Practical Tutorial on Data Manipulation with Numpy and Pandas in Python:-

The pandas library has emerged into a power house of data manipulation tasks in python since it was developed in 2008. With its intuitive syntax and flexible data structure, it's easy to learn and enables faster data computation. The development of numpy and pandas libraries has extended python's multi-purpose nature to solve machine learning problems as well. The acceptance of python language in machine learning has been phenomenal since then.

6 Important things about Numpy and Pandas:-

1. The data manipulation capabilities of pandas are built on top of the numpy library. In a way, numpy is a dependency of the pandas library.
2. Pandas is best at handling tabular data sets comprising different variable types (integer, float, double, etc.). In addition, the pandas library can also be used to perform even the most naive of tasks such as loading data or doing feature engineering on time series data.
3. Numpy is most suitable for performing basic numerical computations such as mean, median, range, etc. Alongside, it also supports the creation of multi-dimensional arrays.
4. Numpy library can also be used to integrate C/C++ and Fortran code.
5. Remember, python is a zero indexing language unlike R where indexing starts at one.
6. The best part of learning pandas and numpy is the strong active community support you'll get from around the world

a) STARTING WITH NUMPY:-

In [5]:
```
import numpy as np
```

In [6]:
```
L = list(range(10))
```

In [29]:
```
#converting integers to string - this style of handling lists is known as list comprehension.
#List comprehension offers a versatile way to handle list manipulations tasks easily. We'll learn about them in future tutorials. Here's an example.

[str(c) for c in L]
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

[type(item) for item in L]
```

Out[29]:
```
[int, int, int, int, int, int, int, int, int, int]
```

a) Creation of array:-

In [28]:
```
#creating arrays
np.zeros(10, dtype='int')
np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])


#creating a 3 row x 5 column matrix
np.ones((3,5), dtype=float)
np.array([[ 1.,  1.,  1.,  1.,  1.],
      [ 1.,  1.,  1.,  1.,  1.],
      [ 1.,  1.,  1.,  1.,  1.]])


#creating a matrix with a predefined value
np.full((3,5),1.23)
np.array([[ 1.23,  1.23,  1.23,  1.23,  1.23],
      [ 1.23,  1.23,  1.23,  1.23,  1.23],
      [ 1.23,  1.23,  1.23,  1.23,  1.23]])


#create an array with a set sequence
np.arange(0, 20, 2)
np.array([0, 2, 4, 6, 8,10,12,14,16,18])


#create an array of even space between the given range of values
np.linspace(0, 1, 5)
np.array([ 0., 0.25, 0.5 , 0.75, 1.])


#create a 3x3 array with mean 0 and standard deviation 1 in a given dimension
np.random.normal(0, 1, (3,3))
np.array([[ 0.72432142, -0.90024075,  0.27363808],
      [ 0.88426129,  1.45096856, -1.03547109],
      [-0.42930994, -1.02284441, -1.59753603]])


#create an identity matrix
np.eye(3)
np.array([[ 1.,  0.,  0.],
      [ 0.,  1.,  0.],
      [ 0.,  0.,  1.]])


#set a random seed
np.random.seed(0)


x1 = np.random.randint(10, size=6) #one dimension
x2 = np.random.randint(10, size=(3,4)) #two dimension
x3 = np.random.randint(10, size=(3,4,5)) #three dimension


print("x3 ndim:", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)
```

```
x3 ndim: 3
x3 shape: (3, 4, 5)
x3 size:  60
```

b) Array Indexing:-

In [35]:
```
x1 = np.array([4, 3, 4, 4, 8, 4])
x1
np.array([4, 3, 4, 4, 8, 4])

#assess value to index zero
x1[0]
4

#assess fifth value
x1[4]
8

#get the last value
x1[-1]
4

#get the second last value
x1[-2]
8

#in a multidimensional array, we need to specify row and column index
x2
np.array([[3, 7, 5, 5],
      [0, 1, 5, 9],
      [3, 0, 5, 0]])


#1st row and 2nd column value
x2[2,3]
0

#3rd row and last value from the 3rd column
x2[2,-1]
0


#replace value at 0,0 index
x2[0,0] = 12
x2
```

Out[35]:
```
array([[12,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6,  7,  7]])
```

c) Array Slicing:-

In [27]:
```
x1 = np.array([4, 3, 4, 4, 8, 4])
x1
np.array([4, 3, 4, 4, 8, 4])

#assess value to index zero
x1[0]
4

#assess fifth value
x1[4]
8

#get the last value
x1[-1]
4

#get the second last value
x1[-2]
8

#in a multidimensional array, we need to specify row and column index
x2= np.array([[3, 7, 5, 5],
      [0, 1, 5, 9],
      [3, 0, 5, 0]])


#1st row and 2nd column value
x2[2,3]
0

#3rd row and last value from the 3rd column
x2[2,-1]
0


#replace value at 0,0 index
x2[0,0] = 12
x2
np.array([[12,  7,  5,  5],
      [ 0,  1,  5,  9],
      [ 3,  0,  5,  0]])
#Array Slicing
#Now, we'll learn to access multiple or a range of elements from an array.

x = np.arange(10)
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])


#from start to 4th position
x[:5]
np.array([0, 1, 2, 3, 4])


#from 4th position to end
x[4:]
np.array([4, 5, 6, 7, 8, 9])


#from 4th to 6th position
x[4:7]
np.array([4, 5, 6])


#return elements at even place
x[ : : 2]
np.array([0, 2, 4, 6, 8])


#return elements from first position step by two
x[1::2]
np.array([1, 3, 5, 7, 9])


#reverse the array
x[::-1]
```

Out[27]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

d) Array Concatination:-

In [38]:
```
#You can concatenate two or more arrays at once.
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = [21,21,21]
np.concatenate([x, y,z])
np.array([ 1,  2,  3,  3,  2,  1, 21, 21, 21])


#You can also use this function to create 2-dimensional arrays.
grid = np.array([[1,2,3],[4,5,6]])
np.concatenate([grid,grid])
np.array([[1, 2, 3],
      [4, 5, 6],
      [1, 2, 3],
      [4, 5, 6]])


#Using its axis parameter, you can define row-wise or column-wise matrix
np.concatenate([grid,grid],axis=1)
```

Out[38]:
```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

b) STARTING WITH PANDAS:-

In [39]:
```
import pandas as pd
```

In [42]:
```
#create a data frame - dictionary is used here where keys get converted to column names and values to row values.
data = pd.DataFrame({'Country': ['Russia','Colombia','Chile','Equador','Nigeria'],
                    'Rank':[121,40,100,130,11]})
data
```

Out[42]:

	Country	Rank
0	Russia	121
1	Colombia	40
2	Chile	100
3	Equador	130
4	Nigeria	11

In [43]:
```
#We can do a quick analysis of any data set using:
data.describe()
```

Out[43]:

	Rank
count	5.000000
mean	80.400000
std	52.300096
min	11.000000
25%	40.000000
50%	100.000000
75%	121.000000
max	130.000000

In [55]:
```
#Let's create another data frame.
data = pd.DataFrame({'group':['a', 'a', 'a', 'b','b', 'b', 'c', 'c','c'],'ounces':[4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

Out[55]:

	group	ounces
0	a	4.0
1	a	3.0
2	a	12.0
3	b	6.0
4	b	7.5
5	b	8.0
6	c	3.0
7	c	5.0
8	c	6.0

In [57]:
```
#Let's sort the data frame by ounces - inplace = True will make changes to the data
data.sort_values(by=['ounces'],ascending=True,inplace=False)
```

Out[57]:

	group	ounces
1	a	3.0
6	c	3.0
0	a	4.0
7	c	5.0
3	b	6.0
8	c	6.0
4	b	7.5
5	b	8.0
2	a	12.0

In [58]:
```
data.sort_values(by=['group','ounces'],ascending=[True,False],inplace=False)
```

Out[58]:

	group	ounces
2	a	12.0
0	a	4.0
1	a	3.0
5	b	8.0
4	b	7.5
3	b	6.0
8	c	6.0
7	c	5.0
6	c	3.0

In [59]:
```
#create another data with duplicated rows
data = pd.DataFrame({'k1':['one']*3 + ['two']*4, 'k2':[3,2,1,3,3,4,4]})
data
```

Out[59]:

	k1	k2
0	one	3
1	one	2
2	one	1
3	two	3
4	two	3
5	two	4
6	two	4

In [60]:
```
#sort values
data.sort_values(by='k2')
```

Out[60]:

	k1	k2
2	one	1
1	one	2
0	one	3
3	two	3
4	two	3
5	two	4
6	two	4

In [61]:
```
#remove duplicates - ta da!
data.drop_duplicates()
```

Out[61]:

	k1	k2
0	one	3
1	one	2
2	one	1
3	two	3
5	two	4

In [62]:
```
data.drop_duplicates(subset='k1')
```

Out[62]:

	k1	k2
0	one	3
3	two	3

In [63]:
```
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon', 'Pastrami','corned beef', 'Bacon', 'pastrami', 'honey ham','nova lox'],
                 'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

Out[63]:

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

In [64]:
```
meat_to_animal = {
'bacon': 'pig',
'pulled pork': 'pig',
'pastrami': 'cow',
'corned beef': 'cow',
'honey ham': 'pig',
'nova lox': 'salmon'
}

def meat_2_animal(series):
    if series['food'] == 'bacon':
        return 'pig'
    elif series['food'] == 'pulled pork':
        return 'pig'
    elif series['food'] == 'pastrami':
        return 'cow'
    elif series['food'] == 'corned beef':
        return 'cow'
    elif series['food'] == 'honey ham':
        return 'pig'
    else:
        return 'salmon'


#create a new variable
data['animal'] = data['food'].map(str.lower).map(meat_to_animal)
data
```

Out[64]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

In [65]:
```
#another way of doing it is: convert the food values to the lower case and apply the function
lower = lambda x: x.lower()
data['food'] = data['food'].apply(lower)
data['animal2'] = data.apply(meat_2_animal, axis='columns')
data
```

Out[65]:

	food	ounces	animal	animal2
0	bacon	4.0	pig	pig
1	pulled pork	3.0	pig	pig
2	bacon	12.0	pig	pig
3	pastrami	6.0	cow	cow
4	corned beef	7.5	cow	cow
5	bacon	8.0	pig	pig
6	pastrami	3.0	cow	cow
7	honey ham	5.0	pig	pig
8	nova lox	6.0	salmon	salmon

In [66]:
```
data.assign(new_variable = data['ounces']*10)
```

Out[66]:

	food	ounces	animal	animal2	new_variable
0	bacon	4.0	pig	pig	40.0
1	pulled pork	3.0	pig	pig	30.0
2	bacon	12.0	pig	pig	120.0
3	pastrami	6.0	cow	cow	60.0
4	corned beef	7.5	cow	cow	75.0
5	bacon	8.0	pig	pig	80.0
6	pastrami	3.0	cow	cow	30.0
7	honey ham	5.0	pig	pig	50.0
8	nova lox	6.0	salmon	salmon	60.0

In [67]:
```
data.drop('animal2',axis='columns',inplace=True)
data
```

Out[67]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

In [68]:
```
#Series function from pandas are used to create arrays
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
```

Out[68]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [69]:
```
#replace -999 with NaN values
data.replace(-999, np.nan,inplace=True)
data
```

Out[69]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [70]:
```
#We can also replace multiple values at once.
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data.replace([-999,-1000],np.nan,inplace=True)
data
```

Out[70]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [71]:
```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),index=['Ohio', 'Colorado', 'New York'],columns=['one', 'two', 'three', 'four'])
data
```

Out[71]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
New York	8	9	10	11

In [72]:
```
#Using rename function
data.rename(index = {'Ohio':'SanF'}, columns={'one':'one_p','two':'two_p'},inplace=True)
data
```

Out[72]:

	one_p	two_p	three	four
SanF	0	1	2	3
Colorado	4	5	6	7
New York	8	9	10	11

In [73]:
```
#You can also use string functions
data.rename(index = str.upper, columns=str.title,inplace=True)
data
```

Out[73]:

	One_P	Two_P	Three	Four
SANF	0	1	2	3
COLORADO	4	5	6	7
NEW YORK	8	9	10	11

In [77]:
```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [78]:
```
#Understand the output - '(' means the value is included in the bin, '[' means the value is excluded
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)
cats
```

Out[78]:
```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

In [79]:
```
#To include the right bin value, we can do:
pd.cut(ages,bins,right=False)
```

Out[79]:
```
[[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), ..., [25, 35), [60, 100), [35, 60), [35, 60), [25, 35)]
Length: 12
Categories (4, interval[int64, left]): [[18, 25) < [25, 35) < [35, 60) < [60, 100)]
```

In [82]:
```
#Let's check how many observations fall under each bin
pd.value_counts(cats)
```

Out[82]:
```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
dtype: int64
```

In [83]:
```
bin_names = ['Youth', 'YoungAdult', 'MiddleAge', 'Senior']
new_cats = pd.cut(ages, bins,labels=bin_names)

pd.value_counts(new_cats)
```

Out[83]:
```
Youth         5
YoungAdult    3
MiddleAge     3
Senior        1
dtype: int64
```

In [84]:
```
#we can also calculate their cumulative sum
pd.value_counts(new_cats).cumsum()
```

Out[84]:
```
Youth          5
YoungAdult     8
MiddleAge     11
Senior        12
dtype: int64
```

In [85]:
```
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                  'key2' : ['one', 'two', 'one', 'two', 'one'],
                  'data1' : np.random.randn(5),
                  'data2' : np.random.randn(5)})
df
```

Out[85]:

	key1	key2	data1	data2
0	a	one	1.254414	1.149076
1	a	two	1.419102	-1.193578
2	b	one	-0.743856	1.141042
3	b	two	-2.517437	1.509445
4	a	one	-1.507096	1.067775

In [88]:
```
dates = pd.date_range('20130101',periods=6)
df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
df
```

Out[88]:

	A	B	C	D
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404
2013-01-03	0.724006	0.359003	1.076121	0.192141
2013-01-04	0.852926	0.018357	0.428304	0.996278
2013-01-05	-0.491150	0.712678	1.113340	-2.153675
2013-01-06	-0.416111	-1.070897	0.221139	-1.123057

In [89]:
```
#get first n rows from the data frame
df[:3]
```

Out[89]:

	A	B	C	D
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404
2013-01-03	0.724006	0.359003	1.076121	0.192141

In [90]:
```
#get first n rows from the data frame
df[:3]
```

Out[90]:

	A	B	C	D
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404
2013-01-03	0.724006	0.359003	1.076121	0.192141

In [91]:
```
#slicing based on column names
df.loc[:,['A','B']]
```

Out[91]:

	A	B
2013-01-01	-0.686589	0.014873
2013-01-02	0.367974	-0.044724
2013-01-03	0.724006	0.359003
2013-01-04	0.852926	0.018357
2013-01-05	-0.491150	0.712678
2013-01-06	-0.416111	-1.070897

In [92]:
```
#slicing based on both row index labels and column names
df.loc['20130102':'20130103',['A','B']]
```

Out[92]:

	A	B
2013-01-02	0.367974	-0.044724
2013-01-03	0.724006	0.359003

In [93]:
```
#slicing based on index of columns
df.iloc[3] #returns 4th row (index is 3rd)
```

Out[93]:
```
A    0.852926
B    0.018357
C    0.428304
D    0.996278
Name: 2013-01-04 00:00:00, dtype: float64
```

In [94]:
```
#returns a specific range of rows
df.iloc[2:4, 0:2]
```

Out[94]:

	A	B
2013-01-03	0.724006	0.359003
2013-01-04	0.852926	0.018357

In [95]:
```
#returns specific rows and columns using lists containing columns or row indexes
df.iloc[[1,5],[0,2]]
```

Out[95]:

	A	C
2013-01-02	0.367974	-0.302375
2013-01-06	-0.416111	0.221139

In [96]:
```
df[df.A > 1]
```

Out[96]:

A	B	C	D

In [97]:
```
#we can copy the data set
df2 = df.copy()
df2['E']=['one', 'one','two','three','four','three']
df2
```

Out[97]:

	A	B	C	D	E
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224	one
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404	one
2013-01-03	0.724006	0.359003	1.076121	0.192141	two
2013-01-04	0.852926	0.018357	0.428304	0.996278	three
2013-01-05	-0.491150	0.712678	1.113340	-2.153675	four
2013-01-06	-0.416111	-1.070897	0.221139	-1.123057	three

In [98]:
```
#select rows based on column values
df2[df2['E'].isin(['two','four'])]
```

Out[98]:

	A	B	C	D	E
2013-01-03	0.724006	0.359003	1.076121	0.192141	two
2013-01-05	-0.491150	0.712678	1.113340	-2.153675	four

In [99]:
```
#select all rows except those with two and four
df2[~df2['E'].isin(['two','four'])]
```

Out[99]:

	A	B	C	D	E
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224	one
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404	one
2013-01-04	0.852926	0.018357	0.428304	0.996278	three
2013-01-06	-0.416111	-1.070897	0.221139	-1.123057	three

In [100…
```
#list all columns where A is greater than C
df.query('A > C')
```

Out[100]:

	A	B	C	D
2013-01-02	0.367974	-0.044724	-0.302375	-2.224404
2013-01-04	0.852926	0.018357	0.428304	0.996278

In [101…
```
#using OR condition
df.query('A < B | C > A')
```

Out[101]:

	A	B	C	D
2013-01-01	-0.686589	0.014873	-0.375666	-0.038224
2013-01-03	0.724006	0.359003	1.076121	0.192141
2013-01-05	-0.491150	0.712678	1.113340	-2.153675
2013-01-06	-0.416111	-1.070897	0.221139	-1.123057

In [102…
```
#create a data frame
data = pd.DataFrame({'group': ['a', 'a', 'a', 'b','b', 'b', 'c', 'c','c'],
                 'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

Out[102]:

	group	ounces
0	a	4.0
1	a	3.0
2	a	12.0
3	b	6.0
4	b	7.5
5	b	8.0
6	c	3.0
7	c	5.0
8	c	6.0