**Transforming Data with dplyr**

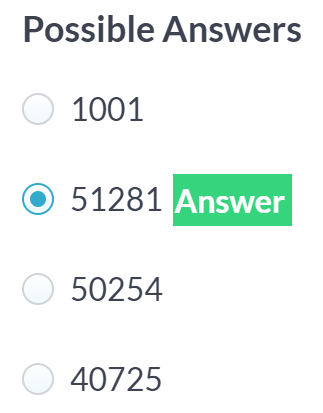
Learn verbs you can use to transform your data, including select, filter, arrange, and mutate. You'll use these functions to modify the counties dataset to view particular observations and answer questions about the data.

# Understanding your data

Take a look at the counties dataset using the glimpse() function.

What is the first value in the income variable?

****

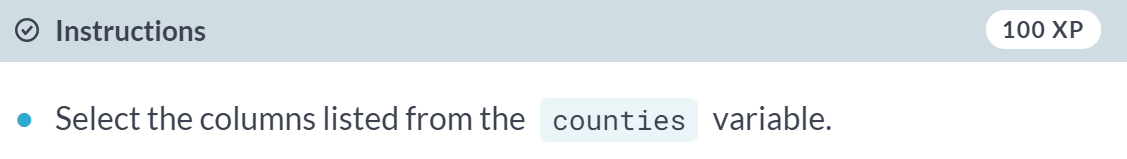
****

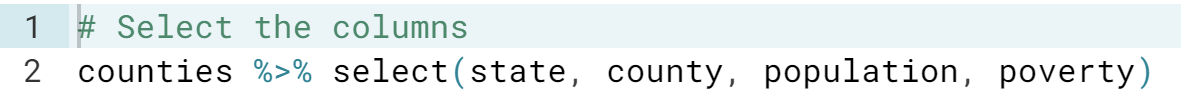
# Selecting columns

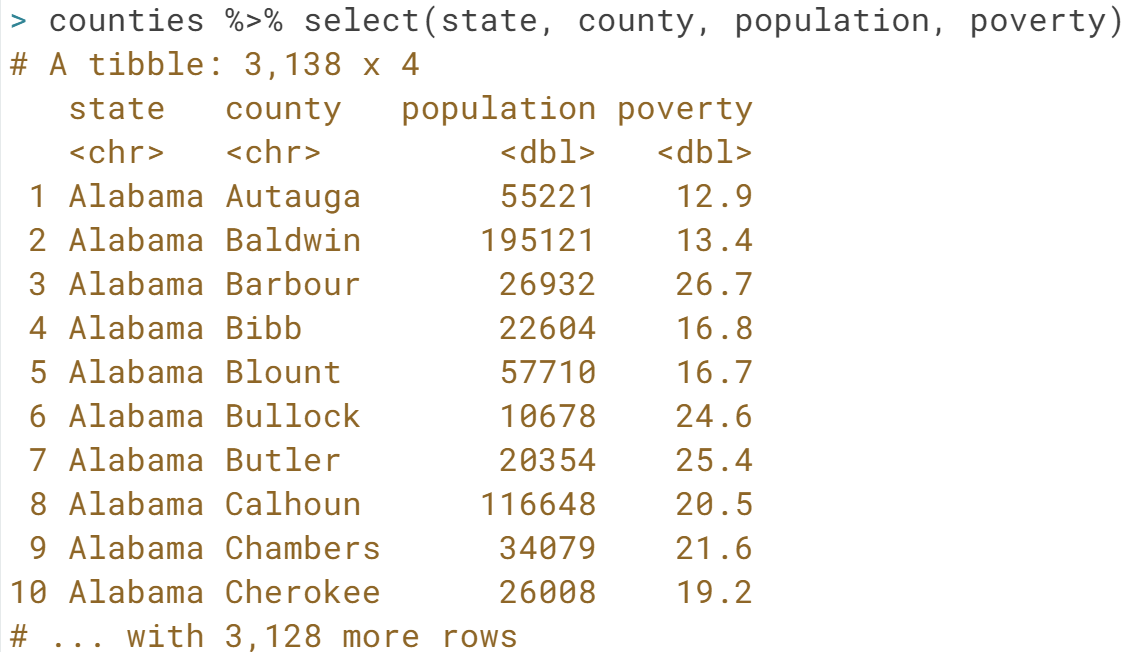
Select the following four columns from the counties variable:

* state
* county
* population
* poverty

You don't need to save the result to a variable.

****

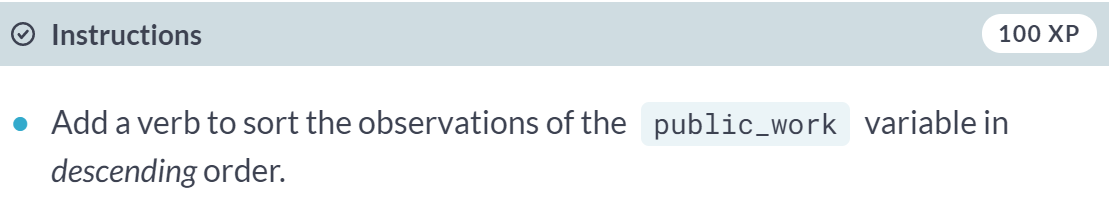
****

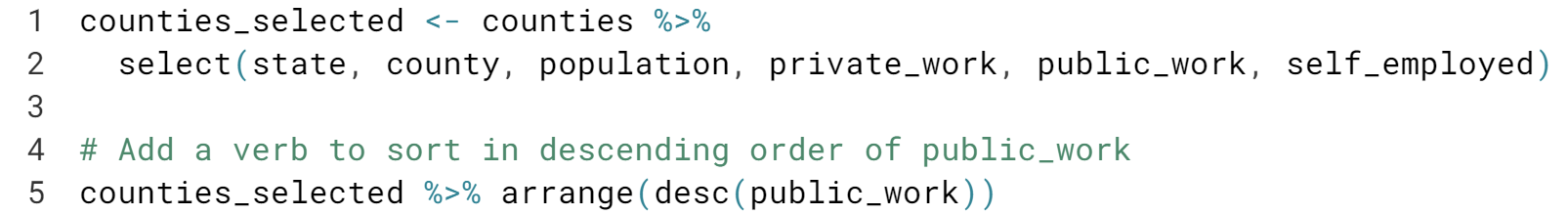
****

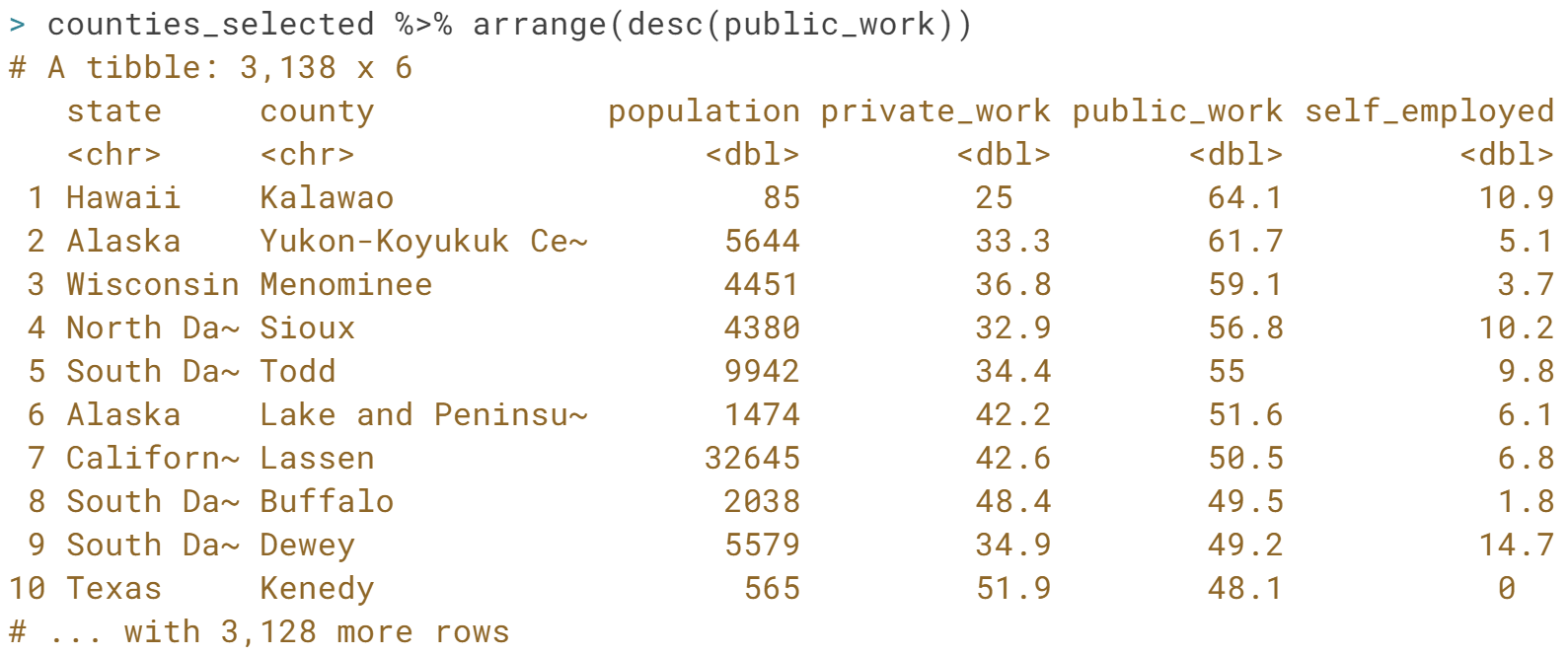
# Arranging observations

Here you see the counties\_selected dataset with a few interesting variables selected. These variables: private\_work, public\_work, self\_employed describe whether people work for the government, for private companies, or for themselves.

In these exercises, you'll sort these observations to find the most interesting cases.

****

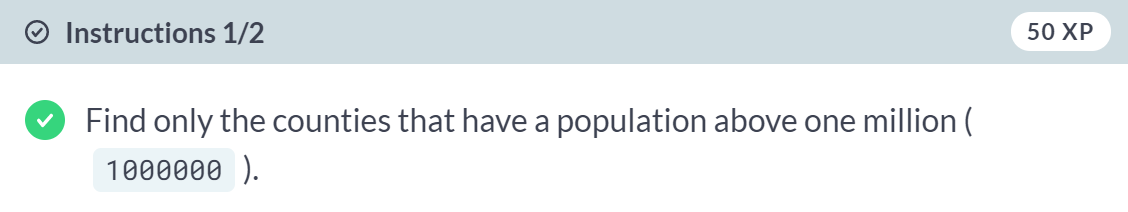
****

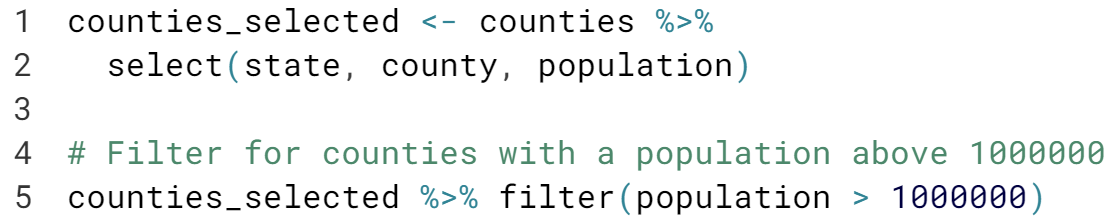
****

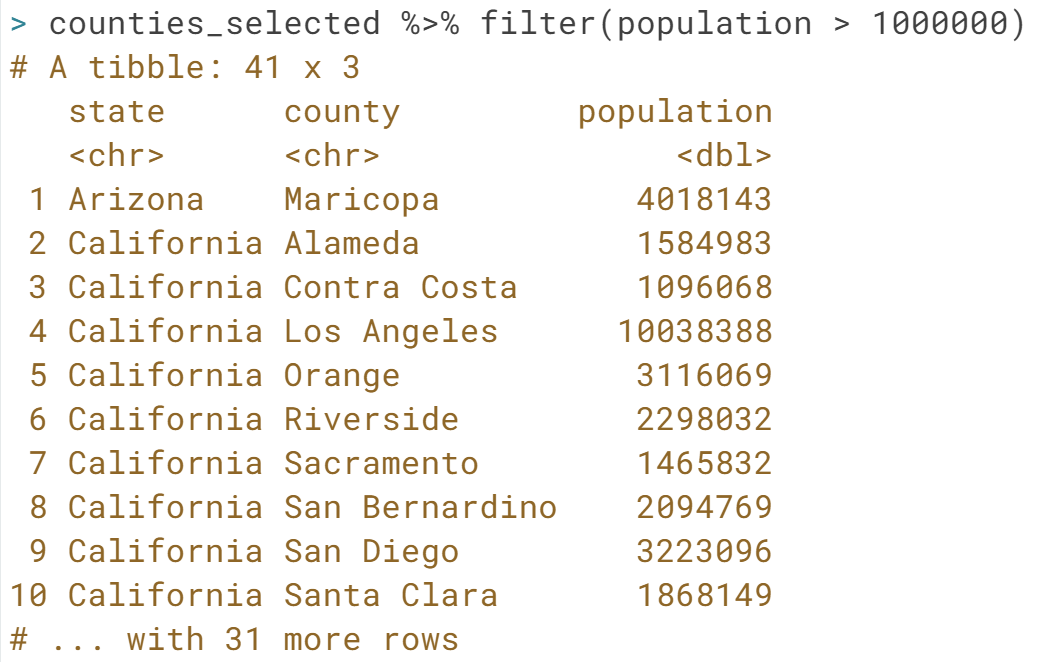
Great! We sorted the counties in descending order according to public\_work. What if we were interested in looking at observations in counties that have a large population or within a specific state? Head over to the next exercise.

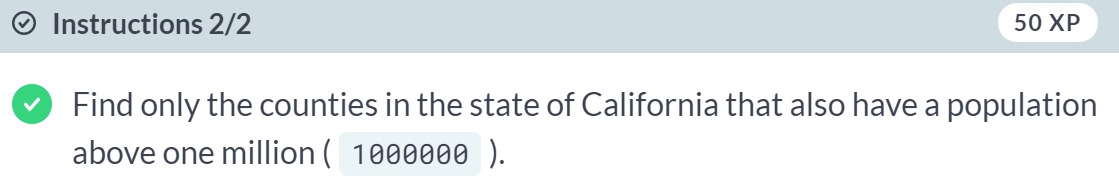
# Filtering for conditions

You use the filter() verb to get only observations that match a particular condition, or match multiple conditions.

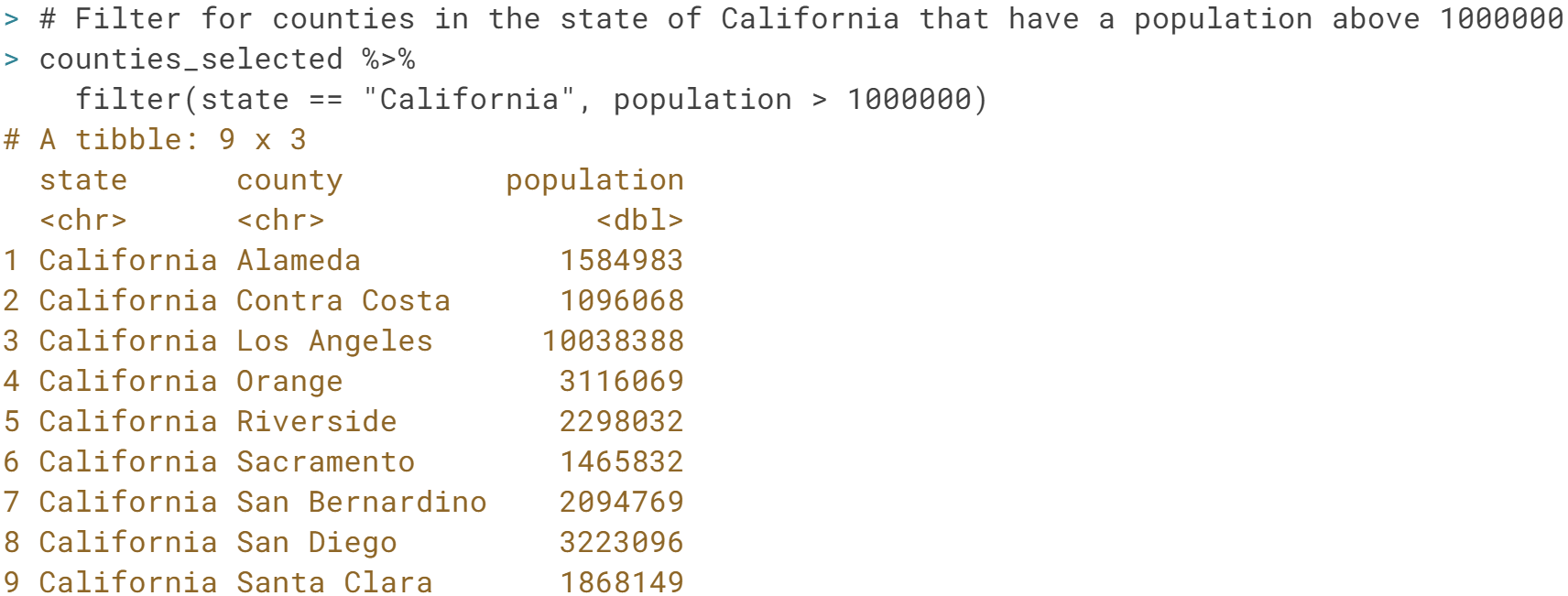




****

****

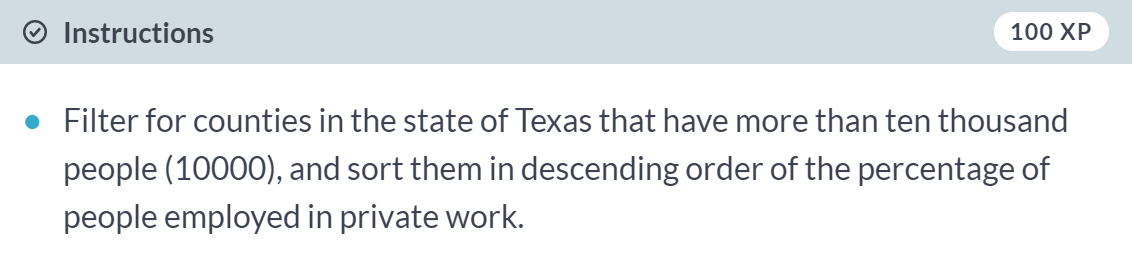
****

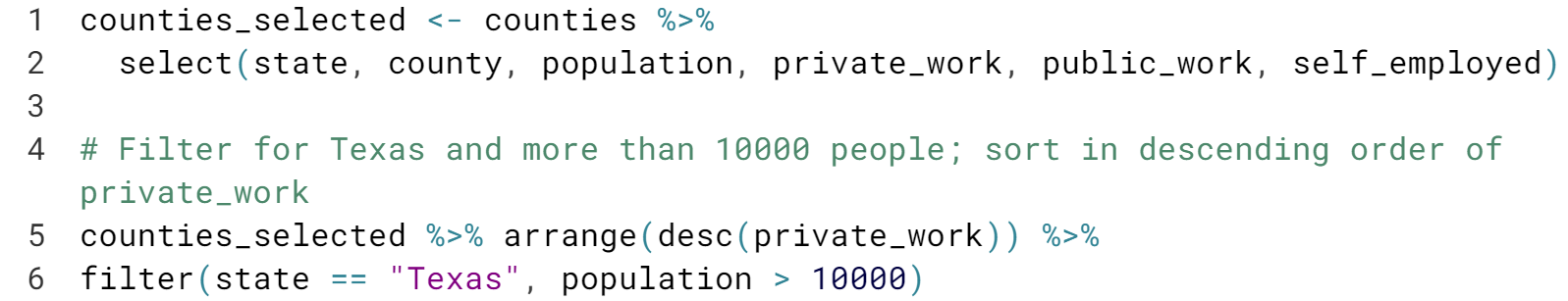
****

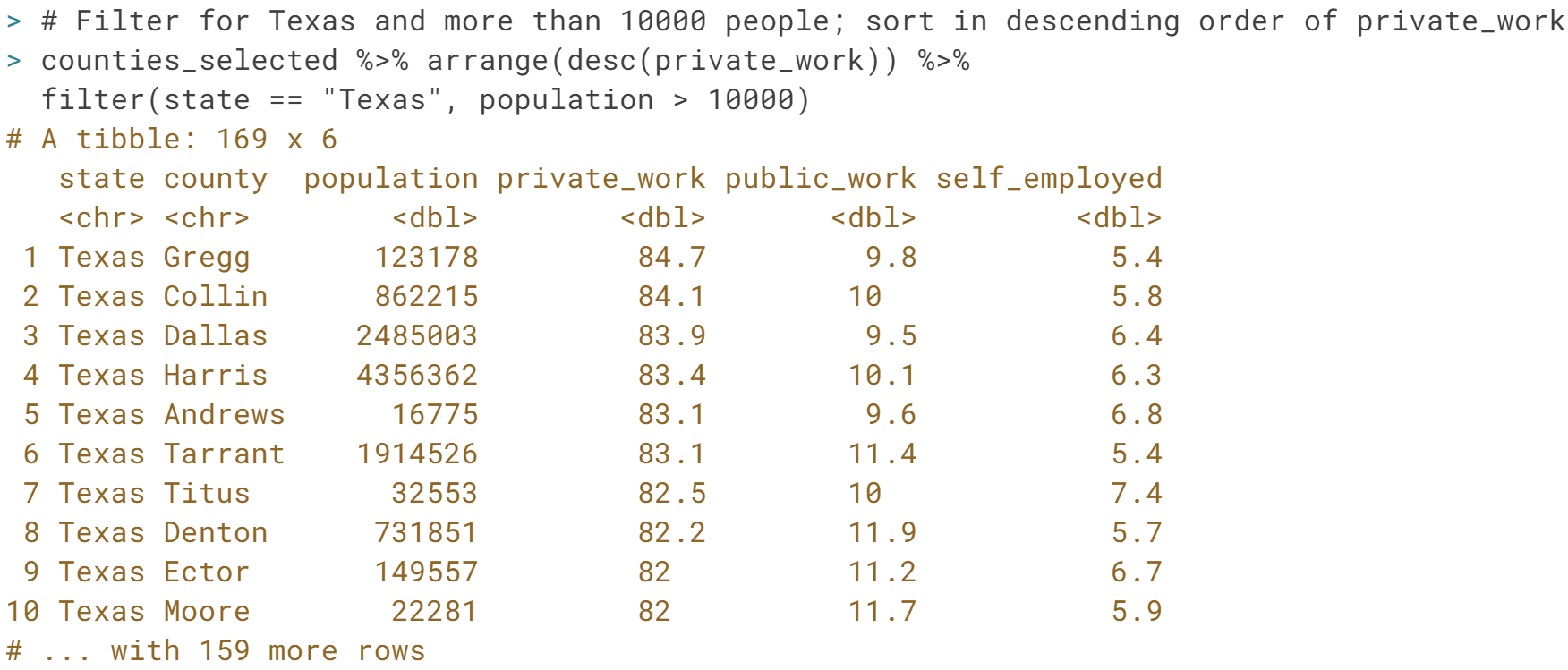
Good work! Now you know that there are 9 counties in the state of California with a population greater than one million. In the next exercise, you'll practice filtering and then sorting a dataset to focus on specific observations.

# Filtering and arranging

We're often interested in both filtering and sorting a dataset, to focus on observations of particular interest to you. Here, you'll find counties that are extreme examples of what fraction of the population works in the private sector.





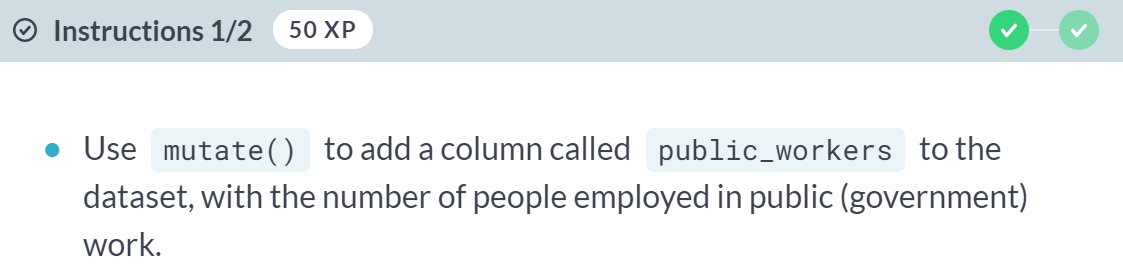


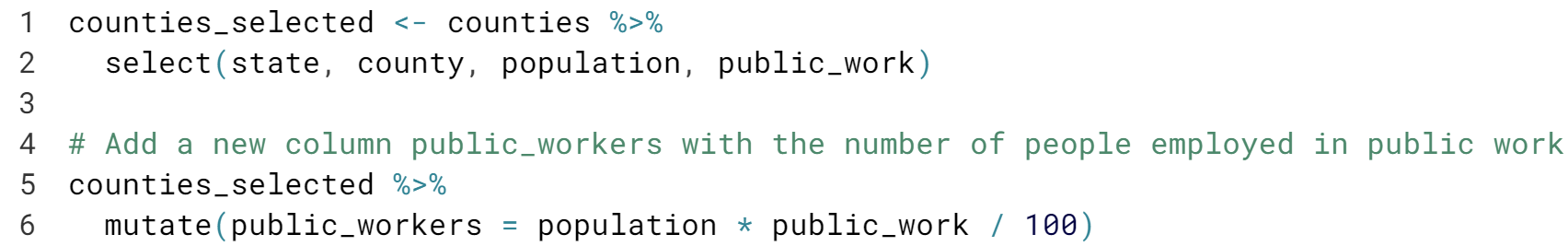
Awesome! You've learned how to filter and sort a dataset to answer questions about the data. Notice that you only need to slightly modify your code if you are interested in sorting the observations by a different column.

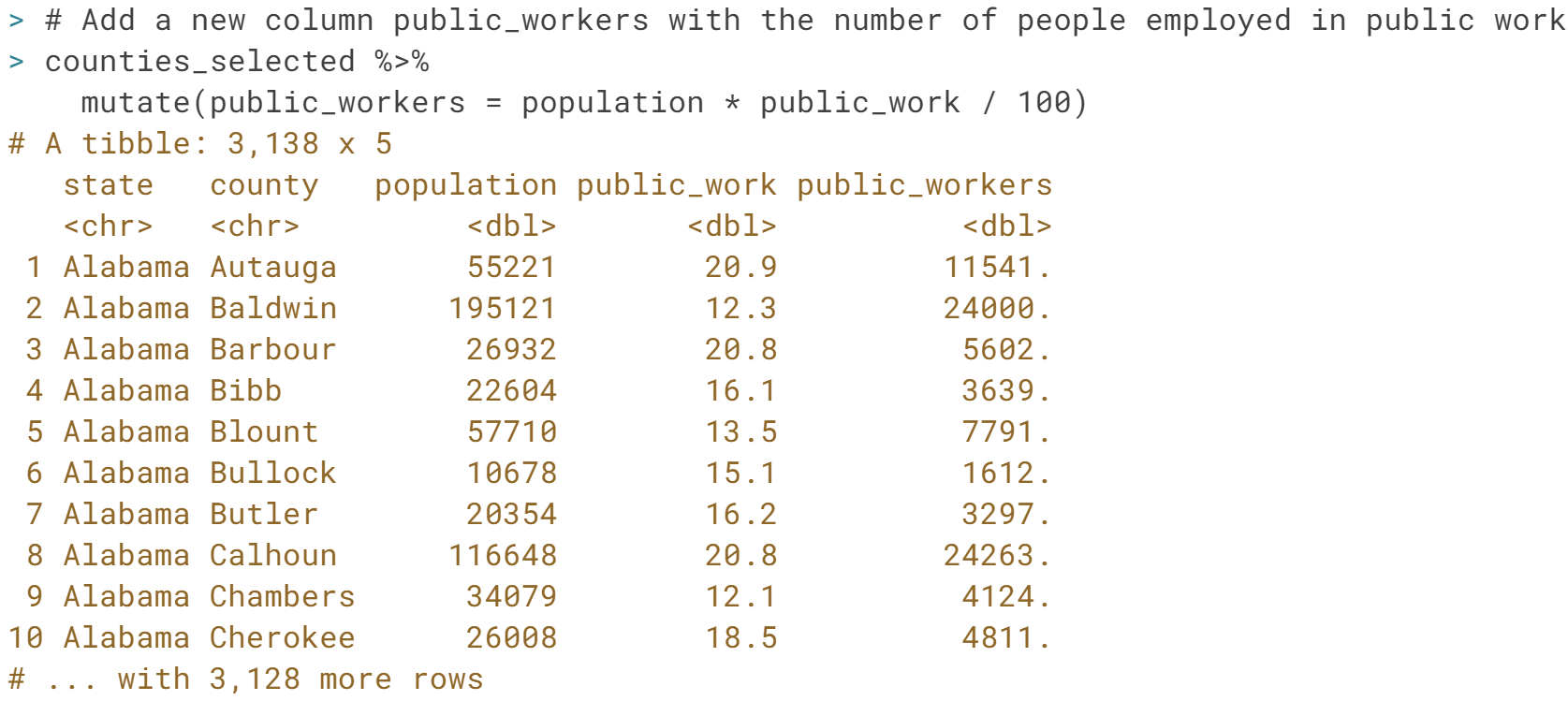
# Calculating the number of government employees

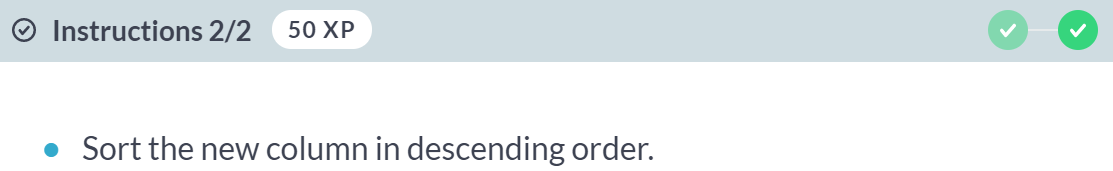
In the video, you used the unemployment variable, which is a percentage, to calculate the number of unemployed people in each county. In this exercise, you'll do the same with another percentage variable: public\_work.

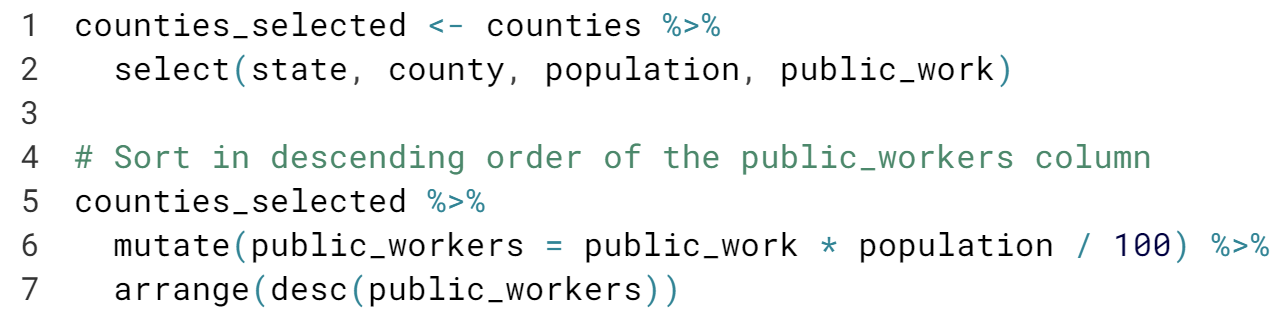
The code provided already selects the state, county, population, and public\_work columns.

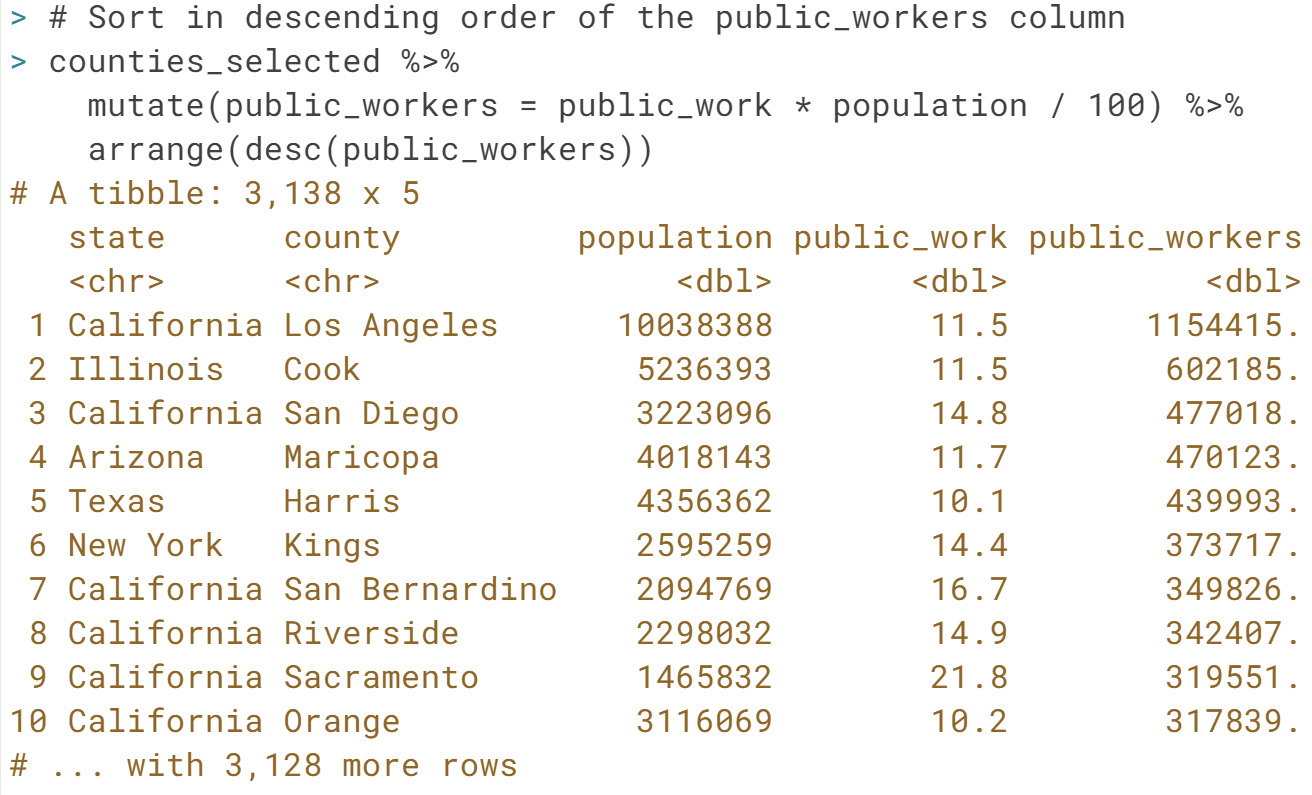










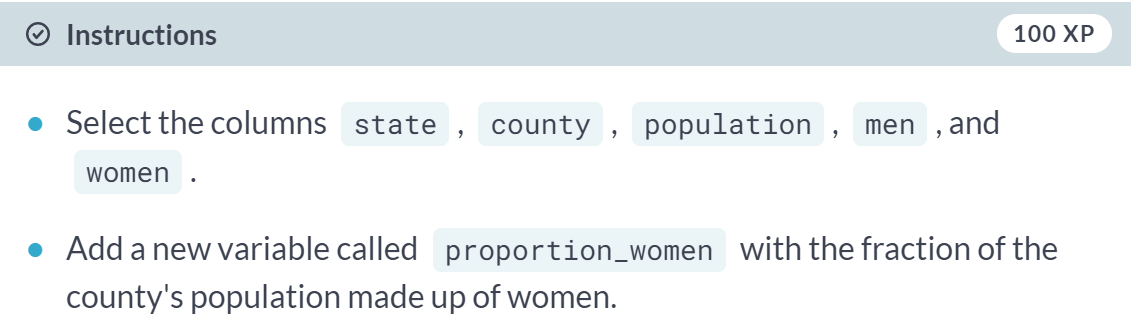


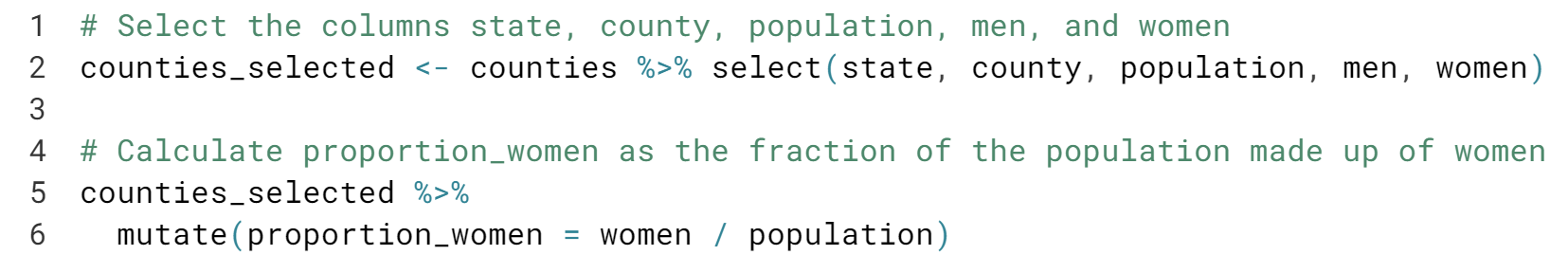
Great work! It looks like Los Angeles is the county with the most government employees.

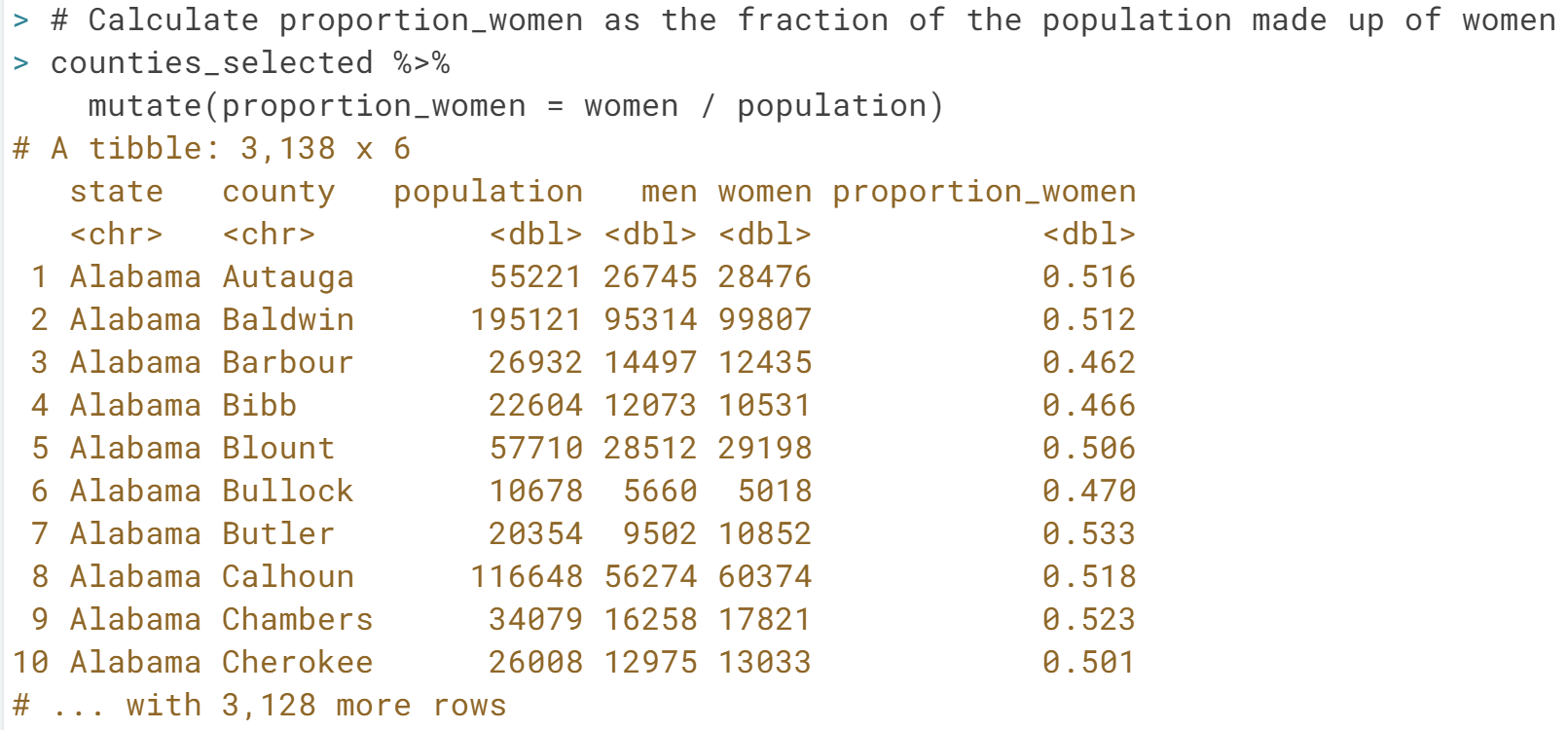
# Calculating the percentage of women in a county

The dataset includes columns for the total number (not percentage) of men and women in each county. You could use this, along with the population variable, to compute the fraction of men (or women) within each county.

In this exercise, you'll select the relevant columns yourself.

****

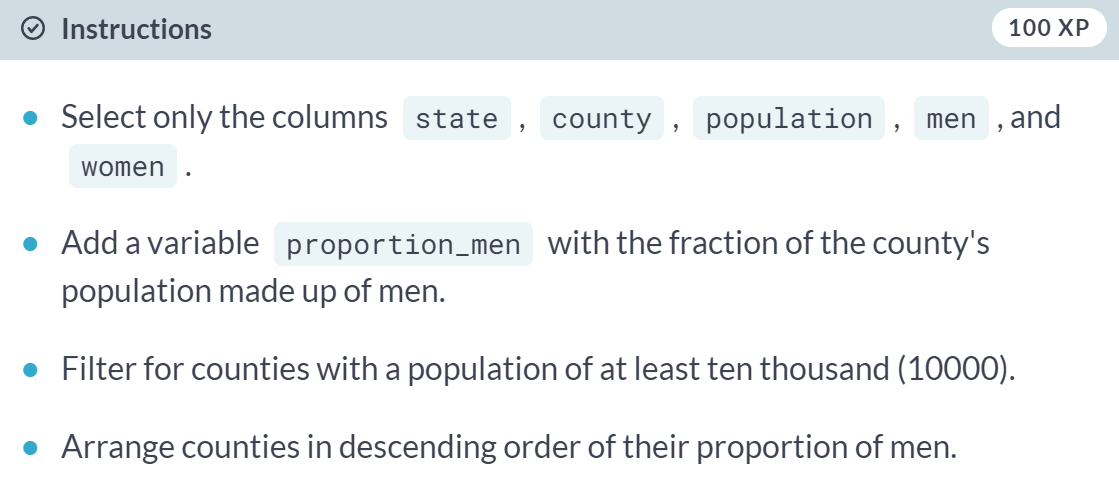
****

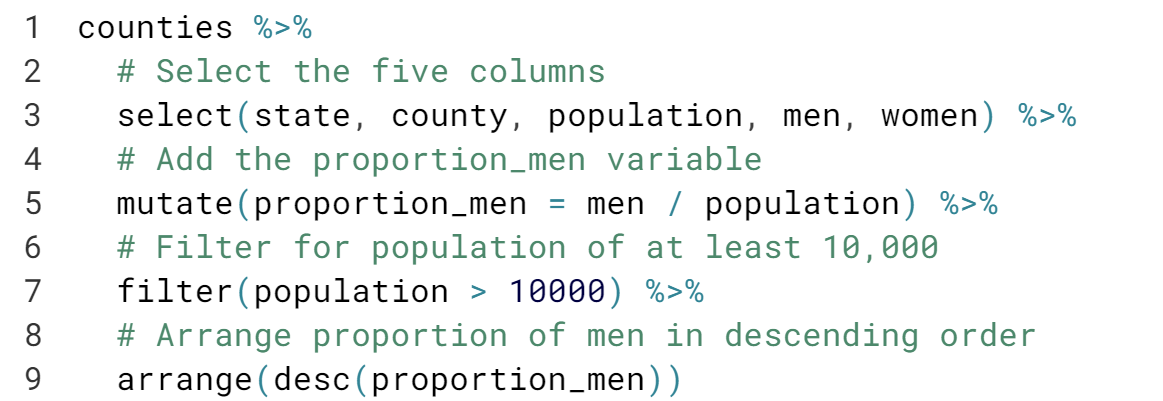
****

Good job! Notice that the ***proportion\_women*** variable was added as a column to the ***counties\_selected*** dataset, and the data now has 6 columns instead of 5.

# Select, mutate, filter, and arrange

In this exercise, you'll put together everything you've learned in this chapter (select(), mutate(), filter() and arrange()), to find the counties with the highest proportion of men.

****

****

****

Right! Notice Sussex County in Virginia is more than two thirds male: this is because of two men's prisons in the county.

#### Aggregating Data

#### Now that you know how to transform your data, you'll want to know more about how to aggregate your data to make it more interpretable. You'll learn a number of functions you can use to take many observations in your data and summarize them, including count, group\_by, summarize, ungroup, and top\_n.

# Counting by region

The counties dataset contains columns for region, state, population, and the number of citizens, which we selected and saved as the counties\_selected table. In this exercise, you'll focus on the region column.

counties\_selected <- counties %>%

select(region, state, population, citizens)

#### 

#### 

#### 

#### Good job! Since the results have been arranged, you can see that the South has the greatest number of counties.

# Counting citizens by state

You can weigh your count by particular variables rather than finding the number of counties. In this case, you'll find the number of citizens in each state.

counties\_selected <- counties %>%

select(region, state, population, citizens)

#### 

#### 

#### 

#### Great! From our result, we can see that California is the state with the most citizens.

# Mutating and counting

You can combine multiple verbs together to answer increasingly complicated questions of your data. For example: "What are the US states where the most people walk to work?"

You'll use the walk column, which offers a percentage of people in each county that walk to work, to add a new column and count based on it.

counties\_selected <- counties %>%

select(region, state, population, walk)

#### 

#### 

#### 

#### Great! We can see that while California had the largest total population, New York state has the largest number of people who walk to work.

# Summarizing

The summarize() verb is very useful for collapsing a large dataset into a single observation.

counties\_selected <- counties %>%

select(county, population, income, unemployment)

#### 

#### 

#### 

#### Good work! If we wanted to take this a step further, we could use *filter()* to determine the specific counties that returned the value for *min\_population* and *max\_unemployment*.

# Summarizing by state

Another interesting column is land\_area, which shows the land area in square miles. Here, you'll summarize both population and land area by state, with the purpose of finding the density (in people per square miles).

counties\_selected <- counties %>%

select(state, county, population, land\_area)

#### 

#### 

#### 

#### 

#### 

#### 

#### Great work! Looks like New Jersey and Rhode Island are the “most crowded” of the US states, with more than a thousand people per square mile.

# Summarizing by state and region

You can group by multiple columns instead of grouping by one. Here, you'll practice aggregating by state and region, and notice how useful it is for performing multiple aggregations in a row.

counties\_selected <- counties %>%

select(region, state, county, population)

#### 

#### 

#### 

#### 

#### 

#### 

#### Great! It looks like the South has the highest *average\_pop* of 7370486, while the North Central region has the highest *median\_pop* of 5580644.

# Selecting a county from each region

Previously, you used the walk column, which offers a percentage of people in each county that walk to work, to add a new column and count to find the total number of people who walk to work in each county.

Now, you're interested in finding the county **within each region** with the highest percentage of citizens who walk to work.

counties\_selected <- counties %>%

select(region, state, county, metro, population, walk)

#### 

#### 

#### 

#### Great! Notice that three of the places lots of people walk to work are low-population nonmetro counties, but that New York City also pops up!

# Finding the highest-income state in each region

You've been learning to combine multiple dplyr verbs together. Here, you'll combine group\_by(), summarize(), and top\_n() to find the state in each region with the highest income.

When you group by multiple columns and then summarize, it's important to remember that the summarize "peels off" one of the groups, but leaves the rest on. For example, if you group\_by(X, Y) then summarize, the result will still be grouped by X.

counties\_selected <- counties %>%

select(region, state, county, population, income)

#### 

#### 

#### 

#### Good work! From our results, we can see that the New Jersey in the Northeast is the state with the highest *average\_income* of 73014.

# Using summarize, top\_n, and count together

In this chapter, you've learned to use five dplyr verbs related to aggregation: count(), group\_by(), summarize(), ungroup(), and top\_n(). In this exercise, you'll use all of them to answer a question: **In how many states do more people live in metro areas than non-metro areas?**

Recall that the metro column has one of the two values "Metro" (for high-density city areas) or "Nonmetro" (for suburban and country areas).

counties\_selected <- counties %>%

select(state, metro, population)

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### Way to go! Notice that 44 states have more people living in Metro areas, and 6 states have more people living in Nonmetro areas.

#### Combination of Instructions 1 to 3

#### 

#### Selecting and Transforming Data

#### Learn advanced methods to select and transform columns. Also learn about select helpers, which are functions that specify criteria for columns you want to choose, as well as the rename and transmute verbs.

# Selecting columns

Using the select verb, we can answer interesting questions about our dataset by focusing in on related groups of verbs. The colon (:) is useful for getting many columns at a time.

#### 

#### 

#### 

#### Great! Notice that when you select a group of related variables, it's easy to find the insights you're looking for.

# Select helpers

In the video you learned about the select helper starts\_with(). Another select helper is ends\_with(), which finds the columns that end with a particular string.

#### 

#### 

#### 

#### Good job! It looks like only a few counties have more than half the population working for the government.

# Renaming a column after count

The rename() verb is often useful for changing the name of a column that comes out of another verb, such as count(). In this exercise, you'll rename the n column from count() (which you learned about in Chapter 2) to something more descriptive.

#### 

#### 

#### 

#### 

#### 

#### 

#### Good work! Notice the difference between column names in the output from the first step to the second step. Don't forget, using *rename()* isn't the only way to choose a new name for a column!

# Renaming a column as part of a select

rename() isn't the only way you can choose a new name for a column: you can also choose a name as part of a select().

#### 

#### 

#### 

#### Great! As you can see, we were able to select the four columns of interest from our dataset, and rename one of those columns, using only the *select()* verb!

# Choosing among verbs

Which of the following verbs would you use to calculate new columns while dropping other columns?

#### 

# Using transmute

As you learned in the video, the transmute verb allows you to control which variables you keep, which variables you calculate, and which variables you drop.

#### 

#### 

#### 

#### Great work! Looks like San Bernadino is the lowest density county with a population about one million.

#### Matching verbs to their definitions

We've learned a number of new verbs in this chapter that you can use to modify and change the variables you have.

#### 

#### 

#### 

# Choosing among the four verbs

In this chapter you've learned about the four verbs: select, mutate, transmute, and rename. Here, you'll choose the appropriate verb for each situation. You will not need to change anything inside the parentheses.

#### 

#### 

#### 

#### 

#### 

#### 

#### Great! Now you know which variable to choose depending on whether you want to keep, drop, rename, or change a variable in the dataset.

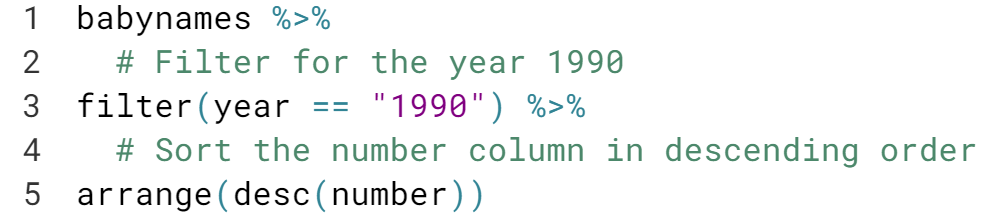
#### Case Study: The babynames Dataset

#### Work with a new dataset that represents the names of babies born in the United States each year. Learn how to use grouped mutates and window functions to ask and answer more complex questions about your data. And use a combination of dplyr and ggplot2 to make interesting graphs to further explore your data.

# Filtering and arranging for one year

The dplyr verbs you've learned are useful for exploring data. For instance, you could find out the most common names in a particular year.

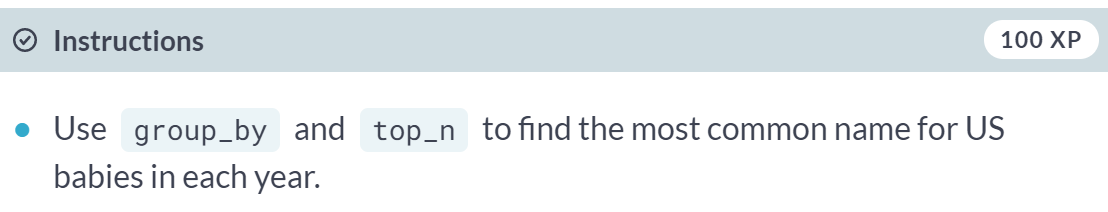


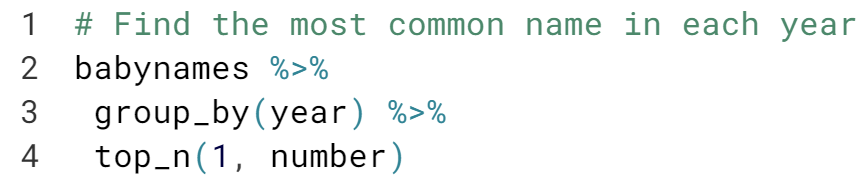


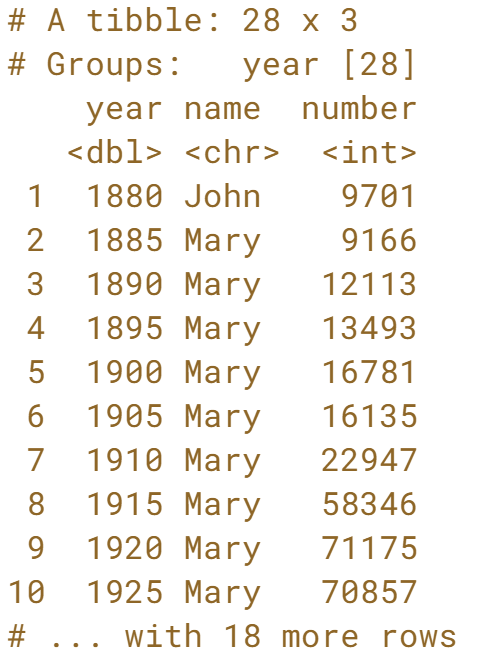
  
Great work! It looks like the most common names for babies born in the US in 1990 were Michael, Christopher, and Jessica.

# Using top\_n with babynames

You saw that you could use filter() and arrange() to find the most common names in one year. However, you could also use group\_by and top\_n to find the most common name in every year.



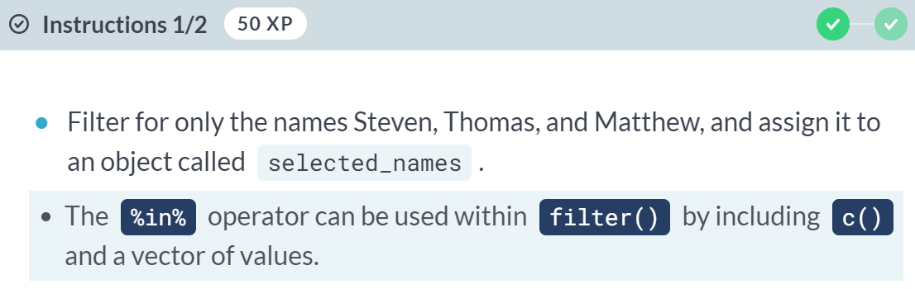


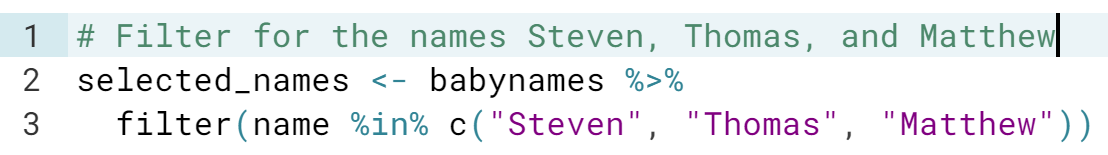


Good work! It looks like John was the most common name in 1880, and Mary was the most common name for a while after that.

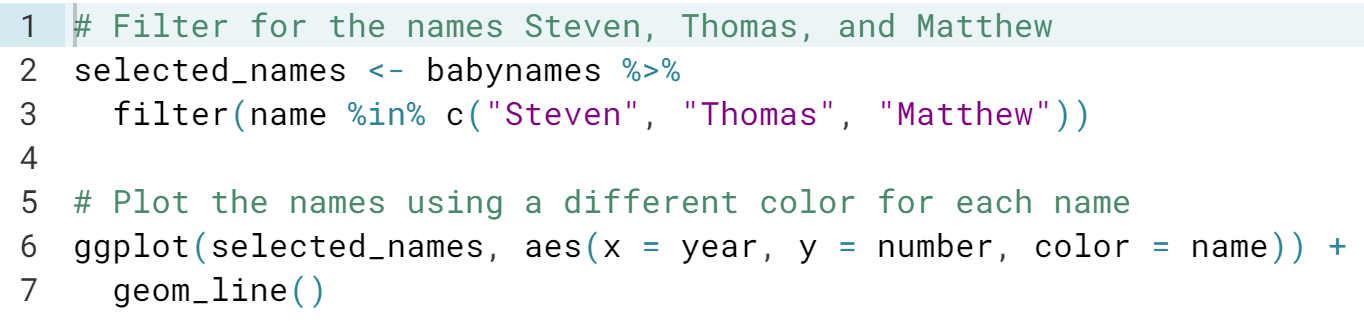
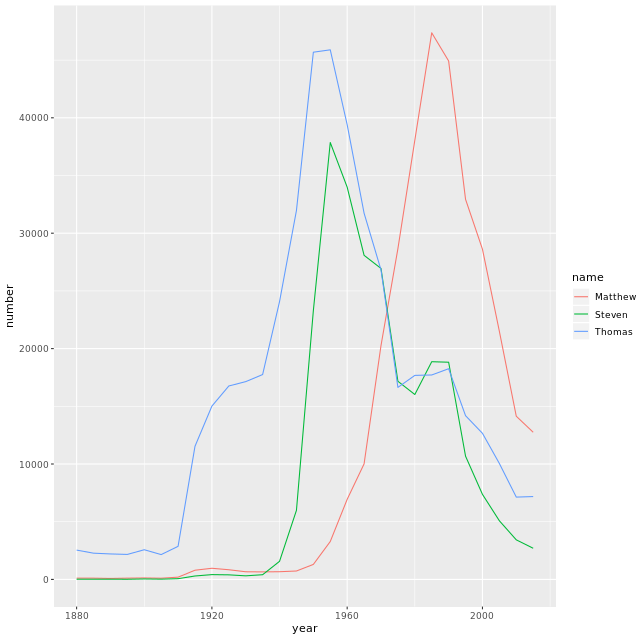
# Visualizing names with ggplot2

The dplyr package is very useful for exploring data, but it's especially useful when combined with other tidyverse packages like ggplot2.





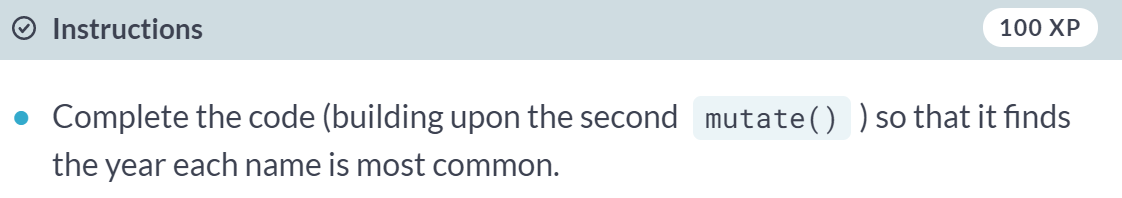
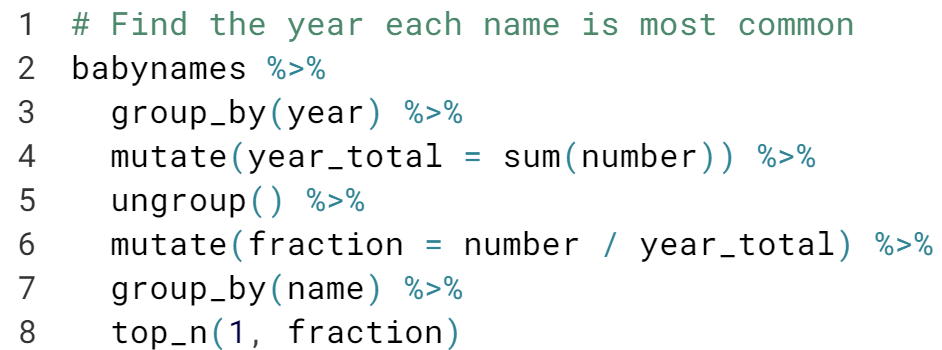


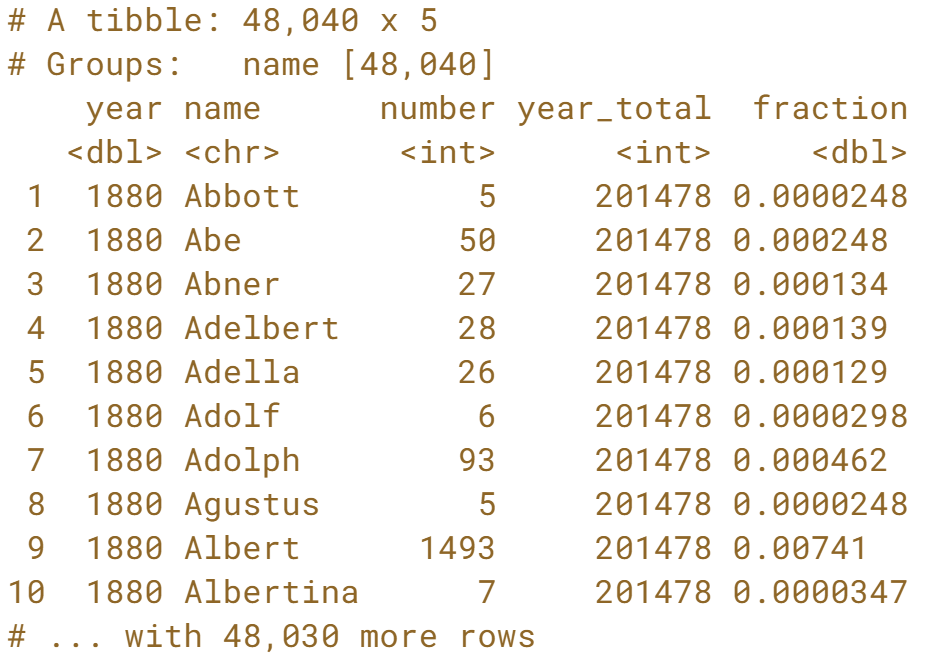
  


# Finding the year each name is most common

In an earlier video, you learned how to filter for a particular name to determine the frequency of that name over time. Now, you're going to explore which year each name was the most common.

To do this, you'll be combining the grouped mutate approach with a top\_n.



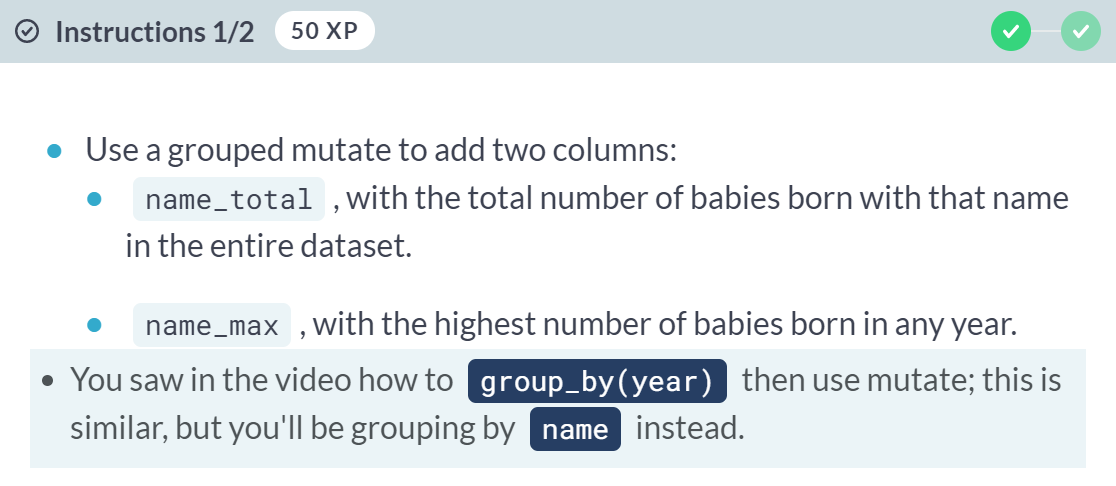
Notice that the results are grouped by year, then name, so the first few entries are names that were most popular in the 1880's that start with the letter A.

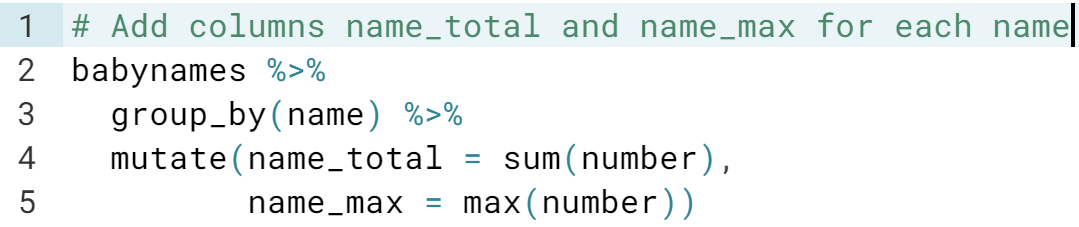
# Adding the total and maximum for each name

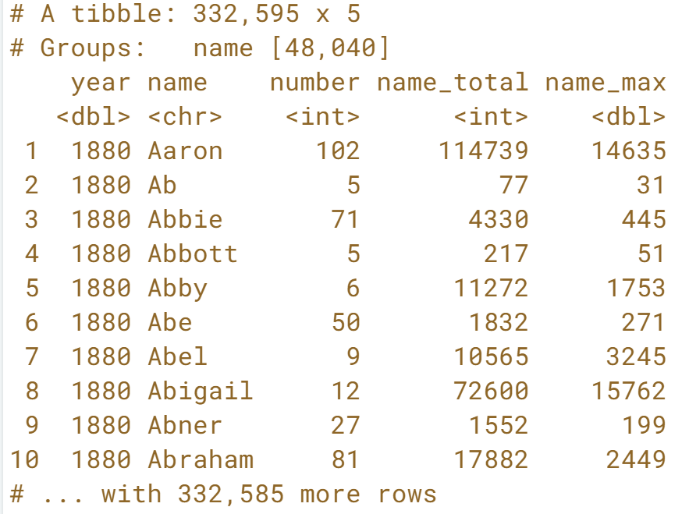
In the video, you learned how you could group by the year and use mutate() to add a total for that year.

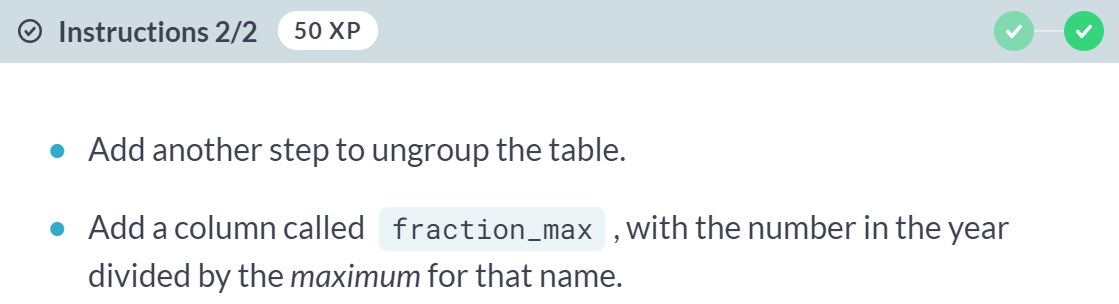
In these exercises, you'll learn to normalize by a different, but also interesting metric: you'll divide each name by the maximum for that name. This means that every name will peak at 1.

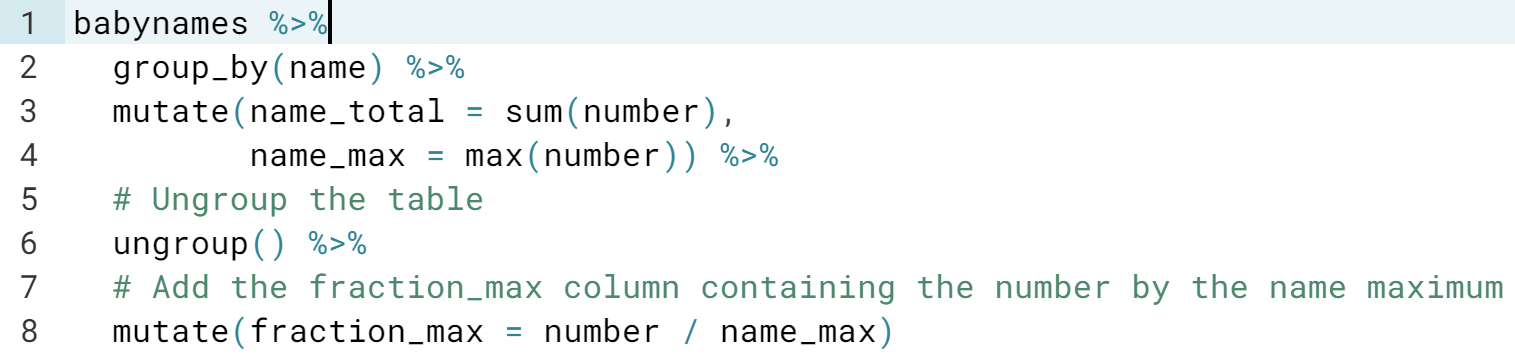
Once you add new columns, the result will still be grouped by name. This splits it into 48,000 groups, which actually makes later steps like mutates slower.

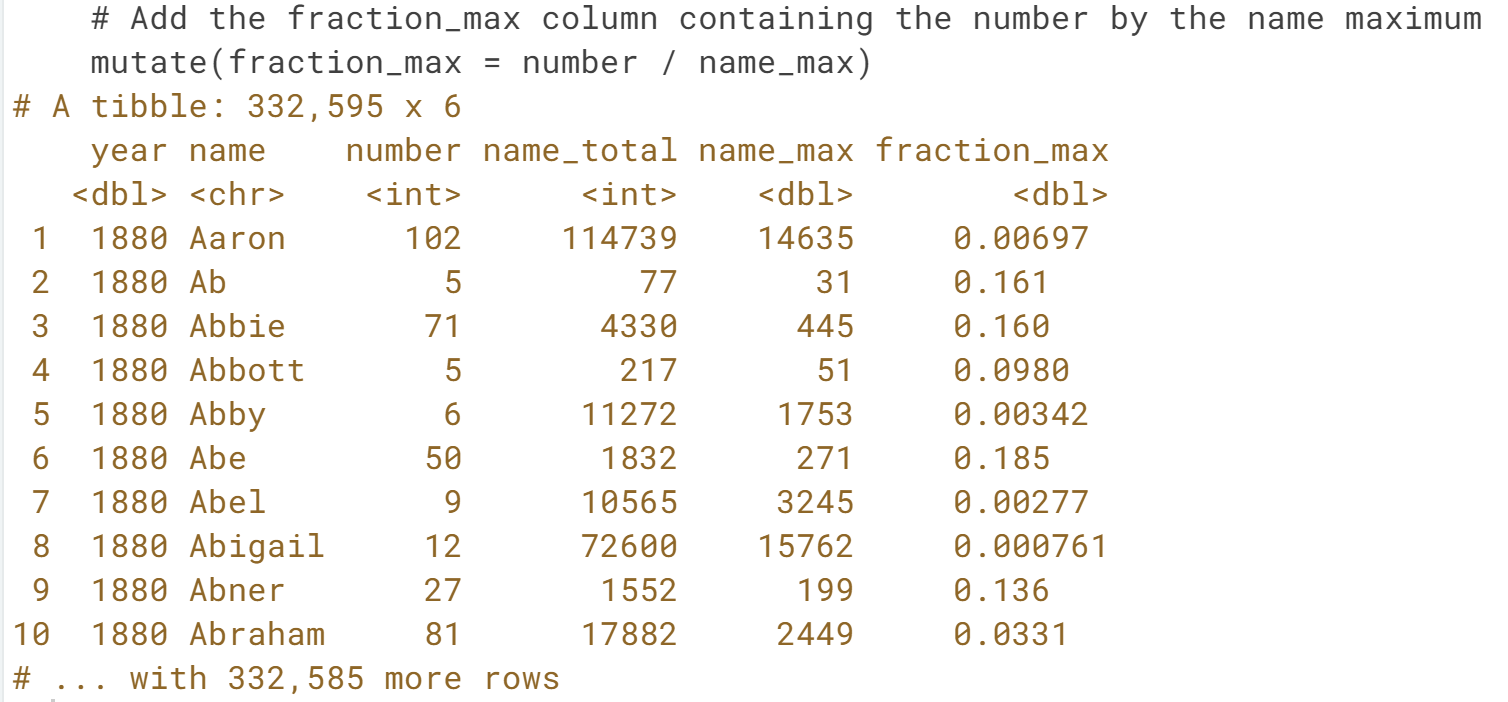












This tells you, for example, that the name Abe was at 18.5% of its peak in the year 1880.

# Visualizing the normalized change in popularity

You picked a few names and calculated each of them as a fraction of their peak. This is a type of "normalizing" a name, where you're focused on the relative change within each name rather than the overall popularity of the name.

In this exercise, you'll visualize the normalized popularity of each name. Your work from the previous exercise, names\_normalized, has been provided for you.

names\_normalized <- babynames %>%

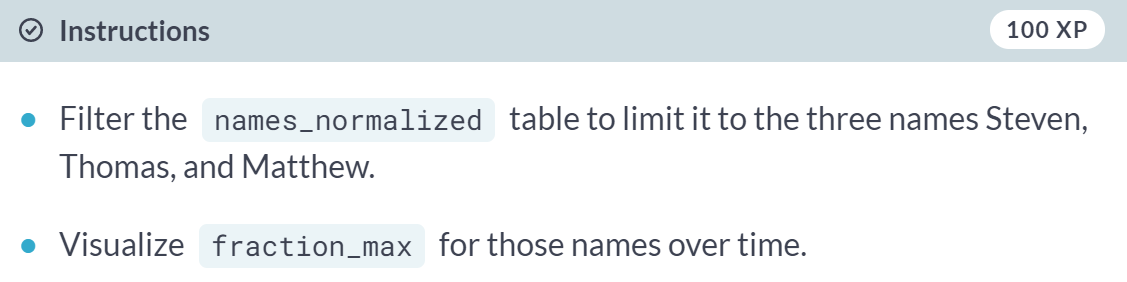
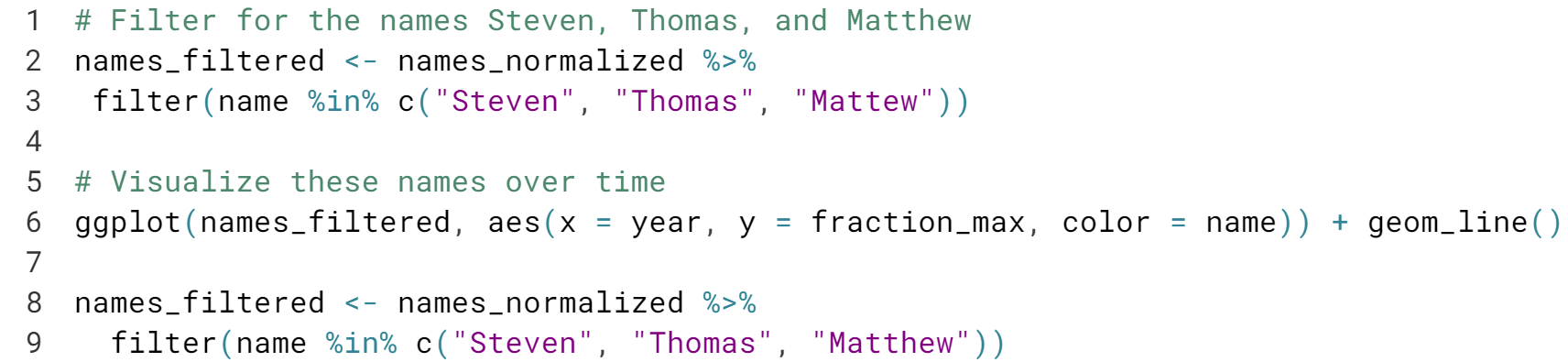
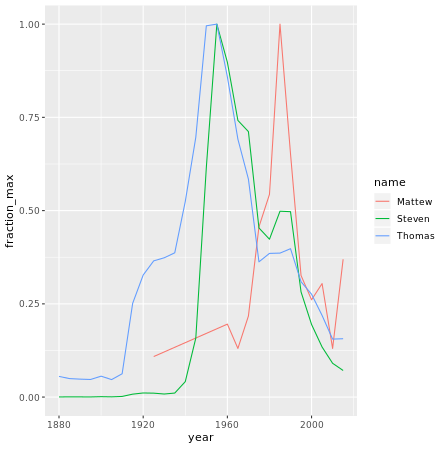
group\_by(name) %>%

mutate(name\_total = sum(number),

name\_max = max(number)) %>%

ungroup() %>%

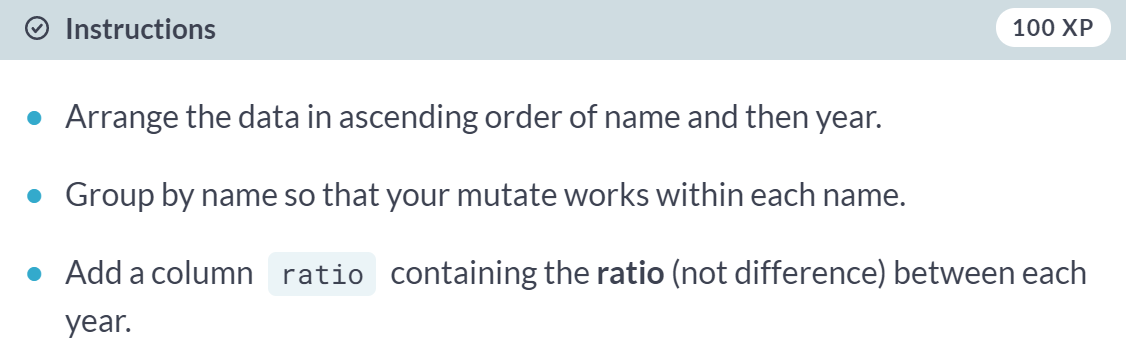
mutate(fraction\_max = number / name\_max)

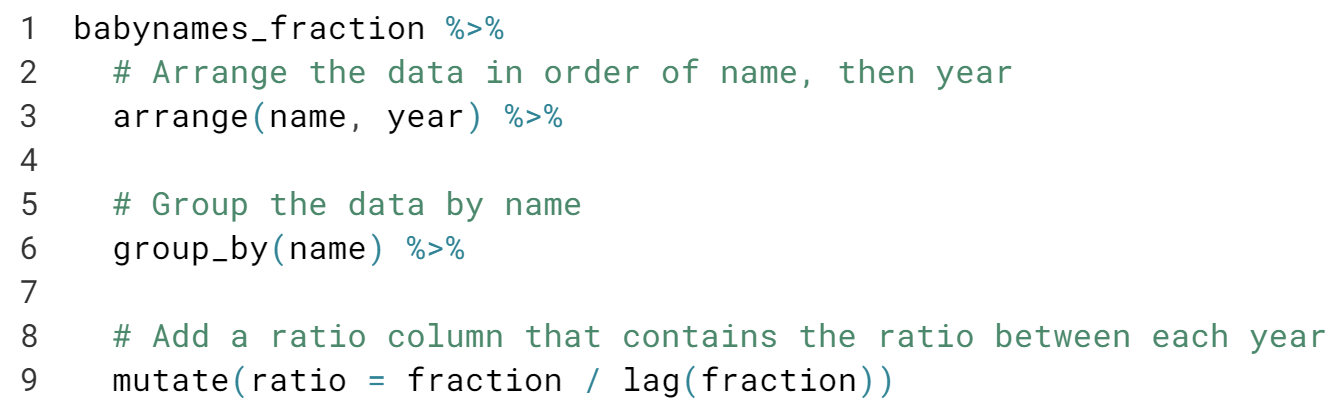
  
  


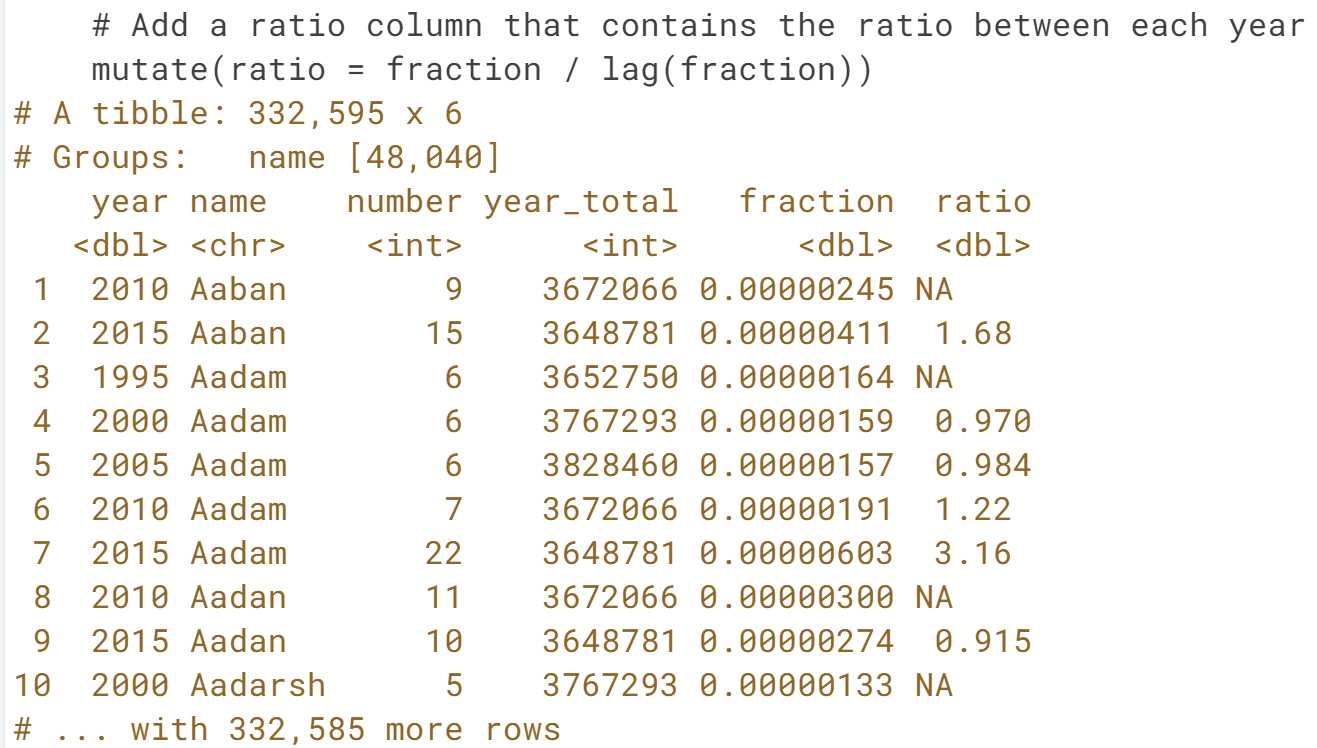
# Using ratios to describe the frequency of a name

In the video, you learned how to find the difference in the frequency of a baby name between consecutive years. What if instead of finding the difference, you wanted to find the ratio?

You'll start with the babynames\_fraction data already, so that you can consider the popularity of each name within each year.





  
Notice the first observation for each name is missing a ratio, since there is no previous year.

# Biggest jumps in a name

Previously, you added a ratio column to describe the ratio of the frequency of a baby name between consecutive years to describe the changes in the popularity of a name. Now, you'll look at a subset of that data, called babynames\_ratios\_filtered, to look further into the names that experienced the biggest jumps in popularity in consecutive years.

babynames\_ratios\_filtered <- babynames\_fraction %>%

arrange(name, year) %>%

group\_by(name) %>%

mutate(ratio = fraction / lag(fraction)) %>%

filter(fraction >= 0.00001)

