NYC DATA SCIENCE ACADEMY

Introduction to Python: matplotlib

NYC Data Science Academy

OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Visualization in Python

- This course covers two parts: matplotlib and seaborn. This slide covers the package matplotlib.

OVERVIEW

- matplotlib overview
  - Basic plot
  - Statistical plot
    - Sample dataset
    - Scatterplot
    - Histogram
    - Barplot
    - Boxplot
  - Multiple figures
  - Save

matplotlib overview

- matplotlib is a python 2D plotting library built on the top of the basic Python language and Numpy.
- More about matplotlib can be found in its documentation.
- We will mainly focus on a sub-module of matplotlib called pyplot. 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, etc.
- More about pyplot can be found in its tutorial.

matplotlib overview

- This following magic function "%matplotlib" is used to enable the inline backend for usage with the IPython Notebook.

```
%matplotlib inline
```

- We then import pyplot and Numpy

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

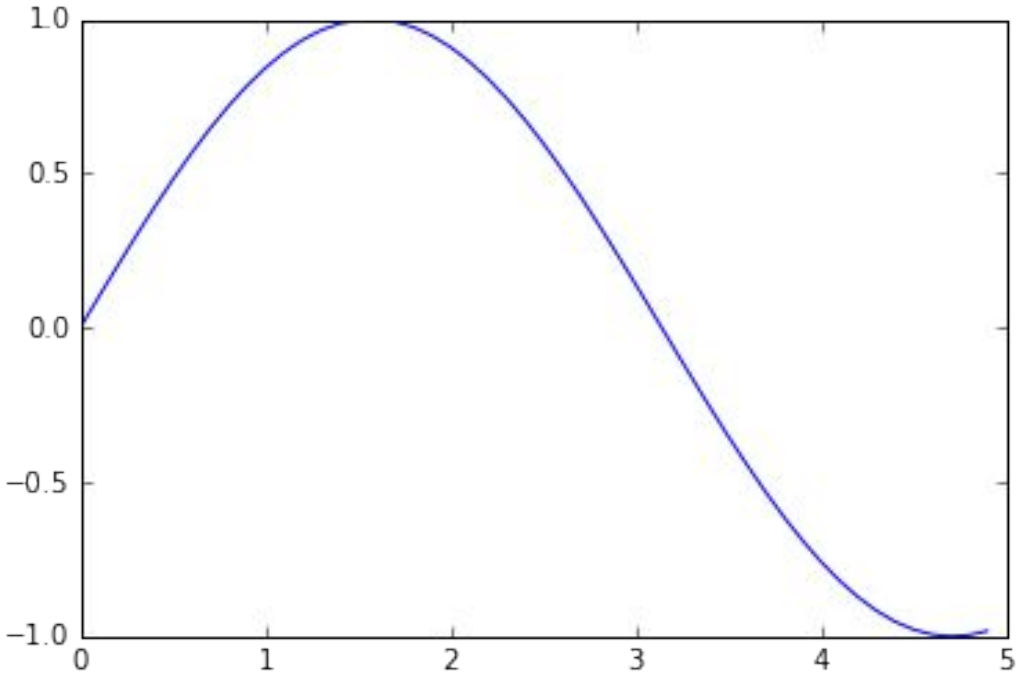
OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Basic plot

- The command "plot" draws a curve by connecting all the points

```
x = np.arange(0, 5, 0.1)
y = np.sin(x)
plt.plot(x, y)
```



Basic plot: line style and marker

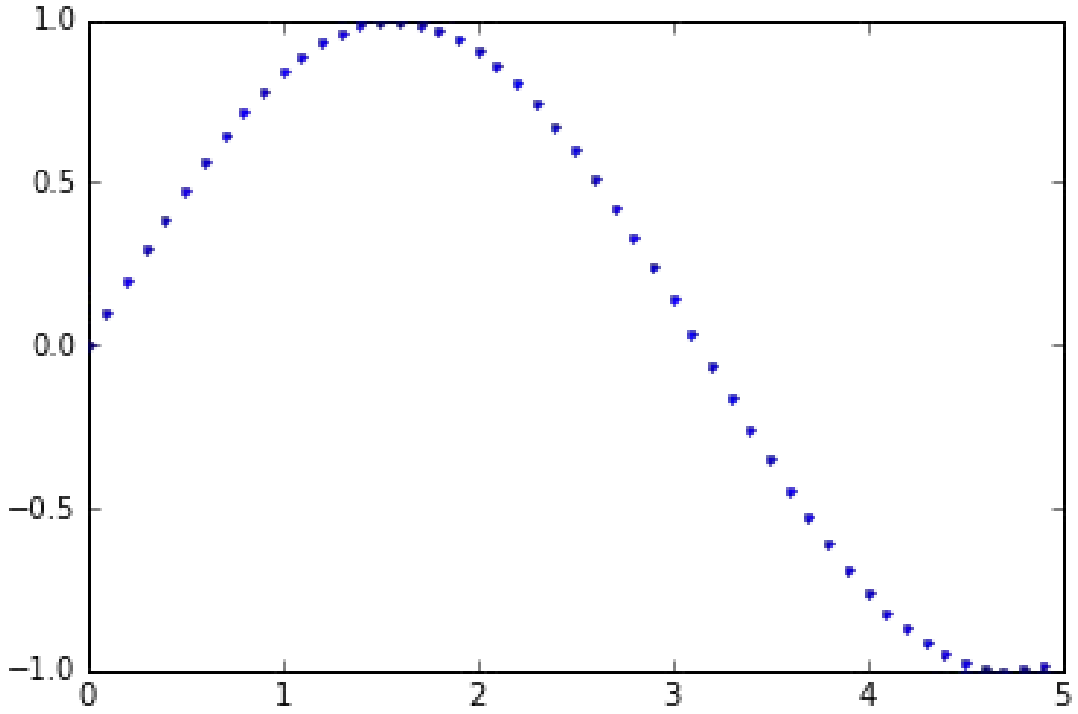
- If a scatter plot is desired, we may add an additional argument into the plot function to specify "shape".

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
...	...

Basic plot: line style and marker

- The command "plot" draws a curve by connecting all the points

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



Basic plot: color and legend

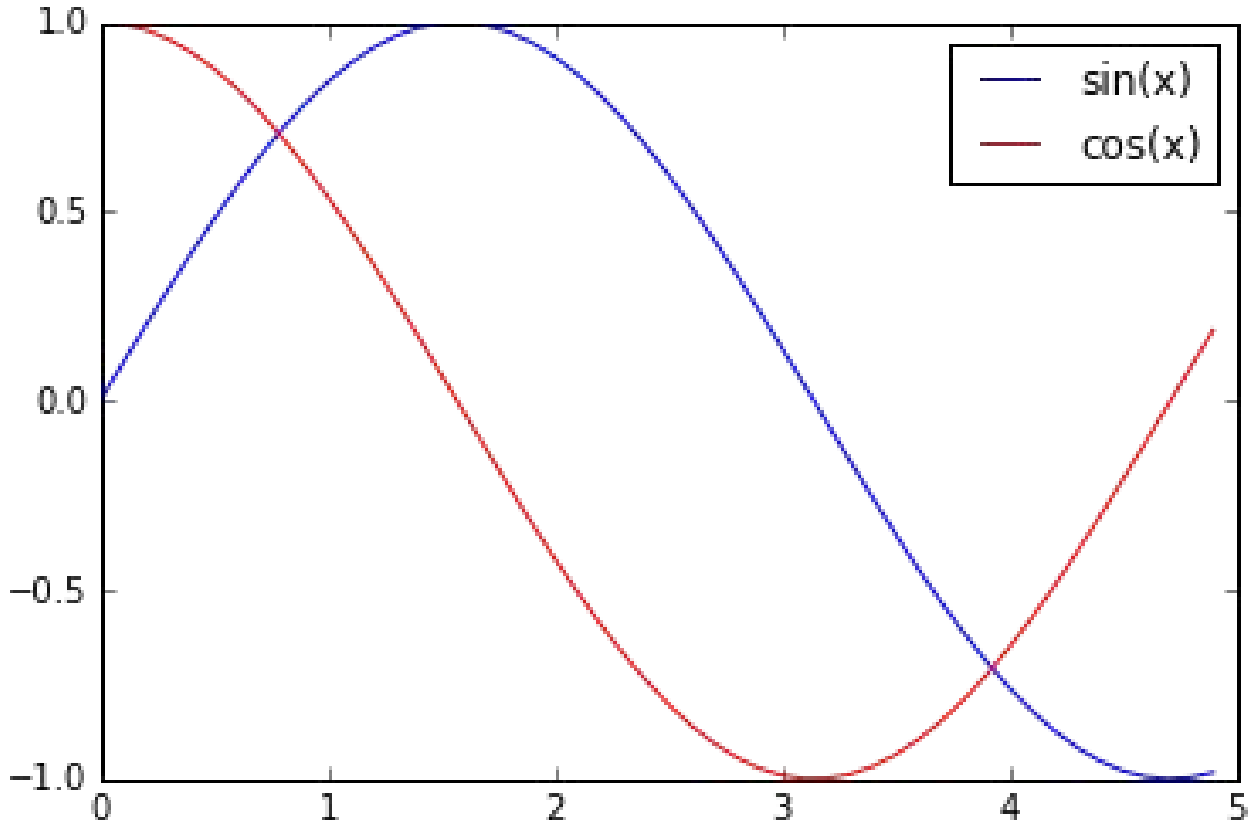
- We can actually specify color and the "name" of each curve with the argument below. x and y define the plot we saw before, x and y2 are paired to define another curve. We may assign different colors as well as names to them.

```
y2 = np.cos(x)
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)
```

- Legend can be generated with the `.legend()` method as above. The number indicates the position: 1 → topright, 2 → topleft, 3 → bottomleft, and 4 → bottomright.

Basic plot: line style and marker

- The plot generated is like:



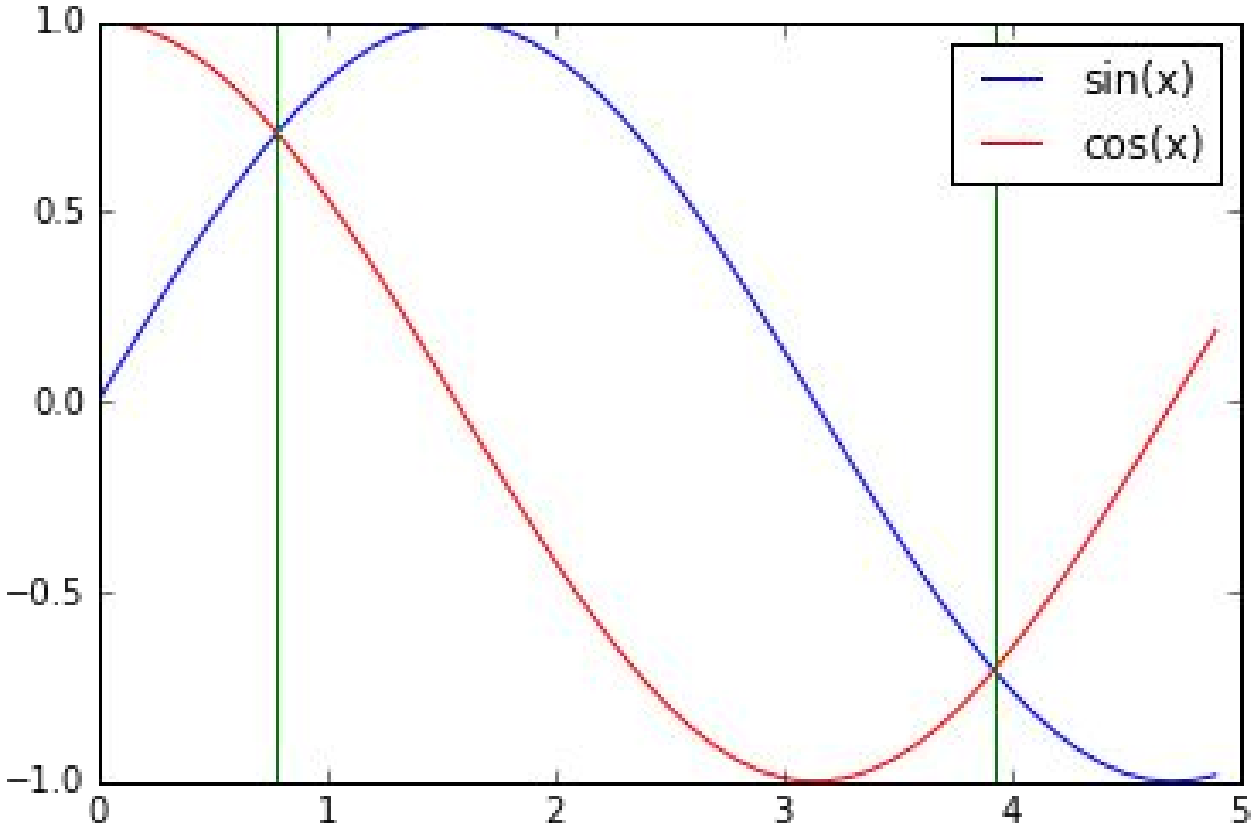
Basic plot: fill with color

- We may also fill a region bounded by curves. With the previous plot as an example, we need to decide the intersection of the curves (this can often be done by binary search which is out of the scope of this class). Let's assume somehow we figure the intersections of the red and blue curves are 0.7854044, 3.926992, we first equally spaced the interval between them(for later use) and then visualize their positions.

```
cut = np.linspace(0.7854044, 3.926992, 100)
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)

plt.plot([cut[0], cut[0]], [-1,1], color='green')
plt.plot([cut[-1], cut[-1]], [-1,1], color='green')
```

Basic plot: fill with color

- The plot generated is like:



Basic plot: fill with color

- With pyplot, in fact we can only fill a polygon. The code below specify a series of points along the blue and red curves from the previous plot.

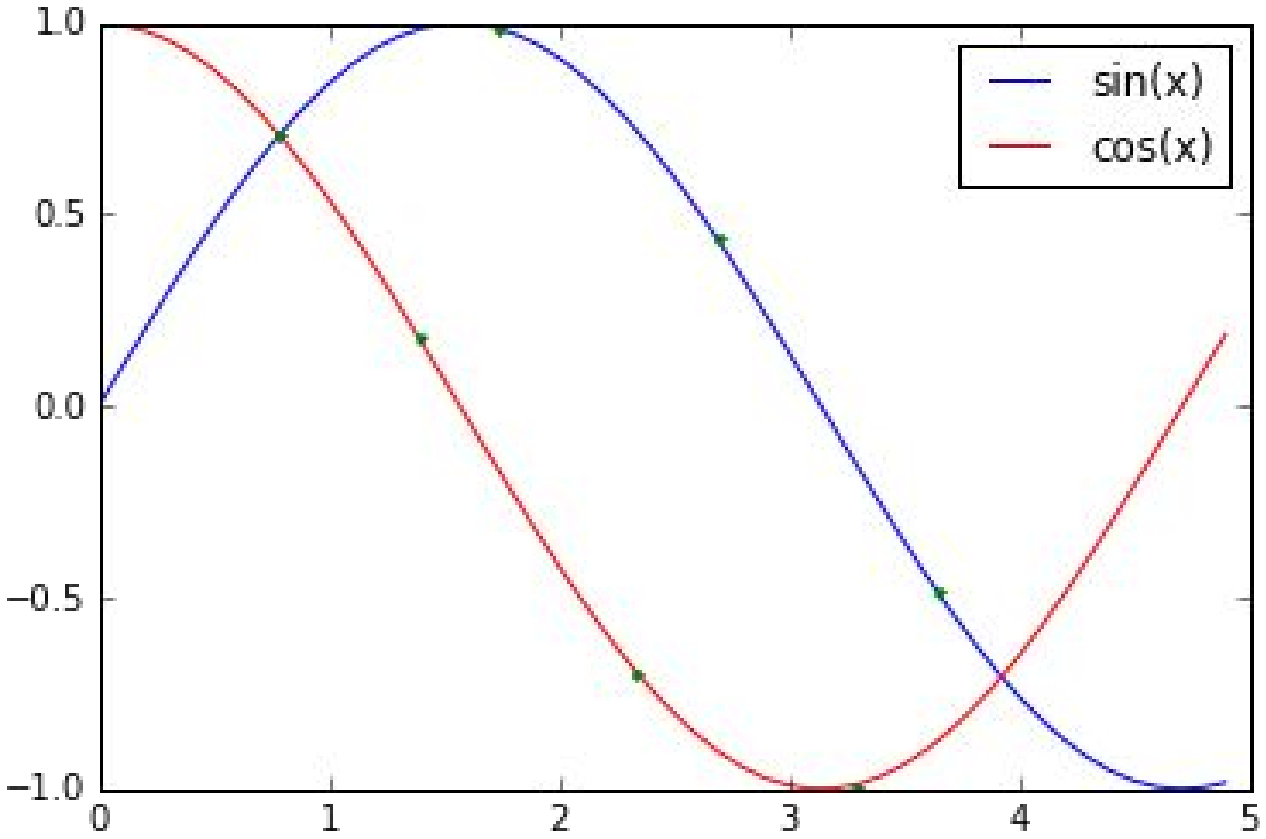
```
xx=np.concatenate([cut, cut[::-1]])
yy=np.concatenate([np.sin(cut),np.cos(cut[::-1])])
```

- Let's visualize some of these points:

```
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)

space=30
plt.plot(xx[::space], yy[::space], '.', color='green')
```

Basic plot: fill with color

- The plot generated is like:



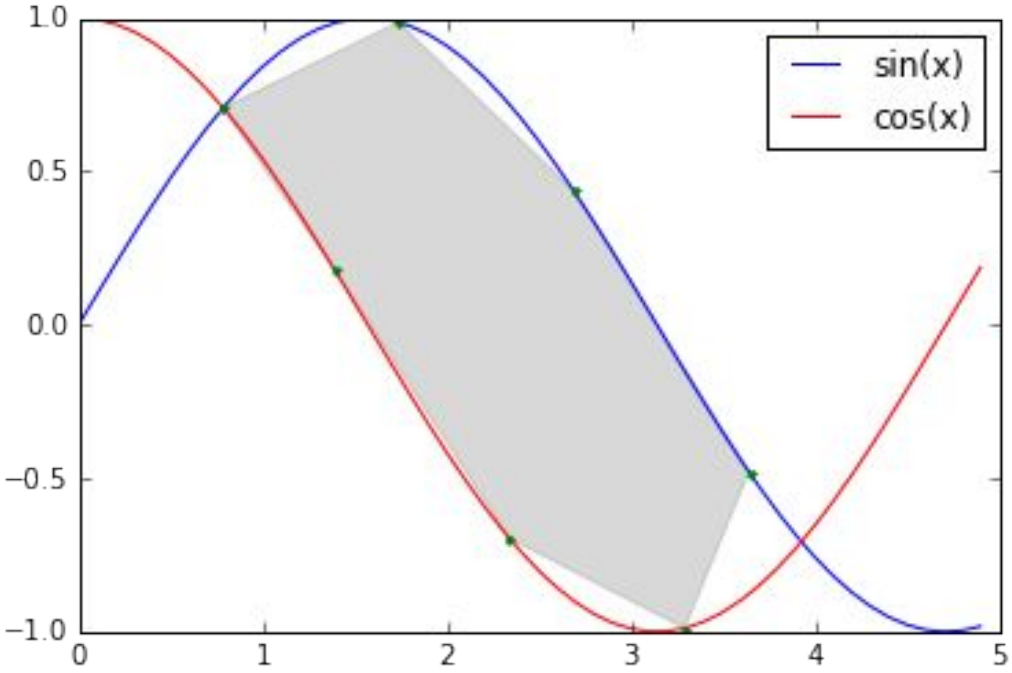
Basic plot: fill with color

- Filling works as below (alpha specify transparency):

```
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)

space=30
plt.plot(xx[::space], yy[::space], '.', color='green')
plt.fill(xx[::space], yy[::space], color='grey', alpha = 0.3)
```

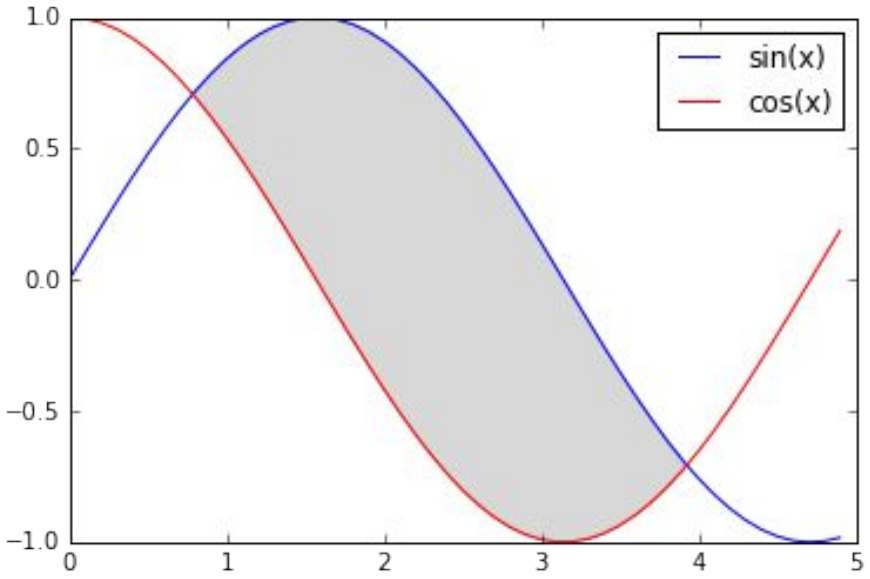
Basic plot: fill with color

- The plot obtained:



Exercise

- Modify the code above, to fill (or make it look like we fill) the region bounded by the blue and the red curves in the previous plot.



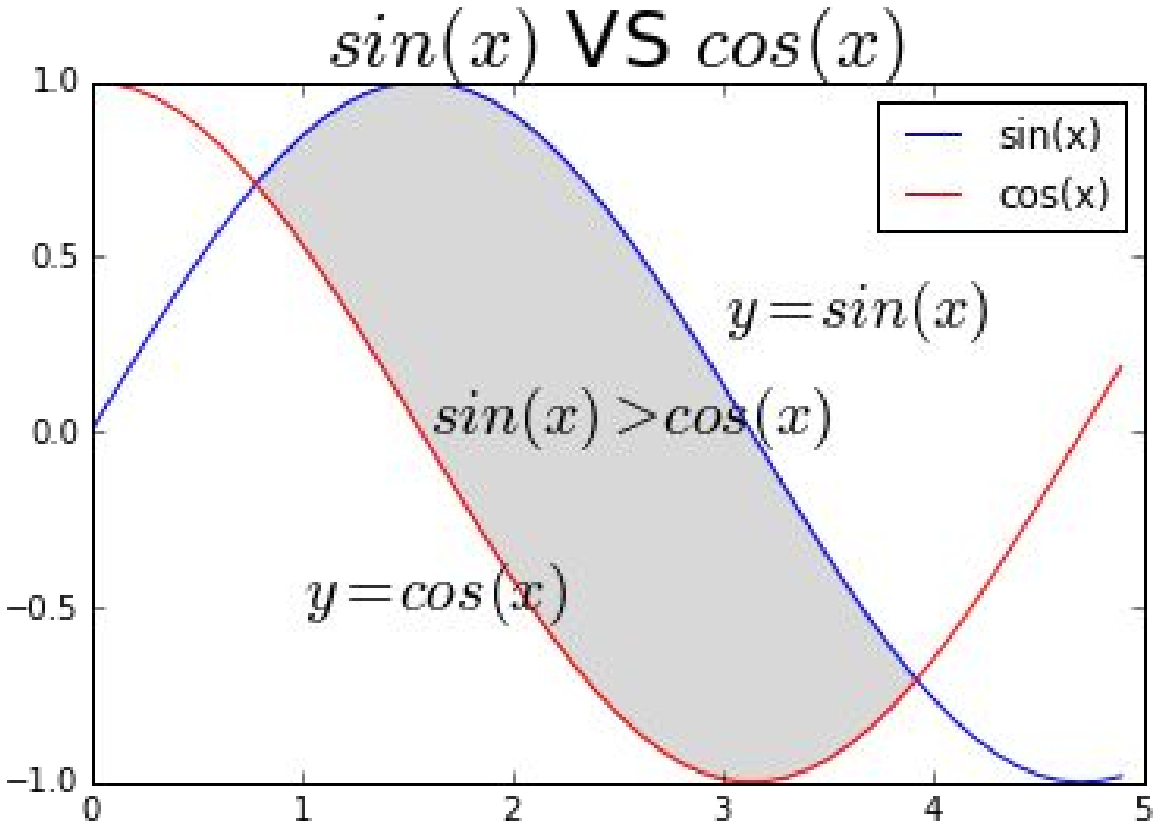
Basic plot: title and text

- **plt.title**: add a title
- **plt.text**: add text in the figure
- We may add mathematical expression in both text or title. The syntax is similar to LaTeX.

```
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)

plt.fill(xx, yy, color='grey', alpha = 0.3)
plt.title('$sin(x)$ VS $cos(x)$', fontsize = 25)
plt.text(1.6, 0, '$sin(x) > cos(x)$', fontsize = 20)
plt.text(1, -0.5, '$y = cos(x)$', fontsize = 20)
plt.text(3, 0.3, '$y = sin(x)$', fontsize = 20)
```

Basic plot: title and text

- The plot obtained:



Basic plot: axis

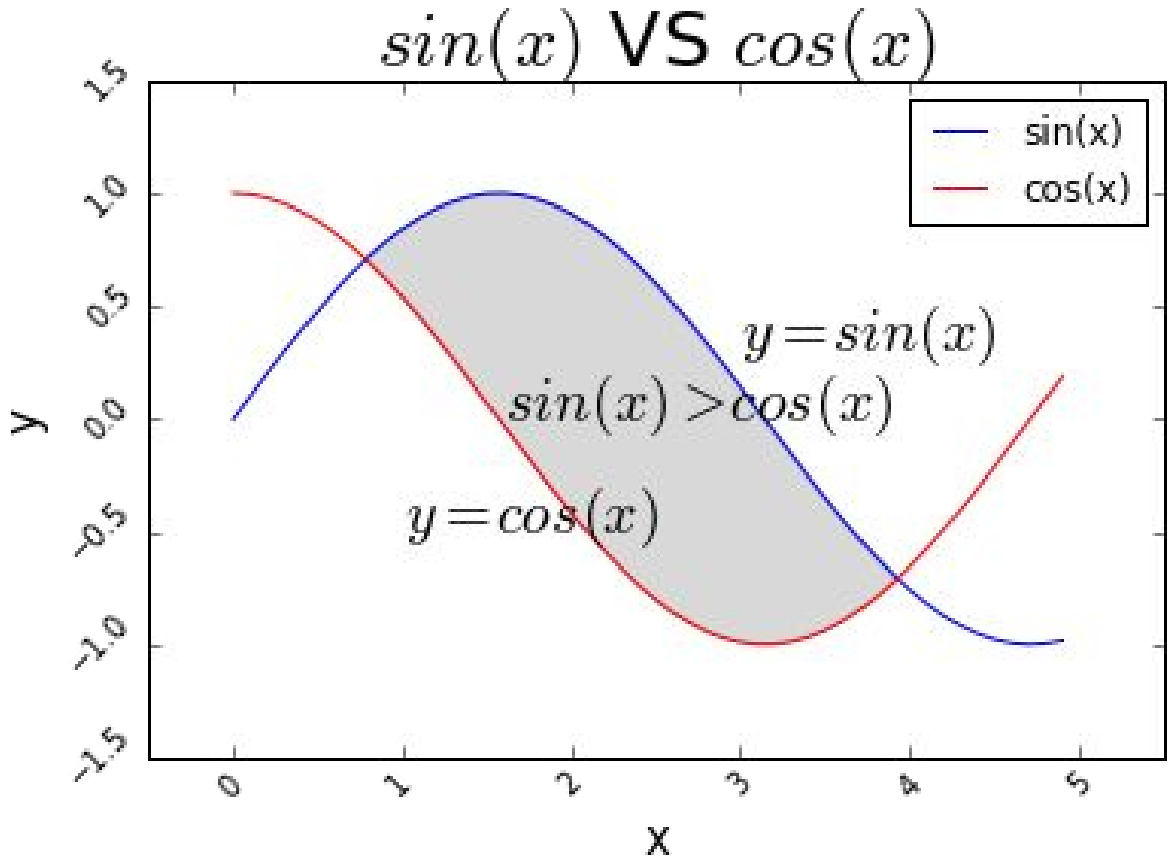
- The functions below are used to control the labels, ticks and ranges on the x and y axis respectively.
  - `plt.xlabel`: change the text of the label of the x axis
  - `plt.ylabel`: change the text of the label of the y axis
  - `plt.xticks`: change the text of the ticks of the x axis
  - `plt.yticks`: change the text of the ticks of the y axis
  - `plt.xlim`: change the range of the x axis to be plot
  - `plt.ylim`: change the range of the y axis to be plot

Basic plot: axis

```
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)
plt.fill(xx, yy, color='grey', alpha = 0.3)
plt.title('$sin(x)$ VS $cos(x)$', fontsize = 25)
plt.text(1.6, 0, '$sin(x) > cos(x)$', fontsize = 20)
plt.text(1, -0.5, '$y = cos(x)$', fontsize = 20)
plt.text(3, 0.3, '$y = sin(x)$', fontsize = 20)

plt.xlabel('x', fontsize = 15)
plt.ylabel('y', fontsize = 15)
plt.xticks(fontsize=10,rotation=45);
plt.yticks(fontsize=10,rotation=45);
plt.xlim(-0.5, 5.5); plt.ylim(-1.5, 1.5)
```

Basic plot: axis

- The plot obtained:



Exercise

- Play with the arguments we listed above. Get familiar with the basics.

OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Statistical plot

- We list some of the most commonly used statistical plot, including:
  - scatter: show the correlation of two variables.
  - histogram: show the distribution of a continuous variable.
  - barplot: show difference between factors.
  - boxplot: show the quantiles.

OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
    - Scatterplot
    - Histogram
    - Barplot
    - Boxplot
- Multiple figures
- Save

Statistical plot: sample dataset

- We will use the abalone dataset, which contains the physical measurement of abalones. the dataset was originally used to demonstrate a supervised learning -- predicting the number of rings (indicating the age of a abalone) with the other features. Run the command below in your iPython notebook and read about this dataset. Find out the feature names.

```
!cat abalone.names
```

Statistical plot: sample dataset

- Load the dataset into a Pandas data frame, and change the column names according to the features.

```
import pandas as pd
abalones = pd.read_csv('abalone.data', header = None)
abalones.columns = ['Sex','Length','Diameter','Height',\
                    'Whole weight','Shucked weight',\
                    'Viscera weight', 'Shell weight',\
                    'Rings']
```

- Take a look at the first five columns of your data frame.

```
abalones.head()
```

OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - **Scatterplot**
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Statistical plot: scatter plot

- We will plot the height of abalones against their weight.

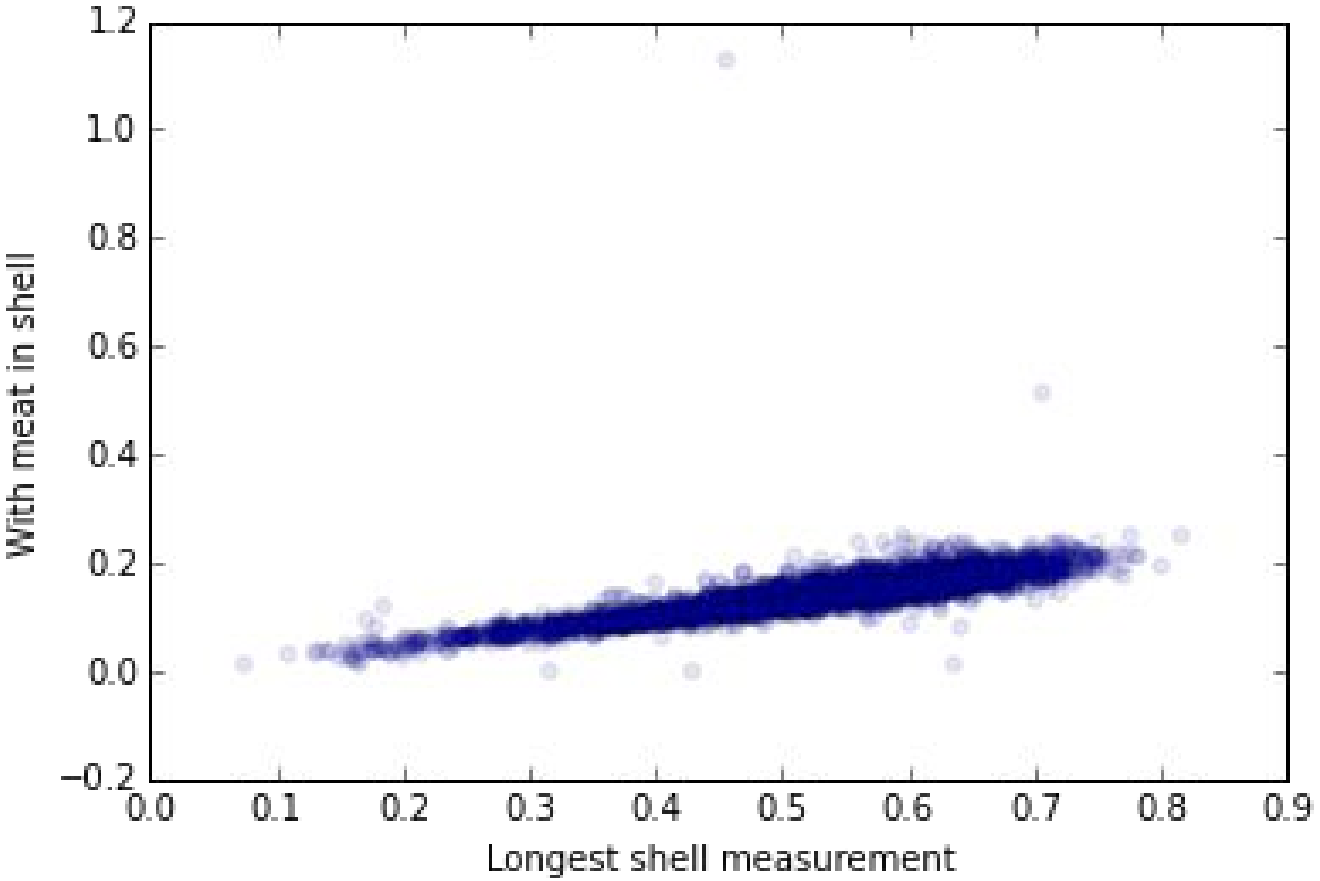
```
Height = abalones.Height
Length = abalones.Length
```

- Scatter plot can be sketched with the function: `.scatter()`

```
plt.scatter(Length, Height, alpha = 0.1)
# similar to plt.plot(Length, Height, 'o', alpha = 0.1)
plt.xlabel('Longest shell measurement')
plt.ylabel('With meat in shell')
```

Statistical plot: scatter plot

- The plot obtained:



OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

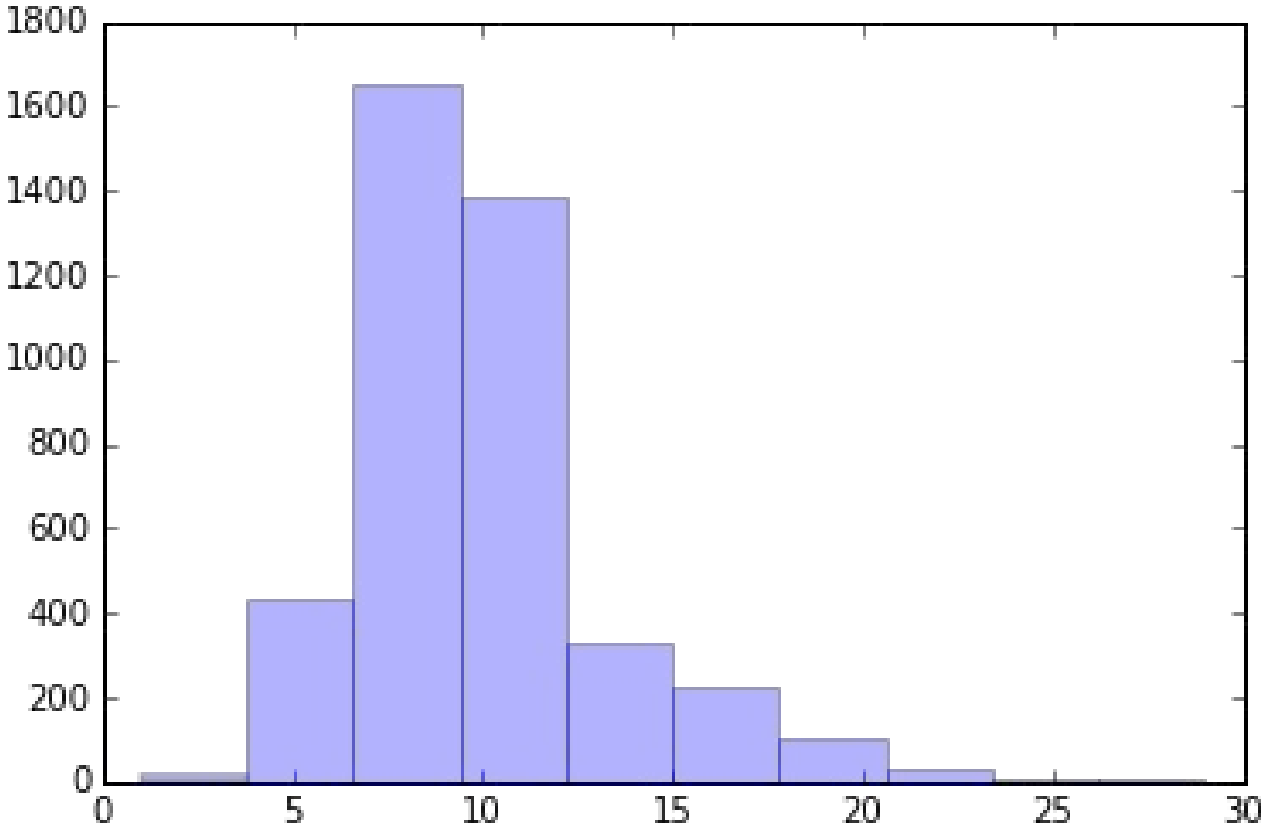
Statistical plot: histogram

- Histogram is often used to visualized the distribution of a numeric feature of a dataset. With pyplot, histogram can be constructed with the function `.hist()`. By default, `.hist()` cut the numeric feature into 10 subintervals, and count the number of observations falling into each subinterval. We demonstrate with the feature "Rings".

```
Ring = abalones.Rings
plt.hist(Ring, alpha = 0.3)
(array([ 17., 431., 1648., 1388., 329., 228., 100.,
         29.,   4.,    3.]),
 array([  1.,  3.8,   6.6,   9.4, 12.2,  15., 17.8,
        20.6,  23.4, 26.2,   29. ]),
 <a list of 10 Patch objects>)
```

Statistical plot: histogram

- The plot obtained:



Exercise

- There are two arrays of numbers returned by the `.hist()` function. Compare those numbers to the ticks in the histogram, what do you think those numbers represent?

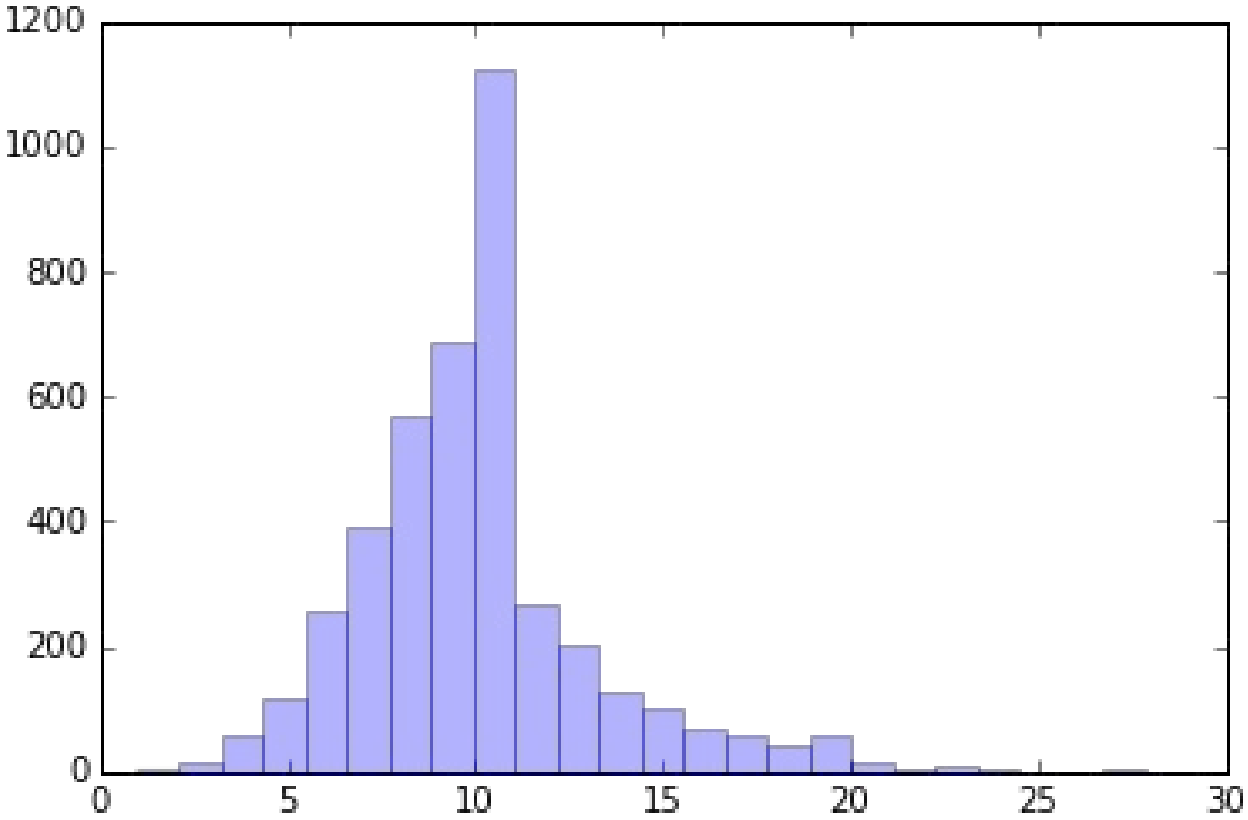
Statistical plot: histogram

- We may change the number of bins we want.

```
plt.hist(Ring, bins = 25, alpha = 0.3)
(array([  2.00000000e+00,   1.50000000e+01,   5.70000000e+01,
          1.15000000e+02,   2.59000000e+02,   3.91000000e+02,
          5.68000000e+02,   6.89000000e+02,   1.12100000e+03,
          2.67000000e+02,   2.03000000e+02,   1.26000000e+02,
          1.03000000e+02,   6.70000000e+01,   5.80000000e+01,
          4.20000000e+01,   5.80000000e+01,   1.40000000e+01,
          6.00000000e+00,   9.00000000e+00,   2.00000000e+00,
          1.00000000e+00,   1.00000000e+00,   2.00000000e+00,
          1.00000000e+00]),
 array([    1.,  2.12, 3.24,  4.36,  5.48,   6.6,  7.72, 8.84,
          9.96, 11.08, 12.2, 13.32, 14.44, 15.56, 16.68, 17.8,
         18.92, 20.04,21.16, 22.28,  23.4, 24.52, 25.64,26.76,
         27.88, 29.]),
 <a list of 25 Patch objects>)
```

Statistical plot: histogram

- The plot obtained:



OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - **Barplot**
  - Boxplot
- Multiple figures
- Save

Statistical plot: barplot

- Barplot is often used to visualize the amount of each class in a categorical feature. With pyplot, we need to count the number manually.

```
def counter(raw_list):
    '''
    raw_slist: a list
    returns: a dict contains each word and it's
    corresponding frequency
    '''
    result = {}
    for word in raw_list:
        result[word] = result.get(word, 0) + 1
    return result
```

Exercise

- We will demonstrate barplot on the sex feature on the abalone dataset. The code below helps to get the count of each class.

```
sexCount=counter(abalones.Sex)
sexCount=sorted(sexCount.items(), key=lambda x: x[1])
```

- Run the code above, track the type of sexCount. Does the type change at some point? If so, why did it change?

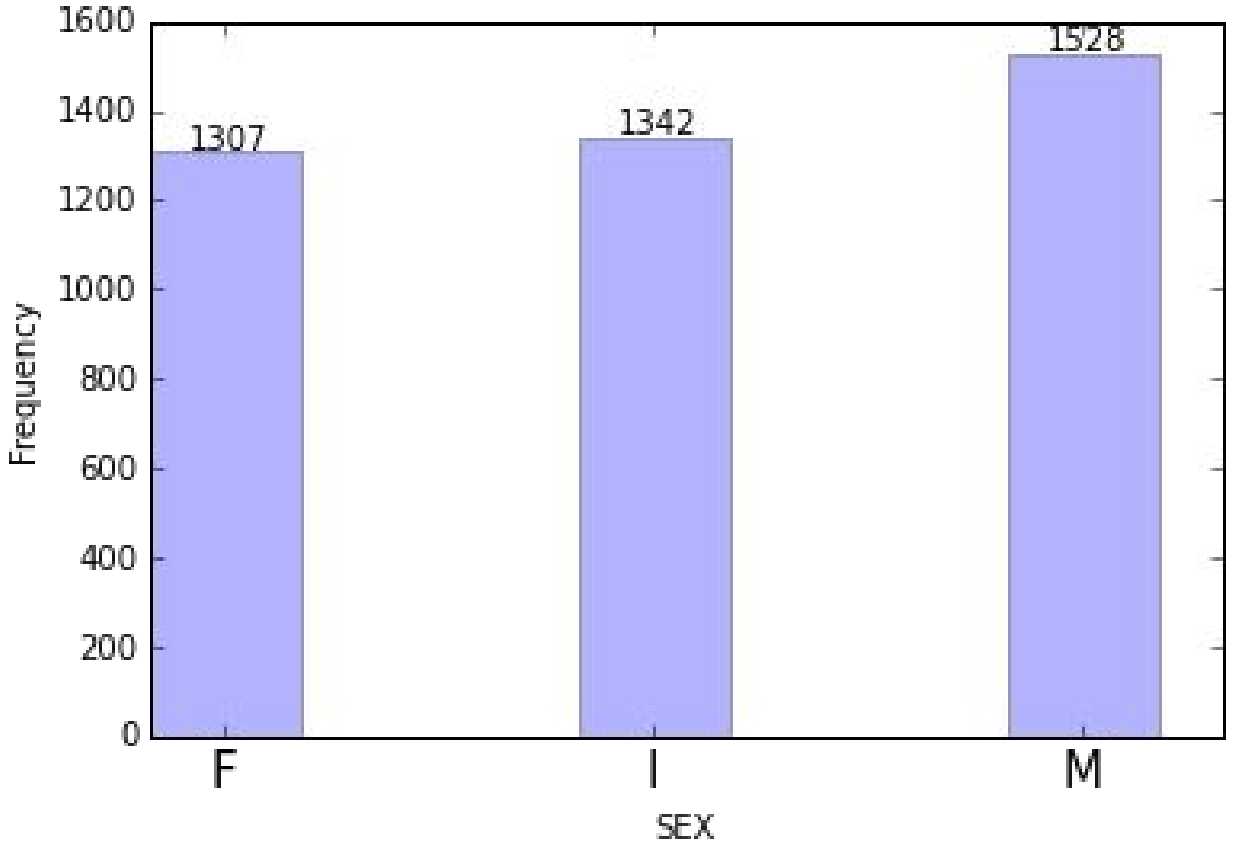
Statistical plot: barplot

- The code below sketch a barplot:

```
keys = [i[0] for i in sexCount]
values = [i[1] for i in sexCount]
plt.bar(np.arange(len(keys)), values, width = 0.35,\
                                      alpha = 0.3)
plt.xticks(np.arange(3) + 0.35/2, keys, fontsize = 15)
plt.xlabel('SEX')
plt.ylabel('Frequency')
for i in range(len(values)):
    plt.text(i + 0.1, values[i] + 10, values[i])
```

Statistical plot: barplot

- The plot obtained:



OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - **Boxplot**
- Multiple figures
- Save

Statistical plot: boxplot

- Boxplot is another way to visualize the distribution of a numeric feature. Let Q1, Q2 and Q3 represent the 25%, 50% and 75% quantile, respectively.
- Boxplot is made of five quantiles: Q1−1.5(Q3−Q1), Q1, Q2, Q3, and Q3+1.5(Q3−Q1). It can be made by function boxplot.
- Because of the shape of the boxplot, the function plt.rcParams['figure.figsize'] is used to change the width and height of plots.

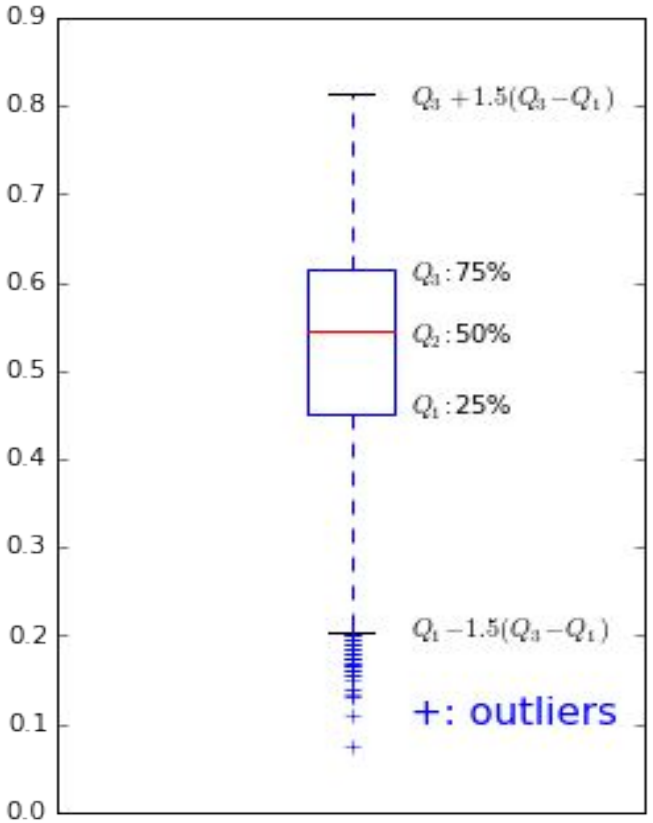
Statistical plot: boxplot

- The code below sketch a boxplot:

```
plt.rcParams['figure.figsize'] = 4, 6
plt.boxplot(Length)
plt.text(1.1, 0.2, '$Q_1 - 1.5(Q_3-Q_1)$')
plt.text(1.1, 0.45, '$Q_1$:25%')
plt.text(1.1, 0.53, '$Q_2$:50%')
plt.text(1.1, 0.60, '$Q_3$:75%')
plt.text(1.1, 0.80, '$Q_1 + 1.5(Q_3-Q_1)$')
plt.text(1.1, 0.1, '+: outliers', color = 'blue',
fontsize=15)
plt.xticks([])
```

Statistical plot: boxplot

- The plot obtained:



OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Statistical plot: multiple figures

- In many cases, we would like to arrange multiple plot in different ways.

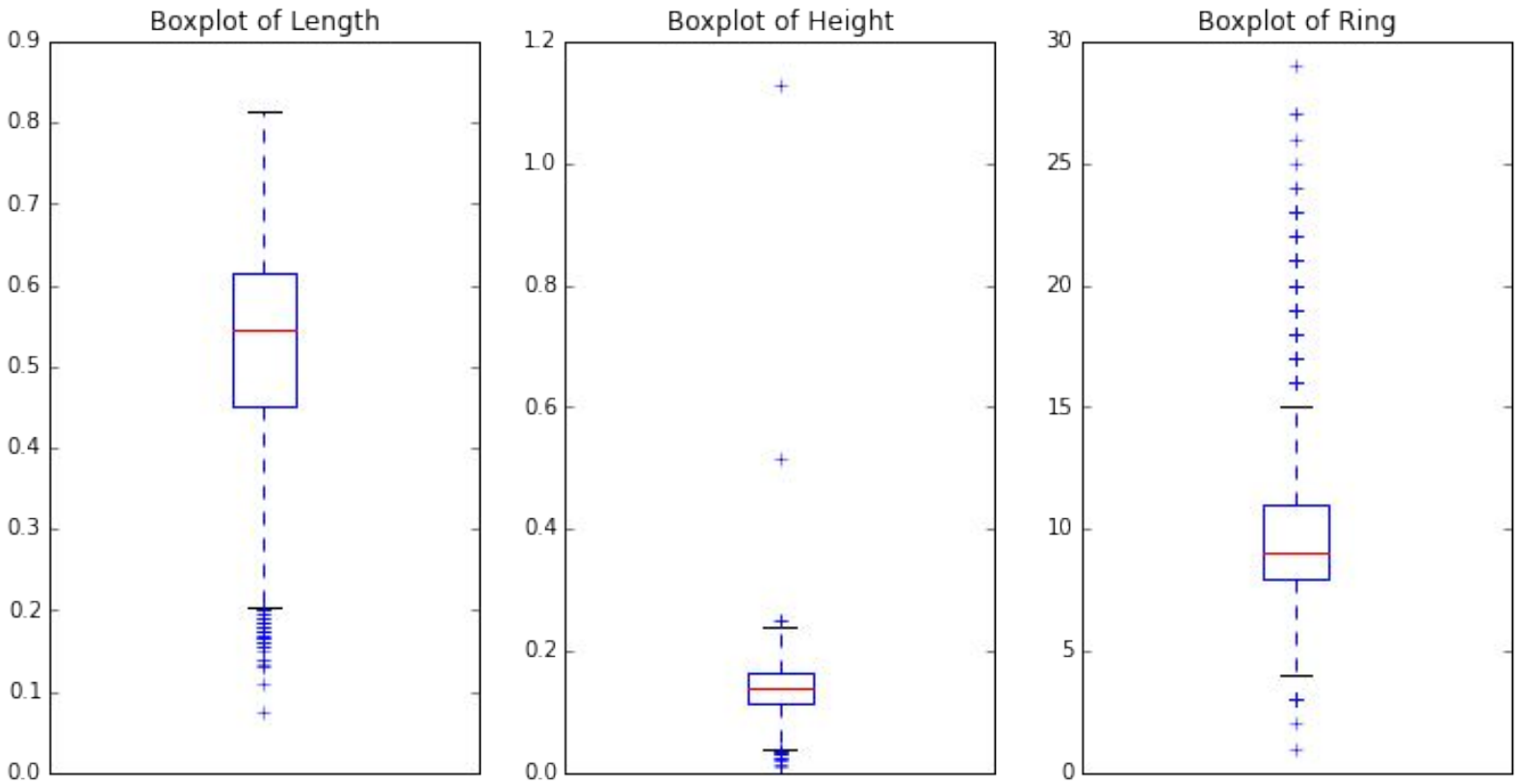
```
plt.rcParams['figure.figsize'] = 12, 6
# the first subplot
plt.subplot(131); plt.boxplot(Length)
plt.xticks([]); plt.title('Boxplot of Length')

# the second subplot
plt.subplot(132); plt.boxplot(Height)
plt.xticks([]); plt.title('Boxplot of Height')

# the third subplot
plt.subplot(133); plt.boxplot(Ring)
plt.xticks([]); plt.title('Boxplot of Ring')
```

Statistical plot: multiple figures

- Within the `subplot()` function, the first number (it is 1 from the example above) indicates the number of rows we want; the second number (it is 3 from the example above) indicates the number of columns we want; the third number (it is 1 from the example above) indicates the particular subplot to be filled in. So the code above results in the plot below:

Statistical plot: multiple figures

- The plot obtained:



Exercise

- For each sex of abalones, sketch the boxplot for their length.

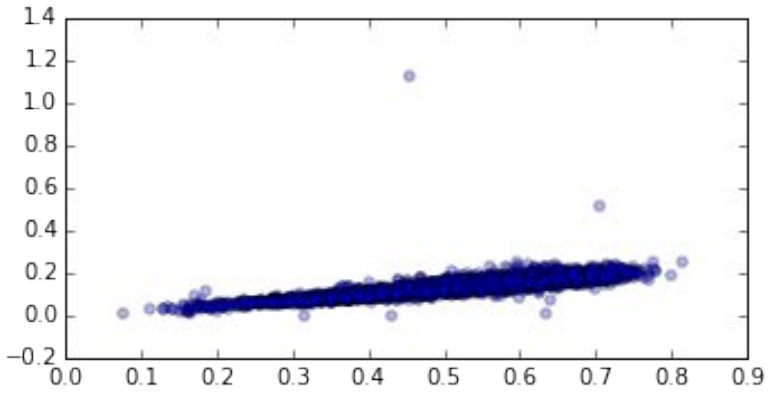
Statistical plot: multiple figures

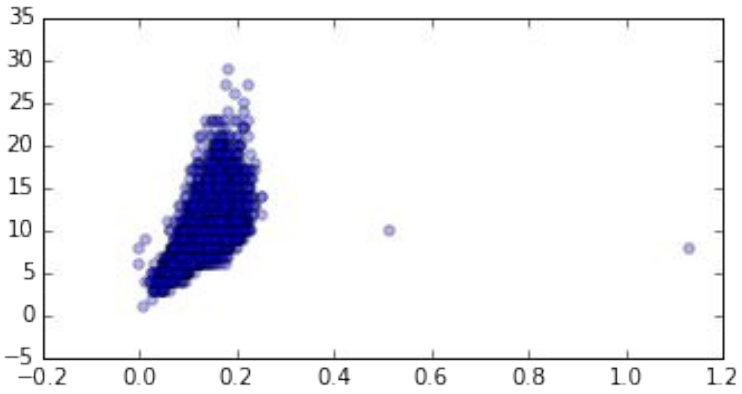
- Multiple columns are also possible:

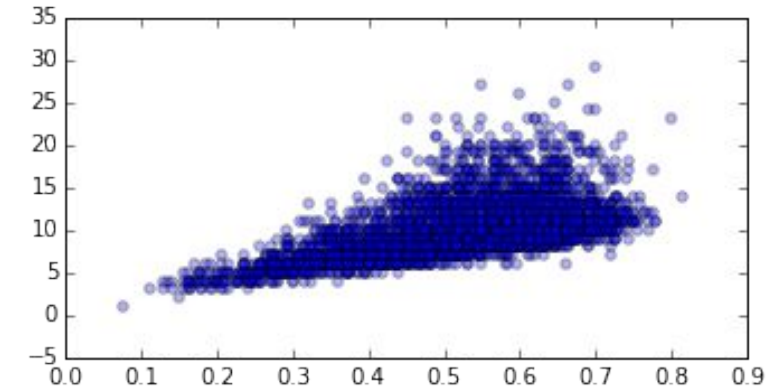
```
plt.rcParams['figure.figsize'] = 12, 6

# the first subplot
plt.subplot(221)
plt.scatter(Length, Height,alpha = 0.3)

# the second subplot (top-right)
plt.subplot(222)
plt.scatter(Height, Ring, alpha = 0.3)

# the third subplot (bottom-left)
plt.subplot(223)
plt.scatter(Length, Ring, alpha = 0.3)
```

Statistical plot: multiple figures

- The plot obtained:







Statistical plot: subplot2grid

- `subplot2grid()` is a more flexible function, which can manipulate the location as well as the shape of a plot.
- For example: plt.subplot2grid((3,3), (0,0), colspan=3)
  - (3, 3): split into 3 rows and 3 columns
  - (0, 0): the plot at 1th row and 1th column
  - colspan = 3: span 3 columns

Statistical plot: subplot2grid

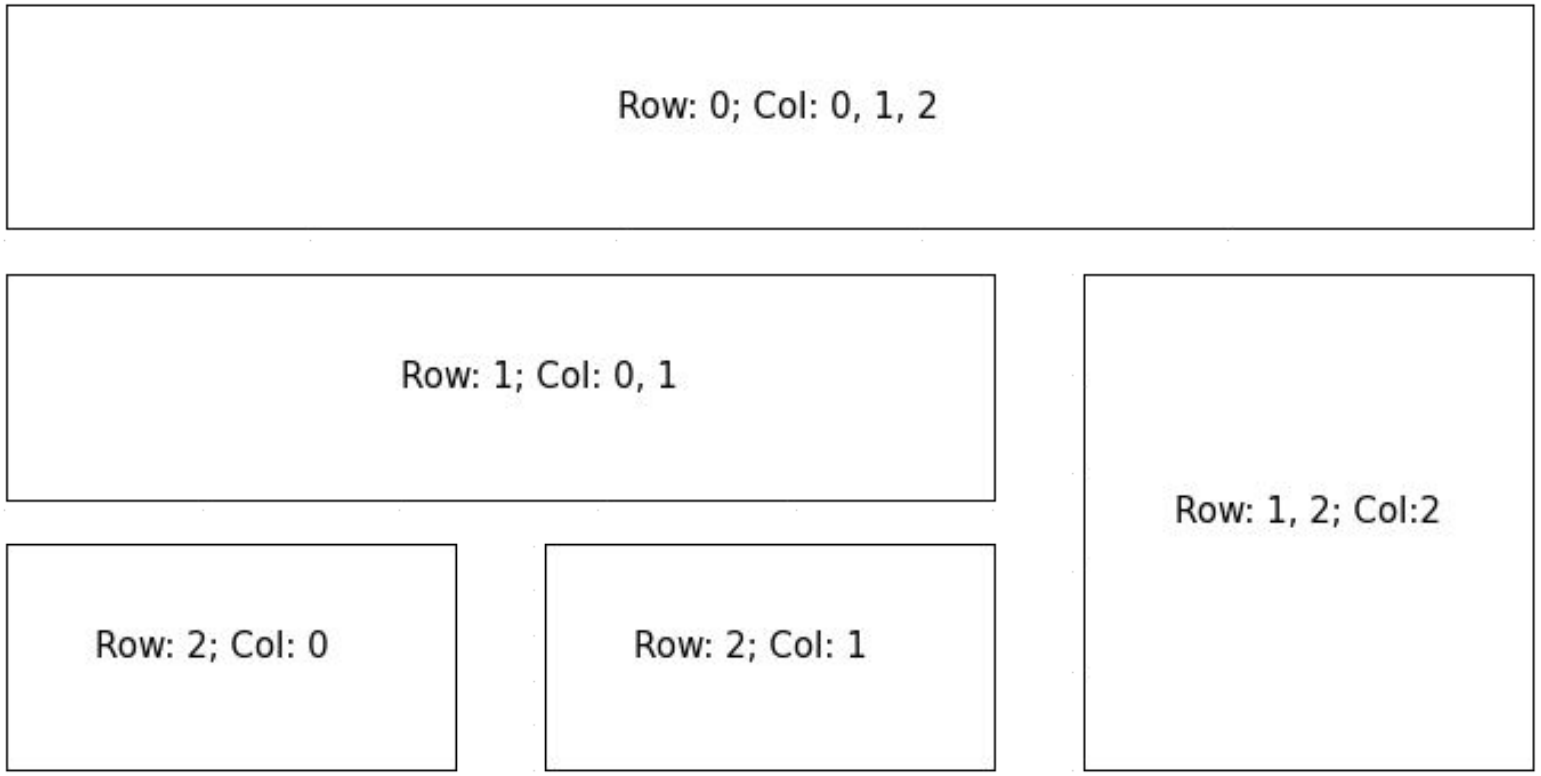
```
plt.rcParams['figure.figsize'] = 12, 6
# plots
plt.subplot2grid((3,3), (0,0), colspan=3)
plt.text(0.4, 0.5, 'Row: 0; Col: 0, 1, 2')

plt.subplot2grid((3,3), (1,0), colspan=2)
plt.text(0.4, 0.5, 'Row: 1; Col: 0, 1')

plt.subplot2grid((3,3), (1, 2), rowspan=2)
plt.text(0.2, 0.5, 'Row: 1, 2; Col:2')
```

Statistical plot: subplot2grid

```
plt.subplot2grid((3,3), (2, 0))
plt.text(0.2, 0.5, 'Row: 2; Col: 0')

plt.subplot2grid((3,3), (2, 1))
plt.text(0.2, 0.5, 'Row: 2; Col: 1')
```

Statistical plot: subplot2grid

- The plot obtained:



OVERVIEW

- matplotlib overview
- Basic plot
- Statistical plot
  - Sample dataset
  - Scatterplot
  - Histogram
  - Barplot
  - Boxplot
- Multiple figures
- Save

Statistical plot: subplot2grid

- Use the function savefig to save the picture.

```
plt.rcParams['figure.figsize'] = 12, 6
x = np.linspace(-5, 5, 100)
y = np.sin(x); y2 = np.cos(x)
plt.plot(x, y, color = 'blue', label='sin(x)')
plt.plot(x, y2, color = 'red', label='cos(x)')
plt.legend(loc=1)
### fill polygon
intersection = np.linspace(0.7854044, 3.926992, 100)
plt.fill(np.concatenate([intersection, intersection[::-1]]),\
         np.concatenate([np.sin(intersection),\
         np.cos(intersection[::-1])]),color='grey',alpha = 0.3)
### save
plt.savefig('matplot.png')
```