**שאלה 1**

Why are pivot\_longer() and pivot\_wider() not perfectly symmetrical? Carefully consider the following example:

stocks <- **tibble**(

year = **c**(2015, 2015, 2016, 2016),

half = **c**( 1, 2, 1, 2),

return = **c**(1.88, 0.59, 0.92, 0.17)

)

stocks %>%

**pivot\_wider**(names\_from = year, values\_from = return) %>%

**pivot\_longer**(`2015`:`2016`, names\_to = "year", values\_to = "return")

*#> # A tibble: 4 x 3*

*#> half year return*

*#> <dbl> <chr> <dbl>*

*#> 1 1 2015 1.88*

*#> 2 1 2016 0.92*

*#> 3 2 2015 0.59*

*#> 4 2 2016 0.17*

(Hint: look at the variable types and think about column names.)

pivot\_longer() has a names\_ptype argument, e.g. names\_ptype = list(year = double()). What does it do?

The functions pivot\_longer() and pivot\_wider() are not perfectly symmetrical because column type information is lost when a data frame is converted from wide to long. The function pivot\_longer() stacks multiple columns which may have had multiple data types into a single column with a single data type. This transformation throws away the individual data types of the original columns. The function pivot\_wider() creates column names from values in column. These column names will always be treated as character values by pivot\_longer() so if the original variable used to create the column names did not have a character data type, then the round-trip will not reproduce the same dataset.

In the provided example, columns have the following data types:

**glimpse**(stocks)

*#> Rows: 4*

*#> Columns: 3*

*#> $ year <dbl> 2015, 2015, 2016, 2016*

*#> $ half <dbl> 1, 2, 1, 2*

*#> $ return <dbl> 1.88, 0.59, 0.92, 0.17*

The pivot\_wider() expression pivots the table to create a data frame with years as column names, and the values in return as the column values.

stocks %>%

**pivot\_wider**(names\_from = year, values\_from = return)

*#> # A tibble: 2 x 3*

*#> half `2015` `2016`*

*#> <dbl> <dbl> <dbl>*

*#> 1 1 1.88 0.92*

*#> 2 2 0.59 0.17*

The pivot\_longer() expression unpivots the table, returning it to a tidy data frame with columns for half, year, and return.

stocks %>%

**pivot\_wider**(names\_from = year, values\_from = return)%>%

**pivot\_longer**(`2015`:`2016`, names\_to = "year", values\_to = "return")

*#> # A tibble: 4 x 3*

*#> half year return*

*#> <dbl> <chr> <dbl>*

*#> 1 1 2015 1.88*

*#> 2 1 2016 0.92*

*#> 3 2 2015 0.59*

*#> 4 2 2016 0.17*

There is one difference, in the new data frame, year has a data type of character rather than numeric. The names\_to column created from column names by pivot\_longer() will be character by default, which is usually a safe assumption, since syntactically valid-column names can only be character values.

The original data types of column which pivot\_wider() used to create the column names was not stored, so pivot\_longer() has no idea that the column names in this case should be numeric values. In the current version of tidyr, the names\_ptype argument does not convert the year column to a numeric vector, and it will raise an error.

stocks %>%

**pivot\_wider**(names\_from = year, values\_from = return)%>%

**pivot\_longer**(`2015`:`2016`, names\_to = "year", values\_to = "return",

names\_ptype = **list**(year = **double**()))

*#> Error: Can't convert <character> to <double>.*

Instead, use the names\_transform argument to pivot\_longer(), which provides a function to coerce the column to a different data type.

stocks %>%

**pivot\_wider**(names\_from = year, values\_from = return)%>%

**pivot\_longer**(`2015`:`2016`, names\_to = "year", values\_to = "return",

names\_transform = **list**(year = as.numeric))

*#> # A tibble: 4 x 3*

*#> half year return*

*#> <dbl> <dbl> <dbl>*

*#> 1 1 2015 1.88*

*#> 2 1 2016 0.92*

*#> 3 2 2015 0.59*

*#> 4 2 2016 0.17*

**שאלה 2**

Why does this code fail?

table4a %>%

**pivot\_longer**(**c**(1999, 2000), names\_to = "year", values\_to = "cases")

*#> Error: Can't subset columns that don't exist.*

*#> ✖ Locations 1999 and 2000 don't exist.*

*#> ℹ There are only 3 columns.*

The code fails because the column names 1999 and 2000 are not non-syntactic variable names.[^non-syntactic] When selecting variables from a data frame, tidyverse functions will interpret numbers, like 1999 and 2000, as column numbers. In this case, pivot\_longer() tries to select the 1999th and 2000th column of the data frame. To select the columns 1999 and 2000, the names must be surrounded in backticks (`) or provided as strings.

table4a %>%

**pivot\_longer**(**c**(`1999`, `2000`), names\_to = "year", values\_to = "cases")

*#> # A tibble: 6 x 3*

*#> country year cases*

*#> <chr> <chr> <int>*

*#> 1 Afghanistan 1999 745*

*#> 2 Afghanistan 2000 2666*

*#> 3 Brazil 1999 37737*

*#> 4 Brazil 2000 80488*

*#> 5 China 1999 212258*

*#> 6 China 2000 213766*

table4a %>%

**pivot\_longer**(**c**("1999", "2000"), names\_to = "year", values\_to = "cases")

*#> # A tibble: 6 x 3*

*#> country year cases*

*#> <chr> <chr> <int>*

*#> 1 Afghanistan 1999 745*

*#> 2 Afghanistan 2000 2666*

*#> 3 Brazil 1999 37737*

*#> 4 Brazil 2000 80488*

*#> 5 China 1999 212258*

*#> 6 China 2000 213766*

**שאלה 3**

What would happen if you widen this table? Why? How could you add a new column to uniquely identify each value?

people <- **tribble**(

~name, ~key, ~value,

*#-----------------|--------|------*

"Phillip Woods", "age", 45,

"Phillip Woods", "height", 186,

"Phillip Woods", "age", 50,

"Jessica Cordero", "age", 37,

"Jessica Cordero", "height", 156

)

**glimpse**(people)

*#> Rows: 5*

*#> Columns: 3*

*#> $ name <chr> "Phillip Woods", "Phillip Woods", "Phillip Woods", "Jessica Cor…*

*#> $ key <chr> "age", "height", "age", "age", "height"*

*#> $ value <dbl> 45, 186, 50, 37, 156*

Widening this data frame using pivot\_wider() produces columns that are lists of numeric vectors because the name and key columns do not uniquely identify rows. In particular, there are two rows with values for the age of “Phillip Woods”.

**pivot\_wider**(people, names\_from="name", values\_from = "value")

*#> Warning: Values are not uniquely identified; output will contain list-cols.*

*#> \* Use `values\_fn = list` to suppress this warning.*

*#> \* Use `values\_fn = length` to identify where the duplicates arise*

*#> \* Use `values\_fn = {summary\_fun}` to summarise duplicates*

*#> # A tibble: 2 x 3*

*#> key `Phillip Woods` `Jessica Cordero`*

*#> <chr> <list> <list>*

*#> 1 age <dbl [2]> <dbl [1]>*

*#> 2 height <dbl [1]> <dbl [1]>*

We could solve the problem by adding a row with a distinct observation count for each combination of name and key.

people2 <- people %>%

**group\_by**(name, key) %>%

**mutate**(obs = **row\_number**())

people2

*#> # A tibble: 5 x 4*

*#> # Groups: name, key [4]*

*#> name key value obs*

*#> <chr> <chr> <dbl> <int>*

*#> 1 Phillip Woods age 45 1*

*#> 2 Phillip Woods height 186 1*

*#> 3 Phillip Woods age 50 2*

*#> 4 Jessica Cordero age 37 1*

*#> 5 Jessica Cordero height 156 1*

We can make people2 wider because the combination of name and obs will uniquely identify the rows in the wide data frame.

**pivot\_wider**(people2, names\_from="name", values\_from = "value")

*#> # A tibble: 3 x 4*

*#> # Groups: key [2]*

*#> key obs `Phillip Woods` `Jessica Cordero`*

*#> <chr> <int> <dbl> <dbl>*

*#> 1 age 1 45 37*

*#> 2 height 1 186 156*

*#> 3 age 2 50 NA*

Another way to solve this problem is by keeping only distinct rows of the name and key values, and dropping duplicate rows.

people %>%

**distinct**(name, key, .keep\_all = TRUE) %>%

**pivot\_wider**(names\_from="name", values\_from = "value")

*#> # A tibble: 2 x 3*

*#> key `Phillip Woods` `Jessica Cordero`*

*#> <chr> <dbl> <dbl>*

*#> 1 age 45 37*

*#> 2 height 186 156*

However, before doing this understand why there are duplicates in the data. The duplicate values may not be just a nuisance, but may indicate deeper problems with the data.

**שאלה 4**

Tidy the simple tibble below. Do you need to make it wider or longer? What are the variables?

preg <- **tribble**(

~pregnant, ~male, ~female,

"yes", NA, 10,

"no", 20, 12

)

To tidy the preg table use pivot\_longer() to create a long table. The variables in this data are:

* sex (“female”, “male”)
* pregnant (“yes”, “no”)
* count, which is a non-negative integer representing the number of observations.

The observations in this data are unique combinations of sex and pregnancy status.

preg\_tidy <- preg %>%

**pivot\_longer**(**c**(male, female), names\_to = "sex", values\_to = "count")

preg\_tidy

*#> # A tibble: 4 x 3*

*#> pregnant sex count*

*#> <chr> <chr> <dbl>*

*#> 1 yes male NA*

*#> 2 yes female 10*

*#> 3 no male 20*

*#> 4 no female 12*

Remove the (male, pregnant) row with a missing value to simplify the tidied data frame.

preg\_tidy2 <- preg %>%

**pivot\_longer**(**c**(male, female), names\_to = "sex", values\_to = "count", values\_drop\_na = TRUE)

preg\_tidy2

*#> # A tibble: 3 x 3*

*#> pregnant sex count*

*#> <chr> <chr> <dbl>*

*#> 1 yes female 10*

*#> 2 no male 20*

*#> 3 no female 12*

This an example of turning an explicit missing value into an implicit missing value, which is discussed in the upcoming section, [Missing Values](https://r4ds.had.co.nz/tidy-data.html#missing-values-3) section. The missing (male, pregnant) row represents an implicit missing value because the value of count can be inferred from its absence. In the tidy data, we can represent rows with missing values of count either explicitly with an NA (as in preg\_tidy) or implicitly by the absence of a row (as in preg\_tidy2). But in the wide data, the missing values can only be represented explicitly.

Though we have already done enough to make the data tidy, there are some other transformations that can clean the data further. If a variable takes two values, like pregnant and sex, it is often preferable to store them as logical vectors.

preg\_tidy3 <- preg\_tidy2 %>%

**mutate**(

female = sex == "female",

pregnant = pregnant == "yes"

) %>%

**select**(female, pregnant, count)

preg\_tidy3

*#> # A tibble: 3 x 3*

*#> female pregnant count*

*#> <lgl> <lgl> <dbl>*

*#> 1 TRUE TRUE 10*

*#> 2 FALSE FALSE 20*

*#> 3 TRUE FALSE 12*

In the previous data frame, I named the logical variable representing the sex female, not sex. This makes the meaning of the variable self-documenting. If the variable were named sex with values TRUE and FALSE, without reading the documentation, we wouldn’t know whether TRUE means male or female.

Apart from some minor memory savings, representing these variables as logical vectors results in more clear and concise code. Compare the filter() calls to select non-pregnant females from preg\_tidy2 and preg\_tidy.

**filter**(preg\_tidy2, sex == "female", pregnant == "no")

*#> # A tibble: 1 x 3*

*#> pregnant sex count*

*#> <chr> <chr> <dbl>*

*#> 1 no female 12*

**filter**(preg\_tidy3, female, !pregnant)

*#> # A tibble: 1 x 3*

*#> female pregnant count*

*#> <lgl> <lgl> <dbl>*

*#> 1 TRUE FALSE 12*