R for Health Data Science

Week 02: Data Manipulation

Sam Stewart

2021-01-14

Objectives

- Load a data set into a data frame
- Clean the columns of the data frame
- Link two datasets
- Transform wide data to long (and vice versa)

Loading and Cleaning Data

- We largely covered this last week
- `read.csv` will be your most commonly used function, but will depend on incoming data format
- Take care in this step - efficient work here can save a lot of time formatting variables/columns
  - Make sure headers/extra rows are handled correctly - first row of data in R should match first observation in dataset
  - Make sure missing values are handled correctly
  - Make sure strings are handled correctly
- Cleaning data is about getting data into the correct format
  - making sure numbers are numbers
  - getting factors formatted correctly
  - formatting dates, creating derived variables

Joining Data

I'll be using a guide available here. I would suggest visiting it, particularly if you're not as familiar with the concept of joins, as the graphics there are helpful.

We'll start with three datasets, `dat01, dat02,` and `dat03`. `dat01` has observations on all subjects 1-10, `dat02` has subjects 1-6, plus two new subjects, and `dat03` has two observations for the first four subjects.

```
#investgating the actions of merge and dplyr with repeated ids
dat01 = data.frame(
  id = 1:10,
  x1 = rep(letters[1:5],2),
  x2 = rep(LETTERS[1:2],5)
)
dat02 = data.frame(
  id = c(1:6,11,12),
  y1 = rep(1:4,2),
  y2 = 'a'
)
```

```
dat03 = data.frame(
  id = rep(1:4,2),
  z1 = -1*(1:8)
)
dat01
```

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

```
dat02
```

```
##   id y1 y2
## 1  1  1  a
## 2  2  2  a
## 3  3  3  a
## 4  4  4  a
## 5  5  1  a
## 6  6  2  a
## 7 11  3  a
## 8 12  4  a
```

```
dat03
```

```
##   id z1
## 1  1 -1
## 2  2 -2
## 3  3 -3
## 4  4 -4
## 5  1 -5
## 6  2 -6
## 7  3 -7
## 8  4 -8
```

Joining with `merge`

```
merge(x, y, by = intersect(names(x), names(y)),
      by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
      sort = TRUE, suffixes = c(".x",".y"), no.dups = TRUE,
      incomparables = NULL, ...)
```

Check out the help file on merge using `help(merge)`. `x` and `y` are the datasets to merge, `by` identifies the variable(s) to merge on, i.e., the variables that identify the subjects.

NOTE Dataframes have rownames as well as column names - you use the rownames much less often, and they're usually just assigned as `1:n` where `n` is the number of rows. I mention it here because, when looking at the output of the `merge()` function you can confuse the rownames (first column) with the `id` (second column).

```r
#inner join
merge(dat01,dat02,by = 'id')
```

```
##   id x1 x2 y1 y2
## 1  1  a  A  1  a
## 2  2  b  B  2  a
## 3  3  c  A  3  a
## 4  4  d  B  4  a
## 5  5  e  A  1  a
## 6  6  a  B  2  a
```

```r
#left join
merge(dat01,dat02,all.x = TRUE)
```

```
##    id x1 x2 y1   y2
## 1   1  a  A  1    a
## 2   2  b  B  2    a
## 3   3  c  A  3    a
## 4   4  d  B  4    a
## 5   5  e  A  1    a
## 6   6  a  B  2    a
## 7   7  b  A NA <NA>
## 8   8  c  B NA <NA>
## 9   9  d  A NA <NA>
## 10 10  e  B NA <NA>
```

```r
#right join
merge(dat01,dat02,all.y=TRUE)
```

```
##   id   x1   x2 y1 y2
## 1  1    a    A  1  a
## 2  2    b    B  2  a
## 3  3    c    A  3  a
## 4  4    d    B  4  a
## 5  5    e    A  1  a
## 6  6    a    B  2  a
## 7 11 <NA> <NA>  3  a
## 8 12 <NA> <NA>  4  a
```

```r
#outer join
merge(dat01,dat02,all.x=TRUE,all.y=TRUE)
```

```
##    id   x1   x2 y1   y2
## 1   1    a    A  1    a
## 2   2    b    B  2    a
## 3   3    c    A  3    a
## 4   4    d    B  4    a
## 5   5    e    A  1    a
## 6   6    a    B  2    a
## 7   7    b    A NA <NA>
## 8   8    c    B NA <NA>
## 9   9    d    A NA <NA>
## 10 10    e    B NA <NA>
## 11 11 <NA> <NA>  3    a
## 12 12 <NA> <NA>  4    a
```

`dat03` will let you play with joins when there are repeated observations on a subject - you can also see this

behaviour in some of the GIFs on the tutorial page.

```
#non-unique data: left unique
#inner
merge(dat01,dat03)
```

```
##   id x1 x2 z1
## 1  1  a  A -1
## 2  1  a  A -5
## 3  2  b  B -2
## 4  2  b  B -6
## 5  3  c  A -3
## 6  3  c  A -7
## 7  4  d  B -4
## 8  4  d  B -8
```

```
#outer
merge(dat01,dat03,all.x=TRUE,all.y=TRUE)
```

```
##    id x1 x2 z1
## 1   1  a  A -1
## 2   1  a  A -5
## 3   2  b  B -2
## 4   2  b  B -6
## 5   3  c  A -3
## 6   3  c  A -7
## 7   4  d  B -4
## 8   4  d  B -8
## 9   5  e  A NA
## 10  6  a  B NA
## 11  7  b  A NA
## 12  8  c  B NA
## 13  9  d  A NA
## 14 10  e  B NA
```

```
#left join
merge(dat01,dat03,all.x = TRUE)
```

```
##    id x1 x2 z1
## 1   1  a  A -1
## 2   1  a  A -5
## 3   2  b  B -2
## 4   2  b  B -6
## 5   3  c  A -3
## 6   3  c  A -7
## 7   4  d  B -4
## 8   4  d  B -8
## 9   5  e  A NA
## 10  6  a  B NA
## 11  7  b  A NA
## 12  8  c  B NA
## 13  9  d  A NA
## 14 10  e  B NA
```

```
#right join
merge(dat01,dat03,all.y=TRUE)
```

```
##   id x1 x2 z1
## 1  1  a  A -1
## 2  1  a  A -5
## 3  2  b  B -2
## 4  2  b  B -6
## 5  3  c  A -3
## 6  3  c  A -7
## 7  4  d  B -4
## 8  4  d  B -8
```

```
#non-unique data: right unique
#inner
merge(dat03,dat01)
```

```
##   id z1 x1 x2
## 1  1 -1  a  A
## 2  1 -5  a  A
## 3  2 -2  b  B
## 4  2 -6  b  B
## 5  3 -3  c  A
## 6  3 -7  c  A
## 7  4 -4  d  B
## 8  4 -8  d  B
```

```
#outer
merge(dat03,dat01,all.x=TRUE,all.y=TRUE)
```

```
##    id z1 x1 x2
## 1   1 -1  a  A
## 2   1 -5  a  A
## 3   2 -2  b  B
## 4   2 -6  b  B
## 5   3 -3  c  A
## 6   3 -7  c  A
## 7   4 -4  d  B
## 8   4 -8  d  B
## 9   5 NA  e  A
## 10  6 NA  a  B
## 11  7 NA  b  A
## 12  8 NA  c  B
## 13  9 NA  d  A
## 14 10 NA  e  B
```

```
#left join
merge(dat03,dat01,all.x = TRUE)
```

```
##   id z1 x1 x2
## 1  1 -1  a  A
## 2  1 -5  a  A
## 3  2 -2  b  B
## 4  2 -6  b  B
## 5  3 -3  c  A
## 6  3 -7  c  A
## 7  4 -4  d  B
## 8  4 -8  d  B
```

```
#right join
merge(dat03,dat01,all.y=TRUE)
```

```
##    id z1 x1 x2
## 1   1 -1  a  A
## 2   1 -5  a  A
## 3   2 -2  b  B
## 4   2 -6  b  B
## 5   3 -3  c  A
## 6   3 -7  c  A
## 7   4 -4  d  B
## 8   4 -8  d  B
## 9   5 NA  e  A
## 10  6 NA  a  B
## 11  7 NA  b  A
## 12  8 NA  c  B
## 13  9 NA  d  A
## 14 10 NA  e  B
```

Other join commands

The tutorial provides a guide to joining using the `dplyr` library. We'll learn about the tidyverse and pipes (`%>%`) later in the course.

You can do all your joining and data manipulation using SQL if you're interested, through the `tidyquery` library, which translates SQL to `dplyr`. There's also a `dbplyr` library to do the reverse, if you want to write data manipulation in R and translate it to SQL.

```
#testing SQL queries
library(tidyquery)
query("select * from
      dat01 left join dat02
      using(id)")
```

```
##    id x1 x2 y1   y2
## 1   1  a  A  1    a
## 2   2  b  B  2    a
## 3   3  c  A  3    a
## 4   4  d  B  4    a
## 5   5  e  A  1    a
## 6   6  a  B  2    a
## 7   7  b  A NA <NA>
## 8   8  c  B NA <NA>
## 9   9  d  A NA <NA>
## 10 10  e  B NA <NA>
```

Long and Wide Data

I'll be loosely following the tutorial here. It's good, it proposes using `gather()` and `spread()`, which have been replaced with `pivot_longer()` and `pivot_wider()`. I'll still use their data though.

Transforming data from wide to long (and vice versa) is a rarely done but always frustrating step in analysis.

Wide data is often how data is captured - one column per value, one row per subject. A good example of this kind of structure is the CIHI DAD, where there are 25 columns for diagnosis, 25 columns for procedure, etc...

When it comes time for analysis (particularly longitudinal analysis) we often want the outcome to be a single column, so we need to transform the data from wide to long. In the example data below each subject had a control reading and then two readings under intervention conditions 1 and 2. The first dataset is the wide format, the second is the long format.

```
#melting and molding datasets in R.

olddata_wide <- read.table(header=TRUE, text='
 subject sex control cond1 cond2
       1   M     7.9  12.3  10.7
       2   F     6.3  10.6  11.1
       3   F     9.5  13.1  13.8
       4   M    11.5  13.4  12.9
')

# Make sure the subject column is a factor
olddata_wide$subject <- factor(olddata_wide$subject)
olddata_long <- read.table(header=TRUE, text='
 subject sex condition measurement
       1   M   control         7.9
       1   M     cond1        12.3
       1   M     cond2        10.7
       2   F   control         6.3
       2   F     cond1        10.6
       2   F     cond2        11.1
       3   F   control         9.5
       3   F     cond1        13.1
       3   F     cond2        13.8
       4   M   control        11.5
       4   M     cond1        13.4
       4   M     cond2        12.9
')
# Make sure the subject column is a factor
olddata_long$subject <- factor(olddata_long$subject)
olddata_wide
```

```
##   subject sex control cond1 cond2
## 1       1   M     7.9  12.3  10.7
## 2       2   F     6.3  10.6  11.1
## 3       3   F     9.5  13.1  13.8
## 4       4   M    11.5  13.4  12.9
```

```
olddata_long
```

```
##    subject sex condition measurement
## 1        1   M   control         7.9
## 2        1   M     cond1        12.3
## 3        1   M     cond2        10.7
## 4        2   F   control         6.3
## 5        2   F     cond1        10.6
## 6        2   F     cond2        11.1
## 7        3   F   control         9.5
## 8        3   F     cond1        13.1
## 9        3   F     cond2        13.8
## 10       4   M   control        11.5
## 11       4   M     cond1        13.4
```

```
## 12       4   M     cond2        12.9
```

W2L with `pivot_longer`

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

`pivot_longer()` is one way to transform data to long, and the one I am trying to force myself to use moving forward. `data` is the dataset, `cols` is the list of columns that are being transformed to long format, and then `names_to` and `values_to` are the names of the new columns

```
library(tidyr)
data_long = pivot_longer(olddata_wide,
                         control:cond2,
                         names_to="condition",
                         values_to="measurement")
#I use the multi-line functions for ease of reading, but that is unnecessary
data_long = pivot_longer(olddata_wide,control:cond2,"condition","measurement")
data_long
```

```
## # A tibble: 12 x 4
##    subject sex   condition value
##    <fct>   <chr> <chr>     <dbl>
##  1 1       M     control     7.9
##  2 1       M     cond1      12.3
##  3 1       M     cond2      10.7
##  4 2       F     control     6.3
##  5 2       F     cond1      10.6
##  6 2       F     cond2      11.1
##  7 3       F     control     9.5
##  8 3       F     cond1      13.1
##  9 3       F     cond2      13.8
## 10 4       M     control    11.5
## 11 4       M     cond1      13.4
## 12 4       M     cond2      12.9
```

L2W with `pivot_wider`

```
pivot_wider(
  data,
  id_cols = NULL,
```

```
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

```
#and from long to wide
data_wide = pivot_wider(olddata_long,
                        names_from="condition",
                        values_from='measurement')
data_wide
```

```
## # A tibble: 4 x 5
##   subject sex   control cond1 cond2
##   <fct>   <chr>   <dbl> <dbl> <dbl>
## 1 1       M         7.9  12.3  10.7
## 2 2       F         6.3  10.6  11.1
## 3 3       F         9.5  13.1  13.8
## 4 4       M        11.5  13.4  12.9
```

W2L with `melt`

```
melt(
  data,
  id.vars,
  measure.vars,
  variable.name = "variable",
  ...,
  na.rm = FALSE,
  value.name = "value",
  factorsAsStrings = TRUE
)
```

Historically the `reshape2` library would be the easiest way to transform data, using `melt()` to take wide data to long. The advantage of `melt()` is that you can drop variables from the dataset by specifying both `id.vars` (the variables that identify a subject) and `measure.vars` (the variables being transformed from wide to long). The `tidyr` functions don't do that because there are other functions within the tidyverse that drop variables, as we'll see later.

```
#historically, reshape2 was more popular than tidyr
#melt makes wide data long
library(reshape2)
#if you specify one of id.vars and measure.vars,
#it will assume everything else falls in the other
melt(olddata_wide,id.vars=c("subject","sex"))
```

```
##    subject sex variable value
## 1        1   M  control   7.9
## 2        2   F  control   6.3
## 3        3   F  control   9.5
```

```
## 4        4   M  control  11.5
## 5        1   M    cond1  12.3
## 6        2   F    cond1  10.6
## 7        3   F    cond1  13.1
## 8        4   M    cond1  13.4
## 9        1   M    cond2  10.7
## 10       2   F    cond2  11.1
## 11       3   F    cond2  13.8
## 12       4   M    cond2  12.9
```

```
melt(olddata_wide,measure.vars = c("control","cond1","cond2"))
```

```
##    subject sex variable value
## 1        1   M  control   7.9
## 2        2   F  control   6.3
## 3        3   F  control   9.5
## 4        4   M  control  11.5
## 5        1   M    cond1  12.3
## 6        2   F    cond1  10.6
## 7        3   F    cond1  13.1
## 8        4   M    cond1  13.4
## 9        1   M    cond2  10.7
## 10       2   F    cond2  11.1
## 11       3   F    cond2  13.8
## 12       4   M    cond2  12.9
```

```
#specifiy both and we can drop variables
#in this one we drop sex from the dataset
melt(olddata_wide,id.vars=c("subject"),measure.vars = c("control","cond1","cond2"))
```

```
##    subject variable value
## 1        1  control   7.9
## 2        2  control   6.3
## 3        3  control   9.5
## 4        4  control  11.5
## 5        1    cond1  12.3
## 6        2    cond1  10.6
## 7        3    cond1  13.1
## 8        4    cond1  13.4
## 9        1    cond2  10.7
## 10       2    cond2  11.1
## 11       3    cond2  13.8
## 12       4    cond2  12.9
```

```
#we can set the variable and value names, as with the pivot_ functions
melt(olddata_wide,id.vars=c("subject","sex"),
     measure.vars = c("control","cond1","cond2"),
     variable.name="condition",
     value.name='measurement')
```

```
##    subject sex condition measurement
## 1        1   M   control         7.9
## 2        2   F   control         6.3
## 3        3   F   control         9.5
## 4        4   M   control        11.5
## 5        1   M     cond1        12.3
```

```
## 6        2   F     cond1        10.6
## 7        3   F     cond1        13.1
## 8        4   M     cond1        13.4
## 9        1   M     cond2        10.7
## 10       2   F     cond2        11.1
## 11       3   F     cond2        13.8
## 12       4   M     cond2        12.9
```

L2W with `dcast`

```
dcast(
  data,
  formula,
  fun.aggregate = NULL,
  ...,
  margins = NULL,
  subset = NULL,
  fill = NULL,
  drop = TRUE,
  value.var = guess_value(data)
)
```

`dcast` takes wide data to long, using a formula. We'll explore formulas a bit more next week, and then more when we get to regression.

```
dcast(olddata_long,subject+sex~condition,value.var="measurement")
```

```
##   subject sex cond1 cond2 control
## 1       1   M  12.3  10.7     7.9
## 2       2   F  10.6  11.1     6.3
## 3       3   F  13.1  13.8     9.5
## 4       4   M  13.4  12.9    11.5
```

Breakout Activity

There are two datasets on the course website, `bpSubjects.csv` and `bpMeasures.csv`. The first records the age, sex and enrollment date for 100 subjects in a study to track the systolic blood pressures of LTC residents over 6 months. The second dataset records the BP measurements taken at 4 time intervals, roughly 1, 2, 3 and 6 months after enrollment.

Download the two datasets and do the following

1. Load the datasets into R
2. Format the variables - get numbers and factors correct

- leave formatting the dates to the end, since we haven't covered that yet

3. Join the two datasets into a single dataset
4. Transform the data into a long dataset

- start with a single long variable recording the BP values
- as a second step, make it a long dataset that records both the BP and the date for each measurement

5. For the dates as `Date` data type. See `as.Date()` for details on formatting date variables