红色标注的语句，为重点。
蓝色下划线标注的语句，说明给出了参考阅读链接，可依兴趣阅读。

Scipy入门 —— Pandas篇

Pandas是一个调用便捷、运算高效的、帮助Python处理类表格数据的模块。基于Python使其调用便捷；基于Numpy使其运算高效；能够处理表格类数据让它成为了Python数据分析必须掌握的基本工具。

Pandas是一个第三方模块，由一整个团队在基金支持下负责维护。因此需要先行安装和调用，安装模块的方法见Lecture1，使用Anaconda的同学可以忽略这一步。

我们通常使用以下语句声明对Pandas的调用：

```
In [ ]: from pandas import DataFrame, Series
        import pandas as pd
```

我们约定俗成地将pandas命名为pd，方便后续调用，引入DataFrame和Series类是为了方便后续展示，默认Lecture4中所有代码均在该前提下执行，不再赘述。

1. Pandas的数据结构

——Dataframe

由于Pandas主要处理表格数据，因此其最重要的数据结构：Dataframe也是以一张二维表的形式设置的。

```
In [6]: df = pd.DataFrame({
            'name':['张三','李四','王五'],
            'age':[20,21,22],
            'sex':['man','women','man']
        })

        df
```

Out[6]:

	name	age	sex
0	张三	20	man
1	李四	21	women
2	王五	22	man

如图所示，建立DataFrame的方法，通过pd.DataFrame()函数，将所有数据传入一个字典Dict中（花括号内表示一个字典，复习见Lecture1附件），字典中的每一个键是列名，该键对应的值即为该列的内容，以列表List形式呈现（方括号内表示一个列表，复习见Lecture1附件）。

要记住的是，DataFrame的建立是按照“列”来传入数据的，并不像我们录入Excel时一行行输入。

通常，我们使用命名一个变量df来存储一个DataFrame，可以很直观地看到，DataFrame在Jupyter里的呈现很像Excel，有着每行的索引，也有着列标题来充当表头（header）。

DataFrame可以存储多种类型的数据，如数字、文本，甚至是对象（不建议大家这么存）。但是，出于效率优化考量，DataFrame的每一列只能存储同一种类型的数据！事实上，这也符合Excel或者数据库中的数据规范。

——Series

此前提到，DataFrame的众多操作都是基于列，因此，Pandas专门设计了一种数据类型来存储列中的一维数据：Series。

```
In [10]: s = pd.Series(['张三','李四','王五'])
         s

Out[10]: 0    张三
         1    李四
         2    王五
         dtype: object
```

Series被专门用来存储相同的一列数据，并且给这一列数据设计了索引，如上图，不指定索引的话，Series很像一个列表List，默认生成从0开始的数字索引。

但我们也可以自己修改索引，只需修改该Series对象的index属性即可：

```
In [12]: s = pd.Series(['张三','李四','王五'])
         s.index = ['No.1','No.2','No.3']
         s

Out[12]: No.1    张三
         No.2    李四
         No.3    王五
         dtype: object
```

甚至可以在建立Series对象的时候，就指定其index：

```
In [11]: s = pd.Series({'No.1':'张三','No.2':'李四','No.3':'王五'})
         s

Out[11]: No.1    张三
         No.2    李四
         No.3    王五
         dtype: object
```

在这种情况下建立Series对象，我们需要传入的数据是以字典Dict形式出现的，每个键代表一个索引，该键对应的值就是索引对应的值。

因此，我们可将Series理解成列表List和字典Dict的混合体。它既像列表一样有序、可以索引，但又像字典一样，索引可以自定义为字符串等其他类型。

2. 文件读写

——多多使用help( )和dir( )命令

从第二部分开始，为了执行各种操作，会涉及大量的Pandas预定义函数。如果对某一个函数不了解，可以使用Python内置的帮助功能来进行学习。

help( )命令会提供传入对象的帮助文档，例如对于一个类，或对于一个函数，可以查询到开发者对其的详细解释，例如如何实例化一个类，一个函数的用途等。

dir( )命令（传入对象时）会直接列出一个类的属性和方法，使用较少，不做过多阐述。

——Excel文件的读写

实际上，我们大多数时间都是和Excel文件打交道，Pandas对Excel读写的支持强大而优雅，远超过之前介绍的xlrd和xlwt模块。

读入.xls文件或.csv文件只需要一句命令，pd.read_excel( )或pd.read_csv( )，输出也只需要对应的df.to_excel( )或df.to_csv( )命令。（这里的df指DataFrame对象，这是DataFrame类的下属方法）

```
In [32]: df = pd.read_excel('data.xlsx')
         df
```

Out[32]:

	name	age	sex
0	张三	20	male
1	李四	21	female
2	王五	22	male
3	赵六	23	female

```
In [33]: df.to_csv('data.csv',index=None)
```

```
In [34]: df2 = pd.read_csv('data.csv')
         df2
```

Out[34]:

	name	age	sex
0	张三	20	male
1	李四	21	female
2	王五	22	male
3	赵六	23	female

——简单的查看

DataFrame提供了几种手段让我们在读取数据后对数据有一个基本的概览：

（注意，以下属性/方法都是归属于DataFrame对象的，不要试图用pd.xxx来调用）

属性/方法名	效果
df.dtypes属性	给出DataFrame每列的数据类型
df.head(x)方法	给出DataFrame的头x行
df.tail(x)方法	给出DataFrame的末x行
df.info( )方法	给出DataFrame的每列的信息，如空值数、数据类型

3. 选取数据

——根据列选取

DataFrame的操作大多围绕列展开，因此提供了众多根据列来选取数据的方法。

我们可以简单地根据列名选取数据，可以是一个列名，也可以以列表形式传入多个列名：

```
In [51]: df = pd.read_excel('data.xlsx')
         print(df['age'])
         print(df[['name','age']])
```

```
0    20
1    21
2    22
3    23
Name: age, dtype: int64
  name  age
0   张三   20
1   李四   21
2   王五   22
3   赵六   23
```

我们也可以为列设定条件，筛选符合其条件的行。可以设定一个条件，也可以用“&”连接多个条件（注意用括号括起每一个子条件）：

```
In [56]: df = pd.read_excel('data.xlsx')
         print(df)
         print(df['age']<=21)
         print(df[df['age']<=21])
         print(df[ (df['age']<=21) & (df['sex']=='male') ])
```

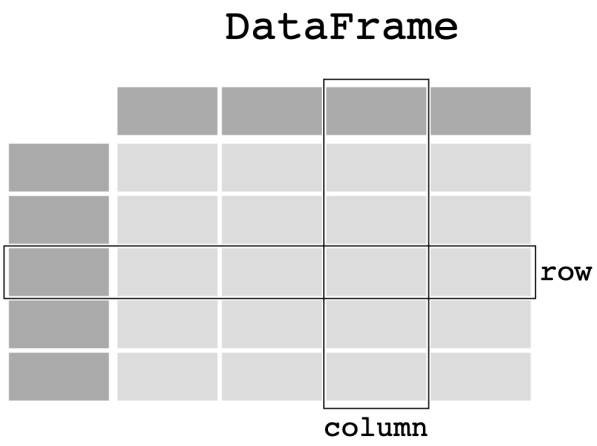
```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
0     True
1     True
2    False
3    False
Name: age, dtype: bool
  name  age     sex
0   张三   20    male
1   李四   21  female
  name  age   sex
0   张三   20  male
```

注意⚠️，上图中的df[‘age’]<=21这一语句，并不起到筛选数据的作用，它只是对df的age这一列中，做<=21这个条件的判断，返回一个由Bool类型组成的Series，以告诉我们age这一列中每个元素是否满足<=21这个条件。

但如果把df[‘age’]<=21这一语句放入df[ ]中，就是筛选df中所有满足age这一列<=21的行，重新生成一个DataFrame来存储。请注意区分这同一个语句在不同场景下的使用区别。

——根据行与列选取

DataFrame是一个二维的数据表，每一行是一个row，而每一列代表一个column，每一行的索引由index存储，每一列的索引由columns存储。（注意，索引中，index不为复数形式，但columns为）



因此，一个很自然的想法就是，按照坐标一样，使用索引index和colunms来定位某一批数据。

df.loc[ ]接受index和column的名称来定位符合条件的行与列（注意，是方括号）：

```
In [83]: df = pd.read_excel('data.xlsx')
         print(df)
         print(df.loc[0,'name'])
         print(df.loc[[0,1],['name','age']])
         print(df.loc[[0,1],:])
         print(df.loc[:,:])
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
张三
  name  age
0   张三   20
1   李四   21
  name  age     sex
0   张三   20    male
1   李四   21  female
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
```

可以看到，我们可以精准地定位到某一个元素，也可以定位某几行某几列的元素。值得注意的是，使用“:”符号来表示所有行/列。

df.iloc[ ]接受index和column的数字来定位符合条件的行与列（注意，也是方括号）：

```
In [89]: df = pd.read_excel('data.xlsx')
         print(df)
         print(df.iloc[0,0])
         print(df.iloc[[0,1],[0,1]])
         print(df.iloc[0:3,:])
         print(df.iloc[:,:])
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
张三
  name  age
0   张三   20
1   李四   21
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
```

值得注意的是，由于df.iloc[ ]使用数字来进行定位，其也支持类似Python列表List的使用“:”符号来切片处理的方式（可见Lecture1附件），例如df.iloc[0:3,:]代表第0-2行，全部列的数据。

4. 新增数据

——按列增加

如果要在DataFrame末尾直接增加一列数据，仅需直接指定即可，也可以在某列后进行插入：

```
In [96]: df = pd.read_excel('data.xlsx')
         print(df)

         #在最后插入新列
         df['score'] = [90,91,92,93]
         print(df)

         #在某列后插入新列
         col = df.columns.to_list()
         col.insert(2,'group')
         df = df.reindex(columns=col)
         print(df)
         df['group'] = [1,2,3,4]
         print(df)
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
  name  age     sex  score
0   张三   20    male     90
1   李四   21  female     91
2   王五   22    male     92
3   赵六   23  female     93
  name  age  group     sex  score
0   张三   20    NaN    male     90
1   李四   21    NaN  female     91
2   王五   22    NaN    male     92
3   赵六   23    NaN  female     93
  name  age  group     sex  score
0   张三   20      1    male     90
1   李四   21      2  female     91
2   王五   22      3    male     92
3   赵六   23      4  female     93
```

注意，在执行“某列后插入新列”操作时，df.columns本身是一个Series对象，为了能让我们在指定位置插入“group”，需要先使用Series对象的to_list( )方法转换成列表List。此外，对DataFrame使用reindex方法重新指定columns时，会自动匹配已有的colunms，因此看到只有新增的group列没有数据，其他的都和先前保持一致。

——按行增加

一般我们使用append方法将某行内容加入到DataFrame的末尾。在指定位置插入一行意义不大（不像在指定位置插入列，可能出于方便相关的列放在一起，便于阅读的考虑），在这里不多介绍。

```
In [109]: df = pd.read_excel('data.xlsx')
          print(df)
          df = df.append(pd.Series({'name':'七七','age':24,'sex':'male'},name=4))
          print(df)
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
4   七七   24    male
```

可以看到，我们为df.append( )方法传入的对象实际上是一个Series，这也是该方法要求的，我们在创建该Series时指定了name属性，作为插入后的index。

注意⚠️，如果我们这里将name设置为3，并不会导致报错，而是会导致“七七”这一条记录的index和“赵六”一样，均为3，这会造成不好的影响，例如使用df.iloc[ ]选取数据的时候会将这两条都选上，为了避免这一现象，可以在df.append( )方法中添加参数ignore_index=True，这样在检查到index冲突时，系统会进行报错。

5. 统计数据

Pandas提供了强大的统计功能。支持min( ),max( ), mean( )等基本操作，这里着重介绍groupby( )这一实用函数。

```
In [110]: df = pd.read_excel('data.xlsx')
          print(df)
          print(df.groupby('sex').mean())
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
        age
sex
female   22
male     21
```

groupby( )系列操作的逻辑是“split，apply，combine”，如图，df.groupby(‘sex’).mean( )指的是，先按照sex这一列的不同将原DataFrame拆分开来，sex值相同的归为一个DataFrame（这就是split）；再对每个split出的子表执行mean( )操作（即为apply）；最后将每个子表计算出的mean值合并到一张表上（combine）。

关于groupby，具体见：https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html?highlight=groupby#pandas.DataFrame.groupby

6.拼接/组合数据

——pd.concat( )方法

可以将concat操作理解成“堆叠”，它允许我们将不同的DataFrame沿着x或y轴拼接起来，不考虑其他因素。

```
In [112]: df = pd.read_excel('data.xlsx')
          df1 = df.iloc[0:2,:]
          df2 = df.iloc[2:4,:]

          print(pd.concat([df1,df2]))
          print(pd.concat([df1,df2],axis=1))
```

```
  name  age     sex
0   张三   20    male
1   李四   21  female
2   王五   22    male
3   赵六   23  female
  name   age     sex name   age     sex
0   张三  20.0    male  NaN   NaN     NaN
1   李四  21.0  female  NaN   NaN     NaN
2  NaN   NaN     NaN   王五  22.0    male
3  NaN   NaN     NaN   赵六  23.0  female
```

concat操作需要指定axis（Lecture3中Numpy部分解释过axis的概念），axis=0即为按照纵方向堆叠，axis=1即为按照横方向拼接。

——pd.merge( )方法

Merge方法对比较熟悉数据库操作的朋友应该易于理解。它对不同DataFrame之间的连接逻辑是按照数据库的主键进行整合的。

```
In [118]: df1 = pd.DataFrame({
              'name':['张三','李四','王五'],
              'age':[20,21,22],
              'sex':['man','women','man']
          })

          df2 = pd.DataFrame({
              'name':['张三','李四','王五'],
              'group':[1,2,3],
              'score':[90,91,92]
          })

          print(pd.merge(df1,df2))
```

```
  name  age    sex  group  score
0   张三   20    man      1     90
1   李四   21  women      2     91
2   王五   22    man      3     92
```

可以看到，两张DataFrame的共有主键为name，因此会自动以name为主键进行自动匹配。

但要注意，现实操作中，往往会有很多特殊情况出现，比如两张表的主键会有缺失，并不能完美匹配上等。

进一步了解pd.concat，pd.merge，可见：https://blog.csdn.net/gdkyxy2013/article/details/80785361