Pandas is used to analyzze big data and make conclusions based on statistical theories.

Pandas is typically used to clean messy data and make them readable and simple to access

Pandas can give co-relation between two or more columns

It can also help us in doing all statistical methodologies as well as plotting and creating graphs

pip installl pandas

In [90]:

```
1 import pandas as pd
```

In [91]:

```
1 pd.__version__
```

Out[91]:

```
'1.2.4'
```

In [ ]:

```
1
```

Series

A Series is a One-Dimensional array like object, containing sequence of values (of similar Type to Numpy) and an associated array of data lables called index

In [92]:

```
1 import pandas as pd
```

In [93]:

```
1 obj = pd.Series([4,7,-5,3])
```

In [94]:

```
1 print(obj)
```

```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [ ]:

```
1
```

In [95]:

```
obj2 = pd.Series([4,7,-5,3],index=['d','b','a','c'])
```

In [96]:

```
print(obj2)
```

```
d    4
b    7
a   -5
c    3
dtype: int64
```

In [ ]:

```

```

Fetching values on the basis of indexing

In [97]:

```
print(obj2)
```

```
d    4
b    7
a   -5
c    3
dtype: int64
```

In [98]:

```
obj2['b']
```

Out[98]:

```
7
```

In [99]:

```
obj2[1]
```

Out[99]:

```
7
```

In [100]:

```
obj2.index   #getting the name of all the indexes
```

Out[100]:

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

In [101]:

```
obj2[['d','a','c']] #access multiple values via indexing
```

Out[101]:

```
d    4
a   -5
c    3
dtype: int64
```

In [102]:

```
obj2['b']
```

Out[102]:

```
7
```

In [103]:

```
obj2['b'] = 40  #changing the value using index
```

In [104]:

```
obj2
```

Out[104]:

```
d     4
b    40
a    -5
c     3
dtype: int64
```

In [105]:

```
obj2
```

Out[105]:

```
d     4
b    40
a    -5
c     3
dtype: int64
```

In [106]:

```
obj2 > 0  #boolean as an output use this
```

Out[106]:

```
d     True
b     True
a    False
c     True
dtype: bool
```

In [107]:

```
obj2[obj2 > 0]  # values as an output use the following code/
```

Out[107]:

```
d     4
b    40
c     3
dtype: int64
```

In [ ]:

```

```

In [108]:

```
obj2
```

Out[108]:

```
d     4
b    40
a    -5
c     3
dtype: int64
```

In [109]:

```
obj2 * 2
```

Out[109]:

```
d     8
b    80
a   -10
c     6
dtype: int64
```

In [110]:

```
obj2
```

Out[110]:

```
d     4
b    40
a    -5
c     3
dtype: int64
```

In [ ]:

```

```

In [111]:

```
obj2
```

Out[111]:

```
d     4
b    40
a    -5
c     3
dtype: int64
```

In [112]:

```
import numpy as np
```

In [113]:

```
np.exp(obj2)
```

Out[113]:

```
d    5.459815e+01
b    2.353853e+17
a    6.737947e-03
c    2.008554e+01
dtype: float64
```

In [114]:

```
np.sin(obj2)
```

Out[114]:

```
d   -0.756802
b    0.745113
a    0.958924
c    0.141120
dtype: float64
```

creating series from python diictionary

In [115]:

```
sdata = {'Mumbai':35000,'Pune':71000,'Nagpur':16000,'Chennai':5000}
```

In [116]:

```
sdata
```

Out[116]:

```
{'Mumbai': 35000, 'Pune': 71000, 'Nagpur': 16000, 'Chennai': 5000}
```

In [117]:

```
obj3 = pd.Series(sdata)
```

In [118]:

```
1 obj3
```

Out[118]:

```
Mumbai     35000
Pune       71000
Nagpur     16000
Chennai     5000
dtype: int64
```

In [ ]:

```
1
```

In [119]:

```
1 hobbies = ('cricket','cooking','swimming','travelling')
```

In [120]:

```
1 pd.Series(hobbies,index=['One','Two','Three','Four'])
```

Out[120]:

```
One          cricket
Two          cooking
Three      swimming
Four     travelling
dtype: object
```

In [ ]:

```
1
```

In [121]:

```
1 mydict = {'one':['val','val1']}
```

In [122]:

```
1 mydict
```

Out[122]:

```
{'one': ['val', 'val1']}
```

In [123]:

```
1 demo = pd.Series(mydict)
```

In [124]:

```
1 demo
```

Out[124]:

```
one    [val, val1]
dtype: object
```

In [125]:

```
1 demo['one'][1]
```

Out[125]:

```
'val1'
```

In [126]:

```
1 obj3
```

Out[126]:

```
Mumbai     35000
Pune       71000
Nagpur     16000
Chennai     5000
dtype: int64
```

In [127]:

```
1 sdata
```

Out[127]:

```
{'Mumbai': 35000, 'Pune': 71000, 'Nagpur': 16000, 'Chennai': 5000}
```

In [128]:

```
1 cities = ['Mumbai','Pune','Nagpur','Chennai','Delhi']
```

In [129]:

```
1 obj4 = pd.Series(sdata,index=cities)
```

In [130]:

```
1 obj4
```

Out[130]:

```
Mumbai     35000.0
Pune       71000.0
Nagpur     16000.0
Chennai     5000.0
Delhi          NaN
dtype: float64
```

identifying not null

In [131]:

```
1 pd.isnull(obj4)
```

Out[131]:

```
Mumbai     False
Pune       False
Nagpur     False
Chennai    False
Delhi       True
dtype: bool
```

In [132]:

```
1 pd.notnull(obj4)
```

Out[132]:

```
Mumbai      True
Pune        True
Nagpur      True
Chennai     True
Delhi      False
dtype: bool
```

In [133]:

```
1 obj3
```

Out[133]:

```
Mumbai     35000
Pune       71000
Nagpur     16000
Chennai     5000
dtype: int64
```

In [134]:

```
1 obj4
```

Out[134]:

```
Mumbai     35000.0
Pune       71000.0
Nagpur     16000.0
Chennai     5000.0
Delhi          NaN
dtype: float64
```

In [135]:

```
obj3 + obj4
```

Out[135]:

```
Chennai     10000.0
Delhi           NaN
Mumbai      70000.0
Nagpur      32000.0
Pune       142000.0
dtype: float64
```

In [ ]:

```

```

Data Frame

A DataFrame represents a rectangular table of data and contains ordered collection of columns each of which can be a different data type(numeric,boolean etc)

The DataFrame has both row and column index, it can be thought of as a dict of Series all sharing the same index. The data is stored as one or more 2-D block rather than a list,dict or some other collection of 1-D array

In [ ]:

```

```

In [136]:

```
data = {
    'State' : ['Mumbai','Pune','Chennai','Nagpur','Delhi'],
    'year':[2001,2011,2018,2019,2020],
    'pop':[1.5,1.7,3.6,2.4,3.2]
}
```

In [137]:

```
data
```

Out[137]:

```
{'State': ['Mumbai', 'Pune', 'Chennai', 'Nagpur', 'Delhi'],
 'year': [2001, 2011, 2018, 2019, 2020],
 'pop': [1.5, 1.7, 3.6, 2.4, 3.2]}
```

In [138]:

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

In [139]:

```
df
```

Out[139]:

	State	year	pop
0	Mumbai	2001	1.5
1	Pune	2011	1.7
2	Chennai	2018	3.6
3	Nagpur	2019	2.4
4	Delhi	2020	3.2

In [140]:

```
df.head(3)
```

Out[140]:

	State	year	pop
0	Mumbai	2001	1.5
1	Pune	2011	1.7
2	Chennai	2018	3.6

In [141]:

```
df.tail(2)
```

Out[141]:

	State	year	pop
3	Nagpur	2019	2.4
4	Delhi	2020	3.2

In [ ]:

```

```

In [ ]:

```

```

In [142]:

```
mydict = {
    'state':['Mumbai','Mumbai','Mumbai','Pune','Pune','Pune'],
    'year':[2010,2011,2012,2011,2012,2013],
    'pop':[1.5,1.7,3.6,2.4,2.9,3.2]
}
```

In [143]:

```
df2 = pd.DataFrame(mydict)
```

In [144]:

```
df2
```

Out[144]:

	state	year	pop
0	Mumbai	2010	1.5
1	Mumbai	2011	1.7
2	Mumbai	2012	3.6
3	Pune	2011	2.4
4	Pune	2012	2.9
5	Pune	2013	3.2

In [145]:

```
pd.DataFrame(mydict,columns=['year','state','pop'])
```

Out[145]:

	year	state	pop
0	2010	Mumbai	1.5
1	2011	Mumbai	1.7
2	2012	Mumbai	3.6
3	2011	Pune	2.4
4	2012	Pune	2.9
5	2013	Pune	3.2

In [146]:

```
# defining your own index names

df3 = pd.DataFrame(mydict,columns=['year','state','pop','debt'],index=['one','t
```

In [147]:

```
1 df3
```

Out[147]:

	year	state	pop	debt
one	2010	Mumbai	1.5	NaN
two	2011	Mumbai	1.7	NaN
three	2012	Mumbai	3.6	NaN
four	2011	Pune	2.4	NaN
five	2012	Pune	2.9	NaN
six	2013	Pune	3.2	NaN

In [148]:

```
1 df3.columns  #names of the columns
```

Out[148]:

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

In [149]:

```
1 df3['state']
```

Out[149]:

```
one      Mumbai
two      Mumbai
three    Mumbai
four       Pune
five       Pune
six        Pune
Name: state, dtype: object
```

In [150]:

```
1 df3
```

Out[150]:

	year	state	pop	debt
one	2010	Mumbai	1.5	NaN
two	2011	Mumbai	1.7	NaN
three	2012	Mumbai	3.6	NaN
four	2011	Pune	2.4	NaN
five	2012	Pune	2.9	NaN
six	2013	Pune	3.2	NaN

In [151]:

```
df3.loc['four'] #fetch on basis of rows
```

Out[151]:

```
year     2011
state    Pune
pop       2.4
debt      NaN
Name: four, dtype: object
```

In [152]:

```
df3.loc['one']
```

Out[152]:

```
year       2010
state    Mumbai
pop         1.5
debt        NaN
Name: one, dtype: object
```

In [ ]:

```

```

In [153]:

```
df3['debt'] = 16.5
```

In [154]:

```
df3
```

Out[154]:

	year	state	pop	debt
one	2010	Mumbai	1.5	16.5
two	2011	Mumbai	1.7	16.5
three	2012	Mumbai	3.6	16.5
four	2011	Pune	2.4	16.5
five	2012	Pune	2.9	16.5
six	2013	Pune	3.2	16.5

In [155]:

```
df3.loc['one']
```

Out[155]:

```
year       2010
state    Mumbai
pop         1.5
debt       16.5
Name: one, dtype: object
```

In [156]:

```
df3
```

Out[156]:

	year	state	pop	debt
one	2010	Mumbai	1.5	16.5
two	2011	Mumbai	1.7	16.5
three	2012	Mumbai	3.6	16.5
four	2011	Pune	2.4	16.5
five	2012	Pune	2.9	16.5
six	2013	Pune	3.2	16.5

In [ ]:

```

```

In [157]:

```
df3.iloc[0] #can fetch on the basis of default index position
```

Out[157]:

```
year       2010
state    Mumbai
pop         1.5
debt       16.5
Name: one, dtype: object
```

In [158]:

```
df3['debt'] = np.arange(1,7)
```

In [159]:

```
df3
```

Out[159]:

	year	state	pop	debt
one	2010	Mumbai	1.5	1
two	2011	Mumbai	1.7	2
three	2012	Mumbai	3.6	3
four	2011	Pune	2.4	4
five	2012	Pune	2.9	5
six	2013	Pune	3.2	6

In [ ]:

```

```

In [160]:

```
val = pd.Series([-1.2,-1.5,-1.7],index=['two','four','five'])
```

In [161]:

```
val
```

Out[161]:

```
two    -1.2
four   -1.5
five   -1.7
dtype: float64
```

In [162]:

```
df3['debt'] = val
```

In [163]:

```
df3
```

Out[163]:

	year	state	pop	debt
one	2010	Mumbai	1.5	NaN
two	2011	Mumbai	1.7	-1.2
three	2012	Mumbai	3.6	NaN
four	2011	Pune	2.4	-1.5
five	2012	Pune	2.9	-1.7
six	2013	Pune	3.2	NaN

In [164]:

```
df3
```

Out[164]:

	year	state	pop	debt
one	2010	Mumbai	1.5	NaN
two	2011	Mumbai	1.7	-1.2
three	2012	Mumbai	3.6	NaN
four	2011	Pune	2.4	-1.5
five	2012	Pune	2.9	-1.7
six	2013	Pune	3.2	NaN

In [165]:

```
# How to Assign Boolean Value to a Column
```

In [ ]:

```
1
```

In [166]:

```
1 df3['western'] = df3['state'] == 'Mumbai'
```

In [167]:

```
1 df3
```

Out[167]:

	year	state	pop	debt	western
one	2010	Mumbai	1.5	NaN	True
two	2011	Mumbai	1.7	-1.2	True
three	2012	Mumbai	3.6	NaN	True
four	2011	Pune	2.4	-1.5	False
five	2012	Pune	2.9	-1.7	False
six	2013	Pune	3.2	NaN	False

In [ ]:

```
1
```

To remove a column From DataFrame

In [168]:

```
1 del df3['western']
```

In [169]:

```
1 df3
```

Out[169]:

	year	state	pop	debt
one	2010	Mumbai	1.5	NaN
two	2011	Mumbai	1.7	-1.2
three	2012	Mumbai	3.6	NaN
four	2011	Pune	2.4	-1.5
five	2012	Pune	2.9	-1.7
six	2013	Pune	3.2	NaN

In [ ]:

```
1
```

Index Objects

In [170]:

```
obj = pd.Series(range(3),index=['a','b','c'])
```

In [171]:

```
print(obj)
```

```
a    0
b    1
c    2
dtype: int64
```

In [172]:

```
obj.index   # Shows you all the indexes of the Series
```

Out[172]:

```
Index(['a', 'b', 'c'], dtype='object')
```

In [173]:

```
my_index = obj.index
```

In [174]:

```
my_index
```

Out[174]:

```
Index(['a', 'b', 'c'], dtype='object')
```

In [175]:

```
my_index[0]
```

Out[175]:

```
'a'
```

In [176]:

```
my_index[1]
```

Out[176]:

```
'b'
```

In [177]:

```
my_index[2]
```

Out[177]:

```
'c'
```

In [178]:

```
1 my_index[0] = 'B'
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-178-d930d799002a> in <module>
----> 1 my_index[0] = 'B'

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in
__setitem__(self, key, value)
   4275     @final
   4276     def __setitem__(self, key, value):
-> 4277         raise TypeError("Index does not support mutable operat
ions")
   4278
   4279     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

In [ ]:

```
1
```

Re- Indexing

In [179]:

```
1 obj = pd.Series([4.1,7.5,-8.3,3.6],index=['d','b','a','c'])
```

In [180]:

```
1 obj
```

Out[180]:

```
d    4.1
b    7.5
a   -8.3
c    3.6
dtype: float64
```

In [181]:

```
1 obj2 = obj.reindex(['a','b','c','d','e'])
```

In [182]:

```
1 obj2
```

Out[182]:

```
a   -8.3
b    7.5
c    3.6
d    4.1
e    NaN
dtype: float64
```

In [ ]:

```
1
```

In [183]:

```
1 obj3 = pd.Series(['blue','purple','yellow'],index=[0,2,4])
```

In [184]:

```
1 obj3
```

Out[184]:

```
0      blue
2    purple
4    yellow
dtype: object
```

```
1 ffill => forward filling
```

In [ ]:

```
1
```

In [185]:

```
1 obj4 = obj3.reindex(range(6),method='ffill')
```

In [186]:

```
1 obj4
```

Out[186]:

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

In [ ]:

```
1
```

Droping Entries from An Axis

In [ ]:

```
1
```

In [187]:

```
1 obj = pd.Series(np.arange(5),index=['a','b','c','d','e'])
```

In [188]:

```
1 obj
```

Out[188]:

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

In [189]:

```
1 new = obj.drop('c')
```

In [190]:

```
1 new
```

Out[190]:

```
a    0
b    1
d    3
e    4
dtype: int64
```

In [ ]:

```
1
```

In [ ]:

```
1 new = obj.drop(['b','e'])   #dropping more than one row at a time
```

In [191]:

```
1 new
```

Out[191]:

```
a    0
b    1
d    3
e    4
dtype: int64
```

In [ ]:

```
1
```

In [192]:

```
1 data = pd.DataFrame(np.arange(16).reshape((4,4)),
2         index= ['Mumbai','Pune','Chennai','Delhi'],
3         columns=['one','two','three','four'])
```

In [193]:

```
1 data
```

Out[193]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11
Delhi	12	13	14	15

In [194]:

```
1 data.drop(['Chennai','Delhi']) #throwing a copy of dropped rows
```

Out[194]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7

In [195]:

```
1 data
```

Out[195]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11
Delhi	12	13	14	15

To remove a column using drop

In [196]:

```
1 data.drop('three',axis=1)
```

Out[196]:

	one	two	four
Mumbai	0	1	3
Pune	4	5	7
Chennai	8	9	11
Delhi	12	13	15

In [197]:

```
1 data
```

Out[197]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11
Delhi	12	13	14	15

In [ ]:

```
1
```

To remove multiple column using drop

In [198]:

```
1 data.drop(['one','four'],axis=1)
```

Out[198]:

	two	three
Mumbai	1	2
Pune	5	6
Chennai	9	10
Delhi	13	14

In [199]:

```
1 data
```

Out[199]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11
Delhi	12	13	14	15

In [ ]:

```
1
```

In [200]:

```
1 data.drop('two',axis=1,inplace=True)
```

In [201]:

```
1 data
```

Out[201]:

	one	three	four
Mumbai	0	2	3
Pune	4	6	7
Chennai	8	10	11
Delhi	12	14	15

In [202]:

```
1 data.drop('Delhi',inplace=True)
```

In [203]:

```
1 data
```

Out[203]:

	one	three	four
Mumbai	0	2	3
Pune	4	6	7
Chennai	8	10	11

In [ ]:

```
1
```

indexing,selection & filtering

In [204]:

```
1 obj = pd.Series(np.arange(4),index=['a','b','c','d'])
```

In [205]:

```
1 obj
```

Out[205]:

```
a    0
b    1
c    2
d    3
dtype: int64
```

In [ ]:

```
1
```

In [206]:

```
1 obj[1:3]
```

Out[206]:

```
b    1
c    2
dtype: int64
```

In [207]:

```
1 obj
```

Out[207]:

```
a    0
b    1
c    2
d    3
dtype: int64
```

In [208]:

```
1 obj<2
```

Out[208]:

```
a     True
b     True
c    False
d    False
dtype: bool
```

In [209]:

```
1 obj[obj<2]
```

Out[209]:

```
a    0
b    1
dtype: int64
```

In [ ]:

```
1
```

In [210]:

```
1 obj
```

Out[210]:

```
a    0
b    1
c    2
d    3
dtype: int64
```

In [211]:

```
1 obj['b':'c'] = 10
```

In [212]:

```
1 obj
```

Out[212]:

```
a     0
b    10
c    10
d     3
dtype: int64
```

In [ ]:

```
1
```

In [213]:

```
1 data
```

Out[213]:

	one	three	four
Mumbai	0	2	3
Pune	4	6	7
Chennai	8	10	11

In [214]:

```
1 data['one'] #fetching record for one col
```

Out[214]:

```
Mumbai     0
Pune       4
Chennai    8
Name: one, dtype: int64
```

In [215]:

```
1 data[['one','four']]  #fetching more than one column record
```

Out[215]:

	one	four
Mumbai	0	3
Pune	4	7
Chennai	8	11

In [ ]:

```
1
```

Arithmetic operation with series

In [216]:

```
1 series1 = pd.Series([7.3,-2.5,3.4,1.5],index=['a','c','d','e'])
```

In [217]:

```
1 series1
```

Out[217]:

```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

In [ ]:

```
1
```

In [218]:

```
1 series2 = pd.Series([-2.1,3.6,-1.5,4,3.1],index=['a','c','e','f','g'])
```

In [219]:

```
1 series2
```

Out[219]:

```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

In [220]:

```
1 series1 + series2
```

Out[220]:

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

In [ ]:

```
1
```

In [221]:

```
1 df1 = pd.DataFrame(np.arange(9).reshape((3,3)),
2                    columns=['b','c','d'],
3                   index=['Mumbai','Pune','Nagpur'])
```

In [222]:

```
1 df1
```

Out[222]:

	b	c	d
Mumbai	0	1	2
Pune	3	4	5
Nagpur	6	7	8

In [223]:

```
1 df2 = pd.DataFrame(np.arange(12).reshape((4,3)),
2                   columns=['b','d','e'],
3                   index=['Delhi','Mumbai','Pune','Chennai'])
```

In [224]:

```
1 df2
```

Out[224]:

	b	d	e
Delhi	0	1	2
Mumbai	3	4	5
Pune	6	7	8
Chennai	9	10	11

In [226]:

```
1 df3 = df1 + df2
```

In [230]:

```
1 df3
```

Out[230]:

	b	c	d	e
Chennai	NaN	NaN	NaN	NaN
Delhi	NaN	NaN	NaN	NaN
Mumbai	3.0	NaN	6.0	NaN
Nagpur	NaN	NaN	NaN	NaN
Pune	9.0	NaN	12.0	NaN

In [231]:

```
1 df1
```

Out[231]:

	b	c	d
Mumbai	0	1	2
Pune	3	4	5
Nagpur	6	7	8

In [232]:

```
1 df2
```

Out[232]:

	b	d	e
Delhi	0	1	2
Mumbai	3	4	5
Pune	6	7	8
Chennai	9	10	11

In [ ]:

```
1
```

In [229]:

```
1 df1.add(df2,fill_value=0)
```

Out[229]:

	b	c	d	e
Chennai	9.0	NaN	10.0	11.0
Delhi	0.0	NaN	1.0	2.0
Mumbai	3.0	1.0	6.0	5.0
Nagpur	6.0	7.0	8.0	NaN
Pune	9.0	4.0	12.0	8.0

In [ ]:

```
1
```

In [ ]:

```
1
```

Function Application & Mapping

In [233]:

```
1 df1 = pd.DataFrame(np.random.randn(4,3),columns=['b','d','e'],index=['Mumbai','
```

In [234]:

```
1 df1
```

Out[234]:

	b	d	e
Mumbai	-0.878239	-0.604510	0.935718
Pune	-0.986209	-0.006721	0.534438
Nagpur	1.120403	-0.969248	0.142273
Thane	0.969137	0.718205	-1.075252

In [ ]:

```
1
```

In [237]:

```
1 # np.abs(df1)
```

In [236]:

```
1 df1
```

Out[236]:

	b	d	e
Mumbai	-0.878239	-0.604510	0.935718
Pune	-0.986209	-0.006721	0.534438
Nagpur	1.120403	-0.969248	0.142273
Thane	0.969137	0.718205	-1.075252

In [239]:

```
1 diff = lambda x: x.max() - x.min()  #customise function
```

In [240]:

```
1 df1.apply(diff)
```

Out[240]:

```
b    2.106611
d    1.687453
e    2.010970
dtype: float64
```

In [ ]:

```
1
```

In [242]:

```
1
```

Out[242]:

```
-0.9862085200297697
```

In [244]:

```
1 df1['b'].max() - df1['b'].min()
```

Out[244]:

```
2.106611274016847
```

In [ ]:

```
1
```

In [245]:

```
1 df1.apply(diff,axis='columns')   #doing operation on the basis of rows
```

Out[245]:

```
Mumbai    1.813957
Pune      1.520647
Nagpur    2.089650
Thane     2.044388
dtype: float64
```

In [261]:

```
1 df1
```

Out[261]:

	b	d	e
Mumbai	-0.878239	-0.604510	0.935718
Pune	-0.986209	-0.006721	0.534438
Nagpur	1.120403	-0.969248	0.142273
Thane	0.969137	0.718205	-1.075252

In [259]:

```
1 def my_diff(x):
2     return pd.Series([x.max(),x.min()],index=['max','min'])
```

In [262]:

```
1 df1.apply(my_diff)
```

Out[262]:

	b	d	e
max	1.120403	0.718205	0.935718
min	-0.986209	-0.969248	-1.075252

In [264]:

```
1 df1.loc['Mumbai'].max()
```

Out[264]:

```
0.9357181181719099
```

In [ ]:

```
1
```

In [265]:

```
1 df1
```

Out[265]:

	b	d	e
Mumbai	-0.878239	-0.604510	0.935718
Pune	-0.986209	-0.006721	0.534438
Nagpur	1.120403	-0.969248	0.142273
Thane	0.969137	0.718205	-1.075252

In [280]:

```
1 df1['e']
```

Out[280]:

```
Mumbai    0.935718
Pune      0.534438
Nagpur    0.142273
Thane    -1.075252
Name: e, dtype: float64
```

In [278]:

```
1 format_decimal = lambda x: '%.2f'  % x
```

In [290]:

```
df1['b'].apply(format_decimal)
```

Out[290]:

```
Mumbai    -0.88
Pune      -0.99
Nagpur     1.12
Thane      0.97
Name: b, dtype: object
```

In [ ]:

```

```

In [ ]:

```

```

sorting & ranking

In [292]:

```
obj = pd.Series(range(4),index=['d','a','b','c'])
```

In [293]:

```
obj
```

Out[293]:

```
d    0
a    1
b    2
c    3
dtype: int64
```

In [294]:

```
obj.sort_index()
```

Out[294]:

```
a    1
b    2
c    3
d    0
dtype: int64
```

In [ ]:

```

```

In [295]:

```
frame = pd.DataFrame(np.arange(8).reshape((2,4)),
                     index=['three','one'],
                     columns=['d','a','b','c'])
```

In [296]:

```
frame
```

Out[296]:

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

In [297]:

```
frame.sort_index()
```

Out[297]:

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

In [298]:

```
frame.sort_index(axis=1)
```

Out[298]:

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

In [299]:

```
frame.sort_index(axis=1,ascending=False)
```

Out[299]:

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

In [ ]:

```

```

In [300]:

```
ser = pd.Series(range(5),index=['a','a','b','b','c'])
```

In [301]:

```
1 ser
```

Out[301]:

```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

In [302]:

```
1 ser['a']
```

Out[302]:

```
a    0
a    1
dtype: int64
```

In [304]:

```
1 ser.index.is_unique
```

Out[304]:

```
False
```

In [ ]:

```
1
```

computing descriptive statistics

In [305]:

```
1 df = pd.DataFrame([[1.4,np.nan],[7.1,4.5],
2                    [np.nan,np.nan],[0.75,-1.3]],
3                    index=['a','b','c','d'],columns=['A','B'])
```

In [306]:

```
1 df
```

Out[306]:

	A	B
a	1.40	NaN
b	7.10	4.5
c	NaN	NaN
d	0.75	-1.3

In [307]:

```
df.sum() #sum of all numbers on basis of column
```

Out[307]:

```
A    9.25
B    3.20
dtype: float64
```

In [308]:

```
df.sum(axis=1) #on basis of rows / use axis=column
```

Out[308]:

```
a     1.40
b    11.60
c     0.00
d    -0.55
dtype: float64
```

In [310]:

```
df
```

Out[310]:

	A	B
a	1.40	NaN
b	7.10	4.5
c	NaN	NaN
d	0.75	-1.3

In [309]:

```
df.mean()
```

Out[309]:

```
A    3.083333
B    1.600000
dtype: float64
```

In [311]:

```
df.mean(axis=1)
```

Out[311]:

```
a    1.400
b    5.800
c      NaN
d   -0.275
dtype: float64
```

In [317]:

```
1 df.mean(axis=1,skipna=False)
```

Out[317]:

```
a      NaN
b    5.800
c      NaN
d   -0.275
dtype: float64
```

In [321]:

```
1 df
```

Out[321]:

	A	B
a	1.40	NaN
b	7.10	4.5
c	NaN	NaN
d	0.75	-1.3

In [318]:

```
1 df.describe()
```

Out[318]:

	A	B
count	3.000000	2.000000
mean	3.083333	1.600000
std	3.493685	4.101219
min	0.750000	-1.300000
25%	1.075000	0.150000
50%	1.400000	1.600000
75%	4.250000	3.050000
max	7.100000	4.500000

In [322]:

```
1 df.describe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
 last)
<ipython-input-322-16ee2051e23e> in <module>
----> 1 df.describe(axis=1)

TypeError: describe() got an unexpected keyword argument 'axis'
```

In [323]:

```
1 df.std()
```

Out[323]:

```
A    3.493685
B    4.101219
dtype: float64
```

In [325]:

```
1 df.std(axis=1)
```

Out[325]:

```
a         NaN
b    1.838478
c         NaN
d    1.449569
dtype: float64
```

In [326]:

```
1 df.var()
```

Out[326]:

```
A    12.205833
B    16.820000
dtype: float64
```

In [327]:

```
1 df.mode()
```

Out[327]:

	A	B
0	0.75	-1.3
1	1.40	4.5
2	7.10	NaN

In [329]:

```
1 df
```

Out[329]:

	A	B
a	1.40	NaN
b	7.10	4.5
c	NaN	NaN
d	0.75	-1.3

In [328]:

```
1 df.count()
```

Out[328]:

```
A    3
B    2
dtype: int64
```

In [ ]:

```
1
```

In [334]:

```
1 # import pandas_datareader.data as web
2 # from datetime import datetime
3 # all_data_info = {
4 #     ticker : web.get_data_yahoo(ticker,start=datetime(2017, 8, 13), end=datet
5 #     for ticker in ['APPL','IBM','GOOG','MSFT']
6 # }
```

In [ ]:

```
1
```

In [335]:

```
1 obj = pd.Series(['c','a','d','a','a','b','b','c','c'])
```

In [336]:

```
1 obj
```

Out[336]:

```
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object
```

In [337]:

```
1 obj.unique()
```

Out[337]:

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

In [338]:

```
1 obj.value_counts()
```

Out[338]:

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

In [ ]:

```
1
```

In [339]:

```
1 obj
```

Out[339]:

```
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object
```

In [340]:

```
1 obj.isin(['c','a'])
```

Out[340]:

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

In [351]:

```
1 cities = {"name":["mumbai","pune","chennai","delhi"],
2           "state":['Maharashtra','Karnataka','Gujrat','Tamil Nadu']}
```

In [354]:

```
1 df = pd.DataFrame(cities,columns=['name','state'])
```

In [355]:

```
1 df
```

Out[355]:

	name	state
0	mumbai	Maharashtra
1	pune	Karnataka
2	chennai	Gujrat
3	delhi	Tamil Nadu

In [356]:

```
1 df[['state','name']]
```

Out[356]:

	state	name
0	Maharashtra	mumbai
1	Karnataka	pune
2	Gujrat	chennai
3	Tamil Nadu	delhi

In [350]:

```
1 df['city'].apply(lambda data : data.upper())
```

Out[350]:

```
0     MUMBAI
1       PUNE
2     NAGPUR
3    CHENNAI
4      DELHI
Name: city, dtype: object
```

In [ ]:

```
1
```

In [1]:

```
1
```

Data Loading and Writing Data

In [3]:

```
1 import pandas as pd
2 import numpy as np
```

In [4]:

```
df = pd.read_csv('ex1.csv')
```

In [8]:

```
df
```

Out[8]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo
3	13	14	15	16	test

In [6]:

```
pd.read_csv('ex1.csv',header=None) # if you dont want col header
```

Out[6]:

	0	1	2	3	4
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo
4	13	14	15	16	test

In [7]:

```
#if you want to put your own col header

pd.read_csv('ex1.csv',names=['ONE','TWO','THREE','FOUR','MESSAGE'])
```

Out[7]:

	ONE	TWO	THREE	FOUR	MESSAGE
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo
4	13	14	15	16	test

In [ ]:

```

```

In [ ]:

```
1
```

In [9]:

```
1 df
```

Out[9]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo
3	13	14	15	16	test

In [11]:

```
1 # to skip rows
2
3 pd.read_csv('ex1.csv',skiprows=[3])
```

Out[11]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	13	14	15	16	test

In [ ]:

```
1
```

In [ ]:

```
1
```

handling missing

In [14]:

```
1 res = pd.read_csv('ex2.csv')
```

In [15]:

```
1 res
```

Out[15]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [16]:

```
1 pd.isnull(res)
```

Out[16]:

	something	a	b	c	d	message
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

In [ ]:

```
1
```

In [17]:

```
1 res1 = pd.read_csv('ex2.csv',na_values=['NULL'])
```

In [18]:

```
1 res1
```

Out[18]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [19]:

```
1 movies = pd.read_csv('movies.csv')
```

In [21]:

```
1 movies
```

Out[21]:

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...
...	...	...	...	...	...	...	...	...
995	995	398. Scent of a Woman (1992)	R	156 min	Drama	8.0	NaN	A prep school student needing money agrees to ...
996	996	399. Aladdin (1992)	G	90 min	Animation, Adventure, Comedy	8.0	86.0	A kindhearted street urchin and a power-hungry...
997	997	400. JFK (1991)	R	189 min	Drama, History, Thriller	8.0	72.0	New Orleans District Attorney Jim Garrison dis...
998	998	301. Nights of Cabiria (1957)	Not Rated	110 min	Drama	8.1	NaN	A waifish prostitute wanders the streets of Ro...

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description
999	999	302. Throne of Blood (1957)	Not Rated	110 min	Drama, History	8.1	NaN	A war-hardened general, egged on by his ambiti...

1000 rows × 10 columns

In [22]:

```
1 movies.isnull()
```

Out[22]:

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	True	False	False	False
996	False	False	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	True	False	False	False
999	False	False	False	False	False	False	True	False	False	False

1000 rows × 10 columns

In [25]:

```
1 pd.set_option('display.max_rows',None) #this is to remove the limit on display
```

In [26]:

```
1 movies
```

Out[26]:

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cas
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Directo Frar Darabont Stars: Ti Robbins,.
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Directo Francis For Coppola Star Marlon.
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Directo Christophe Nolan | Star Christian.

In [ ]:

```
1
```

In [27]:

```
1 pd.read_csv('movies.csv',nrows=50)
```

Out[27]:

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	V 2,29 | C $28
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	V 1,58 | C $134
2	2	3. The Dark Knight	PG-13	152 min	Action, Crime	9.0	84.0	When the menace known as the	Director: Christopher Nolan |	V 2,26

In [29]:

```
1 movies.columns
```

Out[29]:

```
Index(['Unnamed: 0', 'Title', 'Certificate', 'Duration', 'Genre', 'Rat
e',
       'Metascore', 'Description', 'Cast', 'Info'],
      dtype='object')
```

In [34]:

```
1 movies['Rate']
```

```
8       8.8
9       8.8
10      8.8
11      8.8
12      8.8
13      8.7
14      8.7
15      8.7
16      8.7
17      8.7
18      8.7
19      8.6
20      8.6
21      8.6
22      8.6
23      8.6
24      8.6
25      8.6
26      8.6
27      8.6
```

In [36]:

```
1 type(movies['Rate'])
```

Out[36]:

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

In [38]:

```
1 movies['Rate'].value_counts()
```

Out[38]:

```
8.0    560
8.1    249
8.2     63
8.3     46
8.4     29
8.5     21
8.6     13
8.7      6
8.8      5
8.9      4
9.0      2
9.2      1
9.3      1
Name: Rate, dtype: int64
```

In [39]:

```
1 movies['Rate'].max()
```

Out[39]:

```
9.3
```

In [40]:

```
1 movies['Rate'].min()
```

Out[40]:

```
8.0
```

In [42]:

```
1 movies['Certificate'].value_counts()
```

Out[42]:

```
R            345
Not Rated    200
PG-13        165
PG           105
Passed        55
Approved      45
G             41
TV-MA          7
TV-PG          7
GP             2
TV-14          1
Name: Certificate, dtype: int64
```

In [43]:

```
1 res1
```

Out[43]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [44]:

```
1 cleaned = res1.dropna()
```

In [45]:

```
1 cleaned
```

Out[45]:

	something	a	b	c	d	message
2	three	9	10	11.0	12	foo

In [ ]:

```
1 np.nan
```

In [47]:

```
1 data = pd.DataFrame([
2     [1,6.5,3],
3     [1,np.nan,np.nan],
4     [np.nan,np.nan,np.nan],[np.nan,6.5,3]
5 ])
```

In [48]:

```
1 data
```

Out[48]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [49]:

```
1 data.dropna()
```

Out[49]:

	0	1	2
0	1.0	6.5	3.0

In [50]:

```
1 data.dropna(how='all')
```

Out[50]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

In [51]:

```
1 data.dropna(axis=1)
```

Out[51]:

0
1
2
3

In [52]:

```
1 data.dropna(axis=1,how='all')
```

Out[52]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [53]:

```
1 data
```

Out[53]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [56]:

```
1 data.fillna(0)
```

Out[56]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	6.5	3.0

In [57]:

```
1 data
```

Out[57]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [55]:

```
1 data.fillna({0:0,1:5,2:10})
```

Out[55]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	5.0	10.0
2	0.0	5.0	10.0
3	0.0	6.5	3.0

In [58]:

```
1 data
```

Out[58]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [59]:

```
1 data.fillna({0:0,1:5,2:10},inplace=True)
```

In [60]:

```
1 data
```

Out[60]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	5.0	10.0
2	0.0	5.0	10.0
3	0.0	6.5	3.0

In [61]:

```
1 data = pd.Series([1,np.nan,3.5,np.nan,7])
```

In [62]:

```
1 data
```

Out[62]:

```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

In [63]:

```
1 data.fillna(data.mean())
```

Out[63]:

```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

In [64]:

```
1 data.mean()
```

Out[64]:

```
3.8333333333333335
```

In [ ]:

```
1
```