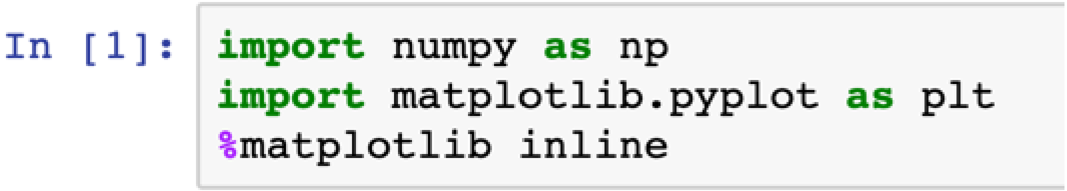
Getting Started with Matplotlib

Douglas Starnes

SOFTWARE ENGINEER / CONFERENCE SPEAKER / TECH AUTHOR

@poweredbyaltnet   http://douglasstarnes.com



```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

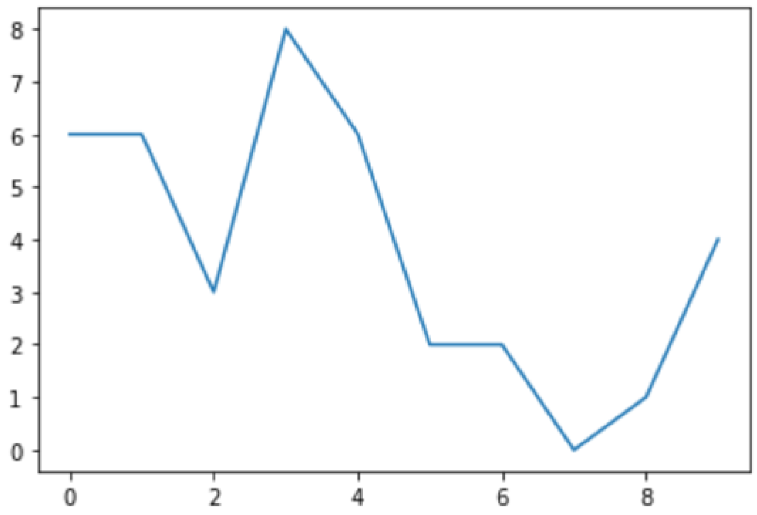
For most of the course assume that:

- Numpy and matplotlib have been imported
- Matplotlib.pyplot (aliased as plt) is where most of the functions we will use live
- The matplotlib magic command has been executed with the inline backend

Line Graph

Rendered with the plot() function

- Accepts at least a single iterable
- Values plotted along the y-axis
- Most of the time this course will use a list or numpy ndarray
- X-axis values default to consecutive integers, starting with 0



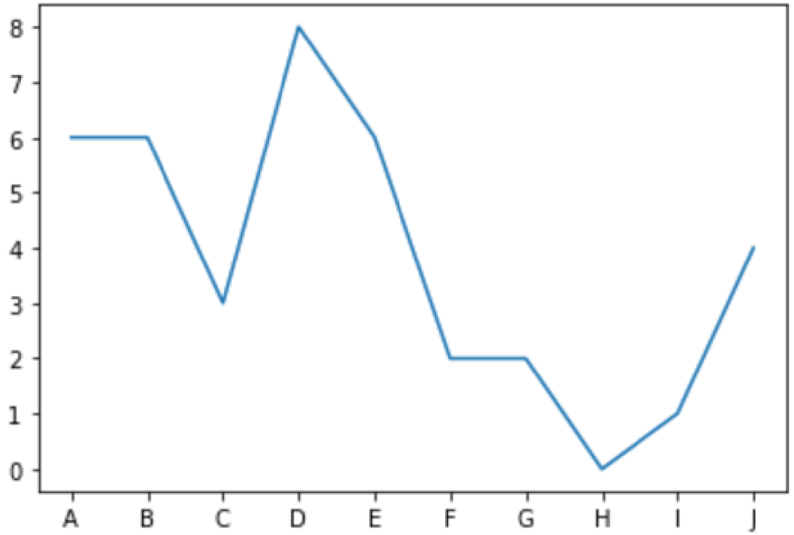
```
data = np.random.randint(0, 10, 10)
plt.plot(data)
```

Line Graph

Matplotlib will infer positions of the ticks

Get more control with the xticks() function

- Takes two iterables
- Locations of the ticks
- Values associated with the ticks



```
import string
plt.plot(data)
plt.xticks(list(range(len(data))),
    list(string.ascii_uppercase[:len(data)]))
```

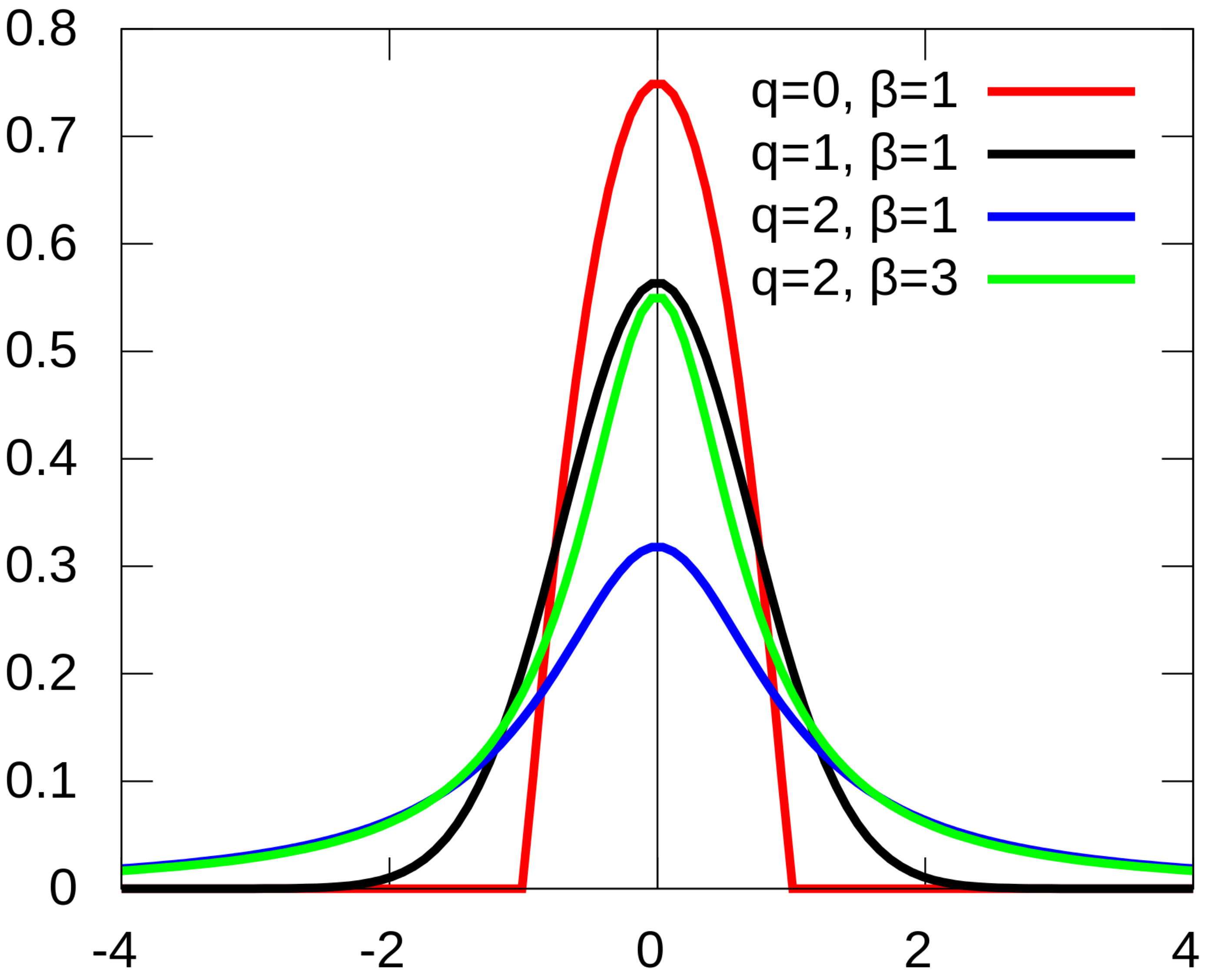
Histogram

Visualizes frequency distributions

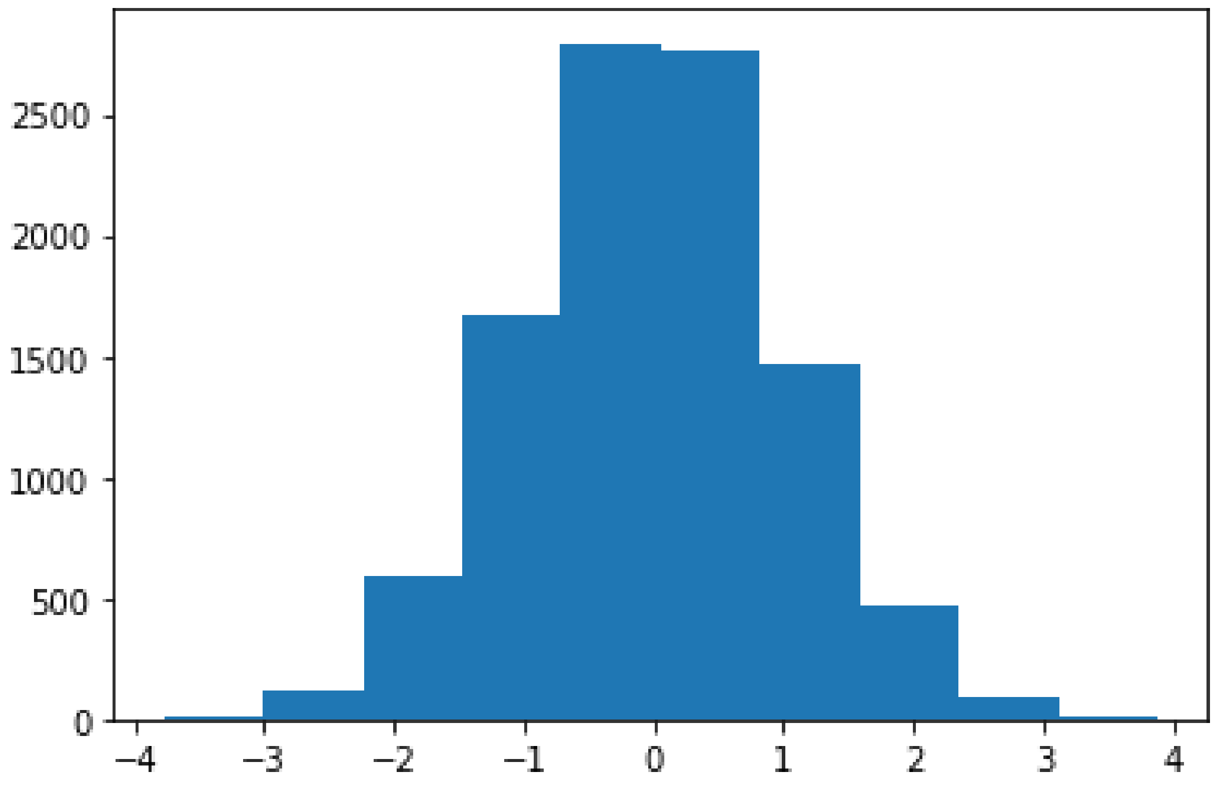
Generated with the hist() function

- Accepts a dataset
- Divides the dataset into evenly spaced intervals
- Matches each value in the dataset with an interval
- Counts the number of values in each interval

Normal Distribution

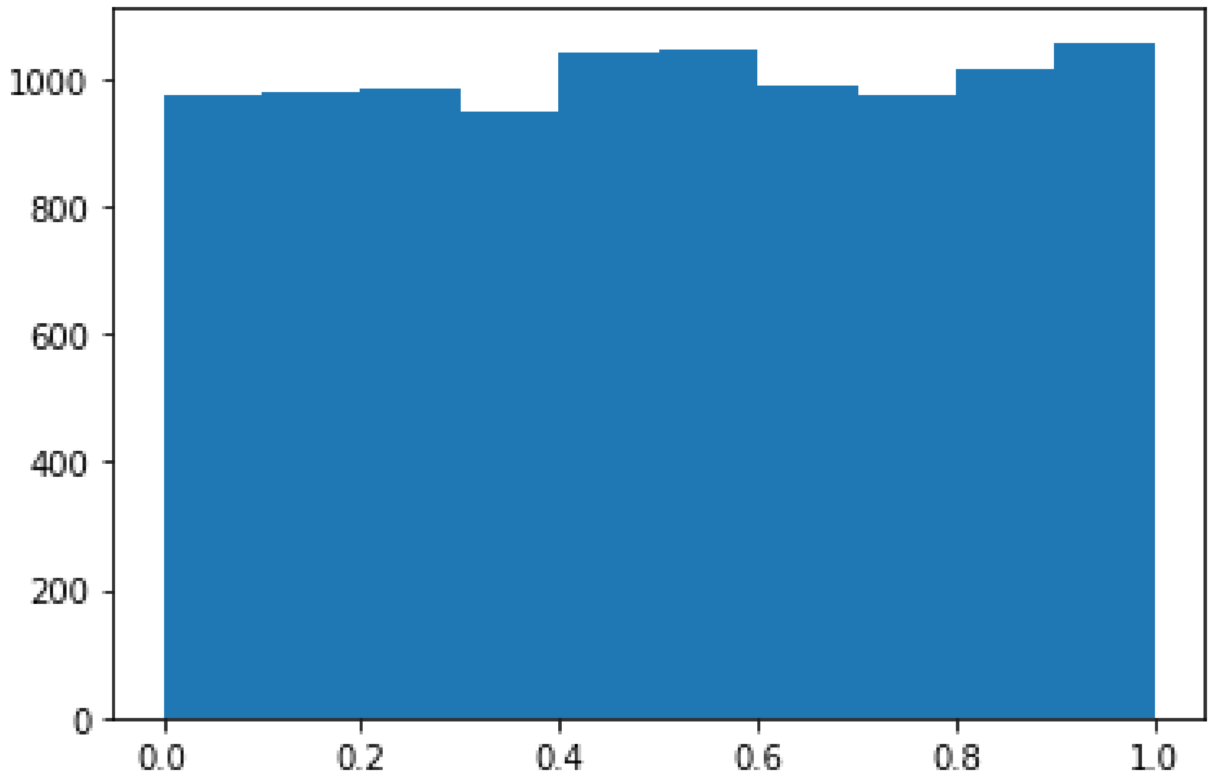


Histogram



```
data = np.random.randn(10000)
plt.hist(data)
```

Histogram: Uniform Distribution



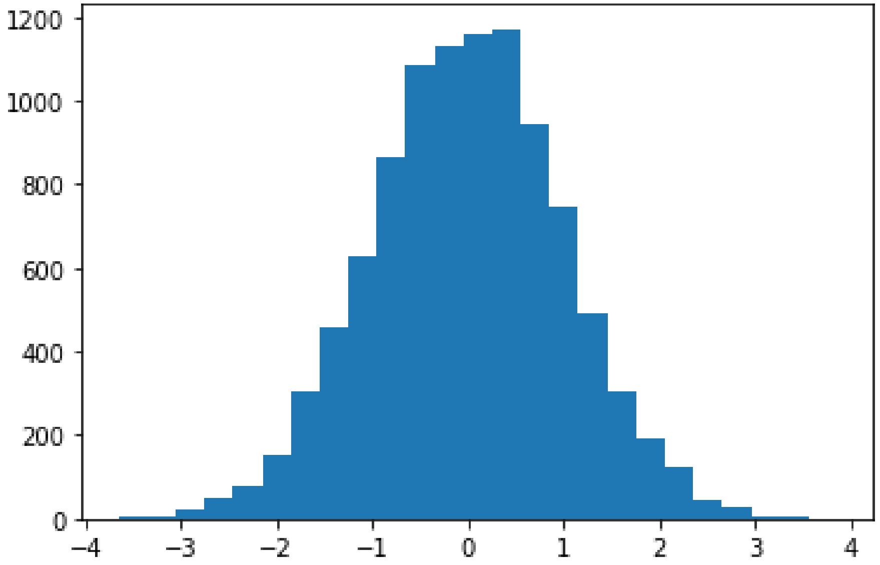
```
data = np.random.uniform(size=10000)
plt.hist(data)
```

Bins

By increasing the number of intervals, we can see more detail

The intervals are called bins

- bins= keyword argument
- Default is 10



```
plt.hist(data, bins=25)
```

2D Histogram

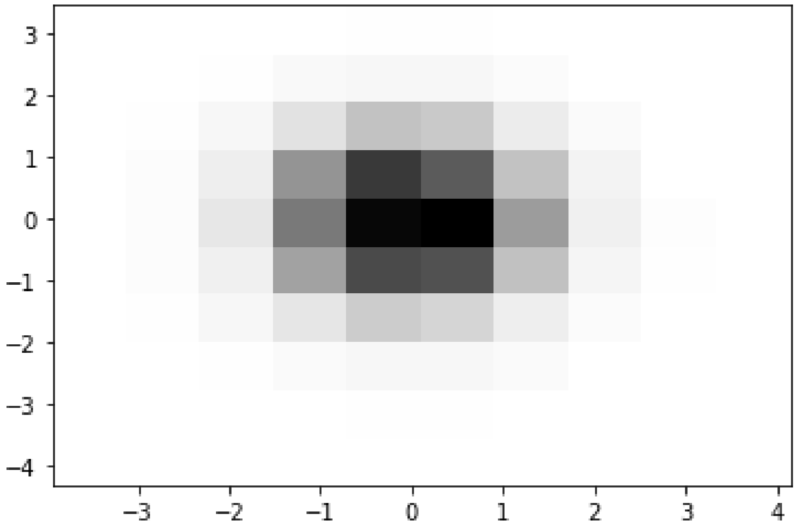
Generated with the function hist2d()

Creates a histogram for each of two datasets

- One becomes the x-axis, other becomes the y-axis

The perspective is top down

- Darker color shows higher values



```
plt.hist2d(np.random.randn(10000),
   np.random.randn(10000))
```

Bar Chart

Also called a column chart

Similar to a line chart

- Represents individual values with bars
- Taller bars represent greater values

A bar chart is not a histogram

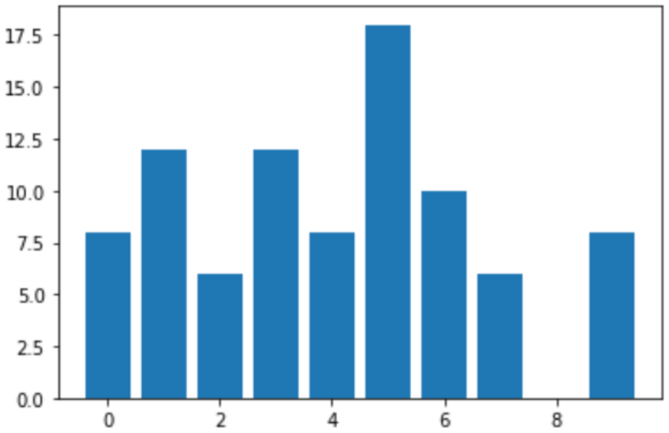
(and vice versa!)

Bar Chart API

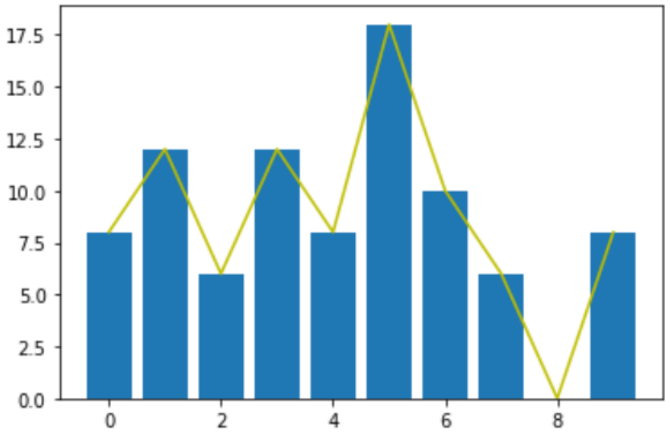
The bar() function

Accepts two iterables

- x-coordinates of the bars
- ‘heights’



```
data = np.random.randint(0, 20, 10)
plt.bar(np.arange(len(data)), data)
```



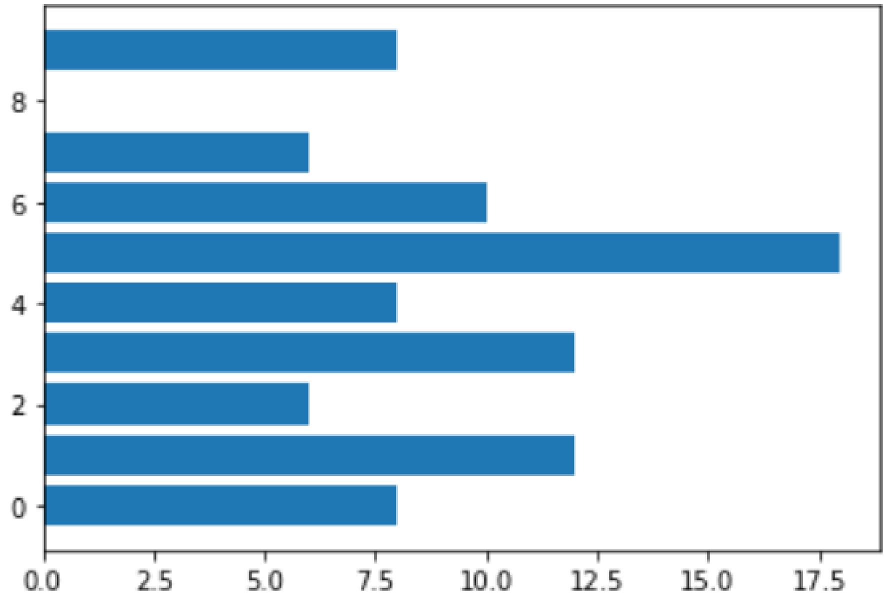
```
plt.bar(np.arange(len(data)), data)
plt.plot(data, c='y')
```

Bar Chart API

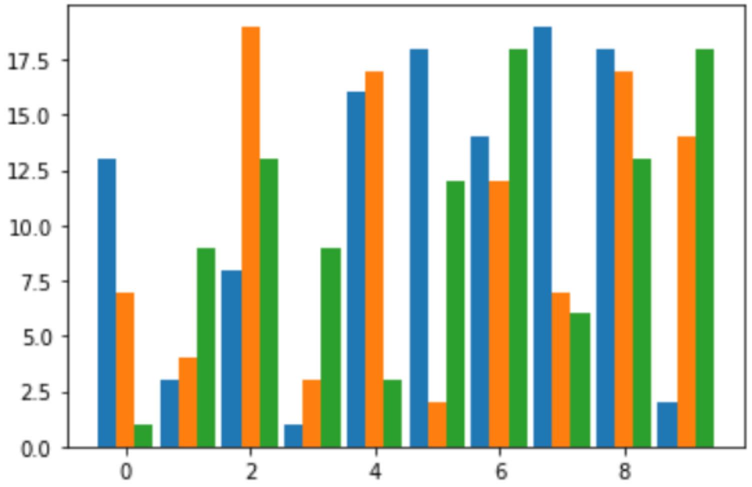
Horizontal bar chart

The barh() function

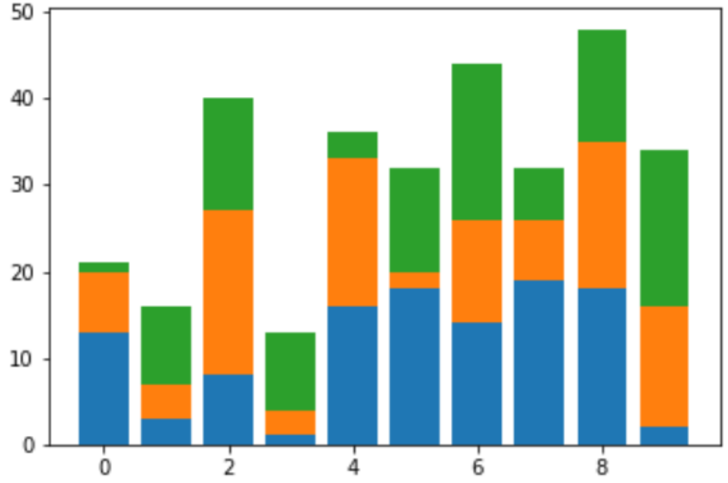
Length of the bar represents the magnitude of the values



Multiple Datasets



Grouped Bar Chart



Stacked Bar Chart

Pie Chart

Like the stacked bar chart, represents parts of a whole

Displays only a single dataset

Values are percentages

Will always add up to 100

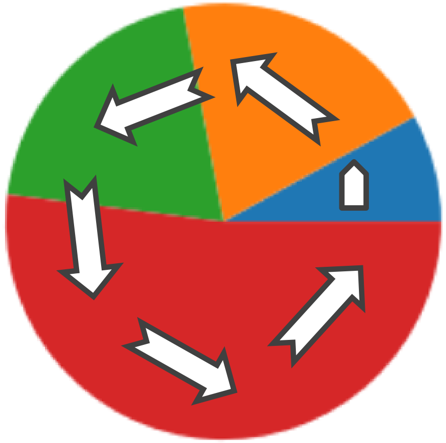
Pie Chart

The entire dataset is represented as a circle

The values are wedges

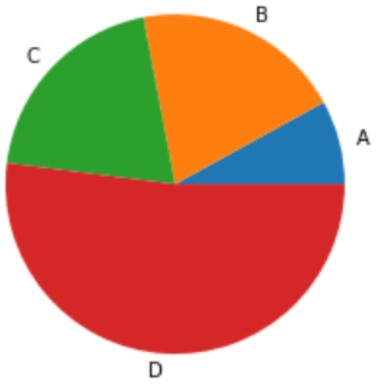
The size of each wedge is proportional to the total of the values

Each wedge/value is represented by a slice of the pie/dataset

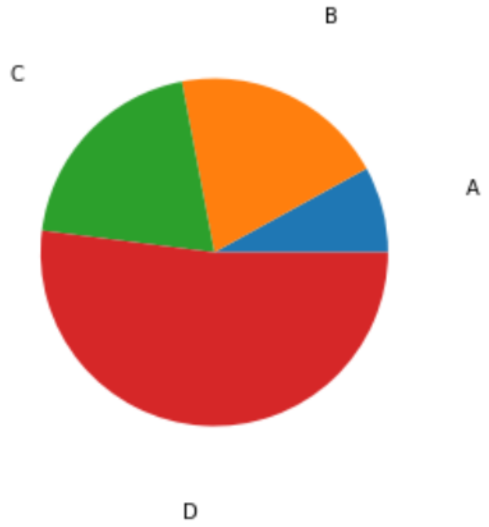


```
wedges = np.array([8, 20, 20, 52])
plt.pie(wedges)
```

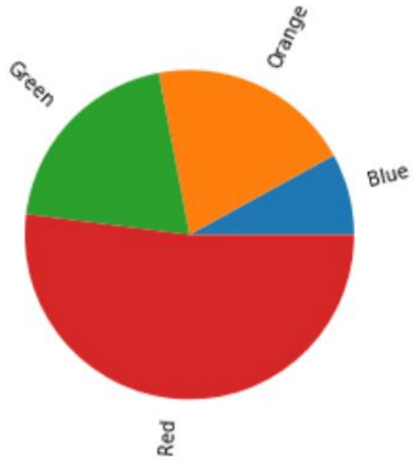
Pie Chart Options



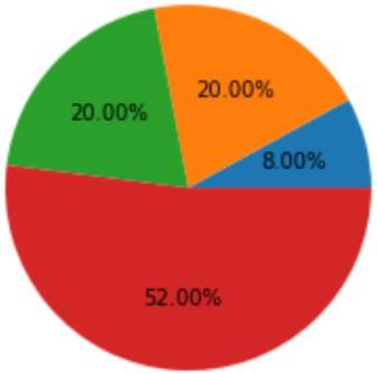
```
plt.pie(wedges, labels=list('ABCD'))
```



```
plt.pie(wedges, labels=list('ABCD’), labeldistance=1.5)
```



```
plt.pie(wedges,
    labels=[‘Blue’, ‘Orange’, ‘Green’, ‘Red’],
    rotatelabels=True)
```



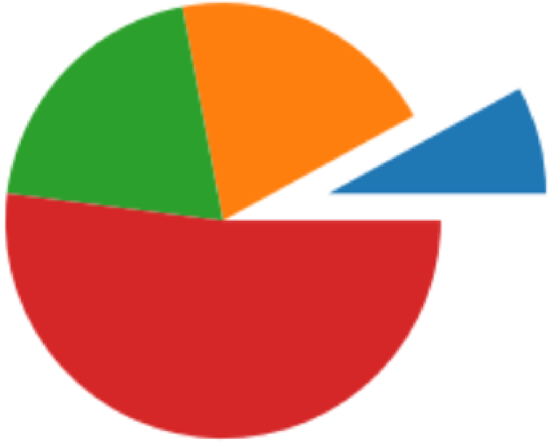
```
plt.pie(wedges,autopct=‘%.2f%%’)
```

Exploding a Wedge

Small wedges can be seen easily if they are offset from the rest of the pie

The explode= keyword argument is a list of values, one for each wedge

The values instruct matplotlib how much to offset the wedges, as a fraction of the radius



```
plt.pie(wedges, explode=[0.5, 0.0, 0.0, 0.0])
```

Scatter Plot

Displays individual points

Order is not significant

Scatter plots show relationships between the points

The scatter() function

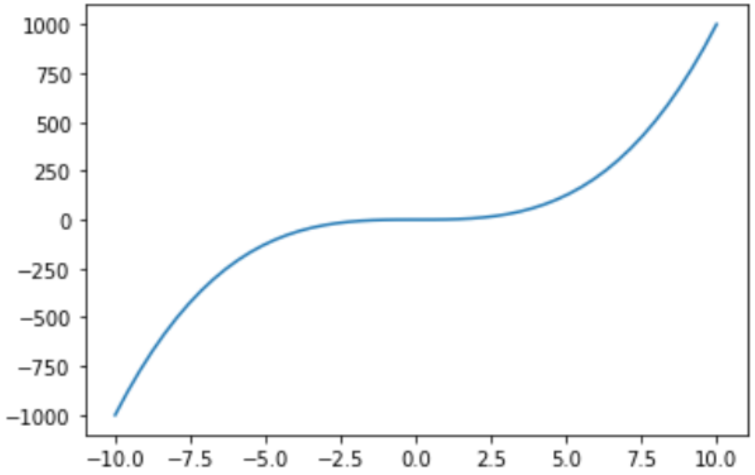
Accepts two iterables

- First is the x-axis
- Second is the y-axis

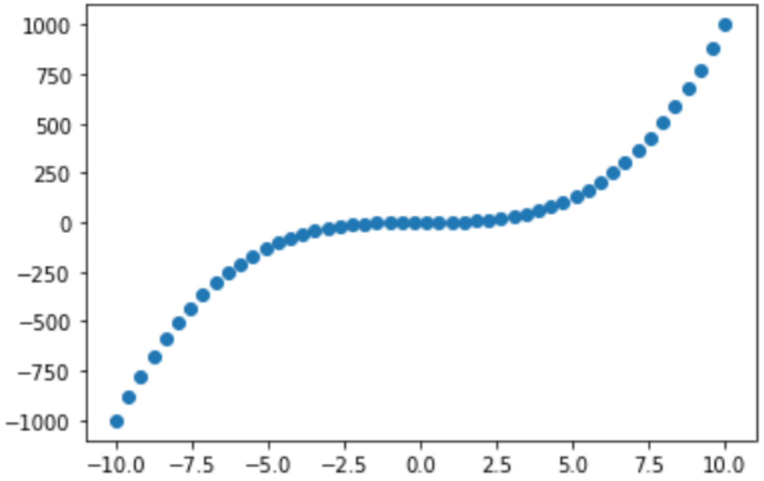
Scatter Plot

```
x = np.linspace(0, 10, 50)
y = x ** 3
```

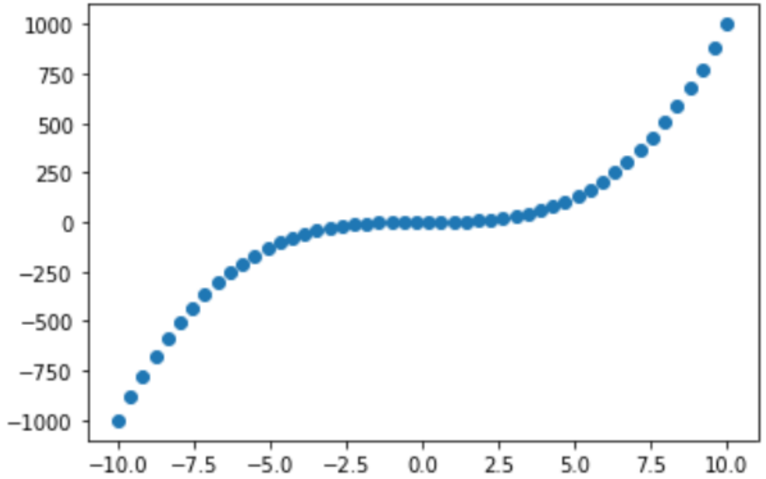
```
plt.plot(x, y)
```



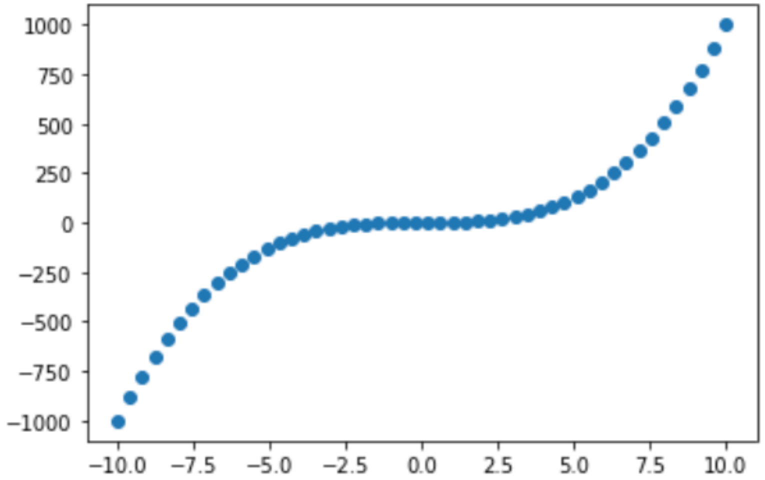
```
plt.scatter(x, y)
```



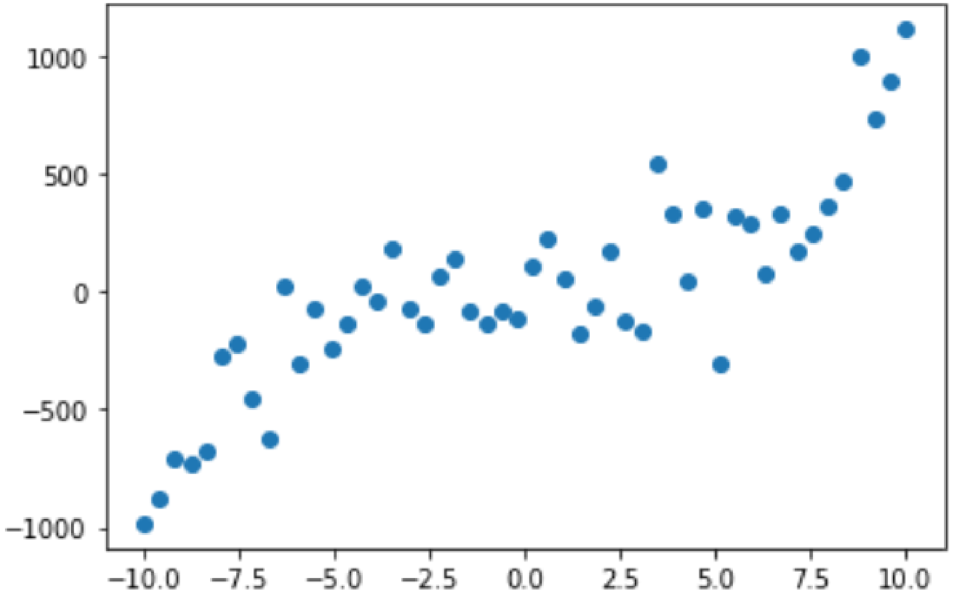
Order Is Insignificant: The Proof Is in the Plots

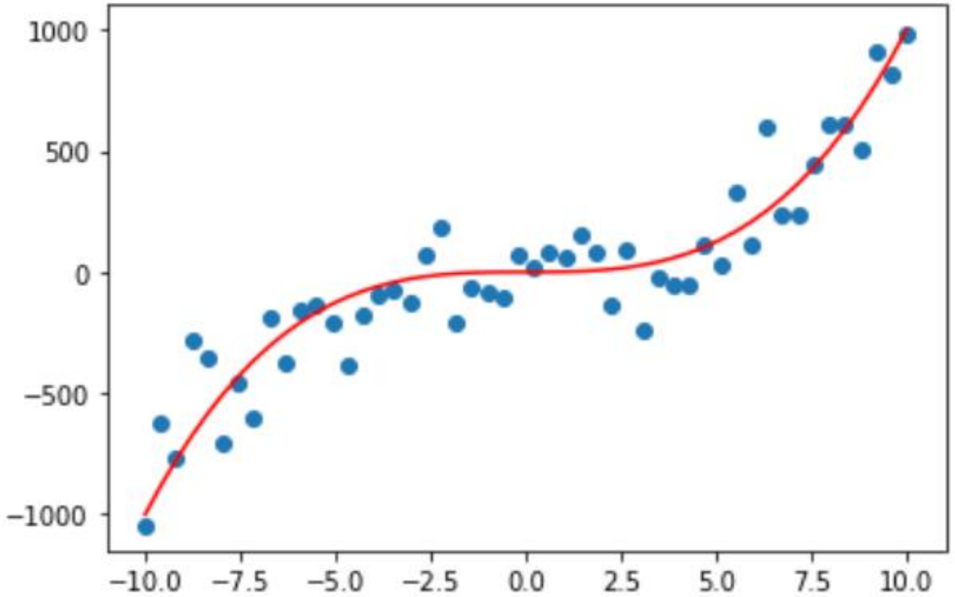


```
plt.scatter(x, y)
```

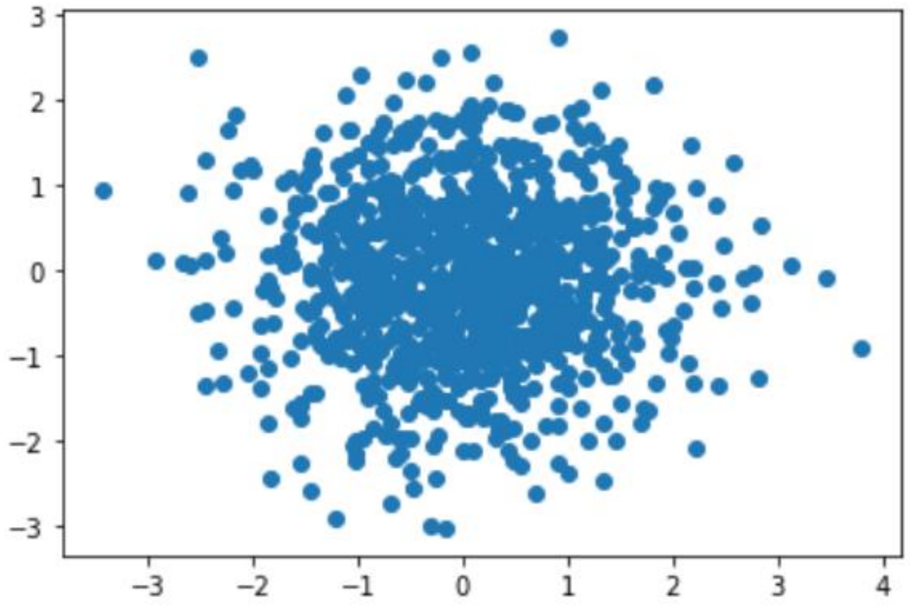


```
import random
points = list(zip(x, y))
random.shuffle(points)
points = np.array(points)
plt.scatter(points[:,0],
  points[:,1])
```

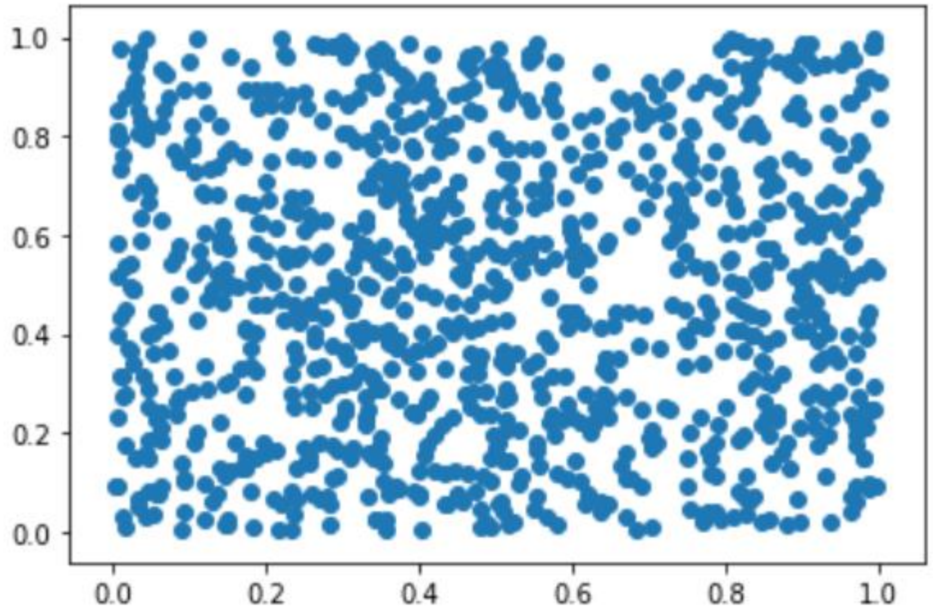

1000
500
0
−500
−1000
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


1000
500
0
−500
−1000
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Random Scatter Plots



```
plt.scatter(
    np.random.normal(size=1000),
    np.random.normal(size=1000))
```



```
plt.scatter(
    np.random.uniform(size=1000),
    np.random.uniform(size=1000))
```