3.10 Pandas数据透视表

1 什么是数据透视表

之前已经学习了如何使用**分组数据处理** `groupby()` 技术来探索数据集内部的关联性。本节将学习Pandas另外一个杀手锏技术**数据透视表**(pivot table)。

任务：需求分析

> 知识点：
>
> - **数据透视表**在实际的数据分析中非常有用，通过它可以从多角度来分析数据，以达到**横看成峰侧成岭**效果。
> - 从**不同角度**对**相同的数据**进行处理和分析，以查看**不同层面**的数据结果，从而得到想要的数据信息。
> - **数据透视表**就像一个**万花筒**，通过旋转这个特别的万花筒，可以从中获得不断变化的事物细节，但是**事物的本身**其实并未发生变化。

> - Excel、数据库、Pandas等数据处理工具都提供了**数据透视表**功能。精通**数据透视表**，就可以在数据分析领域游刃有余。

在学习Pandas提供的**数据透视表**功能之前，先了解下Excel的**数据透视表**功能，有个直观的了解。

- 虽然Excel只需要通过**点点点**操作，但是它在处理大规模数据时，效率非常低，这时Pandas就展现出它魅力之处了。
- 学信息科班出生的人，怎能满足**点点点**操作操作呢？当然要使用程序来实现**数据透视表**分析功能了。

任务：初识数据透视表功能

下图展示了数据透视表的功能。

> 术语(重要)： `Dataframe` 的行和列
>
> - **列**： `Dataframe` 的数据列，也称为**字段**或**特征**。
> - **行**： `Dataframe` 的每行数据称为**数据记录**，或**样本**。
> - 通常，**每行数据**拥有多个**字段**或**特征**。如"销售单"、"国家"、"发货数量"。
> - 每个**字段**都有对应的 `value` 称为**字段值**，如"国家"字段的值有"中国"、"法国"、"日本"。

数据透视表功能：选择一个或多个关键列，将数据进行**多维**分组，然后对分组进行统计。

这不就是 `groupby()` 功能吗？相对于 `groupby()` ，**数据透视表**更像是一种**多维**的 `groupby()` 分组数据处理，它可以对在行或列维度上同时进行分组。

```
# 导入sys模块的argv,winver成员，并为其指定别名v、wv
from sys import argv as v, winver as wv
# 使用导入成员（并指定别名）的语法，直接使用成员的别名访问
print(v[0])
print(wv)

C:\Users\15152\anaconda3\lib\site-packages\ipykernel_launcher.py
3.8
```

```
[3, -2, -7, 9, 1, -4]
```

```
[3, -2, -7, 9, 1, -4]
```

任务：数据导入

下面将使用**泰坦尼克号**的乘客信息数据库来演示使用 `groupby()` 来分组数据处理的局限性。

```
import numpy as np
import pandas as pd
import seaborn as sns
titanic = pd.read_csv('./input/titanic.csv')
titanic.head()
```

[2]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

这份数据包含了每位乘客的信息：性别(gender)、年龄(age)、船舱等级(class)和船票价格(fare paid)等。

任务：groupby()并非想象中的好用

需求分析：分析泰坦尼克号乘客的性别 `sex` 和船舱等级 `class` 与生存率 `survived` 关系。

首先，使用 `groupby()` 分析性别 `sex` 与生存率 `survived` 的关系。

```
# 根据`sex`关键列，对`survived`字段值进行求均值。
titanic.groupby('sex')['survived'].mean()


```

```
sex
female    0.742038
male      0.188908
Name: survived, dtype: float64
```

分组结果表明：有四分之三的女性被救，但只有五分之一的男性被救！

需求分析：希望能进一步分析性别 `sex` 和船舱等级 `class` 与生存率 `survived` 关系。根据 `groupby()` 流程，需要执行如下步骤：

- 首先，选择性别 `sex` 和船舱等级 `class` 作为关键列对数据进行分组。
- 然后，**选择**生存率 `survived` 字段，并调用**均值**数据分析函数。
- 最后，将各组结果**组合**，并通过**行索引转列索引**操作将最里层的行索引转换成列索引，形成数据表。

[4]:

```
# 选择性别`sex`和船舱等级`class`作为关键列对数据进行分组
g_df = titanic.groupby(['sex', 'class'])['survived']

# 并调用均值数据分析函数
res = g_df.agg(np.sum)
print(res)

# unstack() 按内(行)索引转置
print('------\n',res.unstack())
```

```
sex     class
female  First     91
        Second    70
        Third     72
male    First     45
        Second    17
        Third     47
Name: survived, dtype: int64
------
 class   First  Second  Third
sex
female     91      70     72
male       45      17     47
```

结果分析:

- 上述操作，不知觉中使用了 `groupby()` + `unstack()` 实现了**多维**分组数据处理的**数据透视表**功能。
- 虽然上述操作实现了性别 `sex` 和船舱等级 `class` 与生存率 `survived` 关系。
- 但是**代码看上去实在太复杂了**。对于数据处理中的常用功能，Pandas肯定会对其功能进行封装为函数。
- 为此，Pandas提供了便捷数据透视数据处理函数 `pivot_table()` 。

任务：pivot_table()一句语句搞定上述所有操作

Pandas为 `DataFrame` 提供了强大的**数据透视表**函数。定义如下：

```
# pandas 1.0.3版本
df.pivot_table(values=None, index=None, columns=None,
                   aggfunc='mean', fill_value=None, margins=False,
                   dropna=True, margins_name='All', observed=False)
```

> 知识点：参数功能解释
>
> - `values` ： 选择数据列，作为数据透视表的字段值。默认值为除关键列之外的剩余列。
> - `index` ： 关键列，作为数据透视表的行。
> - `columns` ： 关键列，作为数据透视表的列。
> - 上述3个参数，实际对应Excel数据透视表的4个数据操作区域。
> - `aggfunc` ： 对字段值的数据处理函数，默认值为 `mean` 求均值。
> - 注意： `index` 和 `columns` 分组关键列的字段值必须是**离散、可分组**的。

[13]:

```
# 简洁的pivot_table()实现了groupby()繁琐的操作
res = titanic.pivot_table(values = 'survived', index='sex', columns='class')
print(res)
print(pd.__version__)
```

```
class      First    Second     Third
sex
female  0.968085  0.921053  0.500000
male    0.368852  0.157407  0.135447
1.3.3
```

结果分析:

- 与 `groupby()` 相比， `pivot_table()` 代码可读性更强。
- 上述结果与20世纪初的那场灾难的猜想一致，生还率最高的是船舱等级高的女性。
- 一等舱的女性乘客基本全部生还(露丝自然得救)，而三等舱男性乘客的生还率仅为十分之一(杰克为爱牺牲)。

任务：pivot_table()参数与Excel的数据透视表

下面给出了 `pivot_table()` 参数与Excel数据透视表界面的对比。

2 深入使用pivot_table()函数

任务：关键列为连续值的数据透视表

需求：分析性别 `sex` 、年龄 `age` 和船舱等级 `class` 与生存率 `survived` 关系。

- 由于 `pivot_table()` 的分组关键列必须是离散值，而 `age` 列的值为连续值。
- 为此，需要通过调用 `pd.cut()` 函数将 `age` 年龄进行分段(`age` 连续值转化为离散区间，便于分组)。

```
age = pd.cut(titanic['age'], 3)
titanic.pivot_table(values = 'survived', index = ['sex', age], columns = 'class')
```

	class	First	Second	Third
sex	age			
female	(0.34, 26.947]	0.931034	0.933333	0.514706
	(26.947, 53.473]	0.978723	0.926829	0.333333
	(53.473, 80.0]	1.000000	0.666667	1.000000
male	(0.34, 26.947]	0.533333	0.270270	0.157143
	(26.947, 53.473]	0.453125	0.075472	0.149533
	(53.473, 80.0]	0.136364	0.111111	0.000000

结果分析:

- 透视表行索引： `sex` 和 `age` 字段，列索引： `class` 字段。
- 透视表展现了三维数据，行索引二维(二层索引)、列索引二维(一层索引)。

需求分析：分析性别 `sex` 、年龄 `age` 和船票 `fare` 、船舱等级 `class` 与生存率 `survived` 关系。

[11]:

```
fare = pd.qcut(titanic['fare'], 2)
titanic.pivot_table('survived', ['sex', age], [fare, 'class'])
```

fare	**(-0.001, 14.454]**				**(14.454, 512.329]**		
	class	First	Second	Third	First	Second	Third
sex	age						
female	(0.34, 26.947]	NaN	0.857143	0.609756	0.931034	0.956522	0.370370
	(26.947, 53.473]	NaN	0.950000	0.333333	0.978723	0.904762	0.333333
	(53.473, 80.0]	NaN	0.000000	1.000000	1.000000	1.000000	NaN
male	(0.34, 26.947]	NaN	0.052632	0.135922	0.533333	0.500000	0.216216
	(26.947, 53.473]	0.0	0.100000	0.155556	0.483333	0.043478	0.117647
	(53.473, 80.0]	NaN	0.166667	0.000000	0.136364	0.000000	NaN

结果分析：

- 透视表行索引： `sex` 和 `age` 字段，列索引： `fare` 和 `class` 字段。
- 透视表展现了四维数据，行索引二维(二层索引)、列索引二维(二层索引)。
- 数据透视过程中，如果某个分组不存在相应值，就会引起缺失值，例如， `First` 分组。

任务：fill_value和dropna参数缺失值处理

类似于， `fillna()` 与 `dropna()` ，在数据透视时，可以使用 `pivot_table()` 的 `fill_value` 和 `dropna` 参数来处理缺失值。

```
# pandas 1.0.3版本
df.pivot_table(values=None, index=None, columns=None,
                    aggfunc='mean', fill_value=None, margins=False,
                    dropna=True, margins_name='All', observed=False)
```

> 知识点： `fill_value` 使用任意预设值填充缺失值。

[12]:

```
titanic.pivot_table(values = 'survived', index = ['sex', age],
                    columns = [fare, 'class'], fill_value = 0.0)
```

fare	**(-0.001, 14.454]**				**(14.454, 512.329]**		
	class	First	Second	Third	First	Second	Third
sex	age						
female	(0.34, 26.947]	0	0.857143	0.609756	0.931034	0.956522	0.370370
	(26.947, 53.473]	0	0.950000	0.333333	0.978723	0.904762	0.333333
	(53.473, 80.0]	0	0.000000	1.000000	1.000000	1.000000	0.000000
male	(0.34, 26.947]	0	0.052632	0.135922	0.533333	0.500000	0.216216
	(26.947, 53.473]	0	0.100000	0.155556	0.483333	0.043478	0.117647
	(53.473, 80.0]	0	0.166667	0.000000	0.136364	0.000000	0.000000

> 知识点：
>
> - dropna 参数为布尔值。
> - 如果为 True 时，某列的字段值**全部**为 NaN 时，丢弃该列。否则，保留该列。

```
titanic.pivot_table(values = 'survived', index = ['sex', age],
                    columns = [fare, 'class'], dropna = True)
```

fare	(-0.001, 14.454]				(14.454, 512.329]		
	class	First	Second	Third	First	Second	Third
sex	age						
female	(0.34, 26.947]	NaN	0.857143	0.609756	0.931034	0.956522	0.370370
	(26.947, 53.473]	NaN	0.950000	0.333333	0.978723	0.904762	0.333333
	(53.473, 80.0]	NaN	0.000000	1.000000	1.000000	1.000000	NaN
male	(0.34, 26.947]	NaN	0.052632	0.135922	0.533333	0.500000	0.216216
	(26.947, 53.473]	0.0	0.100000	0.155556	0.483333	0.043478	0.117647
	(53.473, 80.0]	NaN	0.166667	0.000000	0.136364	0.000000	NaN

任务：aggfunc参数指定数据处理函数

> 知识点：
>
> - `aggfunc` 参数用于设置聚合函数类型，默认值是均值(`mean`)
> - 类似于 `groupby()` ，该参数传递**聚合函数**，例如 `np.sum()` 、 `min()` 、 `sum()` 等。
> - 或**聚合函数名字符串**，例如 `'sum'` 、 `'mean'` 、 `'count'` 、 `'min'` 、 `'max'` 等。

> 知识点：多个字段执行不同的数据处理函数时，传递**字典**。此时，**不需要**指定 `values` 参数。

```
titanic.pivot_table(index='sex', columns='class',
                    aggfunc={'survived':sum, 'fare':'mean'})
```

	fare			survived		
class	First	Second	Third	First	Second	Third
sex						
female	106.125798	21.970121	16.118810	91	70	72
male	67.226127	19.741782	12.661633	45	17	47

任务：margins参数指定是否汇总

当需要计算每一组的总数时，可以通过 `margins` 参数来设置。

```
titanic.pivot_table(values= 'survived', index='sex',
                    columns='class', margins=True)
```

class	First	Second	Third	All
sex				
female	0.968085	0.921053	0.500000	0.742038
male	0.368852	0.157407	0.135447	0.188908
All	0.629630	0.472826	0.242363	0.383838

结果分析：

- 获取了不同性别下船舱等级与生还率的相关信息、不同船舱等级下性别与生还率的相关信息。
- 全部乘客的生还率为38%。
- `margin` 对应的标签可以通过 `margins_name` 参数进行自定义，默认值是 `"All"` 。

```
titanic.pivot_table(values= 'survived', index='sex',
                    columns='class', margins=True, margins_name='total')
```

class	First	Second	Third	total
sex				
female	0.968085	0.921053	0.500000	0.742038
male	0.368852	0.157407	0.135447	0.188908
total	0.629630	0.472826	0.242363	0.383838

任务：pivot_table()和groupby()比较

> 知识点：
>
> - 在**行**上分组， `pivot_table()` 在只用 `index` 和 `values` 参数时，和 `groupby()` 可以完全等价。
> - 此外， `pivot_table()` 额外引入了 `columns` 与 `margin` 参数，实现了在**列**上的分组和数据的汇总功能。
> - 因此，相对于 `groupby()` ，它的用法更加灵活。

```
# 行分组的等价代码
pf.pivot_table(index=[字段1],values=[字段2],aggfunc=[函数],fill_value=0)

df.groupby([字段1])[字段2].agg(函数).fillna(0)
```

3 案例：火箭队James赛季比赛数据分析

任务：数据导入

本节将对火箭队当家吉祥物James_Harden赛季比赛数据作为数据进行数据透视表分析。

```
import pandas as pd
import numpy as np
df_James = pd.read_csv('./input/James_Harden.csv',encoding='utf8')
df_James.tail()
```

	对手	胜负	主客场	命中	投篮数	投篮命中率	3分命中率	篮板	助攻	得分
0	勇士	胜	客	10	23	0.435	0.444	6	11	27
1	国王	胜	客	8	21	0.381	0.286	3	9	27
2	小牛	胜	主	10	19	0.526	0.462	3	7	29
3	灰熊	负	主	8	20	0.400	0.250	5	8	22
4	76人	胜	客	10	20	0.500	0.250	3	13	27

`pivot_table()` 有四个最重要的参数 `values` 、 `index` 、 `columns` 、 `aggfunc` ，本节将重点围绕这四个参数，讲解数据透视表功能。

任务：Index参数使用

每个 `pivot_table()` 必须拥有一个 `index` 。希望查看James对阵每个队伍的得分，首先将 `对手` 字段设置为 `index` 。

```
# u字符串前缀是因为使用了中文，避免乱码
# 需要对中文使用Unicode编码方式
df_James.pivot_table(index=[u'对手'])
```

	3分命中率	助攻	命中	得分	投篮命中率	投篮数	篮板
对手							
76人	0.33950	10.00	9.0	28.00	0.4405	20.5	3.5
勇士	0.44400	11.00	10.0	27.00	0.4350	23.0	6.0
国王	0.28600	9.00	8.0	27.00	0.3810	21.0	3.0
太阳	0.54500	7.00	12.0	48.00	0.5450	22.0	2.0
小牛	0.46200	7.00	10.0	29.00	0.5260	19.0	3.0
尼克斯	0.36900	9.50	10.5	34.00	0.4175	25.0	3.5
开拓者	0.57100	3.00	16.0	48.00	0.5520	29.0	8.0
掘金	0.14300	9.00	6.0	21.00	0.3750	16.0	8.0
步行者	0.29150	12.50	8.5	27.50	0.3965	21.5	6.5
湖人	0.44400	9.00	13.0	36.00	0.5910	22.0	4.0
灰熊	0.35025	7.75	8.5	27.25	0.4015	21.0	4.5
爵士	0.60400	8.00	13.5	42.50	0.5905	22.0	3.5
猛龙	0.27300	11.00	8.0	38.00	0.3200	25.0	6.0
篮网	0.61500	8.00	13.0	37.00	0.6500	20.0	10.0
老鹰	0.54500	11.00	8.0	29.00	0.5330	15.0	3.0
骑士	0.42900	13.00	8.0	35.00	0.3810	21.0	11.0
鹈鹕	0.40000	17.00	8.0	26.00	0.5000	16.0	1.0
黄蜂	0.40000	11.00	8.0	27.00	0.4440	18.0	10.0

上面将 `对手` 成为了第一层索引。如果还想分析对阵同一对手在不同 `主客场` 下的数据，试着将 `对手` 与 `主客场` 都设置为 `index` 。

```
# 多层分组
df_James.pivot_table(index=[u'对手',u'主客场'])
```

		3分命中率	助攻	命中	得分	投篮命中率	投篮数	篮板
对手	主客场							
76人	主	0.4290	7.0	8.0	29.0	0.381	21.0	4.0
客	0.2500	13.0	10.0	27.0	0.500	20.0	3.0	
勇士	客	0.4440	11.0	10.0	27.0	0.435	23.0	6.0
国王	客	0.2860	9.0	8.0	27.0	0.381	21.0	3.0
太阳	客	0.5450	7.0	12.0	48.0	0.545	22.0	2.0
小牛	主	0.4620	7.0	10.0	29.0	0.526	19.0	3.0
尼克斯	主	0.3850	10.0	12.0	37.0	0.444	27.0	2.0
客	0.3530	9.0	9.0	31.0	0.391	23.0	5.0	
开拓者	客	0.5710	3.0	16.0	48.0	0.552	29.0	8.0
掘金	主	0.1430	9.0	6.0	21.0	0.375	16.0	8.0
步行者	主	0.3330	10.0	8.0	29.0	0.364	22.0	8.0
客	0.2500	15.0	9.0	26.0	0.429	21.0	5.0	
湖人	客	0.4440	9.0	13.0	36.0	0.591	22.0	4.0
灰熊	主	0.3395	8.0	9.5	30.0	0.420	22.5	4.5
客	0.3610	7.5	7.5	24.5	0.383	19.5	4.5	
爵士	主	0.8750	13.0	19.0	56.0	0.760	25.0	2.0
客	0.3330	3.0	8.0	29.0	0.421	19.0	5.0	
猛龙	主	0.2730	11.0	8.0	38.0	0.320	25.0	6.0
篮网	主	0.6150	8.0	13.0	37.0	0.650	20.0	10.0
老鹰	客	0.5450	11.0	8.0	29.0	0.533	15.0	3.0
骑士	主	0.4290	13.0	8.0	35.0	0.381	21.0	11.0
鹈鹕	主	0.4000	17.0	8.0	26.0	0.500	16.0	1.0
黄蜂	客	0.4000	11.0	8.0	27.0	0.444	18.0	10.0

```
# 交换index的顺序
# 虽然数据结果一样，但分组形式会变的不一样
df_James.pivot_table(index=[u'主客场',u'对手'])
```

		3分命中率	助攻	命中	得分	投篮命中率	投篮数	篮板
主客场	对手							
主	76人	0.4290	7.0	8.0	29.0	0.381	21.0	4.0
	小牛	0.4620	7.0	10.0	29.0	0.526	19.0	3.0
	尼克斯	0.3850	10.0	12.0	37.0	0.444	27.0	2.0
	掘金	0.1430	9.0	6.0	21.0	0.375	16.0	8.0
	步行者	0.3330	10.0	8.0	29.0	0.364	22.0	8.0
	灰熊	0.3395	8.0	9.5	30.0	0.420	22.5	4.5
	爵士	0.8750	13.0	19.0	56.0	0.760	25.0	2.0
	猛龙	0.2730	11.0	8.0	38.0	0.320	25.0	6.0
	篮网	0.6150	8.0	13.0	37.0	0.650	20.0	10.0
	骑士	0.4290	13.0	8.0	35.0	0.381	21.0	11.0
	鹈鹕	0.4000	17.0	8.0	26.0	0.500	16.0	1.0
客	76人	0.2500	13.0	10.0	27.0	0.500	20.0	3.0
	勇士	0.4440	11.0	10.0	27.0	0.435	23.0	6.0
	国王	0.2860	9.0	8.0	27.0	0.381	21.0	3.0
	太阳	0.5450	7.0	12.0	48.0	0.545	22.0	2.0
	尼克斯	0.3530	9.0	9.0	31.0	0.391	23.0	5.0
	开拓者	0.5710	3.0	16.0	48.0	0.552	29.0	8.0
	步行者	0.2500	15.0	9.0	26.0	0.429	21.0	5.0
	湖人	0.4440	9.0	13.0	36.0	0.591	22.0	4.0
	灰熊	0.3610	7.5	7.5	24.5	0.383	19.5	4.5
	爵士	0.3330	3.0	8.0	29.0	0.421	19.0	5.0
	老鹰	0.5450	11.0	8.0	29.0	0.533	15.0	3.0
	黄蜂	0.4000	11.0	8.0	27.0	0.444	18.0	10.0

总之， `index` 参数就是层次分组的字段，要通过透视表获取什么信息就按照相应的顺序设置该 `index` 字段。所以在进行pivot之前，首先要想好从什么角度，选择什么字段作为 `index` 来展现数据。

任务：values参数使用

通过上面的操作，获取了james在对阵对手时的所有数据，而values参数，可以筛选指定需要计算的数据。例如，需要统计james在 主客场 和不同 胜负 情况下的 得分 、 篮板 与 助攻 三项数据。

```
df_James.pivot_table(index=[u'主客场',u'胜负'],
                     values=[u'得分',u'助攻',u'篮板'])
```

		助攻	得分	篮板
主客场	胜负			
主	胜	10.555556	34.222222	5.444444
	负	8.666667	29.666667	5.000000
客	胜	9.000000	32.000000	4.916667
	负	8.000000	20.000000	4.000000

任务：aggfunc参数

`aggfunc` 参数可以设置对数据分析时进行的函数操作。当未设置aggfunc时，默认为 `aggfunc='mean'` 计算数据的均值。

希望获得james在 主客场 和不同 胜负 情况下的平均和总 得分 、 篮板 、 助攻 。

```
df_James.pivot_table(index=[u'主客场',u'胜负'],
                     values=[u'得分',u'助攻',u'篮板'],
                     aggfunc=[np.sum,np.mean])
```

		sum			mean		
		助攻	得分	篮板	助攻	得分	篮板
主客场	胜负						
主	胜	95	308	49	10.555556	34.222222	5.444444
	负	26	89	15	8.666667	29.666667	5.000000
客	胜	108	384	59	9.000000	32.000000	4.916667
	负	8	20	4	8.000000	20.000000	4.000000

任务：columns参数

`columns` 参数类似 `index` 可以设置**列层次字段**。

```
# fill_value填充空值
# margins=True进行汇总，结果里多了个All字段
df_James.pivot_table(index=[u'主客场'],columns=[u'胜负'],
                     values=[u'得分',u'助攻',u'篮板'],
                     aggfunc=[np.sum,np.mean],
                     fill_value=0,margins=1)
```

	sum		mean															
	助攻	得分	篮板	助攻	得分	篮板												
胜负	胜	负	All	胜	负	All	胜	负	All	胜	负	All	胜	负	All	胜	负	All
主客场																		
主	95	26	121	308	89	397	49	15	64	10.555556	8.666667	10.083333	34.222222	29.666667	33.083333	5.444444	5.00	5.333333
客	108	8	116	384	20	404	59	4	63	9.000000	8.000000	8.923077	32.000000	20.000000	31.076923	4.916667	4.00	4.846154
All	203	34	237	692	109	801	108	19	127	9.666667	8.500000	9.480000	32.952381	27.250000	32.040000	5.142857	4.75	5.080000

4 案例：美国人的生日

任务：数据导入

本节对美国人生日 `births.csv` 数据进行数据透视表分析。

```
# shell下载数据
#!curl -O https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv
```

```
import numpy as np
import pandas as pd
import seaborn as sns

# 导入births.csv数据
births = pd.read_csv('./input/births.csv')
```

任务：观察数据相关信息

导入数据后，通常会从宏观的角度上，观察数据相关统计信息。

```
# 观察数据的规模(15547, 5)
print(births.shape)

# 浏览前5条数据，观察包含的字段
births.head()

```

```
(15547, 5)
```

	year	month	day	gender	births
0	1969	1	1.0	F	4046
1	1969	1	1.0	M	4440
2	1969	1	2.0	F	4454
3	1969	1	2.0	M	4548
4	1969	1	3.0	F	4548

结果表明：生日数据比较简单，每条数据只包含出生日期、性别、出生人数的字段信息。

```
# 描述数据的统计属性
births.describe()
```

	year	month	day	births
count	15547.000000	15547.000000	15067.000000	15547.000000
mean	1979.037435	6.515919	17.769894	9762.293561
std	6.728340	3.449632	15.284034	28552.465810
min	1969.000000	1.000000	1.000000	1.000000
25%	1974.000000	4.000000	8.000000	4358.000000
50%	1979.000000	7.000000	16.000000	4814.000000
75%	1984.000000	10.000000	24.000000	5289.500000
max	2008.000000	12.000000	99.000000	199622.000000

结果分析：发现year、month、day字段并无实际意义。有用是births字段，可以得到最大、最小出生数等信息。

任务：分析不同年代的男女出生比例

需求分析：

- 首先，要根据 `year` 字段，为数据添加一个新的 `decade` 字段，10年为一个年代。
- 然后，分析年代 `decade` 和性别 `gender` 与生存率 `births` 关系。

```
# 将year转换为decade，比如1969->1960，去掉最后个位
births['decade'] = 10 * (births['year'] // 10)
print(births.head(),'\n')
births_pivot = births.pivot_table(values = 'births', index='decade',
                                  columns='gender', aggfunc='sum')
births_pivot
```

```
   year  month  day gender  births  decade
0  1969      1  1.0      F    4046    1960
1  1969      1  1.0      M    4440    1960
2  1969      1  2.0      F    4454    1960
3  1969      1  2.0      M    4548    1960
4  1969      1  3.0      F    4548    1960
```

gender	F	M
decade		
1960	1753634	1846572
1970	16263075	17121550
1980	18310351	19243452
1990	19479454	20420553
2000	18229309	19106428

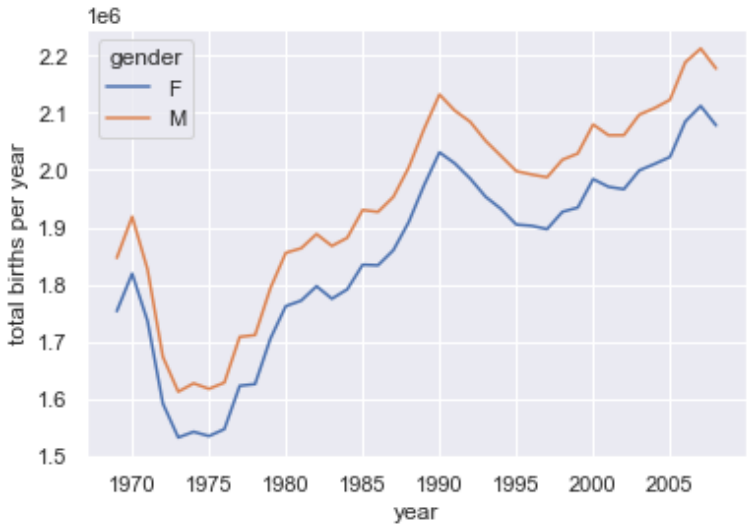
结果表明：每个年代的男性出生率都比女性出生率高。

任务：绘制出生人数趋势图

如果希望更直观地体现这种趋势，可以用Pandas内置的画图功能，将每一年的出生人数画出来。

```
%matplotlib inline
import matplotlib.pyplot as plt
sns.set()  # 使用Seaborn风格

# 类似于groupby(), 对分组结果串联plot()函数绘制曲线
births.pivot_table(values='births', index='year', columns='gender',
aggfunc='sum').plot()
plt.ylabel('total births per year');
```



结果分析：

- 使用数据透视表和 `plot()` ，直观展现了不同性别出生率的趋势。

- 发现过去50年间的男性出生率比女性出生率高5%。

任务：query()选择中位数附近的数据

需求分析：根据 `births` 字段，选择中位数上下一个方差的数据。

- 调用 `np.percentile()` 函数，对 `births` 字段进行25%、50%、75%分位数计算。
- `mu` ：中位数，为50%分位数。
- `sig` ：方差，0.74是指标准正态分布的分位数间距。

```
quartiles = np.percentile(births['births'], [25, 50, 75])
mu = quartiles[1]  # 中位数
sig = 0.74 * (quartiles[2] - quartiles[0]) # 方差

# f-string方式输出mu和sig
print(f'mu:{mu:.2f}, sig:{sig:.2f}')
```

```
mu:4814.00, sig:689.31
```

设定指定条件，调用在 `query()` 筛选满足条件的数据。

```
# 数据筛选条件，f-string，向字符串传递数据
sel = f'(births > {mu} - {sig}) & (births < {mu} + {sig})'
print(sel)

# 调用query()查询满足条件的数据，发挥Dataframe
births = births.query(sel)

# 输出满足条件数据的规模
print(births.shape)

births.head()
```

```
(births > 4814.0 - 689.31) & (births < 4814.0 + 689.31)
(10959, 6)
```

	year	month	day	gender	births	decade
1	1969	1	1.0	M	4440	1960
2	1969	1	2.0	F	4454	1960
3	1969	1	2.0	M	4548	1960
4	1969	1	3.0	F	4548	1960
5	1969	1	3.0	M	4994	1960

任务：不同年代不同星期的日均出生数据

由于 `day` 字段包含了缺失值，所以Dataframe将其存为了是字符串。下面首先将其转换为整数。

```
# 将'day'列设置为整数. 由于其中含有缺失值null, 因此是字符串
# 如果出现警告, 请忽略
births['day'] = births['day'].astype(int)
```

将 `year` 、 `month` 、 `day` 字段组合为一个日期索引。这样就可以使用Pandas提供的日期数据处理方法，根据日期获得星期几信息。

```
# 从年月日字段中创建一个日期索引
births.index = pd.to_datetime(10000 * births.year +
                              100 * births.month +
                              births.day, format='%Y%m%d')

print(births.index)

# 通过dayofweek属性获得星期几信息
births['dayofweek'] = births.index.dayofweek
```

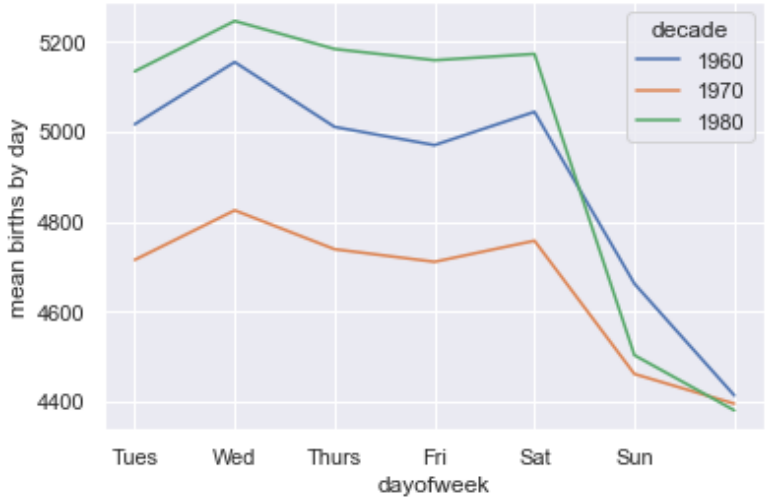
```
DatetimeIndex(['1969-01-01', '1969-01-02', '1969-01-02', '1969-01-03',
               '1969-01-03', '1969-01-04', '1969-01-04', '1969-01-05',
               '1969-01-05', '1969-01-06',
               ...
               '1988-12-18', '1988-12-21', '1988-12-22', '1988-12-23',
               '1988-12-23', '1988-12-24', '1988-12-26', '1988-12-26',
               '1988-12-31', '1988-12-31'],
              dtype='datetime64[ns]', length=10959, freq=None)
```

需求分析：

- 根据 `dayofweek` 和 `decade` 字段，对 `births` 字段建立数据透视表。
- 然后，将结果绘制出不同年代不同星期的日均出生数据趋势。

```
births.pivot_table(values = 'births', index='dayofweek',
                   columns='decade', aggfunc='mean').plot()

# 下面是matplotlib模块的图坐标与标签设置
plt.gca().set_xticklabels(['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'])
plt.ylabel('mean births by day');
```



结果分析：

- 周末的出生人数比工作日要低很多。
- 另外，因为CDC机构只提供了1989年之前的数据，所以没有20世纪90年代和21世纪的数据。

任务：分析各个年份的日均出生人数

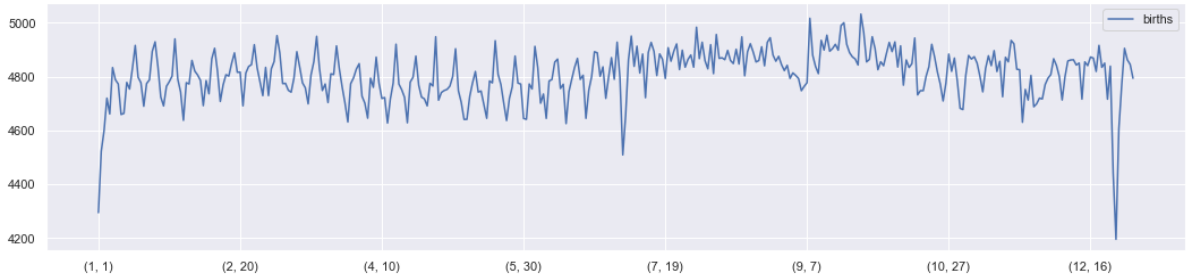
需求分析：对所有年份的 `births` 字段数据，根据 `index.month` 和 `index.day` 进行分组。

```
births_by_date = births.pivot_table(values = 'births',
                                    index = [births.index.month, births.index.day])
births_by_date
```

		births
1	1	4294.466667
2	4522.318182	
3	4599.264706	
4	4720.030303	
5	4661.333333	
...	...	...
12	27	4766.757576
28	4905.240000	
29	4862.520000	
30	4845.444444	
31	4793.962963	

366 rows × 1 columns

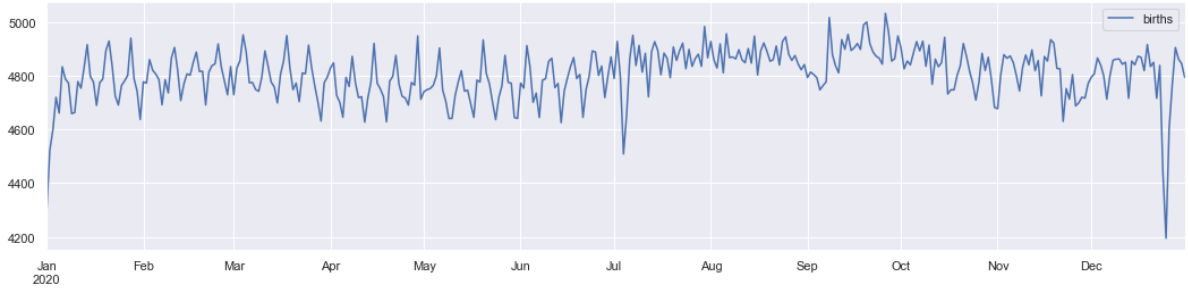
```
# 调整绘图大小 figsize=(18, 4)
fig, ax = plt.subplots(figsize=(18, 4))
births_by_date.plot(ax=ax);
```



结果分析：

- 这是一个包含月和日的多级索引。
- 发现x轴坐标是 `(月,日)` 元组。为了提供更好的显示界面，下面对创建使用 `datetime` 模块，将 `(月,日)` 元组组织到一年时间中，构成新的时间索引。

```
import datetime as dt
births_by_date.index = [dt.datetime(2020, month, day)
                        for (month, day) in births_by_date.index]
births_by_date.head()
# 将结果画成图
fig, ax = plt.subplots(figsize=(18, 4))
births_by_date.plot(ax=ax);
```



结果分析：

- 相对于之前的绘图，此图的x轴坐标直接标明了月份，可以很方便的看出，一年时间内的出生率变化趋势。
- 在美国节假日的时候，出生人数急速下降(例如美国独立日、劳动节、感恩节、圣诞节以及新年)。
- 这种现象可能是由于医院放假导致的接生减少(自己在家生)，而非某种自然生育的心理学效应。