Numpy

is a linear algebra library for python, almost all the libraries in PyData Ecosystem rely on Numpy as one of their main building blocks and it is fast.

Numpy arrays essentially come in two flavors: vectors (1-d arrays) and matrices (2-d arrays with one row or one column)

How to create an array from a list

In [1]:
```
name = ['David', 'Adaugo', 'Deziri', 'Chukwunonso']
print(name)
```

```
['David', 'Adaugo', 'Deziri', 'Chukwunonso']
```

In [2]:
```
import numpy as np
name_array = np.array(name)
print(name_array)
```

```
['David' 'Adaugo' 'Deziri' 'Chukwunonso']
```

In [5]:
```
name_array
```

Out[5]:
```
array(['David', 'Adaugo', 'Deziri', 'Chukwunonso'], dtype='<U11')
```

In [8]:
```
full_name = [['David', 'Deziri'], ['David', 'Precious'], ['David', 'Chukwur
print(full_name)
```

```
[['David', 'Deziri'], ['David', 'Precious'], ['David', 'Chukwunonso']]
```

In [7]:
```
full_name_array = np.array(full_name)
print(full_name_array)
```

```
[['David' 'Deziri']
 ['David' 'Precious']
 ['David' 'Chukwunonso']]
```

In [9]:
```
full_name_array
```

Out[9]:
```
array([['David', 'Deziri'],
       ['David', 'Precious'],
       ['David', 'Chukwunonso']], dtype='<U11')
```

In [15]:
```
type(full_name_array)
```

Out[15]:
```
numpy.ndarray
```

Other ways of creating an array

In [17]:
```
np.arange(0,10)
```

Out[17]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:
```
np.arange(0,10, 2)
```

Out[18]:
```
array([0, 2, 4, 6, 8])
```

In [19]:
```
np.zeros(3)
```

Out[19]:
```
array([0., 0., 0.])
```

In [20]:
```
np.zeros((2,3))
```

Out[20]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [21]:
```
np.ones(4)
```

Out[21]:
```
array([1., 1., 1., 1.])
```

In [22]:
```
np.ones((3,3))
```

Out[22]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [24]:
```
np.ones((3,3))*3
```

Out[24]:
```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

In [25]:
```
#how to return 10 evenly spaced number from 0 to 5:
np.linspace(0,5,10)
```

Out[25]:
```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

In [26]:
```
#identity matrix with np:
np.eye(4)
```

Out[26]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [27]:
```
np.eye(3)
```

Out[27]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [28]:
```
#creating arrays of random numbers:
np.random.rand(5)
```

Out[28]:
```
array([0.46963854, 0.13946154, 0.64787587, 0.62420481, 0.95238374])
```

In [29]:
```
#multidi:
np.random.rand(3,3)
```

Out[29]:
```
array([[0.58734967, 0.67576544, 0.89692854],
       [0.24464632, 0.98284716, 0.69332408],
       [0.88627224, 0.70289769, 0.13432843]])
```

In [30]:
```
np.random.randn(2)
```

Out[30]:
```
array([-0.47465658,  0.19347097])
```

In [40]:
```
np.random.randint(1,10, 5)
```

Out[40]:
```
array([1, 7, 6, 7, 5])
```

In [38]:
```
np.random.randint(1,10, (2,2))
```

Out[38]:
```
array([[4, 6],
       [5, 8]])
```

In [39]:
```
np.random.randint(1,10, (3,3))
```

Out[39]:
```
array([[8, 8, 8],
       [3, 2, 8],
       [3, 5, 3]])
```

Reshape

In [43]:
```
array_shape = np.random.randint(1,10, 6)
array_shape
```

Out[43]:
```
array([2, 9, 4, 6, 5, 6])
```

In [44]:
```
reshaped = array_shape.reshape(3,2)
reshaped
```

Out[44]:
```
array([[2, 9],
       [4, 6],
       [5, 6]])
```

Other methods

In [45]:
```
reshaped.max()
```

Out[45]:
```
9
```

In [46]:
```
reshaped.min()
```

Out[46]:
```
2
```

In [47]:
```
#index location of the max
reshaped.argmax()
```

Out[47]:
```
1
```

In [48]:
```
#index location of the min
reshaped.argmin()
```

Out[48]:
```
0
```

In [49]:
```
#the shape of an array:
reshaped.shape
```

Out[49]:
```
(3, 2)
```

In [50]:
```
array_shape.shape
```

Out[50]:
```
(6,)
```

In [51]:
```
#data type in an array:
reshaped.dtype
```

Out[51]:
```
dtype('int64')
```

Numpy Indexing and Selection

In [55]:
```
age = np.random.randint(1,50, 10)
age
```

Out[55]:
```
array([ 2, 15, 37,  9, 36, 27, 15, 46, 14, 36])
```

In [56]:
```
#to return the value at index 7
age[7]
```

Out[56]:
```
46
```

In [57]:
```
#value in a range
age[2:7]
```

Out[57]:
```
array([37,  9, 36, 27, 15])
```

In [58]:
```
age[2:]
```

Out[58]:
```
array([37,  9, 36, 27, 15, 46, 14, 36])
```

In [59]:
```
age[:2]
```

Out[59]:
```
array([ 2, 15])
```

In [60]:
```
age[:]
```

Out[60]:
```
array([ 2, 15, 37,  9, 36, 27, 15, 46, 14, 36])
```

learn the differences between copy and = in an array

Selection in 2d arrays

In [63]:
```
array_2d = np.random.randint(1,100, (3,3))
array_2d
```

Out[63]:
```
array([[12, 12, 17],
       [18, 73, 99],
       [15, 61, 52]])
```

In [66]:
```
#to select the first item on the first row:
array_2d[0,0]
```

Out[66]:
```
12
```

In [67]:
```
#to select the second item on the second row:
array_2d[1,1]
```

Out[67]:
```
73
```

In [77]:
```
#to select everything from second column and first row:
array_2d[0:,1:]
```

Out[77]:
```
array([[12, 17],
       [73, 99],
       [61, 52]])
```

In [78]:
```
#to select everything from second column and second row:
array_2d[1:,1:]
```

Out[78]:
```
array([[73, 99],
       [61, 52]])
```

In [80]:
```
array_2d[:2,1:]
```

Out[80]:
```
array([[12, 17],
       [73, 99]])
```

Boolean Array

In [81]:
```
age
```

Out[81]:
```
array([ 2, 15, 37,  9, 36, 27, 15, 46, 14, 36])
```

In [87]:
```
adult_bool = age >= 18
adult_bool
```

Out[87]:
```
array([False, False,  True, False,  True,  True, False,  True, False,
        True])
```

In [88]:
```
#returns the values where age >= 18
adult = age[adult_bool]
adult
```

Out[88]:
```
array([37, 36, 27, 46, 36])
```

In [89]:
```
#or
age[age>=18]
```

Out[89]:
```
array([37, 36, 27, 46, 36])
```

In [90]:
```
age[age<18]
```

Out[90]:
```
array([ 2, 15,  9, 15, 14])
```

#

In [91]:
```
array_2d = np.random.randint(1,100, (5,10))
array_2d
```

Out[91]:
```
array([[20, 67, 62, 68, 20, 56,  3, 79, 48, 47],
       [49, 19, 54, 11, 20, 29, 93, 27, 54, 16],
       [74, 45, 18, 88, 76, 92, 72, 21, 31, 76],
       [24, 26, 65, 47, 50, 37, 45, 14, 58, 70],
       [62, 24, 50, 27, 90, 47, 68, 12, 23, 44]])
```

In [92]:
```
array_2d[array_2d>50]
```

Out[92]:
```
array([67, 62, 68, 56, 79, 54, 93, 54, 74, 88, 76, 92, 72, 76, 65, 58, 70,
       62, 90, 68])
```

In [94]:
```
array_2d[1:4,3:7]
```

Out[94]:
```
array([[11, 20, 29, 93],
       [88, 76, 92, 72],
       [47, 50, 37, 45]])
```

In [95]:
```
array_2d[1:,]
```

Out[95]:
```
array([[49, 19, 54, 11, 20, 29, 93, 27, 54, 16],
       [74, 45, 18, 88, 76, 92, 72, 21, 31, 76],
       [24, 26, 65, 47, 50, 37, 45, 14, 58, 70],
       [62, 24, 50, 27, 90, 47, 68, 12, 23, 44]])
```

In [97]:
```
array_2d[:,1:]
```

Out[97]:
```
array([[67, 62, 68, 20, 56,  3, 79, 48, 47],
       [19, 54, 11, 20, 29, 93, 27, 54, 16],
       [45, 18, 88, 76, 92, 72, 21, 31, 76],
       [26, 65, 47, 50, 37, 45, 14, 58, 70],
       [24, 50, 27, 90, 47, 68, 12, 23, 44]])
```

Numpy Operations

Array with array

Array with Scalars

Universal Array Functions

In [98]:
```
age
```

Out[98]:
```
array([ 2, 15, 37,  9, 36, 27, 15, 46, 14, 36])
```

In [108…
```
#linear addition of arrays
new = np.arange(1,11)
new
```

Out[108…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [109…
```
#addition
age + new
```
Out[109…
```
array([ 3, 17, 40, 13, 41, 33, 22, 54, 23, 46])
```
In [110…
```
#subtraction
age - new
```
Out[110…
```
array([ 1, 13, 34,  5, 31, 21,  8, 38,  5, 26])
```
In [111…
```
#multiplication
age * new
```
Out[111…
```
array([  2,  30, 111,  36, 180, 162, 105, 368, 126, 360])
```
In [112…
```
#
age % new
```
Out[112…
```
array([0, 1, 1, 1, 1, 3, 1, 6, 5, 6])
```

Scalars

In [113…
```
#operations with a single value
age * 10
```
Out[113…
```
array([ 20, 150, 370,  90, 360, 270, 150, 460, 140, 360])
```
In [114…
```
age % 2
```
Out[114…
```
array([0, 1, 1, 1, 0, 1, 1, 0, 0, 0])
```
In [115…
```
old_age = age - 2
old_age
```
Out[115…
```
array([ 0, 13, 35,  7, 34, 25, 13, 44, 12, 34])
```
In [116…
```
age ** 2
```
Out[116…
```
array([   4,  225, 1369,   81, 1296,  729,  225, 2116,  196, 1296])
```

Universal Array Functions

In [117…
```
#square root
np.sqrt(age)
```

Out[117…
```
array([1.41421356, 3.87298335, 6.08276253, 3.        , 6.        ,
       5.19615242, 3.87298335, 6.78232998, 3.74165739, 6.        ])
```

In [118…
```
#exponential
np.exp(age)
```

Out[118…
```
array([7.38905610e+00, 3.26901737e+06, 1.17191424e+16, 8.10308393e+03,
       4.31123155e+15, 5.32048241e+11, 3.26901737e+06, 9.49611942e+19,
       1.20260428e+06, 4.31123155e+15])
```

In [119…
```
#maximum value
np.max(age)
```

Out[119…
```
46
```

In [120…
```
# or maximum value
age.max()
```

Out[120…
```
46
```

In [121…
```
#minimum value
np.min(age)
```

Out[121…
```
2
```

In [122…
```
#sin
np.sin(age)
```

Out[122…
```
array([ 0.90929743,  0.65028784, -0.64353813,  0.41211849, -0.99177885,
        0.95637593,  0.65028784,  0.90178835,  0.99060736, -0.99177885])
```

In [124…
```
#cosin
np.cos(age)
```

Out[124…
```
array([-0.41614684, -0.75968791,  0.76541405, -0.91113026, -0.12796369,
       -0.29213881, -0.75968791, -0.43217794,  0.13673722, -0.12796369])
```

In [125…
```
#Tan
np.tan(age)
```

Out[125…
```
array([-2.18503986, -0.8559934 , -0.84077126, -0.45231566,  7.75047091,
       -3.2737038 , -0.8559934 , -2.08661353,  7.24460662,  7.75047091])
```

In [126…
```
#log
np.log(age)
```

Out[126…
```
array([0.69314718, 2.7080502 , 3.61091791, 2.19722458, 3.58351894,
       3.29583687, 2.7080502 , 3.8286414 , 2.63905733, 3.58351894])
```

In [127…
```
np.arange(1,101).reshape(10,10)/100
```

Out[127…
```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

In [128…
```
np.linspace(0.01,1,100).reshape(10,10)
```

Out[128…
```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

Pandas

is an open source library built on top pf Numpy

it allows for fast analysis and data cleaning and preparation

Series, DataFrames, Missing Data, GroupBy, Merging, Joining, Concatenating, Operations, Data Input and Output

In [129…
```
import pandas as pd
```

In [130…
```
name = ['Deziri', 'Finest', 'David']
age = [1, 28, 30]
age_array = np.array(age)
d_data = {'Deziri':1, 'Finest':28, 'David':30}
```

In [131…
```
# create a series
pd.Series(data=age)
```

Out[131…
```
0     1
1    28
2    30
dtype: int64
```

In [132…
```
#using two lists
pd.Series(data=age, index=name)
```

Out[132…
```
Deziri     1
Finest    28
David     30
dtype: int64
```

In [133…
```
pd.Series(age, name)
```

Out[133…
```
Deziri     1
Finest    28
David     30
dtype: int64
```

In [134…
```
#using a numpy array
pd.Series(data=age_array)
```

Out[134…
```
0     1
1    28
2    30
dtype: int64
```

In [135…
```
#using a dictionary
pd.Series(data=d_data)
```

Out[135…
```
Deziri     1
Finest    28
David     30
dtype: int64
```

In [136…
```
family = pd.Series([10,20,30,40], ['Chibuike', 'Somkene', 'Chioma', 'Chinel
family
```

Out[136…
```
Chibuike    10
Somkene     20
Chioma      30
Chinelo     40
dtype: int64
```

In [137…
```
family['Chibuike']
```

Out[137…
```
10
```

In [138…
```
family2 = pd.Series([10,20,30,40], ['Chioma', 'Chinelo', 'Nonso', 'Sochima'
family2
```

Out[138…
```
Chioma     10
Chinelo    20
Nonso      30
Sochima    40
dtype: int64
```

In [139…
```
family + family2
```

Out[139…
```
Chibuike     NaN
Chinelo     60.0
Chioma      40.0
Nonso        NaN
Sochima      NaN
Somkene      NaN
dtype: float64
```

DataFrame

In [140…
```
array_2d = np.random.randint(1,100, (5,4))
```

In [142…
```
df=pd.DataFrame(array_2d, ['A', 'B', 'C', 'D', 'E'], ['W', 'X', 'Y', 'Z'])
```

In [143…
```
df #this is a datafram
```

Out[143…

	W	X	Y	Z
A	58	65	69	55
B	43	4	55	12
C	18	51	5	21
D	99	36	73	3
E	14	20	68	46

In [144…
```
df['W'] #this is one of the series in the dataframe
```

Out[144…
```
A    58
B    43
C    18
D    99
E    14
Name: W, dtype: int64
```

In [146…
```
# dataframe are series that share the same index
```

In [147…
```
type(df['W'])
```

Out[147…
```
pandas.core.series.Series
```

In [148…
```
type(df)
```

Out[148…
```
pandas.core.frame.DataFrame
```

In [150…
```
df[['W', 'Y']]
```

Out[150…

	W	Y
A	58	69
B	43	55
C	18	5
D	99	73
E	14	68

In [151…
```
df['ADD'] = df['W'] + df['X']
```

In [152…
```
df
```

Out[152…

	W	X	Y	Z	ADD
A	58	65	69	55	123
B	43	4	55	12	47
C	18	51	5	21	69
D	99	36	73	3	135
E	14	20	68	46	34

In [153…
```
#creating a new series
df['New'] = [5, 3, 7, 8, 2]
```

In [154…
```
df
```

Out[154…

	W	X	Y	Z	ADD	New
A	58	65	69	55	123	5
B	43	4	55	12	47	3
C	18	51	5	21	69	7
D	99	36	73	3	135	8
E	14	20	68	46	34	2

In [155…
```
#deleting series
df.drop('ADD', axis=1)
```

Out[155…

	W	X	Y	Z	New
A	58	65	69	55	5
B	43	4	55	12	3
C	18	51	5	21	7
D	99	36	73	3	8
E	14	20	68	46	2

In [156…

```
# df.drop('ADD', axis=1) does not delete it
df
```

Out[156…

	W	X	Y	Z	ADD	New
A	58	65	69	55	123	5
B	43	4	55	12	47	3
C	18	51	5	21	69	7
D	99	36	73	3	135	8
E	14	20	68	46	34	2

In [157…

```
#to delete:
df.drop('ADD', axis=1, inplace=True)
```

In [158…

```
df
```

Out[158…

	W	X	Y	Z	New
A	58	65	69	55	5
B	43	4	55	12	3
C	18	51	5	21	7
D	99	36	73	3	8
E	14	20	68	46	2

In [159…

```
#you can drop row
df.drop('E')
```

Out[159…

	W	X	Y	Z	New
A	58	65	69	55	5
B	43	4	55	12	3
C	18	51	5	21	7
D	99	36	73	3	8

selecting Rows

In [160…
```
df.loc['A']
```

Out[160…
```
W      58
X      65
Y      69
Z      55
New     5
Name: A, dtype: int64
```

In [161…
```
#using the index:
df.iloc[0]
```

Out[161…
```
W      58
X      65
Y      69
Z      55
New     5
Name: A, dtype: int64
```

In [162…
```
df.loc['A', 'Z']
```

Out[162…
```
55
```

In [168…
```
df.loc[['A', 'B'],['X', 'Y']]
```

Out[168…

	X	Y
A	65	69
B	4	55

In [169…
```
df.loc[['A', 'B']]
```

Out[169…

	W	X	Y	Z	New
A	58	65	69	55	5
B	43	4	55	12	3

In [178…
```
df
```

Out[178…

	W	X	Y	Z
A	58	65	69	55
B	43	4	55	12
C	18	51	5	21
D	99	36	73	3
E	14	20	68	46

In [179…

```
df<50
```

Out[179…

	W	X	Y	Z
A	False	False	False	False
B	True	True	False	True
C	True	False	True	True
D	False	True	False	True
E	True	True	False	True

In [180…

```
booldf = df<50
df[booldf]
```

Out[180…

	W	X	Y	Z
A	NaN	NaN	NaN	NaN
B	43.0	4.0	NaN	12.0
C	18.0	NaN	5.0	21.0
D	NaN	36.0	NaN	3.0
E	14.0	20.0	NaN	46.0

In [181…

```
df[df<50]
```

Out[181…

	W	X	Y	Z
A	NaN	NaN	NaN	NaN
B	43.0	4.0	NaN	12.0
C	18.0	NaN	5.0	21.0
D	NaN	36.0	NaN	3.0
E	14.0	20.0	NaN	46.0

In [182…

```
df['W']<50
```

Out[182…
```
A    False
B     True
C     True
D    False
E     True
Name: W, dtype: bool
```

In [183…
```
df[df['W']<50] #get only the rows where W is greater than 50
```

Out[183…

	W	X	Y	Z
B	43	4	55	12
C	18	51	5	21
E	14	20	68	46

In [184…
```
df[df['Z']<50]
```

Out[184…

	W	X	Y	Z
B	43	4	55	12
C	18	51	5	21
D	99	36	73	3
E	14	20	68	46

In [186…
```
df[df['Y']<50]
```

Out[186…

	W	X	Y	Z
C	18	51	5	21

In [189…
```
df[df['Y']<50][['X', 'Y']]
```

Out[189…

	X	Y
C	51	5

In [190…
```
# Pandas doesn't use 'AND' but '&'
#eg:
df[(df['W']>50) & (df['Y']>50)]
```

Out[190…

	W	X	Y	Z
A	58	65	69	55
D	99	36	73	3

In [192…
```
df[(df['W']>50) | (df['Y']>50)]
```

Out[192…

	W	X	Y	Z
A	58	65	69	55
B	43	4	55	12
D	99	36	73	3
E	14	20	68	46

In [193…

```
df
```

Out[193…

	W	X	Y	Z
A	58	65	69	55
B	43	4	55	12
C	18	51	5	21
D	99	36	73	3
E	14	20	68	46

In [194…

```
#to reset the index
df.reset_index()
```

Out[194…

	index	W	X	Y	Z
0	A	58	65	69	55
1	B	43	4	55	12
2	C	18	51	5	21
3	D	99	36	73	3
4	E	14	20	68	46

In [196…

```
df.reset_index(drop='index', inplace=True)
```

In [197…

```
df
```

Out[197…

	W	X	Y	Z
0	58	65	69	55
1	43	4	55	12
2	18	51	5	21
3	99	36	73	3
4	14	20	68	46

In [198…
```
newind = 'NG GH SA KY CA'.split() #to create a list
newind
```

Out[198…
```
['NG', 'GH', 'SA', 'KY', 'CA']
```

In [199…
```
df['Afr'] = newind
df
```

Out[199…

	W	X	Y	Z	Afr
0	58	65	69	55	NG
1	43	4	55	12	GH
2	18	51	5	21	SA
3	99	36	73	3	KY
4	14	20	68	46	CA

In [200…
```
#to make a column the index:
df.set_index('Afr')
```

Out[200…

	W	X	Y	Z
Afr				
NG	58	65	69	55
GH	43	4	55	12
SA	18	51	5	21
KY	99	36	73	3
CA	14	20	68	46

In [201…
```
df
```

Out[201…

	W	X	Y	Z	Afr
0	58	65	69	55	NG
1	43	4	55	12	GH
2	18	51	5	21	SA
3	99	36	73	3	KY
4	14	20	68	46	CA

Multi Index Data Frame

In [203…
```
outside = ['G1', 'G1', 'G1', 'G2', 'G2', 'G2' ]
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside, inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

In [205…
```
df = pd.DataFrame(np.random.randint(1,100, (6,2)), hier_index, ['A', 'B'])
```

In [206…
```
df
```

Out[206…

		A	B
G1	**1**	6	50
	2	68	52
	3	90	37
G2	**1**	48	45
	2	64	44
	3	58	59

In [207…
```
df.loc['G1']
```

Out[207…

	A	B
1	6	50
2	68	52
3	90	37

In [208…
```
df.loc['G2']
```

Out[208…

	A	B
1	48	45
2	64	44
3	58	59

In [209…
```
df.loc['G1'].loc[1]
```

Out[209…
```
A     6
B    50
Name: 1, dtype: int64
```

In [210…
```
df.loc['G1'].loc[1].loc['A']
```

Out[210…
```
6
```

In [211…
```
df.index.names
```

Out[211…
```
FrozenList([None, None])
```

In [212…
```
df.index.names = ['Groups', 'Num']
```

In [213…
```
df
```

Out[213…

		A	B
Groups	**Num**		
G1	**1**	6	50
	2	68	52
	3	90	37
G2	**1**	48	45
	2	64	44
	3	58	59

In [214…
```
#cross section
df.xs('G1')
```

Out[214…

	A	B
Num		
1	6	50
2	68	52
3	90	37

In [215…
```
df.xs(1, level='Num')
```

Out[215…

	A	B
Groups		
G1	6	50
G2	48	45

Missing Data

In [218…
```
d_data2 = {'Deziri':[1,2,np.nan], 'Finest':[5, np.nan, np.nan], 'David':[1,
d_data2
```

Out[218…
```
{'Deziri': [1, 2, nan], 'Finest': [5, nan, nan], 'David': [1, 2, 3]}
```

In [219…
```
df = pd.DataFrame(d_data2)
```

In [220…
```
df
```

Out[220…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [223…
```
df.dropna()
```

Out[223…

	Deziri	Finest	David
0	1.0	5.0	1

In [224…
```
df.dropna(axis=1)
```

Out[224…

	David
0	1
1	2
2	3

In [225…
```
df.dropna(thresh=2)
```

Out[225…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	NaN	2

In [226…
```
df.dropna(thresh=2, axis=1)
```

Out[226…

	Deziri	David
0	1.0	1
1	2.0	2
2	NaN	3

In [228…
```
df.dropna(thresh=1)
```

Out[228…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [229…

```
df
```

Out[229…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [230…

```
#to replace NaN with a value:
df.fillna(value=3)
```

Out[230…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	3.0	2
2	3.0	3.0	3

In [231…

```
df.fillna(value=0)
```

Out[231…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	0.0	2
2	0.0	0.0	3

In [232…

```
df.mean()
```

Out[232…

```
Deziri    1.5
Finest    5.0
David     2.0
dtype: float64
```

In [233…

```
df.fillna(value=df.mean())
```

Out[233…

	Deziri	Finest	David
0	1.0	5.0	1
1	2.0	5.0	2
2	1.5	5.0	3

Groupby allows you to group together rows based off of a column and perform an aggregate function on them

In [240…

```
#data generated from https://mockaroo.com
data = [{
  "Company": "Topicstorm",
  "Person": "Matelda",
  "Sales": 455
}, {
  "Company": "Topicstorm",
  "Person": "Lief",
  "Sales": 474
}, {
  "Company": "Kimia",
  "Person": "Albie",
  "Sales": 152
}, {
  "Company": "Kimia",
  "Person": "Robby",
  "Sales": 292
}, {
  "Company": "Eabox",
  "Person": "Fernando",
  "Sales": 174
}, {
  "Company": "Eabox",
  "Person": "Corliss",
  "Sales": 137
}]
```

In [241…

```
df = pd.DataFrame(data)
```

In [242…

```
df
```

Out[242…

	Company	Person	Sales
0	Topicstorm	Matelda	455
1	Topicstorm	Lief	474
2	Kimia	Albie	152
3	Kimia	Robby	292
4	Eabox	Fernando	174
5	Eabox	Corliss	137

In [245…
```
Comp = df.groupby('Company')
Comp
```

Out[245… `<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fa59e9eedc0>`

In [246…
```
Comp.sum()
```

Out[246…

Company	Sales
Eabox	311
Kimia	444
Topicstorm	929

In [247…
```
Comp.mean()
```

Out[247…

Company	Sales
Eabox	155.5
Kimia	222.0
Topicstorm	464.5

In [248…
```
Comp.max()
```

Out[248…

Company	Person	Sales
Eabox	Fernando	174
Kimia	Robby	292
Topicstorm	Matelda	474

In [249…
```
Comp.min()
```

Out[249…

Company	Person	Sales
Eabox	Corliss	137
Kimia	Albie	152
Topicstorm	Lief	455

In [251…
```
Comp.std()
```

Out[251…

Company	Sales
Eabox	26.162951
Kimia	98.994949
Topicstorm	13.435029

In [252…
```
df.groupby('Company').sum().loc['Kimia']
```

Out[252…
```
Sales    444
Name: Kimia, dtype: int64
```

In [256…
```
df.describe()
```

Out[256…

	Sales
count	6.000000
mean	280.666667
std	152.624594
min	137.000000
25%	157.500000
50%	233.000000
75%	414.250000
max	474.000000

In [255…
```
df.groupby('Company').describe()
```

Out[255…

	Sales							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
Company								
Eabox	2.0	155.5	26.162951	137.0	146.25	155.5	164.75	174.0
Kimia	2.0	222.0	98.994949	152.0	187.00	222.0	257.00	292.0
Topicstorm	2.0	464.5	13.435029	455.0	459.75	464.5	469.25	474.0

In [257…
```
df.groupby('Company').describe().transpose()
```

Out[257…

Company		Eabox	Kimia	Topicstorm
Sales	**count**	2.000000	2.000000	2.000000
	mean	155.500000	222.000000	464.500000
	std	26.162951	98.994949	13.435029
	min	137.000000	152.000000	455.000000
	25%	146.250000	187.000000	459.750000
	50%	155.500000	222.000000	464.500000
	75%	164.750000	257.000000	469.250000
	max	174.000000	292.000000	474.000000

Merging, Joining and Concatenating

In [264…

```
df1 = pd.DataFrame([{
  "Code": "NG",
  "City": "Gujba",
  "Car": "Ford",
  "Color": "Puce"
}, {
  "Code": "GR",
  "City": "Melíssia",
  "Car": "Volkswagen",
  "Color": "Violet"
}, {
  "Code": "ID",
  "City": "Ajung",
  "Car": "Cadillac",
  "Color": "Purple"
}, {
  "Code": "MM",
  "City": "Lashio",
  "Car": "Mercury",
  "Color": "Khaki"
}], index=[0,1,2,3])
```

In [266…

```
df2 = pd.DataFrame([{
  "Code": "CN",
  "City": "Donghuang",
  "Car": "BMW",
  "Color": "Goldenrod"
}, {
  "Code": "AU",
  "City": "Sydney",
  "Car": "Pontiac",
  "Color": "Yellow"
}, {
  "Code": "FR",
  "City": "Saint-Claude",
  "Car": "Chevrolet",
  "Color": "Turquoise"
}, {
  "Code": "SN",
  "City": "Joal-Fadiout",
  "Car": "Hyundai",
  "Color": "Red"
}], index=[4,5,6,7])
```

In [271…

```
df3 = pd.DataFrame([{
  "Code": "CM",
  "City": "Yuandun",
  "Car": "Suzuki",
  "Color": "Indigo"
}, {
  "Code": "MD",
  "City": "Rîbniţa",
  "Car": "Plymouth",
  "Color": "Pink"
}, {
  "Code": "GH",
  "City": "Nidri",
  "Car": "Dodge",
  "Color": "Blue"
}, {
  "Code": "MN",
  "City": "Uliastay",
  "Car": "Chrysler",
  "Color": "Aquamarine"
}], index=[8,9,10,11])
```

In [268…

```
df1
```

Out[268…

	Code	City	Car	Color
0	NG	Gujba	Ford	Puce
1	GR	Melíssia	Volkswagen	Violet
2	ID	Ajung	Cadillac	Purple
3	MM	Lashio	Mercury	Khaki

In [269…

```
df2
```

Out[269…

	Code	City	Car	Color
4	CN	Donghuang	BMW	Goldenrod
5	AU	Sydney	Pontiac	Yellow
6	FR	Saint-Claude	Chevrolet	Turquoise
7	SN	Joal-Fadiout	Hyundai	Red

In [272…

```
df3
```

Out[272…

	Code	City	Car	Color
8	CM	Yuandun	Suzuki	Indigo
9	MD	Rîbniţa	Plymouth	Pink
10	GH	Nidri	Dodge	Blue
11	MN	Uliastay	Chrysler	Aquamarine

Concatenation glues together Dataframes

In [273…

```
pd.concat([df1,df2,df3])
```

Out[273…

	Code	City	Car	Color
0	NG	Gujba	Ford	Puce
1	GR	Melíssia	Volkswagen	Violet
2	ID	Ajung	Cadillac	Purple
3	MM	Lashio	Mercury	Khaki
4	CN	Donghuang	BMW	Goldenrod
5	AU	Sydney	Pontiac	Yellow
6	FR	Saint-Claude	Chevrolet	Turquoise
7	SN	Joal-Fadiout	Hyundai	Red
8	CM	Yuandun	Suzuki	Indigo
9	MD	Rîbniţa	Plymouth	Pink
10	GH	Nidri	Dodge	Blue
11	MN	Uliastay	Chrysler	Aquamarine

In [274…

```
pd.concat([df1,df2,df3], axis=1)
```

Out[274...

	Code	City	Car	Color	Code	City	Car	Color	Code	C
0	NG	Gujba	Ford	Puce	NaN	NaN	NaN	NaN	NaN	N
1	GR	Melíssia	Volkswagen	Violet	NaN	NaN	NaN	NaN	NaN	N
2	ID	Ajung	Cadillac	Purple	NaN	NaN	NaN	NaN	NaN	N
3	MM	Lashio	Mercury	Khaki	NaN	NaN	NaN	NaN	NaN	N
4	NaN	NaN	NaN	NaN	CN	Donghuang	BMW	Goldenrod	NaN	N
5	NaN	NaN	NaN	NaN	AU	Sydney	Pontiac	Yellow	NaN	N
6	NaN	NaN	NaN	NaN	FR	Saint-Claude	Chevrolet	Turquoise	NaN	N
7	NaN	NaN	NaN	NaN	SN	Joal-Fadiout	Hyundai	Red	NaN	N
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	CM	Yuanc
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	MD	Rîbr
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	GH	N
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	MN	Ulias

In [275...

```
pd.concat([df1,df2,df3]).transpose()
```

Out[275...

	0	1	2	3	4	5	6	7	8
Code	NG	GR	ID	MM	CN	AU	FR	SN	CM
City	Gujba	Melíssia	Ajung	Lashio	Donghuang	Sydney	Saint-Claude	Joal-Fadiout	Yuandur
Car	Ford	Volkswagen	Cadillac	Mercury	BMW	Pontiac	Chevrolet	Hyundai	Suzuk
Color	Puce	Violet	Purple	Khaki	Goldenrod	Yellow	Turquoise	Red	Indigo

In [276...

```
left=pd.DataFrame([{
  "Year": 1988,
  "Car": "Pontiac",
  "Color": "Fuscia"
}, {
  "Year": 1998,
  "Car": "Porsche",
  "Color": "Red"
}, {
  "Year": 2008,
  "Car": "Nissan",
  "Color": "Mauv"
}, {
  "Year": 2018,
  "Car": "Suzuki",
  "Color": "White"
}])
```

In [280…

```
right=pd.DataFrame([{
   "Year": 1988,
   "Stock": "The Charles Schwab Corporation",
   "Cap": "$800.7M"
}, {
   "Year": 1998,
   "Stock": "FibroGen, Inc",
   "Cap": "$2.08B"
}, {
   "Year": 2008,
   "Stock": "Cross Country Healthcare, Inc.",
   "Cap": "$476.55M"
}, {
   "Year": 2018,
   "Stock": "U.S. Silica Holdings, Inc.",
   "Cap": "$2.75B"
}])
```

In [281…

```
left
```

Out[281…

	Year	Car	Color
0	1988	Pontiac	Fuscia
1	1998	Porsche	Red
2	2008	Nissan	Mauv
3	2018	Suzuki	White

In [282…

```
right
```

Out[282…

	Year	Stock	Cap
0	1988	The Charles Schwab Corporation	$800.7M
1	1998	FibroGen, Inc	$2.08B
2	2008	Cross Country Healthcare, Inc.	$476.55M
3	2018	U.S. Silica Holdings, Inc.	$2.75B

Merging

In [283…

```
pd.merge(left,right, how='inner', on='Year')
```

Out[283…

	Year	Car	Color	Stock	Cap
0	1988	Pontiac	Fuscia	The Charles Schwab Corporation	$800.7M
1	1998	Porsche	Red	FibroGen, Inc	$2.08B
2	2008	Nissan	Mauv	Cross Country Healthcare, Inc.	$476.55M
3	2018	Suzuki	White	U.S. Silica Holdings, Inc.	$2.75B

In [284…

```
pd.merge(left,right, how='outer', on='Year')
```

Out[284…

	Year	Car	Color	Stock	Cap
0	1988	Pontiac	Fuscia	The Charles Schwab Corporation	$800.7M
1	1998	Porsche	Red	FibroGen, Inc	$2.08B
2	2008	Nissan	Mauv	Cross Country Healthcare, Inc.	$476.55M
3	2018	Suzuki	White	U.S. Silica Holdings, Inc.	$2.75B

Joining: different index

In [292…

```
left=pd.DataFrame([{
  "Car": "Pontiac",
  "Color": "Fuscia"
}, {
  "Car": "Porsche",
  "Color": "Red"
}, {
  "Car": "Nissan",
  "Color": "Mauv"
}, {
  "Car": "Suzuki",
  "Color": "White"
}], index=[0,1,2,4])
```

In [293…

```
right=pd.DataFrame([{
  "Year": 1988,
  "Stock": "The Charles Schwab Corporation",
  "Cap": "$800.7M"
}, {
  "Year": 1998,
  "Stock": "FibroGen, Inc",
  "Cap": "$2.08B"
}, {
  "Year": 2008,
  "Stock": "Cross Country Healthcare, Inc.",
  "Cap": "$476.55M"
}, {
  "Year": 2018,
  "Stock": "U.S. Silica Holdings, Inc.",
  "Cap": "$2.75B"
}], index=[0,2,5,3])
```

In [294…

```
left
```

Out[294…

	Car	Color
0	Pontiac	Fuscia
1	Porsche	Red
2	Nissan	Mauv
4	Suzuki	White

In [295…

```
right
```

Out[295…

	Year	Stock	Cap
0	1988	The Charles Schwab Corporation	$800.7M
2	1998	FibroGen, Inc	$2.08B
5	2008	Cross Country Healthcare, Inc.	$476.55M
3	2018	U.S. Silica Holdings, Inc.	$2.75B

In [296…

```
left.join(right)
```

Out[296…

	Car	Color	Year	Stock	Cap
0	Pontiac	Fuscia	1988.0	The Charles Schwab Corporation	$800.7M
1	Porsche	Red	NaN	NaN	NaN
2	Nissan	Mauv	1998.0	FibroGen, Inc	$2.08B
4	Suzuki	White	NaN	NaN	NaN

In [299…

```
left.join(right, how='outer')
```

Out[299…

	Car	Color	Year	Stock	Cap
0	Pontiac	Fuscia	1988.0	The Charles Schwab Corporation	$800.7M
1	Porsche	Red	NaN	NaN	NaN
2	Nissan	Mauv	1998.0	FibroGen, Inc	$2.08B
3	NaN	NaN	2018.0	U.S. Silica Holdings, Inc.	$2.75B
4	Suzuki	White	NaN	NaN	NaN
5	NaN	NaN	2008.0	Cross Country Healthcare, Inc.	$476.55M

In [300…

```
right.join(left)
```

Out[300…

	Year	Stock	Cap	Car	Color
0	1988	The Charles Schwab Corporation	$800.7M	Pontiac	Fuscia
2	1998	FibroGen, Inc	$2.08B	Nissan	Mauv
5	2008	Cross Country Healthcare, Inc.	$476.55M	NaN	NaN
3	2018	U.S. Silica Holdings, Inc.	$2.75B	NaN	NaN

In [301…

```
right.join(left, how='outer')
```

Out[301…

	Year	Stock	Cap	Car	Color
0	1988.0	The Charles Schwab Corporation	$800.7M	Pontiac	Fuscia
1	NaN	NaN	NaN	Porsche	Red
2	1998.0	FibroGen, Inc	$2.08B	Nissan	Mauv
3	2018.0	U.S. Silica Holdings, Inc.	$2.75B	NaN	NaN
4	NaN	NaN	NaN	Suzuki	White
5	2008.0	Cross Country Healthcare, Inc.	$476.55M	NaN	NaN

Operations

.unique() : to get the unique items in a series

.nunique() : to get the number on unique items

.value_counts(): it returns the number of times each unique item occured

if you create a method and you want to apply it on your table: .apply(name_of_method)

In [302…

```
right
```

Out[302…

	Year	Stock	Cap
0	1988	The Charles Schwab Corporation	$800.7M
2	1998	FibroGen, Inc	$2.08B
5	2008	Cross Country Healthcare, Inc.	$476.55M
3	2018	U.S. Silica Holdings, Inc.	$2.75B

In [303…

```
right['Stock'].apply(len)
```

Out[303…

```
0    30
2    13
5    30
3    26
Name: Stock, dtype: int64
```

In [307…

```
def next5years(year):
    return year+5

right['Year'].apply(next5years)
```

Out[307…

```
0    1993
2    2003
5    2013
3    2023
Name: Year, dtype: int64
```

In [306…

```
# or
right['Year'].apply(lambda x: x+5)
```

Out[306…

```
0    1993
2    2003
5    2013
3    2023
Name: Year, dtype: int64
```

In [308…

```
right.index
```

Out[308…

```
Int64Index([0, 2, 5, 3], dtype='int64')
```

In [311…

```
right.columns
```

Out[311…

```
Index(['Year', 'Stock', 'Cap'], dtype='object')
```

In [313…

```
right.sort_values('Cap')
```

Out[313…

	Year	Stock	Cap
2	1998	FibroGen, Inc	$2.08B
3	2018	U.S. Silica Holdings, Inc.	$2.75B
5	2008	Cross Country Healthcare, Inc.	$476.55M
0	1988	The Charles Schwab Corporation	$800.7M

In [314…

```
right.join(left, how='outer').isnull()
```

Out[314…

	Year	Stock	Cap	Car	Color
0	False	False	False	False	False
1	True	True	True	False	False
2	False	False	False	False	False
3	False	False	False	True	True
4	True	True	True	False	False
5	False	False	False	True	True

In [318…

```
data = [{
  "Company": "Topicstorm",
  "Person": "Matelda",
  "Sales": 455,
    "num": 1
}, {
  "Company": "Topicstorm",
  "Person": "Matelda",
  "Sales": 137,
    "num": 3
}, {
  "Company": "Topicstorm",
  "Person": "Albie",
  "Sales": 455,
    "num": 2
}, {
  "Company": "Eabox",
  "Person": "Albie",
  "Sales": 137,
    "num": 5
}, {
  "Company": "Eabox",
  "Person": "Fernando",
  "Sales": 455,
    "num": 4
}, {
  "Company": "Eabox",
  "Person": "Fernando",
  "Sales": 137,
    "num": 1
}]
df=pd.DataFrame(data)
```

In [319…

```
df
```

Out[319…

	Company	Person	Sales	num
0	Topicstorm	Matelda	455	1
1	Topicstorm	Matelda	137	3
2	Topicstorm	Albie	455	2
3	Eabox	Albie	137	5
4	Eabox	Fernando	455	4
5	Eabox	Fernando	137	1

In [320…

```
df.pivot_table(values='num', index = ['Company', 'Person'], columns=['Sales
```

Out[320…

		Sales	137	455
Company	**Person**			
Eabox	**Albie**		5.0	NaN
	Fernando		1.0	4.0
Topicstorm	**Albie**		NaN	2.0
	Matelda		3.0	1.0

Data Input and Output

CSV

Excel

HTML

SQL

To work with HTML we need the following librarys: sqlalchemy, lxml, html5lib, BeautifulSoup4

In [321…

```
# to read csv file:
pd.read_csv('/Users/chukwunonsodavid/Downloads/car.csv')
```

Out[321...

	make	model	price
0	Hyundai	Elantra	8118.17
1	Ford	F-Series	3575.86
2	Toyota	TundraMax	2180.36
3	Chevrolet	Suburban 1500	3112.80
4	GMC	Jimmy	2916.69
...	...	...	...
95	Pontiac	Montana	9378.56
96	Buick	Coachbuilder	9069.88
97	Hyundai	Santa Fe	7640.71
98	GMC	Envoy	5013.41
99	Volkswagen	Golf III	2693.57

100 rows × 3 columns

pd.read_ click tab to display all the options and files pd can read

In [322...

```
df = pd.read_csv('/Users/chukwunonsodavid/Downloads/car.csv').head(10)
```

In [323...

```
df
```

Out[323...

	make	model	price
0	Hyundai	Elantra	8118.17
1	Ford	F-Series	3575.86
2	Toyota	TundraMax	2180.36
3	Chevrolet	Suburban 1500	3112.80
4	GMC	Jimmy	2916.69
5	Dodge	Caravan	8536.79
6	Chevrolet	Tahoe	6971.71
7	Subaru	Impreza	9163.17
8	Dodge	Viper	1913.57
9	Scion	xB	8090.91

In [325...

```
# to save to csv:
df.to_csv('/Users/chukwunonsodavid/Downloads/car10.csv', index=False)
```

In [326...

```
newdf = pd.read_csv('/Users/chukwunonsodavid/Downloads/car10.csv')
newdf
```

Out[326…

	make	model	price
0	Hyundai	Elantra	8118.17
1	Ford	F-Series	3575.86
2	Toyota	TundraMax	2180.36
3	Chevrolet	Suburban 1500	3112.80
4	GMC	Jimmy	2916.69
5	Dodge	Caravan	8536.79
6	Chevrolet	Tahoe	6971.71
7	Subaru	Impreza	9163.17
8	Dodge	Viper	1913.57
9	Scion	xB	8090.91

In [328…

```
#to read excel files:
#pd.read_excel('file.xlsx', sheetname='Sheet1')
#to write excel files:
#df.to_excel('file.xlsx', sheet_name='Sheet1')
```

In [ ]:

```
#to read HTML:
#pd.read_html('link')
```