Data Processing

Erin M. Buchanan

Last Update 2022-02-04

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(reshape)
```

```
##
## Attaching package: 'reshape'

## The following object is masked from 'package:dplyr':
##
##     rename
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following objects are masked from 'package:reshape':
##
##     expand, smiths
```

```
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 tidyr::expand()   masks reshape::expand()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x reshape::rename() masks dplyr::rename()
```

```
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(plyr)
```

```
## ------------------------------------------------------------------------------
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## ------------------------------------------------------------------------------
##
## Attaching package: 'plyr'
## The following object is masked from 'package:purrr':
##
##     compact
## The following objects are masked from 'package:reshape':
##
##     rename, round_any
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
library(LexOPS)
```

Functions

```
# 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)))
  }
```

```r
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(rbind.fill, 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.

```
##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))])

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))

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, and these trials will be marked for potential exclusion. 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.

```
##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).

```
##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)) #filter out NA corrects

#set all Z_RTs to NA for time outs
real_trials_NA$Z_RT = NA
#set all duration to NA for time outs
real_trials_NA$duration = NA

##this section z-scores the rest of the data
##you do include incorrect trials for the z-score
##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
  mutate(Z_RT = scale(duration)) #create a z-score RT

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

##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, and here we provide a simple example from `lexOps`.

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. For averages and standard errors, the incorrect and missing trials will be excluded. The incorrect trials have the `Z_RT` marked as NA, which automatically excludes them from calculation.

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

##create item level data by summarizing
item_data <- real_trials %>% #data frame
  filter(!is.na(word)) %>%  #take out blank participants with no trials
  group_by(word) %>%  #group by word
  dplyr::summarize(avgRT = mean(duration, na.rm = T), #average RT
            avgZ_RT = mean(Z_RT, na.rm = T), #average Z RT
            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
            )

##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.

```
##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(!is.na(word)) %>%
  group_by(word) %>%
  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))))

##make new column names for these calculations
colnames(item_data_2.5)[-1] <- paste("Z2.5_", colnames(item_data_2.5)[-1], 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 2.5 z scores
  filter(!is.na(word)) %>%
  group_by(word) %>%
  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))))

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

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

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 %>%
    select(observation, duration, word, class, correct, Z_RT, sender_id, timestamp) %>%
    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")]
  # 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")

  # 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
  priming_focus <- subset(priming_focus, !is.na(type))

# participants with 100 trials + 80% ----
  participant_summary <- real_trials %>%
    group_by(observation) %>%
    dplyr::summarize(trials = length(duration),
            correct = sum(correct == TRUE, na.rm = T))
  participant_summary$percent <- participant_summary$correct / participant_summary$trials

  use_data <- participant_summary %>%
    filter(trials >= 100) %>%
    filter(percent >= .80)

  priming_focus <- subset(priming_focus, observation %in% use_data$observation)

# only correct answers for checking stimuli counts ----
  priming_Z <- subset(priming_focus, target_correct == TRUE)

# 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.

```r
##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)
```