

# movielens电影评分数据分析(上)

---

## ●数据从哪里来？

给定数据集-MovieLens 1M数据集

一组从20世纪90年末到21世纪初由MovieLens用户提供的电影评分数据。这些数据中包括电影评分、电影元数据（风格类型和年代）以及关于用户的人口统计学数据（年龄、邮编、性别和职业等）。

MovieLens 1M数据集含有来自6000名用户对4000部电影的100万条评分数据。分为三个表：评分、用户信息和电影信息。

ratings.dat   movie.dat   users.dat

## ● 1.1 数据的读取

文本文件读取read\_table和read\_csv常用参数及其说明。

| 参数名称      | 说明  |
|-----------|---|
| filepath  | 接收string。代表文件路径。无默认。                                      |
| sep       | 接收string。代表分隔符。read_csv默认为“,”，read_table默认为制表符“[Tab]”。    |
| header    | 接收int或sequence。表示将某行数据作为列名。默认为infer，表示自动识别。               |
| names     | 接收array。表示列名。默认为None。                                     |
| index_col | 接收int、sequence或False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。 |
| dtype     | 接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。             |
| engine    | 接收c或者python。代表数据解析引擎。默认为c。                                |
| nrows     | 接收int。表示读取前n行。默认为None。                                    |

## ● 1.1 数据的读取

- read\_table和read\_csv函数中的sep参数是指定文本的分隔符的，如果分隔符指定错误，在读取数据的时候，每一行数据将连成一片。
- header参数是用来指定列名的，如果是None则会添加一个默认的列名。
- encoding代表文件的编码格式，常用的编码有utf-8、utf-16、gbk、gb2312、gb18030等。如果编码指定错误数据将无法读取，IPython解释器会报解析错误。

## ●1.1 数据的读取

通过pandas.read\_table将各个表分别读到一个pandas DataFrame对象中

eg：用户表的读取：

```
users = pd.read_table('users.dat', header=None,  
names=['UserID','Gender','Age','Occupation','Zip-code'], sep='::')
```

并打印列表长度 print(len(users)) 及查看前五行数据

## ●1.1 数据的读取

# 同样方法，导入电影评分表

```
ratings = pd.read_table('ratings.dat', header=None, names=['UserID', 'MovieID',  
'Rating', 'Timestamp'], sep='::')
```

# 同样方法，导入电影数据表

```
movies = pd.read_table('movies.dat', header=None, names=['MovieID', 'Title',  
'Genres'], sep='::')
```

## ●问题

对分布在三个表的数据进行分析同时进行分析很难，那必须将所有数据都合并到一个表中进行分析。采用什么方法呢？

下面，用pandas的merge函数将ratings跟users合并到一起，然后再将movies也合并进去。

```
data = pd.merge(pd.merge(ratings,users),movies)
```

pandas会根据列名的重叠情况推断出哪些列是合并（或连接）键

## ● 1.2 数据的查看

# 合并后的每一条记录反映了每个人的年龄，职业，性别，邮编，电影ID，评分，时间戳，电影信息，电影分类等一系列信息

# 比如我们查看用户id为12的所有信息

```
data[data.UserID == 12]
```



## ● 1.3.1 查看每一部电影不同性别的平均评分

|                                   | Gender | F        | M        |
|-----------------------------------|--------|----------|----------|
| Title                             |        |          |          |
| \$1,000,000 Duck (1971)           |        | 3.375000 | 2.761905 |
| 'Night Mother (1986)              |        | 3.388889 | 3.352941 |
| 'Til There Was You (1997)         |        | 2.675676 | 2.733333 |
| 'burbs, The (1989)                |        | 2.793478 | 2.962085 |
| ...And Justice for All (1979)     |        | 3.828571 | 3.689024 |
| 1-900 (1994)                      |        | 2.000000 | 3.000000 |
| 10 Things I Hate About You (1999) |        | 3.646552 | 3.311966 |
| 101 Dalmatians (1961)             |        | 3.791444 | 3.500000 |
| 101 Dalmatians (1996)             |        | 3.240000 | 2.911215 |
| 12 Angry Men (1957)               |        | 4.184397 | 4.328421 |
| 13th Warrior, The (1999)          |        | 3.112000 | 3.168000 |

### ● 1.3.1 查看每一部电影不同性别的平均评分

使用povit\_table函数创建透视表-pivot\_table函数常用参数及其说明

➤ 利用pivot\_table函数可以实现透视表，pivot\_table()函数的常用参数及其使用格式如下。

```
pandas.pivot_table(data, values=None, index=None, columns=None, aggfunc='mean',  
fill_value=None, margins=False, dropna=True, margins_name='All')
```

| 参数名称    | 说明   |
|---------|--|
| data    | 接收DataFrame。表示创建表的数据。无默认。                                      |
| values  | 接收字符串。用于指定想要聚合的数据字段名，默认使用全部数据。默认为None。                         |
| index   | 接收string或list。表示行分组键。默认为None。                                  |
| columns | 接收string或list。表示列分组键。默认为None。                                  |
| aggfunc | 接收functions。表示聚合函数。默认为mean。                                    |
| margins | 接收boolearn。表示汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。默认为True。 |
| dropna  | 接收boolearn。表示是否删掉全为NaN的列。默认为False。                             |

### ● 1.3.2 查看每一部电影不同性别的平均评分

#### ● pivot\_table函数主要的参数调节

- #index表示透视表的行
- #columns表示透视表的列
- #aggfunc表示对分析对象进行的分析，一般默认为求平均值，可以指定
- #margins表示添加每行每列求和的值，默认不添加。

### ● 1.3.2 计算评分分歧，找出男性和女性观众分歧最大的电影并进行排序

```
data_gender['diff'] = data_gender.F - data_gender.M
```

#### # 进行排序

```
data_gender_sorted = data_gender.sort_values(by='diff', ascending=False)
```

ascending 参数用于控制升序或者降序

### ● 1.3.3 算出每部电影平均得分并对其进行排序

# 算出每部电影平均得分

```
data_mean_rating = data.pivot_table(values='Rating', index='Title', aggfunc='mean')
```

# 对电影平均得分排序

```
data_mean_rating_sorted = data_mean_rating.sort_values(by='Rating', ascending=False)
```

## ● 1.3.4 查看评分次数多的电影并进行排序

# 查看评分最多的电影

```
data_rating_num = data.groupby('Title').size()
```

先将 DataFrame 按电影标题分组，接下来利用size方法计算每组样本的个数，

# 评分最多的电影进行排序

```
data_rating_num_sorted = data_rating_num.sort_values(ascending=False)
```

- 1.想过滤掉评分条目数不足250条的电影
- 2.评分最高的十部电影
- 3.不同的电影风格出现的次数

# movielens电影评分数据分析(下)

---



### ● 1.3.5 过滤掉评分条目数不足250条的电影

#过滤掉评分条目数不足250条的电影

```
hot_movies=data_rating_num[data_rating_num>250]
```

#对评分数量进行排序，并取前20条数据

```
data_rating_num_sorted = hot_movies.sort_values(ascending=False)
```

### ●1.3.6 评分最高的十部电影

查看每一部电影被评论过的次数和被打的平均分。取出至少被评论过100次的电影按照平均评分从大到小排序，取最大的10部电影

按照电影名称分组，用agg函数通过一个字典{ 'rating' : [np.size, np.mean]}来按照key即rating这一列聚合

#查看评分最高的十部电影

```
movie_stats = data.groupby('Title').agg({'Rating':[np.size, np.mean]})
```

```
atleast_100 = movie_stats['Rating']['size'] >= 100
```

```
movie_stats[atleast_100].sort_values([('Rating', 'mean')], ascending=False)[:10]
```

### 1.1.2 使用agg方法聚合数据

- **agg和aggregate函数参数及其说明**

- agg, aggregate方法都支持对每个分组应用某函数, 包括Python内置函数或自定义函数。同时这两个方法也能够直接对DataFrame进行函数应用操作。
- 在正常使用过程中, agg函数和aggregate函数对DataFrame对象操作时功能几乎完全相同, 因此只需要掌握其中一个函数即可。它们的参数说明如下表。

*DataFrame.agg(func, axis=0, \*args, \*\*kwargs)*

*DataFrame.aggregate(func, axis=0, \*args, \*\*kwargs)*

| 参数名称 | 说明  |
|------|---|
| func | 接收list、dict、function。表示应用于每行 / 每列的函数。无默认。 |
| axis | 接收0或1。代表操作的轴向。默认为0。                       |

## 1.1.2 使用agg方法聚合数据

### • agg方法求统计量

- 可以使用agg方法一次求出当前数据中所有菜品销量和售价的总和与均值，如 `detail[['counts','amounts']].agg([np.sum,np.mean])`。
- 对于某个字段希望只做求均值操作，而对另一个字段则希望只做求和操作，可以使用字典的方式，将两个字段名分别作为key，然后将NumPy库的求和与求均值的函数分别作为value，如 `detail.agg({'counts':np.sum,'amounts':np.mean})`。
- 在某些时候还希望求出某个字段的多个统计量，某些字段则只要求一个统计量，此时只需要将字典对应key的value变为列表，列表元素为多个目标的统计量即可，如 `detail.agg({'counts':np.sum,'amounts':[np.mean,np.sum]})`

## 1.1.2 使用agg方法聚合数据

- agg方法与自定义的函数

- 在agg方法可传入读者自定义的函数。
- 使用自定义函数需要注意的是NumPy库中的函数np.mean, np.median, np.prod, np.sum, np.std, np.var能够在agg中直接使用, 但是在自定义函数中使用NumPy库中的这些函数, 如果计算的时候是单个序列则会无法得出想要的结果, 如果是多列数据同时计算则不会出现这种问题。
- 使用agg方法能够实现对每一个字段每一组使用相同的函数。
- 如果需要对不同的字段应用不同的函数, 则可以和Dataframe中使用agg方法相同。

### ● 1.3.7 查看不同年龄的分布情况并且采用直方图进行可视化

(1) 查看用户的年龄分布：

```
users.Age.plot.hist(bins=30)
```

```
plt.title("Distribution of users' ages")
```

```
plt.ylabel('count of users')
```

```
plt.xlabel('age');
```

### ●1.3.7 在原数据中标记出用户位于的年龄分组

( 1 ) #用pandas.cut函数将用户年龄分组：

```
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
```

```
data['age_group'] = pd.cut(data.Age, range(0, 81, 10), labels=labels)
```

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.cut.html>

### ● 1.3.8 电影评分表中计算不同类型电影的频数

#### 1.对数据进行规整-movies

规整化处理的方法与步骤：

- [1].以电影所属类型（ genres ）及符号 ' | ' 为依据进行分割；
- [2].如果一个电影有多个类型（ genres ），将分割成多个列表（ List ）；
- [3].将分割后得到的多个列表（ Lists ）转换为一个数据框（ DataFrame ）；
- [4].将数据框的索引设置为movieId

```
movies_tidy1=pd.DataFrame(movies.Genres.str.split('|').tolist(),index=movies.MovieID)
```



### ● 1.3.8 电影评分表中计算不同类型电影的频数

|         | 0         | 1          | 2       | 3    | 4    | 5    |
|---------|-----------|------------|---------|------|------|------|
| MovieID |           |            |         |      |      |      |
| 1       | Animation | Children's | Comedy  | None | None | None |
| 2       | Adventure | Children's | Fantasy | None | None | None |
| 3       | Comedy    | Romance    | None    | None | None | None |

|   | movieId | level_1 | 0         |
|---|---------|---------|-----------|
| 0 | 1       | 0       | Adventure |
| 1 | 1       | 1       | Animation |
| 2 | 1       | 2       | Children  |
| 3 | 1       | 3       | Comedy    |
| 4 | 1       | 4       | Fantasy   |
| 5 | 2       | 0       | Adventure |
| 6 | 2       | 1       | Children  |
| 7 | 2       | 2       | Fantasy   |
| 8 | 3       | 0       | Comedy    |
| 9 | 3       | 1       | Romance   |

[5].按规整化数据的基本原则，采用stack（）函数进行重构，并重新设置行索引

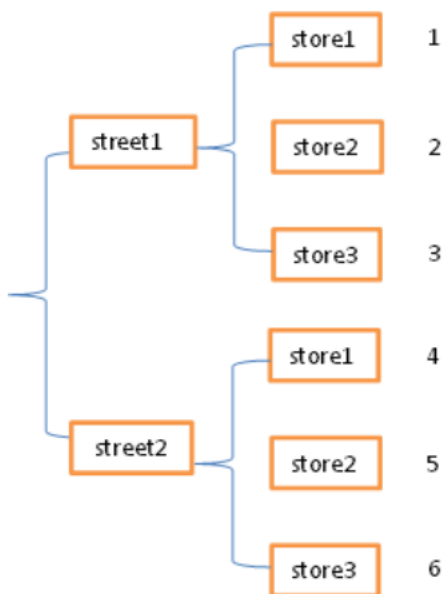
注：重新设置行索引后，原来的行索引保留为movieId列

```
movies_tidy2=movies_tidy1.stack().reset_index()
```

### ●1.3.8 电影评分表中计算不同类型电影的频数

在用pandas进行数据重排时，经常用到stack和unstack两个函数。stack的意思是堆叠，堆积，unstack即“不要堆叠”，我对两个函数是这样理解和区分的。

常见的数据的层次化结构有两种，一种是表格，一种是“花括号”，即下面这样的两种形式：



|         | store1 | store2 | store3 |
|---------|--------|--------|--------|
| street1 | 1      | 2      | 3      |
| street2 | 4      | 5      | 6      |

### ● 1.3.8 电影评分表中计算不同类型电影的频数

[6].删除level\_1列，将columns为0的列重命名为genres,并重新定义数据框为movies\_genres

```
movies_genres=movies_tidy2.drop('level_1',axis=1).rename(columns={0:'genres'})
```

```
movies_genres.head()
```

[7].将原movies数据中的Genres列替换成movies\_genres，得到规整化处理后的movies数据

```
movies=pd.merge(movies.drop('Genres',axis=1),movies_genres)
```

```
movies.head()
```

### ● 1.3.8 电影评分表中计算不同类型电影的频数

基于规整化处理后的movies和ratings，构建电影评分数据集movie\_ratings

[1].去掉movies中的title列，得到新movies

[2].合并新movies与ratings，连接键为movieId

[3].得到新的数据框movie\_ratings

[4].显示数据框movie\_ratings的前5行

```
movie_ratings=ratings.merge(movies.drop('Title',axis=1),on='MovieID',how='inner')
```

```
movie_ratings.head()
```

merge: 合并数据集，通过left，right确定连接字段，默认是两个数据集相同的字段

参数 说明

left 参与合并的左侧DataFrame

right 参与合并的右侧DataFrame

how 连接方式: 'inner' (默认); 还有, 'outer'、'left'、'right'

on 用于连接的列名，必须同时存在于左右两个DataFrame对象中，如果未指定，则以left和right列名的交集作为连接键

left\_on 左侧DataFrame中用作连接键的列

right\_on 右侧DataFrame中用作连接键的列

left\_index 将左侧的行索引用作其连接键

right\_index 将右侧的行索引用作其连接键

sort 根据连接键对合并后的数据进行排序，默认为True。有时在处理大数据集时，禁用该选项可获得更好的性能

suffixes 字符串值元组，用于追加到重叠列名的末尾，默认为 ('\_x','\_y')。例如，左右两个DataFrame对象都有'data'，则结果中就会出现'data\_x'，'data\_y'

copy 设置为False，可以在某些特殊情况下避免将数据复制到结果数据结构中。默认总是赋值

'''

### ● 1.3.8 电影评分表中计算不同类型电影的频数

计算movies\_ratings中不同类型电影的频数

[1].按genres列进行分组处理

[2].计算movieId的频数

[3].按movieId的频次进行降序

```
movie_ratings.groupby(['genres'])['MovieID'].size().sort_values(ascending=False)
```

**谢谢!!!**

---