import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

## 一、产生pandas对象

有三种对象

常用方法：

### 常用方法：

对于二维dataframes，data[col]也是一个series对象。

#### 1.size

data[col].size()：统计该列的个数

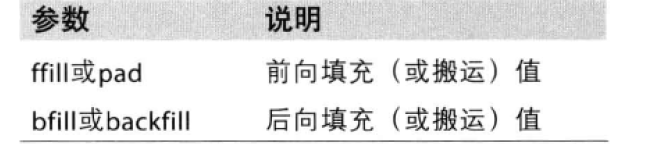
seri.size():统计series元素个数

#### 2.reindex

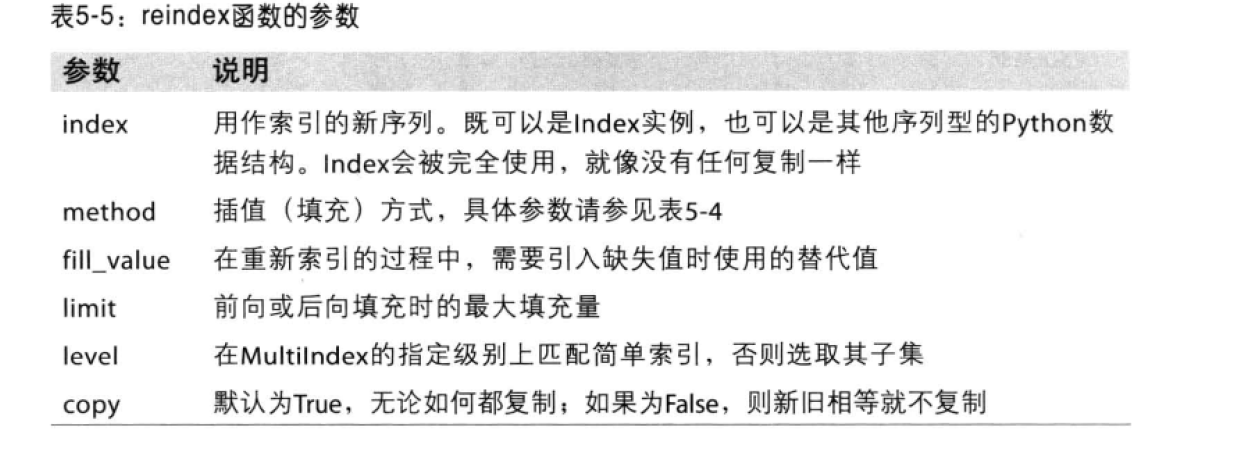
重新索引，对索引进行排序进行排序。

对于原本不存在的索引，默认补nan。或者通过设置fill\_value的值来补充。

有可以通过method参数进行插值



method=’ffill’等等



3.

### 1. Series

一维数组，基于numpy的数组对象得来，但是可以为数据自定义标签索引（index），然后通过索引来访问数组中数据。

#### 创建对象

创建一个 Series 的基本语法如下：

 my\_series=pd.Series(data,index)

上面的 data 参数可以是任意数据对象，比如字典、列表甚至是 NumPy 数组，而index 参数则是对 data 的索引值，类似字典的 key。index可省略，默认为从0开始的数字。

s1=pd.Series([1,3,5,np.nan])

s1

输出：

0 1.0

1 3.0

2 5.0

3 NaN

dtype: float64

默认情况下，`Series` 的下标都是数字（可以使用额外参数指定），类型是统一的。

下面这个例子里，将创建一个 Series 对象，并用字符串对数字列表进行索引：



还可以通过numpy数组、字典等形式创建



也可以通过标量创建：

pd.Series(5,index=[‘a’,’b’,’c’])

则都被填充为5

#### 访问

访问Series也和访问字典/列表是一样的，

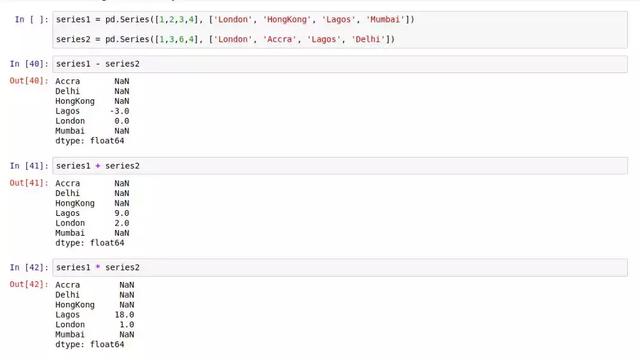
对于有index的，可以使用主键进行访问，也可以按s[0]这种列表访问方式访问。

也可以按照

#### 计算

对 Series 的算术运算都是基于 index 进行的。我们可以用[加减乘除](https://www.baidu.com/s?wd=%E5%8A%A0%E5%87%8F%E4%B9%98%E9%99%A4&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)（+ - \* /）这样的运算符对两个 Series 进行运算，Pandas 将会根据索引 index，对响应的数据进行计算，结果将会以浮点数的形式存储，以避免丢失精度。

如果 Pandas 在两个 Series 里找不到相同的 index，对应的位置就返回一个空值 NaN。



### 2.dataframes

Pandas 的 DataFrame（数据表）是一种 2 维数据结构，数据以表格的形式存储，分成若干行和列。通过 DataFrame，你能很方便地处理数据。常见的操作比如选取、替换行或列的数据，还能重组数据表、修改索引、多重筛选等。

有点像嵌套series的类型。

dataframes的每列的数据格式要相同

#### 创建

##### 利用字典创建

利用字典按列创建，一个键值是一列，值可以是列表、series对象、其他可迭代对象，当个数字，如果是单个数字，则本列都是这个数字。其实最终的值都会被自动转化为series对象。index可以省略。



##### 指定行列创建

eg2：

先创建一个时间序列

dates = pd.date\_range('20130101', periods=6) #从2013-01-01开始，创建连续6天的时间日期

dates:

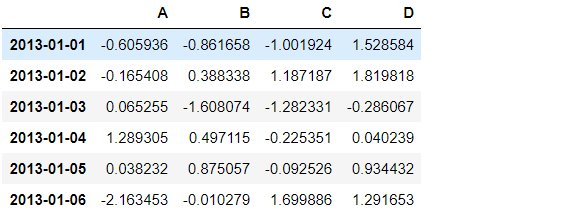
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04', '2013-01-05', '2013-01-06'], dtype='datetime64[ns]', freq='D')

然后创建一个 `DataFrame` 结构：

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

# 行是dates，列是abcd，共6行4列

df



默认情况下，如果不指定 `index` 参数和 `columns`，那么他们的值将用从 `0` 开始的数字替代。

##### 从结构数组中构建

`numpy` 支持结构数组的构造：

首先构建numpy结构数组：

data = np.zeros((2,), dtype=[('A', 'i4'),('B', 'f4'),('C', 'a10')])

data[:] = [(1,2.,'Hello'), (2,3.,"World")]

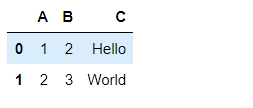
结构数组：

array([(1, 2.0, 'Hello'), (2, 3.0, 'World')],

dtype=[('A', '<i4'), ('B', '<f4'), ('C', 'S10')])

构建dataframes：

pd.DataFrame(data)



#### 插入新列/列赋值

1.插入全新的列：

和字典插入新键值一样：

data[‘col5’]=…….

2.利用现有列生成新列：

data[‘col5’]=data[‘col1’]+data[‘col2’]

3.插入bool列

df['flag'] = df['one'] > 2

插入的列为bool型。

4.如果是赋值单值，则填满该列

df[‘col6’]=1

5.不足补nan

df['one\_trunc'] = df['one'][1:3]

对应的1:3 为赋值的值，其余补nan

像字典一样，直接指定 `F` 列的值为 `s1`，此时以 `df` 已有的 `index` 为标准将二者进行合并，`s1` 中没有的 `index` 项设为 `NaN`，多余的项舍去。

6.指定单元格进行赋值

可以用loc、iloc、at、iat进行指定单元格赋值。

#### 添加行append

添加到最后一行后面

向df中添加新行：

df = pd.DataFrame(np.random.randn(8, 4), columns=['A','B','C','D'])

df

将第三行的值添加到最后：

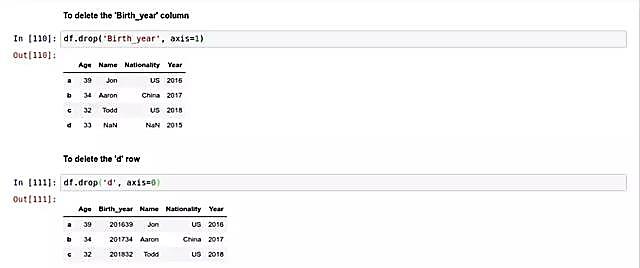
s = df.iloc[3]

df.append(s, ignore\_index=True)

#### 删除行列

##### drop

想要删除某一行或一列，可以用 .drop() 函数。在使用这个函数的时候，你需要先指定具体的删除方向，axis=0 对应的是行 row，而 axis=1 对应的是列 column 。



请务必记住，除非用户明确指定，否则在调用 .drop() 的时候，Pandas 并不会真的永久性地删除这行/列。这主要是为了防止用户误操作丢失数据。

你可以通过调用 df 来确认数据的完整性。如果你确定要永久性删除某一行/列，你需要加上 inplace=True 参数。

##### del

像删除字典键一样删除列

del data[‘col’]

##### pop

data.pop(‘col2’)

删除并返回

#### 数据操作、转置

##### ①获取列/行

data[‘列名1’]、data.列名

要获取一列的数据，还是用中括号 [] 的方式，跟 Series 类似。返回series类型

如果列名不存在，或者该行没有数据，返回nan

获取多列：

data[[‘col1’,’col2’,…]]

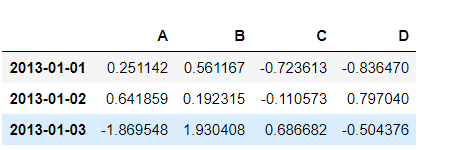
返回dataframes类型

切片获取行：

data[0:3]

也可以对行标进行切片：

df["20130101":"20130103"]



注意：此处不能单独指定获取行，如data[0]或被认为是取列数据。可以使用data[0:1]来代替

##### ②查看列类型

因为我们只获取一列，所以返回的就是一个 Series。可以用 type() 函数确认返回值的类型：

type（data[‘col1’]）

data.dtypes:

返回所有列的类型。返回一个series类型，index为原表列的名称，内容为列的类型。

##### ③查看首尾数据

`head` 和 `tail` 方法可以分别查看最前面几行和最后面几行的数据（默认为 5），向查看几行，往里面填入数字即可：

df.head()

df.tail(3)

##### ④下标（行标）、列标、数据

下标使用 `index` 属性查看：

df.index

列标用columns查看：

df.columns

数据值使用 `values` 查看：

df.values

返回一个二维numpy数组。

##### ⑤iloc、loc、at、iat

.iloc：根据标签的所在位置，从0开始计数，选取行、列

loc：根据DataFrame的具体标签选取行、列

具体选择结果请看示例：

如果只有一个参数，则默认为行。

import pandas as pd

filename='../file/testiloc'

data=pd.read\_excel(filename)

data\_test1=data.iloc[:,:8] #选取位置为[0,8)列的整列数据

data\_test2=data.iloc[0:2,8] #选取位置为8的列的[0,2)行的数据

data\_test3=data.loc[0:2,'工龄'] #选取列名为‘工龄’的[0,2]行的数据

要获取某一行，你需要用 .loc[] 来按索引（标签名）引用这一行，或者用 .iloc[]，按这行在表中的位置（行数）来引用。



同时你可以用 .loc[] 来指定具体的行列范围，并生成一个子数据表，就像在 **NumPy**里做的一样。比如，提取 'c' 行中 'Name’ 列的内容，可以如下操作：



获取一个单元格的值时，可以使用loc、iloc来指定某一行某一列，但是使用at、iat更快。

df.at[‘b’,’name’]

df.iat[0,0]

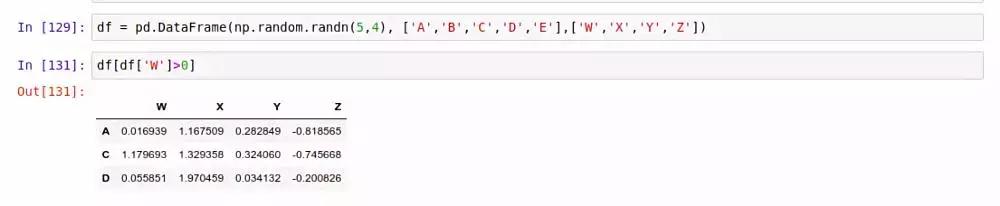
##### ⑥条件筛选

条件筛选的本质是类似numpy中的where，设置条件，返回bool值，利用布尔值进行取值。可以直接在[]中放置bool对象，效果一样。

条件筛选只作用于行，也就是一个bool值对应一行，如果为True，则该行被选取，如果为False，则该行不被选取。

如果想在列上应用，可以使用转置，将列转置为行，使用后，在转置回去。

用中括号 [] 的方式，除了直接指定选中某些列外，还能接收一个条件语句，然后筛选出符合条件的行/列。比如，我们希望在下面这个表格中筛选出 'W'>0 的行：



筛选行的时候，用列做条件，筛选列的时候用行做条件，返回一个dataframes对象，可以进行嵌套查询。

你可以用逻辑运算符 &（与）和 |（或）来链接多个条件语句，以便一次应用多个筛选条件到当前的 DataFrame 上。举个栗子，你可以用下面的方法筛选出同时满足 'W'>0 和'X'>1 的行：



设定满足条件的数值：

此处直接等于-df2即可，会补充相应位置的数值。

df2 = df.copy()

df2[df2 > 0] = -df2

df2

###### if ..then

##### ⑦切片

data[5:7]获取5、6行。或者用iloc也行

##### ⑧转置

df.T

行列转置，名称也跟着转过去。

##### ⑨****删除或填充空值****

dropna()/fillna()

fillna默认是按照列进行处理。如data.fillna(data.mean())是每列填入每列的平均值。

在许多情况下，如果你用 Pandas 来读取大量数据，往往会发现原始数据中会存在不完整的地方。在 DataFrame 中缺少数据的位置， Pandas 会自动填入一个空值，比如 NaN或 Null 。因此，我们可以选择用 .dropna() 来丢弃这些自动填充的值，或是用.fillna() 来自动给这些空值填充数据。

当你使用 .dropna() 方法时，就是告诉 Pandas 删除掉存在一个或多个空值的行（或者列）。删除列用的是 .dropna(axis=1) ，删除行用的是 .dropna(axis=0) 。默认为删除行。只要该行有一个nan或null，删除该行。

也可以指定列或者行进行填充。

data[‘F’].fillna(‘aa’)

按照每列的平均值进行填充：

data.fillna(data.mean())

不会改变原数据，返回改变或的该列。除非设置inplace=True

.dropna() 和 .fillna() 并不会永久性改变你的数据，除非你传入了inplace=True 参数。

##### ⑩检查缺失值的位置

检查缺失数据的位置：

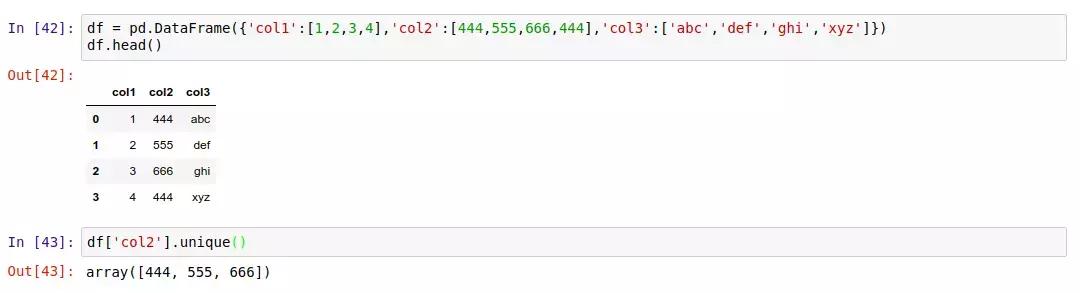
pd.isnull(df1)

返回一个和df1相同大小的bool dataframes，如果该位置缺失，则为True。

##### 查找不重复值

###### 1.unique

比如在下面这个 DataFrame 里，查找 col2 列中所有不重复的值：.



###### 2. nunique

除了列出所有不重复的值，我们还能用 .nunique() 方法，获取所有不重复值的个数：



###### 3.value\_counts()

还可以用 .value\_counts() 同时获得所有值和对应值的计数：



##### 去除重复项

DataFrame.drop\_duplicates(subset=None, keep='first', inplace=False)

参数

这个drop\_duplicate方法是对DataFrame格式的数据，去除特定列下面的重复行。返回DataFrame格式的数据。

subset : 列名或者列名的列表。

用来指定特定的列，默认所有列

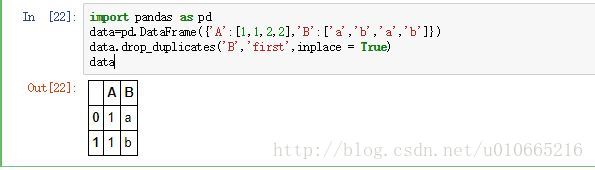
keep : {‘first’, ‘last’, False}, default ‘first’

删除重复项并保留第一次出现的项

inplace : boolean, default False

是直接在原来数据上修改还是保留一个副本

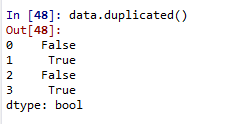
实验：



可以看到，将b列中重复的值去掉了。

##### 删除重复行/列：

pandas判断dataframe是否含有重复行数据用：df.duplicated()



第一次出现的数据为False.重复的数据行就被记录为True。

为True表明前面已经存在该行数据。

参数：

有个参数keep可以设置：

默认keep=’first’ 即第一次出现标为False

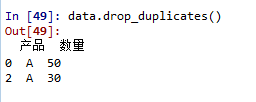
keep=’last’ 最后一次出现标为False

keep=False 所有重复的都标为 True

还有个参数 subset有来指定看哪些列的重复值，为列表或者str

默认 subset=None，即所有列。

去掉重复行数据使用data.drop\_duplicates().



索引会乱，用reset\_index重置索引。

删除重复列，可以先将data转置，然后才去如上操作。

#### 重置/设置索引

##### ①重置索引

如果你觉得当前 DataFrame 的索引有问题，你可以用 .reset\_index() 简单地把整个表的索引都重置掉。这个方法将把目标 DataFrame 的索引保存在一个叫 index 的列中，而把表格的索引变成默认的从零开始的数字，也就是 [0, ..., len(data) - 1] 。比如下面这样：

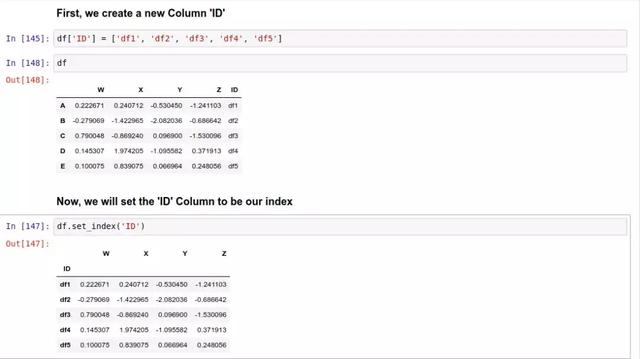


和删除操作差不多，.reset\_index() 并不会永久改变你表格的索引，除非你调用的时候明确传入了 inplace 参数，比如：.reset\_index(inplace=True)

##### ②设置索引

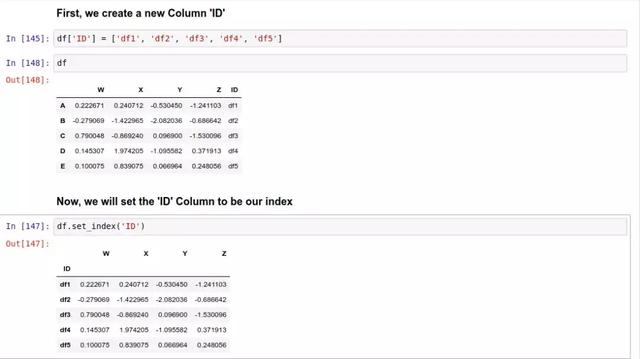
设置 DataFrame 的索引值

类似地，我们还可以用 .set\_index() 方法，将 DataFrame 里的某一列作为索引来用。比如，我们在这个表里新建一个名为 "ID" 的列：



然后将’ID’列设置为索引

df.set\_index(‘ID’)



注意，不像 .reset\_index() 会保留一个备份，然后才用默认的索引值代替原索引，.set\_index() 将会完全覆盖原来的索引值。

多级索引（MultiIndex）以及命名索引的不同等级

#### 修改列名

1.修改所有列名

data.columns=[…..],所有列名都要写入，缺一不可。

2.修改指定列名

使用字典格式来指定。

data.rename(columns={‘A’:’a’,’B’:’b’},inplace=True)

inplace=True，在原数据上直接修改。

#### 统计数据

##### ①groupby（）

设置按某个字段中相同类别的进行分组。类似于mysql中的groupby。

按公司分组，求平均值：



groupby就是将表中的行按照某列进行分割。分割成多个子表。

##### ②count()

用count()方法，可以对dataframes中非nan的数据个数进行统计。默认按列进行统计。

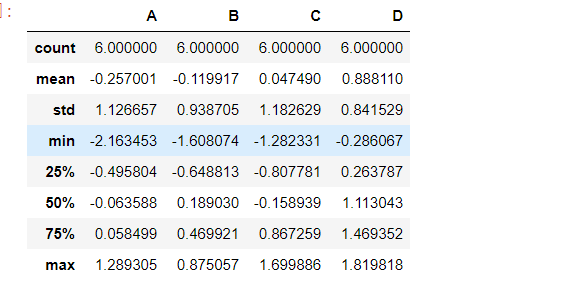
axis=1则按行统计。返回一个series对象，索引为列表或者行标，内容为该行或者该列的非nan值得个数。

还有一个参数**numeric\_only**  布尔值，默认为Flase，如果为True，则只统计数值型。

##### ③describe

查看简单的统计数据：

df.describe()



可以结合groupby来使用：

df.groupby(‘company’).describe()

如果想看某一个数据，可以在后面加上该列名：

df.groupby(‘company’).describe()[‘google’]

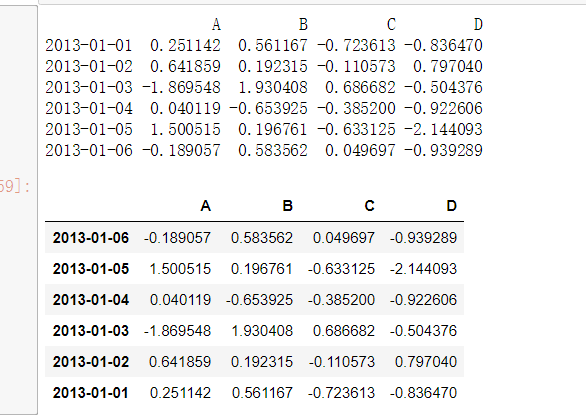
##### ④排序sort\_index/sort\_values

`sort\_index(axis=0, ascending=True)` 方法按照下标大小进行排序，`axis=0` 表示按第 0 维（行标）进行排序。默认为按照从大到小的顺序。类似于excel中的排序。按照某一列的大小对全局表格排序。

ascending=False 为升序排列。

print(df)

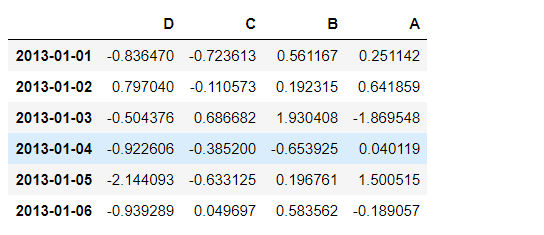
df.sort\_index(ascending=False)



可以看到，行的顺序按照行标的大小进行了重新排列。

df.sort\_index(axis=1, ascending=False) 按照列标进行升序排序。

③



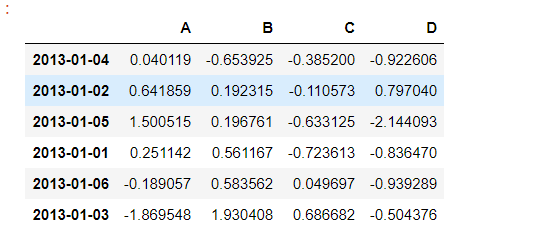
也是从大往小排。

如果想按照某一列或者某一行的顺序进行排序，则添加参数by

`sort\_values(by, axis=0, ascending=True)` 方法按照 `by` 的值的大小进行排序，例如按照 `B` 列的大小：

ascending=False 为升序排列。

df.sort\_values(by="B")



##### ⑤平均值mean

每一列的均值：

df.mean()

每一行的平均值：

df.mean(1)

##### ⑥求和sum

对每列就行分别求和。

##### ⑦idmax,idmin，max，min

data.idmax() 返回一个series，index为列名，内容为该列最大值得行标。

data.max() 返回每列的最大值

#### 数据表的拼接归并

##### 堆叠concat

堆叠基本上就是简单地把多个 DataFrame 堆在一起，拼成一个更大的 DataFrame。当你进行堆叠的时候，请务必注意你数据表的索引和列的延伸方向，堆叠的方向要和它一致。

pd.concat(df1,df2,df3):默认是增加行数，按照纵向（向下）进行堆叠。（axis=0）（一般列索引形相同时使用）

如果想增加列，按照横向（向右）进行堆叠，axis=1。（行索引相同时使用）

如果数据不匹配，则会补nan或者丢弃。

keys=[ ]，指定拼接后的列名。对于将几个series对象拼接成dataframes比较有用。

##### 归并merge

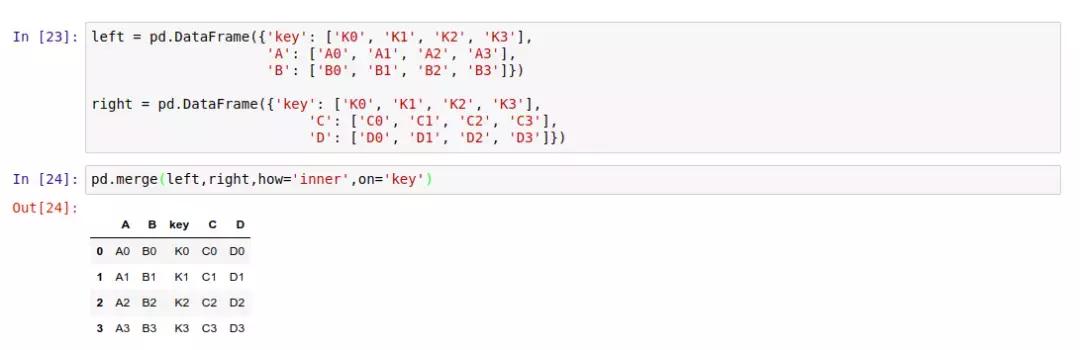
使用 pd.merge() 函数，能将多个 DataFrame 归并在一起，它的合并方式类似合并 SQL 数据表的方式。

归并操作的基本语法是

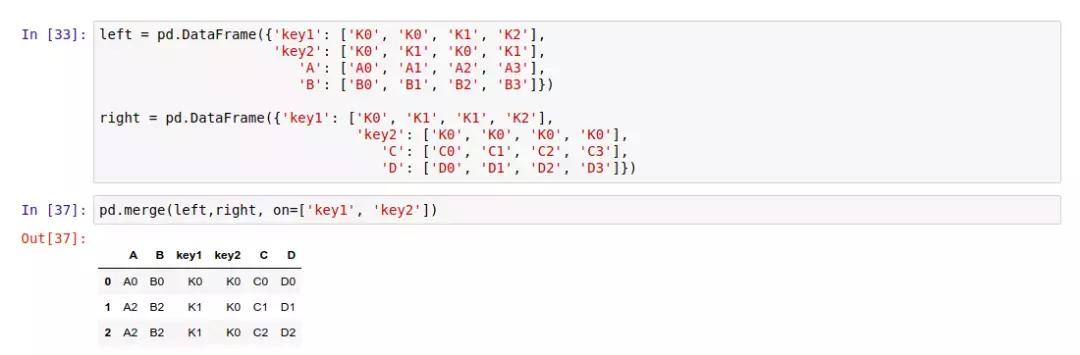
pd.merge(left, right, how='inner', on='Key')

其中 left 参数代表放在左侧的 DataFrame，而 right 参数代表放在右边的 DataFrame；how='inner' 指的是当左右两个 DataFrame 中存在不重合的 Key 时，取结果的方式：inner 代表交集；Outer 代表并集。最后，on='Key' 代表需要合并的键值所在的列，最后整个表格会以该列为准进行归并。

对于两个都含有 key 列的 DataFrame，我们可以这样归并：



on也可以设置多个键，当指定的所有键都相同时，会组合，如果没有匹配到，会被丢弃。



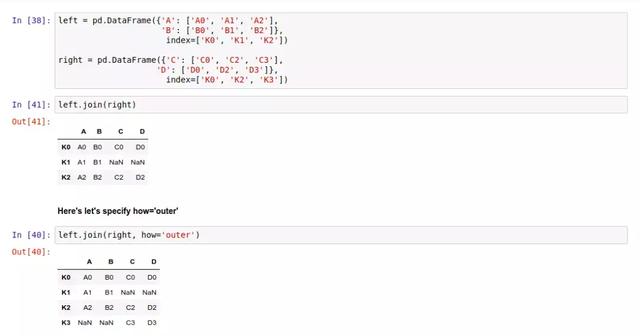
如上：

left中‘k2 k1’未匹配到，就是被丢弃

而left中‘k1 k0’对应right中两个，则会组合成两个。

##### 连接join

如果你要把两个表连在一起，然而它们之间没有太多共同的列，那么你可以试试 .join() 方法。和 .merge() 不同，连接采用索引作为公共的键，而不是某一列。利用的是公共索引。



outer求的是并集，而inner求的是交集

### 数据透视表

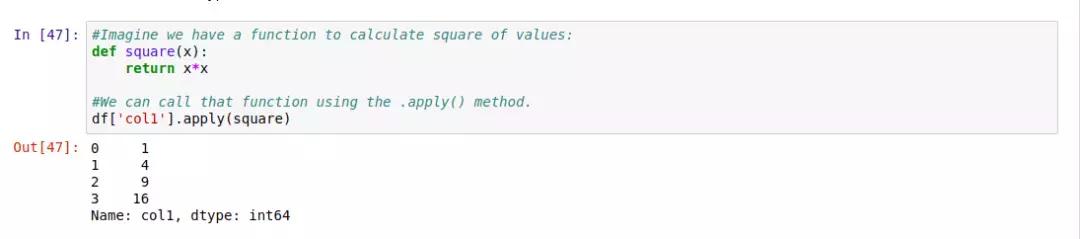
pivot\_table

<https://www.cnblogs.com/onemorepoint/p/8425300.html>

#### apply方法

与 `R` 中的 `apply` 操作类似，接收一个函数，默认是对将函数作用到每一列上：

用 .apply() 方法，可以对 DataFrame 中的数据应用自定义函数，进行数据处理。比如，我们先定义一个 square() 函数，然后对表中的 col1 列应用这个函数：



在上面这个例子中，这个函数被应用到这一列里的每一个元素上。同样，我们也可以调用任意的内置函数。比如对 col3 列取长度 len ：



有的时候，你定义了一个函数，而它其实只会被用到一次。那么，我们可以用 lambda 表达式来代替函数定义，简化代码。比如，我们可以用这样的 lambda 表达式代替上面 In[47] 里的函数定义：



#### 虚拟变量get\_dummies

虚拟变量(dummy variables)

虚拟变量，也叫哑变量和离散特征编码，可用来表示分类变量、非数量因素可能产生的影响。

离散特征的编码分为两种情况：

1、离散特征的取值之间没有大小的意义，比如color：[red,blue],那么就使用one-hot编码

2、离散特征的取值有大小的意义，比如size:[X,XL,XXL],那么就使用数值的映射{X:1,XL:2,XXL:3}

使用pandas可以很方便的对离散型特征进行one-hot编码

[python] [view plain](http://blog.csdn.net/lujiandong1/article/details/52836051) [copy](http://blog.csdn.net/lujiandong1/article/details/52836051)

import pandas as pd

df = pd.DataFrame([

            ['green', 'M', 10.1, 'class1'],

            ['red', 'L', 13.5, 'class2'],

            ['blue', 'XL', 15.3, 'class1']])

df.columns = ['color', 'size', 'prize', 'class label']

size\_mapping = {

           'XL': 3,

           'L': 2,

           'M': 1}

df['size'] = df['size'].map(size\_mapping)

class\_mapping = {label:idx for idx,label in enumerate(set(df['class label']))}

df['class label'] = df['class label'].map(class\_mapping)

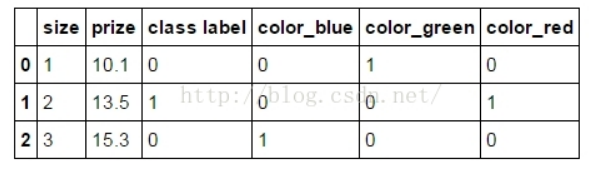
说明：对于有大小意义的离散特征，直接使用映射就可以了，{'XL':3,'L':2,'M':1}



Using the get\_dummies will create a new column for every unique string in a certain column:**使用get\_dummies进行one-hot编码**

[python] [view plain](http://blog.csdn.net/lujiandong1/article/details/52836051) [copy](http://blog.csdn.net/lujiandong1/article/details/52836051)

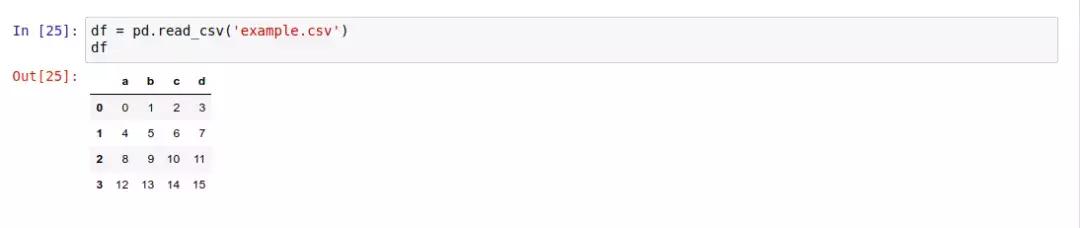
pd.get\_dummies(df)



### 读取存储文件

#### 读取 CSV 文件

简单地说，只要用 pd.read\_csv() 就能将 CSV 文件里的数据转换成 DataFrame 对象：



#### 写入 CSV 文件

将 DataFrame 对象存入 .csv 文件的方法是 .to\_csv()，例如，我们先创建一个 DataFrame 对象：



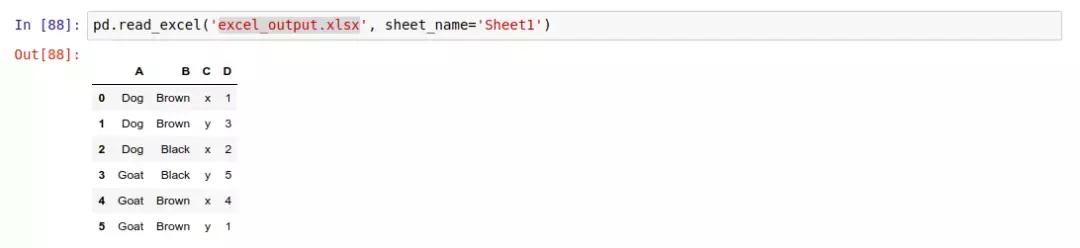
这里传入 index=False 参数是因为不希望 Pandas 把索引列的 0~5 也存到文件中。

为了确保数据已经保存好了，你可以试试用 pd.read\_csv('New\_dataframe') ，把这个文件的内容读取出来看看。

#### 读取 Excel 表格文件

Excel 文件是一个不错的数据来源。使用 pd.read\_excel() 方法，我们能将 Excel 表格中的数据导入 Pandas 中。请注意，Pandas 只能导入表格文件中的数据，其他对象，例如宏、图形和公式等都不会被导入。如果文件中存在有此类对象，可能会导致 pd.read\_excel() 方法执行失败。

举个例子，假设我们有一个 Excel 表格 'excel\_output.xlsx'，然后读取它的数据：



请注意，每个 Excel 表格文件都含有一个或多个工作表，传入 sheet\_name='Sheet1' 这样的参数，就表示只读取 'excel\_output.xlsx' 中的 Sheet1 工作表中的内容。

#### 写入 Excel 表格文件

跟写入 CSV 文件类似，我们可以将一个 DataFrame 对象存成 .xlsx 文件，语法是 .to\_excel() ：



和前面类似，把数据存到 'excel\_output.xlsx' 文件中：

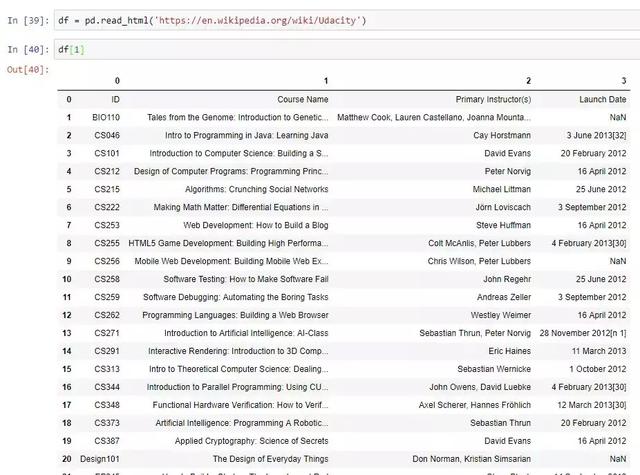


#### 读取 HTML 文件中的数据

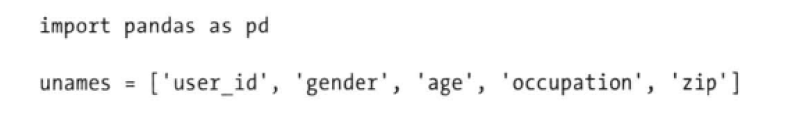
为了读取 HTML 文件，你需要安装 htmllib5，lxml 以及 BeautifulSoup4 库，在终端或者命令提示符运行以下命令来安装：

举个例子，我们用让 Pandas 读取这个页面的数据： https://en.wikipedia.org/wiki/Udacity 。由于一个页面上含有多个不同的表格，我们需要通过下标 [0, ..., len(tables) - 1] 访问数组中的不同元素。

下面的这个例子，我们显示的是页面中的第 2 个表格：



#### 从dat中读入



names为指定列名

sep为分隔符，默认为’\t’

skiprows:开头跳过的行数

skiofooter：结尾跳过的行数

header='infer' 默认会自动推断数据文件头,如果设置为None则无文件头,为1则第一行是文件头 sdf

usecols：列表型的数据，指定使用那几列数据。如果列表列有名，则使用列表名，如果没有，则安顺序指定。

squeeze：布尔型，如果数据只有一列，则返回series。

prefix ：在没有列名时，添加前缀。没有列名时，列名默认为0、1、2…，添加前缀就是在序号前添加。如添加前缀x，则列名变为x0 、x1等。

其他参数见<http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_table.html?highlight=read_table>