

Manipulating Data in R

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

Cheatsheet

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf>

Data transformation with dplyr : : CHEAT SHEET

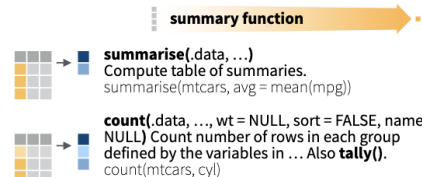


dplyr functions work with pipes and expect **tidy data**. In tidy data:



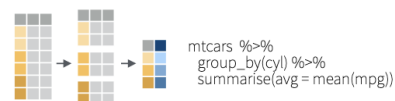
Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



Group Cases

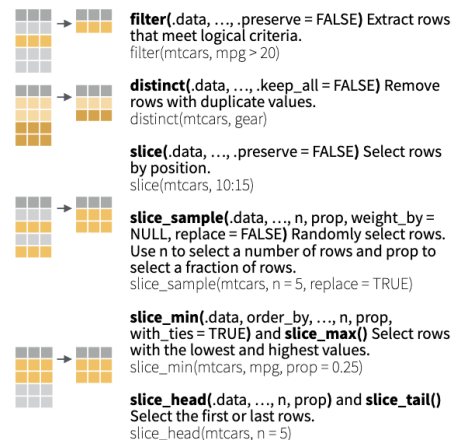
Use **group_by**(data, ..., add = FALSE, drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



Logical and boolean operators to use with filter()

| | | | | | | |
|----|---|----|----------|------|---|-------|
| == | < | <= | is.na() | %in% | | xor() |
| != | > | >= | !is.na() | ! | & | |

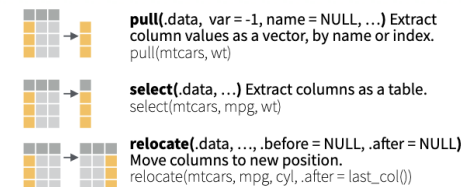
See ?base::Logic and ?Comparison for help.

ARRANGE CASES

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

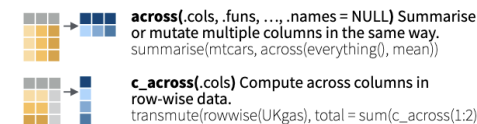


Use these helpers with select() and across()

e.g. select(mtcars, mpg:cyl)

| | | |
|--------------------|--------------------------------|-----------------|
| contains(match) | num_range(prefix, range) | ;, e.g. mpg:cyl |
| ends_with(match) | all_of(x)/any_of(x, ..., vars) | -, e.g. -gear |
| starts_with(match) | matches(match) | everything() |

MANIPULATE MULTIPLE VARIABLES AT ONCE



MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back)

What is wide/long data?

<https://github.com/gadenbuie/tidyexplain/blob/main/images/tidyr-pivoting.gif>

wide

| id | x | y | z |
|----|---|---|---|
| 1 | a | c | e |
| 2 | b | d | f |

What is wide/long data?

Data is stored *differently* in the tibble.

Wide: has many columns

```
# A tibble: 1 × 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 × 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 × 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 × 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.

The diagram illustrates the transformation of data from a wide format to a long format. On the left, a wide table has two rows (Patient 1 and Patient 2) and three columns (Day 1, Day 2, Day 3). On the right, a long table has six rows, each representing a patient-day combination, with two columns (Day and Value). An arrow labeled 'Wide' and 'Long' points from the wide table to the long table.

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

Wide

Long

| | 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 × 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 × 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% |

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 (review)

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

pivot_longer...

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_data} <- {wide_data} %>% pivot_longer(cols = {columns to pivot},  
                                             names_to = {New column name: contains old column names},  
                                             values_to = {New column name: contains cell values})
```

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

wide_data

```
# A tibble: 1 × 3
  June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>          <chr>          <chr>
1 37.2%          36.0%          32.4%
```

```
long_data <- wide_data %>% pivot_longer(cols = everything(),
                                         names_to = "Month",
                                         values_to = "Rate")
```

long_data

```
# A tibble: 3 × 2
  Month      Rate
  <chr>      <chr>
1 June_vacc_rate 37.2%
2 May_vacc_rate  36.0%
3 April_vacc_rate 32.4%
```

Data used: Charm City Circulator

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

```
circ <- jhur::read_circulator()
head(circ, 5)
```

```
# A tibble: 5 × 15
```

| | day | date | orangeBoardings | orangeAlightings | orangeAverage | purpleBoardings |
|---|-----------|---------|-----------------|------------------|---------------|-----------------|
| | <chr> | <chr> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 | Monday | 01/1... | 877 | 1027 | 952 | NA |
| 2 | Tuesday | 01/1... | 777 | 815 | 796 | NA |
| 3 | Wednesday | 01/1... | 1203 | 1220 | 1212. | NA |
| 4 | Thursday | 01/1... | 1194 | 1233 | 1214. | NA |
| 5 | Friday | 01/1... | 1645 | 1643 | 1644 | NA |

```
# ... with 9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,
#   daily <dbl>
```

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

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

long

```
# A tibble: 13,752 × 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

There are many ways to select the columns we want. Use `?tidyr_tidy_select` to look at more column selection options.

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

```
# A tibble: 13,752 × 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 × 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 |

Cleaning up long data

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 × 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
```

Cleaning up long data

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 × 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
```

pivot_wider...

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

`tidyr::pivot_wider` - spreads row data into columns.

- `names_from` = the old column whose contents will be spread into multiple new column names.
- `values_from` = the old column whose contents will fill in the values of those new columns.

```
{wide_data} <- {long_data} %>%  
  pivot_wider(names_from = {Old column name: contains new column names},  
              values_from = {Old column name: contains new cell values})
```

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

long_data

```
# A tibble: 3 × 2
```

| | Month | Rate |
|---|-----------------|-------|
| | <chr> | <chr> |
| 1 | June_vacc_rate | 37.2% |
| 2 | May_vacc_rate | 36.0% |
| 3 | April_vacc_rate | 32.4% |

```
wide_data <- long_data %>% pivot_wider(names_from = "Month",  
                                       values_from = "Rate")
```

wide_data

```
# A tibble: 1 × 3
```

| | June_vacc_rate | May_vacc_rate | April_vacc_rate |
|---|----------------|---------------|-----------------|
| | <chr> | <chr> | <chr> |
| 1 | 37.2% | 36.0% | 32.4% |

Reshaping Charm City Circulator

long

A tibble: 13,752 × 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

Reshaping Charm City Circulator

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

wide

```
# A tibble: 4,584 × 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

data_As

```
# A tibble: 2 × 3
```

| | State | June_vacc_rate | May_vacc_rate |
|---|---------|----------------|---------------|
| | <chr> | <chr> | <chr> |
| 1 | Alabama | 37.2% | 36.0% |
| 2 | Alaska | 47.5% | 46.2% |

data_cold

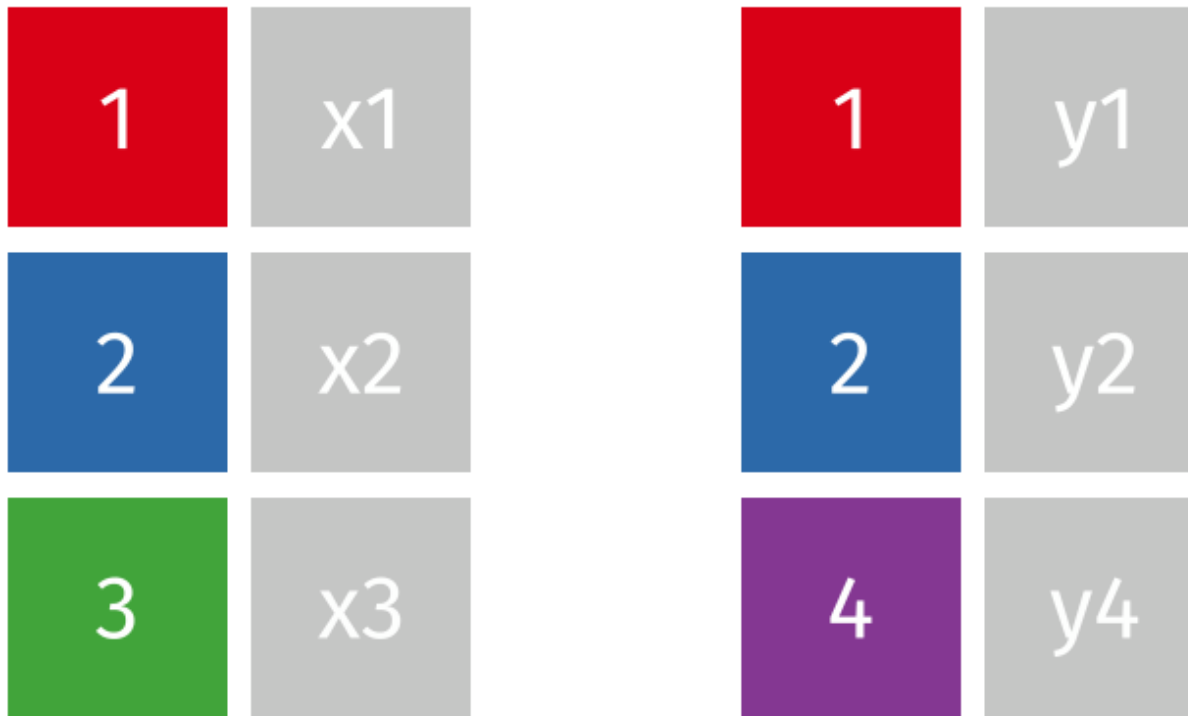
```
# A tibble: 2 × 2
```

| | State | April_vacc_rate |
|---|--------|-----------------|
| | <chr> | <chr> |
| 1 | Maine | 32.4% |
| 2 | Alaska | 41.7% |

Inner Join

<https://github.com/gadenbuie/tidyexplain/blob/main/images/inner-join.gif>

`inner_join(x, y)`



Inner Join

```
ij = inner_join(data_As, data_cold)
```

```
Joining, by = "State"
```

```
ij
```

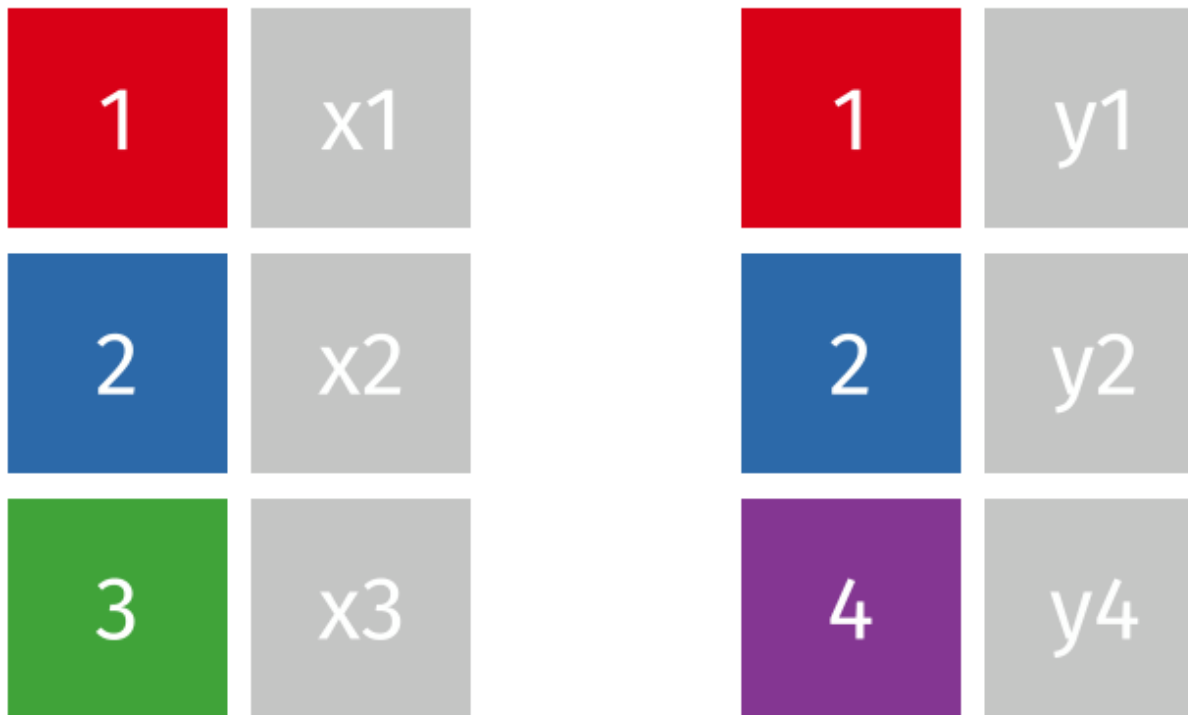
```
# A tibble: 1 × 4
```

| | State | June_vacc_rate | May_vacc_rate | April_vacc_rate |
|---|--------|----------------|---------------|-----------------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alaska | 47.5% | 46.2% | 41.7% |

Left Join

<https://raw.githubusercontent.com/gadenbuie/tidyexplain/main/images/left-join.gif>

`left_join(x, y)`



Left Join

```
lj = left_join(data_As, data_cold)
```

```
Joining, by = "State"
```

```
lj
```

```
# A tibble: 2 × 4
```

| | State | June_vacc_rate | May_vacc_rate | April_vacc_rate |
|---|---------|----------------|---------------|-----------------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alabama | 37.2% | 36.0% | <NA> |
| 2 | Alaska | 47.5% | 46.2% | 41.7% |

Install **tidylog** package to log outputs

```
# install.packages("tidylog")  
library(tidylog)  
left_join(data_As, data_cold)
```

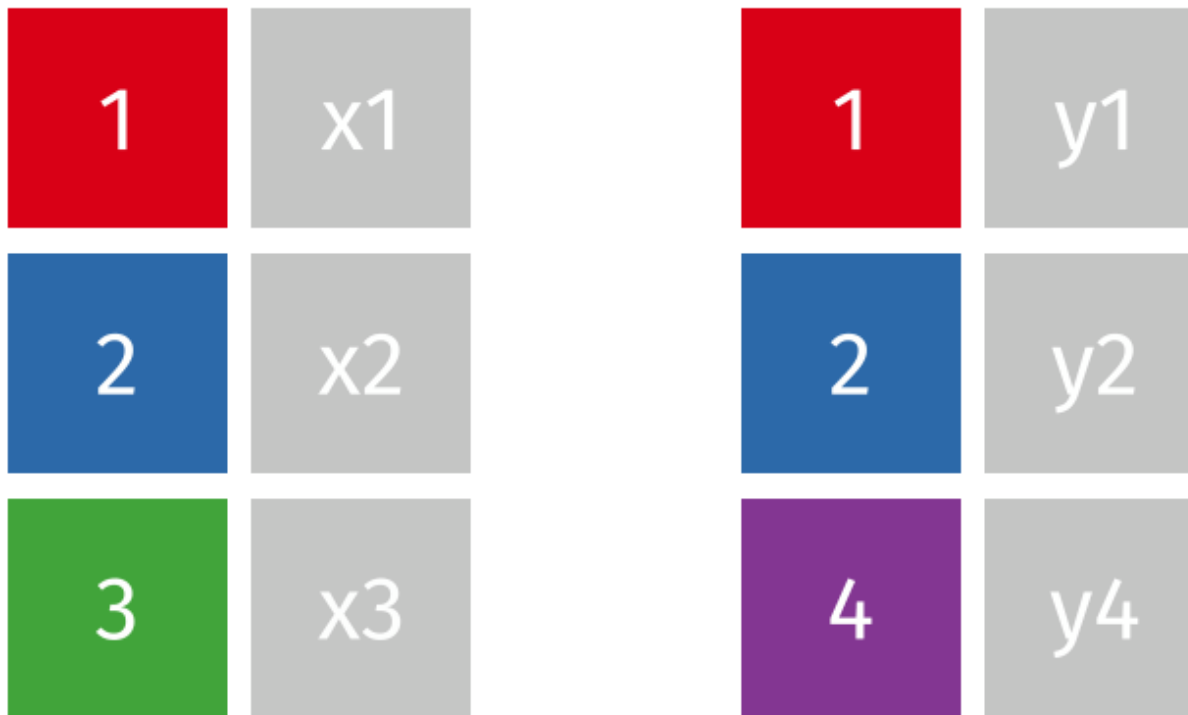
```
Joining, by = "State"  
left_join: added one column (April_vacc_rate)  
> rows only in x 1  
> rows only in y (1)  
> matched rows 1  
> ===  
> rows total 2
```

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

Right Join

<https://raw.githubusercontent.com/gadenbuie/tidyexplain/main/images/right-join.gif>

`right_join(x, y)`



Right Join

```
rj <- right_join(data_As, data_cold)
```

```
Joining, by = "State"
```

```
right_join: added one column (April_vacc_rate)
```

```
> rows only in x (1)
```

```
> rows only in y 1
```

```
> matched rows 1
```

```
> ===
```

```
> rows total 2
```

```
rj
```

```
# A tibble: 2 × 4
```

| | State | June_vacc_rate | May_vacc_rate | April_vacc_rate |
|---|--------|----------------|---------------|-----------------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alaska | 47.5% | 46.2% | 41.7% |
| 2 | Maine | <NA> | <NA> | 32.4% |

Left Join: Switching arguments

```
lj2 <- left_join(data_cold, data_As)
```

```
Joining, by = "State"
```

```
left_join: added 2 columns (June_vacc_rate, May_vacc_rate)
```

```
> rows only in x 1
```

```
> rows only in y (1)
```

```
> matched rows 1
```

```
> ===
```

```
> rows total 2
```

```
lj2
```

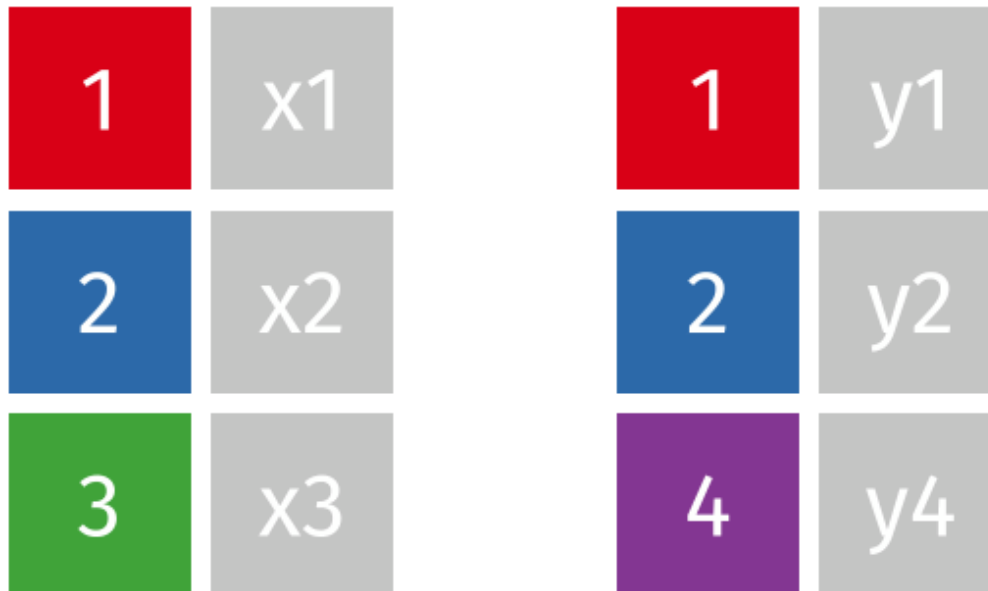
```
# A tibble: 2 × 4
```

| | State | April_vacc_rate | June_vacc_rate | May_vacc_rate |
|---|--------|-----------------|----------------|---------------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Maine | 32.4% | <NA> | <NA> |
| 2 | Alaska | 41.7% | 47.5% | 46.2% |

Full Join

<https://raw.githubusercontent.com/gadenbuie/tidyexplain/main/images/full-join.gif>

`full_join(x, y)`



Full Join

```
fj <- full_join(data_As, data_cold)
```

```
Joining, by = "State"
```

```
full_join: added one column (April_vacc_rate)
```

```
> rows only in x 1
```

```
> rows only in y 1
```

```
> matched rows 1
```

```
> ===
```

```
> rows total 3
```

Full Join

fj

```
# A tibble: 3 × 4
```

| | State | June_vacc_rate | May_vacc_rate | April_vacc_rate |
|---|---------|----------------|---------------|-----------------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alabama | 37.2% | 36.0% | <NA> |
| 2 | Alaska | 47.5% | 46.2% | 41.7% |
| 3 | Maine | <NA> | <NA> | 32.4% |

Watch out for “includes duplicates”

data_As

```
# A tibble: 2 × 2
  State    state_bird
  <chr>    <chr>
1 Alabama wild turkey
2 Alaska  willow ptarmigan
```

data_cold

```
# A tibble: 3 × 3
  State    vacc_rate month
  <chr>    <chr>    <chr>
1 Maine  32.4%    April
2 Alaska 41.7%    April
3 Alaska 46.2%    May
```

Watch out for “includes duplicates”

```
lj <- left_join(data_As, data_cold)
```

```
Joining, by = "State"
```

```
left_join: added 2 columns (vacc_rate, month)
```

```
> rows only in x 1
```

```
> rows only in y (1)
```

```
> matched rows 2 (includes duplicates)
```

```
> ===
```

```
> rows total 3
```

Watch out for “includes duplicates”

Data including the joining column (“State”) has been duplicated.

```
lj
```

```
# A tibble: 3 × 4
```

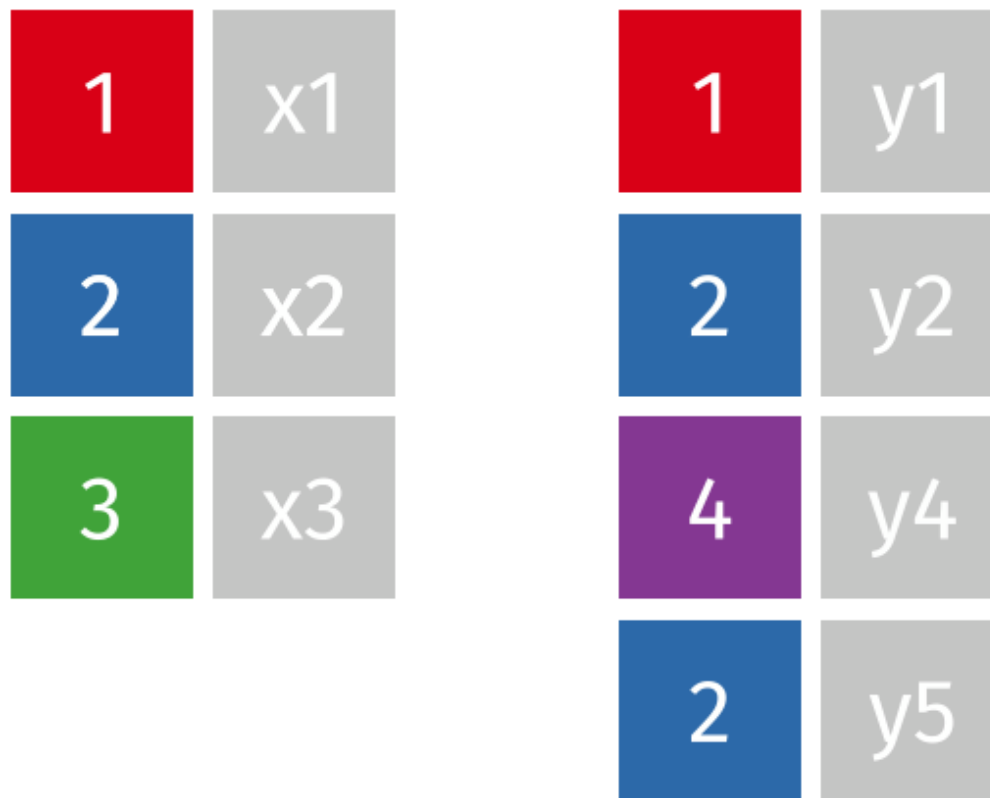
| | State | state_bird | vacc_rate | month |
|---|---------|-------------------|-----------|-------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alabama | wild turkey | <NA> | <NA> |
| 2 | Alaska | williow ptarmigan | 41.7% | April |
| 3 | Alaska | williow ptarmigan | 46.2% | May |

Note that “Alaska willow ptarmigan” appears twice.

Watch out for “includes duplicates”

<https://github.com/gadenbuie/tidyexplain/blob/main/images/left-join-extra.gif>

`left_join(x, y)`



Stop tidylog

```
unloadNamespace("tidylog")
```

Duplicated

- The `uplicated` function can give you indications if there are duplicates in a **vector**:

```
duplicated(1:5)
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

```
duplicated(c(1:5, 1))
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
lj %>% mutate(dup_State = duplicated(State))
```

```
# A tibble: 3 × 5
```

| | State | state_bird | vacc_rate | month | dup_State |
|---|---------|-------------------|-----------|-------|-----------|
| | <chr> | <chr> | <chr> | <chr> | <lgl> |
| 1 | Alabama | wild turkey | <NA> | <NA> | FALSE |
| 2 | Alaska | williow ptarmigan | 41.7% | April | FALSE |
| 3 | Alaska | williow ptarmigan | 46.2% | May | TRUE |

Using the **by** argument

By default joins use the intersection of column names. If **by** is specified, it uses that.

```
full_join(data_As, data_cold, by = "State")
```

```
# A tibble: 4 × 4
```

| | State | state_bird | vacc_rate | month |
|---|---------|-------------------|-----------|-------|
| | <chr> | <chr> | <chr> | <chr> |
| 1 | Alabama | wild turkey | <NA> | <NA> |
| 2 | Alaska | williow ptarmigan | 41.7% | April |
| 3 | Alaska | williow ptarmigan | 46.2% | May |
| 4 | Maine | <NA> | 32.4% | April |

Using the **by** argument

You can join based on multiple columns by using something like `by = c(col1, col2)`.

If the datasets have two different names for the same data, use:

```
full_join(data_As, data_cold, by = c("a" = "b"))
```

Using “setdiff”

We might want to determine what indexes ARE in the first dataset that AREN'T in the second:

```
data_As
```

```
# A tibble: 2 × 2
  State    state_bird
  <chr>    <chr>
1 Alabama wild turkey
2 Alaska willow ptarmigan
```

```
data_cold
```

```
# A tibble: 3 × 3
  State  vacc_rate month
  <chr>  <chr>    <chr>
1 Maine  32.4%    April
2 Alaska 41.7%    April
3 Alaska 46.2%    May
```

Using “setdiff”

Use `setdiff` to determine what indexes ARE in the first dataset that AREN'T in the second:

```
A_states <- data_As %>% pull(State)
cold_states <- data_cold %>% pull(State)
```

```
setdiff(A_states, cold_states)
```

```
[1] "Alabama"
```

```
setdiff(cold_states, A_states)
```

```
[1] "Maine"
```

Lab Part 2

[Website](#)