# Stat 1361 Final Project

Ryan Quinlan

2024-04-02

## Exploratory Data Analysis

Read in both Train and Test data

```
train.df = read.csv("train.csv", header = TRUE)
test.df = read.csv("test.csv", header = TRUE)
```

Looking at the first 6 rows of the data to get a gauge of what predictors to work with

```
head(train.df)
```

```
##   id                      album_name                              track_name
## 1  1 Tum Hi Aana (From "Marjaavaan")       Tum Hi Aana (From "Marjaavaan")
## 2  2                  Hounds Of Love Running Up That Hill (A Deal With God)
## 3  3                       Fine Line                       Watermelon Sugar
## 4  4                  Sunny Mornings                       I Feel It Coming
## 5  5                     Elis Regina                         Tiro Ao Álvaro
## 6  6            Weihnachten Playlist                              Mistletoe
##   popularity duration_ms explicit danceability energy key loudness mode
## 1         70      249126    FALSE        0.473  0.401   4   -7.018    0
## 2         90      298933    FALSE        0.629  0.547  10  -13.123    0
## 3         89      174000    FALSE        0.548  0.816   0   -4.209    1
## 4          3      269186    FALSE        0.773  0.820   0   -5.897    0
## 5         39      162400    FALSE        0.603  0.649  10   -7.006    1
## 6          0      183066    FALSE        0.623  0.668   6   -7.282    0
##   speechiness acousticness instrumentalness liveness valence   tempo
## 1      0.0241        0.629         1.17e-05   0.1540   0.345  90.076
## 2      0.0550        0.720         3.14e-03   0.0604   0.197 108.375
## 3      0.0465        0.122         0.00e+00   0.3350   0.557  95.390
## 4      0.1150        0.394         0.00e+00   0.0739   0.555  92.996
## 5      0.1040        0.463         0.00e+00   0.1010   0.957  93.781
## 6      0.0531        0.475         0.00e+00   0.0862   0.823 161.948
##   time_signature track_genre
## 1              4         pop
## 2              4        rock
## 3              4         pop
## 4              4         pop
## 5              4        jazz
## 6              4         pop
```

List of each predictor type whether it be int, chr, num, etc.

```r
str(train.df)
```

```
## 'data.frame':    1200 obs. of  19 variables:
##  $ id               : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ album_name       : chr  "Tum Hi Aana (From \"Marjaavaan\")" "Hounds Of Love" "Fine Line" "Sunny Mo
##  $ track_name       : chr  "Tum Hi Aana (From \"Marjaavaan\")" "Running Up That Hill (A Deal With God
##  $ popularity       : int  70 90 89 3 39 0 72 5 40 77 ...
##  $ duration_ms      : int  249126 298933 174000 269186 162400 183066 197759 265186 124253 219801 ...
##  $ explicit         : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ danceability     : num  0.473 0.629 0.548 0.773 0.603 0.623 0.636 0.641 0.729 0.52 ...
##  $ energy           : num  0.401 0.547 0.816 0.82 0.649 0.668 0.647 0.346 0.216 0.903 ...
##  $ key              : int  4 10 0 0 10 6 0 0 0 2 ...
##  $ loudness         : num  -7.02 -13.12 -4.21 -5.9 -7.01 ...
##  $ mode             : int  0 0 1 0 1 0 0 1 1 1 ...
##  $ speechiness      : num  0.0241 0.055 0.0465 0.115 0.104 0.0531 0.0436 0.036 0.0795 0.042 ...
##  $ acousticness     : num  0.629 0.72 0.122 0.394 0.463 0.475 0.422 0.913 0.735 0.00517 ...
##  $ instrumentalness: num  1.17e-05 3.14e-03 0.00 0.00 0.00 0.00 0.00 3.99e-05 0.00 0.00 ...
##  $ liveness         : num  0.154 0.0604 0.335 0.0739 0.101 0.0862 0.251 0.104 0.0615 0.254 ...
##  $ valence          : num  0.345 0.197 0.557 0.555 0.957 0.823 0.44 0.187 0.755 0.652 ...
##  $ tempo            : num  90.1 108.4 95.4 93 93.8 ...
##  $ time_signature   : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ track_genre      : chr  "pop" "rock" "pop" "pop" ...
```

## There are a few observations here:

- explicit is logi
- track_genre is chr with the 3 genres as focused on (pop, rock, jazz)
- album_name and track_name are also chr.
- Every other predictor is either num or int

## Data Modification:

- Convert characters into factors in order to use them in my predictions
- Remove album_name, id, and track_name from both training and test sets
- These are just labels for each track and are not needed in the model

```r
# Converting track_genre from chr to fctr
train.df$track_genre = factor(train.df$track_genre, levels = c("pop", "rock", "jazz"))

#Need to apply this to the test set as well
test.df$track_genre = factor(test.df$track_genre, levels = c("pop", "rock", "jazz"))
head(test.df)
```

```
##     id                                       album_name
## 1 1201                          Beste Weihnachtslieder
## 2 1202                                       LUGNA HITS
## 3 1203                                       Blue Train
## 4 1204 Kannil Kannil [From "Sita Ramam (Malayalam)"]
## 5 1205                                       REGNIG DAG
```

```
## 6 1206                                        Shiddat
##                                     track_name duration_ms explicit
## 1                             Frosty The Snowman      131733    FALSE
## 2                                   Out of Time      214190    FALSE
## 3                                    Blue Train      643127    FALSE
## 4 Kannil Kannil (From "Sita Ramam (Malayalam)")      232513    FALSE
## 5                  As Tears Go By - Mono Version      165213    FALSE
## 6                           Shiddat Title Track      230875    FALSE
##   danceability energy key loudness mode speechiness acousticness
## 1        0.579  0.502   8   -7.570    1      0.0513        0.733
## 2        0.629  0.763   0   -4.279    0      0.0453        0.244
## 3        0.503  0.286   8  -15.425    1      0.0450        0.643
## 4        0.516  0.409   6  -11.925    0      0.0566        0.718
## 5        0.329  0.282   7  -10.863    1      0.0294        0.710
## 6        0.489  0.625   9   -6.497    0      0.0584        0.288
##   instrumentalness liveness valence   tempo time_signature track_genre
## 1          0.00000    0.281   0.836  76.783              4        jazz
## 2          0.00000    0.312   0.809  93.023              4         pop
## 3          0.00112    0.156   0.569 136.098              4        jazz
## 4          0.00000    0.350   0.731 179.852              3         pop
## 5          0.00265    0.158   0.393 112.944              4        rock
## 6          0.00000    0.133   0.439 149.921              3         pop
```

Convert explicit into numeric

```
#Converting the explicit variable into numeric form for ease
train.df$explicit = as.numeric(train.df$explicit)
test.df$explicit = as.numeric(test.df$explicit)
```

Now that the track_genre is a fctr and explicit is numeric, we can remove the labels of each track mentioned previously (album_name, id, and track_name) from both the train and test set

```
#only run once when testing otherwise it will blank the dataframe
#Save train ids
train.ids = train.df$id
train.df = train.df[, -which(names(train.df) == "id")]

#Save test ids
test.ids = test.df$id
test.df = test.df[, -which(names(test.df) == "id")]
```

Remove album_name and track_name and save them just like ids

```
#Removing album_name
train.album_name = train.df$album_name
train.df = train.df[, -which(names(train.df) == "album_name")]

test.album_name = test.df$album_name
test.df = test.df[, -which(names(test.df) == "album_name")]
```

```
#Removing track_name
train.track_name = train.df$track_name
train.df = train.df[, -which(names(train.df) == "track_name")]
```

```
test.track_name = test.df$track_name
test.df = test.df[, -which(names(test.df) == "track_name")]
```

I want to get an understanding of the split of the track_genre just for reference

```
table.train = table(train.df$track_genre)
print(table.train)
```

```
##
##  pop rock jazz
##  403  402  395
```

We can see that the genre split is nearly even across the 3 genres.

Now for test data

```
table.test = table(test.df$track_genre)
print(table.test)
```

```
##
##  pop rock jazz
##  162  164  174
```

Once again, relatively even except jazz has a few more observations compared to the other 2.

Summary Statistics for each variable

```
summary(train.df)
```

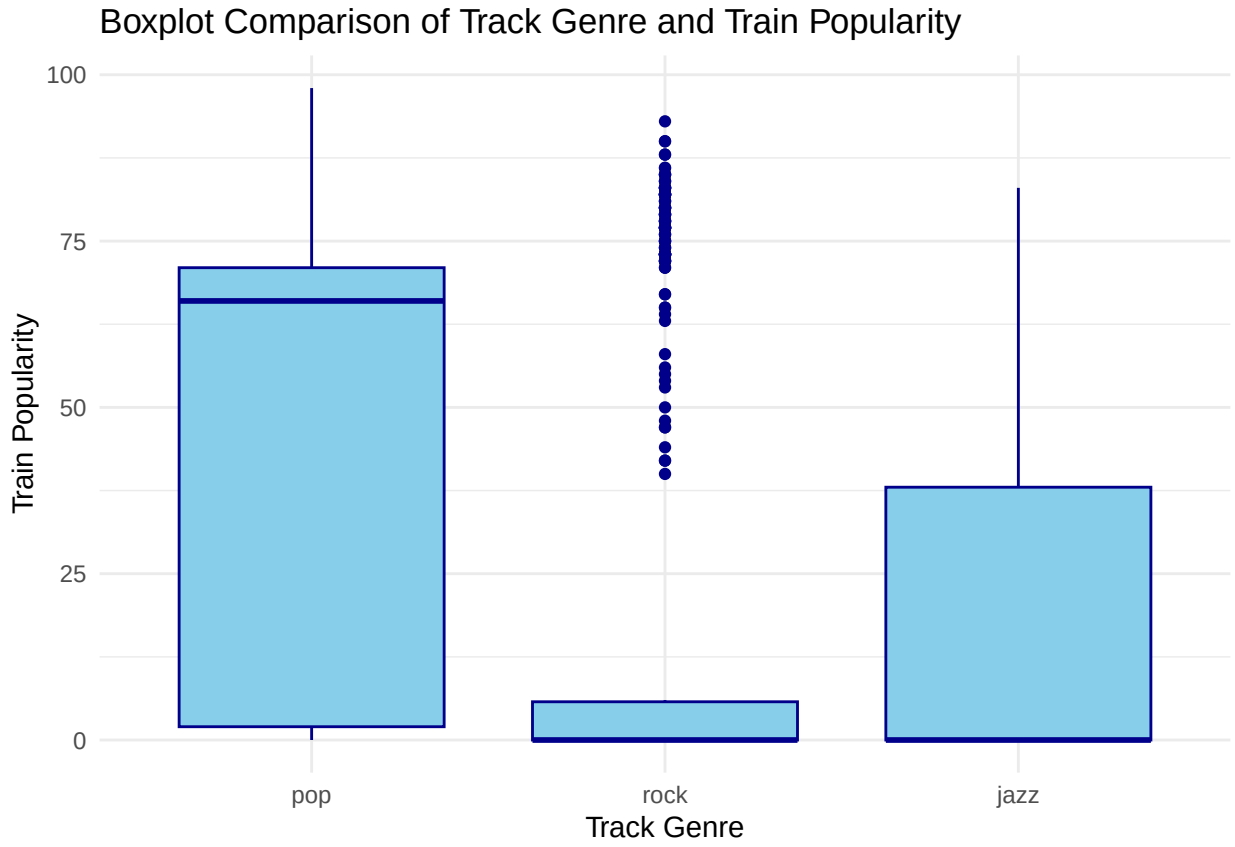
```
##    popularity     duration_ms        explicit        danceability
##  Min.   : 0.00   Min.   : 42440   Min.   :0.00000   Min.   :0.0000
##  1st Qu.: 0.00   1st Qu.:171204   1st Qu.:0.00000   1st Qu.:0.4620
##  Median : 1.00   Median :200229   Median :0.00000   Median :0.5790
##  Mean   :27.38   Mean   :208056   Mean   :0.04167   Mean   :0.5641
##  3rd Qu.:66.00   3rd Qu.:240400   3rd Qu.:0.00000   3rd Qu.:0.6753
##  Max.   :98.00   Max.   :471160   Max.   :1.00000   Max.   :0.9180
##      energy             key            loudness           mode
##  Min.   :0.00756   Min.   : 0.000   Min.   :-29.694   Min.   :0.0000
##  1st Qu.:0.37075   1st Qu.: 2.000   1st Qu.:-10.312   1st Qu.:0.0000
##  Median :0.55300   Median : 5.000   Median : -7.635   Median :1.0000
##  Mean   :0.54386   Mean   : 5.218   Mean   : -8.426   Mean   :0.6608
##  3rd Qu.:0.71900   3rd Qu.: 8.000   3rd Qu.: -5.645   3rd Qu.:1.0000
##  Max.   :0.99000   Max.   :11.000   Max.   : -1.583   Max.   :1.0000
##   speechiness       acousticness       instrumentalness       liveness
##  Min.   :0.00000   Min.   :0.0000103   Min.   :0.0000000   Min.   :0.0179
##  1st Qu.:0.03280   1st Qu.:0.0818000   1st Qu.:0.0000000   1st Qu.:0.0938
##  Median :0.04120   Median :0.4060000   Median :0.0000012   Median :0.1190
##  Mean   :0.06344   Mean   :0.4239071   Mean   :0.0334791   Mean   :0.1648
##  3rd Qu.:0.06153   3rd Qu.:0.7515000   3rd Qu.:0.0002657   3rd Qu.:0.2005
##  Max.   :0.50500   Max.   :0.9950000   Max.   :0.9670000   Max.   :0.9580
##     valence           tempo        time_signature  track_genre
```

```
##  Min.   :0.0000   Min.   :  0.00   Min.   :0.000   pop :403
##  1st Qu.:0.3300   1st Qu.: 93.96   1st Qu.:4.000   rock:402
##  Median :0.5060   Median :118.74   Median :4.000   jazz:395
##  Mean   :0.5086   Mean   :119.68   Mean   :3.872
##  3rd Qu.:0.6830   3rd Qu.:138.91   3rd Qu.:4.000
##  Max.   :0.9800   Max.   :207.48   Max.   :5.000
```

---

Plotting a comparison of track_genre and popularity from the training data

```
library(ggplot2)
ggplot(train.df, aes(x = track_genre, y = popularity)) +
  geom_boxplot(fill = "skyblue", color = "darkblue") +
  theme_minimal() +
  labs(
    x = "Track Genre",
    y = "Train Popularity",
    title = "Boxplot Comparison of Track Genre and Train Popularity")
```




## Data Oddities, extreme values, influential points, etc.

The first checkpoint is checking for missing data in any of the columns. Important to make sure there is no missing data to continue with exploring.

```r
colSums(is.na(train.df))
```

```
##       popularity      duration_ms         explicit     danceability
##                0                0                0                0
##           energy              key         loudness             mode
##                0                0                0                0
##      speechiness     acousticness instrumentalness         liveness
##                0                0                0                0
##          valence            tempo   time_signature      track_genre
##                0                0                0                0
```

There is no missing data in the train set.

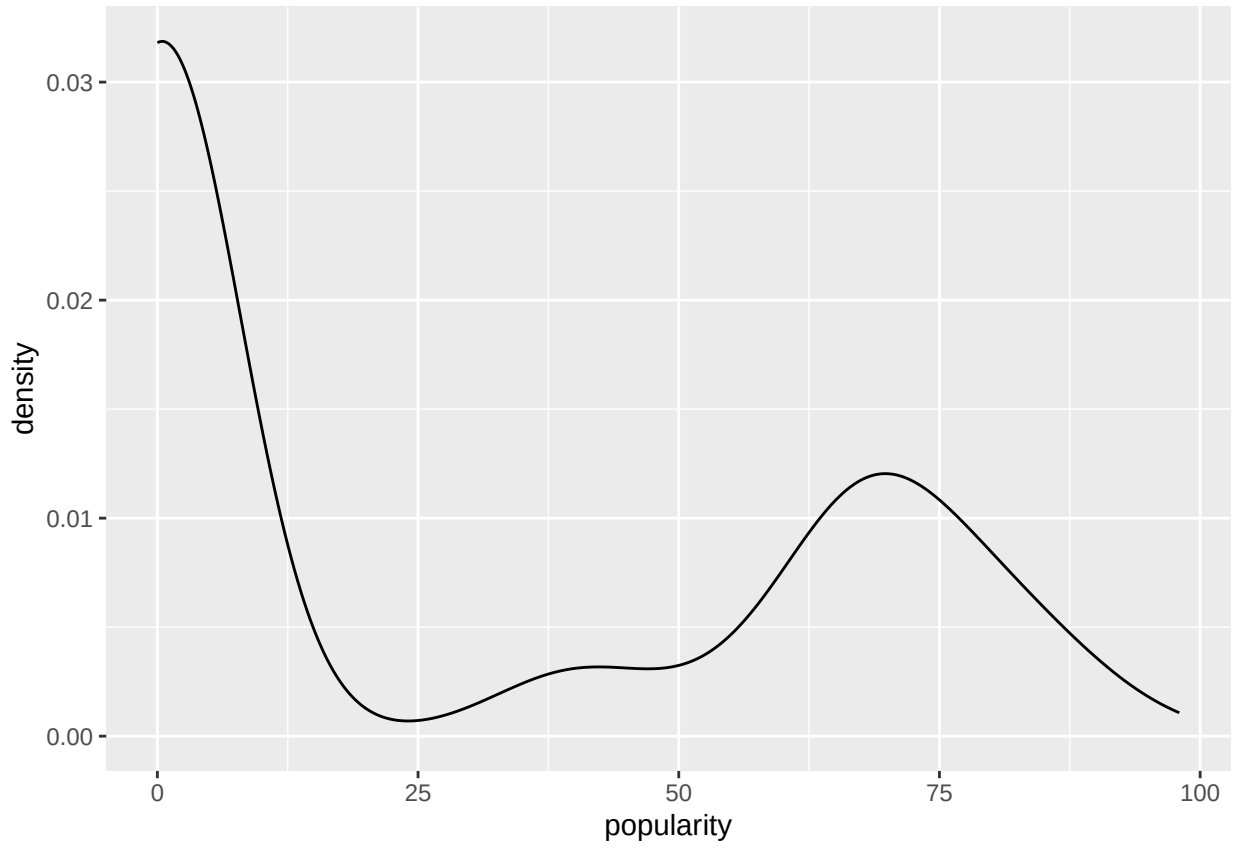
```r
colSums(is.na(test.df))
```

```
##      duration_ms         explicit     danceability           energy
##                0                0                0                0
##              key         loudness             mode      speechiness
##                0                0                0                0
##     acousticness instrumentalness         liveness          valence
##                0                0                0                0
##            tempo   time_signature      track_genre
##                0                0                0
```

There is also no missing data in the test set

Plotting the density of popularity

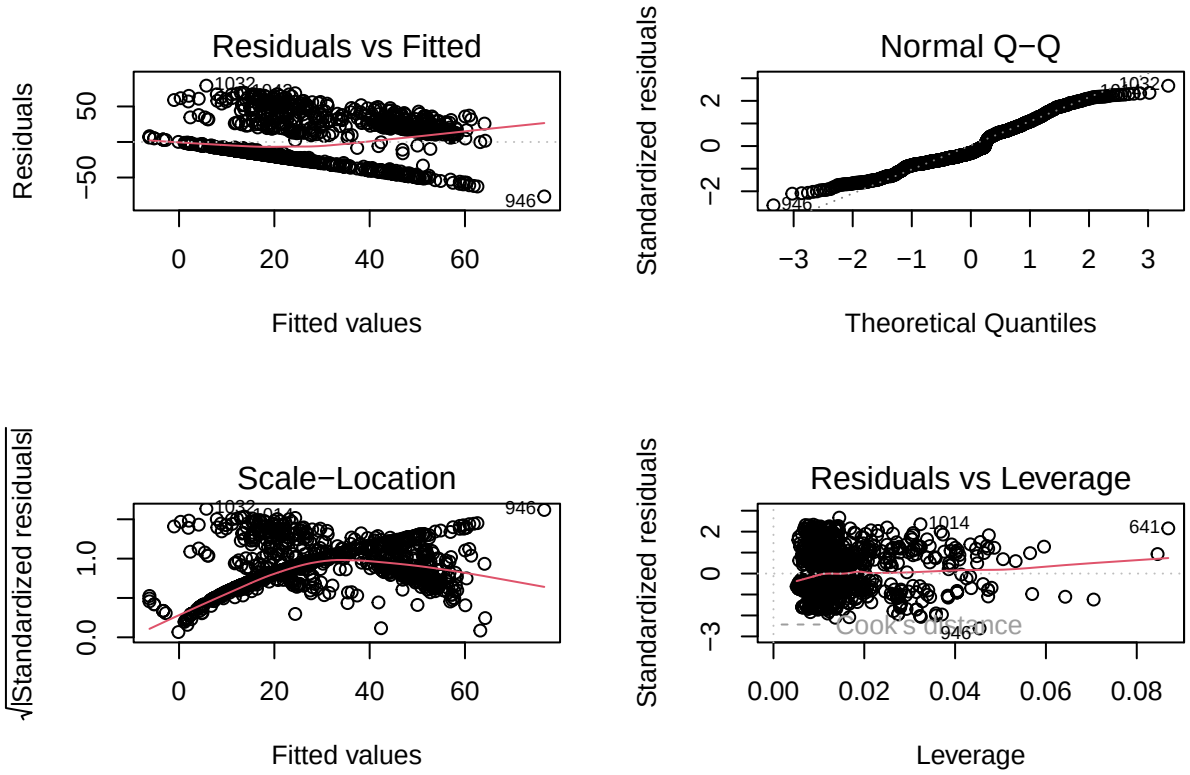
```r
ggplot(train.df, aes(popularity)) +
  geom_density()
```




Here we see that the density of popularity is right-skewed and somewhat bimodal

Fitting the full model

```
#fit the model using popularity as the response against all predictors
lm.fit = lm(popularity ~ ., data = train.df)
par(mfrow= c(2,2))
plot(lm.fit)
```



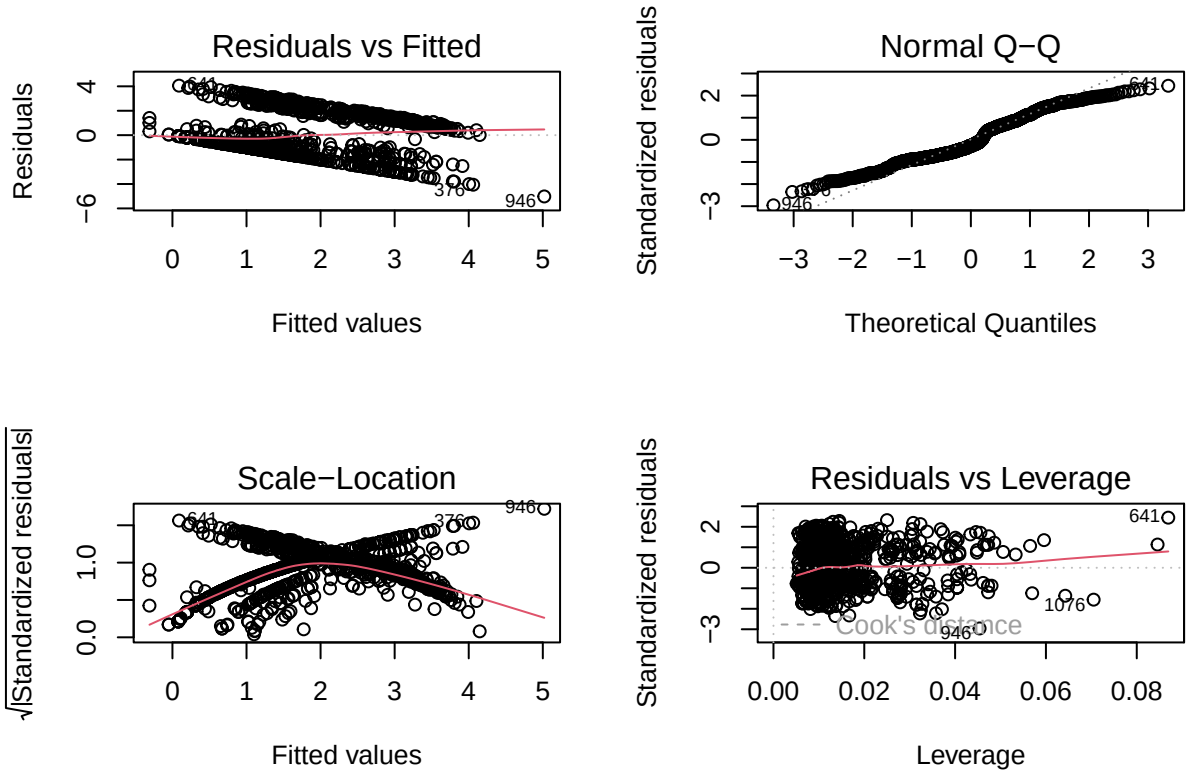

## Observations:

- Q-Q Plot appears to be linear meaning normality is satisfied.
- There is a negative linear pattern in the Residual vs Fitted plot indicating a violation of linearity. There should be no clear pattern.
- While homoscedasticity may be a problem, it is not the end all be all. Given the fact the density is right-skewed, a lot of the variance in the error term is likely to be a result of the skewed density.
- The residuals are normally distributed in this case.

## Transforming the response variable

Two things here: I am transforming the response variable due to the density plot being skewed and bi-modal Adding a scalar to the popularity variable because a lot of the values are at 0. You cannot log 0.

```
#fitting a log transformed model for comparison
log.fit = lm(log(popularity + 1) ~ ., data = train.df)
par(mfrow = c(2,2))
plot(log.fit)
```




Observations:

- Linearity has not changed, still has a negative linear relationship and does not fit the line evenly.
- Looking at the Q-Q plot, normality got slightly worse.
- Homoscedasticity has still not been satisfied.
- Residuals vs. Leverage plot has not changed much when looking at the scale difference on the leverage axis. There are multiple influential points.

## Final Decision:

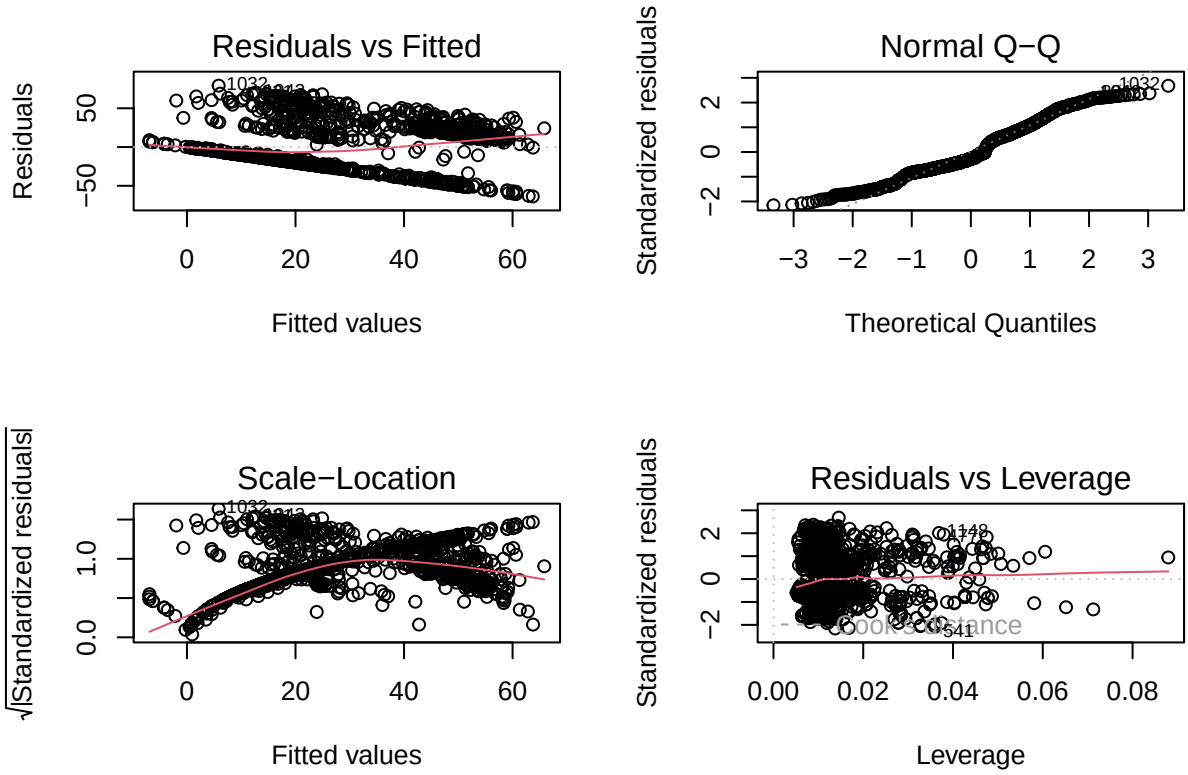
I am going to still use the original un-transformed data and remove a few influential points since there is no significant difference between the un-transformed data and the original data.

Removing Influential Points

```
train.df = train.df[-c(641,946,1014),]
```

Updated

```
fit = lm(popularity ~ ., data = train.df)
par(mfrow= c(2,2))
plot(fit)
```




---

Summary of the Full Regression Model (all predictors)

```
summary(fit)
```

```
##
## Call:
## lm(formula = popularity ~ ., data = train.df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -63.73 -20.70  -8.15  21.73  79.16
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -6.253e+00  1.265e+01  -0.494  0.62124
## duration_ms       6.234e-05  1.581e-05   3.943 8.53e-05 ***
## explicit          5.537e+00  4.483e+00   1.235  0.21708
## danceability      2.257e+01  7.760e+00   2.909  0.00370 **
## energy            2.383e+01  8.978e+00   2.655  0.00804 **
## key               2.136e-01  2.495e-01   0.856  0.39219
## loudness         -4.427e-01  4.018e-01  -1.102  0.27081
## mode             -2.861e+00  1.915e+00  -1.494  0.13554
## speechiness       4.930e+00  1.518e+01   0.325  0.74546
```

```
## acousticness      6.574e+00  4.646e+00   1.415  0.15736
## instrumentalness  8.549e+00  6.557e+00   1.304  0.19254
## liveness          3.479e+00  7.530e+00   0.462  0.64417
## valence          -2.068e+01  5.036e+00  -4.107 4.29e-05 ***
## tempo            -5.950e-02  2.948e-02  -2.019  0.04375 *
## time_signature    5.891e+00  2.048e+00   2.877  0.00409 **
## track_genrerock  -2.634e+01  2.386e+00 -11.039  < 2e-16 ***
## track_genrejazz  -2.604e+01  2.775e+00  -9.386  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.76 on 1180 degrees of freedom
## Multiple R-squared:  0.2384, Adjusted R-squared:  0.228
## F-statistic: 23.08 on 16 and 1180 DF,  p-value: < 2.2e-16
```

Correlation Matrix

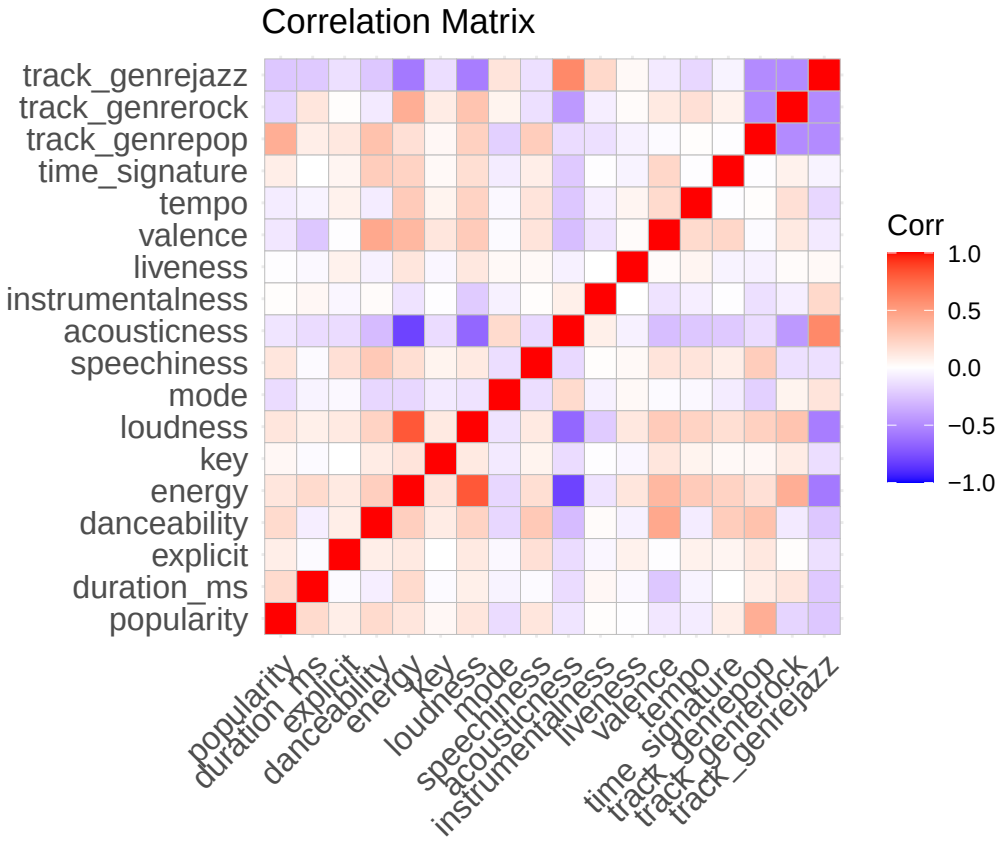
```
library(ggcorrplot)
#Need to convert all the data to numeric form in order to be able to form a Correlation matrix
converted = model.matrix(~ . - 1, data = train.df)

#cor_matrix
corr = round(cor(converted), 2)
corr[, 1:17]
```

```
##                  popularity duration_ms explicit danceability energy   key
## popularity             1.00        0.19     0.09         0.19   0.13  0.04
## duration_ms            0.19        1.00    -0.02        -0.07   0.19 -0.02
## explicit               0.09       -0.02     1.00         0.09   0.11  0.00
## danceability           0.19       -0.07     0.09         1.00   0.25  0.10
## energy                 0.13        0.19     0.11         0.25   1.00  0.14
## key                    0.04       -0.02     0.00         0.10   0.14  1.00
## loudness               0.13        0.08     0.11         0.23   0.81  0.11
## mode                  -0.15       -0.05    -0.03        -0.17  -0.17 -0.09
## speechiness            0.13       -0.02     0.16         0.28   0.17  0.06
## acousticness          -0.11       -0.15    -0.15        -0.29  -0.81 -0.15
## instrumentalness       0.01        0.04    -0.04         0.02  -0.12 -0.01
## liveness              -0.01       -0.03     0.07        -0.06   0.13 -0.04
## valence               -0.10       -0.24    -0.01         0.45   0.37  0.13
## tempo                 -0.08       -0.05     0.07        -0.08   0.27  0.06
## time_signature         0.09        0.00     0.05         0.26   0.23  0.03
## track_genrepop         0.42        0.09     0.12         0.32   0.16  0.04
## track_genrerock       -0.18        0.13     0.01        -0.09   0.42  0.10
## track_genrejazz       -0.24       -0.23    -0.13        -0.24  -0.58 -0.14
##                  loudness  mode speechiness acousticness instrumentalness
## popularity           0.13 -0.15        0.13        -0.11             0.01
## duration_ms          0.08 -0.05       -0.02        -0.15             0.04
## explicit             0.11 -0.03        0.16        -0.15            -0.04
## danceability         0.23 -0.17        0.28        -0.29             0.02
## energy               0.81 -0.17        0.17        -0.81            -0.12
## key                  0.11 -0.09        0.06        -0.15            -0.01
## loudness             1.00 -0.12        0.11        -0.66            -0.22
## mode                -0.12  1.00       -0.14         0.19            -0.06
```

```
## speechiness          0.11 -0.14        1.00        -0.16             0.01
## acousticness        -0.66  0.19       -0.16         1.00             0.08
## instrumentalness    -0.22 -0.06        0.01         0.08             1.00
## liveness             0.12  0.03        0.03        -0.06             0.00
## valence              0.27 -0.02        0.14        -0.28            -0.12
## tempo                0.23 -0.03        0.14        -0.24            -0.07
## time_signature       0.17 -0.08        0.09        -0.23            -0.01
## track_genrepop       0.24 -0.20        0.26        -0.15            -0.13
## track_genrerock      0.31  0.06       -0.13        -0.44            -0.07
## track_genrejazz     -0.56  0.14       -0.13         0.60             0.20
##                   liveness valence tempo time_signature track_genrepop
## popularity           -0.01   -0.10 -0.08           0.09           0.42
## duration_ms          -0.03   -0.24 -0.05           0.00           0.09
## explicit              0.07   -0.01  0.07           0.05           0.12
## danceability         -0.06    0.45 -0.08           0.26           0.32
## energy                0.13    0.37  0.27           0.23           0.16
## key                  -0.04    0.13  0.06           0.03           0.04
## loudness              0.12    0.27  0.23           0.17           0.24
## mode                  0.03   -0.02 -0.03          -0.08          -0.20
## speechiness           0.03    0.14  0.14           0.09           0.26
## acousticness         -0.06   -0.28 -0.24          -0.23          -0.15
## instrumentalness      0.00   -0.12 -0.07          -0.01          -0.13
## liveness              1.00    0.02  0.05          -0.05          -0.06
## valence               0.02    1.00  0.19           0.21          -0.02
## tempo                 0.05    0.19  1.00          -0.01           0.01
## time_signature       -0.05    0.21 -0.01           1.00          -0.01
## track_genrepop       -0.06   -0.02  0.01          -0.01           1.00
## track_genrerock       0.02    0.11  0.16           0.07          -0.50
## track_genrejazz       0.03   -0.09 -0.17          -0.05          -0.50
##                   track_genrerock
## popularity              -0.18
## duration_ms              0.13
## explicit                 0.01
## danceability            -0.09
## energy                   0.42
## key                      0.10
## loudness                 0.31
## mode                     0.06
## speechiness             -0.13
## acousticness            -0.44
## instrumentalness        -0.07
## liveness                 0.02
## valence                  0.11
## tempo                    0.16
## time_signature           0.07
## track_genrepop          -0.50
## track_genrerock          1.00
## track_genrejazz         -0.50
```

```
ggcorrplot(corr, title = "Correlation Matrix", type = "full")
```




The 2 most correlated predictors with popularity are track_genre pop and danceability.

There are two instances of collinearity that occur between: Acousticness and energy, Loudness and energy

Thus multicollinearity is occurring because of energy

## Calculating the VIF values

```
library(car)
```

```
## Loading required package: carData
```

```
# Calculating VIF
vif_values = vif(lm.fit)
print(vif_values)
```

```
##                      GVIF Df GVIF^(1/(2*Df))
## duration_ms      1.247481  1        1.116907
## explicit         1.066493  1        1.032712
## danceability     1.771110  1        1.330830
## energy           5.786752  1        2.405567
## key              1.048257  1        1.023844
## loudness         3.339751  1        1.827499
```

```
## mode             1.113847  1        1.055389
## speechiness      1.202169  1        1.096435
## acousticness     3.320260  1        1.822158
## instrumentalness 1.133694  1        1.064751
## liveness         1.065729  1        1.032341
## valence          1.799399  1        1.341417
## tempo            1.194071  1        1.092736
## time_signature   1.155396  1        1.074894
## track_genre      2.502237  2        1.257715
```

The only slightly concerning value would be for energy which makes sense due to the fact it is multicollinear. This has no effect on the model's predictive ability. Rather, we are using this to interpret the model and its predictors' relationships.

## Deciding Variable Importance

```
library(randomForest)
```

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

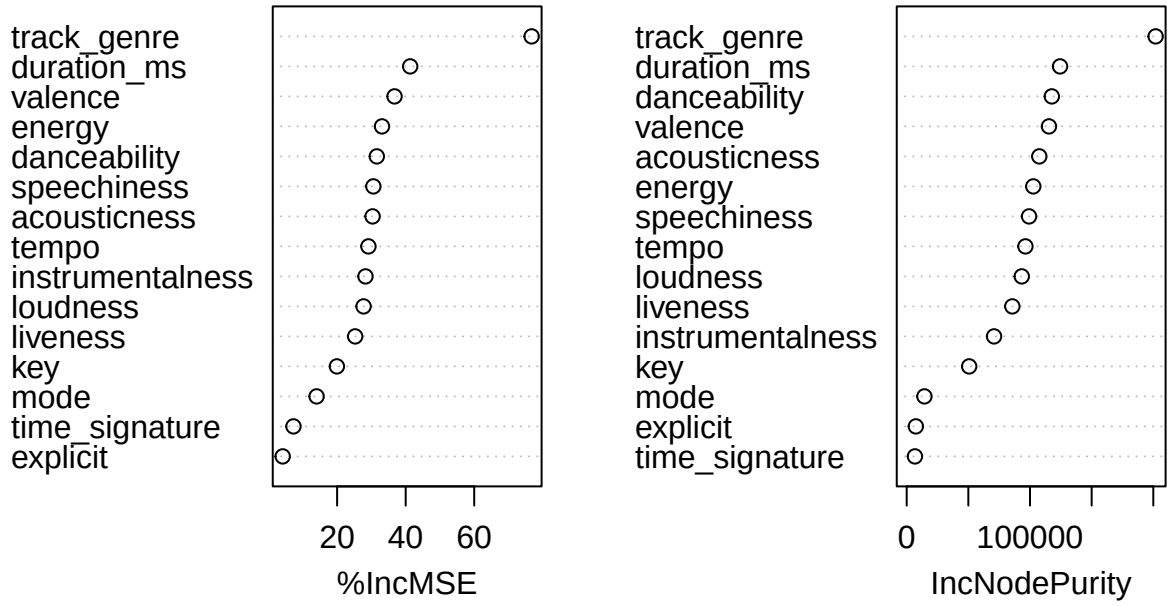
```
## Type rfNews() to see new features/changes/bug fixes.
```

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

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

```
rf <- randomForest(popularity ~ ., data = train.df,
                   mtry = 15/3, ntrees = 500, importance = TRUE)

varImpPlot(rf, main = "Variable Importance")
```




```
importance(rf)
```

```
##                   %IncMSE IncNodePurity
## duration_ms      41.319363    124369.546
## explicit          4.119617      7417.452
## danceability     31.576231    117688.275
## energy           33.094832    102653.371
## key              19.933182     50666.213
## loudness         27.715521     93311.108
## mode             14.019499     14376.984
## speechiness      30.571352     99250.928
## acousticness     30.350411    107551.610
## instrumentalness 28.280151     70843.158
## liveness         25.266236     85665.066
## valence          36.750881    115364.285
## tempo            29.136150     96278.017
## time_signature    7.261560      6679.511
## track_genre      76.755540    201890.264
```

The most important variables are:

- Track_genre
- Duration_ms
- Danceability

- Valence

---

## Models

```
library(boot)
```

```
## Warning: package 'boot' was built under R version 4.2.3
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:car':
##
##     logit
```

```
library(pls)
```

```
## Warning: package 'pls' was built under R version 4.2.3
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
library(leaps)
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.2.3
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.22-3
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

Splitting the training data so we can have a test set to work with

```r
set.seed(11)
attach(train.df)
#Creating a 75-25 split for the training set and test set
train = sample(1:nrow(train.df), nrow(train.df)*(3/4))
test = -train
#Creating the universal y.test
y.test = popularity[test]
```

### Methods

I will be using the following methods to predict popularity

- Linear Regression Model
- Ridge Regression
- Lasso
- Forward Step-wise Selection
- Regression Tree
- Pruned Regression Tree
- Bagging
- Boosting

## Linear Regression Model

I will be using this as the baseline model to compare with and hopefully improve on.

```r
#Predict the popularity for the test data
lm.fit = lm(popularity ~ . , data= train.df, subset = train)
lm.pred = predict(lm.fit, train.df[test,])

#Calculate the MSE
lm.error = mean((lm.pred - y.test)^2)
lm.error
```

```
## [1] 868.8826
```

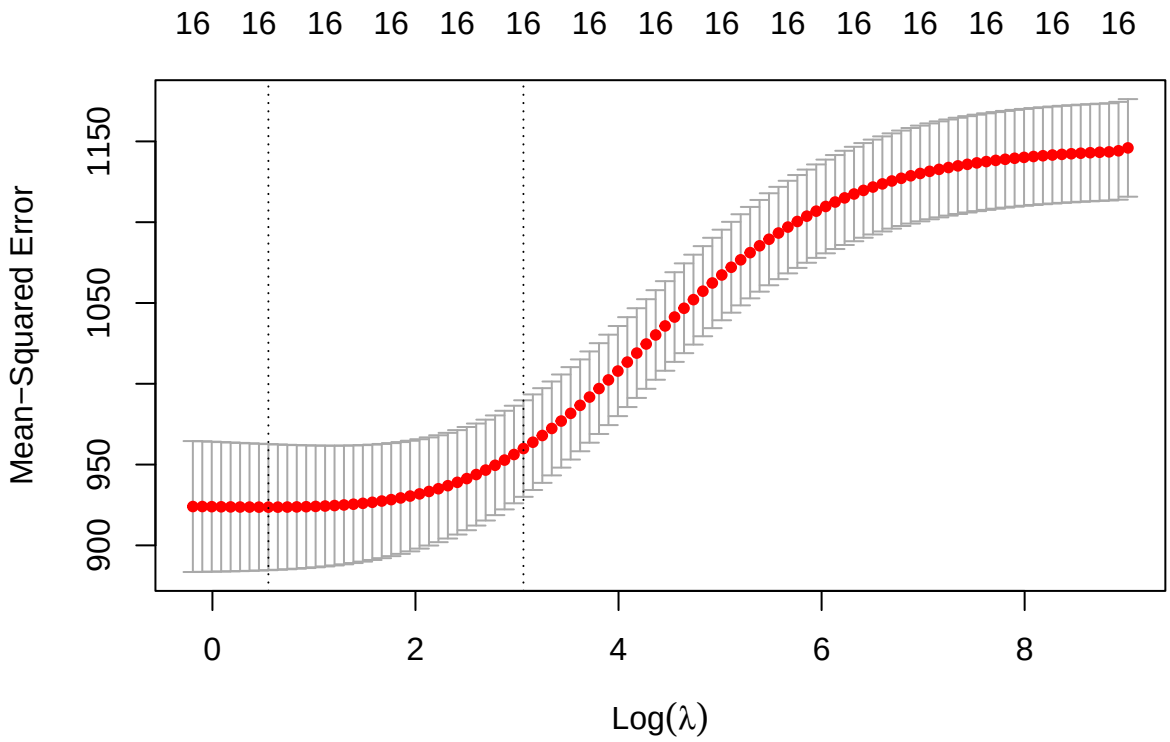
This is the first test error that will be used for comparison

## Ridge Regression (insight into multicollinearity)

```r
#Creating subsets
set.seed(11)
x = model.matrix(popularity ~ ., data = train.df)[, -1]
x.train = x[train,]

y = train.df$popularity
y.train = y[train]
```

```
#Cross-val
#alpha level for ridge is 0
ridge.fit = glmnet(x.train, y.train,alpha=0)
cv.ridge.fit = cv.glmnet(x.train, y.train,alpha=0)
plot(cv.ridge.fit)
```




Calculating the best lambda value that is shown above

```
#penalty term
bestlamda = cv.ridge.fit$lambda.min
bestlamda
```

```
## [1] 1.735794
```

```
ridge.pred = predict(ridge.fit, s=bestlamda, newx=x[test,])
ridge.error = mean((ridge.pred-y.test)^2)
ridge.error
```

```
## [1] 869.4237
```

We see here that the test error is barely greater than the test error of the linear regression model

# LASSO

```
library(glmnet)
#alpha level for LASSO is 1
lasso.fit = glmnet(x.train,y.train,alpha=1)
cv.lasso.fit = cv.glmnet(x.train,y.train,alpha=1)
```

Calculating the best lambda value using Cross-validation.

```
bestlam.lasso = cv.lasso.fit$lambda.min
bestlam.lasso
```

```
## [1] 0.1040581
```

```
lasso.pred = predict(lasso.fit, s=bestlam.lasso, newx=x[test,])
lasso.error = mean((lasso.pred-y.test)^2)
lasso.error
```

```
## [1] 866.6414
```

The lowest test error thus far.

# Forward Step-wise Selection

```
set.seed(11)
#A different syntax for train and test data
#Same as previous train and test variables
#Need this to be able to apply to fss and other models
train.index = sample(1:nrow(train.df), (nrow(train.df)*(3/4)))
train.fss = train.df[train.index,]
test.fss = train.df[-train.index,]
```

We want to find the ideal subset of predictors to use

```
set.seed(11)
fss = regsubsets(popularity ~ . , data = train.fss, nvmax = 16, method = "forward")
#Observe and save summary
fss.sum = summary(fss)
fss.sum
```

```
## Subset selection object
## Call: regsubsets.formula(popularity ~ ., data = train.fss, nvmax = 16,
##     method = "forward")
## 16 Variables  (and intercept)
##                  Forced in Forced out
## duration_ms          FALSE      FALSE
## explicit             FALSE      FALSE
```

```
## danceability         FALSE      FALSE
## energy               FALSE      FALSE
## key                  FALSE      FALSE
## loudness             FALSE      FALSE
## mode                 FALSE      FALSE
## speechiness          FALSE      FALSE
## acousticness         FALSE      FALSE
## instrumentalness     FALSE      FALSE
## liveness             FALSE      FALSE
## valence              FALSE      FALSE
## tempo                FALSE      FALSE
## time_signature       FALSE      FALSE
## track_genrerock      FALSE      FALSE
## track_genrejazz      FALSE      FALSE
## 1 subsets of each size up to 16
## Selection Algorithm: forward
##           duration_ms explicit danceability energy key loudness mode
## 1  ( 1 )  " "         " "      " "          " "    " " " " " "      " "
## 2  ( 1 )  " "         " "      " "          " "    " " " " " "      " "
## 3  ( 1 )  "*"         " "      " "          " "    " " " " " "      " "
## 4  ( 1 )  "*"         " "      " "          " "    " " " " " "      " "
## 5  ( 1 )  "*"         " "      " "          " "    " " " " " "      " "
## 6  ( 1 )  "*"         " "      "*"          " "    " " " " " "      " "
## 7  ( 1 )  "*"         " "      "*"          " "    " " " " " "      "*"
## 8  ( 1 )  "*"         " "      "*"          " "    " " " " " "      "*"
## 9  ( 1 )  "*"         " "      "*"          " "    "*" " " " "      "*"
## 10  ( 1 ) "*"         " "      "*"          " "    "*" " " " "      "*"
## 11  ( 1 ) "*"         "*"      "*"          " "    "*" " " " "      "*"
## 12  ( 1 ) "*"         "*"      "*"          "*"    "*" " " " "      "*"
## 13  ( 1 ) "*"         "*"      "*"          "*"    "*" "*" " "      "*"
## 14  ( 1 ) "*"         "*"      "*"          "*"    "*" "*" " "      "*"
## 15  ( 1 ) "*"         "*"      "*"          "*"    "*" "*" " "      "*"
## 16  ( 1 ) "*"         "*"      "*"          "*"    "*" "*" " "      "*"
##           speechiness acousticness instrumentalness liveness valence tempo
## 1  ( 1 )  " "         " "          " "              " "      " "     " "
## 2  ( 1 )  " "         " "          " "              " "      " "     " "
## 3  ( 1 )  " "         " "          " "              " "      " "     " "
## 4  ( 1 )  " "         " "          " "              " "      " "     " "
## 5  ( 1 )  " "         " "          " "              " "      "*"     " "
## 6  ( 1 )  " "         " "          " "              " "      "*"     " "
## 7  ( 1 )  " "         " "          " "              " "      "*"     " "
## 8  ( 1 )  " "         " "          " "              "*"      "*"     " "
## 9  ( 1 )  " "         " "          " "              "*"      "*"     " "
## 10  ( 1 ) " "         " "          " "              "*"      "*"     "*"
## 11  ( 1 ) " "         " "          " "              "*"      "*"     "*"
## 12  ( 1 ) " "         " "          " "              "*"      "*"     "*"
## 13  ( 1 ) " "         " "          " "              "*"      "*"     "*"
## 14  ( 1 ) " "         "*"          " "              "*"      "*"     "*"
## 15  ( 1 ) " "         "*"          "*"              "*"      "*"     "*"
## 16  ( 1 ) "*"         "*"          "*"              "*"      "*"     "*"
##           time_signature track_genrerock track_genrejazz
## 1  ( 1 )  " "            " "             "*"
## 2  ( 1 )  " "            "*"             "*"
## 3  ( 1 )  " "            "*"             "*"
```

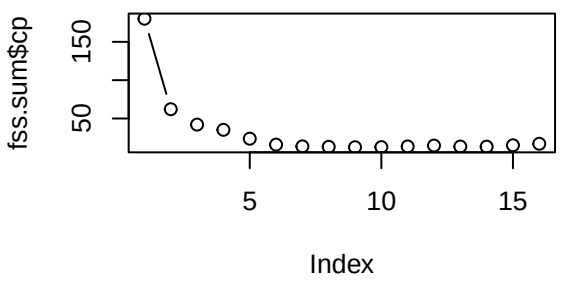
```
## 4  ( 1 )  "*"           "*"           "*"
## 5  ( 1 )  "*"           "*"           "*"
## 6  ( 1 )  "*"           "*"           "*"
## 7  ( 1 )  "*"           "*"           "*"
## 8  ( 1 )  "*"           "*"           "*"
## 9  ( 1 )  "*"           "*"           "*"
## 10  ( 1 ) "*"           "*"           "*"
## 11  ( 1 ) "*"           "*"           "*"
## 12  ( 1 ) "*"           "*"           "*"
## 13  ( 1 ) "*"           "*"           "*"
## 14  ( 1 ) "*"           "*"           "*"
## 15  ( 1 ) "*"           "*"           "*"
## 16  ( 1 ) "*"           "*"           "*"
```

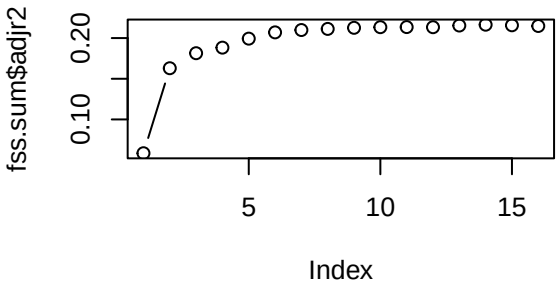
Finding optimal number of predictors using three different methods

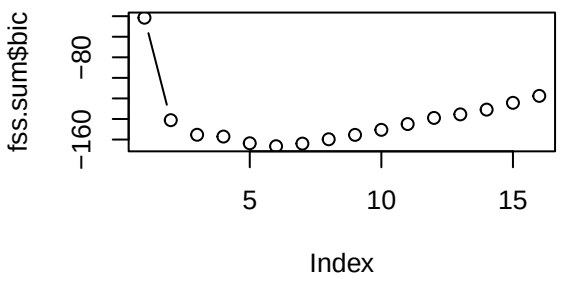
```
par(mfrow = c(2,2))
#Using CP
plot(fss.sum$cp, type = "b")

#Using Adjusted R-Squared
plot(fss.sum$adjr2, type = "b")

#Using Bayesian Information Criteria
plot(fss.sum$bic, type = "b")
```



```r
which.min(fss.sum$cp)
```

```
## [1] 9
```

Using Mallows Cp, the optimal number of predictors is 9

```r
which.max(fss.sum$adjr2)
```

```
## [1] 14
```

Using Adjusted R-squared, the optimal number of predictors is 14

Finding Bayesian

```r
which.min(fss.sum$bic)
```

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

Using Bayesian Information Criteria, the optimal number of predictors is 6

Each optimal number of predictors are different, however, looking at each model we can tell that there is an "elbow" point in which the plot begins to even out and smooth out in its quadratic pattern. The exception is the model using BIC because once it reaches 6 predictors, it begins to increase. This occurs at 6 predictors in every plot above so going forward, 6 predictors will represent the simplest model despite it not having the lowest MSE in every model.

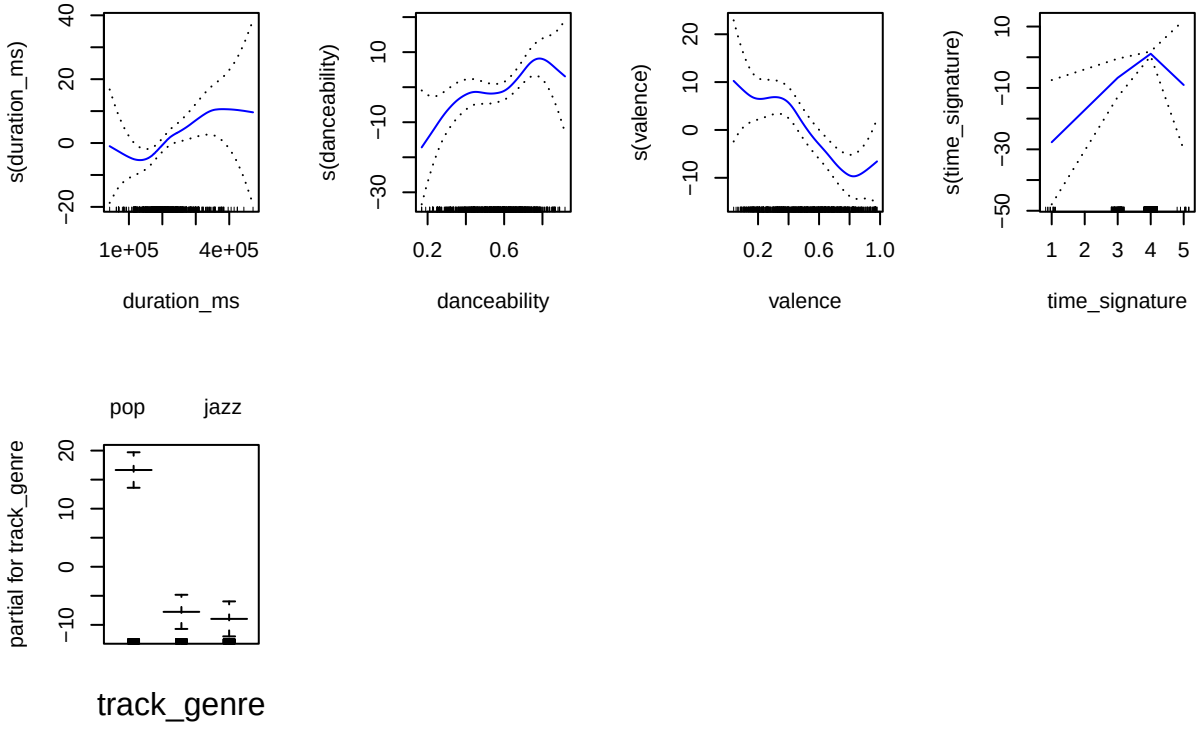
Now using the given dataset (train.df) we see the predictors and their coefficients

```r
fss.model = regsubsets(popularity ~ ., data = train.df, nvmax = 17, method = "forward")
coef(fss.model, 6)
```

```
##     (Intercept)     duration_ms    danceability         valence  time_signature
##   -2.415899e+00    7.471557e-05    2.715281e+01   -1.905143e+01    6.767200e+00
## track_genrerock track_genrejazz
##   -2.621864e+01   -2.693395e+01
```

These predictors reflect the variable importance plot from early in the report. The 6 most important predictors are duration_ms, danceability, valence, time_signature, and the track_genres (2) Interestingly enough, time_signature which was deemed the least important predictor, happens to be included in this model.

```r
#smoothing splines
model.gam = gam(popularity ~ s(duration_ms) + s(danceability) + s(valence) + s(time_signature)
                + track_genre, data = train.fss)
par(mfrow = c(2,4))
plot(model.gam, se = T, col = "blue")
```




```
pred.gam = predict(model.gam, test.fss)
fss.error = mean((pred.gam - test.fss$popularity)^2)
fss.error
```

```
## [1] 843.4284
```

This results in the lowest Test error that we have obtained yet by a significant amount

# Regression Tree

```
require(tree)
```

```
## Loading required package: tree
```

```
#Fit a regression tree to the training set
tree.pop = tree(popularity ~ ., data = train.fss)

#Summary of the model
summary(tree.pop)
```

```
##
```

```
## Regression tree:
## tree(formula = popularity ~ ., data = train.fss)
## Variables actually used in tree construction:
## [1] "track_genre"      "instrumentalness" "duration_ms"      "energy"
## [5] "acousticness"     "key"              "valence"          "speechiness"
## Number of terminal nodes:  10
## Residual mean deviance:  779.6 = 691500 / 887
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -66.660 -19.250  -5.876   0.000  17.340  80.120
```

3 of the 4 most important predictors from previously in the report appear in the tree construction

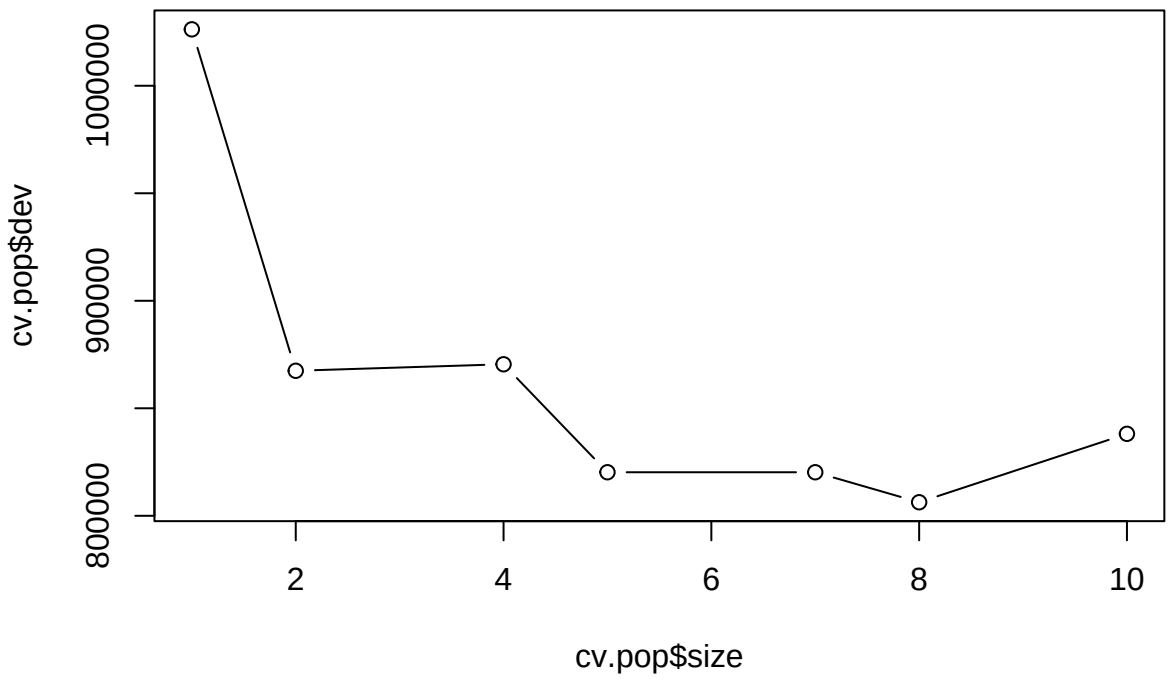
```
#predicting on test
pred = predict(tree.pop, newdata = test.fss)
regtree.error = mean((pred - y.test)^2)
regtree.error
```

```
## [1] 880.2127
```

This is the initial error prior to the pruning that is about to happen. It is still the highest error acquired yet

Using cross-validation to determine

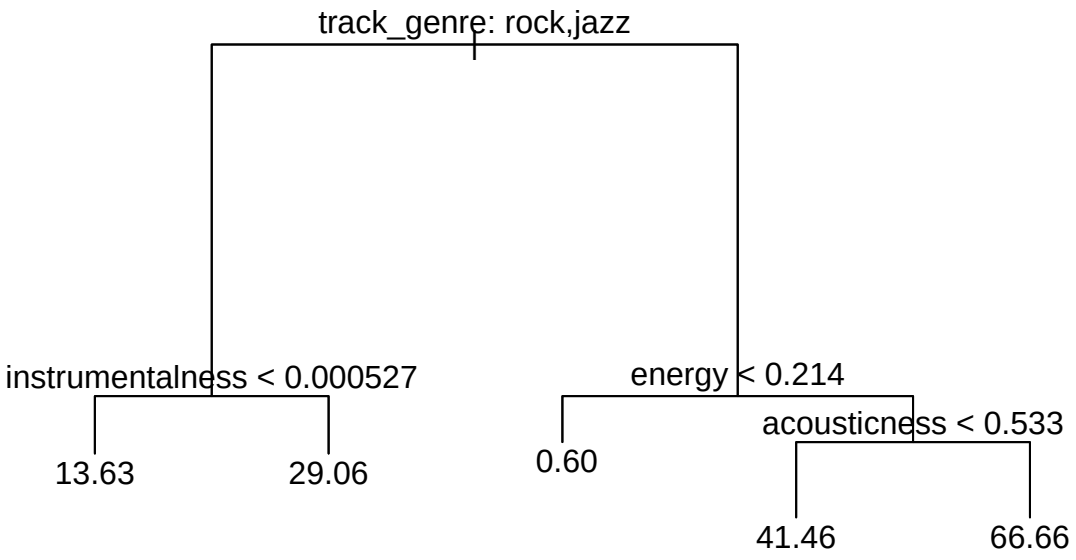
```
#Cross-Validation
cv.pop = cv.tree(tree.pop)
plot(cv.pop$size, cv.pop$dev, type = "b")
```



The optimal size of the tree is 5 via cross-validation.

## Pruned Regression Tree

```
#Prune
prune.pop = prune.tree(tree.pop, best = 5)
plot(prune.pop)
text(prune.pop, pretty=0)
```




```
#predicting on test
pred = predict(prune.pop, test.fss)
pruned.regtree.error = mean((pred-y.test)^2)
pruned.regtree.error
```

```
## [1] 937.5477
```

The pruned tree has a far worse test error than the original tree making it the highest test error yet.

## Bagging

```
pop.bag = randomForest(popularity ~., data = train.fss, mtry=15, ntree = 500, importance=TRUE)
pred.bag = predict(pop.bag, newdata =  test.fss)
bag.error = mean((pred.bag - y.test)^2)
bag.error
```

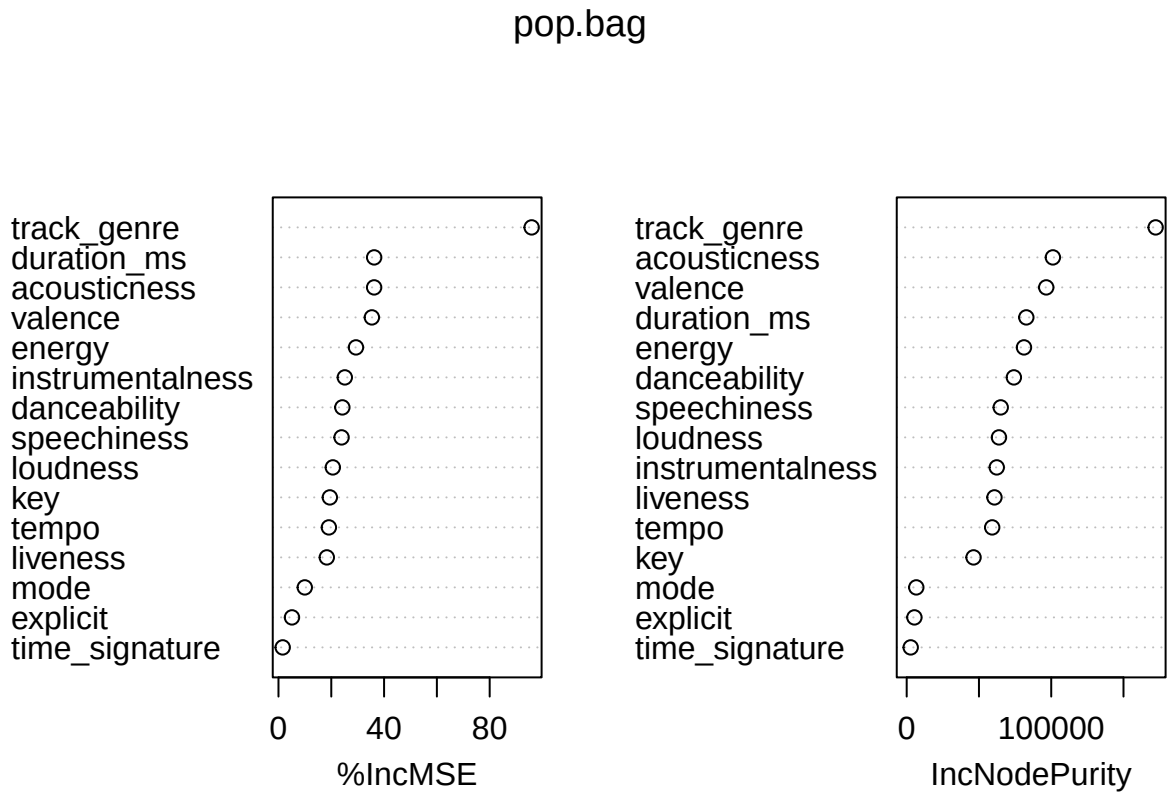
```
## [1] 669.1016
```

We have now obtained an extremely lower test error in comparison to every other model.

```
#Creating variable importance plot for analysis
importance(pop.bag)
```

```
##                    %IncMSE IncNodePurity
## duration_ms      36.238920     82763.829
## explicit          5.111236      5330.586
## danceability     24.198296     74146.769
## energy           29.351079     81150.797
## key              19.458478     46249.732
## loudness         20.590699     63834.672
## mode              9.966807      6645.249
## speechiness      23.895944     65047.851
## acousticness     36.229090    101139.724
## instrumentalness 25.125072     62299.773
## liveness         18.306546     60712.401
## valence          35.377516     96558.117
## tempo            19.092171     59147.869
## time_signature    1.600759      2765.686
## track_genre      95.888095    172247.415
```

```
varImpPlot(pop.bag)
```




The 2 distinct important variables are track_genre and valence. These are the most influential variables in determining the predictions in the model

### Boosting

I would assume this will be the best model since the linear regression model under fits the data

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.2.3
```

```
## Loaded gbm 2.1.9
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.co
```

```
#Fit gbm model
gbm_model <- gbm(popularity ~ ., data = train.fss, n.trees = 500, distribution = "gaussian")

#predicting on test
pred <- predict(gbm_model, newdata = test.fss, n.trees = 500)
boost.error <- mean((pred - y.test)^2)
boost.error
```

```
## [1] 828.4653
```

The test error is the second lowest behind bagging.

---

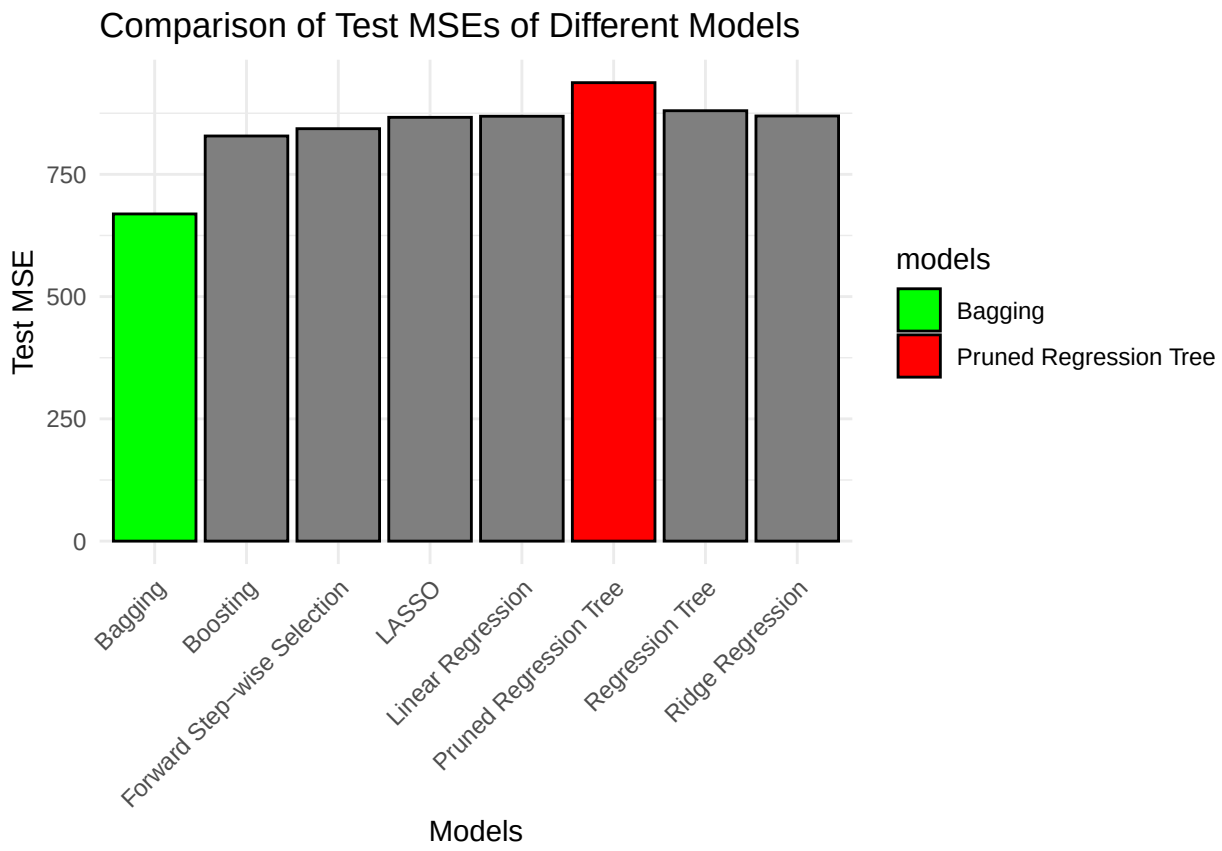
## Results and Conclusion

```
#store names for the models
models <- c("Linear Regression", "Ridge Regression", "LASSO", "Forward Step-wise Selection", "Regressio
#store test mses for each model
test.mses <- c(lm.error, ridge.error, lasso.error, fss.error, regtree.error, pruned.regtree.error,
               bag.error, boost.error)

data = data.frame(models, test.mses)

# Creating the plot
plot = ggplot(data, aes(x = models, y = test.mses, fill = models)) +
  geom_bar(stat = "identity", color = "black") +
  labs(title = "Comparison of Test MSEs of Different Models",
       x = "Models",
       y = "Test MSE") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("Pruned Regression Tree" = "red","Bagging" = "green"))

# Displaying the plot
print(plot)
```




The final model that will be used is Bagging as it has a significantly lower test error than every other model.

```
#Refit the best model using all the data from the original training set
#It was determined that bagging was the most accurate
final.model = randomForest(popularity ~., data=train.df, mtry=15, importance=TRUE)

pred.bag = predict(final.model, newdata =  test.df)

popularity.final = as.numeric(round(pred.bag), 0)
```

These are the summary statistics of the predicted popularity

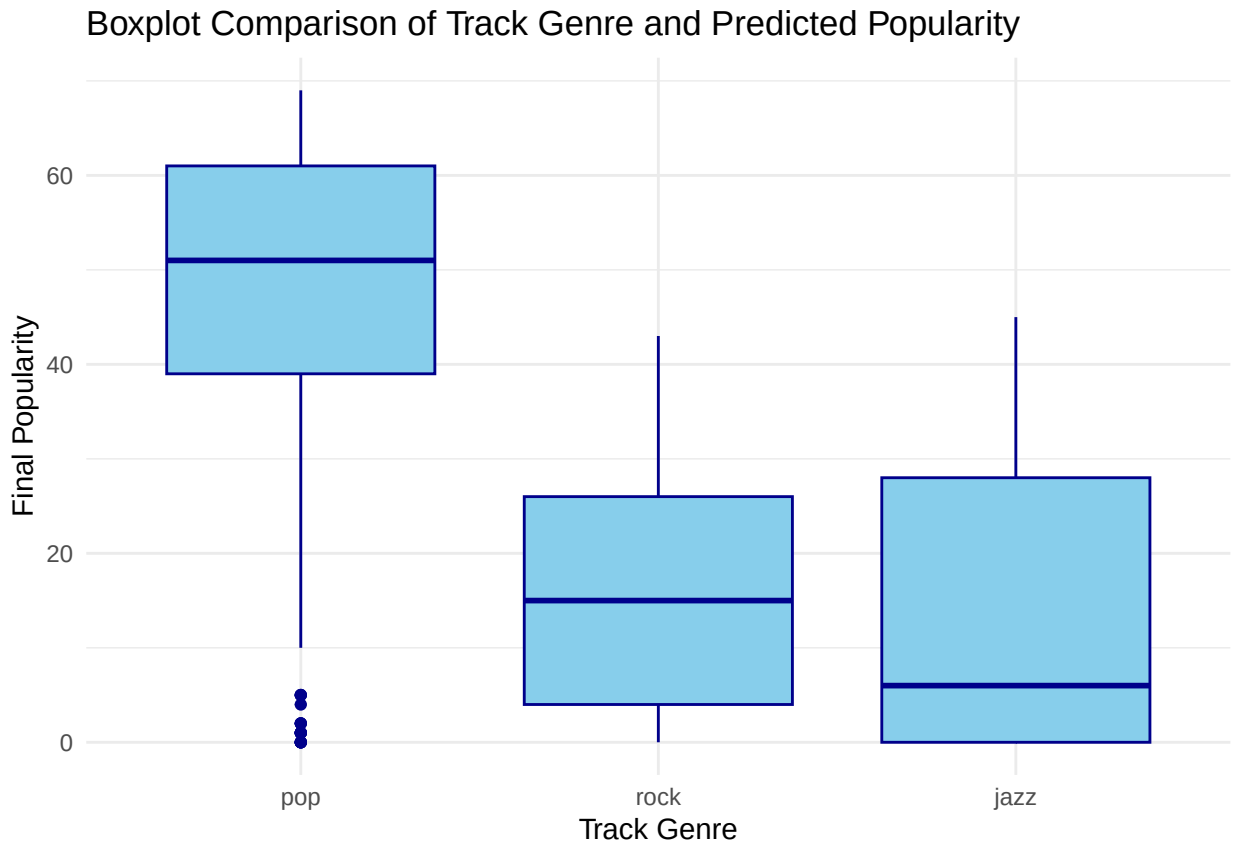
```
#Final summary statistics for the predicted popularity
summary(popularity.final)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    3.75   23.00   24.72   40.00   69.00
```

Comparison Plot

```
#Comparison Plot for Track Genre vs. final predicted popularity
ggplot(test.df, aes(x = track_genre, y = popularity.final)) +
  geom_boxplot(fill = "skyblue", color = "darkblue") +
  theme_minimal() +
  labs(
```

```
    x = "Track Genre",
    y = "Final Popularity",
    title = "Boxplot Comparison of Track Genre and Predicted Popularity")
```




Export data

```
final = data.frame(id = test.ids, popularity = popularity.final)

#Exporting to csv file
write.csv(final, file = "testing_predictions_Quinlan_Ryan_RWQ2.csv", row.names = FALSE)
```