Homework 2

Chike Odenigbo

November 6, 2022

```
get_accuracy <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]
  accuracy = round((TP + TN) / sum(TP,FP,TN,FN), 2)
  return(accuracy)
}

get_classification_error_rate <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]
  classification_error_rate = round((FP + FN) / sum(TP,FP,TN,FN),2)
  return(classification_error_rate)
}

get_precision <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]
  precision = round(TP / (TP + FP), 2)
  return(precision)
}


get_sensitivity <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]
  sensitivity = round(TP / (TP + FN), 2)
  return(sensitivity)
}

get_specificity <- function(predicted, actual){
```

```
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]
  specificity = round(TN / (TN + FP), 2)
  return(specificity)
}

get_f1_score <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]

  precision = round(TP / (TP + FP), 2)
  sensitivity = round(TP / (TP + FN), 2)
  f1_score = round((2 * precision * sensitivity) / (precision + sensitivity), 2)
  return(f1_score)
}

get_false_positive_rate <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]

  fpr = round(FP / (FP + TN), 2)
  return(fpr)
}

get_false_negative_rate <- function(predicted, actual){
  confusion_table = table(predicted, actual)
  TP = confusion_table[2,2]
  TN = confusion_table[1,1]
  FN = confusion_table[1,2]
  FP = confusion_table[2,1]

  fnr = round(FN / (FN + TP), 2)
  return(fnr)
}
```

Section 1

Question 1.1

Create a training set of 800 observations, and a test set containing the rest using the following code:

```
# ```{r pressure, echo=FALSE}
n <- nrow(OJ)
set.seed(1234)
id.train=sample(1:n,size=800)
```

```
id.test=setdiff(1:n,id.train)
OJ$Purchase = as.factor(OJ$Purchase)
OJ$Store7 = as.factor(OJ$Store7)
as.data.frame(sapply(OJ,class))
```

```
##                sapply(OJ, class)
## Purchase                  factor
## WeekofPurchase           numeric
## StoreID                  numeric
## PriceCH                  numeric
## PriceMM                  numeric
## DiscCH                   numeric
## DiscMM                   numeric
## SpecialCH                numeric
## SpecialMM                numeric
## LoyalCH                  numeric
## SalePriceMM              numeric
## SalePriceCH              numeric
## PriceDiff                numeric
## Store7                    factor
## PctDiscMM                numeric
## PctDiscCH                numeric
## ListPriceDiff            numeric
## STORE                    numeric
```

```
OJ.train=OJ[id.train,-3]
OJ.test=OJ[id.test,-3]
```

Question 1.2

Construct an unpruned classification tree to predict the variable purchase using the available predictors. Calculate the false positive rate, false negative rate and overall error rate of this tree on the test data (note: you can use your code from the previous assignment directly for this question).

```
# ```{r pressure, echo=FALSE}
tree <- rpart(OJ.train$Purchase ~ . ,data = OJ.train, method = 'class')
#summary(tree)

pred = predict(tree,newdata = OJ.test,type = c("class"))

print(paste0("False Positive Rate: ", get_false_positive_rate(OJ.test$Purchase,pred)))
```

```
## [1] "False Positive Rate: 0.14"
```

```
print(paste0("Error Rate: ", get_classification_error_rate(OJ.test$Purchase,pred)))
```

```
## [1] "Error Rate: 0.19"
```

```
print(paste0("False Negative Rate: ", get_false_negative_rate(OJ.test$Purchase,pred)))
```

```
## [1] "False Negative Rate: 0.27"
```

Question 1.3

Using all the data (training and test put together), use the bagging approach to do the same analysis again and compare the results of the different error rates.

```
set.seed(1)
n_pred <- ncol(OJ.train) - 1

bag.OJ <- randomForest(OJ.train$Purchase ~ .,
                           data = OJ.train,
                           mtry=n_pred,
                           importance=TRUE)

bag.pred = predict(bag.OJ,newdata = OJ.test,type = c("class"))

print(paste0("False Positive Rate: ", get_false_positive_rate(OJ.test$Purchase,bag.pred)))
```

```
## [1] "False Positive Rate: 0.15"
```

```
print(paste0("Error Rate: ", get_classification_error_rate(OJ.test$Purchase,bag.pred)))
```

```
## [1] "Error Rate: 0.21"
```

```
print(paste0("False Negative Rate: ", get_false_negative_rate(OJ.test$Purchase,bag.pred)))
```

```
## [1] "False Negative Rate: 0.31"
```

Question 1.4

Using all the data (training and test put together), use the random forest approach to redo the same analysis and compare the results of the different error rates.

```
set.seed(1)

rf.OJ <- randomForest(OJ.train$Purchase ~ .,
                           data = OJ.train,
                           #mtry=n_pred,
                           importance=TRUE)

rf.pred = predict(rf.OJ,newdata = OJ.test,type = c("class"))

print(paste0("False Positive Rate: ", get_false_positive_rate(OJ.test$Purchase,rf.pred)))
```

```
## [1] "False Positive Rate: 0.17"
```

```
print(paste0("Error Rate: ", get_classification_error_rate(OJ.test$Purchase,rf.pred)))
```

```
## [1] "Error Rate: 0.21"
```

```
print(paste0("False Negative Rate: ", get_false_negative_rate(OJ.test$Purchase,rf.pred)))
```

```
## [1] "False Negative Rate: 0.28"
```

Question 1.5

Calculate the importance of the variables in the classification tree constructed in 2) and the forest constructed in 4). Compare.

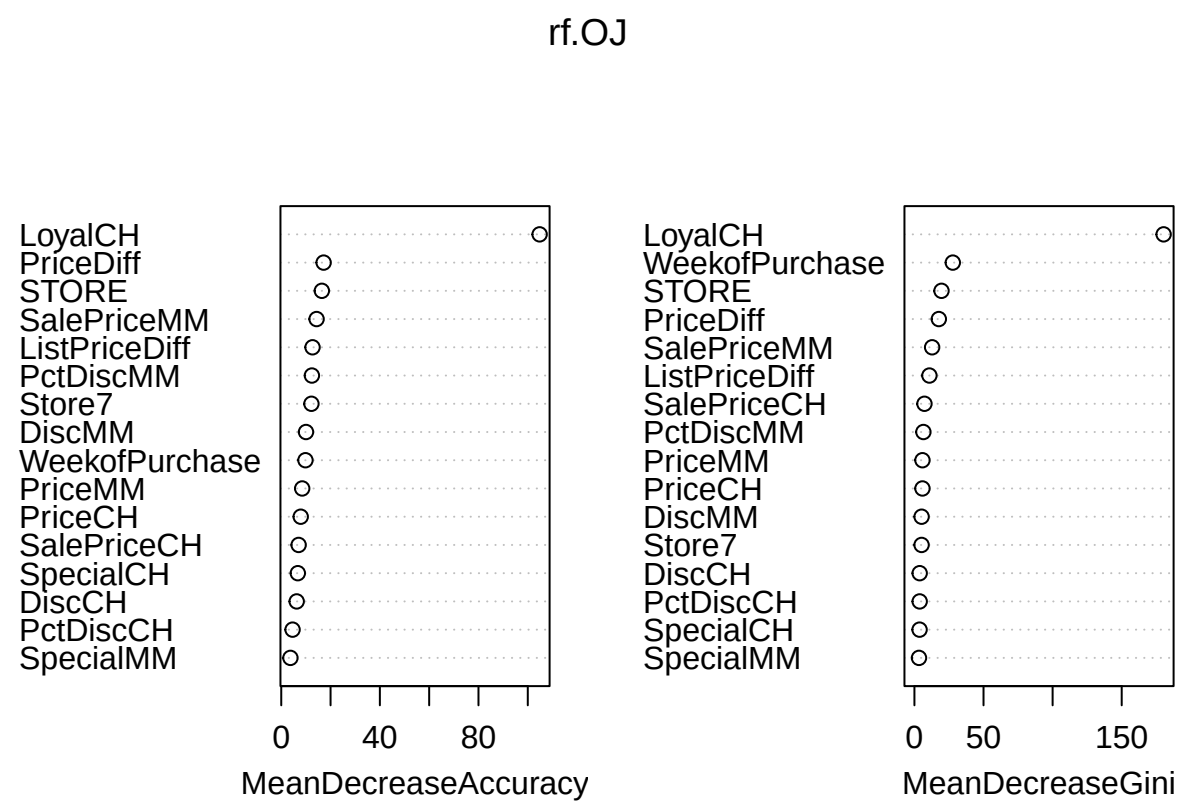
```
set.seed(1)

as.data.frame(importance(rf.OJ))
```

```
##                         CH        MM MeanDecreaseAccuracy MeanDecreaseGini
## WeekofPurchase  6.2370043  6.263470             9.821831        27.774264
```

```
## PriceCH          6.4129690  3.347700           7.894530         5.809040
## PriceMM          3.6162856  7.199457           8.496137         5.816289
## DiscCH           3.2356695  5.311119           6.259811         3.814248
## DiscMM           5.1714256  6.529393          10.040448         5.196968
## SpecialCH        2.1385953  5.812328           6.715932         3.727776
## SpecialMM       -0.4787197  4.741747           3.719254         3.312229
## LoyalCH         73.7608676 98.090066         104.804821       180.327296
## SalePriceMM      6.9341087 11.630595          14.330011        12.811005
## SalePriceCH      5.3705321  3.347173           7.054213         7.263904
## PriceDiff        9.6259636 12.248643          17.187297        17.667201
## Store7           4.4216587 12.366033          12.239986         5.174581
## PctDiscMM        7.1044541  8.672862          12.428228         6.497433
## PctDiscCH        2.3521748  4.010164           4.593497         3.812999
## ListPriceDiff    8.0987352  8.405751          12.687143        10.860433
## STORE            7.2910690 15.037911          16.467642        19.595198
```

```
varImpPlot(rf.OJ)
```




```
set.seed(1)

as.data.frame(tree$variable.importance)
```

```
##                tree$variable.importance
## LoyalCH                      170.416755
## PriceDiff                     21.478485
## SalePriceMM                   20.474434
## ListPriceDiff                 15.244049
```

```
## PctDiscMM                    13.723186
## DiscMM                       12.957004
## PriceMM                       7.244905
## SpecialMM                     7.151039
## WeekofPurchase                5.708685
## SpecialCH                     5.376632
## STORE                         5.077294
## PriceCH                       2.477877
## PctDiscCH                     1.722996
## DiscCH                        1.656462
## SalePriceCH                   1.369650
```

```
#varImpPlot(tree)
```

Question 1.6

Using all the data (training and test put together), use the boosting approach to do the same analysis again and compare the results of the error rates between all tested methods.

```
set.seed(1)

boost.OJ = boosting(Purchase ~ .,
                    data=OJ.train,
                    #mfinal = 100,
                    coeflearn = 'Freund')#,
                    #control=rpart.control(maxdepth=10))
boost.pred = predict(boost.OJ,newdata = OJ.test,type = c("class"))

print(paste0("False Positive Rate: ", get_false_positive_rate(OJ.test$Purchase,boost.pred$class)))
```

```
## [1] "False Positive Rate: 0.17"
```

```
print(paste0("Error Rate: ", get_classification_error_rate(OJ.test$Purchase,boost.pred$class)))
```

```
## [1] "Error Rate: 0.23"
```

```
print(paste0("False Negative Rate: ", get_false_negative_rate(OJ.test$Purchase,boost.pred$class)))
```

```
## [1] "False Negative Rate: 0.31"
```

Section 2

Question 2.1

Create a training and test dataset of size 300 and 100 respectively using the following code:

```
set.seed(123456)
n=nrow(Carseats)
id.train=sample(1:n,size=300)
id.test=setdiff(1:n,id.train)
Carseat.train=Carseats[id.train,]
Carseat.test=Carseats[id.test,]
```

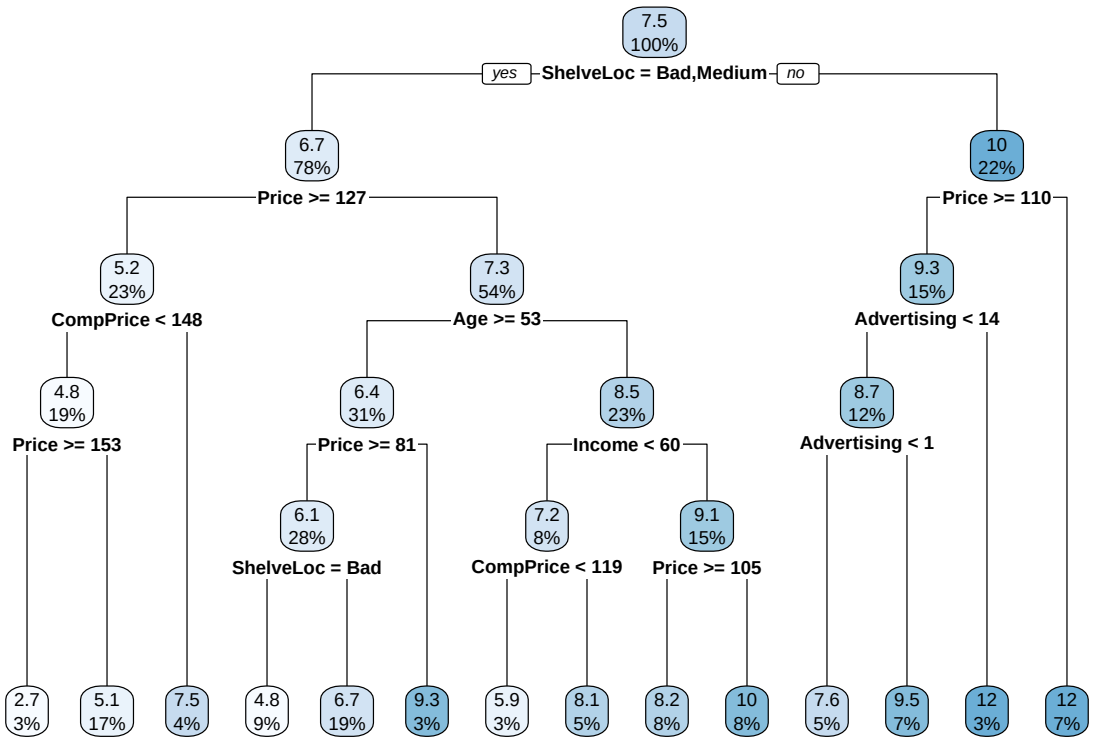
Question 2.2

Construct a regression tree predicting the variable Sales from the other variables available in the training sample. Graph the tree and interpret the results.

ANSWER: *** ***

```
tree.reg <- rpart(Sales ~ . ,data = Carseat.train, method = 'anova')
```

```
rpart.plot(tree.reg)
```



Question 2.3

Calculate the MSE on the test data.

```
pred.reg.tree = predict(tree.reg, newdata = Carseat.test)
mean((pred.reg.tree - Carseat.test$Sales)^2)
```

```
## [1] 4.297449
```

Question 2.4

Use cross-validation to determine the optimal level of complexity parameter to prune the tree. Does pruning improve the MSE?

```
set.seed(400)
data_ctrl = trainControl(method = "cv", number = 10)
mytree.caret = train(Sales~.,data=Carseat.train, method = "rpart",  trControl=data_ctrl)
```

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

```
#mytree.caret$resample
#mytree.caret$results
```

```
mytree.caret$bestTune
```

```
##           cp
## 1 0.06932427
```

```
pred.cv.tree = predict(mytree.caret,newdata = Carseat.test)
mean((pred.cv.tree - Carseat.test$Sales)^2)
```

```
## [1] 4.849398
```

Question 2.5

Repeat the previous question several times (with a different seed) - and look at the differences between the results obtained. What do you conclude?

```
#set.seed(0)
seed.list = sample.int(100, 10)
df = data.frame()

for (seed in seed.list){
  set.seed(seed)
  data_ctrl = trainControl(method = "cv", number = 10)
  mytree.caret = train(Sales~.,data=Carseat.train, method = "rpart",  trControl=data_ctrl)
  #mytree.caret$resample
  #mytree.caret$results
  mytree.caret$bestTune
  pred.cv.tree = predict(mytree.caret,newdata = Carseat.test)
  mse = mean((pred.cv.tree - Carseat.test$Sales)^2)
  output = c(seed,mse)
  df = rbind(df, output)
}
```

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

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

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

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

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

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

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

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

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

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

```
colnames(df)<-c("seed", "mse")
df
```

```
##    seed      mse
## 1    19 4.849398
## 2    86 4.849398
## 3    29 4.849398
## 4    52 4.849398
## 5    25 4.849398
## 6    73 4.849398
## 7    72 4.849398
## 8    26 4.849398
## 9    81 4.849398
## 10   34 4.849398
```

Question 2.6

Repeat the previous question several times (with a different seed) - and look at the differences between the results obtained. What do you conclude?

```
set.seed(1)
n_pred <- ncol(Carseat.train) - 1

bag.reg <- randomForest(Carseat.train$Sales ~ .,
                        data = Carseat.train,
                        mtry=n_pred,
                        importance=TRUE)

bag.reg.pred = predict(bag.reg,newdata = Carseat.test)#,type = c("anova"))
mean((bag.reg.pred - Carseat.test$Sales)^2)
```

```
## [1] 2.004661
```

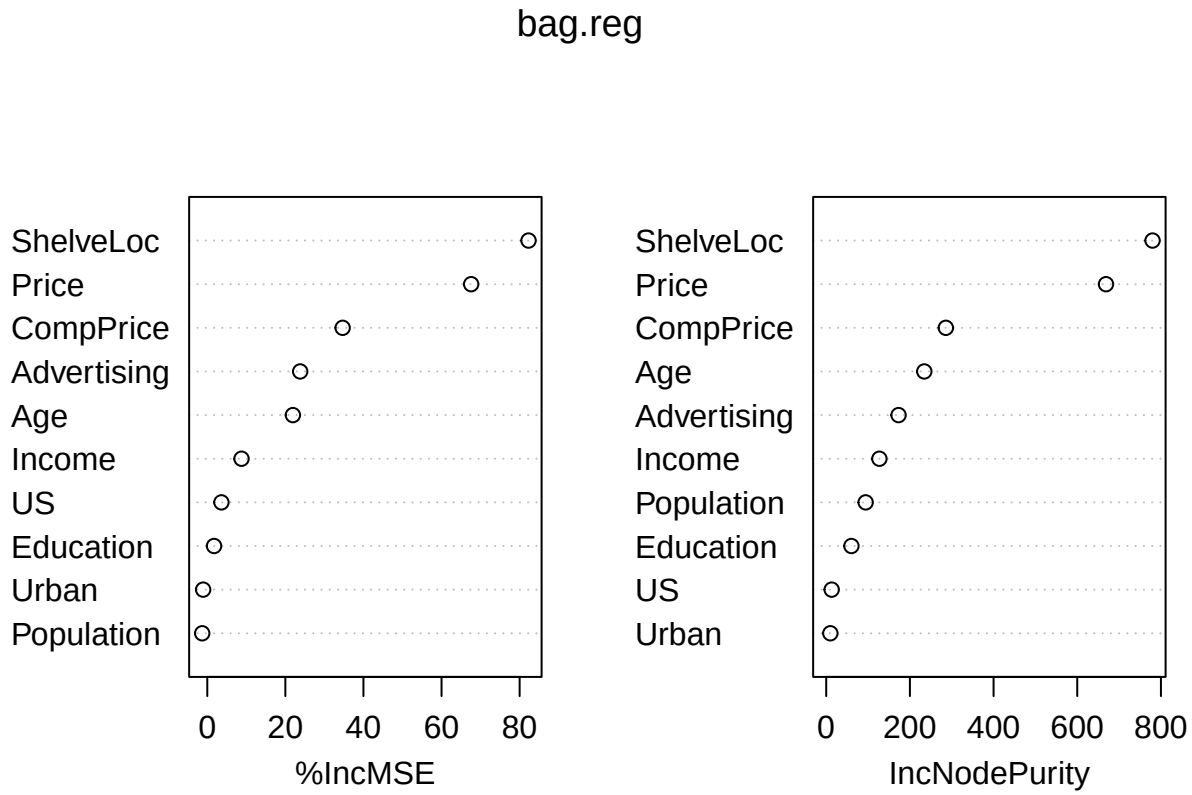
Question 2.7

Use the ***importance*** function to evaluate the importance of variables. Compare with the tree constructed at the beginning of the exercise.

```
as.data.frame(importance(bag.reg))
```

```
##               %IncMSE IncNodePurity
## CompPrice   34.671069    285.920479
## Income       8.766851    127.099848
## Advertising 23.797787    172.906967
## Population  -1.299732     94.290903
## Price       67.591794    668.723626
## ShelveLoc   82.286261    779.935615
## Age         21.926814    234.454410
## Education    1.736532     60.274706
## Urban       -1.056307      9.904163
## US           3.621940     13.064970
```

```
varImpPlot(bag.reg)
```



Question 2.8 Repeat the 2 previous questions using the random forest algorithm. Comment.**

```
set.seed(1)
n_pred <- ncol(Carseat.train) - 1

rf.reg <- randomForest(Carseat.train$Sales ~ .,
                       data = Carseat.train,
                       #mtry=n_pred,
                       importance=TRUE)

rf.reg.pred = predict(rf.reg,newdata = Carseat.test)#,type = c("anova"))
mean((rf.reg.pred - Carseat.test$Sales)^2)
```

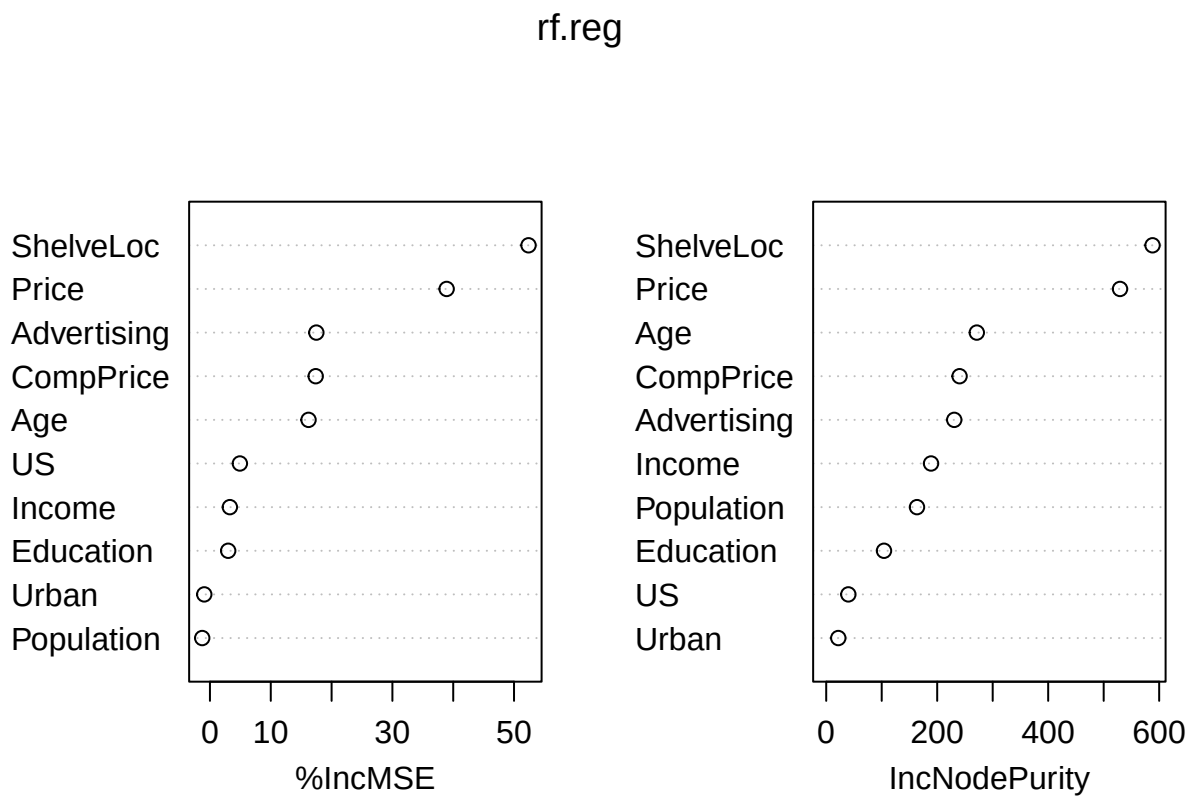
```
## [1] 2.084574
```

```
as.data.frame(importance(rf.reg))
```

```
##                %IncMSE IncNodePurity
## CompPrice   17.3901890     240.34938
## Income       3.2732690     188.93112
## Advertising 17.4952006     230.71428
## Population  -1.2745196     163.59608
## Price       38.9194656     529.45620
## ShelveLoc   52.3866656     588.06611
## Age         16.2092317     271.36590
## Education    2.9971029     104.13966
```

```
## Urban       -0.9348852      21.80153
## US           4.9195707      39.92127
```

```
varImpPlot(rf.reg)
```



Section 3

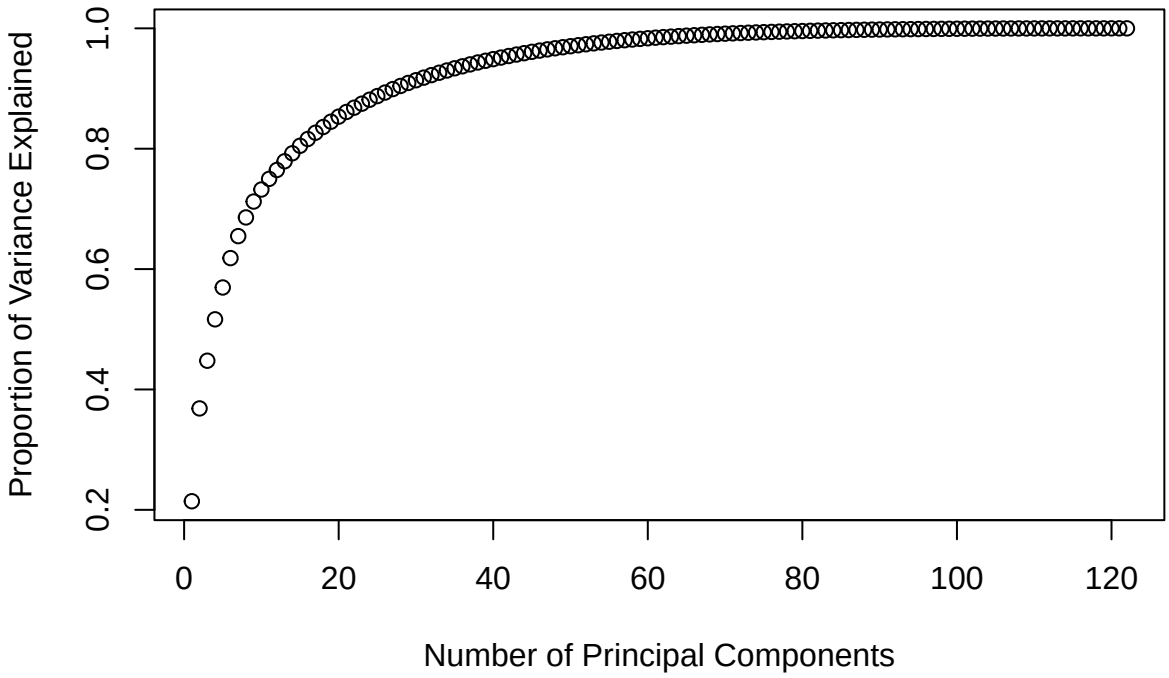
Question 3.1

Take the data and example from the principal component regression tutorial. Using the graph of the number of PCs versus the cumulative proportion of variance explained, identify a number of PCs at which the variance explained seems to increase only marginally (note: I don't expect you all to have the same answer). Run a principal component regression with this number of PCs and compare your results with those in the tutorial (MSE of the validation set).

ANSWER: ***I would save the first 50 components***

```
set.seed(60603)
id=sample(1:nrow(crimeData))
crimeData.pca = crimeData[id[1:1450],][,-123]
crimeData.train=data.frame(apply(crimeData[id[1:1450],],2,scale))
crimeData.valid=data.frame(apply(crimeData[id[1451:nrow(crimeData)],],2,scale))
pca.res = prcomp(crimeData.pca, center = TRUE, scale. = TRUE)
#summary(pca.res)$importance
#round(pca.res$rotation,2)
#data.frame(pca.res$x)

plot(summary(pca.res)$importance[3,], xlab = "Number of Principal Components", ylab = "Proportion of Va
```




```
model.crimeData.pca = pcr(y~., data = crimeData.train)


pred.valid.tutorial=predict(model.crimeData.pca,newdata=crimeData.valid,ncomp=15)
pred.valid.answer=predict(model.crimeData.pca,newdata=crimeData.valid,ncomp=50)
MSE.valid.tutorial = sqrt(mean((crimeData.valid$y-pred.valid.tutorial)^2))
MSE.valid.tutorial
```

```
## [1] 0.6343372
```

```
MSE.valid.answer = sqrt(mean((crimeData.valid$y-pred.valid.answer)^2))
MSE.valid.answer
```

```
## [1] 0.620323
```

```
rbind(MSE.valid.tutorial,MSE.valid.answer)
```

```
##                          [,1]
## MSE.valid.tutorial 0.6343372
## MSE.valid.answer   0.6203230
```

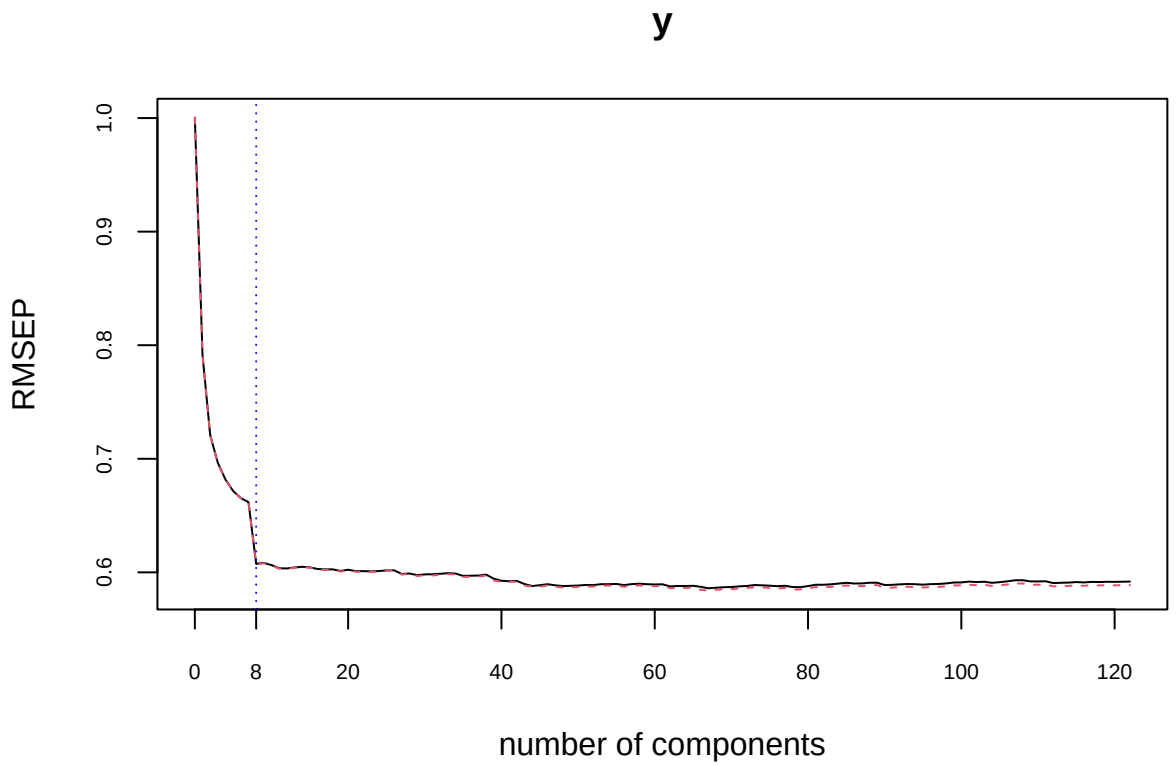
Question 3.2

It is also possible to choose the number of optimal PCs by using other criteria than the explained variance, such as the MSEP (mean squared error of prediction) or the R2 calculated by cross validation. You can obtain the graph of the number of PCs versus one of these measures with the following command (for the MSEP):

ANSWER: ***I would save the first 50 components***

```r
set.seed(60603)
id=sample(1:nrow(crimeData))
crimeData.pca = crimeData[id[1:1450],][,-123]
crimeData.train=data.frame(apply(crimeData[id[1:1450],],2,scale))
crimeData.valid=data.frame(apply(crimeData[id[1451:nrow(crimeData)],],2,scale))
pca.res = prcomp(crimeData.pca, center = TRUE, scale. = TRUE)

model.pcr=pcr(y~., data = crimeData.train, validation="CV")
validationplot(model.pcr, val.type="RMSEP", cex.axis=0.7)
axis(side = 1, at = c(8), cex.axis=0.7)
abline(v = 8, col = "blue", lty = 3)
```



Question 3.3

Repeat the previous question using the R2, with the option **val.type="R2" ANSWER:** ***I would save the first 50 components***

#"'{r} set.seed(60603) id=sample(1:nrow(crimeData)) crimeData.pca = crimeData[id[1:1450],][,-123] crime-Data.train=data.frame(apply(crimeData[id[1:1450],],2,scale)) crimeData.valid=data.frame(apply(crimeData[id[1451:nrow(crime pca.res = prcomp(crimeData.pca, center = TRUE, scale. = TRUE)

model.pcr=pcr(y~., data = crimeData.train, validation="CV") validationplot(model.pcr, val.type="R2", cex.axis=0.7) axis(side = 1, at = c(8), cex.axis=0.7) abline(v = 8, col = "blue", lty = 3) #"'