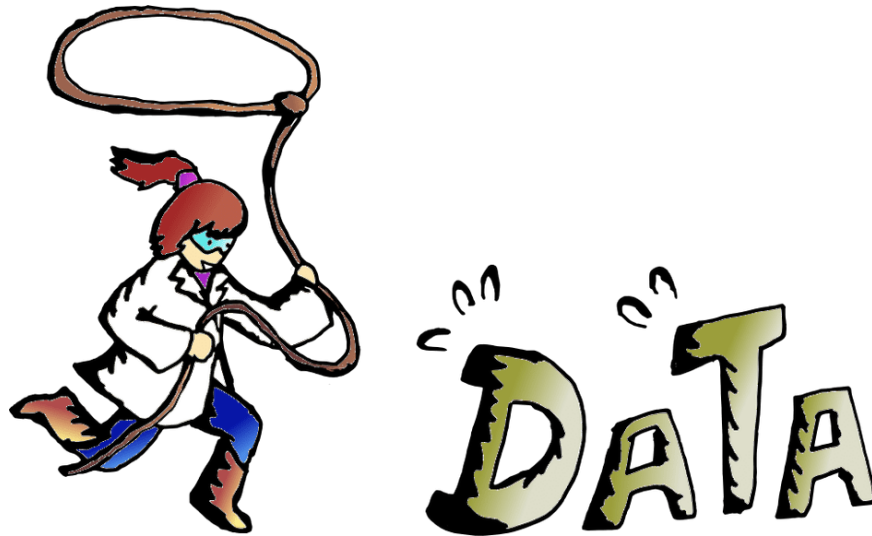


Joining, mapping, and reshaping data



Overview

Joining data frames continued

Creating maps

Reshaping

If there is time: string manipulation

Announcement

Homework 9 late date is set for Sunday November 28th

- i.e., the Sunday at the end of the Thanksgiving break

Regrade requests for older homework are open again until Sunday the 28th at 11pm

- They will not be opened again!

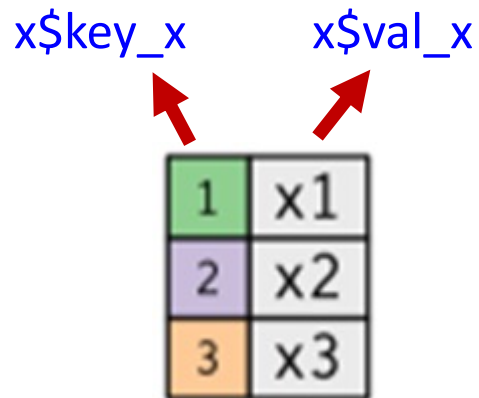
Keep thinking about your final project!

Joining data frames

Left and right tables

Suppose we have two data frames called x and y

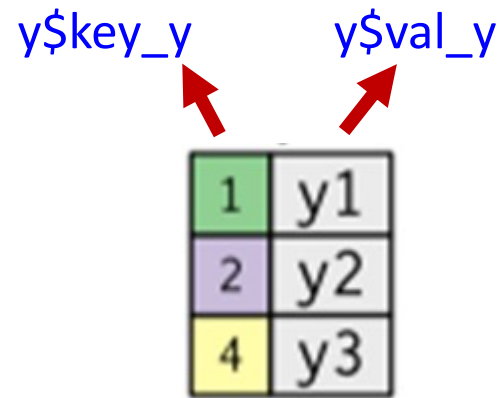
- x have two variables called `key_x`, and `val_x`
- y has two variables called `key_y` and `val_y`



A diagram of a data frame with two columns. The first column contains values 1, 2, and 3, each in a colored box (green, purple, orange). The second column contains values x1, x2, and x3, each in a grey box. Red arrows point from the labels 'x\$key_x' and 'x\$val_x' to the first and second columns respectively.

1	x1
2	x2
3	x3

Data frame x



A diagram of a data frame with two columns. The first column contains values 1, 2, and 4, each in a colored box (green, purple, yellow). The second column contains values y1, y2, and y3, each in a grey box. Red arrows point from the labels 'y\$key_y' and 'y\$val_y' to the first and second columns respectively.

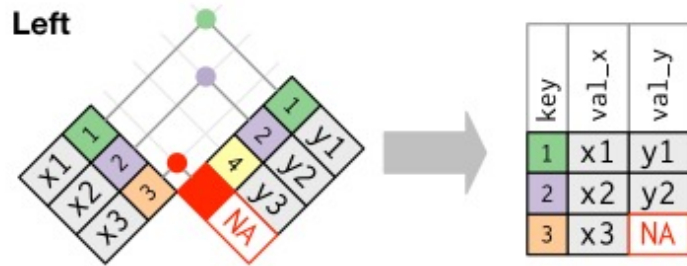
1	y1
2	y2
4	y3

Data frame y

Joins have the general form:

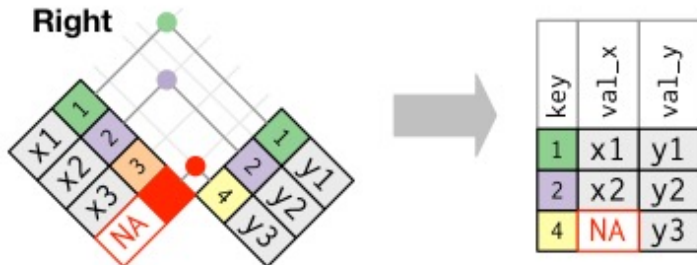
```
join(x, y, by = c("key_x" = "key_y"))
```

Joining data frames



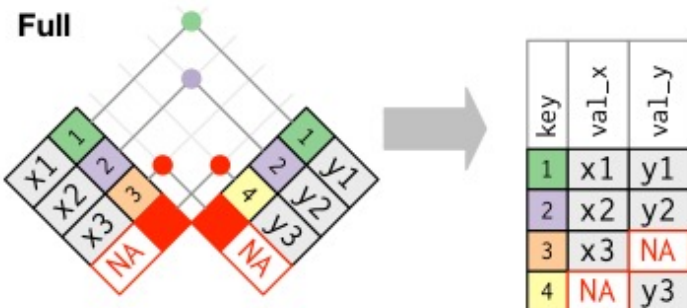
Left joins keep all rows in the left table.

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



Right joins keep all rows in the right table.

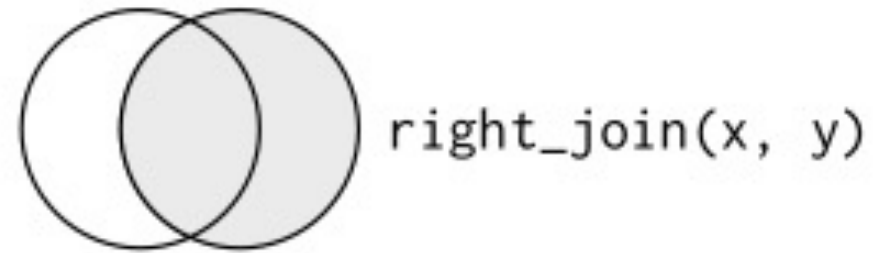
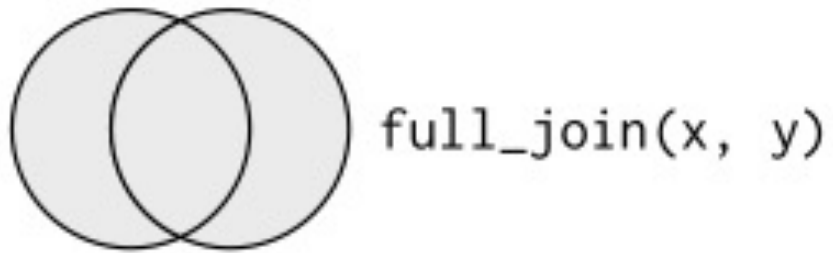
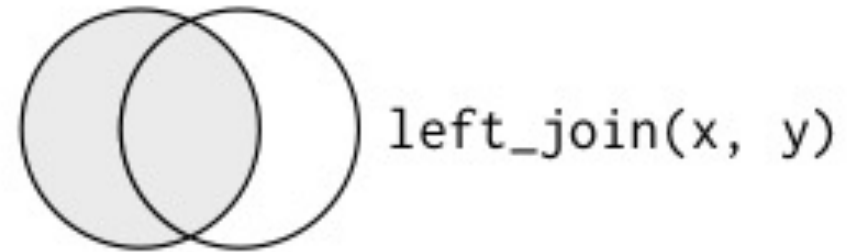
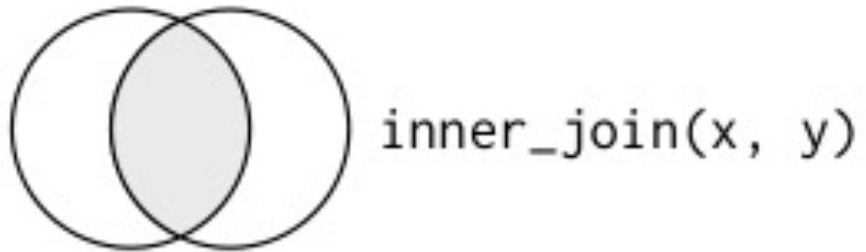
`right_join(x, y, by = c("key_x" = "key_y"))`



Full joins keep all rows in both tables.

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

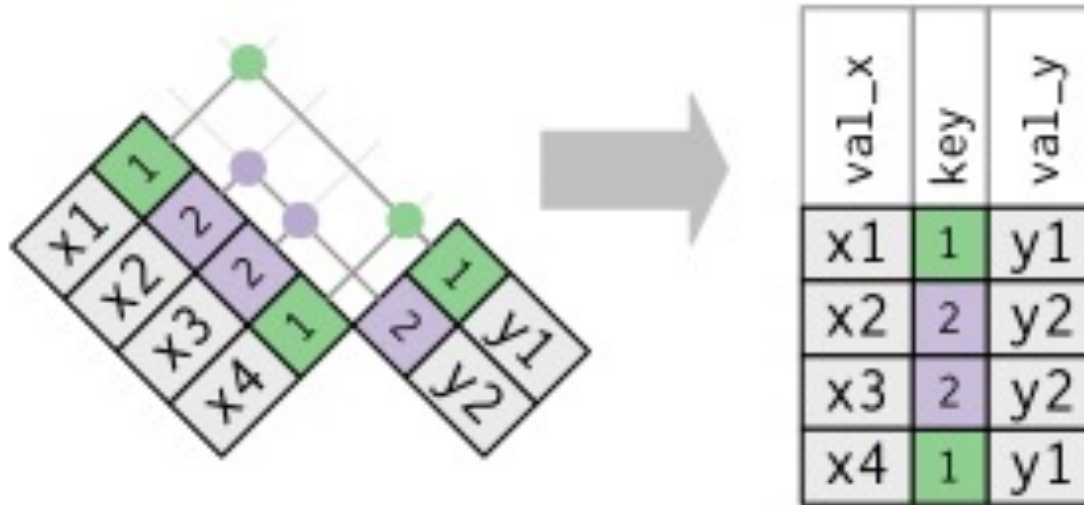
Summary



Duplicate keys

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

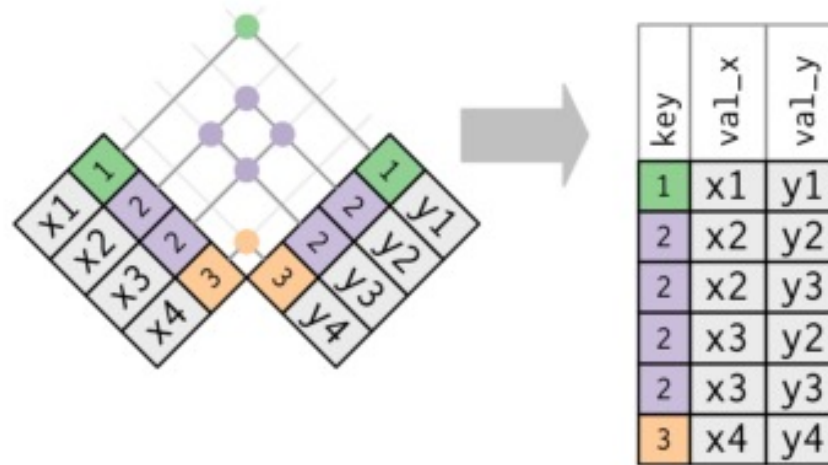
- e.g., duplicates are useful in the left table when doing a left join



Duplicate keys

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

- **This is usually an error!**



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 <- 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

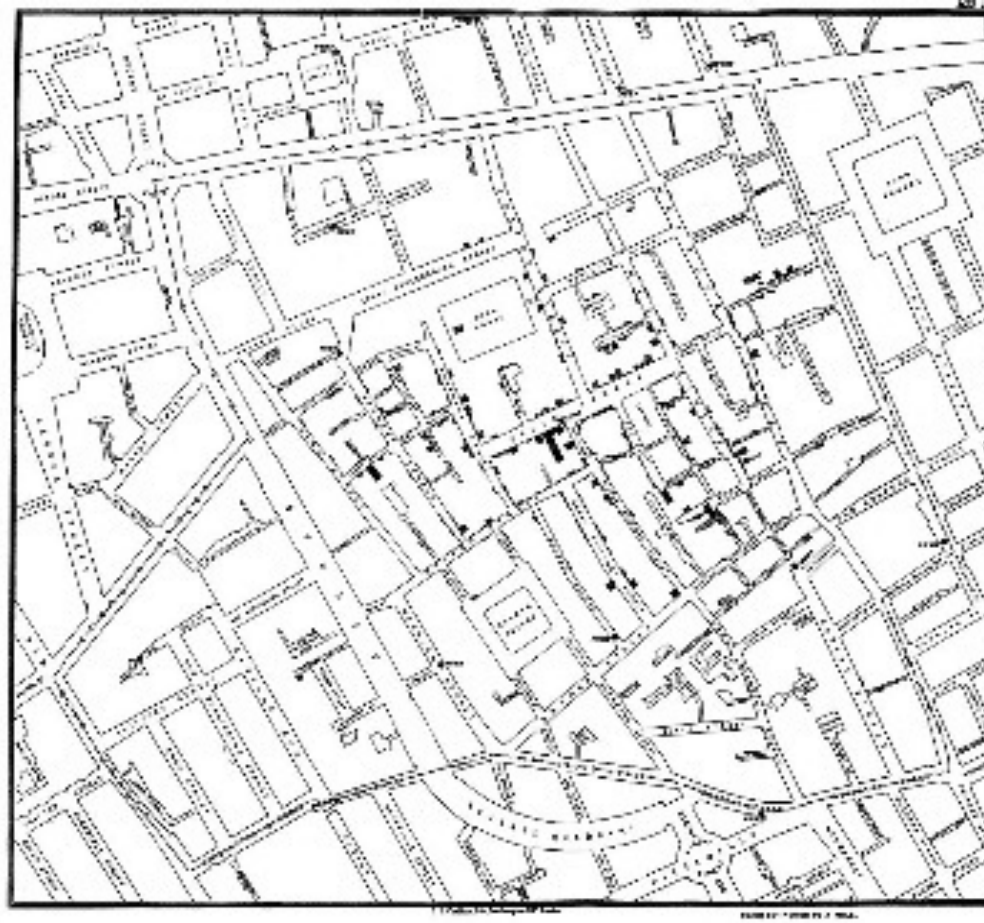
Having multiple tables that can be joined together is common in Relational Database Systems (RDBS).

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

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

Let's try it in R...

Spatial mapping

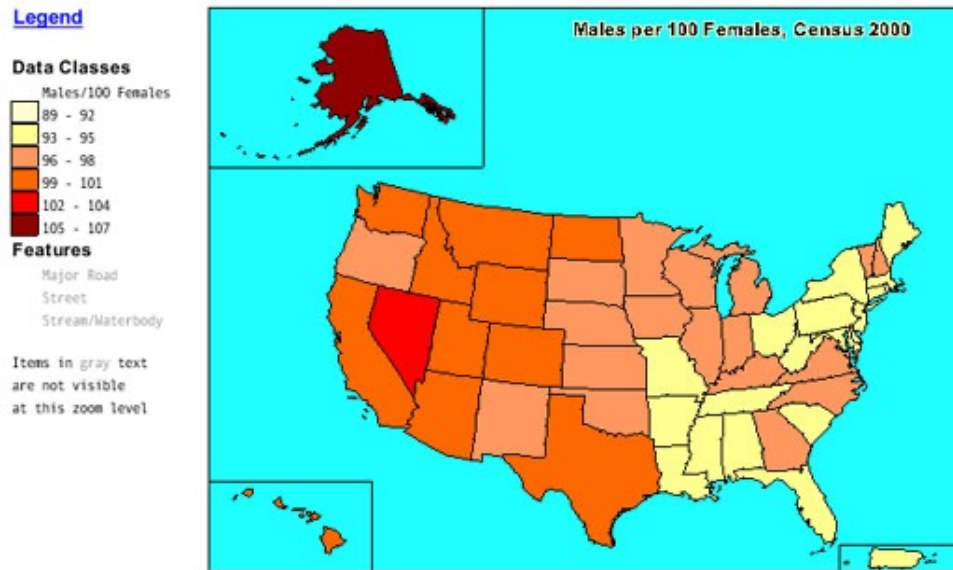


Maps

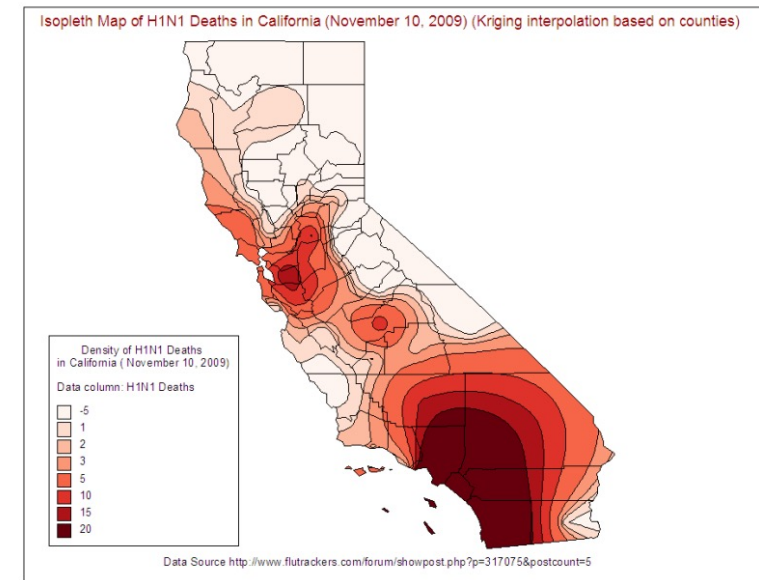
Choropleth maps: shades/colors in predefined areas based on properties of a variable

Isopleth maps: creates regions based on constant values

Choropleth map



Isopleth map



Choropleth maps

has the coordinates for several maps

```
> library('maps')
```

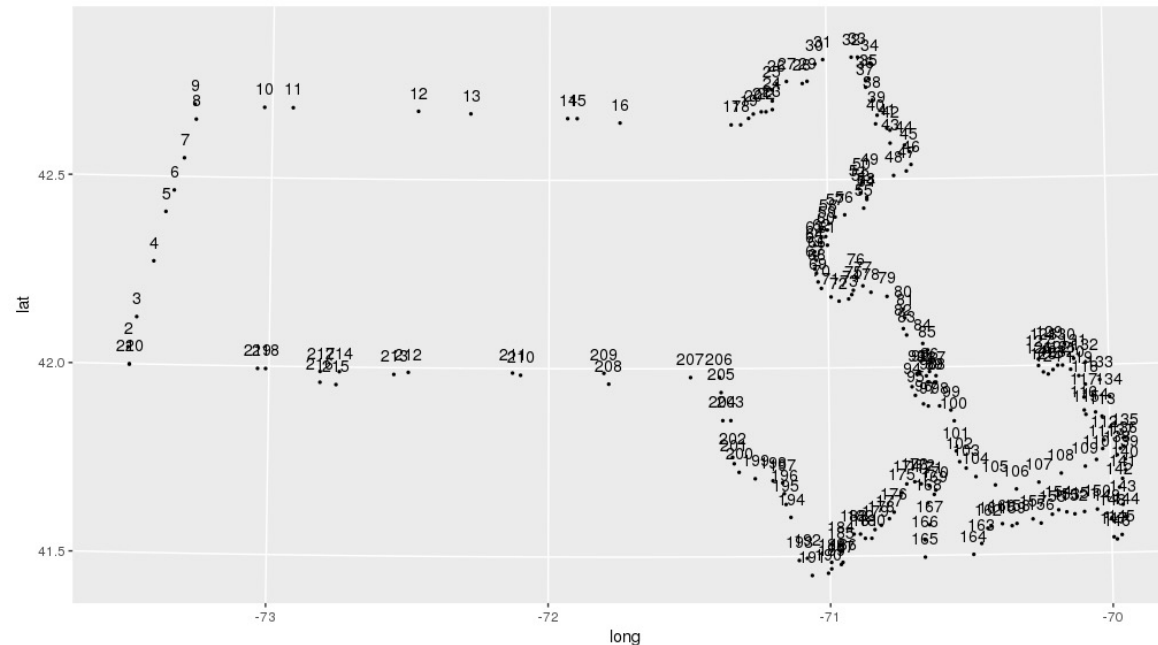
get a data frame with coordinates of states

```
> 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!



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

Narrow 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!