Data Preparation II: Filters, Transformations & Summarization using dplyr

Ethan Brown
10/13/2014

Outline

- Setting up
- `magrittr`: Feeding your your data to functions with `%>%`
- `dplyr`: data munging made elegant

Setting up

We'll be using the same data file as last week. You can download the file, move to your working directory, and then load in R using:

```
load("Data_Subs_Merge.RData")
```

You'll need to install the `dplyr` package, which you can do either through RStudio or directly in R using

```
install.packages("dplyr")
```

(This will also install `magrittr`)

Setting up

Then, load `dplyr`, which will also load `magrittr`

```
library(dplyr)
```

magrittr

Doing things in R: traditional way

Doing things in R: traditional

We can view a summary:

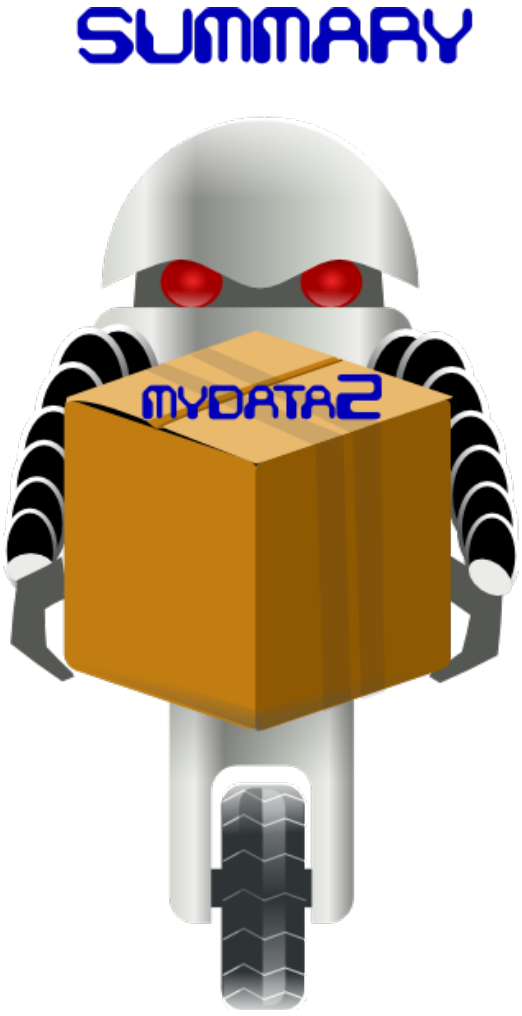
```
summary(mydata2)
```

```
##        id             age           gender      
##  Min.   : 1.00   Min.   :19.00   Min.   :0.0000  
##  1st Qu.: 3.25   1st Qu.:23.00   1st Qu.:0.0000  
##  Median : 5.50   Median :28.00   Median :1.0000  
##  Mean   : 5.50   Mean   :26.56   Mean   :0.5556  
##  3rd Qu.: 7.75   3rd Qu.:30.00   3rd Qu.:1.0000  
##  Max.   :10.00   Max.   :34.00   Max.   :1.0000  
##                  NA's   :1       NA's   :1
```

Doing things in R: traditional

Normally, we think of our data as *inside* a function

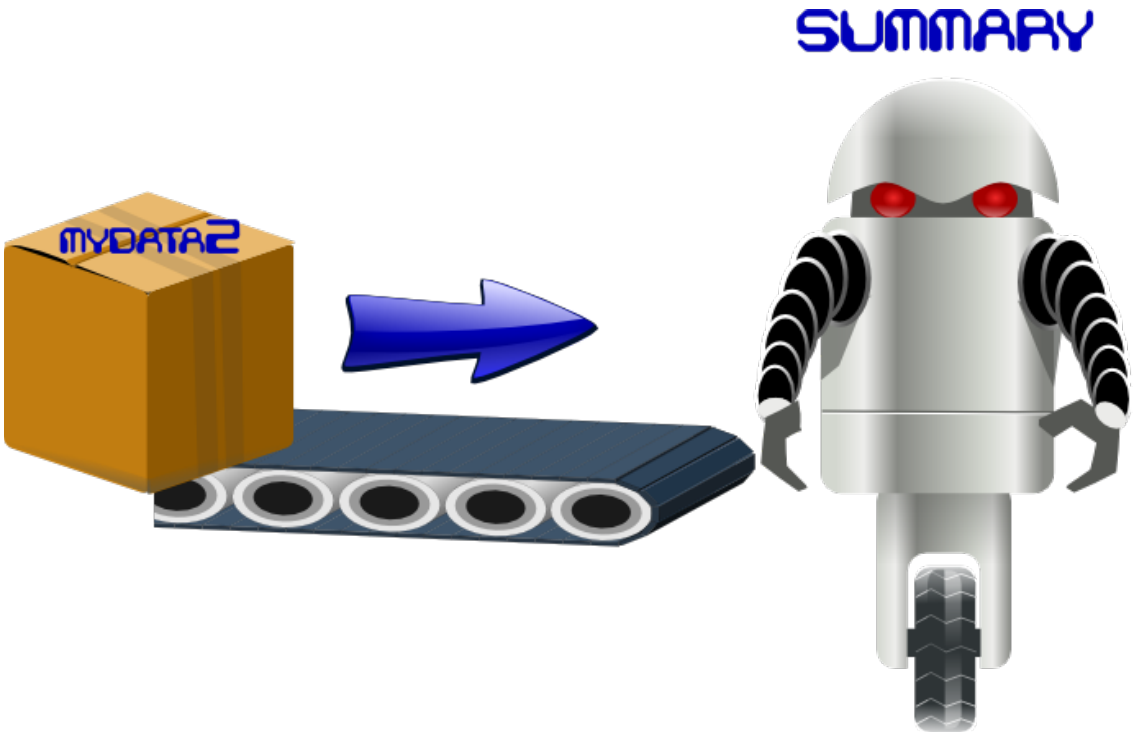
```
summary(mydata2)
```



Doing things in R: %>%

You can read %>% as "feeds into". It's like a conveyor belt, starting with the data. The output is exactly the same as `summary(mydata2)`.

```
mydata2 %>% summary
```



Doing MANY things in R: %>%

Now, we can create a flow of commands that is easy to read and write with several %>%s in a row. The output of the first function in the series is used as the input to the next function.

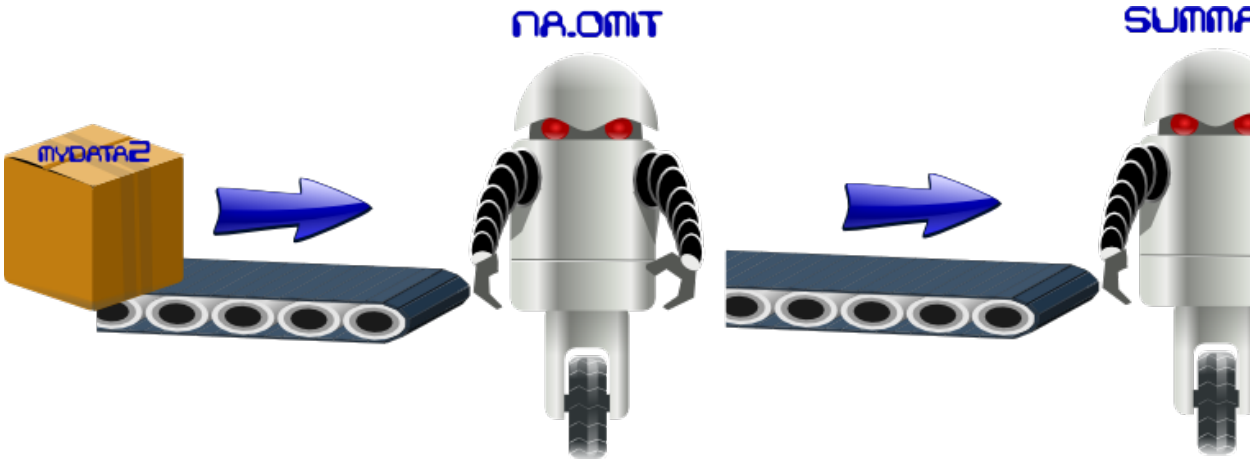
```
mydata2 %>% na.omit %>% summary
```

```
##        id             age           gender      
##  Min.   : 1.00   Min.   :19.00   Min.   :0.000  
##  1st Qu.: 2.75   1st Qu.:22.00   1st Qu.:0.000  
##  Median : 5.50   Median :27.00   Median :1.000  
##  Mean   : 5.50   Mean   :26.00   Mean   :0.625  
##  3rd Qu.: 8.25   3rd Qu.:29.25   3rd Qu.:1.000  
##  Max.   :10.00   Max.   :34.00   Max.   :1.000
```

Doing MANY things in R: %>%

So, we're feeding into `na.omit`, and then feeding the output of that into `summary`.

```
mydata2 %>% na.omit %>% summary
```

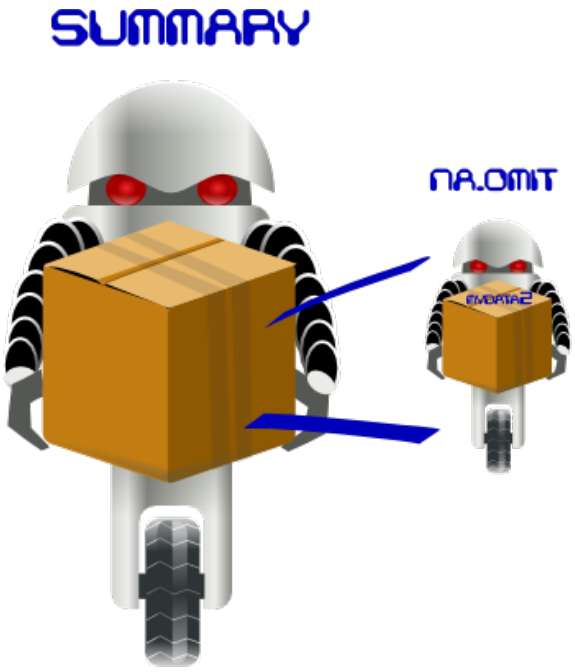


11/28

Nesting

We can do the same thing in multiple steps, or by nesting functions inside each other, but it can be harder to read (especially when using `dplyr`!)

```
summary(na.omit(mydata2))
```



Final %>% tip

%>% always assumes that what comes on the left hand side is the first argument of the function it's feeding into. Additional arguments can also be included in the function.

So,

```
summary(mydata2, digits = 1)
```

is the same as

```
mydata2 %>% summary(digits = 1)
```

dplyr

Why R needs `dplyr`

Example dataset

```
data7
```

```
##    id.student id.school id.class math.score gender
## 1           1         1      101        600      1
## 2           2         1      101        700      1
## 3           3         1      101        550      0
## 4           4         1      102        790      1
## 5           5         1      102        450      1
## 6           6         2      101        640      0
## 7           7         2      101        580      0
## 8           8         2      102        670      0
## 9           9         2      102        720      1
## 10         10         2      102        590      1
```

Why R needs `dplyr`

Summarize mean and SD by school and gender. This is pretty clumsy in base R:

```
aggregate(math.score ~ id.school + gender, data = data7,
          FUN = function(x) c(M = mean(x), SD = sd(x)))
```

```
##   id.school gender math.score.M math.score.SD
## 1         1      0    550.00000            NA
## 2         2      0    630.00000      45.82576
## 3         1      1    635.00000     145.71662
## 4         2      1    655.00000      91.92388
```

Why R needs `dplyr`

How different is each student from their school & gender mean? Just run some code like this to find out!

```
do.call(rbind,
        args = lapply(split(data7, list(data7$gender, data7$id.school
          FUN = function(x) {
           x$diff = x$math.score - mean(x$math.score)
           return(x)
       }))
```

5 `dplyr` functions that save the day

- `arrange()`: A painless way of sorting your data
- `filter()`: create subsets
- `group_by()`: tell `dplyr` the variables you want to group by
- `summarize()`: create a summary of your dataset (i.e. mean, SD)
- `mutate()`: add a new variable

arrange your data

To sort your data, just list the variables in the order you want to sort by (similar to Excel). Say, we want to sort our data for gender and then by math score:

```
data7 %>% arrange(gender, math.score)
```

```
##    id.student id.school id.class math.score gender
## 1           3         1      101        550      0
## 2           7         2      101        580      0
## 3           6         2      101        640      0
## 4           8         2      102        670      0
## 5           5         1      102        450      1
## 6          10         2      102        590      1
## 7           1         1      101        600      1
## 8           2         1      101        700      1
## 9           9         2      102        720      1
## 10          4         1      102        790      1
```

filter

Select specific rows from your data frame. To do this, specify logical conditions to filter.

```
data7 %>% filter(math.score < 600)
```

```
##   id.student id.school id.class math.score gender
## 1          3         1      101        550      0
## 2          5         1      102        450      1
## 3          7         2      101        580      0
## 4         10         2      102        590      1
```

filter

You can also filter using multiple conditions:

```
data7 %>% filter(gender == 1, math.score < 600)
```

```
##   id.student id.school id.class math.score gender
## 1          5         1      102        450      1
## 2         10         2      102        590      1
```

summarize

You can easily calculate statistics by specifying new variables with `summarize`. For instance, mean and SD:

```
data7 %>%
   summarize(M = mean(math.score),
             SD = sd(math.score))
```

```
##     M       SD
## 1 629 96.66092
```

`group_by` with `summarize`

Add a `group_by` to the command before summarize to tell `dplyr` that you want to summarize by a specific group.

```
data7 %>% group_by(id.school) %>%
  summarize(M = mean(math.score), SD = sd(math.score))
```

```
## Source: local data frame [2 x 3]
##
##   id.school   M        SD
## 1         1 618 131.79530
## 2         2 640  57.87918
```

group_by with summarize

You can use similar code to group by 2 variables:

```
data7 %>% group_by(id.school, gender) %>%
  summarize(M = mean(math.score),
            SD = sd(math.score))
```

```
## Source: local data frame [4 x 4]
## Groups: id.school
##
##   id.school gender   M        SD
## 1         1      0 550        NA
## 2         1      1 635 145.71662
## 3         2      0 630  45.82576
## 4         2      1 655  91.92388
```

mutate

Append a new variable to the original dataset with `mutate`:

```
data7 %>%
  mutate(M = mean(math.score),
         SD = sd(math.score))
```

```
##    id.student id.school id.class math.score gender   M       SD
## 1           1         1      101        600      1 629 96.66092
## 2           2         1      101        700      1 629 96.66092
## 3           3         1      101        550      0 629 96.66092
## 4           4         1      102        790      1 629 96.66092
## 5           5         1      102        450      1 629 96.66092
## 6           6         2      101        640      0 629 96.66092
## 7           7         2      101        580      0 629 96.66092
## 8           8         2      102        670      0 629 96.66092
## 9           9         2      102        720      1 629 96.66092
## 10         10         2      102        590      1 629 96.66092
```

group_by with mutate

Again, add group_by to specify which variables you'd want ot group by:

```
data7 %>% group_by(id.school) %>%
  mutate(M = mean(math.score),
         SD = sd(math.score))
```

```
## Source: local data frame [10 x 7]
## Groups: id.school
##
##    id.student id.school id.class math.score gender   M       SD
## 1           1         1      101        600      1 618 131.79530
## 2           2         1      101        700      1 618 131.79530
## 3           3         1      101        550      0 618 131.79530
## 4           4         1      102        790      1 618 131.79530
## 5           5         1      102        450      1 618 131.79530
## 6           6         2      101        640      0 640  57.87918
## 7           7         2      101        580      0 640  57.87918
## 8           8         2      102        670      0 640  57.87918
## 9           9         2      102        720      1 640  57.87918
## 10         10         2      102        590      1 640  57.87918
```

group_by with mutate

How different is each student from school & gender mean?

```
data7 %>% group_by(id.school, gender) %>%
  mutate(M = mean(math.score),
         diff.from.mean = math.score - M)
```

```
## Source: local data frame [10 x 7]
## Groups: id.school, gender
##
##    id.student id.school id.class math.score gender   M diff.from.m
## 1           1         1      101        600      1 635
## 2           2         1      101        700      1 635
## 3           3         1      101        550      0 550
## 4           4         1      102        790      1 635
## 5           5         1      102        450      1 635           -
## 6           6         2      101        640      0 630
## 7           7         2      101        580      0 630
## 8           8         2      102        670      0 630
## 9           9         2      102        720      1 655
## 10         10         2      102        590      1 655
```

Further reading

Hands-on dplyr tutorial for faster data manipulation in R

```
vignette("magrittr", package = "magrittr")
vignette("introduction", package = "dplyr")
```

28/28