In [15]:
```
import pandas as pd
import numpy as np

frame3 = pd.DataFrame (np.random.randn(4,3), columns = list('bde'),
                               index = ['lahore', 'bahawalpur', 'karachi', 'multar

print(frame3)
```

```
                   b         d         e
lahore      0.842605 -1.355124  0.361847
bahawalpur  0.086141 -0.012537  1.714211
karachi    -0.613379  0.911665 -0.620434
multan     -0.509568  1.842854  1.122673
```

In [16]:
```
frame3 = pd.DataFrame (np.random.randn(4,3), columns = list('bde'),
                               index = ['lahore', 'bahawalpur', 'karachi', 'multar

print(frame3)
print(np.abs(frame3))    #  abs function convert negative values into positive va
```

```
                   b         d         e
lahore      1.126847 -0.045868  0.075699
bahawalpur -1.241057 -0.599871 -1.500992
karachi    -0.765867 -1.988766  1.854844
multan     -0.025768 -0.489770 -0.580077
                   b         d         e
lahore      1.126847  0.045868  0.075699
bahawalpur  1.241057  0.599871  1.500992
karachi     0.765867  1.988766  1.854844
multan      0.025768  0.489770  0.580077
```

In [23]:
```
frame3 = pd.DataFrame (np.random.randn(4,3), columns = list('bde'),
                               index = ['lahore', 'bahawalpur', 'karachi', 'multar

print(frame3)
#  print(np.abs(frame3))    #  abs function convert negative values into positive

print(frame3['d'].min())    #  columns d ,find minimum value
print(frame3['d'].max())    #  columns d, find maximum value
print(frame3['d'].max() - frame3['d'].min())      #  In column d, subtracting mir
```

```
                   b         d         e
lahore     -0.814032  0.882637 -0.319169
bahawalpur -0.716620  0.587918 -0.422383
karachi    -0.509567 -0.893871 -0.279014
multan      1.198797 -1.236192 -1.980809
-1.2361917109915523
0.8826374346996682
2.1188291456912207
```

In [32]:
```
frame3 = pd.DataFrame (np.random.randn(4,3), columns = list('bde'),
                               index = ['lahore', 'bahawalpur', 'karachi', 'multar

print(frame3)
f = lambda x: x.max() - x.min()   #  here ,column d , find max and min then subtr
df = frame.apply(f)               #    apply same on three columns
print(df,type(df))
```

```
                   b         d         e
lahore      0.648578 -1.151926  0.593820
bahawalpur -0.039923  0.370504 -0.623413
karachi     1.415547 -0.592531  0.258142
multan     -1.071753 -0.909579  0.613447
b    1.770147
d    1.579227
e    0.608786
dtype: float64 <class 'pandas.core.series.Series'>
```

In [37]:
```
frame3 = pd.DataFrame (np.random.randn(4,3), columns = list('bde'),
                                index = ['lahore', 'bahawalpur', 'karachi', 'multar

print(frame3)
f = lambda x: x.max() - x.min()   #  here ,column d ,frame3 = pd.DataFrame (np.ra
#df = frame.apply(f)               #    apply same on three columns
#print(df,type(df))

print(frame3.apply(f, axis=1 ))     #   here, rows, find max and min then subtrac
print(df)
```

```
                   b         d         e
lahore     -1.155702  0.972328  0.759126
bahawalpur  0.151962 -0.085491  0.133429
karachi     0.873844 -0.476051  0.180900
multan      0.638552  0.514380  1.479885
lahore        2.128031
bahawalpur    0.237453
karachi       1.349895
multan        0.965504
dtype: float64
b    1.770147
d    1.579227
e    0.608786
dtype: float64
```

In [40]:
```
def min_max(x):
    minimum = x.min()
    maximum = x.max()
    return pd.Series([maximum, minimum], index = ['min', 'max'])
df = frame3.apply(min_max)
print(df,type(df))
```

```
            b         d         e
min  0.873844  0.972328  1.479885
max -1.155702 -0.476051  0.133429 <class 'pandas.core.frame.DataFrame'>
```

Sorting and Ranking

In [51]:
```
#   Sort by indexes

frame5 = pd.DataFrame(np.arange(8).reshape((2, 4)),
                      index = ['three', 'one'],
                      columns = ['d','a', 'b' , 'c'])
print(frame5)
print()
print(frame5.sort_index(axis=1, ascending=False))
print()
print(frame5.sort_index())
#default in sort:axis=0 , ascending=True
```

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

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

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

In [17]:
```
#   Sorting by Value
import pandas as pd
import numpy as np
frame5 = pd.DataFrame(np.arange(8).reshape((2, 4)),
                      index = ['three', 'one'],
                      columns = ['d','a', 'b' , 'c'])
#print(frame5)
#print('    ')
print(frame5.sort_values(by = 'b'))
print(frame5.rank(ascending=False , method='max'))
print(frame5.rank(ascending=True , method='min'))
print(frame5.rank(axis = 'columns'))
```

```
       d  a  b  c
three  0  1  2  3
one    4  5  6  7
         d    a    b    c
three  2.0  2.0  2.0  2.0
one    1.0  1.0  1.0  1.0
         d    a    b    c
three  1.0  1.0  1.0  1.0
one    2.0  2.0  2.0  2.0
         d    a    b    c
three  1.0  2.0  3.0  4.0
one    1.0  2.0  3.0  4.0
```

In [16]:
```
print( frame5.sort_values(by='b') )
print(frame5.rank(ascending=False, method='max'))
print(frame5.rank(ascending=True, method='min'))
print( frame5.rank(axis='columns'))
```

```
       d  a  b  c
three  0  1  2  3
one    4  5  6  7
         d    a    b    c
three  2.0  2.0  2.0  2.0
one    1.0  1.0  1.0  1.0
         d    a    b    c
three  1.0  1.0  1.0  1.0
one    2.0  2.0  2.0  2.0
         d    a    b    c
three  1.0  2.0  3.0  4.0
one    1.0  2.0  3.0  4.0
```

Summarizing and computing Descriptive Statistics

In [20]:
```
df = pd.DataFrame ([[1.4, np.nan],[3.3,-5.4],[np.nan,np.nan],[0.3, -4.4]],
                   index = ['a','b','c','d'], columns = ['one','two'])

print(df)
print()
print(df.sum())        #  normal sum
print()
print(df.sum(axis='columns'))    #  if axis = column , then column wise sum
```

```
   one  two
a  1.4  NaN
b  3.3 -5.4
c  NaN  NaN
d  0.3 -4.4

one    5.0
two   -9.8
dtype: float64

a    1.4
b   -2.1
c    0.0
d   -4.1
dtype: float64
```

In [21]:
```
#   use of skipana
```

In [24]:
```
print(df)
x = df.mean(axis='columns' , skipna = False)
print()            #  mean operation / average operation
print(x)
```

```
   one  two
a  1.4  NaN
b  3.3 -5.4
c  NaN  NaN
d  0.3 -4.4

a     NaN
b   -1.05
c     NaN
d   -2.05
dtype: float64
```

In [25]:
```
print(df)
x = df.mean(axis='columns' )    #   Remove skipna
print()            #  mean operation / average operation
print(x)
```

```
   one  two
a  1.4  NaN
b  3.3 -5.4
c  NaN  NaN
d  0.3 -4.4

a    1.40
b   -1.05
c     NaN
d   -2.05
dtype: float64
```

In [26]:
```
#   Unique Values
```

In [27]:
```
df = pd.DataFrame([
                    [1.4,1.4, 1.5, np.nan], [7.1, -4.5, 1.5, 1.4],
                    [1.4, np.nan, 0.5, np.nan], [0.75, -1.3, 1.3, np.nan]
                   ], index=['a', 'b', 'c', 'd'], columns=['one', 'two', 'three',
print(df)
print()
```

```
    one  two  three  four
a  1.40  1.4    1.5   NaN
b  7.10 -4.5    1.5   1.4
c  1.40  NaN    0.5   NaN
d  0.75 -1.3    1.3   NaN
```

In [30]:
```
df = pd.DataFrame([
                    [1.4,1.4, 1.5, np.nan], [7.1, -4.5, 1.5, 1.4],
                    [1.4, np.nan, 0.5, np.nan], [0.75, -1.3, 1.3, np.nan]
                   ], index=['a', 'b', 'c', 'd'], columns=['one', 'two', 'three',
print(df)
print()
print(df['one'].unique(), df['two'].unique())
df['one'].value_counts()
```

```
    one  two  three  four
a  1.40  1.4    1.5   NaN
b  7.10 -4.5    1.5   1.4
c  1.40  NaN    0.5   NaN
d  0.75 -1.3    1.3   NaN

[1.4  7.1  0.75] [ 1.4 -4.5  nan -1.3]
```

Out[30]:
```
1.40    2
0.75    1
7.10    1
Name: one, dtype: int64
```

Remember ! index length must be equal to value (or do not provide index)

In [1]:
```
#Assigning a column that already exist will ________ and
# assigning a column that does not exist will ________
```

In [5]:
```
# how to reindex and use of fill value of parameter method
import pandas as pd
obj3 = pd.Series (['blue', 'purple', 'yellow'], index = [0,3,6])
# print(obj3)

# might create new rows
obj3 = obj3.reindex(range(9))
print(obj3)
```

```
0      blue
1       NaN
2       NaN
3    purple
4       NaN
5       NaN
6    yellow
7       NaN
8       NaN
dtype: object
```

In [7]:
```
# how to reindex and use of fill value of parameter method
import pandas as pd
obj3 = pd.Series (['blue', 'purple', 'yellow'], index = [0,3,6])
# print(obj3)

# might create new rows
#obj3 = obj3.reindex(range(9))
#print(obj3)
obj3 = obj3.reindex(range(9), method = 'ffill')
print(obj3)
```

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

In [8]:
```
# how to reindex and use of fill value of parameter method
import pandas as pd
obj3 = pd.Series (['blue', 'purple', 'yellow'], index = [0,3,6])
# print(obj3)

# might create new rows
#obj3 = obj3.reindex(range(9))
#print(obj3)
#obj3 = obj3.reindex(range(9), method = 'ffill')
#print(obj3)
obj3 = obj3.reindex(range(2,11), method = 'ffill')
print(obj3)
```

```
2       blue
3     purple
4     purple
5     purple
6     yellow
7     yellow
8     yellow
9     yellow
10    yellow
dtype: object
```

In [11]:
```
import numpy as np
import pandas as pd

states = pd.DataFrame(np.arange(9).reshape((3,3)),index = ['a', 'c','d'], columns
print(states)
```

```
   punjab  sindh  balochistan
a       0      1            2
c       3      4            5
d       6      7            8
```

In [15]:
```
import numpy as np
import pandas as pd

states = pd.DataFrame(np.arange(9).reshape((3,3)),index = ['a', 'c','d'], columns
#print(states)
#  for your own working, run following statement without ffill
states = states.reindex(['a','b','c','d'], method = 'ffill')
print(states)
```

```
   punjab  sindh  balochistan
a       0      1            2
b       0      1            2
c       3      4            5
d       6      7            8
```

In [16]:
```
# columns names changing using reindex method
```

In [18]:
```
import numpy as np
import pandas as pd

states = pd.DataFrame(np.arange(9).reshape((3,3)),index = ['a', 'c','d'], columns
print(states)

states_name = ['sindh','kpk','punjab','balochistan']
# can we use ffill parameter in column reindex mode ?
states = states.reindex(columns = states_name)
print(states)
```

```
   punjab  sindh  balochistan
a       0      1            2
c       3      4            5
d       6      7            8
   sindh  kpk  punjab  balochistan
a      1  NaN       0            2
c      4  NaN       3            5
d      7  NaN       6            8
```

Deleting data (row or column in dataframe)

In [22]:
```
import numpy as np
import pandas as pd

data_df = pd.DataFrame(np.arange(16).reshape((4,4)), index = ['punjab', 'kpk','ba
                       columns = ['one', 'two','three', 'four'])
print(data_df)
```

```
            one  two  three  four
punjab        0    1      2     3
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15
```

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

data_df = pd.DataFrame(np.arange(16).reshape((4,4)), index = ['punjab', 'kpk','ba
                       columns = ['one', 'two','three', 'four'])
print(data_df, "\n")
#  you can drop values from the columns by passing axis = 1
#  or   , axis = 'columns'

#data_df = data_df.drop('two' , axis=1 )   this method also drop column

data_df.drop('two',axis=1,inplace= True)  # this method also drop column if inpla
                                          #   if inplace = false then back the co

print(data_df)
```

```
            one  two  three  four
punjab        0    1      2     3
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15

            one  three  four
punjab        0      2     3
kpk           4      6     7
balchistan    8     10    11
sindh        12     14    15
```

In [30]:
```
import numpy as np
import pandas as pd

data_df = pd.DataFrame(np.arange(16).reshape((4,4)), index = ['punjab', 'kpk','ba
                       columns = ['one', 'two','three', 'four'])
print(data_df)

# if i want to remove a row for example 'kpk',
# how can we do that

data_df =data_df.drop(['kpk'])   this method remove the row
print(data_df)
```

```
            one  two  three  four
punjab        0    1      2     3
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15
            one  two  three  four
punjab        0    1      2     3
balchistan    8    9     10    11
sindh        12   13     14    15
```

In [40]:
```
data_df
```

Out[40]:

	one	three	four
punjab	0	2	3
kpk	4	6	7
balchistan	8	10	11
sindh	12	14	15

In [41]:
```
data_df.drop('three' , axis=1 )
```

Out[41]:

	one	four
punjab	0	3
kpk	4	7
balchistan	8	11
sindh	12	15

In [42]:
```
data_df
```

Out[42]:

	one	three	four
punjab	0	2	3
kpk	4	6	7
balchistan	8	10	11
sindh	12	14	15

In [46]:
```
data_df.drop('four' , axis=1 , inplace = True)
```

In [47]:
```
data_df
```

Out[47]:

	one
punjab	0
kpk	4
balchistan	8
sindh	12

Indexing , selection and filtering

In [51]:
```
import numpy as np
import pandas as pd

data_df = pd.DataFrame(np.arange(16).reshape((4,4)), index = ['punjab', 'kpk','ba
                      columns = ['one', 'two','three', 'four'])
#print(data_df, "\n")
df2  = data_df[ ["one", "three"] ]
print(df2)
#print( data_df[2:] ) # same like numpy
#print(data_df["one"] )# dicitonary like style of accessing data
#print(data_df.one[2:])  # filter on both row and column
# Conditonal Selection
#print ( data_df.three[data_df['three'] > 5] )
#print(data_df, "\n")

#f2 = data_df["three"] > 5  # boolean dataframe
#print(df2)
#print( data_df[  data_df["three"] > 5  ] )
```

```
             one  three
punjab         0      2
kpk            4      6
balchistan     8     10
sindh         12     14
```

In [58]:
```
import numpy as np
import pandas as pd

data_df = pd.DataFrame(np.arange(16).reshape((4,4)), index = ['punjab', 'kpk','ba
                    columns = ['one', 'two','three', 'four'])

print(data_df, "\n")
#df2  = data_df[ ["one", "three"] ]
#print(df2)
print(data_df[2:])   #  same like numpy
#print(data_df["one"])  # dictionary like style of accessing data
```

```
            one  two  three  four
punjab        0    1      2     3
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15

            one  two  three  four
balchistan    8    9     10    11
sindh        12   13     14    15
```

In [59]:
```
print(data_df["one"])  # dictionary like style of accessing data
```

```
punjab         0
kpk            4
balchistan     8
sindh         12
Name: one, dtype: int32
```

In [69]:
```
print(data_df.one[2:])  # filter on both row and column
print("   ")
# Conditonal Selection
print ( data_df.three[data_df['three'] > 5] )
```

```
balchistan     8
sindh         12
Name: one, dtype: int32

kpk            6
balchistan    10
sindh         14
Name: three, dtype: int32
```

In [68]:
```
print(data_df, "\n")
#f2 = data_df["three"] > 5  # boolean dataframe
#print(df2)
#print( data_df[  data_df["three"] > 5  ] )
```

```
            one  two  three  four
punjab        0    1      2     3
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15
```

In [70]:
```
df2 = data_df["three"] > 5  # boolean dataFrame
print(df2)
```

```
punjab        False
kpk            True
balchistan     True
sindh          True
Name: three, dtype: bool
```

In [71]:
```
print( data_df[  data_df["three"] > 5  ] )
```

```
            one  two  three  four
kpk           4    5      6     7
balchistan    8    9     10    11
sindh        12   13     14    15
```

Selection with lock and iloc

In [72]:
```
import pandas as pd
import numpy as np

data_df = pd.DataFrame(np.arange(16).reshape((4, 4)),
             index=['Ohio', 'Colorado', 'Utah', 'New York'],
             columns=['one', 'two', 'three', 'four'])

print(data_df, "\n")
# in the loc method specify row label first
# then specify column names
# remember! mutiple column names require array notation
#print( data_df.loc[['Colorado','Ohio'], ['two',  'three'] ] )
#print("\n")
#print( data_df.iloc[2:, [3, 0, 1] ] ) # using number instead of labels
#print( data_df.iloc[:])
#print ( data_df.iloc[ :3 , :3 ] )
#print( data_df.iloc[:, :3])
```

```
          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 [74]:
```
print( data_df.loc[['Colorado','Ohio'], ['two',  'three'] ] )
print("\n")
```

```
          two  three
Colorado    5      6
Ohio        1      2
```

In [77]:
```
print( data_df.iloc[2:, [3, 0, 1] ] ) # using number instead of labels
print("   ")
print( data_df.iloc[:])
```

```
          four  one  two
Utah        11    8    9
New York    15   12   13

          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 [78]:
```
print ( data_df.iloc[ :3 , :3 ] )
```

```
          one  two  three
Ohio        0    1      2
Colorado    4    5      6
Utah        8    9     10
```

In [79]:
```
print( data_df.iloc[:, :3])
```

```
          one  two  three
Ohio        0    1      2
Colorado    4    5      6
Utah        8    9     10
New York   12   13     14
```

Arithematics and Data Alignment

In [82]:
```
print(list('bcd'))

df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)),
                   columns=list('bcd'),
                   index=['Ohio', 'Texas', 'Colorado'])

df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                   columns=list('bde'),
                    index=['Utah', 'Ohio', 'Texas', 'Oregon'])

print(df1)
print(df2)
print()
# applying plus operation between two data frames
df3 = df1 + df2

print(df3)
# your work is to fill all Nan values of  this df3 with a number,
# choice of number is yours
```

```
['b', 'c', 'd']
            b    c    d
Ohio      0.0  1.0  2.0
Texas     3.0  4.0  5.0
Colorado  6.0  7.0  8.0
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0

            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

In [80]:
```
print(list('bcd'))
```

```
['b', 'c', 'd']
```

In [85]:
```
print(df1)
print("   ")
print(df2)
```

```
            b    c    d
Ohio      0.0  1.0  2.0
Texas     3.0  4.0  5.0
Colorado  6.0  7.0  8.0

          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0
```

In [86]:
```
# applying plus operation between two data frames
df3 = df1 + df2

print(df3)
# your work is to fill all Nan values of  this df3 with a number,
# choice of number is yours
```

```
            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

Arithematic method with fill value

In [88]:
```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),

                   columns=list('abcd'))
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                   columns=list('abcde'))
print(df1)
#print(df2)

df2.loc[1, 'b'] = np.nan
print(df2)
df3 = df1 + df2
print()
print("direct + operation without fill_value")
#print(df3)
print("--------")
print()
# We can use add method for filling NaN cells with a value
# Nan will be replaced by 0 and then addition operation will apply
print("addition using a method with replacing Nan with 0")
df3 = df1.add(df2, fill_value=0)
print(df3)
```

```
     a    b     c     d
0  0.0  1.0   2.0   3.0
1  4.0  5.0   6.0   7.0
2  8.0  9.0  10.0  11.0
      a     b     c     d     e
0   0.0   1.0   2.0   3.0   4.0
1   5.0   NaN   7.0   8.0   9.0
2  10.0  11.0  12.0  13.0  14.0
3  15.0  16.0  17.0  18.0  19.0

direct + operation without fill_value
--------

addition using a method with replacing Nan with 0
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

In [89]:
```
# We can use add method for filling NaN cells with a value
# Nan will be replaced by 0 and then addition operation will apply

print("addition using a method with replacing Nan with 0")
df3 = df1.add(df2, fill_value=0)
print(df3)
```

```
addition using a method with replacing Nan with 0
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

In [ ]: