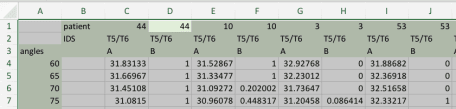
Well, here’s what I was dealing with:



*Exemplar Excel file from collaborator*

Notice that we have 3 header rows, first with patient IDs, second with spine region, and third with variable names (A and B, to protect the innocent).

**Goal**

A dataset that, for each patient and each angle gives us corresponding values of A and B. So this would be a four-column data set with ID, angle, A and B.

**Attempt 1 (readxl)**

d1 <- readxl::read\_excel('spreadsheet1.xlsx')

head(d1)

## # A tibble: 6 x 26  
## X\_\_1 patient `44` `44\_\_1` `10` `10\_\_1` `3` `3\_\_1` `53` `53\_\_1`  
##  
## 1 IDS T5/T6 T5/T6 T5/T6 T5/T6 T5/T6 T5/T6 T5/T6 T5/T6  
## 2 angles A B A B A B A B  
## 3 60 31.83… 1 31.52… 1 32.9… 0 31.8… 0  
## 4 65 31.66… 1 31.33… 1 32.2… 0 32.3… 0  
## 5 70 31.45… 1 31.09… 0.20200… 31.7… 0 32.5… 0  
## 6 75 31.08… 1 30.96… 0.44831… 31.2… 8.641… 32.3… 1  
## # … with 16 more variables: `2` , `2\_\_1` `8` ,  
## # `8\_\_1` , `6` , `6\_\_1` , `43` , `43\_\_1` ,  
## # `48` , `48\_\_1` , `46` , `46\_\_1` , `4` ,  
## # `4\_\_1` , `9` , `9\_\_1`

This strategy gives us funky column names, and pushes two of the headers into data rows. Since the headers are in rows, they’re a little harder to extract and work with. More worrisome is the fact that since the headers leaked into the data rows, the columns are all of type character rather than type numeric, which would now require further careful conversion after cleaning. So I don’t think readxl is the way to go here, if there’s a better solution.

**Attempt 2 (tidyxl)**

d2 <- tidyxl::xlsx\_cells('spreadsheet1.xlsx')

head(d2)

## # A tibble: 6 x 21  
## sheet address row col is\_blank data\_type error logical numeric  
##  
## 1 T5T6 B1 1 2 FALSE character NA NA  
## 2 T5T6 C1 1 3 FALSE numeric NA 44.  
## 3 T5T6 D1 1 4 FALSE numeric NA 44.  
## 4 T5T6 E1 1 5 FALSE numeric NA 10.  
## 5 T5T6 F1 1 6 FALSE numeric NA 10.  
## 6 T5T6 G1 1 7 FALSE numeric NA 3.  
## # … with 12 more variables: date , character ,  
## # character\_formatted , formula , is\_array ,  
## # formula\_ref , formula\_group , comment , height ,  
## # width , style\_format , local\_format\_id

The xlsx\_cells captures the data in a tidy fashion, explicitly calling out rows and columns and other metadata within each cell. We can clean up this data using tidyverse functions:

library(tidyverse)

cleanData1 = function(d) {

angle = d %>% filter(row >= 4, col == 1) %>% pull(numeric)

name = d %>% filter(row %in% c(1,3), col >= 3) %>%

mutate(character = ifelse(is.na(character),

as.character(numeric),

character)) %>%

select(row, col, character) %>%

filter(!is.na(character)) %>%

spread(row, character) %>%

unite(ID, `1`:`3`, sep = '\_') %>%

pull(ID)

data = d %>% filter(row >= 4, col >= 3) %>%

filter(!is.na(numeric)) %>%

select(row, col, numeric) %>%

spread(col, numeric) %>%

select(-row) %>%

set\_names(name) %>%

cbind(angle) %>%

gather(variable, value, -angle) %>%

separate(variable, c('ID','Measure'), sep = '\_') %>%

spread(Measure, value) %>%

select(ID, angle, A, B) %>%

arrange(ID, angle)

return(data)

}

head(cleanData1(d2))

## ID angle A B

## 1 10 60 31.52867 1.000000

## 2 10 65 31.33477 1.000000

## 3 10 70 31.09272 0.202002

## 4 10 75 30.96078 0.448317

## 5 10 80 30.79397 0.670876

## 6 10 85 30.52185 0.461406

This is a lot of data munging, and though dplyr is powerful, it took a lot of trial and error to get the final pipeline done.Nonetheless, I was really psyched about tidyxl, since it automated a job that would have taken manual manipulation (I had 12 spreadsheets like this to process). I was going to write a blog post on this cool package that made my life dealing with messy Excel file a piece of cake. But wait, there’s more…

**Attempt 3 (tidyxl + unpivotr)**

I didn’t know about unpivotr until this post:

When your spreadsheet is too for readxl, tidyxl + unpivotr helps you tackle charming features like “data as formatting” and “data in the layout”. <https://t.co/ABerpfHT8W>

— Jenny Bryan (@JennyBryan) [December 7, 2017](https://twitter.com/JennyBryan/status/938834824688689152?ref_src=twsrc%5Etfw)

So maybe all that complicated munging can be simplfied.

# devtools::install\_github('nacnudus/unpivotr')

library(unpivotr)

cleanData2 = function(d){

bl = d %>% select(row, col, data\_type, numeric, character) %>%

behead('N', ID) %>%

behead('N', spine) %>%

behead('N', variable)

# Extract the angles column

bl1 = bl %>% filter(variable == 'angles') %>% spatter(variable) %>%

select(row, angles)

# Extract the rest of the columns

bl2 = bl %>% filter(variable %in% c('A','B')) %>% select(-spine, -col) %>%

spatter(ID) %>% # Spread to columns

select(-character) %>% # All my variables are numeric

gather(ID, value, -row, -variable) %>%

spread(variable, value)

final = bl1 %>% left\_join(bl2) %>% # put things back together

arrange(ID, angles) %>%

select(ID, everything(),-row) # re-arrange columns

return(final)

}

cleanData2(d2)

## # A tibble: 588 x 4

## ID angles A B

##

## 1 10 60. 31.5 1.00

## 2 10 65. 31.3 1.00

## 3 10 70. 31.1 0.202

## 4 10 75. 31.0 0.448

## 5 10 80. 30.8 0.671

## 6 10 85. 30.5 0.461

## 7 10 90. 30.3 0.245

## 8 10 95. 30.0 0.159

## 9 10 100. 29.7 0.170

## 10 10 105. 29.2 0.421

## # ... with 578 more rows

In this example, I’m using the behead function (available in the development version of unpivotr on GitHub) to extract out the three rows of headers. Then I’m extracting out the angles column separately and merging it with the rest of the columns.

In case you’re wondering about the “N” in the behead code, unpivotr has a geographic options system as to where the headers are with respect to the main code. This [vignette](https://nacnudus.github.io/unpivotr/articles/compass-directions.html) explains this nomenclature.

**Attempt 4 (tidyxl + unpivotr)**

After re-reading the unpivotr documentation, I realized that the angles column could be treated as a row header in the unpivotr code. So I further modified the function:

cleanData3 = function(d) {

final = d %>%

select(row, col, data\_type, numeric, character) %>%

behead('N', ID) %>% # Extract column headers

behead('N', spine) %>%

behead('N', variable) %>%

behead('W', angles) %>% # angles as row header

select(numeric, ID:angles, data\_type, -spine) %>% # all vars are numeric

filter(variable %in% c'A','B')) %>% # Kills off some extra columns

spatter(variable) # Spreads, using data\_type, numeric

return(final)

}

cleanData3(d2)

## A tibble: 588 x 4

## ID angles A B

##

## 1 10 60. 31.5 1.00

## 2 10 65. 31.3 1.00

## 3 10 70. 31.1 0.202

## 4 10 75. 31.0 0.448

## 5 10 80. 30.8 0.671

## 6 10 85. 30.5 0.461

## 7 10 90. 30.3 0.245

## 8 10 95. 30.0 0.159

## 9 10 100. 29.7 0.170

##10 10 105. 29.2 0.421

## ... with 578 more rows

We get to the same output, but with much cleaner code. This is cool!!I’m going to go deeper into the unpivotr documentation and see what else can be in my regular pipeline. A big thank you to the tool-makers that create these tools that make everyday activies easier and make us stay saner.