Plotting

Plotting with Python

`Matplotlib` is a Python plotting library, which is inspired by the MATLAB plotting functionality, so the commands used are similar to those used in MATLAB.

`Pyplot` is a shell like interface to Matplotlib to make it easier to use. It is a collection of command style functions that make Matplotlib work like MATLAB plotting functionality.

The text talks about `Pylab`, which combines both the `Pyplot` and `NumPy` packages into a single namespace, to make it easier to import.

Source: https://matplotlib.org/users/pyplot_tutorial.html

Note about Plotting with Spyder

Depending on the Spyder default settings, plots may appear in a figure window or in the iPython console.

To open plots in a figure window, per session, you should type the command:

`%matplotlib qt` (resets when session ends)

To open plots in a figure window by default across sessions, go to:

`Tools->Preferences`, select **`iPython Console ->Graphics`**

then set **`Graphics Backend`** to **`Automatic`**

Plotting with pyplot

The `pyplot` package contains several commands used for creating and formatting plots in Python.

Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels.

In `matplotlib.pyplot` various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current plot axes.

Pyplot summary: https://matplotlib.org/api/pyplot_summary.html

Basic Plotting Functions

The `plot` function has different forms, depending on the input arguments.

If y is a list, plot(y) produces a piecewise graph of the elements of (y) versus the index of the elements of (y).

Since Python ranges start with 0, the default x list has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

If you specify two lists as arguments, plot(x,y) produces a graph of y versus x.

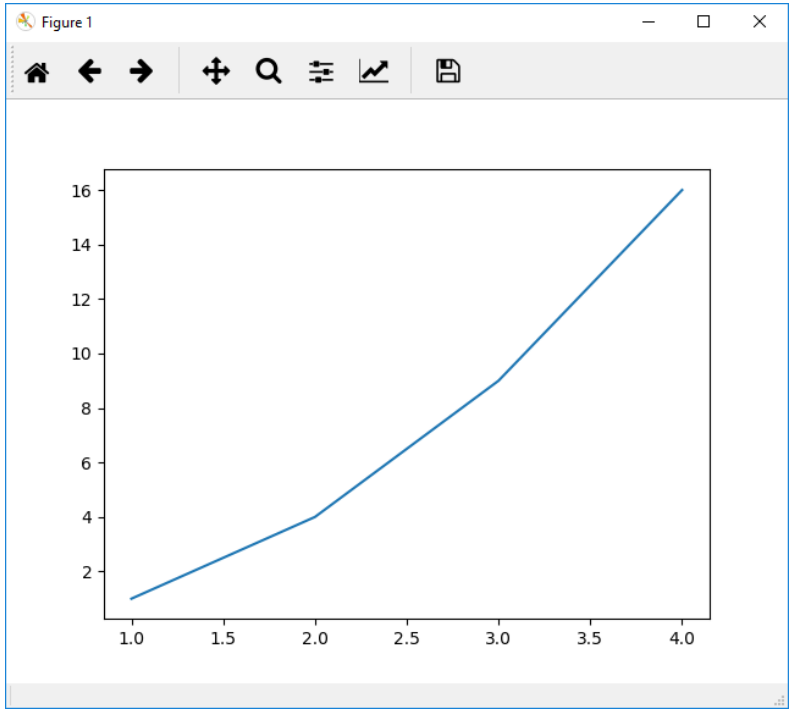
Both lists **must** have the same number of elements

Creating plots with pyplot

plot() is a versatile command and will take an arbitrary number of arguments.

For example, to plot x versus y, you can issue the command :

```
plot([1, 2, 3, 4], [1, 4, 9, 16])
```



Creating plots with pyplot

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot.

The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string.

The default format string is ‘b-‘, which is a solid blue line. For example, to plot the above with red circles, you would issue the command:

```
plot([1,2,3,4], [1,4,9,16], 'ro')
```

Plotting Format Options

line color		line marker		line style	
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star		
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

Formatting Plots with pyplot

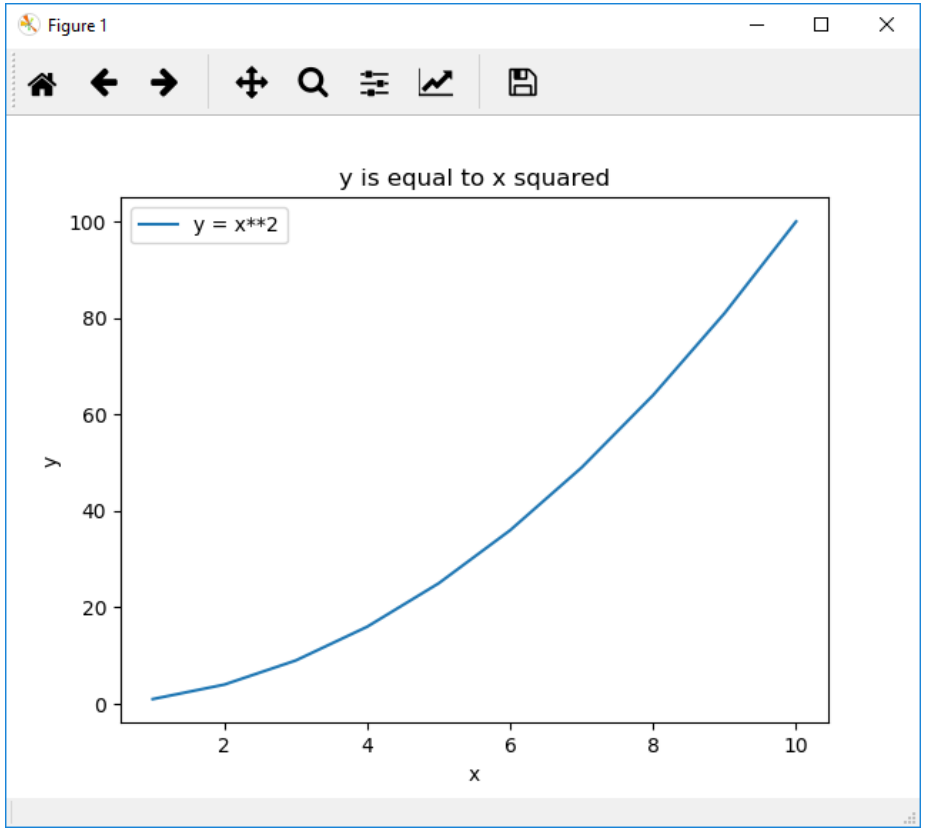
The result of a `plot` function call is not a finished product, since there are no titles, axis labels, or grid lines on the plot.

The details can be added with the functions:

- `title`,
- `xlabel`,
- `ylabel`,
- `grid`
- `legend`
- `axis`
- `etc…`

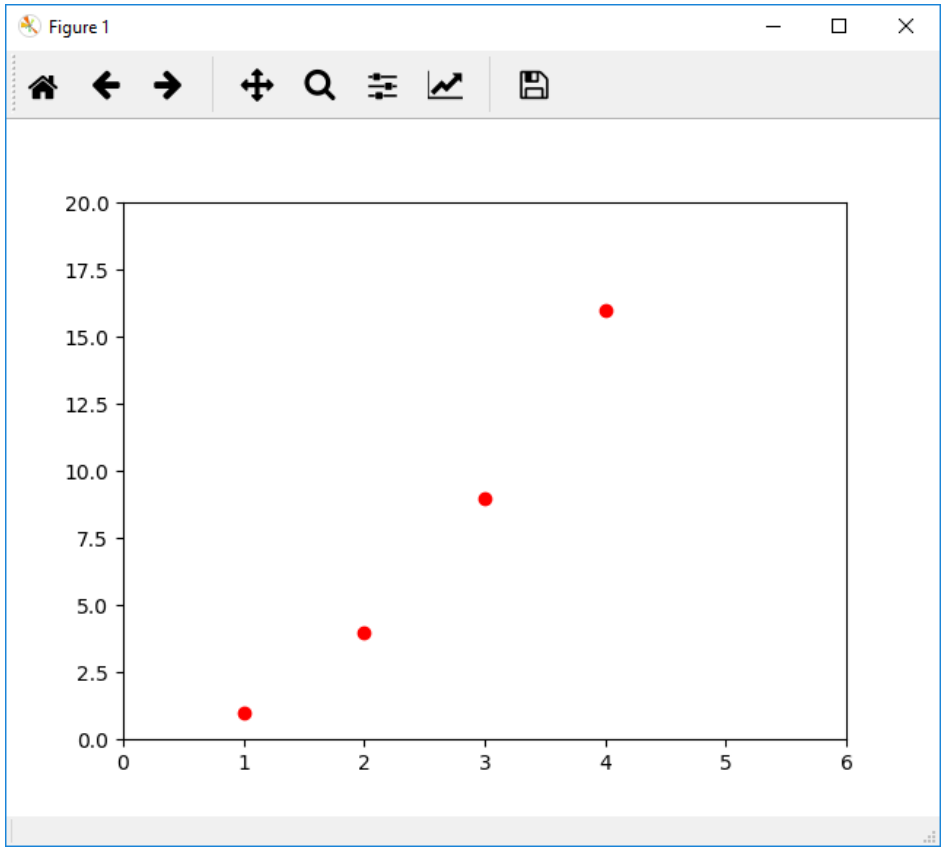
Formatting with pyplot – numpy arrays

```
from matplotlib.pyplot import *
from numpy import *
clf()
x = arange(1,11)
y = x**2


plot(x,y)
xlabel('x')
ylabel('y')
title('y is equal to x squared')
legend(['y = x**2'])
```



Creating plots with pyplot - axis

```
from matplotlib.pyplot import *

#clears the current figure window
clf()

#creates the plot with x, y values red circle markers
plot([1,2,3,4], [1,4,9,16], 'ro')

#changes the limits of the x and y axis
#[xmin, xmax, ymin, ymax]
axis([0, 6, 0, 20])
```

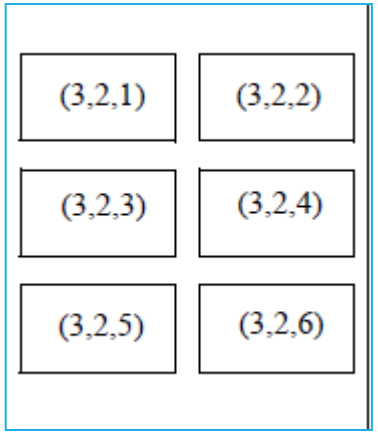


Displaying Multiple Plots in One Figure

`subplot(m,n,p)`

This splits the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot.

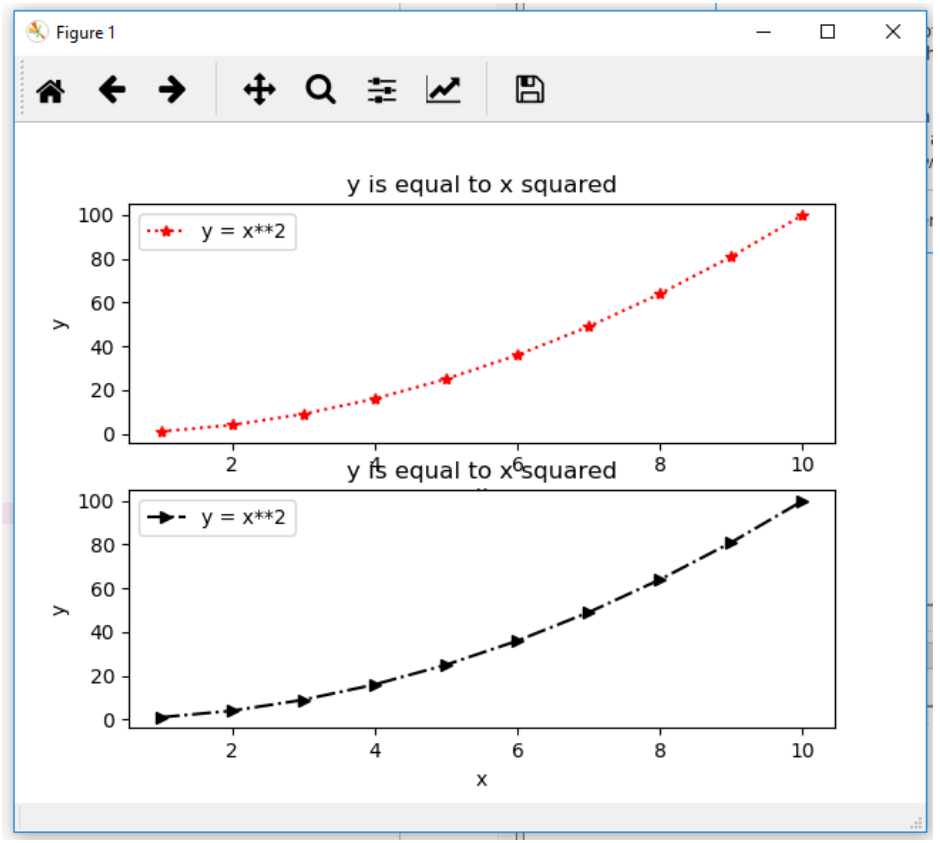
For example, the command `subplot(3,2,1)` creates six areas arranged in three rows and two columns as shown, and makes the upper left subplot current.



Plotting: Subplots

```
clf()
x = arange(1,11)
y = x**2

subplot(2,1,1)
plot(x,y,'r:*')
xlabel('x')
ylabel('y')
title('y is equal to x squared')
legend(['y = x**2'])

subplot(2,1,2)
plot(x,y,'k-.>')
xlabel('x')
ylabel('y')
title('y is equal to x squared')
legend(['y = x**2'])
grid('on')
grid('off')
```

See: 11_plot5.py



Question:

Write a script to initialize two numpy arrays, **max_temp** (with random integers between 1 and 45] and **min_temp** (with random integers between -20 and 20] for the maximum temperatures and minimum temperatures of all months in a year, respectively.

Calculate the absolute temperature differences for each month (difference between maximum and minimum temperatures) and store in numpy array, **diff_temp**.

Find the average temperature (**average)** of the **diff_temp** array

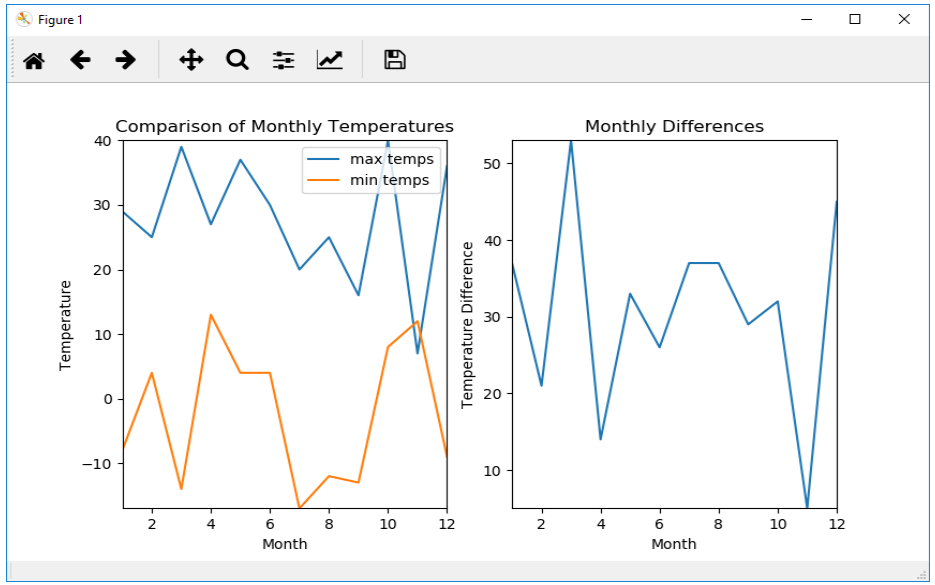
Display the month numbers of those months whose temperature difference is more than **average**

Plot the maximum and minimum temperatures in the first chart and the temperature differences in the second chart as given below. Put the titles and axis labels of the charts as below.

Arrange the axis limits appropriately (**x axis will have the bounds:** the month numbers in both charts and **y axis will have the bounds:** the maximum value of maximum temperatures list and the minimum value of the minimum temperatures list, in the first chart and the maximum value of temperature differences and the minimum value of the temperature differences in the second chart.

Output:

Plot:



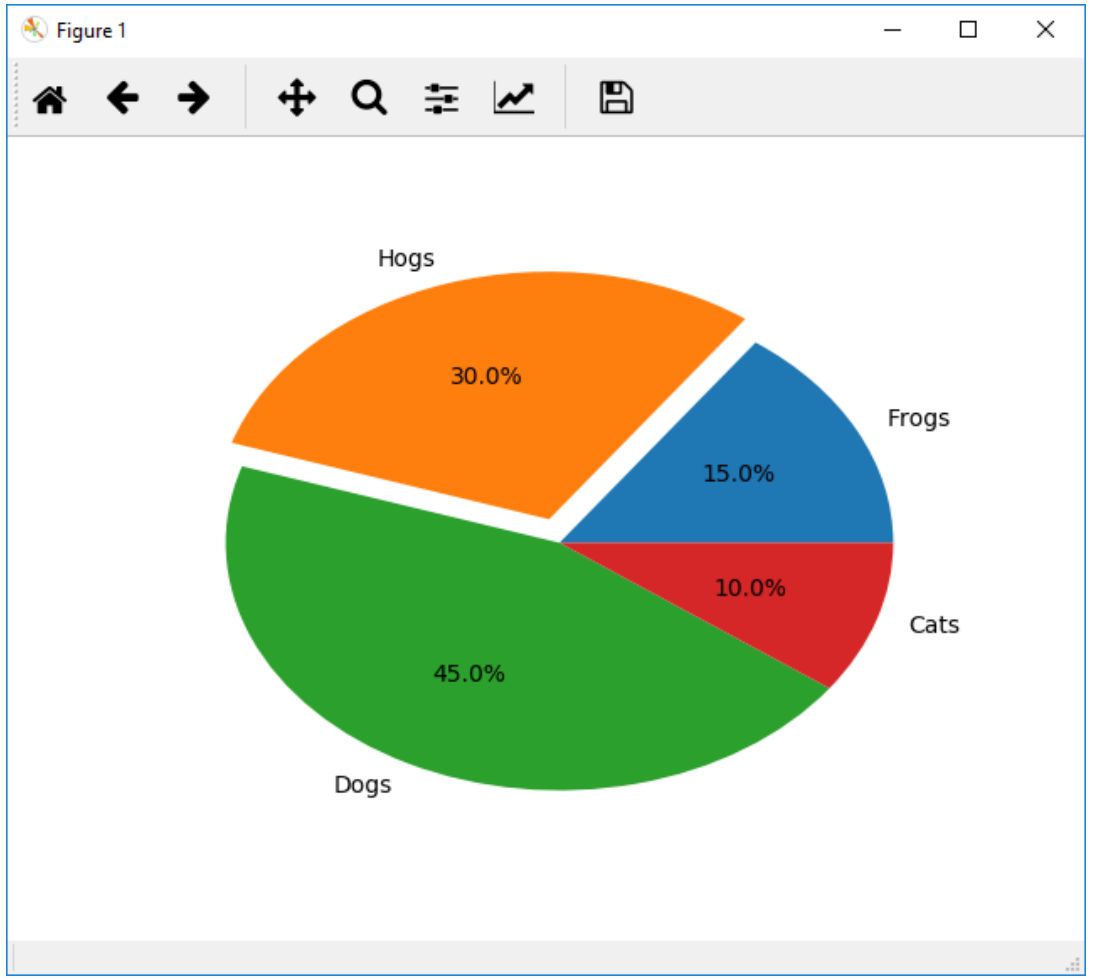
Sample Output:

```
[43, 34, 15, 1, 36, 3, 26, 33, 38, 20, 4, 45]
[-19, 20, -13, 16, -20, 15, -9, -14, 15, -20, -10, 9]
Monthly differences:  [62, 14, 28, 15, 56, 12, 35, 47, 23, 40, 14, 36]
Average difference:  31.833333333333332
Months with difference above average:
[1 5 7 8 10 12]
```

See: **11_plotQuestion.py**

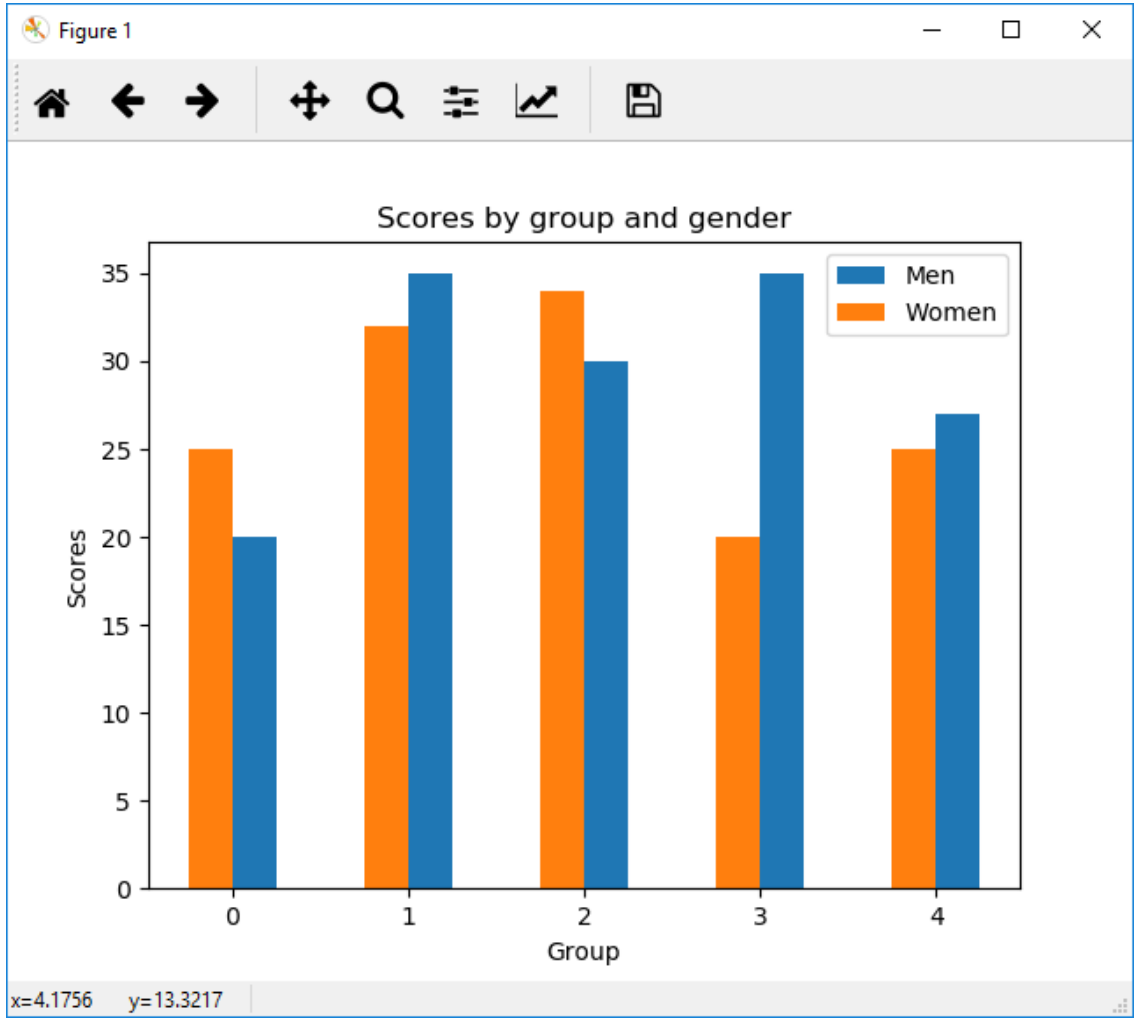
Pie Charts

```
from matplotlib.pyplot import *

clf()

# Pie chart,slices will be ordered,
# plotted counter-clockwise:

labels = 'Frogs', 'Hogs', 'Dogs', 'Cats'
sizes = [15, 30, 45, 10]

# only "explode" the 2nd slice(i.e. 'Hogs')
explode = (0, 0.1, 0, 0)

pie(sizes, explode=explode,
labels=labels,autopct='%1.1f%%')
```



Bar Charts

```
from matplotlib.pyplot import *
from numpy import *

clf()
n_groups = 5
index = arange(n_groups)

means_men = (20, 35, 30, 35, 27)
means_women = (25, 32, 34, 20, 25)

bar_width = 0.25
bar(index,means_men, bar_width,align='edge')
bar(index,means_women, -bar_width,align='edge')

xlabel('Group')
ylabel('Scores')
title('Scores by group and gender')
legend(['Men', 'Women'])
```



Histogram

Often you want to get an understanding of the distribution of certain numerical variables within it when exploring a dataset.

One way of visualizing the distribution of a single numerical variable is by using a histogram.

A histogram divides the values within a numerical variable into “bins” and counts the number of observations that fall into each bin.

By visualizing these binned counts in a columnar fashion, we can obtain an understanding of the distribution of values within a variable.

A histogram can only be used to plot numerical values and it is usually used for large data sets. It is useful for detecting outliers and/or gaps in the data set.

Histogram – Creating Manually

Example: We want to construct a histogram of the following scores in a math exam where the maximum possible mark is 20.

Scores:

```
[2, 4, 14, 14, 16, 17, 13, 16, 7,2, 4, 14, 14, 16, 17, 13, 16, 7,8, 9, 10, 11, 19, 18,
 15, 15, 16,8, 9, 10, 11, 19, 18, 15, 15, 16,13, 12, 7, 8, 9, 12, 11, 18]
```

1. First order the data to make it easier to group.

```
[2, 2, 4, 4, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 14,
14, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 17, 17, 18, 18, 18, 19, 19]
```

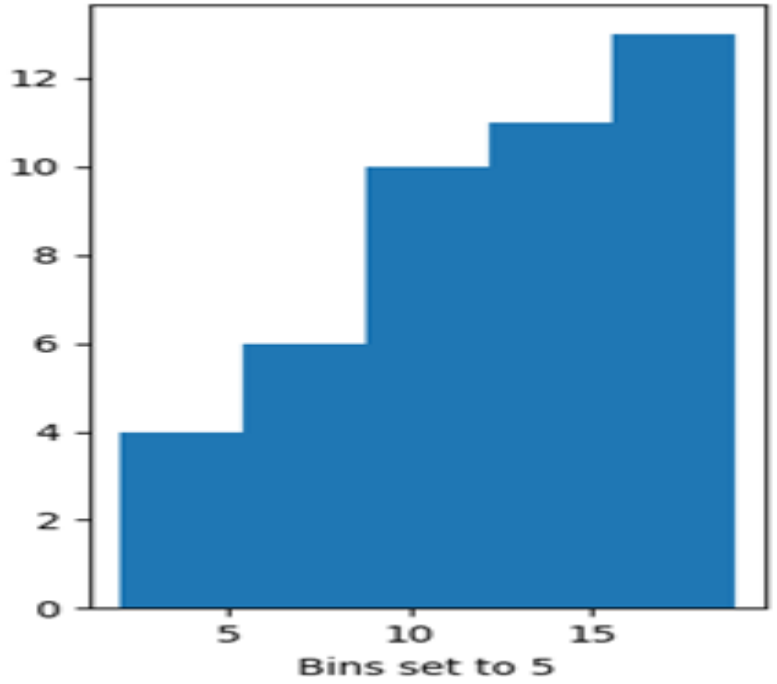
1. Next, look at the data and find the minimum and maximum values to choose the width of each 'bucket'. The minimum is 2 and the maximum is 19. Let's choose a width of 4. So there will be 5 buckets: [2-5.4), [5.4-8.8), [8.8 - 12.2), [12.2-15.6), [15.6-19].

Histogram – Creating Manually

3) Next, calculate the number of each values in each bucket.

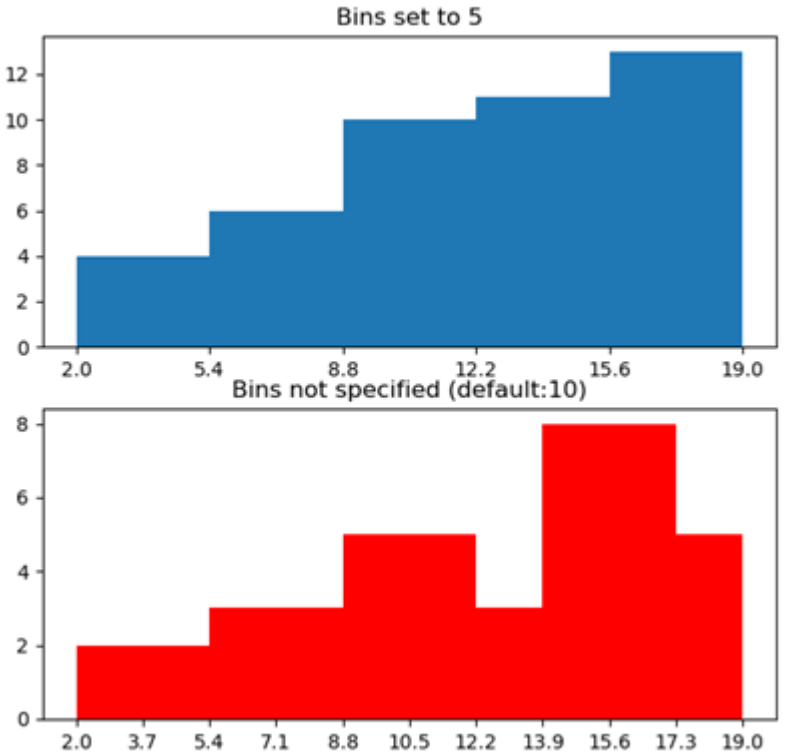
2 - 5.4	4
5.4 - 8.8	6
8.8 - 12.2	10
12.2 - 15.6	11
15.6 - 19	13

4) Finally, create a histogram that displays the frequency data.



Histogram – Example with Scores

```
import numpy as np
import matplotlib.pyplot as plt

score_arr = np.loadtxt('scores.csv',skiprows=1)

plt.figure(1) #opens a new active figure window
plt.clf() #clears the figure window

plt.subplot(2,1,1)
res = plt.hist(score_arr,5) #plots scores with 5 bins/buckets
plt.xticks(res[1])
plt.title('Bins set to 5')

ax = plt.subplot(2,1,2)
counts,bins, patches = plt.hist(score_arr, color = 'red')
plt.title('Bins not specified (default:10)')
ax.set_xticks(bins.round(2))
```



See: `11_histogram_scores.py`

Additional Details:

For more information about plotting in Python:

https://matplotlib.org/api/pyplot_summary.html

Question 1:

Plot the function $y=3x^3 – 26x + 10$, and first and second derivatives, for -2<=x<=4, (increment by 0.1) all in the same plot.

See: `11_Q1.py`

Question 2:

Make two separate plots on the same figure window for the function

$$f(x)= (x-3)(x+2)(4x-0.75)-e^{1/2}/2,$$

one plot for -2<=x<=1 and one for -4<=x<=4 (increment by 0.1).

See: `11_Q2.py`

Steps to Understanding Experimental Data

Conduct an experiment to gather data

- Physical(e.g.), in a biology lab
- Social(e.g. questionnaires)

Use theory to generate some questions

Design a computation to help answer questions about the data.

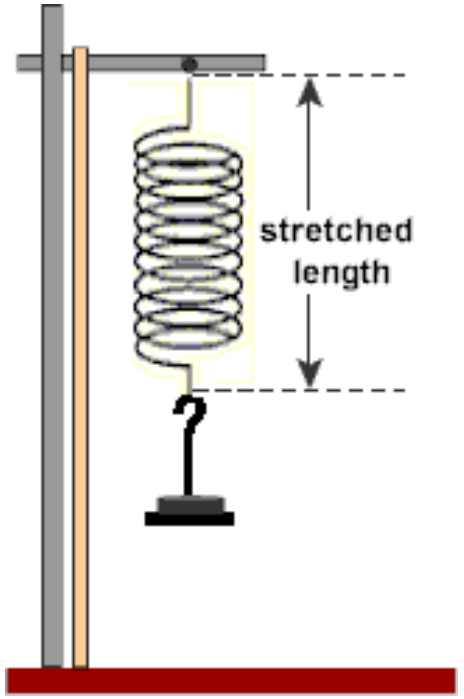
Example - Springs

When springs are compressed or stretched by some force, they store energy.

When the force is no longer applied they release the stored energy.

Imagine an experiment where a weight is hung from the bottom of a spring, stretching the spring.

The data we obtain is the difference between the un-stretched and the stretched spring length vs. the mass applied to the spring.



Hooke's Law

The theory that we are going to use to analyse the data is Hooke's law of elasticity.

The force F stored in a spring is ***linearly*** related to the distance the spring is compressed/stretched.

Holds for a variety of materials and systems but does not hold for very large force.

All springs have an elastic limit beyond which they cannot be stretched.

Hooke's Law

The formula is:

```
F = -kx
```

F: represents the force

x: is the stretched distance

k: is the spring constant If the spring is stiff k is large, if the spring is weak, k is small.

Force: `(mass * gravity)`

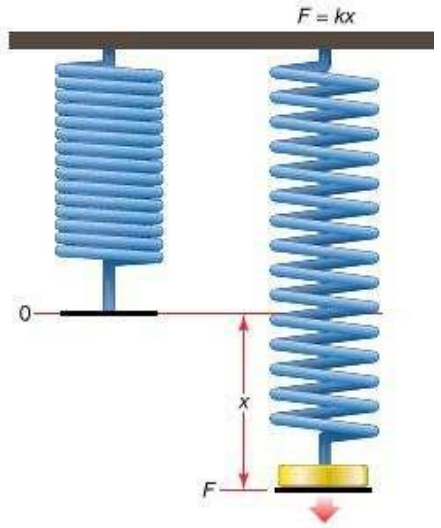
```
F = m * 9.81
```

Spring constant(`k`):

```
k = -(m*g)/ x -> (Newtons/meter)
```

Data:

Distance (m)	Mass (kg)
0.0865	0.1
0.1015	0.15
0.1106	0.2
0.1279	0.25
0.1892	0.3
0.2695	0.35
0.2888	0.4
0.2425	0.45
0.3465	0.5
0.3225	0.55
0.3764	0.6
0.4263	0.65
0.4562	0.7



The Program:

- The following function read distance and mass data from a text into two lists, and returns the lists containing the data.

```
import pylab
def getData(filename):
    dataFile = open(filename, 'r')
    distances = []
    masses = []
    dataFile.readline()
    for line in dataFile:
        d,m = line.split(' ')
        distances.append(float(d))
        masses.append(float(m))
    dataFile.close()
    return(masses, distances)
```

Plotting the Data:

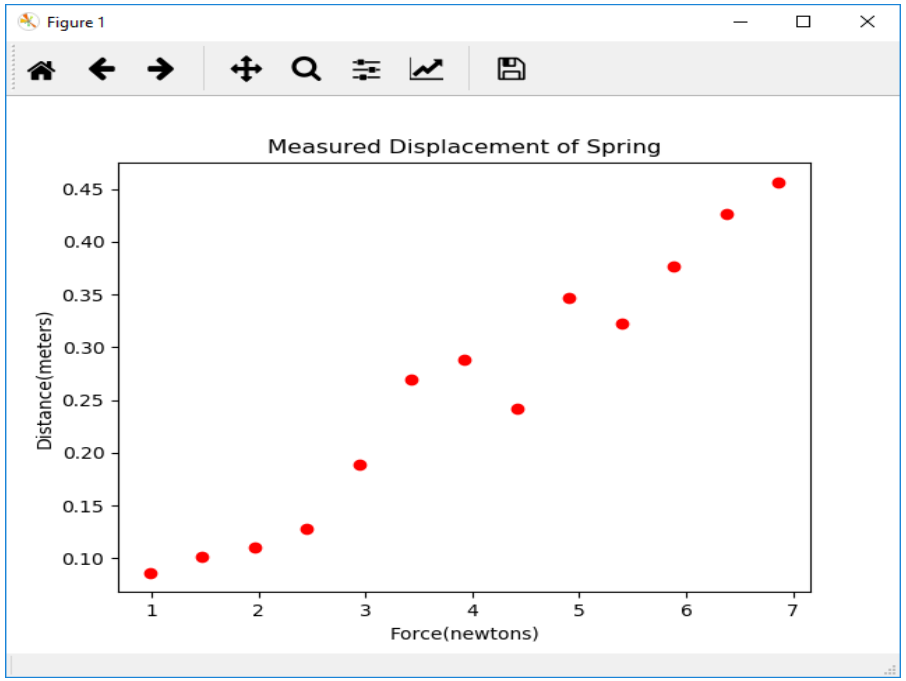
The following function gets the experimental data from the file, calculates the force from the data, and produces a plot of the data.

```
def plotData(inputfile):
        masses, distances = getData(inputfile)
        forces = []
        for m in masses:
                #9.81 represents gravitational force
                forces.append(m * 9.81)
        pylab.plot(forces, distances, 'ro')
        pylab.title('Measured Displacement of Spring')
        pylab.xlabel('Force(newtons)')
        pylab.ylabel('Distance(meters)')

plotData('11_springData.txt')
```

The Plot:

The plot shown is not what Hooke's law predicts.

Hooke's law tells us that the distance should increase linearly with the mass. I.e. the points should lie on a straight line.



Experimental Error

When we take real measurements, the experimental data are rarely a perfect match for the theory.

Measurement error is to be expected, so we should expect the points to lie around a line rather than on a line.

It would be nice to have a line that represents our best guess of where the points would have been if we had no measurement error.

The usual way to do this is to fit a line to the data, called linear interpolation.

Fitting the Data

Fitting analytical curves to experimental data is a common task in engineering.

To fit a curve, linear regression is used.

The function used in linear regression is the least squares function.

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

Let observed and predicted be vectors of equal length where observed contains the measured points and predicted the corresponding data points on the proposed fit.

`polyfit` function

Pylab/NumPy provide built-in functions to find a polynomial fit for the data points.

```
np.polyfit(observedX, observedY, n )
```

Finds ***coefficients*** of a polynomial of degree n, that provides a best least squares fit for the observed data.

- n = 1 -> best line $y = ax + b$
- n = 2 -> best parabola $y = ax^2 + bx + c$

Polyfit returns coefficients a,b for a polynomial with degree of 1. We can find the predicted distances using these values.

Fitting the Data with `polyfit`

The program below fits the data using the polyfit function and plots both the measured points and the linear model.

```
def fitData(inputfile):
    masses, distances = getData(inputfile)
    forces = []
    for m in masses:
        #9.81 represents gravitational force
        forces.append(m * 9.81) plot(forces, distances, 'ro',label = 'Measured Displacements')
    title('Measured Displacement of Spring')
    xlabel('|Force|(newtons)')
    ylabel('Distance(meters)')

    #find linear fit
    a,b = polyfit(forces, distances, 1)
    predictedDistances = []
    for f in forces:
        predictedDistances.append(a*f+b)

    plot(forces, predictedDistances,
                          label = 'Displacements predicted by '\
                 'linear fit')
    legend(loc = 'best')

fitData('11_springData.txt')
```

The Plot:

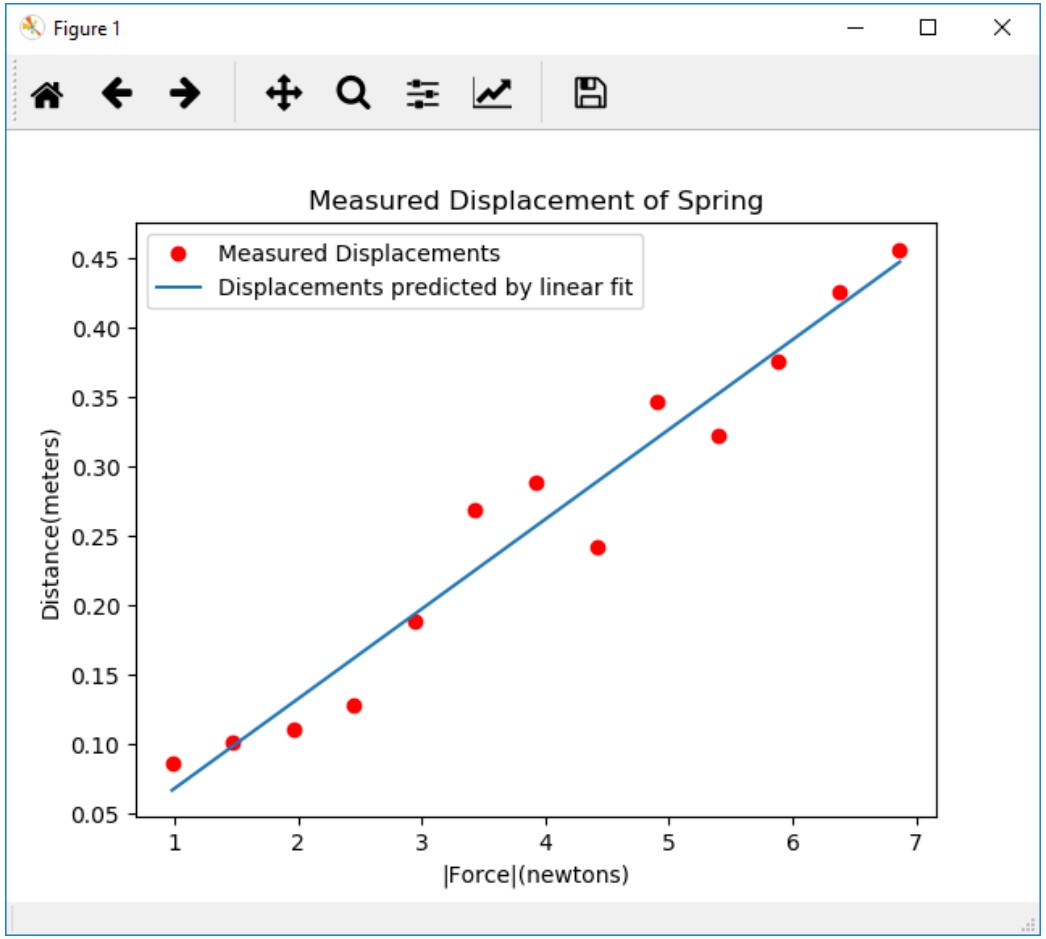


Figure 1
Measured Displacement of Spring
Measured Displacements
Displacements predicted by linear fit
Distance(meters)
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
1
2
3
4
5
6
7
|Force|(newtons)

Polyval Function (NumPy package)

`polyval(p,x)` evaluates the polynomial `p` at each point in `x`. `p` is a vector of length `n+1` whose elements are the coefficients of an nth degree polynomial.

```
p[0]*x**(N-1) + p[1]*x**(N-2) + ... + p[N-2]*x + p[N-1]
```

polyval

```
def fitData(inputfile):
    masses, distances = getData(inputfile)
    forces = []
            for m in masses:
                    #9.81 represents gforce
                    forces.append(m * 9.81)
    plot(forces, distances, 'ro',label = 'Measured Displacements')
    title('Measured Displacement of Spring')
    xlabel('|Force|(newtons)')
    ylabel('Distance(meters)')

    #find linear fit using polyfit/polyval
    fit = polyfit(forces, distances, 1)
    predictedDistances = polyval(fit, forces)

    plot(forces, predictedDistances,
                            label = 'Displacements predicted by '\
                'linear fit')
    legend(loc = 'best')

clf()
fitData('11_springData.txt')
```

Terms of Use

- This presentation was adapted from lecture materials provided in MIT Introduction to Computer Science and Programming in Python.
- Licenced under terms of Creative Commons License.