Relational data

Four main types of operations with two tables

- **Binding,** which simply stacks tables on top of or beside each other
- **Mutating joins**, which add new variables to one data frame from matching observations in another.
- **Filtering joins**, which filter observations from one data frame based on whether or not they match an observation in the other table.
- **Set operations**, which treat observations as if they were set elements.

Row binds and column binds

- Row binding (e.g. dplyr::bind_rows() or base::rbind())
  - Different approaches for row binding have different combinations of flexibility vs rigidity
  - Usually works when it should and usually doesn’t when it shouldn’t

- Column binding (e.g. dplyr::bind_cols() or base::cbind())
  - Your job to ensure the rows are aligned – easy to make mistakes
  - Dangerous – avoid whenever possible
  - Use join functions instead

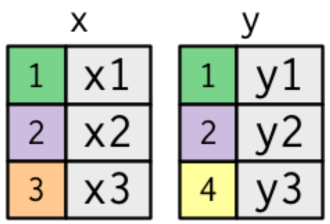
Keys

- A variable **(or set of variables)** that uniquely identifies an observation
    - A **primary key** uniquely identifies an observation in its own table [can be a set of variables]. For example, planes$tailnum is a primary key because it uniquely identifies each plane in the planes table.
    - A **foreign key** uniquely identifies an observation in another table [can be a set of variables]. For example, the flights$tailnum is a foreign key because it appears in the flights table where it matches each flight to a unique plane.

Exercise

- We just added the full airline names to our flights2 dataset. Now try to instead add information about each plane from the dataset named planes

Understanding joins



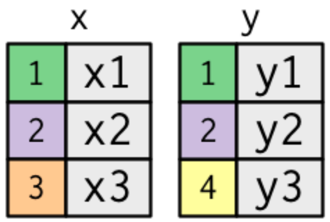
x

1	x1
2	x2
3	x3

y

1	y1
2	y2
4	y3

Understanding joins



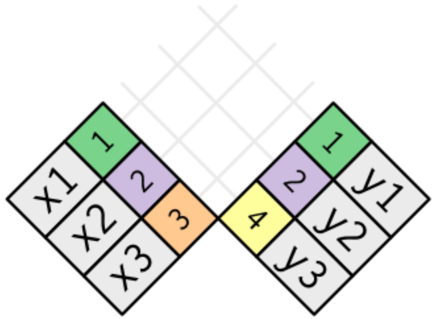
x

1	x1
2	x2
3	x3

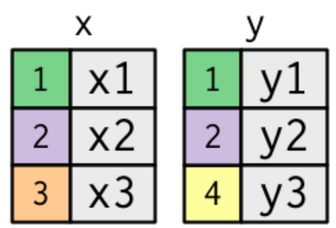
y

1	y1
2	y2
4	y3

Each potential match



Understanding joins



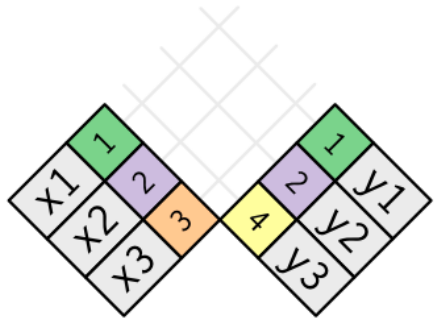
x

1	x1
2	x2
3	x3

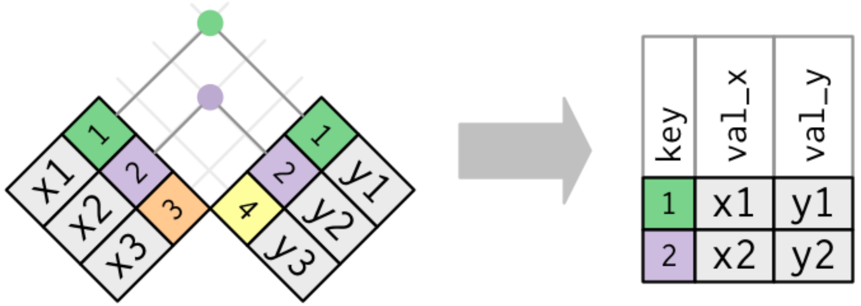
y

1	y1
2	y2
4	y3

Each potential match



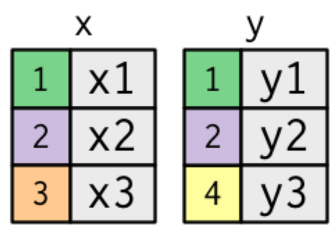
Number of actual matches



key	val_x	val_y
1	x1	y1
2	x2	y2

Inner join: Unmatched rows are not included in the output

Understanding joins



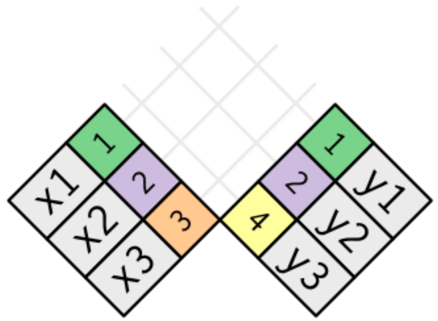
x

1	x1
2	x2
3	x3

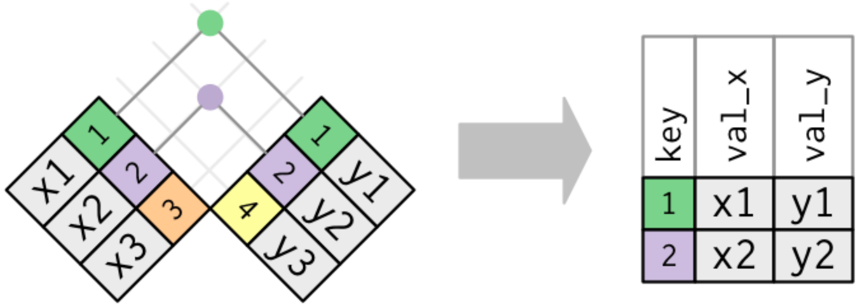
y

1	y1
2	y2
4	y3

Each potential match



Number of actual matches



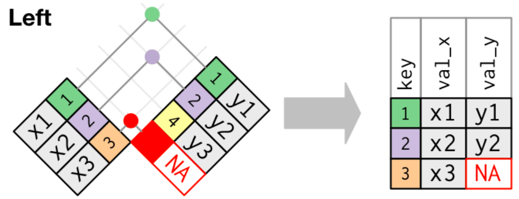
key	val_x	val_y
1	x1	y1
2	x2	y2

Inner join: Unmatched rows are not included in the output

Could result in loss of observations – use with care

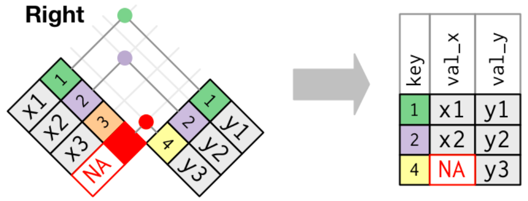
Outer joins

Keeps all observations in x



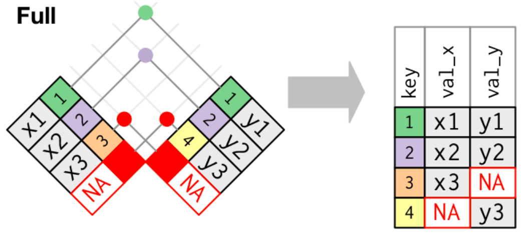
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Keeps all observations in y



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

Keeps all observations in x and y

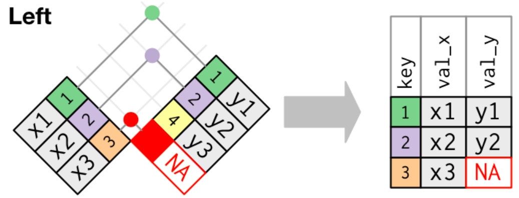


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Outer joins

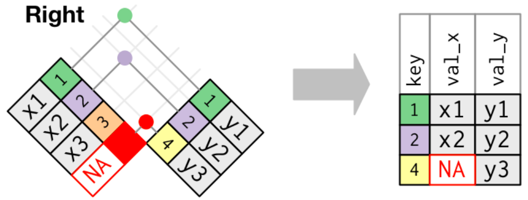
Left joins should be our default

Keeps all observations in x



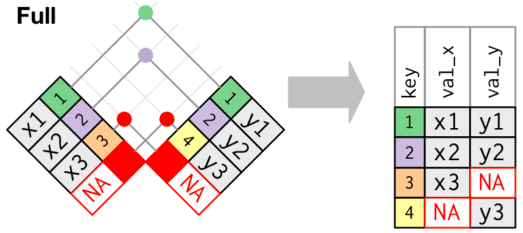
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Keeps all observations in y



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

Keeps all observations in x and y



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

Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::full_join(a, b, by = "x1")

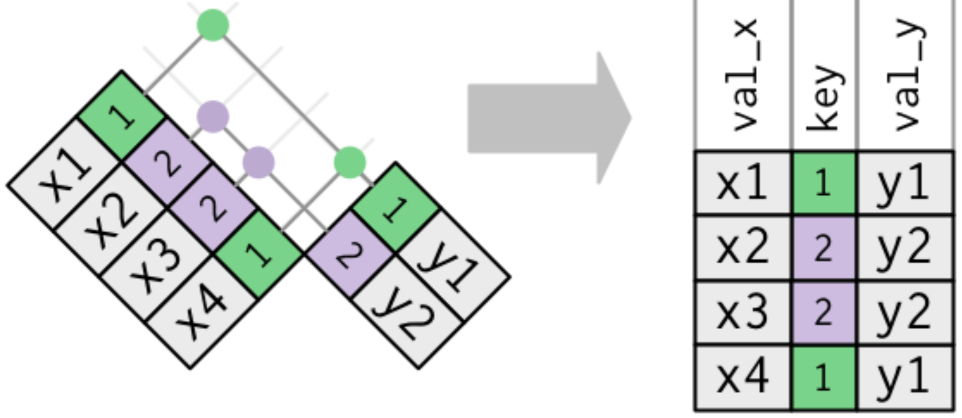
Join data. Retain all values, all rows.

Relations

- Typically one-to-many
  - Each flight has one plane, but each plane has many flights
- Can also be many-to-many
  - Each airline flies to many airports; each airport hosts many airlines
- Can also be one-to-one

Duplicate keys

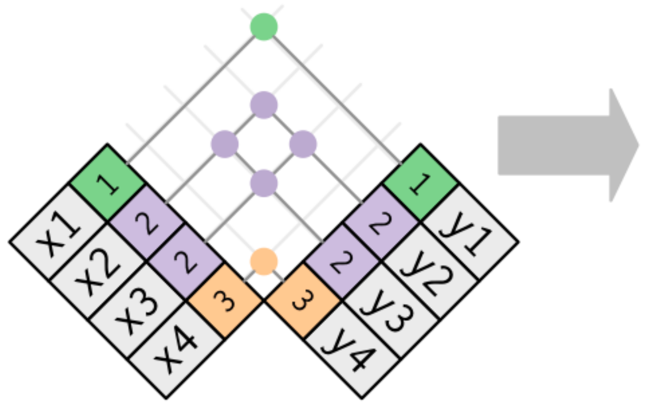
One table has duplicate keys (typically a one-to-many relationship)
e.g. “dest” in the flights tibble



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

Duplicate keys

Both tables have duplicate keys (typically an error)



key	val1_x	val1_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Exercise

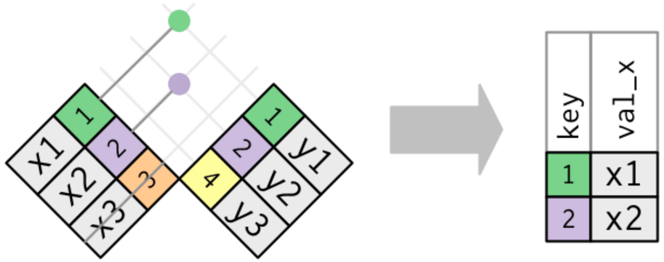
- Add the location of the origin *and* destination (i.e. the lat and lon) to flights. You may want to use the suffix parameter to disambiguate variable names in your output

- OPTIONAL QUESTION:
  Is there a relationship between the age of a plane and its delays? [You will have to work with the full flights data to access delay data]

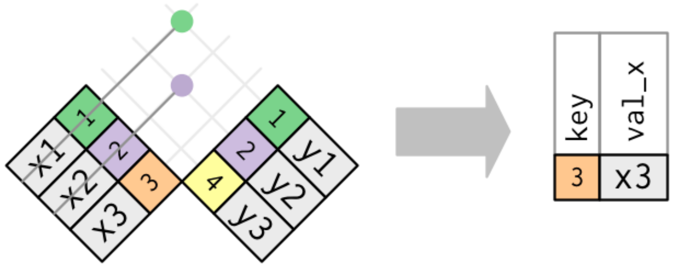
Filtering

- Filters the observations, does not add variables
    - semi_join(x, y) **keeps** all observations in x that have a match in y.
    - anti_join(x, y) **drops** all observations in x that have a match in y.

Semi-join



Anti-join



Exercises

- Filter flights to only show flights with planes that have flown at least 100 flights (hint: you will need to first use summarize())
- Combine fueleconomy::vehicles and fueleconomy::common to find only the records for the most common models

Join problems – how to troubleshoot

- Start by identifying the variables that form the primary key in each table based on your understanding of the data
- Check that none of the variables in the primary key are missing. If a value is missing then it can't identify an observation!
- Check that your foreign keys match primary keys in another table. The best way to do this is with an anti_join()