Final Project: Popularity of Spotify Songs

Ashley Fidler and Courtney Shoen

Questions and Goals

In the following analysis we are going to attempt to determine what characteristics of a song most determine the songs overall popularity. The hope is that we will be able to accurately predict popularity, leading us to the most important elements of the songs.

Data Acquisition

We acquired our data from Leonardo Henrique on Kaggle. It appears as if Henrique got the data from Spotify and then put it into a workable dataset. The data consists of the top Spotify songs according to Billboard from 2010 to 2019 along with the various characteristics of each song.

```
library("tidyverse");theme_set(theme_bw())
library("tidymodels")
library("olsrr")
library("janitor")
library("dplyr")
library("glmnet")
library("gridExtra")
tidymodels_prefer()
library("parallel")
library("ranger")
library("baguette")
library("parsnip")

top10 <- read.csv("top10s.csv")
```

Data Preprocessing

Here we see a cleaning of the data by renaming the variables in a more workable and clearer manner. We also created several new variables that will be important in our later analysis. These include average_popularity which gives the average popularity rating of all songs across each year. There is also popularity_level which groups different levels of popularity into 4 categories: smash hit, very popular, popular, somewhat popular. The specifics for each variable can be found outlined in the table below.

```
top10 <- dplyr::rename(top10,
                 ID = X,
                 top_genre = top.genre,
                 tempo = bpm,
                 energy = nrgy,
                 danceability = dnce,
                 loudness = dB,
                 mood = val,
                 length = dur,
                 acoustic = acous,
```

```
                 spoken_word = spch,
                 popularity = pop)


#checking missing values
sum(is.na(top10))
```

```
## [1] 0
```

```
top10 <- top10 %>%
    group_by(year) %>%
    mutate(average_popularity = mean(popularity)) %>%
    ungroup()

top10 %>%
    summarise(mini = min(popularity), maxi = max(popularity), quant = quantile(popularity))
```

```
## # A tibble: 5 x 3
##    mini  maxi quant
##   <int> <int> <dbl>
## 1     0    99     0
## 2     0    99    60
## 3     0    99    69
## 4     0    99    76
## 5     0    99    99
```

```
top10 <- top10 %>%
mutate(popularity_level = case_when(
    popularity <= "20"  ~ "not popular",
    "20" < popularity & popularity <= "40" ~ "somewhat popular",
    "40" < popularity & popularity <= "60" ~ "popular",
    "60" < popularity & popularity <= "80" ~ "very popular",
    "80" < popularity ~ "smash hit"))
```

Variable	Description
ID	ID value in dataset
title	Song's title
artist	Song's artist
top_genre	the genre of the track
year	song's year in the Billboard
tempo	beats per minute
energy	the higher the value, the more energetic the song
danceability	the higher the value, the easier it is to dance to the song
loudness	the higher the value (dB) the louder the song
live	the higher the value, the more likely the song is a live recording
mood	the higher the value, the more positive the song's mood
length	the duration of the song
acoustic	the higher the value, the more acoustic the song
spoken_word	the higher the value, the more spoken word in the song
popularity	the higher the value, the more popular the song
average_popularity	average popularity per year
popularity_level	popularity sorted into 4 categories

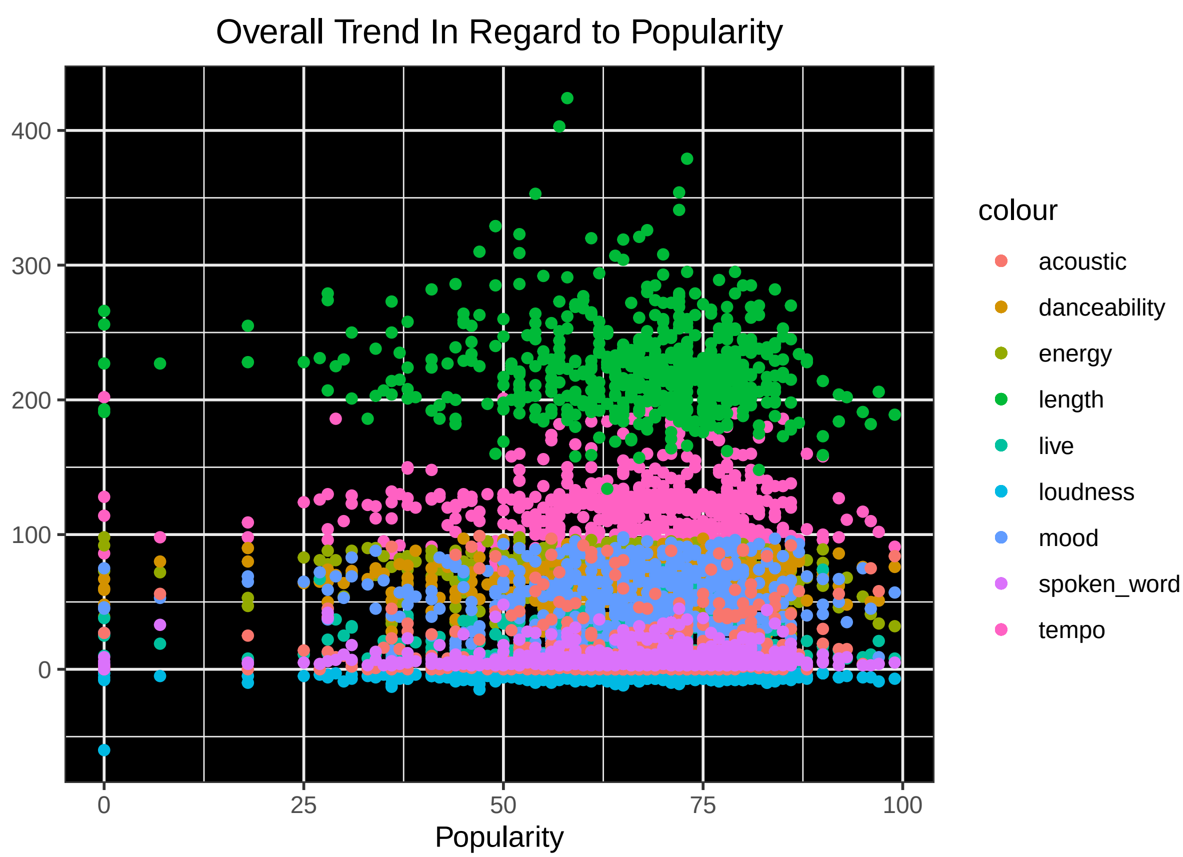
Exploratory Data

We then wanted to take a closer look at our data to determine the potential best model type along with possible predictors.

Overview of the Data

Here we see an overview of all of the predictors graphed against the response. We see several variables that appear to remain the same regardless of the level of popularity. These predictors include temp, spoken_word, and loudness. These variables may not have as much predictive power over popularity and will be important to watch for.

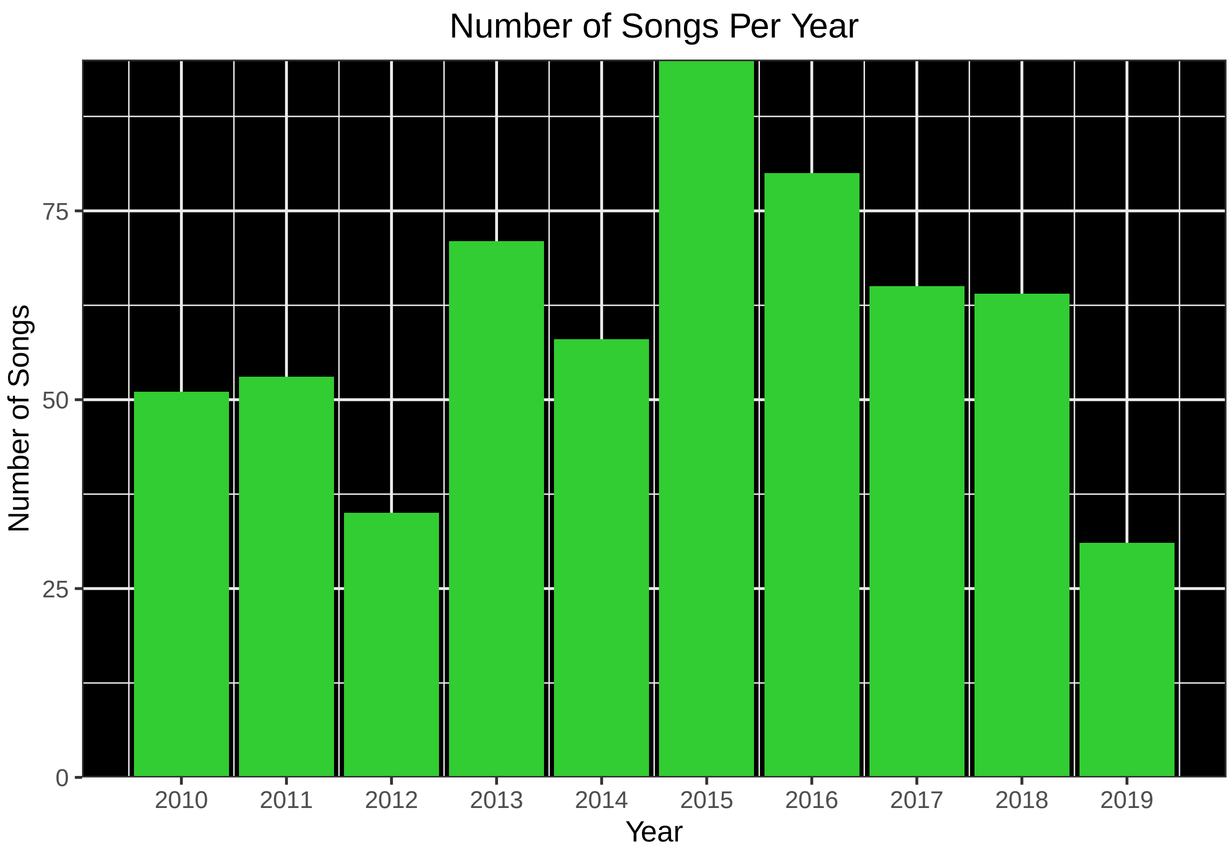
```
ggplot(top10, aes(x = popularity)) +
    geom_point(aes(y = tempo, colour = "tempo")) +
    geom_point(aes(y = energy, colour = "energy")) +
    geom_point(aes(y = danceability, colour = "danceability")) +
    geom_point(aes(y = loudness, colour = "loudness")) +
    geom_point(aes(y = live, colour = "live")) +
    geom_point(aes(y = mood, colour = "mood")) +
    geom_point(aes(y = length, colour = "length")) +
    geom_point(aes(y = acoustic, colour = "acoustic")) +
    geom_point(aes(y = spoken_word, colour = "spoken_word")) +
    ggtitle("Overall Trend In Regard to Popularity") +
    labs(x = "Popularity", y = "") +
    theme(panel.background = element_rect(fill = "black"), plot.title = element_text(hjust = 0.5))
```



Total Songs Per Year

Now we look at the number of songs on the list for each year. There are years with quite a few more songs than others. Regardless the range of total songs only ranges from about 35 to 100. Since this is not too large of a difference and the majority of the years have similar counts, this should not cause problems. However, we will want to watch out for year 2019 and 2015 as those are the two outliers.

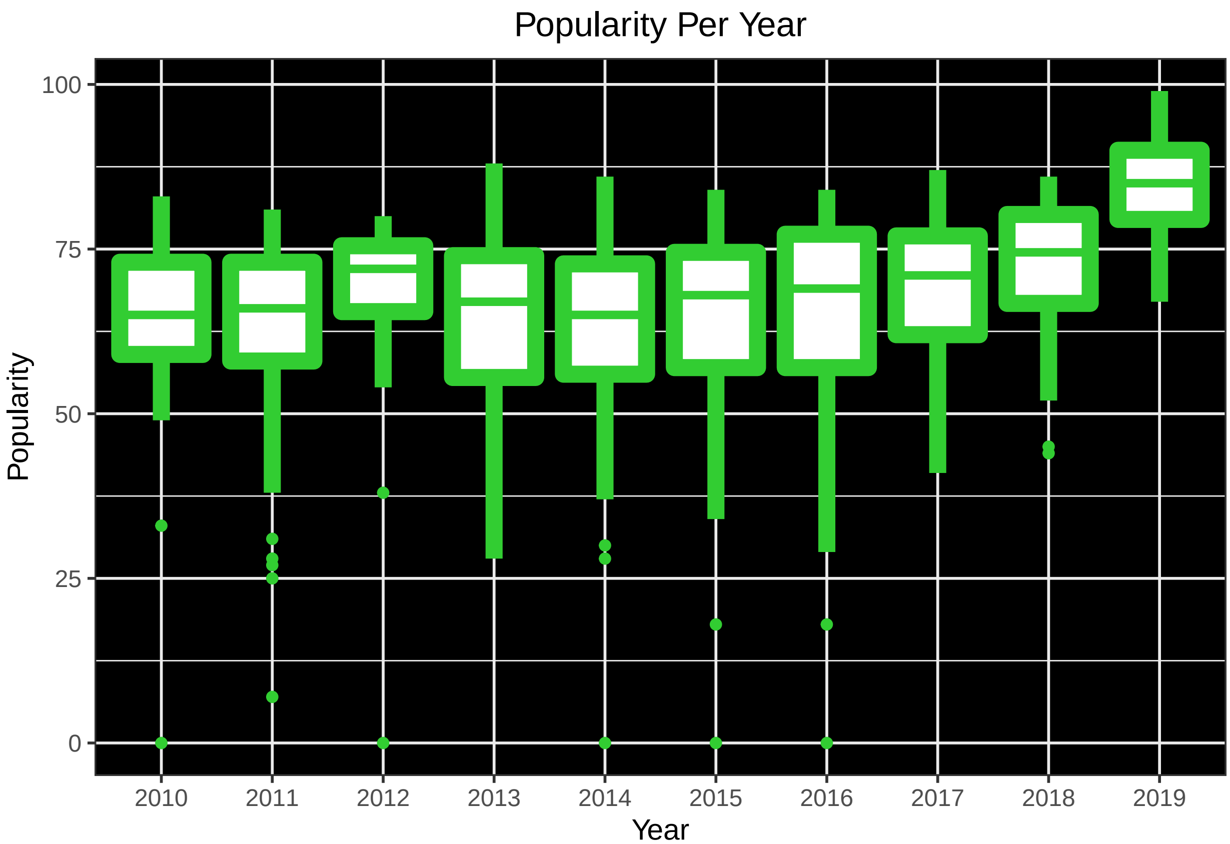
```
ggplot(top10) +
    geom_bar(aes(year), fill = "limegreen") +
    ggtitle("Number of Songs Per Year") +
    labs(x = "Year", y = "Number of Songs") +
    scale_x_continuous(breaks = c(2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020)) +
    scale_y_continuous(expand = c(0,0)) +
    theme(panel.background = element_rect(fill = "black"), plot.title = element_text(hjust = 0.5))
```



Popularity Distribution Each Year

Here we see the distributions of popularity for each year. The average popularity for each year is very similar and around 70. We do see a rise in popularity between 2017 and 2019. This leads us to believe that year may have some influence over popularity and could make an important predictor.

```
#Popularity Over the Years
ggplot(top10, aes(as.factor(year), popularity)) +
    geom_boxplot(colour = "limegreen", lwd = 3, fatten = .5) +
    ggtitle("Popularity Per Year") +
    labs(x = "Year", y = "Popularity") +
    theme(panel.background = element_rect(fill = "black"), plot.title = element_text(hjust = 0.5))
```



Correlations

With the data looking good and the variables appearing to be possible predictors of popularity, we then check for any correlation. We first chose variables that could logically have a correlation with one another. The only correlation that was even relatively high was between mood and energy. However, this correlation was only 0.4 and so it is not high enough to be of concern. Overall, there does not tend to be much correlation among predictors.

```
cor(top10$tempo, top10$energy)
```

```
## [1] 0.1261701
```

```
cor(top10$energy, top10$danceability)
```

```
## [1] 0.1672089
```

```
cor(top10$mood, top10$energy)
```

```
## [1] 0.4095773
```

```
cor(top10$live, top10$loudness)
```

```
## [1] 0.08193405
```

```
cor(top10$acoustic, top10$spoken_word)
```

```
## [1] 0.002762547
```

```
cor(top10$loudness, top10$danceability)
```

```
## [1] 0.2331701
```

```r
cor(top10$tempo, top10$danceability)
```

```
## [1] -0.1313007
```

Counts for Artist and Genre

Finally we look at how many times each artist and genre occurs across the data in order to check the categorical variables. We see that there do tend to be a large number of repeats for certain artists and genres. Thus, genre and artist may potentially be predictors of popularity

```r
top10 %>%
    count(artist) %>%
    arrange(desc(n))
```

```
## # A tibble: 184 x 2
##    artist               n
##    <chr>            <int>
##  1 Katy Perry          17
##  2 Justin Bieber       16
##  3 Maroon 5            15
##  4 Rihanna             15
##  5 Lady Gaga           14
##  6 Bruno Mars          13
##  7 Ed Sheeran          11
##  8 Pitbull             11
##  9 Shawn Mendes        11
## 10 The Chainsmokers    11
## # ... with 174 more rows
```

```r
top10%>%
    count(top_genre) %>%
    arrange(desc(n))
```

```
## # A tibble: 50 x 2
##    top_genre                    n
##    <chr>                    <int>
##  1 dance pop                  327
##  2 pop                         60
##  3 canadian pop                34
##  4 barbadian pop               15
##  5 boy band                    15
##  6 electropop                  13
##  7 british soul                11
##  8 big room                    10
##  9 canadian contemporary r&b    9
## 10 neo mellow                   9
## # ... with 40 more rows
```

Modeling and Analysis

```r
split <- initial_split(top10, prop = .9)

train <- training(split)
test <- testing(split)
```

```r
folds <- vfold_cv(train, v = 10)
```

Regression Models

Given the fact that not only my response, but most of our predictors were numeric, we decided to try regression models. We will see that neither regression model was doing a very good job of predicting popularity. It appeared as if the models were predicting very close to the average and failing to identify the full range of possible values.

Penalized Regression - LASSO Here we see year, energy, and loudness as the most prominent predictors of popularity when penalty is 0.095. The only variable that appears to have no predictive power is acoustic. We then see that the R-squared value is only 5%, meaning that the model is not predicting very much variability in popularity. Thus this model is not a good fit for the data. At a closer look we see that the residuals are very closely related to the popularity value. When popularity is high the residual is very high and when popularity is low the absolute value of the residual is very high. On the other hand when the popularity level lies closer to average the residual is low. This shows us that the model is sticking too close to average and not predicting the true values.

```r
recipe <- recipe(popularity ~ year + tempo + energy + danceability +
                  loudness + live + mood + length + acoustic + spoken_word,
                  data = test) %>%
    step_center(all_predictors()) %>%
    step_scale(all_predictors())


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

#putting the recipe (data) and model (method of fitting data) into the workflow
work <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(model)

grid <- grid_regular(penalty(), levels = 50)

#combining the model, recipe, possible penalties to put into the folds and test
tuned_grid <- tune_grid(work, resamples = folds, grid = grid)

#determining the best penalty for the model according to rmse
best_fit <- tuned_grid %>%
  select_best(metric = "rmse")

best_fit
```

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1   0.153 Preprocessor1_Model46
```

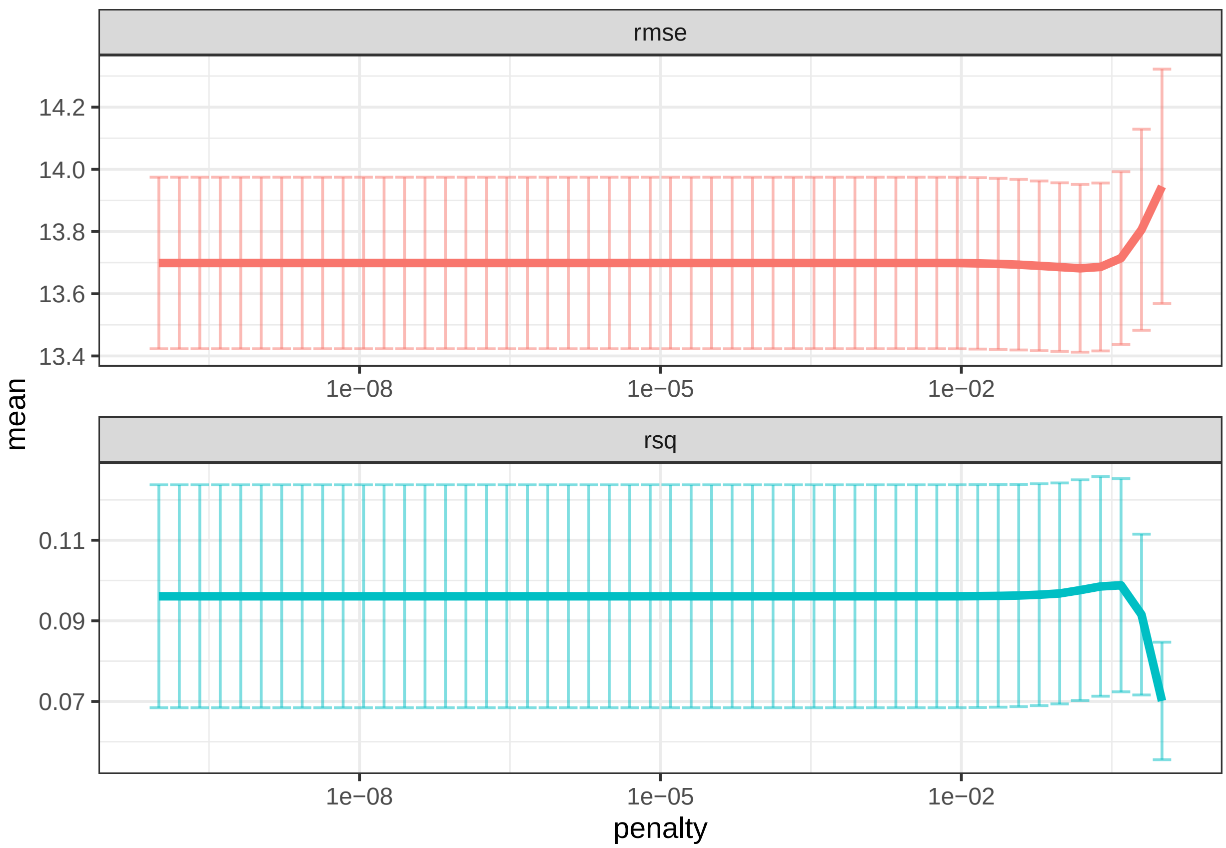
```r
#inserting the best penalty into the workflow (model and recipe)
work_final <- work %>%
  finalize_workflow(best_fit)

#training model based on optimal metrics (penalty)
```

```
final_fit <- work_final %>%
  last_fit(split = split) %>%
  collect_metrics()
final_fit
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard     15.3    Preprocessor1_Model1
## 2 rsq     standard      0.0357 Preprocessor1_Model1
```

```
#visualizing the plot
tuned_grid %>%
    collect_metrics() %>%
    ggplot(aes(penalty, mean, color = .metric)) +
    geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err), alpha = 0.5) +
    geom_line(size = 1.5) +
    facet_wrap(~.metric, scales = "free", nrow = 2) +
    scale_x_log10() +
    theme(legend.position = "none")
```




```
#gives predictions of final model with best penalty using test data
pred <- work_final %>%
    last_fit(split) %>%
    collect_predictions()

#provides the estimates of the predictors when the optimal penalty is applied
estimate_fit <- work_final %>%
```

```
  last_fit(split = split)
estimate_fit$.workflow[[1]] %>%
  extract_fit_parsnip() %>%
  tidy()
```

```
## # A tibble: 11 x 3
##    term         estimate penalty
##    <chr>           <dbl>   <dbl>
##  1 (Intercept)    66.6     0.153
##  2 year            3.02    0.153
##  3 tempo           0.105   0.153
##  4 energy         -2.00    0.153
##  5 danceability    0.945   0.153
##  6 loudness        3.41    0.153
##  7 live           -0.578   0.153
##  8 mood            0.240   0.153
##  9 length         -0.566   0.153
## 10 acoustic        0       0.153
## 11 spoken_word     0.145   0.153
```

```
#analyzing results
L_preds <- pred %>%
    mutate(residual = popularity - .pred) %>%
    select(popularity, .pred, residual) %>%
    arrange(desc(residual))
L_preds
```

```
## # A tibble: 61 x 3
##    popularity .pred residual
##         <int> <dbl>    <dbl>
##  1         85  62.4     22.6
##  2         90  71.0     19.0
##  3         86  67.3     18.7
##  4         79  61.2     17.8
##  5         79  62.0     17.0
##  6         76  61.8     14.2
##  7         81  66.8     14.2
##  8         81  67.7     13.3
##  9         78  64.8     13.2
## 10         76  63.6     12.4
## # ... with 51 more rows
```

Decision Tree In this case we see a very similar trend as that in LASSO. However, R-squared is not 20% which is a significant improvement from before, but still not in an acceptable region. We see here that the model is predicting values farther from average than before, but is still struggling to get the very high and low popularity values. As a result, we decided to try a different type of model.

```
# set model
model2 <- decision_tree(mode = "regression",
                       cost_complexity = tune(),   #creating a model where you tune two metrics
                        tree_depth = tune()) %>%   #in tree will either be regression or classification
          set_engine("rpart")

# data recipe
recipe2 <- recipe(popularity ~ artist + year + tempo + energy + danceability +
```

```
                  loudness + live + mood + length + acoustic + spoken_word,
                  data = test)

# workflow
work2 <- workflow() %>%
    add_recipe(recipe2) %>%
    add_model(model2)

# create grid
tree_grid <- grid_regular(cost_complexity(),   #choosing tree_depth and cost_complexity values to test
                          tree_depth(),
                          levels = 5)

# does the tuning
tree_tuned <- work2 %>%
    tune_grid(
        resamples = folds,
        grid = tree_grid)
tree_tuned
```

```
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [487/55]> Fold01 <tibble [50 x 6]> <tibble [0 x 1]>
##  2 <split [487/55]> Fold02 <tibble [50 x 6]> <tibble [0 x 1]>
##  3 <split [488/54]> Fold03 <tibble [50 x 6]> <tibble [0 x 1]>
##  4 <split [488/54]> Fold04 <tibble [50 x 6]> <tibble [0 x 1]>
##  5 <split [488/54]> Fold05 <tibble [50 x 6]> <tibble [0 x 1]>
##  6 <split [488/54]> Fold06 <tibble [50 x 6]> <tibble [0 x 1]>
##  7 <split [488/54]> Fold07 <tibble [50 x 6]> <tibble [0 x 1]>
##  8 <split [488/54]> Fold08 <tibble [50 x 6]> <tibble [0 x 1]>
##  9 <split [488/54]> Fold09 <tibble [50 x 6]> <tibble [0 x 1]>
## 10 <split [488/54]> Fold10 <tibble [50 x 6]> <tibble [0 x 1]>
```

```
# find the best collection of cost_complexity and tree_depth
best_tree <- tree_tuned %>%
    select_best("rmse")

# finalize the workflow
final_wf <- work2 %>%
    finalize_workflow(best_tree)    #puts best metrics back into workflow

# metrics of the tuned model
final_wf %>%
    last_fit(split) %>%
    collect_metrics()         #want to train model based on optimal values
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard     16.4    Preprocessor1_Model1
## 2 rsq     standard      0.0720 Preprocessor1_Model1
```

```r
# predictions of tuned model
preds <- final_wf %>%
    last_fit(split) %>%
    collect_predictions() #gives predictions of final model with best  metrics using test data

tree_preds <- preds %>%
  mutate(residual = popularity - .pred) %>%
  select(.pred, popularity, residual) %>%
  arrange(desc(residual))
tree_preds
```

```
## # A tibble: 61 x 3
##    .pred popularity residual
##    <dbl>      <int>    <dbl>
##  1  43           86     43
##  2  43           76     33
##  3  49.3         79     29.7
##  4  58.0         81     23.0
##  5  49.3         72     22.7
##  6  58.0         79     21.0
##  7  62.9         80     17.1
##  8  68.1         85     16.9
##  9  68.1         85     16.9
## 10  58.0         74     16.0
## # ... with 51 more rows
```

Classification Models

The regression models we tried were struggling to predict the full realm of popularity. So we decided to make popularity categorical by grouping values into categories. This can be seen in the Data Preprocessing section.

Bagging Here we attempt to predict popularity_level with a classification bagging model using rpart. Here we work to tune cost_complexity, tree_depth, and min_n according to roc_auc and get 0, 8, and 2 respectively. In this case we get an accuracy level of 68%. So, our model is accurately predicting the correct popularity_level 68% of the time. This is a big improvement from the regression models and so we look further.

```r
bag_recipe <- recipe(popularity_level ~ year + tempo + energy +
                     danceability + loudness + live + mood + length + acoustic
                     + spoken_word, train)

bag_model <- bag_tree(cost_complexity = tune(),
                      tree_depth = tune(),
                      min_n = tune()) %>%
             set_mode("classification") %>%
             set_engine("rpart", times = 25)

bag_workflow <- workflow() %>%
                add_recipe(bag_recipe) %>%
                add_model(bag_model)

bag_grid <- grid_regular(cost_complexity(),tree_depth(), min_n(), levels = 5)

#bag_tuned <- bag_workflow %>%
```

```r
              #tune_grid(resamples = folds,
                         #grid = bag_grid)
load("/cloud/project/bag_data.RData")

bag_tuned
```

```
## Warning: This tuning result has notes. Example notes on model fitting include:
## internal: No observations were detected in `truth` for level(s): 'not popular'
## Computation will proceed by ignoring those levels.
## internal: No observations were detected in `truth` for level(s): 'not popular', 'somewhat popular'
## Computation will proceed by ignoring those levels.
## internal: No observations were detected in `truth` for level(s): 'not popular'
## Computation will proceed by ignoring those levels.
```

```
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits           id     .metrics           .notes
##    <list>           <chr>  <list>             <list>
##  1 <split [405/46]> Fold01 <tibble [250 x 7]> <tibble [1 x 1]>
##  2 <split [406/45]> Fold02 <tibble [250 x 7]> <tibble [0 x 1]>
##  3 <split [406/45]> Fold03 <tibble [250 x 7]> <tibble [0 x 1]>
##  4 <split [406/45]> Fold04 <tibble [250 x 7]> <tibble [1 x 1]>
##  5 <split [406/45]> Fold05 <tibble [250 x 7]> <tibble [0 x 1]>
##  6 <split [406/45]> Fold06 <tibble [250 x 7]> <tibble [0 x 1]>
##  7 <split [406/45]> Fold07 <tibble [250 x 7]> <tibble [1 x 1]>
##  8 <split [406/45]> Fold08 <tibble [250 x 7]> <tibble [0 x 1]>
##  9 <split [406/45]> Fold09 <tibble [250 x 7]> <tibble [1 x 1]>
## 10 <split [406/45]> Fold10 <tibble [250 x 7]> <tibble [1 x 1]>
```

```r
best_bag <- bag_tuned %>%
            select_best("roc_auc")

final_bag <- bag_workflow %>%
            finalize_workflow(best_bag)

final_bag %>%
    last_fit(split) %>%
    collect_metrics()
```

```
## ! train/test split: internal: No observations were detected in `truth` for level(s): 'not po...
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.607 Preprocessor1_Model1
## 2 roc_auc  hand_till      0.571 Preprocessor1_Model1
```

```r
pred_bag <- final_bag %>%
    last_fit(split) %>%
    collect_predictions() %>%
    select(.row, .pred_class, popularity_level)
```

```
## ! train/test split: internal: No observations were detected in `truth` for level(s): 'not po...
```

Boosting With the success of classification we moved onto a Boosting model using XGboost. Here we tune trees, min_n, and tree_depth and get 1, 21, and 4 respectively. In this case we get an accuracy of 57% meaning that popularity_level is correctly predicted 57% of the time. This is not quite as good at Bagging but still a decent level of prediction.

```
#boosting

#boost_split <- initial_split(top10, prop = .9)
#
#boost_train <- training(boost_split)
#boost_test <- testing(boost_split)
#
#boost_folds <- vfold_cv(boost_train, v = 10)
#
# boost_recipe <- recipe(popularity_level ~ year + tempo + energy +
#                         danceability + loudness + live + mood + length + acoustic
#                         + spoken_word, boost_train)
#
# boost_model <- boost_tree(trees = tune(),
#                            min_n = tune(),
#                            tree_depth = tune()) %>%
#                set_mode("classification") %>%
#                set_engine("xgboost")
#
# boost_workflow <- workflow() %>%
#                  add_recipe(boost_recipe) %>%
#                  add_model(boost_model)
#
# boost_grid <- grid_regular(trees(), min_n(), tree_depth(), levels = 5)

 # boost_tuned <- boost_workflow %>%
 #                tune_grid(resamples = boost_folds,
 #                          grid = boost_grid)

 load("/cloud/project/boost_results.RData")

 best_boost <-boost_tuned %>%
            select_best("roc_auc")


 final_boost <- boost_workflow %>%
             finalize_workflow(best_boost)

 final_boost %>%
     last_fit(boost_split) %>%
     collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.639 Preprocessor1_Model1
## 2 roc_auc  hand_till      0.625 Preprocessor1_Model1
```

```
 pred_boost <- final_boost %>%
     last_fit(boost_split) %>%
```

```
    collect_predictions() %>%
    select(.row, .pred_class, popularity_level)
```

Random Forest Finally we tried the random forest model using ranger. We tuned trees and min_n and got 2000 and 30 respectively. In this case we get an accuracy of 72% meaning that the popularity_level is being predicted correctly 75% of the time.

```
rf_model <- rand_forest(trees = tune(), min_n = tune()) %>%
            set_engine("ranger") %>%
            set_mode("classification")

rf_data <- recipe(popularity_level ~ year + tempo + energy +
                  danceability + loudness + live + mood + length + acoustic +
                  spoken_word, train)

# workflow
rf_wf <- workflow() %>%
        add_model(rf_model) %>%
        add_recipe(rf_data)

rf_grid <- grid_regular(trees(), min_n(), levels = 5)

# does the tuning
rf_tuned <- rf_wf %>%
            tune_grid(
                resamples = folds,
                grid = rf_grid)
```

```
## ! Fold01: internal: No observations were detected in `truth` for level(s): 'not po...

## ! Fold02: internal: No observations were detected in `truth` for level(s): 'not po...

## ! Fold07: internal: No observations were detected in `truth` for level(s): 'not po...

## ! Fold09: internal: No observations were detected in `truth` for level(s): 'not po...
```

```
rf_tuned
```

```
## Warning: This tuning result has notes. Example notes on model fitting include:
## internal: No observations were detected in `truth` for level(s): 'not popular'
## Computation will proceed by ignoring those levels.
## internal: No observations were detected in `truth` for level(s): 'not popular'
## Computation will proceed by ignoring those levels.
## internal: No observations were detected in `truth` for level(s): 'not popular'
## Computation will proceed by ignoring those levels.

## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits            id     .metrics           .notes
##    <list>            <chr>  <list>             <list>
##  1 <split [487/55]> Fold01 <tibble [50 x 6]> <tibble [1 x 1]>
##  2 <split [487/55]> Fold02 <tibble [50 x 6]> <tibble [1 x 1]>
##  3 <split [488/54]> Fold03 <tibble [50 x 6]> <tibble [0 x 1]>
##  4 <split [488/54]> Fold04 <tibble [50 x 6]> <tibble [0 x 1]>
##  5 <split [488/54]> Fold05 <tibble [50 x 6]> <tibble [0 x 1]>
##  6 <split [488/54]> Fold06 <tibble [50 x 6]> <tibble [0 x 1]>
```

```
##  7 <split [488/54]> Fold07 <tibble [50 x 6]> <tibble [1 x 1]>
##  8 <split [488/54]> Fold08 <tibble [50 x 6]> <tibble [0 x 1]>
##  9 <split [488/54]> Fold09 <tibble [50 x 6]> <tibble [1 x 1]>
## 10 <split [488/54]> Fold10 <tibble [50 x 6]> <tibble [0 x 1]>
```

```
# find the best collection of cost_complexity and tree_depth

best_rf <- rf_tuned %>%
           select_best("roc_auc")

# finalize the workflow
final_rf <- rf_wf %>%
            finalize_workflow(best_rf)

# metrics of the tuned model
final_rf %>%
    last_fit(split) %>%
    collect_metrics()
```

```
## ! train/test split: internal: No observations were detected in `truth` for level(s): 'not po...
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.623 Preprocessor1_Model1
## 2 roc_auc  hand_till      0.603 Preprocessor1_Model1
```

```
# predictions of tuned model
preds <- final_rf %>%
    last_fit(split) %>%
    collect_predictions() %>%
    select(.row, .pred_class, popularity_level)
```

```
## ! train/test split: internal: No observations were detected in `truth` for level(s): 'not po...
```

Conclusion

Overall, the classification models ended up doing a good job of predicting popularity_level, telling us that the characteristics of a song do have some ability to predict how popular it will be. The only concern here is that the classification models are predicting popularity categories that are a stretch of 20 popularity values. So the large residuals we saw in the regression models are in a way being eliminated by the categories we created. This could potentially be misleading and so it is important to note that in this case we are predicting the general popularity level and not the actual value. In conclusion if given the characteristics of a song we would be able to predict the general level of popularity that song will reach but not the exact popularity rating.