Introduction to Numpy Arrays

In [5]:

```
import numpy as np
# from numpy import *
```

In [6]:

```
list1 = [1,2,3,4]
```

In [7]:

```
type(list1)
```

Out[7]:

```
list
```

In [9]:

```
a = np.array(list1)
```

In [10]:

```
a
```

Out[10]:

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

In [11]:

```
type(a)
```

Out[11]:

```
numpy.ndarray
```

In [13]:

```
a.ndim
```

Out[13]:

```
1
```

Numpy - Indexing and Slicing

In [16]:

```
list1 = [1,2,"John",45]
```

In [17]:

```
import numpy as np
a = np.array(list1)
```

In [18]:

```
a
```

Out[18]:

```
array(['1', '2', 'John', '45'], dtype='<U11')
```

In [21]:

```
a[2]
```

Out[21]:

```
'John'
```

In [22]:

```
a[:]
```

Out[22]:

```
array(['1', '2', 'John', '45'], dtype='<U11')
```

In [23]:

```
a[:3]
```

Out[23]:

```
array(['1', '2', 'John'], dtype='<U11')
```

In [24]:

```
a[1:3]
```

Out[24]:

```
array(['2', 'John'], dtype='<U11')
```

In [25]:

```
# 2D array - with nested lists
list1 = [[1,2,3],[4,5,6],[6,7,8],[8,9,10]]
```

In [26]:

```
type(list1)
```

Out[26]:

```
list
```

In [27]:

```
a = np.array(list1)
```

In [28]:

```
a
```

Out[28]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 6,  7,  8],
       [ 8,  9, 10]])
```

In [29]:

```
a.ndim
```

Out[29]:

```
2
```

In [30]:

```
a.shape
```

Out[30]:

```
(4, 3)
```

In [31]:

```
a[0]
```

Out[31]:

```
array([1, 2, 3])
```

In [32]:

```
a[1:3]
```

Out[32]:

```
array([[4, 5, 6],
       [6, 7, 8]])
```

In [34]:

```
a[0:4,1:2]
```

Out[34]:

```
array([[2],
       [5],
       [7],
       [9]])
```

In [36]:

```
a[:,1]
```

Out[36]:

```
array([2, 5, 7, 9])
```

In [37]:
```
a[:,1:2]
```

Out[37]:
```
array([[2],
       [5],
       [7],
       [9]])
```

In [40]:
```
a[1:3,1:]
```

Out[40]:
```
array([[5, 6],
       [7, 8]])
```

In [41]:
```
a[1,1]
```

Out[41]:
```
5
```

In [42]:
```
a[[0,1,2],[0,1,2]]
```

Out[42]:
```
array([1, 5, 8])
```

In [43]:
```
a[1,1] = 25
```

In [44]:
```
a
```

Out[44]:
```
array([[ 1,  2,  3],
       [ 4, 25,  6],
       [ 6,  7,  8],
       [ 8,  9, 10]])
```

In [45]:
```
# Concept of the broadcasting
a[1:] = 99
```

In [46]:
```
a
```

Out[46]:
```
array([[ 1,  2,  3],
       [99, 99, 99],
       [99, 99, 99],
       [99, 99, 99]])
```

Numpy Functions - Functions of Arrays

```
arange -> Gives 1D array of mentioned range
zeros
ones
linspace -> Gives evenly spaced numbers over a specied interval
rand
randint
min
max
argmax
argmin
unique
```

concatinate
split
ravel
flatten -> Convert multi-dimensional data into single dimension
(*) Flatten will copy the elements of multi-dimension array but ravel will not copy the elements of multi-dimmnsional array. But it will be the reference ot it.
transpose
shape
reshape

arange

```
In [47]:  a = np.arange(10)
```

```
In [48]:  a
```

```
Out[48]:  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [51]:  a = np.arange(10,20)
```

```
In [52]:  a
```

```
Out[52]:  array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
In [53]:  a = np.arange(10,20,2)
```

```
In [54]:  a
```

```
Out[54]:  array([10, 12, 14, 16, 18])
```

zeros

```
In [57]:  a = np.zeros((3))
```

```
In [58]:  a
```

```
Out[58]:  array([0., 0., 0.])
```

```
In [59]:  a = np.zeros((3,2))
```

```
In [60]:  a
```

```
Out[60]:  array([[0., 0.],
                 [0., 0.],
                 [0., 0.]])
```

ones

```
In [61]:  a = np.ones((2,3))
```

In [62]:
```
a
```

Out[62]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

eye

In [63]:
```
a = np.eye(3)
```

In [64]:
```
a
```

Out[64]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

linspace

In [66]:
```
a = np.linspace(10,20)
```

In [67]:
```
a
```

Out[67]:
```
array([10.        , 10.20408163, 10.40816327, 10.6122449 , 10.81632653,
       11.02040816, 11.2244898 , 11.42857143, 11.63265306, 11.83673469,
       12.04081633, 12.24489796, 12.44897959, 12.65306122, 12.85714286,
       13.06122449, 13.26530612, 13.46938776, 13.67346939, 13.87755102,
       14.08163265, 14.28571429, 14.48979592, 14.69387755, 14.89795918,
       15.10204082, 15.30612245, 15.51020408, 15.71428571, 15.91836735,
       16.12244898, 16.32653061, 16.53061224, 16.73469388, 16.93877551,
       17.14285714, 17.34693878, 17.55102041, 17.75510204, 17.95918367,
       18.16326531, 18.36734694, 18.57142857, 18.7755102 , 18.97959184,
       19.18367347, 19.3877551 , 19.59183673, 19.79591837, 20.        ])
```

In [70]:
```
a = np.linspace(10,20,5)
```

In [71]:
```
a
```

Out[71]:
```
array([10. , 12.5, 15. , 17.5, 20. ])
```

min max

In [72]:
```
a.min()
```

Out[72]:
```
10.0
```

In [73]:
```
a.max()
```

Out[73]:
```
20.0
```

```
In [74]: a.argmin()

Out[74]: 0
```

```
In [75]: a.argmax()

Out[75]: 4
```

random

```
In [76]: b = np.random.rand(2)
```

```
In [77]: b

Out[77]: array([0.21197496, 0.86338924])
```

```
In [78]: b = np.random.rand(3,3)
```

```
In [79]: b

Out[79]: array([[0.39066352, 0.6097913 , 0.38130661],
                [0.67228565, 0.52537071, 0.68651863],
                [0.7479389 , 0.25953038, 0.1577244 ]])
```

Integer

```
In [80]: c = np.random.randint(100)
```

```
In [81]: c

Out[81]: 69
```

```
In [86]: d = np.random.randint(1,100,10)
```

```
In [87]: d

Out[87]: array([51, 51, 24, 47, 92, 57, 46, 14, 42, 10])
```

flatten

```
In [88]: f = np.arange(16)
         f

Out[88]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
In [89]: g = f.reshape(4,4)
In [90]:  g

Out[90]:  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In [91]:  h = g.ravel()
```

```
In [92]:  h

Out[92]:  array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
In [102]:  h[6] = 45
```

```
In [103]:  h

Out[103]:  array([ 0,  1,  2,  3, 20,  5, 45,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
In [104]:  g

Out[104]:  array([[ 0,  1,  2,  3],
                  [20,  5, 45,  7],
                  [ 8,  9, 10, 11],
                  [12, 13, 14, 15]])
```

```
In [105]:  i = g.flatten()
```

```
In [106]:  i

Out[106]:  array([ 0,  1,  2,  3, 20,  5, 45,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
In [107]:  i[4] = 39
```

```
In [108]:  i

Out[108]:  array([ 0,  1,  2,  3, 39,  5, 45,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
In [109]:  g

Out[109]:  array([[ 0,  1,  2,  3],
                  [20,  5, 45,  7],
                  [ 8,  9, 10, 11],
                  [12, 13, 14, 15]])
```

Transpose

```
In [110]:  j = np.transpose(g)
In [111]:  j

Out[111]: array([[ 0, 20,  8, 12],
                 [ 1,  5,  9, 13],
                 [ 2, 45, 10, 14],
                 [ 3,  7, 11, 15]])
```

```
In [112]:  k = j.T
```

```
In [113]:  k

Out[113]: array([[ 0,  1,  2,  3],
                 [20,  5, 45,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In [116]:  l = np.concatenate((j,k))
```

```
In [117]:  l

Out[117]: array([[ 0, 20,  8, 12],
                 [ 1,  5,  9, 13],
                 [ 2, 45, 10, 14],
                 [ 3,  7, 11, 15],
                 [ 0,  1,  2,  3],
                 [20,  5, 45,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In [118]:  l = np.concatenate((j,k), axis = 1)
```

```
In [119]:  l

Out[119]: array([[ 0, 20,  8, 12,  0,  1,  2,  3],
                 [ 1,  5,  9, 13, 20,  5, 45,  7],
                 [ 2, 45, 10, 14,  8,  9, 10, 11],
                 [ 3,  7, 11, 15, 12, 13, 14, 15]])
```

Split

```
In [121]:  p = np.split(l,2)
```

```
In [122]:  p

Out[122]: [array([[ 0, 20,  8, 12,  0,  1,  2,  3],
                  [ 1,  5,  9, 13, 20,  5, 45,  7]]),
           array([[ 2, 45, 10, 14,  8,  9, 10, 11],
                  [ 3,  7, 11, 15, 12, 13, 14, 15]])]
```

In [125]:
```
p = np.split(l,3,axis = 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
S:\Anaconda\lib\site-packages\numpy\lib\shape_base.py in split(ary, indices_or_sections, a
xis)
    864     try:
--> 865         len(indices_or_sections)
    866     except TypeError:

TypeError: object of type 'int' has no len()

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-125-c7e9ffb8087d> in <module>
----> 1 p = np.split(l,3,axis = 1)

<__array_function__ internals> in split(*args, **kwargs)

S:\Anaconda\lib\site-packages\numpy\lib\shape_base.py in split(ary, indices_or_sections, a
xis)
    869         if N % sections:
    870             raise ValueError(
--> 871                 'array split does not result in an equal division')
    872     return array_split(ary, indices_or_sections, axis)
    873 

ValueError: array split does not result in an equal division
```

In [126]:
```
p
```

Out[126]:
```
[array([[ 0, 20,  8, 12],
        [ 1,  5,  9, 13],
        [ 2, 45, 10, 14],
        [ 3,  7, 11, 15]]), array([[ 0,  1,  2,  3],
        [20,  5, 45,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]])]
```

Unique

In [128]:
```
np.unique(l)
```

Out[128]:
```
array([ 0,  1,  2,  3,  5,  7,  8,  9, 10, 11, 12, 13, 14, 15, 20, 45])
```

In [129]:
```
np.unique(l, return_counts = True)
```

Out[129]:
```
(array([ 0,  1,  2,  3,  5,  7,  8,  9, 10, 11, 12, 13, 14, 15, 20, 45]),
 array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int64))
```

Numpy Arithematic Operations

In [130]:
```
arr = np.arange(0,10)
```

In [131]:
```
arr
```

Out[131]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [132]:
```
arr1 = arr + arr
```

In [133]:
```
arr1
```

Out[133]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [134]:
```
np.sqrt(arr)
```

Out[134]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [135]:
```
np.exp(arr)
```

Out[135]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [136]:
```
np.sin(arr)
```

Out[136]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [137]:
```
arr>4
```

Out[137]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [138]:
```
arr[arr>4]
```

Out[138]:
```
array([5, 6, 7, 8, 9])
```

In [139]:
```
arr[arr%2==0]
```

Out[139]:
```
array([0, 2, 4, 6, 8])
```

Pandas Series - For cleaning and analysis of data

In [140]:
```
import pandas as pd
```

In [141]:
```
# Data Types -
# [1] Series - Have only one column
# [2] Data Frame - Multi-Dimensional
```

In [144]:

```
list1 = [1,2,3,4]
ser = pd.Series(list1)
```

In [146]:

```
type(list1)
```

Out[146]:

```
list
```

In [147]:

```
type(ser)
```

Out[147]:

```
pandas.core.series.Series
```

In [148]:

```
ser
```

Out[148]:

```
0    1
1    2
2    3
3    4
dtype: int64
```

In [150]:

```
labels = ["a","b","c","d"]
ser = pd.Series(list1, index = labels)
```

In [151]:

```
ser
```

Out[151]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

In [154]:

```
student = ["Srinivas","Rahul","Sapireddy","Sri"]
marks = [80,82,84,83]
table = pd.Series(data = marks, index = student)
```

In [155]:

```
table
```

Out[155]:

```
Srinivas     80
Rahul        82
Sapireddy    84
Sri          83
dtype: int64
```

In [156]:

```
# Series with the help of dictionaries
dict = {"Ram":80,"Srinivas":65,"Pandas":79,"Sri":80}
```

In [157]:

```
dict
```

Out[157]:

```
{'Ram': 80, 'Srinivas': 65, 'Pandas': 79, 'Sri': 80}
```

In [158]:

```
ser = pd.Series(dict)
```

In [159]:
```
ser
```
Out[159]:
```
Ram         80
Srinivas    65
Pandas      79
Sri         80
dtype: int64
```

In [160]:
```
ser["Ram"]
```
Out[160]:
```
80
```

Pandas Dataframes - 2D

In [161]:
```
data = pd.DataFrame(np.arange(16).reshape(4,4))
```

In [162]:
```
data
```
Out[162]:

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

In [163]:
```
type(data)
```
Out[163]:
```
pandas.core.frame.DataFrame
```

In [164]:
```
data = pd.DataFrame(np.arange(16).reshape(4,4), index = "a b c d".split(), columns = "x y z w".split(
```

In [165]:
```
data
```
Out[165]:

	x	y	z	w
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [166]:
```
string = "welcome to python"
string.split()
```
Out[166]:
```
['welcome', 'to', 'python']
```

In [167]:
```
data = pd.DataFrame(np.arange(16).reshape(4,4), index = ["a", "b","c","d"], columns = "x y z w".split
```

In [168]:

```
data
```

Out[168]:

```
    x   y   z   w
a   0   1   2   3
b   4   5   6   7
c   8   9  10  11
d  12  13  14  15
```

In [171]:

```
data["w"]
```

Out[171]:

```
a     3
b     7
c    11
d    15
Name: w, dtype: int32
```

In [172]:

```
data.w
```

Out[172]:

```
a     3
b     7
c    11
d    15
Name: w, dtype: int32
```

In [173]:

```
data[["y","w"]]
```

Out[173]:

```
    y   w
a   1   3
b   5   7
c   9  11
d  13  15
```

In [175]:

```
data["a"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
S:\Anaconda\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, t
olerance)
   2896             try:
-> 2897                 return self._engine.get_loc(key)
   2898             except KeyError:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item
()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item
()

KeyError: 'a'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-175-49e0c99eb6c4> in <module>
----> 1 data["a"]

S:\Anaconda\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   2978             if self.columns.nlevels > 1:
   2979                 return self._getitem_multilevel(key)
-> 2980             indexer = self.columns.get_loc(key)
   2981             if is_integer(indexer):
   2982                 indexer = [indexer]

S:\Anaconda\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, t
olerance)
   2897                 return self._engine.get_loc(key)
   2898             except KeyError:
-> 2899                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2900         indexer = self.get_indexer([key], method=method, tolerance=tolerance)
   2901         if indexer.ndim > 1 or indexer.size > 1:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item
()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item
()

KeyError: 'a'
```

In [178]:

```
# To grab the rows you have to use the function called loc (location)
data.loc["a"]
```

Out[178]:

```
x   0
y   1
z   2
w   3
Name: a, dtype: int32
```

In [180]:

```
data.loc[["a","b"]]
```

Out[180]:

	x	y	z	w
a	0	1	2	3
b	4	5	6	7

In [181]:

```
data
```

Out[181]:

	x	y	z	w
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [182]:

```
data.iloc[2:4, 1:]
```

Out[182]:

	y	z	w
c	9	10	11
d	13	14	15

In [186]:

```
data["new"] = [0,1,2,3,4]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-186-4877c4eacde5> in <module>
----> 1 data["new"] = [0,1,2,3,4]

S:\Anaconda\lib\site-packages\pandas\core\frame.py in __setitem__(self, key, value)
   3470         else:
   3471             # set column
-> 3472             self._set_item(key, value)
   3473 
   3474     def _setitem_slice(self, key, value):

S:\Anaconda\lib\site-packages\pandas\core\frame.py in _set_item(self, key, value)
   3547 
   3548         self._ensure_valid_index(value)
-> 3549         value = self._sanitize_column(key, value)
   3550         NDFrame._set_item(self, key, value)
   3551 

S:\Anaconda\lib\site-packages\pandas\core\frame.py in _sanitize_column(self, key, value, b
roadcast)
   3732 
   3733             # turn me into an ndarray
-> 3734             value = sanitize_index(value, self.index, copy=False)
   3735             if not isinstance(value, (np.ndarray, Index)):
   3736                 if isinstance(value, list) and len(value) > 0:

S:\Anaconda\lib\site-packages\pandas\core\internals\construction.py in sanitize_index(da
ta, index, copy)
    610 
    611     if len(data) != len(index):
--> 612         raise ValueError("Length of values does not match length of index")
    613 
    614     if isinstance(data, ABCIndexClass) and not copy:

ValueError: Length of values does not match length of index
```

In [187]:

```
data["new"] = [0,1,2,3]
```

In [188]:

```
data
```

Out[188]:

	x	y	z	w	new
a	0	1	2	3	0
b	4	5	6	7	1
c	8	9	10	11	2
d	12	13	14	15	3

Python - BuiltIn Functions

Python - Builtin Functions

In [189]:
```
a = np.random.randint(1,100,56)
```

In [190]:
```
dict_a = {"A": np.random.rand(5), "B": np.random.rand(5), "C": np.random.rand(5)}
df = pd.DataFrame(data = dict_a, index = ['a','b','c','d','e'])
df
```

Out[190]:
```
          A         B         C
a  0.717055  0.089419  0.344361
b  0.298037  0.515611  0.868315
c  0.156920  0.527104  0.233556
d  0.037176  0.975781  0.917907
e  0.680020  0.267801  0.844523
```

In [191]:
```
dict = {"a": [3,4,5,6], "b": [4,5,6,7], "c": [7,8,9,10], "d": [11,12,13,14]}
```

In [192]:
```
data = pd.DataFrame(dict)
```

In [193]:
```
data
```

Out[193]:
```
   a  b   c   d
0  3  4   7  11
1  4  5   8  12
2  5  6   9  13
3  6  7  10  14
```

In [194]:
```
data = pd.DataFrame(dict, index = "h k j l ".split())
```

In [195]:
```
data
```

Out[195]:
```
   a  b   c   d
h  3  4   7  11
k  4  5   8  12
j  5  6   9  13
l  6  7  10  14
```

In [267]:
```
a = pd.DataFrame(np.random.randint(1,100,56).reshape(7,8), index = "a b c d e f g".split(), columns
```

In [206]:

```
a
```

Out[206]:

	0	1	2	3	4	5	6	7
0	19	78	97	56	7	41	21	41
1	25	28	9	56	20	97	86	77
2	69	22	15	83	95	87	33	98
3	48	48	21	7	87	83	51	64
4	75	92	82	97	51	46	78	14
5	37	11	55	68	50	23	46	26
6	24	27	59	5	73	98	31	98

In [210]:

```
a.iloc[4:,3:]
```

Out[210]:

	3	4	5	6	7
4	97	51	46	78	14
5	68	50	23	46	26
6	5	73	98	31	98

InBuilt Methods of DataFrames

In [229]:

```
a['new'] = ["34","2","3","4","5","6","7"]
```

In [230]:

```
a
```

Out[230]:

	0	1	2	3	4	5	6	7	new	new
0	19	78	97	56	7	41	21	41	34	34
1	25	28	9	56	20	97	86	77	2	2
2	69	22	15	83	95	87	33	98	3	3
3	48	48	21	7	87	83	51	64	4	4
4	75	92	82	97	51	46	78	14	5	5
5	37	11	55	68	50	23	46	26	6	6
6	24	27	59	5	73	98	31	98	7	7

In [231]:

```
# Making new column as index
a.set_index("new")
```

Out[231]:

```
      0   1   2   3   4   5   6   7  new
new
 34  19  78  97  56   7  41  21  41   34
  2  25  28   9  56  20  97  86  77    2
  3  69  22  15  83  95  87  33  98    3
  4  48  48  21   7  87  83  51  64    4
  5  75  92  82  97  51  46  78  14    5
  6  37  11  55  68  50  23  46  26    6
  7  24  27  59   5  73  98  31  98    7
```

In [232]:

```
a.reset_index()
```

Out[232]:

```
   index   0   1   2   3   4   5   6   7  new  new
0      0  19  78  97  56   7  41  21  41   34   34
1      1  25  28   9  56  20  97  86  77    2    2
2      2  69  22  15  83  95  87  33  98    3    3
3      3  48  48  21   7  87  83  51  64    4    4
4      4  75  92  82  97  51  46  78  14    5    5
5      5  37  11  55  68  50  23  46  26    6    6
6      6  24  27  59   5  73  98  31  98    7    7
```

In [233]:

```
a.head()
```

Out[233]:

```
    0   1   2   3   4   5   6   7  new  new
0  19  78  97  56   7  41  21  41   34   34
1  25  28   9  56  20  97  86  77    2    2
2  69  22  15  83  95  87  33  98    3    3
3  48  48  21   7  87  83  51  64    4    4
4  75  92  82  97  51  46  78  14    5    5
```

In [235]:

```
a.head(2)
```

Out[235]:

	0	1	2	3	4	5	6	7	new	new
0	19	78	97	56	7	41	21	41	34	34
1	25	28	9	56	20	97	86	77	2	2

In [236]:

```
a.tail(2)
```

Out[236]:

	0	1	2	3	4	5	6	7	new	new
5	37	11	55	68	50	23	46	26	6	6
6	24	27	59	5	73	98	31	98	7	7

In [237]:

```
# For sataistical information of the dataset
a.describe()
```

Out[237]:

	0	1	2	3	4	5	6	7
count	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000
mean	42.428571	43.714286	48.285714	53.142857	54.714286	67.857143	49.428571	59.714286
std	22.434030	30.532574	34.315761	35.352645	32.958126	30.454494	24.446248	33.727903
min	19.000000	11.000000	9.000000	5.000000	7.000000	23.000000	21.000000	14.000000
25%	24.500000	24.500000	18.000000	31.500000	35.000000	43.500000	32.000000	33.500000
50%	37.000000	28.000000	55.000000	56.000000	51.000000	83.000000	46.000000	64.000000
75%	58.500000	63.000000	70.500000	75.500000	80.000000	92.000000	64.500000	87.500000
max	75.000000	92.000000	97.000000	97.000000	95.000000	98.000000	86.000000	98.000000

In [238]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 10 columns):
0      7 non-null int32
1      7 non-null int32
2      7 non-null int32
3      7 non-null int32
4      7 non-null int32
5      7 non-null int32
6      7 non-null int32
7      7 non-null int32
new    7 non-null object
new    7 non-null object
dtypes: int32(8), object(2)
memory usage: 464.0+ bytes
```

In [239]:

```
a
```

Out[239]:

	0	1	2	3	4	5	6	7	new	new
0	19	78	97	56	7	41	21	41	34	34
1	25	28	9	56	20	97	86	77	2	2
2	69	22	15	83	95	87	33	98	3	3
3	48	48	21	7	87	83	51	64	4	4
4	75	92	82	97	51	46	78	14	5	5
5	37	11	55	68	50	23	46	26	6	6
6	24	27	59	5	73	98	31	98	7	7

In [245]:

```
a["new"].unique()
```

Out[245]:

```
array(['34', '2', '3', '4', '5', '6', '7'], dtype=object)
```

In [246]:

```
a["new"].value_counts()
```

Out[246]:

```
5     1
7     1
34    1
2     1
3     1
4     1
6     1
Name: new, dtype: int64
```

Missing Values

In [247]:

```
a.isnull().any
```

Out[247]:

```
<bound method DataFrame.any of      0      1      2      3      4      5      6      7    new    new
0  False  False  False  False  False  False  False  False  False  False
1  False  False  False  False  False  False  False  False  False  False
2  False  False  False  False  False  False  False  False  False  False
3  False  False  False  False  False  False  False  False  False  False
4  False  False  False  False  False  False  False  False  False  False
5  False  False  False  False  False  False  False  False  False  False
6  False  False  False  False  False  False  False  False  False  False>
```

In [248]:

```
a.isnull().any()
```

Out[248]:

```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
new    False
new    False
dtype: bool
```

In [250]:

```
a
```

Out[250]:

	0	1	2	3	4	5	6	7	new	new
0	19	78	97	56	7	41	21	41	34	34
1	25	28	9	56	20	97	86	77	2	2
2	69	22	15	83	95	87	33	98	3	3
3	48	48	21	7	87	83	51	64	4	4
4	75	92	82	97	51	46	78	14	5	5
5	37	11	55	68	50	23	46	26	6	6
6	24	27	59	5	73	98	31	98	7	7

Dropping Columns

In [253]:

```
a.drop('new', axis =1)
```

Out[253]:

	0	1	2	3	4	5	6	7	new
0	19	78	97	56	7	41	21	41	34
1	25	28	9	56	20	97	86	77	2
2	69	22	15	83	95	87	33	98	3
3	48	48	21	7	87	83	51	64	4
4	75	92	82	97	51	46	78	14	5
5	37	11	55	68	50	23	46	26	6
6	24	27	59	5	73	98	31	98	7

In [254]:
```
a.drop('new', axis =1)
```

Out[254]:

	0	1	2	3	4	5	6	7	new
0	19	78	97	56	7	41	21	41	34
1	25	28	9	56	20	97	86	77	2
2	69	22	15	83	95	87	33	98	3
3	48	48	21	7	87	83	51	64	4
4	75	92	82	97	51	46	78	14	5
5	37	11	55	68	50	23	46	26	6
6	24	27	59	5	73	98	31	98	7

In [255]:
```
a
```

Out[255]:

	0	1	2	3	4	5	6	7	new	new
0	19	78	97	56	7	41	21	41	34	34
1	25	28	9	56	20	97	86	77	2	2
2	69	22	15	83	95	87	33	98	3	3
3	48	48	21	7	87	83	51	64	4	4
4	75	92	82	97	51	46	78	14	5	5
5	37	11	55	68	50	23	46	26	6	6
6	24	27	59	5	73	98	31	98	7	7

In [258]:
```
a.drop("new", axis = 1, inplace = True) # axis = 1 -> To delete the rows only
```

In [264]:
```
a.drop(1)
```

Out[264]:

	0	1	2	3	4	5	6	7	new
0	19	78	97	56	7	41	21	41	34
2	69	22	15	83	95	87	33	98	3
3	48	48	21	7	87	83	51	64	4
4	75	92	82	97	51	46	78	14	5
5	37	11	55	68	50	23	46	26	6
6	24	27	59	5	73	98	31	98	7

In [259]:

```
a
```

Out[259]:

	0	1	2	3	4	5	6	7	new
0	19	78	97	56	7	41	21	41	34
1	25	28	9	56	20	97	86	77	2
2	69	22	15	83	95	87	33	98	3
3	48	48	21	7	87	83	51	64	4
4	75	92	82	97	51	46	78	14	5
5	37	11	55	68	50	23	46	26	6
6	24	27	59	5	73	98	31	98	7

In [266]:

```
  File "<ipython-input-266-518d9ca2d645>", line 1
    a = pd.DataFrame(np.random.rand(1,100,56).reshape(7,8) index = "a b c d e f g".split(), co
lumns = "p q h i j k l m".split())
                                                          ^
SyntaxError: invalid syntax
```

In [268]:

```
a = pd.DataFrame(np.random.randint(1,100,56).reshape(7,8) ,index = "a b c d e f g".split(), columns
```

In [269]:

```
a
```

Out[269]:

	p	q	h	i	j	k	l	m
a	67	75	2	65	19	43	21	14
b	6	44	76	68	2	77	84	15
c	33	4	91	91	42	89	77	47
d	23	61	35	91	78	1	13	2
e	99	90	27	69	38	46	17	77
f	40	85	6	20	2	39	91	53
g	98	24	44	46	16	81	15	36

In [271]:

```
a<6
```

Out[271]:

	p	q	h	i	j	k	l	m
a	False	False	True	False	False	False	False	False
b	False	False	False	False	True	False	False	False
c	False	True	False	False	False	False	False	False
d	False	False	False	False	False	True	False	True
e	False	False	False	False	False	False	False	False
f	False	False	False	False	True	False	False	False
g	False	False	False	False	False	False	False	False

In [274]:

```
dataset = a[a>6]
```

In [275]:

```
dataset
```

Out[275]:

	p	q	h	i	j	k	l	m
a	67.0	75.0	NaN	65	19.0	43.0	21	14.0
b	NaN	44.0	76.0	68	NaN	77.0	84	15.0
c	33.0	NaN	91.0	91	42.0	89.0	77	47.0
d	23.0	61.0	35.0	91	78.0	NaN	13	NaN
e	99.0	90.0	27.0	69	38.0	46.0	17	77.0
f	40.0	85.0	NaN	20	NaN	39.0	91	53.0
g	98.0	24.0	44.0	46	16.0	81.0	15	36.0

In [276]:

```
dataset.isnull().any
```

Out[276]:

```
<bound method DataFrame.any of       p      q      h      i      j      k      l      m
a  False  False   True  False  False  False  False  False
b   True  False  False  False   True  False  False  False
c  False   True  False  False  False  False  False  False
d  False  False  False  False  False   True  False   True
e  False  False  False  False  False  False  False  False
f  False  False   True  False   True  False  False  False
g  False  False  False  False  False  False  False  False>
```

In [277]:

```
dataset.fillna(dataset["q"].mean(),inplace= True)
```

In [278]:

```
dataset
```

Out[278]:

	p	q	h	i	j	k	l	m
a	67.000000	75.000000	63.166667	65	19.000000	43.000000	21	14.000000
b	63.166667	44.000000	76.000000	68	63.166667	77.000000	84	15.000000
c	33.000000	63.166667	91.000000	91	42.000000	89.000000	77	47.000000
d	23.000000	61.000000	35.000000	91	78.000000	63.166667	13	63.166667
e	99.000000	90.000000	27.000000	69	38.000000	46.000000	17	77.000000
f	40.000000	85.000000	63.166667	20	63.166667	39.000000	91	53.000000
g	98.000000	24.000000	44.000000	46	16.000000	81.000000	15	36.000000

In [280]:

```
dataset.fillna(dataset["h"].median(),inplace= True)
```

In [281]:

```
dataset
```

Out[281]:

	p	q	h	i	j	k	l	m
a	67.000000	75.000000	63.166667	65	19.000000	43.000000	21	14.000000
b	63.166667	44.000000	76.000000	68	63.166667	77.000000	84	15.000000
c	33.000000	63.166667	91.000000	91	42.000000	89.000000	77	47.000000
d	23.000000	61.000000	35.000000	91	78.000000	63.166667	13	63.166667
e	99.000000	90.000000	27.000000	69	38.000000	46.000000	17	77.000000
f	40.000000	85.000000	63.166667	20	63.166667	39.000000	91	53.000000
g	98.000000	24.000000	44.000000	46	16.000000	81.000000	15	36.000000

In [282]:

```
dataset.fillna(dataset["m"].median(),inplace= True)
```

In [283]:

```
dataset
```

Out[283]:

	p	q	h	i	j	k	l	m
a	67.000000	75.000000	63.166667	65	19.000000	43.000000	21	14.000000
b	63.166667	44.000000	76.000000	68	63.166667	77.000000	84	15.000000
c	33.000000	63.166667	91.000000	91	42.000000	89.000000	77	47.000000
d	23.000000	61.000000	35.000000	91	78.000000	63.166667	13	63.166667
e	99.000000	90.000000	27.000000	69	38.000000	46.000000	17	77.000000
f	40.000000	85.000000	63.166667	20	63.166667	39.000000	91	53.000000
g	98.000000	24.000000	44.000000	46	16.000000	81.000000	15	36.000000

In [284]:

```
dataset.isnull().any
```

Out[284]:

```
<bound method DataFrame.any of        p     q     h     i     j     k     l     m
a  False  False  False  False  False  False  False  False
b  False  False  False  False  False  False  False  False
c  False  False  False  False  False  False  False  False
d  False  False  False  False  False  False  False  False
e  False  False  False  False  False  False  False  False
f  False  False  False  False  False  False  False  False
g  False  False  False  False  False  False  False  False>
```

In [305]:

```
# Mode
p = pd.DataFrame({"name": ["Srinivas", "Rahul", np.NaN, np.NaN, "Srinivas", "Sapi","Sapi"]})
```

In [306]:

```
p
```

Out[306]:

	name
0	Srinivas
1	Rahul
2	NaN
3	NaN
4	Srinivas
5	Sapi
6	Sapi

In [307]:

```
p.isnull().any
```

Out[307]:

```
<bound method DataFrame.any of     name
0  False
1  False
2   True
3   True
4  False
5  False
6  False>
```

In [308]:

```
p["name"].mode()
```

Out[308]:

```
0       Sapi
1   Srinivas
dtype: object
```

In [312]:

```
p.fillna(p["name"].mode()[0], inplace = True)
```

In [313]:

```
p
```

Out[313]:

	name
0	Srinivas
1	Rahul
2	Sapi
3	Sapi
4	Srinivas
5	Sapi
6	Sapi

Visualization - Matplotlib

In [314]:

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

In [323]:

```
x = np.arange(10)
```

In [324]:

```
x
```

Out[324]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [327]:

```
y = np.random.randint(10,20,10)
```

In [328]:

```
y
```

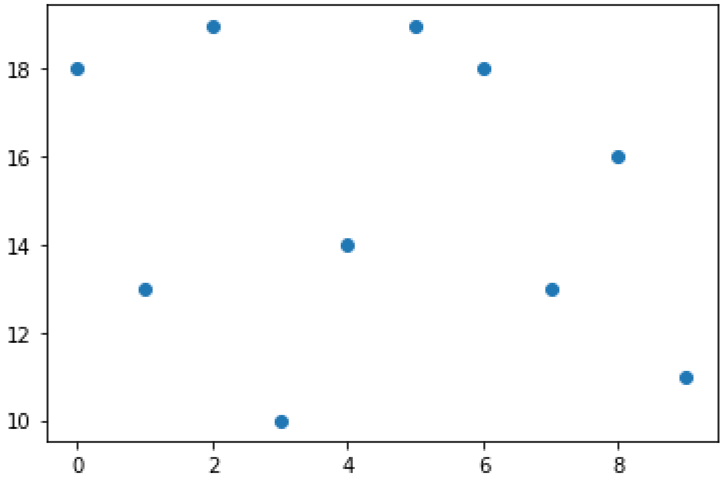
Out[328]:

```
array([18, 13, 19, 10, 14, 19, 18, 13, 16, 11])
```

Scatter Plot

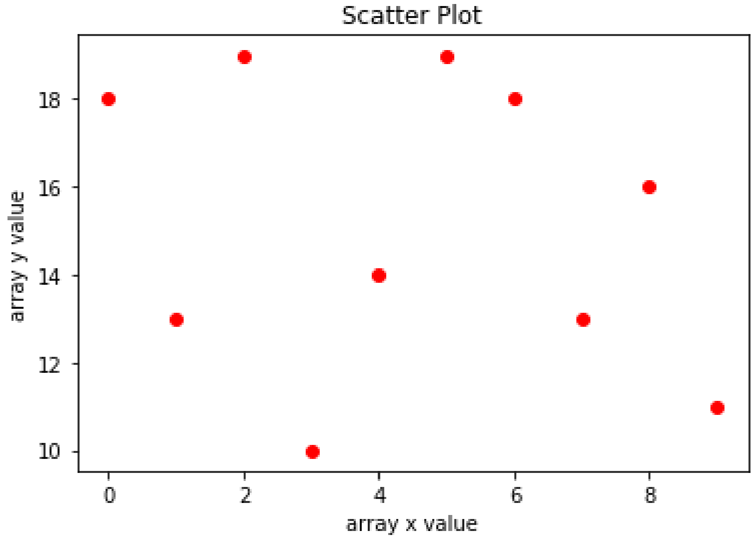
In [331]:

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



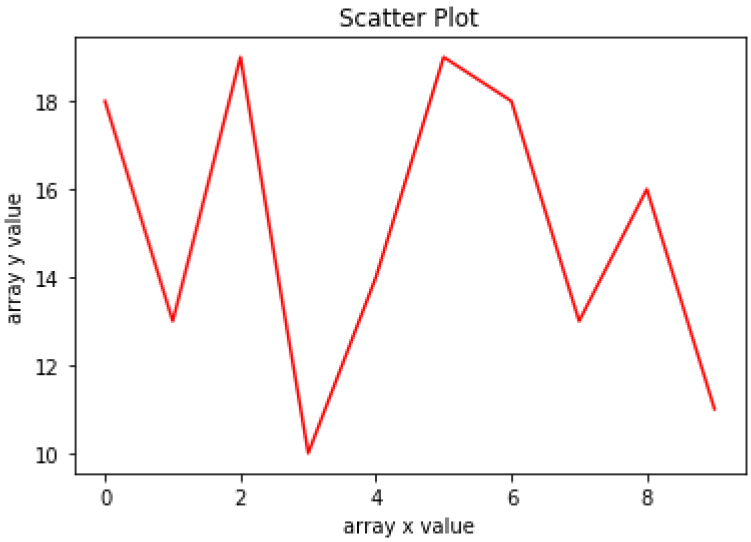
In [335]:

```
plt.scatter(x,y, color = "r")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Scatter Plot")
plt.show()
```



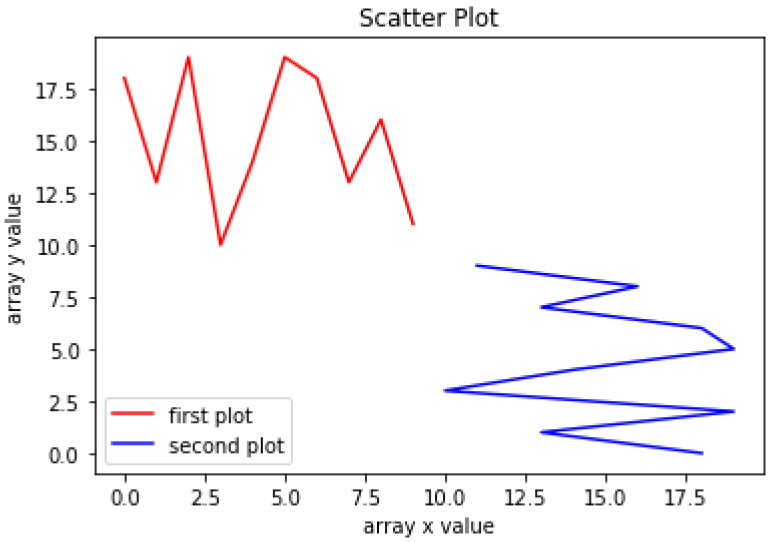
In [336]:

```
plt.plot(x,y, color = "r")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Scatter Plot")
plt.show()
```



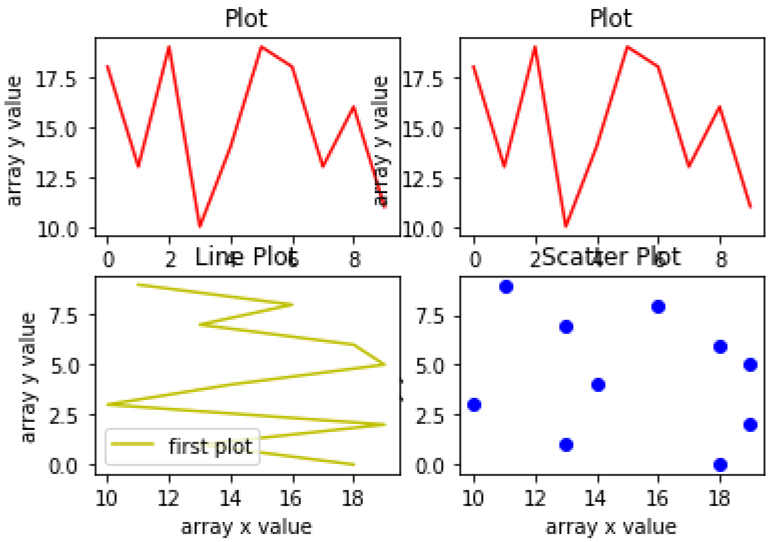
In [343]:

```
plt.plot(x,y, color = "r", label = "first plot")
plt.plot(y,x, color = "b", label = "second plot")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Scatter Plot")
plt.legend(loc = "lower left")
plt.show()
```



In [352]:

```
plt.subplot(2,2,1)
plt.plot(x,y, color = "r", label = "first plot")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Plot")
plt.subplot(2,2,2)
plt.plot(x,y, color = "r", label = "first plot")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Plot")
plt.subplot(2,2,4)
plt.scatter(y,x, color = "b", label = "second plot")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Scatter Plot")
plt.subplot(2,2,3)
plt.plot(y,x, color = "y", label = "first plot")
plt.xlabel("array x value")
plt.ylabel("array y value")
plt.title("Line Plot")
plt.legend(loc = "lower left")
plt.show()
```



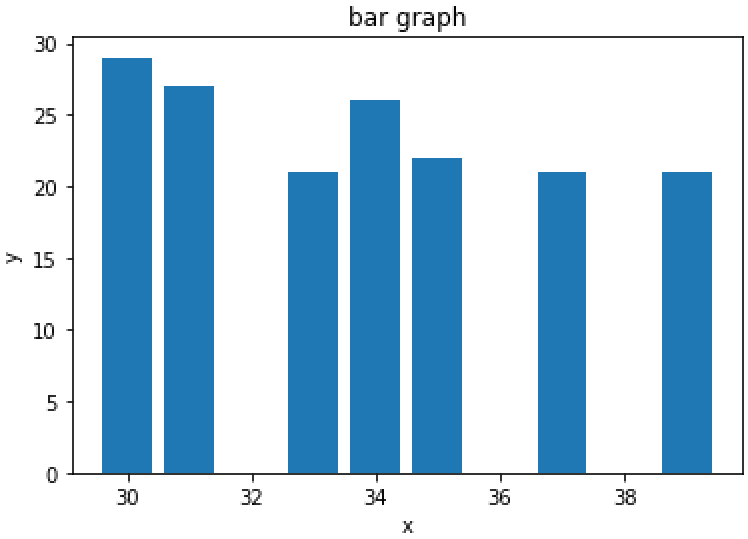
Bar Plot

In [353]:

```
a = np.random.randint(30,40,10)
b = np.random.randint(20,30,10)
```

In [354]:

```
plt.bar(a,b)
plt.xlabel("x")
plt.ylabel("y")
plt.title("bar graph")
plt.show()
```



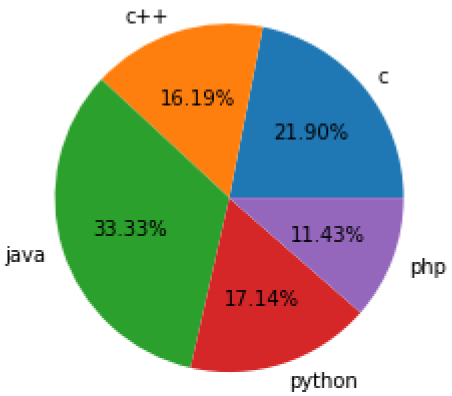
Pie Chart

In [355]:

```
languages = ["c", "c++", "java", "python", "php"]
students = [23, 17, 35, 18, 12]
```

In [359]:

```
plt.pie(students,  labels = languages, autopct = "%1.2f%%")
plt.show()
```



Visualization - Seaborn

In [389]:

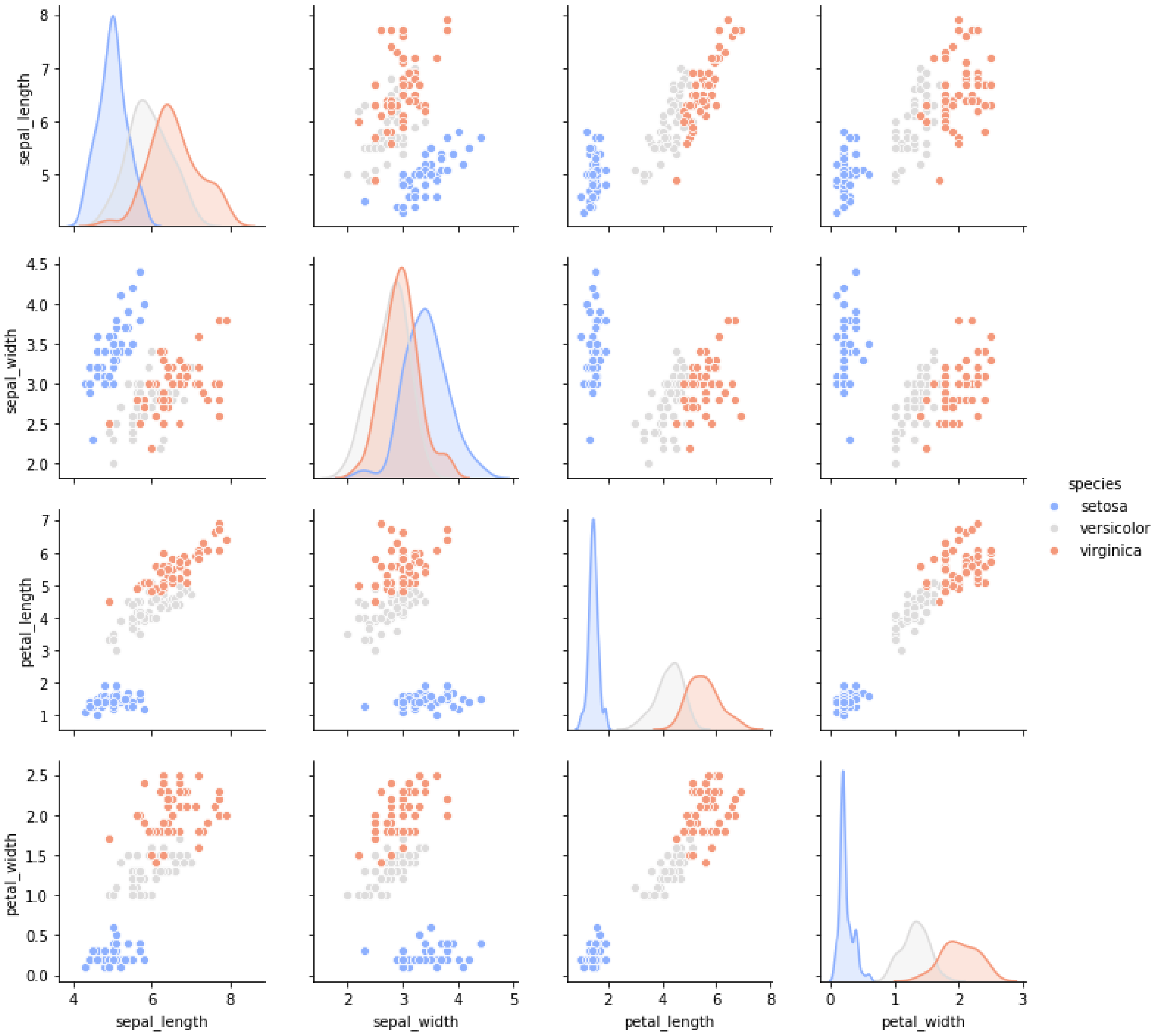
```
import seaborn as sns
iris = sns.load_dataset("iris")
```

In [390]:

```
# Subplot grid for more flexible plotting of pairwise relationships.
sns.pairplot(iris, hue = "species", palette = "coolwarm")
```

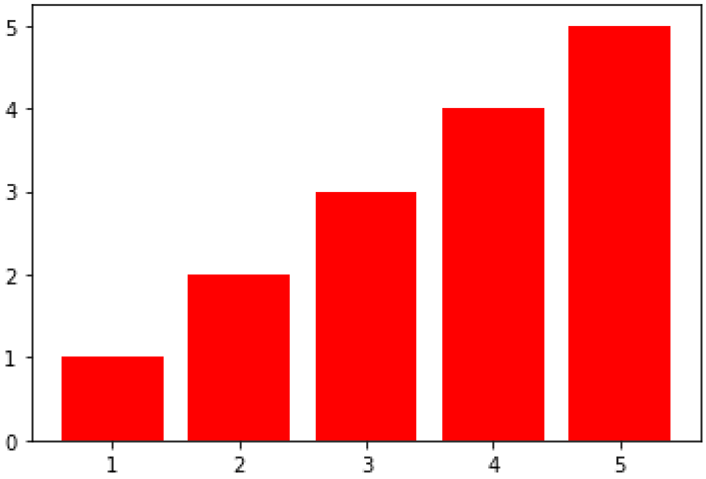
Out[390]:

```
<seaborn.axisgrid.PairGrid at 0x1980c14c048>
```



In [391]:

```
x = [1,2,3,4,5]
y = [1,2,3,4,5]
plt.bar(x,y, color = "red")
plt.show()
```



In [383]:

```
iris
```

Out[383]:

```
{'data': array([[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],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

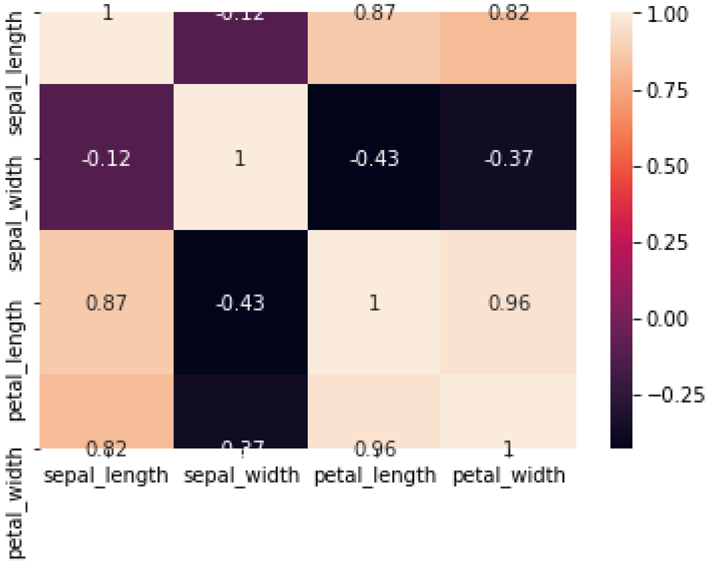
Heat Map

In [392]:

```
sns.heatmap(iris.corr(), annot = True)
```

Out[392]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1980da321c8>
```



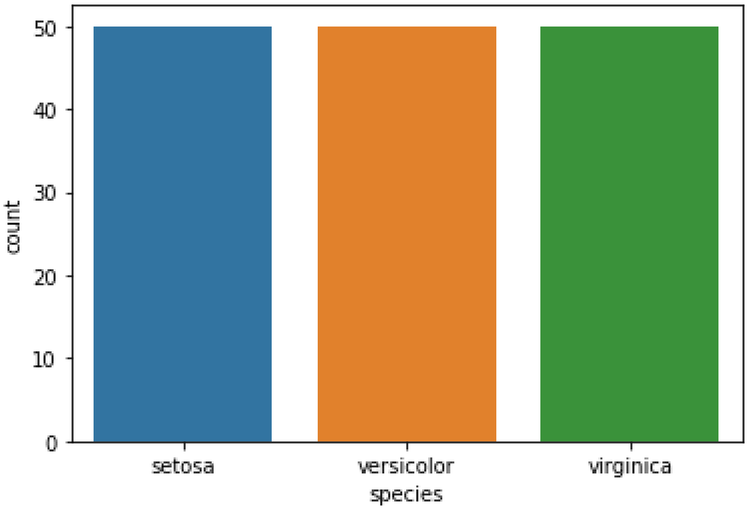
Count Plot -> Categorical Data

In [394]:

```
sns.countplot(x = "species", data = iris)
```

Out[394]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1980db58208>
```



In [397]:

```
iris["species"].value_counts()
```

Out[397]:

```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

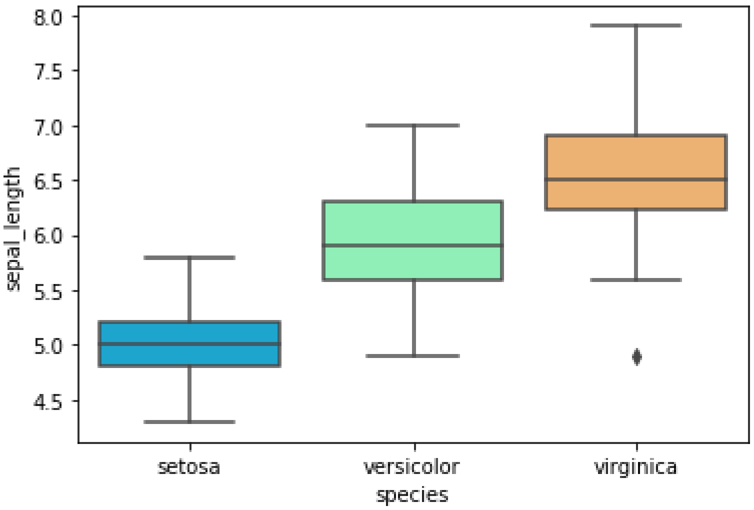
Box Plot -> Gives the distribution of the categorical data

In [405]:

```
sns.boxplot(x = "species", y = "sepal_length", data = iris ,palette = "rainbow")
```

Out[405]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1980dbc0088>
```



In [404]:

```
iris
```

Out[404]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

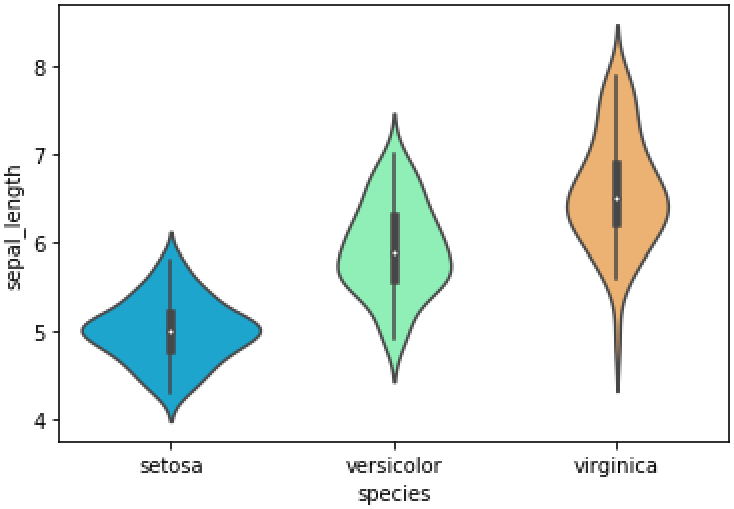
Violin Plot

In [406]:

```
sns.violinplot(x = "species", y = "sepal_length", data = iris ,palette = "rainbow")
```

Out[406]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1980dc53808>
```



In [ ]:

```
```