Exercise: Pivoting Data Frames

Table of Contents

[1 Overview 1](#_Toc160142786)

[2 Step 1: Evaluating your Script File for Task Sub-setting 1](#_Toc160142787)

[3 Step 2: Evaluating your Script File for Pivoting/Transforming 2](#_Toc160142788)

[4 Step 3: Adding the Pivoting/Transforming Code 2](#_Toc160142789)

[5 Step 4: Cleaning your Long File 3](#_Toc160142790)

[6 Step 5: Summarizing Data 3](#_Toc160142791)

[6.1 General Summarizing Approaches: 3](#_Toc160142792)

[6.2 Symmetry Span 3](#_Toc160142793)

[6.3 Go/No-Go 4](#_Toc160142794)

[7 Understanding the Pivot Function 4](#_Toc160142795)

# 1 Overview

For this exercise, you will work with your project data to prepare your pivoting script for transforming your data from wide format to long format.

# 2 Step 1: Evaluating your Script File for Task Sub-setting

You should have a "champ\_all\_waves\_create\_<subtask>\_subset.R" script that reads the original raw CHAMP data file for the purpose of sub-setting it for a specific task.

Evaluate the file to ensure that your script:

1. Sources the "champ\_all\_waves\_initial\_clean.R" script (which reads and cleans the "champ\_all\_waves.Rds" data file, and saves the data frame as "champ\_all\_waves\_initial\_clean.Rds"); and
2. Reads in that created "champ\_all\_waves\_initial\_clean.Rds" data file; and
3. Contains code to subset the data based on the variables relevant to the sub-task; and
4. Writes the subset as a data file to "<subtask>\_wide.Rds"; and also
5. Contains (a) details on authorship, (b) a description of purpose, (c) annotation about sub-components of the code, and also loads libraries and functions to carry out the above tasks.

# 3 Step 2: Evaluating your Script File for Pivoting/Transforming

You should also have a "<subtask>\_wide\_to\_long.R" script that reads the data subset for the purpose of transforming the data frame from wide format to long format.

Evaluate the file to ensure that your script:

1. Sources the "champ\_all\_waves\_create\_<subtask>\_subset.R" script (which reads and subsets "champ\_all\_waves\_initial\_clean.Rds", and saves the sub-setted wide data frame as "<subtask>\_wide.Rds").
2. Reads in that created "<subtask>\_wide.Rds" data file; and
3. Contains code to pivots the data frame from wide to long; and
4. Writes the data file as "<subtask>\_long.Rds"; and also
5. Contains (a) details on authorship, (b) a description of purpose, (c) annotation about sub-components of the code, and also loads libraries and functions to carry out the above tasks.

# 4 Step 3: Adding the Pivoting/Transforming Code

You should edit your "<subtask>\_wide\_to\_long.R" script so that you can transform the data frame from wide format to long format. In the code section for performing the pivoting, you will want to pivot to long and assign to an object. Then you will want to write the long data file so that you can clean it with your cleaning script.

Here is some code to help you:

# the prefix of the columns you want to pivot (look at your data to discover this)
task\_prefix <- ""

DATA\_FRAME |>
 tidyr::pivot\_longer(
 cols = starts\_with(task\_prefix),
 names\_to = c(".value", "trial"),
 names\_pattern = "([A-Za-z]+)(\\d+)"
 )

To help you understand how this function works, see the supplemental subsection [**Understanding the Pivot Function**](#understanding-the-pivot-function).

# 5 Step 4: Cleaning your Long File

After you have written the long file, the next step would involve cleaning the data (e.g., filtering, mutating, etc.) and saving your "<subtask>\_long\_cleaned.Rds" data file. You have a script named "<subtask>\_long\_cleaned.R" that performs some of this cleaning and save that data file, although some details may be added based on your liaison’s instructions or requests of you. You can always make changes to this script as necessary and the

# 6 Step 5: Summarizing Data

After you have written the long file, you are prepared to summarize the data. You should verify how you should summarize the data for your liaison. For example, find out how you would need to group the data in order to obtain the summaries they would like to see.

You may need to create different data summaries using different numeric variables to include in a summary and by what metrics. Similarly, you may need to determine what variables, if any, you would need to group by for performing a summary.

## 6.1 General Summarizing Approaches:

## 6.2 Symmetry Span

1. Group by id and wave in order to obtain participant-level mean and median complete and partial span scores for each person for an (e.g., *participant-level aggregation*). Then, you would grouped by wave in order to obtain a group-level mean, median, standard-error of the mean, and sample size for complete and partial span scores for all people in a group (e.g., *group-level aggregation*).
2. Group-level aggregation, however, may be for all people but also for other grouping approaches (e.g., drug use, attitudes/belief, family structure, etc.). A grouping variable of interest may also exist in a different data frame (e.g., demographic questions), which will require you to join/merge the two data frames. You will need to ask your liaison specifically how they want you to aggregate, group, and summarize the data using descriptive statistics.
3. You may need to mutate new variables in order to summarize.

## 6.3 Go/No-Go

1. Group by id, wave, and go/no-go trials in order to obtain participant-level mean and median accuracy and response time data for each person for an (e.g., *participant-level aggregation*). Then, you would grouped by wave and go/no-go trials in order to obtain a group-level mean, median, standard-error of the mean, and sample size for accuracy and response time data for all people in a group (e.g., *group-level aggregation*).
2. Group-level aggregation, however, may be for all people but also for other grouping approaches (e.g., drug use, attitudes/belief, family structure, etc.). A grouping variable of interest may also exist in a different data frame (e.g., demographic questions), which will require you to join/merge the two data frames. You will need to ask your liaison specifically how they want you to aggregate, group, and summarize the data using descriptive statistics.
3. You may need to mutate new variables in order to summarize.
4. For each data summary, you may even need to/want to write them out as .Rds files so that you can access them for your R Markdown report.

# 7 Understanding the Pivot Function

(WIDE <-
 tibble::tribble(
 ~id\_subject, ~task\_correct1, ~task\_correct2,
 1, 500, 555,
 2, 545, 489,
))

## # A tibble: 2 × 3
## id\_subject task\_correct1 task\_correct2
## <dbl> <dbl> <dbl>
## 1 1 500 555
## 2 2 545 489

You might wish to pivot the tasks from wide to long. At very least, you need to specify the columns, cols.

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_")
 )

## # A tibble: 4 × 3
## id\_subject name value
## <dbl> <chr> <dbl>
## 1 1 task\_correct1 500
## 2 1 task\_correct2 555
## 3 2 task\_correct1 545
## 4 2 task\_correct2 489

The data frame pivots and by default name is used to represent the column variable containing the column information from cols.

You can change the name of the variable by passing a character string to names\_to, which will represent the new column name. If those columns were *trials*, you could specify "trials".

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = "trials",
 names\_prefix = "task\_correct"
 )

## # A tibble: 4 × 3
## id\_subject trials value
## <dbl> <chr> <dbl>
## 1 1 1 500
## 2 1 2 555
## 3 2 1 545
## 4 2 2 489

And clean up the prefix by removing the pattern passed to names\_prefix. We can remove the prefix "task\_correct".

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = "trials",
 names\_prefix = "task\_correct"
 )

## # A tibble: 4 × 3
## id\_subject trials value
## <dbl> <chr> <dbl>
## 1 1 1 500
## 2 1 2 555
## 3 2 1 545
## 4 2 2 489

This process will fail, however, when you have more than a single type of column. Let’s add additional variables to the data frame to illustrate.

(WIDE <- tibble::tribble(
 ~id\_subject, ~task\_correct1, ~task\_correct2, ~task\_stim1, ~task\_stim2,
 1, 500, 555, "word", "picture",
 2, 545, 489, "picture", "word"
))

## # A tibble: 2 × 5
## id\_subject task\_correct1 task\_correct2 task\_stim1 task\_stim2
## <dbl> <dbl> <dbl> <chr> <chr>
## 1 1 500 555 word picture
## 2 2 545 489 picture word

Because there are two different subsets of columns to pivot (e.g., correct and stim), both cannot have the same name; both columns cannot be named "trials". For example, you can try:

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = "trials"
 )

But there will be an error.

Error in `tidyr::pivot\_longer()`:
! Can't combine `task\_correct1` <double> and `task\_stim1` <character>.

We can, however, change the argument passed to names\_to from a single character string (e.g., "trials") to a vector that contains two character strings c("correct", "stimulus").

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = c("correct", "stimulus")
 )

But there will be an error.

Error in `tidyr::pivot\_longer()`:
! If you supply multiple names in `names\_to` you must also supply one of
 `names\_sep` or `names\_pattern`.

We will need to use names\_pattern to extract the two components by looking for one pattern to match the names and another pattern to match the numbers. If the column variables were composed of the word “correct” plus some number, we could search for the specific character pattern, "correct". If we select only the first three columns, we can match that specific pattern along with numbers.

* "(correct)": match the pattern with the literal characters “correct”
* "(\\d+)": match one or more digits following the letters in the column names

Putting them together:

* "(correct)(\\d+)": match “correct” followed by one or more digits

**Note:** The parentheses are used to enclose each pattern separately, otherwise the match would be for the full pattern. We want to separate the letters from the numbers.

WIDE |>
 select(1:3) |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = c(".value", "trial"),
 names\_pattern = "(correct)(\\d+)"
 )

## # A tibble: 4 × 3
## id\_subject trial correct
## <dbl> <chr> <dbl>
## 1 1 1 500
## 2 1 2 555
## 3 2 1 545
## 4 2 2 489

Because the variable names are "task\_correct1", "task\_correct2", "task\_stim1", "task\_stim2", we need a pattern that is flexible enough to match both "correct" and "stim". The solution is to match any letter in the alphabet.

* "([A-Za-z]+)": match one or more alphabetical characters that are either uppercase or lowercase
* "(\\d+)": match one or more digits following the letters in the column names.

Putting them together:

* "([A-Za-z]+)(\\d+)": match any alphabetical characters followed by any digits

WIDE |>
 tidyr::pivot\_longer(
 cols = starts\_with("task\_"),
 names\_to = c(".value", "trial"),
 names\_pattern = "([A-Za-z]+)(\\d+)"
 )

## # A tibble: 4 × 4
## id\_subject trial correct stim
## <dbl> <chr> <dbl> <chr>
## 1 1 1 500 word
## 2 1 2 555 picture
## 3 2 1 545 picture
## 4 2 2 489 word