Lab5

Colleen McCamy

2023-02-07

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.

Option 1, classifying by users was chosen for this lab. 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.

Spotify API Set-Up

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.

```
# client ID for the Spotify API
Sys.setenv(SPOTIFY_CLIENT_ID = '01a3a178c81140fc902174c43c18ea4a')
Sys.setenv(SPOTIFY_CLIENT_SECRET = '30062c844d7d42328414f2a682c3fd9f')

# getting access token
access_token <- get_spotify_access_token() #takes ID and SECRET, sends to Spotify and receives an acces
```

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.

```
# getting spotify data
cm_spotify_2 <- ceiling(get_my_saved_tracks(include_meta_info = TRUE)[['total']] / 50) |>
  seq() |>
  map(function(x){
    get_my_saved_tracks(limit = 50, offset = (x - 1) * 50)}) |>
  reduce(rbind) |>
  write_csv('raw_myFavTracks.cvs')
```

```r
# selecting first 6,000 rows of spotify data
cm_spotify_2 <- cm_spotify_2[(1:6000),]
```

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.

```r
# splitting the dataframe into 60 dataframes of 100 rows each
df_list <- split(cm_spotify_2, rep(1:60, each = 100))

# apply get_track_audio_features() to each split dataframe in the list since it can only do 100 requests
feature_list <- lapply(df_list,
                       function(cm_spotify_2)
                         get_track_audio_features(cm_spotify_2$track.id))

# combine all rows into one dataframe
feature_df <- do.call(rbind, feature_list)
```

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.

```r
# adding the track name column to the audio features
cm_audio_feat<- cbind(feature_df, cm_spotify_2$track.name) #|>
    #write_csv('cm_spotify_data.csv') # added write csv to provide Lewis the data
```

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.

```r
# reading in Lewis's saved songs and adding column for Lewis indicator
lw_audio_feat <- read_csv("/Users/colleenmccamy/Documents/MEDS/classes/winter/eds_232_machine_learning/
  select(-c(listener, primary_artist)) |>
  add_column(listener = 1)
```

```
## Rows: 3219 Columns: 21
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): type, id, uri, track_href, analysis_url, track.name, primary_artis...
## dbl (13): danceability, energy, key, loudness, mode, speechiness, acousticne...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# adding column for Colleen indicator
cm_audio_feat <- cm_audio_feat|>
  add_column(listener = 0) |>
  rename(track.name = "cm_spotify_2$track.name")

# joining Lewis and my audio feature data
audio_feat <- rbind(lw_audio_feat, cm_audio_feat)
```

Data Exploration & Visualization

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

Highlighting Key Stats

```
# using skim to check out the data
#skimr::skim(audio_feat)

# determining the longest track
longest_song <- max(audio_feat$duration_ms)
longest_song_title <- audio_feat$track.name[audio_feat$duration_ms == longest_song]
print(paste0("The longest song is ", "'", longest_song_title, "' at ", round(longest_song/1000/60, 2),
```

```
## [1] "The longest song is 'Note to Self' at 14.59 minutes."
```

```
# most dance-able song
dance_song <- max(audio_feat$danceability)
dance_song_title <- audio_feat$track.name[audio_feat$danceability == dance_song]
print(paste0("The most danceable song is ", "'", dance_song_title, "'."))
```

```
## [1] "The most danceable song is 'Conceited'."
```

```
# most dance-able song that we both have
# creating a similar song dataframe
similar_songs <- audio_feat |>
  mutate(duplicate = case_when(duplicated(track.name) == TRUE ~ 1,
                               duplicated(track.name) == FALSE ~ 0)) |>
  filter(duplicate == 1)

# determining most danciest song that both listeners have saved
dance_song_sim <- max(similar_songs$danceability)
dance_song_title_sim <- similar_songs$track.name[similar_songs$danceability == dance_song_sim]
print(paste0("The most danceable song that both listeners have saved is ", "'", dance_song_title_sim, "
```

```
## [1] "The most danceable song that both listeners have saved is 'SexyBack (feat. Timbaland)'."
```

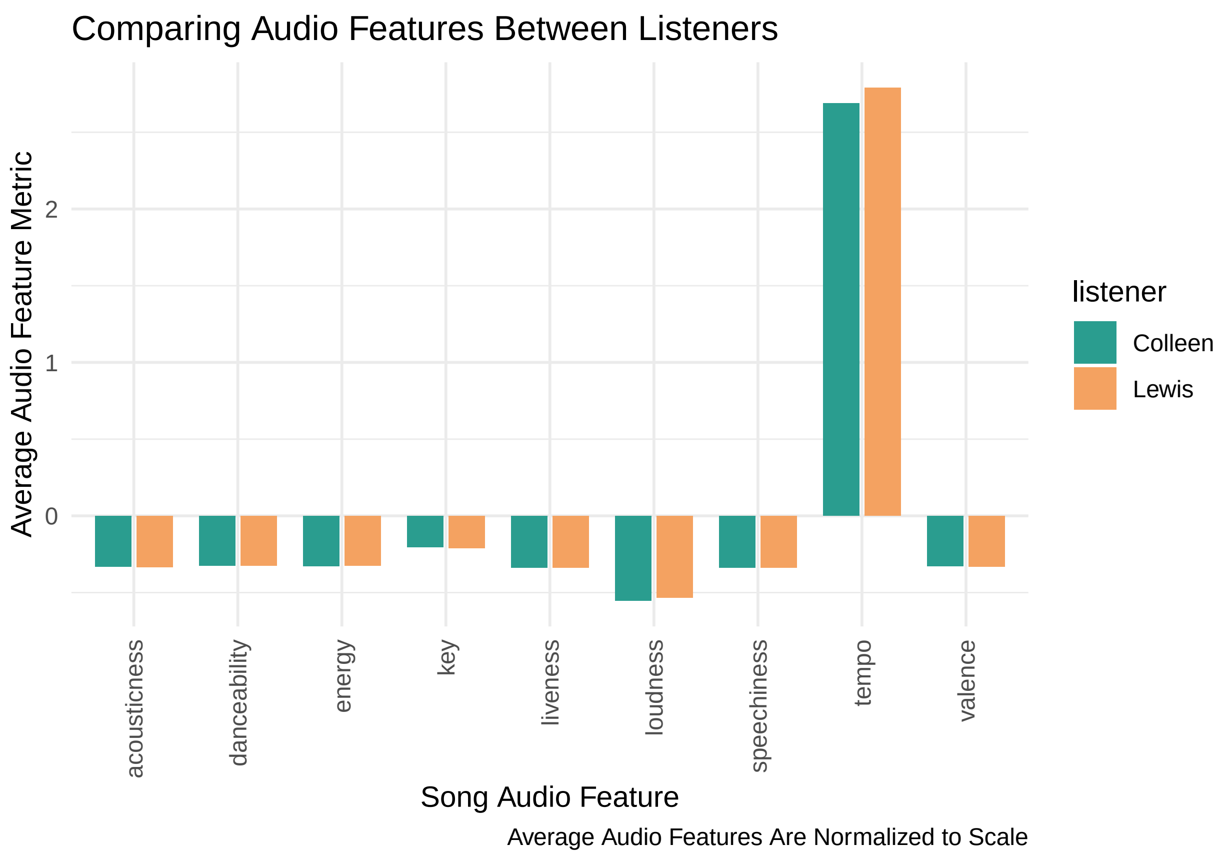
```
# determining how many hours of the same songs
roadtrip <- sum(similar_songs$duration_ms)/3600000
print(paste0("Lewis and Colleen have ", roadtrip, " hours of songs that they both like."))
```

```
## [1] "Lewis and Colleen have 76.4922608333333 hours of songs that they both like."
```

Lewis and Colleen have This means that they could go on a road trip from UC Santa Barbara to Chicago and back, and listen to songs that they both like for the entire trip.

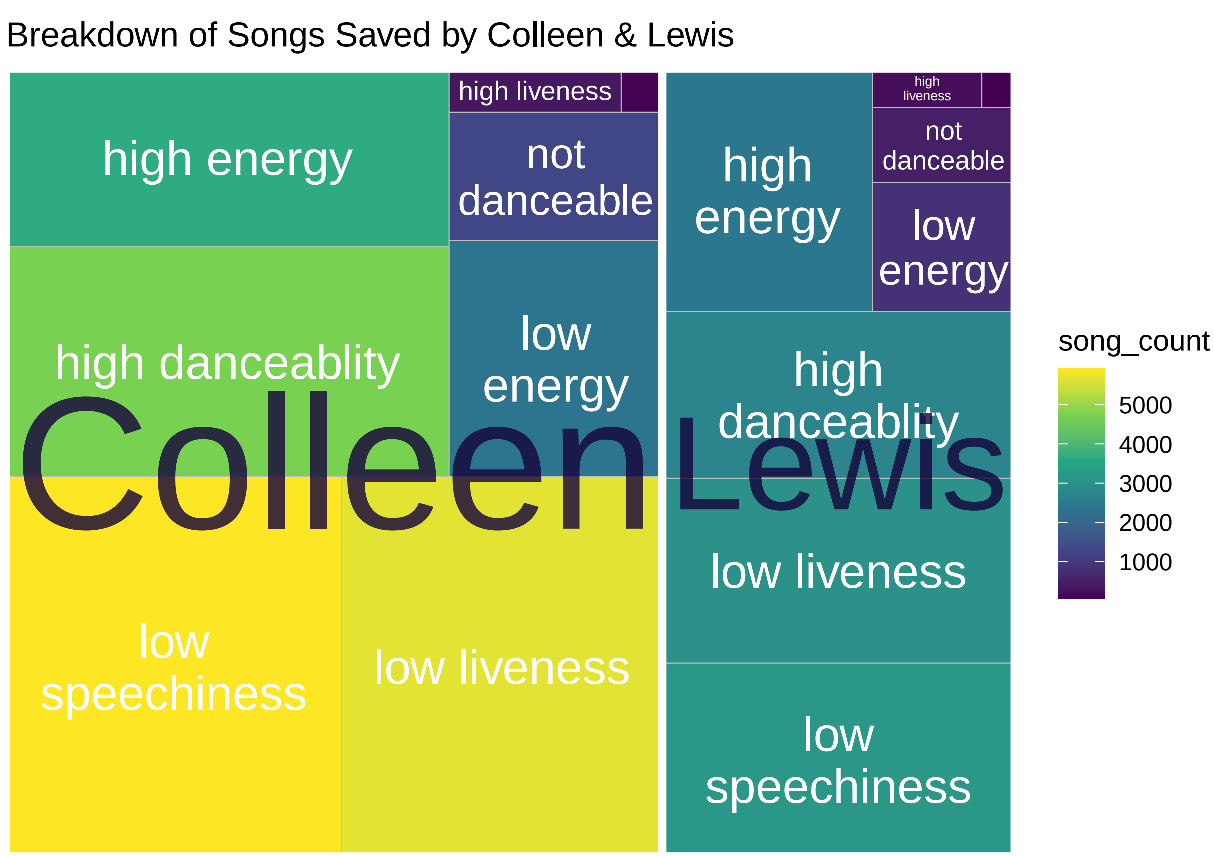
Creating Visualizations Exploring the Data

```r
# bar chart data wrangling
bar_dat <- audio_feat |>
  group_by(listener) |>
  summarize(danceability = mean(danceability),
            energy = mean(energy),
            key = mean(key),
            loudness = mean(loudness),
            speechiness = mean(speechiness),
            acousticness = mean(acousticness),
            liveness = mean(liveness),
            valence = mean(valence),
            tempo = mean(tempo)) |>
  pivot_longer(cols = danceability:tempo,
               names_to = "audio_feat",
               values_to = "mean") |>
  mutate(mean = scale(mean)) |>
  mutate(listener = case_when(listener == 0 ~ "Colleen",
                              listener == 1 ~ "Lewis"))

# creating the bar chart
audio_feat_comp <- ggplot(bar_dat,
                          aes(fill = listener,
                              y = mean,
                              x = audio_feat)) +
  geom_bar(stat="identity", width=0.7,
           position=position_dodge(width=0.8)) + theme_minimal() +
  scale_fill_manual(values = c("#2a9d8f", "#f4a261")) +
  labs(x = "Song Audio Feature",
       y = "Average Audio Feature Metric",
       caption = "Average Audio Features Are Normalized to Scale",
       title = "Comparing Audio Features Between Listeners") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

# calling back the chart
audio_feat_comp
```




```r
# creating treemap dataframe
treemap_df <- audio_feat |>
  mutate(danceability = case_when(danceability < 0.5 ~ 1,
                                   danceability >= 0.5 ~ 2)) |>
  mutate(energy = case_when(energy < 0.5 ~ 1,
                             energy >= 0.5 ~ 2)) |>
  mutate(liveness = case_when(liveness < 0.5 ~ 1,
                               liveness >= 0.5 ~2)) |>
  mutate(speechiness = case_when(speechiness < 0.5 ~ 1,
                                  speechiness >= 0.5 ~ 2)) |>
  dplyr::select(c(danceability, energy, acousticness,
                  liveness, speechiness, listener)) |>
  group_by(listener) |>
  summarize("not danceable" = sum(danceability == 1),
            "high danceablity" = sum(danceability == 2),
            "low energy" = sum(energy == 1),
            "high energy" = sum(energy == 2),
            "low liveness" = sum(liveness == 1),
            "high liveness" = sum(liveness == 2),
            "low speechiness" = sum(speechiness == 1),
            "high speechiness" = sum(speechiness == 2)) |>
   pivot_longer(cols = "not danceable":"high speechiness",
               names_to = "audio_feat",
               values_to = "song_count") |>
  mutate(listener = case_when(listener == 0 ~ "Colleen",
                               listener == 1 ~ "Lewis"))
```

```r
# plotting the treemap
tree_map <- ggplot(treemap_df, aes(area = song_count,
                       fill = song_count,
                       label = audio_feat,
                       subgroup = listener)) +
  geom_treemap() +
  geom_treemap_subgroup_border(colour="white") +
  geom_treemap_text(colour = "white",
                    place = "centre",
                    grow = F,
                    reflow = T) +
  geom_treemap_subgroup_text(place = "centre",
                             grow = T,
                             alpha = 0.8,
                             colour = "#14023b",
                             min.size = 0) +
  scale_fill_continuous(type = "viridis") +
  labs(title = "Breakdown of Songs Saved by Colleen & Lewis")

# calling back the treemap
tree_map
```



Modeling

The following four models predict song listeners using a binary outcome. Even though there are over 1,200 songs in which both Colleen and Lewis saved these models will only predict one listener.

Creating the training and testing data splits

```
# selecting columns for the data set to model
model_dat <- audio_feat |>
  select(-c("type", "id", "uri", "track_href",
            "analysis_url", "duration_ms", "time_signature",
            "track.name", "type")) |>
  mutate(listener = as.factor(listener)) |>
  mutate_if(is.ordered, factor, ordered = FALSE)

set.seed(123) # setting seed for reproducibility
#initial split of data, 70/30 split
audio_split <- initial_split(model_dat,
                             prop = 0.70,
                             strata = "listener")
audio_test <- testing(audio_split)
audio_train <- training(audio_split)
```

K Nearest Neighbors Model - Binary Outcome

```
############################################
############ Initiating the Model ##########
# ---------------------------------------- #

# setting the recipe
knn_rec <- recipe(listener ~., data = audio_train) %>%
  step_dummy(all_nominal(),-all_outcomes(),one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes(),)%>%
  prep()

#bake
baked_audio <- bake(knn_rec, audio_train)

# applying recipe to test data
baked_test <- bake(knn_rec, audio_test)

# specifying nearest neighbor model (not tuned)
knn_spec <- nearest_neighbor(neighbors = 7) |>
  set_engine("kknn") |>
  set_mode("classification")

# fitting the specification to the data
knn_fit <- knn_spec |>
  fit(listener ~ ., data = audio_train)
```

```r
# setting seed for reproducibility
set.seed(123)

# adding 5-fold cross validation to the training dataset
cv_folds <- audio_train |> vfold_cv(v = 5)

# adding this all to a workflow
knn_workflow <- workflow() |>
  add_model(knn_spec) |>
  add_recipe(knn_rec)

# adding resamples to the workflow
knn_res <- knn_workflow |>
  fit_resamples(
    resamples = cv_folds,
    control = control_resamples(save_pred = TRUE)
  )

# checking performance
knn_res |> collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.643     5 0.00571 Preprocessor1_Model1
## 2 roc_auc  binary     0.621     5 0.00388 Preprocessor1_Model1
```

```r
############################################
###### Adding Tuning to the KNN Model ######
# ---------------------------------------- #

# defining the KNN model with tuning for number of neighbors
knn_spec_tune <-
  nearest_neighbor(neighbors = tune()) |>
  set_mode("classification") |>
  set_engine("kknn")

# defining the new workflow
wf_knn_tune <- workflow() |>
  add_model(knn_spec_tune) |>
  add_recipe(knn_rec)

# fitting the workflow on the predifined folds and grid of hyperparameters
fit_knn_cv <- wf_knn_tune |>
  tune_grid(
    cv_folds,
    grid = data.frame(neighbors = c(1,5, seq(10, 100, 10)))
  )

# collecting the model metrics
fit_knn_cv |> collect_metrics()
```

```
## # A tibble: 24 x 7
```

```
##    neighbors .metric  .estimator  mean     n std_err .config
##        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy binary     0.604     5 0.00624 Preprocessor1_Model01
##  2         1 roc_auc  binary     0.560     5 0.00669 Preprocessor1_Model01
##  3         5 accuracy binary     0.631     5 0.00576 Preprocessor1_Model02
##  4         5 roc_auc  binary     0.612     5 0.00420 Preprocessor1_Model02
##  5        10 accuracy binary     0.654     5 0.00615 Preprocessor1_Model03
##  6        10 roc_auc  binary     0.633     5 0.00318 Preprocessor1_Model03
##  7        20 accuracy binary     0.670     5 0.00675 Preprocessor1_Model04
##  8        20 roc_auc  binary     0.658     5 0.00205 Preprocessor1_Model04
##  9        30 accuracy binary     0.678     5 0.00795 Preprocessor1_Model05
## 10        30 roc_auc  binary     0.669     5 0.00218 Preprocessor1_Model05
## # ... with 14 more rows
```

```
############################################
########## Finalizing the Model ############
# ---------------------------------------- #

# setting the final workflow with the initial workflow and the best model
final_wf <- wf_knn_tune |>
  finalize_workflow(select_best(fit_knn_cv, metric = "accuracy"))

# fitting the final workflow with the training data
final_fit <- final_wf |>
  fit(data = audio_train)

#############################################
# Predicting on the final model (test data) #
# ----------------------------------------- #

# adding last fit to go over the final fit and workflow
audio_knn_final <- final_fit |>
  last_fit(audio_split)

# collect the metrics
knn_metric <- audio_knn_final |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("K Nearest Neighbor")) |>
  mutate(outcome = paste("binary"))
```

Decision Tree - Binary Outcome

```
############################################
########## Setting Up the Model ############
# ---------------------------------------- #

# preprocessing the data
tree_rec <- recipe(listener ~., data = audio_train) |>
  step_dummy(all_nominal(),-all_outcomes(),one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes(),) |>
  prep()
```

```r
tree_rec_down <- recipe(listener ~., data = audio_train) |>
  step_dummy(all_nominal(),-all_outcomes(),one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes(),) |>
  step_downsample(listener) |>
  prep()

# tree specification
tree_spec_tune <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()) |> # tuning minimum node size
  set_engine("rpart") |>
             set_mode("classification")

# retrieving a tuning grid
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          min_n(),
                          levels = 5)

# setting up the decision tree workflow
wf_tree_tune <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(tree_spec_tune)

############################################
########## Run & Tune the Model ############
# ---------------------------------------- #

# setting up computing to run in parrellel
doParallel::registerDoParallel()

# using tune grid to run the model
tree_rs <-tune_grid(
  tree_spec_tune,
  listener ~ .,
  resamples = cv_folds,
  grid = tree_grid,
  metrics = metric_set(accuracy)
  )

############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best tree parameters
best_param <- select_best(tree_rs, metric = "accuracy")

# finalizing the model
final_tree <- finalize_model(tree_spec_tune, best_param)

# fitting the final model with the data
final_tree_fit <- last_fit(final_tree, listener ~ ., audio_split)
```

```r
# seeing the predictions
final_tree_fit$.predictions
```

```
## [[1]]
## # A tibble: 2,766 x 6
##    .pred_0 .pred_1  .row .pred_class listener .config
##      <dbl>   <dbl> <int> <fct>       <fct>    <chr>
##  1   0.779   0.221    12 0           1        Preprocessor1_Model1
##  2   0.753   0.247    14 0           1        Preprocessor1_Model1
##  3   0.572   0.428    16 0           1        Preprocessor1_Model1
##  4   0.658   0.342    20 0           1        Preprocessor1_Model1
##  5   0.718   0.282    24 0           1        Preprocessor1_Model1
##  6   0.865   0.135    25 0           1        Preprocessor1_Model1
##  7   0.658   0.342    29 0           1        Preprocessor1_Model1
##  8   0.826   0.174    33 0           1        Preprocessor1_Model1
##  9   0.753   0.247    37 0           1        Preprocessor1_Model1
## 10   0.865   0.135    40 0           1        Preprocessor1_Model1
## # ... with 2,756 more rows
```

```r
# collecting metrics of the decision tree model
tree_metric <- final_tree_fit |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Decision Tree")) |>
  mutate(outcome = paste("binary"))
```

Bagged Tree - Binary Outcome

```
- 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 ra
```

```r
###########################################
########## Setting Up the Model ############
# --------------------------------------- #

# specification for the bagged tree model
tree_bag_spec <- bag_tree(cost_complexity = tune(),
                          tree_depth = tune(),
                          min_n = tune()) |>
  set_engine("rpart", times = 50) |>
  set_mode("classification")

# creating the bagged tree workflow
tree_bag_wf <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(tree_bag_spec)

# setting the grid
tree_bag_grid <- grid_regular(cost_complexity(),
                              tree_depth(),
                              min_n(),
```

```r
                              levels = 5)

############################################
########## Run & Tune the Model ############
# ---------------------------------------- #

# parallel computing for speed
doParallel::registerDoParallel()

# fitting the data to the bagged tree specification
tree_bag_rs <- tune_grid(
  tree_bag_wf,
  listener ~ .,
  resamples = cv_folds,
  grid = tree_bag_grid,
  metrics = metric_set(accuracy)
  )
```

```
## Warning: The `...` are not used in this function but one or more objects were
## passed: ''
```

```r
############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best tree parameters
best_bag_param <- select_best(tree_bag_rs, metric = "accuracy")

# finalizing the model
final_tree <- finalize_model(tree_bag_spec, best_bag_param)

# fitting the final model with the data
final_tree_bag_fit <- last_fit(final_tree, listener ~ ., audio_split)

# seeing the predictions
final_tree_bag_fit$.predictions
```

```
## [[1]]
## # A tibble: 2,766 x 6
##    .pred_0 .pred_1  .row .pred_class listener .config
##      <dbl>   <dbl> <int> <fct>       <fct>    <chr>
##  1   0.711  0.289     12 0           1        Preprocessor1_Model1
##  2   0.782  0.218     14 0           1        Preprocessor1_Model1
##  3   0.572  0.428     16 0           1        Preprocessor1_Model1
##  4   0.699  0.301     20 0           1        Preprocessor1_Model1
##  5   0.711  0.289     24 0           1        Preprocessor1_Model1
##  6   0.864  0.136     25 0           1        Preprocessor1_Model1
##  7   0.424  0.576     29 1           1        Preprocessor1_Model1
##  8   0.713  0.287     33 0           1        Preprocessor1_Model1
##  9   0.765  0.235     37 0           1        Preprocessor1_Model1
## 10   0.910  0.0901    40 0           1        Preprocessor1_Model1
## # ... with 2,756 more rows
```

```r
# collecting metrics of the decision tree model
bag_metric <- final_tree_bag_fit |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Bagged Decission Trees")) |>
  mutate(outcome = paste("binary"))
```

Random Forest - Binary Outcome

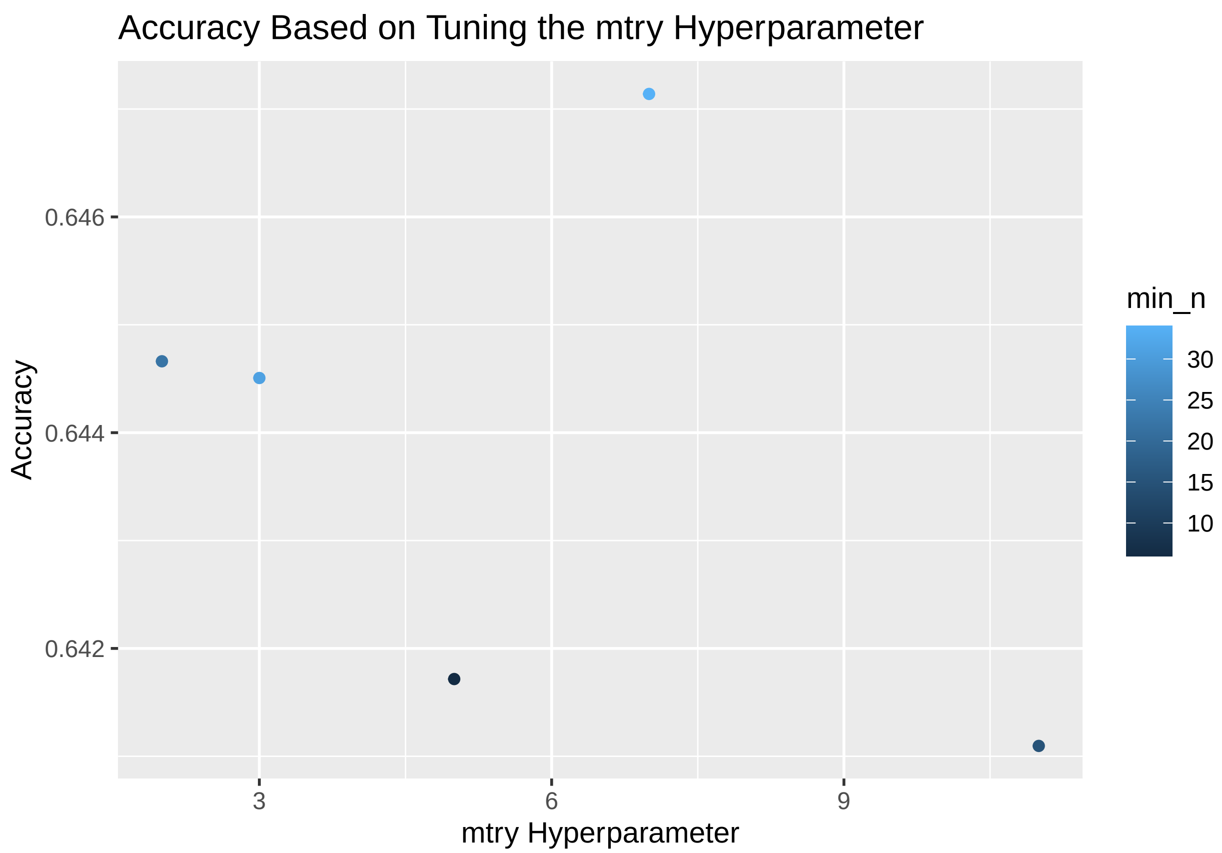
```
- rand_forest()
- m_try() is the new hyperparameter of interest for this type of model.  Make sure to include it in your
```

```r
############################################
########## Setting Up the Model ############
# ---------------------------------------- #

# defining the model
rand_spec <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()) |>
  set_mode("classification") |>
  set_engine("ranger")


# setting up a workflow
rand_wf <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(rand_spec)

############################################
########## Run & Tune the Model ############
# ---------------------------------------- #

doParallel::registerDoParallel()
set.seed(123)

# creating grid for turning on folds
rand_res <- tune_grid(
  rand_wf,
  resamples = cv_folds,
  grid = 5
)
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

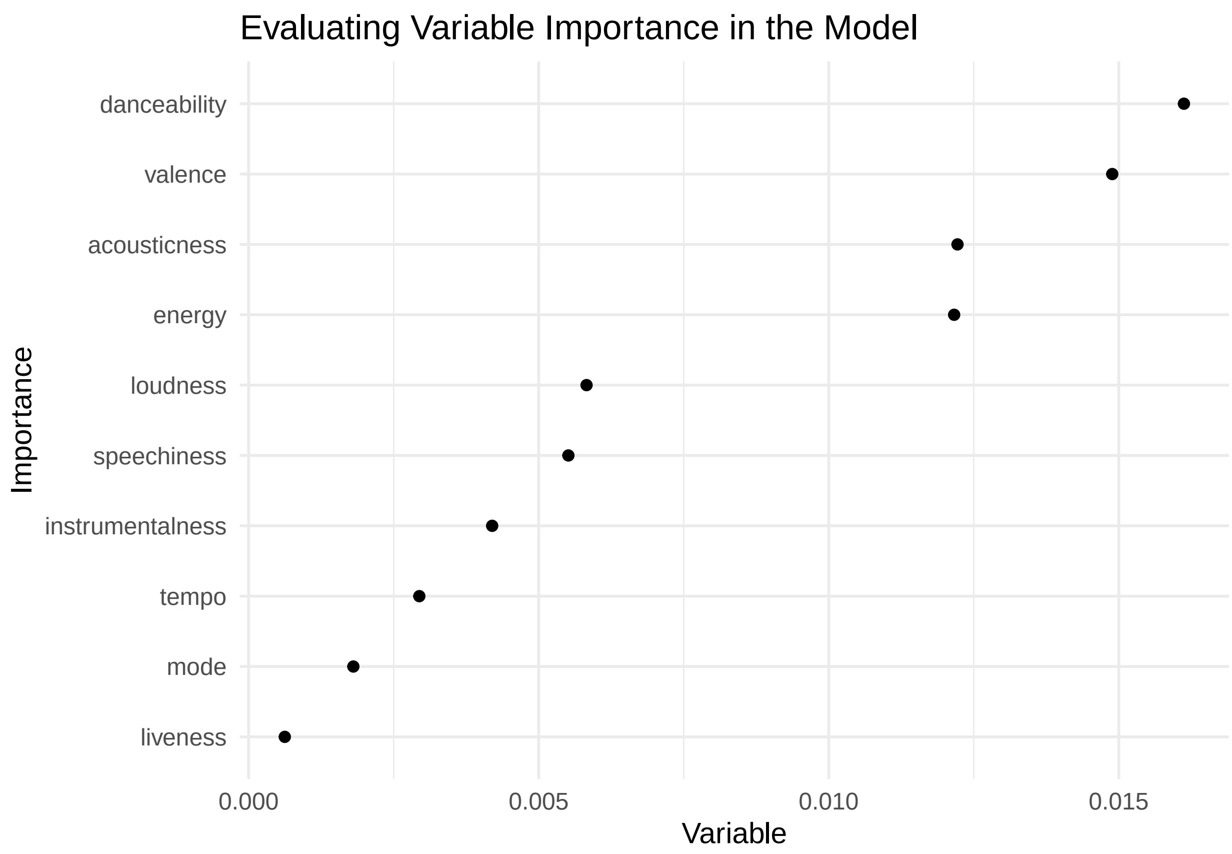
```r
# plotting the tuning results
rand_res |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  ggplot(aes(x = mtry,
             y = mean,
             color = min_n)) +
  geom_point(show.legend = TRUE) +
```

```r
  labs(x = "mtry Hyperparameter",
       y = "Accuracy",
       title = "Accuracy Based on Tuning the mtry Hyperparameter")
```




```r
#############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best model
best_rand <- select_best(rand_res, metric = "accuracy")

# finalizing the model
final_rand <- finalize_model(
  rand_spec,
  best_rand
)


#############################################
########### Evaluating the Model ###########
# ---------------------------------------- #

# looking at variable importance
final_rand |>
  set_engine("ranger",
```

```
            importance = "permutation") |>
  fit(listener ~ .,
      data = audio_train) |>
  vip(geom = "point") + theme_minimal() +
  labs(x = "Importance",
       y = "Variable",
       title = "Evaluating Variable Importance in the Model")
```




```
# verifying model on testing data
final_rand_wf <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(final_rand)

# fitting the last data
rand_result <- final_rand_wf |>
  last_fit(audio_split)

rand_metric <- rand_result |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Random Forest")) |>
  mutate(outcome = paste("binary"))
```

Evaluating Model Performance - Binary Outcome

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_metrics <- rbind(knn_metric, tree_metric, bag_metric, rand_metric) |>
  dplyr::select(model, .estimate, outcome) |>
  rename("Model Type" = model,
          Accuracy = .estimate,
         Outcome = outcome)

model_metrics |> gt::gt()
```

Model Type	Accuracy	Outcome
K Nearest Neighbor	0.6854664	binary
Decision Tree	0.6702820	binary
Bagged Decission Trees	0.6858279	binary
Random Forest	0.6388286	binary

RESPONSE: We can see that the bagged decision trees model was most accurate with k-nearest neighbor following close behind. However, this was one of the most computationally most intense model. Looking at these accuracy metrics, I would suggest to perform the K Nearest Neighbor to keep the accuracy but reduce the computational need.

Modeling Part II: Non-Binary Outcome

Adding a Similarity Category for Three Output Variables (Non-Binary Outcome)

The following models conduct the same analysis as above, however include three outcome options. Colleen to save the song, Lewis to save the song, or both to have saved the song.

```
############################################
######### Creating the Data Split ##########
# -------------------------------------- #

# selecting columns for the data set to model
model_dat_similar <- audio_feat |>
    mutate(duplicate = case_when(duplicated(track.name) == TRUE ~ 1,
                           duplicated(track.name) == FALSE ~ 0)) |>
    mutate(listener = case_when(duplicate == 1 ~ 2,
                                duplicate == 0 ~ listener)) |>
  dplyr::select(-c("type", "id", "uri", "track_href",
             "analysis_url", "duration_ms", "time_signature",
             "track.name", "type", "duplicate")) |>
   mutate(listener = as.factor(listener)) |>
```

```
  mutate_if(is.ordered, factor, ordered = FALSE)

set.seed(123) # setting seed for reproducibility
#initial split of data, 70/30 split
audio_split_sim <- initial_split(model_dat_similar,
                             prop = 0.70,
                             strata = "listener")
audio_test_sim <- testing(audio_split_sim)
audio_train_sim <- training(audio_split_sim)
```

K-Nearest Neighbor - Non-Binary

```
############################################
############ Initiating the Model ##########
# ---------------------------------------- #

# setting the recipe
knn_rec <- recipe(listener ~., data = audio_train_sim) %>%
  step_dummy(all_nominal(),-all_outcomes(),one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes(),)%>%
  prep()

#bake
baked_audio <- bake(knn_rec, audio_test_sim)

# applying recipe to test data
baked_test <- bake(knn_rec, audio_test_sim)

# specifying nearest neighbor model (not tuned)
knn_spec <- nearest_neighbor(neighbors = 7) |>
  set_engine("kknn") |>
  set_mode("classification")

# fitting the specification to the data
knn_fit <- knn_spec |>
  fit(listener ~ ., data = audio_train_sim)

# setting seed for reproducibility
set.seed(123)

# adding 5-fold cross validation to the training dataset
cv_folds_sim <- audio_train_sim |> vfold_cv(v = 5)

# adding this all to a workflow
knn_workflow <- workflow() |>
  add_model(knn_spec) |>
  add_recipe(knn_rec)

# adding resamples to the workflow
knn_res <- knn_workflow |>
  fit_resamples(
```

```r
    resamples = cv_folds_sim,
    control = control_resamples(save_pred = TRUE)
  )

# checking performance
knn_res |> collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.512     5 0.00283 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.562     5 0.00427 Preprocessor1_Model1
```

```r
############################################
###### Adding Tuning to the KNN Model ######
# -------------------------------------- #

# defining the KNN model with tuning for number of neighbors
knn_spec_tune <-
  nearest_neighbor(neighbors = tune()) |>
  set_mode("classification") |>
  set_engine("kknn")

# defining the new workflow
wf_knn_tune <- workflow() |>
  add_model(knn_spec_tune) |>
  add_recipe(knn_rec)

# fitting the workflow on the predifined folds and grid of hyperparameters
fit_knn_cv <- wf_knn_tune |>
  tune_grid(
    cv_folds_sim,
    grid = data.frame(neighbors = c(1,5, seq(10, 100, 10)))
  )

# collecting the model metrics
fit_knn_cv |> collect_metrics()
```

```
## # A tibble: 24 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy multiclass 0.463     5 0.00435 Preprocessor1_Model01
##  2         1 roc_auc  hand_till  0.527     5 0.00277 Preprocessor1_Model01
##  3         5 accuracy multiclass 0.483     5 0.00461 Preprocessor1_Model02
##  4         5 roc_auc  hand_till  0.557     5 0.00387 Preprocessor1_Model02
##  5        10 accuracy multiclass 0.531     5 0.00370 Preprocessor1_Model03
##  6        10 roc_auc  hand_till  0.570     5 0.00485 Preprocessor1_Model03
##  7        20 accuracy multiclass 0.564     5 0.00303 Preprocessor1_Model04
##  8        20 roc_auc  hand_till  0.586     5 0.00510 Preprocessor1_Model04
##  9        30 accuracy multiclass 0.575     5 0.00572 Preprocessor1_Model05
## 10        30 roc_auc  hand_till  0.593     5 0.00569 Preprocessor1_Model05
## # ... with 14 more rows
```

```
############################################
########## Finalizing the Model ############
# ---------------------------------------- #

# setting the final workflow with the initial workflow and the best model
final_wf <- wf_knn_tune |>
  finalize_workflow(select_best(fit_knn_cv, metric = "accuracy"))

# fitting the final workflow with the training data
final_fit <- final_wf |>
  fit(data = audio_train_sim)

#############################################
# Predicting on the final model (test data) #
# ----------------------------------------- #

# adding last fit to go over the final fit and workflow
audio_knn_final <- final_fit |>
  last_fit(audio_split_sim)

# collect the metrics
knn_metric_sim <- audio_knn_final |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("K Nearest Neighbor"))|>
  mutate(outcome = paste("non-binary"))
```

Decision Tree - Non-Binary

```
############################################
########## Setting Up the Model ############
# ---------------------------------------- #

# preprocessing the data
tree_rec_down <- recipe(listener ~., data = audio_train_sim) |>
  step_dummy(all_nominal(),-all_outcomes(),one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes(),) |>
  step_downsample(listener) |>
  prep()

# tree specification
tree_spec_tune <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()) |> # tuning minimum node size
  set_engine("rpart") |>
             set_mode("classification")

# retrieving a tuning grid
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          min_n(),
```

```
                          levels = 5)

# setting up the decision tree workflow
wf_tree_tune <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(tree_spec_tune)

############################################
########## Run & Tune the Model ############
# -------------------------------------- #

# setting up computing to run in parrellel
doParallel::registerDoParallel()

# using tune grid to run the model
tree_rs <-tune_grid(
  tree_spec_tune,
  listener ~ .,
  resamples = cv_folds_sim,
  grid = tree_grid,
  metrics = metric_set(accuracy)
  )

############################################
###### Finalizing & Fitting the Model ######
# -------------------------------------- #

# selecting the best tree parameters
best_param <- select_best(tree_rs, metric = "accuracy")

# finalizing the model
final_tree <- finalize_model(tree_spec_tune, best_param)

# fitting the final model with the data
final_tree_fit <- last_fit(final_tree, listener ~ ., audio_split_sim)

# seeing the predictions
final_tree_fit$.predictions
```

```
## [[1]]
## # A tibble: 2,767 x 7
##    .pred_0 .pred_1 .pred_2  .row .pred_class listener .config
##      <dbl>   <dbl>   <dbl> <int> <fct>       <fct>    <chr>
##  1   0.632   0.238   0.130     4 0           1        Preprocessor1_Model1
##  2   0.632   0.238   0.130     8 0           1        Preprocessor1_Model1
##  3   0.225   0.641   0.134    10 1           1        Preprocessor1_Model1
##  4   0.632   0.238   0.130    11 0           1        Preprocessor1_Model1
##  5   0.632   0.238   0.130    12 0           1        Preprocessor1_Model1
##  6   0.632   0.238   0.130    14 0           1        Preprocessor1_Model1
##  7   0.632   0.238   0.130    15 0           1        Preprocessor1_Model1
##  8   0.632   0.238   0.130    19 0           1        Preprocessor1_Model1
##  9   0.632   0.238   0.130    22 0           1        Preprocessor1_Model1
## 10   0.553   0.321   0.126    24 0           1        Preprocessor1_Model1
```

```
## # ... with 2,757 more rows
```

```r
# collecting metrics of the decision tree model
tree_metric_sim <- final_tree_fit |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Decision Tree")) |>
  mutate(outcome = paste("non-binary"))
```

Bagged Tree - Non-Binary

```r
###########################################
########## Setting Up the Model ############
# --------------------------------------- #

# specification for the bagged tree model
tree_bag_spec <- bag_tree(cost_complexity = tune(),
                          tree_depth = tune(),
                          min_n = tune()) |>
  set_engine("rpart", times = 50) |>
  set_mode("classification")

# creating the bagged tree workflow
tree_bag_wf <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(tree_bag_spec)

# setting the grid
tree_bag_grid <- grid_regular(cost_complexity(),
                              tree_depth(),
                              min_n(),
                              levels = 5)

###########################################
########## Run & Tune the Model ############
# --------------------------------------- #

# parallel computing for speed
doParallel::registerDoParallel()

# fitting the data to the bagged tree specification
tree_bag_rs <- tune_grid(
  tree_bag_wf,
  listener ~ .,
  resamples = cv_folds_sim,
  grid = tree_bag_grid,
  metrics = metric_set(accuracy)
  )
```

```
## Warning: The `...` are not used in this function but one or more objects were
## passed: ''
```

```r
#############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best tree parameters
best_bag_param <- select_best(tree_bag_rs, metric = "accuracy")

# finalizing the model
final_tree <- finalize_model(tree_bag_spec, best_bag_param)

# fitting the final model with the data
final_tree_bag_fit <- last_fit(final_tree, listener ~ ., audio_split_sim)

# seeing the predictions
final_tree_bag_fit$.predictions
```

```
## [[1]]
## # A tibble: 2,767 x 7
##    .pred_0 .pred_1 .pred_2  .row .pred_class listener .config
##      <dbl>   <dbl>   <dbl> <int> <fct>       <fct>    <chr>
##  1   0.534   0.335   0.131     4 0           1        Preprocessor1_Model1
##  2   0.534   0.335   0.131     8 0           1        Preprocessor1_Model1
##  3   0.534   0.335   0.131    10 0           1        Preprocessor1_Model1
##  4   0.534   0.335   0.131    11 0           1        Preprocessor1_Model1
##  5   0.534   0.335   0.131    12 0           1        Preprocessor1_Model1
##  6   0.534   0.335   0.131    14 0           1        Preprocessor1_Model1
##  7   0.534   0.335   0.131    15 0           1        Preprocessor1_Model1
##  8   0.534   0.335   0.131    19 0           1        Preprocessor1_Model1
##  9   0.534   0.335   0.131    22 0           1        Preprocessor1_Model1
## 10   0.534   0.335   0.131    24 0           1        Preprocessor1_Model1
## # ... with 2,757 more rows
```

```r
# collecting metrics of the decision tree model
bag_metric_sim <- final_tree_bag_fit |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Bagged Decission Trees")) |>
  mutate(outcome = paste("non-binary"))
```

Random Forest - Non-Binary

```r
# defining the model
rand_spec <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()) |>
  set_mode("classification") |>
  set_engine("ranger")


# setting up a workflow
rand_wf <- workflow() |>
```

```r
  add_recipe(tree_rec_down) |>
  add_model(rand_spec)

############################################
########## Run & Tune the Model ############
# ---------------------------------------- #

doParallel::registerDoParallel()
set.seed(123)

# creating grid for turning on folds
rand_res <- tune_grid(
  rand_wf,
  resamples = cv_folds_sim,
  grid = 5
)
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

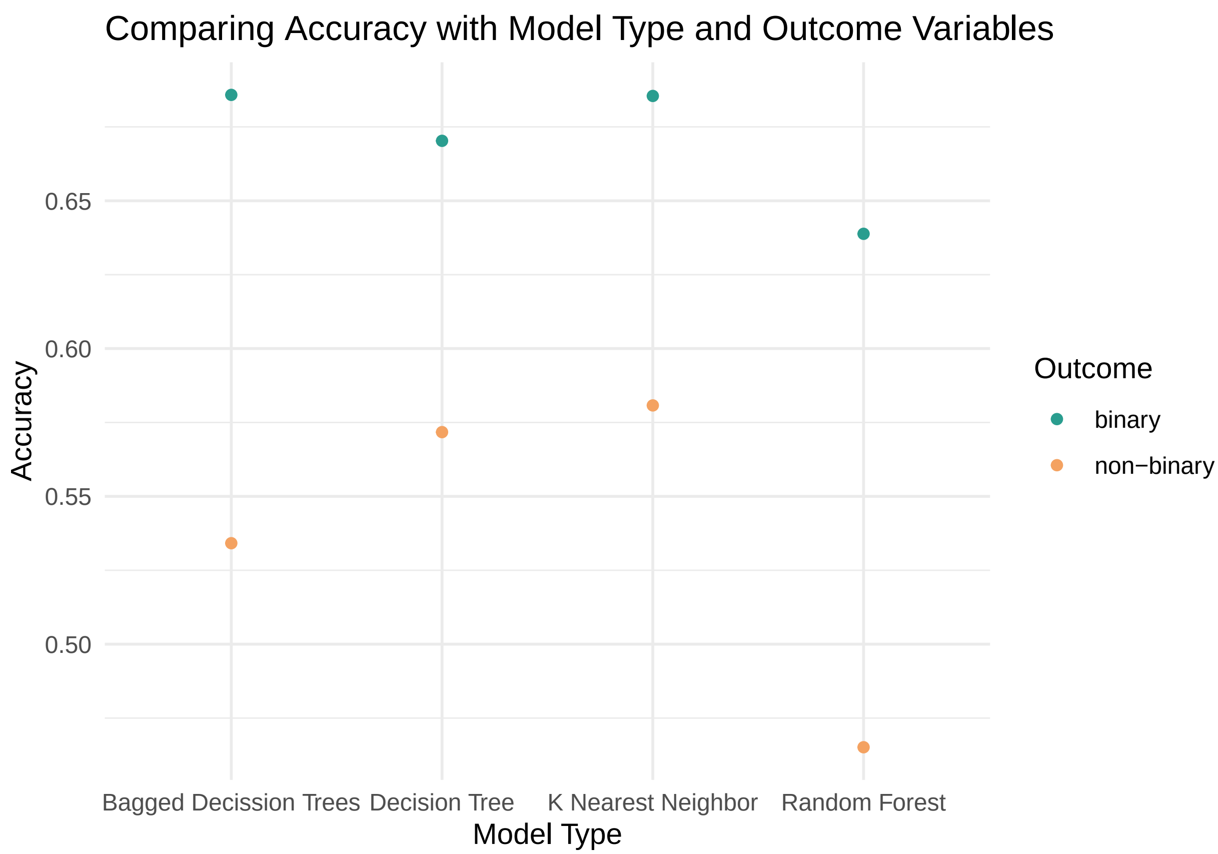
```r
############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best model
best_rand <- select_best(rand_res, metric = "accuracy")

# finalizing the model
final_rand <- finalize_model(
  rand_spec,
  best_rand
)


############################################
########### Evaluating the Model ###########
# ---------------------------------------- #

# verifying model on testing data
final_rand_wf <- workflow() |>
  add_recipe(tree_rec_down) |>
  add_model(final_rand)

rand_result <- final_rand_wf |>
  last_fit(audio_split_sim)

rand_metric_sim <- rand_result |> collect_metrics() |>
  filter(.metric == "accuracy") |>
  mutate(model = paste("Random Forest"))  |>
  mutate(outcome = paste("non-binary"))
```

Evaluating Model Performance of the All Models Ran

```r
# combining all final metrics
model_metrics_total <- rbind(knn_metric, knn_metric_sim,
                             tree_metric, tree_metric_sim,
                             bag_metric, bag_metric_sim,
                             rand_metric, rand_metric_sim) |>
  dplyr::select(model, .estimate, outcome) |>
  rename(model_type = model,
          Accuracy = .estimate,
         Outcome = outcome)

# putting these into a table
model_metrics_total |> gt::gt()
```

model_type	Accuracy	Outcome
K Nearest Neighbor	0.6854664	binary
K Nearest Neighbor	0.5807734	non-binary
Decision Tree	0.6702820	binary
Decision Tree	0.5717383	non-binary
Bagged Decission Trees	0.6858279	binary
Bagged Decission Trees	0.5341525	non-binary
Random Forest	0.6388286	binary
Random Forest	0.4651247	non-binary

```r
# creating a plot looking at these differences
accuracy_plot <- ggplot(model_metrics_total, aes(x = model_type,
                                                 y = Accuracy,
                                                 col = Outcome)) +
  geom_point() +
  scale_color_manual(values = c("#2a9d8f", "#f4a261")) +
  theme_minimal() +
  labs(x = "Model Type",
       y = "Accuracy",
       title = "Comparing Accuracy with Model Type and Outcome Variables")

# calling back the plot
accuracy_plot
```



RESPONSE: It is interesting to see that for all models adding a third outcome which would indicate that both Lewis and Colleen liked the song decreased the accuracy of the models. I would assume this is because it would be harder for the model to distinguish which songs both Colleen and Lewis would save compared to which songs only one of them would save.