Data Wrangling in R

Erica Bishop

2023-02-24

Data Wrangling in R Examples

The code below is excerpted from a machine learning lab in which my goal was to build four different types of classification models to predict whether a song belonged to my saved track list on Spotify or my lab partner's saved track list. The original work is saved in the labg5__get__spotify__data.R script and the lab5.Rmd markdown file within this repository if you'd like to check out the resulting models and accuracy comparisons. Metadata and variable definitions are available here.

The spotify data wrangling presented a unique challenge because the API requests are limited to 50 rows at a time, so although the data was pretty clean out of the gate in terms of missing data and variable consistency, it required some manipulation to join and process for modeling. I've included a brief demonstration of how I've used the `janitor` and `lubridate` packages to work with data that was messy in other ways at the end of this document

Preparing API data for classification models

Step 1: Request my data from the Spotify API

```r
###Get access token
#access and edit .renviron with usethis::edit_r_environ
access_token <- get_spotify_access_token(client_id = Sys.getenv("SPOTIFY_CLIENT_ID"), #client secret an
                                          client_secret = Sys.getenv("SPOTIFY_CLIENT_SECRET"))

###Request saved track list
offsets_list <- c(seq(0, 2900, 50)) #create list to get saved songs

saved_tracks_eb <- lapply(X = offsets_list,
                          FUN = get_my_saved_tracks,
                          limit = 50 #produce list of dfs for each call of 50
) |>
  bind_rows() #bind lists into one df

#Request saved track attributes
track_id_list <- saved_tracks_eb$track.id #create list of track ids

track_attributes_eb <- lapply(X = track_id_list, #get audio features for each track id
                              FUN = get_track_audio_features) |>
  bind_rows() #combine resulting list of dfs
```

Step 2: Wrangle my data into a usable format for my lab partner (Jillian)

```
#append track list name to the track attributes df
saved_track_names <- saved_tracks_eb |>
  select(track.name, track.id) |> #select out just track name and id
  rename(id = track.id) #rename id column to join

eb_tracks_df <- left_join(track_attributes_eb, saved_track_names, by = "id") #join df

#add user column
eb_tracks_df <- eb_tracks_df |>
  add_column(user = "Erica")

#save dataframe as a csv to send to Jillian
write_csv(eb_tracks_df, here("eb_track_attributes.csv"))
```

Step 3: Load my lab partner's data

```
#read in Jillian's data
ja_tracks_df <- read_csv(here("jillian_spotify_data.csv")) |> #saved to repo for ease of re-running
  select(-track_id) |> #drop track_id column (repeats id column)
  rename(track.name = track_name) #renametrack_name to match
```

```
## Rows: 931 Columns: 21
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): type, id, uri, track_href, analysis_url, track_name, track_id, user
## 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.
```

Step 4: Join data and investigate overlaps

For this lab, I decided to create a third classification for songs that were in both my and my partner's saved track lists.

```
#take a closer look at variables and see if the names match
names(eb_tracks_df)
```

```
##  [1] "danceability"     "energy"           "key"              "loudness"
##  [5] "mode"             "speechiness"      "acousticness"     "instrumentalness"
##  [9] "liveness"         "valence"          "tempo"            "type"
## [13] "id"               "uri"              "track_href"       "analysis_url"
## [17] "duration_ms"      "time_signature"   "track.name"       "user"
```

```
names(ja_tracks_df)
```

```
##  [1] "danceability"     "energy"           "key"              "loudness"
##  [5] "mode"             "speechiness"      "acousticness"     "instrumentalness"
##  [9] "liveness"         "valence"          "tempo"            "type"
## [13] "id"               "uri"              "track_href"       "analysis_url"
## [17] "duration_ms"      "time_signature"   "track.name"       "user"
```

```
#Investigate the overlap of saved songs
eb_ja_shared_faves <- semi_join(eb_tracks_df, ja_tracks_df, by = "id") #use semi_join to get df of trac
#Over 100 tracks in common - worth creating a third class
```

```
#Combine both datasets
eb_ja_tracks <- full_join(eb_tracks_df,
                          ja_tracks_df,
                          by = c("danceability", "energy", "key", "loudness", #joining by all variables
                                 "mode", "speechiness", "acousticness", "instrumentalness",
                                 "liveness", "valence", "tempo", "type", "id", "uri", "track_href",
                                 "analysis_url", "duration_ms", "time_signature", "track.name")) |>
  mutate(user = case_when(
    user.y == "Jillian" & user.x == "Erica" ~ "Both", #relabel songs in both user lists with case_when
    user.y == "Jillian" & is.na(user.x) ~ "Jillian",
    TRUE ~ "Erica"
  )) |>  #drop individual user columns and a few other less useful predictors
  select(-c(user.x, user.y, uri, track_href, analysis_url, type))

#save output df as a csv
write_csv(eb_ja_tracks, here("eb_ja_tracks.csv"))
```

Step 5: Pre-process data for modeling (encoding and standardizing)

```
#remove unique identifiers from dataset (track id and song name)
tracks_df <- eb_ja_tracks |>
  select(-c(id, track.name)) |>
  mutate(user = as.factor(user)) #make outcome variable a factor

#split the data FIRST
set.seed(123) #set seed for reproducibility

tracks_split <- initial_split(tracks_df) #split
tracks_train <- training(tracks_split) #training dataset
tracks_test <- testing(tracks_split) #testing dataset

#Preprocess the data for tidymodels modeling - encoding with recipe
tracks_recipe <- recipe(user ~., data = tracks_train) |>
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) |> #one hot encoding of nominal variables
  step_normalize(all_numeric(), -all_outcomes()) |> #normalize scale of numeric variables
  prep()
```

Cleaning messy data with `Janitor`, `Lubridate`, and `zoo`

For my final project in my statistics course, I wanted to examine the relationship between the number of electric vehicles registered and the ground-level ozone pollution in the Denver Metro area. A full write-up on this project can be accessed through my blog post.

The EV data was downloaded from https://www.atlasevhub.com/materials/state-ev-registration-data/#data - zip codes are for some reason in multiple different formats and lengths - over 1300 unique zips (should be no more than 500 or so) - date range: January 2010 - July 2022 - includes fuel cell, plug in hybrid, and EV - NOT using cumulative counts becuase this dataset includes original registrations and renewals without distinction, so according the the recommendations from the data source, treating like a snapshot in time

The air quality data is from the EPA's Outdoor Air Quality Data, downloadable here. - Includes Daily AQI, Daily Ozone, and Annual AQI data by county

```
#set up data directory
datadir <- ("/Users/ericabishop/Documents/MEDS-fall-classwork/EDS222-stats/final_project/data") #contai
```

```
figdir <- ("/Users/ericabishop/Documents/MEDS-fall-classwork/EDS222-stats/final_project/figs")#to save
```

Step 1: read-in and wrangle EV registration and county/zip code data

```
## EV Data

#read in EV data and format dates
co_evs <- read_csv(file.path(datadir, "co_ev_registrations_public.csv")) |>
  janitor::clean_names()
```

```
## Rows: 1246097 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): DMV Snapshot, State, Registration Valid Date, Registration Expirati...
## dbl (2): DMV ID, ZIP Code
##
## 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.
```

```
#format dates
co_evs <- co_evs |>
  mutate(registration_valid_date = lubridate::mdy(registration_valid_date),
         registration_expiration_date = lubridate::mdy(registration_expiration_date))

#read in county data to validate zips and match with EPA data, drop state column
co_counties <- read_csv(file.path(datadir, "CO_counties.csv")) |>
  clean_names() |>
  select("county", "zip_code") #validating zip codes by those that match a county
```

```
## Rows: 661 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): State, County
## dbl (1): ZIP Code
##
## 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.
```

```
#attach county names associated with zip code to co_evs
co_evs_counties <- left_join(co_evs, co_counties, by = "zip_code")

#rename date column to join
co_evs_counties <- co_evs_counties |>
  rename(date_local = registration_valid_date)


#manipulate variables of interest into one DF - BY COUNTY
#select columns of interest from EVs into new DF
evs_clean <- co_evs_counties |>
  select("county", "date_local", "zip_code") |>
  mutate(my_date = as.yearmon(date_local))
```

Step 2: Read in EPA ozone data and filter to Colorado

```
## OZONE Data

#read in daily ozone data
daily_ozone_2010 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2010.csv"))
```

```
## Rows: 378545 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2011 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2011.csv"))
```

```
## Rows: 383872 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2012 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2012.csv"))
```

```
## Rows: 386439 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2013 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2013.csv"))
```

```
## Rows: 389373 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
daily_ozone_2014 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2014.csv"))
```

```
## Rows: 388613 Columns: 29
## -- Column specification ---------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
daily_ozone_2015 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2015.csv"))
```

```
## Rows: 386404 Columns: 29
## -- Column specification ---------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
daily_ozone_2016 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2016.csv"))
```

```
## Rows: 392952 Columns: 29
## -- Column specification ---------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
daily_ozone_2017 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2017.csv"))
```

```
## Rows: 404217 Columns: 29
## -- Column specification ---------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
daily_ozone_2018 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2018.csv"))
```

```
## Rows: 402906 Columns: 29
## -- Column specification ---------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2019 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2019.csv"))
```

```
## Rows: 389651 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2020 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2020.csv"))
```

```
## Rows: 391845 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2021 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2021.csv"))
```

```
## Rows: 390562 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
## 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.
```

```
daily_ozone_2022 <- read_csv(file.path(datadir, "daily_summary_ozone", "daily_44201_2022.csv"))
```

```
## Rows: 233601 Columns: 29
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (16): State Code, County Code, Site Num, Datum, Parameter Name, Sample ...
## dbl  (10): Parameter Code, POC, Latitude, Longitude, Observation Count, Obse...
## lgl   (1): Method Code
## date  (2): Date Local, Date of Last Change
##
```

```
## 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.
```

```
#write function to filter data to just Colorado
filter_CO <- function(df) {
  df_filtered <- df |>
    filter(`State Name` == "Colorado") |>
    clean_names()
  return(df_filtered)
}

#filter to just CO
dozo_10_CO <- filter_CO(daily_ozone_2010)
dozo_11_CO <- filter_CO(daily_ozone_2011)
dozo_12_CO <- filter_CO(daily_ozone_2012)
dozo_13_CO <- filter_CO(daily_ozone_2013)
dozo_14_CO <- filter_CO(daily_ozone_2014)
dozo_15_CO <- filter_CO(daily_ozone_2015)
dozo_16_CO <- filter_CO(daily_ozone_2016)
dozo_17_CO <- filter_CO(daily_ozone_2017)
dozo_18_CO <- filter_CO(daily_ozone_2018)
dozo_19_CO <- filter_CO(daily_ozone_2019)
dozo_20_CO <- filter_CO(daily_ozone_2020)
dozo_21_CO <- filter_CO(daily_ozone_2021)
dozo_22_CO <- filter_CO(daily_ozone_2022)


#combine into one df with rbind
daily_ozo_CO <- rbind(dozo_10_CO, dozo_11_CO, dozo_12_CO, dozo_13_CO, dozo_14_CO, dozo_15_CO, dozo_16_C

#remove precursor datasets from memory to free up space
rm(dozo_10_CO, dozo_11_CO, dozo_12_CO, dozo_13_CO, dozo_14_CO, dozo_15_CO, dozo_16_CO, dozo_17_CO, dozo_
rm(daily_ozone_2010, daily_ozone_2011, daily_ozone_2012, daily_ozone_2013, daily_ozone_2014, daily_ozon
```

Step 3: Comnbine data and aggregate to monthly time scale

```
#select columns of interest from ozone into new DF
ozone_clean <- daily_ozo_CO |>
  select("date_local", "units_of_measure", "parameter_name", "aqi", "county_name", "local_site_name", "
  mutate(my_date = as.yearmon(date_local)) |>
  rename(county = county_name)

#aggregate both dfs monthly for analysis
evs_summary <- evs_clean |>
  group_by(my_date) |>
  summarize(ev_reg_count = n())

ozone_summary <- ozone_clean |>
  group_by(my_date) |>
  summarize(monthly_ozone_avg_ppm = mean(arithmetic_mean))

#combine summary tables into one dataframe of monthly/yearly (my) data
my_summary <- left_join(evs_summary, ozone_summary, by  = "my_date")
```