Guillermo Romero

```r
library(spotifyr) #API interaction
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   1.0.1
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.5.0
v readr   2.1.3      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```r
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.0.0 --
v broom        1.0.2     v rsample      1.1.1
v dials        1.1.0     v tune         1.0.1
v infer        1.0.4     v workflows    1.1.2
v modeldata    1.0.1     v workflowsets 1.0.0
v parsnip      1.0.3     v yardstick    1.1.0
v recipes      1.0.4
-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
* Search for functions across packages at https://www.tidymodels.org/find/
```

```r
library(readr)
library(skimr)
library(ggpubr)
library(patchwork)
library(caret)
```

```
Loading required package: lattice

Attaching package: 'caret'

The following objects are masked from 'package:yardstick':

    precision, recall, sensitivity, specificity

The following object is masked from 'package:purrr':

    lift
```

```r
library(corrplot)
```

```
corrplot 0.92 loaded
```

```r
library(flextable)
```

```
Attaching package: 'flextable'

The following objects are masked from 'package:ggpubr':

    border, font, rotate

The following object is masked from 'package:purrr':

    compose
```

```r
library(baguette)
library(ranger)
library(ggplot2)
```

```r
library(vip)
```

```
Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

Data Wrangling

```r
Sys.setenv(SPOTIFY_CLIENT_ID = '22120fa7a3ee4f6dbb73fe9378000b6a')

Sys.setenv(SPOTIFY_CLIENT_SECRET = '32cf0124f63f4b9699c489ba57261666')
```

```r
access_token <-get_spotify_access_token(
  client_id ="22120fa7a3ee4f6dbb73fe9378000b6a",
  client_secret = "32cf0124f63f4b9699c489ba57261666" )
```

```r
get_all_tracks <- function(access_token, limit = 50, offset = 0, market = 'US') {
  tracks <- data.frame()

  repeat {
    response_data <- get_my_saved_tracks(limit = limit, offset = offset, market = market)
    tracks <- bind_rows(tracks,response_data)

    if (offset + limit >= 350) {
      break
    }
    offset <- offset + limit
  }

  tracks
}
```

```r
all_tracks <- get_all_tracks(access_token)
```

```r
feature1 <-  get_track_audio_features(all_tracks$track.id[1:100])
feature2 <- get_track_audio_features(all_tracks$track.id[101:200])
feature3 <- get_track_audio_features(all_tracks$track.id[201:300])
feature4 <- get_track_audio_features(all_tracks$track.id[301:328])

track_features <- bind_rows(feature1,feature2, feature3, feature4) |>
  bind_cols(track_name = all_tracks$track.name)

#write_csv(track_features, 'track_features.csv')
```

```r
# 0 = Guillermo
#1 = Dalila
features_0 <- track_features |>
  mutate(data_owned_by = as.factor(0)) |>
  select(-type, -id,-uri,-track_href,-analysis_url) |>
  slice(1:200)


dalilas_tracks <- read.csv('dalila_spotify.csv') |>
  mutate(data_owned_by = as.factor(1)) |>
  select(-X,-type, -id,-uri,-track_href,-analysis_url)

tracks <- features_0 |>
  bind_rows(dalilas_tracks)
```

Data Exploration

```r
names(tracks)
```

```
 [1] "danceability"     "energy"           "key"              "loudness"
 [5] "mode"             "speechiness"      "acousticness"     "instrumentalness"
 [9] "liveness"         "valence"          "tempo"            "duration_ms"
[13] "time_signature"   "track_name"       "data_owned_by"
```

```r
skimr::skim(tracks)
```

Table 1: Data summary

Name	tracks
Number of rows	400
Number of columns	15
Column type frequency:	
character	1
factor	1
numeric	13
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
track_name	0	1	2	102	0	393	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
data_owned_by	0	1	FALSE	2	0: 200, 1: 200

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
danceability	0	1	0.59	0.17	0.17	0.47	0.59	0.72	0.98	
energy	0	1	0.61	0.22	0.05	0.46	0.59	0.78	1.00	
key	0	1	5.83	3.55	0.00	2.75	7.00	9.00	11.00	
loudness	0	1	-7.45	3.13	-17.30	-9.33	-7.28	-5.14	0.84	
mode	0	1	0.69	0.46	0.00	0.00	1.00	1.00	1.00	
speechiness	0	1	0.11	0.12	0.02	0.04	0.06	0.13	0.77	
acousticness	0	1	0.39	0.31	0.00	0.08	0.35	0.67	0.99	
instrumentalness	0	1	0.03	0.13	0.00	0.00	0.00	0.00	0.96	
liveness	0	1	0.23	0.17	0.03	0.11	0.16	0.30	0.89	
valence	0	1	0.59	0.25	0.04	0.42	0.59	0.80	0.98	
tempo	0	1	121.33	31.83	65.24	95.34	116.96	143.37	202.10	
duration_ms	0	1	203203.86	62177.06	58200.00	158103.50	193497.00	235171.50	579293.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
time_signature	0	1	3.79	0.51	1.00	4.00	4.00	4.00	5.00	

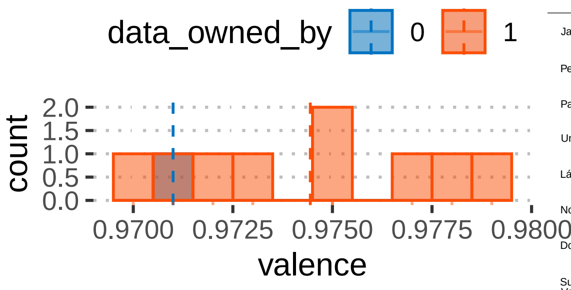
Top Ten

```
# 0 = Guillermo
#1 = Dalila
danceability_top_ten <-  tracks |>
  select(track_name, danceability, data_owned_by) |>
  slice_max(n = 10,order_by = danceability)

energy_top_ten <- tracks |>
  select(track_name, energy, data_owned_by) |>
  slice_max(n = 10,order_by = energy)

valence_top_ten <- tracks |>
  select(track_name, valence, data_owned_by) |>
  slice_max(n = 10,order_by = valence)

instrument_top_ten <- tracks |>
  select(track_name, instrumentalness, data_owned_by) |>
  slice_max(n = 10,order_by = instrumentalness)
```

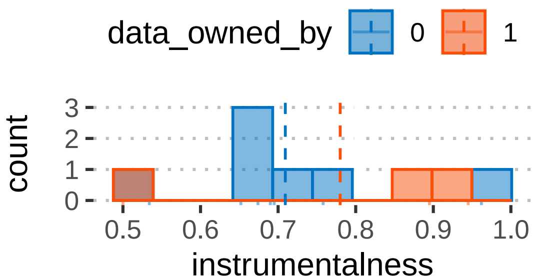
```
# 0 = Guillermo
#1 = Dalila

danceability_10 <- danceability_top_ten  |>
  gghistogram( x = "danceability", bins = 10,
   add = "mean", rug = TRUE,
   color = "data_owned_by", fill = "data_owned_by",
   palette = c("#0073C2FF", "#FC4E07"))+
  theme_pubclean()

energy_10 <- energy_top_ten  |>
  gghistogram( x = "energy", bins = 10,
   add = "mean", rug = TRUE,
   color = "data_owned_by", fill = "data_owned_by",
   palette = c("#0073C2FF", "#FC4E07"))+
  theme_pubclean()
```

```
valence_10 <- valence_top_ten  |>
  gghistogram( x = "valence", bins = 10,
   add = "mean", rug = TRUE,
   color = "data_owned_by", fill = "data_owned_by",
   palette = c("#0073C2FF", "#FC4E07"))+
  theme_pubclean()

instrument_10 <- instrument_top_ten  |>
  gghistogram( x = "instrumentalness", bins = 10,
   add = "mean", rug = TRUE,
   color = "data_owned_by", fill = "data_owned_by",
   palette = c("#0073C2FF", "#FC4E07"))+
  theme_pubclean()

d10 <- flextable(danceability_top_ten)
e10 <- flextable(energy_top_ten)
v10 <- flextable(valence_top_ten)
i10 <- flextable(instrument_top_ten)
```

```
((valence_10 + gen_grob(v10, fit = "width", just = "top", scaling = 'full'))
/
(instrument_10 + gen_grob(i10, fit = "width", just = "top", scaling = 'full'))  )
```

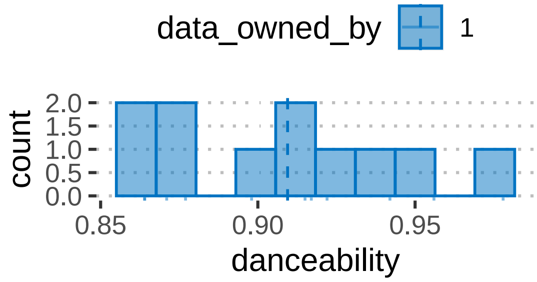


track_name	valence	data_owned_by
Jambalaya	0.979	1
Perro Bichi Y Mujeriego	0.978	1
Paz en Este Amor	0.977	1
Una Aventura	0.975	1
Lágrimas De Cristal	0.975	1
No Mencionan	0.973	1
Dos Hojas Sin Rumbo	0.972	1
Sugar Dumpling (Original Version)	0.971	0

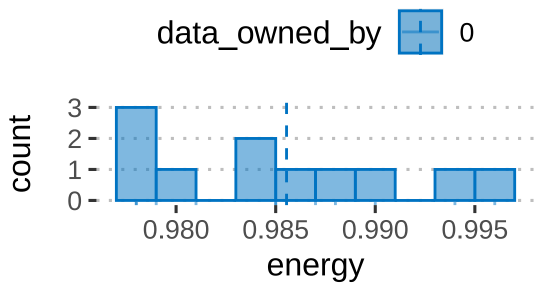


track_name	instrumentalness	data_owned_by
Biochemical Equation – Instrumental	0.962	0
Enamorado	0.945	1
Mr. Rager	0.895	1
Desperate	0.758	0
No Guilt	0.695	0
List & Heel	0.690	0
No Man Born Evil	0.674	0

```
((danceability_10 + gen_grob(d10, fit = 'width', just = "top", scaling = 'full')) /
(energy_10 + gen_grob(e10, fit = "width", just = "top", scaling = 'full')) )
```



track_name	danceability	data_owned_by
Uno	0.978	1
Whole Lotta Choppas (Remix) [feat. Nicki Minaj]	0.956	1
Un Fin en Culiacán	0.942	1
La Mamá de la Mamá	0.922	1
Break My Stride	0.917	1
Playa Sola	0.915	1
Paz en Este Amor	0.898	1
LOYAL (feat. Drake and Bad Bunny) – Remix	0.877	1

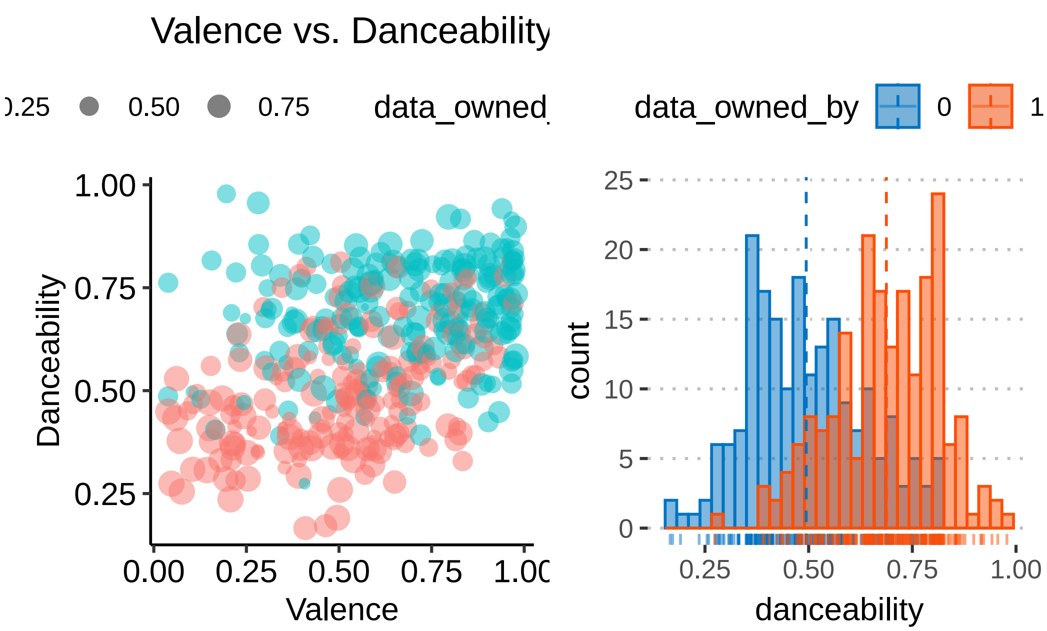


track_name	energy	data_owned_by
Dealer / Debtor	0.996	0
I Believe You	0.994	0
Mothers' Sons	0.990	0
No Man Born Evil	0.988	0
List & Heel	0.987	0
All Roads	0.985	0
Lancaster	0.985	0
What If I – 2005 Remaster	0.981	0

Danceability

```
# overall distribution in danceability between two
danceability <- tracks  |>
  gghistogram( x = "danceability", bins = 30,
   add = "mean", rug = TRUE,
   color = "data_owned_by", fill = "data_owned_by",
   palette = c("#0073C2FF", "#FC4E07"))+
  theme_pubclean()




danceabilty_valence <- tracks |>
  ggplot(aes(x = valence, y = danceability, color = data_owned_by, size = energy)) +
  scale_size(range = c(0, 4)) +
  geom_count(alpha = 0.5) +
  labs(x= "Valence", y= "Danceability") +
  ggtitle("Valence vs. Danceability For Merged Datasets") +
  theme(plot.title = element_text(face="bold")) +
  theme(legend.position="bottom") +
```

```
  guides(col=guide_legend(ncol = 3)) +
  theme_pubr()


danceabilty_valence + danceability
```



KNN

```
tracks_merged <- tracks |>
  select(-track_name)

# Create training (70%) and test (30%) sets for the
set.seed(123)  # for reproducibility (random sample)
spotify_split <- initial_split(tracks_merged, prop = 0.70)
spotify_train <- training(spotify_split)
spotify_test  <- testing(spotify_split)
spotify_split
```

```
<Training/Testing/Total>
<280/120/400>
```

```r
spotify_rec <- recipe(data_owned_by ~ ., data = spotify_train) |>
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes()) |>
  prep()
```

```r
knn_spec_tune <- nearest_neighbor(neighbors = tune())  |>
  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
```

```r
knn_fit <- knn_spec_tune %>%
  fit(data_owned_by ~. , data = spotify_train)
```

```
Warning: tune samples were requested but there were 280 rows in the data. 275
will be used.
```

```r
set.seed(123)
# 10-fold CV on the training dataset
cv_folds <- spotify_train |>  vfold_cv(v = 10)
```

```r
# Define our KNN model with tuning
# you can specify neigbors default five, can also put in tune() to tune
knn_spec_tune <-
  nearest_neighbor(neighbors = tune()) |>
  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
```

```r
# Define the workflow
wf_knn_tune <-
    workflow() |>
    add_model(knn_spec_tune) |>
    add_recipe(spotify_rec)
```

```r
# Fit the workflow on our predefined folds and hyperparameters
fit_knn_cv <- wf_knn_tune |>
  tune_grid(cv_folds,
            grid = data.frame(neighbors = c(1,5, seq(10, 100, 10))))

# Check the performance with collect_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     0.789    10  0.0294 Preprocessor1_Model01
 2         1 roc_auc  binary     0.800    10  0.0314 Preprocessor1_Model01
 3         5 accuracy binary     0.8      10  0.0297 Preprocessor1_Model02
 4         5 roc_auc  binary     0.892    10  0.0237 Preprocessor1_Model02
 5        10 accuracy binary     0.825    10  0.0195 Preprocessor1_Model03
 6        10 roc_auc  binary     0.903    10  0.0213 Preprocessor1_Model03
 7        20 accuracy binary     0.85     10  0.0218 Preprocessor1_Model04
 8        20 roc_auc  binary     0.904    10  0.0216 Preprocessor1_Model04
 9        30 accuracy binary     0.836    10  0.0233 Preprocessor1_Model05
10        30 roc_auc  binary     0.905    10  0.0200 Preprocessor1_Model05
# ... with 14 more rows
```

```r
final_wf <-
    wf_knn_tune |>
    finalize_workflow(select_best(fit_knn_cv, metric = "accuracy"))
# Check out the final workflow object
```

```r
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: nearest_neighbor()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_dummy()
* step_normalize()

-- Model -----------------------------------------------------------------------
K-Nearest Neighbor Model Specification (classification)

Main Arguments:
  neighbors = 20

Computational engine: kknn
```

```r
# Fitting our final workflow
final_fit <- final_wf|> fit(data = spotify_train)
# Examine the final workflow
final_fit
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: nearest_neighbor()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_dummy()
* step_normalize()

-- Model -----------------------------------------------------------------------

Call:
kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(20,     data, 5))
```

```
Type of response variable: nominal
Minimal misclassification: 0.1464286
Best kernel: optimal
Best k: 20
```

```
churn_pred <-  final_fit |> predict(new_data = spotify_test)

churn_pred |> head()
```

```
# A tibble: 6 x 1
  .pred_class
  <fct>
1 0
2 1
3 0
4 0
5 0
6 0
```

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

Decision Tree

```
# Create training (70%) and test (30%) sets for the
set.seed(123)  # for reproducibility (random sample)
spotify_split <- initial_split(tracks_merged, prop = 0.70)
spotify_train <- training(spotify_split)
spotify_test  <- testing(spotify_split)
spotify_split
```

```
<Training/Testing/Total>
<280/120/400>
```

```r
spotify_rec <- recipe(data_owned_by ~ ., data = spotify_train) |>
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes()) |>
  prep()
```

```r
#new spec, 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 x 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
```

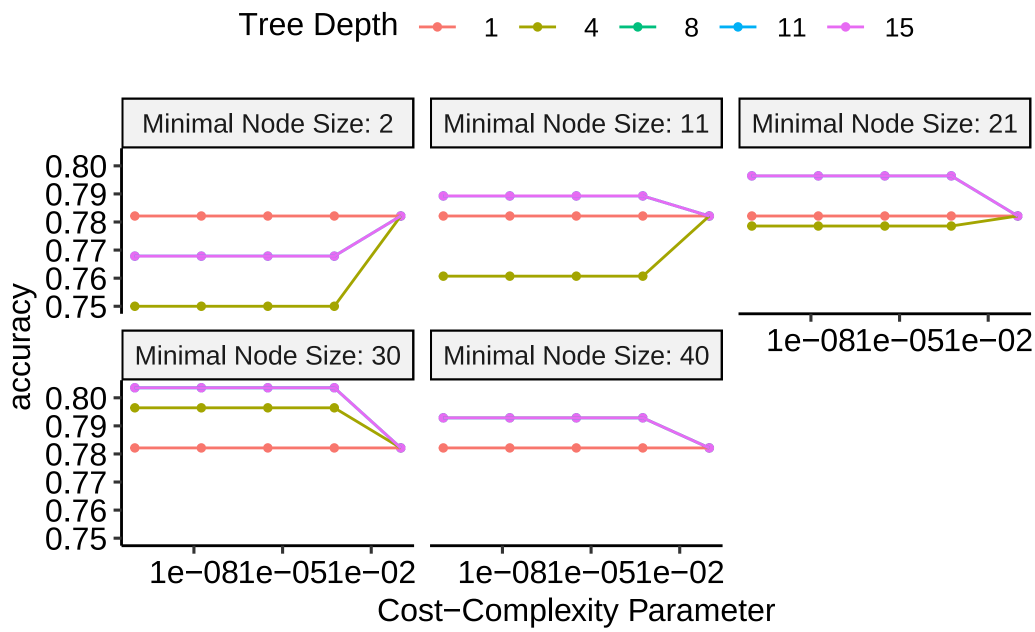
```r
wf_tree_tune <- workflow() |>
  add_recipe(spotify_rec) |>
  add_model(tree_spec_tune)
```

```r
#set up k-fold cv. This can be used for all the algorithms
decision_cv = spotify_train |>
  vfold_cv(v=10) #10 standard default lower for computational resources
```

```r
doParallel::registerDoParallel() #build trees in parallel
#200s
tree_rs <- tune_grid(
  tree_spec_tune,
  #specification
  data_owned_by ~ .,
  # model function
  resamples = decision_cv,
  #resample specificaton
  grid = tree_grid,
  #parameters to try
  metrics = metric_set(accuracy) #asses which combination of parameters is best
)
tree_rs
```

```
# Tuning results
# 10-fold cross-validation
# A tibble: 10 x 4
   splits           id     .metrics           .notes
   <list>           <chr>  <list>             <list>
 1 <split [252/28]> Fold01 <tibble [125 x 7]> <tibble [0 x 3]>
 2 <split [252/28]> Fold02 <tibble [125 x 7]> <tibble [0 x 3]>
 3 <split [252/28]> Fold03 <tibble [125 x 7]> <tibble [0 x 3]>
 4 <split [252/28]> Fold04 <tibble [125 x 7]> <tibble [0 x 3]>
 5 <split [252/28]> Fold05 <tibble [125 x 7]> <tibble [0 x 3]>
 6 <split [252/28]> Fold06 <tibble [125 x 7]> <tibble [0 x 3]>
 7 <split [252/28]> Fold07 <tibble [125 x 7]> <tibble [0 x 3]>
 8 <split [252/28]> Fold08 <tibble [125 x 7]> <tibble [0 x 3]>
 9 <split [252/28]> Fold09 <tibble [125 x 7]> <tibble [0 x 3]>
10 <split [252/28]> Fold10 <tibble [125 x 7]> <tibble [0 x 3]>
```

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




```
show_best <- show_best(tree_rs)
select_best <- select_best(tree_rs)

best <- flextable(show_best)
show <- flextable(select_best)

best
```

cost_complexity	tree_depth	min_n	.metric	.estimator	mean	n	std_err	.config
0.00000000010000	8	30	accuracy	binary	0.8035714	10	0.01433523	Preprocessor1_Mode
0.00000001778279	8	30	accuracy	binary	0.8035714	10	0.01433523	Preprocessor1_Mode
0.00000316227766	8	30	accuracy	binary	0.8035714	10	0.01433523	Preprocessor1_Mode
0.00056234132519	8	30	accuracy	binary	0.8035714	10	0.01433523	Preprocessor1_Mode
0.00000000010000	11	30	accuracy	binary	0.8035714	10	0.01433523	Preprocessor1_Mode

```
show
```

cost_complexity	tree_depth	min_n	.config
0.0000000001	8	30	Preprocessor1_Model086

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

```
Decision Tree Model Specification (classification)

Main Arguments:
  cost_complexity = 1e-10
  tree_depth = 8
  min_n = 30

Computational engine: rpart
```

```
final_tree_fit <- last_fit(final_tree,
                           data_owned_by~.,
                           spotify_split)

predict <- as.data.frame(final_tree_fit$.predictions) |> select(-.config) |> head()

flextable(predict) |>  theme_zebra()
```

.pred_0	.pred_1	.row	.pred_class	data_owned_by
0.8888889	0.11111111	10	0	
0.8888889	0.11111111	30	0	
0.9523810	0.04761905	60	0	
0.8888889	0.11111111	80	0	
0.8888889	0.11111111	120	0	
0.8888889	0.11111111	150	0	

```
tree_perf <- final_tree_fit %>%
  collect_metrics() %>%
  filter(.metric == "accuracy")
```

```r
tree_perf
```

```
# A tibble: 1 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.758 Preprocessor1_Model1
```

Bagged Tree

```r
# Create training (70%) and test (30%) sets for the
set.seed(123)  # for reproducibility (random sample)
spotify_split <- initial_split(tracks_merged, prop = 0.70)
spotify_train <- training(spotify_split)
spotify_test  <- testing(spotify_split)
spotify_split
```

```
<Training/Testing/Total>
<280/120/400>
```

```r
spotify_rec <- recipe(data_owned_by ~ ., data = spotify_train) |>
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) |>
  step_normalize(all_numeric(), -all_outcomes()) |>
  prep()
```

```r
bag_cv <- spotify_train %>% vfold_cv(v=5)

bag_spec_tune <- bag_tree(cost_complexity = tune(),
                          tree_depth = tune(),
                          min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart", times = 50)

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

```
# A tibble: 125 x 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_tune <- workflow() %>%
  add_recipe(spotify_rec) %>%
  add_model(bag_spec_tune)
wf_bag_tune
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: bag_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_dummy()
* step_normalize()

-- Model -----------------------------------------------------------------------
Bagged Decision Tree Model Specification (classification)

Main Arguments:
  cost_complexity = tune()
  tree_depth = tune()
  min_n = tune()

Engine-Specific Arguments:
  times = 50

Computational engine: rpart
```

```r
doParallel::registerDoParallel() #build trees in parallel
bag_rs <- tune_grid(
  wf_bag_tune,
  data_owned_by ~.,
  resamples = bag_cv, #resamples to use
  grid = bag_grid,
  metrics = metric_set(accuracy))
```

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

```r
bag_rs |> collect_metrics()
```

```
# A tibble: 125 x 9
   cost_complexity tree_depth min_n .metric  .esti~1  mean     n std_err .config
             <dbl>      <int> <int> <chr>    <chr>   <dbl> <int>   <dbl> <chr>
 1    0.0000000001          1     2 accuracy binary  0.779     5  0.0332 Prepro~
 2    0.0000000178          1     2 accuracy binary  0.779     5  0.0332 Prepro~
 3    0.00000316            1     2 accuracy binary  0.782     5  0.0345 Prepro~
 4    0.000562              1     2 accuracy binary  0.779     5  0.0332 Prepro~
 5    0.1                   1     2 accuracy binary  0.782     5  0.0345 Prepro~
 6    0.0000000001          4     2 accuracy binary  0.832     5  0.0332 Prepro~
 7    0.0000000178          4     2 accuracy binary  0.821     5  0.0282 Prepro~
 8    0.00000316            4     2 accuracy binary  0.818     5  0.0222 Prepro~
 9    0.000562              4     2 accuracy binary  0.821     5  0.0271 Prepro~
10    0.1                   4     2 accuracy binary  0.782     5  0.0345 Prepro~
# ... with 115 more rows, and abbreviated variable name 1: .estimator
```

```r
# get best model based on the metric
best_bag_mod <- select_best(bag_rs, "accuracy")

# fit the best model to the training data
final_bag <- finalize_workflow(wf_bag_tune, best_bag_mod) %>%
  fit(data = spotify_train)
```

```r
# make predictions on the test set using the fitted model
test_pred <- final_bag %>%
  predict(new_data = spotify_test) %>%
  bind_cols(spotify_test)
```

```r
# evaluate the accuracy of the fitted model on the test set
bag_perf <- test_pred %>%
  metrics(truth = data_owned_by, estimate = .pred_class)
```

Random Forest

```r
# Create training (70%) and test (30%) sets
set.seed(123)
spotify_split <- initial_split(tracks_merged, prop = 0.70)
spotify_train <- training(spotify_split)
spotify_test <- testing(spotify_split)

set.seed(234)
val_set <- validation_split(spotify_train,
                            strata = data_owned_by,
                            prop = 0.70)


# Create a recipe
spotify_rec <- recipe(data_owned_by ~ ., data = spotify_train) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  prep()
```

```r
# Create a Random Forest specification
rf_spec <-
  rand_forest(mtry = tune(),
              min_n = tune(),
              trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# Create a Random Forest workflow
rf_workflow <- workflow() %>%
  add_recipe(spotify_rec) %>%
  add_model(rf_spec)
```

```r
set.seed(123)
doParallel::registerDoParallel()
set.seed(345)
rf_res <-
  rf_workflow %>%
  tune_grid(val_set,
            grid = 25,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy))
```

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

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

```r
rf_res |> collect_metrics()
```

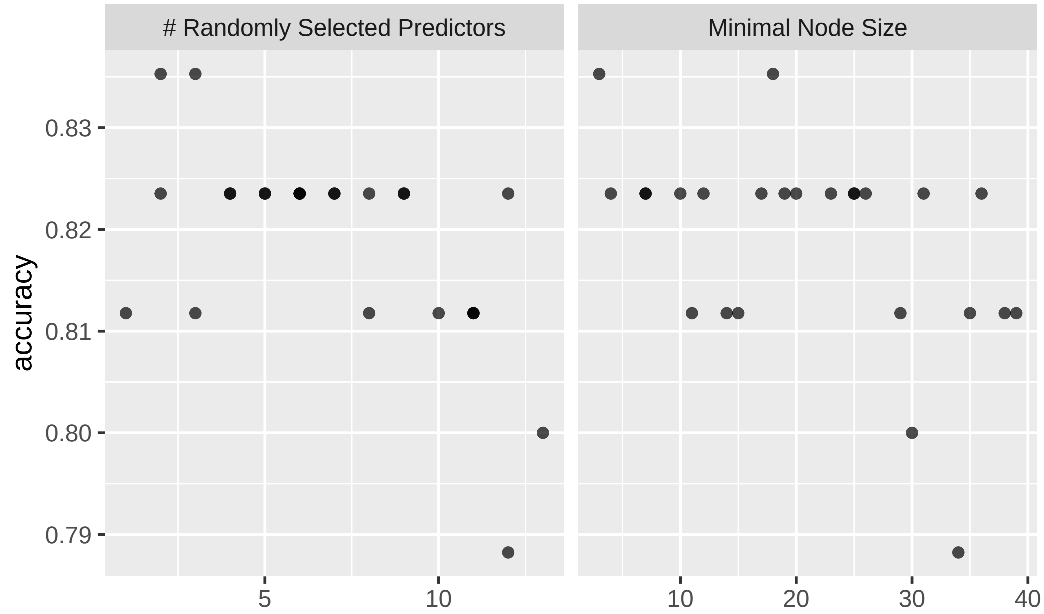
```
# A tibble: 25 x 8
    mtry min_n .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     6     7 accuracy binary     0.824     1      NA Preprocessor1_Model01
 2    13    30 accuracy binary     0.8       1      NA Preprocessor1_Model02
 3     4    25 accuracy binary     0.824     1      NA Preprocessor1_Model03
 4    12    34 accuracy binary     0.788     1      NA Preprocessor1_Model04
 5     6     4 accuracy binary     0.824     1      NA Preprocessor1_Model05
 6     8    10 accuracy binary     0.824     1      NA Preprocessor1_Model06
 7     1    38 accuracy binary     0.812     1      NA Preprocessor1_Model07
 8     5    25 accuracy binary     0.824     1      NA Preprocessor1_Model08
 9     8    11 accuracy binary     0.812     1      NA Preprocessor1_Model09
10     9    19 accuracy binary     0.824     1      NA Preprocessor1_Model10
# ... with 15 more rows
```

```r
rf_res %>%
  show_best(metric = "accuracy")
```

```
# A tibble: 5 x 8
   mtry min_n .metric  .estimator  mean     n std_err .config
  <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1     2     3 accuracy binary     0.835     1      NA Preprocessor1_Model21
```

```
2     3    18 accuracy binary     0.835     1       NA Preprocessor1_Model24
3     6     7 accuracy binary     0.824     1       NA Preprocessor1_Model01
4     4    25 accuracy binary     0.824     1       NA Preprocessor1_Model03
5     6     4 accuracy binary     0.824     1       NA Preprocessor1_Model05
```

```
autoplot(rf_res)
```




```
# Get the best Random Forest model
best_rf <- select_best(rf_res, "accuracy")
best_rf
```

```
# A tibble: 1 x 3
   mtry min_n .config
  <int> <int> <chr>
1     2     3 Preprocessor1_Model21
```

```
rf_res %>%
  collect_predictions()
```

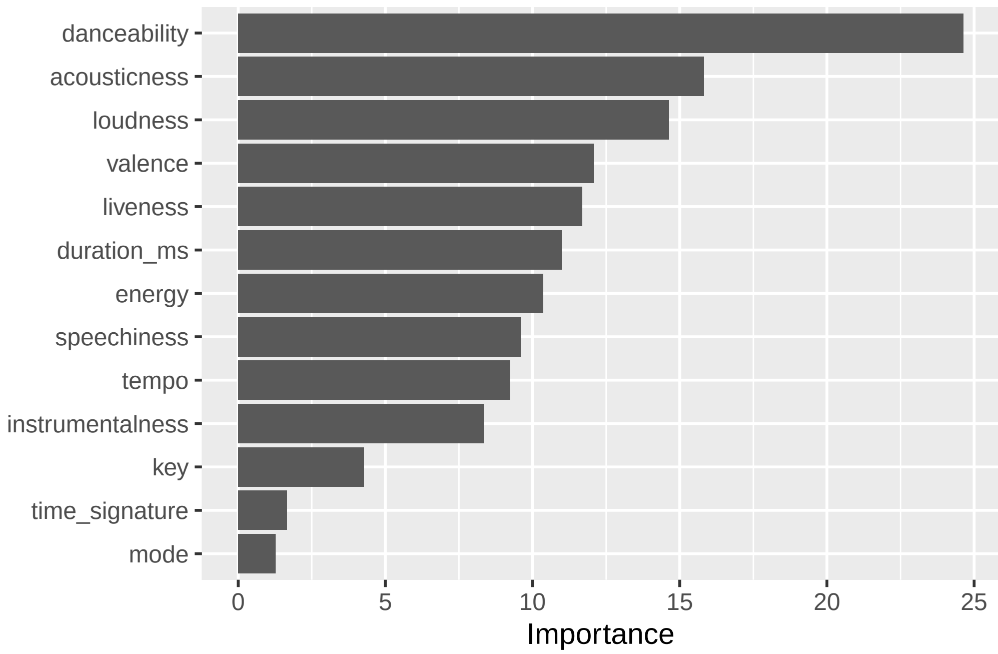
```
# A tibble: 2,125 x 7
   id         .pred_class  .row  mtry min_n data_owned_by .config
   <chr>      <fct>       <int> <int> <int> <fct>         <chr>
 1 validation 1               3     6     7 0             Preprocessor1_Model01
 2 validation 1               6     6     7 1             Preprocessor1_Model01
 3 validation 0              18     6     7 0             Preprocessor1_Model01
 4 validation 1              20     6     7 1             Preprocessor1_Model01
 5 validation 0              21     6     7 0             Preprocessor1_Model01
 6 validation 0              22     6     7 0             Preprocessor1_Model01
 7 validation 1              26     6     7 0             Preprocessor1_Model01
 8 validation 0              27     6     7 0             Preprocessor1_Model01
 9 validation 0              33     6     7 0             Preprocessor1_Model01
10 validation 1              40     6     7 1             Preprocessor1_Model01
# ... with 2,115 more rows
```

```r
doParallel::registerDoParallel()

# the last model
last_rf_mod <-
  rand_forest(mtry = 2, min_n = 3, trees = 1000) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

# the last workflow
last_rf_workflow <-
  rf_workflow %>%
  update_model(last_rf_mod)

# the last fit
set.seed(345)
last_rf_fit <-
  last_rf_workflow %>%
  last_fit(spotify_split)

last_rf_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [280/120]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
rf_metrics <- last_rf_fit %>%
  collect_metrics()
```

```r
last_rf_fit |>
  extract_fit_parsnip() |>
  vip::vip(num_features = 20)
```

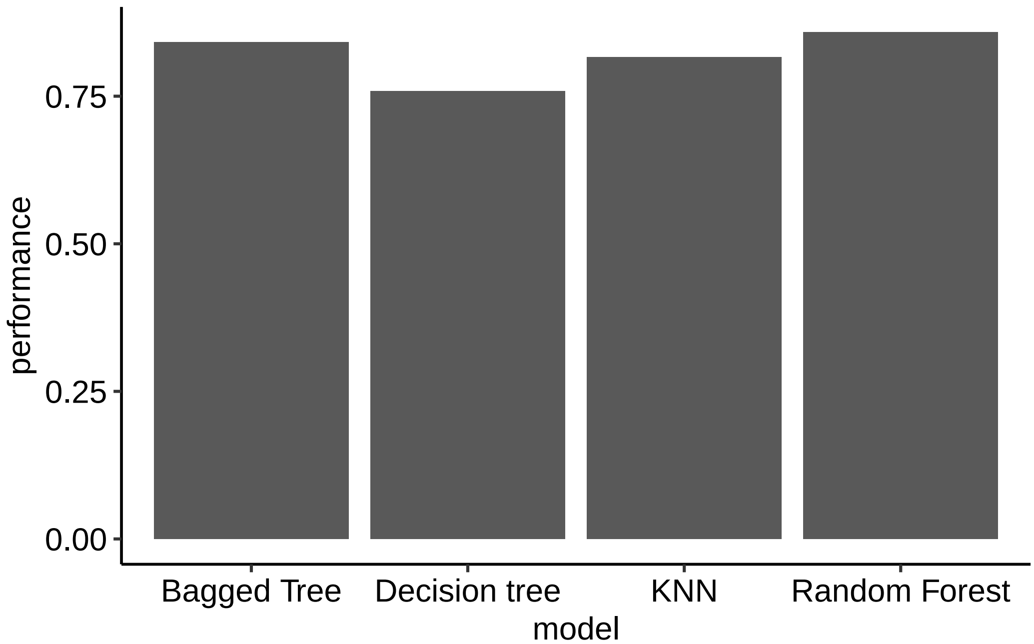



```r
rf_performance <- rf_metrics|>
  select(performance = .estimate, .metric) |>
  mutate(model = 'Random Forest') |>
  slice(1:1)

tree_performance <- tree_perf |>
  select(performance = .estimate, .metric) |>
  mutate(model = 'Decision tree') |>
  slice(1:1)

bag_performance <- bag_perf |>
  select(performance = .estimate, .metric) |>
  mutate(model = 'Bagged Tree') |>
```

```
  slice(1:1)

knn_performance <- knn_perf |>
  select(performance = .estimate, .metric) |>
  mutate(model = 'KNN') |>
  slice(1:1)

perfomance_metrics <- bind_rows(rf_performance,tree_performance, bag_performance, knn_perf
  mutate(model = as.factor(model))


flextable(perfomance_metrics) |>  theme_booktabs()
```

performance	metric	model
0.8583333	accuracy	Random Forest
0.7583333	accuracy	Decision tree
0.8416667	accuracy	Bagged Tree
0.8166667	accuracy	KNN

```
ggplot(perfomance_metrics, aes(x = model, y = performance)) +
  geom_col()+ theme_pubr()
```


performance
0.75
0.50
0.25
0.00
Bagged Tree
Decision tree
KNN
Random Forest
model