**数据分析——Pandas常用操作合集（代码部分）**

**作者：MCR 2021/01/03**



**1.2 导入Pandas**

1. **import** pandas as pd
2. #InteractiveShell可以让jupyter notebook一个cell显示多条结果，推荐使用
3. **from** IPython.core.interactiveshell **import** InteractiveShell
4. InteractiveShell.ast\_node\_interactivity = ‘all'

**1.3 数据的导入和保存**

1. df = pd.read\_csv(“file path/filename.csv”)
2. df.to\_csv(“file path/filename.csv")

**1.4 数据查看**

1. df.head(n)      #查看DataFrame对象的前n行
2. df.tail(n)      #查看DataFrame对象的最后n行
3. df.shape        #查看行数和列数
4. df.info()      #查看索引、数据类型和内存信息
5. df.columns      #查看列
6. df.describe()   #查看数据的统计分析，最大最小值、平均值、四分位数等

**1.5 数据选取**

数据的选取一般有切片法，loc定位法和iloc定位法。.iloc()与.loc()的区别在于，loc是根据标签名选取，iloc是根据位置选取， 官方解释：

**loc for label based indexing**

**iloc for positional indexing**

数据选取按功能可以分为行选取，列选取和单元格选取，具体用法如下：

1. #选取行
2. df[A:B]   #选取索引为A~B的行
3. df.loc[1,:]   #选取索引为1的行,返回值为Series
4. df.loc[1:1,:]  #选取索引为1的行,返回值为dataframe
5. df.loc[[1,3,5]]  #选取索引为1,3,5的行
7. #选取列
8. df.A , df["A"]  #选取列名为A的一列
9. df[["A","B", ……]]  #选取列名为A，B......的多列，注意是两个括号！！
10. df.loc[:,["A","B"]] #选取列名为A，B......的多列，与上面的功能相同
11. df.iloc[:,0:2]  #选取前两列数据
13. #选取单元格
14. df.loc[0,"A"]  #选取索引为0的行，列名为A的单元格
15. df.iloc[n,m]  #选取第n行,第m列的单元格

总结一下：

（1）loc[ ]的用法，df.loc[A:B, ["C","D"]]表示：选取行标签为A~B，列名为C和D的数据。（2）df.loc[ ]还可以用来赋值, 在给表格中的数据赋值时，不要用df[“column”][n]=value, 而要用df.loc[n,”A”]=value, 不然会出现warning的情况。

（3）选取数据时，返回值存在以下3种情况：

如果返回值包括单行多列或多行单列时，返回值为Series对象；

如果返回值包括多行多列时，返回值为DataFrame对象；

如果返回值仅为一个单元格（单行单列）时，返回值为基本数据类型。

**1.6 其他常用操作**

1. #删除行和列
2. df=df.drop(columns="A")   #删除列名为A的列
3. df=df.drop(columns=["A","B"]) #删除列名为A，B.....的列
4. df=df.drop(labels=0)   # axis默认等于0，即按行删除，这里表示按行删除第0行
6. #更改列名A变为B, C变为D，注意是大括号！
7. df=df.rename(columns={"A":"B","C":"D"})
9. #数据类型转换
10. df.column = df.column.astype("int")
12. #查看DataFrame中是否有NaN并填写为0
13. df.isnull().sum()
14. df=df.fillna(0)

**2.2数据的条件查询**

数据的条件查询主要有以下几种方法：1.使用.loc方法查询；2.使用.query（）函数查询；3.使用lambda表达式查询等方法。其中.loc方法和.query()函数查询最为常用，而.query()可以以简洁优雅的方式进行条件查询，非常实用。三种条件查询的其使用方法如下：

1. #loc方法查询
2. df[df['A'] > X]   #选取某一列中大于X的值
3. df.loc[df['A']>X]   #同上
4. df.loc[（df['A']<X2）&（df['A']>X1）] #选取某一列中大于X1小于X2的值
6. #query()方法查询
7. df.query('A == 1')  #查询dataframe中列A中等于1的行，返回值为dataframe
8. df.query('A == “day”')  #查询dataframe中列A中等于“day”的行
9. df.query('A > X')  #查询dataframe中列A中大于X的行
10. df.query('A > X & B == Y') #支持逻辑符，查询列A中大于X并且列B中等于Y的行
11. df.query('A == @variable\_name') #可以在环境中引用变量，在变量前面加上@字符
13. #lambda表达式查询
14. df.loc[**lambda** df : (df.['A']>=X)&(df.['B']<=Y),:] #查询列A中大于X并且列B中等于Y的行

**2.3列表的排序(sort\_values,reindex, reset\_index, set\_index)**

列表的排序主要有**sort\_values (), reindex(), reset\_index(), set\_index()**四种方法，**sort\_values ()**用于对列的值进行排序；**reindex()**用于改变行索引和列索引；**reset\_index()**用于从0开始对行索引进行重新排序；**set\_index()**用于设置索引，可以设置单索引和复合索引。具体使用方法如下：

1. lists.sort\_values(by="A",inplace=True,ascending=True)：
2. #根据A列的值进行升序排列，添加参数ascending=False可改为降序排列
3. df2=df1.reindex(['a','b','c']) #覆盖原来的索引的0,1,2位置为a，b，c
4. df2=df1.set\_index('c') #在第0行插入索引为c的行，之前的索引向下顺移
5. df2=df1.reset\_index()  #覆盖原来的索引，从0开始重新排序

**2.4 列表的拼接（concat，merge，join和append）**

Pandas提供了concat，merge，join和append四种方法用于dataframe的拼接，其区别如下：

|  |  |  |
| --- | --- | --- |
| **函数** | **适用场景** | **调用方法** |
| **.concat()** | 列表的内联或外联拼接，默认行拼接，取并集 | df = pd.concat( [df1,df2], axis=1 ) |
| **.merge()** | 支持左联、右联、内联和外联等全部四种SQL连接操作类型，默认列拼接，取交集 | result=pd.merge(df1, df2,how=‘left’) |
| **.join()** | 默认列方向拼接操作，支持左联、右联、内联和外联四种操作类型 | df1.join(df2) |
| **.append()** | concat的简略形式，行方向拼接操作 | df1.append(df2) |

四种列表拼接函数的使用方法如下：

1. #concat
2. df = pd.concat([df1, df2], axis=0, join='outer') #将df1和df2取并集后进行行拼接。默认axis=0, join='outer'
3. #axis=1 -> 变为列拼接
4. #join='inner' -> 变为取交集
6. #merge
7. df = pd.merge(df1,df2,how=‘inner',on='A') #将df1和df2根据列A的值是否相同取并集（df1和df2都需要有列A）
8. #how -> 连接方式，有inner（取交集）、left（左联）、right（右联）、outer（取并集），默认为inner。
9. #on -> 用于连接的列索引名称
10. #left\_on -> 左侧DataFrame中用于连接键的列名，用于左右列名不同但代表的含义相同时
11. #right\_on -> 右侧DataFrame中用于连接键的列名
13. #join
14. #参数意义和merge基本相同，默认左外连接how=’left’
15. df1.join(df2) #无重复列名的两个表df1和df2基于行索引进行列拼接
16. df1.join(df2, lsuffix='\_l', rsuffix='\_r') #df1和df2 基于行索引进行列拼接
17. #若两个列表中有相同名称的列，则需要指定lsuffix和rsuffix参数
19. #append
20. df1.append(df2) #append是concat的简略形式,只不过只能在axis=0上进行合并

**2.5其他常用操作**

1. #根据某一列的值找到对应的行：
2. df\_index = df["A"].str.contains("abc")
3. df[df\_index]
5. #显示所有列
6. pd.set\_option('display.max\_columns', None)
7. #显示所有行
8. pd.set\_option('display.max\_rows', None)