

Manipulating Data in R

Introduction to R for Public Health Researchers

Reshaping Data

In this module, we will show you how to:

1. Reshaping data from wide (fat) to long (tall)
2. Reshaping data from long (tall) to wide (fat)
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>

What is wide/long data?

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

What is wide/long data?

More accurately, data is wide or long **with respect** to certain variables.

Data used: Charm City Circulator

http://johnmuschelli.com/intro_to_r/data/Charm_City_Circulator_Ridership.csv

```
circ = read_csv(  
  paste0("http://johnmuschelli.com/intro_to_r/",  
        "data/Charm_City_Circulator_Ridership.csv")  
)  
head(circ, 2)
```

```
# A tibble: 2 x 15  
   day      date orangeBoardings orangeAlightings orangeAverage  
   <chr>    <chr>          <int>           <int>           <dbl>  
1 Monday 01/11/2010          877           1027           952  
2 Tuesday 01/12/2010          777           815           796  
# ... with 10 more variables: purpleBoardings <int>,  
#   purpleAlightings <int>, purpleAverage <dbl>, greenBoardings <int>,  
#   greenAlightings <int>, greenAverage <dbl>, bannerBoardings <int>,  
#   bannerAlightings <int>, bannerAverage <dbl>, daily <dbl>
```

Creating a Date class from a character date

```
library(lubridate) # great for dates!  
library(dplyr) # mutate/summarise functions
```

Creating a Date class from a character date

```
sum(is.na(circ$date))
```

```
[1] 0
```

```
sum( circ$date == "" )
```

```
[1] 0
```

```
circ = mutate(circ, date = mdy(date))  
sum( is.na(circ$date) ) # all converted correctly
```

```
[1] 0
```

```
head(circ$date, 3)
```

```
[1] "2010-01-11" "2010-01-12" "2010-01-13"
```

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

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


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

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

tidyr package

`tidyr` allows you to "tidy" your data. We will be talking about:

- `gather` - make multiple columns into variables, (wide to long)
- `spread` - make a variable into multiple columns, (long to wide)
- `separate` - string into multiple columns
- `unite` - multiple columns into one string

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",
              -day, -date, -daily)
head(long, 4)
```

```
# A tibble: 4 x 5
   day      date  daily      var number
  <chr>   <date>   <dbl>   <chr>   <dbl>
1 Monday 2010-01-11  952.0 orangeBoardings 877
2 Tuesday 2010-01-12  796.0 orangeBoardings 777
3 Wednesday 2010-01-13 1211.5 orangeBoardings 1203
4 Thursday 2010-01-14 1213.5 orangeBoardings 1194
```

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

- Could be explicit on what we want to gather

```
long = gather(circ, key = "var", value = "number",  
              starts_with("orange"), starts_with("purple"),  
              starts_with("green"), starts_with("banner"))  
head(long, 4)
```

```
# A tibble: 4 x 5
```

	day <chr>	date <date>	daily <dbl>	var <chr>	number <dbl>
1	Monday	2010-01-11	952.0	orangeBoardings	877
2	Tuesday	2010-01-12	796.0	orangeBoardings	777
3	Wednesday	2010-01-13	1211.5	orangeBoardings	1203
4	Thursday	2010-01-14	1213.5	orangeBoardings	1194

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

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

bannerAlightings	bannerAverage	bannerBoardings	greenAlightings
1146	1146	1146	1146
greenAverage	greenBoardings	orangeAlightings	orangeAverage
1146	1146	1146	1146
orangeBoardings	purpleAlightings	purpleAverage	purpleBoardings
1146	1146	1146	1146

Lab Part 1

[Website](#)

Making a separator

We will use `str_replace` from `stringr` to put periods in the names (periods are **not** special when in a replacement)

```
library(stringr)
long = long %>% mutate(
  var = var %>% str_replace("Board", ".Board") %>%
    str_replace("Alight", ".Alight") %>%
    str_replace("Average", ".Average")
)
table(long$var)
```

banner.Alightings	banner.Average	banner.Boardings	green.Alightings
1146	1146	1146	1146
green.Average	green.Boardings	orange.Alightings	orange.Average
1146	1146	1146	1146
orange.Boardings	purple.Alightings	purple.Average	purple.Boardings
1146	1146	1146	1146

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

Now each `var` is boardings, averages, or alightings. We want to separate these so we can have these by line. Remember "." is special character:

```
long = separate(long, var, into = c("line", "type"),  
                sep = "[.]")  
head(long, 2)
```

```
# A tibble: 2 x 6  
   day      date daily line      type number  
  <chr>   <date> <dbl> <chr>   <chr>   <dbl>  
1 Monday 2010-01-11   952 orange Boardings    877  
2 Tuesday 2010-01-12   796 orange Boardings    777
```

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

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

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

```
[1] "Boardings" "Alightings" "Average"
```


Re-uniting all the lines

If we had the opposite problem, we could use the `unite` function:

```
reunited = long %>%  
  unite(col = var, line, type, sep = ".")  
reunited %>% select(day, var) %>% head(3) %>% print
```

```
# A tibble: 3 x 2  
  day      var  
  <chr>    <chr>  
1 Monday orange.Boardings  
2 Tuesday orange.Boardings  
3 Wednesday orange.Boardings
```

We could also use `paste/paste0`.

Making column names a little more separated

Alternative: We could have replaced the column names first **then** reshaped:

```
cn = colnames(circ)
cn = cn %>%
  str_replace("Board", ".Board") %>%
  str_replace("Alight", ".Alight") %>%
  str_replace("Average", ".Average")
colnames(circ) = cn # then reshape using gather!
```

Lab Part 2

[Website](#)

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

In `tidyr`, 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)
```

```
# A tibble: 6 x 7
```

	day	date	daily	line	Alightings	Average	Boardings
	<chr>	<date>	<dbl>	<chr>	<dbl>	<dbl>	<dbl>
1	Friday	2010-01-15	1644.0	banner	NA	NA	NA
2	Friday	2010-01-15	1644.0	green	NA	NA	NA
3	Friday	2010-01-15	1644.0	orange	1643	1644	1645
4	Friday	2010-01-15	1644.0	purple	NA	NA	NA
5	Friday	2010-01-22	1394.5	banner	NA	NA	NA
6	Friday	2010-01-22	1394.5	green	NA	NA	NA

Lab Part 3

[Website](#)

Merging: Simple Data

```
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

Joining in `dplyr`

- Merging/joining data sets together - usually on key variables, usually "id"
- `?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>
- `inner_join(x, y)` - only rows that match for `x` and `y` are kept
- `full_join(x, y)` - all rows of `x` and `y` are kept
- `left_join(x, y)` - all rows of `x` are kept even if not merged with `y`
- `right_join(x, y)` - all rows of `y` are kept even if not merged with `x`

Inner Join

```
ij = inner_join(base, visits)
```

```
Joining, by = "id"
```

```
dim(ij)
```

```
[1] 24  4
```

```
tail(ij)
```

	id	Age	visit	Outcome
19	7	58.33333	1	20.43478
20	7	58.33333	3	34.34783
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

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

Right Join: Switching arguments

```
rj2 = right_join(visits, base)
```

```
Joining, by = "id"
```

```
dim(rj2)
```

```
[1] 26  4
```

```
tail(rj2)
```

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

```
identical(rj2, lj) ## after some rearranging
```

```
[1] TRUE
```

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

Duplicated

- The `duplicated` command can give you indications if there are duplications in a vector:

```
duplicated(1:5)
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

```
duplicated(c(1:5, 1))
```

```
[1] FALSE FALSE FALSE FALSE FALSE TRUE
```

Lab Part 4

[Website](#)

Data Merging/Append in Base R

- `merge()` is the most common way to do this with data sets
 - we will use the "join" functions from `dplyr`
- `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()` can transpose data but not

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
```

More Data Manipulation

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.

```
head(wide, 3)
```

```
# A tibble: 3 x 7
```

	day <chr>	date <date>	daily <dbl>	line <chr>	Alightings <dbl>	Average <dbl>	Boardings <dbl>
1	Friday	2010-01-15	1644	banner	NA	NA	NA
2	Friday	2010-01-15	1644	green	NA	NA	NA
3	Friday	2010-01-15	1644	orange	1643	1644	1645

```
not_namat = !is.na(select(wide, Alightings, Average, Boardings))  
head(not_namat, 2)
```

	Alightings	Average	Boardings
1	FALSE	FALSE	FALSE
2	FALSE	FALSE	FALSE

```
wide$good = rowSums(not_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)
```

```
# A tibble: 6 x 7
```

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

Finding the First (or Last) record

- Slice allows you to select **records** (compared to first/last on a **vector**)

```
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() command) lines
first_and_last %>% head(4)
```

```
# A tibble: 4 x 6
# Groups:   line [2]
   day      date    daily  line      type number
  <chr>   <date>   <dbl> <chr>   <chr>   <dbl>
1 Monday 2012-06-04 13342.5 banner Boardings    520
2 Friday 2013-03-01      NA banner Boardings    817
3 Tuesday 2011-11-01  8873.0 green Boardings    887
4 Friday 2013-03-01      NA green Boardings   2592
```

Website

[Website](#)