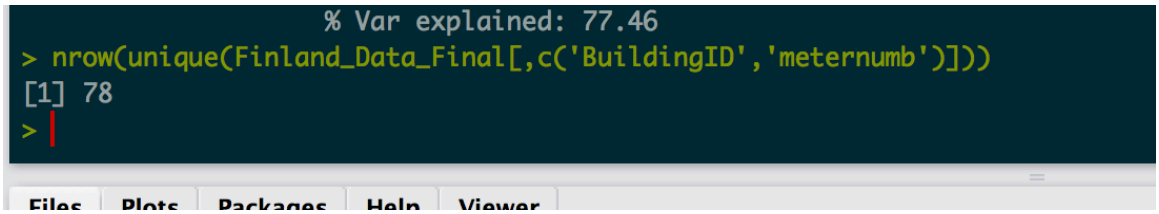
Part 2:

- You will need to create a daily base_hr_usage by {building_id, type, meter_no, weekday, month, holiday}.
- Use the average hours {0,1,2,3,4,22, 23} to compute this value.
- Create a column base_hr_usage and input the calculated value in that column. Note that youwill have one value for each hour for the whole day.
- Create another column called “Base_Hour_Class”. If the actual consumption is greater than thebase_hr_usage, use High Else Low

→ We can find number of unique combinations of Building id and meter number:

nrow(unique(Finland_Data_Final[,c('BuildingID','meternumb')]))




```
151
152 #create a daily base_hr_usage by {building_id, type, meter_no, weekday, month, holiday}.
153 #Use the average hours {0,1,2,3,4,22, 23} to compute this value.
154 Finland_Data_Final <- Finland_Data_Final %>%
155                   group_by(date,BuildingID,type, meternumb, Weekday, month, Holiday) %>%
156                   mutate(base_hr_usage = mean(Consumption[which(hour %in% c(0,1,2,3,4,22,23))]))
157
158 #nrow(unique(Finland_Data_Final[,c('BuildingID','meternumb')]))
159
160 #"Base_Hour_Class". If the actual consumption is greater than the base_hr_usage, use High Else Low
161 Finland_Data_Final <- Finland_Data_Final %>% mutate(Base_Hour_Class = ifelse(Consumption>base_hr_usage,"High","Low"))
162
163 Finland_Data_Final$Humidity <- as.numeric(as.character(Finland_Data_Final$Humidity))
164 Finland_Data_Final <- Finland_Data_Final %>% mutate(Base_hour_Flag = ifelse(Base_hour_Flag == "True" , 1, 0))
165 Finland_Data_Final$Wind_SpeedMPH <- as.numeric(as.character(Finland_Data_Final$Wind_SpeedMPH))
166
167
168 Finland_Data_Final_test <- Finland_Data_Final
169 #Split dataframe ainto 78 dataframes
170 split_dataset_final <- split(Finland_Data_Final, with(Finland_Data_Final, interaction(BuildingID,meternumb)), drop = TRUE)
171
172 #write.csv(Finland_Data_Final, "FinalDataIt.csv")
173
```

Filter

	BuildingID	meternumb
1	5198	1
2	5199	1
3	5286	1
4	5288	1
5	5290	1
6	5290	3
7	5290	8
8	5304	1
9	5304	3
10	5306	1
11	5308	1
12	5310	1
13	5311	1
14	5313	1
15	5314	1
16	5316	1
17	5317	1
18	5318	1
19	5322	1
20	5323	1
21	5325	1
22	5329	1

23	5332	1
24	5333	1
25	5335	1
26	5340	1
27	5351	1
28	5352	1
29	5355	1
30	5356	1
31	5357	1
32	5358	1
33	28096	1
34	28096	4
35	28108	3
36	28161	1
37	28162	1
38	28168	1
39	28168	4
40	30602	1
41	75689	1
42	75690	1
43	75691	1
44	75692	1

	BuildingID	meternumb
45	75693	1
46	75694	1
47	75696	1
48	75696	2
49	75697	1
50	75698	1
51	75699	1
52	75700	1
53	75701	1
54	75702	1
55	75703	1
56	75704	1
57	75705	1
58	75706	1
59	75707	1
60	75708	1
61	75709	1
62	75710	1
63	75711	1
64	75712	1
65	75713	1
66	75714	1

57	75705	1
58	75706	1
59	75707	1
60	75708	1
61	75709	1
62	75710	1
63	75711	1
64	75712	1
65	75713	1
66	75714	1
67	75715	1
68	75716	1
69	75717	1
70	75718	1
71	75719	1
72	75720	1
73	75721	1
74	77662	2
75	81909	1
76	82254	1
77	83427	1
78	84681	1

- Prediction

Neural Network Prediction:

Using the `gml()` function instead of the `lm()` this will become useful later when cross validating the linear model.

```
set.seed(123)
data2 <- split_dataset_final[[1]]
data2 <- data2[,c(18,6,8,9,10,12,19,21,22,16,26)]
data2 <- as.data.frame(data2)
data_original <- data2
apply(data2,2,function(x) sum(is.na(x)))
normalize<-function(x)(return((x-min(x))/(max(x)-min(x))))
data1<-as.data.frame(lapply(data2[,c(1:9,11)],normalize))
data3 <- as.data.frame(data2$Normalized_Consumption)
colnames(data3) <- c("Normalized_Consumption")
data <- cbind(data1, data3)

index <- sample(1:nrow(data),round(0.75*nrow(data)))
index_original <- index
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(Normalized_Consumption~TemperatureF+hour+dayOfWeek+Weekday+month
              +Holiday+Dew_PointF+Sea_Level_PressureIn
              +VisibilityMPH+WindDirDegrees
              , data=train)
summary(lm.fit)
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$Normalized_Consumption)^2)/nrow(test)


```

> MSE.lm
[1] 0.002173556524

Preparing to fit the neural network

It is good practice to normalize your data before training a neural network. Usually scaling in the intervals [0,1] or [-1,1] tends to give better results. We therefore scale and split the data before moving on:

`scale` returns a matrix that needs to be coerced into a data.frame.

```r
#Preparing to fit the neural network
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)

scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins))
train_ <- scaled[index,]
test_ <- scaled[-index,]
```

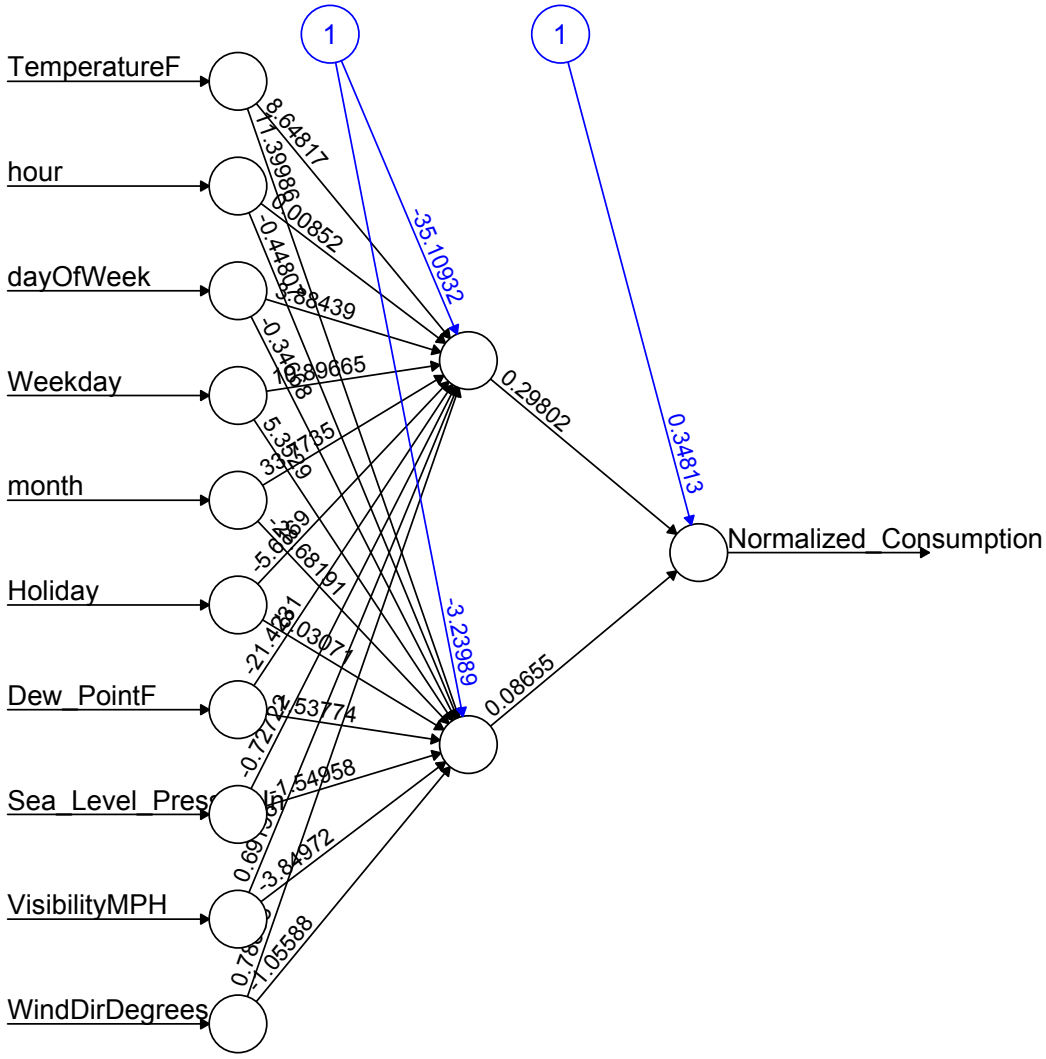
Usually, if at all necessary, one hidden layer is enough for a vast numbers of applications. As far as the number of neurons is concerned, it should be between the input layer size and the output layer size, usually 2/3 of the input size.

2 hidden layers will be used

The input layer has 10 inputs, the two hidden layers have 2 neurons and the output layer has, of course, a single output since we are doing regression.

fit the net:

```r
n <- names(train_)
f <- as.formula(paste("Normalized_Consumption ~", paste(n[!n %in% "Normalized_Consumption"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=2,linear.output=T)
nn <- neuralnet(f,data=train_,hidden=2,linear.output=T)
plot(nn)
print(nn)
```



Error: 15.542083 Steps: 15082

The black lines show the connections between each layer and the weights on each connection while the blue lines show the bias term added in each step.

Predicting Normalized Consumption using the neural network

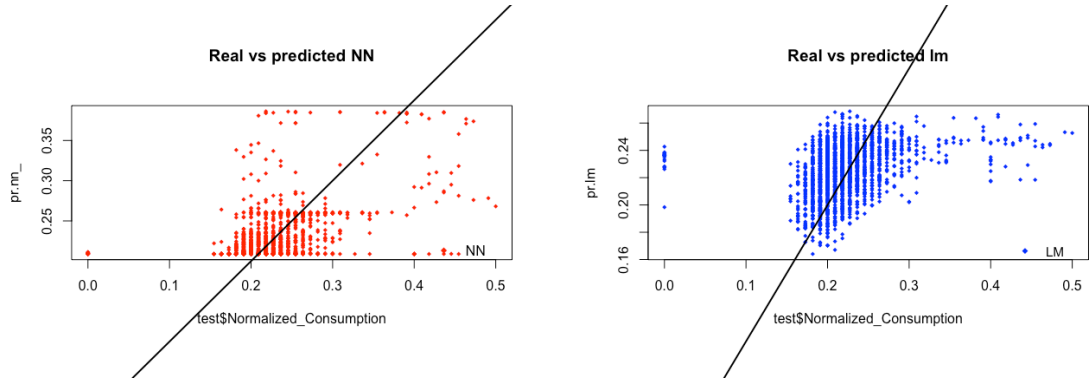
```
#Predicting Normalized Consumption using the neural network

pr.nn <- compute(nn,test_[,1:10])
pr.nn_ <- pr.nn$net.result*(max(data$Normalized_Consumption)-
                            min(data$Normalized_Consumption))+min(data$Normalized_Consumption)
test.r <- (test_$Normalized_Consumption)*(max(data$Normalized_Consumption)-
                                          min(data$Normalized_Consumption))+min(data$Normalized_Consumption)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
#we then compare the two MSEs

print(paste(MSE.lm,MSE.nn))
```

> print(paste(MSE.lm,MSE.nn))
[1] "0.00217355652434915 0.00185239396341829"

Apparently the net is doing a better work than the linear model at prediction.

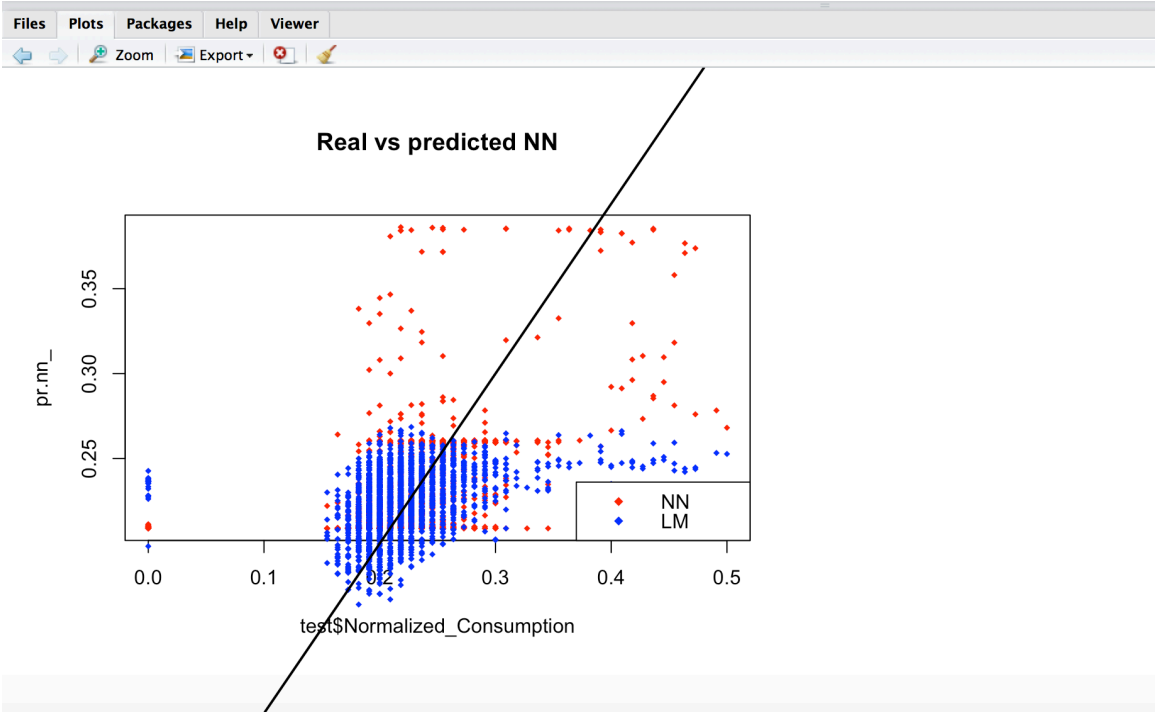
Below, after the visual plot, we are going to perform a fast cross validation in order to be more confident about the results.

```
par(mfrow=c(1,2))
plot(test$Normalized_Consumption,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')

plot(test$Normalized_Consumption,pr.lm,col='blue',main='Real vs predicted lm',pch=18, cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='LM',pch=18,col='blue', bty='n', cex=.95)
```



By visually inspecting the plot we can see that the predictions made by the neural network are (in general) more concentrated around the line (a perfect alignment with the line would indicate a MSE of 0 and thus an ideal perfect prediction) than those made by the linear model.

A perhaps more useful visual comparison is plotted below:



Prediction:

```
#Prediction

pr.nn_ <- as.data.frame(pr.nn_)
colnames(pr.nn_) <- c("Predicted value")
data2_test <- data_original[-index_original,]
Final_prediction_nn <- cbind(data2_test, pr.nn_)

#calculating mean square value
rmse(pr.nn_,data2_test$Normalized_Consumption )
rmse <- c("RMS", rmse(pr.nn_,data2_test$Normalized_Consumption))
mae <- c("MAE", mae(pr.nn_,data2_test$Normalized_Consumption))

# Function that returns Mean Absolute Percentage Error

mape <- function(error)  {
  mean(abs(error/Neuraldataset$kWh) * 100)
}

mape <- c("MAPE", mape(pr.nn_,data2_test$Normalized_Consumption))
```

standard_deviation <- sd(Final_prediction_nn[,10])

> standard_deviation
[1] 0.04907350636

> Final_prediction_nn <- Final_prediction_nn %>% mutate(Outlier_Tag = ifelse(residual >= (2*standard_deviation) , 1, 0))
> unique(Final_prediction_nn$Outlier_Tag)
[1] 0 1

```
   TemperatureF hour dayOfWeek Weekday month Holiday
2          33.8    1         2       1     1       1
3          33.8    2         2       1     1       1
5          35.6    4         2       1     1       1
10         37.4    9         2       1     1       1
16         37.4   15         2       1     1       1
17         37.4   16         2       1     1       1
   Dew_PointF Sea_Level_PressureIn VisibilityMPH
2        32.0                29.42           1.9
3        33.8                29.39           3.7
5        33.8                29.33           5.0
10       35.6                29.30           4.3
16       35.6                29.33           1.9
17       37.4                29.33           2.5
   Normalized_Consumption WindDirDegrees Predicted value
2            0.2272727273            150    0.2290355289
3            0.2363636364            150    0.2267607404
5            0.2272727273            150    0.2272411944
10           0.2363636364            180    0.2290167902
16           0.2272727273            190    0.2306024999
17           0.2181818182            190    0.2297830560
            residual Outlier_Tag
2  -0.00176280167322           0
3   0.00960289597545           0
5   0.00003153285363           0
10  0.00734684614090           0
16 -0.00332977258711           0
17 -0.01160123784468           0
```

#Calculating mape, mae, rmse

```r
#calculating mean square value
rmse(pr.nn_,data2_test$Normalized_Consumption )
rmse <- c("RMS", rmse(pr.nn_,data2_test$Normalized_Consumption))
mae <- c("MAE", mae(pr.nn_,data2_test$Normalized_Consumption))

# Function that returns Mean Absolute Percentage Error

mape <- abs((Final_prediction_nn$residual/Final_prediction_nn$Normalized_Consumption)*100)
mape <- mean(mape)
mape <- c("MAPE", mape)
neural_performance <- rbind(rmse, mae, mape, deparse.level = 0)
```

[1,] "RMS" "0.0430394465974911"
[2,] "MAE" "0.0270693978301965"
[3,] "MAPE" "Inf"

Regression Prediction

Fit a regression tree to the data set. create a training set, and fit the tree to the training data.

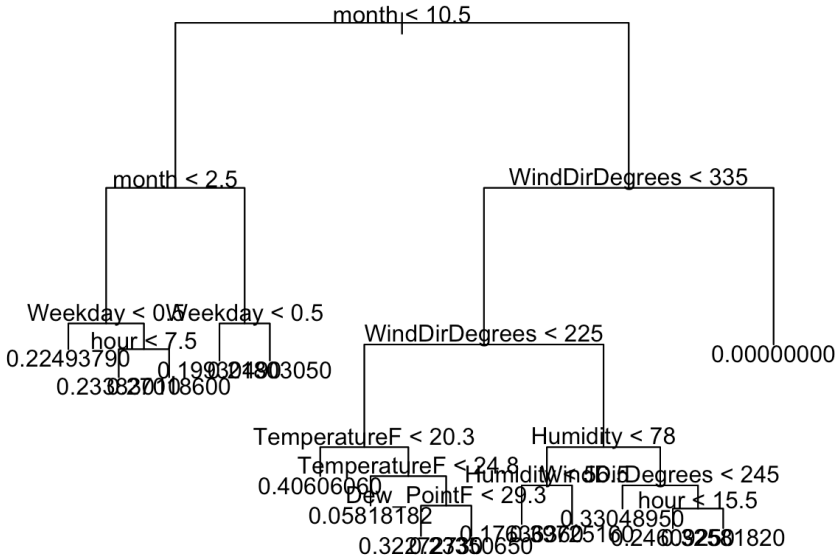
```
#fit a regression tree to the  data set. create a training set, and fit the tree to the training data.
set.seed(123)
data_reg_tree <- split_dataset_final[[1]]
data_reg_tree <- data_reg_tree[,c(18,6,9,10,12,14,19,20,26,16)]

train = sample (1:nrow(data_reg_tree), nrow(data_reg_tree)/2)
tree.data = tree(Normalized_Consumption~.,data_reg_tree,subset=train)
summary (tree.data)
```

Regression tree:
tree(formula = Normalized_Consumption ~ ., data = data_reg_tree,
subset = train)
Variables actually used in tree construction:
[1] "month"
[2] "Weekday"
[3] "hour"
[4] "WindDirDegrees"
[5] "TemperatureF"
[6] "Dew_PointF"
[7] "Humidity"
Number of terminal nodes: 15
Residual mean deviance: 0.001415731 = 5.790341 / 4090
Distribution of residuals:
Min. 1st Qu.
-0.322727300 -0.018030520
Median Mean
0.000151299 0.000000000
3rd Qu. Max.
0.018333120 0.268181800

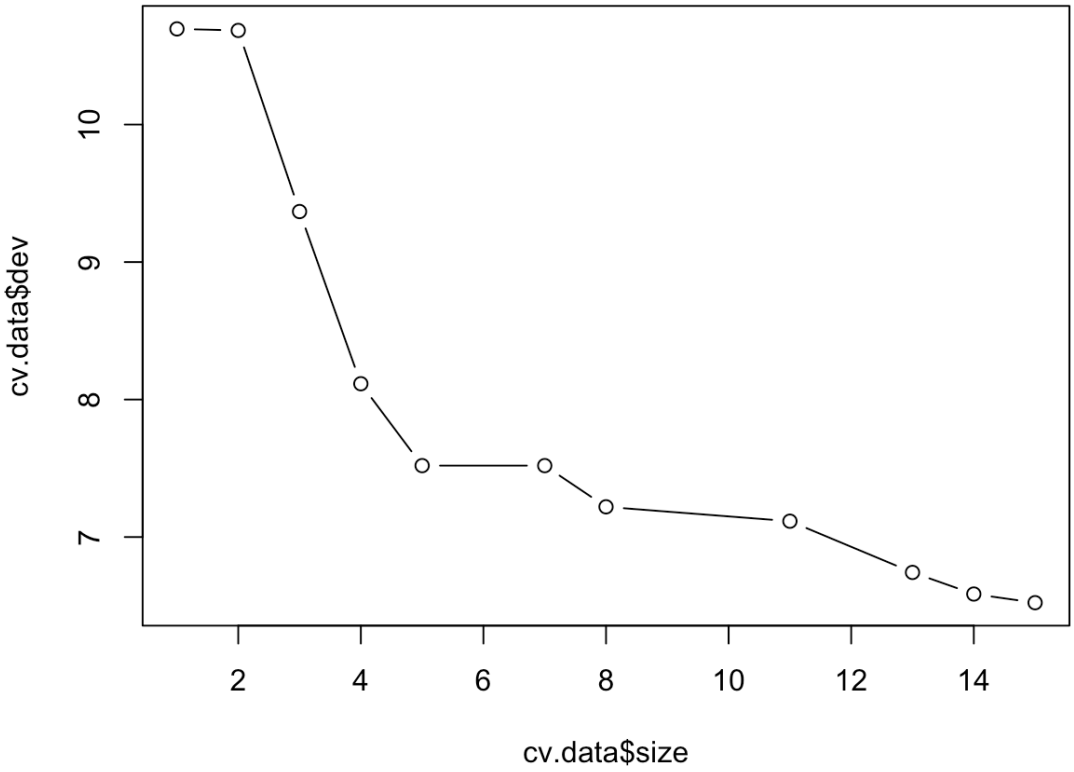
Only, 7 variables have been used to construct a tree

#Plotting a tree

```
#plot the tree.
plot (tree.data)
text (tree.data, pretty = 0)
#Cross-validation for Choosing Tree Complexity
cv.data = cv.tree (tree.data)
plot (cv.data$size, cv.data$dev, type='b')
#prune the tree, we could use the prune.tree() function:
prune.data =prune.tree(tree.data, best = 5)
plot(prune.data)
text(prune.data, pretty = 0)
#In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.
yhat=predict (tree.data, newdata =data_reg_tree [-train,])
data.test=data_reg_tree [-train,"Normalized_Consumption"]
data.test1 <- as.double(data.test[['Normalized_Consumption']])
plot(yhat,data.test1)
abline (0,1)
```

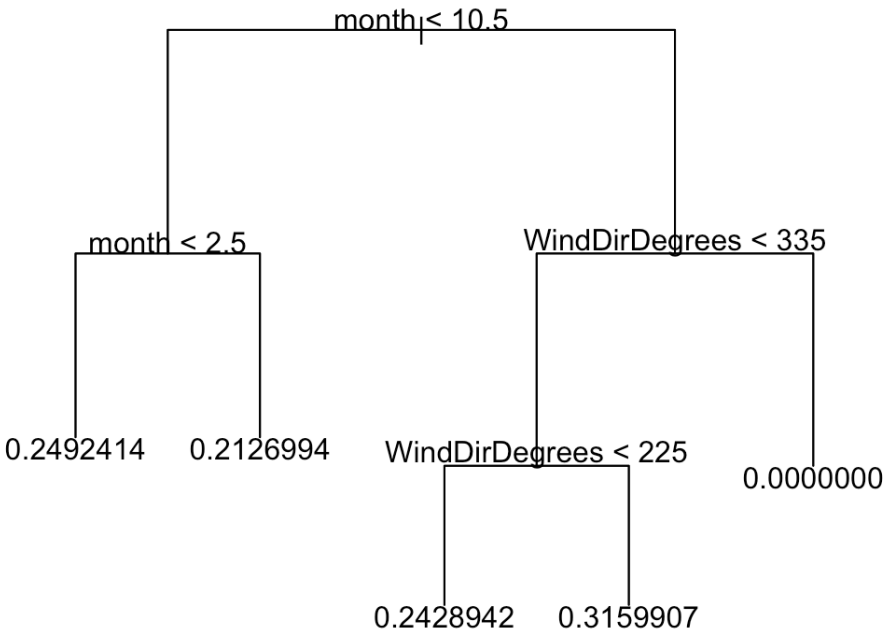


#Cross-validation for Choosing Tree Complexity

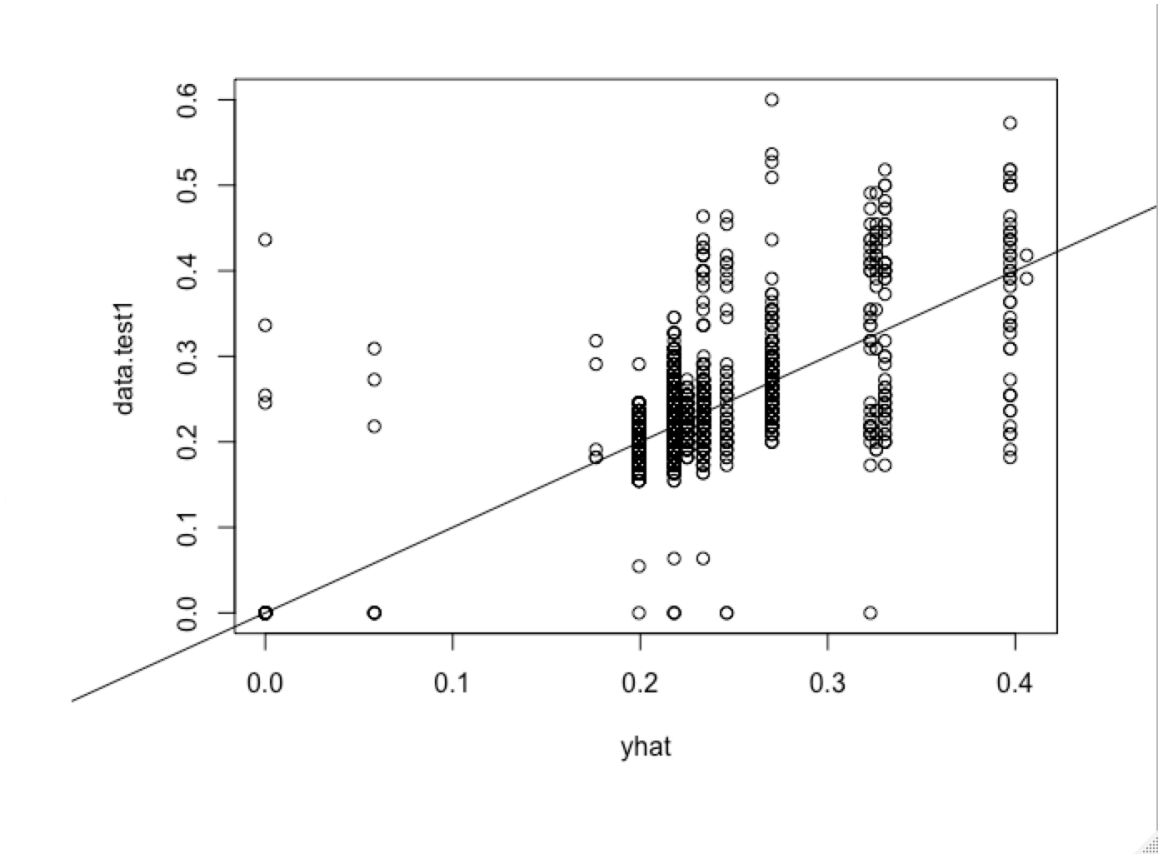


See whether the pruning will improve the performance:

prune the tree, we could use the prune.tree() function:



In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

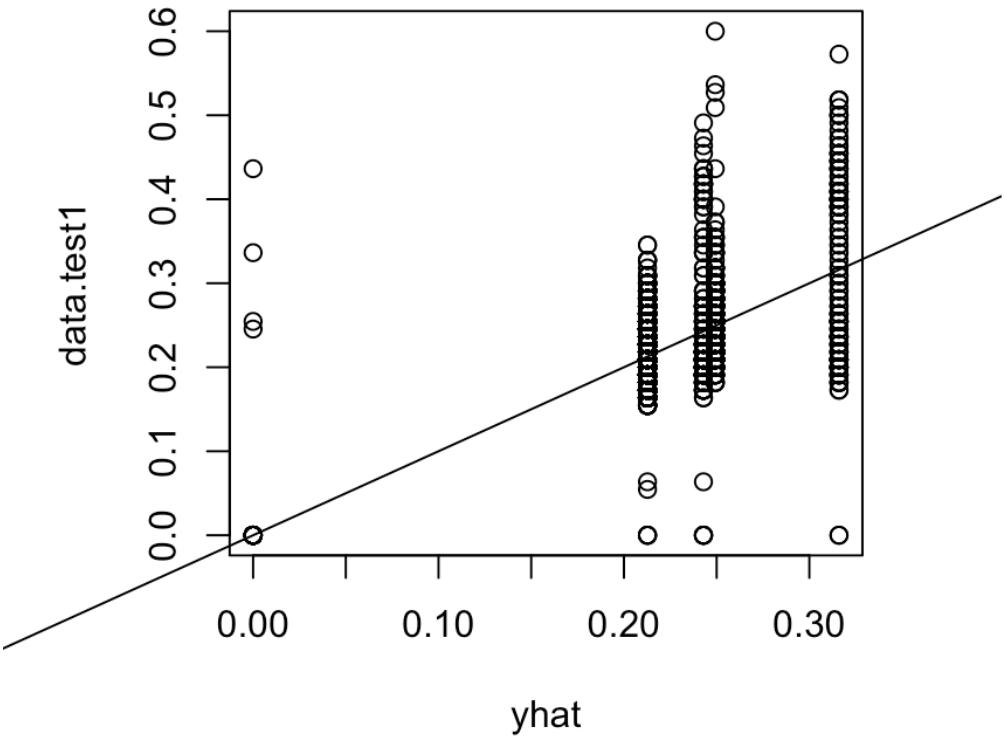



```
#MSE
mean((yhat -data.test)^2)
accuracy_1 <- accuracy(yhat, data.test1)
```

> mean((yhat -data.test)^2)
[1] 0.001599257638
> accuracy_1 <- accuracy(yhat, data.test1)
> accuracy_1
ME RMSE MAE MPE MAPE
Test set 0.0001223930285 0.03999071939 0.02517541932 -Inf Inf

After removing the least important splits, the MSE only increases a little

```
#After removing the least important splits, the MSE only increases a little
yhat=predict (prune.data, newdata =data_reg_tree [-train,])
data.test=data_reg_tree [-train,"Normalized_Consumption"]
data.test1 <- as.double(data.test[['Normalized_Consumption']])
plot(yhat,data.test1)
abline (0,1)
#MSE
mean((yhat -data.test)^2)
accuracy_2 <- accuracy(yhat, data.test1)
```



> mean((yhat -data.test)^2)
[1] 0.001894604213
> accuracy_2 <- accuracy(yhat, data.test1)
> accuracy_2
ME RMSE MAE MPE MAPE
Test set -0.00008241686817 0.04352705151 0.02824680772 -Inf Inf

So, mean is improved now

#Prediction

```
test <- as.data.frame( data_reg_tree [-train,variable_unique])
yhat_1 <- as.data.frame(yhat)
colnames(yhat_1) <- c("Predicted Value")
data.test_1 <- as.data.frame(data.test)
Final_prediction <- cbind(test,yhat_1,data.test_1)
```

```
> Final_prediction$residual <- Final_prediction$Normalized_Consumption -
+   Final_prediction_nn$`Predicted value`
> standard_deviation <- sd(Final_prediction$residual)
> standard_deviation
[1] 0.05694269232
> Final_prediction <- Final_prediction %>% mutate(Outlier_Tag = ifelse(residual >= (2*standard_deviation) , 1, 0))
> head(Final_prediction)
  month Weekday hour WindDirDegrees TemperatureF Dew_PointF Humidity Predicted Value
1     1       1    1            150         33.8       32.0       93    0.2492413617
2     1       1    2            150         33.8       33.8      100    0.2492413617
3     1       1    4            150         35.6       33.8       93    0.2492413617
4     1       1    6            180         37.4       35.6       93    0.2492413617
5     1       1    7            180         37.4       35.6       93    0.2492413617
6     1       1    9            180         37.4       35.6       93    0.2492413617
  Normalized_Consumption          residual Outlier_Tag
1           0.2272727273 -0.00176280167322           0
2           0.2363636364  0.00960289597545           0
3           0.2272727273  0.00003153285363           0
4           0.2272727273 -0.00174406295001           0
5           0.2454545455  0.01485204559471           0
6           0.2363636364  0.00658058033714           0
>
```

Random Forest Prediction

Random forests improve predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables), classifying a case using each tree in this new "forest", and deciding a final predicted outcome by combining the results across all of the trees (an average in regression, a majority vote in classification).

```r
# Random Forests
# Random forests improve predictive accuracy by generating a large number of bootstrapped t

library(randomForest)
set.seed(123)
data_reg_random_forest <- split_dataset_final[[1]]
index_1 <- sample(1:nrow(data_reg_random_forest),round(0.75*nrow(data_reg_random_forest)))
index_original_1 <- index_1
train <- data_reg_random_forest[index_1,]
test <- data_reg_random_forest[-index_1,]

fit <- randomForest(Normalized_Consumption ~ TemperatureF+hour+Weekday+month
                    +Dew_PointF +WindDirDegrees +Humidity,    data = train)
print(fit) # view results
importance(fit) # importance of each predictor
predicion_random_forest <- predict(fit, test)
accuracy(predicion_random_forest, test$Normalized_Consumption)
#-------------------------------------------------------------------------------------
```

Call:
randomForest(formula = Normalized_Consumption ~ TemperatureF + hour + Weekday + month + Dew_PointF + WindDirDegrees + Humidity, data = train)
Type of random forest: regression
Number of trees: 500
No. of variables tried at each split: 2

Mean of squared residuals: 0.0008449698134
% Var explained: 68.7

```
> importance(fit) # importance of each predictor
              IncNodePurity
TemperatureF    1.950842558
hour            2.165137500
Weekday         1.160022237
month           3.363550462
Dew_PointF      1.776769290
WindDirDegrees  2.648233099
Humidity        1.608882317
```

```
data2_test <- test[,c(18,6,9,10,19,26,20,16)]
predicion_random_forest <- as.data.frame(predicion_random_forest)
Final_prediction_rr <- cbind(data2_test, predicion_random_forest)
Final_prediction_rr$residual <- Final_prediction_rr$Normalized_Consumption -
  Final_prediction_rr$predicion_random_forest
standard_deviation <- sd(Final_prediction_rr$residual)
Final_prediction_rr <- Final_prediction_rr %>%
  mutate(Outlier_Tag = ifelse(residual >= (2*standard_deviation) , 1, 0))

#calculating mean square value
accuracy(predicion_random_forest, test$Normalized_Consumption)
#-------------------------------------------------------------------------------
```

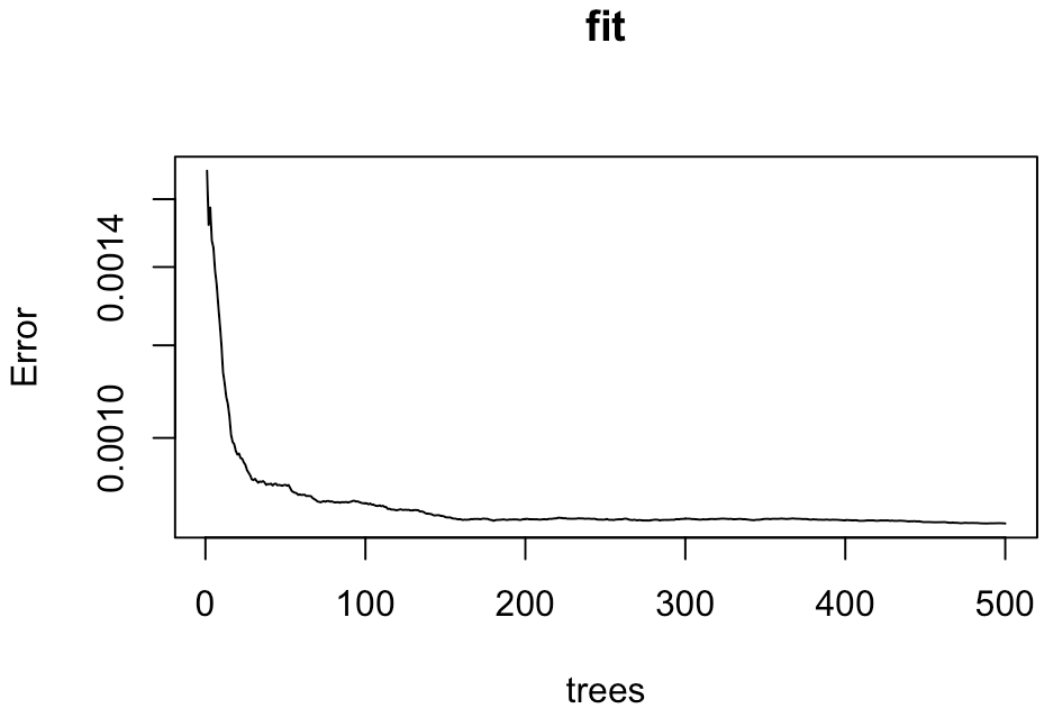
- **accuracy(predicion_random_forest, Final_prediction_rr$Normalized_Consumption)**
-

```
> head(Final_prediction_rr)
  TemperatureF hour Weekday month Dew_PointF WindDirDegrees Humidity
1         33.8    1       1     1       32.0            150       93
2         33.8    2       1     1       33.8            150      100
3         35.6    4       1     1       33.8            150       93
4         37.4    9       1     1       35.6            180       93
5         37.4   15       1     1       35.6            190       93
6         37.4   16       1     1       37.4            190      100
  Normalized_Consumption predicion_random_forest        residual Outlier_Tag
1           0.2272727273            0.2318583389 -0.004585611631           0
2           0.2363636364            0.2290686577  0.007294978621           0
3           0.2272727273            0.2371340468 -0.009861319527           0
4           0.2363636364            0.2273727179  0.008990918438           0
5           0.2272727273            0.2317184482 -0.004445720887           0
6           0.2181818182            0.2338186263 -0.015636808137           0
>
```

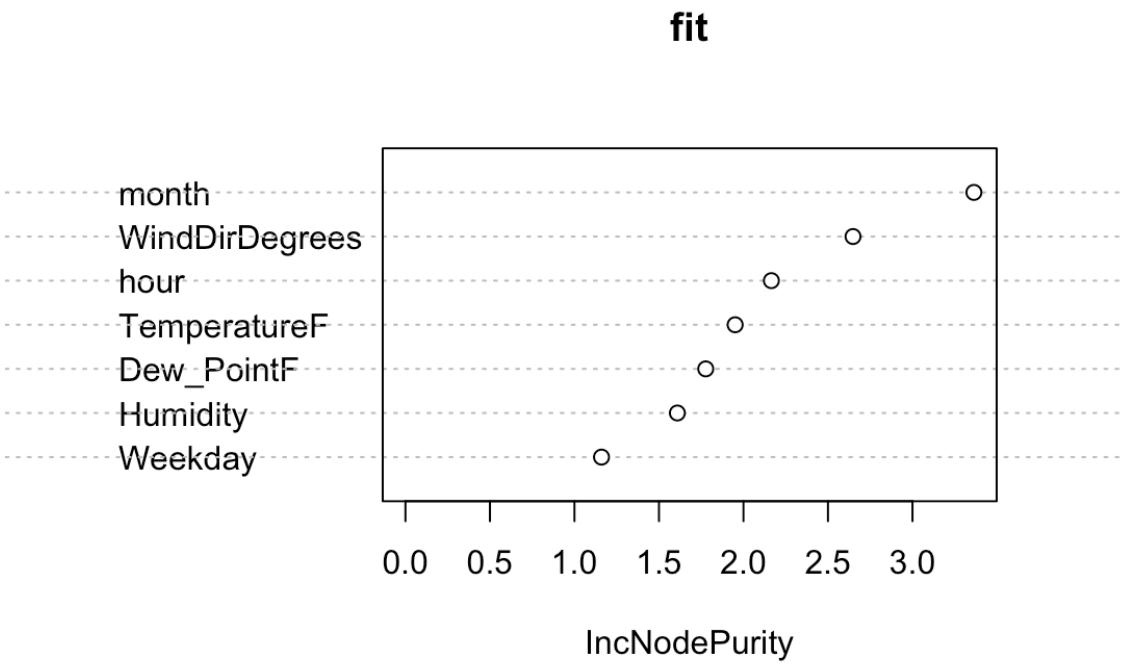
> randomforest_performance

	ME	RMSE	MAE	MPE	MAPE
Test set	**-0.001211284622**	**0.04832920999**	**0.01674909995**	**-Inf**	**Inf**

plot(fit, log="y")



varImpPlot(fit)



knn-Regression Prediction

```r
install.packages("FNN")
library(FNN)
data2 <- split_dataset_final[[1]]
data2 <- data2[,c(18,6,8,9,10,12,19,21,22,16,26)]
data2 <- as.data.frame(data2)
data_original <- data2

train <- sample(1:nrow(data2),round(0.75*nrow(data2)))
traindata <- data2[train,]
testdata <- data2[-train,]

tree_knn<-knn.reg(traindata,y=traindata$Normalized_Consumption,testdata,k=3)
testdata$Normalized_Consumption

error=as.double(tree_knn[[4]])-testdata$Normalized_Consumption
error <- as.data.frame(error)
performance <- accuracy(tree_knn[[4]],testdata$Normalized_Consumption)
tree_knn[[4]] <- as.data.frame(tree_knn[[4]])
colnames(tree_knn[[4]]) <- c("Predicted value")
Final_prediction_knn_reg <- cbind(testdata, tree_knn[[4]])
Final_prediction_knn_reg$residual <- Final_prediction_knn_reg$Normalized_Consumption -
  Final_prediction_knn_reg$`Predicted value`
standard_deviation <- sd(Final_prediction_knn_reg$residual)
Final_prediction_knn_reg <- Final_prediction_knn_reg %>% mutate(Outlier_Tag = ifelse(residual >= (2*standard_deviation) , 1, 0))

write.csv( Final_prediction_knn_reg, "5198_1_knn_reg_Prediction.csv")
write.csv(performance, "5198_1_knn_reg_performance.csv")
```

Filter

	ME	RMSE	MAE	MPE	MAPE
Test set	0.0008885740011	0.04306537475	0.02618784041	-Inf	Inf

```
> head(Final_prediction_knn_reg)
   TemperatureF hour dayOfWeek Weekday month Holiday Dew_PointF Sea_Level_PressureIn VisibilityMPH
2          33.8    1         2       1     1       1       32.0                29.42           1.9
13         37.4   12         2       1     1       1       35.6                29.30           6.2
17         37.4   16         2       1     1       1       37.4                29.33           2.5
24         37.4   23         2       1     1       1       35.6                29.42           4.3
27         37.4    2         3       1     1       0       35.6                29.44           3.7
33         35.6    8         3       1     1       0       35.6                29.47           1.2
   Normalized_Consumption WindDirDegrees Predicted value        residual Outlier_Tag
2            0.2272727273            150    0.2303030303 -0.003030303030           0
13           0.2363636364            190    0.2272727273  0.009090909091           0
17           0.2181818182            190    0.2303030303 -0.012121212121           0
24           0.2272727273            200    0.2212121212  0.006060606061           0
27           0.2272727273            190    0.2121212121  0.015151515152           0
33           0.2545454545            210    0.1969696970  0.057575757576           0
>
```

Best Model:

```
> knnReg_RMSE
[1] 0.04306537475
> NEURALnode_RMSE
[1] 0.0430394466
> randomForest_RMSE
[1] 0.04832920999
> regression_tree_rmse
[1] 0.04352705151
>
```

```
> min(knnReg_RMSE, NEURALnode_RMSE, randomForest_RMSE, regression_tree_rmse)
[1] 0.0430394466
```

Hence least rmse belongs to Neural node

Thus, Neural Node Model is the best model

Classification

Logistic Regression:

```r
#Some descriptive statistics of Consumption data
data_logi <- split_dataset_final[[1]]
#data_logi <- data_logi[,c(18,6,8,9,10,12,16,14,19,20,21,22,24,26,28)]

summary(data_logi)

data_logi$Base_Hour_Class <- as.factor(data_logi$Base_Hour_Class)
table(data_logi$Base_Hour_Class)

#Construct a logistic regression model for factor Base_Hour_Class using all other variables
fit1<-glm(Base_Hour_Class~month+hour+TemperatureF+Dew_PointF+Humidity+Sea_Level_PressureIn+VisibilityMPH+WindDir
        +dayOfWeek+Weekday+Holiday,data=data_logi,family=binomial(link="logit"))

summary(fit1)

#Construct a logistic regression model for factor Base_Hour_Class
fit2 <- glm(Base_Hour_Class ~ month + Sea_Level_PressureIn + Normalized_Consumption + Weekday,
           data=data_logi, family=binomial(link="logit"))
summary(fit2)

#Parameters of fit2
coef(fit2)

#Predict the probability of the outcome
newdata <- data.frame(month=mean(data_logi$month),
                      Sea_Level_PressureIn=mean(data_logi$Sea_Level_PressureIn),
                      Normalized_Consumption=mean(data_logi$Normalized_Consumption),
                      Weekday=mean(data_logi$Weekday))
newdata
prob <- predict(fit2, newdata=newdata, type="response")
prob

```

summary(data_logi)

```
   BuildingID     building           meternumb           type           date
Min.   :5198   Length:8211        Min.   :1   Dist_Heating:   0   Min.   :2013-01-01
1st Qu.:5198   Class :character   1st Qu.:1   elect       :8211   1st Qu.:2013-03-27
Median :5198   Mode  :character   Median :1                       Median :2013-06-21
Mean   :5198                      Mean   :1                       Mean   :2013-06-20
3rd Qu.:5198                      3rd Qu.:1                       3rd Qu.:2013-09-14
Max.   :5198                      Max.   :1                       Max.   :2013-12-08

     hour          Consumption         dayOfWeek          Weekday             month
Min.   : 0.00000   Min.   : 0.00000   Min.   :0.00000   Min.   :0.0000000   Min.   : 1.000000
1st Qu.: 5.00000   1st Qu.:22.00000   1st Qu.:1.00000   1st Qu.:0.0000000   1st Qu.: 3.000000
Median :11.00000   Median :24.00000   Median :3.00000   Median :1.0000000   Median : 6.000000
Mean   :11.49665   Mean   :24.69358   Mean   :3.00475   Mean   :0.7131896   Mean   : 6.158568
3rd Qu.:17.00000   3rd Qu.:26.00000   3rd Qu.:5.00000   3rd Qu.:1.0000000   3rd Qu.: 9.000000
Max.   :23.00000   Max.   :66.00000   Max.   :6.00000   Max.   :1.0000000   Max.   :12.000000

Base_hour_Flag         Holiday                        X..address   area_floor._m.sqr
Min.   :0.0000000   Min.   :0.00000000   Metsahovintie 113 :8211   Min.   :110
1st Qu.:0.0000000   1st Qu.:0.00000000    Arkadiankatu 24  :   0   1st Qu.:110
Median :0.0000000   Median :0.00000000    Otaniementie 17  :   0   Median :110
Mean   :0.2919255   Mean   :0.04676653   Betonimiehenkuja 1:   0   Mean   :110
3rd Qu.:1.0000000   3rd Qu.:0.00000000   Betonimiehenkuja 3:   0   3rd Qu.:110
Max.   :1.0000000   Max.   :1.00000000   Betonimiehenkuja 5:   0   Max.   :110
                                         (Other)           :   0
AirportCode        Normalized_Consumption      Time           TemperatureF
Length:8211        Min.   :0.0000000      Min.   :00:00:00   Min.   :-22.00000
Class :character   1st Qu.:0.2000000      1st Qu.:05:50:00   1st Qu.: 30.20000
Mode  :character   Median :0.2181818      Median :11:50:00   Median : 44.60000
                   Mean   :0.2244871      Mean   :11:50:37   Mean   : 44.13898
                   3rd Qu.:0.2363636      3rd Qu.:17:50:00   3rd Qu.: 59.00000
                   Max.   :0.6000000      Max.   :23:50:00   Max.   : 86.00000

  Dew_PointF          Humidity         Sea_Level_PressureIn VisibilityMPH
Min.   :-27.4000   Min.   : 12.00000   Min.   :28.74000     Min.   : 0.100000
1st Qu.: 24.8000   1st Qu.: 64.00000   1st Qu.:29.74000     1st Qu.: 6.200000
Median : 37.4000   Median : 82.00000   Median :29.92000     Median : 6.200000
Mean   : 36.3288   Mean   : 76.96395   Mean   :29.89768     Mean   : 5.880088
3rd Qu.: 50.0000   3rd Qu.: 93.00000   3rd Qu.:30.09000     3rd Qu.: 6.200000
Max.   : 68.0000   Max.   :100.00000   Max.   :30.54000     Max.   :31.000000

Wind_Direction     Wind_SpeedMPH        Conditions        WindDirDegrees    base_hr_usage
```

> table(data_logi$Base_Hour_Class)

High Low
4396 3815

#Construct a logistic regression model for factor Base_Hour_Class using all other variables

```
> summary(fit1)

Call:
glm(formula = Base_Hour_Class ~ month + hour + TemperatureF +
    Dew_PointF + Humidity + Sea_Level_PressureIn + VisibilityMPH +
    WindDirDegrees + Wind_SpeedMPH + Normalized_Consumption +
    dayOfWeek + Weekday + Holiday, family = binomial(link = "logit"),
    data = data_logi)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.8998817  -0.9992677  -0.3381317   0.9487127   3.6045067

Coefficients:
                           Estimate     Std. Error   z value              Pr(>|z|)
(Intercept)            22.6651607901   3.4284203182   6.61096    0.00000000003818272 ***
month                   0.0377364198   0.0100193800   3.76634             0.00016566 ***
hour                    0.0018339065   0.0037541854   0.48850             0.62519820
TemperatureF            0.0121815535   0.0185665438   0.65610             0.51175826
Dew_PointF             -0.0101085397   0.0192513787  -0.52508             0.59952665
Humidity               -0.0018830176   0.0078822536  -0.23889             0.81118832
Sea_Level_PressureIn   -0.5518168023   0.1069706266  -5.15858    0.00000024882646802 ***
VisibilityMPH          -0.0074748007   0.0107418127  -0.69586             0.48651634
WindDirDegrees          0.0002039154   0.0002405438   0.84773             0.39659009
Wind_SpeedMPH          -0.0004078618   0.0064862830  -0.06288             0.94986153
Normalized_Consumption -28.6528311999   1.0535959556 -27.19527 < 0.00000000000000222 ***
dayOfWeek               0.0051566266   0.0124140516   0.41539             0.67785916
Weekday                -0.4215383205   0.0571347357  -7.37797    0.00000000000016072 ***
Holiday                -0.1015298885   0.1142595591  -0.88859             0.37422353
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 11341.7178  on 8210  degrees of freedom
Residual deviance:  9440.3229  on 8197  degrees of freedom
AIC: 9468.3229

Number of Fisher Scoring iterations: 5
```

Here, month + Sea_Level_PressureIn + Normalized_Consumption + Weekday are the only important variables with suitable p-values

Hence, Construct a logistic regression model for factor Base_Hour_Class

Summary of selected variables model:

```
Call:
glm(formula = Base_Hour_Class ~ month + Sea_Level_PressureIn +
    Normalized_Consumption + Weekday, family = binomial(link = "logit"),
    data = data_logi)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.8974590  -0.9884293  -0.3397656   0.9591171   3.6630697

Coefficients:
                           Estimate    Std. Error   z value              Pr(>|z|)
(Intercept)             19.720544822   2.845829666   6.92963     0.0000000000042194 ***
month                    0.042281736   0.008377415   5.04711     0.0000004485423604 ***
Sea_Level_PressureIn    -0.447278048   0.093443831  -4.78660     0.0000016963217799 ***
Normalized_Consumption -29.454532956   1.022155052 -28.81611 < 0.00000000000000222 ***
Weekday                 -0.399792181   0.056487605  -7.07752     0.0000000000014676 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 11341.7178  on 8210  degrees of freedom
Residual deviance:  9468.9183  on 8206  degrees of freedom
AIC: 9478.9183

Number of Fisher Scoring iterations: 5
```

Coefficients:

```
> coef(fit2)
          (Intercept)                 month   Sea_Level_PressureIn Normalized_Consumption
        19.72054482184         0.04228173613        -0.44727804777        -29.45453295563
              Weekday
        -0.39979218116
```

#Predict the probability of the outcome

```
> newdata <- data.frame(month=mean(data_logi$month),
+                       Sea_Level_PressureIn=mean(data_logi$Sea_Level_PressureIn),
+                       Normalized_Consumption=mean(data_logi$Normalized_Consumption),
+                       Weekday=mean(data_logi$Weekday))
> newdata
        month Sea_Level_PressureIn Normalized_Consumption      Weekday
1 6.158567775          29.89768481           0.2244871071 0.7131896237
```

Predict the probability: 0.4282661186

```r

#Split the data into training and testing
smp_size <- floor(0.80 * nrow(data_logi))
set.seed(123)
train_ind <- sample(seq_len(nrow(data_logi)), size = smp_size)
train <- data_logi[train_ind,]
test <- data_logi[-train_ind,]
#Build a logistic regression model on the training data
fit <- glm(Base_Hour_Class ~ month + Sea_Level_PressureIn + Normalized_Consumption + Weekday,
           data=train, family=binomial(link="logit"))
summary(fit)

#Run the model on the test set with cutoff value = 0.5
test.probs <- predict(fit,test,type="response")
pred <- rep("High",length(test.probs))
pred[test.probs>=0.5] <- "Low"

#classification matrix using confusionMatrix() function in caret package
library(caret)
confusionMatrix(test$Base_Hour_Class, pred)
#Generate ROC curve using ROCR package
library(ROCR)
prediction <- prediction(test.probs, test$Base_Hour_Class)
performance <- performance(prediction, measure = "tpr",x.measure = "fpr")
plot(performance, main = "ROC curve", xlab = "1-Specificity", ylab = "Sensitivity")

#Generate cumulative lift curve using lift() function in caret package
test$probs = test.probs
test$probs = sort(test$probs, decreasing = T)
lift <- lift(Base_Hour_Class ~ prob, data = test)
lift
xyplot(lift, plot = "gain")

```

#Split the data into training and testing and #Build a logistic regression model on the training data

Summary:

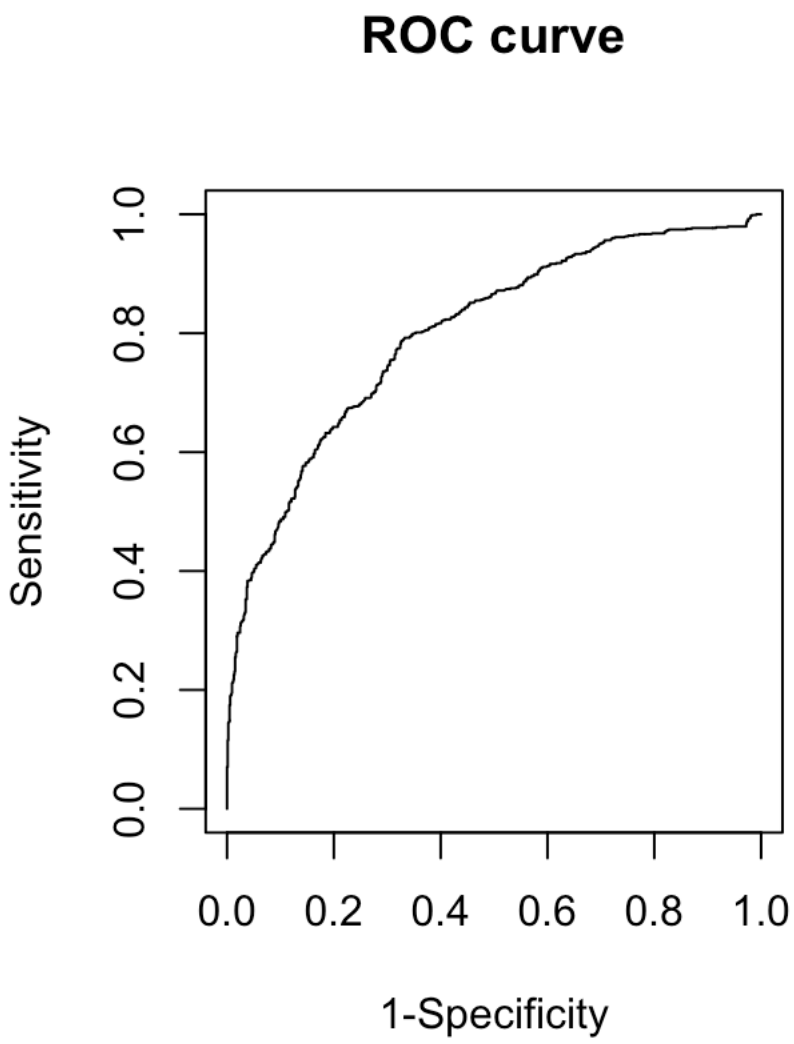
```
Call:
glm(formula = Base_Hour_Class ~ month + Sea_Level_PressureIn +
    Normalized_Consumption + Weekday, family = binomial(link = "logit"),
    data = train)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.8863156  -0.9865846  -0.3474295   0.9571399   3.6059481

Coefficients:
                           Estimate    Std. Error   z value              Pr(>|z|)
(Intercept)             21.474539708   3.167970517   6.77864      0.000000000012131 ***
month                    0.041005671   0.009378262   4.37242      0.000012287853249 ***
Sea_Level_PressureIn    -0.505529769   0.104006077  -4.86058      0.000001170428313 ***
Normalized_Consumption -29.550995928   1.144693674 -25.81564 < 0.00000000000000222 ***
Weekday                 -0.389219253   0.063248039  -6.15385      0.000000000756219 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 9067.3857  on 6567  degrees of freedom
Residual deviance: 7574.6660  on 6563  degrees of freedom
AIC: 7584.666

Number of Fisher Scoring iterations: 5
```

#Run the model on the test set with cutoff value = 0.5
#classification matrix using confusionMatrix() function in caret package

Confusion Matrix and Statistics

```
          Reference
Prediction High Low
      High  625 238
      Low   233 547

               Accuracy : 0.7133293
                 95% CI : (0.6907915, 0.7351024)
    No Information Rate : 0.5222155
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.4253688
 Mcnemar's Test P-Value : 0.8537701

            Sensitivity : 0.7284382
            Specificity : 0.6968153
         Pos Pred Value : 0.7242178
         Neg Pred Value : 0.7012821
             Prevalence : 0.5222155
         Detection Rate : 0.3804017
   Detection Prevalence : 0.5252587
      Balanced Accuracy : 0.7126268

       'Positive' Class : High
```

#Generate ROC curve using ROCR package

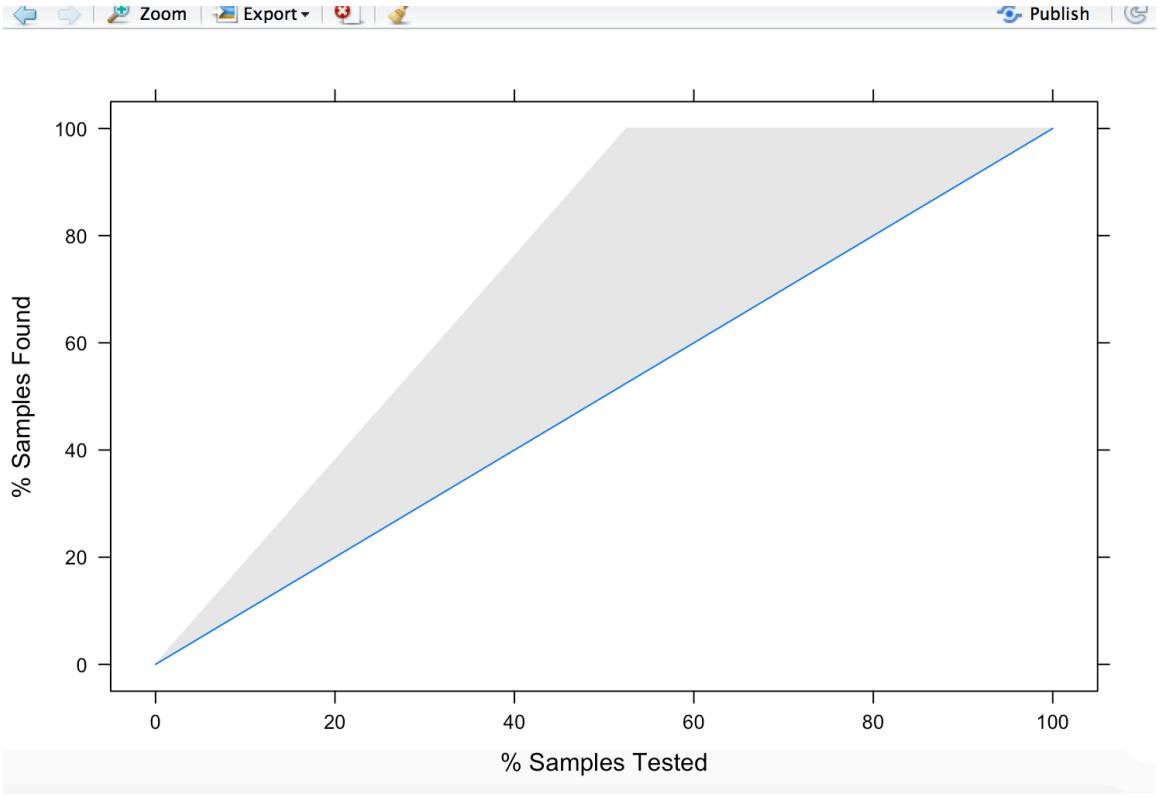


#Generate cumulative lift curve using lift() function in caret package

> lift

Call:
lift.formula(x = Base_Hour_Class ~ prob, data = test)

Models: prob
Event: High (52.5%)



	month	Sea_Level_PressureIn	Normalized_Consumption	Weekday	Base_Hour_Class	pred	OutlierDay
989	7	29.86	0.2181818182	1	Low	High	False
990	7	29.89	0.2181818182	0	High	Low	False
991	7	29.86	0.1818181818	0	Low	Low	False
992	7	29.89	0.1727272727	0	Low	Low	False
993	7	29.89	0.1909090909	0	Low	Low	False
994	7	29.89	0.1727272727	0	Low	Low	False
995	7	29.92	0.2090909091	0	High	Low	False
996	7	29.92	0.2090909091	0	High	Low	False
997	7	29.95	0.2090909091	0	High	Low	False
998	7	30.01	0.1909090909	0	Low	Low	False
999	7	30.01	0.1818181818	0	Low	Low	False
1000	7	30.06	0.2000000000	1	Low	Low	False
1001	7	30.06	0.1909090909	1	Low	Low	False
1002	7	29.98	0.2000000000	1	Low	Low	False

Random Forest

```
set.seed(123)
data <- split_dataset_final[[1]]
data <- data[,c(18,6,8,9,10,12,7,14,19,20,21,22,24,26,28)]
data$Base_Hour_Class <- as.factor(data$Base_Hour_Class)
library(randomForest)
table(data$Base_Hour_Class)/nrow(data)
#High          Low
#0.5353793691 0.4646206309

#53% of the observations has target variable "High" and remaining 46% observations ta

#Now, we will split the data sample into development and validation samples.

sample.ind <- sample(2,
                     nrow(data),
                     replace = T,
                     prob = c(0.6,0.4))
cross.sell.dev <- data[sample.ind==1,]
cross.sell.val <- data[sample.ind==2,]

table(cross.sell.dev$Base_Hour_Class)/nrow(cross.sell.dev)
#        High          Low
#0.5331720105 0.4668279895

table(cross.sell.val$Base_Hour_Class)/nrow(cross.sell.val)
#High          Low
#0.5387453875 0.4612546125

class(cross.sell.dev$Base_Hour_Class)

## [1] "factor"

varNames <- names(cross.sell.dev)
# Exclude ID or Response variable
varNames <- varNames[!varNames %in% c("Base_Hour_Class")]
```

table(data$Base_Hour_Class)/nrow(data)
#High Low
#0.5353793691 0.4646206309

#53% of the observations has target variable "High" and remaining 46% observations take value "Low".

#Now, we will split the data sample into development and validation samples.

After split:

```
> table(cross.sell.dev$Base_Hour_Class)/nrow(cross.sell.dev)

      High        Low
0.5331720105 0.4668279895
```

```
> table(cross.sell.val$Base_Hour_Class)/nrow(cross.sell.val)

      High        Low
0.5387453875 0.4612546125
```

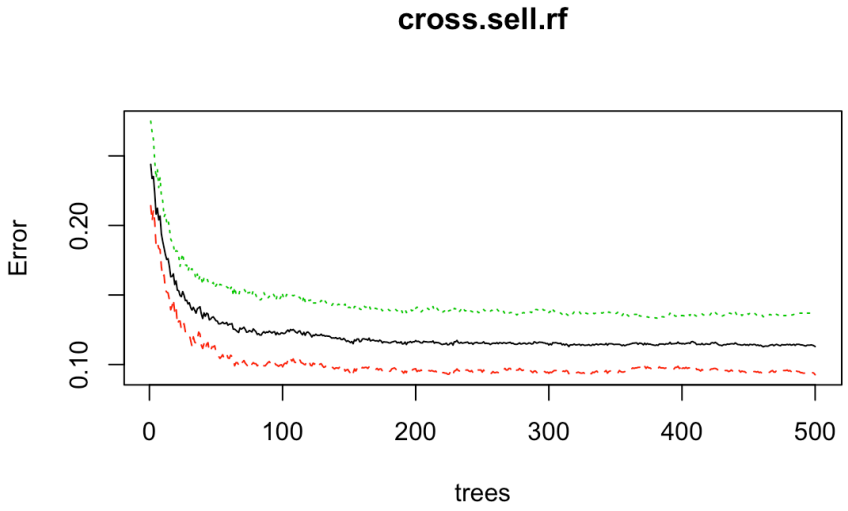
Both development and validation samples have similar target variable distribution. This is just a sample validation.

If target variable is factor, classification decision tree is built. We can check the type of response variable.

```
# add + sign between exploratory variables
varNames1 <- paste(varNames, collapse = "+")

# Add response variable and convert to a formula object
rf.form <- as.formula(paste("Base_Hour_Class", varNames1, sep = " ~ "))

cross.sell.rf <- randomForest(rf.form,
                              cross.sell.dev,
                              ntree=500,
                              importance=T)

plot(cross.sell.rf)

# Variable Importance Plot
varImpPlot(cross.sell.rf,
           sort = T,
           main="Variable Importance",
           n.var=5)

# Variable Importance Table
var.imp <- data.frame(importance(cross.sell.rf,
                                 type=2))
# make row names as columns
var.imp$Variables <- row.names(var.imp)
var.imp[order(var.imp$MeanDecreaseGini,decreasing = T),]

# Predicting response variable
cross.sell.dev$predicted.response <- predict(cross.sell.rf ,cross.sell.dev)
```

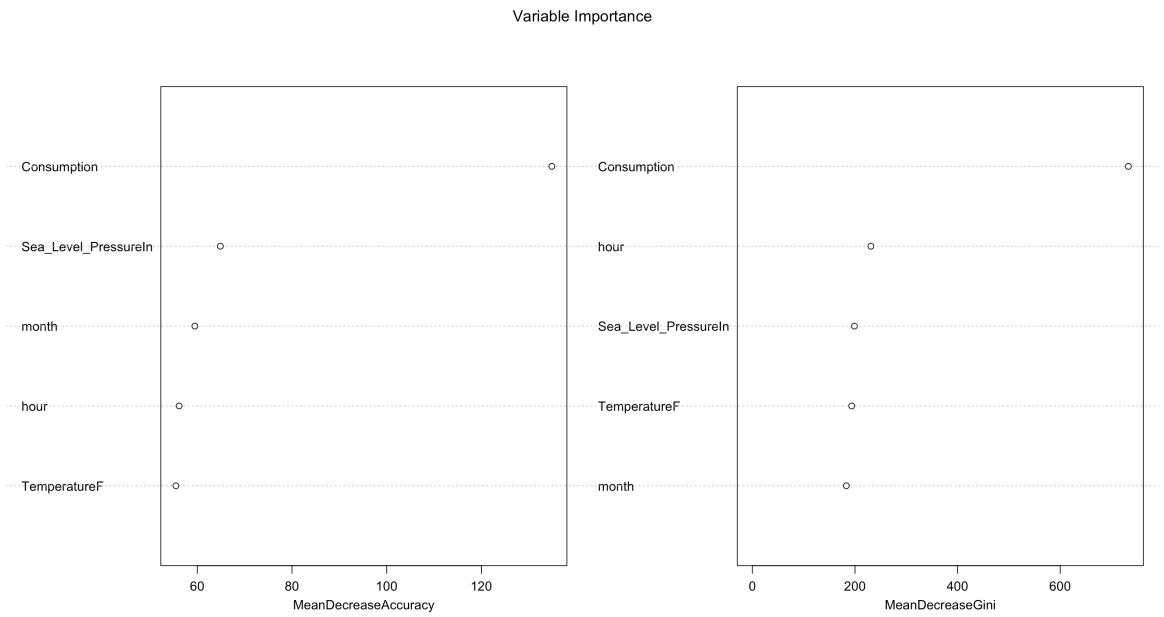
 # Add response variable and convert to a formula object

Building Random Forest using R
Now, we have a sample data and formula for building Random Forest model. Let's build 500 decision trees using Random Forest.



500 decision trees or a forest has been built using the Random Forest algorithm based learning. We can plot the error rate across decision trees. The plot seems to indicate that after 100 decision trees, there is not a significant reduction in error rate.

Variable importance plot is also a useful tool and can be plotted using **varImpPlot** function. Top 5 variables are selected and plotted



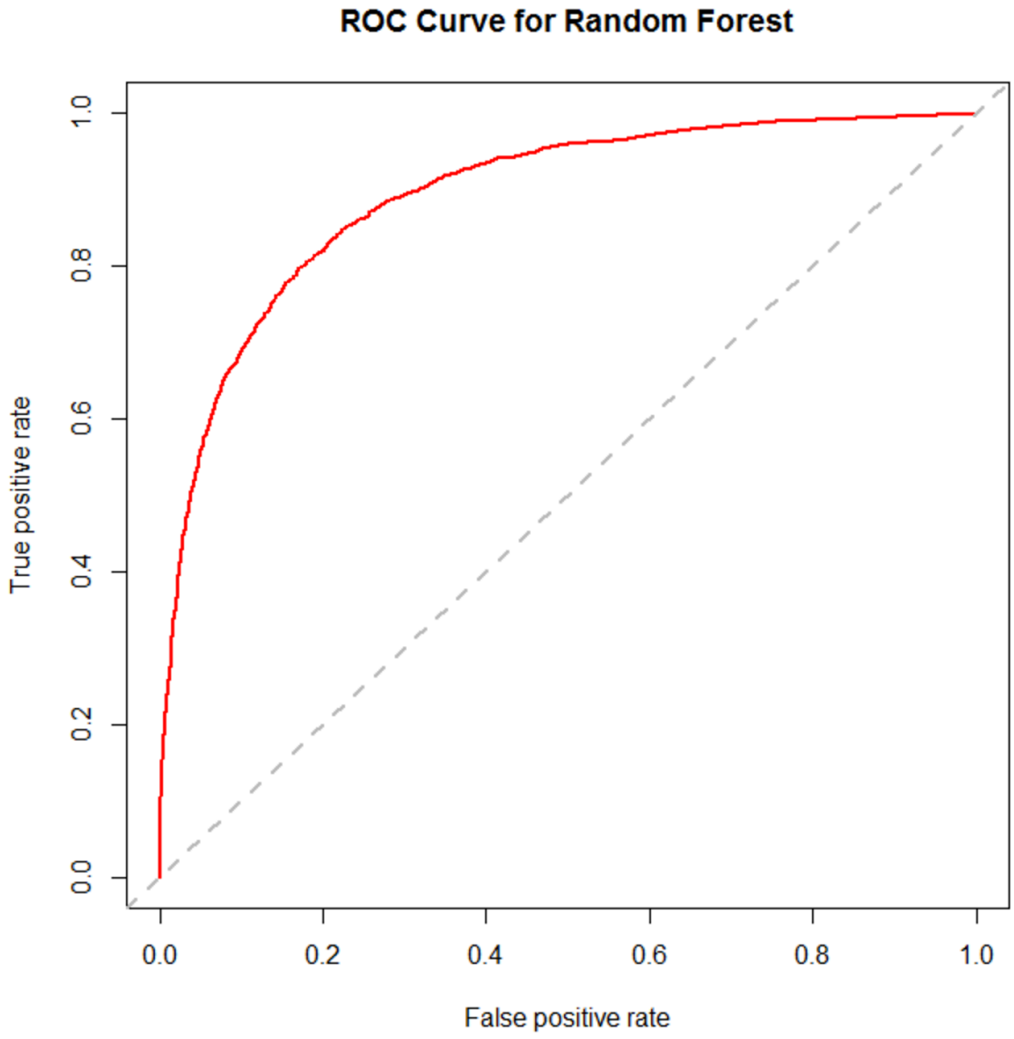
 # Predicting response variable

Generic predict function can be used for predicting response variable using Random Forest object.

cross.sell.dev$predicted.response <- predict(cross.sell.rf ,cross.sell.dev)

Confusion Matrix

confusionMatrix function from caret package can be used for creating confusion matrix based on actual response variable and predicted value.


ROC Curve for Random Forest
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

```
          Reference
Prediction High  Low
      High 1565  180
      Low   187 1320

               Accuracy : 0.8871464
                 95% CI : (0.8757678, 0.8978233)
    No Information Rate : 0.5387454
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.7730054
 Mcnemar's Test P-Value : 0.7541307

            Sensitivity : 0.8800000
            Specificity : 0.8932648
         Pos Pred Value : 0.8759124
         Neg Pred Value : 0.8968481
             Prevalence : 0.4612546
         Detection Rate : 0.4059041
   Detection Prevalence : 0.4634071
      Balanced Accuracy : 0.8866324

       'Positive' Class : Low
```

It has 88% accuracy

```
var.imp[order(var.imp$MeanDecreaseGini,decreasing = T),]

# Predicting response variable
cross.sell.dev$predicted.response <- predict(cross.sell.rf ,cross.sell.dev)

#Confusion Matrix

#confusionMatrix function from caret package can be used for creating confusion matri

# Load Library or packages
library(e1071)
library(caret)
## Loading required package: lattice
## Loading required package: ggplot2
# Create Confusion Matrix
confusionMatrix(data=cross.sell.dev$predicted.response,
                reference=cross.sell.dev$Base_Hour_Class,
                positive='Low')

# Predicting response variable
cross.sell.val$predicted.response <- predict(cross.sell.rf ,cross.sell.val)

# Create Confusion Matrix
confusionMatrix(data=cross.sell.val$predicted.response,
                reference=cross.sell.val$Base_Hour_Class,
                positive='Low')

```

```
> head(cross.sell.val)
  TemperatureF hour dayOfWeek Weekday month Holiday Consumption
1         33.8    1         2       1     1       1          25
2         33.8    3         2       1     1       1          25
3         35.6    4         2       1     1       1          25
4         37.4    7         2       1     1       1          27
5         37.4   10         2       1     1       1          26
6         37.4   12         2       1     1       1          26
  area_floor._m.sqr Dew_PointF Humidity Sea_Level_PressureIn VisibilityMPH
1               110       32.0       93                29.42           1.9
2               110       33.8      100                29.36           4.3
3               110       33.8       93                29.33           5.0
4               110       35.6       93                29.30           6.2
5               110       37.4      100                29.30           5.0
6               110       35.6       93                29.30           6.2
  Wind_SpeedMPH WindDirDegrees Base_Hour_Class
1          17.3            150             Low
2          15.0            150             Low
3          15.0            150             Low
4          12.7            180            High
5          16.1            190            High
6          15.0            190            High
  cross.sell.val$predicted.response OutlierDay
1                               Low       True
2                               Low       True
3                               Low       True
4                              High       True
5                              High       True
6                              High       True
```

KNN_Classification

```r
#look up the structure of the dataset#
data_knn <- split_dataset_final$`5198.1`
data_knn <- data_knn[,c(18,6,8,9,10,12,16,14,19,20,21,22,24,26,28)]

str(data_knn)
head(data_knn)
table(data_knn$Base_Hour_Class)

#Mix up dataset#
set.seed(9850)
group<- runif(nrow(data_knn))
data_knn<- data_knn[order(group),]
summary(data_knn[, c(1:14)])

#Normalization#
normalize<-function(x)(return((x-min(x))/(max(x)-min(x))))
data_knn_n<-as.data.frame(lapply(data_knn[,c(1:7,9:14)],normalize))
summary(data_knn_n)

#Separate training&testing dataset
data_knn_n_train<-data_knn_n[1:7000, ]
data_knn_n_test<-data_knn_n[7001:8211, ]
data_knn_train_target<-data_knn[1:7000,15]
data_knn_test_target<-data_knn[7001:8211,15]

# fit in knn algorithm
require(class)
m1<- knn(train= data_knn_n_train, test= data_knn_n_test, cl=as.factor(data_knn_train_target[['Base_Hour_Class']]), k=92)
table(as.factor(data_knn_test_target[['Base_Hour_Class']]),m1)

#install.packages("gmodels")

library(gmodels)
#result and comparison

CrossTable(x=as.factor(data_knn_test_target[['Base_Hour_Class']]), y=m1, prop.chisq = FALSE)

```

> table(data_knn$Base_Hour_Class)

High Low
4396 3815

#Mix up dataset#

Since it is a very nice organized dataset, we need to mix the entire dataset up. To do this, we have to create a 150 random number dataset and make the original dataset re-organized follow the order of the random number dataset.

```
> summary(data_knn[, c(1:14)])
  TemperatureF            hour           dayOfWeek         Weekday              month              Holiday
 Min.   :-22.00000   Min.   : 0.00000   Min.   :0.00000   Min.   :0.0000000   Min.   : 1.000000   Min.   :0.00000000
 1st Qu.: 30.20000   1st Qu.: 5.00000   1st Qu.:1.00000   1st Qu.:0.0000000   1st Qu.: 3.000000   1st Qu.:0.00000000
 Median : 44.60000   Median :11.00000   Median :3.00000   Median :1.0000000   Median : 6.000000   Median :0.00000000
 Mean   : 44.13898   Mean   :11.49665   Mean   :3.00475   Mean   :0.7131896   Mean   : 6.158568   Mean   :0.04676653
 3rd Qu.: 59.00000   3rd Qu.:17.00000   3rd Qu.:5.00000   3rd Qu.:1.0000000   3rd Qu.: 9.000000   3rd Qu.:0.00000000
 Max.   : 86.00000   Max.   :23.00000   Max.   :6.00000   Max.   :1.0000000   Max.   :12.000000   Max.   :1.00000000
 Normalized_Consumption area_floor._m.sqr   Dew_PointF         Humidity         Sea_Level_PressureIn VisibilityMPH
 Min.   :0.0000000      Min.   :110       Min.   :-27.4000   Min.   : 12.00000   Min.   :28.74000     Min.   : 0.100000
 1st Qu.:0.2000000      1st Qu.:110       1st Qu.: 24.8000   1st Qu.: 64.00000   1st Qu.:29.74000     1st Qu.: 6.200000
 Median :0.2181818      Median :110       Median : 37.4000   Median : 82.00000   Median :29.92000     Median : 6.200000
 Mean   :0.2244871      Mean   :110       Mean   : 36.3288   Mean   : 76.96395   Mean   :29.89768     Mean   : 5.880088
 3rd Qu.:0.2363636      3rd Qu.:110       3rd Qu.: 50.0000   3rd Qu.: 93.00000   3rd Qu.:30.09000     3rd Qu.: 6.200000
 Max.   :0.6000000      Max.   :110       Max.   : 68.0000   Max.   :100.00000   Max.   :30.54000     Max.   :31.000000
 Wind_SpeedMPH      WindDirDegrees
 Min.   : 1.200000   Min.   :  0.000
 1st Qu.: 4.600000   1st Qu.: 90.000
 Median : 8.100000   Median :200.000
 Mean   : 8.250432   Mean   :183.393
 3rd Qu.:11.500000   3rd Qu.:270.000
 Max.   :33.400000   Max.   :360.000
```

#Normalization#

```
> summary(data_knn_n)
  TemperatureF            hour           dayOfWeek         Weekday              month              Holiday
 Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.00000000
 1st Qu.:0.4833333   1st Qu.:0.2173913   1st Qu.:0.1666667   1st Qu.:0.0000000   1st Qu.:0.1818182   1st Qu.:0.00000000
 Median :0.6166667   Median :0.4782609   Median :0.5000000   Median :1.0000000   Median :0.4545455   Median :0.00000000
 Mean   :0.6123980   Mean   :0.4998544   Mean   :0.5007916   Mean   :0.7131896   Mean   :0.4689607   Mean   :0.04676653
 3rd Qu.:0.7500000   3rd Qu.:0.7391304   3rd Qu.:0.8333333   3rd Qu.:1.0000000   3rd Qu.:0.7272727   3rd Qu.:0.00000000
 Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000   Max.   :1.00000000
 Normalized_Consumption   Dew_PointF         Humidity         Sea_Level_PressureIn VisibilityMPH
 Min.   :0.0000000      Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000    Min.   :0.0000000
 1st Qu.:0.3333333      1st Qu.:0.5471698   1st Qu.:0.5909091   1st Qu.:0.5555556    1st Qu.:0.1974110
 Median :0.3636364      Median :0.6792453   Median :0.7954545   Median :0.6555556    Median :0.1974110
 Mean   :0.3741452      Mean   :0.6680168   Mean   :0.7382267   Mean   :0.6431582    Mean   :0.1870579
 3rd Qu.:0.3939394      3rd Qu.:0.8113208   3rd Qu.:0.9204545   3rd Qu.:0.7500000    3rd Qu.:0.1974110
 Max.   :1.0000000      Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000    Max.   :1.0000000
 Wind_SpeedMPH      WindDirDegrees
 Min.   :0.0000000   Min.   :0.0000000
 1st Qu.:0.1055901   1st Qu.:0.2500000
 Median :0.2142857   Median :0.5555556
 Mean   :0.2189575   Mean   :0.5094250
 3rd Qu.:0.3198758   3rd Qu.:0.7500000
 Max.   :1.0000000   Max.   :1.0000000
```

#Separate training&testing dataset

 # fit in knn algorithm

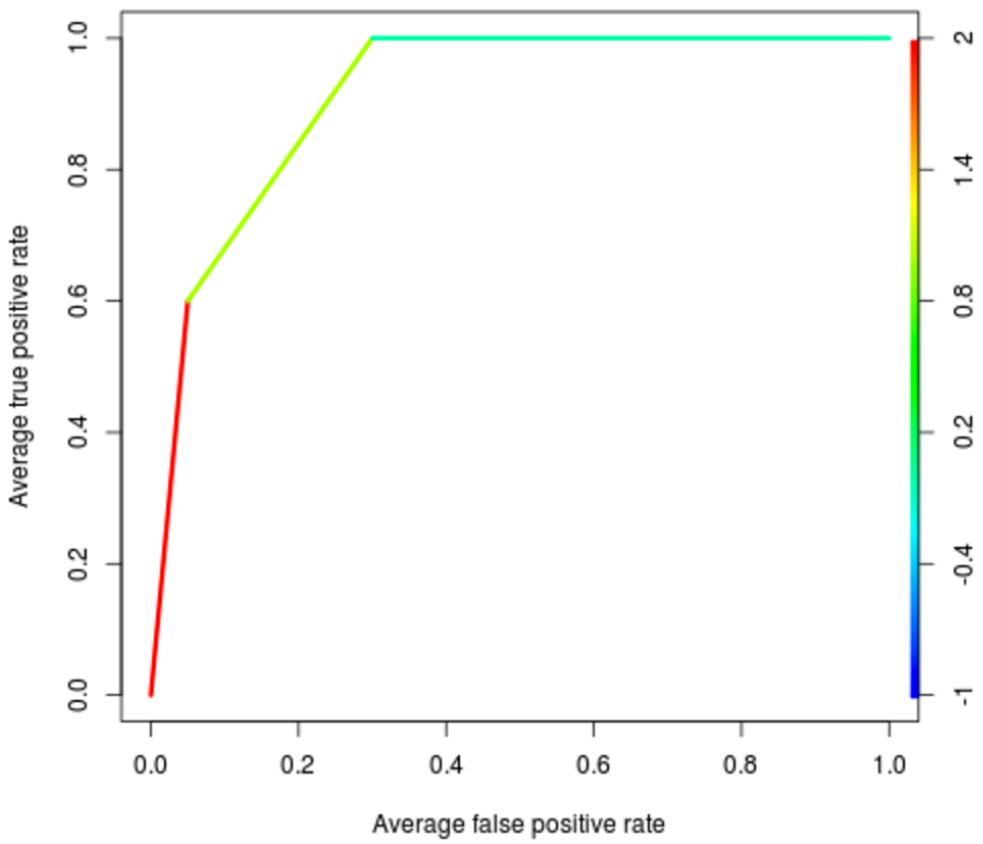
We use knn() function to do the training, we need to fit in the training data frame, the test data frame and the training target variables. before that we have to choose a k value, as k is a place holder for how many nearest neighbor that you want the algorithm to use. The rule of thumb is to take the square root of the total number of observations you have, in this case should be 92

To see how well the model predicted, we use table() to see the difference between predict result and what species they actually are.

```
> table(as.factor(data_knn_test_target[['Base_Hour_Class']]),m1)
    m1

        High L ow
 High  463     170
 Low   221     357
```

```
> CrossTable(x=as.factor(data_knn_test_target[['Base_Hour_Class']]), y=m1, prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  1211


                                                    | m1
as.factor(data_knn_test_target[["Base_Hour_Class"]]) |      High |       Low | Row Total |
-----------------------------------------------------|-----------|-----------|-----------|
                                                High |       463 |       170 |       633 |
                                                     |     0.731 |     0.269 |     0.523 |
                                                     |     0.677 |     0.323 |           |
                                                     |     0.382 |     0.140 |           |
-----------------------------------------------------|-----------|-----------|-----------|
                                                 Low |       221 |       357 |       578 |
                                                     |     0.382 |     0.618 |     0.477 |
                                                     |     0.323 |     0.677 |           |
                                                     |     0.182 |     0.295 |           |
-----------------------------------------------------|-----------|-----------|-----------|
                                        Column Total |       684 |       527 |      1211 |
                                                     |     0.565 |     0.435 |           |
-----------------------------------------------------|-----------|-----------|-----------|
```

```
> Final_data[767:770,]
    TemperatureF hour dayOfWeek Weekday month Holiday Normalized_Consumption
767         39.2    1         5       1     4       0           0.1818181818
768         44.6    6         0       0    11       0           0.4090909091
769          6.8    0         1       1     3       0           0.1818181818
770         23.0   23         0       0    12       0           0.0000000000
    area_floor._m.sqr Dew_PointF Humidity Sea_Level_PressureIn VisibilityMPH
767               110       39.2      100                29.53           1.7
768               110       39.2       81                29.39           6.2
769               110       -5.8       56                30.36           6.2
770               110       19.4       86                29.86           6.2
    Wind_SpeedMPH WindDirDegrees Base_Hour_Class Prediction_value OutlierDay
767          10.4            200             Low             High      False
768          15.0            250            High              Low      False
769           9.2             50             Low              Low       True
770           5.8             20             Low              Low       True
>
```



Best Model: Random Forest with accuracy: 88%

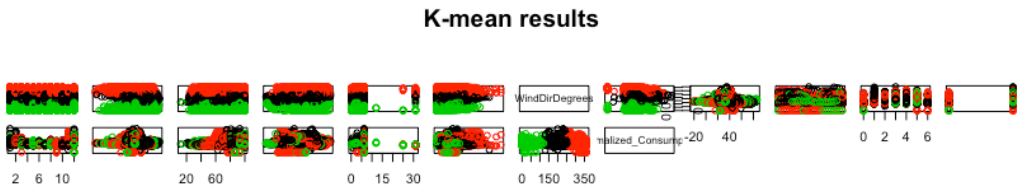
Clustering

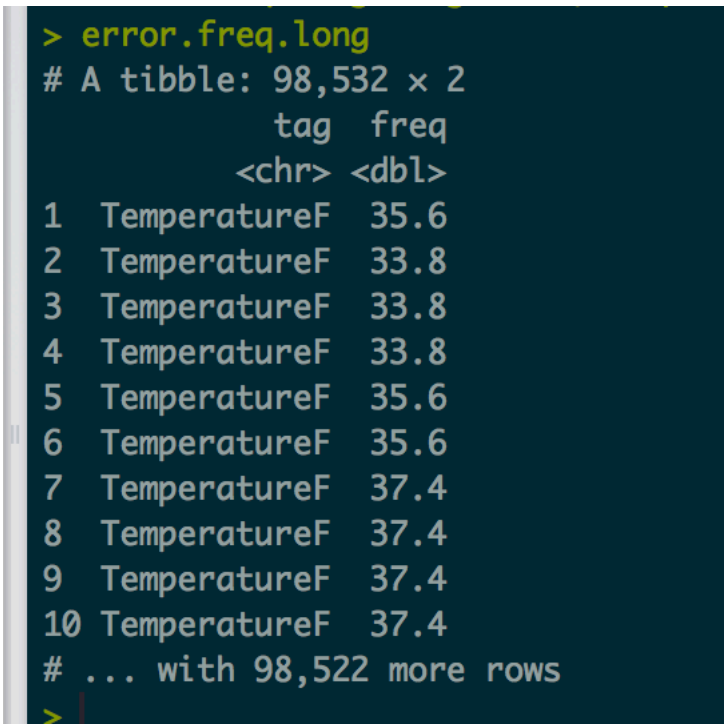
```r
data <- split_dataset_final[[1]]
data <- data[,c(18,6,8,9,10,19,20,21,22,24,26,16)]
json_data <- fromJSON(paste(readLines("config.json"), collapse=""))
#nstart tells how many times algorithm
km.out <- kmeans(data,as.numeric(json_data$Number_of_clusters),nstart = as.numeric(json_data$nstart))
names(km.out)

#kmeans results
km.out$cluster

#cluster tagging
View(cbind(data,km.out$cluster))
View(data)
#k-mean results with k=3

#Scatter plot matrix
plot(data, col=(km.out$cluster), main = "K-mean results")

error.freq.long <- gather(data, tag, freq)


#### Hierarchical clustering
rough_data <- sample(1:nrow(data),round(0.60*nrow(data)))

k_mean_data <- data[rough_data,]
kmeans_testdata <- data[-rough_data,]

#Scaling the data
data=scale(k_mean_data)
# Complete linkage type
hc.complete=hclust(dist(data),method="complete")
# Average linkage type
hc.average=hclust(dist(data),method="average")

```

```r
kmeans_testdata <- data[-rough_data,]

#Scaling the data
data=scale(k_mean_data)
# Complete linkage type
hc.complete=hclust(dist(data),method="complete")
# Average linkage type
hc.average=hclust(dist(data),method="average")

par(mfrow=c(1,2)) #Plotting in a matrix form
plot(hc.complete,main='Complete')


### Bend Graph
rough_data <- sample(1:nrow(data),round(0.75*nrow(data)))
k_mean_data <- data[rough_data,]
nrow(k_mean_data)
names(data)
data=scale(k_mean_data)
#Scaling the data
wss <- (nrow(data)-1)*sum(apply(data,2,var))
for(i in 2:15){
  wss[i] <- sum(kmeans(data,centers = i)$withinss)
}

plot(1:15, wss,type="b",xlab = "Number of clusters", ylab="Within groups sum of squares")
```

#Scatter plot matrix

plot(data, col=(km.out$cluster), main = "K-mean results")





```
> error.freq.long
# A tibble: 98,532 × 2
            tag  freq
          <chr> <dbl>
1  TemperatureF  35.6
2  TemperatureF  33.8
3  TemperatureF  33.8
4  TemperatureF  33.8
5  TemperatureF  35.6
6  TemperatureF  35.6
7  TemperatureF  37.4
8  TemperatureF  37.4
9  TemperatureF  37.4
10 TemperatureF  37.4
# ... with 98,522 more rows
>
```

Complete linkage type

> hc.complete

Call:
hclust(d = dist(data), method = "complete")

Cluster method : complete
Distance : euclidean
Number of objects: 4927

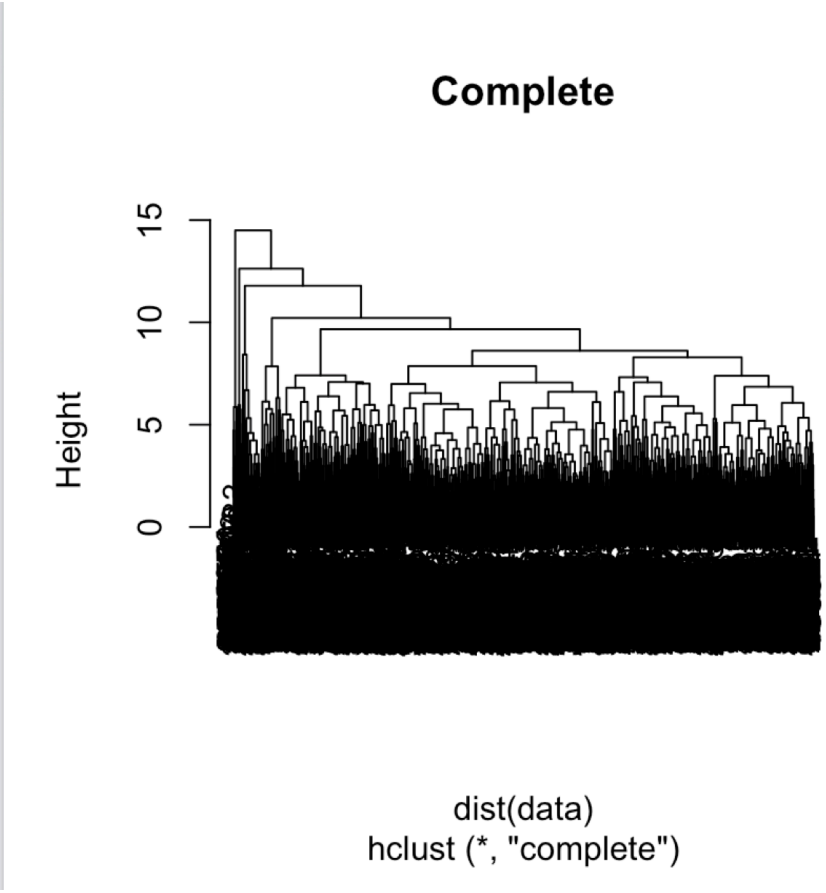
Average linkage type

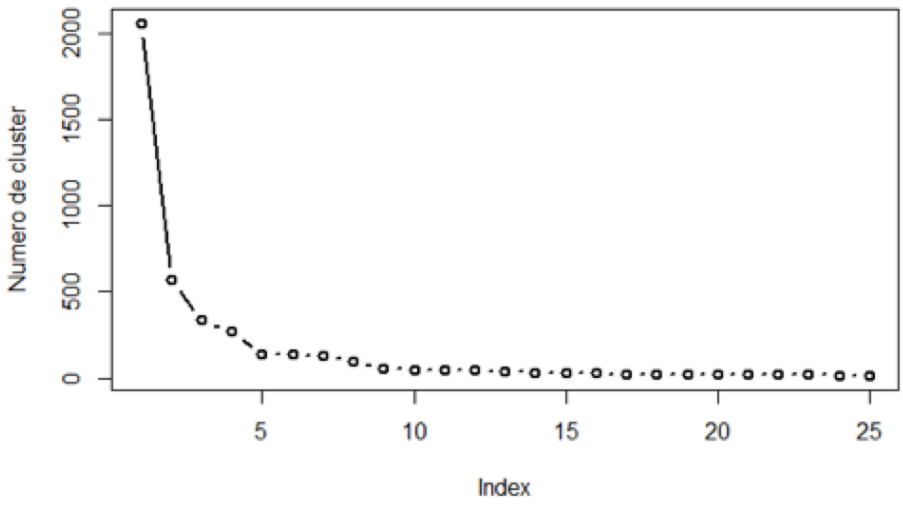
> hc.average

Call:
hclust(d = dist(data), method = "average")

Cluster method : average
Distance : euclidean
Number of objects: 4927

#Plotting in a matrix form
Hierarchical clustering




Numero de cluster
0
500
1000
1500
2000
5
10
15
20
25
Index