Thanks for your feedback on Lecture 17, basic intro of Pandas. I am Akshansh (+91 8384891269, akshanshofficial@gmail.com (mailto:akshanshofficial@gmail.com)) and I am here again to take you on a ride with basic Pandas. This is again, put this right under your pillow. Let's Get Started,

In [1]:

```
import numpy as np
import pandas as pd
```

Pandas is one of the most used library of Python and it takes advantage of Numpy. (here Numpy playing me and pandas being my Ex, taking advantage). Like my Ex, pandas is open source and interacted with many. It has become more common and being used on various feilds.

Pandas has two reasons to talk about (just like Lana Rose, from Russia has, now don't google her :-P) - Series and DataFrame.

SERIES

Data in a single line is called Series. It is just like when you first met your gf, simple and innocent, most importantly single too.

1) Creating Series

How you made her your gf? by passing a list of messages, gifts and fake promises(aha, don't hide now). This is how we create Series by passing list of inputs. List of input is called one dimentional array. Read Lecture 16 for more info, don't call me please ha ha ha

In [2]:

```
obj=pd.Series([4,7,-5,3])  #cerated a Series name obj
```

In [3]:

```
obj  #prints obj Series
```

Out[3]:

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

Simple? 0,1,2,3 is index numbers. Indexing by default starts from 0. You remember or you forgot?

2) knowing index and values

In [4]:

```
obj.index  #prints index
```

Out[4]:

```
RangeIndex(start=0, stop=4, step=1)
```

In [5]:

```
obj.values  #prints values
```

Out[5]:

```
array([ 4,  7, -5,  3])
```

3) changing index

I didn't like the index, like many thngs including Trump. I am gonna change it. AAh not Trump but index.

In [6]:

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

In [7]:

```
obj2
```

Out[7]:

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

see passing a new list named index separated by comma can do this for you. I wish it could be as easy as it to change Trump or Imran from Pakistan.

##AkshanshTips - Here is another way to so the same

In [8]:

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

In [9]:

```
obj2
```

Out[9]:

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

4) Taking values from Series

values in Series are linked with Index. You most probably take help from you gf's best friend (not in case she has a male bestfried, go kill him first other wise 'est' in bestfriend will cahnge to 'oy'). Index and values are bestfriends, they party together and stays together. If you want to take out gf, take her bestfriend first. If you want to take value, take index first.

In [10]:

```
obj2['a'] #takeout value of 'a' index in obj2
```

Out[10]:

```
4
```

In [11]:

```
obj2[['b','a']] #takeout multiple values by index
```

Out[11]:

```
b    7
a    4
dtype: int64
```

In [12]:

```
obj2>2 #checks what is greater than2 in obj2
```

Out[12]:

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

this is not what I wanted, did you? Let's do my way-

##AkshanshTips- pass bool value to Series to get values in []

In [13]:

```
obj2[obj2>2]
```

Out[13]:

```
a    4
b    7
d    3
dtype: int64
```

See my tips always saves you ass. Doesn't it?

5) Basic operations on Series

In [14]:

```
obj2*2  #multiply wit 2
```

Out[14]:

```
a     8
b    14
c   -10
d     6
dtype: int64
```

In [15]:

```
obj2/2  #obj2 Sereis divided by 2
```

Out[15]:

```
a    2.0
b    3.5
c   -2.5
d    1.5
dtype: float64
```

In [16]:

```
obj2+1  #adding one to each value in obj2 Series
```

Out[16]:

```
a    5
b    8
c   -4
d    4
dtype: int64
```

Ilet's print obj2

In [17]:

```
obj2
```

Out[17]:

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

##AkshanshTips - basic operations don't change Sereis permanently. Originality is preferred - Good Ethics, well done Series

6) Taking advantage of Numpy mehods and functions

here is the best part, your ex's favorite - Taking advantage.

In [18]:

```
np.sqrt(obj2)  #takes sqare root of Series obj2
```

```
/home/akshansh/.local/lib/python3.6/site-packages/pandas/core/series.p
y:679: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[18]:

```
a    2.000000
b    2.645751
c         NaN
d    1.732051
dtype: float64
```

##AkshanshTips- What the fuck is NaN? My tip here is - Have some patience, you'll learn slowly. You ain't Iron Man :-P

In [19]:

```
np.exp(obj2)  #taking exponential, that is log base e
```

Out[19]:

```
a      54.598150
b    1096.633158
c       0.006738
d      20.085537
dtype: float64
```

##AkshanshTips - You know you can check if an index is present in Series or not, wanna see - See below -

In [20]:

```
'b' in obj2  #b is in index of obj2
```

Out[20]:

```
True
```

In [21]:

```
'e' in obj2  #e isn't in the index of obj2
```

Out[21]:

```
False
```

7) Making Series out of Dictionary

In [22]:

```
sdata= {'ohio' : 35000,'texas':71000,'oregon':16000,'utah':5000}
```

I've created a dictionary of state data of america. Like ohio sate has 35000 girls and texas has 71000 (next time visit Texas, high probability to get a gf there)

In [23]:

```
obj3=pd.Series(sdata)  #pass dict as input
```

In [24]:

```
obj3 #let's get this printed
```

Out[24]:

```
ohio      35000
texas     71000
oregon    16000
utah       5000
dtype: int64
```

keys of dict become index and values of dict become, values. That's wierd.

##AkshanshTips- You can override the order of the index just by changing order in passable index list

In [25]:

```
states=['california','texas','ohio','oregon']
```

In [26]:

```
obj4=pd.Series(sdata,index=states)
```

In [27]:

```
obj4 #let's see what we got
```

Out[27]:

```
california        NaN
texas         71000.0
ohio          35000.0
oregon        16000.0
dtype: float64
```

see idex order has been changed. I introduced california there, since california was not in sdata dictionary it automatically assigned NaN (Not a number) value to it. It means nothing is there for california index.

One more thing to notice, utah is not states, so it was not included in obj4

8) let's check where is NaN values in our Series?

In [28]:

```
obj4.isnull() #checks where is NaN
```

Out[28]:

```
california     True
texas         False
ohio          False
oregon        False
dtype: bool
```

In [29]:

```
obj4.notnull()  #checks where NaN is not present
```

Out[29]:

```
california    False
texas          True
ohio           True
oregon         True
dtype: bool
```

9) Playing with multiple Series.

You can play with multiple Series just like yuor Ex did with boys/girls

In [30]:

```
 obj3 #we created this earlier
```

Out[30]:

```
ohio      35000
texas     71000
oregon    16000
utah       5000
dtype: int64
```

In [31]:

```
obj4 #we created this earlier too
```

Out[31]:

```
california        NaN
texas         71000.0
ohio          35000.0
oregon        16000.0
dtype: float64
```

In [32]:

```
obj3+obj4 #don't read comment use common sese brother/girl (not going to say si
```

Out[32]:

```
california         NaN
ohio           70000.0
oregon         32000.0
texas         142000.0
utah               NaN
dtype: float64
```

In [33]:

```
obj3-obj4
```

Out[33]:

```
california    NaN
ohio          0.0
oregon        0.0
texas         0.0
utah          NaN
dtype: float64
```

you can do other things too. Do I need to tell you everything, did you ex did? did she? Noh!

10) naming your Series

Let's name your Series obj4 as "poluplation of girls"

In [34]:

```
obj4.name='population of girls'
```

In [35]:

```
obj4
```

Out[35]:

```
california        NaN
texas         71000.0
ohio          35000.0
oregon        16000.0
Name: population of girls, dtype: float64
```

you can name your Index as well

In [36]:

```
obj4.index.name='states'
```

In [37]:

```
obj4
```

Out[37]:

```
states
california        NaN
texas         71000.0
ohio          35000.0
oregon        16000.0
Name: population of girls, dtype: float64
```

DATAFRAME

This is another reason to love pandas. This is second datatype in Pandas. Let me explain what is DataFrame.

It is also like series, it too has index, unlike Sereis it doesn't has only one column. It has multiple columns. Series had one values in col but DataFrame can have stings (words) numrics boolean (true/false).

##AkshanshTips- Series is one dimentional array, DataFrame is multidimentional array. (Are BC, indian will get it)

1) How we create DataFrame

there are many ways to create a DataFrame, most common and easy way is creating from a dictionary of equal length lists or NumPy arrays

##AkshanshTips - Each key of dict must have equal elements in list that will be passed as value

In [38]:

```
data={'state': ['ohio','ohio','ohio','nevada','nevada','nevada'],
      'year': [2000,2001,2002,2001,2002,2003],
        'population':[1.5,1.7,3.6,2.4,2.9,3.2]}
```

In [39]:

```
data #see what we got in data dictionary
```

Out[39]:

```
{'state': ['ohio', 'ohio', 'ohio', 'nevada', 'nevada', 'nevada'],
 'year': [2000, 2001, 2002, 2001, 2002, 2003],
 'population': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
```

In [40]:

```
frame=pd.DataFrame(data)  #creates DataFrame from Dict data
```

In [41]:

```
1 frame #see what's there in frame
```

Out[41]:

	state	year	population
0	ohio	2000	1.5
1	ohio	2001	1.7
2	ohio	2002	3.6
3	nevada	2001	2.4
4	nevada	2002	2.9
5	nevada	2003	3.2

See DataFrame, isn't is beautiful? If you can't see beauty in this. That's why your ex left you :-P Either you are blind or just blind.

##AkshanshTips - If your data set is too large, you can use frame.head() , it will disply only first 5 rows so that you can see what kind of data you are handling. Or you can pass a number of rows like frame.head(10) to see first 10 rows

In [42]:

```
1 frame.head(3) #prints 3 rows from begining
```

Out[42]:

	state	year	population
0	ohio	2000	1.5
1	ohio	2001	1.7
2	ohio	2002	3.6

2) changing order of cols in DataFrame

pass a list of col and everything will be in order for you.

In [43]:

```
pd.DataFrame(data,columns=['year','state','population'])
```

Out[43]:

	year	state	population
0	2000	ohio	1.5
1	2001	ohio	1.7
2	2002	ohio	3.6
3	2001	nevada	2.4
4	2002	nevada	2.9
5	2003	nevada	3.2

3) adding a new col to DataFrame

if you passing a list in method columns, if an element is not present in already existed cols, it will add a new

In [44]:

```
frame2=pd.DataFrame(data,columns=['year','state','population','debt'])
```

In [45]:

```
frame2
```

Out[45]:

	year	state	population	debt
0	2000	ohio	1.5	NaN
1	2001	ohio	1.7	NaN
2	2002	ohio	3.6	NaN
3	2001	nevada	2.4	NaN
4	2002	nevada	2.9	NaN
5	2003	nevada	3.2	NaN

see it added a new col named debt and automatically all the values is missing in this column that is NaN (not a numner) type

4) changing the index

by default, indexing is always like 0,1,2,... but I don't like it like my ex. What to do then? I do the same like i did earlier, changed the girl and now I'll change index too. Simple.

In [46]:

```
frame2=pd.DataFrame(data,index=['one','two','three','four','five','six'])
```

In [47]:

```
frame2
```

Out[47]:

	state	year	population
one	ohio	2000	1.5
two	ohio	2001	1.7
three	ohio	2002	3.6
four	nevada	2001	2.4
five	nevada	2002	2.9
six	nevada	2003	3.2

5) accessing cols from DataFrame

In [48]:

```
frame2['state']  #prints state with index
```

Out[48]:

```
one        ohio
two        ohio
three      ohio
four     nevada
five     nevada
six      nevada
Name: state, dtype: object
```

##AkshanshTips - there is another way around to look into col

In [49]:

```
frame2.state
```

Out[49]:

```
one        ohio
two        ohio
three      ohio
four     nevada
five     nevada
six      nevada
Name: state, dtype: object
```

In [50]:

```
frame2[['state','population']]  #prints multiple cols
```

Out[50]:

	state	population
one	ohio	1.5
two	ohio	1.7
three	ohio	3.6
four	nevada	2.4
five	nevada	2.9
six	nevada	3.2

6) accessing row from DataFrame

If you want too access rows do bit of slicing like this. Its like cutting your data in terms of rows

In [51]:

```
frame2['one':'three']
```

Out[51]:

	state	year	population
one	ohio	2000	1.5
two	ohio	2001	1.7
three	ohio	2002	3.6

##AkshanshTips - use 'loc' to get all the collective data for one row

In [52]:

```
frame2.loc['three']
```

Out[52]:

```
state         ohio
year          2002
population     3.6
Name: three, dtype: object
```

In [53]:

```
frame2.loc['three':'four']  #this is kind of slicing in loc
```

Out[53]:

	state	year	population
three	ohio	2002	3.6
four	nevada	2001	2.4

7) filling NaN values

suppose we a DataFrame

In [54]:

```
frame3=pd.DataFrame(data,columns=['year','state','population','debt'])
```

In [55]:

```
frame3
```

Out[55]:

	year	state	population	debt
0	2000	ohio	1.5	NaN
1	2001	ohio	1.7	NaN
2	2002	ohio	3.6	NaN
3	2001	nevada	2.4	NaN
4	2002	nevada	2.9	NaN
5	2003	nevada	3.2	NaN

In [56]:

```
frame3.debt=16.5
```

In [57]:

```
frame3
```

Out[57]:

	year	state	population	debt
0	2000	ohio	1.5	16.5
1	2001	ohio	1.7	16.5
2	2002	ohio	3.6	16.5
3	2001	nevada	2.4	16.5
4	2002	nevada	2.9	16.5
5	2003	nevada	3.2	16.5

##AkshanshTips - frame3['debt']=16.5 will do the same

You can take advantage of NumPy too

In [58]:

```
frame3['debt']=np.arange(1,7)
```

In [59]:

```
frame3
```

Out[59]:

	year	state	population	debt
0	2000	ohio	1.5	1
1	2001	ohio	1.7	2
2	2002	ohio	3.6	3
3	2001	nevada	2.4	4
4	2002	nevada	2.9	5
5	2003	nevada	3.2	6

you can assign values only at your desired place too

In [60]:

```
val=pd.Series([-1.2,-1.5,-1.7],index=[3,4,5])
```

In [61]:

```
frame3['debt']=val
```

In [62]:

```
frame3
```

Out[62]:

	year	state	population	debt
0	2000	ohio	1.5	NaN
1	2001	ohio	1.7	NaN
2	2002	ohio	3.6	NaN
3	2001	nevada	2.4	-1.2
4	2002	nevada	2.9	-1.5
5	2003	nevada	3.2	-1.7

it assigned the values to the debt at index we passed in val series created just now

8) Deleting a col from DataFrame

In [63]:

```
del frame3['debt']
```

In [64]:

```
frame3
```

Out[64]:

	year	state	population
0	2000	ohio	1.5
1	2001	ohio	1.7
2	2002	ohio	3.6
3	2001	nevada	2.4
4	2002	nevada	2.9
5	2003	nevada	3.2

9) Transpose of a DataFrame

In this method, rows become cols and cols become rows

In [65]:

```
frame3
```

Out[65]:

	year	state	population
0	2000	ohio	1.5
1	2001	ohio	1.7
2	2002	ohio	3.6
3	2001	nevada	2.4
4	2002	nevada	2.9
5	2003	nevada	3.2

In [66]:

```
frame3.T
```

Out[66]:

	0	1	2	3	4	5
year	2000	2001	2002	2001	2002	2003
state	ohio	ohio	ohio	nevada	nevada	nevada
population	1.5	1.7	3.6	2.4	2.9	3.2

This was basic DataFrame and we are moving to more asking category now. Hope you are enjoying your time here.

Index Objects

--->Essential Functionality

My intention is not here to present an exhaustive documentation of library of Pandas, rather than going into this, I'll be making you familiar with most used/important functions.

1) Reindexing

a)Series

In [67]:

```
obj=pd.Series([4.5,7.2,-5.3,3.6],index=['d','b','a','c'])
```

In [68]:

```
obj
```

Out[68]:

```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

In [69]:

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

In [70]:

```
obj2
```

Out[70]:

```
a   -5.3
b    7.2
c    3.6
d    4.5
dtype: float64
```

everything is in order now, look at the index

b)DataFrame

In [71]:

```
frame=pd.DataFrame(np.arange(9).reshape((3,3)),
                   index=['a','c','d'],
                   columns=['ohio','texas','california'])
```

In [72]:

```
frame
```

Out[72]:

	ohio	texas	california
a	0	1	2
c	3	4	5
d	6	7	8

In [73]:

```
frame2=frame.reindex(['a','b','c','d'])
```

In [74]:

```
frame2
```

Out[74]:

	ohio	texas	california
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

see, index has been orderes in numerics way. Find that, b was not there initially, however when I passed it as a index, NaN automatically assigned to that.

coulumns can be reindexed by passing columns like I did for th index

In [75]:

```
states=['texas','utah','california']
```

In [76]:

```
frame.reindex(columns=states)
```

Out[76]:

	texas	utah	california
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

2) dropping entries from Axis

a) Series

In [77]:

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

In [78]:

```
obj
```

Out[78]:

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

In [79]:

```
obj.drop('c')  #it will drop c index
```

Out[79]:

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

What if i want to drop many, like your ex did to all the boys, when she found a perfect match for marriage

In [80]:

```
obj.drop(['c','d'])  #drops c and d from index
```

Out[80]:

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

b)DataFrame

In [81]:

```
df=pd.DataFrame(np.arange(16).reshape((4,4)),
                  index=['ohio','colorado','utah','New york'],
                  columns=['one','two','three','four'])
```

In [82]:

```
df #prints DataFrame we just created
```

Out[82]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11
New york	12	13	14	15

dropping is easy like girls do. Just say drop

In [83]:

```
df.drop('colorado') #colorado is no more :(
```

Out[83]:

	one	two	three	four
ohio	0	1	2	3
utah	8	9	10	11
New york	12	13	14	15

In [84]:

```
df.drop(['colorado','ohio'])
```

Out[84]:

	one	two	three	four
utah	8	9	10	11
New york	12	13	14	15

##AkshanshTips- dropping allows you to drop particular thing, however it doesn't get permanently deleted from the DataFrame.

In [85]:

```
df #let's check dropping is permanent or not?
```

Out[85]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11
New york	12	13	14	15

see, it is like your gf, not permanent. :-P

##AkshanshTips - axis=0 means rows, what if you want to drop cols, use axis=1

In [86]:

```
df.drop('two',axis=1)
```

Out[86]:

	one	three	four
ohio	0	2	3
colorado	4	6	7
utah	8	10	11
New york	12	14	15

'two' is not there in columns and again it is not permanent

##AkshanshTips - if you want to drop permanently, use inplace=True

In [87]:

```
df
```

Out[87]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11
New york	12	13	14	15

In [88]:

```
df.drop('ohio',axis=0,inplace=True)
```

In [89]:

```
df
```

Out[89]:

	one	two	three	four
colorado	4	5	6	7
utah	8	9	10	11
New york	12	13	14	15

see, ohio, got permanently deleted. You may use axis=0 or not. It's optional but when you are dropping cols, must use axis=1

3) Indexing, Selection, Filtering

In [90]:

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

In [91]:

```
obj
```

Out[91]:

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

In [92]:

```
obj['b'] #returns what value index b has
```

Out[92]:

```
1.0
```

In [93]:

```
obj[0] #passing default index that always start from 0
```

Out[93]:

```
0.0
```

In [94]:

```
obj[1:4] #slicing for sleection
```

Out[94]:

```
b    1.0
c    2.0
d    3.0
dtype: float64
```

In [95]:

```
obj>2 #prints boolean True False Shit that i don't like
```

Out[95]:

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

In [96]:

```
obj[obj>2] #prints wherre it found True
```

Out[96]:

```
d    3.0
dtype: float64
```

In [97]:

```
obj #look carefully to the series
```

Out[97]:

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

##AkshanshTips- You can assign a new value which is common by using slicing

In [98]:

```
obj['b':'d']=5
```

In [99]:

```
obj
```

Out[99]:

```
a    0.0
b    5.0
c    5.0
d    5.0
dtype: float64
```

Let's do this in DataFrame

In [100]:

```
df=pd.DataFrame(np.arange(16).reshape((4,4)),
                  index=['ohio','colorado','utah','New york'],
                  columns=['one','two','three','four'])
```

In [101]:

```
df
```

Out[101]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11
New york	12	13	14	15

In [102]:

```
df['two'] #prints values held by column two
```

Out[102]:

```
ohio         1
colorado     5
utah         9
New york    13
Name: two, dtype: int64
```

In [103]:

```
df[['two','three']]  #pass list of multiple cols in a list
```

Out[103]:

	two	three
ohio	1	2
colorado	5	6
utah	9	10
New york	13	14

In [104]:

```
df['ohio':'utah']  #prints rows from ohio to utah
```

Out[104]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11

In [105]:

```
df[0:3]  #can be done by passing default indexing
```

Out[105]:

	one	two	three	four
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11

In [106]:

```
df>5
```

Out[106]:

	one	two	three	four
ohio	False	False	False	False
colorado	False	False	True	True
utah	True	True	True	True
New york	True	True	True	True

In [107]:

```
df[df>5]  #do I need to tell this again?
```

Out[107]:

	one	two	three	four
ohio	NaN	NaN	NaN	NaN
colorado	NaN	NaN	6.0	7.0
utah	8.0	9.0	10.0	11.0
New york	12.0	13.0	14.0	15.0

let's remove NaN Values

In [108]:

```
df[df<5]= 0
```

In [109]:

```
df
```

Out[109]:

	one	two	three	four
ohio	0	0	0	0
colorado	0	5	6	7
utah	8	9	10	11
New york	12	13	14	15

##AkshanshTips- you can fill NaN values in a dataFrame by using fillna(0) method. Like df.fillna(0)

4) loc and iloc

loc and iloc enalble you to select a subset of the rows and columns from a DataFrame with Numpy like notation using eiher axis labels(loc) or integers (iloc)

##AkshanshTips - loc works with labels, iloc works with interger that's why 'i' is there

In [110]:

```
df
```

Out[110]:

	one	two	three	four
ohio	0	0	0	0
colorado	0	5	6	7
utah	8	9	10	11
New york	12	13	14	15

In [111]:

```
df.loc['colorado',['two','three']]  #what colorado has in two and three col
```

Out[111]:

```
two      5
three    6
Name: colorado, dtype: int64
```

In [112]:

```
df.iloc[1,[1,2]]  #1 is colorado, another 1,2 is col index
```

Out[112]:

```
two      5
three    6
Name: colorado, dtype: int64
```

In [114]:

```
df.iloc[2]  #print index2 for each col
```

Out[114]:

```
one       8
two       9
three    10
four     11
Name: utah, dtype: int64
```

In [116]:

```
df.iloc[[1,2],[3,1]]  #prints 1,2 index rows, and col index3 and 1
```

Out[116]:

	four	two
colorado	7	5
utah	11	9

In [120]:

```
df.iloc[1:3,1:3]  #this is slicing, rows, col
```

Out[120]:

	two	three
colorado	5	6
utah	9	10

see there are many ways to extract values, and arrage them. You can use whatever you like.

Airthmatic and DataFrame alignment

In [122]:

```
s1=pd.Series([7.3,-2.5,3.4,1.5],index=['a','c','d','e'])
```

In [123]:

```
s2=pd.Series([-2.1,3.6,-1.5,4,3.1],
             index=['a','c','e','f','g'])
```

In [127]:

```
s1
```

Out[127]:

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

In [128]:

```
s2
```

Out[128]:

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

In [130]:

```
s1+s2 #adds what's common there (index) other is NaN
```

Out[130]:

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

This was just addition, you can do other stuffs like subtraction and multiplication. Play with data like you play with your girlfriend's feeling :-P

In [131]:

```
df1=pd.DataFrame(np.arange(9).reshape((3,3)),
                 columns=list('bcd'),
                 index=['ohio','texas','colorado'])
```

look carefully I passed col, observe and you will learn

In [132]:

```
df2=pd.DataFrame(np.arange(12).reshape((4,3)),
                 columns=list('bde'),
                 index=['Utah','ohio','texas','oregon'])
```

In [135]:

```
df1
```

Out[135]:

	b	c	d
ohio	0	1	2
texas	3	4	5
colorado	6	7	8

In [136]:

```
df2
```

Out[136]:

	b	d	e
Utah	0	1	2
ohio	3	4	5
texas	6	7	8
oregon	9	10	11

In [137]:

```
df1+df2
```

Out[137]:

	b	c	d	e
Utah	NaN	NaN	NaN	NaN
colorado	NaN	NaN	NaN	NaN
ohio	3.0	NaN	6.0	NaN
oregon	NaN	NaN	NaN	NaN
texas	9.0	NaN	12.0	NaN

Again common index has been added, rest are just NaN. Do more things like subtraction and other stuff by your own.

Function application and mapping

In [144]:

```
df=pd.DataFrame(np.arange(12).reshape((4,3)),
                columns=list('bde'),
                index=['utah','ohio','texas','oregon'])
```

In [145]:

```
df
```

Out[145]:

	b	d	e
utah	0	1	2
ohio	3	4	5
texas	6	7	8
oregon	9	10	11

In [146]:

```
def my_func(x):
    return x.max()-x.min()
```

In [152]:

```
df.apply(my_func,axis=0)  #applies my function to dataFrame
```

Out[152]:

```
b    9
d    9
e    9
dtype: int64
```

##AkshanshTips - same can be done axis=0, means 'rows'

In [153]:

```
df.apply(my_func,axis='rows')
```

Out[153]:

```
b    9
d    9
e    9
dtype: int64
```

In [154]:

```
df.apply(my_func,axis='columns')
```

Out[154]:

```
utah      2
ohio      2
texas     2
oregon    2
dtype: int64
```

But it is not looking good, is it? Let me find you some other way around so that it can, atleast, look like a dataframe-

I'll return a Series from the function where value will be {max,min} and there will be two index, min and max

In [155]:

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

In [157]:

```
df.apply(new_func) #;et's apply new function
```

Out[157]:

	b	d	e
min	0	1	2
max	9	10	11

now it looks great, isn't it?

sorting and ranking

In [159]:

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

In [160]:

```
obj
```

Out[160]:

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

index is not right, alphabatically at least. I am here to sort this out for you.

In [161]:

```
obj.sort_index()
```

Out[161]:

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

In [162]:

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

In [163]:

```
frame
```

Out[163]:

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

In [165]:

```
frame.sort_index()  #sorts index alphabatically
```

Out[165]:

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

In [167]:

```
frame.sort_index(axis='columns')  #you know what it means
```

Out[167]:

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

In [168]:

```
frame.sort_index(axis='columns',ascending=False)
```

Out[168]:

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

cols are not in ascending order. That's what I meant by ascending = False

##AkshanshTips- if dataset has missing values, NaN, these will be printed in the last when it gets sorted

a) sorting by one specific col in dataframe

In [170]:

```
df=pd.DataFrame({'b':[4,7,-3,2], 'a':[0,1,0,1]})
```

created DataFrame with help of dictionary, use mind, I've already taught this in lecture note17 and probably in this lecture too. You've to find this by your own.

In [171]:

```
df
```

Out[171]:

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

In [174]:

```
df.sort_values('b')  #ascending order in b, rest remain same
```

Out[174]:

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

look carefully, index has been change accordance to b, because b is priority now

##AksanshTips- you can use sort_values(by=['a','b']) high priority will be given to 'a' and then 'b'. This is how you can sort multiple cols

Ranking

Ranking assigns ranks from one through the number of valid data points in an array. The rank methods for Series and DataFrame are the place to look; by default rank breaks ties by assigning each group the mean rank:

In [175]:

```
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

In [176]:

```
obj
```

Out[176]:

```
0    7
1   -5
2    7
3    4
4    2
5    0
6    4
dtype: int64
```

In [177]:

```
obj.rank()
```

Out[177]:

```
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

dealing with duplicates

Suppose your data has duplicate labels what will you do? No option man, you have to deal with them like you are dealing with all the problems you having right now. It depends how to deal with those.

In [178]:

```
obj=pd.Series(range(5),
              index=list('aabbc'))
```

In [179]:

```
obj
```

Out[179]:

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

let's check if your dataset is unique

et s c ec you dataset s u que

In [185]:

```
obj.is_unique  #it tells about the value
```

Out[185]:

```
True
```

In [187]:

```
obj.index.is_unique  #tells about the index
```

Out[187]:

```
False
```

In [189]:

```
obj['a'] #this is how you get it
```

Out[189]:

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

Same applies on DataFrame too

Summarizing and computing

I suggest you to get this know, it is kind of most usefull dealing with the numerical data in your company

In [212]:

```
df=pd.DataFrame([[1.4,np.nan],[7.1,-4.5],
                 [np.nan,np.nan],[0.75,-1.3]],
                index=list('abcd'),
                columns=['one','two'])
```

In [213]:

```
df
```

Out[213]:

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

In [215]:

```
df.sum()  #full col sum for each col
```

Out[215]:

```
one    9.25
two   -5.80
dtype: float64
```

In [217]:

```
df.sum(axis='rows')  #can be done so too,veritcal sum
```

Out[217]:

```
one    9.25
two   -5.80
dtype: float64
```

In [220]:

```
df.sum(axis='columns')  #sum of each col, row wise, horizontal Sum
```

Out[220]:

```
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

In [223]:

```
df.mean(axis='rows')
```

Out[223]:

```
one    3.083333
two   -2.900000
dtype: float64
```

In [224]:

```
df.mean(axis='columns')
```

Out[224]:

```
a    1.400
b    1.300
c      NaN
d   -0.275
dtype: float64
```

some methods like idxmin and idxmax return indirect statistics like the index value where the minimum value is there

(

In [225]:

```
df.idxmax()
```

Out[225]:

```
one    b
two    d
dtype: object
```

In [226]:

```
df.idxmin()
```

Out[226]:

```
one    d
two    b
dtype: object
```

##AkshanshTips- Why run after each statistics, when you can just - describe it

In [228]:

```
df.describe()
```

Out[228]:

	one	two
count	3.000000	2.000000
mean	3.083333	-2.900000
std	3.493685	2.262742
min	0.750000	-4.500000
25%	1.075000	-3.700000
50%	1.400000	-2.900000
75%	4.250000	-2.100000
max	7.100000	-1.300000

In [ ]:

```

```