Pandas学习笔记

一、Pandas概述

Pandas是另外一个用于处理高级数据结构和数据分析的Python库，Pandas是基于Numpy构建的一种工具，，纳入了大量的模块和库一些标准数据模型，提高了Python处理大数据的性能。

特点：

- DataFrame是一种高效快速的数据结构模式，Pandas支持DataFrame格式，从而可以自定义索引
- 可以将不同格式的数据文件加载到内存中
- 未对齐及其索引方式不同的数据可按轴自动对齐
- 可处理时间序列或非时间序列数据
- 可基于标签来切片索引，获得大数据集子集
- 可进行高性能数据分组、聚合、添加、删除
- 灵活处理数据缺失、重组、空格

Pandas广泛用于金融、经济、数据分析、统计等商业领域，为各个领域数据从业者提供了便捷。

Pandas的安装与Numpy相似，如果你已经安装了Anaconda，那么直接导入即可，

安装命令：

```
pip install pandas
```

导入命令 起个别名

```
import pandas as pd
```

二、Pandas数据结构

Pandas被广泛使用的数据结构主要有Series和DataFrame,两者皆为python进行数据分析提供了基础

2.1 Series

series类似于一维数组，由一组数据产生，Series数组由数据和索引标签组成，索引在左侧，值在右侧。创建Series,可以使用Series函数。

（1）创建Series数组

```
import pandas as pd

# 将列表作为数据导入 转换成Series
s1 = pd.Series([1,2,3,4,5])
```

```
print('s1:{}'.format(s1))  # 格式化字符串函数 str.format ()
```

```
s1:0    1
1    2
2    3
3    4
4    5
dtype: int64
```

补充：这里用到了格式化字符串的函数str.format(),增强了字符串格式化的功能，其基本语法是通过{} 和 ： 来代替之前的% ， format函数可以接受若干个参数，位置可以不按照顺序

左边的列表表示索引，右边的列表表示值，默认从0开始创建索引，可以指定索引，设置index参数。

```
import pandas as pd
s2 = pd.Series([1,2,3,4,5],index = ['第一','第二','第三','第四','第五'])
print('s2 : {}'.format(s2))
```

```
s2:第一    1
第二    2
第三    3
第四    4
第五    5
dtype: int64
```

这里，指定了参数index,自行设置了索引

（2）Series的索引和切片

通过Series的values和index属性可以获取Series中的索引和数值

查看Series的索引和数值

```
import pandas as pd
s2 = pd.Series([1,2,3,4,5],index = ['第一','第二','第三','第四','第五'])
print('s2:{}'.format(s2))
print('s2索引: {}'.format(s2.index))
print('s2数值: {}'.format(s2.values))
```

```
s2:第一    1
第二    2
```

```
第三    3
第四    4
第五    5
dtype: int64
s2索引: Index(['第一', '第二', '第三', '第四', '第五'], dtype='object')
s2数值: [1 2 3 4 5]
```

每一个数组都有与之对应的索引，所以在Series中，可以通过索引的方式选取或者修改Series的数值。**但是index对象是不可以修改的，这样才可以保证index对象在多个数据结构中共享**

根据索引修改数值

```
print('s2中 第二 对应的数值:  {}'.format(s2['第二']))
s2['第二'] = 10
print('s2中 第二 对应的数值:  {}'.format(s2['第二']))
```

```
s2中 第二 对应的数值:  2
s2中 第二 对应的数值:  10
```

除了单个数值索引之外，Series还可以索引多个数值

```
print('s2中 第二第四第五 对应的数值: {}'.format(s2[['第二','第四','第五']]))
```

```
s2中 第二第四第五 对应的数值: 第二    10
第四     4
第五     5
dtype: int64
```

这里的索引值需要加上两层中括号

如果对于连续索引，可以使用冒号进行设置：

```
print('s2中 第二到第五 对应的数值: {}'.format(s2['第二':'第五']))
```

```
s2中 第二到第五 对应的数值: 第二    10
第三     3
第四     4
第五     5
```

```
dtype: int64
```

注意：这里的切片与Python中的切片是不一样的，Series的切片末端元素是包含在内的，所以末端元素仍然可以被输出

（3）字典类型创建Series 前面都是使用列表数据类型创建Series,也可以使用字典数据类型创建Series,

```
s3_dic = {'First':1,'Second':2,'Third':3,'Fourth':4,'Fifth':5}
s3 = pd.Series(s3_dic)
print('s4: {}'.format(s3))
```

```
s4: First     1
Second    2
Third     3
Fourth    4
Fifth     5
dtype: int64
```

可以看到，直接使用字典数据类型数据创建Series,字典中key对应Series的索引，字典的value对应Series的数值。Series数组的排列按照索引首字符顺次进行排序，如果希望按照指定顺序进行排序，可在Series创建时传入一个index列表，就像使用列表创建Series一样

```
s4_dic = {'First':1,'Second':2,'Third':3,'Fourth':4,'Fifth':5}
s4 = pd.Series(s4_dic,index = ['First','Second','Third','Fourth','Fifth'])
print('s4:{}'.format(s4))
```

```
s4:First     1
Second    2
Third     3
Fourth    4
Fifth     5
dtype: int64
```

可用于字典中的某些函数，比如in not in 可以用于Series数组的索引中

查看某些元素是否在Series数组中，

```
print('s4 中含有 sixth:{}'.format('sixth' in s4))
print('s4中不含有sixth:{}'.format('sixth' not in s4))
```

```
s4 中含有 sixth:False
s4中不含有sixth:True
```

这里的in not in只是用来判断索引值存不存在

如果传入的index参数中含有原字典中不含有的索引标签，那么索引参数与数据字典value值无法匹配成功。未匹配成功的index对应的数值位置就记录为空，用NAN来表示，代表缺失值，可以用is null 和 not null函数判断是否存在缺失值

查看是否存在缺失值

```
s4_dic = {'First':1,'Second':2,'Third':3,'Fourth':4,'Fifth':5}
s4 = pd.Series(s4_dic,index = ['First','Second','Third','Fourth','Tenth'])
print('s4:{}'.format(s4))
```

```
s4:First     1.0
Second    2.0
Third     3.0
Fourth    4.0
Tenth     NaN
dtype: float64
```

Tenth在原来的字典中不存在相应的键值对，所以生成Series时，索引Tenth对应的值就是空的。

```
print('数据缺失: {}'.format(s4.isnull()))
print('数据不缺失:{}'.format(s4.notnull()))
```

```
数据缺失: First     False
Second    False
Third     False
Fourth    False
Tenth      True
dtype: bool
数据不缺失:First      True
Second     True
Third      True
```

```
Fourth     True
Tenth     False
dtype: bool
```

（4）Series的算术运算 不同的Series数组间可做算术运算，在算数运算中，不同的索引对应的数据会自动的对齐。

Series数组运算数据自动对齐：

```
print('s3 + s4: {}'.format(s3 + s4))
```

```
s3 + s4: Fifth     NaN
First      2.0
Fourth     8.0
Second     4.0
Tenth      NaN
Third      6.0
dtype: float64
```

可以看到相应索引处的数值实现了加法运算，s4中Tenth和S3中的Fifth分别做NAN缺失处理。

2.2 DataFrame数据结构

DataFrame是Pandas中的另外一种数据结构，与Series数组结构不同的是，DataFrame是二维表格型结构，既含有行索引，又包含列索引，每一列的元素可能是不同类型的数据，例如字符串、整形数据、布尔类型数据。

（1）DataFrame的创建 DatAFrame的创建与Series类似，可以直接使用函数pd.DataFrame传入一个列表或者字典。

创建DataFrame的代码：

```
df_dic = {'color':['red','yellow','blue','purple','pink'],'size':
['medium','small','big','medium','small'],'taste':
['sweet','sour','salty','sweet','spicy']}
df = pd.DataFrame(df_dic)
print(df)
```

```
    color    size  taste
0     red  medium  sweet
1  yellow   small   sour
2    blue     big  salty
3  purple  medium  sweet
```

```
4    pink   small  spicy
```

解析：每一组数据都自动的添加了索引，序列按照列名称首字母进行排序，如果希望设置排序，可以在pd.DataFrame()函数中传入columns参数。这个就类似于Series中的index列表

指定DataFrame中的columns：

```
df1 = pd.DataFrame(df_dic,columns = ['taste','color','size'])
print(df1)
```

```
   taste   color    size
0  sweet     red  medium
1   sour  yellow   small
2  salty    blue     big
3  sweet  purple  medium
4  spicy    pink   small
```

如果传入的columns中含有与源字典数据key值不匹配的列名称时，该列会被记作NAN列。

```
df1 = pd.DataFrame(df_dic,columns = ['taste','color','size','category'])
print(df1)
```

```
   taste   color    size category
0  sweet     red  medium      NaN
1   sour  yellow   small      NaN
2  salty    blue     big      NaN
3  sweet  purple  medium      NaN
4  spicy    pink   small      NaN
```

DataFrame的表头可以设置列索引名称的标题和行索引名称的标题，需要使用name函数进行设置。

```
df1.index.name = 'sample'
df1.columns.name = 'feature'
print(df1)
```

```

```

```
feature  taste   color    size
sample
0        sweet     red  medium
1         sour  yellow   small
2        salty    blue     big
3        sweet  purple  medium
4        spicy    pink   small
```

这里为列名称设置了feature,为索引名称设置了sample

使用values函数可以获得DataFrame中的所有数据，以二维数组的形式返回（数组与列表的区别是逗号）

```
print('df1的values值为：  {}'.format(df1.values))
```

```
df1的values值为：  [['sweet' 'red' 'medium']
 ['sour' 'yellow' 'small']
 ['salty' 'blue' 'big']
 ['sweet' 'purple' 'medium']
 ['spicy' 'pink' 'small']]
```

注意：数组元素之间都没有逗号

（2）DataFrame的索引

获取DataFrame中的列，可以采用以下两种方式：

```
print('df1中的color列：  {}'.format(df1['color']))
print('df1中的color列：  {}'.format(df1.color))
```

```
df1中的color列：  sample
0       red
1    yellow
2      blue
3    purple
4      pink
Name: color, dtype: object
df1中的color列：  sample
0       red
1    yellow
2      blue
3    purple
4      pink
Name: color, dtype: object
```

可以看到打印的结果都是一样的。

对于行方向上的索引，如果希望获取某一行，可以使用行索引字段ix。

```
print(df1.ix[3])
```

```
feature
taste     sweet
color    purple
size     medium
Name: 3, dtype: object
```

通过DatAFrame数据的索引，可对特定的数组进行修改。

```
import numpy as np
df1['category'] = np.arange(5)
print(df1)
```

```
feature  taste   color    size  category
sample
0        sweet     red  medium         0
1         sour  yellow   small         1
2        salty    blue     big         2
3        sweet  purple  medium         3
4        spicy    pink   small         4
```

可以看到使用Numpy生成一维数组，然后直接填补DataFrame的一列

但是如果只是想填补其中的部分数值，可精确匹配DataFrame中缺失值的索引，然后填补缺失值

```
import numpy as np
df1['category'] = pd.Series([2,3,4],index = [0,2,4])
print(df1)
```

```
feature  taste   color    size  category
sample
0        sweet     red  medium       2.0
1         sour  yellow   small       NaN
2        salty    blue     big       3.0
```

```
3        sweet  purple  medium       NaN
4        spicy    pink   small       4.0
```

可以看到，如果为不存在的列赋值将会创建一个新的列。

```
df1['country'] = pd.Series(['China','UK','USA','Australia','Japan'])
print(df1)
```

```
feature  taste   color    size  category    country
sample
0        sweet     red  medium       2.0      China
1         sour  yellow   small       NaN         UK
2        salty    blue     big       3.0        USA
3        sweet  purple  medium       NaN  Australia
4        spicy    pink   small       4.0      Japan
```

上面的代码，使用Series创建一个数组，并且指定了索引，

DataFrame中可以使用布尔型数组选取行：

```
print(df1[df1['category'] < 3])
```

```
feature  taste color    size  category country
sample
0        sweet   red  medium       2.0   China
```

这里选取了category小于等于3的样本数据，用到了列索引，所以也可以写成df1.category

二、数学与统计计算

Pandas是一个高性能的数据计算库，其中包含一些高效处理数学以及统计运算的函数

Pandas提供了对Series和DataFrame进行汇总统计的函数，比如求和，求平均数、求分位数。

DataFrame数学统计函数：

首先创建一个DataFrame,这里使用二维列表进行创建，不仅指定了index,而且指定了columns

```
df5 = pd.DataFrame([[3,2,3,1],[2,5,3,6],[3,4,5,2],[9,5,3,1]],index =
['a','b','c','d'],columns = ['one','two','three','four'])
```

```
print(df5)
```

```
   one  two  three  four
a    3    2      3     1
b    2    5      3     6
c    3    4      5     2
d    9    5      3     1
```

使用DataFrame中的sum函数将会返回一个按列或者按行求和的的Series。

```
print('按列求和:  {}'.format(df5.sum()))
print('按行求和:  {}'.format(df5.sum(axis = 1)))
```

```
按列求和:  one      17
two      16
three    14
four     10
dtype: int64
按行求和:  a     9
b    16
c    14
d    18
dtype: int64
```

设定axis = 1将进行行求和。不设定默认列求和

consum函数用于计算累计求和值，按照指定顺序依次求和。

```
print('从上到下累计求和:  {}'.format(df5.cumsum()))
print('从左往右累计求和:  {}'.format(df5.cumsum(axis = 1)))
```

```
从上到下累计求和:     one  two  three  four
a    3    2      3     1
b    5    7      6     7
c    8   11     11     9
d   17   16     14    10
从左往右累计求和:     one  two  three  four
a    3    5      8     9
b    2    7     10    16
c    3    7     12    14
```

```
d    9   14     17    18
```

Pandas还定义了其他DataFrame的统计指标，下面就列出相关数据统计函数。

统计函数	解释
mean	均值
median	中位数
count	非缺失值数量
min、max	最大最小值
describe	汇总统计
var	方差
std	标准差
skew	偏度
kurt	峰度
diff	一阶差分
cumin、cumax	累计最大值、累计最小值
cumsum、cumprod	累计和、累计积
cov、corr	协方差、相关系数

三、DataFrame的文件操作

Pandas提供了多种读取文件函数和写入文件函数，可将原始数据文件转换成DataFrame类型的数据结构

3.1 读取文件

Pandas常用的读取数据文件函数如表所示：

读取数据文件函数	解释
pd.read_csv(filename)	从csv文件导入数据，默认分隔符为“,"
pd.read_table(filename)	从文本文件导入数据，默认分隔符为制表符
pd.read_excel(filename)	从Excel文件导入数据
pd.read_sql(query,connection_object)	从SQL表/库中导入数据
pd.read_json(json_string)	从json文件导入数据
pd.read_html(url)	解析url、字符串或者HTML文件，提取数据表格
pd.DataFrame(dict)	从字典对象中读入数据

Panads读取csv文件：

```
pd.read_csv('df.csv',encoding = 'utf-8')
```

第一个参数时原数据文件的存储路径，这里的数据文件存储在当前目录下，encoding参数用于设置编码方式，这里设置为utf-8

3.2 写入文件

Python 常用的写入文件函数如下表：

读取数据文件函数	解释
pd.to_csv(filename)	导入数据至csv文件
pd.to_excel(filename)	导入数据至excel文件
pd.to_sql(table_name,connection_object)	导入数据至SQL表
pd.to_json(json_string)	导出数据为json格式
pd.to_html(url)	导出数据为html文件
pd.to_clipboard(filename)	导出数据到剪切板

Pandas写入csv文件：

```
df.to_csv('df.csv',seq = ',',header = True,index = True,encoding = 'utf-8')
```

第一个参数为写入文件路径，这里表示写入当前目录，seq参数用于设置写入文件的分隔符，header参数表示写入文件是否写入标题行，默认值为True,index参数表示是否写入行索引，默认值为True。

四、数据处理

在做数据分析时，读取的数据有时候不符合是数据分析的要求，可能会存在一些缺失值、重复值，Pandas提供了对Series数组和DataFrame进行数据预处理（数据清洗的方法）

4.1 缺失值处理

（1）不存在型空值，也就是无法获取的值 （2）存在性空值，样本的该特征时存在的，但是暂时无法获取数据，之后该信息一旦被确定，就可以补充数据，使信息趋于完全。 （3）占位型空值，无法确定是存在型空值还是不存在型空值，随着时间的推移来确定

1. 查找缺失值 Pandas中可使用isnull函数来判断是否存在缺失值。DataFrame中的缺失值一般记作：numpy.nan,表示数据空缺。

查找DataFrame中的缺失值

```

```

```
import pandas as pd
import numpy as np
df6 = pd.DataFrame([[3,np.nan,3,1],[2,5,np.nan,6],[3,4,5,np.nan],[5,3,1,3]],index
= ['a','b','c','d'],columns = ['one','two','three','four'])
print(df6.isnull())
```

```
     one    two  three   four
a  False   True  False  False
b  False  False   True  False
c  False  False  False   True
d  False  False  False  False
```

使用DataFrame中的isnull函数，会逐个遍历数组中的每一个元素，每一个索引位置都返回一个布尔值表示其是否为缺失值，缺失值np.nan的位置返回True, 非缺失值的位置False，构成一个由布尔值组成的DataFrame类型数据。

通过此布尔型返回值，结合any函数可以对原DataFrame进行切片，提取所有包含缺失值的数据。

```
# 输出含有缺失值的行  所有的行
print(df6[df6.isnull().any(axis = 1)])
```

```
   one  two  three  four
a    3  NaN    3.0   1.0
b    2  5.0    NaN   6.0
c    3  4.0    5.0   NaN
```

2. 过滤缺失值

dropna函数用于过滤缺失值，可返回不含有缺失值的数据和索引，对于Series数组使用dropna函数进行过滤的实例如下所示：

过滤Series中的缺失值：

```
# 创建一个Series数组
arr = pd.Series([1,2,3,np.nan,5,6])
print(arr)
print(arr.dropna())
```

```
0    1.0
1    2.0
```

```
2    3.0
3    NaN
4    5.0
5    6.0
dtype: float64
0    1.0
1    2.0
2    3.0
4    5.0
5    6.0
dtype: float64
```

使用dropna函数之后，arr中的缺失数据被过滤，但是返回的只是一个副本

可以接着输出：`python print(arr)`

```
0    1.0
1    2.0
2    3.0
3    NaN
4    5.0
5    6.0
dtype: float64
```

原Series数组的缺失数据仍然存在，所以dropna返回的是一个执行删除操作的新的数组，删除操作不改变原来的数组，如果希望改变原来的数组，可以执行如下操作：

```
arr = arr.dropna()
print(arr)
```

将删除之后的数组再次赋值给原数组

对于DataFrame的过滤方法，dropna函数的使用方法与过滤Series数组类似

```
print(df6.dropna())
```

```
   one  two  three  four
d    5  3.0    1.0   3.0
```

dropna函数传入how = 'all'可以删除全为缺失值NAN的行或者列

```
df6['fifth'] = np.NAN
print(df6)
print(df6.dropna(how = 'all',axis = 1,inplace = True))
```

```
   one  two  three  four  fifth
a    3  NaN    3.0   1.0    NaN
b    2  5.0    NaN   6.0    NaN
c    3  4.0    5.0   NaN    NaN
d    5  3.0    1.0   3.0    NaN
```

3. 填充缺失值 fillna函数是处理缺失值最常用的方法，调用fillna函数，传入替换之后的数值，即可完成缺失值的替换。

```
df6['fifth'] = np.NAN
print(df6)
print(df6.fillna(0))
```

```
   one  two  three  four  fifth
a    3  NaN    3.0   1.0    NaN
b    2  5.0    NaN   6.0    NaN
c    3  4.0    5.0   NaN    NaN
d    5  3.0    1.0   3.0    NaN
   one  two  three  four  fifth
a    3  0.0    3.0   1.0    0.0
b    2  5.0    0.0   6.0    0.0
c    3  4.0    5.0   0.0    0.0
d    5  3.0    1.0   3.0    0.0
```

解析：df6中的缺失值全部被替换成为0，**但是数据分析常用哦过的填充数据是数据当前列的中位数或者均值，分别使用Pandas中的median函数和mean函数**，如图所示。

```
print(df6)
print(df6.fillna(df6.median()))
```

```
   one  two  three  four
a    3  NaN    3.0   1.0
b    2  5.0    NaN   6.0
c    3  4.0    5.0   NaN
   one  two  three  four
```

```
a    3  NaN    3.0   1.0
b    2  5.0    NaN   6.0
c    3  4.0    5.0   NaN
d    5  3.0    1.0   3.0
   one  two  three  four
a    3  4.0    3.0   1.0
b    2  5.0    3.0   6.0
c    3  4.0    5.0   3.0
d    5  3.0    1.0   3.0
```

Pandas还提供了向上向下填充缺失值的函数，分别为ffill函数和bfill函数。向上填充法使用缺失值位置的前一个数据代替缺失值，向下填充法使用缺失值的后一个数据代替缺失值。

```
print(df6)
print(df6.fillna(df6.ffill()))
```

```
   one  two  three  four
a    3  NaN    3.0   1.0
b    2  5.0    NaN   6.0
c    3  4.0    5.0   NaN
d    5  3.0    1.0   3.0
   one  two  three  four
a    3  NaN    3.0   1.0
b    2  5.0    3.0   6.0
c    3  4.0    5.0   6.0
d    5  3.0    1.0   3.0
```

```
print(df6)
print(df6.fillna(df6.bfill()))
```

```
   one  two  three  four
a    3  NaN    3.0   1.0
b    2  5.0    NaN   6.0
c    3  4.0    5.0   NaN
d    5  3.0    1.0   3.0
   one  two  three  four
a    3  5.0    3.0   1.0
b    2  5.0    5.0   6.0
c    3  4.0    5.0   3.0
d    5  3.0    1.0   3.0
```

4.2 重复值处理

DataFrame中可能会存在重复的行或者列，或者几行中存在重复的几列，数据的重复或者冗余会影响数据分析的准确性，去除重复值是数据清洗过程的一个重要的环节，duplicated函数是一个可以用来查看是否存在重复值。

1. 查看DataFrame中的重复值

查看DataFrame中的重复值：

```
df7 = pd.DataFrame([[3,5,3,1],[2,5,5,6],[3,4,5,3],[5,3,1,3],[3,4,5,3],
[3,4,6,8]],index = ['a','b','c','d','e','f'],columns =
['one','two','three','four'])

print(df7[df7.duplicated()])
print(df7[df7.duplicated(subset = ['one','two'])])
```

```
   one  two  three  four
e    3    4      5     3
   one  two  three  four
e    3    4      5     3
f    3    4      6     8
```

解析：结合使用布尔型切片查看重复行，duplicated函数中subset参数默认值为None,表示考虑DataFrame中的所有列。如果subset如本例所示指定为某几列，则会针对这几列进行重复值查询。

duplicated函数默认只保留第一次出现重复的行，subset参数用于识别重复的列标签或者列标签序列，默认为所有的列标签，可以根据列筛选重复的行。

2. 去处DataFrame中重复值 使用drop_duplicates函数即可对数据进行去重

```
print(df7.drop_duplicates(subset = ['one','two'],keep = 'first'))
```

```
   one  two  three  four
a    3    5      3     1
b    2    5      5     6
c    3    4      5     3
d    5    3      1     3
```

这里设置参数subset，只对前两列进行检查，之后有设置了参数keep,first表示保留第一次出现的重复值，keep还有另外两个参数值，分别为last、false。

3. 数据记录合并与分组

不同的DataFrame中的数据有时需要放在一起分析，Pandas常用的数据合并方法有append、concat、merge等。

1. 使用append函数连接两个DataFrame（列索引必须相同），

```
df8 = pd.DataFrame([[3,3,2,4],[5,4,3,3]],index = ['g','h'],columns =
['one','two','three','four'])
print(df8.append(df7))
```

```
   one  two  three  four
g    3    3      2     4
h    5    4      3     3
a    3    5      3     1
b    2    5      5     6
c    3    4      5     3
d    5    3      1     3
e    3    4      5     3
f    3    4      6     8
```

可以看到合并了两行，列数还是四列。

2. 使用concat函数合并数据记录 concat函数也可以对DataFrame连接，可以指定两个DataFrame按照某个轴进行连接，也可以指定二者连接的方式，axis参数可以指定连接的轴向，axis默认值为0，表示列对齐，两表上下合并，与append()结果相同，axis = 1时，表示行对齐，两表左右合并

```
# 默认上下连接
print(pd.concat([df7,df8]))
```

```
   one  two  three  four
a    3    5      3     1
b    2    5      5     6
c    3    4      5     3
d    5    3      1     3
e    3    4      5     3
f    3    4      6     8
g    3    3      2     4
h    5    4      3     3
```

```
# 左右连接
print(pd.concat([df8,df7],axis = 1))
```

```
   one  two  three  four  one  two  three  four
a  NaN  NaN    NaN   NaN  3.0  5.0    3.0   1.0
b  NaN  NaN    NaN   NaN  2.0  5.0    5.0   6.0
c  NaN  NaN    NaN   NaN  3.0  4.0    5.0   3.0
d  NaN  NaN    NaN   NaN  5.0  3.0    1.0   3.0
e  NaN  NaN    NaN   NaN  3.0  4.0    5.0   3.0
f  NaN  NaN    NaN   NaN  3.0  4.0    6.0   8.0
g  3.0  3.0    2.0   4.0  NaN  NaN    NaN   NaN
h  5.0  4.0    3.0   3.0  NaN  NaN    NaN   NaN
```

解析： concat函数上下连接时相同的列索引的数据进行合并，左右连接时相同的行索引的数据合并。

join参数用来设置连接方式，join的默认值为'outer'，表示两个数据集若存在不重合索引，则取并集，未匹配的位置处记录为缺失值NAN,join = 'inner' 表示对两数据集取交集，只返回都匹配成功的数据。

3. 使用merge函数合并数据记录 merge函数可以根据两个DataFrame共有的某个字段进行数据合并，类似于关系数据库的连接，通过一个或者多个键将这两个数据集的行连接在一起，合并之后的DataFrame行数没有增加，列数为两个DataFrame的总列数减去连接键的数量。

```
df_dic11 = {'color':['red','yellow','blue','purple','pink'],'size':
['medium','small','big','medium','small'],'taste':
['sweet','sour','salty','sweet','spicy'],'category':[2,3,4,5,6]}
df9 = pd.DataFrame(df_dic11,columns = ['taste','color','size','category'])
print(df9)
df_dic12 = {'country':['China','UK','USA','Australia','Japan'],'quality':
['good','normal','excellent','good','bad'],'category':[2,3,4,5,6]}
df10 = pd.DataFrame(df_dic12,columns = ['country','quality','category'])
print(df10)
print(pd.merge(df9,df10,left_on = 'category',right_on = 'category',how = 'left'))
```

```
   taste   color    size  category
0  sweet     red  medium         2
1   sour  yellow   small         3
2  salty    blue     big         4
3  sweet  purple  medium         5
4  spicy    pink   small         6
     country    quality  category
0      China       good         2
1         UK     normal         3
2        USA  excellent         4
```

```
3  Australia       good         5
4      Japan        bad         6
   taste   color    size  category    country    quality
0  sweet     red  medium         2      China       good
1   sour  yellow   small         3         UK     normal
2  salty    blue     big         4        USA  excellent
3  sweet  purple  medium         5  Australia       good
4  spicy    pink   small         6      Japan        bad
```

left_on参数表示主键在左侧DatFrame中的列名称，right_on表示主键在右侧的列名称，两个的名称可以不相同，

how表示DataFrame的连接方式，默认值为'inner',表示根据主键对两表匹配时，若未完全匹配，则保留匹配成功的部分，也就是两者的交集，how参数还可以设置为right、outer，参数设置为outer时，两个DataFrame中未匹配成功的部分全部保留，也就是两者的并集，how = left对于未匹配的部分保留左边DatFrame中含有，但是右边DataFrame中不含有的部分。

结束，后续知识，遇到之后在进行补充改进