Data wrangling with dplyr, tidyr, pipes, et al

Mark Andrews

April 5, 2017

Introduction

Most time spent doing data analysis is spent processing, cleaning, formatting, manipulating data, which we will collectively refer to as data *wrangling*. A collection of tools centered around `dplyr`, `tidyr`, *pipes* etc., make wrangling a lot less painful and time consuming than it normally is.

All these tools can be loaded using the `tidyverse` package of packages.

```
library(tidyverse)
```

For the following examples, we will work with the lexical decision data.

```
Df <- read.csv('../data/LexicalDecision.csv', header=T)
```

Filtering rows

We can select all rows that meet certain criteria. For example, we might wish to filter out incorrect responses and those that are too slow. Note that `filter` returns a new data frame, and so it does not affect your original data frame (which is a good idea really):

```
fast.quick.Df <- filter(Df, accuracy == 1, latency < 2000)
```

We can see that our new data frame is a subset of the original:

```
dim(Df)
```

```
## [1] 3908    7
```

```
dim(fast.quick.Df)
```

```
## [1] 3810    7
```

The comma separated filtering conditions used above are conjunctions. We can use Boolean expressions too.

```
# Any observation where latency is not less than 500
# or greater than 1500ms
medium.speed.Df <- filter(Df, !(latency < 500 | latency > 1500) )
```

Selecting columns

Sometimes we need to make our life easier by just keeping a subset of the variables we have. This obviously especially applies to big data sets.

- Select *item*, *accuracy*, and *latency* only

```
Df.new <- select(Df, item, accuracy, latency)
head(Df.new)
```

```
##      item accuracy latency
## 1   alive        1     498
## 2 bandage        1     716
## 3  bright        1     559
## 4 carcass        1     564
## 5   cheer        1     538
## 6   coast        1     463
```

- Select all cols from *item* to *valence*

```
Df.new <- select(Df, item:valence)
head(Df.new)
```

```
##      item accuracy latency valence
## 1   alive        1     498    7.25
## 2 bandage        1     716    4.54
## 3  bright        1     559    7.50
## 4 carcass        1     564    3.34
## 5   cheer        1     538    8.10
## 6   coast        1     463    5.98
```

- Select all cols except those postitioned between *item* and *valence*:

```
Df.new <- select(Df, -(item:valence))
head(Df.new)
```

```
##   subject length frequency
## 1       1      5     42.54
## 2       1      7      2.53
## 3       1      6     55.40
## 4       1      7      1.40
## 5       1      5      7.81
## 6       1      5     47.01
```

Renaming

Rename some variables

```
Df.new <- rename(Df, correct = accuracy, rt = latency)
head(Df.new)
```

```
##   subject    item correct  rt valence length frequency
## 1       1   alive       1 498    7.25      5     42.54
## 2       1 bandage       1 716    4.54      7      2.53
## 3       1  bright       1 559    7.50      6     55.40
## 4       1 carcass       1 564    3.34      7      1.40
## 5       1   cheer       1 538    8.10      5      7.81
## 6       1   coast       1 463    5.98      5     47.01
```

Create new variable with `mutate`

We might like to create a new variable that is the log of the frequency of the word, and another that is the log of the reaction time. These are then appended to the existing columns.

```
Df.new <- mutate(Df,
                 log.latency = log(latency),
                 log.frequency = log(frequency))
head(Df.new)
```

```
##   subject    item accuracy latency valence length frequency log.latency
## 1       1   alive        1     498    7.25      5     42.54    6.210600
## 2       1 bandage        1     716    4.54      7      2.53    6.573680
## 3       1  bright        1     559    7.50      6     55.40    6.326149
## 4       1 carcass        1     564    3.34      7      1.40    6.335054
## 5       1   cheer        1     538    8.10      5      7.81    6.287859
## 6       1   coast        1     463    5.98      5     47.01    6.137727
##   log.frequency
## 1     3.7504448
## 2     0.9282193
## 3     4.0145796
## 4     0.3364722
## 5     2.0554050
## 6     3.8503603
```

You can also use `transmute` to keep just the newly created variables:

```
Df.new <- transmute(Df,
                    log.latency = log(latency),
                    log.frequency = log(frequency))
head(Df.new)
```

```
##   log.latency log.frequency
## 1    6.210600     3.7504448
## 2    6.573680     0.9282193
## 3    6.326149     4.0145796
## 4    6.335054     0.3364722
## 5    6.287859     2.0554050
## 6    6.137727     3.8503603
```

Grouping and summarizing

The base R `aggregate` is very useful when grouping and summarizing data. The `dplyr` way is to use `group_by` and then `summarize` (or `summarise` for proud British English speakers). For example, lets group by word length and then get the mean, median, sd, etc of the reaction time.

```
by_length <- group_by(Df, length)
summarise(by_length,
          mean = mean(latency),
          median = median(latency),
          sd = sd(latency),
          min = min(latency),
          max = max(latency),
          iqr = IQR(latency))
```

```
## # A tibble: 7 × 7
##   length     mean median       sd   min   max    iqr
##    <int>    <dbl>  <dbl>    <dbl> <int> <int>  <dbl>
## 1      3 545.5809  499.0 258.3921   327  3840 144.75
```

```
## 2      4 568.8735  503.0 305.6936   286  5049 133.75
## 3      5 561.3415  507.0 226.0227    38  3035 149.00
## 4      6 573.8737  530.0 196.5429   157  2392 150.00
## 5      7 629.6379  558.0 321.1949   268  4279 184.50
## 6      8 556.3384  515.5 170.1440   303  1836 127.25
## 7      9 708.7632  571.5 342.7053   402  2080 228.00
```

Pipes

Notice that above, for every operation we applied, we passed in a data frame and got a new data frame back, which we then saved under a new name. A great convenience when doing multiple operations on the same data frame is to *pipe* the output from one command to the input to another, using the pipe operator `%>%`. That way, we can chain commands together to make powerful combinations.

For example, let use filter, rename, mutate, select, and then summarize using a pipe line:

```
Df %>% filter(latency < 2000 & latency > 250) %>%
  rename(rt = latency) %>%
  mutate(log.frequency = log(frequency),
         log.rt = log(rt)) %>%
  select(item, length, accuracy, log.frequency, log.rt) %>%
  group_by(length) %>%
  summarize(mean = mean(log.rt),
            median = median(log.rt),
            iqr = IQR(log.rt))
```

```
## # A tibble: 7 × 4
##   length     mean   median       iqr
##    <int>    <dbl>    <dbl>     <dbl>
## 1      3 6.244769 6.210600 0.2822049
## 2      4 6.265550 6.218600 0.2604695
## 3      5 6.275233 6.227524 0.2815794
## 4      6 6.308966 6.272877 0.2785398
## 5      7 6.374942 6.324359 0.3124746
## 6      8 6.287654 6.245137 0.2425880
## 7      9 6.455452 6.347389 0.3564837
```

And there was much rejoicing.

Long to wide, wide to long using `tidyr`

In a *tidy* data set, every column is a variable and every row is an observation. Often your data needs to be beaten to this shape.

Let's read in a wide format data, which is a commonly used by SPSS users.

```
(Df.wide <- read.csv('../data/widedata.csv', header=T))
```

```
##   subject conditionA conditionB conditionC
## 1       1         11         14         15
## 2       2         11         12         11
## 3       3         15         11         11
## 4       4         17         11          8
## 5       5         13         13         19
```

```
## 6       6          7         10         14
## 7       7          8          9         10
```

This is fake data, but we'll pretend it gives the memory recall rate of each of 7 subjects in each of three experimental conditions. We can make this into a long, and tidy, format with `gather`. We need to specify the columns to pull together and then the name, or `key` for the newly gathered variables, and then name of the values of these variables.

```
(Df.long <- gather(Df.wide, conditionA, conditionB, conditionC, key='condition', value='recall'))
```

```
##    subject  condition recall
## 1        1 conditionA     11
## 2        2 conditionA     11
## 3        3 conditionA     15
## 4        4 conditionA     17
## 5        5 conditionA     13
## 6        6 conditionA      7
## 7        7 conditionA      8
## 8        1 conditionB     14
## 9        2 conditionB     12
## 10       3 conditionB     11
## 11       4 conditionB     11
## 12       5 conditionB     13
## 13       6 conditionB     10
## 14       7 conditionB      9
## 15       1 conditionC     15
## 16       2 conditionC     11
## 17       3 conditionC     11
## 18       4 conditionC      8
## 19       5 conditionC     19
## 20       6 conditionC     14
## 21       7 conditionC     10
```

The opposite of a `gather` is a `spread`. This converts a long to a wide format. To illustrate, we'll just go backwards from `D.long` to `Df.wide`. Here, we need only state the variable to "spread" and which variable's values to use as the values of the newly spread variables.

```
spread(Df.long, key=condition, value=recall)
```

```
##   subject conditionA conditionB conditionC
## 1       1         11         14         15
## 2       2         11         12         11
## 3       3         15         11         11
## 4       4         17         11          8
## 5       5         13         13         19
## 6       6          7         10         14
## 7       7          8          9         10
```