Activity 3: Data visualization: fundamentals of ggplot

Pete Nelson

Setting up

First, let's install and load our packages. You should already have the `here` package installed from last time, but you may not have `ggplot2` installed. Install it in your console. Then, read it in from the library in your code.

```
# The first 4 lines of code hide the output that loading these libraries would typically gene
library(tidyverse)
library(here)
```

Read in the data

```
# Read in the data and store as "ci_np" data object
ci_np <- read.csv("ci_np.csv")

# display the first 6 rows
head(ci_np)
```

```
  region state code                      park_name          type visitors year
1     PW    CA CHIS Channel Islands National Park National Park     1200 1963
2     PW    CA CHIS Channel Islands National Park National Park     1500 1964
3     PW    CA CHIS Channel Islands National Park National Park     1600 1965
4     PW    CA CHIS Channel Islands National Park National Park      300 1966
5     PW    CA CHIS Channel Islands National Park National Park    15700 1967
6     PW    CA CHIS Channel Islands National Park National Park    31000 1968
```

This dataset is for the Channel Islands National Park visitation data. It looks like the first 5 columns are categorical data, while the last two (`visitors` and `year`) are continuous. Every row (every datapoint) is the number of visitors for a given National Park in a given year.

Q1: Fetch column names

Dataframes can be very large and have many columns; it is sometimes useful to be able to concisely see all of the column names of a dataframe. Just like with the `head()` function earlier, use the `colnames()` function to retrieve and look at a vector of all of the column names of this dataframe.

```
colnames(ci_np)
```

```
[1] "region"    "state"     "code"      "park_name" "type"      "visitors"
[7] "year"
```

```
# Why not use?
names(ci_np)
```

```
[1] "region"    "state"     "code"      "park_name" "type"      "visitors"
[7] "year"
```

Graphing basics

Data and axes

Start by giving the ggplot function some data:

```
ggplot(data = ci_np)
```

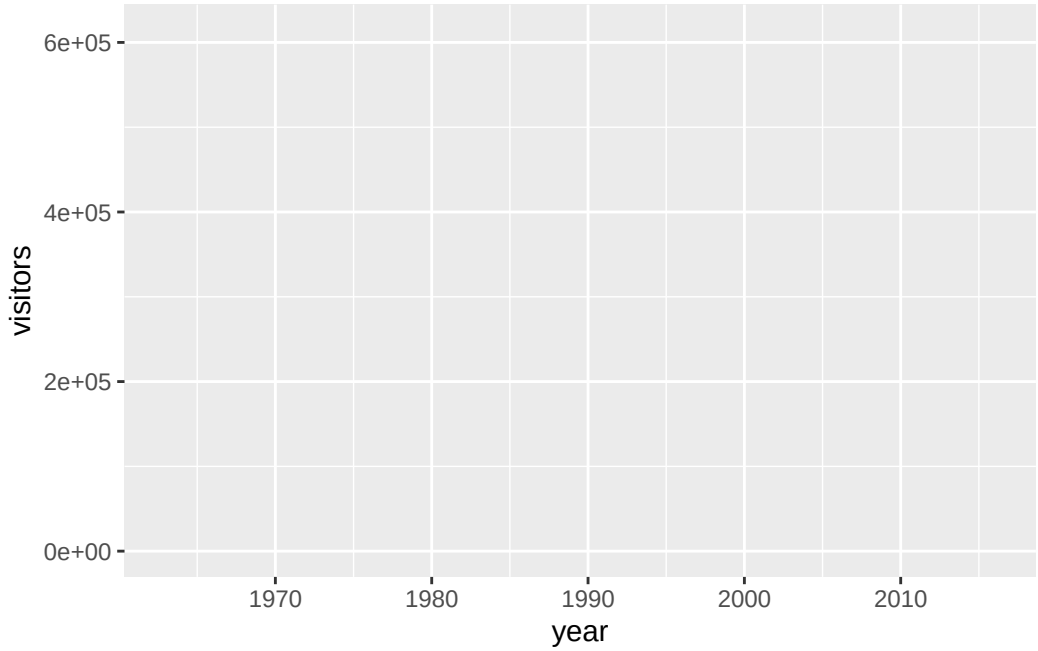
```
# Note that this just creates a blank space! Why? Because you haven't told ggplot what to do
```

Here is where we come up with the question that we want to answer. Let's ask: **How does the number of visitors to the Channel Islands National Park change over time?**

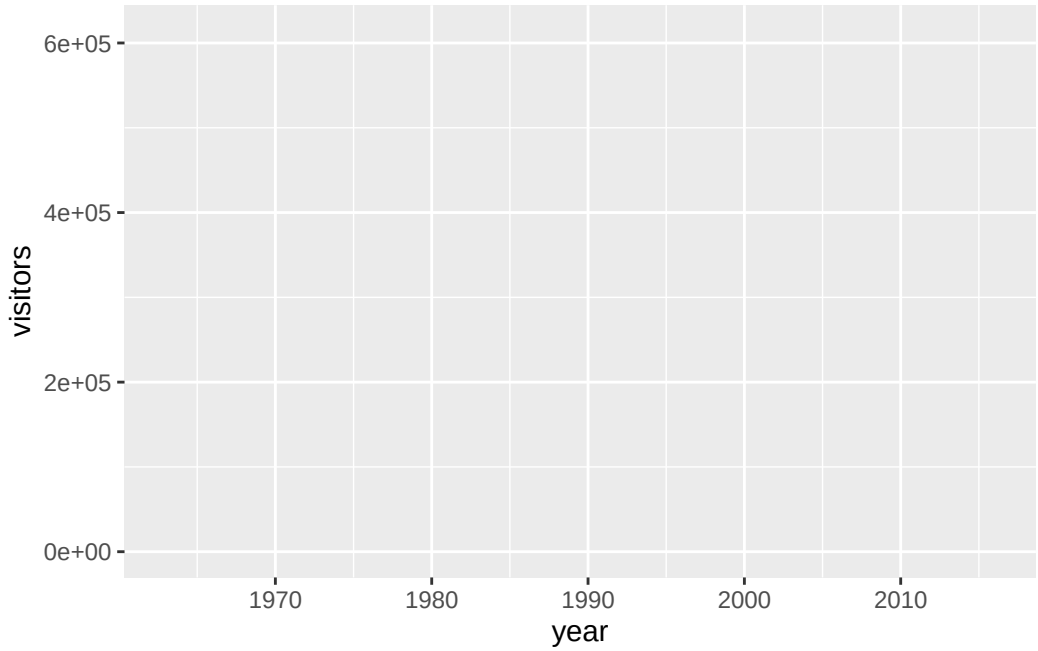
To tell ggplot what to plot, we need to designate the "aesthetics" of the graph using the `aes()` function. The aesthetics tell `ggplot()` how the graph is going to look. Most importantly, you need an x and y axis!

Let's try this again with ggplot's "mapping" argument. To answer our question, we will plot the two continuous variables: number of `visitors` to the park and `year`. `Year` will be on the x-axis, as our predictor variable, while `visitors` will be on the y-axis, as our response variable. Remember to spell/capitalize the column names correctly here!

```
ggplot(ci_np,
       aes(year, visitors))
```




```
# that's shorthand for...
ggplot(data = ci_np, mapping = aes(x = year, y = visitors))
```



```
# note that, while we've told ggplot what we want to graph, we haven't told it *how* those da
```

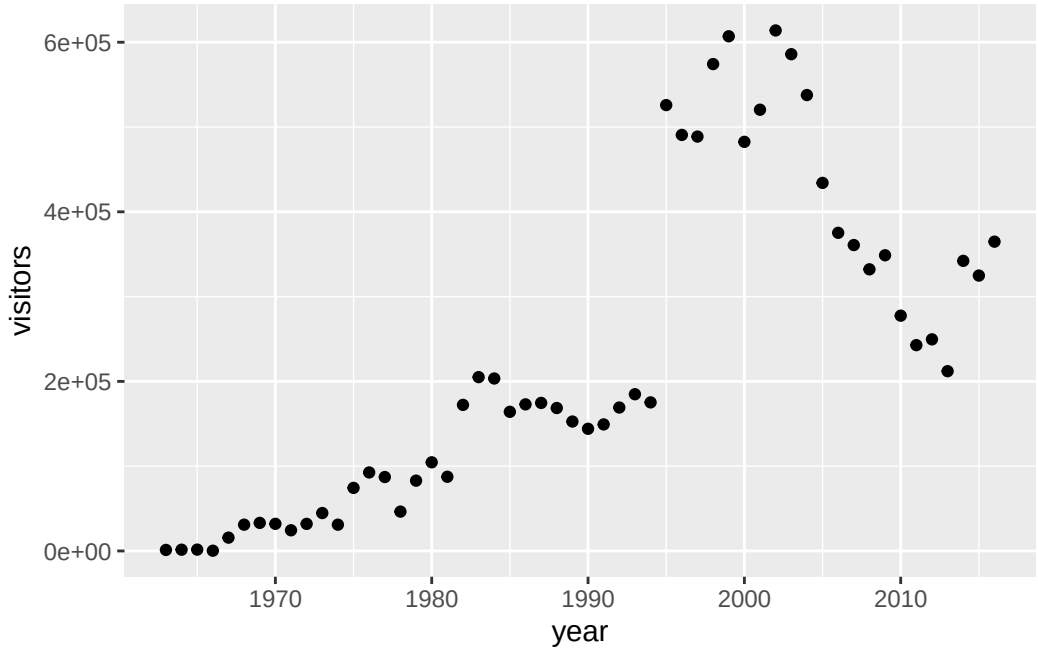
Now what has changed? We have axis titles and numbers - great! But why can't we see any of the data? That's because we need to tell ggplot what kind of **graph** we want.

We do this using the built-in series of geom (graph type) functions. These have the format of `geom_*()`, where `*` is a type of graph. For example, `geom_point()` is a scatterplot, `geom_line()` is a line graph, `geom_col()` is a column graph.

We add layers (including geoms) to a ggplot by putting a "+" after the closed parenthesis of the `ggplot()` function and then creating a new line with the geom we want.

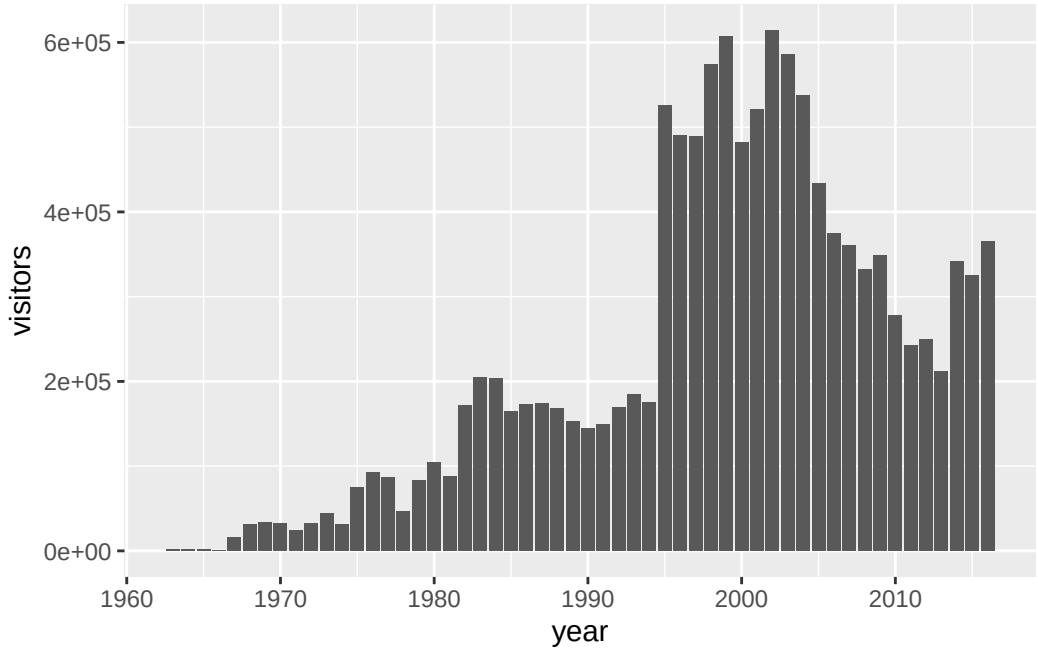
Let's make it a scatterplot, meaning we have a point for every combination of `year` and `visitors`, using the `geom_point()` function.

```
ggplot(ci_np,
       aes(year, visitors)) +
  geom_point()
```



Now let's try again using a column graph:

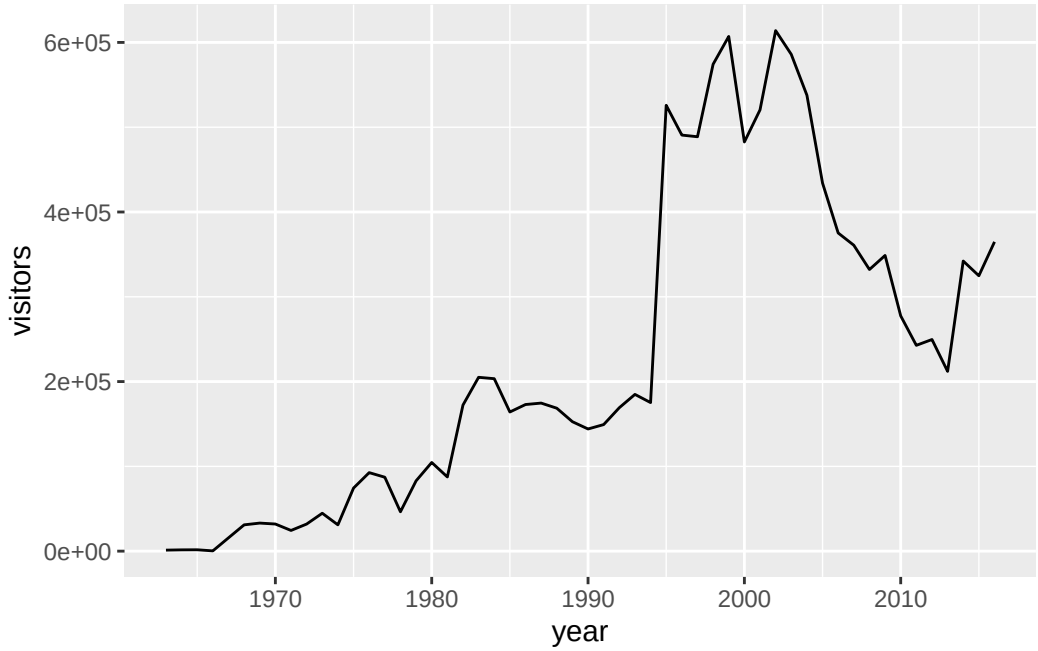
```
ggplot(ci_np,
       aes(year, visitors)) +
  geom_col()
```



Q2: Create a line plot

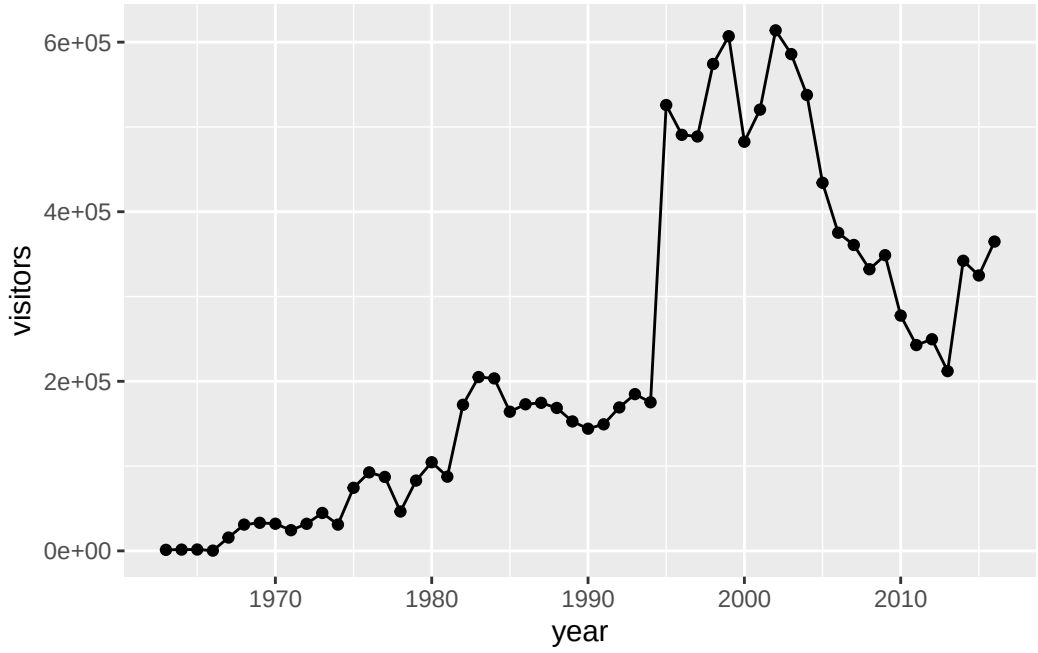
Replicate the same graph, but this time with a line graph.

```
p <-
  ggplot(ci_np,
       aes(year, visitors))
p + geom_line()
```



Add multiple geoms to one graph by simply adding another line with a `+`. The layers are applied in order, so a later layer may cover or obscure the former layer(s).

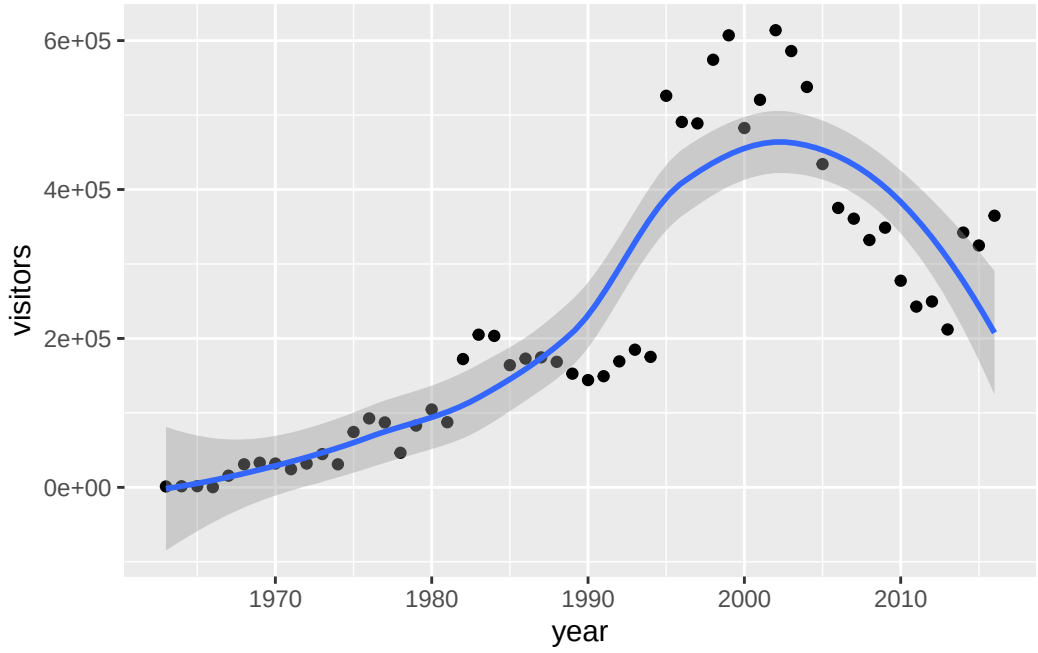
```
p + geom_point() + geom_line()
```



Another useful geom is `geom_smooth()`, which helps visualize patterns in the data by fitting some basic models to the data (note: this does not replace doing statistics on the data!)

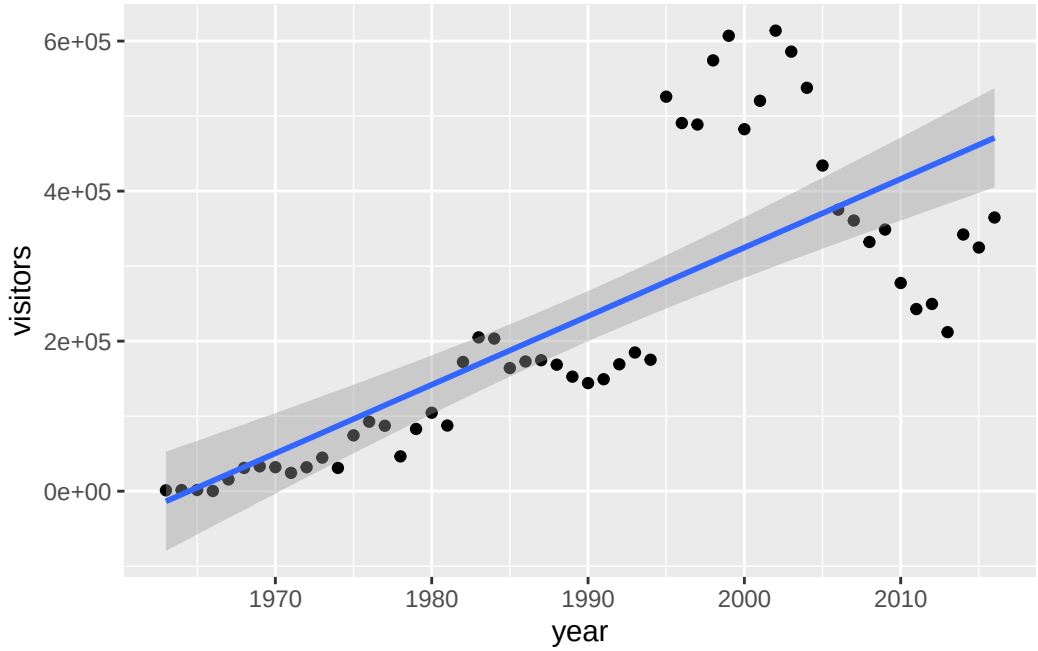
```
p + geom_point() + geom_smooth()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```
p + geom_point() + geom_smooth(method = lm)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

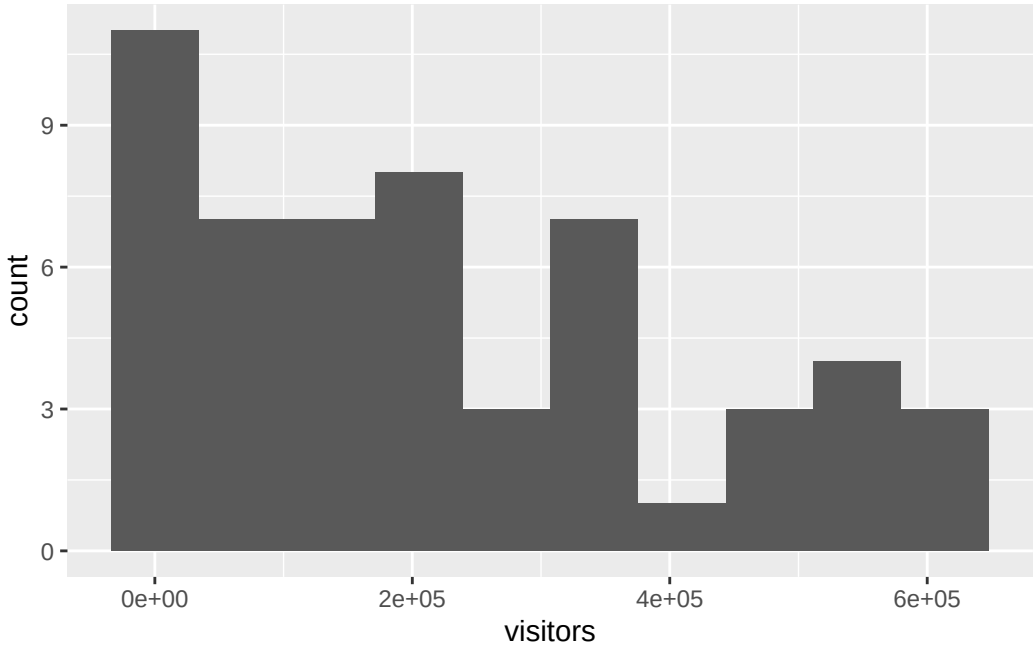


Plotting only one variable

To look at the distribution of a single continuous variable to ask how the values are distributed (are there mostly small values with a few large values as outliers?), we can use `geom_histogram()` or `geom_density()`.

When we use these two functions, we only need to provide an x-axis in the `aes()` and can skip the y-axis. Let's use a histogram to look at the distribution of the `visitors` variable:

```
ggplot(data = ci_np, mapping = aes(x = visitors)) +
  geom_histogram(bins = 10)
```

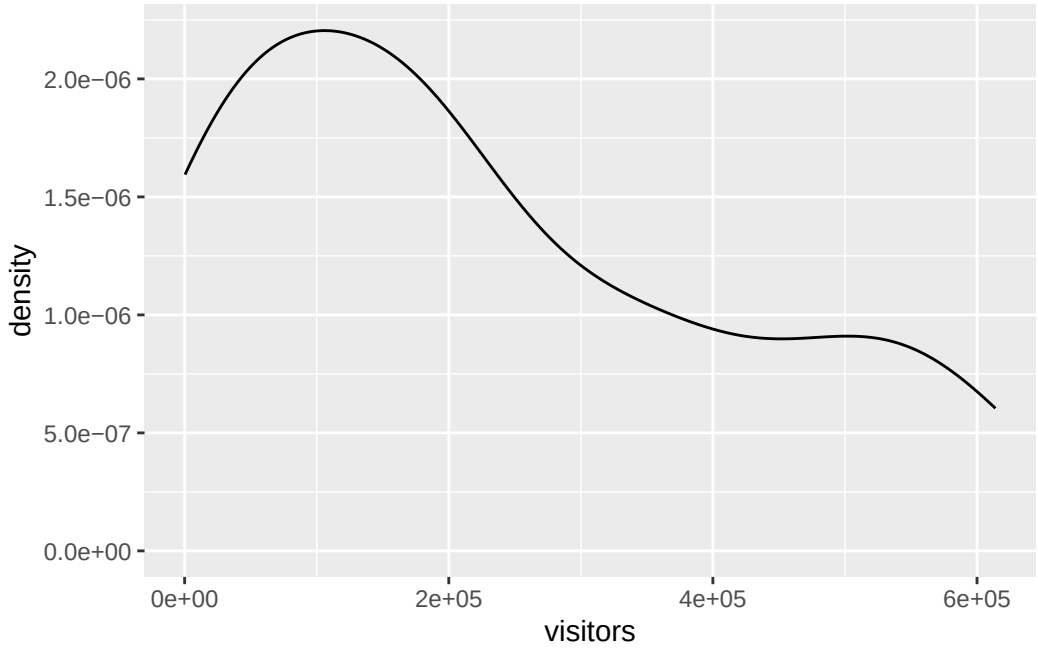



```
# the default bin size is 30...run the code with simply 'geom_histogram()' and see what that
```

Alternately, we can use `geom_density()` which is a smoothed version of a histogram. It does the same thing but fits a line to it, calculating what's called a "kernel density estimate". Looks like the data in the `visitors` column is mostly small values, with a healthy handful of large values.

```
h <- ggplot(ci_np, aes(visitors))

h + geom_density()
```



Customizing things

Graphs in ggplot are highly customizable. We can modify all sorts of aspects of this graph (**MUCH** more on this next class).

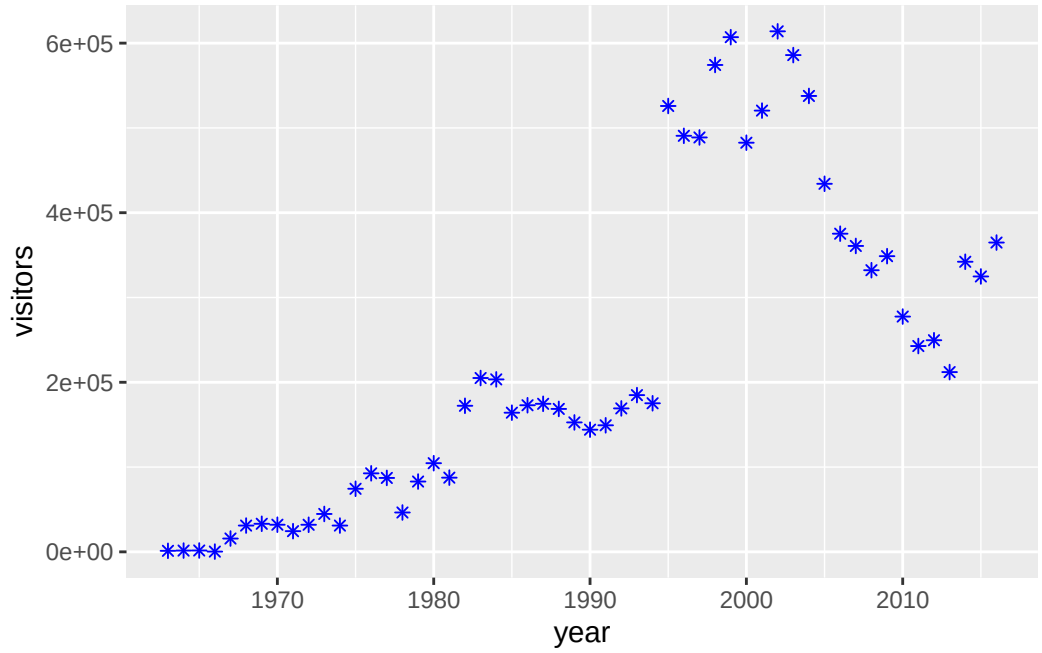
Some common arguments to modify aspects of the geoms are:

- `color =` or `colour =`: update point or line colors
- `fill =`: update fill color for objects with areas (like columns and certain point shapes)
- `linetype =`: update the line type (dashed, long dash, etc.)
- `shape =` or `pch =`: update the point shape/style
- `size =`: update the element size (e.g. of points or line thickness)
- `alpha =`: update element opacity (1 = opaque, 0 = transparent)

Let's start by changing the shape and the color of the points. The bottom of this handy cheatsheet has the possible shapes you can use, numbered 0 through 25: *https://posit.co/wp-content/uploads/2022/10/data-visualization-1.pdf*

We modify specific, unchanging aspects of the points within the geom_point function itself. What I mean by "unchanging" is that the color, shape, etc, is going to be consistent across all points in the graph.

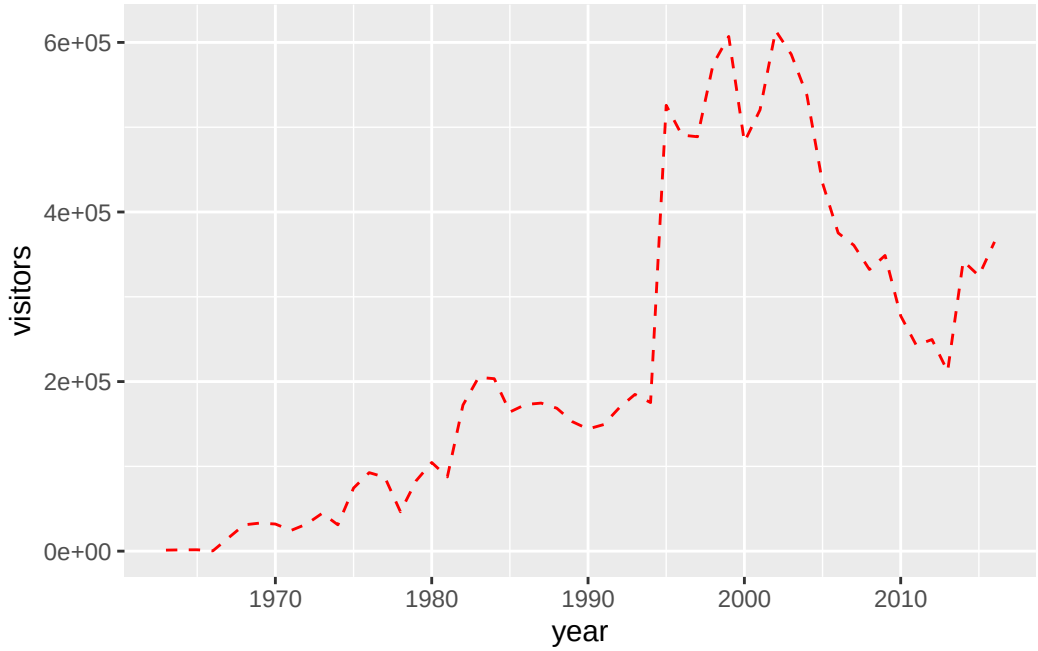
```
p + geom_point(shape = 8, color = "blue")
```



Q3: Modify the lineplot

Let's go back to the lineplot. Create the lineplot, but this time, with a dashed linetype and a different colored line.

```
ggplot(ci_np,
       aes(year, visitors)) +
  geom_line(linetype = 2, color = "red")
```



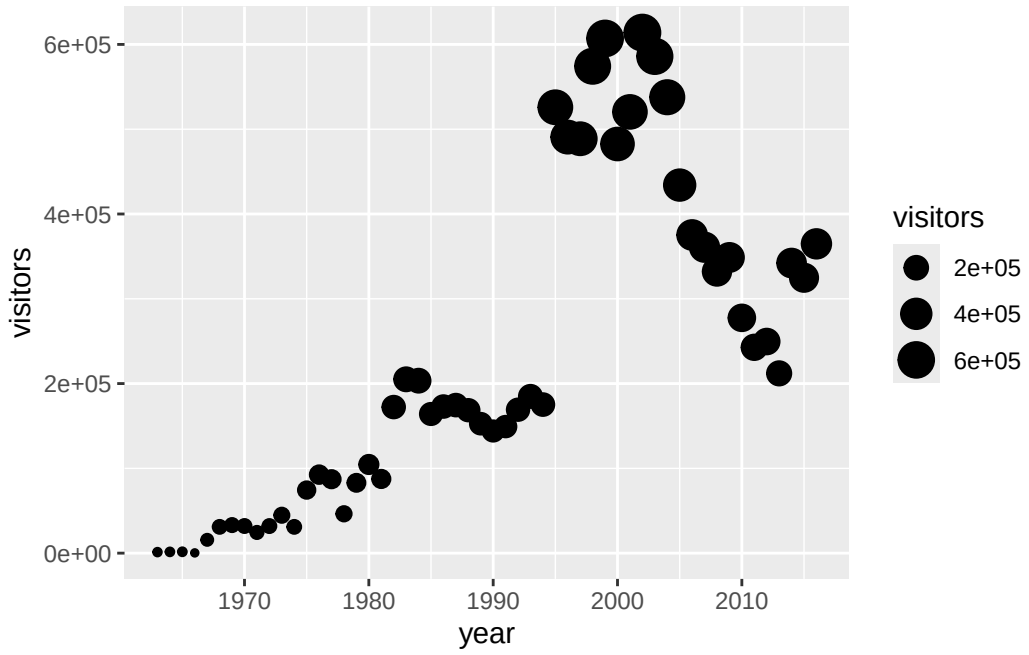

```
# linetype can be 0-6
```

Mapping other variables onto aesthetics

So far we've changed different elements (shape, color) of the geoms based on constant inputs (i.e. the color is the same for every single point). Often, though, we want to have different variables represented in ways other than the x and y axes, such as having different colored points for different categories of the data, or having points grow as another value does. To do that, we will **map variables onto graph aesthetics**, meaning we will change how an element on the graph looks based on a different variable.

Let's practice by making the size of the points vary with the `visitors` column (which is also mapped onto our y-axis). When we want to customize a graph element based on a variable's value, add the element argument within `aes()` in the appropriate `geom_*()` layer:

```
p + geom_point(aes(size = visitors))
```




```
# because the number of visitors is graphed onto the y-axis, linking the size of the points

# note that p + geom_point(size = visitors) doesn't work. why?
```

Neat! We can add in multiple aesthetics at once (though with this example, things are getting unnecessarily complicated… Making clear and concise graphs is an art!).

Lots of parks all at once

Q4: Reading in the new data

Let's practice mapping additional aesthetics with an expanded dataset. We've been working with the `ci_np` dataset, which includes only Channel Islands National Park. The `ca_np.csv` dataset includes *all* California National Parks! Read this .csv file in and store it as a data object called `ca_np`

Let's look at what parks are present in this new dataset. You can create a vector from a column, isolating all of its values in the order they appear in the dataframe, by putting a `$` after the dataframe's name: `dataframe$columnname`. Then, we can put that output within the `unique()` function to remove duplicate values and see what unique parks we have.

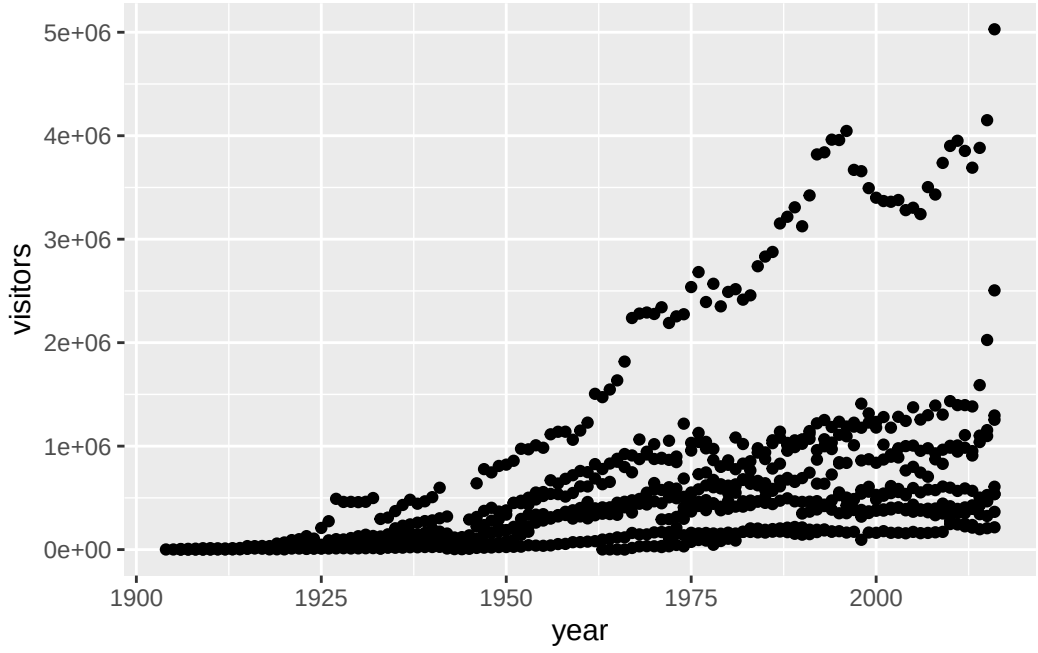
```
unique(ca_np$park_name)
```

```
[1] "Channel Islands National Park" "Death Valley National Park"
[3] "Joshua Tree National Park"     "Kings Canyon National Park"
[5] "Lassen Volcanic National Park" "Pinnacles National Park"
[7] "Redwood National Park"         "Sequoia National Park"
[9] "Yosemite National Park"
```

Helpful! Now we know that there are 9 unique parks in this dataset.

Let's make a simple scatterplot of this data as before, with year on the x and visitors on the y.

```
ggplot(ca_np,
       aes(year, visitors)) +
  geom_point()
```

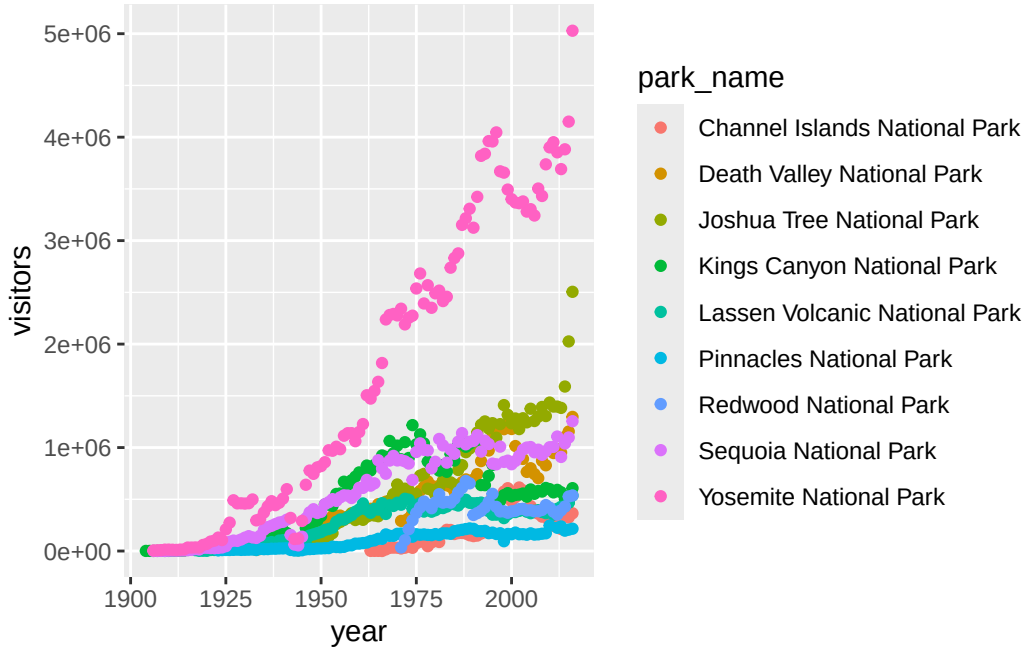


Q5: Mapping park name onto the aesthetics

Duplicate the scatterplot from above, but make the color of points vary with the park name.

```
p2 <- ggplot(ca_np,
       aes(year, visitors))

p2 + geom_point(aes(color = park_name))
```

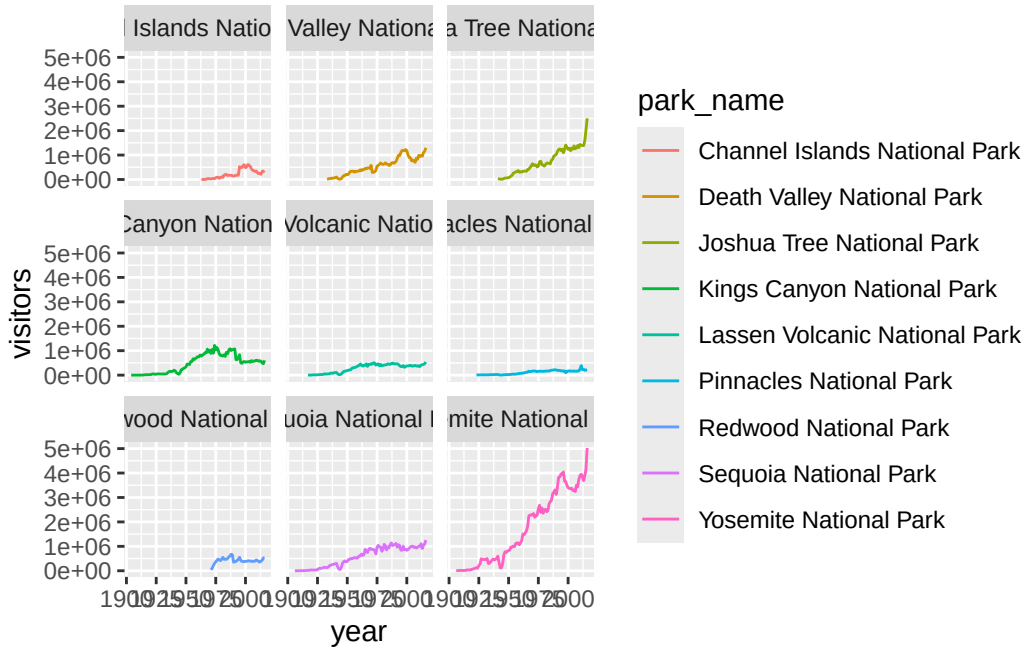


Multi-pane graphs through facet_wrap()

We can also separate out categorical data that exists in the dataset into separate panels of graphs that ggplot calls "facets". That might make viewing the separate parks a bit easier. We do this with the `facet_wrap()` or `facet_grid` functions. We'll use the `facet_wrap()` function today; `facet_grid()` is a better choice if you want the columns and rows of the facets to represent different things (but don't worry too much about it - I still get confused on which one to use).

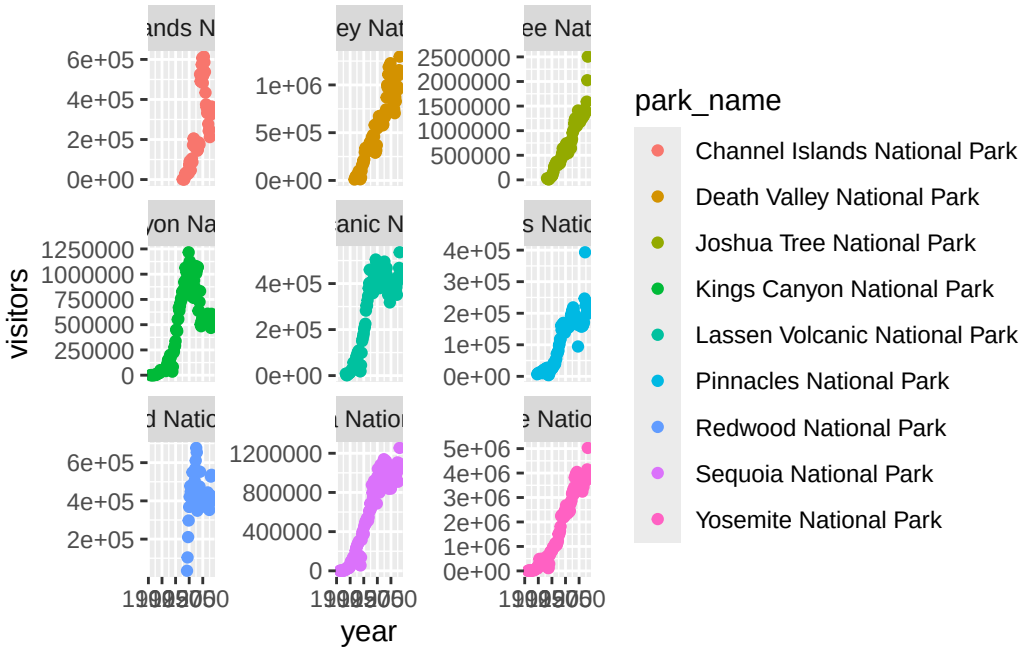
Within the function, put the variable name you want to facet followed by a `~`, and then a `.`: `facet_wrap(variable ~ .)`. The `.` is a placeholder for a second variable (i.e. its presence indicates that we are only faceting by one variable).

```
p2 + geom_line(aes(color = park_name)) +
  facet_wrap(park_name ~ .)
```



We can change the scales by changing the default input to the `scales =` argument. In this case, let's make the y-axis "free" instead of "fixed".

```
p2 + geom_point(aes(color = park_name)) +
  facet_wrap(park_name ~ ., scales = "free_y")
```



Abalone landings data

In this next part of the activity, you will take what you have learned - from reading in data to modifying graphs - and apply it to a new dataset about abalone landings in California: `abalone_landings.csv`. Work together to do the three things below, and come to us if you have any questions.

Q6 Write a research question

Explore this dataset a little bit and then write a research question that you would like to answer.

```
abs <- read_csv("abalone_landings.csv")
```

```
Rows: 125 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): Abalone_Species
dbl (2): Year, Abalone_Landings_lbs
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(abs)
```

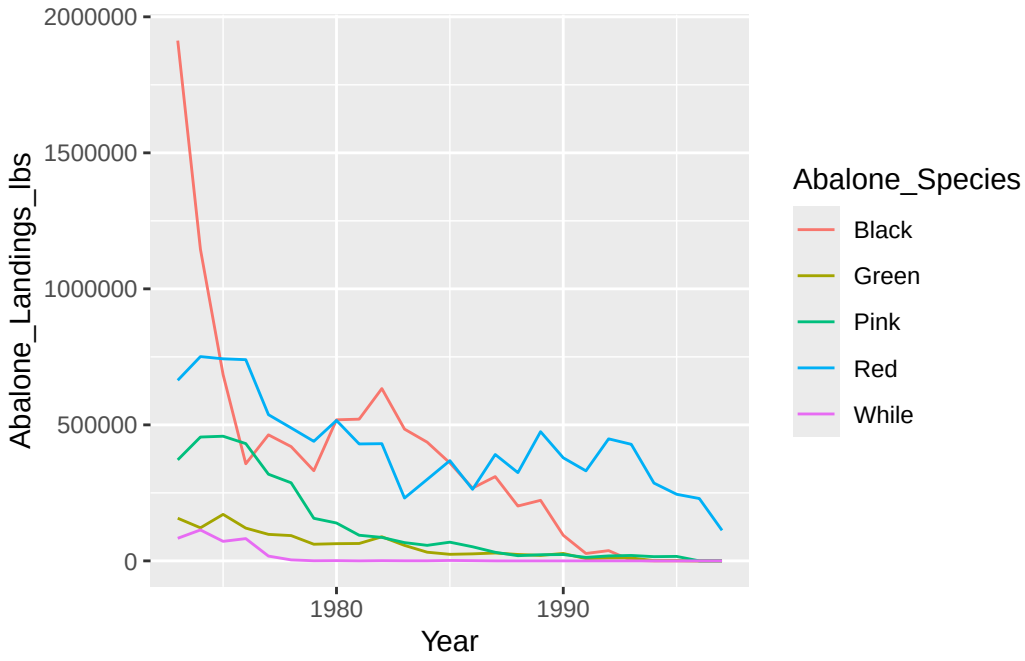
```
# A tibble: 6 x 3
   Year Abalone_Species Abalone_Landings_lbs
  <dbl> <chr>                          <dbl>
1  1973 Black                        1912519
2  1974 Black                        1145396
3  1975 Black                         684793
4  1976 Black                         356951
5  1977 Black                         463301
6  1978 Black                         420045
```

```
view(abs)
```

How do annual abalone landings vary among species?

Q7 Make a graph to answer your question

```
p3 <- ggplot(abs,
             aes(Year, Abalone_Landings_lbs))
p3 + geom_line(aes(color = Abalone_Species))
```



After you have made a graph, write 1) a sentence or two about what you have learned from the graph and 2) a follow-up research question (is there additional data that you might want to acquire for your follow-up question?).

Discussion

Landings for all five abalone species have declined; presently, only Reds are harvested legally. Blacks are notable for the precipitous drop during the first 6-7 years, perhaps due to withering disease. It'd be interesting to learn if the decline in landings for Black abalone is indeed attributable to disease. It'd also be nice to learn how to assign colors to each of the species based on their names (i.e., "black" for Blacks, "green" for Greens, etc.)

```
ggsave("abalone_landings.jpg", get_last_plot())
```

```
Saving 5.5 x 3.5 in image
```