Rating Scale Analysis

Internal Consistency (coefficient alpha)

This code template can handle a set of input files that represent different forms (e.g., different combinations of age-range and rater). It generates a single .csv output file summarizing internal consistency analysis across all forms. The output table is formatted as in this example:

```
   form      n scale alpha   SEM CI_90    CI_95
 1 CP     1000 S1     0.08  2.72 6 -- 15  5 -- 16
 2             S2     0.12  2.73 6 -- 15  5 -- 16
 3             S3     0.04  2.73 6 -- 15  5 -- 15
 4             S4     0.03  2.75 6 -- 15  5 -- 16
 5             S5     0     2.76 6 -- 15  5 -- 16
 6             TOT    0.52  6.13 41 -- 61 39 -- 63
 7 CT     1000 S1     0.03  2.75 6 -- 15  5 -- 16
 8             S2     0.08  2.75 6 -- 15  5 -- 16
 9             S3     0.07  2.73 6 -- 15  5 -- 16
10             S4     0.04  2.76 6 -- 15  5 -- 16
```

Where `CI_90` and `CI_95` are the 90% and 95% confidence intervals, respectively. Note that alpha values are unexpectedly low, due to the use of simulated (as opposed to human-generated) data.

1. Load packages, read data   EXECUTABLE CODE

```r
suppressPackageStartupMessages(library(here))
suppressMessages(suppressWarnings(library(tidyverse)))
suppressMessages(library(psych))

urlRemote_path  <- "https://raw.github.com/"
github_path <- "wpspublish/DSHerzberg-RATING-SCALE-ANALYSIS/master/INPUT-FILES/"

data_RS_sim_child_parent <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-child-parent.csv")
)))
data_RS_sim_child_teacher <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-child-teacher.csv")
)))
data_RS_sim_teen_parent <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-teen-parent.csv")
)))
data_RS_sim_teen_teacher <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-teen-teacher.csv")
)))

form_acronyms <- c("cp", "ct", "tp", "tt")

scale_items_suffix <- c("S1", "S2", "S3", "S4", "S5", "TOT")

form_scale_cols <-
```

```
  crossing(str_to_upper(form_acronyms), scale_items_suffix) %>%
  set_names(c("form", "scale"))

input_recode_list <- map(
  lst(
    data_RS_sim_child_parent,
    data_RS_sim_child_teacher,
    data_RS_sim_teen_parent,
    data_RS_sim_teen_teacher
  ),
  ~
    .x %>%
    mutate(
      across(
        contains("cpi") |
          contains("cti") | contains("tpi") | contains("tti"),
        ~ case_when(
          .x == "never" ~ 1,
          .x == "occasionally" ~ 2,
          .x == "frequently" ~ 3,
          .x == "always" ~ 4
        )
      )
    )
)
```

COMMENTED SNIPPETS

Load packages for file path specification (`here`), data wrangling (`tidyverse`), and psychometric data simulation and analysis (`psych`).

```
suppressPackageStartupMessages(library(here))
suppressMessages(suppressWarnings(library(tidyverse)))
suppressMessages(library(psych))
```

Specify file paths and retrieve data from a remote host. The script reads in four input files consisting of simulated rating-scale (`RS`) data:

- `data-RS-sim-child-parent.csv`: `child` age range (5-12 yo), `parent` report
- `data-RS-sim-child-teacher`: `child` age range (5-12 yo), `teacher` report
- `data-RS-sim-teen-parent.csv`: `teen` age range (13-18 yo), `parent` report
- `data-RS-sim-teen-teacher.csv`: `teen` age range (13-18 yo), `teacher` report

```
urlRemote_path  <- "https://raw.github.com/"
github_path <- "wpspublish/DSHerzberg-RATING-SCALE-ANALYSIS/master/INPUT-FILES/"

data_RS_sim_child_parent <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-child-parent.csv")
)))
data_RS_sim_child_teacher <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-child-teacher.csv")
)))
data_RS_sim_teen_parent <- suppressMessages(read_csv(url(
  str_c(urlRemote_path, github_path, "data-RS-sim-teen-parent.csv")
)))
data_RS_sim_teen_teacher <- suppressMessages(read_csv(url(
```

```
  str_c(urlRemote_path, github_path, "data-RS-sim-teen-teacher.csv")
)))
```

2. Set up data structures for analysis, output EXECUTABLE CODE

```
form_acronyms <- c("cp", "ct", "tp", "tt")

scale_items_suffix <- c("S1", "S2", "S3", "S4", "S5", "TOT")

form_scale_cols <-
  crossing(str_to_upper(form_acronyms), scale_items_suffix) %>%
  set_names(c("form", "scale"))

input_recode_list <- map(
  lst(
    data_RS_sim_child_parent,
    data_RS_sim_child_teacher,
    data_RS_sim_teen_parent,
    data_RS_sim_teen_teacher
  ),
  ~
    .x %>%
    mutate(
      across(
        contains("cpi") |
          contains("cti") | contains("tpi") | contains("tti"),
        ~ case_when(
          .x == "never" ~ 1,
          .x == "occasionally" ~ 2,
          .x == "frequently" ~ 3,
          .x == "always" ~ 4
        )
      )
    )
)

TOT_item_names_list <- map(form_acronyms,
                           ~
                             str_c(str_c(.x, "i"), str_pad(
                               as.character(1:50), 2, side = "left", pad = "0"
                             )))

nth_element <- function(vector, starting_position, interval) {
  vector[seq(starting_position, length(vector), interval)]
}

scale_item_vectors <- map(TOT_item_names_list,
                          ~ splice(map(1:5, ~ nth_element(.y, .x, 5), .y = .x), .x)) %>%
  flatten()

scale_item_data <- tibble(
  data = rep(input_recode_list,
             each = 6),
  item_names = scale_item_vectors) %>%
```

```
  bind_cols(form_scale_cols) %>%
  mutate(items = map2(data, item_names, ~ .x %>% select(all_of(.y))))

scale_n_mean_sd <- map_df(
  lst(
    data_RS_sim_child_parent,
    data_RS_sim_child_teacher,
    data_RS_sim_teen_parent,
    data_RS_sim_teen_teacher
  ),
  ~
    .x %>%
    select(contains("raw")) %>%
    describe(fast = T) %>%
    rownames_to_column(var = "scale_name") %>%
    mutate(
      form = str_sub(scale_name, 1, 2),
      scale = str_sub(scale_name, 3,-5)
    ) %>%
    select(form, scale, n, mean, sd) %>%
    tibble()
)
```

COMMENTED SNIPPETS

Initialize containers for form and scale abbreviations, to be used elsewhere in the script.

`form_scale_cols` is a 24-row data frame holding the `form` and `scale` columns of the final output table. It is assembled with `tidyr::crossing()` which takes the two character vectors `form_acronyms` and `scale_items_suffix` as its arguments. `crossing()` creates a data frame with the two input vectors as columns, expanding each vector so that all possible combinations of the vector elements are represented. This provides the required structure for the output table, where each `form` has an identical set of six `scale` rows. We wrap `form_acronyms` in `stringr::str_to_upper()` to change the case of the form acronyms.

```
form_acronyms <- c("cp", "ct", "tp", "tt")

scale_items_suffix <- c("S1", "S2", "S3", "S4", "S5", "TOT")

form_scale_cols <-
  crossing(str_to_upper(form_acronyms), scale_items_suffix) %>%
  set_names(c("form", "scale"))
```

To conduct the analysis, we need to recode the item responses to numeric variables. We put the four input data frames into a list, using `tibble::lst()`, which retains the input object names as list element names. We then `map()` an anonymous recoding function over this list. `case_when()` specifies the logic for recoding the input strings into numbers (e.g., if the input cell value is `"never"`, the recode output is `1`). `across()` specifies the subset of columns that will be recoded. `contains()` is a `tidyselect` helper that captures columns whose names contain a certain substring (e.g., `"cpi`, the acronym for child-parent items).

To account for the four different item name prefixes in the four input files, we pass a four-element predicate as the first argument to `across()`, in which we use the `|` operator to denote a disjunctive set. That is, if *any* of the four item name prefixes is present in the current iteration of the input data set, `across()` will capture and recode all the columns whose names contain that prefix. The mapping operation returns a four-element list containing the four input data frames, with all item responses recoded to numeric.

```r
input_recode_list <- map(
  lst(
    data_RS_sim_child_parent,
    data_RS_sim_child_teacher,
    data_RS_sim_teen_parent,
    data_RS_sim_teen_teacher
  ),
  ~
    .x %>%
    mutate(
      across(
        contains("cpi") |
          contains("cti") | contains("tpi") | contains("tti"),
        ~ case_when(
          .x == "never" ~ 1,
          .x == "occasionally" ~ 2,
          .x == "frequently" ~ 3,
          .x == "always" ~ 4
        )
      )
    )
)
```

The output table provides alpha coefficients for the total score (`TOT`) and five subscale scores (`S1`, `S2`, `S3`, `S4`, `S5`). To get these values, we need character vectors containing the column names of the items that contribute to each score. We start with vectors for the `TOT` items. These can be considered "master" vectors in that the subscale item names are simply subsets of the `TOT` item names.

To get the `TOT` vectors, we `map()` a string concatenation function over the character vector `form_acronyms`, whose four elements are needed to assemble the item column names. The concatenation function uses `stringr::str_c` to combine string elements. In the inner call of `str_c()`, the token `.x` represents the element of `form_acronyms` in the current iteration of `map()`. It is combined with `"i"` to yield the four item-name prefixes `cti`, `cpi`, `tpi`, `tti`. The item name suffixes are a numerical sequence `1:50`, with single-digit numbers left-padded with zeros using `stringr::str_pad()`. `map()` returns a list containing the four `TOT` vectors, whose elements (the item column names) are `cpi01`, `cpi02`, `cpi03` and so on (for example).

```r
TOT_item_names_list <- map(form_acronyms,
                           ~
                             str_c(str_c(.x, "i"), str_pad(
                               as.character(1:50), 2, side = "left", pad = "0"
                             )))
```

Now that we have vectors containing all possible item column names, we can define a function named `nth_element()` that extracts subsets of names for each subscale. Conveniently, the subscale structure in the input files is regularized, such that in the `child-parent` input, for example, the `S1` names start with item `cpi01`, and include every fifth item (e.g., `cpi06`, `cpi11`, and so on). The item composition of the other subscales follows this same logic (e.g., `S2` includes `cpi02`, `cpi07`, `cpi12`, and so on).

`function(vector, starting_position, interval)` specifies that `nth_element` will take three arguments: a `TOT` item vector, the starting position of the first item of the subscale, and the number of items (or interval) between each subscale item. The body of `nth_element()` is defined within curly braces `{}`. It returns a subset of the input `TOT` vector, using the expression `vector[]`, where the straight braces `[]` specify how the input is to be subsetted. That specification is provided as a numerical sequence using `base::seq()`, which takes three arguments: the starting number of the sequence (given by the `starting_position` argument passed to `nth_element`), the largest possible number in the sequence (given by the `base::length()` of the input vector), and the increment of the sequence (given by the `interval` argument passed to `nth_element()`).

Thus, `seq(1, 50, 5)` returns the vector of positions `1 6 11 16 21 26 31 36 41 46`.

```
nth_element <- function(vector, starting_position, interval) {
  vector[seq(starting_position, length(vector), interval)]
}
```

We want to write robust, flexible code that can specify item-name vectors for all forms and subscales in a single operation. In general, if we have k forms, and each form includes a `TOT` scale and j subscales, then we need to specify $k * (\, j + 1)$ item-name vectors. In the current example, $k = 4$ and $j = 5$, thus necessitating 24 item name vectors across all forms.

To get the 24 vectors into a list, we employ a nested mapping structure, using the inner call of `map()` to deploy the `nth_element()` subsetting function. In this inner call, we map `nth_element()` over a numerical vector, the sequence `1:5`. We pass `.y`, `.x`, and 5 as the three arguments to `nth_element()`:

- `vector` is passed as `.y`, which refers to the `.x` referent from the *outer* call of `map()`.
- `starting_position` is passed as `.x`, which refers to the `.x` referent from the *inner* call of `map()`.
- `interval` is passed as `5`, a constant in this application.

Because the inner `map()` call is processing two vectors (its own `.x` argument, and the `.x` argument from the outer `map()` call) we need to also supply the `.y = .x` argument, which tells R that the `.y` argument of the inner `map()` call refers to the `.x` vector passed by the outer `map()` call.

The mapping operation is completed by the outer `map()` call, which iterates over `TOT_item_names_list`, the list containing the item column name vectors for the four `TOT` scales. The function applied to this list is the inner `map` call, wrapped in `purrr::splice()`.

To understand how `splice` works in this application, we need to unpack the iterations of the inner `map()` call. Each iteration returns a vector of column names specific to the pairing of a particular form (e.g., `cp`) passed from the outer `map()` call, and a particular subscale (e.g., `S1`) identified by the inner `map()` call. The complete iteration cycle of the inner `map()` call thus returns a list of the five subscale item-name vectors for a particular form (e.g., `cp`).

But, because the inner `map()` call is nested within the outer `map()` call, that complete iteration cycle itself iterates four times, over the four elements of `TOT_item_names_list` (the `.x` argument of the outer `map()` call). The result of this outer iteration cycle is a "list of lists", i.e., a four-element list (one element for each form), with each element itself a five-element list (containing the five subscale item vectors for a particular form). The "list of lists" thus contains all subscale vectors across all forms, thus constituting 20 of the 24 needed vectors. The remaining four are the `TOT` item vectors held by `TOT_item_names_list`.

`splice()` joins two objects into a single list. Here it takes two arguments: the current iteration of the `.x` argument from the outer `map()` call, and the output of a single iteration of the inner `map()` call. Thus, for any single iteration of the outer `map()`call, `splice()` is joining into a single list the `TOT` item vector for a particular form (`.x` from the outer `map` call) with the five subscale item vectors for that same form (the output of a single iteration of the inner `map()` call). The complete iteration cylcle of the outer `map()` call thereby returns a four-element "list of lists", in which each list element is itself a list of all six item vectors (`TOT` and subscales) for a particular form. This "list of lists" contains all 24 required vectors.

A final step is required to prepare the output for downstream processing. Piping the "list of lists" through `purrr::flatten()` removes one level of hierarchy and returns the 24 vectors in a single flat list named `scale_item_vectors`. The item-name vectors are arranged in the desired sequence, e.g., all six vectors for one form, followed by all six vectors for the next form, and so on.

```
scale_item_vectors <- map(TOT_item_names_list,
                          ~ splice(map(1:5, ~ nth_element(.y, .x, 5), .y = .x), .x)) %>%
  flatten()
```

The next snippet uses the `purrr` list-column workflow to subset the input data sets so that they include only the item columns needed for the internal consistency analysis of each scale. In the same operation, we also set up the basic structure of the final output table, which includes 24 rows (four forms X six scales), each containing the internal consistency output for one scale.

We initialize a new data frame `scale_item_data`, and call `tibble::tibble()` to create two 24-row list-columns. The `data` list-column contains the four recoded input data frames (held in `input_recode_list`, which is repeated six times with `base::rep(each = 6)`). The `item_names` list-column holds the list (created in the previous snippet) of the 24 scale-wise item-column name vectors. Thus, each row of the `data` list-column contains a data frame, and each row of the `item_names` list-column contains a vector. At this point, `scale_item_data` has the input data sets paired up row-wise with their item-column names for `TOT` and the five subscales.

Next we use `dplyr::bind_cols()` to join two additional two columns containing scale and form acronyms (held, in the required sort order, in the previously created `form_scale_cols` data frame).

The final operation in this snippet is to subset the input data frames. The list-column workflow enables the row-wise mapping of `select()`, sourcing required arguments from the same row. In essence, each row of piped data object becomes a self-contained data-processing workflow, holding the input data frame, other arguments passed to the mapped-on `select()` function, and the ouput data frame.

`select()` takes two arguments: an input data frame, and a vector of column names that determines the structure of the output data frame. We use `purrr::map2()` to map `select()` over the `data` and `item_names` list-columns, each of which passes an argument to `select()`. `data` contains the input data frame (the `.x` referent within `map2()`), and `item_names` contains the vector of column names for a particular scale (the `.y` referent within `map2()`). `mutate()` puts the output of this mapping into a new list-column `items`, which holds data frames consisting of the subsets of item columns required for the internal consistency analysis of each scale.

```
scale_item_data <- tibble(
  data = rep(input_recode_list,
             each = 6),
  item_names = scale_item_vectors) %>%
  bind_cols(form_scale_cols) %>%
  mutate(items = map2(data, item_names, ~ .x %>% select(all_of(.y))))
```

We now extract certain raw score descriptive statistics needed for the final output table. The four input data sets are put into a named list using `lst()`, and we map an anonymous function over this input list using `map_df()`, which returns a single data frame containing the descriptive statistics for all four input files.

The anonymous function (set off with the formula shorthand `~`) pipes the input data frame (the `.x` referent of `map_df()`) into `select(contains("raw"))`, which returns only the raw score columns. We obtain descriptive statistics on these columns using `psych::describe()`. The argument `fast = T` limits the output to the most frequently reported measures (e.g., `n`, `mean`, `sd`, `min`, `max`, `range`, `se`). These measures become column names for a summary data object, which now has a row for each score being analyzed, as in:

```
          vars    n  mean   sd min max range   se
CPS1_raw     1 1000 10.27 2.84   1  19    18 0.09
CPS2_raw     2 1000 10.19 2.92   2  20    18 0.09
CPS3_raw     3 1000 10.08 2.79   2  21    19 0.09
CPS4_raw     4 1000 10.30 2.79   2  21    19 0.09
CPS5_raw     5 1000 10.21 2.77   1  19    18 0.09
CPTOT_raw    6 1000 51.05 8.82  18  73    55 0.28
```

We then use a series of functions to create `form` and `scale` columns for the summary data object:

- `tibble::rownames_to_column(var = "scale_name")`: extracts the scale names (which are row names in the current data object) into a regular column, named `scale_name`.
- `mutate()`: initializes the `form` and `scale` columns, and populates them (using `stringr::str_sub()`) with substrings from the `scale_name` column.
- `select()`: keeps only the columns needed for the final output table.

- `tibble()`: regularizes the object into a tibble, removing unneeded attributes created by the `psych` package.

```
scale_n_mean_sd <- map_df(
  lst(
    data_RS_sim_child_parent,
    data_RS_sim_child_teacher,
    data_RS_sim_teen_parent,
    data_RS_sim_teen_teacher
  ),
  ~
    .x %>%
    select(contains("raw")) %>%
    describe(fast = T) %>%
    rownames_to_column(var = "scale_name") %>%
    mutate(
      form = str_sub(scale_name, 1, 2),
      scale = str_sub(scale_name, 3,-5)
    ) %>%
    select(form, scale, n, mean, sd) %>%
    tibble()
)
```

Here is the output structure of `scale_n_mean_sd`:

```
form  scale     n  mean    sd
CP    S1     1000  10.3  2.84
CP    S2     1000  10.2  2.92
CP    S3     1000  10.1  2.79
...[intervening rows]...
TT    S4     1000  10.3  2.82
TT    S5     1000  10.2  2.90
TT    TOT    1000  51.1  9.11
```

3. Conduct internal consistency analysis; assemble and write final output table EXECUTABLE CODE

```
alpha_output <- scale_item_data %>%
  mutate(alpha = map(items, ~ alpha(cor(.x))[["total"]])) %>%
  unnest(alpha) %>%
  select(form, scale, raw_alpha) %>%
  rename(alpha = raw_alpha) %>%
  left_join(scale_n_mean_sd, by = c("form", "scale")) %>%
  group_by(form) %>%
  mutate(
    SEM = sd * (sqrt(1 - alpha)),
    CV_90_UB = round(mean + 1.6449 * SEM),
    CV_90_LB = round(mean - 1.6449 * SEM),
    CV_95_UB = round(mean + 1.96 * SEM),
    CV_95_LB = round(mean - 1.96 * SEM),
    CI_90 = str_c(CV_90_LB, "--", CV_90_UB),
    CI_95 = str_c(CV_95_LB, "--", CV_95_UB),
    across(is.numeric, ~ round(., 2)),
    form = case_when(row_number() == 1 ~ form,
                     T ~ NA_character_),
    n = case_when(row_number() == 1 ~ n,
```

```
                T ~ NA_real_)
  ) %>%
  select(form, n, scale, alpha, SEM, CI_90, CI_95)

write_csv(alpha_output,
          here("OUTPUT-FILES/TABLES/alpha-summary-by-form.csv"),
          na = "")
```

COMMENTED SNIPPETS

We are now ready to conduct the internal consistency analysis and obtain coefficient alpha for each scale. The required output structure, with rows sorted by form, and scale rows nested within each form, is present in the previously created `scale_item_data` data frame. `scale_item_data` is piped into `mutate()` to create a new list-column`alpha`, which will hold the analysis output of the `psych::alpha()` function.

Within `mutate()`, `map()` is used to apply a function to each row and return the ouput to the new `alpha` column. The `.x` argument to `map()` is the same-row element of the `items` column (i.e., the data frame containing the item columns for the scale being analyzed in that row).

`alpha()` requires as its primary argument a correlation matrix of the columns to be analyzed (supplied here by `stats::cor(.x)`). `alpha()` returns a list, with the desired analysis output held in the `"total"` element as single-row data frame, which is extracted using double brackets `[[]]` and placed by `mutate()` into the new `alpha` list-column. By calling `tidyr::unnest()`, we flatten out the data frames contained in the `alpha` list-column into regular columns in the piped data object. We then `select()` and `rename()` columns needed for the output table.

```
alpha_output <- scale_item_data %>%
  mutate(alpha = map(items, ~ alpha(cor(.x))[["total"]])) %>%
  unnest(alpha) %>%
  select(form, scale, raw_alpha) %>%
  rename(alpha = raw_alpha) %>%
```

Using `dplyr::left_join()`, we bring in the descriptive statistics columns in the `scale_n_mean_sd` data frame, indexing `by = c("form", "scale")` to ensure correct alignment of rows. We then `group_by(form)` so that subsequent functions can process subsets of rows consisting of the six scales for each form.

```
  left_join(scale_n_mean_sd, by = c("form", "scale")) %>%
  group_by(form) %>%
```

The remaining code in this section uses `mutate()` to create new columns for SEM and confidence intervals, to round numerical values, and to make labeling in the `form` and `scale` columns more readable in the final output table. We then `select()` the columns for the final output table and write that table to .csv. Some additional detail:

- `CI_90 = str_c(CV_90_LB, "--", CV_90_UB)`: `str_c()` concatenates numerical lower and upper bounds (`LB`, `UB`) into a character string, the 90% confidence interval. `"--"` is used as a separator to prevent MS Excel from automatically reformatting the confidence interval as a date when it opens the .csv output table.
- `across(is.numeric, ~ round(., 2))`: selects only numeric columns to `round()`, by including the predicate `base::is.numeric()` as its first argument.
- `form = case_when(row_number() == 1 ~ form, T ~ NA_character_)`: makes the `form` column more readable in the output table. Prior to this operation, the `form` column repeats a form's label in each of its six scale rows. `case_when` specifies that the label will appear only in the first row for that form (`row_number() == 1`), and not in the that form's remaining five rows (`T ~ NA_character_`). Because the data object was previously grouped by `form`, `case_when()` treats each form's set of six rows as an independent group, with row numbers going from 1 to 6. Along the 24 rows of `form`, therefore, row labels remain present only in the first row of each form group.

- `write_csv(na = "")`: writes `NA` values as blank cells in the output .csv.