2)Pandas

October 1, 2022

#Series #Series is the first main datatype that will be working with pandas.Series it can be indexed by a label.

[5]:
```
import numpy as np
import pandas as pd#Creating various series from various object types
labels=['a','b','c']
my_data=[10,20,30]
arr=np.array(my_data)
d={'a':10,'b':20,'c':30}
pd.Series(data = my_data)#INIT SIGNATURE:pd.series(data=None, index=None,␣
 ↪dtype=None, name=None,copy=False, fastpath=False)
```

[5]:
```
0    10
1    20
2    30
dtype: int64
```

[6]:
```
pd.Series(data=my_data,index=labels)
```

[6]:
```
a    10
b    20
c    30
dtype: int64
```

[7]:
```
pd.Series(my_data,labels)
```

[7]:
```
a    10
b    20
c    30
dtype: int64
```

[9]:
```
pd.Series(arr,labels)
```

[9]:
```
a    10
b    20
c    30
dtype: int32
```

[10]:
```
pd.Series(d)
```

[10]:
```
a    10
b    20
c    30
dtype: int64
```

[11]:
```
d
```

[11]:
```
{'a': 10, 'b': 20, 'c': 30}
```

[12]:
```
pd.Series(data=labels)
```

[12]:
```
0    a
1    b
2    c
dtype: object
```

[13]:
```
pd.Series(data=[sum,print,len])
```

[13]:
```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

[14]:
```
#Grabing info from a series
ser1=pd.Series([1,2,3,4],['USA','Germany','USSR','Japan'])
ser1
```

[14]:
```
USA        1
Germany    2
USSR       3
Japan      4
dtype: int64
```

[15]:
```
ser2=pd.Series([1,2,5,4],['USA','Germany','Italy','Japan'])
ser2
```

[15]:
```
USA        1
Germany    2
Italy      5
Japan      4
dtype: int64
```

[16]:
```
ser1['USA']#Now grabbing info
```

[16]:
```
1
```

[17]:
```
ser3=pd.Series(data=labels)
ser3
```

[17]:
```
0    a
1    b
2    c
dtype: object
```

[18]:
```
ser3[0]
```

[18]:
```
'a'
```

[19]:
```
ser1
```

[19]:
```
USA        1
Germany    2
USSR       3
Japan      4
dtype: int64
```

[20]:
```
ser2
```

[20]:
```
USA        1
Germany    2
Italy      5
Japan      4
dtype: int64
```

[21]:
```
ser1+ser2#Here in italy we get Null or Nan bcoz italy has no match ser1
#When you performing operations with pandas base objects you're integers are
#going to be converted into floats and that's so you cannot accidentally lose
#info based of some weird division and that has the fact that true division vs
#class division is actually differentialted in the older version.
```

[21]:
```
Germany    4.0
Italy      NaN
Japan      8.0
USA        2.0
USSR       NaN
dtype: float64
```

#Data frames

[40]:
```
import numpy as np
import pandas as pd
from numpy.random import randn
np.random.seed(101)#Here seed mean is just to make sure that we get the same
                   #random numbers.
```

[41]:
```
df=pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])
            #INIT Signature: pd.DataFrame(data=None,index=None,columns=None,
            #dtype=None,copy=False)
```

[42]:
```
df
```

[42]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[43]:
```
df['W']#grabbing values using indexing and selection
```

[43]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

[44]:
```
type(df['W'])
```

[44]:
```
pandas.core.series.Series
```

[45]:
```
#This one type of grabbing columns
type(df)#Data frame is just a bunch of series that share the same index
```

[45]:
```
pandas.core.frame.DataFrame
```

[46]:
```
df.W#This method may get confused
```

[46]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

[47]:
```
df[['W','Z']]
```

[47]:
```
          W         Z
A  2.706850  0.503826
B  0.651118  0.605965
C -2.018168 -0.589001
D  0.188695  0.955057
E  0.190794  0.683509
```

[48]:
```
df['new']=df['W']+df['Y']
df
```

[48]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

[49]:
```
df.drop('new',axis=1,inplace=True)#For removing columns#axis=0 is refers to␣
 ↪index.If you
#want to refer to columns then it is axis=1
    #df.drop(
    #labels=None,
    #axis: 'Axis' = 0,
    #index=None,
    #columns=None,
    #level: 'Level | None' = None,
    #inplace: 'bool' = False,
    #errors: 'str' = 'raise',) #Here we use inplace bcoz to not loose info
```

[50]:
```
df
```

[50]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

[51]:
```
df.drop('E')
```

[51]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

[52]:
```
df.shape
```

[52]:
```
(5, 4)
```

[53]:
```
df.loc['A']#First way of grabbinf a row
```

[53]:
```
W    2.706850
X    0.628133
Y    0.907969
```

```
Z    0.503826
Name: A, dtype: float64
```

[54]:
```
df.iloc[0]#Second way of selecting row
```

[54]:
```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

[55]:
```
#Selecting subsets of rows and columns
df.loc['B','Y']
```

[55]:
```
-0.8480769834036315
```

[56]:
```
df.loc[['A','B'],['W','Y']]
```

[56]:
```
          W         Y
A  2.706850  0.907969
B  0.651118 -0.848077
```

1 Conditional Selection

[57]:
```
df>0#Here i will get data frame values true if it is >0 or false if <0
```

[57]:
```
       W      X      Y      Z
A   True   True   True   True
B   True  False  False   True
C  False   True   True  False
D   True  False  False   True
E   True   True   True   True
```

[58]:
```
booldf=df>0
booldf
```

[58]:
```
       W      X      Y      Z
A   True   True   True   True
B   True  False  False   True
C  False   True   True  False
D   True  False  False   True
E   True   True   True   True
```

[59]:
```
df[booldf]
```

[59]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
```

```
B  0.651118       NaN       NaN  0.605965
C       NaN  0.740122  0.528813       NaN
D  0.188695       NaN       NaN  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[60]:
```
df[df>0]
```

[60]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118       NaN       NaN  0.605965
C       NaN  0.740122  0.528813       NaN
D  0.188695       NaN       NaN  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[61]:
```
df['W']>0
```

[61]:
```
A     True
B     True
C    False
D     True
E     True
Name: W, dtype: bool
```

[62]:
```
df['W']
```

[62]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

[63]:
```
df[df['W']>0]
```

[63]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[64]:
```
df[df['Z']<0]
```

[64]:
```
          W         X         Y         Z
C -2.018168  0.740122  0.528813 -0.589001
```

[65]:
```
resultdf=df[df['W']>0]
resultdf
```

[65]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[66]:
```
resultdf['X']#Grabbing the X column
```

[66]:
```
A    0.628133
B   -0.319318
D   -0.758872
E    1.978757
Name: X, dtype: float64
```

[67]:
```
df[df['W']>0]['X']#here we are grabbing X column from the result of W>0
```

[67]:
```
A    0.628133
B   -0.319318
D   -0.758872
E    1.978757
Name: X, dtype: float64
```

[68]:
```
df[df['W']>0][['Y','X']]
```

[68]:
```
          Y         X
A  0.907969  0.628133
B -0.848077 -0.319318
D -0.933237 -0.758872
E  2.605967  1.978757
```

[69]:
```
boolser=df['W']>0
boolser
```

[69]:
```
A     True
B     True
C    False
D     True
E     True
Name: W, dtype: bool
```

[70]:
```
boolser=df['W']>0
result=df[boolser]
result
```

[70]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
```

```
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[71]:
```
boolser=df['W']>0
result=df[boolser]
result[['X','Y']]
```

[71]:
```
          X         Y
A  0.628133  0.907969
B -0.319318 -0.848077
D -0.758872 -0.933237
E  1.978757  2.605967
```

[72]:
```
boolser=df['W']>0
result=df[boolser]
mycols=['X','Y']
result[mycols]
```

[72]:
```
          X         Y
A  0.628133  0.907969
B -0.319318 -0.848077
D -0.758872 -0.933237
E  1.978757  2.605967
```

[73]:
```
#Using multiple conditions
```

[74]:
```
df[(df['W']>0)&(df['Y']>1)]#We will get the o/p where it satisfies bot
```

[74]:
```
          W         X         Y         Z
E  0.190794  1.978757  2.605967  0.683509
```

[76]:
```
df[(df['W']>0) | (df['Y']>1)]#Here we cannot use the "and,or" bcoz of ambiguous
```

[76]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

2 Resetting the index or setting it to something else

[77]:
```
df
```

[77]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
```

```
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[80]:
```
df.reset_index()#df.reset_index(
    #level: 'Hashable | Sequence[Hashable] | None' = None,
   # drop: 'bool' = False,
   # inplace: 'bool' = False,
   # col_level: 'Hashable' = 0,
   # col_fill: 'Hashable' = '',)
```

[80]:

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

[81]:
```
newind='CA NY WY OR CO'.split()#Setting into something
```

[84]:
```
newind
```

[84]:
```
['CA', 'NY', 'WY', 'OR', 'CO']
```

[90]:
```
df['States']=newind
```

[91]:
```
df
```

[91]:

	W	X	Y	Z	States	India
A	2.706850	0.628133	0.907969	0.503826	CA	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR	OR
E	0.190794	1.978757	2.605967	0.683509	CO	CO

[98]:
```
df.set_index('States')#in this it will not allow you to have the old column
                      #reset index
```

[98]:

States	W	X	Y	Z	India
CA	2.706850	0.628133	0.907969	0.503826	CA
NY	0.651118	-0.319318	-0.848077	0.605965	NY
WY	-2.018168	0.740122	0.528813	-0.589001	WY
OR	0.188695	-0.758872	-0.933237	0.955057	OR
CO	0.190794	1.978757	2.605967	0.683509	CO

[96]:
```
df
```

[96]:
```
          W         X         Y         Z States India
A  2.706850  0.628133  0.907969  0.503826     CA    CA
B  0.651118 -0.319318 -0.848077  0.605965     NY    NY
C -2.018168  0.740122  0.528813 -0.589001     WY    WY
D  0.188695 -0.758872 -0.933237  0.955057     OR    OR
E  0.190794  1.978757  2.605967  0.683509     CO    CO
```

3 MULTI INDEX DATA

[5]:
```
import numpy as np
import pandas as pd
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)
```

[6]:
```
hier_index
```

[6]:
```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

[10]:
```
df = pd.DataFrame(np.random.randn(6,2),hier_index,['A','B'])
df
```

[10]:
```
              A         B
G1 1  1.339117  1.165432
   2 -0.348278 -0.081270
   3 -1.028825 -1.491024
G2 1  0.081635 -0.135000
   2  0.118729  0.418711
   3  0.648210  2.139573
```

[11]:
```
df.loc['G1']
```

[11]:
```
          A         B
1  1.339117  1.165432
2 -0.348278 -0.081270
3 -1.028825 -1.491024
```

[15]:
```
df.loc['G1'].loc[1]
```

[15]:
```
A    1.339117
B    1.165432
Name: 1, dtype: float64
```

[18]:
```
df.index.names=['Groups','Numbers']#for naming
df
```

[18]:
```
                      A         B
Groups Numbers
G1     1        1.339117  1.165432
       2       -0.348278 -0.081270
       3       -1.028825 -1.491024
G2     1        0.081635 -0.135000
       2        0.118729  0.418711
       3        0.648210  2.139573
```

[21]:
```
df.loc['G2'].loc[2]['B']#Here we grabbed the number from G2 in 2nd line in B col
```

[21]:
```
0.41871098062857814
```

[22]:
```
df.loc['G1'].loc[3]['A']
```

[22]:
```
-1.0288249773715503
```

[23]:
```
df.xs('G1')#This xs func returns a cross section of rows and columns from a␣
 ↪series
      #data frames we use it in multilevel index
```

[23]:
```
                A         B
Numbers
1        1.339117  1.165432
2       -0.348278 -0.081270
3       -1.028825 -1.491024
```

[24]:
```
df.xs(1,level='Numbers')#here we grabbed the numbers from '1' in G1&G2 in␣
 ↪level='numbers'
```

[24]:
```
                A         B
Groups
G1       1.339117  1.165432
G2       0.081635 -0.135000
```

4 MISSING DATA

[ ]:
```
#This missing data was used when you're using pandas to read in data if you
# missing points what's going to happen is pandas will automatically fill in
# missing point.
```

[1]:
```
import numpy as np
import pandas as pd
d={'A':[1,2,np.nan],'B':[5,np.nan,np.nan],'C':[1,2,3]}
df=pd.DataFrame(d)     #Creating a dataframe from a dictionary as well
df                      #Here np.nan signifies missing number or variable
```

[1]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

[2]:
```
#Drop and a method
```

[3]:
```
df.dropna()#here it drops only the rows which doesn't have any nan values
```

[3]:

	A	B	C
0	1.0	5.0	1

[4]:
```
df.dropna(axis=1)
```

[4]:

	C
0	1
1	2
2	3

[5]:
```
df.dropna(thresh=2)#here row one has atleast two non nan values
```

[5]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

[7]:
```
df.dropna(thresh=2)
```

[7]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

[ ]:
```
#now filling values in missing positions
```

[8]:
```
df.fillna(value='FILL VALUE')#df.fillna(
   # value: 'object | ArrayLike | None' = None,
   # method: 'FillnaOptions | None' = None,
    #axis: 'Axis | None' = None,
   # inplace: 'bool' = False,
    #limit=None,
   # downcast=None,)
```

[8]:
```
              A           B  C
0         1.0         5.0  1
1         2.0  FILL VALUE  2
2  FILL VALUE  FILL VALUE  3
```

[9]:
```
df['A'].fillna(value=df['A'].mean())
```

[9]:
```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

5 Groupby

[ ]:
```
#groupby allows you to group together rows based off of a column and perform
#an aggregate function on them
```

[2]:
```
import numpy as np
import pandas as pd
data={'company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
      'person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
     'Sales':[200,120,340,124,243,350]}
df=pd.DataFrame(data)
df
```

[2]:
```
  company   person  Sales
0    GOOG      Sam    200
1    GOOG  Charlie    120
2    MSFT      Amy    340
3    MSFT  Vanessa    124
4      FB     Carl    243
5      FB    Sarah    350
```

[3]:
```
df.groupby('company')
```

[3]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001D2B48E4FA0>
```

[4]:
```
bycomp=df.groupby('company')
bycomp.mean()
```

[4]:
```
         Sales
company
FB       296.5
GOOG     160.0
MSFT     232.0
```

[5]:
```
bycomp.sum()
```

[5]:
```
          Sales
company
FB          593
GOOG        320
MSFT        464
```

[6]:
```
bycomp.std()
```

[6]:
```
              Sales
company
FB        75.660426
GOOG      56.568542
MSFT     152.735065
```

[7]:
```
bycomp.sum().loc['FB']
```

[7]:
```
Sales    593
Name: FB, dtype: int64
```

[8]:
```
df.groupby('company').sum().loc['FB']
```

[8]:
```
Sales    593
Name: FB, dtype: int64
```

[9]:
```
df.groupby('company').count()
```

[9]:
```
         person  Sales
company
FB            2      2
GOOG          2      2
MSFT          2      2
```

[10]:
```
df.groupby('company').max()
```

[10]:
```
          person  Sales
company
FB         Sarah    350
GOOG         Sam    200
MSFT     Vanessa    340
```

[12]:
```
df.groupby('company').min()
```

[12]:
```
          person  Sales
company
FB          Carl    243
GOOG     Charlie    120
MSFT         Amy    124
```

[14]:
```
df.groupby('company').describe()
```

[14]:

	Sales							
	count	mean	std	min	25%	50%	75%	max
company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

[15]:
```
df.groupby('company').describe().transpose()
```

[15]:

company		FB	GOOG	MSFT
Sales	count	2.000000	2.000000	2.000000
	mean	296.500000	160.000000	232.000000
	std	75.660426	56.568542	152.735065
	min	243.000000	120.000000	124.000000
	25%	269.750000	140.000000	178.000000
	50%	296.500000	160.000000	232.000000
	75%	323.250000	180.000000	286.000000
	max	350.000000	200.000000	340.000000

[16]:
```
df.groupby('company').describe().transpose()['FB']
```

[16]:

Sales	count	2.000000
	mean	296.500000
	std	75.660426
	min	243.000000
	25%	269.750000
	50%	296.500000
	75%	323.250000
	max	350.000000

Name: FB, dtype: float64

6 merging,joining and concatenating

[17]:
```
import numpy as np
import pandas as pd
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
```

```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])
df1
```

[17]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

[18]:
```
df2
```

[18]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

[19]:
```
df3
```

[19]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Concatenation Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use pd.concat and pass in a list of DataFrames to concatenate together:

[20]:
```
pd.concat([df1,df2,df3])
```

[20]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

[21]:
```
pd.concat([df1,df2,df3],axis=1)
```

[21]:

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

[27]:
```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                     'B':['B0','B1','B2','B3']})
right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                          'C': ['C0', 'C1', 'C2', 'C3'],
                          'D': ['D0', 'D1', 'D2', 'D3']})
```

[28]:
```
left
```

[28]:

	key	A	B
0	k0	A0	B0
1	k1	A1	B1
2	k2	A2	B2
3	k3	A3	B3

[29]:
```
right
```

[29]:

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

7 Merging

The merge function allows you to merge DataFrames together using a similar logic as merging SQL Tables together. For example:

[35]:
```
pd.merge(left,right,how='inner',on='key')
```

[35]:
```
Empty DataFrame
Columns: [key, A, B, C, D]
Index: []
```

[ ]:
```
#complicated example
```

[36]:
```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                                'key2': ['K0', 'K0', 'K0', 'K0'],
                                   'C': ['C0', 'C1', 'C2', 'C3'],
                                   'D': ['D0', 'D1', 'D2', 'D3']})
```

[37]:
```
pd.merge(left,right,on=['key1','key2'])
```

[37]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

[38]:
```
pd.merge(left,right,how='outer',on=['key1','key2'])
```

[38]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

[39]:
```
pd.merge(left,right,how='right',on=['key1','key2'])
```

[39]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

[41]:
```
pd.merge(left,right,how='left',on=['key1','key2'])
```

[41]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2

```
4   K2   K1  A3  B3  NaN  NaN
```

Joining Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

[43]:
```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                    'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

[44]:
```
left.join(right)
```

[44]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

[45]:
```
left.join(right,how='outer')
```

[45]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

8 operations

[75]:
```
import numpy as np
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],
                   'col2':[444,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

[75]:

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

[76]:
```
#finding unique values in column2
df['col2'].unique()
```

[76]:
```
array([444, 555, 666], dtype=int64)
```

[77]:
```
#if you want the number of value itself from column2
#1st method
len(df['col2'].unique())
```

[77]:
```
3
```

[78]:
```
#2nd method
df['col2'].nunique()
```

[78]:
```
3
```

[79]:
```
#if you want a table of the unique values and how many times they show up
df['col2'].value_counts()
```

[79]:
```
444    2
555    1
666    1
Name: col2, dtype: int64
```

[80]:
```
#selecting data
df
```

[80]:
```
   col1  col2 col3
0     1   444  abc
1     2   555  def
2     3   666  ghi
3     4   444  xyz
```

[81]:
```
df[df['col1']>2]
```

[81]:
```
   col1  col2 col3
2     3   666  ghi
3     4   444  xyz
```

[82]:
```
df['col1']>2
```

[82]:
```
0    False
1    False
2     True
3     True
Name: col1, dtype: bool
```

[83]:
```
#if you want to combine conditions
df[(df['col1']>2) & (df['col2']==444)]
```

[83]:
```
   col1  col2 col3
3     4   444  xyz
```

9 Applying functions

[84]:
```
def times2(x):
    return x*2
```

[85]:
```
df['col1']
```

[85]:
```
0    1
1    2
2    3
3    4
Name: col1, dtype: int64
```

[86]:
```
df['col1'].sum()
```

[86]:
```
10
```

[87]:
```
df['col1'].apply(times2)
```

[87]:
```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

[88]:
```
#if you want a col that represent the length of the string
df['col3'].apply(len)
```

[88]:
```
0    3
1    3
2    3
3    3
Name: col3, dtype: int64
```

[89]:
```
#lambda expression times 2
df['col2'].apply(lambda x: x*2)
```

[89]:
```
0     888
1    1110
2    1332
3     888
Name: col2, dtype: int64
```

[90]:
```
#removing columns
df.drop('col1',axis=1)
```

[90]:
```
   col2 col3
0   444  abc
```

```
1   555  def
2   666  ghi
3   444  xyz
```

[92]:
```
df.columns#if you want the column names
```

[92]:
```
Index(['col1', 'col2', 'col3'], dtype='object')
```

[94]:
```
df.index#now it gives the info index
```

[94]:
```
RangeIndex(start=0, stop=4, step=1)
```

[95]:
```
#Sorting and ordering data frame
df
```

[95]:
```
   col1  col2 col3
0     1   444  abc
1     2   555  def
2     3   666  ghi
3     4   444  xyz
```

[96]:
```
df.sort_values(by='col2') #inplace=False by default
```

[96]:
```
   col1  col2 col3
0     1   444  abc
3     4   444  xyz
1     2   555  def
2     3   666  ghi
```

[97]:
```
#** Find Null Values or Check for Null Values**
df.isnull()
```

[97]:
```
    col1   col2   col3
0  False  False  False
1  False  False  False
2  False  False  False
3  False  False  False
```

[98]:
```
# Drop rows with NaN Values
df.dropna()
```

[98]:
```
   col1  col2 col3
0     1   444  abc
1     2   555  def
2     3   666  ghi
3     4   444  xyz
```

[2]:
```
#** Filling in NaN values with something else: **
import numpy as np
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

[2]:

	col1	col2	col3
0	1.0	NaN	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	NaN	444.0	xyz

[3]:
```
df.fillna('FILL')
```

[3]:

	col1	col2	col3
0	1.0	FILL	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	FILL	444.0	xyz

[6]:
```
data = {'A':['foo','foo','foo','bar','bar','bar'],
     'B':['one','one','two','two','one','one'],
       'C':['x','y','x','y','x','y'],
       'D':[1,3,2,5,4,1]}

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

[6]:

	A	B	C	D
0	foo	one	x	1
1	foo	one	y	3
2	foo	two	x	2
3	bar	two	y	5
4	bar	one	x	4
5	bar	one	y	1

10 pivot table

[7]:
```
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
#df.pivot_table(values=None, index=None,columns=None, aggfunc='mean',
   # fill_value=None,
    #margins=False,
    #dropna=True,
    #margins_name='All',
    #observed=False,
```

```
    # sort=True,)
```

[7]:
```
C          x    y
A   B
bar one  4.0  1.0
    two  NaN  5.0
foo one  1.0  3.0
    two  2.0  NaN
```

11 Data input and output

[19]:
```
import numpy as np
import pandas as pd
#NOW HOW TO OPEN AND READ CSV FILES
```

[29]:
```
#CSV
pd.read_csv('data.csv')
```

[ ]:
```
df=pd.read_csv('example.csv')
```

[26]:
```
df.to_csv('My_output',index=False)
```

[ ]:
```
pd.read_csv('My_output')
```

[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:

[ ]: