```
import numpy as np
import pandas as pd
```

为了方便维护，一般公司的数据在数据库内都是分表存储的，比如用一个表存储所有用户的基本信息，一个表存储用户的消费情况。所以，在日常的数据处理中，经常需要将两张表拼接起来使用，这样的操作对应到SQL中是join，在Pandas中则是用merge来实现。这篇文章就讲一下merge的主要原理。

上面的引入部分说到merge是用来拼接两张表的，那么拼接时自然就需要将用户信息一一对应地进行拼接，所以进行拼接的两张表需要有一个共同的识别用户的键（key）。总结来说，整个merge的过程就是将信息一一对应匹配的过程，下面介绍merge的四种类型，分别为`inner`、`left`、`right`和`outer`。

merge() 函数的法格式如下：

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

- `left/right`　两个不同的 DataFrame 对象。
- `how`　要执行的合并类型，从 {'left', 'right', 'outer', 'inner'} 中取值，默认为“inner”内连接。

- `on` 指定用于连接的键（即列标签的名字），该键必须同时存在于左右两个 DataFrame 中，如果没有指定，并且其他参数也未指定， 那么将会以两个 DataFrame 的列名交集做为连接键。
- `left_on` 指定左侧 DataFrame 中作连接键的列名。该参数在左、右列标签名不相同，但表达的含义相同时非常有用。
- `right_on` 指定左侧 DataFrame 中作连接键的列名。
- `left_index` 布尔参数，默认为 False。如果为 True 则使用左侧 DataFrame 的行索引作为连接键
- `right_index` 布尔参数，默认为 False。如果为 True 则使用左侧 DataFrame 的行索引作为连接键
- `sort` 布尔值参数， False，则按照 how 给定的参数值进行排序。设置为 True，它会将合并后的数据进行排序；
- `suffixes` 字符串组成的元组。当左右 DataFrame 存在相同列名时，通过该参数可以在相同的列名后附加后缀名，默认为('*x*','y')。
- `copy` 默认为 True，表示对数据进行复制。

> 注意：Pandas 库的 merge() 支持各种内外连接，与其相似的还有 join() 函数（默认为左连接）。

一、inner

merge的`inner`的类型称为内连接，它在拼接的过程中会取两张表的键（key）的交集进行拼接。什么意思呢？

下面以图解的方式来一步一步拆解。

首先我们有以下的数据，左侧和右侧的数据分别代表了用户的基础信息和消费信息，连接两张表的键是userid。

df_1：用户基础信息

	userid	age
0	a	23
1	b	46
2	c	32
3	d	19

df_2：用户消费信息

	userid	payment
0	a	2000
1	c	3500

```
df_1 = pd.DataFrame({
                      "userid":['a', 'b', 'c', 'd'],
                      "age":[23, 46, 32, 19]
                     })
df_1
```

	userid	age
0	a	23
1	b	46
2	c	32
3	d	19

```
df_2 = pd.DataFrame({
        "userid":['a', 'c'],
        "payment":[2000, 3500]
    })
df_2
```

	userid	payment
0	a	2000
1	c	3500

```
df_1.merge(df_2,on='userid')
```

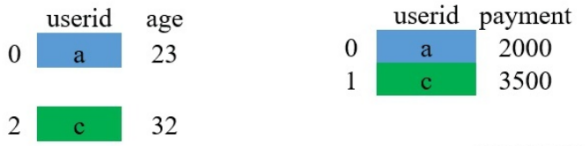
	userid	age	payment
0	a	23	2000
1	c	32	3500

```
pd.merge(df_1, df_2, on='userid')
```

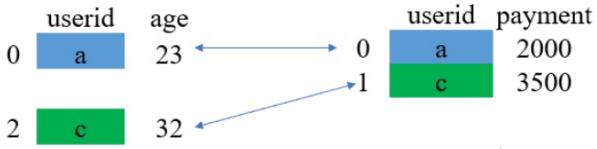
	userid	age	payment
0	a	23	2000
1	c	32	3500

过程图解：

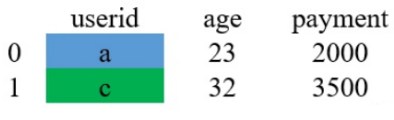
①取两张表的键的交集，这里df_1和df_2的userid的交集是{a,c}



②对应匹配



③结果



相信整个过程并不难理解，上面演示的是同一个键下，两个表对应只有一条数据的情况（一个用户对应一条消费记录），那么，如果一个用户对应了多条消费记录的话，那又是怎么拼接的呢？

假设现在的数据变成了下面这个样子，在df_2中，有两条和a对应的数据：

df_1：用户基础信息

	userid	age
0	a	23
1	b	46
2	c	32
3	d	19

df_2：用户消费信息

	userid	payment
0	a	2000
1	c	3500
2	a	500
3	b	1000

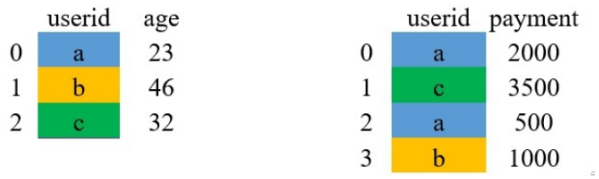
```
# 同样用inner的方式进行merge:
df_1 = pd.DataFrame({
                     "userid":['a', 'b', 'c', 'd'],
                     "age":[23, 46, 32, 19]
                    })

df_2 = pd.DataFrame({
        "userid":['a', 'c','a', 'd'],
        "payment":[2000, 3500, 500, 1000]
    })
pd.merge(df_1, df_2, on="userid")
```

	userid	age	payment
0	a	23	2000
1	a	23	500
2	c	32	3500
3	d	19	1000

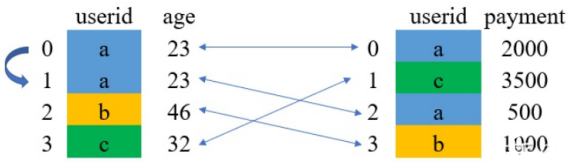
整个过程除了对应匹配阶段，其他和上面基本都是一致的。

过程图解：

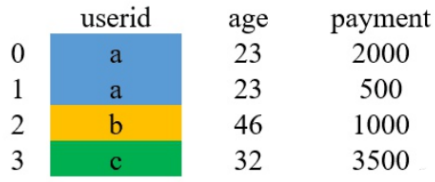
①取两张表的键的交集，这里df_1和df_2的userid的交集是{a,b,c}



②对应匹配时，由于这里的a有两条对应的消费记录，故在拼接时，会将用户基础信息表中a对应的数据复制多一行来和右边进行匹配。



③结果



二、left 和right

'left'和'right'的merge方式其实是类似的，分别被称为左连接和右连接。这两种方法是可以互相转换的，所以在这里放在一起介绍。

```
'left'
```

merge时，以左边表格的键为基准进行配对，如果左边表格中的键在右边不存在，则用缺失值NaN填充。

```
'right'
```

merge时，以右边表格的键为基准进行配对，如果右边表格中的键在左边不存在，则用缺失值NaN填充。

什么意思呢？用一个例子来具体解释一下，这是演示的数据

df_1：用户基础信息

	userid	age
0	a	23
1	b	46
2	c	32
3	d	19

df_2：用户消费信息

	userid	payment
0	a	2000
1	c	3500
2	e	600

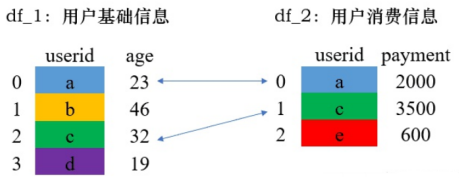
现在用'left'的方式进行merge

```
df_1 = pd.DataFrame({
                     "userid":['a', 'b', 'c', 'd'],
                     "age":[23, 46, 32, 19]
                    })

df_2 = pd.DataFrame({
        "userid":['a', 'c','e'],
        "payment":[2000, 3500, 600]
    })
pd.merge(df_1, df_2,how='left', on="userid")
```

	userid	age	payment
0	a	23	2000.0
1	b	46	NaN
2	c	32	3500.0
3	d	19	NaN

过程图解：

①以左边表格的所有键为基准进行配对。图中，因为右表中的e不在左表中，故不会进行配对。



②若右表中的payment列合并到左表中，对于没有匹配值的用缺失值NaN填充

	userid	age	payment
0	a	23	2000
1	b	46	NaN
2	c	32	3500
3	d	19	NaN

对于'right'类型的merge和'left'其实是差不多的，只要把两个表格的位置调换一下，两种方式返回的结果就是一样的（），如下：

```
pd.merge(df_1, df_2,how='right', on="userid")
```

	userid	age	payment
0	a	23.0	2000
1	c	32.0	3500
2	e	NaN	600

三、outer

'outer'是外连接，在拼接的过程中它会取两张表的键（key）的并集进行拼接。看文字不够直观，还是上例子吧！

还是使用上方用过的演示数据

df_1：用户基础信息

	userid	age
0	a	23
1	b	46
2	c	32
3	d	19

df_2：用户消费信息

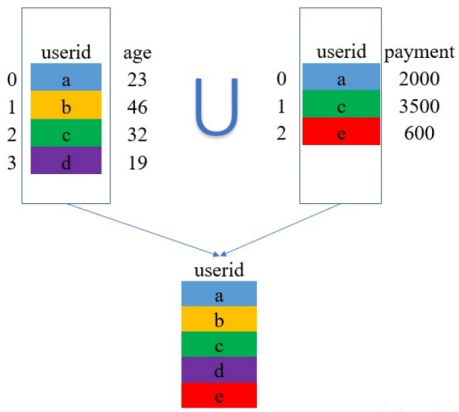
	userid	payment
0	a	2000
1	c	3500
2	e	600

```
1  pd.merge(df_1, df_2,how='outer',on='userid')
```

	userid	age	payment
0	a	23.0	2000.0
1	b	46.0	NaN
2	c	32.0	3500.0
3	d	19.0	NaN
4	e	NaN	600.0

图解如下：

①取两张表键的并集，这里是{a,b,c,d,e}



其他

set_index函数详解

专门用来将某一列设置为index的方法

`DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False)`

- `keys` ：要设置为索引的列名（如有多个应放在一个列表里）
- `drop` ：将设置为索引的列删除，默认为True
- `append` ：是否将新的索引追加到原索引后（即是否保留原索引），默认为False
- `inplace` ：是否在原DataFrame上修改，默认为False
- `verify_integrity` ：是否检查索引有无重复，默认为False

```
df = pd.DataFrame({'month': [1, 4, 7, 10],
                   'year': [2012, 2014, 2013, 2014],
                   'sale': [55, 40, 84, 31]})
df
```

	month	year	sale
0	1	2012	55
1	4	2014	40
2	7	2013	84
3	10	2014	31

```
#将索引设置为“month”列：
df.set_index('month')
```

	year	sale
month		
1	2012	55
4	2014	40
7	2013	84
10	2014	31

```
# 将month列设置为index之后，并保留原来的列
df.set_index('month',drop=False)
```

	month	year	sale
month			
1	1	2012	55
4	4	2014	40
7	7	2013	84
10	10	2014	31

```
# 保留原来的index列
df.set_index('month', append=True)
df
```

	month	year	sale
0	1	2012	55
1	4	2014	40
2	7	2013	84
3	10	2014	31

```
# 使用inplace参数取代原来的对象
df.set_index('month', inplace=True)
df
```

	year	sale
month		
1	2012	55
4	2014	40
7	2013	84
10	2014	31

```
# 通过新建Series并将其设置为index
df.set_index(pd.Series(range(4)))
```

	month	year	sale
0	1	2012	55
1	4	2014	40
2	7	2013	84
3	10	2014	31

Pandas去重函数： drop_duplicates()

“去重”通过字面意思不难理解，就是删除重复的数据。在一个数据集中，找出重复的数据删并将其删除，最终只保存一个唯一存在的数据项，这就是数据去重的整个过程。删除重复数据是数据分析中经常会遇到的一个问题。通过数据去重，不仅可以节省内存空间，提高写入性能，还可以提升数据集的精确度，使得数据集不受重复数据的影响。

Panda DataFrame 对象提供了一个数据去重的函数 drop_duplicates()

`DataFrame.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)`

- `subset` ： 表示要进去重的列名，默认为 None。
- `keep` ： 有三个可选参数，分别是 first、last、False，默认为 first，表示只保留第一次出现的重复项，删除其余重复项，last 表示只保留最后一次出现的重复项，False 则表示删除所有重复项
- `inplace` ： 布尔值参数，默认为 False 表示删除重复项后返回一个副本，若为 Ture 则表示直接在原数据上删除重复项

```
df = pd.DataFrame({
    'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
    'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 3.5, 15, 5]

})
df
```

	brand	style	rating
0	Yum Yum	cup	4.0
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0

```
# 默认情况下，它会基于所有列删除重复的行
df.drop_duplicates()
```

	brand	style	rating
0	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0

```
# 删除特定列上的重复项，使用子集
df.drop_duplicates(subset=['brand'])
```

	brand	style	rating
0	Yum Yum	cup	4.0
2	Indomie	cup	3.5

```
# 删除重复项并保留最后出现的项，请使用“保留”。
df.drop_duplicates(subset=['brand', 'style'], keep='last')
```

	brand	style	rating
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
4	Indomie	pack	5.0

tolist()

pandas的tolist()函数用于将一个系列或数据帧中的列转换为列表。

```
df.index
```

```
[0, 1, 2, 3, 4]
```

```
df.index.tolist()
```

```
[0, 1, 2, 3, 4]
```

```
df['brand']
```

```
0    Yum Yum
1    Yum Yum
2    Indomie
3    Indomie
4    Indomie
Name: brand, dtype: object
```

```
dict(df['brand'])
```

```
{0: 'Yum Yum', 1: 'Yum Yum', 2: 'Indomie', 3: 'Indomie', 4: 'Indomie'}
```

```
df['brand'].to_dict()
```

```
{0: 'Yum Yum', 1: 'Yum Yum', 2: 'Indomie', 3: 'Indomie', 4: 'Indomie'}
```

```

```