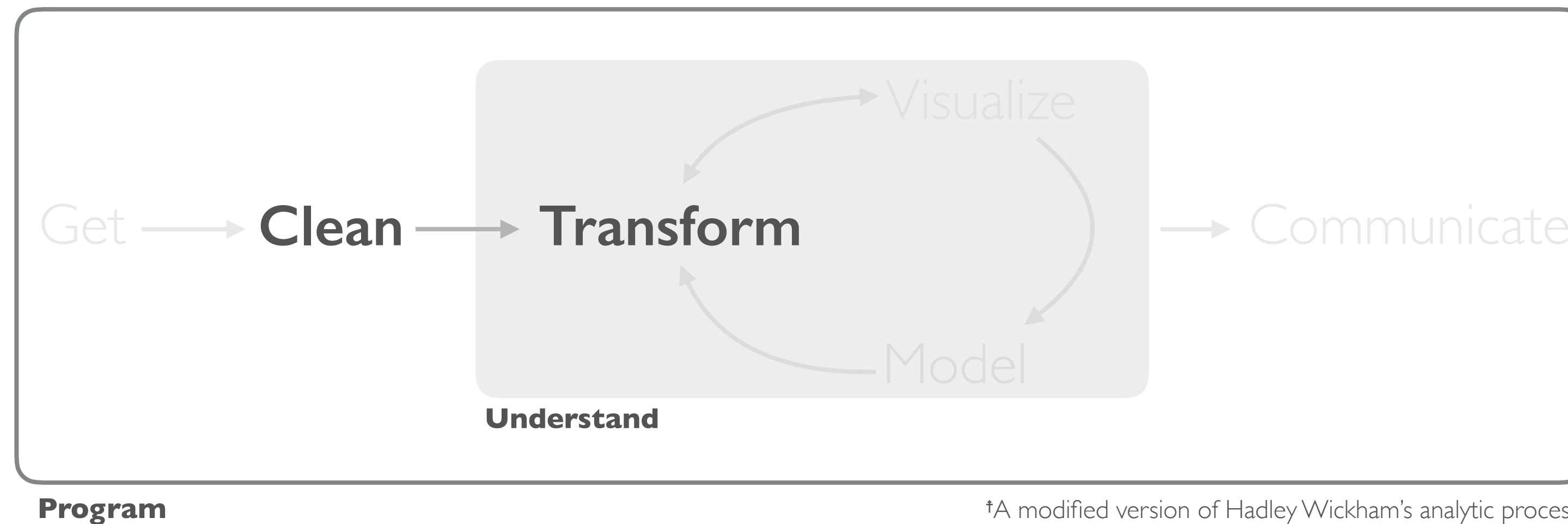


RELATIONAL DATA



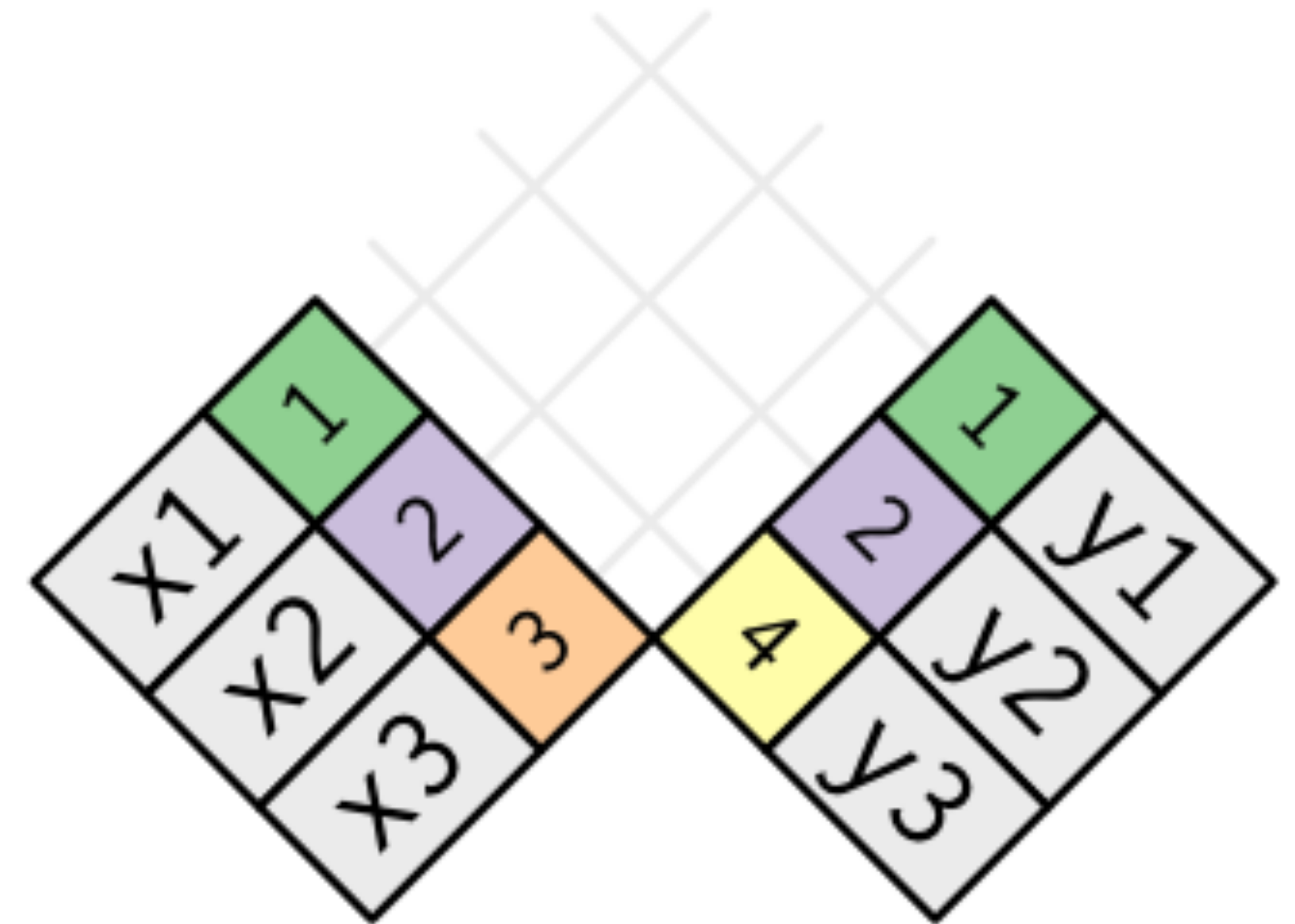
†A modified version of Hadley Wickham's analytic process

“It’s rare that a data analysis involves only a single table of data. Typically you have many tables of data, and you must combine them to answer the questions that you’re interested in.”

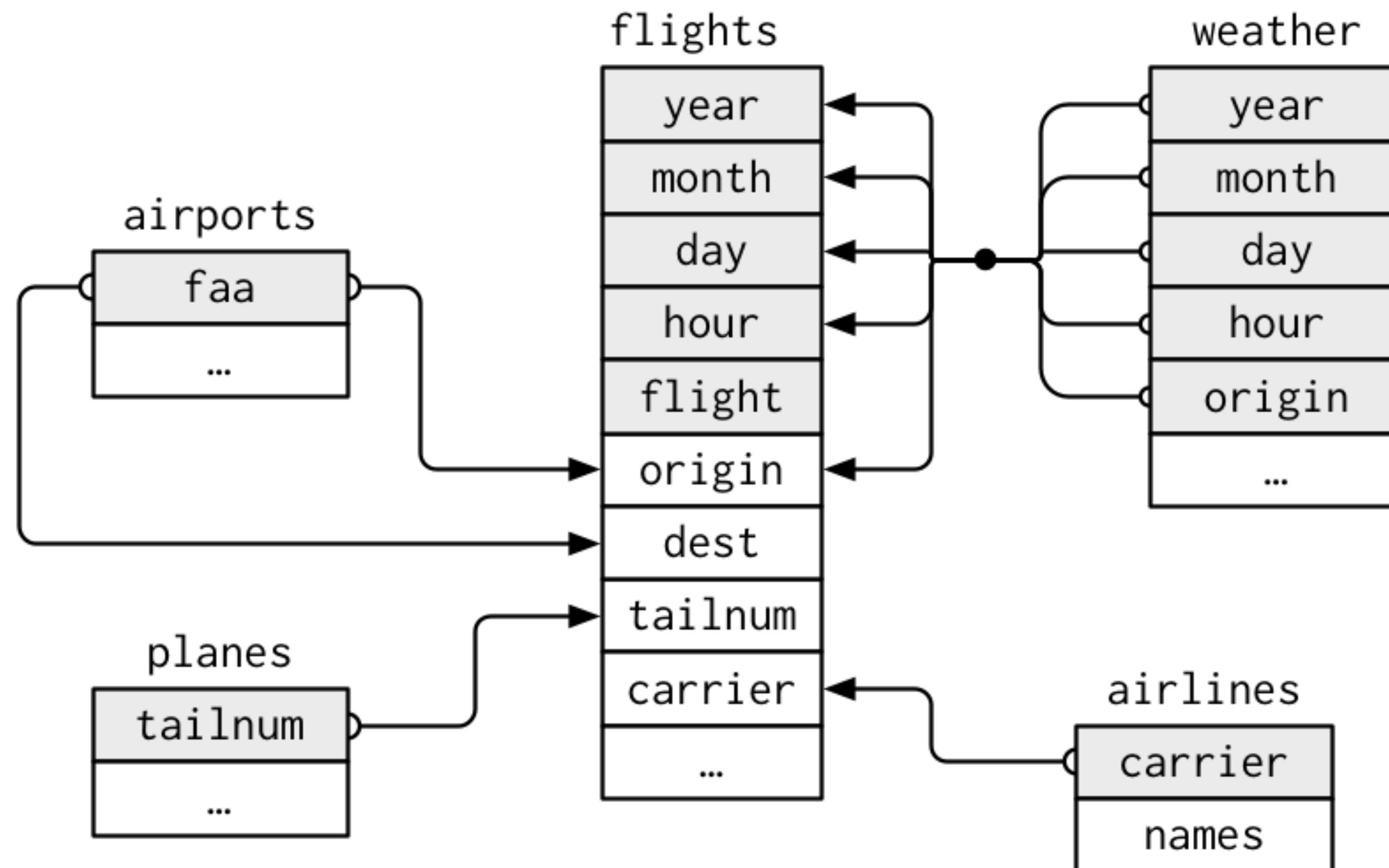
- Garrett Grolemund & Hadley Wickham

WHAT IS RELATIONAL DATA?

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3



WHAT IS RELATIONAL DATA?



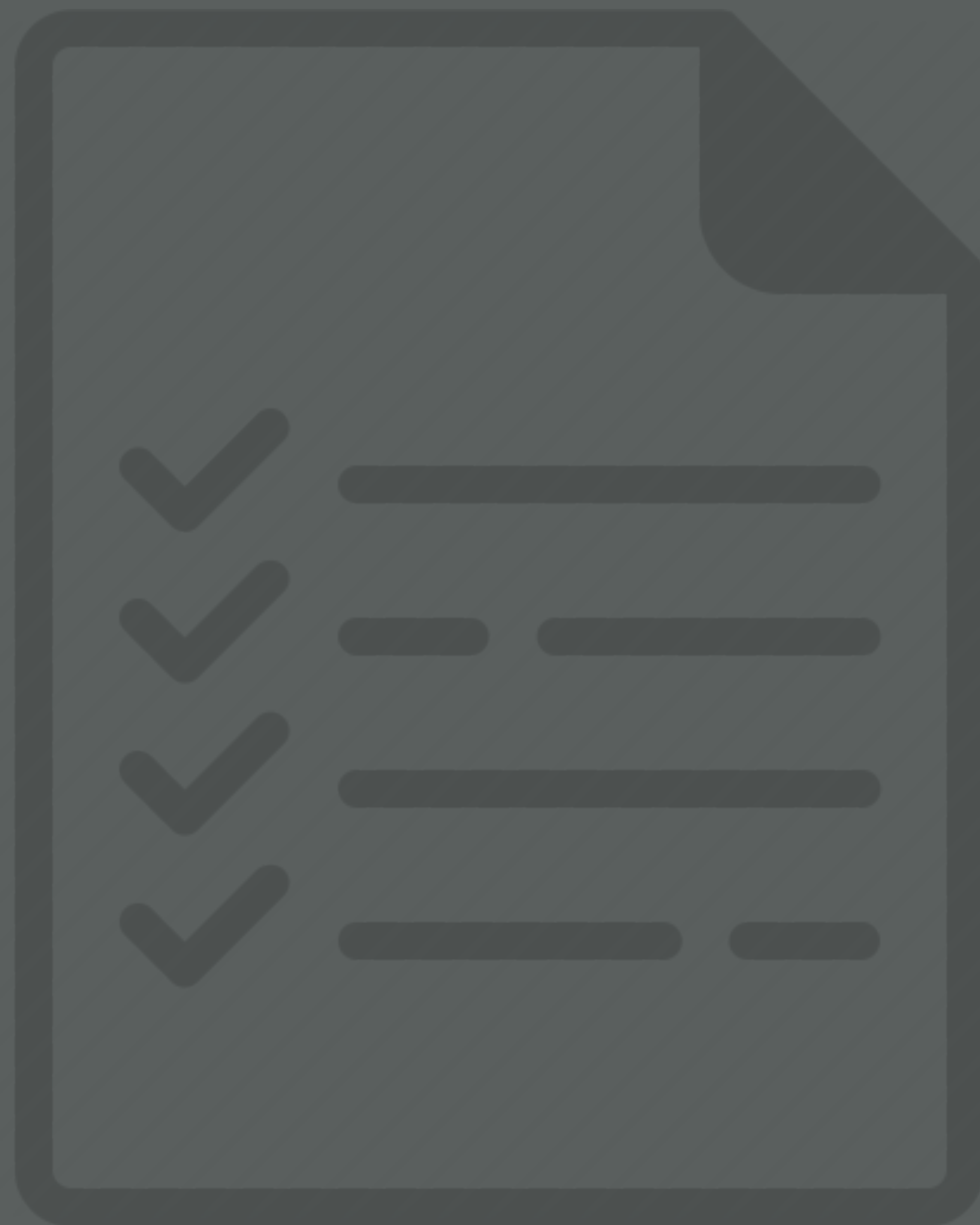
TYPES OF JOINS

To work with relational data you need join operations that work with pairs of tables. There are three families of verbs designed to work with relational data:

- **Mutating joins:** add new variables to one data frame by matching observations in another.
- **Filter joins:** filter observations from one data frame based on whether or not they match an observation in the other table.
- **Set operations:** treat observations as if they were set elements



PREREQUISITES



PACKAGE PREREQUISITE

```
library(nycflights13)
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages -----
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

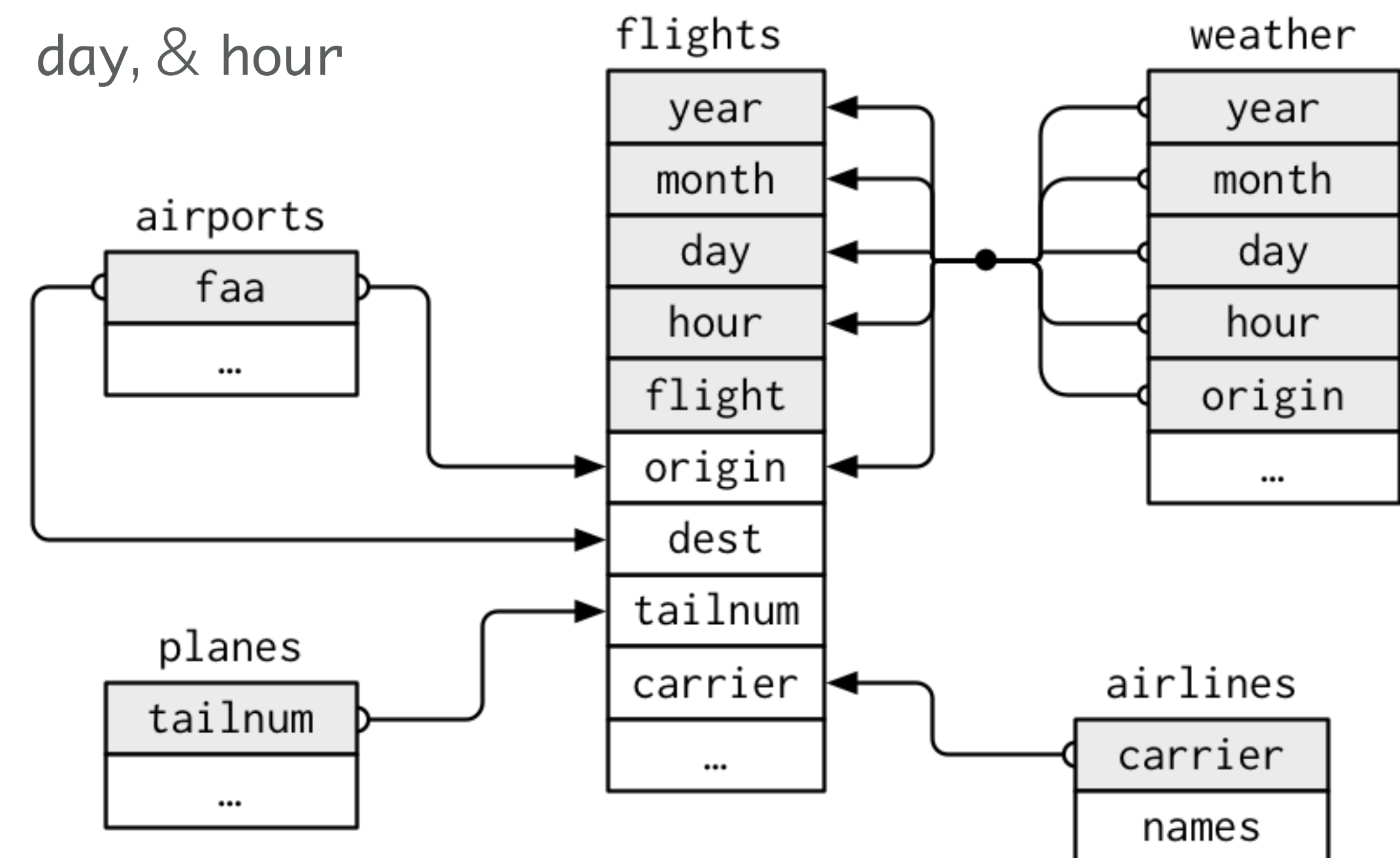
EXAMPLE DATA PREREQUISITE

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3

```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  3, "x3"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2",
  4, "y3"
)
```

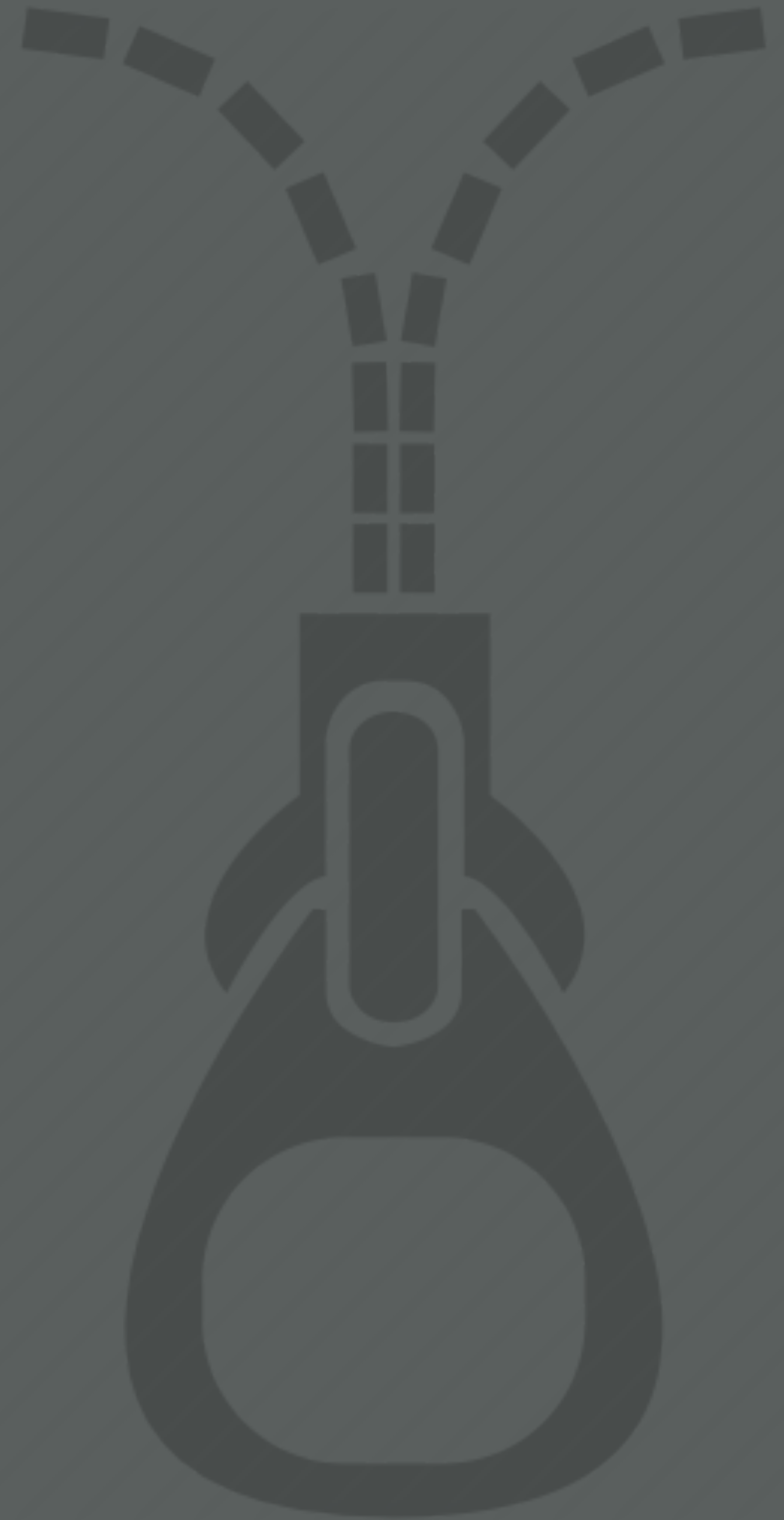

EXERCISE DATA PREREQUISITE

- For nycflights13:
 - flights connects to planes via tailnum
 - flights connects to airlines via carrier
 - flights connects to airports via origin & dest
 - flights connects to weather via origin, year, month, day, & hour



MUTATING JOINS

Adding variables



INNER JOIN

- Simplest type of join
- matches pairs of observations whenever their **keys** are equal
- **keys** are variables that connect pairs of tables

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3

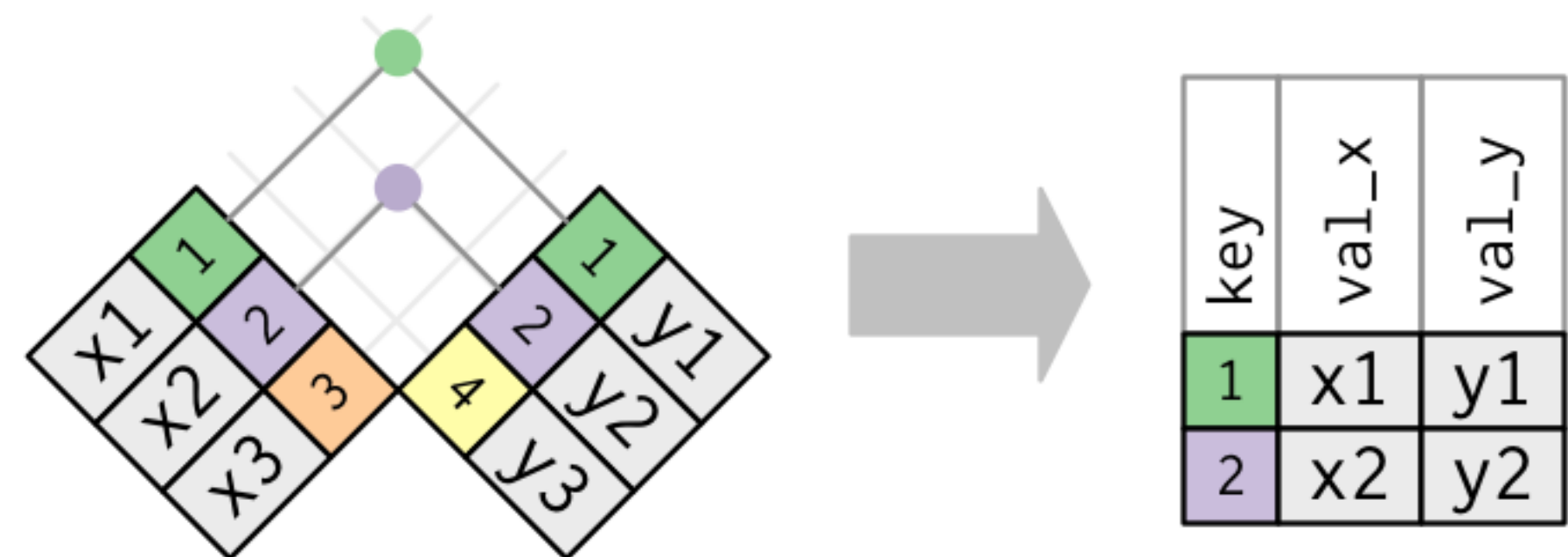
INNER JOIN

- use **by** to tell dplyr which variable is the **key**
- unmatched rows are not included in the result

```
x %>% inner_join(y, by = "key")
```

```
# A tibble: 2 x 3
```

	key	val_x	val_y
	<dbl>	<chr>	<chr>
1	1	x1	y1
2	2	x2	y2

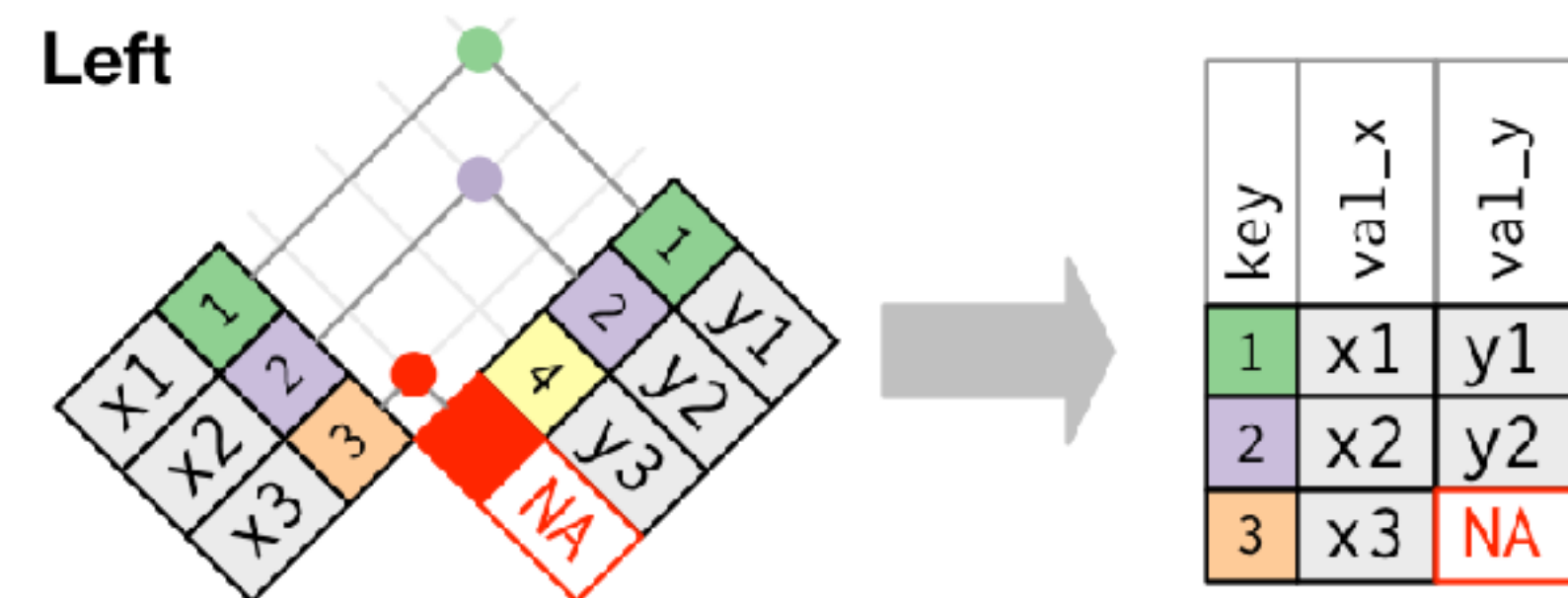


OUTER JOINS

- Outer joins keep observations that appear in at least one of the tables
- There are 3 types of outer joins:

OUTER JOINS

- Outer joins keep observations that appear in at least one of the tables
- There are 3 types of outer joins:
 - left join**: keeps all observations in x



```
x %>% left_join(y, by = "key")
```

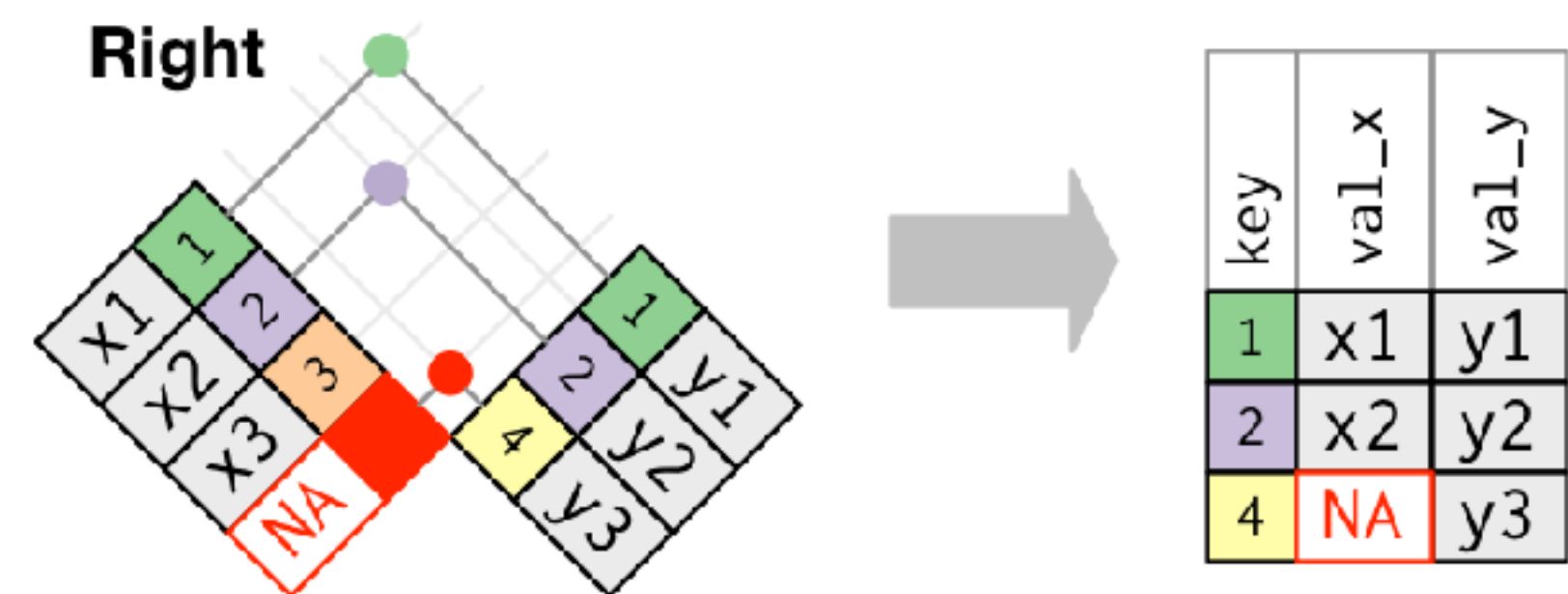
```
# A tibble: 3 x 3
```

```
  key val_x val_y
<dbl> <chr> <chr>
1     1    x1    y1
2     2    x2    y2
3     3    x3  <NA>
```

Note how missing values get filled in with NA

OUTER JOINS

- Outer joins keep observations that appear in at least one of the tables
- There are 3 types of outer joins:
 - left join: keeps all observations in x
 - **right join**: keeps all observations in y



```
x %>% right_join(y, by = "key")
```

```
# A tibble: 3 × 3
```

```
  key val_x val_y
<dbl> <chr> <chr>
1     1    x1    y1
2     2    x2    y2
3     4  <NA>    y3
```

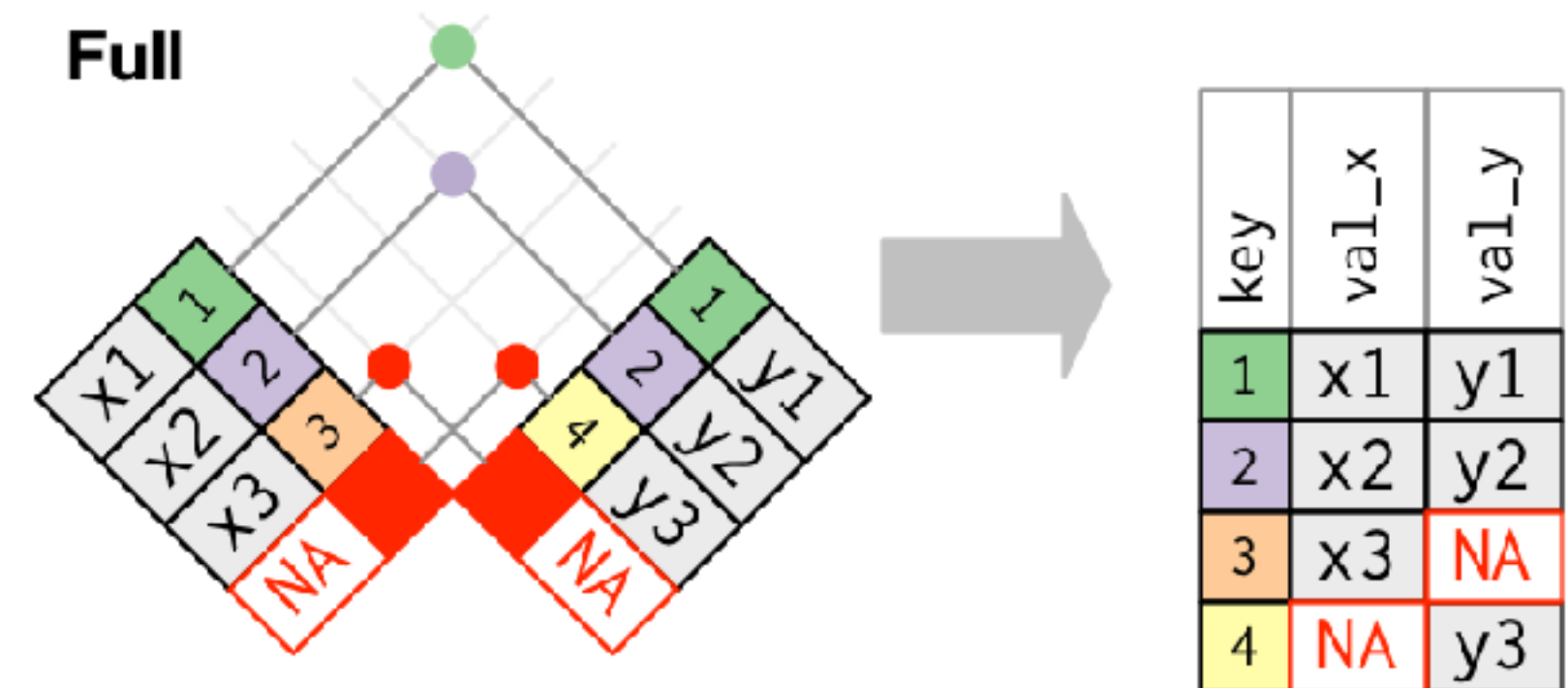
OUTER JOINS

- Outer joins keep observations that appear in at least one of the tables
- There are 3 types of outer joins:
 - left join: keeps all observations in x
 - right join: keeps all observations in y
 - full join**: keeps all observations in x and y

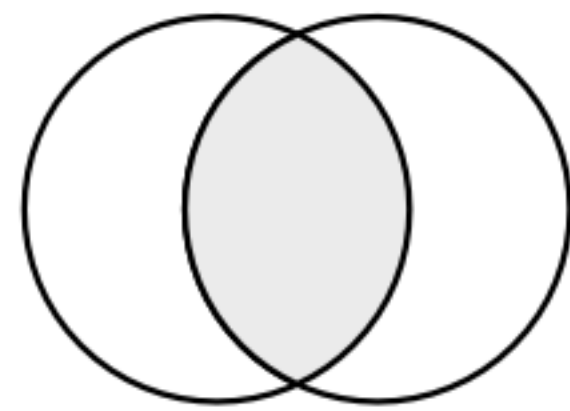
```
x %>% full_join(y, by = "key")
```

```
# A tibble: 4 x 3
```

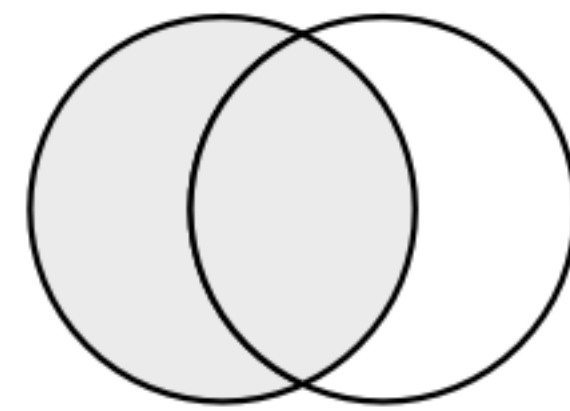
```
  key val_x val_y
<dbl> <chr> <chr>
1     1    x1    y1
2     2    x2    y2
3     3    x3   <NA>
4     4   <NA>   y3
```



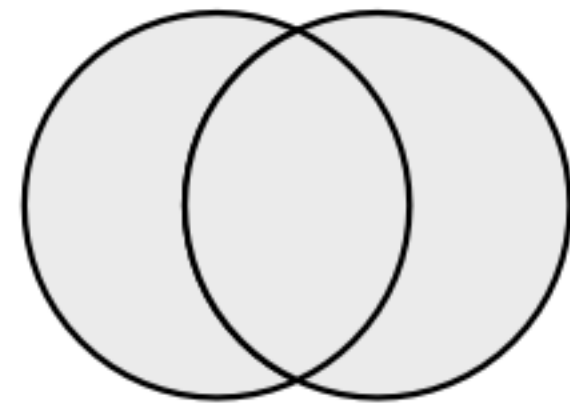
COMPARING JOINS



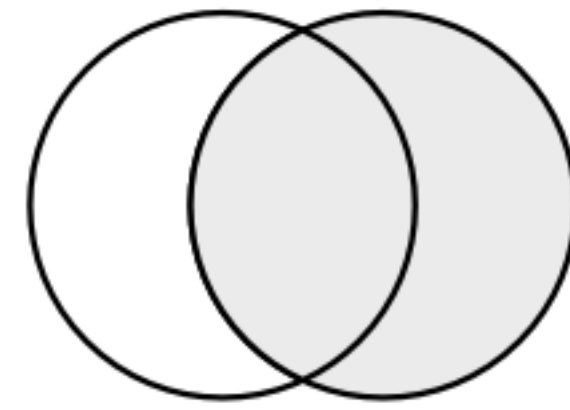
inner_join(x, y)



left_join(x, y)



full_join(x, y)



right_join(x, y)

DEFINING KEYS

- What if our **key names** don't match?

```
x <- tribble(
  ~key1, ~val_x,
    1, "x1",
    2, "x2",
    3, "x3"
)
y <- tribble(
  ~key2, ~val_y,
    1, "y1",
    2, "y2",
    4, "y3"
)
```

DEFINING KEYS

- What if our **keys** don't match?

```
x %>% inner_join(y, by = c("key1" = "key2"))  
# A tibble: 2 x 3  
  key1 val_x val_y  
  <dbl> <chr> <chr>  
1     1    x1    y1  
2     2    x2    y2
```

YOUR TURN!

1. take the flights data and then
 1. left join airlines data
 2. filter for “Virgin America”
 3. group by time_hour
 4. summarise data by computing the mean dep_delay
 5. create ggplot with $x = \text{time_hour}$, $y = \text{average dep_delay}$, and create a line plot with geom_line
2. Can you figure out how to add the location of the origin and destination (i.e. the **lat** and **lon**) from **airports** to **flights** data? Hint: use two consecutive left_joins.

FILTERING JOINS

Filtering variables based on another data set

A	0
B	1
C	1
D	0
E	1
F	0
G	1

A	1
B	0
C	0
D	0
E	1
F	1
G	1

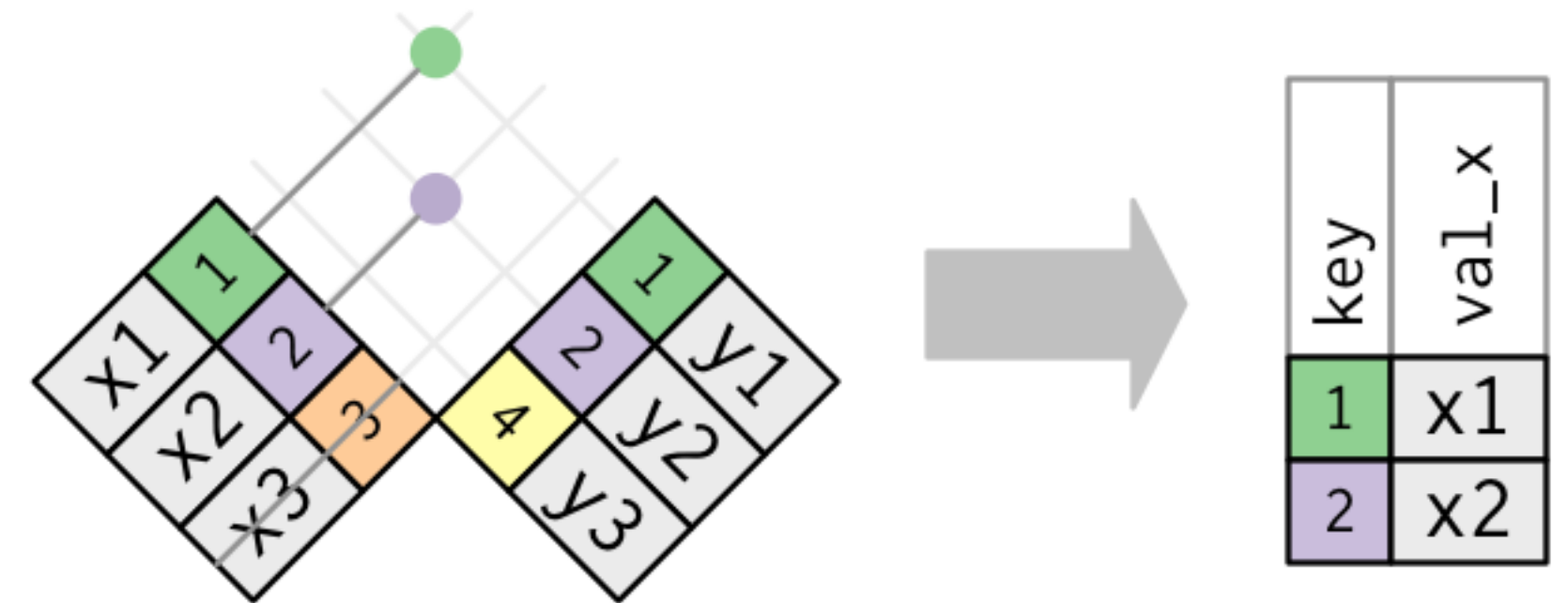
FILTERING JOINS

- Filtering joins affect the observations rather than adding variables
- There are 2 types of filtering joins:

FILTERING JOINS

- Filtering joins affect the observations rather than adding variables
- There are 2 types of filtering joins:
 - **semi join**: keeps all observations in x that have a match in y

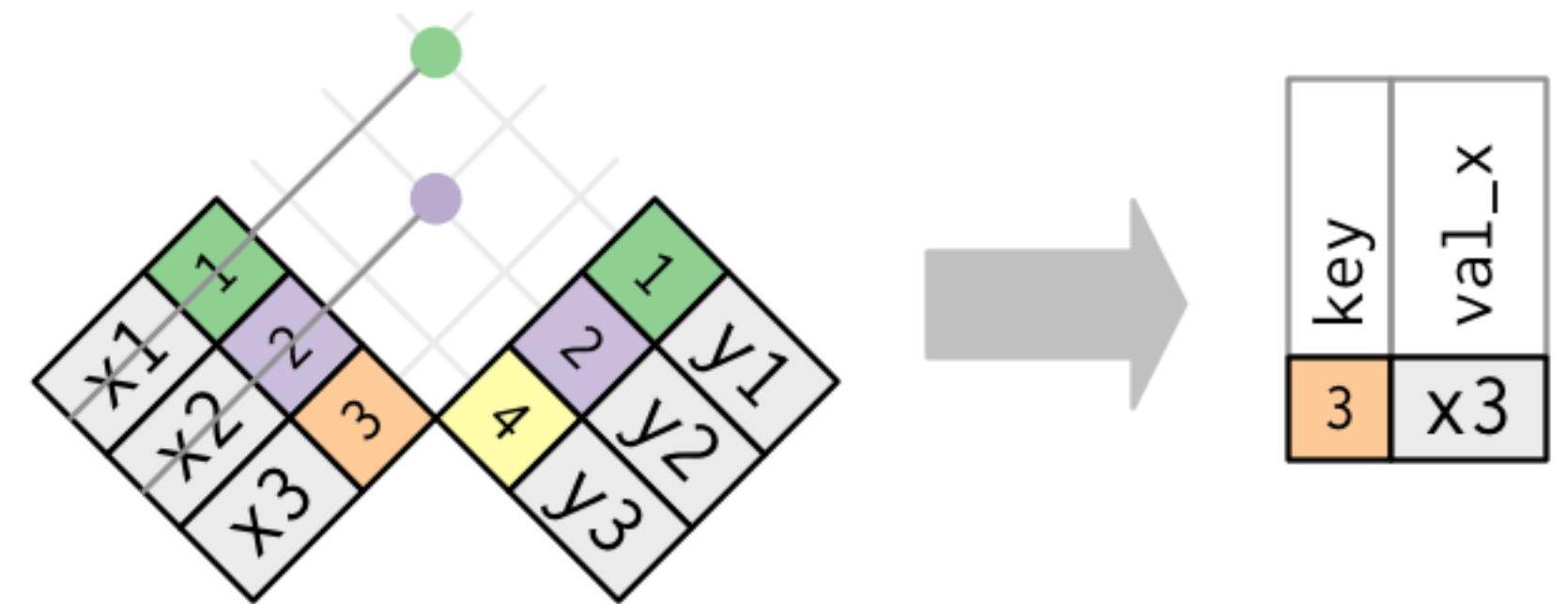
```
x %>% semi_join(y, by = "key")  
# A tibble: 2 × 2  
  key val_x  
  <dbl> <chr>  
1     1   x1  
2     2   x2
```



FILTERING JOINS

- Filtering joins affect the observations rather than adding variables
- There are 2 types of filtering joins:
 - semi join: keeps all observations in x that have a match in y
 - **anti join**: drops all observations in x that have a match in y

```
x %>% anti_join(y, by = "key")  
# A tibble: 1 x 2  
  key val_x  
  <dbl> <chr>  
1     3   x3
```



YOUR TURN!

1. How many flights in the **flights** data have matching **plane** metadata (**tailnum** is your key)? How many do not? Hint: use **tally()** after your joining functions.
2. Filter the **airports** data for those airports that do not have matching destination values in the **flights** data (**faa** and **dest** are your keys). How many unique airports do you find? Hint: use the **distinct()** and **tally()** functions after your joining function.

SET OPERATIONS

Treat observations as set elements



SET OPERATIONS

- I use these least frequently
- Compares **entire row** in each data set

SET OPERATIONS

- I use these least frequently
- Compares **entire row** in each data set
- `intersect(x, y)`: return only observations in both **x** and **y**.
- `union(x, y)`: return unique observations in **x** and **y**.
- `setdiff(x, y)`: return observations in **x**, but not in **y**

Illustrate with these
two data sets

```
df1 <- tribble(
  ~x, ~y,
  1,  1,
  2,  1
)
df2 <- tribble(
  ~x, ~y,
  1,  1,
  1,  2
)
```

SET OPERATIONS

- I use these least frequently
- Compares **entire row** in each data set
- **intersect(x, y)**: return only observations in both **x** and **y**.
- **union(x, y)**: return unique observations in **x** and **y**.
- **setdiff(x, y)**: return observations in **x**, but not in **y**

```
intersect(df1, df2)
```

```
# A tibble: 1 × 2
```

	x	y
	<dbl>	<dbl>
1	1	1

Illustrate with these
two data sets

```
df1 <- tribble(
  ~x, ~y,
  1,  1,
  2,  1
)
df2 <- tribble(
  ~x, ~y,
  1,  1,
  1,  2
)
```

SET OPERATIONS

- I use these least frequently
- Compares **entire row** in each data set
- `intersect(x, y)`: return only observations in both `x` and `y`.
- `union(x, y)`: return unique observations in `x` and `y`.
- `setdiff(x, y)`: return observations in `x`, but not in `y`

```
union(df1, df2)
# A tibble: 3 × 2
      x     y
  <dbl> <dbl>
1     1     2
2     2     1
3     1     1
```

Illustrate with these
two data sets

```
df1 <- tribble(
  ~x, ~y,
  1,  1,
  2,  1
)
df2 <- tribble(
  ~x, ~y,
  1,  1,
  1,  2
)
```

SET OPERATIONS

- I use these least frequently
- Compares **entire row** in each data set
- `intersect(x, y)`: return only observations in both `x` and `y`.
- `union(x, y)`: return unique observations in `x` and `y`.
- `setdiff(x, y)`: return observations in `x`, but not in `y`

```
setdiff(df1, df2)
# A tibble: 1 x 2
      x     y
  <dbl> <dbl>
1     2     1
```

Illustrate with these
two data sets

```
df1 <- tribble(
  ~x, ~y,
  1,  1,
  2,  1
)
df2 <- tribble(
  ~x, ~y,
  1,  1,
  1,  2
)
```

CHALLENGE



DOES WEATHER INFLUENCE DELAYS?

- I. Using the `%>%` operator, take the `flights` data and then
 - i. left join `airlines` data using `carrier` as the key
 - ii. Filter for only Virgin America flights with a departure delay greater than one hour
 - iii. inner join `weather` data with the keys: `by = c("year", "month", "day", "origin", "hour")`
 - iv. select `dep_delay` and `temp:visib`
 - v. gather the `temp:visib` variables into one variable
 - vi. Plot the relationships between departure delay and each of the weather variables (dewpoint, humidity, precipitation, pressure, temp, visibility, wind direction, wind gust, and wind speed).
 - vii. Can you use faceting to compare all these relationships at once?

WHAT TO REMEMBER



FUNCTIONS TO REMEMBER

Operator/Function	Description
<code>inner_join</code> , <code>left_join</code> , <code>right_join</code> , <code>full_join</code>	mutating join: add new variables to one data frame by matching observations in another.
<code>semi_join</code> , <code>anti_join</code>	filtering joins: filter observations from one data frame based on whether or not they match an observation in the other table
<code>intersect</code> , <code>union</code> , <code>setdiff</code>	set operations: treat observations as if they were set elements