Relational Data with dplyr

Outline

Intro to relational data

Mutating Joins

Filtering Joins

Set Operations

Relational Data

Relational data - multiple tables of data that share some common variables / attributes (relations and not individual datasets that are important)

A relation will always be between a pair of tables.

Subsequent relations will build off of relations of pairs.

Verbs to work with pairs of tables:

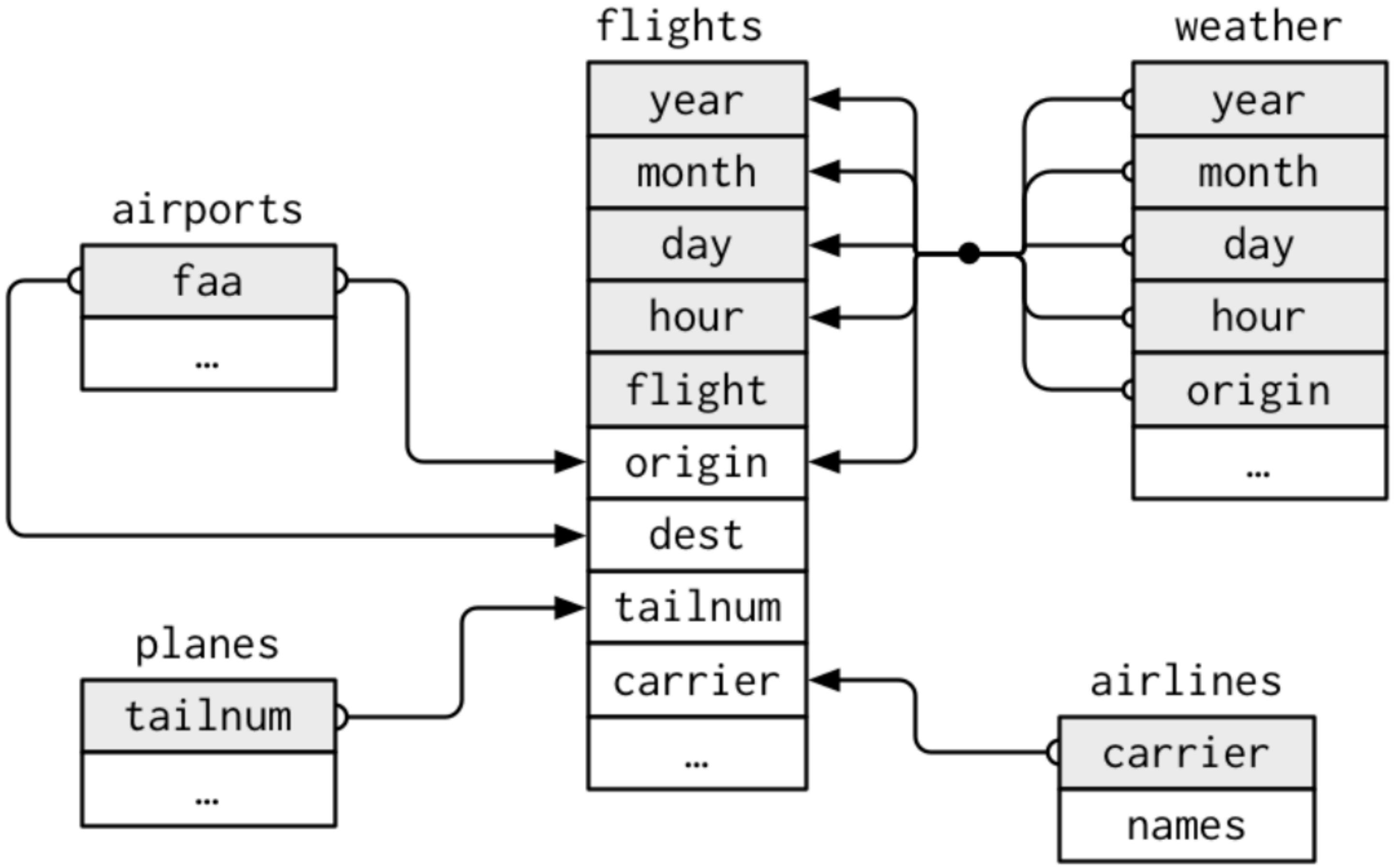
Mutating joins - add new variables from matching observations in another

Filtering joins - filter observations from one dataset based on whether or not they match an observation in the other table

Set operations - treats observations like set elements

nycflights13

The *nycflights13* R package contains similar info as the Houston flights data but also contains data frames about the airlines, weather, and airports.



Credit: R4DS

Keys

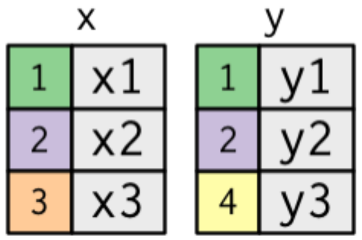
The variables we use to connect the datasets are called keys

Three types of keys:

- Primary Key - uniquely identifies an observation in its own table
- Foreign Key - uniquely identifies an observation in another table
- A variable can be both a primary and foreign key!
- Surrogate Key - a key that is created because the dataset lacks a primary key

A primary key and the corresponding foreign key in another table form the *relation.*

Mutating Joins



x

1	x1
2	x2
3	x3

y

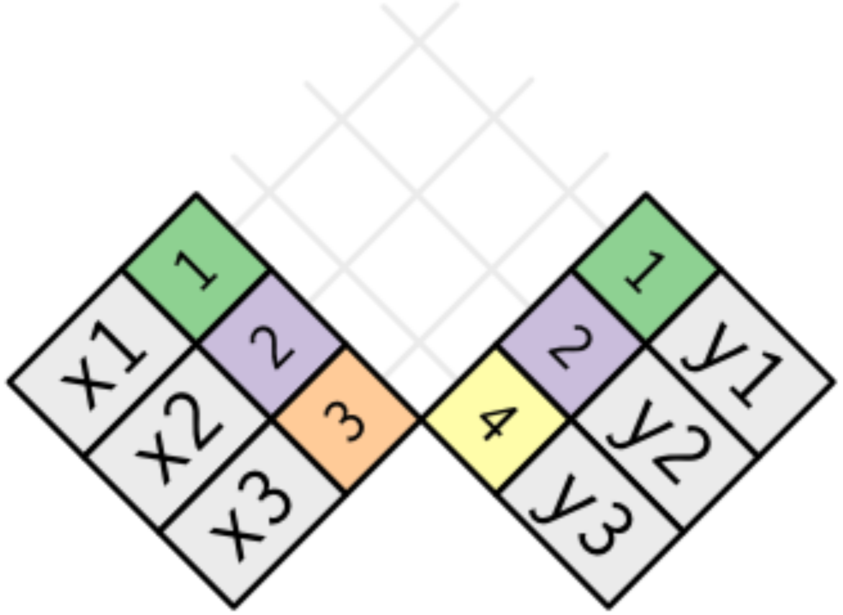
1	y1
2	y2
4	y3

Setup:

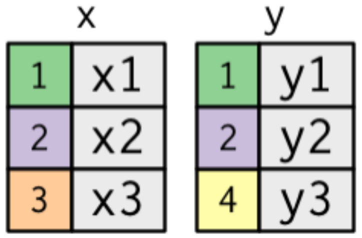
```
x <- tribble(
    ~key, ~val_x,
    1, "x1",
    2, "x2",
    3, "x3"
)
```

```
y <- tribble(
    ~key, ~val_y,
    1, "y1",
    2, "y2",
    4, "y3"
)
```

A join is a way of connecting the rows in x to 0,1, or more rows in y



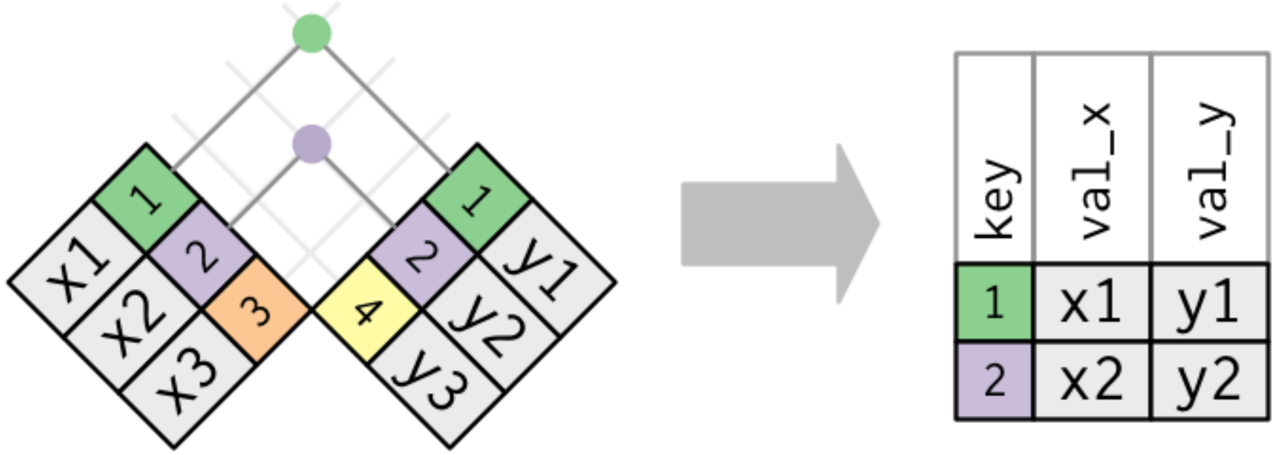
Mutating Joins



x	
1	x1
2	x2
3	x3

y	
1	y1
2	y2
4	y3

inner_join() - matches pairs of observations whenever their keys are equal (drops everything else)



key	val_x	val_y
1	x1	y1
2	x2	y2

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

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

Mutating Joins

Outer joins (left_join(), right_join(), full_join()) - keeps observations that appear in at least one of the tables

left_join() - keeps all observations in x (left dataset)

right_join() - keeps all observations in y (right dataset)

full_join() - keeps all observations in both datasets

Mutating Joins

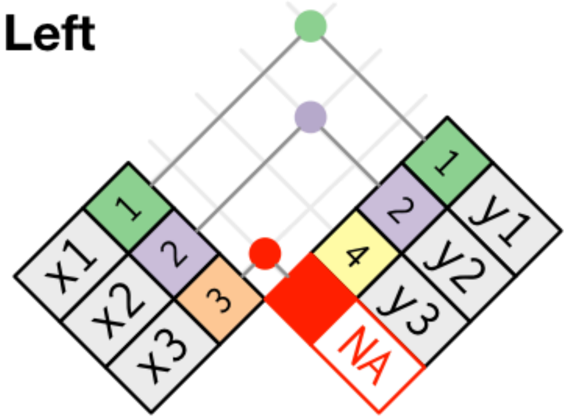
x

1	x1
2	x2
3	x3

y

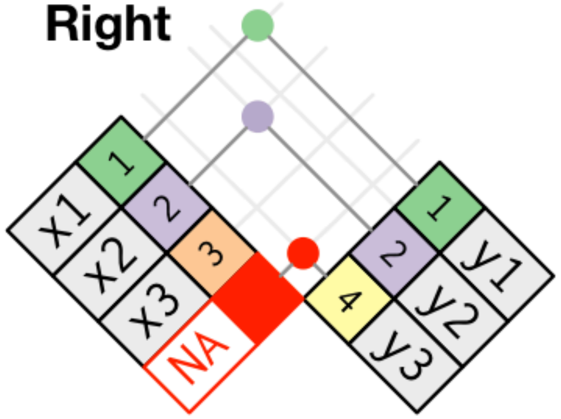
1	y1
2	y2
4	y3

Left



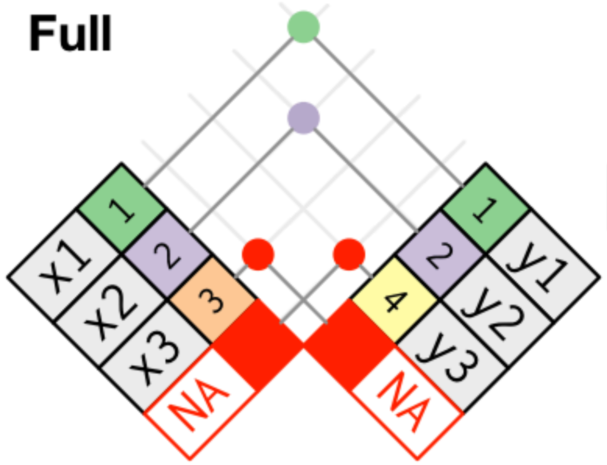
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

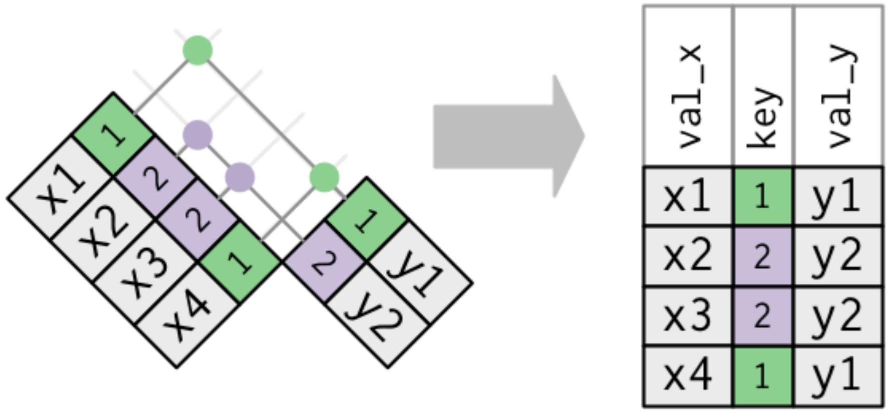
Full



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

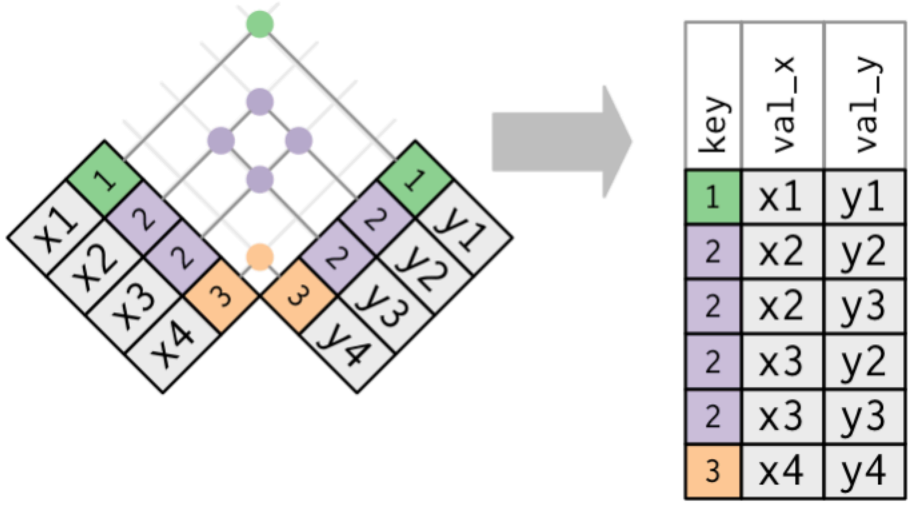
Mutating Joins

Duplicate keys - When the keys are not unique.



val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

When you join duplicate keys, you get all possible combos



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Defining Key Columns

There are multiple ways to connect tables with the “by” arg.

`by = NULL` - uses all variables in both datasets (natural join)

`by = "x"` - a character vector. Uses only variables defined in key

`by = c("a" = "b")` - named character vector. Will match variable *a* in table *x* to variable *b* in table *y*

`by = join_by(a == b)` - The join_by() function is a new function that specifies all types of joins. This will match variable *a* in table *x* to variable *b* in table *y* as well

Your Turn!

The dplyr package has two datasets called *band_members* and *band_instruments* that contain information about people from famous bands.

Use a natural join with the 4 different mutating join functions to see how the output differs.

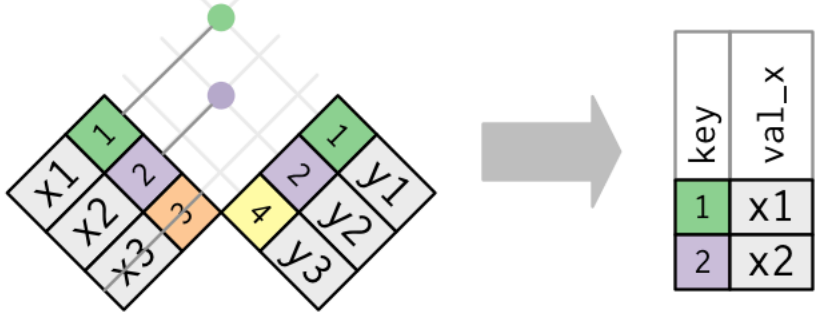
Filtering Joins

semi_join() and anti_join()

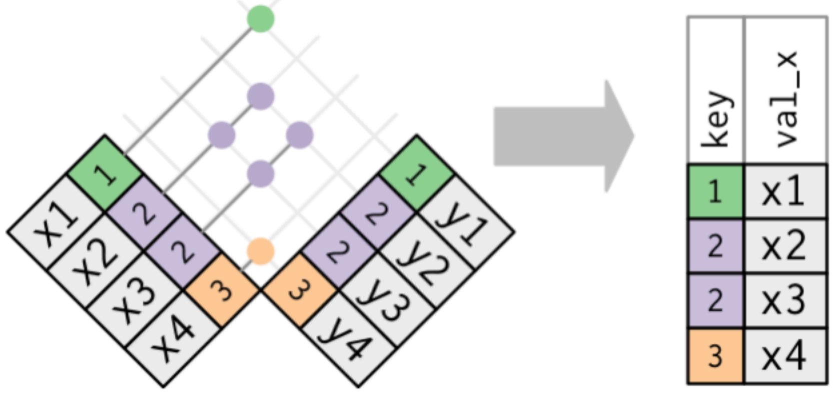
semi_join() - keeps all observations in x that have a match in y

anti_join() - drops all observations in x that have a match in y

semi_join():

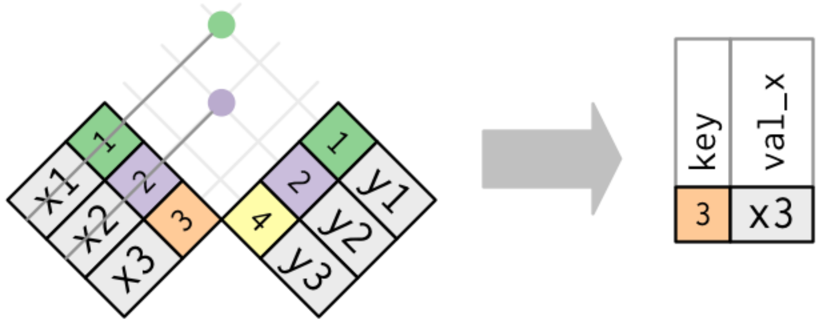


key	val_x
1	x1
2	x2



key	val_x
1	x1
2	x2
2	x3
3	x4

anti_join():



key	val_x
3	x3

Your Turn!

The dplyr package has two datasets called *band_members* and *band_instruments* that contain information about people from famous bands.

Use a natural join with the 2 different filtering join functions to see how the output differs.

Set Operations

Set operations can be helpful when you break down a single complex filter into simpler filters

intersect(x, y) - returns only observations in both x and y

union(x, y) - returns unique observations in x and y

setdiff(x, y) - returns observations in x, but not y