

Merging Data

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

`gastonsanchez.com`

`github.com/gastonstat/stat133`

Course web: `gastonsanchez.com/teaching/stat133`

Merging Data

Merging

Merging two or more data tables is another frequent type of operation

Motivation

```
A <- data.frame(  
  v1 = seq(2, 10, 2),  
  v2 = 16:20)
```

A

##		v1	v2
##	1	2	16
##	2	4	17
##	3	6	18
##	4	8	19
##	5	10	20

```
B <- data.frame(  
  v1 = seq(0.1, 0.5, 0.1),  
  v2 = seq(0.6, 1, 0.1))
```

B

##		v1	v2
##	1	0.1	0.6
##	2	0.2	0.7
##	3	0.3	0.8
##	4	0.4	0.9
##	5	0.5	1.0

Super basic merging

- ▶ R provides the functions `cbind()` and `rbind()`
- ▶ Column binding requires same number of rows
- ▶ It works on matrices and data frames

Binding columns with cbind()

```
# column binding
```

```
cbind(A, B)
```

```
##   v1 v2  v1  v2  
## 1  2 16 0.1 0.6  
## 2  4 17 0.2 0.7  
## 3  6 18 0.3 0.8  
## 4  8 19 0.4 0.9  
## 5 10 20 0.5 1.0
```

Super basic merging

- ▶ Row binding requires same number of columns
- ▶ It works on matrices and data frames
- ▶ `rbind()` on data frames
 - require columns having same names
 - require column factors having same levels

Binding rows with rbind()

```
# row binding
```

```
rbind(A, B)
```

```
##      v1  v2  
## 1  2.0 16.0  
## 2  4.0 17.0  
## 3  6.0 18.0  
## 4  8.0 19.0  
## 5 10.0 20.0  
## 6  0.1  0.6  
## 7  0.2  0.7  
## 8  0.3  0.8  
## 9  0.4  0.9  
## 10 0.5  1.0
```


Merging

- ▶ Row and column binding provide a very basic type of “merging”
- ▶ For more advanced merging operations, we usually require tables to have an **id** column

Motivation

```
X <- data.frame(  
  id = letters[1:5],  
  x1 = 5:1,  
  x2 = 10:6)
```

X

##		id	x1	x2
##	1	a	5	10
##	2	b	4	9
##	3	c	3	8
##	4	d	2	7
##	5	e	1	6

```
Y <- data.frame(  
  id = letters[1:5],  
  y1 = seq(0.1, 0.5, 0.1),  
  y2 = seq(0.6, 1, 0.1))
```

Y

##		id	y1	y2
##	1	a	0.1	0.6
##	2	b	0.2	0.7
##	3	c	0.3	0.8
##	4	d	0.4	0.9
##	5	e	0.5	1.0

Merging with `merge()`

- ▶ The behavior of `merge()` depends on a combination of several arguments
- ▶ Let's see some of the frequent scenarios

Basic merge()

```
# default merge
```

```
merge(X, Y)
```

```
##   id x1 x2  y1  y2  
## 1  a  5 10 0.1 0.6  
## 2  b  4  9 0.2 0.7  
## 3  c  3  8 0.3 0.8  
## 4  d  2  7 0.4 0.9  
## 5  e  1  6 0.5 1.0
```

merge() will search a common column in both data frames (i.e. column with same name)

Basic merge()

We can explicitly define the name of the column used for merging:

```
merge(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2
## 1  a  5 10 0.1 0.6
## 2  b  4  9 0.2 0.7
## 3  c  3  8 0.3 0.8
## 4  d  2  7 0.4 0.9
## 5  e  1  6 0.5 1.0
```

Let's make it more interesting

```
# shuffling rows in data frame Y
```

```
Y <- Y[c(3, 2, 5, 4, 1), ]
```

```
Y
```

```
##   id  y1  y2
```

```
## 3   c 0.3 0.8
```

```
## 2   b 0.2 0.7
```

```
## 5   e 0.5 1.0
```

```
## 4   d 0.4 0.9
```

```
## 1   a 0.1 0.6
```

Basic merge()

```
merge(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2  
## 1  a  5 10 0.1 0.6  
## 2  b  4  9 0.2 0.7  
## 3  c  3  8 0.3 0.8  
## 4  d  2  7 0.4 0.9  
## 5  e  1  6 0.5 1.0
```

No common column names

```
X <- data.frame(  
  x1 = c(10, 20),  
  x2 = c(30, 40))
```

X

```
##    x1 x2  
## 1 10 30  
## 2 20 40
```

```
Y <- data.frame(  
  y1 = c(0.1, 0.2, 0.3),  
  y2 = c(0.4, 0.5, 0.6))
```

Y

```
##    y1 y2  
## 1 0.1 0.4  
## 2 0.2 0.5  
## 3 0.3 0.6
```


Recycled merge()

If there are no common column names, then recycling is applied:

```
merge(X, Y)
```

```
##    x1 x2  y1  y2  
## 1  10 30 0.1 0.4  
## 2  20 40 0.1 0.4  
## 3  10 30 0.2 0.5  
## 4  20 40 0.2 0.5  
## 5  10 30 0.3 0.6  
## 6  20 40 0.3 0.6
```

Advanced Mergings

Different number of rows

```
X <- data.frame(  
  id = letters[1:4],  
  x1 = 4:1,  
  x2 = 10:7)
```

X

##		id	x1	x2
##	1	a	4	10
##	2	b	3	9
##	3	c	2	8
##	4	d	1	7

```
Y <- data.frame(  
  id = c('b', 'd'),  
  y1 = c(0.1, 0.2),  
  y2 = c(0.6, 0.7))
```

Y

##		id	y1	y2
##	1	b	0.1	0.6
##	2	d	0.2	0.7

Default merge()

Merging rows that share id

```
merge(X, Y, by = 'id')
```

```
##    id x1 x2  y1  y2  
## 1  b  3  9 0.1 0.6  
## 2  d  1  7 0.2 0.7
```

Different number of rows

Values of rows not matching the `id` are filled with NA's

```
merge(X, Y, by = 'id', all = TRUE)
```

```
##   id x1 x2  y1  y2
## 1  a  4 10  NA  NA
## 2  b  3  9 0.1 0.6
## 3  c  2  8  NA  NA
## 4  d  1  7 0.2 0.7
```

Advanced merge()

We can specify which data frame columns to include:

```
merge(X, Y, by = 'id', all.x = TRUE)
```

```
##   id x1 x2  y1  y2
## 1  a  4 10  NA  NA
## 2  b  3  9 0.1 0.6
## 3  c  2  8  NA  NA
## 4  d  1  7 0.2 0.7
```

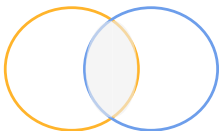
Advanced merge()

We can specify which data frame columns to include:

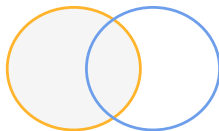
```
merge(X, Y, by = 'id', all.y = TRUE)
```

```
##   id x1 x2  y1  y2  
## 1  b  3  9 0.1 0.6  
## 2  d  1  7 0.2 0.7
```

Merging Types



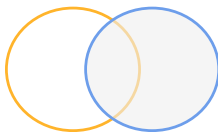
default



all.x = TRUE



all = TRUE



all.y = TRUE

More Mergings

Different id names

```
X <- data.frame(  
  idx = letters[1:4],  
  x1 = 4:1,  
  x2 = 10:7)
```

X

##	idx	x1	x2
## 1	a	4	10
## 2	b	3	9
## 3	c	2	8
## 4	d	1	7

```
Y <- data.frame(  
  idy = c('b', 'd'),  
  y1 = c(0.1, 0.2),  
  y2 = c(0.6, 0.7))
```

Y

##	idy	y1	y2
## 1	b	0.1	0.6
## 2	d	0.2	0.7

More merge() options

Sometimes the id columns have different names:

```
merge(X, Y, by.x = 'idx', by.y = 'idy')
```

```
##   idx x1 x2  y1  y2  
## 1   b  3  9 0.1 0.6  
## 2   d  1  7 0.2 0.7
```

More merge() options

```
merge(X, Y, by.x = 'idx', by.y = 'idy', all = TRUE)
```

```
##   idx x1 x2  y1  y2
## 1   a  4 10  NA  NA
## 2   b  3  9 0.1 0.6
## 3   c  2  8  NA  NA
## 4   d  1  7 0.2 0.7
```

More merge() options

```
merge(X, Y, by.x = 'idx', by.y = 'idy', all.x = TRUE)
```

```
##   idx x1 x2  y1  y2
## 1   a  4 10  NA  NA
## 2   b  3  9 0.1 0.6
## 3   c  2  8  NA  NA
## 4   d  1  7 0.2 0.7
```

Basic merge()

```
merge(X, Y, by.x = 'idx', by.y = 'idy', all.y = TRUE)
```

```
##      idx x1 x2  y1  y2  
## 1     b  3  9 0.1 0.6  
## 2     d  1  7 0.2 0.7
```

Package "dplyr"

Package "dplyr"

```
# install "dplyr"  
install.packages("dplyr")  
  
# load "dplyr"  
library(dplyr)
```


Join Functions

- ▶ `inner_join()`
- ▶ `left_join()`
- ▶ `right_join()`
- ▶ `full_join()`
- ▶ `semi_join()`
- ▶ `anti_join()`

Different number of rows

```
X <- data.frame(  
  id = letters[1:4],  
  x1 = 4:1,  
  x2 = 10:7,  
  stringsAsFactors = FALSE)
```

X

##	id	x1	x2
## 1	a	4	10
## 2	b	3	9
## 3	c	2	8
## 4	d	1	7

```
Y <- data.frame(  
  id = c('b', 'd'),  
  y1 = c(0.1, 0.2),  
  y2 = c(0.6, 0.7),  
  stringsAsFactors = FALSE)
```

Y

##	id	y1	y2
## 1	b	0.1	0.6
## 2	d	0.2	0.7

Function `inner_join()`

`inner_join()` returns all rows from X where there are matching values in Y, and all columns from X and Y.

```
inner_join(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2
## 1  b  3  9 0.1 0.6
## 2  d  1  7 0.2 0.7
```

Function left_join()

left_join() returns all rows from X, and all columns from X and Y.

```
left_join(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2
## 1  a  4 10  NA  NA
## 2  b  3  9 0.1 0.6
## 3  c  2  8  NA  NA
## 4  d  1  7 0.2 0.7
```

Function right_join()

right_join() returns all rows from Y, and all columns from X and Y.

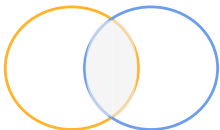
```
right_join(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2
## 1  b  3  9 0.1 0.6
## 2  d  1  7 0.2 0.7
```

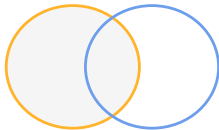
Function full_join()

```
full_join(X, Y, by = 'id')
```

```
##   id x1 x2  y1  y2  
## 1  a  4 10  NA  NA  
## 2  b  3  9 0.1 0.6  
## 3  c  2  8  NA  NA  
## 4  d  1  7 0.2 0.7
```



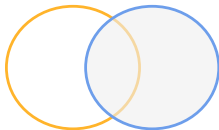
`inner_join`



`left_join`



`full_join`



`right_join`

Function `semi_join()`

`semi_join()` returns all rows from X where there are matching values in Y, keeping just columns from X.

```
semi_join(X, Y, by = 'id')
```

```
##   id x1 x2
## 1  b  3  9
## 2  d  1  7
```


Function anti_join()

`anti_join()` returns all rows from X where there are not matching values in Y, keeping just columns from X.

```
anti_join(X, Y, by = 'id')
```

```
##   id x1 x2
## 1  c  2  8
## 2  a  4 10
```