飞象工场

7 天 Python 基础训练营讲义

第六讲

2019 年 3 月 23 日-2019 年 3 月 29 日

下面是来自飞象君的温馨小提示(〃'▽'〃)

欢迎大家来到飞象工场的 7 天 python 训练营！！

本次课程我们会使用 wenwen 老师推荐的 Anaconda， 自带 jupyter notebook 和 python3，新手友好～

童鞋们在学习 python 时会可能会遇到各种各样的问题，是非常正常的，千万不要着急呦！这个时候我们需要沉下心来寻找解决的办法。工场助教们会尽力帮助大家，回复不过来的时候也请童鞋们谅解，并学会自己解决简单问题～最有效的就是在百度或者谷歌上搜索一下自己的问题，一般都能够找到答案~同时我们还要注意看 python 的 warning 提示，它会清楚地告诉大家问题的原因是什么，要怎么解决，觉得英文阅读起来有困难的童鞋可常备任意一款英汉字典～

学习编程就是一个不断探索、出现问题解决问题并坚持到底的过程，大家一起加油！！

目录

一、 pd.Series

这一讲和下一讲的内容，我们将主要围绕 pandas 这个包来展开，并会带到一点 numpy 的知识。所以大家在开始课程之前，要记得把 pandas 和 numpy 这两个包安装并命名好哦~具体如何操作可以参看第一讲的作业答案。

Pandas 是基于 numpy 的一个包，其数据格式有 series 和 dataframe 两种，其中 series 是一列一列数据的感觉，和 numpy 里的 array 有些相像，而 dataframe 是类似表格的感觉。Pandas 可以进行整行整列的处理，非常方便快速。那么我们首先讲 series。

1. pd.Series

就像上面所说的，pd.Series 是一个一维的数列的感觉，并且本身就是一个函数，所以我们在用的时候可以直接创建一个对象，并在括号内填入一个列表。

例 1：

```
s1 = pd.Series([1,2,3,4,5])  #把列表变成series打印出来
print(s1)
```

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

这样，列表就被转化成竖着打印的一列数字（右边一列）。左边的一列是 index 索引值，在之后会有更详细的说明。

我们可以用 s1.values 查看 series 里的值,可以发现输出的是 array 形式。这也体现了 pandas 是基于 numpy 建立的一个包。

例 2：

```
s1.values    #查看s1里的值
```

```
array([1, 2, 3, 4, 5])
```

另外我们可以用 s1.index 来查看左边一列的数。也就是对列表里每一个数的位置的索引。

例 3：

```
s1.index    #print的左边那一列就是Index
```

```
RangeIndex(start=0, stop=5, step=1)
```

2. 索引

和列表相似，我们可以通过在 series 的某个位置找到那个数。比如说，在列表[1,2,3,4,5]中，3 位于 2 号位上。那么把它转换成 series 以后，查找 2 号位，输出的也是 3.

例 4：

```
s1[2] #查看s1里2号位置的值是多少
```

```
3
```

但 series 也有和列表不一样的地方，在 series 中我们可以自定义索引是什么样子。在查看 index 的时候，发现也和之前默认的 0，1，2，3，4 这个 index 不同了。

例 5：

```
s2 = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e'])  #新建一个s2，同时自定义一个index。Index与值一一对应。
print(s2)
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

```
s2.index  #查看s2的index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

那么在这种情况下，要找 3 这个值，就要输入我们设置的 index“c”.

例 6：

```
s2['c']  #这次找3这个值，输入的是‘c’
```

```
3
```

但其实，输入 3 所处的位置仍然可以找到这个数。

例 7：

```
s2[2]   #输入2同样可以找到3这个值
```

```
3
```

总结一下，列表只有默认的不显示的位置索引，也就是 0,1,2,3,4 这样。但 series 可以用默认的位置索引，也可以自己设置一个索引方式，我们不论用哪一种方式都可以找到想要的那个值。

另外，如果一开始写 series 的时候忘记设置 index，也可以在之后用 s1.index 给 index 赋值，也就是重新自定义 index。

例 8：

```
s1
```

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

```
s1.index = ['a','b','c','d','e']  #查看s1的index
```

```
print(s1)
s1.index
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64

Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

3. series 中的运算

之前也说过，pandas 里可以直接对整行整列进行运算。比如把之前定义的 s1 和 s2 相加，同一个位置上的数就会点对点相加，然后输出一个新的 series。

例 9：

```
s1+s2  #pandas可以对整行整列进行快速的运算，直接将s1和s2相加，点对点（index要相同）相加。
```

```
a     2
b     4
c     6
d     8
e    10
dtype: int64
```

但是，如果索引不相同（比如一个用默认的索引，一个用自己设置的索引），就没有办法相加。

例 10：

```
s3 = pd.Series([1,2,3,4])    #定义一个s3，默认索引index。
s3
```

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

```
s1+s3  #它们不可以相加，因为它们没有共有的索引。
```

```
a   NaN
b   NaN
c   NaN
d   NaN
e   NaN
0   NaN
1   NaN
2   NaN
3   NaN
dtype: float64
```

如果把 s3 的 index 改为 a,b,c,d，那么就可以相加了。由于 s3 里只有 4 个数，找不到 e 位置的数，所以输出了缺失值。而且因为出现了

缺失值，所以把前面的数字全都变成了浮点数。

例 11：

```
s3.index = ['a','b','c','d']  #给s3重新定义index,和s1相同的。
```

```
s4 = s1+s3
print(s4)   #这次就可以相加了。  因为s3中没有e位置的数字，因此没办法相加。是一个缺失值。
            #因为这一列出现了缺失值，因此输出的结果把之前的整数都变成了浮点数。
```

```
a    2.0
b    4.0
c    6.0
d    8.0
e    NaN
dtype: float64
```

可以用 isnull 查找一个 series 中哪些是缺失值，是缺失值就输出 True，否则就输出 False。也可以用 notnull，和前面相反，是缺失值就输出 False，不是就输出 True。

例 12：

```
s4.notnull()  #isnull:是缺失值就输出True。  notnull:是缺失值就输出False，因此可以看到e的位置出现了False
```

```
a     True
b     True
c     True
d     True
e    False
dtype: bool
```

二、 pd.DataFrame

1. pd.DataFrame

DataFrame 是一个类似表格的形式，输入一个字典，就可以输出一个表格。不加 print 的话是 notebook 自动显示的版本，实际和加了 print 是一样的。

例 1：

```
dat1 = {'A':[1,2,3,4],'B':[3,4,5,6],'C':[4,5,6,7],'D':[11,12,13,14]} #新建一个字典dat1
df1 = pd.DataFrame(dat1)    #D和F一定要大写。
print(df1)
df1   #notebook自动显示的版本
```

```
   A  B  C   D
0  1  3  4  11
1  2  4  5  12
2  3  5  6  13
3  4  6  7  14
```

	A	B	C	D
0	1	3	4	11
1	2	4	5	12
2	3	5	6	13
3	4	6	7	14

注：D 和 F 一定要记得大写哦！

但如果字典里，每一个键的键值只有一个数的话，转换成 dataframe 时会报错。需要你给它设置一个 index。

例 2：不设置 index，报错

```
dat2 = {'A':1, 'B':2, 'C':3, 'D':4}
df2 = pd.DataFrame(dat2)     #python认为你输入的不是数列而是一个数，需要自定义一个index
df2
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-4-fbb935351bd6> in <module>
      1 dat2 = {'A':1, 'B':2, 'C':3, 'D':4}
----> 2 df2 = pd.DataFrame(dat2)
      3 df2

~\Anaconda3\lib\site-packages\pandas\core\frame.py in __init__(self, data, index, columns, dtype, copy)
    346                                 dtype=dtype, copy=copy)
    347         elif isinstance(data, dict):
--> 348             mgr = self._init_dict(data, index, columns, dtype=dtype)
    349         elif isinstance(data, ma.MaskedArray):
    350             import numpy.ma.mrecords as mrecords

~\Anaconda3\lib\site-packages\pandas\core\frame.py in _init_dict(self, data, index, columns, dtype)
    457             arrays = [data[k] for k in keys]
    458
--> 459         return _arrays_to_mgr(arrays, data_names, index, columns, dtype=dtype)
    460
    461     def _init_ndarray(self, values, index, columns, dtype=None, copy=False):

~\Anaconda3\lib\site-packages\pandas\core\frame.py in _arrays_to_mgr(arrays, arr_names, index, columns, dtype)
   7354     # figure out the index, if necessary
   7355     if index is None:
-> 7356         index = extract_index(arrays)
   7357
   7358     # don't force copy because getting jammed in an ndarray anyway
```

```
~\Anaconda3\lib\site-packages\pandas\core\frame.py in extract_index(data)
   7391
   7392         if not indexes and not raw_lengths:
-> 7393             raise ValueError('If using all scalar values, you must pass'
   7394                              ' an index')
   7395

ValueError: If using all scalar values, you must pass an index
```

例 3：设置一个 index 即可

```
dat2 = {'A':1, 'B':2, 'C':3, 'D':4}      #同样新建一个字典
df2 = pd.DataFrame(dat2, index = [0])
df2
```

	A	B	C	D
0	1	2	3	4

也可以在定义字典的时候，把一个整数变成一个数列，这样就不需要加 index 了。

例 4:

```
dat3 = {'A':[1], 'B':[2], 'C':[3], 'D':[4]}
df3 = pd.DataFrame(dat3)  #此时就是包含一个数的数列列表，不需要自定义index也可以了。
df3
```

	A	B	C	D
0	1	2	3	4

2. np.random.randn

这是一个用来生成随机数的函数，数据类型也是一个多维的 array。

例 5：一维的数列

```
np.random.randn(3)  #用来生成随机数的函数，这里相当于生成1维：1*3的数列
```

```
array([ 0.1904534 ,  0.22547748, -0.84024409])
```

例 6：三维的数组

```
np.random.randn(2,3,3) #生成了一个3维的数组。2个3*3的数组。
```

```
array([[[ 0.26050776,  0.22951066, -2.06442036],
        [-0.55918321,  1.71016914,  0.2826039 ],
        [-1.2157877 , -1.05661059, -0.96968164]],

       [[ 0.62282187,  0.08106054,  0.06977985],
        [-1.05882272, -0.0621616 ,  1.75176964],
        [ 1.02828284,  1.31675856,  0.50949415]]])
```

这个函数比较常用于做模拟的情况，下面结合 dataframe 举一个栗子。首先使用这个函数生成一个 10 行 4 列的数组，然后转换成 dataframe，并定义每一列的名字（column）和 index。

例 7：

```
dat4 = np.random.randn(10,4)  #生成了10*4的数列，10行4列。
df4 = pd.DataFrame(np.random.randn(10,4),columns = ['A','B','C','D']  #columns 是每一列的名字，前面额字典默认键就是columns了。
                   ,index = ['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',  #index 是每一行的索引
                             '2018-01-05', '2018-01-06', '2018-01-07', '2018-01-08',
                             '2018-01-09', '2018-01-10'])
df4
```

	A	B	C	D
2018-01-01	-1.759251	-3.613955	-1.818334	-0.902213
2018-01-02	-0.719548	-0.604099	-0.316759	-0.610632
2018-01-03	1.000082	-1.176882	0.659925	1.927054
2018-01-04	-0.429215	1.834204	0.058672	1.517117
2018-01-05	0.306051	0.893561	3.401997	0.409252
2018-01-06	-0.932212	0.092858	-0.571614	-0.946640
2018-01-07	2.347043	-0.411748	-0.308534	-0.997240
2018-01-08	-0.246878	-0.598923	0.180351	0.388415
2018-01-09	1.097674	-0.680087	-0.897286	0.615629
2018-01-10	-0.439487	1.483093	-0.664894	-1.450790

3. 索引

在 dataframe 里索引要用列的名字，即 column，这时搜出来的是那一列数，以及前面的 index。用每一行的 index 会报错，显示搜不到这个名字。

例 8：

```
df4['A']   #索引的时候放的是列的名字。
```

```
2018-01-01   -1.759251
2018-01-02   -0.719548
2018-01-03    1.000082
2018-01-04   -0.429215
2018-01-05    0.306051
2018-01-06   -0.932212
2018-01-07    2.347043
2018-01-08   -0.246878
2018-01-09    1.097674
2018-01-10   -0.439487
Name: A, dtype: float64
```

我们也可以同时搜索多列数据，这个时候要记得用 2 层中括号。

例 9：

```
df4[['A','B','D']] #因为我们找的是3列，所以相当于找的是列表，所以这里是两层中括号。
```

	A	B	D
2018-01-01	-1.759251	-3.613955	-0.902213
2018-01-02	-0.719548	-0.604099	-0.610632
2018-01-03	1.000082	-1.176882	1.927054
2018-01-04	-0.429215	1.834204	1.517117
2018-01-05	0.306051	0.893561	0.409252
2018-01-06	-0.932212	0.092858	-0.946640
2018-01-07	2.347043	-0.411748	-0.997240
2018-01-08	-0.246878	-0.598923	0.388415
2018-01-09	1.097674	-0.680087	0.615629
2018-01-10	-0.439487	1.483093	-1.450790

如果要找某几行的数据的话，就要使用冒号，就像在列表中进行切片的感觉。

例 10：

```
df4['2018-01-01':'2018-01-02'] #冒号，类似于之前的切片。告诉系统从行开始找。
```

	A	B	C	D
2018-01-01	-1.759251	-3.613955	-1.818334	-0.902213
2018-01-02	-0.719548	-0.604099	-0.316759	-0.610632

另外，如果只找一列数据的话，输出的类型是 series，找多列输出的类型就是 dataframe（但其实这个是根据你用的是一层括号还是 2 层括号来决定的，即使只找一列数据，如果用的是 2 层中括号的话，输出的类型也是 dataframe）。

例 11：一层中括号

```
print(type(df4['A']))
print(type(df4[['A','B','D']]))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
```

例 12：2 层中括号

```
print(type(df4[['A']]))
print(type(df4[['A','B','D']]))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

要找其中的一个数的话，先找一列数据，这时就变成了一个 series，然后可以用自定义的 index 名字或者位置来找到那一个数。

例 13：

```
df4['A']['2018-01-01']
```

```
-1.7592514080168924
```

```
df4['A'][0]    #两种方式来索引这一列中的一个数
```

```
-1.7592514080168924
```

也可以实现只找某几列的某几行数，先找列还是先找行都可以成功。

例 14：

```
df4[['A','B']]['2018-01-01':'2018-01-01'] #选取两个数，位置互换来找也可以成功。
```

	A	B
2018-01-01	-1.759251	-3.613955

注：要只找一行数据的话，记得要像例 14 里这样写，不能只写一个[‘2018-01-01’]哦！

4. loc&iloc

要进行精准索引的话，可以使用 loc 和 iloc 这两个方程。

loc 的中括号里面，逗号左边填关于行的索引（可以跳着写），右边填关于列的索引，填的都是真实的名字，然后就可以实现快速准确的索引了。

例 15：

```
df4.loc[['2018-01-01','2018-01-03'],['A','B']]  #常用的索引方式1：loc 中括号里左边是行的索引，右边是列的索引。填入的是真实的列名、行名
```

	A	B
2018-01-01	-1.759251	-3.613955
2018-01-03	1.000082	-1.176882

iloc 也是左边填行，右边填列，使用的是默认的位置 index。

例 16：

```
df4.iloc[[0,1,2],[0,1]] #常用的索引方式2：iloc填入的index columns都是数。更加方便。
```

	A	B
2018-01-01	-1.759251	-3.613955
2018-01-02	-0.719548	-0.604099
2018-01-03	1.000082	-1.176882

5. 赋值和简单运算

可以使用 iloc 直接给表格中的一些数赋值。如例 17。

例 17：

```
df4.iloc[[0],[0,1]] = 0 #让第0行第0，1列的两个值为0。
df4
```

	A	B	C	D
2018-01-01	0.000000	0.000000	-1.818334	-0.902213
2018-01-02	-0.719548	-0.604099	-0.316759	-0.610632
2018-01-03	1.000082	-1.176882	0.659925	1.927054
2018-01-04	-0.429215	1.834204	0.058672	1.517117
2018-01-05	0.306051	0.893561	3.401997	0.409252
2018-01-06	-0.932212	0.092858	-0.571614	-0.946640
2018-01-07	2.347043	-0.411748	-0.308534	-0.997240
2018-01-08	-0.246878	-0.598923	0.180351	0.388415
2018-01-09	1.097674	-0.680087	-0.897286	0.615629
2018-01-10	-0.439487	1.483093	-0.664894	-1.450790

也可以直接新加一列赋值，如例 18.

例 18：

```
df4['E'] = 3.33 #新加一列'E'，等于3.33
df4
```

	A	B	C	D	E
2018-01-01	0.000000	0.000000	-1.818334	-0.902213	3.33
2018-01-02	-0.719548	-0.604099	-0.316759	-0.610632	3.33
2018-01-03	1.000082	-1.176882	0.659925	1.927054	3.33
2018-01-04	-0.429215	1.834204	0.058672	1.517117	3.33
2018-01-05	0.306051	0.893561	3.401997	0.409252	3.33
2018-01-06	-0.932212	0.092858	-0.571614	-0.946640	3.33
2018-01-07	2.347043	-0.411748	-0.308534	-0.997240	3.33
2018-01-08	-0.246878	-0.598923	0.180351	0.388415	3.33
2018-01-09	1.097674	-0.680087	-0.897286	0.615629	3.33
2018-01-10	-0.439487	1.483093	-0.664894	-1.450790	3.33

也可以直接对整列进行运算，如例 19.

例 19：

```
df4['F'] = df4['A']+df4['B']-df4['C']  #新建一列F，可以直接进行整列计算。F列等于A列加B列减去C列。大大滴提高了运算速度。
df4
```

	A	B	C	D	E	F
2018-01-01	0.000000	0.000000	-1.818334	-0.902213	3.33	1.818334
2018-01-02	-0.719548	-0.604099	-0.316759	-0.610632	3.33	-1.006888
2018-01-03	1.000082	-1.176882	0.659925	1.927054	3.33	-0.836725
2018-01-04	-0.429215	1.834204	0.058672	1.517117	3.33	1.346317
2018-01-05	0.306051	0.893561	3.401997	0.409252	3.33	-2.202385
2018-01-06	-0.932212	0.092858	-0.571614	-0.946640	3.33	-0.267741
2018-01-07	2.347043	-0.411748	-0.308534	-0.997240	3.33	2.243829
2018-01-08	-0.246878	-0.598923	0.180351	0.388415	3.33	-1.026152
2018-01-09	1.097674	-0.680087	-0.897286	0.615629	3.33	1.314873
2018-01-10	-0.439487	1.483093	-0.664894	-1.450790	3.33	1.708500

这样整列运算可以避免循环的问题，大大提高运算速度哦~

这一讲的内容就到这里啦，我们主要学习了 pandas 的两个最重要的数据格式，series 和 dataframe，以及与之相关的索引和运算等操作。这是我们接触 pandas 包的第一步，也对未来我们使用 pandas 做数据分析等工作有很大的帮助，所以大家记得多多练习，熟练掌握哦~

在下一讲，也就是我们训练营的最后一天里，老师将给我们讲述用 pandas 实现的更多炫酷操作哦~是不是很期待呢~

第五讲作业答案：

Q1：使用 Data1，画一个柱状图

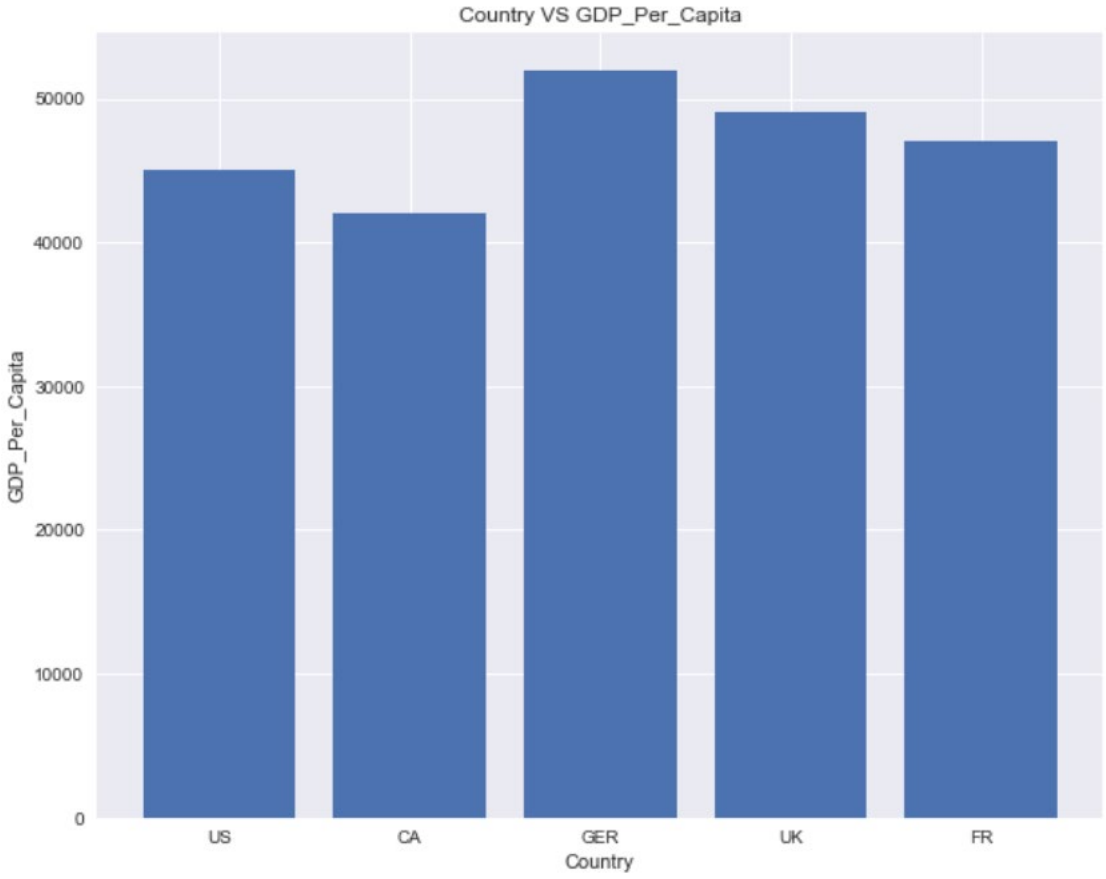
Q2：使用 Data2，画一个线形图

Q3：使用 Data3，分别用 matplotlib 和 seaborn 画一个点状图

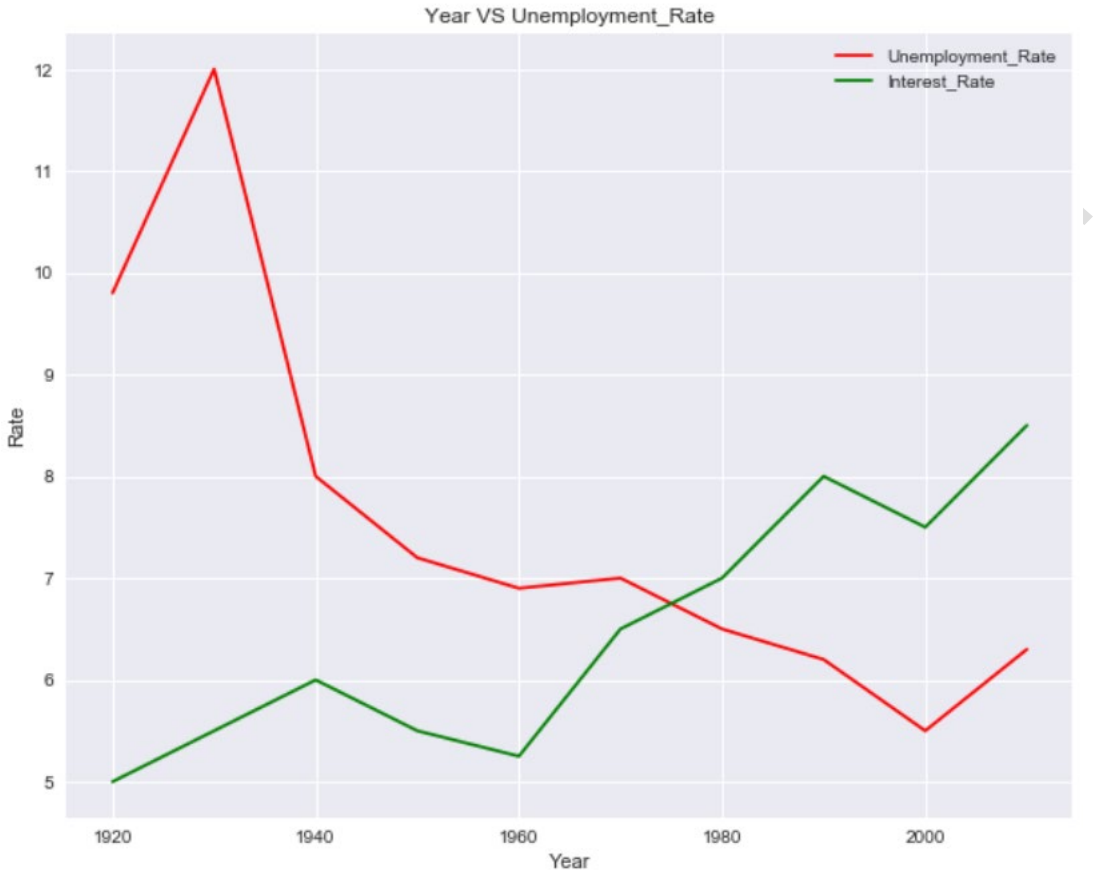
答案：

```
import matplotlib.pyplot as plt
import seaborn as sns
```

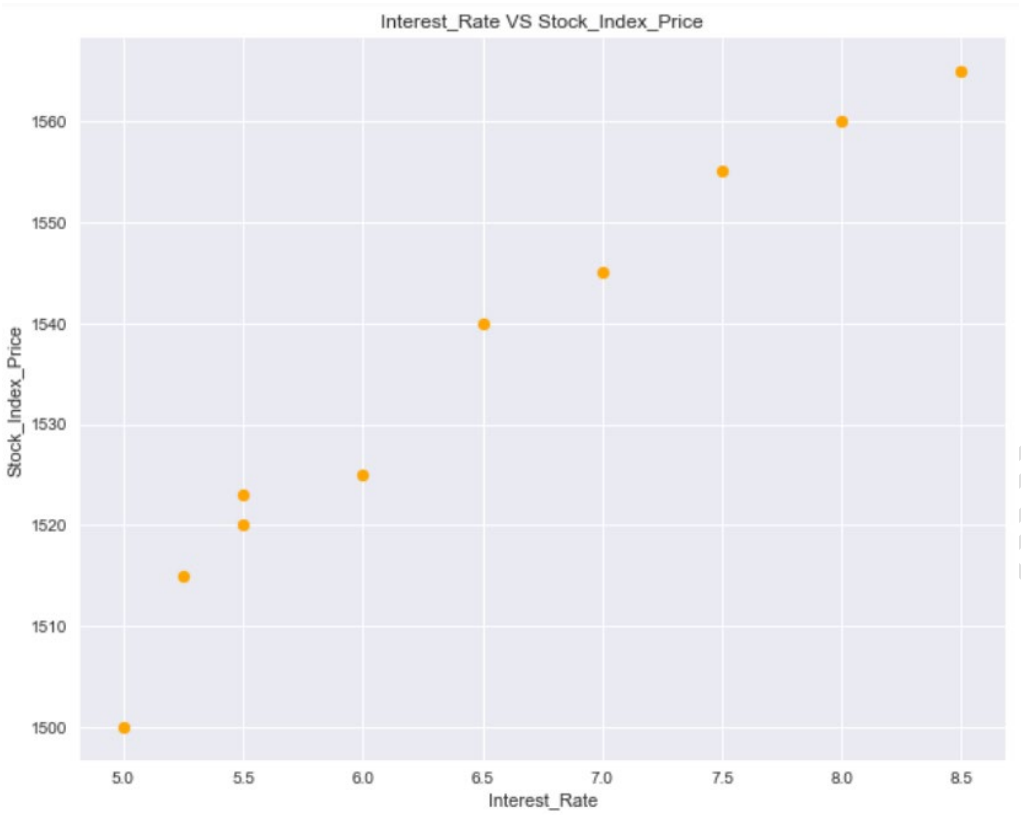
```
## Q1
plt.figure(figsize=(10,8))
plt.bar(range(5),Data1['GDP_Per_Capita'])
plt.xticks(range(5), Data1['Country'])
plt.title('Country VS GDP_Per_Capita')
plt.xlabel('Country')
plt.ylabel('GDP_Per_Capita')
plt.show()
```



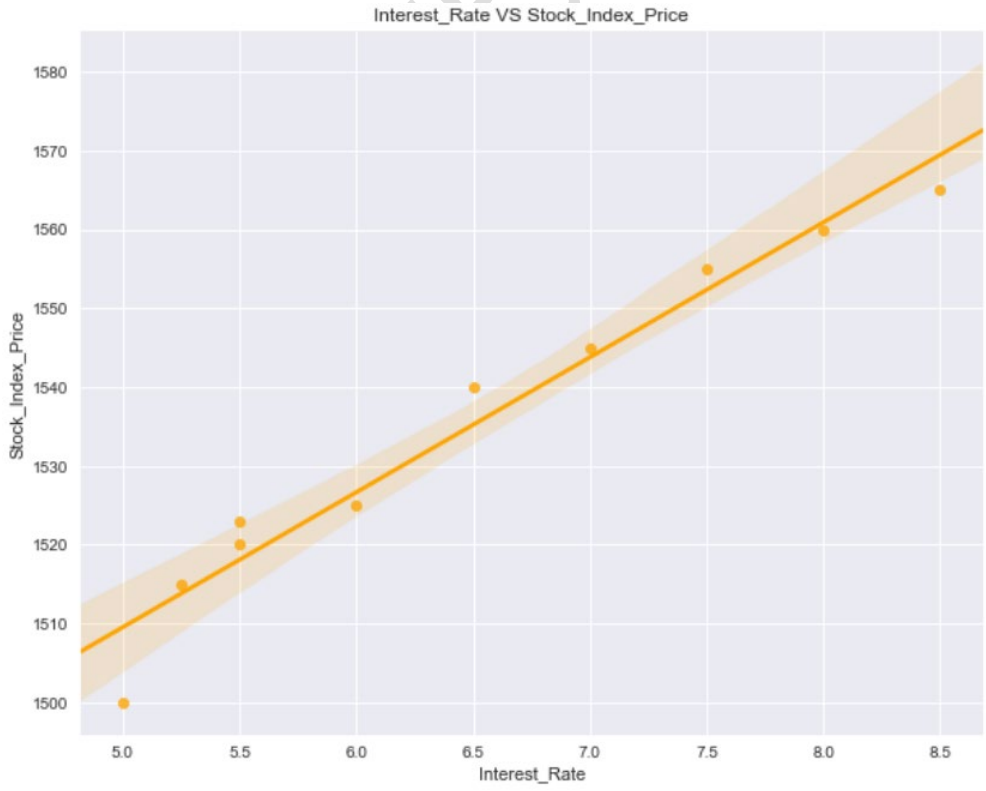
```
## Q2
plt.figure(figsize=(10,8))
plt.plot(Data2['Year'],Data2['Unemployment_Rate'],color = 'red',label = 'Unemployment_Rate')
plt.plot(Data2['Year'],Data2['Interest_Rate'],color = 'green',label = 'Interest_Rate')
plt.title('Year VS Unemployment_Rate')
plt.xlabel('Year')
plt.ylabel('Rate')
plt.legend()
plt.show()
```




```
## Q3
plt.figure(figsize=(10,8))
plt.scatter(Data3['Interest_Rate'],Data3['Stock_Index_Price'],color = 'orange')
plt.title('Interest_Rate VS Stock_Index_Price')
plt.xlabel('Interest_Rate')
plt.ylabel('Stock_Index_Price')
plt.show()
```




```
plt.figure(figsize=(10,8))
sns.regplot(np.array(Data3['Interest_Rate']), np.array(Data3['Stock_Index_Price']),color = 'orange')
plt.title('Interest_Rate VS Stock_Index_Price')
plt.xlabel('Interest_Rate')
plt.ylabel('Stock_Index_Price')
plt.show()
```



第六讲作业：

Q1：

建立一个series如下

```
s1
```

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

通过更改index和赋值的方式使它变成这样。NaN值可以使用np.NaN

```
#A:
```

```
s1
```

```
h    1.0
i    2.0
j    8.0
k    4.0
l    NaN
dtype: float64
```

把s1变成一个pandas dataframe

```
#A:
```

	0
h	1.0
i	2.0
j	8.0
k	4.0
l	NaN

Q2：

用不同的方法构建DataFrame d1 和 d2. 请用设置好的seed，使得每次被填充的随机数是一样的

```
np.random.seed(15)
#A:
d1
```

	A	B	C	D
0	-0.312328	0.235569	-0.305170	0.689518
1	0.339285	-1.763605	-0.473748	0.410590
2	-0.155909	-1.095862	-0.200595	-0.564978
3	-0.501790	-1.087766	0.355197	0.599391

```
np.random.seed(108)
#A:
d2
```

	A	B	C	D
0	-1.026905	0.221749	1.130390	1.146185
1	-0.592734	0.118784	-0.484430	-1.944913
2	0.092077	0.902169	1.314469	0.771102
3	-0.540147	-0.284115	-0.889331	0.404169

更改d2的index = ['18-01','18-02','18-03','18-04']，并且使用3种方法切片，使得结果是这样的。

```
#A:
```

	B	C
18-02	0.118784	-0.484430
18-03	0.902169	1.314469

Q3：

运行以下代码，用心体会如何利用条件进行切片~

```
df = pd.DataFrame([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],columns = ['A','B','C','D'])
```

```
#根据条件进行切片
df[df['A']>3]
```

	A	B	C	D
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

```
df[df<10]
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0
2	9.0	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN

```
df[(df > 3) & (df < 10)]
```

	A	B	C	D
0	NaN	NaN	NaN	4.0
1	5.0	6.0	7.0	8.0
2	9.0	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN

对d1进行条件切片，使得结果如下

```
#A:
```

	A	B	C	D
2	-0.155909	-1.095862	-0.200595	-0.564978

扫码关注飞象工场，解锁更多精彩课程哦！