Lesson 5

July 16, 2020

1 Lesson 5: Data Visualization I

1.1 Big Data

When we were working with Mendel's peas, we were only looking at a few numbers at a time. However, in science, we often get lots and lots of numbers that we want to analyze, and computation enables us to do so. Computation also allows us to create beautiful visualizations of patterns in data that would be impossible to generate by hand. In this part of the lesson, we're going to show you some simple examples of how we can use our newfound Python skills to visualize a large amount of data very quickly.

One example of an area that generates lots of data is human phenotypes. For example, scientists have worked to aggregate human height and weight data to study things like growth and obesity. We have downloaded a publicly available dataset from UCLA that we can work with for this lesson (containing data from 25,000 individuals!).

1.2 Packages and Data Import

Instead of writing our own code to deal with large data sets, we will use what we refer to as a Python "package." Packages are collections of code someone else has already written to perform complicated tasks that you can then use so you don't have to reinvent the wheel! There are many packages available to serve all sorts of applications and needs.

1.2.1 Importing packages

It's not super important for you to understand the details of how package importing works, but we'll show you what it looks like. Since there are so many packages, we have to tell Python which ones, if any, we want to make available to ourselves while working on our program. Therefore, importing a package just tells Python that we want to use that particular package in our code.

Here, we're importing a package called Pandas. Pandas contains many functions that aid in loading data and performing analysis. We could, of course, spend a lot of time re-writing complicated functions that give the same end-result, but why re-invent the wheel when one simple line of code can import all of these functions for us to use!

1.2.2 Storing & analyzing data

Now we're going to load the data, using the Pandas package we just imported. Again, the syntax is not important for you to understand right now. We're going to read in a file of the heights and weights of 25,000 people and put it in an object called human_data. You can look at the raw data

file, which is stored in the second tab of the coding console, titled HumanHeightWeightData.csv (you can also download the data to take a look at it yourself by clicking on the file name). As you can probably surmise by looking at the data, the filetype "csv" stands for "comma separated values."

So what exactly did we just create? Let's take a look. Using the head() command, you can see the top of the file you just loaded.

Note: Because we're loading multiple packages and a large data set, please allow some time for the code to run (up to 1 minute).

[4]:
```
import pandas # this package is for working with large data sets
import numpy # this packages contains statistical functions

# Load data
human_data = pandas.read_csv('HumanHeightWeightData.csv')

# Using the pandas head() function, you were able to see the top of the file␣
 ↪you just loaded
print(human_data.head())

# The shape of the matrix stored in the human_data variable can be assessed␣
 ↪with the pandas shape command
print(human_data.shape)
```

```
   height_inches  weight_pounds
0       65.78331       112.9925
1       71.51521       136.4873
2       69.39874       153.0269
3       68.21660       142.3354
4       67.78781       144.2971
(25000, 2)
```

Using the shape property, you can see how big the file is. The first number (25000) is the number of rows (in this data, each row is one individual person), and the second number (2) is the number of columns, which makes sense because we know that there are two columns - one for height and one for weight.

1.3 Average height and weight

Because the data take so long to load, we'll be working with a smaller file for the rest of the lesson (only 1000 rows). However, don't think that computers can't work with big data sets; it's only because the file needs to be loaded into each window that it seems too slow. Normally, you only have to load the data once, and then you can perform all of your calculations relatively quickly!

Now, try going to the website the data came from and see if you can figure out what the average height and weight of these individuals is… we bet you can't! Let's use Python to calculate the mean in fractions of a second using the mean property!

[3]:
```
import pandas # this package is for working with large data sets
import numpy # this packages contains statistical functions

# Load smaller data set
human_data = pandas.read_csv('HumanHeightWeightData_1000.csv')

# Calculate average height and weight of all of the individuals included in the␣
 ↪dataset using the numpy function mean()
print(numpy.mean(human_data.height_inches)) # Average height
print(numpy.mean(human_data.weight_pounds)) # Average weight
```

```
67.9931135968
127.07942116080001
```

1.4 Histograms

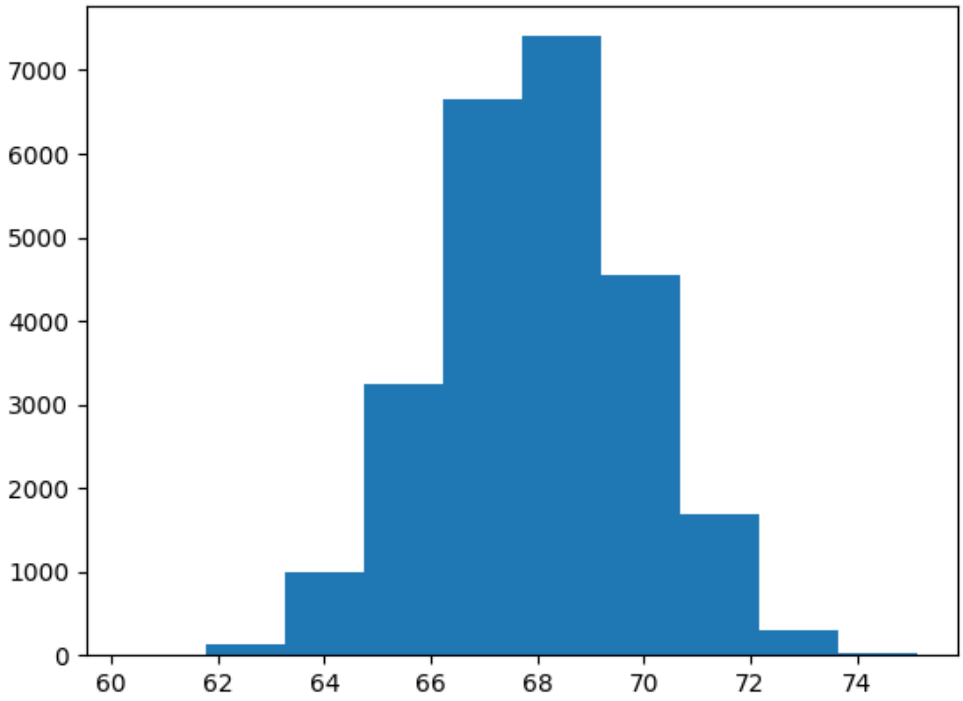
We can use Python to start to get a sense of what our data actually looks like.

One tool for visualizing data quickly is a graph called a histogram. Histograms display the frequency of different values in your data so that you can see which values are most and least common. Histograms are essentially bar plots where the x-axis represents the values in your data and the y-axis represents counts (i.e. how many of each value we observed in our data).

Histograms in Python are super simple! It's just a few lines of code. Let's make one to look at the distribution of height in our data set. In addition to the Pandas package we used earlier, we're going to use another package (Matplotlib) to help us make visualizations like this one.

[ ]:
```
import pandas # this package is for working with large data sets
import matplotlib.pyplot as plt # this package allows us to graph data

# Load data
human_data = pandas.read_csv('HumanHeightWeightData_1000.csv')

# Create a histogram of human height data
fig1 = plt.figure() # this tells Python that we are about to create a graph
plt.hist(human_data.height_inches) # creates a histogram plot of human height␣
 ↪data
fig1.savefig('HeightHistogram1.png') # saves the histogram as a picture in a␣
 ↪file called "HeightHistogram1.png"
```

In order to view the histogram plot created by this code, click the third tab that should appear upon running the code titled HeightHistogram1.png (in order to see this tab, you might have to click the icon that looks like a page with the corner folded in the left side of the terminal). It should look like this:



We call the shape of the histogram a "distribution." The x-axis represents different values of height in the data, ranging from 60 inches to 76 inches. The y-axis tells us how many people in the data set were measured in each range of height (e.g. 62-64 inches, 64-66 inches, etc.). For example, we can see that there are about 25 people measured between 63 and 65 inches.

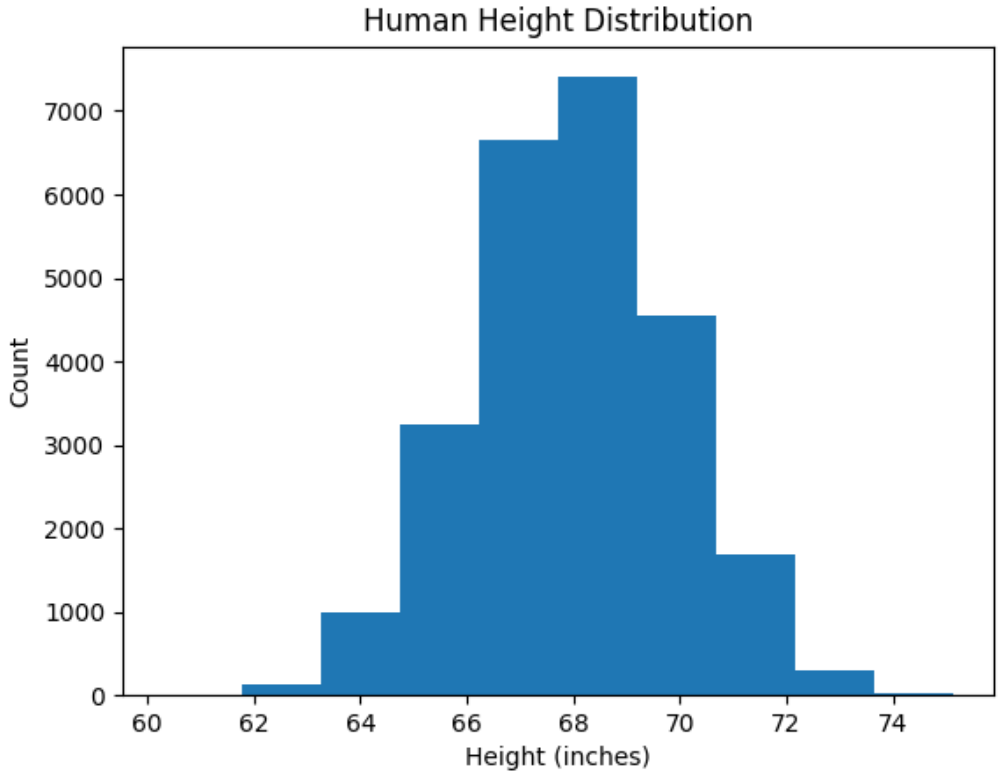
1.5 Polishing Histograms

Let's make this graph more descriptive. It might help to have a title, x-axis label, and y-axis label so we know what we're looking at so that if you showed it to someone else, they would understand it better.

[ ]:
```
import pandas # this package is for working with large data sets
import matplotlib.pyplot as plt # this package allows us to graph data

# Load data
human_data = pandas.read_csv('HumanHeightWeightData_1000.csv')

# Create a histogram of human height data
fig2 = plt.figure()
plt.hist(human_data.height_inches)
plt.title('Human Height Distribution') # this is the title of the graph (note␣
 ↪it's a string)
plt.xlabel('Height (inches)') # this is the label for the x-axis
```

```
plt.ylabel('Count') # this is the label for the y-axis
fig2.savefig('HeightHistogram2.png')
```

It should be found in a tab titled HeightHistogram2.png and look like this:



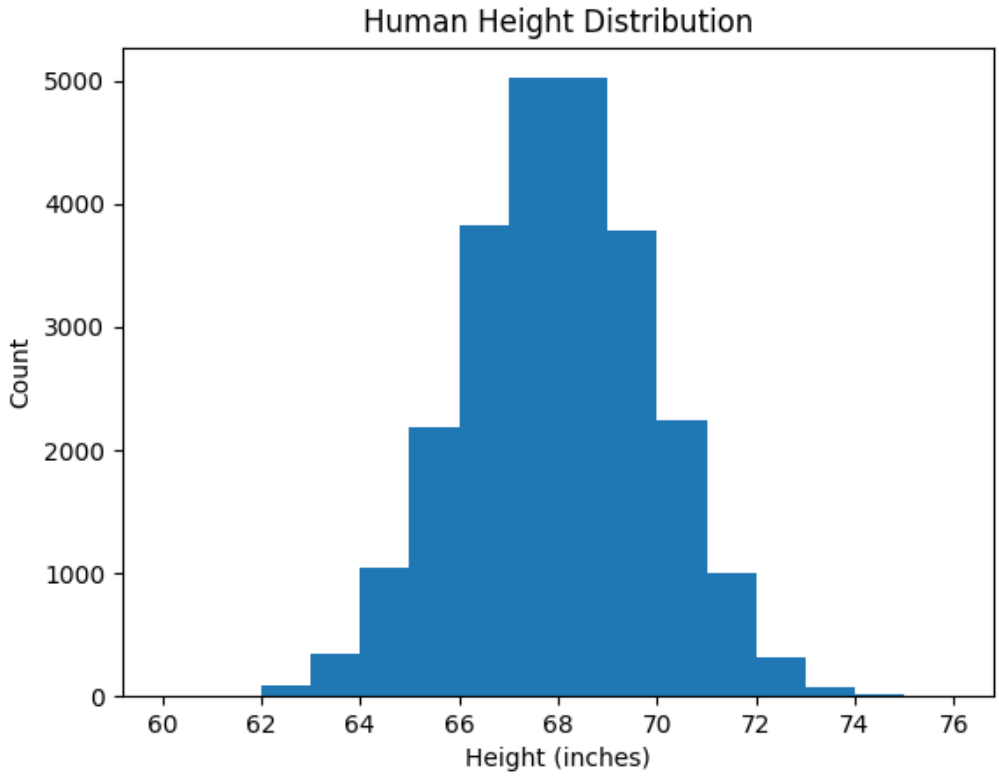
Here we have about ~2 inch resolution, so we can see how many people fall into each 2-inch range. We call these ranges "bins." What if instead we wanted to see how many people fall into each 1-inch bin (e.g. how many people are between 66 inches and 67 inches)?

We can do that by changing the number of bins in the histogram. In the previous histogram, there were 10 bins. Now, let's make 16 bins (since our range is between 60 inches and 76 inches, and 76 - 60 = 16). We will also specify our range of between 60 and 76 to ensure that the bins each get drawn containing exactly one inch each.

[ ]:
```
import pandas # this package is for working with large data sets
import matplotlib.pyplot as plt # this package allows us to graph data

# Load data
human_data = pandas.read_csv('HumanHeightWeightData_1000.csv')

# Create a histogram of human height data
fig3 = plt.figure()
```

```
plt.hist(human_data.height_inches, bins = 16, range = [60,76]) # now there are␣
↪16 bins
plt.title('Human Height Distribution')
plt.xlabel('Height (inches)')
plt.ylabel('Count')
fig3.savefig('HeightHistogram3.png')
```

The resulting graph should be found in a tab titled HeightHistogram3.png and look like this:



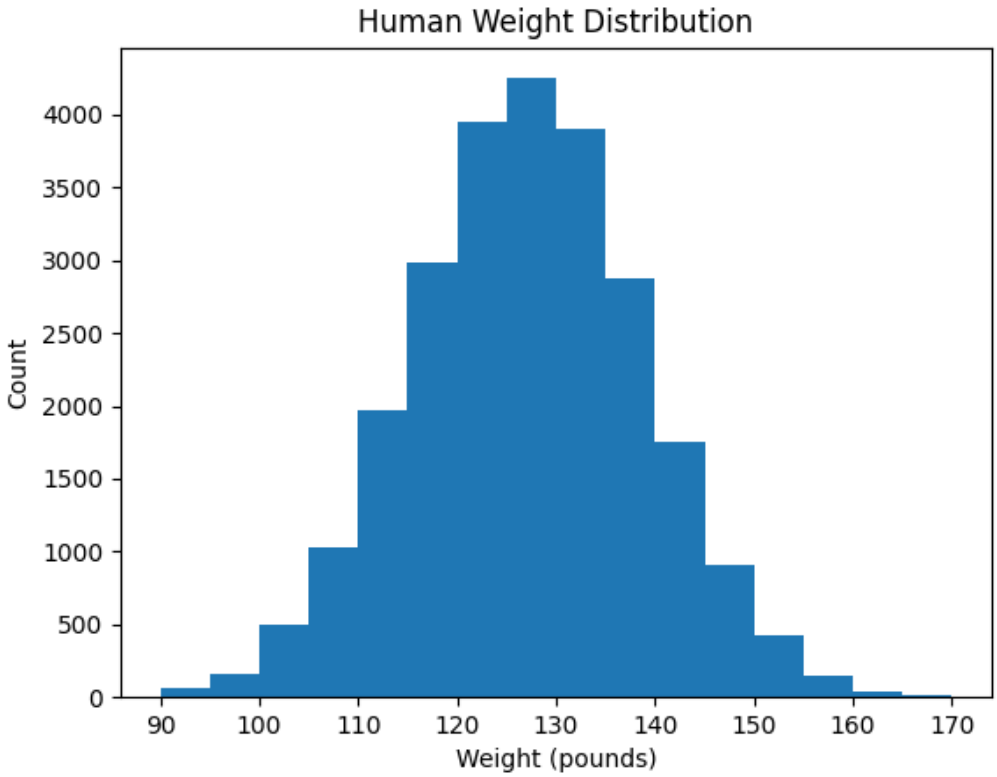
Notice that when we use this breakdown, the symmetry of the distribution is even more obvious than it was before. This is a good lesson in the importance of data visualization and that your interpretation can be subject to slight changes in your visualization method.

1.6 Checkpoint #1

Make a histogram with labeled axes and a title, where each bin represents 5 pounds to show the distribution of human weight.

[ ]:
```
import pandas # this package is for working with large data sets
import matplotlib.pyplot as plt # this package allows us to graph data

# Load data
human_data = pandas.read_csv('HumanHeightWeightData_1000.csv')
```

```
# Create a histogram of human height data
fig4 = plt.figure()
plt.hist(human_data.weight_pounds, bins = 16, range = [90,170])
plt.title('Human Weight Distribution')
plt.xlabel('Weight (pounds)')
plt.ylabel('Count')
fig4.savefig('WeightHistogram1.png')
```



1.7 Summary Statistics

We already discussed the average (mean) earlier, which is the sum of all of the values divided by the number of values. The mean can be a useful tool for quickly summarize your data. However, the mean is an incomplete picture of what your distribution actually looks like. Specifically, it gives a sense of where the center of the data is, but not the spread of the data (how widely the values range).

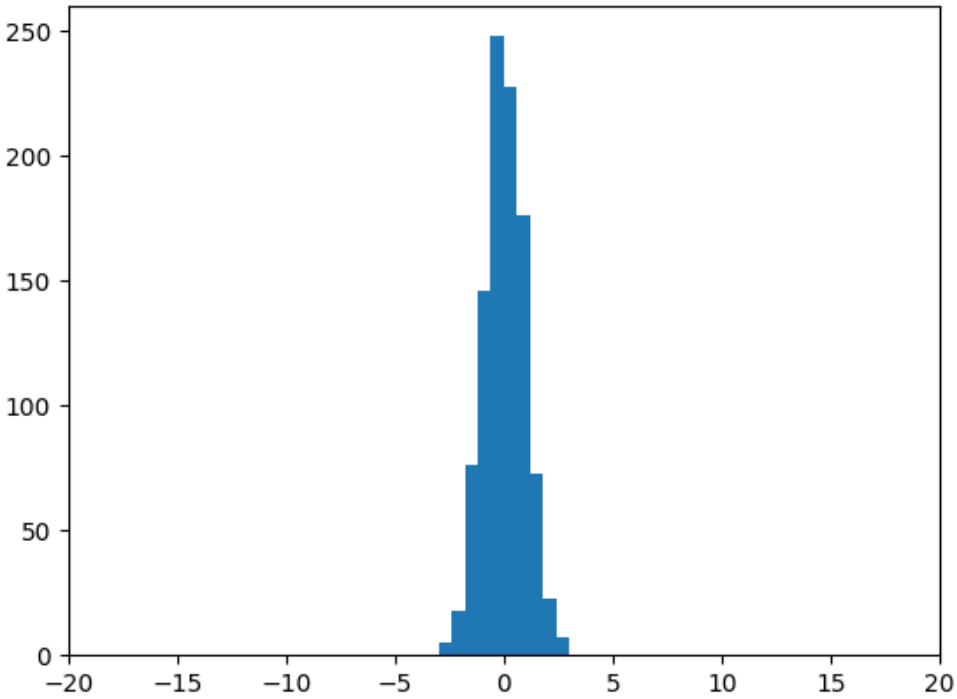
Why is range important? Imagine living in a climate where the mean temperature year-round is 65 degrees Fahrenheit, and it's almost never below 60 or above 70 degrees. Compare that to living in a climate where the mean temperature year-round is 65 degrees, but it can go as low as 0 or as high as 100. Though the means are the same, these climates are obviously very different!

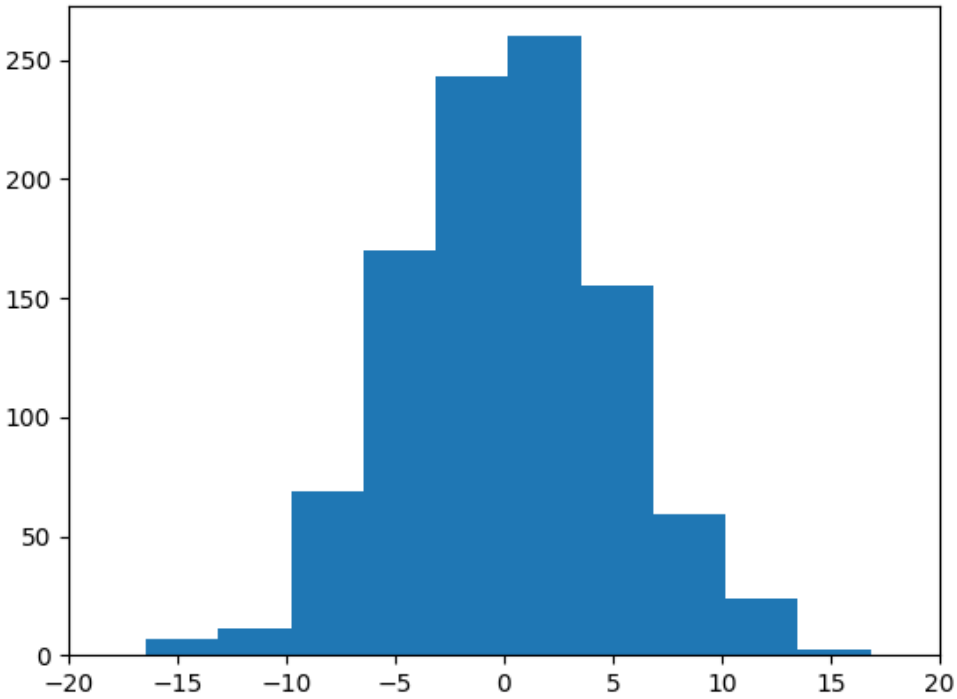
We can create histograms with the same mean (0) but that look very different with similar lines of code. The code for the other two histograms is very similar, but only differs in the value of the

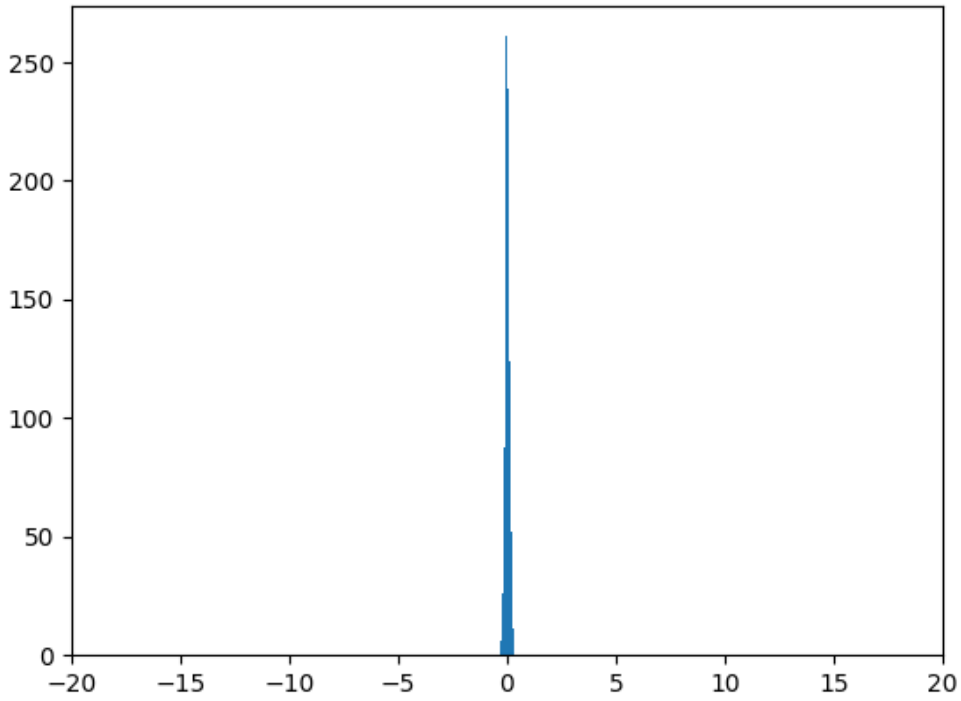
scale parameter in the second line of code (equal to 1, 5, or 0.1 in each graph, respectively).

[ ]:
```
import numpy # this package allows us to use some statistical functions
import matplotlib.pyplot as plt # this package allows us to graph data

# Create a histogram with mean 0 and standard deviation 1
fig1 = plt.figure()
plt.hist(numpy.random.normal(loc = 0, scale = 1, size = 1000)) # Create a␣
 ↪histogram with data (size, 1000 data points) that has mean (loc) 0 and␣
 ↪standard deviation (scale) 1
plt.xlim(-20,20) # Set the x axis to range from -20 to 20 to visualize full␣
 ↪distribution and to aid in comparison to other histograms
fig1.savefig('Histogram[0,1].png')

# Create a histogram with mean 0 and standard deviation 5
fig2 = plt.figure()
plt.hist(numpy.random.normal(loc = 0, scale = 5, size = 1000)) # Create a␣
 ↪histogram with data (size, 1000 data points) that has mean (loc) 0 and␣
 ↪standard deviation (scale) 5
plt.xlim(-20,20) # Set the x axis to range from -20 to 20 to visualize full␣
 ↪distribution and to aid in comparison to other histograms
fig2.savefig('Histogram[0,5].png')

# Create a histogram with mean 0 and standard deviation 0.1
fig3 = plt.figure()
plt.hist(numpy.random.normal(loc = 0, scale = 0.1, size = 1000)) # Create a␣
 ↪histogram with data (size, 1000 data points) that has mean (loc) 0 and␣
 ↪standard deviation (scale) 0.1
plt.xlim(-20,20) # Set the x axis to range from -20 to 20 to visualize full␣
 ↪distribution and to aid in comparison to other histograms
fig3.savefig('Histogram[0,0.1].png')
```

In order to view the three histograms plotted by this code, click the second, third, and fourth tabs that should appear upon running the code, titled Histogram[0,?].png. They should look like this:


250
200
150
100
50
0
−20
−15
−10
−5
0
5
10
15
20


250
200
150
100
50
0
−20
−15
−10
−5
0
5
10
15
20



The difference in spread can be quantified by something called the "standard deviation." For now, the details of how this is calculated don't matter. The intuition is that for each value, we calculate how far it is from the mean and then combine all those differences together into a single number (the standard deviation). Therefore, if many of the values are far from the mean, we get a big standard deviation, meaning that the spread of the values is high. Alternatively, if most of the values are close to the mean, we will get a small standard deviation, and that means that the spread is narrow.

2 Summary

In this lesson, we've learned the utility of packages in handling big data, as well as how import and use these packages. We've explored a real-world data set containing human height and weight values, learning and calculating summary statistics like mean and standard deviation. We've also learned how to create histograms in Python, visualiing the data in a number of ways.

Let's put everything together in one more exercise:

1. What is the temperature in January in Mobile, Alabama? Hint: Mobile is the first city listed in the data
2. How many cities are included in the data?
3. Make a histogram of temperatures in January across the United States. Be sure to give your graph a title and label both axes.

```
[ ]: import pandas
     import numpy
     import matplotlib.pyplot as plt

     # Load data
     city_temp_lat_data = pandas.read_csv('CityTempLat.csv')

     # See the top of the file
     print(city_temp_lat_data.head())

     # The shape of the matrix stored in the human_data variable can be assessed␣
      ↪with the pandas shape command
     print(city_temp_lat_data.shape)

     # Create a histogram of temperatures
     fig1 = plt.figure()
     plt.hist(city_temp_lat_data.JanTemp)
     plt.title('Temperatures Across the US in January')
     plt.xlabel('Temperature (˚F)')
     plt.ylabel('Count')
     fig4.savefig('TemperatureHistogram.png')
```