Merge

Zack Roman[1]

[1]Center for Research Methods and Data Analysis

2018

Outline

Goals of This Session

Conceptual:

- Types of merges
- Merging vocabulary
- When to use merges

Skill Building:

- Practicing merging variants
- Different implementations of merging in R
- Dangers associated with improper merging and how to avoid them

Outline

Small Example

- This is an example provided with R (R Core Team, 2017)

```
authors
```

```
   surname nationality deceased
1    Tukey          US      yes
2 Venables   Australia       no
3  Tierney          US       no
4   Ripley          UK       no
5   McNeil   Australia       no
```

```
books
```

```
      name                         title     other_author
1    Tukey Exploratory Data Analysis                 <NA>
2 Venables Modern Applied Statistics               Ripley
3  Tierney                     LISP-STAT             <NA>
4   Ripley            Spatial Statistics             <NA>
5   Ripley         Stochastic Simulation             <NA>
6   McNeil Interactive Data Analysis                 <NA>
7   R Core          An Introduction to R Venables & Smith
```

Small Example ...

```
merge(x = authors, y = books, by.x = "surname",
    by.y = "name")
```

```
   surname nationality deceased                         title other_author
1   McNeil   Australia       no Interactive Data Analysis           <NA>
2   Ripley          UK       no        Spatial Statistics           <NA>
3   Ripley          UK       no     Stochastic Simulation           <NA>
4  Tierney          US       no                 LISP-STAT           <NA>
5    Tukey          US      yes Exploratory Data Analysis           <NA>
6 Venables   Australia       no Modern Applied Statistics        Ripley
```

Merge Arguments

```
merge(x, y, by.x, by.y, by, incomparables, sort,
    all.x, all.y, all )
```

1. `x` Specifies the left data set
2. `y` Specifies the right data set
3. `by.x, by.y, by` specifies the key as a character string. `by` is common to both `x` and `y`.
4. `incomparables` provides values in the key to not be used for matching, such as NA, blank space, or NaN (not a number).
5. `sort` Logical (TRUE or FALSE), sorts the output
6. `all.x, all.y, all` Logical, will help us determine the behavior of the merge. We will talk more about this as we go

Outline

Binding is not a merging

- The functions `rbind()` and `cbind()` can be used to “stack” matrices on top of each other (rows bound together), or place them side by side (columns bound together)
- Binding puts data sets together, but if the rows (or columns) are not in exactly the same order, it will corrupt the result. Binding two data sets is not merging
- Merging takes into account a ”Key” variable (typically an ID # or Name), so that the correct rows are aligned with each other.

SQL Terminology

- SQL = "Structured Query Language". Very widely used general purpose data-base framework.
- R merge developed in isolation, used different terminology.
- Next we show that the SQL terms "left join", "inner join" and so forth can be achieved by properly setting the value of the merges `all` parameter (`all`, `all.x`, and `all.y`)

Left Join

The "Left Join" is used when the goal data set should **only** have rows that are present in X. The key variable is used to scan Y for matches, which are then merged with the X rows.

```
dat_legs
```

```
  animal legs
1    dog    4
2   cats    4
3  human    2
4  snake    0
5   tree    0
```

```
dat_fur
```

```
  animal    fur
1    dog    yes
2   cats Mostly
3  human     No
4   bird     No
```

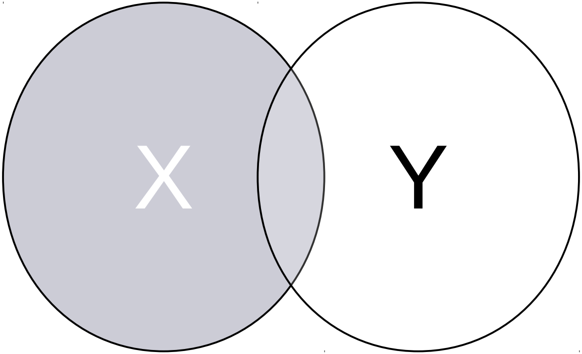
Left Join ...

```
merge(x = dat_legs, y = dat_fur, by = "animal",
    all.x = TRUE)
```

```
  animal legs    fur
1   cats    4 Mostly
2    dog    4    yes
3  human    2     No
4  snake    0   <NA>
5   tree    0   <NA>
```

Setting "all.x" to **TRUE** produces a "Left Join". The output data will contain rows that are in x and there will be additional columns aligned from y.

Left Join



Left Join Switched

Let's do a Left Join again, but switch the data sets.

```
dat_legs
```

```
  animal legs
1    dog    4
2   cats    4
3  human    2
4  snake    0
5   tree    0
```

```
dat_fur
```

```
  animal    fur
1    dog    yes
2   cats Mostly
3  human     No
4   bird     No
```

```
merge(x = dat_fur, y = dat_legs, by = "animal",
    all.x = TRUE)
```

Left Join Switched ...

```
  animal    fur legs
1   bird     No   NA
2   cats Mostly    4
3    dog    yes    4
4  human     No    2
```

Situations calling for Left Join

- You want to investigate the relationship between fur and legs in animals
- You have a data set of the animals you are interested in and their fur status
- You obtain a list of **all** animals legs count
  - Key = Animal Name
  - Output data is the length of the fur data set
- You want to investigate the effect of tuition on retention rate in Florida
- You have Floridian school tuition rates data set
- You obtain a nationwide data set of retention rates
  - Key = School Name
  - Output data is the length of the tuition rates data set

Inner Join

The "Inner join" is used when the goal data set should only have rows that have keys in both the X and Y data.

all = FALSE is the default setting. It is not required to achieve an inner join

```
dat_legs
```

```
  animal legs
1    dog    4
2   cats    4
3  human    2
4  snake    0
5   tree    0
```

```
dat_fur
```

```
  animal    fur
1    dog    yes
2   cats Mostly
3  human     No
4   bird     No
```

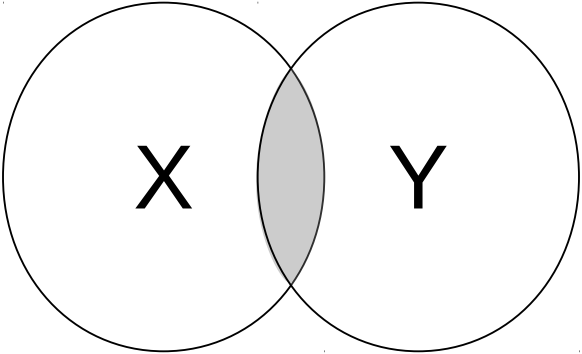
Inner Join ...

```
merge(x = dat_legs, y = dat_fur, by = "animal")
```

```
  animal legs    fur
1   cats    4 Mostly
2    dog    4    yes
3  human    2     No
```

Omitting `all`, or setting `all = FALSE` produces an "Inner Join". The output data will only contain rows that have matching key values on **both** input data sets.

Inner Join



Qualities of Inner Joins

- Pro, result data set will be more complete than other merges.
- Con, result data set loses more information than other merges.

Full Join

Full Join keeps all data rows, filling in unmatched rows with missing values.

```
dat_legs
```

```
  animal legs
1    dog    4
2   cats    4
3  human    2
4  snake    0
5   tree    0
```

```
dat_fur
```

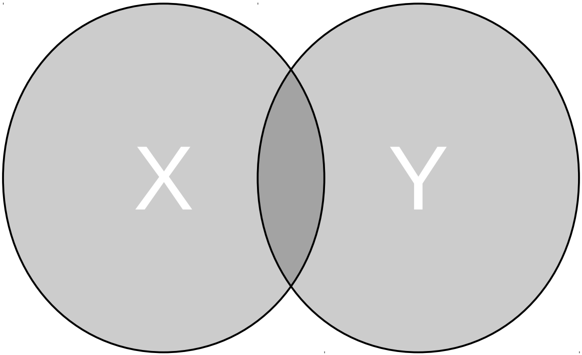
```
  animal    fur
1    dog    yes
2   cats Mostly
3  human     No
4   bird     No
```

Full Join

```r
merge(x = dat_legs, y = dat_fur, by = "animal",
    all = TRUE)
```

```
  animal legs    fur
1   bird   NA     No
2   cats    4 Mostly
3    dog    4    yes
4  human    2     No
5  snake    0   <NA>
6   tree    0   <NA>
```

Full Join



Properties of Full Joins

- Output set includes rows for all cases from both data sets
- There may be lots of "missing" values where rows are not present in one set or the other
- You don't lose any information, but the value of wholly missing rows may be low

Outline

Practice

```
dat1
```

```
  Company Earnings
1       A   126345
2       B   492012
3       C   234512
4       D   -28124
5       E   128675
```

```
dat2
```

```
  Company    Region
1       A   Midwest
2       B Southeast
3       C      West
4       F     North
```

Can you:

- Left Join the data so we have all Earnings in the Output set.
- Left Join the data so we have all Regions in the Output set.
- Inner Join the data so we have no missing data.
- Full Join the data so we have everything in the Output set.

Practice: Answer 1

```
merge(x = dat1, y = dat2, by = "Company", all.x =
    TRUE)
```

```
  Company Earnings    Region
1       A   126345   Midwest
2       B   492012 Southeast
3       C   234512      West
4       D   -28124      <NA>
5       E   128675      <NA>
```

- Left Join the data so we have all Earnings in the Output set.

Practice: Answer 2

```
merge(x = dat2, y = dat1, by = "Company", all.x =
    TRUE)
```

```
  Company    Region Earnings
1       A   Midwest   126345
2       B Southeast   492012
3       C      West   234512
4       F     North       NA
```

- Left Join the data so we have all Regions in the Output set.

Practice: Answer 3

```r
merge(x = dat1, y = dat2, by = "Company", all =
    FALSE)
```

```
  Company Earnings    Region
1       A   126345   Midwest
2       B   492012 Southeast
3       C   234512      West
```

- Inner Join the data so we have no missing data.

Practice: Answer 4

```
merge(x = dat1, y = dat2, by = "Company", all =
    TRUE)
```

```
  Company Earnings    Region
1       A   126345   Midwest
2       B   492012 Southeast
3       C   234512      West
4       D   -28124      <NA>
5       E   128675      <NA>
6       F       NA     North
```

- Full Join the data so we have everything in the Output set.

Outline

Longitudinal Data

- Data comes in 2 typical formats
  1. Wide: Columns that describe units of observation (one row per state, or per school, or per child)

state	region
Alabama	south
Alaska	north
⋮	

  2. Long: Repeated observations, several times for each unit.

year	state	poverty
2000	Alabama	13
2001	Alabama	12
⋮		
2017	Wisconsin	11

- We often want to merge the information about the units from the wide format onto the longitudinal data that is in the long format.

Example: Merging Wide data onto Longitudinal Data

The longitudinal data is about children measured at 3 time points

```
dat_long
```

```
  child_id Time FSIQ
1      110    1   98
2      110    2  102
3      110    3  104
4      210    1   89
5      210    2   91
6      210    3   95
```

Separate data about the education of parents is available for some children

```
dat_edu
```

```
  child_id par_edu
1      210      BA
2      110      HS
```

Longitudinal Data: Long

```
merge(x = dat_long, y = dat_edu, by = "child_id",
    all = TRUE)
```

```
  child_id Time FSIQ par_edu
1      110    1   98      HS
2      110    2  102      HS
3      110    3  104      HS
4      210    1   89      BA
5      210    2   91      BA
6      210    3   95      BA
```

- This is a full join
- No problems encountered, result *seems* adequate.

Points of caution in the full join

1. If information about some families is missing from the wide data, then missing values will be created in the result
   Example:
   We change the wide data by removing one child

```
  child_id par_edu
1      210      BA
```

```
merge(x = dat_long, y = dat_edu2, by =
    "child_id", all = TRUE)
```

```
  child_id Time FSIQ par_edu
1      110    1   98    <NA>
2      110    2  102    <NA>
3      110    3  104    <NA>
4      210    1   89      BA
5      210    2   91      BA
6      210    3   95      BA
```

Points of caution in the full join ...

2. If wide data includes information about children/families that are not tracked in the long data, then the full join will create "extra" all missing lines in the longitudinal part.
   Example:
   We only change `dat_edu` by inserting additional rows for some children.

```
  child_id par_edu
1      210      BA
2      110      HS
3      400      ES
4      501      HS
```

Why would this happen in real life? Suppose these are child/parent data rows from a different study in which some of the children participated.

Points of caution in the full join ...

```
merge(x = dat_long, y = dat_edu2, by =
    "child_id", all = TRUE)
```

```
  child_id Time FSIQ par_edu
1      110    1   98      HS
2      110    2  102      HS
3      110    3  104      HS
4      210    1   89      BA
5      210    2   91      BA
6      210    3   95      BA
7      400   NA   NA      ES
8      501   NA   NA      HS
```

Points of caution in the full join ...

3. Some users may prefer to think of this as a left join, keeping only rows about children in a study (and omitting rows about families of children who are not in the study)

```
merge(x = dat_long, y = dat_edu2, by =
    "child_id", all.x = TRUE, all.y = FALSE)
```

```
  child_id Time FSIQ par_edu
1      110    1   98      HS
2      110    2  102      HS
3      110    3  104      HS
4      210    1   89      BA
5      210    2   91      BA
6      210    3   95      BA
```

Longitudinal Data: Long Data by Long Data

```
dat_long1
```

```
  child_id Time FSIQ
1      110    1   98
2      110    2  102
3      110    3  104
4      210    1   89
5      210    2   91
6      210    3   95
```

```
dat_long2
```

```
  child_id Time Reaction
1      210    1     0.34
2      210    2     0.28
3      210    3     0.19
4      110    1     0.33
5      110    2     0.32
6      110    3     0.28
```

Notice here, the dangers are repeating ID's in both data sets.

Longitudinal Data: Long Data by Long Data

```
head(merge(x = dat_long1, y = dat_long2, by =
    "child_id", all.x = TRUE), 12)
```

```
   child_id Time.x FSIQ Time.y Reaction
1       110      1   98      1     0.33
2       110      1   98      2     0.32
3       110      1   98      3     0.28
4       110      2  102      1     0.33
5       110      2  102      2     0.32
6       110      2  102      3     0.28
7       110      3  104      1     0.33
8       110      3  104      2     0.32
9       110      3  104      3     0.28
10      210      1   89      1     0.34
11      210      1   89      2     0.28
12      210      1   89      3     0.19
```

This is **WRONG!!!** look closely.

Longitudinal Data: Long Data by Long Data

To solve our problem we provide multiple Keys to the "by" argument:

```
merge(x = dat_long1, y = dat_long2, by =
    c("child_id", "Time"), all.x = TRUE)
```

```
  child_id Time FSIQ Reaction
1      110    1   98     0.33
2      110    2  102     0.32
3      110    3  104     0.28
4      210    1   89     0.34
5      210    2   91     0.28
6      210    3   95     0.19
```

That is much better, notice the fix:

```
by = c("child_id", "Time")
```

Longitudinal Data: Long Data by Long Data

An intuitive way to determine when you need to supply multiple keys to the "by" argument is to ask yourself:

- Can every occurrence of my ID variable be uniquely identified ?
- If not, which other variable is necessary to produce an uniquely identified ID ?

Longitudinal Data: QUIZ

Which columns together create the proper uniquely identifiable key set?

```
dat_nat
```

```
   ID Year Quarter population illnesses
1 USA 1990      Q1  10.585529  97.15840
2 USA 1990      Q2  10.709466  90.80678
3 USA 1991      Q1   9.890697  98.83752
4 USA 1991      Q2   9.546503 118.17312
5  UK 1990      Q1  10.605887 103.70628
6  UK 1990      Q2   8.182044 105.20216
7  UK 1991      Q1  10.630099  92.49468
8  UK 1991      Q2   9.723816 108.16900
```

Longitudinal Data: A Useful way to Identify Keys

```
table(dat_nat$ID)
```

```
UK  USA
 4    4
```

Not unique, we need another key

```
table(dat_nat$ID, dat_nat$Quarter)
```

```
      Q1  Q2
 UK    2   2
 USA   2   2
```

getting closer

```
table(dat_nat$ID, dat_nat$Quarter, dat_nat$Year)
```

Longitudinal Data: A Useful way to Identify Keys ...

```
, ,  = 1990


      Q1  Q2
  UK    1   1
  USA   1   1

, ,  = 1991


      Q1  Q2
  UK    1   1
  USA   1   1
```

Winner! Each data point can be uniquely identified as being collected from a country, during a year, and a quarter.

Outline

Different Key Names

```
head(datX)
```

```
   ID Year Quarter       pop illnesses
1 USA 1990      Q1  9.113642  84.02290
2 USA 1990      Q2  9.668422 118.05098
3 USA 1991      Q1 11.120713  95.18353
4 USA 1991      Q2 10.298724 106.20380
5  UK 1990      Q1 10.779622 106.12123
6  UK 1990      Q2 11.455785  98.37689
```

```
head(datY)
```

```
  Country year Semester percipitation      cars
1     USA 1990       Q1     12.049190 111.28511
2     USA 1990       Q2     11.632446  76.19642
3     USA 1991       Q1     10.254271  89.39734
4     USA 1991       Q2     10.491188 109.37141
5      UK 1990       Q1      9.675913 108.54452
6      UK 1990       Q2      8.337950 114.60729
```

Different Key Names

```
head(datX)
```

```
   ID Year Quarter       pop illnesses
1 USA 1990      Q1 10.583188 106.91171
2 USA 1990      Q2  8.693201 108.23795
3 USA 1991      Q1  9.459614 121.45065
4 USA 1991      Q2 11.947693  76.53056
5  UK 1990      Q1 10.053590 101.49592
6  UK 1990      Q2 10.351663  86.57469
```

```
head(datY)
```

```
  Country year Semester percipitation      cars
1     USA 1990       Q1      9.413120  89.50647
2     USA 1990       Q2      8.167623 123.30512
3     USA 1991       Q1     10.888139 114.02705
4     USA 1991       Q2     11.593488 109.42601
5      UK 1990       Q1     10.516855 108.26258
6      UK 1990       Q2      8.704328  91.88460
```

Different Key Names

```
merge(x = datX, y = datY, by.x = c("ID", "Year",
    "Quarter"), by.y = c("Country", "year",
    "Semester"), all = TRUE)
```

```
   ID Year Quarter       pop illnesses percipitation      cars
1  UK 1990      Q1 10.053590 101.49592     10.516855 108.26258
2  UK 1990      Q2 10.351663  86.57469      8.704328  91.88460
3  UK 1991      Q1  9.329023 105.53303     10.054616 104.76248
4  UK 1991      Q2 10.277954 115.89963      9.215351 110.21258
5 USA 1990      Q1 10.583188 106.91171      9.413120  89.50647
6 USA 1990      Q2  8.693201 108.23795      8.167623 123.30512
7 USA 1991      Q1  9.459614 121.45065     10.888139 114.02705
8 USA 1991      Q2 11.947693  76.53056     11.593488 109.42601
```

Matching Missing

```
datX
```

```
   ID cars      fear
1 111    6  90.61873
2 112    5  97.35806
3  NA    7  91.15475
4 114    6  94.99807
5 115    5 106.76902
6 116    5 114.09072
7  NA    9 109.50524
```

```
datY
```

```
   ID pets
1 111    5
2  NA    4
3 113    4
4 114    8
5 115    6
6  NA    4
7 117    7
```

Matching Missing:The Problem

```
merge(x = datX, y = datY, by = "ID", all.x = TRUE)
```

```
   ID cars       fear pets
1 111    6   90.61873    5
2 112    5   97.35806   NA
3 114    6   94.99807    8
4 115    5  106.76902    6
5 116    5  114.09072   NA
6  NA    7   91.15475    4
7  NA    7   91.15475    4
8  NA    9  109.50524    4
9  NA    9  109.50524    4
```

Oops! That is a dangerous outcome: Keys with NA values will be row-aligned

Matching Missing: The Remedy

`incomparables` to the rescue

```
merge(x = datX, y = datY, by = "ID", all= FALSE,
    incomparables = "NA")
```

```
   ID cars      fear pets
1 111    6  90.61873    5
2 114    6  94.99807    8
3 115    5 106.76902    6
```

That is much better! Always remember to use the `incomparables` argument if you have any missing data on keys.

Outline

kutils::mergeCheck

```
df1
```

```
  id          x
1  1 -0.9806329
2  2  0.6873321
3  3 -0.5050435
4  4  2.1577198
5  5 -0.5997976
6  6 -0.6945467
7  7  0.2239254
```

```
df2
```

```
  id          x
1  2 -1.1562233
2  3  0.4224185
3  4 -1.3247553
4  5  0.1410843
5  6 -0.5360480
6  9 -0.3116061
7 10  1.5561096
```

Kutils::mergeCheck

```
library(kutils)
mergeCheck(df1, df2, by = "id")
```

```
Merge difficulties detected

Unmatched cases from df1 and df2 :
df1
  id          x
1  1 -0.9806329
7  7  0.2239254
df2
  id          x
6  9 -0.3116061
7 10  1.5561096
```

- mergeCheck alerts you to potential merging issues
- ID 1 and 7 in the df1 dont have matching IDs in df2
- ID 9 and 10, in the df1 dont have matching id in df2

kutils::mergeCheck

```
df1
```

```
  idx           x
1   1 -0.44803329
2   2  0.32112354
3   3 -1.23017225
4   4 -1.32405869
5   5  1.26124227
6  NA  1.31923172
7 NaN -0.08075376
```

```
df2
```

```
  idy           x
1   2 -0.50508981
2   3 -0.05215359
3   4  0.62886063
4   5  2.18000240
5   6 -0.06901731
6   9  1.54486360
7  10  1.32145202
```

Kutils::mergeCheck

```
mergeCheck(df1, df2, by.x = "idx", by.y = "idy")
```

```
Merge difficulties detected

Unacceptable key values
df1
  idx           x
6  NA  1.31923172
7 NaN -0.08075376
Unmatched cases from df1 and df2 :
df1
  idx           x
1   1 -0.44803329
6  NA  1.31923172
7 NaN -0.08075376
df2
  idy           x
5   6 -0.06901731
6   9  1.54486360
7  10  1.32145202
```

- In this situation we are warned of:
    - Unacceptable key values: NA and NaN

Kutils::mergeCheck ...

- Again, unmatched IDs: 1,6,7,9,10

kutils::mergeCheck

Load `library(kutils)` and run `example(mergeCheck)` to learn more about the function. Our kutils package has much more to offer! check out the kutils help page with `help(package = "kutils")`

More Information

- The CRMDA has a guide available on merges:
  - https://crmda.ku.edu/guide-41-merge_R_SQL

References

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 19.04

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] kutils_1.69
```

Session ...

```
loaded via a namespace (and not attached):
 [1] compiler_3.6.0 plyr_1.8.4     tools_3.6.0    foreign_0.8-71
      lavaan_0.6-3   Rcpp_1.0.1
 [7] mnormt_1.5-5   pbivnorm_0.6.0 xtable_1.8-4   zip_2.0.2
      openxlsx_4.1.0 stats4_3.6.0
```