statistics — Statistical Calculations

Purpose: Implementations of common statistical calculations.

The `statistics` module implements many common statistical formulas for efficient calculations using Python's various numerical types (`int`, `float`, `Decimal`, and `Fraction`).

Averages

There are three forms of averages supported, the mean, the median, and the mode. Calculate the arithmetic mean with `mean()`.

```
# statistics_mean.py

from statistics import *

data = [1, 2, 2, 5, 10, 12]

print('{:0.2f}'.format(mean(data)))
```

The return value for integers and floats is always a `float`. For `Decimal` and `Fraction` input data, the result is of the same type as the inputs.

```
$ python3 statistics_mean.py

5.33
```

Calculate the most common data point in a data set using `mode()`.

```
# statistics_mode.py

from statistics import *

data = [1, 2, 2, 5, 10, 12]

print(mode(data))
```

The return value is always a member of the input data set. Because `mode()` treats the input as a set of discrete values, and counts the recurrences, the inputs do not actually need to be numerical values.

```
$ python3 statistics_mode.py

2
```

There are four variations for calculating the median, or middle, value. The first three are straightforward versions of the usual algorithm, with different solutions for handling data sets with an even number of elements.

```
# statistics_median.py

from statistics import *

data = [1, 2, 2, 5, 10, 12]

print('median     : {:0.2f}'.format(median(data)))
print('low        : {:0.2f}'.format(median_low(data)))
print('high       : {:0.2f}'.format(median_high(data)))
```

`median()` finds the center value, and if the data set has an even number of values it averages the two middle items. `median_low()` always returns a value from the input data set, using the lower of the two middle items for data sets with an even number of items. `median_high()` similarly returns the higher of the two middle items.

```
$ python3 statistics_median.py

median     : 3.50
low        : 2.00
high       : 5.00
```

The fourth version of the median calculation, `median_grouped()`, treats the inputs as continuous data and calculates the 50% percentile median by first finding the median range using the provided interval width and then interpolating within that range using the position of the actual value(s) from the data set that fall in that range.

```
# statistics_median_grouped.py

from statistics import *

data = [10, 20, 30, 40]

print('1: {:0.2f}'.format(median_grouped(data, interval=1)))
print('2: {:0.2f}'.format(median_grouped(data, interval=2)))
print('3: {:0.2f}'.format(median_grouped(data, interval=3)))
```

As the interval width increases, the median computed for the same data set changes.

```
$ python3 statistics_median_grouped.py

1: 29.50
2: 29.00
3: 28.50
```

Variance

Statistics uses two values to express how disperse a set of values is relative to the mean. The *variance* is the average of the square of the difference of each value and the mean, and the *standard deviation* is the square root of the variance (which is useful because taking the square root allows the standard deviation to be expressed in the same units as the input data). Large values for variance or standard deviation indicate that a set of data is disperse, while small values indicate that the data is clustered closer to the mean.

```
# statistics_variance.py

from statistics import *
import subprocess


def get_line_lengths():
    cmd = 'wc -l ../[a-z]*/*.py'
    out = subprocess.check_output(
        cmd, shell=True).decode('utf-8')
    for line in out.splitlines():
        parts = line.split()
        if parts[1].strip().lower() == 'total':
            break
        nlines = int(parts[0].strip())
        if not nlines:
            continue  # skip empty files
        yield (nlines, parts[1].strip())


data = list(get_line_lengths())

lengths = [d[0] for d in data]
sample = lengths[::2]

print('Basic statistics:')
print('  count     : {:3d}'.format(len(lengths)))
print('  min       : {:6.2f}'.format(min(lengths)))
print('  max       : {:6.2f}'.format(max(lengths)))
print('  mean      : {:6.2f}'.format(mean(lengths)))

print('\nPopulation variance:')
```

```
print('  pstdev    : {:6.2f}'.format(pstdev(lengths)))
print('  pvariance : {:6.2f}'.format(pvariance(lengths)))

print('\nEstimated variance for sample:')
print('  count     : {:3d}'.format(len(sample)))
print('  stdev     : {:6.2f}'.format(stdev(sample)))
print('  variance  : {:6.2f}'.format(variance(sample)))
```

Python includes two sets of functions for computing variance and standard deviation, depending on whether the data set represents the entire population or a sample of the population. This example uses `wc` to count the number of lines in the input files for all of the example programs and then uses `pvariance()` and `pstdev()` to compute the variance and standard deviation for the entire population before using `variance()` and `stddev()` to compute the sample variance and standard deviation for a subset created by using the length of every second file found.

```
$ python3 statistics_variance.py

Basic statistics:
  count     : 1282
  min       :   4.00
  max       : 228.00
  mean      :  27.79

Population variance:
  pstdev    :  17.86
  pvariance : 319.04

Estimated variance for sample:
  count     : 641
  stdev     :  16.94
  variance  : 286.99
```

See also

- Standard library documentation for statistics
- mathtips.com: Median for Discrete and Continuous Frequency Type Data (grouped data) – Discussion of median for continuous data
- **PEP 450** – Adding A Statistics Module To The Standard Library

math — Mathematical Functions

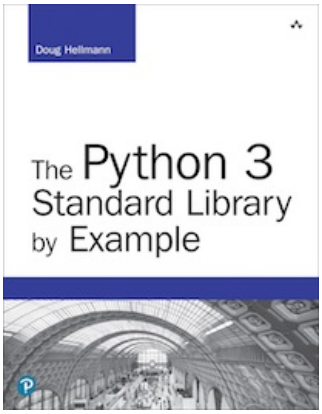
The File System

Quick Links

This page was last updated 2018-12-09.

Navigation

math — Mathematical Functions
The File System



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example