PSTAT 131 Final Project

Alec Chen

2022-06-09

Contents

Introduction **1**
Data Codebook . . . 1

Data Preparation **2**

Exploratory Data Analysis **2**
Data Cleaning . . . 3
Data Pre-processing . . . 5
Lasso Regresion . . . 13
SVM . . . 15
Regression Tree . . . 19
Random Forest . . . 25

Introduction

I am interested in exploring what makes a song popular or not and what predictors contribute the most with EDA and thus based on that create 4 different machine learning models to predict songs' popularity. The data set consists of songs of different genres, artists, energy, liveliness, and other traits. Some of them top the chart list and become very popular and others are not so common among the crowd. I believe that each song has some unique combination of attributes that makes it popular. We are considering the data from Spotify, a popular application for listening to music. Each song has been rated on different factors in the data. With this analysis, a music company can predetermine how popular the song can come about to be. The model can also be used by companies like Spotify to predict the popularity of an upcoming song and thus suggest it to its user. To recommend new music to users, and to be able to internally classify songs, Spotify assigns each song value from 13 different features. These features are mostly numerical values but include some categorical data as well. Spotify also assigns each song a popularity score, based on the total number of clicks.

Data Collection: I found this data set from Kaggle https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db?resource=download which consists of the features of songs. There are 10,000 songs per genre. There are 26 genres so it is a total of 232,725 tracks. This size of data set could be too big and inefficient to run models, therefore, I have randomly stratified the data with a size of 5000 to make sure the data set is still able to represent large data set.

Data Codebook

Numerical: -acousticness: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. -danceability: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable. -duration_ms: The duration of the track in milliseconds. -energy: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. -liveliness: Detects the presence of an audience in the recording. -instrumentalness: Predicts whether a track contains no vocals. -loudness: The overall loudness of a track in decibels (dB) -speechiness: Speechiness detects the presence of spoken words in a track. -valence: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. -time_signature: An estimated time signature. -tempo: The overall estimated tempo of a track in beats per minute (BPM). Dummy Code: -mode: 0 = Minor, 1 = Major

Data Preparation

I loaded the necessary packages for later EDA and fit in linear regression, random forest, SVM for regression and boosted tree.

```
library(tidymodels)
library(tidyverse)
library(ISLR)
library(ISLR2)
library(discrim)
library(poissonreg)
library(corrr)
library(ggplot2)
library(corrplot)
library(ggthemes)
library(kernlab)
library(e1071)
library(caret)
library(rpart)
library(rpart.plot)
tidymodels_prefer()

#Import the Data
spotify <-read_csv("Data/SpotifyFeatures.csv")
spotify
```

```
## # A tibble: 232,725 x 18
##    genre artist_name    track_name track_id popularity acousticness danceability
##    <chr> <chr>          <chr>      <chr>         <dbl>        <dbl>        <dbl>
##  1 Movie Henri Salvador C'est bea~ 0BRjO6g~          0      0.611          0.389
##  2 Movie Martin & les ~ Perdu d'a~ 0BjC1Nf~          1      0.246          0.59
##  3 Movie Joseph Willia~ Don't Let~ 0CoSDzo~          3      0.952          0.663
##  4 Movie Henri Salvador Dis-moi M~ 0Gc6TVm~          0      0.703          0.24
##  5 Movie Fabien Nataf   Ouverture  0IuslXp~          4      0.95           0.331
##  6 Movie Henri Salvador Le petit ~ 0Mf1jKa~          0      0.749          0.578
##  7 Movie Martin & les ~ Premières~ 0NUiKYR~          2      0.344          0.703
##  8 Movie Laura Mayne    Let Me Le~ 0PbIF9Y~         15      0.939          0.416
##  9 Movie Chorus         Helka      0ST6uPf~          0      0.00104        0.734
## 10 Movie Le Club des J~ Les bisou~ 0VSqZ3K~         10      0.319          0.598
## # ... with 232,715 more rows, and 11 more variables: duration_ms <dbl>,
## #   energy <dbl>, instrumentalness <dbl>, key <chr>, liveness <dbl>,
## #   loudness <dbl>, mode <chr>, speechiness <dbl>, tempo <dbl>,
## #   time_signature <chr>, valence <dbl>
```

Exploratory Data Analysis

Data Cleaning

We have 5,000 observations and 12 variables with two classes as numeric and integers dropping somer of the missing values rows and get a summary of the data set for better regression analysis.

```
# Data Set Summary
str(spotify)
```

```
## spec_tbl_df [232,725 x 18] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ genre           : chr [1:232725] "Movie" "Movie" "Movie" "Movie" ...
##  $ artist_name     : chr [1:232725] "Henri Salvador" "Martin & les fées" "Joseph Williams" "Henri Sal
##  $ track_name      : chr [1:232725] "C'est beau de faire un Show" "Perdu d'avance (par Gad Elmaleh)"
##  $ track_id        : chr [1:232725] "0BRjO6ga9RKCKjfDqeFgWV" "0BjC1NfoEOOusryehmNudP" "0CoSDzoNIKCRs
##  $ popularity      : num [1:232725] 0 1 3 0 4 0 2 15 0 10 ...
##  $ acousticness    : num [1:232725] 0.611 0.246 0.952 0.703 0.95 0.749 0.344 0.939 0.00104 0.319 ...
##  $ danceability    : num [1:232725] 0.389 0.59 0.663 0.24 0.331 0.578 0.703 0.416 0.734 0.598 ...
##  $ duration_ms     : num [1:232725] 99373 137373 170267 152427 82625 ...
##  $ energy          : num [1:232725] 0.91 0.737 0.131 0.326 0.225 0.0948 0.27 0.269 0.481 0.705 ...
##  $ instrumentalness: num [1:232725] 0 0 0 0 0.123 0 0 0 0.00086 0.00125 ...
##  $ key             : chr [1:232725] "C#" "F#" "C" "C#" ...
##  $ liveness        : num [1:232725] 0.346 0.151 0.103 0.0985 0.202 0.107 0.105 0.113 0.0765 0.349 ..
##  $ loudness        : num [1:232725] -1.83 -5.56 -13.88 -12.18 -21.15 ...
##  $ mode            : chr [1:232725] "Major" "Minor" "Minor" "Major" ...
##  $ speechiness     : num [1:232725] 0.0525 0.0868 0.0362 0.0395 0.0456 0.143 0.953 0.0286 0.046 0.028
##  $ tempo           : num [1:232725] 167 174 99.5 171.8 140.6 ...
##  $ time_signature  : chr [1:232725] "4/4" "4/4" "5/4" "4/4" ...
##  $ valence         : num [1:232725] 0.814 0.816 0.368 0.227 0.39 0.358 0.533 0.274 0.765 0.718 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   genre = col_character(),
##   ..   artist_name = col_character(),
##   ..   track_name = col_character(),
##   ..   track_id = col_character(),
##   ..   popularity = col_double(),
##   ..   acousticness = col_double(),
##   ..   danceability = col_double(),
##   ..   duration_ms = col_double(),
##   ..   energy = col_double(),
##   ..   instrumentalness = col_double(),
##   ..   key = col_character(),
##   ..   liveness = col_double(),
##   ..   loudness = col_double(),
##   ..   mode = col_character(),
##   ..   speechiness = col_double(),
##   ..   tempo = col_double(),
##   ..   time_signature = col_character(),
##   ..   valence = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
colSums(is.na(spotify))
```

```
##            genre      artist_name       track_name         track_id
##                0                0                0                0
##       popularity     acousticness     danceability      duration_ms
##                0                0                0                0
```

```
##           energy instrumentalness              key         liveness
##                0                0                0                0
##         loudness             mode      speechiness            tempo
##                0                0                0                0
##   time_signature          valence
##                0                0
```

```
spotify <- drop_na(spotify)
summary(spotify)
```

```
##     genre            artist_name         track_name          track_id
##  Length:232725      Length:232725      Length:232725      Length:232725
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##    popularity      acousticness     danceability     duration_ms
##  Min.   :  0.00   Min.   :0.0000   Min.   :0.0569   Min.   :  15387
##  1st Qu.: 29.00   1st Qu.:0.0376   1st Qu.:0.4350   1st Qu.: 182857
##  Median : 43.00   Median :0.2320   Median :0.5710   Median : 220427
##  Mean   : 41.13   Mean   :0.3686   Mean   :0.5544   Mean   : 235122
##  3rd Qu.: 55.00   3rd Qu.:0.7220   3rd Qu.:0.6920   3rd Qu.: 265768
##  Max.   :100.00   Max.   :0.9960   Max.   :0.9890   Max.   :5552917
##      energy         instrumentalness       key               liveness
##  Min.   :2.03e-05   Min.   :0.0000000   Length:232725      Min.   :0.00967
##  1st Qu.:3.85e-01   1st Qu.:0.0000000   Class :character   1st Qu.:0.09740
##  Median :6.05e-01   Median :0.0000443   Mode  :character   Median :0.12800
##  Mean   :5.71e-01   Mean   :0.1483012                      Mean   :0.21501
##  3rd Qu.:7.87e-01   3rd Qu.:0.0358000                      3rd Qu.:0.26400
##  Max.   :9.99e-01   Max.   :0.9990000                      Max.   :1.00000
##     loudness            mode            speechiness         tempo
##  Min.   :-52.457   Length:232725      Min.   :0.0222   Min.   : 30.38
##  1st Qu.:-11.771   Class :character   1st Qu.:0.0367   1st Qu.: 92.96
##  Median : -7.762   Mode  :character   Median :0.0501   Median :115.78
##  Mean   : -9.570                      Mean   :0.1208   Mean   :117.67
##  3rd Qu.: -5.501                      3rd Qu.:0.1050   3rd Qu.:139.05
##  Max.   :  3.744                      Max.   :0.9670   Max.   :242.90
##  time_signature        valence
##  Length:232725      Min.   :0.0000
##  Class :character   1st Qu.:0.2370
##  Mode  :character   Median :0.4440
##                     Mean   :0.4549
##                     3rd Qu.:0.6600
##                     Max.   :1.0000
```

With the unprocessed data set we need to rename some of the columns for better access, change the class of the explicit column from character to numeric and creates subset of data set for correlation matrics.

```
spotify<- spotify %>% select("danceability","energy","loudness","mode","speechiness",
              "acousticness","instrumentalness","liveness","valence","tempo","duration_ms","popularity"
spotify <- spotify %>%
  mutate(mode = as.numeric(case_when(
    (mode == "Major") ~ "1",
    (mode == "Minor")~ "0")))
```

Data Pre-processing

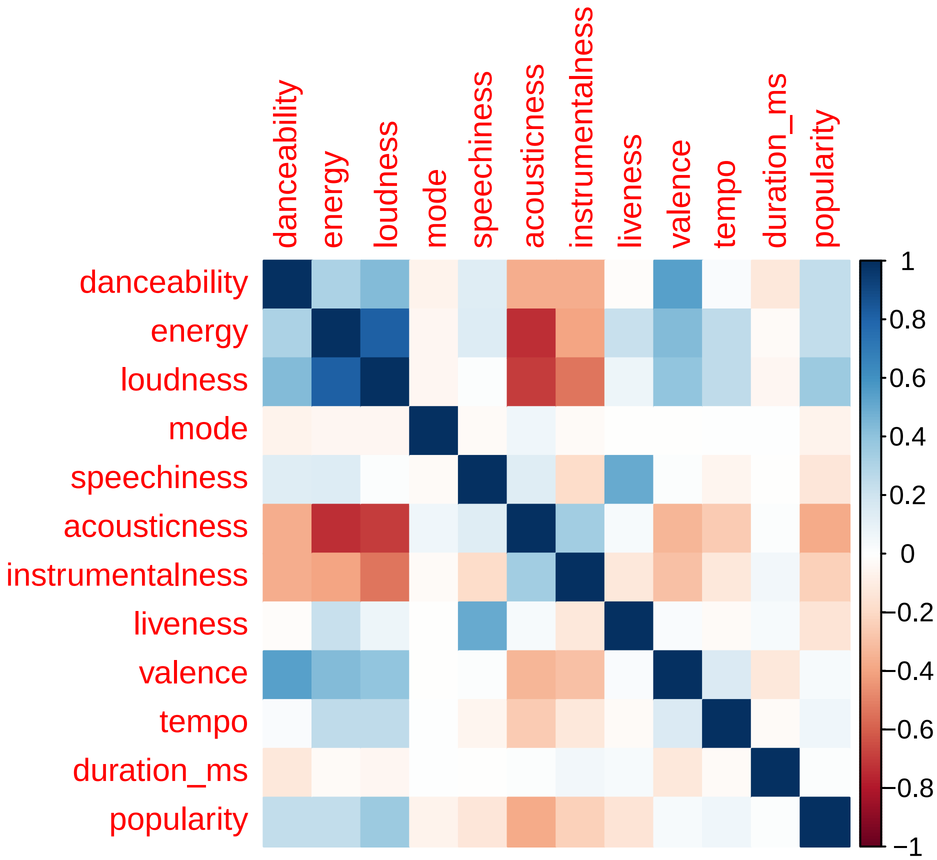
We splitting the data set into two, a training set and a testing set and also perform a cross-fold validation.

```
# Data Train-Test Split
set.seed(111)
# Reduced data size for efficiency
spotify <- spotify[sample(1:nrow(spotify), 5000,
   replace=FALSE),]
# Split data set 80% training 20% testing
spotify_split <- initial_split(spotify, prop = 0.80,
                                strata = popularity)
spotify_train <- training(spotify_split)
spotify_test <- testing(spotify_split)
# Cross fold validation with 10 folds
spotify_folds <- vfold_cv(spotify_train, v = 10)
spotify_folds
```

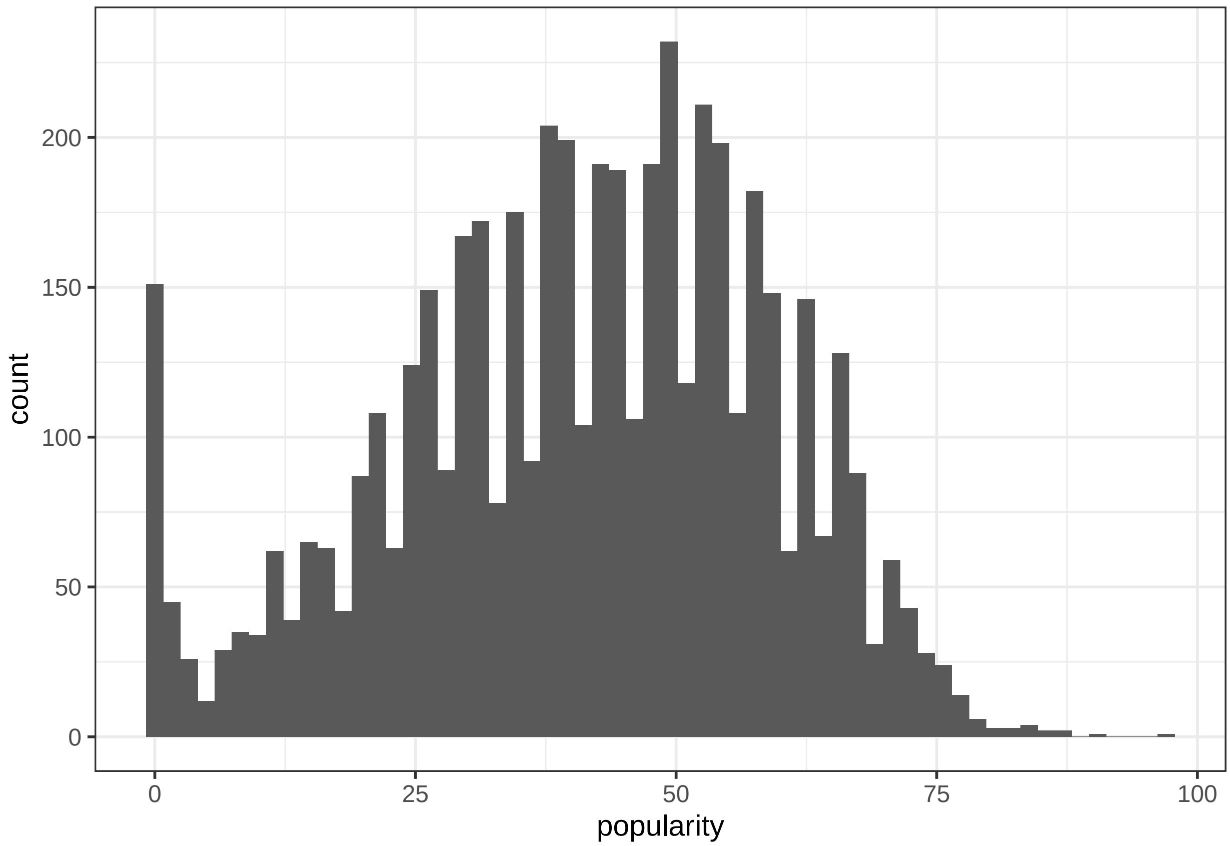
```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [3600/400]> Fold01
##  2 <split [3600/400]> Fold02
##  3 <split [3600/400]> Fold03
##  4 <split [3600/400]> Fold04
##  5 <split [3600/400]> Fold05
##  6 <split [3600/400]> Fold06
##  7 <split [3600/400]> Fold07
##  8 <split [3600/400]> Fold08
##  9 <split [3600/400]> Fold09
## 10 <split [3600/400]> Fold10
```

With the correlation matrics of the features we find that most of the features have little or no correlation with one another beside the loudness and energy have a stronger positive correlation and acousticness and energy have a negative correlation. And, the histogram gives us the visualization of the popularity distribution within the data set. With Principal Components Regression we are able to see the intercept term in the test RMSE is 18.11 if we add in the first principal component it drops to 17.14.We can see that adding additional principal components actually leads to an increase in test RMSE. Thus, it appears that it would be optimal to only use two principal components in the final model.For the variance explained by using just the first principal component, we can explain 31.84% of the variation in the response variable.by adding in the second principal component, we can explain 46.21% of the variation in the response variable.We can explain more variance by using more principal components.

```
#Correlation of the predictors
corrplot::corrplot(cor(spotify),  method = 'color')
```




```r
# Data set song's popularity distribution
spotify %>%
  ggplot(aes(x = popularity)) +
  geom_histogram(bins = 60) +
  theme_bw()
```

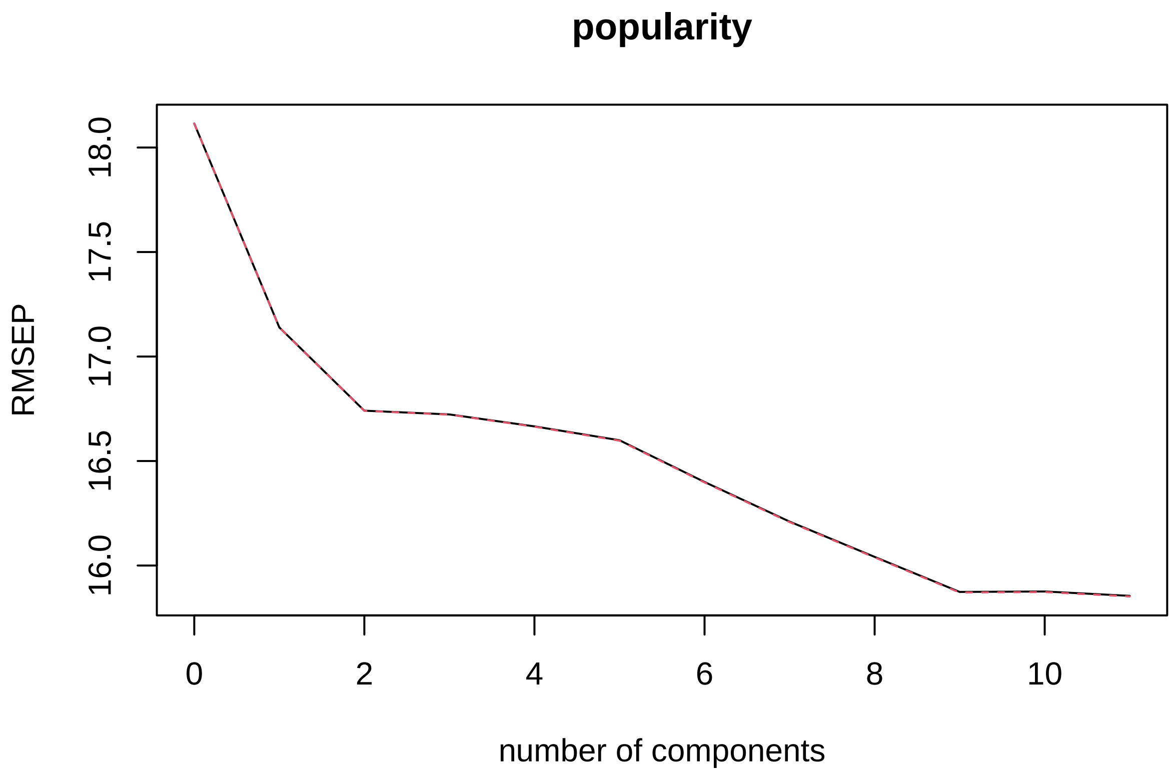



```r
#PCR
set.seed(111)
#fit PCR model
library(pls)
PCR_model <- pcr(popularity~., data=spotify, scale=TRUE, validation="CV")
summary(PCR_model)
```

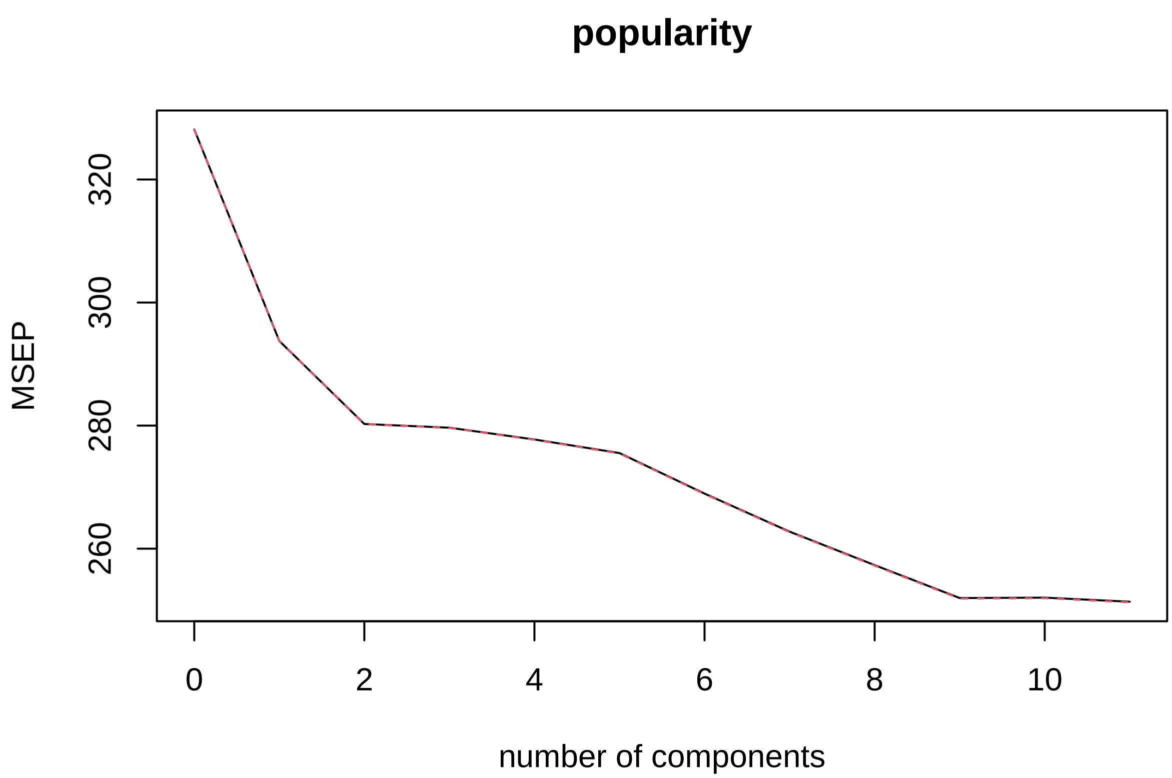
```
## Data:    X dimension: 5000 11
##  Y dimension: 5000 1
## Fit method: svdpc
## Number of components considered: 11
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           18.11    17.14    16.74    16.72    16.67     16.6     16.4
## adjCV        18.11    17.14    16.74    16.72    16.66     16.6     16.4
##        7 comps  8 comps  9 comps  10 comps  11 comps
## CV       16.21    16.04    15.87     15.88     15.85
## adjCV    16.21    16.04    15.87     15.87     15.85
##
## TRAINING: % variance explained
##             1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X             31.84    46.21    56.82    66.07    74.65    81.92    88.25
## popularity    10.57    14.71    14.99    15.60    16.45    18.39    20.36
##             8 comps  9 comps  10 comps  11 comps
## X             92.79    96.44     98.94    100.00
```

```
## popularity     22.02     23.67      23.67      23.92
```

```
#visualize cross-validation plots
validationplot(PCR_model)
```




```
validationplot(PCR_model, val.type="MSEP")
```



#

Model Building ## Linear Regression With the simple linear regression we can see that it predicts pretty poorly that the data points hardly follow a diagonal line. With the high of RMSE 15.68 and the low of rsq explaining 25% of the variance.

```r
## Linear Regression
# Create a Recipe
spotify_recipe <- recipe(popularity ~ ., data = spotify_train) %>%
  step_dummy(all_nominal_predictors())

lm_model <- linear_reg() %>%
  set_engine("lm")
lm_wflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(spotify_recipe)

lm_fit <- fit(lm_wflow, spotify_train)
lm_fit %>%
  # This returns the parsnip object:
  extract_fit_parsnip() %>%
  # Now tidy the linear model object:
  tidy()
```

```
## # A tibble: 12 x 5
##    term                estimate  std.error statistic   p.value
##    <chr>                  <dbl>      <dbl>     <dbl>     <dbl>
##  1 (Intercept)       58.0        2.62          22.2  8.09e-103
##  2 danceability      16.4        1.86           8.83 1.54e- 18
##  3 energy            -4.04       2.10          -1.92 5.44e-  2
##  4 loudness           0.675      0.0836         8.07 9.17e- 16
##  5 mode              -1.03       0.528         -1.95 5.08e-  2
##  6 speechiness       -7.44       1.74          -4.28 1.88e-  5
##  7 acousticness     -12.1        1.17         -10.4  5.61e- 25
##  8 instrumentalness  -4.66       1.01          -4.61 4.11e-  6
##  9 liveness         -11.1        1.54          -7.16 9.63e- 13
## 10 valence          -15.8        1.26         -12.5  5.00e- 35
## 11 tempo             -0.0147     0.00857       -1.71 8.71e-  2
## 12 duration_ms        0.00000332 0.00000196     1.70 8.92e-  2
```

```r
lm_fitt <- fit(lm_wflow, spotify_test)
lm_fitt %>%
  # This returns the parsnip object:
  extract_fit_parsnip() %>%
  # Now tidy the linear model object:
  tidy()
```

```
## # A tibble: 12 x 5
##    term                estimate  std.error statistic  p.value
##    <chr>                  <dbl>      <dbl>     <dbl>    <dbl>
##  1 (Intercept)       58.3        5.23          11.1   2.81e-27
##  2 danceability      18.9        3.63           5.20  2.47e- 7
##  3 energy            -8.79       4.20          -2.09  3.65e- 2
##  4 loudness           0.759      0.162          4.69  3.19e- 6
##  5 mode              -2.27       1.06          -2.15  3.19e- 2
##  6 speechiness       -5.59       3.37          -1.66  9.74e- 2
##  7 acousticness     -12.4        2.36          -5.27  1.72e- 7
```

```
##  8 instrumentalness  -6.15       1.97          -3.12  1.87e- 3
##  9 liveness          -8.05       3.06          -2.63  8.65e- 3
## 10 valence          -12.1        2.50          -4.82  1.66e- 6
## 11 tempo             -0.0171     0.0174        -0.985 3.25e- 1
## 12 duration_ms        0.00000614 0.00000498     1.23  2.18e- 1
```

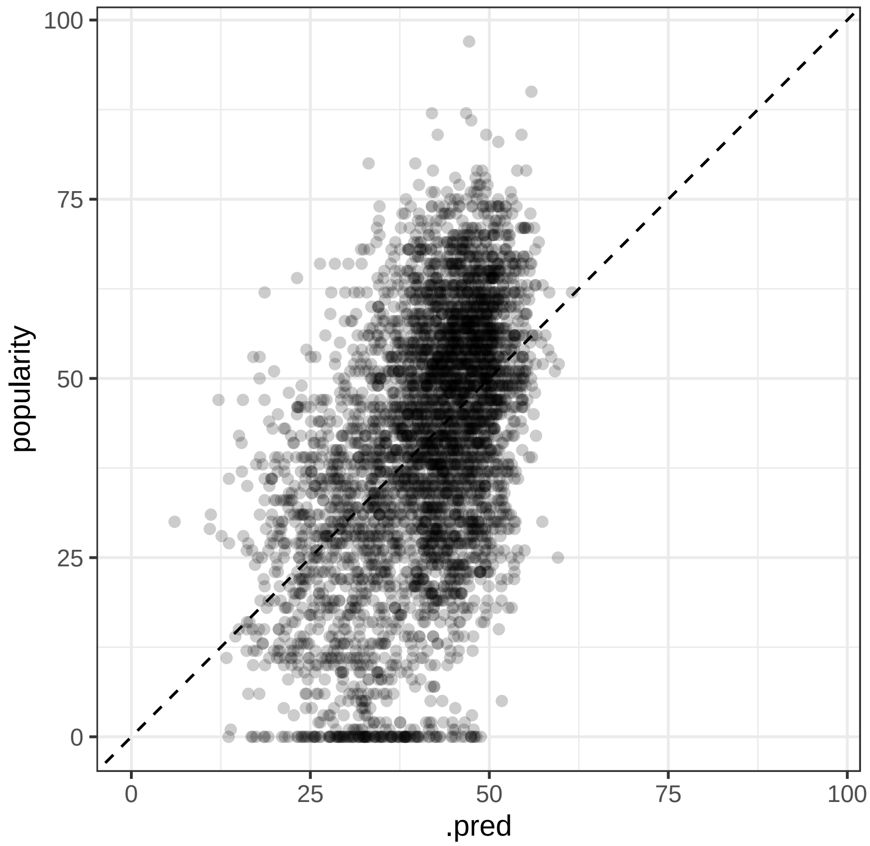
```
# Train Recipe
spotify_train_res <- predict(lm_fit, new_data = spotify_train %>% select(-popularity))
spotify_train_res %>%
  head()
```

```
## # A tibble: 6 x 1
##    .pred
##    <dbl>
## 1  42.9
## 2  41.1
## 3  47.9
## 4  42.4
## 5  48.9
## 6  44.0
```

```
# Test Recipe
spotify_train_res <- bind_cols(spotify_train_res, spotify_train %>% select(popularity))
spotify_train_res %>%
  head()
```

```
## # A tibble: 6 x 2
##    .pred popularity
##    <dbl>      <dbl>
## 1  42.9         26
## 2  41.1         19
## 3  47.9         27
## 4  42.4         29
## 5  48.9         24
## 6  44.0         28
```

```
spotify_train_res %>%
  ggplot(aes(x = .pred, y = popularity)) +
  geom_point(alpha = 0.2) +
  geom_abline(lty = 2) +
  theme_bw() +
  coord_obs_pred()
```




```
rmse(spotify_train_res, truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        15.8
```

```
spotify_metrics <- metric_set(rmse, rsq, mae)
spotify_metrics(spotify_train_res, truth = popularity,
                estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      15.8
## 2 rsq     standard       0.236
## 3 mae     standard      12.7
```

```
spotify_test_res <- predict(lm_fitt, new_data = spotify_test %>% select(-popularity))
spotify_test_res %>%
  head()
```

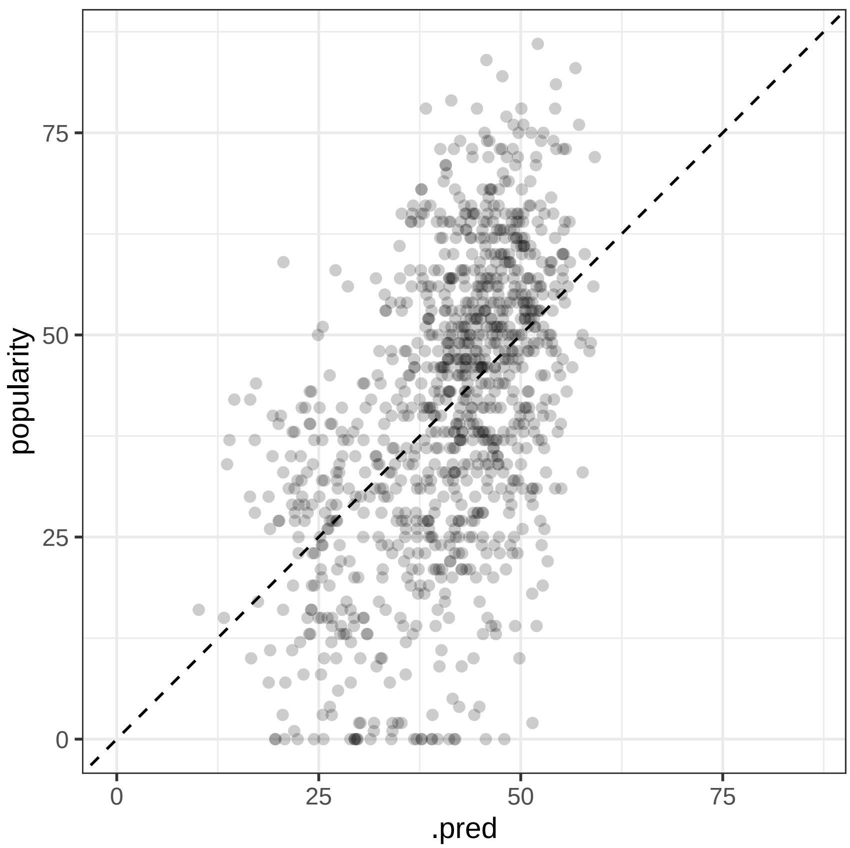
```
## # A tibble: 6 x 1
##   .pred
##   <dbl>
## 1  47.9
## 2  41.6
## 3  41.2
## 4  49.6
```

```
## 5  36.1
## 6  48.5
```

```
spotify_test_res <- bind_cols(spotify_test_res, spotify_test %>% select(popularity))
spotify_test_res %>%
  head()
```

```
## # A tibble: 6 x 2
##    .pred popularity
##    <dbl>      <dbl>
## 1  47.9         37
## 2  41.6         32
## 3  41.2         43
## 4  49.6         72
## 5  36.1         40
## 6  48.5         47
```

```
spotify_test_res %>%
  ggplot(aes(x = .pred, y = popularity)) +
  geom_point(alpha = 0.2) +
  geom_abline(lty = 2) +
  theme_bw() +
  coord_obs_pred()
```




```
rmse(spotify_test_res, truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##    .metric .estimator .estimate
##    <chr>   <chr>          <dbl>
```

```
## 1 rmse    standard        15.7
```

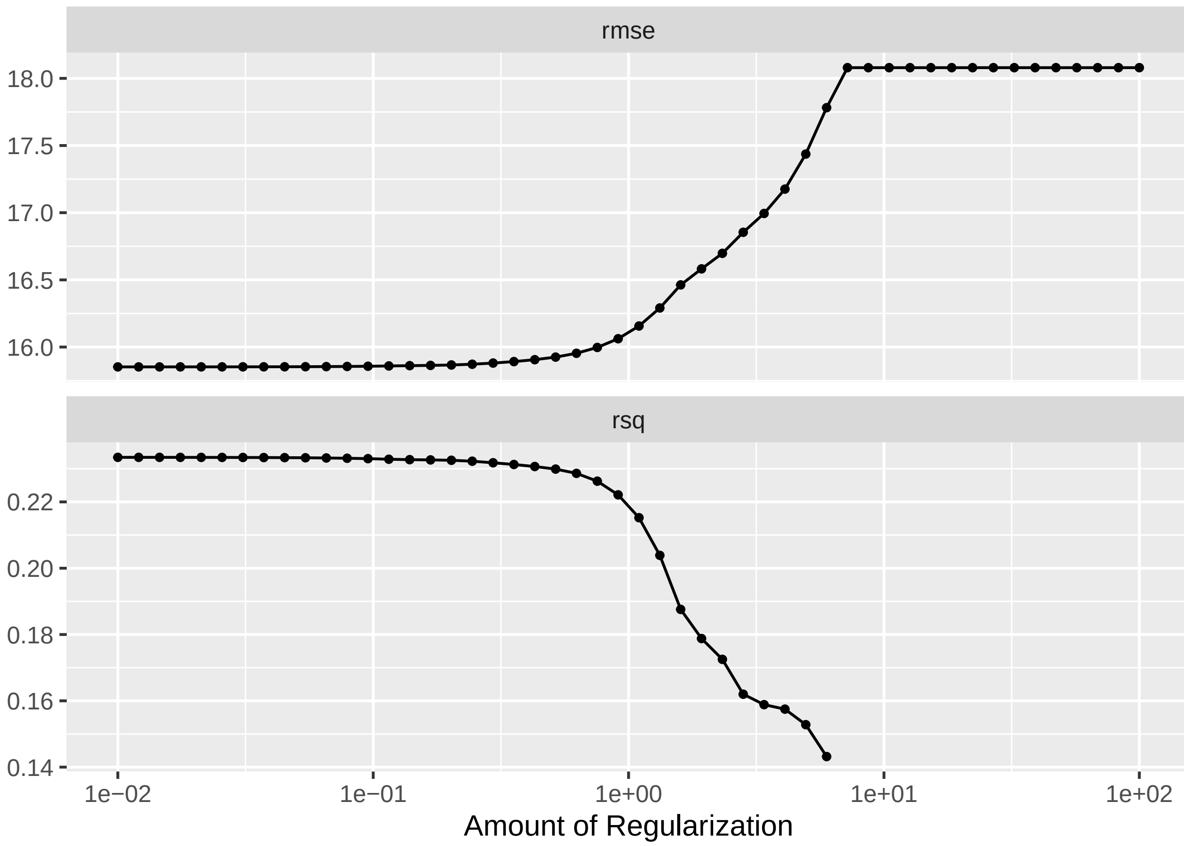
```
spotify_metrics <- metric_set(rmse, rsq, mae)
spotify_metrics(spotify_test_res, truth = popularity,
                estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      15.7
## 2 rsq     standard       0.259
## 3 mae     standard      12.5
```

Lasso Regresion

To make sure that the linear regression might not be the best for predicting this data set, we use lasso regression to see if it provides a better prediction accuracy. In comparison with the simple linear regression we still see that it predicts pretty poorly with the high of RMSE 15.76 and the low of rsq explaining 25% of the variance.

```
# Lasso Regression
lasso_recipe <-
  recipe(formula = popularity ~ ., data = spotify_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

lasso_spec <-
  linear_reg(penalty = tune(), mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_workflow <- workflow() %>%
  add_recipe(lasso_recipe) %>%
  add_model(lasso_spec)
penalty_grid <- grid_regular(penalty(range = c(-2, 2)), levels = 50)

tune_res <- tune_grid(
  lasso_workflow,
  resamples = spotify_folds,
  grid = penalty_grid
)
autoplot(tune_res)
```




```
collect_metrics(tune_res)
```

```
## # A tibble: 100 x 7
##    penalty .metric .estimator   mean     n std_err .config
##      <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1  0.01   rmse    standard   15.9      10 0.178   Preprocessor1_Model01
##  2  0.01   rsq     standard    0.233    10 0.00984 Preprocessor1_Model01
##  3  0.0121 rmse    standard   15.9      10 0.178   Preprocessor1_Model02
##  4  0.0121 rsq     standard    0.233    10 0.00984 Preprocessor1_Model02
##  5  0.0146 rmse    standard   15.9      10 0.178   Preprocessor1_Model03
##  6  0.0146 rsq     standard    0.233    10 0.00984 Preprocessor1_Model03
##  7  0.0176 rmse    standard   15.9      10 0.178   Preprocessor1_Model04
##  8  0.0176 rsq     standard    0.233    10 0.00984 Preprocessor1_Model04
##  9  0.0212 rmse    standard   15.9      10 0.178   Preprocessor1_Model05
## 10  0.0212 rsq     standard    0.233    10 0.00983 Preprocessor1_Model05
## # ... with 90 more rows
```

```
best_penalty <- select_best(tune_res, metric = "rsq")
best_penalty
```

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1  0.0146 Preprocessor1_Model03
```

```
lasso_final <- finalize_workflow(lasso_workflow, best_penalty)

lasso_final_fit <- fit(lasso_final, data = spotify_train)
```

```r
augment(lasso_final_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        15.8
```

```r
augment(lasso_final_fit, new_data = spotify_test) %>%
  rsq(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.251
```

```r
augment(lasso_final_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        15.8
```

SVM

With SVM models we start off with just fitting into radial model the RMSE was high as 15 and with the hyper parameters tuning of scaling and centering the rmse slighly decreases then with the cross validation has a more significant decreases on RMSE.

```r
#SVM
#Preprocessing Model1
modelsvm = svm(spotify$popularity~.,spotify)
set.seed(1)
model1 <- train(
  popularity~.,
  data = spotify,
  method = 'svmRadial'
)
model1
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 5000 samples
##   11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5000, 5000, 5000, 5000, 5000, 5000, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  15.05170  0.3168386  11.81862
##   0.50  15.07151  0.3167673  11.82390
##   1.00  15.14422  0.3132502  11.87142
##
```

```
## Tuning parameter 'sigma' was held constant at a value of 0.08594709
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.08594709 and C = 0.25.
```

```r
#Preprocessing Model2
set.seed(1)

model2 <- train(
  popularity~.,
  data = spotify,
  method = 'svmRadial',
  preProcess = c("center", "scale")
)
model2
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 5000 samples
##   11 predictor
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5000, 5000, 5000, 5000, 5000, 5000, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  15.05170  0.3168386  11.81862
##   0.50  15.07151  0.3167673  11.82390
##   1.00  15.14422  0.3132502  11.87142
##
## Tuning parameter 'sigma' was held constant at a value of 0.08594709
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.08594709 and C = 0.25.
```

```r
# Splitting data
set.seed(1)

inTraining <- createDataPartition(spotify$popularity, p = .80, list = FALSE)
training <- spotify[inTraining,]
testing  <- spotify[-inTraining,]
set.seed(1)

model3 <- train(
  popularity ~ .,
  data = training,
  method = 'svmRadial',
  preProcess = c("center", "scale")
)
model3
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4000 samples
##   11 predictor
##
## Pre-processing: centered (11), scaled (11)
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4000, 4000, 4000, 4000, 4000, 4000, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  14.97092  0.3159534  11.74475
##   0.50  15.00488  0.3150187  11.77565
##   1.00  15.10408  0.3096636  11.86376
##
## Tuning parameter 'sigma' was held constant at a value of 0.08991346
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.08991346 and C = 0.25.
```

```r
# calculate the RMSE and r2 to compare to the model above.
test.features = subset(testing, select=-c(popularity))
test.target = subset(testing, select=popularity)[,1]

predictions = predict(model3, newdata = test.features)

# RMSE
sqrt(mean((test.target - predictions)^2))
```

```
## [1] NA
```

```r
# R2
cor(test.target, predictions) ^ 2
```

```
##                   [,1]
## popularity 0.3162256
```

```r
#Cross Validation
set.seed(1)
ctrl <- trainControl(
  method = "cv",
  number = 10,
)

set.seed(1)

model4 <- train(
 popularity ~ .,
  data = training,
  method = 'svmRadial',
  preProcess = c("center", "scale"),
  trCtrl = ctrl
)
model4
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4000 samples
##   11 predictor
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4000, 4000, 4000, 4000, 4000, 4000, ...
## Resampling results across tuning parameters:
```

```
##
##   C     RMSE      Rsquared   MAE
##   0.25  14.97092  0.3159534  11.74475
##   0.50  15.00488  0.3150187  11.77565
##   1.00  15.10408  0.3096636  11.86376
##
## Tuning parameter 'sigma' was held constant at a value of 0.08991346
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.08991346 and C = 0.25.
```

```r
# calculate the RMSE and r2 to compare to the model above.
test.features = subset(testing, select=-c(popularity))
test.target = subset(testing, select=popularity)[,1]

predictions = predict(model4, newdata = test.features)

# RMSE
sqrt(mean((test.target - predictions)^2))
```

```
## [1] NA
```

```r
# R2
cor(test.target, predictions) ^ 2
```

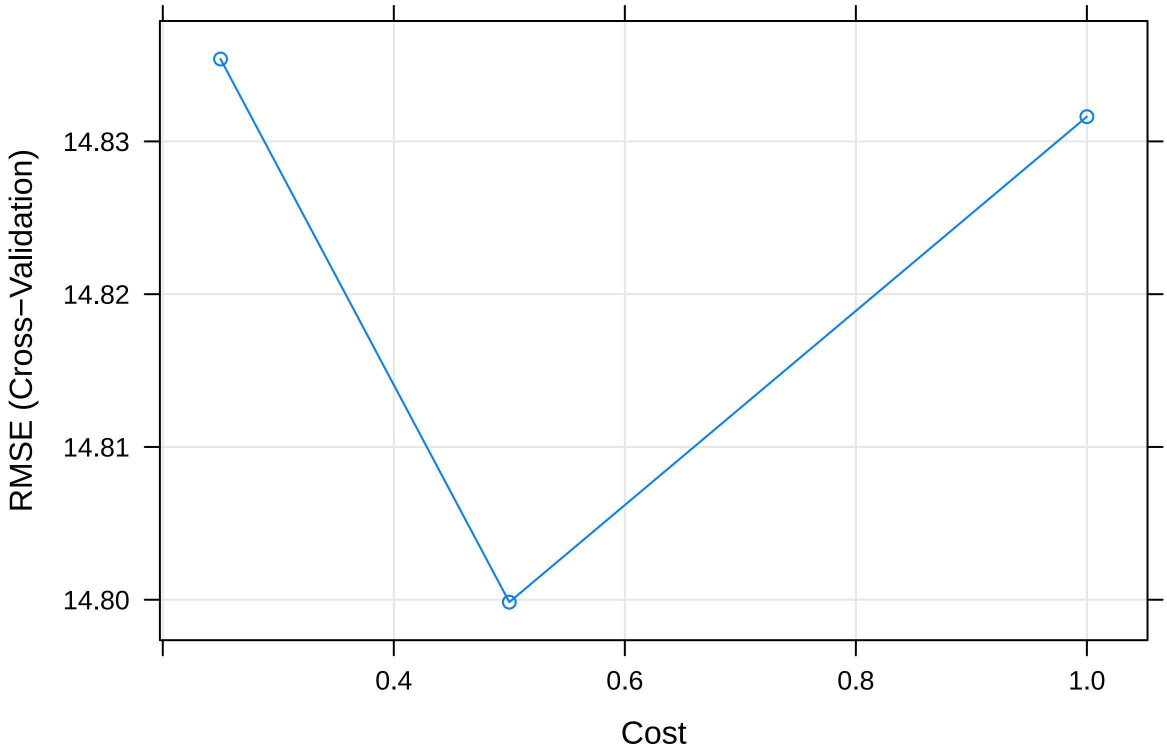
```
##                 [,1]
## popularity 0.3162256
```

```r
#Tuning Hyper Parameters
set.seed(1)

tuneGrid <- expand.grid(
  C = c(0.25, .5, 1),
  sigma = 0.1
)

model5 <- train(
 popularity ~ .,
  data = training,
  method = 'svmRadial',
  preProcess = c("center", "scale"),
  trControl = ctrl,
  tuneGrid = tuneGrid
)
model5
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4000 samples
##   11 predictor
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3600, 3601, 3601, 3600, 3599, 3601, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  14.83539  0.3312740  11.63474
```

```
##   0.50  14.79984  0.3347396  11.60035
##   1.00  14.83162  0.3331954  11.63081
##
## Tuning parameter 'sigma' was held constant at a value of 0.1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1 and C = 0.5.
```

```
plot(model5)
```



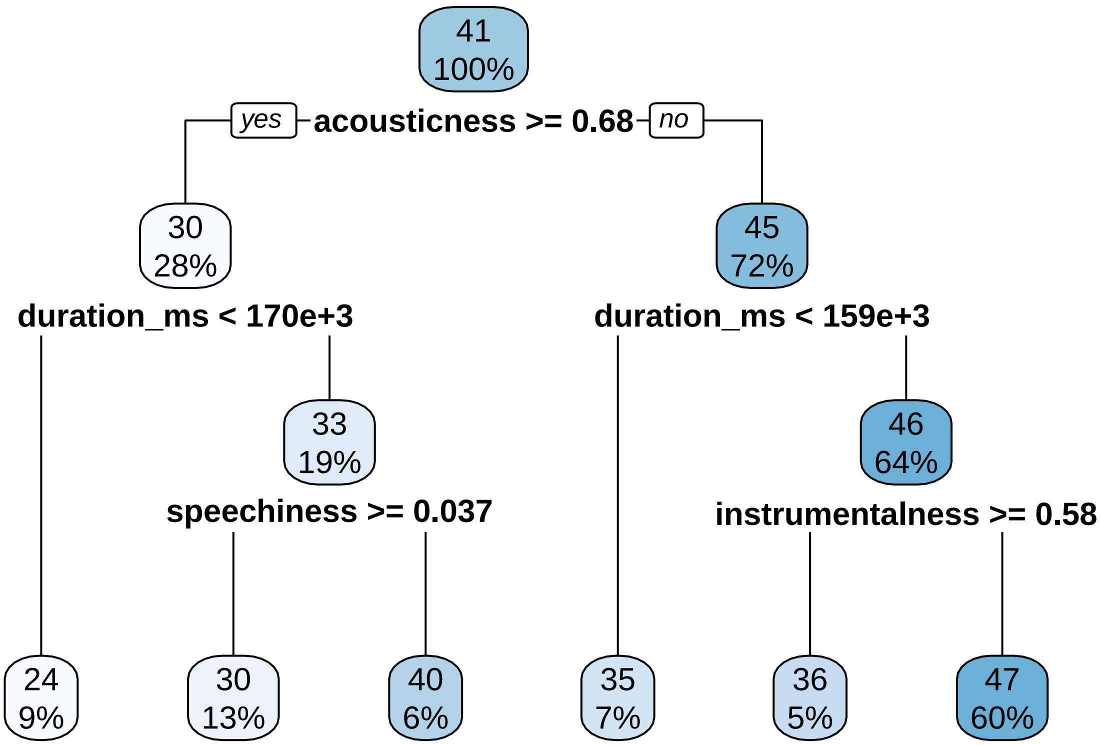
Regression Tree

```
#M1
tree_ml1 <- rpart(popularity ~ .,
             method = "anova", data = spotify_train)
tree_ml1
```

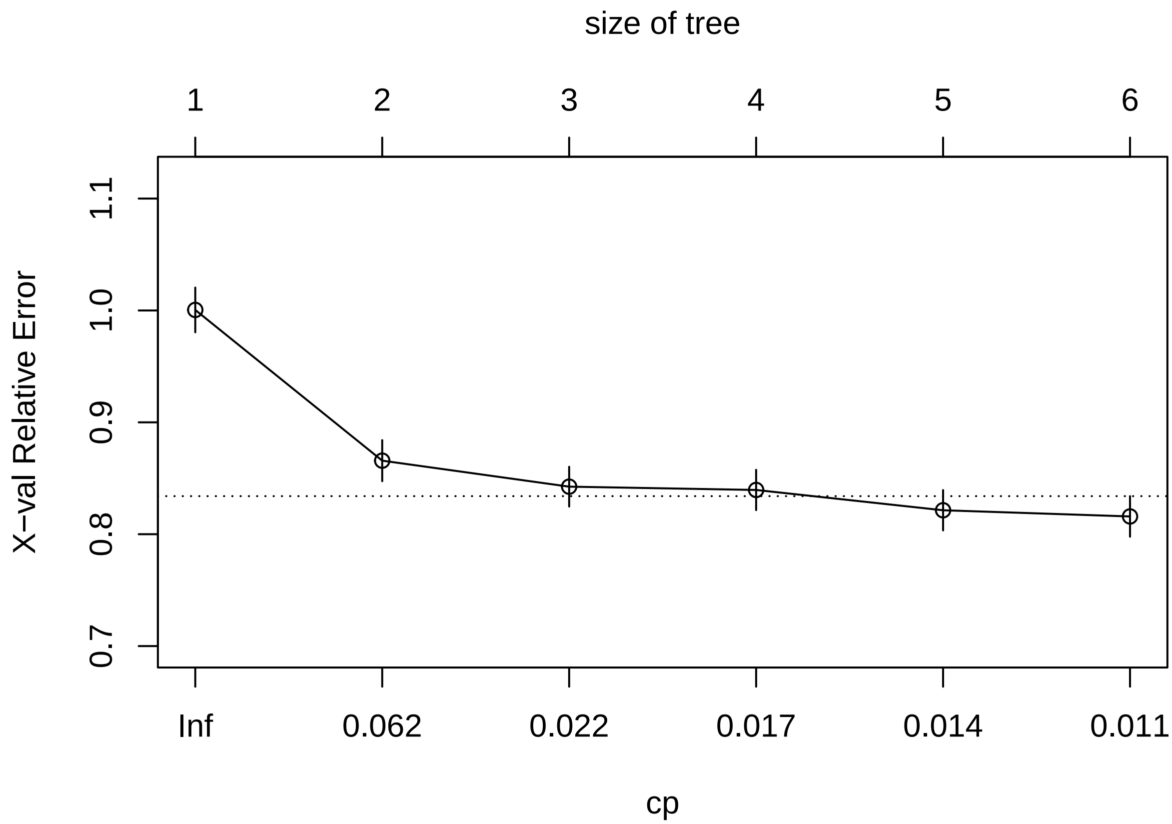
```
## n= 4000
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 4000 1308097.00 40.94250
##    2) acousticness>=0.6795 1133  346563.30 30.10591
##      4) duration_ms< 170300 375   95025.80 23.73067 *
##      5) duration_ms>=170300 758  228755.80 33.25989
##       10) speechiness>=0.03665 529  154602.00 30.30435 *
##       11) speechiness< 0.03665 229   58858.25 40.08734 *
##    3) acousticness< 0.6795 2867  775903.90 45.22497
##      6) duration_ms< 158539 288  117154.70 34.69792 *
##      7) duration_ms>=158539 2579  623269.20 46.40054
```

```
##        14) instrumentalness>=0.5845 189   25655.25 36.35450 *
##        15) instrumentalness< 0.5845 2390  577031.10 47.19498 *
```

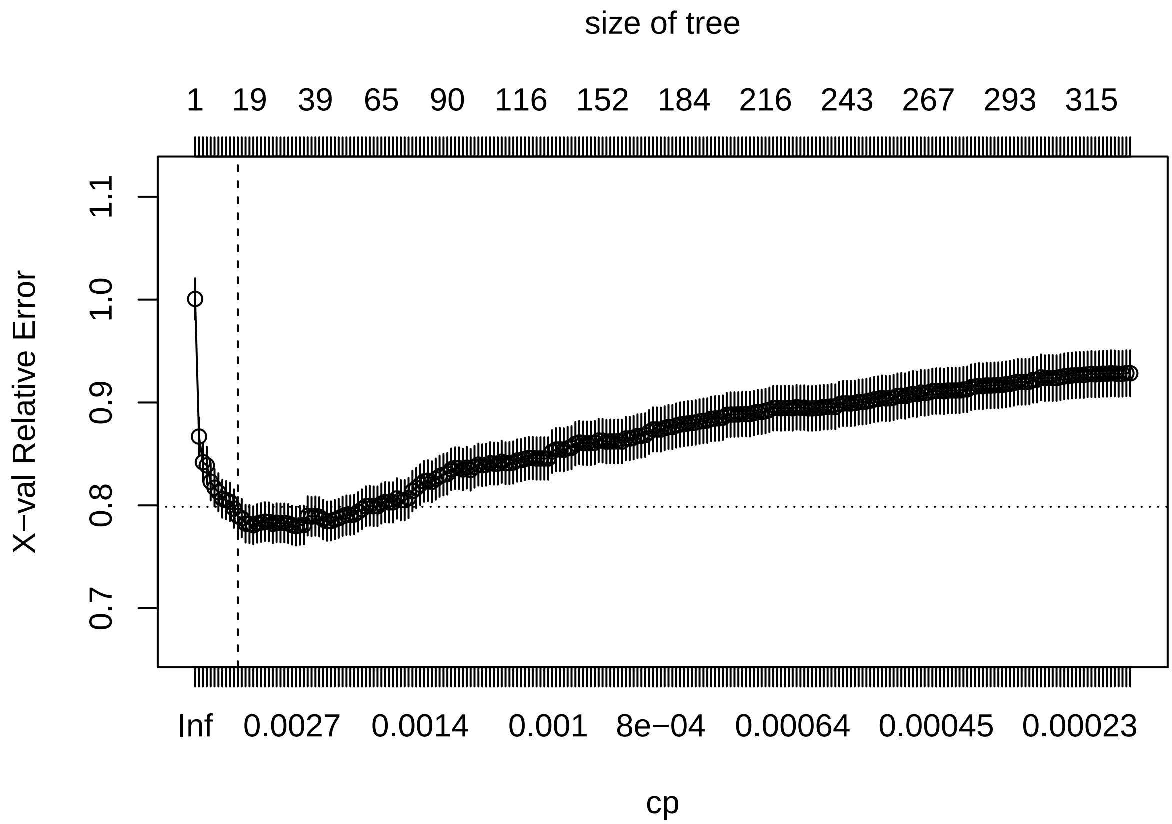
```
rpart.plot(tree_ml1)
```




```
plotcp(tree_ml1)
```




```
#M2
tree_ml2 <- rpart(
    formula = popularity ~ .,
    data    = spotify_train,
    method  = "anova",
    control = list(cp = 0, xval = 10)
)

plotcp(tree_ml2)
abline(v = 12, lty = "dashed")
```

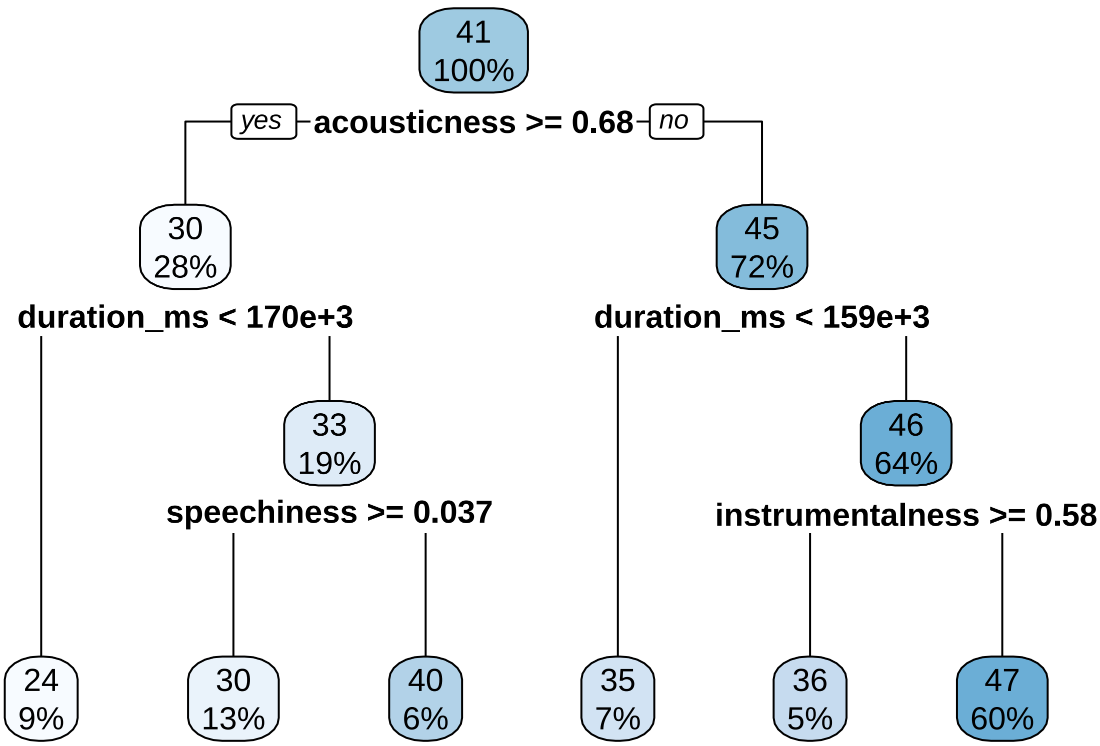



```
#M3
m3 <- rpart(
    formula = popularity ~ .,
    data    = spotify_train,
    method  = "anova",
    control = list(minsplit = 10, maxdepth = 12, xval = 10)
)
```

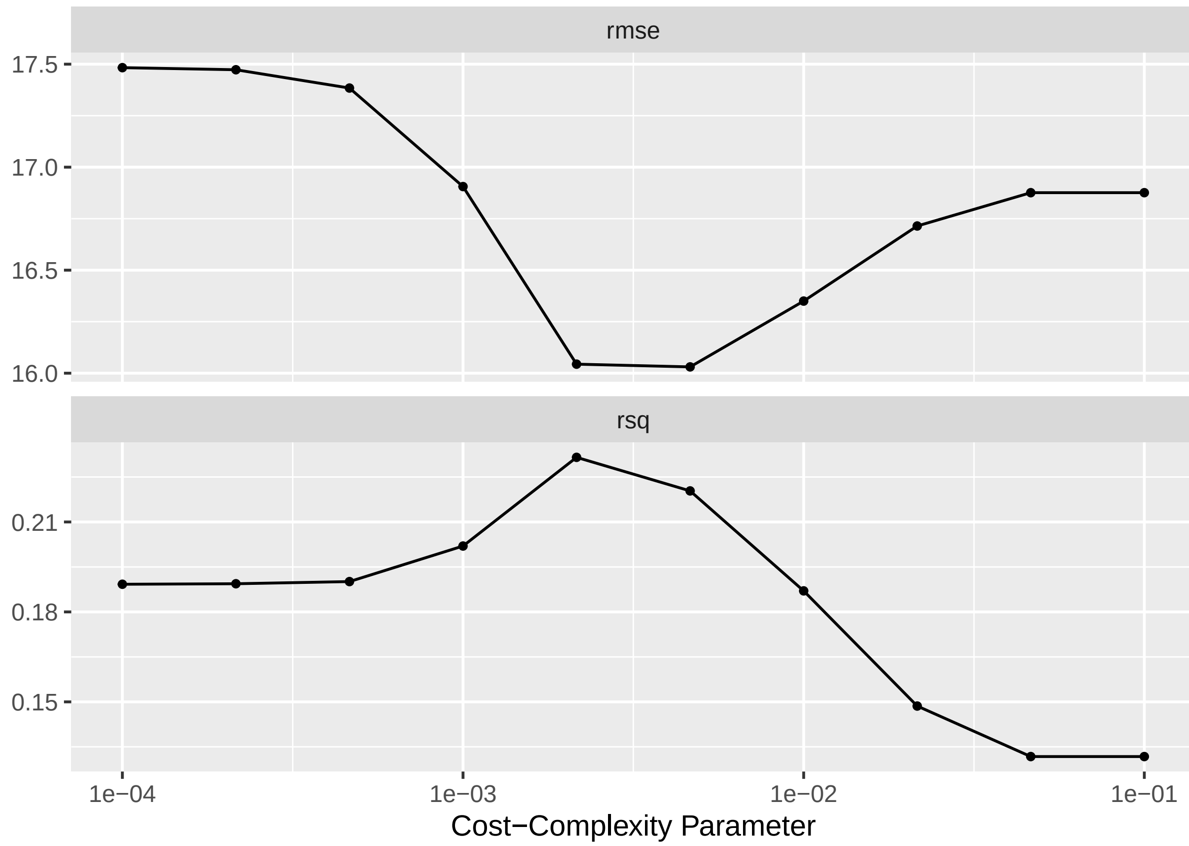
```
tree_spec <- decision_tree() %>%
  set_engine("rpart")
reg_tree_spec <- tree_spec %>%
  set_mode("regression")
reg_tree_fit <- fit(reg_tree_spec, popularity ~ ., spotify_train)
augment(reg_tree_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        16.2
```

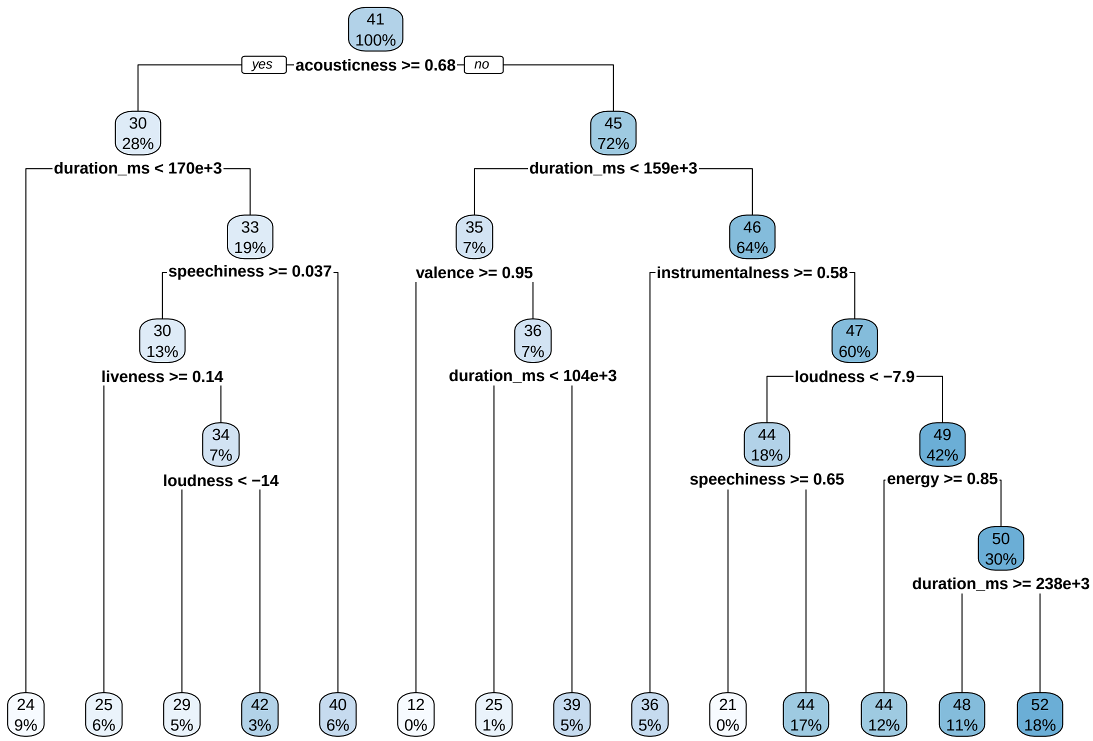
```
reg_tree_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```




```r
reg_tree_wf <- workflow() %>%
  add_model(reg_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_formula(popularity ~ .)

param_grid <- grid_regular(cost_complexity(range = c(-4, -1)), levels = 10)

tune_res <- tune_grid(
  reg_tree_wf,
  resamples = spotify_folds,
  grid = param_grid
)
autoplot(tune_res)
```




```
best_complexity <- select_best(tune_res, metric = "rmse")

reg_tree_final <- finalize_workflow(reg_tree_wf, best_complexity)

reg_tree_final_fit <- fit(reg_tree_final, data = spotify_train)
reg_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```




```
augment(reg_tree_final_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        15.8
```

Random Forest

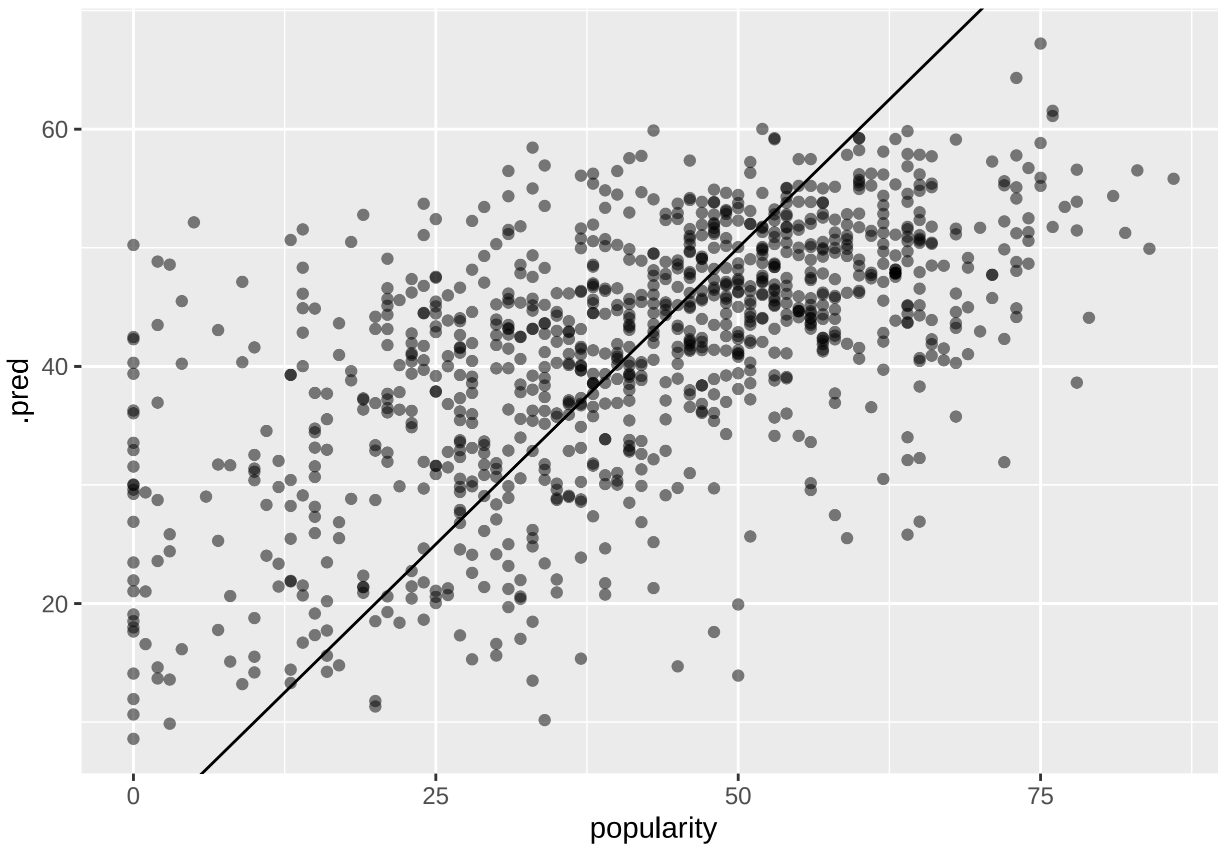
With boosted trees we can see that there are some predictors that weighted more than other predictors such as acousticness with high of importance of 15% incomparably with other predictors ranging 5-10%, which is inconsistent with what we have seen from the correlation matrix indicating that loudness is more correlated with popularity than other predictors. With the increase of tree depth doesnt help with the RMSE but it show the acousticness as taking more importnace in the prediction.

```
bagging_spec <- rand_forest(mtry = .cols()) %>%
  set_engine("randomForest", importance = TRUE) %>%
  set_mode("regression")
bagging_fit <- fit(bagging_spec, popularity ~ .,
                   data = spotify_train)
augment(bagging_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

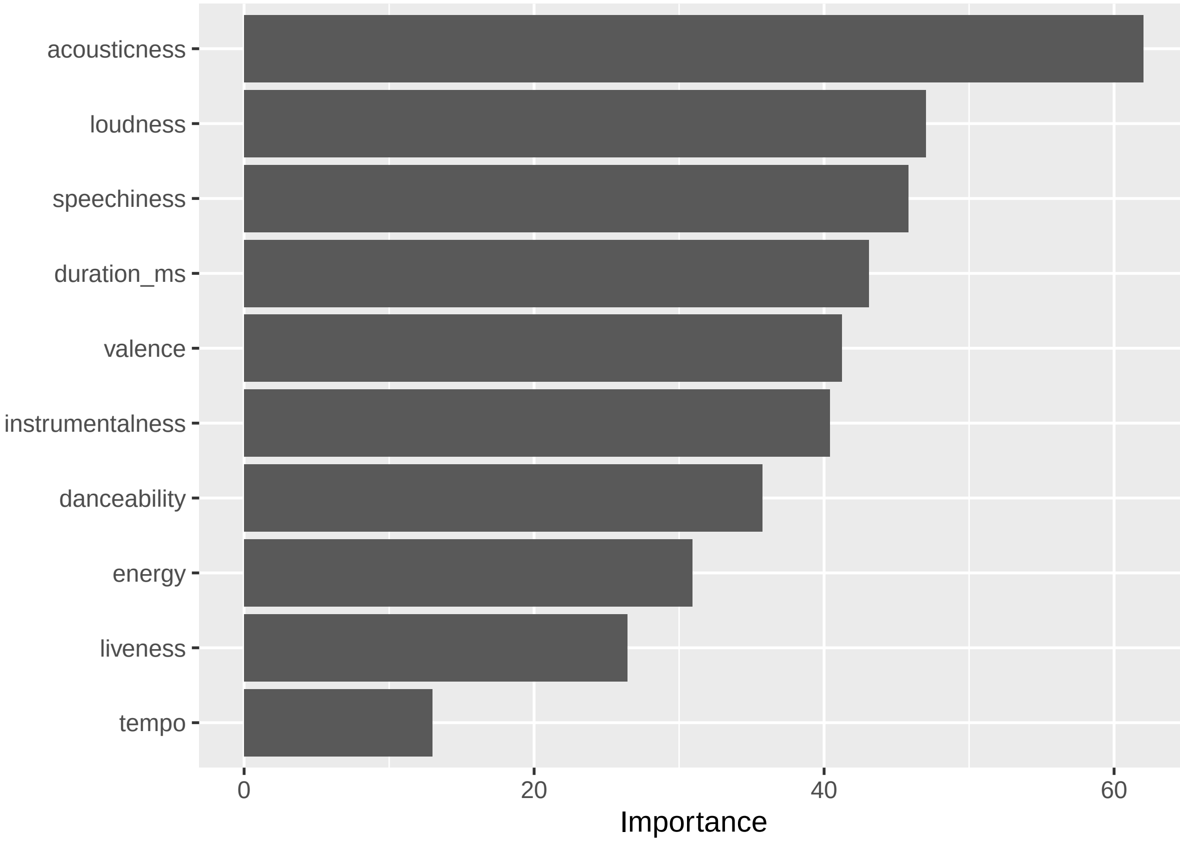
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        14.6
```

```
augment(bagging_fit, new_data = spotify_test) %>%
  ggplot(aes(popularity, .pred)) +
  geom_abline() +
```

```
  geom_point(alpha = 0.5)
```



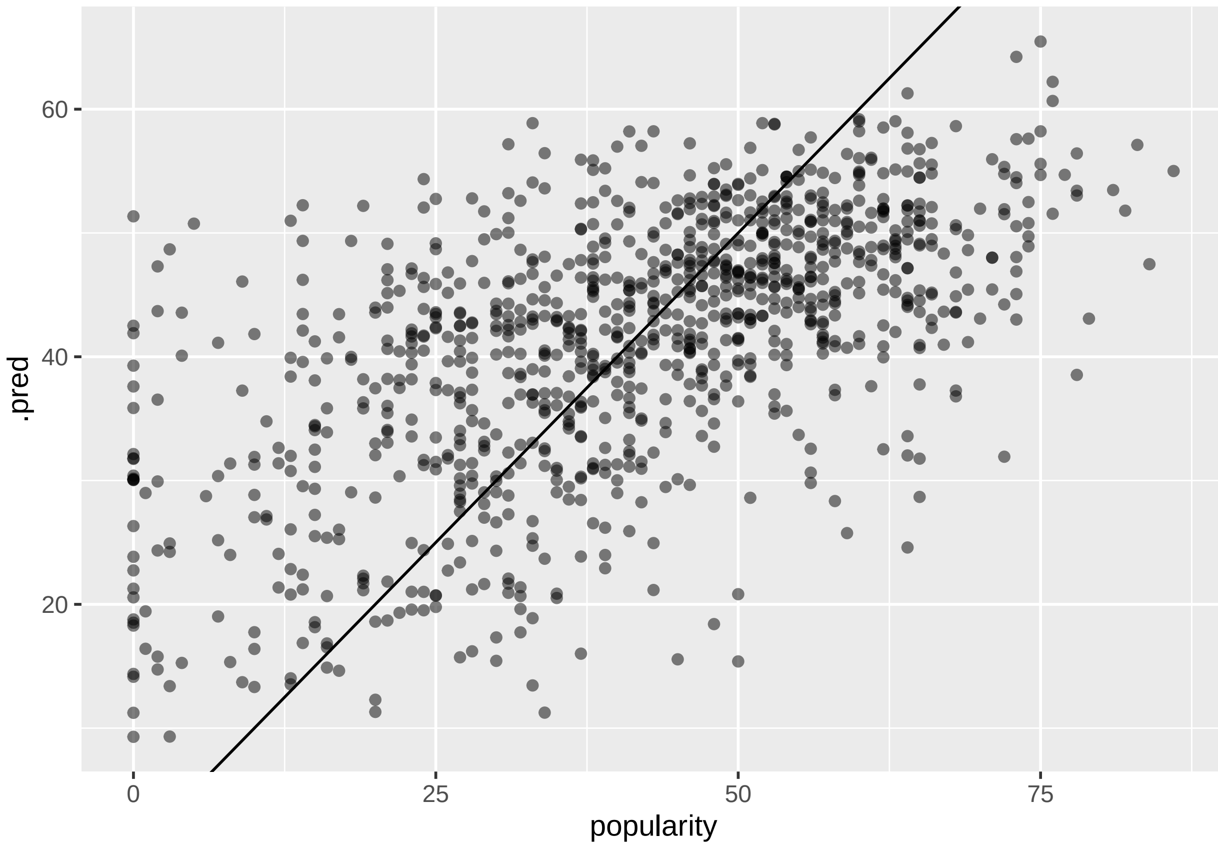

```
library(vip)
vip(bagging_fit)
```



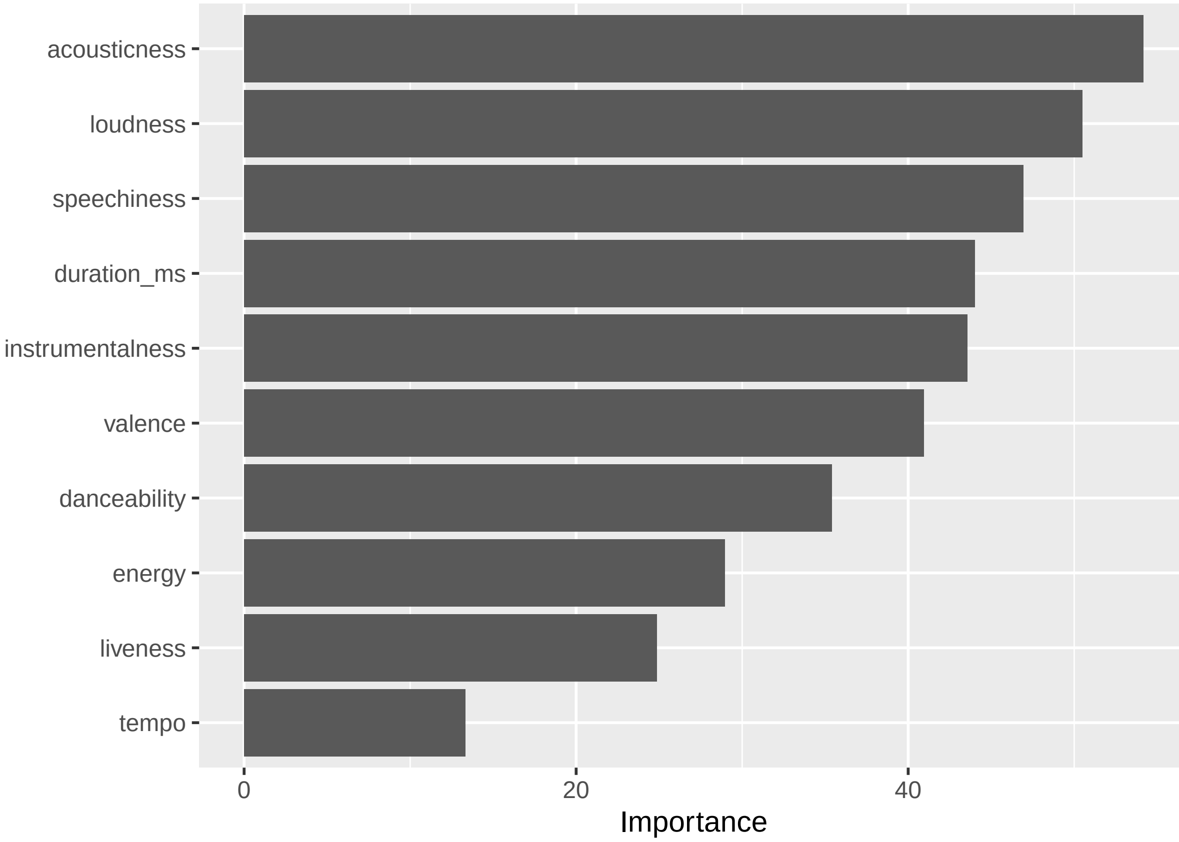

```r
rf_spec <- rand_forest(mtry = 6) %>%
  set_engine("randomForest", importance = TRUE) %>%
  set_mode("regression")
rf_fit <- fit(rf_spec, popularity ~ ., data = spotify_train)
augment(rf_fit, new_data = spotify_train) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        6.07
```

```r
augment(rf_fit, new_data = spotify_test) %>%
  ggplot(aes(popularity, .pred)) +
  geom_abline() +
  geom_point(alpha = 0.5)
```




```
vip(rf_fit)
```

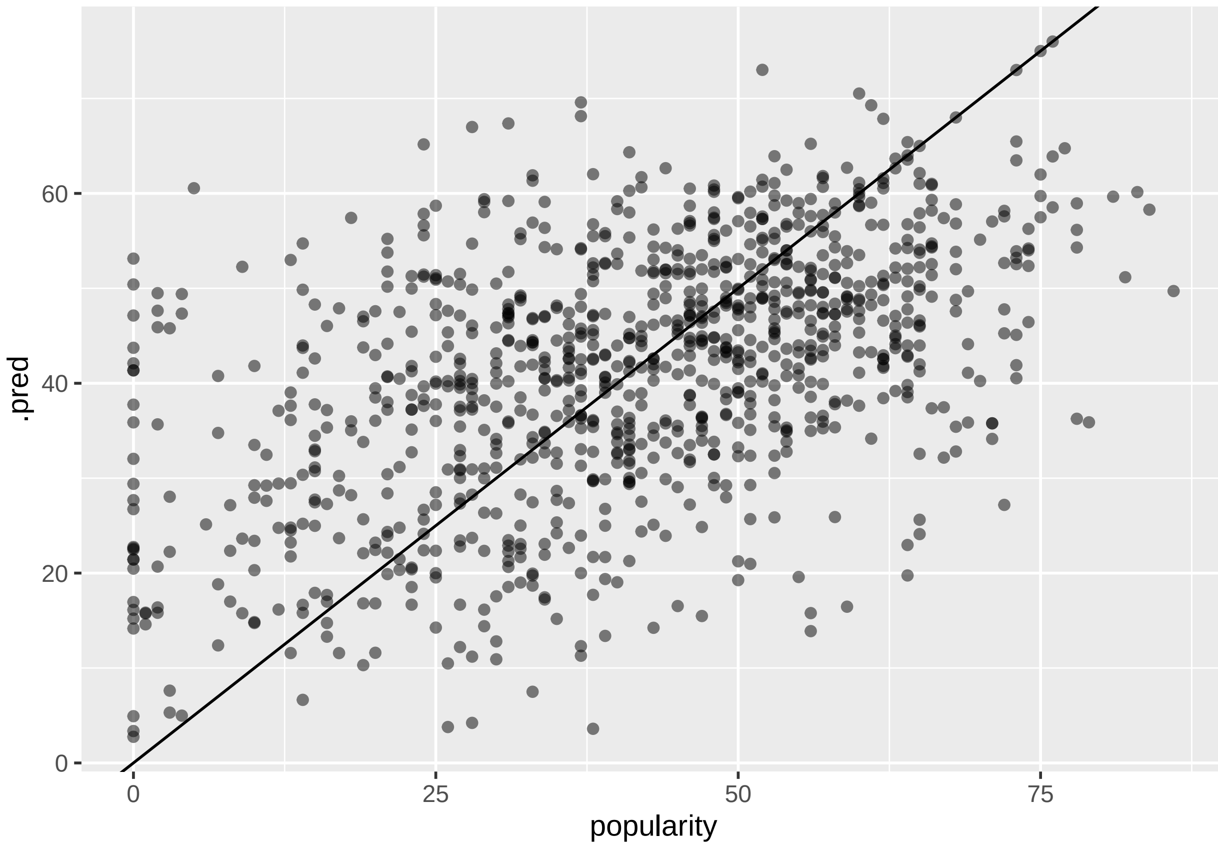


Boosted Trees With boosted trees we can see that there are some predictors that weighted more than other predictors such as acousticness with high of importance of 15% incomparably with other predictors ranging 5-10%, which is inconsistent with what we have seen from the correlation matrix indicating that loudness is more correlated with popularity than other predictors. With the increase of tree depth doesnt help with the RMSE but it show the acousticness as taking more importnace in the prediction.

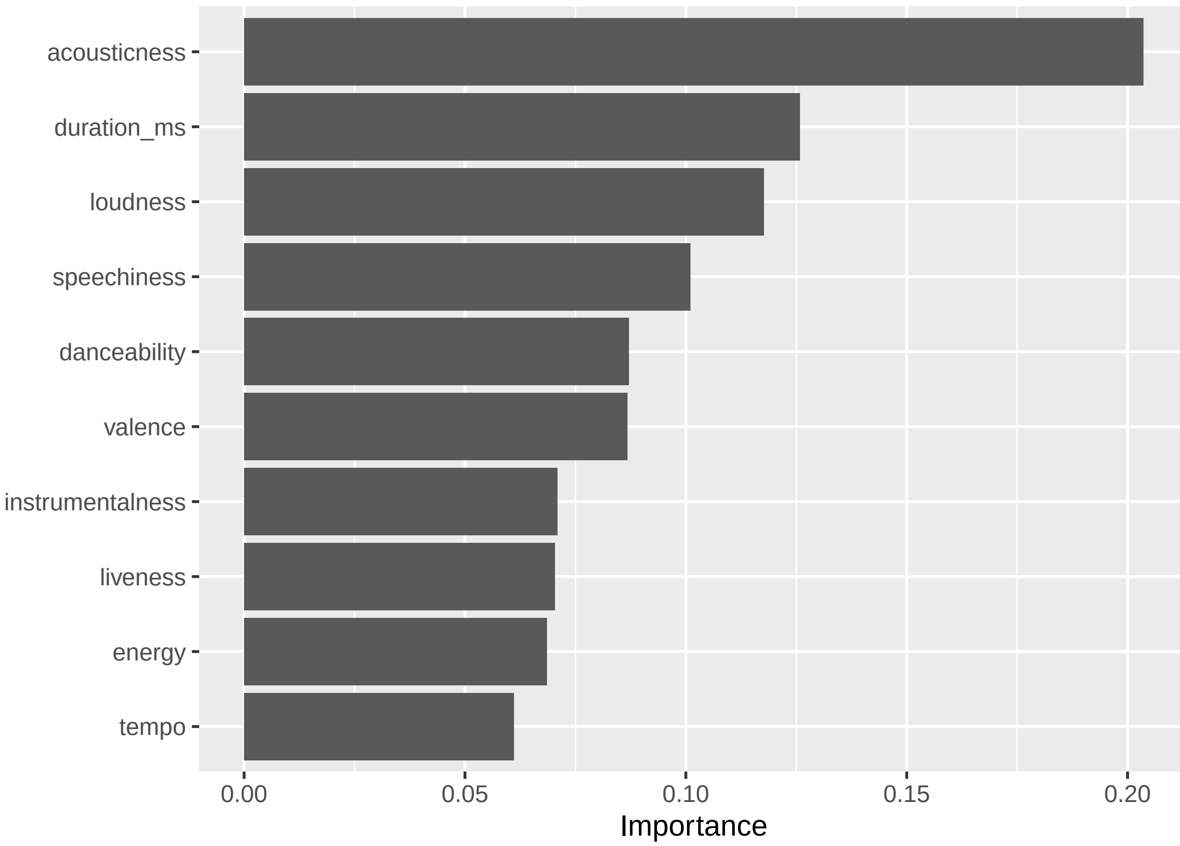
```
# Boosted Trees
boost_spec <- boost_tree(trees = 5000, tree_depth = 5) %>%
  set_engine("xgboost") %>%
  set_mode("regression")
boost_fit <- fit(boost_spec, popularity ~ ., data = spotify_train)
augment(boost_fit, new_data = spotify_test) %>%
  rmse(truth = popularity, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        16.2
```

```
augment(boost_fit, new_data = spotify_test) %>%
  ggplot(aes(popularity, .pred)) +
  geom_abline() +
  geom_point(alpha = 0.5)
```




```
vip(boost_fit)
```



#

Conclusion Overall we have done linear regression, Ridge regression, Lasso regression, Random forest, SVM, and boosted tree all of them have the similar RMSE of around 16 on the testing set prediction beside the random forest has comparably the low of 6. Overall I am surprised by the high RMSE of the models that produce usually it would be best to be around 0.2 to 0.5. From the regression models fitting, we already see that the correlation among the predictors and the popularity are not linear thus they would have a poor prediction. High RMSE could be indicating the overfitting that it trains well with training set however does it poor prediction with the testing set. Usually Boosted trees should be performing better than random forest, however, the reason that the random forest performs better is because the overfitting of the boosted trees.