Experiment 6

Frason Francis / 201903020 / SE-IT

a) Different types of plots using Numpy and Matplotlob

b)Basic operations using pandas like series, data frames, indexing, filtering, combining and merging data frames.

THEORY:

- Matplotlib is a plotting library for Python. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython

- Matplotlib is the most powerful visualization library. To use it, you need to import its sub package pyplot.

- To get something productive from that data, we need to do some statistical analysis on it, and Numpy is a great tool for that. The various statistical operations include mean, median, standard deviation, variance, etc.

- All these operations can either be done on the whole array, or can be done on just the rows or the columns which can be observed on below jupyter notebook.

Pandas Data Visualization

Frason Francis / SE-IT / 201903020

In [81]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [82]:

```
from numpy.random import randn, randint, uniform, sample
```

In [85]:

```
df = pd.DataFrame(randn(1000), index = pd.date_range('2019-06-07', periods = 1000), columns
ts = pd.Series(randn(1000), index = pd.date_range('2019-06-07', periods = 1000))
df.head()
```

Out[85]:

	value
2019-06-07	1.232051
2019-06-08	-0.261482
2019-06-09	1.242395
2019-06-10	0.895954
2019-06-11	0.736072

In [87]:

```
df['value'] = df['value'].cumsum()
df.head()
```

Out[87]:

	value
2019-06-07	1.232051
2019-06-08	2.202620
2019-06-09	4.415583
2019-06-10	7.524501
2019-06-11	11.369490

In [88]:

```
ts = ts.cumsum()
ts.head()
```

Out[88]:

```
2019-06-07   -0.173462
2019-06-08   -1.410669
2019-06-09   -1.760961
2019-06-10   -1.244528
2019-06-11   -0.363838
Freq: D, dtype: float64
```

In [89]:

```
type(df), type(ts)
```

Out[89]:

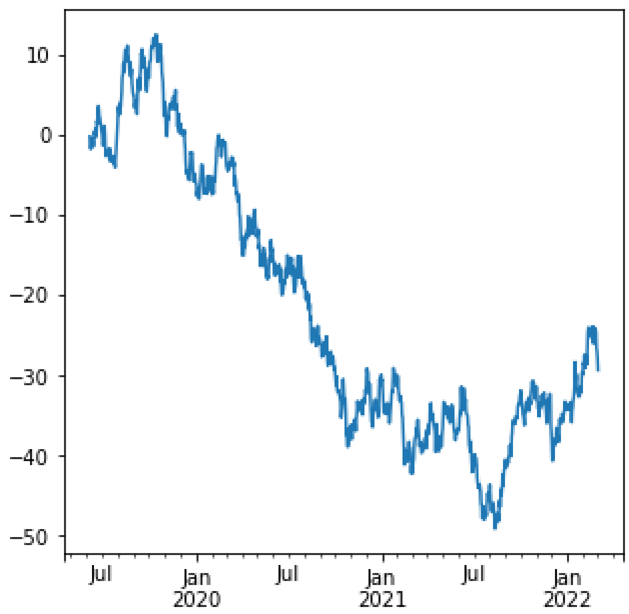
```
(pandas.core.frame.DataFrame, pandas.core.series.Series)
```

In [92]:

```
ts.plot(figsize=(5,5))
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ac85a20>
```

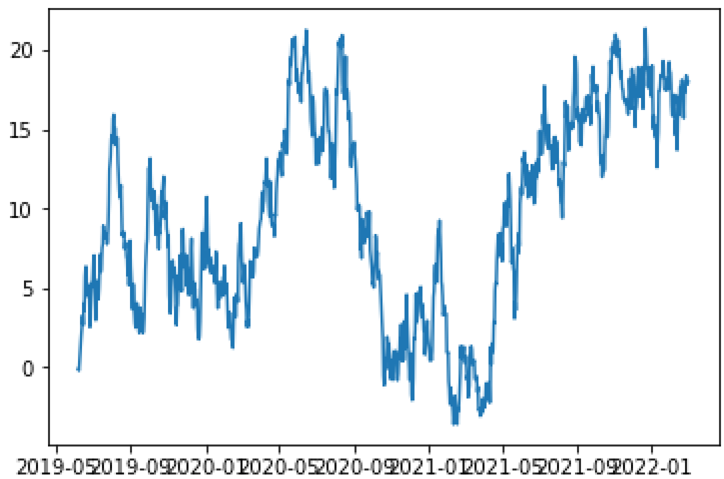


In [8]:

```
plt.plot(ts)
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x1a1ac73668>]
```

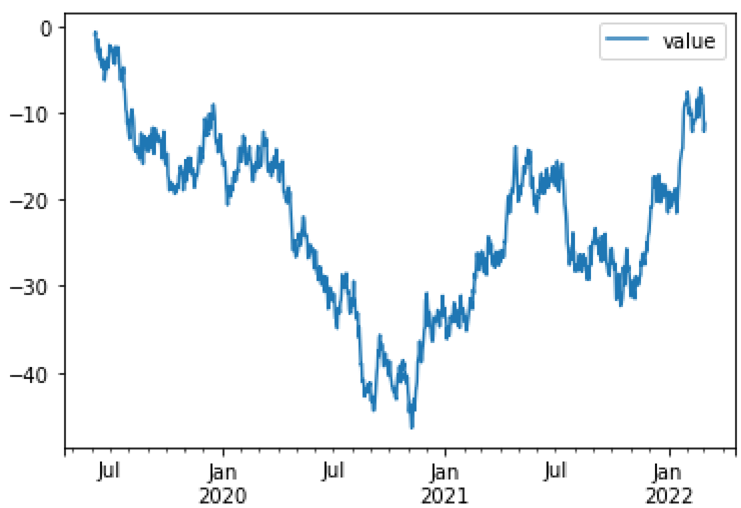


In [9]:

```
df.plot()
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1aa34160>
```



In [10]:

```
iris = sns.load_dataset('iris')
iris.head()
```

Out[10]:

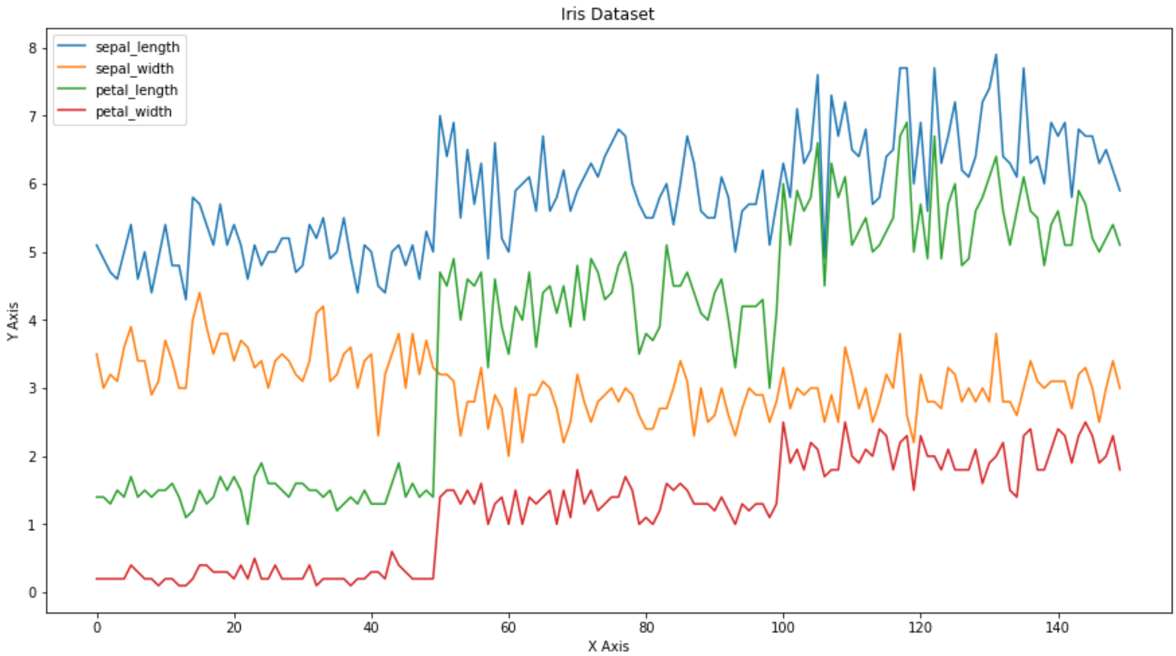
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [11]:

```
ax = iris.plot(figsize=(15,8), title='Iris Dataset')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
```

Out[11]:

```
Text(0, 0.5, 'Y Axis')
```

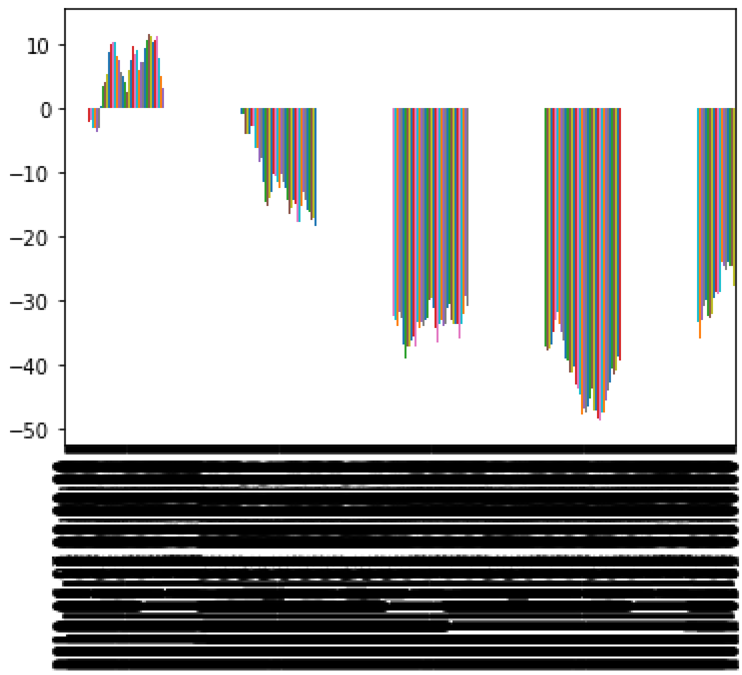


In [94]:

```
ts.plot(kind = 'bar')
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ae7d080>
```



In [95]:

```
df = iris.drop(['species'], axis = 1)
```

In [97]:

```
df.iloc[0]
```

Out[97]:

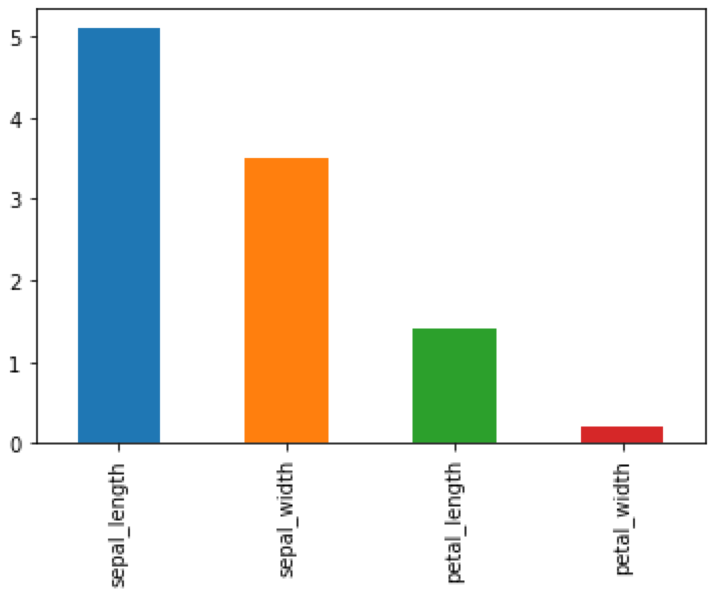
```
sepal_length    5.1
sepal_width     3.5
petal_length    1.4
petal_width     0.2
Name: 0, dtype: float64
```

In [15]:

```
df.iloc[0].plot(kind='bar')
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1bbc4dd8>
```

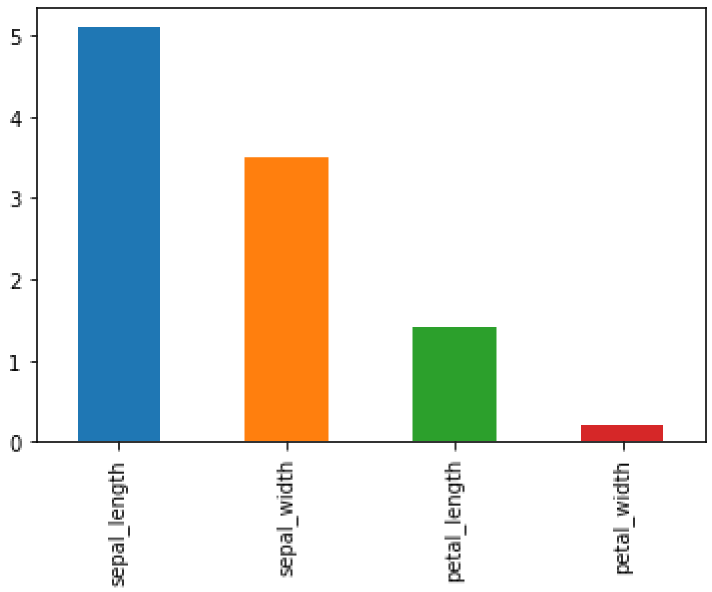


In [16]:

```
df.iloc[0].plot.bar()
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c99e940>
```



In [98]:

```
titanic = sns.load_dataset('titanic')
```

In [99]:

```
titanic.head()
```

Out[99]:

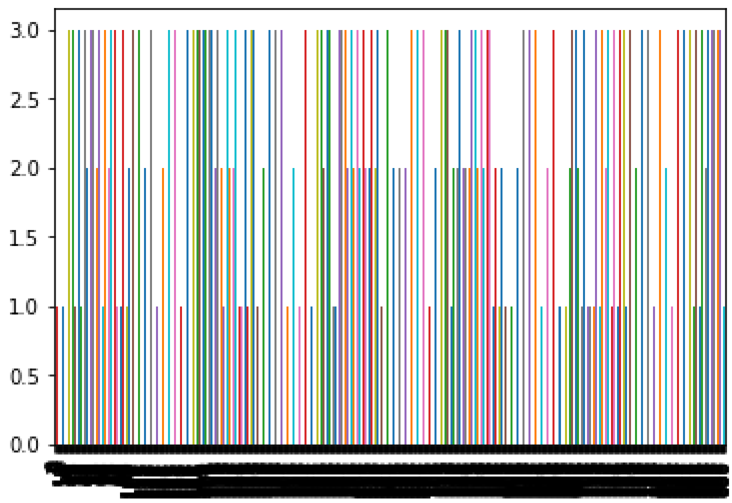
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

In [100]:

```
titanic['pclass'].plot(kind = 'bar')
```

Out[100]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22dee6d8>
```



In [101]:

```
df = pd.DataFrame(randn(10, 4), columns=['a', 'b', 'c', 'd'])
df.head(10)
```

Out[101]:

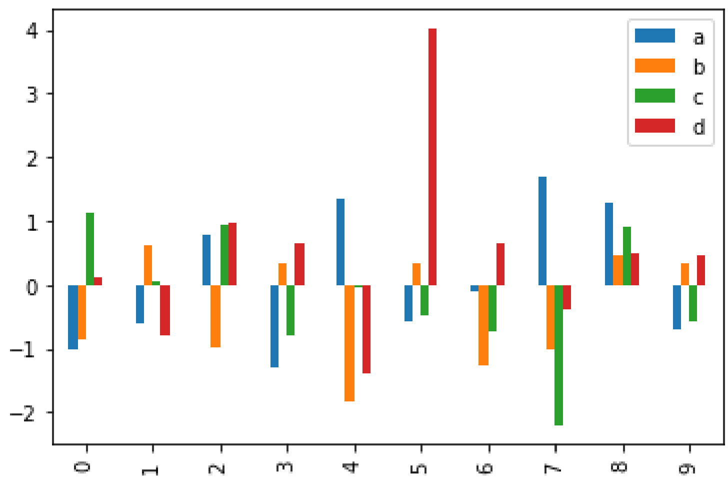
	a	b	c	d
0	0.326203	2.331026	-0.454617	0.107559
1	0.468923	1.185780	0.008878	0.723547
2	2.054247	1.828960	1.536323	-1.792616
3	0.170623	0.640836	1.402193	0.045841
4	0.009997	-0.727844	0.079510	-1.533088
5	-0.197923	0.135551	1.871942	1.361573
6	0.798528	-0.079833	1.438415	0.397582
7	0.995109	-1.384738	-0.012644	-1.937791
8	1.436894	-0.254240	-0.760523	-0.523546
9	-0.210206	-0.766180	-1.179217	-1.280725

In [21]:

```
df.plot.bar()
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c994860>
```

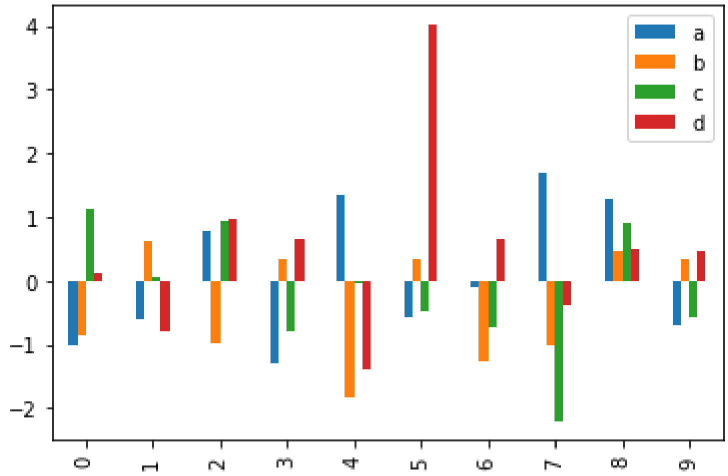


In [22]:

```
df.plot(kind = 'bar')
```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1c9dc9e8>

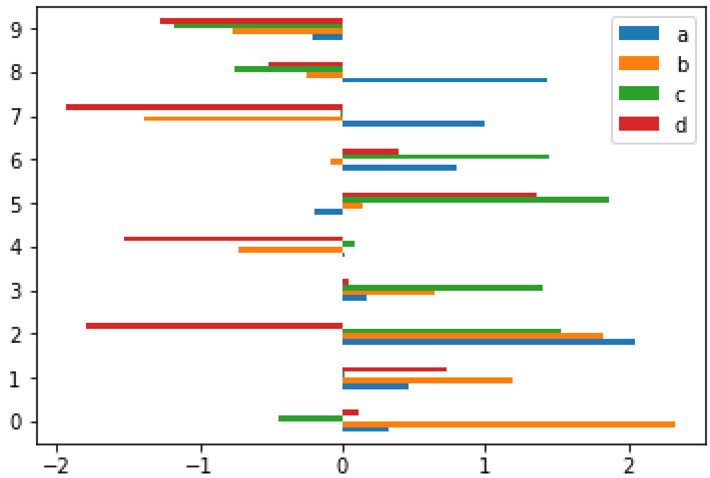


In [102]:

```
df.plot.barh()
```

Out[102]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a2b6de208>

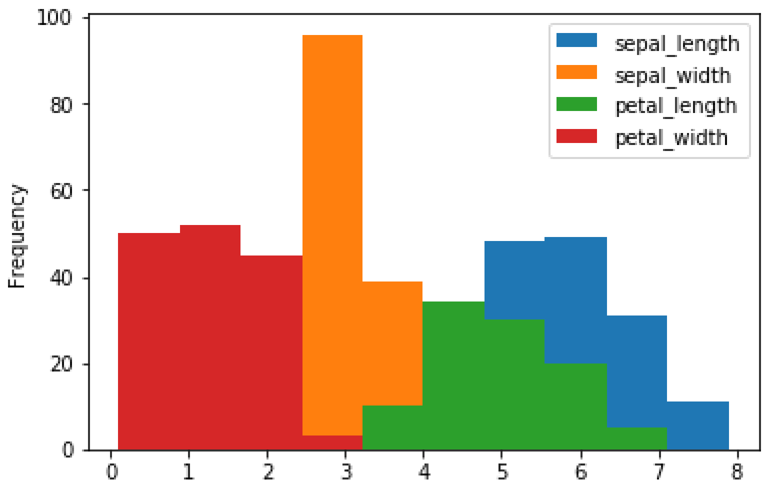


In [103]:

```
iris.plot.hist()
```

Out[103]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a2c1e9c18>

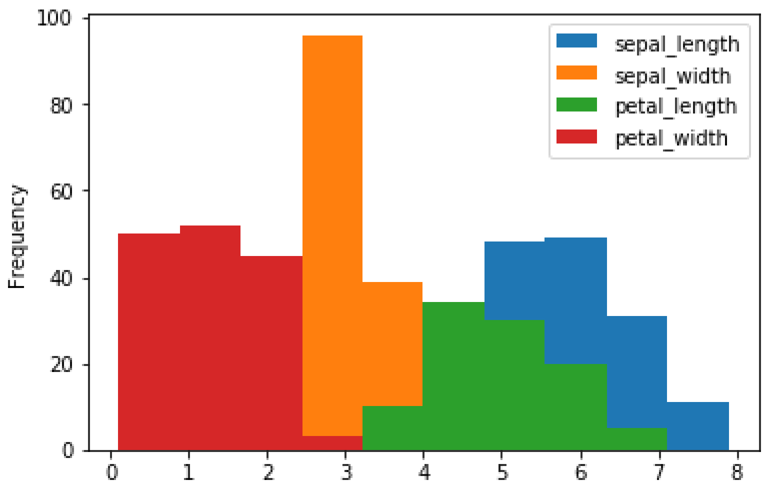


In [25]:

```
iris.plot(kind = 'hist')
```

Out[25]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1d22ec50>

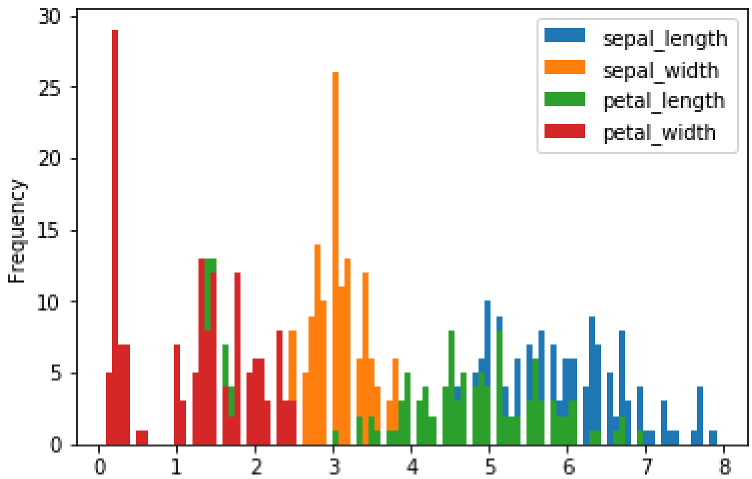


In [110]:

```
iris.plot(kind = 'hist', stacked = False, bins = 100)
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2d5a3b70>
```

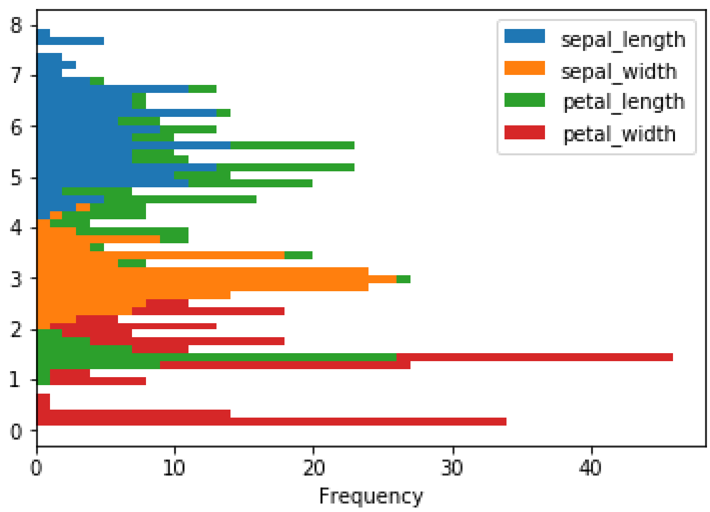


In [111]:

```
iris.plot(kind = 'hist', stacked = True, bins = 50, orientation = 'horizontal')
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2da82320>
```



In [112]:

```
iris['sepal_width'].diff()
```

Out[112]:

```
0       NaN
1      -0.5
2       0.2
3      -0.1
4       0.5
5       0.3
6      -0.5
7       0.0
8      -0.5
9       0.2
10      0.6
11     -0.3
12     -0.4
13      0.0
14      1.0
15      0.4
16     -0.5
17     -0.4
18      0.3
19      0.0
20     -0.4
21      0.3
22     -0.1
23     -0.3
24      0.1
25     -0.4
26      0.4
27      0.1
28     -0.1
29     -0.2
       ...
120     1.0
121    -0.4
122     0.0
123    -0.1
124     0.6
125    -0.1
126    -0.4
127     0.2
128    -0.2
129     0.2
130    -0.2
131     1.0
132    -1.0
133     0.0
134    -0.2
135     0.4
136     0.4
137    -0.3
138    -0.1
139     0.1
140     0.0
141     0.0
142    -0.4
```

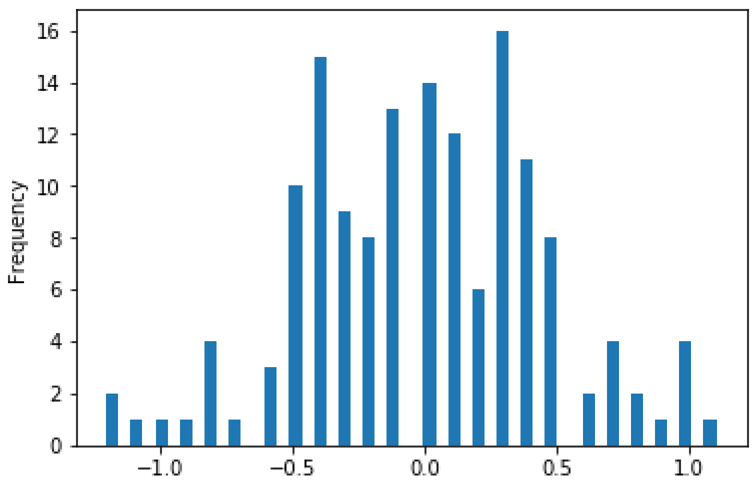
```
143    0.5
144    0.1
145   -0.3
146   -0.5
147    0.5
148    0.4
149   -0.4
Name: sepal_width, Length: 150, dtype: float64
```

In [113]:

```
iris['sepal_width'].diff().plot(kind = 'hist', stacked = True, bins = 50)
```

Out[113]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2dcbc198>
```



In [ ]:

```

```

In [118]:

```
df = iris.drop(['species'], axis = 1)
df.diff().head()
```

Out[118]:

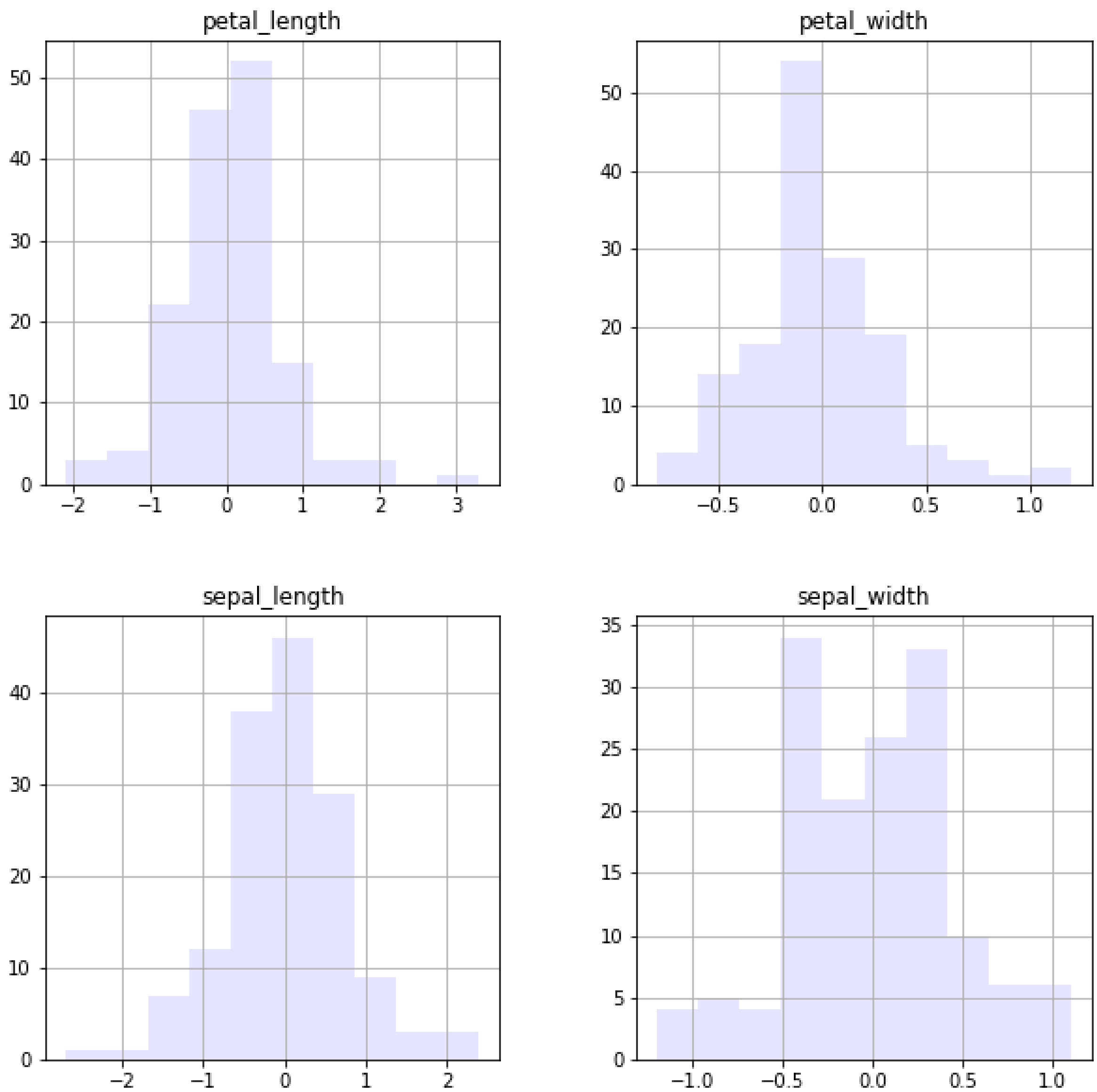
	sepal_length	sepal_width	petal_length	petal_width
0	NaN	NaN	NaN	NaN
1	-0.2	-0.5	0.0	0.0
2	-0.2	0.2	-0.1	0.0
3	-0.1	-0.1	0.2	0.0
4	0.4	0.5	-0.1	0.0

In [121]:

```
df.diff().hist(color = 'b', alpha = 0.1, figsize=(10,10))
```

Out[121]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a2dcbceb8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2eace7b8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a2eaf3d30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2eb242e8>]],
      dtype=object)
```



In [122]:

```
color = {'boxes': 'DarkGreen', 'whiskers': 'b'}
color
```

Out[122]:

```
{'boxes': 'DarkGreen', 'whiskers': 'b'}
```

In [ ]:

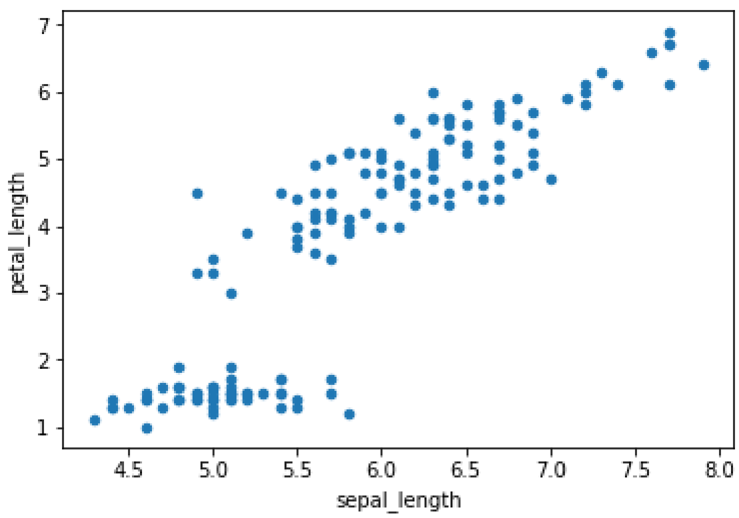
```
```

In [123]:

```
df.plot.scatter(x = 'sepal_length', y = 'petal_length')
```

Out[123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ebce518>
```

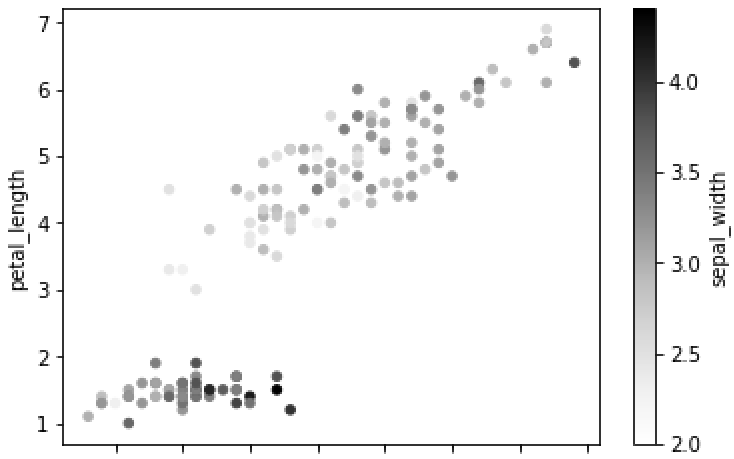


In [125]:

```
df.plot.scatter(x = 'sepal_length', y = 'petal_length', c = 'sepal_width')
```

Out[125]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ef4ec18>
```



In [126]:

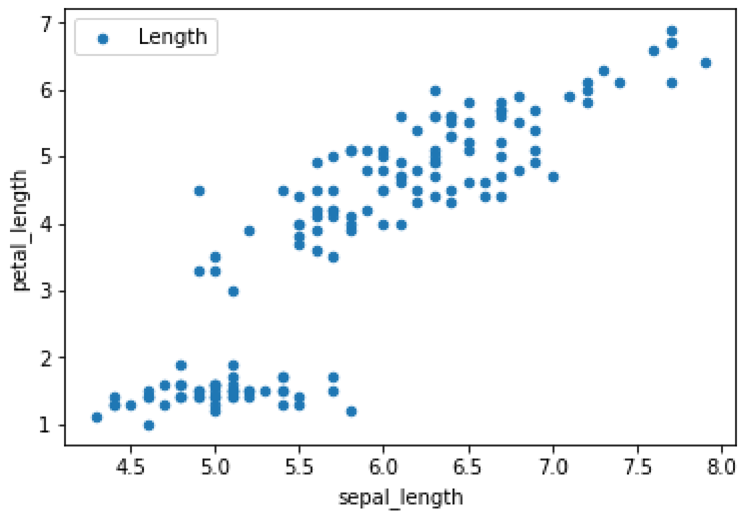
```
df.head()
```

Out[126]:

	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

In [131]:

```
df.plot.scatter(x = 'sepal_length', y = 'petal_length', label = 'Length');
#df.plot.scatter(x = 'sepal_width', y = 'petal_width', label = 'Width', ax = ax, color = 'r
#df.plot.scatter(x = 'sepal_width', y = 'petal_length', label = 'Width', ax = ax, color = '
```



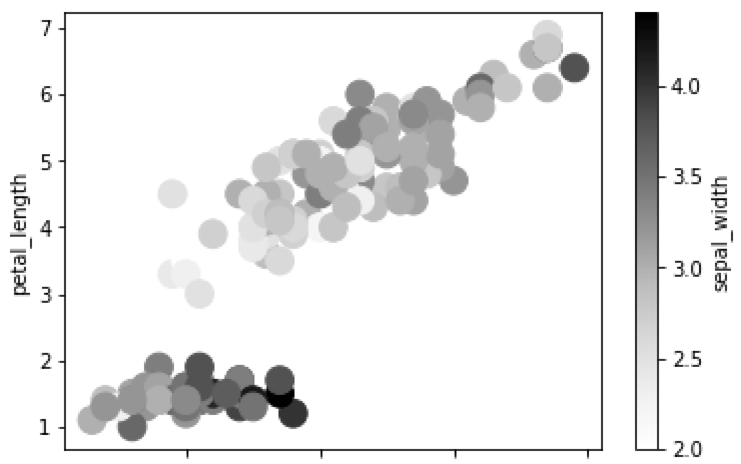
In [ ]:

In [133]:

```
df.plot.scatter(x = 'sepal_length', y = 'petal_length', c = 'sepal_width', s = 190)
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2f5fde80>
```



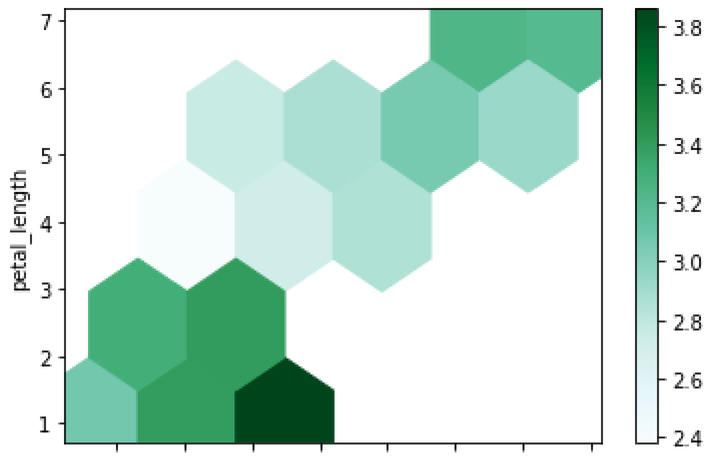
In [ ]:

In [135]:

```
df.plot.hexbin(x = 'sepal_length', y = 'petal_length', gridsize = 5, C = 'sepal_width')
```

Out[135]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2f883b00>
```



In [ ]:

```

```

In [136]:

```
d = df.iloc[0]
d
```

Out[136]:

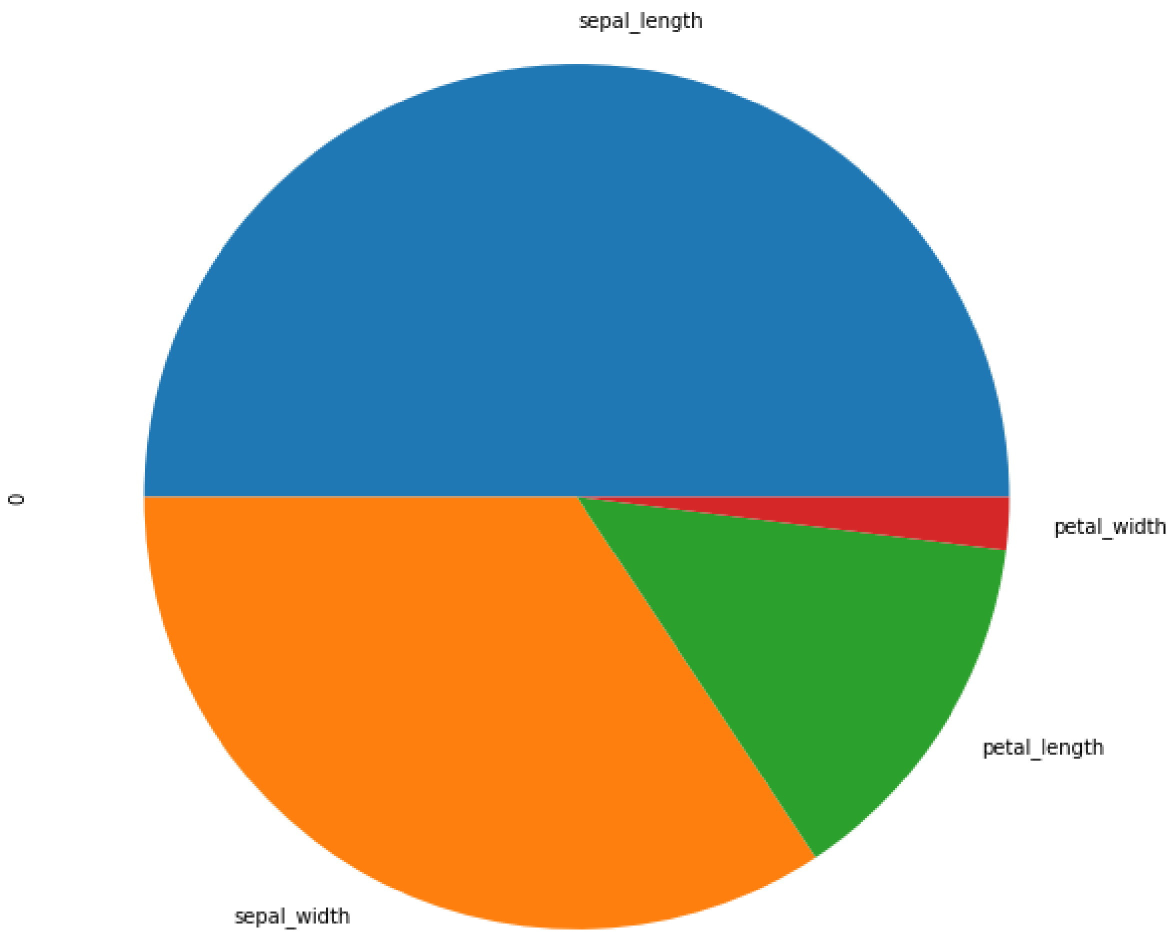
```
sepal_length    5.1
sepal_width     3.5
petal_length    1.4
petal_width     0.2
Name: 0, dtype: float64
```

In [39]:

```
d.plot.pie(figsize = (10,10))
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e0fea58>
```



In [137]:

```
d = df.head(3).T
```

In [138]:

```
d
```

Out[138]:

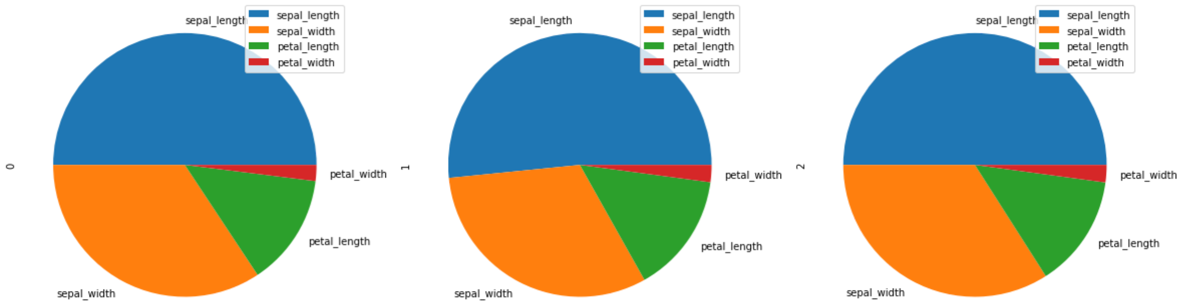
	0	1	2
sepal_length	5.1	4.9	4.7
sepal_width	3.5	3.0	3.2
petal_length	1.4	1.4	1.3
petal_width	0.2	0.2	0.2

In [140]:

```
d.plot.pie(subplots = True, figsize = (20, 20))
```

Out[140]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1a2f7d3f28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a2fa52e80>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a2fa81320>],
      dtype=object)
```

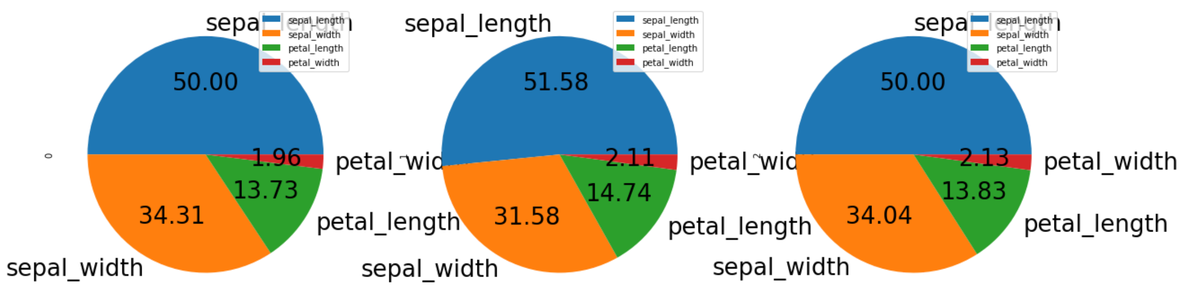


In [142]:

```
d.plot.pie(subplots = True, figsize = (20, 20), fontsize = 26, autopct = '%.2f')
```

Out[142]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1a303d2630>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a3040fcf8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a30662198>],
      dtype=object)
```



In [44]:

```
[0.1]*4
```

Out[44]:

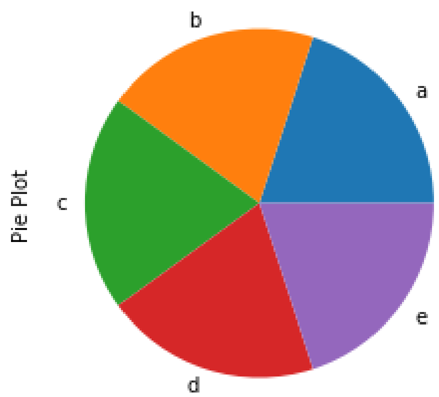
```
[0.1, 0.1, 0.1, 0.1]
```

In [144]:

```
series = pd.Series([0.2]*5, index = ['a','b','c', 'd','e'], name = 'Pie Plot')
series.plot.pie()
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a306f4ba8>
```



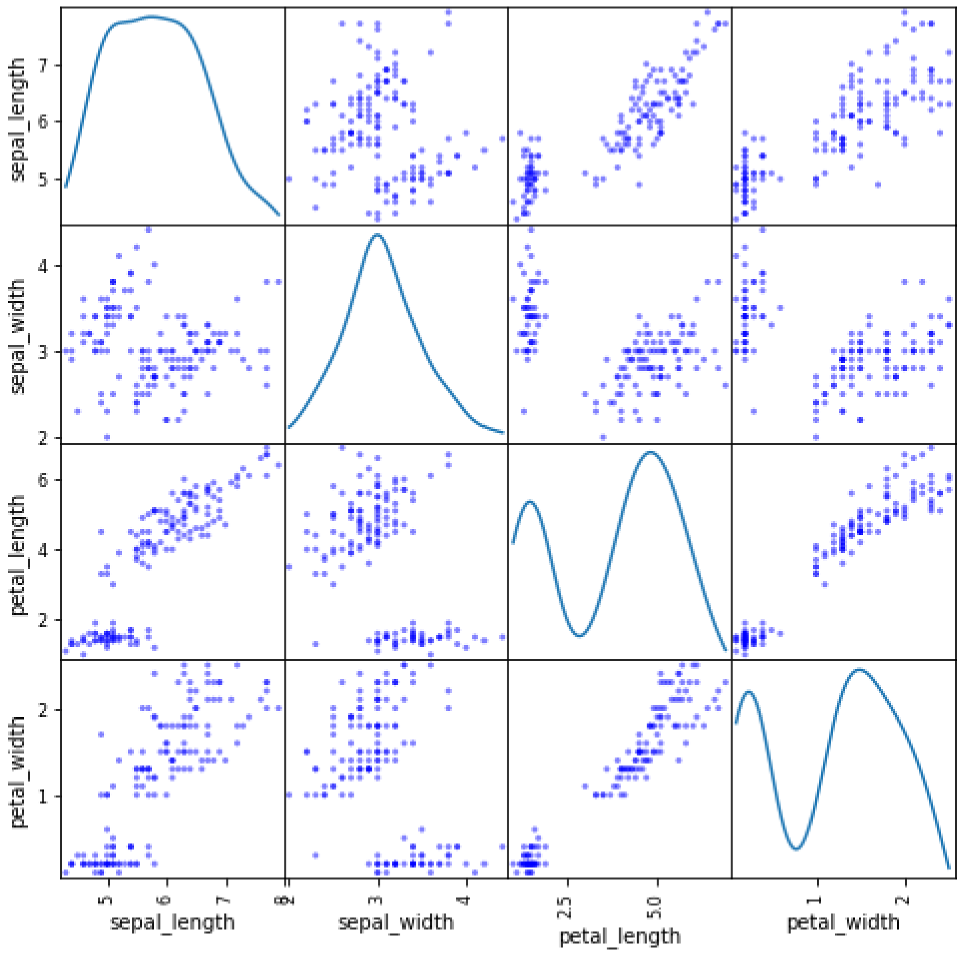
In [ ]:

In [46]:

```
from pandas.plotting import scatter_matrix
```

In [47]:

```
scatter_matrix(df, figsize= (8,8), diagonal='kde', color = 'b')
plt.show()
```

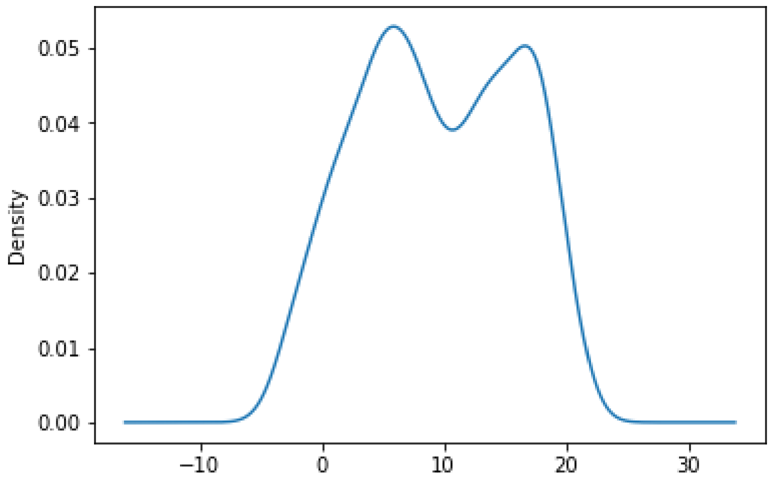


In [48]:

```
ts.plot.kde()
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f41df98>
```



In [ ]:

```

```

In [49]:

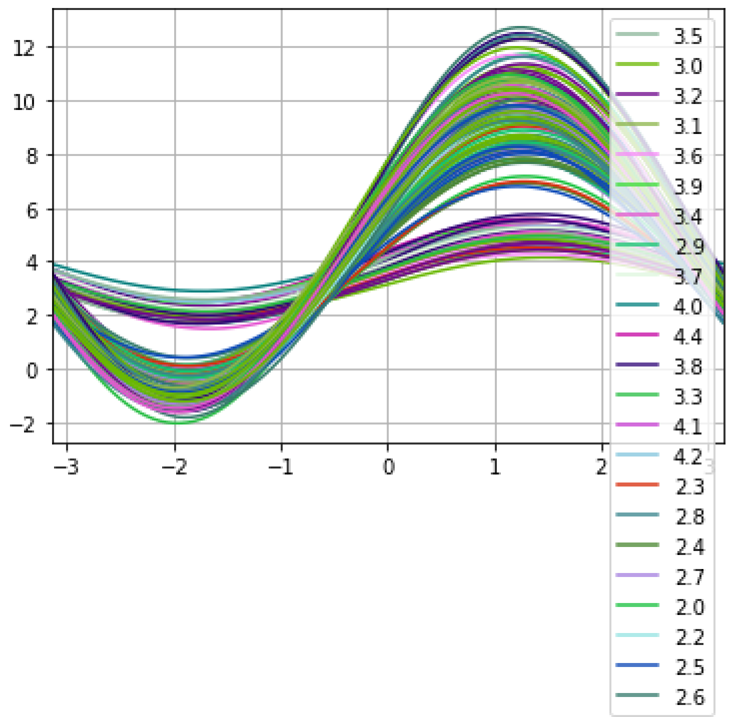
```
from pandas.plotting import andrews_curves
```

In [50]:

```
andrews_curves(df, 'sepal_width')
```

Out[50]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1eda8dd8>



In [ ]:

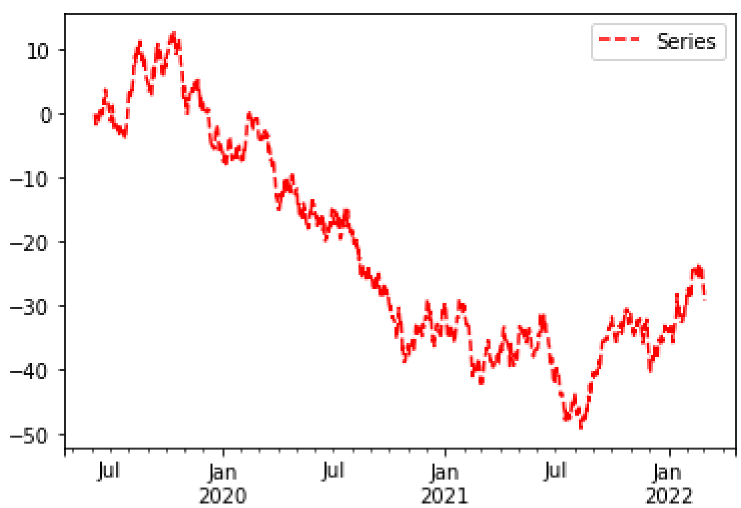
```

```

In [145]:

```
ts.plot(style = 'r--', label = 'Series', legend = True)
```

Out[145]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a30949f28>

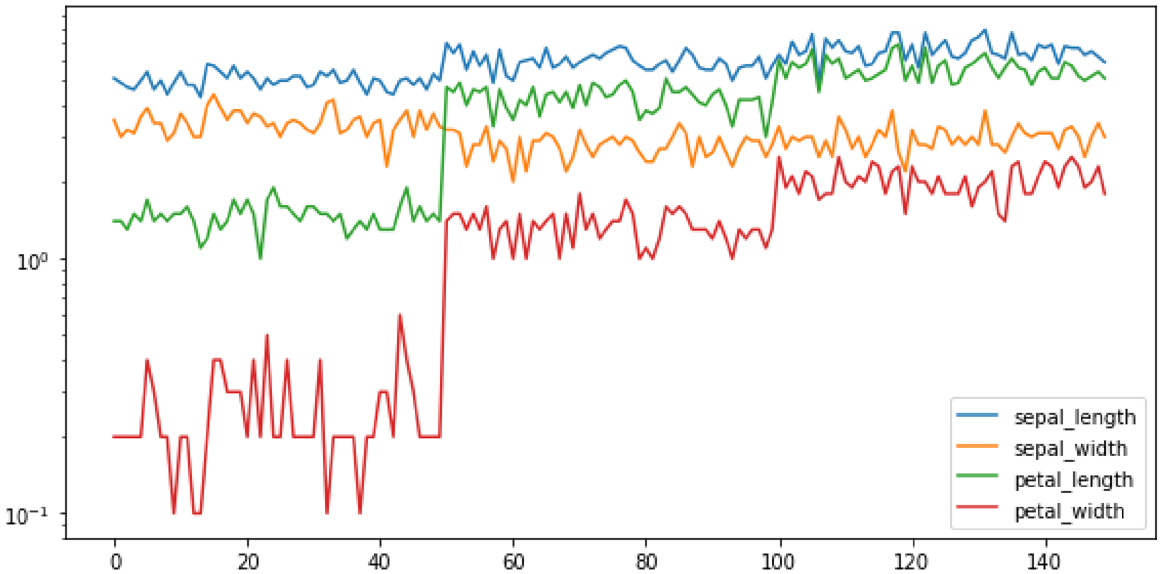


In [146]:

```
df.plot(legend = True, figsize = (10, 5), logy = True)
```

Out[146]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a30a13ac8>
```



In [53]:

```
df.head(0)
```

Out[53]:

sepal_length	sepal_width	petal_length	petal_width

In [54]:

```
x = df.drop(['sepal_width', 'petal_width'], axis = 1)
x.head()
```

Out[54]:

	sepal_length	petal_length
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4

In [55]:

```
y = df.drop(['sepal_length', 'petal_length'], axis = 1)
y.head()
```

Out[55]:

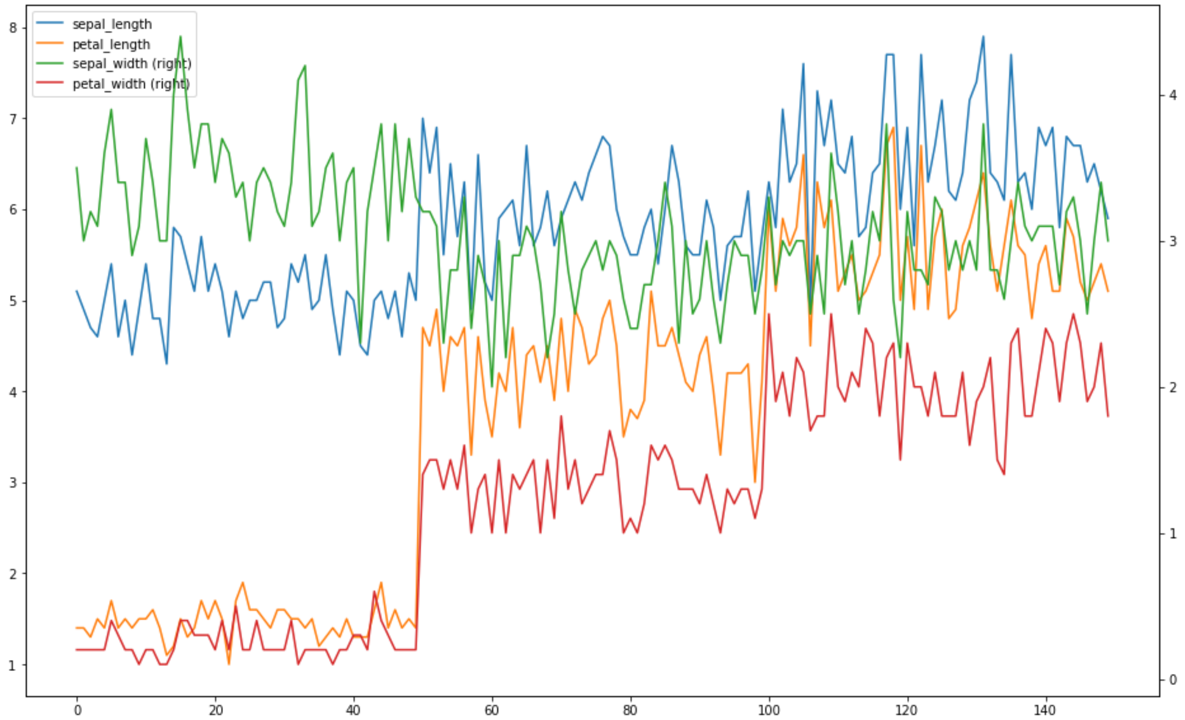
	sepal_width	petal_width
0	3.5	0.2
1	3.0	0.2
2	3.2	0.2
3	3.1	0.2
4	3.6	0.2

In [56]:

```
ax = x.plot()
y.plot(figsize = (16,10), secondary_y=True, ax = ax)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e6d7e80>
```

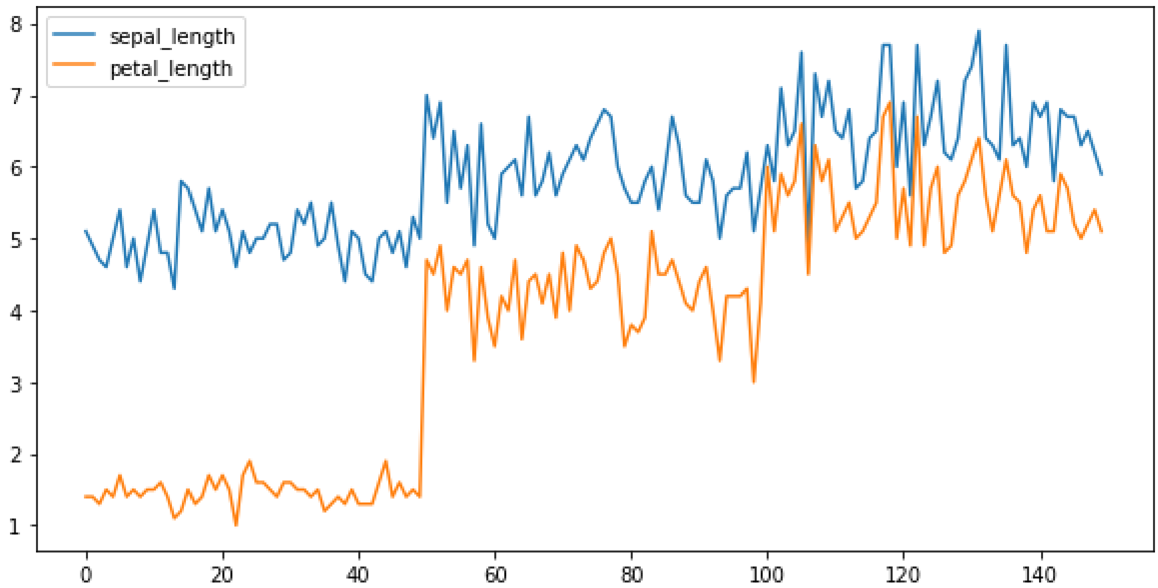


In [57]:

```
x.plot(figsize=(10,5), x_compat = True)
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e6c3a58>
```



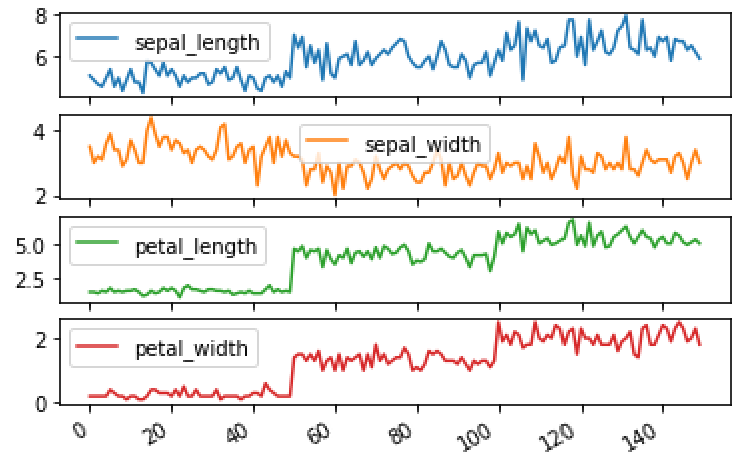
In [ ]:

In [58]:

```
df.plot(subplots = True)
```

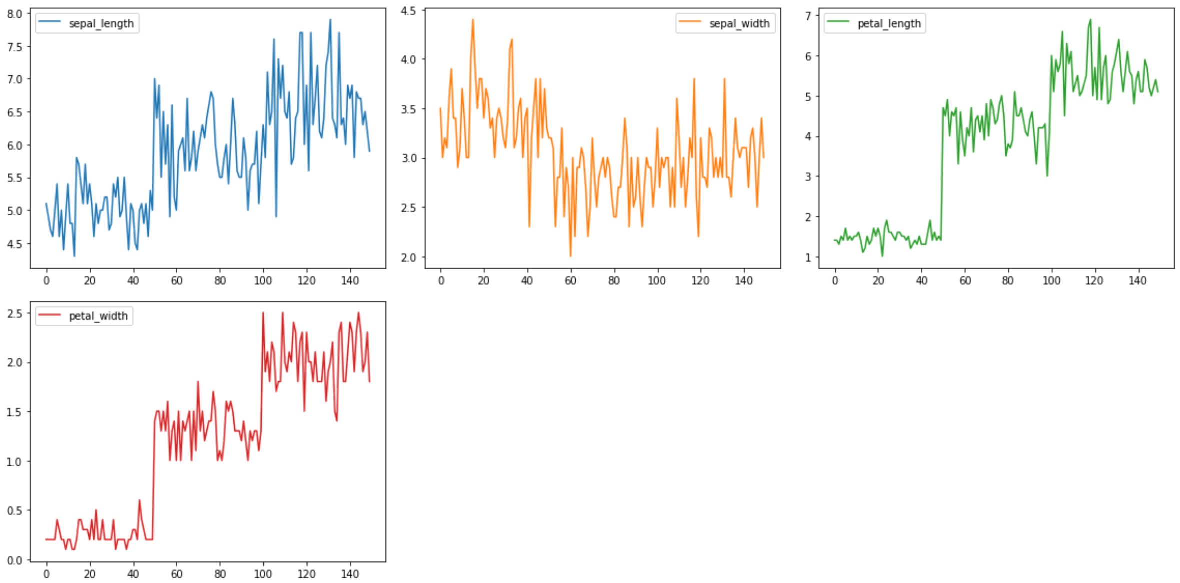
Out[58]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1a1ec1ea58>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a1ecbcef0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a1faab240>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a1fad26a0>],
      dtype=object)
```



In [59]:

```
df.plot(subplots = True, sharex = False, layout = (2,3), figsize = (16,8))
plt.tight_layout()
```



In [ ]:

In [78]:

```
from descartes import PolygonPatch
```

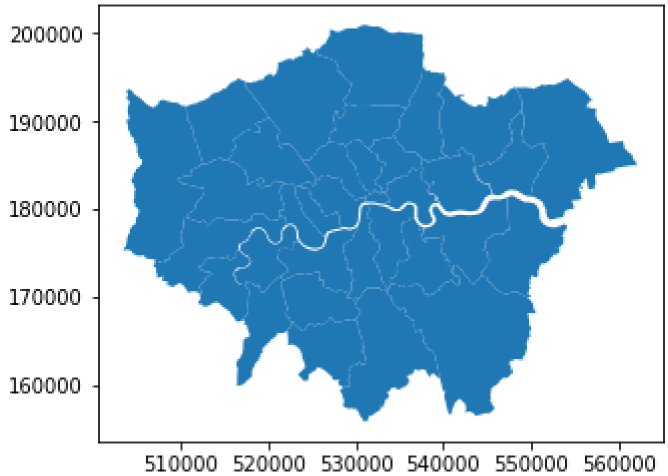
In [ ]:

In [79]:

```
map_df = gpd.read_file("London_Borough_Excluding_MHW.shp")
# check data type so we can see that this is not a normal dataframe, but a GEOdataframe
map_df.head()
map_df.plot()
```

Out[79]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e9f9630>
```



In [71]:

```
import geopandas as gpd
```

In [80]:

```
map_df
```

Out[80]:

	NAME	GSS_CODE	HECTARES	NONLD_AREA	ONS_INNER	SUB_2009	SUB_2006
0	Kingston upon Thames	E09000021	3726.117	0.000	F	None	None
1	Croydon	E09000008	8649.441	0.000	F	None	None
2	Bromley	E09000006	15013.487	0.000	F	None	None
3	Hounslow	E09000018	5658.541	60.755	F	None	None
4	Ealing	E09000009	5554.428	0.000	F	None	None
5	Havering	E09000016	11445.735	210.763	F	None	None
6	Hillingdon	E09000017	11570.063	0.000	F	None	None
7	Harrow	E09000015	5046.330	0.000	F	None	None
8	Brent	E09000005	4323.270	0.000	F	None	None
9	Barnet	E09000003	8674.837	0.000	F	None	None

	NAME	GSS_CODE	HECTARES	NONLD_AREA	ONS_INNER	SUB_2009	SUB_2006
10	Lambeth	E09000022	2724.940	43.927	T	None	None
11	Southwark	E09000028	2991.340	105.139	T	None	None
12	Lewisham	E09000023	3531.706	16.795	T	None	None
13	Greenwich	E09000011	5044.190	310.785	F	None	None
14	Bexley	E09000004	6428.649	370.619	F	None	None
15	Enfield	E09000010	8220.025	0.000	F	None	None
16	Waltham Forest	E09000031	3880.793	0.000	F	None	None
17	Redbridge	E09000026	5644.225	2.300	F	None	None
18	Sutton	E09000029	4384.698	0.000	F	None	None
19	Richmond upon Thames	E09000027	5876.111	135.443	F	None	None
20	Merton	E09000024	3762.466	0.000	F	None	None
21	Wandsworth	E09000032	3522.022	95.600	T	None	None

	NAME	GSS_CODE	HECTARES	NONLD_AREA	ONS_INNER	SUB_2009	SUB_2006
22	Hammersmith and Fulham	E09000013	1715.409	75.648	T	None	None
23	Kensington and Chelsea	E09000020	1238.379	25.994	T	None	None
24	Westminster	E09000033	2203.005	54.308	T	None	None
25	Camden	E09000007	2178.932	0.000	T	None	None
26	Tower Hamlets	E09000030	2157.501	179.707	T	None	None
27	Islington	E09000019	1485.664	0.000	T	None	None
28	Hackney	E09000012	1904.902	0.000	T	None	None
29	Haringey	E09000014	2959.837	0.000	T	None	None
30	Newham	E09000025	3857.806	237.637	T	None	None
31	Barking and Dagenham	E09000002	3779.934	169.150	F	None	None
32	City of London	E09000001	314.942	24.546	T	None	None

In [ ]:

In [ ]:

Numpy Operations on Array

By: Frason Francis - SE-IT:201903020

In [5]:

```
import numpy as np
type(np.array(["frason", 2, 3]))
```

Out[5]:

```
numpy.ndarray
```

In [2]:

```
#Upcasting:
```

In [3]:

```
np.array([1, 2, 3.0])
```

Out[3]:

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

In [4]:

```
#two dimensions
```

In [4]:

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

Out[4]:

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

In [6]:

```
#Minimum dimensions 2:
```

In [6]:

```
np.array([1, 2, 3], ndmin=5)
```

Out[6]:

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

In [8]:

```
#dtype
```

In [7]:

```
 np.array([1, 2, 3], dtype=complex)
```

Out[7]:

```
array([1.+0.j, 2.+0.j, 3.+0.j])
```

In [12]:

```
#Data-type consisting of more than one element:
```

In [15]:

```
x = np.array([(1,2),(3,4)],dtype=[('a','<i2'),('b','<i8')])
x
```

Out[15]:

```
array([(1, 2), (3, 4)], dtype=[('a', '<i2'), ('b', '<i8')])
```

In [39]:

```
type(x[1][0])
```

Out[39]:

```
numpy.int16
```

In [13]:

```

#Creating an array from sub-classes:
```

In [17]:

```
np.mat(np.array([[1, 2],[4,7]]))
```

Out[17]:

```
matrix([[1, 2],
        [4, 7]])
```

In [21]:

```
np.mat('1 2; 3 4')
```

Out[21]:

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

numpy.asarray

In [16]:

```
#Convert the input to an array.
```

In [18]:

```
#Convert a list into an array:
```

In [27]:

```
 a = [1, 2]
type(a)
```

Out[27]:

```
list
```

In [25]:

```
type(np.asarray(a))
```

Out[25]:

```
numpy.ndarray
```

In [31]:

```
a = np.array([1, 2]) #Existing arrays are not copied
type(a)
```

Out[31]:

```
numpy.ndarray
```

In [42]:

```
np.asarray((1,2))
```

Out[42]:

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

In [25]:

```
#If dtype is set, array is copied only if dtype does not match:
```

In [43]:

```
a = np.array([1, 2], dtype=np.float32)
a
```

Out[43]:

```
array([1., 2.], dtype=float32)
```

In [33]:

```
np.asarray([1,2]) is a
```

Out[33]:

```
False
```

In [36]:

```
np.asarray([1,2])
```

Out[36]:

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

In [34]:

```
np.asarray(a, dtype=np.float64) is a
```

Out[34]:

```
False
```

In [29]:

```
# ndarray subclasses are not passed through
```

In [44]:

```
issubclass(np.matrix, np.ndarray)
```

Out[44]:

```
True
```

In [48]:

```
a = np.matrix([[1, 2]])
np.asanyarray(a)
```

Out[48]:

```
matrix([[1, 2]])
```

In [49]:

```
np.asarray(a) is a
```

Out[49]:

```
False
```

In [50]:

```
np.asanyarray(a) is a
```

Out[50]:

```
True
```

In [51]:

```
type(np.asarray(a))
```

Out[51]:

```
numpy.ndarray
```

numpy.asanyarray

In [50]:

```
a = [1, 2]
np.asanyarray(a)
```

Out[50]:

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

In [86]:

```
a = np.matrix([1, 2])
a
```

Out[86]:

```
matrix([[1, 2]])
```

In [52]:

```
np.asanyarray([1,2]) is a
```

Out[52]:

```
True
```

```
#numpy.copy
```

In [52]:

```
np.array(a, copy=True)
```

Out[52]:

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

In [40]:

```
#Create an array x, with a reference y and a copy z:
```

In [56]:

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

Out[56]:

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

In [62]:

```
y = x
```

In [63]:

```
z = np.copy(x)
```

In [64]:

```
y[0] = 100
x
```

Out[64]:

```
array([100,   2,   3])
```

In [78]:

```
x
```

Out[78]:

```
array([100,   2,   3])
```

In [89]:

```
id(x[0])
```

Out[89]:

```
4767545360
```

In [91]:

```
id(y[0])
```

Out[91]:

```
4767545408
```

In [ ]:

```

```

In [92]:

```
id(x)
```

Out[92]:

```
4768156128
```

In [93]:

```
id(y)
```

Out[93]:

```
4768156128
```

numpy.fromfunction

In [58]:

```
#Construct an array by executing a function over each coordinate.
```

In [94]:

```
np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)
```

Out[94]:

```
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
```

In [119]:

```
np.fromfunction(lambda i, j: i * j, (3, 3), dtype=int)
```

Out[119]:

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

In [61]:

```
#Create a new 1-dimensional array from an iterable object.
```

In [97]:

```
iterable = (x*x for x in range(5))
iterable
```

Out[97]:

```
<generator object <genexpr> at 0x11c326b10>
```

In [98]:

```
np.fromiter(iterable, float)
```

Out[98]:

```
array([ 0.,  1.,  4.,  9., 16.])
```

In [64]:

```
#A new 1-D array initialized from text data in a string
```

In [101]:

```
a = np.fromstring('234 234',sep=' ')
a
```

Out[101]:

```
array([234., 234.])
```

In [102]:

```
np.fromstring('1, 2', dtype=int, sep=',')
```

Out[102]:

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

In [79]:

```
# How to create create a record array from a (flat) list of arrays
```

In [103]:

```
x1=np.array([1,2,3,4])
```

In [116]:

```
x2=np.array(['a','dd','xyz','12'])
```

In [117]:

```
x3=np.array([1.1,2,3,4])
x4=np.array([1.1,2,3,4])
type(x4)
```

Out[117]:

```
numpy.ndarray
```

In [118]:

```
r = np.core.records.fromarrays([x1,x2,x3,x4],names='a,b,c,d')
r
```

Out[118]:

```
rec.array([(1, 'a', 1.1, 1.1), (2, 'dd', 2. , 2. ), (3, 'xyz', 3. , 3. ),
           (4, '12', 4. , 4. )],
          dtype=[('a', '<i8'), ('b', '<U3'), ('c', '<f8'), ('d', '<f8')])
```

In [112]:

```
print(r[1]["a"])
```

```
2
```

```
x1[1]=34
```

In [130]:

```
x1[1]
```

Out[130]:

```
2
```

data types

In [120]:

```
my_list = [1,2,3]
import numpy as np
arr = np.array(my_list)
print("Type/Class of this object:",type(arr))
print("Here is the vector\n--------------------\n",arr)
```

```
Type/Class of this object: <class 'numpy.ndarray'>
Here is the vector
--------------------
 [1 2 3]
```

In [121]:

```
my_mat = [[1,2,3],[4,5,6],[7,8,9]]
mat = np.array(my_mat)
print("Type/Class of this object:",type(mat))
print("Here is the matrix\n----------\n",mat,"\n----------")
print("Dimension of this matrix: ",mat.ndim,sep='') #ndim gives the dimensison, 2 for a mat
print("Size of this matrix: ", mat.size,sep='') #size gives the total number of elements
print("Shape of this matrix: ", mat.shape,sep='') #shape gives the number of elements along
print("Data type of this matrix: ", mat.dtype,sep='') #dtype gives the data type contained
```

```
Type/Class of this object: <class 'numpy.ndarray'>
Here is the matrix
----------
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
----------
Dimension of this matrix: 2
Size of this matrix: 9
Shape of this matrix: (3, 3)
Data type of this matrix: int64
```

In [122]:

```
my_mat = [[1.1,2,3],[4,5.2,6],[7,8.3,9]]
mat = np.array(my_mat)
print("Data type of the modified matrix: ", mat.dtype,sep='') #dtype gives the data type co
print("\n\nEven tuples can be converted to ndarrays...")
```

```
Data type of the modified matrix: float64


Even tuples can be converted to ndarrays...
```

In [111]:

```
b = np.array([(1.5,2,3), (4,5,6)])
print("We write b = np.array([(1.5,2,3), (4,5,6)])")
print("Matrix made from tuples, not lists\n---------------------------------------")
print(b)
```

```
We write b = np.array([(1.5,2,3), (4,5,6)])
Matrix made from tuples, not lists
---------------------------------------
[[1.5 2.  3. ]
 [4.  5.  6. ]]
```

arange and linspace

In [127]:

```
print("A series of numbers:",type(np.arange(5,16)))
np.arange(5,16,2.3)# A series of numbers from low to high
```

```
A series of numbers: <class 'numpy.ndarray'>
```

Out[127]:

```
array([ 5. ,  7.3,  9.6, 11.9, 14.2])
```

In [128]:

```
list(range(5,16,2))
```

Out[128]:

```
[5, 7, 9, 11, 13, 15]
```

In [129]:

```
list(range(50,-1,5))
```

Out[129]:

```
[]
```

In [135]:

```
print("Numbers spaced apart by 2:",np.arange(50,-1,5)) # Numbers spaced apart by 2
```

```
Numbers spaced apart by 2: []
```

In [131]:

```
print("Numbers spaced apart by float:",np.arange(0,11,2.5)) # Numbers spaced apart by 2.5
```

```
Numbers spaced apart by float: [ 0.   2.5  5.   7.5 10. ]
```

In [132]:

```
print("Every 5th number from 50 in reverse order\n",np.arange(5.0,-1,-5))
```

```
Every 5th number from 50 in reverse order
 [5. 0.]
```

In [134]:

```
print("21 linearly spaced numbers between 1 and 5\n--------------------------------------
print((np.linspace(1,5,50)))
```

```
21 linearly spaced numbers between 1 and 5
--------------------------------------------
[1.         1.08163265 1.16326531 1.24489796 1.32653061 1.40816327
 1.48979592 1.57142857 1.65306122 1.73469388 1.81632653 1.89795918
 1.97959184 2.06122449 2.14285714 2.2244898  2.30612245 2.3877551
 2.46938776 2.55102041 2.63265306 2.71428571 2.79591837 2.87755102
 2.95918367 3.04081633 3.12244898 3.20408163 3.28571429 3.36734694
 3.44897959 3.53061224 3.6122449  3.69387755 3.7755102  3.85714286
 3.93877551 4.02040816 4.10204082 4.18367347 4.26530612 4.34693878
 4.42857143 4.51020408 4.59183673 4.67346939 4.75510204 4.83673469
 4.91836735 5.        ]
```

Matrix creation

In [136]:

```
print("Vector of zeroes\n---------------------")
print(np.zeros(5))
```

```
Vector of zeroes
---------------------
[0. 0. 0. 0. 0.]
```

In [139]:

```
print("Matrix of zeroes\n--------------------")
print(np.zeros((3,4))) # Notice Tuples
```

```
Matrix of zeroes
--------------------
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [127]:

```
print("Vector of ones\n---------------------")
print(np.ones(5))
```

```
Vector of ones
---------------------
[1. 1. 1. 1. 1.]
```

In [140]:

```
print("Matrix of ones\n---------------------")
print(np.ones((5,2,8))) # Note matrix dimension specified by Tuples
```

```
Matrix of ones
---------------------
[[[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. 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.]]]
```

In [147]:

```
print("Matrix of 5's\n---------------------")
print(5+np.ones((3,5)))
```

```
Matrix of 5's
---------------------
[[6. 6. 6. 6. 6.]
 [6. 6. 6. 6. 6.]
 [6. 6. 6. 6. 6.]]
```

In [148]:

```
print("Empty matrix\n-------------\n", np.empty((3,5)))
```

```
Empty matrix
-------------
 [[6. 6. 6. 6. 6.]
 [6. 6. 6. 6. 6.]
 [6. 6. 6. 6. 6.]]
```

In [149]:

```
mat1 = np.eye(4)
print("Identity matrix of dimension", mat1.shape)
print(mat1)
```

```
Identity matrix of dimension (4, 4)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [132]:

```
np.arange(3)
```

Out[132]:

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

In [80]:

```
np.arange(3.0)
```

Out[80]:

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

In [81]:

```
 np.arange(3,7)
```

Out[81]:

```
array([3, 4, 5, 6])
```

In [82]:

```
np.arange(3,7,2)
```

Out[82]:

```
array([3, 5])
```

In [150]:

```
np.linspace(2.0, 3.0, num=5)
```

Out[150]:

```
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
```

In [151]:

```
np.linspace(2.0, 3.0, num=5, endpoint=False)
```

Out[151]:

```
array([2. , 2.2, 2.4, 2.6, 2.8])
```

In [153]:

```
np.linspace(2.0, 3.0, num=9, retstep=True)
```

Out[153]:

```
(array([2.   , 2.125, 2.25 , 2.375, 2.5  , 2.625, 2.75 , 2.875, 3.   ]), 0.1
25)
```

In [147]:

```
#Return numbers spaced evenly on a log scale.
np.linspace(2.0, 3.0, num=4)
```

Out[147]:

```
array([2.        , 2.33333333, 2.66666667, 3.        ])
```

In [154]:

```
np.logspace(2.0, 3.0, num=4,base = 10)
```

Out[154]:

```
array([ 100.        ,  215.443469  ,  464.15888336, 1000.        ])
```

In [113]:

```
np.logspace(2.0, 3.0, num=4, endpoint=False)
```

Out[113]:

```
array([ 100.        ,  177.827941  ,  316.22776602,  562.34132519])
```

In [155]:

```
np.logspace(2.0, 3.0, num=4, base=2.0)
```

Out[155]:

```
array([4.        , 5.0396842 , 6.34960421, 8.        ])
```

In [115]:

```
#Extract a diagonal or construct a diagonal array.
```

In [179]:

```
x = np.arange(16).reshape((-1,4))
```

In [180]:

```
x
```

Out[180]:

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

In [172]:

```
np.diag(x)
```

Out[172]:

```
array([ 0,  5, 10, 15])
```

In [182]:

```
np.diag(x, k=2
       )
```

Out[182]:

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

In [183]:

```
np.diag(x, k=-1)
```

Out[183]:

```
array([ 4,  9, 14])
```

In [184]:

```
np.diag(np.diag(x))
```

Out[184]:

```
array([[ 0,  0,  0,  0],
       [ 0,  5,  0,  0],
       [ 0,  0, 10,  0],
       [ 0,  0,  0, 15]])
```

In [122]:

```
#Create a two-dimensional array with the flattened input as a diagonal.
```

In [180]:

```
np.diagflat([[1,2], [3,4]])
```

Out[180]:

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

In [124]:

```
np.diagflat([1,2], 1)
```

Out[124]:

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

In [125]:

```
#An array with ones at and below the given diagonal and zeros elsewhere.
```

In [187]:

```
np.tri(3, 5, 1, dtype=int)
```

Out[187]:

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

In [191]:

```
np.tri(3, 5,k=-1)
```

Out[191]:

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

In [128]:

```
#return a Lower triangle of an array.
```

In [205]:

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

Out[205]:

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

In [130]:

```
#return Upper triangle of an array.
```

In [206]:

```
np.triu([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], 1)
```

Out[206]:

```
array([[0, 2, 3],
       [0, 0, 6],
       [0, 0, 0],
       [0, 0, 0]])
```

Random number generation

In [197]:

```
print("Random number generation (from Uniform distribution)")
print(np.random.rand(2,3)) # 2 by 3 matrix with random numbers ranging from 0 to 1, Note no
```

```
Random number generation (from Uniform distribution)
[[0.79234365 0.26636652 0.84876143]
 [0.78766627 0.06627396 0.8829639 ]]
```

In [209]:

```
print("Numbers from Normal distribution with zero mean and standard deviation 1 i.e. standa
print(np.random.randn(4,3))
```

```
Numbers from Normal distribution with zero mean and standard deviation 1 i.
e. standard normal
[[ 0.08550888  0.73651756  0.08329424]
 [ 0.75179279  0.42441618  0.47752353]
 [ 0.23835886  1.06582108 -0.01790105]
 [ 0.48190152 -0.24940656 -0.44992115]]
```

In [202]:

```
print("Random integer vector:",np.random.randint(1,10)) #randint (low, high, # of samples t
print ("\nRandom integer matrix")
```

```
Random integer vector: 4

Random integer matrix
```

In [208]:

```
print(np.random.randint(1,100,(4,4))) #randint (low, high, # of samples to be drawn in a tu
print("\n20 samples drawn from a dice throw:",np.random.randint(1,7,20)) # 20 samples drawn
```

```
[[28 90 28 25]
 [14 51 32 98]
 [10 98 11 71]
 [12 42 54 77]]

20 samples drawn from a dice throw: [1 6 2 1 1 1 5 2 5 1 1 5 5 1 6 6 5 4 1
1]
```

Reshaping

In [211]:

```
from numpy.random import randint as ri
a = ri(1,100,30)
b = a.reshape(2,3,5)
c = a.reshape(6,-19878)
c
```

Out[211]:

```
array([[51, 63, 51, 54, 96],
       [11, 93,  2, 51, 34],
       [62, 35, 42, 95, 58],
       [88, 44, 36, 93, 83],
       [86, 56, 30,  2, 15],
       [68, 40, 77, 48, 57]])
```

In [209]:

```
print ("Shape of a:", a.shape)
print ("Shape of b:", b.shape)
print ("Shape of c:", c.shape)
```

```
Shape of a: (30,)
Shape of b: (2, 3, 5)
Shape of c: (6, 5)
```

In [219]:

```
print("\na looks like\n",'-'*20,"\n",a,"\n",'-'*20)
print("\nb looks like\n",'-'*20,"\n",b,"\n",'-'*20)
print("\nc looks like\n",'-'*20,"\n",c,"\n",'-'*20)
```

```
a looks like
 --------------------
 [72 69 61 67 42  2 92 42 38 56 22 71 21 81 81 47  2  1  6 94 52 14 87 71
 38 57 99 87 62 88]
 --------------------

b looks like
 --------------------
 [[[72 69 61 67 42]
  [ 2 92 42 38 56]
  [22 71 21 81 81]]

 [[47  2  1  6 94]
  [52 14 87 71 38]
  [57 99 87 62 88]]]
 --------------------

c looks like
 --------------------
 [[72 69 61 67 42]
 [ 2 92 42 38 56]
 [22 71 21 81 81]
 [47  2  1  6 94]
 [52 14 87 71 38]
 [57 99 87 62 88]]
 --------------------
```

In [212]:

```
A = ri(1,100,10) # Vector of random interegrs
print("\nVector of random integers\n",'-'*50,"\n",A)
print("\nHere is the sorted vector\n",'-'*50,"\n",np.sort(A))
```

```
Vector of random integers
 --------------------------------------------------
 [69 99 45 72 41 16 50 80 80 22]

Here is the sorted vector
 --------------------------------------------------
 [16 22 41 45 50 69 72 80 80 99]
```

In [221]:

```
M = ri(1,100,25).reshape(5,5) # Matrix of random interegrs
#print("\nHere is the sorted matrix along each row\n",'-'*50,"\n",np.sort(M, kind='mergesor
print("\nHere is the sorted matrix along each column\n",'-'*50,"\n",np.sort(M, axis=1, kind
M
```

```
Here is the sorted matrix along each column
 --------------------------------------------------
 [[18 25 38 73 94]
 [18 42 75 79 91]
 [21 44 62 78 85]
 [15 17 44 89 91]
 [ 3 24 38 62 86]]
```

Out[221]:

```
array([[18, 94, 25, 38, 73],
       [75, 42, 79, 18, 91],
       [21, 44, 85, 78, 62],
       [91, 15, 17, 44, 89],
       [62, 38,  3, 24, 86]])
```

In [214]:

```
print("Max of a:", M.max())
print("Max of b:", b.max())
b
```

```
Max of a: 99
Max of b: 96
```

Out[214]:

```
array([[[51, 63, 51, 54, 96],
        [11, 93,  2, 51, 34],
        [62, 35, 42, 95, 58]],

       [[88, 44, 36, 93, 83],
        [86, 56, 30,  2, 15],
        [68, 40, 77, 48, 57]]])
```

In [217]:

```
M
```

Out[217]:

```
array([[46, 88, 23, 66, 57],
       [50, 90, 30, 46, 36],
       [31, 19, 42, 23, 31],
       [ 4, 44, 86, 10, 81],
       [73, 35, 30, 99, 43]])
```

In [218]:

```
print("Max of a location:", M.argmax(axis= 1 ))
print("Max of b location:", b.argmax())
print("Max of c location:", b.argmax())
```

```
Max of a location: [1 1 2 2 3]
Max of b location: 4
Max of c location: 4
```

Indexing and slicing

In [229]:

```
arr = np.arange(0,11)
print("Array:",arr)
```

```
Array: [ 0  1  2  3  4  5  6  7  8  9 10]
```

In [161]:

```
print("Element at 7th index is:", arr[7])
```

```
Element at 7th index is: 7
```

In [230]:

```
print("Elements from 3rd to 5th index are:", arr[3:6:2])
```

```
Elements from 3rd to 5th index are: [3 5]
```

In [231]:

```
print("Elements up to 4th index are:", arr[:4])
arr
```

```
Elements up to 4th index are: [0 1 2 3]
```

Out[231]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [233]:

```
print("Elements from last backwards are:", arr[-1:7:-1])
```

```
Elements from last backwards are: [10  9  8]
```

In [164]:

```
print("3 Elements from last backwards are:", arr[-1:-6:2])
```

```
3 Elements from last backwards are: []
```

In [165]:

```
arr = np.arange(0,21,2)
print("New array:",arr)
```

```
New array: [ 0  2  4  6  8 10 12 14 16 18 20]
```

In [64]:

```
print("Elements at 2nd, 4th, and 9th index are:", arr[[2,4,9]]) # Pass a list as a index to
```

```
Elements at 2nd, 4th, and 9th index are: [ 4  8 18]
```

In [237]:

```
import numpy as np
mat = np.array(ri(10,100,15)).reshape(3,5)
print("Matrix of random 2-digit numbers\n--------------------------------\n",mat)
mat[1:4,3:5]
mat[0:3,[1,3]]
```

```
Matrix of random 2-digit numbers
--------------------------------
 [[35 75 44 59 85]
 [71 58 75 72 69]
 [76 97 49 63 53]]
```

Out[237]:

```
array([[75, 59],
       [58, 72],
       [97, 63]])
```

In [238]:

```
mat[0:3,[1,3]]
mat
```

Out[238]:

```
array([[35, 75, 44, 59, 85],
       [71, 58, 75, 72, 69],
       [76, 97, 49, 63, 53]])
```

In [241]:

```
print("\nDouble bracket indexing\n------------------------")
print("Element in row index 1 and column index 2:", mat[1][1])
mat
```

```
Double bracket indexing
------------------------
Element in row index 1 and column index 2: 58
```

Out[241]:

```
array([[35, 75, 44, 59, 85],
       [71, 58, 75, 72, 69],
       [76, 97, 49, 63, 53]])
```

In [242]:

```
print("\nSingle bracket with comma indexing\n----------------------------------")
print("Element in row index 1 and column index 2:", mat[1,2])
print("\nRow or column extract\n----------------------")
```

```
Single bracket with comma indexing
----------------------------------
Element in row index 1 and column index 2: 75

Row or column extract
----------------------
```

In [172]:

```
print("Entire row at index 2:", mat[2])
print("Entire column at index 3:", mat[:,3])
```

```
Entire row at index 2: [22 10 57 59 61]
Entire column at index 3: [92 88 59]
```

In [7]:

```
print("\nSubsetting sub-matrices\n--------------------------")
print("Matrix with row indices 1 and 2 and column indices 3 and 4\n", mat[1:3,3:5])
```

```
Subsetting sub-matrices
--------------------------
Matrix with row indices 1 and 2 and column indices 3 and 4
 [[14 32]
 [62 27]]
```

In [156]:

```
print("Matrix with row indices 0 and 1 and column indices 1 and 3\n", mat[0:2,[1,3]])
```

```
Matrix with row indices 0 and 1 and column indices 1 and 3
 [[77 69]
 [64 17]]
```

Subseting

In [173]:

```
mat = np.array(ri(10,100,15)).reshape(3,5)
print("Matrix of random 2-digit numbers\n--------------------------------\n",mat)
mat>50
```

```
Matrix of random 2-digit numbers
--------------------------------
 [[51 99 67 32 43]
 [53 61 80 97 51]
 [82 76 18 26 11]]
```

Out[173]:

```
array([[ True,  True,  True, False, False],
       [ True,  True,  True,  True,  True],
       [ True,  True, False, False, False]])
```

In [174]:

```
print ("Elements greater than 50\n", mat[mat>50])
```

```
Elements greater than 50
 [51 99 67 53 61 80 97 51 82 76]
```

Slicing

In [175]:

```
mat = np.array([[11,12,13],[21,22,23],[31,32,33]])
print("Original matrix")
print(mat)
```

```
Original matrix
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

In [176]:

```
mat_slice = mat[:2,:2]
print ("\nSliced matrix")
print(mat_slice)
print ("\nChange the sliced matrix")
```

```
Sliced matrix
[[11 12]
 [21 22]]

Change the sliced matrix
```

In [18]:

```
mat_slice[0,0] = 1000
print (mat_slice)
```

```
[[1000   12]
 [  21   22]]
```

In [75]:

```
print("\nBut the original matrix? WHOA! It got changed too!")
print(mat)
```

```
But the original matrix? WHOA! It got changed too!
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

In [177]:

```
# Little different way to create a copy of the slixed matrix
print ("\nDoing it again little differently now...\n")
mat = np.array([[11,12,13],[21,22,23],[31,32,33]])
print("Original matrix")
print(mat)
```

```
Doing it again little differently now...

Original matrix
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

In [21]:

```
mat_slice = np.array(mat[:2,:2]) # Notice the np.array command to create a new array not ju
print ("\nSliced matrix")
print(mat_slice)
```

```
Sliced matrix
[[11 12]
 [21 22]]
```

In [178]:

```
print ("\nChange the sliced matrix")
mat_slice[0,0] = 1000
print (mat_slice)
```

```
Change the sliced matrix
[[1000   12]
 [  21   22]]
```

In [22]:

```
print("\nBut the original matrix? NO CHANGE this time:)")
print(mat)
```

```
But the original matrix? NO CHANGE this time:)
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

Universal Functions

In [247]:

```
mat1 = np.array(ri(1,10,9)).reshape(3,3)
mat2 = np.array(ri(1,10,9)).reshape(3,3)
print("\n1st Matrix of random single-digit numbers\n---------------------------------------
print("\n2nd Matrix of random single-digit numbers\n---------------------------------------
```

```
1st Matrix of random single-digit numbers
----------------------------------------
 [[3 3 9]
 [6 1 8]
 [5 9 1]]

2nd Matrix of random single-digit numbers
----------------------------------------
 [[3 5 7]
 [4 4 9]
 [1 2 5]]
```

In [248]:

```
mat1*mat2
```

Out[248]:

```
array([[ 9, 15, 63],
       [24,  4, 72],
       [ 5, 18,  5]])
```

In [ ]:

```

```

In [249]:

```
#print("\nAddition\n------------------\n", mat1+mat2)
print("\nMultiplication\n------------------\n", mat1@mat2)
```

```
Multiplication
------------------
 [[ 30  45  93]
 [ 30  50  91]
 [ 52  63 121]]
```

In [251]:

```
print("\nDivision\n------------------\n", mat1/0)
#print("\nLineaer combination: 3*A - 2*B\n-----------------------------\n", 3*mat1-2*mat2)
```

```
Division
------------------
 [[inf inf inf]
 [inf inf inf]
 [inf inf inf]]

/Users/sudhanshukumar/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:1: RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

In [252]:

```
print("\nAddition of a scalar (100)\n-------------------------\n", 100+mat1)
```

```
Addition of a scalar (100)
-------------------------
 [[103 103 109]
 [106 101 108]
 [105 109 101]]
```

In [253]:

```
print("\nExponentiation, matrix cubed here\n----------------------------------------\n", ma
print("\nExponentiation, sq-root using pow function\n--------------------------------------
```

```
Exponentiation, matrix cubed here
----------------------------------------
 [[ 27  27 729]
 [216   1 512]
 [125 729   1]]

Exponentiation, sq-root using pow function
-------------------------------------------
 [[ 27  27 729]
 [216   1 512]
 [125 729   1]]
```

Broadcasting

In [174]:

```
#NumPy operations are usually done on pairs of arrays on an element-by-element basis.
#In the simplest case, the two arrays must have exactly the same shape.
#NumPy’s broadcasting rule relaxes this constraint when the arrays’ shapes meet certain con
#When operating on two arrays, NumPy compares their shapes element-wise. It starts with the
#dimensions, and works its way forward. Two dimensions are compatible when
#they are equal, or one of them is 1
```

In [254]:

```
start = np.zeros((4,4))
start= start+100
start
```

Out[254]:

```
array([[100., 100., 100., 100.],
       [100., 100., 100., 100.],
       [100., 100., 100., 100.],
       [100., 100., 100., 100.]])
```

In [256]:

```
# create a rank 1 ndarray with 3 values
add_rows = np.array([1, 0, 2,5])
print(add_rows)
```

```
[1 0 2 5]
```

In [257]:

```
y = start + add_rows  # add to each row of 'start' using broadcasting
print(y)
```

```
[[101. 100. 102. 105.]
 [101. 100. 102. 105.]
 [101. 100. 102. 105.]
 [101. 100. 102. 105.]]
```

In [262]:

```
# create an ndarray which is 4 x 1 to broadcast across columns
add_cols = np.array([[0,1,2,3]])
add_cols = add_cols.T
print(add_cols)
```

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

In [263]:

```
# add to each column of 'start' using broadcasting
y = start + add_cols
print(y)
```

```
[[100. 100. 100. 100.]
 [101. 101. 101. 101.]
 [102. 102. 102. 102.]
 [103. 103. 103. 103.]]
```

In [264]:

```
# this will just broadcast in both dimensions
add_scalar = np.array([100])
print(start+y)
```

```
[[200. 200. 200. 200.]
 [201. 201. 201. 201.]
 [202. 202. 202. 202.]
 [203. 203. 203. 203.]]
```

Array Math

In [265]:

```
mat1 = np.array(ri(1,10,9)).reshape(3,3)
mat2 = np.array(ri(1,10,9)).reshape(3,3)
print("\n1st Matrix of random single-digit numbers\n\n",mat1)
print("\n2nd Matrix of random single-digit numbers\n------------\n",mat2)
```

```

1st Matrix of random single-digit numbers

 [[9 5 4]
 [1 7 3]
 [8 3 5]]

2nd Matrix of random single-digit numbers
------------
 [[4 8 6]
 [7 6 8]
 [4 6 5]]
```

In [266]:

```
print("\nSq-root of 1st matrix using np\n------------------\n", np.sqrt(mat1))
```

```
Sq-root of 1st matrix using np
------------------
 [[3.         2.23606798 2.        ]
 [1.         2.64575131 1.73205081]
 [2.82842712 1.73205081 2.23606798]]
```

In [267]:

```
print("\nExponential power of 1st matrix using np\n",'-'*50,"\n", np.exp(mat1))
```

```
Exponential power of 1st matrix using np
 --------------------------------------------------
 [[8.10308393e+03 1.48413159e+02 5.45981500e+01]
 [2.71828183e+00 1.09663316e+03 2.00855369e+01]
 [2.98095799e+03 2.00855369e+01 1.48413159e+02]]
```

In [268]:

```
print("\n10-base logarithm on 1st matrix using np\n",'-'*50,"\n", np.log10(mat1))
print(mat1)
print(mat2)
```

```
10-base logarithm on 1st matrix using np
 --------------------------------------------------
 [[0.95424251 0.69897    0.60205999]
 [0.         0.84509804 0.47712125]
 [0.90308999 0.47712125 0.69897   ]]
[[9 5 4]
 [1 7 3]
 [8 3 5]]
[[4 8 6]
 [7 6 8]
 [4 6 5]]
```

In [270]:

```
print("\nModulo reminder using np\n",'-'*50,"\n", np.fmod(mat1,mat2))
mat1%mat2
```

```

Modulo reminder using np
 --------------------------------------------------
 [[1 5 4]
 [1 1 3]
 [0 3 0]]
```

Out[270]:

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

In [2]:

```
print("\nCombination of functions by showing exponetial decay of a sine wave\n",'-'*70)
```

```

Combination of functions by showing exponetial decay of a sine wave
 ----------------------------------------------------------------------
```

In [3]:

```
A = np.linspace(0,12*np.pi,1001)
```

In [4]:

```
A
```

Out[4]:

```
array([ 0.        ,  0.03769911,  0.07539822, ..., 37.62371362,
       37.66141273, 37.69911184])
```