


[learn-co-curriculum](#) / [dsc-implementing-statistics-with-functions-lab](#) Public View license 0 stars  116 forks Star Watch ▾[Code](#) [Issues](#) [Pull requests](#) 1 [Actions](#) [Projects](#) [Security](#) [Insights](#) solution ▾

...

This branch is [26 commits ahead](#), [34 commits behind](#) master. Contribute ▾

Chester Ismay Fix height to height\_np typo ...

on Nov 16, 2021  31[View code](#) README.md

# Implementing Statistics with Functions - Lab

## Introduction

In this lab you'll dive deep into calculating the measures of central tendency and dispersion introduced in previous lessons. You will code the formulas for these functions in Python which will require you to use the programming skills that you have gained in the other lessons of this section. Let's get started!

## Objectives

You will be able to:

- Calculate the measures of dispersion for a dataset
- Compare the different measures of dispersion

- Calculate the measures of central tendency for a dataset
- Compare the different measures of central tendency

## Dataset

---

For this lab, we'll use the [NHIS dataset](#), which contains weights, heights, and some other attributes for a number of surveyed individuals. The context of this survey is outside the scope this lab, so we'll just go ahead and load the heights column as a list for us to run some simple statistical experiments. We'll use the `pandas` library to import the data into our Python environment. This process will be covered in detail in a later section. For now, we'll do this part for you to give you a head start.

Run the cell below to import the data.

```
import pandas as pd
df = pd.read_csv('nhis.csv')
height = list(df['height'])
```

We are only interested in the height column, so we saved it as a list in the variable `height` in the cell above.

In the cells below:

- Display the number of items in `height`
- Slice and display the first 10 items from `height`

```
num_records = len(height)
```

```
num_records
```

```
4785
```

```
first_10 = height[:10]
```

```
first_10
```

```
[74, 70, 61, 68, 66, 98, 99, 70, 65, 64]
```

So, around 4800 records of height. That's great. Next, we'll try plotting some basic *histograms* for these records.

## Plotting Histograms

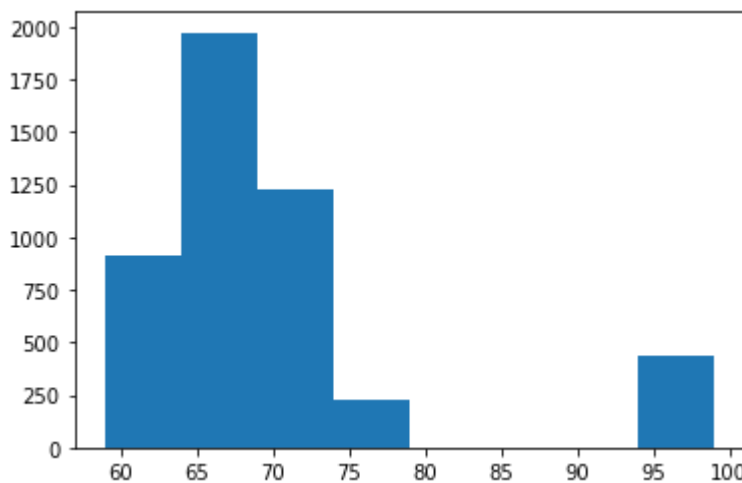
We'll begin by importing the `pyplot` module from the library `matplotlib` and setting an alias of `plt` for it (so that we only have to type `plt.` instead of `matplotlib.pyplot.` each time we want to use it). Note that `plt` is considered the *standard alias* for Matplotlib.

Run the cell below to import Matplotlib and use it to create a histogram of our `height` data with 8 different bins.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Next, we'll use Matplotlib to create a histogram by passing in our data, as well as the parameter `bins=8`, into the `hist` function.

```
# The most frequent , or central , or typical value is between 65 and 70.
# Around 400 individuals are extremely tall , nearing 8 feet
# The distribution seems to have OUTLIERS
plt.hist(height, bins=8);
```



Do you spot anything unusual above? Some outliers, maybe?

## Measures of Central Tendency

### Calculating the Mean

We're just beginning to dig into the data stored in `height`. We'll begin by writing a function to calculate the mean of the data. Recall the formula for calculating mean:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Using the Python skills you have learned so far, create a function `get_mean()` to perform the following tasks:

- Input a list of numbers (like the height list we have above)
- Calculate the sum of numbers and length of the list
- Calculate mean from above, round off to 2 decimals and return it.

```
def get_mean(data):  
    mean = sum(data)/len(data)  
  
    return round(mean, 2)
```

```
test1 = [5, 4, 1, 3, 2]  
test2 = [4, 2, 3, 1]
```

```
print(get_mean(test1))  
print(get_mean(test2))
```

```
3.0  
2.5
```

Now, we'll test the function by passing in the height list.

```
mean = get_mean(height)  
  
print("Sample Mean:", mean)
```

```
Sample Mean: 69.58
```

So, we have our mean length, 69.58, and this confirms our observations from the histogram. But we also have some outliers in our data above and we know outliers affect the mean calculation by pulling the mean value in their direction. So, let's remove these outliers and create a new list to see if our mean shifts or stays. We'll use a threshold of 80 inches, i.e. filter out any values greater than 80.

Perform following tasks:

- Create a function `filter_height_outliers` that takes a list as an argument
- Perform a `for` loop to iteratively check and append values to a new list if the value is less than 80, for every element in the original list
- Return the new list

```
def filter_height_outliers(data):  
  
    filtered_data = []  
  
    for height in data:  
        if height < 80:  
            filtered_data.append(height)  
  
    return filtered_data  
  
test = [60, 70, 80, 90]  
filter_height_outliers(test)
```

```
[60, 70]
```

Great, now we can use `filter_height_outliers()` to filter our `height` list and plot a new histogram to see if things change considerably.

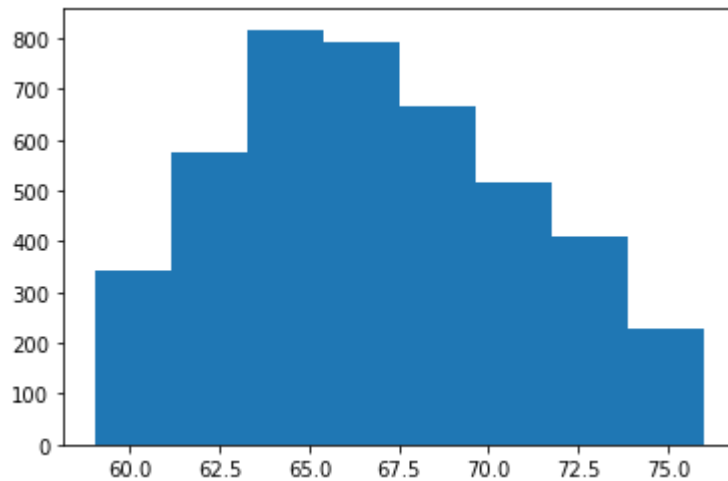
```
filtered_height = filter_height_outliers(height)  
  
len(filtered_height)
```

```
4347
```

Now that we have filtered the outliers out of our data and reduced the size of the dataset from 4785 to 4347, let's recreate our histogram with 8 bins using our filtered data.

**NOTE:** You do not need to reimport `matplotlib.pyplot` as `plt` -- once it's been imported, it's stored in memory and can be accessed whenever we like in other cells.

```
# Plot a histogram for the new list - use 8 bins as before  
plt.hist(filtered_height, bins = 8);
```



Since we've filtered our data to remove outliers, we should also recalculate the mean. Do this now in the cell below, using our `get_mean()` function.

```
new_mean = get_mean(filtered_height)
```

```
new_mean
```

```
66.85
```

Does the mean height of our filtered data match up with what we see in our histogram of our filtered data?

Note that in some analytical situations we may not be able to exclude the outliers in such a naive manner. So, let's go ahead and calculate other measures of central tendency as well. We'll start by calculating the median value for our original (unfiltered) height data.

## Calculating the Median

The median is the value directly in the middle of the dataset. In statistical terms, this is the **Median Quartile**. If the dataset was sorted from lowest value to highest value, the median is the value that would be larger than the first 50% of the data, and smaller than the second 50%.

If the dataset has an odd number of values, then the median is the middle number. If the dataset has an even number of values, then we take the mean of the middle two numbers.

In the cell below, write a function that takes in a list of numbers and returns the median value for that dataset. Make sure you first check for even / odd number of data points and perform the computation accordingly. The best approach to calculate the median is as follows:

1. Sort the data
2. Check if the data has even or odd number of data points
3. Calculate the median of the sorted data now that you know if the count is even or odd.

Hints:

- You can use the modulo operator `%` in Python to check if a value is even or odd -- odd numbers `% 2` (e.g. `5 % 2`) will equal `1`, while even numbers `% 2` (e.g. `4 % 2`) will equal `0`!
- You can use integer division `//` to calculate the index -- for even numbers this just means that the result is an integer (e.g. `4 // 2` is `2` rather than `2.0`), while for odd numbers this means that the remainder is cut off (e.g. `7 // 2` is `3`, not `3.5`)

```
"""
```

```
How this algorithm works with the provided examples:
```

```
1. [5, 4, 1, 3, 2]
```

```
Sorted, this is [1, 2, 3, 4, 5]
```

```
The count is 5
```

```
5 % 2 is 1, so the block below the else statement runs
```

```
5 // 2 is 2
```

```
Return the value at index 2, which is 3 (the middle one, we are zero-indexed)
```

```
2. [4, 2, 3, 1]
```

```
Sorted, this is [1, 2, 3, 4]
```

```
The count is 4
```

```
4 % 2 is 0, so the block below the if statement runs
```

```
Right: 4 // 2 is 2. The value at index 2 is 3
```

```
Left: 2 - 1 is 1. The value at index 1 is 2
```

```
The value at index 2 is 3 (right), the value at index 1 is 2 (left)
```

```
We return the average of 3 and 2 (right and left of the middle), which is 2.5
```

```
"""
```

```
def get_median(data):
```

```
    data_sorted = sorted(data)
```

```
count = len(data_sorted)

if count % 2 == 0:
    # If count is even, return the average of the middle two numbers
    right_index = count // 2
    left_index = right_index - 1
    return (data_sorted[left_index] + data_sorted[right_index]) / 2

else:
    # If count is odd, return the middle number
    med_index = count // 2
    return data_sorted[med_index]

test1 = [5, 4, 1, 3, 2]
test2 = [4, 2, 3, 1]

print(get_median(test1))
print(get_median(test2))

3
2.5
```

Great, now we can pass in our original `height` list to this function to check the median.

```
median = get_median(height)

median

67
```

So, we have 67, which is much closer to the filtered list mean (66.85) than the mean we calculated with actual list (69.58). So, median in this case seems to be a much better indicator of the central tendency found in the dataset. This makes sense because we've already learned that medians are less sensitive to outliers than mean values are!

Next, we'll calculate the mode. This could give us better insight into the typical values in the dataset based on how frequent a value is.

## Calculating the Mode



The mode is the value that shows up the most in a dataset. A dataset can have 0 or more modes. If no value shows up more than once, the dataset is considered to have no mode value. If two numbers show up the same number of times, that dataset is considered bimodal. Datasets where multiple values all show up the same number of times are considered multimodal.

In the cell below, write a function that takes in a list of numbers and returns another list containing the mode value(s). In the case of only one mode, the list would have a single element.

*Hint:* Building a **frequency distribution** table using dictionaries is probably the easiest way to approach this problem. Use each unique element from the height list as a key, and the frequency of this element as the value and build a dictionary. You can then simply identify the keys (heights) with maximum values.

```
def get_mode(data):

    # Create and populate frequency distribution
    frequency_dict = {}

    for height in data:
        # If an element is not in the dict, add it to the dict with value 1
        if height not in frequency_dict:
            frequency_dict[height] = 1
        # If an element is already in the dict, +1 the value in place
        else:
            frequency_dict[height] += 1

    # Find the frequency of the mode(s) by finding the largest
    # value in frequency_dict
    highest_freq = max(frequency_dict.values())

    # Create a list for mode values
    modes = []

    # From the dictionary, add element(s) to the modes list with max frequency
    for height, freq in frequency_dict.items():
        if freq == highest_freq:
            modes.append(height)

    # Return the mode list
    return modes

test1 = [1, 2, 3, 5, 5, 4]
test2 = [1, 1, 1, 2, 3, 4, 5, 5, 5]
```

```
print(get_mode(test1))  
print(get_mode(test2))
```

```
[5]  
[1, 5]
```

That's done. Now you can use the above function to calculate the mode of the original `height` list to compare it with our mean and median values.

```
mode = get_mode(height)
```

```
mode
```

```
[64]
```

So, the mode value is much lower than our mean and median calculated earlier. What do you make of this? The answer to that could be subjective and depends on the problem. i.e. if your problem is to identify sizes for garments that would sell the most, you cannot disregard mode. However, if you want to get an idea about the general or typical height of individuals, you can probably still do that with the median and the average.

To get an even clearer picture, we know we need to see how much the values deviate from the central values we have identified. We have seen variance and standard deviation before as measures of such dispersion. Let's have a go at these to strengthen our understanding of this data.

## Measures of Dispersion

---

### Calculating the Variance

The formula for variance is:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

Note that this formula is for the **sample** variance. The formula is slightly different than the formula for calculating population variance. Read more about the difference [here](#). In the cell below, write a function that takes a list of numbers as input and returns the variance (rounded to two decimal places) of the sample as output.

```
def get_variance(sample):  
  
    # First, calculate the sample mean using get_mean()  
    sample_mean = get_mean(sample)  
  
    sum_of_squares = 0  
    for height in sample:  
        # Now, calculate the sum of squares by subtracting the sample mean  
        # from each height, squaring the result, and adding it to the total  
        sum_of_squares += (height - sample_mean)**2  
  
    # Divide the total by the number of items in the sample -1  
    variance = sum_of_squares / (len(sample) - 1)  
  
    return round(variance, 2)  
  
test1 = [1, 2, 3, 5, 5, 4]  
test2 = [1, 1, 1, 2, 3, 4, 5, 5, 5]  
print(get_variance(test1))  
print(get_mean(test1))  
print(get_variance(test2))
```

2.67

3.33

3.25

Now we can test the variance of our list `height` with our new `get_variance()` function.

```
variance = get_variance(height)
```

```
variance
```

87.74

So this value, as we learned earlier, tells us a bit about the deviation but not in the units of underlying data. This is because it squares the values of deviations. Standard deviation, however, can deal with this issue as it takes the square roots of differences. So that would probably be a bit more revealing.

## Calculating the Standard Deviation

---

In the cell below, write a function that takes a list of numbers as input and returns the standard deviation of that sample as output.

Recall that the formula for Standard Deviation is:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

To find the square root of a value in Python, you have two options (**either** approach will work):

One option is the `sqrt()` function from `math` library:

```
from math import sqrt
sqrt(100) # 10.0
```

Alternatively, another approach would be to raise that number to the power of `0.5`:

```
100**0.5 # 10.0
```

```
from math import sqrt

def get_stddev(sample):

    stddev = sqrt(get_variance(sample))

    return round(stddev, 2)
```

```
test = [120,112,131,211,312,90]
```

```
get_stddev(test)
```

```
84.03
```

So now we can finally calculate the standard deviation for our `height` list and inspect the results.

```
standard_deviation = get_stddev(height)

standard_deviation
```

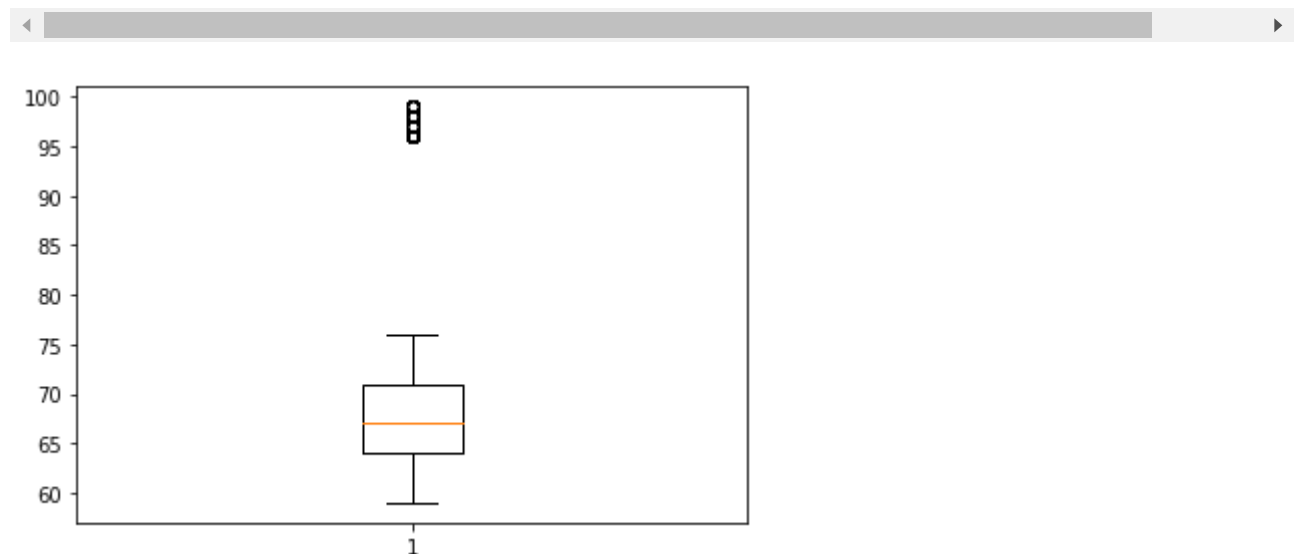
9.37

So 9.37 inches is the amount of deviation present in our dataset. As we are still including outlier values, this might be slightly affected but these results are now much more reliable.

Finally, we will build a boxplot for height data and see if it agrees with our understanding for this data that we have developed up to this point. Use the `matplotlib`'s `boxplot()` function with height data and comment on the output.

```
plt.boxplot(height);
```

```
# the median is at 67 inches and there are three outliers at the upper end of the sp
```



## Simplifying the Process with NumPy

We hope writing these functions was a useful experience in terms of deepening your understanding of these statistical measures as well as sharpening your Python skills. However in reality there is almost never a need to write these kinds of functions "by hand", since libraries like NumPy and SciPy can typically handle them for us in a single line.

Below is a demonstration of the same calculations performed above, written using Python libraries side-by-side with the results of the functions you've just written:

```
import numpy as np
from scipy import stats

print("Mean:")
```

```
print(mean, "(our version)")
print(round(np.mean(height), 2), "(NumPy version)")
print()
print("Median:")
print(median, "(our version)")
print(np.median(height), "(NumPy version)")
print()
print("Mode:")
print(mode, "(our version)")
print(stats.mode(height).mode, "(SciPy version)")
print()
print("Variance:")
print(variance, "(our version)")
print(round(np.var(height, ddof=1), 2), "(NumPy version)")
print()
print("Standard Deviation:")
print(standard_deviation, "(our version)")
print(round(np.std(height, ddof=1), 2), "(NumPy version)")
```

Mean:

69.58 (our version)

69.58 (NumPy version)

Median:

67 (our version)

67.0 (NumPy version)

Mode:

[64] (our version)

[64] (SciPy version)

Variance:

87.74 (our version)

87.74 (NumPy version)

Standard Deviation:

9.37 (our version)

9.37 (NumPy version)

## Summary

---

In this lab, we performed a basic, yet detailed, statistical analysis around measuring the tendencies of center and spread for a given dataset. We looked at building a number of functions to calculate different measures and also used some statistical visualizations to strengthen our intuitions around the dataset. We shall see how we can simplify this process as we study `numpy` and `pandas` libraries to ease out the programming load while calculating basic statistics.

## Releases

No releases published

## Packages

No packages published

## Contributors 13



[+ 2 contributors](#)

## Languages

● Jupyter Notebook 97.5%    ● Python 2.5%