自学报告：Pandas库

李帅 2016013270

周展平 2016013253

本文档说明：

本文档是针对Pandas 0.23.4的学习报告，主要是从实际的数据分析场景出发，以各个环节为线索学习Pandas的特性。

一、Pandas 简介

Pandas 是一个 Python 的开源项目，是数据分析的一个常用的工具。官方的文档中如下描述它：

> `pandas` is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

(linkage：http://pandas.pydata.org/pandas-docs/stable/overview.html)

Pandas 数据结构的实现基于Numpy，可视化方面则基于 Matplotlib，因此 Pandas 对于它们均有很好的兼容性。

二、基本数据类型

三、文件读写

四、数据索引(index)、排序(sort)

五、数据分组(groupby)

数据分组的含义包含3个方面：

1. **Split**：将数据按照一定的标准进行分类。
2. **Apply**：对于每一个类别的数据，进行特定的操作。其中包括：
   （1）**Aggregation**：计算类别内数据的总体特征，如和、均值、维数
   （2）**Transformation**：对类内数据总体进行处理，如标准化、填补NA
   （3）**Filtration**：对某些类别的数据进行丢弃、筛选等
3. **Combine**：将经过操作的所有类别的数据重新按照某种方式组合起来 。这一部分内容将在

   下一节介绍。

下面简单介绍具体的方法：

1. Split

常用方法：

`grouped = obj.groupby()` 方法：接受的参数包括：

- 1个python函数，对指定方向(axis)的标签(label)进行处理，用来进行分类
- 1个列表或numpy数组对象，对应于指定方向(axis)的标签(label)

- 1个字典或Series对象，用于制定标签(label)到类名(group name)的映射关系
- 如果调用者是1个DataFrame对象，字符串指定了分类用到的列(colomn)

`grouped.get_group(key)` 方法：从所有分组中获得指定key的组。

`grouped.filter()` 方法：筛选分组

`grouped.count()` 方法：输出每个分组中的元素个数

常用属性：

`grouped.groups` 属性：获得所有分组

迭代：`for name, group in grouped:`

实例：

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
                          'foo', 'bar', 'foo', 'foo'],
                   'C' : np.random.randn(8),
                   'D' : np.random.randn(8),
                   'B' : ['one', 'one', 'two', 'three',
                          'two', 'two', 'one', 'three']
                   })

# 可以指定colomn
grouped = df.groupby('A')
grouped = df.groupby(['A', 'B'])
print(grouped.groups)
# 输出结果:
{('bar', 'one'): Int64Index([1], dtype='int64'), ('bar', 'three'):
Int64Index([3], dtype='int64'), ('bar', 'two'): Int64Index([5],
dtype='int64'), ('foo', 'one'): Int64Index([0, 6], dtype='int64'), ('foo',
'three'): Int64Index([7], dtype='int64'), ('foo', 'two'): Int64Index([2, 4],
dtype='int64')}

# 输出每个分组中的元素个数
print(grouped.count())
# 输出结果
            C  D
A   B
```

```
bar one    1  1
    three  1  1
    two    1  1
foo one    2  2
    three  1  1
    two    2  2

# 可以指定方法by以及方向axis
# 下面是按照colomn名是否为元音字母来进行分组
def get_letter_type(letter):
    if letter.lower() in 'aeiou':
       return 'vowel'
    else:
       return 'consonant'

grouped = df.groupby(get_letter_type, axis=1)
for group in grouped:
    print(group)
# 输出结果:
('consonant',         B         C         D
0    one  0.350096  1.082806
1    one  0.324642  0.611490
2    two  0.075571  0.654428
3  three  0.028201  0.024015
4    two  1.286625  1.217880
5    two  0.975612 -0.914238
6    one  0.282143  1.009814
7  three  1.181507  0.257534)
('vowel',      A
0  foo
1  bar
2  foo
3  bar
4  foo
5  bar
6  foo
7  foo)

# 取消默认的将关键字排序
grouped = df.groupby(get_letter_type, axis=1,sort=False)
print(grouped.get_group('consonant'))
# 输出结果：此时第二组的输出顺序为C, D, B
```

```
          C         D      B
0 -0.612971  0.758547    one
1 -0.601868  0.605106    one
2 -0.307514 -0.638541    two
3 -0.100397 -0.728291  three
4  0.399796  1.092549    two
5  0.555295  0.849624    two
6  0.139331  1.831764    one
7 -0.194881  0.244773  three
```

2. Apply

aggregation:

常用方法:

`grouped.aggregate(method)` 方法：使用method参数处理grouped中的每一个分组。

`grouped.agg([method1[,method2[,...]]])` 方法：一次性用多个方法进行处理。

`grouped.agg({colomn:method,...})` 方法：用不同方法处理不同列的数据。

实例:

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
                          'foo', 'bar', 'foo', 'foo'],
                   'C' : np.random.randn(8),
                   'D' : np.random.randn(8),
                   'B' : ['one', 'one', 'two', 'three',
                          'two', 'two', 'one', 'three']
                   })

grouped = df.groupby('A',as_index=False)

# 使用np.sum函数对每组求和
print(grouped.aggregate(np.sum))
# 输出结果:
     A         C         D
0  bar -1.000992 -0.081658
1  foo -1.340381  3.082666
```

```
# 效果与sum()函数相同:
print(grouped.sum())
# 输出结果:
     A         C         D
0  bar -1.000992 -0.081658
1  foo -1.340381  3.082666

# 多个方法处理
print(grouped['C'].agg([np.sum, np.mean, np.std]))
# 输出结果:
          sum      mean       std
A
bar -0.467876 -0.155959  0.300311
foo  2.787968  0.557594  0.748056

# 用不同方法处理不同列的数据
print(grouped.agg({'C' : 'sum', 'D' : 'std'}))
# 输出结果:
     A         C         D
0  bar  0.613597  0.415764
1  foo -2.231638  0.773108
```

常用方法:

`grouped.describe()` 方法：计算一系列的统计量，这些统计量也有专门的函数可以单独调用。

实例:

```
# 计算统计量
print(grouped.describe())
# 输出结果:
      C
    D
  count      mean       std       min       25%       50%       75%       max
count      mean       std       min       25%       50%       75%       max
0   1.0  0.254161       NaN  0.254161  0.254161  0.254161  0.254161  0.254161
  1.0  1.511763       NaN  1.511763  1.511763  1.511763  1.511763  1.511763
1   1.0  0.215897       NaN  0.215897  0.215897  0.215897  0.215897  0.215897
  1.0 -0.990582       NaN -0.990582 -0.990582 -0.990582 -0.990582 -0.990582
2   1.0 -0.077118       NaN -0.077118 -0.077118 -0.077118 -0.077118 -0.077118
  1.0  1.211526       NaN  1.211526  1.211526  1.211526  1.211526  1.211526
3   2.0 -0.491888  0.117887 -0.575247 -0.533567 -0.491888 -0.450209 -0.408530
  2.0  0.807291  0.761937  0.268520  0.537905  0.807291  1.076676  1.346061
4   1.0 -0.862495       NaN -0.862495 -0.862495 -0.862495 -0.862495 -0.862495
  1.0  0.024580       NaN  0.024580  0.024580  0.024580  0.024580  0.024580
5   2.0  0.024925  1.652692 -1.143704 -0.559389  0.024925  0.609240  1.193555
  2.0  0.592714  1.462816 -0.441652  0.075531  0.592714  1.109898  1.627081
```

transformation

常用方法:

`grouped.transform(method)` 方法：使用method方法对每组中的数据进行处理

实例:

```
index = pd.date_range('10/1/1999', periods=1100)
ts = pd.Series(np.random.normal(0.5, 2, 1100), index)
ts = ts.rolling(window=100,min_periods=100).mean().dropna()

key = lambda x: x.year
zscore = lambda x: (x - x.mean()) / x.std() #用于transform的函数，进行归一化
transformed = ts.groupby(key).transform(zscore)

grouped_trans = transformed.groupby(key)
print(grouped_trans.mean()) #期望
print(grouped_trans.std()) #标准差
# 输出结果
2000   -2.699790e-16
2001    1.861525e-16
```

```
2002   -6.561138e-16
dtype: float64
2000    1.0
2001    1.0
2002    1.0
dtype: float64
```

filtration:

常用方法:

`grouped.filter(method)` 方法：使用method方法对每组中的数据进行判断，返回True或False，从而筛选出返回True的数据

实例:

```
sf = pd.Series([1, 1, 1, 1, 2, 3, 4])
print(sf.groupby(sf).filter(lambda x: x.sum() > 3)) # 筛选出数据的和大于3的分组
# 输出结果
0    1
1    1
2    1
3    1
6    4
dtype: int64
```

apply:

常用方法:

`grouped.apply(method)` 方法：对每一个group应用apply方法

实例:

```
sf = pd.Series(np.random.normal(0.5, 2, 10), index=np.arange(10))
def f(x):
   return pd.Series([ x, x**2 ], index = ['x', 'x^2'])
print(sf.apply(f))
# 输出结果
          x        x^2
0  1.465540   2.147806
1  1.018819   1.037991
2  0.927661   0.860555
```

```
3  3.959855  15.680452
4  0.750213   0.562820
5  0.784470   0.615393
6  3.476535  12.086294
7  0.099393   0.009879
8 -1.783570   3.181121
9  0.427524   0.182777
```

（本节参考资料：http://pandas.pydata.org/pandas-docs/stable/groupby.html）

六、合并(concatenate,merge,join)

1. concatenate:

常用方法:

`pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False,keys=None, levels=None, names=None, verify_integrity=False,copy=True)` 方法：

- `objs`：需要合并的pandas对象的序列或者字典，其中字典的键值会被当做keys参数
- `axis`：合并的方向
- `join`：如何处理其他方向的index的合并，outer代表取并集，inner代表取交集
- `ignore_index`：是否忽略以前的index
- `keys`：作为新的index添加到合并后的元素中，形成多级的索引结构
- `levels`：一个列表序列，用来建立 multiIndex
- `names`：为新产生的index指定名字
- `verify_integrity`：查看合并方向上是否有重复
- `copy`：是否复制元素（深拷贝）

实例:

```
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'],
                    'E': ['E4', 'E5', 'E6', 'E7'],
                    'F': ['F4', 'F5', 'F6', 'F7']},
                     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])

# concantenate展示1: join='inner',keys=['x', 'y', 'z'],names=['level1']
concatenated1 = pd.concat([df1,df2,df3],join='inner',keys=['x', 'y',
'z'],names=['level1'])
print(concatenated1)
# 输出结果:
             A    B
level1
x      0    A0   B0
       1    A1   B1
       2    A2   B2
       3    A3   B3
y      4    A4   B4
       5    A5   B5
       6    A6   B6
       7    A7   B7
z      8    A8   B8
       9    A9   B9
       10  A10  B10
       11  A11  B11

# concantenate展示2: join='outer'
concatenated2 = pd.concat([df1,df2,df3],join='outer')
print(concatenated2)
# 输出结果:
      A    B    C    D    E    F
0    A0   B0   C0   D0  NaN  NaN
1    A1   B1   C1   D1  NaN  NaN
2    A2   B2   C2   D2  NaN  NaN
```

```
3    A3   B3   C3   D3  NaN  NaN
4    A4   B4  NaN  NaN   E4   F4
5    A5   B5  NaN  NaN   E5   F5
6    A6   B6  NaN  NaN   E6   F6
7    A7   B7  NaN  NaN   E7   F7
8    A8   B8   C8   D8  NaN  NaN
9    A9   B9   C9   D9  NaN  NaN
10  A10  B10  C10  D10  NaN  NaN
11  A11  B11  C11  D11  NaN  NaN

# concantenate展示3: axis=1,join='outer',ignore_index=True
concatenated3 =
pd.concat([df1,df2,df3],axis=1,join='outer',ignore_index=True)
print(concatenated3)
# 输出结果:
     0    1    2    3    4    5    6    7    8    9    10   11
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   E4   F4  NaN  NaN  NaN  NaN
5   NaN  NaN  NaN  NaN   A5   B5   E5   F5  NaN  NaN  NaN  NaN
6   NaN  NaN  NaN  NaN   A6   B6   E6   F6  NaN  NaN  NaN  NaN
7   NaN  NaN  NaN  NaN   A7   B7   E7   F7  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
```

2. merge:

此方法是针对两个 `DataFrame` 对象的、类似于数据库方法的高效合并的方法。

常用方法:

`pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)` 方法:

- `left` 、 `right`: 需要合并的 `DataFrame` 对象
- `how` : 当左右的键值集合不相同时如何解决，可以取的值为 `'left'`, `'right'`, `'outer'`, `'inner'`

- `on`：选择按照哪些键值进行merge操作
- `left_on` ： 从 `left` 对象中选择的作为键值的column或index
- `right_on` ： 从 `right` 对象中选择的作为键值的column或index
- `sort`：是否将keys按照自字典序排序
- `suffixes`：当column名相同时用以加以区分的后缀
- `copy`：是否复制元素（深拷贝）
- `indicator`：用来指示每一行的数据的来源，在结果中额外增加`_merge_` 列，值可以为 `'left_only'`, `'right_only'`, `'both'`
- `validate` ：验证合并的两元素的关系，取值为 '1:1' ,'1:m','m:1','m:m'

实例:

```
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']})

# merge展示1: on=['key1', 'key2']
merged1 = pd.merge(left, right, on=['key1', 'key2'])
print(merged1)
# 输出结果:
  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

# merge展示2: how='left', on=['key1', 'key2'], indicator=True, 键值组合集合以
left为准
merged2 = pd.merge(left, right, how='left', on=['key1', 'key2'],
indicator=True)
print(merged2)
# 输出结果:
  key1 key2   A   B    C    D      _merge
0   K0   K0  A0  B0   C0   D0        both
1   K0   K1  A1  B1  NaN  NaN   left_only
2   K1   K0  A2  B2   C1   D1        both
```

```
3   K1   K0  A2  B2   C2   D2       both
4   K2   K1  A3  B3  NaN  NaN  left_only
# merge展示3: how='right', on=['key1', 'key2'], indicator=True, 键值组合集合以right为准
merged3 = pd.merge(left, right, how='right', on=['key1', 'key2'], indicator=True)
print(merged3)
# 输出结果:
  key1 key2    A    B   C   D      _merge
0   K0   K0   A0   B0  C0  D0        both
1   K1   K0   A2   B2  C1  D1        both
2   K1   K0   A2   B2  C2  D2        both
3   K2   K0  NaN  NaN  C3  D3  right_only
# merge展示4: how='outer', on=['key1', 'key2'], indicator=True, 键值组合集合求并集
merged4 = pd.merge(left, right, how='outer', on=['key1', 'key2'], indicator=True)
print(merged4)
# 输出结果:
  key1 key2    A    B    C    D      _merge
0   K0   K0   A0   B0   C0   D0        both
1   K0   K1   A1   B1  NaN  NaN   left_only
2   K1   K0   A2   B2   C1   D1        both
3   K1   K0   A2   B2   C2   D2        both
4   K2   K1   A3   B3  NaN  NaN   left_only
5   K2   K0  NaN  NaN   C3   D3  right_only
# merge展示5: how='inner', on=['key1', 'key2'], indicator=True, 键值组合集合求交集
merged5 = pd.merge(left, right, how='inner', on=['key1', 'key2'], indicator=True)
print(merged5)
# 输出结果:
  key1 key2   A   B   C   D _merge
0   K0   K0  A0  B0  C0  D0   both
1   K1   K0  A2  B2  C1  D1   both
2   K1   K0  A2  B2  C2  D2   both
```

3. join:

此方法是针对两个 `DataFrame` 对象，与 `merge` 方法类似。

常用方法:

`DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False)` 方法：

- `other`：`DataFrame` 对象、带有 `name` 属性的`Series` 对象或者`DataFrame` 对象的列表
- `on`：选择进行合并时选用的列
- `how`：可以取的值为`'left'`, `'right'`, `'outer'`, `'inner'`，与merge方法的含义类似
- `lsuffix`、`rsuffix`：当column名相同时用以加以区分的后缀
- `sort`：是否对键值对按照字典序进行排序

事实上，以下两个函数是等价的：

```
left.join(right, on=key_or_keys)
pd.merge(left, right, left_on=key_or_keys, right_index=True,
      how='left', sort=False)
```

因此，理解了 `merge` 函数也就理解了 `join` 函数，此处不再举例。

（本节参考资料：http://pandas.pydata.org/pandas-docs/stable/merging.html）

七、可视化

在这里，我们针对数据集 `MovieLens` (https://grouplens.org/datasets/movielens/) 进行可视化分析。

1、散点图：

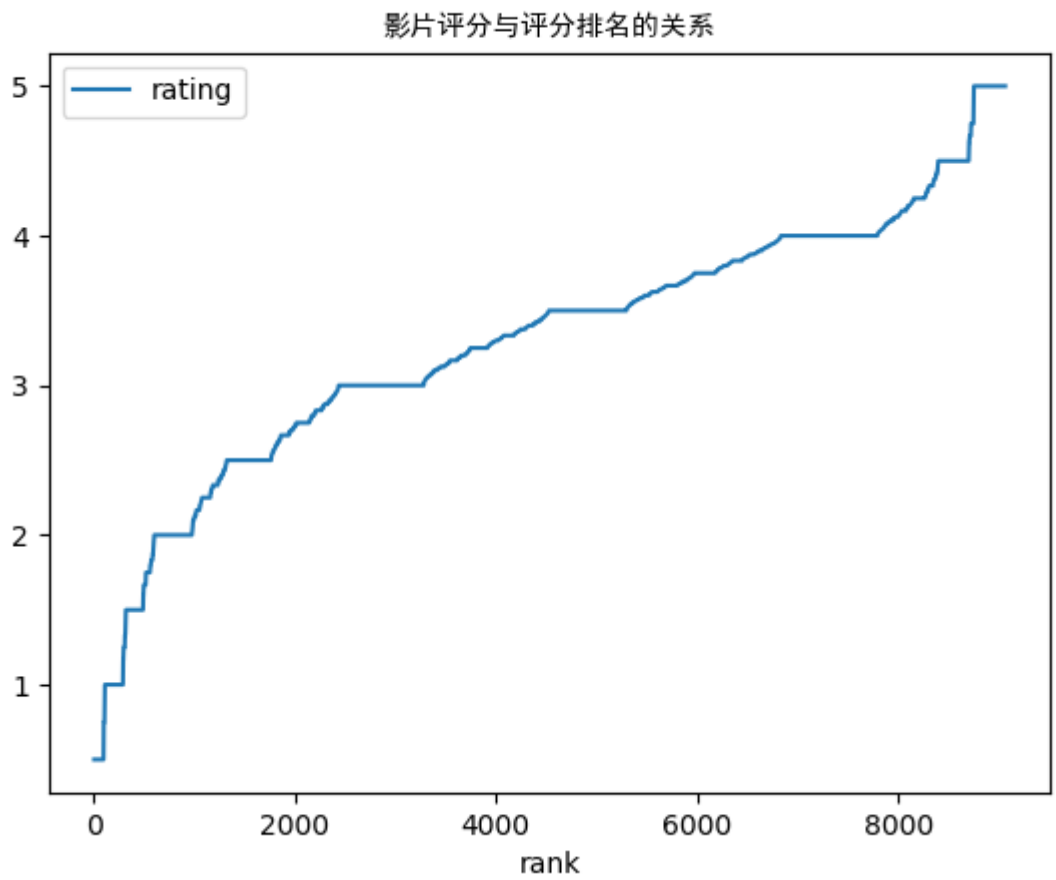
方法： `series_obj.plot()` 或 `dataframe_obj.plot()`，

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# 读入数据
path = '../dataset/ml-latest-small/'
links = pd.read_csv(path+'links.csv', encoding='latin-1')
movies = pd.read_csv(path+'movies.csv', encoding='latin-1')
ratings = pd.read_csv(path+'ratings.csv', encoding='latin-1')
tags = pd.read_csv(path+'tags.csv', encoding='latin-1')
```

```
# 综合各表数据
movie_ratings = pd.merge(movies, ratings)
# 计算每部影片的评分的平均值
ratings_mean = ratings.groupby(by='movieId').mean()
# 按照平均分排序
ratings_mean_sorted = ratings_mean.sort_values(by='rating')
ratings_mean_sorted['rank'] =
ratings_mean_sorted['rating'].rank(ascending=1,method='first')
# 绘制评分的散点图，按照评分从低到高
ratings_mean_sorted.plot(x='rank',y='rating')
plt.title(u"影片评分与评分排名的关系", fontproperties="SimHei")
plt.show()
```

输出如图:



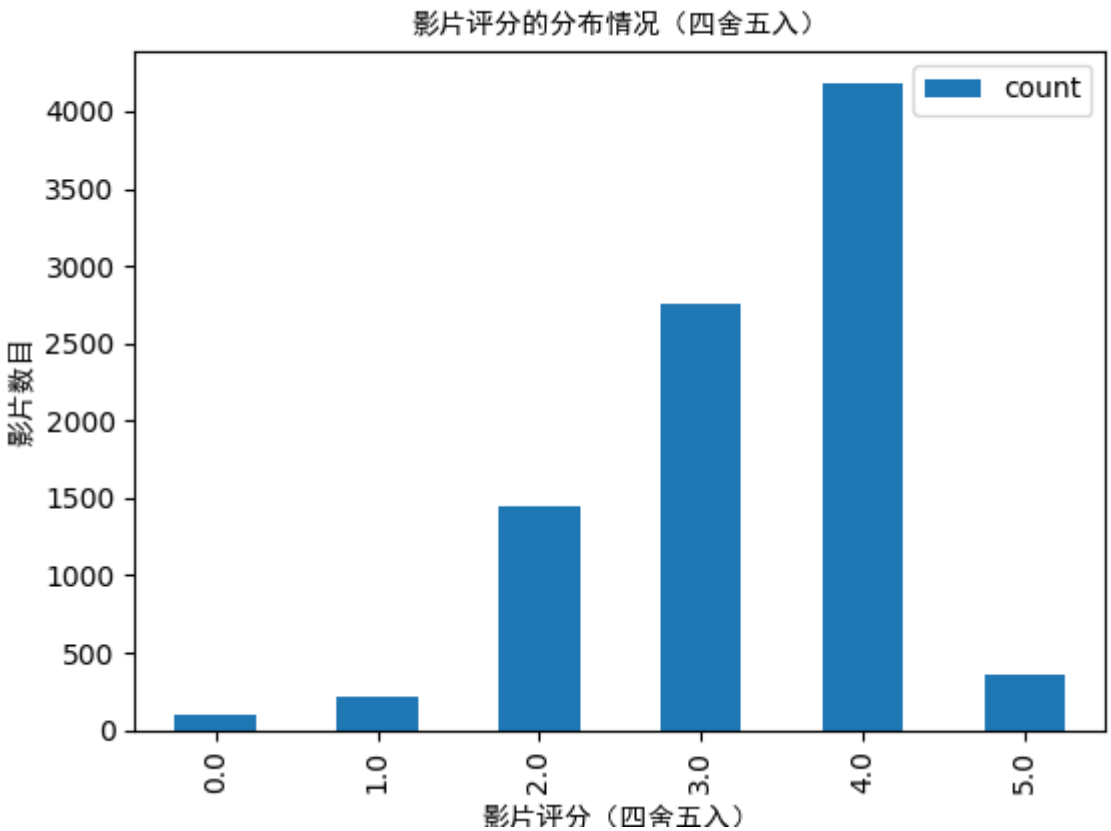
2. 条形图

方法: `series_obj.plot.bar()` 或 `dataframe_obj.plot.bar()` ,

```
# 四舍五入
```

```
ratings_mean['rating'] = ratings_mean['rating'].apply(np.round)
# 按照四舍五入后的评分进行分组
grouped = ratings_mean.groupby('rating')
count = []
names = []
for name, group in grouped:
    names.append(name)
    count.append(len(group))
rel = pd.DataFrame({'count':count},index=names)
# 条形图
rel.plot.bar()
plt.title(u"影片评分的分布情况（四舍五入）", fontproperties="SimHei")
plt.xlabel(u"影片评分（四舍五入）", fontproperties="SimHei")
plt.ylabel(u"影片数目", fontproperties="SimHei")
plt.show()
```

输出如图：

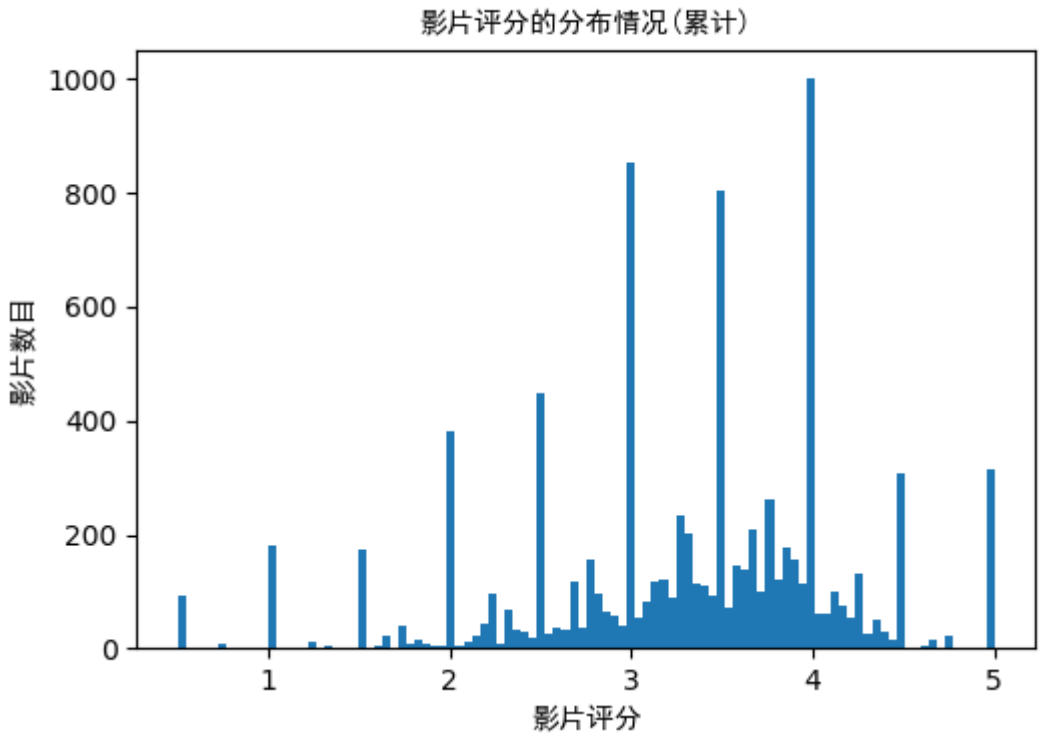


3. 直方图

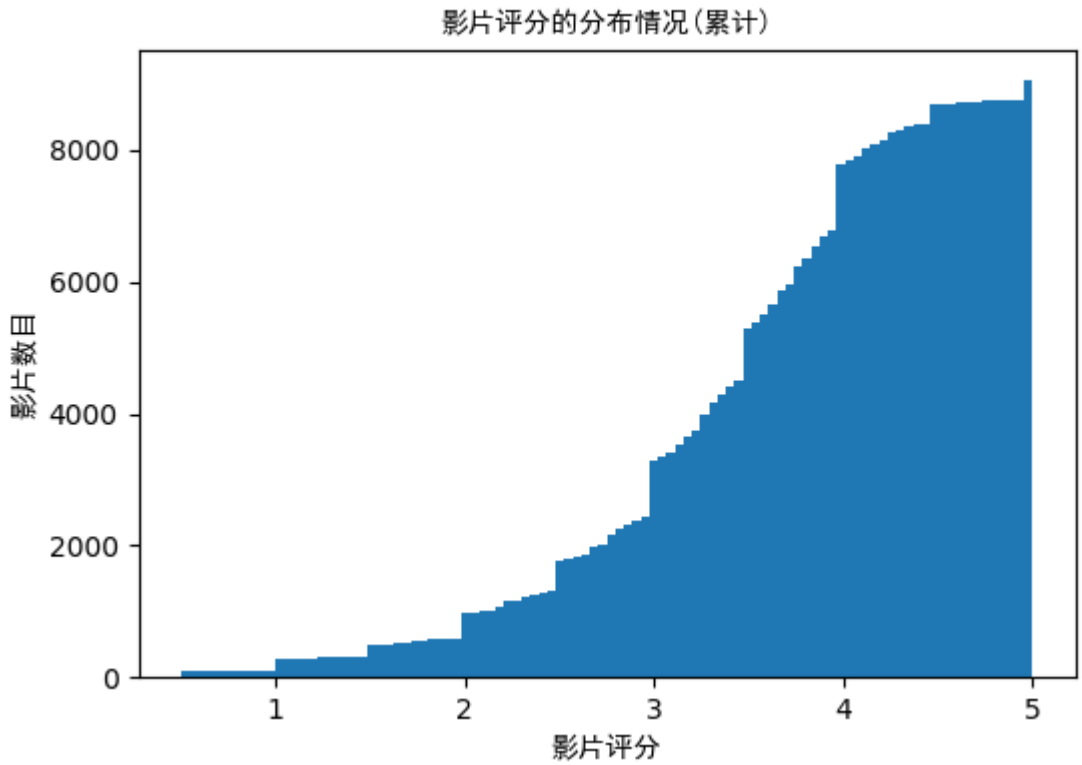
方法： `series_obj.plot.hist([alpha=])` 或 `dataframe_obj.plot.hist([alpha=])` ，

```
# 直方图
ratings_mean['rating'].plot.hist(bins=100)
plt.title(u"影片评分的分布情况", fontproperties="SimHei")
plt.xlabel(u"影片评分", fontproperties="SimHei")
plt.ylabel(u"影片数目", fontproperties="SimHei")
plt.show()
```

输出如图：



设置 cumulative = True 后，得到累计图：

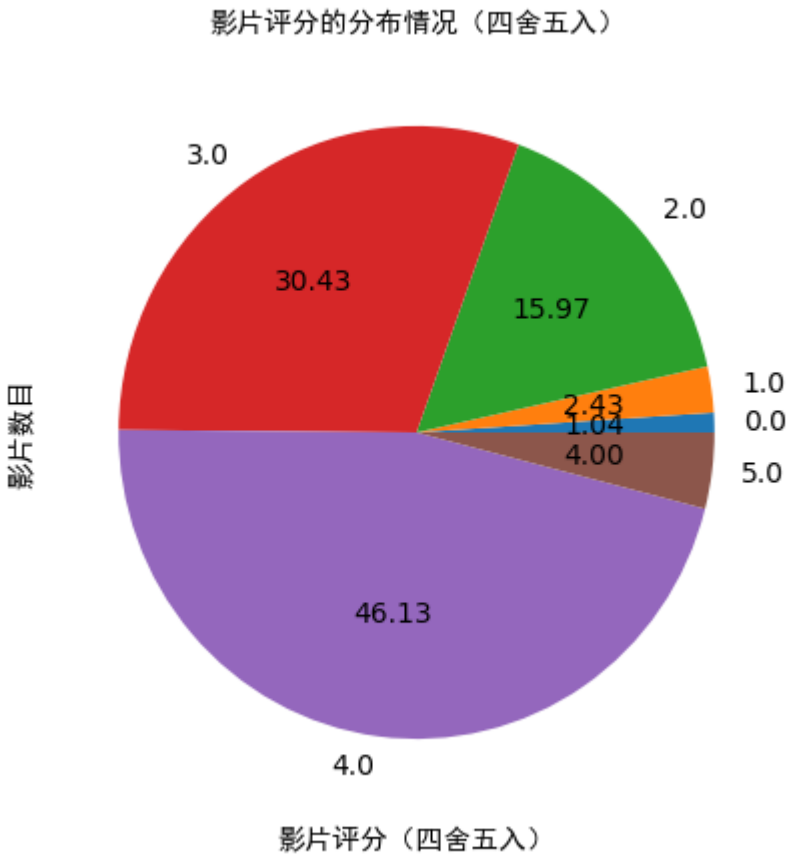


4. 扇形图

方法： `series_obj.plot.pie(figsize=(,))`或`dataframe_obj.plot.pie(figsize=(,))`，

```
# 扇形图
rel['count'].plot.pie(figsize=(6,6),autopct='%.2f')
```

输出结果：

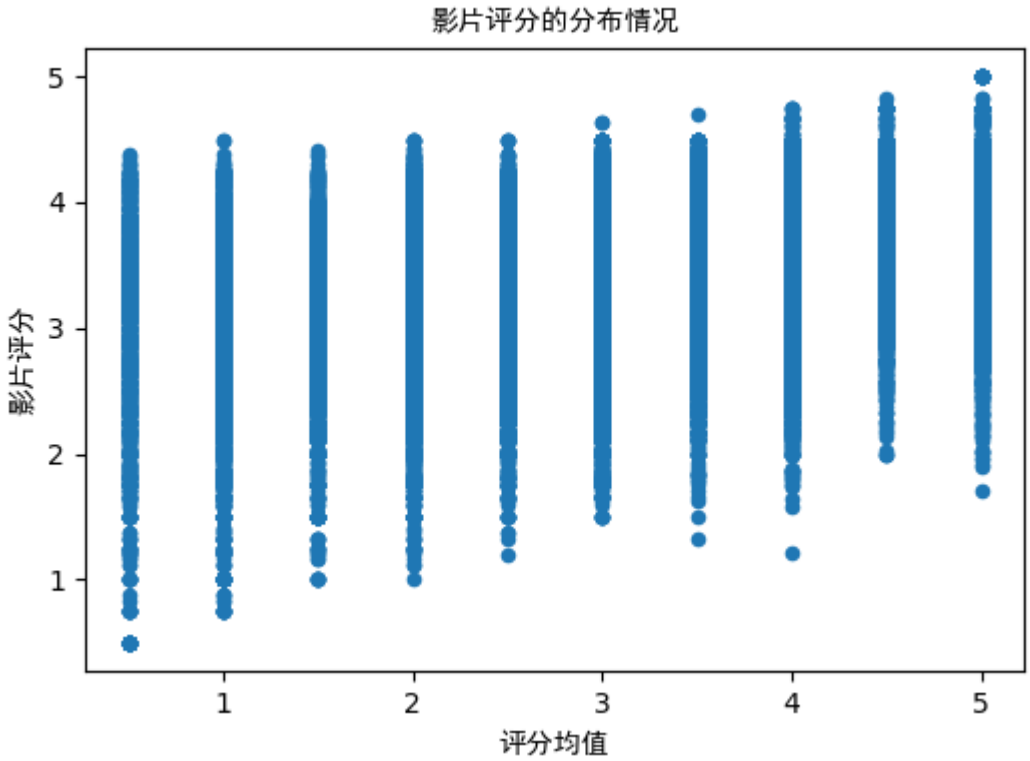


5. 散点图

方法： `series_obj.plot.scatter(x=, y= [,c= [,...]])` 或 `dataframe_obj.plot.scatter(x=,y= [,c= [,...]])` ，

```
# 将实际评分与平均分结合在一起
scatter = pd.merge(movie_ratings,ratings_mean,on='movieId')
scatter.plot.scatter(x='rating_x',y='rating_y')
# 散点图
plt.title(u"影片评分的分布情况", fontproperties="SimHei")
plt.xlabel(u"评分均值", fontproperties="SimHei")
plt.ylabel(u"影片评分", fontproperties="SimHei")
```

输出结果：

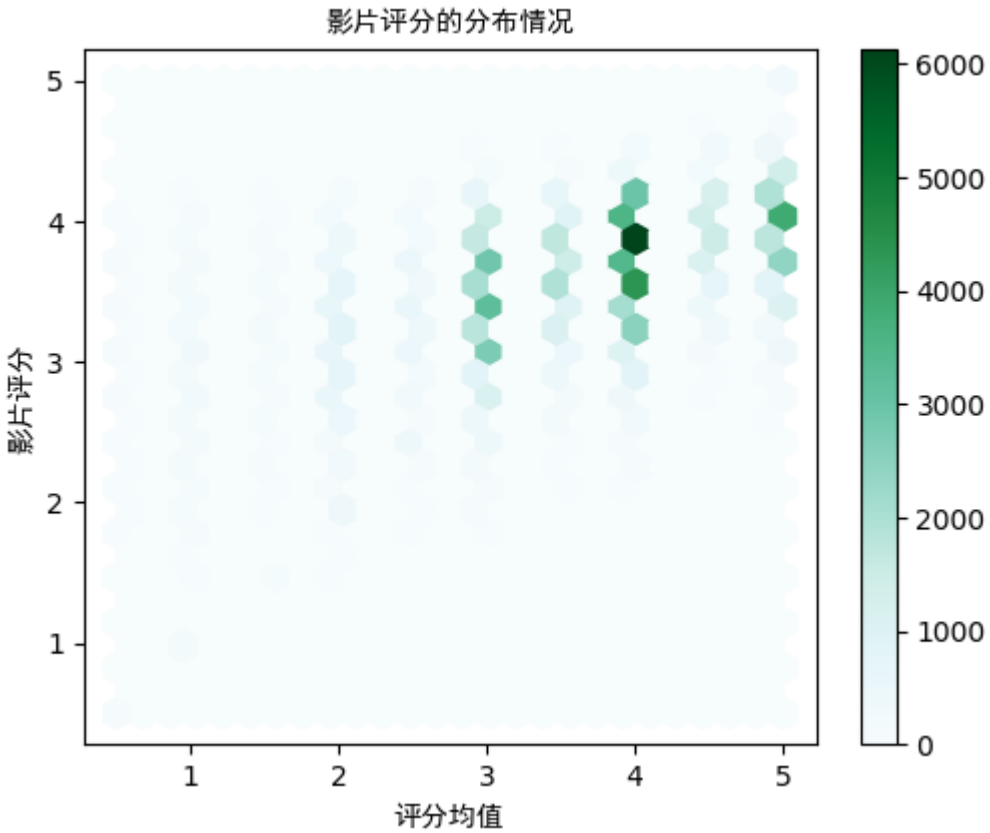


6. Hexagonal Bin图：

方法： `series_obj.plot.hexbin(x=, y= [,gridsize= [,...]])` 或 `dataframe_obj.plot.hexbin(x=,y= [,gridsize= [,...]])` ，

```
# Hexagonal Bin图:
scatter.plot.hexbin(x='rating_x',y='rating_y',gridsize=25)
```

输出结果：



八、参考资料