

Manipulating Data in R

Recap of Data Cleaning

- `is.na()`, `any(is.na())`, `all(is.na())`, `count()`, and functions from `naniar` like `gg_miss_var()` and `miss_var_summary` can help determine if we have NA values
- `miss_var_which()` can help you drop columns that have any missing values.
- `filter()` automatically removes NA values
- `drop_na()` can help you remove NA values
- NA values can change your calculation results
- think about what NA values represent - don't drop them if you shouldn't
- `replace_na()` will replace `NA values with a particular value

Recap of Data Cleaning

- `case_when()` can recode **entire values** based on conditions
 - remember `case_when()` needs `TRUE ~ variable` to keep values that aren't specified by conditions, otherwise will be `NA`
- `stringr` package has great functions for looking for specific **parts of values** especially `filter()` and `str_detect()` combined
 - also has other useful string manipulation functions like `str_replace()` and more!
 - `separate()` can split columns into additional columns
 - `unite()` can combine columns

▮ [Cheatsheet](#)

Manipulating Data

In this module, we will show you how to:

1. Reshape data from wide to long
2. Reshape data from long to wide
3. Merge Data/Joins

What is wide/long data?

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

Wide

Long

	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

What is wide/long data?

Data is stored *differently* in the tibble.

Here's a small dataset looking at vaccination rates over three months in Alabama.

Wide: has many columns

```
# A tibble: 1 × 4
  State    June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>         <dbl>         <dbl>         <dbl>
1 Alabama      0.516         0.514         0.511
```

Long: column names become data

```
# A tibble: 3 × 3
  State    name      value
  <chr>   <chr>      <dbl>
1 Alabama June_vacc_rate 0.516
2 Alabama May_vacc_rate  0.514
3 Alabama April_vacc_rate 0.511
```

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>          <dbl>         <dbl>         <dbl>
1 Alabama      0.516          0.514          0.511
2 Alaska       0.627          0.626          0.623
```

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

```
# A tibble: 6 × 3
  State      name      value
  <chr>    <chr>      <dbl>
1 Alabama June_vacc_rate 0.516
2 Alabama May_vacc_rate  0.514
3 Alabama April_vacc_rate 0.511
4 Alaska  June_vacc_rate 0.627
5 Alaska  May_vacc_rate  0.626
6 Alaska  April_vacc_rate 0.623
```

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

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>          <dbl>         <dbl>         <dbl>
1 Alabama      0.516          0.514          0.511
2 Alaska       0.627          0.626          0.623
```

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

```
# A tibble: 6 × 3
  State      name      value
  <chr>    <chr>      <dbl>
1 Alabama June_vacc_rate 0.516
2 Alabama May_vacc_rate  0.514
3 Alabama April_vacc_rate 0.511
4 Alaska  June_vacc_rate 0.627
5 Alaska  May_vacc_rate  0.626
6 Alaska  April_vacc_rate 0.623
```

Pivoting using the **tidyr** package (part of **tidyverse**)

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)

The `reshape` command exists. Its arguments are considered more confusing, so we don't recommend it.

You might see old functions `gather` and `spread` when googling. These are older iterations of `pivot_longer` and `pivot_wider`, respectively.

pivot_longer...

Reshaping data from wide to long

`pivot_longer()` - puts column data into rows (tidyr package)

- First describe which columns we want to “pivot_longer”

```
{long_data} <- {wide_data} %>% pivot_longer(cols = {columns to pivot})
```

Reshaping data from wide to long

ex_wide

```
# A tibble: 2 × 4
```

	State	June_vacc_rate	May_vacc_rate	April_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Alabama	0.516	0.514	0.511
2	Alaska	0.627	0.626	0.623

```
ex_long <- ex_wide %>% pivot_longer(cols = ends_with("rate"))  
ex_long
```

```
# A tibble: 6 × 3
```

	State	name	value
	<chr>	<chr>	<dbl>
1	Alabama	June_vacc_rate	0.516
2	Alabama	May_vacc_rate	0.514
3	Alabama	April_vacc_rate	0.511
4	Alaska	June_vacc_rate	0.627
5	Alaska	May_vacc_rate	0.626
6	Alaska	April_vacc_rate	0.623

GUT CHECK!

What does `pivot_longer()` do?

- A. Summarize data
- B. Import data
- C. Reshape data

Reshaping wide to long: Better column names

`pivot_longer()` - puts column data into rows (`tidyr` package)

- First describe which columns we want to “pivot_longer”
- `names_to` = new name for old columns
- `values_to` = new name for old cell values

```
{long_data} <- {wide_data} %>% pivot_longer(cols = {columns to pivot},  
                                           names_to = {name for old columns},  
                                           values_to = {name for cell values})
```

Reshaping wide to long: Better column names

Newly created column names (“Month” and “Rate”) are enclosed in quotation marks. It helps us be more specific than “name” and “value”.

```
ex_long <- ex_wide %>% pivot_longer(cols = ends_with("rate"),  
                                   names_to = "Month",  
                                   values_to = "Rate")
```

ex_long

```
# A tibble: 6 × 3  
  State   Month      Rate  
  <chr>   <chr>    <dbl>  
1 Alabama June_vacc_rate 0.516  
2 Alabama May_vacc_rate 0.514  
3 Alabama April_vacc_rate 0.511  
4 Alaska  June_vacc_rate 0.627  
5 Alaska  May_vacc_rate 0.626  
6 Alaska  April_vacc_rate 0.623
```


Data used: Nitrate exposure

Let's look at some data on levels of nitrate in water from Washington. This dataset reports the amount of people in Washington exposed to excess levels of nitrate in their water between 1999 and 2020.

```
wide_nitrate <-  
  read_csv(file = "https://daseh.org/data/Nitrate_Exposure_for_WA_Public_Water_Systems_byquarter_data.csv")  
head(wide_nitrate)  
  
# A tibble: 6 × 11  
  year quarter pop_on_sampled_PWS `pop_0-3ug/L` `pop_>3-5ug/L` `pop_>5-10ug/L`  
  <dbl> <chr>      <dbl>      <dbl>      <dbl>      <dbl>  
1  1999 Q1        106720      67775          0          32  
2  1999 Q2         85541      55476          0         212  
3  1999 Q3        559137     319252     231186         212  
4  1999 Q4         26995      25969         420          0  
5  2000 Q1         34793       5904          0          92  
6  2000 Q2        184521     157396          0          32  
#   5 more variables: `pop_>10-20ug/L` <dbl>, `pop_>20ug/L` <dbl>,  
#   `pop_on_PWS_with_non-detect` <dbl>, pop_exposed_to_exceedances <dbl>,  
#   perc_pop_exposed_to_exceedances <dbl>
```

Mission: Average population exposed by concentration

Let's imagine we want to see what proportion of the population was exposed to different nitrate concentrations. Results should look something like:

```
# A tibble: 3 × 2
  concentration_cat avg_prop_exposedpop
  <chr>              <dbl>
1 0-3ug/L           0.593
2 10-20ug/L         0.000678
3 more_than_20ug/L  0.000129
```

Remove some columns we don't need

```
wide_nitrate <- wide_nitrate %>%  
  select(!ends_with("exceedances"))  
wide_nitrate
```

```
# A tibble: 88 × 9
```

	year	quarter	pop_on_sampled_PWS	`pop_0-3ug/L`	`pop_>3-5ug/L`	`pop_>5-10ug/L`
	<dbl>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	1999	Q1	106720	67775	0	32
2	1999	Q2	85541	55476	0	212
3	1999	Q3	559137	319252	231186	212
4	1999	Q4	26995	25969	420	0
5	2000	Q1	34793	5904	0	92
6	2000	Q2	184521	157396	0	32
7	2000	Q3	42081	20407	345	0
8	2000	Q4	407219	358828	995	412
9	2001	Q1	90054	49552	150	0
10	2001	Q2	83521	43633	2536	90

```
# 78 more rows
```

```
# 3 more variables: `pop_>10-20ug/L` <dbl>, `pop_>20ug/L` <dbl>,  
# `pop_on_PWS_with_non-detect` <dbl>
```

Reshaping data from wide to long

```
long_nitrate <- wide_nitrate %>%  
  pivot_longer(!c(year, quarter, pop_on_sampled_PWS))  
long_nitrate
```

```
# A tibble: 528 × 5
```

	year	quarter	pop_on_sampled_PWS	name	value
	<dbl>	<chr>	<dbl>	<chr>	<dbl>
1	1999	Q1	106720	pop_0-3ug/L	67775
2	1999	Q1	106720	pop_>3-5ug/L	0
3	1999	Q1	106720	pop_>5-10ug/L	32
4	1999	Q1	106720	pop_>10-20ug/L	0
5	1999	Q1	106720	pop_>20ug/L	0
6	1999	Q1	106720	pop_on_PWS_with_non-detect	38913
7	1999	Q2	85541	pop_0-3ug/L	55476
8	1999	Q2	85541	pop_>3-5ug/L	0
9	1999	Q2	85541	pop_>5-10ug/L	212
10	1999	Q2	85541	pop_>10-20ug/L	60

```
#   518 more rows
```

Reshaping data from wide to long

Un-pivoted columns (year, quarter, pop_on_sampled_PWS) are still columns.

```
long_nitrate
```

```
# A tibble: 528 × 5
```

	year	quarter	pop_on_sampled_PWS	name	value
	<dbl>	<chr>	<dbl>	<chr>	<dbl>
1	1999	Q1	106720	pop_0-3ug/L	67775
2	1999	Q1	106720	pop_>3-5ug/L	0
3	1999	Q1	106720	pop_>5-10ug/L	32
4	1999	Q1	106720	pop_>10-20ug/L	0
5	1999	Q1	106720	pop_>20ug/L	0
6	1999	Q1	106720	pop_on_PWS_with_non-detect	38913
7	1999	Q2	85541	pop_0-3ug/L	55476
8	1999	Q2	85541	pop_>3-5ug/L	0
9	1999	Q2	85541	pop_>5-10ug/L	212
10	1999	Q2	85541	pop_>10-20ug/L	60

```
#   518 more rows
```

Cleaning up long data

Let's make the `conc_count` into a proportion.

```
long_nitrate <- long_nitrate %>%  
  mutate(conc_prop = value / pop_on_sampled_PWS)  
long_nitrate
```

```
# A tibble: 528 × 6
```

	year	quarter	pop_on_sampled_PWS	name	value	conc_prop
	<dbl>	<chr>	<dbl>	<chr>	<dbl>	<dbl>
1	1999	Q1	106720	pop_0-3ug/L	67775	0.635
2	1999	Q1	106720	pop_>3-5ug/L	0	0
3	1999	Q1	106720	pop_>5-10ug/L	32	0.000300
4	1999	Q1	106720	pop_>10-20ug/L	0	0
5	1999	Q1	106720	pop_>20ug/L	0	0
6	1999	Q1	106720	pop_on_PWS_with_non-detect	38913	0.365
7	1999	Q2	85541	pop_0-3ug/L	55476	0.649
8	1999	Q2	85541	pop_>3-5ug/L	0	0
9	1999	Q2	85541	pop_>5-10ug/L	212	0.00248
10	1999	Q2	85541	pop_>10-20ug/L	60	0.000701

```
#   518 more rows
```

Mission: Average population exposed by concentration

Now our data is more tidy, and we can take the averages easily!

```
long_nitrate %>%  
  group_by(name) %>%  
  summarize("avg_prop_exposedpop" = mean(conc_prop))
```

```
# A tibble: 6 × 2
```

	name <chr>	avg_prop_exposedpop <dbl>
1	pop_0-3ug/L	0.593
2	pop_>10-20ug/L	0.000678
3	pop_>20ug/L	0.000129
4	pop_>3-5ug/L	0.182
5	pop_>5-10ug/L	0.0189
6	pop_on_PWS_with_non-detect	0.206

Reshaping data from wide to long

There are many ways to **select** the columns we want. Check out https://dplyr.tidyverse.org/reference/dplyr_tidy_select.html to look at more column selection options.

pivot_wider...

Reshaping data from long to wide

`pivot_wider()` - spreads row data into columns (tidyr package)

- `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 to wide

We can use `pivot_wider` to convert long data to wide format. Let's try it with the vaccine data from earlier.

```
ex_long
```

```
# A tibble: 6 × 3
  State      Month      Rate
  <chr>    <chr>    <dbl>
1 Alabama June_vacc_rate 0.516
2 Alabama May_vacc_rate 0.514
3 Alabama April_vacc_rate 0.511
4 Alaska  June_vacc_rate 0.627
5 Alaska  May_vacc_rate 0.626
6 Alaska  April_vacc_rate 0.623
```

Reshaping data from long to wide

We can use `pivot_wider` to convert long data to wide format. Let's try it with the vaccine data from earlier.

```
ex_wide2 <- ex_long %>% pivot_wider(names_from = "Month",  
                                   values_from = "Rate")  
ex_wide2
```

```
# A tibble: 2 × 4
```

	State	June_vacc_rate	May_vacc_rate	April_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Alabama	0.516	0.514	0.511
2	Alaska	0.627	0.626	0.623

Reshaping nitrate exposure data

Let's go back to the nitrate exposure dataset. What if we wanted to make a wide version of the data that displayed the proportion of people at each level of nitrate exposure, with each quarter as a column?

```
long_nitrate
```

```
# A tibble: 528 × 6
```

	year	quarter	pop_on_sampled_PWS	name	value	conc_prop
	<dbl>	<chr>	<dbl>	<chr>	<dbl>	<dbl>
1	1999	Q1	106720	pop_0-3ug/L	67775	0.635
2	1999	Q1	106720	pop_>3-5ug/L	0	0
3	1999	Q1	106720	pop_>5-10ug/L	32	0.000300
4	1999	Q1	106720	pop_>10-20ug/L	0	0
5	1999	Q1	106720	pop_>20ug/L	0	0
6	1999	Q1	106720	pop_on_PWS_with_non-detect	38913	0.365
7	1999	Q2	85541	pop_0-3ug/L	55476	0.649
8	1999	Q2	85541	pop_>3-5ug/L	0	0
9	1999	Q2	85541	pop_>5-10ug/L	212	0.00248
10	1999	Q2	85541	pop_>10-20ug/L	60	0.000701

```
# 518 more rows
```

Reshaping nitrate exposure data

Drop some columns we don't need.

```
long_nitrate <- long_nitrate %>%  
  select(!c(pop_on_sampled_PWS, value))  
long_nitrate
```

```
# A tibble: 528 × 4  
  year quarter name conc_prop  
  <dbl> <chr> <chr> <dbl>  
1 1999 Q1 pop_0-3ug/L 0.635  
2 1999 Q1 pop_>3-5ug/L 0  
3 1999 Q1 pop_>5-10ug/L 0.000300  
4 1999 Q1 pop_>10-20ug/L 0  
5 1999 Q1 pop_>20ug/L 0  
6 1999 Q1 pop_on_PWS_with_non-detect 0.365  
7 1999 Q2 pop_0-3ug/L 0.649  
8 1999 Q2 pop_>3-5ug/L 0  
9 1999 Q2 pop_>5-10ug/L 0.00248  
10 1999 Q2 pop_>10-20ug/L 0.000701  
# 518 more rows
```

Reshaping nitrate exposure data

Pivot the data!

```
wide_nitrate <- long_nitrate %>%  
  pivot_wider(names_from = "quarter", values_from = "conc_prop")  
wide_nitrate
```

```
# A tibble: 132 × 6
```

	year	name	Q1	Q2	Q3	Q4
	<dbl>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	1999	pop_0-3ug/L	0.635	0.649	0.571	0.962
2	1999	pop_>3-5ug/L	0	0	0.413	0.0156
3	1999	pop_>5-10ug/L	0.000300	0.00248	0.000379	0
4	1999	pop_>10-20ug/L	0	0.000701	0	0
5	1999	pop_>20ug/L	0	0	0	0
6	1999	pop_on_PWS_with_non-detect	0.365	0.348	0.0152	0.0224
7	2000	pop_0-3ug/L	0.170	0.853	0.485	0.881
8	2000	pop_>3-5ug/L	0	0	0.00820	0.00244
9	2000	pop_>5-10ug/L	0.00264	0.000173	0	0.00101
10	2000	pop_>10-20ug/L	0	0	0	0

```
# 122 more rows
```

Summary

- `tidyr` package helps us convert between wide and long data
- `pivot_longer()` goes from wide -> long
 - Specify columns you want to pivot
 - Specify `names_to =` and `values_to =` for custom naming
- `pivot_wider()` goes from long -> wide
 - Specify `names_from =` and `values_from =`

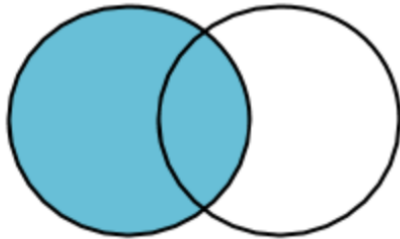
Lab Part 1

▮ [Class Website](#)

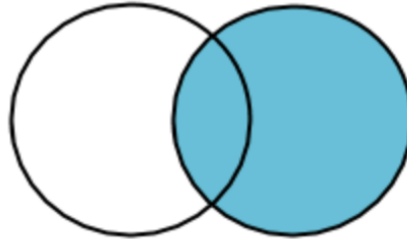
▮ [Lab](#)

Joining

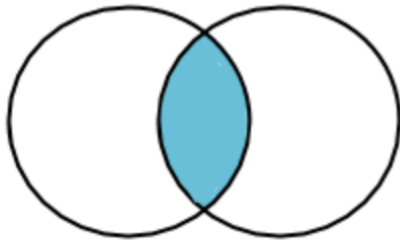
“Combining datasets”



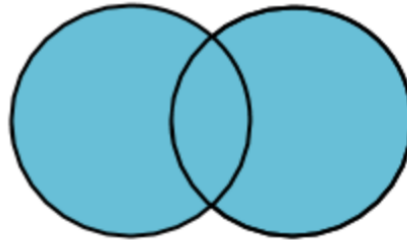
Left Join



Right Join



Inner Join



**Full Outer
Join**

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 <- read_csv(  
  file = "https://daseh.org/data/data_As_1.csv")  
data_cold <- read_csv(  
  file = "https://daseh.org/data/data_cold_1.csv")
```

data_As

```
# A tibble: 2 × 3  
  State      June_vacc_rate May_vacc_rate  
  <chr>          <dbl>         <dbl>  
1 Alabama      0.516          0.514  
2 Alaska       0.627          0.626
```

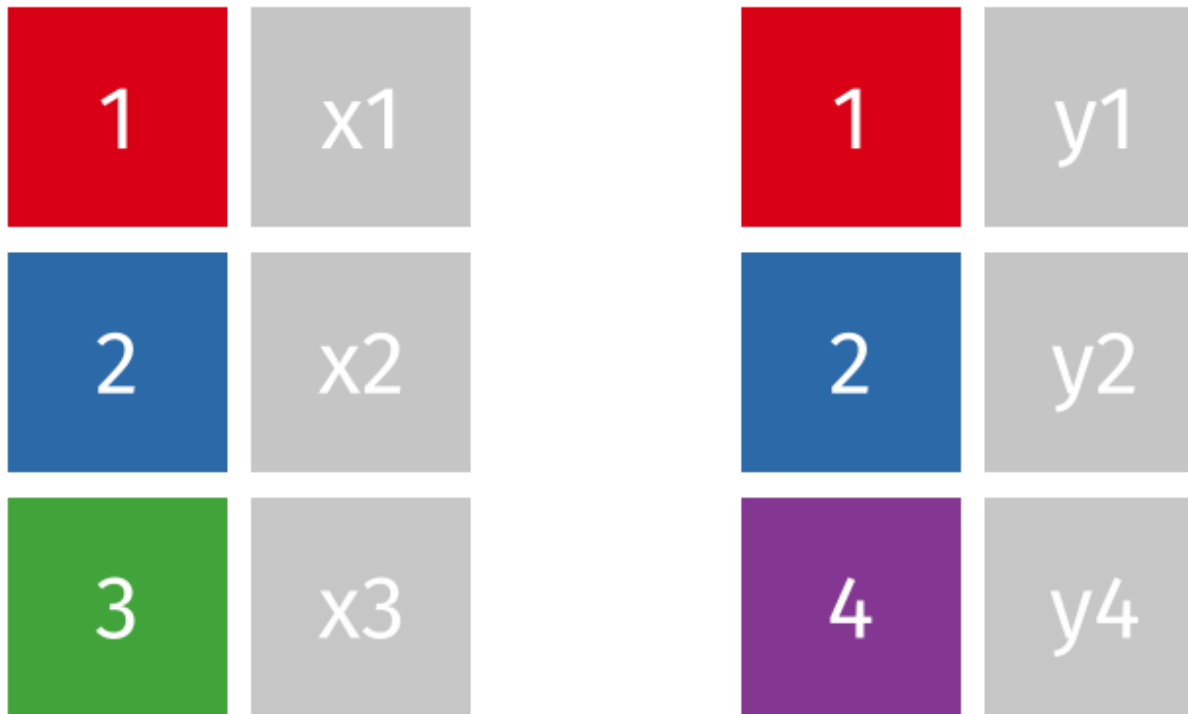
data_cold

```
# A tibble: 2 × 2  
  State      April_vacc_rate  
  <chr>          <dbl>  
1 Maine      0.795  
2 Alaska     0.623
```

Inner Join

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

`inner_join(x, y)`



Inner Join

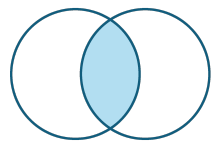
```
ij <- inner_join(data_As, data_cold)
```

Joining with `by = join_by(State)`

```
ij
```

```
# A tibble: 1 × 4
```

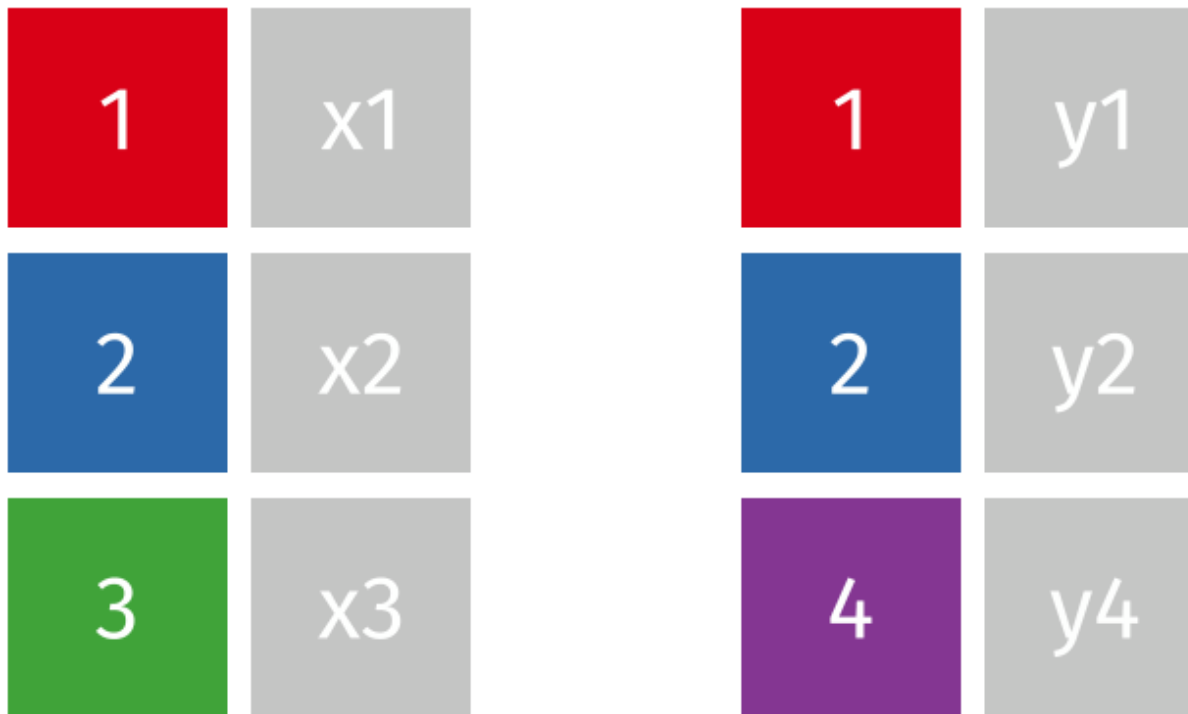
	State	June_vacc_rate	May_vacc_rate	April_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Alaska	0.627	0.626	0.623



Left Join

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

`left_join(x, y)`



Left Join

“Everything to the left of the comma”

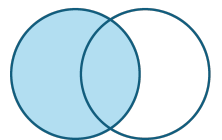
```
lj <- left_join(data_As, data_cold)
```

Joining with `by = join_by(State)`

```
lj
```

```
# A tibble: 2 × 4
```

	State <chr>	June_vacc_rate <dbl>	May_vacc_rate <dbl>	April_vacc_rate <dbl>
1	Alabama	0.516	0.514	NA
2	Alaska	0.627	0.626	0.623

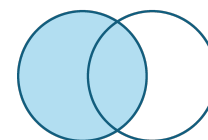


Install **tidylog** package to log outputs

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

```
Joining with `by = join_by(State)`
left_join: added one column (April_vacc_rate)
> rows only in data_As 1
> rows only in data_cold (1)
> matched rows 1
> ===
> rows total 2
```

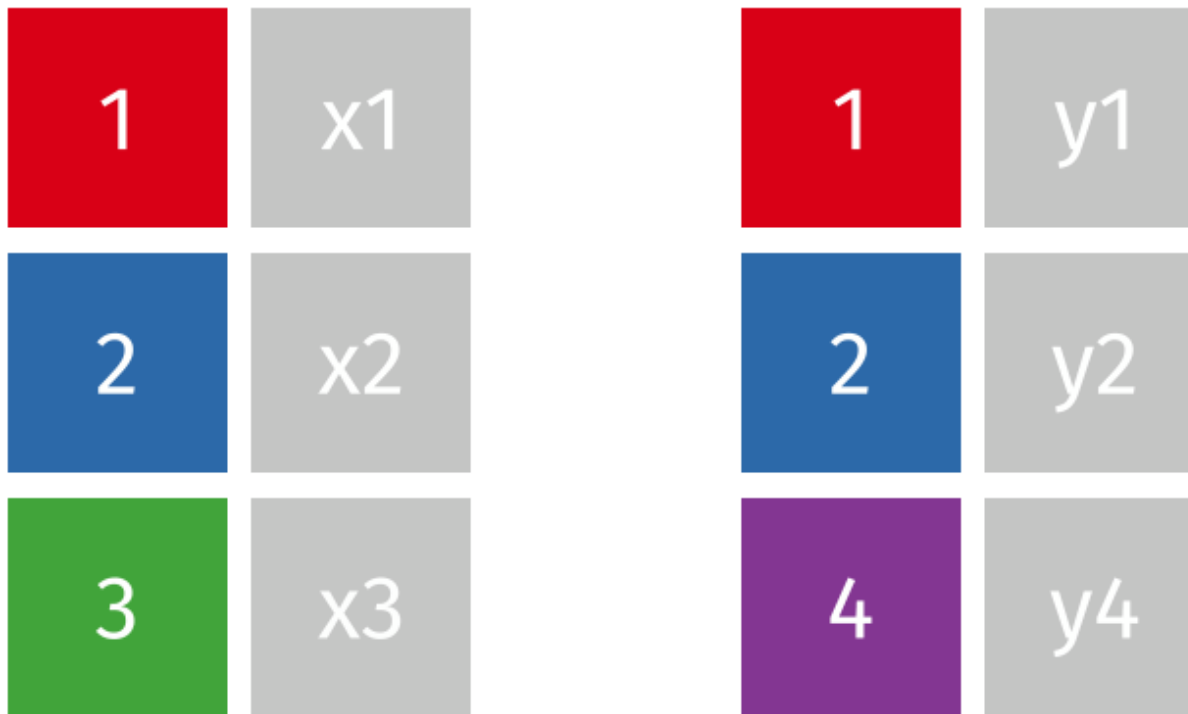
```
# A tibble: 2 × 4
  State      June_vacc_rate May_vacc_rate April_vacc_rate
  <chr>          <dbl>         <dbl>         <dbl>
1 Alabama      0.516         0.514          NA
2 Alaska       0.627         0.626         0.623
```



Right Join

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

`right_join(x, y)`



Right Join

“Everything to the right of the comma”

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

Joining with `by = join_by(State)`

right_join: added one column (April_vacc_rate)

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

```
> rows only in data_cold 1
```

```
> matched rows 1
```

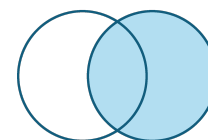
```
> ===
```

```
> rows total 2
```

```
rj
```

```
# A tibble: 2 × 4
```

	State	June_vacc_rate	May_vacc_rate	April_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Alaska	0.627	0.626	0.623
2	Maine	NA	NA	0.795



Left Join: Switching arguments

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

Joining with `by = join_by(State)`

left_join: added 2 columns (June_vacc_rate, May_vacc_rate)

> rows only in data_cold 1

> rows only in data_As (1)

> matched rows 1

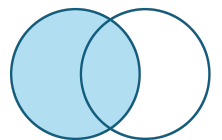
> ===

> rows total 2

```
lj2
```

```
# A tibble: 2 × 4
```

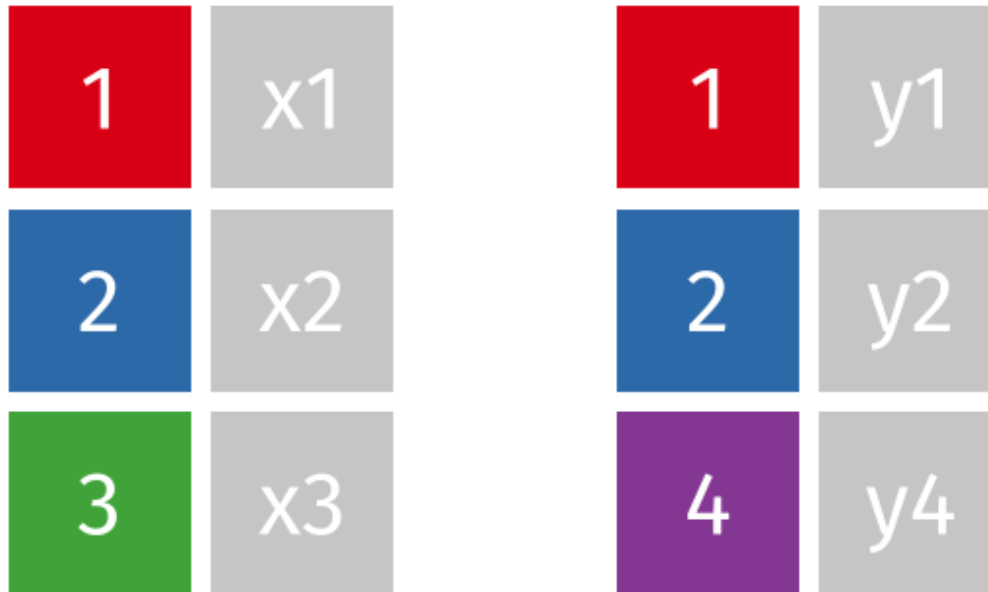
	State	April_vacc_rate	June_vacc_rate	May_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Maine	0.795	NA	NA
2	Alaska	0.623	0.627	0.626



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)
```

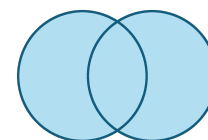
```
Joining with `by = join_by(State)`  
full_join: added one column (April_vacc_rate)
```

```
> rows only in data_As 1  
> rows only in data_cold 1  
> matched rows 1  
> ===  
> rows total 3
```

```
fj
```

```
# A tibble: 3 × 4
```

	State	June_vacc_rate	May_vacc_rate	April_vacc_rate
	<chr>	<dbl>	<dbl>	<dbl>
1	Alabama	0.516	0.514	NA
2	Alaska	0.627	0.626	0.623
3	Maine	NA	NA	0.795



"includes duplicates"

```
data_As <- read_csv(  
  file = "https://daseh.org/data/data_As_2.csv")  
data_cold <- read_csv(  
  file = "https://daseh.org/data/data_cold_2.csv")
```

data_As

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

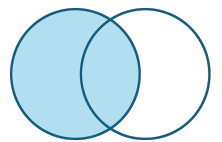
data_cold

```
# A tibble: 3 × 3  
  State    vacc_rate month  
  <chr>    <dbl> <chr>  
1 Maine      0.795 April  
2 Alaska     0.623 April  
3 Alaska     0.626 May
```

“includes duplicates”

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

```
Joining with `by = join_by(State)`  
left_join: added 2 columns (vacc_rate, month)  
> rows only in data_As 1  
> rows only in data_cold (1)  
> matched rows 2 (includes duplicates)  
> ===  
> rows total 3
```



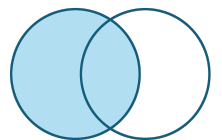
“includes duplicates”

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

lj

```
# A tibble: 3 × 4
  State    state_bird    vacc_rate month
  <chr>    <chr>          <dbl> <chr>
1 Alabama wild turkey      NA    <NA>
2 Alaska  willow ptarmigan  0.623 April
3 Alaska  willow ptarmigan  0.626 May
```

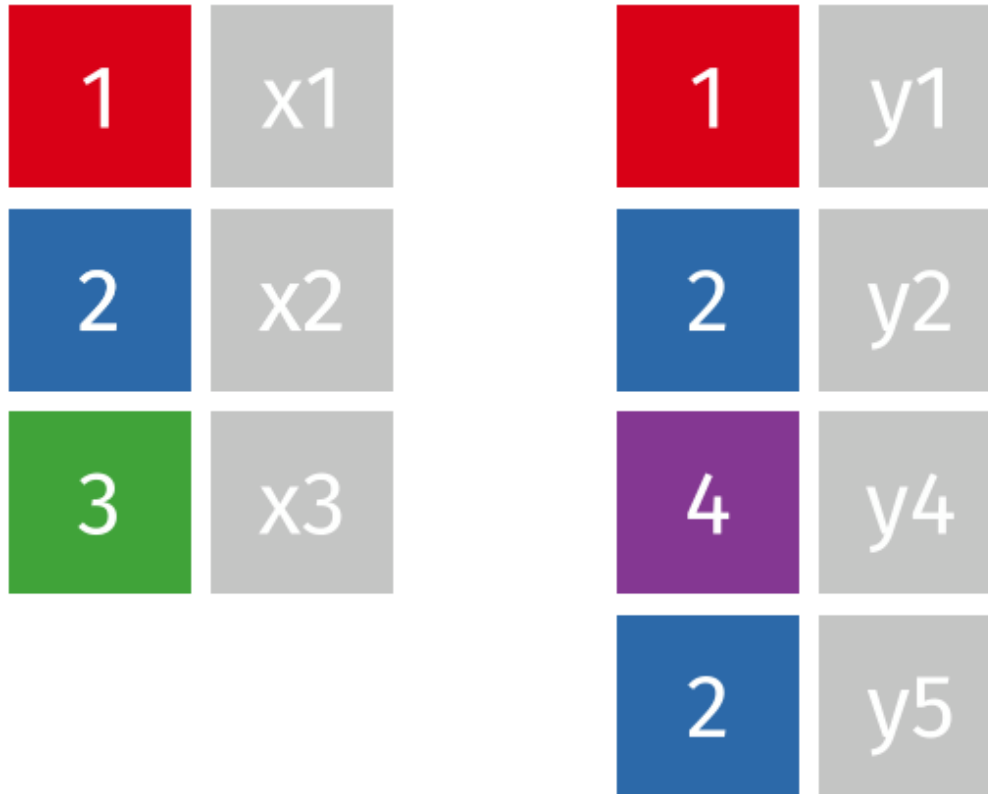
Note that “Alaska willow ptarmigan” appears twice.



“includes duplicates”

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

`left_join(x, y)`



Stop tidylog

`unloadNamespace()` does the opposite of `library()`.

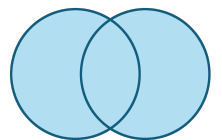
```
unloadNamespace("tidylog")
```

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>      <dbl> <chr>
1 Alabama wild turkey      NA   <NA>
2 Alaska willow ptarmigan 0.623 April
3 Alaska willow ptarmigan 0.626 May
4 Maine   <NA>          0.795 April
```

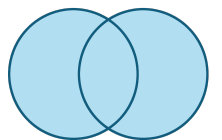


Using the **by** argument

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(x, y, by = c("a" = "b"))
```



anti_join: what's missing

Entries in data_As but not in data_cold

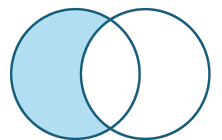
```
anti_join(data_As, data_cold, by = "State")
```

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

Entries in data_cold but not in data_As

```
anti_join(data_cold, data_As, by = "State") # order switched
```

```
# A tibble: 1 × 3
  State vacc_rate month
  <chr>    <dbl> <chr>
1 Maine      0.795 April
```



GUT CHECK!

Why use `join` functions?

- A. Combine different data sources
- B. Connect Rmd to other files
- C. Using one data source is too easy and we want our analysis ~ fancy ~

Summary

- Merging/joining data sets together - assumes all column names that overlap
 - use the `by = c("a" = "b")` if they differ
- `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
- Use the `tidylog` package for a detailed summary
- `anti_join(x, y)` shows what is only in x (missing from y)

Lab Part 2

- ▮ [Class Website](#)
- ▮ [Lab](#)
- ▮ [Day 6 Cheatsheet](#)
- ▮ [Posit's tidyr Cheatsheet](#)
- ▮ [Posit's dplyr Cheatsheet](#)



Image by [Gerd Altmann](#) from [Pixabay](#)

Additional Slides

Getting the set difference with **setdiff**

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

For this to work, the datasets need the same columns.

We'll just select the index using `select()`.

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

Getting the set difference with **setdiff**

States in A_states but not in cold_states

```
dplyr::setdiff(A_states, cold_states)
```

```
# A tibble: 1 × 1  
  State  
  <chr>  
1 Alabama
```

States in cold_states but not in A_states

```
dplyr::setdiff(cold_states, A_states)
```

```
# A tibble: 1 × 1  
  State  
  <chr>  
1 Maine
```

Getting the set difference with `setdiff`

Why did we use `dplyr::setdiff`?

There is a base R function, also called `setdiff` that requires vectors.

In other words, we use `dplyr::` to be specific about the package we want to use.

More set operations can be found here:

<https://dplyr.tidyverse.org/reference/setops.html>

Fast manipulation using **collapse** package

<https://sebkrantz.github.io/collapse/>

Might be helpful if your data is very large. However, `dplyr` and `tidyr` functions are great for most applications.