Data Processing

Erin M. Buchanan

Last Update 2022-03-09

Libraries

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(psych)
library(tidyr)
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```
library(rio)
library(RSQLite)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.6     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x ggplot2::%+%()   masks psych::%+%()
## x ggplot2::alpha() masks psych::alpha()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'

## The following object is masked from 'package:purrr':
##
##     flatten
```

```
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(widyr)
library(LexOPS)
current_year <- 2022
```

Functions

```r
# from labjs docs
processData <- function(database) {
  con <- dbConnect(
    drv=RSQLite::SQLite(),
    dbname=database
  )

  # Extract main table
  d <- dbGetQuery(
    conn=con,
    statement='SELECT * FROM labjs'
  )

  # Close connection
  dbDisconnect(
    conn=con
  )

  # Discard connection
  rm(con)

  d.meta <- map_dfr(d$metadata, fromJSON) %>%
    dplyr::rename(
      observation=id
    )

  d <- d %>%
    bind_cols(d.meta) %>%
    select(
      -metadata # Remove metadata column
    )

  # Remove temporary data frame
  rm(d.meta)

  count_unique <- function(x) {
    return(length(unique(x)))
  }

  information_preserved <- function(x, length) {
    return(
      count_unique(str_sub(x, end=i)) ==
        count_unique(x)
    )
  }

  # Figure out the length of the random ids needed
  # to preserve the information therein. (five characters
  # should usually be enougth, but better safe)
  for (i in 5:36) {
    if (
      information_preserved(d$session, i) &&
      information_preserved(d$observation, i)
    ) {
      break()
    }
  }

  d <- d %>%
    dplyr::mutate(
      session=str_sub(session, end=i),
      observation=str_sub(observation, end=i)
    )

  rm(i, count_unique, information_preserved)

  parseJSON <- function(input) {
    return(input %>%
             fromJSON(flatten=T) %>% {
               # Coerce lists
               if (class(.) == 'list') {
                 discard(., is.null) %>%
                   as_tibble()
               } else {
                 .
               } } %>%
             # Sanitize names
             janitor::clean_names() %>%
            # Use only strings for now, and re-encode types later
            mutate_all(as.character)
    )
  }

  d.full <- d %>%
    dplyr::filter(payload == 'full')

  if (nrow(d.full) > 0) {
    d.full %>%
      group_by(observation, id) %>%
      do(
        { map_dfr(.$data, parseJSON) } %>%
          bind_rows()
      ) %>%
      ungroup() %>%
      select(-id) -> d.full
  } else {
    # If there are no full datasets, start from an entirely empty df
    # in order to avoid introducing unwanted columns into the following
    # merge steps.
    d.full <- tibble()
  }

  d %>%
    dplyr::filter(payload %in% c('incremental', 'latest')) %>%
    group_by(observation, id) %>%
    do(
      { map_dfr(.$data, parseJSON) } %>%
        bind_rows()
    ) %>%
    ungroup() %>%
    select(-id) -> d.incremental

  if (nrow(d.full) > 0){

  d.output <- d.full %>%
    bind_rows(
      d.incremental %>% filter(!(observation %in% d.full$observation))
    ) %>%
    type_convert()

  } else {

    d.output <- d.incremental %>% type_convert()

  }

  d.output %>%
    group_by(observation) %>%
    fill(matches('code'), .direction='down') %>%
    fill(matches('code'), .direction='up') %>%
    ungroup() -> d.output

  return(d.output)
}
```

Data Processing Example

Please note this file has been updated based on the `sqlite` files exported from the pilot testing of the experiment. This file will also be updated when we have real data. We will exclude this pilot data at that point.

Note: after the pilot test, we will need to store these files in GitHub releases, as they will get very large pretty quickly … we likely will need to zip them as well.

```
# collected data
en_data_1 <- processData("input_data/data_en.sqlite")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   duration = col_double(),
##   time_run = col_double(),
##   time_render = col_double(),
##   time_show = col_double(),
##   time_end = col_double(),
##   time_commit = col_double(),
##   timestamp = col_datetime(format = ""),
##   time_switch = col_double(),
##   url_lab = col_double(),
##   meta_timezone_offset = col_double(),
##   meta_screen_width = col_double(),
##   meta_screen_height = col_double(),
##   meta_scroll_width = col_double(),
##   meta_scroll_height = col_double(),
##   meta_window_inner_width = col_double(),
##   meta_window_inner_height = col_double(),
##   meta_device_pixel_ratio = col_double(),
##   which_year_were_you_born = col_double(),
##   x = col_logical(),
##   correct = col_logical()
## )
## i Use `spec()` for the full column specifications.
```

```
en_data_2 <- processData("input_data/data_en-z.sqlite")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   duration = col_double(),
##   time_run = col_double(),
##   time_render = col_double(),
##   time_show = col_double(),
##   time_end = col_double(),
##   time_commit = col_double(),
##   timestamp = col_datetime(format = ""),
##   time_switch = col_double(),
##   url_lab = col_double(),
##   meta_timezone_offset = col_double(),
##   meta_screen_width = col_double(),
##   meta_screen_height = col_double(),
##   meta_scroll_width = col_double(),
##   meta_scroll_height = col_double(),
##   meta_window_inner_width = col_double(),
##   meta_window_inner_height = col_double(),
##   meta_device_pixel_ratio = col_double(),
##   which_year_were_you_born = col_double(),
##   x = col_logical(),
##   correct = col_logical()
## )
## i Use `spec()` for the full column specifications.
```

```
en_data_all <- do.call(bind_rows, list(en_data_1, en_data_2))
```

Participant and Experiment Information

In this section, we will put together the demographic data and experiment information data to save as an overall participant information file. This information can be merged with the other data using the `observation` column.

We will mark participants who do not meet our criteria to exclude below:

Participant did not indicate at least 18 years of age. Participant did not complete at least 100 trials. Participant did not achieve 80% correct.

```
##create demographics only data
demos <- en_data_all %>% #data frame
  filter(sender == "Demographics Form") #filter out only demographics lines

##create experiment information data
exp <- en_data_all %>%
  filter(sender == "Consent Form")

demo_cols <- c("observation", "duration",
             colnames(demos)[grep("^time", colnames(demos))],
             "please_tell_us_your_gender", "which_year_were_you_born",
             "please_tell_us_your_education_level", "native_language")
exp_cols <- c("observation", "duration",
              colnames(exp)[grep("^time", colnames(exp))],
              "url_lab",
              colnames(exp)[grep("meta", colnames(exp))])
```

```r
participant_DF <- merge(demos[ , demo_cols],
                        exp[ , exp_cols],
                        by = "observation",
                        all = T)

colnames(participant_DF) <- gsub(".x$", "_demographics", colnames(participant_DF))
colnames(participant_DF) <- gsub(".y$", "_consent", colnames(participant_DF))

participant_DF$keep <- "keep"

# only above 18
participant_DF$keep[(current_year - as.numeric(participant_DF$which_year_were_you_born)) < 18] <- "exclude"

# at least 100 trials + 80%
number_trials <- en_data_all %>% #data frame
  filter(sender == "Stimulus Real") %>%  #filter out only the real stimuli
  group_by(observation) %>%
  summarize(n_trials = n(),
            correct = sum(correct, na.rm = T) / n())

# merge with participant data
participant_DF <- merge(participant_DF,
                        number_trials,
                        by = "observation")

# mark those last few as excluded
participant_DF$keep[participant_DF$n_trials < 100] <- "exclude"
participant_DF$keep[participant_DF$correct < .80] <- "exclude"

export(participant_DF, "output_data/participant_data.csv", row.names = F)
```

Trial Data

Each language will be saved in a separate file with an item specific trial identification number to allow for matching concepts across languages (i.e., cat → katze → gatta).

Participants are expected to incorrectly answer trials; however, they are include in the raw trial level data for this output. Further, computer errors or trials due to missing data (i.e., participant inattentiveness and timeout trials, internet disconnection, computer crashes) are already marked as such in the final data with NA values.

```r
##grab only real trials
real_trials <- en_data_all %>% #data frame
  filter(sender == "Stimulus Real") %>%  #filter out only the real stimuli
  select(observation, sender_id, response, response_action, ended_on, duration,
         colnames(en_data_all)[grep("^time", colnames(en_data_all))],
         word, class, correct_response, correct)
```

The response latencies from each participant's session will be z-scored in line with recommendations from Faust, Balota, Spieler, and Ferraro (1999). We will z-score these *without* the excluded trials:

- Timeout trials (i.e., no response given in 5 s window).
- Incorrectly answered trials.
- Response latencies shorter than 160 ms.

Note that it's ok if we include participants in this file that should be overall excluded, as they will get excluded in the descriptive statistics calculations and before item level results. Basically, everything stays in this file but we mark them for keep or not keep.

```r
real_trials$original_duration <- real_trials$duration #hang on to original time

##Separate out NA data for the z-score part
##this mostly controls for timeouts
real_trials_NA <- real_trials %>% #data frame
  filter(is.na(correct) | #grab time out trials OR
           correct == FALSE | #grab incorrect trials OR
           duration < 160) %>%  #grab short rts
  mutate(Z_RT = NA, #set all Z_RTs to NA for these trials
         duration = NA,
         keep = "exclude")

##this section z-scores the rest of the data
##just not time outs
```

```
real_trials_nonNA <-
  real_trials %>% #data frame
  group_by(observation) %>% #group by participant
  filter(!is.na(correct)) %>% #take out the NA timeouts
  filter(correct == TRUE) %>% #only correct trials
  filter(duration >= 160) %>% #longer response latencies
  mutate(Z_RT = scale(duration), #create a z-score RT
         keep = "keep")

##put the time outs with the answered trials
real_trials <- bind_rows(real_trials_NA, real_trials_nonNA)

##indicate what participants to exclude
real_trials <- real_trials %>% left_join((participant_DF %>% select(observation, keep) %>%
                                            rename(keep_participant = keep)),
                                          by = c("observation" = "observation"))

##write out raw trial data
write.csv(real_trials, "output_data/trial_data.csv", row.names = F)
```

Item Data

The item file will contain lexical information about all stimuli (length, frequency, orthographic neighborhood, bigram frequency). We will merge that information at the end of the study.

The descriptive statistics calculated from the trial level data will then be included: average response latency, average standardized response latency, sample size, standard errors of response latencies, and accuracy rate. The exclusions applied above created `Z_RT` as NA, therefore, they are automatically excluded here as well.

```
##read in stimuli data
stimuli_data <- import("input_data/en_words.csv")

describeBy(real_trials$Z_RT, group = real_trials$keep) # to ensure we are not calculating any numbers on excluded trials
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf

## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning
## -Inf

##
##  Descriptive statistics by group
## INDICES: exclude
##    vars n mean sd median trimmed mad min  max range skew kurtosis se
## X1    1 0  NaN NA     NA     NaN  NA Inf -Inf  -Inf   NA       NA NA
## ------------------------------------------------------------
## INDICES: keep
##    vars     n mean sd median trimmed  mad   min  max range skew kurtosis se
## X1    1 49112    0  1  -0.25   -0.16 0.57 -2.03 15.4 17.43  3.2    18.52  0
```

```
describeBy(real_trials$duration, group = real_trials$keep)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf

## Warning in min(x, na.rm = na.rm): no non-missing arguments to max; returning
## -Inf

##
##  Descriptive statistics by group
## group: exclude
##    vars n mean sd median trimmed mad min  max range skew kurtosis se
## X1    1 0  NaN NA     NA     NaN  NA Inf -Inf  -Inf   NA       NA NA
## ------------------------------------------------------------
## group: keep
##    vars     n   mean    sd median trimmed    mad    min     max  range skew
## X1    1 49112 731.21 362.6    634  666.59 183.81 166.92 4913.23 4746.3 3.35
##    kurtosis   se
## X1    17.97 1.64
```

```
##create item level data by summarizing
item_data <- real_trials %>% #data frame
  filter(keep_participant == "keep") %>% #participants to keep
  #filter(keep == "keep") %>% #trials to keep
  #note that duration is NA for excluded trials
  #note that Z_RT is NA for excluded trials
  #so the keep is just an extra column
  #need to keep all trials because otherwise accuracy is screwy
  group_by(word, class) %>%  #group by word
  dplyr::summarize(avgRT = mean(duration, na.rm = T), #average RT
            avgZ_RT = mean(Z_RT, na.rm = T), #average Z RT
```

```r
            samplesize = length(na.omit(Z_RT)), #sample size correct
            seRT = sd(duration, na.rm = T)/sqrt(length(na.omit(duration))), #SE RT
            seZ_RT = sd(Z_RT, na.rm = T)/sqrt(length(na.omit(Z_RT))), #SE Z RT
            accuracy = length(na.omit(Z_RT))/length(Z_RT) #accuracy
            ) #word type
```

```
## `summarise()` has grouped output by 'word'. You can override using the
## `.groups` argument.
```

```r
##remove words that aren't part of the real data (testers)
item_data <- item_data %>%
  filter(word %in% c(stimuli_data$en_cue, stimuli_data$en_target))
```

No data will be excluded for being a potential outlier, however, we will recommend cut off criterion for z-score outliers at 2.5 and 3.0 and will calculate these same statistics with those subsets of trials excluded.

```r
##example outlier exclusion for Z > 2.5
##same as above with one extra filter
item_data_2.5 <- real_trials %>%
  filter(abs(Z_RT) < 2.5) %>% #take out trials above 2.5 z scores
  filter(keep_participant == "keep") %>% #participants to keep
  #filter(keep == "keep") %>% #trials to keep
  group_by(word, class) %>%
  dplyr::summarize(avgRT = mean(duration, na.rm = T),
            avgZ_RT = mean(Z_RT, na.rm = T),
            samplesize = length(na.omit(Z_RT)),
            seRT = sd(duration, na.rm = T)/sqrt(length(na.omit(duration))),
            seZ_RT = sd(Z_RT, na.rm = T)/sqrt(length(na.omit(Z_RT))))
```

```
## `summarise()` has grouped output by 'word'. You can override using the
## `.groups` argument.
```

```r
##make new column names for these calculations
colnames(item_data_2.5)[-c(1,2)] <- paste("Z2.5_", colnames(item_data_2.5)[-c(1,2)], sep = "")

##example outlier exclusion for Z > 3.0
##same as above with one extra filter
item_data_3.0 <- real_trials %>%
  filter(abs(Z_RT) < 3.0) %>% #take out trials above 3.0 z scores
  filter(keep_participant == "keep") %>% #participants to keep
  #filter(keep == "keep") %>% #trials to keep
  group_by(word, class) %>%
  dplyr::summarize(avgRT = mean(duration, na.rm = T),
            avgZ_RT = mean(Z_RT, na.rm = T),
            samplesize = length(na.omit(Z_RT)),
            seRT = sd(duration, na.rm = T)/sqrt(length(na.omit(duration))),
            seZ_RT = sd(Z_RT, na.rm = T)/sqrt(length(na.omit(Z_RT))))
```

```
## `summarise()` has grouped output by 'word'. You can override using the
## `.groups` argument.
```

```r
##make new column names for these calculations
colnames(item_data_3.0)[-c(1,2)] <- paste("Z3.0_", colnames(item_data_3.0)[-c(1,2)], sep = "")

#merge together two z score calculations
item_data_combo <- item_data %>%
  left_join(item_data_2.5,
            by = c("word" = "word", "class" = "class")) %>%
  left_join(item_data_3.0,
            by = c("word" = "word", "class" = "class"))
```

For all real words, the age of acquisition, imageability, concreteness, valence, dominance, arousal, and familiarity values will be indicated because these values do not exist for nonwords. (Example provided from lexops, real data to come after stimuli select set)

```r
##merge with stimuli data
item_data <- merge(item_data_combo,
                   lexops,
                   by.x = "word",
                   by.y = "string",
                   all.x = T)

##write out item level data
write.csv(item_data_combo, "output_data/item_data.csv", row.names = F)
```

Priming Data

Priming is defined as the subtraction of average z-scored related response latency for an item from the corresponding item in the unrelated condition. Also, we've included the calculation for non-scaled data, but the z-score calculation is recommended.

```r
# figure out trial type ----
  # only select only a few columns
  priming_trials <- real_trials %>%
  filter(keep_participant == "keep") %>% #participants to keep
  # note that we don't exclude trials here because we need to keep
  # them in order to pair together cue-target
  # they will excluded in a minute
    select(observation, duration, word, class, correct, Z_RT, sender_id, timestamp, keep) %>%
    arrange(observation, timestamp)
  # add trial code and if it's cue/target
  priming_trials$trial_code <- NA
  priming_trials$which <- NA
  # add that information
  for (person in unique(priming_trials$observation)){

    priming_trials$trial_code[priming_trials$observation == person] <-
      rep(1:400, each = 2, length.out = length(priming_trials$trial_code[priming_trials$observation == person]))

    priming_trials$which[priming_trials$observation == person] <-
      rep(c("cue", "target"), times = 2,
          length.out = length(priming_trials$trial_code[priming_trials$observation == person]))

  }

  # pivot wider with information you need
  priming_trials$unique_trial <- paste(priming_trials$observation,
                                       priming_trials$trial_code, sep = "_")
  # do it with merge because ugh pivot
  priming_wide <- merge(
    priming_trials[priming_trials$which == "cue" , ], #just cues
    priming_trials[priming_trials$which == "target" , ], #just targets
    by = "unique_trial",
    all = T
  )
  # take just what we need
  priming_wide <- priming_wide[ , c("unique_trial", "observation.x", "word.x",
                                    "class.x", "correct.x", "trial_code.x",
                                    "duration.y", "word.y", "class.y", "correct.y",
                                    "Z_RT.y", "keep.y")]
  # good names
  colnames(priming_wide) <- c("unique_trial", "observation", "cue_word",
                              "cue_type", "cue_correct", "trial_order",
                              "target_duration", "target_word", "target_type",
                              "target_correct", "target_Z_RT", "target_keep")

  # only focus on related-unrelated
  priming_focus <- subset(priming_wide, target_type == "word" & cue_type == "word")
  priming_focus$word_combo <- paste0(priming_focus$cue_word, priming_focus$target_word)

  # add if it's related or unrelated
  stimuli_data$word_combo <- paste0(stimuli_data$en_cue, stimuli_data$en_target)
  priming_focus <- merge(priming_focus, stimuli_data[ , c("type", "word_combo")],
                  by = "word_combo", all.x = T)

  # subset out NAs they are test / practice trials
  # only correct answers and trials to keep
  priming_Z <- priming_focus %>%
    filter(!is.na(type)) %>%
    filter(target_keep == "keep") %>%
    filter(target_correct == TRUE) #probably not necessary since keep does this but doesn't hurt

# Calculate Statistics ----------------------------------------------------

  priming_Z_summary <- priming_Z %>%
  ##group them by target word and condition related/unrelated
  group_by(target_word, type) %>%
  ##create average scores by condition
  dplyr::summarize(avgRT = mean(target_duration, na.rm = T),
            avgZ_RT = mean(target_Z_RT, na.rm = T),
            samplesize = length(na.omit(target_Z_RT)),
            seRT = sd(target_duration, na.rm = T)/sqrt(length(na.omit(target_duration))),
            seZ_RT = sd(target_Z_RT, na.rm = T)/sqrt(length(na.omit(target_Z_RT)))) %>%
  ##spread that into wide format so we can subtract
  pivot_wider(names_from = "type",
              values_from = c("avgRT", "avgZ_RT", "samplesize", "seRT", "seZ_RT")) %>%
  ##create the priming scores by subtracting unrelated - related for that target word only
  mutate(avgRT_prime = avgRT_unrelated - avgRT_related) %>%
  mutate(avgZ_prime = avgZ_RT_unrelated - avgZ_RT_related)
```

```
## `summarise()` has grouped output by 'target_word'. You can override using the
## `.groups` argument.
```

```
## this process will be repeated for 2.5 and 3.0 z score outliers excluded
  priming_Z_summary_no2.5 <- priming_Z %>%
  ##filter out z score outliers
  filter(target_Z_RT < 2.50) %>%
  ##group them by target word and condition related/unrelated
  group_by(target_word, type) %>%
  ##create average scores by condition
  dplyr::summarize(avgRT = mean(target_duration, na.rm = T),
            avgZ_RT = mean(target_Z_RT, na.rm = T),
            samplesize = length(na.omit(target_Z_RT)),
            seRT = sd(target_duration, na.rm = T)/sqrt(length(na.omit(target_duration))),
            seZ_RT = sd(target_Z_RT, na.rm = T)/sqrt(length(na.omit(target_Z_RT)))) %>%
  ##spread that into wide format so we can subtract
  pivot_wider(names_from = "type",
              values_from = c("avgRT", "avgZ_RT", "samplesize", "seRT", "seZ_RT")) %>%
  ##create the priming scores by subtracting unrelated - related for that target word only
  mutate(avgRT_prime = avgRT_unrelated - avgRT_related) %>%
  mutate(avgZ_prime = avgZ_RT_unrelated - avgZ_RT_related)
```

```
## `summarise()` has grouped output by 'target_word'. You can override using the
## `.groups` argument.
```

```
  priming_Z_summary_no3.0 <- priming_Z %>%
  ##filter out z score outliers
  filter(target_Z_RT < 3.0) %>%
  ##group them by target word and condition related/unrelated
  group_by(target_word, type) %>%
  ##create average scores by condition
  dplyr::summarize(avgRT = mean(target_duration, na.rm = T),
            avgZ_RT = mean(target_Z_RT, na.rm = T),
            samplesize = length(na.omit(target_Z_RT)),
            seRT = sd(target_duration, na.rm = T)/sqrt(length(na.omit(target_duration))),
            seZ_RT = sd(target_Z_RT, na.rm = T)/sqrt(length(na.omit(target_Z_RT)))) %>%
  ##spread that into wide format so we can subtract
  pivot_wider(names_from = "type",
              values_from = c("avgRT", "avgZ_RT", "samplesize", "seRT", "seZ_RT")) %>%
  ##create the priming scores by subtracting unrelated - related for that target word only
  mutate(avgRT_prime = avgRT_unrelated - avgRT_related) %>%
  mutate(avgZ_prime = avgZ_RT_unrelated - avgZ_RT_related)
```

```
## `summarise()` has grouped output by 'target_word'. You can override using the
## `.groups` argument.
```

The similarity scores calculated during stimuli selection will be included, as well as other popular measures of similarity if they are available in that language.

```
##merge target information with the similarity scores
## -- to be added after calculation for final pairs --

##write out the priming data
write.csv(priming_Z_summary, "output_data/prime_data.csv", row.names = F)
write.csv(priming_Z_summary_no2.5, "output_data/prime_data_no2.5.csv", row.names = F)
write.csv(priming_Z_summary_no3.0, "output_data/prime_data_no3.0.csv", row.names = F)
```