Homework 3: Histograms and Distributions

Due: Friday, 09.19.2025 at 11:59 PM ET

This homework asks you to write and use the functions we discussed in the lecture on histograms to find the "optimal" number of bins for a data set. We then ask you to explore some data to identify its distribution.

Goals

In this homework, you will:

1. Learn how to install new Python modules
2. Build up a complex analysis code by building smaller functions first
3. Perform some basic data exploration

Background

Python modules

Python has a lot of built-in functionality right out of the box: basic data structures like lists, sets, and dictionaries, functions that help use those data structures (like \`len\`), etc. There are a vast number of Python modules that provide additional functionality too. This functionality is not built in -- not everyone needs it -- but Python comes with tools to make it easy to use those modules.

Installing a module

To install a module, you can use Python's built-in package manager, \`pip\`. We will provide instructions for installing modules from the command line. These instructions will work with Python3 installations on Scholar, ECE Grid, most Linux distributions, and Mac OS. There are a number of different ways to get Python3 on Windows, so you will have to look at the documentation for your version to determine how to install a new module.

Modules can be installed globally (so everyone on a machine has access to them) or locally (so only you have access to them). Installing modules globally requires root access to the machine (or other specially set permissions), so we will provide instructions for installing modules locally.

To install a module named `scipy`, you can use the following command:

```
python3 -mpip install --user scipy
```

> Note for Windows Users: if `python3 -mpip install` doesn't work, try to use `python -mpip install`.

This command can let you install the latest version of the module. You can also install an exact version of the module by adding the version number after the module's name followed by the sign ==:

```
python3 -mpip install --user scipy==2.0.8 # install the scipy module with version number 2.0.8
```

If `scipy` is already installed on your system but you want to upgrade to a new version of it, you can use the command:

```
python3 -mpip install --user --upgrade scipy
```

To check if a module is already installed on your system, you can use the command `pip list`. This will list all the installed packages and their versions.

There are also other choices of package managers, for example, if you are using `conda` Getting Started with Conda to manage your Python environment, you can do the same thing as `pip` does with the following commands:

```
conda install scipy # Install the latest version of scipy module
```

```
conda install scipy=2.0.8 # Install the scipy version 2.0.8
```

```
conda update scipy # Update the module scipy
```

> Note: In order to specify a specific version number `pip` uses == while `conda` uses =.

To complete this homework, you will need to install the following modules with the specified versions or higher:

1. `numpy==1.24.4`: this is a module that provides array and matrix classes, and many mathematical operations on those classes. It is the foundation of many of the modules that are used in data science.
2. `scipy==1.10.1`: this module provides many other useful functions for data analysis, including functions for dealing with probability distributions.
3. `matplotlib==3.7.2`: a basic plotting/visualization library. We will use this library to create a histogram in problem 1.

> Note 1: You are welcome to use different versions of the above modules since a lot of them may already have been installed in your environment. There might be a slight chance that potential problems will occur if you are using modules that are of a different version, especially older versions than the ones provided above. Therefore, please make sure your version is at least as large as the version number provided above.

> Note 2: If you encounter an error message of `ModuleNotFoundError: No module named 'tkinter'` then, in your code, replace the line:
>
> ```
> import matplotlib.pyplot as plt
> ```
>
> with the following lines:
>
> ```
> import matplotlib
> matplotlib.use('agg')
> import matplotlib.pyplot as plt
> ```

Using a module

The functions in a module are in that module's 'namespace'. To make sure that the function names do not collide with functions in other modules (or Python's built-in functions), the functions need to be accessed through a prefix. To load a module, you have to tell Python (a) which module to load; and (b) what prefix to use when accessing the functions of that module. For example, the following code:

```
import numpy as np
```

Tells Python you want to use the module `numpy`, and that you want to access the functions of `numpy` using the prefix `np`. For example, the following code will read in a list of numbers stored in a text file and give you back a list with those numbers in it:

```
data = np.loadtxt('input.txt')
```

You can also use the `import` keyword to bring in functions from other files (think of these like `#include` directives in C). The following command will import `function1` from a file called `myfile.py`:

```
from myfile import function1
```

You can also import all functions defined in another file, like `helper.py` for example, using the asterisk operator:

```
from helper import *
```

Incremental Development

In this homework, we will ask you to write a fairly complex piece of code: finding the number of histogram bins that result in the lowest error for a given data set. When you need to write complex code like this, your goal should be to break the problem down into smaller pieces. Write functions that solve each of the smaller pieces, then figure out how to connect those functions together (some of them might call other functions you wrote) to solve the overall problem.

This approach makes it much easier to write complex code, both because you do not have to solve the problem all in one go, and because it makes it easier to *test* your code: you can test each of your smaller pieces individually to make sure that they work properly.

In this homework, we will walk you through one particular way you can break down the problem (and, in fact, we want you to solve the problem in this way -- we will test the individual pieces for partial credit).

Store returned values from a function

For problem 2, you would be returning the coordinates of the data points in the probability plot (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.probplot.html). The probplot return format is unusual (it returns a tuple of two tuples of which we need the first tuple that corresponds to the coordinates of the data points).

To return these coordinate values you can use the following command:

```
import scipy.stats as stats
data = np.loadtxt('input.txt')
val1,val2=stats.probplot(data, dist = 'uniform', plot=plt)
```

For this example, this will store both return values from the probplot and will store them in val1 and val2 respectively. Only val1 (per documentation) will be related to the coordinates. Here, val1 stores the coordinates X and Y which would be used to calculate the distance from the coordinates of the probplot to the identity line (X=Y).

Instructions

0) Download the homework

Download the files for HW 3 from Brightspace.

The folder should contain 12 files:

1. This README and its PDF.
2. Eight input data files : `input.txt`, which you will use in problem 1, and `sample_expon.csv`, `sample_norm.csv`, `sample_uniform.csv`, `sample_wald.csv`,`distA.csv`, `distB.csv`, and `distC.csv`, which you will use in problem 2
3. `hw3.py`, a skeleton script for Problem 1 and 2

4. `testbin.py`, to help individually test the functions you will write for Problem1

1) Problem 1: Histogram Bin Width Optimization

In this problem, you will implement histogram bin width optimization from a data set. You will use the histogram function in `matplotlib.pyplot`, accessed as `matplotlib.pyplot.hist` or `plt.hist` if you `import matplotlib.pyplot as plt`. Please read the documentation: `matplotlib.pyplot.hist`. (Note in particular that the function returns a tuple of three elements: n, `bins`, and `patches`, but you only need n, so be sure to unpack the output accordingly.)

An example of unpacking a tuple:

```
Tuple1 = ('String1', 'String2')
str1, str2 = Tuple1 # str1 == 'String1' and str2 == 'String2'
```

For problem 1, the histograms are represented as lists with each entry in the list referring to the number of samples in a bin.

We have broken the Problem 1 part into four functions for you to fill in. These functions collectively work together to optimize the creation of a histogram. The `norm_histogram` function transforms a list of counts into a list of probabilities, which enables the `compute_j` function to calculate j, which is a measure of error or inaccuracy of the input. These functions are designed as helper functions of `sweep_n`, which takes in a data set and calculates j for a range of number of bins. The resulting output can then be passed to `find_min` in order to determine the smallest j values, or errors, and their corresponding indices. You are encouraged to call these functions inside other functions, as it will significantly reduce the length of your code. **Keep the signatures (or function headers) of these functions the same as you are filling them in; we will use these to assign partial credit**. For reference, a signature/function header looks like this: `my_function(input_1, input_2)`.

1. `norm_histogram` takes a list of counts as input; creates output as a list of probabilities. To explain norm_histogram below is a sample calculation that shows how the list of probabilities is calculated from the histogram (with the data represented from the histogram shown just for your understanding)

```
Data: [0,8,3,7,9,8,6,8,5,2,1,9,8,5,5,3,4,4,8,2]
Histogram: [2   4   5   2  7] (For this example we used 5 bins) (This is
Number of samples: 20   (added up the samples: 2 + 4 + 5 + 2 +7 = 20)
Normalized list of probabilities: [0.1   0.2   0.25  0.1  0.35]
(the first bin of the original histogram which has 2 is 10% of the total
The second bin of the original histogram has 4 samples which is 20% of t
```

2. `compute_j` takes list of counts, bin width and total number of samples as input, uses `norm_histogram` function above to output the list of probabilities, then computes the value of j for a given list of probabilities and the corresponding bin width (reference slides: histogram.pdf page19)

3. `sweep_n` computes the `j` value for different *number of bins*, where *number of bins* will take values from *min_bins* to *max_bins* (we want the range *min_bins* to *max_bins* to be inclusive). Therefore, `sweep_n` should return a list of `compute_j` values. You will need to use the `compute_j` and `matplotlib.pyplot.hist` functions in your implementation. Note that `sweep_n` cares about the number of bins while `j` cares about the width of the bins -- make sure to do the conversion!.

4. `find_min` is a generic function that takes a list of numbers and returns the three smallest number in that list along with their index. More specifically, the function returns a returns a dictionary, i.e. {index_of_the_smallest_value: the_smallest_value, index_of_the_second_smallest_value: the_second_smallest_value, index_of_the_third_smallest_value: the_third_smallest_value}. For example, if the input is [14, 27, 15, 49, 23, 41, 147], then you should return {0: 14, 2: 15, 4: 23}.

   (Hint: Not necessary, but List slicing might be useful for this function.)

> Note: Any native python functions, such as `sorted` and `sort`, are allowed as part of your solution.

You can use `input.txt`, provided in the folder, as test data. To test each function individually please refer to `testbin.py`. There are instructions provided in the file to test each portion of your code.

Within `testbin.py` if:

1. `norm_histogram` runs correctly then the output will be

```
histogram:  [  5.   20.   49. 145. 201. 234. 210.   87.   37.   12.]
normalized histogram:  [0.005, 0.02, 0.049, 0.145, 0.201, 0.234, 0.21, 0
```

2. `compute_j` works then the output will be `-0.017002803765276 04`
3. `sweep_n` works then the output (shown only up to 4 points post decimal) should be
```
[-0.0107 -0.0110 -0.0174 -0.0173 -0.0170 -0.0183 -0.0183 -0.0180
-0.0185 -0.0183 -0.0184 -0.0186 -0.0184 -0.0184 -0.0187 -0.0184
-0.0184 -0.0186 -0.0184 -0.0184 -0.0186 -0.0184 -0.0185 -0.0185
-0.0184 -0.0184 -0.0185 -0.0184 -0.0183 -0.0185 -0.0185 -0.0184
-0.0184 -0.0184 -0.0184 -0.0184 -0.0184 -0.0185 -0.0184 -0.0184
-0.0183 -0.0183 -0.0183 -0.0184 -0.0183 -0.0183 -0.0182 -0.0182
-0.0185 -0.0182 -0.0181 -0.0182 -0.0181 -0.0183 -0.0183 -0.0181
-0.0182 -0.0181 -0.0181 -0.0182 -0.0182 -0.0181 -0.0181 -0.0182
-0.0181 -0.0181 -0.0183 -0.0179 -0.0181 -0.0181 -0.0181 -0.0180
-0.0180 -0.0181 -0.0180 -0.0179 -0.0179 -0.0178 -0.0179 -0.0182
-0.0178 -0.0179 -0.0180 -0.0179 -0.0179 -0.0179 -0.0179 -0.0178
-0.0178 -0.0180, -0.0176 -0.0177 -0.0178 -0.0178 -0.0177 -0.0177
-0.0176 -0.0179 -0.0176 -0.0176]
```

> PLEASE NOTE: you are not to truncate the values. We have only done so to keep the write up brief.

4. `find_min` executes then your output should be `{14: -0.01868362283870376, 17: -0.01863540572320631, 20: -0.018591577843945776}`

The output from `find_min` will be checked up to four points post decimal.

If your functions all work, and you run the test code that is included in `hw3.py` by uncommenting the problem 1 test part, you should produce the following output: `{14: -0.01868362283870376, 17: -0.01863540572320631, 20: -0.018591577843945776}`.

> The `if __name__ == '__main__'` line in `hw3.py` is a useful way to write tests for your code: this is code that will *only* run if you run this file as the main script; if this file is included from another script, this test code will not run.

Problem 2: Distributions

In this problem we will draw from your probability plot understanding to create a function that for any dataset reports what is the closest distribution fit between:

- Gaussian (`norm`)
- Exponential (`expon`)
- Uniform (`uniform`)
- Wald (`wald`)

> Note: for this problem we will assume that the best fit distribution is that in which the sum of squared distances from the coordinates of the probplot to the identity line (X=Y) is minimized.

In problem 2, you will be writing functions which assesses which distribution type best fits the input data and how well it fits. In the get_coordinates function, you will retrieve coordinates that are used in the calculate_distance function to quantify how well the input data fits with a specified distribution type. Then, in the find_dist function, you will look through the calculated distances and determine which of the distribution types fits best and its corresponding distance value.

To complete this problem, do the following steps.

1. Complete the `get_coordinates` function. This function takes in an array of data and the name of a distribution. It then calculates the QQ plot by calling the `stats.probplot`

function with the dataset and the named distribution. The `stats.probplot` function returns a bizarre data structure: a tuple of two tuples; we're concerned with the two values of the first tuple. More concretely, the `stats.probplot` function returns something with a structure like `((X, Y), (c, d, e))`, you will need to return the elements in the position of `X` and `Y` from the `get_coordinates` function (return it as a tuple like `(X, Y)`).

2. Complete the `calculate_distance()` function. This function takes in two floats and returns the calculated distance. The formula you need to use for this function is (in LaTeX form):

$$\sqrt{(x-\frac{x+y}{2})^2+(y-\frac{x+y}{2})^2}$$

   It performs vector projection to the identity line.

> Note: If you can't read LaTeX, you can copy and paste that formula in an online LaTeX compiler like QuickLaTeX.

3. Complete the `find_dist` function. This function takes in a dictionary of distribution names and their respective errors (sum of squared distances). Your code must find the minimum value in the values of the dict and the corresponding key. Returns a tuple that contains the distribution selected and the error calculated. For example `('norm', 9.87546)`.

If your functions all work, and you run the test code that is included in `hw3.py` by uncommenting the problem 2 test part and commenting problem 1 part. If your code is correct you should get the following results for the files `sample_norm.csv`, `sample_expon.csv`,`sample_uniform.csv`, `sample_wald.csv`, `distA.csv`, `distB.csv`, `distC.csv` respectively:

```
('norm', 96.90230310278383)
('expon', 155.95940064211737)
('uniform', 30.477151216719985)
('wald', 2366.701864399592)
('norm', 74.04334951283488)
('uniform', 20.14387449444089)
('wald', 84.49813497099728)
```

> **Important Note**: We will only check your values upto 4 decimal places. Values of data type np.float64 instead of float will also be accepted; for example, both `-0.01868362283870376` and `np.float(-0.01868362283870376)` will be accepted.

What to Submit

Please submit ONLY `hw3.py` with all the appropriate functions filled in for Problem1 and Problem2.

Submitting your code

Please submit the latest version of your code to Gradescope. Do not change the submission file name or the function names within the handout.