# Rendering Plots With Matplotlib

CSE/IT 107L

NMT Department of Computer Science and Engineering

---

> "The greatest value of a picture is when it forces us to notice what we never expected to see."
>
> —John Tukey

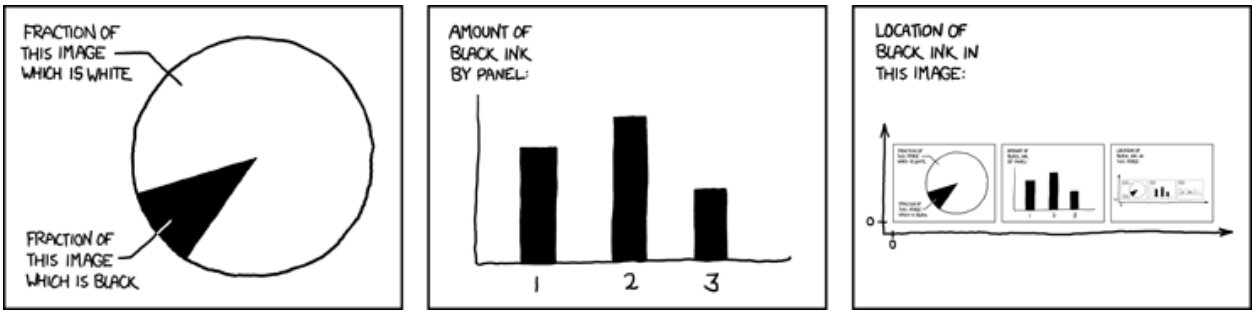



**Figure 1:** Self description. `https://xkcd.com/688/`

---

# Contents



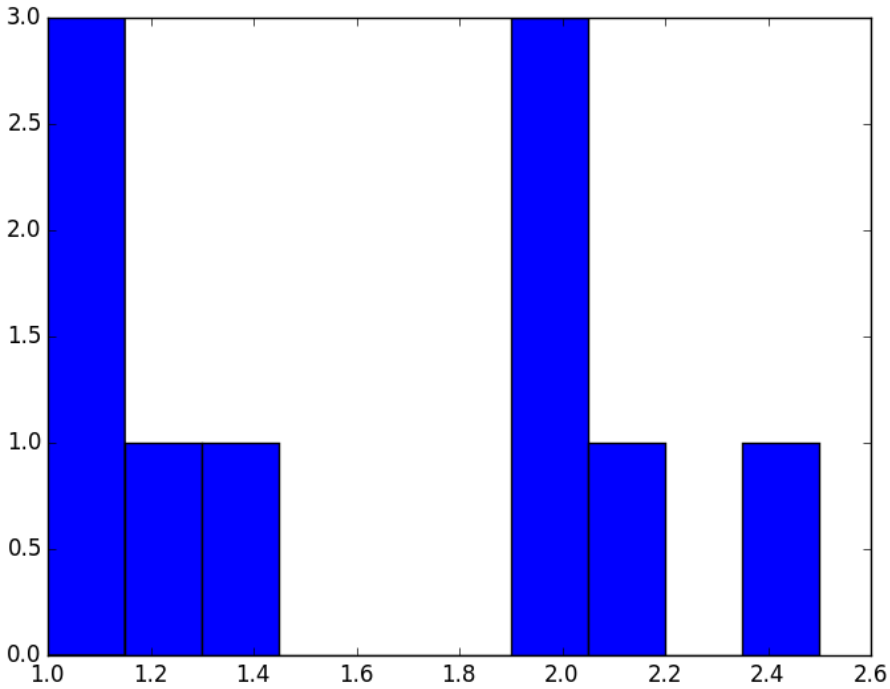
# Introduction

In this lab you will continue to build your understanding about generating 2D graphics using Matplotlib. You will learn about Python's list comprehensions, which help when modifying lists of data. There's also an extra credit where you use the built-in CSV module to read CSV files. The key lesson this lab should teach you is to read Python library documentation. With this skill you can learn to use many of Python's free and open-source libraries.

# 1 Histograms using the `hist` Function

Matplotlib's `plt.hist()` can draw the histogram of a list, which shows how often a value occurs.

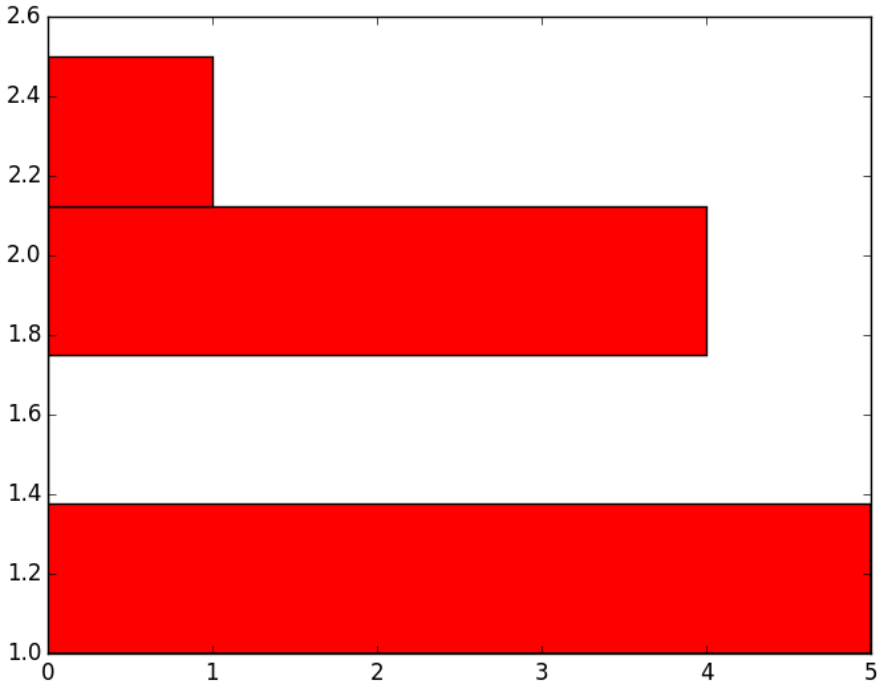
```
>>> import matplotlib.pyplot as plt
>>> plt.hist([1, 1, 1, 1.2, 1.3, 1.9, 1.9, 2, 2.1, 2.5])
>>> plt.show()
```




The `bins` argument is used to draw fewer bins of data. The histogram is rotated by passing `orientation='horizontal'`. See the documentation at:

`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist`

```
>>> import matplotlib.pyplot as plt
>>> plt.hist([1, 1, 1, 1.2, 1.3, 1.9, 1.9, 2, 2.1, 2.5]),
...          bins=4, orientation='horizontal', color='red')
>>> plt.show()
```



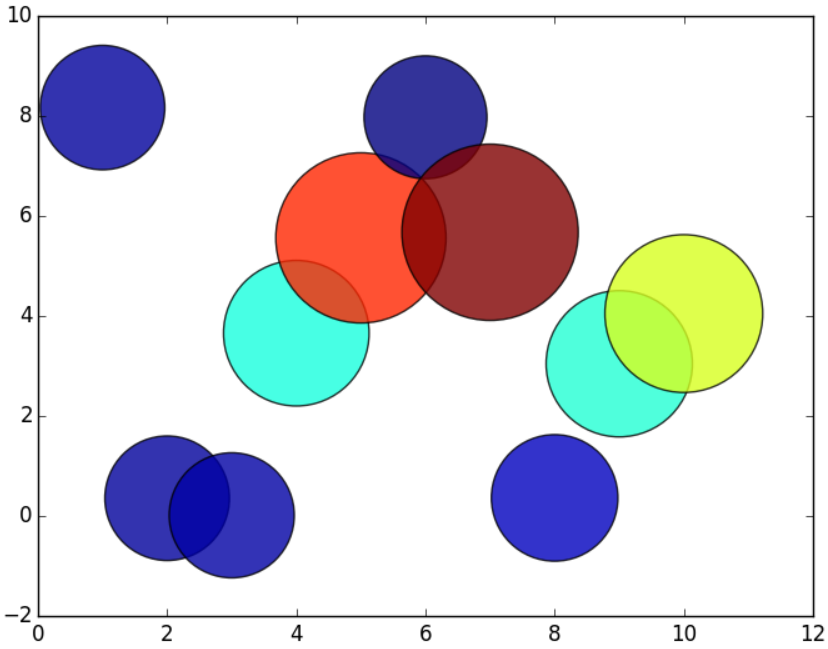

### 1.1 Scatter plots with `scatter` Function

Scatter plots may be generated using the `plt.scatter` function. These plots should be given three arguments `x, y, s`; these are arrays indicating the x and y values of the data and the size of each data point. The `c` and `alpha` keyword arguments can be used to modify color and opacity of data points. Colors are automatically picked if an array of values is given. Documentation available at

`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist`

This plots data with x-values from 0 to 10, randomly generated y-values between 0 and 10, and sizes between 1200 and 30 with some variation. The color scheme is automatically picked according to size.

```
>>> import random
>>> import matplotlib.pyplot as plt
>>> x = [x + 1 for x in range(0, 10)]
>>> y = [(elem + random.random() * 10) % 10 for elem in x]
>>> sizes = [200 * elem + random.random() * 9300 for elem in x]
>>> plt.scatter(x, y, s=sizes, c=sizes, alpha=0.8)
>>> plt.show()
```



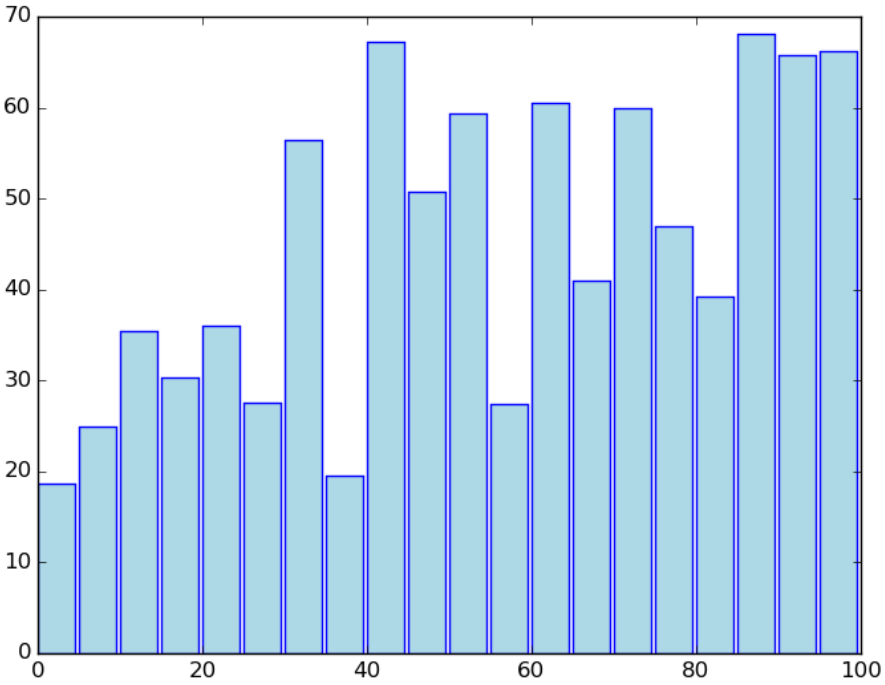

### 1.2 Bar Plots with the `bar` Function

Matplotlib can draw a plot of rectangles, each one bounded by the four edges: `left`, `left + width`, `bottom`, and `bottom + height`. Here's a demonstration using the `color` and `edgecolor` keyword arguments. Documentation is here:

`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar`

The rectangles have randomly generated heights and the same width and space between them.

```
>>> import random
>>> import matplotlib.pyplot as plt
>>> lefts = [x for x in range(0,100,5)]
>>> print(lefts)
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
>>> heights = [x + random.random()*50 + 10 for x in range(0,20)]
>>> print(heights)
[49.714391342616054, 19.172956449982053, 26.30296873725814, 35.617456459379966,
 41.00284816312063, 36.42499187921262, 20.105984017686016, 34.56418052389644,
 24.48833113475763, 54.51007088879054, 51.26910728592864, 27.689576280094144,
 67.57824233176122, 41.335596453966545, 47.96576291980089, 45.292422625538435,
 34.460458928440374, 31.992222247836935, 65.24156104706233, 66.47014183737392]
>>> plt.bar(left=lefts, height=heights, width=4.5, bottom=0,
...         color='lightblue', edgecolor='blue')
>>> plt.show()
```




### 1.3 Drawing Many Plots with the `subplot` Function

If you want to draw many plots in the same figure, use the `plt.subplot` function to split the figure into several rows and columns. The first and second arguments are the number of rows and columns. The third argument is the figure number, this should be between 1 and rows * columns. Documentation available at

`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot`

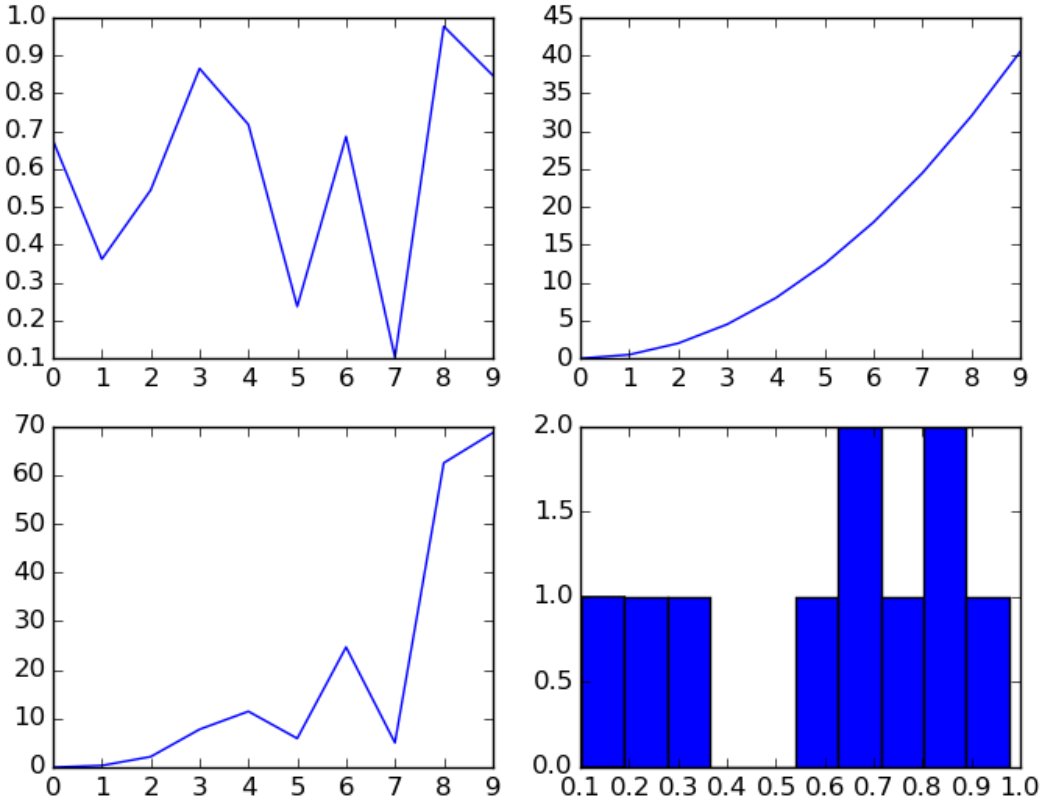
Here we draw 4 plots, three `plt.plots` and one `plt.hist` at the bottom corner.

```
1  import random
2  import matplotlib.pyplot as plt
3
4  x = [elem for elem in range(0,10)]
5  y = [random.random() for elem in x]
6
7  plt.subplot(2, 2, 1)
8  plt.plot(x, y)
9
10 plt.subplot(2, 2, 2)
11 #plt.plot(x, 0.5 * x ** 2)
12 plt.plot(x, [elem ** 2 * 0.5 for elem in x])
13
14 plt.subplot(2, 2, 3)
15 #plt.plot(x, y * x ** 2)
```

```
plt.plot(x, [elem * index ** 2 for index, elem in enumerate(y)])

plt.subplot(2, 2, 4)
plt.hist(y)

plt.show()
```



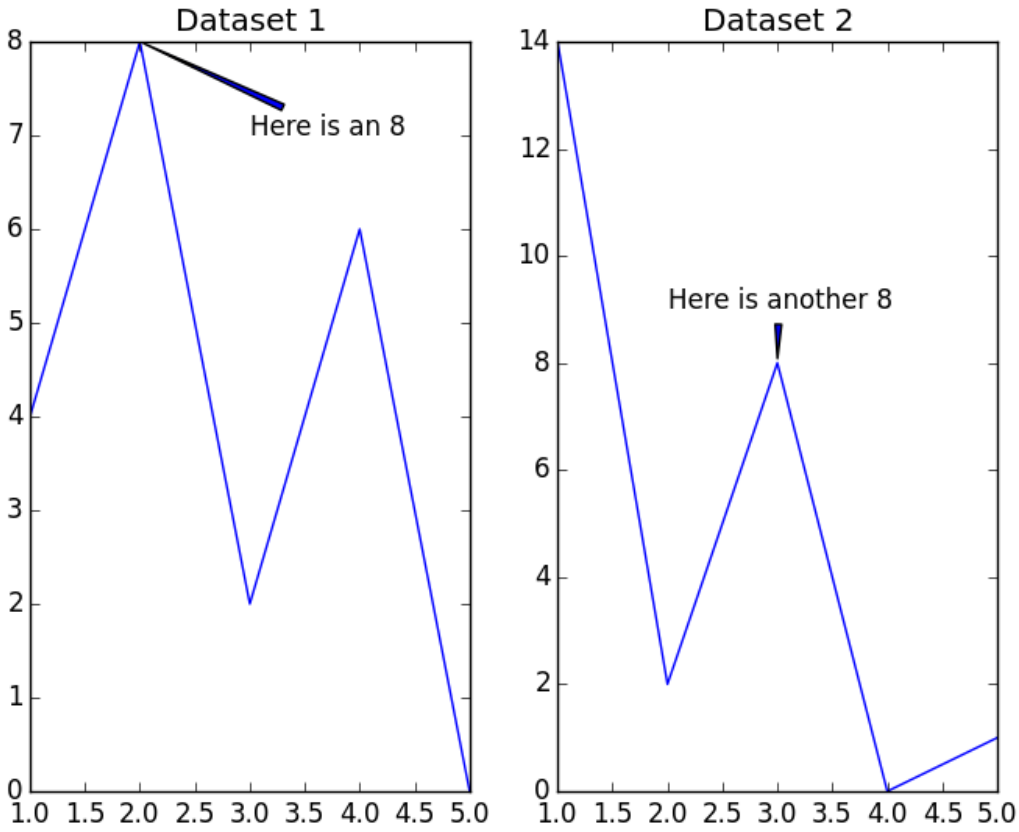
## 1.4 Subplots and Annotations

Here's an example where subplots have annotations and titles.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y1 = [4, 8, 2, 6, 0]
y2 = [14, 2, 8, 0, 1]

plt.subplot(1, 2, 1)
plt.plot(x, y1)
plt.annotate(s='Here is an 8', xy=(2, 8), xytext=(3,7),
             arrowprops={'arrowstyle': "wedge,tail_width=0.25"})
plt.title('Dataset 1')
```

```
11
12 plt.subplot(1, 2, 2)
13 plt.plot(x, y2)
14 plt.annotate(s='Here is another 8', xy=(3, 8), xytext=(2,9),
15              arrowprops={'arrowstyle': "wedge,tail_width=0.25"})
16 plt.title('Dataset 2')
17
18 plt.show()
```



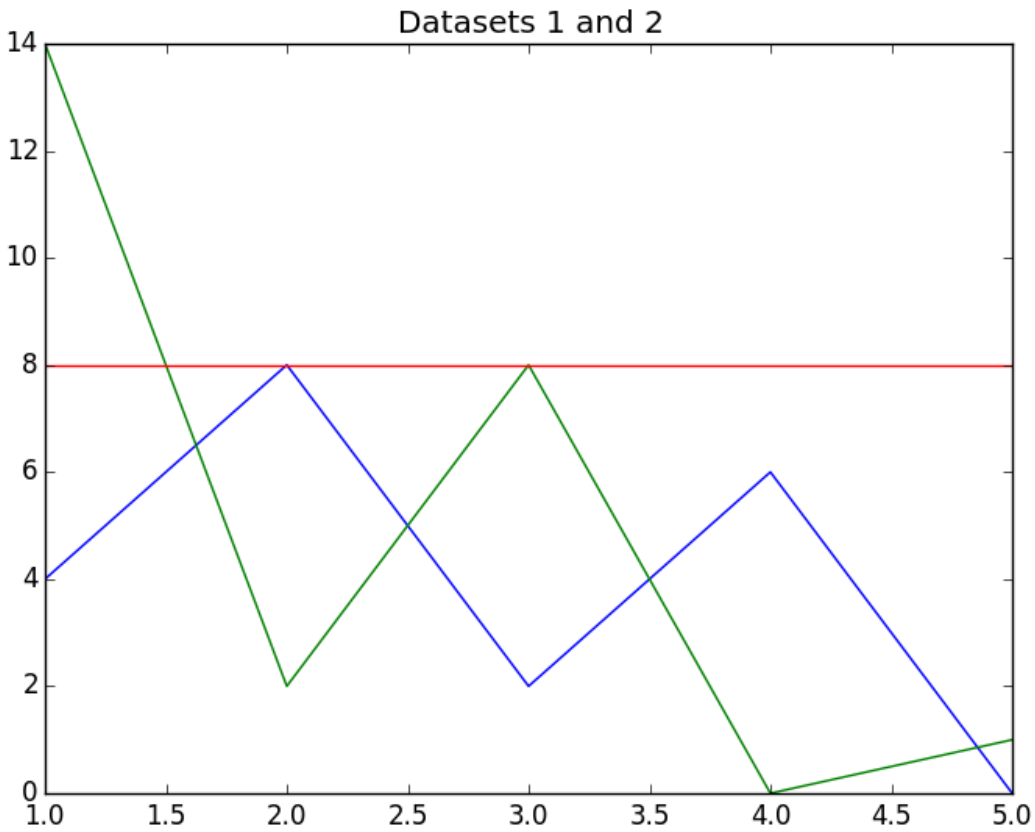

## 1.5 Overlaying Plots

Plots may be overlayed by calling plot functions multiple times. For example, here are plots of two data sets on the same figure. There is also a line drawn at y=8.

```
1 import matplotlib.pyplot as plt
2 x = [1, 2, 3, 4, 5]
3 y1 = [4, 8, 2, 6, 0]
4 y2 = [14, 2, 8, 0, 1]
5
6 plt.plot(x, y1)
7 plt.plot(x, y2)
```

```
plt.plot([1,5], [8,8])
plt.title('Datasets 1 and 2')

plt.show()
```




### 1.6 Saving Plots to a File Using `plt.savefig`

To save a plot to a file, replace `plt.show()` with a call to the `savefig` function:

```
plt.plot([1, 2, 3], [4, 5, 6])
plt.title('Silly plot')
plt.savefig('plot.png')
```

You can save to different formats by changing the extension of the filename. For example, the function would produce a JPG file when given a parameter "filename.jpg".

# 2 Exploring APIs

An important part of this lab is reading module documentation. You have already seen one third party module (Matplotlib), and built-in modules such as `math`. Almost every good Python module has good documentation and if you want to learn how to use one you should start by reading its documentation. For example, Matplotlib supports animation, image rendering, stream plots, error plots, contour plots, log scaled axis, and many other features. You can see examples and detailed specifications in the Matplotlib docs.

## 2.1 Other Libraries

Other powerful, third-party and well-documented Python modules include 1. *Scrapy:* a web crawling library. 2. *Newspaper:* an Easy to use natural language processor and news downloader. 3. *astropy:* astronomy and astrophysics packages. 4. *vtk:* highly functional 3D visualization library. 5. *pillow:* the friendly fork of the Python Imaging Library. 6. *OpenCV:* a computer vision library. 7. *Django:* used to deliver and manage dynamic websites; powers many large websites.

Visit their websites for information on usage and installation. Here's a list of more interesting libraries:

`https://github.com/vinta/awesome-python`

# 3 Exercises

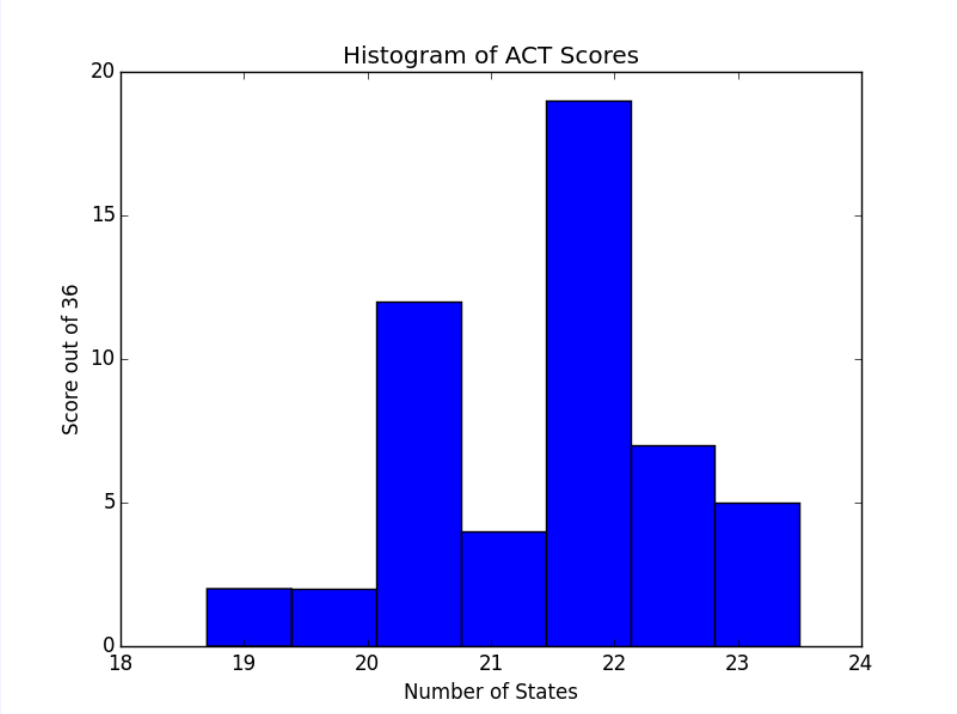
**Labels**

Remember to label your axes using `plt.xticks()` and `plt.yticks()` and title your graph using `plt.title()`. Read the documentation to find out more.

**Exercise 3.1** (design, scorehist.py).
Before attempting to code this problem, create a file `design` that contains some analysis of how you think the problem will be solved. Examples include but are not limited to: a flowchart of events in the program, pseudocode, or a step-by-step process written in plain English. If you choose to scan your design, please make sure that it is legible.

This exercise relies on the `readscores.py` and `actsat.txt` files from the File I/O lab. You may also rewrite that code using the CSV module.

Write a program that generates a histogram of average ACT scores. For example:




**Exercise 3.2** (scorebar.py).
This exercise relies on the `readscores.py` and `actsat.txt` files from the File I/O lab. You may also rewrite that code using the CSV module.

Write a program that draw the following charts.

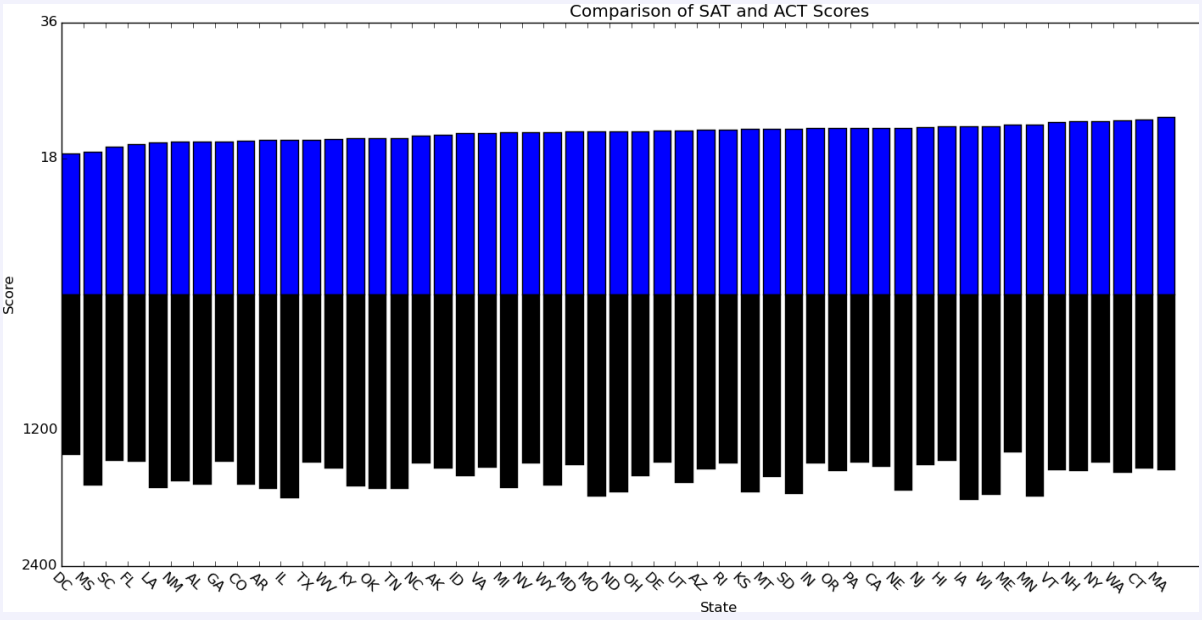
1. A bar chart where each state/territory is represented by two bars, one for the composite ACT score and the other for the total score of all 3 SAT tests.

Each SAT section is out of 800 points, while the ACT is out of 36 points. Each bar should have the same maximum heights, but the labels should have the correct SAT or ACT score. See the example image.

2. Produce the same chart as in part 2, but only for states in which less than 50% take the ACT and more than 50% take the SAT. (There should be 21 states/territories like this.)

*Hint:* Notice the scale is different for each plot. One way of adjusting the scales is to divide each number in the data so that every number is between 0 and 1. For the SAT scores, a score of 0 would be 0 and a score of 2400 would be 1.0. Then assign the label "0" to 0, and the label "1800" to 0.5, and the label "2400" to 1.0. Similarly, -1 would have label "36".

This is an example of the first bar plot:




**Note**

None of these plot have to be exactly the same as the examples, they need only convey the same information in an understandable plot.

**Exercise 3.3** (README).

For the following labs you will be required to create and submit a README file.

A README file is a simple text file that contains basic information regarding the purpose and use of the software program, utility, or game. README files often contain instructions or additional help regarding the software it accompanies.

For this lab you must have the following in your README file::

- **Purpose**: Describes what each program does and what problem it solves. You can keep this breif.

- **Conclusion**:
  - What you learned during the lab? What new aspect of programming did you learn from the lab? Be analytical about what you learned.
  - Did pair programming help in solving the problems and completing the prelab? Did you have problems with your budy?
  - Did you work with your budy on the lab? What sections did you discuss? Did you and your budy carry out a review session with each others' code?
  - Did you encounter any problems? How did you fix those problems?
  - What improvements could you make?

The conclusion does not have to be lengthy, but it should be thorough. You will include the README file with your submissions.

## 4 The CSV Module

CSV files are used to store tabular data. They have comma separated values on each line. Most spreadhseet programs support CSV output and input. Python comes with the CSV module for reading and writing such files. It supports any delimiter so you can also manage space-separated and tab-separated data. To use the module, create a CSV reader using the filename of the data file. For example, suppose we have the file `short.csv`. It has column labels on the first row and columns are separated by commas.

```
Entity,x         ,y,  z
Fruit tree,1,2,3
Bats, 1, 3, 10
```

We can use the `csv.DictReader` function to parse this data. It takes an open file as the first argument and returns something that you can use a for-loop to traverse. It essentially returns a list of dictionaries. For example,

```
>>> import csv
>>> with open('short.csv') as f:
...     l = [e for e in csv.DictReader(f)]
>>> print(l)
[{'Entity': 'Fruit tree', 'x': '1', 'y': '2', 'z': '3'},
 {'Entity': 'Bats', 'x': ' 1', 'y': ' 3', 'z': ' 10'}]
```

Read the documentation to learn about writing CSV files and the `csv.Sniffer`, which can be used to deduce the format of a CSV file.

`https://docs.python.org/3/library/csv.html`

The data from `http://www.gapminder.org/data/` was converted to CSV and parsed using Python's CSV module to form the file `country_data.py`.

**Extra Credit (10 Points)**

Use matplotlib to draw scatter plots relating worldwide life expectancy, income per person, and population for the years 2010 and 1960.

You may access the data by completing the extra credit or by importing `country_data.py`. The data is stored in the `data1960` and `data2010` variables and looks like:

```
data1960 = [{'country': 'Andorra',
    'expectancy': None,
    'gdp': 15234.0,
    'population': 13414.0},
   {'country': 'Eritrea',
    'expectancy': 41.8278,
    'gdp': 885.0,
    'population': 1407631.0}, ...
```

See "200 Years, 200 Countries" by Hans Rosling if you want a good presentation on this data.

Notice the special value `None` is present. To check if a variable `x` is `None`, type `x == None` or `x is None`.

You are required to:

1. Draw a scatter plot with x and y coordinates set to income and life expectancy. If there is a `None` value, replace it with the minimum income or the minimum life expectancy or an area set to zero.
2. The area of each data point should be related to the population of every country. You can choose how to scale the data, but to make it look like the example, you may use:

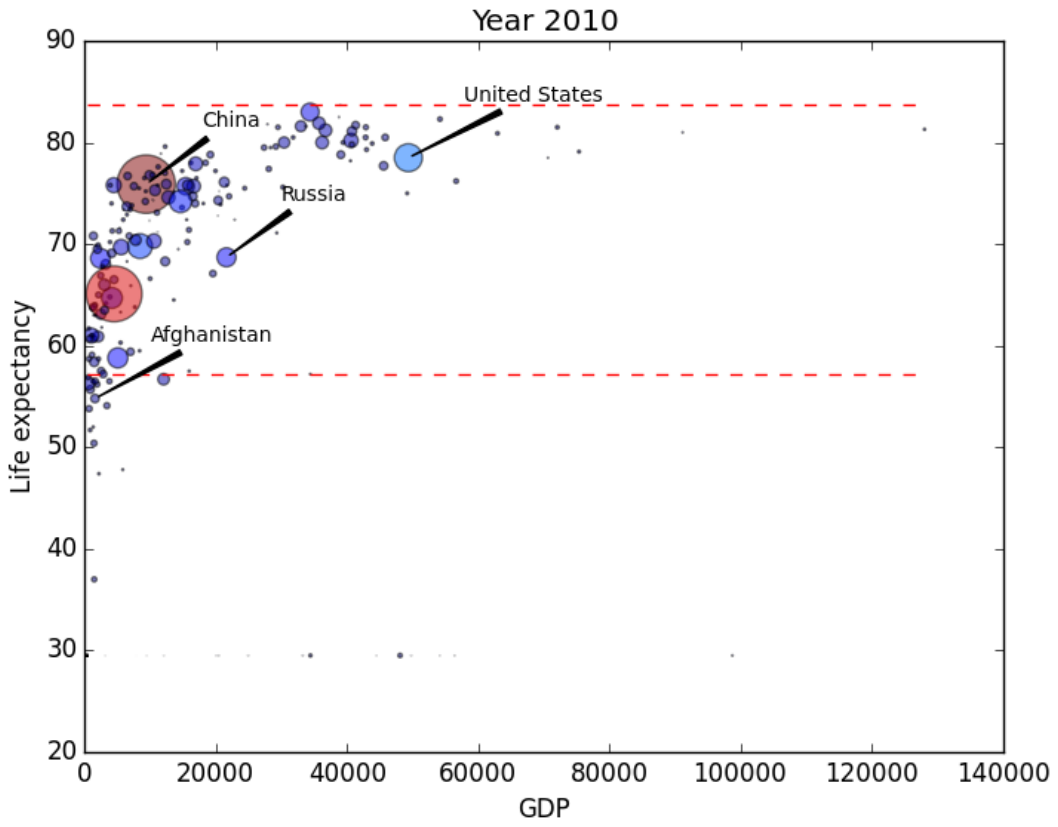
   ```
   scaled_pop = [800 * p / max(pop) for p in pop]
   ```

3. The scatter plots must be labeled and titled. X label: "Income", y label: "Life expectancy", title: "Year (year of data)".
4. Annotate China, Russia, United States, and Afghanistan as shown. Annotation text should be drawn away from the annotation coordinates. You can get an arrow like in the example with this keyword argument:

   ```
   arrowprops={'arrowstyle': 'wedge,tail_width=0.25', 'color': 'black'}
   ```

5. Draw dashed red lines at the mean and at the max life expectancy. Python's max function is useful for this.

Here is the plot for the 2010 and 1960 data:




**Supplementary Files**

The `country_data.py` file is available on canvas.

## Submitting

You should submit your code as a `.zip` that contains all the exercise files for this lab. The submitted file should be named

`cse107_firstname_lastname_lab11.zip`

**Upload your .zip file to Canvas.**

## List of Files to Submit



Exercises start on page 9.