R and RStudio – Tutorial

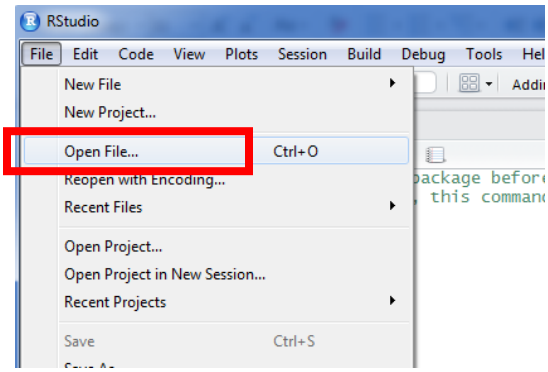
Decision Trees and Random Forests in R

STEP 1 – Open the script file "decision_tree_gt_mgt.R"

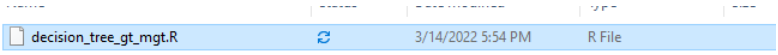
Two ways to do it:

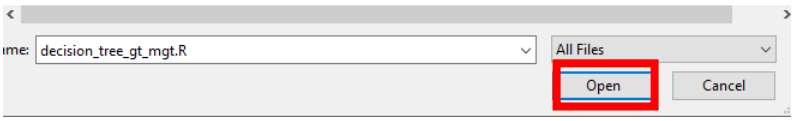
- **Method 1**: Do not open RStudio. Just double-click on the script code for the Decision Tree. This will do two things. First, it will open the R script code in RStudio. Second, it will set the working directory to that particular directory where the script code file was.
  - **It seems that for some versions of macOS, if you double-click it opens R instead of RStudio – if that happens, use Method 2.**

- **Method 2:** Open RStudio first. Then go to "File -> Open file".
  (a) Navigate to the script file on your computer



(b) Select the file and click Open





STEP 2 – Install necessary libraries

(a) Run the following commands to install required libraries – you only need to do this once for every install.packages command

```
> install.packages("rpart")
> install.packages("rpart.plot")
> install.packages("tidyverse")
```

(b) Load the libraries every time you need to run the code (the fact that they are installed does not mean they are loaded when you start R).

```
> library(rpart)
> library(rpart.plot)
> library(tidyverse)
```

STEP 3 – Load the dataset for analysis

Run the following command to load the data file in R. Copy and paste the link to the location of the data file on your PC in the code below.

- **Method 1:** If you opened the script file by double-clicking and your titanic.csv file is in the same folder, that is already the working directory and you do not need to set the path. Simply execute:

  ```
  >titanic <- read.csv("titanic.csv")
  ```

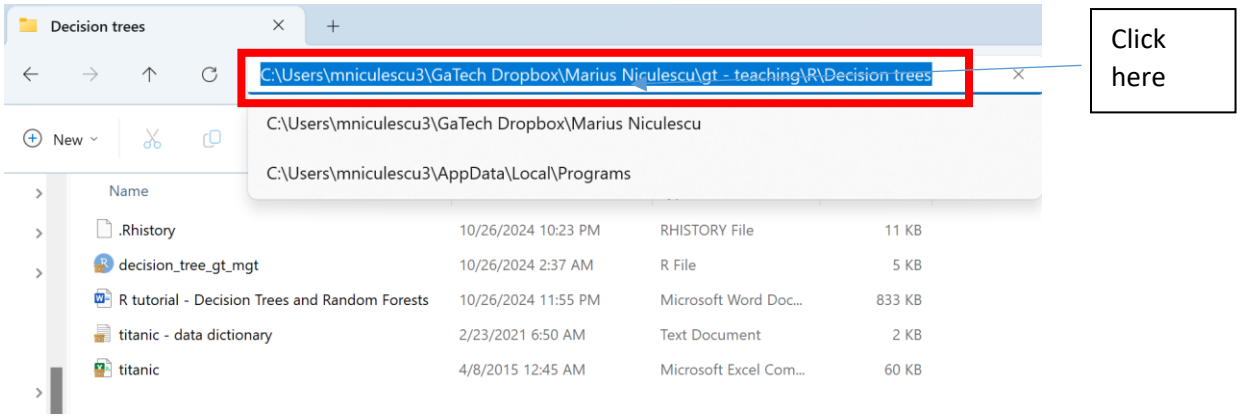
- **Method 2:** If titanic.csv file is not in your working directory, then you have to **add the path** The code below is an example (with the path on the instructor's computer) to load the data in the transaction format:

  ```
  > titanic <- read.csv("C:/Users/mniculescu3/GaTech Dropbox/Marius
  Niculescu/gt - teaching/R/Decision trees/titanic.csv")
  ```

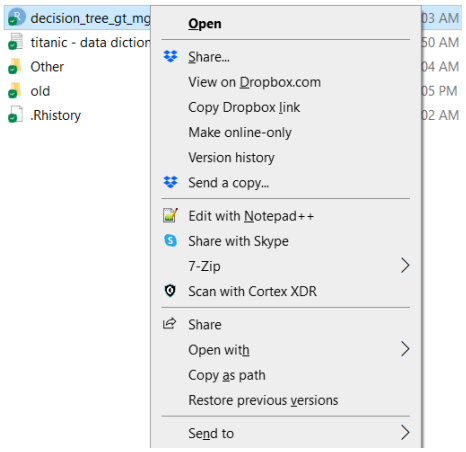
  Note!!! – in the script file, you have the example of the path on the instructor's computer – **you have to change that to the path on your computer** for the folder containing the gr oceries file.

  How to find the **PATH** of the file on your computer:

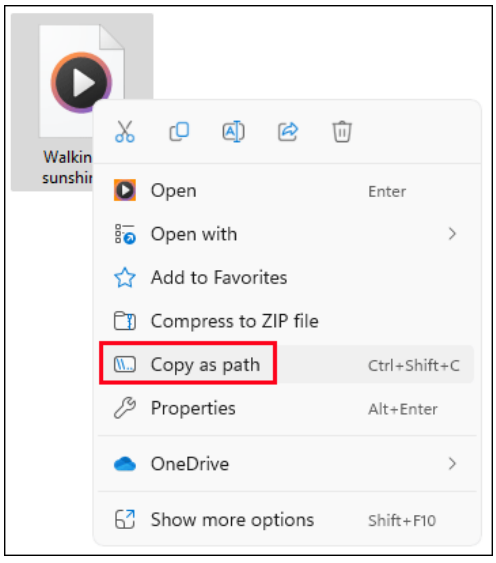
  (i) Windows 10/11 - solution 1 – once you are in the respective folder, the path of the folder will be at the top (if you click on the top box, not on the text but on the white space). Just select it, copy it, and pasted it in R.



(ii) <u>Windows 10 - solution 2</u> – holding down SHIFT key, right click on the file and select "Copy as path".



(iii) <u>For Windows 11</u> – you can right-click directly on the target file and just select "Copy a s path".



Remember that Windows users need to change all the separators from "\" to "/" in the address link.

(iv) Mac OS - several methods

Approach 1 - The easiest is to locate the file in Finder, then select it, then press simultaneously **OPTION + COMMAND + C**. Instead of copying the file, this copies the path. Then go to R code and paste the path using **COMMAND + V**.

Approach 2 - Right-Click Method:

- Locate the file or folder in Finder.
- **Right-click** on the file or folder while holding down the **Option** key.
- Select **Copy [item] as Pathname** from the context menu. This will copy the file path to your clipboard

Additional resources:

- https://setapp.com/how-to/how-to-find-the-path-of-a-file-in-mac

- **Method 3 - Setting the working directory**

Note that you can put all your files in the same directory and use the command setwd() to set that directory as the working directory (thus effectively eliminating the need to put the path every time you load a file). Do keep again in mind this is the pass on the instructor system so you need to put the path on your own system to that directory

```
>setwd ("C:/Users/mniculescu3/GaTech Dropbox/Marius
Niculescu/gt - teaching/R/Decision trees/")
```

STEP 4 – Data overview

Get a grasp of the data by calculating descriptive statistics for the data. There are several options can be used here:

(a) Use **summary()** function to get the descriptive statistics

```
> summary(titanic)
```

R will generate the following type of output. Each column provides the summary statistics for a particular variable. We can see that for continuous variables R calculates minimum, maximum, mean, and median values. For discrete variables, frequencies for each entry are calculated.

```
  PassengerId       Survived          Pclass                                      Name
Min.   :  1.0   Min.   :0.0000   Min.   :1.000   Abbing, Mr. Anthony                  :  1
1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000   Abbott, Mr. Rossmore Edward          :  1
Median :446.0   Median :0.0000   Median :3.000   Abbott, Mrs. Stanton (Rosa Hunt)     :  1
Mean   :446.0   Mean   :0.3838   Mean   :2.309   Abelson, Mr. Samuel                  :  1
3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000   Abelson, Mrs. Samuel (Hannah Wizosky):  1
Max.   :891.0   Max.   :1.0000   Max.   :3.000   Adahl, Mr. Mauritz Nils Martin       :  1
                                                 (Other)                              :885
    Sex           Age            SibSp           Parch             Ticket         Fare
female:314   Min.   : 0.42   Min.   :0.000   Min.   :0.0000   1601     :  7   Min.   :  0.00
male  :577   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000   347082   :  7   1st Qu.:  7.91
             Median :28.00   Median :0.000   Median :0.0000   CA. 2343:  7   Median : 14.45
             Mean   :29.70   Mean   :0.523   Mean   :0.3816   3101295  :  6   Mean   : 32.20
             3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000   347088   :  6   3rd Qu.: 31.00
             Max.   :80.00   Max.   :8.000   Max.   :6.0000   CA 2144  :  6   Max.   :512.33
             NA's   :177                                      (Other)  :852
        Cabin     Embarked
            :687     :  2
B96 B98     :  4   C:168
C23 C25 C27:  4   Q: 77
G6          :  4   S:644
C22 C26     :  3
D           :  3
(Other)     :186
```

(b) Another useful command to calculate the summary statistics is **stargazer()** .

To use this command, you need to install and load the package "stargazer" first

```
> install.packages("stargazer")
> library(stargazer)

> stargazer(titanic, type="text", iqr=TRUE)
```

R will generate the following output in the Console.

```
===============================================================
Statistic    N   Mean   St. Dev.  Min  Pctl(25) Pctl(75)  Max
---------------------------------------------------------------
PassengerId 891 446.000 257.354    1    223.5    668.5    891
Survived    891  0.384   0.487     0      0        1        1
Pclass      891  2.309   0.836     1      2        3        3
Age         714 29.699   14.526  0.420  20.125   38.000  80.000
SibSp       891  0.523   1.103     0      0        1        8
Parch       891  0.382   0.806     0      0        0        6
Fare        891 32.204   49.693  0.000  7.910    31.000  512.329
---------------------------------------------------------------
```

Stargazer() is similar to summary in terms of the calculated statistics. The scope of the command is limited to the continuous variables only. Advantages of **stargazer()** is that is providing output in a little bit more readable form.

STEP 5 – Partition the data set into training and test sets

Partitioning is the common practice to ensure that the created classification rules of the decision tree are accurate and reliable.

Partitioning breaks the dataset into two parts: training, and testing.

- **Training** subset of the dataset used to train the decision tree algorithm to generate the rules for the dataset.
- **Testing** subset is used to ensure the consistency of the performance of the classification rules of the decision tree.

The following R code allows to partition the dataset into training, and testing subsets:

(a) First we randomly assign rows of the titanic dataset into the training and testing subsets. We do so with the help of the sample() function.

```
> ind = sample(1:nrow(titanic),floor(nrow(titanic)*0.6))
```

The general form for the sample command is:

sample(x, size, replace = FALSE, prob = NULL)

where:

x – data we sample from. In our case `1:nrow(titanic)` or all the rows of the titanic datase t.

size – the size of the sample . In our case `nrow(titanic)*0.6` or 60% of all rows in titanic da taset.

Replace=FALSE – we do sample selection without replacement. FALSE is a default statement th us it can be omitted.

prob = NULL – probability weights. Prob=NULL is a default statement and can be omitted we do not assign any probability weights.

The resulting `ind` variable contains the vector of 60% rows from the titanic dataset that were selected at random. Now we simply subset titanic dataset to leave only these rows using the following function:

```
> titanic_train = titanic[ind,]
```

We just created our training subset.
Logically, the rest of the rows that we did not use(`[-ind,]`) will belong to the testing subset.

```
> titanic_test = titanic[-ind,]
```

STEP 6 – Running the decision tree algorithm

In our example we will use **rpart** function from the **Rpart** library to create a decision tree.

```
> titanic_tree_class <- rpart(Survived ~ Pclass + Sex + SibSp + Age +
Fare + Parch, method="class", data=titanic_train)
```

The general form of the rpart function is:

rpart(formula, data, method, cp)

where

formula - a formula, with a response but no interaction terms. In our case it is `Survived ~ Pclass + Sex + SibSp + Age + Fare + Parch.` This simply indicates that we

want to associate a tag of 1 or 0 in terms of survival and we want the classification to take into account Pclass, Sex, SibSp, Age, Fare, Parch, Embarked.

method - can take the following values : class, exp, anova, poisson. The default method is **anova. Anova** is used when the dependent variable is continuous. **Class** method is applied when the dependent variable (in our case `Survived`) is a factor (0 or 1 in our example). **Poisson** is used when the dependent variable has two columns. **Exp** is used to create decision tree for survival analysis.

data – data.frame reference. Pay attention that we refer to the training subset of the dataset (`titanic_train`).

cp - adjusts the precision - complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.

Be aware that execution of the rpart function will not create any output in the Console. In order to see the created decision tree, you need to use **rpart.plot()** function that will visualize the results. **rpart.plot()** function will be discussed in the STEP 7.

STEP 7 – Visualize the decision tree algorithm

The **rpart.plot()** function below can be used to visualize the decision tree. You can learn a lot of the details of this function at

http://www.milbo.org/rpart-plot/prp.pdf

```
> rpart.plot(titanic_tree_class)
```

The general form for the **rpart.plot()** function is :

```
rpart.plot(x, tweak=, gap=, main= "", box.palette="", branch.lty=, shadow.col="", nn=, extra=, cp =)
```

where:

x – the object containing the decision tree created by one of the methods using **rpart** function (In our case we can refer to the objects `titanic_tree_class` or `titanic_tree_reg` that we created in the STEP 5.

tweak –argument that increases the text size

gap – reduces the space between the boxes and the tree space (makes the tree look less crowded)

main – adds a title to the decision tree visualization

box.pallette –defines the color scheme of the tree (some options include : 0- no color, “auto”, “grays”, “pink”, “blues”)

branch – changes angles of branch lines

branch.lty – branch line type (=1 by default). Plots branches using dotted lines)

shadow.col – adds shadow to boxes

nn – enables nodes' names (TRUE or FALSE)

extra – provides additional information at the node

There are many more different arguments you can use!

Refer to this links for the complete list of arguments:

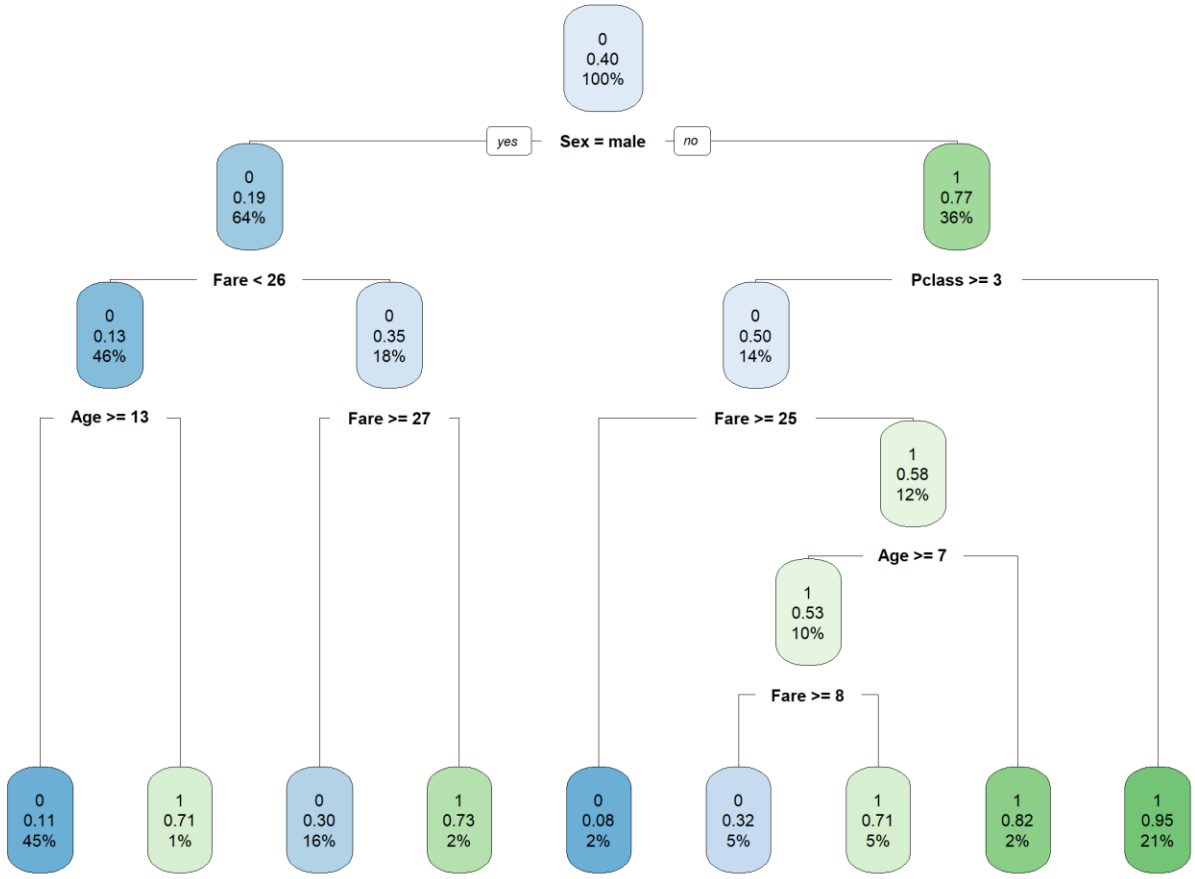
https://cran.r-project.org/web/packages/rpart.plot/rpart.plot.pdf

http://www.milbo.org/rpart-plot/prp.pdf

Let's look at the examples of the visualizations that we can get. The plots visualize the decision tree for `titanic_tree_class` and `titanic_tree_reg.` R will generate the decision tree in the Plot window.

The function below generates the default visualization of the decision tree. As we can see it is not very readable.

```
> rpart.plot(titanic_tree_class)
```



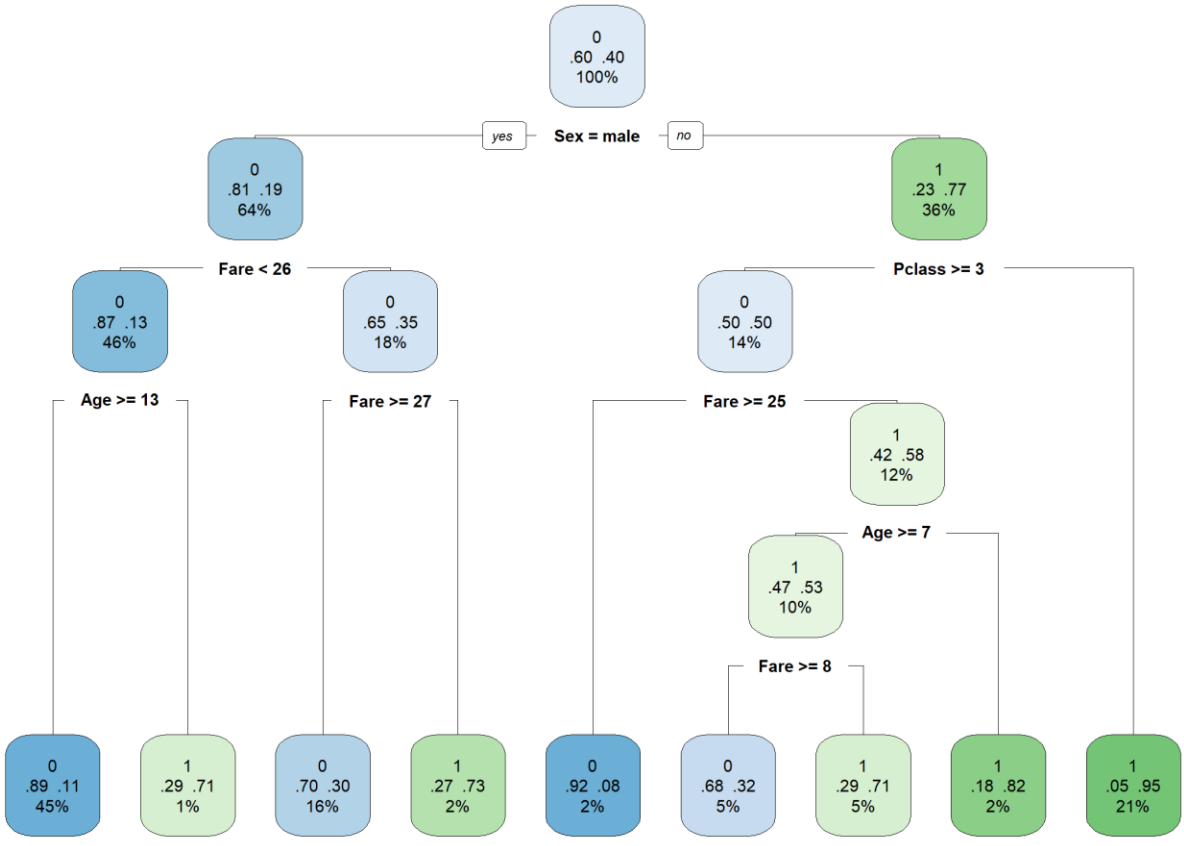
Each node shows:
-predicted class (0-died, 1-survived) – indicative of majority outcome for records at that node
-predicted probability of the binary event =1 (in our case survived=1)
-the percentage of observations in the node
-color scheme is colder towards 0 (tones of blue), warmer towards 1 (tones of green) based on likelihood

Let's try to use some of the rpart.plot arguments to augment the visualization:

(a) Argument extra = 104 – adds probability of both 0 and 1 – preferred for this course

```
> rpart.plot(titanic_tree_class, extra=104)
```

The topmost node (also often called the **root node**) corresponds to the best predictor of the respons e variable. In our example the root node is gender.
The **leaf nodes** are the nodes in very bottom of the decision tree represent the decision for the respo nse variable we used in the model based on our classification (in our example whether person is pred icted to survive or die).



Each node shows:
-predicted class (0-died, 1-survived)
-predicted probability of the binary events 0 and 1 in that order (in our case survived=1, did not survive is 0) – likelihood of 0 to the left, likelihood of 1 to the right
-the percentage of observations in the node
-color scheme is colder towards 0 (tones of blue), warmer towards 1 (tones of green) based on likelihood

(b) Increase the size of the labels and text – use **tweak** option

- **rpart.plot(titanic_tree_class, extra = 104, tweak = 1.5)**

STEP 8 – Prediction using the decision tree

(a) Now when we have created the classification rules with the help of rpart() function based on the training subset, we would like to use this model to predict the outcome for other observations. We can do this by using predict() function.

```
> survived_predict <- predict(titanic_tree_class, titanic_test, type="class")
```

The general form for the **predict()** function is:

```
predict(object, testdata, type="")
```

where:

object - fitted model object that we created using rpart() function (for example `titanic_tree_class`).

testdata - testing subset of the dataset (in our case it is `titanic_test`).

type – Can take 4 options ("vector", "prob", "class", "matrix"). Prob and class are used for classification trees and represent class probability and a factor of classifications respectively. Matrix types gives a matrix of full responses (mean response for regression, for classification it generates class counts, class probabilities etc.). Vector type generates a vector of predicted responses (regression case).

We selected type="class" in our case because we use **predict()** function for `titanic_tree_class` model that was generated for a binary dependent variable `Survived` in STEP 6 using class method in **rpart()** function.

You can read more here: https://cran.r-project.org/web/packages/rpart/rpart.pdf

(b) At this point we used our classification rules developed based on the training subset to predict survival outcome in the test subset. Now let's compare whether our predicted outcome matches the actual data. In other words, we would like to know the misclassification rate.

a. First, let's look at our prediction for the `Survived` that we created using the **predict()** function.

```
> survived_predict <- predict(titanic_tree_class, titanic_test, type="class")
```

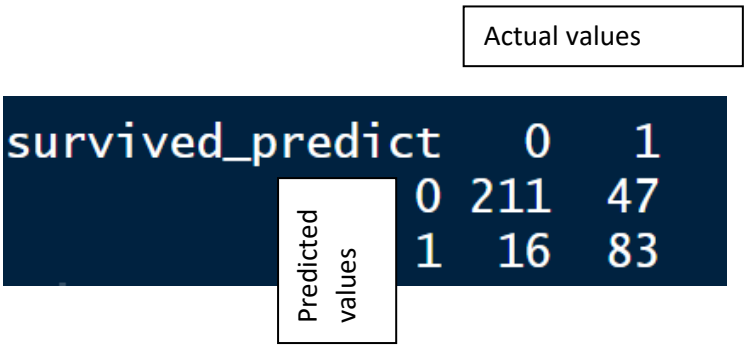
Create a **confusion matrix** using the following command:

```
> table(survived_predict, titanic_test$Survived)
```

NOTE – the code above gives a confusion matrix that has PREDICTED VALUES in the rows and ACTUAL VALUES in the columns (similar to the lecture slides). Sometimes confusion matrices are displayed the other way around, with actual values in the rows and predicted values in the columns.

R will create the following output in the **Console** window.

Note that the numbers you get are highly likely to be different because each time we randomly split into training and test sets and, as a result, we get different trees. Moreover, in the test set, you might get different number of survivors and nonsurvivors (again due to the random sampling).



We can interpret this table as follows:

The total number of passengers from the test subset that died (0) is 227 (the sum of 211 and 16). Of those 227, our decision tree was able to correctly predict the outcome for 211 people, yet 16 people were incorrectly classified as survivors.

The same logic follows for the survivor group. The total number of people who actually survived is 130. From those 130, 83 were predicted as survivors correctly and 47 were classified incorrectly.

STEP 9. Compute the **importance** of the **variables** in the decision tree using the following command:

```
> importance <- titanic_tree_class$variable.importance
> print(importance)
```

```
      Sex      Fare    Pclass     Parch       Age     SibSp
81.374941 34.480687 22.489998 11.602531  6.777761  6.706708
```

The rpart package has a built-in feature that records the **variable importance** for our decision tree models. The importance score is calculated based on the number of appearances of the variable in the nodes (as a splitting criterion) and the corresponding reduction in Gini Impurity achieved.

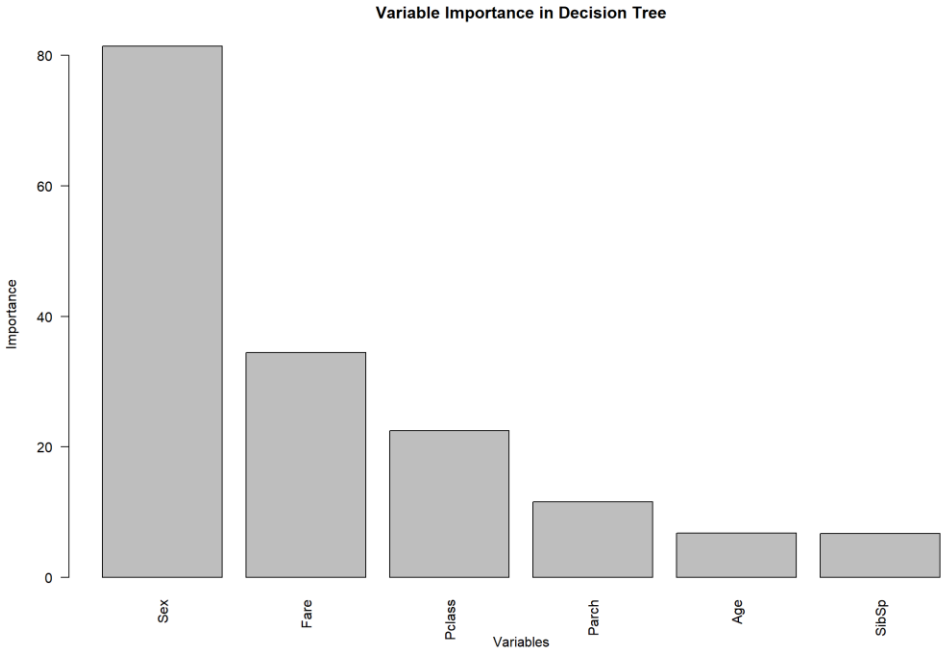
** Moreover, some variables ***that do not show up*** as direct split criteria can also get assigned importance. rpart assigns some credit for reducing impurity to **surrogates** based on how closely they match the primary split. This ensures that surrogates that strongly mirror the primary split's behavior contribute to the variable importance measure.

Surrogate splits are alternative splits used when the data for the primary splitting variable is missing. Surrogates are chosen based on how well they mimic the behavior of the primary split.

As you can see, **SibSp** and **Parch** have importance assigned too, even they do not show up in the tree. That is because of their ability as surrogates to match to some degree the primary split. As it happens, Parch actually has higher importance score than Age, with the latter appearing in the tree.

We can further visualize the output using **barplot** command as follows:

```
> barplot(importance, main = "Variable Importance in Decision Tree", xlab =
"Variables", ylab = "Importance", las = 2, col = "grey")
```



R will generate a bar-plot that shows the variable importance of each independent variable.

As expected, the variable associated with the split at the **root node** (i.e., Gender) is the most important variable in our prediction model.

To read more about the way the rpart algorithms works and now variable importance is computed, you can read the following document (section 3.4 for variable importance computation) : https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Note that you can see a normalized and rounded variable importance also by running the summary command (summary(titanic_tree_class))

STEP 10 – Preliminary steps to generate a random forest

Firstly, we need to install and load the needed packages:

```
> install.packages("caTools")
> install.packages("randomForest")

> library(caTools)
> library(randomForest)
```

The randomForest function in R does not handle well missing data. We need to remove records from the training and test sets for instances where there is some missing data at that record. We do that by using command na.omit() - it removes the data entries that contains missing data. [1]

```
> titanic_train_RF <- na.omit(titanic_train)
> titanic_test_RF <- na.omit(titanic_test)
```

Then we need to convert the target variable (in this case – Survived) to a factor (a categorial) variable for classification purposes:

```
> titanic_train_RF$Survived <- as.factor(titanic_train_RF$Survived)
> titanic_test_RF$Survived <- as.factor(titanic_test_RF$Survived)
```

STEP 11 – Build a Random Forest Model

The decision tree models we just covered are intuitive and straightforward. However, they are also prone to overfitting and skewness in the training and test set. The random forest model builds multiple decision trees and merges them together to get a more accurate prediction.

First we can specify a seed for the random forest generator to ensure that our results are reproducible (i.e., the same seed generates the same sets when sampled with replacement). Here we choose the seed 120.

```
> set.seed(120)
```

Each tree for a random forest model is trained on a random sample of the training data. The common sampling method is called **Bootstrap Sampling**, which is sampling with replacement. (i.e. an observation might be chosen multiple times for a sample). These trees are then merged together for the final prediction result through **Aggregation**. Sometimes you may see people call it **Bagging** (Bootstrap Aggregation).

We use the randomForest() function to build the model.

[1] Another way of handling missing data is imputation, which fills in missing values with the mean/ median/a random number etc

The general form of randomForest() function are as follows:

```
randomForest(formula, data, ntree, mtry)
```

where

formula - a formula, with a response but no interaction terms. In our case it is `Survived ~ Pclass + Sex + SibSp + Age + Fare + Parch`

data - the training data.

ntree - Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

mtry - Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

Note that when the dataset involves many variables, it may take a long time to build a random forest. In this case, we can use the maxnodes argument to specify how deep each tree can be:

maxnodes - Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued.

In our demo, we build a random forest with 500 trees and specify mtry to 2. The code looks like:

```
> titanic_RF <- randomForest(Survived ~ Pclass + Sex + SibSp + Age +
Fare + Parch, data=titanic_train_RF, ntree = 500, mtry = 2)
```

STEP 12. Compute the **importance** of the **variables** in the random forest model using the following command:

```
> importance(titanic_RF)
```

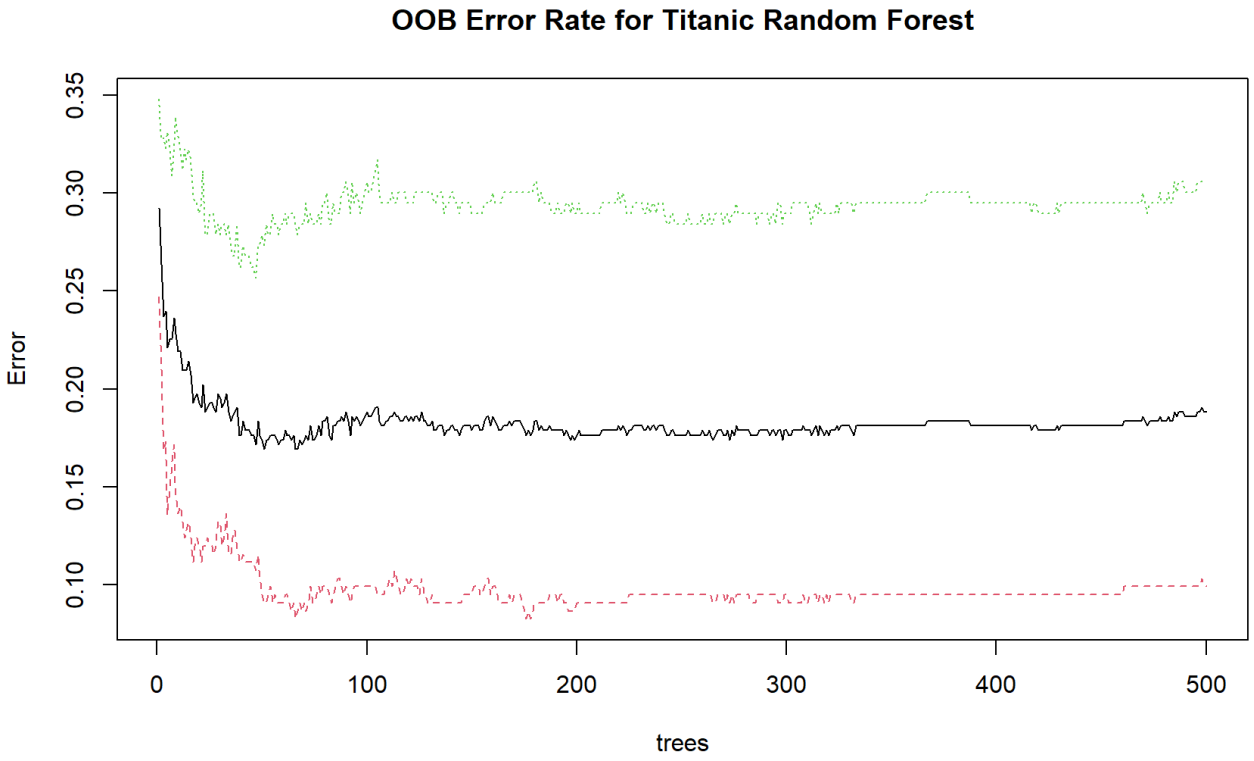
```
       MeanDecreaseGini
Pclass         20.463610
Sex            53.869526
SibSp           8.239648
Age            34.002416
Fare           35.703401
Parch           7.342970
```

For the random forest model we built, Gender is shown as the most important variable. Note that now Age is featuring more prominently in terms of importance (ranked 3rd , but very close to 2nd). Again, importance is based on primary splits and also surrogate splits.

STEP 13. Compute the **OOB (out-of-bag error) for the random forest**

```
> plot(titanic_RF, main = "OOB Error Rate for Titanic Random Forest")
```

Second argument "main = "allows you to change the title of the plot



- **Black Line**: This represents the **overall OOB error rate**. It shows the misclassification rate for the entire dataset as the number of trees increases.
- **Red Line**: This represents the OOB error rate for **class 0** (e.g., passengers who did not survive).
- **Green Line**: This represents the OOB error rate for **class 1** (e.g., passengers who survived).

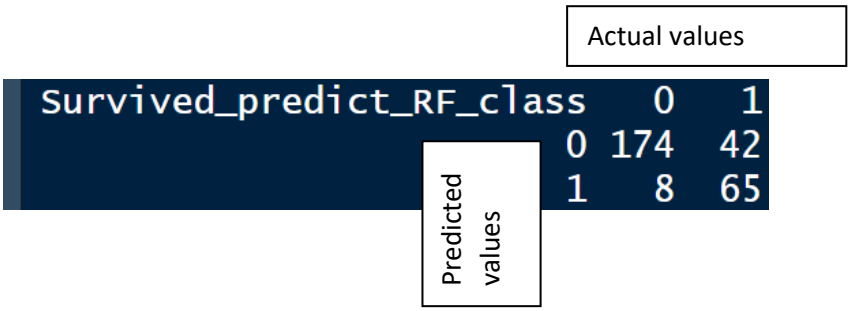
As we can see from the plot, the OOB **error** decreases with the number of trees and stabilizes after about 100 trees.

STEP 14 – Predict with Random Forest Model

Now, let's take our model into the test. We will start with generating the predictions on the test dataset.

```
> Survived_predict_RF_class <- predict(titanic_RF, newdata =
titanic_test_RF, type = "class")
```

```
> cm_RF_class <- table(Survived_predict_RF_class,
titanic_test_RF$Survived)
> print(cm_RF_class)
```



Note that here the total number of observations in the test set does not match with our previous observations for decision tree model due to omitting missing data.