Nathan Roe
OOP with C++
EN.605.604 Section 81
2/6/2022

Module 2 Assignment: Statistical Class

Design

Project requirements state that the code must create a class to track statistics for a distribution of doubles, following the given interface:

```
class Statistic
{
public:
      //Constructor: initialize private data
      Statistic();

      // add an item to the statistics
      void add(double x);

      // get average
      double average() const;

      // get Standard deviation: using the hint
      double STD() const;

private:
      // private member data
};
```

The Statistic object must be able to receive doubles one at a time, and be able to retrieve the sample mean and sample distribution after values are added.

- The sample mean for a distribution can be calculated with:

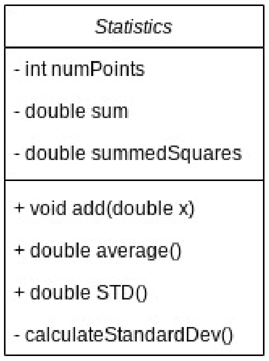
$$\mu_{sample} = \frac{\Sigma x_i}{n}$$

- The sample standard deviation can be calculated with

$$\sigma_{sample} = \frac{\Sigma x_i^2 - (\frac{\Sigma x_i \cdot \Sigma X_i}{n})}{n-1}$$

This means in order to track the mean and standard deviation, I will have to keep a tally of:

- $n - numPoints$
- $\Sigma x_i - sum$
- $\Sigma x_i^2 - summedSquares$



```cpp
// Accepts command line args for double values
// (ints promoted to doubles) and calculates the
// sample mean and sample standard deviation for
// the given distribution

#include "Statistic/statistic.hpp"
#include <iostream>
#include <string>

using namespace std;

// Function main begins statistic evaluation loop
int main(int argc, char *argv[])
{
    Statistic *statCalc = new Statistic();

    // Iterate over command line args
    for (int i = 1; i < argc; i++)
    {
        // Add numbers to distribution
        try
        {
            double x = stod(argv[i]);
            statCalc->add(x);
        }
        // Catch inputs that can't be converted to numbers
        catch (const std::invalid_argument &e)
        {
            cout << "Could not interpret <" << argv[i] << "> as a double." << endl;
            cout << "Please update inputs to only include doubles." << endl;
            return 1;
        }
    }

    // Print out statistical info
    cout << "Mean: " << statCalc->average() << endl;
    cout << "Std Deviation: " << statCalc->STD() << endl;
    return 0;
} // End function main
```

```cpp
#include <list>
using namespace std;

// Creates Statistic object that provides
// sample mean and sample standard deviations for
// provided distributions. Values are added to
// the distribution as floats one at a time.
class Statistic
{
public:
    // Uses Default empty constructor

    // Add given double to the distribution
    void add(double x);

    // Retrieve the average (sample mean) for the
    // current distribution
    double average() const;

    // Retrieve the sample standard deviation for
    // the current distribution
    double STD() const;

private:
    // Number of points added to the distribution
    int numPoints = 0;
    // Current total of all values in distribution
    double sum = 0;
    // Current sum of the squares of all values in distribution
    double summedSquares = 0;

    // Calculates and returns the current sample varience
    // for the distribution
    double calcSampleVarience() const;
};
```

```cpp
#include "statistic.hpp"
#include <iostream>
#include <cmath>
using namespace std;

// Function to calculate and retrieve the average
// (sample mean) of the current distribution
double Statistic::average() const
{
    // Calculate average of values added
    if (this->numPoints > 0)
    {
        // Integer numPoints promoted to double
        return this->sum / this->numPoints;
    }
    // Avoid divide by 0 error
    else
    {
        return 0;
    }
} // End function average

// Function to calculate and retrieve the sample standard deviation of
// the current distribution
double Statistic::STD() const
{
    return sqrt(calcSampleVarience());
} // End function STD

// Function to add new value to the distribution.
void Statistic::add(double x)
{
    // Update data point count
    ++this->numPoints;

    // Update sums required for tracking the
    // running mean/standard dev
    this->sum += x;
    this->summedSquares += x * x;
} // End function add

// Function to calculate the sample varience of the current distribution
double Statistic::calcSampleVarience() const
{
    // Var = [Σx^2 * (Σx*Σx/n)] / (n-1)
    if (this->numPoints > 1)
    {
        // Integer numPoints is promoted to double
        double squaredSum = this->sum * this->sum / this->numPoints;
        return (this->summedSquares - squaredSum) / (this->numPoints - 1);
    }
    // Avoid divide by 0 error
    else
    {
        return 0;
    }
} // End function calcSampleVarience
```

```python
""" Runs tests comparing the Python 'statistics' to C++ class

This will read lines from file 'tests.csv' stored in
the same folder as this file, and calculate the sample
mean and sample standard deviation using the
python 'statistics' module, and the TestStatistics C++
class stored in the ../ folder.
"""

import csv
import os
import statistics


def run_stats(values):
    """Calculate the mean and samle std dev for given values

    Args:
        values (list[int or float]): List of values to run stats on
    """
    try:
        # Attempt conversion to floats
        values = [float(v) for v in values]
        # Calculate statistics
        print('Mean:', statistics.mean(values))
        print('Std Deviation:', statistics.stdev(values))
    # Catch float conversion errors
    except ValueError as e:
        print(e)


def main():
    """Main function. Reads distributions from tests.csv
    and calculates stats in two different ways.
    """
    # Setup working directory
    start_path = os.getcwd()
    os.chdir(os.path.dirname(os.path.abspath(__file__)))

    # Read 'tests.csv' in current folder
    with open(r'tests.csv') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            row = [value.strip() for value in row]
            if row:
                # Print distribution to console
                print('CSV Values:', row)

                # Run python stats and print results
                print('\nPython Results:')
                run_stats(row)

                # Run C++ stats and print results
                print('\nC++ Results:')
                test_string = '../StatisticTest ' + ' '.join(row)
                print(test_string)
                os.system(test_string)

                print('\n')

    # Return working dir to starting location
    os.chdir(start_path)


if __name__ == "__main__":
    main()
```

```
// C++ style comments were removed for test running

// Test lists of doubles
1.0, 2.0, 3.0, 4.0, 5.0
10.1, 12.0, 23.1, 23.0, 16.1, 21.0, 16.1

// Test lists of integers
1, 2, 3, 4, 5
10, 12, 23, 23, 16, 21, 16

// Test mixed values
56, 18, 22, 0.5

// Test invalid characters
1, 2, f
a, b, c
$, ^, 4@

// Test longer list of doubles
// Newlines added to help with printing
83.2831,65.5039,14.6599,73.6619,76.3575,84.1906,79.1789,90.2266,50.2694,64.0549,26.8703,
28.4595,11.3642,23.7067,43.0236,30.9179,76.0219,76.9771,6.4089,11.4425,74.7286,15.9673,
26.8426,30.3849,5.306,12.0642,81.2853,35.3401,70.3817,6.4583,41.6357,17.7007,8.6751,
6.6893,89.0175,14.228,84.4672,67.0832,22.9599,10.5394,61.427,40.5194,81.9153,94.8238,
46.3909,10.3985,78.6326,84.1039,84.6922,24.8002,46.4194,31.6437,76.2617,77.3686,61.7926,
24.7442,33.3867,59.8989,47.6458,86.7503,85.2989,60.5278,15.3923,17.3211,46.4935,57.0191,
51.2853,47.4769,19.5403,18.9644,20.5448,3.8412,27.3385,86.3373,75.0321,13.1798,77.21,
20.9916,30.6003,21.5349,22.6894,1.5226,80.0837,63.0935,71.3989,43.3208,69.7409,6.3577,
23.613,3.6974,91.8189,56.3994,78.6034,69.981,86.1948,89.2944,98.4073,51.8113,92.3735,
65.1536

// Calculate stats for sequence from Mod 1 discussion
1, 1, 2, 5, 15, 52, 203, 877, 4140
```

```
CSV Values: ['1.0', '2.0', '3.0', '4.0', '5.0']

Python Results:
Mean: 3.0
Std Deviation: 1.5811388300841898

C++ Results:
../StatisticTest 1.0 2.0 3.0 4.0 5.0
Mean: 3
Std Deviation: 1.58114


CSV Values: ['10.1', '12.0', '23.1', '23.0', '16.1', '21.0', '16.1']

Python Results:
Mean: 17.342857142857142
Std Deviation: 5.206040447677788

C++ Results:
../StatisticTest 10.1 12.0 23.1 23.0 16.1 21.0 16.1
Mean: 17.3429
Std Deviation: 5.20604


CSV Values: ['1', '2', '3', '4', '5']

Python Results:
Mean: 3.0
Std Deviation: 1.5811388300841898

C++ Results:
../StatisticTest 1 2 3 4 5
Mean: 3
Std Deviation: 1.58114


CSV Values: ['10', '12', '23', '23', '16', '21', '16']

Python Results:
Mean: 17.285714285714285
Std Deviation: 5.219012860502955

C++ Results:
../StatisticTest 10 12 23 23 16 21 16
Mean: 17.2857
Std Deviation: 5.21901


CSV Values: ['56', '18', '22', '0.5']

Python Results:
Mean: 24.125
Std Deviation: 23.21053999084611

C++ Results:
../StatisticTest 56 18 22 0.5
Mean: 24.125
```

```
Std Deviation: 23.2105


CSV Values: ['1', '2', 'f']

Python Results:
could not convert string to float: 'f'

C++ Results:
../StatisticTest 1 2 f
Could not interpret <f> as a double.
Please update inputs to only include doubles.


CSV Values: ['a', 'b', 'c']

Python Results:
could not convert string to float: 'a'

C++ Results:
../StatisticTest a b c
Could not interpret <a> as a double.
Please update inputs to only include doubles.


CSV Values: ['$', '^', '4@']

Python Results:
could not convert string to float: '$'

C++ Results:
../StatisticTest $ ^ 4@
Could not interpret <$> as a double.
Please update inputs to only include doubles.


CSV Values: ['83.2831', '65.5039', '14.6599', '73.6619', '76.3575', '84.1906',
'79.1789', '90.2266', '50.2694', '64.0549', '26.8703', '28.4595', '11.3642',
'23.7067', '43.0236', '30.9179', '76.0219', '76.9771', '6.4089', '11.4425', '74.7286',
'15.9673', '26.8426', '30.3849', '5.306', '12.0642', '81.2853', '35.3401', '70.3817',
'6.4583', '41.6357', '17.7007', '8.6751', '6.6893', '89.0175', '14.228', '84.4672',
'67.0832', '22.9599', '10.5394', '61.427', '40.5194', '81.9153', '94.8238', '46.3909',
'10.3985', '78.6326', '84.1039', '84.6922', '24.8002', '46.4194', '31.6437',
'76.2617', '77.3686', '61.7926', '24.7442', '33.3867', '59.8989', '47.6458',
'86.7503', '85.2989', '60.5278', '15.3923', '17.3211', '46.4935', '57.0191',
'51.2853', '47.4769', '19.5403', '18.9644', '20.5448', '3.8412', '27.3385', '86.3373',
'75.0321', '13.1798', '77.21', '20.9916', '30.6003', '21.5349', '22.6894', '1.5226',
'80.0837', '63.0935', '71.3989', '43.3208', '69.7409', '6.3577', '23.613', '3.6974',
'91.8189', '56.3994', '78.6034', '69.981', '86.1948', '89.2944', '98.4073', '51.8113',
'92.3735', '65.1536']

Python Results:
Mean: 48.534356
Std Deviation: 29.286976359940972

C++ Results:
../StatisticTest 83.2831 65.5039 14.6599 73.6619 76.3575 84.1906 79.1789 90.2266
50.2694 64.0549 26.8703 28.4595 11.3642 23.7067 43.0236 30.9179 76.0219 76.9771 6.4089
11.4425 74.7286 15.9673 26.8426 30.3849 5.306 12.0642 81.2853 35.3401 70.3817 6.4583
```

```
41.6357 17.7007 8.6751 6.6893 89.0175 14.228 84.4672 67.0832 22.9599 10.5394 61.427
40.5194 81.9153 94.8238 46.3909 10.3985 78.6326 84.1039 84.6922 24.8002 46.4194
31.6437 76.2617 77.3686 61.7926 24.7442 33.3867 59.8989 47.6458 86.7503 85.2989
60.5278 15.3923 17.3211 46.4935 57.0191 51.2853 47.4769 19.5403 18.9644 20.5448 3.8412
27.3385 86.3373 75.0321 13.1798 77.21 20.9916 30.6003 21.5349 22.6894 1.5226 80.0837
63.0935 71.3989 43.3208 69.7409 6.3577 23.613 3.6974 91.8189 56.3994 78.6034 69.981
86.1948 89.2944 98.4073 51.8113 92.3735 65.1536
Mean: 48.5344
Std Deviation: 29.287

CSV Values: ['1', '1', '2', '5', '15', '52', '203', '877', '4140']

Python Results:
Mean: 588.4444444444445
Std Deviation: 1361.823053035077

C++ Results:
../StatisticTest 1 1 2 5 15 52 203 877 4140
Mean: 588.444
Std Deviation: 1361.82
```