The University of Texas at Dallas
Naveen Jindal School of Management

Supermarket Organic product data Analysis

A supermarket is offering a new line of organic products. The supermarket's management wants to determine which customers are likely to purchase these products. The supermarket has a customer loyalty program. As an initial buyer incentive plan, the supermarket provided coupons for the organic products to all of the loyalty program participants and collected data that includes whether these customers purchased any of the organic products.

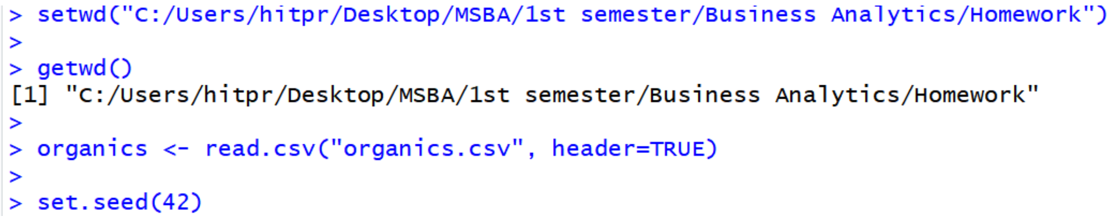
The ORGANICS data set contains 13 variables and over 22,000 observations. The variables in the data set are shown below with the appropriate roles and levels:

Name	Model Role	Data Type	Description
ID	ID	Categoric	Customer loyalty identification number
DemAffl	Input	Numeric	Affluence grade on a scale from 1 to 30
DemAge	Input	Numeric	Age, in years
DemCluster	Rejected	Categoric	Type of residential neighborhood
DemClusterGroup	Input	Categoric	Neighborhood group
DemGender	Input	Categoric	M = male, F = female, U = unknown
DemRegion	Input	Categoric	Geographic region
DemTVReg	Input	Categoric	Television region
PromClass	Input	Categoric	Loyalty status: tin, silver, gold, or platinum
PromSpend	Input	Numeric	Total amount spent
PromTime	Input	Numeric	Time as loyalty card member
TargetBuy	Target	Numeric	Organics purchased? 1 = Yes, 0 = No
TargetAmt	Rejected	Numeric	Number of organic products purchased

Although two target variables are listed, these exercises concentrate on the binary target variable TargetBuy.

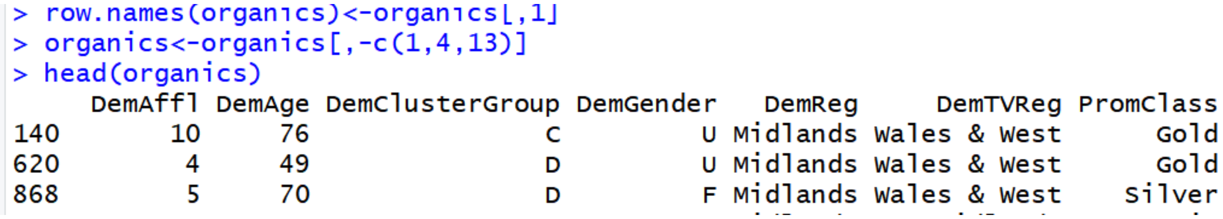
Data cleaning and missing value imputation:

Install packages “rpart” and “rpart.plot”. Import required libraries. Import the data file organics.csv and set seed to 42.



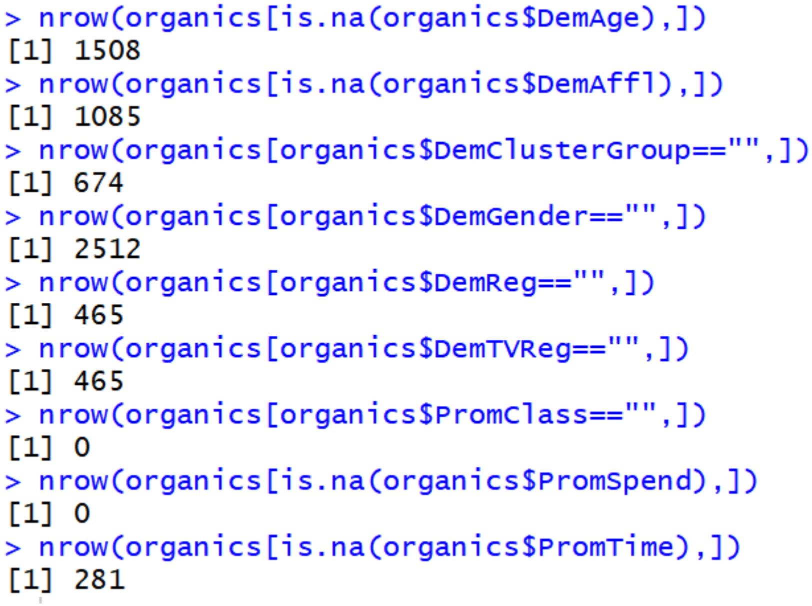
```
> setwd("C:/Users/hitpr/Desktop/MSBA/1st semester/Business Analytics/Homework")
>
> getwd()
[1] "C:/Users/hitpr/Desktop/MSBA/1st semester/Business Analytics/Homework"
>
> organics <- read.csv("organics.csv", header=TRUE)
>
> set.seed(42)
```

Set row names as the ID for organics i.e. column 1. Then remove the variables- ID, DemCluster and Target Amount i.e columns 1, 4 and 13 respectively.



```
> row.names(organics)<-organics[,1]
> organics<-organics[,-c(1,4,13)]
> head(organics)
     DemAffl DemAge DemClusterGroup DemGender   DemReg     DemTVReg PromClass
140       10     76               C         U Midlands Wales & West      Gold
620        4     49               D         U Midlands Wales & West      Gold
868        5     70               D         F Midlands Wales & West    Silver
```

Now check the columns that have missing values or NA. Display the count of null values for each column

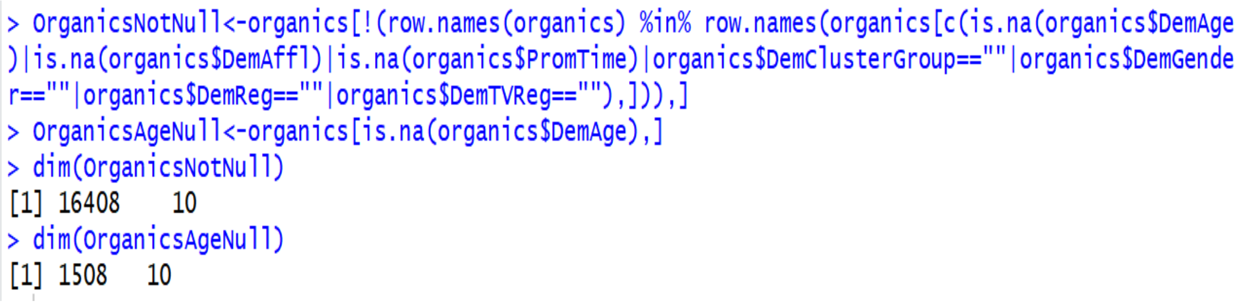


```
> nrow(organics[is.na(organics$DemAge),])
[1] 1508
> nrow(organics[is.na(organics$DemAffl),])
[1] 1085
> nrow(organics[organics$DemClusterGroup=="",])
[1] 674
> nrow(organics[organics$DemGender=="",])
[1] 2512
> nrow(organics[organics$DemReg=="",])
[1] 465
> nrow(organics[organics$DemTVReg=="",])
[1] 465
> nrow(organics[organics$PromClass=="",])
[1] 0
> nrow(organics[is.na(organics$PromSpend),])
[1] 0
> nrow(organics[is.na(organics$PromTime),])
[1] 281
```

Other than PromClass and PromSpend all other variables contain null values. We need to impute values for all continuous variables using linear regression.

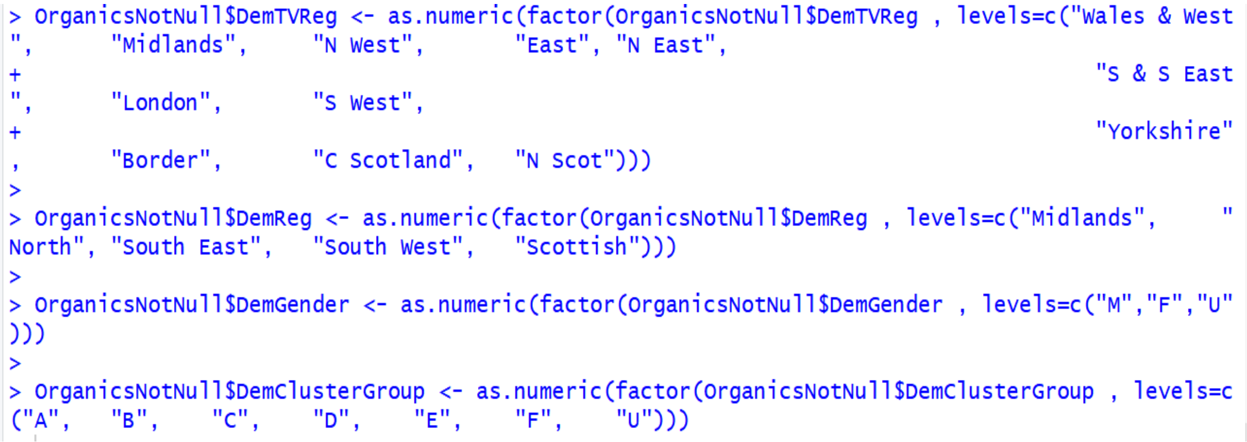
There are 3 variables- DemAge, DemAffl, PromTime that are continuous and whose values need to be imputed using linear regression. But before we build our regression model we need to build a dataframe for the model that does not contain rows that have missing value for any column.

So create two data frames- OrganicsNotNull that contains only the records that do not have any missing values or NA in any column and OrganicsAgeNull that contains only the records where DemAge is missing.



```
> OrganicsNotNull<-organics[!(row.names(organics) %in% row.names(organics[c(is.na(organics$DemAge
)|is.na(organics$DemAffl)|is.na(organics$PromTime)|organics$DemClusterGroup==""|organics$DemGende
r==""|organics$DemReg==""|organics$DemTVReg==""),])),]
> OrganicsAgeNull<-organics[is.na(organics$DemAge),]
> dim(OrganicsNotNull)
[1] 16408    10
> dim(OrganicsAgeNull)
[1] 1508   10
```

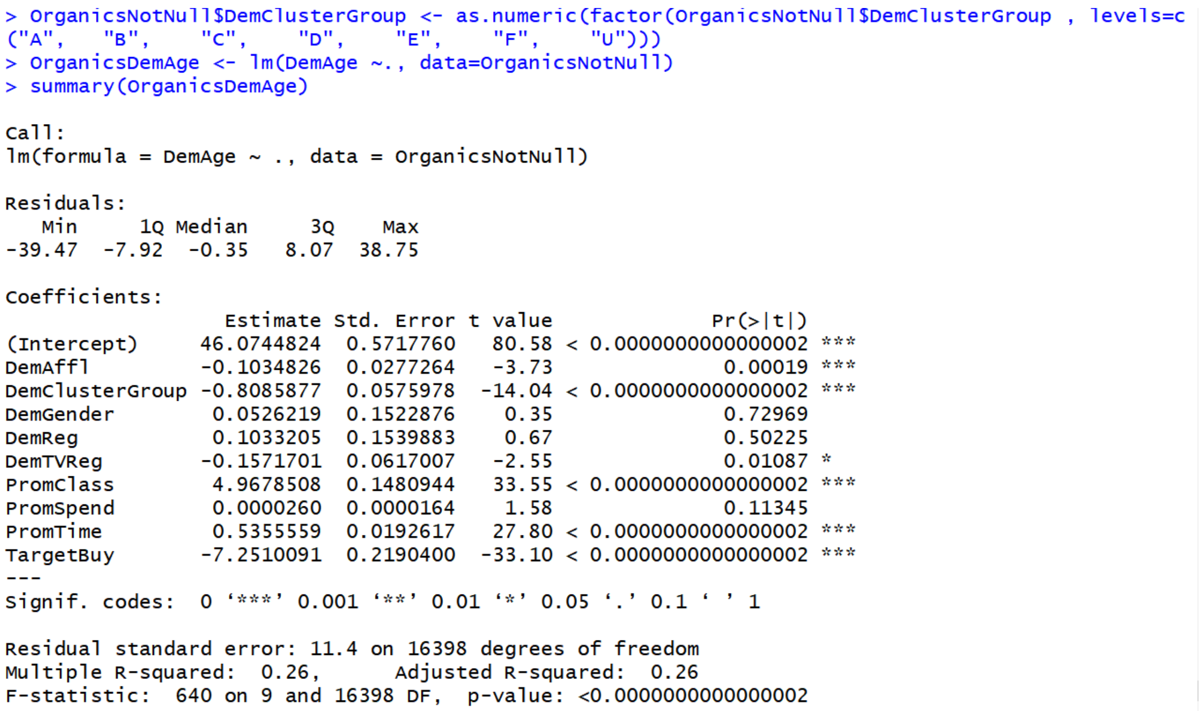
Now, to create a regression model using OrganicsNotNull we need to change all the categorical variables that have character values to numeric values by specifying levels for each.



```
> OrganicsNotNull$DemTVReg <- as.numeric(factor(OrganicsNotNull$DemTVReg , levels=c("Wales & West
",      "Midlands",     "N West",       "East", "N East",
+                                                                                   "S & S East
",      "London",       "S West",
+                                                                                   "Yorkshire"
,       "Border",       "C Scotland",   "N Scot")))
>
> OrganicsNotNull$DemReg <- as.numeric(factor(OrganicsNotNull$DemReg , levels=c("Midlands",     "
North", "South East",   "South West",   "Scottish")))
>
> OrganicsNotNull$DemGender <- as.numeric(factor(OrganicsNotNull$DemGender , levels=c("M","F","U"
)))
>
> OrganicsNotNull$DemClusterGroup <- as.numeric(factor(OrganicsNotNull$DemClusterGroup , levels=c
("A",   "B",    "C",    "D",    "E",    "F",    "U")))
```

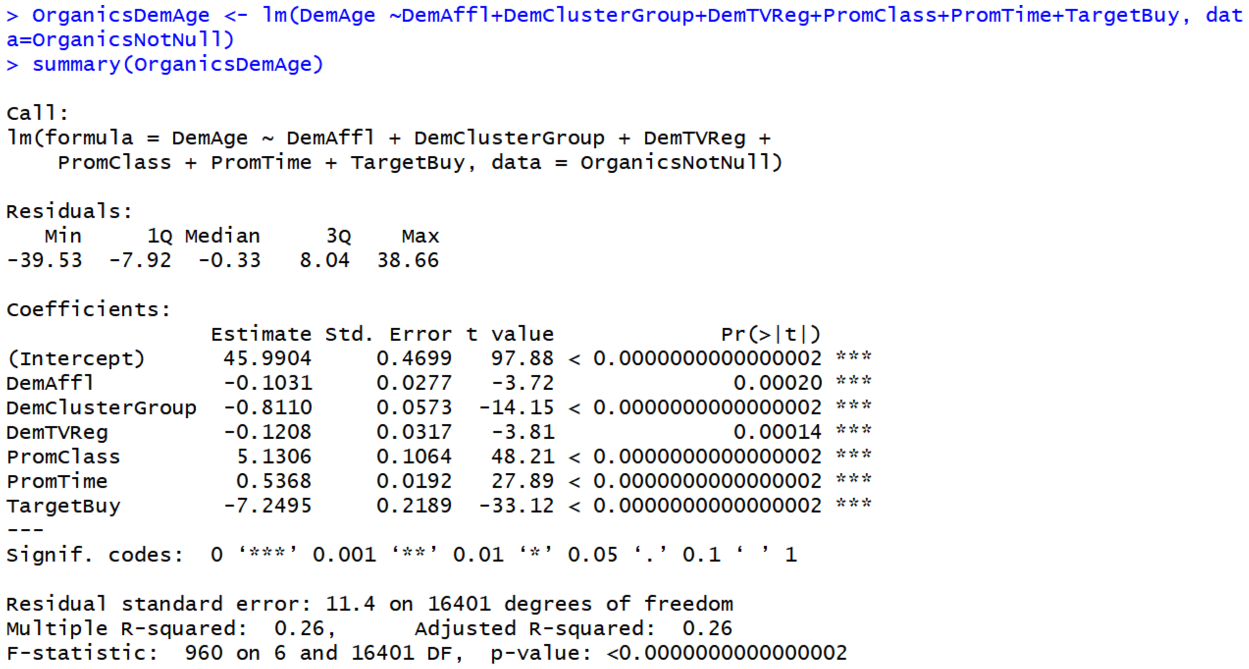
After converting to numeric values create a regression model for DemAge using all variables.

It can be seen in this model that DemAffl, DemClusterGroup, DemTVReg, PromClass, PromTime and TargetBuy are significant variables.



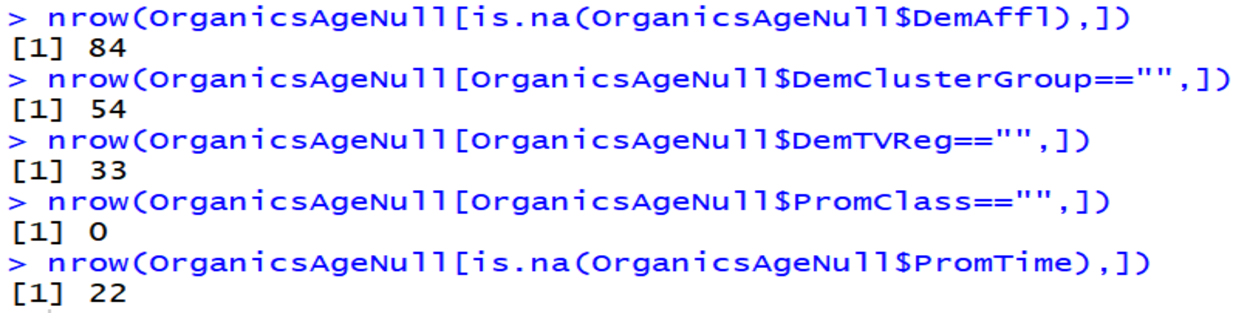
```
> OrganicsNotNull$DemClusterGroup <- as.numeric(factor(OrganicsNotNull$DemClusterGroup , levels=c
("A",    "B",     "C",     "D",     "E",     "F",     "U")))
> OrganicsDemAge <- lm(DemAge ~., data=OrganicsNotNull)
> summary(OrganicsDemAge)

Call:
lm(formula = DemAge ~ ., data = OrganicsNotNull)

Residuals:
   Min      1Q  Median      3Q     Max 
-39.47   -7.92   -0.35    8.07   38.75 

Coefficients:
                  Estimate  Std. Error t value             Pr(>|t|)    
(Intercept)     46.0744824   0.5717760   80.58 < 0.0000000000000002 ***
DemAffl         -0.1034826   0.0277264   -3.73              0.00019 ***
DemClusterGroup -0.8085877   0.0575978  -14.04 < 0.0000000000000002 ***
DemGender        0.0526219   0.1522876    0.35              0.72969    
DemReg           0.1033205   0.1539883    0.67              0.50225    
DemTVReg        -0.1571701   0.0617007   -2.55              0.01087 *  
PromClass        4.9678508   0.1480944   33.55 < 0.0000000000000002 ***
PromSpend        0.0000260   0.0000164    1.58              0.11345    
PromTime         0.5355559   0.0192617   27.80 < 0.0000000000000002 ***
TargetBuy       -7.2510091   0.2190400  -33.10 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11.4 on 16398 degrees of freedom
Multiple R-squared:  0.26,	Adjusted R-squared:  0.26 
F-statistic:  640 on 9 and 16398 DF,  p-value: <0.0000000000000002
```

Run another regression using just the significant variables.

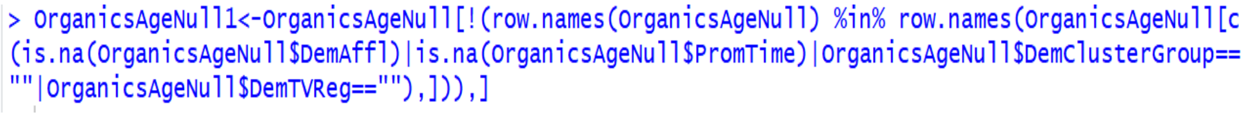


```
> OrganicsDemAge <- lm(DemAge ~DemAffl+DemClusterGroup+DemTVReg+PromClass+PromTime+TargetBuy, dat
a=OrganicsNotNull)
> summary(OrganicsDemAge)

Call:
lm(formula = DemAge ~ DemAffl + DemClusterGroup + DemTVReg + 
    PromClass + PromTime + TargetBuy, data = OrganicsNotNull)

Residuals:
   Min      1Q  Median      3Q     Max 
-39.53   -7.92   -0.33    8.04   38.66 

Coefficients:
                Estimate Std. Error t value             Pr(>|t|)    
(Intercept)      45.9904     0.4699   97.88 < 0.0000000000000002 ***
DemAffl          -0.1031     0.0277   -3.72              0.00020 ***
DemClusterGroup  -0.8110     0.0573  -14.15 < 0.0000000000000002 ***
DemTVReg         -0.1208     0.0317   -3.81              0.00014 ***
PromClass         5.1306     0.1064   48.21 < 0.0000000000000002 ***
PromTime          0.5368     0.0192   27.89 < 0.0000000000000002 ***
TargetBuy        -7.2495     0.2189  -33.12 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11.4 on 16401 degrees of freedom
Multiple R-squared:  0.26,	Adjusted R-squared:  0.26 
F-statistic:  960 on 6 and 16401 DF,  p-value: <0.0000000000000002
```

We need to predict DemAge in the data frame we defined earlier i.e. OrganicsAgeNull. However, if there are rows with missing values in columns that are used in the model we created, we cannot predict the DemAge value for these rows. So first check the columns that have missing values in OrganicsAgeNull and remove these rows. Create a new data frame OrganicsAgeNull1 where you want to predict DemAge.

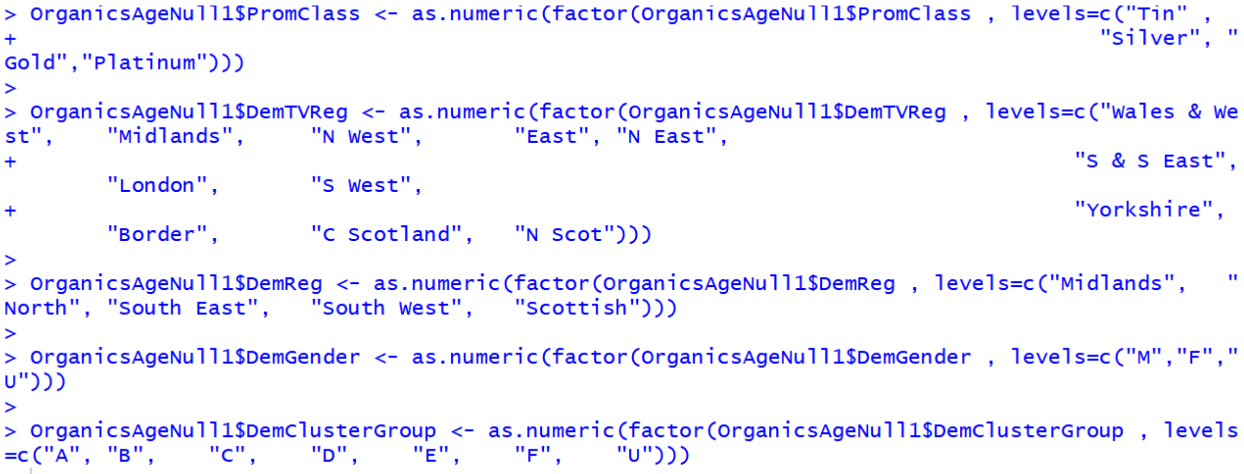


```
> nrow(OrganicsAgeNull[is.na(OrganicsAgeNull$DemAffl),])
[1] 84
> nrow(OrganicsAgeNull[OrganicsAgeNull$DemClusterGroup=="",])
[1] 54
> nrow(OrganicsAgeNull[OrganicsAgeNull$DemTVReg=="",])
[1] 33
> nrow(OrganicsAgeNull[OrganicsAgeNull$PromClass=="",])
[1] 0
> nrow(OrganicsAgeNull[is.na(OrganicsAgeNull$PromTime),])
[1] 22
```



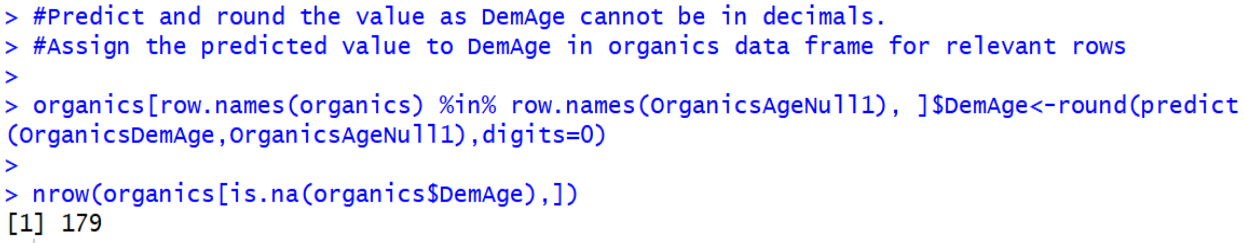
```
> OrganicsAgeNull1<-OrganicsAgeNull[!(row.names(OrganicsAgeNull) %in% row.names(OrganicsAgeNull[c
(is.na(OrganicsAgeNull$DemAffl)|is.na(OrganicsAgeNull$PromTime)|OrganicsAgeNull$DemClusterGroup==
""|OrganicsAgeNull$DemTVReg==""),])),]
```

Before running prediction we first need to convert the categorical variables from characters to numeric as we did while developing the model. This is because the model was developed on predictor numeric variables and will therefore rum on numeric variables used as predictors.



```
> OrganicsAgeNull1$PromClass <- as.numeric(factor(OrganicsAgeNull1$PromClass , levels=c("Tin" ,
+                                                                        "Silver", "
Gold","Platinum")))
>
> OrganicsAgeNull1$DemTVReg <- as.numeric(factor(OrganicsAgeNull1$DemTVReg , levels=c("Wales & We
st",    "Midlands",     "N West",       "East", "N East",
+                                                                      "S & S East",
        "London",       "S West",
+                                                                      "Yorkshire",
        "Border",       "C Scotland",   "N Scot")))
>
> OrganicsAgeNull1$DemReg <- as.numeric(factor(OrganicsAgeNull1$DemReg , levels=c("Midlands",   "
North", "South East",   "South West",   "Scottish")))
>
> OrganicsAgeNull1$DemGender <- as.numeric(factor(OrganicsAgeNull1$DemGender , levels=c("M","F","
U")))
>
> OrganicsAgeNull1$DemClusterGroup <- as.numeric(factor(OrganicsAgeNull1$DemClusterGroup , levels
=c("A", "B",    "C",    "D",    "E",    "F",    "U")))
```

#Predict on OrganicsAgeNull1 and round the values, as DemAge cannot be in decimals. Assign the predicted value to DemAge in organics data frame to only relevant rows

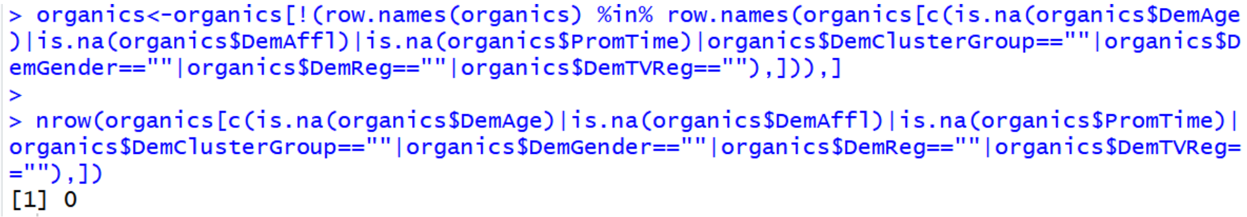


```
> #Predict and round the value as DemAge cannot be in decimals.
> #Assign the predicted value to DemAge in organics data frame for relevant rows
>
> organics[row.names(organics) %in% row.names(OrganicsAgeNull1), ]$DemAge<-round(predict
(OrganicsDemAge,OrganicsAgeNull1),digits=0)
>
> nrow(organics[is.na(organics$DemAge),])
[1] 179
```

As can be seen, out of 1508 missing values of DemAge 1329 missing values have been predicted and 179 values are remaining. These 179 rows are the ones that have some missing predictor variable and hence DemAge cannot be predicted.

Carry out the same process for DemAffl and PromTime exactly as we did for DemAge to impute their respective values.

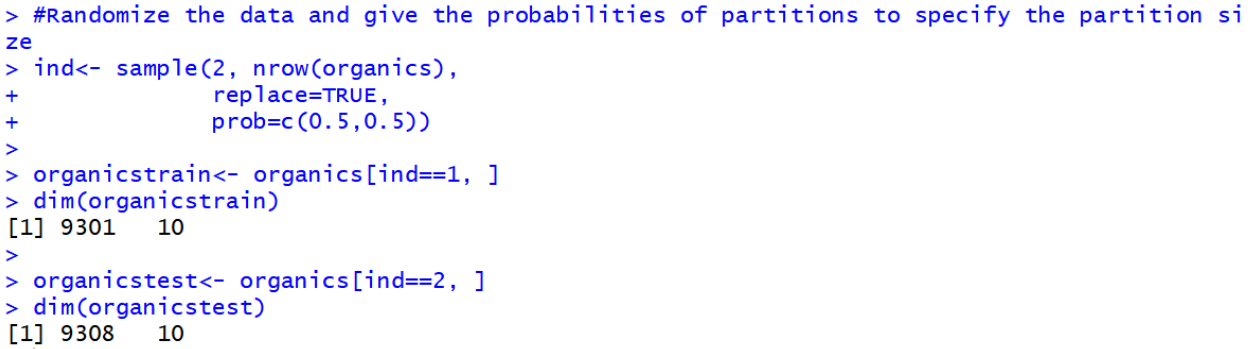
Since the loss of rows is minimized to an extent we can remove the remaining rows with missing values for other columns (including categorical variables) and check if there are still any missing values after doing this process.



```
> organics<-organics[!(row.names(organics) %in% row.names(organics[c(is.na(organics$DemAge
)|is.na(organics$DemAffl)|is.na(organics$PromTime)|organics$DemClusterGroup==""|organics$D
emGender==""|organics$DemReg==""|organics$DemTVReg==""),])),]
>
> nrow(organics[c(is.na(organics$DemAge)|is.na(organics$DemAffl)|is.na(organics$PromTime)|
organics$DemClusterGroup==""|organics$DemGender==""|organics$DemReg==""|organics$DemTVReg=
=""),])
[1] 0
```

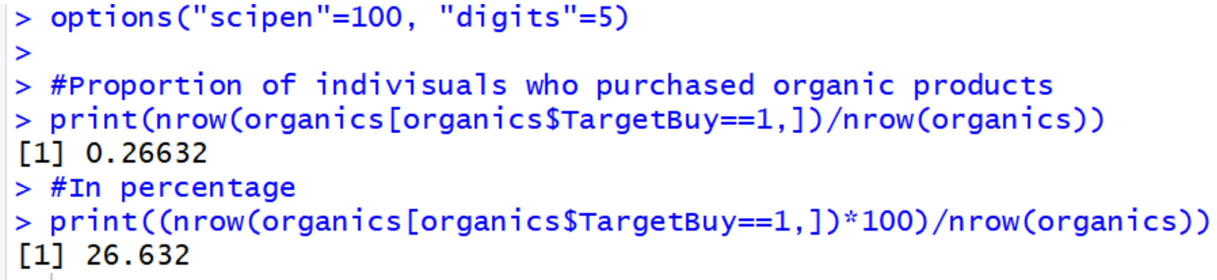
After the data cleaning and imputation is done we can now proceed to our main tasks i.e. applying classification.

1.> Randomize the data and give the probabilities of partitions to specify the partition size i.e.0.5 for each.



```
> #Randomize the data and give the probabilities of partitions to specify the partition si
ze
> ind<- sample(2, nrow(organics),
+              replace=TRUE,
+              prob=c(0.5,0.5))
>
> organicstrain<- organics[ind==1, ]
> dim(organicstrain)
[1] 9301   10
>
> organicstest<- organics[ind==2, ]
> dim(organicstest)
[1] 9308   10
```

2.> Use scipen to remove scientific notation from any values so it does not appear in terms of "e" i.e. exponent of 10. Then examine the distribution of the target variables. The proportion of individuals who purchased organic products is 0.26632 i.e. 26.632%

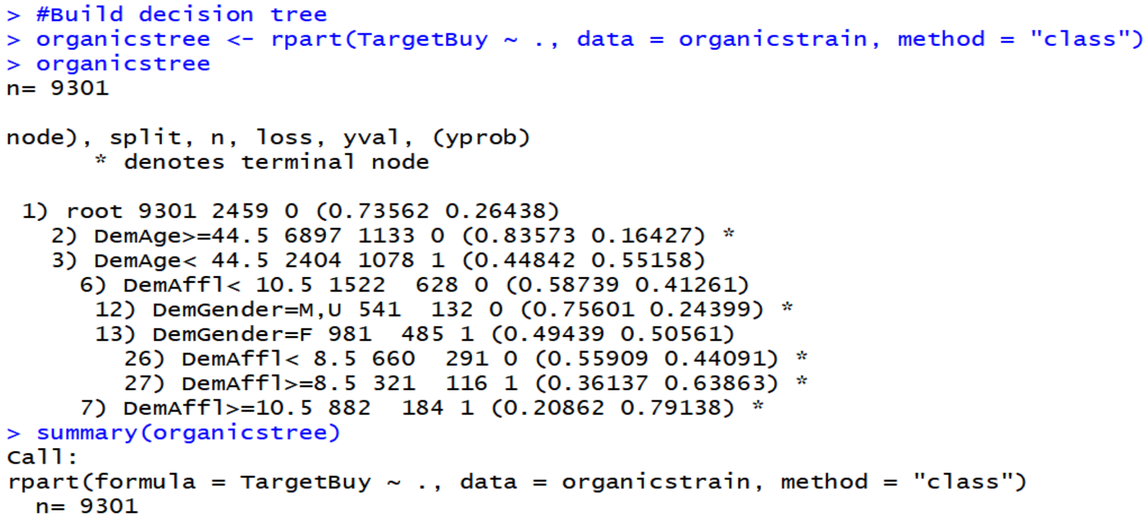


```
> options("scipen"=100, "digits"=5)
>
> #Proportion of indisivuals who purchased organic products
> print(nrow(organics[organics$TargetBuy==1,])/nrow(organics))
[1] 0.26632
> #In percentage
> print((nrow(organics[organics$TargetBuy==1,])*100)/nrow(organics))
[1] 26.632
```

3.> Only TargetBuy will be used for this analysis and should have a role of Target. This is because in the Assignment question it is specifically stated that "The supermarket's management wants to determine which customers are likely to purchase these products". This means that they just want to know if the product will be purchased(1) or not(0). So the result should be a binary variable i.e TargetBuy and therefore can be predicted using binary decision tree classification or logistic regression.

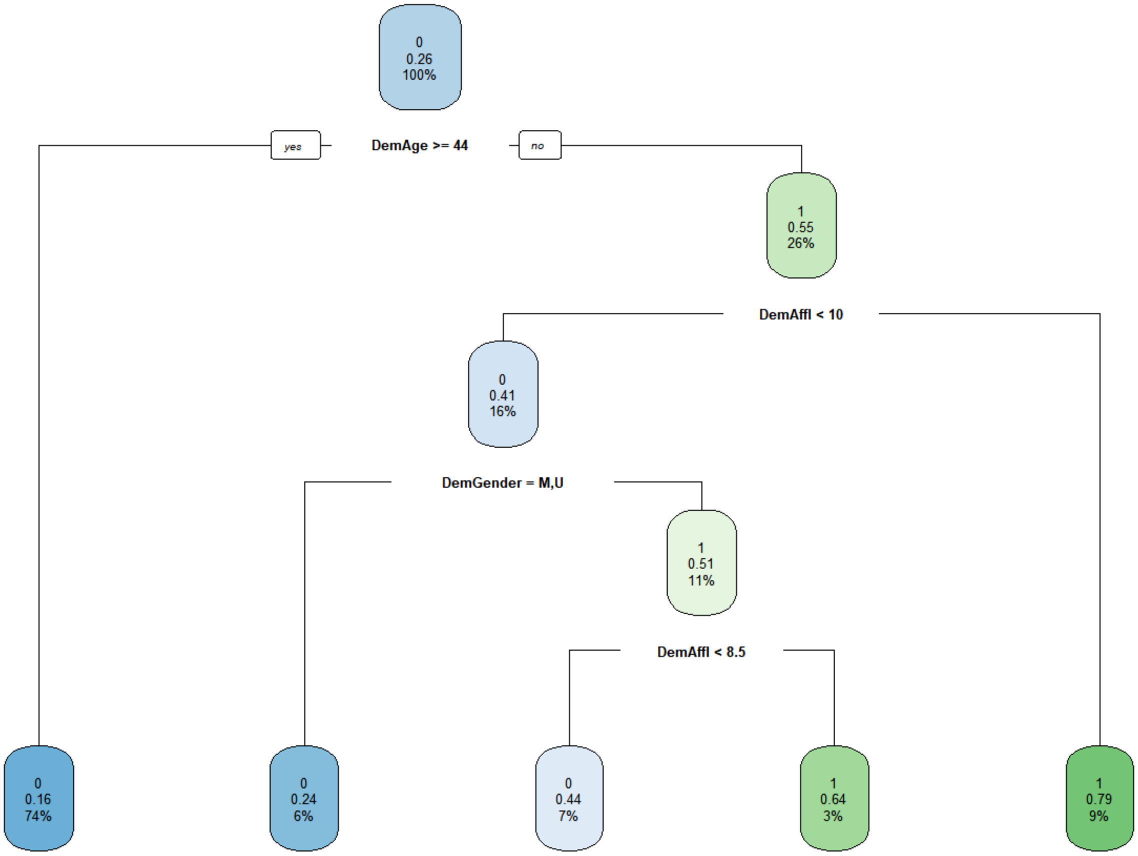
TargetAmt should not be used as an input for a model used to predict TargetBuy because these two are directly related. TargetAmt=0 means TargetBuy=0 and TargetAmt=1,2,3 means TargetBuy=1.

4.> Seed is already set to 42. Implementing decision tree on training data.



```
> #Build decision tree
> organicstree <- rpart(TargetBuy ~ ., data = organicstrain, method = "class")
> organicstree
n= 9301

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 9301 2459 0 (0.73562 0.26438)
   2) DemAge>=44.5 6897 1133 0 (0.83573 0.16427) *
   3) DemAge< 44.5 2404 1078 1 (0.44842 0.55158)
     6) DemAffl< 10.5 1522  628 0 (0.58739 0.41261)
      12) DemGender=M,U 541  132 0 (0.75601 0.24399) *
      13) DemGender=F 981  485 1 (0.49439 0.50561)
        26) DemAffl< 8.5 660  291 0 (0.55909 0.44091) *
        27) DemAffl>=8.5 321  116 1 (0.36137 0.63863) *
     7) DemAffl>=10.5 882  184 1 (0.20862 0.79138) *
> summary(organicstree)
Call:
rpart(formula = TargetBuy ~ ., data = organicstrain, method = "class")
  n= 9301
```

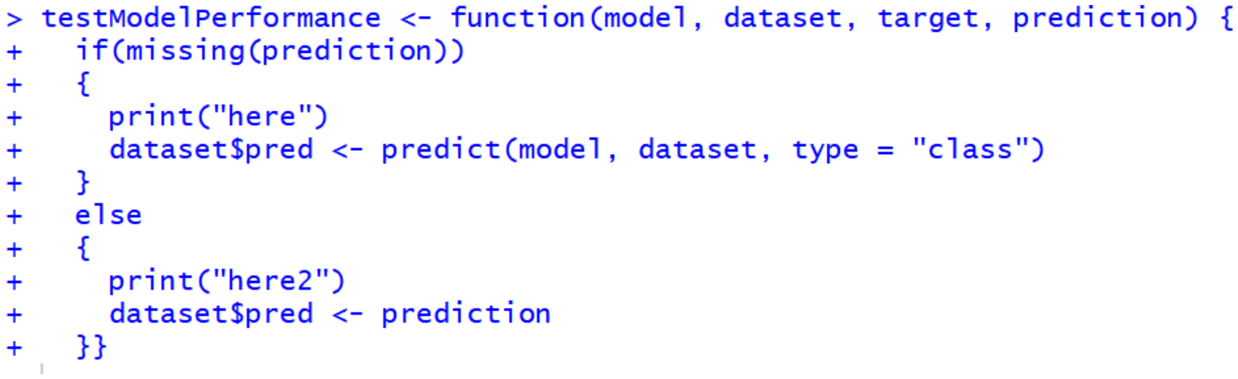
Next plot tree



a.> As seen above, there are 5 leaves

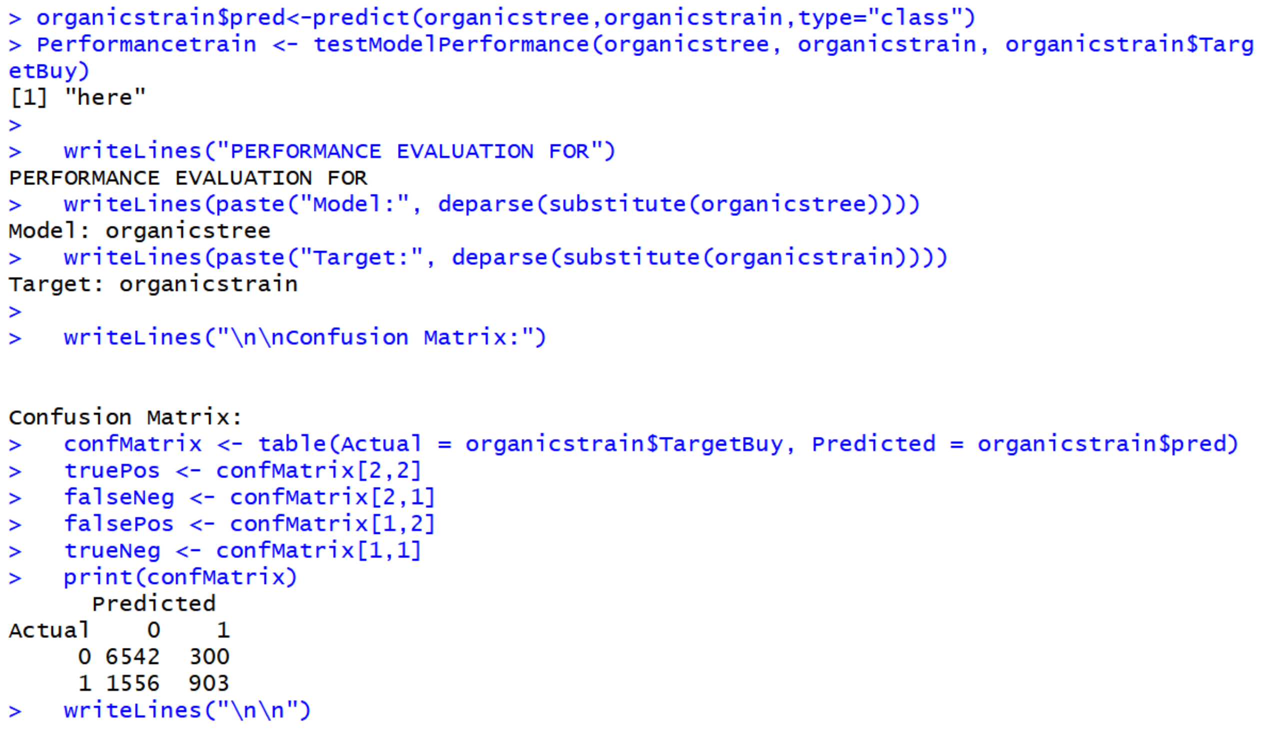
b.> The first split as in the root node is done on DemAge

c.> To create a 2x2 confusion matrix first create the testModelPerformance function.



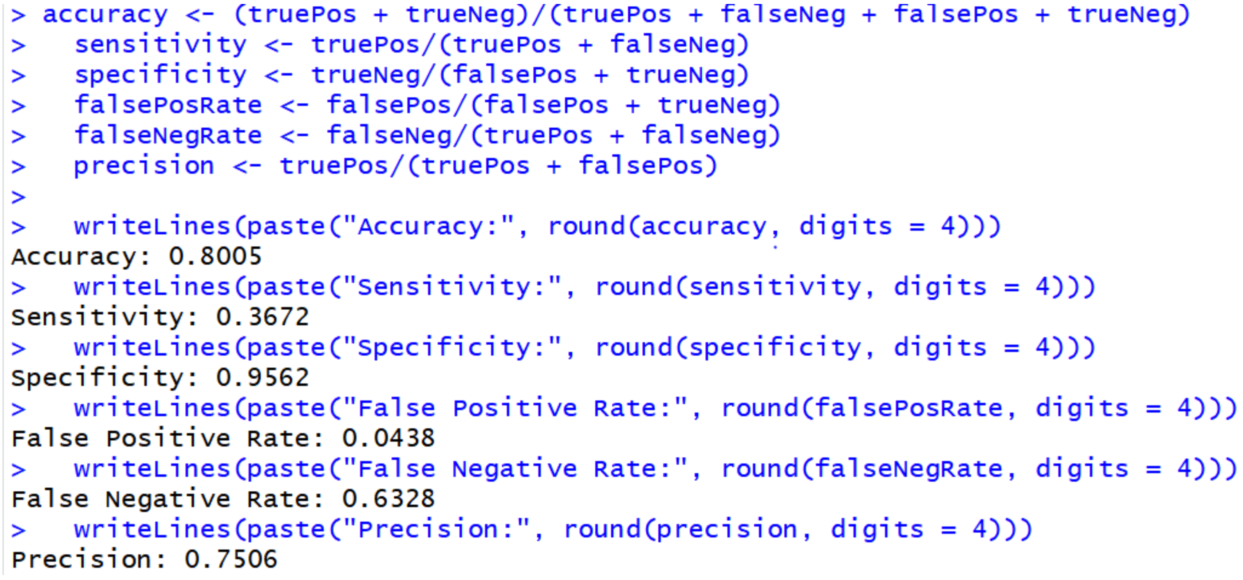
```
> testModelPerformance <- function(model, dataset, target, prediction) {
+   if(missing(prediction))
+   {
+     print("here")
+     dataset$pred <- predict(model, dataset, type = "class")
+   }
+   else
+   {
+     print("here2")
+     dataset$pred <- prediction
+   }}
```

Now, use this function and build the confusion matrix



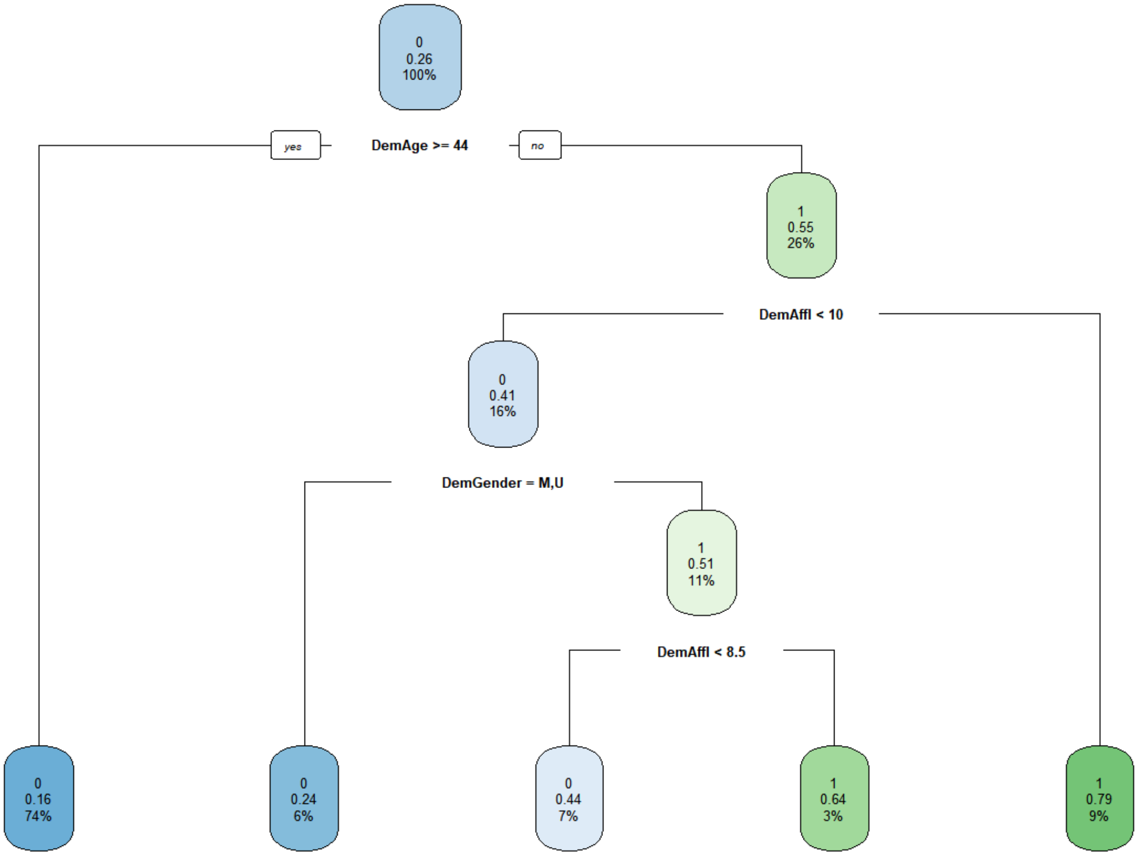
```
> organicstrain$pred<-predict(organicstree,organicstrain,type="class")
> Performancetrain <- testModelPerformance(organicstree, organicstrain, organicstrain$Targ
etBuy)
[1] "here"
>
>   writeLines("PERFORMANCE EVALUATION FOR")
PERFORMANCE EVALUATION FOR
>   writeLines(paste("Model:", deparse(substitute(organicstree))))
Model: organicstree
>   writeLines(paste("Target:", deparse(substitute(organicstrain))))
Target: organicstrain
>
>   writeLines("\n\nConfusion Matrix:")


Confusion Matrix:
>   confMatrix <- table(Actual = organicstrain$TargetBuy, Predicted = organicstrain$pred)
>   truePos <- confMatrix[2,2]
>   falseNeg <- confMatrix[2,1]
>   falsePos <- confMatrix[1,2]
>   trueNeg <- confMatrix[1,1]
>   print(confMatrix)
      Predicted
Actual    0    1
     0 6542  300
     1 1556  903
>   writeLines("\n\n")
```

Now find the metrics like accuracy, sensitivity, specificity, precision etc.



```
> accuracy <- (truePos + trueNeg)/(truePos + falseNeg + falsePos + trueNeg)
>   sensitivity <- truePos/(truePos + falseNeg)
>   specificity <- trueNeg/(falsePos + trueNeg)
>   falsePosRate <- falsePos/(falsePos + trueNeg)
>   falseNegRate <- falseNeg/(truePos + falseNeg)
>   precision <- truePos/(truePos + falsePos)
>
>   writeLines(paste("Accuracy:", round(accuracy, digits = 4)))
Accuracy: 0.8005
>   writeLines(paste("Sensitivity:", round(sensitivity, digits = 4)))
Sensitivity: 0.3672
>   writeLines(paste("Specificity:", round(specificity, digits = 4)))
Specificity: 0.9562
>   writeLines(paste("False Positive Rate:", round(falsePosRate, digits = 4)))
False Positive Rate: 0.0438
>   writeLines(paste("False Negative Rate:", round(falseNegRate, digits = 4)))
False Negative Rate: 0.6328
>   writeLines(paste("Precision:", round(precision, digits = 4)))
Precision: 0.7506
```

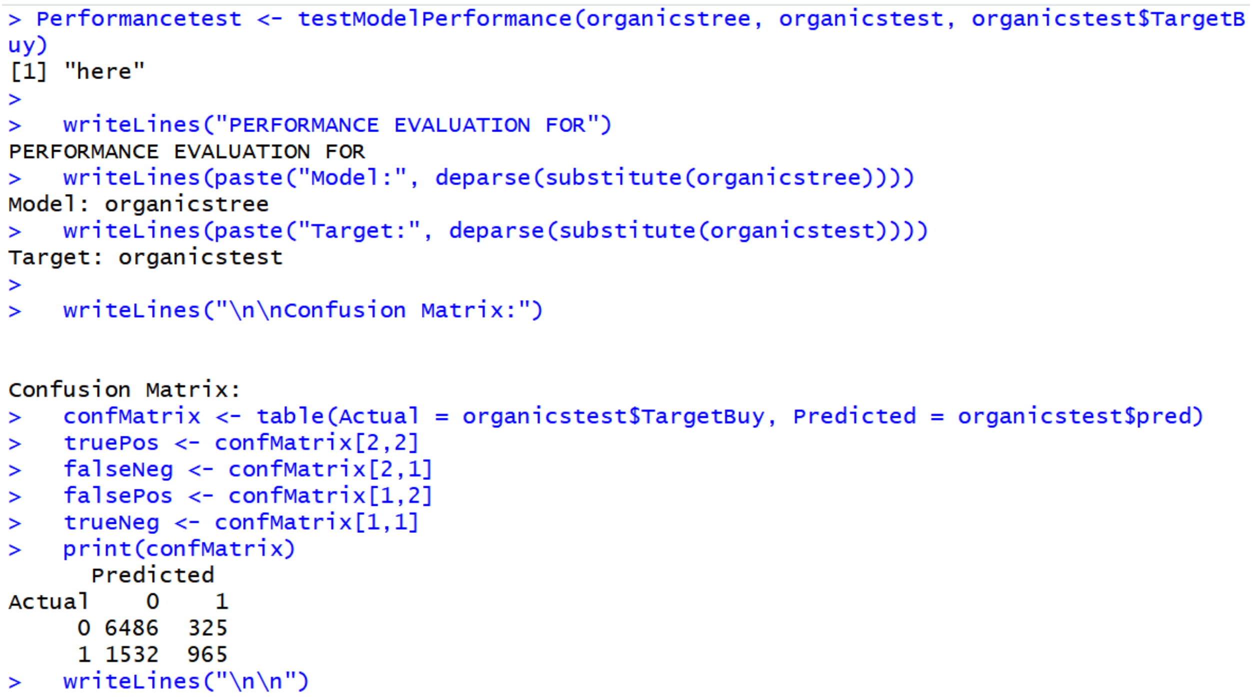
d.> The final decision tree screenshot is as below



5.> Applying classification model from the training data set to the test data.

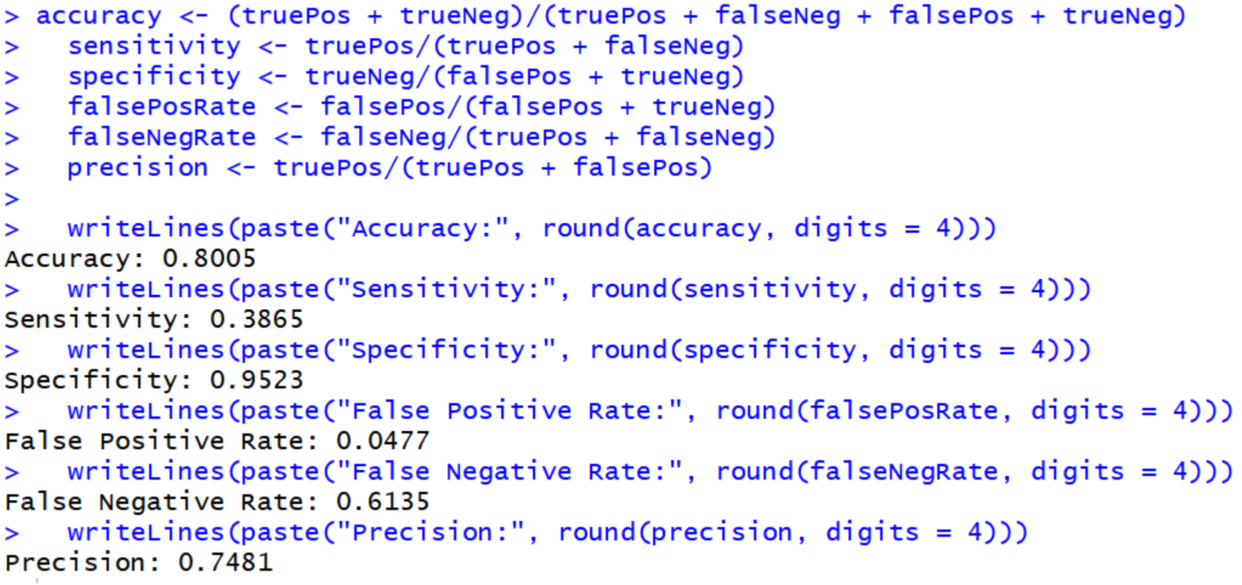
```
> organicstest$pred<-predict(organicstree,organicstest,type="class")
```

Then build the confusion matrix.



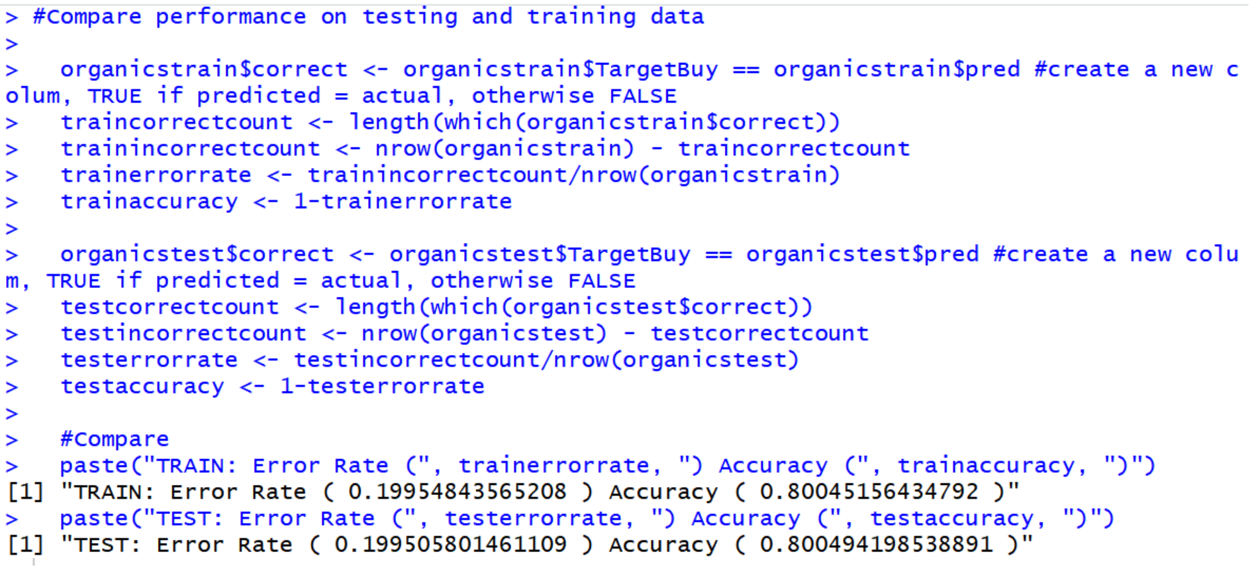
```
> Performancetest <- testModelPerformance(organicstree, organicstest, organicstest$TargetB
uy)
[1] "here"
>
>   writeLines("PERFORMANCE EVALUATION FOR")
PERFORMANCE EVALUATION FOR
>   writeLines(paste("Model:", deparse(substitute(organicstree))))
Model: organicstree
>   writeLines(paste("Target:", deparse(substitute(organicstest))))
Target: organicstest
>
>   writeLines("\n\nConfusion Matrix:")


Confusion Matrix:
>   confMatrix <- table(Actual = organicstest$TargetBuy, Predicted = organicstest$pred)
>   truePos <- confMatrix[2,2]
>   falseNeg <- confMatrix[2,1]
>   falsePos <- confMatrix[1,2]
>   trueNeg <- confMatrix[1,1]
>   print(confMatrix)
      Predicted
Actual    0    1
     0 6486  325
     1 1532  965
>   writeLines("\n\n")
```

Now find the metrics like accuracy, sensitivity, specificity, precision etc.



```
> accuracy <- (truePos + trueNeg)/(truePos + falseNeg + falsePos + trueNeg)
>   sensitivity <- truePos/(truePos + falseNeg)
>   specificity <- trueNeg/(falsePos + trueNeg)
>   falsePosRate <- falsePos/(falsePos + trueNeg)
>   falseNegRate <- falseNeg/(truePos + falseNeg)
>   precision <- truePos/(truePos + falsePos)
>
>   writeLines(paste("Accuracy:", round(accuracy, digits = 4)))
Accuracy: 0.8005
>   writeLines(paste("Sensitivity:", round(sensitivity, digits = 4)))
Sensitivity: 0.3865
>   writeLines(paste("Specificity:", round(specificity, digits = 4)))
Specificity: 0.9523
>   writeLines(paste("False Positive Rate:", round(falsePosRate, digits = 4)))
False Positive Rate: 0.0477
>   writeLines(paste("False Negative Rate:", round(falseNegRate, digits = 4)))
False Negative Rate: 0.6135
>   writeLines(paste("Precision:", round(precision, digits = 4)))
Precision: 0.7481
```

6.> Compare the accuracy of classification of your test and training data sets using the decision tree classification approach.

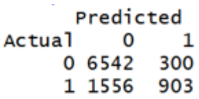


```
> #Compare performance on testing and training data
>
>   organicstrain$correct <- organicstrain$TargetBuy == organicstrain$pred #create a new c
olum, TRUE if predicted = actual, otherwise FALSE
>   traincorrectcount <- length(which(organicstrain$correct))
>   trainincorrectcount <- nrow(organicstrain) - traincorrectcount
>   trainerrorrate <- trainincorrectcount/nrow(organicstrain)
>   trainaccuracy <- 1-trainerrorrate
>
>   organicstest$correct <- organicstest$TargetBuy == organicstest$pred #create a new colu
m, TRUE if predicted = actual, otherwise FALSE
>   testcorrectcount <- length(which(organicstest$correct))
>   testincorrectcount <- nrow(organicstest) - testcorrectcount
>   testerrorrate <- testincorrectcount/nrow(organicstest)
>   testaccuracy <- 1-testerrorrate
>
>   #Compare
>   paste("TRAIN: Error Rate (", trainerrorrate, ") Accuracy (", trainaccuracy, ")")
[1] "TRAIN: Error Rate ( 0.19954843565208 ) Accuracy ( 0.80045156434792 )"
>   paste("TEST: Error Rate (", testerrorrate, ") Accuracy (", testaccuracy, ")")
[1] "TEST: Error Rate ( 0.199505801461109 ) Accuracy ( 0.800494198538891 )"
```

The error rate and the accuracy are approximately same for training and test data.

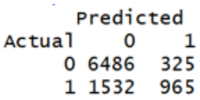
The confusion matrix for both test and train data predictions.

Training data



```
       Predicted
Actual    0    1
     0 6542  300
     1 1556  903
```

Test data

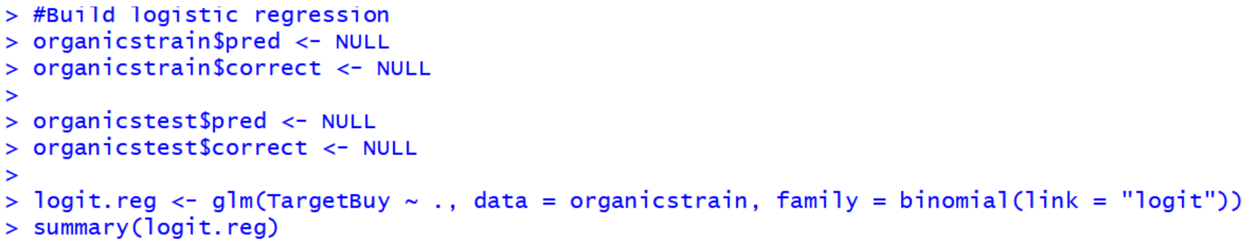


```
       Predicted
Actual    0    1
     0 6486  325
     1 1532  965
```

Below is the metrics comparison table:

Metrics	Training data	Test data
Accuracy	*0.8005*	*0.8005*
Error Rate	*0.1995*	*0.1995*
Sensitivity	*0.3672*	*0.3865*
Specificity	*0.9562*	*0.9523*
False Positive Rate	*0.0438*	*0.0477*
False Negative Rate	*0.6328*	*0.6135*
Precision	*0.7506*	*0.7481*

7.> Build a logistic regression model for classification of the dataset.



```
> #Build logistic regression
> organicstrain$pred <- NULL
> organicstrain$correct <- NULL
>
> organicstest$pred <- NULL
> organicstest$correct <- NULL
>
> logit.reg <- glm(TargetBuy ~ ., data = organicstrain, family = binomial(link = "logit"))
> summary(logit.reg)
```

Use only the significant variables to build a new model

```
> logit.reg <- glm(TargetBuy ~ DemAffl+DemAge+DemGender, data = organicstrain, family = bi
nomial(link = "logit"))
> summary(logit.reg)

Call:
glm(formula = TargetBuy ~ DemAffl + DemAge + DemGender, family = binomial(link = "logit"),

    data = organicstrain)

Deviance Residuals:
   Min       1Q   Median       3Q      Max
-2.142   -0.699   -0.423    0.505    2.891

Coefficients:
            Estimate Std. Error z value             Pr(>|z|)
(Intercept) -0.47210    0.14190   -3.33              0.00088 ***
DemAffl      0.27451    0.00917   29.94 < 0.0000000000000002 ***
DemAge      -0.05253    0.00223  -23.52 < 0.0000000000000002 ***
DemGenderM  -1.00693    0.06507  -15.48 < 0.0000000000000002 ***
DemGenderU  -1.93630    0.15355  -12.61 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 10744.3  on 9300  degrees of freedom
Residual deviance:  8266.1  on 9296  degrees of freedom
AIC: 8276

Number of Fisher Scoring iterations: 5
```

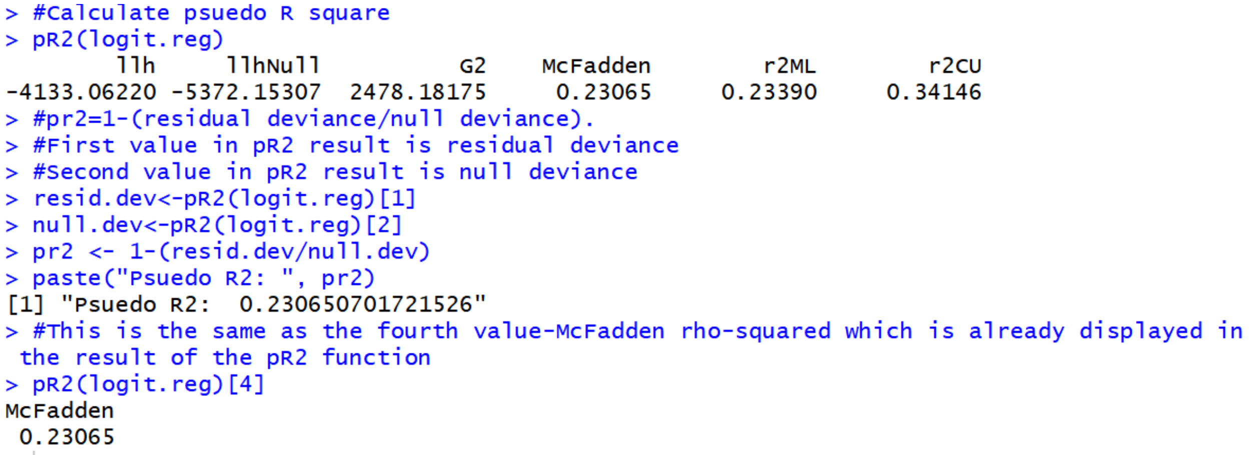
Build confidence intervals and calculate the odds ratio for this model: the odds ratio represents how the odds of the event occurring change with a 1 unit increase in that variable, all other things being equal. Here, the event is TargetBuy.

```
> confint.default(logit.reg) #Build confidence intervals
                2.5 %    97.5 %
(Intercept) -0.750218 -0.193991
DemAffl      0.256544  0.292482
DemAge      -0.056909 -0.048154
DemGenderM  -1.134451 -0.879401
DemGenderU  -2.237248 -1.635348
> exp(coef(logit.reg)) #Calculate odds ratio
(Intercept)     DemAffl      DemAge  DemGenderM  DemGenderU
    0.62369     1.31589     0.94882     0.36534     0.14424
```

Calculate Chi-Square value:

```
> #Calculate Chi-Square
> devdiff <- with(logit.reg, null.deviance - deviance) #difference in deviance between nul
l and this model
> dofdiff <- with(logit.reg, df.null - df.residual) #difference in degrees of freedom betw
een null and this model
> pval <- pchisq(devdiff, dofdiff, lower.tail = FALSE )
> paste("Chi-Square: ", devdiff, " df: ", dofdiff, " p-value: ", pval)
[1] "Chi-Square:  2478.18175284126  df:  4  p-value:  0"
```

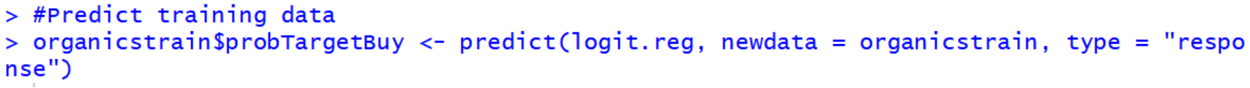
Calculate pseudo R square for the model:



```
> #Calculate psuedo R square
> pR2(logit.reg)
         llh      llhNull           G2     McFadden         r2ML         r2CU 
-4133.06220 -5372.15307  2478.18175     0.23065     0.23390     0.34146 
> #pr2=1-(residual deviance/null deviance).
> #First value in pR2 result is residual deviance
> #Second value in pR2 result is null deviance
> resid.dev<-pR2(logit.reg)[1]
> null.dev<-pR2(logit.reg)[2]
> pr2 <- 1-(resid.dev/null.dev)
> paste("Psuedo R2: ", pr2)
[1] "Psuedo R2:  0.230650701721526"
> #This is the same as the fourth value-McFadden rho-squared which is already displayed in
 the result of the pR2 function
> pR2(logit.reg)[4]
McFadden 
 0.23065 
```

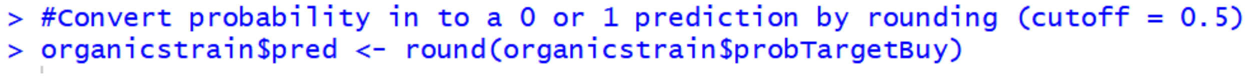
Pseudo R squared is 0.23065. A value of pseudo R squared between 0.2 and 0.4 is considered good.

Apply prediction on the training data



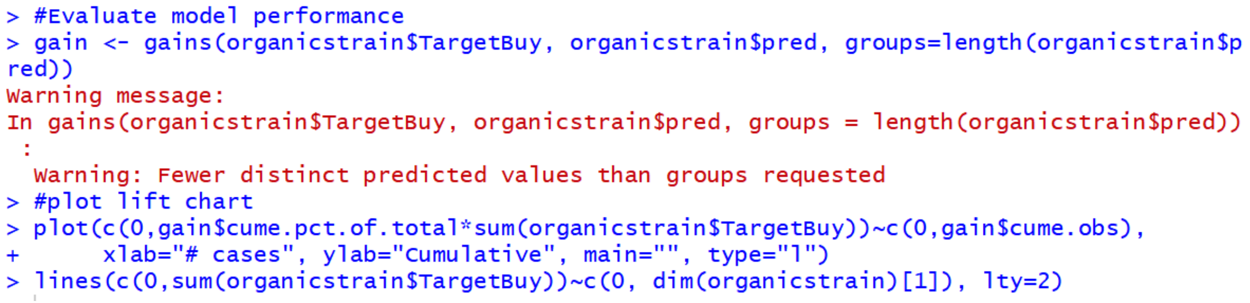
```
> #Predict training data
> organicstrain$probTargetBuy <- predict(logit.reg, newdata = organicstrain, type = "respo
nse")
```

Convert probability in to a 0 or 1 prediction by rounding (cutoff = 0.5)

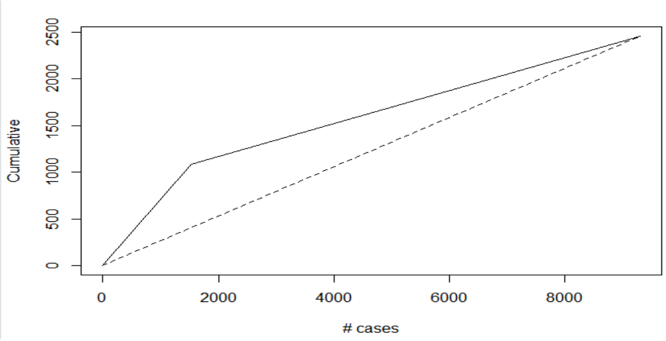


```
> #Convert probability in to a 0 or 1 prediction by rounding (cutoff = 0.5)
> organicstrain$pred <- round(organicstrain$probTargetBuy)
```

Evaluate performance of the model. Plot the lift chart on training data.



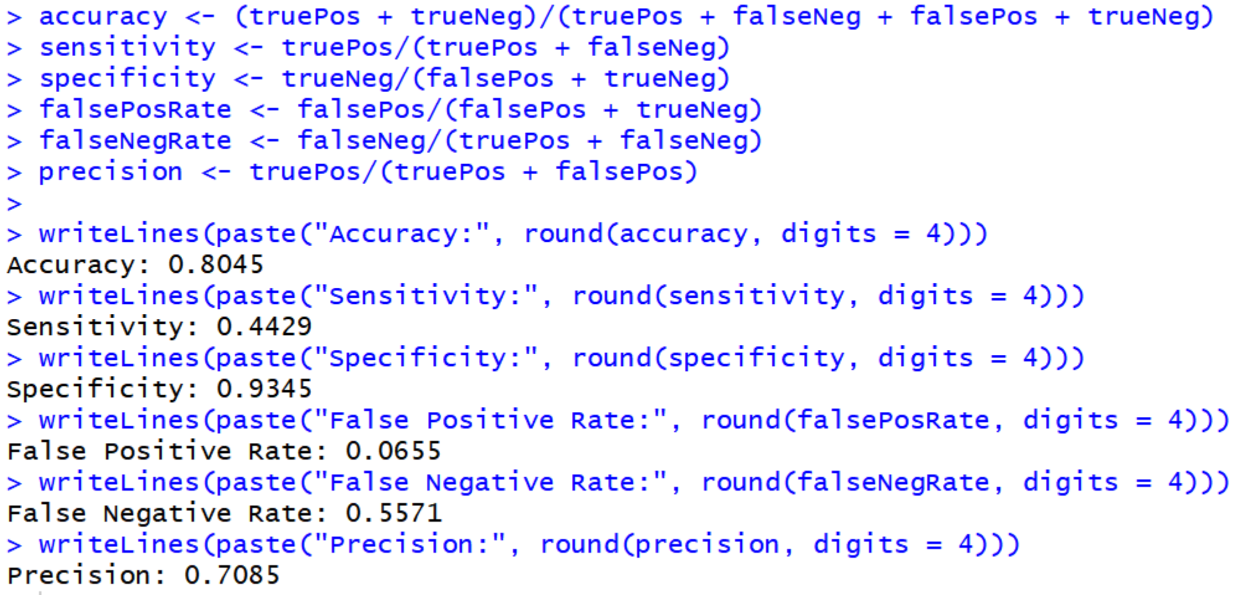
```
> #Evaluate model performance
> gain <- gains(organicstrain$TargetBuy, organicstrain$pred, groups=length(organicstrain$p
red))
Warning message:
In gains(organicstrain$TargetBuy, organicstrain$pred, groups = length(organicstrain$pred))
 :
  Warning: Fewer distinct predicted values than groups requested
> #plot lift chart
> plot(c(0,gain$cume.pct.of.total*sum(organicstrain$TargetBuy))~c(0,gain$cume.obs), 
+      xlab="# cases", ylab="Cumulative", main="", type="l")
> lines(c(0,sum(organicstrain$TargetBuy))~c(0, dim(organicstrain)[1]), lty=2)
```



Next build the confusion matrix with prediction on training data

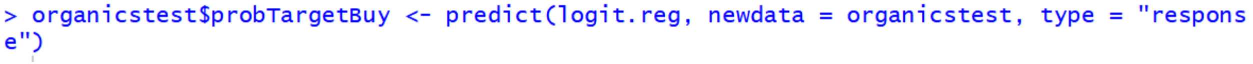
```
> #Confusion Matrix
> writeLines("PERFORMANCE EVALUATION FOR")
PERFORMANCE EVALUATION FOR
> writeLines(paste("Model:", deparse(substitute(logit.reg))))
Model: logit.reg
> writeLines(paste("Target:", deparse(substitute(organicstrain))))
Target: organicstrain
>
> writeLines("\n\nConfusion Matrix:")


Confusion Matrix:
> confMatrix <- table(Actual = organicstrain$TargetBuy, Predicted = organicstrain$pred)
> truePos <- confMatrix[2,2]
> falseNeg <- confMatrix[2,1]
> falsePos <- confMatrix[1,2]
> trueNeg <- confMatrix[1,1]
> print(confMatrix)
      Predicted
Actual    0    1
     0 6394  448
     1 1370 1089
> writeLines("\n\n")
```

Calculate the metrics using the confusion matrix on the predicted training data



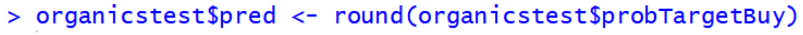
```
> accuracy <- (truePos + trueNeg)/(truePos + falseNeg + falsePos + trueNeg)
> sensitivity <- truePos/(truePos + falseNeg)
> specificity <- trueNeg/(falsePos + trueNeg)
> falsePosRate <- falsePos/(falsePos + trueNeg)
> falseNegRate <- falseNeg/(truePos + falseNeg)
> precision <- truePos/(truePos + falsePos)
>
> writeLines(paste("Accuracy:", round(accuracy, digits = 4)))
Accuracy: 0.8045
> writeLines(paste("Sensitivity:", round(sensitivity, digits = 4)))
Sensitivity: 0.4429
> writeLines(paste("Specificity:", round(specificity, digits = 4)))
Specificity: 0.9345
> writeLines(paste("False Positive Rate:", round(falsePosRate, digits = 4)))
False Positive Rate: 0.0655
> writeLines(paste("False Negative Rate:", round(falseNegRate, digits = 4)))
False Negative Rate: 0.5571
> writeLines(paste("Precision:", round(precision, digits = 4)))
Precision: 0.7085
```

Apply prediction on the test data



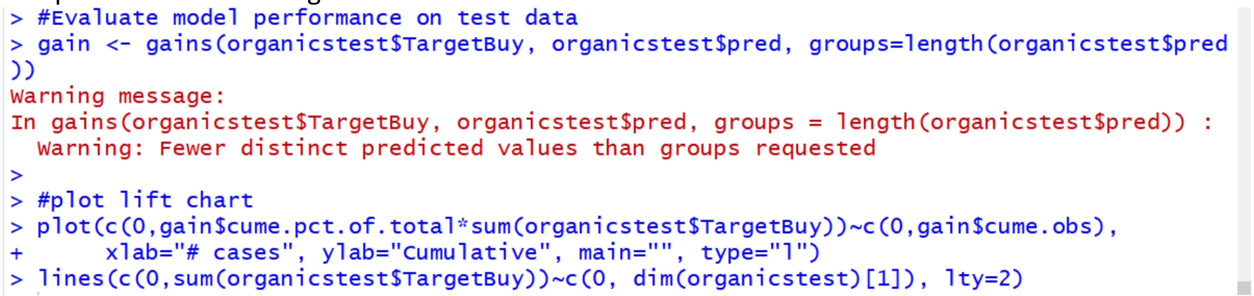
```
> organicstest$probTargetBuy <- predict(logit.reg, newdata = organicstest, type = "respons
e")
```

Convert probability in to a 0 or 1 prediction by rounding (cutoff = 0.5)

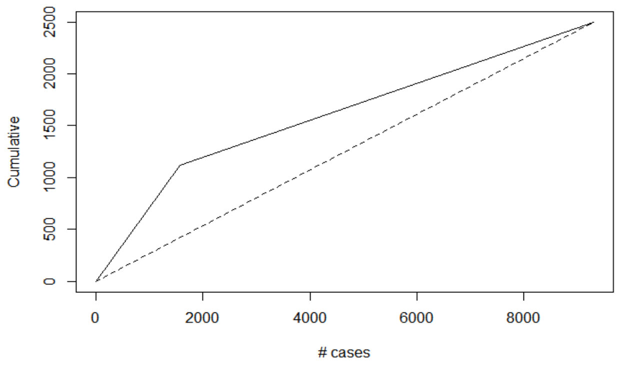


```
> organicstest$pred <- round(organicstest$probTargetBuy)
```

Evaluate performance of the model. Plot the lift chart on test data. For a given number of records (x-axis), the lift curve value on the y-axis tells us how much better we are doing compared to random assignment.



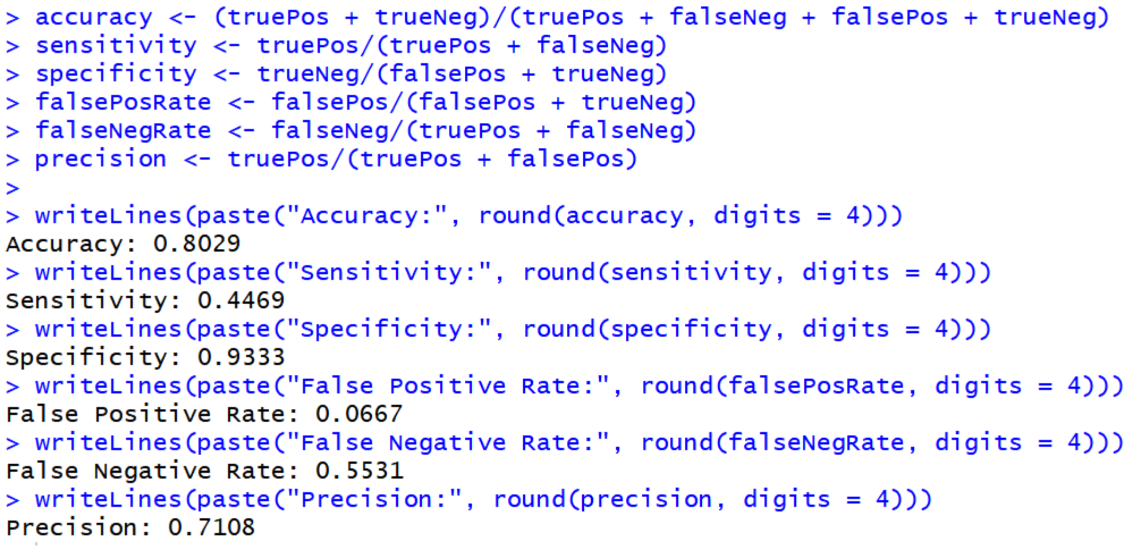
```
> #Evaluate model performance on test data
> gain <- gains(organicstest$TargetBuy, organicstest$pred, groups=length(organicstest$pred
))
Warning message:
In gains(organicstest$TargetBuy, organicstest$pred, groups = length(organicstest$pred)) :
  Warning: Fewer distinct predicted values than groups requested
>
> #plot lift chart
> plot(c(0,gain$cume.pct.of.total*sum(organicstest$TargetBuy))~c(0,gain$cume.obs),
+      xlab="# cases", ylab="Cumulative", main="", type="l")
> lines(c(0,sum(organicstest$TargetBuy))~c(0, dim(organicstest)[1]), lty=2)
```



Next build the confusion matrix with prediction on test data

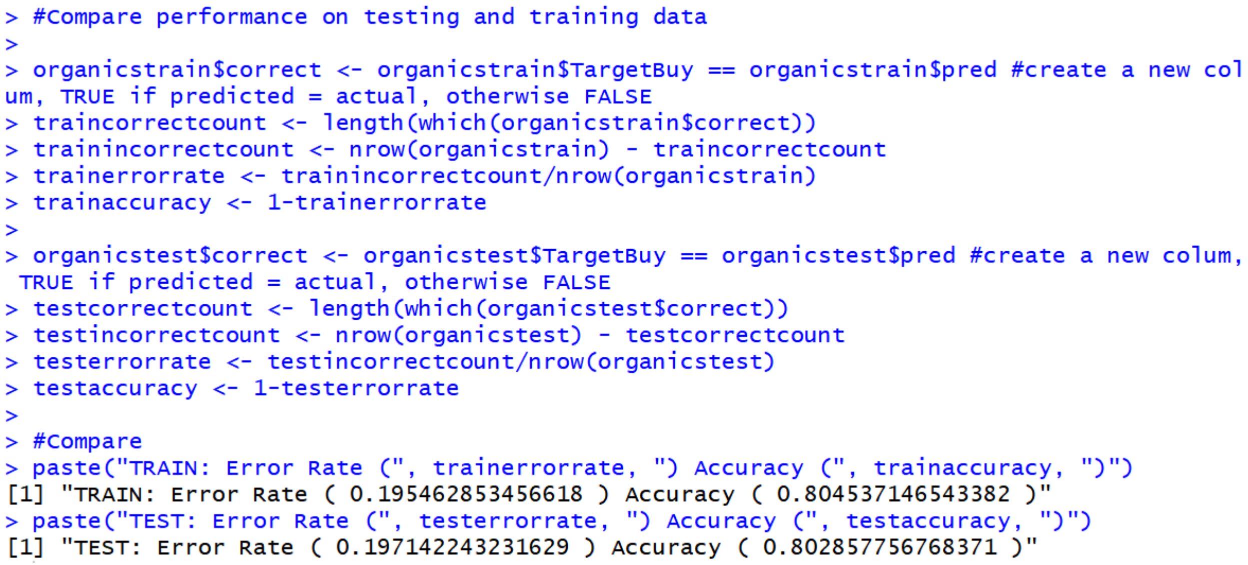
```
> #Confusion Matrix
>
> writeLines("PERFORMANCE EVALUATION FOR")
PERFORMANCE EVALUATION FOR
> writeLines(paste("Model:", deparse(substitute(logit.reg))))
Model: logit.reg
> writeLines(paste("Target:", deparse(substitute(organicstest))))
Target: organicstest
>
> writeLines("\n\nConfusion Matrix:")


Confusion Matrix:
> confMatrix <- table(Actual = organicstest$TargetBuy, Predicted = organicstest$pred)
> truePos <- confMatrix[2,2]
> falseNeg <- confMatrix[2,1]
> falsePos <- confMatrix[1,2]
> trueNeg <- confMatrix[1,1]
> print(confMatrix)
      Predicted
Actual    0    1
     0 6357  454
     1 1381 1116
> writeLines("\n\n")
```

Calculate the metrics using the confusion matrix on the predicted test data



```
> accuracy <- (truePos + trueNeg)/(truePos + falseNeg + falsePos + trueNeg)
> sensitivity <- truePos/(truePos + falseNeg)
> specificity <- trueNeg/(falsePos + trueNeg)
> falsePosRate <- falsePos/(falsePos + trueNeg)
> falseNegRate <- falseNeg/(truePos + falseNeg)
> precision <- truePos/(truePos + falsePos)
>
> writeLines(paste("Accuracy:", round(accuracy, digits = 4)))
Accuracy: 0.8029
> writeLines(paste("Sensitivity:", round(sensitivity, digits = 4)))
Sensitivity: 0.4469
> writeLines(paste("Specificity:", round(specificity, digits = 4)))
Specificity: 0.9333
> writeLines(paste("False Positive Rate:", round(falsePosRate, digits = 4)))
False Positive Rate: 0.0667
> writeLines(paste("False Negative Rate:", round(falseNegRate, digits = 4)))
False Negative Rate: 0.5531
> writeLines(paste("Precision:", round(precision, digits = 4)))
Precision: 0.7108
```

8.> Compare performance of the logit prediction models on your test and training data sets.



```
> #Compare performance on testing and training data
>
> organicstrain$correct <- organicstrain$TargetBuy == organicstrain$pred #create a new col
um, TRUE if predicted = actual, otherwise FALSE
> traincorrectcount <- length(which(organicstrain$correct))
> trainincorrectcount <- nrow(organicstrain) - traincorrectcount
> trainerrorrate <- trainincorrectcount/nrow(organicstrain)
> trainaccuracy <- 1-trainerrorrate
>
> organicstest$correct <- organicstest$TargetBuy == organicstest$pred #create a new colum,
 TRUE if predicted = actual, otherwise FALSE
> testcorrectcount <- length(which(organicstest$correct))
> testincorrectcount <- nrow(organicstest) - testcorrectcount
> testerrorrate <- testincorrectcount/nrow(organicstest)
> testaccuracy <- 1-testerrorrate
>
> #Compare
> paste("TRAIN: Error Rate (", trainerrorrate, ") Accuracy (", trainaccuracy, ")")
[1] "TRAIN: Error Rate ( 0.195462853456618 ) Accuracy ( 0.804537146543382 )"
> paste("TEST: Error Rate (", testerrorrate, ") Accuracy (", testaccuracy, ")")
[1] "TEST: Error Rate ( 0.197142243231629 ) Accuracy ( 0.802857756768371 )"
```

Below is the metrics comparison table for prediction on training and test data using logit:

Metrics	**Training data**	**Test data**
Accuracy	*0.8045*	*0.8029*
Error Rate	*0.1955*	*0.1971*
Sensitivity	*0.4429*	*0.4469*
Specificity	*0.9345*	*0.9333*
False Positive Rate	*0.0655*	*0.0667*
False Negative Rate	*0.5571*	*0.5531*
Precision	*0.7085*	*0.7108*

9.> Comparison of metrics from Decision tree prediction and logit prediction on both training and test dataset.

Metrics	**Decision tree**		**Logistic regression**	
Metrics	**Training data**	**Test data**	**Training data**	**Test data**
Accuracy	*0.8005*	*0.8005*	*0.8045*	*0.8029*
Error Rate	*0.1995*	*0.1995*	*0.1955*	*0.1971*
Sensitivity	*0.3672*	*0.3865*	*0.4429*	*0.4469*
Specificity	*0.9562*	*0.9523*	*0.9345*	*0.9333*
False Positive Rate	*0.0438*	*0.0477*	*0.0655*	*0.0667*
False Negative Rate	*0.6328*	*0.6135*	*0.5571*	*0.5531*
Precision	*0.7506*	*0.7481*	*0.7085*	*0.7108*

From the above comparison of the two classification techniques deployed, we can see that the accuracy of the logistic regression and decision tree is approximately the same. However, the Specificity of logistic regression is higher than that of the decision tree.

The observed differences are the false positive rate of the logistic regression is more than that of the decision tree. However, the false negative rate of logit is less than that of the decision tree. The sensitivity of the logit is higher but the specificity is lower than that of the decision tree. However, both are cases of low sensitivity and very high specificity. This means that they may overlook a positive, but they will rarely classify an actual negative as a positive.