Data Reshaping in R

Reshaping and joining data in R

by Martin Frigaard

Written: September 21 2021

Updated: November 30 2021

Created using the "λέξις" theme

Objectives

1) Recap `dplyr`'s data manipulation verbs

2) `tidyr`'s pivoting functions

3) Joins with `dplyr`

Materials

Follow along with the exercises:

https://mjfrigaard.github.io/csuc-data-journalism/lessons.html

A web version of these slides is located:

https://mjfrigaard.github.io/csuc-data-journalism/slides.html

Data Manipulation Recap

We previously learned how to:

1) View data with `glimpse()`

2) Select columns with `select()`

3) Filter rows with `filter()`

4) Arrange data with `arrange()`

5) Create/change columns with `mutate()`

`dplyr` = a package for *manipulating* data `tidyr` = a package for *reshaping* data

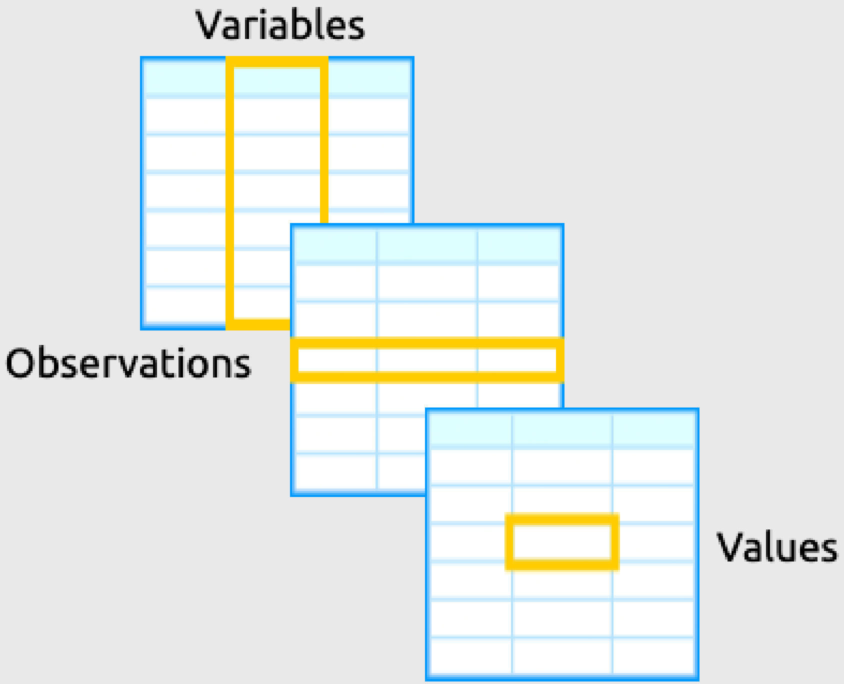
Tidy data

What are tidy data?

Observations are in rows

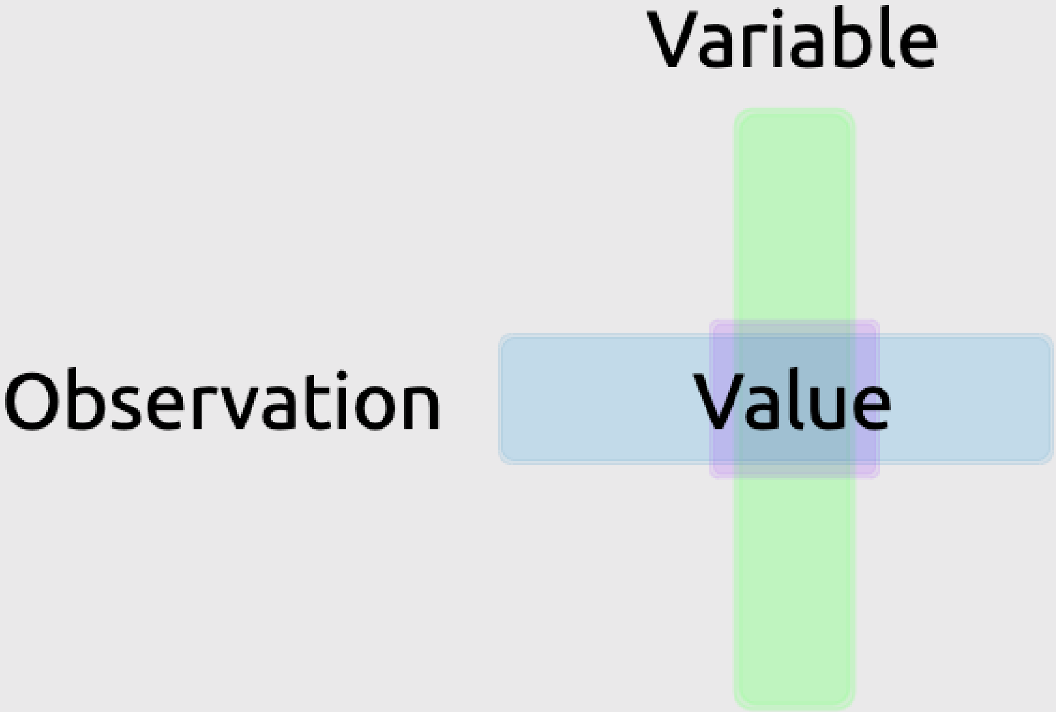
Variables are in columns

Values are in cells



Tidy data

Values are the *intersection* of observations and variables



Non-tidy data

Copy and paste the code below to create `NotTidy`

```
# copy and paste me!
NotTidy <- tibble::tribble(
       ~group,   ~`2019`,   ~`2020`,   ~`2021`,
          "A", "102/100", "123/100", "161/100",
          "B", "179/100", "199/100", "221/100",
          "C", "223/100", "146/100", "288/100")
```

group <chr>	2019 <chr>	2020 <chr>	2021 <chr>
A	102/100	123/100	161/100
B	179/100	199/100	221/100
C	223/100	146/100	288/100

3 rows

Non-tidy data

Why aren't they tidy?

```
NotTidy %>% View("NotTidy")
```

NotTidy

Filter

	group	2019	2020	2021
1	A	102/100	123/100	161/100
2	B	179/100	199/100	221/100
3	C	223/100	146/100	288/100

year is across columns...
multiple values in cells...

Tidying un-tidy data

covered in slides

- `pivot_longer()` - wide to long
- `pivot_wider()` - long to wide

---

covered in exercises

`separate()` - pull columns apart
`separate_rows()` - split columns down rows
`unite()` - stick columns together

`unnest()` - flatten columns
`uncount()` - duplicate rows according to a weighting variable

Quick tip: viewing your data

Make sure you view the data before assigning it to an object

Use `glimpse()` or `View("Name")`

tidyr::pivot_longer()

Make wide data long

```
NotTidy %>%
  pivot_longer(
    cols = -group,
    names_to = "year",
    values_to = "rate") %>%
  View("Tidy")
```

How it pivot_longer() works

NotTidy

Filter

	group	2019	2020	2021
1	A	102/100	123/100	161/100
2	B	179/100	199/100	221/100
3	C	223/100	146/100	288/100

Tidy

Filter

	group	year	rate
1	A	2019	102/100
2	A	2020	123/100
3	A	2021	161/100
4	B	2019	179/100
5	B	2020	199/100
6	B	2021	221/100
7	C	2019	223/100
8	C	2020	146/100
9	C	2021	288/100

tidyr::pivot_longer()

How it works:

`cols` = these are the **columns we want to reshape**

```
NotTidy %>%
  pivot_longer(cols = -group,
```

`names_to` = this is the new variable that will contain the previous **column names**

```
NotTidy %>%
  pivot_longer(cols = -group,
               names_to = "year",
```

`values_to` = this is the new variable that will contain the reshaped **values**

```
NotTidy %>%
  pivot_longer(cols = -group,
               names_to = "year",
               values_to = "rate")
```

tidyr::pivot_longer()

Looks correct?

```
NotTidy %>%
  pivot_longer(cols = -group,
               names_to = "year",
              values_to = "rate") %>%
  View("Tidy")
```

Not quite

Tidy

Filter

	group	year	rate
1	A	2019	102/100
2	A	2020	123/100
3	A	2021	161/100
4	B	2019	179/100
5	B	2020	199/100
6	B	2021	221/100
7	C	2019	223/100
8	C	2020	146/100
9	C	2021	288/100

`pivot_longer()` = names_transform

Print `Tidy` to console

```
Tidy
```

group	year	rate
<chr>	<chr>	<chr>
A	2019	102/100
A	2020	123/100
A	2021	161/100
B	2019	179/100
B	2020	199/100
B	2021	221/100
C	2019	223/100
C	2020	146/100
C	2021	288/100

9 rows

Note the format of the columns

The new `year` variable should be numeric

`pivot_longer()` = `names_transform`

We can control this behavior with `names_transform = list()`

```
NotTidy %>%
  pivot_longer(
      cols = -group,
      names_to = "year",
      values_to = "rate",
      names_transform = list(
        year = as.numeric))
```

group <chr>	year <dbl>	rate <chr>
A	2019	102/100
A	2020	123/100
A	2021	161/100
B	2019	179/100
B	2020	199/100
B	2021	221/100
C	2019	223/100
C	2020	146/100
C	2021	288/100

9 rows

`pivot_longer()` = names_sep

Create the `SiteRates` data (example of COVID rates as various hospitals)

```
SiteRates <- tibble::tribble(
         ~site, ~`2019_Q1`, ~`2019_Q2`, ~`2019_Q3`, ~`2019_Q4`,
      "Boston",         52,         31,         26,       33.4,
"Philadelphia",       7.42,       5.51,       5.82,       6.99,
  "Cincinnati",       6.73,       4.87,       5.02,       4.66,
       "Texas",       18.2,       16.6,         17,         19)
```

`pivot_longer()` = names_sep

Create the `SiteRates` data (example of COVID rates as various hospitals)

```
SiteRates %>% View("SiteRates")
```

SiteRates

Filter

	site	2019_Q1	2019_Q2	2019_Q3	2019_Q4
1	Boston	52.00	31.00	26.00	33.40
2	Philadelphia	7.42	5.51	5.82	6.99
3	Cincinnati	6.73	4.87	5.02	4.66
4	Texas	18.20	16.60	17.00	19.00

pivot_longer() = names_sep

SiteRates has *two* variables in the same column

We can use names_sep uses a pattern to split the column (_Q)

SiteRates

Filter

	site	2019_Q1	2019_Q2	2019_Q3	2019_Q4
1	Boston	52.00	31.00	26.00	33.40
2	Philadelphia	7.42	5.51	5.82	6.99
3	Cincinnati	6.73	4.87	5.02	4.66
4	Texas	18.20	16.60	17.00	19.00

`pivot_longer()` = names_sep

- Add `names_sep = "_Q"`
- `year` and `quarter` should also be numeric

```
SiteRates %>%
  pivot_longer(
  -site,
  names_to =
    c("year", "quarter"),
  values_to = "rate",
  names_sep = "_Q",
  names_transform = list(
    year = as.integer,
    quarter = as.integer)) %>%
  View("TidySites")
```

TidySites

Filter

	site	year	quarter	rate
1	Boston	2019	1	52.00
2	Boston	2019	2	31.00
3	Boston	2019	3	26.00
4	Boston	2019	4	33.40
5	Philadelphia	2019	1	7.42
6	Philadelphia	2019	2	5.51
7	Philadelphia	2019	3	5.82
8	Philadelphia	2019	4	6.99
9	Cincinnati	2019	1	6.73
10	Cincinnati	2019	2	4.87
11	Cincinnati	2019	3	5.02
12	Cincinnati	2019	4	4.66
13	Texas	2019	1	18.20
14	Texas	2019	2	16.60
15	Texas	2019	3	17.00
16	Texas	2019	4	19.00

More `pivot_longer()` options

Create the `SpaceDogs` data. These are names of dogs in the [Soviet Space Dogs database](#).

Some dogs have multiple names, and some share names.

```r
# data.frame prints to the screen better
SpaceDogs <- data.frame(
   date = c("1966-02-22", "1961-03-25",
              "1961-03-09", "1960-12-22",
              "1960-12-01", "1960-09-22"),
   result = c("recovered safely", "recovered safely",
              "recovered safely", "recovered",
              "both dogs died","recovered safely"),
   name_1 = c("Ugolyok / Snezhok", "Zvezdochka",
              "Chernuskha", "Shutka", "Mushka",
              "Kusachka / Otvazhnaya"),
   name_2 = c("Veterok / Bzdunok", NA, NA,
              "Kometka", "Pchyolka", "Neva"))
```

More `pivot_longer()` options

```r
SpaceDogs %>%
  View("SpaceDogs")
```

SpaceDogs

Filter

	date	result	name_1	name_2
1	1966-02-22	recovered safely	Ugolyok / Snezhok	Veterok / Bzdunok
2	1961-03-25	recovered safely	Zvezdochka	NA
3	1961-03-09	recovered safely	Chernuskha	NA
4	1960-12-22	recovered	Shutka	Kometka
5	1960-12-01	both dogs died	Mushka	Pchyolka
6	1960-09-22	recovered safely	Kusachka / Otvazhnaya	Neva

The `SpaceDogs` data has missing values in `name_2`

...two of the columns have a similar prefix (`name_`)

More `pivot_longer()` options

To tidy these data, we can combine what we've learned:

1. `dplyr::starts_with()` to select the `name_` columns

2. `names_to` has a special `".value"` argument, which is the name of the new column with the `name_` values. We need to include an index column to track the values across different names (`dog_id`).

3. We can remove missing values with `values_drop_na`

More `pivot_longer()` options

```r
SpaceDogs %>%
  pivot_longer( # columns with `name_` prefix
   starts_with("name_"),
```

```r
SpaceDogs %>%
  pivot_longer(
   starts_with("name_"),
   names_sep = "_",
   names_to = # new column for `name_` values
     c(".value", "dog_id"),
```

```r
SpaceDogs %>%
  pivot_longer(
   starts_with("name_"),
   names_sep = "_",
   names_to =
     c(".value", "dog_id"), # remove missing
   values_drop_na = TRUE)
```

More `pivot_longer()` options

Add these arguments to `pivot_longer()` and add `View()`

```r
SpaceDogs %>%

  pivot_longer(
   starts_with("name_"),
   names_sep = "_",
   names_to =
     c(".value", "dog_id"),
   values_drop_na = TRUE) %>%

  View("TidySpaceDogs")
```

TidySpaceDogs

Filter

	date	result	dog_id	name
1	1966-02-22	recovered safely	1	Ugolyok / Snezhok
2	1966-02-22	recovered safely	2	Veterok / Bzdunok
3	1961-03-25	recovered safely	1	Zvezdochka
4	1961-03-09	recovered safely	1	Chernuskha
5	1960-12-22	recovered	1	Shutka
6	1960-12-22	recovered	2	Kometka
7	1960-12-01	both dogs died	1	Mushka
8	1960-12-01	both dogs died	2	Pchyolka
9	1960-09-22	recovered safely	1	Kusachka / Otvazhnaya
10	1960-09-22	recovered safely	2	Neva

More `pivot_longer()` options

We can see the new `dog_id` index

The missing values have been removed

SpaceDogs

	date	result	name_1	name_2
1	1966-02-22	recovered safely	Ugolyok / Snezhok	Veterok / Bzdunok
2	1961-03-25	recovered safely	Zvezdochka	NA
3	1961-03-09	recovered safely	Chernuskha	NA
4	1960-12-22	recovered	Shutka	Kometka
5	1960-12-01	both dogs died	Mushka	Pchyolka
6	1960-09-22	recovered safely	Kusachka / Otvazhnaya	Neva

TidySpaceDogs

	date	result	dog_id	name
1	1966-02-22	recovered safely	1	Ugolyok / Snezhok
2	1966-02-22	recovered safely	2	Veterok / Bzdunok
3	1961-03-25	recovered safely	1	Zvezdochka
4	1961-03-09	recovered safely	1	Chernuskha
5	1960-12-22	recovered	1	Shutka
6	1960-12-22	recovered	2	Kometka
7	1960-12-01	both dogs died	1	Mushka
8	1960-12-01	both dogs died	2	Pchyolka
9	1960-09-22	recovered safely	1	Kusachka / Otvazhnaya
10	1960-09-22	recovered safely	2	Neva

`tidyr::pivot_wider()`

We've made wide data long, now we will make long data wide

But, why???

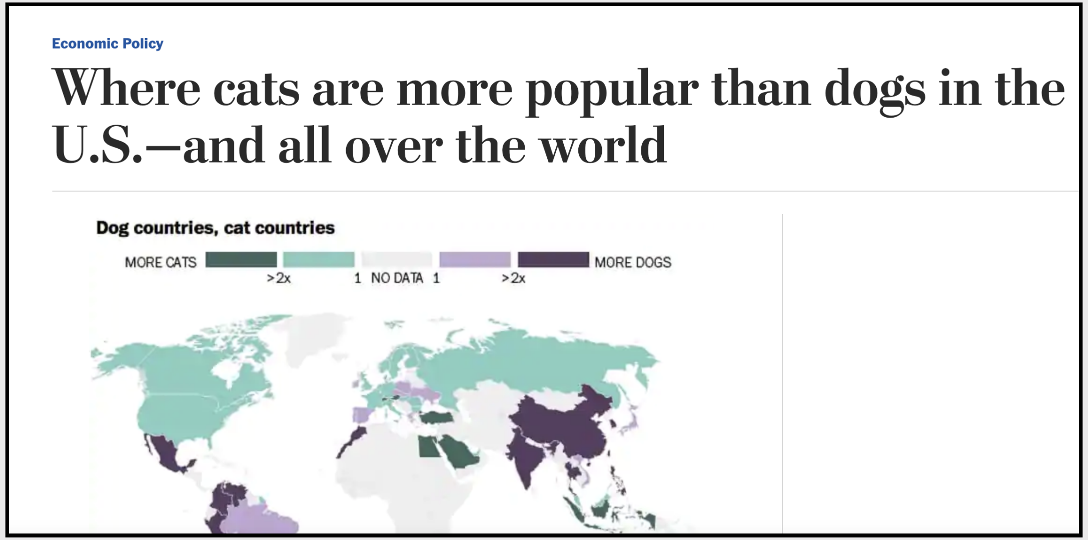
Wide data is usually better for displaying summaries

Long data is better for graphing/modeling

tidyr::pivot_wider()

Consider the `CatVsDogWide` data

These data come from this article in the Washington Post.



tidyr::pivot_wider()

Create `CatVsDogWide`

```
CatVsDogWide <- tibble::tribble(
                   ~metric,   ~CA,  ~TX,  ~FL,  ~NY,  ~PA,
    "no_of_households", 12974, 9002, 7609, 7512, 5172,
"no_of_pet_households",  6865, 5265, 4138, 3802, 2942,
"no_of_dog_households",  4260, 3960, 2718, 2177, 1702,
"no_of_cat_households",  3687, 2544, 2079, 2189, 1748)
```

tidyr::pivot_wider()

```r
CatVsDogWide %>% View("CvDWide")
```

CvDWide

Filter

	metric	CA	TX	FL	NY	PA
1	no_of_households	12974	9002	7609	7512	5172
2	no_of_pet_households	6865	5265	4138	3802	2942
3	no_of_dog_households	4260	3960	2718	2177	1702
4	no_of_cat_households	3687	2544	2079	2189	1748

tidyr::pivot_wider()

Step 1. `pivot_long()` states into `state` column, values into `value` column

```
CatVsDogWide %>%
  pivot_longer(-metric,
    names_to = "state",
    values_to = "value") %>%
  # view
  View("CvDLong")
```

CvDLong

Filter

	metric	state	value
1	no_of_households	CA	12974
2	no_of_households	TX	9002
3	no_of_households	FL	7609
4	no_of_households	NY	7512
5	no_of_households	PA	5172
6	no_of_pet_households	CA	6865
7	no_of_pet_households	TX	5265
8	no_of_pet_households	FL	4138
9	no_of_pet_households	NY	3802
10	no_of_pet_households	PA	2942
11	no_of_dog_households	CA	4260
12	no_of_dog_households	TX	3960
13	no_of_dog_households	FL	2718
14	no_of_dog_households	NY	2177
15	no_of_dog_households	PA	1702
16	no_of_cat_households	CA	3687
17	no_of_cat_households	TX	2544
18	no_of_cat_households	FL	2079
19	no_of_cat_households	NY	2189
20	no_of_cat_households	PA	1748

tidyr::pivot_wider()

Calculate percent pets per household, dog owners, and cat owners

Step 1. `pivot_long()` states into `state` column, values into `value` column
Step 2. `pivot_wider()` the `metric` across columns

```
CatVsDogWide %>%
  pivot_longer(-metric, names_to = "state", values_to = "value") %>%
  pivot_wider(names_from = "metric", values_from = "value") %>%
  View("CvDWide")
```

CvDWide

Filter

	state	no_of_households	no_of_pet_households	no_of_dog_households	no_of_cat_households
1	CA	12974	6865	4260	3687
2	TX	9002	5265	3960	2544
3	FL	7609	4138	2718	2079
4	NY	7512	3802	2177	2189
5	PA	5172	2942	1702	1748

`tidyr::pivot_wider()`

Calculate percent pets per household, dog owners, and cat owners

Step 1. `pivot_long()` states into `state` column, values into `value` column
Step 2. `pivot_wider()` the `metric` and `values` across columns
Step 3. `mutate()` the `perc` (percentage) columns

```
CatVsDogWide %>%
   pivot_longer(-metric, names_to = "state", values_to = "value") %>%
   pivot_wider(names_from = "metric", values_from = "value") %>%
  mutate(
   perc_pet_household = no_of_pet_households / no_of_households * 100,
   perc_pet_household = round(perc_pet_household, digits = 1),
   perc_dog_owners = no_of_dog_households / no_of_households * 100,
   perc_dog_owners = round(perc_dog_owners, digits = 1),
   perc_cat_owners = no_of_cat_households / no_of_households * 100,
   perc_cat_owners = round(perc_cat_owners, digits = 1)) %>%
  View("CvDWide")
```

tidyr::pivot_wider()

Calculate percent pets per household, dog owners, and cat owners

Step 1. `pivot_long()` states into `state` column, values into `value` column
Step 2. `pivot_wider()` the `metric` and `values` across columns
Step 3. `mutate()` the `perc` (percentage) columns

CvDWide

Filter

	state	no_of_households	no_of_pet_households	no_of_dog_households	no_of_cat_households	perc_pet_household	perc_dog_owners	perc_cat_owners
1	CA	12974	6865	4260	3687	52.9	32.8	28.4
2	TX	9002	5265	3960	2544	58.5	44.0	28.3
3	FL	7609	4138	2718	2079	54.4	35.7	27.3
4	NY	7512	3802	2177	2189	50.6	29.0	29.1
5	PA	5172	2942	1702	1748	56.9	32.9	33.8

tidyr::pivot_wider()

We could assign here, but why stop?

Add `select()` helpers to reorganize data!

```
# ...code omitted
   perc_cat_owners = no_of_cat_households / no_of_households * 100,
   perc_cat_owners = round(perc_cat_owners, digits = 1)) %>%
    # pass the output from pivot_wider() to select()
  select(state,
         contains("perc_dog"), contains("perc_cat"))
```

`tidyr::pivot_wider()` (***Final code***)

```
CatVsDogWide %>%
  # pivots
   pivot_longer(
     -metric, names_to = "state", values_to = "value") %>%
   pivot_wider(
     names_from = "metric", values_from = "value") %>%
  # mutate
  mutate(
   perc_pet_household = no_of_pet_households / no_of_households * 100,
   perc_pet_household = round(perc_pet_household, digits = 1),
   perc_dog_owners = no_of_dog_households / no_of_households * 100,
   perc_dog_owners = round(perc_dog_owners, digits = 1),
   perc_cat_owners = no_of_cat_households / no_of_households * 100,
   perc_cat_owners = round(perc_cat_owners, digits = 1)) %>%
  # select
  select(state,
        contains("perc_dog"), contains("perc_cat"))
```

`tidyr::pivot_wider()` (***Final output***)

CvsDPercent

Filter

	state	perc_dog_owners	perc_cat_owners
1	CA	32.8	28.4
2	TX	44.0	28.3
3	FL	35.7	27.3
4	NY	29.0	29.1
5	PA	32.9	33.8

More tidying in the exercises!

`separate()` - pull columns apart

`separate_rows()` - split columns down rows

`unite()` - stick columns together

`unnest()` - flatten columns

`uncount()` - duplicate rows according to a weighting variable `dplyr` = a package for *manipulating relational* data

The `dplyr` joining functions

`dplyr` has functions for joining multiple tibbles or data.frames

`left_join()`

`right_join()`

`inner_join()`

`full_join()`

*Recall that `tibble`s and `data.frame`'s are nearly identical

dplyr joins

Toy data X

```r
# create X table
X <- tibble::tribble(
    ~A, ~B, ~C,
    "a", "t", 1L,
    "b", "u", 2L,
    "c", "v", 3L)
```

A	B	C
<chr>	<chr>	<int>
a	t	1
b	u	2
c	v	3

3 rows

Toy data Y

```r
# create Y table
Y <- tibble::tribble(
    ~A, ~B, ~D,
    "a", "t", 3L,
    "b", "u", 2L,
    "d", "W", 1L)
```

A	B	D
<chr>	<chr>	<int>
a	t	3
b	u	2
d	W	1

3 rows

`dplyr` left joins

```
left_join(x = , y = )
```

...joins on matches *from* right-hand table (Y) *to* left-hand table (X)

Keep all data from X, and only matching data from Y

```
left_join(
  x = X,
  y = Y
  )
```

This creates:

A	B	C	D
<chr>	<chr>	<int>	<int>
a	t	1	3
b	u	2	2
c	v	3	*NA*

3 rows

dplyr left joins (1)

Left joins use all the data from X (the left-hand table)

X

A	B	C
a	t	1
b	u	2
c	v	3

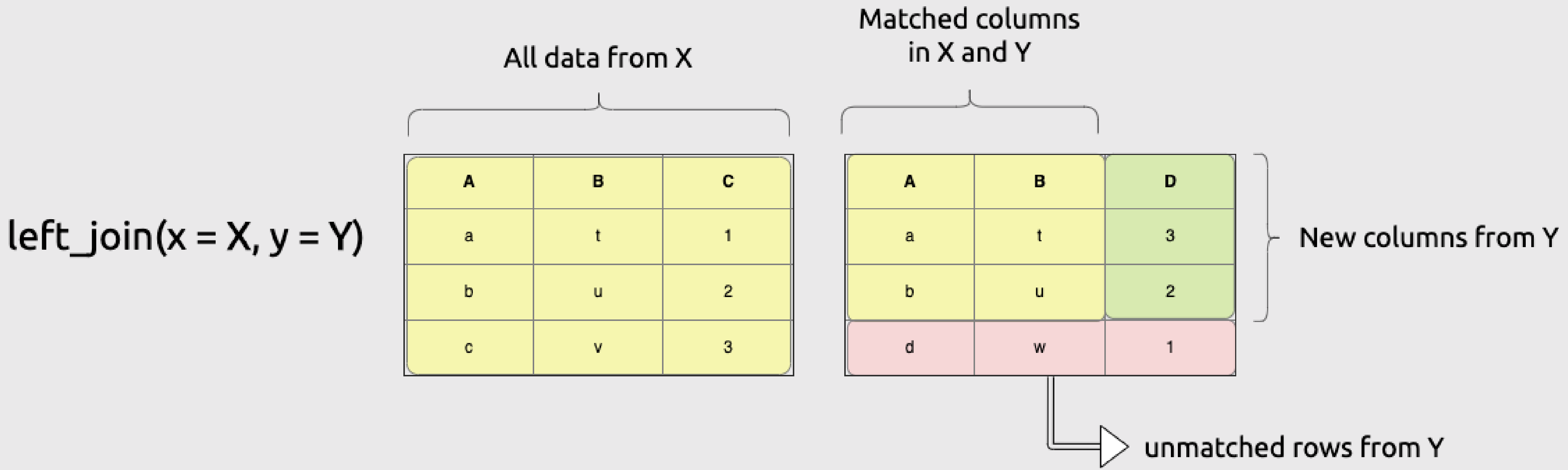
Y

A	B	D
a	t	3
b	u	2
d	w	1

left_join(x = X, y = Y)

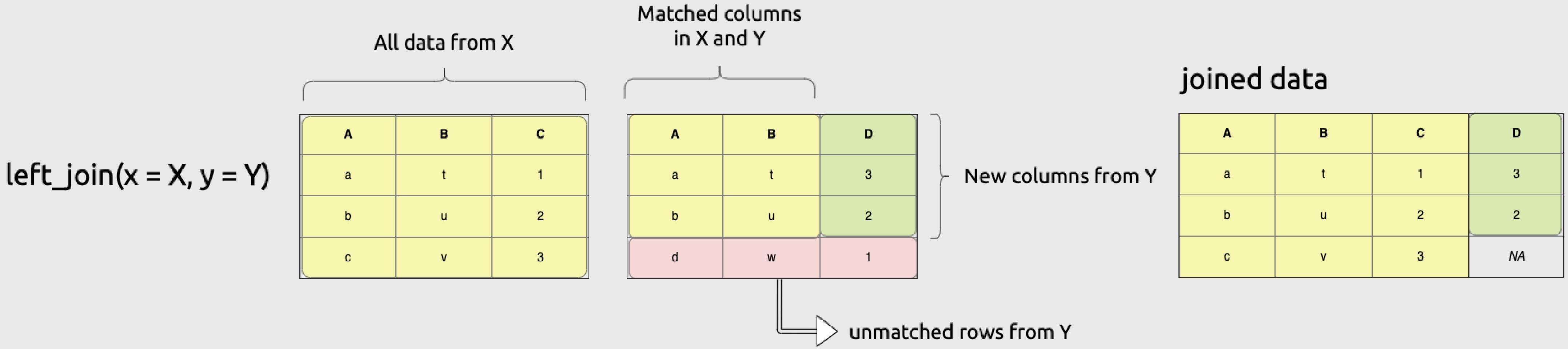
dplyr left joins (2)

Left joins include matched data in the right-hand table Y, and it carries over any new corresponding columns



`dplyr` left join (3)

The final data includes the new column(s) from Y (the right-hand table), and missing values for the unmatched rows.



dplyr right joins

right_join(x = , y =)

...join on matches *from* right-hand table (Y) *to* left-hand table (X)

Keep all data from Y, and only matching data from X

```
right_join(x = X, y = Y)
```

This creates:

A	B	C	D
<chr>	<chr>	<int>	<int>
a	t	1	3
b	u	2	2
d	W	*NA*	1

3 rows

`dplyr` right joins (1)

Right joins use all the data from Y (the right-hand table)

X

A	B	C
a	t	1
b	u	2
c	v	3

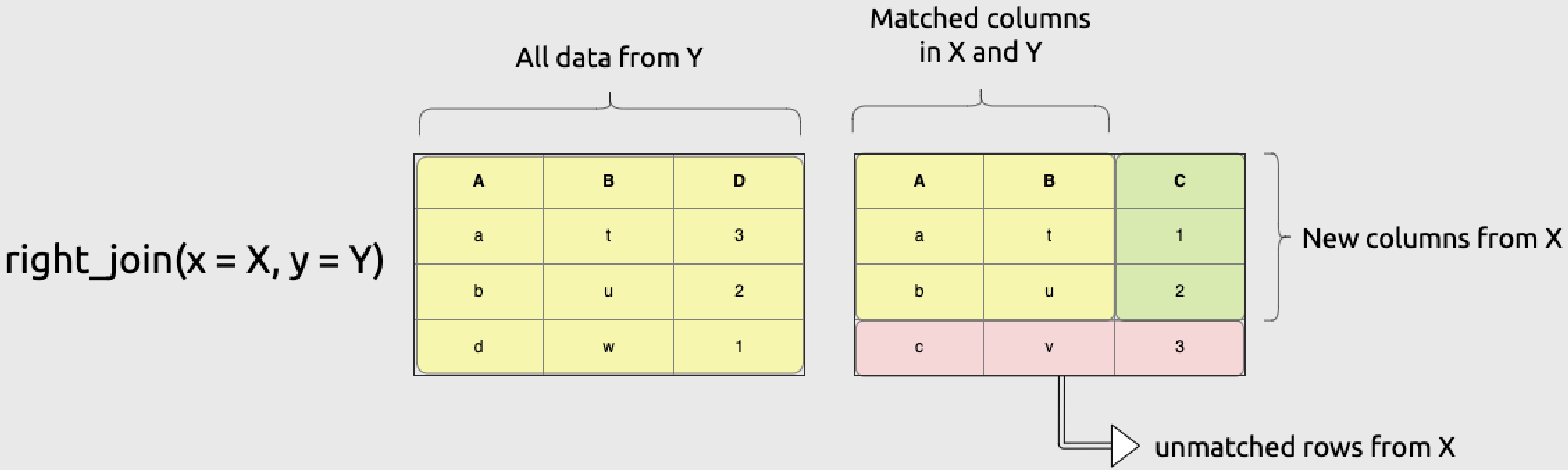
Y

A	B	D
a	t	3
b	u	2
d	w	1

right_join(x = X, y = Y)

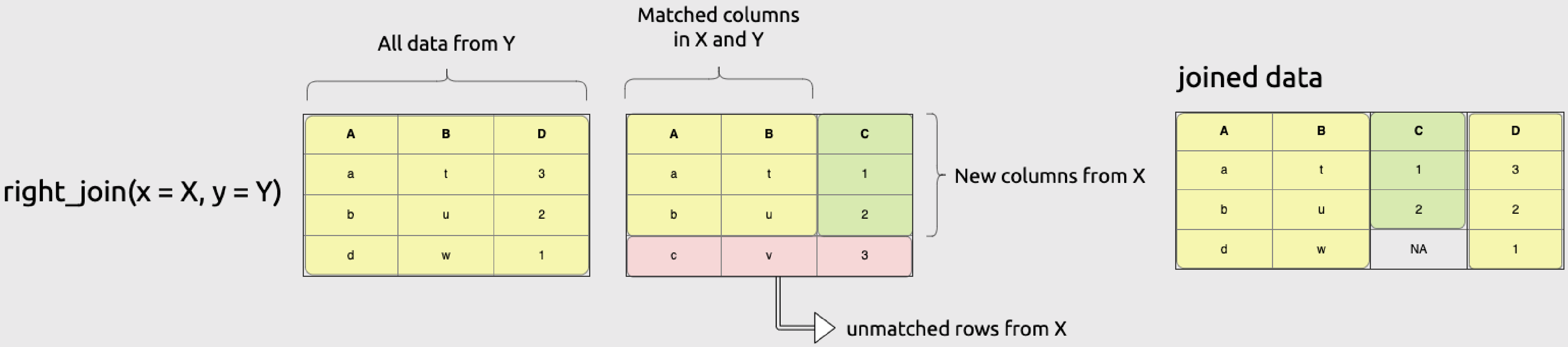
`dplyr` right joins (2)

Right joins include the matched data in the left-hand table X, and they carry over any new corresponding columns



`dplyr` right joins (3)

The final data includes the new column(s) from X (the left-hand table), and missing values for the unmatched rows.



`dplyr` inner joins

`inner_join(x = , y = )`

...keep only matches in *both* x *and* y

Keep only the matching data from X and Y

```
inner_join(x = X, y = Y)
```

This creates:

A <chr>	B <chr>	C <int>	D <int>
a	t	1	3
b	u	2	2

2 rows

`dplyr` inner joins (1)

Inner joins use only the matched data from both the X and Y tables

X

A	B	C
a	t	1
b	u	2
c	v	3

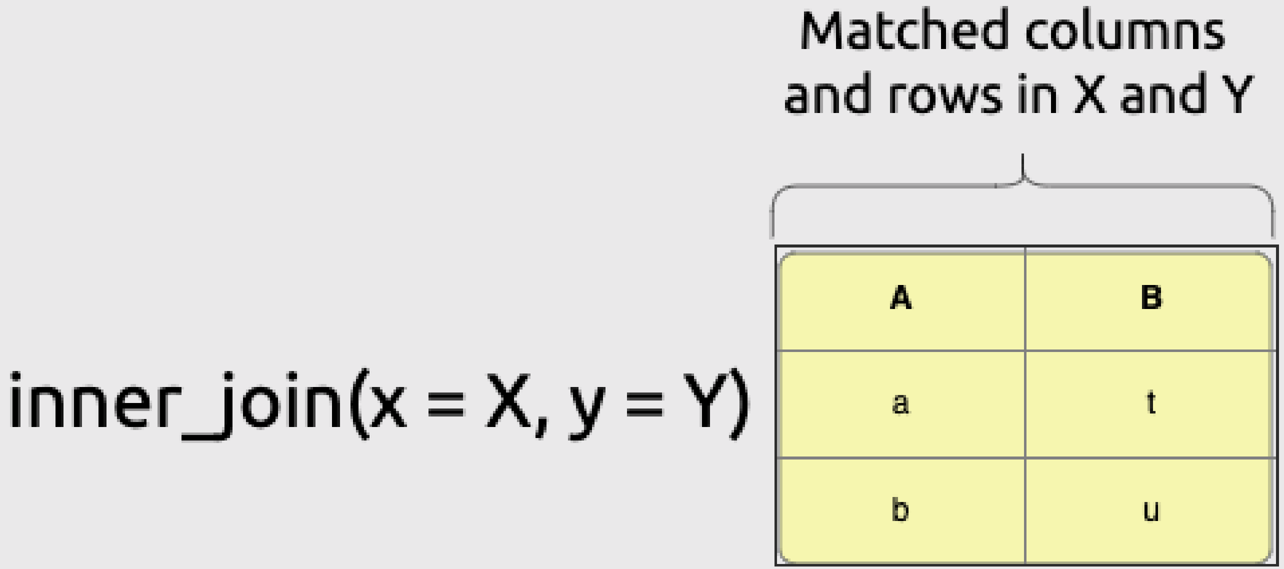
Y

A	B	D
a	t	3
b	u	2
d	w	1

inner_join(x = X, y = Y)

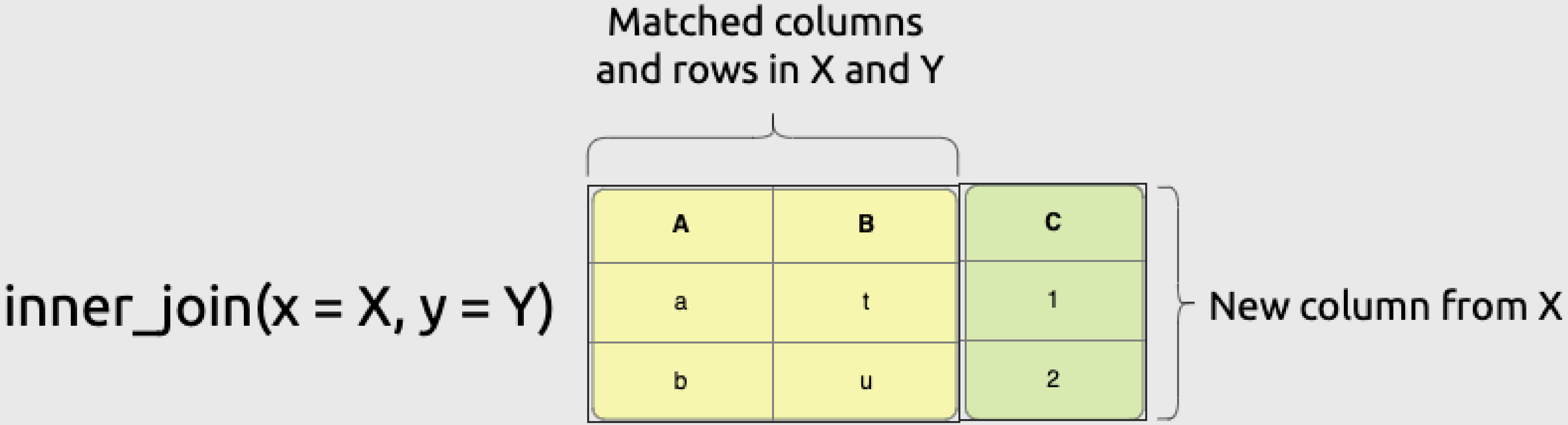
dplyr inner joins (2)

Columns A and B are matched in both X and Y tables



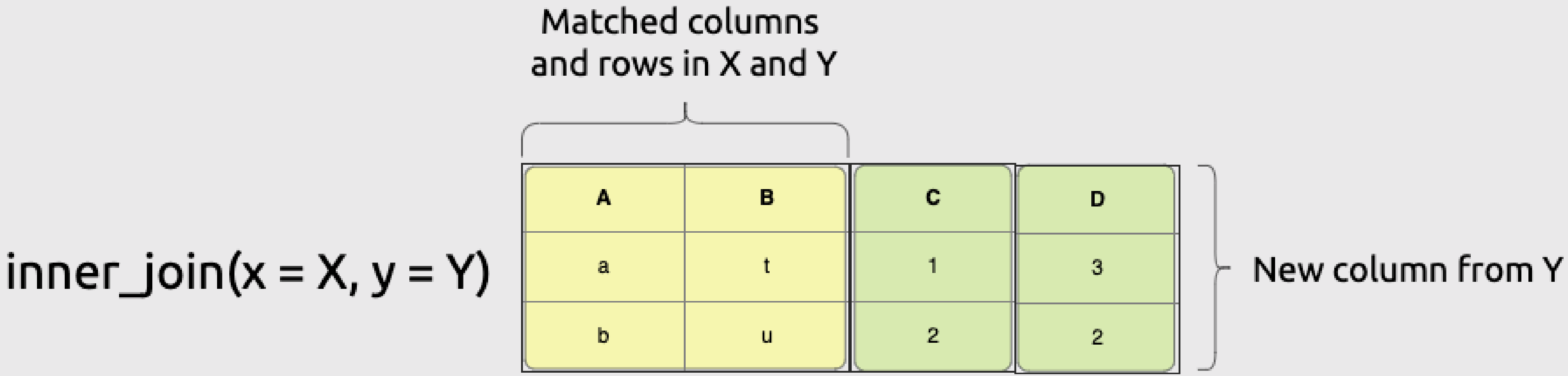
dplyr inner joins (3)

Column C from table X gets joined on matching columns A and B



`dplyr` inner joins (4)

Column D from table Y gets joined on matching columns A and B



dplyr full joins

full_join(x = X, y = Y)

...keep *all* data in both x and y

Keep all data from Y and X

```
full_join(x = X, y = Y)
```

This creates:

A <chr>	B <chr>	C <int>	D <int>
a	t	1	3
b	u	2	2
c	v	3	*NA*
d	W	*NA*	1

4 rows

dplyr full joins (1)

Full joins include all data from both tables X and Y

X

A	B	C
a	t	1
b	u	2
c	v	3

Y

A	B	D
a	t	3
b	u	2
d	w	1

full_join(x = X, y = Y)

dplyr full joins (2)

Full joins start with all data in table X

full_join(x = X, y = Y)

All data from X

A	B	C
a	t	1
b	u	2
c	v	3

dplyr full joins (3)

Full joins start with all data in table X and include the columns and rows from table Y

full_join(x = X, y = Y)

All data from X

A	B	C
a	t	1
b	u	2
c	v	3

All data from X

Data from Y

A	B	C	D
a	t	1	3
b	u	2	2
c	v	3	*NA*
d	w	*NA*	1

Data from Y

Data from Y

Resources for Data Tidying

1. R for Data Science
2. Data Wrangling with R
3. Stack Overflow questions tagged with `tidyr`
4. RStudio Community posts tagged `tidyr`