Week 10 - Data Wrangling Part II

Table of contents

In contrast to the last set of activities, this section is more of a reference guide than an end-to-end cleaning example. That’s because the tools you learn here might not always pop up, or at least are unlikely to all pop up in the one data frame. Nonetheless, when you combine the functions you learn here to clean data with the functions from last week, you will be able to handle an impressive amount (probably around 80%) of any data cleaning challenges you will encounter in your research.

By the end of this session, you should be capable of the following:

* Understand the concepts of wide and long data and be capable of pivoting (i.e., transforming) from one format to another.
* Know how to merge together separate data frames into one file that you can clean.
* Identifying and handling missing data (NA values).

## Let’s Get Set Up

Let’s begin by ensuring your working environment is ready for today’s session. Open RStudio or Posit Cloud and complete the following tasks to set everything up.

## Activity 1: Set Up Your Working Directory & R Script

Make sure you have a week10 inside your course directory and you have set it as your working directory.

|  |
| --- |
| Reminder on Steps to Set Up Your Working Directory |
| 1. Click: **Session → Set Working Directory → Choose Directory** 2. Navigate to the folder you created for this course (this should be the same folder you used for previous workshops). 3. Create a new folder called week10 inside this directory. 4. Select the week10 folder and click **Open**.   Don’t forget to verify your working directory before we get started  You can always check your current working directory by typing in the following command in the console:  getwd()  [1] "C:/Users/0131045s/Desktop/Programming/R/Workshops/rintro/activities/week10p2" |

Next, we will create our ***second R script*** for today. Call your second script 10-data-wrangling-p2.

|  |
| --- |
| Reminder on creating an R script |
| 1. Go to the menu bar and select: **File → New File → R Script**  * This will open an untitled R script.  1. To save and name your script, select:  * **File→ Save As**, then enter the name: * 10-data-wrangling-p2 * Click **Save** |

## Activity 2: Installing / Loading R Packages

We only need one package for data wrangling - tidyverse.

library(tidyverse)

**REMEMBER**: If you encounter an error message like “Error in library(package name): there is no packaged calledpackage name”, you’ll need to install the package first by editing the following for your console:

install.packages("tidyverse") #replace "package name" with the actual package name

## Activity 3: Load in your datasets

You need to download several files for this activity and move them to your week 10 folder.

1. **demographics.csv**
2. **reaction\_time.csv**
3. **wide\_personality.csv**
4. **example\_missing\_data.csv**

Once they are in your week 10 folder, make sure to load each of them in R in your new script.

df\_demographics <- read.csv("demographics.csv")  
  
df\_rt <- read.csv("reaction\_time.csv")  
  
df\_personality\_wide <- read.csv("wide\_personality.csv")   
  
df\_missing <- read.csv("example\_missing\_data.csv")

## Activity 4: Transforming Data from Wide to Long

|  |
| --- |
| What is long and wide data? |
| Wide Data In wide data, each row represents a unique participant, and each column represents a separate variable.  In wide data, each row represents a unique participant, and each column represents a separate variable. Table 6.1 shows an example of data in wide format. Each row contains all the information on a specific participant across each variable collected. For example, in one row of information, we can observe that participant 2 is 25 years old, 165 centimeters tall, weighs 60kg, and has a BMI score of 22.   | ID | Age | Height | Weight | BMI | | --- | --- | --- | --- | --- | | 1 | 30 | 175 | 76 | 24.8 | | 2 | 25 | 165 | 60 | 22 | | 3 | 35 | 185 | 80 | 23.4 |   If you are like most psychologists, you are used to seeing data in wide formats. Data is often inputted in wide format in software like Excel or SPSS, as it easier for humans to read. We are used to scanning data horizontally (left-right) rather than vertically (up-down). Because each participant is in a single row, repetition in the data frame is minimized, again making it easier for us to read. Long Data In long data, each row contains a participant’s single response to a single variable. The table below illustrates data in long format. Instead of having a column for each variable, there is one column that identifies the measured variable, and another column contains the participant’s response to that variable. If multiple variables are collected, each participant has several rows, with each row representing a single response to a single variable.   | ID | Variable | Value | | --- | --- | --- | | 1 | Age | 30 | | 1 | Height | 175 | | 1 | Weight | 76 | | 1 | BMI | 24.8 | | 2 | Age | 25 | | 2 | Height | 165 | | 2 | Weight | 60 | | 2 | BMI | 22 | | 3 | Age | 35 | | 3 | Height | 185 | | 3 | Weight | 80 | | 3 | BMI | 23.4 |   Each row in Table 6.2 represents a participant’s response to a single variable. For example, in row 1, we see that participant 1 reported their age (**Variable**) as 30 (**Value**). But I have to look to other rows to see this participant’s score on other variables.  It is more difficult to scan long data to quickly capture the information that we need. However, it is often easier for computers and programming languages to work with long data. That’s why you will sometimes get long-data from external sources. |

It’s highly likely that your data will need to be in wide rather than long format for your analyses. However, there is definitely a chance that you will need to convert your data to long format for certain advanced statistical analyses. If those do situations do occur, then you can follow these steps to convert your data from wide format to long format.

### Our Dataset

For this activity, we are going to be using the df\_personality\_wide dataset

head(df\_personality\_wide)

ID Wellbeing Extraversion Neuroticism Conscientiousness Openness  
1 1 3.55 3.98 2.15 3.34 2.44  
2 2 3.82 3.29 2.83 2.76 2.83  
3 3 5.25 3.32 2.18 3.72 1.99  
4 4 4.06 3.09 2.42 3.70 4.74  
5 5 4.10 2.56 2.50 3.66 3.97  
6 6 5.37 4.43 1.65 3.55 2.10

We can see that our data is currently in wide format, where each row represents a unique participant, and each column represents a separate personality variable (and ID variable).

### Pivoting from Wide to Long with pivot\_longer() function

The pivot\_longer() function converts a wide data frame into long format. Typing ?pivot\_longer into the console provides detailed information about this function in RStudio through the Help tab in the Files Pane.

There is a lot of information that will appear in the help section. I want to draw your attention to the Usage section, which contains the arguments (inputs) that we can specify in the pivot\_longer() function.

There is a lot of potential inputs we can throw in, but I want to highlight the key arguments that you will use most of the time when you use this function

| Argument | Meaning |
| --- | --- |
| data | Here you specify the wide data frame that you want to convert to long format |
| cols | The column(s) that will be moved or altered when you pivot the data frame. |
| names\_to | The names of each variable identified in cols will be stored in a new column in our long data frame. The names\_to argument specifies the name(s) of that new column(s). |
| values\_to | The values associated with each variable identified in cols will be stored in a new column in our long data frame. The values\_to argument specifies the name of that new column. |
|  |  |

Let’s use this function on our df\_personality\_wide and call the transformed dataframe df\_personality\_long

df\_personality\_long <- pivot\_longer(  
 df\_personality\_wide,  
 cols = Wellbeing:Openness, #we will pivot everything except ID  
 names\_to = "Variable",  
 values\_to = "Response"  
)  
  
head(df\_personality\_long)

# A tibble: 6 × 3  
 ID Variable Response  
 <int> <chr> <dbl>  
1 1 Wellbeing 3.55  
2 1 Extraversion 3.98  
3 1 Neuroticism 2.15  
4 1 Conscientiousness 3.34  
5 1 Openness 2.44  
6 2 Wellbeing 3.82

One important to note is that we rarely everpivot the ID column, because that enables us to identify which participant’s score each variable. Let’s look at what happens if we do include ID in the cols argument

pivot\_longer(  
 df\_personality\_wide,  
 cols = ID:Openness, #pivot everything  
 values\_to = "Response"  
)

# A tibble: 60 × 2  
 name Response  
 <chr> <dbl>  
 1 ID 1   
 2 Wellbeing 3.55  
 3 Extraversion 3.98  
 4 Neuroticism 2.15  
 5 Conscientiousness 3.34  
 6 Openness 2.44  
 7 ID 2   
 8 Wellbeing 3.82  
 9 Extraversion 3.29  
10 Neuroticism 2.83  
# ℹ 50 more rows

Now we have lost our record for identifying which participant contributed to which data point. This identifies a key about using pivot\_longer() in that not EVERYTHING needs to pivoted, it depends on our analytical needs are.

|  |
| --- |
| Pivoting Image |
| include\_graphics("../../img/06-pivot-visualisation.png") |

## Activity 5: Transforming Data from Long to Wide

|  |
| --- |
| What is long and wide data? |
| Wide Data In wide data, each row represents a unique participant, and each column represents a separate variable.  In wide data, each row represents a unique participant, and each column represents a separate variable. Table 6.1 shows an example of data in wide format. Each row contains all the information on a specific participant across each variable collected. For example, in one row of information, we can observe that participant 2 is 25 years old, 165 centimeters tall, weighs 60kg, and has a BMI score of 22.   | ID | Age | Height | Weight | BMI | | --- | --- | --- | --- | --- | | 1 | 30 | 175 | 76 | 24.8 | | 2 | 25 | 165 | 60 | 22 | | 3 | 35 | 185 | 80 | 23.4 |   If you are like most psychologists, you are used to seeing data in wide formats. Data is often inputted in wide format in software like Excel or SPSS, as it easier for humans to read. We are used to scanning data horizontally (left-right) rather than vertically (up-down). Because each participant is in a single row, repetition in the data frame is minimized, again making it easier for us to read. Long Data In long data, each row contains a participant’s single response to a single variable. The table below illustrates data in long format. Instead of having a column for each variable, there is one column that identifies the measured variable, and another column contains the participant’s response to that variable. If multiple variables are collected, each participant has several rows, with each row representing a single response to a single variable.   | ID | Variable | Value | | --- | --- | --- | | 1 | Age | 30 | | 1 | Height | 175 | | 1 | Weight | 76 | | 1 | BMI | 24.8 | | 2 | Age | 25 | | 2 | Height | 165 | | 2 | Weight | 60 | | 2 | BMI | 22 | | 3 | Age | 35 | | 3 | Height | 185 | | 3 | Weight | 80 | | 3 | BMI | 23.4 |   Each row in Table 6.2 represents a participant’s response to a single variable. For example, in row 1, we see that participant 1 reported their age (**Variable**) as 30 (**Value**). But I have to look to other rows to see this participant’s score on other variables.  It is more difficult to scan long data to quickly capture the information that we need. However, it is often easier for computers and programming languages to work with long data. That’s why you will sometimes get long-data from external sources. |

There has been a growing trend/push for researchers to store their data in long format when making it available or downloadable online, as this can make data easier to find and extract using computational methods.

So it is worthwhile to know how to pivot your data from Long to Wide if you ever come across a dataset you need that is in long format.

To do this, we can use the pivot\_wider() function. Typing ?pivot\_woder into the console provides detailed information about this function in RStudio through the Help tab in the Files Pane. Check the collapse box for what that should look like.

The table below shows the key arguments of this function.

|  |
| --- |
| Help information on the pivot\_wider() function |
| The following should appear in the help pane when you type ?pivot\_wider into your console.  The arguments that we can pass to the pivot_wider() function  The arguments that we can pass to the pivot\_wider() function |

| Argument | Meaning |
| --- | --- |
| data | The long data frame that you want to convert to wide format |
| id\_cols | The columns that help identify each participant. This is often the values that are repeated in each row within a long data frame (e.g., like ID or any independent variables) |
| names\_from | When we pivot from long to wide, we will be creating new columns for each variable that we collected data on. We need to tell R where to find the names for those variables. |
| values\_from | We need to tell R where to find the values for the new columns that we are creating. |

Let’s use pivot\_wider() to convert our long\_df back into wide format.

pivot\_wider(df\_personality\_long,   
 id\_cols = ID,   
 names\_from = Variable,  
 values\_from = Response)

# A tibble: 10 × 6  
 ID Wellbeing Extraversion Neuroticism Conscientiousness Openness  
 <int> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 1 3.55 3.98 2.15 3.34 2.44  
 2 2 3.82 3.29 2.83 2.76 2.83  
 3 3 5.25 3.32 2.18 3.72 1.99  
 4 4 4.06 3.09 2.42 3.7 4.74  
 5 5 4.1 2.56 2.5 3.66 3.97  
 6 6 5.37 4.43 1.65 3.55 2.1   
 7 7 4.37 3.4 3.67 3.44 2.68  
 8 8 2.99 1.43 3.12 2.95 2.63  
 9 9 3.45 3.56 2.09 2.76 3.62  
10 10 3.64 2.62 4 2.7 2.93

Look familiar? If you compare it to our original df\_personality\_wide, you’ll notice they look exactly the same - which is what we should expect!

head(df\_personality\_wide, n = 10)

ID Wellbeing Extraversion Neuroticism Conscientiousness Openness  
1 1 3.55 3.98 2.15 3.34 2.44  
2 2 3.82 3.29 2.83 2.76 2.83  
3 3 5.25 3.32 2.18 3.72 1.99  
4 4 4.06 3.09 2.42 3.70 4.74  
5 5 4.10 2.56 2.50 3.66 3.97  
6 6 5.37 4.43 1.65 3.55 2.10  
7 7 4.37 3.40 3.67 3.44 2.68  
8 8 2.99 1.43 3.12 2.95 2.63  
9 9 3.45 3.56 2.09 2.76 3.62  
10 10 3.64 2.62 4.00 2.70 2.93

## Activity 6: Handling Missing Data

Missing values, often represented as NA in R, occur when data is not available or cannot be recorded for certain observations. This can occur due to various reasons such as non-response in surveys, data entry errors, or incomplete data collection processes. It’s essential to understand and address missing values appropriately to avoid biased or misleading results in data analysis. In psychological research, dealing with missing data is a common challenge that requires careful consideration to ensure the integrity and validity of analyses.

Detecting missing values in datasets is the first step towards handling them effectively.The quickest way to check if you have missing data in R is through the functions colSums() and is.na(). The function colSums we count up the number of certain values within each column of a data frame, and we can tell it to search for missing (NA) values through the is.na() function.

### Our Dataset

Here we are going to use the df\_missing data frame which contains data on demographic information.

head(df\_missing)

id age gender  
1 1 25 Male  
2 2 NA Female  
3 3 30 Male  
4 4 28 <NA>  
5 5 35 Female  
6 6 NA Male

We can see immediately there are some missing values, but let’s count exactly how many they are using the colnames() and is.na() function.

colSums(is.na(df\_missing))

id age gender   
 0 3 3

We can see there are no missing values for id, but we have three missing values for age and gender.

To remove rows or columns with missing values, we can use functions like drop\_na(). Here’s how we can remove rows with missing values in the age column:

df\_removed <- drop\_na(df\_missing)  
  
head(df\_removed)

id age gender  
1 1 25 Male  
2 3 30 Male  
3 5 35 Female  
4 8 22 Male

colSums(is.na(df\_removed))

id age gender   
 0 0 0

## Activity 7: Merging Data (Joining Different Datasets Together)

In psychological research, we’ll often encounter situations where data from multiple sources or studies need to be combined for analysis. We might have collected demographic information separately from participants answers on experimental tasks. We may have collected data using a variety of platforms (e.g., survey data using Qualtrics and response data using PsychoPy). Additionally, the research software tools we use might modulate the data. For example, if we run a study in Gorilla Research, then each separate task and questionnaire gets downloaded as separate files.

Merging data involves combining data frames based on their common variables. Let’s image we have two data frames called df\_demographics and df\_rt. These data frames contain information on both participants demographic information and their reaction time on a specific task and which condition they were randomly assigned to. Let’s load both of these data frames into R (make sure you have downloaded them and put them into your working directory before running the following code).

head(df\_demographics)

ID gender age  
1 EoYncPX1QK Female 24  
2 Zn1yzAeYA4 Non-Binary 21  
3 B4iCIhzgPi Male 24  
4 9sJnqQM0lo Female 23  
5 FPSgiOjwA7 Non-Binary 23  
6 0g0AFLyHCe Male 24

head(df\_rt)

ID condition mean\_rt  
1 EoYncPX1QK no caffeine 269  
2 Zn1yzAeYA4 no caffeine 206  
3 B4iCIhzgPi no caffeine 187  
4 9sJnqQM0lo no caffeine 217  
5 FPSgiOjwA7 no caffeine 160  
6 0g0AFLyHCe no caffeine 255

Ideally, we would have one merged data frame that would contain a participant’s response on all our variables. However, there are some complications with these two data frames. If you check the number of rows in each data frame, we can see there are differing number of participants.

nrow(df\_demographics)

[1] 60

nrow(df\_rt)

[1] 42

There are 60 participants in the demographics data frame whereas there are only 42 participants in the reaction time data frame. If the study was online, maybe participants gave up after completing the demographic information, maybe there was connection issues, maybe the data did not save correctly. Whatever the reason for this mismatch in participants, we need to account for this when we merge these data frames together.

I am going to show you two ways we can join these datasets: inner\_join, and left\_join

### Inner\_Join

The inner\_join() function joins together two data frames, but it will only keep the rows that have matching values in both data frames.

When we use inner\_join() we need to specify the value(s) that we want to match across both data frames. Once we do, then in the case of the df\_demographics and df\_rt data frames, what this means is that only the participants who match on that specified value(s) in both data frames will merged together.

Let’s create a merged data frame using inner\_join and call it df\_inner. The syntax for inner\_join is: inner\_join(df1, df2, by = join\_by(column(s))

df\_inner <- inner\_join(df\_demographics, df\_rt, by = "ID")  
  
head(df\_inner)

ID gender age condition mean\_rt  
1 EoYncPX1QK Female 24 no caffeine 269  
2 Zn1yzAeYA4 Non-Binary 21 no caffeine 206  
3 B4iCIhzgPi Male 24 no caffeine 187  
4 9sJnqQM0lo Female 23 no caffeine 217  
5 FPSgiOjwA7 Non-Binary 23 no caffeine 160  
6 0g0AFLyHCe Male 24 no caffeine 255

We can see that our df\_inner has combined the gender and age columns from df\_demographics with the condition and mean\_rt columns from df\_rt. When we use inner\_join the order in which specify the data frames is the order in which the columns will be added. So if we wanted the condition and mean\_rt columns to come first, then we can change the order:

inner\_join(df\_rt, df\_demographics, by = join\_by(ID))

ID condition mean\_rt gender age  
1 EoYncPX1QK no caffeine 269 Female 24  
2 Zn1yzAeYA4 no caffeine 206 Non-Binary 21  
3 B4iCIhzgPi no caffeine 187 Male 24  
4 9sJnqQM0lo no caffeine 217 Female 23  
5 FPSgiOjwA7 no caffeine 160 Non-Binary 23  
6 0g0AFLyHCe no caffeine 255 Male 24  
7 hlmrG4AyLu no caffeine 145 Female 22  
8 oOUz7EpZDf no caffeine 240 Non-Binary 24  
9 QhvvMEVq1X no caffeine 212 Male 24  
10 WHdme1YyZv no caffeine 207 Female 22  
11 yFTywINVDl no caffeine 236 Non-Binary 24  
12 E4TDInCFgc no caffeine 157 Male 23  
13 w15ouKhYjX no caffeine 139 Female 24  
14 PRHjvltTq9 no caffeine 185 Non-Binary 23  
15 T1INTYDoxW low caffeine 261 Male 23  
16 wQgCowzLTF low caffeine 229 Female 22  
17 gAPefpxFu2 low caffeine 314 Non-Binary 23  
18 kJRT78syM1 low caffeine 150 Male 24  
19 zigHVms3ZU low caffeine 205 Female 23  
20 MdaNDDZypx low caffeine 353 Non-Binary 24  
21 1kVtNcCJZR low caffeine 201 Male 21  
22 vWP6tk42hT low caffeine 245 Female 23  
23 uSbQmTf4hR low caffeine 242 Non-Binary 24  
24 F5JS8n0pw8 low caffeine 281 Male 24  
25 GNNjyhr80i low caffeine 203 Female 22  
26 gg587jx1wz low caffeine 260 Non-Binary 22  
27 QG48CYj01m low caffeine 113 Male 22  
28 QkyZzgywzF low caffeine 162 Female 23  
29 t8x4iOKwnT high caffeine 204 Non-Binary 22  
30 fAazPW5qCz high caffeine 170 Male 23  
31 Aug70zOtfX high caffeine 211 Female 22  
32 DECqK4tIyT high caffeine 114 Non-Binary 23  
33 GbdjGe0yh2 high caffeine 227 Male 24  
34 yuSrPEfg8O high caffeine 175 Female 24  
35 Vq0SB0N0gt high caffeine 177 Non-Binary 23  
36 rCqGbWbmxW high caffeine 192 Male 22  
37 crbKl4mP8f high caffeine 171 Female 22  
38 PPL7VpIJAA high caffeine 114 Non-Binary 23  
39 I8wcEVbUwc high caffeine 230 Male 22  
40 i8MDEDiJHv high caffeine 201 Female 23  
41 5ROBC8QMSK high caffeine 48 Non-Binary 22  
42 QGslHuqlDI high caffeine 255 Male 23

If we check the number of rows, we will see that it matches the number of rows in df\_rt rather than df\_demographics.

nrow(df\_inner)

[1] 42

### Left\_Join

The function left\_join keeps every participant (row) in the first data frame we feed it. It then matches participants responses in the second data frame and joins them together, once we specify a value that needs to be matched. If there is not a match on that column, then it fills the results with NA values.

Let’s create the data frame df\_left using left\_join(). The syntax for this function is: `left\_join(df1, df2, by = join\_by(ID))``

df\_left <- left\_join(df\_demographics, df\_rt, by = join\_by(ID))  
  
  
head(df\_left)

ID gender age condition mean\_rt  
1 EoYncPX1QK Female 24 no caffeine 269  
2 Zn1yzAeYA4 Non-Binary 21 no caffeine 206  
3 B4iCIhzgPi Male 24 no caffeine 187  
4 9sJnqQM0lo Female 23 no caffeine 217  
5 FPSgiOjwA7 Non-Binary 23 no caffeine 160  
6 0g0AFLyHCe Male 24 no caffeine 255

tail(df\_left) #prints out the last six rows of a data frame

ID gender age condition mean\_rt  
55 TNfJ2PV63D Female 24 <NA> NA  
56 VZjyLYJOyd Non-Binary 23 <NA> NA  
57 oZv0WxMU7K Male 21 <NA> NA  
58 T1UopshE5K Female 24 <NA> NA  
59 MZV79pikQY Non-Binary 24 <NA> NA  
60 D8j28H49Lt Male 23 <NA> NA

nrow(df\_left)

[1] 60

We can see that every participant in the df\_demographics is included inside the df\_left data frame. If that participant does not have scores on condition and mean\_rt, then NA is substituted in.

The function is called left\_join() because it joins whatever is put first (i.e., left) in the function is given priority over what comes second (i.e., right).

### Other Join Functions

There are two other join functions that you can use: right\_join and full\_join. The textbook provides more information on both. I have only left\_join and inner\_join here as they are probably the most important to know.