Lab 5

Elke Windschitl & Lewis White

2023-02-08

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 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 (https://www.spotify.com/us/signup)

Once you have an account, go to Spotify for developers (https://developer.spotify.com/ (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. On the app's dashboard page you will find your Client ID just under the header name of your app. Click "Show Client Secret" to access your secondary Client ID. When you do this you'll be issued a Spotify client ID and client secret key.

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.

```
# Load libraries
library(spotifyr) #API interaction
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0     ✔ purrr   1.0.1
## ✔ tibble  3.1.8     ✔ dplyr   1.1.0
## ✔ tidyr   1.3.0     ✔ stringr 1.5.0
## ✔ readr   2.1.3     ✔ forcats 1.0.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## ── Attaching packages ────────────────────────────────────── tidymodels 1.0.0 ──
## ✔ broom        1.0.3     ✔ rsample      1.1.1
## ✔ dials        1.1.0     ✔ tune         1.0.1
## ✔ infer        1.0.4     ✔ workflows    1.1.2
## ✔ modeldata    1.1.0     ✔ workflowsets 1.0.0
## ✔ parsnip      1.0.3     ✔ yardstick    1.1.0
## ✔ recipes      1.0.4
## ── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(rsample)
library(recipes)
library(skimr)
library(kknn)
library(hrbrthemes)
```

```
## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use these themes.
##       Please use hrbrthemes::import_roboto_condensed() to install Roboto Condensed an
d
##       if Arial Narrow is not on your system, please see https://bit.ly/arialnarrow
```

```
library(viridis)
```

```
## Loading required package: viridisLite
##
## Attaching package: 'viridis'
##
## The following object is masked from 'package:scales':
##
##     viridis_pal
```

```
library(workflows)
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.

```
access_token <- get_spotify_access_token() #takes ID and SECRET, sends to Spotify and re
ceives an access token
```

> *This may result in an error:*
>
> INVALID_CLIENT: Invalid redirect URI
>
> *This can be resolved by editing the callback settings on your app. Go to your app and click "Edit Settings". Under redirect URLs paste this: http://localhost:1410/ (http://localhost:1410/) and click save at the bottom.*

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.

```
# Get first 50 songs
my_songs <- get_my_saved_tracks(limit = 50)
```

```
## Auto-refreshing stale OAuth token.
```

```
# for each set of 50 songs, bind to my_songs
for(i in seq(50, 450, 50)) {
  songs <- get_my_saved_tracks(limit = 50, offset(i))
  my_songs <- rbind(my_songs, songs)
}

#selecting the track id and track name from liked tracks so I can use left_join and only add the track name to the audio features data set
my_songs_for_joining <- my_songs %>%
  select(track.id, track.name, track.artists) %>%
  mutate(primary_artist = unlist(lapply(my_songs$track.artists,
                                        function(x) x$name[1]))) %>%
  select(-track.artists)
```

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.

```
# Initiate empty song features vector
song_features <- c()

# Retreive the song features on track.id for all songs
for(i in seq(1, 401, 100)) {
  feats <- get_track_audio_features(my_songs$track.id[seq(i, (i + 99), 1)])
  song_features <- rbind(song_features, feats)
}

# Bind song names to track features
song_features <- cbind(song_features, my_songs_for_joining) %>%
  select(-track.id)

#write_csv(song_features, "elke_liked_tracks.csv")
```

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.

```
lewis_liked_tracks <- read_csv("lewis_liked_tracks.csv") %>%
  mutate(listener = "Lewis")
```

```
## Rows: 1050 Columns: 20
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (7): type, id, uri, track_href, analysis_url, track.name, primary_artist
## 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.
```

```
elke_liked_tracks <- song_features %>%
  mutate(listener = "Elke")

all_tracks <- rbind(lewis_liked_tracks, elke_liked_tracks) %>%
  select(-(type:analysis_url)) #remove unnecessary columns
```

Data Exploration

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 users?

From the brief (and non-exhaustive) exploration below, we can see that Lewis on average listens to songs that are more danceable, energetic, and speechy than me. I tend to listen to songs that are more acoustic than Lewis. Also, I listen to slightly longer songs than Lewis. There is not a significant difference (at a

significance level of 0.05) between the instrumentalness of the songs that Lewis and I listen to.

```
# Sort and find my top 5 artists
sorted_table_e <- sort(table(elke_liked_tracks$primary_artist), decreasing = TRUE)
top_five_artists_e <- sorted_table_e[1:5]
print(top_five_artists_e)
```

```
##
##    Noah Kahan  Taylor Swift   Miley Cyrus    Lord Huron The Lumineers
##            58            34            25            16            12
```

```
# Sort and find lewis's top 5 artists
sorted_table_l <- sort(table(lewis_liked_tracks$primary_artist), decreasing = TRUE)
top_five_artists_l <- sorted_table_l[1:5]
print(top_five_artists_l)
```

```
##
##     Lana Del Rey     BROCKHAMPTON       Kanye West Childish Gambino
##               48               22               20               18
##            Drake
##               18
```

```
# Find my most danceable songs
dancable <- elke_liked_tracks %>%
  arrange(desc(danceability)) %>%
  slice(1:5)
print(dancable[,c("track.name", "danceability")])
```

```
##                                                track.name danceability
## 1 This Must Be the Place (Naive Melody) - 2005 Remaster        0.942
## 2                        WAP (feat. Megan Thee Stallion)        0.935
## 3                                                   4x4        0.925
## 4                      Let Me Down (with Chelsea Cutler)        0.843
## 5                                                Gnarly        0.841
```

```
# Find my most acoustic songs
acoustic <- elke_liked_tracks %>%
  arrange(desc(acousticness)) %>%
  slice(1:5)
print(acoustic[,c("track.name", "acousticness")])
```

```
##   track.name acousticness
## 1      saman        0.994
## 2  That Home        0.991
## 3      Wash.        0.990
## 4  Interlude        0.988
## 5     Wolves        0.972
```

```
# Find my highest energy songs
energy <- elke_liked_tracks %>%
  arrange(desc(energy)) %>%
  slice(1:5)
print(energy[,c("track.name", "energy")])
```

```
##                              track.name energy
## 1                                   Sex  0.974
## 2                     All Day All Night  0.946
## 3 Sunday Bloody Sunday - Remastered 2008  0.944
## 4        Go Your Own Way - 2004 Remaster  0.941
## 5                             Chocolate  0.938
```

```
# Find my speechiest songs
speechy <- elke_liked_tracks %>%
  arrange(desc(speechiness)) %>%
  slice(1:5)
print(speechy[,c("track.name", "speechiness")])
```

```
##                       track.name speechiness
## 1                     Dirty AF1s       0.385
## 2 WAP (feat. Megan Thee Stallion)       0.375
## 3                   October Eyes       0.367
## 4                          WOMAN       0.361
## 5      Roxanne - Remastered 2003       0.354
```

```
# Find my livest songs
liveness <- elke_liked_tracks %>%
  arrange(desc(liveness)) %>%
  slice(1:5)
print(liveness[,c("track.name", "liveness")])
```

```
##              track.name liveness
## 1 Rhiannon - Live 2005    0.985
## 2    Twenty Long Years    0.926
## 3        Dancing Queen    0.760
## 4            Landslide    0.699
## 5     The Night We Met    0.641
```

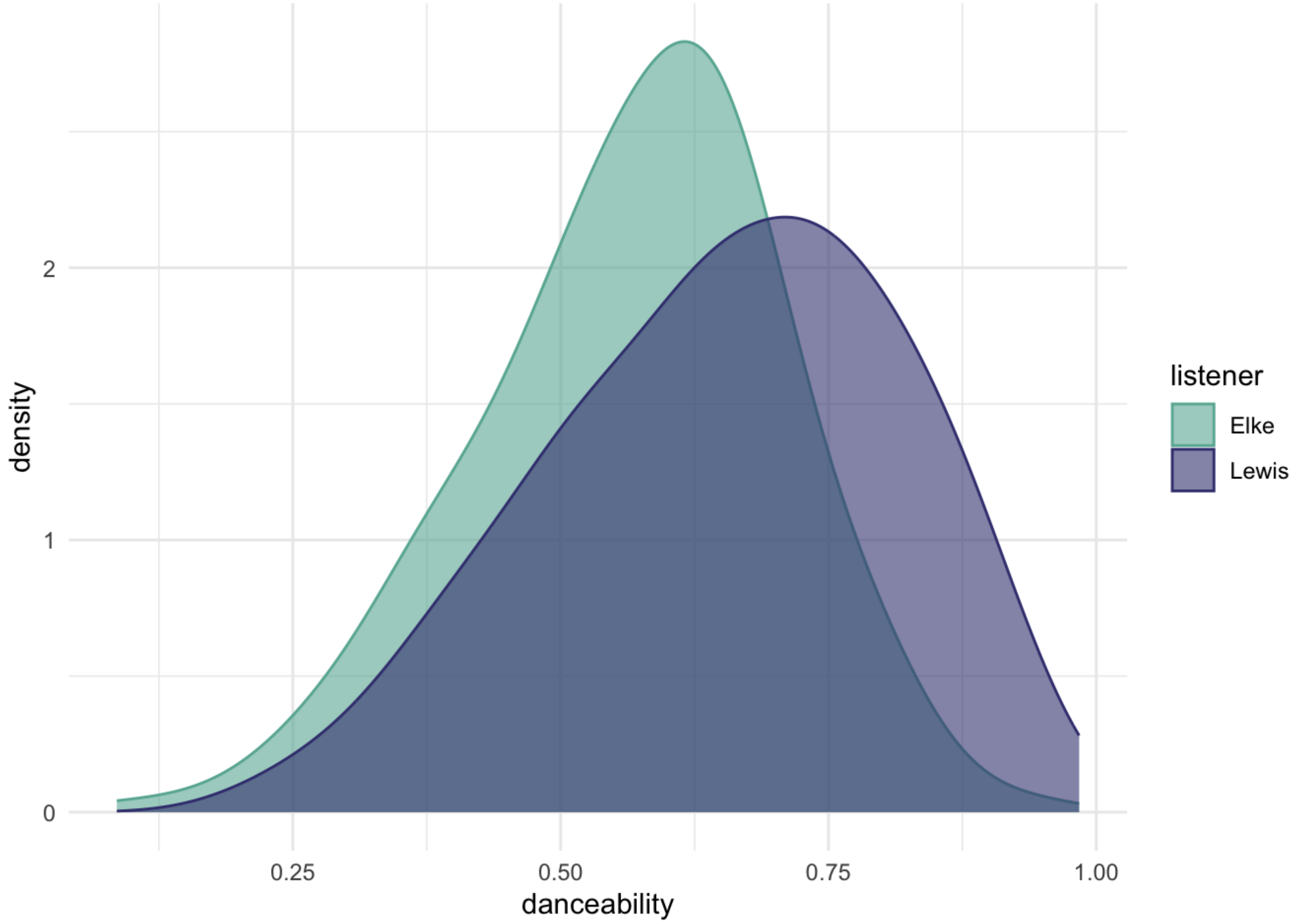
```
# Find my highest tempo songs
tempo <- elke_liked_tracks %>%
  arrange(desc(tempo)) %>%
  slice(1:5)
print(tempo[,c("track.name", "tempo")])
```

```
##                           track.name   tempo
## 1                                 FU 190.097
## 2                              10/10 187.783
## 3               Tusk - 2015 Remaster 180.837
## 4                        Unthinkable 180.068
## 5 Dusk Till Dawn (feat. Sia) - Radio Edit 180.042
```

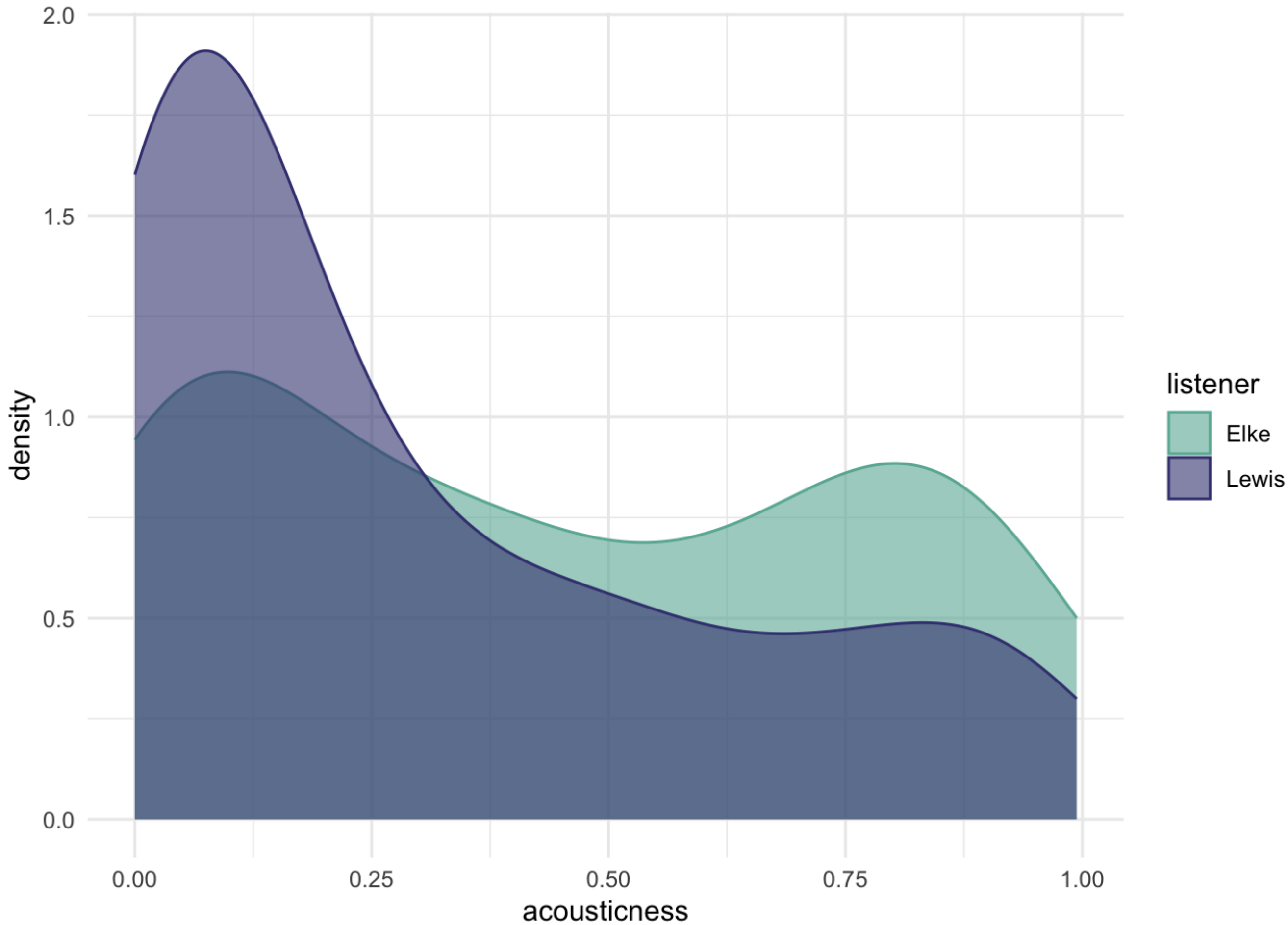
```
# Find my longest songs
length <- elke_liked_tracks %>%
  arrange(desc(duration_ms)) %>%
  slice(1:5)
print(length[,c("track.name", "duration_ms")])
```

```
##                                                                     track.name
## 1                                                                  Time's Blur
## 2 All Too Well (10 Minute Version) (Taylor's Version) (From The Vault)
## 3                                                          Rhiannon - Live 2005
## 4                                                             Show Them The Way
## 5                              This Bitter Earth / On The Nature Of Daylight
##   duration_ms
## 1      858291
## 2      613027
## 3      421800
## 4      391332
## 5      372747
```

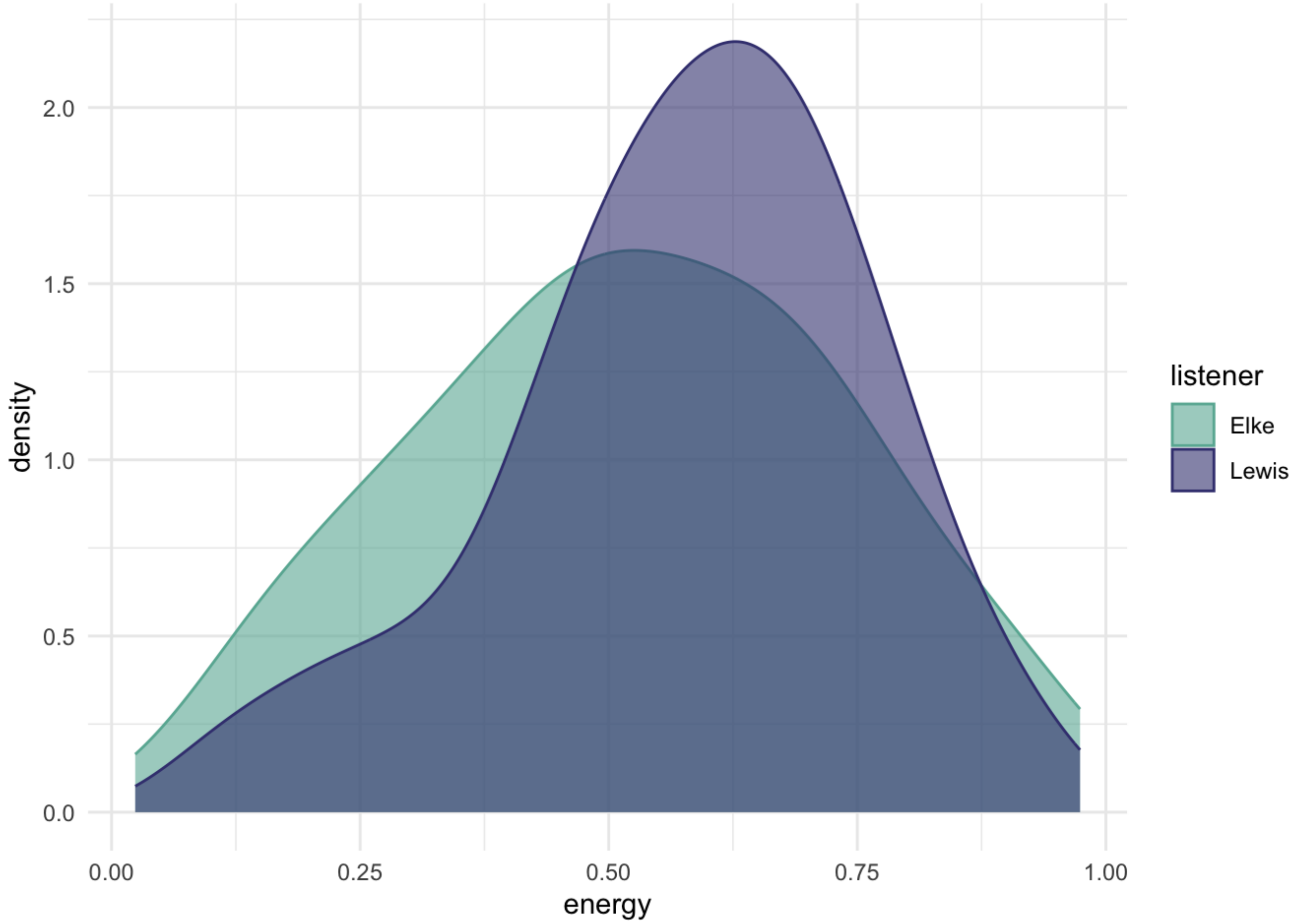
```
# Danceablility Plot
ggplot(data = all_tracks, aes(x = danceability, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```



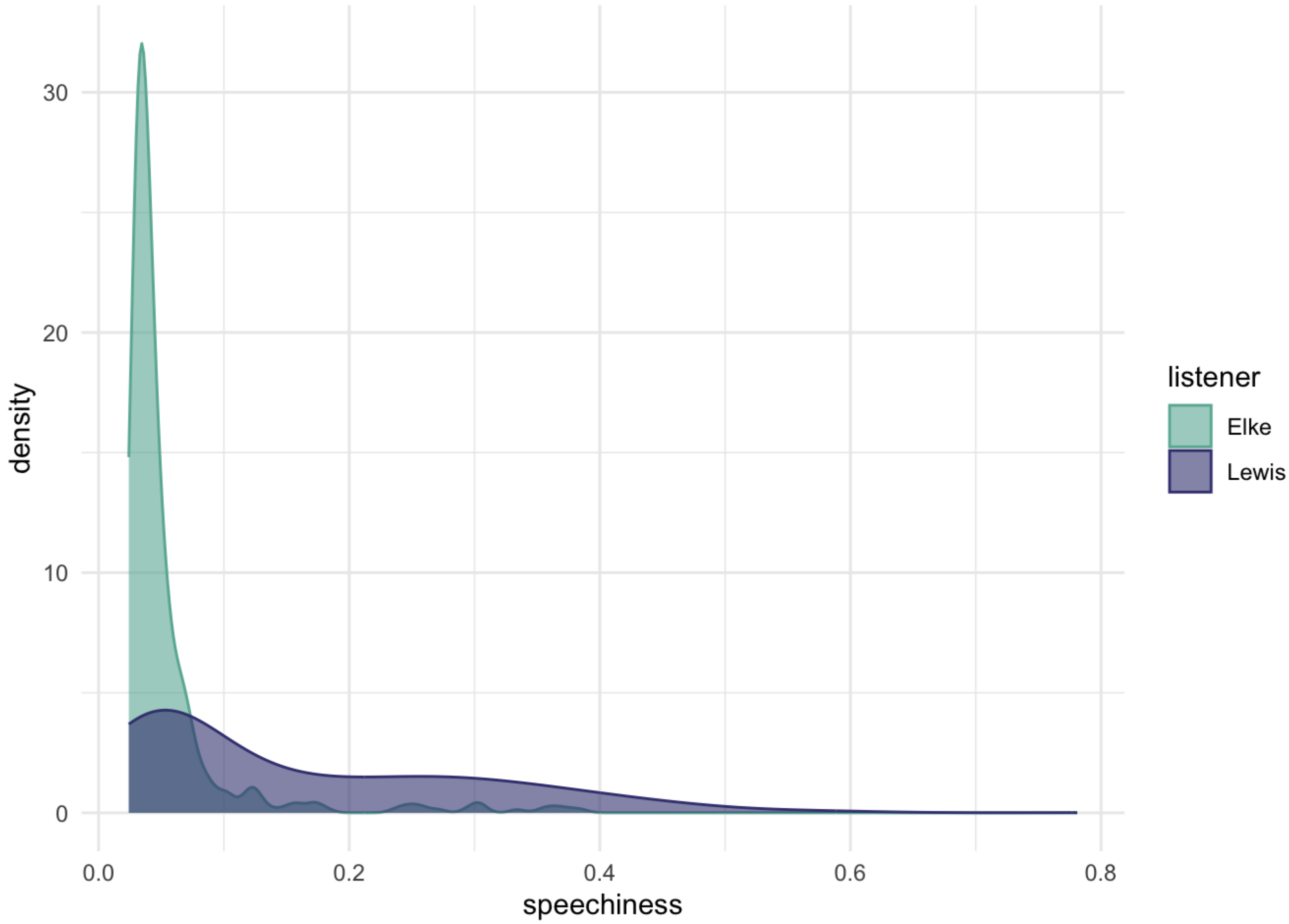

```
# Acousitcness Plot
ggplot(data = all_tracks, aes(x = acousticness, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```



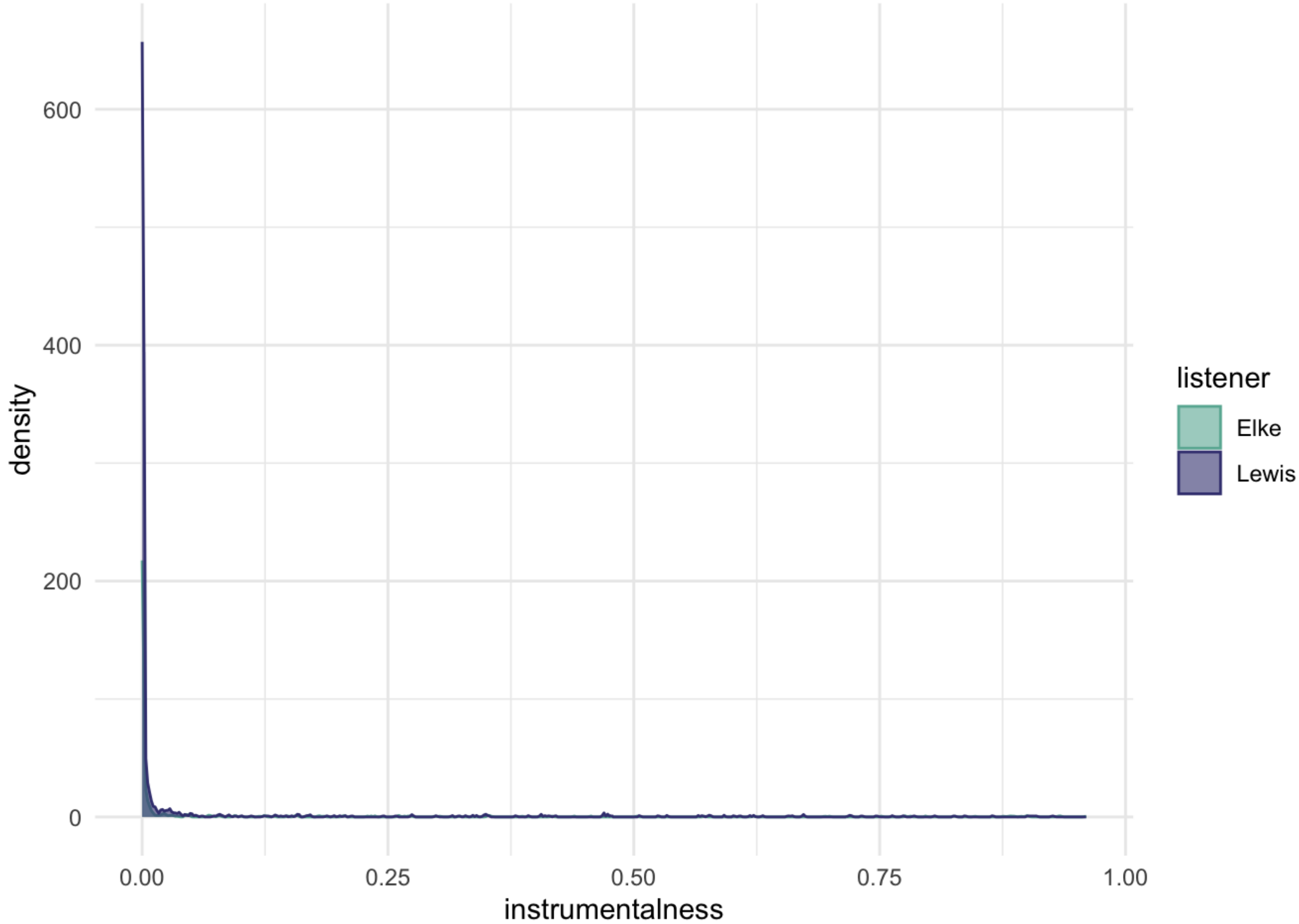

```
# Energy Plot
ggplot(data = all_tracks, aes(x = energy, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```



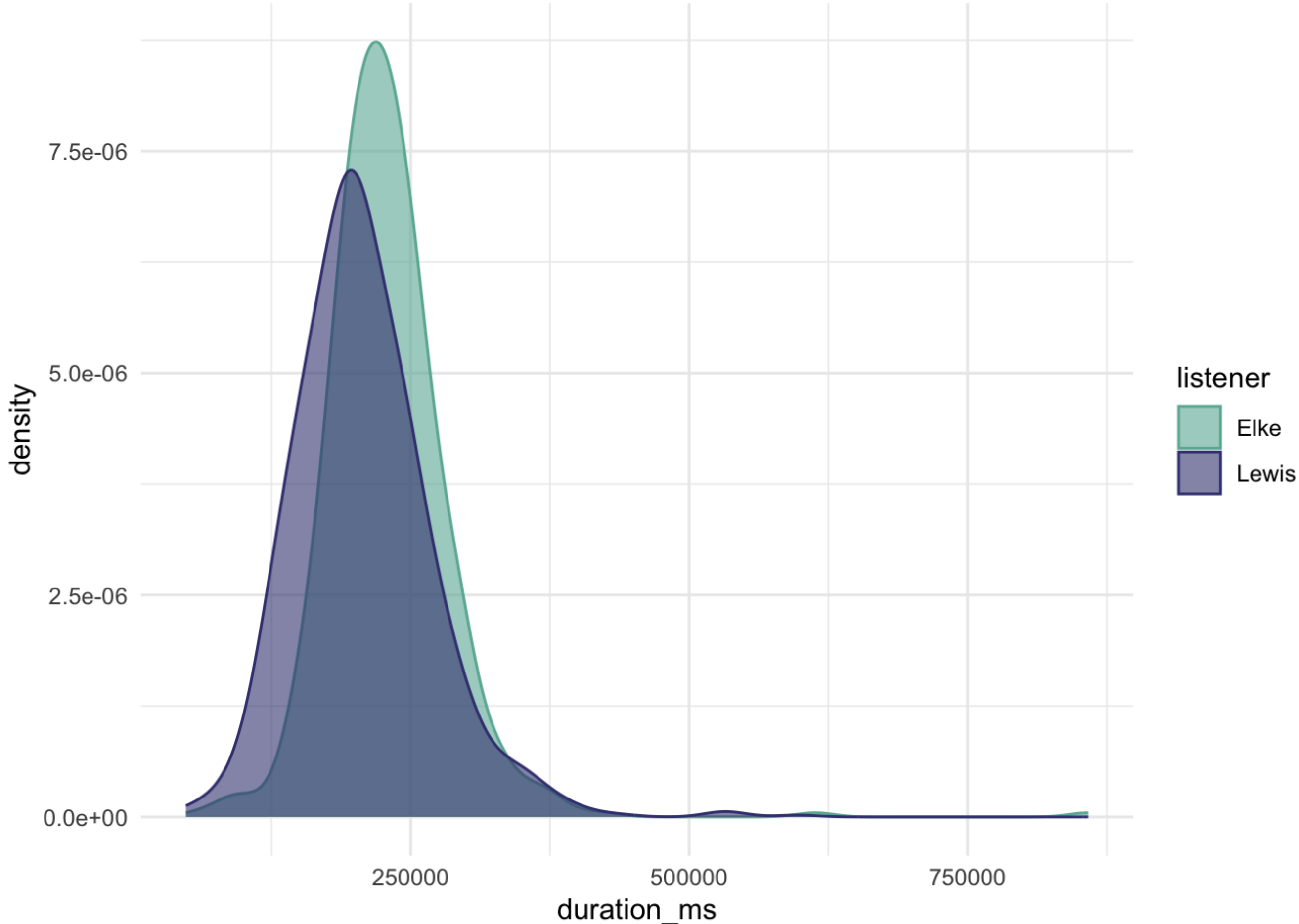

```
# Speechiness Plot
ggplot(data = all_tracks, aes(x = speechiness, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```




```
# Instrumentalness Plot
ggplot(data = all_tracks, aes(x = instrumentalness, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```




```
# Duration Plot
ggplot(data = all_tracks, aes(x = duration_ms, color = listener, fill= listener)) +
  geom_density(adjust=1.5, alpha=.6) +
  theme_minimal() +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  scale_color_manual(values = c("#69b3a2", "#404080"))
```




```
# t.test on lewis vs elke danciness
t.test(danceability ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  danceability by listener
## t = -10.711, df = 1126.5, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  -0.10232873 -0.07064212
## sample estimates:
##  mean in group Elke mean in group Lewis
##           0.5667060           0.6531914
```

```
# t.test on lewis vs elke acousticness
t.test(acousticness ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  acousticness by listener
## t = 7.4069, df = 909.35, p-value = 2.949e-13
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  0.09493404 0.16337781
## sample estimates:
##  mean in group Elke mean in group Lewis
##           0.4281137           0.2989578
```

```
# t.test on lewis vs elke energy
t.test(energy ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  energy by listener
## t = -4.8834, df = 860.26, p-value = 1.243e-06
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  -0.07615596 -0.03248973
## sample estimates:
##  mean in group Elke mean in group Lewis
##           0.5237322           0.5780550
```

```
# t.test on lewis vs elke speechiness
t.test(speechiness ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  speechiness by listener
## t = -21.402, df = 1498.9, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  -0.11260072 -0.09369343
## sample estimates:
##  mean in group Elke mean in group Lewis
##           0.0549194           0.1580665
```

```
# t.test on lewis vs elke instrumentalness
t.test(instrumentalness ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  instrumentalness by listener
## t = 1.8366, df = 801.98, p-value = 0.06664
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  -0.001109232  0.033367589
## sample estimates:
##  mean in group Elke mean in group Lewis
##          0.05274085          0.03661167
```

```
# t.test on lewis vs elke duration
t.test(duration_ms ~ listener, data = all_tracks)
```

```
##
##  Welch Two Sample t-test
##
## data:  duration_ms by listener
## t = 6.8798, df = 1031.7, p-value = 1.038e-11
## alternative hypothesis: true difference in means between group Elke and group Lewis i
s not equal to 0
## 95 percent confidence interval:
##  15735.27 28293.16
## sample estimates:
##  mean in group Elke mean in group Lewis
##            229269.5            207255.3
```

Modeling

Create four models, that predict whether a track belongs to you or your partner's collection.

Then validate and compare the performance of the two models you have created.

Make sure to use appropriate resampling to select the best version of each algorithm to compare and some appropriate visualization of your results.

Create four final candidate models:

1. k-nearest neighbor
2. decision tree
3. bagged tree
    - 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

- 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.

K-Nearest Neighbor

```
# If feature is a factor DON'T order
tracks <- all_tracks %>% mutate_if(is.ordered, .funs = factor, ordered = F) %>%
  select(-track.name) %>%
  select(-primary_artist)

tracks$listener <- as.factor(tracks$listener)
```

```
set.seed(123)
#initial split of data, default 70/30
tracks_split <- initial_split(tracks, 0.7)
tracks_test <- testing(tracks_split)
tracks_train <- training(tracks_split)
```

```
# Preprocessing
tracks_recipe <- recipe(listener ~ ., data = tracks_train) %>% # listener is outcome var
iable, use all variables
  #step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>% # normalize for knn model
  prep()

# Bake
tracks_train <- bake(tracks_recipe, tracks_train)
tracks_test <- bake(tracks_recipe, tracks_test)
```

```
set.seed(123)
# 10-fold CV on the training dataset
cv_folds <-tracks_train %>%
  vfold_cv(v=10) #10 is default
cv_folds
```

```
## #  10-fold cross-validation
## # A tibble: 10 × 2
##    splits            id
##    <list>            <chr>
##  1 <split [976/109]> Fold01
##  2 <split [976/109]> Fold02
##  3 <split [976/109]> Fold03
##  4 <split [976/109]> Fold04
##  5 <split [976/109]> Fold05
##  6 <split [977/108]> Fold06
##  7 <split [977/108]> Fold07
##  8 <split [977/108]> Fold08
##  9 <split [977/108]> Fold09
## 10 <split [977/108]> Fold10
```

```
# Define our KNN model with tuning
knn_spec_tune <- nearest_neighbor(neighbors = tune()) %>% # tune k
  set_mode("classification") %>%
  set_engine("kknn")

# Check the model
knn_spec_tune
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = tune()
##
## Computational engine: kknn
```

```
# Define a new workflow
wf_knn_tune <- workflow() %>%
  add_model(knn_spec_tune) %>%
  add_recipe(tracks_recipe)

# Fit the workflow on our predefined folds and hyperparameters
fit_knn_cv <- wf_knn_tune %>%
  tune_grid(
    cv_folds, # does tuning based on folds
    grid = data.frame(neighbors = c(1,5,seq(10,100,10)))) # K=1, K=5, K=10, K=20..., K=1
00. For each different value for k parameter, model will try it on all folds

# Check the performance with collect_metrics()
print(n = 24, fit_knn_cv %>% collect_metrics())
```

```
## # A tibble: 24 × 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy binary     0.692    10 0.0212  Preprocessor1_Model01
##  2         1 roc_auc  binary     0.655    10 0.0233  Preprocessor1_Model01
##  3         5 accuracy binary     0.712    10 0.0164  Preprocessor1_Model02
##  4         5 roc_auc  binary     0.751    10 0.0182  Preprocessor1_Model02
##  5        10 accuracy binary     0.727    10 0.0167  Preprocessor1_Model03
##  6        10 roc_auc  binary     0.767    10 0.0161  Preprocessor1_Model03
##  7        20 accuracy binary     0.740    10 0.0125  Preprocessor1_Model04
##  8        20 roc_auc  binary     0.781    10 0.0140  Preprocessor1_Model04
##  9        30 accuracy binary     0.745    10 0.0108  Preprocessor1_Model05
## 10        30 roc_auc  binary     0.785    10 0.0133  Preprocessor1_Model05
## 11        40 accuracy binary     0.736    10 0.00937 Preprocessor1_Model06
## 12        40 roc_auc  binary     0.788    10 0.0128  Preprocessor1_Model06
## 13        50 accuracy binary     0.748    10 0.00786 Preprocessor1_Model07
## 14        50 roc_auc  binary     0.790    10 0.0127  Preprocessor1_Model07
## 15        60 accuracy binary     0.749    10 0.00877 Preprocessor1_Model08
## 16        60 roc_auc  binary     0.792    10 0.0128  Preprocessor1_Model08
## 17        70 accuracy binary     0.747    10 0.00974 Preprocessor1_Model09
## 18        70 roc_auc  binary     0.792    10 0.0125  Preprocessor1_Model09
## 19        80 accuracy binary     0.748    10 0.0107  Preprocessor1_Model10
## 20        80 roc_auc  binary     0.791    10 0.0125  Preprocessor1_Model10
## 21        90 accuracy binary     0.753    10 0.0116  Preprocessor1_Model11
## 22        90 roc_auc  binary     0.790    10 0.0121  Preprocessor1_Model11
## 23       100 accuracy binary     0.747    10 0.0108  Preprocessor1_Model12
## 24       100 roc_auc  binary     0.790    10 0.0118  Preprocessor1_Model12
```

```
# The final workflow for our KNN model
final_wf <-
  wf_knn_tune %>%
  finalize_workflow(select_best(fit_knn_cv))
```

```
## Warning: No value of `metric` was given; metric 'roc_auc' will be used.
```

```
# Check out the final workflow object
final_wf
```

```
## ══ Workflow ════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## ── Preprocessor ────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_normalize()
##
## ── Model ───────────────────────────────────────────────────────────────
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = 70
##
## Computational engine: kknn
```

```
# Fitting our final workflow
final_fit <- final_wf %>%
  fit(data = tracks_train)
# Examine the final workflow
final_fit
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## ── Preprocessor ────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_normalize()
##
## ── Model ───────────────────────────────────────────────────────────────
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(70,     data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.2543779
## Best kernel: optimal
## Best k: 70
```

```
# Fit the model to the test data
tracks_pred <- predict(final_fit, new_data = tracks_test)
# Bind to track dataframe
tracks_final <- cbind(tracks_test, tracks_pred)
# Build a confusion matrix
con_matrix <- tracks_final %>%
  select(listener, .pred_class) %>%
  table()

# print table
con_matrix
```

```
##         .pred_class
## listener Elke Lewis
##    Elke    77    61
##    Lewis   53   274
```

```
# Calculate dummy classifier
dummy <- nrow(lewis_liked_tracks) / (nrow(lewis_liked_tracks) + nrow(elke_liked_tracks))
print(dummy)
```

```
## [1] 0.6774194
```

```
# Write over 'final_fit' with this last_fit() approach
final_fit <- final_wf %>% last_fit(tracks_split)
# Collect metrics on the test data!
tibble <- final_fit %>% collect_metrics()
tibble
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.755 Preprocessor1_Model1
## 2 roc_auc  binary         0.802 Preprocessor1_Model1
```

```
final_accuracy <- tibble %>%
  filter(.metric == "accuracy") %>%
  pull(.estimate)

print(paste0("We see here that our k-nearest neighbors model had a higher accuracy at pr
edicting listener than the dummy classifier. The accuracy of the model was ", round(fina
l_accuracy, 3), " and the dummy classifier accuracy was ", round(dummy, 3), "."))
```

```
## [1] "We see here that our k-nearest neighbors model had a higher accuracy at predicti
ng listener than the dummy classifier. The accuracy of the model was 0.755 and the dummy
classifier accuracy was 0.677."
```

We see here that our k-nearest neighbors model had a higher accuracy at predicting listener than the dummy classifier.

Decision Tree

```
#Preprocess the data
listener_rec <- recipe(listener~., data = tracks_train) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

```
#Tell the model that we are tuning hyperparams
tree_spec_tune <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification")

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

tree_grid
```

```
## # A tibble: 125 × 3
##    cost_complexity tree_depth min_n
##              <dbl>      <int> <int>
##  1    0.0000000001          1     2
##  2    0.0000000178          1     2
##  3    0.00000316            1     2
##  4    0.000562              1     2
##  5    0.1                   1     2
##  6    0.0000000001          4     2
##  7    0.0000000178          4     2
##  8    0.00000316            4     2
##  9    0.000562              4     2
## 10    0.1                   4     2
## # … with 115 more rows
```

```
wf_tree_tune <- workflow() %>%
  add_recipe(listener_rec) %>%
  add_model(tree_spec_tune)
```

```
#set up k-fold cv. This can be used for all the algorithms
listener_cv = tracks_train %>%
  vfold_cv(v = 5)
listener_cv
```

```
## #  5-fold cross-validation
## # A tibble: 5 × 2
##   splits            id
##   <list>            <chr>
## 1 <split [868/217]> Fold1
## 2 <split [868/217]> Fold2
## 3 <split [868/217]> Fold3
## 4 <split [868/217]> Fold4
## 5 <split [868/217]> Fold5
```

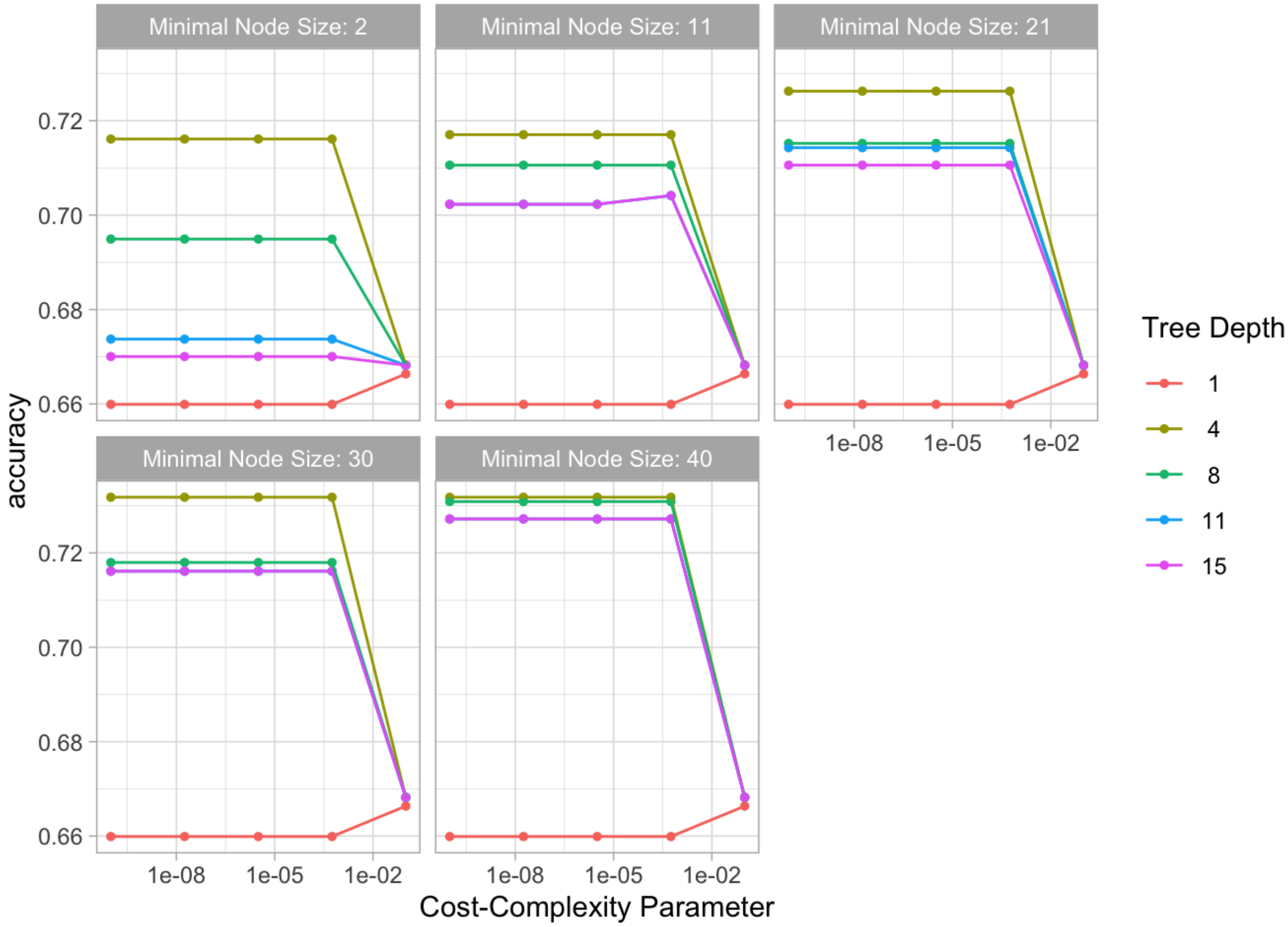
```
doParallel::registerDoParallel() #build trees in parallel
#200s
tree_rs <- tune_grid(
  wf_tree_tune,
  listener~.,
  resamples = listener_cv,
  grid = tree_grid,
  metrics = metric_set(accuracy)
)
```

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

```
tree_rs
```

```
## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 × 4
##   splits            id    .metrics           .notes
##   <list>            <chr> <list>             <list>
## 1 <split [868/217]> Fold1 <tibble [125 × 7]> <tibble [0 × 3]>
## 2 <split [868/217]> Fold2 <tibble [125 × 7]> <tibble [0 × 3]>
## 3 <split [868/217]> Fold3 <tibble [125 × 7]> <tibble [0 × 3]>
## 4 <split [868/217]> Fold4 <tibble [125 × 7]> <tibble [0 × 3]>
## 5 <split [868/217]> Fold5 <tibble [125 × 7]> <tibble [0 × 3]>
```

```
#Use autoplot() to examine how different parameter configurations relate to accuracy
autoplot(tree_rs) + theme_light()
```




```
# select best hyperparameterw
show_best(tree_rs)
```

```
## # A tibble: 5 × 9
##   cost_complexity tree_depth min_n .metric  .estim…¹  mean     n std_err .config
##             <dbl>      <int> <int> <chr>    <chr>    <dbl> <int>   <dbl> <chr>
## 1    0.0000000001          4    30 accuracy binary   0.732     5  0.0147 Prepro…
## 2    0.0000000178          4    30 accuracy binary   0.732     5  0.0147 Prepro…
## 3    0.00000316            4    30 accuracy binary   0.732     5  0.0147 Prepro…
## 4    0.000562              4    30 accuracy binary   0.732     5  0.0147 Prepro…
## 5    0.0000000001          4    40 accuracy binary   0.732     5  0.0152 Prepro…
## # … with abbreviated variable name ¹.estimator
```

```
select_best(tree_rs)
```

```
## # A tibble: 1 × 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1    0.0000000001          4    30 Preprocessor1_Model081
```

```
final_tree <- finalize_model(tree_spec_tune, select_best(tree_rs))
```

```
final_tree_fit <- last_fit(final_tree, listener~., tracks_split) # does training fit the
n final prediction as well
final_tree_fit$.predictions
```

```
## [[1]]
## # A tibble: 465 × 6
##    .pred_Elke .pred_Lewis  .row .pred_class listener .config
##         <dbl>       <dbl> <int> <fct>       <fct>    <chr>
##  1     0.0577       0.942     3 Lewis       Lewis    Preprocessor1_Model1
##  2     0.643        0.357     7 Elke        Lewis    Preprocessor1_Model1
##  3     0.643        0.357    12 Elke        Lewis    Preprocessor1_Model1
##  4     0.25         0.75     14 Lewis       Lewis    Preprocessor1_Model1
##  5     0.643        0.357    15 Elke        Lewis    Preprocessor1_Model1
##  6     0.0577       0.942    20 Lewis       Lewis    Preprocessor1_Model1
##  7     0.643        0.357    21 Elke        Lewis    Preprocessor1_Model1
##  8     0.538        0.462    22 Elke        Lewis    Preprocessor1_Model1
##  9     0.0577       0.942    23 Lewis       Lewis    Preprocessor1_Model1
## 10     0.538        0.462    27 Elke        Lewis    Preprocessor1_Model1
## # … with 455 more rows
```

```
final_tree_fit$.metrics
```

```
## [[1]]
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.673 Preprocessor1_Model1
## 2 roc_auc  binary         0.751 Preprocessor1_Model1
```

```
tibble_tree <- final_tree_fit %>% collect_metrics()
tibble_tree
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.673 Preprocessor1_Model1
## 2 roc_auc  binary         0.751 Preprocessor1_Model1
```

```
final_tree_accuracy <- tibble_tree %>%
  filter(.metric == "accuracy") %>%
  pull(.estimate)

print(paste0("We see here that our decision tree model had a lower accuracy at predictin
g listener than the dummy classifier or the k-nearest neighbor model. The accuracy of th
e decision tree was ", round(final_tree_accuracy, 3), "."))
```

```
## [1] "We see here that our decision tree model had a lower accuracy at predicting list
ener than the dummy classifier or the k-nearest neighbor model. The accuracy of the deci
sion tree was 0.673."
```

Bagging

```
set.seed(123)
# Bagging specifications
bag_spec <-
  bag_tree(cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()) %>%
  set_engine("rpart", times = 75) %>% # 25 ensemble members
  set_mode("classification")

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

bag_grid
```

```
## # A tibble: 125 × 3
##    cost_complexity tree_depth min_n
##              <dbl>      <int> <int>
##  1    0.0000000001          1     2
##  2    0.0000000178          1     2
##  3    0.00000316            1     2
##  4    0.000562              1     2
##  5    0.1                   1     2
##  6    0.0000000001          4     2
##  7    0.0000000178          4     2
##  8    0.00000316            4     2
##  9    0.000562              4     2
## 10    0.1                   4     2
## # … with 115 more rows
```

```
wf_bag <- workflow() %>%
  add_recipe(listener_rec) %>%
  add_model(bag_spec)
```

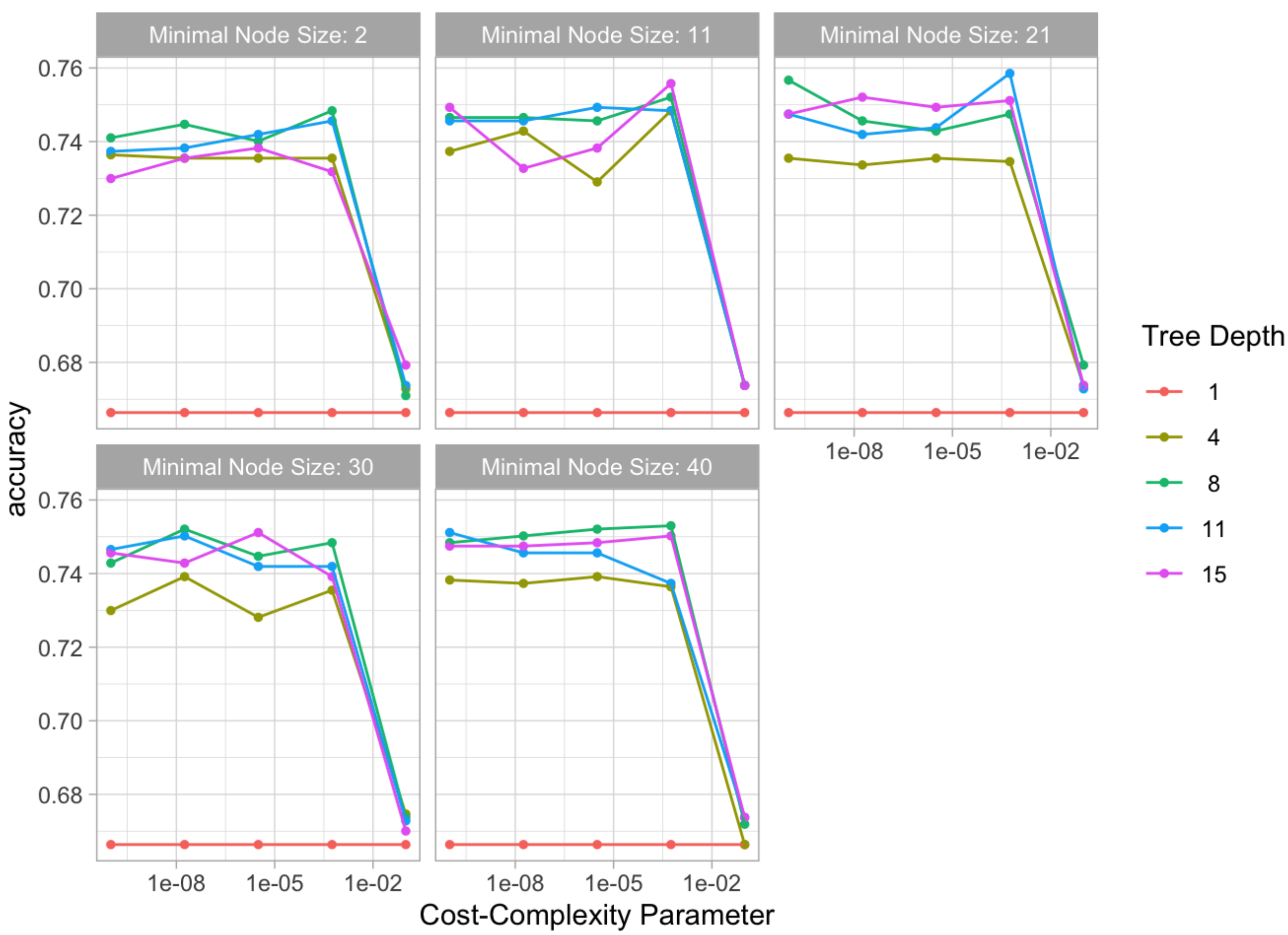
```
doParallel::registerDoParallel() #build trees in parallel

bag_rs <- tune_grid(
  wf_bag,
  listener~.,
  resamples = listener_cv,
  grid = bag_grid,
  metrics = metric_set(accuracy)
)
```

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

```
bag_rs
```

```
## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 × 4
##   splits            id    .metrics           .notes
##   <list>            <chr> <list>             <list>
## 1 <split [868/217]> Fold1 <tibble [125 × 7]> <tibble [0 × 3]>
## 2 <split [868/217]> Fold2 <tibble [125 × 7]> <tibble [0 × 3]>
## 3 <split [868/217]> Fold3 <tibble [125 × 7]> <tibble [0 × 3]>
## 4 <split [868/217]> Fold4 <tibble [125 × 7]> <tibble [0 × 3]>
## 5 <split [868/217]> Fold5 <tibble [125 × 7]> <tibble [0 × 3]>
```

```
# Use autoplot() to examine how different parameter configurations relate to accuracy
autoplot(bag_rs) + theme_light()
```




```
# Select hyperparameters
show_best(bag_rs)
```

```
## # A tibble: 5 × 9
##   cost_complexity tree_depth min_n .metric  .estim…¹  mean     n std_err .config
##             <dbl>      <int> <int> <chr>    <chr>    <dbl> <int>   <dbl> <chr>
## 1    0.000562             11    21 accuracy binary   0.759     5  0.0178 Prepro…
## 2    0.0000000001          8    21 accuracy binary   0.757     5  0.0190 Prepro…
## 3    0.000562             15    11 accuracy binary   0.756     5  0.0198 Prepro…
## 4    0.000562              8    40 accuracy binary   0.753     5  0.0219 Prepro…
## 5    0.000562              8    11 accuracy binary   0.752     5  0.0210 Prepro…
## # … with abbreviated variable name ¹.estimator
```

```
select_best(bag_rs)
```

```
## # A tibble: 1 × 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1        0.000562         11    21 Preprocessor1_Model069
```

```
final_bag <- finalize_model(bag_spec, select_best(bag_rs))
```

```
final_bag_fit <- last_fit(final_bag, listener~., tracks_split) # does training fit then final prediction as well
final_bag_fit$.predictions
```

```
## [[1]]
## # A tibble: 465 × 6
##    .pred_Elke .pred_Lewis  .row .pred_class listener .config
##         <dbl>       <dbl> <int> <fct>       <fct>    <chr>
##  1     0.0921       0.908     3 Lewis       Lewis    Preprocessor1_Model1
##  2     0.623        0.377     7 Elke        Lewis    Preprocessor1_Model1
##  3     0.647        0.353    12 Elke        Lewis    Preprocessor1_Model1
##  4     0.373        0.627    14 Lewis       Lewis    Preprocessor1_Model1
##  5     0.674        0.326    15 Elke        Lewis    Preprocessor1_Model1
##  6     0.0464       0.954    20 Lewis       Lewis    Preprocessor1_Model1
##  7     0.577        0.423    21 Elke        Lewis    Preprocessor1_Model1
##  8     0.370        0.630    22 Lewis       Lewis    Preprocessor1_Model1
##  9     0.0453       0.955    23 Lewis       Lewis    Preprocessor1_Model1
## 10     0.612        0.388    27 Elke        Lewis    Preprocessor1_Model1
## # … with 455 more rows
```

```
final_bag_fit$.metrics
```

```
## [[1]]
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.748 Preprocessor1_Model1
## 2 roc_auc  binary         0.812 Preprocessor1_Model1
```

```
tibble_bag <- final_bag_fit %>% collect_metrics()
tibble_bag
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.748 Preprocessor1_Model1
## 2 roc_auc  binary         0.812 Preprocessor1_Model1
```

```
final_bag_accuracy <- tibble_bag %>%
  filter(.metric == "accuracy") %>%
  pull(.estimate)

print(paste0("We see here that our bagging model had a higher accuracy at predicting lis
tener than the decision tree or dummy classifier. The accuracy of the bagging was ", rou
nd(final_bag_accuracy, 3), ". This is still lower than the knn model."))
```

```
## [1] "We see here that our bagging model had a higher accuracy at predicting listener
than the decision tree or dummy classifier. The accuracy of the bagging was 0.748. This
is still lower than the knn model."
```

Random Forest

```
set.seed(123)
# Bagging specifications
forest_spec <-
  rand_forest(min_n = tune(),
  mtry = tune(),
  trees = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

forest_grid <- grid_regular(min_n(), mtry(c(1,13)), trees(), levels = 5)

forest_grid
```

```
## # A tibble: 125 × 3
##    min_n  mtry trees
##    <int> <int> <int>
##  1     2     1     1
##  2    11     1     1
##  3    21     1     1
##  4    30     1     1
##  5    40     1     1
##  6     2     4     1
##  7    11     4     1
##  8    21     4     1
##  9    30     4     1
## 10    40     4     1
## # … with 115 more rows
```

```
wf_forest <- workflow() %>%
  add_recipe(listener_rec) %>%
  add_model(forest_spec)
```

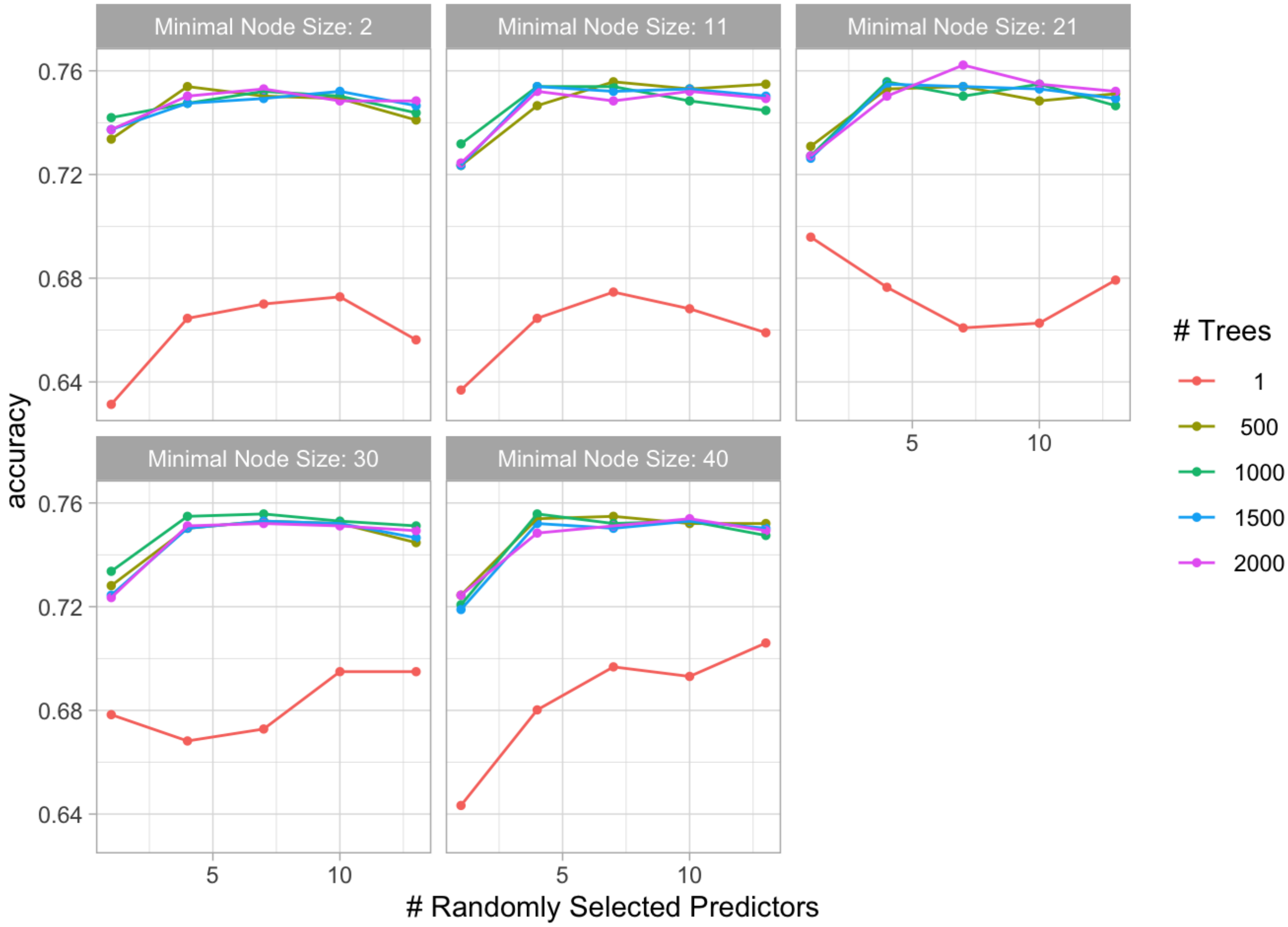
```
doParallel::registerDoParallel() #build trees in parallel

forest_rs <- tune_grid(
  wf_forest,
  listener~.,
  resamples = listener_cv,
  grid = forest_grid,
  metrics = metric_set(accuracy)
)
```

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

```
forest_rs
```

```
## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 × 4
##   splits            id    .metrics           .notes
##   <list>            <chr> <list>             <list>
## 1 <split [868/217]> Fold1 <tibble [125 × 7]> <tibble [0 × 3]>
## 2 <split [868/217]> Fold2 <tibble [125 × 7]> <tibble [0 × 3]>
## 3 <split [868/217]> Fold3 <tibble [125 × 7]> <tibble [0 × 3]>
## 4 <split [868/217]> Fold4 <tibble [125 × 7]> <tibble [0 × 3]>
## 5 <split [868/217]> Fold5 <tibble [125 × 7]> <tibble [0 × 3]>
```

```
# Use autoplot() to examine how different parameter configurations relate to accuracy
autoplot(forest_rs) + theme_light()
```




```
# Select hyperparameters
show_best(forest_rs)
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric  .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     7  2000    21 accuracy binary     0.762     5  0.0175 Preprocessor1_Model…
## 2     7   500    11 accuracy binary     0.756     5  0.0144 Preprocessor1_Model…
## 3     4  1000    21 accuracy binary     0.756     5  0.0193 Preprocessor1_Model…
## 4     4  1000    40 accuracy binary     0.756     5  0.0162 Preprocessor1_Model…
## 5     7  1000    30 accuracy binary     0.756     5  0.0180 Preprocessor1_Model…
```

```
select_best(forest_rs)
```

```
## # A tibble: 1 × 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     7  2000    21 Preprocessor1_Model113
```

```
final_forest <- finalize_model(forest_spec, select_best(forest_rs))
```

```
final_forest_fit <- last_fit(final_forest, listener~., tracks_split) # does training fit
then final prediction as well
final_forest_fit$.predictions
```

```
## [[1]]
## # A tibble: 465 × 6
##    .pred_Elke .pred_Lewis  .row .pred_class listener .config
##         <dbl>       <dbl> <int> <fct>       <fct>    <chr>
##  1     0.114        0.886     3 Lewis       Lewis    Preprocessor1_Model1
##  2     0.605        0.395     7 Elke        Lewis    Preprocessor1_Model1
##  3     0.663        0.337    12 Elke        Lewis    Preprocessor1_Model1
##  4     0.372        0.628    14 Lewis       Lewis    Preprocessor1_Model1
##  5     0.634        0.366    15 Elke        Lewis    Preprocessor1_Model1
##  6     0.0303       0.970    20 Lewis       Lewis    Preprocessor1_Model1
##  7     0.530        0.470    21 Elke        Lewis    Preprocessor1_Model1
##  8     0.339        0.661    22 Lewis       Lewis    Preprocessor1_Model1
##  9     0.0665       0.934    23 Lewis       Lewis    Preprocessor1_Model1
## 10     0.634        0.366    27 Elke        Lewis    Preprocessor1_Model1
## # … with 455 more rows
```

```
final_forest_fit$.metrics
```

```
## [[1]]
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.757 Preprocessor1_Model1
## 2 roc_auc  binary         0.829 Preprocessor1_Model1
```

```
tibble_forest <- final_forest_fit %>% collect_metrics()
tibble_forest
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.757 Preprocessor1_Model1
## 2 roc_auc  binary         0.829 Preprocessor1_Model1
```

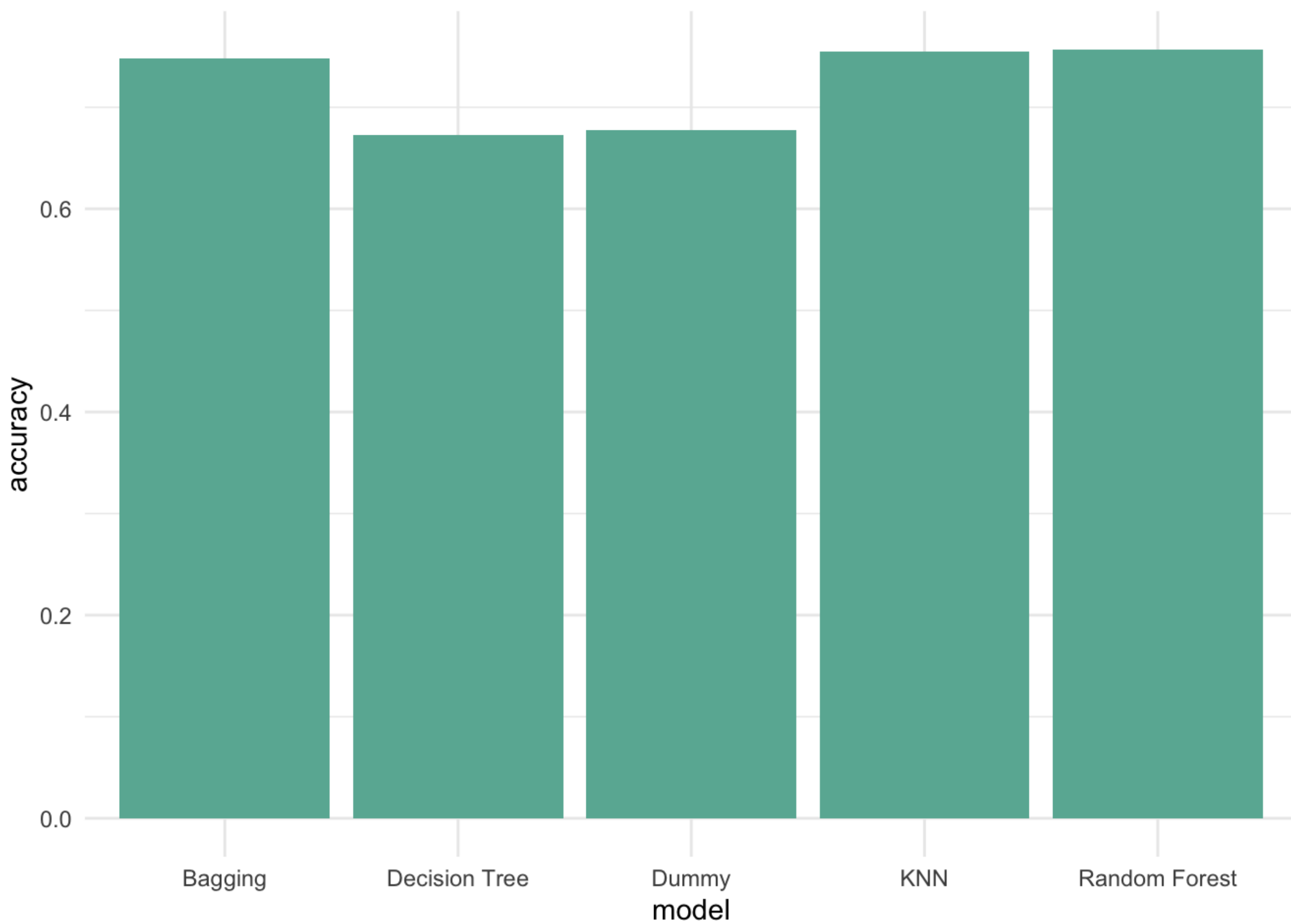
```
final_forest_accuracy <- tibble_forest %>%
  filter(.metric == "accuracy") %>%
  pull(.estimate)

print(paste0("We see here that our random forest had the highest accuracy at predicting
listener than the other models. The accuracy of the forest was ", round(final_forest_acc
uracy, 3), "."))
```

```
## [1] "We see here that our random forest had the highest accuracy at predicting listen
er than the other models. The accuracy of the forest was 0.757."
```

```
model <- c("Dummy", "KNN", "Decision Tree", "Bagging", "Random Forest")
accuracy <- c(dummy, final_accuracy, final_tree_accuracy, final_bag_accuracy, final_fore
st_accuracy)

accuracy_df <- data.frame(model, accuracy)
print(accuracy_df)
```

```
##           model  accuracy
## 1         Dummy 0.6774194
## 2           KNN 0.7548387
## 3 Decision Tree 0.6731183
## 4       Bagging 0.7483871
## 5 Random Forest 0.7569892
```

```
ggplot(accuracy_df, aes(x = model, y = accuracy)) +
  geom_col(fill = "#69b3a2") +
  theme_minimal()
```



From this lab, we can see that using bagging and a random forest greatly improves the accuracy of a decision tree. However, this came with the tradeoff of computation time. It took around 30 minutes to build trees in parallel, whereas it only took a few seconds to make my single decision tree. The k-nearest neighbor algorithm also worked very well here and had essentially the same accuracy as the random forest.