Chapter 3 Data Skills HW 3: Data wrangling and summary statistics

By Gaby Mahrholz and Carolina E. Kuepper-Tetzel, adapted by Abigail Noyce

Learning Outcomes

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

- apply data wrangling functions `group_by()`, `summarize()`, `select()` and `pivot_longer()` to novel datasets
- read and interpret error messages
- realize there are several ways of getting to the results

3.1 Activity 1: Setup

We can work in the same project as in the Intro to R and RStudio HW.

- 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 2: Data Skills') - you can also change the title later. Feel free to add an `author` field with your name or Andrew ID.
- Once the `.Rmd` is opened, you need to save the file.
- To save it, click **File > Save As…** or click on the little disc icon. Name it something meaningful (e.g., "data-wrangling.Rmd"). Make sure there are no spaces in the name - R is not very fond of spaces… This file will automatically be saved in your project folder (i.e.,

your working directory) so you should now see this file appear in your file viewer pane.

Remember: Don't ever save a new project **inside** another project directory. This can cause some hard-to-resolve problems.

3.2 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` .

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.

3.3 Activity 3: Calculating demographics

3.3.1 ...for the full sample using `summarize()`

The `summarize()` function is part of the tidyverse's six core data manipulation tools, alongside `group_by()` , `select()` , `filter()` , `mutate()` , and `arrange()` . Remember that we can use the pipe syntax `%>%` to pass data objects from one function to another.

Within `summarize()` , we can use the `n()` function, which calculates the number of rows in the dataset. Since each row corresponds to a unique participant, this gives us the total number of participants.

To calculate the mean age and the standard deviation of age, we need to use the functions `mean()` and `sd()` on the column `Age` respectively.

```
demo_total <- data_prp %>%
  summarize(n = n(), # participant number
            mean_age = mean(Age), # mean age
            sd_age = sd(Age)) # standard deviation of age
```

```
Warning: There were 2 warnings in `summarize()`.
The first warning was:
ℹ In argument: `mean_age = mean(Age)`.
Caused by warning in `mean.default()`:
! argument is not numeric or logical: returning NA
ℹ Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
demo_total
```

n	mean_age	sd_age
89	NA	NA

R did not give us an error message per se, but the output is not quite as expected either. There are `NA` values in the `mean_age` and `sd_age` columns. Looking at the warning message and at `Age`, can you explain what happened?

▶ Hint

3.3.1.1 Fixing `Age`

Might be wise to look at the unique answers in column `Age` to determine what is wrong. We can do that with the function `distinct()`.

```
age_distinct <- data_prp %>%
  distinct(Age)
```

```
age_distinct
```

One cell has the string “years” added to their number 25, which has converted the entire column into a character column.

We can easily fix this by extracting only the numbers from the column and converting it into a numeric data type. The `parse_number()` function, which is part of the `tidyverse` package, handles both steps in one go (so there’s no need to load additional packages).

We will combine this with the `mutate()` function to create a new column called `Age` (containing those numeric values), effectively replacing the old `Age` column (which had the character values).

```
data_prp <- data_prp %>%
  mutate(Age = parse_number(Age))

typeof(data_prp$Age) # fixed
```

```
[1] "double"
```

3.3.1.2 Computing summary stats

Excellent. Now that the numbers are in a numeric format, let's try calculating the demographics for the total sample again.

```
demo_total <- data_prp %>%
  summarize(n = n(), # participant number
            mean_age = mean(Age), # mean age
            sd_age = sd(Age)) # standard deviation of age

demo_total
```

n	mean_age	sd_age
89	NA	NA

Even though there's no error or warning, the table still shows `NA` values for `mean_age` and `sd_age`. So, what could possibly be wrong now?

▶ Hint

3.3.1.3 Computing summary stats - third attempt

To ensure R ignores missing values during calculations, we need to add the extra argument `na.rm = TRUE` to the `mean()` and `sd()` functions.

```
demo_total <- data_prp %>%
  summarize(n = n(), # participant number
            mean_age = mean(Age, na.rm = TRUE), # mean age
            sd_age = sd(Age, na.rm = TRUE)) # standard deviation of age

demo_total
```

n	mean_age	sd_age
89	21.88506	3.485603

3.3.2 ... per gender using `summarize()` and `group_by()`

Now we want to compute the summary statistics for each gender. The code inside the `summarize()` function remains unchanged; we just need to use the `group_by()` function beforehand to tell R that we want to compute the summary statistics for each group separately. It's also a good practice to use `ungroup()` afterwards, so you are not taking groupings forward unintentionally.

```
demo_by_gender <- data_prp %>%
  group_by(Gender) %>% # split data up into groups (here Gender)
  summarize(n = n(), # participant number
            mean_age = mean(Age, na.rm = TRUE), # mean age
            sd_age = sd(Age, na.rm = TRUE)) %>%  # standard deviation of age
  ungroup()

demo_by_gender
```

Gender	n	mean_age	sd_age
1	17	23.31250	5.770254
2	69	21.57353	2.738973
3	3	21.33333	1.154700

3.3.3 Adding percentages

Sometimes, it may be useful to calculate percentages, such as for the gender split. You can do this by adding a line within the `summarize()` function to perform the calculation. All we need to do is take the number of female, male, and non-binary participants (stored in the `n` column of `demo_by_gender`), divide it by the total number of participants (stored in the `n` column of `demo_total`), and multiply by 100. Let's add `percentage` to the `summarize()` function of `demo_by_gender`. Make sure that the code for `percentages` is placed after the value for `n` has been computed.

Accessing the value of `n` for the different gender categories is straightforward because we can refer back to it directly. However, since the total number of participants is stored in a different data object, we need to use a base R function to access it – specifically the `$` operator. To do this, you simply type the name of the data object (in this case, `demo_total`), followed by the `$` symbol (with no spaces), and then the name of the column you want to retrieve (in this case, `n`). The general pattern is `data$column`.

```
demo_by_gender <- data_prp %>%
  group_by(Gender) %>%
  summarize(n = n(),
            # n from the line above divided by n from demo_total *100
            percentage = n/demo_total$n *100,
            mean_age = mean(Age, na.rm = TRUE),
            sd_age = sd(Age, na.rm = TRUE)) %>%
  ungroup()

demo_by_gender
```

Gender	n	percentage	mean_age	sd_age
1	17	19.101124	23.31250	5.770254
2	69	77.528090	21.57353	2.738973
3	3	3.370786	21.33333	1.154700

3.3.4 Write it up

Using either an inline code chunk, or just copying the values, create a level 2 heading "Participant demographics". For each gender category, report the number of participants and their mean age and standard deviation. (Hint: you may need to look at the codebook to interpret the gender values.)

3.4 Activity 4: Scoring Questionable Research Practices (QRPs)

This questionaire asks participants to report how acceptable they find a number of questionable research practices (QRPs). The main goal is to compute the mean QRP score per participant for time point 1. At the moment, the data is in wide format. The table below shows data from the first 3 participants:

```
head(data_prp, n = 3)
```

Code	Gender	Age	Ethnicity	Secondyeargrade	Opptional_mod	Opptional_m
Tr10	2	22	White European	2	1	Research met first year
Bi07	2	20	White British	3	2	NA
SK03	2	22	White British	1	2	NA

Looking at the QRP data at time point 1, you determine that:

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

According to the codebook and the data table above, we just have to **compute the average score for QRP items 1 to 11**, since items 12 to 15 are distractor items. Seems quite straightforward.

However, as you can see in the table above, each item is in a separate column, meaning the data is in **wide format**. It would be much easier to calculate the mean scores if the items were arranged in **long format**.

Let's tackle this problem step by step. It's best to create a separate data object for this. If we tried to compute it within `data_prp`, it could quickly become messy.

- **Step 1**: Select the relevant columns `Code`, and `QRPs_1_Time1` to `QRPs_11_Time1` and store them in an object called `qrp_t1`
- **Step 2**: Pivot the data from wide format to long format using `pivot_longer()` so we can calculate the average score more easily (in step 3)
- **Step 3**: Calculate the average QRP score (`QRPs_Acceptance_Time1_mean`) per participant using `group_by()` and `summarize()`

We'll walk through each of these in detail.

3.4.1 `select()`

The select function allows to include or exclude certain variables (columns). Here we want to focus on the participant ID column (i.e., `Code`) and the QRP items at time point 1. We can either list them all individually, i.e., Code, QRPs_1_Time1, QRPs_2_Time1, QRPs_3_Time1, and so forth (you get the gist), but that would take forever to type.

A shortcut is to use the colon operator `:`. It allows us to select all columns that fall within the range of `first_column_name` to `last_column_name`. We can apply this here since the QRP items (1 to 11) are sequentially listed in `data_prp`.

```r
qrp_step1 <- data_prp %>%
  select(Code, QRPs_1_Time1:QRPs_11_Time1)

# show first 5 rows of qrp_step1
head(qrp_step1, n = 5)
```

Code	QRPs_1_Time1	QRPs_2_Time1	QRPs_3_Time1	QRPs_4_Time1	QRPs_5_T
Tr10	7	7	5	7	
Bi07	7	7	2	7	
SK03	7	7	6	6	
SM95	7	7	2	6	
St01	7	7	6	7	

3.4.2 `pivot_longer()`

As you can see, the table we got from Step 1 is in wide format. To get it into wide format, we need to define:

- the columns that need to be reshuffled from wide into long format (`col` argument). Here we selected "everything except the `Code` column", as indicated by `-Code` [minus `Code`]. However, `QRPs_1_Time1:QRPs_11_Time1` would also work and give you the exact same result.
- the `names_to` argument. R is creating a new column in which all the column names from the columns you selected in `col` will be stored in. Here we are naming this column "Items" but you could pick something equally sensible if you like.
- the `values_to` argument. R creates this second column to store all responses the participants gave to the individual questions, i.e., all the numbers in this case. We named it "Scores" here, but you could have called it something different, like "Responses"

```r
qrp_step2 <- qrp_step1 %>%
  pivot_longer(cols = -Code, names_to = "Items", values_to = "Scores")

# show first 15 rows of qrp_step2
head(qrp_step2, n = 15)
```

Code	Items	Scores
Tr10	QRPs_1_Time1	7
Tr10	QRPs_2_Time1	7
Tr10	QRPs_3_Time1	5
Tr10	QRPs_4_Time1	7
Tr10	QRPs_5_Time1	3
Tr10	QRPs_6_Time1	4
Tr10	QRPs_7_Time1	5
Tr10	QRPs_8_Time1	7
Tr10	QRPs_9_Time1	6
Tr10	QRPs_10_Time1	7
Tr10	QRPs_11_Time1	7
Bi07	QRPs_1_Time1	7
Bi07	QRPs_2_Time1	7
Bi07	QRPs_3_Time1	2
Bi07	QRPs_4_Time1	7

3.4.3 `group_by()` and `summarize()`

This follows exactly the same sequence we used when calculating descriptive statistics by gender. The only difference is that we are now grouping the data by the participant's `Code` instead of `Gender` .

`summarize()` works exactly the same way: `summarize(new_column_name = function_to_calculate_something(column_name_of_numeric_values))`

The `function_to_calculate_something` can be `mean()` , `sd()` or `sum()` for mean scores, standard deviations, or summed-up scores respectively. You could also use `min()` or `max()` if you wanted to determine the lowest or the highest score for each participant.

3.4.4 Putting it all together:

```
qrp_t1 <- data_prp %>%

  #Step 1
  select(Code, QRPs_1_Time1:QRPs_11_Time1) %>%

  # Step 2
  pivot_longer(cols = -Code, names_to = "Items", values_to = "Scores") %>%

  # Step 3
  group_by(Code) %>% # grouping by participant id
  summarize(QRPs_Acceptance_Time1_mean = mean(Scores)) %>% # calculating the averag

  ungroup() # just make it a habit
```

Finally, let's compute mean and standard deviations across the whole sample for `QRPs_Acceptance_Time1_mean` . You can look back at the examples in Activity 3 for this.

3.4.5 Write it up

Add a level 2 heading “QRP acceptance scores at Time 1” and report the mean and standard deviation scores across the sample.

3.5 Activity 5: Knitting

Once you’ve completed your R Markdown file, the final step is to “knit” it, which converts the `.Rmd` file into a HTML file. Knitting combines your code, text, and output (like tables and plots) into a single cohesive document. This is a really good way to check your code is working.

To knit the file, **click the `Preview` button** at the top of your RStudio window. The document will be generated and, depending on your setting, automatically opened in the viewer in the `Output pane` or an external browser window.

If any errors occur during knitting, RStudio will show you an error message with details to help you troubleshoot.

If you want to **intentionally keep any errors** we tackled today to keep a reference on how you solved them, you could add `error=TRUE` or `eval=FALSE` to the code chunk that isn’t running.

3.6 Activity 6: Export a data file

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(qrp_t1, "2025-01-13_qrp-scores-t1.csv")
```

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

3.7 Activity 5: Save and quit!

We’re done with all the coding activities for this HW, so save and quit.

- First, make sure your R Markdown is saved. If there are any unsaved changes then the save icon will be in blue, if it’s greyed out it means there are no unsaved changes. But just to be safe, always hit `Ctrl + s` or click `File – Save` which will save your file.
- Then, exit RStudio completely by clicking “File - Quit session” or using the shortcut “Ctrl + Q” (Windows) or “Cmd + Q” (Mac).

3.8 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 participant demographics, and your sentences reporting them.
- Your code that computes the mean QRP scores, and your sentence reporting them.
- Any notes for yourself about things you figured out along the way

All code in your R Notebook should run successfully.