Data Manipulation w dplyr and reshape2

Jessica McCloskey

September 20, 2017

Intro

- dplyr
  - verbs
  - `group_by()`
  - pipes
- reshape2
  - melt
  - cast

Asides:

- **tidyverse** (related packages/universe)
- data.table (for larger and faster)

dplyr

Verbs (dplyr)

1. `arrange()`
2. `filter()`
3. `select()`
4. `mutate()`
5. `summarise()`

Verbs (dplyr)

The first three sort and subset data, and would generally be used with conditions and/or other functions.

1. `arrange()` => ordering rows
2. `filter()` => subsetting rows
3. `select()` => subsetting columns

Verbs (dplyr)

The last two both create new columns, and would generally be used with other functions. The main difference is the format returned as output.

1. `mutate()` => output is a dataframe w/ the same number of rows as the input dataframe, and generally retains original columns by default; ("window" function)

2. `summarise()` => output may have fewer rows and fewer columns, depending on the context; ("summary" function)

Choice of which to use has more to do with what you're trying to do with the results, than with what you're trying to calculate.

Grouping (dplyr)

And … `group_by()`

This can be used with all of the verbs.

I think it's probably the most powerful aspect of the dplyr approach. You can group data based on variable values, then use the other verbs to operate on the groups.

Pipes

This is the pipe operator: `%>%`

It works with dplyr verbs and some other stuff in R, but not everything.

It's from the magrittr package. Automatically loaded with dplyr, no seperate install or library call needed.

Consider the following function notation:

$$y = f(g(h(x)))$$

Rewrite in terms of order:

$$x \text{ -> } h() \text{ -> } g() \text{ -> } f() = y$$

Pipes can be used to similarly rewrite R code!

input `%>%` first operation `%>%` second operation -> output

Putting it all Together

Verb functions can be used alone or combined with pipes.

Major difference compared with base R: “non-standard evaluation”

Generally don't need repeated references to dataframe within a command or piped together string of command … (`x` vs. `df$x`)

Check out the following slides for examples.

Focus on unemployed individuals from Dec. 2007 to Jan. 2015.

Use `filter()` to subset rows.

```
ex <- filter(df, date %in% 200712:201501 &
lfs == "U")
```

Compared to base R:

```
ex <- df[df$date %in% 200712:201501 & df$lfs
== "U", ]
```

Which states had the most and least U in January 2008? (weighted version)

```
ex %>%
 filter(date == 200801) %>%
 group_by(state) %>%
 summarize(nU = sum(wi)) %>%
 filter(nU == max(nU) | nU == min(nU))
```

```
# A tibble: 2 x 2
  state         nU
  <chr>      <dbl>
1    CA 1151.27592
2    WY    9.24853
```

unweighted => CA has max at 355, and NM has min at 20.

Create a new column giving the state's max monthly U in a given year.

```
ex %>%
group_by(state, date) %>%
mutate(mon.U = sum(wi)) %>%
group_by(state, hryear4) %>%
mutate(max.mon.U = max(mon.U)) -> ex00
```

The output `ex00` should have all orginial rows and columns from `ex` with two new rows – one giving the count of U each month, and one giving the year max. The max basically gets assigned to everyone in the state in that year.

Swap in a `summarize()` for one of the `mutate()` lines to see how output changes!!!

Replace first `mutate()` with `summarize()`

```r
 ex %>%
  group_by(state, date) %>%
  summarize(mon.U = sum(wi)) -> xx
# gives U by state & month
xx[1:3, ] # first 3 rows of the output
```

```
# A tibble: 3 x 3
# Groups:   state [1]
  state   date    mon.U
  <chr>  <dbl>    <dbl>
1    AK 200712 24.27012
2    AK 200801 27.12459
3    AK 200802 30.28619
```

Full code – note that `hryear4` included with the first `group_by()` this time. If left out, it would be dropped by the `summarize()`.

```
 ex %>%
  group_by(state, date, hryear4) %>%
  summarize(mon.U = sum(wi)) %>%
  group_by(state, hryear4) %>%
  mutate(max.mon.U = max(mon.U)) -> yy
yy[1:3, ] # first 3 rows of the output
```

```
# A tibble: 3 x 5
# Groups:   state, hryear4 [2]
  state   date hryear4    mon.U max.mon.U
  <chr>  <dbl>   <dbl>    <dbl>     <dbl>
1    AK 200712    2007 24.27012  24.27012
2    AK 200801    2008 27.12459  30.28619
3    AK 200802    2008 30.28619  30.28619
```

reshape2

Melting and Casting

Step 1: put data in melted form

```
melted_df <- melt(df, id.vars = c(ind_id,
date))
```

Step 2: cast data into any form you want

```
casted_df1 <- dcast(df, date + left_var1 ~
ind_id + right_var1)
```

```
casted_df2 <- dcast(df, ind_id + left_var1 ~
date + right_var1)
```

Will post reference sheet …

Examples with CPS …