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 ~ varaible` 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

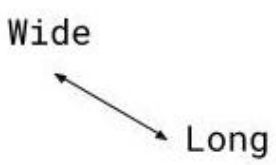# Manipulating Data

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

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

CC-BY jhudatascience.org

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>             <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 `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. 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**

```
wide_vacc <- read_csv(
  file = "https://jhudatascience.org/intro_to_r/data/wide_vacc.csv")

wide_vacc

# A tibble: 1 × 3
  June_vacc_rate May_vacc_rate April_vacc_rate
           <dbl>         <dbl>           <dbl>
1          0.516         0.514           0.511

long_vacc <- wide_vacc %>% pivot_longer(cols = everything())
long_vacc

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

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 data from **wide to long**

```
wide_vacc

# A tibble: 1 × 3
  June_vacc_rate May_vacc_rate April_vacc_rate
           <dbl>         <dbl>           <dbl>
1          0.516         0.514           0.511

long_vacc <- wide_vacc %>% pivot_longer(cols = everything(),
                                        names_to = "Month",
                                        values_to = "Rate")
long_vacc

# A tibble: 3 × 2
  Month           Rate
  <chr>          <dbl>
1 June_vacc_rate  0.516
2 May_vacc_rate   0.514
3 April_vacc_rate 0.511
```

Newly created column names are enclosed in quotation marks.

Data used: Charm City Circulator

```
circ <- read_csv("http://jhudatascience.org/intro_to_r/data/Charm_City_Circulato
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
#  9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,
#   daily <dbl>
```

Mission: Taking the average boardings by line

Let's imagine we want to create a table of average boardings by route/line. Results should look something like:

```
# A tibble: 4 × 2
  line    avg_boardings
  <chr>   <chr>
1 orange  600(?)
2 purple  700(?)
3 green   500(?)
4 banner  400(?)
```

Reshaping data from **wide to long**

```
long <- circ %>%
  pivot_longer(starts_with(c("orange","purple","green","banner")))
long

# A tibble: 13,752 × 5
   day    date        daily name            value
   <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
#  13,742 more rows
```

Reshaping data from **wide to long**

Un-pivoted columns (`day`, `date`, `daily`) are similar

```
circ %>% select(day, date, daily) %>% head()

# A tibble: 6 × 3
  day        date        daily
  <chr>      <chr>       <dbl>
1 Monday     01/11/2010   952
2 Tuesday    01/12/2010   796
3 Wednesday  01/13/2010  1212.
4 Thursday   01/14/2010  1214.
5 Friday     01/15/2010  1644
6 Saturday   01/16/2010  1490.

long %>% select(day, date, daily) %>% head()

# A tibble: 6 × 3
  day     date         daily
  <chr>   <chr>        <dbl>
1 Monday  01/11/2010    952
2 Monday  01/11/2010    952
3 Monday  01/11/2010    952
4 Monday  01/11/2010    952
5 Monday  01/11/2010    952
6 Monday  01/11/2010    952
```

Cleaning up long data

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

```
long <- long %>% mutate(
  name = str_replace(string = name, pattern = "B", replacement = "_B"),
  name = str_replace(string = name, pattern = "A", replacement = "_A")
)
long

# A tibble: 13,752 × 5
   day     date         daily name              value
   <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
#  13,742 more rows
```

Cleaning up long data with **separate()**

- first argument - which column should be split up?

- "`into =`" gives names to the new columns

- "`sep =`" to show where the separation should happen.

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

# A tibble: 13,752 × 6
   day     date        daily line   type        value
   <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
#  13,742 more rows
```

Mission: Taking the average boardings by line

Filter by Boardings only..

```r
long <- long %>%
  filter(type == "Boardings")
long
```

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

Mission: Taking the average boardings by line

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

```
long %>%
  group_by(line) %>%
  summarize("avg_boardings" = mean(value, na.rm = TRUE))

# A tibble: 4 × 2
  line    avg_boardings
  <chr>           <dbl>
1 banner           830.
2 green           1929.
3 orange          3031.
4 purple          4127.
```

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.

```
circ %>%
  pivot_longer( !c(day, date, daily))

# A tibble: 13,752 × 5
   day    date         daily name             value
   <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
#  13,742 more rows
```

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

```
long_vacc

# A tibble: 3 × 2
  Month              Rate
  <chr>             <dbl>
1 June_vacc_rate   0.516
2 May_vacc_rate    0.514
3 April_vacc_rate  0.511

wide_vacc <- long_vacc %>% pivot_wider(names_from = "Month",
                                       values_from = "Rate")
wide_vacc

# A tibble: 1 × 3
  June_vacc_rate May_vacc_rate April_vacc_rate
           <dbl>         <dbl>           <dbl>
1          0.516         0.514           0.511
```

Reshaping Charm City Circulator

```
long

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

Reshaping Charm City Circulator

```
wide <- long %>% pivot_wider(names_from = "line",
                             values_from = "value")
wide

# A tibble: 1,146 × 8
   day       date       daily type      orange purple green banner
   <chr>     <chr>      <dbl> <chr>      <dbl>  <dbl>  <dbl> <dbl>
 1 Monday    01/11/2010  952  Boardings  877     NA     NA    NA
 2 Tuesday   01/12/2010  796  Boardings  777     NA     NA    NA
 3 Wednesday 01/13/2010 1212. Boardings 1203     NA     NA    NA
 4 Thursday  01/14/2010 1214. Boardings 1194     NA     NA    NA
 5 Friday    01/15/2010 1644  Boardings 1645     NA     NA    NA
 6 Saturday  01/16/2010 1490. Boardings 1457     NA     NA    NA
 7 Sunday    01/17/2010  888. Boardings  839     NA     NA    NA
 8 Monday    01/18/2010 1000. Boardings  999     NA     NA    NA
 9 Tuesday   01/19/2010 1035  Boardings 1023     NA     NA    NA
10 Wednesday 01/20/2010 1396. Boardings 1375     NA     NA    NA
# ⎯ 1,136 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 =`
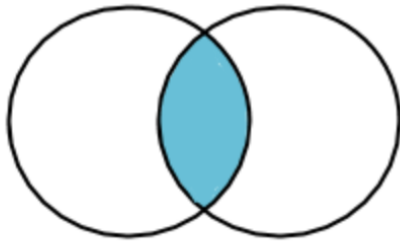
Lab Part 1

 Class Website

 Lab

Joining

"Combining datasets"
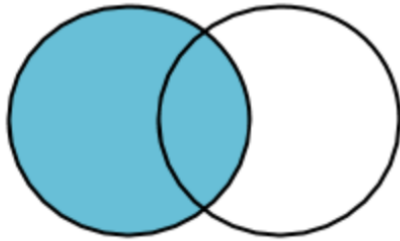
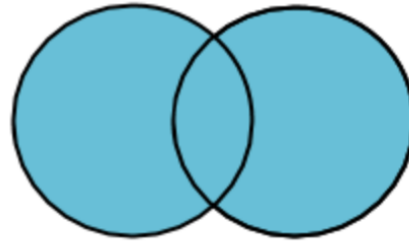Left Join
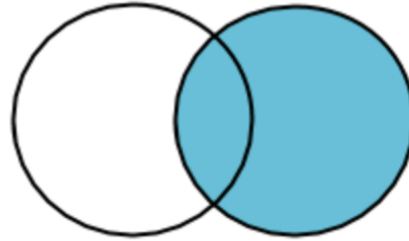
Right Join

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://jhudatascience.org/intro_to_r/data/data_As_1.csv")
data_cold <- read_csv(
  file = "https://jhudatascience.org/intro_to_r/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
```
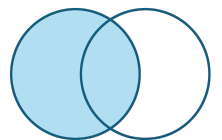
Inner Join

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)

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4
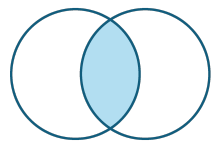
Left Join

"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   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
```

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
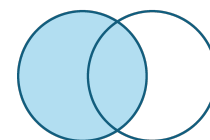
Right Join

"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
```
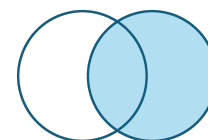
Left Join: Switching arguments

```
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
```
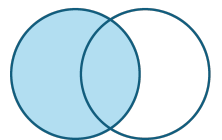
Full Join

https://raw.githubusercontent.com/gadenbuie/tidyexplain/main/images/full-join.gif
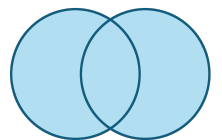
Full Join

```
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
```

Watch out for "`includes duplicates`"

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

data_As
```

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

```r
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
```

Watch out for "**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
```
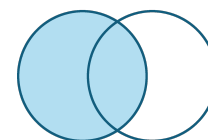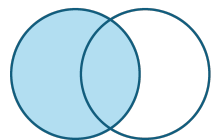
Watch out for "`includes duplicates`"

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

```
lj

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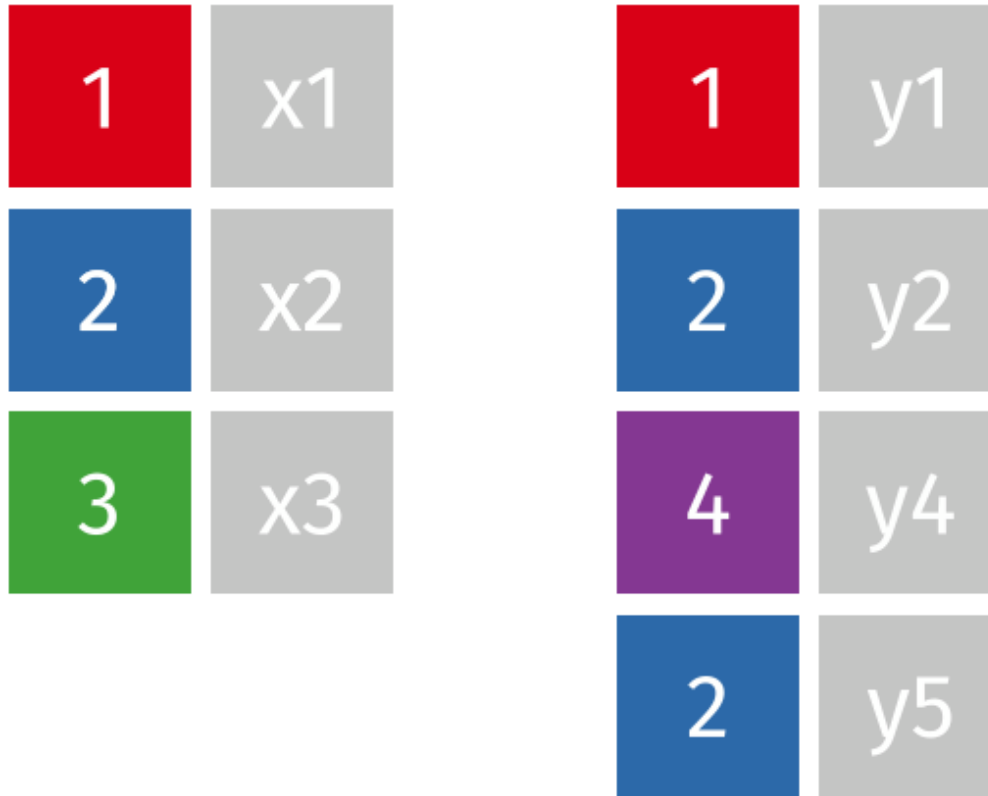
Note that "Alaska willow ptarmigan" appears twice.

Watch out for "**includes duplicates**"

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

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
```
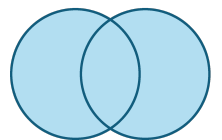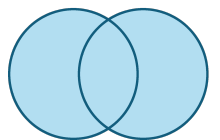
Using the **by** argument

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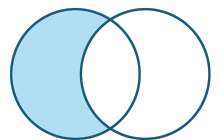
```
anti_join(data_As, data_cold, by = "State")

# 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
```

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
- `antijoin(x, y)` shows what is only in `x` (missing from `y`)

Lab Part 2

 Class Website

 Lab

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

Inconsistencies in non-pivoted columns?

Notice "daily" column has different values

```
long2

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

Inconsistencies in non-pivoted columns?

R won't drop data while pivoting.

```
wide2 <- long2 %>% pivot_wider(names_from = "type",
                               values_from = "value")
wide2
```

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

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.