TECH2 mandatory assignment

Deadline: October 6, 2024 at 23:59

Standard deviation of a sequence of numbers

You are having a discussion with your boss and a colleague regarding the fastest method to calculate the standard deviation of a sequence of numbers.

The standard deviation σ characterizes the dispersion of a sequence of data $(x_1, x_2, \ldots, x_N)$ around its mean $\overline{x}$. A high σ indicates that many of the values in the sequence are far away from the mean, whereas a low value indicates that most values are close to the mean.

The standard deviation is computed as the square root of the variance σ^2, defined as

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \Big(x_i - \overline{x} \Big)^2$$

where N is the number of elements, and the mean $\overline{x}$ is defined as

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

The above formula for the variance can be rewritten as

$$\sigma^2 = \left(\frac{1}{N} \sum_{i=1}^{N} x_i^2 \right) - \overline{x}^2$$

This suggests the following algorithm to compute the standard deviation:

1. Compute the mean $\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$
2. Compute the mean of squares $S = \frac{1}{N} \sum_{i=1}^{N} x_i^2$
3. Compute the variance $\sigma^2 = S - \overline{x}^2$
4. Compute the standard deviation $\sigma = \sqrt{\sigma^2}$

There are several ways to compute the standard deviation of a sequence of values in Python:

1. Your colleague thinks that the fastest solution is to compute the mean and sum of the squares in the algorithm above by using solely `for` loops, and that no functions are required.
2. Your boss believes that `for` loops are slow and should therefore be avoided. He instead suggests using Python's built-in functions such as `sum()` and `len()` to reduce the number of loops.
3. You, however, know that NumPy has a function called `std()` that can be used to calculate the standard deviation of a sequence of numbers. You believe that it is generally better to use pre-existing Python functions instead of generating your own functions.

Your task is to compute the standard deviation of a sequence of numbers using all three approaches.

Part A

Write a Python script that uses all three approaches above to compute the standard deviation of a sequence of values.

For this purpose, the file `part_A.py` contains the headers of the following two functions:

[1]:
```
def std_loops(x):
    """
    Compute standard deviation of x using loops.

    Parameters
    ----------
    x: Sequence of numbers

    Returns
    -------
    sd : float
        Standard deviation of the list of numbers.
    """

def std_builtin(x):
    """
    Compute standard deviation of x using the built-in functions sum()
    and len().

    Parameters
    ----------
    x: Sequence of numbers

    Returns
    -------
    sd : float
        Standard deviation of the list of numbers.
    """
```

Provide an implementation for each function, the first using loops and the second using `sum()` and `len()`.

Write code to demonstrate that these two functions and `std()` from NumPy calculate the same standard deviations for the following list of numbers:

```
num_lst = [1, 2, 3, 4, 5]
```

Hint: You can use the built-in `sqrt()` function from the math module to compute the square root in step (4) of the algorithm outlined above:

```
from math import sqrt
```

Solution.

[2]:
```
from math import sqrt
import numpy as np

def std_loops(x):
    """
    Compute standard deviation of x using loops.

    Parameters
    ----------
    x: Sequence of numbers

    Returns
```

```
    -------
    sd : float
        Standard deviation of the list of numbers.
    """

    # Initialize counter variable
    N = 0

    # Compute mean
    mean = 0.0
    for xi in x:
        mean += xi
        N += 1
    mean /= N

    # Compute the mean of squares
    S = 0.0
    for xi in x:
        S += xi**2.0
    S /= N

    # Compute variance
    var = S - mean**2.0

    # Compute standard deviation
    sd = sqrt(var)

    return sd

def std_builtin(x):
    """
    Compute standard deviation of x using the built-in functions sum()
    and len().

    Parameters
    ----------
    x: Sequence of numbers

    Returns
    -------
    sd : float
        Standard deviation of the list of numbers.
    """

    N = len(x)
    # Compute mean
    mean = sum(x) / N

    # Compute the mean of squares
    S = sum(xi**2.0 for xi in x) / N

    # Compute variance
    var = S - mean**2.0

    # Compute standard deviation
    sd = sqrt(var)

    return sd


if __name__ == '__main__':

    # List of 5 integers
```

```python
    num_lst = [1, 2, 3, 4, 5]

    # Calculate standard deviation
    sd1 = std_loops(num_lst)
    sd2 = std_builtin(num_lst)
    sd3 = np.std(num_lst)

    # Print results using 8 decimal digits
    print(f'Std. dev. for the sequence: {num_lst}')
    print(f'  1. Loops:     {sd1:.8f}')
    print(f'  2. Built-ins: {sd2:.8f}')
    print(f'  3. NumPy:     {sd3:.8f}')
```

```
Std. dev. for the sequence: [1, 2, 3, 4, 5]
  1. Loops:     1.41421356
  2. Built-ins: 1.41421356
  3. NumPy:     1.41421356
```

Part B

Your boss is not convinced that `std()` from NumPy can compute the standard deviation faster than the two other approaches. He therefore gives you a file containing randomly generated data, and he asks you to compare the run time of all three approaches.

The file `data.csv` contains the following three columns:

- Column 1: sequence of 100 numbers between 0 and 1
- Column 2: sequence of 1,000 numbers between 0 and 1
- Column 3: sequence of 10,000 numbers between 0 and 1

This is a comma-separated text file where the values in each row are separated by a comma. He knows that you are not familiar with this file format, so he suggests that you look at this link to see how you can import the data in the file by using Python's built-in `open()` function.

Use the empty notebook `part_B.ipynb` in this repository to complete the following tasks:

1. Import the file and store each column of values in a list.
2. Compute the standard deviation of all three sequences using the three different approaches
3. Record the run-time of each approach for each sequence. Recall that you can use the magic function `%timeit` to time the execution of a function in a Jupyter notebook.

Summarize your conclusion for your boss. Which approach computes the standard deviation the fastest? Is one approach always faster or does it depend on the length of the sequence? Try to explain your findings.

Solution.

Read in the data file

A function to read in all three columns from `data.csv` can be implemented as follows:

```python
[3]: def read_data(path):
         """
         Read in data file from given path.

         Parameters
         ----------
         path : str
```

```
        Path or file name of data file to read

    Returns
    -------
    x1 : list
        Non-missing values in column 1
    x2 : list
        Non-missing values in column 2
    x3 : list
        Non-missing values in column 3
    """

    file = open('data.csv', 'rt')

    # Initialize lists to store numbers
    x1 = []
    x2 = []
    x3 = []

    # Loop over all lines in file
    for line in file:

        # Convert line to list
        line_lst = line.split(',')

        # Append numbers to lists if not missing
        if line_lst[0] != '': x1.append(float(line_lst[0]))
        if line_lst[1] != '': x2.append(float(line_lst[1]))
        if line_lst[2] != '': x3.append(float(line_lst[2]))

    file.close()

    return x1, x2, x3
```

Alternatively, we could use NumPy's `genfromtxt()` to read in the data:

[4]:
```
import numpy as np

def read_data_numpy(path):
    """
    Read in data file from given path using genfromtxt().

    Parameters
    ----------
    path : str
        Path or file name of data file to read

    Returns
    -------
    x1 : array
        Non-missing values in column 1
    x2 : array
        Non-missing values in column 2
    x3 : array
        Non-missing values in column 3
    """

    data = np.genfromtxt(path, delimiter=',', filling_values=np.nan)

    # Extract non-missing values in each column
    x1 = data[~np.isnan(data[:,0]), 0]
    x2 = data[~np.isnan(data[:,1]), 1]
    x3 = data[~np.isnan(data[:,2]), 2]
```

```
    return x1, x2, x3
```

Compute standard deviations

[5]:
```
def display_sd(data):

    # Compute standard deviation using all three implementations
    sd1 = std_loops(data)
    sd2 = std_builtin(data)
    sd3 = np.std(data)

    print(f'  Loops:     {sd1:.8f}')
    print(f'  Built-ins: {sd2:.8f}')
    print(f'  NumPy:     {sd3:.8f}')
```

[6]:
```
x1, x2, x3 = read_data('data.csv')

for i, x in enumerate([x1, x2, x3]):
    print(f'Standard deviation of column {i+1} ({len(x)} numbers):')
    display_sd(x)
    print()
```

```
Standard deviation of column 1 (100 numbers):
  Loops:     0.28237211
  Built-ins: 0.28237211
  NumPy:     0.28237211

Standard deviation of column 2 (1000 numbers):
  Loops:     0.28467443
  Built-ins: 0.28467443
  NumPy:     0.28467443

Standard deviation of column 3 (10000 numbers):
  Loops:     0.28540453
  Built-ins: 0.28540453
  NumPy:     0.28540453
```

Compare run time

100 numbers

[7]:
```
%timeit std_loops(x1)
```

```
7.64 µs ± 28.6 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

[8]:
```
%timeit std_builtin(x1)
```

```
6.65 µs ± 13 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

[9]:
```
%timeit np.std(x1)
```

```
16.5 µs ± 61.4 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

1,000 numbers

[10]:
```
%timeit std_loops(x2)
```

```
79.1 µs ± 483 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[11]:
```
%timeit std_builtin(x2)
```

```
60.1 µs ± 568 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[12]:
```
%timeit np.std(x2)
```

```
50 µs ± 337 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

10,000 numbers

[13]:
```
%timeit std_loops(x3)
```

```
802 µs ± 1.67 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[14]:
```
%timeit std_builtin(x3)
```

```
615 µs ± 2.02 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[15]:
```
%timeit np.std(x3)
```

```
372 µs ± 155 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Conclusion

Note: Using `std` from `numpy` is even faster if the sequence of numbers are arrays instead of lists.

[16]:
```
x_array = np.array(x3)
```

[17]:
```
%timeit std_loops(x_array)
```

```
2.27 ms ± 23.2 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[18]:
```
%timeit std_loops(x_array)
```

```
2.27 ms ± 17.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[19]:
```
%timeit np.std(x_array)
```

```
21.1 µs ± 36.5 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Assessment

The requirements to pass the assignment are as follows:

- You have submitted within the deadline.
- Your submission is in a GitHub repository. All commits in this repository must be prior to the deadline. The repository must be publicly accessible (set visibility to public in the repository settings):

- The repository contains a Python script that attempts to calculate the standard deviation using all three approaches using the list of numbers (part A).
- You must complete a peer-review of the assignments of two other students on Canvas (deadline: Friday, October 11, 2024 at 14:00).

In addition, the following are recommended, but not required to pass the assignment:

- You create and use the Anaconda environment defined in `environment.yml`.
- Your Python script uses functions to solve part A of the assignment.
- Your code is well documented.
- Your repository contains a Jupyter notebook that solves part B of the assignment.
- You use git to manage your repository.