In-class worksheet 8

Feb 8, 2018

In this worksheet, we will use the library tidyverse:

```
library(tidyverse)
```

1. Making wide tables longer

Consider the following data set, which contains information about income and religious affiliation in the US:

```
pew <- read.csv("http://wilkelab.org/classes/SDS348/data_sets/pew.csv", strings
AsFactors=F, check.names=F)
head(pew)
```

```
##             religion below10k from10to20k from20to30k from30to40k
## 1           Agnostic       27          34          60          81
## 2            Atheist       12          27          37          52
## 3           Buddhist       27          21          30          34
## 4           Catholic      418         617         732         670
## 5 Don't know/refused       15          14          15          11
## 6   Evangelical Prot      575         869        1064         982
##   from40to50k from50to75k from75to100k from100to150k above150k no_answer
## 1          76         137          122           109        84        96
## 2          35          70           73            59        74        76
## 3          33          58           62            39        53        54
## 4         638        1116          949           792       633      1489
## 5          10          35           21            17        18       116
## 6         881        1486          949           723       414      1529
```

This table is not tidy, because income levels are used as column headers rather than as levels of an `income` variable.

Use `gather()` to turn this table into a table with three columns, one for religion, one for income (called `income`), and one for the count of people with the respective combination of income and religion (called `count`).

```
pew %>% gather(income, count, below10k:no_answer) %>% head()
```

```
##             religion   income count
## 1           Agnostic below10k    27
## 2            Atheist below10k    12
## 3           Buddhist below10k    27
## 4           Catholic below10k   418
## 5 Don't know/refused below10k    15
## 6   Evangelical Prot below10k   575
```

Now call the income column `income_level` and the count column `number_of_people` .

```
pew %>% gather(income_level, number_of_people, below10k:no_answer) %>% head()
```

```
##             religion income_level number_of_people
## 1           Agnostic     below10k               27
## 2            Atheist     below10k               12
## 3           Buddhist     below10k               27
## 4           Catholic     below10k              418
## 5 Don't know/refused     below10k               15
## 6   Evangelical Prot     below10k              575
```

Now, instead of gathering data from all columns, gather only the data from columns `below10k` , `from20to30k` , and `from50to75k` , such that your final dataframe contains only these three income levels. Sort your final data frame according to religion and then income_level.

```
pew %>% select(religion, below10k, from20to30k, from50to75k) %>%
  gather(income_level, number_of_people, below10k:from50to75k) %>%
  arrange(religion, income_level) %>% head()
```

```
##   religion income_level number_of_people
## 1 Agnostic     below10k               27
## 2 Agnostic  from20to30k               60
## 3 Agnostic  from50to75k              137
## 4  Atheist     below10k               12
## 5  Atheist  from20to30k               37
## 6  Atheist  from50to75k               70
```

2. Making long tables wider

Consider the following data set, which contains information about the sex, weight, and height of 200 individuals:

```
persons <- read.csv("http://wilkelab.org/classes/SDS348/data_sets/persons.csv",
stringsAsFactors=F)
head(persons)
```

```
##   subject indicator value
## 1       1       sex     M
## 2       1    weight    77
## 3       1    height   182
## 4       2       sex     F
## 5       2    weight    58
## 6       2    height   161
```

Is this data set tidy? And can you rearrange it so that you have one column for subject, one for sex, one for weight, and one for height?

The data set is not tidy, because neither `indicator` nor `value` are variables. The variables are `subject`, `sex`, `weight`, `height`. The following version of the table is tidy:

```
persons %>% spread(indicator, value) %>% head()
```

```
##   subject height sex weight
## 1       1    182   M     77
## 2       2    161   F     58
## 3       3    161   F     53
## 4       4    177   M     68
## 5       5    157   F     59
## 6       6    170   M     76
```

For the data set `diamonds` from the ggplot2 package, create a table displaying the mean price for each combination of cut and clarity. Then use `spread()` to rearrange this table into a wide format, such that there is a column of mean prices for each cut level (Fair, Good, Very Good, etc.).

```
diamonds %>% group_by(cut, clarity) %>%
  summarize(mean.price=mean(price)) -> price.by.cut
price.by.cut
```

```
## # A tibble: 40 x 3
## # Groups:   cut [?]
##      cut clarity mean.price
##    <ord>   <ord>      <dbl>
##  1  Fair      I1   3703.533
##  2  Fair     SI2   5173.916
##  3  Fair     SI1   4208.279
##  4  Fair     VS2   4174.724
##  5  Fair     VS1   4165.141
##  6  Fair    VVS2   3349.768
##  7  Fair    VVS1   3871.353
##  8  Fair      IF   1912.333
##  9  Good      I1   3596.635
## 10  Good     SI2   4580.261
## # ... with 30 more rows
```

```
price.by.cut %>% spread(cut, mean.price)
```

```
## # A tibble: 8 x 6
##   clarity     Fair     Good `Very Good`  Premium    Ideal
## *   <ord>    <dbl>    <dbl>       <dbl>    <dbl>    <dbl>
## 1      I1 3703.533 3596.635    4078.226 3947.332 4335.726
## 2     SI2 5173.916 4580.261    4988.688 5545.937 4755.953
## 3     SI1 4208.279 3689.533    3932.391 4455.269 3752.118
## 4     VS2 4174.724 4262.236    4215.760 4550.331 3284.550
## 5     VS1 4165.141 3801.446    3805.353 4485.462 3489.744
## 6    VVS2 3349.768 3079.108    3037.765 3795.123 3250.290
## 7    VVS1 3871.353 2254.774    2459.441 2831.206 2468.129
## 8      IF 1912.333 4098.324    4396.216 3856.143 2272.913
```

3. If this was easy

Take the sepal lengths from the `iris` dataset and put them into a wide table so that is one data column per species. You might be tempted to do this with the following code, which however doesn't work. Can you explain why?

```
# If you remove the # sign in the line below you will get an error; this code d
oesn't work
# iris %>% select(Sepal.Length, Species) %>% spread(Species, Sepal.Length)
```

The problem is that `spread()` does not like to put data into the same row if it isn't sure that they actually belong together. In the iris table, there is no indication which "setosa" values, for example, should go with which "versicolor" values. Therefore, `spread()` throws an error complaining about

“duplicate identifiers for rows …”

We can avoid this issue by adding a `row` column that is repeated among the three groups. This column simply counts rows within each group, from 1 to 50 in this particular data set. This trick forces the data from the same rows for different species into one row. (That means, rows 1 of setosa, versicolor, and virginica get combined, then rows 2, and so on.)

```
iris %>% select(Sepal.Length, Species) %>%
  group_by(Species) %>%
  mutate(row = 1:n()) %>%
  spread(Species, Sepal.Length)
```

```
## # A tibble: 50 x 4
##      row setosa versicolor virginica
##  * <int>  <dbl>      <dbl>     <dbl>
##  1     1    5.1        7.0       6.3
##  2     2    4.9        6.4       5.8
##  3     3    4.7        6.9       7.1
##  4     4    4.6        5.5       6.3
##  5     5    5.0        6.5       6.5
##  6     6    5.4        5.7       7.6
##  7     7    4.6        6.3       4.9
##  8     8    5.0        4.9       7.3
##  9     9    4.4        6.6       6.7
## 10    10    4.9        5.2       7.2
## # ... with 40 more rows
```

At the end, if you want, you can delete this column again:

```
iris %>% select(Sepal.Length, Species) %>%
  group_by(Species) %>%
  mutate(row = 1:n()) %>%
  spread(Species, Sepal.Length) %>%
  select(-row)
```

```
## # A tibble: 50 x 3
##    setosa versicolor virginica
##  *  <dbl>      <dbl>     <dbl>
##  1    5.1        7.0       6.3
##  2    4.9        6.4       5.8
##  3    4.7        6.9       7.1
##  4    4.6        5.5       6.3
##  5    5.0        6.5       6.5
##  6    5.4        5.7       7.6
##  7    4.6        6.3       4.9
##  8    5.0        4.9       7.3
##  9    4.4        6.6       6.7
## 10    4.9        5.2       7.2
## # ... with 40 more rows
```