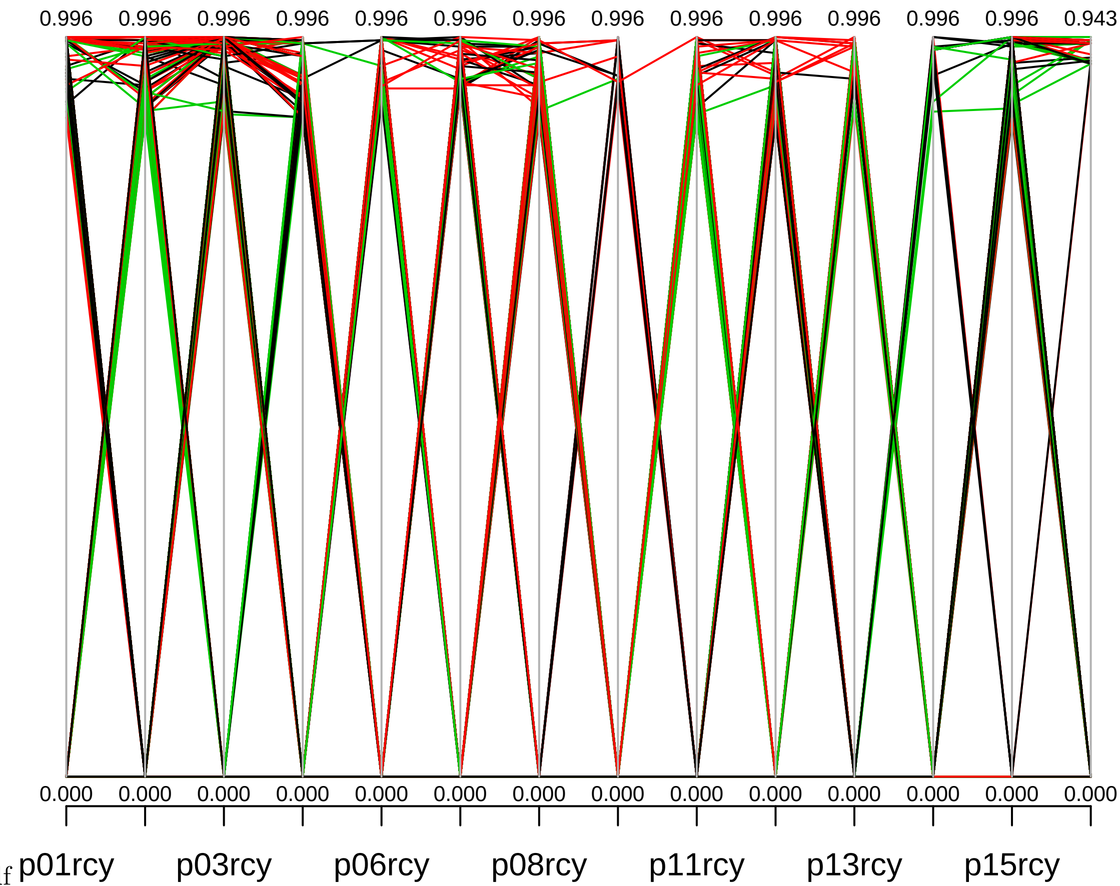
3-1.pdf

```
## Plotting the parallel coordinates.
recent.cols<- c ('p01rcy','p02rcy','p03rcy','p04rcy','p06rcy','p07rcy','p08rcy',
                 'p09rcy','p11rcy','p12rcy','p13rcy','p14rcy','p15rcy','p16rcy')
dmm.recency <- dmm.train[,recent.cols]

##Plot parallel co-ordinates usiing objective to understand the relation
# Between the recency.
parcoord(dmm.recency, col=dmm.train$Objective,var.label = TRUE)
```



3-2.pdf

```
parcoord(dmm.recency, col=fit.km$cluster,var.label = TRUE)
```



3-3.pdf

Correlation Analysis

In order to find correlation between the variables we perform analysis based on their correlation coffecient. To find this, we write the code:

```
matrixcor <- cor(dmm.train[,1:200])
diag(matrixcor) <- 0
upper.tri <- upper.tri(matrixcor)
matrixcor[lower.tri(matrixcor)] <- 0
xz <- as.data.frame(as.table(matrixcor))
names(xz) <- c("variable1", "variable2","Correlation")
head(xz[order(abs(xz$Correlation),decreasing=T),],n=10)
```

```
##       variable1 variable2 Correlation
## 28939     first    tenure   0.9997998
## 39597      tf94      tf95   0.9951427
## 25928  rlcathol rlrcathol   0.9947177
## 16080   ineflow  ineflowp   0.9930688
## 14837  mtfrench  hlfrench   0.9916916
## 24900  lfmtempl    pwmpop   0.9878138
## 38391      tf88      tf89   0.9864421
## 10437  mtfrench  etfrench   0.9858847
## 17487   inhhlow  inhhlowp   0.9853747
## 33165      tf37      tf38   0.9841510
```

When n=200 we could find correlation for all the variables. In order to check for the magnitude of the correlation large enough and thus is it really significant, we implement `cor.test` function and see for the `p-value` and `confidence interval`. The value `p<0.05` indicates that corelation is significant. And for the `confidence interval`, if `0` is in the interval then it is possible that there is no correlation.

```
cor.test(dmm.train$first, dmm.train$tenure, method="pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  dmm.train$first and dmm.train$tenure
## t = 1940.381, df = 1508, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9997785 0.9998190
## sample estimates:
##       cor
## 0.9997998
```

Testing for Independence of categorial variables:

For categorial variables, represnted by factors, we check them for correlation using the `chi-squared` test. We could use `table` function to get a contigency table from the two factors. I we perform the `summary()` function, we get `chi-squared` test output. Again, we check for `p-value`. If its less then the threshold value which is 0.05, we say that the variables are correlated.

```
summary(table(dmm_data$highincome, dmm_data$lowincome))
```

```
## Number of cases in table: 2158
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 35.82, df = 1, p-value = 2.169e-09
```

We could perform `chisq.test()` to perform the same.

After performing the same for 5 categorial variables, we get `v140 & v139` (highincome & lowincome), `v200 & v141` (gender & gender2), `v200 & v137` (gender & gender1) and `v141 & v137` (gender2 & gender1) correlated with each other. So the gender variable male, female and unkown are combined into one variable i.e; gender variable.

To combine gender variables:

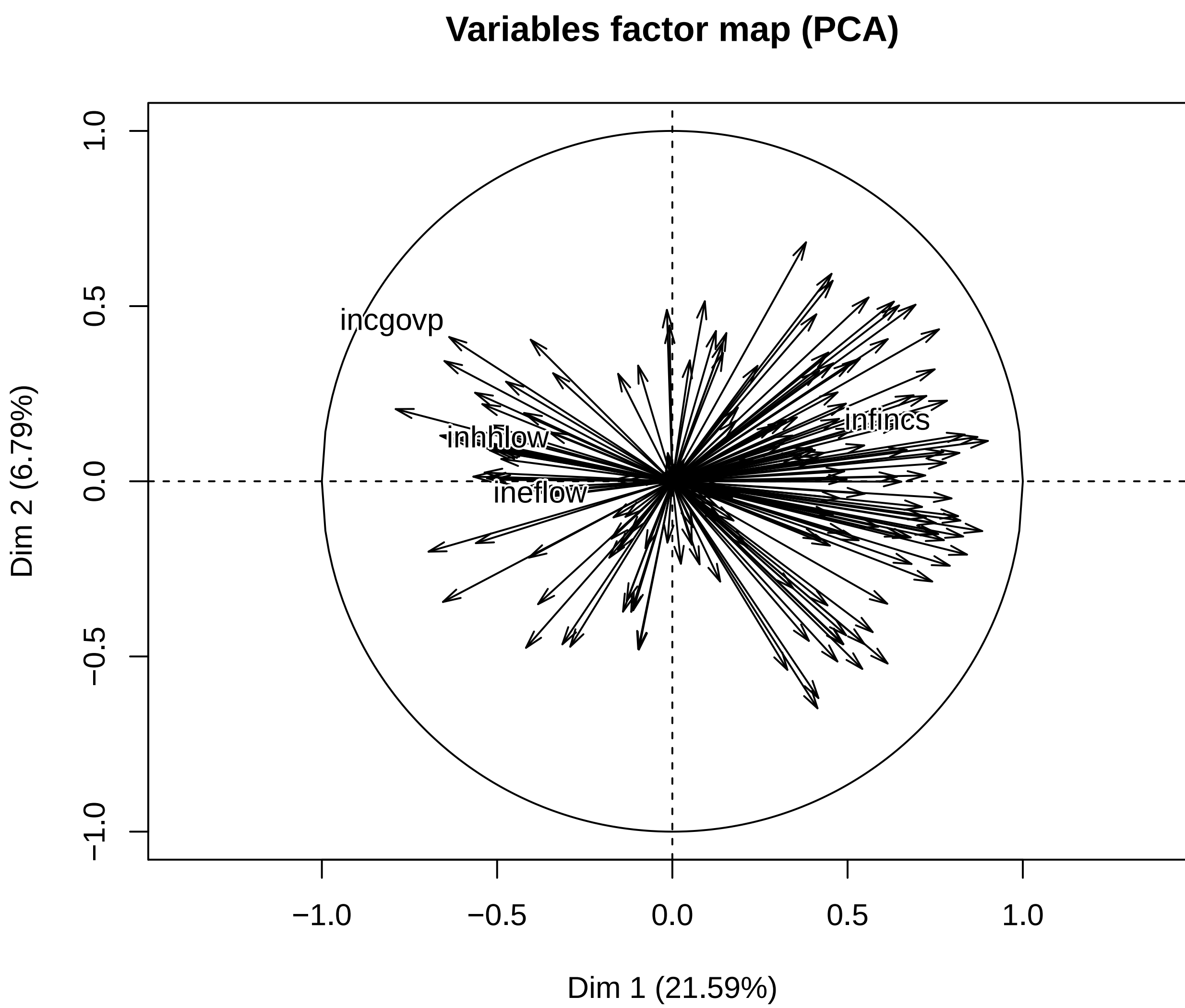
```
# Adding combined variable to the training data
m<-gsub("1", "1", dmm.train$gender1)
f<-gsub("1", "2", dmm.train$gender2)
u<-gsub("1", "3", dmm.train$gender)
m<-cbind(m)
f<-cbind(f)
u<-cbind(u)
m<-as.numeric(as.character(m[,1]))
m<-cbind(m)
f<-as.numeric(as.character(f[,1]))
f<-cbind(f)
u<-as.numeric(as.character(u[,1]))
```

```r
u<-cbind(u)
genders<-m+f+u
dmm.train$genders<-as.factor(genders)

# Adding combined variable to the test data
m<-gsub("1", "1", dmm.test$gender1)
f<-gsub("1", "2", dmm.test$gender2)
u<-gsub("1", "3", dmm.test$gender)
m<-cbind(m)
f<-cbind(f)
u<-cbind(u)
m<-as.numeric(as.character(m[,1]))
m<-cbind(m)
f<-as.numeric(as.character(f[,1]))
f<-cbind(f)
u<-as.numeric(as.character(u[,1]))
u<-cbind(u)
genders<-m+f+u
dmm.test$genders<-as.factor(genders)

m<-gsub("1", "1", dmm.validate$gender1)
f<-gsub("1", "2", dmm.validate$gender2)
u<-gsub("1", "3", dmm.validate$gender)
m<-cbind(m)
f<-cbind(f)
u<-cbind(u)
m<-as.numeric(as.character(m[,1]))
m<-cbind(m)
f<-as.numeric(as.character(f[,1]))
f<-cbind(f)
u<-as.numeric(as.character(u[,1]))
u<-cbind(u)
genders<-m+f+u
dmm.validate$genders<-as.factor(genders)
```

PCA

We use Principal Component Analysis to further reduce the data. We analyse for the overlapping variables in the PCA factor map. For instance, when we explore for the highly correlated variables we have written below code and get `ineflow` variables highly correlated one to be reduced:

```r
vars <- c('ineflow','incgovp','infincs','inhhlow')
## For better graphics
print(plot(res, choix = 'var',shadow = TRUE,select = vars,unselect = 0))
```



components-1.pdf

```
## $x
## [1] -0.8003055 -0.3772728  0.6134224 -0.4981274
##
## $y
## [1]  0.45369637 -0.02983587  0.17815932  0.12719694
```

Data Reduction

From the analysis done so far, we could reduce the dataset from 201 variables to 151 variables after deleting the below highly correlated variables:

To further reduce the dataset, we design `random forest` model to see top variables which are importand and ignoring least important variables. Modelling is done in the next section. When Random model was made on full dataset, after analysing the top variales which are significant by comparing the `Meandecreaseaccuracy` of the variables.

Hence, in total 128 variables were reduced.

```
# Deleting the correlated variables
delete.cols<-c("lfmtunemp","lfmtunemr","lfttempl","moy5mov","rlcathol","tf50",
"tf56","tf70","tf73","tf89","tf91","tf95","tf96","p16rcy","mtsingres",
```

```r
"etbritish","etfrench","fiinca","p05spend","hiinca","hlenglish",
"hlfrench","hlnonoff","improvres","imuk","incgovp","ineflow",
"ineflowp","inf30plus","inf7to15","infinca","infincm","infincs",
"inhhlow","inhhlowp","inm15to30","inm30plus","inm7to15","inmfemina",
"inminca","inmincm","inmincs","knenglish","knfren","lfmaempl",
"lfmaunemr","lfmtempl","gender1", "gender2", "gender", "tf80","highincome",
"dwminor", "mtmultlin", "fsllabf", "tf57", "p05trans", "hiu20", "mttagalog","
fslonepar", "hiincs", "mtspanish", "ocfteach", "tf55", "dwmajor", "fem40to44",
"dwmaint", "tf42", "p16spend", "ocmscieng", "mpshealth", "tf36", "fpshealth",
"p09tenure", "fp2child", "cftotmar", "fpshuman", "fi20to35","tf74", "tf27",
"cfhuswife", "sl9to13nc", "dw46to60", "p07rc", "ocmmanage","tf75", "slunivnd",
"mpssocial", "dwperroom", "lfttunemr", "dw86to91", "mpseng","lowincome",
"moy5intrn","tf35","ocfmanage","fp1child","mpscomm","p16trans","ndtgovser",
"ndtbusser","tf88","tf62","hi20to35","cfwchcom","slunivnc","hh2fam","nfamrel",
"tf65","p14tenure","lftaempl","moy1intep","fiu20","mtnengnon","ndtallind",
"mps","tf47","hh6ppers","moy5non","rlrcathol","tf71","tf76","tf58","tf101",
"fsmlabf","mtfrench","p16tenure","tf94")


## Deleting the columns.

dmm.train <- dmm.train[ , -which(names(dmm.train) %in% delete.cols)]
dmm.test <- dmm.test[ , -which(names(dmm.test) %in% delete.cols)]
validate.reduce <- dmm.validate[ , -which(names(dmm.validate) %in% delete.cols)]
```

Modelling and Evaluation:

Data Mining is an iterative process.

From the dataset reduced to 73 variables we make Four types of model. The data partition as done initially is 70/15/15. Crossvalidation used with 3 folds.

```r
set.seed(1234)
# set label name and predictors
labelName <- 'Objective'
predictors <- names(dmm.train)[names(dmm.train) != labelName]
dmm.train <- data.frame(lapply(dmm.train, as.numeric))
dmm.test <- data.frame(lapply(dmm.test, as.numeric))
dmm.validate <- data.frame(lapply(dmm.test, as.numeric))
library(caret)
require(pROC)
## create a caret control object to control the number of cross-validations performed
# Setting: adptive_cv as method
# Either the number of folds or number of resampling iterations = 3
# Don't save any summary Matrix from resample
var.control <- trainControl(method='cv', number=3, returnResamp='none')

# Final model 1 Nueral Network
model_nn <- train(dmm.train[,predictors], dmm.train[,labelName], method='nnet', trControl=var.control)
```

```
## # weights:  78
## initial  value 262.292316
## iter  10 value 179.442865
```

```
## iter  20 value 168.615598
## iter  30 value 164.501768
## iter  40 value 161.653907
## iter  50 value 160.261481
## iter  60 value 158.297515
## iter  70 value 152.113018
## iter  80 value 148.285183
## iter  90 value 147.403802
## iter 100 value 147.240593
## final  value 147.240593
## stopped after 100 iterations
## # weights:  232
## initial  value 259.362189
## iter  10 value 188.063713
## iter  20 value 158.817999
## iter  30 value 145.768844
## iter  40 value 132.308292
## iter  50 value 121.435431
## iter  60 value 116.740376
## iter  70 value 116.310830
## iter  80 value 116.301858
## iter  90 value 116.301685
## final  value 116.301394
## converged
## # weights:  386
## initial  value 252.378922
## iter  10 value 180.404956
## iter  20 value 157.426403
## iter  30 value 139.466113
## iter  40 value 124.246703
## iter  50 value 113.478837
## iter  60 value 103.759734
## iter  70 value 98.468018
## iter  80 value 95.324617
## iter  90 value 94.012863
## iter 100 value 93.375174
## final  value 93.375174
## stopped after 100 iterations
## # weights:  78
## initial  value 255.465279
## iter  10 value 188.980856
## iter  20 value 180.656771
## iter  30 value 179.477555
## iter  40 value 179.150792
## iter  50 value 179.083589
## final  value 179.083574
## converged
## # weights:  232
## initial  value 275.503406
## iter  10 value 200.305637
## iter  20 value 177.440050
## iter  30 value 168.502421
## iter  40 value 167.062405
## iter  50 value 166.204540
```

```
## iter  60 value 165.734676
## iter  70 value 165.217447
## iter  80 value 165.062628
## iter  90 value 164.976553
## iter 100 value 164.941372
## final  value 164.941372
## stopped after 100 iterations
## # weights:  386
## initial  value 256.648889
## iter  10 value 195.024710
## iter  20 value 176.053474
## iter  30 value 170.165778
## iter  40 value 166.358719
## iter  50 value 164.293791
## iter  60 value 163.276333
## iter  70 value 162.667254
## iter  80 value 162.241062
## iter  90 value 162.026938
## iter 100 value 161.956513
## final  value 161.956513
## stopped after 100 iterations
## # weights:  78
## initial  value 272.735434
## iter  10 value 231.597826
## iter  20 value 189.566855
## iter  30 value 181.032002
## iter  40 value 175.456284
## iter  50 value 171.528365
## iter  60 value 170.075225
## iter  70 value 169.574619
## iter  80 value 169.412519
## iter  90 value 169.314654
## iter 100 value 169.220637
## final  value 169.220637
## stopped after 100 iterations
## # weights:  232
## initial  value 258.019137
## iter  10 value 193.102908
## iter  20 value 158.040227
## iter  30 value 141.735899
## iter  40 value 131.477165
## iter  50 value 125.038358
## iter  60 value 122.717923
## iter  70 value 121.307844
## iter  80 value 119.712290
## iter  90 value 118.670676
## iter 100 value 117.928677
## final  value 117.928677
## stopped after 100 iterations
## # weights:  386
## initial  value 255.175906
## iter  10 value 179.632050
## iter  20 value 151.440631
## iter  30 value 127.732197
```

```
## iter  40 value 106.189606
## iter  50 value 95.927457
## iter  60 value 91.731797
## iter  70 value 89.892220
## iter  80 value 88.850583
## iter  90 value 87.744376
## iter 100 value 85.880094
## final  value 85.880094
## stopped after 100 iterations
## # weights:  78
## initial  value 272.917456
## iter  10 value 191.430013
## iter  20 value 172.096267
## iter  30 value 167.112407
## iter  40 value 160.058293
## iter  50 value 155.227195
## iter  60 value 154.669241
## final  value 154.668751
## converged
## # weights:  232
## initial  value 285.336056
## iter  10 value 217.163593
## iter  20 value 175.573609
## iter  30 value 163.831979
## iter  40 value 148.484293
## iter  50 value 136.227724
## iter  60 value 127.416784
## iter  70 value 120.637409
## iter  80 value 118.467273
## iter  90 value 118.059549
## iter 100 value 117.859905
## final  value 117.859905
## stopped after 100 iterations
## # weights:  386
## initial  value 351.208947
## iter  10 value 179.678241
## iter  20 value 168.984660
## iter  30 value 148.817893
## iter  40 value 129.678999
## iter  50 value 114.271545
## iter  60 value 110.165463
## iter  70 value 108.967142
## iter  80 value 108.388155
## iter  90 value 108.366343
## iter 100 value 108.363033
## final  value 108.363033
## stopped after 100 iterations
## # weights:  78
## initial  value 251.997603
## iter  10 value 188.822070
## iter  20 value 179.092043
## iter  30 value 178.483029
## iter  40 value 178.442151
## iter  50 value 178.430253
```

```
## final  value 178.430248
## converged
## # weights:  232
## initial  value 290.550277
## iter  10 value 209.427662
## iter  20 value 182.074645
## iter  30 value 178.231642
## iter  40 value 176.374652
## iter  50 value 173.395067
## iter  60 value 172.387060
## iter  70 value 171.838049
## iter  80 value 171.284772
## iter  90 value 171.085322
## iter 100 value 171.038950
## final  value 171.038950
## stopped after 100 iterations
## # weights:  386
## initial  value 282.317340
## iter  10 value 209.120534
## iter  20 value 178.025029
## iter  30 value 168.429247
## iter  40 value 165.668283
## iter  50 value 165.392975
## iter  60 value 164.685050
## iter  70 value 163.817107
## iter  80 value 162.874631
## iter  90 value 162.573958
## iter 100 value 162.121605
## final  value 162.121605
## stopped after 100 iterations
## # weights:  78
## initial  value 256.319534
## iter  10 value 184.873866
## iter  20 value 172.936168
## iter  30 value 167.158290
## iter  40 value 164.932651
## iter  50 value 163.210481
## iter  60 value 162.432336
## iter  70 value 161.528891
## iter  80 value 160.140486
## iter  90 value 158.749163
## iter 100 value 158.020539
## final  value 158.020539
## stopped after 100 iterations
## # weights:  232
## initial  value 261.809907
## iter  10 value 172.483721
## iter  20 value 154.970112
## iter  30 value 140.967807
## iter  40 value 125.214835
## iter  50 value 116.664011
## iter  60 value 114.172198
## iter  70 value 112.776044
## iter  80 value 111.959385
```

```
## iter  90 value 111.200710
## iter 100 value 110.805864
## final  value 110.805864
## stopped after 100 iterations
## # weights:  386
## initial  value 256.011987
## iter  10 value 185.797214
## iter  20 value 158.807931
## iter  30 value 134.477931
## iter  40 value 118.719290
## iter  50 value 99.715239
## iter  60 value 92.224348
## iter  70 value 87.180169
## iter  80 value 84.428780
## iter  90 value 82.417894
## iter 100 value 80.587610
## final  value 80.587610
## stopped after 100 iterations
## # weights:  78
## initial  value 289.983798
## iter  10 value 236.079850
## iter  20 value 185.885639
## iter  30 value 173.364582
## iter  40 value 166.870598
## iter  50 value 162.510180
## iter  60 value 154.180927
## iter  70 value 149.853921
## iter  80 value 148.937646
## iter  90 value 148.888745
## final  value 148.888714
## converged
## # weights:  232
## initial  value 283.718179
## iter  10 value 251.702933
## iter  20 value 187.654895
## iter  30 value 175.061398
## iter  40 value 151.558474
## iter  50 value 137.341193
## iter  60 value 124.385738
## iter  70 value 118.162956
## iter  80 value 117.382085
## iter  90 value 117.348987
## iter 100 value 117.348683
## final  value 117.348683
## stopped after 100 iterations
## # weights:  386
## initial  value 292.211583
## iter  10 value 193.780939
## iter  20 value 169.994275
## iter  30 value 151.640994
## iter  40 value 142.264265
## iter  50 value 130.175294
## iter  60 value 115.628290
## iter  70 value 110.265956
```

```
## iter  80 value 109.094693
## iter  90 value 108.993752
## iter 100 value 108.967737
## final  value 108.967737
## stopped after 100 iterations
## # weights:  78
## initial  value 277.772076
## iter  10 value 194.267082
## iter  20 value 185.452323
## iter  30 value 180.700417
## iter  40 value 180.267209
## iter  50 value 180.114989
## iter  60 value 180.104678
## iter  60 value 180.104676
## iter  60 value 180.104676
## final  value 180.104676
## converged
## # weights:  232
## initial  value 256.129153
## iter  10 value 199.294019
## iter  20 value 182.675374
## iter  30 value 172.293457
## iter  40 value 166.344337
## iter  50 value 165.631210
## iter  60 value 165.577875
## iter  70 value 165.459335
## iter  80 value 165.382056
## iter  90 value 164.781610
## iter 100 value 163.801459
## final  value 163.801459
## stopped after 100 iterations
## # weights:  386
## initial  value 267.682522
## iter  10 value 205.173638
## iter  20 value 174.007822
## iter  30 value 166.601829
## iter  40 value 163.005832
## iter  50 value 161.263113
## iter  60 value 160.605368
## iter  70 value 159.836678
## iter  80 value 159.295079
## iter  90 value 158.771765
## iter 100 value 158.469535
## final  value 158.469535
## stopped after 100 iterations
## # weights:  78
## initial  value 261.500735
## iter  10 value 187.933613
## iter  20 value 172.088633
## iter  30 value 167.113415
## iter  40 value 163.768149
## iter  50 value 161.784476
## iter  60 value 160.164336
## iter  70 value 159.075932
```

```
## iter  80 value 158.368156
## iter  90 value 157.683564
## iter 100 value 157.224124
## final  value 157.224124
## stopped after 100 iterations
## # weights:  232
## initial  value 250.404220
## iter  10 value 179.298601
## iter  20 value 167.852766
## iter  30 value 164.470985
## iter  40 value 162.043002
## iter  50 value 154.945506
## iter  60 value 147.658440
## iter  70 value 141.934645
## iter  80 value 139.238074
## iter  90 value 135.136173
## iter 100 value 130.630552
## final  value 130.630552
## stopped after 100 iterations
## # weights:  386
## initial  value 258.465526
## iter  10 value 180.443033
## iter  20 value 154.766148
## iter  30 value 130.354067
## iter  40 value 113.666950
## iter  50 value 100.925581
## iter  60 value 96.778640
## iter  70 value 94.600405
## iter  80 value 93.630904
## iter  90 value 92.619500
## iter 100 value 91.422921
## final  value 91.422921
## stopped after 100 iterations
## # weights:  232
## initial  value 388.761262
## iter  10 value 285.802320
## iter  20 value 261.757584
## iter  30 value 252.533363
## iter  40 value 250.909165
## iter  50 value 250.578148
## iter  60 value 250.423216
## iter  70 value 250.361293
## iter  80 value 250.324304
## iter  90 value 250.312594
## iter 100 value 250.308313
## final  value 250.308313
## stopped after 100 iterations
```

```
pred_nn <-  (ifelse(predict(object=model_nn, dmm.test[,predictors])< 0.5,0,1))
auc_nn <- roc(dmm.test[,labelName], pred_nn)
table(dmm.test$Objective,pred_nn)
```

```
##    pred_nn
##       0   1
```

```
##    0 105  54
##    1  54 112
```

```r
acc_nn <- sum(dmm.test$Objective==pred_nn) / nrow(dmm.test) #Accuracy: 0.52
print(auc_nn$auc)  # Area under the curve: 0.5196636
```

```
## Area under the curve: 0.6675
```

```r
# benchmark model 2 Random Forest
model.rf <- train(dmm.train[,predictors], dmm.train[,labelName], method='rf', trControl=var.control)
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The
## response has five or fewer unique values.  Are you sure you want to do
## regression?
```

```
pred_rf <-  (ifelse(predict(object=model.rf, dmm.test[,predictors])< 0.5,0,1))
auc_rf <- roc(dmm.test[,labelName], pred_rf)
table(dmm.test$Objective,pred_rf)
```

```
##    pred_rf
##       0   1
##   0 109  50
##   1  39 127
```

```
acc_rf <- sum(dmm.test$Objective==pred_rf) / nrow(dmm.test) # Accuracy: 0.7630769231
print(auc_rf$auc)  # Area under the curve: 0.7654202
```

```
## Area under the curve: 0.7253
```

```
# benchmark model 3 Decision Tree
model.rpart <- train(dmm.train[,predictors], dmm.train[,labelName], method='rpart', trControl=var.contro
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
pred_rpart <-  (ifelse(predict(object=model.rpart, dmm.test[,predictors])< 0.5,0,1))
auc_rpart <- roc(dmm.test[,labelName], pred_rpart)
table(dmm.test$Objective,pred_rpart)
```

```
##    pred_rpart
##       0   1
##   0  82  77
##   1  22 144
```

```
acc_rpart <- sum(dmm.test$Objective==pred_rpart) / nrow(dmm.test) #Accuracy: 0.69538
print(auc_rpart$auc)  # Area under the curve: 0.7518754
```

```
## Area under the curve: 0.6916
```

```
accuracies <- c(acc_nn,acc_rpart)
auc <- c(auc_nn$auc,auc_rpart$auc)
fbmodels <- as.data.frame(cbind(accuracies,auc))
colnames(fbmodels) <- c('Accuracies','AUC')
rownames(fbmodels) <- c('FNN','FDT')
fbmodels[ "Model" ] <- rownames(fbmodels)
```

As basic models accuracy doens't improve beyond the 75% we are building the model with advanced algorithms like gbm (Generalized Boosted Model), tree bagging and ensemble.

```
## Using some advance models
# Tree Bag
model_treebag <- train(dmm.train[,predictors], dmm.train[,labelName],
                       method='treebag', trControl=var.control)
# Random Forest
```

```r
model_rff <- train(dmm.train[,predictors], dmm.train[,labelName],
                   method='rf', trControl=var.control)
# gbm
model_gbm <- train(dmm.train[,predictors], dmm.train[,labelName],
                   method='gbm', trControl=var.control)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2412             nan     0.1000    0.0095
##      2        0.2329             nan     0.1000    0.0079
##      3        0.2262             nan     0.1000    0.0067
##      4        0.2212             nan     0.1000    0.0053
##      5        0.2169             nan     0.1000    0.0045
##      6        0.2127             nan     0.1000    0.0031
##      7        0.2097             nan     0.1000    0.0027
##      8        0.2072             nan     0.1000    0.0017
##      9        0.2053             nan     0.1000    0.0006
##     10        0.2029             nan     0.1000    0.0019
##     20        0.1880             nan     0.1000    0.0008
##     40        0.1736             nan     0.1000    0.0001
##     60        0.1655             nan     0.1000    0.0001
##     80        0.1603             nan     0.1000   -0.0002
##    100        0.1557             nan     0.1000   -0.0001
##    120        0.1521             nan     0.1000   -0.0002
##    140        0.1489             nan     0.1000   -0.0001
##    150        0.1473             nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2383             nan     0.1000    0.0101
##      2        0.2282             nan     0.1000    0.0086
##      3        0.2192             nan     0.1000    0.0080
##      4        0.2120             nan     0.1000    0.0059
##      5        0.2066             nan     0.1000    0.0050
##      6        0.2024             nan     0.1000    0.0036
##      7        0.1982             nan     0.1000    0.0036
##      8        0.1945             nan     0.1000    0.0026
##      9        0.1922             nan     0.1000    0.0014
##     10        0.1893             nan     0.1000    0.0026
##     20        0.1725             nan     0.1000   -0.0001
##     40        0.1575             nan     0.1000   -0.0004
##     60        0.1464             nan     0.1000   -0.0010
##     80        0.1383             nan     0.1000   -0.0000
##    100        0.1309             nan     0.1000   -0.0005
##    120        0.1247             nan     0.1000   -0.0003
##    140        0.1192             nan     0.1000   -0.0006
##    150        0.1170             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2371             nan     0.1000    0.0119
##      2        0.2266             nan     0.1000    0.0094
##      3        0.2171             nan     0.1000    0.0083
##      4        0.2091             nan     0.1000    0.0065
##      5        0.2023             nan     0.1000    0.0052
##      6        0.1962             nan     0.1000    0.0041
```

```
##      7        0.1917             nan     0.1000    0.0037
##      8        0.1886             nan     0.1000    0.0013
##      9        0.1849             nan     0.1000    0.0028
##     10        0.1816             nan     0.1000    0.0023
##     20        0.1638             nan     0.1000   -0.0001
##     40        0.1459             nan     0.1000   -0.0005
##     60        0.1334             nan     0.1000   -0.0003
##     80        0.1240             nan     0.1000   -0.0006
##    100        0.1155             nan     0.1000   -0.0001
##    120        0.1073             nan     0.1000   -0.0005
##    140        0.1010             nan     0.1000   -0.0004
##    150        0.0981             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2405             nan     0.1000    0.0098
##      2        0.2329             nan     0.1000    0.0082
##      3        0.2260             nan     0.1000    0.0069
##      4        0.2202             nan     0.1000    0.0054
##      5        0.2153             nan     0.1000    0.0042
##      6        0.2119             nan     0.1000    0.0034
##      7        0.2093             nan     0.1000    0.0023
##      8        0.2065             nan     0.1000    0.0027
##      9        0.2040             nan     0.1000    0.0022
##     10        0.2016             nan     0.1000    0.0017
##     20        0.1863             nan     0.1000    0.0007
##     40        0.1716             nan     0.1000   -0.0001
##     60        0.1633             nan     0.1000   -0.0001
##     80        0.1581             nan     0.1000   -0.0006
##    100        0.1534             nan     0.1000   -0.0001
##    120        0.1495             nan     0.1000   -0.0001
##    140        0.1465             nan     0.1000   -0.0002
##    150        0.1449             nan     0.1000   -0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2379             nan     0.1000    0.0119
##      2        0.2269             nan     0.1000    0.0092
##      3        0.2185             nan     0.1000    0.0074
##      4        0.2120             nan     0.1000    0.0057
##      5        0.2062             nan     0.1000    0.0052
##      6        0.2014             nan     0.1000    0.0043
##      7        0.1969             nan     0.1000    0.0036
##      8        0.1947             nan     0.1000    0.0013
##      9        0.1917             nan     0.1000    0.0021
##     10        0.1901             nan     0.1000    0.0001
##     20        0.1724             nan     0.1000    0.0002
##     40        0.1557             nan     0.1000   -0.0002
##     60        0.1459             nan     0.1000   -0.0003
##     80        0.1380             nan     0.1000   -0.0002
##    100        0.1306             nan     0.1000   -0.0003
##    120        0.1243             nan     0.1000   -0.0007
##    140        0.1189             nan     0.1000   -0.0001
##    150        0.1171             nan     0.1000   -0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
```

```
##      1        0.2361             nan     0.1000    0.0123
##      2        0.2252             nan     0.1000    0.0088
##      3        0.2157             nan     0.1000    0.0081
##      4        0.2074             nan     0.1000    0.0066
##      5        0.2010             nan     0.1000    0.0054
##      6        0.1954             nan     0.1000    0.0042
##      7        0.1911             nan     0.1000    0.0035
##      8        0.1875             nan     0.1000    0.0023
##      9        0.1838             nan     0.1000    0.0027
##     10        0.1807             nan     0.1000    0.0016
##     20        0.1610             nan     0.1000    0.0002
##     40        0.1430             nan     0.1000    0.0000
##     60        0.1310             nan     0.1000   -0.0004
##     80        0.1205             nan     0.1000   -0.0001
##    100        0.1117             nan     0.1000   -0.0007
##    120        0.1038             nan     0.1000   -0.0005
##    140        0.0970             nan     0.1000   -0.0003
##    150        0.0939             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2412             nan     0.1000    0.0090
##      2        0.2341             nan     0.1000    0.0069
##      3        0.2289             nan     0.1000    0.0057
##      4        0.2238             nan     0.1000    0.0045
##      5        0.2197             nan     0.1000    0.0038
##      6        0.2165             nan     0.1000    0.0027
##      7        0.2141             nan     0.1000    0.0021
##      8        0.2116             nan     0.1000    0.0022
##      9        0.2095             nan     0.1000    0.0015
##     10        0.2083             nan     0.1000    0.0001
##     20        0.1945             nan     0.1000    0.0003
##     40        0.1804             nan     0.1000   -0.0000
##     60        0.1734             nan     0.1000   -0.0003
##     80        0.1675             nan     0.1000   -0.0004
##    100        0.1630             nan     0.1000   -0.0002
##    120        0.1590             nan     0.1000   -0.0002
##    140        0.1559             nan     0.1000   -0.0002
##    150        0.1544             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2396             nan     0.1000    0.0109
##      2        0.2296             nan     0.1000    0.0087
##      3        0.2226             nan     0.1000    0.0066
##      4        0.2167             nan     0.1000    0.0054
##      5        0.2113             nan     0.1000    0.0047
##      6        0.2066             nan     0.1000    0.0037
##      7        0.2030             nan     0.1000    0.0027
##      8        0.2001             nan     0.1000    0.0018
##      9        0.1972             nan     0.1000    0.0023
##     10        0.1942             nan     0.1000    0.0017
##     20        0.1797             nan     0.1000    0.0004
##     40        0.1637             nan     0.1000    0.0001
##     60        0.1533             nan     0.1000   -0.0004
##     80        0.1448             nan     0.1000   -0.0000
```

```
##    100        0.1372             nan     0.1000   -0.0003
##    120        0.1316             nan     0.1000   -0.0003
##    140        0.1256             nan     0.1000   -0.0006
##    150        0.1231             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2378             nan     0.1000    0.0121
##      2        0.2275             nan     0.1000    0.0090
##      3        0.2199             nan     0.1000    0.0066
##      4        0.2133             nan     0.1000    0.0041
##      5        0.2065             nan     0.1000    0.0052
##      6        0.2017             nan     0.1000    0.0032
##      7        0.1973             nan     0.1000    0.0038
##      8        0.1936             nan     0.1000    0.0032
##      9        0.1904             nan     0.1000    0.0016
##     10        0.1878             nan     0.1000    0.0011
##     20        0.1699             nan     0.1000    0.0001
##     40        0.1514             nan     0.1000   -0.0007
##     60        0.1377             nan     0.1000   -0.0005
##     80        0.1270             nan     0.1000   -0.0002
##    100        0.1184             nan     0.1000   -0.0005
##    120        0.1097             nan     0.1000   -0.0005
##    140        0.1023             nan     0.1000   -0.0002
##    150        0.0990             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2389             nan     0.1000    0.0105
##      2        0.2300             nan     0.1000    0.0094
##      3        0.2217             nan     0.1000    0.0078
##      4        0.2152             nan     0.1000    0.0060
##      5        0.2098             nan     0.1000    0.0049
##      6        0.2055             nan     0.1000    0.0037
##      7        0.2015             nan     0.1000    0.0039
##      8        0.1983             nan     0.1000    0.0029
##      9        0.1954             nan     0.1000    0.0025
##     10        0.1932             nan     0.1000    0.0015
##     20        0.1782             nan     0.1000    0.0005
##     40        0.1649             nan     0.1000   -0.0003
##     60        0.1564             nan     0.1000   -0.0002
##     80        0.1498             nan     0.1000   -0.0003
##    100        0.1447             nan     0.1000   -0.0002
```

```
# Building the Ensemble model
final_blender_model <- train(dmm.validate[,predictors], dmm.validate[,labelName],
                             method='gbm', trControl=var.control)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2421             nan     0.1000    0.0053
##      2        0.2359             nan     0.1000    0.0054
##      3        0.2304             nan     0.1000    0.0036
##      4        0.2254             nan     0.1000    0.0027
##      5        0.2214             nan     0.1000    0.0003
##      6        0.2180             nan     0.1000    0.0012
##      7        0.2144             nan     0.1000    0.0029
```

```
##      8        0.2105             nan     0.1000    0.0019
##      9        0.2061             nan     0.1000    0.0013
##     10        0.2029             nan     0.1000    0.0004
##     20        0.1767             nan     0.1000    0.0004
##     40        0.1516             nan     0.1000   -0.0005
##     60        0.1317             nan     0.1000    0.0005
##     80        0.1196             nan     0.1000   -0.0002
##    100        0.1092             nan     0.1000   -0.0003
##    120        0.0988             nan     0.1000   -0.0006
##    140        0.0917             nan     0.1000   -0.0010
##    150        0.0886             nan     0.1000   -0.0017
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2406             nan     0.1000    0.0085
##      2        0.2308             nan     0.1000    0.0051
##      3        0.2221             nan     0.1000    0.0043
##      4        0.2138             nan     0.1000    0.0025
##      5        0.2078             nan     0.1000    0.0027
##      6        0.2028             nan     0.1000   -0.0005
##      7        0.1963             nan     0.1000    0.0028
##      8        0.1895             nan     0.1000    0.0040
##      9        0.1865             nan     0.1000   -0.0034
##     10        0.1819             nan     0.1000    0.0011
##     20        0.1449             nan     0.1000    0.0002
##     40        0.1090             nan     0.1000   -0.0013
##     60        0.0853             nan     0.1000   -0.0013
##     80        0.0698             nan     0.1000    0.0000
##    100        0.0583             nan     0.1000   -0.0010
##    120        0.0491             nan     0.1000   -0.0006
##    140        0.0418             nan     0.1000   -0.0003
##    150        0.0384             nan     0.1000   -0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2396             nan     0.1000    0.0048
##      2        0.2272             nan     0.1000    0.0085
##      3        0.2138             nan     0.1000    0.0099
##      4        0.2055             nan     0.1000    0.0025
##      5        0.1989             nan     0.1000    0.0018
##      6        0.1924             nan     0.1000    0.0012
##      7        0.1862             nan     0.1000    0.0006
##      8        0.1805             nan     0.1000    0.0020
##      9        0.1747             nan     0.1000    0.0024
##     10        0.1688             nan     0.1000    0.0005
##     20        0.1251             nan     0.1000   -0.0002
##     40        0.0846             nan     0.1000   -0.0021
##     60        0.0594             nan     0.1000   -0.0005
##     80        0.0433             nan     0.1000   -0.0006
##    100        0.0335             nan     0.1000   -0.0003
##    120        0.0264             nan     0.1000   -0.0005
##    140        0.0205             nan     0.1000   -0.0001
##    150        0.0177             nan     0.1000   -0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2390             nan     0.1000    0.0099
```

```
##      2        0.2355             nan     0.1000    0.0001
##      3        0.2282             nan     0.1000    0.0076
##      4        0.2213             nan     0.1000    0.0064
##      5        0.2149             nan     0.1000    0.0021
##      6        0.2106             nan     0.1000    0.0018
##      7        0.2070             nan     0.1000    0.0022
##      8        0.2018             nan     0.1000    0.0042
##      9        0.1974             nan     0.1000    0.0024
##     10        0.1952             nan     0.1000    0.0002
##     20        0.1733             nan     0.1000   -0.0010
##     40        0.1469             nan     0.1000   -0.0026
##     60        0.1315             nan     0.1000   -0.0001
##     80        0.1189             nan     0.1000   -0.0001
##    100        0.1072             nan     0.1000   -0.0014
##    120        0.0990             nan     0.1000   -0.0004
##    140        0.0905             nan     0.1000   -0.0005
##    150        0.0870             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2353             nan     0.1000    0.0103
##      2        0.2253             nan     0.1000    0.0063
##      3        0.2174             nan     0.1000    0.0056
##      4        0.2098             nan     0.1000    0.0044
##      5        0.2031             nan     0.1000    0.0049
##      6        0.1979             nan     0.1000    0.0031
##      7        0.1910             nan     0.1000    0.0051
##      8        0.1848             nan     0.1000    0.0035
##      9        0.1798             nan     0.1000    0.0022
##     10        0.1753             nan     0.1000    0.0004
##     20        0.1423             nan     0.1000    0.0003
##     40        0.1047             nan     0.1000    0.0005
##     60        0.0822             nan     0.1000   -0.0008
##     80        0.0667             nan     0.1000   -0.0010
##    100        0.0545             nan     0.1000   -0.0005
##    120        0.0462             nan     0.1000   -0.0004
##    140        0.0397             nan     0.1000   -0.0005
##    150        0.0365             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2331             nan     0.1000    0.0081
##      2        0.2196             nan     0.1000    0.0111
##      3        0.2108             nan     0.1000    0.0049
##      4        0.1991             nan     0.1000    0.0038
##      5        0.1927             nan     0.1000    0.0004
##      6        0.1838             nan     0.1000    0.0041
##      7        0.1775             nan     0.1000    0.0005
##      8        0.1736             nan     0.1000   -0.0015
##      9        0.1650             nan     0.1000    0.0031
##     10        0.1586             nan     0.1000    0.0046
##     20        0.1210             nan     0.1000    0.0002
##     40        0.0806             nan     0.1000   -0.0014
##     60        0.0601             nan     0.1000   -0.0002
##     80        0.0458             nan     0.1000   -0.0011
##    100        0.0356             nan     0.1000   -0.0006
```

```
##    120        0.0275             nan     0.1000   -0.0002
##    140        0.0212             nan     0.1000   -0.0003
##    150        0.0191             nan     0.1000   -0.0006
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2423             nan     0.1000    0.0078
##      2        0.2373             nan     0.1000    0.0030
##      3        0.2335             nan     0.1000    0.0014
##      4        0.2273             nan     0.1000    0.0058
##      5        0.2216             nan     0.1000    0.0051
##      6        0.2189             nan     0.1000    0.0011
##      7        0.2144             nan     0.1000    0.0010
##      8        0.2103             nan     0.1000    0.0022
##      9        0.2050             nan     0.1000    0.0027
##     10        0.2029             nan     0.1000    0.0005
##     20        0.1817             nan     0.1000    0.0006
##     40        0.1571             nan     0.1000   -0.0019
##     60        0.1368             nan     0.1000   -0.0005
##     80        0.1229             nan     0.1000   -0.0010
##    100        0.1117             nan     0.1000   -0.0018
##    120        0.1024             nan     0.1000   -0.0012
##    140        0.0957             nan     0.1000   -0.0002
##    150        0.0922             nan     0.1000   -0.0007
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2395             nan     0.1000    0.0063
##      2        0.2309             nan     0.1000    0.0038
##      3        0.2261             nan     0.1000   -0.0004
##      4        0.2211             nan     0.1000    0.0006
##      5        0.2158             nan     0.1000    0.0011
##      6        0.2127             nan     0.1000   -0.0029
##      7        0.2047             nan     0.1000    0.0027
##      8        0.2019             nan     0.1000   -0.0024
##      9        0.1963             nan     0.1000    0.0032
##     10        0.1925             nan     0.1000   -0.0002
##     20        0.1571             nan     0.1000   -0.0024
##     40        0.1209             nan     0.1000   -0.0012
##     60        0.0954             nan     0.1000   -0.0003
##     80        0.0796             nan     0.1000   -0.0015
##    100        0.0671             nan     0.1000   -0.0005
##    120        0.0555             nan     0.1000   -0.0013
##    140        0.0474             nan     0.1000   -0.0005
##    150        0.0440             nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2393             nan     0.1000    0.0040
##      2        0.2238             nan     0.1000    0.0146
##      3        0.2150             nan     0.1000    0.0061
##      4        0.2071             nan     0.1000    0.0041
##      5        0.1993             nan     0.1000    0.0016
##      6        0.1907             nan     0.1000    0.0044
##      7        0.1831             nan     0.1000    0.0014
##      8        0.1775             nan     0.1000    0.0018
##      9        0.1743             nan     0.1000   -0.0020
```

```
##     10        0.1688             nan     0.1000    0.0024
##     20        0.1309             nan     0.1000   -0.0038
##     40        0.0890             nan     0.1000   -0.0020
##     60        0.0672             nan     0.1000   -0.0009
##     80        0.0509             nan     0.1000   -0.0009
##    100        0.0406             nan     0.1000   -0.0008
##    120        0.0319             nan     0.1000   -0.0007
##    140        0.0258             nan     0.1000   -0.0006
##    150        0.0232             nan     0.1000   -0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.2416             nan     0.1000    0.0080
##      2        0.2347             nan     0.1000    0.0060
##      3        0.2293             nan     0.1000    0.0022
##      4        0.2246             nan     0.1000    0.0023
##      5        0.2218             nan     0.1000    0.0007
##      6        0.2186             nan     0.1000   -0.0008
##      7        0.2138             nan     0.1000    0.0048
##      8        0.2110             nan     0.1000    0.0014
##      9        0.2087             nan     0.1000    0.0011
##     10        0.2060             nan     0.1000    0.0007
##     20        0.1841             nan     0.1000    0.0001
##     40        0.1597             nan     0.1000    0.0001
##     60        0.1444             nan     0.1000   -0.0002
##     80        0.1321             nan     0.1000   -0.0009
##    100        0.1233             nan     0.1000   -0.0003
```

```
## Validating the model
dmm.validate$gbm_PROB <- predict(object=model_gbm, dmm.validate[,predictors])
dmm.validate$rf_PROB <- predict(object=model_rff, dmm.validate[,predictors])
dmm.validate$treebag_PROB <- predict(object=model_treebag, dmm.validate[,predictors])
## Doing final predicions
dmm.test$gbm_PROB <- (ifelse(predict(object=model_gbm, dmm.test[,predictors])<0.5,0,1))
dmm.test$rf_PROB <- (ifelse(predict(object=model_rff, dmm.test[,predictors])<0.5,0,1))
dmm.test$treebag_PROB <- (ifelse(predict(object=model_treebag, dmm.test[,predictors])<0.5,0,1))

# see how each individual model performed on its own
acc_gbm <- sum(dmm.test$Objective==dmm.test$gbm_PROB) / nrow(dmm.test) # Accuracy:
auc_gbm <- roc(dmm.test[,labelName], dmm.test$gbm_PROB )
print(auc_gbm$auc)
```

```
## Area under the curve: 0.7286
```

```
acc_rff <- sum(dmm.test$Objective== dmm.test$rf_PROB) / nrow(dmm.test) # Accuracy:
auc_rff <- roc(dmm.test[,labelName], dmm.test$rf_PROB )
print(auc_rff$auc) # Area under the curve:
```

```
## Area under the curve: 0.7252
```

```
acc_tree <- sum(dmm.test$Objective== dmm.test$treebag_PROB) / nrow(dmm.test) # Accuracy:
auc_tree <- roc(dmm.test[,labelName], dmm.test$treebag_PROB )
print(auc_tree$auc)
```

```
## Area under the curve: 0.7223
```

```r
# run a final model to blend all the probabilities together
predictors <- names(dmm.validate)[names(dmm.validate) != labelName]

# See final prediction and AUC of blended ensemble
preds_ens <- predict(object=final_blender_model, dmm.test[,predictors])
auc_ens <- roc(dmm.test[,labelName], preds_ens)
preds_ens
```

```
##   [1]  0.648836380  0.926433308  0.885172181  0.785216619  0.363387439
##   [6]  0.038134473  0.617036472  0.639208074 -0.143324600  0.378172357
##  [11]  0.731273900  0.271830355  1.153179351  0.852608099  0.305477918
##  [16]  0.582469142  0.187610484  1.026217128  0.688250237  0.398620802
##  [21]  0.409045405 -0.134540224  0.635108868  0.720195230  0.762263831
##  [26]  0.083684614  0.087493178  0.861076440 -0.073093861  0.630572032
##  [31]  0.615611803  0.146311241  0.827401983  0.910729869  0.861942839
##  [36]  0.927409359  0.451899499 -0.007638585  0.820973972  0.492316322
##  [41]  0.724342355  0.620576490  0.593510283  0.135010315  0.907323550
##  [46]  0.279071900  0.793471183  0.912641257  0.456602735  0.263376836
##  [51]  0.747551380  0.242474489  0.231133275  0.598340013  0.194382854
##  [56]  0.690959192  0.672354339  0.056115215  0.404374319  0.304340518
##  [61]  0.757341501  0.413462405  0.592280082  0.491080334  0.270359766
##  [66]  0.770936385  0.849113182  0.695080328  0.313218203  0.751673458
##  [71]  0.711195845  0.393611509  0.623871643  0.785283878  0.461013956
##  [76]  0.588281270  0.540198767  0.668348373  0.353513125  0.708090135
##  [81]  0.550144748  0.386900045  0.641371724  0.442051648  0.597540933
##  [86]  0.488752699  0.579075241  0.369962821  0.964872853  0.958802553
##  [91]  0.839492113  0.299716380  0.680539412  0.611861035  0.862595374
##  [96]  0.376882085  1.097543480  0.800227605  0.288487822  0.257601574
## [101]  0.735581849  0.696950187  0.905822550  0.733780172  1.041473451
## [106]  0.807235876  0.814721298  0.795637820  0.245286737  0.787841749
## [111]  0.452248360  0.674171979  0.658730961  0.894105381  0.322638391
## [116]  0.189751443  0.491349548  0.836440803  0.499829635  0.648354948
## [121]  0.409532475  0.016077664  0.162142899  0.865310880  0.398881158
## [126]  0.220378749  0.066782292  0.120032320  0.653670782  0.683975256
## [131]  0.742501479  0.100944863  0.342148197  0.781515907  0.569366656
## [136]  0.709883894  0.223653960  0.656837769  0.827862181  0.716904111
## [141]  0.561539518  0.557980491  0.629689375  0.218987071  0.191519917
## [146]  0.288014500  0.129206655  0.778306149  0.547474097  0.912231683
## [151]  0.082517707  0.701283431  0.628486045  0.060519528  0.084803806
## [156]  0.370868343  0.343093920  0.283787296  0.568323387  0.535131186
## [161]  0.819487479  0.541675674  0.581324000  0.279490374  0.432032742
## [166]  0.277771166  0.180001072  0.153908031  0.501332871  0.640151932
## [171]  0.558032224  0.932411026  0.511510777  0.592027512  0.932493433
## [176]  0.927367172  0.669745998  0.300932670  0.810271156  0.803920142
## [181]  0.698644767  0.647671678  0.062498734  0.871459218  0.279344586
## [186]  0.802110927  0.731676141  0.520592338  0.171011458  0.800786150
## [191]  0.649179737  0.415482216  1.014374214  1.015613010  0.020982873
## [196]  0.361520530  0.670410258  0.077242002  0.223453154  0.538402865
## [201]  0.360365977  0.724849194  0.861145587  0.259601413  0.472907519
## [206]  0.814264039  0.767682490  0.575970963  0.877367044  1.075086711
## [211]  0.676937098  0.723436503  0.149976936  0.303248198  0.044313487
## [216]  0.485563010  0.203939394  0.362435065  0.240125701  0.045573747
```

```
## [221]  0.570887594  0.708405218  0.237061148 -0.241330646  0.105729080
## [226]  0.538410780  0.038177400  0.720049125  0.224093178  0.657302680
## [231]  0.256759609  0.168491686  0.425171346  0.594507436  0.209127052
## [236]  0.487048620  0.668201259  0.475775803  1.071559515  0.507644084
## [241]  0.548165357  0.760759126  0.530807623  0.295297334  0.722876693
## [246]  0.526995122  0.452853642  0.849172832  0.868840570  0.313887397
## [251]  0.478638452  0.472726279  0.414762080  0.650554210  0.091036570
## [256]  0.603062324  0.156558739 -0.104764539  0.013361865  0.159966272
## [261]  0.279920546  0.234470975  0.395506253  0.510707111  0.401869013
## [266]  0.399921289  0.388754935  0.108777808  0.207270228  0.326943999
## [271]  0.752080708  0.603503232  0.967643886  0.505004017  0.654661092
## [276]  0.728262243  0.644324057  0.627577928  0.107055635 -0.128372698
## [281]  0.731059400  0.376485564  0.661285315  0.355950100  0.901807205
## [286]  0.221697920  0.680625493  0.419615196  0.325234454  0.404788058
## [291]  0.628012364  0.167305429  0.559441665  0.104194716  0.610500486
## [296]  0.406535551  0.433384023  0.754495985  0.292699673  1.019700256
## [301]  0.514449458  0.453511732  0.132063977  0.533078806  0.372645001
## [306]  0.205665563  0.167694708  0.566102967  0.769580976  0.381752220
## [311]  0.849775568  0.647062515  0.177711756  0.404840304  0.232221832
## [316]  0.557071695  0.032330820  0.395247313  0.373482772  0.599597567
## [321]  0.324627011  0.596786378  0.528071289  0.788082576  0.408174835
```

```r
preds_ens <- (ifelse(preds_ens < 0.5,0,1))
table(dmm.test$Objective,preds_ens)
```

```
##    preds_ens
##       0   1
##   0 132  27
##   1  19 147
```

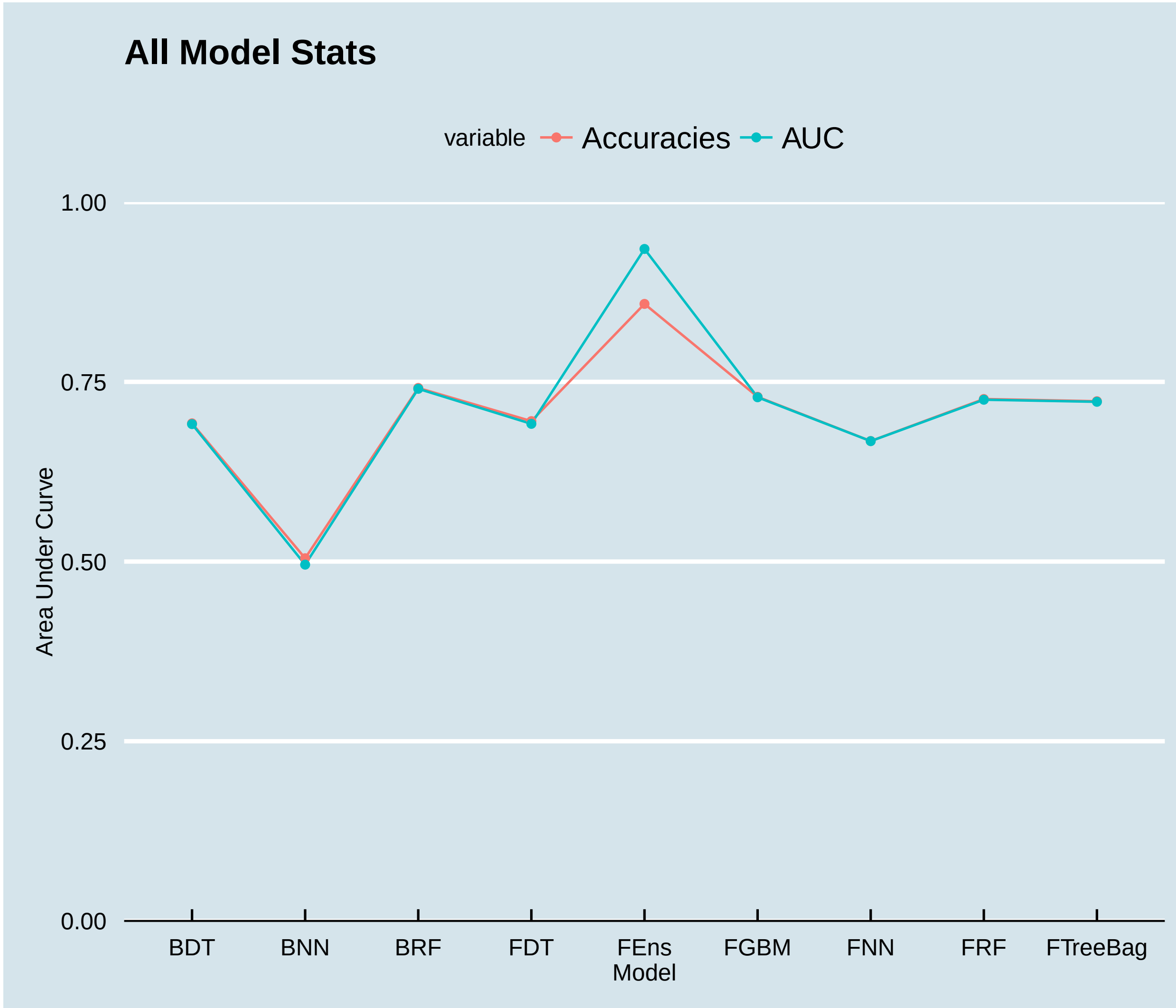
```r
acc_ens <- sum(dmm.test$Objective==preds_ens) / nrow(dmm.test) # Accuracy:0.7630769231
print(auc_ens$auc)  # Area under the curve:
```

```
## Area under the curve: 0.9349
```

```r
## Combining all the models into one Data Frame
accuracies <- c(acc_gbm,acc_rff,acc_tree,acc_ens)
auc <- c(auc_gbm$auc,auc_rff$auc,auc_tree$auc,auc_ens$auc)
famodels <- as.data.frame(cbind(accuracies,auc))
colnames(famodels) <- c('Accuracies','AUC')
rownames(famodels) <- c('FGBM','FRF','FTreeBag','FEns')

famodels[ "Model" ] <- rownames(famodels)
## Adding all the models
fmodels <- rbind(bmodels,fbmodels,famodels)
library(reshape)
fmodels.melt <- melt( fmodels, id.vars="Model", value.name="Accuracies", variable.name="AUC" )


## Table Showing Accuracy and AUC
fmodels
```

```
##          Accuracies       AUC    Model
## BNN       0.5046154 0.4956998      BNN
## BRF       0.7415385 0.7403577      BRF
## BDT       0.6923077 0.6912366      BDT
## FNN       0.6676923 0.6675381      FNN
## FDT       0.6953846 0.6915966      FDT
## FGBM      0.7292308 0.7285747     FGBM
## FRF       0.7261538 0.7251648      FRF
## FTreeBag  0.7230769 0.7222854 FTreeBag
## FEns      0.8584615 0.9349094     FEns
```

```r
require(ggplot2)
require(ggthemes)
ggplot(data=fmodels.melt, aes(x= Model, y=value, group = variable, colour = variable)) +
        theme_economist() +
        geom_line() +
        xlab("Model") +
        ylab("Area Under Curve") +
        ggtitle("All Model Stats") +
        coord_cartesian(ylim= c (0,1)) +
        geom_point()
```



Model-1.pdf

Conclusion:

Ensemble accuracy 85.85 with Area Under the Curve 93.49.

After applying the data mining process on dataset, we can say that for the better prediction of valuable customers who are likely to respond the model Ensemble out-performs from other models like random forests, neural networks, generic boost model, tree boost . If gender is unkown that individual is most likely not to respond to the mails. Recencies of Product plays important role while predicting the target customers like customer buying products three, fifteen and seventeen. Also person having higher income and living with high family income likely to respond.

References:

- Data Mining and Knowledge Discovery – Springer
- Variable selection using Random Forests - Robin Genuer, Jean Michel Poggi, Christine Tuleau-Malo
- R Cookbook - O'Reilly
- The Art of R Programming - Norman Matloff
- http://www.r-tutor.com/gpu-computing/clustering/hierarchical-cluster-analysis

- http://www.r-bloggers.com/k-means-clustering-from-r-in-action/
- http://amunategui.github.io/blending-models/

Appendix:

This report has been written as an R Markdown document. The implementation of R Markdown is provided by two packages:

knitr — Weaves Rmd files into plain markdown (.md) files markdown — Converts markdown files into HTML documents

Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.

Tools used in the project: RStudio, RMD.

Packages used: knitr,caret, memisc, Rattle, ggplot2, FactoMineR, NbClust, MASS, HSAUR, cluster, fpc, reshape, ggthemes