Codebook

Article: Kalustian, K. K. & Ruth, N. (2021). “Evacuate the Dancefloor”: Exploring and Classifying Spotify Music Listening Before and During the COVID-19 Pandemic in DACH Countries.

by Kework K. Kalustian

Note

> The code chunks can easily be copied to the clipboard by using the respective darkred-colored button on the top right corner of each code chunk. That is, the code chunks can be pasted in the R console of the user’s machine and can be run from there. Alternatively, the entire script can also be run at once (this will take some hours!). Plot-generating codes are here provided, too. With this, the reusability of the embedded as well as separately provided datasets is ensured.

Loading Packages

To run all the following code chunks, the following packages need to be installed/loaded:

```
################
### Packages ###
################

if (!require(pacman))
  install.packages("pacman", repo = "http://cran.us.r-project.org")

pacman::p_load("tidyverse", "magrittr", "rvest", "spotifyr", "lubridate", "scales",
               "ggpubr", "factoextra", "car", "rgl", "coin", "rstatix",
               "ggbeeswarm","caret", "e1071",  "parallel",
               "parallelMap", "vip", "ModelMetrics","pdp")
```

Data Retrieval I – Web-Scraping

To retrieve the necessary data, the following web-scraping techniques need to be used for each country and for each period (see the adjustments for the repetitions, e.g., “de”, “at”, and “ch”):

```
##############################################
### Data Acquisition for each DACH country  ###
##############################################


# First, we have to store the permanent Spotify-link.

# Inserting the respective Link (i.e., for "de", "at", and "ch")

url <- "https://spotifycharts.com/regional/de/daily/"

# Here we specify the entire streaming period (i.e., the sequence of n days).

streaming_period <- seq(as.Date("2019/03/11"), as.Date("2019/06/14"),
                        by = "day")

# For 2020: de-comment this chunk.
#
# streaming_period <- seq(as.Date("2020/03/11"), as.Date("2020/06/14"),
#                         by = "day")


# Next, we write a generic function that combines or, respectively, concatenates
# the URLs (for the entire period) by taking the permanent link from above and a
# blank argument (x) as another argument to which the URL should refer
# (i.e., our streaming_period).

gathering_urls <- function(x){paste0(url, x)}

# Using the just created function to apply it on the streaming period to finally
# get those n URLs.

all_urls  <- gathering_urls(streaming_period)

# Everything looks fine thus far. Hence, we create now a function that fills
# the desired column-names with the information we are going to retrieve from
# those n URLs (i.e., chart position, song/track title, artist, stream counts,
# dates, track_id).

spotifyR_scrapeR <- function(x) {page <- x

# Retrieving the 200 chart positions of each day.
chart_position <- page %>%
  read_html() %>%
  html_nodes(".chart-table-position") %>%
  html_text()

#Retrieving the 200 song/track titles of each day.

title <- page %>%
  read_html() %>%
  html_nodes("strong") %>%
  html_text()

# Retrieving the 200 artist names of each day.

artist <- page %>%
  read_html() %>%
  html_nodes(".chart-table-track span") %>%
  html_text()

# Retrieving the 200 stream counts of each day.
```

```
stream_count <- page %>%
  read_html() %>%
  html_nodes("td.chart-table-streams") %>%
  html_text()

# Retrieving the dates of for each day of the period.

date <- page %>%
  read_html() %>%
  html_nodes(".responsive-select~ .responsive-select+
                    .responsive-select .responsive-select-value") %>%
  html_text()

# Retrieving the track_id of for each day of the period.

track_id <-  page %>%
  read_html() %>%
  html_nodes("a") %>%
  html_attr("href") %>%
  str_remove("https://open.spotify.com/track/") %>%
  .[c(7:206)]

# Putting these chunks together in a table of the class.

tab <- data.frame(chart_position, title, artist, stream_count, date, track_id)

return(tab)}

# As the amount of data that should be retrieved is not that small, we can
# expect that this process will take some time. To know how long this process
# will last, we calculate the difference between the process initialization and
# its end.

init_time <- Sys.time()

# The actual process of web scraping: Applying the spotifyR_scrapeR-function
# to the object of that definitive URLs for each list element. That is, the just
# created spotifyR_scrapeR-function retrieves from each URL the desired
# information. (Please adjust the name of the data.frame for the other countries,
# i.e, "AT_Tracks", "CH_Tracks").

DE_Tracks_19 <- map_df(all_urls, spotifyR_scrapeR)

# End time of the retrieving-process.

end_time <- Sys.time()

# Difference (i.e., processing time to retrieve the desired information).

process_time <- end_time - init_time
print(process_time)

DE_Tracks_19$country <- "DE"

# Exporting and saving the retrieved charts for Germany as .csv-file.
write_csv(DE_Tracks_19, "DE_Tracks_19.csv")
```

Once this procedure is done for each period and for each country by adjusting the respective objects (cf., `DE_Tracks` , `AT_Tracks` , `CH_Tracks`), all retrieved tracks should be combined and saved/exported in one table:

```
# For the period in 2019

DE_Tracks_19 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/DE_Tracks_19.csv")
AT_Tracks_19 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/AT_Tracks_19.csv")
CH_Tracks_19 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/CH_Tracks_19.csv")

# Creating country cols.
AT_Tracks_19$country <- "AT"
DE_Tracks_19$country <- "DE"
CH_Tracks_19$country <- "CH"

DACH_charts_19 <- rbind(DE_Tracks_19, AT_Tracks_19, CH_Tracks_19)

# For the period in 2020

DE_Tracks_20 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/DE_Tracks_20.csv")

AT_Tracks_20 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/AT_Tracks_20.csv")

CH_Tracks_20 <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Countries/CH_Tracks_20.csv")

# Creating country cols.
AT_Tracks_20$country <-  "AT"
DE_Tracks_20$country <-  "DE"
CH_Tracks_20$country <-  "CH"


DACH_charts_20 <- rbind(DE_Tracks_20, AT_Tracks_20, CH_Tracks_20)

# Combining both periods

DACH_tracks <- rbind(DACH_charts_19, DACH_charts_20)

# Exporting the dataset
write_csv(DACH_tracks, "DACH_tracks.csv")
```

Data Retrieval II – Audio Features

Audio features for all tracks can be retrieved as follows. However, to use Spotify's API, users need to register as developers so they can enter their client ID and client secret (cf. the lines "PLEASE ENTER HERE YOUR ID/[CLIENT SECRET]"). To register with Spotify as a developer, please follow the instruction here Spotify, 2021 (https://developer.spotify.com/dashboard/login). Registration is free of charge.

```
########################
### Loading the Data ###
########################

DACH_charts <-  read.csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DA
CH/main/Datasets/Overall/DACH_tracks.csv")

################
### Scraping ###
################

# Extracting the Spotify IDs from a given data.frame
# to retrieve the audio features.

id <- unique(DACH_tracks$track_id)

set.seed(1, sample.kind = "Rounding")
id_s <- sample(id, replace = F)

# Overcoming the obstacle that the "get_track_audio_features" function can
# only retrieve audio features for 100 tracks at once.

Sys.setenv(SPOTIFY_CLIENT_ID = "PLEASE ENTER HERE YOUR ID")

# Developer secret

Sys.setenv(SPOTIFY_CLIENT_SECRET = "PLEASE ENTER HERE YOUR CLIENT SECRET")

# Generating an access token to use Spotify's API

token <- get_spotify_access_token(Sys.getenv("SPOTIFY_CLIENT_ID"),
                                  Sys.getenv("SPOTIFY_CLIENT_SECRET"))

Feat_scraper <- function(x) {
  # omitting warnings
  base::options(warn =-1)
  # assigning length of an ID vector to a proxy object
  entire <- length(x)
  # setting seed for repo purposes
  set.seed(1, sample.kind = "Rounding")
  # assigning 100 sampled IDs to a vector to account for Spotify's limit
  v1a <- as.character(sample(x, 100, replace = F))
  # assigning a tibble with features of those 100 IDs. This tibble will be
  # extended below.
  tib <- spotifyr::get_track_audio_features(v1a, token)
  # replacing any IDs with new ones if those IDs are already in the tibble
  if (any(x %in% tib$id) == T) {x = x[which(!x %in% tib$id)]}
  # creating a while loop on the condition that the rows of the tibble are
  # less and/or equal to the length of the entire object
  while (nrow(tib) <= entire) {
    # Setting seed for repo purposes
    set.seed(42, sample.kind = "Rounding")
    # assigning 100 sampled IDs from the new IDs from above to a base vector
    # according to Spotify's limit as long as the object IDs are greater
    # than 100. If the remaining IDs are less than 100, these remaining IDs
    # will be sampled.
    v1b <- as.character(sample(x, ifelse(length(x) > 100, 100, length(x)),
                               replace = F))
    # extending the tibble from above to create a complete tibble with all
    # retrieved audio features of all track IDs of the object in question
    tib %<>% full_join(spotifyr::get_track_audio_features(v1b,token),
                       by = c("danceability", "energy", "key", "loudness",
```

```
                              "mode", "speechiness", "acousticness",
                              "instrumentalness", "liveness",
                              "valence", "tempo", "type", "id", "uri",
                              "track_href", "analysis_url", "duration_ms",
                              "time_signature"))
    # replacing any IDs with new ones if those IDs are already in the tibble
    if (any(x %in% tib$id) == T) {x = x[which(!x %in% tib$id)]}
    # If the rows of the tibble are equal to the length of the entire object
    # in question…,
    if (nrow(tib) == entire)
      #…break the loop.
      break
  }
  # outputting the entire tibble
  return(tib)
}


Feats <- Feat_scraper(id_s)

Feats_id <- Feats %>%
  rename(track_id = id)

# Creating the main research dataset:
DACH_complete <- merge(DACH_tracks, Feats_id, by = "track_id", all = T )%>%
  group_by(country, date, chart_position) %>%
  mutate(date = as.Date(as.character(date), "%m/%d/%Y"),
         artist = sub("by\\s", "", artist)) %>%
  arrange(country, date, chart_position)

write_csv(DACH_complete , "DACH_complete.csv")
```

Exploratory Data Analysis

To get an overview of the stream count distributions, it is useful to plot histograms of the just scraped tracks and their stream counts frequencies.

```
#############
### DATA ###
#############

DACH_charts <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DAC
H/main/Datasets/Overall/DACH_complete.csv",
                        col_types = cols(uri = col_skip(), track_href = col_skip(), analysis
_url = col_skip(), type = col_skip()))

head(DACH_charts)
```

```
## # A tibble: 6 x 20
##   track_id    chart_position title    artist     stream_count date       country
##   <chr>                <dbl> <chr>    <chr>             <dbl> <date>     <chr>
## 1 14f1n3XCW3…              1 Wolke 10 MERO              62873 2019-03-11 AT
## 2 4BD9yUEFI0…              2 DNA      KC Rebell…        28890 2019-03-11 AT
## 3 5iIgIxtMNe…              3 Gib Ihm  SHIRIN DA…        24503 2019-03-11 AT
## 4 6VMAFT0zfi…              4 Capital… Capital B…        22216 2019-03-11 AT
## 5 0qzX2dAs1o…              5 Kameham… Azet, Zuna        21781 2019-03-11 AT
## 6 2pu5rVWy3z…              6 2x       Mathea            21356 2019-03-11 AT
## # … with 13 more variables: danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, time_signature <dbl>
```

```
# Data prep.

anyNA(DACH_charts)
```

```
## [1] TRUE
```

```
# Identifying NAs
apply(is.na(DACH_charts), 2, which)$title
```

```
## [1]  13535  90303 115201
```

```
DACH_charts[13535,]
```

```
## # A tibble: 1 x 20
##   track_id chart_position title artist stream_count date       country
##   <chr>             <dbl> <chr> <chr>         <dbl> <date>     <chr>
## 1 #                   135 <NA>  <NA>           6044 2019-05-17 AT
## # … with 13 more variables: danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, time_signature <dbl>
```

```
DACH_charts[90303,]
```

```
## # A tibble: 1 x 20
##   track_id chart_position title artist stream_count date       country
##   <chr>             <dbl> <chr> <chr>         <dbl> <date>     <chr>
## 1 #                   103 <NA>  <NA>         106195 2019-05-17 DE
## # … with 13 more variables: danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, time_signature <dbl>
```

```
DACH_charts[115201,]
```

```
## # A tibble: 1 x 20
##   track_id chart_position title artist stream_count date       country
##   <chr>             <dbl> <chr> <chr>         <dbl> <date>     <chr>
## 1 <NA>                 NA <NA>  <NA>             NA NA         <NA>
## # … with 13 more variables: danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, time_signature <dbl>
```

```
# Removing NAs
DACH_charts <- na.omit(DACH_charts)

# Creating a new dataframe without NAs

df_ml <- DACH_charts %>%

  # Selecting track_id, stream_count, country, and chart_position as extrinsic
  # features and Spotify's audio features as song-intrinsic characteristics


  dplyr::select(c(country, date, track_id, title, artist, stream_count,
                  chart_position, danceability, energy, loudness, acousticness,
                  instrumentalness, speechiness, liveness, valence, duration_ms,
                  tempo, mode)) %>%

  # Re-scaling some features to have all song-intrinsic features on the same
  # scale.

  mutate(country = as.factor(country),

         stream_count_rescaled = rescale(.$stream_count, to = c(0,1),
                                         from = c(min(DACH_charts$stream_count),
                                                  max(DACH_charts$stream_count))),

         chart_position_rescaled = rescale(.$chart_position, to = c(0,1),
                                           from = c(max(DACH_charts$chart_position),
                                                    min(DACH_charts$chart_position))) ,

         loudness_rescaled = rescale(.$loudness, to = c(0,1),
                                     from = c(min(DACH_charts$loudness),
                                              max(DACH_charts$loudness))),

         duration_ms_rescaled = rescale(.$duration_ms, to = c(0,1),
                                        from = c(min(DACH_charts$duration_ms),
                                                 max(DACH_charts$duration_ms))),

         tempo_rescaled =  tempo/1000,


         mode_fct = as.factor(mode))

str(df_ml)
```

```
## tibble [115,198 × 24] (S3: tbl_df/tbl/data.frame)
##  $ country                : Factor w/ 3 levels "AT","CH","DE": 1 1 1 1 1 1 1 1 1 1 ...
##  $ date                   : Date[1:115198], format: "2019-03-11" "2019-03-11" ...
##  $ track_id               : chr [1:115198] "14f1n3XCW3fQhAODW0CwLK" "4BD9yUEFI07WaDJ381DD
Mq" "5iIgIxtMNeogKaFUfvuAbs" "6VMAFT0zfide2Jg8MVZAPz" ...
##  $ title                  : chr [1:115198] "Wolke 10" "DNA" "Gib Ihm" "Capital Bra je m'a
ppelle" ...
##  $ artist                 : chr [1:115198] "MERO" "KC Rebell, Summer Cem, Capital Bra" "S
HIRIN DAVID" "Capital Bra" ...
##  $ stream_count           : num [1:115198] 62873 28890 24503 22216 21781 ...
##  $ chart_position         : num [1:115198] 1 2 3 4 5 6 7 8 9 10 ...
##  $ danceability           : num [1:115198] 0.77 0.828 0.928 0.626 0.8 0.845 0.732 0.73 0.
839 0.719 ...
##  $ energy                 : num [1:115198] 0.797 0.686 0.402 0.768 0.648 0.509 0.583 0.72
5 0.733 0.704 ...
##  $ loudness               : num [1:115198] -4.99 -5.24 -10.64 -4.65 -5.64 ...
##  $ acousticness           : num [1:115198] 0.0662 0.0519 0.217 0.159 0.37 0.224 0.519 0.3
87 0.488 0.0691 ...
##  $ instrumentalness       : num [1:115198] 3.81e-06 2.23e-06 3.38e-05 1.86e-04 0.00 0.00
1.97e-05 0.00 8.28e-05 0.00 ...
##  $ speechiness            : num [1:115198] 0.0693 0.264 0.32 0.166 0.326 0.102 0.125 0.31
8 0.273 0.0476 ...
##  $ liveness               : num [1:115198] 0.0858 0.11 0.0957 0.0848 0.149 0.171 0.108 0.
153 0.0923 0.166 ...
##  $ valence                : num [1:115198] 0.393 0.544 0.578 0.753 0.73 0.282 0.277 0.625
0.409 0.628 ...
##  $ duration_ms            : num [1:115198] 172827 253346 162997 187000 160139 ...
##  $ tempo                  : num [1:115198] 100 156 98 80.1 93 ...
##  $ mode                   : num [1:115198] 0 1 0 0 1 1 1 0 0 1 ...
##  $ stream_count_rescaled  : num [1:115198] 0.03033 0.01308 0.01085 0.00969 0.00947 ...
##  $ chart_position_rescaled: num [1:115198] 1 0.995 0.99 0.985 0.98 ...
##  $ loudness_rescaled      : num [1:115198] 0.839 0.828 0.581 0.855 0.81 ...
##  $ duration_ms_rescaled   : num [1:115198] 0.264 0.438 0.242 0.294 0.236 ...
##  $ tempo_rescaled         : num [1:115198] 0.1 0.156 0.098 0.0801 0.093 ...
##  $ mode_fct               : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 2 1 1 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:3] 13535 90303 115201
##   ..- attr(*, "names")= chr [1:3] "13535" "90303" "115201"
```

```
anyNA(df_ml)
```

```
## [1] FALSE
```

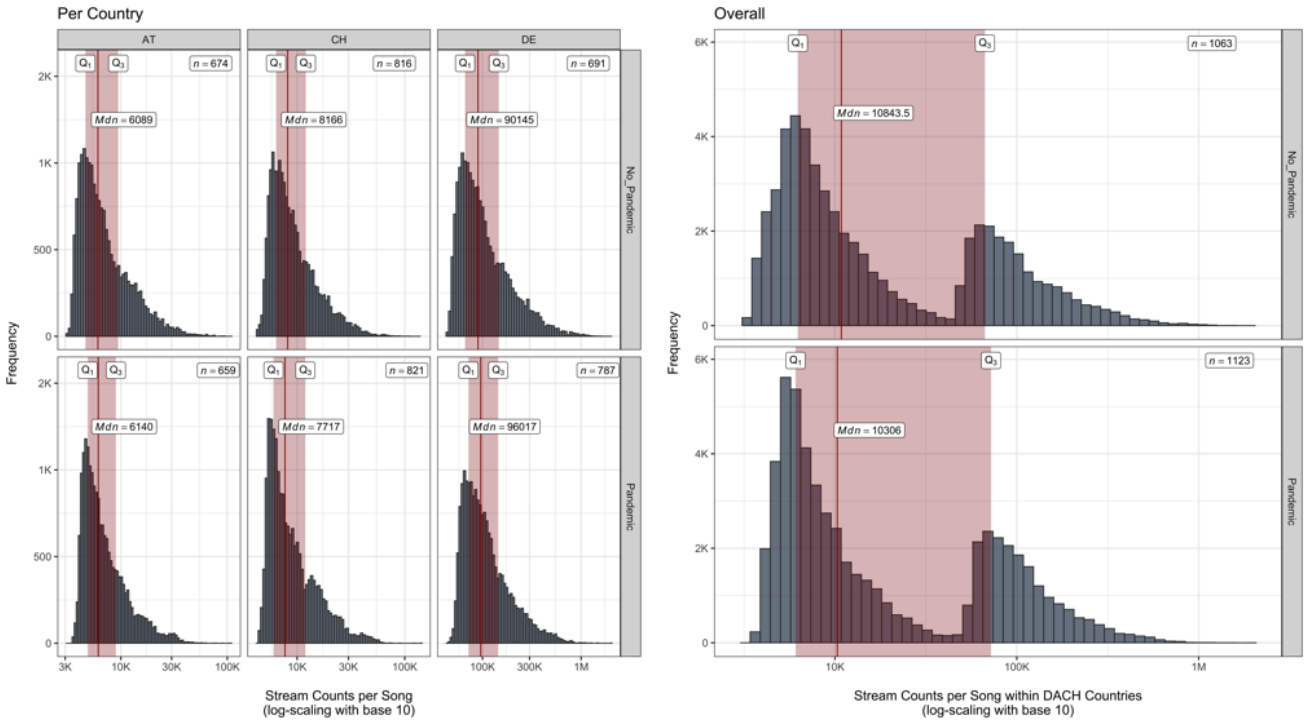
```
# Creating a new factor col. of the pandemic periods.
df_ml$Pandemic <- ifelse(df_ml$date > as.Date("2020-03-10"), "Pandemic",
                         "No_Pandemic" )

df_ml %<>%
  mutate( Pandemic = factor(Pandemic, labels = c( "No_Pandemic", "Pandemic"),
                            ordered = T))

######################################################################## #
# exporting the data. This dataset will be used ################## #
# in the code/script: "Analysis_Step2_kMeans+Difference Tests" ### #
# So it is a good idea to run all scripts of this repo in the same #
# working directory.############################################## #

write_csv(df_ml, "df_ml.csv")
```

To get finally a visual impression of the stream count frequencies, the histograms can be plotted with these code lines:

```
################################
### Histogram | Distribution ###
################################

# Global Plotting Layout
layout <- theme_bw(base_size = 14) +
  theme(axis.title.y = element_text(size = 14),
        axis.title.x = element_text(size = 14),
        legend.position = "top",
        legend.key = element_rect(color = "white"),
        legend.background = element_rect(fill = "white",
                                         linetype = "solid",
                                         color = "grey90"),
        plot.margin = unit(c(.66, .33, .66, .66), "cm"))

# Summary stats per country
distinct_songs <- df_ml %>%
  group_by(Pandemic, country) %>%
  summarize(distict_songs = n_distinct(track_id))


quants <- df_ml %>%
  group_by(Pandemic, country) %>%
  summarize(Q1=quantile(stream_count, .25),
            Median = median(stream_count),
            Q3=quantile(stream_count, .75),
            Max=max(stream_count))


# Histograms per country

p <- ggplot(data=df_ml, aes(stream_count, fill = Pandemic))+

  geom_histogram(color="grey20",fill ="#2e4057", alpha = .7,
                 binwidth = function(x) 2 * IQR(x) / (length(x)^(1/3)),
                 show.legend = F)+

  geom_rect(data=quants,
            aes(x = NULL,y = NULL, xmin = Q1,  xmax=Q3,
                ymin = -Inf, ymax = Inf),alpha=0.3,fill = "darkred",
            show.legend = F) +

  geom_vline(data=quants,aes(xintercept = Median), color = "darkred",
             linetype = "solid", size=.5) +

  geom_label(data=quants,aes(x = Q1, y = 1575), fill = "white",
             label = "Q[1]", parse = T) +

  geom_label(data=quants, aes( x = Median *1.75, y = 1250), fill = "white",
             label = paste0("italic(Mdn) == ", quants$Median), parse=T)+

  geom_label( data=quants,aes( x = Q3, y = 1575), fill ="white",
              label = "Q[3]", parse = T, )+

  geom_label(data=quants, aes( x =  Max*0.75, y = 1575), fill ="white",
             label = paste0("italic(n) == ", distinct_songs$distict_songs),
             parse=T)+

  scale_y_continuous(labels = label_number_si(accuracy = NULL))+

  scale_x_log10(labels = label_number_si(accuracy = NULL))+
```

```
  labs(x = "\nStream Counts per Song\n(log-scaling with base 10)",
       y = "Frequency\n") +

  ggtitle(label="Per Country")+

  layout

p <- p +  facet_grid(Pandemic~country, scales = "free")


# Summary stats across all countries
distinct_songs2 <- df_ml %>%
  group_by(Pandemic) %>%
  summarize(distict_songs = n_distinct(track_id))


quants2 <- df_ml %>%
  group_by(Pandemic) %>%
  summarize(Q1=quantile(stream_count, .25),
            Median = median(stream_count),
            mean = mean(stream_count),
            Q3=quantile(stream_count, .75),
            Max=max(stream_count))


p2 <- ggplot(data=df_ml, aes(stream_count, fill = Pandemic))+
  geom_histogram(color="grey20",fill ="#2e4057", alpha = .7,
                 binwidth = function(x) 2 * IQR(x) / (length(x)^(1/3)),
                 show.legend = F)+

  geom_rect(data=quants2,
            aes(x = NULL,y = NULL, xmin = Q1,  xmax=Q3,
                ymin = -Inf, ymax = Inf),alpha=0.3,fill = "darkred",
            show.legend = F) +

  geom_vline(data=quants2,aes(xintercept = Median), color = "darkred",
             linetype = "solid", size=.5) +

  geom_label(data=quants2,aes(x = Q1, y = 5975), fill = "white",
             label = "Q[1]", parse = T) +

  geom_label(data=quants2, aes( x = Median*1.5, y = 4500), fill = "white",
             label = paste0("italic(Mdn) == ", quants2$Median), parse=T)+

  geom_label( data=quants2,aes( x = Q3, y = 5975), fill ="white",
              label = "Q[3]", parse = T )+

  geom_label(data=quants2, aes( x =  Max*.75, y = 5975), fill ="white",
             label = paste0("italic(n) == ", distinct_songs2$distict_songs),
             parse=T)+

  scale_y_continuous(labels = label_number_si(accuracy = NULL))+

  scale_x_log10(labels = label_number_si(accuracy = NULL))+

  labs(x = "\nStream Counts per Song within DACH Countries\n(log-scaling with base 10)",
       y = "Frequency\n") +

  ggtitle(label="Overall")+

  layout

p2 <- p2 + facet_grid(Pandemic~., scales = "free")
```

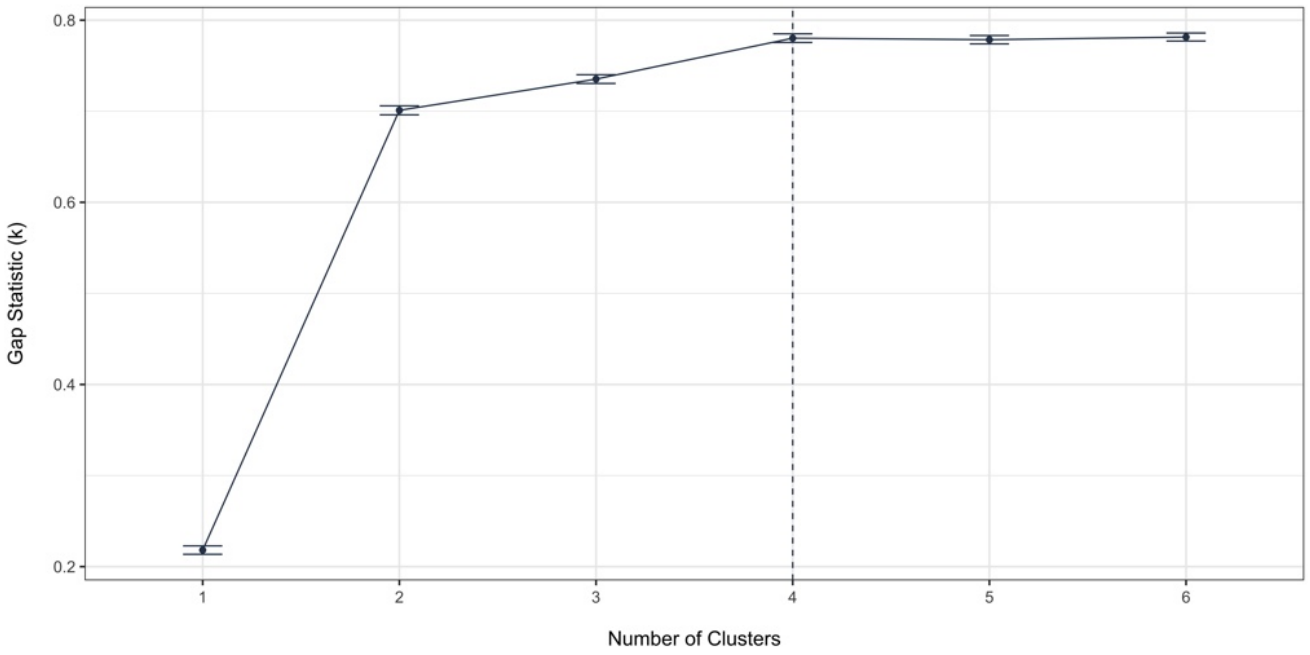
```
# Combining both plots
ggarrange(p, p2)
```



k-Means Clustering and Difference Tests

To reduce the dimensionality of the audio features, the k-means algorithm was used.

```
# importing the data that has previously been saved in the working directory (cf.
# code/script: "Analysis_Step1_Cleaning_&_EDA")

df_ml <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DACH/mai
n/Datasets/Overall/df_ml.csv")
### Finding K

df_ml_km_k <- df_ml %>%
  group_by(track_id, title, mode) %>%
  dplyr::select( danceability, energy, loudness_rescaled, valence,
                 tempo_rescaled) %>%
  distinct() %>% ungroup()

set.seed(123, sample.kind = "Rounding")

gap_stat <- cluster::clusGap(
  df_ml_km_k[-c(1:2)], FUN = kmeans, nstart = 50, iter.max = 100,
  K.max = 6, B = 500)


set.seed(1234, sample.kind = "Rounding")
fviz_gap_stat(gap_stat,
              linecolor = "#2e4057",
              maxSE = list(method="Tibs2001SEmax", SE.factor =1))+
  labs(x = "\nNumber of Clusters" , y = "Gap Statistic (k)\n" , title = "",
       subtitle = "") +
  layout
```



The actual k-means algorithm, and a PCA for visual purposes:

```
# 4 Clusters seem reasonable.

# k-means clustering

df_ml_ttl  <- df_ml %>%
  group_by(track_id, title, mode) %>%
  dplyr::select(danceability, energy, loudness_rescaled, valence,
                tempo_rescaled) %>% ungroup()


set.seed(14, sample.kind = "Rounding")

km <- kmeans(df_ml_ttl[-c(1:2)], 4, iter.max = 100, nstart = 50,
             algorithm = "Hartigan")


km$centers
```

```
##   mode danceability    energy loudness_rescaled   valence tempo_rescaled
## 1    1    0.6718704 0.5715458         0.7383946 0.3348963      0.1201192
## 2    0    0.7544339 0.7160936         0.8115806 0.6783576      0.1214350
## 3    1    0.7375228 0.7223513         0.8227384 0.6400781      0.1190675
## 4    0    0.6789342 0.6050810         0.7479485 0.3496640      0.1195543
```

```
km$betweenss/km$totss*100
```

```
## [1] 83.2686
```

```
# PCA for viz

pca <- prcomp(df_ml_ttl[-c(1:2)], center = T, scale. = T)

ind_coord <- as.data.frame(get_pca_ind(pca)$coord)

# Adding clusters from the k-means algo

ind_coord$clust <- as.factor(as.numeric(km$cluster))

# Adding countries from the original data set

ind_coord$Pandemic <- as.factor(df_ml$Pandemic)

head(ind_coord)
```

```
##        Dim.1       Dim.2        Dim.3      Dim.4       Dim.5       Dim.6 clust
## 1 -0.7767113 -0.10810725 -0.075203369 -1.2319571 -0.97155263  0.30591955     4
## 2  1.7751974 -2.90097435  0.202194774 -0.1405736 -0.07788222  0.16286207     4
## 3 -0.7452492  0.02955894 -0.009488748  1.7721170 -0.55024842 -0.17828879     3
## 4 -0.5389675 -0.80148600 -1.595316579  0.4238355  0.49987002 -0.19245571     3
## 5 -1.2240486 -0.18953229 -0.515724971 -1.7422439  1.10154714 -0.02483177     2
## 6 -0.6901429 -0.51983209  0.487565982 -0.4536766 -1.20154405  0.05218646     4
##      Pandemic
## 1 No_Pandemic
## 2 No_Pandemic
## 3 No_Pandemic
## 4 No_Pandemic
## 5 No_Pandemic
## 6 No_Pandemic
```

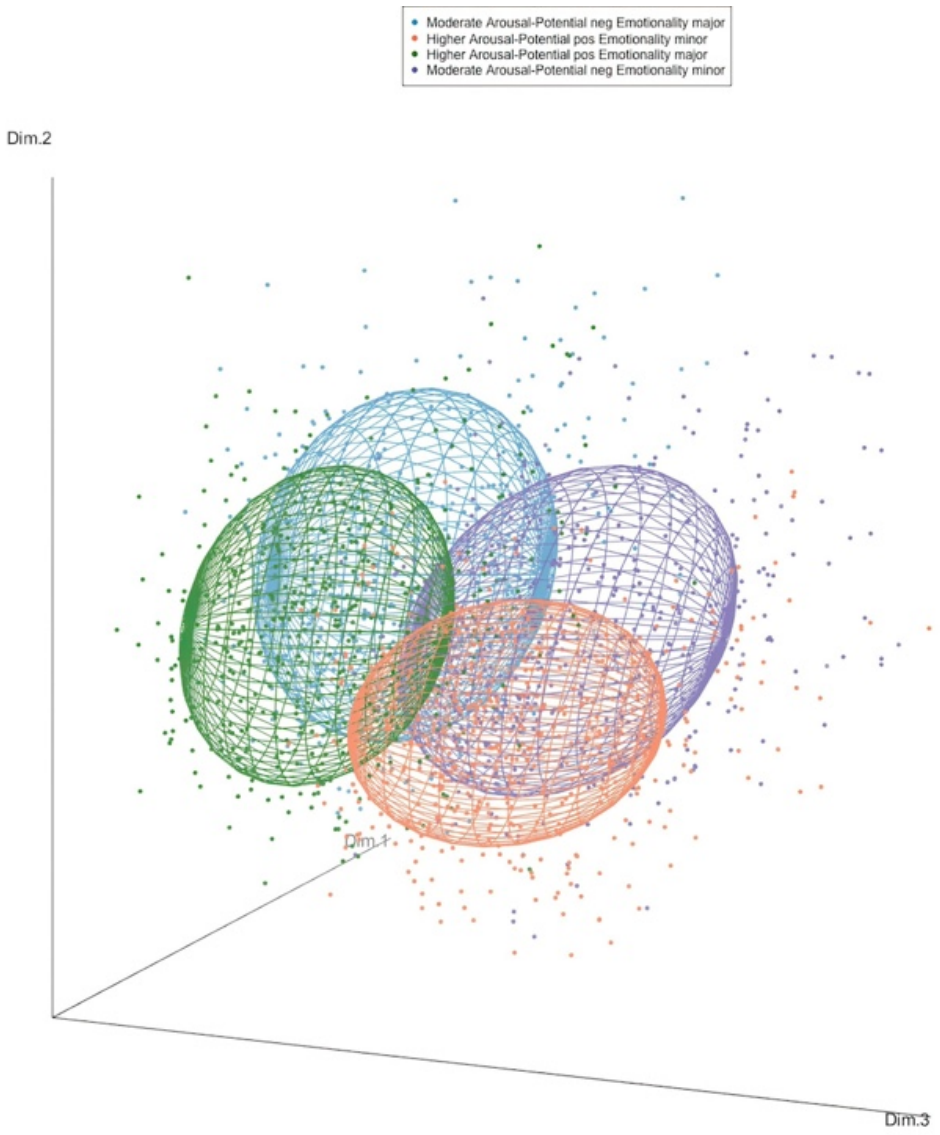
```
eigenvalue <- round(get_eigenvalue(pca), 1)

variance_percent <- eigenvalue$variance.percent

print(eigenvalue)
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1        2.0             33.0                         33.0
## Dim.2        1.2             20.0                         53.0
## Dim.3        1.0             16.6                         69.6
## Dim.4        0.9             15.8                         85.4
## Dim.5        0.6             10.0                         95.4
## Dim.6        0.3              4.6                        100.0
```

```
ind_coord$clust <- as.factor(as.numeric(ind_coord$clust))

Dim.1 <- ind_coord$Dim.1

Dim.2 <- ind_coord$Dim.2

Dim.3 <- ind_coord$Dim.3

# Adding the clusters to the original data set

df_ml$mood_clust_fct <- as.factor(km$clust)

# ################################################### #
# exporting the data. This dataset will be used ####### #
# in the code/script: "Analysis_Step3_SVM+VIP+PDP." ### #
# So it is a good idea to run all scripts of this repo# #
# in the same working directory.###################### #

write_csv(df_ml, "df_ml_full.csv")

# cluster labels
Mood_cluster <- df_ml %>%
  group_by(country, track_id, stream_count, Pandemic) %>%
  mutate(mood_clust_fct =
           recode_factor(mood_clust_fct,
                         '1' = "Moderate Arousal-Potential neg Emotionality major",
                         '2' = "Higher Arousal-Potential pos Emotionality minor",
                         '3' = "Higher Arousal-Potential pos Emotionality major",
                         '4' = "Moderate Arousal-Potential neg Emotionality minor"))
```

A 3D plot is helpful to get an impression of the compactness and dispersion of the distinct mood clusters:

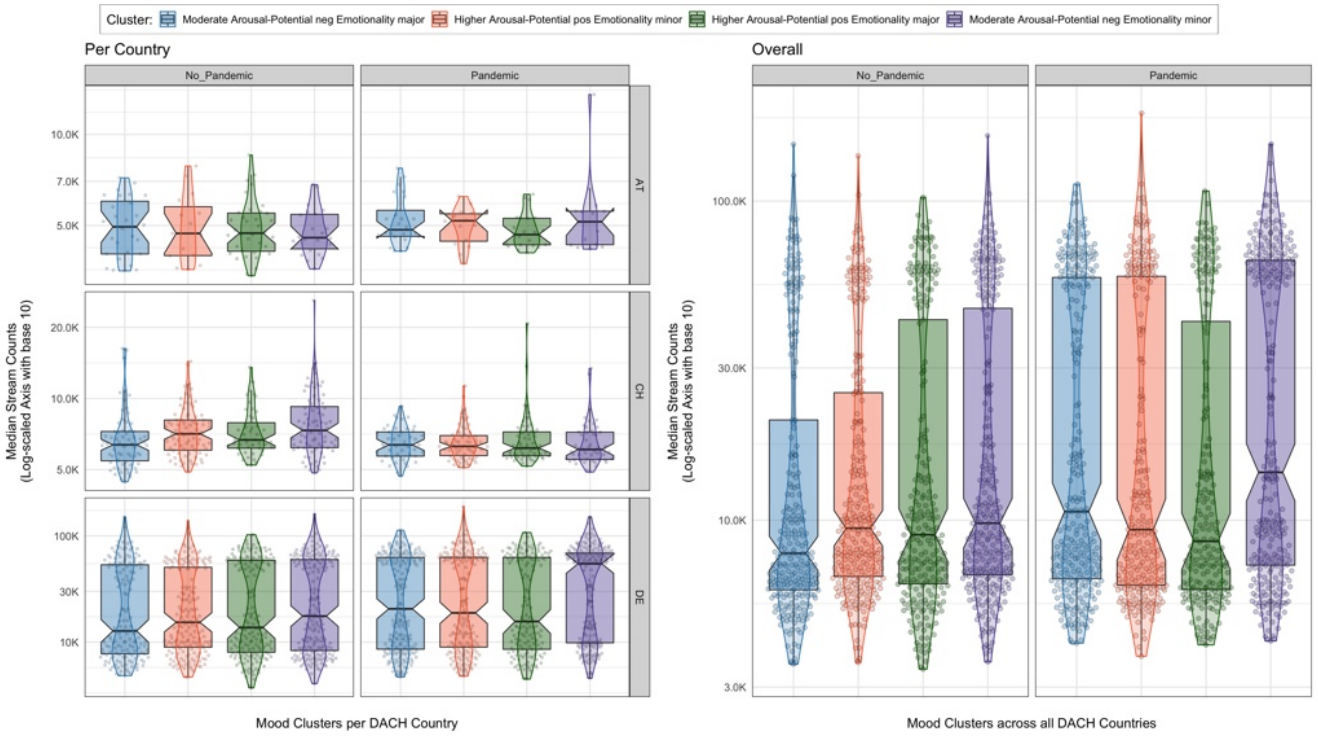
```
# 3D Plotting PCA
# color palette

cols <- c("#3288BD", "#F46D43", "darkgreen" , "#5E4FA2")

# 3D Plot

scatter3d(x = Dim.1, y = Dim.2, z = Dim.3,
          # xlab = "Dim.1 (33%)", ylab = "Dim.2 (20%)",
          # zlab = "Dim.3 (16.6%)",
          groups = as.factor(df_ml$mood_clust_fct) , surface = F,
          grid = T, surface.col = I(cols), ellipsoid = T, axis.scales = F ,
          axis.ticks = F,
          ellipsoid.alpha = 0.0, level = .6827,
          axis.col = c("black", "black", "black"))
# perspective
view3d(theta = -75, phi = 0) # zoom = .815

# window size (width & height in px)
par3d(windowRect=c(0,0,1920,1080))

# legend
legend3d("topright", legend = levels(Mood_cluster$mood_clust_fct), col=cols, pch=16)
```



Differences

Next, possible differences in the stream counts in the identified clusters can be plotted and tested like this:

```
# ########################## #
# Plots and Difference Tests #
# ########################## #
cols <- c("#3288BD", "#F46D43", "darkgreen" , "#5E4FA2")

Mood_cluster1 <- Mood_cluster %>%
  group_by(track_id) %>%
  dplyr::select(country, Pandemic, track_id, stream_count, mood_clust_fct) %>%
  summarize(track_id = track_id[1],
            country =country[1],
            Pandemic=Pandemic[1],
            mood_clust_fct = mood_clust_fct[1],
            streams = median(stream_count)) %>%
  arrange(desc(streams))

M1 <- Mood_cluster1 %>%
  ggplot( aes(x=mood_clust_fct, y=streams, fill = mood_clust_fct))+
  geom_boxplot(notch = T, alpha =.4, outlier.alpha = 2,
               outlier.size = .1, outlier.color = "grey")+
  geom_violin(width = 0.4, aes(color=mood_clust_fct), alpha=.2)+
  geom_quasirandom(shape = 21,size=1, dodge.width = .5, color = "black",
                   alpha=.2,show.legend = F)+
  scale_y_log10(labels = label_number_si(accuracy = NULL))+
  scale_color_manual("Cluster:", values = I(cols))+
  scale_fill_manual("Cluster:", values = I(cols)) +
  ggtitle(label="Per Country")+
  labs(x="\nMood Clusters per DACH Country",
       y="Median Stream Counts\n(Log-scaled Axis with base 10)\n")+
  layout+
  theme(axis.text.x  = element_blank(),
        axis.ticks = element_blank())+
  facet_grid(country~Pandemic, scales = "free")

M2 <- Mood_cluster1%>%
  ggplot( aes(x=mood_clust_fct, y=streams, fill = mood_clust_fct))+
  geom_boxplot(notch = T, alpha =.4, outlier.alpha = 2,
               outlier.size = .1, outlier.color = "grey")+
  geom_violin(width = 0.5, aes(color=mood_clust_fct), alpha=.2)+
  geom_quasirandom(shape = 21,size=2, dodge.width = .5, color = "black",
                   alpha=.3,show.legend = F)+
  scale_y_log10(labels = label_number_si(accuracy = NULL))+
  scale_color_manual("Cluster:", values = I(cols))+
  scale_fill_manual("Cluster:", values = I(cols)) +
  labs(x="\nMood Clusters across all DACH Countries",
       y="Median Stream Counts\n(Log-scaled Axis with base 10)\n")+
  ggtitle(label="Overall")+
  layout+
  theme(axis.text.x  = element_blank(),
        axis.ticks = element_blank())+
  facet_grid(~Pandemic, scales = "free")


ggarrange( M1, M2, common.legend = T)
```




```
# ################ #
# Difference Tests #
# ################ #

# Summary stats across all countries for each cluster and both periods
Mood_cluster1 %>%
  group_by(mood_clust_fct, Pandemic) %>%
  get_summary_stats(type = "median")
```

```
## # A tibble: 8 x 5
##   Pandemic    mood_clust_fct                                  variable     n median
##   <chr>       <fct>                                           <chr>    <dbl>  <dbl>
## 1 No_Pandemic Moderate Arousal-Potential neg Emotionality… streams    250  7874.
## 2 Pandemic    Moderate Arousal-Potential neg Emotionality… streams    275 10639
## 3 No_Pandemic Higher Arousal-Potential pos Emotionality m… streams    268  9436.
## 4 Pandemic    Higher Arousal-Potential pos Emotionality m… streams    255  9342.
## 5 No_Pandemic Higher Arousal-Potential pos Emotionality m… streams    257  9003
## 6 Pandemic    Higher Arousal-Potential pos Emotionality m… streams    228  8591.
## 7 No_Pandemic Moderate Arousal-Potential neg Emotionality… streams    288  9786.
## 8 Pandemic    Moderate Arousal-Potential neg Emotionality… streams    283 14124
```

```
# Summary stats per country for each cluster and both periods
Mood_cluster1 %>%
  group_by(mood_clust_fct, Pandemic, country) %>%
  get_summary_stats(type = "median")
```

```
## # A tibble: 24 x 6
##    country Pandemic   mood_clust_fct                       variable     n median
##    <chr>   <chr>      <fct>                                <chr>    <dbl>  <dbl>
##  1 AT      No_Pandem… Moderate Arousal-Potential neg Emot… streams     28  4952.
##  2 CH      No_Pandem… Moderate Arousal-Potential neg Emot… streams     68  6370.
##  3 DE      No_Pandem… Moderate Arousal-Potential neg Emot… streams    154 12744
##  4 AT      Pandemic   Moderate Arousal-Potential neg Emot… streams     24  4844.
##  5 CH      Pandemic   Moderate Arousal-Potential neg Emot… streams     45  6366
##  6 DE      Pandemic   Moderate Arousal-Potential neg Emot… streams    206 20576
##  7 AT      No_Pandem… Higher Arousal-Potential pos Emotio… streams     15  4710
##  8 CH      No_Pandem… Higher Arousal-Potential pos Emotio… streams     79  7084
##  9 DE      No_Pandem… Higher Arousal-Potential pos Emotio… streams    174 15366
## 10 AT      Pandemic   Higher Arousal-Potential pos Emotio… streams     16  5186.
## # … with 14 more rows
```

```
# ###################### #
# Overall DACH countries #
# ###################### #

# Dunn tests with holm correction across all countries for each cluster and both periods
Mood_cluster1 %>%
  group_by(mood_clust_fct) %>%
  dunn_test(streams ~ Pandemic) %>%
  adjust_pvalue(method = "holm")
```

```
## # A tibble: 4 x 10
##   mood_clust_fct      .y.    group1 group2    n1    n2 statistic       p   p.adj
## * <fct>               <chr>  <chr>  <chr>  <int> <int>     <dbl>   <dbl>   <dbl>
## 1 Moderate Arousal-P… strea… No_Pa… Pande…   250   275      3.53  4.15e-4 0.00153
## 2 Higher Arousal-Pot… strea… No_Pa… Pande…   268   255      0.791 4.29e-1 0.858
## 3 Higher Arousal-Pot… strea… No_Pa… Pande…   257   228      0.126 9.00e-1 0.900
## 4 Moderate Arousal-P… strea… No_Pa… Pande…   288   283      3.55  3.84e-4 0.00153
## # … with 1 more variable: p.adj.signif <chr>
```

```
# Effect sizes for the differences from above by using the z-values of the dunn
# tests via rstatix::wilcox_effsize
Mood_cluster1 %>%
  group_by(mood_clust_fct) %>%
  wilcox_effsize(streams ~ Pandemic)
```

```
## # A tibble: 4 x 8
##   .y.    group1   group2 effsize mood_clust_fct               n1    n2 magnitude
## * <chr>  <chr>    <chr>    <dbl> <fct>                     <int> <int> <ord>
## 1 strea… No_Pand… Pande… 0.154   Moderate Arousal-Potenti…   250   275 small
## 2 strea… No_Pand… Pande… 0.0346  Higher Arousal-Potential…   268   255 small
## 3 strea… No_Pand… Pande… 0.00572 Higher Arousal-Potential…   257   228 small
## 4 strea… No_Pand… Pande… 0.149   Moderate Arousal-Potenti…   288   283 small
```

```
# ################ #
# Per DACH country #
# ################ #

# Dunn tests with holm correction per country for each cluster and both periods
Mood_cluster1 %>%
  group_by(mood_clust_fct, country) %>%
  dunn_test(streams ~ Pandemic) %>%
  adjust_pvalue(method = "holm")
```

```
## # A tibble: 12 x 11
##    country mood_clust_fct     .y.    group1 group2    n1    n2 statistic       p
##  * <chr>   <fct>              <chr>  <chr>  <chr>  <int> <int>     <dbl>   <dbl>
##  1 AT      Moderate Arousal-… strea… No_Pa… Pande…    28    24    0.973  3.31e-1
##  2 CH      Moderate Arousal-… strea… No_Pa… Pande…    68    45    0.293  7.69e-1
##  3 DE      Moderate Arousal-… strea… No_Pa… Pande…   154   206    2.33   1.98e-2
##  4 AT      Higher Arousal-Po… strea… No_Pa… Pande…    15    16    0.553  5.80e-1
##  5 CH      Higher Arousal-Po… strea… No_Pa… Pande…    79    63   -2.87   4.16e-3
##  6 DE      Higher Arousal-Po… strea… No_Pa… Pande…   174   176    2.12   3.40e-2
##  7 AT      Higher Arousal-Po… strea… No_Pa… Pande…    36    23    0.0389 9.69e-1
##  8 CH      Higher Arousal-Po… strea… No_Pa… Pande…    48    59   -2.05   3.99e-2
##  9 DE      Higher Arousal-Po… strea… No_Pa… Pande…   173   146    1.08   2.80e-1
## 10 AT      Moderate Arousal-… strea… No_Pa… Pande…    22    15    1.07   2.86e-1
## 11 CH      Moderate Arousal-… strea… No_Pa… Pande…    76    51   -3.70   2.17e-4
## 12 DE      Moderate Arousal-… strea… No_Pa… Pande…   190   217    3.75   1.79e-4
## # … with 2 more variables: p.adj <dbl>, p.adj.signif <chr>
```

```
# Effect sizes for the differences from above by using the z-values of the dunn
# tests via rstatix::wilcox_effsize
Mood_cluster1 %>%
  group_by(mood_clust_fct, country) %>%
  wilcox_effsize(streams ~ Pandemic)
```

```
## # A tibble: 12 x 9
##    .y.    group1  group2 effsize country mood_clust_fct       n1    n2 magnitude
##  * <chr>  <chr>   <chr>    <dbl> <chr>   <fct>              <int> <int> <ord>
##  1 strea… No_Pan… Pande… 0.135   AT      Moderate Arousal…    28    24 small
##  2 strea… No_Pan… Pande… 0.0276  CH      Moderate Arousal…    68    45 small
##  3 strea… No_Pan… Pande… 0.123   DE      Moderate Arousal…   154   206 small
##  4 strea… No_Pan… Pande… 0.0994  AT      Higher Arousal-P…    15    16 small
##  5 strea… No_Pan… Pande… 0.241   CH      Higher Arousal-P…    79    63 small
##  6 strea… No_Pan… Pande… 0.113   DE      Higher Arousal-P…   174   176 small
##  7 strea… No_Pan… Pande… 0.00506 AT      Higher Arousal-P…    36    23 small
##  8 strea… No_Pan… Pande… 0.199   CH      Higher Arousal-P…    48    59 small
##  9 strea… No_Pan… Pande… 0.0604  DE      Higher Arousal-P…   173   146 small
## 10 strea… No_Pan… Pande… 0.175   AT      Moderate Arousal…    22    15 small
## 11 strea… No_Pan… Pande… 0.328   CH      Moderate Arousal…    76    51 moderate
## 12 strea… No_Pan… Pande… 0.186   DE      Moderate Arousal…   190   217 small
```

With these findings, we got evidence to support the first hypothesis (see article).

Building a Support Vector Machine Classifier

Next, the actual classification procedure follows. In other words, the following code chunks account for the second hypothesis (see article):

```
df_ml_full <- read_csv("https://raw.githubusercontent.com/KewKalustian/Spotify_COVID-19_DACH/main/Datasets/Overall/df_ml_full.csv")

# Selecting model variables

mod_df <- df_ml_full  %>%
  dplyr::select(Pandemic,
         country,
         track_id,
         stream_count_rescaled,
         chart_position_rescaled ,
         acousticness,
         speechiness,
         instrumentalness,
         liveness,
         duration_ms_rescaled,
         mood_clust_fct) %>%
  mutate(Pandemic = as.factor(Pandemic),
         country = as.factor(country),
         mood_clust_fct = as.factor(mood_clust_fct))

str(mod_df)
```

```
## tibble [115,198 × 11] (S3: tbl_df/tbl/data.frame)
##  $ Pandemic               : Factor w/ 2 levels "No_Pandemic",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ country                : Factor w/ 3 levels "AT","CH","DE": 3 3 3 3 3 3 3 3 3 3 ...
##  $ track_id               : chr [1:115198] "14f1n3XCW3fQhAODW0CwLK" "5iIgIxtMNeogKaFUfvuAbs" "4BD9yUEFI07WaDJ381DDMq" "0qzX2dAs1otoAABKNHUwRb" ...
##  $ stream_count_rescaled  : num [1:115198] 0.577 0.232 0.225 0.177 0.166 ...
##  $ chart_position_rescaled: num [1:115198] 1 0.995 0.99 0.985 0.98 ...
##  $ acousticness           : num [1:115198] 0.0662 0.217 0.0519 0.37 0.159 0.488 0.387 0.519 0.379 0.592 ...
##  $ speechiness            : num [1:115198] 0.0693 0.32 0.264 0.326 0.166 0.273 0.318 0.125 0.298 0.334 ...
##  $ instrumentalness       : num [1:115198] 3.81e-06 3.38e-05 2.23e-06 0.00 1.86e-04 8.28e-05 0.00 1.97e-05 0.00 0.00 ...
##  $ liveness               : num [1:115198] 0.0858 0.0957 0.11 0.149 0.0848 0.0923 0.153 0.108 0.131 0.0881 ...
##  $ duration_ms_rescaled   : num [1:115198] 0.264 0.242 0.438 0.236 0.294 ...
##  $ mood_clust_fct         : Factor w/ 4 levels "1","2","3","4": 4 4 3 3 2 4 2 1 3 4 ...
```

```
#############
### MODEL ###
#############

set.seed(1, sample.kind = "Rounding")

# 80/20 split

test_index <- createDataPartition(y = mod_df$Pandemic, times = 1, p = .2,
                                  list = F)

DACH_train <- mod_df[-test_index,]

temp <- mod_df[test_index,]

# Making sure that track_id and countries in the test set are also in train set

DACH_test <- temp %>%
  semi_join(DACH_train, by = "track_id") %>%
  semi_join(DACH_train, by = "country")

# Adding rows that are removed from the test set back into train set
removed <- anti_join(temp, DACH_test)
DACH_train <- rbind(DACH_train, removed)

str(DACH_test)
```

```
## tibble [22,969 × 11] (S3: tbl_df/tbl/data.frame)
##  $ Pandemic               : Factor w/ 2 levels "No_Pandemic",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ country                : Factor w/ 3 levels "AT","CH","DE": 3 3 3 3 3 3 3 3 3 3 ...
##  $ track_id               : chr [1:22969] "3gs4lX2JSAFc8v8piogE3i" "25sgk305KZfyuqVBQIahi
m" "61A9J4906Zkuqd22oasZVR" "4xguhUOQ0YeguPC9Q3JSfz" ...
##  $ stream_count_rescaled  : num [1:22969] 0.1649 0.1195 0.0832 0.0821 0.0804 ...
##  $ chart_position_rescaled: num [1:22969] 0.975 0.95 0.844 0.834 0.829 ...
##  $ acousticness           : num [1:22969] 0.488 0.0691 0.0565 0.376 0.296 0.371 0.135 0.0
602 0.121 0.112 ...
##  $ speechiness            : num [1:22969] 0.273 0.0476 0.217 0.303 0.147 0.0308 0.16 0.05
26 0.0666 0.0882 ...
##  $ instrumentalness       : num [1:22969] 8.28e-05 0.00 1.23e-04 0.00 3.01e-06 0.00 0.00
1.12e-06 7.08e-06 1.48e-03 ...
##  $ liveness               : num [1:22969] 0.0923 0.166 0.0971 0.159 0.0793 0.231 0.152 0.
704 0.103 0.0873 ...
##  $ duration_ms_rescaled   : num [1:22969] 0.271 0.295 0.302 0.289 0.276 ...
##  $ mood_clust_fct         : Factor w/ 4 levels "1","2","3","4": 4 3 3 3 1 1 3 2 3 2 ...
```

```
str(DACH_train)
```

```
## tibble [92,229 × 11] (S3: tbl_df/tbl/data.frame)
##  $ Pandemic               : Factor w/ 2 levels "No_Pandemic",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ country                : Factor w/ 3 levels "AT","CH","DE": 3 3 3 3 3 3 3 3 3 3 ...
##  $ track_id               : chr [1:92229] "14f1n3XCW3fQhAODW0CwLK" "5iIgIxtMNeogKaFUfvuAbs" "4BD9yUEFI07WaDJ381DDMq" "0qzX2dAs1otoAABKNHUwRb" ...
##  $ stream_count_rescaled  : num [1:92229] 0.577 0.232 0.225 0.177 0.166 ...
##  $ chart_position_rescaled: num [1:92229] 1 0.995 0.99 0.985 0.98 ...
##  $ acousticness           : num [1:92229] 0.0662 0.217 0.0519 0.37 0.159 0.387 0.519 0.379 0.592 0.236 ...
##  $ speechiness            : num [1:92229] 0.0693 0.32 0.264 0.326 0.166 0.318 0.125 0.298 0.334 0.24 ...
##  $ instrumentalness       : num [1:92229] 3.81e-06 3.38e-05 2.23e-06 0.00 1.86e-04 0.00 1.97e-05 0.00 0.00 0.00 ...
##  $ liveness               : num [1:92229] 0.0858 0.0957 0.11 0.149 0.0848 0.153 0.108 0.131 0.0881 0.165 ...
##  $ duration_ms_rescaled   : num [1:92229] 0.264 0.242 0.438 0.236 0.294 ...
##  $ mood_clust_fct         : Factor w/ 4 levels "1","2","3","4": 4 4 3 3 2 2 1 3 4 2 ...
```

Cross-Validation

To find the optimal values for the hyperparameters, we need to run a cross-validation:

```
# ######### #
# 5-fold CV #
# ######### #

set.seed(3271, sample.kind = "Rounding")

# Only 20% of the data for cv due to computational costs.
train <- sample_frac(DACH_train, 0.2)

parallelStartSocket(cpus = detectCores())

set.seed(42, sample.kind = "Rounding")
hyperparams <-  tune(svm, Pandemic ~ .,
                     data = train[-3],
                     kernel = "radial", ranges = list(cost = 10^(-1:4),
                                                      gamma = c(0.5,1:5)),
                     tunecontrol = tune.control(cross = 5))
```

```
hyperparams$best.performance
```

```
## [1] 0.07015075
```

```
hyperparams$performances
```

```
##     cost gamma      error  dispersion
## 1  1e-01   0.5 0.25945985 0.010421101
## 2  1e+00   0.5 0.16209506 0.011114359
## 3  1e+01   0.5 0.10088939 0.005191021
## 4  1e+02   0.5 0.07622277 0.005727235
## 5  1e+03   0.5 0.07871630 0.002796600
## 6  1e+04   0.5 0.08375803 0.003498716
## 7  1e-01   1.0 0.20882530 0.012691100
## 8  1e+00   1.0 0.11802037 0.005683355
## 9  1e+01   1.0 0.07730694 0.004139774
## 10 1e+02   1.0 0.07286138 0.004697817
## 11 1e+03   1.0 0.07448787 0.004422128
## 12 1e+04   1.0 0.08104775 0.006724816
## 13 1e-01   2.0 0.15461337 0.010447125
## 14 1e+00   2.0 0.09075142 0.003356615
## 15 1e+01   2.0 0.07015075 0.003609469
## 16 1e+02   2.0 0.07085542 0.003748031
## 17 1e+03   2.0 0.07150604 0.003181710
## 18 1e+04   2.0 0.07600588 0.005950648
## 19 1e-01   3.0 0.13233191 0.008688274
## 20 1e+00   3.0 0.08337861 0.004247951
## 21 1e+01   3.0 0.07139766 0.005196293
## 22 1e+02   3.0 0.07226503 0.003740400
## 23 1e+03   3.0 0.07248189 0.004972032
## 24 1e+04   3.0 0.07437956 0.005727803
## 25 1e-01   4.0 0.12197730 0.008140839
## 26 1e+00   4.0 0.08028847 0.003464767
## 27 1e+01   4.0 0.07237340 0.004856462
## 28 1e+02   4.0 0.07362028 0.004770858
## 29 1e+03   4.0 0.07242767 0.005191767
## 30 1e+04   4.0 0.07486735 0.006062073
## 31 1e-01   5.0 0.12430760 0.014488884
## 32 1e+00   5.0 0.07996317 0.004998991
## 33 1e+01   5.0 0.07416237 0.005316643
## 34 1e+02   5.0 0.07508409 0.005364631
## 35 1e+03   5.0 0.07448775 0.005244750
## 36 1e+04   5.0 0.07622261 0.005270354
```

```
hyperparams$best.parameters
```

```
##    cost gamma
## 15   10     2
```

Model Training / Testing

The actual model training and testing runs as follows:

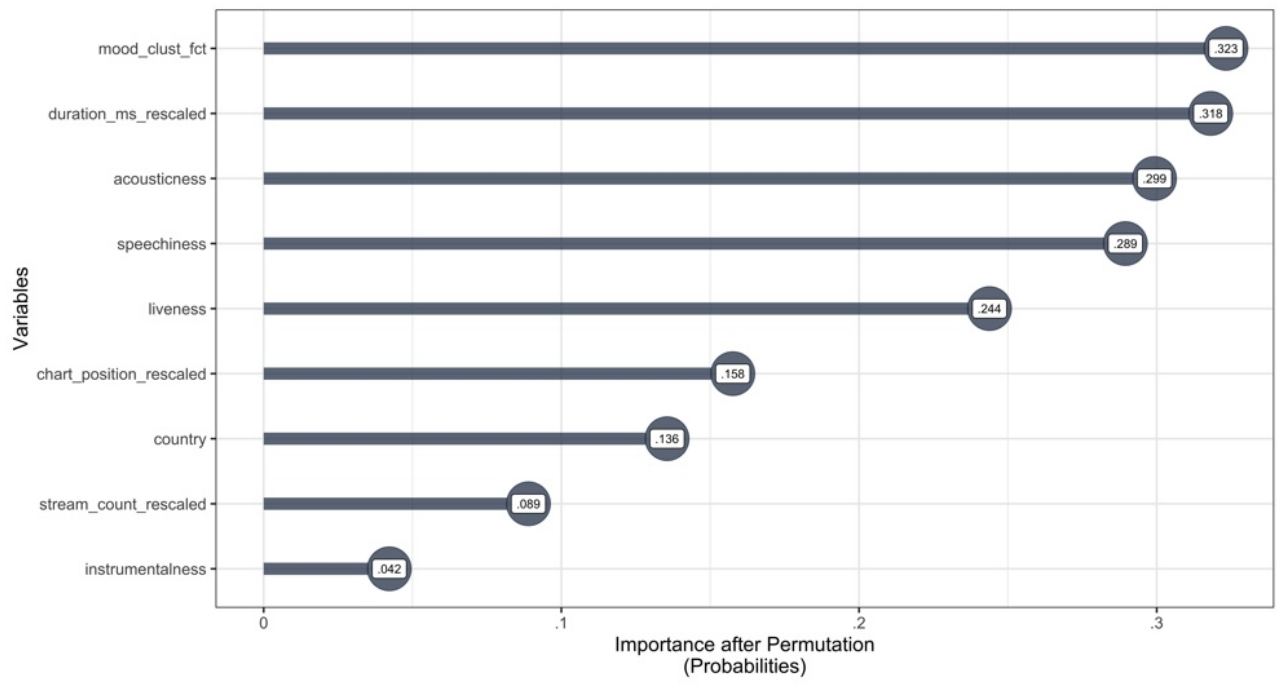
```
# ############## #
# Model training #
# ############## #

set.seed(1, sample.kind = "Rounding")

svm_fit <-  svm(Pandemic ~ ., DACH_train[,-3],
                kernel = "radial",
                cost = 10,              # see hyperparams
                gamma = 2,              # see hyperparams
                scale = T,
                probability = T)

# ############# #
# Test scenario #
# ############# #

prd <- predict(svm_fit, DACH_test[,-3], probability = T)

# ########### #
# Performance #
# ########### #

caret::confusionMatrix(prd, DACH_test$Pandemic, mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    No_Pandemic Pandemic
##   No_Pandemic       11236      238
##   Pandemic            251    11244
##
##                Accuracy : 0.9787
##                  95% CI : (0.9768, 0.9805)
##     No Information Rate : 0.5001
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9574
##
##  Mcnemar's Test P-Value : 0.5874
##
##               Precision : 0.9793
##                  Recall : 0.9781
##                      F1 : 0.9787
##              Prevalence : 0.5001
##          Detection Rate : 0.4892
##    Detection Prevalence : 0.4995
##       Balanced Accuracy : 0.9787
##
##        'Positive' Class : No_Pandemic
##
```

Interpretable Machine Learning

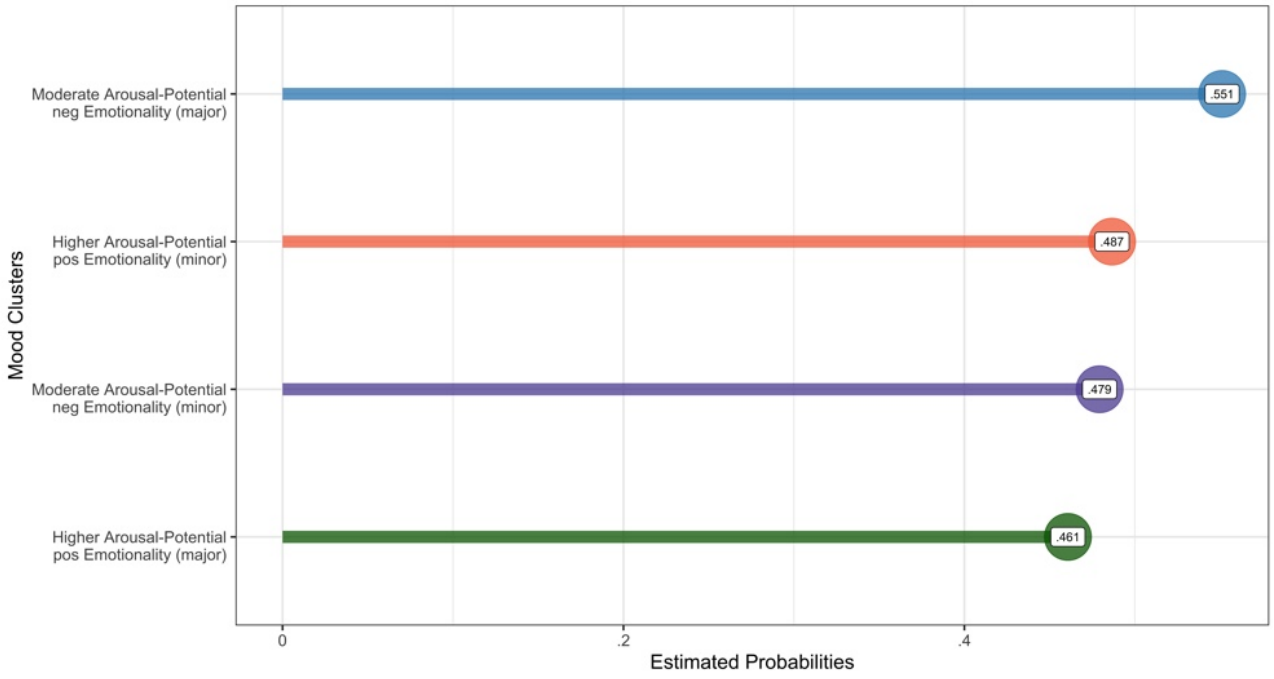
To get an overview of the different variable importance of the built model, we can extract and plot this importance as follows:

```
# ################### #
# Variable Importance #
# ################### #

# Pred/classification function
prob_no_pan <- function(object, newdata) {
  res <- as.vector(predict(object, newdata = newdata))
  return(res)}

set.seed(2827)  # for reproducibility

imp <-vip(svm_fit,
          train = as.data.frame(DACH_train[,-3]),
          method = "permute",
          target = "Pandemic",
          metric = "auc",
          nsim = 5,
          pred_wrapper =  prob_no_pan,
          reference_class = "Pandemic",
          all_permutations = T)

# Extracting values
imp_values <- imp$data

# Plotting

ggplot(data= imp_values, aes( x = reorder(Variable, Importance), y = Importance,
                              label = round(Importance,3) )) +
  geom_segment(aes(x=reorder(Variable, Importance), xend=reorder(Variable, Importance),
                   y=0, yend = Importance-0.007), color = "#2e4057", size =4, alpha=.75)+
  geom_point(size= 14, shape = 21, color = "#2e4057", fill = "#2e4057", alpha = 0.75) +
  geom_label(nudge_y = 0, nudge_x = 0, size = 3)+
  labs(y = "Importance after Permutation\n(Probabilities)\n",  x ="\nVariables")+
  layout+
  coord_flip()
```



To even improve interpretability, a closer look at the partial dependence of the most important variable `mood_clust_fct` according to the variable importance plot is helpful:

```
# ################## #
# Partial Dependence #
# ################## #

# Pred/classification function
custom_pred <- function(object, newdata) {
  res <- as.vector(predict(object, data.frame(newdata), type = "prob"))[,2]
  return(res)}

# Extracting values
pd_values <- pdp::partial(svm_fit, train = as.data.frame(DACH_train[,-3]),
                          pred.var = "mood_clust_fct", prob = TRUE,
                          which.class = "Pandemic")
# Plotting

ggplot(data = pd_values, aes(reorder(mood_clust_fct, yhat), yhat,
                             label = round(yhat,3)))+
  geom_segment(aes(x=reorder(mood_clust_fct, yhat),
                   xend=reorder(mood_clust_fct, yhat), y=0,
                   yend=yhat), color= I(cols), size = 4, alpha=.75)+
  geom_point(size= 15, shape = 21, color = I(cols), fill = I(cols),
             alpha = .75) +
  geom_label(nudge_y = 0, nudge_x = 0, size = 3)+
  scale_x_discrete(limits= c("3","4","2","1"),
                   labels= c("\nHigher Arousal-Potential\npos Emotionality (major)",
                             "\nModerate Arousal-Potential\nneg Emotionality (minor)",
                             "\nHigher Arousal-Potential\npos Emotionality (minor)",
                             "\nModerate Arousal-Potential\nneg Emotionality (major)"))+
  labs(x="\nMood Clusters", y="Estimated Probabilities\n")+
  coord_flip()+
  layout

parallelStop()

# ### #
# END #
# ### #
```



With this, the entire script as well as datasets that have been used for the analyses in the article are here in this *Codebook* freely accessible. Additionally, the entire scripts and datasets can be found on this dedicated Github repository (https://github.com/KewKalustian/Spotify_COVID-19_DACH), too.

Session Info

Information on the computing environment:

```
## $version
## [1] "R version 4.1.0 (2021-05-18)"
##
## $os
## [1] "macOS Big Sur 10.16"
##
## $system
## [1] "x86_64, darwin17.0"
##
## $ui
## [1] "X11"
##
## $language
## [1] "(EN)"
##
## $collate
## [1] "en_US.UTF-8"
```

```
##  package          * version    date       lib source
##  abind              1.4-5      2016-07-21 [1] CRAN (R 4.1.0)
##  assertthat         0.2.1      2019-03-21 [1] CRAN (R 4.1.0)
##  backports          1.2.1      2020-12-09 [1] CRAN (R 4.1.0)
##  BBmisc             1.11       2017-03-10 [1] CRAN (R 4.1.0)
##  beeswarm           0.4.0      2021-06-01 [1] CRAN (R 4.1.0)
##  broom              0.7.6      2021-04-05 [1] CRAN (R 4.1.0)
##  bslib              0.2.5.1    2021-05-18 [1] CRAN (R 4.1.0)
##  car              * 3.0-10     2020-09-29 [1] CRAN (R 4.1.0)
##  carData          * 3.0-4      2020-05-22 [1] CRAN (R 4.1.0)
##  caret            * 6.0-88     2021-05-15 [1] CRAN (R 4.1.0)
##  cellranger         1.1.0      2016-07-27 [1] CRAN (R 4.1.0)
##  checkmate          2.0.0      2020-02-06 [1] CRAN (R 4.1.0)
##  class              7.3-19     2021-05-03 [1] CRAN (R 4.1.0)
##  cli                3.0.0      2021-06-30 [1] CRAN (R 4.1.0)
##  codetools          0.2-18     2020-11-04 [1] CRAN (R 4.1.0)
##  coin             * 1.4-1      2021-02-08 [1] CRAN (R 4.1.0)
##  colorspace         2.0-2      2021-06-24 [1] CRAN (R 4.1.0)
##  cowplot            1.1.1      2020-12-30 [1] CRAN (R 4.1.0)
##  crayon             1.4.1      2021-02-08 [1] CRAN (R 4.1.0)
##  crosstalk          1.1.1      2021-01-12 [1] CRAN (R 4.1.0)
##  curl               4.3.2      2021-06-23 [1] CRAN (R 4.1.0)
##  data.table         1.14.0     2021-02-21 [1] CRAN (R 4.1.0)
##  DBI                1.1.1      2021-01-15 [1] CRAN (R 4.1.0)
##  dbplyr             2.1.1      2021-04-06 [1] CRAN (R 4.1.0)
##  digest             0.6.27     2020-10-24 [1] CRAN (R 4.1.0)
##  dplyr            * 1.0.7      2021-06-18 [1] CRAN (R 4.1.0)
##  e1071            * 1.7-7      2021-05-23 [1] CRAN (R 4.1.0)
##  ellipsis           0.3.2      2021-04-29 [1] CRAN (R 4.1.0)
##  evaluate           0.14       2019-05-28 [1] CRAN (R 4.1.0)
##  extrafont          0.17       2014-12-08 [1] CRAN (R 4.1.0)
##  extrafontdb        1.0        2012-06-11 [1] CRAN (R 4.1.0)
##  factoextra       * 1.0.7      2020-04-01 [1] CRAN (R 4.1.0)
##  fansi              0.5.0      2021-05-25 [1] CRAN (R 4.1.0)
##  farver             2.1.0      2021-02-28 [1] CRAN (R 4.1.0)
##  fastmap            1.1.0      2021-01-25 [1] CRAN (R 4.1.0)
##  forcats          * 0.5.1      2021-01-27 [1] CRAN (R 4.1.0)
##  foreach            1.5.1      2020-10-15 [1] CRAN (R 4.1.0)
##  foreign            0.8-81     2020-12-22 [1] CRAN (R 4.1.0)
##  fs                 1.5.0      2020-07-31 [1] CRAN (R 4.1.0)
##  generics           0.1.0      2020-10-31 [1] CRAN (R 4.1.0)
##  genius             2.2.2      2020-05-28 [1] CRAN (R 4.1.0)
##  ggbeeswarm       * 0.6.0      2017-08-07 [1] CRAN (R 4.1.0)
##  ggplot2          * 3.3.5      2021-06-25 [1] CRAN (R 4.1.0)
##  ggpubr           * 0.4.0      2020-06-27 [1] CRAN (R 4.1.0)
##  ggrepel            0.9.1      2021-01-15 [1] CRAN (R 4.1.0)
##  ggsignif           0.6.2      2021-06-14 [1] CRAN (R 4.1.0)
##  glue               1.4.2      2020-08-27 [1] CRAN (R 4.1.0)
##  gower              0.2.2      2020-06-23 [1] CRAN (R 4.1.0)
##  gridExtra          2.3        2017-09-09 [1] CRAN (R 4.1.0)
##  gtable             0.3.0      2019-03-25 [1] CRAN (R 4.1.0)
##  haven              2.4.1      2021-04-23 [1] CRAN (R 4.1.0)
##  highr              0.9        2021-04-16 [1] CRAN (R 4.1.0)
##  hms                1.1.0      2021-05-17 [1] CRAN (R 4.1.0)
##  htmltools          0.5.1.1    2021-01-22 [1] CRAN (R 4.1.0)
##  htmlwidgets        1.5.3      2020-12-10 [1] CRAN (R 4.1.0)
##  httpuv             1.6.1      2021-05-07 [1] CRAN (R 4.1.0)
##  httr               1.4.2      2020-07-20 [1] CRAN (R 4.1.0)
##  ipred              0.9-11     2021-03-12 [1] CRAN (R 4.1.0)
##  iterators          1.0.13     2020-10-15 [1] CRAN (R 4.1.0)
##  janeaustenr        0.1.5      2017-06-10 [1] CRAN (R 4.1.0)
##  janitor            2.1.0      2021-01-05 [1] CRAN (R 4.1.0)
```

```
##  jquerylib          0.1.4      2021-04-26 [1] CRAN (R 4.1.0)
##  jsonlite           1.7.2      2020-12-09 [1] CRAN (R 4.1.0)
##  klippy             0.0.0.9500 2021-07-24 [1] Github (rlesur/klippy@378c247)
##  knitr              1.33       2021-04-24 [1] CRAN (R 4.1.0)
##  later              1.2.0      2021-04-23 [1] CRAN (R 4.1.0)
##  lattice          * 0.20-44    2021-05-02 [1] CRAN (R 4.1.0)
##  lava               1.6.9      2021-03-11 [1] CRAN (R 4.1.0)
##  libcoin            1.0-8      2021-02-08 [1] CRAN (R 4.1.0)
##  lifecycle          1.0.0      2021-02-15 [1] CRAN (R 4.1.0)
##  lubridate        * 1.7.10     2021-02-26 [1] CRAN (R 4.1.0)
##  magrittr         * 2.0.1      2020-11-17 [1] CRAN (R 4.1.0)
##  manipulateWidget   0.11.0     2021-05-31 [1] CRAN (R 4.1.0)
##  MASS               7.3-54     2021-05-03 [1] CRAN (R 4.1.0)
##  Matrix             1.3-3      2021-05-04 [1] CRAN (R 4.1.0)
##  matrixStats        0.59.0     2021-06-01 [1] CRAN (R 4.1.0)
##  mime               0.11       2021-06-23 [1] CRAN (R 4.1.0)
##  miniUI             0.1.1.1    2018-05-18 [1] CRAN (R 4.1.0)
##  ModelMetrics     * 1.2.2.2    2020-03-17 [1] CRAN (R 4.1.0)
##  modelr             0.1.8      2020-05-19 [1] CRAN (R 4.1.0)
##  modeltools         0.2-23     2020-03-05 [1] CRAN (R 4.1.0)
##  multcomp           1.4-17     2021-04-29 [1] CRAN (R 4.1.0)
##  munsell            0.5.0      2018-06-12 [1] CRAN (R 4.1.0)
##  mvtnorm            1.1-2      2021-06-07 [1] CRAN (R 4.1.0)
##  nlme               3.1-152    2021-02-04 [1] CRAN (R 4.1.0)
##  nnet               7.3-16     2021-05-03 [1] CRAN (R 4.1.0)
##  openxlsx           4.2.3      2020-10-27 [1] CRAN (R 4.1.0)
##  pacman           * 0.5.1      2019-03-11 [1] CRAN (R 4.1.0)
##  parallelMap      * 1.5.1      2021-06-28 [1] CRAN (R 4.1.0)
##  pdp              * 0.7.0      2018-08-27 [1] CRAN (R 4.1.0)
##  pillar             1.6.1      2021-05-16 [1] CRAN (R 4.1.0)
##  pkgconfig          2.0.3      2019-09-22 [1] CRAN (R 4.1.0)
##  plyr               1.8.6      2020-03-03 [1] CRAN (R 4.1.0)
##  pROC               1.17.0.1   2021-01-13 [1] CRAN (R 4.1.0)
##  prodlim            2019.11.13 2019-11-17 [1] CRAN (R 4.1.0)
##  promises           1.2.0.1    2021-02-11 [1] CRAN (R 4.1.0)
##  proxy              0.4-26     2021-06-07 [1] CRAN (R 4.1.0)
##  purrr            * 0.3.4      2020-04-17 [1] CRAN (R 4.1.0)
##  R6                 2.5.0      2020-10-28 [1] CRAN (R 4.1.0)
##  Rcpp               1.0.6      2021-01-15 [1] CRAN (R 4.1.0)
##  readr            * 1.4.0      2020-10-05 [1] CRAN (R 4.1.0)
##  readxl             1.3.1      2019-03-13 [1] CRAN (R 4.1.0)
##  recipes            0.1.16     2021-04-16 [1] CRAN (R 4.1.0)
##  reprex             2.0.0      2021-04-02 [1] CRAN (R 4.1.0)
##  reshape2           1.4.4      2020-04-09 [1] CRAN (R 4.1.0)
##  rgl              * 0.106.8    2021-04-23 [1] CRAN (R 4.1.0)
##  rio                0.5.26     2021-03-01 [1] CRAN (R 4.1.0)
##  rlang              0.4.11     2021-04-30 [1] CRAN (R 4.1.0)
##  rmarkdown          2.9        2021-06-15 [1] CRAN (R 4.1.0)
##  rpart              4.1-15     2019-04-12 [1] CRAN (R 4.1.0)
##  rstatix          * 0.7.0      2021-02-13 [1] CRAN (R 4.1.0)
##  rstudioapi         0.13       2020-11-12 [1] CRAN (R 4.1.0)
##  Rttf2pt1           1.3.8      2020-01-10 [1] CRAN (R 4.1.0)
##  rvest            * 1.0.0      2021-03-09 [1] CRAN (R 4.1.0)
##  sandwich           3.0-1      2021-05-18 [1] CRAN (R 4.1.0)
##  sass               0.4.0      2021-05-12 [1] CRAN (R 4.1.0)
##  scales           * 1.1.1      2020-05-11 [1] CRAN (R 4.1.0)
##  sessioninfo        1.1.1      2018-11-05 [1] CRAN (R 4.1.0)
##  shiny              1.6.0      2021-01-25 [1] CRAN (R 4.1.0)
##  snakecase          0.11.0     2019-05-25 [1] CRAN (R 4.1.0)
##  SnowballC          0.7.0      2020-04-01 [1] CRAN (R 4.1.0)
##  spotifyr         * 2.2.1      2021-06-17 [1] CRAN (R 4.1.0)
##  stringi            1.7.3      2021-07-16 [1] CRAN (R 4.1.0)
##  stringr          * 1.4.0      2019-02-10 [1] CRAN (R 4.1.0)
```

```
##  survival         * 3.2-11     2021-04-26 [1] CRAN (R 4.1.0)
##  TH.data            1.0-10     2019-01-21 [1] CRAN (R 4.1.0)
##  tibble           * 3.1.2      2021-05-16 [1] CRAN (R 4.1.0)
##  tidyr            * 1.1.3      2021-03-03 [1] CRAN (R 4.1.0)
##  tidyselect         1.1.1      2021-04-30 [1] CRAN (R 4.1.0)
##  tidytext           0.3.1      2021-04-10 [1] CRAN (R 4.1.0)
##  tidyverse        * 1.3.1      2021-04-15 [1] CRAN (R 4.1.0)
##  timeDate           3043.102   2018-02-21 [1] CRAN (R 4.1.0)
##  tokenizers         0.2.1      2018-03-29 [1] CRAN (R 4.1.0)
##  utf8               1.2.1      2021-03-12 [1] CRAN (R 4.1.0)
##  vctrs              0.3.8      2021-04-29 [1] CRAN (R 4.1.0)
##  vip              * 0.3.2      2020-12-17 [1] CRAN (R 4.1.0)
##  vipor              0.4.5      2017-03-22 [1] CRAN (R 4.1.0)
##  withr              2.4.2      2021-04-18 [1] CRAN (R 4.1.0)
##  xfun               0.24       2021-06-15 [1] CRAN (R 4.1.0)
##  xml2               1.3.2      2020-04-23 [1] CRAN (R 4.1.0)
##  xtable             1.8-4      2019-04-21 [1] CRAN (R 4.1.0)
##  yaml               2.2.1      2020-02-01 [1] CRAN (R 4.1.0)
##  zip                2.2.0      2021-05-31 [1] CRAN (R 4.1.0)
##  zoo                1.8-9      2021-03-09 [1] CRAN (R 4.1.0)
##
## [1] /Library/Frameworks/R.framework/Versions/4.1/Resources/library
```