**Pivot Tables with dplyr**

*## attach libraries*

library(tidyverse)

library(readxl)

library(here)

library(skimr) *# install.packages('skimr')*

library(kableExtra) *# install.packages('kableExtra')*

*## read in data*

lobsters <- read\_xlsx(here("data/lobsters.xlsx"), skip=4) – to skip the first 4 line in the excel file

To look at summary statistics we’ve used summary, which is good for numeric columns, but it doesn’t give a lot of useful information for non-numeric data. So it means it wouldn’t tell us how many unique sites there are in this dataset. To have a look there I like using the skimr package:

*# explore data*

skimr::skim(lobsters)

This skimr:: notation is a reminder to me that skim is from the skimr package. It is a nice convention: it’s a reminder to others (especially you!).

skim lets us look more at each variable. Here we can look at our character variables and see that there are 5 unique sites (in the n\_unique output). Also, I particularly like looking at missing data. There are 6 missing values in the size\_mm variable.

## **group\_by() %>% summarize()**

In R, we can create the functionality of pivot tables with the same logic: we will tell R to group by something and then summarize by something. Visually, it looks like this:

data %>%

group\_by() %>%

summarize()

### **group\_by one variable**

Let’s use group\_by() %>% summarize() with our lobsters data, just like we did in Excel. We will first group\_by year and then summarize by count, using the function n() (in the dplyr package). n() counts the number of times an observation shows up, and since this is uncounted data, this will count each row.

We can say this out loud while we write it: “take the lobsters data and then group\_by year and then summarize by count in a new column we’ll call count\_by\_year.”

lobsters %>%

group\_by(year) %>%

summarize(count\_by\_year = n())

### group\_by multiple variables

Great. Now let’s summarize by both year and site like we did in the pivot table. We are able to group\_by more than one variable. Let’s do this together:

lobsters %>%

group\_by(site, year) %>%

summarize(count\_by\_siteyear = n())

We put the site first because that is what we want as an end product. But we could easily have put year first. We saw visually what would happen when we did this in the Pivot Table.

### summarize multiple variables

We can summarize multiple variables at a time.

So far we’ve summarized the count of lobster observations. Let’s also calculate the mean and standard deviation. First let’s use the mean() function to calculate the mean. We do this within the same summarize() function, but we can add a new line to make it easier to read. Notice how when you put your curser within the parenthesis and hit return, the indentation will automatically align.

lobsters %>%

group\_by(site, year) %>%

summarize(count\_by\_siteyear = n(),

mean\_size\_mm = mean(size\_mm, na.rm=TRUE),

sd\_size\_mm = sd(size\_mm, na.rm=TRUE))

### Table formatting with kable()

There are several options for formatting tables in RMarkdown; we’ll show one here from the kableExtra package and learn more about it tomorrow.

It works nicely with the pipe operator, so we can build do this from our new object:

*## make a table with our new variable*

siteyear\_summary %>%

kable()

Write this **in Markdown** but replace the # with a backtick (`): “There are #r nrow(lobsters)# total lobsters included in this report.” Let’s knit to see what happens.

I hope you can start to imagine the possibilities. If you wanted to write which year had the most observations, or which site had a decreasing trend, you would be able to.

### Activity

1. Build from our analysis and calculate the median lobster size for each site year. Your calculation will use the size\_mm variable and function to calculate the median (Hint: ?median)
2. create and ggsave() a plot.

Then, save, commit, and push your .Rmd, .html, and .png.

Solution (no peeking):

siteyear\_summary <- lobsters %>%

group\_by(site, year) %>%

summarize(count\_by\_siteyear = n(),

mean\_size\_mm = mean(size\_mm, na.rm = TRUE),

sd\_size\_mm = sd(size\_mm, na.rm = TRUE),

median\_size\_mm = median(size\_mm, na.rm = TRUE))

## `summarise()` regrouping output by 'site' (override with `.groups` argument)

*## a ggplot option:*

ggplot(data = siteyear\_summary, aes(x = year, y = median\_size\_mm, color = site)) +

geom\_line()

How to publish Quarto using terminal:

quarto publish quarto-pub

library(gt)

exibble %>%

gt() %>%

fmt\_number(

columns = num,

decimals = 3,

use\_seps = FLASE

)

### dplyr::count()

Now that we’ve spent time with group\_by %>% summarize, there is a shortcut if you only want to summarize by count. This is with a function called count(), and it will group\_by your selected variable, count, and then also ungroup. It looks like this:

lobsters %>%

count(site, year)

*## This is the same as:*

lobsters %>%

group\_by(site, year) %>%

summarize(n = n()) %>%

ungroup()

Hey, we could update our RMarkdown text knowing this: There are #r count(lobsters)# total lobsters included in this summary.

## mutate()

There are a lot of times where you don’t want to summarize your data, but you do want to operate beyond the original data. This is often done by adding a column. We do this with the mutate() function from dplyr. Let’s try this with our original lobsters data. The sizes are in millimeters but let’s say it was important for them to be in meters. We can add a column with this calculation:

lobsters %>%

mutate(size\_m = size\_mm / 1000)

If we want to add a column that has the same value repeated, we can pass it just one value, either a number or a character string (in quotes). And let’s save this as a variable called lobsters\_detailed

lobsters\_detailed <- lobsters %>%

mutate(size\_m = size\_mm / 1000,

millenia = 2000,

observer = "Allison Horst")

## select()

We will end with one final function, select. This is how to choose, retain, and move your data by columns:

Let’s say that we want to present this data finally with only columns for date, site, and size in meters. We would do this:

lobsters\_detailed %>%

select(date, site, size\_m)

# Tidying

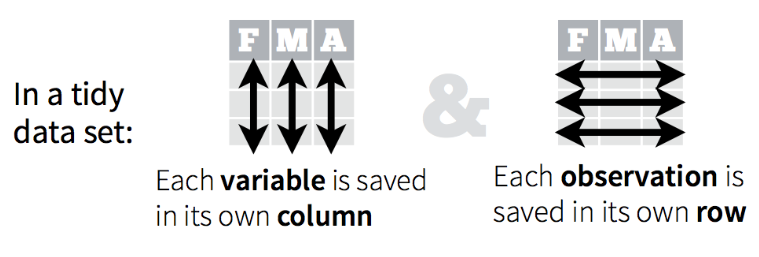
## **Summary**

In previous sessions, we learned to read in data, do some wrangling, and create a graph and table.

Here, we’ll continue by reshaping data frames (converting from long-to-wide, or wide-to-long format), separating and uniting variable (column) contents, and finding and replacing string patterns.

### Tidy data

“Tidy” might sound like a generic way to describe non-messy looking data, but it is actually a specific data structure. When data is tidy, it is rectangular with each variable as a column, each row an observation, and each cell contains a single value



### **Objectives**

In this session we’ll learn some tools to help make our data **tidy** and more coder-friendly. Those include:

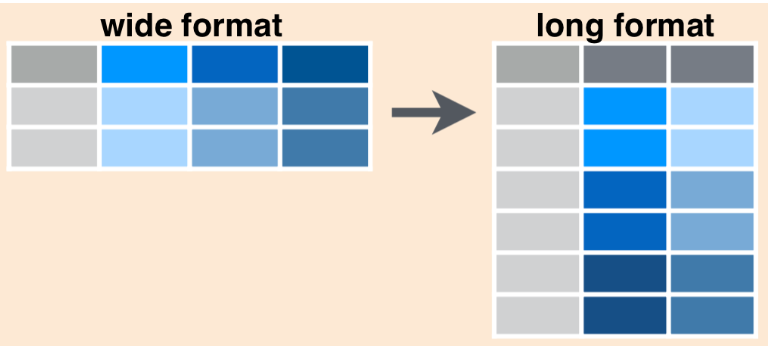
* Use tidyr::pivot\_wider() and tidyr::pivot\_longer() to reshape data frames
* janitor::clean\_names() to make column headers more manageable
* tidyr::unite() and tidyr::separate() to merge or separate information from different columns
* Detect or replace a string with stringr functions

## tidyr::pivot\_longer() to reshape from wider-to-longer format

If we look at inverts, we can see that the year variable is actually split over 3 columns, so we’d say this is currently in **wide format**.

There may be times when you want to have data in wide format, but often with code it is more efficient to convert to **long format** by gathering together observations for a variable that is currently split into multiple columns.

Schematically, converting from wide to long format using pivot\_longer() looks like this:



We’ll use tidyr::pivot\_longer() to gather data from all years in *inverts* (columns 2016, 2017, and 2018) into two columns:

* one called *year*, which contains the year
* one called *sp\_count* containing the number of each species observed.

The new data frame will be stored as *inverts\_long*:

*# Note: Either single-quotes, double-quotes, OR backticks around years work!*

inverts\_long <- pivot\_longer(data = inverts,

cols = '2016':'2018',

names\_to = "year",

values\_to = "sp\_count")

One thing that isn’t obvious at first (but would become obvious if you continued working with this data) is that since those year numbers were initially column names (characters), when they are stacked into the *year* column, their class wasn’t auto-updated to numeric.

**Explore the class of *year* in *inverts\_long*:**

class(inverts\_long$year)

## [1] "character"

That’s a good thing! We don’t want R to update classes of our data without our instruction. We’ll use dplyr::mutate() in a different way here: to create a new column (that’s how we’ve used mutate() previously) that has the same name of an existing column, in order to update and overwrite the existing column.

In this case, we’ll mutate() to add a column called *year*, which contains an as.numeric() version of the existing *year* variable:

*# Coerce "year" class to numeric:*

inverts\_long <- inverts\_long %>%

mutate(year = as.numeric(year))

Checking the class again, we see that *year* has been updated to a numeric variable:

class(inverts\_long$year)

## [1] "numeric"