Project_Netflix

I will first begin by splitting the Netflix data into the training and test data sets.

```
library(ISLR2)
library(tree)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
Netfl <- read.csv("Best Movies Netflix.csv")

Net <- subset(Netfl, select = c(RELEASE_YEAR:MAIN_GENRE))

Net$MAIN_GENRE <- as.factor(Net$MAIN_GENRE)


Netflix<-Net%>%
  as_tibble()

set.seed(456)

netflix_index = sample(1:nrow(Netflix), nrow(Netflix)/2)

NetflixTrain_set = Netflix[netflix_index,]

NetflixTest_set = Netflix[-netflix_index,]
```

The regression tree will only work on factor variables and numeric variables, so the 'MAIN_GENRE' variable was changed in order to prevent NA's introduced by coercion. This is also why the 'TITLE' variable and

‘MAIN_PRODUCTION’ variable were not included because transforming these variables created too many factors.

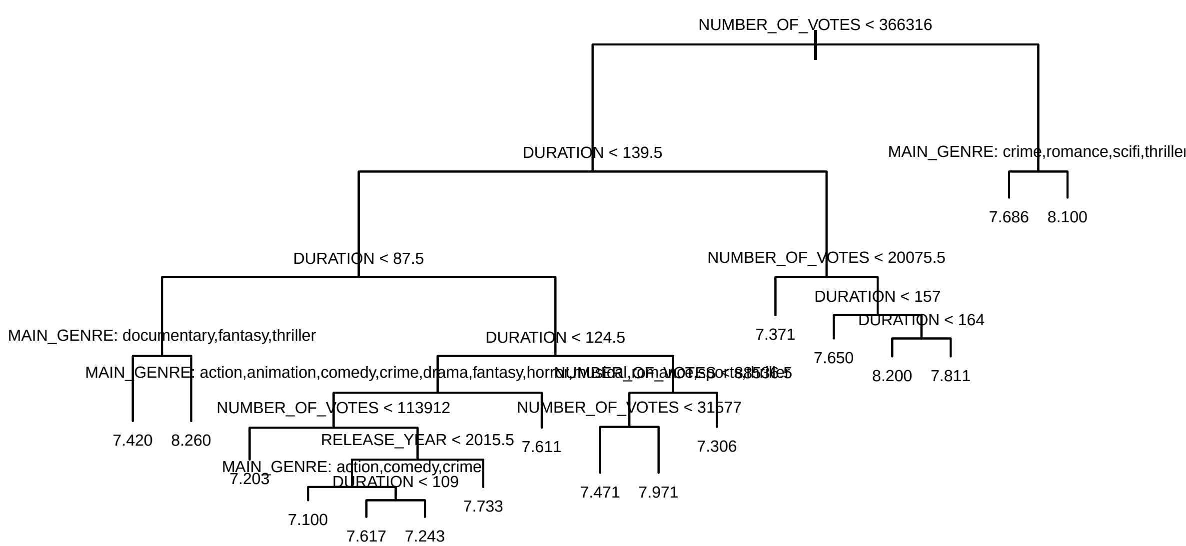
The regression tree will be fitted.

```r
Netflix_regressiontree <- tree(SCORE~., NetflixTrain_set)

summary(Netflix_regressiontree)
```

```
##
## Regression tree:
## tree(formula = SCORE ~ ., data = NetflixTrain_set)
## Number of terminal nodes:  17
## Residual mean deviance:  0.08655 = 15.23 / 176
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.61110 -0.21110 -0.00625  0.00000  0.20000  0.75000
```

This tree was plotted in order to develop a visualization of it.

```r
plot(Netflix_regressiontree)
text(Netflix_regressiontree,pretty=0,cex =0.5)
```



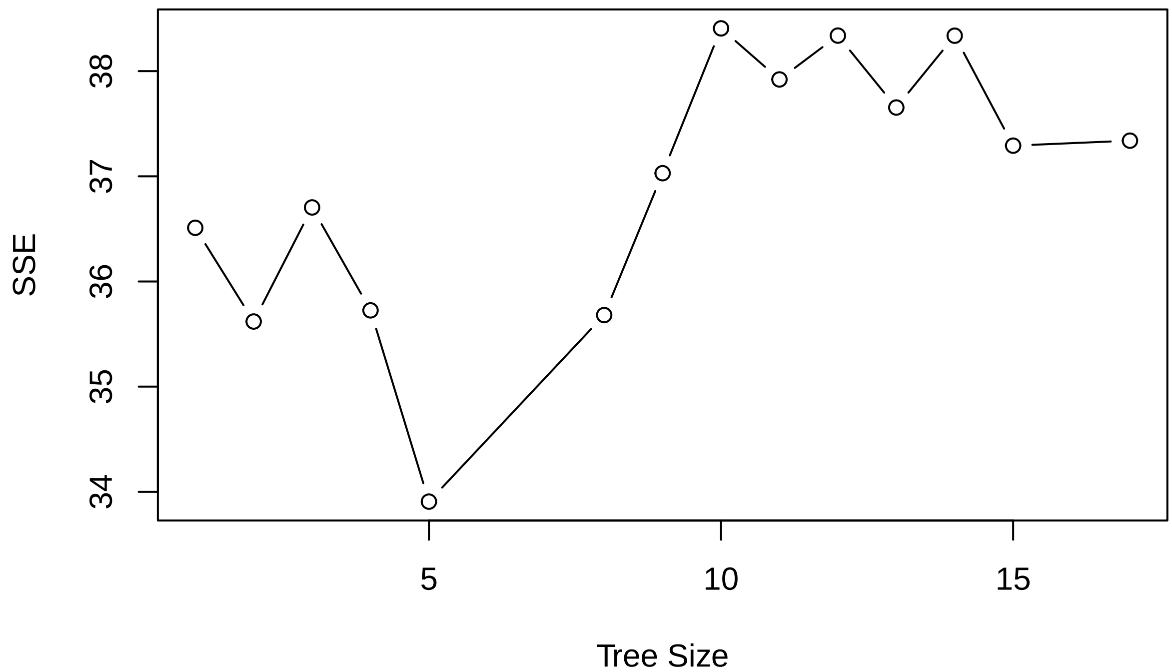
This regression tree will also be pruned with cross-validation.

```r
set.seed(456)

Pruned_Netflix <-cv.tree(Netflix_regressiontree)

Pruned_Netflix
```

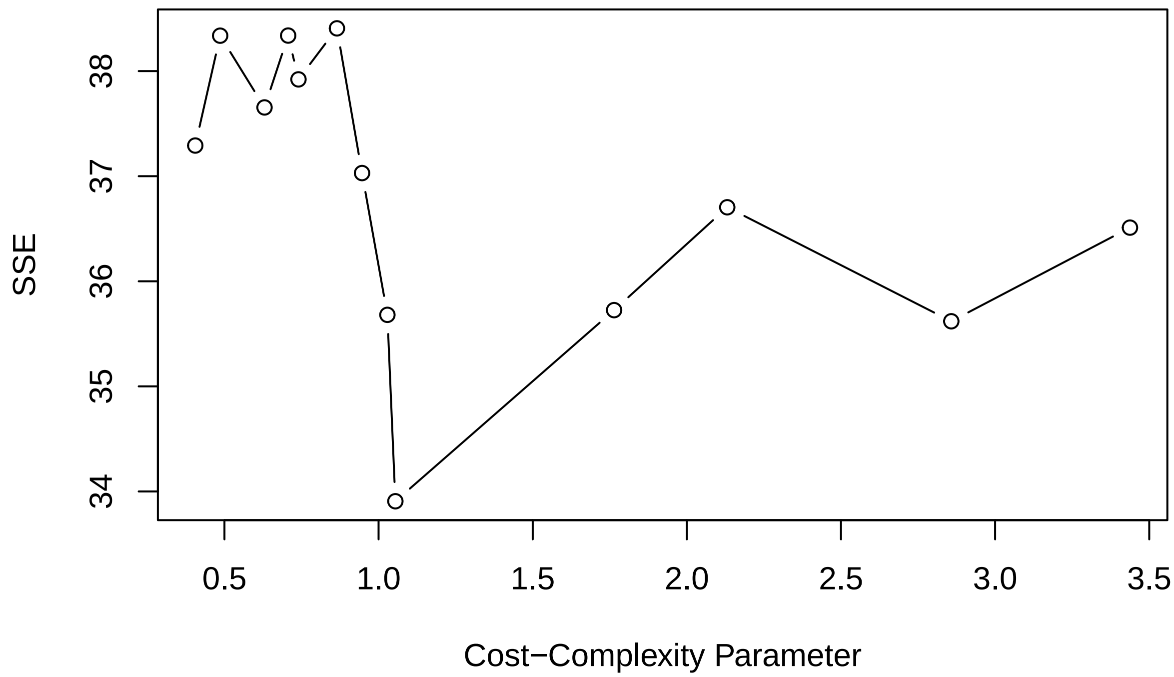
```
## $size
##  [1] 17 15 14 13 12 11 10  9  8  5  4  3  2  1
##
## $dev
##  [1] 37.33907 37.29157 38.33699 37.65415 38.33797 37.92118 38.40678 37.02964
##  [9] 35.68048 33.90672 35.72537 36.70415 35.61927 36.51040
##
## $k
##  [1]      -Inf 0.4053175 0.4861111 0.6300000 0.7067227 0.7401389 0.8648868
##  [8] 0.9463416 1.0285714 1.0544949 1.7640000 2.1310744 2.8578339 3.4374900
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Plots were developed in order to see the results from performing cross-validation on this pruned tree.

```r
plot(Pruned_Netflix$size, Pruned_Netflix$dev, type = "b",
     xlab = "Tree Size", ylab = "SSE")
```




```r
plot(Pruned_Netflix$k, Pruned_Netflix$dev, type = "b",
     xlab = "Cost-Complexity Parameter", ylab = "SSE")
```



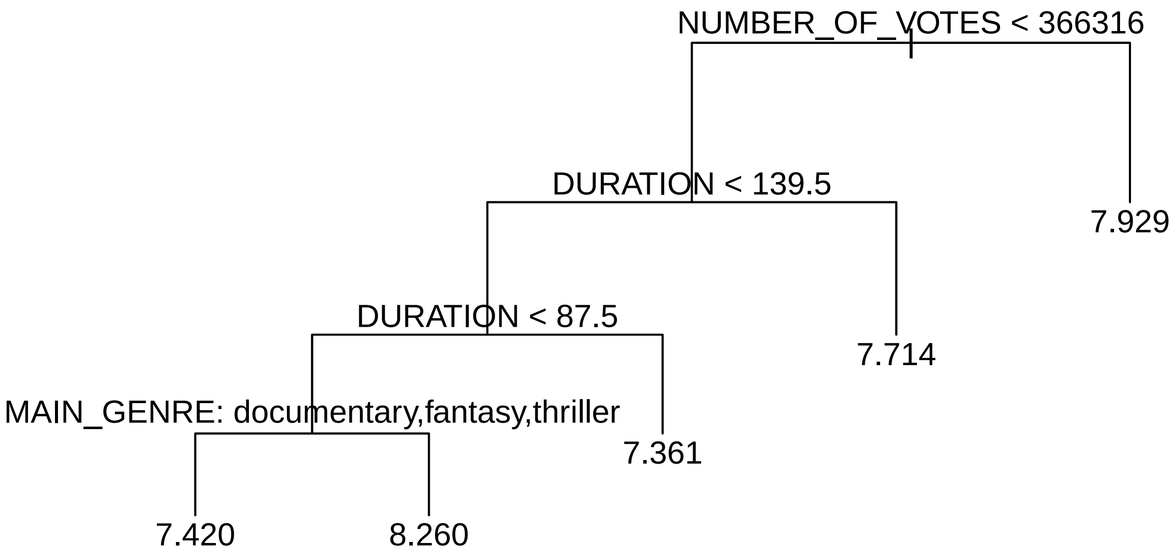
We should choose the tree with the lowest error, which is the tree with 5 nodes. Then, we will predict on the test dataset.

```
NetflixLowestErrorTree = prune.tree(Netflix_regressiontree, best = 5)
summary(NetflixLowestErrorTree)
```

```
##
## Regression tree:
## snip.tree(tree = Netflix_regressiontree, nodes = c(3L, 5L, 9L
## ))
## Variables actually used in tree construction:
## [1] "NUMBER_OF_VOTES" "DURATION"        "MAIN_GENRE"
## Number of terminal nodes:  5
## Residual mean deviance:  0.1309 = 24.61 / 188
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.72940 -0.26110 -0.01429  0.00000  0.23890  0.98570
```

We will create a plot of this pruned tree.

```
plot(NetflixLowestErrorTree)
text(NetflixLowestErrorTree, pretty = 0)
```



This tree will be used to form the predictions on the test data set.

```
predictedbestNextflixtree <-predict(NetflixLowestErrorTree,newdata=NetflixTest_set)

predictedbestNextflixtree
```

```
##        1        2        3        4        5        6        7        8
## 7.420000 7.929412 7.929412 7.929412 8.260000 7.714286 7.361069 7.714286
##        9       10       11       12       13       14       15       16
## 7.714286 8.260000 7.929412 7.714286 7.361069 7.361069 7.714286 7.361069
##       17       18       19       20       21       22       23       24
## 8.260000 7.361069 7.929412 7.714286 7.714286 7.420000 7.929412 8.260000
##       25       26       27       28       29       30       31       32
## 7.714286 7.714286 7.714286 7.361069 7.361069 7.361069 7.420000 7.714286
##       33       34       35       36       37       38       39       40
## 7.929412 7.714286 7.929412 7.361069 7.929412 7.714286 7.840000 7.361069
##       41       42       43       44       45       46       47       48
## 7.929412 7.361069 7.929412 7.929412 7.420000 8.260000 7.361069 8.260000
##       49       50       51       52       53       54       55       56
## 7.714286 7.714286 7.929412 7.929412 7.929412 7.361069 7.929412 7.929412
##       57       58       59       60       61       62       63       64
## 7.361069 7.361069 7.714286 7.361069 7.361069 7.361069 7.361069 7.361069
##       65       66       67       68       69       70       71       72
## 7.361069 7.420000 7.929412 7.714286 7.361069 7.929412 7.361069 7.361069
##       73       74       75       76       77       78       79       80
## 7.361069 7.714286 7.361069 7.714286 7.361069 7.361069 7.361069 7.840000
```

```
##       81       82       83       84       85       86       87       88
## 7.929412 7.361069 7.361069 7.361069 7.714286 7.714286 7.361069 7.361069
##       89       90       91       92       93       94       95       96
## 7.714286 7.361069 7.361069 7.361069 7.929412 7.361069 7.361069 7.361069
##       97       98       99      100      101      102      103      104
## 7.361069 7.361069 7.714286 7.714286 7.714286 7.361069 7.361069 7.361069
##      105      106      107      108      109      110      111      112
## 7.361069 7.929412 7.361069 7.361069 7.714286 7.361069 7.714286 7.361069
##      113      114      115      116      117      118      119      120
## 7.361069 7.361069 7.714286 7.361069 7.420000 7.361069 7.714286 7.361069
##      121      122      123      124      125      126      127      128
## 7.361069 7.361069 7.714286 7.361069 7.714286 7.361069 7.840000 7.361069
##      129      130      131      132      133      134      135      136
## 7.714286 7.714286 7.361069 7.420000 7.361069 7.361069 7.361069 7.361069
##      137      138      139      140      141      142      143      144
## 7.361069 7.361069 7.361069 7.840000 7.361069 7.361069 7.361069 7.714286
##      145      146      147      148      149      150      151      152
## 7.361069 7.361069 7.361069 7.361069 7.361069 7.361069 7.361069 7.361069
##      153      154      155      156      157      158      159      160
## 7.361069 7.361069 7.714286 8.260000 7.361069 7.361069 7.361069 7.714286
##      161      162      163      164      165      166      167      168
## 7.361069 7.714286 7.714286 7.361069 7.361069 7.361069 7.361069 7.361069
##      169      170      171      172      173      174      175      176
## 7.361069 7.361069 7.361069 7.361069 7.361069 7.361069 7.714286 7.361069
##      177      178      179      180      181      182      183      184
## 7.714286 7.361069 7.714286 7.361069 7.714286 7.361069 7.361069 7.714286
##      185      186      187      188      189      190      191      192
## 7.361069 7.361069 7.361069 7.714286 7.361069 7.361069 7.714286 7.361069
##      193      194
## 7.361069 7.361069
```

We will compute the RMSE through the creation of this function.

```r
rmse<-function(actual, predicted){
  rmse=sqrt(mean((actual - predicted) ^ 2))
  mse= mean((actual-predicted)^2)
  c(rmse,mse)
}
```

The performance of this tree will be evaluated through using RMSE.

```r
rmse(NetflixTest_set$SCORE,predictedbestNextflixtree)
```

```
## [1] 0.4234998 0.1793521
```

We will load this library in order to perform random forest and bagging.

```r
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

We will perform bagging with atleast 500 trees.

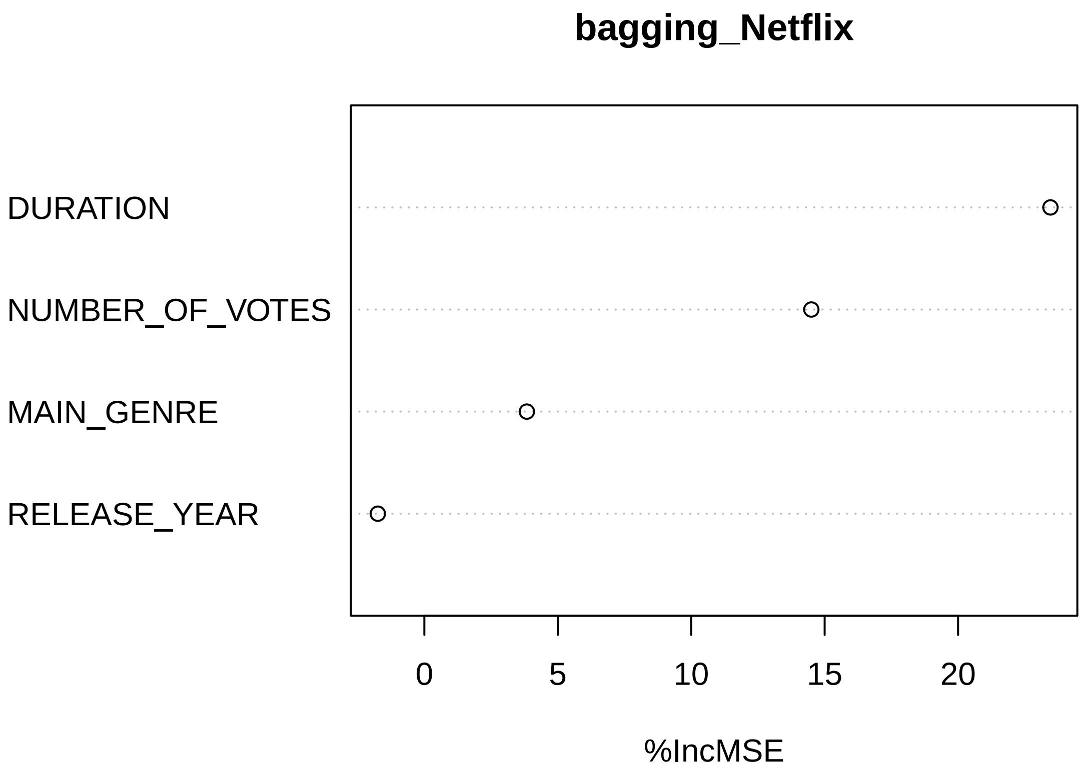
```
set.seed(458)

bagging_Netflix <-randomForest(SCORE~.,data=NetflixTrain_set,mtry=4,importance=TRUE,ntree=500)

bagging_Netflix
```

```
##
## Call:
##  randomForest(formula = SCORE ~ ., data = NetflixTrain_set, mtry = 4,      importance = TRUE, ntree =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.1647302
##                     % Var explained: 8.64
```

```
importance(bagging_Netflix,type = 1)
```

```
##                    %IncMSE
## RELEASE_YEAR    -1.744600
## NUMBER_OF_VOTES 14.500684
## DURATION        23.462023
## MAIN_GENRE       3.841185
```

```
varImpPlot(bagging_Netflix,type = 1)
```



The test data set will be used for the predictions.

```
baggingpredictions_Netflix <- predict(bagging_Netflix,newdat=NetflixTest_set)

rmse(NetflixTest_set$SCORE, baggingpredictions_Netflix)
```

```
## [1] 0.4102320 0.1682903
```

Random forest will now be implemented on the Netflix dataset. Since this is regression, the total number of predictors divided by 3 will be the value that is selected for the mtry function.

```
set.seed(458)

Netflix_randomforests <- randomForest(SCORE~.,data=NetflixTrain_set,mtry=1.333333,importance=TRUE,ntree=

Netflix_randomforests
```

```
##
## Call:
##  randomForest(formula = SCORE ~ ., data = NetflixTrain_set, mtry = 1.333333,      importance = TRUE,
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1632537
##                     % Var explained: 9.46
```

The predictions will be developed on the test set now.

```
randomforestpredictions_Netflix <-predict(Netflix_randomforests,newdat=NetflixTest_set)

rmse(NetflixTest_set$SCORE, randomforestpredictions_Netflix)
```

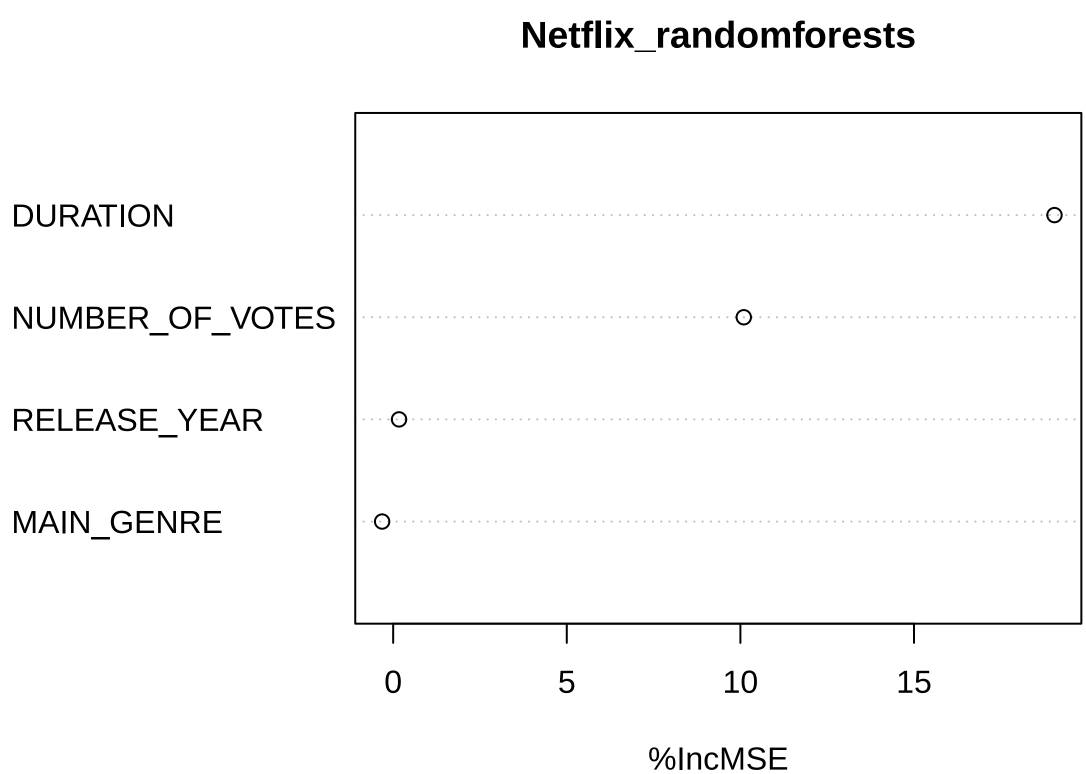
```
## [1] 0.4120424 0.1697789
```

We will begin to examine the importance of each variable and how they operate in the splits of the 500 trees through these two visualizations.

```
importance(Netflix_randomforests,type = 1)
```

```
##                    %IncMSE
## RELEASE_YEAR     0.1692325
## NUMBER_OF_VOTES 10.0981472
## DURATION        19.0453766
## MAIN_GENRE      -0.3175460
```

```
varImpPlot(Netflix_randomforests,type = 1)
```



This visual uses the information from random forest's variable importance plot to create a colorful visualization in the form of a bar plot.

```r
library(ggplot2)

RF <- data.frame(match = c("Duration", "Number of Votes", "Release Year", "Main Genre"),
                 runs = c(19.0453766, 10.0981472, 0.1692325, -0.3175460))

RandomForestBarPlot <-ggplot(data = RF, aes(x =match, y = runs, fill = match)) +
  geom_bar(stat = "identity", width = 0.5) +
  labs(y = "%IncMSE",
       x = "Predictors",
       title = "Variable Importance: Random Forest")

RandomForestBarPlot + theme(legend.position ="none")
```