

Manipulating Data in R

John Muschelli

June 16, 2016

Reshaping Data

In this module, we will show you how to:

1. Reshaping data from long (tall) to wide (fat)
2. Reshaping data from wide (fat) to long (tall)
3. Merging Data
4. Perform operations by a grouping variable

Setup

We will show you how to do each operation in base R then show you how to use the `dplyr` or `tidyr` package to do the same operation (if applicable).

See the “Data Wrangling Cheat Sheet using `dplyr` and `tidyr`”:

- ▶ <https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Data used: Charm City Circulator

`http://www.aejaffe.com/winterR_2016/data/Charm_City_Circulator_Ridership.csv`

```
circ = read.csv("http://www.aejaffe.com/winterR_2016/data/C  
head(circ, 2)
```

	day	date	orangeBoardings	orangeAlightings	orangeBoardings	orangeAlightings	purpleBoardings	purpleAlightings	purpleAverage	greenBoardings	greenAlightings	greenAverage	bannerBoardings	bannerAlightings	bannerAverage	daily
1	Monday	01/11/2010	877	1027												
2	Tuesday	01/12/2010	777	815												
1		NA	NA	NA												
2		NA	NA	NA												
1		NA	NA	NA												
2		NA	NA	NA												
1		NA	952													
2		NA	796													

Creating a Date class from a character date

```
library(lubridate) # great for dates!  
library(dplyr) # mutate/summarise functions  
circ = mutate(circ, date = mdy(date))  
sum( is.na(circ$date) ) # all converted correctly
```

```
[1] 0
```

```
head(circ$date)
```

```
[1] "2010-01-11" "2010-01-12" "2010-01-13" "2010-01-14" "20  
[6] "2010-01-16"
```

```
class(circ$date)
```

```
[1] "Date"
```

Making column names a little more separated

We will use `str_replace` from `stringr` to put periods in the column names.

```
library(stringr)
cn = colnames(circ)
cn = cn %>%
  str_replace("Board", ".Board") %>%
  str_replace("Alight", ".Alight") %>%
  str_replace("Average", ".Average")
colnames(circ) = cn
cn
```

[1]	"day"	"date"	"orange.Boardi
[4]	"orange.Alightings"	"orange.Average"	"purple.Boardi
[7]	"purple.Alightings"	"purple.Average"	"green.Boardin
[10]	"green.Alightings"	"green.Average"	"banner.Boardi
[13]	"banner.Alightings"	"banner.Average"	"daily"

Removing the daily ridership

We want to look at each ridership, and will remove the `daily` column:

```
circ$daily = NULL
```

Reshaping data from wide (fat) to long (tall)

See http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/

- ▶ Wide - multiple columns per observation
 - ▶ e.g. visit1, visit2, visit3

	id	visit1	visit2	visit3
1	1	10	4	3
2	2	5	6	NA

- ▶ Long - multiple rows per observation

	id	visit	value
1	1	1	10
2	1	2	4
3	1	3	3
4	2	1	5
5	2	2	6

Reshaping data from wide (fat) to long (tall): base R

The reshape command exists. It is a **confusing** function. Don't use it.

Reshaping data from wide (fat) to long (tall): tidyr

`tidyr::gather` - puts column data into rows.

We want the column names into “var” variable in the output dataset and the value in “number” variable. We then describe which columns we want to “gather:”

```
library(tidyr)
long = gather(circ, key = "var", value = "number",
              starts_with("orange"),
              starts_with("purple"),
              starts_with("green"),
              starts_with("banner"))
head(long, 2)
```

	day	date	var	number
1	Monday	2010-01-11	orange.Boardings	877
2	Tuesday	2010-01-12	orange.Boardings	777

```
table(long$var)
```

Reshaping data from wide (fat) to long (tall): tidy

Now each var is boardings, averages, or alightings. We want to separate these so we can have these by line.

```
long = separate_(long, "var",  
                  into = c("line", "type"),  
                  sep = "[.]")  
  
head(long, 3)
```

	day	date	line	type	number
1	Monday	2010-01-11	orange	Boardings	877
2	Tuesday	2010-01-12	orange	Boardings	777
3	Wednesday	2010-01-13	orange	Boardings	1203

```
unique(long$line)
```

```
[1] "orange" "purple" "green"  "banner"
```

```
unique(long$type)
```

```
[1] "Boardings" "Alightings" "Averages"
```

Finding the First (or Last) record

```
long = long %>% filter(!is.na(number) & number > 0)
first_and_last = long %>% arrange(date) %>% # arrange by date
  filter(type %in% "Boardings") %>% # keep boardings only
  group_by(line) %>% # group by line
  slice( c(1, n())) # select ("slice") first and last (n())
first_and_last %>% head(4)
```

Source: local data frame [4 x 5]

Groups: line [2]

	day (chr)	date (date)	line (chr)	type (chr)	number (dbl)
1	Monday	2012-06-04	banner	Boardings	520
2	Friday	2013-03-01	banner	Boardings	817
3	Tuesday	2011-11-01	green	Boardings	887
4	Friday	2013-03-01	green	Boardings	2592

Reshaping data from long (tall) to wide (fat): tidyrr

In tidyrr, the spread function spreads rows into columns. Now we have a long data set, but we want to separate the Average, Alightings and Boardings into different columns:

```
# have to remove missing days  
wide = filter(long, !is.na(date))  
wide = spread(wide, type, number)  
head(wide)
```

	day	date	line	Alightings	Average	Boardings
1	Friday	2010-01-15	orange	1643	1644.0	1645
2	Friday	2010-01-22	orange	1388	1394.5	1401
3	Friday	2010-01-29	orange	1322	1332.0	1342
4	Friday	2010-02-05	orange	1204	1217.5	1231
5	Friday	2010-02-12	orange	678	671.0	664
6	Friday	2010-02-19	orange	1647	1642.0	1637

Reshaping data from long (tall) to wide (fat): tidyr

We can use `rowSums` to see if any values in the row is NA and keep if the row, which is a combination of date and line type has any non-missing data.

```
# wide = wide %>%  
#   select(Alightings, Average, Boardings) %>%  
#   mutate(good = rowSums(is.na(.)) > 0)  
namat = !is.na(select(wide, Alightings, Average, Boardings))  
head(namat)
```

	Alightings	Average	Boardings
1	TRUE	TRUE	TRUE
2	TRUE	TRUE	TRUE
3	TRUE	TRUE	TRUE
4	TRUE	TRUE	TRUE
5	TRUE	TRUE	TRUE
6	TRUE	TRUE	TRUE

```
wide$good = rowSums(namat) > 0
```

Reshaping data from long (tall) to wide (fat): tidyr

Now we can filter only the good rows and delete the good column.

```
wide = filter(wide, good) %>% select(-good)
head(wide)
```

	day	date	line	Alightings	Average	Boardings
1	Friday	2010-01-15	orange	1643	1644.0	1645
2	Friday	2010-01-22	orange	1388	1394.5	1401
3	Friday	2010-01-29	orange	1322	1332.0	1342
4	Friday	2010-02-05	orange	1204	1217.5	1231
5	Friday	2010-02-12	orange	678	671.0	664
6	Friday	2010-02-19	orange	1647	1642.0	1637

Data Merging/Append in Base R

- ▶ Merging - joining data sets together - usually on key variables, usually “id”
- ▶ `merge()` is the most common way to do this with data sets
- ▶ `rbind/cbind` - row/column bind, respectively
 - ▶ `rbind` is the equivalent of “appending” in Stata or “setting” in SAS
 - ▶ `cbind` allows you to add columns in addition to the previous ways
- ▶ `t()` is a function that will transpose the data

Merging

```
base <- data.frame(id = 1:10, Age= seq(55,60, length=10))  
base[1:2,]
```

	id	Age
1	1	55.00000
2	2	55.55556

```
visits <- data.frame(id = rep(1:8, 3), visit= rep(1:3, 8),  
                     Outcome = seq(10,50, length=24))  
visits[1:2,]
```

	id	visit	Outcome
1	1	1	10.00000
2	2	2	11.73913

Merging

```
merged.data <- merge(base, visits, by="id")  
merged.data[1:5,]
```

	id	Age	visit	Outcome
1	1	55.00000	1	10.00000
2	1	55.00000	3	23.91304
3	1	55.00000	2	37.82609
4	2	55.55556	2	11.73913
5	2	55.55556	1	25.65217

```
dim(merged.data)
```

```
[1] 24 4
```

Merging

```
all.data <- merge(base, visits, by="id", all=TRUE)
tail(all.data)
```

	id	Age	visit	Outcome
21	7	58.33333	2	48.26087
22	8	58.88889	2	22.17391
23	8	58.88889	1	36.08696
24	8	58.88889	3	50.00000
25	9	59.44444	NA	NA
26	10	60.00000	NA	NA

```
dim(all.data)
```

```
[1] 26 4
```

Joining in dplyr

- ▶ ?join - see different types of joining for dplyr
- ▶ Let's look at <https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Left Join

```
lj = left_join(base, visits)
```

Joining by: "id"

```
dim(lj)
```

```
[1] 26  4
```

```
tail(lj)
```

	id	Age	visit	Outcome
21	7	58.33333	2	48.26087
22	8	58.88889	2	22.17391
23	8	58.88889	1	36.08696
24	8	58.88889	3	50.00000
25	9	59.44444	NA	NA
26	10	60.00000	NA	NA

Right Join

```
rj = right_join(base, visits)
```

Joining by: "id"

```
dim(rj)
```

```
[1] 24  4
```

```
tail(rj)
```

	id	Age	visit	Outcome
19	3	56.11111	1	41.30435
20	4	56.66667	2	43.04348
21	5	57.22222	3	44.78261
22	6	57.77778	1	46.52174
23	7	58.33333	2	48.26087
24	8	58.88889	3	50.00000

Full Join

```
fj = full_join(base, visits)
```

Joining by: "id"

```
dim(fj)
```

```
[1] 26  4
```

```
tail(fj)
```

	id	Age	visit	Outcome
21	7	58.33333	2	48.26087
22	8	58.88889	2	22.17391
23	8	58.88889	1	36.08696
24	8	58.88889	3	50.00000
25	9	59.44444	NA	NA
26	10	60.00000	NA	NA

Perform Operations By Groups: base R

The `tapply` command will take in a vector (`X`), perform a function (`FUN`) over an index (`INDEX`):

```
args(tapply)
```

```
function (X, INDEX, FUN = NULL, ..., simplify = TRUE)  
NULL
```


Perform Operations By Groups: base R

Let's get the mean Average ridership by line:

```
tapply(wide$Average, wide$line, mean, na.rm = TRUE)
```

banner	green	orange	purple
836.5637	1969.9668	3041.1924	4029.1071

Perform Operations By Groups: dplyr

Let's get the mean Average ridership by line We will use `group_by` to group the data by line, then use `summarize` (or `summarise`) to get the mean Average ridership:

```
gb = group_by(wide, line)
summarize(gb, mean_avg = mean(Average))
```

Source: local data frame [4 x 2]

	line (chr)	mean_avg (dbl)
1	banner	836.5637
2	green	1969.9668
3	orange	3041.1924
4	purple	4029.1071

Perform Operations By Groups: dplyr with piping

Using piping, this is:

```
wide %>%  
  group_by(line) %>%  
  summarise(mean_avg = mean(Average))
```

Source: local data frame [4 x 2]

	line (chr)	mean_avg (dbl)
1	banner	836.5637
2	green	1969.9668
3	orange	3041.1924
4	purple	4029.1071

Perform Operations By Multiple Groups: dplyr

This can easily be extended using `group_by` with multiple groups. Let's define the year of riding:

```
wide = wide %>% mutate(year = year(date),  
                        month = month(date))  
  
wide %>%  
  group_by(line, year) %>%  
  summarise(mean_avg = mean(Average))
```

Source: local data frame [13 x 3]

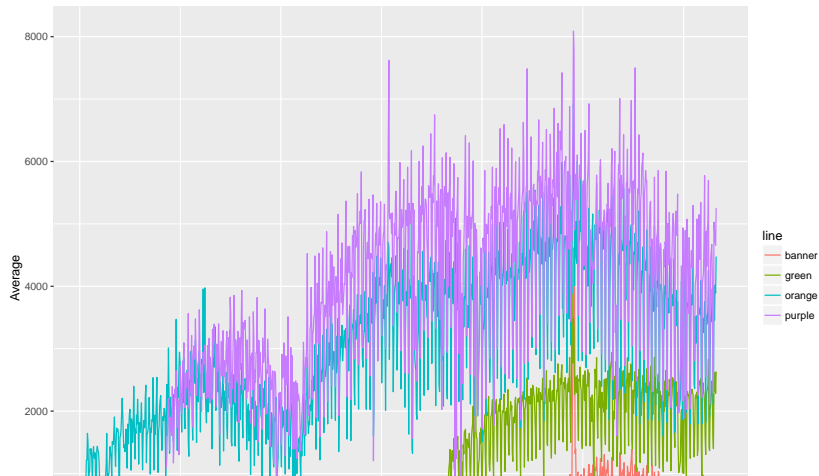
Groups: line [?]

	line (chr)	year (dbl)	mean_avg (dbl)
1	banner	2012	894.8768
2	banner	2013	635.3833
3	green	2011	1455.1667
4	green	2012	2045.5870
5	green	2013	2028.5250

Perform Operations By Multiple Groups: dplyr

We can then easily plot each day over time:

```
library(ggplot2)
ggplot(aes(x = date, y = Average,
           colour = line), data = wide) + geom_line()
```



Perform Operations By Multiple Groups: dplyr

Let's create the middle of the month (the 15th for example), and name it mon.

```
mon = wide %>%  
  dplyr::group_by(line, month, year) %>%  
  dplyr::summarise(mean_avg = mean(Average))  
mon = mutate(mon,  
              mid_month = dmy(paste0("15-", month, "-", year  
head(mon)
```

Source: local data frame [6 x 5]

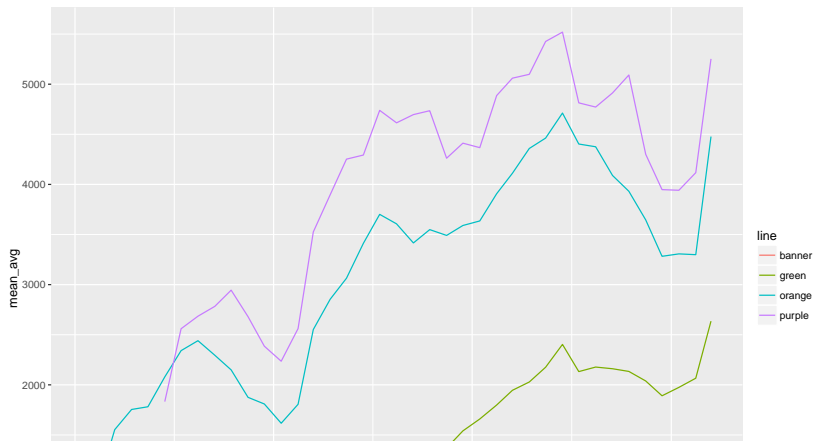
Groups: line, month [6]

	line (chr)	month (dbl)	year (dbl)	mean_avg (dbl)	mid_month (date)
1	banner	1	2013	610.3226	2013-01-15
2	banner	2	2013	656.4643	2013-02-15
3	banner	3	2013	822.0000	2013-03-15
4	banner	6	2012	1288.1296	2012-06-15

Perform Operations By Multiple Groups: dplyr

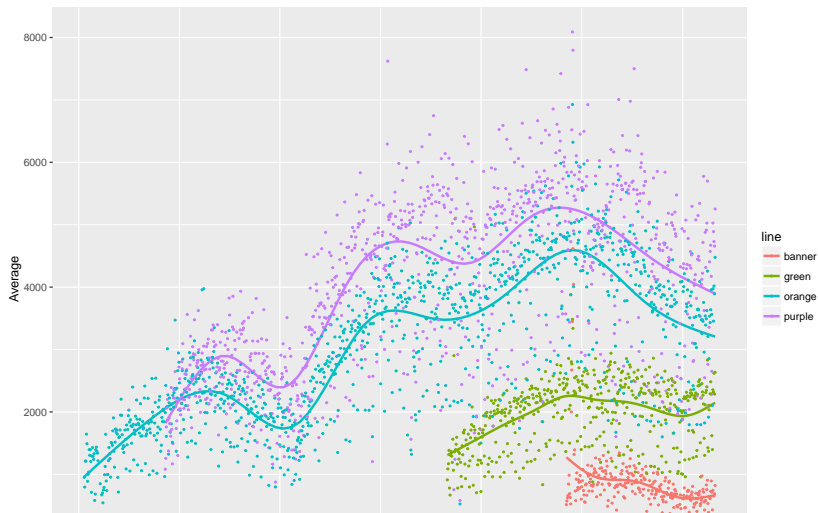
We can then easily plot the mean of each month to see a smoother output:

```
ggplot(aes(x = mid_month,  
           y = mean_avg,  
           colour = line), data = mon) + geom_line()
```



Bonus! Points with a smoother!

```
ggplot(aes(x = date, y = Average, colour = line),  
       data = wide) + geom_smooth(se = FALSE) +  
       geom_point(size = .5)
```



Extra group_by examples

group_by

`group_by` is a form of replacement for `tapply` (not a complete replacement).

Example using Bike Lanes: [http:](http://www.aejaffe.com/summerR_2016/data/Bike_Lanes.csv)

[//www.aejaffe.com/summerR_2016/data/Bike_Lanes.csv](http://www.aejaffe.com/summerR_2016/data/Bike_Lanes.csv)

```
bike = read.csv(  
  "http://www.aejaffe.com/summerR_2016/data/Bike_Lanes.csv"  
  as.is = TRUE)
```

Summarizing data with group_by and summarize

You commonly need to run group_by with summarize.

Average bike length BY project:

```
bike %>%  
  group_by(project) %>%  
  summarise(mean(length)) # get the average length
```

Source: local data frame [13 x 2]

	project (chr)	mean(length) (dbl)
1		214.3288
2	CHARM CITY CIRCULATOR	276.6658
3	COLLEGETOWN	320.6836
4	COLLEGETOWN NETWORK	213.6373
5	ENGINEERING CONSTRUCTION	512.0976
6	GUILFORD AVE BIKE BLVD	197.2782
7	MAINTENANCE	1942.1523
8	OPERATION ORANGE CONF	250.0784

Naming columns in output in summarize

Using `summarise/summarize(my_new_column_name = output)` allows you to name the column in the output:

```
bike %>%  
  group_by(project) %>%  
  summarize(mean_length = mean(length)) %>%  
  head(4) # head ONLY for slide printing
```

Source: local data frame [4 x 2]

	project (chr)	mean_length (dbl)
1		214.3288
2	CHARM CITY CIRCULATOR	276.6658
3	COLLEGETOWN	320.6836
4	COLLEGETOWN NETWORK	213.6373