Quantitative comparisons: bar-charts

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

datacamp

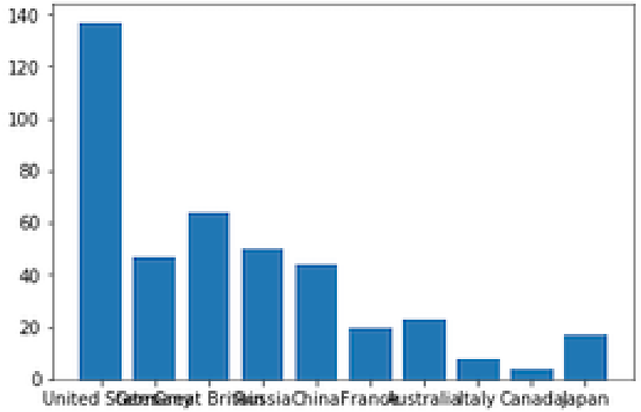
Olympic medals

```
,Gold, Silver, Bronze
United States, 137, 52, 67
Germany, 47, 43, 67
Great Britain, 64, 55, 26
Russia, 50, 28, 35
China, 44, 30, 35
France, 20, 55, 21
Australia, 23, 34, 25
Italy, 8, 38, 24
Canada, 4, 4, 61
Japan, 17, 13, 34
```

2. Olympic medals
Let's look at a dataset that contains information about the number of medals won by a few countries in the 2016 Olympic Games. The data is not very large. Here is all of it. Although you can see all of it in front of you, it's not that easy to make comparisons between different countries and see which countries won which medals.

Olympic medals: visualizing the data

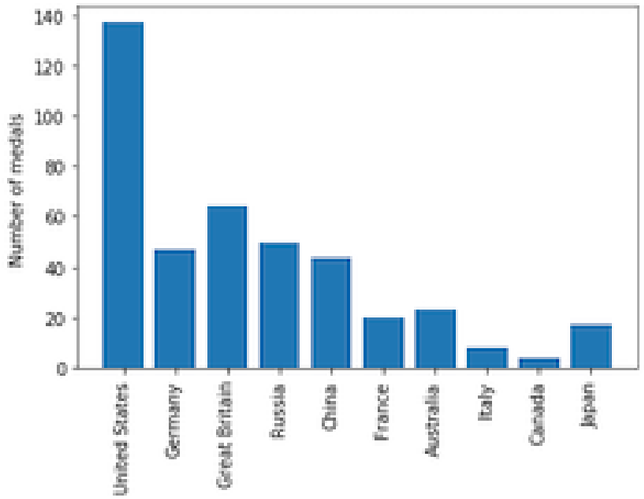
```
medals = pd.read_csv('medals_by_country_2016.csv', index_col=0)
fig, ax = plt.subplots()
ax.bar(medals.index, medals["Gold"])
plt.show()
```



3. Olympic medals: visualizing the data
Let's start by reading the data in from a file. We tell Pandas to create a DataFrame from a file that contains the data and to use the first column, which contains the country names, as the index for the DataFrame. Next, we can visualize the data about gold medals. We create a Figure and an Axes object and call the Axes bar method to create a bar chart. This chart shows a bar for every row in the "Gold" column of the DataFrame, where the height of the bar represents the number in that row. The labels of the x-axis ticks correspond to the index of the DataFrame, which contains the names of the different countries in the data table. **Unfortunately, these names are rather long, so they overlap with each other.** Let's fix that first.

Interlude: rotate the tick labels

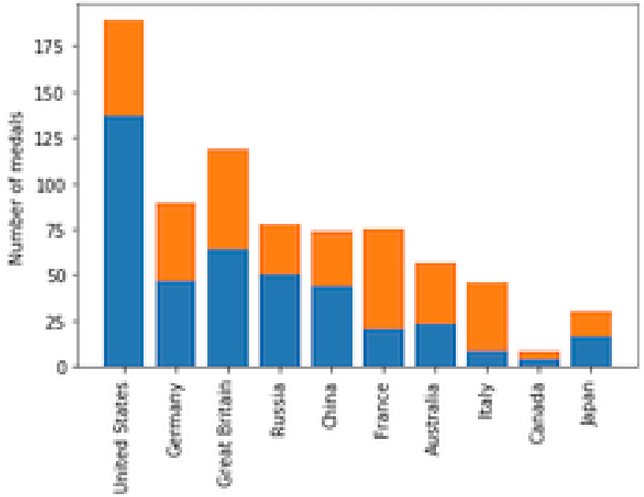
```
fig, ax = plt.subplots()
ax.bar(medals.index, medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
plt.show()
```



4. Interlude: rotate the tick labels
To fix these labels, we can rotate them by 90 degrees. This is done by using the set-underscore-xticklabels method of the Axes. We also take the opportunity to add a label on the y-axis, telling us that the height corresponds to the number of medals. This looks good. Visualizing the data in this way shows us which countries got a high or low number of gold medals, but also allows us to see the differences between countries, based on the difference in heights between the bars.

Olympic medals: visualizing the other medals

```
fig, ax = plt.subplots
ax.bar(medals.index, medals["Gold"])
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
plt.show()
```



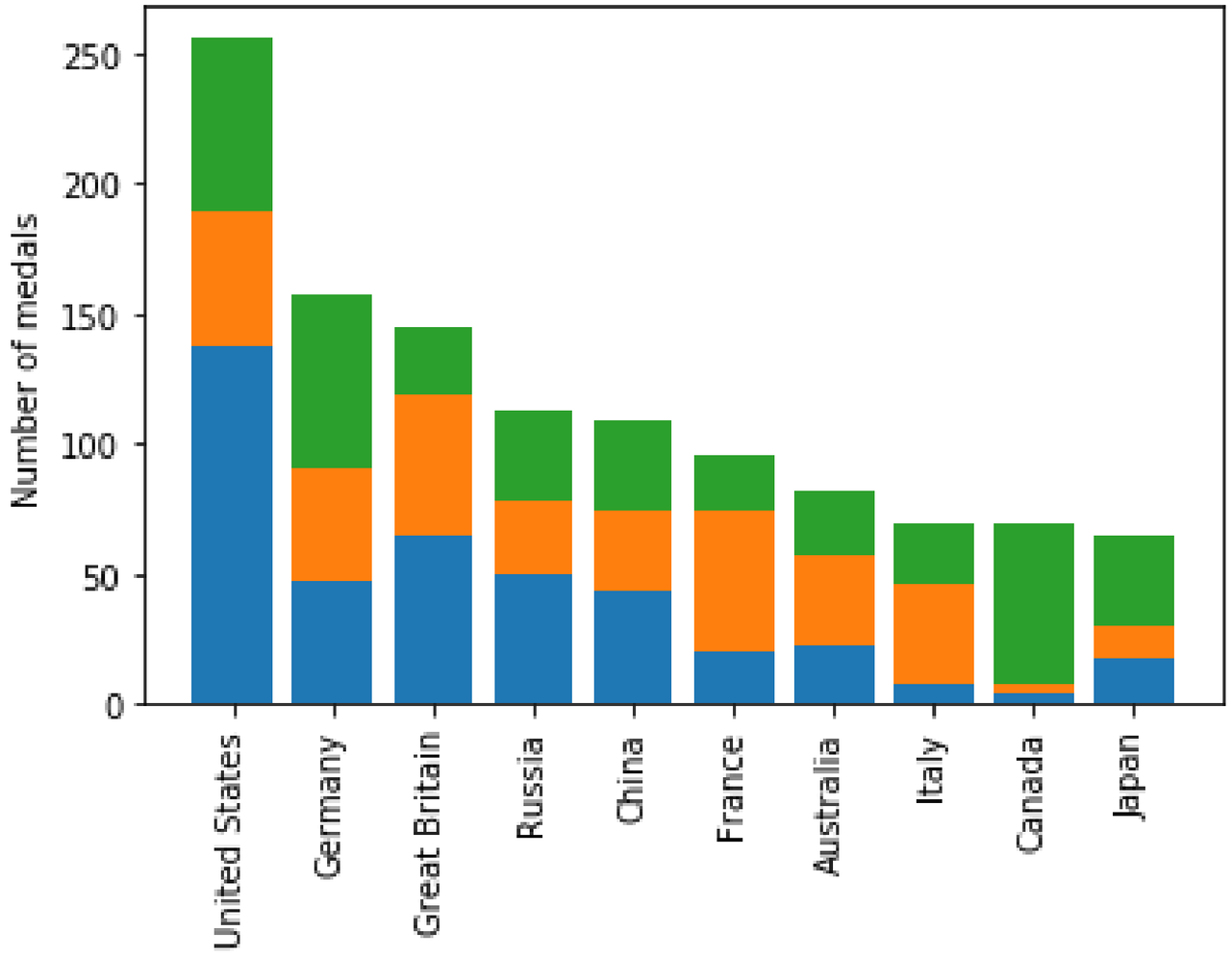
5. Olympic medals: visualizing the other medals
Next, we would like to add the data about the other medals: Silver and Bronze. To add this information into the same plot, we'll create a stacked bar chart. This means that each new data will be stacked on top of the previous data. It starts the same way as before. Next, we add another call to the bar method to add the data from the "Silver" column of the DataFrame. We add the bottom key-word argument to tell Matplotlib that the bottom of this column's data should be at the height of the previous column's data. We add the x-axis tick labels, rotating them by 90 degrees, set the y-axis labels, and call plt-dot-show.

Olympic medals: visualizing all three

```
fig, ax = plt.subplots
ax.bar(medals.index, medals["Gold"])

ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])
ax.bar(medals.index, medals["Bronze"],
       bottom=medals["Gold"] + medals["Silver"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
plt.show()
```

6. Olympic medals: visualizing all three
Similarly, we can add in the number of Bronze medals, setting the bottom of this bar to be the sum of the number of gold medals and the number of silver medals.

Stacked bar chart



Adding a legend

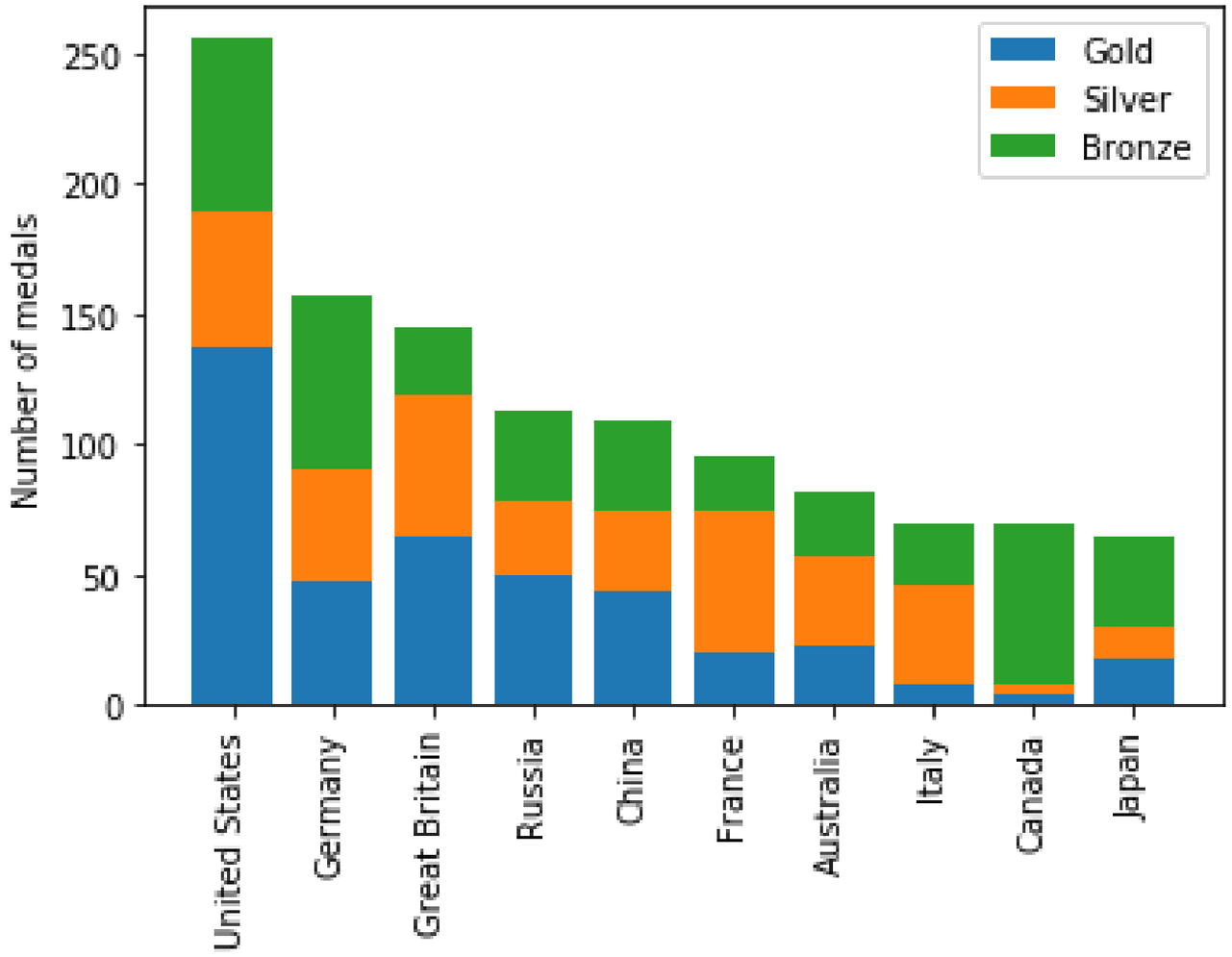
```
fig, ax = plt.subplots
ax.bar(medals.index, medals["Gold"])
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])
ax.bar(medals.index, medals["Bronze"],
       bottom=medals["Gold"] + medals["Silver"])

ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
```

8. Adding a legend
To make this figure easier to read and understand, we would also like to label which color corresponds to which medal. To do this we need to add two things.
9. Adding a legend
The first is to add the label key-word argument to each call of the bar method with the label for the bars plotted in this call. The second is to add a call to the Axes legend method before calling show. This adds in a legend that tells us which color stands for which medal.

Adding a legend

```
fig, ax = plt.subplots
ax.bar(medals.index, medals["Gold"], label="Gold")
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"],
       label="Silver")
ax.bar(medals.index, medals["Bronze"],
       bottom=medals["Gold"] + medals["Silver"],
       label="Bronze")

ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
ax.legend()
plt.show()
```

Stacked bar chart with legend



Create a bar chart!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Quantitative comparisons: histograms

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

This visualization is useful because it shows us the entire distribution of values within a variable.

Ariel Rokem
Data Scientist

datacamp

Histograms

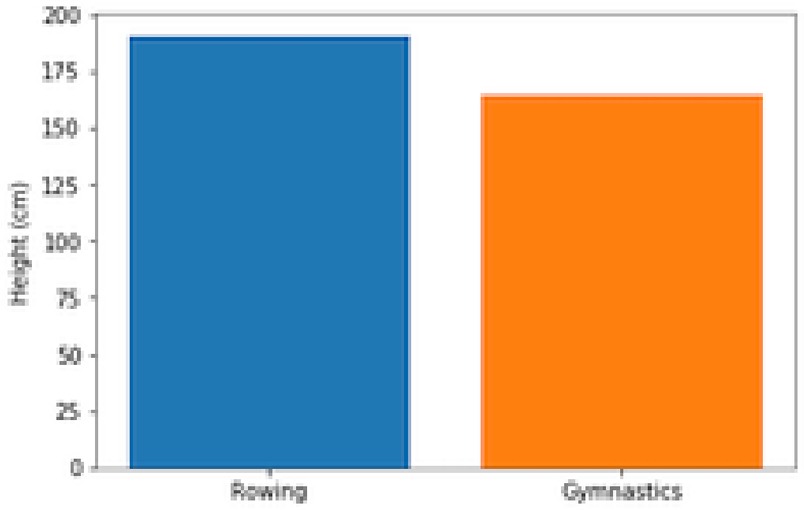
	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
158	62	Giovanni Abagnale	M	21.0	198.0	90.0	Italy	ITA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Coxless Pairs	Bronze
11648	6346	Jrmie Azou	M	27.0	178.0	71.0	France	FRA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Double Sculls	Gold
14871	8025	Thomas Gabriel Jrmie Baroukh	M	28.0	183.0	70.0	France	FRA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Coxless Fours	Bronze
15215	8214	Jacob Jepsen Barse	M	27.0	188.0	73.0	Denmark	DEN	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Coxless Fours	Silver
18441	9764	Alexander Belonogoff	M	26.0	187.0	90.0	Australia	AUS	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Quadruple Sculls	Silver

2. Histograms

Let's look at another example. In this case, we are looking at data about the athletes who participated in the 2016 Olympic Games. We've extracted two DataFrames from this data: all of the medal winners in men's gymnastics and all of the medal winners in men's rowing. Here are the five first rows in the men's rowing DataFrame. You can see that the data contains different kinds of information: what kinds of medals each competitor won, and also the competitor's height and weight.

A bar chart again

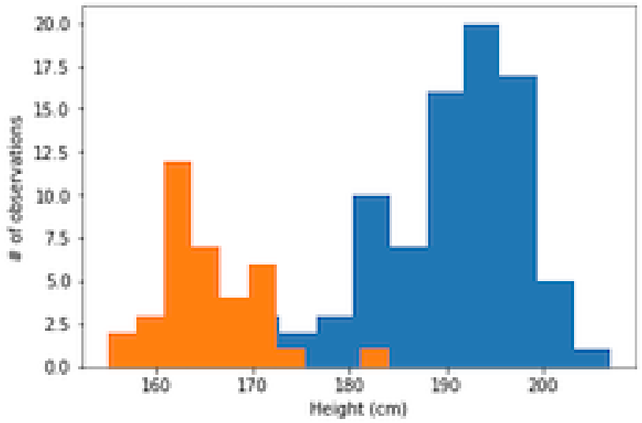
```
fig, ax = plt.subplots()
ax.bar("Rowing", mens_rowing["Height"].mean())
ax.bar("Gymnastics", mens_gymnastics["Height"].mean())
ax.set_ylabel("Height (cm)")
plt.show()
```



3. A bar chart again
Let's start by seeing what a comparison of heights would look like with a bar chart. After creating the Figure and Axes objects, we add to them a bar with the mean of the rowing "Height" column. Then, we add a bar with the mean of the gymnastics "Height" column. We set the y-axis label and show the figure, which gives us a sense for the difference between the groups.

Introducing histograms

```
fig, ax = plt.subplots()
ax.hist(mens_rowing["Height"])
ax.hist(mens_gymnastic["Height"])
ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
plt.show()
```



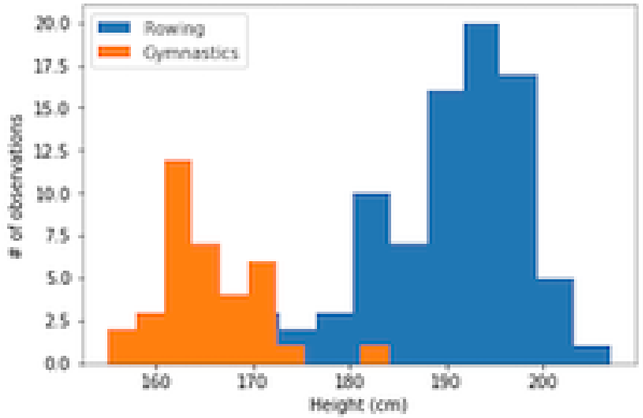
4. Introducing histograms

But a histogram would instead show the full distribution of values within each variable. Let's see that. We start again by initializing a Figure and Axes. We then call the Axes hist method with the entire "Height" column of the men's rowing DataFrame. We repeat this with the men's gymnastics DataFrame. In the histogram shown, the x-axis is the values within the variable and the height of the bars represents the number of observations within a particular bin of values. For example, there are 12 gymnasts with heights between 164 and 167 centimeters, so the highest bar in the orange histogram is 12 units high. Similarly, there are 20 rowers with heights between 188 and 192 centimeters, and the highest bar in the blue histogram is 20 units high.

Labels are needed

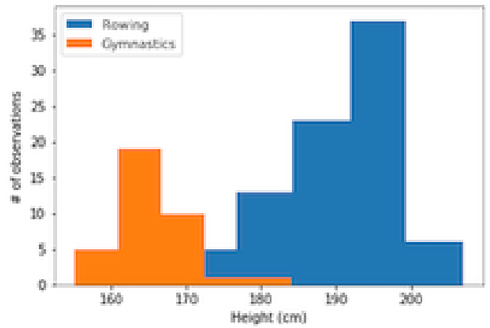
```
ax.hist(mens_rowing["Height"], label="Rowing")
ax.hist(mens_gymnastic["Height"], label="Gymnastics")
ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```

5. Labels are needed
Because the x-axis label no longer provides information about which color represents which variable, labels are really needed in histograms. As before, we can label a variable by calling the hist method with the label key-word argument and then calling the legend method before we call plt-dot-show, so that a legend appears in the figure.



Customizing histograms: setting the number of bins

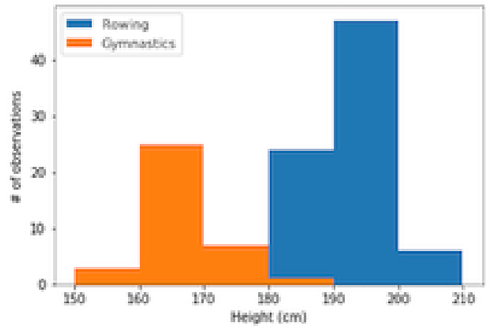
```
ax.hist(mens_rowing["Height"], label="Rowing", bins=5)
ax.hist(mens_gymnastic["Height"], label="Gymnastics", bins=5)
ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```



6. Customizing histograms: setting the number of bins
You might be wondering how Matplotlib decides how to divide the data up into the different bars. Per default, the number of bars or bins in a histogram is 10, but we can customize that. If we provide an integer number to the bins key-word argument, the histogram will have that number of bins.

Customizing histograms: setting bin boundaries

```
ax.hist(mens_rowing["Height"], label="Rowing",
        bins=[150, 160, 170, 180, 190, 200, 210])

ax.hist(mens_gymnastic["Height"], label="Gymnastics",
        bins=[150, 160, 170, 180, 190, 200, 210])

ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```



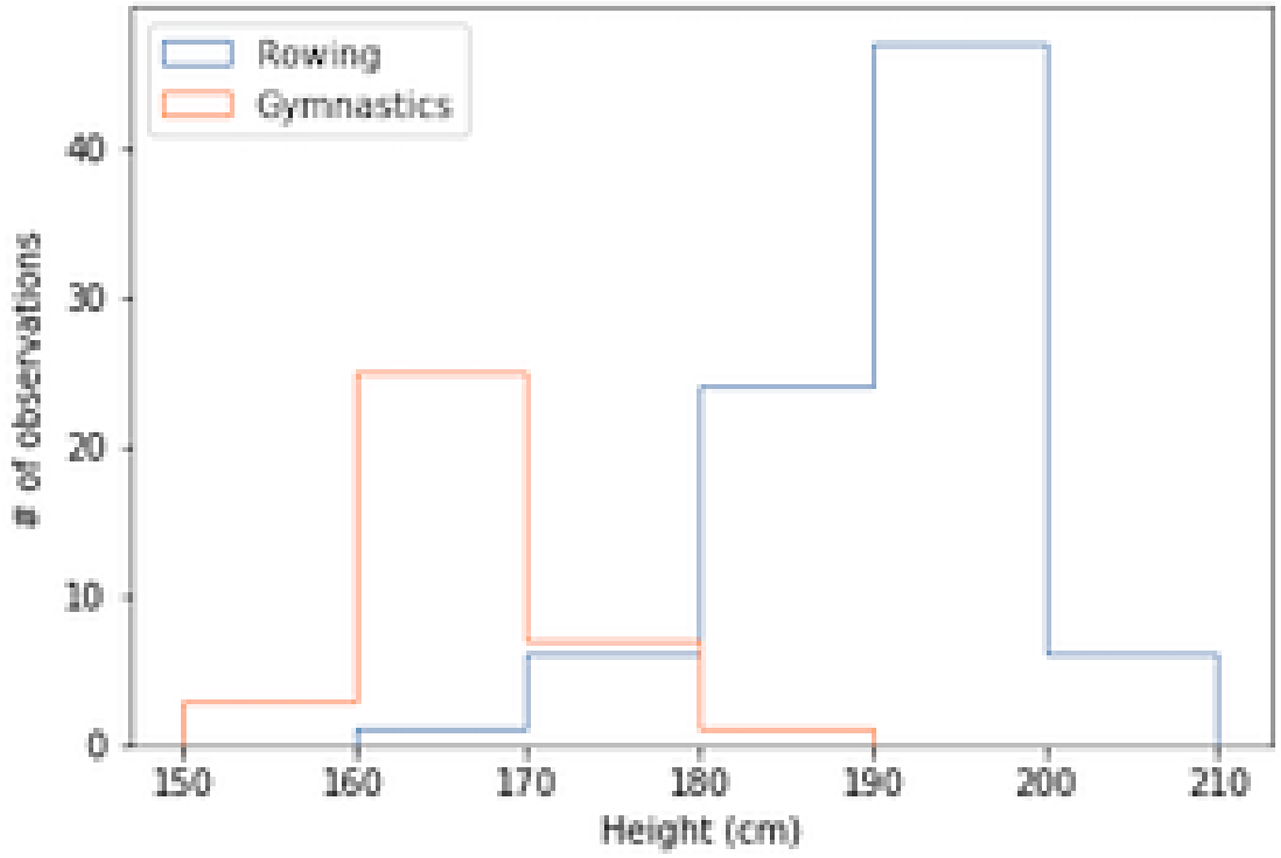
7. Customizing histograms: setting bin boundaries
If we instead provide a sequence of values, these numbers will be set to be the boundaries between the bins, as shown here. There is one last thing to customize. Looking at this figure, you might wonder whether there are any rowing medalists with a height of less than 180 centimeters. This is hard to tell because the bars for the gymnastics histogram are occluding this information.

Customizing histograms: transparency

```
ax.hist(mens_rowing["Height"], label="Rowing",
        bins=[150, 160, 170, 180, 190, 200, 210],
        histtype="step")

ax.hist(mens_gymnastic["Height"], label="Gymnastics",
        bins=[150, 160, 170, 180, 190, 200, 210],
        histtype="step")

ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```

8. Customizing histograms: transparency
The occlusion can be eliminated by changing the type of histogram that is used. Instead of the "bar" type that is used per default, you can specify a histtype of "step", which displays the histogram as thin lines, instead of solid bars,

Histogram with a histtype of step



9. Histogram with a histtype of step exposing that yes: there are rowers with a height of less than 180 centimeters.

Create your own histogram!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Statistical plotting

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

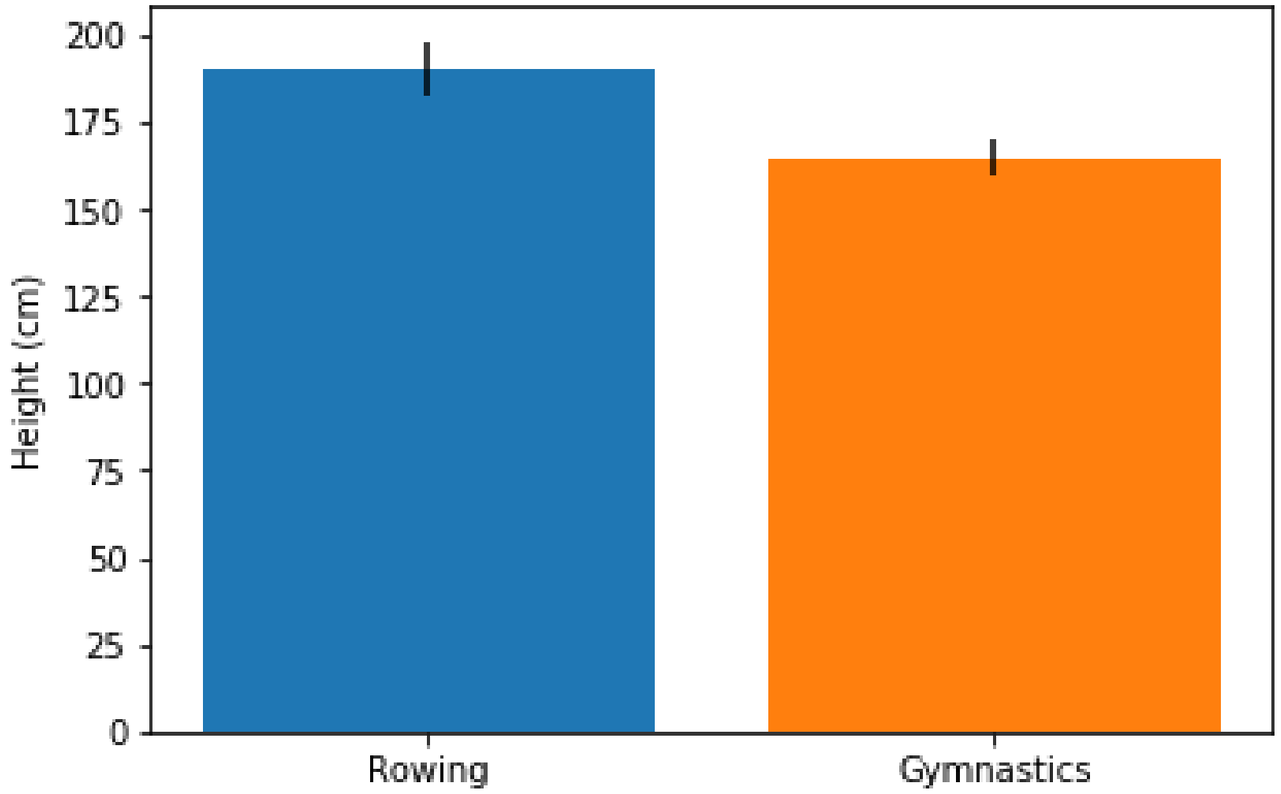
datacamp

Adding error bars to bar charts

```
fig, ax = plt.subplots()

ax.bar("Rowing",
       mens_rowing["Height"].mean(),
       yerr=mens_rowing["Height"].std())

ax.bar("Gymnastics",
       mens_gymnastics["Height"].mean(),
       yerr=mens_gymnastics["Height"].std())

ax.set_ylabel("Height (cm)")

plt.show()
```

2. Adding error bars to bar charts
The first is the use of error bars in plots. These are additional markers on a plot or bar chart that tell us something about the distribution of the data. Histograms, that you have seen in the previous lesson, show the entire distribution. Error bars instead summarize the distribution of the data in one number, such as the standard deviation of the values. To demonstrate this, we'll use the data about heights of medalists in the 2016 Olympic Games. There are at least two different ways to display error bars. Here, we add the error bar as an argument to a bar chart. Each call to the ax-dot-bar method takes an x argument and a y argument. In this case, y is the mean of the "Height" column. The yerr key-word argument takes an additional number. In this case, the standard deviation of the "Height" column, and displays that as an additional vertical marker.

Error bars in a bar chart



3. Error bars in a bar chart
Here is the plot. It is helpful because it summarizes the full distribution that you saw in the histograms in two numbers: the mean value, and the spread of values, quantified as the standard deviation.

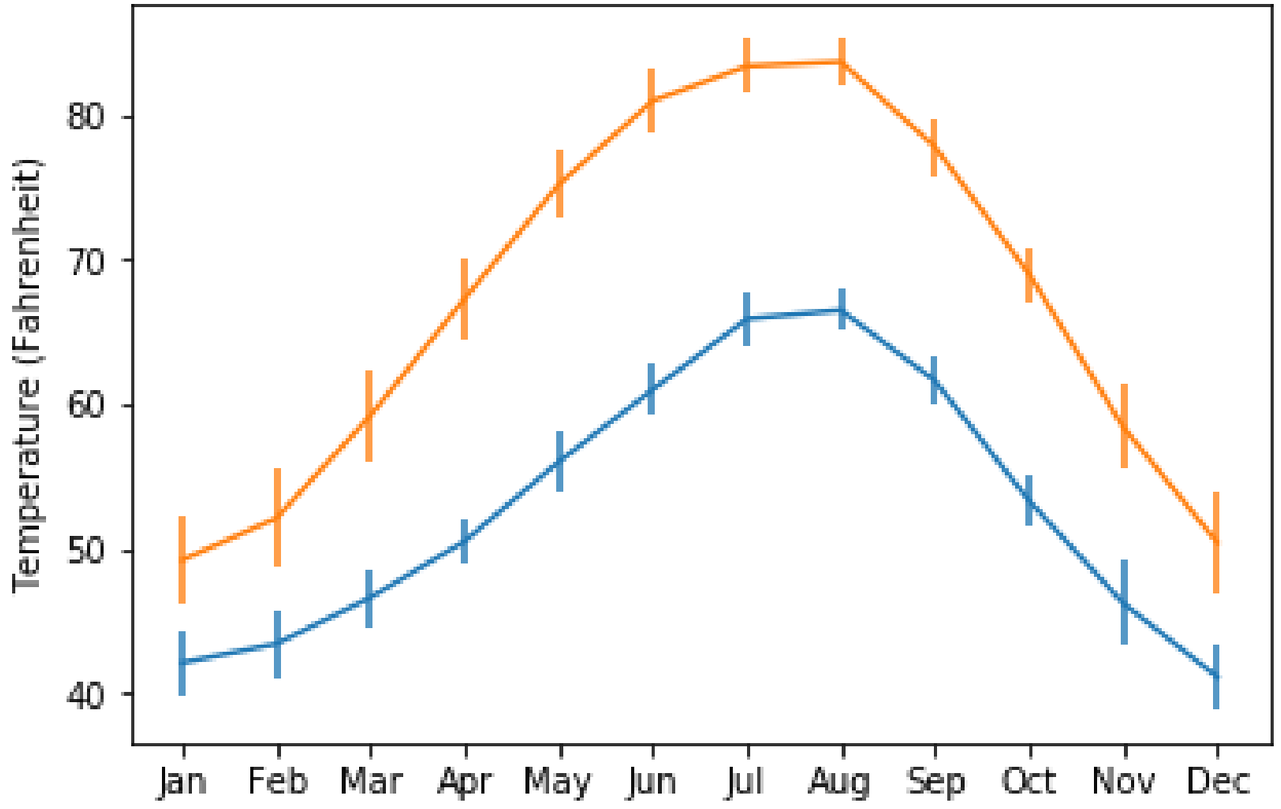
Adding error bars to plots

```
fig, ax = plt.subplots()

ax.errorbar(seattle_weather["MONTH"],
            seattle_weather["MLY-TAVG-NORMAL"],
            yerr=seattle_weather["MLY-TAVG-STDDEV"])

ax.errorbar(austin_weather["MONTH"],
            austin_weather["MLY-TAVG-NORMAL"],
            yerr=austin_weather["MLY-TAVG-STDDEV"])

ax.set_ylabel("Temperature (Fahrenheit)")

plt.show()
```

4. Adding error bars to plots

We can also add error bars to a line plot. For example, let's look at the weather data that we used in the first chapter of this course. To plot this data with error bars, we will use the Axes errorbar method. Like the plot method, this method takes a sequence of x values, in this case, the "MONTH" column, and a sequence of y values, in this case, the column with the normal average monthly temperatures. In addition, a yerr key-word argument can take the column in the data that contains the standard deviations of the average monthly temperatures.

Error bars in plots



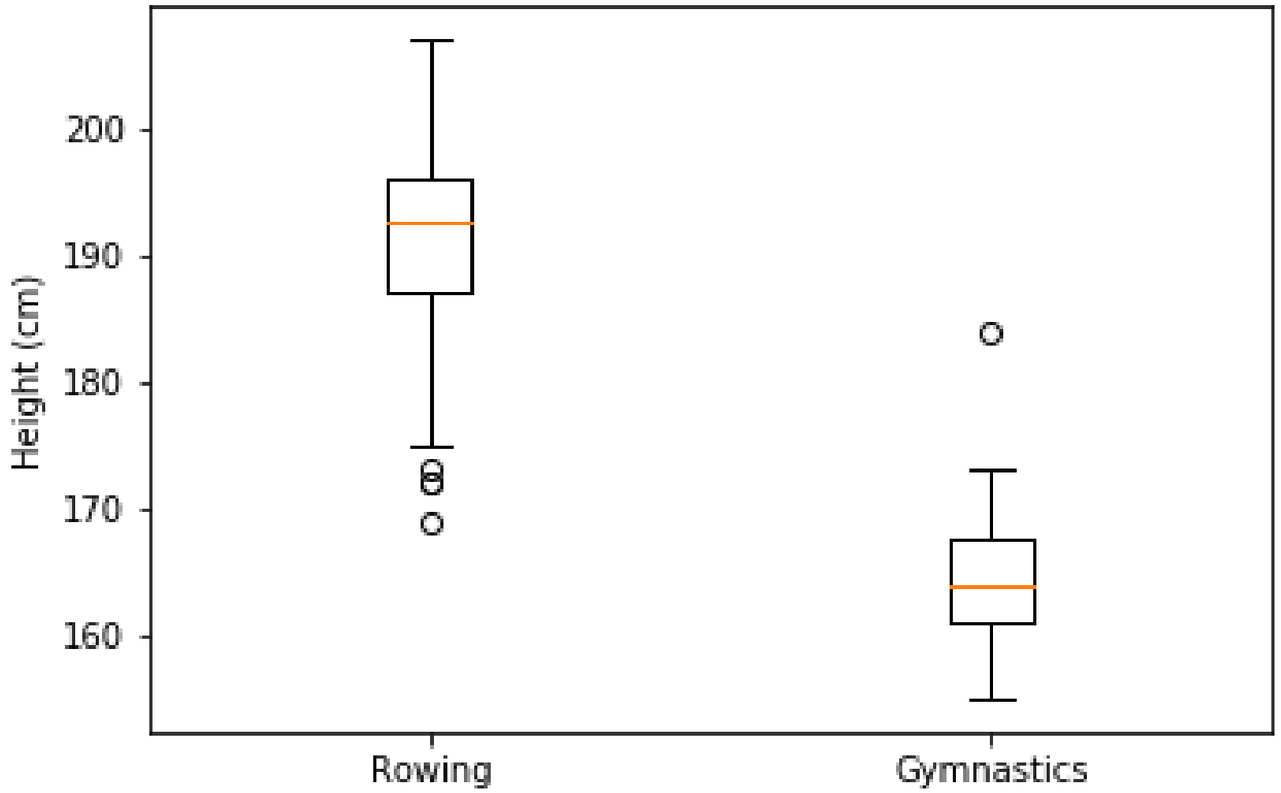
5. Error bars in plots
Similar to before, this adds vertical markers to the plot, which look like this.

Adding boxplots

```
fig, ax = plt.subplots()
ax.boxplot([mens_rowing["Height"],
            mens_gymnastics["Height"]])
ax.set_xticklabels(["Rowing", "Gymnastics"])
ax.set_ylabel("Height (cm)")

plt.show()
```

6. Adding boxplots
The second statistical visualization technique we will look at is the boxplot, a visualization technique invented by John Tukey, arguably the first data scientist. It is implemented as a method of the Axes object. We can call it with a sequence of sequences. In this case, we create a list with the men's rowing "Height" column and the men's gymnastics "Height" column and pass that list to the method. Because the box-plot doesn't know the labels on each of the variables, we add that separately, labeling the y-axis as well. Finally, we show the figure, which looks

Interpreting boxplots



7. Interpreting boxplots
like this. This kind of plot shows us several landmarks in each distribution. The red line indicates the median height. The edges of the box portion at the center indicate the inter-quartile range of the data, between the 25th and the 75th percentiles. The whiskers at the ends of the thin bars indicate **one and a half times the size of the inter-quartile range beyond the 75th and 25th percentiles.** This should encompass roughly 99 percent of the distribution if the data is Gaussian or normal. Points that appear beyond the whiskers are outliers. That means that they have values larger or smaller than what you would expect for 99 percent of the data in a Gaussian or normal distribution. For example, there are three unusually short rowers in this sample, and one unusually high gymnast.

Try it yourself!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Quantitative comparisons: scatter plots

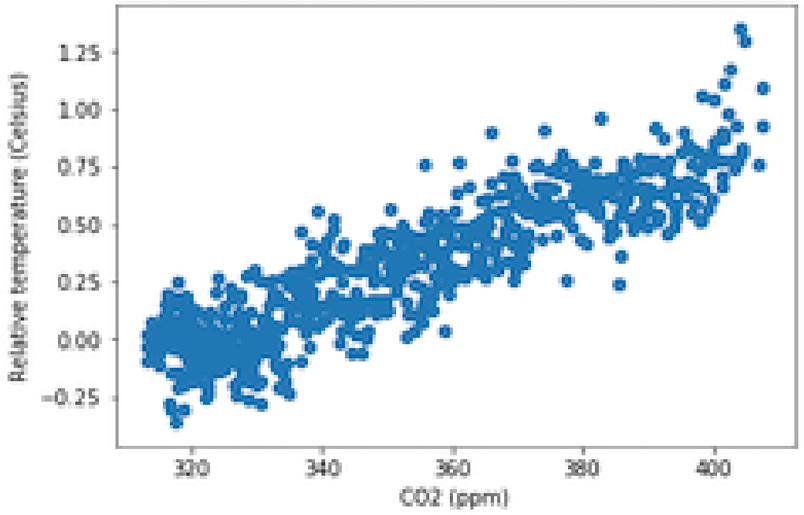
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

datacamp

Introducing scatter plots

```
fig, ax = plt.subplots()
ax.scatter(climate_change["co2"], climate_change["relative_temp"])
ax.set_xlabel("CO2 (ppm)")
ax.set_ylabel("Relative temperature (Celsius)")
plt.show()
```



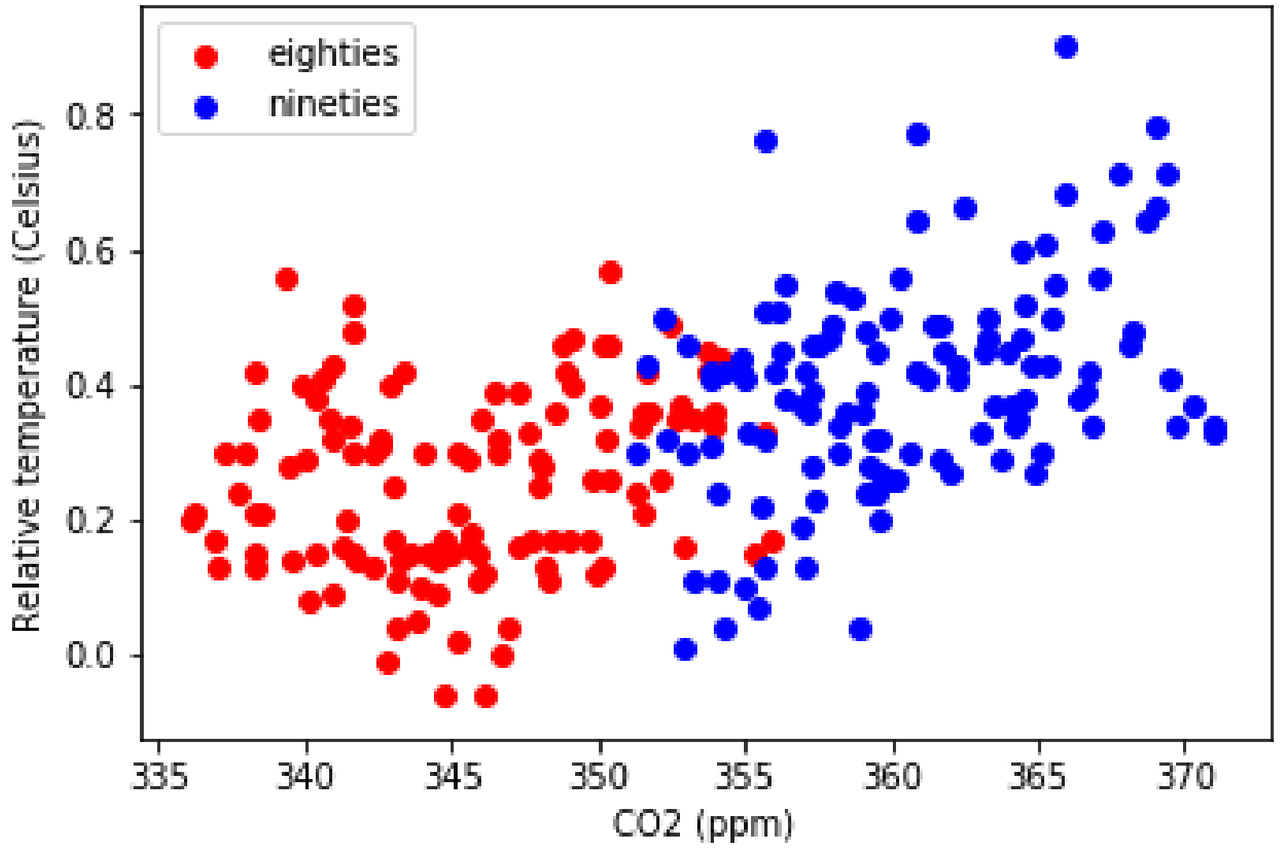
2. Introducing scatter plots
A standard visualization for bi-variate comparisons is a scatter plot. Let's look at an example. We'll use the climate change data that we have used previously. Recall that this dataset has a column with measurements of carbon dioxide and a column with concurrent measurements of the relative temperature. Because these measurements are paired up in this way, we can represent each measurement as a point, with the distance along the x-axis representing the measurement in one column and the height on the y-axis representing the measurement in the other column. To create this plot, we initialize a Figure and Axes objects and call the Axes scatter method. The first argument to this method will correspond to the distance along the x-axis and the second argument will correspond to the height along the y-axis. We also set the x-axis and y-axis labels, so that we can tell how to interpret the plot and call plt-dot-show to display the figure.

Customizing scatter plots

```
eighties = climate_change["1980-01-01":"1989-12-31"]
nineties = climate_change["1990-01-01":"1999-12-31"]
fig, ax = plt.subplots()
ax.scatter(eighties["co2"], eighties["relative_temp"],
           color="red", label="eighties")
ax.scatter(nineties["co2"], nineties["relative_temp"],
           color="blue", label="nineties")
ax.legend()

ax.set_xlabel("CO2 (ppm)")
ax.set_ylabel("Relative temperature (Celsius)")

plt.show()
```

3. Customizing scatter plots
We can customize scatter plots in a manner that is similar to the customization that we introduced in other plots. For example, if we want to show two bivariate comparisons side-by-side, we want to make sure that they are visually distinct. Here, we are going to plot two scatter plots on the same axes. In one, we'll show the data from the nineteen-eighties and in the other, we'll show the data from the nineteen-nineties. We can select these parts of the data using the time-series indexing that you've seen before to create two DataFrames called eighties and nineties. Then, we add each one of these DataFrames into the Axes object. First, we add the data from the eighties. We add customization: we set the color of the points to be red and we label these data with the string "eighties". Then, we add the data from the nineties. These points will be blue and we label them with the string "nineties". We call the legend method to add a legend that will tell us which DataFrame is identified with which color, we add the axis labels and call plt-dot-show.

Encoding a comparison by color



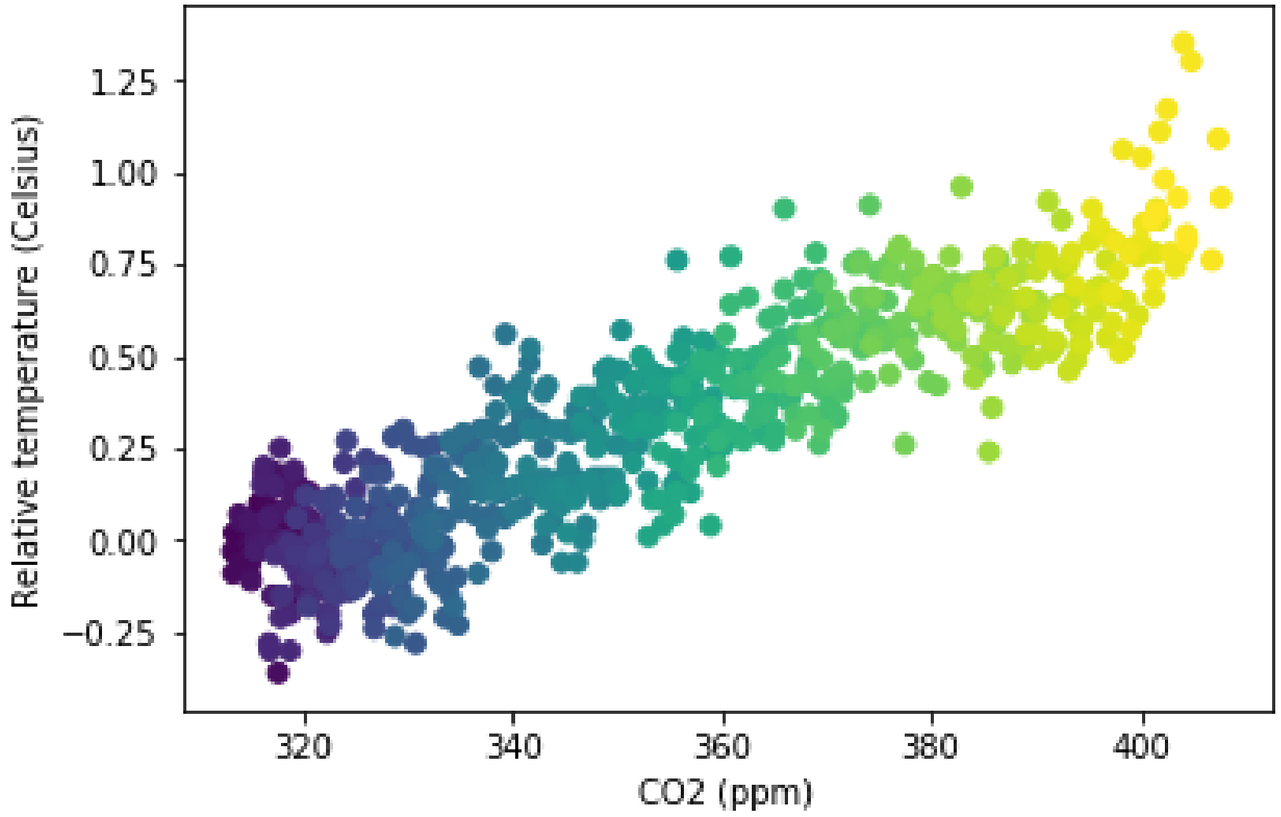
4. Encoding a comparison by color
This is what this figure looks like. You can see that the relationship between temperatures and carbon dioxide didn't change much during these years, but both levels of carbon dioxide and temperatures continued to rise in the nineties. Color can be used for a comparison, as we did here.

Encoding a third variable by color

```
fig, ax = plt.subplots()
ax.scatter(climate_change["co2"], climate_change["relative_temp"],
           c=climate_change.index)
ax.set_xlabel("CO2 (ppm)")
ax.set_ylabel("Relative temperature (Celsius)")
plt.show()
```

5. Encoding a third variable by color
But we can also use the color of the points to encode a third variable, providing additional information about the comparison. In the climate change data, we have a continuous variable denoting time stored in the DataFrame index. If we enter the index as input to the c key-word argument, this variable will get encoded as color. Note that this is not the color key-word argument that we used before, but is instead just the letter c. As before, we set the axis labels and call plt-dot-show.

Encoding time in color



6. Encoding time in color
Now, time of the measurements is encoded in the brightness of the color applied to the points, with dark blue points early on and later points in bright yellow.

Practice making your own scatter plots!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB