# Big data: Classification (Titanic dataset)

*Ignacio Amaya and Justin Vailhere*

*November 4, 2015*

### Getting and cleaning the data

We read the Titanic file from the subject folder.

```
titanic <- read.csv("C:\\Users\\Ignacio\\Documents\\MIS DOCUMENTOS\\DATA SCIENCE MASTER (EIT DIGITAL)\\I
```

The variables in this data set are:

| Key | Description |
|---|---|
| pclass | Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd) |
| survival | Survival (0 = No; 1 = Yes) |
| name | Name |
| sex | Sex |
| age | Age |
| sibsp | Number of Siblings/Spouses Aboard |
| parch | Number of Parents/Children Aboard |
| ticket | Ticket Number |
| fare | Passenger Fare |
| cabin | Cabin |
| embarked | Port of Embarkation (C,Q or S) |
| boat | Lifeboat |
| body | Body Identification Number |
| home.dest | Home/Destination |

We can have an overview of all the values we have.

```
summary(titanic)
```

```
##   PassengerId       Survived          Pclass
##  Min.   :  1.0   Min.   :0.0000   Min.   :1.000
##  1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000
##  Median :446.0   Median :0.0000   Median :3.000
##  Mean   :446.0   Mean   :0.3838   Mean   :2.309
##  3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000
##  Max.   :891.0   Max.   :1.0000   Max.   :3.000
##
##                                     Name         Sex           Age
##  Abbing, Mr. Anthony                  :  1   female:314   Min.   : 0.42
##  Abbott, Mr. Rossmore Edward          :  1   male  :577   1st Qu.:20.12
##  Abbott, Mrs. Stanton (Rosa Hunt)     :  1                Median :28.00
##  Abelson, Mr. Samuel                  :  1                Mean   :29.70
##  Abelson, Mrs. Samuel (Hannah Wizosky):  1                3rd Qu.:38.00
##  Adahl, Mr. Mauritz Nils Martin       :  1                Max.   :80.00
##  (Other)                              :885                NA's   :177
```

```
##      SibSp          Parch            Ticket         Fare       
##  Min.   :0.000   Min.   :0.0000   1601    :  7   Min.   :  0.00  
##  1st Qu.:0.000   1st Qu.:0.0000   347082  :  7   1st Qu.:  7.91  
##  Median :0.000   Median :0.0000   CA. 2343:  7   Median : 14.45  
##  Mean   :0.523   Mean   :0.3816   3101295 :  6   Mean   : 32.20  
##  3rd Qu.:1.000   3rd Qu.:0.0000   347088  :  6   3rd Qu.: 31.00  
##  Max.   :8.000   Max.   :6.0000   CA 2144 :  6   Max.   :512.33  
##                                   (Other) :852                   
##          Cabin     Embarked  
##  B96 B98    :  4   C   :168  
##  C23 C25 C27:  4   Q   : 77  
##  G6         :  4   S   :644  
##  C22 C26    :  3   NA's:  2  
##  D          :  3             
##  (Other)    :186             
##  NA's       :687             
```

Some variables stored in the dataframe are treated as numerical, but they are categorical. So we are going to transform them into categorical variables.

```
titanic$Survived=as.factor(titanic$Survived)
titanic$Pclass=as.factor(titanic$Pclass)
titanic$SibSp=as.factor(titanic$SibSp)
titanic$Parch=as.factor(titanic$Parch)
```

**Question 1: Which are the variables that are discarded and why?** We want to discard those variables that don't give us any information in order to predict if a passenger lives or dies. There are some variables that don't give us any extra information, because they have a lot of missing values (for example, the cabin variable). Others are also discarded because their values are all different (for example, the names), so we can't extract information from them.

```
titanic_s1=subset.data.frame(titanic,select=c(Survived,Pclass,Sex,Age,SibSp,Parch,Fare,Embarked))
```

We can get a summary with the information of the subset we've selected.

```
summary(titanic_s1)
```

```
##  Survived Pclass      Sex           Age        SibSp   Parch  
##  0:549    1:216   female:314   Min.   : 0.42   0:608   0:678  
##  1:342    2:184   male  :577   1st Qu.:20.12   1:209   1:118  
##           3:491                Median :28.00   2: 28   2: 80  
##                                Mean   :29.70   3: 16   3:  5  
##                                3rd Qu.:38.00   4: 18   4:  4  
##                                Max.   :80.00   5:  5   5:  5  
##                                NA's   :177     8:  7   6:  1  
##       Fare        Embarked  
##  Min.   :  0.00   C   :168  
##  1st Qu.:  7.91   Q   : 77  
##  Median : 14.45   S   :644  
##  Mean   : 32.20   NA's:  2  
##  3rd Qu.: 31.00             
##  Max.   :512.33             
## 
```

We observe that we have 177 NA in age and 2 NA in Embarked. We don't want to have missing values, so we remove the entries where those NA appear. Our final cleaned dataset has 712 observations of 8 variables.

```
titanic_cleaned=na.omit(titanic_s1,titanic_s1$Age,titanic_s1$Embarked)
summary(titanic_cleaned)
```

```
##  Survived Pclass      Sex           Age            SibSp   Parch  
##  0:424    1:184   female:259   Min.   : 0.42   0:469   0:519  
##  1:288    2:173   male  :453   1st Qu.:20.00   1:183   1:110  
##           3:355                Median :28.00   2: 25   2: 68  
##                                Mean   :29.64   3: 12   3:  5  
##                                3rd Qu.:38.00   4: 18   4:  4  
##                                Max.   :80.00   5:  5   5:  5  
##                                                8:  0   6:  1  
##       Fare        Embarked
##  Min.   :  0.00   C:130   
##  1st Qu.:  8.05   Q: 28   
##  Median : 15.65   S:554   
##  Mean   : 34.57           
##  3rd Qu.: 33.00           
##  Max.   :512.33           
## 
```

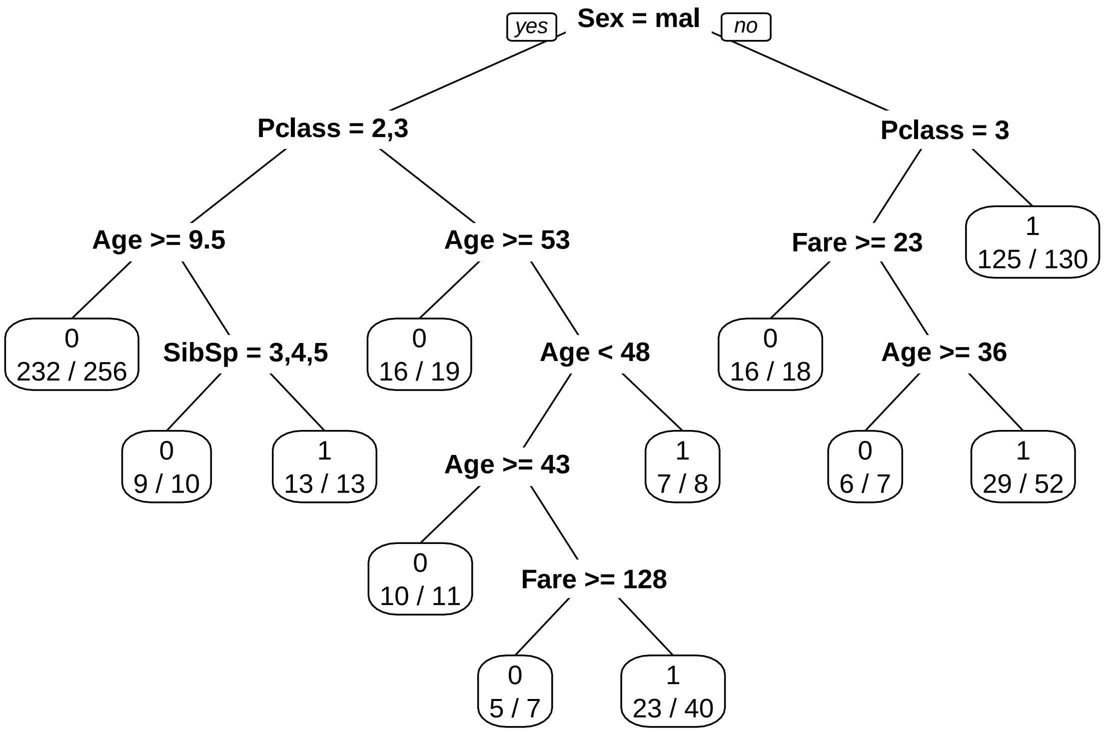
## Training a single decison tree

**Question 2: Plot the tree with the prp() function (package rpart.plot that we must install if it is not already installed). Which are the most important variables according to the tree? Does the tree change if we rerun the rpart() function? Why?** First of all, we load the packages we need.

```
library(caret)
library(rpart)
library(rpart.plot)
```

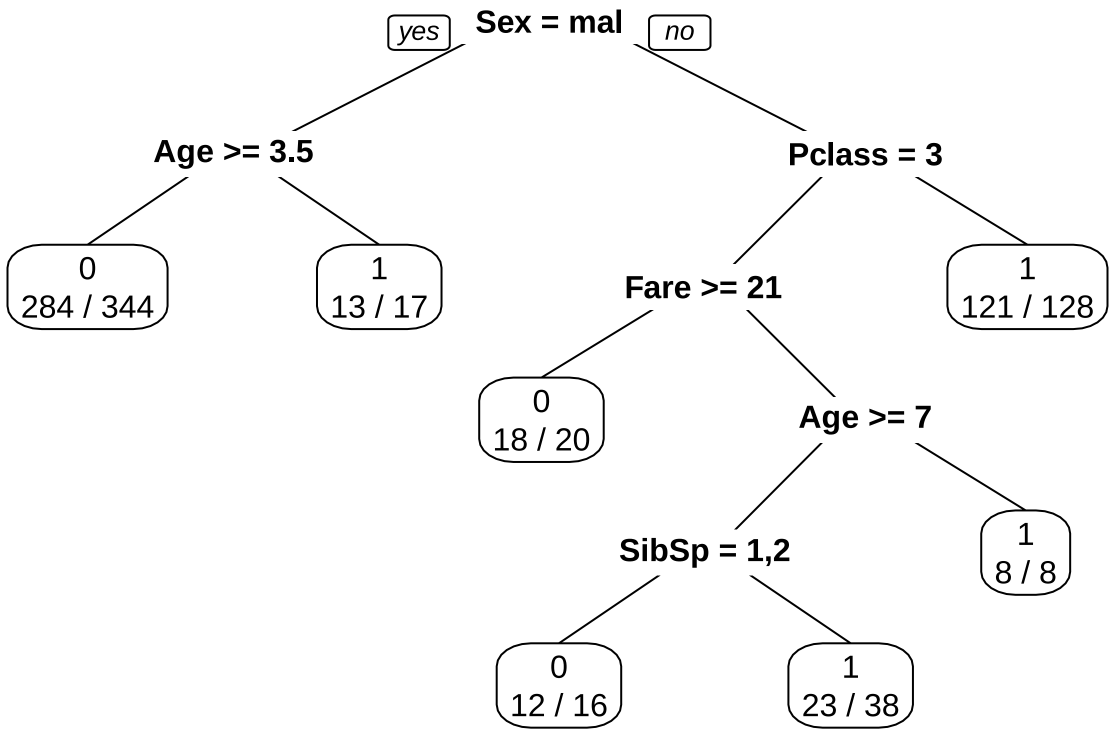
We set the seed, so that way all the values we are going to get each time we execute the code are going to be the same. We divide our data into a training set and a testing set (80% of the entries are going to the training set and 20% to the testing set).

```
set.seed(100)
partition=createDataPartition(titanic_cleaned$Survived,p = 0.8,list=FALSE)
train_set=titanic_cleaned[partition,]
test_set=titanic_cleaned[-partition,]

random_tree=rpart(train_set)

prp(random_tree,extra=2)
```




The most important variables are the sex and the Pclass. If the passenger is a woman and is not in third class, then the probability of her surviving is very high (only 5 out of 130 in this case died). If we rerun only the rpart function the tree obtained doesn't change. This is because the partition used is the same. But if we rerun the partition without setting the seed we would obtain different results, as we have a different partition. An exmple of this would be:

```
partition2=createDataPartition(titanic_cleaned$Survived,p = 0.8,list=FALSE)
train_set2=titanic_cleaned[partition2,]
test_set2=titanic_cleaned[-partition2,]
random_tree2=rpart(train_set2)
prp(random_tree2,extra=2)
```




```r
testPred <- predict(random_tree, test_set,type="class")
confusionMatrix(testPred,test_set$Survived)
```

**Question 3: Which are the performance values (accuracy, sensitivity, specificity, etc.) of the learned model on the testing subset?**

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 63 14
##          1 21 43
##
##                Accuracy : 0.7518
##                  95% CI : (0.6721, 0.8206)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 7.372e-05
##
##                   Kappa : 0.4946
##  Mcnemar's Test P-Value : 0.3105
##
##             Sensitivity : 0.7500
##             Specificity : 0.7544
```

```
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.6719
##              Prevalence : 0.5957
##          Detection Rate : 0.4468
##    Detection Prevalence : 0.5461
##       Balanced Accuracy : 0.7522
##
##        'Positive' Class : 0
##
```

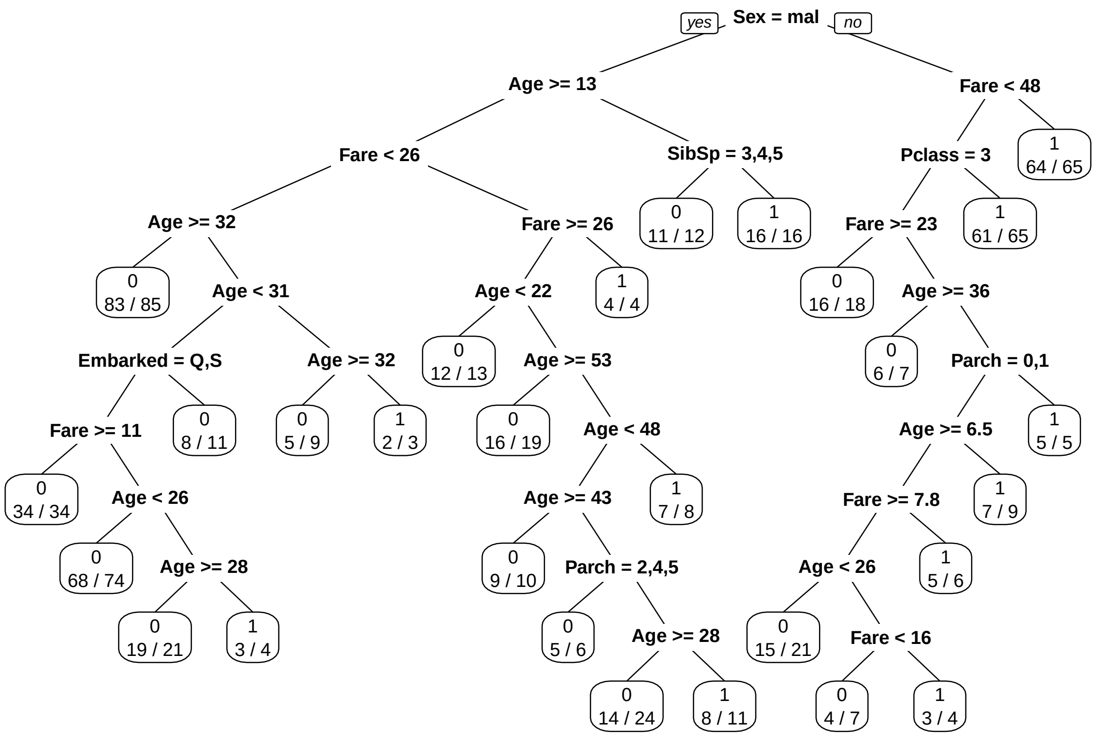
As we can see the performance is good. We have an accuracy above 75% and the sensitivity (true positive rate) and specificity (true negative rate) are also good. As they have similar values we know that the predictions of people dying and surviving have the same rate of success. In the confusion we can see that the decision tree failed in 35 cases out of 141 (number of entries in our testing set).

**Question 4: We have learned a decision tree model with the CART algorithm. Which are the values for the parameters of this algorithm (minbucket, minsplit, complexity parameter, cost, etc.)? (CLUE: check the rpart() function documentation.)** The default parameters for the rpart function are: *rpart.control(minsplit = 20, minbucket = round(minsplit/3), cp = 0.01, maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10, surrogatestyle = 0, maxdepth = 30, . . . )*

**cost**
*A vector of non-negative costs, one for each variable in the model. Defaults to one for all variables. These are scalings to be applied when considering splits, so the improvement on splitting on a variable is divided by its cost in deciding which split to choose.*

```r
random_tree_modified=rpart(train_set,cost = c(3,1,1,2,1,1,1),control = rpart.control(minsplit = 10, minb
              maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
              surrogatestyle = 0, maxdepth = 10))
prp(random_tree_modified,extra=2)
```

**Question 5: Try different combinations of values for some of the parameters (decreasing minsplit, minbucket, cp and cost values, for example) and check the performance of each combination on the testing subset. How does this performance change? How do the obtained trees change? Is there any relationship between the parameters values and the shape of the trees?**



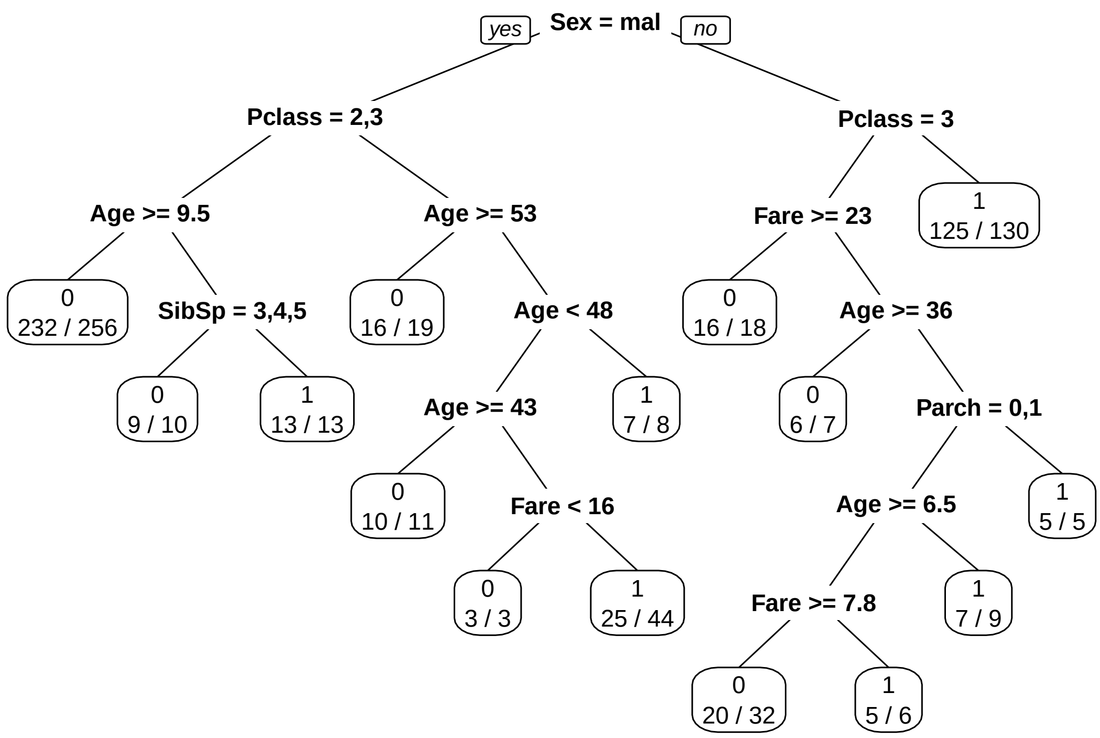

```r
testPredMod <- predict(random_tree_modified, test_set,type="class")
confusionMatrix(testPredMod,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 69 21
##          1 15 36
##
##                Accuracy : 0.7447
##                  95% CI : (0.6645, 0.8143)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 0.0001522
##
##                   Kappa : 0.4608
##  Mcnemar's Test P-Value : 0.4046568
##
##             Sensitivity : 0.8214
##             Specificity : 0.6316
##          Pos Pred Value : 0.7667
##          Neg Pred Value : 0.7059
##              Prevalence : 0.5957
##          Detection Rate : 0.4894
##    Detection Prevalence : 0.6383
```

```
##        Balanced Accuracy : 0.7265
##
##         'Positive' Class : 0
##
```

The accuracy obtained is similar than the obtained with the default values. We picked a small value for cp, so the tree has a lot of branches and it is very deep.

```r
random_tree_modified=rpart(train_set,cost = c(1,1,1,1,1,1,1),control = rpart.control(minsplit = 10, minb
              maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
              surrogatestyle = 0, maxdepth = 10))
prp(random_tree_modified,extra=2)
```



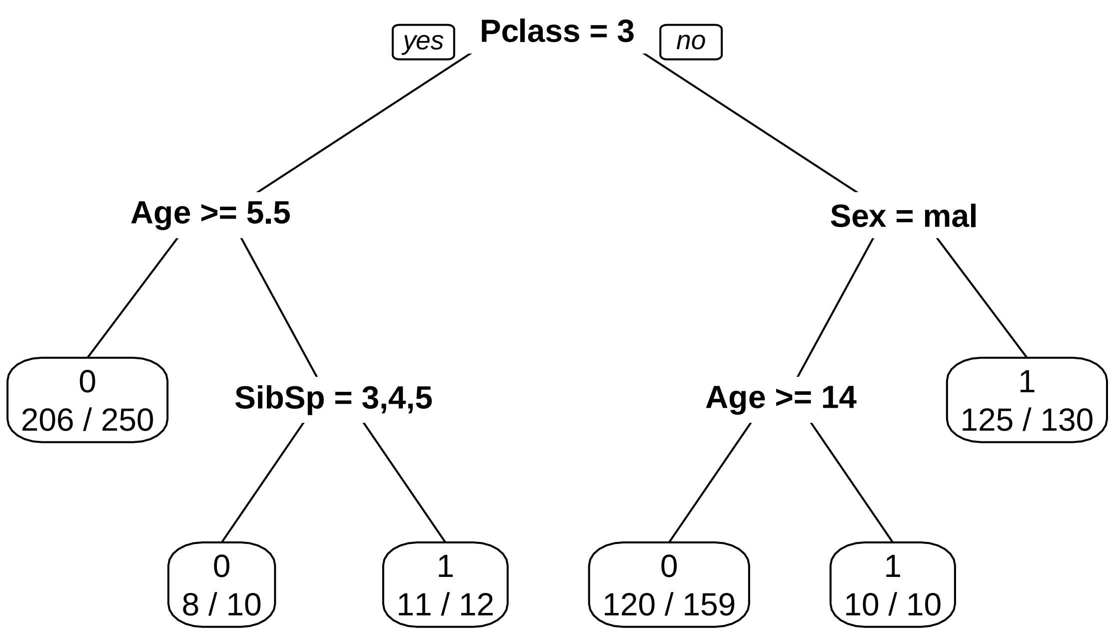

```r
testPredMod <- predict(random_tree_modified, test_set,type="class")
confusionMatrix(testPredMod,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 71 19
##          1 13 38
##
##                Accuracy : 0.773
##                  95% CI : (0.695, 0.8393)
```

```
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 6.723e-06
##
##                   Kappa : 0.5207
##  Mcnemar's Test P-Value : 0.3768
##
##             Sensitivity : 0.8452
##             Specificity : 0.6667
##          Pos Pred Value : 0.7889
##          Neg Pred Value : 0.7451
##              Prevalence : 0.5957
##          Detection Rate : 0.5035
##    Detection Prevalence : 0.6383
##       Balanced Accuracy : 0.7560
##
##        'Positive' Class : 0
##
```

In this other case the cp value is not so low, so the overfitting is reduced. That's why we have a better accuracy (around 77%), which is higher than the one obtained with the default parameters.

```
random_tree_modified=rpart(train_set,cost = c(1,3,1,1,1,1,1),control = rpart.control(minsplit = 10, minb
              maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
              surrogatestyle = 0, maxdepth = 10))
prp(random_tree_modified,extra=2)
```

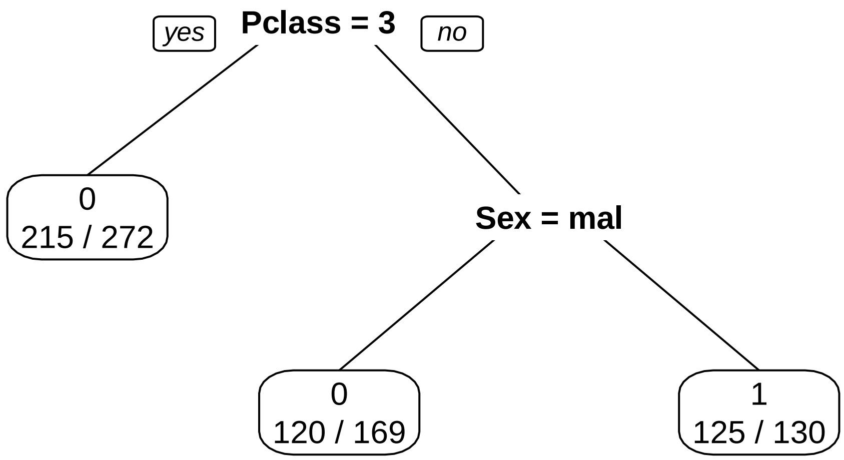


```r
testPredMod <- predict(random_tree_modified, test_set,type="class")
confusionMatrix(testPredMod,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 80 29
##          1  4 28
##
##                Accuracy : 0.766
##                  95% CI : (0.6873, 0.8331)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 1.551e-05
##
##                   Kappa : 0.4772
##  Mcnemar's Test P-Value : 2.943e-05
##
##             Sensitivity : 0.9524
##             Specificity : 0.4912
##          Pos Pred Value : 0.7339
##          Neg Pred Value : 0.8750
##              Prevalence : 0.5957
##          Detection Rate : 0.5674
##    Detection Prevalence : 0.7730
##       Balanced Accuracy : 0.7218
##
##        'Positive' Class : 0
##
```

Here the accuracy is a little bit worse. This might be because we've forced the Pclass to be more important than the sex assigning to it a higher cost. We also can see a big difference between the sensitivity and the specificity. That's because deaths are predicted worse than the survivals.

```r
random_tree_modified=rpart(train_set,cost = c(1,3,1,1,1,1,1),control = rpart.control(minsplit = 40, minb
              maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
              surrogatestyle = 0, maxdepth = 10))
prp(random_tree_modified,extra=2)
```




```
testPredMod <- predict(random_tree_modified, test_set,type="class")
confusionMatrix(testPredMod,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  0  1
##          0 80 34
##          1  4 23
##
##                Accuracy : 0.7305
##                  95% CI : (0.6493, 0.8017)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 0.000584
##
##                   Kappa : 0.3888
##  Mcnemar's Test P-Value : 2.546e-06
##
##             Sensitivity : 0.9524
##             Specificity : 0.4035
##          Pos Pred Value : 0.7018
##          Neg Pred Value : 0.8519
##              Prevalence : 0.5957
##          Detection Rate : 0.5674
##    Detection Prevalence : 0.8085
```

```
##        Balanced Accuracy : 0.6779
##
##         'Positive' Class : 0
##
```

In this last example we have a high value of cp, so our tree is underfitted. That's why the accuracy value is worse.

Summarizing, when we have a small cp the trees are deeper. The trees also change depending on the costs assigned, so if we assign a high cost value to a variable, the probability of splitting the tree according to that variable would be higher.

**Question 6: Which are the combinations of parameters values tested by the train() function? Are there any changes in the performance of the algorithm when different combinations of values are used (according to the results of the cross validation)?** We change the tuneLength to have more parameters tested by default.

```
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 1)
treeFit <- train(train_set[,2:7],train_set[,1],
                 method = "rpartCost",
                 trControl = fitControl,tuneLength = 8)
treeFit
```

```
## Cost-Sensitive CART
##
## 571 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 514, 513, 514, 514, 514, 514, ...
## Resampling results across tuning parameters:
##
##   Cost  cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   1     0.00000000  0.7931942  0.5678883  0.049218917  0.10406017
##   1     0.06617192  0.7722928  0.5139953  0.051797259  0.10405088
##   1     0.13234385  0.7828191  0.5419774  0.062604256  0.13021897
##   1     0.19851577  0.7828191  0.5419774  0.062604256  0.13021897
##   1     0.26468769  0.7828191  0.5419774  0.062604256  0.13021897
##   1     0.33085962  0.7828191  0.5419774  0.062604256  0.13021897
##   1     0.39703154  0.7828191  0.5419774  0.062604256  0.13021897
##   2     0.00000000  0.8019056  0.5721818  0.066182059  0.14518498
##   2     0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   2     0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   2     0.19851577  0.8055051  0.5648747  0.035941035  0.08519026
##   2     0.26468769  0.6305505  0.1210261  0.058084662  0.19578535
##   2     0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   2     0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
```

```
##   3      0.00000000  0.8106776  0.5845858  0.045037441  0.10349900
##   3      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   3      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   3      0.19851577  0.8055051  0.5648747  0.035941035  0.08519026
##   3      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   3      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   3      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##   4      0.00000000  0.8124622  0.5849283  0.035583772  0.08304251
##   4      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   4      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   4      0.19851577  0.8055051  0.5648747  0.035941035  0.08519026
##   4      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   4      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   4      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##   5      0.00000000  0.8159407  0.5931636  0.039210661  0.09075601
##   5      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   5      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   5      0.19851577  0.8055051  0.5648747  0.035941035  0.08519026
##   5      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   5      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   5      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##   6      0.00000000  0.8037810  0.5619537  0.035227988  0.08511210
##   6      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   6      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   6      0.19851577  0.7077435  0.3081931  0.099966261  0.26942824
##   6      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   6      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   6      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##   7      0.00000000  0.7827284  0.5098137  0.033902820  0.08267277
##   7      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   7      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   7      0.19851577  0.6673926  0.1994096  0.095380130  0.26095745
##   7      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   7      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   7      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##   8      0.00000000  0.7844828  0.5124330  0.037388036  0.09146136
##   8      0.06617192  0.8055051  0.5648747  0.035941035  0.08519026
##   8      0.13234385  0.8055051  0.5648747  0.035941035  0.08519026
##   8      0.19851577  0.6270417  0.0892509  0.067699908  0.18950992
##   8      0.26468769  0.5954628  0.0000000  0.003252191  0.00000000
##   8      0.33085962  0.5954628  0.0000000  0.003252191  0.00000000
##   8      0.39703154  0.5954628  0.0000000  0.003252191  0.00000000
##
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were cp = 0 and Cost = 5.
```

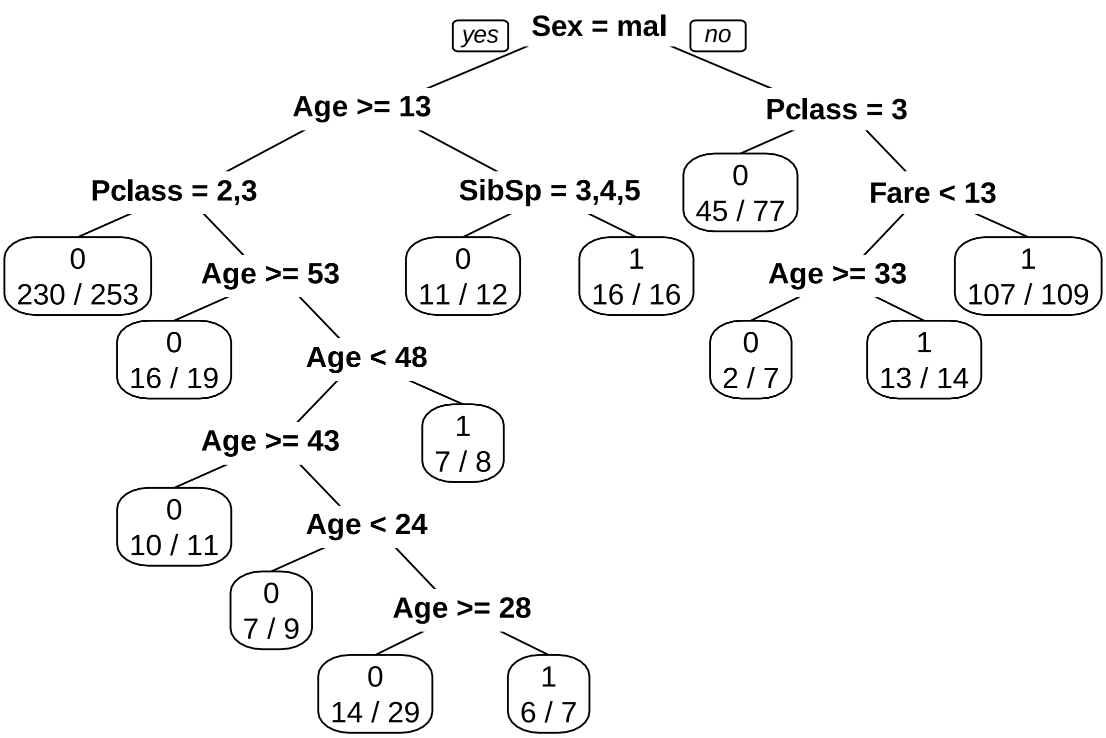
When different combinations are used the performance changes, as we can see above. The combination with better performance is the one picked for the final model.

```
treeFit$bestTune
```

**Question 7: Which is the final combination of parameters values used? Which is the shape of the tree trained with this automatic tuning function? (CLUE: if the result of calling to the train() function is stored in an object called rpartFit, check the rpartFit$finalModel object).**

```
##    cp Cost
## 29  0    5
```

```
prp(treeFit$finalModel,extra=2)
```




```
testPredTreeFit <- predict(treeFit$finalModel, test_set,type="class")
confusionMatrix(testPredTreeFit,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 75 32
##          1  9 25
##
##                Accuracy : 0.7092
##                  95% CI : (0.6268, 0.7826)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 0.0034120
##
##                   Kappa : 0.3544
```

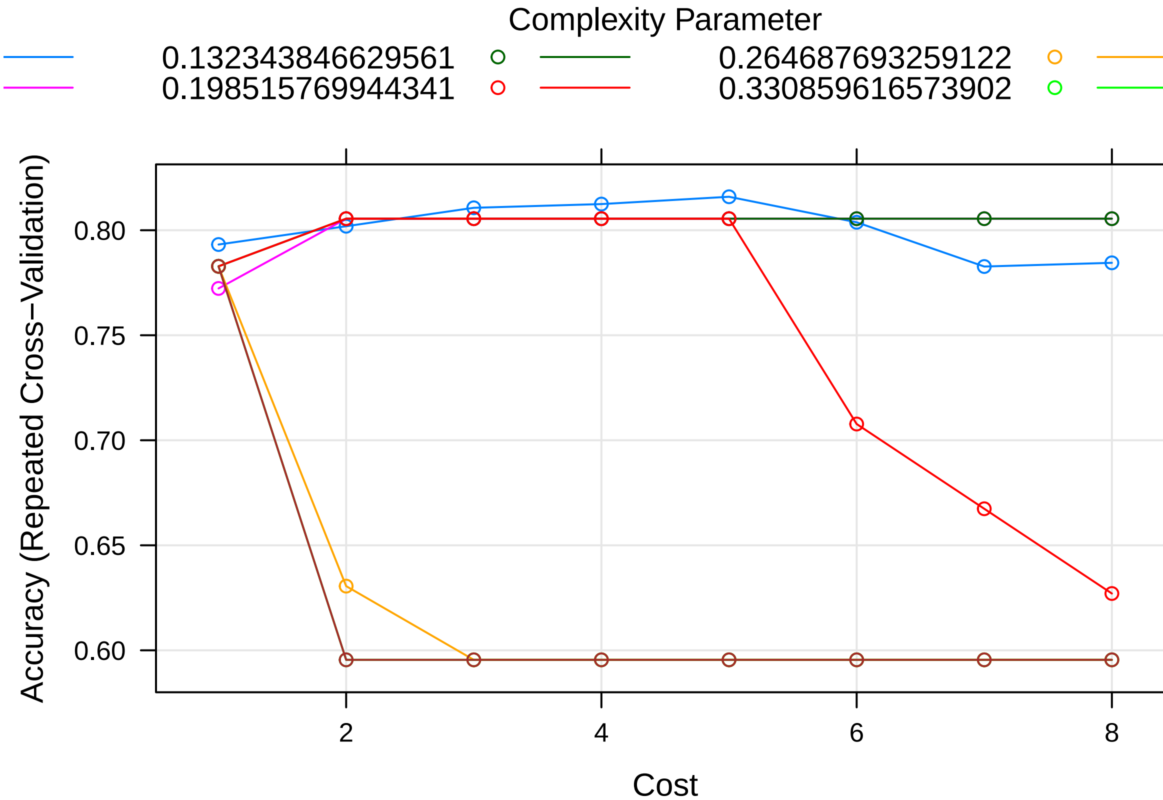
```
##   Mcnemar's Test P-Value : 0.0005908
##
##              Sensitivity : 0.8929
##              Specificity : 0.4386
##           Pos Pred Value : 0.7009
##           Neg Pred Value : 0.7353
##               Prevalence : 0.5957
##           Detection Rate : 0.5319
##     Detection Prevalence : 0.7589
##        Balanced Accuracy : 0.6657
##
##         'Positive' Class : 0
##
```

The final values used for the model were cp = 0 and Cost = 5 because the accuracy there is the highest.

We checked the accuracy in our testing data set and it is around 70%. The main variables used are sex, age and Pclass. We can see the shape of the tree obtained in the image above.

```r
plot(treeFit)
```

**Question 8: Plot the result of calling the train() function with the plot() function. What does**




**this plot represent?**

We can see here the variation of the accuracy for the different combinations used in the train function. The one picked is the one with better accuracy and less complexity (in case several combinations have the same accuracy values).

**Question 9: Answer Question 6 again but now with the results of this new run. (tuning the paramenters)**

**Question 10: Analogously to Question 9, respond to Question 7 with the results of this new run.**

```
treeGrid <- expand.grid(Cost=c(1,   2,  3,  5,  10),
cp  =   c(0,    0.01,   0.02,   0.04,   0.07,   0.10))
T1<-Sys.time()
treeFit2 <- train(train_set[,2:7],train_set[,1],
                 method = "rpartCost",
                 trControl = fitControl,
                 tuneGrid = treeGrid)
T2<-Sys.time()
timeDecisionTree=difftime(T2,T1)
treeFit2
```

**Question 11: Answer Question 8 again but now with the results of this new run.**

```
## Cost-Sensitive CART
##
## 571 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 513, 514, 514, 514, 514, 514, ...
## Resampling results across tuning parameters:
##
##   Cost  cp    Accuracy   Kappa      Accuracy SD  Kappa SD
##    1    0.00  0.8054749  0.5953473  0.05857563   0.12065422
##    1    0.01  0.7897762  0.5556960  0.04704474   0.10162248
##    1    0.02  0.8038113  0.5679461  0.02984207   0.06750979
##    1    0.04  0.7932849  0.5440617  0.04227041   0.08344708
##    1    0.07  0.7828191  0.5423179  0.04982924   0.10015239
##    2    0.00  0.7967937  0.5667897  0.04444794   0.08636616
##    2    0.01  0.8090744  0.5796051  0.02818998   0.06576674
##    2    0.02  0.8143376  0.5871147  0.03340694   0.07754291
##    2    0.04  0.8055656  0.5655358  0.02823179   0.06616199
##    2    0.07  0.8055656  0.5655358  0.02823179   0.06616199
##    3    0.00  0.8125832  0.5918952  0.04637065   0.10022165
##    3    0.01  0.8143376  0.5876819  0.03340694   0.07778032
##    3    0.02  0.8160920  0.5907112  0.03344046   0.07759869
##    3    0.04  0.8055656  0.5655358  0.02823179   0.06616199
##    3    0.07  0.8055656  0.5655358  0.02823179   0.06616199
##    5    0.00  0.8143376  0.5889401  0.03969463   0.09088321
```

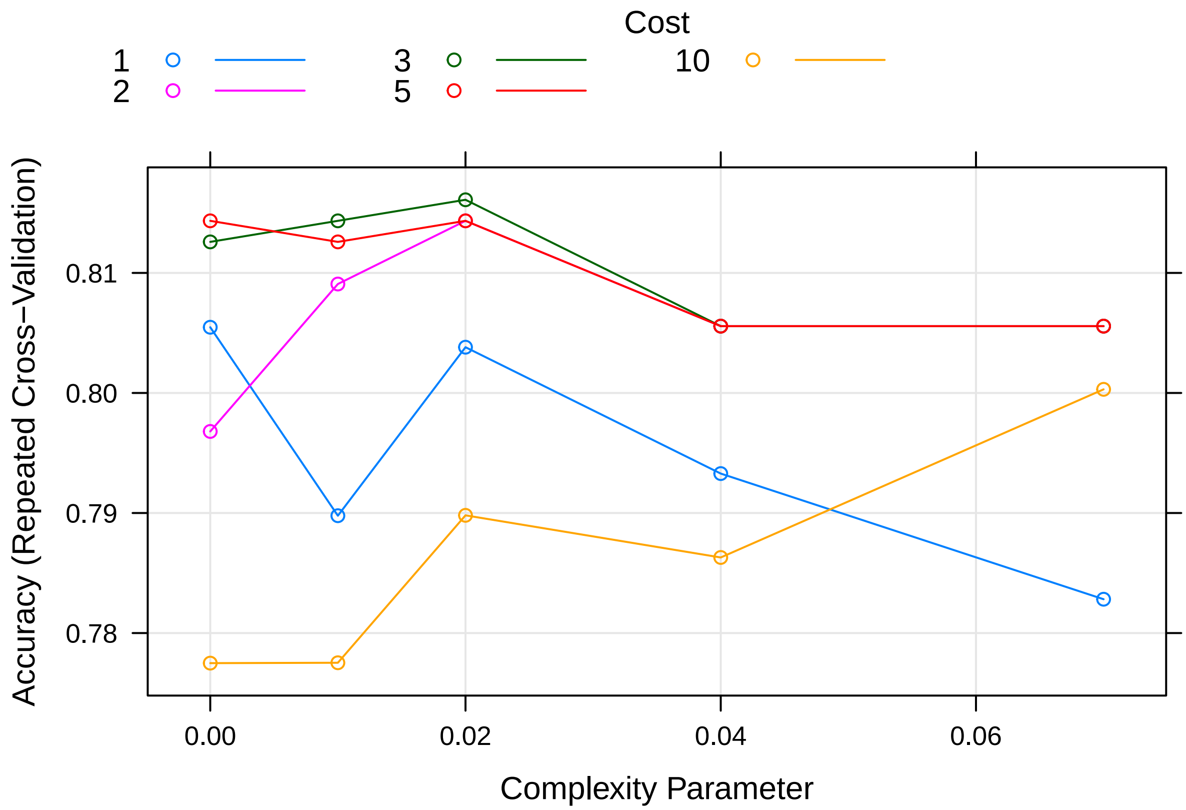
```
##    5    0.01  0.8125832  0.5820159  0.03622340   0.08523496
##    5    0.02  0.8143376  0.5863302  0.03539517   0.08294011
##    5    0.04  0.8055656  0.5655358  0.02823179   0.06616199
##    5    0.07  0.8055656  0.5655358  0.02823179   0.06616199
##   10    0.00  0.7774955  0.4952884  0.04865154   0.11877539
##   10    0.01  0.7775257  0.4962626  0.03734329   0.09040739
##   10    0.02  0.7898064  0.5263073  0.02768478   0.06659218
##   10    0.04  0.7862976  0.5184680  0.02487103   0.06028931
##   10    0.07  0.8003025  0.5525916  0.02918680   0.06951318
## 
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were cp = 0.02 and Cost = 3.
```

```r
testPredTreeFit2 <- predict(treeFit2$finalModel, test_set,type="class")
confusionMatrix(testPredTreeFit2,test_set$Survived)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  0  1
##          0 79 30
##          1  5 27
##                                           
##                Accuracy : 0.7518          
##                  95% CI : (0.6721, 0.8206)
##     No Information Rate : 0.5957          
##     P-Value [Acc > NIR] : 7.372e-05       
##                                           
##                   Kappa : 0.4456          
##  Mcnemar's Test P-Value : 4.976e-05       
##                                           
##             Sensitivity : 0.9405          
##             Specificity : 0.4737          
##          Pos Pred Value : 0.7248          
##          Neg Pred Value : 0.8438          
##              Prevalence : 0.5957          
##          Detection Rate : 0.5603          
##    Detection Prevalence : 0.7730          
##       Balanced Accuracy : 0.7071          
##                                           
##        'Positive' Class : 0               
## 
```

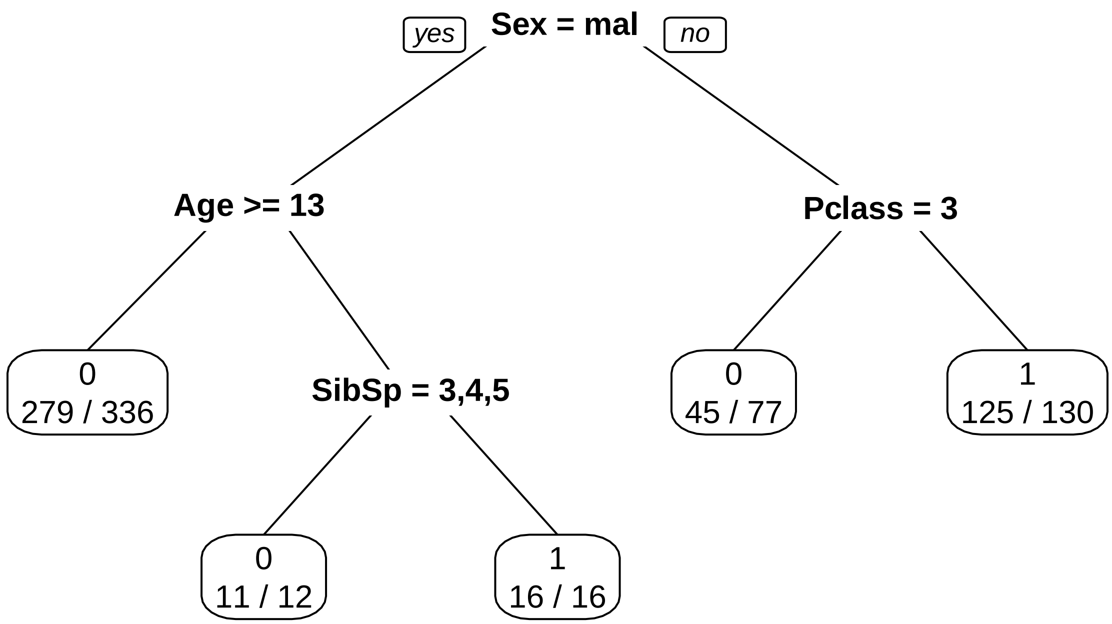
The accuracy of the best combination is around 81%. We can also check the accuracy value in our testing set, which is better than the one from the previous question.

```r
plot(treeFit2)
```




We can see the combination picked above and the plot of the different combination, where we can see which one was the best.

```r
prp(treeFit2$finalModel,extra=2)
```




The shape of the tree is plotted above. The tree obtained is simple and we can see that the rule of being a woman and not being in third class is still present. We can also see that male children with 3,4 or 5 siblings died, but the ones with less siblings survived.

## Training a random forest

**Question 12: Answer Question 6 again but now with the results of this new run.**

**Question 13: Which is the final combination of the parameters used?**

**Question 14: Answer Question 8 again but now with the results of this new run.** We load the random forest library.

```
library(randomForest)
```

```
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 1)
T1<-Sys.time()
treeFitRandom <- train(train_set[,2:7],train_set[,1],
                 method = "rf",
                 trControl = fitControl,
```

```r
                 ntree=2000,
                 tuneLength = 5)
T2<-Sys.time()
timeRandomForest=difftime(T2,T1)

treeFitRandom
```

```
## Random Forest
##
## 571 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 514, 514, 514, 514, 514, 513, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##   2     0.8265275  0.6293173  0.05718243   0.12101680
##   3     0.8265578  0.6327745  0.04194302   0.08688276
##   4     0.8265880  0.6344535  0.03266540   0.06631840
##   5     0.8178463  0.6173292  0.03815299   0.07832908
##   6     0.8108590  0.6039120  0.03859495   0.07861150
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 4.
```

Above we can see the value for mtry picked in the best combination (the one with highest accuracy).

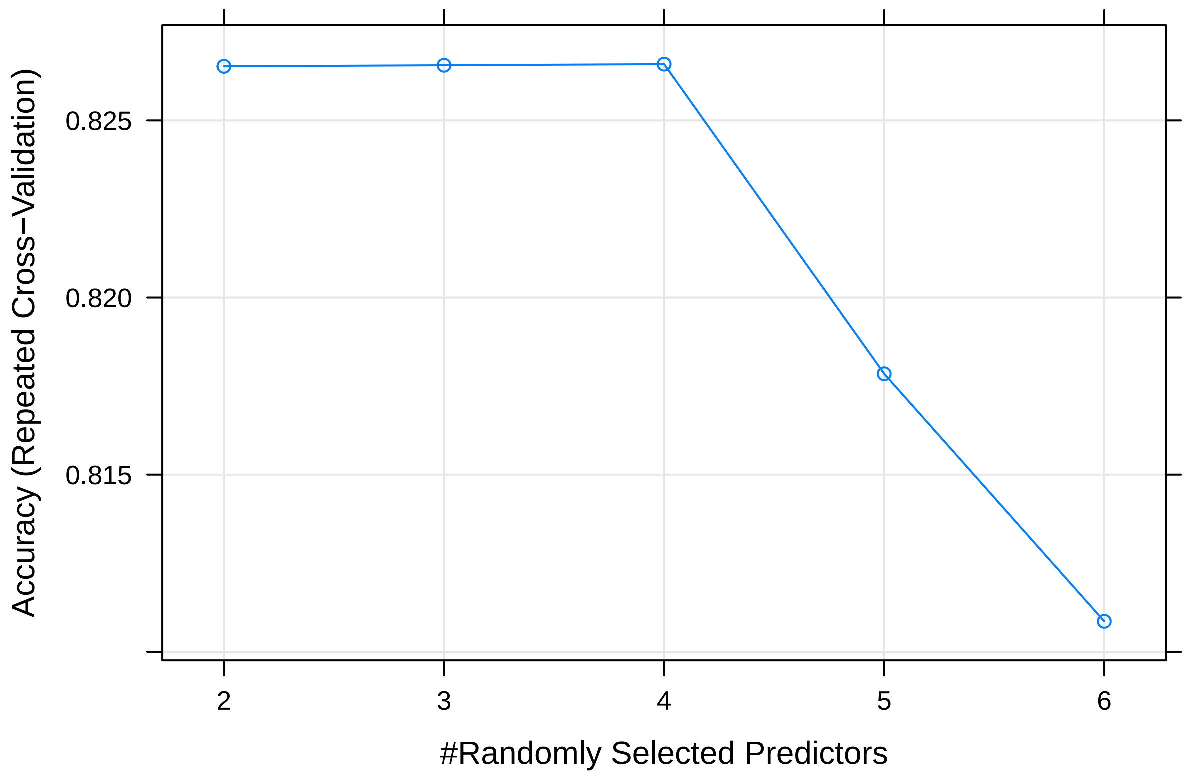
```r
testPredTreeFitRandom <- predict(treeFitRandom$finalModel, test_set,type="class")
confusionMatrix(testPredTreeFitRandom,test_set$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 67 18
##          1 17 39
##
##                Accuracy : 0.7518
##                  95% CI : (0.6721, 0.8206)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 7.372e-05
##
##                   Kappa : 0.4832
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7976
##             Specificity : 0.6842
##          Pos Pred Value : 0.7882
##          Neg Pred Value : 0.6964
##              Prevalence : 0.5957
```

```
##          Detection Rate : 0.4752
##    Detection Prevalence : 0.6028
##       Balanced Accuracy : 0.7409
##
##        'Positive' Class : 0
##
```

The accuracy is around 75%.

```r
plot(treeFitRandom)
```




Here we can see the combinations plotted and the best one is clear in the plot.

```r
treeRandomForest=getTree(treeFitRandom$finalModel,1,labelVar = TRUE)
treeRandomForest
```

```
##      left daughter right daughter split var split point status prediction
## 1                2              3    Pclass     3.00000      1       <NA>
## 2                4              5      Fare    16.89790      1       <NA>
## 3                6              7       Age     1.50000      1       <NA>
## 4                8              9     Parch     1.00000      1       <NA>
## 5               10             11       Sex     1.00000      1       <NA>
## 6                0              0      <NA>     0.00000     -1          1
## 7               12             13       Sex     1.00000      1       <NA>
## 8               14             15      Fare    13.75000      1       <NA>
```

```
## 9                 0              0      <NA>    0.00000      -1           1
## 10               16             17       Age    2.50000       1        <NA>
## 11               18             19       Age   18.00000       1        <NA>
## 12               20             21      Fare   20.80000       1        <NA>
## 13               22             23       Age   32.50000       1        <NA>
## 14               24             25       Sex    1.00000       1        <NA>
## 15                0              0      <NA>    0.00000      -1           0
## 16                0              0      <NA>    0.00000      -1           0
## 17               26             27      Fare   26.12500       1        <NA>
## 18                0              0      <NA>    0.00000      -1           1
## 19               28             29    Pclass    1.00000       1        <NA>
## 20               30             31      Fare   16.40000       1        <NA>
## 21               32             33     Parch   32.00000       1        <NA>
## 22               34             35       Age   30.50000       1        <NA>
## 23                0              0      <NA>    0.00000      -1           0
## 24               36             37       Age   36.00000       1        <NA>
## 25               38             39      Fare   12.93750       1        <NA>
## 26               40             41       Age   43.00000       1        <NA>
## 27                0              0      <NA>    0.00000      -1           1
## 28               42             43      Fare  127.81665       1        <NA>
## 29               44             45     SibSp    2.00000       1        <NA>
## 30               46             47      Fare   13.43750       1        <NA>
## 31               48             49     SibSp    3.00000       1        <NA>
## 32               50             51      Fare   30.25625       1        <NA>
## 33                0              0      <NA>    0.00000      -1           0
## 34               52             53       Age   24.50000       1        <NA>
## 35               54             55      Fare    7.83750       1        <NA>
## 36                0              0      <NA>    0.00000      -1           1
## 37               56             57       Age   41.50000       1        <NA>
## 38               58             59       Age   21.00000       1        <NA>
## 39               60             61      Fare   13.25000       1        <NA>
## 40                0              0      <NA>    0.00000      -1           1
## 41               62             63     Parch    1.00000       1        <NA>
## 42               64             65       Age   39.00000       1        <NA>
## 43                0              0      <NA>    0.00000      -1           0
## 44               66             67      Fare   26.12500       1        <NA>
## 45                0              0      <NA>    0.00000      -1           0
## 46               68             69     Parch    1.00000       1        <NA>
## 47               70             71      Fare   15.67500       1        <NA>
## 48                0              0      <NA>    0.00000      -1           1
## 49                0              0      <NA>    0.00000      -1           0
## 50                0              0      <NA>    0.00000      -1           0
## 51                0              0      <NA>    0.00000      -1           1
## 52               72             73       Age    9.50000       1        <NA>
## 53               74             75       Age   27.50000       1        <NA>
## 54                0              0      <NA>    0.00000      -1           0
## 55               76             77      Fare    8.20625       1        <NA>
## 56                0              0      <NA>    0.00000      -1           0
## 57                0              0      <NA>    0.00000      -1           1
## 58               78             79       Age   18.50000       1        <NA>
## 59                0              0      <NA>    0.00000      -1           0
## 60               80             81       Age   30.50000       1        <NA>
## 61                0              0      <NA>    0.00000      -1           0
## 62                0              0      <NA>    0.00000      -1           0
```

```
## 63          0          0  <NA>   0.00000   -1       1
## 64         82         83  Fare  64.97915    1    <NA>
## 65         84         85  Fare  73.86460    1    <NA>
## 66         86         87   Age  33.00000    1    <NA>
## 67          0          0  <NA>   0.00000   -1       0
## 68         88         89  Fare  10.87920    1    <NA>
## 69          0          0  <NA>   0.00000   -1       1
## 70          0          0  <NA>   0.00000   -1       0
## 71         90         91  Fare  15.97500    1    <NA>
## 72         92         93  Fare  24.82500    1    <NA>
## 73         94         95  Fare   7.22710    1    <NA>
## 74         96         97   Age  26.50000    1    <NA>
## 75         98         99  Fare   8.77500    1    <NA>
## 76          0          0  <NA>   0.00000   -1       1
## 77          0          0  <NA>   0.00000   -1       0
## 78          0          0  <NA>   0.00000   -1       0
## 79          0          0  <NA>   0.00000   -1       1
## 80          0          0  <NA>   0.00000   -1       0
## 81        100        101   Age  41.00000    1    <NA>
## 82        102        103  Fare  27.06875    1    <NA>
## 83        104        105  Fare  86.08540    1    <NA>
## 84        106        107   Age  75.50000    1    <NA>
## 85        108        109  Fare  81.33750    1    <NA>
## 86          0          0  <NA>   0.00000   -1       1
## 87          0          0  <NA>   0.00000   -1       0
## 88        110        111  Fare   7.76250    1    <NA>
## 89          0          0  <NA>   0.00000   -1       1
## 90          0          0  <NA>   0.00000   -1       1
## 91          0          0  <NA>   0.00000   -1       0
## 92          0          0  <NA>   0.00000   -1       1
## 93        112        113  Fare  30.25625    1    <NA>
## 94        114        115   Age  20.50000    1    <NA>
## 95        116        117   Age  19.50000    1    <NA>
## 96        118        119  Fare   7.83540    1    <NA>
## 97          0          0  <NA>   0.00000   -1       1
## 98          0          0  <NA>   0.00000   -1       0
## 99        120        121   Age  28.75000    1    <NA>
## 100       122        123   Age  32.50000    1    <NA>
## 101         0          0  <NA>   0.00000   -1       0
## 102         0          0  <NA>   0.00000   -1       1
## 103       124        125  Fare  28.87500    1    <NA>
## 104       126        127   Age  27.50000    1    <NA>
## 105         0          0  <NA>   0.00000   -1       1
## 106       128        129   Age  57.00000    1    <NA>
## 107         0          0  <NA>   0.00000   -1       1
## 108       130        131 SibSp   1.00000    1    <NA>
## 109       132        133  Fare 101.18960    1    <NA>
## 110       134        135  Fare   6.98750    1    <NA>
## 111       136        137   Age  50.00000    1    <NA>
## 112         0          0  <NA>   0.00000   -1       0
## 113       138        139   Age   2.50000    1    <NA>
## 114         0          0  <NA>   0.00000   -1       0
## 115       140        141  Fare   7.13335    1    <NA>
## 116       142        143 Parch   1.00000    1    <NA>
```

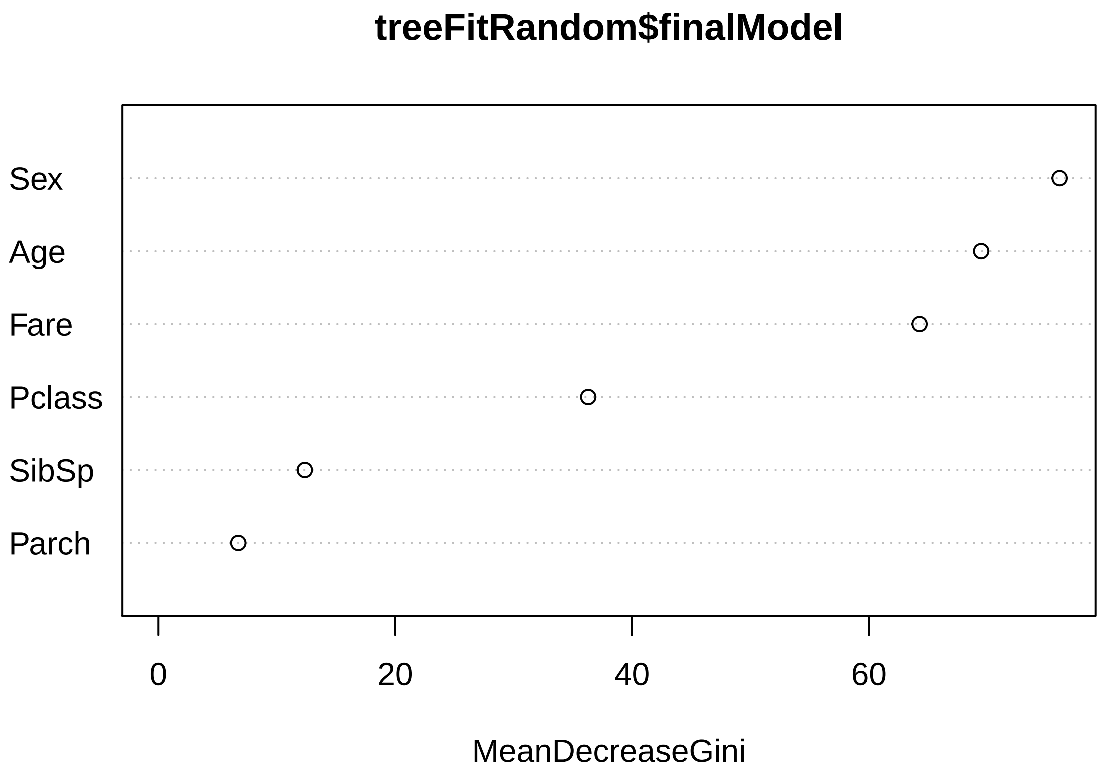
```
## 117            0             0    <NA>  0.00000    -1        0
## 118          144           145    Fare  7.75835     1     <NA>
## 119          146           147   SibSp  1.00000     1     <NA>
## 120            0             0    <NA>  0.00000    -1        0
## 121            0             0    <NA>  0.00000    -1        1
## 122            0             0    <NA>  0.00000    -1        1
## 123            0             0    <NA>  0.00000    -1        1
## 124            0             0    <NA>  0.00000    -1        0
## 125          148           149     Age 34.00000     1     <NA>
## 126          150           151    Fare 77.00835     1     <NA>
## 127            0             0    <NA>  0.00000    -1        0
## 128          152           153    Fare 52.27710     1     <NA>
## 129            0             0    <NA>  0.00000    -1        0
## 130            0             0    <NA>  0.00000    -1        0
## 131            0             0    <NA>  0.00000    -1        1
## 132          154           155     Age 47.00000     1     <NA>
## 133            0             0    <NA>  0.00000    -1        0
## 134            0             0    <NA>  0.00000    -1        0
## 135            0             0    <NA>  0.00000    -1        1
## 136          156           157     Age 16.00000     1     <NA>
## 137            0             0    <NA>  0.00000    -1        1
## 138            0             0    <NA>  0.00000    -1        0
## 139          158           159     Age  6.00000     1     <NA>
## 140            0             0    <NA>  0.00000    -1        0
## 141            0             0    <NA>  0.00000    -1        1
## 142          160           161     Age 18.50000     1     <NA>
## 143            0             0    <NA>  0.00000    -1        0
## 144            0             0    <NA>  0.00000    -1        0
## 145            0             0    <NA>  0.00000    -1        1
## 146          162           163     Age 25.50000     1     <NA>
## 147            0             0    <NA>  0.00000    -1        0
## 148            0             0    <NA>  0.00000    -1        1
## 149          164           165    Fare 52.82710     1     <NA>
## 150            0             0    <NA>  0.00000    -1        1
## 151            0             0    <NA>  0.00000    -1        0
## 152          166           167    Fare 43.68125     1     <NA>
## 153          168           169    Fare 59.05210     1     <NA>
## 154            0             0    <NA>  0.00000    -1        0
## 155            0             0    <NA>  0.00000    -1        1
## 156            0             0    <NA>  0.00000    -1        0
## 157          170           171     Age 20.00000     1     <NA>
## 158            0             0    <NA>  0.00000    -1        1
## 159            0             0    <NA>  0.00000    -1        0
## 160            0             0    <NA>  0.00000    -1        0
## 161          172           173    Fare  8.10415     1     <NA>
## 162            0             0    <NA>  0.00000    -1        0
## 163          174           175    Fare 13.34165     1     <NA>
## 164            0             0    <NA>  0.00000    -1        1
## 165            0             0    <NA>  0.00000    -1        0
## 166          176           177     Age 49.00000     1     <NA>
## 167            0             0    <NA>  0.00000    -1        0
## 168            0             0    <NA>  0.00000    -1        1
## 169            0             0    <NA>  0.00000    -1        0
## 170          178           179    Fare  7.81460     1     <NA>
```

```
## 171          180          181     Fare     8.29375     1       <NA>
## 172          182          183     Fare     7.85000     1       <NA>
## 173            0            0     <NA>     0.00000    -1          0
## 174            0            0     <NA>     0.00000    -1          0
## 175            0            0     <NA>     0.00000    -1          1
## 176          184          185      Age    46.00000     1       <NA>
## 177          186          187      Age    53.50000     1       <NA>
## 178            0            0     <NA>     0.00000    -1          0
## 179            0            0     <NA>     0.00000    -1          1
## 180            0            0     <NA>     0.00000    -1          1
## 181            0            0     <NA>     0.00000    -1          0
## 182            0            0     <NA>     0.00000    -1          0
## 183            0            0     <NA>     0.00000    -1          1
## 184          188          189     Fare    29.36040     1       <NA>
## 185            0            0     <NA>     0.00000    -1          0
## 186            0            0     <NA>     0.00000    -1          1
## 187          190          191     Fare    33.09790     1       <NA>
## 188          192          193      Age    42.50000     1       <NA>
## 189          194          195      Age    42.50000     1       <NA>
## 190            0            0     <NA>     0.00000    -1          0
## 191            0            0     <NA>     0.00000    -1          1
## 192            0            0     <NA>     0.00000    -1          0
## 193            0            0     <NA>     0.00000    -1          0
## 194            0            0     <NA>     0.00000    -1          1
## 195            0            0     <NA>     0.00000    -1          0
```

Random forests are a black box, so we can't visualize the output. If we want to have an idea of how the final tree would look like we can use the getTree function and get a representation of a sample tree from the random forest (as dine above).

```r
varImpPlot(treeFitRandom$finalModel)
```

**Question 15: Plot the importance of each variable for the model with function VarImPlot() from package caret. Which are the most relevant variables according to their mean decrease of the Gini index? Are these variables the ones selected when we built our decision trees?**




Sex is the most important parameter, while Parch and SibSp are the lest important ones. The most important ones were the ones selected in the trees before. There's a big gap between the most important variables and the less important ones and that is linked with the depth where they appear in the trees.

**Question 16: Answer Question 6 again but now with the results of this new run. (tuning the paramenters)**

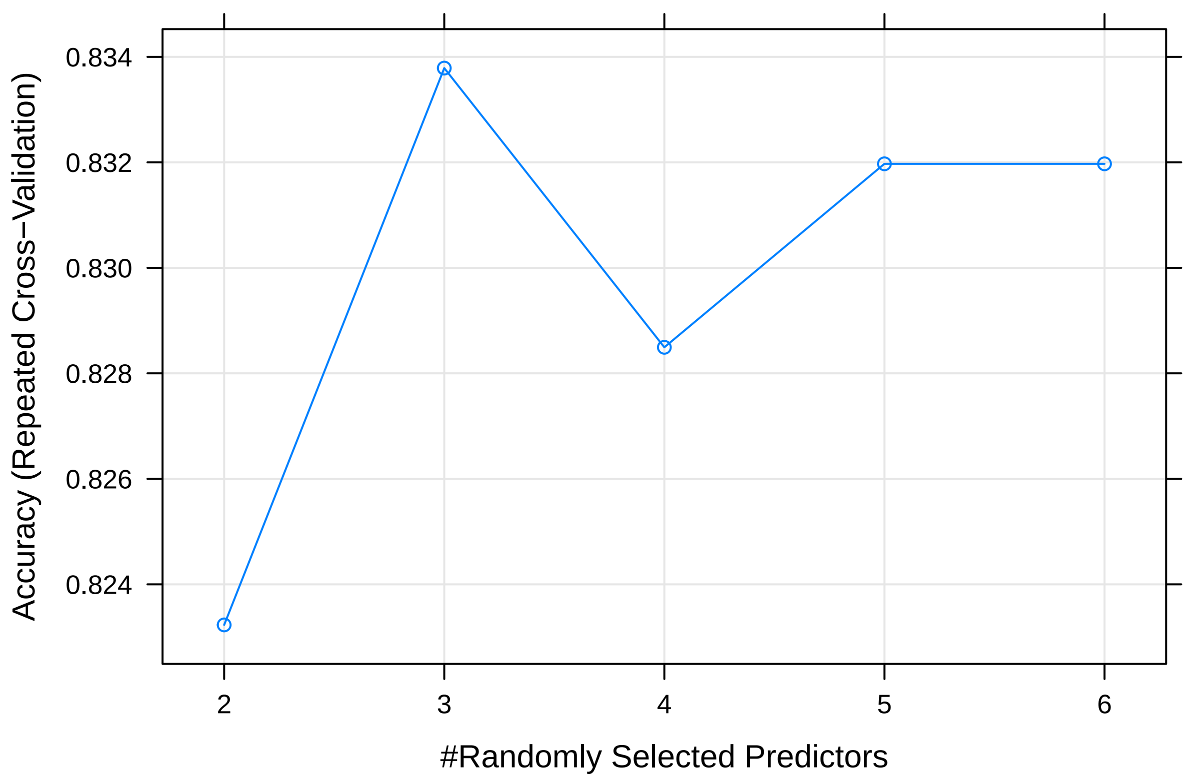
**Question 17: Answer Question 13 again but now with the results of this new run.**

**Question 18: Answer Question 8 again but now with the results of this new run.** The mtry is the number of variables per level, so we are trying to tune it to see when we have the best results.

```
treeGridRF <- expand.grid(mtry=c(2,3,4,5,6))
fitControl <- trainControl(## 10-fold CV
                          method = "repeatedcv",
                          number = 10,
                          ## repeated ten times
                          repeats = 1)
set.seed(100)
treeFitRandom2 <- train(train_set[,2:7],train_set[,1],
                 method = "rf",
                 trControl = fitControl,
                 ntree=2000,
                 tuneGrid = treeGridRF)
treeFitRandom2
```

```
## Random Forest
##
## 571 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 514, 514, 514, 514, 514, 514, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##   2     0.8232305  0.6234820  0.05393756   0.1176481
##   3     0.8337871  0.6482701  0.06290706   0.1354272
##   4     0.8284936  0.6394213  0.06449337   0.1371823
##   5     0.8319722  0.6467077  0.05325205   0.1146701
##   6     0.8319722  0.6460280  0.04638761   0.1012548
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 3.
```

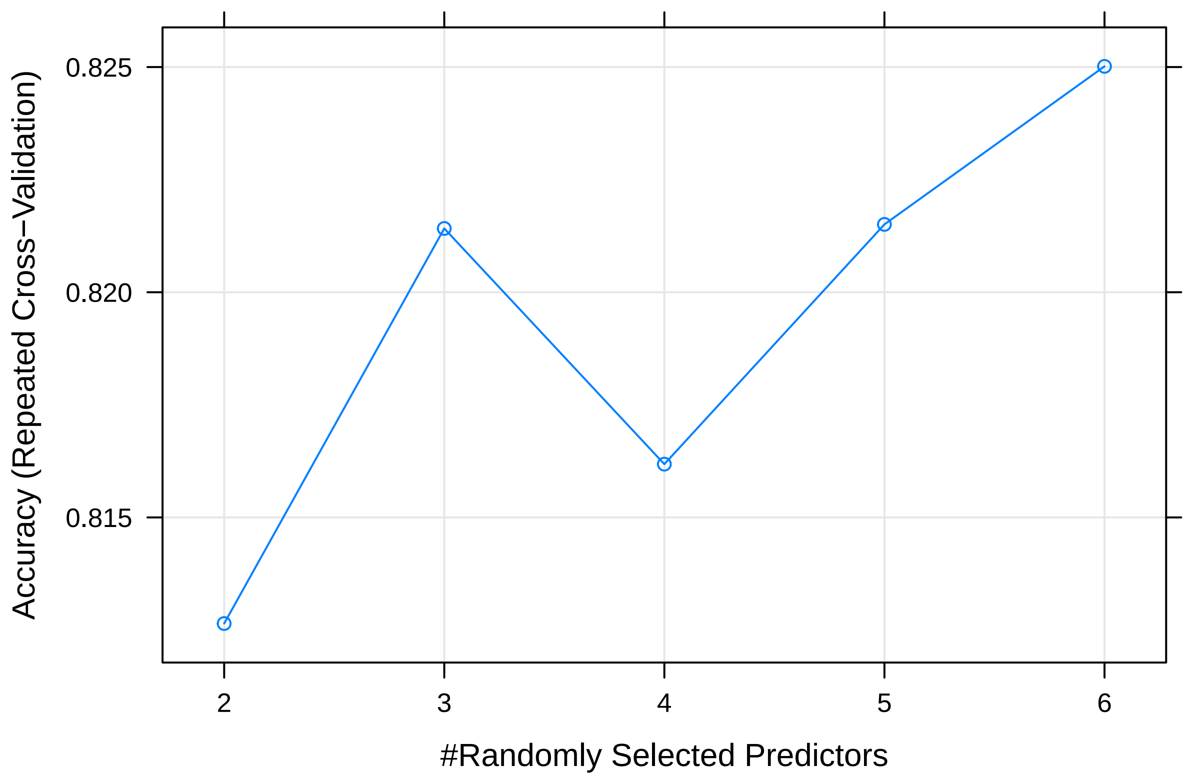
```
plot(treeFitRandom2)
```




```
treeRandomForest2=getTree(treeFitRandom2$finalModel,1,labelVar = TRUE)
```

We've observed that if we train it in a different way we have different results. This is weird, as we think if we set the seed the results should be the same.

```
set.seed(100)
treeFitRandomTEST <- train(as.factor(Survived)~.,data=train_set[,1:7],
                   method = "rf",
                   trControl = fitControl,
                   ntree=2000,
                   tuneGrid = treeGridRF)
plot(treeFitRandomTEST)
```




```
testPredTreeFitRandom2 <- predict(treeFitRandom2$finalModel, test_set,type="class")
confusionMatrix(testPredTreeFitRandom2,test_set$Survived)
```

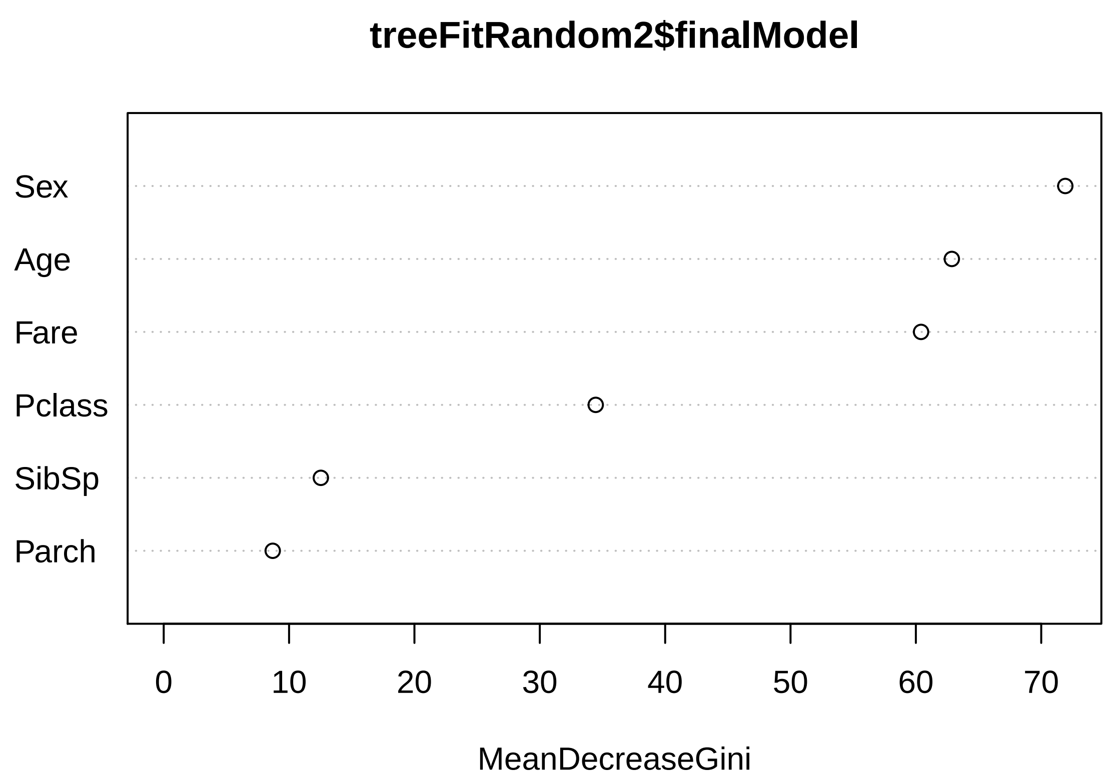
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 70 20
##          1 14 37
##
##                Accuracy : 0.7589
##                  95% CI : (0.6797, 0.8269)
##     No Information Rate : 0.5957
##     P-Value [Acc > NIR] : 3.444e-05
```

```
##
##                  Kappa : 0.4908
##  Mcnemar's Test P-Value : 0.3912
##
##             Sensitivity : 0.8333
##             Specificity : 0.6491
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.7255
##              Prevalence : 0.5957
##          Detection Rate : 0.4965
##    Detection Prevalence : 0.6383
##       Balanced Accuracy : 0.7412
##
##        'Positive' Class : 0
##
```

The accuracy have increased slightly compared with the run we did before.

```r
varImpPlot(treeFitRandom2$finalModel)
```

**Question 19: Answer Question 15 again but now with the results of this new run.**




The importance of the variables is almost the same. Age and fare are closer and sex a little bit further, but in general the importances are the same.

**Question 20: Which is the difference in performance with regards to the testing subset between the best decision tree model and the best random forest model?** With decision trees we got the best accuracy in the second tree we did in question 4 (with an accuracy of 0.773 ). With random forest we achived less accuracy (0.7589). We think this is because the dataset is too simple for using random forest. Even if there was a slight improvement, still decision trees would be better in this case because of the time it takes to run.

The time it tooks in the cases we got the best results with each method are:

```
timeDecisionTree
```

```
## Time difference of 4.105108 secs
```

```
timeRandomForest
```

```
## Time difference of 2.310153 mins
```