Python NumPy, Scipy, Sympy

Panchatcharam M

NumPy

- NumPy is the core library for scientific computing in python
- High-performance multi-dimensional array object and tools for working with these arrays
- Familiar with MATLAB??

- A NumPy array is a grid of values, all of the same type
- indexed by a tuple of nonnegative integers
- number of dimensions is the rank of the array
- the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4
```

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"
c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77
```

```
# Two ways of accessing the data in the middle row of the array.
# Mixing integer indexing with slices yields an array of lower rank,
# while using only slices yields an array of the same rank as the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

```
import numpy as np
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a)  # prints "array([[ 1,  2,  3],
          #                [ 4,  5,  6],
          #                [ 7,  8,  9],
          #                [10, 11, 12]])"
# Create an array of indices
b = np.array([0, 2, 0, 1])
# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print(a)  # prints "array([[11,  2,  3],
          #                [ 4,  5, 16],
          #                [17,  8,  9],
          #                [10, 21, 12]])
```

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)    # Find the elements of a that are bigger than 2;
                      # this returns a numpy array of Booleans of the same
                      # shape as a, where each slot of bool_idx tells
                      # whether that element of a is > 2.

print(bool_idx)       # Prints "[[False False]
                      #          [ True  True]
                      #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

```
import numpy as np

x = np.array([1, 2])    # Let numpy choose the datatype
print(x.dtype)          # Prints "int64"

x = np.array([1.0, 2.0])    # Let numpy choose the datatype
print(x.dtype)              # Prints "float64"

x = np.array([1, 2], dtype=np.int64)    # Force a particular datatype
print(x.dtype)
```

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

```
import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])
# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x)     # Prints "[[1 2]
             #          [3 4]]"
print(x.T)   # Prints "[[1 3]
             #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)     # Prints "[1 2 3]"
print(v.T)   # Prints "[1 2 3]"
```

Linear Algebra

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
b = np.array([[5], [6]])
print("A: ",A)
print("b: ",b)
print("Inverse: of ",np.linalg.inv(A))
print("Solution Ax=b", np.linalg.solve(A,b))
print("Determinant of A: ", np.linalg.det(A))
print("Column Sum norm: ",np.linalg.norm(A,1))
print("Row sum norm: ",np.linalg.norm(A,np.inf))
```

Python Matplotlib

Panchatcharam M

Associate Professor

Department of Mathematics and Statistics,
IIT Tirupati

- Plotting library
- Huge number of examples for tacking unique problems

Matplotlib-plot function

- `import matplotlib as mpl`
- `import matplotlib.pyplot as plt`
- `plt.plot(x_values, y_values, format_string [, x, y, format, ])`

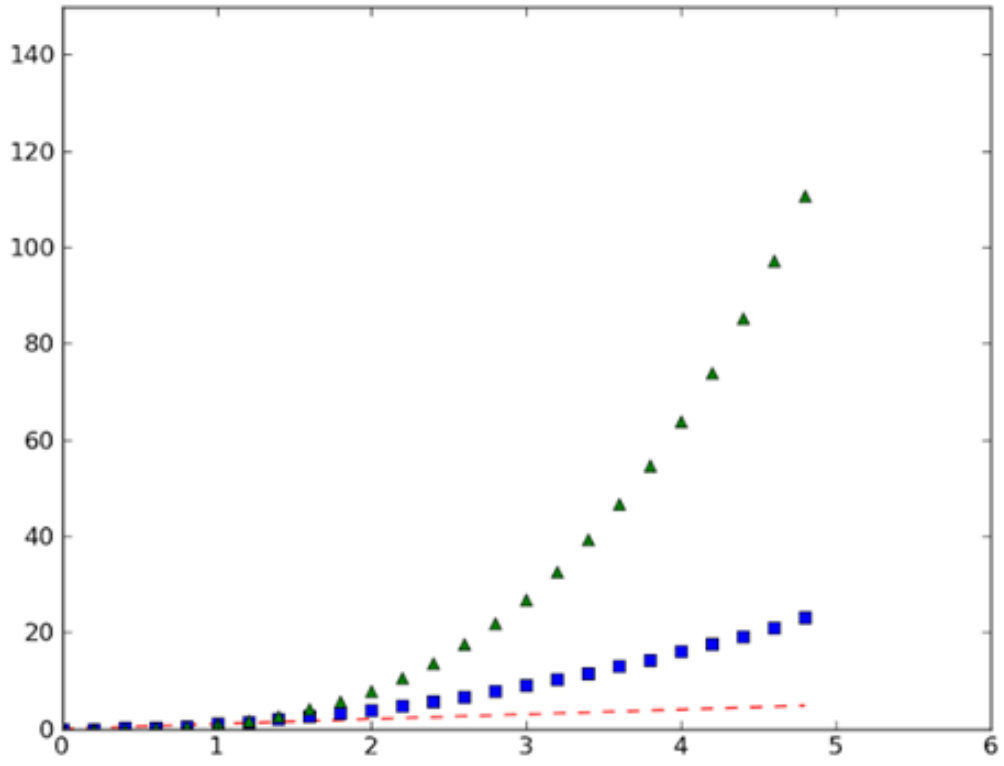
Matplotlib-plot function

- `import matplotlib as mpl`
- `import matplotlib.pyplot as plt`
- `plt.plot(x_values, y_values, format_string [, x, y, format, ])`
- `x_values` and `y_values`: numpy arrays, if not, internal conversion to numpy array
- `format_string`: the color and line type of the plot
  - 'bs' blue squares
  - 'ro': red circles
- Line properties: Set them via keyword arguments to the plot function.
  - `label, linewidth, animated, color,` etc…

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

# evenly sampled time at .2 intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.axis([0, 6, 0, 150]) # x and y range of axis
plt.show()
```

Matplotlib-plot function

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

# evenly sampled time at .2 intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.axis([0, 6, 0, 150]) # x and y range of ax
plt.show()
```

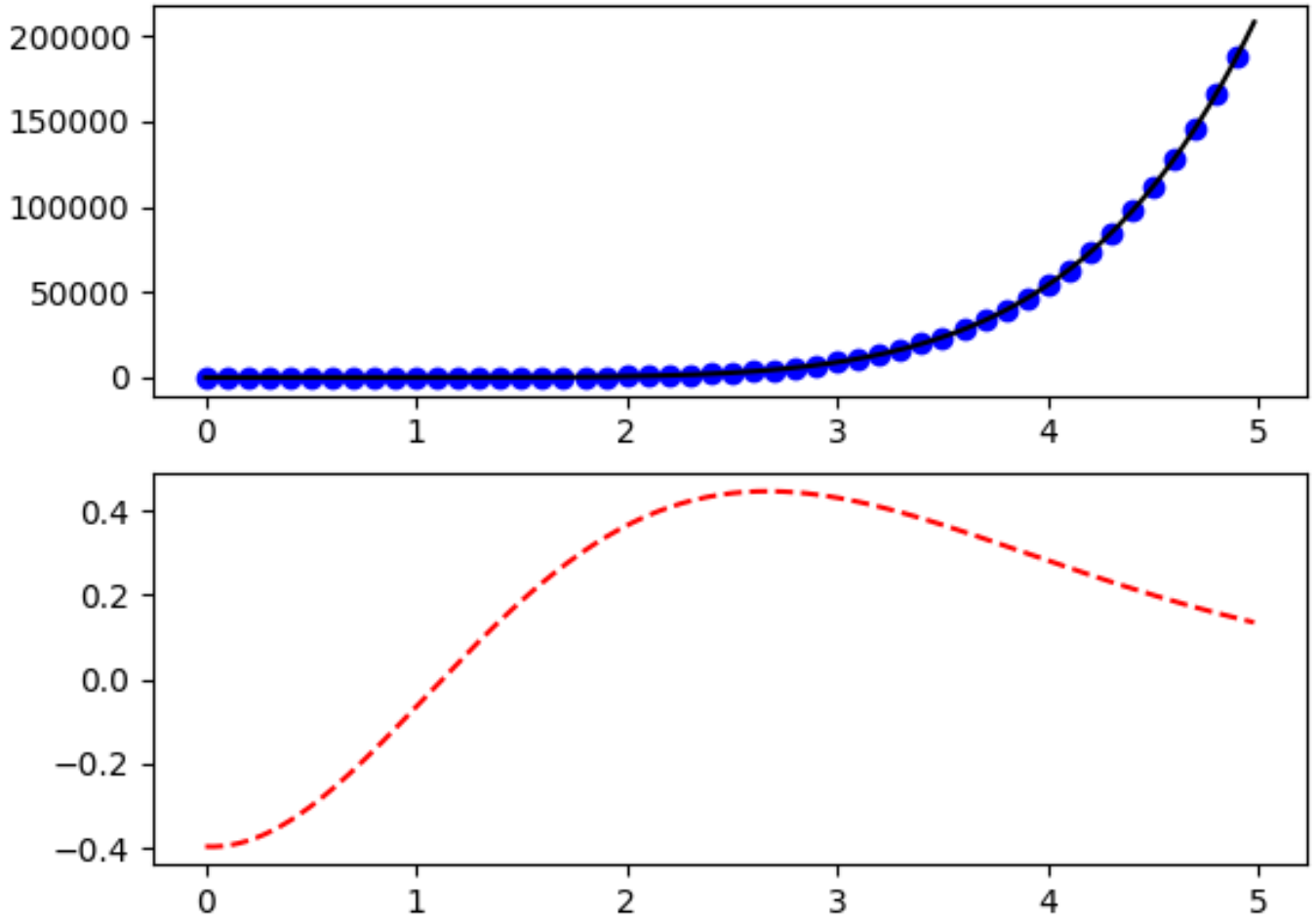


```
import numpy as np
import matplotlib.figure as figure

t = np.arange(0, 5, .2)

f = figure.Figure()
axes = f.add_subplot(111)
axes.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
axes.axis([0, 6, 0, 150])
```

Matplotlib-subplot function

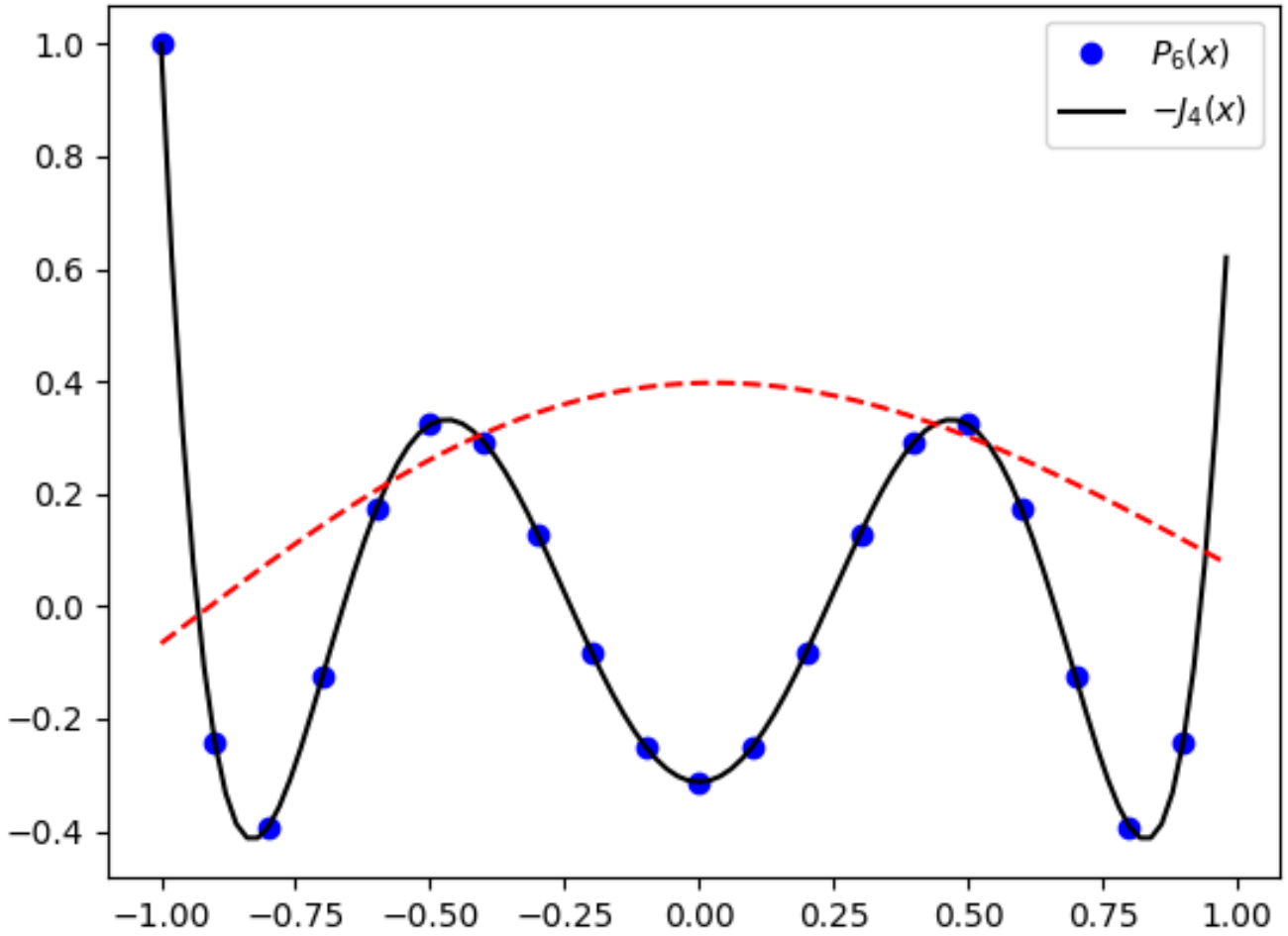
- A script can generate multiple figures, but typically you'll only have one.
- To create multiple plots within a figure, either use the subplot() function which manages the layout of the figure or use add_axes().

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

def Legendre(t,n):
    p=scipy.special.legendre(n)
    return p(t)
def Bessel(t,n):
    return scipy.special.jv(t,n)
t1 = np.arange(0.0, 5.0, 0.1) #50
t2 = np.arange(0.0, 5.0, 0.02) #250
plt.figure(1)
n=6
plt.subplot(211) # 2 rows, 1 column, 1st plot
plt.plot(t1, Legendre(t1,n), 'bo', t2, Legendre(t2,n), 'k')
plt.subplot(212) # 2 rows, 1 column, 2nd plot
plt.plot(t2, Bessel(t2,4), 'r--')
plt.show()
```


200000
150000
100000
50000
0
0
1
2
3
4
5
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5

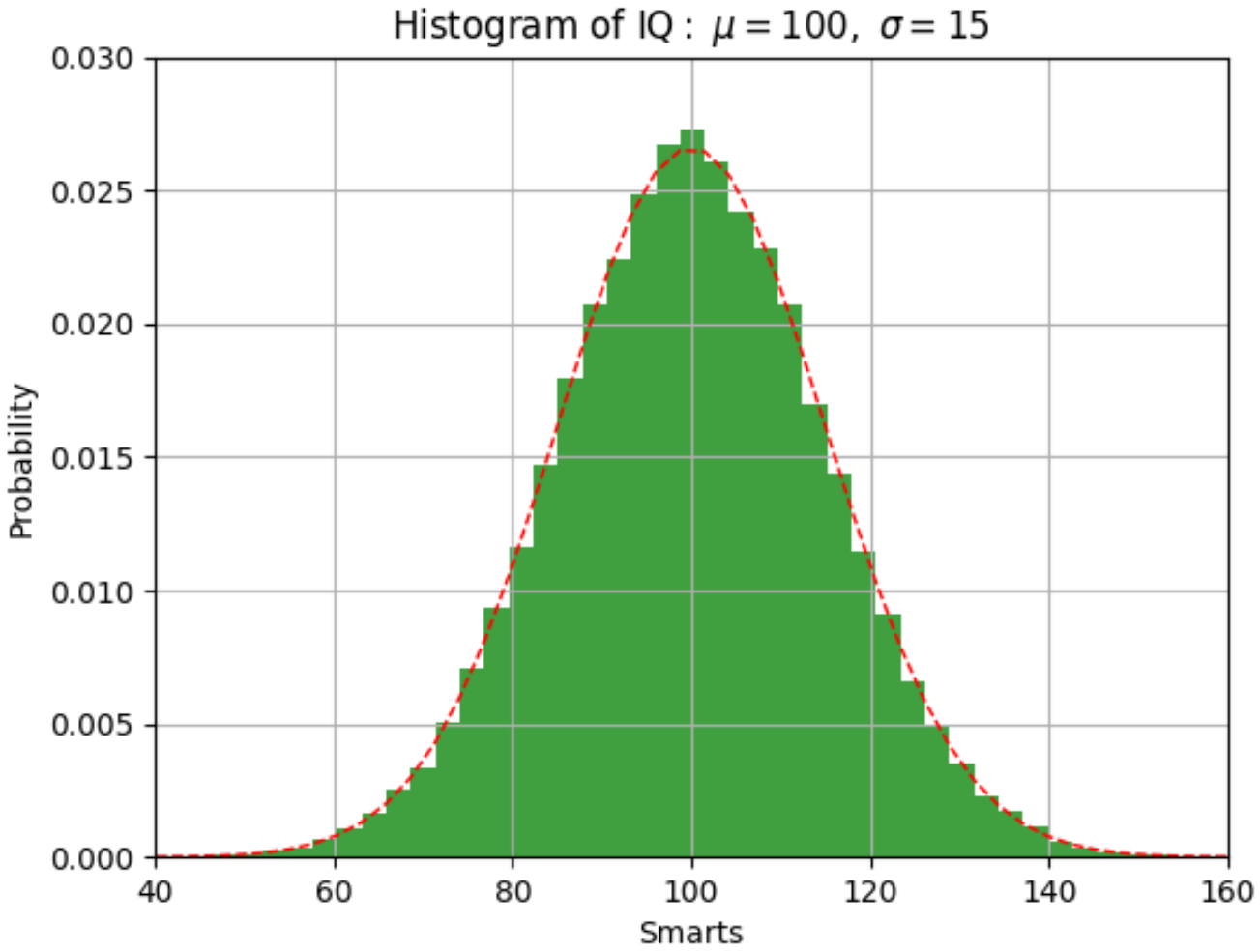
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special

def Legendre(t,n):
    p=scipy.special.legendre(n)
    return p(t)
def Bessel(t,n):
    return scipy.special.jv(t,n)
t1 = np.arange(-1.0, 1.0, 0.1)
t2 = np.arange(-1.0, 1.0, 0.02)
plt.figure(1)
n=6
plt.plot(t1, Legendre(t1,n), 'bo', t2, Legendre(t2,n), 'k')
plt.plot(t2, -Bessel(t2,4), 'r--')
plt.legend(['$P_6(x)$','$-J_4(x)$'])
plt.show()
```

Matplotlib-plot function



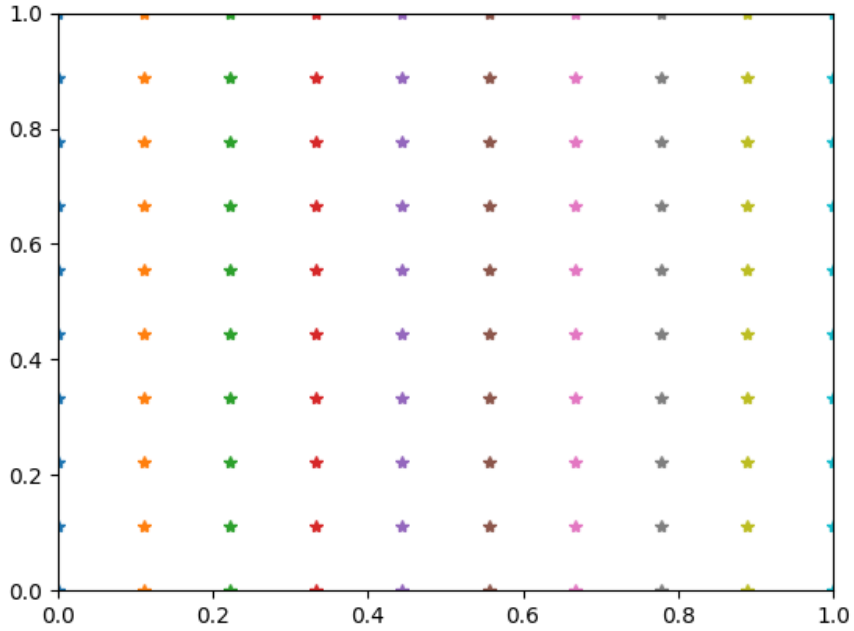
- `text()`: to add text in an arbitrary location
- `xlabel()`, `ylabel()`, `title()`, axes labels and title
- `clear()` removes all plots from the axes.

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
mu, sigma = 100, 15
x = mu + sigma*np.random.randn(100000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=True, facecolor='green', alpha=0.75)
# add a 'best fit' line
y = norm.pdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

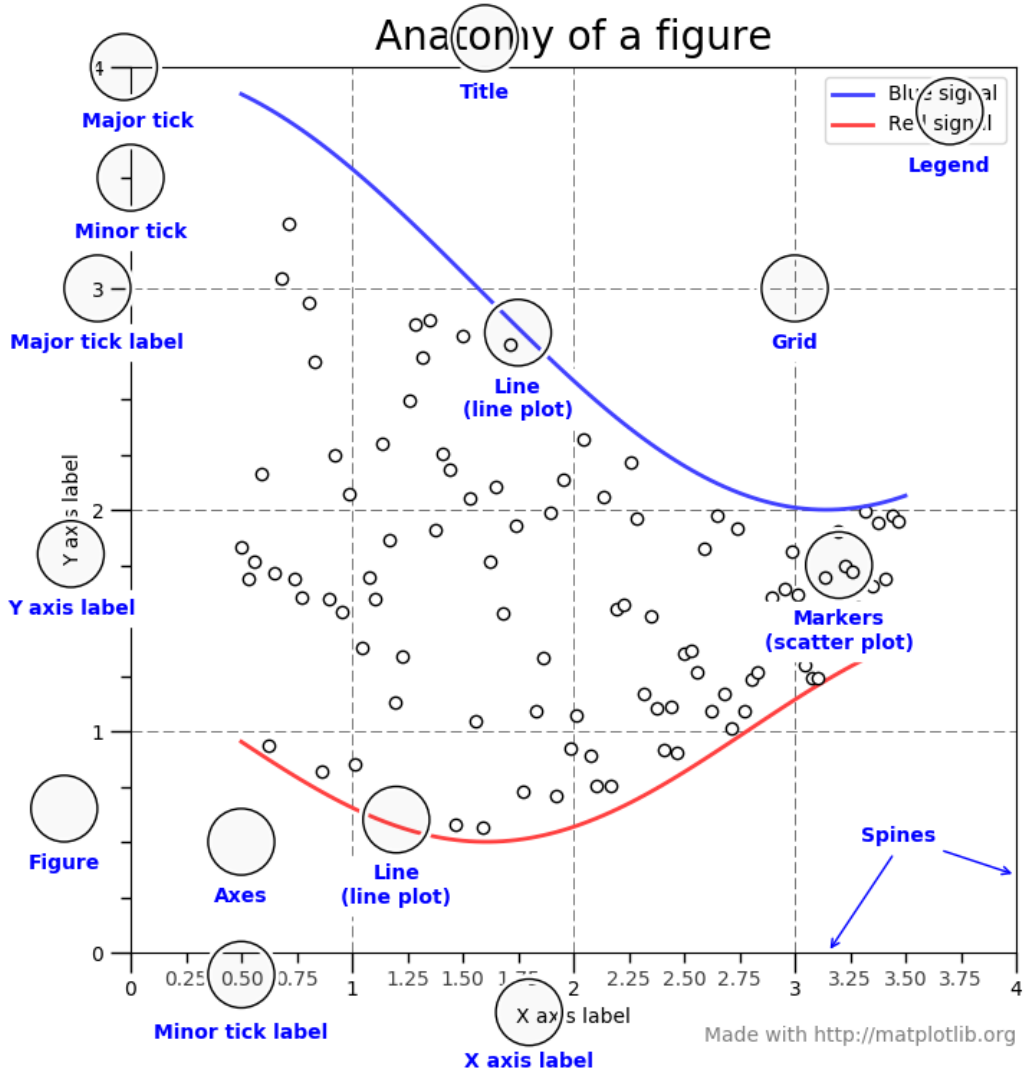

Histogram of IQ : $\mu = 100$, $\sigma = 15$
Probability
0.030
0.025
0.020
0.015
0.010
0.005
0.000
40
60
80
100
120
140
160
Smarts

Matplotlib-plot function

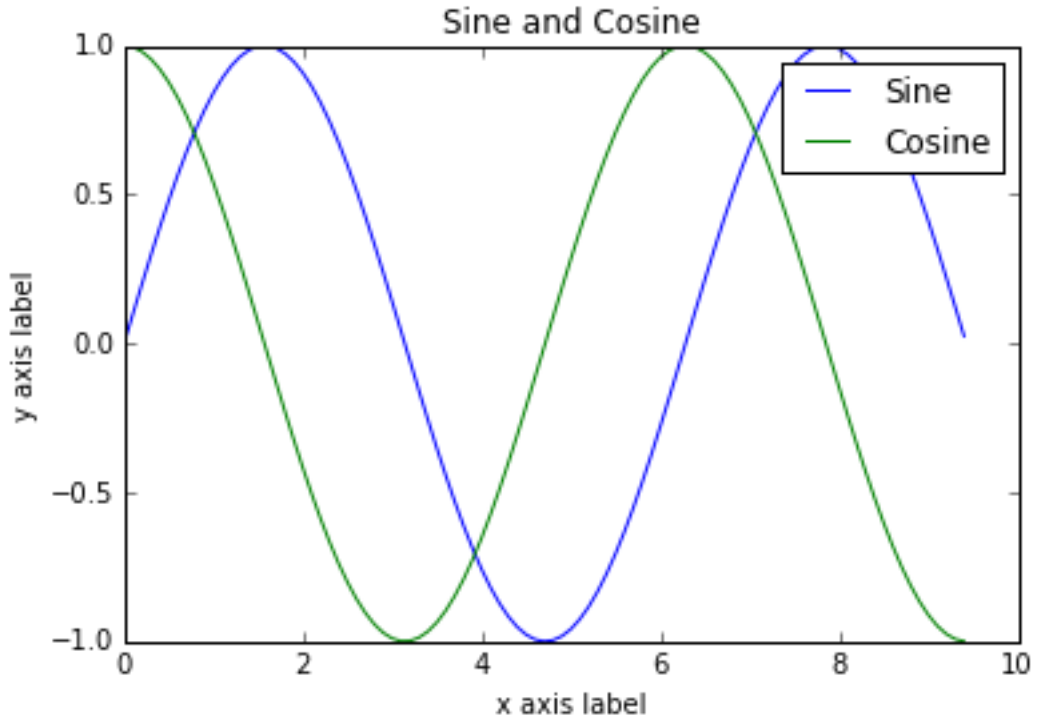
```
#FDM Discretization
import numpy as np
x=np.linspace(0,1,num=10,endpoint=True)
y=np.linspace(0,1,num=10,endpoint=True)
X,Y=np.meshgrid(x,y)
import matplotlib.pyplot as plt
plt.plot(X,Y,'*')
plt.axis([0,1,0,1])
plt.show()
```



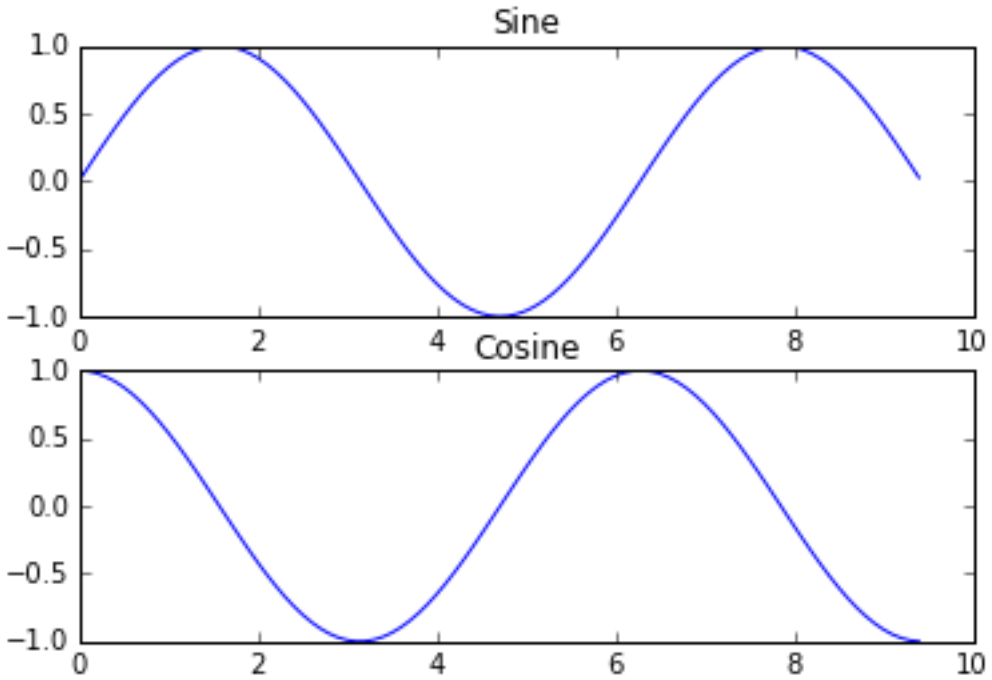
Matplotlib-plot function



Produce the following Graph using Matplotlib

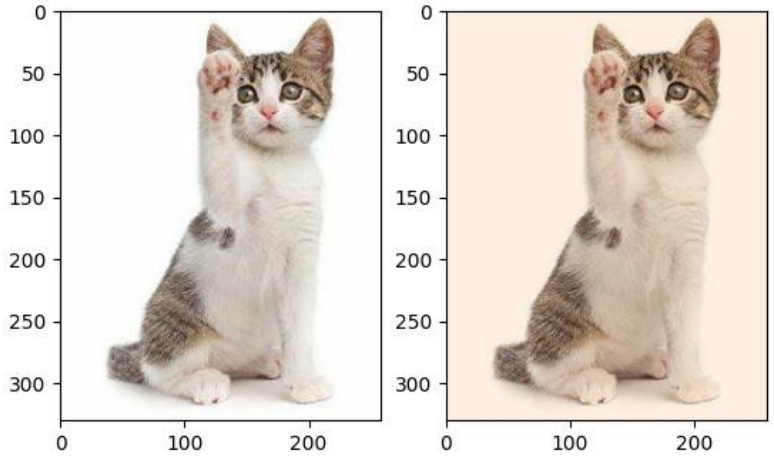


Produce the following Graph using Matplotlib

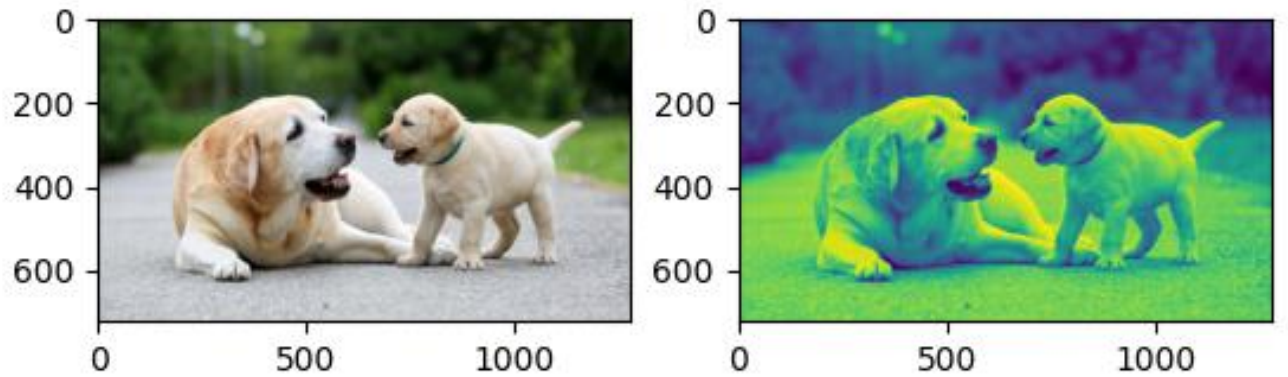


Matplotlib-plot function

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

img = imread('cat.jpg')
img_tinted = img * [1, 0.95, 0.9]

plt.subplot(1, 2, 1)
plt.imshow(img)


plt.subplot(1, 2, 2)
plt.imshow(np.uint8(img_tinted))
plt.show()
```

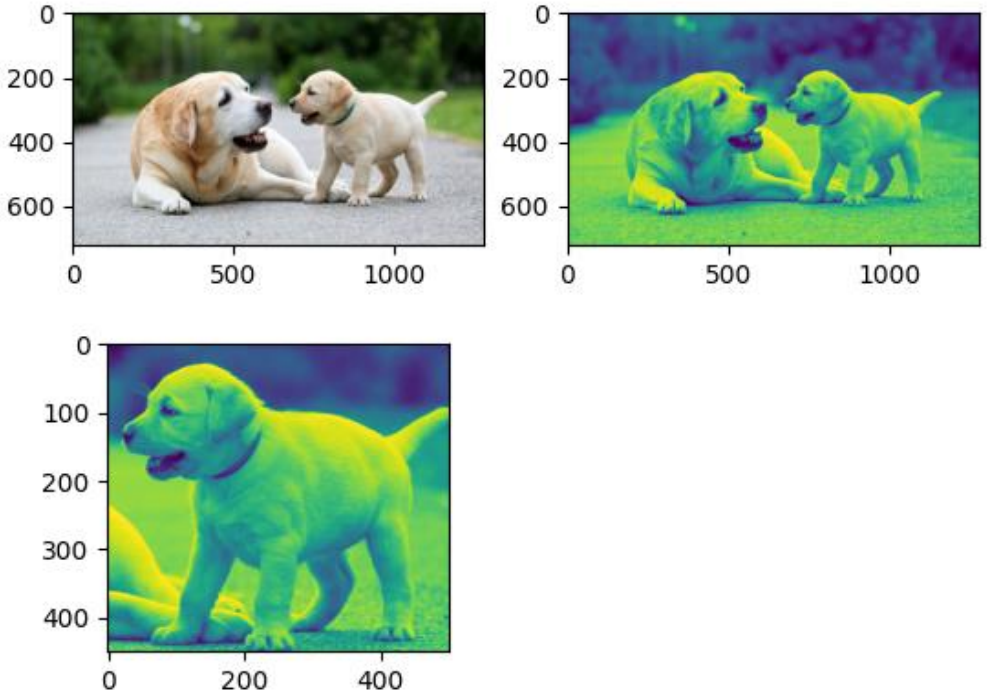


```
##Dog
import numpy as np
from matplotlib.pyplot import imread
import matplotlib.pyplot as plt

def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    return 0.2989 * r + 0.5870 * g + 0.1140 * b
img = imread('dog.jpg')
grayscale=rgb2gray(img)
print(grayscale.shape)
plt.subplot(1, 2, 1)
plt.imshow(img)
plt.subplot(1, 2, 2)
plt.imshow(grayscale)
plt.show()
```

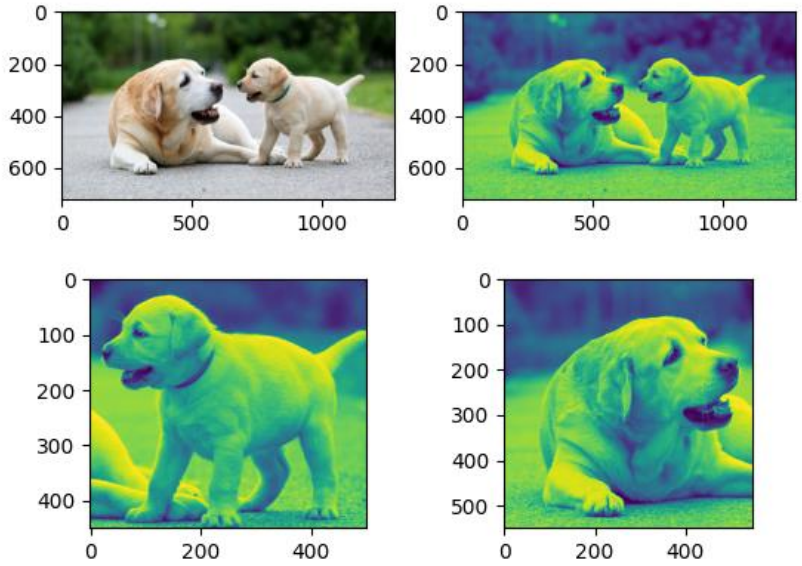


Matplotlib-plot Slicing

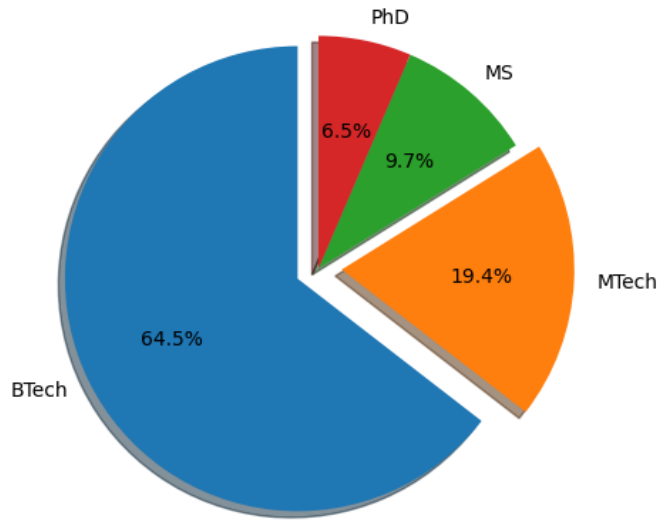
```
plt.subplot(2, 2, 1)
plt.imshow(img)

plt.subplot(2, 2, 2)
plt.imshow(grayscale)

plt.subplot(2, 2, 3)
plt.imshow(grayscale[150:600,650:1150])
plt.show()
```



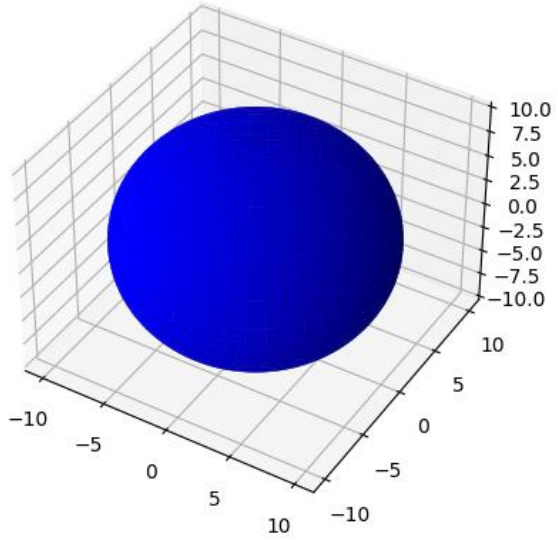
Produce the following Graph using Matplotlib



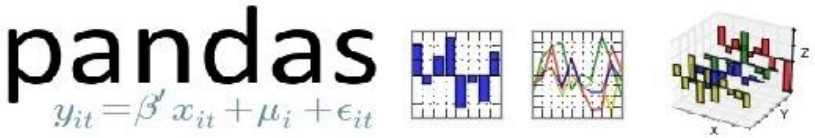
```
import matplotlib.pyplot as plt

labels = 'BTech', 'MTech', 'MS', 'PhD'
sizes = [100, 30, 15, 10]
explode = (0.1, 0.1, 0, 0)  # only "explode"

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode,
labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Make data
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 10 * np.outer(np.cos(u), np.sin(v))
y = 10 * np.outer(np.sin(u), np.sin(v))
z = 10 * np.outer(np.ones(np.size(u)),
np.cos(v))

# Plot the surface
ax.plot_surface(x, y, z, color='b')
plt.show()
```



End of Python Matplotlib



Python Pandas

Panchatcharam M

Associate Professor

Department of Mathematics and Statistics,
IIT Tirupati

- A Python Library – Working with datasets
- Manipulating Tables
- Built on top of Numpy
- Easy to visualize using matplotlib
- Data Structures – Series and DataFrame

- Analyze Data
- Clean Data
- Explore Data
- Manipulate Data
- Pandas – Panel Data, Python Data Analysis
- Created by McKinney (2008)

- Analyze Big Data
- Make Conclusions based on Statistical Theories
- Clean Messy Datasets, make it readable and relevant so that useful in data science

- `import numpy as np`
- `import pandas as pd`

Basic Data Structures in Pandas

- **Series:** A one-dimensional labelled array holding data of any type
- Integers, string, python objects etc

```
s=pd.Series([1,3,5,np.nan,6,9])
print(s)
```

- **DataFrame:** A two-dimensional data structure that holds data like two-dimensional array or a table with rows and columns

```
d={'col1':[1,2],'col2':[3,4]}
df=pd.DataFrame(data=d)
Print(df)
```

```
import pandas as pd
import numpy as np
dates = pd.date_range("20240201", periods=8)
df = pd.DataFrame(np.random.randn(8, 6), index=dates,
columns=list("ABCDEF"))
print(df)
```

```
mydata={
        "Roll Number":["MA24M001","MA24M002","MA24M004","MA24M005"],
        "Name":[“ABHAY",“ABLIPSA",“DEBASISH P",“DEV UTTAM"],
        "MA508L": [85,75,95,45],
        "MA509L": [65,35,85,65],
        "MA510L": [82,72,52,55],
        "MA511L": [81,71,65,75],
        "MA632L": [55,61,45,25],
        "MA601L": [52,75,62,95],
        "MA522M": [85,71,65,75],
    }
df=pd.DataFrame(data=mydata)
print(df)
```

```
df = pd.DataFrame(
     {
          "Roll Number":["MA24M001","MA24M002","MA24M004","MA24M005"],
          "Name":[“ABHAY",“ABLIPSA",“DEBASISH P",“DEV UTTAM"],
          "MA508L": [85,75,95,45],
          "MA509L": [65,35,85,65],
          "MA510L": [82,72,52,55],
          "MA511L": [81,71,65,75],
          "MA632L": [55,61,45,25],
          "MA601L": [52,75,62,95],
          "MA522M": [85,71,65,75],
     }
)
print(df)
```

```
df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
                   columns=['a', 'b', 'c’])
print(df2)
```

Syntax:

```
pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
```

Syntax:

```
pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
```

- Data: ndarray (homogenous or structure, iterable, dict or DataFrame
  - Dict: Series, arrays, constants, dataclass or list-like objects.
    - Column order follows insertion-order.
  - Series which have an index defined, it is aligned by its index.
    - This alignment also occurs if data is a Series or a DataFrame itself.
    - Alignment is done on Series/DataFrame inputs.
  - dicts, column order follows insertion-order.
- Index:
  - Rangeindex if no indexing provided
- Columns:
  - Default rangeindex, otherwise column labels

```
df.head(2) #To view the top 2 rows
df.tail(3) #To view the bottom 3 rows
df.index   #To display the index labels of the DataFrame (default 0,1,2)
#RangeIndex(start=0, stop=4, step=1)
df.columns #To display the column name
#Index(['Roll Number', 'Name', 'MA508L', 'MA509L', 'MA510L', 'MA511L',
'MA632L', 'MA601L', 'MA522M'],

df.to_numpy() #converts data to numpy array.
```

Note: Numpy arrays have one dtype for the entire array while pandas DataFrames have one dtype per column

```
df.ndim     #To number of dimension (2 for above case)
df.size     #number of elements in the object (36)
df.sum(axis=1) #To compute column sum
df.sum(axis=0) #To compute row sum
df.T # Transpose of your data
```

```
df.to_numpy() #converts data to numpy array.
```

Note: Numpy arrays have one dtype for the entire array while pandas DataFrames have one dtype per column

Quick Statistic Summary of your Data

```
df.describe()
```

This command shows a quick statistics summary of your data

```
          MA508L     MA509L     MA510L     MA511L     MA632L   MA601L  \
count   4.000000   4.000000   4.000000   4.000000   4.000000   4.0000
mean   75.000000  62.500000  65.250000  73.000000  46.500000  71.0000
std    21.602469  20.615528  14.221463   6.733003  15.779734  18.5652
min    45.000000  35.000000  52.000000  65.000000  25.000000  52.0000
25%    67.500000  57.500000  54.250000  69.500000  40.000000  59.5000
50%    80.000000  65.000000  63.500000  73.000000  50.000000  68.5000
75%    87.500000  70.000000  74.500000  76.500000  56.500000  80.0000
max    95.000000  85.000000  82.000000  81.000000  61.000000  95.0000

          MA522M
count   4.000000
mean   74.000000
std     8.406347
min    65.000000
25%    69.500000
50%    73.000000
75%    77.500000
max    85.000000
```

Count the number of null observations
Max/min/mean/std: max/min/mean/std values in the object
25%/50%/75%: respective percentiles

```
df.mean() #mean
df.mean(axis=1) #mean in axis=1
df.max(axis=0)    #max in axis=0
df.std(axis=0)    #standard deviations
df.sum(axis=0)    #sum of the values over the requested axis
df.skew(axis=0)   #unbiased skew over requested axis
df.sem(axis=0)    #unbiased standard error of the mean over requested axis
df.kurt(axis=0)   #unbiased kurtosis
df.kurtosis(axis=0)     #unbiased kurtosis using Fisher's definition
df.corr(method='pearson') #correlation of columns
#method: pearson. Kendall, spearman
df.corrwith(other,axis=0) #compute pairwise correlation
```

Sorting (by index)

```
df.sort_index(axis=1,ascending=False)
#axis 0 for row, 1 for column, default 0
```

Syntax:

```
df.sort_index(*, axis=0, level=None, ascending=True, inplace=False,
kind='quicksort', na_position='last', sort_remaining=True,
ignore_index=False, key=None)[source]
```

Kind: quicksort, mergesort, heapsort, stable, default: quicksort

```
df = pd.DataFrame({"a": [1, 2, 3, 4]}, index=['A', 'b', 'C', 'd'])

df.sort_index(key=lambda x: x.str.lower())
```

Sorting (by values)

```
df.sort_values(by= "MA508L") #Sort by values along either axis
```

Syntax

```
df.sort_values(by, *, axis=0, ascending=True, inplace=False, kind='quick
sort', na_position='last', ignore_index=False, key=None)
```

```
Kind: quicksort, mergesort, heapsort, stable, default: quicksort
```

```
df[“MA508L”] #To get the column corresponds to MA508L
df.MA508L   #To get the column corresponds to MA508L
df[1:3]     #slice matching rows

df.loc[0]   #selecting a row matching the label
df.loc[:, [“MA508L”, “MA509L”] #multiple columns

df.iloc[2]  #select by position of the passed integers
df.iloc[3:5, 0:2] #select slices similar to numpy
df.iloc[[1, 2, 4], [0, 2]] #Lists of integer position locations

df[df[“MA508L"] > 75]   #select rows where df.MA508L is greater than 75
```

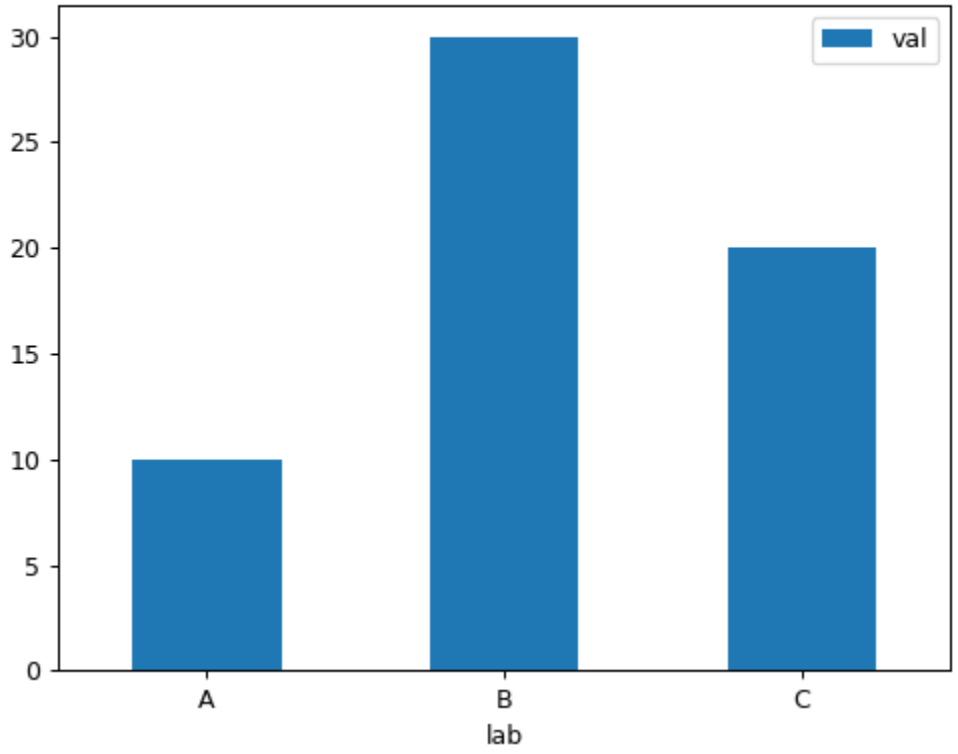
```
df.insert(1,”Age”,[21,22,20,22]) #insert a new column
df.insert(1,”Age”,[21,22,20,22],allow_duplicates=True)
#insert a new if already same column name exists
newdata={'Roll
Number':’MA24M006','Age':24,'Name’:’Krishna','MA508L':75,'MA509L':55,'MA51
0L':98,'MA511L':87,’MA632L':75,'MA601L':82,'MA522M':63}
df=df.append(newdata,ignore_index=True)
newdata=pd.DataFrame({'Roll
Number':’MA24M011','Age':24,'Name':’Sandeep','MA508L':75,'MA509L':55,'MA51
0L':98,'MA511L':87,’MA632L':75,'MA601L':82,'MA522M':63},index=[0])
df2=pd.concat([newdata,df.loc[:]]).reset_index(drop=True)
print(df2)
```

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
print(pd.options.display.max_rows)
```

Read json Files

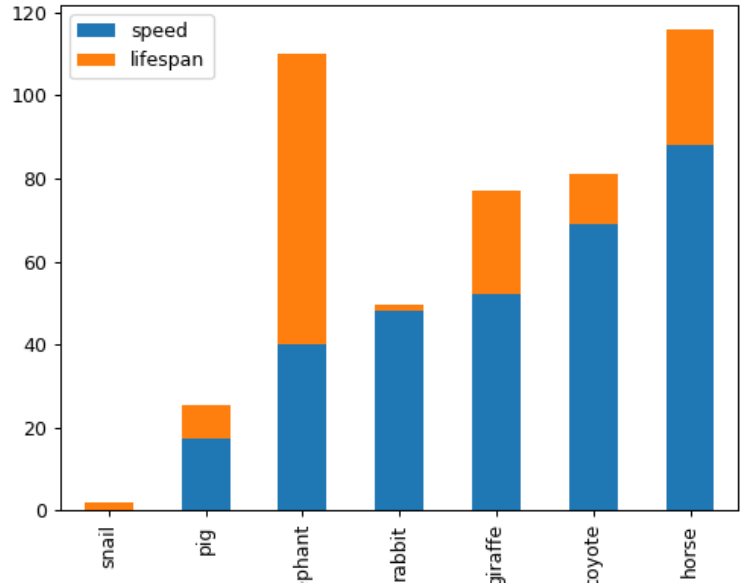
```
import pandas as pd
df = pd.read_json('data.json')
print(df.to_string())
```

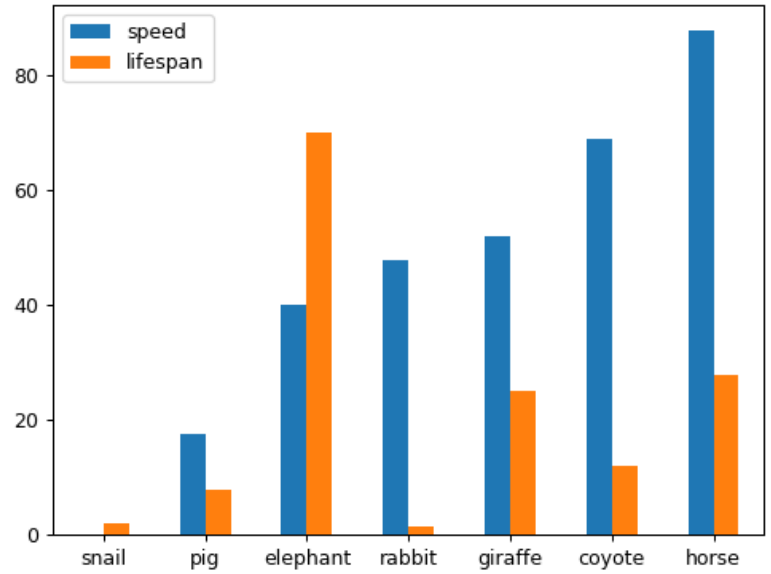
```
df = pd.DataFrame({'lab':['A', 'B', 'C'], 'val':[10, 30, 20]})
ax = df.plot.bar(x='lab', y='val', rot=0)
```



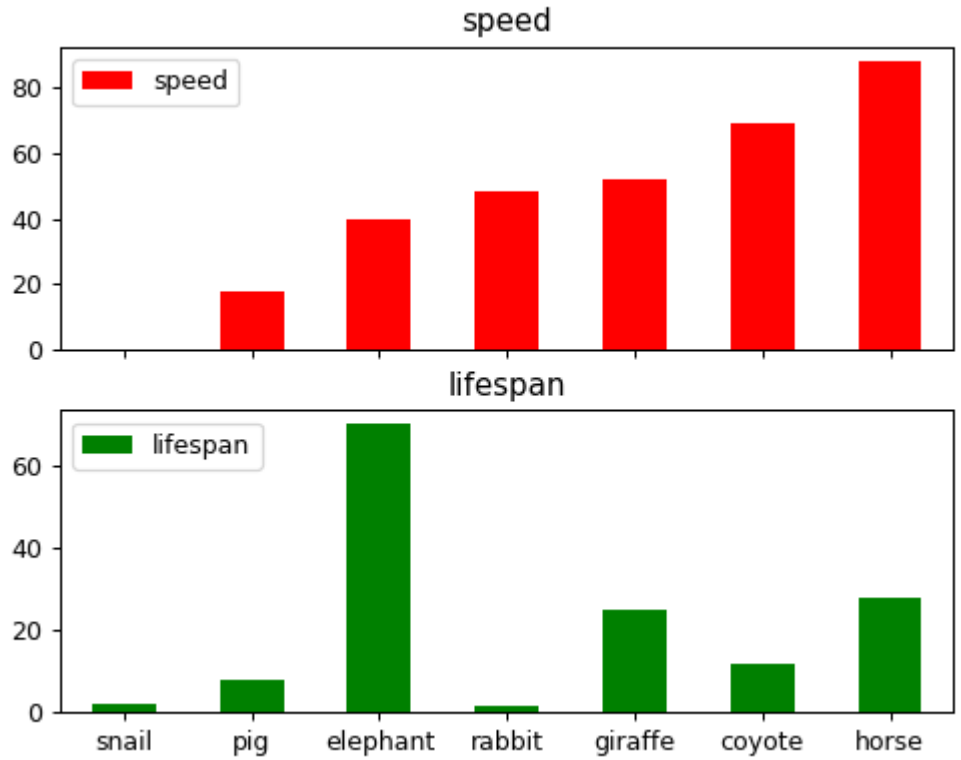
```
speed = [0.1, 17.5, 40, 48, 52, 69, 88]
lifespan = [2, 8, 70, 1.5, 25, 12, 28]
index = ['snail', 'pig', 'elephant','rabbit', 'giraffe', 'coyote', 'horse']
df = pd.DataFrame({'speed': speed,'lifespan': lifespan}, index=index)
ax = df.plot.bar(rot=0)
```

```
ax = df.plot.bar(stacked=True)
```

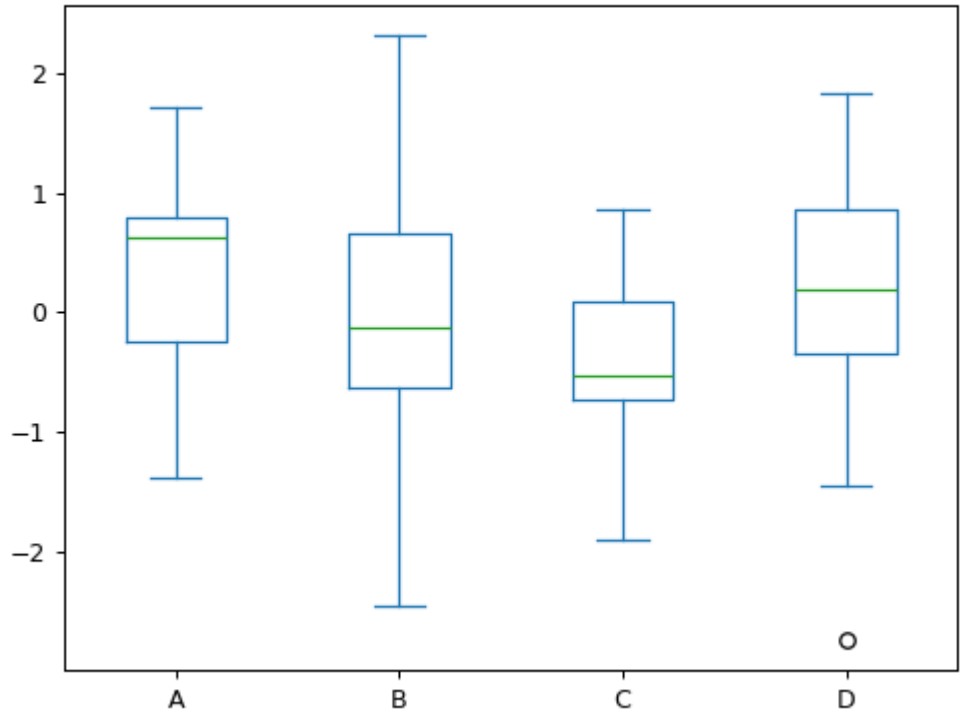




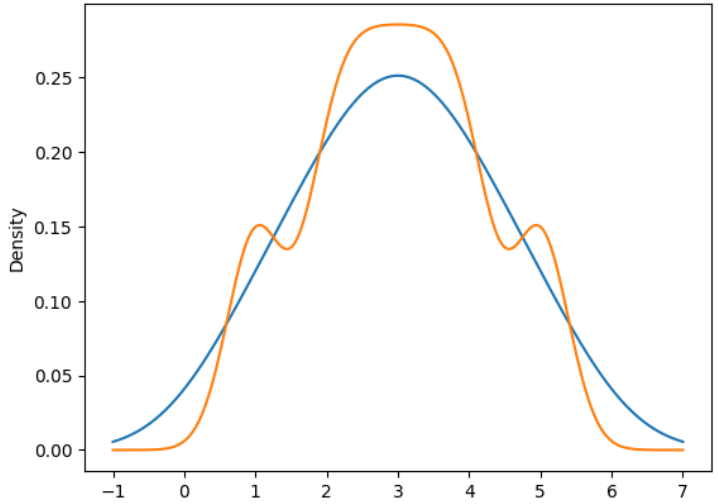
```
axes = df.plot.bar( ... rot=0, subplots=True,
       color={"speed": "red", "lifespan": "green"} ... )
axes[1].legend(loc=2)
```



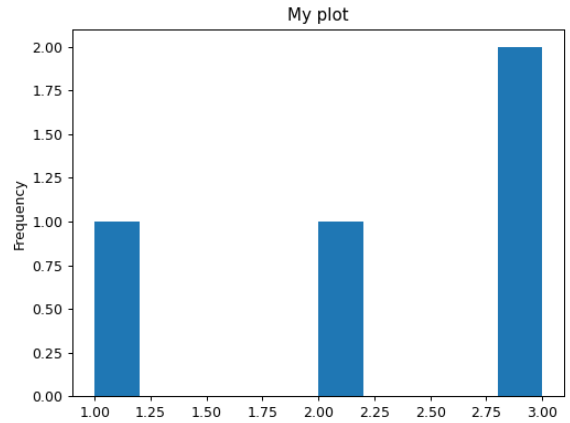
```
data = np.random.randn(25, 4)
df = pd.DataFrame(data, columns=list('ABCD'))
ax = df.plot.box()
```



```
s = pd.Series([1, 2, 2.5, 3, 3.5, 4, 5])
ax = s.plot.kde()
ax = s.plot.kde(bw_method=0.3)
```

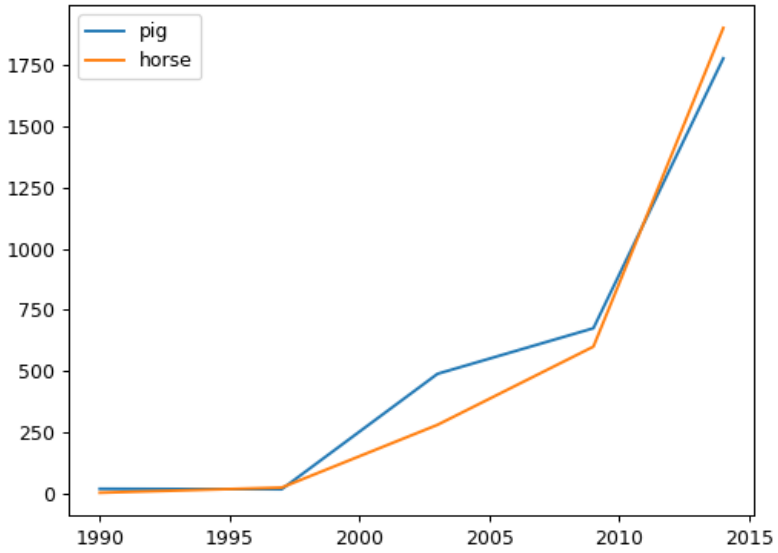



```
ser = pd.Series([1, 2, 3, 3])
plot = ser.plot(kind='hist', title="My plot")
```

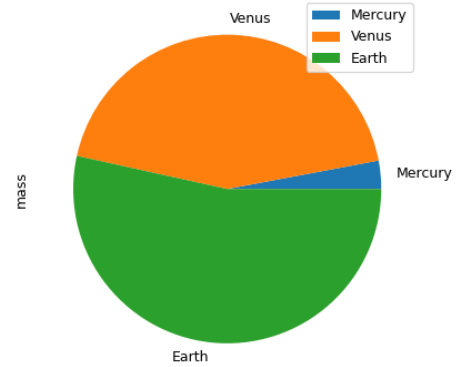


Line and Pie Plotting

```
df = pd.DataFrame({
    'pig': [20, 18, 489, 675, 1776],
    'horse': [4, 25, 281, 600, 1900]
    }, index=[1990, 1997, 2003, 2009, 2014])
lines = df.plot.line()
```




```
df = pd.DataFrame({'mass': [0.330, 4.87 , 5.97],
                   'radius': [2439.7, 6051.8, 6378.1]},
                  index=['Mercury', 'Venus', 'Earth'])
plot = df.plot.pie(y='mass', figsize=(5, 5))
```



End of Python Pandas