Predicting Train Arrival Status - On Time or Late

Introduction

Adanna Alutu

June 6, 2017

At the beginning of the project, it was hard to come up with a good data to analyze and predict the outcome. Initially I wanted to work on data from my job but after we coudn't see much dependence among the fields that made sense, my mentor Dr. Shmuel Naaman advised me to scout for data from other internet sites he recommended.

Since I take the train most of the time and experienced delay issues many times that has ranged from 10 mins to 2 hours, I became interested in working on transportation data for trains. This is because I want to experience the process of predicting outcomes which is made possible through Data Science. I want to focus on the steps that will make it possible for me and my mentor Dr Shmuel Naaman to predict the arrival times of the train. The possibility of cutting down the delays experienced in waiting for the train no longer seems to be far fetched. My mentor agreed with me and the Septa Train data from Kaggle website was a good option to work on. There were 3 different datasets available to work on but I chose the "on time performance" which I felt has more relevant features, variables and observations and also has sufficient data for the analysis, tests involved.

The variables in the dataset include: 1. train_id 2. status 3. origin 4. direction 5. next_station 6. timeStamp 7. date

subtitle:

Data Exploration

Several steps were taken to ensure elaborate data analysis and wrangling. Every bit of the data was maximized. We went beyond using the provided variables by creating new ones, removing unnecessary data and testing with reliable tools to get quality, reliable results that can be tested with any dataset.

It was necessary to take the following steps to ensure that all the combinations, dicing and testing would yield a meaningful interpretation and prediction that will help tell us with high confidence when the train will be late:

I. We first tried to plot charts with the entire data but the plots were too crowded and blurry to make any sense. The scales were distorted with big units affected by the outliers.

II. GGPlot bar charts were used to plot and observe the trends and statistics summary but the dataset was too huge for the charts.

III. My mentor suggested shuffling the data and taking the first 20percent as sample to work on. Using the formula below, the row-wise shuffling was done first before the column was then shuffled:

IV. We used the data to fit in several models which include:

- GGPlot with different combination of the variables.
- Linear regression model which was used different ways to get the best stastical summary. Including using some of the observations as variables.
- CART model with focus on the classification method because most of the variables in the data are categorical and the prediction is binary with 0 as "on Time" and 1 as "Late"
- Random Forest which created it's own model that highlighted the top more meaningful variables that contributed majorly in predicting the outcome.

Each of these models were implemented because the train dataset contains a mixture of numerical and categorical variables. Converting their types to either numeric or factors wasn't sufficient. To get the benefit of all the variables, it was essential to test these models.

subtitle:

Data Wrangling

Some data manipulations were done which include: + splitting some of the original variables nto separate variables. For example, time stamp variable was split into six variables. year, month day, hour, min, seconds. + Irrelevant variables were removed or set to null so they would not appear in the dataframe used for the predictions. + Some of observations from the wkday and day of month variables were converted to variables and they significantly improved the statistics of the models. The additions however increased the number of variables from 11 to 58. + Units attached to the dependent variable observations were removed to enable convesions to different types and allow plotting with only the observations of the same type. + The dependent variable "status" observations of "on Time" were replaced with "0" using gsub so that all the observations for the variable will match and easier to manipulate."On time" meant the train arrived as scheduled so it made sense to use "0" to represent no delay.

subtitle:

A Peek into some new variables

This section shows the summary of the SEPTA train data and the first few records using the head().

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
## Warning: Too many values at 3000 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
## 'data.frame':    3000 obs. of  12 variables:
##  $ X.1         : int  84286 38729 22134 61549 149035 178337 137327 138757 77291 95823 ...
##  $ X           : int  92291 1380663 752547 39786 795013 1649075 1440399 757500 1654914 660273 ...
##  $ next_station: Factor w/ 153 levels "30th Street Station",..: 86 123 94 38 118 42 97 90 19 46 ...
##  $ direction   : Factor w/ 2 levels "N","S": 2 2 2 1 2 1 2 2 1 1 ...
##  $ status      : chr  "4 " "5 " "0" "0" ...
##  $ origin      : Factor w/ 88 levels "16th St Jct",..: 26 66 57 33 16 73 24 24 73 48 ...
##  $ train_id    : Factor w/ 785 levels "1054","1055",..: 96 590 301 353 515 377 348 364 418 652 ...
##  $ monthday    : chr  "01" "08" "13" "27" ...
##  $ hour        : chr  "16" "08" "05" "09" ...
##  $ minute      : chr  "55" "54" "04" "32" ...
##  $ wkday       : chr  "Friday" "Thursday" "Monday" "Sunday" ...
##  $ month       : num  4 9 6 3 6 10 9 6 10 6 ...
```

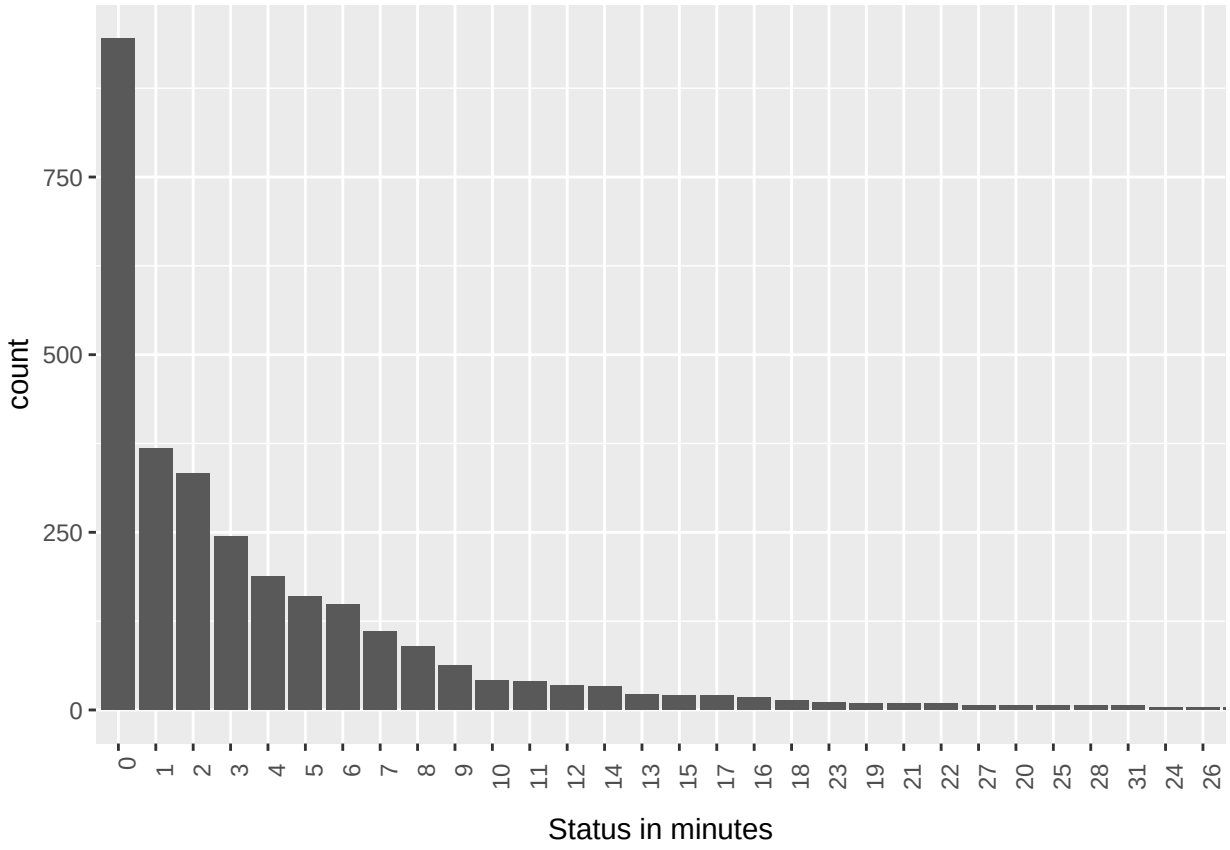
subtitle:

Some Initial plots

GGplot graphs used initally to see trends and relationships within the datasets. ####Status variable chart Status is the name of the dependent variable being predicted in this project. The bar chart shows the frequency of the delays experienced by passengers at the train station when the train is late.

From the chart, we can tell that the trains are on time ~50% of the time and late 50% of the time. In this project, we want to predict when to expect the train to be late and when it will be early to avoid waste of time when possible.

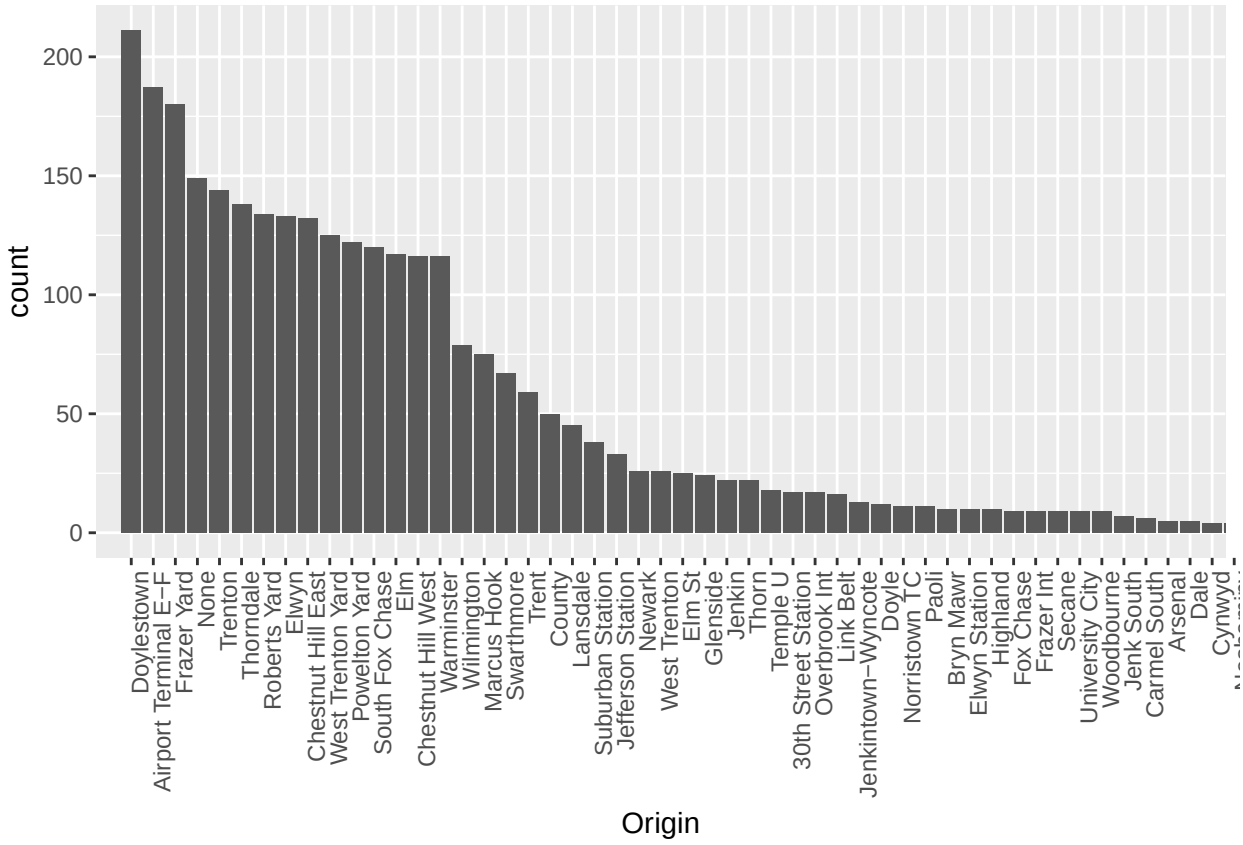
```
library("ggplot2")
#set bar levels in descending order

train_var <- train_data$status
train_data2 <- within(train_data,
                      train_var <- factor(train_var,
                                          levels = names(sort(table(train_var),
                                                              decreasing = TRUE))))
trainstat_graph <- ggplot(train_data2, aes(x = train_var)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1)) + coord_cartesian(xlim = c(1, 30)) + scale_x_dis

trainstat_graph
```



Origin variable bar chart This chart shows the origin which is also the station where each trip begins.

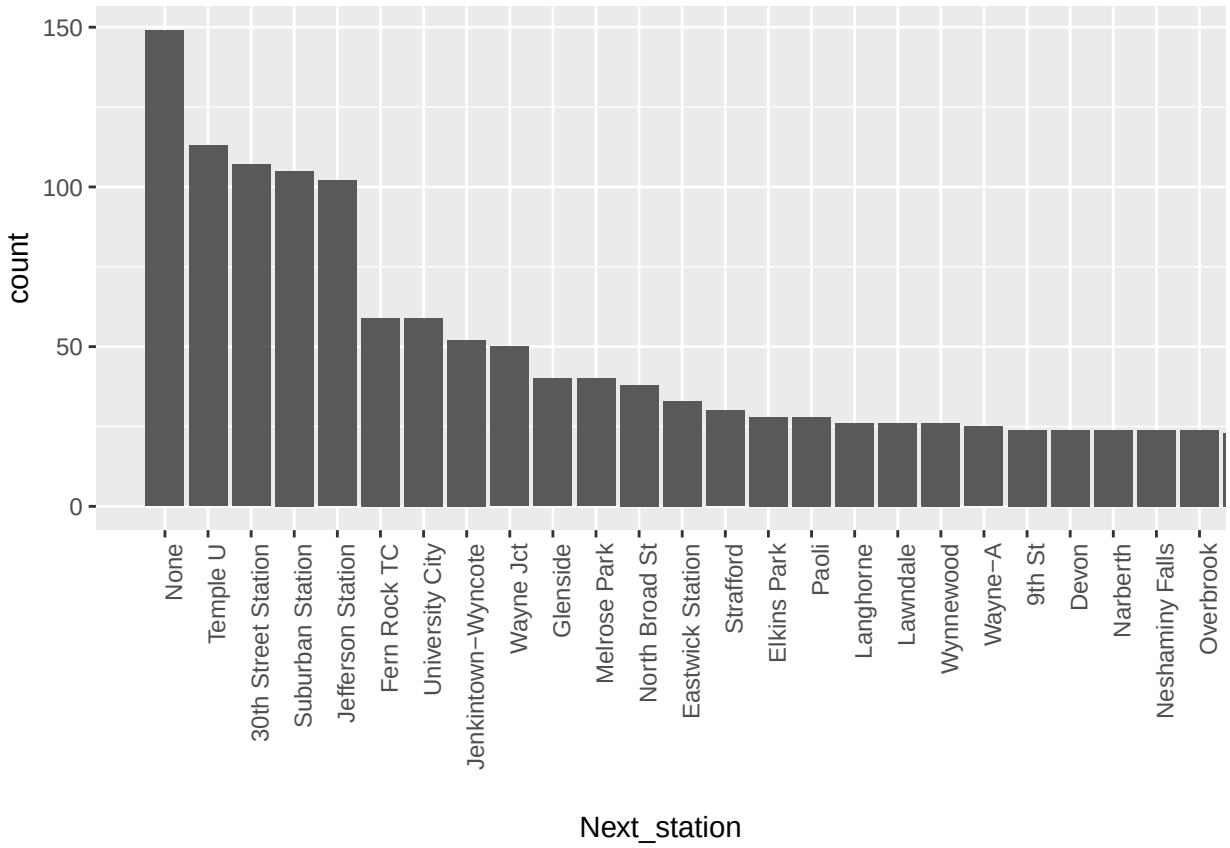
```
train_var <- train_data$origin

train_data2 <- within(train_data,
                      train_var <- factor(train_var,
                                          levels = names(sort(table(train_var),
```

```
                                                    decreasing = TRUE)), ordered = TRUE))

trainorig_graph <- ggplot(train_data2, aes(x =train_var)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1)) + coord_cartesian(xlim = c(0, 50)) + scale_x_dis


trainorig_graph
```



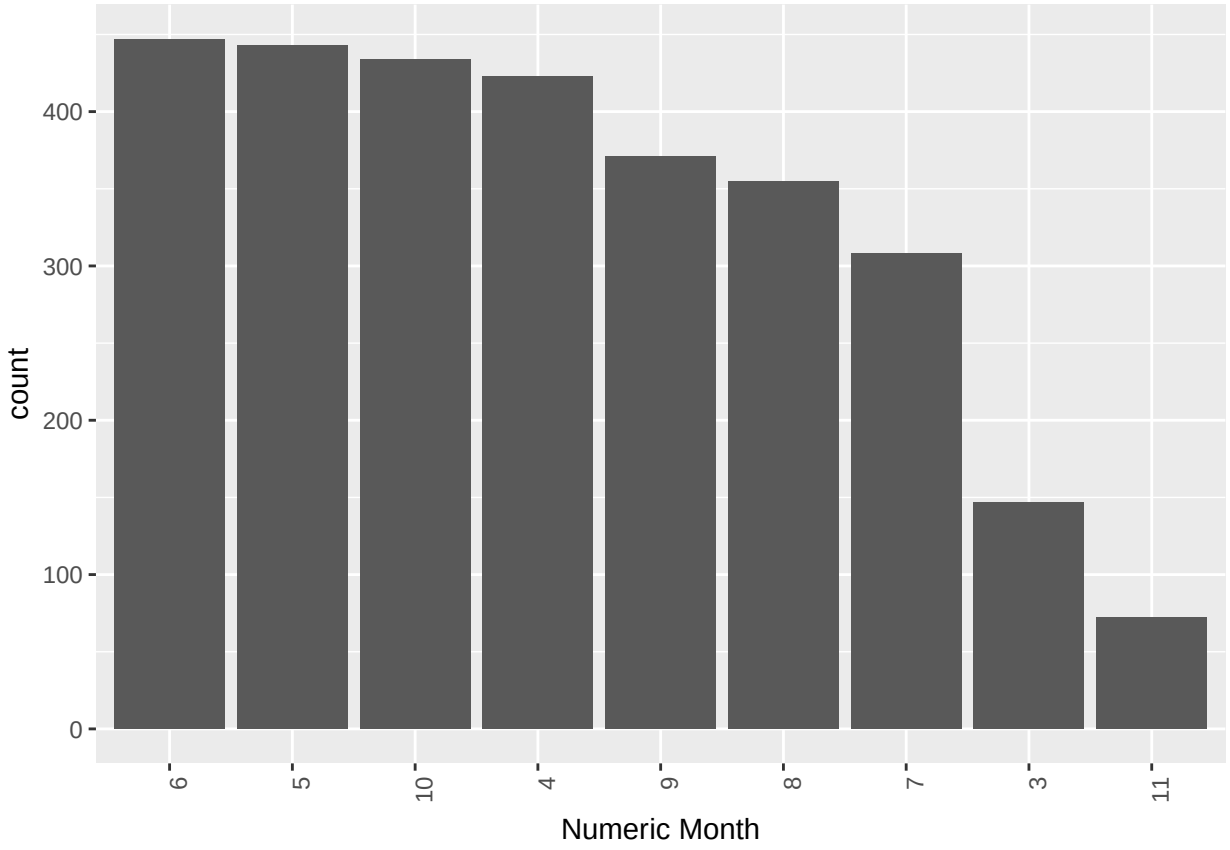
Next Station variable bar chart NextStation variable represents destination for each train ride.

```
train_var <- train_data$next_station


train_data2 <- within(train_data,
                      train_var <- factor(train_var,
                                          levels = names(sort(table(train_var),
                                                              decreasing = TRUE)), ordered = TRUE))

trainnext_graph <- ggplot(train_data2, aes(x = train_var)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1)) + coord_cartesian(xlim = c(0, 25))+ scale_x_dis

trainnext_graph
```



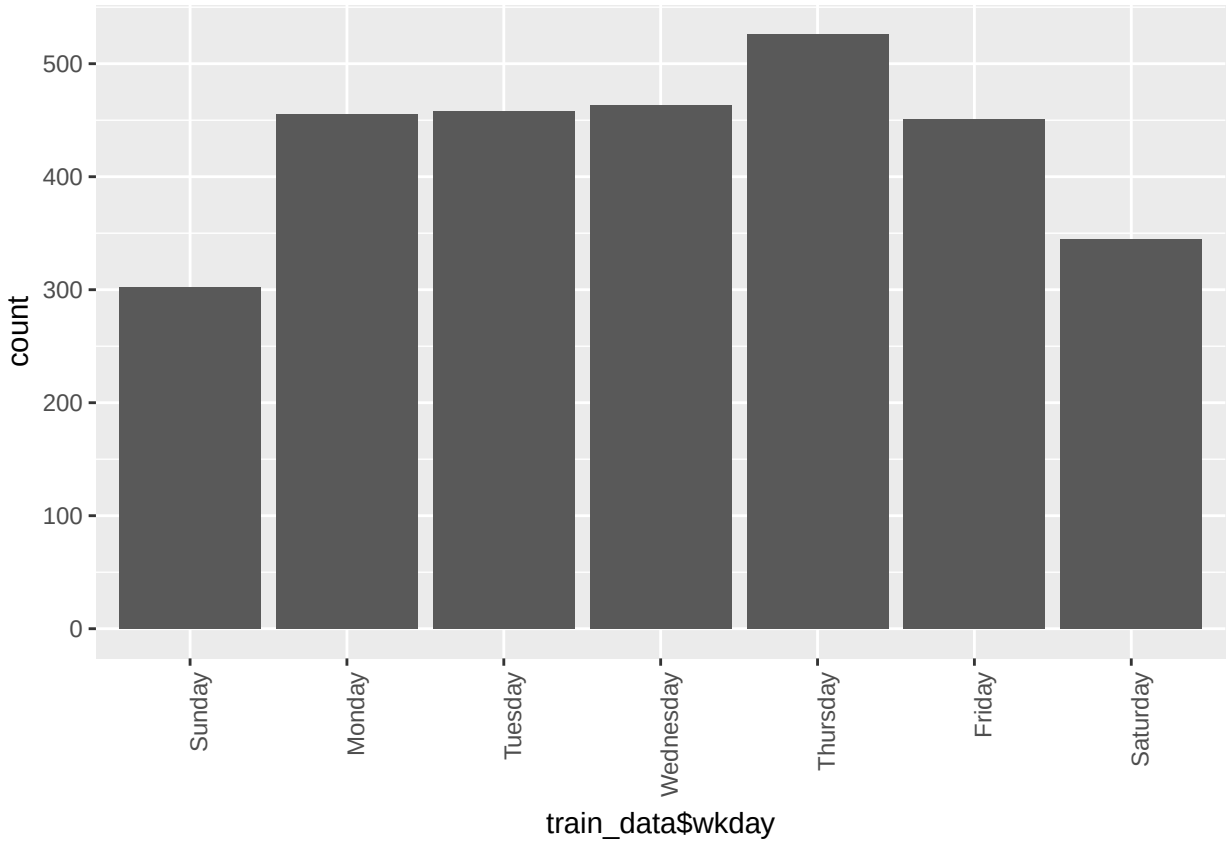
Month bar chart This is one of the new variables improvised by splitting up the timestamp variable.

```
train_var <- train_data$month


train_data2 <- within(train_data,
                      train_var <- factor(train_var,
                                          levels = names(sort(table(train_var),
                                                              decreasing = TRUE)), ordered = TRUE))
trainmonth_graph <- ggplot(train_data2, aes(x = train_var)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1))+ scale_x_discrete(name = "Numeric Month")

trainmonth_graph
```



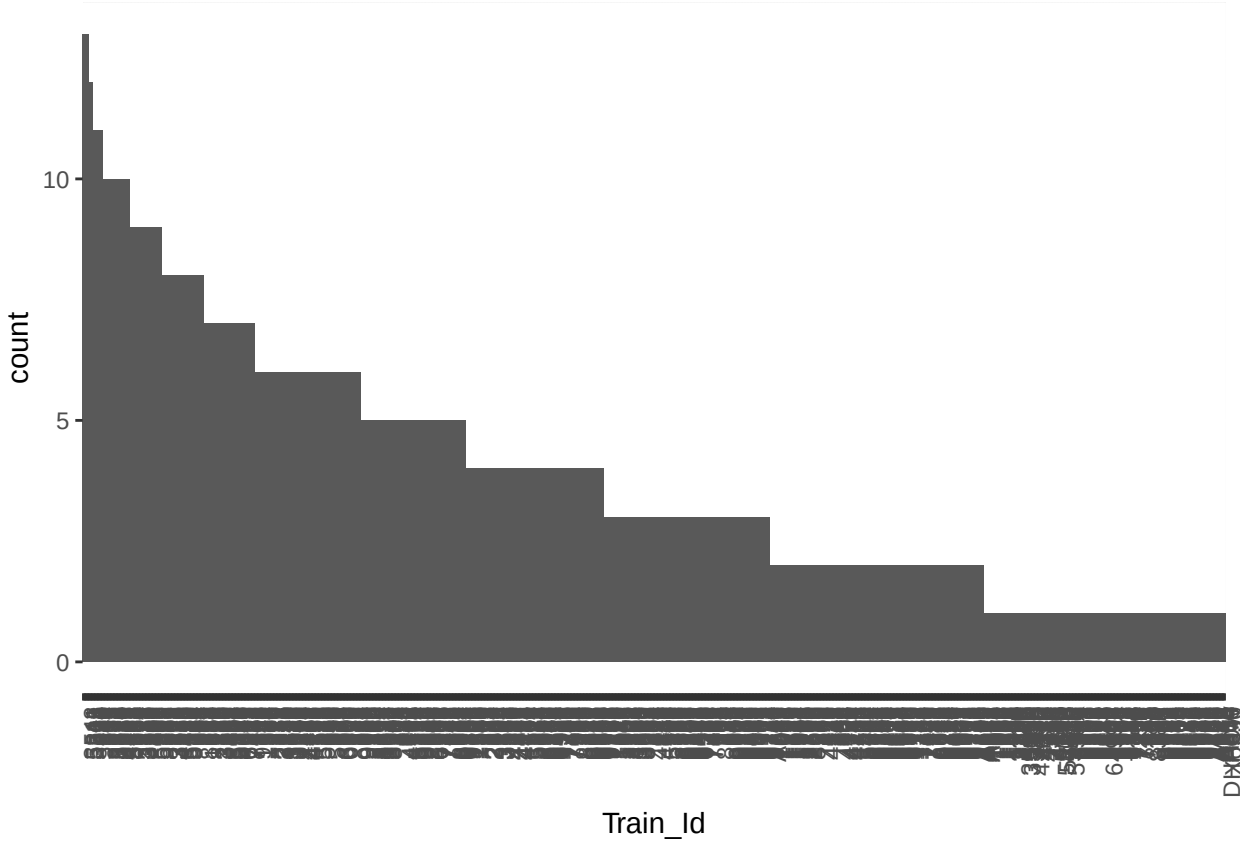
The Weekday chart The weekday chart shows the number of trains that run different days of the week. More trains run during the week and fewer trains on the weekends. The busiest day is Thursday.

```
train_data2 <- within(train_data,
                      train_data$wkday <- factor(train_data$wkday,
                                                 levels = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursda


trainwkday_graph <- ggplot(train_data2, aes(x = train_data$wkday)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1))

trainwkday_graph
```



Train id This is a chart of all the train ids in descending order. These are all the trains that transported passengers during the period of one year in our dataset.

```
train_var <- train_data$train_id


train_data2 <- within(train_data,
                      train_var <- factor(train_var,
                                          levels = names(sort(table(train_var),
                                                              decreasing = TRUE)), ordered = TRUE))

trainid_graph <- ggplot(train_data2, aes(x = train_var)) +
  geom_bar() +
   theme(axis.text.x = element_text(angle=90, hjust=1))+ scale_x_discrete(name = "Train_Id")

trainid_graph
```



subtitle:

More conversion of observations into variables

Convert the weekday and monthday observations to columns. The purpose is to increase the number of variables that contribute to the status (delay and on time arrivals) of the train.

To achieve this, a matrix was used. In this case, the matrix translated the values of the new columns to "0" and "1"."1" was printed when the train travelled in the specified day or month. "0" was used to fill the rest of the observations that the train did not ride during the week day or day of the month. for example, the new column "wkday1" represents "Monday". The value "1" in that column represents the train rides that happened on Mondays.

```
#Added contrasts to print all travel days and all days of the months otherwise some are skipped.

train_data <- cbind(train_data, as.data.frame(model.matrix(~ wkday + monthday, data = train_data, contra
```

subtitle:

Linear Regression model test

Linear model statistical summary for Status based on the independent variable x = origin. The linear regression model is one of the models implemented in this project in efforts to predict the delays of the train.

```
train_data$monthday <- as.numeric(train_data$monthday)

library(broom)

glance(summary(lm(log(as.numeric( train_data$status)+1) ~ origin, data = train_data)))
```

```
##   r.squared adj.r.squared     sigma statistic      p.value df
## 1 0.1760523     0.1514357 0.9109724  7.151781 1.983336e-72 88
```

Added more variables to see the effect on the status of the train, which is the variable we are trying to predict.

```
logvar <- log(as.numeric( train_data$status)+1)
glance(summary(lm(logvar ~ origin+ hour + month + monthday + wkday1 + wkday2 + wkday3 + wkday4 + wkday5
```

```
##   r.squared adj.r.squared     sigma statistic       p.value  df
## 1 0.2417608     0.2112524 0.8782777  7.924399 3.865318e-105 117
```

This is the same regression model with all the 45 variables.

```
glance(summary(lm(as.numeric( train_data$status) ~ origin+ hour + minute + month + wkday1 + wkday2 + wk
```

```
##   r.squared adj.r.squared  sigma statistic   p.value  df
## 1 0.1077999   -0.01275857 19.751 0.8941711 0.9134457 358
```

Next is the linear model chat which was used to get better statistics. To achieve a much better R-value > 24%, all significant independent variables were added including the new ones created by the matrix that were converted from observations to variables.

The original variable count was 11, the addition of the new variables increased the variable count to 58.The improvement of the variables count definitely contributed to a better statistics which increased from .06% to 24%.

```
#Use glance() to print only the statistics and not both statistics and train_data summary



glance(summary(lm(logvar ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday3 + wkday4 + wkday5 +
```

```
##   r.squared adj.r.squared     sigma statistic      p.value  df
## 1 0.3350643     0.2452149 0.8591609  3.729178 1.037887e-83 358
```

subtitle:

CART Model /Decision Tree

:

In this section, the CART model is implemented. The two options considered are Classification and Regression CART models/trees but the reression model is preferred so that the results can be compared with the linear regression model used above. It's like comparing apples to apples or oranges to oranges.

I found this site very helpful because they explained in detail the conditions for the variables before a succesful model can be achieved - https://rstudio-pubs-static.s3.amazonaws.com/27179_e64f0de316fc4f169d6ca300f18ee2aa.html.

Prior to finding this site, only the root or just one circle with a number (4.6) was drawn.

```
library(caTools)
set.seed(3000)


#convert train_data$status to numeric
train_data$status = factor(train_data$status)
train_data$status = as.numeric(train_data$status)

#create split
split = sample.split(train_data$status, SplitRatio = 0.8)

#create training and test set using the subset function

Trainset = subset(train_data, split == TRUE)
Testset = subset(train_data, split == FALSE)

#build CART model

library(rpart)
library(rpart.plot)

#now create the CART model. Use rpart to build a linear regression tree since the status variable being

#use rpart formula to fit the data.

logvar <- log(as.numeric( train_data$status)+1)
TraindataTree2 = rpart(log(Trainset$status +1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday

drpstat <- c("status")
actual <- log(Trainset$status + 1)
actualts <- log(Testset$status + 1)
#since actual is a log of the Trainset$status, also take log of the Testset status
#fitvarUsed <- log(!(names(Testset) %in% drpstat)+1)
#nostatrain <- log(!(names(Trainset) %in% drpstat)+1)

fitvarUsed <- !(names(Testset) %in% drpstat)
#nostatrain <- log(!(names(Trainset) %in% drpstat)+1)

preds <- predict(TraindataTree2, data = Testset[,fitvarUsed])


R2 <- 1 - (sum((actual-preds )^2)/sum((actual-mean(actual))^2))
#print R2
R2
```

```
## [1] 0.3180778
```

The r-squared value is .318 ~ 32%. So far, the CART linear regression model performance is slightly better because the r-squared value is .335 ~ 34%.

subtitle:

Check Performance of CART Regression model

Further test to check the performance of the Training and Test variables. The Trainset data will be trained and performance measured at 5%, 10%, 30%, 50%, 70%, 80% of the observations. A for loop is used to accomplish this in the code.

The result from the plot shows that the Training set performed better than the Testset. The Training set started leveling around 15% of the data used. So no need to use the rest of the data since they don't add more value or improve the performance. The test data on the other hand started leveling at around 10% and the numbers are on the negative side. So only the first 5% contributes to the performance of the Test set.

```
actualtst <- log(Testset$status + 1)

j <- .05 * 3000
#j
R80trn <- 0
R80tst <- 0
for(i in 1:j){
    if(j >=3000){
      #get out of loop once all data has been processed
      break
    }
TraindataTreePerf = rpart(log(Trainset$status +1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wk

#added complete.cases to remove NAs

predstrn <- predict(TraindataTreePerf, data = Trainset[1:j,-c(status)])

predstest <- predict(TraindataTreePerf, data = Testset[1:j,-c(status)])

#the c() is used to add the R2 values into the c() array.

R80trn <- c(R80trn, (1 - (sum((actual-predstrn )^2)/sum((actual-mean(actual))^2))))

R80tst <- c(R80tst, (1 - (sum((actualtst - predstest )^2)/sum((actualtst-mean(actualtst))^2))))

j <- j+j

}
```

First is a check on the Test data.

```
## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length
```

```
## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length
```

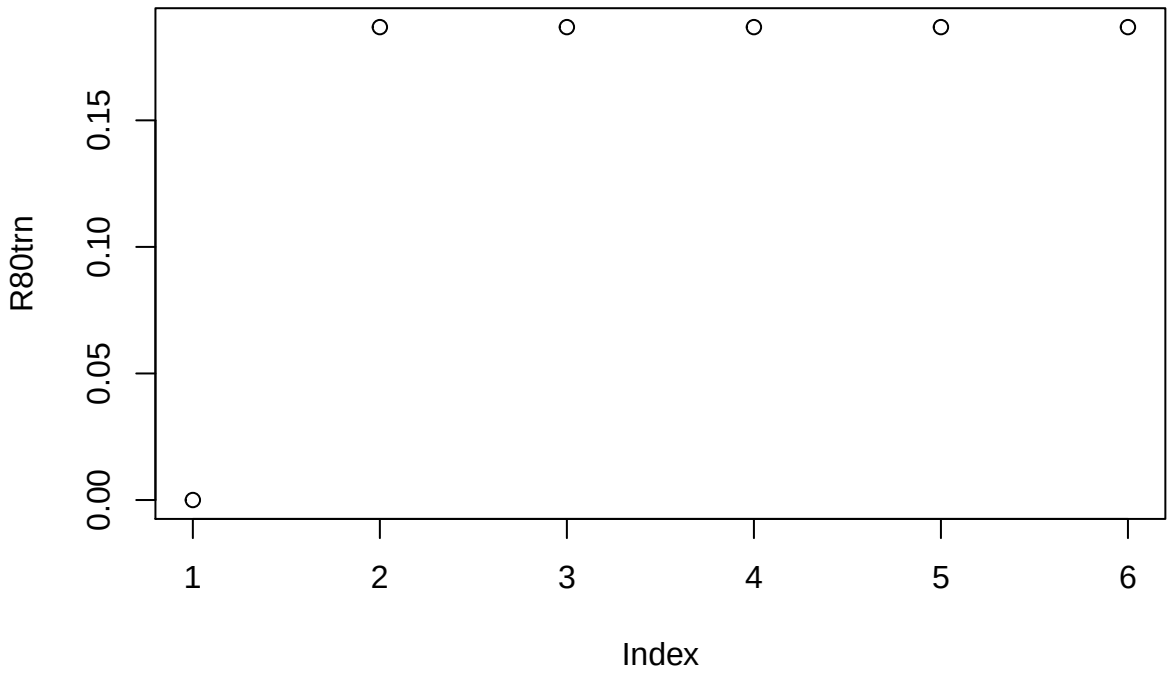
```
#r-squared for CART Training set
R80trn
```

```
## [1] 0.0000000 0.1868266 0.1868266 0.1868266 0.1868266 0.1868266
```

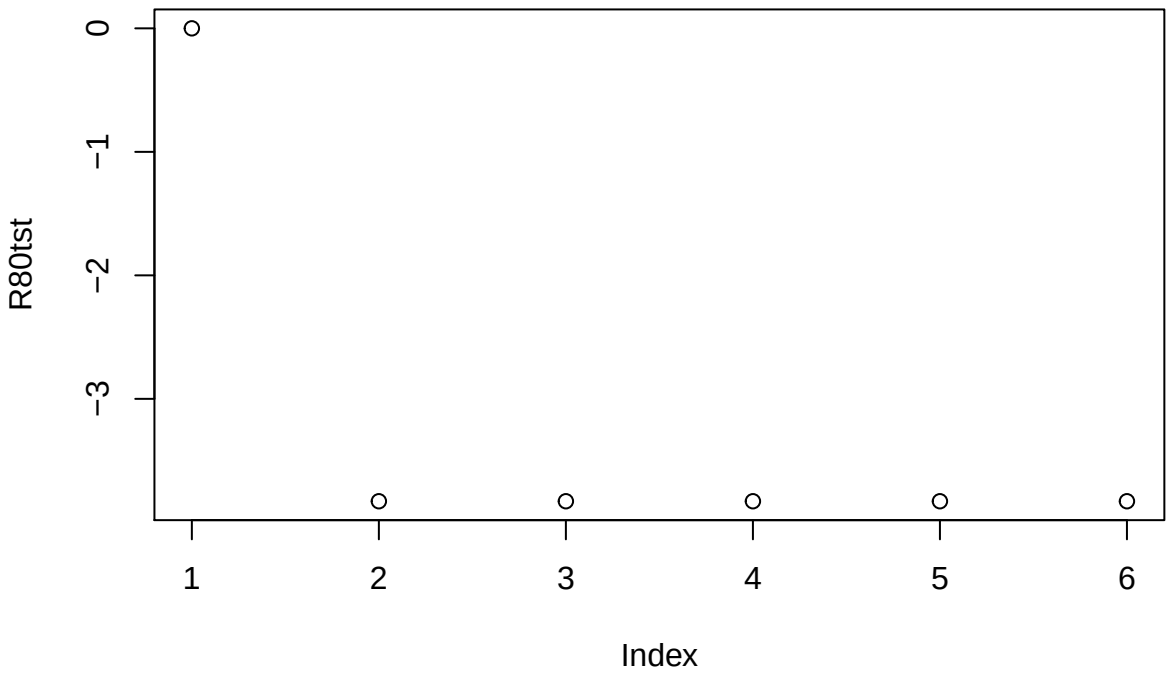
```
#r-squared for CART Testing set
R80tst
```

```
## [1]  0.000000 -3.829003 -3.829003 -3.829003 -3.829003 -3.829003
```

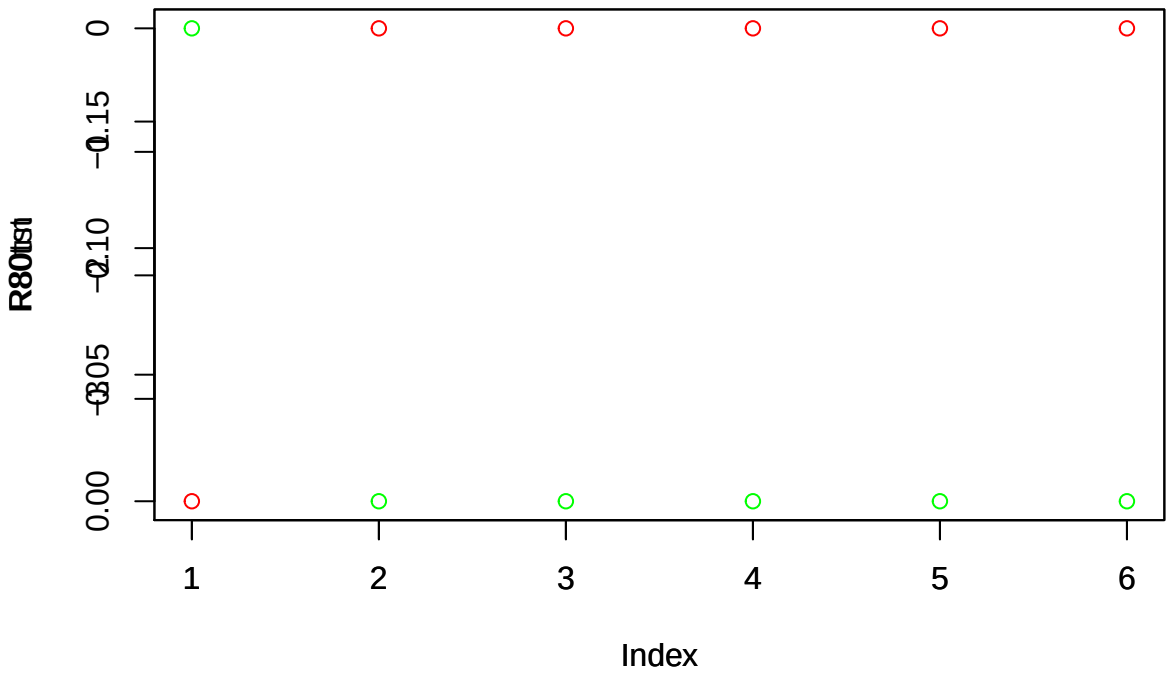
```
plot(R80trn)
```




```
plot(R80tst)
```




```
plot(R80trn, col="red")
par(new=TRUE)
plot(R80tst, col="green")
```



subtitle:

Checking max depth on the Test set for the CART models

The depths start leveling around the sixth level. There is no need to increase the depth more than the sixth level.

```
#R2ts <- numeric()
R2ts <- 0
for (iv in 1:10){

TestsetTree100p = rpart(log(Trainset$status +1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkda


preds <- predict(TestsetTree100p, data = Testset[,-c(status)])

R2ts <- c(R2ts, (1 - (sum((actualtst-preds )^2)/sum((actualtst-mean(actualtst))^2))))

}
```

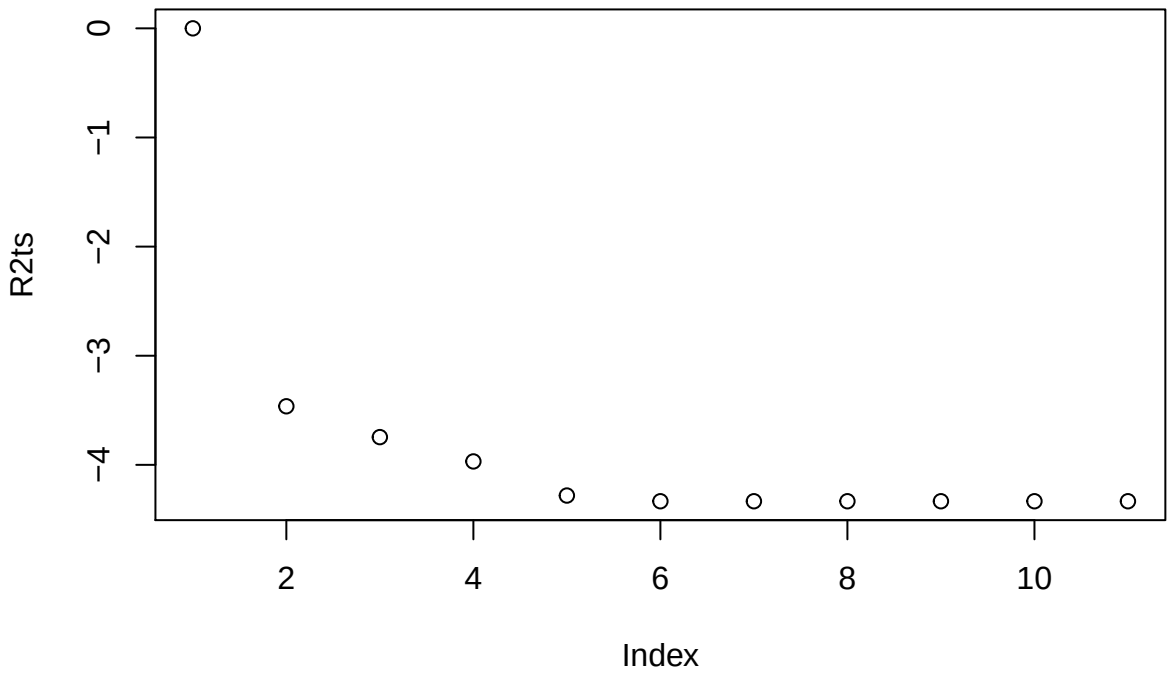
```
## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
```

```
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - preds: longer object length is not a multiple of
## shorter object length
```

```
#list the r-squared values at different depths
R2ts
```

```
##  [1]  0.000000 -3.463245 -3.745840 -3.968569 -4.281177 -4.333407 -4.333407
##  [8] -4.333407 -4.333407 -4.333407 -4.333407
```

```
plot(R2ts)
```



subtitle:

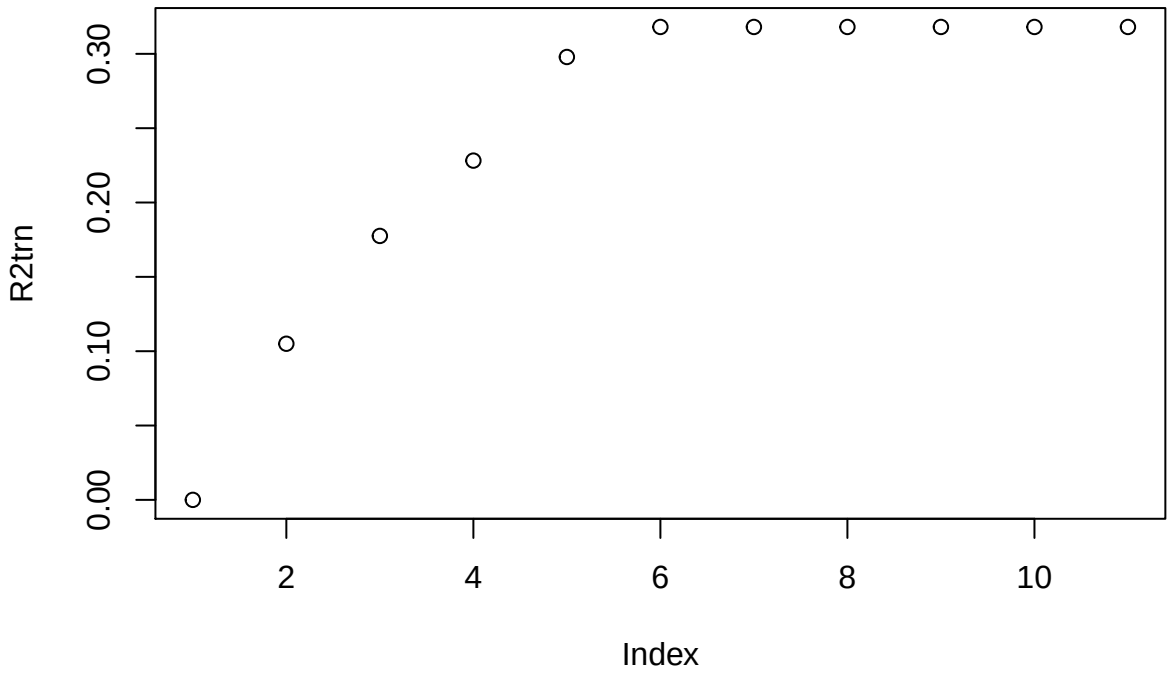
Checking max depth on the Training set for the CART models

Testing max depth on training set. The result from the plot shows that the best performance is reached at max_depth = 5. There is no need to grow a tree with any depth greater than 5.

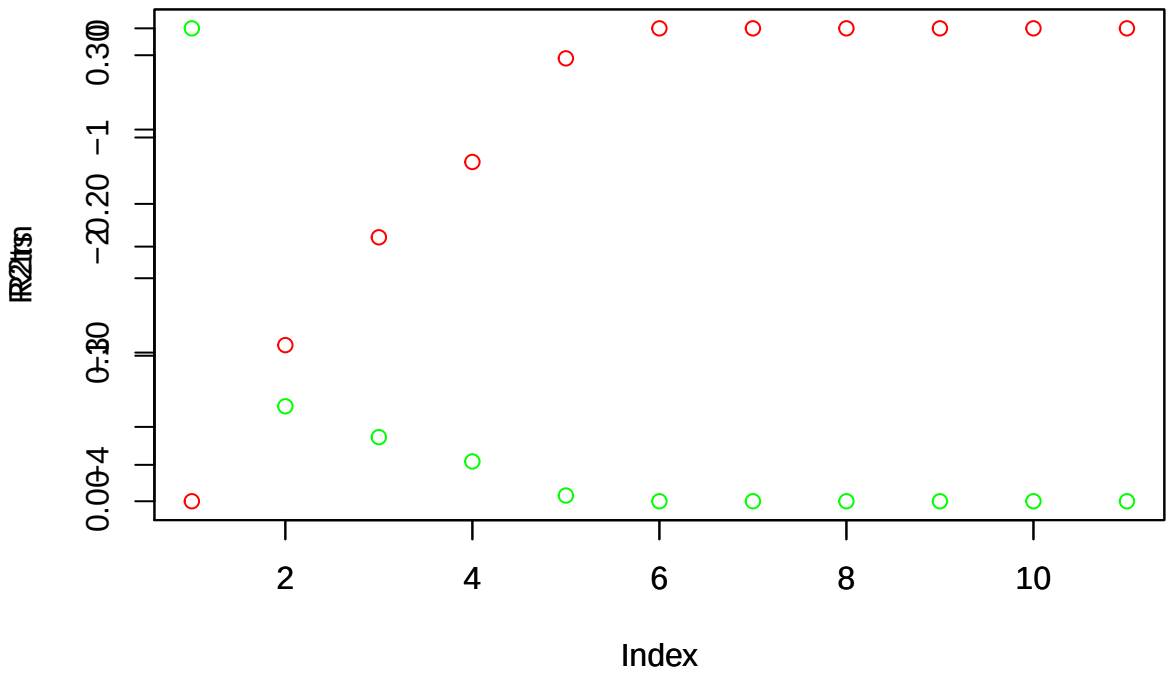
```
#R2trn <- numeric()
R2trn <- 0
for (ig in 1:10){

TraindataTree2 = rpart(log(Trainset$status +1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday


preds <- predict(TraindataTree2, data = Trainset[,-c(status)])

R2trn <- c(R2trn, (1 - (sum((actual-preds )^2)/sum((actual-mean(actual))^2))))

}

#r-squared for Training set at different maximum depths, calculated from depth 1 to 10.

R2trn
```

```
##  [1] 0.0000000 0.1050171 0.1775327 0.2281915 0.2978866 0.3180778 0.3180778
##  [8] 0.3180778 0.3180778 0.3180778 0.3180778
```

```
plot(R2trn)
```




```
#plot both maxdepth graphs in one chart.
plot(R2trn, col="red")
par(new=TRUE)

plot(R2ts, col = "green")
```



subtitle:

Random Forest model

Random Forest model implementation. This is the fourth model used in this project to try to come up with a reasonable prediction for the time delay prediction for the train dataset. The result at the end is similar to the Linear regression Cart model.

```
set.seed(123)
## features for model

splitlistings <- train_data[c("status", "origin", "hour", "minute", "month", "wkday1", "wkday2","wkday3

library(caret)
```

```
## Loading required package: lattice
```

```
library(randomForest)
```

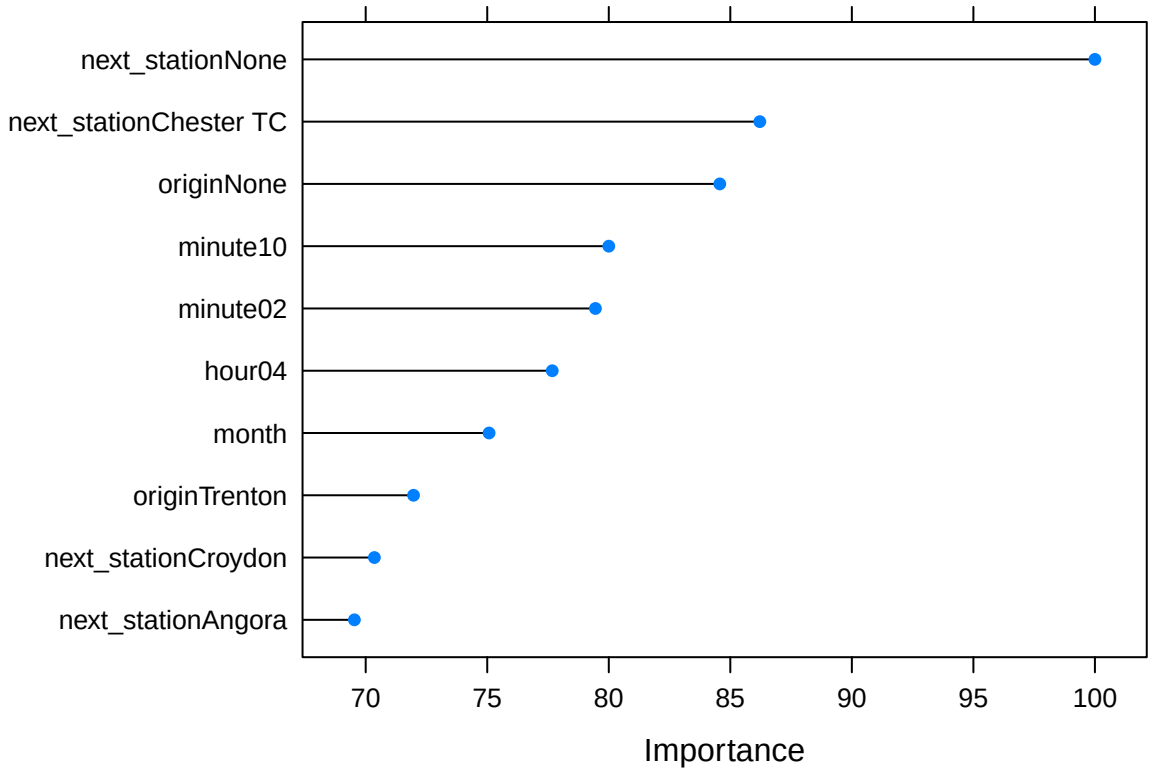
```
## randomForest 4.6-12
```

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

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

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

```
inTraining <- createDataPartition(splitlistings$status, p = .8, list = FALSE)
# save the training and testing sets as data frames
train_1 <- train_data[ inTraining,]
test_1 <- train_data[-inTraining,]


 # fit the randomforest model
model <- train(log(status + 1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday3 + wkday4 + wk

# what are the important variables (via permutation)
vi <- varImp(model, type=1)
plot(vi, top=10)
```




```
# predict the outcome of the training data
predicted_tr <- predict(model, newdata=train_1, select = -c(status))
actual_tr <- log(train_1$status + 1)
rsq_tr <- 1-sum((actual_tr-predicted_tr)^2)/sum((actual_tr-mean(actual_tr))^2)
rsq_tr #print rsq training value.
```

```
## [1] 0.2142874
```

```
# predict the outcome of the testing data
predicted <- predict(model, newdata=test_1, select = -c(status))
actual_ts <- log(test_1$status + 1)

rsqd <- 1-sum((actual_ts-predicted)^2)/sum((actual_ts-mean(actual_ts))^2)

rsqd #r-squared of the Test set.
```

```
## [1] 0.1014539
```

subtitle:

Check performance on the RandomForest Training and Testing data

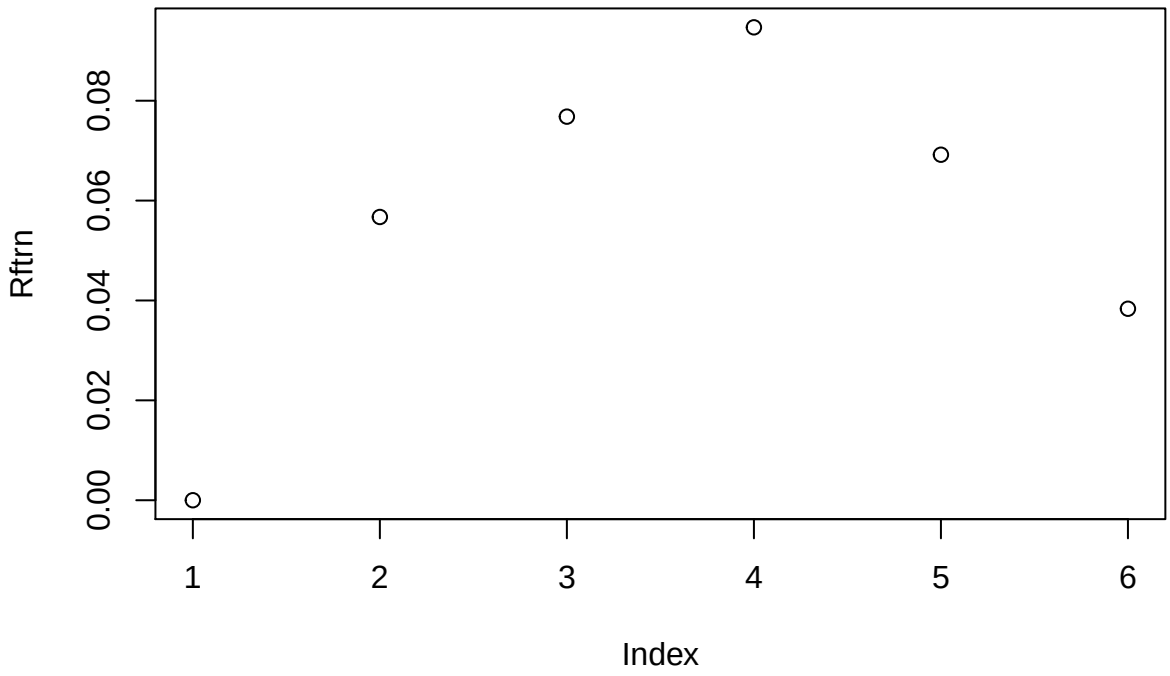
The plots show that the Training set performed better than the Test set.

```
actual_tsmx <- log(test_1$status + 1)


z <- .05 * 3000
#j
Rftrn <- 0
Rftst <- 0
for(b in 1:z){
    if(z >=3000){
      #get out of loop once all data has been processed
      break
    }

 # fit the randomforest model
modelx <- train(log(status + 1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday3 + wkday4 + wk

#added complete.cases to remove NAs

predstrn <- predict(modelx, data = train_1[1:z,-c(status)])

predstest <- predict(modelx, data = test_1[1:z,-c(status)])

#calculate the R2 values for the training and testing dataset
Rftrn <- c(Rftrn, (1 - (sum((actual_tr-predstrn )^2)/sum((actual_tr-mean(actual_tr))^2))))

Rftst <- c(Rftst, (1 - (sum((actual_tsmx - predstest )^2)/sum((actual_tsmx-mean(actual_tsmx))^2))))



z <- z+z

}
```

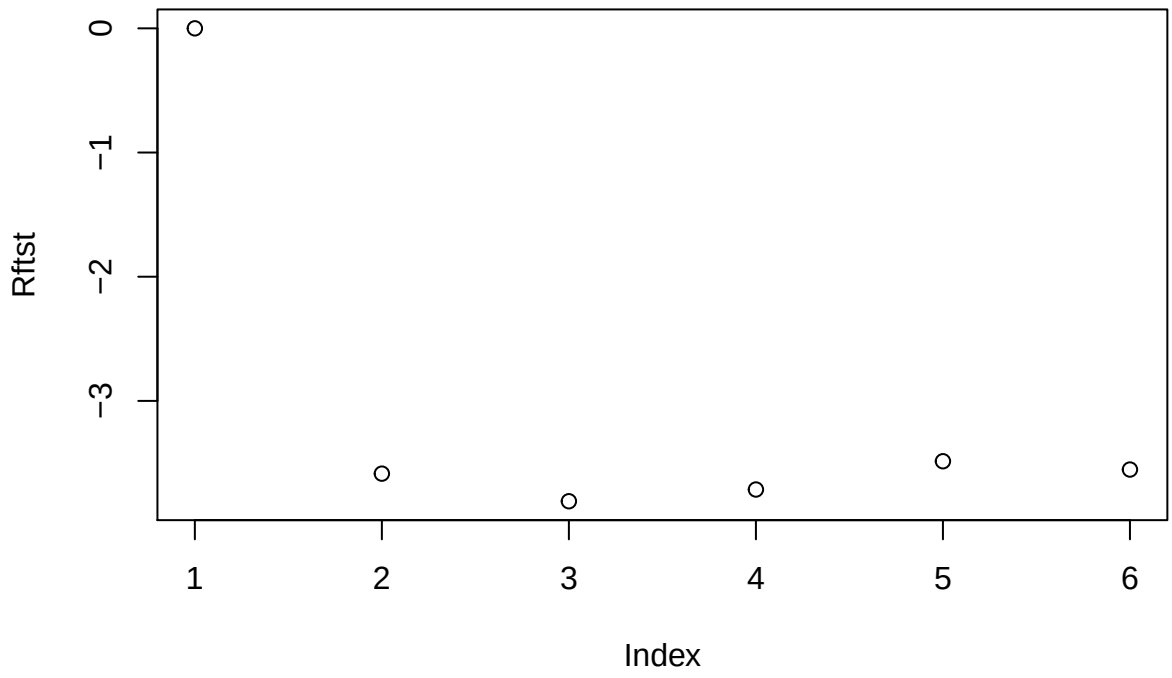
```
## Warning in actual_tsmx - predstest: longer object length is not a multiple
## of shorter object length

## Warning in actual_tsmx - predstest: longer object length is not a multiple
```

```
## of shorter object length

## Warning in actual_tsmx - predstest: longer object length is not a multiple
## of shorter object length

## Warning in actual_tsmx - predstest: longer object length is not a multiple
## of shorter object length

## Warning in actual_tsmx - predstest: longer object length is not a multiple
## of shorter object length
```

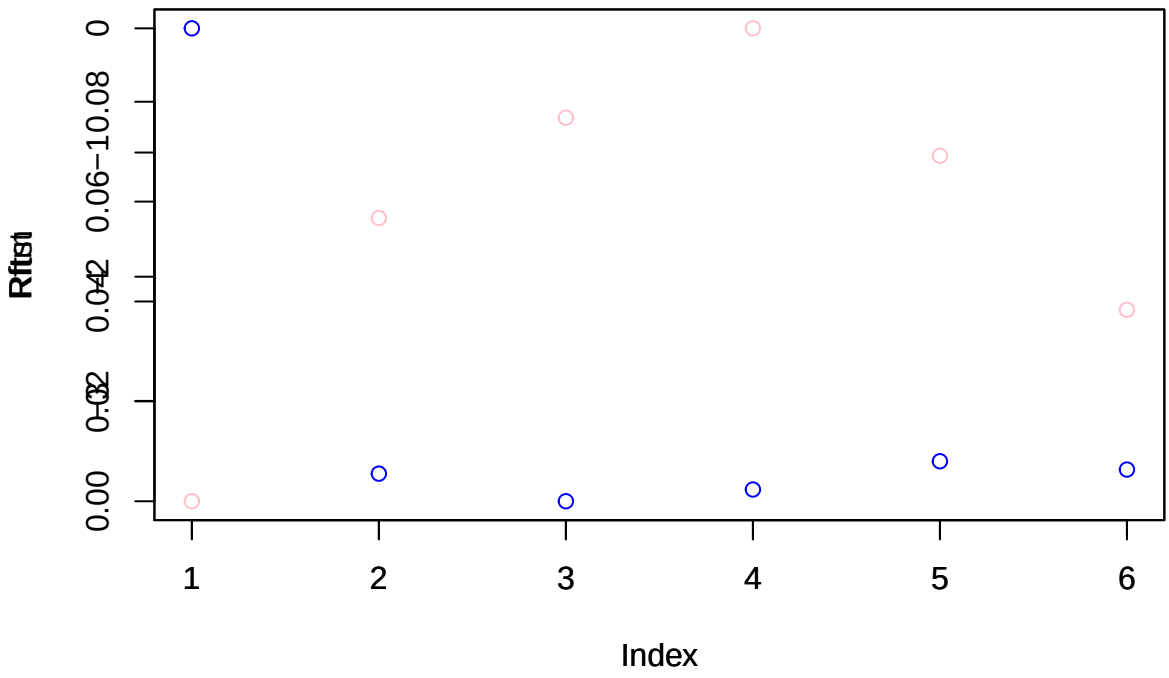
```
plot(Rftrn)
```




```
plot(Rftst)
```




```
plot(Rftrn, col="pink")
par(new=TRUE)
plot(Rftst, col="blue")
```




```
#Print the r-squared values for the Random forest training and testing dataset.
Rftrn
```

```
## [1] 0.00000000 0.05671883 0.07681142 0.09469131 0.06918366 0.03834287
```

```
Rftst
```

```
## [1]  0.000000 -3.585752 -3.807936 -3.713547 -3.485904 -3.552617
```

Check r-squared for Training and Test data for Regression model

```
#actual_tsmx <- log(test_1$status + 1)


x <- .05 * 3000
#j
Rftrn <- 0
Rftst <- 0
for(b in 1:x){
    if(x >=3000){
      #get out of loop once all data has been processed
      break
    }
```

```
TraindataLM <- lm(log(Trainset$status +1) ~ origin+ hour + minute + month + wkday1 + wkday2 + wkday3 + w


predstrn <- predict(TraindataLM, data = Trainset[1:x,-c(status)])

predstest <- predict(TraindataLM, data = Testset[1:x,-c(status)])

#the c() is used to add the R2 values into the c() array.

R2lmtrn <- c(R80trn, (1 - (sum((actual-predstrn )^2)/sum((actual-mean(actual))^2))))

R2lmtst <- c(R80tst, (1 - (sum((actualtst - predstest )^2)/sum((actualtst-mean(actualtst))^2))))

x <- x+x

}
```

```
## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length

## Warning in actualtst - predstest: longer object length is not a multiple of
## shorter object length
```

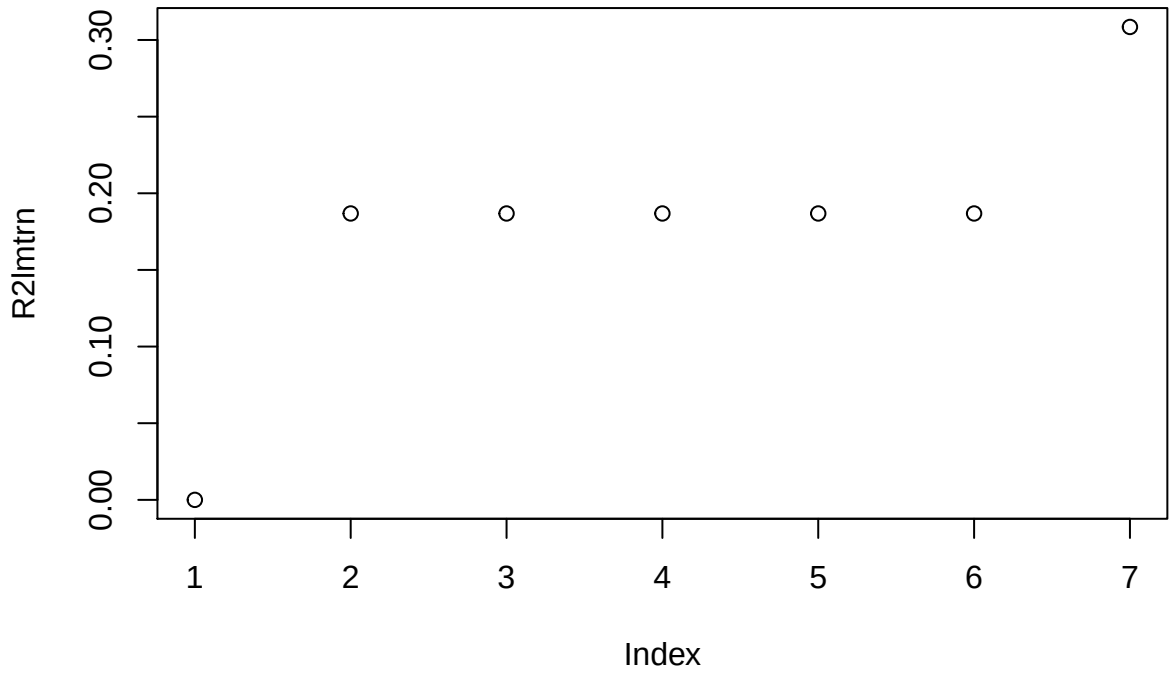
```
#list r-squared for Linear Regression Training and Test data
R2lmtrn
```

```
## [1] 0.0000000 0.1868266 0.1868266 0.1868266 0.1868266 0.1868266 0.3084889
```

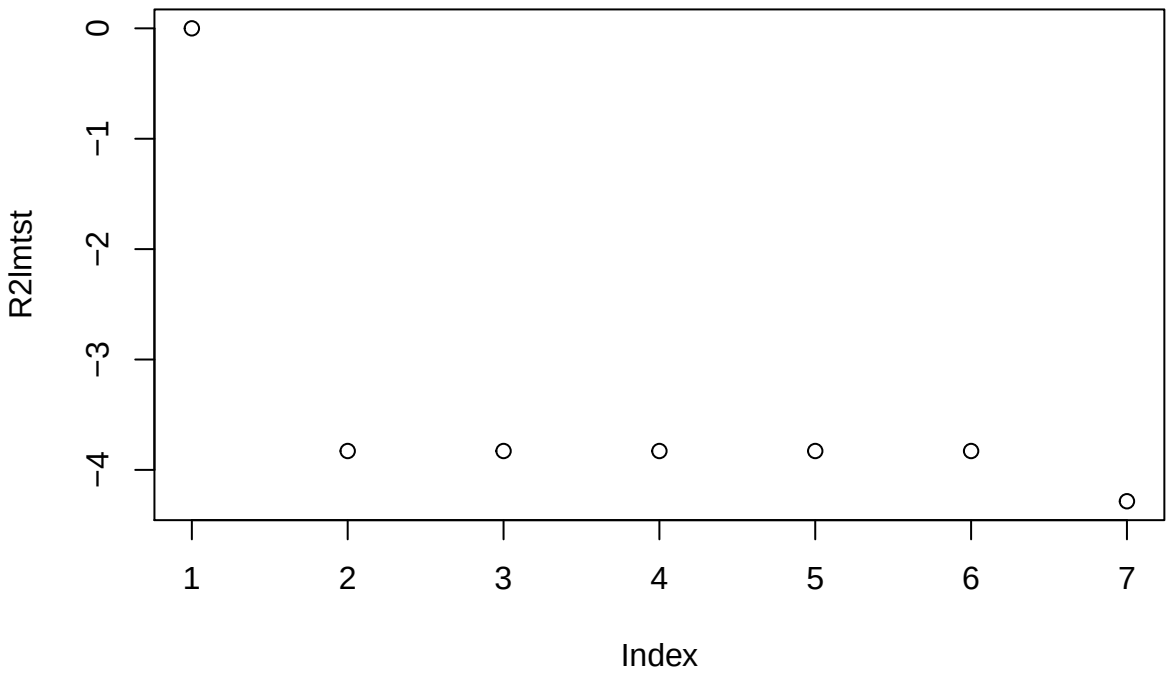
```
R2lmtst
```

```
## [1]  0.000000 -3.829003 -3.829003 -3.829003 -3.829003 -3.829003 -4.284031
```

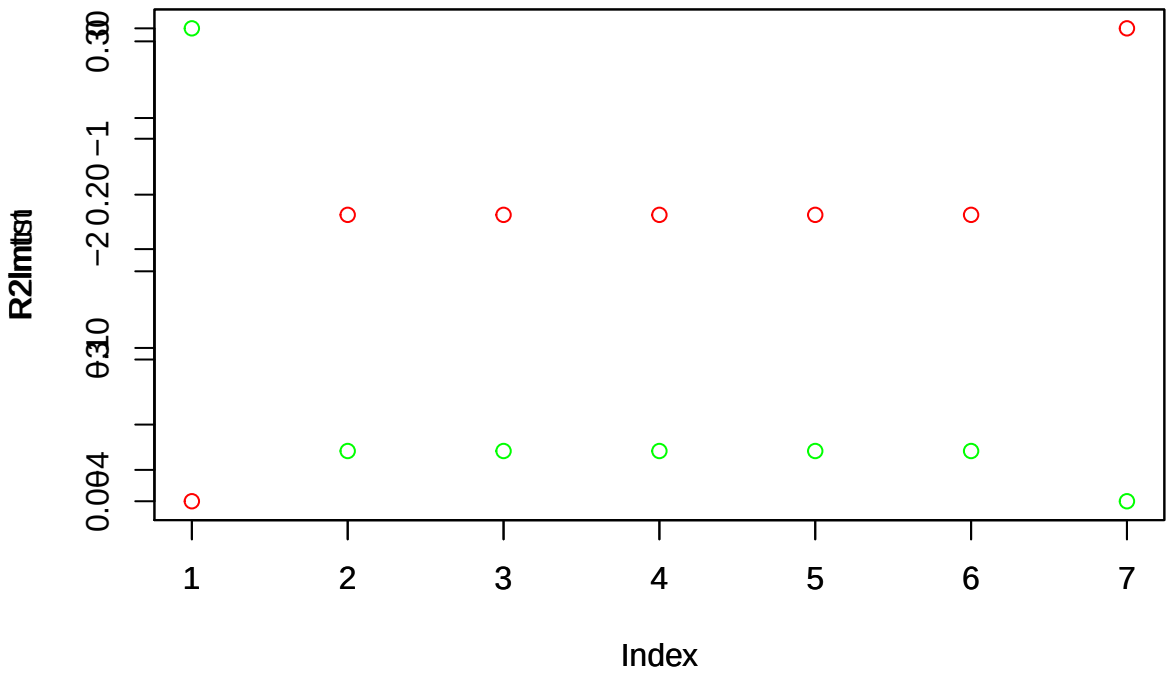
```
plot(R2lmtrn)
```




```
plot(R2lmtst)
```




```
plot(R2lmtrn, col="red")
par(new=TRUE)
plot(R2lmtst, col="green")
```



subtitle:

Conclusion from the models

The r-squared value from the Linear Regression model is 0.33506 ~34%. The Training set performance (r-squared) is .308 ~31%. The Testing set r-squared is 0.

The r-squared value from CART Regression model/ Decision Tree is .318 ~ 32%. The Training set best performance is 18% while the Test performance is 0.

The r-squared value from the Random Forest Training model at ntree=22 is .214 ~ 21% and as expected, the performance on the Training set is better than the performance on the Testing set.The r-squared value from the Random Forest Testing model is .101~ 10%. The word document named "finalproject_randomforest.docx" lists the different r-squared values at ntree values = 15, 10, 5 and 1. At the ntree value of 5, the r-squared or performance is 21% which is slightly better than the performance at ntree = 22 which is .202 ~ 20%.

The model that gave us the best performance is the Linear Regression model because it's R-squared is slightly higher than the CART regression and Random Forest models. Rhe Testing set performance is 31%, which is higher than the training performance of the other models.