Park-Lab5

Patty Park

2023-02-07

```
{{r setup, echo = FALSE, message = FALSE, warning = FALSE}} knitr::opts_chunk$set(fig.width
= 4, fig.height = 3,                          echo = TRUE, message = FALSE, warning =
FALSE)
```

This week's lab is a musical lab. You'll be requesting data from the Spotify API and using it to build k-nearest neighbor and decision tree models.

In order to use the Spotify API you must have a Spotify account. If you don't have one, sign up for a free one here: https://www.spotify.com/us/signup

Once you have an account, go to Spotify for developers (https://developer.spotify.com/) and log in. Click the green "Create a Client ID" button to fill out the form to create an app create an app so you can access the API.

On your developer dashboard page, click on the new app you just created. Go to Settings -> Basic Information and you will find your Client ID . Click "View client secret" to access your secondary Client ID. Scroll down to Redirect URIs and enter: http://localhost:1410/

You have two options for completing this lab.

Option 1: **Classify by users**. Build models that predict whether a given song will be in your collection vs. a partner in class. This requires that you were already a Spotify user so you have enough data to work with. You will download your data from the Spotify API and then exchange with another member of class.

Option 2: **Classify by genres**. Build models that predict which genre a song belongs to. This will use a pre-existing Spotify dataset available from Kaggle.com (https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify)

```
library(spotifyr) #API interaction
library(tidyverse)
library(tidymodels)
library(janitor)
library(patchwork)
library(kableExtra)
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'
```

```
library(baguette)
```

Client ID and Client Secret are required to create and access token that is required to interact with the API. You can set them as system values so we don't have to do provide them each time.

```
Sys.setenv(SPOTIFY_CLIENT_ID = 'b61d658fd88f4a138cc245a7943dbeeb')
Sys.setenv(SPOTIFY_CLIENT_SECRET = '1eb796bb985d4820be69fb6b45d6eb27')

#authorization_code <- get_spotify_authorization_code(scope = scopes()[c(1:19)]) #sets an authorization

#access_token <- get_spotify_access_token() #takes ID and SECRET, sends to Spotify and receives an acce
```

Option 1: Data Preparation

You can use get_my_saved_tracks() to request all your liked tracks. It would be good if you had at least 150-200 liked tracks so the model has enough data to work with. If you don't have enough liked tracks, you can instead use get_my_recently_played(), and in that case grab at least 500 recently played tracks if you can.

The Spotify API returns a dataframe of tracks and associated attributes. However, it will only return up to 50 (or 20) tracks at a time, so you will have to make multiple requests. Use a function to combine all your requests in one call.

Once you have your tracks, familiarize yourself with this initial dataframe. You'll need to request some additional information for the analysis. If you give the API a list of track IDs using get_track_audio_features(), it will return an audio features dataframe of all the tracks and some attributes of them.

These track audio features are the predictors we are interested in, but this dataframe doesn't have the actual names of the tracks. Append the 'track.name' column from your favorite tracks database.

Find a class mate whose data you would like to use. Add your partner's data to your dataset. Create a new column that will contain the outcome variable that you will try to predict. This variable should contain two values that represent if the track came from your data set or your partner's.

Option 2: Data preparation

Download the Spotify dataset from https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify

Inspect the data. Choose two genres you'd like to use for the classification task. Filter down the data to include only the tracks of that genre.

```
#read in data
set.seed(50)
kaggle_genre <- read_csv(here::here("week_5", "data", "genres_v2.csv"))
kaggle_playlist <- read_csv(here::here("week_5", "data", "playlists.csv"))
```

```
#data filtering
unique(kaggle_genre$genre)
```

```
##  [1] "Dark Trap"       "Underground Rap" "Trap Metal"      "Emo"
##  [5] "Rap"             "RnB"             "Pop"             "Hiphop"
##  [9] "techhouse"       "techno"          "trance"          "psytrance"
## [13] "trap"            "dnb"             "hardstyle"
```

```
table(kaggle_genre$genre)
```

```
##
##       Dark Trap             dnb             Emo       hardstyle          Hiphop
##            4578            2966            1680            2936            3028
##             Pop       psytrance             Rap             RnB       techhouse
##             461            2961            1848            2099            2975
##          techno          trance            trap      Trap Metal Underground Rap
##            2956            2999            2987            1956            5875
```

```
genre_dat <- kaggle_genre %>%
  select(-c(type, uri, track_href, analysis_url, `Unnamed: 0`, title, tempo, id, song_name)) %>% #remov
  filter(genre == "Pop"|genre == "RnB") %>%
  mutate(genre = str_replace(genre, "Pop", "pop")) %>%
  mutate(genre = str_replace(genre, "RnB", "rnb")) %>%
  mutate(genre = as.factor(genre))
```

###Data Exploration (both options)

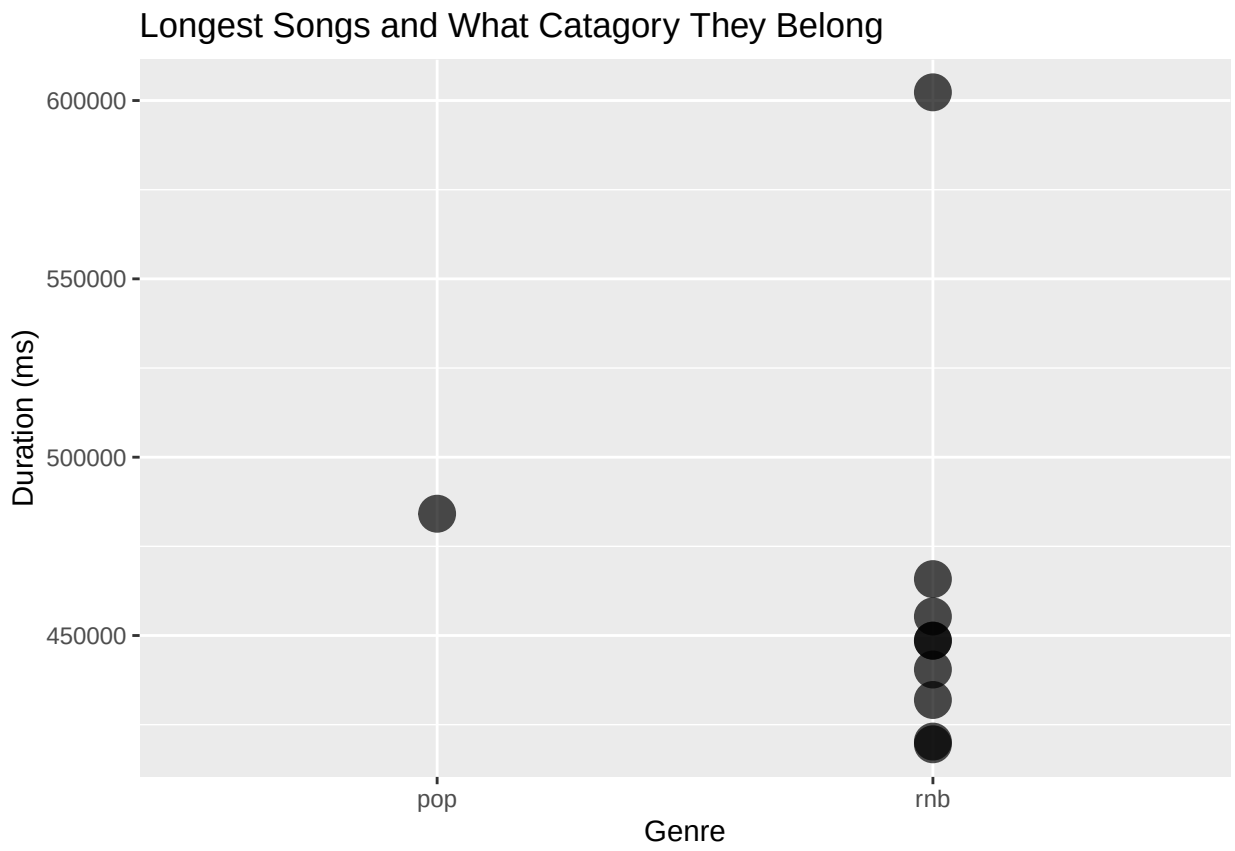
Let's take a look at your data. Do some exploratory summary stats and visualization.

For example: What are the most danceable tracks in your dataset? What are some differences in the data between genres (Option 2)?

```
#look at data
#longest songs
longest_songs <- genre_dat %>%
  slice_max(duration_ms, n = 10)
head(longest_songs)
```

```
## # A tibble: 6 x 13
##   danceability energy   key loudness  mode speechiness acousticness
##          <dbl>  <dbl> <dbl>    <dbl> <dbl>       <dbl>        <dbl>
## 1        0.355  0.298     8   -12.3      1      0.0581       0.31
## 2        0.574  0.512     5    -6.66     0      0.0503       0.234
## 3        0.392  0.473     8   -11.4      0      0.118        0.824
## 4        0.324  0.852    11    -6.68     0      0.109        0.0415
## 5        0.687  0.723     7    -4.75     1      0.0709       0.122
## 6        0.685  0.725     7    -4.79     1      0.177        0.13
## # i 6 more variables: instrumentalness <dbl>, liveness <dbl>, valence <dbl>,
## #   duration_ms <dbl>, time_signature <dbl>, genre <fct>
```

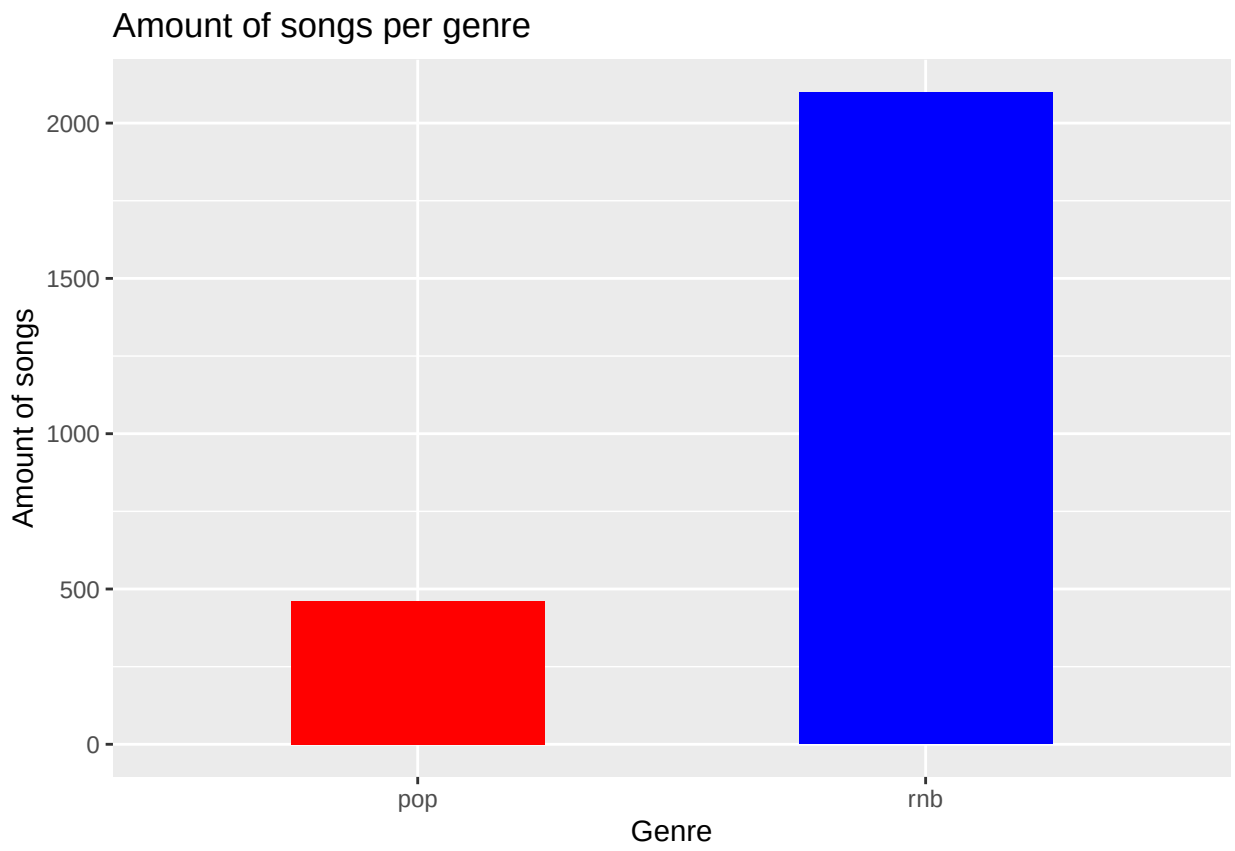
```
#visualization
ggplot(data = longest_songs, aes(x = genre, y = duration_ms)) +
  geom_point(size = 6, alpha = 0.7) +
  labs(title = "Longest Songs and What Catagory They Belong",
       x = "Genre",
       y = "Duration (ms)")
```




```
#how many songs per genre
amount_pop <- genre_dat %>%
  count(genre = genre)
amount_pop
```

```
## # A tibble: 2 x 2
##   genre     n
##   <fct> <int>
## 1 pop     461
## 2 rnb    2099
```

```
#visualization
ggplot(data = amount_pop, aes(x = genre, y = n)) +
  geom_col(width = 0.5, fill = c("red", "blue")) +
  labs(title = "Amount of songs per genre",
       x = "Genre",
       y = "Amount of songs")
```



Modeling

Create competing models that predict whether a track belongs to:

Option 1. you or your partner's collection

Option 2. genre 1 or genre 2

You will eventually create four final candidate models:

1. k-nearest neighbor (Week 5)
2. decision tree (Week 5)
3. bagged tree (Week 6)
    - bag_tree()
    - Use the "times =" argument when setting the engine during model specification to specify the number of trees. The rule of thumb is that 50-500 trees is usually sufficient. The bottom of that range should be sufficient here.
4. random forest (Week 6)
    - rand_forest()
    - m_try() is the new hyperparameter of interest for this type of model. Make sure to include it in your tuning process

Go through the modeling process for each model:

Preprocessing. You can use the same recipe for all the models you create.

Resampling. Make sure to use appropriate resampling to select the best version created by each algorithm.

Tuning. Find the best values for each hyperparameter (within a reasonable range).

Compare the performance of the four final models you have created.

Use appropriate performance evaluation metric(s) for this classification task. A table would be a good way to display your comparison. Use at least one visualization illustrating your model results.

Model 1: k-nearest neighbor

```
#setting seed
set.seed(50)
#split data and create training and testing data
genre_split <- initial_split(genre_dat, prop = 0.8)
genre_train <- training(genre_split)
genre_test<- testing(genre_split)
```

```
#preprocessing
rec_genre <- recipe(genre ~., data = genre_train) %>% #analysing genre on all other predictors
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  prep()

#bake the recipe to view the recipe
#baked_train_genre <- bake(knn_rec_genre, genre_train)
```

```
#specify k-nearest neighbor model
#tell it how many neighbors to look at to make a decision
knn_model_genre <- nearest_neighbor(neighbors = tune()) %>% #set neighbors to tune parameter
  set_mode("classification") %>%
  set_engine("kknn") #method of estimation (a distinction)
```

```
#set seed again
set.seed(50)

#set folds to be 5
cv_folds_knn <- genre_train %>% vfold_cv(v = 5)
```

```
#put in workflow with tune parameter
knn_workflow_genre <- workflow() %>%
  add_model(knn_model_genre) %>%
  add_recipe(rec_genre)
```

```
# Fit the workflow on our predefined folds and a grid of hyperparameters
# fit_knn_cv <- knn_workflow_genre %>%
#   tune_grid(
#     cv_folds_knn,
#     grid = 5) #evaluate each fold per indicated values

#load premade .rda file
load(file = here::here("week_5",  "fit_knn_cv.rda"))
```

```
# Check the performance with collect_metrics()
collect_metrics(fit_knn_cv)
```

```
## # A tibble: 10 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         2 accuracy binary     0.765     5 0.00715 Preprocessor1_Model1
##  2         2 roc_auc  binary     0.656     5 0.0204  Preprocessor1_Model1
##  3         6 accuracy binary     0.794     5 0.0109  Preprocessor1_Model2
##  4         6 roc_auc  binary     0.711     5 0.0246  Preprocessor1_Model2
##  5         7 accuracy binary     0.795     5 0.0118  Preprocessor1_Model3
##  6         7 roc_auc  binary     0.713     5 0.0242  Preprocessor1_Model3
##  7        10 accuracy binary     0.809     5 0.00913 Preprocessor1_Model4
##  8        10 roc_auc  binary     0.723     5 0.0210  Preprocessor1_Model4
##  9        12 accuracy binary     0.814     5 0.00942 Preprocessor1_Model5
## 10        12 roc_auc  binary     0.729     5 0.0197  Preprocessor1_Model5
```

```
#save file
#save(fit_knn_cv, file = "fit_knn_cv.rda")
```

```
# final workflow for KNN model
final_wf_knn <- knn_workflow_genre %>%
  finalize_workflow(select_best(fit_knn_cv, metric = "accuracy"))
```

```
#fit final workflow
final_fit_knn <- final_wf_knn %>%
  last_fit(genre_split) #giving both the test and training data together

# Collect metrics on the test data
final_fit_metrics_knn <- final_fit_knn %>% collect_metrics()

#collect prediction on test data
final_fit_prediction_knn <- final_fit_knn %>% collect_predictions()
```

Model 2: Decision Tree

```
#creating decision tree to tune hyperparameters
genre_dt_tune <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()
) %>%
  set_engine("rpart") %>%
  set_mode("classification")
```

```
#create the resample grid
tree_grid <- grid_regular(cost_complexity(), tree_depth(), min_n(), levels = 5)
```

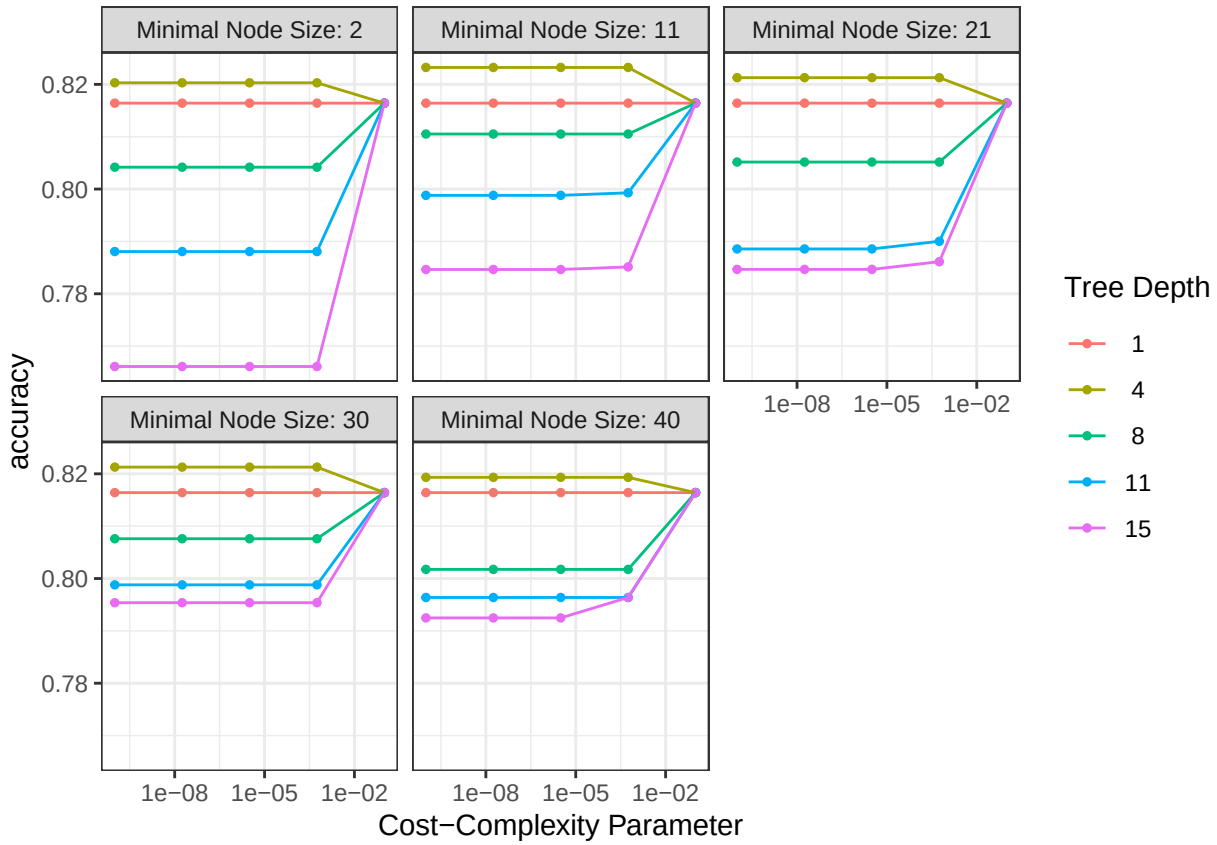
```r
#create workflow for decision tree
wf_tree_tune <- workflow() %>%
  add_recipe(rec_genre) %>%
  add_model(genre_dt_tune)
```

```r
#set seed again
set.seed(50)

#set up k-fold cv. This can be used for all the algorithms
genre_cv_tree <- genre_train %>% vfold_cv(v = 10)
```

```r
# doParallel::registerDoParallel() #build trees in parallel
# #200s
# #tune the decision tree workflow
# system.time(
#   tree_rs_tune <- tune_grid(
#     wf_tree_tune, #or tree_spec_tune
#     genre ~.,
#     resamples = genre_cv,
#     grid = tree_grid,
#     metrics = metric_set(accuracy)
#   )
# )

#save to rda file
#save(tree_rs_tune, file = "tree_rs_tune.rda")

#load in precreated .rda filw
load(file = here::here("week_5",  "tree_rs_tune.rda"))

#view the results from the tuned grid
tree_rs_tune
```

```
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits             id     .metrics           .notes
##    <list>             <chr>  <list>             <list>
##  1 <split [1843/205]> Fold01 <tibble [125 x 7]> <tibble [0 x 3]>
##  2 <split [1843/205]> Fold02 <tibble [125 x 7]> <tibble [0 x 3]>
##  3 <split [1843/205]> Fold03 <tibble [125 x 7]> <tibble [0 x 3]>
##  4 <split [1843/205]> Fold04 <tibble [125 x 7]> <tibble [0 x 3]>
##  5 <split [1843/205]> Fold05 <tibble [125 x 7]> <tibble [0 x 3]>
##  6 <split [1843/205]> Fold06 <tibble [125 x 7]> <tibble [0 x 3]>
##  7 <split [1843/205]> Fold07 <tibble [125 x 7]> <tibble [0 x 3]>
##  8 <split [1843/205]> Fold08 <tibble [125 x 7]> <tibble [0 x 3]>
##  9 <split [1844/204]> Fold09 <tibble [125 x 7]> <tibble [0 x 3]>
## 10 <split [1844/204]> Fold10 <tibble [125 x 7]> <tibble [0 x 3]>
```

```r
#plot the results
autoplot(tree_rs_tune) + theme_bw()
```




```
#find best model in the decision tree models
show_best(tree_rs_tune)
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric  .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>    <chr>      <dbl> <int>   <dbl>
## 1    0.0000000001          4    11 accuracy binary     0.823    10 0.00969
## 2    0.0000000178          4    11 accuracy binary     0.823    10 0.00969
## 3    0.00000316            4    11 accuracy binary     0.823    10 0.00969
## 4    0.000562              4    11 accuracy binary     0.823    10 0.00969
## 5    0.0000000001          4    21 accuracy binary     0.821    10 0.00970
## # i 1 more variable: .config <chr>
```

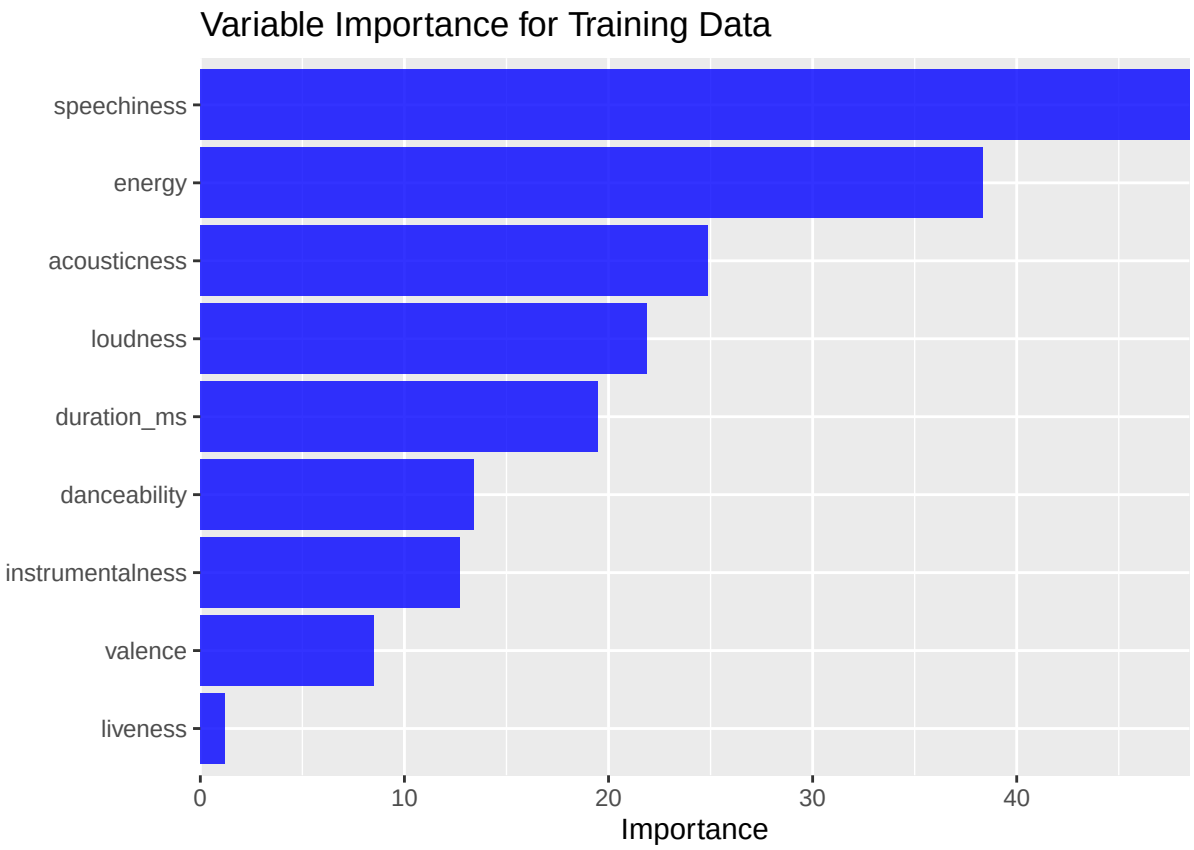
```
#finalize workflow
final_tree <- finalize_workflow(wf_tree_tune, select_best(tree_rs_tune))

#view results of finalized workflow
#final_tree
```

```
#fit the final model with the training data
final_tree_fit_train <- fit(final_tree, data = genre_train)

#fit the final model with the testing data
final_tree_fit_test <- fit(final_tree, data = genre_test)
```

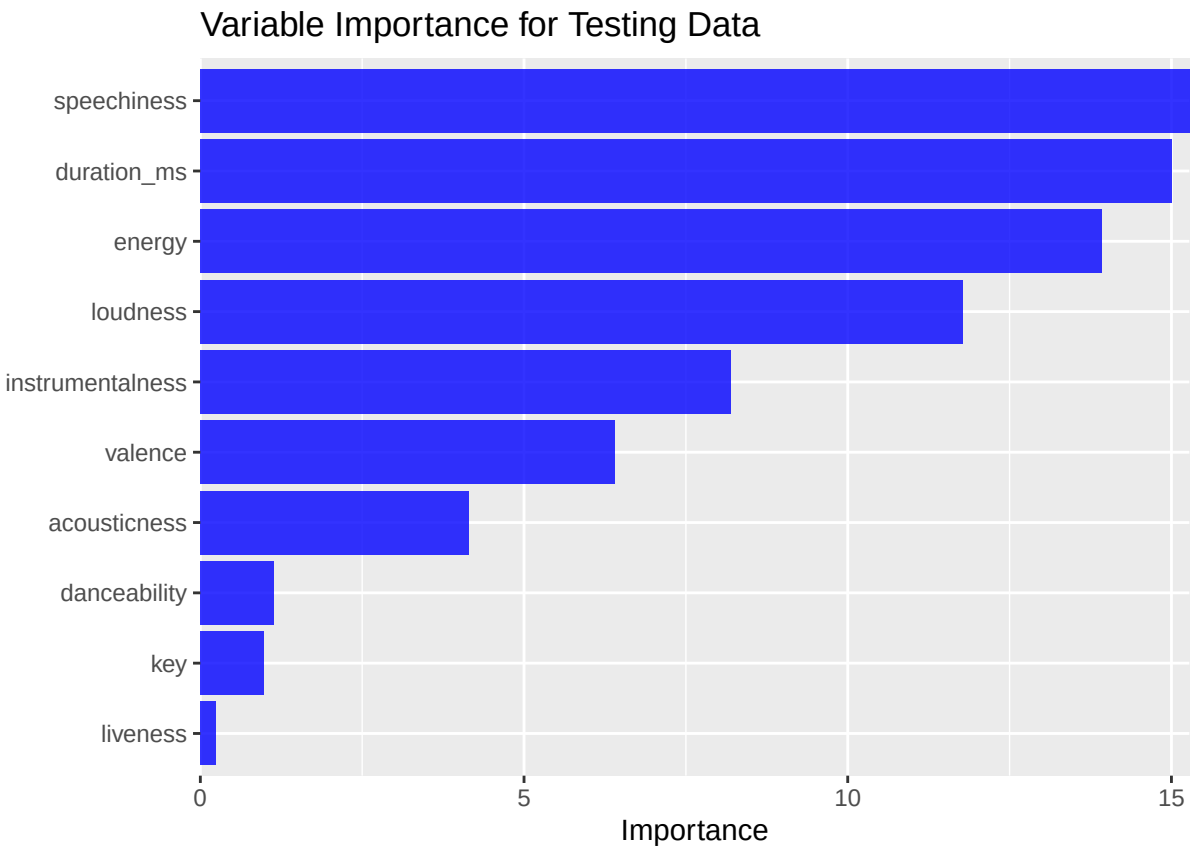
```r
#get the final model on both training and testing data and get results for testing data and get metrics
final_tree_result_metrics <- last_fit(final_tree, genre_split) %>%
  collect_metrics()

#collect predictions for testing data
final_tree_result_predictions <- last_fit(final_tree, genre_split) %>%
  collect_predictions()

#View final fit info
final_tree_result_metrics
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.838 Preprocessor1_Model1
## 2 roc_auc  binary         0.679 Preprocessor1_Model1
```

```r
#visualize the final fit and variable importance for training data
final_tree_fit_train %>%
  vip::vip(geom = "col", aesthetics = list(fill = "blue", alpha = 0.8)) +
  scale_y_continuous(expand = c(0,0)) +
  labs(title = "Variable Importance for Training Data")
```



```r
#visualize the final fit and variable importance for testing data
final_tree_fit_test %>%
  vip::vip(geom = "col", aesthetics = list(fill = "blue", alpha = 0.8)) +
  scale_y_continuous(expand = c(0,0))+
  labs(title = "Variable Importance for Testing Data")
```



Model 3: Bagged tree

```r
#creating bag tree model to tune hyperparameters
genre_bag_tune_model <-  bag_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()
) %>%
  set_engine("rpart", times = 50) %>% #set to 50 for how many trees we want to make
  set_mode("classification")
```

```r
#create the resample grid
tree_grid <- grid_regular(cost_complexity(), tree_depth(), min_n(), levels = 5)
```

```r
#create workflow for bag tree
wf_bag_tune <- workflow() %>%
```

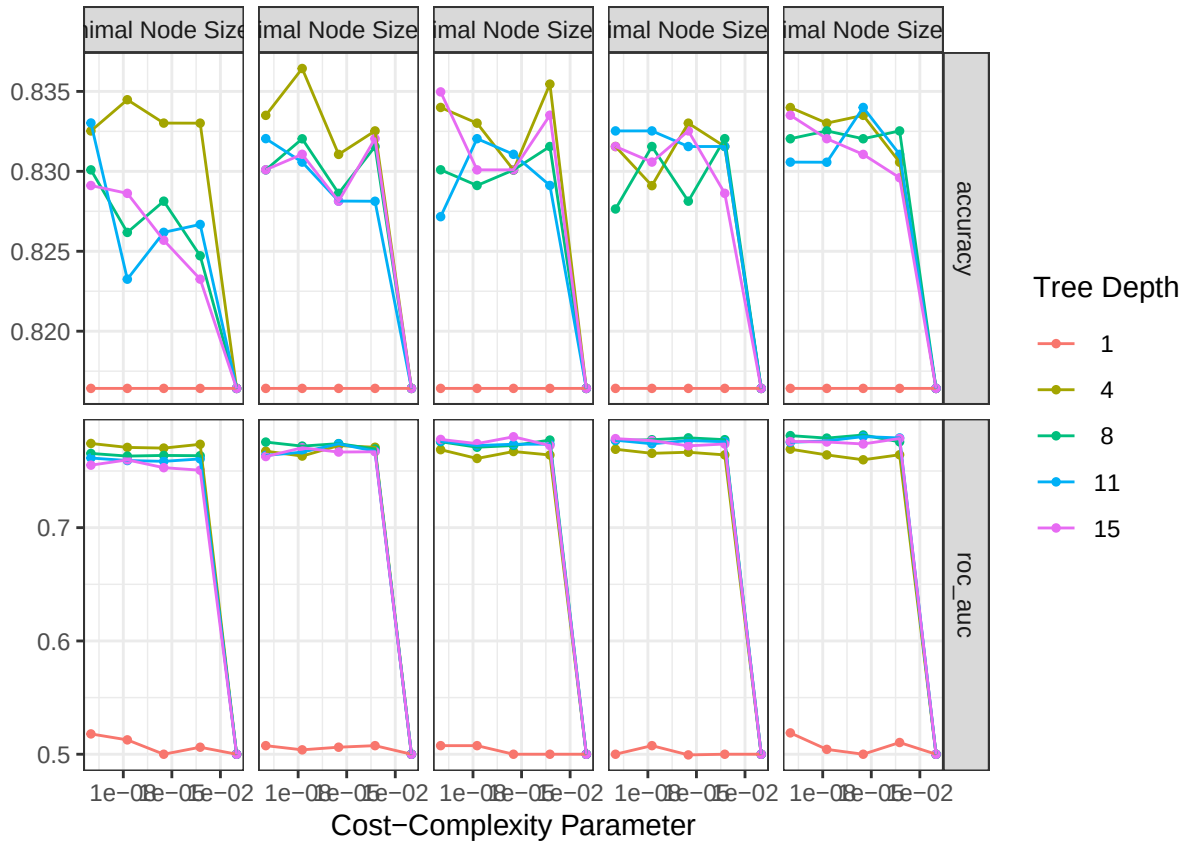
```
  add_recipe(rec_genre) %>%
  add_model(genre_bag_tune_model)
```

```
#set seed again
set.seed(50)

#set up k-fold cv
genre_cv_bag <- genre_train %>% vfold_cv(v = 5)
```

```
# doParallel::registerDoParallel() #build trees in parallel
# # #200s
# # #tune the bag tree
# system.time(
#   bag_rs_tune <- tune_grid(
#     wf_bag_tune,
#     genre ~.,
#     resamples = genre_cv_bag,
#     grid = tree_grid
#    # metrics = metric_set(accuracy)
#   )
# )

#save to rda file
# save(bag_rs_tune, file = "bag_rs_tune.rda")

#load in precreated .rda filw
load(file = here::here("week_5",  "bag_rs_tune.rda"))

#view results from tuned bag variable
bag_rs_tune
```

```
## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits             id    .metrics           .notes
##   <list>             <chr> <list>             <list>
## 1 <split [1638/410]> Fold1 <tibble [250 x 7]> <tibble [0 x 3]>
## 2 <split [1638/410]> Fold2 <tibble [250 x 7]> <tibble [0 x 3]>
## 3 <split [1638/410]> Fold3 <tibble [250 x 7]> <tibble [0 x 3]>
## 4 <split [1639/409]> Fold4 <tibble [250 x 7]> <tibble [0 x 3]>
## 5 <split [1639/409]> Fold5 <tibble [250 x 7]> <tibble [0 x 3]>
```

```
#plot the results
autoplot(bag_rs_tune) + theme_bw()
```




```
#show best model from the tuned bag variable
show_best(bag_rs_tune)
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl>
## 1    0.00000316            8    40 roc_auc binary     0.782     5  0.0178
## 2    0.0000000001          8    40 roc_auc binary     0.781     5  0.0160
## 3    0.00000316           11    40 roc_auc binary     0.780     5  0.0161
## 4    0.00000316           15    21 roc_auc binary     0.780     5  0.0176
## 5    0.00000316            8    30 roc_auc binary     0.779     5  0.0138
## # i 1 more variable: .config <chr>
```

```
#finalize workflow
final_bag <- finalize_workflow(wf_bag_tune, select_best(bag_rs_tune, metric = "accuracy"))

#show finalized workflow
#final_bag
```

```
#fit final workflow
final_bag_fit <- fit(final_bag, data = genre_train)

#fit final workflow using both training and testing data and collect metrics
final_bag_result_metrics <- last_fit(final_bag, genre_split) %>%
  collect_metrics()
```

```r
#fit final workflow and collect predictions
final_bag_result_predictions <- last_fit(final_bag, genre_split) %>%
  collect_predictions()

#view results for both fitted workflow
#final_bag_fit
final_bag_result_metrics
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.842 Preprocessor1_Model1
## 2 roc_auc  binary         0.735 Preprocessor1_Model1
```

Model 4: Random Forest

```r
#create random forest model
genre_rf_model <- rand_forest(trees = tune(), mtry = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

```r
#create random forest workflow
rf_workflow <- workflow() %>%
  add_recipe(rec_genre) %>%
  add_model(genre_rf_model)

#view workflow
#rf_workflow
```

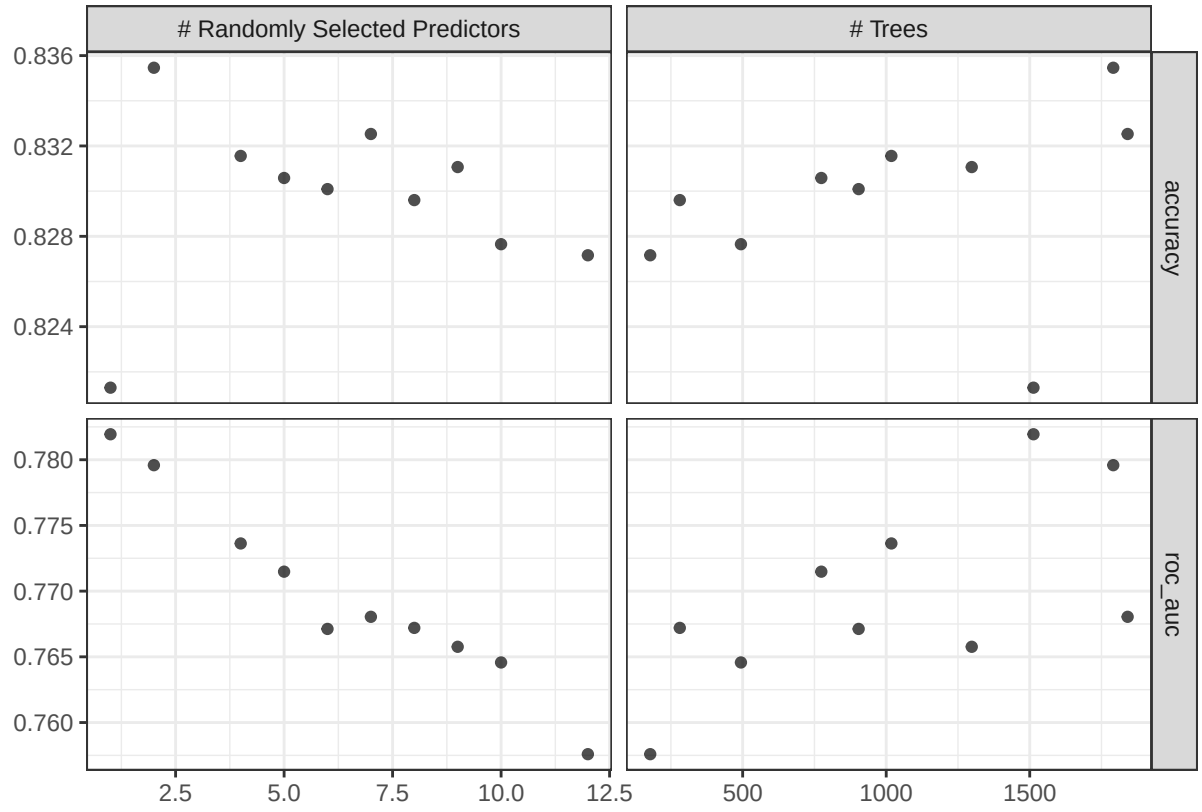
```r
#set seed again
set.seed(50)

#create the into 5 folds
cv_folds <- vfold_cv(genre_train, v = 5)
```

```r
#tune the workflow to the resample grid
# rf_cv_tune <- rf_workflow %>%
#   tune_grid(resamples = cv_folds, grid = 10)

#save the random forest tuned workflow
# save(rf_cv_tune, file = "rf_cv_tune.rda")

#load in precreated .rda filw
load(file = here::here("week_5",  "rf_cv_tune.rda"))

#view metrics of tuned random forest
head(collect_metrics(rf_cv_tune))
```

```
## # A tibble: 6 x 8
##    mtry trees .metric  .estimator  mean     n std_err .config
```

```
##   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     7  1841 accuracy binary     0.833     5 0.00507 Preprocessor1_Model01
## 2     7  1841 roc_auc  binary     0.768     5 0.0151  Preprocessor1_Model01
## 3     2  1791 accuracy binary     0.835     5 0.00672 Preprocessor1_Model02
## 4     2  1791 roc_auc  binary     0.780     5 0.0152  Preprocessor1_Model02
## 5     9  1298 accuracy binary     0.831     5 0.00607 Preprocessor1_Model03
## 6     9  1298 roc_auc  binary     0.766     5 0.0156  Preprocessor1_Model03
```

```
#plot cv results for parameter tuning
autoplot(rf_cv_tune) +
  theme_bw()
```




```
#find best model from tuned workflow variable
rf_best <- show_best(rf_cv_tune, n = 1, metric = "roc_auc") #get metrics for best random forest model

#view best model results
rf_best
```

```
## # A tibble: 1 x 8
##    mtry trees .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     1  1513 roc_auc binary     0.782     5  0.0140 Preprocessor1_Model08
```

```
#finalize workflow
rf_final <- finalize_workflow(rf_workflow,
                              select_best(rf_cv_tune, metric = "roc_auc"))
```

```
#fit the workflow to training data
train_fit_rf <- fit(rf_final, genre_train) #fit the KNN model to the training set

test_fit_rf <- last_fit(rf_final, genre_split)

#fit the workflow to both testing and training data
test_fit_rf_metrics <- last_fit(rf_final, genre_split) %>%
  collect_metrics()

test_fit_rf_predict <- last_fit(rf_final, genre_split) %>%
  collect_predictions()

#View fitted results
#train_fit_rf
test_fit_rf_metrics
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.842 Preprocessor1_Model1
## 2 roc_auc  binary         0.759 Preprocessor1_Model1
```

```
#predict using the model using testing genre
test_predict_rf <-predict(train_fit_rf, genre_test) %>%
  bind_cols(genre_test) %>%  #bind to testing column
  mutate(genre = as.factor(genre))
```

```
#find accuracy of testing predictions
accuracy(test_predict_rf, truth = genre, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.842
```

Visualization

```
options(kableExtra.auto_format = FALSE)

# kable(mean_birth_smoking, format = "latex",
#       col.names = c("Tobacco", "Mean Birth Weight", "Mean Birth Weight Difference"),
#       caption = "Mean Difference in Birth Weight between Smokers and Non-smokers") %>%
#   kable_styling(font_size = 7, latex_options = "hold_position")


#knn table
```

```
knn_table <- kable(final_fit_metrics_knn, format = 'latex', caption = "kNN fitted workflow metrics") %>%

#decision tree table
decision_tree_table <- kbl(final_tree_result_metrics, format = 'latex', caption = "Decision Tree fitted

#bagging table
bagging_table <- kbl(final_bag_result_metrics, format = 'latex', caption = "Bagging fitted workflow meti

#random forest table
random_forest_table <- kbl(test_fit_rf_metrics, format = 'latex', caption = "Random Tree fitted workflo

print(knn_table)
```

```
## \begin{table}[!h]
##
## \caption{\label{tab:unnamed-chunk-42}kNN fitted workflow metrics}
## \centering
## \fontsize{7}{9}\selectfont
## \begin{tabular}[t]{l|l|r|l}
## \hline
## .metric & .estimator & .estimate & .config\\
## \hline
## accuracy & binary & 0.8144531 & Preprocessor1\_Model1\\
## \hline
## roc\_auc & binary & 0.6819397 & Preprocessor1\_Model1\\
## \hline
## \end{tabular}
## \end{table}
```

```
print(decision_tree_table)
```

```
## \begin{table}
##
## \caption{\label{tab:unnamed-chunk-42}Decision Tree fitted workflow metrics}
## \centering
## \begin{tabular}[t]{l|l|r|l}
## \hline
## .metric & .estimator & .estimate & .config\\
## \hline
## accuracy & binary & 0.8378906 & Preprocessor1\_Model1\\
## \hline
## roc\_auc & binary & 0.6791156 & Preprocessor1\_Model1\\
## \hline
## \end{tabular}
## \end{table}
```

```
print(bagging_table)
```

```
## \begin{table}
##
## \caption{\label{tab:unnamed-chunk-42}Bagging fitted workflow metrics}
```

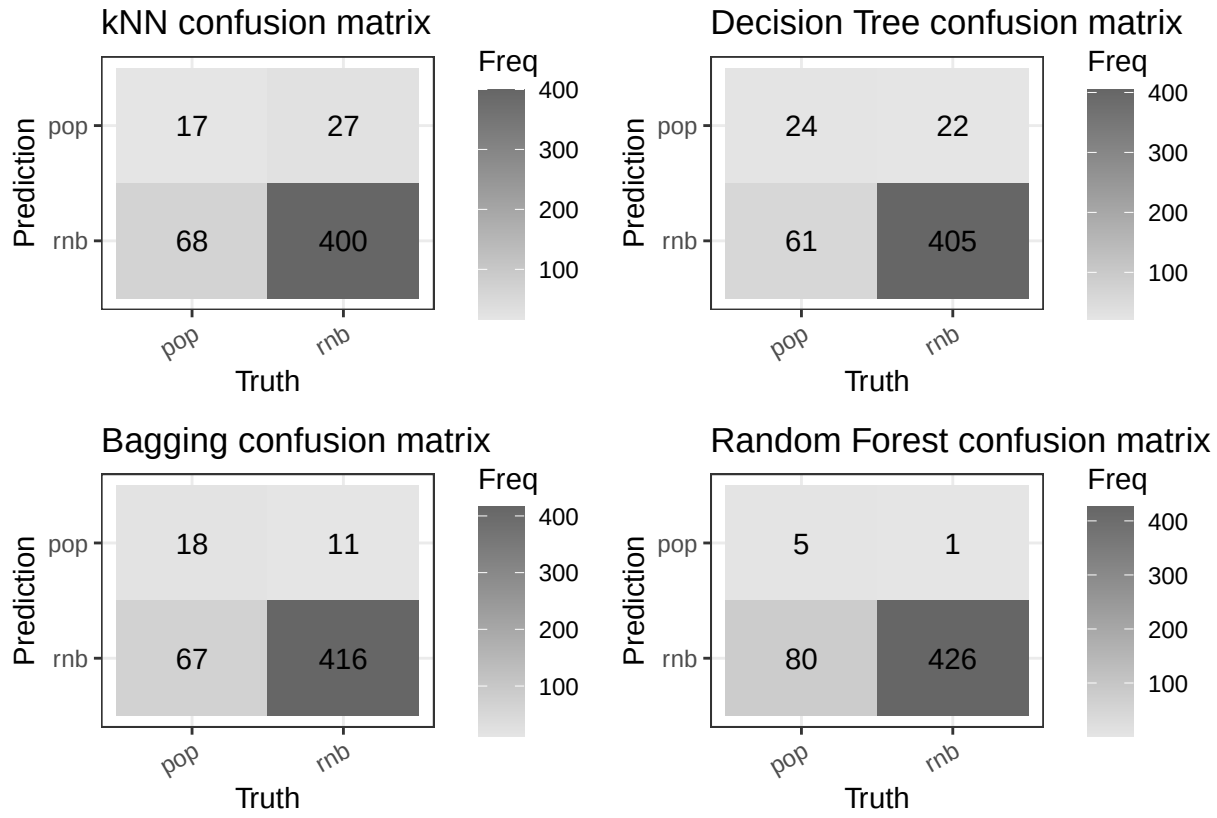
```
## \centering
## \begin{tabular}[t]{l|l|r|l}
## \hline
## .metric & .estimator & .estimate & .config\\
## \hline
## accuracy & binary & 0.8417969 & Preprocessor1\_Model1\\
## \hline
## roc\_auc & binary & 0.7346880 & Preprocessor1\_Model1\\
## \hline
## \end{tabular}
## \end{table}
```

```
print(random_forest_table)
```

```
## \begin{table}
##
## \caption{\label{tab:unnamed-chunk-42}Random Tree fitted workflow metrics}
## \centering
## \begin{tabular}[t]{l|l|r|l}
## \hline
## .metric & .estimator & .estimate & .config\\
## \hline
## accuracy & binary & 0.8417969 & Preprocessor1\_Model1\\
## \hline
## roc\_auc & binary & 0.7588649 & Preprocessor1\_Model1\\
## \hline
## \end{tabular}
## \end{table}
```

```
#confusion matrix
#k-nearest neighbor
knn_cm <- final_fit_prediction_knn %>%
  conf_mat(truth = genre, estimate = .pred_class) %>% #create confusion matrix
  autoplot(type = "heatmap") + #plot confusion matrix with heatmap
  theme_bw() + #change theme
  theme(axis.text.x = element_text(angle = 30, hjust=1)) +
  #rotate axis labels
  labs(title = "kNN confusion matrix")

#Decision Tree
df_cm <- final_tree_result_predictions %>%
  conf_mat(truth = genre, estimate = .pred_class) %>% #create confusion matrix
  autoplot(type = "heatmap") + #plot confusion matrix with heatmap
  theme_bw() + #change theme
  theme(axis.text.x = element_text(angle = 30, hjust=1)) +
  #rotate axis labels
  labs(title = "Decision Tree confusion matrix")

#Bagging
bag_cm <- final_bag_result_predictions %>%
  conf_mat(truth = genre, estimate = .pred_class) %>% #create confusion matrix
  autoplot(type = "heatmap") + #plot confusion matrix with heatmap
  theme_bw() + #change theme
```

```
  theme(axis.text.x = element_text(angle = 30, hjust=1)) +
  #rotate axis labels
  labs(title = "Bagging confusion matrix")

#Random Forest
rf_cm <- test_predict_rf %>%
  conf_mat(truth = genre, estimate = .pred_class) %>% #create confusion matrix
  autoplot(type = "heatmap") + #plot confusion matrix with heatmap
  theme_bw() + #change theme
  theme(axis.text.x = element_text(angle = 30, hjust=1)) +
  #rotate axis labels
  labs(title = "Random Forest confusion matrix")


(knn_cm + df_cm)/(bag_cm + rf_cm)
```



Reflection: Looking at all of the models, I would predict that the best model to use to determine how alike a song is from two genres is the **decision tree model**. Here, we received a estimate of about 0.841797 for the accuracy metric. For all the models, the accuracy was a better metric to use in terms of comparing the models because of its high estimate. Also seen in the plotting results for both bagging and random trees, the accuracy metric had the highest estimate verses the roc-auc estimate. Also, out of all the models and metrics combined, the decision tree had a high enough estimate that was comparable to the bagging and random forest estimate, which were slightly higher.

Looking at the confusion matrix created for each model, the decision tree arguably has a better distribution compared to the other confusion matrix. We do see that there is 61 false negatives and 22 false positives.

This confusion matrix has the lowest amount of false negatives compared to other the other models. We can interpret this as it still stating that it will state that two songs are not alike when they are, but this type of result will not happen as often as compared to the other models. Compared to the random forest confusion matrix, which has 80 false negatives verses the 1 false positive, this means that the random forest model will more often state that two songs are not alike when they are in actuality similar to each other.