Python Fundamentals for Machine Learning

Edited By : Dr.Karthikeyan , Machine Learning Lab # Date : 5/3/2022

My First Program

In [1]:

```
# Program to find simple interest
p = int(input("\n Enter the principal Amount:"))
t = int(input("\n Enter the time period:"))
r = float(input("\n Enter the rate of interest:"))
si = p*t*r/100
print("\n Simple Interest:",si)
```

```
 Enter the principal Amount:1000

 Enter the time period:2

 Enter the rate of interest:1.2

 Simple Interest: 24.0
```

1.0 Output using Print

In [2]:

```
print('''This sentence is output to the screen''')
a=5
print("The value of a is:",a)
print('x:',1,2,3,4)
x = 5 ; y = 10
print('The value of x is {} and y is {}'.format(x,y))
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

```
This sentence is output to the screen
The value of a is: 5
x: 1 2 3 4
The value of x is 5 and y is 10
I love bread and butter
I love butter and bread
```

In [3]:

```
print('Hello {name}, {greeting}'.format(greeting = 'Good Morning!!',\
                                        name = 'John'))
```

```
Hello John, Good Morning!!
```

In [4]:

```
x = 12.3456789
print('The value of x is %3.2f' %x)
print('The value of x is %3.4f' %x)
```

```
The value of x is 12.35
The value of x is 12.3457
```

In [5]:

```
for x in range(1, 11):
     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

```
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

In [6]:

```
table = {'Raju': 9480123526, 'Ravi': 9480123527, 'Rahul': 9480123528}
for name, phone in table.items():
     print('{0:10} ==> {1:10d}'.format(name, phone))
```

```
Raju       ==> 9480123526
Ravi       ==> 9480123527
Rahul      ==> 9480123528
```

In [7]:

```
import math
print('The value of PI is approximately %5.3f.' % math.pi)
```

```
The value of PI is approximately 3.142.
```

1.1 Input using input

In [8]:

```
x = input('Enter a string: ')
print("The entered string is :{0}".format(x))
y = int(input('Enter a integer: '))
print("The entered integer is :",y)
z = float(input('Enter a floating point number:'))
print("The entered real number  is :",z)
```

```
Enter a string: tgs
The entered string is :tgs
Enter a integer: 100
The entered integer is : 100
Enter a floating point number:23.5
The entered real number  is : 23.5
```

1.3 Mutiline Statements

In [9]:

```
# Example of implicit line continuation
x = ('1' + '2' +
     '3' + '4')
# Example of explicit line continuation
y = '1' + '2' + \
'11' + '12'
weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
weekday = {'one':
           'Monday'}
print ('x has a value of', x)
print ('y has a value of', y)
print(weekdays)
print(weekday)
```

```
x has a value of 1234
```

```
y has a value of 121112
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
{'one': 'Monday'}
```

In [10]:

```
import os; x = 'Hello'; print(x)
```

```
Hello
```

2.0 Conditional Execution

Example code for a simple ‘if’ statement

$$y = f(x^2)/2$$

In [11]:

```
var = -1
if var < 0:
    print(var)
    print("the value of var is negative")
# If there is only a single clause then it may go on the same line as the
# header statement
if ( var == -1 ) :
    print("the value of var is negative")
```

```
-1
the value of var is negative
the value of var is negative
```

#Example code for ‘if else’ statement

In [12]:

```
var = 1
if var < 0:
    print("the value of var is negative")
    print(var)
else:
    print("the value of var is positive")
    print(var)
```

```
the value of var is positive
1
```

Example for nested if else

In [13]:

```
score = 95
if score >= 99:
    print('A')
elif score >=75:
    print('B')
elif score >= 60:
    print('C')
elif score >= 35:
    print('D')
else:
    print('F')
```

```
B
```

3.0 Iterations

Usage of For Loop

In [14]:

```
# First Example
print("First Example")
for item in [1,2,3,4,5]:
    print('item :', item)
# Second Example
print("Second Example")
letters = ['A', 'B', 'C']
for index in range(len(letters)):
    print('First loop letter :', letters[index])
```

```
First Example
item : 1
item : 2
item : 3
item : 4
item : 5
Second Example
First loop letter : A
First loop letter : B
First loop letter : C
```

#While loop: The while statement repeats a set of code until the condition is true.

In [15]:

```
#Example code for while loop statement
count = 0
while (count <3):
    print('The count is:', count)
    count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
```

4.1 LISTS

Python's lists are the most flexible data type. It can be created by writing a list of comma separated values between square brackets. Note that that the items in the list need not be of the same data type.

In [16]:

```
# Example code for accessing lists
# Create lists
list_1 = ['Statistics', 'Programming', 2016, 2017, 2018]
list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7 ]
# Accessing values in lists
print("list_1[0]: ", list_1[0])
print("list2_[1:5]: ", list_2[1:5])
```

```
list_1[0]:  Statistics
list2_[1:5]:  ['b', 1, 2, 3]
```

In [17]:

```
#Example code for adding new values to lists
print("list_1 values: ", list_1)
# Adding new value to list
list_1.append(2019)
print("list_1 values post append: ", list_1)
```

```
list_1 values:  ['Statistics', 'Programming', 2016, 2017, 2018]
list_1 values post append:  ['Statistics', 'Programming', 2016, 2017, 2018, 2019]
```

In [18]:

In [18]:

```
#Example code for updating existing values of lists
print("Values of list_1: ", list_1)
# Updating existing value of list
print("Index 2 value : ", list_1[2])
list_1[2] = 2015;
print("Index 2's new value : ", list_1[2])
```

```
Values of list_1:  ['Statistics', 'Programming', 2016, 2017, 2018, 2019]
Index 2 value :  2016
Index 2's new value :  2015
```

In [19]:

```
#Example code for deleting a list element
print("list_1 values: ", list_1)
# Deleting list element
del list_1[5];
print("After deleting value at index 2 : ", list_1)
```

```
list_1 values:  ['Statistics', 'Programming', 2015, 2017, 2018, 2019]
After deleting value at index 2 :  ['Statistics', 'Programming', 2015, 2017, 2018]
```

Example code for basic operations on lists

In [20]:

```
import math
import string
import operator
#Example code for basic operations on lists

print("Length: ", len(list_1))

print("Concatenation: ", [1,2,3] + [4, 5, 6])

print("Repetition :", ['Hello'] * 4)

print("Membership :", 3 in [1,2,3])

print("Iteration :")
for x in [1,2,3]: print(x)

# Negative sign will count from the right
print("slicing :", list_1[-2])

# If you dont specify the end explicitly, all elements from the specified
#start index will be printed
print("slicing range: ", list_1[1:])

print("Max of list: ", max([1,2,3,4,5]))

print("Min of list: ", min([1,2,3,4,5]))

print("Count number of 1 in list: ", [1,1,2,3,4,5,].count(1))

list_1.extend(list_2)

print("Extended :", list_1)

print("Index for Programming:",list_1.index('Programming'))
print (list_1)
print("pop last item in list: ", list_1.pop())
print("pop the item with index 2: ", list_1.pop(2))
list_1.remove('b')
print("removed b from list: ", list_1)
list_1.reverse()
print("Reverse: ", list_1)
list_1 = ['a','c','b']
list_1.sort()
print("Sort ascending: ", list_1)
list_1.sort(reverse = True)
```

```
print("Sort descending: ", list_1)
```

```
Length:  5
Concatenation:  [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
1
2
3
slicing : 2017
slicing range:  ['Programming', 2015, 2017, 2018]
Max of list:  5
Min of list:  1
Count number of 1 in list:  2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
Index for Programming: 1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
pop last item in list:  7
pop the item with index 2:  2015
removed b from list:  ['Statistics', 'Programming', 2017, 2018, 'a', 1, 2, 3, 4, 5, 6]
Reverse:  [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistics']
Sort ascending:  ['a', 'b', 'c']
Sort descending:  ['c', 'b', 'a']
```

4.2 Tuples

A Python tuple is a sequences or series of immutable Python objects very much similar to the lists. However there exist some essential differences between lists and tuples, which are the following.

1. Unlike list, the objects of tuples cannot be changed.
2. Tuples are defined by using parentheses, but lists are defined by square brackets

In [21]:

```
# Example code for creating tuple
# Creating a tuple
Tuple = ()
print("Empty Tuple: ", Tuple)
Tuple = (1,)
print("Tuple with single item: ", Tuple)
Tuple = ('a','b','c','d',1,2,3)
print("Sample Tuple :", Tuple)
```

```
Empty Tuple:  ()
Tuple with single item:  (1,)
Sample Tuple : ('a', 'b', 'c', 'd', 1, 2, 3)
```

In [22]:

```
#Example code for accessing tuple
# Accessing items in tuple
Tuple = ('a', 'b', 'c', 'd', 1, 2, 3)
print("3rd item of Tuple:", Tuple[2])
print("First 3 items of Tuple", Tuple[0:2])
```

```
3rd item of Tuple: c
First 3 items of Tuple ('a', 'b')
```

In [23]:

```
#Example code for deleting tuple
# Deleting tuple
print("Sample Tuple: ", Tuple)
del Tuple
print(Tuple) # Will throw an error message as the tuple does not exist
```

```
Sample Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-ab07a54b7d35> in <module>()
      3 print("Sample Tuple: ", Tuple)
      4 del Tuple
----> 5 print(Tuple) # Will throw an error message as the tuple does not exist

NameError: name 'Tuple' is not defined
```

In [24]:

```
# Example code for basic operations on tupe (not exhaustive)
# Basic Tuple operations
Tuple = ('a','b','c','d',1,2,3)
print("Length of Tuple: ", len(Tuple))
Tuple_Concat = Tuple + (7,8,9)
print("Concatinated Tuple: ", Tuple_Concat)

print("Repetition: ", (1,'a',2, 'b') * 3)
print("Membership check: ", 3 in (1,2,3))
# Iteration
for x in (1, 2, 3): print(x)
print("Negative sign will retrieve item from right: ", Tuple_Concat[-2])
print("Sliced Tuple [2:] ", Tuple_Concat[2:])
# Find max
print("Max of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
max((1,2,3,4,5,6,7,8,9,10)))
print("Min of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
min((1,2,3,4,5,6,7,8,9,10)))
print("List [1,2,3,4] converted to tuple: ", type(tuple([1,2,3,4])))
```

```
Length of Tuple:  7
Concatinated Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3, 7, 8, 9)
Repetition:  (1, 'a', 2, 'b', 1, 'a', 2, 'b', 1, 'a', 2, 'b')
Membership check:  True
1
2
3
Negative sign will retrieve item from right:  8
Sliced Tuple [2:]  ('c', 'd', 1, 2, 3, 7, 8, 9)
Max of the Tuple (1,2,3,4,5,6,7,8,9,10):  10
Min of the Tuple (1,2,3,4,5,6,7,8,9,10):  1
List [1,2,3,4] converted to tuple:  <class 'tuple'>
```

4.3 Dictionary

The Python dictionary will have a key and value pair for each item that is part of it. The key and value should be enclosed in curly braces. Each key and value is separated using a colon (:), and further each item is separated by commas (,). Note that the keys are unique within a specific dictionary and must be immutable data types such as strings, numbers, or tuples, whereas values can take duplicate data of any type.

In [25]:

```
# Example code for creating dictionary
# Creating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

In [26]:

```
# Example code for accessing dictionary
# Accessing items in dictionary
print("Value of key Name, from sample dictionary:", dict['Name'])
```

```
Value of key Name, from sample dictionary: Jivin
```

In [27]:

```
#Example for deleting dictionary
# Deleting a dictionary

dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict0)
k=1
for i in dict0:
    print(k,i,dict0[i])
    k=k+1
del (dict0['Name']) # Delete specific item

print("Sample dictionary post deletion of item Name:", dict0)

dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
dict0.clear() # Clear all the contents of dictionary
print("dict post dict.clear():", dict0)

dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
del (dict0) # Delete the dictionary
#print(dict0)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
1 Name Jivin
2 Age 6
3 Class First
Sample dictionary post deletion of item Name: {'Age': 6, 'Class': 'First'}
dict post dict.clear(): {}
```

In [28]:

```
#Example code for updating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
dict['Age'] = 6.5
print("Dictionary post age value update: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Dictionary post age value update:  {'Name': 'Jivin', 'Age': 6.5, 'Class': 'First'}
```

In [29]:

```
#Example code for basic operations on dictionary
# Basic operations
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Length of dict: ", len(dict))

# Copy the dict
dict1 = dict.copy()
print("Copy:\n",dict1)

# Retrieve value for a given key
print("Value for Age: ", dict.get('Age'))

# Return items of dictionary
print("dict items: ", dict.items())

# Return items of keys
print("dict keys: ", dict.keys())

# return values of dict
print("Value of dict: ", dict.values())

# Concatenate dicts
dict1 = {'Name': 'Jivin', 'Age': 6}
dict2 = {'Sex': 'male' }
dict1.update(dict2)
print("dict1.update(dict2) = ", dict1)
```

```
Length of dict:  3
Copy:
 {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

```
Value for Age:  6
dict items:  dict_items([('Name', 'Jivin'), ('Age', 6), ('Class', 'First')])
dict keys:  dict_keys(['Name', 'Age', 'Class'])
Value of dict:  dict_values(['Jivin', 6, 'First'])
dict1.update(dict2) =  {'Name': 'Jivin', 'Age': 6, 'Sex': 'male'}
```

5.0 User-Defined Functions

A user-defined function is a block of related code statements that are organized to achieve a single related action. The key objective of the user-defined functions concept is to encourage modularity and enable reusability of code.

Syntax for creating functions without argument: def functoin_name(): 1st block line 2nd block line ...

In [30]:

```
# Example code for creating functions without argument

# Simple function
def someFunction():
    print("Hello World")

# Call the function
someFunction()
```

```
Hello World
```

Syntax for Creating Functions with Argument def functoin_name(parameters): 1st block line 2nd block line ... return [expression]

In [31]:

```
#Example code for creating functions with arguments

# Simple function to add two numbers
def sum_two_numbers(x, y):
        return x + y

# after this line x will hold the value 3
print(sum_two_numbers(1,2))
```

```
3
```

Scope of Variables The availability of a variable or identifier within the program during and after the execution is determined by the scope of a variable. There are two fundamental variable scopes in Python. 1. Global variables 2. Local variables

In [32]:

```
#Example code for defining variable scopes
# Global variable
x = 10
# Simple function to add two numbers
def sum_two_numbers(y):
    return x + y
# Call the function and print result
print(sum_two_numbers(10))
```

```
20
```

In [33]:

```
#Variable Length Arguments
# Example code for passing argumens as *args
# Simple function to loop through arguments and print them

def sample_function(*args):
    for a in args:
        print(a)

# Call the function
sample_function(1,2,3)
```

```
1
2
3
```

In [34]:

```python
#Example code for passing argumens as 2D
# Simple function to loop through arguments and print them

def sample_function(**args):
    for a in args:
        print(a, args[a])
# Call the function
sample_function(name='John', age=27)
```

```
name John
age 27
```

In [35]:

```python
#Lambda Function
def add(x, y):
    return x + y
print("FUNCTION ADD:\n",add(3,2))

add = lambda x, y : x + y
print("LAMBDA ADD :\n",add(3,2))
```

```
FUNCTION ADD:
 5
LAMBDA ADD :
 5
```

6.0 Machine Learning Python Packages

Machine Learning is a collection of algorithms and techniques used to create computational systems that learn from data in order to make predictions and inferences. AI Process Loop : [• Observe – identify patterns using the data • Plan – find all possible solutions • Optimize – find optimal solution from the list of possible solutions • Action – execute the optimal solution • Learn and Adapt – is the result giving expected result, if no adapt] ML Process Loop : [There are six major phases : • Business understanding • Data understanding • Data preparation • Modeling • Evaluation • Deployment] There is a rich number of open source libraries available to facilitate practical machine learning. These are mainly known as scientific Python libraries and are generally put to use when performing elementary machine learning tasks. At a high level we can divide these libraries into data analysis and core machine learning libraries based on their usage/purpose.

Data analysis packages: These are the sets of packages that provide us the mathematic and scientific functionalities that are essential to perform data preprocessing and transformation. Core Machine learning packages: These are the set of packages that provide us with all the necessary machine learning algorithms and functionalities that can be applied on a given dataset to extract the patterns.

6.1: Data Analysis Packages

There are four key packages that are most widely used for data analysis.

6.1.1: NumPy

6.1.2: SciPy

6.1.3: Matplotlib

6.1.4: Pandas

6.1.1 : NumPy

NumPy is the core library for scientific computing in Python. It provides a highperformance multidimensional array object, and tools for working with these array

In [36]:

```
#Example code for initializing NumPy array
import numpy as np
# Create a rank 1 array
a = np.array([0, 1, 2])
print(type(a))
# this will print the dimension of the array
print(a.shape)
print(a[0])
print(a[1])
print(a[2])
# Change an element of the array
a[0] = 5
print(a)
```

```
<class 'numpy.ndarray'>
(3,)
0
1
2
[5 1 2]
```

In [37]:

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

```
(2, 3)
[[0 1 2]
 [3 4 5]]
0 1 3
```

In [38]:

```
# Create a 3x3 array of all zeros
a = np.zeros((3,3))
print(a)
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

In [39]:

```
# Create a 2x2 array of all ones
b = np.ones((2,2))
print(b)
```

```
[[ 1.  1.]
 [ 1.  1.]]
```

In [40]:

```
# Create a 3x3 constant array
c = np.full((3,3), 7)
print(c)
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

In [41]:

```
# Create a 3x3 array filled with random values
d = np.random.random((3,3))
print(d)
```

```
[[ 0.94091236  0.39063363  0.18148016]
 [ 0.51461673  0.39967844  0.08791003]
 [ 0.0350918   0.86724652  0.45428522]]
```

In [42]:

```
# Create a 3x3 identity matrix
e = np.eye(3)
print(e)
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

In [43]:

```
# convert list to array
f = np.array([2, 3, 1, 0])
print(f)
#print([1,2,3])
```

```
[2 3 1 0]
```

In [44]:

```
# arange() will create arrays with regularly incrementing values
g = np.arange(2,10)
print(g)
```

```
[2 3 4 5 6 7 8 9]
```

In [45]:

```
# note mix of tuple and lists
h = np.array([[0,1,2.0],[0,0,0],(1+1j,3.,2.)])
print(h)
```

```
[[ 0.+0.j  1.+0.j  2.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j]
 [ 1.+1.j  3.+0.j  2.+0.j]]
```

In [46]:

```
# create an array of range with float data type
i = np.arange(1, 8, dtype=np.float)
print(i)
```

```
[ 1.  2.  3.  4.  5.  6.  7.]
```

In [47]:

```
# linspace() will create arrays with a specified number of items which are
# spaced equally between the specified beginning and end values
j = np.linspace(2., 4., 5)
print(j)
```

```
[ 2.   2.5  3.   3.5  4. ]
```

In [48]:

```
# indices() will create a set of arrays stacked as a one-higher
# dimensioned array, one per dimension with each representing variation
# in that dimension
k = np.indices((3,3))
print(k)
```

```
[[[0 0 0]
  [1 1 1]
  [2 2 2]]

 [[0 1 2]
  [0 1 2]
  [0 1 2]]]
```

In [49]:

```
#NumPy datatypes
# Let numpy choose the datatype
x = np.array([0, 1])
y = np.array([2.0, 3.0])
# Force a particular datatype
z = np.array([5, 6], dtype=np.int64)
print(x.dtype, y.dtype, z.dtype)
```

```
int32 float64 int64
```

In [50]:

```
# Basic slicing : The basic slice syntax is i: j: k,
# where i is the starting index,j is the stopping index,
# and k is the step and k is not equal to 0.
x = np.array([5, 6, 7, 8, 9])
print(x[1:7:2])
print(x[-2:5])
print(x[-1:1:-1])
```

```
[6 8]
[8 9]
[9 8 7]
```

In [51]:

```
#Boolean array indexing
a=np.array([[1,2], [3, 4], [5, 6]])
# Find the elements of a that are bigger than 2
print (a > 2)
# to get the actual value
print (a[a > 2])
```

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
```

In [52]:

```
import numpy as np
x=np.array([[1,2],[3,4],[5,6]])
y=np.array([[7,8],[9,10],[11,12]])
# Elementwise sum; both produce the array
print(x+y)
print(np.add(x, y))
# Elementwise difference; both produce the array
print(x-y)
print(np.subtract(x, y))
```

```
[[ 8 10]
 [12 14]
```

```
 [16 18]]
[[ 8 10]
 [12 14]
 [16 18]]
[[-6 -6]
 [-6 -6]
 [-6 -6]]
[[-6 -6]
 [-6 -6]
 [-6 -6]]
```

In [53]:

```
# Elementwise product; both produce the array
print(x*y)
print(np.multiply(x, y))
```

```
[[ 7 16]
 [27 40]
 [55 72]]
[[ 7 16]
 [27 40]
 [55 72]]
```

In [54]:

```
print(x/y)
print(np.divide(x, y))
```

```
[[ 0.14285714  0.25      ]
 [ 0.33333333  0.4       ]
 [ 0.45454545  0.5       ]]
[[ 0.14285714  0.25      ]
 [ 0.33333333  0.4       ]
 [ 0.45454545  0.5       ]]
```

In [55]:

```
print(np.sqrt(x))
```

```
[[ 1.          1.41421356]
 [ 1.73205081  2.        ]
 [ 2.23606798  2.44948974]]
```

In [56]:

```
x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])
a=np.array([9,10])
b=np.array([11, 12])
# Inner product of vectors; both produce 219
print(a.dot(b))
print(np.dot(a, b))
```

```
219
219
```

In [57]:

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(a))
print(np.dot(x, a))
```

```
[29 67]
[29 67]
```

In [58]:

```
# Matrix / matrix product; both produce the rank 2 array
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [59]:

```
# Sum function
x=np.array([[1,2],[3,4]])
# Compute sum of all elements
print (np.sum(x))
# Compute sum of each column
print (np.sum(x, axis=0))
# Compute sum of each row
print (np.sum(x, axis=1))
```

```
10
[4 6]
[3 7]
```

In [60]:

```
#Transpose function
x=np.array([[1,2], [3,4]])
print(x)
print(x.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

In [61]:

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

```
[1 2 3]
[1 2 3]
```

Broadcasting : Broadcasting enables arithmetic operations to be performed between different shaped arrays

In [62]:

```
# Broadcasting using NumPy
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
v = np.array([1, 0, 1])
# Add v to each row of a using broadcasting
b = a + v
print(b)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]]
```

In [63]:

```
# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column
x=np.array([[1,2,1], [3,4,1]])
print("X.T:\n", x.T)
```

```
print("X.T:\n",x.T)
w= np.array([1,1])
print("W:\n",w)

print("Final :\n",(x.T + w).T)
```

```
X.T:
 [[1 3]
 [2 4]
 [1 1]]
W:
 [1 1]
Final :
 [[2 3 2]
 [4 5 2]]
```

In [64]:

```
# Multiply a matrix by a constant:
# x has shape (2, 3). Numpy treats scalars as arrays of shape ();
# these can be broadcast together to shape (2, 3)
print(x * 2)
```

```
[[2 4 2]
 [6 8 2]]
```

6.1.2 : PANDAS

In [65]:

```
import pandas as pd
```

Pandas are an open source Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. Pandas are well suited for tabular data with heterogeneously typed columns, as in an SQL table or Excel spreadsheet

Data Structures:

Pandas introduces two new data structures to Python [both of which are built on top of NumPy (this means it’s fast).]

1. Series
2. DataFrame

1.Series : This is a one-dimensional object similar to column in a spreadsheet or SQL table. By default each item will be assigned an index label from 0 to N.

In [66]:

```
#Creating a pandas series
# creating a series by passing a list of values, and a custom index label.
#Note that the labeled index reference for each row and it can have
#duplicate values
s = pd.Series([1,2,3,np.nan,5,6], index=['A','B','C','D','E','F'])
print(s)
```

```
A    1.0
B    2.0
C    3.0
D    NaN
E    5.0
F    6.0
dtype: float64
```

2.DataFrame : It is a two-dimensional object similar to a spreadsheet or an SQL table. This is the most commonly used pandas object

In [67]:

```
#Creating a pandas dataframe
data = {'Gender': ['F', 'M', 'M'],'Emp_ID': ['E01', 'E02',
'E03'], 'Age': [25, 27, 25]}
# We want the order the columns, so lets specify in columns parameter
df = pd.DataFrame(data, columns=['Emp_ID','Gender', 'Age'])
df
```

Out[67]:

	Emp_ID	Gender	Age
0	E01	F	25
1	E02	M	27
2	E03	M	25

Reading and Writing Data

In [69]:

```
#Reading / writing data from csv, text, Excel

# Reading frome csv
df=pd.read_csv('C:/Users/thyagaragu/Desktop/Data/PF/CSV/CHS2.csv')
print("READ CSV:\n",df)
#Writing to csv
df.to_csv('C:/Users/thyagaragu/Desktop/Data/PF/CSV/CHS0.csv', index=False)
print("WRITE CSV:\n",df)

# Reading from text Files
df=pd.read_csv('C:/Users/thyagaragu/Desktop/Data/PF/TEXT/ex.txt', sep='\t') # from text file
print("READ TXT:\n",df)
#Writing to text Files
df.to_csv('C:/Users/thyagaragu/Desktop/Data/PF/TEXT/ex0.txt', sep='\t', index=False)
print("WRITE TXT:\n",df)

#Reading from Excel File
df=pd.read_excel('C:/Users/thyagaragu/Desktop/Data/PF/EXCEL/Heart_Patient.xlsx','Sheet1') # from E
xcel
print("READ EXCEL:\n",df)

#Writing to Excel File
df.to_excel('C:/Users/thyagaragu/Desktop/Data/PF/EXCEL/Heart_Patient0.xlsx',sheet_name='Sheet1', i
ndex = False)
print("WRITE EXCEL:\n",df)

# reading from multiple sheets of same Excel into different dataframes
xlsx = pd.ExcelFile('C:/Users/thyagaragu/Desktop/Data/PF/EXCEL/Heart_Patient.xlsx')
sheet1_df = pd.read_excel(xlsx, 'Sheet1')
print("EXCEL SHEET1:\n",sheet1_df)
sheet2_df = pd.read_excel(xlsx, 'Sheet2')
print("EXCEL SHEET2:\n",sheet2_df)

# writing
# index = False parameter will not write the index values, default is True
```

```
READ CSV:
     Gender  Height (in)
0     Male           72
1     Male           72
2   Female           63
3   Female           62
4   Female           62
5     Male           73
6   Female           64
7   Female           63
8   Female           67
9     Male           71
10    Male           72
11  Female           63
12    Male           71
13  Female           67
```

```
14  Female           62
15  Female           63
16    Male           66
17  Female           60
18  Female           68
19  Female           65
20  Female           64
WRITE CSV:
     Gender  Height (in)
0     Male           72
1     Male           72
2   Female           63
3   Female           62
4   Female           62
5     Male           73
6   Female           64
7   Female           63
8   Female           67
9     Male           71
10    Male           72
11  Female           63
12    Male           71
13  Female           67
14  Female           62
15  Female           63
16    Male           66
17  Female           60
18  Female           68
19  Female           65
20  Female           64
READ TXT:
 Empty DataFrame
Columns: [ 0.00632  18.00   2.310  0  0.5380   ]
Index: []
WRITE TXT:
 Empty DataFrame
Columns: [ 0.00632  18.00   2.310  0  0.5380   ]
Index: []
READ EXCEL:
     ATS     SSM   HBP   FH  ADM     O
0   Yes  Abnorm  High  Yes  Yes  0.94
1   Yes  Abnorm  High  Yes   No  0.93
2   Yes  Abnorm  High   No  Yes  0.92
3   Yes  Abnorm  High   No   No  0.91
4   Yes  Abnorm  Norm  Yes   No  0.84
5   Yes  Abnorm  Norm  Yes   No  0.82
6   Yes  Abnorm  Norm   No  Yes  0.80
7   Yes  Abnorm  Norm   No   No  0.78
8   Yes    Norm  High  Yes  Yes  0.91
9   Yes    Norm  High  Yes   No  0.91
10  Yes    Norm  High   No  Yes  0.90
11  Yes    Norm  High   No   No  0.89
12  Yes    Norm  Norm  Yes  Yes  0.80
13  Yes    Norm  Norm  Yes   No  0.78
14  Yes    Norm  Norm   No  Yes  0.76
15   No  Abnorm  High  Yes  Yes  0.79
16   No  Abnorm  High  Yes   No  0.77
WRITE EXCEL:
     ATS     SSM   HBP   FH  ADM     O
0   Yes  Abnorm  High  Yes  Yes  0.94
1   Yes  Abnorm  High  Yes   No  0.93
2   Yes  Abnorm  High   No  Yes  0.92
3   Yes  Abnorm  High   No   No  0.91
4   Yes  Abnorm  Norm  Yes   No  0.84
5   Yes  Abnorm  Norm  Yes   No  0.82
6   Yes  Abnorm  Norm   No  Yes  0.80
7   Yes  Abnorm  Norm   No   No  0.78
8   Yes    Norm  High  Yes  Yes  0.91
9   Yes    Norm  High  Yes   No  0.91
10  Yes    Norm  High   No  Yes  0.90
11  Yes    Norm  High   No   No  0.89
12  Yes    Norm  Norm  Yes  Yes  0.80
13  Yes    Norm  Norm  Yes   No  0.78
14  Yes    Norm  Norm   No  Yes  0.76
15   No  Abnorm  High  Yes  Yes  0.79
16   No  Abnorm  High  Yes   No  0.77
EXCEL SHEET1:
```

```
EXCEL SHEET1:
     ATS      SSM   HBP   FH  ADM     O
0    Yes  Abnorm  High  Yes  Yes  0.94
1    Yes  Abnorm  High  Yes   No  0.93
2    Yes  Abnorm  High   No  Yes  0.92
3    Yes  Abnorm  High   No   No  0.91
4    Yes  Abnorm  Norm  Yes   No  0.84
5    Yes  Abnorm  Norm  Yes   No  0.82
6    Yes  Abnorm  Norm   No  Yes  0.80
7    Yes  Abnorm  Norm   No   No  0.78
8    Yes    Norm  High  Yes  Yes  0.91
9    Yes    Norm  High  Yes   No  0.91
10   Yes    Norm  High   No  Yes  0.90
11   Yes    Norm  High   No   No  0.89
12   Yes    Norm  Norm  Yes  Yes  0.80
13   Yes    Norm  Norm  Yes   No  0.78
14   Yes    Norm  Norm   No  Yes  0.76
15    No  Abnorm  High  Yes  Yes  0.79
16    No  Abnorm  High  Yes   No  0.77
EXCEL SHEET2:
     ATS      SSM   HBP
0    Yes  Abnorm  High
1    Yes  Abnorm  High
2    Yes  Abnorm  High
3    Yes  Abnorm  High
4    Yes  Abnorm  Norm
5    Yes  Abnorm  Norm
6    Yes  Abnorm  Norm
7    Yes  Abnorm  Norm
8    Yes    Norm  High
9    Yes    Norm  High
10   Yes    Norm  High
11   Yes    Norm  High
12   Yes    Norm  Norm
13   Yes    Norm  Norm
14   Yes    Norm  Norm
15    No  Abnorm  High
16    No  Abnorm  High
```

Loading Data From URL

In [70]:

```python
# Load CSV from URL using NumPy
from numpy import loadtxt
from urllib.request import urlopen
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'
raw_data = pd.read_csv(urlopen(url))
print(raw_data.shape)
print(raw_data.head())
```

```
(767, 9)
   6  148  72  35    0  33.6  0.627  50  1
0  1   85  66  29    0  26.6  0.351  31  0
1  8  183  64   0    0  23.3  0.672  32  1
2  1   89  66  23   94  28.1  0.167  21  0
3  0  137  40  35  168  43.1  2.288  33  1
4  5  116  74   0    0  25.6  0.201  30  0
```

Loading Data From Library

In [71]:

```python
from sklearn.datasets import load_iris
import numpy as np
iris=load_iris()
#iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
#X.head()
print("Shape:\n",X.shape)
print("Head:\n",X.head(5))
```

```
print("Head:\n",X.head(5))
print("Tail:\n",X.tail(5))
```

```
Shape:
 (150, 4)
Head:
    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
Tail:
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8
```

Basic Statistics Summary

Pandas has some built-in functions to help us to get better understanding of data using basic statistical summary methods describe()- will returns the quick stats such as count, mean, std (standard deviation), min, first quartile, median, third quartile, max on each column of the dataframe

In [72]:

```
df = pd.DataFrame(iris.data)
df.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
df.describe()
```

Out[72]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

cov() - Covariance indicates how two variables are related. A positive covariance means the variables are positively related, while a negative covariance means the variables are inversely related. Drawback of covariance is that it does not tell you the degree of positive or negative relation

In [73]:

```
df.cov()
```

Out[73]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
Sepal_Length	0.685694	-0.039268	1.273682	0.516904
Sepal_Width	-0.039268	0.188004	-0.321713	-0.117981
Petal_Length	1.273682	-0.321713	3.113179	1.296387
Petal_Width	0.516904	-0.117981	1.296387	0.582414

corr() - Correlation is another way to determine how two variables are related. In addition to telling you whether variables are positively or inversely related, correlation also tells you the degree to which the variables tend to move together. When you say that two items correlate, you are saying that the change in one item effects a change in another item. You will always talk about correlation as a range between -1 and 1. In the below example code, petal length is 87% positively related to sepal length that means a change in petal length results in a positive 87% change to sepal lenth and vice versa.

In [74]:

```
df.corr()
```

Out[74]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
Sepal_Length	1.000000	-0.109369	0.871754	0.817954
Sepal_Width	-0.109369	1.000000	-0.420516	-0.356544
Petal_Length	0.871754	-0.420516	1.000000	0.962757
Petal_Width	0.817954	-0.356544	0.962757	1.000000

Grouping

Grouping involves one or more of the following steps: • Splitting the data into groups based on some criteria, • Applying a function to each group independently, • Combining the results into a data structure

In [75]:

```
#Grouping operation
df = pd.DataFrame({'Name' : ['jack', 'jane', 'jack', 'jane', 'jack', 'jane',
'jack', 'jane'],'State' : ['SFO', 'SFO', 'NYK', 'CA', 'NYK', 'NYK','SFO', 'CA'],
'Grade':['A','A','B','A','C','B','C','A'],
'Age' : np.random.uniform(24, 50, size=8),
'Salary' : np.random.uniform(3000, 5000, size=8),})
# Note that the columns are ordered automatically in their alphabetic order
print(df)
# for custom order please use below code
# df = pd.DataFrame(data, columns = ['Name', 'State', 'Age','Salary'])
# Find max age and salary by Name / State
# with groupby, we can use all aggregate functions such as min, max, mean,
#count, cumsum
df.groupby(['Name','State']).max()
```

```
         Age Grade  Name       Salary State
0  39.977721     A  jack  4438.213129   SFO
1  34.659439     A  jane  4509.536654   SFO
2  29.689284     B  jack  3666.848792   NYK
3  37.916489     A  jane  3017.508677    CA
4  40.281893     C  jack  3089.576097   NYK
5  26.757244     B  jane  3437.291535   NYK
6  33.918859     C  jack  3201.872703   SFO
7  27.322148     A  jane  3183.482365    CA
```

Out[75]:

		Age	Grade	Salary
Name	**State**			
jack	**NYK**	40.281893	C	3666.848792
	SFO	39.977721	C	4438.213129
jane	**CA**	37.916489	A	3183.482365
	NYK	26.757244	B	3437.291535
	SFO	34.659439	A	4509.536654

7.0 Matplotlib

In [76]:

```
import matplotlib.pyplot as plt
```

Matplotlib is a numerical mathematics extension NumPy and a great package to view or present data in a pictorial or graphical format. It enables analysts and decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns. There are two broad ways of using pyplot.

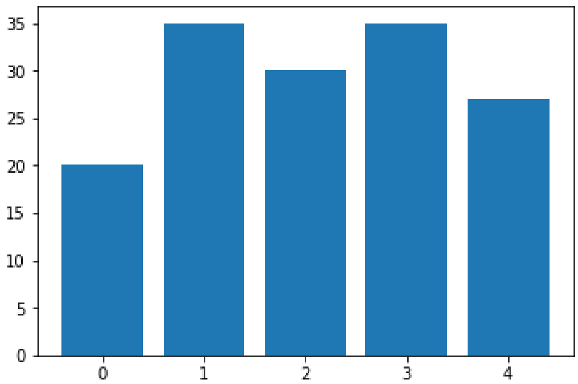
1. Using Global Functions

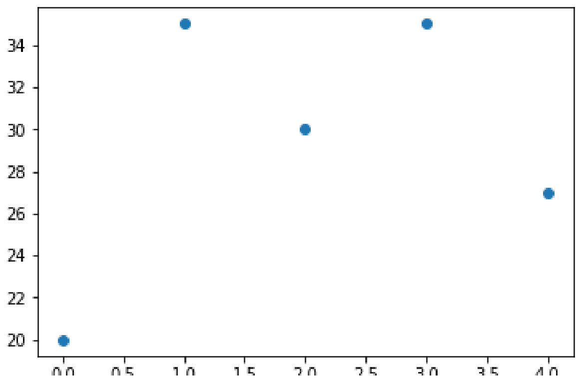
The most common and easy approach is by using global functions to build and display a global figure using matplotlib as a global state machine. Some of the most commonly used charts.

```
# plt.scatter - makes a scatter plot
# plt.bar - creates a bar chart
# plt.boxplot - makes a box and whisker plot
# plt.hist - makes a histogram
# plt.plot - creates a line plot
```

In [77]:

```
# Creating plot on variables
# simple bar and scatter plot

x = np.arange(5) # assume there are 5 students
y = (20, 35, 30, 35, 27) # their test scores
plt.bar(x,y) # Bar plot

# need to close the figure using show() or close(), if not closed
# any follow plot commands will use same figure.

plt.show() # Try commenting this an run
plt.scatter(x,y) # scatter plot
plt.show()
```

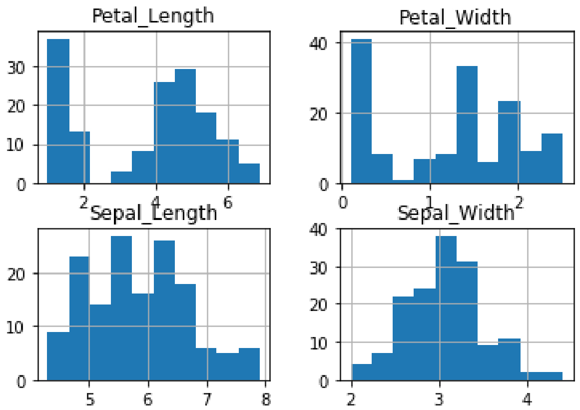




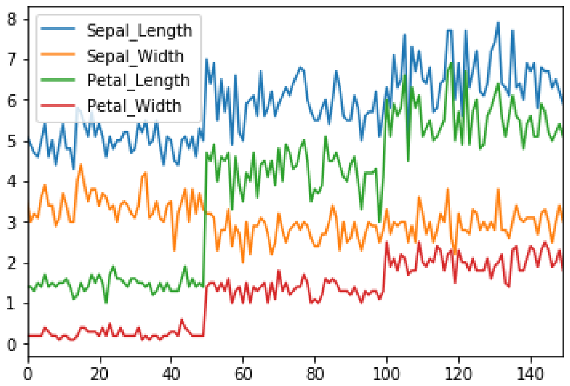
In [78]:

```
# Read sample data
df = pd.DataFrame(iris.data)
df.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
print("Histogram:\n")
df.hist() # Histogram
plt.show()
print("Line Graph:\n")
df.plot()  # Line Graph
plt.show()
print("Box Plot:\n")
df.boxplot()  # Box plot
plt.show()
```

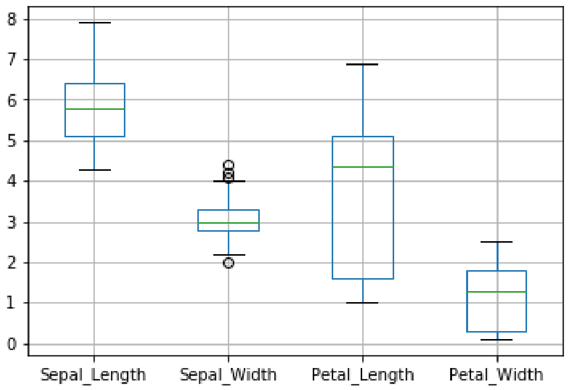
```
Histogram:
```




```
Line Graph:
```



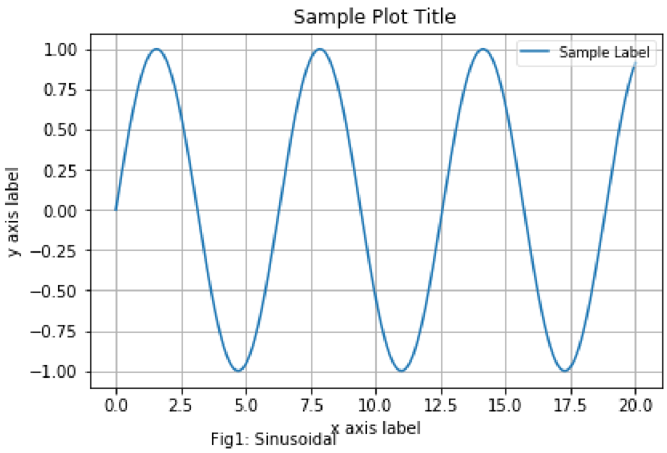

```
Box Plot:
```



Customizing Labels

In [79]:

```
#Customize labels

# generate sample data
x = np.linspace(0, 20, 1000) #100 evenly-spaced values from 0 to 50
y = np.sin(x)

# customize axis labels
plt.plot(x, y, label = 'Sample Label')
plt.title('Sample Plot Title') # chart title
plt.xlabel('x axis label') # x axis title
plt.ylabel('y axis label') # y axis title
plt.grid(True) # show gridlines

# add footnote
plt.figtext(0.5, 0.01, 'Fig1: Sinusoidal', ha='right', va='bottom')

# add legend, location pick the best automatically
plt.legend(loc='best', framealpha=0.5, prop={'size':'small'})

# tight_layout() can take keyword arguments of pad, w_pad and h_pad.
# these control the extra padding around the figure border and between
#subplots. The pads are specified in fraction of fontsize.
plt.tight_layout(pad=1)

# Saving chart to a file
#plt.savefig('filename.png')
#plt.close()
# Close the current window to allow new plot creation on
#separate window / axis, alternatively we can use show()
plt.show()
```



8.0 Machine Learning Libraries

In [80]:

```
# Python version
import sys
print('Python: {}'.format(sys.version))
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
# numpy
import numpy
print('numpy: {}'.format(numpy.__version__))
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# pandas
```

```
# pandas
import pandas
print('pandas: {}'.format(pandas.__version__))
# scikit-learn
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
import seaborn
print('seaborn: {}'.format(seaborn.__version__))
import pgmpy
print('pgmpy: {}'.format(pgmpy.__name__))
import urllib
print('urlib: {}'.format(urllib.__name__))
import csv
print('csv: {}'.format(csv.__version__))
```

```
Python: 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)]
scipy: 0.19.1
numpy: 1.13.3
matplotlib: 2.1.0
pandas: 0.20.3
sklearn: 0.19.1
seaborn: 0.8.0
pgmpy: pgmpy
urlib: urllib
csv: 1.0
```

8.1 Scipy

SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

In [81]:

```
import numpy as np
print(np.linspace(1., 4., 6))
```

```
[ 1.   1.6  2.2  2.8  3.4  4. ]
```

In [82]:

```
#K-Means Implementation in SciPy
from scipy.cluster.vq import kmeans,vq,whiten
from numpy import vstack,array
from numpy.random import rand

# data generation with three features
data = vstack((rand(100,3) + array([.5,.5,.5]),rand(100,3)))
#print(data)
# whitening of data for normalizing
data = whiten(data)
#print(data)
# computing K-Means with K = 3 (2 clusters)
centroids,_ = kmeans(data,3)
print("Centroids:\n",centroids)
# assign each sample to a cluster
clx,_ = vq(data,centroids)
print("Cluster:\n",clx)
```

```
Centroids:
 [[ 1.12229616  0.96730452  1.15748527]
 [ 1.91765881  2.42546485  1.32970073]
 [ 2.80961316  2.57661677  2.82922036]]
Cluster:
 [2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 0 2 2
 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 1 1 1 1 2 2
 1 1 1 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0]
```

In [83]:

```
#Fast Fourier Transform
#Importing the fft and inverse fft functions from fftpackage
from scipy.fftpack import fft

#create an array with random n numbers
x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])

#Applying the fft function
y = fft(x)
print("FFT :\n",y)

from scipy.fftpack import ifft
yinv = ifft(y)
print("FFT Inverse:\n",yinv)
```

```
FFT :
 [ 4.50000000+0.j          2.08155948-1.65109876j -1.83155948+1.60822041j
 -1.83155948-1.60822041j  2.08155948+1.65109876j]
FFT Inverse:
 [ 1.0+0.j  2.0+0.j  1.0+0.j -1.0+0.j  1.5+0.j]
```

In [84]:

```
#Discrete Cosine Transform
from scipy.fftpack import dct
print ("DCT:\n",dct(np.array([4., 3., 5., 10., 5., 3.])))

#Inverse Discrete Cosine Transform
from scipy.fftpack import idct
print("IDCT:\n",idct(np.array([4., 3., 5., 10., 5., 3.])))
```

```
DCT:
 [ 60.          -3.48476592 -13.85640646  11.3137085    6.          -6.31319305]
IDCT:
 [ 39.15085889 -20.14213562  -6.45392043   7.13341236   8.14213562
  -3.83035081]
```

SciPy - Integrate

The general form of quad is scipy.integrate.quad(f, a, b), Where ‘f’ is the name of the function to be integrated. Whereas, ‘a’ and ‘b’ are the lower and upper limits, respectively. Let us see an example of the Gaussian function, integrated over a range of 0 and 1.

$$f(x) = (e^x)^2$$

∫f(x)dx

In [85]:

```
# Single Integration
import scipy.integrate
from numpy import exp
f= lambda x:exp(-x**2)
i = scipy.integrate.quad(f, 0, 1)
print(i)
```

```
(0.7468241328124271, 8.291413475940725e-15)
```

Linear Algebra

x + 3y + 5z = 10

2x + 5y + z = 8

2x + 3y + 8z = 3

In [86]:

```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy arrays
a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
b = np.array([10, 8, 3])

#Passing the values to the solve function
X = linalg.solve(a, b)

#printing the result array
print (X)
```

```
[-9.28  5.16  0.76]
```

Finding a Determinant

In [87]:

```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy array
A = np.array([[1,2],[3,4]])

#Passing the values to the det function
x = linalg.det(A)

#printing the result
print (x)
```

```
-2.0
```

Eigenvalues and Eigenvectors

In [88]:

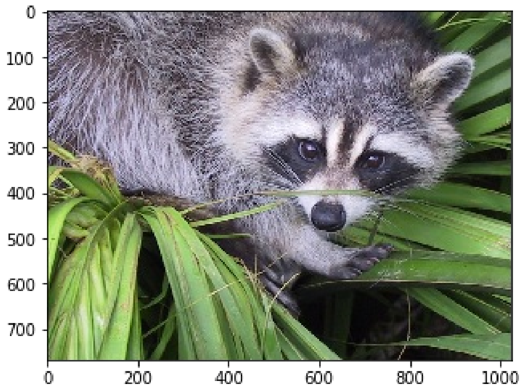
```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy array
A = np.array([[1,2],[3,4]])

#Passing the values to the eig function
l, v = linalg.eig(A)

#printing the result for eigen values
print ("Eigen Values :\n",l)

#printing the result for eigen vectors
print ("Eigen Vectors:\n",v)
```

```
Eigen Values :
 [-0.37228132+0.j  5.37228132+0.j]
Eigen Vectors:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Image Processing

In [89]:

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f) # uses the Image module (PIL)

import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```



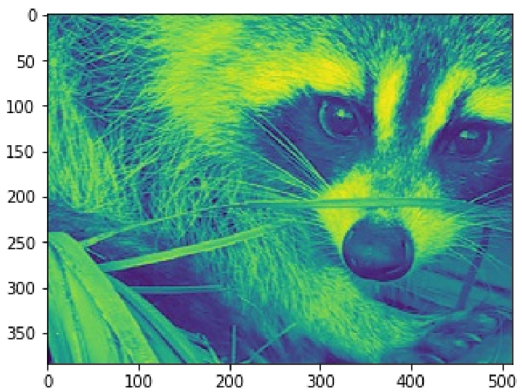
In [90]:

```
# Statistical Information of the image
from scipy import misc
face = misc.face(gray = False)
print(face.mean(), face.max(), face.min())
```

```
110.162743886 255 0
```

In [91]:

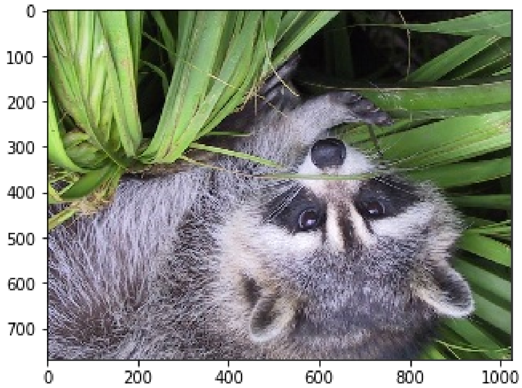
```
# Cropping
from scipy import misc
face = misc.face(gray = True)
lx,ly = face.shape
# Cropping
crop_face = face[lx//4 : -lx//4 , ly//4 : -ly//4]
import matplotlib.pyplot as plt
plt.imshow(crop_face)
plt.show()
```



In [92]:

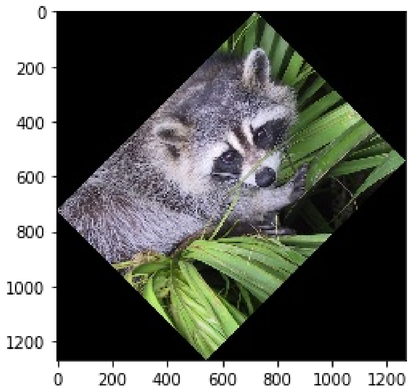
```
# up <-> down flip
from scipy import misc
```

```
face = misc.face()
flip_ud_face = np.flipud(face)

import matplotlib.pyplot as plt
plt.imshow(flip_ud_face)
plt.show()
```



In [93]:

```
# rotation
from scipy import misc,ndimage
face = misc.face()
rotate_face = ndimage.rotate(face, 45)

import matplotlib.pyplot as plt
plt.imshow(rotate_face)
plt.show()
```

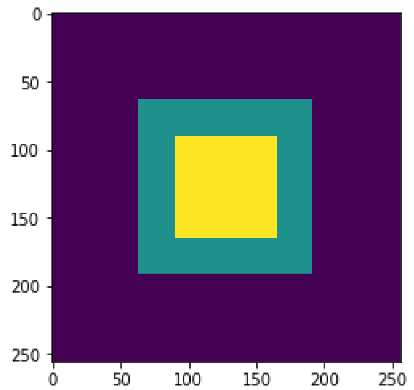


In [94]:

```
# Blurring
from scipy import misc
face = misc.face()
blurred_face = ndimage.gaussian_filter(face, sigma=3)
import matplotlib.pyplot as plt
plt.imshow(blurred_face)
plt.show()
```

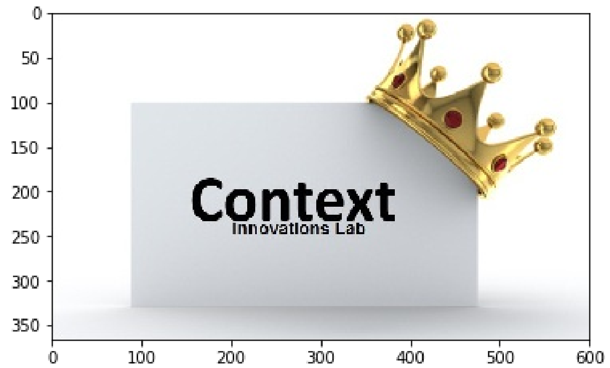
In [95]:

```
# Edge Detection
import scipy.ndimage as nd
import numpy as np

im = np.zeros((256, 256))
im[64:-64, 64:-64] = 1
im[90:-90,90:-90] = 2
im = ndimage.gaussian_filter(im, 0)

import matplotlib.pyplot as plt
plt.imshow(im)
plt.show()
```



In [97]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img=mpimg.imread('C:/Users/thyagaragu/Desktop/Data/Image/C1.jpg')
#print(img)
plt.imshow(img)
plt.show()
```



8.2 sklearn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes: NumPy: Base n-dimensional array package SciPy: Fundamental library for scientific computing Matplotlib: Comprehensive 2D/3D plotting IPython: Enhanced interactive console Sympy: Symbolic mathematics Pandas: Data structures and analysis

In [98]:

```
import sklearn
```

Scikit Learn Loading Dataset

In [99]:

```
from sklearn import datasets
```

In [100]:

```
# Data sets available in sklearn
iris= datasets.load_iris()
houseprice = datasets.load_boston()
diabetes = datasets.load_diabetes()
digits = datasets.load_digits()
linerud= datasets.load_linnerud()   #Fitness Club Data Set
wine = datasets.load_wine()
breastcancer = datasets.load_breast_cancer()
```

In [101]:

```
print(digits.target_names)
print(digits.data[0])
```

```
[0 1 2 3 4 5 6 7 8 9]
[  0.   0.   5.  13.   9.   1.   0.   0.   0.   0.  13.  15.  10.  15.   5.
   0.   0.   3.  15.   2.   0.  11.   8.   0.   0.   4.  12.   0.   0.   8.
   8.   0.   0.   5.   8.   0.   0.   9.   8.   0.   0.   4.  11.   0.   1.
  12.   7.   0.   0.   2.  14.   5.  10.  12.   0.   0.   0.   0.   6.  13.
  10.   0.   0.   0.]
```

In [102]:

```
print(houseprice.feature_names)
print(houseprice.data[0])
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
[  6.32000000e-03   1.80000000e+01   2.31000000e+00   0.00000000e+00
   5.38000000e-01   6.57500000e+00   6.52000000e+01   4.09000000e+00
   1.00000000e+00   2.96000000e+02   1.53000000e+01   3.96900000e+02
   4.98000000e+00]
```

In [103]:

```
print(diabetes.feature_names)
print(diabetes.data[0])
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235  -0.03482076
 -0.04340085 -0.00259226  0.01990842 -0.01764613]
```

In [104]:

```
print(linerud.data[0])
print(linerud.feature_names)
```

```
[   5.  162.   60.]
['Chins', 'Situps', 'Jumps']
```

In [105]:

```
print(wine.feature_names)
print(wine.data[0])
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
```

```
'od280/od315_of_diluted_wines', 'proline']
[  1.42300000e+01   1.71000000e+00   2.43000000e+00   1.56000000e+01
   1.27000000e+02   2.80000000e+00   3.06000000e+00   2.80000000e-01
   2.29000000e+00   5.64000000e+00   1.04000000e+00   3.92000000e+00
   1.06500000e+03]
```

In [106]:

```
print(breastcancer.feature_names)
print(breastcancer.data[0])
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
[  1.79900000e+01   1.03800000e+01   1.22800000e+02   1.00100000e+03
   1.18400000e-01   2.77600000e-01   3.00100000e-01   1.47100000e-01
   2.41900000e-01   7.87100000e-02   1.09500000e+00   9.05300000e-01
   8.58900000e+00   1.53400000e+02   6.39900000e-03   4.90400000e-02
   5.37300000e-02   1.58700000e-02   3.00300000e-02   6.19300000e-03
   2.53800000e+01   1.73300000e+01   1.84600000e+02   2.01900000e+03
   1.62200000e-01   6.65600000e-01   7.11900000e-01   2.65400000e-01
   4.60100000e-01   1.18900000e-01]
```

In [107]:

```
# Print shape of data to confirm data is loaded
print("IRIS:\n",iris.data.shape)
print("HOUSEPRICE:\n",houseprice.data.shape)
print("DIABETES:\n",diabetes.data.shape)
print("DIGITS:\n",digits.data.shape)
print("LINERUD:\n",linerud.data.shape)
print("WINE:\n",wine.data.shape)
print("BREASTCANCER:\n",breastcancer.data.shape)
```

```
IRIS:
 (150, 4)
HOUSEPRICE:
 (506, 13)
DIABETES:
 (442, 10)
DIGITS:
 (1797, 64)
LINERUD:
 (20, 3)
WINE:
 (178, 13)
BREASTCANCER:
 (569, 30)
```

In [108]:

```
# see what's available in iris:
iris.keys()
print("IRIS KEYS:\n",iris.keys())
n_samples, n_features = iris.data.shape
print ("IRIS # SAMPLES:\n",n_samples)
print ("IRIS # FEATURES:\n",n_features)
print ("IRIS FIRST FEW ROWS:\n",iris.data[0:10])
print("IRIS TARGETS NAMES",iris.target_names)
print("IRIS FEATURE NAMES",iris.feature_names)
print("IRIS TARGET",iris.target)
print("IRIS DESCR",iris.DESCR)
iris_X = iris.data
iris_y = iris.target
np.unique(iris_y)
```

```
IRIS KEYS:
```

```
IRIS KEYS:
 dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
IRIS # SAMPLES:
 150
IRIS # FEATURES:
 4
IRIS FIRST FEW ROWS:
 [[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]]
IRIS TARGETS NAMES ['setosa' 'versicolor' 'virginica']
IRIS FEATURE NAMES ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm
)']
IRIS TARGET [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
IRIS DESCR Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
```

```
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Out[108]:

```
array([0, 1, 2])
```

In [109]:

```python
# Split iris data in train and test data
# A random permutation, to split the data randomly
np.random.seed(0)
indices = np.random.permutation(len(iris_X))
iris_X_train = iris_X[indices[:-10]]
iris_y_train = iris_y[indices[:-10]]
iris_X_test = iris_X[indices[-10:]]
iris_y_test = iris_y[indices[-10:]]
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform')
print("Predicted :\n",knn.predict(iris_X_test))
print("Actual:\n",iris_y_test)
```

```
Predicted :
 [1 2 1 0 0 0 2 1 2 0]
Actual:
 [1 1 1 0 0 0 2 1 2 0]
```

Linear regression

LinearRegression, in its simplest form, fits a linear model to the data set by adjusting a set of parameters in order to make the sum of the squared residuals of the model as small as possible

In [110]:

```python
diabetes = datasets.load_diabetes()
diabetes_X_train = diabetes.data[:-20]
diabetes_X_test = diabetes.data[-20:]
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

In [111]:

```python
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
print("Regression Coef:\n",regr.coef_)
print("Mean:\n",np.mean((regr.predict(diabetes_X_test)-diabetes_y_test)**2))
# Explained variance score: 1 is perfect prediction
# and 0 means that there is no linear relationship
# between X and y.
regr.score(diabetes_X_test, diabetes_y_test)
```

```
Regression Coef:
 [  3.03499549e-01  -2.37639315e+02   5.10530605e+02   3.27736980e+02
  -8.14131709e+02   4.92814588e+02   1.02848452e+02   1.84606489e+02
   7.43519617e+02   7.60951722e+01]
Mean:
```

```
Mean:
 2004.56760269
```

Out[111]:

```
0.58507530226905735
```

In [112]:

```
# Sample Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        50
          1       1.00      1.00      1.00        50
          2       1.00      1.00      1.00        50

avg / total       1.00      1.00      1.00       150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

8.3 pgmpy

Probabilistic Graphical Models using pgmpy

Probabilistic Graphical Model is a way of compactly representing Joint Probability distribution over random variables using the independence conditions of the variables

In [113]:

```
import pgmpy
```

In [114]:

```
# Generate data
import numpy as np
import pandas as pd

raw_data = np.array([0] * 30 + [1] * 70) # Representing heads by 0 and tails by 1
data = pd.DataFrame(raw_data, columns=['coin'])
print(data[25:35])
```

```
    coin
25     0
26     0
27     0
```

```
28    0
29    0
30    1
31    1
32    1
33    1
34    1
```

In [115]:

```
# Defining the Bayesian Model
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

model = BayesianModel()
model.add_node('coin')

# Fitting the data to the model using Maximum Likelihood Estimator
model.fit(data, estimator=MaximumLikelihoodEstimator)
print(model.get_cpds('coin'))
```

coin(0)	0.3
coin(1)	0.7

In [116]:

```
# Fitting the data to the model using Bayesian Estimator with Dirichlet prior with equal pseudo co
unts.
model.fit(data, estimator=BayesianEstimator, prior_type='dirichlet', pseudo_counts={'coin': [50, 50
]})
print(model.get_cpds('coin'))
```

```
WARNING:root:Replacing existing CPD for coin
```

coin(0)	0.4
coin(1)	0.6

We can see that we get the results as expected. In the maximum likelihood case we got the probability just based on the data where as in the bayesian case we had a prior of P(H) = 0.5 and P(T) = 0.5 , therefore with 30% heads and 70% tails in the data we got a posterior of P(H) = 0.4 and P(T) = 0.6 . Similarly we can learn in case of more complex model. Let's take an example of the student model and compare the results in case of Maximum Likelihood estimator and Bayesian Estimator.

In [117]:

```
# Generating radom data with each variable have 2 states and equal probabilities for each state
import numpy as np
import pandas as pd

raw_data = np.random.randint(low=0, high=2, size=(1000, 5))
data = pd.DataFrame(raw_data, columns=['D', 'I', 'G', 'L', 'S'])
print(data[100: 111])
```

```
     D  I  G  L  S
100  1  1  1  0  0
101  1  1  1  1  0
102  1  0  0  1  0
103  1  0  1  1  0
104  1  1  0  1  0
105  0  0  0  1  0
106  0  1  0  0  0
107  1  0  1  0  0
108  0  0  1  1  1
109  0  0  1  1  0
110  1  1  0  1  1
```

In [118]:

```python
# Defining the model
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

model = BayesianModel([('D', 'G'), ('I', 'G'), ('I', 'S'), ('G', 'L')])

# Learing CPDs using Maximum Likelihood Estimators
model.fit(data, estimator=MaximumLikelihoodEstimator)
for cpd in model.get_cpds():
    print("CPD of {variable}:".format(variable=cpd.variable))
    print(cpd)
```

CPD of D:

D(0)	0.48
D(1)	0.52

CPD of G:

D	D(0)	D(0)	D(1)	D(1)
I	I(0)	I(1)	I(0)	I(1)
G(0)	0.4618320610687023	0.46788990825688076	0.5	0.44402985074626866
G(1)	0.5381679389312977	0.5321100917431193	0.5	0.5559701492537313

CPD of I:

I(0)	0.514
I(1)	0.486

CPD of L:

G	G(0)	G(1)
L(0)	0.45726495726495725	0.4981203007518797
L(1)	0.5427350427350427	0.5018796992481203

CPD of S:

I	I(0)	I(1)
S(0)	0.5038910505836576	0.5164609053497943
S(1)	0.4961089494163424	0.4835390946502058

As the data was randomly generated with equal probabilities for each state we can see here that all the probability values are close to 0.5 which we expected. Now coming to the Bayesian Estimator:

Python Collection Counter

In [119]:

```python
# Learning with Bayesian Estimator using dirichlet prior for each variable.

pseudo_counts = {'D': [300, 700], 'I': [500, 500], 'G': [800, 200], 'L': [500, 500], 'S': [400, 600]
}
model.fit(data, estimator=BayesianEstimator, prior_type='dirichlet', pseudo_counts=pseudo_counts)
for cpd in model.get_cpds():
    print("CPD of {variable}:".format(variable=cpd.variable))
    print(cpd)
```

```
WARNING:root:Replacing existing CPD for D
WARNING:root:Replacing existing CPD for G
```

```
WARNING:root:Replacing existing CPD for G
WARNING:root:Replacing existing CPD for I
WARNING:root:Replacing existing CPD for S
WARNING:root:Replacing existing CPD for L
```

CPD of D:

D(0)	0.39
D(1)	0.61

CPD of G:

D	D(0)	D(0)	D(1)	D(1)
I	I(0)	I(1)	I(0)	I(1)
G(0)	0.7297939778129953	0.7405582922824302	0.7396166134185304	0.7247634069400631
G(1)	0.27020602218700474	0.2594417077175698	0.26038338658146964	0.2752365930599369

CPD of I:

I(0)	0.507
I(1)	0.493

CPD of L:

G	G(0)	G(1)
L(0)	0.48637602179836514	0.4993472584856397
L(1)	0.5136239782016349	0.5006527415143603

CPD of S:

I	I(0)	I(1)
S(0)	0.43527080581241745	0.43808882907133245
S(1)	0.5647291941875826	0.5619111709286676

Since the data was randomly generated with equal probabilities for each state, the data tries to bring the posterior probabilities close to 0.5. But because of the prior we will get the values in between the prior and 0.5.

In [120]:

```
# Tally occurrences of words in a list
from collections import Counter
cnt = Counter()
for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
        cnt[word] += 1

print(cnt)
```

```
Counter({'blue': 3, 'red': 2, 'green': 1})
```

Python Number random() Method

In [121]:

```
# Example
import random

# First random number
print("random() : ", random.random())

# Second random number
print("random() : ", random.random())
```

```
random() :  0.333601455009063
random() :  0.7649358107137664
```

In [122]:

```
import random
print(random.randint(0, 5))
```

```
5
```

In [123]:

```
import random
print(random.random() * 100)
```

```
72.49014831732813
```

In [124]:

```
print(random.choice( ['red', 'black', 'green'] ))
```

```
green
```

In [125]:

```
import random
for x in range(10):
      print(random.randint(1,101))
```

```
27
62
99
4
21
26
3
18
41
62
```

In [126]:

```
import random
for i in range(3):
    print (random.randrange(0, 101, 5)) #range(start, stop, step)
```

```
20
40
80
```

Python Agg Function

In [127]:

```
import pandas as pd
import numpy as np
df = pd.DataFrame([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9],
                    [np.nan, np.nan, np.nan]],
                  columns=['A', 'B', 'C'])
```

In [128]:

```
df.agg(['sum', 'min'])
```

Out[128]:

	A	B	C
sum	12.0	15.0	18.0
min	1.0	2.0	3.0

In [129]:

```
df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
```

Out[129]:

	A	B
max	NaN	8.0
min	1.0	2.0
sum	12.0	NaN

In [130]:

```
def add(x):
    return x
```

In [131]:

```
df2 = df.agg({'A':[add,lambda x: x/2]})['A']
```

In [132]:

```
print(df2)
```

```
   add  <lambda>
0  1.0       0.5
1  4.0       2.0
2  7.0       3.5
3  NaN       NaN
```

Python Pandas - GroupBy

Any groupby operation involves one of the following operations on the original object. They are −

Splitting the Object

Applying a function

Combining the results

In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations −

Aggregation − computing a summary statistic

Transformation − perform some group-specific operation

Filtration − discarding the data with some condition

Filtration – discarding the data with some condition

In [133]:

```
#import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
         'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
         'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
         'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
         'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print (df)
```

```
    Points  Rank    Team  Year
0      876     1  Riders  2014
1      789     2  Riders  2015
2      863     2  Devils  2014
3      673     3  Devils  2015
4      741     3   Kings  2014
5      812     4   kings  2015
6      756     1   Kings  2016
7      788     1   Kings  2017
8      694     2  Riders  2016
9      701     4  Royals  2014
10     804     1  Royals  2015
11     690     2  Riders  2017
```

Split Data into Groups

Pandas object can be split into any of their objects. There are multiple ways to split an object like −

1. obj.groupby('key')

2. obj.groupby(['key1','key2'])

3. obj.groupby(key,axis=1)

Let us now see how the grouping objects can be applied to the DataFrame object

Example

In [134]:

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
         'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
         'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
         'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
         'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print(df.groupby('Team'))
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x0D691A50>
```

View Groups

In [135]:

```
print(df.groupby('Team').groups)
```

```
{'Devils': Int64Index([2, 3], dtype='int64'), 'Kings': Int64Index([4, 6, 7], dtype='int64'), 'Ride
rs': Int64Index([0, 1, 8, 11], dtype='int64'), 'Royals': Int64Index([9, 10], dtype='int64'),
'kings': Int64Index([5], dtype='int64')}
```

Example

Group by with multiple columns −

In [136]:

```
print(df.groupby(['Team','Year']).groups)
```

```
{('Devils', 2014): Int64Index([2], dtype='int64'), ('Devils', 2015): Int64Index([3],
dtype='int64'), ('Kings', 2014): Int64Index([4], dtype='int64'), ('Kings', 2016): Int64Index([6],
dtype='int64'), ('Kings', 2017): Int64Index([7], dtype='int64'), ('Riders', 2014): Int64Index([0],
dtype='int64'), ('Riders', 2015): Int64Index([1], dtype='int64'), ('Riders', 2016):
Int64Index([8], dtype='int64'), ('Riders', 2017): Int64Index([11], dtype='int64'), ('Royals',
2014): Int64Index([9], dtype='int64'), ('Royals', 2015): Int64Index([10], dtype='int64'),
('kings', 2015): Int64Index([5], dtype='int64')}
```

Iterating through Groups

With the groupby object in hand, we can iterate through the object similar to itertools.obj.

In [137]:

```
grouped = df.groupby('Year')

for name,group in grouped:
    print (name)
    print (group)
```

```
2014
   Points  Rank    Team  Year
0     876     1  Riders  2014
2     863     2  Devils  2014
4     741     3   Kings  2014
9     701     4  Royals  2014
2015
    Points  Rank    Team  Year
1      789     2  Riders  2015
3      673     3  Devils  2015
5      812     4   kings  2015
10     804     1  Royals  2015
2016
   Points  Rank    Team  Year
6     756     1   Kings  2016
8     694     2  Riders  2016
2017
    Points  Rank    Team  Year
7      788     1   Kings  2017
11     690     2  Riders  2017
```

Select a Group

Using the get_group() method, we can select a single group.

In [138]:

```
grouped = df.groupby('Year')
print(grouped.get_group(2014))
```

```
   Points  Rank    Team  Year
0     876     1  Riders  2014
2     863     2  Devils  2014
4     741     3   Kings  2014
```

```
9     701     4  Royals  2014
```

Aggregations

An aggregated function returns a single aggregated value for each group. Once the group by object is created, several aggregation operations can be performed on the grouped data. An obvious one is aggregation via the aggregate or equivalent ## agg method −

In [139]:

```
import numpy as np
grouped = df.groupby('Year')
print(grouped['Points'].agg(np.mean))
```

```
Year
2014    795.25
2015    769.50
2016    725.00
2017    739.00
Name: Points, dtype: float64
```

In [140]:

```
# Another way to see the size of each group is by applying the size() function −
grouped = df.groupby('Team')
print(grouped.agg(np.size))
```

```
        Points  Rank  Year
Team
Devils       2     2     2
Kings        3     3     3
Riders       4     4     4
Royals       2     2     2
kings        1     1     1
```

Applying Multiple Aggregation Functions at Once

With grouped Series, you can also pass a list or dict of functions to do aggregation with, and generate DataFrame as output −

In [141]:

```
grouped = df.groupby('Team')
print(grouped['Points'].agg([np.sum, np.mean, np.std]))
```

```
         sum        mean         std
Team
Devils  1536  768.000000  134.350288
Kings   2285  761.666667   24.006943
Riders  3049  762.250000   88.567771
Royals  1505  752.500000   72.831998
kings    812  812.000000         NaN
```

Transformations

Transformation on a group or a column returns an object that is indexed the same size of that is being grouped. Thus, the transform should return a result that is the same size as that of a group chunk.

In [142]:

```
grouped = df.groupby('Team')
score = lambda x: (x - x.mean()) / x.std()*10
print(grouped.transform(score))
```

```
       Points       Rank       Year
0   12.843272 -15.000000 -11.618950
1    3.020286   5.000000  -3.872983
```

```
2    7.071068  -7.071068  -7.071068
3   -7.071068   7.071068   7.071068
4   -8.608621  11.547005 -10.910895
5         NaN        NaN        NaN
6   -2.360428  -5.773503   2.182179
7   10.969049  -5.773503   8.728716
8   -7.705963   5.000000   3.872983
9   -7.071068   7.071068  -7.071068
10   7.071068  -7.071068   7.071068
11  -8.157595   5.000000  11.618950
```

Filtration

Filtration filters the data on a defined criteria and returns the subset of data. The filter() function is used to filter the data.

In [143]:

```
print(df.groupby('Team').filter(lambda x: len(x) >= 3))
```

```
    Points  Rank    Team  Year
0      876     1  Riders  2014
1      789     2  Riders  2015
4      741     3   Kings  2014
6      756     1   Kings  2016
7      788     1   Kings  2017
8      694     2  Riders  2016
11     690     2  Riders  2017
```

numpy.random.rand() :

In [144]:

```
import numpy as np
np.random.rand(3,2)
```

Out[144]:

```
array([[ 0.68551272,  0.71602392],
       [ 0.86216627,  0.50804434],
       [ 0.461094  ,  0.96511632]])
```

In [145]:

```
np.random.rand(10)
```

Out[145]:

```
array([ 0.79651226,  0.55873099,  0.33061707,  0.845238  ,  0.45543639,
        0.09268519,  0.45490427,  0.8719684 ,  0.44828215,  0.01434915])
```

In [146]:

```
import random
print(random.randint(0,9))
```

```
5
```

Row or Column Wise Function Application: apply()

Arbitrary functions can be applied along the axes of a DataFrame or Panel using the apply() method, which, like the descriptive statistics methods, takes an optional axis argument. By default, the operation performs column wise, taking each column as an array-like.

Example 1

In [147]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
print(df)
df.apply(np.mean)
print (df.apply(np.mean))
```

```
       col1      col2      col3
0  0.433455  1.172939  2.608326
1 -0.038906  0.710814  0.474578
2  1.979317 -0.646196  0.233477
3  1.564120 -0.301892 -1.139473
4  0.224736 -0.502332  1.375642
col1    0.832544
col2    0.086667
col3    0.710510
dtype: float64
```

Example 2:

By passing axis parameter, operations can be performed row wise

In [148]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
df.apply(np.mean,axis=1)
print(df.apply(np.mean))
```

```
col1    0.023225
col2   -0.062539
col3   -0.695665
dtype: float64
```

Example 3:

In [149]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5,3),columns=['col1','col2','col3'])
df.apply(lambda x: x.max() - x.min())
print (df.apply(np.mean))
```

```
col1   -0.419467
col2    0.338818
col3    0.321762
dtype: float64
```

In [150]:

```
df = pd.DataFrame([[4, 9],] * 3, columns=['A', 'B'])
df
```

Out[150]:

	A	B
0	4	9

	A	B
1	4	9
2	4	9

In [151]:

```
df.apply(np.sqrt)
```

Out[151]:

	A	B
0	2.0	3.0
1	2.0	3.0
2	2.0	3.0

In [152]:

```
df.apply(np.sum, axis=0)
```

Out[152]:

```
A    12
B    27
dtype: int64
```

In [153]:

```
df.apply(np.sum, axis=1)
```

Out[153]:

```
0    13
1    13
2    13
dtype: int64
```

In [154]:

```
df.apply(lambda x: [1, 2], axis=1)
```

Out[154]:

	A	B
0	1	2
1	1	2
2	1	2

In [155]:

```
df.apply(lambda x: pd.Series([1, 2], index=['foo', 'bar']), axis=1)
```

Out[155]:

	foo	bar
0	1	2
1	1	2
2	1	2

In [156]:

```
import pandas as pd
```

In [157]:

```
url = 'http://bit.ly/kaggletrain'
train = pd.read_csv(url)
train.head(3)
```

Out[157]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

map() function as a Series method

Mostly used for mapping categorical data to numerical data

In [158]:

```
# create new column
train['Sex_num'] = train.Sex.map({'female':0, 'male':1})
```

In [159]:

```
# let's compared Sex and Sex_num columns
# here we can see we map male to 1 and female to 0
train.loc[0:4, ['Sex', 'Sex_num']]
```

Out[159]:

	Sex	Sex_num
0	male	1
1	female	0
2	female	0
3	female	0
4	male	1

apply() function as a Series method

Applies a function to each element in the Series

In [160]:

```
# say we want to calculate length of string in each string in "Name" column

# create new column
# we are applying Python's len function
train['Name_length'] = train.Name.apply(len)
```

In [161]:

```
# the apply() method applies the function to each element
train.loc[0:4, ['Name', 'Name_length']]
```

Out[161]:

	Name	Name_length
0	Braund, Mr. Owen Harris	23
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	51
2	Heikkinen, Miss. Laina	22
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	44
4	Allen, Mr. William Henry	24

In [162]:

```
import numpy as np

# say we look at the "Fare" column and we want to round it up
# we will use numpy's ceil function to round up the numbers
train['Fare_ceil'] = train.Fare.apply(np.ceil)
```

In [163]:

```
train.loc[0:4, ['Fare', 'Fare_ceil']]
```

Out[163]:

	Fare	Fare_ceil
0	7.2500	8.0
1	71.2833	72.0
2	7.9250	8.0
3	53.1000	54.0
4	8.0500	9.0

In [164]:

```
# let's extract last name of each person

# we will use a str method
# now the series is a list of strings
# each cell has 2 strings in a list as you can see below
train.Name.str.split(',').head()
```

Out[164]:

```
0                            [Braund,  Mr. Owen Harris]
1     [Cumings,  Mrs. John Bradley (Florence Briggs ...
2                             [Heikkinen,  Miss. Laina]
3       [Futrelle,  Mrs. Jacques Heath (Lily May Peel)]
4                           [Allen,  Mr. William Henry]
Name: Name, dtype: object
```

In [165]:

```
# we just want the first string from the list
# we create a function to retrieve
def get_element(my_list, position):
    return my_list[position]
```

In [166]:

```
# use our created get_element function
# we pass position=0
train.Name.str.split(',').apply(get_element, position=0).head()
```

Out[166]:

```
0       Braund
1      Cumings
2    Heikkinen
3     Futrelle
4        Allen
Name: Name, dtype: object
```

apply() function as a DataFrame method

Applies a function on either axis of the DataFrame

In [167]:

```
url = 'http://bit.ly/drinksbycountry'
drinks = pd.read_csv(url)
drinks.head()
```

Out[167]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa

In [168]:

```
drinks.loc[:, 'beer_servings':'wine_servings'].head()
```

Out[168]:

	beer_servings	spirit_servings	wine_servings
0	0	0	0
1	89	132	54
2	25	0	14
3	245	138	312
4	217	57	45

In [169]:

```
# you want apply() method to travel axis=0 (downwards, column)
# apply Python's max() function
drinks.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=0)
```

Out[169]:

```
beer_servings      376
spirit_servings    438
wine_servings      370
dtype: int64
```

In [170]:

```
# you want apply() method to travel axis=1 (right, row)
# apply Python's max() function
drinks.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=1)
```

Out[170]:

```
0        0
1      132
2       25
3      312
4      217
5      128
6      221
7      179
8      261
9      279
10      46
11     176
12      63
13       0
14     173
15     373
16     295
17     263
18      34
19      23
20     167
21     173
22     173
23     245
24      31
25     252
26      25
27      88
28      37
29     144
      ... 
163    178
164     90
165    186
166    280
167     35
168     15
169    258
170    106
171      4
172     36
173     36
174    197
175     51
176     51
177     71
178     41
179     45
180    237
181    135
182    219
183     36
184    249
185    220
186    101
187     21
188    333
189    111
190      6
191     32
192     64
Length: 193, dtype: int64
```

In [171]:

```
# finding which column is the maximum's category name
drinks.loc[:, 'beer_servings':'wine_servings'].apply(np.argmax, axis=1)
```

Out[171]:

```
0        beer_servings
1      spirit_servings
2        beer_servings
3        wine_servings
4        beer_servings
5      spirit_servings
6        wine_servings
7      spirit_servings
8        beer_servings
9        beer_servings
10     spirit_servings
11     spirit_servings
12     spirit_servings
13       beer_servings
14     spirit_servings
15     spirit_servings
16       beer_servings
17       beer_servings
18       beer_servings
19       beer_servings
20       beer_servings
21     spirit_servings
22       beer_servings
23       beer_servings
24       beer_servings
25     spirit_servings
26       beer_servings
27       beer_servings
28       beer_servings
29       beer_servings
             ...
163    spirit_servings
164      beer_servings
165      wine_servings
166      wine_servings
167    spirit_servings
168    spirit_servings
169    spirit_servings
170      beer_servings
171      wine_servings
172      beer_servings
173      beer_servings
174      beer_servings
175      beer_servings
176      beer_servings
177    spirit_servings
178    spirit_servings
179      beer_servings
180    spirit_servings
181    spirit_servings
182      beer_servings
183      beer_servings
184      beer_servings
185      wine_servings
186    spirit_servings
187      beer_servings
188      beer_servings
189      beer_servings
190      beer_servings
191      beer_servings
192      beer_servings
Length: 193, dtype: object
```

Iterator

Iterator is an object which allows a programmer to traverse through all the elements of a collection, regardless of its specific implementation. In Python, an iterator object implements two methods, iter() and next().

String, List or Tuple objects can be used to create an Iterator.

In [172]:

```
import sys
list = [1,2,3,4]
it = iter(list) # this builds an iterator object
print ("Next Available Element in iterator:", next(it)) #prints next available element in iterator
print ("Next Available Element in iterator:", next(it))
#Iterator object can be traversed using regular for statement

while True:
    try:
        print (next(it))
    except StopIteration:
        sys.exit()
```

```
Next Available Element in iterator: 1
Next Available Element in iterator: 2
3
4
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

```
C:\Users\thyagaragu\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2870:
UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

isinstance()

In [173]:

```
# Python code for  isinstance()
# Syntax : isinstance(object, classinfo)
#The isinstance() takes two parameters:
# object : object to be checked
# classinfo : class, type, or tuple of classes and types

class Test:
    a = 5
TestInstance = Test()

print(isinstance(TestInstance, Test))
```

```
True
```

PPRINT

In [176]:

```
import pprint
```

In [177]:

```
data = {'a':2, 'b':{'x':3, 'y':{'t1': 4, 't2':5}}}
print(data)
pprint.pprint(data)
```

```
{'a': 2, 'b': {'x': 3, 'y': {'t1': 4, 't2': 5}}}
{'a': 2, 'b': {'x': 3, 'y': {'t1': 4, 't2': 5}}}
```

In [178]:

```
data = { 'key1' : 'value1',
                    'key2' : 'value2',
                    'key3' : { 'key3a': 'value3a' },
```

```
                        'key4' : { 'key4a': { 'key4aa': 'value4aa',
                                              'key4ab': 'value4ab',
                                              'key4ac': 'value4ac'},
                                   'key4b': 'value4b'}
                       }
print("USING PRINT:\n",data)
print("\n\n USING PPRINT:\n")
pprint.pprint(data)
```

```
USING PRINT:
 {'key1': 'value1', 'key2': 'value2', 'key3': {'key3a': 'value3a'}, 'key4': {'key4a': {'key4aa': '
value4aa', 'key4ab': 'value4ab', 'key4ac': 'value4ac'}, 'key4b': 'value4b'}}


 USING PPRINT:

{'key1': 'value1',
 'key2': 'value2',
 'key3': {'key3a': 'value3a'},
 'key4': {'key4a': {'key4aa': 'value4aa',
                    'key4ab': 'value4ab',
                    'key4ac': 'value4ac'},
          'key4b': 'value4b'}}
```

floor() and ceil() function Python

In [179]:

```
# Python program to demonstrate the use of floor() method

# This will import math module
import math

# prints the ceil using floor() method
print("math.floor(-23.11) : ", math.floor(-23.11))
print("math.floor(300.16) : ", math.floor(300.16))
print("math.floor(300.72) : ", math.floor(300.72))
```

```
math.floor(-23.11) :  -24
math.floor(300.16) :  300
math.floor(300.72) :  300
```

In [180]:

```
# Python program to demonstrate the use of ceil() method

# This will import math module
import math

# prints the ceil using ceil() method
print("math.ceil(-23.11) : ", math.ceil(-23.11))
print("math.ceil(300.16) : ", math.ceil(300.16))
print("math.ceil(300.72) : ", math.ceil(300.72))
```

```
math.ceil(-23.11) :  -23
math.ceil(300.16) :  301
math.ceil(300.72) :  301
```

In [181]:

```
from math import ceil
from math import floor
print("ceil(-23.11) : ",ceil(-23.11))
print("ceil(300.16) : ",ceil(300.16))
print("ceil(300.72) : ",ceil(300.72))
print("floor(-23.11): ",floor(-23.11))
print("floor(300.16): ",floor(300.16))
print("floor(300.72): ",floor(300.72))
```

```
ceil(-23.11) :  -23
```

```
ceil(300.16) :  301
ceil(300.72) :  301
floor(-23.11):  -24
floor(300.16):  300
floor(300.72):  300
```

Numpy Sort

In [182]:

```
import numpy as np
a = np.array([[3,7],[9,1]])
b = np.array([3,7,9,1])

print('Array a is:')
print(a)
print('\n')
print('Array b is:')
print(b)

print('\n Sorted Array a is :')
print(np.sort(a))
print('\n')
print('Sorted Array b is :')
print(np.sort(b))

print('\nSort along axis 0:')
print(np.sort(a, axis = 0))
print('\n')
```

```
Array a is:
[[3 7]
 [9 1]]


Array b is:
[3 7 9 1]

 Sorted Array a is :
[[3 7]
 [1 9]]


Sorted Array b is :
[1 3 7 9]

Sort along axis 0:
[[3 1]
 [9 7]]
```

Numpy Clip

In [183]:

```
import numpy as np
my_array = np.array([[100, 200], [300, 400]],np.uint16)
my_array.clip(0,255) # clip(min, max)
```

Out[183]:

```
array([[100, 200],
       [255, 255]], dtype=uint16)
```

In [184]:

```
x=np.array([1,2,3,4,5])
h = [(np.abs(x - x[i])) for i in range(5)]
```

```
print(x)
print(h)
```

```
[1 2 3 4 5]
[array([0, 1, 2, 3, 4]), array([1, 0, 1, 2, 3]), array([2, 1, 0, 1, 2]), array([3, 2, 1, 0, 1]),
array([4, 3, 2, 1, 0])]
```

In [185]:

```
x=np.array([1,2,3,4,5])
h = [(np.abs(x - x[i]))[3] for i in range(5)] # Third Element
print(x)
print(h)
```

```
[1 2 3 4 5]
[3, 2, 1, 0, 1]
```

In [186]:

```
x=np.array([1,2,3,4,5])
print("x:\n",x)
print("x-x[i]:\n",[abs(x-x[i]) for i in range(5)])
h = [(np.abs(x - x[i]))[4] for i in range(5)] # To determine the nearest points to 4
print("Forth in  h:\n",h)
h = [np.sort(np.abs(x - x[i])) for i in range(5)]
print("sorted h:\n",h)
h = [np.sort(np.abs(x - x[i]))[4] for i in range(5)]
print(" The Largest Distance Points sets :\n",h)
```

```
x:
 [1 2 3 4 5]
x-x[i]:
 [array([0, 1, 2, 3, 4]), array([1, 0, 1, 2, 3]), array([2, 1, 0, 1, 2]), array([3, 2, 1, 0, 1]),
array([4, 3, 2, 1, 0])]
Forth in  h:
 [4, 3, 2, 1, 0]
sorted h:
 [array([0, 1, 2, 3, 4]), array([0, 1, 1, 2, 3]), array([0, 1, 1, 2, 2]), array([0, 1, 1, 2, 3]),
array([0, 1, 2, 3, 4])]
 The Largest Distance Points sets :
 [4, 3, 2, 3, 4]
```

In [187]:

```
print(x[:, None])
```

```
[[1]
 [2]
 [3]
 [4]
 [5]]
```

In [188]:

```
print(x[None, :])
```

```
[[1 2 3 4 5]]
```

In [189]:

```
print((x[:, None] - x[None, :]))
```

```
[[ 0 -1 -2 -3 -4]
 [ 1  0 -1 -2 -3]
 [ 2  1  0 -1 -2]
 [ 3  2  1  0 -1]
 [ 4  3  2  1  0]]
```

In [190]:

```
print(np.abs(x[:, None] - x[None, :]))
```

```
[[0 1 2 3 4]
 [1 0 1 2 3]
 [2 1 0 1 2]
 [3 2 1 0 1]
 [4 3 2 1 0]]
```

In [191]:

```
print(h)
```

```
[4, 3, 2, 3, 4]
```

In [192]:

```
print(np.abs(x[:, None] - x[None, :])/ h)
```

```
[[ 0.          0.33333333  1.          1.          1.        ]
 [ 0.25        0.          0.5         0.66666667  0.75      ]
 [ 0.5         0.33333333  0.          0.33333333  0.5       ]
 [ 0.75        0.66666667  0.5         0.          0.25      ]
 [ 1.          1.          1.          0.33333333  0.        ]]
```

In [193]:

```
 w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
print(w)
```

```
[[ 0.          0.33333333  1.          1.          1.        ]
 [ 0.25        0.          0.5         0.66666667  0.75      ]
 [ 0.5         0.33333333  0.          0.33333333  0.5       ]
 [ 0.75        0.66666667  0.5         0.          0.25      ]
 [ 1.          1.          1.          0.33333333  0.        ]]
```

In [196]:

```
delta = np.ones(5)
print(delta)
```

```
[ 1.  1.  1.  1.  1.]
```

In [198]:

```
y= 1 + np.random.randn(5)
print(y)
```

```
[ 0.13186982 -0.94434505 -0.07433116  3.46267489 -0.90062144]
```

In [200]:

```
delta = np.ones(5)
for i in range(5):
    weights = delta * w[:, i] # Assigning Weights to each point
    b = np.array([np.sum(weights * y), np.sum(weights * y * x)]) # Matrix B
    A = np.array([[np.sum(weights), np.sum(weights * x)],
                      [np.sum(weights * x), np.sum(weights * x * x)]]) # Matrix A

print(b)
print(A)
```

```
[ 0.25211417  2.06653039]
[[  2.5    5. ]
 [  5.   12.5]]
```

Usage of c_ and r_

In [1]:

```
import numpy as np
from numpy import c_
np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])]
```

Out[1]:

```
array([[1, 2, 3, 0, 0, 4, 5, 6]])
```

In [6]:

```
x0 = np.linspace(-3, 3, num=10)
print("X0:\n",x0)
x0 = np.r_[1, x0]
print("rX0:\n",x0)
```

```
X0:
 [-3.         -2.33333333 -1.66666667 -1.         -0.33333333  0.33333333
  1.          1.66666667  2.33333333  3.        ]
rX0:
 [ 1.         -3.         -2.33333333 -1.66666667 -1.         -0.33333333
  0.33333333  1.          1.66666667  2.33333333  3.        ]
```

pinv

In [12]:

```
import numpy as np
from numpy import linalg
A = np.array([[1,-2],[3,5]])
print("The Matrix A: \n",A)
print("The dimension of A:\n ",A.shape)
print("The inverse of A: \n",linalg.inv(A))
print("The pinverse of A: \n",linalg.pinv(A))
```

```
The Matrix A:
 [[ 1 -2]
 [ 3  5]]
The dimension of A:
  (2, 2)
The inverse of A:
 [[ 0.45454545  0.18181818]
 [-0.27272727  0.09090909]]
The pinverse of A:
 [[ 0.45454545  0.18181818]
 [-0.27272727  0.09090909]]
```

@ Python - Object Oriented

Creating Classes

In [2]:

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
```

```
    def displayCount(self):
      print("Total Employee %d" % Employee.empCount)

    def displayEmployee(self):
       print("Name : ", self.name,  ", Salary: ", self.salary)
```

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its init method accepts.

The variable empCount is a class variable whose value is shared among all instances of a this class. This can be accessed as Employee.empCount from inside the class or outside the class.

The first method init() is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

You declare other class methods like normal functions with the exception that the first argument to each method is self. Python adds the self argument to the list for you; you do not need to include it when you call the methods.

In [3]:

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows −

In [5]:

```
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)
```

```
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
Total Employee 2
```