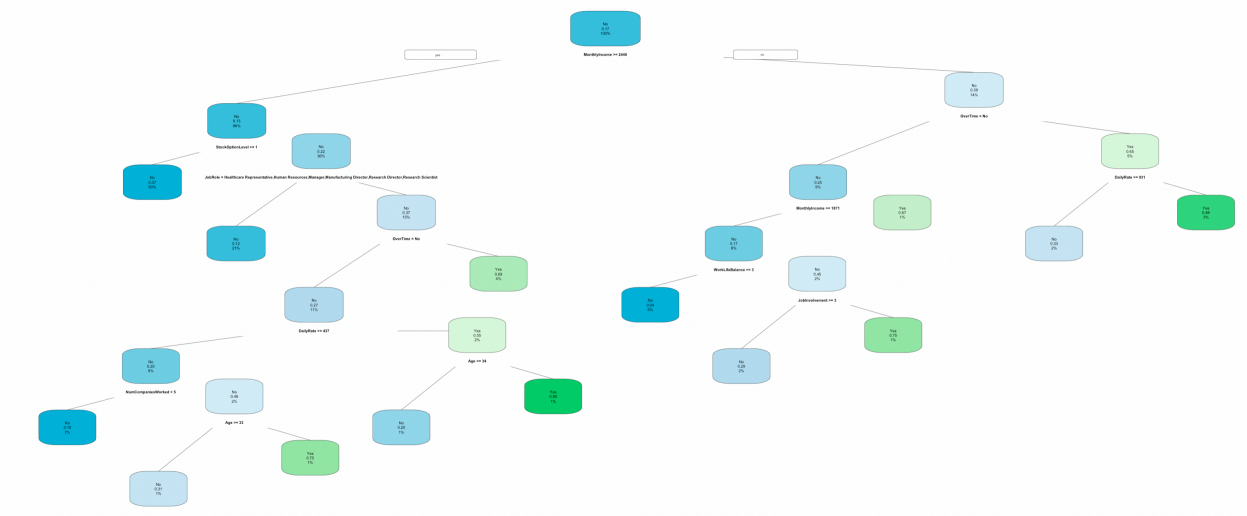
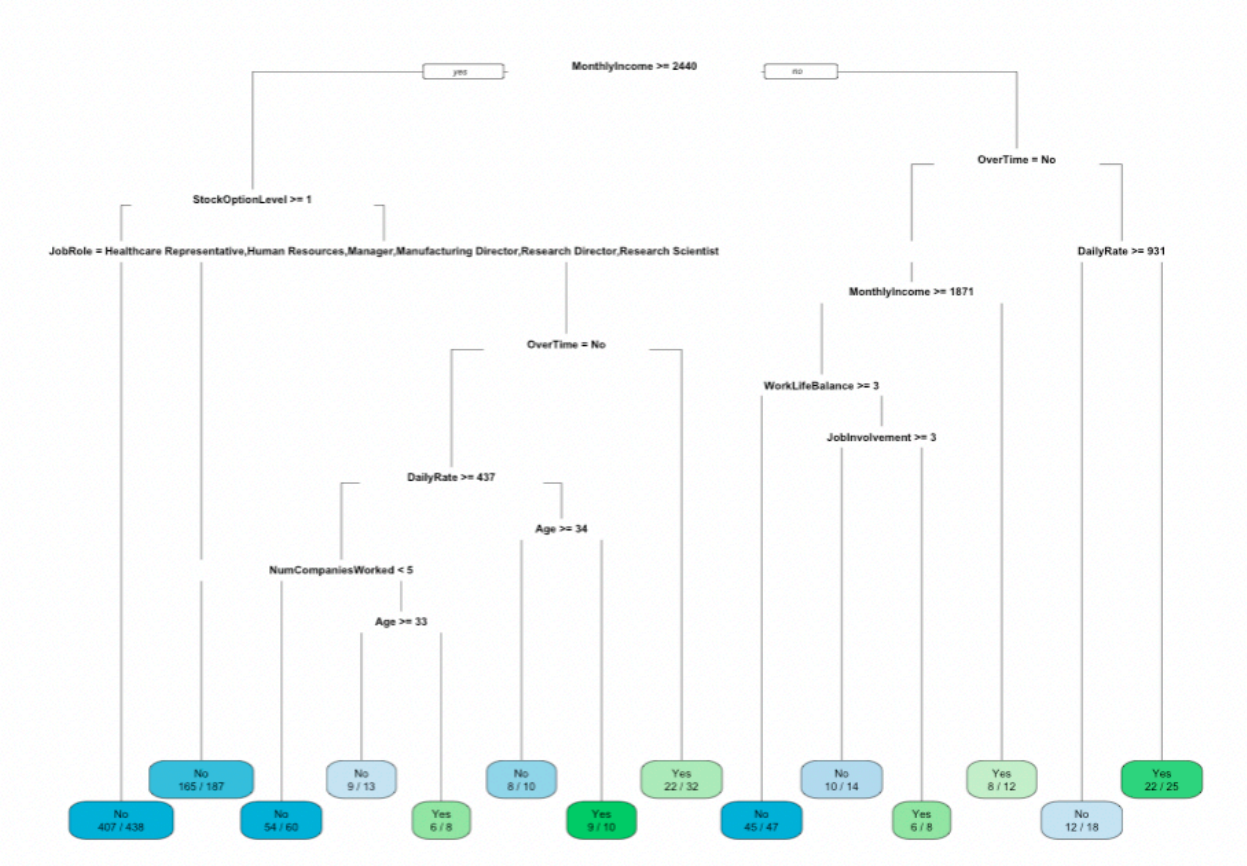
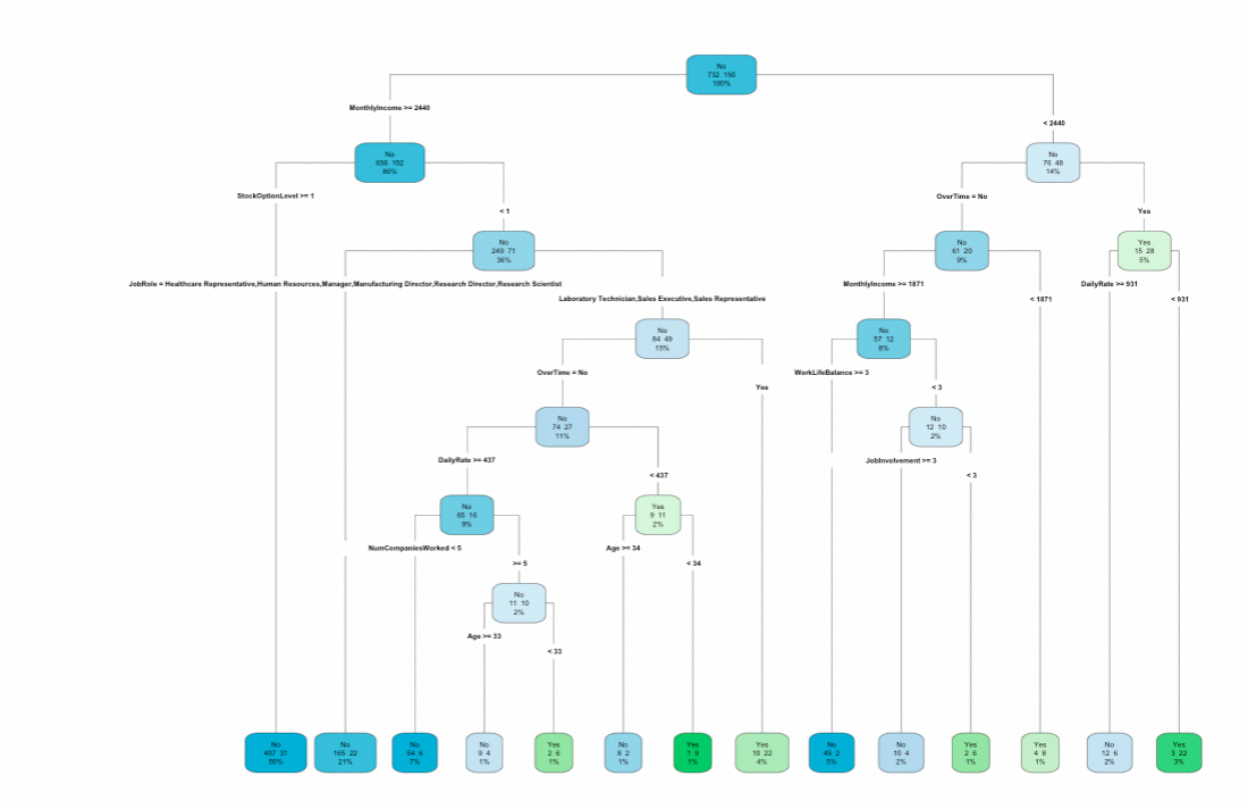
```r
# AD 699
# Assignment 4
# Tiange Chang
# Food: onions

install.packages('rpart')
library(rpart)
install.packages('rpart.plot')
library(rpart.plot)
install.packages('caret')
library(caret)

# Task 1: Classification Tree

# 1.
library(tidyverse)
setwd('/Users/t/Desktop')
pa <- read_csv('people_analytics.csv')
View(pa)
# 2.
# seed: 20
set.seed(20)
train <- sample_frac(pa, 0.6)
valid <- setdiff(pa, train)

# 3.
tree <- rpart(Attrition~., method = 'class', data=train)
tree
# 4.
rpart.plot(tree, tweak = 1.8, fallen.leaves = FALSE)

# 4. a
rpart.plot(tree, type = 0, extra = 2)

# 4. b
rpart.plot(tree, type = 4, extra = 101)

# 4. c
# For the first one, the text is too small, then I set fallen.leaves to FALSE,
# and then use the tweak argument to make the text larger.

# For the second tree, I used the default type which is type = 0. Extra = 2
# showed the class rate.

# For the third one, I used type = 4 which is the left and right split labels,
# and the extra = 101, this showed nbr of obs and percentages.

# I like the first one best, because it has a better looking. Unlike the other
# two trees, the leaves are at the bottom of the plot. By setting
# fallen.leaves = FALSE, it provided a left and right split. This split gave my
# model a more real tree looking.
```



4.


yes
MonthlyIncome >= 2440
no
StockOptionLevel >= 1
JobRole = Healthcare Representative,Human Resources,Manager,Manufacturing Director,Research Director,Research Scientist
OverTime = No
OverTime = No
MonthlyIncome >= 1871
DailyRate >= 931
DailyRate >= 437
Age >= 34
NumCompaniesWorked < 5
Age >= 33
WorkLifeBalance >= 3
JobInvolvement >= 3
No 407 / 438
No 165 / 187
No 54 / 60
No 9 / 13
Yes 6 / 8
No 8 / 10
Yes 9 / 10
Yes 22 / 32
No 45 / 47
No 10 / 14
Yes 6 / 8
Yes 8 / 12
No 12 / 18
Yes 22 / 25

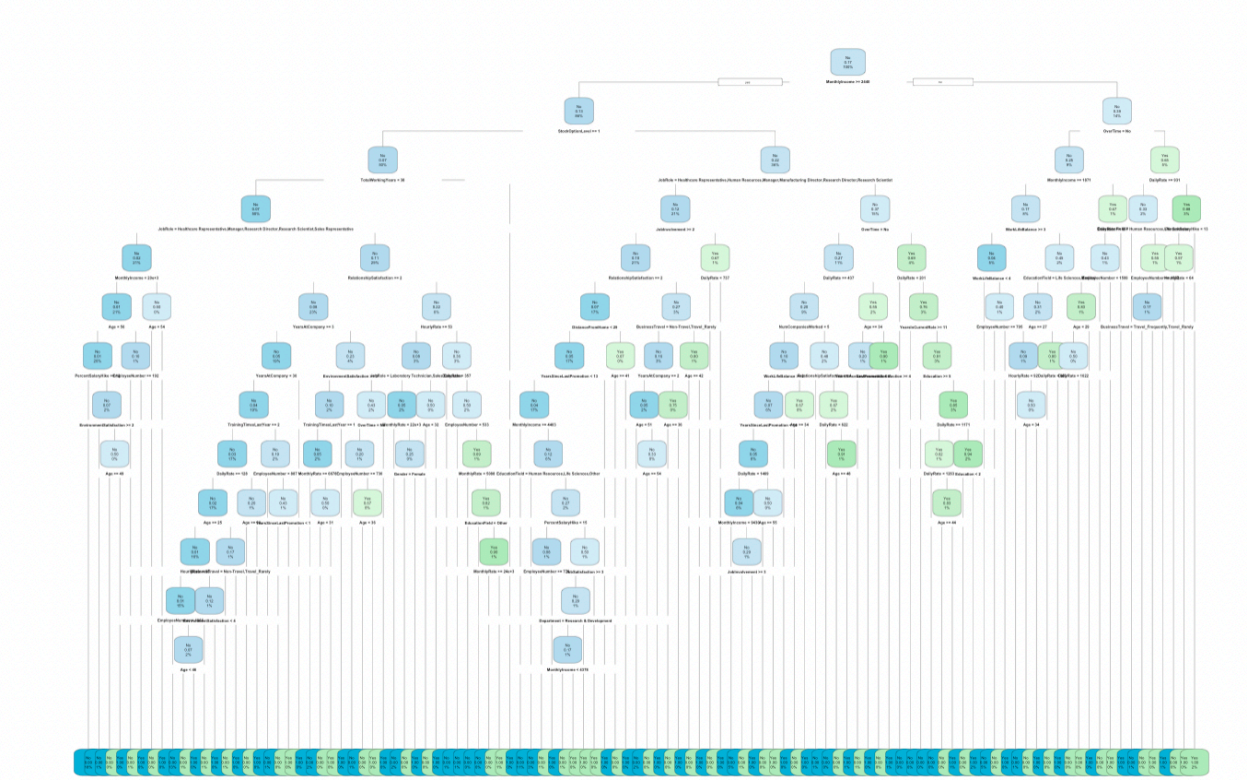
4. a



4. b

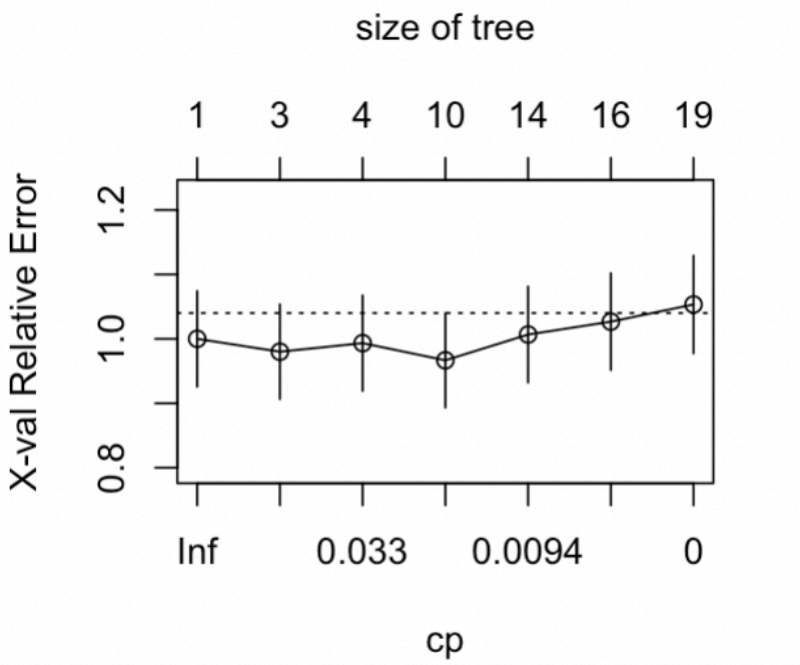
```
# 5.
# The root node is "MonthlyIncome". It splits at the point of 2440. If it was
# greater than or equal to 2440, it goes to "yes" and then go to the next node
# "StockOptionLevel". If it was less than 2440, it goes to "no" and then go to
# the next node "OverTime".

# The root node is significant is because it is the highest node in the tree
# structure, and has no parent. This node is a global element and represents
# the entire message.

# 6.
# Not all the input variables from the dataset appear in the model diagram.
# If all the input variables were appear in the diagram, it will be too large.
# It would cause overfitting.

# 7.
# In order to leave the company, you will need to have a less than 2440 monthly
# income, not eligible for overtime pay, and a less than 931 daily rate.
```

```
# 8.
# Overfit Tree = OFT
OFT <- rpart(Attrition~., method = 'class',
                    cp = 0, minsplit = 2, data = train)
rpart.plot(OFT)
```

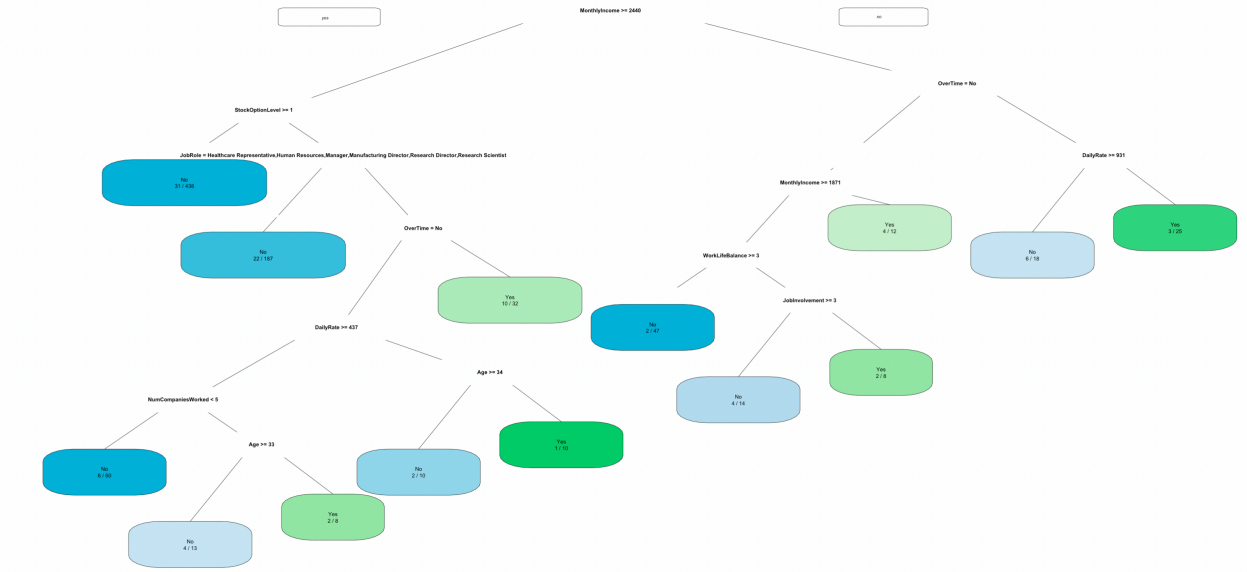


8.

```
# 9.
# Five-fold Cross-validation Tree = FCT
FCT <- rpart(Attrition~., method = 'class', cp = 0, xval = 5, data = train)
printcp(FCT)
plotcp(FCT)
# The cp value I chose is  0.0133333.
Classification tree:
rpart(formula = Attrition ~ ., data = train, method = "class",
    cp = 0, xval = 5)

Variables actually used in tree construction:
 [1] Age                    DailyRate              Education
 [4] EducationField         JobInvolvement         JobRole
 [7] MonthlyIncome          NumCompaniesWorked     OverTime
[10] RelationshipSatisfaction StockOptionLevel      WorkLifeBalance

Root node error: 150/882 = 0.17007

n= 882

         CP nsplit rel error  xerror     xstd
1 0.0433333      0   1.00000 1.00000 0.074383
2 0.0400000      2   0.91333 0.98000 0.073786
3 0.0266667      3   0.87333 0.99333 0.074186
4 0.0133333      9   0.71333 0.96667 0.073382
5 0.0066667     13   0.66000 1.00667 0.074580
6 0.0044444     15   0.64667 1.02667 0.075162
7 0.0000000     18   0.63333 1.05333 0.075923
```




```r
# 10.
# Optimal complexity parameter tree = CPT
CPT <- rpart(Attrition~., method = 'class', cp = 0.0133333, data = train)

# 11.
rpart.plot(CPT, type=0, extra=3, fallen.leaves = FALSE, tweak = 1.8)
```



11.

```r
# 12. a
pred_train <- predict(OFT, train, type = 'class')
confusionMatrix(pred_train, as.factor(train$Attrition))

pred_valid <- predict(OFT, valid, type = 'class')
confusionMatrix(pred_valid, as.factor(valid$Attrition))

# The accuracy for the train set is 1.
# The accuracy for the valid set is 0.784.

# The performance of my huge tree against my training set is better than
# the validation set.
```

```
> confusionMatrix(pred_train, as.factor(train$Attrition))
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  732   0
       Yes   0 150

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.8299
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8299
         Detection Rate : 0.8299
   Detection Prevalence : 0.8299
      Balanced Accuracy : 1.0000

       'Positive' Class : No
```

```
> confusionMatrix(pred_valid, as.factor(valid$Attrition))
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  420  46
       Yes  81  41

               Accuracy : 0.784
                 95% CI : (0.7485, 0.8166)
    No Information Rate : 0.852
    P-Value [Acc > NIR] : 0.999996

                  Kappa : 0.2655

 Mcnemar's Test P-Value : 0.002553

            Sensitivity : 0.8383
            Specificity : 0.4713
         Pos Pred Value : 0.9013
         Neg Pred Value : 0.3361
             Prevalence : 0.8520
         Detection Rate : 0.7143
   Detection Prevalence : 0.7925
      Balanced Accuracy : 0.6548

       'Positive' Class : No
```

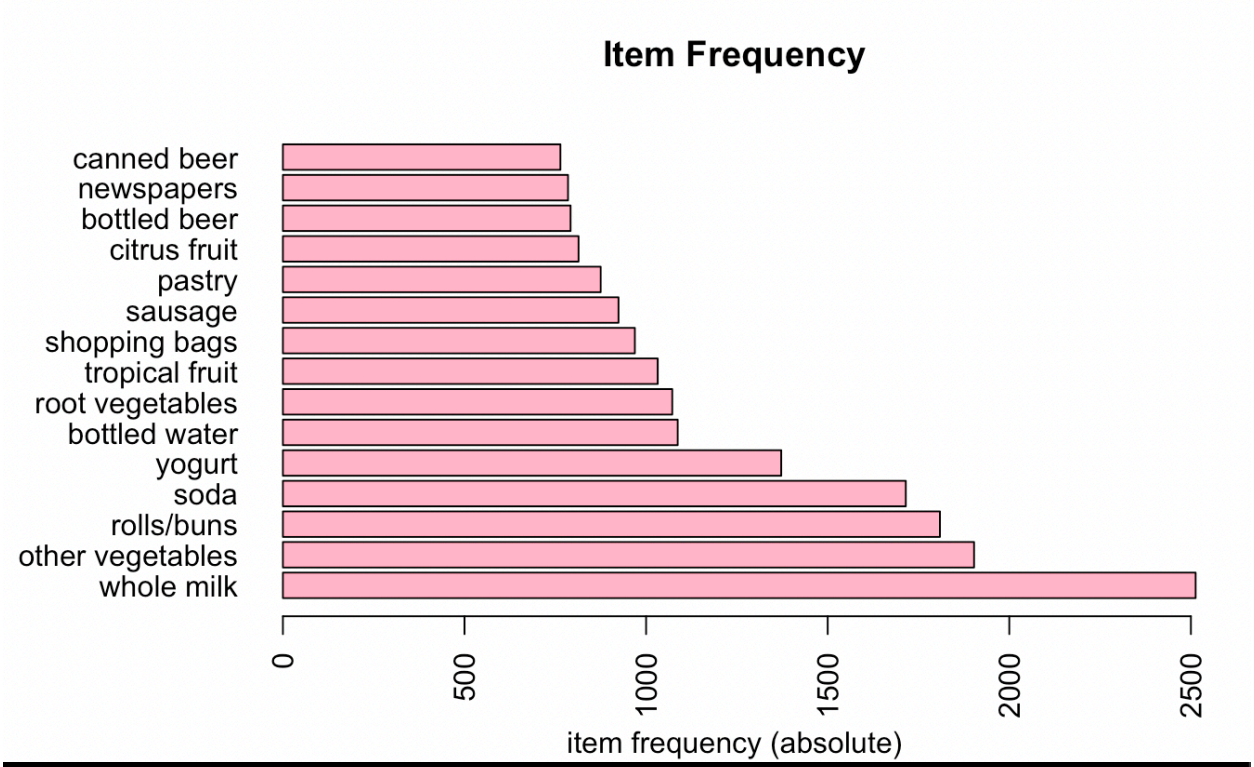
```r
# 12. b
pred_train2 <- predict(CPT, train, type = 'class')
confusionMatrix(pred_train2, as.factor(train$Attrition))

pred_valid2 <- predict(CPT, valid, type = 'class')
confusionMatrix(pred_valid2, as.factor(valid$Attrition))

# The accuracy for the train set is 0.8878.
# The accuracy for the valid set is 0.8486.

# The performance of my optimal cp tree against my training set is slightly
# better than the validation set.

# The difference between the two accuracy values are very low in my optimal cp
# tree compared to the huge tree.
```

```
> confusionMatrix(pred_train2, as.factor(train$Attrition))
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  710  77
       Yes  22  73

               Accuracy : 0.8878
                 95% CI : (0.865, 0.9078)
    No Information Rate : 0.8299
    P-Value [Acc > NIR] : 0.00000098822

                  Kappa : 0.5345

 Mcnemar's Test P-Value : 0.00000005724

            Sensitivity : 0.9699
            Specificity : 0.4867
         Pos Pred Value : 0.9022
         Neg Pred Value : 0.7684
             Prevalence : 0.8299
         Detection Rate : 0.8050
   Detection Prevalence : 0.8923
      Balanced Accuracy : 0.7283

       'Positive' Class : No
```

```
> confusionMatrix(pred_valid2, as.factor(valid$Attrition))
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  464  52
       Yes  37  35

               Accuracy : 0.8486
                 95% CI : (0.8171, 0.8766)
    No Information Rate : 0.852
    P-Value [Acc > NIR] : 0.6190

                  Kappa : 0.3536

 Mcnemar's Test P-Value : 0.1378

            Sensitivity : 0.9261
            Specificity : 0.4023
         Pos Pred Value : 0.8992
         Neg Pred Value : 0.4861
             Prevalence : 0.8520
         Detection Rate : 0.7891
   Detection Prevalence : 0.8776
      Balanced Accuracy : 0.6642

       'Positive' Class : No
```

```
# 12. c
# When we use the huge tree, it has all the data to fit the training set.
# However, it overfitted when we used it to the validation set. As a result,
# the difference in between is also huge. When we switched to a pruned tree,
# there is no overfitting, so the difference between two sets are low.

# Task 2: Association rules
install.packages('arules')
library(arules)
data(Groceries)

# 1.
class(Groceries)
Groceries
# The class of "Groceries" is transactions.
# Groceries contain 9835 rows & 169 columns.
```

```r
# 2.
itemFrequencyPlot(Groceries, topN = 15, type = "absolute",
                  main = "Item Frequency", horiz = TRUE, col = "pink")
```



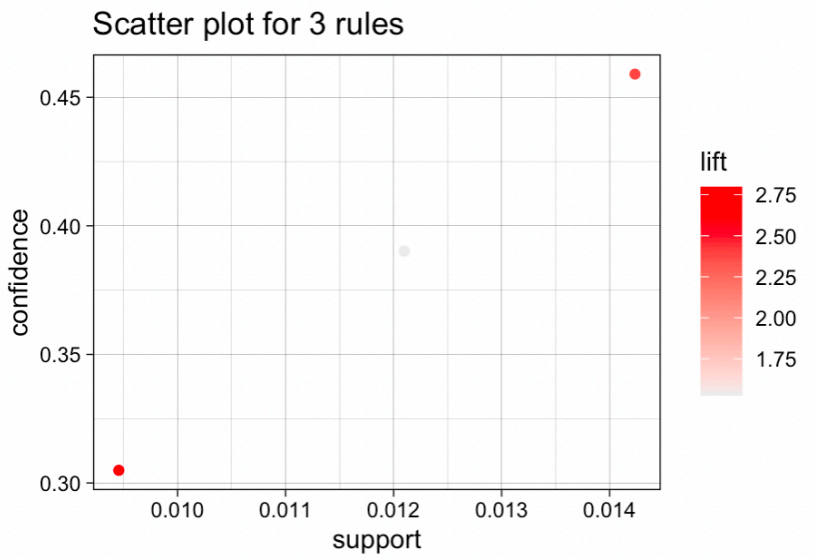

```r
# 3.
# LHS
# To find out what products were purchased after/along with product X

# The is a case to find out the Customers who bought 'onions' also bought..
# In the equation, 'onions' is in LHS (left hand side).

rules_LHS <- apriori(data = Groceries, # those who bought 'onions' also bought..
                 parameter = list(supp = 0.001, conf = 0.15, minlen = 2),
                 appearance = list(default = "rhs", lhs = "onions"),
                 control = list(verbose = F))

rules_LHS_conf <- sort(rules_LHS, # 'high-confidence' rules
                       by = "confidence", decreasing = TRUE)
inspect(head(rules_LHS_conf))
```

```
> inspect(head(rules_LHS_conf))
    lhs         rhs                  support     confidence coverage   lift     count
[1] {onions} => {other vegetables} 0.014234875 0.4590164  0.03101169 2.372268 140
[2] {onions} => {whole milk}       0.012099644 0.3901639  0.03101169 1.526965 119
[3] {onions} => {root vegetables}  0.009456024 0.3049180  0.03101169 2.797452  93
[4] {onions} => {yogurt}           0.007219115 0.2327869  0.03101169 1.668702  71
[5] {onions} => {rolls/buns}       0.006812405 0.2196721  0.03101169 1.194293  67
[6] {onions} => {bottled water}    0.005897306 0.1901639  0.03101169 1.720572  58
```

```r
# RHS
# To find what factors influenced purchase of product X

# To find out what customers had purchased before buying 'onions'. This will
# help you understand the patterns that led to the purchase of 'onions'.

rules_RHS <- apriori(data = Groceries, # get rules that lead to buying 'onions'
                     parameter = list(supp = 0.001, conf = 0.08),
                     appearance = list(default = "lhs", rhs = "onions"),
                     control = list(verbose = F))

rules_RHS_lift <- sort(rules_RHS, # 'high-lift' rules
                       by = "lift", decreasing = TRUE)
inspect(head(rules_RHS_lift))

# Hi roommate, the first model with onions on the left hand side showed that
# there is a great chance customers who bought 'onions' also bought other
# vegetables, whole milk, root vegetables, yogurt and so on...

# The second model with onions on the right hand side showed that what customers
# had purchased before buying 'onions'. The most likely purchase patterns is
# root vegetables, other vegetables, whole milk, butter, and then onions.
```

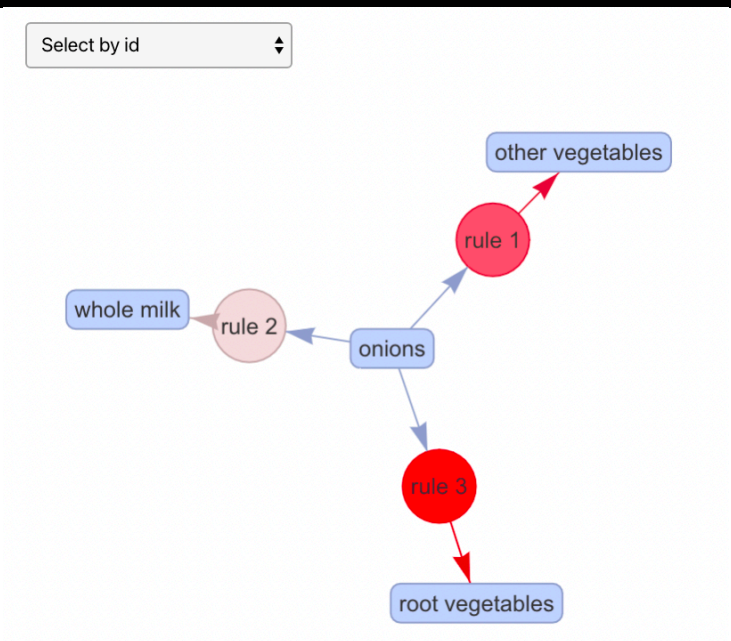
```
> inspect(head(rules_RHS_lift))
    lhs                                                rhs      support     confidence coverage    lift      count
[1] {root vegetables, other vegetables, whole milk, butter} => {onions} 0.001321810 0.3170732  0.004168785 10.224310 13
[2] {tropical fruit, butter, yogurt}                       => {onions} 0.001118454 0.2444444  0.004575496  7.882332 11
[3] {root vegetables, other vegetables, butter}            => {onions} 0.001525165 0.2307692  0.006609049  7.441362 15
[4] {beef, tropical fruit, other vegetables}               => {onions} 0.001016777 0.2272727  0.004473818  7.328614 10
[5] {root vegetables, other vegetables, napkins}           => {onions} 0.001016777 0.2222222  0.004575496  7.165756 10
[6] {root vegetables, other vegetables, frozen vegetables} => {onions} 0.001321810 0.2166667  0.006100661  6.986612 13
```

```r
# 4.
# These rules could help a store like Star Market to do target marketing.
# Like the example we learned in the lecture, it is possible to know the body
# condition of the customer, such as whether they are pregnant or bold by
# analyzing these buying patterns.

# 5.
install.packages('arulesViz')
library(arulesViz)

df <- head(sort(rules_LHS, by = "confidence"), 3)
plot(df)
inspect(df)
# The plot showed the relationship between confidence, support, and lift.
# The reddest point has the highest lift value, but has the lowest support and
# confidence. The second reddest point has the middle lift value, but has the
# highest value of support and confidence. The last point with the lowest lift
# value has the middle value of support and confidence.
```



5.

```
# 6.
plot(df, method = "graph", engine = "htmlwidget")
# I see a plot with onions in the middle, and three rules around it with sort
# by ID. The size of rules are the same, but the color of it are different.
# It is the same color from the previous plot: red, orange, and grey.
# It means customers who bought onions are most likely to buy root vegetables,
# other vegetables, and whole milk.
```



6.