pandas-Dataframe

In [2]:
```
import numpy as np
import pandas as pd
```

In [12]:
```
data = {'name':['jack','tom','marry'],
      'age':[18,19,20],
      'gender':['m','m','w']}

frame = pd.DataFrame(data)
print(frame)
print('-'*30)
print(frame.index)
print(frame.columns)
print('-'*30)
print(frame.values)
# 查看数据，数据类型为dataframe
# .index查看行标签
# .columns查看列标签
# .values查看值，数据类型为ndarray
```

```
    name  age gender
0   jack   18      m
1    tom   19      m
2  marry   20      w
------------------------------
RangeIndex(start=0, stop=3, step=1)
Index(['name', 'age', 'gender'], dtype='object')
------------------------------
[['jack' 18 'm']
 ['tom' 19 'm']
 ['marry' 20 'w']]
```

In [28]:
```
# Dataframe 创建方法一：由数组/list组成的字典
# 创建方法:pandas.Dataframe()

data1 = {'a':[1,2,3],
       'b':[4,5,6],
       'c':[7,8,9]}

data2 = {'one':np.random.rand(3)*10,
       'two':np.random.rand(3)}

frame = pd.DataFrame(data1)
print(frame)
df2 = pd.DataFrame(data2)
print(df2)
# 由数组/list组成的字典 创建Dataframe，columns为字典key，index为默认数字标签
# 字典的值的长度必须保持一致！


df1 = pd.DataFrame(data1, columns = ['b','c','a','d']) # 进行排序的顺序
print(df1)
print('-'*30)
df1 = pd.DataFrame(data1, columns = ['b','c'])
```

```
print(df1)
# columns参数：可以重新指定列的顺序，格式为list，如果现有数据中没有该列（比如'd'），则产生NaN值
# 如果columns重新指定时候，列的数量可以少于原数据
print('-'*30)
df2 = pd.DataFrame(data2, index = ['f1','f2','f3'])  # 这里如果尝试  index = ['f1','f2','f3','f4'] 会怎么样？
print(df2)
# index参数：重新定义index，格式为list，长度必须保持一致
```

```
   a  b  c
0  1  4  7
1  2  5  8
2  3  6  9
        one       two
0  8.359706  0.883102
1  8.525053  0.349763
2  6.402519  0.586262
   b  c  a    d
0  4  7  1  NaN
1  5  8  2  NaN
2  6  9  3  NaN
------------------------------
   b  c
0  4  7
1  5  8
2  6  9
------------------------------
         one       two
f1  8.359706  0.883102
f2  8.525053  0.349763
f3  6.402519  0.586262
```

In [35]:
```
# Dataframe 创建方法二：由Series组成的字典

data1 = {'one':pd.Series(np.random.rand(2)),
        'two':pd.Series(np.random.rand(3))}  # 没有设置index的Series

df1 = pd.DataFrame(data1)
print(df1)

data2 = {'one':pd.Series(np.random.rand(2), index = ['a','b']),
        'two':pd.Series(np.random.rand(3),index = ['a','b','c'])}  # 设置了index的Series
print('-'*30)
df2 = pd.DataFrame(data2)
print(df2)

# 由Seris组成的字典 创建Dataframe，columns为字典key，index为Series的标签（如果Series没有指定标签，则是默认数字标签）
# Series可以长度不一样，生成的Dataframe会出现NaN值
```

```
        one       two
0  0.822240  0.142140
1  0.055269  0.227005
2       NaN  0.562796
------------------------------
        one       two
a  0.730579  0.059789
b  0.106293  0.820228
c       NaN  0.977859
```

In [43]:
```
# Dataframe 创建方法三：通过二维数组直接创建
```

```
ar = np.random.rand(9).reshape(3,3)
print(ar)

df1 = pd.DataFrame(ar)
print('-'*30)
print(df1)
df1 = pd.DataFrame(ar,index = ['a', 'b', 'c'], columns = ['one','two','three'])
print('-'*30)
print(df1)
# 通过二维数组直接创建Dataframe，得到一样形状的结果数据，如果不指定index和columns，两者均返回默认数字格式
# index和colunms指定长度与原数组保持一致
```

```
[[0.1433563  0.85255645 0.65786729]
 [0.84964587 0.49705117 0.24952551]
 [0.91708589 0.41655625 0.9772038 ]]
------------------------------
          0         1         2
0  0.143356  0.852556  0.657867
1  0.849646  0.497051  0.249526
2  0.917086  0.416556  0.977204
------------------------------
        one       two     three
a  0.143356  0.852556  0.657867
b  0.849646  0.497051  0.249526
c  0.917086  0.416556  0.977204
```

In [53]:

```
# Dataframe 创建方法四：由字典组成的列表

data = [{'one': 1, 'two': 2}, {'one': 5, 'two': 10, 'three': 20}]
df1 = pd.DataFrame(data)
print(df1)
df2 = pd.DataFrame(data, index=['a','b'])
df3 = pd.DataFrame(data, columns=['one','two','three'])
print('*'*30)
print(df2)
print('*'*30)
print(df3)
# 由字典组成的列表创建Dataframe，columns为字典的key，index不做指定则为默认数组标签
# colunms和index参数分别重新指定相应列及行标签
```

```
   one  two  three
0    1    2    NaN
1    5   10   20.0
******************************
   one  two  three
a    1    2    NaN
b    5   10   20.0
******************************
   one  two  three
0    1    2    NaN
1    5   10   20.0
```

In [56]:

```
# Dataframe 创建方法五：由字典组成的字典

data = {'Jack':{'math':90,'english':89,'art':78},
       'Marry':{'math':82,'english':95,'art':92},
       'Tom':{'math':78,'english':67}}
df1 = pd.DataFrame(data)
```

```
print('*'*30)
print(df1)
# 由字典组成的字典创建Dataframe，columns为字典的key，index为子字典的key

df2 = pd.DataFrame(data, columns = ['Jack','Tom','Bob'])
df3 = pd.DataFrame(data, index = ['a','b','c'])
print('*'*30)
print(df2)
print('*'*30)
print(df3)

# columns参数可以增加和减少现有列，如出现新的列，值为NaN
# index在这里和之前不同，并不能改变原有index，如果指向新的标签，值为NaN （非常重要！）
```

```
******************************
         Jack  Marry   Tom
math       90     82  78.0
english    89     95  67.0
art        78     92   NaN
******************************
         Jack   Tom  Bob
math       90  78.0  NaN
english    89  67.0  NaN
art        78   NaN  NaN
******************************
   Jack  Marry  Tom
a   NaN    NaN  NaN
b   NaN    NaN  NaN
c   NaN    NaN  NaN
```

Pandas数据结构Dataframe：索引

Dataframe既有行索引也有列索引，可以被看做由Series组成的字典（共用一个索引）

选择列 / 选择行 / 切片 / 布尔判断

In [71]:
```
#选择行与列

df = pd.DataFrame(np.random.rand(12).reshape(3,4)*100,
                  index = ['one','two','three'],
                  columns = ['a','b','c','d'])

print(df)

data1 = df['a']
print('*'*30)
print(data1)
#选择多个，但是可以不连续的
data2 = df[['a','c']]
print('*'*30)
print(data2)
# 按照列名选择列，只选择一列输出Series，选择多列输出Dataframe

data3 = df.loc['one']
print('*'*30)
print(data3)

data4 = df.loc[['one','two']]
print(data4)
```

```
# loc->location
# 按照index选择行，只选择一行输出Series，选择多行输出Dataframe
```

```
               a          b          c          d
one    91.439561  40.723045  85.234958  47.866584
two    12.716526  12.255741  90.906728  97.568355
three  59.562995  29.086188  92.682476  62.330247
******************************
one      91.439561
two      12.716526
three    59.562995
Name: a, dtype: float64
******************************
               a          c
one    91.439561  85.234958
two    12.716526  90.906728
three  59.562995  92.682476
******************************
a    91.439561
b    40.723045
c    85.234958
d    47.866584
Name: one, dtype: float64
             a          b          c          d
one  91.439561  40.723045  85.234958  47.866584
two  12.716526  12.255741  90.906728  97.568355
```

In [77]:

```
# df[] - 选择列
# 一般用于选择列，也可以选择行

df = pd.DataFrame(np.random.rand(12).reshape(3,4)*100,
                  index = ['one','two','three'],
                  columns = ['a','b','c','d'])

data1 = df['a']
data2 = df[['b','c']]
print(data1)
print('*'*30)
print(data2)

# df[]默认选择列，[]中写列名（所以一般数据colunms都会单独制定，不会用默认数字列名，以免和
index冲突）
# 单选列为Series，print结果为Series格式
# 多选列为Dataframe，print结果为Dataframe格式

data3 = df[:1] # 选择行
# data3 = df['one'] 这是不行的
data3
```

```
one      26.473206
two      17.507101
three    29.139300
Name: a, dtype: float64
******************************
               b          c
one    67.265092  35.822231
two    79.662735  22.102997
three  71.590499  78.895695
```

Out[77]:

	a	b	c	d

```
one   26.473206   67.265092   35.822231   99.416607
```

In [92]:
```
df1 = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   index = ['one','two','three','four'],
                   columns = ['a','b','c','d'])

df2 = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   columns = ['a','b','c','d'])

data1 = df1.loc['one']
data2 = df2.loc[1]
print(data1)
print('*'*30)
print(data2)

# 单个标签索引，返回Series
data3 = df1.loc[['two','three','four']]
data4 = df2.loc[[3,2,1]]
print('*'*30)
print(data3)
print('*'*30)
print(data4)
# 顺序可变


data5 = df1.loc['one':'three']
data6 = df2.loc[1:3]
print('*'*30)
print(data5)
print('*'*30)
print(data6)
# 可以做切片对象
# 末端包含

# 核心笔记：df.loc[label]主要针对index选择行，同时支持指定index，及默认数字index
```

```
a    33.751278
b    40.184099
c    51.829051
d    95.905189
Name: one, dtype: float64
******************************
a    29.035088
b    79.715775
c    50.964033
d    12.604729
Name: 1, dtype: float64
******************************
               a          b          c          d
two    87.799284  50.757090  40.120530  59.808117
three  36.927921  10.501423  19.039488  96.755305
four   47.615627  81.047672  99.798638  70.743673
******************************
           a          b          c          d
3  22.337857  45.546321  71.705527  82.887664
2  98.806837  37.886307   9.644451  18.052237
1  29.035088  79.715775  50.964033  12.604729
******************************
               a          b          c          d
one    33.751278  40.184099  51.829051  95.905189
```

```
two    87.799284  50.757090  40.120530  59.808117
three  36.927921  10.501423  19.039488  96.755305
******************************
          a          b          c          d
1  29.035088  79.715775  50.964033  12.604729
2  98.806837  37.886307   9.644451  18.052237
3  22.337857  45.546321  71.705527  82.887664
```

In [99]:
```
# df.iloc[] - 按照整数位置（从轴的0到length-1）选择行
# 类似list的索引，其顺序就是dataframe的整数位置，从0开始计

df = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                  index = ['one','two','three','four'],
                  columns = ['a','b','c','d'])

print(df)

print(df.iloc[0])
print(df.iloc[-1])

# 单位置索引
# 和loc索引不同，不能索引超出数据行数的整数位置

print(df.iloc[[0,2]])
print(df.iloc[[3,2,1]])
print('多位置索引\n-----')
# 多位置索引
# 顺序可变

print(df.iloc[1:3])
print(df.iloc[::2])
print('切片索引')
# 切片索引
# 末端不包含
```

```
              a          b          c          d
one    33.594644  14.151517  81.779949  57.028864
two    26.914832  11.108404  48.340908  86.074874
three  94.723426   5.063484  62.065909  55.522443
four   95.153937  70.695947  31.834829  72.034005
a    33.594644
b    14.151517
c    81.779949
d    57.028864
Name: one, dtype: float64
a    95.153937
b    70.695947
c    31.834829
d    72.034005
Name: four, dtype: float64
              a          b          c          d
one    33.594644  14.151517  81.779949  57.028864
three  94.723426   5.063484  62.065909  55.522443
              a          b          c          d
four   95.153937  70.695947  31.834829  72.034005
three  94.723426   5.063484  62.065909  55.522443
two    26.914832  11.108404  48.340908  86.074874
多位置索引
-----
              a          b          c          d
two    26.914832  11.108404  48.340908  86.074874
```

```
three  94.723426   5.063484  62.065909  55.522443
               a          b          c          d
one    33.594644  14.151517  81.779949  57.028864
three  94.723426   5.063484  62.065909  55.522443
```
切片索引

In [114]:
```
#布尔型索引
#和Series原理相同

df = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                  index = ['one','two','three','four'],
                  columns = ['a','b','c','d'])

print(df)
print('*'*30)

b1 = df < 20
print(b1)

print(df[b1])
print('*'*30)
#不做索引则会对数据每个值进行判断
#索引结果保留 所有数据：True返回原数据，False返回值为NaN

b2 = df['a']>20
print(b2)
print(df[b2])
#单列做判断
#索引结果保留 单列判断为True的行数据，!!!!包括其他列！！！

print('*'*30)
b3 = df[['a','b']] > 50
print(b3,type(b3))
print(df[b3])  #也可以书写为 df[df[['a','b']] > 50]
#多列做判断
#索引结果保留 所有数据：True返回原数据，False返回值为NaN

print('*'*30)
b4 = df.loc[['one','three']] < 50
print(b4,type(b4))
print(df[b4])  #也可以书写为 df[df.loc[['one','three']] < 50]
#多行做判断
#索引结果保留 所有数据：True返回原数据，False返回值为NaN
```

```
               a          b          c          d
one    38.254517  97.829681  47.332028  45.648618
two    11.186750  29.351951  87.972177  39.929152
three  32.920143  82.571036  34.331755  82.615511
four    4.390666  10.218779  21.107333  48.503572
******************************
           a      b      c      d
one    False  False  False  False
two     True  False  False  False
three  False  False  False  False
four    True   True  False  False
               a          b   c   d
one          NaN        NaN NaN NaN
two    11.186750        NaN NaN NaN
three        NaN        NaN NaN NaN
four    4.390666  10.218779 NaN NaN
******************************
```

```
one      True
two     False
three    True
four    False
Name: a, dtype: bool
               a          b          c          d
one    38.254517  97.829681  47.332028  45.648618
three  32.920143  82.571036  34.331755  82.615511
******************************
           a      b
one    False   True
two    False  False
three  False   True
four   False  False <class 'pandas.core.frame.DataFrame'>
        a          b   c   d
one   NaN  97.829681 NaN NaN
two   NaN        NaN NaN NaN
three NaN  82.571036 NaN NaN
four  NaN        NaN NaN NaN
******************************
          a      b     c      d
one    True  False  True   True
three  True  False  True  False <class 'pandas.core.frame.DataFrame'>
               a   b          c          d
one    38.254517 NaN  47.332028  45.648618
two          NaN NaN        NaN        NaN
three  32.920143 NaN  34.331755        NaN
four         NaN NaN        NaN        NaN
```

In [129]:
```python
#多重索引：比如同时索引行和列
#先选择列再选择行 —— 相当于对于一个数据，先筛选字段，再选择数据量

df = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                  index = ['one','two','three','four'],
                  columns = ['a','b','c','d'])

print(df)
print('-'*30)



print(df['a'].loc[['one','three']])
print(df[['b','c','d']].iloc[::2])
print(df[df['a'] < 50].iloc[:2])
```

```
               a          b          c          d
one    79.844148  64.537916  87.937310  81.135267
two    31.473157  34.889708  82.891536   1.757400
three  10.476268  23.633666  24.770819  85.184144
four   93.539393  57.062154  31.033067  45.945455
------------------------------
one      79.844148
three    10.476268
Name: a, dtype: float64
               b          c          d
one    64.537916  87.937310  81.135267
three  23.633666  24.770819  85.184144
               a          b          c          d
two    31.473157  34.889708  82.891536   1.757400
three  10.476268  23.633666  24.770819  85.184144
```

Pandas数据结构Dataframe：基本技巧

数据查看、转置 / 添加、修改、删除值 / 对齐 / 排序

In [136]:
```
#数据查看、转置

df = pd.DataFrame(np.random.rand(16).reshape(8,2)*100,
                  columns = ['a','b'])

print(df)
print('-'*40)

print(df.head(2))
print(df.tail())
#.head()查看头部数据
#.tail()查看尾部数据
#默认查看5条

print(df.T)
#.T 转置
```

```
           a          b
0  94.134553  98.075649
1  30.252216  44.216585
2  45.262321  38.177946
3  89.299540  35.097431
4  92.330446  63.388054
5  33.678413  42.643000
6  51.967086  83.342968
7   4.949641  30.002276
----------------------------------------
           a          b
0  94.134553  98.075649
1  30.252216  44.216585
           a          b
3  89.299540  35.097431
4  92.330446  63.388054
5  33.678413  42.643000
6  51.967086  83.342968
7   4.949641  30.002276
           0          1          2          3          4          5  \
a  94.134553  30.252216  45.262321  89.299540  92.330446  33.678413
b  98.075649  44.216585  38.177946  35.097431  63.388054  42.643000

           6          7
a  51.967086   4.949641
b  83.342968  30.002276
```

In [148]:
```
#添加与修改

df = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   columns = ['a','b','c','d'])
print(df)

df['e'] = 100
print(df)

df.loc[4] = 20
print(df)
```

```
df['e'] = 20
df[['a','c']] = 100
print(df)
# 索引后直接修改值
```

```
           a          b          c          d
0   7.271823  93.396596   3.610341  54.918616
1  63.964942  61.393722  48.892515  11.967147
2  27.879457  70.951428   5.525834  41.719079
3   5.662954  61.123307  60.132085  31.985112
           a          b          c          d    e
0   7.271823  93.396596   3.610341  54.918616  100
1  63.964942  61.393722  48.892515  11.967147  100
2  27.879457  70.951428   5.525834  41.719079  100
3   5.662954  61.123307  60.132085  31.985112  100
           a          b          c          d    e
0   7.271823  93.396596   3.610341  54.918616  100
1  63.964942  61.393722  48.892515  11.967147  100
2  27.879457  70.951428   5.525834  41.719079  100
3   5.662954  61.123307  60.132085  31.985112  100
4  20.000000  20.000000  20.000000  20.000000   20
     a          b    c          d   e
0  100  93.396596  100  54.918616  20
1  100  61.393722  100  11.967147  20
2  100  70.951428  100  41.719079  20
3  100  61.123307  100  31.985112  20
4  100  20.000000  100  20.000000  20
```

In [153]:

```
# 删除  del / drop()

df = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   columns = ['a','b','c','d'])
print(df)

del df['a']
print(df)
print('-----')
# del语句 - 删除列

print(df.drop(0))
print(df.drop([1,2]))
print(df)
print('-----')
# drop()删除行，inplace=False → 删除后生成新的数据，不改变原数据

print(df.drop(['d'], axis = 1))
print(df)
# drop()删除列，需要加上axis = 1，inplace=False → 删除后生成新的数据，不改变原数据
```

```
           a          b          c          d
0  85.001326  48.855630   3.487921  16.907450
1  91.066253  68.237074  41.642766  47.639123
2  17.283172  88.421410  64.556022  83.689086
3  55.104442  24.115355  75.151845  73.234390
           b          c          d
0  48.855630   3.487921  16.907450
1  68.237074  41.642766  47.639123
2  88.421410  64.556022  83.689086
3  24.115355  75.151845  73.234390
-----
           b          c          d
```

```
1  68.237074  41.642766  47.639123
2  88.421410  64.556022  83.689086
3  24.115355  75.151845  73.234390
           b          c         d
0  48.855630   3.487921  16.90745
3  24.115355  75.151845  73.23439
           b          c          d
0  48.855630   3.487921  16.907450
1  68.237074  41.642766  47.639123
2  88.421410  64.556022  83.689086
3  24.115355  75.151845  73.234390
-----
           b          c
0  48.855630   3.487921
1  68.237074  41.642766
2  88.421410  64.556022
3  24.115355  75.151845
           b          c          d
0  48.855630   3.487921  16.907450
1  68.237074  41.642766  47.639123
2  88.421410  64.556022  83.689086
3  24.115355  75.151845  73.234390
```

In [155]:
```
# 对齐

df1 = pd.DataFrame(np.random.randn(10, 4), columns=['A', 'B', 'C', 'D'])
df2 = pd.DataFrame(np.random.randn(7, 3), columns=['A', 'B', 'C'])
print(df1 + df2)
# DataFrame对象之间的数据自动按照列和索引（行标签）对齐
```

```
          A         B         C   D
0 -1.577108 -1.454698  3.958815 NaN
1  0.890740  0.448808 -0.562226 NaN
2 -2.130694  2.840875  2.391692 NaN
3 -0.139730 -2.709792  0.124406 NaN
4  1.178131 -1.702700  0.647456 NaN
5  0.761656  0.418825 -1.010436 NaN
6  1.369980  1.839608  1.374236 NaN
7       NaN       NaN       NaN NaN
8       NaN       NaN       NaN NaN
9       NaN       NaN       NaN NaN
```

In [167]:
```
# 排序1 - 按值排序 .sort_values
# 同样适用于Series

df1 = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   columns = ['a','b','c','d'])
print(df1)
print('-'*30)

print(df1.sort_values(['a'], ascending = True))  # 升序
print(df1.sort_values(['a'], ascending = False))  # 降序
print('------')
# ascending参数：设置升序降序，默认升序
# 单列排序

df2 = pd.DataFrame({'a':[1,2,4,5,6,2,2,2],
                  'b':list(range(8)),
                  'c':list(range(8,0,-1))})
```

```
print(df2)
print(df2.sort_values(['a','c']))
# 多列排序，按列顺序排序
```

```
           a          b          c          d
0  70.081595  89.657068  71.451934  52.245061
1  86.381561  11.528359  75.568016   5.487527
2  85.441581  88.979582  55.457414  28.395030
3  94.576180  97.458084  74.735408  33.160755
------------------------------
           a          b          c          d
0  70.081595  89.657068  71.451934  52.245061
2  85.441581  88.979582  55.457414  28.395030
1  86.381561  11.528359  75.568016   5.487527
3  94.576180  97.458084  74.735408  33.160755
           a          b          c          d
3  94.576180  97.458084  74.735408  33.160755
1  86.381561  11.528359  75.568016   5.487527
2  85.441581  88.979582  55.457414  28.395030
0  70.081595  89.657068  71.451934  52.245061
------
   a  b  c
0  1  0  8
1  2  1  7
2  4  2  6
3  5  3  5
4  6  4  4
5  2  5  3
6  2  6  2
7  2  7  1
   a  b  c
0  1  0  8
7  2  7  1
6  2  6  2
5  2  5  3
1  2  1  7
2  4  2  6
3  5  3  5
4  6  4  4
```

In [168]:

```
# 排序2 - 索引排序 .sort_index

df1 = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   index = [5,4,3,2],
                    columns = ['a','b','c','d'])
df2 = pd.DataFrame(np.random.rand(16).reshape(4,4)*100,
                   index = ['h','s','x','g'],
                    columns = ['a','b','c','d'])

print(df1)
print(df1.sort_index())
print(df2)
print(df2.sort_index())
# 按照index排序
# 默认 ascending=True, inplace=False
```

```
           a          b          c          d
5  53.797739  88.938062  38.752000  82.099178
4  95.662650  12.107602  45.770935  24.270212
3   2.800246  87.805481  75.853768  58.195603
2  43.664318  74.672200  29.804986  91.308756
```

```
          a          b          c          d
2  43.664318  74.672200  29.804986  91.308756
3   2.800246  87.805481  75.853768  58.195603
4  95.662650  12.107602  45.770935  24.270212
5  53.797739  88.938062  38.752000  82.099178
          a          b          c          d
h  28.948093  77.820071  67.598648  73.429437
s  49.438319  77.008631  21.661274  22.068433
x  70.773152  75.985549  37.034978  39.888356
g  21.505651  88.557211   4.941319  64.071087
          a          b          c          d
g  21.505651  88.557211   4.941319  64.071087
h  28.948093  77.820071  67.598648  73.429437
s  49.438319  77.008631  21.661274  22.068433
x  70.773152  75.985549  37.034978  39.888356
```