reshaping data

Packages used in this tutorial

- readr
- dplyr
- tidyr

How to use this tutorial

-  *add text*: type the prose verbatim into the Rmd file
-  *add code*: insert a code chunk then transcribe the R code
-  *Knit* after each addition.

preparing to reshape the data

```
# Preparing to reshape the data

For analysis, the data set should be in long form, with every column a variable and every row an observa

In a calibration, an observation is the set of conditions producing a single sensor reading in mV. For t

- observ (observation number)
- cycle (cycle number)
- test\_pt (test point number and direction)
- input\_lb (applied reference force)
- output\_mV (sensor readings)
```

Learning R Markdown:

- The "hyphen, space, text" markup in Rmd, e.g., `- observ`, creates an itemized list (bullet list). The list must start as if it were a new paragraph, with a line space between it and adjacent paragraphs.
- To print an underscore in the Rmd prose we have to "escape" the character by writing `\_`.

```
Each mV reading in a row is a separate observation. To reshape the data, we want to rewrite every mV rea

We begin this process by writing code to identify which of the columns include the character string, *cy
```

Of course, the point of having the code *find* the relevant columns instead of just subsetting the columns manually is to support reproducibility. Your data will change: the next set might have more cycles than this one; or a collaborator might change the order of the columns. Part of doing reproducible work is to anticipate reasonable differences between this data set and the next.

```
# extract the indices of the column names that include "cycle"
all_col_names  <- colnames(data_received)
is_a_cycle_col <- grep('cycle', all_col_names, ignore.case = TRUE)
```

Learning R:

- *colnames()* returns the data frame column names.
- *grep()* is a string pattern-matching function. Here, I use it to compare the string 'cycle' to the information in `all_col_names`.
- I ignore case because the testing lab might send me future data with 'Cycle' capitalized.
- For readable R code, align the assignment operator `<-` of sequential lines of code where feasible.

```
# the column indices
is_a_cycle_col
```

```
## [1] 3 4 5
```

Learning R:

- Writing a variable on a line of its own, e.g., `is_a_cycle_col`, prints its value(s)
- This output tells me that columns 3, 4, 5 have "cycle" in their columns names.
- `is_a_cycle_col` is a vector of integers. The number in brackets in the output `## [1]` is the index of the first element.

reshaping data from wide to long

We're ready to reshape `data_received` from wide form to long form. Long form is necessary for effective analysis.

```
# Reshaping data from wide to long
```

In this work, the cycle numbers (the original column headings) are gathered in one new column and the mV readings (the original column entries) are gathered in another new column. The *gather()* function from the *tidyr* package arranges each cycle and reading side by side in a new observation row.

The new data frame has as many rows as there are observations in the original table.

The columns "not gathered" remain, e.g., test_point, input_lb, with their entries copied into the new rows, maintaining the relationships described in the original data set.

```
library(tidyr)
long_data <- data_received %>%
    gather(cycle, output_mV, is_a_cycle_col)
```

Learning R:

- This code chunk could be read as, "Assign `data_received` to a new data frame `long_data`, *then* gather the columns designated by `is_a_cycle_col` into two new columns, `cycle` and `output_mV`."
- the column names are gathered in the new `cycle` column
- the mV readings are gathered in the new `output_mV`column

Examine the result.

```
long_data # print it
str(long_data) # examine its structure
summary(long_data) # examine the summary statistics of each column
```

To check your work, I've included the output of *summary()*

```
##   test_point           input_lb      cycle             output_mV
##  Length:24          Min.   :0.5   Length:24          Min.   : 8.70
##  Class :character   1st Qu.:1.5   Class :character   1st Qu.:30.70
##  Mode  :character   Median :2.5   Mode  :character   Median :49.70
##                     Mean   :2.5                      Mean   :50.01
##                     3rd Qu.:3.5                      3rd Qu.:69.40
##                     Max.   :4.5                      Max.   :91.60
##                                                      NA's   :7
```

This summary shows, first, that I have the columns I expected.

Second, all the NA values are in the mV readings column. These are not actually missing values. They rep

```
library(dplyr)
long_data <- long_data %>%
    filter(! output_mV %in% NA)
str(long_data)
```

Learning R:

- The *dplyr* package *filter()* function is a row operation that keeps all rows for which its argument is TRUE
- `%in%` returns a logical vector indicating a match or not between the arguments on either side. Thus we are comparing the contents of the output_mV column to NA
- The phrase `output_mV %in% NA` would return TRUE for all NA entries. But we want the reverse, to keep the rows that aren't NA. Thus we use the logical NOT (`!`) in front of the argument, i.e., `! output_mV %in% NA`
- *filter()* keeps the rows that are TRUE for "not NA"

It's a small enough data set, with 17 observations in 4 columns, that I can print the full set.

```
print(long_data)
```

Your output should look like this:

```
## Source: local data frame [17 x 4]
##
##    test_point input_lb   cycle output_mV
##         <chr>    <dbl>   <chr>     <dbl>
## 1        3 up      2.5 cycle_1      51.1
## 2        4 up      3.5 cycle_1      70.4
## 3        5 up      4.5 cycle_1      88.8
## 4        4 dn      3.5 cycle_1      69.4
## 5        3 dn      2.5 cycle_1      49.5
## 6        2 dn      1.5 cycle_1      30.7
## 7        1 dn      0.5 cycle_1       8.7
## 8        2 up      1.5 cycle_2      29.9
## 9        3 up      2.5 cycle_2      49.4
## 10       4 up      3.5 cycle_2      70.0
## 11       5 up      4.5 cycle_2      91.6
## 12       4 dn      3.5 cycle_2      69.0
## 13       3 dn      2.5 cycle_2      50.1
## 14       2 dn      1.5 cycle_2      30.8
## 15       1 dn      0.5 cycle_2      10.9
## 16       2 up      1.5 cycle_3      30.2
## 17       3 up      2.5 cycle_3      49.7
```

write to file

Write the long-form data to file in the data directory.

```
write_csv(long_data, "data/01_calibr_long-form-data.csv")
```

In the next step, we add some final touches to make the data tidy, ready for analysis.
