- 9 Removals + 12 Additions

```
# SPM-2: derived normalized T-scores for each case in standardization sample;
# create raw-to-T lookup tables.

suppressMessages(library(here)) # BEST WAY TO SPECIFY FILE PATHS
suppressMessages(suppressWarnings(library(tidyverse)))
library(bestNormalize) # NORMALIZATION METHODS
suppressMessages(library(psych)) # DESCRIPTIVE TABLES
suppressMessages(library(data.table))

# READ FINALIZED STAND SAMPLE ---------------------------------------------

Child_512_Home <-
  suppressMessages(as_tibble(read_csv(
    here("INPUT-FILES/CHILD/ALLDATA-DESAMP-NORMS-INPUT/Child-512-Home-allData-desamp.csv")
  )))

score_names <- c("TOT", "SOC", "VIS", "HEA", "TOU", "TS", "BOD", "BAL", "PLA")

# Check for any dupIDs (anyDuplicated() returns row number of FIRST dup ID encountered)
anyDuplicated(Child_512_Home$IDNumber)

# Check for any NAs on IDNumber, returns TRUE if NA exist
any(is.na(Child_512_Home$IDNumber))

# extract cases with Dup ID numbers or NA on IDNumber, write out for investigation
Child_512_Home_dupMissIDs <- Child_512_Home %>%
  mutate(dup = duplicated(IDNumber)) %>%
  filter(dup == TRUE | is.na(IDNumber)) %>%
  select(-dup) %>%
  write_csv(
    here(
      paste0(
        'OUTPUT-FILES/CHILD/DUP-IDS/Child-512-Home-dupIDs-missingIDs-',
        format(Sys.Date(), "%Y-%m-%d"),
        '.csv'
      )
    ),
    na = 'missing'
  )

# DETERMINE BEST NORMALIZATION MODEL --------------------------------------

# (NOTE: THIS SECTION SHOULD BE TOGGLED OFF AFTER SELECTION OF NORMALIZATION
# MODEL)

# # # create a bestNormalize object to lock down the normalizing function that will be used on repeated runs of the norms.
# TOT_nz_obj <- bestNormalize(Child_512_Home$TOT_r
```

```
# SPM-2: derived normalized T-scores for each case in standardization sample;
# create raw-to-T lookup tables.

suppressMessages(library(here)) # BEST WAY TO SPECIFY FILE PATHS
suppressMessages(suppressWarnings(library(tidyverse)))
library(bestNormalize) # NORMALIZATION METHODS
suppressMessages(library(psych)) # DESCRIPTIVE TABLES
suppressMessages(library(data.table))

# READ FINALIZED STAND SAMPLE ---------------------------------------------

Child_512_Home <-
  suppressMessages(as_tibble(read_csv(
    here("INPUT-FILES/CHILD/ALLDATA-DESAMP-NORMS-INPUT/Child-512-Home-allData-desamp.csv")
  )))

score_names <- c("TOT", "SOC", "VIS", "HEA", "TOU", "TS", "BOD", "BAL", "PLA")

# Check for any dupIDs (anyDuplicated() returns row number of FIRST dup ID encountered)
anyDuplicated(Child_512_Home$IDNumber)

# Check for any NAs on IDNumber, returns TRUE if NA exist
any(is.na(Child_512_Home$IDNumber))

# extract cases with Dup ID numbers or NA on IDNumber, write out for investigation
Child_512_Home_dupMissIDs <- Child_512_Home %>%
  mutate(dup = duplicated(IDNumber)) %>%
  filter(dup == TRUE | is.na(IDNumber)) %>%
  select(-dup) %>%
  write_csv(
    here(
      paste0(
        'OUTPUT-FILES/CHILD/DUP-IDS/Child-512-Home-dupIDs-missingIDs-',
        format(Sys.Date(), "%Y-%m-%d"),
        '.csv'
      )
    ),
    na = 'missing'
  )

# DETERMINE BEST NORMALIZATION MODEL --------------------------------------

# (NOTE: THIS SECTION SHOULD BE TOGGLED OFF AFTER SELECTION OF NORMALIZATION
# MODEL)

# # # create a bestNormalize object to lock down the normalizing function that will be used on repeated runs of the norms.
# TOT_nz_obj <- bestNormalize(Child_512_Home$TOT_r
aw)
#
# # print transformation
# TOT_nz_obj$chosen_transform
#
# # Extract transformation type
# chosen_transform <- class(TOT_nz_obj$chosen_transform)[1]
#
# # apply the chosen method to create normalized z-scores for each case.
# TOT_nz_transform <- eval(as.name(chosen_transform))(Child_512_Home$TOT_raw)


# APPLY SELECTED NORMALIZATION MODEL TO CREATE NORMALIZED Z-SCORES --------

# Apply a static, repeatable transformation to create normalized z-scores for
# each case.

# create char vec with names for the nine score transformations
nz_transform_names <- c(paste0(score_names, '_nz_transform'))

# pull nine raw score columns into a list
raw_score_cols_list <- map(score_names, ~ Child_512_Home %>%
              pull(
                !!as.name(paste0(.x, '_raw'))
              )
)

# create the nine named objects that contain the normalization for each score
# distribution. In this call of `purrr::walk2()`, the .f calls assign(), because
# the central purpose of this code is to use assign to create a series of named
# objects in the global environment. The `walk` functions are used when the
# output of interest is a side effect. The names for these objects are contained
# in the .x argument (a char vec). The data to be normalized is in the list of
# nine raw score columns `raw_score_cols_list`, which as assigned to the .y
# argument of walk2(), using the dot . shorthand. Within assign(), the value
# argument allows the selected normalization transformation to be applied to the
# .y data.

# NOTE: MUST SUBSITUTE NAMED TRANSFORMATION FROM PREVIOUS STEP IN THIS LINE:
# value = orderNorm(.y), e.g., value = [SELECTED TRANSFORMATION](.y),

raw_score_cols_list %>%
  walk2(
    .x = c(nz_transform_names),         # names to assign
    .y = .,                # object to be assigned
    .f = ~ assign(x = .x,
                  value = orderNorm(.y),
```

```
aw)
#
# # print transformation
# TOT_nz_obj$chosen_transform
#
# # Extract transformation type
# chosen_transform <- class(TOT_nz_obj$chosen_transform)[1]
#
# # apply the chosen method to create normalized z-scores for each case.
# TOT_nz_transform <- eval(as.name(chosen_transform))(Child_512_Home$TOT_raw)


# APPLY SELECTED NORMALIZATION MODEL TO CREATE NORMALIZED Z-SCORES --------

# Apply a static, repeatable transformation to create normalized z-scores for
# each case.

# create char vec with names for the nine score transformations
nz_transform_names <- c(paste0(score_names, '_nz_transform'))

# pull nine raw score columns into a list
raw_score_cols_list <- map(score_names, ~ Child_512_Home %>%
              pull(
                !!as.name(paste0(.x, '_raw'))
              )
)

# create the nine named objects that contain the normalization for each score
# distribution. In this call of `purrr::walk2()`, the .f calls assign(), because
# the central purpose of this code is to use assign to create a series of named
# objects in the global environment. The `walk` functions are used when the
# output of interest is a side effect. The names for these objects are contained
# in the .x argument (a char vec). The data to be normalized is in the list of
# nine raw score columns `raw_score_cols_list`, which as assigned to the .y
# argument of walk2(), using the dot . shorthand. Within assign(), the value
# argument allows the selected normalization transformation to be applied to the
# .y data.

# NOTE: MUST SUBSITUTE NAMED TRANSFORMATION FROM PREVIOUS STEP IN THIS LINE:
# value = orderNorm(.y), e.g., value = [SELECTED TRANSFORMATION](.y),

raw_score_cols_list %>%
  walk2(
    .x = c(nz_transform_names),         # names to assign
    .y = .,                # object to be assigned
    .f = ~ assign(x = .x,
                  value = orderNorm(.y),
                  envir = .GlobalEnv)
  )

# Each of the named objects (normalization for each score) created in the
# previous smippet is a list. Use base::mget to put these named objects into a
# 'list of lists'. Here, mget takes a single argument, a char vec holding the
# names of the lists that are to be put into the new list `nz_transform_list`

nz_transform_list <- mget(nz_transform_names)

# Create a char vec containing the names of the output objects for the next
# step. These output objects are single-column named dfs containing the
# normalized z scores corresponding to the raw score for each case. The objects and the single
# columns within them have the same names
nz_names <- c(paste0(score_names, '_nz'))

# Create nine single-column named dataframes, each containing the normalized z
# scores for each case. The input is the list `nz_transform_list` containing the
# nine normalization objects (each itself a list). That input is assigned to the
# .y argument of `walk2()`, while the names of the output objects `nz_names` are
# assigned to the .x argument. Within assign(), the value argument has as its
# innermost function `purrr::pluck()`, which extracts an element of a list in
# the .y input. In this case, what's being extracted is the `x.t`, the vector of
# normalized z scores. That vector is wrapped in `data.frame`, to coerce it into
# a data frame, which is then wrapped in `setNames`, which names the column of
# the resulting data frame using the variable names contained in the .x
# argument.
nz_transform_list %>%
  walk2(
    .x = c(nz_names),
    .y = .,
    .f = ~ assign(x = .x,
                  value = setNames(data.frame(pluck(.y, 'x.t')), c(.x)),
                  envir = .GlobalEnv)
  )

# remove the normalization objects, which are no longer needed
rm(list = ls(nz_transform_list))


# DERIVE NORMALIZED T-SCORES FOR EACH CASE --------------------------------

# put the nine single column normalized z-score data frames into a list
nz_col_list <- mget(nz_names)

# Next snippet replaces the normalized z-score wit
```

```
                  envir = .GlobalEnv)
  )

# Each of the named objects (normalization for each score) created in the
# previous smippet is a list. Use base::mget to put these named objects into a
# 'list of lists'. Here, mget takes a single argument, a char vec holding the
# names of the lists that are to be put into the new list `nz_transform_list`

nz_transform_list <- mget(nz_transform_names)

# Create a char vec containing the names of the output objects for the next
# step. These output objects are single-column named dfs containing the
# normalized z scores corresponding to the raw score for each case. The objects and the single
# columns within them have the same names
nz_names <- c(paste0(score_names, '_nz'))

# Create nine single-column named dataframes, each containing the normalized z
# scores for each case. The input is the list `nz_transform_list` containing the
# nine normalization objects (each itself a list). That input is assigned to the
# .y argument of `walk2()`, while the names of the output objects `nz_names` are
# assigned to the .x argument. Within assign(), the value argument has as its
# innermost function `purrr::pluck()`, which extracts an element of a list in
# the .y input. In this case, what's being extracted is the `x.t`, the vector of
# normalized z scores. That vector is wrapped in `data.frame`, to coerce it into
# a data frame, which is then wrapped in `setNames`, which names the column of
# the resulting data frame using the variable names contained in the .x
# argument.
nz_transform_list %>%
  walk2(
    .x = c(nz_names),
    .y = .,
    .f = ~ assign(x = .x,
                  value = setNames(data.frame(pluck(.y, 'x.t')), c(.x)),
                  envir = .GlobalEnv)
  )

# remove the normalization objects, which are no longer needed
rm(list = ls(nz_transform_list))


# DERIVE NORMALIZED T-SCORES FOR EACH CASE --------------------------------

# put the nine single column normalized z-score data frames into a list
nz_col_list <- mget(nz_names)

# Next snippet replaces the normalized z-score wit
```

```
h a normalized T-score (and
# truncates the T-score distribution).

# map2_dfc takes a list of data frames as input, and outputs a single data
# frame, binding the transformed output columns together. In map2_dfc, the input
# list is assigned to the .x argument, and the vector of score_names is assigned
# to the .y argument. Note the use of unquoting `!!`, `as.name`, and the
# specialized equals sign `:=` for NSE (non-standard evavluation)
NT_cols <- map2_dfc(nz_col_list, score_names, ~
.x %>% mutate(
  !!as.name(paste0(.y, '_NT')) := round((!!as.name(paste0(.y, '_nz'))*10)+50)
) %>% mutate_at(
  vars(paste0(.y, '_NT')), ~ case_when(
    .x < 25 ~ 25,
    .x > 75 ~ 75,
    TRUE ~ .x
  )
) %>%
  select(
    paste0(.y, '_NT')
  )
) %>%
  mutate_if(is.numeric, as.integer)

# Bind the normalized T-score columns to the table containing raw scores for
# each case.
Child_512_Home <- Child_512_Home %>% bind_cols(NT_cols) %>%
  mutate(clin_status = 'typ',
         clin_dx = NA) %>%
  select(IDNumber, Age, age_range, Gender:Region, data, clin_status, clin_dx, everything())

# write T-scores per case table to .csv
write_csv(Child_512_Home, here(
    'OUTPUT-FILES/CHILD/T-SCORES-PER-CASE/Child-512-Home-T-Scores-per-case.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/T-SCORES-PER-CASE/Child-512-Home-T-Scores-per-case-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)

# clean up environment
rm(list = ls(pattern='.*_nz'))

# histogram to check normality
# MASS::truehist(Child_512_Home$TOT_NT, h = 1)
# hist_plot <- ggplot(data = Child_512_Home, aes(TOT_NT)) +
#   geom_histogram(
#     binwidth = .2,
#     col = "red"
#   ) +

```

```
h a normalized T-score (and
# truncates the T-score distribution).

# map2_dfc takes a list of data frames as input, and outputs a single data
# frame, binding the transformed output columns together. In map2_dfc, the input
# list is assigned to the .x argument, and the vector of score_names is assigned
# to the .y argument. Note the use of unquoting `!!`, `as.name`, and the
# specialized equals sign `:=` for NSE (non-standard evavluation)
NT_cols <- map2_dfc(nz_col_list, score_names, ~
.x %>% mutate(
  !!as.name(paste0(.y, '_NT')) := round((!!as.name(paste0(.y, '_nz'))*10)+50)
) %>% mutate_at(
  vars(paste0(.y, '_NT')), ~ case_when(
    .x < 40 ~ 40,
    .x > 80 ~ 80,
    TRUE ~ .x
  )
) %>%
  select(
    paste0(.y, '_NT')
  )
) %>%
  mutate_if(is.numeric, as.integer)

# Bind the normalized T-score columns to the table containing raw scores for
# each case.
Child_512_Home <- Child_512_Home %>% bind_cols(NT_cols) %>%
  mutate(clin_status = 'typ',
         clin_dx = NA) %>%
  select(IDNumber, Age, age_range, Gender:Region, data, clin_status, clin_dx, everything())

# write T-scores per case table to .csv
write_csv(Child_512_Home, here(
    'OUTPUT-FILES/NORMS-OUTPUT-4080T/Child-512-Home-T-Scores-per-case-4080T.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/T-SCORES-PER-CASE/Child-512-Home-T-Scores-per-case-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)

# clean up environment
rm(list = ls(pattern='.*_nz'))

# histogram to check normality
# MASS::truehist(Child_512_Home$TOT_NT, h = 1)
# hist_plot <- ggplot(data = Child_512_Home, aes(TOT_NT)) +
#   geom_histogram(
#     binwidth = .2,
#     col = "red"
#   ) +

#   scale_y_continuous(breaks = seq(0, 250, 25)) +
#   labs(title = "TOT_NT")
# print(hist_plot)

# GENERATE RAW-TO-T LOOKUP TABLES -----------------------------------------

# Generate raw-to-T lookup columns. Handle TOT and subscale scores separately,
# because each type has different raw score range. Start wtih TOT. Input is
# stand sample with raw scores and normalized T scores for each case. Group
# cases by raw score, relationship between raw and T is many-to-one.
TOT_lookup <- Child_512_Home %>% group_by(
  TOT_raw
) %>%
  # Because raw-to-T is many to one, all values of T are identical for each raw,
  # and summarizing by the min value of T per raw yields the ONLY value of T per
  # raw. But we need the raw column to contain all possible values of raw, and
  # not all possible values of raw are represented in the stand sample. Thus
  # current data object jumps possible raw values (e.g, raw = 62 and raw = 65
  # might be adjacent rows in this table)
  summarise(
    TOT_NT = min(TOT_NT)
  ) %>%
  # complete expands the table vertically, filling in missing values of raw
  # within the range given. This leaves NA cells for T for those rows that
  # didn't have raw values in the input object.
  complete(
    TOT_raw = 10:240
  ) %>%
  # fill replaces NA in T going down the table, with values from the last
  # preceding (lagging) cell that was not NA.
  fill(
    TOT_NT
  ) %>%
  # A second call of fill is needed to handle inputs where the first cell(s) of
  # T are NA. 2nd fill call is uses direction up to fill those first NA cells
  # with the value from the first subsequent (leading) cell that is not NA.
  fill(
    TOT_NT,
    .direction = "up"
  ) %>%
  rename(
    raw = TOT_raw
  ) %>%
  mutate_at(
    vars(TOT_NT), ~ case_when(
      raw < 60 ~ NA_integer_,
      TRUE ~ .x
    )
  )

```

```
#   scale_y_continuous(breaks = seq(0, 250, 25)) +
#   labs(title = "TOT_NT")
# print(hist_plot)

# GENERATE RAW-TO-T LOOKUP TABLES -----------------------------------------

# Generate raw-to-T lookup columns. Handle TOT and subscale scores separately,
# because each type has different raw score range. Start wtih TOT. Input is
# stand sample with raw scores and normalized T scores for each case. Group
# cases by raw score, relationship between raw and T is many-to-one.
TOT_lookup <- Child_512_Home %>% group_by(
  TOT_raw
) %>%
  # Because raw-to-T is many to one, all values of T are identical for each raw,
  # and summarizing by the min value of T per raw yields the ONLY value of T per
  # raw. But we need the raw column to contain all possible values of raw, and
  # not all possible values of raw are represented in the stand sample. Thus
  # current data object jumps possible raw values (e.g, raw = 62 and raw = 65
  # might be adjacent rows in this table)
  summarise(
    TOT_NT = min(TOT_NT)
  ) %>%
  # complete expands the table vertically, filling in missing values of raw
  # within the range given. This leaves NA cells for T for those rows that
  # didn't have raw values in the input object.
  complete(
    TOT_raw = 10:240
  ) %>%
  # fill replaces NA in T going down the table, with values from the last
  # preceding (lagging) cell that was not NA.
  fill(
    TOT_NT
  ) %>%
  # A second call of fill is needed to handle inputs where the first cell(s) of
  # T are NA. 2nd fill call is uses direction up to fill those first NA cells
  # with the value from the first subsequent (leading) cell that is not NA.
  fill(
    TOT_NT,
    .direction = "up"
  ) %>%
  rename(
    raw = TOT_raw
  ) %>%
  mutate_at(
    vars(TOT_NT), ~ case_when(
      raw < 60 ~ NA_integer_,
      TRUE ~ .x
    )
  )

```

```
# Repeat above for subscale raw-to-T columns.
subscale_names <- score_names[2:9]

subscale_lookup <- map(
  subscale_names,
  ~ Child_512_Home %>% group_by(
    !!as.name(paste0(.x, '_raw'))
  ) %>%
    summarise(
      !!as.name(paste0(.x, '_NT')) := min(!!as.name(paste0(.x, '_NT')))
    ) %>%
    complete(
      !!as.name(paste0(.x, '_raw')) := 10:240
    ) %>%
    fill(
      paste0(.x, '_NT')
    ) %>%
    fill(
      paste0(.x, '_NT'),
      .direction = "up"
    ) %>%
    rename(
      raw = !!as.name(paste0(.x, '_raw'))
    ) %>%
    mutate_at(
      vars(!!as.name(paste0(.x, '_NT'))), ~ case_when(
        raw > 40 ~ NA_integer_,
        TRUE ~ .x
      )
    )
) %>%
  reduce(
    left_join,
    by = 'raw'
  )

# join TOT and subscale columns
all_lookup <- full_join(TOT_lookup, subscale_lookup, by = 'raw')

all_lookup_col_names <- c(paste0(score_names, '_raw'))

# write final raw-to-T lookup table to .csv
write_csv(all_lookup, here(
    'OUTPUT-FILES/CHILD/RAW-T-LOOKUP-TABLES/Child-512-Home-raw-T-lookup.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/RAW-T-LOOKUP-TABLES/Child-512-Home-raw-T-lookup-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)


# generate print pub format raw-to-T table
all_lookup_pub <- all_lookup %>%
  # gather collapses wide table into three-column tall table with key-value
  # pairs: rawscore, scale(key var, many rows for each scale), T(value
```

```
# Repeat above for subscale raw-to-T columns.
subscale_names <- score_names[2:9]

subscale_lookup <- map(
  subscale_names,
  ~ Child_512_Home %>% group_by(
    !!as.name(paste0(.x, '_raw'))
  ) %>%
    summarise(
      !!as.name(paste0(.x, '_NT')) := min(!!as.name(paste0(.x, '_NT')))
    ) %>%
    complete(
      !!as.name(paste0(.x, '_raw')) := 10:240
    ) %>%
    fill(
      paste0(.x, '_NT')
    ) %>%
    fill(
      paste0(.x, '_NT'),
      .direction = "up"
    ) %>%
    rename(
      raw = !!as.name(paste0(.x, '_raw'))
    ) %>%
    mutate_at(
      vars(!!as.name(paste0(.x, '_NT'))), ~ case_when(
        raw > 40 ~ NA_integer_,
        TRUE ~ .x
      )
    )
) %>%
  reduce(
    left_join,
    by = 'raw'
  )

# join TOT and subscale columns
all_lookup <- full_join(TOT_lookup, subscale_lookup, by = 'raw')

all_lookup_col_names <- c(paste0(score_names, '_raw'))

# write final raw-to-T lookup table to .csv
write_csv(all_lookup, here(
    'OUTPUT-FILES/NORMS-OUTPUT-4080T/Child-512-Home-raw-T-lookup-4080T.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/RAW-T-LOOKUP-TABLES/Child-512-Home-raw-T-lookup-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)


# generate print pub format raw-to-T table
all_lookup_pub <- all_lookup %>%
  # gather collapses wide table into three-column tall table with key-value
  # pairs: rawscore, scale(key var, many rows for each scale), T(value
  # var, one row for each value of T within each scale)
  gather(scale, T,-raw) %>%
  group_by(scale) %>%
  # expand the table vertically, adding new rows, so there's a row for every possible T value
  complete(T = 25:75) %>%
  ungroup() %>%
  # regroup table by two levels
  group_by(scale, T) %>%
  # filter step retains all 1-row groups, and the first and last rows of any
  # multi-row groups. n() == 1 returns 1-row groups; n() > 1 & row_number()
  # %in% c(1, n()) returns rows of multi-row groups with the row number of
  # either 1 (first row), or n() which is the number of rows and also the
  # number of the last row. The first and last rows hold the min and max
  # values of raw for that value of T (the grouping variable)
  filter(n() == 1 | n() > 1 & row_number()  %in% c(1, n())) %>%
  # Summarise creates a table with one row per group (one row per
  # possible value of T). For the 1-row groups, str_c simply passes the
  # value of raw as a string; for the multi-row groups, str_c joins the min
  # and max values of raw with the '--' separator.
  summarise(raw = str_c(raw, collapse = '--')) %>%
  # recode missing values of raw to '-'
  mutate_at(vars(raw), ~ case_when(is.na(.x) ~ '-', TRUE ~ .x)) %>%
  # sort on two levels
  arrange(scale, desc(T)) %>%
  # spread table back to wide, all values of T (one row for each), scale
  # columns filled with values of rawscore
  spread(scale, raw) %>%
  # sort descending on T
  arrange(desc(T)) %>%
  # rename with desired final column names
  rename_at(vars(ends_with('_NT')), ~ gsub("_NT", "_raw", .)) %>%
  # order columns left-to-right
  select(T, all_lookup_col_names)

# write final print format raw-to-T lookup table to .csv
write_csv(all_lookup_pub, here(
'OUTPUT-FILES/CHILD/PRINT-FORMAT-NORMS-TABLES/Child-512-Home-print-raw-T-lookup.csv'
    # paste0(
  #   'OUTPUT-FILES/CHILD/PRINT-FORMAT-NORMS-TABLES/Child-512-Home-print-raw-T-lookup-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
```

```
  # var, one row for each value of T within each scale)
  gather(scale, T,-raw) %>%
  group_by(scale) %>%
  # expand the table vertically, adding new rows, so there's a row for every possible T value
  complete(T = 40:80) %>%
  ungroup() %>%
  # regroup table by two levels
  group_by(scale, T) %>%
  # filter step retains all 1-row groups, and the first and last rows of any
  # multi-row groups. n() == 1 returns 1-row groups; n() > 1 & row_number()
  # %in% c(1, n()) returns rows of multi-row groups with the row number of
  # either 1 (first row), or n() which is the number of rows and also the
  # number of the last row. The first and last rows hold the min and max
  # values of raw for that value of T (the grouping variable)
  filter(n() == 1 | n() > 1 & row_number()  %in% c(1, n())) %>%
  # Summarise creates a table with one row per group (one row per
  # possible value of T). For the 1-row groups, str_c simply passes the
  # value of raw as a string; for the multi-row groups, str_c joins the min
  # and max values of raw with the '--' separator.
  summarise(raw = str_c(raw, collapse = '--')) %>%
  # recode missing values of raw to '-'
  mutate_at(vars(raw), ~ case_when(is.na(.x) ~ '-', TRUE ~ .x)) %>%
  # sort on two levels
  arrange(scale, desc(T)) %>%
  # spread table back to wide, all values of T (one row for each), scale
  # columns filled with values of rawscore
  spread(scale, raw) %>%
  # sort descending on T
  arrange(desc(T)) %>%
  # rename with desired final column names
  rename_at(vars(ends_with('_NT')), ~ gsub("_NT", "_raw", .)) %>%
  # order columns left-to-right
  select(T, all_lookup_col_names) %>%
  # drop row where T == NA
  filter(!is.na(T))

# write final print format raw-to-T lookup table to .csv
write_csv(all_lookup_pub, here(
'OUTPUT-FILES/NORMS-OUTPUT-4080T/Child-512-Home-print-raw-T-lookup-4080T.csv'
    # paste0(
  #   'OUTPUT-FILES/CHILD/PRINT-FORMAT-NORMS-TABLES/Child-512-Home-print-raw-T-lookup-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
```

```
)

# write raw score descriptives for all scales (using psych::describe)
Child_512_Home_raw_desc <-
  Child_512_Home %>%
  select(contains('raw')) %>%
  describe(fast = T) %>%
  rownames_to_column() %>%
  rename(scale = rowname) %>%
  select(scale, n, mean, sd) %>%
  mutate_at(vars(mean, sd), ~(round(., 2)))

write_csv(Child_512_Home_raw_desc, here(
  'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-raw-desc.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-raw-desc-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)

# write table of demographic counts

var_order <- c("data", "age_range", "Age", "Gender", "ParentHighestEducation", "HighestEducation",
               "Ethnicity", "Region")

cat_order <- c(
  # data
  NA, "SM", "Qual", "Sp", "Daycare", "In-house-Eng", "In-house-Sp", "In-house-Alt",
  # age_range
  NA, "3.5 to 6 mo", "03.5 to 10 mo", "7 to 10.5 mo", "09.5 to 20 mo",  "11 to 31.5 mo",
  "21 to 31.5 mo", "5 to 8 years", "9 to 12 years", "12 to 13 years", "14 to 15 years",
  "16 to 17 years", "18 to 21 years", "21.00 to 30.99 years", "31.00 to 40.99 years",
  "41.00 to 50.99 years", "51.00 to 64.99 years", "65.00 to 99.99 years",
  # Age
  "2", "3", "4", "5",
  # Gender
  NA, "Male", "Female",
  # ParentHighestEducation & HighestEducation
  NA, "Did not complete high school (no diploma)", "High school graduate (including GED)",
  "Some college or associate degree", "Bachelor's degree or higher",
  # Ethnicity
  NA, "Hispanic", "Asian", "Black", "White", "AmericanIndAlaskanNat",
  "NativeHawPacIsl", "MultiRacial", "Other",
  # Region
  NA, "northeast", "midwest", "south", "west")


Child_512_Home_demo_counts <- Child_512_Home %>%
  select(data, age_range, ParentHighestEducation, Gender, Ethnicity, Region) %>%
  gather("Variable", "Category") %>%
  group_by(Variable, Category) %>%
```

```
)

# write raw score descriptives for all scales (using psych::describe)
Child_512_Home_raw_desc <-
  Child_512_Home %>%
  select(contains('raw')) %>%
  describe(fast = T) %>%
  rownames_to_column() %>%
  rename(scale = rowname) %>%
  select(scale, n, mean, sd) %>%
  mutate_at(vars(mean, sd), ~(round(., 2)))

write_csv(Child_512_Home_raw_desc, here(
  'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-raw-desc.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-raw-desc-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = ''
)

# write table of demographic counts

var_order <- c("data", "age_range", "Age", "Gender", "ParentHighestEducation", "HighestEducation",
               "Ethnicity", "Region")

cat_order <- c(
  # data
  NA, "SM", "Qual", "Sp", "Daycare", "In-house-Eng", "In-house-Sp", "In-house-Alt",
  # age_range
  NA, "3.5 to 6 mo", "03.5 to 10 mo", "7 to 10.5 mo", "09.5 to 20 mo",  "11 to 31.5 mo",
  "21 to 31.5 mo", "5 to 8 years", "9 to 12 years", "12 to 13 years", "14 to 15 years",
  "16 to 17 years", "18 to 21 years", "21.00 to 30.99 years", "31.00 to 40.99 years",
  "41.00 to 50.99 years", "51.00 to 64.99 years", "65.00 to 99.99 years",
  # Age
  "2", "3", "4", "5",
  # Gender
  NA, "Male", "Female",
  # ParentHighestEducation & HighestEducation
  NA, "Did not complete high school (no diploma)", "High school graduate (including GED)",
  "Some college or associate degree", "Bachelor's degree or higher",
  # Ethnicity
  NA, "Hispanic", "Asian", "Black", "White", "AmericanIndAlaskanNat",
  "NativeHawPacIsl", "MultiRacial", "Other",
  # Region
  NA, "northeast", "midwest", "south", "west")


Child_512_Home_demo_counts <- Child_512_Home %>%
  select(data, age_range, ParentHighestEducation, Gender, Ethnicity, Region) %>%
  gather("Variable", "Category") %>%
  group_by(Variable, Category) %>%
  count(Variable, Category) %>%
  arrange(match(Variable, var_order), match(Category, cat_order)) %>%
  ungroup() %>%
  mutate(Variable = case_when(
    lag(Variable) == "data" & Variable == "data" ~ "",
    lag(Variable) == "age_range" & Variable == "age_range" ~ "",
    lag(Variable) == "Gender" & Variable == "Gender" ~ "",
    lag(Variable) == "ParentHighestEducation" & Variable == "ParentHighestEducation" ~ "",
    lag(Variable) == "Ethnicity" & Variable == "Ethnicity" ~ "",
    lag(Variable) == "Region" & Variable == "Region" ~ "",
    TRUE ~ Variable
  ))

write_csv(Child_512_Home_demo_counts, here(
  'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-demo-counts.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-demo-counts-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = '(missing)'
)
```

```
  count(Variable, Category) %>%
  arrange(match(Variable, var_order), match(Category, cat_order)) %>%
  ungroup() %>%
  mutate(Variable = case_when(
    lag(Variable) == "data" & Variable == "data" ~ "",
    lag(Variable) == "age_range" & Variable == "age_range" ~ "",
    lag(Variable) == "Gender" & Variable == "Gender" ~ "",
    lag(Variable) == "ParentHighestEducation" & Variable == "ParentHighestEducation" ~ "",
    lag(Variable) == "Ethnicity" & Variable == "Ethnicity" ~ "",
    lag(Variable) == "Region" & Variable == "Region" ~ "",
    TRUE ~ Variable
  ))

write_csv(Child_512_Home_demo_counts, here(
  'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-demo-counts.csv'
  # paste0(
  #   'OUTPUT-FILES/CHILD/DESCRIPTIVES/Child-512-Home-demo-counts-',
  #   format(Sys.Date(), "%Y-%m-%d"),
  #   '.csv'
  # )
),
na = '(missing)'
)
```