

ANT 6973: DATA VISUALIZATION AND EXPLORATION

# DATA MANIPULATION, PART 3: TWO-TABLE OPERATIONS



# TWO TABLE VERBS

## Vector Functions

TO USE WITH MUTATE ()

**mutate()** and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

**vectorized function**

### OFFSETS

**dplyr::lag()** - Offset elements by 1  
**dplyr::lead()** - Offset elements by -1

### CUMULATIVE AGGREGATES

**dplyr::cumall()** - Cumulative all()  
**dplyr::cumany()** - Cumulative any()  
**dplyr::cummax()** - Cumulative max()  
**dplyr::cummean()** - Cumulative mean()  
**dplyr::cummin()** - Cumulative min()  
**dplyr::cumprod()** - Cumulative prod()  
**dplyr::cumsum()** - Cumulative sum()

### RANKINGS

**dplyr::cume\_dist()** - Proportion of all values <= value  
**dplyr::dense\_rank()** - rank w ties = min, no gaps  
**dplyr::min\_rank()** - rank with ties = min  
**dplyr::ntile()** - bins into n bins  
**dplyr::percent\_rank()** - min\_rank scaled to [0,1]  
**dplyr::row\_number()** - rank with ties = "first"

### MATH

**+**, **-**, **\***, **/**, **^**, **%/%**, **%%** - arithmetic ops  
**log()**, **log2()**, **log10()** - logs  
**<**, **<=**, **>**, **>=**, **!=** - logical comparisons  
**dplyr::between()** - x >= left & x <= right  
**dplyr::near()** - safe == for floating point numbers

### MISC

**dplyr::case\_when()** - multi-case if\_else()  
**iris %>% mutate(Species = case\_when(**  
  **Species == "versicolor" ~ "versj",**  
  **Species == "virginica" ~ "virgi",**  
  **TRUE ~ Species))**  
**dplyr::coalesce()** - first non-NA values by element across a set of vectors  
**dplyr::if\_else()** - element-wise if() + else()  
**dplyr::na\_if()** - replace specific values with NA  
**dplyr::pmax()** - element-wise max()  
**dplyr::pmin()** - element-wise min()  
**dplyr::recode()** - Vectorized switch()  
**dplyr::recode\_factor()** - Vectorized switch() for factors



RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

## Summary Functions

TO USE WITH SUMMARISE ()

**summarise()** applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

**summary function**

### COUNTS

**dplyr::n()** - number of values/rows  
**dplyr::n\_distinct()** - # of uniques  
**sum(is.na())** - # of non-NA's

### LOCATION

**mean()** - mean, also **mean(is.na())**  
**median()** - median

### LOGICALS

**mean()** - Proportion of TRUE's  
**sum()** - # of TRUE's

### POSITION/ORDER

**dplyr::first()** - first value  
**dplyr::last()** - last value  
**dplyr::nth()** - value in nth location of vector

### RANK

**quantile()** - nth quantile  
**min()** - minimum value  
**max()** - maximum value

### SPREAD

**IQR()** - Inter-Quartile Range  
**mad()** - median absolute deviation  
**sd()** - standard deviation  
**var()** - variance

## Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

**rownames\_to\_column()**  
Move row names into col.  
**col <- rownames\_to\_column(iris, var = "C")**

**column\_to\_rownames()**  
Move col in row names.  
**column\_to\_rownames(a, var = "C")**

Also has **rownames()**, **remove\_rownames()**

## Combine Tables

COMBINE VARIABLES

Use **bind\_cols()** to paste tables beside each other as they are.

**bind\_cols(...)** Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join matching values from y to x.

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join matching values from x to y.

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join data. Retain only rows with matches.

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join data. Retain all values, all rows.

**Use by = c("col1", "col2", ...) to specify one or more common columns to match on.**  
**left\_join(x, y, by = "A")**

**Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.**  
**left\_join(x, y, by = c("C" = "D"))**

**Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.**  
**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

COMBINE CASES

Use **bind\_rows()** to paste tables below each other as they are.

**bind\_rows(..., .id = NULL)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

**intersect(x, y, ...)** Rows that appear in both x and y.

**setdiff(x, y, ...)** Rows that appear in x but not y.

**union(x, y, ...)** Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

**Use setequal() to test whether two data sets contain the exact same rows (in any order).**

**Use by = c("col1", "col2", ...) to specify one or more common columns to match on.**  
**left\_join(x, y, by = "A")**

**Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.**  
**left\_join(x, y, by = c("C" = "D"))**

**Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.**  
**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

## Combine Tables

COMBINE VARIABLES

Use **bind\_cols()** to paste tables beside each other as they are.

**bind\_cols(...)** Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join matching values from y to x.

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join matching values from x to y.

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join data. Retain only rows with matches.

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)**  
Join data. Retain all values, all rows.

**Use by = c("col1", "col2", ...) to specify one or more common columns to match on.**  
**left\_join(x, y, by = "A")**

**Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.**  
**left\_join(x, y, by = c("C" = "D"))**

**Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.**  
**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

COMBINE CASES

Use **bind\_rows()** to paste tables below each other as they are.

**bind\_rows(..., .id = NULL)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

**intersect(x, y, ...)** Rows that appear in both x and y.

**setdiff(x, y, ...)** Rows that appear in x but not y.

**union(x, y, ...)** Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

**Use setequal() to test whether two data sets contain the exact same rows (in any order).**

EXTRACT ROWS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.

Use a "Filtering Join" to filter one table against the rows of another.

**semi\_join(x, y, by = NULL, ...)** Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

**anti\_join(x, y, by = NULL, ...)** Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.



# TYPES OF TWO-TABLE OPERATIONS

- Combining variables and cases
- Set operations
- Joins

# COMBINING VARIABLES AND CASES

# COMBINING VARIABLES (COLUMNS)

# COMBINING VARIABLES (COLUMNS)

Use `bind_cols()` to paste tables as they are as a single table side-by-side.

*X*

v1	v2	v3
a	t	1
b	u	2
c	v	3

*y*

v4	v5
d	4
e	5
f	6

`bind_cols(x, y)`

v1	v2	v3	v4	v5
a	t	1	d	4
b	u	2	e	5
c	v	3	f	6

# COMBINING VARIABLES (COLUMNS)

*x*

v1	v2	v3
a	t	1
b	u	2
c	v	3

Use `bind_cols()` to paste tables as they are as a single table side-by-side.

Be sure that rows align! Tables must have same number of rows.

*y*

v4	v5
d	4
e	5

`bind_cols(x, y)`

v1	v2	v3	v4	v5
a	t	1		
b	u	2	?	
c	v	3		

# COMBINING VARIABLES (COLUMNS)

*X*

v1	v2	v3
a	t	1
b	u	2
c	v	3

*y*

v4	v5
d	4
e	5

Use `bind_cols()` to paste tables as they are as a single table side-by-side.

Be sure that rows align! Tables must have same number of rows.

```
bind_cols(x, y)
```

Error



COMBINING CASES (ROWS)

# COMBINING CASES (ROWS)

`bind_rows()` binds multiple data frames by row  
(i.e., paste one table "below" the other)

*X*

v1	v2	v3
a	t	1
b	u	2
c	v	3

*y*

v1	v2	v3
d	w	4
e	x	5
f	y	6

`bind_rows(x, y)`

v1	v2	v3
a	t	1
b	u	2
c	v	3
d	w	4
e	x	5
f	y	6

# COMBINING CASES (ROWS)

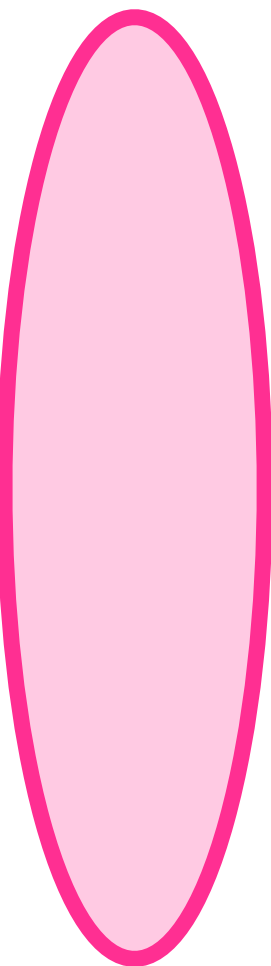
When row-binding, columns are matched by name, and any missing columns will be filled with NA.

**x**

v1	v2	v3
a	t	1
b	u	2
c	v	3

**y**

v1	v2
d	w
e	x
f	y



`bind_rows(x, y)`

v1	v2	v3
a	t	1
b	u	2
c	v	3
d	w	NA
e	x	NA
f	y	NA

# COMBINING CASES (ROWS)

When row-binding, columns are matched by name, and any missing columns will be filled with NA.

**X**

v1	v2	v3
a	t	1
b	u	2
c	v	3

**y**

v4	v5
d	w
e	x
f	y

`bind_rows(x, y)`

v1	v2	v3	v4	v5
a	t	1	NA	NA
b	u	2	NA	NA
c	v	3	NA	NA
NA	NA	NA	d	w
NA	NA	NA	e	x
NA	NA	NA	f	y

Remember messy\_ktc\_data.xlsx?



messy_ktc_data																												
HomeInsertPage LayoutFormulasDataReviewView																												
NormalPage Break PreviewPage LayoutCustom ViewsRulerFormula BarZoom 200%GridlinesHeadingsZoom to 100%Freeze PanesFreeze Top RowFreeze First ColumnSplitViewRecord Macro																												
X287/9/2012																												
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Z	AA	AB	
17	A	SW2	2	1	2	1	1.5	HVV	28-Jul-08			Mollusc data																
18	A	SW2	2	1	2	1	1.7	HVV	28-Jul-08			Trench	Unit	Context	Taxon	Element	Mass (g)	Count	Max L/W m	Burnt/Not Burr	Cutmark/N	Recorder	NOTES	Date				
19	A	SW2	2	1	2	1	6.3	HVV	28-Jul-08			A	SW2		2 Unknown	Bivalve	0.9	1	NA	N	N	HVV		9-Jul-12				
20	A	SW2	2	1	2	1	3.5	HVV	28-Jul-08			Trench	Unit	Context	Taxon	Element	Mass (g)	Count	Max L/W m	Burnt/Not Burr	Cutmark/N	Recorder	NOTES	Date				
21	A	SW2	2	1	2	1	1.4	HVV	28-Jul-08			A	SW4		5 F. Viviparid	Gastropod	1.3	1	NA	N	N	HVV		9-Jul-12				
22	A	SW2	2	1	2	1	3	HVV	28-Jul-08			A	SW4		5 F. Arcidae	Bivalve	0.8	1	NA	N	N	HVV		9-Jul-12				
23	A	SW2	2	1	2	1	2.3	HVV	28-Jul-08			A	SW4		5 F. Muricida	Gastropod	1.5	1	NA	N	N	HVV		9-Jul-12				
24	A	SW2	2	1	2	1	4.6	HVV	28-Jul-08			A	SW4		5 F. Potamidi	Gastropod	16.7	8	NA	N	N	HVV		9-Jul-12				
25	A	SW2	2	1	2	1	3.1	HVV	28-Jul-08			A	SW4		5 Unknown	Gastropod	3.2	8	NA	N	N	HVV		9-Jul-12				
26	A	SW2	2	1	2	1	1.8	HVV	28-Jul-08			A	SW4		5 Unknown	Bivalve	1.4	12	NA	N	N	HVV		9-Jul-12				
27	A	SW2	2	1	2	1	1.2	HVV	28-Jul-08			A	SW4		5 Unknown	Bivalve	3.7	4	NA	N	N	HVV		9-Jul-12				
28	A	SW2	2	1	2	1	3.2	HVV	28-Jul-08			A	SW4		5 F. Cyclophc	Gastropod	1	2	NA	N	N	HVV		9-Jul-12				
29	A	SW2	2	1	2	1	1.1	HVV	28-Jul-08			Trench	Unit	Context	Taxon	Element	Mass (g)	Count	Max L/W m	Burnt/Not Burr	Cutmark/N	Recorder	NOTES	Date				
30	A	SW2	2	1	2	1	1.3	HVV	28-Jul-08			A	SW5		9 F. Cyclophc	Gastropod	1.3	1	21.16	N	N	CC	F-100-0168	10-Jul-12				
31	A	SW2	2	1	2	1	0.8	HVV	28-Jul-08			A	SW5		9 F. Neritidae	Gastropod	1.7	1	25.06	N	N	CC	F-100-0161	10-Jul-12				
32	A	SW2	2	1	2	1	1.1	HVV	28-Jul-08			A	SW5		9 F. Cyclophc	Gastropod	2.7	1	28.24	N	N	CC	F-100-0174	10-Jul-12				
33	A	SW2	2	1	2	1	1	HVV	28-Jul-08			A	SW5		9 F. Cyclophc	Gastropod	2.5	1	27.34	N	N	CC	F-100-0169	10-Jul-12				
34	A	SW2	2	1	2	1	1.5	HVV	28-Jul-08			A	SW5		9 F. Viviparid	Gastropod	4.5	2	NA	N	N	CC		10-Jul-12				
35	A	SW2	2	1	2	1	1	HVV	28-Jul-08			A	SW5		9 F. Camaeni	Gastropod	3.2	1	NA	N	N	CC		10-Jul-12				
36	A	SW2	2	1	2	1	0.9	HVV	28-Jul-08			A	SW5		9 F. Cyclophc	Gastropod	2.5	3	NA	N	N	CC		10-Jul-12				
37	A	SW2	2	1	2	1	4.4	HVV	28-Jul-08			A	SW5		9 F. Muricida	Gastropod	9.6	4	NA	N	N	CC		10-Jul-12				
38	A	SW2	2	1	2	1	1.9	HVV	28-Jul-08			A	SW5		9 F. Potamidi	Gastropod	40.8	3	NA	N	N	CC		10-Jul-12				
39	A	SW2	2	2	3	1	0.9	CC	28-Jul-08			A	SW5		9 F. Arcidae	Bivalve	36.7	3	NA	N	N	CC		10-Jul-12				
40	A	SW3	3	3	5	7	12.5	HVV	28-Jul-08			A	SW5		9 F. Amblemi	Bivalve	9	1	NA	N	N	CC		10-Jul-12				
41	A	SW3	3	3	5	3	12.1	HVV	28-Jul-08			A	SW5		9 F. Potamidi	Gastropod	224.6	134	NA	N	N	CC		10-Jul-12				
42	A	SW3	3	3	5	5	10.8	HVV	28-Jul-08			A	SW5		9 Unknown	Gastropod	55.7	75	NA	N	N	CC		10-Jul-12				
43	A	SW3	3	3	5	6	19.9	HVV	28-Jul-08			A	SW5		9 Unknown	Bivalve	151	150	NA	N	N	CC		10-Jul-12				
44	A	SW3	3	3	5	11	35.4	HVV	28-Jul-08			Trench	Unit	Context	Taxon	Element	Mass (g)	Count	Max L/W m	Burnt/Not Burr	Cutmark/N	Recorder	NOTES	Date				
45	A	SW3	3	3	5	21	56.7	HVV	28-Jul-08			A	SW6		12 F. Potamidi	Gastropod	89.9	52	NA	N	N	CC		11-Jul-12				
46	A	SW4	4	3	5	46	65.7	CC	28-Jul-08			A	SW6		12 F. Potamidi	Gastropod	15.4	2	NA	N	N	CC		11-Jul-12				
47	A	SW4	4	3	5	1	2.4	CC	28-Jul-08			A	SW6		12 F. Cyclophc	Gastropod	1.8	2	NA	N	N	CC		11-Jul-12				
48	A	SW4	4	3	5	1	1.8	CC	28-Jul-08			A	SW6		12 Unknown	Gastropod	32.4	54	NA	N	N	CC		11-Jul-12				
49	A	SW4	4	3	5	1	9.2	CC	28-Jul-08			A	SW6		12 Unknown	Bivalve	117.6	115	NA	N	N	HVV		11-Jul-12				
50	A	SW4	4	3	5	1	5.8	CC	28-Jul-08			A	SW6		12 F. Cyclophc	Gastropod	1.4	1	25.19	N	N	CC	F-100-0187	11-Jul-12				
51	A	SW4	4	3	5	1	4.5	CC	28-Jul-08			A	SW6		12 F. Cyclophc	Gastropod	0.4	1	25.93	N	N	CC	F-100-0184	11-Jul-12				
52	A	SW4	4	3	5	1	5.9	CC	28-Jul-08			A	SW6		12 GB	Gastropod	3.2	1	NA	N	N	CC	F-100-0179	11-Jul-12				
53	A	SW4	4	3	7	3	10.2	HVV	28-Jul-08			A	SW6		12 F. Amblemi	Bivalve	29.6	9	NA	N	N	HVV		11-Jul-12				
54	A	SW4	4	3	7	17	17	HVV	28-Jul-08			A	SW6		12 BiA	Bivalve	14	1	NA	N	N	HVV	F-100-0190	11-Jul-12				
55	A	SW4	4	3	7	15	33.1	HVV	28-Jul-08			Trench	Unit	Context	Taxon	Element	Mass (g)	Count	Max L/W m	Burnt/Not Burr	Cutmark/N	Recorder	NOTES	Date				
56	A	SW4	4	3	7	10	11.1	HVV	28-Jul-08			A	SW6		13 F. Potamidi	Gastropod	25.8	19	NA	N	N	CC		11-Jul-12				
57	A	SW4	4	3	7	16	16.7	HVV	28-Jul-08			A	SW6		13 GC	Gastropod	0.3	1	NA	N	N	CC	F-100-0194	11-Jul-12				
58	A	SW4	4	3	7	1	7.6	HVV	28-Jul-08			A	SW6		13 F. Achatinic	Gastropod	1.3	1	NA	N	N	CC		11-Jul-12				
59	A	SW4	4	3	7	1	4.6	HVV	28-Jul-08			A	SW6		13 F. Potamidi	Gastropod	9.6	1	NA	N	N	CC		11-Jul-12				
60	A	SW4	4	3	7	1	4.4	HVV	28-Jul-08			A	SW6		13 F. Amblemi	Bivalve	2.8	1	NA	N	N	CC		11-Jul-12				
61	A	SW4	4	3	7	1	1.3	HVV	28-Jul-08			A	SW6		13 Unknown	Bivalve	14.2	27	NA	N	N	CC		11-Jul-12				
62	A	SW4	4	3	7	1	0.6	HVV	28-Jul-08			A	SW6		13 Unknown	Gastropod	8.4	20	NA	N	N	CC		11-Jul-12				
63	A	SW4	4	3	7	1	1.3	HVV	28-Jul-08			A	SW6		13 F. Cyclophc	Gastropod	3.8	3	NA	N	N	CC		11-Jul-12				



# Review from Data Import module

```
artifacts_path <- here::here("08_data-import/spreadsheets", "messy_ktc_data.xlsx")
```

```
molluscs_sw4 <- artifacts_path %>%  
  read_excel(range = "L20:X28",  
             col_types = c("text", "text", "text", "text", "text", "numeric", "numeric",  
                           "numeric", "text", "text", "text", "text", "text", "date")) %>%  
  clean_names()
```

Read in the "SW4" chunk

Read in the "SW6" chunk

```
molluscs_sw6 <- artifacts_path %>% ...
```

Bind them together.

```
sw4_6 <- bind_rows(molluscs_sw4, molluscs_sw6)
```

# A tibble: 18 x 13

	trench	unit	context	taxon	element	mass_g	count	max_l_w_mm	burnt_not_burnt	cutmark_no_cutmark	recorder	notes	date
	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dtm>
1	A	SW4	5	F. Viviparidae	Gastropod	1.3	1	NA	N	N	HVV	NA	2012-07-09 00:00:00
2	A	SW4	5	F. Arcidae Anadara sp.	Bivalve	0.8	1	NA	N	N	HVV	NA	2012-07-09 00:00:00
3	A	SW4	5	F. Muricidae	Gastropod	1.5	1	NA	N	N	HVV	NA	2012-07-09 00:00:00
4	A	SW4	5	F. Potamididae	Gastropod	16.7	8	NA	N	N	HVV	NA	2012-07-09 00:00:00
5	A	SW4	5	Unknown	Gastropod	3.2	8	NA	N	N	HVV	NA	2012-07-09 00:00:00
6	A	SW4	5	Unknown	Bivalve	1.4	12	NA	N	N	HVV	NA	2012-07-09 00:00:00
7	A	SW4	5	Unknown	Bivalve	3.7	4	NA	N	N	HVV	NA	2012-07-09 00:00:00
8	A	SW4	5	F. Cyclophoridae	Gastropod	1	2	NA	N	N	HVV	NA	2012-07-09 00:00:00
9	A	SW6	12	F. Potamididae	Gastropod	89.9	52	NA	N	N	CC	NA	2012-07-11 00:00:00
10	A	SW6	12	F. Potamididae Telescopium telescopium	Gastropod	15.4	2	NA	N	N	CC	NA	2012-07-11 00:00:00
11	A	SW6	12	F. Cyclophoridae	Gastropod	1.8	2	NA	N	N	CC	NA	2012-07-11 00:00:00
12	A	SW6	12	Unknown	Gastropod	32.4	54	NA	N	N	CC	NA	2012-07-11 00:00:00
13	A	SW6	12	Unknown	Bivalve	118.	115	NA	N	N	HVV	NA	2012-07-11 00:00:00
14	A	SW6	12	F. Cyclophoridae Rhiostoma housei	Gastropod	1.4	1	25.2	N	N	CC	F-100-0187 B-100-0188	2012-07-11 00:00:00
15	A	SW6	12	F. Cyclophoridae Rhiostoma housei	Gastropod	0.4	1	25.9	N	N	CC	F-100-0184 B-100-0185	2012-07-11 00:00:00
16	A	SW6	12	GB	Gastropod	3.2	1	NA	N	N	CC	F-100-0179 B-100-0182	2012-07-11 00:00:00
17	A	SW6	12	F. Amblemidae Pseudodon sp.	Bivalve	29.6	9	NA	N	N	HVV	NA	2012-07-11 00:00:00
18	A	SW6	12	BiA	Bivalve	14	1	NA	N	N	HVV	F-100-0190 B-100-0189	2012-07-11 00:00:00

# SET OPERATIONS



# SET OPERATIONS

### Vector Functions

TO USE WITH MUTATE ()

**mutate()** and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

**vectorized function**

OFFSETS

**dplyr::lag()** - Offset elements by 1  
**dplyr::lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

**dplyr::cumall()** - Cumulative all()  
**dplyr::cumany()** - Cumulative any()  
**dplyr::cummax()** - Cumulative max()  
**dplyr::cummean()** - Cumulative mean()  
**dplyr::cummin()** - Cumulative min()  
**dplyr::cumprod()** - Cumulative prod()  
**dplyr::cumsum()** - Cumulative sum()

RANKINGS

**dplyr::cume\_dist()** - Proportion of all values <=  
**dplyr::dense\_rank()** - rank w ties = min, no gaps  
**dplyr::min\_rank()** - rank with ties = min  
**dplyr::ntile()** - bins into n bins  
**dplyr::percent\_rank()** - min\_rank scaled to [0,1]  
**dplyr::row\_number()** - rank with ties = "first"

MATH

**+**, **-**, **\***, **/**, **^**, **%/%**, **%%** - arithmetic ops  
**log()**, **log2()**, **log10()** - logs  
**<**, **<=**, **>**, **>=**, **!=** - logical comparisons  
**dplyr::between()** - x >= left & x <= right  
**dplyr::near()** - safe == for floating point numbers

MISC

**dplyr::case\_when()** - multi-case if\_else()  
**iris %>% mutate(Species = case\_when(Species == "versicolor" ~ "versj", Species == "virginica" ~ "virgi", TRUE ~ Species))**  
**dplyr::coalesce()** - first non-NA values by element across a set of vectors  
**dplyr::if\_else()** - element-wise if() + else()  
**dplyr::na\_if()** - replace specific values with NA  
**dplyr::pmax()** - element-wise max()  
**dplyr::pmin()** - element-wise min()  
**dplyr::recode()** - Vectorized switch()  
**dplyr::recode\_factor()** - Vectorized switch() for factors

### Summary Functions

TO USE WITH SUMMARISE ()

**summarise()** applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

**summary function**

COUNTS

**dplyr::n()** - number of values/rows  
**dplyr::n\_distinct()** - # of uniques  
**sum(is.na())** - # of non-NA's

LOCATION

**mean()** - mean, also **mean(is.na())**  
**median()** - median

LOGICALS

**mean()** - Proportion of TRUE's  
**sum()** - # of TRUE's

POSITION/ORDER

**dplyr::first()** - first value  
**dplyr::last()** - last value  
**dplyr::nth()** - value in nth location of vector

RANK

**quantile()** - nth quantile  
**min()** - minimum value  
**max()** - maximum value

SPREAD

**IQR()** - Inter-Quartile Range  
**mad()** - median absolute deviation  
**sd()** - standard deviation  
**var()** - variance

### Combine Tables

COMBINE VARIABLES

Use **bind\_cols()** to paste tables beside each other as they are.

**bind\_cols(...)** Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join matching values from y to x.

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join matching values from x to y.

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join data. Retain only rows with matches.

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join data. Retain all values, all rows.

COMBINE CASES

Use **bind\_rows()** to paste tables below each other as they are.

**bind\_rows(..., .id = NULL)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

**intersect(x, y, ...)** Rows that appear in both x and y.

**setdiff(x, y, ...)** Rows that appear in x but not y.

**union(x, y, ...)** Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.  
**left\_join(x, y, by = "A")**

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.  
**left\_join(x, y, by = c("C" = "D"))**

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.  
**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

Use a "Filtering Join" to filter one table against the rows of another.

**semi\_join(x, y, by = NULL, ...)** Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

**anti\_join(x, y, by = NULL, ...)** Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.


### Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

**rownames\_to\_column()** Move row names into col.  
**col <- rownames\_to\_column(iris, var = "C")**

**column\_to\_rownames()** Move col in row names.  
**column\_to\_rownames(a, var = "C")**


Also has **rownames()**, **remove\_rownames()**



RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

## Combine Tables



### COMBINE VARIABLES

Use **bind\_cols()** to paste tables beside each other as they are.

**bind\_cols(...)** Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join matching values from y to x.

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join matching values from x to y.

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join data. Retain only rows with matches.

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)** Join data. Retain all values, all rows.

### COMBINE CASES

Use **bind\_rows()** to paste tables below each other as they are.

**bind\_rows(..., .id = NULL)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

**intersect(x, y, ...)** Rows that appear in both x and y.

**setdiff(x, y, ...)** Rows that appear in x but not y.

**union(x, y, ...)** Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.  
**left\_join(x, y, by = "A")**

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.  
**left\_join(x, y, by = c("C" = "D"))**

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.  
**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

Use a "Filtering Join" to filter one table against the rows of another.

**semi\_join(x, y, by = NULL, ...)** Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

**anti\_join(x, y, by = NULL, ...)** Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

Set operations



# SET OPERATIONS

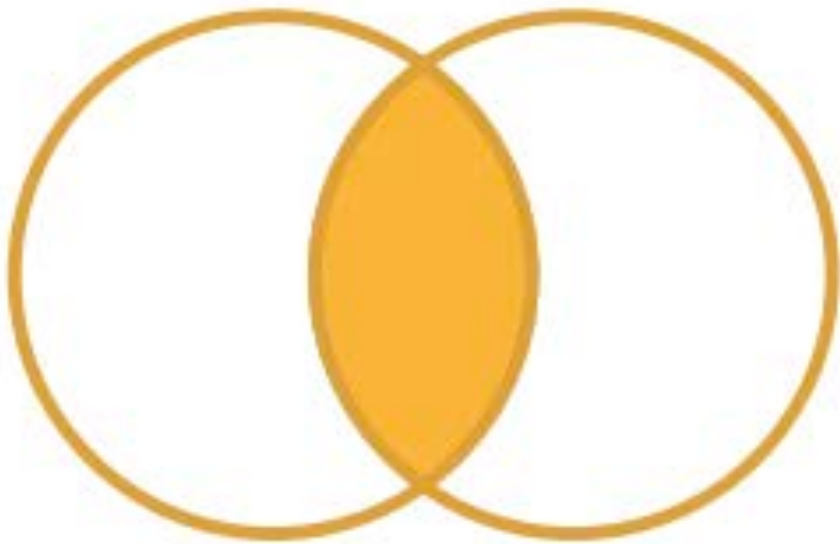
**intersect(x, y, ...)**  
Rows that appear in both x and y.

x

v1	v2	v3
a	t	1
b	u	2
c	v	3

y

v1	v2	v3
c	v	3
d	w	4



v1	v2	v3
c	v	3

# SET OPERATIONS

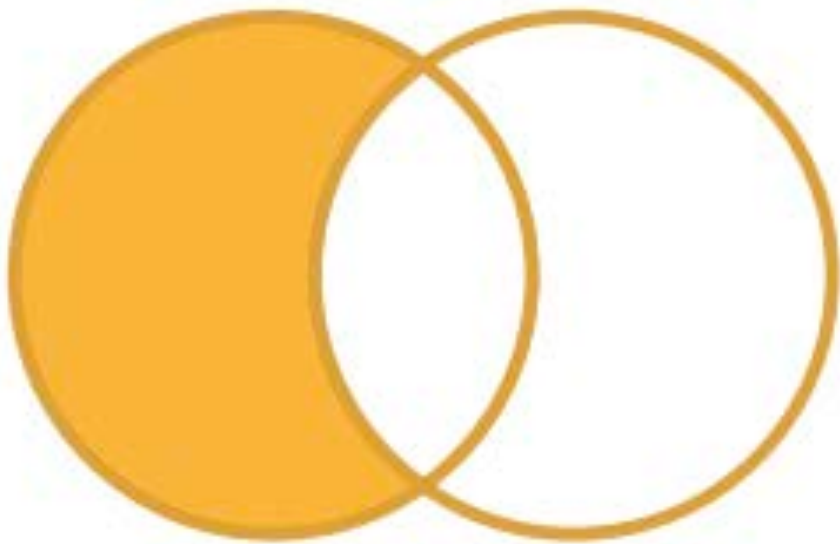
*X*

v1	v2	v3
a	t	1
b	u	2
c	v	3

*y*

v1	v2	v3
c	v	3
d	w	4

**setdiff(x, y, ...)**  
Rows that appear in x but not y.



v1	v2	v3
a	t	1
b	u	2

# SET OPERATIONS

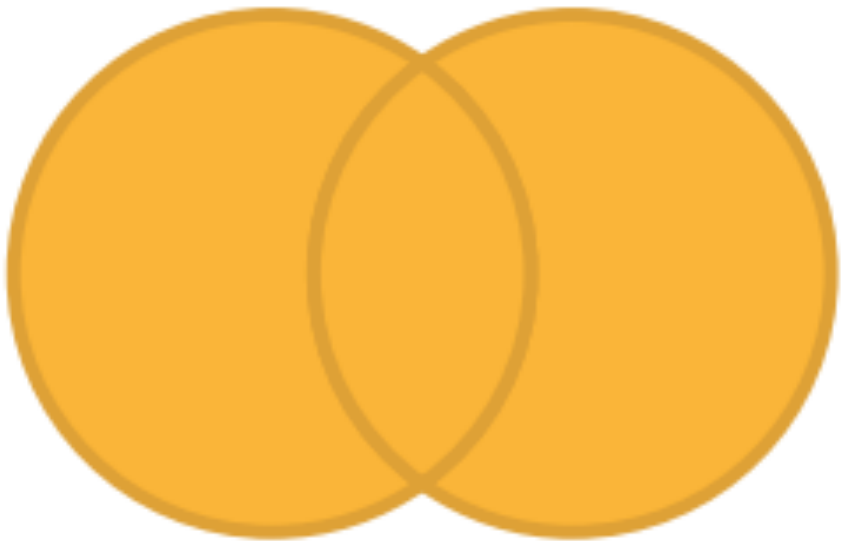
*X*

v1	v2	v3
a	t	1
b	u	2
c	v	3

*y*

v1	v2	v3
c	v	3
d	w	4

**union(x, y, ...)**  
Rows that appear in x or y.  
(Duplicates removed).



v1	v2	v3
a	t	1
b	u	2
c	v	3
d	w	4

JOINS

# nycflights13



Data about every flight that departed La Guardia, JFK, or Newark airports in 2013

```
# install.packages("nycflights13")  
library("nycflights13")
```



# flights

336,776 x 19

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5	15	2013-01-01 05:00:00
2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5	29	2013-01-01 05:00:00
2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5	40	2013-01-01 05:00:00
2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5	45	2013-01-01 05:00:00
2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6	0	2013-01-01 06:00:00
2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5	58	2013-01-01 05:00:00
2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	849	851	-2	B6	49	N793JB	JFK	PBI	149	1028	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	853	856	-3	B6	71	N657JB	JFK	TPA	158	1005	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	924	917	7	UA	194	N29129	JFK	LAX	345	2475	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	923	937	-14	UA	1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00
2013	1	1	559	600	-1	941	910	31	AA	707	N3DUAA	LGA	DFW	257	1389	6	0	2013-01-01 06:00:00
2013	1	1	559	559	0	702	706	-4	B6	1806	N708JB	JFK	BOS	44	187	5	59	2013-01-01 05:00:00
2013	1	1	559	600	-1	854	902	-8	UA	1187	N76515	EWR	LAS	337	2227	6	0	2013-01-01 06:00:00
2013	1	1	600	600	0	851	858	-7	B6	371	N595JB	LGA	FLL	152	1076	6	0	2013-01-01 06:00:00
2013	1	1	600	600	0	837	825	12	MQ	4650	N542MQ	LGA	ATL	134	762	6	0	2013-01-01 06:00:00
2013	1	1	601	600	1	844	850	-6	B6	343	N644JB	EWR	PBI	147	1023	6	0	2013-01-01 06:00:00
2013	1	1	602	610	-8	812	820	-8	DL	1919	N971DL	LGA	MSP	170	1020	6	10	2013-01-01 06:00:00
2013	1	1	602	605	-3	821	805	16	MQ	4401	N730MQ	LGA	DTW	105	502	6	5	2013-01-01 06:00:00



## Details

⚠ The class of service you searched may not be available on one or more flights

**BNA - ORD**

Flight 1 of 2

**ORD - YVR**

Flight 2 of 2

Nashville, TN to Chicago, IL

Thursday, July 26, 2018

4:10 PM → 6:03 PM

AA 3246 ■ CR7-Canadair RJ 700 📶  
Operated by SkyWest Airlines As American  
Eagle

### Travel info

Travel time: 1h 53m  
Connection time: 2h 33m

### Main Cabin

Meals: Beverage service  
Booking code: V  
Class: Economy

### Business

Meals: Beverage service  
Booking code: I  
Class: First

\* This is based on information from the month of May 2018

\*\* The on-time arrival percentage for the selected flight is based on arrival within 14 minutes after

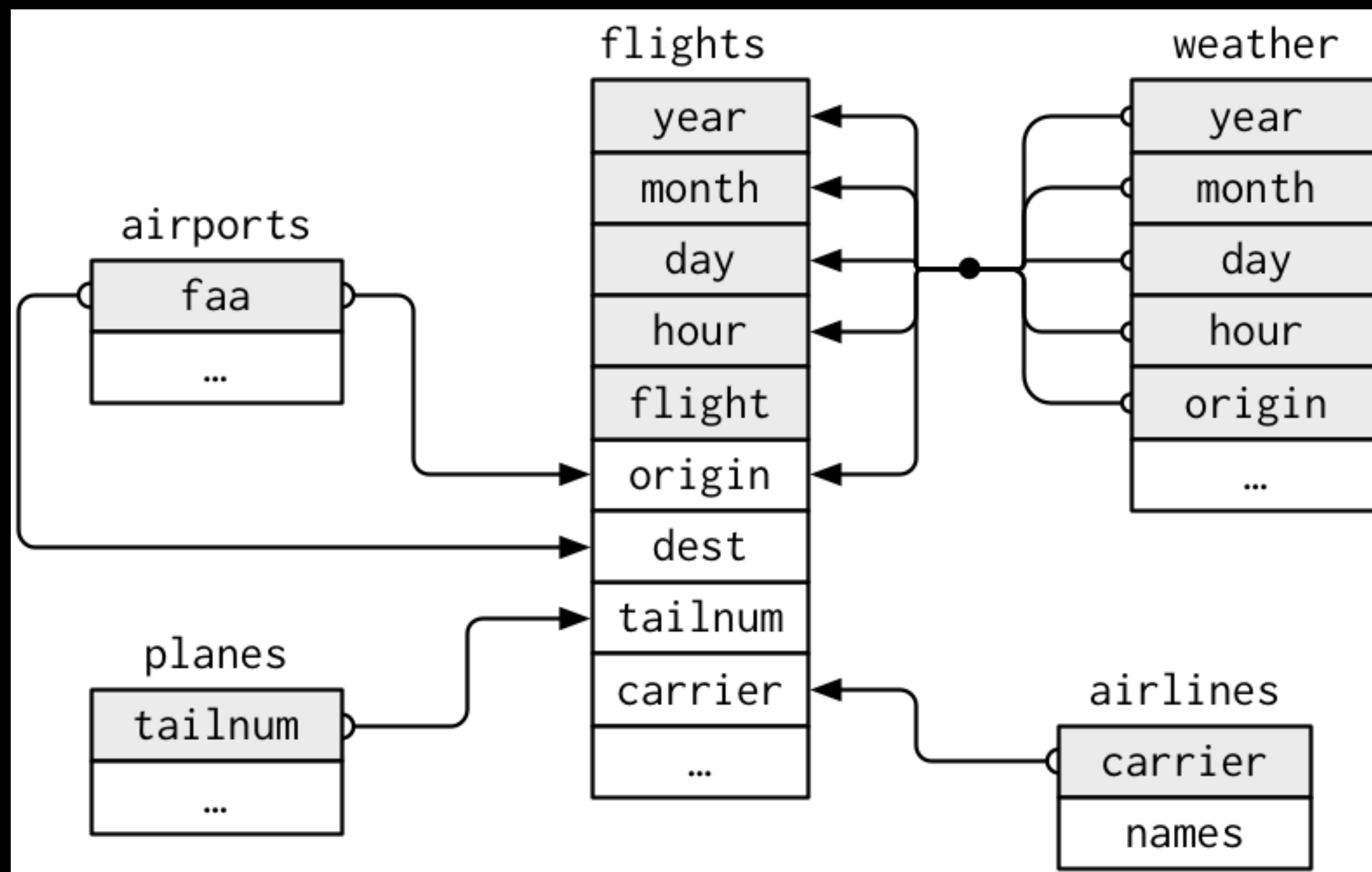
## Performance\*

On time: 52%\*\*

Late: 43%

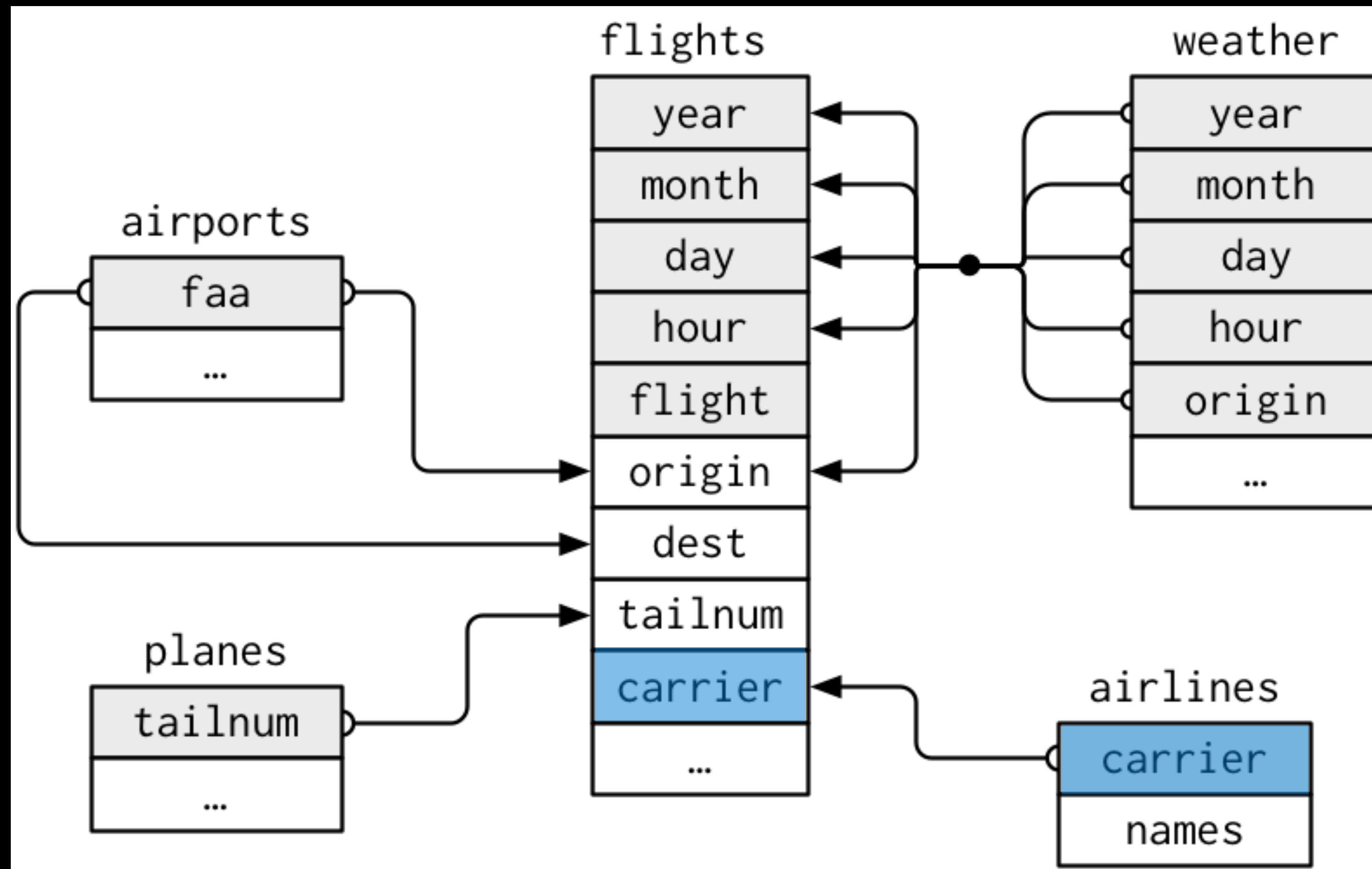
\*\* The on-time arrival percentage for the selected flight is based on arrival within 14 minutes after the scheduled arrival as reported monthly to the U.S. Department of Transportation.

# nycflights13



# nycflights13

- What airline had the longest delays?





```
flights %>%  
  select(carrier) %>%  
  View()
```

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6
8	EV
9	B6
10	AA

One row per  
flight (many  
duplicate carriers)

One row per  
carrier  
(no duplicates)

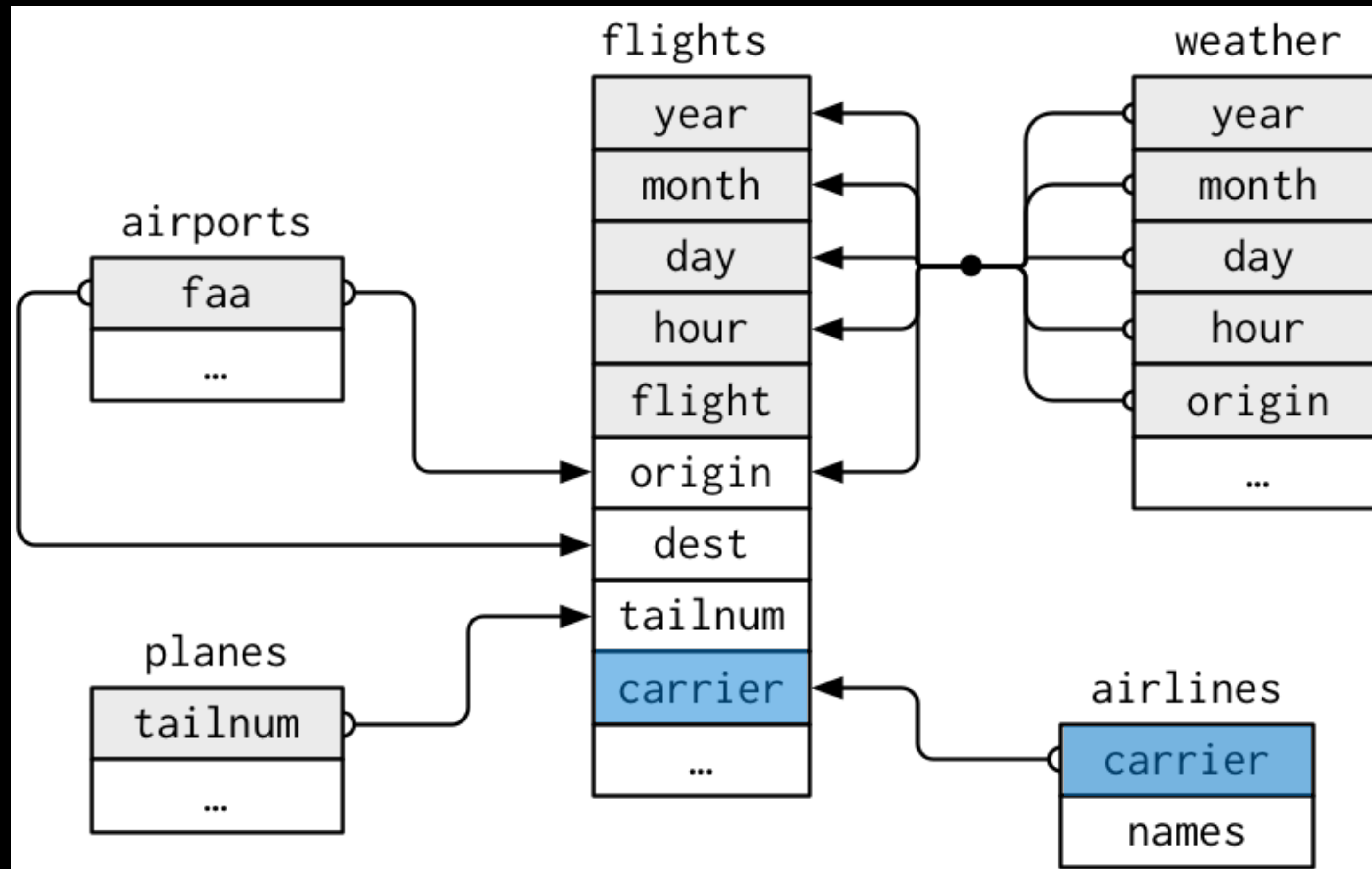
```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.
8	FL	AirTran Airways Corporation
9	HA	Hawaiian Airlines Inc.
10	MO	Envoy Air



# nycflights13

- What airline had the longest delays?



# MUTATING JOINS

ADD NEW VARIABLES TO ONE TABLE FROM MATCHING ROWS IN ANOTHER

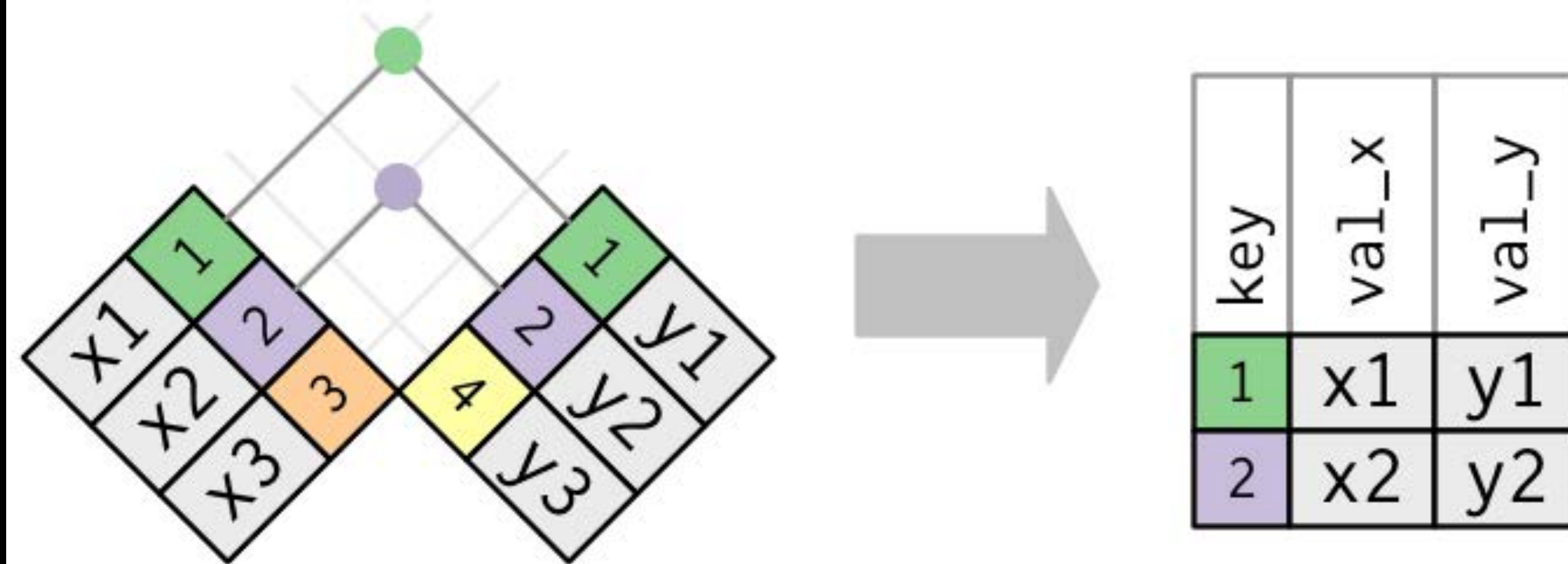
# MUTATING VS. FILTERING JOINS

- **Mutating joins** use information from one data set to add variables to another data set (like `mutate()`).
- **Filtering joins** use information from one data set to extract cases from another data set (like `filter()`).

## MUTATING JOINS

# INNER JOIN

- Matches pairs of observations whenever their keys are equal.
- Output of an inner join is a new data frame that contains the key, the x values, and the y values.
- *Unmatched rows are not included in the result!*
- This can be useful, but it can also be dangerous (you'll lose observations if you're not careful).



# COMMON SYNTAX

Each join function returns a data frame / tibble.

```
inner_join(x, y, by = ...)
```

Join  
function

Two data  
frames to  
join

Names of columns  
to join on

# PRACTICE DATA

```
band <- tibble(name = ...,  
               band = ...))
```

band

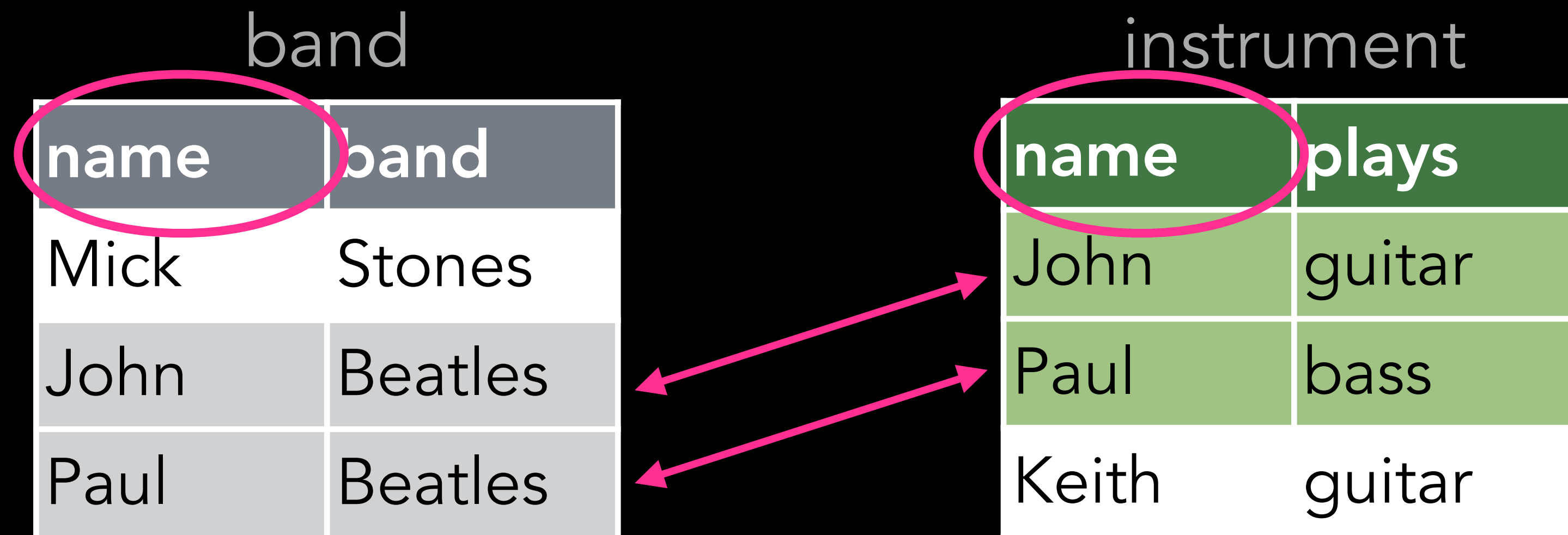
name	band
Mick	Stones
John	Beatles
Paul	Beatles

```
instrument <- tibble(name = ...,  
                     plays = ...))
```

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

# PRACTICE DATA





# INNER JOIN

```
band %>% inner_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

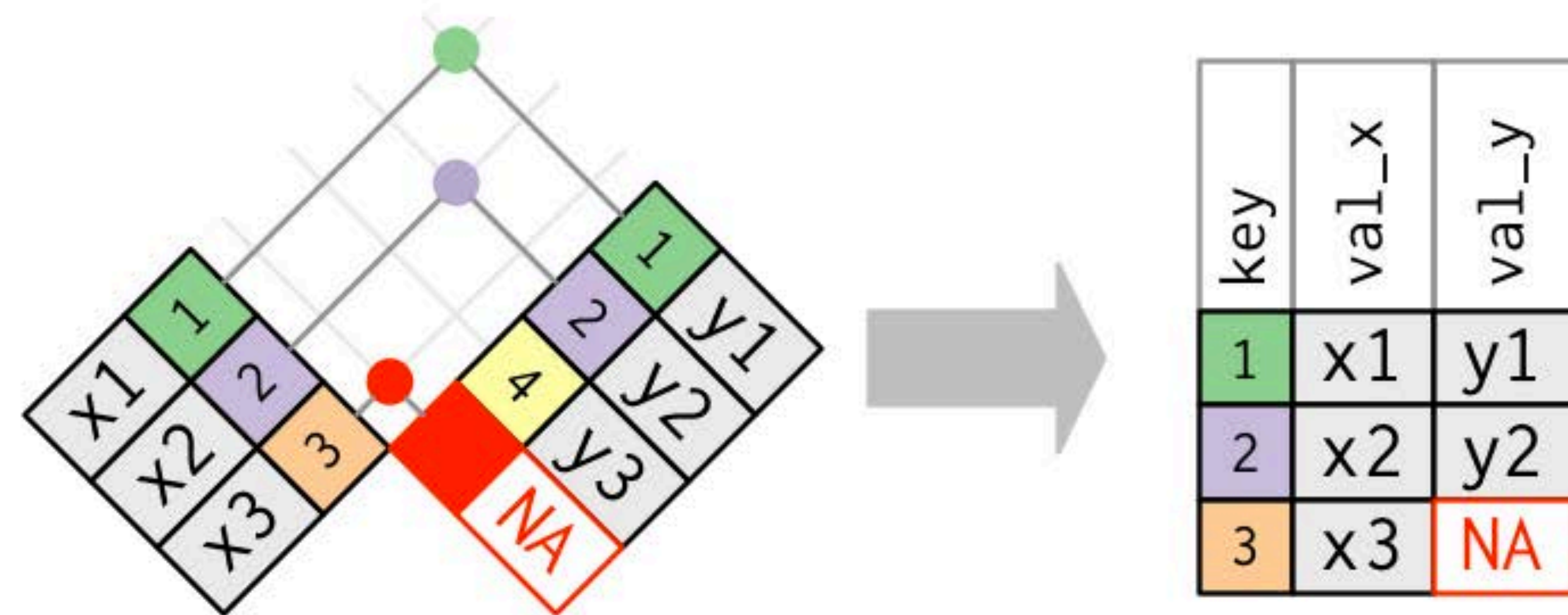
name	plays
John	guitar
Paul	bass
Keith	guitar

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

## MUTATING JOINS

# LEFT JOIN

- Preserves the original observations from first ("left") table, even when there isn't a match.
- Use whenever you look up additional data from another table.
- Should be your default join.



# LEFT JOIN

```
band %>% left_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

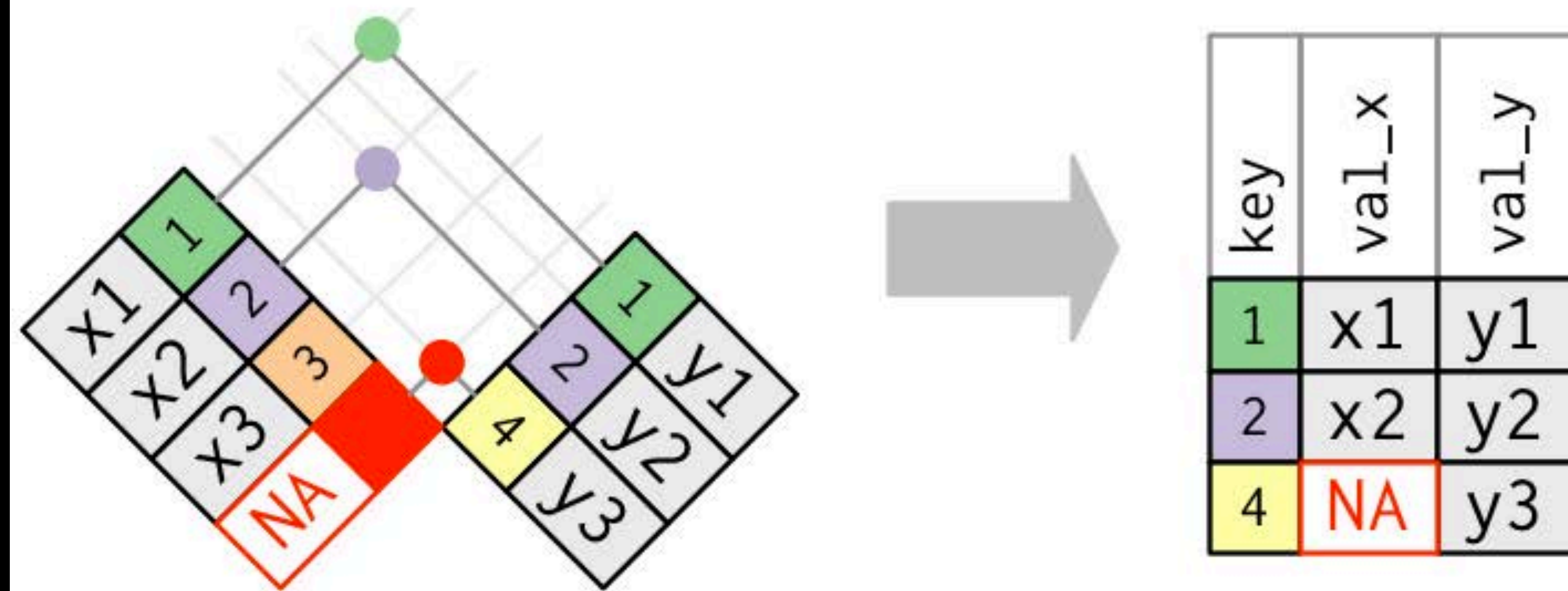
name	plays
John	guitar
Paul	bass
Keith	guitar

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

## MUTATING JOINS

# RIGHT JOIN

- Like a left-join, but preserves observations in the second ("right") table.
- Usually no reason to use: just switch the order of the tables and make it a left join.
- Only situation where I might use a right join is at the end of a chain of piped operations.



# RIGHT JOIN

```
band %>% right_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

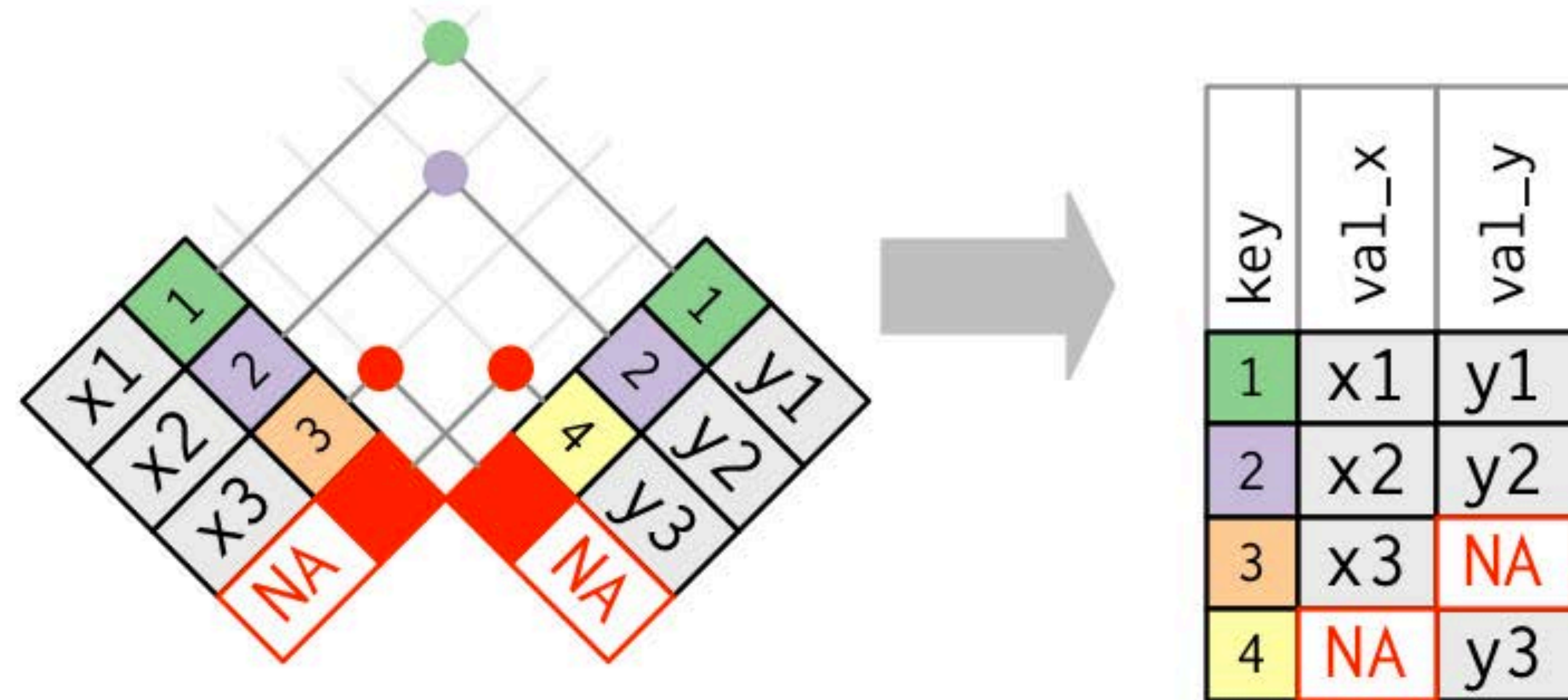
name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar



## MUTATING JOINS

# FULL JOIN

- Preserves all observations from both tables, even when there isn't a match.



# FULL JOIN

```
band %>% full_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar



# AIRLINE NAMES

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6
8	EV
9	B6
10	AA

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.
8	FL	AirTran Airways Corporation
9	HA	Hawaiian Airlines Inc.
10	MO	Envoy Air

# ACTIVITY 1

Which airlines had the largest arrival delays? Complete the code below.

```
flights %>%  
  drop_na(arr_delay) %>%  
  ----- %>%  
  group_by(-----) %>%  
  ----- %>%  
  arrange(-----)
```

1. Join airlines to flights

2. Compute and order the average arrival delays by airline. Display full names, no codes.

Note: arrival delay NAs are cancelled flights

```
flights %>%  
  drop_na(arr_delay) %>%  
  left_join(airlines, by = "carrier") %>%  
  group_by(name) %>%  
  summarise(delay = mean(arr_delay)) %>%  
  arrange(delay)
```

```
## # A tibble: 16 × 2
```

```
##           name      delay  
##           <chr>    <dbl>  
## 1   Alaska Airlines Inc. -9.9308886  
## 2 Hawaiian Airlines Inc. -6.9152047  
## 3 American Airlines Inc.  0.3642909  
## 4   Delta Air Lines Inc.  1.6443409  
## 5   Virgin America    1.7644644
```



# BACK TO THIS DATA...

```
band <- tibble(name = ...,  
               band = ...))
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

```
instrument <- tibble(name = ...,  
                    plays = ...))
```

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

# WHAT IF THE NAMES DON'T MATCH?

```
band <- tibble(name = ...),  
              band = ...))
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

```
instrument <- tibble(artist = ...),  
                    plays = ...))
```

instrument

artist	plays
John	guitar
Paul	bass
Keith	guitar

# WHAT IF THE NAMES DON'T MATCH?

Use the **by** argument to specify the different names.

```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

Column name in the  
first data set

Column name in the  
second data set



```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument2

artist	plays
John	guitar
Paul	bass
Keith	guitar

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

# AIRPORT NAMES

```
airports %>% select(1:2)
```

<b>faa</b> <chr>	<b>name</b> <chr>
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport
0G6	Williams County Airport
0G7	Finger Lakes Regional Airport

```
flights %>% select(14:15)
```

<b>dest</b> <chr>	<b>air_time</b> <dbl>
IAH	227
IAH	227
MIA	160
BQN	183
ATL	116
ORD	150
FLL	158
IAD	53

# ACTIVITY 2

- Use flights and airports to compute the average arr\_delay by destination airport (names only, not codes), and the number of flights that this is based on.
- Arrange by average delay, from worst to best.

```
# A tibble: 101 x 3
```

	name	mean_delay	n_flights
	<chr>	<dbl>	<int>
1	Columbia Metropolitan	41.8	106
2	Tulsa Intl	33.7	294
3	Will Rogers World	30.6	315
4	Jackson Hole Airport	28.1	21

```
flights %>%  
  drop_na(arr_delay) %>%  
  left_join(airports, by = c("dest" = "faa")) %>%  
  group_by(name) %>%  
  summarise(mean_delay = mean(arr_delay),  
            n_flights = n()) %>%  
  arrange(desc(mean_delay))
```

```
# A tibble: 101 x 3
```

	name	mean_delay	n_flights
	<chr>	<dbl>	<int>
1	Columbia Metropolitan	41.8	106
2	Tulsa Intl	33.7	294
3	Will Rogers World	30.6	315
4	Jackson Hole Airport	28.1	21



# ACTIVITY 3

- Can you figure out how to get the full name of **both** the origin airport (`origin`) and the destination airport (`dest`) in the `flights` table?
- Do as before, but group by both origin name and destination name.

```
# A tibble: 218 x 4
```

	origin_name	dest_name	mean_delay	n_flights
	<chr>	<chr>	<dbl>	<int>
1	Newark Liberty Intl	Columbia Metropolitan	44.6	94
2	Newark Liberty Intl	Mc Ghee Tyson	41.2	313
3	Newark Liberty Intl	Tulsa Intl	33.7	294
4	Newark Liberty Intl	Will Rogers World	30.6	315

```

flights %>%
  drop_na(arr_delay),
  left_join(select(airports, faa, origin_name = name),
            by = c("origin" = "faa")) %>%
  left_join(select(airports, faa, dest_name = name),
            by = c("dest" = "faa")) %>%
  group_by(origin_name, dest_name) %>%
  summarise(mean_delay = mean(arr_delay),
            n_flights = n()) %>%
  arrange(desc(mean_delay))

```

1. First join gets the origin name.

2. Rename on the fly in select().

3. Match "origin" column to "faa."

4. Second join gets the destination name by repeating the three steps.

5. Group by both origin and destination

# A tibble: 218 x 4

	origin_name	dest_name	mean_delay	n_flights
	<chr>	<chr>	<dbl>	<int>
1	Newark Liberty Intl	Columbia Metropolitan	44.6	94
2	Newark Liberty Intl	Mc Ghee Tyson	41.2	313

# FILTERING JOINS

FILTER ONE TABLE AGAINST THE ROWS OF ANOTHER

# MUTATING VS. FILTERING JOINS

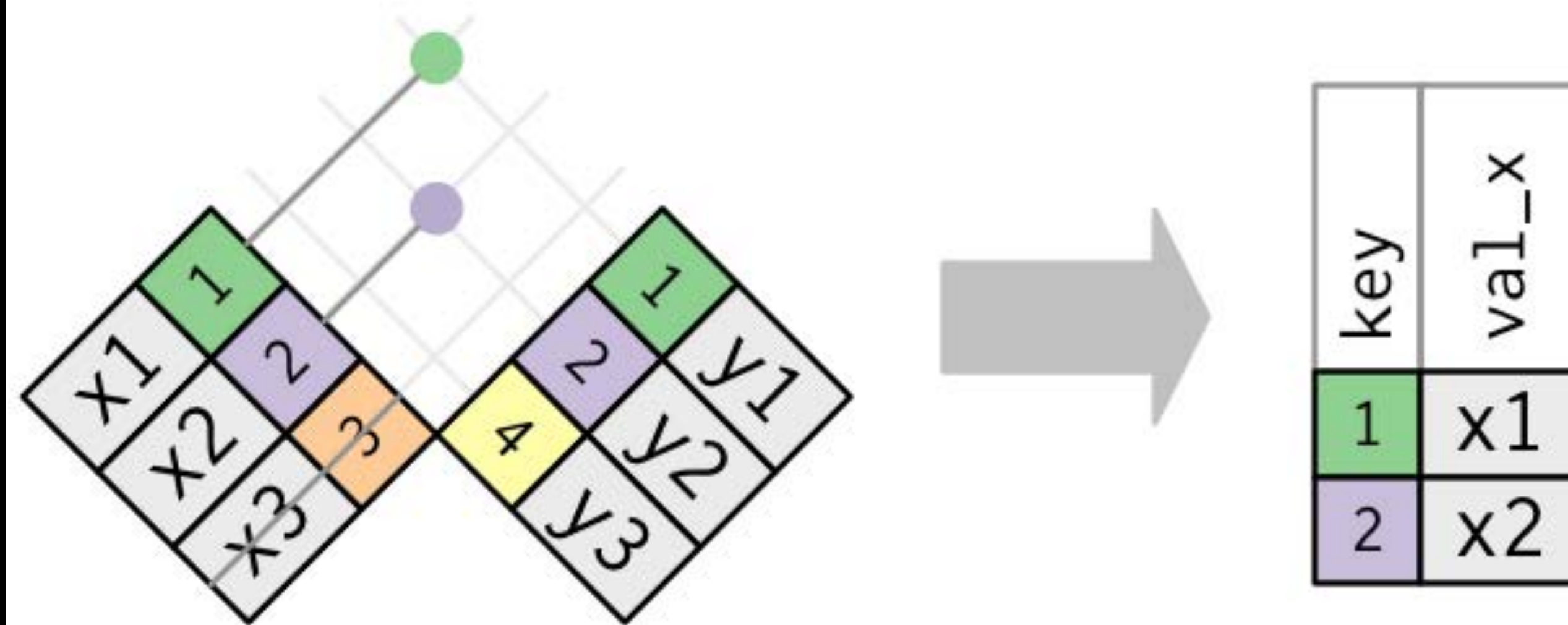
- **Mutating joins** use information from one data set to add variables to another data set (like `mutate()`)
- **Filtering joins** use information from one data set to extract cases from another data set (like `filter()`)



## FILTERING JOINS

# SEMI-JOIN

- Keeps all observations in first table that have a match in second table.
- Useful for matching filtered summary tables back to the original rows.



# SEMI-JOIN

```
band %>% semi_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

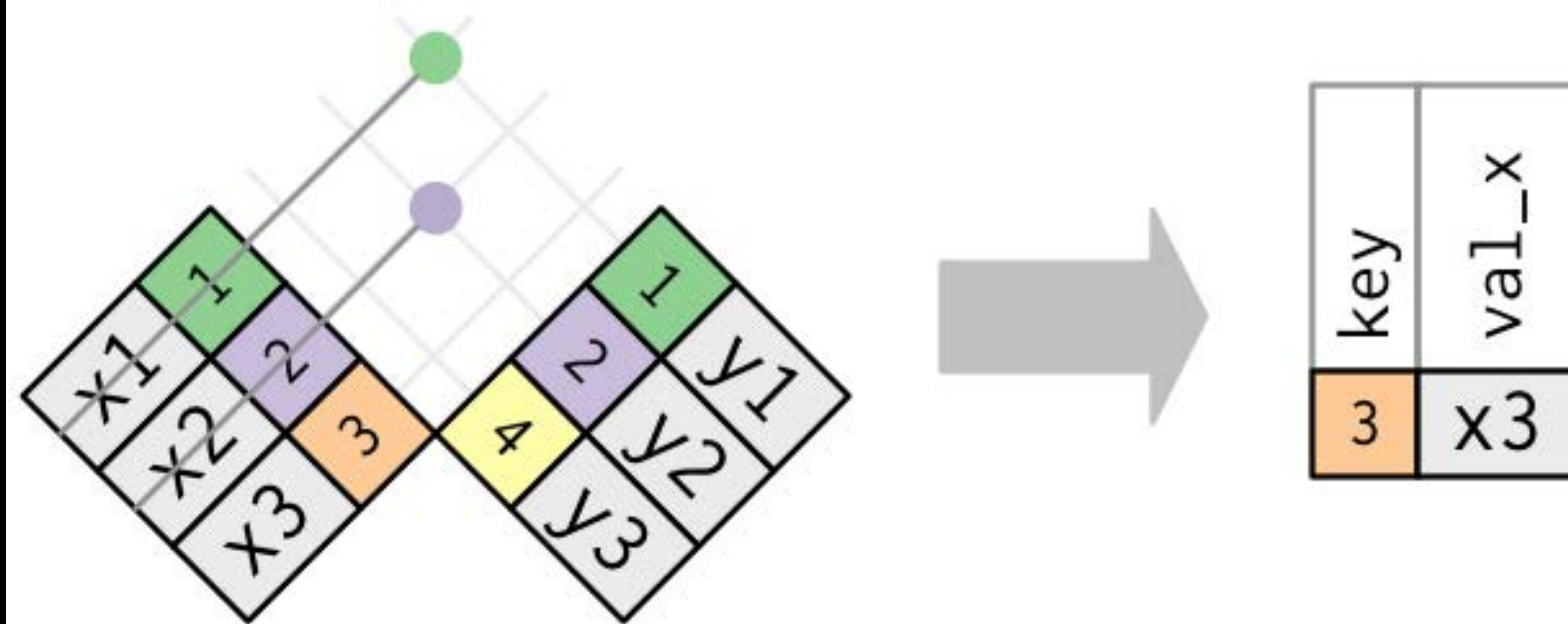
name	plays
John	guitar
Paul	bass
Keith	guitar

name	band
John	Beatles
Paul	Beatles

## FILTERING JOINS

# ANTI-JOIN

- Keeps the rows that **don't** have a match (inverse of semi-join).
- Most useful for diagnosing join mismatches.



# ANTI-JOIN

```
band %>% anti_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

name	band
Mick	Stones



# AIRPORT NAMES

```
airports %>% select(1:2)
```

<b>faa</b> <chr>	<b>name</b> <chr>
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport
0G6	Williams County Airport
0G7	Finger Lakes Regional Airport

```
flights %>% select(14:15)
```

<b>dest</b> <chr>	<b>air_time</b> <dbl>
IAH	227
IAH	227
MIA	160
BQN	183
ATL	116
ORD	150
FLL	158
IAD	53

# ACTIVITY 4

- How many airports in **airports** are serviced by flights originating in New York (i.e. flights in our dataset?)

```
# A tibble: 101 x 8
  faa   name                                lat   lon   alt   tz dst  tzone
  <chr> <chr>                                <dbl> <dbl> <int> <dbl> <chr> <chr>
1 ABQ   Albuquerque International Sunport    35.0 -107.  5355  -7 A    America/Denver
2 ACK   Nantucket Mem                        41.3  -70.1   48   -5 A    America/New_York
3 ALB   Albany Intl                          42.7  -73.8   285  -5 A    America/New_York
4 ANC   Ted Stevens Anchorage Intl          61.2 -150.   152  -9 A    America/
Anchorage
5 ATL   Hartsfield Jackson Atlanta Intl     33.6  -84.4  1026  -5 A    America/New_York
6 AUS   Austin Bergstrom Intl                30.2  -97.7   542  -6 A    America/Chicago
```

```
airports %>%
  semi_join(flights, by = c("faa" = "dest"))
```

# A tibble: 101 x 8

	faa	name	lat	lon	alt	tz	dst	tzone
	<chr>	<chr>	<dbl>	<dbl>	<int>	<dbl>	<chr>	<chr>
1	ABQ	Albuquerque International Sunport	35.0	-107.	5355	-7	A	America/Denver
2	ACK	Nantucket Mem	41.3	-70.1	48	-5	A	America/New_York
3	ALB	Albany Intl	42.7	-73.8	285	-5	A	America/New_York
4	ANC	Ted Stevens Anchorage Intl	61.2	-150.	152	-9	A	America/Anchorage
5	ATL	Hartsfield Jackson Atlanta Intl	33.6	-84.4	1026	-5	A	America/New_York
6	AUS	Austin Bergstrom Intl	30.2	-97.7	542	-6	A	America/Chicago

# ACTIVITY 5

- What are the unique **dest** codes present in the **flights** table that have no corresponding information in the **airports** table?.

```
# A tibble: 4 x 1
  dest
  <chr>
1 BQN
2 SJU
3 STT
4 PSE
```



```
flights %>%  
  anti_join(airports, c("dest" = "faa")) %>%  
  distinct(dest)
```

Removes rows with duplicate values in a column, and (by default) discards other columns.

```
# A tibble: 4 x 1  
  dest  
  <chr>  
1 BQN  
2 SJU  
3 STT  
4 PSE
```

```
flights %>%
```

```
  anti_join(rename(airports, dest = faa)) %>%
```

```
  distinct(dest, .keep_all = TRUE)
```

Keep all columns with  
**.keep\_all = TRUE**

```
# A tibble: 4 x 10
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	...
1	2013	1	1	544	545	-1	1004	1022	-18	B6	...
2	2013	1	1	615	615	0	1039	1100	-21	B6	...
3	2013	1	1	909	810	59	1331	1315	16	AA	...
4	2013	1	1	2353	2359	-6	425	445	-20	B6	...

# RECAP: JOINS

 **left\_join()** retains all cases in **left** data set

 **right\_join()** retains all cases in **right** data set

 **full\_join()** retains all cases in **either** data set

 **inner\_join()** retains only cases in **both** data sets

 **semi\_join()** extracts cases that **have a match**

 **anti\_join()** extracts cases that **do not have a match**