Chapter 4 HW 4. Recoding and joins

Learning Outcomes

By the end of this HW, you should be able to:

- recode data values using the `mutate()` function
- isolate messy data using `separate()` and `case_when()`
- rearrange data using `pivot_wider()`, `pivot_longer()`, and `???_join()` functions
- realize there are several ways of getting to the results

We will also review the `select()`, `pivot_longer()`, and `summarize()` functions from last HW. The main purpose of the skills reviewed in this chapter is to wrangle your data into shape for data analysis and visualization.

4.1 Activity 1: Setup

We can work in the same project as in the previous HWs.

- Open a new R Markdown notebook: click **File > New File > R Notebook** or click on the little page icon with a green plus sign (top left).
- Give it a meaningful `title` (e.g., 'HW 4: Data wrangling') - you can also change the title later. Feel free to add an `author` field with your name or Andrew ID. Then save the file - remember, no spaces!

4.2 Activity 2: Load in the libraries and read in the data

We will use `tidyverse` today, and we want to create a data object `data_prp` that stores the data from the file `prp_data_reduced.csv` . We also want to load a data object `qrp_t1` from the file we saved in HW 3. Mine is called `2025-01-13_qrp-scores-t1.csv` but yours may be different!

You can look back at the previous HW assignment to remind yourself how to do these steps.

And remember to have a quick `glimpse()` at your data.

4.3 Activity 3: Calculate participants' confidence in understanding Open Science practices

The main goal is to compute the mean Understanding score per participant.The mean Understanding score for time point 2 has already been calculated (in the `Time2_Understanding_OS` column), but we still need to compute it for time point 1.

Looking at the Understanding data at time point 1, you determine that

- individual item columns are character, and
- according to the codebook, there are no reverse-coded items in this questionnaire.

The steps are quite similar to those for QRP, but we need to add an extra step: converting the character labels into numbers.

Again, let's do this step by step:

- **Step 1**: Select the relevant columns `Code` , and every Understanding column from time point 1 (e.g., from `Understanding_OS_1_Time1` to `Understanding_OS_12_Time1`) and store them in an object called `understanding_t1`
- **Step 2**: Pivot the data from wide format to long format using `pivot_longer()` so we can recode the labels into values (step 3) and calculate the average score (in step 4) more easily

- **Step 3**: Recode the values “Not at all confident” as 1 and “Entirely confident” as 7. All other values are already numbers. We can use functions `mutate()` in combination with `case_match()` for that
- **Step 4**: Calculate the average Understanding Open Science score (`Time1_Understanding_OS`) per participant using `group_by()` and `summarise()`

4.3.1 Steps 1 and 2: Select and pivot

Try the first 2 steps yourself using the code from Activity 4 as a template. If you get stuck, you can peek at the solution below.

▶ Solution

4.3.2 Step 3: Recoding the values

We now want to recode the values in the `Responses` column (or whatever name you picked for your column that has some of the numbers in it) so that “Not at all confident” = 1 and “Entirely confident” = 7. We want to keep all other values as they are (2-6 look already quite “numeric”).

Let’s create a new column `Responses_corrected` that stores the new values with `mutate()`. Then we can combine that with the `case_match()` function.

- The first argument in `case_match()` is the column name of the variable you want to recode.
- Then you can start recoding the values in the way of `CurrentValue ~ NewValue` (~ is a tilde). Make sure you use the `~` and not `=` !
- Lastly, the `.default` argument tells R what to do with values that are neither “Not at all confident” nor “Entirely confident”. Here, we want to replace them with the original value of the `Responses` column. In other datasets, you may want to set the default to `NA` for missing values, a character string or a number, and `case_match()` is happy to oblige.

```
understanding_t1 <- understanding_t1 %>%
  mutate(Responses_corrected = case_match(Responses, # column of the values to reco
                                          "Not at all confident" ~ 1, # values to r
                                          "Entirely confident" ~ 7,
                                          .default = Responses # all other values t
  ))
```

```
Error in `mutate()`:
ℹ In argument: `Responses_corrected = case_match(...)`.
Caused by error in `case_match()`:
! Can't combine `..1 (right)` <double> and `.default` <character>.
```

Error!!?! Can you explain what is happening here?

▶ Hint: Have a look at the error message.

So how do we fix this? Actually, there are several ways this could be done. Here are three possible solutions.

▶ **Fix option 1**

▶ **Fix option 2**

▶ **Fix option 3**

4.3.3 Your Turn

Choose the option that works best for you to **modify the code of** `understanding_t1` above that didn't work/ gave you an error message. Once you do that, you should be able to calculate the **mean Understanding Score per participant**. Store the average scores in a variable called `Time1_Understanding_OS` . If you need help, refer to the hint below or use HW 3, Activity 4 as guidance.

▶ Hint:

Finally, let's compute mean and standard deviations across the whole sample for `Time1_Understanding_OS` . You can look back at the previous HWs for examples of thus.

4.3.4 Write it up

Add a level 2 heading “Open Science Understanding Scores at Time 1” and report the mean and standard deviation scores across the sample.

4.4 Activity 4: Survey of Attitudes Towards Statistics (SATS-28)

The main goal is to compute the mean SATS-28 score for each of the 4 subscales per participant for time point 1. Looking at the SATS data at time point 1, you determine that

- individual item columns are numeric, and
- according to the codebook, there are some reverse-coded items in this questionnaire.
- Additionally, we are looking to compute the means for the 4 different subscales of the SAT-28 which are **Affect**, **CognitiveCompetence**, **Value**, and **Difficulty**.

This scenario is slightly more tricky than the previous ones due to the reverse-coding and the 4 subscales. So, let’s tackle this step by step again:

- **Step 1**: Select the relevant columns `Code`, and every SATS28 column from time point 1 (e.g., from `SATS28_1_Affect_Time1` to `SATS28_28_Difficulty_Time1`) and store them in an object called `sats_t1`
- **Step 2**: Pivot the data from wide format to long format using `pivot_longer()` so we can recode the labels into values (step 3) and calculate the average score (in step 4) more easily
- **Step 3**: We need to know which items belong to which subscale - fortunately, we have that information in the variable name and can use the `separate()` function to access it.
- **Step 4**: We need to know which items are reverse-coded and then reverse-score them - unfortunately, the info is only in the codebook and we need to find a work-around. `case_when()` can help identify and re-score the reverse-coded items.
- **Step 5**: Calculate the average SATS score per participant and subscale using `group_by()` and `summarise()`

- **Step 6**: use `pivot_wider()` to spread out the dataframe into wide format and `rename()` to tidy up the column names

4.4.1 Steps 1 and 2: select and pivot

The selecting and pivoting are exactly the same way as we already practiced in the other 2 questionnaires. Apply them here to this questionnaire.

4.4.2 Step 3: separate Subscale information

If you look at the `Items` column more closely, you can see that there is information on the `Questionnaire`, the `Item_number`, the `Subscale`, and the `Timepoint` the data was collected at.

We can separate the information into separate columns using the `separate()` function. The function's first argument is the column to separate, then define `into` which columns you want the original column to split up, and lastly, define the separator `sep` (here an underscore). For our example, we would write:

```
separate(Items, into = c("SATS", "Item_number", "Subscale", "Time"), sep = "_")
```

However, we don't need all of those columns, so we could just drop the ones we are not interested in by replacing them with `NA`.

```
separate(Items, into = c(NA, "Item_number", "Subscale", NA), sep = "_")
```

We might also add an extra argument of `convert = TRUE` to have numeric columns (i.e., `Item_number`) converted to numeric as opposed to keeping them as characters. Saves us typing a few quotation marks later in Step 4.

```
sats_t1 <- sats_t1 %>%
  # Step 3
  separate(Items, into = c(NA, "Item_number", "Subscale", NA), sep = "_", convert =
```

4.4.3 Step 4: identify reverse-coded items and then correct them

We can use `case_when()` within the `mutate()` function here to create a new column `FW_RV` that stores information on whether the item is a reverse-coded item or not.

`case_when()` works similarly to `case_match()`, however `case_match()` only allows us to "recode" values (i.e., replace one value with another), whereas `case_when()` is more flexible. It allows us to use **conditional statements** on the left side of the tilde which is useful when you want to change only *some* of the data based on specific conditions.

Looking at the codebook, it seems that items 2, 3, 4, 6, 7, 8, 9, 12, 13, 16, 17, 19, 20, 21, 23, 25, 26, 27, and 28 are reverse-coded. The rest are forward-coded.

We want to tell R now, that

- **if** the `Item_number` is any of those numbers listed above, R should write "Reverse" into the new column `FW_RV` we are creating. Since we have a few possible matches for `Item_number`, we need the Boolean expression `%in%` rather than `==`.
- **if** `Item_number` is none of those numbers, then we would like the word "Forward" in the `FW_RV` column to appear. We can achieve that by specifying a `.default` argument again, but this time we want a "word" rather than a value from another column.

```
sats_t1 <- sats_t1 %>%
  mutate(FW_RV = case_when(
    Item_number %in% c(2, 3, 4, 6, 7, 8, 9, 12, 13, 16, 17, 19, 20, 21, 23, 25, 26,
    .default = "Forward"
  ))
```

Moving on to correcting the scores: Once again, we can use `case_when ()` within the `mutate()` function to create another **conditional statement**. This time, the condition is:

- **if** `FW_RV` column has a value of "Reverse" then we would like to turn all 1 into 7, 2 into 6, etc.

- **if** `FW_RV` column has a value of “Forward” then we would like to keep the score from the `Response` column

There is a quick way and a not-so-quick way to achieve the actual **reverse-coding**.

4.4.3.1 Option 1 (quick)

The easiest way to reverse-code scores is to take the maximum value of the scale, add 1 unit, and subtract the original value. For example, on a 5-point Likert scale, it would be 6 minus the original rating; for a 7-point Likert scale, 8 minus the original rating, etc.

Here we are using a Boolean expression to check if the string “Reverse” is present in the `FW_RV` column. If this condition is `TRUE`, the value in the new column we’re creating, `Scores_corrected`, will be calculated as 8 minus the value from the Response column. If the condition is FALSE (handled by the .default argument), the original values from the `Response` column will be retained.

```
sats_t1 <- sats_t1 %>%
  mutate(Scores_corrected = case_when(
    FW_RV == "Reverse" ~ 8-Response,
    .default = Response
  ))
```

4.4.3.2 Option 2 (not so quick)

This involves using two conditional statements.

As stated above, the longer approach involves using two conditional statements. The first condition checks if the value in the `FW_RV` column is “Reverse”, while the second condition checks if the value in the `Response` column equals a specific number. **When both conditions are met**, the corresponding value on the right side of the tilde is placed in the newly created `Scores_corrected_v2` column.

For example, line 3 would read: if the value in the `FW_RV` column is “Reverse” **AND** the value in the `Response` column is 1, then assign a value of 7 to the `Scores_corrected_v2` column.

```
sats_t1 <- sats_t1 %>%
  mutate(Scores_corrected_v2 = case_when(
    FW_RV == "Reverse" & Response == 1 ~ 7,
    FW_RV == "Reverse" & Response == 2 ~ 6,
    FW_RV == "Reverse" & Response == 3 ~ 5,
    # no need to recode 4 as 4
    FW_RV == "Reverse" & Response == 5 ~ 3,
    FW_RV == "Reverse" & Response == 6 ~ 2,
    FW_RV == "Reverse" & Response == 7 ~ 1,
    .default = Response
  ))
```

As you can see now in `sats_t1`, both columns `Scores_corrected` and `Scores_corrected_v2` are identical.

4.4.3.3 Check the reverse-coding output

One way to **check whether our reverse-coding worked** is by examining the `distinct` values in the original `Response` column and comparing them with the `Scores_corrected`. We should also retain the `FW_RV` column to observe how the reverse-coding applied.

To see the patterns more clearly, we can use `arrange()` to sort the values in a meaningful order. Remember, the default sorting order is ascending, so if you want to sort values in descending order, you'll need to wrap your variable in the desc() function.

```
check_coding <- sats_t1 %>%
  distinct(FW_RV, Response, Scores_corrected) %>%
  arrange(desc(FW_RV), Response)
```

4.4.4 Step 5: Calculate mean scores

Now that we know everything worked out as intended, we can calculate the mean scores of each subscale for each participant in `sats_t1`.

▶ Hint

4.4.5 Step 6: Transform data back to wide format

The final step is to transform the data back into wide format, ensuring that each subscale has its own column. This will make it easier to join the data objects later on. In `pivot_wider()`, the first argument, `names_from`, specifies the column you want to use for your new column headings. The second argument, `values_from`, tells R which column should provide the cell values.

We should also **rename the column names** to match those in the codebook. Conveniently, we can use a function called `rename()` that works exactly like `select()` (following the pattern `new_name = old_name`), but it keeps all other column names the same rather than reducing the number of columns.

```
sats_t1 <- sats_t1 %>%
  pivot_wider(names_from = Subscale, values_from = mean_score) %>%
  rename(SATS28_Affect_Time1_mean = Affect,
         SATS28_CognitiveCompetence_Time1_mean = CognitiveCompetence,
         SATS28_Value_Time1_mean = Value,
         SATS28_Difficulty_Time1_mean = Difficulty)
```

4.4.6 Write it up

Compute the mean score and the standard deviation across the sample for each subscale. Add a level 2 heading “SATS-28 Scores at Time 1” and report the results.

4.5 Activity 5: Join everything together with `???_join()`

Time to join all the relevant data files into a single dataframe, which will be used in the next chapters on data visualization. There are four ways to join data: `inner_join()`, `left_join()`, `right_join()`, and `full_join()`. Each function behaves differently in terms of what information is retained from the two data objects. Here is a quick overview:

- `inner_join()` returns only the rows where the values in the column specified in the `by =` statement match in both tables.
- `left_join()` retains the complete first (left) table and adds values from the second (right) table that have matching values in the column specified in the `by =` statement. Rows in the left table with no match in the right table will have missing values (`NA`) in the new columns.
- `right_join()` retains the complete second (right) table and adds values from the first (left) table that have matching values in the column specified in the `by =` statement. Rows in the right table with no match in the left table will have missing values (`NA`) in the new columns.
- `full_join()` returns all rows and all columns from both tables. `NA` values fill unmatched rows.

From our original `data_prp` , we need to select demographics data and all summarised questionnaire data from time point 2. Next, we will join this with all other aggregated datasets from time point 1 which are currently stored in separate data objects in the `Global Environment` .

While you may be familiar with `inner_join()` from last year, for this task, we want to retain all data from all the data objects. Therefore, we will use `full_join()` . Keep in mind, you can only join two data objects at a time, so the upcoming code chunk will involve a fair bit of piping and joining.

Note: Since I (Gaby) like my columns arranged in a meaningful way, I will use `select()` at the end to order them better.

```
data_prp_final <- data_prp %>%
  select(Code:Plan_prereg, Pre_reg_group:Time2_Understanding_OS) %>%
  full_join(qrp_t1) %>%
  full_join(understanding_t1) %>%
  full_join(sats_t1) %>%
  select(Code:Plan_prereg, Pre_reg_group, SATS28_Affect_Time1_mean, SATS28_Cognitiv
```

4.6 Activity 6: Knit and export

Knit the `.Rmd` file to ensure everything runs as expected. Once it does, export the data object `data_prp_final` as a csv. Name it something meaningful, something like `data_prp_wrangled.csv` .

To avoid having to repeat the same steps in future, it's a good idea to save the data objects you've created today as csv files. You can do this by using the `write_csv()` function from the `readr` package. The csv files will appear in your project folder.

The basic syntax is:

```
write_csv(data_object, "filename.csv")
```

Now, let's export the object `data_prp_final` .

```
write_csv(data_prp_final, "2025-01-13_data_prp_wrangled.csv")
```

I like to name my files with a date and a "slug" so that they sort in order and are easily read. However, feel free to choose a name that makes sense to you.

4.7 Check for completeness

For this HW, upload your R notebook (.Rmd file), and your Preview html file (.nb.html). Both should include:

- Code chunks that load the tidyverse packages, read in the data, and inspect it.
- Your code that computes the OSP understanding and SATS-28 scores, and your sentences reporting them.
- Any notes for yourself about things you figured out along the way

All code in your R Notebook should run successfully.