Project 2 - regression

Data cleaning and exploration

The first data set I decided to use was a Formula 1 dataset through Ergast, retrieved from here: http://ergast.com/mrd/db/

Ultimately, what I would like my model to do for this data set is predict a given lap's lap time based on total race times, final position, the track, position of driver in the race at time of lap, nationality, the year in which the race took place, constructor, driver age, whether it is the first lap, the driver's fastest lap time for that race, the interaction between final time and track, and whether or not the driver took a pit stop that lap. Obviously, the predictors will be different for different models; for instance kNN works better with fewer dimensions.

The first step in predicting lap times is cleaning up the data. The fields themselves were incredibly messy and some had characters which were not reading properly. The first thing I did was write a python script to clean up certain names and to make all of the codes match so that they can be appropriately divided into factors later in the process.

The data in the database was extensive, but spread throughout 13 different files of different formats, so getting relevant variables into a single data frame was the next step.

To get the relevant data into a single data frame, I first read in all of the CSV files which contained the data I will need to predict on, and to name them appropriately according to the database schema.

```
laptimes <- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f1db_
races <- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f1db_csv
driver <- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f1db_cs
pitstops <- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f1db_
constructors <- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f
race_results<- read.csv("C:/Users/Keaton/OneDrive/Documents/School/SP2019/Machine Learning/Project 2/f1
```

```
names(laptimes) <- c("raceId", "driverId", "lap_number", "position", "lap_time", "lap_milliseconds")
names(races) <- c("raceId", "year", "round", "circuitId", "race name", "date", "start time", "url1")
names(driver) <- c("driverId", "driverRef", "number", "code", "forename", "surname", "dob", "nationalit
names(pitstops) <- c("raceId", "driverId", "stop", "lap_number", "timeOfStop", "pit_duration", "pit_mil
names(constructors) <- c("constructorId","constructorRef","constructor_name", "constructor_nationality"
names(race_results) <- c("resultId","raceId","driverId","constructorId","number","grid","final_position
```

The files are all organized into structures based on database schemas. Let's take a look at the laptimes dataframe which we will eventually use to make our prediction.

```
dim(laptimes)
```

```
## [1] 450998      6
```

```
head(laptimes)
```

```
##   raceId driverId lap_number position lap_time lap_milliseconds
## 1    841       20          1        1 1:38.109            98109
## 2    841       20          2        1 1:33.006            93006
## 3    841       20          3        1 1:32.713            92713
## 4    841       20          4        1 1:32.803            92803
## 5    841       20          5        1 1:32.342            92342
## 6    841       20          6        1 1:32.605            92605
```

So laptimes is 450998 observations with 6 variables, but the table doesn't contain many of the factors I want to use to predict lap times. For instance, there is no information on pitstops or constructors here, or any information telling us which year the lap took place in. Let's take a look at the races dataframe.

```
dim(races)
```

```
## [1] 1018    8
```

```
head(races)
```

```
##   raceId year round circuitId             race name       date start time
## 1      1 2009     1         1 Australian Grand Prix 2009-03-29   06:00:00
## 2      2 2009     2         2  Malaysian Grand Prix 2009-04-05   09:00:00
## 3      3 2009     3        17    Chinese Grand Prix 2009-04-19   07:00:00
## 4      4 2009     4         3    Bahrain Grand Prix 2009-04-26   12:00:00
## 5      5 2009     5         4    Spanish Grand Prix 2009-05-10   12:00:00
## 6      6 2009     6         6     Monaco Grand Prix 2009-05-24   12:00:00
##                                                 url1
## 1 http://en.wikipedia.org/wiki/2009_Australian_Grand_Prix
## 2  http://en.wikipedia.org/wiki/2009_Malaysian_Grand_Prix
## 3    http://en.wikipedia.org/wiki/2009_Chinese_Grand_Prix
## 4    http://en.wikipedia.org/wiki/2009_Bahrain_Grand_Prix
## 5    http://en.wikipedia.org/wiki/2009_Spanish_Grand_Prix
## 6     http://en.wikipedia.org/wiki/2009_Monaco_Grand_Prix
```

Here we have a list of the individual races. You'll notice that this table is much smaller; only 1018 observations, or 1018 races. Now our challenge is to merge these tables in a way where the laptimes dataframe contains the year and other relevant info for every single row in laptimes so that we can use those variables in our predictions. Mindlessly merging tables won't get us the result we want, however. For instance, this is further complicated by the fact that a driver can drive for multiple constructors over the course of his career. Special care needs to be taken to preserve the data we need.

The solution to our problem here is to use R's merge() function. What this function does is merge tables on certain key attributes. In the case of the laptimes and races tables, the common variable would be raceId. What merge would do in this case is add all of the columns of races to a row in laptimes where the raceIds match. It is functionally equivalent to a join in database systems. In this code chunk, I'll merge all of the data we need into laptimes so that I can begin to operate on it, and remove some columns we won't be using.

```
constructors[c(2, 4, 5)] <- list(NULL)
race_results[c(1, 5, 6, 8:12, 14:18)] <- list(NULL)

laptimes <- merge(laptimes, driver, by="driverId")
laptimes <- merge(laptimes, races, by="raceId")
laptimes <- merge(laptimes, pitstops, all.x=TRUE)
laptimes <- merge(laptimes, race_results, by=c("raceId","driverId"))
laptimes <- merge(laptimes, constructors)

laptimes[c(8:12, 15, 17, 19, 20:24, 29)] <- list(NULL)
```

Now that everything has been merged, let's take another look at laptimes.

```r
dim(laptimes)
```

```
## [1] 450998     15
```

```r
head(laptimes)
```

```
##   constructorId raceId driverId lap_number position lap_time
## 1             1      1        1          1       13 1:49.088
## 2             1      1        1          2       12 1:33.740
## 3             1      1        1          3       11 1:31.600
## 4             1      1        1          4       10 1:31.067
## 5             1      1        1          5       10 1:32.129
## 6             1      1        1          6        9 1:30.469
##   lap_milliseconds      dob nationality year circuitId pit_duration
## 1           109088 1/7/1985     British 2009         1         <NA>
## 2            93740 1/7/1985     British 2009         1         <NA>
## 3            91600 1/7/1985     British 2009         1         <NA>
## 4            91067 1/7/1985     British 2009         1         <NA>
## 5            92129 1/7/1985     British 2009         1         <NA>
## 6            90469 1/7/1985     British 2009         1         <NA>
##   pit_milliseconds final_position final_milliseconds
## 1               NA            \\N                \\N
## 2               NA            \\N                \\N
## 3               NA            \\N                \\N
## 4               NA            \\N                \\N
## 5               NA            \\N                \\N
## 6               NA            \\N                \\N
```

You can see now that laptimes contains all of the proper variables! This is great, now we can get ready to operate on it. The next step is to designate factors to the variables which have different levels. We'll also add a column called pit_stop which is a factor 0 or 1 based on whether a lap has a pit_duration. Finally, I'm going to add a factor which separates years 3 at a time. We'll use this factor to predict lap times, but I'm hoping that a 3 year factor will help to expand the number of samples for each unit of year that we use in our prediction. It will allow us more data for each track, for instance, since a track only hosts one event per year. I will also clean up the data set a bit more here and get rid of a significant number of rows which do not have data necessary for our prediction.

```r
# convert dob column to just year for simple age calculation
laptimes$dob <- as.Date(laptimes$dob, "%m/%d/%Y")
laptimes$dob <- as.integer(substring(laptimes$dob, 1, 4))

# convert from factor to integer
laptimes$final_milliseconds <- as.integer(as.character(laptimes$final_milliseconds))
```

```
## Warning: NAs introduced by coercion
```

```r
laptimes$final_position <- as.integer(as.character(laptimes$final_position))
```

```
## Warning: NAs introduced by coercion
```

```r
# create new columns for age, and factor columns for whether or not the lap is the first lap or there h
laptimes$age <- (laptimes$year - laptimes$dob)
laptimes$pit_stop <- (ifelse(is.na(laptimes$pit_duration), FALSE, TRUE))
laptimes$first_lap <- (ifelse(laptimes$lap_number==1, TRUE, FALSE))

# add a factor column year_split which contains groups of 3 years, depending on the year the lap was ru
laptimes$year_split <- as.factor(findInterval(laptimes$year, c(1996, 1999, 2002, 2005, 2008, 2011, 2014

laptimes[c(12, 13)] <- list(NULL)

# remove many old laps with incomplete data
laptimes <- na.omit(laptimes)

head(laptimes)
```

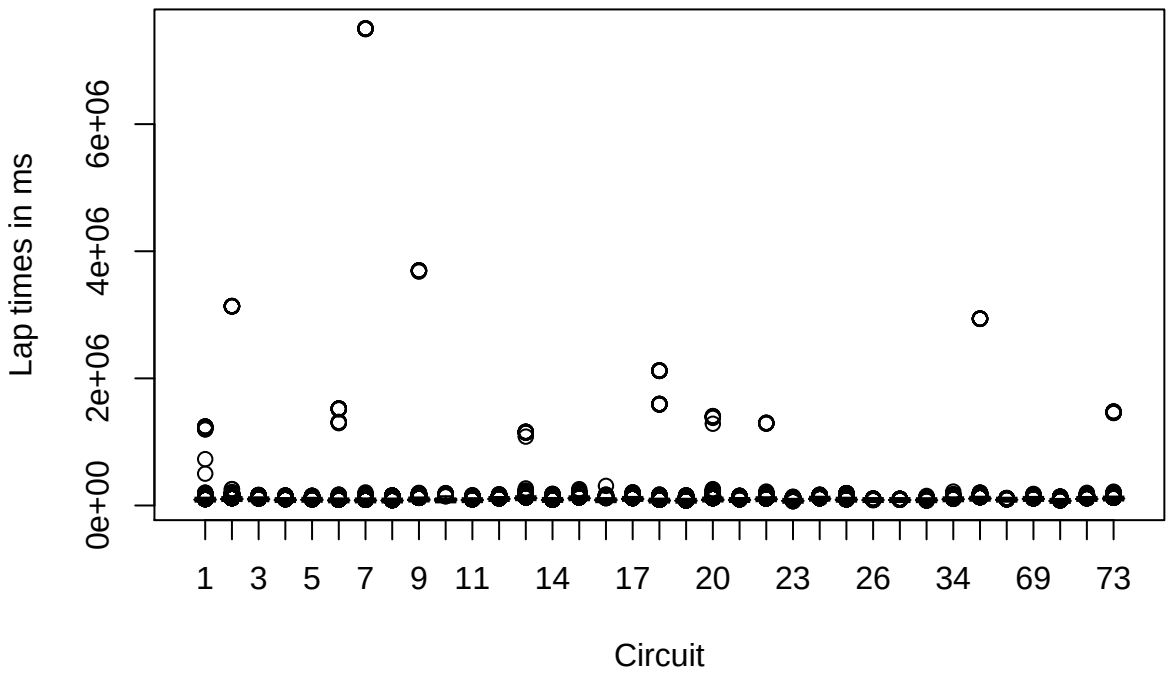
```
##    constructorId raceId driverId lap_number position lap_time
## 90             1    941       18          1       10 2:30.176
## 91             1    941       18          2        9 2:37.119
## 92             1    941       18          3        9 2:28.895
## 93             1    941       18          4       10 1:47.020
## 94             1    941       18          5       10 1:45.861
## 95             1    941       18          6       10 1:45.438
##    lap_milliseconds  dob nationality year circuitId final_position
## 90           150176 1980     British 2015        71              9
## 91           157119 1980     British 2015        71              9
## 92           148895 1980     British 2015        71              9
## 93           107020 1980     British 2015        71              9
## 94           105861 1980     British 2015        71              9
## 95           105438 1980     British 2015        71              9
##    final_milliseconds age pit_stop first_lap year_split
## 90            5910491  35    FALSE      TRUE          7
## 91            5910491  35    FALSE     FALSE          7
## 92            5910491  35    FALSE     FALSE          7
## 93            5910491  35    FALSE     FALSE          7
## 94            5910491  35    FALSE     FALSE          7
## 95            5910491  35    FALSE     FALSE          7
```

The separation of 3 years is important because some VERY impactful changes made their way into Formula 1. For instance, in 2011 FIA introduced DRS, a system which allowed higher speed on straights that drastically reduced lap times. The way we are separating into 3 year chunks accounts for many of these changes.

Another huge step I took in the previous code chunk was removing data from before 1996. This is because lap data was not kept for the years 1950-1995, so many of my predictors were not present in laps that took place in those years. This also cut down significantly on computation time because we removed 200k+ rows. Even with our current number (219406), running knn takes a few minutes on my home computer, so removing these rows has made the dataset much less unwieldy.

Let's take a look at the data through some graphs to see if we can better understand it.

```r
boxplot(laptimes$lap_milliseconds~laptimes$circuitId, xlab="Circuit", ylab="Lap times in ms")
```




```
summary(laptimes$lap_milliseconds)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   66957   82623   92530   96752  103745 7506656
```

We can see here that we have some pretty extreme outliers for each given track. Some laps take up to 7506 seconds! That is highly irregular. No average time for any track throughout history is longer than 4 minutes. I'll start by removing laps which take greater than 5 minutes. This doesn't eliminate all of the laps that the boxplot considers outliers, but I don't want to accidentally eliminate a row which could just be a slow lap.

```
laptimes <- laptimes[-which(laptimes$lap_milliseconds > 250000),]
```

Let's see how that's affected our summary.

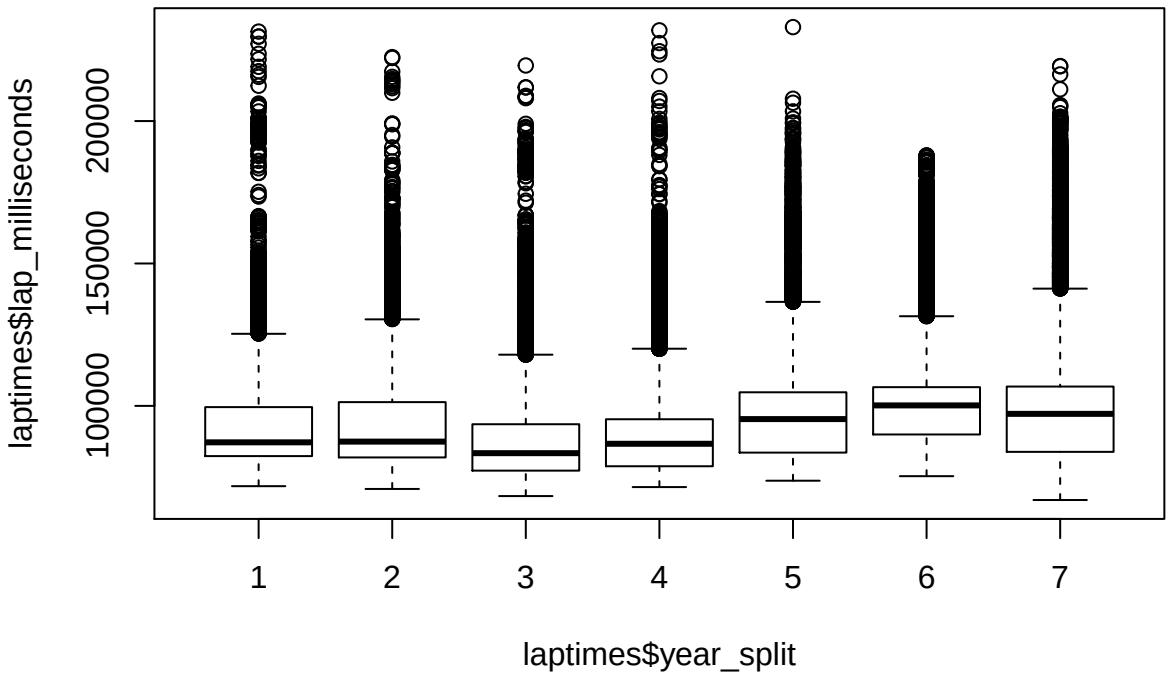
```
summary(laptimes$lap_milliseconds)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   66957   82615   92514   95140  103718  232983
```

This is much more in line with what we'd expect.

Let's see how our year_split has done in segmenting lap times.

```
plot(laptimes$lap_milliseconds~laptimes$year_split)
```



Algorithms and analysis

Linear regression

Now we have our dataframe and we can start running some algorithms. Let's split into train and test and begin with linear regression.

```
set.seed(1234)
i <- sample(1:nrow(laptimes), 0.75*nrow(laptimes), replace=FALSE)
train <- laptimes[i,]
test <- laptimes[-i,]
```

Here I've run linear regression with the predictors discussed before. One addition I made was adding a predictor which is an interaction effect between circuitId and the final lap time (final_milliseconds). Since these two values are closely related, I think they help give us a more complete picture.

```
# Linear Regression
lm_laps <- lm(lap_milliseconds~final_milliseconds+final_position+position+nationality+circuitId+construc
summary(lm_laps)
```

```
##
## Call:
## lm(formula = lap_milliseconds ~ final_milliseconds + final_position +
##     position + nationality + circuitId + constructorId + age +
##     pit_stop + year_split + first_lap + circuitId * final_milliseconds,
##     data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -40619 -10481  -1949   6892 139065
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  9.534e+04  1.689e+03  56.462  < 2e-16 ***
## final_milliseconds          -7.945e-04  6.070e-05 -13.089  < 2e-16 ***
## final_position              -4.462e+01  1.893e+01  -2.357 0.018417 *
## position                     5.818e+02  1.520e+01  38.272  < 2e-16 ***
## nationalityArgentine         1.428e+04  3.117e+03   4.581 4.63e-06 ***
## nationalityAustralian       -2.773e+03  1.648e+03  -1.683 0.092392 .
## nationalityAustrian         -2.454e+03  1.668e+03  -1.471 0.141278
## nationalityBelgian          -1.992e+03  1.820e+03  -1.095 0.273565
## nationalityBrazilian        -2.452e+03  1.643e+03  -1.492 0.135729
## nationalityBritish          -2.533e+03  1.641e+03  -1.543 0.122815
## nationalityCanadian         -1.224e+03  1.663e+03  -0.736 0.461803
## nationalityColombian        -3.558e+03  1.669e+03  -2.132 0.032997 *
## nationalityDanish           -5.470e+03  1.688e+03  -3.241 0.001193 **
## nationalityDutch            -3.303e+03  1.664e+03  -1.985 0.047122 *
## nationalityFinnish          -2.614e+03  1.643e+03  -1.591 0.111578
## nationalityFrench           -9.743e+02  1.645e+03  -0.592 0.553747
## nationalityGerman           -2.218e+03  1.640e+03  -1.352 0.176343
## nationalityItalian          -3.611e+03  1.647e+03  -2.193 0.028330 *
## nationalityJapanese         -2.821e+03  1.662e+03  -1.698 0.089581 .
## nationalityMexican          -1.064e+03  1.656e+03  -0.643 0.520364
## nationalityMonegasque       -1.596e+03  1.820e+03  -0.877 0.380560
## nationalityNew Zealander    -7.363e+03  2.306e+03  -3.193 0.001408 **
## nationalityPolish           -1.414e+03  1.669e+03  -0.848 0.396705
## nationalityPortuguese        9.055e+03  2.840e+03   3.188 0.001432 **
## nationalityRussian           2.306e+03  1.683e+03   1.370 0.170597
## nationalitySpanish          -2.478e+03  1.645e+03  -1.507 0.131802
## nationalitySwedish          -6.090e+03  1.767e+03  -3.446 0.000568 ***
## nationalitySwiss            -5.267e+02  1.727e+03  -0.305 0.760397
## nationalityThai             -1.694e+02  2.938e+03  -0.058 0.954029
## nationalityVenezuelan       -3.014e+03  1.687e+03  -1.787 0.073942 .
## circuitId                   -1.562e+03  2.070e+01 -75.470  < 2e-16 ***
## constructorId                3.303e+00  8.746e-01   3.776 0.000159 ***
## age                         -1.178e+01  9.149e+00  -1.287 0.197994
## pit_stopTRUE                 1.150e+04  3.034e+02  37.902  < 2e-16 ***
## year_split2                  1.205e+03  1.832e+02   6.579 4.76e-11 ***
## year_split3                 -3.892e+03  1.864e+02 -20.882  < 2e-16 ***
## year_split4                 -1.669e+03  1.759e+02  -9.494  < 2e-16 ***
## year_split5                  3.058e+03  1.725e+02  17.728  < 2e-16 ***
## year_split6                  5.398e+03  1.748e+02  30.882  < 2e-16 ***
## year_split7                  3.161e+03  1.665e+02  18.991  < 2e-16 ***
## first_lapTRUE                1.353e+04  2.940e+02  46.024  < 2e-16 ***
```

```
## final_milliseconds:circuitId  2.826e-04  3.523e-06  80.220  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15220 on 164512 degrees of freedom
## Multiple R-squared:  0.1784, Adjusted R-squared:  0.1782
## F-statistic: 871.4 on 41 and 164512 DF,  p-value: < 2.2e-16
```

The results of linear regression were not great. We see here we have an r-squared value of 0.1784, despite a lot of really decent predictors and a low p-value. It looks like our model is underperforming. Let's make a prediction with it and look at the correlation.

```
pred <- predict(lm_laps, newdata=test)
cor(pred, test$lap_milliseconds)
```

```
## [1] 0.421463
```

Correlation is scaled between 0 and 1, and the result was really not terrible, but not good either. After many attempts, I can't get linear regression to perform any better, either.

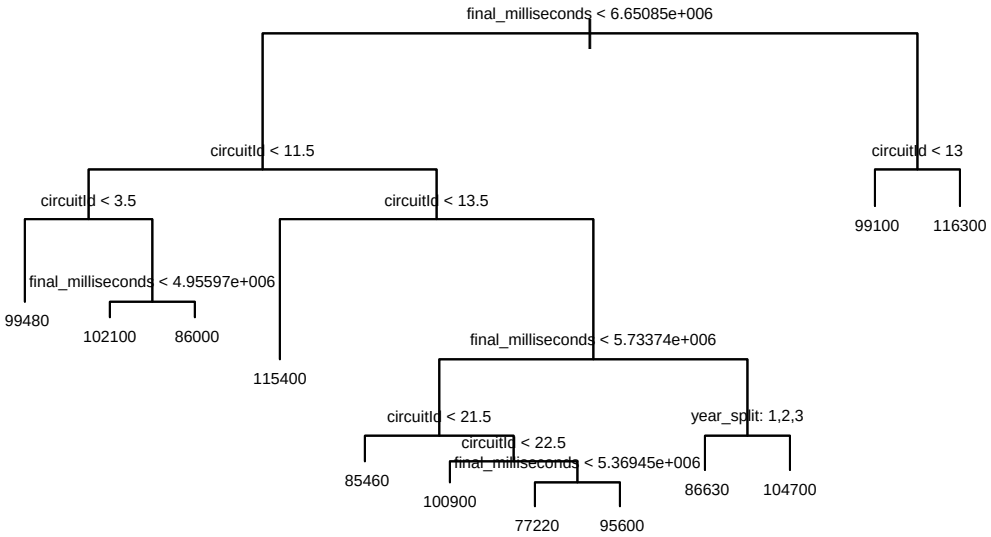
Regression Tree

Let's try another algorithm: a regression tree.

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.5.3
```

```
race_tree <- tree(lap_milliseconds~final_milliseconds+final_position+position+circuitId+constructorId+a
plot(race_tree)
text(race_tree, cex=0.5, pretty=0)
```

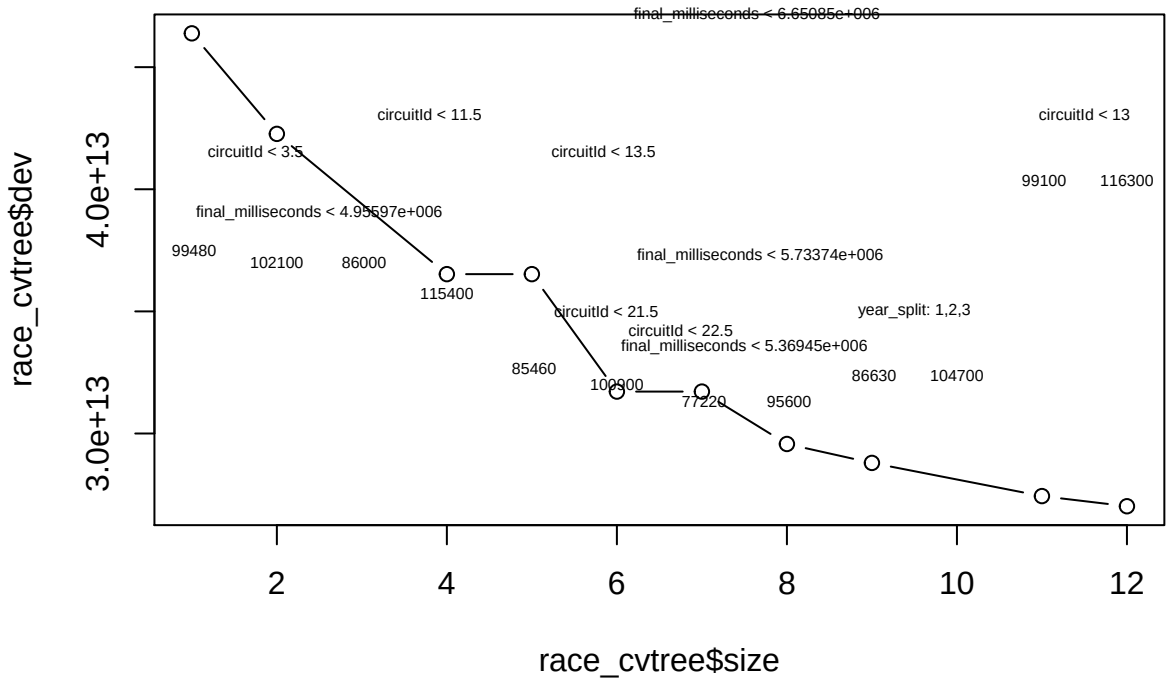



```
race_pred <- predict(race_tree, newdata = test)
cor(race_pred, test$lap_milliseconds)
```

```
## [1] 0.6414477
```

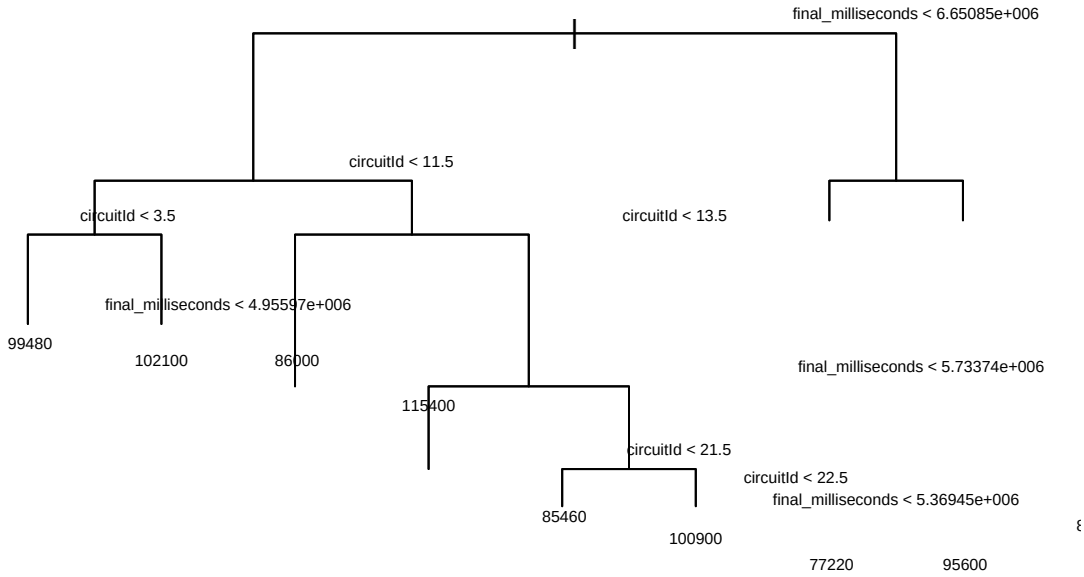
Our tree has performed significantly better than our linear regression model! Let's see if we can optimize further by pruning.

```
race_cvtree <- cv.tree(race_tree)
plot(race_cvtree$size, race_cvtree$dev, type='b')
text(race_tree, cex=0.5, pretty=0)
```



Let's try using best=8 when we prune the tree. This should provide a model which strikes a good balance between overfit and underfit.

```
tree_pruned <- prune.tree(race_tree, best=8)
plot(tree_pruned)
text(race_tree, cex=0.5, pretty=0)
```




```
race_pred <- predict(tree_pruned, newdata = test)
cor(race_pred, test$lap_milliseconds)
```

```
## [1] 0.5990288
```

Our pruned tree performs worse than the fully grown tree. This isn't terribly unexpected because fully grown trees tend to overfit data and sometimes outperform pruned trees in general.

Overall we've gotten a much better result with regression trees than with linear regression. One reason this may be is because of the complexity of the data. Trees can outperform regression in cases where more complex data is involved. It also seems to imply that final_milliseconds is one of our best predictors, since that is where the first split occurs.

kNN

Fianally, we'll run the kNN regression algorithm. I've performed cross-validation separately; it took about 25 minutes to run, so I didn't keep the code in this final result. k=7 performed the best from 3-19 on odd numbers. I've also removed a lot of our predictors and kept only those which I've found to be the most powerful because kNN tends to perform better with fewer dimensions.

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
race_knn <- knnreg(lap_milliseconds~final_milliseconds+position+final_position+circuitId, data=train, k=
predictions <- predict(race_knn, test[,c(5, 11:13, 15, 16)])
cor(predictions, test$lap_milliseconds)
```

```
## [1] 0.8040097
```

Knn seems to be far outperforming our other models.

Overall kNN performed the best of the three models. I think this is probably because all lap times are pretty tightly packed, since consistency is a core tenet of the sport. Unfortunately, for the same reason I don't think any of our models performed particularly well overall. The nature of Formula 1 races is such that, barring any serious incidents between vehicles, everyone's lap times should be pretty close to eachother, usually within a few seconds seconds. This means that our model couldn't really give us any insight into lap times. In fact, it's very likely that a seasoned fan could guess lap times better than our model, just off the cuff. This makes sense when you think about it; if someone could consistently predict lap times given this data, then they would be doing it. They'd make a fortune gambling on races.