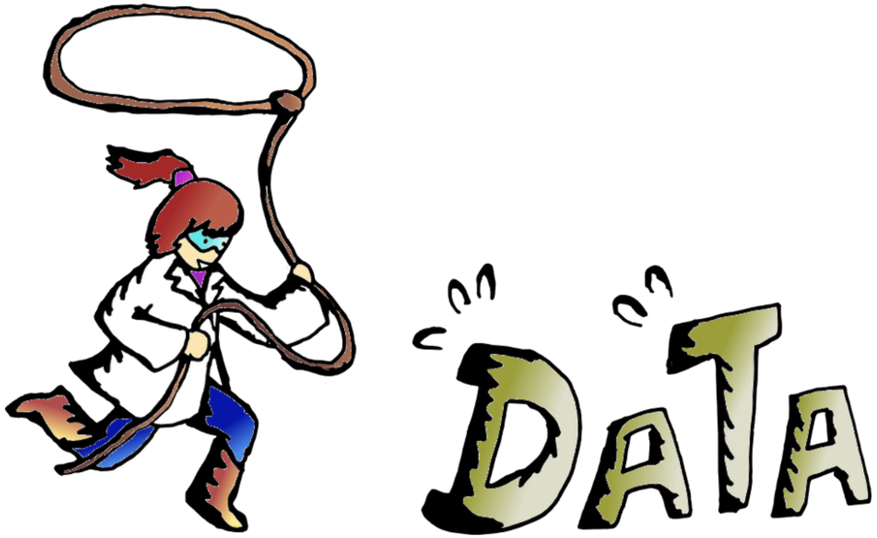
Joining, mapping, and reshaping data frames



Overview

Very quick review of logistic regression

Joining data frames

Creating maps

If there is time: reshaping data

Very quick review of logistic regression

In **logistic regression** we try to predict if a case belongs to one of two categories

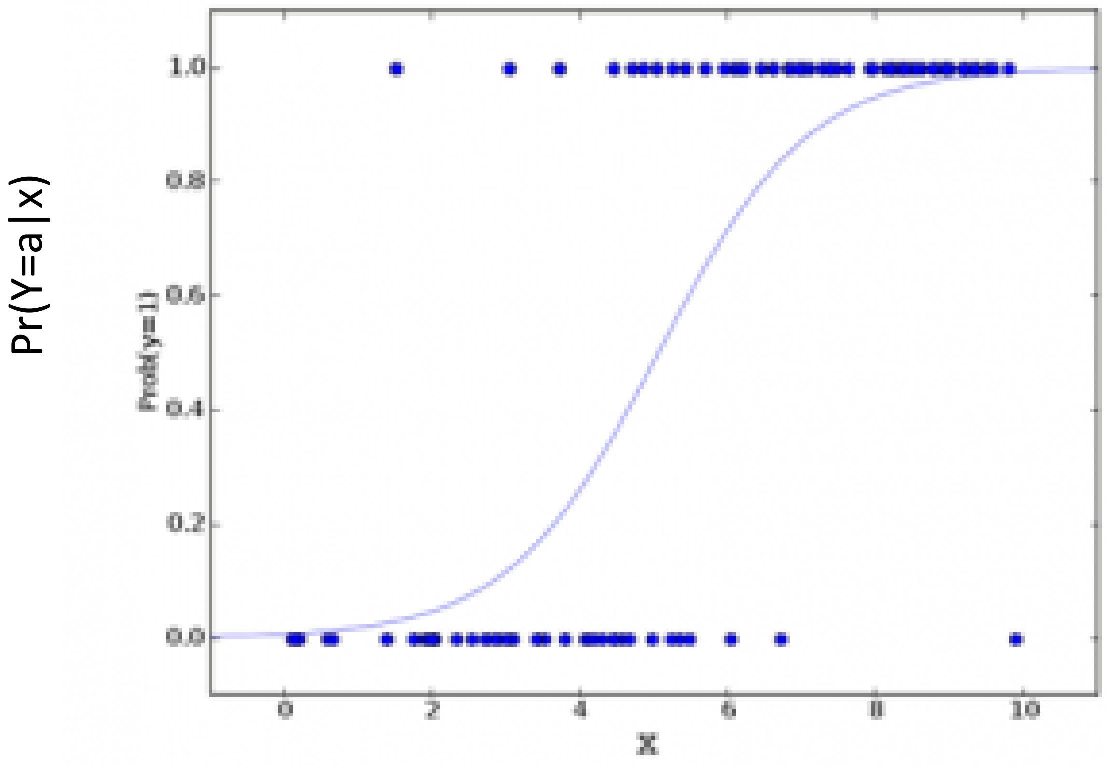
To do this we model the log-odds as a linear function of predictor variables:

$$log\left(\frac{P(Y=a|x)}{1-P(Y=a|x)}\right) = \beta_0 + \beta_1 \cdot x$$

If we write the above equation in terms of the probability of being in class a we get:

$$P(Y=a|x) = \frac{\exp(\beta_0+\beta_1 \cdot x_1)}{1+\exp(\beta_0+\beta_1 \cdot x_1)} = \frac{e^{\beta_0+\beta_1 \cdot x_1}}{1+e^{\beta_0+\beta_1 \cdot x_1}}$$

Very quick review of logistic regression



$$P(Y = a|x_1) = \frac{e^{\beta_0 + \beta_1 \cdot x_1}}{1 + e^{\beta_0 + \beta_1 \cdot x_1}}$$

Very quick review of logistic regression

We can easily extend our logistic regression model to include multiple explanatory variables

$$log(\frac{P(Y=a|x)}{1-P(Y=a|x)}) = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_1 + \hat{\beta}_2 \cdot x_2 + ... + \hat{\beta}_k \cdot x_k$$

When using a categorical predictor, x_2, in a logistic regression model, the exponential of the regression coefficient $e^{\hat{\beta}_2}$ is the ***odds ratio***

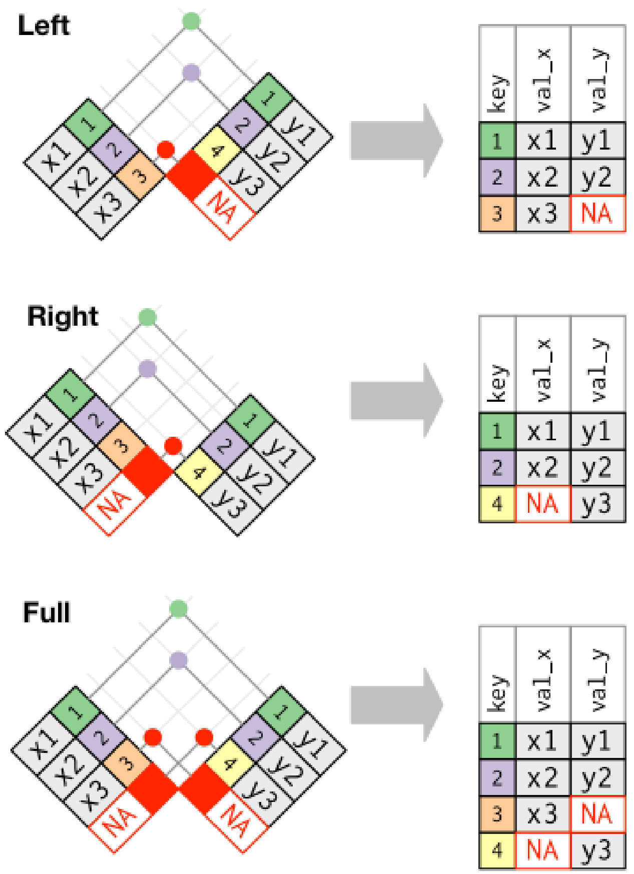
- Tells us how many times greater the odds are when $x_2 = 1$ vs. when $x_2 = 0$

We can fit logistic regression models in R using the glm() function

```
> lr_fit  <-  glm(rank_name ~ salary_tot,  data = assistant_full_data, family = "binomial")
```

Questions?

Joining data frames



Left and right tables

Suppose we have two data frames called x and y

- x have two variables called key_x, and val_x
- y has two variables called key_y and val_y

x$key_x x$val_x

1	x1
2	x2
3	x3

Data frame x

y$key_y y$val_y

1	y1
2	y2
4	y3

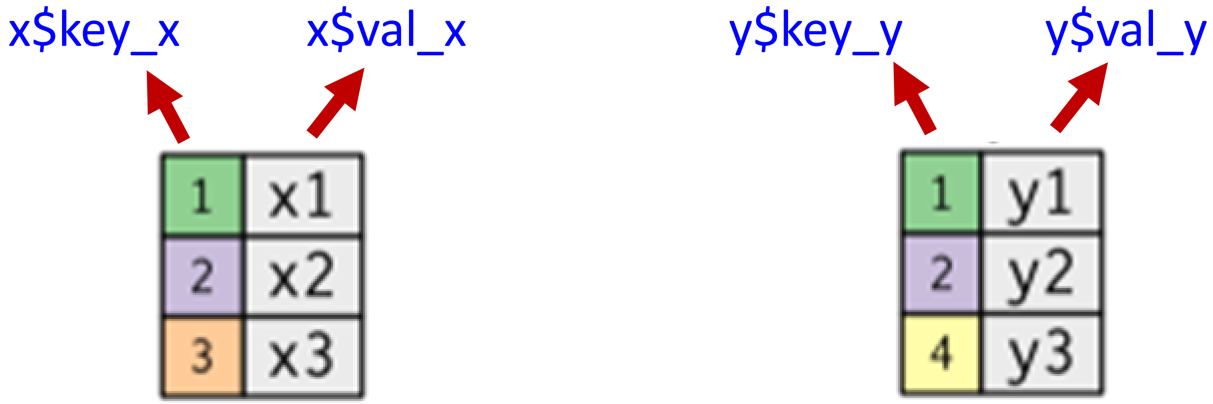
Data frame y

SDS230:download_data('x_y_join.rda')

Left and right tables

Suppose we have two data frames called x and y

- x have two variables called key_x, and val_x
- y has two variables called key_y and val_y



Data frame x **Data frame y**

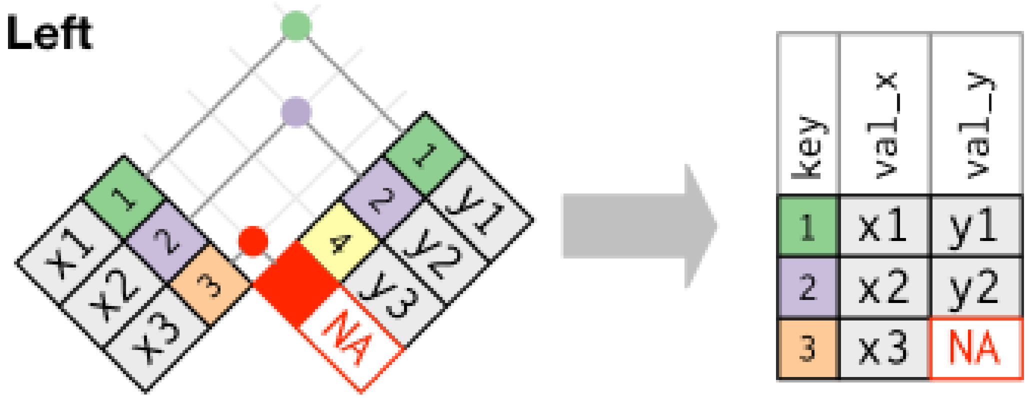
Joins have the general form:

```
join(x, y, by = c("key_x" = "key_y"))
```

Left joins

Left joins keep all rows in the left table.

Data from right table is added when there is a matching key, otherwise NA as added.

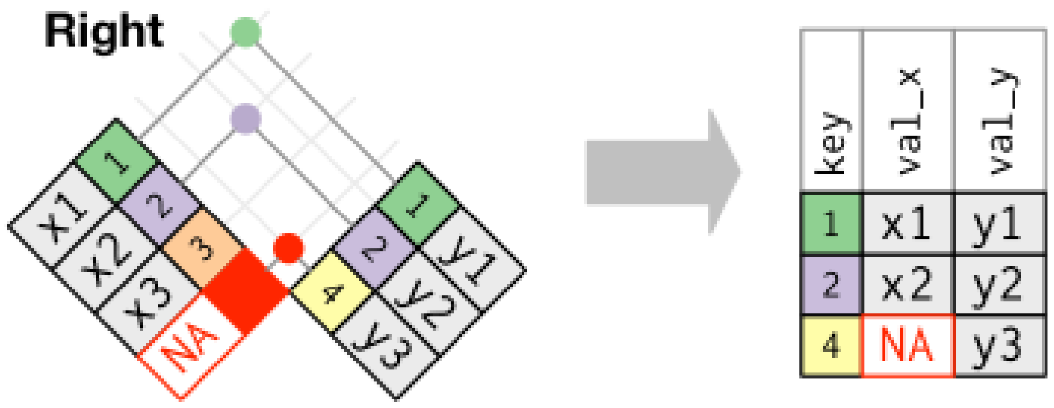



```
> left_join(x, y, by = c("key_x" = "key_y"))
```

Right joins

Right joins keep all rows in the right table.

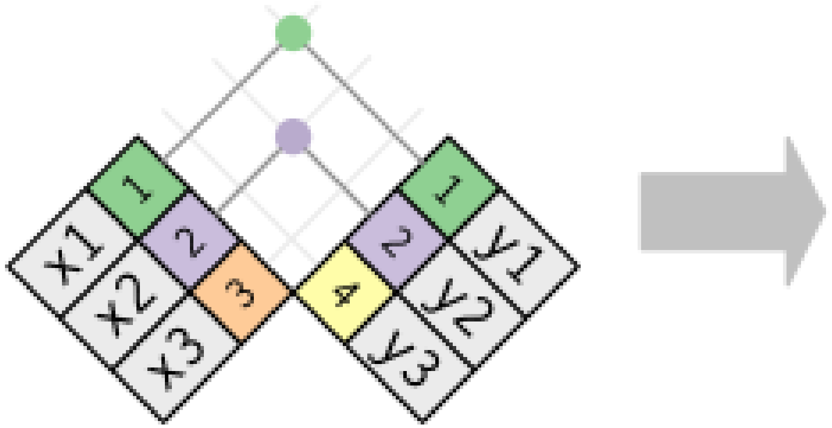
Data from left table added when there is a matching key, otherwise NA as added.



> right_join(x, y, by = c("key_x" = "key_y"))

Inner joins

Inner joins only keep rows in which there are matches between the keys in both tables.



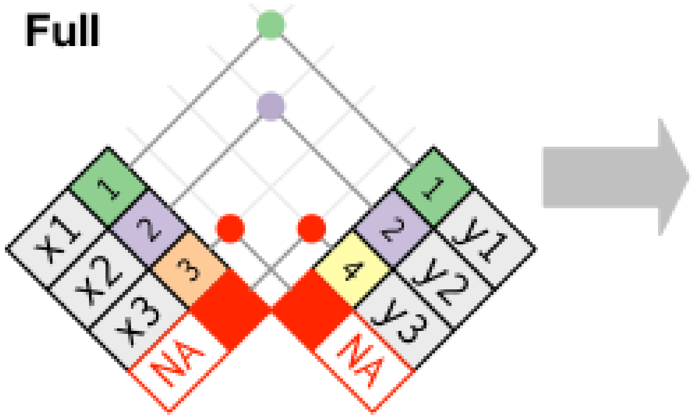
key	val_x	val_y
1	x1	y1
2	x2	y2

```
> inner_join(x, y, by = c("key_x" = "key_y"))
```

Full joins

Full joins keep all rows in both table.

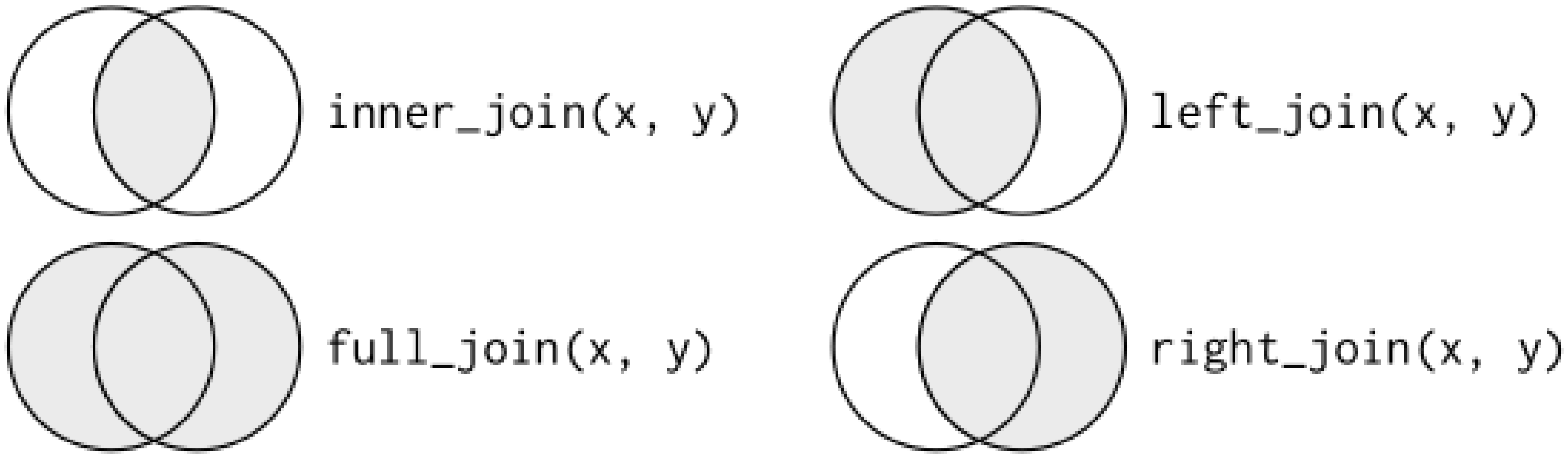
NAs are added where there are no matches.



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

> full_join(x, y, by = c("key_x" = "key_y"))

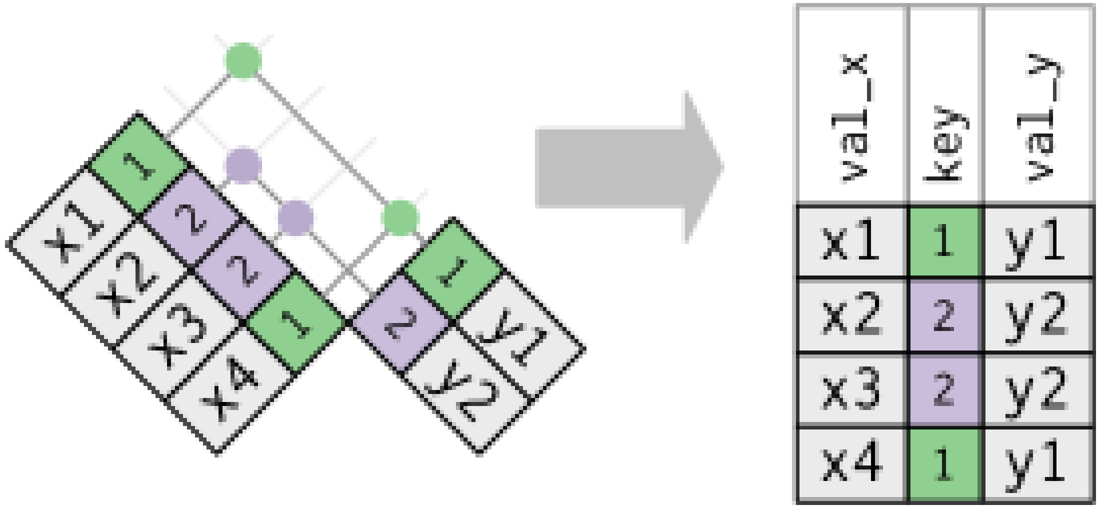
Summary



Duplicate keys

Duplicate keys are useful if there is a many-to-one relationship

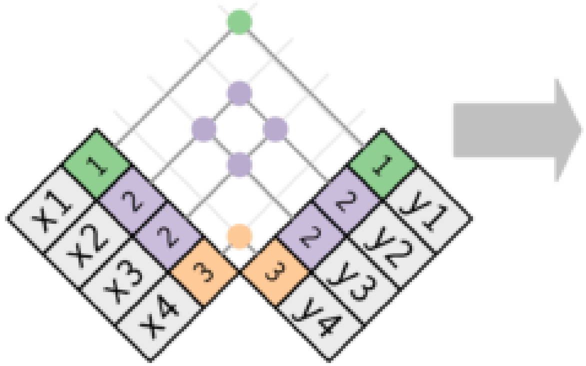
- e.g., duplicates are useful in the left table when doing a left join



Duplicate keys

If both tables have duplicate keys you get all possible combinations of joined values (Cartesian product).

- **This is usually an error!**



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

Always check the output size after you join a table because even if there is not a syntax error you might not get the table you are expecting!

- You can check how many rows a data frame has using the nrow() function

Duplicate keys

To deal with duplicate keys in both tables, we can join the tables using multiple keys in order to make sure that each row is uniquely specified.

We can do this using the syntax:

```
join(x2, y2, by = c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

Duplicate keys

```
> x2 <- data.frame(key1_x = c(1, 2, 2),
                   key2_x = c("a", "a", "b"),
                   val_x = c("y1", "y2", "y3"))

> y2 <- y2 <- data.frame(key1_y = c(1, 2, 2, 3, 3),
                   key2_y = c("a", "a", "b", "a", "b"),
                   val_y = c("y1", "y2", "y3", "y4", "y5"))

> left_join(x2, y2, c("key1_x" = "key1_y"))

> left_join(x2, y2, c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

Structured Query Language

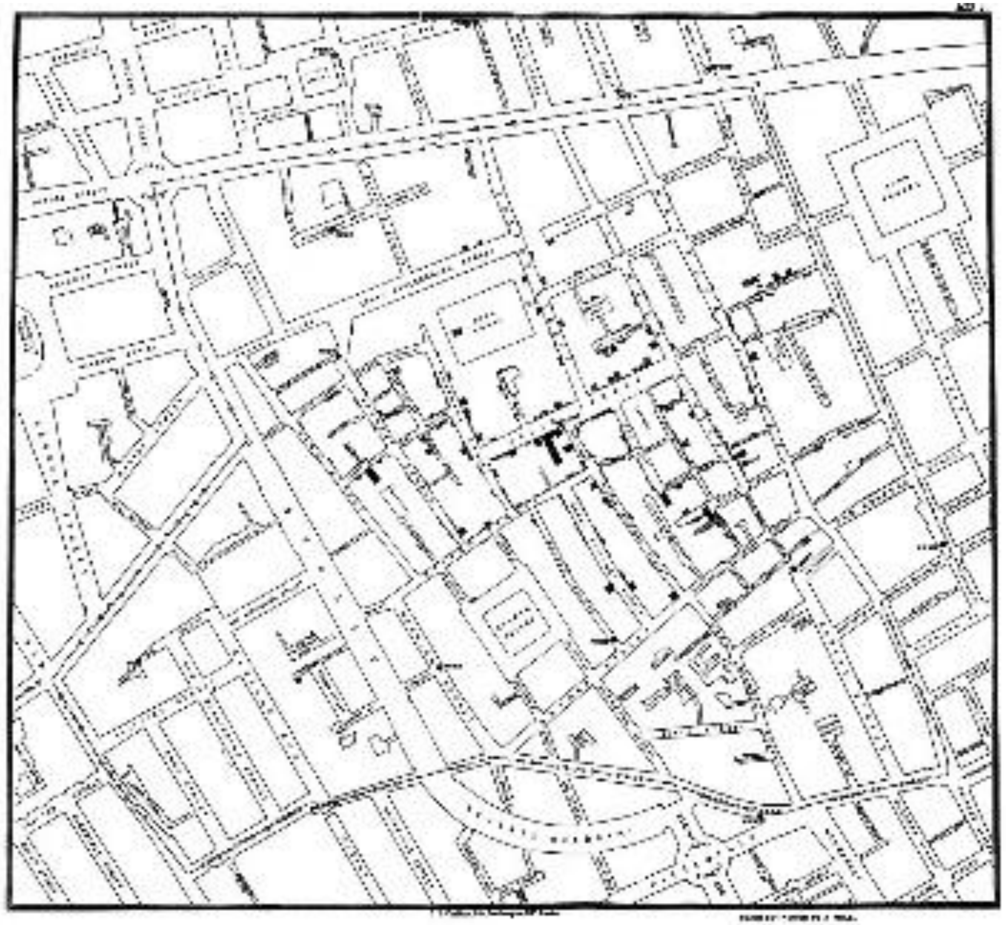
Having multiple tables that can be joined together is common in Relational Database Systems (RDBS).

- A common language used by RDBS is Structured Query Language (SQL)

dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

Let’s try it in R…

Spatial mapping

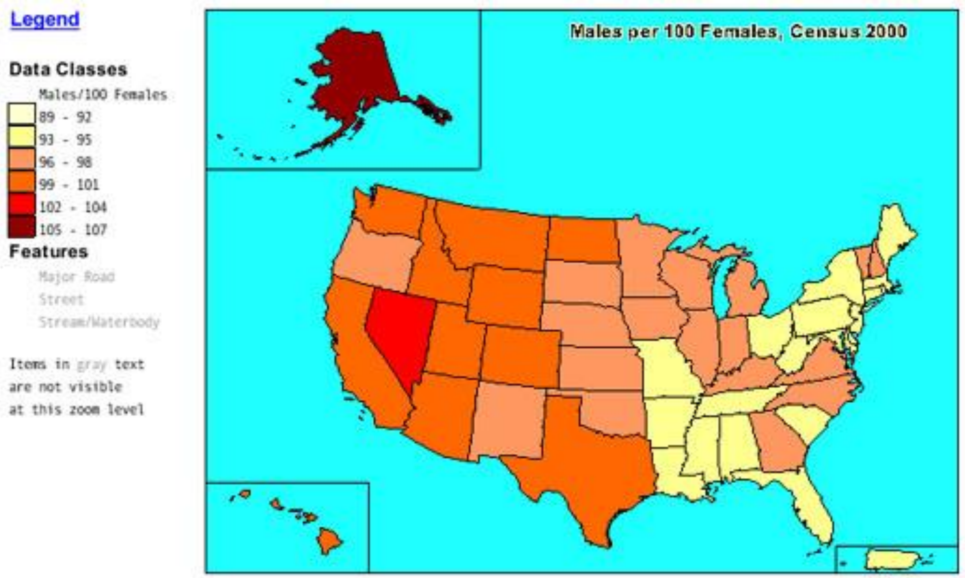


Maps

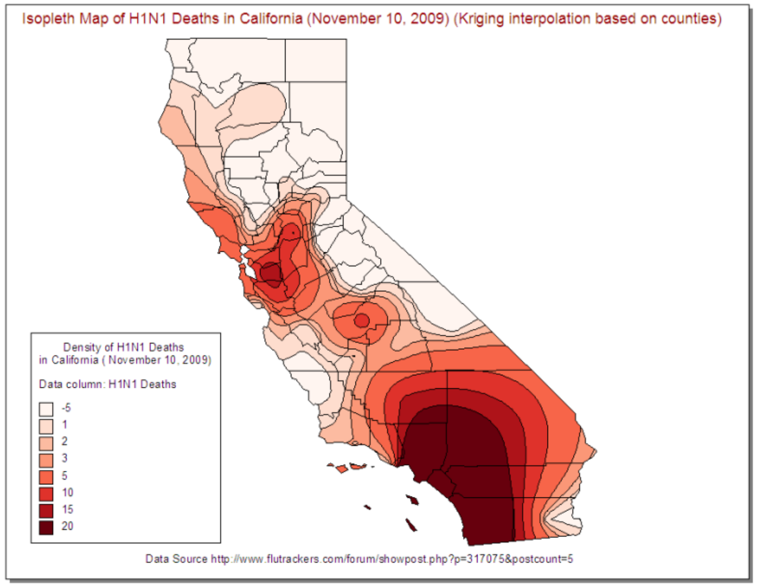
Choropleth maps: shades/colors in predefined areas based on properties of a variable

Isopleth maps: creates regions based on constant values

Choropleth map



Isopleth map



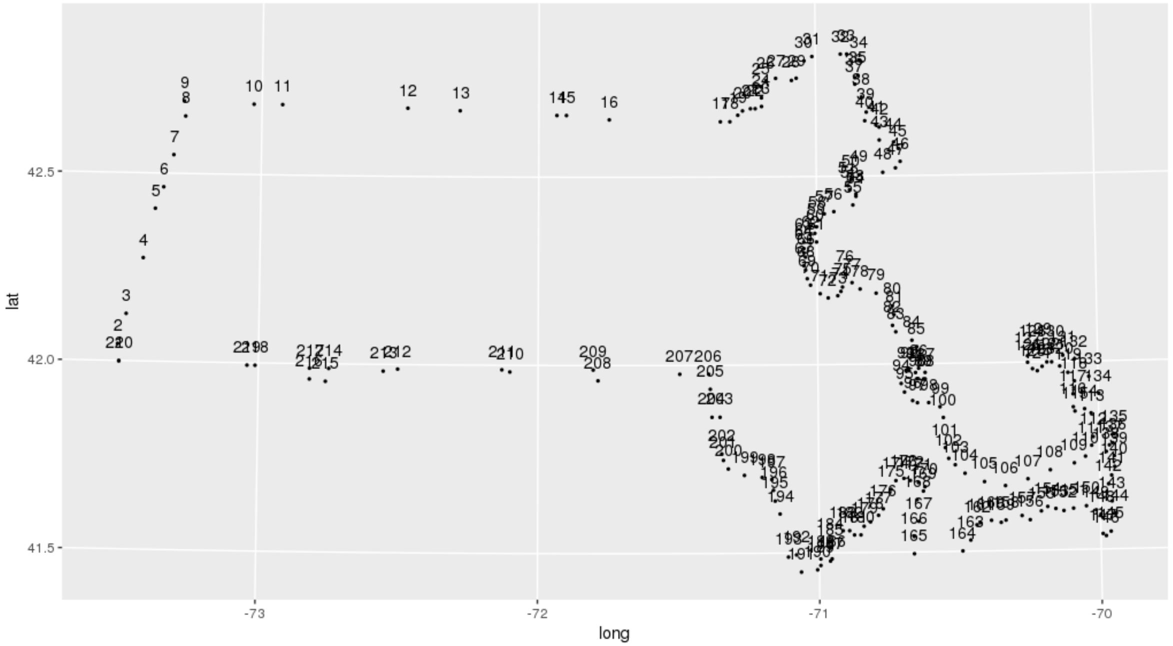
Choropleth maps

```
# has the coordinates for several maps
> library('maps')

# get a data frame with coordinates of states
> states_map <- map_data("state")
```

	long	lat	group	order	region	subregion
1	-87.46201	30.38968	1	1	alabama	NA
2	-87.48493	30.37249	1	2	alabama	NA
3	-87.52503	30.37249	1	3	alabama	NA
4	-87.53076	30.33239	1	4	alabama	NA
5	-87.57087	30.32665	1	5	alabama	NA

Choropleth maps

geom_polygon() works by connecting the dots:



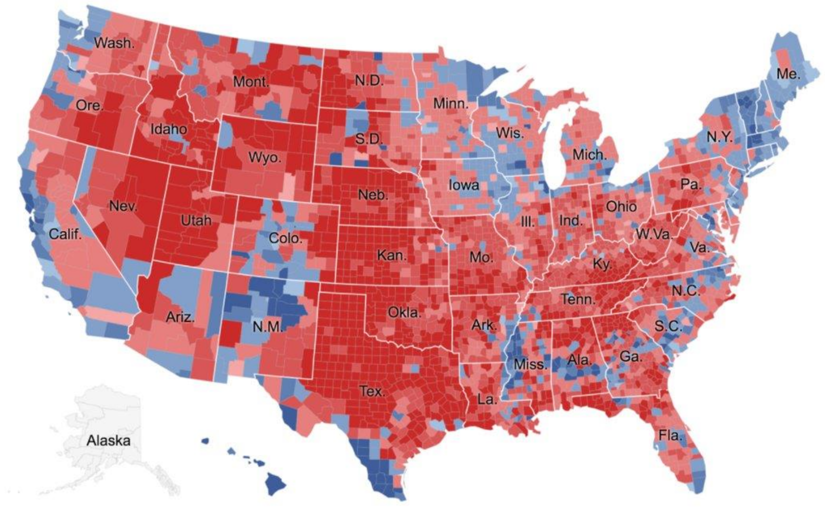
Often need to arrange points first: arrange(states_map, group, order)

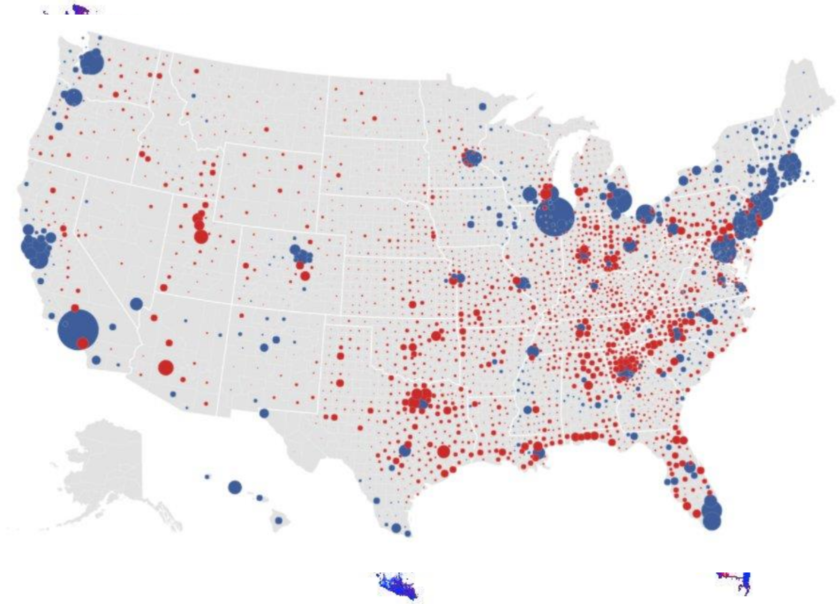
Choropleth maps

```
# has the coordinates for several maps
> library('maps')

# get a data frame with coordinates of states
> states_map <- map_data("state")

# filled white states with black borders
> ggplot(states_map,
        aes(x = long, y = lat, group = group)) +
        geom_polygon(fill = "white", color = "black")
```

Let’s try it in R!

Cloropleth maps can sometimes be misleading





Looks like most of the country
voted republican

Reshaping data

Wide vs. Long data

Plotting data using ggplot requires that data is in the right format

- i.e., requires data transformations.

Often this involves converting data from a **wide format** to **long format**

Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70

Long data

Person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

library(tidyr)

tidyr::pivot_longer()

pivot_longer(df, cols) converts data from **wide** to **long**

- Takes multiple columns and converts them into two columns: name and value
  - Column names become categorical variable levels of a new variable called **name**
  - The data in rows become entries in a variable called **value**

Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70

Long data

Person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

tidyr::pivot_wider()

pivot_wider(df, names_from, values_from) converts data from narrow to wide

- Turns the levels of categorical data into columns in a data frame

Narrow data

person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

→

Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70

Let’s try it in R!