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/Joins

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

```
# A tibble: 2 x 4
     id visit1 visit2 visit3
  <int>  <dbl>  <dbl>  <dbl>
1     1     10      4      3
2     2      5      6     NA
```

- Long - multiple rows per observation

```
# A tibble: 5 x 3
     id visit value
  <dbl> <int> <dbl>
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 purpleBoardin
  <chr>   <chr>              <dbl>            <dbl>         <dbl>        <db
1 Monday  01/11/…              877             1027           952
2 Tuesday 01/12/…              777              815           796
# … with 9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,
#   daily <dbl>
```

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

```
[1] "character"
```

Creating a Date class from a character date

```
library(lubridate) # great for dates!

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:

- `pivot_longer` - make multiple columns into variables, (wide to long)

- `pivot_wider` - make a variable into multiple columns, (long to wide)

- `separate` - string into multiple columns

- `unite` - multiple columns into one string

- All the "join" functions for mergin are in `dplyr`

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

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

We first describe which columns we want to "pivot_longer." We want the column names in a "var" variable in the output dataset and the values in a "number" variable:

```
long = pivot_longer(circ, !c(day, date, daily), # NOT pivoting these
                        names_to = "var", values_to = "number")
head(long, 4)
```

```
# A tibble: 4 x 5
  day     date         daily var                number
  <chr>   <date>       <dbl> <chr>               <dbl>
1 Monday 2010-01-11      952 orangeBoardings       877
2 Monday 2010-01-11      952 orangeAlightings     1027
3 Monday 2010-01-11      952 orangeAverage         952
4 Monday 2010-01-11      952 purpleBoardings        NA
```

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

- Could be explicit on what we want to pivot

```
long = pivot_longer(circ,
                    starts_with(c("orange","purple","green","banner")),
                    names_to = "var", values_to = "number")
long
```

```
# A tibble: 13,752 x 5
   day    date        daily var              number
   <chr>  <date>      <dbl> <chr>             <dbl>
 1 Monday 2010-01-11    952 orangeBoardings     877
 2 Monday 2010-01-11    952 orangeAlightings   1027
 3 Monday 2010-01-11    952 orangeAverage       952
 4 Monday 2010-01-11    952 purpleBoardings      NA
 5 Monday 2010-01-11    952 purpleAlightings     NA
 6 Monday 2010-01-11    952 purpleAverage        NA
 7 Monday 2010-01-11    952 greenBoardings       NA
 8 Monday 2010-01-11    952 greenAlightings      NA
 9 Monday 2010-01-11    952 greenAverage         NA
10 Monday 2010-01-11    952 bannerBoardings      NA
# … with 13,742 more rows
```

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

```
long %>% count(var)
```

```
# A tibble: 12 x 2
   var                   n
   <chr>             <int>
 1 bannerAlightings   1146
 2 bannerAverage      1146
 3 bannerBoardings    1146
 4 greenAlightings    1146
 5 greenAverage       1146
 6 greenBoardings     1146
 7 orangeAlightings   1146
 8 orangeAverage      1146
 9 orangeBoardings    1146
10 purpleAlightings   1146
11 purpleAverage      1146
12 purpleBoardings    1146
```

Lab Part 1

Website

Making a separator

We will use `str_replace` from `stringr` to put _ in the names

```
long = long %>% mutate(
  var = var %>%
    str_replace("Board", "_Board") %>%
    str_replace("Alight", "_Alight") %>%
    str_replace("Average", "_Average")
)
long %>% count(var)
```

```
# A tibble: 12 x 2
   var                    n
   <chr>              <int>
 1 banner_Alightings   1146
 2 banner_Average      1146
 3 banner_Boardings    1146
 4 green_Alightings    1146
 5 green_Average       1146
 6 green_Boardings     1146
 7 orange_Alightings   1146
 8 orange_Average      1146
 9 orange_Boardings    1146
10 purple_Alightings   1146
11 purple_Average      1146
12 purple_Boardings    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 Monday  2010-01-11   952 orange Alightings    1027
```

```
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 Monday  orange_Alightings
3 Monday  orange_Average
```

We could also use `paste/paste0`.

Lab Part 2

Website

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

In `tidyr`, the `pivot_wider` 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 = long %>% filter(!is.na(date))
wide = wide %>% pivot_wider(names_from = "type",
                            values_from = "number")
head(wide)
```

```
# A tibble: 6 x 7
  day     date        daily line   Boardings Alightings Average
  <chr>   <date>       <dbl> <chr>     <dbl>      <dbl>   <dbl>
1 Monday  2010-01-11     952 orange      877       1027     952
2 Monday  2010-01-11     952 purple       NA         NA      NA
3 Monday  2010-01-11     952 green        NA         NA      NA
4 Monday  2010-01-11     952 banner       NA         NA      NA
5 Tuesday 2010-01-12     796 orange      777        815     796
6 Tuesday 2010-01-12     796 purple       NA         NA      NA
```

Lab Part 3

Website

Merging: Simple Data

`base` has baseline data for ids `1` to `10` and Age

```
base <- tibble(id = 1:10, Age = seq(55,60, length=10))
head(base, 2)
```

```
# A tibble: 2 x 2
      id   Age
   <int> <dbl>
1      1    55
2      2  55.6
```

`visits` has ids 1 to 8, then 11 (new id), and 3 visits and outcome

```
visits <- tibble(id = c(rep(1:8, 3), 11), visit= c(rep(1:3, 8), 3),
                 Outcome = seq(10,50, length=25))
tail(visits, 2)
```

```
# A tibble: 2 x 3
     id visit Outcome
  <dbl> <dbl>   <dbl>
1     8     3    48.3
2    11     3      50
```

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`

- `anti_join(x, y)` - all rows from `x` not in `y` keeping just columns from `x`.

Inner Join

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

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

```
dim(ij)
```

```
[1] 24  4
```

```
tail(ij)
```

```
# A tibble: 6 x 4
     id   Age visit Outcome
  <dbl> <dbl> <dbl>   <dbl>
1     7  58.3     1      20
2     7  58.3     3    33.3
3     7  58.3     2    46.7
4     8  58.9     2    21.7
5     8  58.9     1      35
6     8  58.9     3    48.3
```

Left Join

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

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

```
dim(lj)
```

```
[1] 26  4
```

```
tail(lj)
```

```
# A tibble: 6 x 4
     id   Age visit Outcome
  <dbl> <dbl> <dbl>   <dbl>
1     7  58.3     2    46.7
2     8  58.9     2    21.7
3     8  58.9     1    35
4     8  58.9     3    48.3
5     9  59.4    NA    NA
6    10  60      NA    NA
```

Logging the joins

The `tidylog` package can show you log outputs from `dplyr` (newly added). You will need to install to use.

```
library(tidylog)
left_join(base, visits)

Joining, by = "id"

left_join: added 2 columns (visit, Outcome)

            > rows only in x     2

            > rows only in y   ( 1)

            > matched rows      24     (includes duplicates)

            >                   ====

            > rows total        26

# A tibble: 26 x 4
      id   Age visit Outcome
   <dbl> <dbl> <dbl>   <dbl>
 1     1    55     1      10
 2     1    55     3    23.3
 3     1    55     2    36.7
 4     2  55.6     2    11.7
 5     2  55.6     1      25
```

Right Join

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

```
Joining, by = "id"

right_join: added 2 columns (visit, Outcome)

          > rows only in x   ( 2)

          > rows only in y     1

          > matched rows      24

          >                  ====

          > rows total        25
```

```
tail(rj, 3)
```

```
# A tibble: 3 x 4
     id   Age visit Outcome
  <dbl> <dbl> <dbl>   <dbl>
1     8  58.9     1      35
2     8  58.9     3    48.3
3    11  NA       3      50
```

Right Join: Switching arguments

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

```
Joining, by = "id"

right_join: added one column (Age)

          > rows only in x   ( 1)

          > rows only in y     2

          > matched rows      24     (includes duplicates)

          >                        ====

          > rows total        26
```

```
tail(rj2, 3)
```

```
# A tibble: 3 x 4
     id visit Outcome   Age
  <dbl> <dbl>   <dbl> <dbl>
1     8     3    48.3  58.9
2     9    NA      NA  59.4
3    10    NA      NA  60
```

```
select: no changes
```

```
select: columns reordered (id, visit, Outcome, Age)
```

Full Join

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

```
Joining, by = "id"

full_join: added 2 columns (visit, Outcome)

          > rows only in x     2

          > rows only in y     1

          > matched rows      24     (includes duplicates)

          >                  ====

          > rows total        27
```

```
tail(fj, 4)
```

```
# A tibble: 4 x 4
     id   Age visit Outcome
  <dbl> <dbl> <dbl>   <dbl>
1     8  58.9     3    48.3
2     9  59.4    NA      NA
3    10  60      NA      NA
4    11  NA       3      50
```

Using the `by` argument

By default - uses intersection of column names. If `by` specified, then uses that, but if other columns with same name, adds `suffix`.

```
base = base %>% mutate(x = 5)
visits = visits %>% mutate(x = 4)
head(full_join(base, visits))
```

```
Joining, by = c("id", "x")

# A tibble: 6 x 5
     id    Age     x visit Outcome
  <dbl> <dbl> <dbl> <dbl>   <dbl>
1     1   55       5    NA      NA
2     2   55.6     5    NA      NA
3     3   56.1     5    NA      NA
4     4   56.7     5    NA      NA
5     5   57.2     5    NA      NA
6     6   57.8     5    NA      NA
```

```
head(full_join(base, visits, by = "id"))
```

```
# A tibble: 6 x 6
     id    Age   x.x visit Outcome   x.y
  <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
1     1   55       5     1      10      4
2     1   55       5     3    23.3      4
3     1   55       5     2    36.7      4
4     2   55.6     5     2    11.7      4
```

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
```

```
fj %>% mutate(dup_id = duplicated(id))
```

```
# A tibble: 27 x 5
      id   Age visit Outcome dup_id
   <dbl> <dbl> <dbl>   <dbl> <lgl>
 1     1  55       1      10 FALSE
 2     1  55       3    23.3 TRUE
 3     1  55       2    36.7 TRUE
 4     2  55.6     2    11.7 FALSE
 5     2  55.6     1      25 TRUE
 6     2  55.6     3    38.3 TRUE
 7     3  56.1     3    13.3 FALSE
 8     3  56.1     2    26.7 TRUE
 9     3  56.1     1      40 TRUE
10     4  56.7     1      15 FALSE
# … with 17 more rows
```

Lab Part 4

Website

Finding the First (or Last) record

`pivot_longer` and `pivot_wider` are new (as of 2019) `tidyr` functions.

See link below:

https://tidyr.tidyverse.org/dev/articles/pivot.html

Website

Website

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    date        daily line   Boardings Alightings Average
  <chr>  <date>        <dbl> <chr>     <dbl>      <dbl>   <dbl>
1 Monday 2010-01-11      952 orange      877       1027     952
2 Monday 2010-01-11      952 purple       NA         NA      NA
3 Monday 2010-01-11      952 green        NA         NA      NA
```

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

```
     Alightings Average Boardings
[1,]       TRUE    TRUE      TRUE
[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 = wide %>% filter(good) %>% select(-good)
head(wide)
```

```
# A tibble: 6 x 7
  day        date          daily line     Boardings Alightings Average
  <chr>      <date>        <dbl> <chr>         <dbl>      <dbl>   <dbl>
1 Monday     2010-01-11  952   orange          877       1027     952
2 Tuesday    2010-01-12  796   orange          777        815     796
3 Wednesday  2010-01-13 1212.  orange         1203       1220    1212.
4 Thursday   2010-01-14 1214.  orange         1194       1233    1214.
5 Friday     2010-01-15 1644   orange         1645       1643    1644
6 Saturday   2010-01-16 1490.  orange         1457       1524    1490.
```

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 == "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. banner Boardings     520
2 Friday  2013-03-01    NA  banner Boardings     817
3 Tuesday 2011-11-01  8873  green  Boardings     887
4 Friday  2013-03-01    NA  green  Boardings    2592
```

Merging in base R (not covered)

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 doesn't make it a `data.frame`

Merging

```
merged.data <- merge(base, visits, by = "id")
head(merged.data, 5)
```

```
  id       Age x.x visit  Outcome x.y
1  1 55.00000   5     1 10.00000   4
2  1 55.00000   5     3 23.33333   4
3  1 55.00000   5     2 36.66667   4
4  2 55.55556   5     2 11.66667   4
5  2 55.55556   5     1 25.00000   4
```

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

```
[1] 24  6
```

Merging

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

```
   id      Age x.x visit  Outcome x.y
22  8 58.88889   5     2 21.66667   4
23  8 58.88889   5     1 35.00000   4
24  8 58.88889   5     3 48.33333   4
25  9 59.44444   5    NA       NA  NA
26 10 60.00000   5    NA       NA  NA
27 11       NA  NA     3 50.00000   4
```

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

```
[1] 27  6
```