Pandas -- Series and Data Frames ¶

```
In [1]: import pandas as pd
        import numpy  as np
```

Series

- a one-dimensional array-like object containing a sequence of values
- associated array of data labels, called its index

```
In [2]: np.random.seed(123)
        scores = np.random.randint(60, 90, 6)

        a = pd.Series(scores)
        a

Out[2]: 0    73
        1    62
        2    88
        3    62
        4    66
        5    77
        dtype: int64
```

```
In [3]: a.values

Out[3]: array([73, 62, 88, 62, 66, 77])
```

```
In [4]: a.index

Out[4]: RangeIndex(start=0, stop=6, step=1)
```

```
In [5]: a[1]

Out[5]: 62
```

```
In [6]: a[[1, 4]]

Out[6]: 1    62
        4    66
        dtype: int64
```

```
In [7]: a[::-2]

Out[7]: 5    77
        3    62
        1    62
        dtype: int64
```

In [8]:

```
b = pd.Series(scores, index = ['Alice', 'Bob', 'Charlie', 'Dave', 'Ed', 'Fred'])
b
```

Out[8]:

```
Alice      73
Bob        62
Charlie    88
Dave       62
Ed         66
Fred       77
dtype: int64
```

In [9]:

```
b['Bob']
```

Out[9]:

```
62
```

In [10]:

```
b[['Bob', 'Ed']]
```

Out[10]:

```
Bob    62
Ed     66
dtype: int64
```

In [11]:

```
b[::-2]
```

Out[11]:

```
Fred    77
Dave    62
Bob     62
dtype: int64
```

In [12]:

```
b[b > 70]
```

Out[12]:

```
Alice      73
Charlie    88
Fred       77
dtype: int64
```

In [13]:

```
b + 10
```

Out[13]:

```
Alice      83
Bob        72
Charlie    98
Dave       72
Ed         76
Fred       87
dtype: int64
```

In [14]:

```
b
```

Out[14]:

```
Alice      73
Bob        62
Charlie    88
Dave       62
Ed         66
Fred       77
dtype: int64
```

In [15]:
```
np.cumsum(b)
```

Out[15]:
```
Alice       73
Bob        135
Charlie    223
Dave       285
Ed         351
Fred       428
dtype: int64
```

In [16]:
```
np.average(b)
```

Out[16]:
```
71.33333333333333
```

In [17]:
```
b.describe()
```

Out[17]:
```
count     6.000000
mean     71.333333
std      10.152175
min      62.000000
25%      63.000000
50%      69.500000
75%      76.000000
max      88.000000
dtype: float64
```

In [18]:
```
'Charlie' in b
```

Out[18]:
```
True
```

In [19]:
```
'Robert' in b
```

Out[19]:
```
False
```

In [20]:
```
b.index.name = 'FirstName'
b
```

Out[20]:
```
FirstName
Alice      73
Bob        62
Charlie    88
Dave       62
Ed         66
Fred       77
dtype: int64
```

Series from dictionary data

In [21]:
```
c = pd.Series({'R': 60, 'Python': 75, 'Java': 50})
c
```

Out[21]:
```
R         60
Python    75
Java      50
dtype: int64
```

```
In [22]: d = pd.Series({'R': 60, 'Python': 75, 'Java': 50},
                       index=['Java', 'Python', 'R', 'C++'])
         d
```

```
Out[22]: Java      50.0
         Python    75.0
         R         60.0
         C++        NaN
         dtype: float64
```

```
In [23]: pd.isnull(d)
```

```
Out[23]: Java      False
         Python    False
         R         False
         C++        True
         dtype: bool
```

```
In [24]: pd.notnull(d)
```

```
Out[24]: Java       True
         Python     True
         R          True
         C++       False
         dtype: bool
```

```
In [25]: c + d
```

```
Out[25]: C++         NaN
         Java      100.0
         Python    150.0
         R         120.0
         dtype: float64
```

```
In [26]: (c + d).dropna()
```

```
Out[26]: Java      100.0
         Python    150.0
         R         120.0
         dtype: float64
```

DataFrame

- represents a rectangular table of data
- contains an ordered collection of columns
- each column can be a different value type
- has both a row and column index

In [27]:

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}

df1 = pd.DataFrame(data)
df1
```

Out[27]:

	state	year	pop
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

In [28]:

```
df1 = pd.DataFrame(data, columns = ['year', 'state', 'pop'])
df1
```

Out[28]:

	year	state	pop
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 [29]:

```
df1.head()
```

Out[29]:

	year	state	pop
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

In [30]:

```
df1.tail(n=3)
```

Out[30]:

	year	state	pop
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

In [31]:

```
df2 = pd.DataFrame(data, columns = ['year', 'state', 'pop', 'debt'])
df2
```

Out[31]:

	year	state	pop	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 [32]:

```
df2.columns
```

Out[32]:

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

In [33]:

```
df2.index
```

Out[33]:

```
RangeIndex(start=0, stop=6, step=1)
```

Retrieve columns

In [34]:

```
df2['year']
```

Out[34]:

```
0    2000
1    2001
2    2002
3    2001
4    2002
5    2003
Name: year, dtype: int64
```

In [35]:

```
df2.year
```

Out[35]:

```
0    2000
1    2001
2    2002
3    2001
4    2002
5    2003
Name: year, dtype: int64
```

In [36]:

```
df2[['year', 'state']]
```

Out[36]:

	year	state
0	2000	Ohio
1	2001	Ohio
2	2002	Ohio
3	2001	Nevada
4	2002	Nevada
5	2003	Nevada

Retrieve rows

In [37]:
```
df2 = pd.DataFrame(data, columns = ['year', 'state', 'pop', 'debt'])
df2
```

Out[37]:

	year	state	pop	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 [38]:
```
df2.iloc[2]
```

Out[38]:
```
year      2002
state     Ohio
pop        3.6
debt       NaN
Name: 2, dtype: object
```

In [39]:
```
type(df2.iloc[2])
```

Out[39]:
```
pandas.core.series.Series
```

In [40]:
```
df2.iloc[[2]]
```

Out[40]:

	year	state	pop	debt
2	2002	Ohio	3.6	NaN

In [41]:
```
type(df2.iloc[[2]])
```

Out[41]:
```
pandas.core.frame.DataFrame
```

In [42]:
```
df2.iloc[[2,5]]
```

Out[42]:

	year	state	pop	debt
2	2002	Ohio	3.6	NaN
5	2003	Nevada	3.2	NaN

In [43]:

```
df2.index = ['one', 'two', 'three', 'four', 'five', 'six']
df2
```

Out[43]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

In [44]:

```
df2.loc['two']
```

Out[44]:

```
year      2001
state     Ohio
pop        1.7
debt       NaN
Name: two, dtype: object
```

In [45]:

```
df2.loc[['two','five']]
```

Out[45]:

	year	state	pop	debt
two	2001	Ohio	1.7	NaN
five	2002	Nevada	2.9	NaN

In [ ]:

In [46]:

```
df2['debt'] = 20.5
df2
```

Out[46]:

	year	state	pop	debt
one	2000	Ohio	1.5	20.5
two	2001	Ohio	1.7	20.5
three	2002	Ohio	3.6	20.5
four	2001	Nevada	2.4	20.5
five	2002	Nevada	2.9	20.5
six	2003	Nevada	3.2	20.5

In [47]:

```
df2['debt'] = np.arange(df2.shape[0])
df2
```

Out[47]:

	year	state	pop	debt
one	2000	Ohio	1.5	0
two	2001	Ohio	1.7	1
three	2002	Ohio	3.6	2
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	4
six	2003	Nevada	3.2	5

In [48]:

```
# adding a column

df2['east'] = df2.state == 'Ohio'
df2
```

Out[48]:

	year	state	pop	debt	east
one	2000	Ohio	1.5	0	True
two	2001	Ohio	1.7	1	True
three	2002	Ohio	3.6	2	True
four	2001	Nevada	2.4	3	False
five	2002	Nevada	2.9	4	False
six	2003	Nevada	3.2	5	False

In [49]:

```
# deleting a column

del df2['east']
df2
```

Out[49]:

	year	state	pop	debt
one	2000	Ohio	1.5	0
two	2001	Ohio	1.7	1
three	2002	Ohio	3.6	2
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	4
six	2003	Nevada	3.2	5

In [50]:

```
df2.T
```

Out[50]:

	one	two	three	four	five	six
year	2000	2001	2002	2001	2002	2003
state	Ohio	Ohio	Ohio	Nevada	Nevada	Nevada
pop	1.5	1.7	3.6	2.4	2.9	3.2
debt	0	1	2	3	4	5

In [51]:

```
# nested dictionaries

pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

df3 = pd.DataFrame(pop)
df3
```

Out[51]:

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Reindexing

In [52]:

```
df1 = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
df1
```

Out[52]:

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

In [53]:

```
df2 = df1.reindex(['a', 'b', 'c', 'd', 'e'])
df2
```

Out[53]:

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

In [54]:

```
df3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
df3
```

Out[54]:

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

In [55]:

```
df3.reindex(np.arange(6))
```

Out[55]:

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

```
In [56]: # forward fill missing values

         df3.reindex(np.arange(6), method='ffill')
```

```
Out[56]: 0       blue
         1       blue
         2     purple
         3     purple
         4     yellow
         5     yellow
         dtype: object
```

```
In [57]: # backward fill missing values

         df3.reindex(np.arange(6), method='bfill')
```

```
Out[57]: 0       blue
         1     purple
         2     purple
         3     yellow
         4     yellow
         5        NaN
         dtype: object
```

```
In [58]: df4 = pd.DataFrame(np.arange(9).reshape((3, 3)),
                            index=['a', 'c', 'd'],
                            columns=['Ohio', 'Texas', 'California'])
         df4
```

Out[58]:

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```
In [59]: df4.reindex(['a', 'b', 'c', 'd'])
```

Out[59]:

	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

```
In [60]: # for reindexing columns

         df4.reindex(columns = ['Utah', 'Ohio', 'Texas'])
```

Out[60]:

	Utah	Ohio	Texas
a	NaN	0	1
c	NaN	3	4
d	NaN	6	7

Dropping entries from an Axis

In [61]:

```
# For Series

df1 = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])
df1
```

Out[61]:

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

In [62]:

```
df1.drop('b')
```

Out[62]:

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

In [63]:

```
df1.drop(['a', 'c'])
```

Out[63]:

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

In [64]:

```
# For Data Frame

df2 = pd.DataFrame(np.arange(16).reshape((4, 4)),
                   index=['Ohio', 'Colorado', 'Utah', 'New York'],
                   columns=['one', 'two', 'three', 'four'])
df2
```

Out[64]:

	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 [65]:

```
# Default axis is rows (0)
df2.drop('Ohio')
```

Out[65]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

In [66]:

```
df2.drop(['Colorado', 'Ohio'])
```

Out[66]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

In [67]:

```
# From dropping columns

df2.drop('two', axis='columns')
```

Out[67]:

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

In [68]:

```
df2.drop(['two', 'four'], axis=1)
```

Out[68]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

In [69]:

```
df2.drop(['two', 'four'], axis=1, inplace = True)
df2
```

Out[69]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

Indexing, Selection, and Filtering

In [70]:

```
df1 = pd.Series(np.arange(10,14), index=['a', 'b', 'c', 'd'])
df1
```

Out[70]:

```
a    10
b    11
c    12
d    13
dtype: int64
```

In [71]:

```
df1['c']
```

Out[71]:

```
12
```

In [72]:

```
df1[2]
```

Out[72]:

```
12
```

In [73]:
```
df1[1:3]
```

Out[73]:
```
b    11
c    12
dtype: int64
```

In [74]:
```
# inclusive end-point

df1['b':'d']
```

Out[74]:
```
b    11
c    12
d    13
dtype: int64
```

In [75]:
```
df1[[3,1]]
```

Out[75]:
```
d    13
b    11
dtype: int64
```

In [76]:
```
df1[['d', 'b']]
```

Out[76]:
```
d    13
b    11
dtype: int64
```

In [77]:
```
df1[df1 < 12]
```

Out[77]:
```
a    10
b    11
dtype: int64
```

In [78]:
```
df1['b':'d'] = 50
df1
```

Out[78]:
```
a    10
b    50
c    50
d    50
dtype: int64
```

In [79]:
```
# For Data Frame

df2 = pd.DataFrame(np.arange(16).reshape((4, 4)),
                   index=['Ohio', 'Colorado', 'Utah', 'New York'],
                   columns=['one', 'two', 'three', 'four'])
df2
```

Out[79]:

	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 [80]:

```
df2['two']
```

Out[80]:

```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

In [81]:

```
df2[['two', 'one']]
```

Out[81]:

	two	one
Ohio	1	0
Colorado	5	4
Utah	9	8
New York	13	12

In [82]:

```
# Special cases

df2[:2]
```

Out[82]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

In [83]:

```
df2['three'] < 10
```

Out[83]:

```
Ohio         True
Colorado     True
Utah        False
New York    False
Name: three, dtype: bool
```

In [84]:

```
df2[df2['three'] < 10]
```

Out[84]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

In [85]:

```
df2
```

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

In [86]:

```
df2[df2 < 10] = -1
df2
```

Out[86]:

	one	two	three	four
Ohio	-1	-1	-1	-1
Colorado	-1	-1	-1	-1
Utah	-1	-1	10	11
New York	12	13	14	15

Selecting with loc and iloc

- for DataFrame label-indexing on the rows
- loc (using axis labels)
- iloc (using integer index)

In [87]:

```
df2 = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
df2
```

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]:

```
df2.loc['Colorado']
```

Out[88]:

```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

In [89]:

```
df2.loc['Colorado', ['two', 'four']]
```

Out[89]:

```
two     5
four    7
Name: Colorado, dtype: int64
```

In [90]:

```
df2.iloc[1]
```

Out[90]:

```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

In [91]:

```
df2.iloc[1, [1, 3]]
```

Out[91]:

```
two     5
four    7
Name: Colorado, dtype: int64
```

In [92]:

```
df2.iloc[[1, 2]]
```

Out[92]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11

In [93]:

```
df2.iloc[[1, 2], [1, 3]]
```

Out[93]:

	two	four
Colorado	5	7
Utah	9	11

In [94]:

```
df2.loc[:'Utah']
```

Out[94]:

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

In [95]:

```
df2.loc[:'Utah', ['two', 'three']]
```

Out[95]:

	two	three
Ohio	1	2
Colorado	5	6
Utah	9	10

In [96]:

```
df2.iloc[:, :3]
```

Out[96]:

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

In [97]:

```
df2.iloc[:, :3][df2.three > 5]
```

Out[97]:

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

Function application and mapping

In [98]:

```
df1 = pd.DataFrame(np.random.randn(4, 3), columns=list('abc'),
                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])
df1
```

Out[98]:

	a	b	c
Utah	-0.578600	1.651437	-2.426679
Ohio	-0.428913	1.265936	-0.866740
Texas	-0.678886	-0.094709	1.491390
Oregon	-0.638902	-0.443982	-0.434351

In [99]:

```
np.abs(df1)
```

Out[99]:

	a	b	c
Utah	0.578600	1.651437	2.426679
Ohio	0.428913	1.265936	0.866740
Texas	0.678886	0.094709	1.491390
Oregon	0.638902	0.443982	0.434351

In [100]:

```
# apply a function on 1-D arrays to each column or row
```

In [101]:

```
# default axis = 'rows'

df1.apply(lambda x: x.max() - x.min())
```

Out[101]:

```
a    0.249974
b    2.095418
c    3.918069
dtype: float64
```

In [102]:

```
# invoke once per row

df1.apply(lambda x: x.max() - x.min(), axis = 'columns')
```

Out[102]:

```
Utah      4.078116
Ohio      2.132677
Texas     2.170276
Oregon    0.204551
dtype: float64
```

In [103]:

```
df1
```

Out[103]:

	a	b	c
Utah	-0.578600	1.651437	-2.426679
Ohio	-0.428913	1.265936	-0.866740
Texas	-0.678886	-0.094709	1.491390
Oregon	-0.638902	-0.443982	-0.434351

In [104]:

```
# function returning multiple values

df1.apply(lambda x: pd.Series([x.min(), x.max()], index = ['min', 'max']))
```

Out[104]:

	a	b	c
min	-0.678886	-0.443982	-2.426679
max	-0.428913	1.651437	1.491390

In [105]:

```
df1.apply(lambda x: pd.Series([x.min(), x.max()], index = ['min', 'max']),
          axis='columns')
```

Out[105]:

	min	max
Utah	-2.426679	1.651437
Ohio	-0.866740	1.265936
Texas	-0.678886	1.491390
Oregon	-0.638902	-0.434351

Sorting

- sort lexicographically by row or column index

In [106]:

```
# Series

df1 = pd.Series(np.arange(10,14), index=['d', 'a', 'b', 'c'])
df1
```

Out[106]:

```
d    10
a    11
b    12
c    13
dtype: int64
```

In [107]:

```
df2 = df1.sort_index()
df2
```

Out[107]:

```
a    11
b    12
c    13
d    10
dtype: int64
```

In [108]:

```
df2.sort_values()
```

Out[108]:

```
d    10
a    11
b    12
c    13
dtype: int64
```

In [109]:

```
# DataFrame

df1 = pd.DataFrame(np.arange(8).reshape((2, 4)),
                   index=['three', 'one'],
                   columns=['d', 'a', 'b', 'c'])
df1
```

Out[109]:

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

In [110]:

```
df1.sort_index()
```

Out[110]:

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

In [111]:

```
df1.sort_index(axis=1)
```

Out[111]:

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

In [112]:

```
df1
```

Out[112]:

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

In [113]:

```
df1.sort_values(by ='b', ascending = False)
```

Out[113]:

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

In [114]:

```
df1.sort_values(by ='one', axis = 1, ascending = False)
```

Out[114]:

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

Axis indices with duplicate labels

In [115]:

```
# Series

df1 = pd.Series(np.arange(10,15), index=['a', 'a', 'b', 'b', 'c'])
df1
```

Out[115]:

```
a    10
a    11
b    12
b    13
c    14
dtype: int64
```

In [116]:

```
df1['b']
```

Out[116]:

```
b    12
b    13
dtype: int64
```

In [117]:

```
df1.index.is_unique
```

Out[117]:

```
False
```

In [118]:

```
# DataFrame

df2 = pd.DataFrame(np.random.randint(60, 90, (4, 3)), index=['a', 'a', 'b', 'b'])
df2
```

Out[118]:

	0	1	2
a	67	62	80
a	75	84	89
b	76	67	69
b	63	88	88

In [119]:

```
df2.loc['b']
```

Out[119]:

	0	1	2
b	76	67	69
b	63	88	88

Descriptive Statistics

In [120]:

```
df1 = pd.DataFrame([[1.5, np.nan], [7.5, -5.5],
                     [np.nan, np.nan], [1.0, -4.5]],
                    index=['a', 'b', 'c', 'd'],
                    columns=['one', 'two'])

df1
```

Out[120]:

	one	two
a	1.5	NaN
b	7.5	-5.5
c	NaN	NaN
d	1.0	-4.5

```
In [121]: df1.sum()
```

```
Out[121]: one     10.0
          two    -10.0
          dtype: float64
```

```
In [122]: df1.sum(axis=0)
```

```
Out[122]: one     10.0
          two    -10.0
          dtype: float64
```

```
In [123]: df1.sum(axis='rows')
```

```
Out[123]: one     10.0
          two    -10.0
          dtype: float64
```

```
In [124]: df1.sum(axis=1)
```

```
Out[124]: a    1.5
          b    2.0
          c    0.0
          d   -3.5
          dtype: float64
```

```
In [125]: df1.sum(axis='columns')
```

```
Out[125]: a    1.5
          b    2.0
          c    0.0
          d   -3.5
          dtype: float64
```

```
In [126]: df1
```

Out[126]:

	one	two
a	1.5	NaN
b	7.5	-5.5
c	NaN	NaN
d	1.0	-4.5

idxmax, idxmin

- index labels of maximum and minimum values

argmax, argmin (Series)

- index locations of maximum and minimum values for a Series

```
In [127]: print(df1)

          df1.idxmax()
```

```
   one  two
a  1.5  NaN
b  7.5 -5.5
c  NaN  NaN
d  1.0 -4.5
```

```
Out[127]: one     b
          two     d
          dtype: object
```

```
In [128]: df1.idxmax(axis='columns')
```

```
Out[128]: a     one
          b     one
          c     NaN
          d     one
          dtype: object
```

accumulations

- cumsum, cumprod, cummin, cummax

```
In [129]: print(df1)

          df1.cumsum()
```

```
   one  two
a  1.5  NaN
b  7.5 -5.5
c  NaN  NaN
d  1.0 -4.5
```

Out[129]:

	one	two
a	1.5	NaN
b	9.0	-5.5
c	NaN	NaN
d	10.0	-10.0

In [130]:

```
df1.describe()
```

Out[130]:

	one	two
count	3.000000	2.000000
mean	3.333333	-5.000000
std	3.617089	0.707107
min	1.000000	-5.500000
25%	1.250000	-5.250000
50%	1.500000	-5.000000
75%	4.500000	-4.750000
max	7.500000	-4.500000

In [131]:

```
df1
```

Out[131]:

	one	two
a	1.5	NaN
b	7.5	-5.5
c	NaN	NaN
d	1.0	-4.5

In [132]:

```
np.random.seed(123)
df2 = pd.DataFrame(np.random.randint(60, 90, (4, 3)),
                                 index=['a', 'b', 'c', 'd'],
                                columns = ['one', 'two', 'three'])
df2
```

Out[132]:

	one	two	three
a	73	62	88
b	62	66	77
c	79	70	87
d	85	82	61

In [133]:

```
df2.diff()
```

Out[133]:

	one	two	three
a	NaN	NaN	NaN
b	-11.0	4.0	-11.0
c	17.0	4.0	10.0
d	6.0	12.0	-26.0

```
In [134]: df2.diff(axis='columns')
```

Out[134]:

	one	two	three
a	NaN	-11.0	26.0
b	NaN	4.0	11.0
c	NaN	-9.0	17.0
d	NaN	-3.0	-21.0

```
In [135]: df2
```

Out[135]:

	one	two	three
a	73	62	88
b	62	66	77
c	79	70	87
d	85	82	61

```
In [136]: df2.pct_change()
```

Out[136]:

	one	two	three
a	NaN	NaN	NaN
b	-0.150685	0.064516	-0.125000
c	0.274194	0.060606	0.129870
d	0.075949	0.171429	-0.298851

```
In [137]: df2['one'].cov(df2['two'])

Out[137]: 62.666666666666664
```

```
In [138]: df2['one'].corr(df2['two'])

Out[138]: 0.7392185280134137
```

```
In [139]: df2.cov()
```

Out[139]:

	one	two	three
one	96.250000	62.666667	-46.916667
two	62.666667	74.666667	-93.333333
three	-46.916667	-93.333333	156.916667

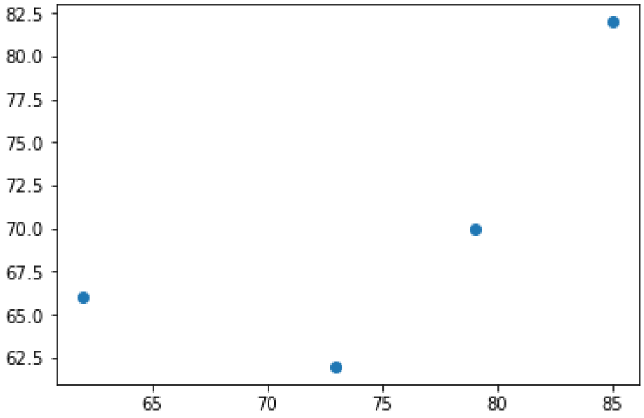
```
In [140]: df2.corr()
```

Out[140]:

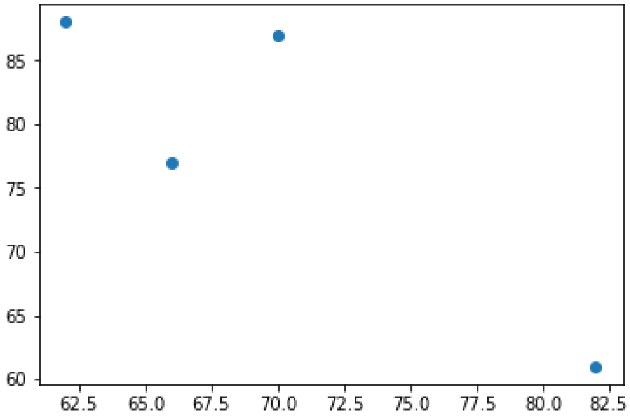
	one	two	three
one	1.000000	0.739219	-0.381762
two	0.739219	1.000000	-0.862261
three	-0.381762	-0.862261	1.000000

```
In [141]: import matplotlib.pyplot as plt
```

```
In [142]: plt.scatter(df2['one'], df2['two']);
```



```
In [143]: plt.scatter(df2['two'], df2['three']);
```



Unique values and value counts

```
In [144]: np.random.seed(123)
          scores = np.random.randint(60, 70, 10)

          a = pd.Series(scores)
          a
```

```
Out[144]: 0    62
          1    62
          2    66
          3    61
          4    63
          5    69
          6    66
          7    61
          8    60
          9    61
          dtype: int64
```

```
In [145]: a.unique()
```

```
Out[145]: array([62, 66, 61, 63, 69, 60])
```

In [146]:

```
a.value_counts()
```

Out[146]:

```
61    3
62    2
66    2
63    1
60    1
69    1
dtype: int64
```

In [147]:

```
a.values
```

Out[147]:

```
array([62, 62, 66, 61, 63, 69, 66, 61, 60, 61])
```

In [148]:

```
pd.value_counts(a)
```

Out[148]:

```
61    3
62    2
66    2
63    1
60    1
69    1
dtype: int64
```

In [149]:

```
pd.value_counts(a.values)
```

Out[149]:

```
61    3
62    2
66    2
63    1
60    1
69    1
dtype: int64
```

In [150]:

```
pd.value_counts(a.values, sort=False)
```

Out[150]:

```
66    2
69    1
60    1
61    3
62    2
63    1
dtype: int64
```

In [151]:

```
a.unique()
```

Out[151]:

```
array([62, 66, 61, 63, 69, 60])
```

In [152]:

```
pd.Index(a.unique()).get_indexer(a)
```

Out[152]:

```
array([0, 0, 1, 2, 3, 4, 1, 2, 5, 2])
```

In [153]:

```
np.random.seed(321)
df2 = pd.DataFrame(np.random.randint(60, 70, (10, 4)),
                        columns = ['Q1', 'Q2', 'Q3', 'Q4'])
df2
```

Out[153]:

	Q1	Q2	Q3	Q4
0	64	69	68	61
1	68	68	64	65
2	68	63	65	61
3	64	66	65	67
4	67	62	62	63
5	69	62	61	62
6	61	61	60	64
7	64	63	60	63
8	67	64	65	67
9	60	68	67	61

In [154]:

```
df2.apply(pd.value_counts)
```

Out[154]:

	Q1	Q2	Q3	Q4
60	1.0	NaN	2.0	NaN
61	1.0	1.0	1.0	3.0
62	NaN	2.0	1.0	1.0
63	NaN	2.0	NaN	2.0
64	3.0	1.0	1.0	1.0
65	NaN	NaN	3.0	1.0
66	NaN	1.0	NaN	NaN
67	2.0	NaN	1.0	2.0
68	2.0	2.0	1.0	NaN
69	1.0	1.0	NaN	NaN

In [155]:

```
df2.apply(pd.value_counts).dropna()
```

Out[155]:

	Q1	Q2	Q3	Q4
61	1.0	1.0	1.0	3.0
64	3.0	1.0	1.0	1.0

```
In [156]: df2.apply(pd.value_counts).fillna(0)
```

Out[156]:

	Q1	Q2	Q3	Q4
60	1.0	0.0	2.0	0.0
61	1.0	1.0	1.0	3.0
62	0.0	2.0	1.0	1.0
63	0.0	2.0	0.0	2.0
64	3.0	1.0	1.0	1.0
65	0.0	0.0	3.0	1.0
66	0.0	1.0	0.0	0.0
67	2.0	0.0	1.0	2.0
68	2.0	2.0	1.0	0.0
69	1.0	1.0	0.0	0.0

```
In [ ]:
```