Intro to R

Manipulating Data in R

Introduction to R for Public Health Researchers

Reshaping Data

In this module, we will show you how to:

1. Reshape data from wide (fat) to long (tall)
2. Reshape data from long (tall) to wide (fat)
3. Merge Data/Joins
4. Perform operations by a grouping variable

What is wide/long data?

Data is stored *differently* in the tibble.

Wide: has many columns

```
# A tibble: 1 x 4
  State   June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>   <chr>          <chr>         <chr>
1 Alabama 37.2%          36.0%         32.4%
```

Long: column names become data

```
# A tibble: 3 x 3
  State   name            value
  <chr>   <chr>           <chr>
1 Alabama June_vacc_rate  37.2%
2 Alabama May_vacc_rate   36.0%
3 Alabama April_vacc_rate 32.4%
```

What is wide/long data?

Wide: multiple columns per individual, values spread across multiple columns

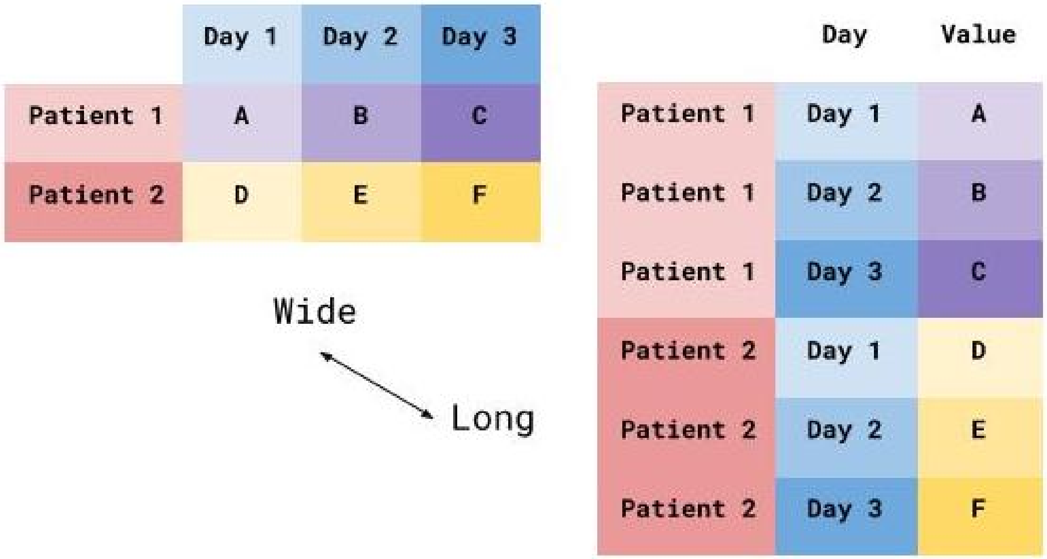
```
# A tibble: 2 x 4
  State   June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>   <chr>          <chr>         <chr>          
1 Alabama 37.2%          36.0%         32.4%          
2 Alaska  47.5%          46.2%         41.7%          
```

Long: multiple rows per observation, a single column contains the values

```
# A tibble: 6 x 3
  State   name            value
  <chr>   <chr>           <chr>
1 Alabama June_vacc_rate  37.2%
2 Alabama May_vacc_rate   36.0%
3 Alabama April_vacc_rate 32.4%
4 Alaska  June_vacc_rate  47.5%
5 Alaska  May_vacc_rate   46.2%
6 Alaska  April_vacc_rate 41.7%
```

What is wide/long data?

Data is wide or long **with respect** to certain variables.



	Day 1	Day 2	Day 3
Patient 1	A	B	C
Patient 2	D	E	F

	Day	Value
Patient 1	Day 1	A
Patient 1	Day 2	B
Patient 1	Day 3	C
Patient 2	Day 1	D
Patient 2	Day 2	E
Patient 2	Day 3	F

CC-BY jhudatascience.org

Why do we need to switch between wide/long data?

Wide: **Easier for humans to read**

```
# A tibble: 2 x 4
  State   June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>   <chr>          <chr>         <chr>          
1 Alabama 37.2%          36.0%         32.4%          
2 Alaska  47.5%          46.2%         41.7%          
```

Long: **Easier for R to make plots & do analysis**

```
# A tibble: 6 x 3
  State   name            value
  <chr>   <chr>           <chr>
1 Alabama June_vacc_rate  37.2%
2 Alabama May_vacc_rate   36.0%
3 Alabama April_vacc_rate 32.4%
4 Alaska  June_vacc_rate  47.5%
5 Alaska  May_vacc_rate   46.2%
6 Alaska  April_vacc_rate 41.7%
```

Data used: Charm City Circulator

http://jhudatascience.org/intro_to_r/data/Charm_City_Circulator_Ridership.csv

```
circ = read_csv(
  paste0("http://jhudatascience.org/intro_to_r/",
         "data/Charm_City_Circulator_Ridership.csv"))
head(circ, 5)
```

```
# A tibble: 5 x 15
  day     date    orangeBoardings orangeAlightings orangeAverage purpleBoardi
  <chr>   <chr>             <dbl>            <dbl>         <dbl>           <db
1 Monday  01/11/…             877             1027          952 
2 Tuesday 01/12/…             777              815          796 
3 Wednes… 01/13/…            1203             1220         1212.
4 Thursd… 01/14/…            1194             1233         1214.
5 Friday  01/15/…            1645             1643         1644 
# … with 9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,
#   daily <dbl>
```

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

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

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

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

- First describe which columns we want to “pivot_longer”
- `names_to` = gives a new name to the pivoted columns
- `values_to` = gives a new name to the values that used to be in those columns

```
long = circ %>%
  pivot_longer(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>  <chr>      <dbl> <chr>              <dbl>
 1 Monday 01/11/2010   952 orangeBoardings      877
 2 Monday 01/11/2010   952 orangeAlightings    1027
 3 Monday 01/11/2010   952 orangeAverage        952
 4 Monday 01/11/2010   952 purpleBoardings       NA
 5 Monday 01/11/2010   952 purpleAlightings      NA
 6 Monday 01/11/2010   952 purpleAverage         NA
 7 Monday 01/11/2010   952 greenBoardings        NA
 8 Monday 01/11/2010   952 greenAlightings       NA
 9 Monday 01/11/2010   952 greenAverage          NA
10 Monday 01/11/2010   952 bannerBoardings       NA
# … with 13,742 more rows
```

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

We have many columns here, so we could instead use the `!` to say which columns we *don't* want to pivot.

```r
long = circ %>% pivot_longer(!c(day, date, daily),
                    names_to = "var", values_to = "number")
long
```

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

Making a separator

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

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

```
# A tibble: 13,752 x 5
   day    date       daily var               number
   <chr>  <chr>      <dbl> <chr>              <dbl>
 1 Monday 01/11/2010   952 orange_Boardings     877
 2 Monday 01/11/2010   952 orange_Alightings   1027
 3 Monday 01/11/2010   952 orange_Average       952
 4 Monday 01/11/2010   952 purple_Boardings      NA
 5 Monday 01/11/2010   952 purple_Alightings     NA
 6 Monday 01/11/2010   952 purple_Average        NA
 7 Monday 01/11/2010   952 green_Boardings       NA
 8 Monday 01/11/2010   952 green_Alightings      NA
 9 Monday 01/11/2010   952 green_Average         NA
10 Monday 01/11/2010   952 banner_Boardings      NA
# … with 13,742 more rows
```

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

Now each `var` is Boardings, Averages, or Alightings. We use “`into =`” to name the new columns and “`sep =`” to show where the separation should happen.

```
long =
  long %>%
  separate(var, into = c("line", "type"), sep = "_")
long
```

```
# A tibble: 13,752 x 6
   day    date       daily line   type        number
   <chr>  <chr>      <dbl> <chr>  <chr>        <dbl>
 1 Monday 01/11/2010   952 orange Boardings      877
 2 Monday 01/11/2010   952 orange Alightings    1027
 3 Monday 01/11/2010   952 orange Average        952
 4 Monday 01/11/2010   952 purple Boardings       NA
 5 Monday 01/11/2010   952 purple Alightings      NA
 6 Monday 01/11/2010   952 purple Average         NA
 7 Monday 01/11/2010   952 green  Boardings       NA
 8 Monday 01/11/2010   952 green  Alightings      NA
 9 Monday 01/11/2010   952 green  Average         NA
10 Monday 01/11/2010   952 banner Boardings       NA
# … with 13,742 more rows
```

Re-uniting all the lines

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

```
reunited = long %>%
  unite(var, line, type, sep = "_")
reunited
```

```
# A tibble: 13,752 x 5
   day    date       daily var                 number
   <chr>  <chr>      <dbl> <chr>                <dbl>
 1 Monday 01/11/2010   952 orange_Boardings       877
 2 Monday 01/11/2010   952 orange_Alightings     1027
 3 Monday 01/11/2010   952 orange_Average         952
 4 Monday 01/11/2010   952 purple_Boardings        NA
 5 Monday 01/11/2010   952 purple_Alightings       NA
 6 Monday 01/11/2010   952 purple_Average          NA
 7 Monday 01/11/2010   952 green_Boardings         NA
 8 Monday 01/11/2010   952 green_Alightings        NA
 9 Monday 01/11/2010   952 green_Average           NA
10 Monday 01/11/2010   952 banner_Boardings        NA
# … with 13,742 more rows
```

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:

```
wide = long %>% pivot_wider(names_from = "type",
                            values_from = "number")
wide
```

```
# A tibble: 4,584 x 7
   day       date       daily line   Boardings Alightings Average
   <chr>     <chr>      <dbl> <chr>      <dbl>      <dbl>   <dbl>
 1 Monday    01/11/2010  952  orange       877       1027    952
 2 Monday    01/11/2010  952  purple        NA         NA     NA
 3 Monday    01/11/2010  952  green         NA         NA     NA
 4 Monday    01/11/2010  952  banner        NA         NA     NA
 5 Tuesday   01/12/2010  796  orange       777        815    796
 6 Tuesday   01/12/2010  796  purple        NA         NA     NA
 7 Tuesday   01/12/2010  796  green         NA         NA     NA
 8 Tuesday   01/12/2010  796  banner        NA         NA     NA
 9 Wednesday 01/13/2010 1212. orange      1203       1220   1212.
10 Wednesday 01/13/2010 1212. purple        NA         NA     NA
# … with 4,574 more rows
```

Lab Part 1

[Website]

Joining in `dplyr`

- Merging/joining data sets together - usually on key variables, usually “id”
- `?join` - see different types of joining for `dplyr`
- `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`.

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 `2` to `11`, 3 different visits, and an outcome

```
visits <- tibble(id = rep(2:11, 3), visit= rep(1:3, 10),
                    Outcome = seq(10,50, length=30))
head(visits, 2)
```

```
# A tibble: 2 x 3
     id visit Outcome
  <int> <int>   <dbl>
1     2     1    10  
2     3     2    11.4
```

Inner Join

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

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

```
dim(ij)
```

```
[1] 27  4
```

```
head(ij)
```

```
# A tibble: 6 x 4
     id   Age visit Outcome
  <int> <dbl> <int>   <dbl>
1     2  55.6     1    10  
2     2  55.6     2    23.8
3     2  55.6     3    37.6
4     3  56.1     2    11.4
5     3  56.1     3    25.2
6     3  56.1     1    39.0
```

Left Join

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

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

```
dim(lj)
```

```
[1] 28  4
```

```
head(lj)
```

```
# A tibble: 6 x 4
     id   Age visit Outcome
  <int> <dbl> <int>   <dbl>
1     1  55      NA    NA
2     2  55.6     1    10
3     2  55.6     2    23.8
4     2  55.6     3    37.6
5     3  56.1     2    11.4
6     3  56.1     3    25.2
```

Install `tidylog` package to log outputs

```
# install.packages("tidylog")
library(tidylog)
left_join(base, visits)
```

```
Joining, by = "id"

left_join: added 2 columns (visit, Outcome)

           > rows only in x    1

           > rows only in y  ( 3)

           > matched rows     27    (includes duplicates)

           >                 ====

           > rows total       28

# A tibble: 28 x 4
      id   Age visit Outcome
   <int> <dbl> <int>   <dbl>
 1     1  55      NA    NA
 2     2  55.6     1    10
 3     2  55.6     2    23.8
 4     2  55.6     3    37.6
 5     3  56.1     2    11.4
 6     3  56.1     3    25.2
 7     3  56.1     1    39.0
```

Right Join

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

```
Joining, by = "id"

right_join: added 2 columns (visit, Outcome)

            > rows only in x  ( 1)

            > rows only in y    3

            > matched rows     27

            >                 ====

            > rows total       30
```

Left Join: Switching arguments

```
lj2 = left_join(visits, base)
```

```
Joining, by = "id"
left_join: added one column (Age)
           > rows only in x    3
           > rows only in y  ( 1)
           > matched rows     27
           >                 ====
           > rows total       30
```

Full Join

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

```
Joining, by = "id"

full_join: added 2 columns (visit, Outcome)

           > rows only in x     1

           > rows only in y     3

           > matched rows      27    (includes duplicates)

           >                 ====

           > rows total        31
```

Full Join

Note what tidylog means by `includes duplicates`. Data from `base` is being duplicated.

```
# fj = full_join(base, visits)
head(fj, 10)
```

```
# A tibble: 10 x 4
      id   Age visit Outcome
   <int> <dbl> <int>   <dbl>
 1     1  55      NA    NA
 2     2  55.6     1    10
 3     2  55.6     2    23.8
 4     2  55.6     3    37.6
 5     3  56.1     2    11.4
 6     3  56.1     3    25.2
 7     3  56.1     1    39.0
 8     4  56.7     3    12.8
 9     4  56.7     1    26.6
10     4  56.7     2    40.3
```

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: 31 x 5
      id   Age visit Outcome dup_id
   <int> <dbl> <int>   <dbl> <lgl>
 1     1  55      NA    NA   FALSE
 2     2  55.6     1    10   FALSE
 3     2  55.6     2    23.8 TRUE
 4     2  55.6     3    37.6 TRUE
 5     3  56.1     2    11.4 FALSE
 6     3  56.1     3    25.2 TRUE
 7     3  56.1     1    39.0 TRUE
 8     4  56.7     3    12.8 FALSE
 9     4  56.7     1    26.6 TRUE
10     4  56.7     2    40.3 TRUE
# … with 21 more rows
```

Using the `by` argument

By default - uses intersection of column names. If `by` specified, then uses that.

```
# for multiple, by = c(col1, col2)
head(full_join(base, visits, by = "id"))
```

```
# A tibble: 6 x 4
     id   Age visit Outcome
  <int> <dbl> <int>   <dbl>
1     1  55      NA     NA 
2     2  55.6     1     10 
3     2  55.6     2     23.8
4     2  55.6     3     37.6
5     3  56.1     2     11.4
6     3  56.1     3     25.2
```

Lab Part 2

Website

- Using rowSums() to get rid of missing data (NAs)
- Finding the first and last record using slice()

Other tidy topics (not covered)

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>  <chr>      <dbl> <chr>      <dbl>      <dbl>   <dbl>
1 Monday 01/11/2010   952 orange       877       1027     952
2 Monday 01/11/2010   952 purple        NA         NA      NA
3 Monday 01/11/2010   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>     <chr>      <dbl> <chr>      <dbl>      <dbl>   <dbl>
1 Monday    01/11/2010  952  orange       877       1027    952 
2 Tuesday   01/12/2010  796  orange       777        815    796 
3 Wednesday 01/13/2010 1212. orange      1203       1220   1212.
4 Thursday  01/14/2010 1214. orange      1194       1233   1214.
5 Friday    01/15/2010 1644  orange      1645       1643   1644 
6 Saturday  01/16/2010 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>   <chr>      <dbl> <chr>  <chr>       <dbl>
1 Tuesday 01/01/2013   NA  banner Boardings    317
2 Monday  12/31/2012   NA  banner Boardings    615
3 Sunday  01/01/2012 3940. green  Boardings    706
4 Monday  12/31/2012   NA  green  Boardings   1925
```

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 visit  Outcome
1  2 55.55556     1 10.00000
2  2 55.55556     2 23.79310
3  2 55.55556     3 37.58621
4  3 56.11111     3 25.17241
5  3 56.11111     2 11.37931
```

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

```
[1] 27  4
```

Merging

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

```
   id Age visit  Outcome
26 10  60     3 21.03448
27 10  60     1 34.82759
28 10  60     2 48.62069
29 11  NA     1 22.41379
30 11  NA     2 36.20690
31 11  NA     3 50.00000
```

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

```
[1] 31  4
```