gaussian_code_exercise

November 30, 2019

1 Gaussian Code Exercise

Read through the code below and fill out the TODOs. You'll find a cell at the end of the Jupyter notebook containing unit tests. After you've run the code cell with the Gaussian class, you can run the final cell to check that your code functions as expected.

This exercise includes a file called 'numbers.txt', which you can see if you click on the 'Jupyter' icon at the top of the workspace and then go into the folder titled 3.OOP_code_gaussian_class. The 'numbers.txt' file is read in by the read_data_file() method. There is also a solution in the 3.OOP_code_gaussian_class folder in a file called answer.py.

```
In [21]: import math
         import matplotlib.pyplot as plt
         import numpy as np
         import statistics as st
         from scipy.stats import norm

         class Gaussian():
             """ Gaussian distribution class for calculating and
             visualizing a Gaussian distribution.

             Attributes:
                 mean (float) representing the mean value of the distribution
                 stdev (float) representing the standard deviation of the distribution
                 data_list (list of floats) a list of floats extracted from the data file

             """
             def __init__(self, mu = 0, sigma = 1):

                 self.mean = mu
                 self.stdev = sigma
                 self.data = []



             def calculate_mean(self):

                 """Method to calculate the mean of the data set.
```

```
        Args:
            None

        Returns:
            float: mean of the data set

        """

        #TODO: Calculate the mean of the data set. Remember that the data set is stored
        # Change the value of the mean attribute to be the mean of the data set
        # Return the mean of the data set
        self.mean = np.mean(np.array(self.data))
        return self.mean



    def calculate_stdev(self, sample=True):

        """Method to calculate the standard deviation of the data set.

        Args:
            sample (bool): whether the data represents a sample or population

        Returns:
            float: standard deviation of the data set

        """

        # TODO:
        #   Calculate the standard deviation of the data set
        #
        #   The sample variable determines if the data set contains a sample or a popul
        #   If sample = True, this means the data is a sample.
        #   Keep the value of sample in mind for calculating the standard deviation
        #
        #   Make sure to update self.stdev and return the standard deviation as well

        # numpy uses population standard deviation by default, which is similar to pstd
        # If you want to use it to calculate sample standard deviation, use an addition
        # called ddof and set it to 1.

        if sample:
            self.stdev = st.stdev(self.data)
        else:
            self.stdev = st.pstdev(self.data)
        return self.stdev
```

```
def read_data_file(self, file_name, sample=True):

    """Method to read in data from a txt file. The txt file should have
    one number (float) per line. The numbers are stored in the data attribute.
    After reading in the file, the mean and standard deviation are calculated

    Args:
        file_name (string): name of a file to read from

    Returns:
        None

    """

    # This code opens a data file and appends the data to a list called data_list
    with open(file_name) as file:
        data_list = []
        line = file.readline()
        while line:
            data_list.append(int(line))
            line = file.readline()
    file.close()

    # TODO:
    #   Update the self.data attribute with the data_list
    #   Update self.mean with the mean of the data_list.
    #       You can use the calculate_mean() method with self.calculate_mean()
    #   Update self.stdev with the standard deviation of the data_list. Use the
    #       calcaulte_stdev() method.
    self.data = data_list
    self.mean = self.calculate_mean()
    self.stdev = self.calculate_stdev(sample)


def plot_histogram(self):
    """Method to output a histogram of the instance variable data using
    matplotlib pyplot library.

    Args:
        None

    Returns:
        None
    """

    # TODO: Plot a histogram of the data_list using the matplotlib package.
    #       Be sure to label the x and y axes and also give the chart a title
```

```
        plt.xlabel("data")
        plt.ylabel("density")
        plt.title(r'Histogram')
        plt.plot(self.data)
        plt.show()


    def pdf(self, x):
        """Probability density function calculator for the gaussian distribution.

        Args:
            x (float): point for calculating the probability density function


        Returns:
            float: probability density function output
        """

        # TODO: Calculate the probability density function of the Gaussian distribution
        #       at the value x. You'll need to use self.stdev and self.mean to do the c
        return norm(self.mean,self.stdev).pdf(x)

    def plot_histogram_pdf(self, n_spaces = 50):

        """Method to plot the normalized histogram of the data and a plot of the
        probability density function along the same range

        Args:
            n_spaces (int): number of data points

        Returns:
            list: x values for the pdf plot
            list: y values for the pdf plot

        """

        #TODO: Nothing to do for this method. Try it out and see how it works.

        mu = self.mean
        sigma = self.stdev

        min_range = min(self.data)
        max_range = max(self.data)

         # calculates the interval between x values
        interval = 1.0 * (max_range - min_range) / n_spaces

        x = []
```

```
        y = []

        # calculate the x values to visualize
        for i in range(n_spaces):
            tmp = min_range + interval*i
            x.append(tmp)
            y.append(self.pdf(tmp))

        # make the plots
        fig, axes = plt.subplots(2,sharex=True)
        fig.subplots_adjust(hspace=.5)
        axes[0].hist(self.data, density=True)
        axes[0].set_title('Normed Histogram of Data')
        axes[0].set_ylabel('Density')

        axes[1].plot(x, y)
        axes[1].set_title('Normal Distribution for \n Sample Mean and Sample Standard D
        axes[0].set_ylabel('Density')
        plt.show()

        return x, y
```

In [22]:
```
# Unit tests to check your solution

import unittest

class TestGaussianClass(unittest.TestCase):
    def setUp(self):
        self.gaussian = Gaussian(25, 2)

    def test_initialization(self):
        self.assertEqual(self.gaussian.mean, 25, 'incorrect mean')
        self.assertEqual(self.gaussian.stdev, 2, 'incorrect standard deviation')

    def test_pdf(self):
        self.assertEqual(round(self.gaussian.pdf(25), 5), 0.19947,\
         'pdf function does not give expected result')

    def test_meancalculation(self):
        self.gaussian.read_data_file('numbers.txt', True)
        self.assertEqual(self.gaussian.calculate_mean(),\
         sum(self.gaussian.data) / float(len(self.gaussian.data)), 'calculated mean not

    def test_stdevcalculation(self):
        self.gaussian.read_data_file('numbers.txt', True)
        self.assertEqual(round(self.gaussian.stdev, 2), 92.87, 'sample standard deviati
        self.gaussian.read_data_file('numbers.txt', False)
        self.assertEqual(round(self.gaussian.stdev, 2), 88.55, 'population standard dev
```

```
        tests = TestGaussianClass()

        tests_loaded = unittest.TestLoader().loadTestsFromModule(tests)

        unittest.TextTestRunner().run(tests_loaded)
```

```
...
----------------------------------------------------------------------
Ran 4 tests in 0.017s

OK
```

Out[22]: <unittest.runner.TextTestResult run=4 errors=0 failures=0>

In [ ]: