* 4-1 Python資料分析概論圖表之設定
* 4-2 NumPy各種圖表之呈現
* 4-3 Pandas多圖表繪製
* 4-4 CSV 檔案繪製圖表
* 4-5 Numpy 模組應用
* 4-6 隨機數的應用

學習目標

資料分析是由原始資料轉換的幾個步驟組成的過程，基於收集的資料進行處理，以產生可視化結果，並且可以通過數學模型化來進行預測。資料分析由以下幾個階段組成：

• 問題定義

•資料提取

•資料清理

•資料轉換

•資料探索

•預測建模

•模型驗證/測試

•結果的可視化和解釋

Python 本身核心就有支援資料分析的工具，包括串列、索引與分片、字典資料型態等，將在3-1中分別介紹。

若要處理大量資料，NumPy就是用於處理陣列資料的高效函式庫，將在3-2中介紹。當涉及嚴重的數據清理時，我們可以使用多功能的 Pandas 套件包，可以為 Python提供快速簡化的資料處理和資料分析工具。

# 4-1 圖表之設定

## 4-1-1 標題，文字，圖例設定

串列基本操作如下範例所示：

#### ICON2範例程式E4-1-1-1.ipynb

串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#串列)

方括號（[]）表示一個串列，串列中的各個元素用逗號分隔。

|  |  |
| --- | --- |
| In[1]: | [1, 2, 3, 1 , 2] |
| Out[1] | [1, 2, 3, 1, 2] |

|  |  |
| --- | --- |
| In[2]: | numbers = [1, 2, 3]  type(numbers) |
| Out[2] | list |

|  |  |
| --- | --- |
| In[3]: | words = ['cat', 'bat', 'rat', 'elephant']  words |
| Out[3] | ['cat', 'bat', 'rat', 'elephant'] |

|  |  |
| --- | --- |
| In[4]: | mixed\_elements = ['hello', 3.1415, True, None, 42]  mixed\_elements |
| Out[4] | ['hello', 3.1415, True, None, 42] |

|  |  |
| --- | --- |
| In[5]: | empty\_list = []  empty\_list |
| Out[5] | [] |

存取串列元素[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#存取串列元素)

|  |  |
| --- | --- |
| In[6]: | animals = ['cat', 'bat', 'rat', 'elephant']  animals[3] |
| Out[6] | ['bat', 'elephant'] |

|  |  |
| --- | --- |
| In[7]: | animals[0] + 'woman and ' + animals[1] + 'man' |
| Out[7] | 'catwoman and batman' |

負值索引(Negative Indexes[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#負值索引(Negative-Indexes)

如何獲取串列最後幾個元素?

|  |  |
| --- | --- |
| In[8]: | animals[-1] |
| Out[8] | 'elephant' |

|  |  |
| --- | --- |
| In[9]: | animals[-3] |
| Out[9] | 'bat' |

透過切片(Slices)獲得子串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#透過切片(Slices)獲得子串列)

animals [2]*是一個帶索引的串列（一個整數）* animals [1：4]是一個帶切片的串列（兩個整數）    在切片中，第一個整數是切片開始的索引。 第二個整數是切片結束的索引。 切片到但不包括第二個索引處的值。 切片被視為為新的串列值。

|  |  |
| --- | --- |
| In[10]: | animals[1:4] |
| Out[10] | ['bat', 'rat', 'elephant'] |

|  |  |
| --- | --- |
| In[11]: | animals[0:-1] |
| Out[11] | ['cat', 'bat', 'rat'] |

|  |  |
| --- | --- |
| In[12]: | animals[-2:-1] |
| Out[12] | ['rat'] |

作為快捷方式，可以在切片的冒號兩側省略一個或兩個索引，省略第一個索引與使用0或串列的開頭相同，省略第二個索引與使用串列的長度進行切片相同，串列的長度將切換到串列的末尾。

|  |  |
| --- | --- |
| In[13]: | animals[:3] |
| Out[13] | ['cat', 'bat', 'rat'] |

|  |  |
| --- | --- |
| In[14]: | animals[::2] |
| Out[14] | ['cat', 'rat'] |

經由len()指令取得串列長度[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#經由len()指令取得串列長度)

|  |  |
| --- | --- |
| In[15]: | fst\_sentence = ['Call', 'me', 'Ishmael']  len(fst\_sentence) |
| Out[15] | 3 |

利用索引更改串列中的值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#利用索引更改串列中的值)

|  |  |
| --- | --- |
| In[16]: | fst\_sentence = ['Call', 'me', 'Ishmael']  fst\_sentence[1] = 'him'  fst\_sentence |
| Out[16] | ['Call', 'him', 'Ishmael'] |

|  |  |
| --- | --- |
| In[17]: | fst\_sentence[0] = fst\_sentence[1]  fst\_sentence |
| Out[17] | ['him', 'him', 'Ishmael'] |

|  |  |
| --- | --- |
| In[18]: | fst\_sentence[-1] = 1000  fst\_sentence |
| Out[18] | ['him', 'him', 1000] |

串列連接和串列複製[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#串列連接和串列複製)

   +*運算符組合了兩個串列來創建一個新的串列* \*運算符也可以與列來和整數值一起使用來複製列來

|  |  |
| --- | --- |
| In[19]: | fst\_sentence = ['Call', 'me', 'Ishmael']  numbers = [1, 2, 3, 4]  concat = fst\_sentence + numbers  concat |
| Out[19] | ['Call', 'me', 'Ishmael', 1, 2, 3, 4] |

|  |  |
| --- | --- |
| In[20]: | fst\_sentence \* 3 |
| Out[20] | ['Call', 'me', 'Ishmael', 'Call', 'me', 'Ishmael', 'Call', 'me', 'Ishmael'] |

使用del語句從串列中刪除值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#使用del語句從串列中刪除值)

del語句將刪除串列中索引的值。 所有的值在刪除值之後的串列中將向上移動一個索引。

|  |  |
| --- | --- |
| In[21]: | fst\_sentence = ['Call', 'me', 'Ishmael']  del fst\_sentence[1]  fst\_sentence |
| Out[21] | ['Call', 'Ishmael'] |

使用sorted（）對List的值進行排序[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#使用sorted（）對List的值進行排序)

sorted（）函數返回相應串列的排序**副本**。

|  |  |
| --- | --- |
| In[22]: | fst\_sentence = ['Call', 'me', 'Ishmael']  sorted\_sentence = sorted(fst\_sentence)  print(sorted\_sentence)  print(fst\_sentence) |
| Out[22] | ['Call', 'Ishmael', 'me']  ['Call', 'me', 'Ishmael'] |
| In[23]: | numbers = [2, 3, 1, -5]  sorted\_numbers = sorted(numbers)  print(sorted\_numbers)  print(numbers) |
| Out[23] | [-5, 1, 2, 3]  [2, 3, 1, -5] |

順序(Sequences)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#順序(Sequences))

range返回給定步驟從開始到結束的一系列數字。

|  |  |
| --- | --- |
| In[24]: | range(5) |
| Out[24] | range(0, 5) |

|  |  |
| --- | --- |
| In[25]: | list(range(5)) |
| Out[25] | [0, 1, 2, 3, 4] |

|  |  |
| --- | --- |
| In[26]: | list(range(3, 10)) |
| Out[26] | [3, 4, 5, 6, 7, 8, 9] |

|  |  |
| --- | --- |
| In[27]: | list(range(3, 20, 4)) |
| Out[27] | [3, 7, 11, 15, 19] |

|  |  |
| --- | --- |
| In[28]: | list(range(30, 10, -2)) |
| Out[28] | [30, 28, 26, 24, 22, 20, 18, 16, 14, 12] |

使用sort（）方法對List的值進行排序[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-1-1.html#使用sort（）方法對List的值進行排序)

可以使用sort（）方法對數值串列或字串串列進行排序。 您還可以為reverse關鍵字參數傳遞True，以使sort（）按相反順序對值進行排序。

|  |  |
| --- | --- |
| In[29]: | values = [2, 5, 3.14, 1, -7]  values.sort()  values |
| Out[29] | [-7, 1, 2, 3.14, 5] |

|  |  |
| --- | --- |
| In[30]: | fst\_sentence = ['Call', 'me', 'Ishmael']  fst\_sentence.sort()  fst\_sentence |
| Out[30] | ['Call', 'Ishmael', 'me'] |

|  |  |
| --- | --- |
| In[31]: | values = [2, 5, 3.14, 1, -7]  values.sort(reverse=True)  values |
| Out[31] | [5, 3.14, 2, 1, -7] |

您不能對其中包含數值*和*字串值的串列進行排序，因為Python不知道如何比較這些值。

|  |  |
| --- | --- |
| In[32]: | values = ['Call', 'me', 'Ishmael', 2, 5, 3.14, 1, -7]  values.sort()  values |
| Out32[] | ---------------------------------------------------------------------------  TypeError Traceback (most recent call last)  <ipython-input-73-8905006236e4> in <module>()  1 values = ['Call', 'me', 'Ishmael', 2, 5, 3.14, 1, -7]  ----> 2 values.sort()  3 values  TypeError: unorderable types: int() < str() |

sort（）使用“ASCIIbetical order”而不是實際的字母順序來排序字串。 這意味著大寫字母在小寫字母之前。 因此，對小寫字母a進行排序，使其位於大寫字母Z之後。

|  |  |
| --- | --- |
| In[33]: | fst\_sentence = ['Call', 'call', 'me', 'Me', 'ishmael', 'Ishmael']  fst\_sentence.sort()  fst\_sentence |
| Out[33] | ['Call', 'Ishmael', 'Me', 'call', 'ishmael', 'me'] |

如果需要按常規字母順序對值進行排序，請在sort（）方法呼叫中為key關鍵字參數傳遞str.lower，使sort（）函數處理所有項目。 串列就好像它們是小寫而不實際更改串列中的值。

|  |  |
| --- | --- |
| In[34]: | fst\_sentence = ['a', 'z', 'A', 'Z']  fst\_sentence.sort(key=str.lower)  fst\_sentence |
| Out[34] | ['a', 'A', 'z', 'Z'] |

## 4-1-2 線條設定

#### ICON2範例程式E4-1-2-1.ipynb

索引和分片[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-2-1.html#索引和分片)

索引[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-2-1.html#索引)

對於一個有序序列，可以通過索引的方法來訪問對應位置的值。字串便是一個有序序列的例子，**Python**使用 [] 來對有序序列進行索引。

|  |  |
| --- | --- |
| In[1]: | s = "hello world"  s[0] |
| Out[1] | 'h' |

**Python**中索引是從 0 開始的，所以索引 0 對應與序列的第 1 個元素。為了得到第 4 個元素，需要使用索引值 4 。

|  |  |
| --- | --- |
| In[2]: | s[3] |
| Out[2] | 'l' |

除了正向索引，**Python**還引入了負索引值的用法，即從後向前開始計數，例如，索引 -2 表示倒數第 2 個元素：

|  |  |
| --- | --- |
| In[3]: | s[-2] |
| Out[3] | 'l' |

單個索引大於等於字串的長度時，會回應錯誤：

|  |  |
| --- | --- |
| In[4]: | s[11] |
| Out[4] | ---------------------------------------------------------------------------  IndexError Traceback (most recent call last)  <ipython-input-4-665bb6993e1f> in <module>()  ----> 1 s[11]  IndexError: string index out of range |

分片[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-2-1.html#分片)

分片用來從序列中提取出想要的子序列，其用法為：

var[lower:upper:step]

其範圍包括 lower ，但不包括 upper ，即 [lower, upper)， step 表示取值間隔大小，如果沒有默認為1。

|  |  |
| --- | --- |
| In[5]: | s |
| Out[5] | 'hello world' |

|  |  |
| --- | --- |
| In[6]: | s[1:3] |
| Out[6] | 'el' |

分片中包含的元素的個數為 3-1=2 。

也可以使用負索引來指定分片的範圍：

|  |  |
| --- | --- |
| In[7]: | s[1:-2] |
| Out[7] | 'ello wor' |

包括索引 1 但是不包括索引 -2 。

lower和upper可以省略，省略lower意味著從開頭開始分片，省略upper意味著一直分片到結尾。

|  |  |
| --- | --- |
| In[8]: | s[:3] |
| Out[8] | 'hel' |

|  |  |
| --- | --- |
| In[9]: | s[-3:] |
| Out[9] | 'rld' |

|  |  |
| --- | --- |
| In[10]: | s[:] |
| Out[10] | ['hello world' |

每隔两個取一個值：

|  |  |
| --- | --- |
| In[11]: | s[::2] |
| Out[11] | 'hlowrd' |

當step的值為負時，省略lower意味著從結尾開始分片，省略upper意味著一直分片到開頭。

|  |  |
| --- | --- |
| In[12]: | s[::-1] |
| Out[12] | 'dlrow olleh' |

當給定的upper超出字串的長度（注意：因為不包含upper，所以可以等於）時，Python並不會報錯，不過只會計算到結尾。

|  |  |
| --- | --- |
| In[13]: | s[:100] |
| Out[13] | 'hello world' |

## 4-1-3 標記樣式

如下範例所示：

#### ICON2範例程式E4-1-3-1.ipynb

字典資料型態提供了一種訪問和組織資料的靈活方式。 像串列一樣，字典是許多值的集合。 但與串列索引不同，字典索引可以使用許多不同的資料型態，而不僅僅是整數。 字典的索引稱為鍵，具有關聯值的鍵稱為鍵值對。 在代碼中，字典用大括號“{}”鍵入。

|  |  |
| --- | --- |
| In[1]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image |
| Out[1] | {'color': 'greyscale',  'size': 256789,  'type': 'jpg',  'address': 'file:PIC\_0390.jpg'} |

這為image變數分配了一個字典。 這本詞典的鍵是“'color”，“size”，“type”和“地址”。 這些鍵的值分別是''greyscale'，289983，''jpg'和'https：// upload.wikimedia.org/wikipedia/comons/7 / 7b / Moby\_Dick\_p510\_illustration.jpg'。。 您可以通過其鍵訪問這些值。

|  |  |
| --- | --- |
| In[2]: | image['color'] |
| Out[2] | 'greyscale' |

|  |  |
| --- | --- |
| In[3]: | image['size'] |
| Out[3] | 256789 |

字典仍然可以使用整數值作為鍵，就像列表使用整數作為索引一樣，但它們不必從“0”開始，可以是任何數字。

|  |  |
| --- | --- |
| In[4]: | image = {510: 'page in book', 'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image[510] |
| Out[4] | 'page in book' |

字典與串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#字典與串列)

與串列不同，字典中的項目是無序的。 名為values的串列中的第一項將是values [0]。 但字典中沒有“第一”項。 雖然項目的順序對於確定兩個串列是否相同很重要，但鍵值對在字典中鍵入的順序無關緊要。

|  |  |
| --- | --- |
| In[5]: | fst\_sentence = ['Call', 'me', 'Howard']  fst\_sentence\_juggled = ['Howard', 'me', 'Call']  fst\_sentence == fst\_sentence\_juggled |
| Out[5] | False |

|  |  |
| --- | --- |
| In[6]: | fst\_sentence = {1: 'Call', 2: 'me', 3: 'Howard'}  fst\_sentence\_juggled = {3: 'Howard', 2: 'me', 1: 'Call'}  fst\_sentence == fst\_sentence\_juggled |
| Out[6] | True |

由於字典不是有序的，因此不能像串列那樣切片。

訪問字典中的值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#訪問字典中的值)

要獲取與鍵關聯的值，請提供字典的名稱，然後將鍵放在一組方括號內。嘗試訪問字典中不存在的鍵將導致出現“KeyError”錯誤消息，就像列表的“超出範圍”的“IndexError”錯誤消息一樣。

|  |  |
| --- | --- |
| In[7]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image['size'] |
| Out[7] | 256789 |

|  |  |
| --- | --- |
| In[8]: | image['author'] |
| Out[8] | ---------------------------------------------------------------------------  KeyError Traceback (most recent call last)  <ipython-input-9-03c3970ec369> in <module>()  ----> 1 image['author']  KeyError: 'author' |

添加新的鍵值對[¶](file:///C:\\Users\\User\\TQC+%20網頁資料擷取與分析特訓教材\\第3章\\3-1\\E3-1-3-1.html" \l "添加新的鍵值對)

字典是動態結構，您可以隨時向字典添加新的鍵值對。 例如，要添加新的鍵值對，可以給出字典的名稱，後跟方括號中的新鍵以及新值。

|  |  |
| --- | --- |
| In[9]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image['source'] = 'Wikipedia'  image |
| Out[9] | {'color': 'greyscale',  'size': 256789,  'type': 'jpg',  'address': 'file:PIC\_0390.jpg',  'source': 'Wikipedia'} |

修改字典中的值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#修改字典中的值)

要修改字典中的值，請使用方括號中的鍵給出字典的名稱，然後使用與該鍵關聯的新值。

|  |  |
| --- | --- |
| In[10]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image['color'] = 'Black&White'  image |
| Out[10] | {'color': 'Black&White',  'size': 256789,  'type': 'jpg',  'address': 'file:PIC\_0390.jpg'} |

刪除鍵值對[¶](file:///C:\\Users\\User\\TQC+%20網頁資料擷取與分析特訓教材\\第3章\\3-1\\E3-1-3-1.html" \l "刪除鍵值對)

可以使用del語句來完全刪除鍵值對，其中填入字典的名稱和要刪除的鍵。

|  |  |
| --- | --- |
| In[11]: | del image['color']  image |
| Out[11] | {'size': 256789, 'type': 'jpg', 'address': 'file:PIC\_0390.jpg'} |

keys（），values（）和items（）方法[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#keys（），values（）和items（）方法)

有三種字典方法將返回字典的鍵，值或鍵和值的類似列表值：keys（），values（）和items（）。 這些方法返回的值不是真正的串列：它們**不能修改，也沒有**append（）**方法**。 但是這些資料型態（分別是dict\_keys，dict\_values和dict\_items）可以用於for迴圈。

|  |  |
| --- | --- |
| In[12]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  for key in image.keys():  print(key) |
| Out[12] | color  size  type  address |

|  |  |
| --- | --- |
| In[13]: | for value in image.values():  print(value) |
| Out[13] | greyscale  256789  jpg  file:PIC\_0390.jpg |

|  |  |
| --- | --- |
| In[14]: | for key, value in image.items():  print(key)  print('\t -' + value) |
| Out[14] | color  -greyscale  size  ---------------------------------------------------------------------------  TypeError Traceback (most recent call last)  <ipython-input-15-26f864fe4c7b> in <module>()  1 for key, value in image.items():  2 print(key)  ----> 3 print('\t -' + value)  TypeError: must be str, not int |

檢查字典中是否存在鍵或值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#檢查字典中是否存在鍵或值)

利用in和not in運算子可以檢查字典中是否存在某個鍵或值。

|  |  |
| --- | --- |
| In[15]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  'color' in image.keys() |
| Out[15] | True |

|  |  |
| --- | --- |
| In[16]: | 289983 in image.values() |
| Out[16] | False |

|  |  |
| --- | --- |
| In[17]: | 'compression' not in image.keys() |
| Out[17] | True |

get（）方法[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#get（）方法)

字典有一個get（）方法，它接受兩個參數：要檢索的值的鍵，如果該鍵不存在則返回的回退值。

|  |  |
| --- | --- |
| In[18]: | image = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  color\_val = image.get('color', 'unknown')  designer\_val = image.get('designer', 'unknown')  designer\_val |
| Out[18] | 'unknown' |

setdefault（）方法[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#setdefault（）方法)

setdefault（）方法 傳遞給方法的第一個參數是檢查的鍵，第二個參數是在該鍵不存在時設置在該鍵的值。 如果密鑰確實存在，setdefault（）方法返回鍵的值。

|  |  |
| --- | --- |
| In[19]: | # A simple character counter using the setdefault() method  fst\_paragraph = '''  Call me Ishmael. Some years ago—never mind how long precisely—having  little or no money in my purse, and nothing particular to interest me on  shore, I thought I would sail about a little and see the watery part of  the world. It is a way I have of driving off the spleen and regulating  the circulation. Whenever I find myself growing grim about the mouth;  whenever it is a damp, drizzly November in my soul; whenever I find  myself involuntarily pausing before coffin warehouses, and bringing up  the rear of every funeral I meet; and especially whenever my hypos get  such an upper hand of me, that it requires a strong moral principle to  prevent me from deliberately stepping into the street, and methodically  knocking people’s hats off—then, I account it high time to get to  sea as soon as I can. This is my substitute for pistol and ball. With  a philosophical flourish Cato throws himself upon his sword; I quietly  take to the ship. There is nothing surprising in this. If they but knew  it, almost all men in their degree, some time or other, cherish very  nearly the same feelings towards the ocean with me.'''  count = {}  for character in fst\_paragraph:  count.setdefault(character, 0)  count[character] += 1  print(count) |
| Out[19] | {'\n': 16, 'C': 2, 'a': 57, 'l': 45, ' ': 182, 'm': 30, 'e': 107, 'I': 12, 's': 52, 'h': 51, '.': 8, 'S': 1, 'o': 62, 'y': 22, 'r': 56, 'g': 24, '—': 3, 'n': 61, 'v': 13, 'i': 68, 'd': 21, 'w': 15, 'p': 25, 'c': 16, 't': 74, 'u': 26, ',': 10, 'b': 9, 'f': 22, 'W': 2, ';': 4, 'z': 2, 'N': 1, 'q': 2, 'k': 4, '’': 1, 'T': 2} |

嵌套[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#嵌套)

有時，您需要將一組字典存儲在串列中，或將項目串列存儲為字典中的值。 這稱為 *nesting* 。您可以在串列中嵌套一組字典，在字典中嵌套項目串列，甚至在另一個字典中嵌套字典。 嵌套是一個強大的功能，如下面的示例所示。

通常，串列對於包含有序的一系列值很有用，而字典對於將鍵與值相關聯很有用。

字典串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#字典串列)

|  |  |
| --- | --- |
| In[20]: | image\_0 = {'color': 'greyscale', 'size': 256789, 'type': 'jpg',  'address': 'file:PIC\_0390.jpg'}  image\_1 = {'color': 'greyscale', 'size': 492872, 'type': 'jpg',  'address': 'https://upload.wikimedia.org/wikipedia/commons/f/f7/Queequeg.JPG'}  image\_2 = {'color': 'greyscale', 'size': 497121, 'type': 'jpg',  'address': 'https://upload.wikimedia.org/wikipedia/commons/8/8b/Moby\_Dick\_final\_chase.jpg'}  article\_images = [image\_0, image\_1, image\_2]  article\_images |
| Out[20] | [{'color': 'greyscale',  'size': 256789,  'type': 'jpg',  'address': 'file:PIC\_0390.jpg'},  {'color': 'greyscale',  'size': 492872,  'type': 'jpg',  'address': 'https://upload.wikimedia.org/wikipedia/commons/f/f7/Queequeg.JPG'},  {'color': 'greyscale',  'size': 497121,  'type': 'jpg',  'address': 'https://upload.wikimedia.org/wikipedia/commons/8/8b/Moby\_Dick\_final\_chase.jpg'}] |

字典中的串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#字典中的串列)

而不是將字典放在串列中，有時將串列放在字典中是可行的。

|  |  |
| --- | --- |
| In[21]: | images = {'color': 'greyscale', 'size': [256789, 492872, 497121], 'type': 'jpg',  'address': ['file:PIC\_0390.jpg',  'https://upload.wikimedia.org/wikipedia/commons/f/f7/Queequeg.JPG',  'https://upload.wikimedia.org/wikipedia/commons/8/8b/Moby\_Dick\_final\_chase.jpg']  }  images['size'][-1] |
| Out[21] | 497121 |

|  |  |
| --- | --- |
| In[22]: | for key, value in images.items():  print("\n" + key.title())    if type(value) == list:  for element in value:  print("\t \* " + str(element))  else:  print("\t" + value) |
| Out[22] | Color  greyscale  Size  \* 256789  \* 492872  \* 497121  Type  jpg  Address  \* file:PIC\_0390.jpg  \* https://upload.wikimedia.org/wikipedia/commons/f/f7/Queequeg.JPG  \* https://upload.wikimedia.org/wikipedia/commons/8/8b/Moby\_Dick\_final\_chase.jpg |

字典中的字典[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#字典中的字典)

可以將字典嵌套在另一個字典中，但是執行此操作時，代碼可能會很快變得複雜。 例如，如果您有一個網站的多個用戶，每個用戶都有一個唯一的用戶名，您可以使用用戶名作為字典中的鍵。 然後，可以使用字典作為與其用戶名關聯的值來存儲有關每個用戶的信息。 在下面的清單中，我們存儲了有關每個用戶的三條信息：他們的名字，姓氏和位置。 我們將通過循環訪問用戶名和與每個用戶名關聯的信息字典來訪問此信息：

|  |  |
| --- | --- |
| In[23]: | users = {  'aeinstein': {  'first': 'albert',  'last': 'einstein',  'locations': ['princeton', 'copenhagen'],  },    'mcurie': {  'first': 'marie',  'last': 'curie',  'locations': ['paris', 'athens'],  },  }  for username, user\_info in users.items():  print("\nUsername: " + username)  full\_name = user\_info['first'] + " " + user\_info['last']  locations = user\_info['locations']    print("\tFull name: " + full\_name.title())  for location in locations:  print("\tLocation: " + location.title()) |
| Out[23] | Username: aeinstein  Full name: Albert Einstein  Location: Princeton  Location: Copenhagen  Username: mcurie  Full name: Marie Curie  Location: Paris  Location: Athens |

漂亮的印刷[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-1\E3-1-3-1.html#漂亮的印刷)

如果將pprint模塊導入到程序中，您將可以訪問pprint（）和pformat（）函數，這些函數將“漂亮打印”字典的值。 當您希望在字典中清晰地顯示項目時，這比“print（）”提供的內容更有用。

|  |  |
| --- | --- |
| In[24]: | print(users) |
| Out[24] | {'aeinstein': {'first': 'albert', 'last': 'einstein', 'locations': ['princeton', 'copenhagen']}, 'mcurie': {'first': 'marie', 'last': 'curie', 'locations': ['paris', 'athens']} |

|  |  |
| --- | --- |
| In[25]: | import pprint  pprint.pprint(users) |
| Out[25] | {'aeinstein': {'first': 'albert',  'last': 'einstein',  'locations': ['princeton', 'copenhagen']},  'mcurie': {'first': 'marie',  'last': 'curie',  'locations': ['paris', 'athens']}} |

## 4-1-4 座標軸範圍，標籤與刻度

# 4-2 各種圖表之呈現

NumPy是[Python語言](https://zh.wikipedia.org/wiki/Python)的一個擴充程式庫，支援高速的[維度](https://zh.wikipedia.org/wiki/%E5%A4%9A%E7%B6%AD)[陣列](https://zh.wikipedia.org/wiki/%E9%99%A3%E5%88%97)與[矩陣](https://zh.wikipedia.org/wiki/%E7%9F%A9%E9%99%A3)運算，此外也針對陣列運算提供大量的[數學](https://zh.wikipedia.org/wiki/%E6%95%B8%E5%AD%B8)[函式](https://zh.wikipedia.org/wiki/%E5%87%BD%E6%95%B8)[函式庫](https://zh.wikipedia.org/wiki/%E5%87%BD%E5%BC%8F%E5%BA%AB)。在NumPy上只要能被表示為針對陣列或矩陣運算的演算法，其執行效率幾乎都可以與編譯過的等效[C語言](https://zh.wikipedia.org/wiki/C%E8%AA%9E%E8%A8%80" \o "C語言)程式碼一樣快。

## 4-2-1 折線圖

NumPy的核心功能是"ndarray"(即n-dimensional array，多維陣列)資料結構。這是一個表示多維度、同質並且固定大小的陣列物件。

#### ICON2範例程式E4-2-1-1.ipynb

建立一維NumPy Arrays[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-1-1.html#建立一維NumPy-Arrays)

可以經由傳送串列(list)或元組(tuple)給np.array即可建立一維陣列.

|  |  |
| --- | --- |
| In[1]: | import numpy as np  a = np.array([1, 2, 3, 4])  b = np.array((1, 2, 3, 4))  print(a)  print(b) |
| Out[1] | [1 2 3 4]  [1 2 3 4] |

也可以透過np.arange建立一維NumPy array，其使用方式類似Python標準庫中的range，如下面實例可在3-30(包含3但不含30)間隔3建立一維陣列。

|  |  |
| --- | --- |
| In[]: | c = np.arange(3, 30, 3)  print(c) |
| Out[] | [ 3 6 9 12 15 18 21 24 27] |

np.linspace 則可建立一個間隔相同的陣列，如下面實例可在0-10間建立有5個元素的陣列，間隔相同(均為2.5)

|  |  |
| --- | --- |
| In[3]: | pi\_steps = np.linspace(0, 10, 5)  print(pi\_steps) |
| Out[3] | [ 0. 2.5 5. 7.5 10. ] |

一維陣列以索引(Index)存取元素[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-1-1.html#一維陣列以索引(Index)存取元素)

|  |  |
| --- | --- |
| In[4]: | print(c)  print(c[0]) # 取得在index位置0的元素  print(c[1]) # 取得在index位置1的元素  print(c[2:6]) # 取得在index位置2-5(包含2但不含6)的子陣列(subarray)  print(c[1:-1:2]) # 取得陣列c中index位置1,3,5,..(各間隔1個元素來選取)的子陣列 |
| Out[4] | [ 3 6 9 12 15 18 21 24 27]  3  6  [ 9 12 15 18]  [ 6 12 18 24] |

多維陣列(multi-dimensional arrays)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-1-1.html#多維陣列(multi-dimensional-arrays))

可以經由直接傳送多維串列(list)給np.array即可建立多維陣列，接著可以透過np.shape取得其外形(shape)。如下面實例中two\_dim為2X3陣列(2列3行)；three\_dim則為2X3X4陣列：

|  |  |
| --- | --- |
| In[5]: | two\_dim = np.array([[ 1, 2, 3, 4],  [ 5, 6, 7, 8],  [ 9, 10, 11, 12]])  three\_dim = np.array([[[ 1, 2, 3, 4],  [ 5, 6, 7, 8],  [ 9, 10, 11, 12]],  [[13, 14, 15, 16],  [17, 18, 19, 20],  [21, 22, 23, 24]]])  print(two\_dim.shape)  print(three\_dim.shape) |
| Out[5] | (3, 4)  (2, 3, 4) |

也可以建立一維陣列，接著利用numpy的reshape可以轉為所需要的各種外形。如下面實例中，a原為1X25陣列，可以reshape為5X5陣列；也可以直接建立陣列後呼叫reshape方法轉為3X4陣列。

|  |  |
| --- | --- |
| In[6]: | a = np.arange(1, 26).reshape(5, 5)  print(a.shape)  print(a)  print(a[2, 4])  print(a.shape)  np.arange(12).reshape(3, 4) |
| Out[6] | [[ 1 2 3 4 5]  [ 6 7 8 9 10]  [11 12 13 14 15]  [16 17 18 19 20]  [21 22 23 24 25]]  15  (5, 5)  array([[ 0, 1, 2, 3],  [ 4, 5, 6, 7],  [ 8, 9, 10, 11]]) |

多維陣列以Index存取元素[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-1-1.html#多維陣列以Index存取元素)

|  |  |
| --- | --- |
| In[7]: | a = np.arange(1, 26).reshape(5, 5)  print(a)  print(a[0, 1:4]) # 獲取由row 0(第0個橫列)中第1個-第3個元素(不含第4個)產生的一維子陣列  print(a[1:4, 0]) # 獲取由column 0(第0個直行)中第1個-第3個元素(不含第4個)產生的一維子陣列  print(a[::2,::2]) # 獲取一個二維陣列，由橫列與直行中隔個元素產生(0、2、4列與0、2、4行，也就是(0,0),(0,2),(0,4),(2,0),(2,2),(4),(4,0),(4,2),(4,4等9個元素)  print(a[:, 1]) # 獲取由整個column 1(第1個直行)中元素產生的一維子陣列 |
| Out[7] | [[ 1 2 3 4 5]  [ 6 7 8 9 10]  [11 12 13 14 15]  [16 17 18 19 20]  [21 22 23 24 25]]  [2 3 4]  [ 6 11 16]  [[ 1 3 5]  [11 13 15]  [21 23 25]]  [ 2 7 12 17 22] |

|  |  |
| --- | --- |
| In[8]: | a = np.arange(0, 60).reshape((4, 3, 5))  print("陣列a\n",a)  print("a[0, 1, 2]=",a[0, 1, 2]) # 存取a[0][1][2]  print("a[:,2]=\n",a[:,2]) # 產生由內部二維陣列中row 2(第三個橫列)組成的串列  print("a[:,:,2]=\n",a[:,:,2]) # 產生由內部二維陣列中column 2(第三個直行)組成的串列  print("a[::2,::2,::2]=\n",a[::2,::2,::2]) # 產生一個三維陣列，由橫列與直行中隔個元素產生 |
| Out[8] | 陣列a  [[[ 0 1 2 3 4]  [ 5 6 7 8 9]  [10 11 12 13 14]]  [[15 16 17 18 19]  [20 21 22 23 24]  [25 26 27 28 29]]  [[30 31 32 33 34]  [35 36 37 38 39]  [40 41 42 43 44]]  [[45 46 47 48 49]  [50 51 52 53 54]  [55 56 57 58 59]]]  a[0, 1, 2]= 7  a[:,2]=  [[10 11 12 13 14]  [25 26 27 28 29]  [40 41 42 43 44]  [55 56 57 58 59]]  a[:,:,2]=  [[ 2 7 12]  [17 22 27]  [32 37 42]  [47 52 57]]  a[::2,::2,::2]=  [[[ 0 2 4]  [10 12 14]]  [[30 32 34]  [40 42 44]]] |

#### ICON2範例程式E4-2-1-2.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import numpy as np  nf1 = np.genfromtxt('2017pig.csv', delimiter=',' ,skip\_header=1)  print("市場全年成交最高平均重量"+str(nf1[:,1].max(axis=0)))  print("市場全年成交最低平均價"+str(nf1[:,2].min(axis=0)))  print("市場全年總成交頭數"+str(nf1[:,0].sum(axis=0)))  total\_sales=(nf1[:,0]\*nf1[:,1]\*nf1[:,2])  print("市場全年總成交金額"+str(total\_sales.sum(axis=0)))  print("市場全年成交平均每頭金額"+str(total\_sales.sum(axis=0)/nf1[:,0].sum(axis=0))) |

#### ICON2範例程式說明

* 1行import numpy以進行資料分析。
* 2行運用numpy的genfromtxt方法由'2017pig.csv'(此為某市場2017年全年毛豬交易行情資料，主要欄位：total\_amt(成交頭數-總數)、average\_weight(成交頭數-平均重量、average\_price(成交頭數-平均價格)讀入資料，檔案中以","分隔，同時跳過開頭第一行(均為標題)，設定為nf1物件。
* 3行運用numpy的max方法輸出nf1的市場全年成交最高平均重量。
* 4行運用numpy的min方法輸出nf1的市場全年成交最低平均價
* 5行運用numpy的sum方法輸出nf1的市場全年總成交頭數。
* 6行計算每日成交金額，設定為total\_sales物件。
* 7行運用numpy的sum方法輸出total\_sales的市場全年總成交金額。
* 8行計算並列印市場全年成交平均每頭金額。

#### ICON2輸出結果

市場全年成交最高平均重量148.18

市場全年成交最低平均價66.35

市場全年總成交頭數3383747.0

市場全年總成交金額32115493123.458595

市場全年成交平均每頭金額9491.103538018237

#### ICON2範例程式E4-2-1-3.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import numpy as np  data = np.genfromtxt('president\_heights.csv', delimiter=',',skip\_header=1 )  heights = np.array(data[:,2])  heights = np.array(data['height(cm)'])  print(heights)  print("Mean height: ", heights.mean())  print("Standard deviation:", heights.std())  print("Minimum height: ", heights.min())  print("Maximum height: ", heights.max())  print("25th percentile: ", np.percentile(heights, 25))  print("Median: ", np.median(heights))  print("75th percentile: ", np.percentile(heights, 75)) |

#### ICON2範例程式說明

* 1行import numpy以進行資料分析。
* 2行運用numpy的genfromtxt方法由'president\_heights.csv' (此為美國歷任總統身高資料，主要欄位：order(排序)、name(姓名)、height(cm) (身高)讀入資料，檔案中以","分隔，同時跳過開頭第一行(均為標題)，設定為data物件。
* 3-4行將data物件的第三個欄位height(cm)另存為heights陣列，並列印輸出。
* 5行運用numpy的mean 方法輸出美國總統平均身高。
* 6行運用numpy的mean 方法輸出美國總統平均身高。
* 7行運用numpy的std 方法輸出美國總統身高標準差。
* 8行運用numpy的min 方法輸出美國總統最矮身高。
* 9行運用numpy的max方法輸出美國總統最高身高。
* 10行運用numpy的percentile方法輸出美國總統身高第一個四分位數。
* 11行運用numpy的median方法輸出美國總統身高中位數。
* 12行運用numpy的percentile方法輸出美國總統身高第三個四分位數。

#### ICON2輸出結果

[189. 170. 189. 163. 183. 171. 185. 168. 173. 183. 173. 173. 175. 178.

183. 193. 178. 173. 174. 183. 183. 168. 170. 178. 182. 180. 183. 178.

182. 188. 175. 179. 183. 193. 182. 183. 177. 185. 188. 188. 182. 185.

190.]

Mean height: 179.97674418604652

Standard deviation: 7.023178807524852

Minimum height: 163.0

Maximum height: 193.0

25th percentile: 174.5

Median: 182.0

75th percentile: 184.0

## 4-2-2 散佈圖

#### ICON2範例程式E4-2-2-1.ipynb

Numpy的聚合操作(Aggregation operations)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-2-1.html#Numpy的聚合操作(Aggregation-operations))

通常在面對大量資料時，要先計算相關資料的摘要統計資訊，NumPy具有內置的聚合操作功能，可用於處理陣列，本範例將展示這些功能。

加總陣列中的資料值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-2-1.html#加總陣列中的資料值)

進行加總，可以利用Python內建的 sum 功能:

|  |  |
| --- | --- |
| In[1]: | import numpy as np  L = np.random.random(10)  print(L)  print(sum(L)) |
| Out[1] | [0.27519442 0.21383682 0.37824873 0.21087795 0.8437888 0.12147178  0.7981921 0.26025743 0.91075127 0.03418838]  4.046807691021659 |

在NumPy中np.sum語法與Python內建功能類似，也可以得到相同結果:

|  |  |
| --- | --- |
| In[]: | np.sum(L) |
| Out[] | 4.046807691021659 |

但是因為利用編譯後程式碼執行，NumPy版本遠比Python版本快:

|  |  |
| --- | --- |
| In[3]: | big\_array = np.random.rand(1000000)  print("Python內建版本：")  %timeit sum(big\_array)  print("Numpy版本：")  %timeit np.sum(big\_array) |
| Out[3] | Python內建版本：  133 ms ± 1.11 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)  Numpy版本：  939 µs ± 5.05 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each) |

最小值與最大值[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-2-1.html#最小值與最大值)

同樣的， Python也有內建的min 與 max 功能，可用來找尋所給陣列中的最小值與最大值:

|  |  |
| --- | --- |
| In[4]: | min(big\_array), max(big\_array) |
| Out[4] | (5.678456824753653e-07, 0.9999999024603322) |

在NumPy中也有np.min與np.max，也可以得到相同結果，但是速度截然不同

|  |  |
| --- | --- |
| In[5]: | np.min(big\_array), np.max(big\_array) |
| Out[5] | (5.678456824753653e-07, 0.9999999024603322) |

|  |  |
| --- | --- |
| In[6]: | print("Python內建版本：")  %timeit min(big\_array)  print("Numpy版本：")  %timeit np.min(big\_array) |
| Out[6] | Python內建版本：  54.2 ms ± 356 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)  Numpy版本：  438 µs ± 21.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each) |

對於min，max，sum和其他幾個NumPy聚合功能，更短的寫法是使用陣列物件本身的方法：

|  |  |
| --- | --- |
| In[7]: | print(big\_array.min(), big\_array.max(), big\_array.sum()) |
| Out[7] | 5.678456824753653e-07 0.9999999024603322 500441.63962966343 |

多維陣列聚合操作[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-2-1.html#多維陣列聚合操作)

假設您有一些儲存存在二維陣列中的資料，預設是對整個陣列進行聚合操作，但也可以進行聚合操作是沿行或列聚合：

|  |  |
| --- | --- |
| In[8]: | M = np.random.random((4, 4))  print(M)  print(M.sum()) |
| Out[8] | [[0.2161298 0.24452895 0.9039042 0.06511674]  [0.95245665 0.96426864 0.66881504 0.32197578]  [0.43814338 0.03070089 0.30242855 0.33516801]  [0.45101403 0.73766608 0.16497214 0.2081843 ]]  7.005473168787838 |

聚合函數採用另一個參數來指定計算聚合的*軸*。 例如，我們可以通過指定axis = 0找到每列中的最小值，指定axis = 1找到每行中的最小值：

|  |  |
| --- | --- |
| In[9]: | print(M.min(axis=0))  print(M.min(axis=1)) |
| Out[9] | [0.2161298 0.03070089 0.16497214 0.06511674]  [0.06511674 0.32197578 0.03070089 0.16497214] |

max與sum也可以進行類似的操作。

|  |  |
| --- | --- |
| In[10]: | print(M.max(axis=0))  print(M.max(axis=1))  print(M.sum(axis=0))  print(M.sum(axis=1)) |
| Out[10] | [0.95245665 0.96426864 0.9039042 0.33516801]  [0.9039042 0.96426864 0.43814338 0.73766608]  [2.05774385 1.97716456 2.04011993 0.93044482]  [1.42967968 2.90751611 1.10644083 1.56183655] |

在此處指定軸的方式中，axis關鍵字指定將折疊*的陣列的*維度，而不是將返回的維度。因此，指定axis = 0意味著第一個軸將被折疊：對於二維陣列，這意味著將聚合每列中的值。

其他聚合操作[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-2-1.html#其他聚合操作)

NumPy提供了許多其他聚合函數，大多數聚合都有一個“NaN”安全對應物來計算結果，同時忽略缺失值，缺失值由特殊的IEEE浮點“NaN”值標記。 下表提供了NumPy中可用的有用聚合函數的列表：

| 功能名稱 | NaN安全版 | 說明 |
| --- | --- | --- |
| np.sum | np.nansum | 計算元素之和 |
| np.prod | np.nanprod | 計算元素的乘積 |
| np.mean | np.nanmean | 計算元素的中位數 |
| np.std | np.nanstd | 計算標準差 |
| np.var | np.nanvar | 計算方差 |
| np.min | np.nanmin | 找到最小值 |
| np.max | np.nanmax | 找到最大值 |
| np.argmin | np.nanargmin | 查找最小值索引 |
| np.argmax | np.nanargmax | 查找最大值的索引 |
| np.median | np.nanmedian | 計算元素的中位數 |
| np.percentile | np.nanpercentile | 計算元素的基於排名的統計數據 |
| np.any | N / A | 評估是否有任何元素是真的 |
| np.all | N / A | 評估所有元素是否都為真 |

#### ICON2範例程式E4-2-2-2.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import numpy as np  data = [37, 24, 6, 51, 83, 28, 51, 58, 82, 95,  8, 43, 86, 78, 71, 82, 58, 10, 15, 56,  4, 75, 6, 95, 23, 79, 90, 35, 72, 25,  50, 29, 44, 67, 67, 61, 40, 44, 13, 59,  60, 67, 93, 69, 71, 8, 76, 81, 17, 72,  83, 6, 42, 53, 98, 6, 90, 4, 59, 87,  28, 17, 28, 46, 40, 53, 70, 49, 55, 41,  74, 57, 31, 55, 5, 65, 44, 98, 36, 4]  data = np.array(data)  print('資料型態：%s' % type(data))  print('平均值：%.2f' % np.mean(data))  print('中位數：%.2f' % np.median(data))  print('標準差：%.2f' % np.std(data))  print('變異數：%.2f' % np.var(data))  print('極差值：%.2f' % np.ptp(data)) |

#### ICON2範例程式說明

* 1行import所需套件。
* 2-3行運用numpy輸入資料建立陣列物件data。
* 4行運用numpy的type方法列印data物件的資料型態。
* 5行運用numpy的mean方法列印data物件的平均值。
* 6行運用numpy的median方法列印data物件的中位數。
* 7行運用numpy的std方法列印data物件的標準差。
* 8行運用numpy的var方法列印data物件的變異數。
* 9行運用numpy的ptp(方法列印data物件的極差值。

#### ICON2輸出結果

資料型態：<class 'numpy.ndarray'>

平均值：50.48

中位數：53.00

標準差：27.57

變異數：760.27

極差值：94.00

1008 興華公園 12 20180704221341 28

…略

#### 範例程式E4-2-2-3.py



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import numpy as np  matrix1 = np.random.randint(1, 51, 12).reshape(3, 4)  matrix2 = np.random.randint(1, 51, 20).reshape(4, 5)  print('matrix1: \n%s' % matrix1)  print('\nmatrix2: \n%s' % matrix2)  print('\nmatrix1每一列的最大值：%s' % np.amax(matrix1, axis=1))  print("\nmatrix1第2列小於30的個數：%d" % np.sum(matrix1[2, :] < 30))  print('\nmatrix2每一欄的最大值：%s' % np.amax(matrix2, axis=0))  print("\nmatrix2第2欄小於30的個數：%d" % np.sum(matrix2[:, 2] < 30))  print('\nmatrix1第一列和matrix2第一列的聯集結果：%s' % np.union1d(matrix1[0,:], matrix2[0,:]))  print('\nmatrix1 \* matrix2: \n%s' % np.dot(matrix1, matrix2)) |

#### ICON2範例程式說明

* 1行import所需套件。
* 2行運用numpy的random.randint方法取得從1-51的12個隨機數，reshape為3\*4陣列matrix1。
* 3行運用numpy的random.randint方法取得從1-51的20個隨機數，reshape為4\*5陣列matrix2。
* 4-5行#顯示matrix1與matrix2內容。
* 6行顯示matrix1每一列的最大值。
* 7行輸出matrix1第2列小於30的個數。
* 8行顯示matrix2每一欄的最大值。
* 9行輸出matrix2第2欄小於30的個數。
* 10行輸出matrix1第一列和matrix2第一列的聯集結果。
* 11行輸出matrix1\*matrix2的結果。

#### ICON2輸出結果

matrix1:

[[18 36 38 2]

[29 50 29 44]

[17 33 12 4]]

matrix2:

[[24 37 20 28 6]

[ 7 50 2 11 9]

[15 4 21 20 41]

[41 26 29 39 21]]

matrix1每一列的最大值：[38 50 33]

matrix1第2列小於30的個數：3

matrix2每一欄的最大值：[41 50 29 39 41]

matrix2第2欄小於30的個數：4

matrix1第一列和matrix2第一列的聯集結果：[ 2 6 18 20 24 28 36 37 38]

matrix1 \* matrix2:

[[1336 2670 1288 1738 2032]

[3285 4833 2565 3658 2737]

[ 983 2431 774 1235 975]]

## 4-2-3 長條圖

#### ICON2範例程式E4-2-3-1.ipynb

陣列排序(Sorting Arrays)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-3-1.html#陣列排序(Sorting-Arrays))

本範例介紹與NumPy陣列中的值排序相關的演算法，例如，一個簡單的選擇排序重複從列表中查找最小值，並進行交換直到列表排序。 我們可以在幾行Python中完成程式編寫：

|  |  |
| --- | --- |
| In[1]: | import numpy as np  def selection\_sort(x):  for i in range(len(x)):  swap = i + np.argmin(x[i:])  (x[i], x[swap]) = (x[swap], x[i])  return x  x = np.array([2, 1, 4, 6, 3, 5])  selection\_sort(x) |
| Out[1] | array([1, 2, 3, 4, 5, 6]) |

在NumPy中快速排序方式：np.sort和np.argsort[¶](file:///C:\\Users\\User\\TQC+%20網頁資料擷取與分析特訓教材\\第3章\\3-2\\E3-2-3-1.html" \l "在NumPy中快速排序方式：np.sort和np.argsort)

儘管Python內建了sort和sorted函數來處理串列，但效率較差。預設情況下，np.sort使用 *quicksort* 算法，但 *mergesort* 和 *heapsort* 也可用。 要在不修改輸入的情況下返回陣列的排序版本，可以使用np.sort：

|  |  |
| --- | --- |
| In[]: | x = np.array([2, 1, 4, 6, 3, 5])  np.sort(x) |
| Out[] | array([1, 2, 3, 4, 5, 6]) |

如果您希望就地對陣列進行排序，則可以使用陣列的sort方法：

|  |  |
| --- | --- |
| In[3]: | x.sort()  print(x) |
| Out[3] | [1 2 3 4 5 6] |

相關的函數是argsort，它返回已排序元素的 *indices* ：

|  |  |
| --- | --- |
| In[4]: | x = np.array([2, 1, 4, 6, 3, 5])  i = np.argsort(x)  print(i) |
| Out[4] | [1 0 4 2 5 3] |

此結果的第一個元素給出最小元素的索引，第二個值給出第二個最小元素的索引，依此類推。如果需要，可以通過花式索引使用這些索引建構排序陣列：

|  |  |
| --- | --- |
| In[5]: | x[i] |
| Out[5] | array([1, 2, 3, 4, 5, 6]) |

經由列或行排序[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-3-1.html#經由列或行排序)

NumPy排序演算法的一個有用特性是能夠使用axis參數對多維陣列的特定行或列進行排序。 例如：

|  |  |
| --- | --- |
| In[6]: | rand = np.random.RandomState(42)  X = rand.randint(0, 10, (4, 6))  print(X) |
| Out[6] | [[6 3 7 4 6 9]  [2 6 7 4 3 7]  [7 2 5 4 1 7]  [5 1 4 0 9 5]] |

|  |  |
| --- | --- |
| In[7]: | # sort each column of X  np.sort(X, axis=0) |
| Out[7] | array([[2, 1, 4, 0, 1, 5],  [5, 2, 5, 4, 3, 7],  [6, 3, 7, 4, 6, 7],  [7, 6, 7, 4, 9, 9]] |

|  |  |
| --- | --- |
| In[8]: | # sort each row of X  np.sort(X, axis=1) |
| Out[8] | array([[3, 4, 6, 6, 7, 9],  [2, 3, 4, 6, 7, 7],  [1, 2, 4, 5, 7, 7],  [0, 1, 4, 5, 5, 9]]) |

部分排序：分區[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-2\E3-2-3-1.html#部分排序：分區)

有時我們對排序整個陣列不感興趣，但只想在陣列中找到 *k* 最小值。 NumPy在np.partition函數中提供了這個功能。 np.partition取一個陣列和一個數字 *K* ; 結果是一個新陣列，在分區左邊有最小的 *K* 值，剩下的值按任意順序在右邊：

|  |  |
| --- | --- |
| In[9]: | x = np.array([7, 2, 3, 1, 6, 5, 4])  np.partition(x, 4) |
| Out[9] | array([2, 1, 3, 4, 5, 6, 7]) |

請注意，結果陣列中的前四個值是陣列中的最小四個值，其餘陣列位置包含其餘值。 在這兩個分區中，元素具有任意順序。與排序類似，我們可以沿多維陣列的任意軸進行分區：

|  |  |
| --- | --- |
| In[10]: | np.partition(X, 2, axis=1) |
| Out[10] | array([[3, 4, 6, 7, 6, 9],  [2, 3, 4, 7, 6, 7],  [1, 2, 4, 5, 7, 7],  [0, 1, 4, 5, 9, 5]]) |

結果是一個陣列，其中每列中的前兩個元素為該列中的最小值，其餘值填充剩餘的元素。

#### 範例程式E4-2-3-2.py



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import numpy as np  arr = np.array([[ 3, 2],[ 1, 6],[ 12, 11],[ 10, 9],[ 4, 8],[ 5, 7]])  print(arr)  print(arr.shape)  print(np.sort(arr))  print(np.sort(arr,axis=-1))  print(np.sort(arr,axis=0))  arr1=arr.reshape(3,2,2)  print(np.sort(arr1,axis=0))  print(np.sort(arr1,axis=1))  print(np.sort(arr1,axis=-1)) |

#### ICON2範例程式說明

* 1行匯入 numpy 套件。
* 2行輸入資料，讀入成為numpy的陣列arr。
* 3行列印輸出陣列arr。
* 4行列印輸出陣列arr的shape，arr陣列為6\*2陣列。
* 5行對arr進行排序(未指定則預設對最後一個軸)並輸出。
* 6行對arr進行排序(指定對最後一個軸，因此與5行結果相同)並輸出。
* 7行對arr進行排序(指定對第一個軸)並輸出。
* 8行將arr調整成3維陣列3\*2\*2，設定為arr1物件。
* 9行對arr1進行排序(指定對第一個軸)並輸出。
* 10行對arr1進行排序(指定對第二個軸)並輸出。
* 11行對arr1進行排序(指定對最後一個軸)並輸出。

#### ICON2輸出結果

[[ 3 2]

[ 1 6]

[12 11]

[10 9]

[ 4 8]

[ 5 7]]

(6, 2)

[[ 2 3]

[ 1 6]

[11 12]

[ 9 10]

[ 4 8]

[ 5 7]]

[[ 2 3]

[ 1 6]

[11 12]

[ 9 10]

[ 4 8]

[ 5 7]]

[[ 1 2]

[ 3 6]

[ 4 7]

[ 5 8]

[10 9]

[12 11]]

[[[ 3 2]

[ 1 6]]

[[ 4 8]

[ 5 7]]

[[12 11]

[10 9]]]

[[[ 1 2]

[ 3 6]]

[[10 9]

[12 11]]

[[ 4 7]

[ 5 8]]]

[[[ 2 3]

[ 1 6]]

[[11 12]

[ 9 10]]

[[ 4 8]

[ 5 7]]]

#### ICON2範例程式E4-2-3-3.py

|  |  |
| --- | --- |
| 1  2  3  4  5 | import numpy as np  data = [(2,'c',85.4),(3,'java',90),(4,'php',88)]  arr2 = np.array(data,dtype =[('no',int),('name','S10'),('score',float)])  print(np.sort(arr2,order='score'))  print(np.sort(arr2,order = ['no','score'])) |

#### ICON2範例程式說明

* 1行匯入 numpy 套件。
* 2行輸入資料，設定為data物件。
* 3行將data讀入成為numpy的陣列，並指定各欄位名稱與資料型態。
* 4行指定'score'欄位進行遞增排序後輸出。
* 5行指定'no','score'欄位進行多列组合排序排序後輸出。

#### ICON2輸出結果

[(2, b'c', 85.4) (4, b'php', 88. ) (3, b'java', 90. )]

[(2, b'c', 85.4) (3, b'java', 90. ) (4, b'php', 88. )]

## 4-2-4 直方圖

## 4-2-5 圓形圖

## 4-2-6 柱狀圖

# 4-3 多圖表繪製

## 4-3-1 pandas的物件

通過標準庫中的 json 模組，使用函數 dumps()與loads() 進行 json 資料基本讀寫。json.dumps() 是將Python中的五件序列化為 json 格式的 str，而 json.loads() 是反向操作，將已編碼的 JSON 字串解碼為 Python 物件。

Encoding basic Python object hierarchies:

#### ICON2範例程式E3-4-1-1.ipynb

Pandas 物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Pandas-物件)

在最基本的層面上，Pandas物件可以被認為是NumPy結構化陣列的增強版本，其中行和列用標籤而不是簡單的整數索引來標識。由此Pandas在基本資料結構之上提供了許多有用的工具，方法和功能，這三個基本的Pandas資料結構是：Series，DataFrame和Index。

先導入NumPy和Pandas：

|  |  |
| --- | --- |
| In[1]: | import numpy as np  import pandas as pd |

Pandas Series物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Pandas-Series物件)

PandasSeries是索引資料的一維陣列，可以從串列或陣列創建，如下所示：

|  |  |
| --- | --- |
| In[2]: | data = pd.Series([0.25, 0.5, 0.75, 1.0])  data |
| Out[2] | 0 0.25  1 0.50  2 0.75  3 1.00  dtype: float64 |

如下面輸出，Series包含了一系列值和一系列索引，我們可以使用values和index屬性來訪問。values就只是一個NumPy陣列，index則是類型為pd.Index的類似陣列的物件：

|  |  |
| --- | --- |
| In[3]: | data.values |
| Out[3] | array([0.25, 0.5 , 0.75, 1. ]) |

|  |  |
| --- | --- |
| In[4]: | data.index |
| Out[4] | RangeIndex(start=0, stop=4, step=1) |

與NumPy陣列一樣，相關索引可以通過熟悉的Python方括號表示法訪問資料：

|  |  |
| --- | --- |
| In[5]: | data[1] |
| Out[5] | 0.5 |

|  |  |
| --- | --- |
| In[6]: | data[1:3] |
| Out[6] | 1 0.50  2 0.75  dtype: float64 |

PandasSeries比一維NumPy陣列更加通用和靈活。

Series 可被看一般化的NumPy 陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Series-可被看一般化的NumPy-陣列)

可能看起來Series物件可以與一維NumPy陣列互換，其本質區別在於索引的存在：Numpy陣列有一個*隱式定義的*整數索引用於訪問值時，PandasSeries有一個*顯式定義的*索引與值相關聯。這個顯式索引定義為Series物件提供了額外的功能。 例如，索引不必是整數，但可以包含任何所需類型的值。甚至如果我們願意，我們可以使用字符串作為索引：

|  |  |
| --- | --- |
| In[7]: | data = pd.Series([0.25, 0.5, 0.75, 1.0],  index=['a', 'b', 'c', 'd'])  data |
| Out[7] | a 0.25  b 0.50  c 0.75  d 1.00  dtype: float64 |

項目訪問按預期進行：

|  |  |
| --- | --- |
| In[8]: | data['b'] |
| Out[8] | 0.5 |

我們甚至可以使用非連續或無順序的索引：

|  |  |
| --- | --- |
| In[9]: | data = pd.Series([0.25, 0.5, 0.75, 1.0],  index=[2, 5, 3, 7])  data |
| Out[9] | 2 0.25  5 0.50  3 0.75  7 1.00  dtype: float64 |

|  |  |
| --- | --- |
| In[10]: | data[5] |
| Out[10] | 0.5 |

Series 可以當作特殊的字典[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Series-可以當作特殊的字典)

通過這種方式，可以將PandasSeries`視為Python字典的特化。字典是將任意鍵映射到一組任意值的結構，而Series是將鍵入的鍵映射到一組鍵入值的結構。這種類型很重要：正如NumPy陣列後面的特定於類型的編譯代碼使其比某些操作的Python串列更有效，PandasSeries的類型信息使得它比Python字典更有效操作。 通過直接從Python字典構造一個Series物件，可以使Series``-as-dictionary類比更加清晰：

|  |  |
| --- | --- |
| In[11]: | population\_dict = {'California': 39250017,  'Texas': 27862596,  'Florida': 20612439,  'New York': 19745289,  'Illinois': 12801539}  population = pd.Series(population\_dict)  population |
| Out[11] | California 39250017  Texas 27862596  Florida 20612439  New York 19745289  Illinois 12801539  dtype: int64 |

預設情況下，將創建一個Series，其中索引是從排序鍵中提取的。從這裡，可以執行典型的字典式項目訪問：

|  |  |
| --- | --- |
| In[12]: | population['California'] |
| Out[12] | 39250017 |

但是與字典不同，Series也支持陣列樣式的操作，例如切片：

|  |  |
| --- | --- |
| In[13]: | population['California':'Illinois'] |
| Out[13] | California 39250017  Texas 27862596  Florida 20612439  New York 19745289  Illinois 12801539  dtype: int64 |

建構Series物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#建構Series物件)

我們已經看到了從頭開始構建PandasSeries的幾種方法; 所有這些都是以下版本的某些版本：

pd.Series（data, index=index） 其中index是一個可選參數，data可以是許多實體之一。例如，data可以是串列或NumPy陣列，在這種情況下index預設為整數序列：

|  |  |
| --- | --- |
| In[14]: | pd.Series([2, 4, 6]) |
| Out[14] | 0 2  1 4  2 6  dtype: int64 |

data可以是純量，或是重複填入特定的索引；data也可以是一個字典，其中index預設為排序的字典鍵；不館在每種情況下，如果會有不同的結果，最好以顯式設置索引：

|  |  |
| --- | --- |
| In[15]: | pd.Series(5, index=[100, 200, 300]) |
| Out[15] | 100 5  200 5  300 5  dtype: int64 |

|  |  |
| --- | --- |
| In[16]: | pd.Series({2:'a', 1:'b', 3:'c'}) |
| Out[16] | 2 a  1 b  3 c  dtype: object |

|  |  |
| --- | --- |
| In[17]: | pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2]) |
| Out[17] | 3 c  2 a  dtype: object |

Pandas DataFrame物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Pandas-DataFrame物件)

Pandas的DataFrame被認為是NumPy陣列的一般化，也可以被認為是Python字典的特化。

DataFrame作為一般化的NumPy陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#DataFrame作為一般化的NumPy陣列)

DataFrame是具有靈活劣索引和靈活行名的二維陣列的模擬，可以將“DataFrame”視為一系列對齊的Series物件。在這裡，“對齊”是指它們共享相同的索引。下面首先構建一個新的Series，列出上一節討論的五個狀態中每個狀態的區域：

|  |  |
| --- | --- |
| In[18]: | area\_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,  'Florida': 170312, 'Illinois': 149995}  area = pd.Series(area\_dict)  area |
| Out[18] | California 423967  Texas 695662  New York 141297  Florida 170312  Illinois 149995  dtype: int64 |

現在我們已經將它與之前的populationSeries一起使用，我們可以使用字典來建構包含這些信息的單個二維物件：

|  |  |
| --- | --- |
| In[19]: | states = pd.DataFrame({'population': population,  'area': area})  states |
| Out[19] | |  | population | area | | --- | --- | --- | | California | 39250017 | 423967 | | Florida | 20612439 | 170312 | | Illinois | 12801539 | 149995 | | New York | 19745289 | 141297 | | Texas | 27862596 | 695662 | |

DataFrame有一個index屬性，可以訪問索引標籤，還有一個columns屬性，它是一個包含行標籤的Index物件：：

|  |  |
| --- | --- |
| In[20]: | states.index |
| Out[20] | Index(['California', 'Florida', 'Illinois', 'New York', 'Texas'], dtype='object') |

|  |  |
| --- | --- |
| In[21]: | states.columns |
| Out[21] | Index(['population', 'area'], dtype='object') |

因此，DataFrame可以被認為是二維NumPy陣列的一般化，其中行和列都具有用於訪問資料的通用索引。

DataFrame作為特殊的字典[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#DataFrame作為特殊的字典)

同樣，我們也可以將DataFrame視為字典的特化。當字典將鍵映射到值時，“DataFrame”將行名稱映射到行資料的“Series”。例如，要求'area'屬性返回包含我們之前看到的面積的Series物件：

|  |  |
| --- | --- |
| In[22]: | states['area'] |
| Out[22] | California 423967  Florida 170312  Illinois 149995  New York 141297  Texas 695662  Name: area, dtype: int64 |

注意這裡潛在的混淆點：在一個二維NumPy陣列中，data [0]將返回第一個*列*。 對於DataFrame，data ['col0']將返回第一個*行*。因此，最好將“DataFrame”視為通用字典而不是通用陣列，儘管兩種方式都可以看到這種情況。

建構DataFrame物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#建構DataFrame物件)

PandasDataFrame可以通過多種方式建構。這裡我們舉幾個例子。

來自單個Series物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#來自單個Series物件)

DataFrame是Series物件的集合，單列DataFrame可以從單個Series建構：

|  |  |
| --- | --- |
| In[23]: | pd.DataFrame(population, columns=['population']) |
| Out[23] | |  | population | | --- | --- | | California | 39250017 | | Texas | 27862596 | | Florida | 20612439 | | New York | 19745289 | | Illinois | 12801539 | |

從一個字典串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#從一個字典串列)

任何字典串列都可以製作成“DataFrame”。我們將使用簡單的串列推導來創建一些資料：

|  |  |
| --- | --- |
| In[24]: | data = [{'a': i, 'b': 2 \* i}  for i in range(3)]  pd.DataFrame(data) |
| Out[24] | |  | a | b | | --- | --- | --- | | 0 | 0 | 0 | | 1 | 1 | 2 | | 2 | 2 | 4 | |

即使字典中的某些鍵丟失，Pandas也會用“NaN`”（即“非數字”）值填充它們：

|  |  |
| --- | --- |
| In[25]: | pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}]) |
| Out[25] | |  | a | b | c | | --- | --- | --- | --- | | 0 | 1.0 | 2 | NaN | | 1 | NaN | 3 | 4.0 | |

從Series物件的字典[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#從Series物件的字典)

正如我們之前看到的那樣，DataFrame也可以從Series物件的字典建構：

|  |  |
| --- | --- |
| In[26]: | pd.DataFrame({'population': population,  'area': area}) |
| Out[26] | |  | population | area | | --- | --- | --- | | California | 39250017 | 423967 | | Florida | 20612439 | 170312 | | Illinois | 12801539 | 149995 | | New York | 19745289 | 141297 | | Texas | 27862596 | 695662 | |

來自二維NumPy陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#來自二維NumPy陣列)

給定一個二維資料陣列，我們可以創建一個帶有任何指定行和索引名稱的DataFrame。 如果省略，將為每個使用整數索引：

|  |  |
| --- | --- |
| In[27]: | pd.DataFrame(np.random.rand(3, 2),  columns=['foo', 'bar'],  index=['a', 'b', 'c']) |
| Out[27] | |  | foo | bar | | --- | --- | --- | | a | 0.511105 | 0.024971 | | b | 0.106175 | 0.401511 | | c | 0.488746 | 0.884107 | |

來自NumPy結構化陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#來自NumPy結構化陣列)

PandasDataFrame的運行方式與結構化陣列非常相似，可以直接從一個陣列創建：

|  |  |
| --- | --- |
| In[28]: | A = np.zeros(3, dtype=[('A', 'i8'), ('B', 'f8')])  A |
| Out[28] | array([(0, 0.), (0, 0.), (0, 0.)], dtype=[('A', '<i8'), ('B', '<f8')]) |

|  |  |
| --- | --- |
| In[29]: | pd.DataFrame(A) |
| Out[29] | |  | A | B | | --- | --- | --- | | 0 | 0 | 0.0 | | 1 | 0 | 0.0 | | 2 | 0 | 0.0 | |

Pandas Index物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#Pandas-Index物件)

我們在這裡看到Series和DataFrame物件都包含一個顯式的 *index* ，它允許你引用和修改資料。 這個Index物件可以被認為是*不可變數組*或*有序集*，舉個簡單的例子，讓我們從整數串列建構一個Index：

|  |  |
| --- | --- |
| In[30]: | ind = pd.Index([2, 3, 5, 7, 11])  ind |
| Out[30] | Int64Index([2, 3, 5, 7, 11], dtype='int64') |

索引為不可變陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-1-1.html#索引為不可變陣列)

Index在很多方面都像陣列一樣運作。例如，我們可以使用標準的Python索引表示法來檢索值或切片：

|  |  |
| --- | --- |
| In[31]: | ind[1] |
| Out[31] | 3 |

|  |  |
| --- | --- |
| In[32]: | ind[::2] |
| Out[32] | Int64Index([2, 5, 11], dtype='int64') |

Index物件也有許多NumPy陣列中熟悉的屬性：

|  |  |
| --- | --- |
| In[33]: | print(ind.size, ind.shape, ind.ndim, ind.dtype) |
| Out[33] | 5 (5,) 1 int64 |

Index物件和NumPy陣列之間的一個區別是索引是不可變的 - 也就是說，它們不能通過常規方式修改：

|  |  |
| --- | --- |
| In[34]: | ind[1] = 0 |
| Out[34] | ---------------------------------------------------------------------------  TypeError Traceback (most recent call last)  <ipython-input-35-906a9fa1424c> in <module>()  ----> 1 ind[1] = 0  ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in \_\_setitem\_\_(self, key, value)  2048  2049 def \_\_setitem\_\_(self, key, value):  -> 2050 raise TypeError("Index does not support mutable operations")  2051  2052 def \_\_getitem\_\_(self, key):  TypeError: Index does not support mutable operations |

這使得不變性可以更安全地共享多個DataFrames和陣列之間的索引，而不用於從無意索引修改副作用的可能性。

索引作為有序集[¶](file:///C:\\Users\\User\\TQC+%20網頁資料擷取與分析特訓教材\\第3章\\3-3\\E3-3-1-1.html" \l "索引作為有序集)

Pandas物件旨在促進跨資料集的連接等操作，這取決於集合算術的許多方面。Index物件遵循Python內置的set資料結構使用的許多約定，因此可以用熟悉的方式計算聯合，交集，差異和其他組合：

|  |  |
| --- | --- |
| In[35]: | indA = pd.Index([1, 3, 5, 7, 9])  indB = pd.Index([2, 3, 5, 7, 11])  indA & indB # intersection |
| Out[35] | Int64Index([3, 5, 7], dtype='int64') |

|  |  |
| --- | --- |
| In[36]: | indA | indB # union |
| Out[36] | Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64') |

|  |  |
| --- | --- |
| In[37]: | indA ^ indB # symmetric difference |
| Out[37] | Int64Index([1, 2, 9, 11], dtype='int64') |

## 4-3-2 pandas的索引與資料選取

#### ICON2範例程式E3-4-2-1.ipynb

資料索引(Indexing)與選取(Selection)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#資料索引(Indexing)與選取(Selection))

在NumPy陣列中訪問，設置和修改值的方法和工具包括索引（例如，arr [2,1]），切片（例如，arr [：，1：5]），遮罩（例如，arr [arr> 0] ），花式索引（例如，arr [0，[1,5]]）及其組合（例如，arr [：，[1,5]]）。在PandasSeries和DataFrame物件中訪問和修改值有類似方法。

Series中資料的選取[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#Series中資料的選取)

Series物件可用於一維NumPy陣列與標準的Python字典。記住這兩個重疊的類比，可幫助理解這些陣列中資料索引和選擇的模式。

將 Series當作dict[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#將-Series當作dict)

像字典一樣，Seriesj物件提供從一組鍵到一組值的映射：

|  |  |
| --- | --- |
| In[]: | import pandas as pd  data = pd.Series([0.25, 0.5, 0.75, 1.0],  index=['a', 'b', 'c', 'd'])  data |
| Out[] | a 0.25  b 0.50  c 0.75  d 1.00  dtype: float64 |

|  |  |
| --- | --- |
| In[]: | data['b'] |
| Out[] | 0.5 |

我們還可以使用類似字典的Python表示式和方法來檢查鍵/索引和值：

|  |  |
| --- | --- |
| In[]: | 'a' in data |
| Out[] | True |

|  |  |
| --- | --- |
| In[]: | data.keys() |
| Out[] | Index(['a', 'b', 'c', 'd'], dtype='object') |

|  |  |
| --- | --- |
| In[]: | list(data.items()) |
| Out[] | [('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)] |

Series物件甚至可以用類似字典的語法修改。如同通過分配新鍵來擴展字典一樣，可以通過分配新的索引值來擴展Series：

|  |  |
| --- | --- |
| In[]: | data['e'] = 1.25  data |
| Out[] | a 0.25  b 0.50  c 0.75  d 1.00  e 1.25  dtype: float64 |

物件的容易可變性是一個方便特性：在物件內，Pandas正在決定可能需要進行的記憶體佈局和資料複製; 用戶通常不需要擔心這些問題。

將Series當作一維陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#將Series當作一維陣列)

一個Series建立在這個類似字典的界面上，並通過與NumPy陣列相同的基本機制提供陣列樣式的項目選擇 - 即 *slices* ， *masking* 和 *fancy indexing* 。這些例子如下：

|  |  |
| --- | --- |
| In[]: | # slicing by explicit index  data['a':'c'] |
| Out[] | a 0.25  b 0.50  c 0.75  dtype: float64 |

|  |  |
| --- | --- |
| In[]: | # slicing by implicit integer index  data[0:2] |
| Out[] | a 0.25  b 0.50  dtype: float64 |

|  |  |
| --- | --- |
| In[]: | # masking  data[(data > 0.3) & (data < 0.8)] |
| Out[] | b 0.50  c 0.75  dtype: float64 |

|  |  |
| --- | --- |
| In[]: | # fancy indexing  data[['a', 'e']] |
| Out[] | a 0.25  e 1.25  dtype: float64 |

其中切片可能是混亂的根源。請留意，當使用顯式索引進行切片時（即data ['a'：'c']），最終索引在切片中包含*，而在使用隱式索引進行切片時（即*data [0：2]*），最終索引從切片中排除*。

索引採用指令: loc, iloc, and ix[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#索引採用指令:-loc,-iloc,-and-ix)

這些切片和索引約定可能會引起混淆。例如，如果你的Series有一個顯式的整數索引，那麼索引操作如data [1]將使用顯式索引，而切片操作如data [1：3] 將使用隱式的Python風格索引。

|  |  |
| --- | --- |
| In[]: | data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])  data |
| Out[] | 1 a  3 b  5 c  dtype: object |

|  |  |
| --- | --- |
| In[]: | # explicit index when indexing  data[1] |
| Out[] | 'a' |

|  |  |
| --- | --- |
| In[]: | # implicit index when slicing  data[1:3] |
| Out[] | 3 b  5 c  dtype: object |

由於在整數索引的情況下存在這種潛在的混淆，Pandas提供了一些特殊的 *indexer* 屬性，這些屬性明確地顯示了某些索引方案。這些不是函數方法，而是將特定切片接口顯示給Series中的資料的屬性。

首先，loc屬性允許索引和切片始終引用顯式索引：

|  |  |
| --- | --- |
| In[]: | data.loc[1] |
| Out[] | 'a' |

|  |  |
| --- | --- |
| In[]: | data.loc[1:3] |
| Out[] | 1 a  3 b  dtype: object |

iloc屬性允許索引和切片引用隱式的Python樣式索引：

|  |  |
| --- | --- |
| In[]: | data.iloc[1] |
| Out[] | 'b' |

|  |  |
| --- | --- |
| In[]: | data.iloc[1:3] |
| Out[] | 3 b  5 c  dtype: object |

第三個索引屬性ix是兩者的混合，而Series物件相當於標準的[]索引。ix索引器的目的將在DataFrame物件的上下文中變得更加明顯。

Python代碼的一個指導原則是“顯式優於隱式”。loc和iloc的顯式特性使它們在維護清晰可讀的代碼時非常有用; 特別是在整數索引的情況下，儘量使用這兩者來使代碼更容易閱讀和理解，並防止由於混合索引/切片約定導致的細微錯誤。

DataFrame中的資料選取[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#DataFrame中的資料選取)

DataFrame類同二維或結構化陣列與共享相同索引的Series結構字典。

將DataFrame當作字典(dictionary)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#將DataFrame當作字典(dictionary))

將DataFrame作為相關Series物件的字典。以美國各州面積與人口數的來舉例：

|  |  |
| --- | --- |
| In[]: | area = pd.Series({'California': 423967, 'Texas': 695662,  'New York': 141297, 'Florida': 170312,  'Illinois': 149995})  pop = pd.Series({'California': 39250017,'Texas': 27862596,  'Florida': 20612439,'New York': 19745289,  'Illinois': 12801539})  data = pd.DataFrame({'area':area, 'pop':pop})  data |
| Out[] | |  | area | pop | | --- | --- | --- | | California | 423967 | 39250017 | | Florida | 170312 | 20612439 | | Illinois | 149995 | 12801539 | | New York | 141297 | 19745289 | | Texas | 695662 | 27862596 | |

構成DataFrame行的單個Series可以通過column名稱的字典式索引來訪問：

|  |  |
| --- | --- |
| In[]: | data['area'] |
| Out[] | California 423967  Florida 170312  Illinois 149995  New York 141297  Texas 695662  Name: area, dtype: int64 |

同樣，我們可以使用屬性樣式訪問，其column名是字串：

|  |  |
| --- | --- |
| In[]: | data.area |
| Out[] | California 423967  Florida 170312  Illinois 149995  New York 141297  Texas 695662  Name: area, dtype: int64 |

此屬性樣式行訪問實際上訪問與字典樣式訪問完全相同的物件：

|  |  |
| --- | --- |
| In[]: | data.area is data['area'] |
| Out[] | True |

雖然這是一個有用的簡寫，但請記住，它並不適用於所有情況！ 例如，如果行名不是字串，或者行名與“DataFrame”的方法衝突，則無法進行此屬性樣式的訪問。 例如，DataFrame有一個pop（）方法，所以data.pop將指向這個而不是“pop”行：

|  |  |
| --- | --- |
| In[]: | data.pop is data['pop'] |
| Out[] | False |

特別是要避免通過屬性嘗試行分配的誘惑（即使用data ['pop'] = z而不是data.pop = z）。 與前面討論的Series物件一樣，這種字典式語法也可用於修改物件，在這種情況下添加一個新行：

|  |  |
| --- | --- |
| In[]: | data['density'] = data['pop'] / data['area']  data |
| Out[] | |  | area | pop | density | | --- | --- | --- | --- | | California | 423967 | 39250017 | 92.578000 | | Florida | 170312 | 20612439 | 121.027520 | | Illinois | 149995 | 12801539 | 85.346438 | | New York | 141297 | 19745289 | 139.743158 | | Texas | 695662 | 27862596 | 40.051916 | |

DataFrame當作二維陣列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#DataFrame當作二維陣列)

如前所述，我們還可以將DataFrame視為增強的二維陣列。我們可以使用values屬性檢查原始底層資料陣列：

|  |  |
| --- | --- |
| In[]: | data.values |
| Out[] | array([[4.23967000e+05, 3.92500170e+07, 9.25780002e+01],  [1.70312000e+05, 2.06124390e+07, 1.21027520e+02],  [1.49995000e+05, 1.28015390e+07, 8.53464382e+01],  [1.41297000e+05, 1.97452890e+07, 1.39743158e+02],  [6.95662000e+05, 2.78625960e+07, 4.00519160e+01]]) |

考慮到這一點，許多熟悉的類似陣列的觀察可以在DataFrame本身上完成。例如，我們可以將完整的DataFrame轉置為交換行和列：

|  |  |
| --- | --- |
| In[]: | data.T |
| Out[] | |  | California | Florida | Illinois | New York | Texas | | --- | --- | --- | --- | --- | --- | | area | 4.239670e+05 | 1.703120e+05 | 1.499950e+05 | 1.412970e+05 | 6.956620e+05 | | pop | 3.925002e+07 | 2.061244e+07 | 1.280154e+07 | 1.974529e+07 | 2.786260e+07 | | density | 9.257800e+01 | 1.210275e+02 | 8.534644e+01 | 1.397432e+02 | 4.005192e+01 | |

然而，當談到“DataFrame”物件的索引時，很明顯行的字典式索引排除了我們將其簡單地視為NumPy陣列的能力。 特別是，將單個索引傳遞給陣列會訪問一列：

|  |  |
| --- | --- |
| In[]: | data.values[0] |
| Out[] | array([4.23967000e+05, 3.92500170e+07, 9.25780002e+01]) |

將單個“索引”傳遞給DataFrame可以訪問一行：

|  |  |
| --- | --- |
| In[]: | data['area'] |
| Out[] | California 423967  Florida 170312  Illinois 149995  New York 141297  Texas 695662  Name: area, dtype: int64 |

因此，對於陣列樣式的索引，我們需要另一個約定。 Pandas再次使用前面提到的loc，iloc和ix索引器。使用iloc索引器，我們可以將底層陣列索引為好像它是一個簡單的NumPy陣列（使用隱式的Python樣式索引），但結果中保留了DataFrame索引和行標籤：

|  |  |
| --- | --- |
| In[]: | data.iloc[:3, :2] |
| Out[] | |  | area | pop | | --- | --- | --- | | California | 423967 | 39250017 | | Florida | 170312 | 20612439 | | Illinois | 149995 | 12801539 | |

類似地，使用loc索引器，我們可以使用顯式索引和行名稱以類似陣列的樣式索引基礎資料：

|  |  |
| --- | --- |
| In[]: | data.loc[:'Illinois', :'pop'] |
| Out[] | |  | area | pop | | --- | --- | --- | | California | 423967 | 39250017 | | Florida | 170312 | 20612439 | | Illinois | 149995 | 12801539 | |

ix索引器允許這兩種方法的混合：

|  |  |
| --- | --- |
| In[]: | data.ix[:3, :'pop'] |
| Out[] | |  | area | pop | | --- | --- | --- | | California | 423967 | 39250017 | | Florida | 170312 | 20612439 | | Illinois | 149995 | 12801539 | |

請記住，對於整數索引，ix索引器受到與整數索引的Series物件討論的相同的潛在混淆來源。

任何熟悉的NumPy風格的資料訪問模式都可以在這些索引器中使用。 例如，在loc索引器中，我們可以組合遮罩和花式索引，如下所示：

|  |  |
| --- | --- |
| In[]: | data.loc[data.density > 100, ['pop', 'density']] |
| Out[] | |  | pop | density | | --- | --- | --- | | Florida | 20612439 | 121.027520 | | New York | 19745289 | 139.743158 | |

任何這些索引約定也可用於設置或修改值; 也可以使用NumPy的標準方式完成：

|  |  |
| --- | --- |
| In[]: | data.iloc[0, 2] = 90  data |
| Out[] | |  | area | pop | density | | --- | --- | --- | --- | | California | 423967 | 39250017 | 90.000000 | | Florida | 170312 | 20612439 | 121.027520 | | Illinois | 149995 | 12801539 | 85.346438 | | New York | 141297 | 19745289 | 139.743158 | | Texas | 695662 | 27862596 | 40.051916 | |

其他索引約定[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-2-1.html#其他索引約定)

有一些額外的索引約定可能與前面的討論不一致，但在實踐中可能非常有用。 首先， *indexing* 引用行， *slicing* 引用列：

|  |  |
| --- | --- |
| In[]: | data['Florida':'Illinois'] |
| Out[] | |  | area | pop | density | | --- | --- | --- | --- | | Florida | 170312 | 20612439 | 121.027520 | | Illinois | 149995 | 12801539 | 85.346438 | |

這樣的切片也可以通過數字而不是索引來引用列：

|  |  |
| --- | --- |
| In[]: | data[1:3] |
| Out[] | |  | area | pop | density | | --- | --- | --- | --- | | Florida | 170312 | 20612439 | 121.027520 | | Illinois | 149995 | 12801539 | 85.346438 | |

類似地，直接遮罩操作也是按列而不是按行解釋的：

|  |  |
| --- | --- |
| In[]: | data[data.density > 100] |
| Out[] | |  | area | pop | density | | --- | --- | --- | --- | | Florida | 170312 | 20612439 | 121.027520 | | New York | 141297 | 19745289 | 139.743158 | |

這兩個約定在語法上類似於NumPy陣列上的約定，雖然這些約定可能不完全符合Pandas約定的模型，但它們在實踐中非常有用。

以下範例程式展示了Python 如何解碼 JSON 物件：

#### ICON2範例程式E3-4-2-2.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import pandas as pd  df1 = pd.read\_csv('2017pig.csv',encoding="utf-8", sep=",")  df1.columns = [ 'total\_amt', 'average\_weight', 'average\_price']  print(df1.describe())  print(df1.average\_price.max())  print(df1.sort\_values("average\_price", ascending=False).head(5))  print(df1[df1.average\_price>90]) |

#### ICON2範例程式說明

* 1行import pandas以進行資料分析。
* 2-3行運用pandas的read\_csv方法由'2017pig.csv'(此為某市場2017年全年毛豬交易行情資料，主要欄位：total\_amt(成交頭數-總數)、average\_weight(成交頭數-平均重量、average\_price(成交頭數-平均價格)讀入資料，檔案中以","分隔，編碼方式為"utf-8"，設定為df1物件。
* 4行運用pandas的describe 方法輸出對df1的敘述統計結果。
* 5行運用pandas的max方法輸出df1的市場全年成交最高平均價。
* 6行運用pandas的sort\_values 方法以"average\_price"為排序欄位進行由大到小排序後，將結果輸出前五位的資料。
* 7行輸出平均成交價大於90的資料。

#### ICON2輸出結果

total\_amt average\_weight average\_price

count 3001.000000 3001.000000 3001.000000

mean 1127.539820 123.920560 76.744558

std 747.706354 5.858829 3.445397

min 77.000000 94.290000 66.350000

25% 592.000000 119.790000 74.440000

50% 894.000000 122.740000 76.790000

75% 1539.000000 127.540000 78.950000

max 5231.000000 148.180000 95.410000

95.41

total\_amt average\_weight average\_price

2976 100 96.46 95.41

2999 91 103.11 94.50

2992 91 102.42 93.83

2985 90 99.20 93.43

2980 90 100.47 92.98

total\_amt average\_weight average\_price

2942 97 101.68 90.99

2955 101 101.59 90.57

2965 101 94.29 90.78

2969 97 102.70 91.89

2973 101 99.81 91.76

2976 100 96.46 95.41

2980 90 100.47 92.98

2985 90 99.20 93.43

2992 91 102.42 93.83

2999 91 103.11 94.50

## 4-3-3 pandas的聚合操作

#### ICON2範例程式E4-3-3-1.ipynb

聚合操作(Aggregation)與群聚(Grouping)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#聚合操作(Aggregation)與群聚(Grouping))

對大量資料進行總結主要利用聚合操作：如`sum（），mean（），median（），min（）和 max（）``，由此可以深入了解大型資料集中潛在的性質。

|  |  |
| --- | --- |
| In[]: | import numpy as np  import pandas as pd  class display(object):  """Display HTML representation of multiple objects"""  template = """<div style="float: left; padding: 10px;">  <p style='font-family:"Courier New", Courier, monospace'>{0}</p>{1}  </div>"""  def \_\_init\_\_(self, \*args):  self.args = args    def \_repr\_html\_(self):  return '\n'.join(self.template.format(a, eval(a).\_repr\_html\_())  for a in self.args)    def \_\_repr\_\_(self):  return '\n\n'.join(a + '\n' + repr(eval(a))  for a in self.args) |

行星資料[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#行星資料)

在此使用的行星資料集，提供了天文學家在其他恆星周圍發現的超過1000個太陽系外行星資料，可通過Seaborn軟件包獲得：

|  |  |
| --- | --- |
| In[]: | import seaborn as sns  planets = sns.load\_dataset('planets')  planets.shape |
| Out[] | (1035, 6) |

|  |  |
| --- | --- |
| In[]: | planets.head() |
| Out[] | |  | method | number | orbital\_period | mass | distance | year | | --- | --- | --- | --- | --- | --- | --- | | 0 | Radial Velocity | 1 | 269.300 | 7.10 | 77.40 | 2006 | | 1 | Radial Velocity | 1 | 874.774 | 2.21 | 56.95 | 2008 | | 2 | Radial Velocity | 1 | 763.000 | 2.60 | 19.84 | 2011 | | 3 | Radial Velocity | 1 | 326.030 | 19.40 | 110.62 | 2007 | | 4 | Radial Velocity | 1 | 516.220 | 10.50 | 119.47 | 2009 | |

Pandas簡單聚合操作[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#Pandas簡單聚合操作)

|  |  |
| --- | --- |
| In[]: | rng = np.random.RandomState(42)  ser = pd.Series(rng.rand(5))  ser |
| Out[] | 0 0.374540  1 0.950714  2 0.731994  3 0.598658  4 0.156019  dtype: float64 |

|  |  |
| --- | --- |
| In[]: | ser.sum() |
| Out[] | 2.811925491708157 |

|  |  |
| --- | --- |
| In[]: | ser.mean() |
| Out[] | 0.5623850983416314 |

對於DataFrame預設情況下聚合返回每列中的結果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In[]: | df = pd.DataFrame({'A': rng.rand(5),  'B': rng.rand(5)})  df | | | |
| Out[] |  | A | B |
|  | 0 | 0.155995 | 0.020584 |
|  | 1 | 0.058084 | 0.969910 |
|  | 2 | 0.866176 | 0.832443 |
|  | 3 | 0.601115 | 0.212339 |
|  | 4 | 0.708073 | 0.181825 |

|  |  |
| --- | --- |
| In[]: | df.mean() |
| Out[] | A 0.477888  B 0.443420  dtype: float64 |

通過指定axis參數，可以在每行內進行聚合操作：

|  |  |
| --- | --- |
| In[]: | df.mean(axis='columns') |
| Out[] | 0 0.088290  1 0.513997  2 0.849309  3 0.406727  4 0.444949  dtype: float64 |

Pandas 中Series 與DataFrame包含所有常用的聚合操作[包括Min, Max, ...]，甚至還有一個 describe() 指令提供最常用的敘述統計結果:

|  |  |
| --- | --- |
| In[]: | planets.dropna().describe() |
| Out[] | |  | number | orbital\_period | mass | distance | year | | --- | --- | --- | --- | --- | --- | | count | 498.00000 | 498.000000 | 498.000000 | 498.000000 | 498.000000 | | mean | 1.73494 | 835.778671 | 2.509320 | 52.068213 | 2007.377510 | | std | 1.17572 | 1469.128259 | 3.636274 | 46.596041 | 4.167284 | | min | 1.00000 | 1.328300 | 0.003600 | 1.350000 | 1989.000000 | | 25% | 1.00000 | 38.272250 | 0.212500 | 24.497500 | 2005.000000 | | 50% | 1.00000 | 357.000000 | 1.245000 | 39.940000 | 2009.000000 | | 75% | 2.00000 | 999.600000 | 2.867500 | 59.332500 | 2011.000000 | | max | 6.00000 | 17337.500000 | 25.000000 | 354.000000 | 2014.000000 | |

由此可以開始了解資料集的整體屬性，例如，我們在年份欄中看到，雖然早在1989年就發現了系外行星，但是直到2010年或之後才發現了一半已知的系外行星。 這主要歸功於開普勒計畫提供了一種專門設計用於尋找其他恆星周圍的遮蔽行星的太空望遠鏡。下表總結了其他一些內置的Pandas聚合操作方法：

| 聚合操作 | 說明 |
| --- | --- |
| count（） | 項目總數 |
| first（），last（） | 第一個和最後一個項目 |
| mean（），median（） | 平均值和中位數 |
| min（），max（） | 最小值和最大值 |
| std（），var（） | 標準差和變異數 |
| mad（） | 平均絕對偏差 |
| prod（） | 所有項目的乘積 |
| sum（） | 所有項目的總和 |

這些都是DataFrame和Series物件的方法。

GroupBy: Split, Apply, Combine[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#GroupBy:-Split,-Apply,-Combine)

簡單的聚合可以為您提供資料集的風格，但我們通常更願意在某些標籤或索引上有條件地聚合：這是在所謂的groupby操作中實現。groupby可以進行下列功能：

spli)：根據指定鍵的值分解和DataFrame分組。

apply ：計算單個組內的某些功能，通常是聚合，轉換或過濾。

Combine：將這些操作的結果合併到輸出陣列中。

|  |  |
| --- | --- |
| In[]: | df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],  'data': range(6)}, columns=['key', 'data'])  df |
| Out[] | |  | key | data | | --- | --- | --- | | 0 | A | 0 | | 1 | B | 1 | | 2 | C | 2 | | 3 | A | 3 | | 4 | B | 4 | | 5 | C | 5 | |

最基本的split-apply-combine操作可以使用DataFrame的groupby（）方法計算，傳遞所需鍵列的名稱：

|  |  |
| --- | --- |
| In[]: | df.groupby('key') |
| Out[] | <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000203130DE710> |

請注意，返回的不是一組DataFrame，而是一個DataFrameGroupBy物件。 這個物件就是神奇的地方：你可以把它想像成DataFrame的特殊視圖，它可以深入挖掘群組，但在應用聚合操作之前不會進行實際計算。 為了產生結果，我們可以將聚合應用於這個DataFrameGroupBy對象，該對象將執行適當的應用/組合步驟以產生所需的結果：

|  |  |
| --- | --- |
| In[]: | df.groupby('key').sum() |
| Out[] | |  | data | | --- | --- | | key |  | | A | 3 | | B | 5 | | C | 7 | |

`sum（）方法只是這裡的一種可能性; 可以應用幾乎任何常見的Pandas或NumPy聚合函數，以及幾乎任何有效的DataFrame``操作，我們將在下面的討論中看到。

GroupBy物件[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#GroupBy物件)

GroupBy物件是一個非常靈活的抽象。在許多方面，你可以簡單地將它視為“DataFrame”的集合，它可以解決困難的問題。 讓我們看一些使用Planets資料的例子。也許由GroupBy提供的最重要的操作是 *aggregate* ， *filter* ， *transform* 和 *apply* 。 我們將在[“聚合，過濾，轉換，應用”]（＃Aggregate，-Filter，-Transform，-Apply）中更全面地討論這些內容，但在此之前，我們將介紹一些可以與之配合使用的其他功能。 基本的GroupBy操作。

行索引(Column indexing)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#行索引(Column-indexing))

`GroupBy物件以與DataFrame相同的方式支持行索引，並返回修改後的GroupBy``物件。 例如：

|  |  |
| --- | --- |
| In[]: | planets.groupby('method') |
| Out[] | <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000203130DE780> |

|  |  |
| --- | --- |
| In[]: | planets.groupby('method')['orbital\_period'] |
| Out[] | <pandas.core.groupby.groupby.SeriesGroupBy object at 0x0000020313108DA0> |

在這裡，我們通過引用其列名從原始的DataFrame組中選擇了一個特定的Series組。與GroupBy物件一樣，在我們呼叫物件上的聚合之前，不會進行任何計算：

|  |  |
| --- | --- |
| In[]: | planets.groupby('method')['orbital\_period'].median() |
| Out[] | method  Astrometry 631.180000  Eclipse Timing Variations 4343.500000  Imaging 27500.000000  Microlensing 3300.000000  Orbital Brightness Modulation 0.342887  Pulsar Timing 66.541900  Pulsation Timing Variations 1170.000000  Radial Velocity 360.200000  Transit 5.714932  Transit Timing Variations 57.011000  Name: orbital\_period, dtype: float64 |

這給出了每種方法對軌道周期（以天為單位）的一般尺度的概念。

群組迭代(Iteration over groups)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#群組迭代(Iteration-over-groups))

GroupBy物件支持對群組進行直接迭代，將每個群組作為Series或DataFrame返回：

|  |  |
| --- | --- |
| In[]: | for (method, group) in planets.groupby('method'):  print("{0:30s} shape={1}".format(method, group.shape)) |
| Out[] | Astrometry shape=(2, 6)  Eclipse Timing Variations shape=(9, 6)  Imaging shape=(38, 6)  Microlensing shape=(23, 6)  Orbital Brightness Modulation shape=(3, 6)  Pulsar Timing shape=(5, 6)  Pulsation Timing Variations shape=(1, 6)  Radial Velocity shape=(553, 6)  Transit shape=(397, 6)  Transit Timing Variations shape=(4, 6) |

Dispatch方法[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#Dispatch方法)

通過一些Python類別魔術，任何未由GroupBy物件顯式實現的方法都將被傳遞並呼叫群組，無論它們是DataFrame還是Series物件。 例如，您可以使用DataFrame的describe（）方法來執行一組聚合來描述資料中的每個群組：

|  |  |
| --- | --- |
| In[]: | planets.groupby('method')['year'].describe().unstack() |
| Out[] | method  count Astrometry 2.000000  Eclipse Timing Variations 9.000000  Imaging 38.000000  Microlensing 23.000000  Orbital Brightness Modulation 3.000000  Pulsar Timing 5.000000  Pulsation Timing Variations 1.000000  Radial Velocity 553.000000  Transit 397.000000  Transit Timing Variations 4.000000  mean Astrometry 2011.500000  Eclipse Timing Variations 2010.000000  Imaging 2009.131579  Microlensing 2009.782609  Orbital Brightness Modulation 2011.666667  Pulsar Timing 1998.400000  Pulsation Timing Variations 2007.000000  Radial Velocity 2007.518987  Transit 2011.236776  Transit Timing Variations 2012.500000  std Astrometry 2.121320  Eclipse Timing Variations 1.414214  Imaging 2.781901  Microlensing 2.859697  Orbital Brightness Modulation 1.154701  Pulsar Timing 8.384510  Pulsation Timing Variations NaN  Radial Velocity 4.249052  Transit 2.077867  Transit Timing Variations 1.290994  ...  50% Astrometry 2011.500000  Eclipse Timing Variations 2010.000000  Imaging 2009.000000  Microlensing 2010.000000  Orbital Brightness Modulation 2011.000000  Pulsar Timing 1994.000000  Pulsation Timing Variations 2007.000000  Radial Velocity 2009.000000  Transit 2012.000000  Transit Timing Variations 2012.500000  75% Astrometry 2012.250000  Eclipse Timing Variations 2011.000000  Imaging 2011.000000  Microlensing 2012.000000  Orbital Brightness Modulation 2012.000000  Pulsar Timing 2003.000000  Pulsation Timing Variations 2007.000000  Radial Velocity 2011.000000  Transit 2013.000000  Transit Timing Variations 2013.250000  max Astrometry 2013.000000  Eclipse Timing Variations 2012.000000  Imaging 2013.000000  Microlensing 2013.000000  Orbital Brightness Modulation 2013.000000  Pulsar Timing 2011.000000  Pulsation Timing Variations 2007.000000  Radial Velocity 2014.000000  Transit 2014.000000  Transit Timing Variations 2014.000000  Length: 80, dtype: float64 |

Aggregate, filter, transform, apply[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#Aggregate,-filter,-transform,-apply)

GroupBy對像有aggregate（），filter（），transform（）和apply（）有效實現各種有用的方法 組合分組資料之前的操作。在此將使用這個DataFrame：

|  |  |
| --- | --- |
| In[]: | rng = np.random.RandomState(0)  df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],  'data1': range(6),  'data2': rng.randint(0, 10, 6)},  columns = ['key', 'data1', 'data2'])  df |
| Out[] | |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0 | 5 | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 3 | A | 3 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 | |

聚合操作(Aggregation)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#聚合操作(Aggregation))

我們現在熟悉GroupBy聚合與`sum（），median（）等，但aggregate（）``方法允許更多的靈活性。 它可以採用字符串，函數或其列表，並一次計算所有聚合。 這是一個結合所有這些的快速示例：

|  |  |
| --- | --- |
| In[]: | df.groupby('key').aggregate(['min', np.median, max]) |
| Out[] | |  | data1 | | | data2 | | | | --- | --- | --- | --- | --- | --- | --- | |  | min | median | max | min | median | max | | key |  |  |  |  |  |  | | A | 0 | 1.5 | 3 | 3 | 4.0 | 5 | | B | 1 | 2.5 | 4 | 0 | 3.5 | 7 | | C | 2 | 3.5 | 5 | 3 | 6.0 | 9 | |

另一個有用的模式是將字典映射行名稱映射到要應用於該行的操作：

|  |  |
| --- | --- |
| In[]: | df.groupby('key').aggregate({'data1': 'min',  'data2': 'max'}) |
| Out[] | |  | data1 | data2 | | --- | --- | --- | | key |  |  | | A | 0 | 5 | | B | 1 | 7 | | C | 2 | 9 | |

過濾操作(Filtering)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#過濾操作(Filtering))

過濾操作允許您根據群組屬性刪除資料。例如，我們可能希望保留標準偏差大於某個臨界值的所有組：

|  |  |
| --- | --- |
| In[]: | def filter\_func(x):  return x['data2'].std() > 4  display('df', "df.groupby('key').std()", "df.groupby('key').filter(filter\_func)") |
| Out[] | df   |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0 | 5 | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 3 | A | 3 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 |   df.groupby('key').std()   |  | data1 | data2 | | --- | --- | --- | | key |  |  | | A | 2.12132 | 1.414214 | | B | 2.12132 | 4.949747 | | C | 2.12132 | 4.242641 |   df.groupby('key').filter(filter\_func)   |  | key | data1 | data2 | | --- | --- | --- | --- | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 | |

filter函數應返回一個布林值，指定群組是否通過過濾。 這裡因為群組A沒有大於4的標準差，所以從結果中刪除它。

轉換(Transformation)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#轉換(Transformation))

雖然聚合必須返回資料的簡化版本，但轉換可以返回完整資料的某些轉換版本以重新組合。對於這種變換，輸出與輸入的形狀相同。一個常見的例子是通過減去分組均值來使資料居中：

|  |  |
| --- | --- |
| In[]: | df.groupby('key').transform(lambda x: x - x.mean()) |
| Out[] | |  | data1 | data2 | | --- | --- | --- | | 0 | -1.5 | 1.0 | | 1 | -1.5 | -3.5 | | 2 | -1.5 | -3.0 | | 3 | 1.5 | -1.0 | | 4 | 1.5 | 3.5 | | 5 | 1.5 | 3.0 | |

apply() 方法[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#apply()-方法)

apply（）方法允許您將任意函數應用於群組結果。該函數應該採用DataFrame，並返回一個Pandas物件（例如，DataFrame，Series）或一個純量; 組合操作將根據返回的輸出類型進行調整。 例如，這裡是一個apply（），它將第一行標準化為第二行的總和：

|  |  |
| --- | --- |
| In[]: | def norm\_by\_data2(x):  # x is a DataFrame of group values  x['data1'] /= x['data2'].sum()  return x  display('df', "df.groupby('key').apply(norm\_by\_data2)") |
| Out[] | df   |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0 | 5 | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 3 | A | 3 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 |   df.groupby('key').apply(norm\_by\_data2)   |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0.000000 | 5 | | 1 | B | 0.142857 | 0 | | 2 | C | 0.166667 | 3 | | 3 | A | 0.375000 | 3 | | 4 | B | 0.571429 | 7 | | 5 | C | 0.416667 | 9 | |

指定拆分鍵¶

串列，陣列，系列或索引均可提供分組鍵，key可以是任何系列或串列，其長度與“DataFrame”的長度相匹配。 例如：

|  |  |
| --- | --- |
| In[]: | L = [0, 1, 0, 1, 2, 0]  display('df', 'df.groupby(L).sum()') |
| Out[] | df   |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0 | 5 | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 3 | A | 3 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 |   df.groupby(L).sum()   |  | data1 | data2 | | --- | --- | --- | | 0 | 7 | 17 | | 1 | 4 | 3 | | 2 | 4 | 7 | |

當然，這意味著還有另一種更冗長的方式來實現之前的df.groupby（'key'）

|  |  |
| --- | --- |
| In[]: | display('df', "df.groupby(df['key']).sum()") |
| Out[] | df   |  | key | data1 | data2 | | --- | --- | --- | --- | | 0 | A | 0 | 5 | | 1 | B | 1 | 0 | | 2 | C | 2 | 3 | | 3 | A | 3 | 3 | | 4 | B | 4 | 7 | | 5 | C | 5 | 9 |   df.groupby(df['key']).sum()   |  | data1 | data2 | | --- | --- | --- | | key |  |  | | A | 3 | 8 | | B | 5 | 7 | | C | 7 | 12 | |

字典或系列映射索引到組[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#字典或系列映射索引到組)

另一種方法是提供將索引值映射到組鍵的字典：

|  |  |
| --- | --- |
| In[]: | df2 = df.set\_index('key')  mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}  display('df2', 'df2.groupby(mapping).sum()') |
| Out[] | df2   |  | data1 | data2 | | --- | --- | --- | | key |  |  | | A | 0 | 5 | | B | 1 | 0 | | C | 2 | 3 | | A | 3 | 3 | | B | 4 | 7 | | C | 5 | 9 |   df2.groupby(mapping).sum()   |  | data1 | data2 | | --- | --- | --- | | consonant | 12 | 19 | | vowel | 3 | 8 | |

任何Python函數[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#任何Python函數)

與映射類似，您可以傳遞任何將輸入索引值並輸出組的Python函數：

|  |  |
| --- | --- |
| In[]: | display('df2', 'df2.groupby(str.lower).mean()') |
| Out[] | df2   |  | data1 | data2 | | --- | --- | --- | | key |  |  | | A | 0 | 5 | | B | 1 | 0 | | C | 2 | 3 | | A | 3 | 3 | | B | 4 | 7 | | C | 5 | 9 |   df2.groupby(str.lower).mean()   |  | data1 | data2 | | --- | --- | --- | | a | 1.5 | 4.0 | | b | 2.5 | 3.5 | | c | 3.5 | 6.0 | |

有效鍵串列[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#有效鍵串列)

此外，任何前面的鍵選擇可以組合在一個多索引上分組：

|  |  |
| --- | --- |
| In[]: | df2.groupby([str.lower, mapping]).mean() |
| Out[] | |  |  | data1 | data2 | | --- | --- | --- | --- | | a | vowel | 1.5 | 4.0 | | b | consonant | 2.5 | 3.5 | | c | consonant | 3.5 | 6.0 | |

Grouping實例[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-3-1.html#Grouping實例)

下面這個例子可以在幾行Python代碼中將所有這些放在一起，並通過方法計算每十年發現的行星：

|  |  |
| --- | --- |
| In[]: | decade = 10 \* (planets['year'] // 10)  decade = decade.astype(str) + 's'  decade.name = 'decade'  planets.groupby(['method', decade])['number'].sum().unstack().fillna(0) |
| Out[] | | decade | 1980s | 1990s | 2000s | 2010s | | --- | --- | --- | --- | --- | | method |  |  |  |  | | Astrometry | 0.0 | 0.0 | 0.0 | 2.0 | | Eclipse Timing Variations | 0.0 | 0.0 | 5.0 | 10.0 | | Imaging | 0.0 | 0.0 | 29.0 | 21.0 | | Microlensing | 0.0 | 0.0 | 12.0 | 15.0 | | Orbital Brightness Modulation | 0.0 | 0.0 | 0.0 | 5.0 | | Pulsar Timing | 0.0 | 9.0 | 1.0 | 1.0 | | Pulsation Timing Variations | 0.0 | 0.0 | 1.0 | 0.0 | | Radial Velocity | 1.0 | 52.0 | 475.0 | 424.0 | | Transit | 0.0 | 0.0 | 64.0 | 712.0 | | Transit Timing Variations | 0.0 | 0.0 | 0.0 | 9.0 | |

這顯示了在查看真實資料集時，結合我們討論過的許多操作的強大功能。我們立即大致了解過去幾十年內行星何時以及如何被發現！

#### ICON2範例程式E4-3-3-2.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import pandas as pd  df1 = pd.read\_csv('2017pig.csv',encoding="utf-8", sep=",")  df1.columns = [ 'total\_amt', 'average\_weight', 'average\_price']  print(df1.describe())  print(df1.average\_price.max())  print(df1.sort\_values("average\_price", ascending=False).head(5))  print(df1[df1.average\_price>90]) |

#### ICON2輸出結果

['1001', '大鵬華城', '38']

['1002', '汐止火車站', '56']

['1003', '汐止區公所', '46']

['1004', '國泰綜合醫院', '56']

['1005', '裕隆公園', '40']

['1006', '捷運大坪林站(3號出口)', '32']

['1007', '汐科火車站(北)', '34']

['1008', '興華公園', '40']

['1009', '三重國民運動中心', '68']

['1010', '捷運三重站(3號出口)', '34']

## 4-3-4 Pasdas排序

XML(eXtensible Markup Language，可延伸標記式語言)，是一種[標記式語言](https://zh.wikipedia.org/wiki/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)。標記(Markup)指[電腦](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA)所能理解的資訊符號，通過此種標記，電腦之間可以處理包含各種資訊的文章等。XML是從[標準通用標記式語言](https://zh.wikipedia.org/wiki/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)(SGML)中簡化修改出來的。它主要用到的有可延伸標記式語言、[可延伸樣式語言](https://zh.wikipedia.org/wiki/%E5%8F%AF%E6%89%A9%E5%B1%95%E6%A0%B7%E5%BC%8F%E8%AF%AD%E8%A8%80)(XSL)、[XBRL](https://zh.wikipedia.org/wiki/XBRL)和[XPath](https://zh.wikipedia.org/wiki/XPath)等。如下面的範例檔 menu.xml 所示：

<?xml version="1.0"?>

<menu>

<breakfast hours="7-11">

<item price="$60">breakfast burritos</item>

<item price="$40">pancakes</item>

</breakfast>

<lunch hours="13-3">

<item price="$50">hamburger</item>

</lunch>

<dinner hours="3-10">

<item price="80">spaghetti</item>

</dinner>

</menu>

以下是 XML 的一些重要特性：

1. 標籤以一個 < 字元開頭，例如示例中的標籤 menu、breakfast、lunch、dinner 和 item；
2. 忽略空格；
3. 通常開始標籤（例如 <menu>）後接著一段內容，最後是相匹配的結束標籤(例如 </menu>)；
4. 標籤間可能存在多級嵌套，例如範例檔中，標籤 item 是標籤 breakfast、lunch 和 dinner 的子標籤，也是標籤 menu 的子標籤；
5. 可選屬性（attribute）可以出現在開始標籤裡，例如 price 是 item 的一個屬性；
6. 標籤中可以包含值（value），本例中每個 item 都會有一個值，比如第二個 breakfast item 的 pancakes；
7. 如果一個命名為 thing 的標籤沒有內容或者子標籤，它可以用一個在右尖括弧的前面添加斜杠的簡單標籤所表示，例如 <thing/> 代替開始和結束都存在的標籤 <thing> 和 </thing>；
8. 存放資料的位置可以是任意的——屬性、值或者子標籤。例如也可以把最後一個 item 標籤寫作 <item price ="$8.00" food ="spaghetti"/>。
9. XML 通常用於資料傳送和消息，它存在一些子格式，如 RSS 和 Atom。工業界有許多定制化的 XML 格式，例如[金融領域](http://www.service-architecture.com/articles/xml/finance_xml.html)。
10. XML 的靈活性導致出現了很多方法和性能各異的 Python 庫。
11. 在 Python 中解析 XML 最簡單的方法是使用 ElementTree，下面的代碼用來解析 menu.xml 檔以及輸出一些標籤和屬性：

讀 xml 文檔時，通過 ElementTree() 構建空樹，parse() 讀入 xml 文檔，解析映射到空樹；getroot() 獲取根節點，通過下標可訪問相應的節點；tag 獲取節點名，attrib 獲取節點屬性字典，text 獲取節點文本；find() 返回匹配到節點名的第一個節點，findall() 返回匹配到節點名的所有節點，find()、findall() 兩者都僅限當前節點的一級子節點，都支持 xpath 路徑提取節點；iter() 創建樹反覆運算器，遍歷當前節點的所有子節點，返回匹配到節點名的所有節點；remove() 移除相應的節點。

下面的代碼用來解析 menu.xml 檔以及輸出一些標籤和屬性：

#### ICON2範例程式E4-3-4-1.ipynb

Pandas排序[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-4-1.html#Pandas排序)

Series[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-4-1.html#Series)

Series是一個類似於一維陣列的物件，包含資料陣列和相關的資料標籤貞烈。 資料可以是任何NumPy資料類型，標籤是Series的索引。Series就像一個固定長度的有序字典，傳入一個dict創建一個Series：

|  |  |
| --- | --- |
| In[]: | from pandas import Series, DataFrame  import pandas as pd  import numpy as np  dict\_1 = {'apple' : 100, 'ball' : 200, 'car' : 300}  ser\_3 = Series(dict\_1)  ser\_3 |
| Out[] | apple 100  ball 200  car 300  dtype: int64 |

通過傳入索引重新排序Series（未找到的索引是NaN）：

|  |  |
| --- | --- |
| In[]: | index = ['foo', 'bar', 'baz', 'donkey']  ser\_4 = Series(dict\_1, index=index)  ser\_4 |
| Out[] | foo NaN  bar NaN  baz NaN  donkey NaN  dtype: float64 |

使用pandas方法檢查NaN，下面這兩種方式等效：

|  |  |
| --- | --- |
| In[]: | pd.isnull(ser\_4) |
| Out[] | foo True  bar True  baz True  donkey True  dtype: bool |

|  |  |
| --- | --- |
| In[]: | ser\_4.isnull() |
| Out[] | foo True  bar True  baz True  donkey True  dtype: bool |

Series在算術運算中自動對齊不同的索引資料：

|  |  |
| --- | --- |
| In[]: | ser\_3 + ser\_4 |
| Out[] | apple NaN  ball NaN  bar NaN  baz NaN  car NaN  donkey NaN  foo NaN  dtype: float64 |

命名 Series與 Series索引:

|  |  |
| --- | --- |
| In[]: | ser\_4.name = 'appleballcardonkey'  ser\_4.index.name = 'label'  ser\_4 |
| Out[] | label  foo NaN  bar NaN  baz NaN  donkey NaN  Name: appleballcardonkey, dtype: float64 |

重命名 Series的索引：

|  |  |
| --- | --- |
| In[]: | ser\_4.index = ['ap', 'ba', 'ca', 'do']  ser\_4 |
| Out[] | ap NaN  ba NaN  ca NaN  do NaN  Name: appleballcardonkey, dtype: float64 |

排序(Sorting)和排名(Ranking)[¶](file:///C:\Users\User\TQC+%20網頁資料擷取與分析特訓教材\第3章\3-3\E3-3-4-1.html#排序(Sorting)和排名(Ranking))

按索引對Series進行排序：

|  |  |
| --- | --- |
| In[]: | ser\_4.sort\_index() |
| Out[] | ap NaN  ba NaN  ca NaN  do NaN  Name: appleballcardonkey, dtype: float64 |

按Series值排序Series：

|  |  |
| --- | --- |
| In[]: | ser\_4.sort\_values() |
| Out[] | Out[13]:  ap NaN  ba NaN  ca NaN  do NaN  Name: appleballcardonkey, dtype: float64 |

|  |  |
| --- | --- |
| In[]: | df\_12 = DataFrame(np.arange(12).reshape((3, 4)),  index=['three', 'one', 'two'],  columns=['c', 'a', 'b', 'd'])  df\_12 |
| Out[] | |  | c | a | b | d | | --- | --- | --- | --- | --- | | three | 0 | 1 | 2 | 3 | | one | 4 | 5 | 6 | 7 | | two | 8 | 9 | 10 | 11 | |

按索引對DataFrame進行排序：

|  |  |
| --- | --- |
| In[]: | df\_12.sort\_index() |
| Out[] | |  | c | a | b | d | | --- | --- | --- | --- | --- | | one | 4 | 5 | 6 | 7 | | three | 0 | 1 | 2 | 3 | | two | 8 | 9 | 10 | 11 | |

按行降序排列DataFrame：

|  |  |
| --- | --- |
| In[]: | df\_12.sort\_index(axis=1, ascending=False) |
| Out[] | |  | d | c | b | a | | --- | --- | --- | --- | --- | | three | 3 | 0 | 2 | 1 | | one | 7 | 4 | 6 | 5 | | two | 11 | 8 | 10 | 9 | |

按行對DataFrame的值進行排序：

|  |  |
| --- | --- |
| In[]: | df\_12.sort\_values(by=['d', 'c']) |
| Out[] | |  | c | a | b | d | | --- | --- | --- | --- | --- | | three | 0 | 1 | 2 | 3 | | one | 4 | 5 | 6 | 7 | | two | 8 | 9 | 10 | 11 | |

排名類似於numpy.argsort，除了通過為每個群組分配平均排名來打破關聯：

|  |  |
| --- | --- |
| In[]: | ser\_11 = Series([7, -5, 7, 4, 2, 0, 4, 7])  ser\_11 = ser\_11.sort\_values()  ser\_11 |
| Out[] | 1 -5  5 0  4 2  3 4  6 4  0 7  2 7  7 7  dtype: int64 |

|  |  |
| --- | --- |
| In[]: | ser\_11.rank() |
| Out[] | 1 1.0  5 2.0  4 3.0  3 4.5  6 4.5  0 7.0  2 7.0  7 7.0  dtype: float64 |

根據Series出現在資料中的時間對Series進行排名：

|  |  |
| --- | --- |
| In[]: | ser\_11.rank(method='first') |
| Out[] | 1 1.0  5 2.0  4 3.0  3 4.0  6 5.0  0 6.0  2 7.0  7 8.0  dtype: float64 |

使用群組的最大排名按降序排列Series：

|  |  |
| --- | --- |
| In[]: | ser\_11.rank(ascending=False, method='max') |
| Out[] | 1 8.0  5 7.0  4 6.0  3 5.0  6 5.0  0 3.0  2 3.0  7 3.0  dtype: float64 |

DataFrame可以對行或列進行排名：

|  |  |
| --- | --- |
| In[]: | df\_13 = DataFrame({'apple' : [7, -5, 7, 4, 2, 0, 4, 7],  'ball' : [-5, 4, 2, 0, 4, 7, 7, 8],  'car' : [-1, 2, 3, 0, 5, 9, 9, 5]})  df\_13 |
| Out[] | |  | apple | ball | car | | --- | --- | --- | --- | | 0 | 7 | -5 | -1 | | 1 | -5 | 4 | 2 | | 2 | 7 | 2 | 3 | | 3 | 4 | 0 | 0 | | 4 | 2 | 4 | 5 | | 5 | 0 | 7 | 9 | | 6 | 4 | 7 | 9 | | 7 | 7 | 8 | 5 | |

在列上排名DataFrame：

|  |  |
| --- | --- |
| In[]: | df\_13.rank() |
| Out[] | |  | apple | ball | car | | --- | --- | --- | --- | | 0 | 7.0 | 1.0 | 1.0 | | 1 | 1.0 | 4.5 | 3.0 | | 2 | 7.0 | 3.0 | 4.0 | | 3 | 4.5 | 2.0 | 2.0 | | 4 | 3.0 | 4.5 | 5.5 | | 5 | 2.0 | 6.5 | 7.5 | | 6 | 4.5 | 6.5 | 7.5 | | 7 | 7.0 | 8.0 | 5.5 | |

在行上排名DataFrame：

|  |  |
| --- | --- |
| In[]: | df\_13.rank(axis=1) |
| Out[] | |  | apple | ball | car | | --- | --- | --- | --- | | 0 | 3.0 | 1.0 | 2.0 | | 1 | 1.0 | 3.0 | 2.0 | | 2 | 3.0 | 1.0 | 2.0 | | 3 | 3.0 | 1.5 | 1.5 | | 4 | 1.0 | 2.0 | 3.0 | | 5 | 1.0 | 2.0 | 3.0 | | 6 | 1.0 | 2.0 | 3.0 | | 7 | 2.0 | 3.0 | 1.0 | |

#### ICON2範例程式E3-4-4-2.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import json  import pandas as pd  with open("AQI.json",encoding = 'utf8') as file:  data = json.load(file)  df = pd.DataFrame(data)  print(df)  df1=df.sort\_values(by="AQI", ascending=False)  print('以AQI遞減排序')  print(df1['AQI'])  print(df1.groupby("County").count()["SiteName"])  df2=df1['AQI']  print(df2.describe()) |

#### ICON2範例程式說明

* 1-2行import所需套件。
* 3-6行開啟"AQI.json"後運用json的load方法讀取資料，設定為data物件，接著讀取為pandas DataFrame格式的df物件，再將之列印輸出。
* 7-9行運用pandas的sort\_values方法對"AQI"欄位進行遞減排序，設定為df1物件，接著列印輸出AQI欄位資料。
* 10行運用pandas的groupby方法對"County"欄位進行群組處理，接著計算每個群組的筆數後輸出。
* 11-12行對AQI欄位進行敘述統計後輸出。

#### ICON2輸出結果

AQI CO CO\_8hr County ... SiteName Status WindDirec WindSpeed

0 55 0.26 0.3 基隆市 ... 基隆 普通 255 1

1 32 0.41 0.4 新北市 ... 汐止 良好 245 1.4

2 52 0.24 0.3 新北市 ... 萬里 普通 229 2.2

3 48 0.25 0.2 新北市 ... 新店 良好 183 1

4 66 0.46 0.4 新北市 ... 土城 普通 327 0.5

…下略

[77 rows x 22 columns]

以AQI遞減排序

4 66

7 61

63 60

69 59

15 58

0 55

6 55

5 53

…下略

Name: AQI, Length: 77, dtype: object

County

南投縣 3

嘉義市 1

嘉義縣 2

基隆市 1

宜蘭縣 2

屏東縣 3

彰化縣 3

新北市 12

新竹市 1

新竹縣 2

桃園市 6

澎湖縣 1

臺中市 5

臺北市 7

臺南市 4

臺東縣 2

花蓮縣 1

苗栗縣 3

連江縣 1

金門縣 1

雲林縣 4

高雄市 12

Name: SiteName, dtype: int64

count 77

unique 42

top 33

freq 4

Name: AQI, dtype: object

# 4-4 CSV 檔案繪製圖表

# 4-5 Numpy 模組應用

# 4-6 隨機數的應用

綜合範例

### Whack_Notepad_++ 綜合範例1

* 某日各果菜批發市場之西瓜與香瓜之拍賣行情(價與量)如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 西瓜價 | 西瓜量 | 香瓜價 | 香瓜量 |
| 三重區 | 9.00 | 203674 | 13.20 | 18894 |
| 台中市 | 11.70 | 180785 | 12.30 | 54894 |
| 台北一 | 10.10 | 127802 | 14.70 | 18563 |
| 台北二 | 11.80 | 28604 | 14.90 | 21963 |
| 台東市 | 13.20 | 600 | 13.10 | 900 |
| 板橋區 | 6.90 | 38071 | 9.60 | 3555 |
| 高雄市 | 12.10 | 35660 | 10.60 | 9005 |
| 嘉義市 | 12.00 | 15000 | 13.00 | 12000 |
| 鳳山區 | 11.70 | 48770 | 9.10 | 14370 |
| 豐原區 | 9.84 | 6100 | 11.89 | 8980 |

* 上表資料已經輸入在**GE3-1.py中，**請撰寫一程式，讀入所輸入資料，其中行標題為項目(西瓜價、西瓜量、香瓜價、香瓜價)，列題標為交易市場；接著完成下列：  
  A.輸出上面表格。  
  B.以西瓜價遞減排序後，輸出各市場的西瓜價  
  C.輸出後三位的行情(西瓜/香瓜價雨量)  
  D.台北一市場的行情(西瓜/香瓜價與量)  
  E.將”三重市”改為”三重區”；”香瓜價”改為”洋香瓜價”； ”香瓜量”改為”洋香瓜量”；重新輸出整個表格。。

#### bestpractices 提示

1. 需要import pandas以進行資料分析。
2. 採用DataFrame資料架構，分別以串列方式輸入datas、index、column，接著直接列印DataFrame物件即可得上面表格。
3. 用sort\_values指令即可以西瓜價遞減排序，輸出各市場的西瓜價則須用pandas的切片功能達成。
4. 利用tail指令即可輸出後三位的行情(西瓜/香瓜價與量)。
5. 利用loc指令可以輸出台北一市場的行情(西瓜/香瓜價與量)，
6. 運用index/column串列定位即可修改特定元素的值。

#### ICON2輸入與輸出樣本

輸入

datas = [[9,203674,13.2,18894], [11.7,180785,12.3,54894], [10.1,127802,14.7,18563] ,

[11.8,28604,14.9,21963], [13.2,600,13.1,900], [6.9,38071,9.6,3555],

[12.1,35660,10.6,9005], [12,15000,13,12000], [11.7,48770,9.1,14370],

[9.84,6100,11.89,8980]]

indexs= ["三重市","台中市","台北一","台北二","台東市","板橋區","高雄市","嘉義市","鳳山區","豐原區"]

columns = ["西瓜價", "西瓜量", "香瓜價", "香瓜量"]

輸出

行標題為項目，列題標為交易市場

西瓜價 西瓜量 香瓜價 香瓜量

三重市 9.00 203674 13.20 18894

台中市 11.70 180785 12.30 54894

台北一 10.10 127802 14.70 18563

台北二 11.80 28604 14.90 21963

台東市 13.20 600 13.10 900

板橋區 6.90 38071 9.60 3555

高雄市 12.10 35660 10.60 9005

嘉義市 12.00 15000 13.00 12000

鳳山區 11.70 48770 9.10 14370

豐原區 9.84 6100 11.89 8980

以西瓜價遞減排序

台東市 13.20

高雄市 12.10

嘉義市 12.00

台北二 11.80

台中市 11.70

鳳山區 11.70

台北一 10.10

豐原區 9.84

三重市 9.00

板橋區 6.90

Name: 西瓜價, dtype: float64

後三位的西瓜/香瓜價雨量

西瓜價 西瓜量 香瓜價 香瓜量

豐原區 9.84 6100 11.89 8980

三重市 9.00 203674 13.20 18894

板橋區 6.90 38071 9.60 3555

台北一市場的行情

西瓜價 10.1

西瓜量 127802.0

香瓜價 14.7

香瓜量 18563.0

Name: 台北一, dtype: float64

全體市場行情

西瓜價 西瓜量 洋香瓜價 洋香瓜量

三重區 9.00 203674 13.20 18894

台中市 11.70 180785 12.30 54894

台北一 10.10 127802 14.70 18563

台北二 11.80 28604 14.90 21963

台東市 13.20 600 13.10 900

板橋區 6.90 38071 9.60 3555

高雄市 12.10 35660 10.60 9005

嘉義市 12.00 15000 13.00 12000

鳳山區 11.70 48770 9.10 14370

豐原區 9.84 6100 11.89 8980

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | datas = [[9,203674,13.2,18894], [11.7,180785,12.3,54894], [10.1,127802,14.7,18563] ,  [11.8,28604,14.9,21963], [13.2,600,13.1,900], [6.9,38071,9.6,3555],  [12.1,35660,10.6,9005], [12,15000,13,12000], [11.7,48770,9.1,14370],  [9.84,6100,11.89,8980]]  indexs= ["三重市","台中市","台北一","台北二","台東市","板橋區","高雄市","嘉義市","鳳山區","豐原區"]  columns = ["西瓜價", "西瓜量", "香瓜價", "香瓜量"]  import pandas as pd  df = pd.DataFrame(datas, columns=columns, index=indexs)  print('行標題為項目，列題標為交易市場')  print(df)  print()  df1=df.sort\_values(by="西瓜價", ascending=False)  print('以西瓜價遞減排序')  print(df1['西瓜價'])  print()  print('後三位的西瓜/香瓜價與量')  print(df1.tail(3))  print()  df.loc["台北一","西瓜價"]  print('台北一市場的行情')  print(df.loc["台北一",:])  indexs[0] = "三重區"  df.index = indexs  columns[2] = "洋香瓜價"  columns[3] = "洋香瓜量"  print()  print('全體市場行情')  df.columns = columns  print(df) |

#### ICON2參考解答程式說明

* 1-3行分別以串列方式輸入datas、indexs、columns等資料內容，
* 4行import所需套件pandas以進行資料分析。
* 5行採用DataFrame資料架構，將1-3行讀入資料匯入pandas，設定為df物件。
* 6-7行列印df物件即可得上面表格。8行為列印分隔行。
* 9行用sort\_values指令即可以西瓜價遞減排序。
* 10-11行用pandas的切片功能達成輸出各市場的西瓜價。13行為列印分隔行。
* 14行利用tail指令輸出後三位的行情(西瓜/香瓜價與量)。15行為列印分隔行。
* 17-18行利用loc指令輸出台北一市場的行情(西瓜/香瓜價與量)，
* 19-25行運用index/column串列定位修改特定元素的值。
* 26行列印修改結果。

### Whack_Notepad_++ 綜合範例2

請撰寫一程式，讀取'2017pig.csv'（此為某市場2017年全年毛豬交易行情資料，主要欄位：total\_amt(成交頭數-總數)、average\_weight(成交頭數-平均重量)、average\_price(成交頭數-平均價格)，本題需要完成下列：  
A.進行簡單敘述統計  
B. 依照average\_price遞減排序後列印前五位資料

C.找出成交頭數-平均價格>90的資料  
D.依照average\_price遞減排序後列印average\_price。  
E.列印後三位的資料。

#### bestpractices 提示

1. 需要import pandas以進行資料分析。
2. 運用pandas的read\_csv方法由'2017pig.csv'讀入資料。
3. 運用describe進行敘述統計。
4. 運用max可以找出欄位的最大值。
5. 運用sort\_values方法進行遞減排序，接者運用hesd方法輸出前面5列。
6. Pandas可以針對欄位設定條件進行切片。
7. Pandas可以運用tail方法找後三位的資料。

#### ICON2輸入與輸出樣本

輸入：

讀取'2017pig.csv'（此為某市場2017年全年毛豬交易行情資料）

輸出

total\_amt average\_weight average\_price

count 3001.000000 3001.000000 3001.000000

mean 1127.539820 123.920560 76.744558

std 747.706354 5.858829 3.445397

min 77.000000 94.290000 66.350000

25% 592.000000 119.790000 74.440000

50% 894.000000 122.740000 76.790000

75% 1539.000000 127.540000 78.950000

max 5231.000000 148.180000 95.410000

95.41

total\_amt average\_weight average\_price

2976 100 96.46 95.41

2999 91 103.11 94.50

2992 91 102.42 93.83

2985 90 99.20 93.43

2980 90 100.47 92.98

total\_amt average\_weight average\_price

2942 97 101.68 90.99

2955 101 101.59 90.57

2965 101 94.29 90.78

2969 97 102.70 91.89

2973 101 99.81 91.76

2976 100 96.46 95.41

2980 90 100.47 92.98

2985 90 99.20 93.43

2992 91 102.42 93.83

2999 91 103.11 94.50

以average\_price遞減排序

0 77.01

1 75.94

2 75.28

3 73.68

4 72.50

…中略

2996 83.90

2997 82.36

2998 84.37

2999 94.50

3000 84.55

Name: average\_price, Length: 3001, dtype: float64

後三位的資料

total\_amt average\_weight average\_price

2998 84 121.44 84.37

2999 91 103.11 94.50

3000 77 120.64 84.55

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import pandas as pd  df1 = pd.read\_csv('2017pig.csv',encoding="utf-8", sep=",")  df1.columns = [ 'total\_amt', 'average\_weight', 'average\_price']  print(df1.describe())  print(df1.average\_price.max())  print(df1.sort\_values("average\_price", ascending=False).head(5))  pr int(df1[df1.average\_price>90])  print('以average\_price遞減排序')  print(df1['average\_price'])  print()  print('後三位的資料')  print(df1.tail(3))  print() |

#### ICON2參考解答程式說明

* 1行import pandas以進行資料分析。
* 2行運用pandas的read\_csv方法由'2017pig.csv'讀入資料，編碼方式採用"utf-8",檔案中以","分隔，設定為df1物件，為DataFrame資料架構。
* 4行輸出df1的敘述統計結果。
* 5行輸出df1中average\_price欄位的最大值。
* 6行用sort\_values方法對df1中average\_price欄位進行遞減排序，接者輸出前面5列。
* 7行列印df中average\_price欄位>90的資料。
* 8-9行列印df1中average\_price欄位資料。10行為列印分隔行。
* 11-13行列印df1中後三位的資料。

### Whack_Notepad_++ 綜合範例3

請撰寫程式，讀取"AQI.json"(此為環保署每小時提供各測站之空氣品質指標（AQI），主要欄位：SiteName(測站名稱)、County(縣市)、AQI(空氣品質指標)、Pollutant(空氣污染指標物)、Status(狀態)、SO2(二氧化硫(ppb))、CO(一氧化碳(ppm))、CO\_8hr(一氧化碳8小時移動平均(ppm))、O3(臭氧(ppb))、O3\_8hr(臭氧8小時移動平均(ppb))、PM10(懸浮微粒(μg/m3))、PM2.5(細懸浮微粒(μg/m3))、NO2(二氧化氮(ppb))、NOx(氮氧化物(ppb))、NO(一氧化氮(ppb))、WindSpeed(風速(m/sec))、WindDirec(風向(degrees))、PublishTime(資料建置日期)、PM2.5\_AVG(細懸浮微粒移動平均值(μg/m3))、PM10\_AVG(懸浮微粒移動平均值(μg/m3))、Latitude(經度)、Longitude(緯度)，執行下列：  
A.列印基隆測站AQI  
B.列印基隆測站全部資料  
C.將index中”基隆”改為”基隆測站”  
D.將columns中"AQI"改為"空氣品質指標","Pollutant"改為"空氣污染指標物"  
E.列印全台灣測站資料。

#### bestpractices 提示

1. 需要import json以讀取json檔案； 需要import pandas以進行資料分析

#### ICON2輸入與輸出樣本

輸入

讀取"AQI.json"，其資料範例如下：

[{"SiteName":"基隆","County":"基隆市","AQI":"55","Pollutant":"細懸浮微粒","Status":"普通","SO2":"3","CO":"0.26","CO\_8hr":"0.3","O3":"6","O3\_8hr":"11","PM10":"27","PM2.5":"17","NO2":"16","NOx":"19","NO":"3.4","WindSpeed":"1","WindDirec":"255","PublishTime":"2018-06-29 06:00","PM2.5\_AVG":"17","PM10\_AVG":"26","Latitude":"25.129167","Longitude":"121.760056"},{"SiteName":"汐止","County":"新北市","AQI":"32","Pollutant":"","Status":"良好","SO2":"1.8","CO":"0.41","CO\_8hr":"0.4","O3":"7.6","O3\_8hr":"10","PM10":"33","PM2.5":"13","NO2":"17","NOx":"24","NO":"7.1","WindSpeed":"1.4","WindDirec":"245","PublishTime":"2018-06-29 06:00","PM2.5\_AVG":"10","PM10\_AVG":"30","Latitude":"25.067131","Longitude":"121.6423"},{"SiteName":"萬里","County":"新北市","AQI":"52","Pollutant":"細懸浮微粒","Status":"普通","SO2":"3.5","CO":"0.24","CO\_8hr":"0.3","O3":"16","O3\_8hr":"20","PM10":"30","PM2.5":"14","NO2":"7.8","NOx":"8.6","NO":"0.8","WindSpeed":"2.2","WindDirec":"229","PublishTime":"2018-06-29 06:00","PM2.5\_AVG":"16","PM10\_AVG":"30","Latitude":"25.179667","Longitude":"121.689881"},

…下略

輸出

NO.0001 捷運市政府站(3號出口) 20180627154432 57

AQI CO CO\_8hr County ... SiteName Status WindDirec WindSpeed

0 55 0.26 0.3 基隆市 ... 基隆 普通 255 1

1 32 0.41 0.4 新北市 ... 汐止 良好 245 1.4

2 52 0.24 0.3 新北市 ... 萬里 普通 229 2.2

3 48 0.25 0.2 新北市 ... 新店 良好 183 1

…中略

76 32 0.14 0.2 雲林縣 ... 崙背 良好 202 0.7

[77 rows x 22 columns]

基隆測站AQI55

基隆測站資料

AQI 55

CO 0.26

CO\_8hr 0.3

County 基隆市

Latitude 25.129167

Longitude 121.760056

NO 3.4

NO2 16

NOx 19

O3 6

O3\_8hr 11

PM10 27

PM10\_AVG 26

PM2.5 17

PM2.5\_AVG 17

Pollutant 細懸浮微粒

PublishTime 2018-06-29 06:00

SO2 3

Status 普通

WindDirec 255

WindSpeed 1

Name: 基隆, dtype: object

全台灣測站資料

County Latitude Longitude ... PublishTime SO2 Status

SiteName ...

基隆測站 基隆市 25.129167 121.760056 ... 2018-06-29 06:00 3 普通

汐止 新北市 25.067131 121.6423 ... 2018-06-29 06:00 1.8 良好

萬里 新北市 25.179667 121.689881 ... 2018-06-29 06:00 3.5 普通

…下略

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import json  import pandas as pd  with open("AQI.json",encoding = 'utf8') as file:  data = json.load(file)  df = pd.DataFrame(data)  print(df)  df.set\_index("SiteName" , inplace=True)  print('基隆測站AQI'+df.loc["基隆","AQI"])  print('基隆測站資料')  print(df.loc["基隆",:])  df.rename(index={"基隆":"基隆測站"},inplace=True)  df.rename(columns={"AQI":"空氣品質指標","Pollutant":"空氣污染指標物"},inplace=True)  print()  print('全台灣測站資料')  print(df.loc[:,"County":"Status"]) |

#### ICON2參考解答程式說明

* 1-2行import所需套件。
* 3-4行以json.load方法開啟所給檔案，設定為data物件。
* 5-6行將data物件讀取為pandas的DataFrame資料結構，設定為df物件，並將之輸出。
* 7行設定df物件的index為"SiteName"。
* 8行運用pandas的loc方法列印基隆測站AQI。
* 9-10行運用loc方法切片取得基隆測站資料並列印。
* 11-12行將index中"基隆"改為"基隆測站"；columns中”AQI"改為"空氣品質指標","Pollutant"改為"空氣污染指標物"。
* 13-15行運用loc方法列印全台灣測站資料。

### Whack_Notepad_++ 綜合範例4

請撰寫一程式，讀取'F-D0047-093.zip'（此為中央氣象局鄉鎮天氣預報資料集，主要欄位：CITY(縣市),DISTRICT(鄉鎮市區),GEOCODE(鄉鎮市區編碼),DAY(預報日期),TIME(預報時間),T(溫度)、,Td(露點溫度)、,RH(相對濕度)、Wd(風向)、Ws(風速)、,AT(體感溫度)、Wx(天氣現象)、Wx\_n(天氣現象編號)、PoP6h(降雨機率6小時分段)、PoP12h(降雨機率12小時分段)、get\_day(取得日期)）找出臺北市與高雄市各區72小時氣象預報資料，並將其轉存為GE3-4-output.csv。。

#### bestpractices 提示

1. 需要import json以讀取Json檔案；需要import csv以寫入csv檔案。

#### ICON2輸入與輸出樣本

輸入

'F-D0047-093.zip'為為中央氣象局鄉鎮天氣預報資料集壓縮檔，包含各鄉鎮市區72小時、一週、二週天氣預報；其代碼採內政部戶役政資訊系統資料代碼，編碼如下：

[臺北市](https://zh.wikipedia.org/wiki/%E8%87%BA%E5%8C%97%E5%B8%82) 63/[高雄市](https://zh.wikipedia.org/wiki/%E9%AB%98%E9%9B%84%E5%B8%82) 64/[新北市](https://zh.wikipedia.org/wiki/%E6%96%B0%E5%8C%97%E5%B8%82) 65/[臺中市](https://zh.wikipedia.org/wiki/%E8%87%BA%E4%B8%AD%E5%B8%82) 66/[臺南市](https://zh.wikipedia.org/wiki/%E8%87%BA%E5%8D%97%E5%B8%82) 67/[桃園市](https://zh.wikipedia.org/wiki/%E6%A1%83%E5%9C%92%E5%B8%82) 68/  
[宜蘭縣](https://zh.wikipedia.org/wiki/%E5%AE%9C%E8%98%AD%E7%B8%A3) 10002/[新竹縣](https://zh.wikipedia.org/wiki/%E6%96%B0%E7%AB%B9%E7%B8%A3) 10004/[苗栗縣](https://zh.wikipedia.org/wiki/%E8%8B%97%E6%A0%97%E7%B8%A3) 10005/[彰化縣](https://zh.wikipedia.org/wiki/%E5%BD%B0%E5%8C%96%E7%B8%A3) 10007/[南投縣](https://zh.wikipedia.org/wiki/%E5%8D%97%E6%8A%95%E7%B8%A3) 10008

[雲林縣](https://zh.wikipedia.org/wiki/%E9%9B%B2%E6%9E%97%E7%B8%A3) 10009/[嘉義縣](https://zh.wikipedia.org/wiki/%E5%98%89%E7%BE%A9%E7%B8%A3) 10010/[屏東縣](https://zh.wikipedia.org/wiki/%E5%B1%8F%E6%9D%B1%E7%B8%A3) 10013/[臺東縣](https://zh.wikipedia.org/wiki/%E8%87%BA%E6%9D%B1%E7%B8%A3) 10014/[花蓮縣](https://zh.wikipedia.org/wiki/%E8%8A%B1%E8%93%AE%E7%B8%A3) 10015

[澎湖縣](https://zh.wikipedia.org/wiki/%E6%BE%8E%E6%B9%96%E7%B8%A3) 10016/[基隆市](https://zh.wikipedia.org/wiki/%E5%9F%BA%E9%9A%86%E5%B8%82) 10017/[新竹市](https://zh.wikipedia.org/wiki/%E6%96%B0%E7%AB%B9%E5%B8%82) 10018/[嘉義市](https://zh.wikipedia.org/wiki/%E5%98%89%E7%BE%A9%E5%B8%82) 10020/[連江縣](https://zh.wikipedia.org/wiki/%E9%80%A3%E6%B1%9F%E7%B8%A3_(%E4%B8%AD%E8%8F%AF%E6%B0%91%E5%9C%8B)) 09007

[金門縣](https://zh.wikipedia.org/wiki/%E9%87%91%E9%96%80%E7%B8%A3) 09020

並以各別xml檔案形式壓縮在壓縮檔中，因此若要找尋個別縣市資料只要在程式中選取特定縣市解壓縮即可。例如本題要找尋臺北市與高雄市各區72小時氣象預報資料，故需要處理壓縮檔中下列檔案：

'63\_72hr\_CH.xml','64\_72hr\_CH.xml'(63為台北市；64為高雄市)

輸出

以溫度遞減排序

201 34

177 34

208 34

32 34

200 34

88 34

280 34

80 34

185 34

209 34

184 34

272 34

248 34

56 34

176 34

152 34

1168 34

65 33

57 33

1032 33

904 33

…中略

723 22

724 21

725 21

742 21

741 21

734 21

726 21

733 21

Name: T, Length: 1200, dtype: object

目前最高氣溫為34

目前最低氣溫為21

且所得之GE3-4-output.csv中資料如下：

CITY,DISTRICT,GEOCODE,DAY,TIME,T,TD,RH,WD,WS,BF,AT,Wx,Wx\_n,PoP6h,PoP12h,get\_day

臺北市,南港區,6300900,2018-07-21,12:00:00,31,29,87,西北風,2,2,36,短暫陣雨,26,50,50,2018-07-21

臺北市,南港區,6300900,2018-07-21,15:00:00,31,28,84,偏西風,1,<= 1,35,陰,03,50,50,2018-07-21

臺北市,南港區,6300900,2018-07-21,18:00:00,29,27,88,西南風,1,<= 1,34,陰,03,20,50,2018-07-21

臺北市,南港區,6300900,2018-07-21,21:00:00,28,27,93,偏東風,1,<= 1,33,陰,03,20,50,2018-07-21

臺北市,南港區,6300900,2018-07-22,00:00:00,27,27,97,偏東風,1,<= 1,32,陰,03,20,50,2018-07-21

臺北市,南港區,6300900,2018-07-22,03:00:00,27,27,100,東南風,1,<= 1,32,多雲,02,20,50,2018-07-21

臺北市,南港區,6300900,2018-07-22,06:00:00,27,25,93,西南風,1,<= 1,31,短暫陣雨,26,50,50,2018-07-21

…中略

臺北市,大安區,6300300,2018-07-21,12:00:00,32,29,86,西北風,2,2,37,短暫陣雨,26,50,50,2018-07-21

臺北市,大安區,6300300,2018-07-21,15:00:00,31,28,82,偏西風,2,<= 1,36,陰,03,50,50,2018-07-21

臺北市,大安區,6300300,2018-07-21,18:00:00,30,27,84,偏西風,1,<= 1,34,陰,03,20,50,2018-07-21

臺北市,大安區,6300300,2018-07-21,21:00:00,29,27,91,偏東風,1,<= 1,34,陰,03,20,50,2018-07-21

臺北市,大安區,6300300,2018-07-22,00:00:00,28,27,94,東北風,1,<= 1,33,陰,03,20,60,2018-07-21

臺北市,大安區,6300300,2018-07-22,03:00:00,28,27,98,偏北風,1,<= 1,32,多雲,02,20,60,2018-07-21

…中略

高雄市,甲仙區,6403300,2018-07-23,03:00:00,23,22,96,偏東風,1,<= 1,26,陰,03,30,30,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,06:00:00,23,22,94,西南風,1,<= 1,25,陰,03,20,30,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,09:00:00,28,28,89,西南風,1,<= 1,33,陰,03,20,30,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,12:00:00,32,29,86,偏南風,2,<= 1,38,短暫陣雨或雷雨,36,60,60,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,15:00:00,32,29,88,偏南風,1,<= 1,38,陰,03,60,60,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,18:00:00,29,26,95,偏南風,1,<= 1,34,陰,03,20,60,2018-07-21

高雄市,甲仙區,6403300,2018-07-23,21:00:00,26,24,96,東南風,1,<= 1,30,陰,03,20,60,2018-07-21

高雄市,甲仙區,6403300,2018-07-24,00:00:00,25,22,88,東南風,1,<= 1,28,陰,03,20,80,2018-07-21

高雄市,甲仙區,6403300,2018-07-24,03:00:00,24,20,80,偏南風,1,<= 1,26,陰,03,20,80,2018-07-21

高雄市,甲仙區,6403300,2018-07-24,06:00:00,23,20,80,偏南風,1,<= 1,26,陰,03,80,80,2018-07-21

高雄市,甲仙區,6403300,2018-07-24,09:00:00,29,27,80,偏南風,1,<= 1,33,短暫陣雨,26,80,80,2018-07-21

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | import pandas as pd  import numpy as np  from bs4 import BeautifulSoup  import datetime  import zipfile  f=zipfile.ZipFile('F-D0047-093.zip')  file = ['63\_72hr\_CH.x12.46ml','64\_72hr\_CH.xml']  CITY = []  DISTRICT = []  GEOCODE = []  DAY = []  TIME = []  T = []  TD = []  RH = []  WD = []  WS = []  BF = []  AT = []  Wx = []  Wx\_n = []  PoP6h = []  PoP12h = []  get\_day = []  for filename in file:  try:  data = f.read(filename).decode('utf8')  soup = BeautifulSoup(data,"xml")  city = soup.locationsName.text  a = soup.find\_all("location")  for i in range(0,len(a)):  location = a[i]  district = location.find\_all("locationName")[0].text  geocode = location.geocode.text  weather = location.find\_all("weatherElement")  time = weather[1].find\_all("dataTime")  for j in range(0,len(time)):  x = time[j].text.split("T")  DAY.append(x[0])  time\_1 = x[1].split("+")  TIME.append(time\_1[0])  CITY.append(city)  DISTRICT.append(district)  GEOCODE.append(geocode)  get\_day.append(today)  for t in weather[0].find\_all("value"):  T.append(t.text)  for td in weather[1].find\_all("value"):  TD.append(td.text)  for rh in weather[2].find\_all("value"):  RH.append(rh.text)  for wd in weather[5].find\_all("value"):  WD.append(wd.text)  ws = weather[6].find\_all("value")  for k in range(0,len(ws),2):  WS.append(ws[k].text)  BF.append(ws[k+1].text)  for at in weather[8].find\_all("value"):  AT.append(at.text)  wx = weather[9].find\_all("value")  for w in range(0,len(wx),2):  Wx.append(wx[w].text)  Wx\_n.append(wx[w+1].text)  rain1 = weather[3].find\_all("value")  for l in range(0,len(rain1)):  pop6 = rain1[l].text  PoP6h.append(pop6)  PoP6h.append(pop6)  #PoP6h.append("x") #1200時  #PoP6h.append("x") #1200時  rain2 = weather[4].find\_all("value")  for m in range(0,len(rain2)):  pop12 = rain2[m].text  PoP12h.append(pop12)  PoP12h.append(pop12)  PoP12h.append(pop12)  PoP12h.append(pop12)  except:  break  f.close()  data = {"CITY":CITY,"DISTRICT":DISTRICT,"GEOCODE":GEOCODE,"DAY" : DAY,"TIME" : TIME,"T":T,"TD" : TD,"RH":RH,  "WD" : WD,"WS" : WS,"BF":BF,"AT" : AT,"Wx": Wx,"Wx\_n":Wx\_n,"PoP6h" : PoP6h,"PoP12h" :PoP12h,"get\_day":get\_day}  df = pd.DataFrame(data,columns=["CITY","DISTRICT","GEOCODE","DAY","TIME","T","TD","RH","WD","WS","BF","AT","Wx","Wx\_n","PoP6h","PoP12h","get\_day"])  df1=df.sort\_values(by="T", ascending=False)  print('以溫度遞減排序')  print(df1['T'])  print("目前最高氣溫為"+str(df1["T"].max()))  print("目前最低氣溫為"+str(df1["T"].min()))  save\_name = "GE3-4-output.csv"  df.to\_csv(save\_name,index=False,encoding="utf\_8\_sig") |

#### ICON2參考解答程式說明

* 1-6行import所需套件。
* 7-8行對獲取的"F-D0047-093.zip"壓縮檔檔案進行解壓縮，並設定需要檔名。'63\_72hr\_CH.xml'為台北市各區72小時氣象預報資料，'64\_72hr\_CH.xml'為高雄市各區72小時氣象預報資料。
* 9-25行設定需要收集資料初始空串列，其中欄位意義如下：  
  CITY(縣市)、DISTRICT(鄉鎮市區)、GEOCODE(編碼)、DAY(日期),TIME(時間) T(溫度)、,Td(露點溫度)、,RH(相對濕度)、Wd(風向)、Ws(風速)、,AT(體感溫度)、Wx(天氣現象)、Wx\_n(天氣現象編號)、PoP6h(降雨機率6小時分段)、PoP12h(降雨機率12小時分段)、get\_day(取得日期)
* 26-80行利用BeautifulSoup的"xml"模式，解析各xml檔案中，找出所需要資料新增到各資料串列中。
* 81-82行將所得資料以pandas的DataFrame資料格式整理，設定為df1物件。
* 83-85行對df1物件以以溫度遞減排序後列印。
* 86-87行列印最高與最低氣溫。
* 88-89行寫入csv檔案。

### Whack_Notepad_++ 綜合範例5

某日各市場之商品A與商品B之成交價與成交量如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 商品A價 | 商品A量 | 商品B價 | 商品B量 |
| NY | 9.00 | 203674 | 13.20 | 18894 |
| HK | 11.70 | 180785 | 12.30 | 54894 |
| SH | 10.10 | 127802 | 14.70 | 18563 |
| TP | 11.80 | 28604 | 14.90 | 21963 |
| TO | 13.20 | 600 | 13.10 | 900 |
| SZ | 6.90 | 38071 | 9.60 | 3555 |
| SG | 12.10 | 35660 | 10.60 | 9005 |
| MB | 12.00 | 15000 | 13.00 | 12000 |
| DB | 11.70 | 48770 | 9.10 | 14370 |
| FR | 9.84 | 6100 | 11.89 | 8980 |

請撰寫程式，讀入所輸入資料，其中prices為各市場(商品A成交價, 商品B成交價)，amounts為各市場(商品A成交量, 商品B成交量)，接著完成下列：  
A.計算並輸出商品A最高成交價  
B.計算並輸出商品B最低成交量  
C.計算並輸出商品A總成交量與商品B總成交量。(總成交量為該商品成交量的總和)  
D.計算並輸出商品A總成交金額與商品B總成交金額。(總成交金額為該商品在各市場成交金額(等於成交價\*成交量)的總和)  
E.計算並輸出商品A總均價與商品B總均價。(總均價為該商品成交總成交金額/總成交量)

#### bestpractices 提示

1. 需要import numpy以快速分析資料。

#### ICON2輸入與輸出樣本

輸入

prices = np.array([[9,13.2],

[11.7,12.3],

[10.1,14.7] ,

[11.8,14.9],

[13.2,13.1],

[6.9,9.6],

[12.1,10.6],

[12,13],

[11.7,9.1],

[9.84,11.89]])

amounts = np.array([[203674,18894],

[180785,54894],

[127802,18563] ,

[28604,21963],

[600,900],

[38071,3555],

[35660,9005],

[15000,12000],

[48770,14370],

[6100,8980]])

輸出

商品A最高成交價13.2

商品B最低成交量54894

商品A總成交量685066 商品B總成交量163124

商品A總成交金額7089306.800000001 商品B總成交金額2059632.0

商品A總均價10.348355924830601 商品B總均價12.626173953556803

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import numpy as np  prices = np.array([[9,13.2],  [11.7,12.3],  [10.1,14.7] ,  [11.8,14.9],  [13.2,13.1],  [6.9,9.6],  [12.1,10.6],  [12,13],  [11.7,9.1],  [9.84,11.89]])  amounts = np.array([[203674,18894],  [180785,54894],  [127802,18563] ,  [28604,21963],  [600,900],  [38071,3555],  [35660,9005],  [15000,12000],  [48770,14370],  [6100,8980]])  total\_amounts=amounts.sum(axis=0)  print("商品A最高成交價"+str(prices[0].max(axis=0)))  print("商品B最低成交量"+str(amounts[1].min(axis=0)))  print("商品A總成交量"+str(total\_amounts[0])+" 商品B總成交量"+str(total\_amounts[1]))  total\_sales=(amounts\*prices).sum(axis=0)  print("商品A總成交金額"+str(total\_sales[0])+" 商品B總成交金額"+str(total\_sales[1]))  print("商品A總均價"+str(total\_sales[0]/total\_amounts[0])+" 商品B總均價"+str(total\_sales[1]/total\_amounts[1])) |

#### ICON2參考解答程式說明

* 1行import所需套件
* 2-3行讀取prices與amounts，分別以numpy陣列結構儲存。
* 4行以sum方法計算兩種商品總成交量。
* 5行以max方法計算並輸出商品A最高成交價
* 6行以max方法計算並輸出商品B最低成交量
* 7行計算並輸出商品A總成交量與商品B總成交量。(總成交量為該商品成交量的總和)
* 8行計算並輸出商品A總成交金額與商品B總成交金額。(總成交金額為該商品在各市場成交金額(等於成交價\*成交量)的總和)
* 9行計算並輸出商品A總均價與商品B總均價。(總均價為該商品成交總成交金額/總成交量)

### Whack_Notepad_++ 綜合範例6

請撰寫一程式，讀取Dengue\_Daily\_last12m.csv（此為登革熱近12個月每日確定病例統計，主要欄位： 發病日、個案研判日、通報日、性別、年齡層、居住縣市、居住鄉鎮、居住村里、最小統計區、最小統計區中心點X、最小統計區中心點Y、一級統計區、二級統計區、感染縣市、感染鄉鎮、感染村里、是否境外移入、感染國家、確定病例數、居住村里代碼、感染村里代碼、血清型、內政部居住縣市代碼、內政部居住鄉鎮代碼、內政部感染縣市代碼、內政部感染鄉鎮代碼）。接著配合下列要求輸出：  
A.輸出居住縣市病例人數，並按遞減順序顯示。  
B.輸出感染國家病例人數，並按遞減順序顯示。  
C.輸出台北市各區病例人數  
D.輸出台北市最近病例。  
bestpractices 提示

1. 需要import xml.etree.ElementTree以讀取xml檔案；需要import csv以寫入csv檔案。
2. 中華郵政公司縣市鄉鎮中英對照資訊主要欄位說明：欄位1：郵遞區號、欄位2：縣市鄉鎮(中文)、欄位3：縣市鄉鎮(英文)。

#### ICON2輸入與輸出樣本

輸入

讀取Dengue\_Daily\_last12m.csv，內容如下：

發病日,個案研判日,通報日,性別,年齡層,居住縣市,居住鄉鎮,居住村里,最小統計區,最小統計區中心點X,最小統計區中心點Y,一級統計區,二級統計區,感染縣市,感染鄉鎮,感染村里,是否境外移入,感染國家,確定病例數,居住村里代碼,感染村里代碼,血清型,內政部居住縣市代碼,內政部居住鄉鎮代碼,內政部感染縣市代碼,內政部感染鄉鎮代碼

2017/06/04,2017/06/10,2017/06/09,男,30-34,屏東縣,屏東市,勝豐里,A1301-0841-00,120.49159,22.66706,A1301-54-008,A1301-54,None,None,None,是,泰國,1,1001301-019,None,None,10013,1001301,None,None

2017/06/04,2017/06/06,2017/06/06,男,20-24,新竹市,北區,崇禮里,A1802-0597-00,120.96078,24.80306,A1802-43-002,A1802-43,None,None,None,是,馬來西亞,1,1001802-005,None,None,10018,1001802,None,None

2017/06/05,2017/06/13,2017/06/12,男,45-49,苗栗縣,苑裡鎮,房裡里,A0502-0149-00,120.65068,24.43575,A0502-05-009,A0502-05,None,None,None,是,越南,1,1000502-008,None,第一型,10005,1000502,None,None

…下略

輸出

發病日 個案研判日 通報日 性別 ... 內政部居住縣市代碼 內政部居住鄉鎮代碼 內政部感染縣市代碼 內政部感染鄉鎮代碼

居住縣市 ...

台北市 76 76 76 76 ... 76 76 76 76

新北市 60 60 60 60 ... 60 60 60 60

台中市 43 43 43 43 ... 43 43 43 43

桃園市 43 43 43 43 ... 43 43 43 43

高雄市 42 42 42 42 ... 42 42 42 42

台南市 20 20 20 20 ... 20 20 20 20

彰化縣 14 14 14 14 ... 14 14 14 14

新竹市 8 8 8 8 ... 8 8 8 8

屏東縣 7 7 7 7 ... 7 7 7 7

南投縣 6 6 6 6 ... 6 6 6 6

新竹縣 5 5 5 5 ... 5 5 5 5

苗栗縣 5 5 5 5 ... 5 5 5 5

嘉義縣 4 4 4 4 ... 4 4 4 4

雲林縣 3 3 3 3 ... 3 3 3 3

宜蘭縣 2 2 2 2 ... 2 2 2 2

基隆市 2 2 2 2 ... 2 2 2 2

台東縣 2 2 2 2 ... 2 2 2 2

澎湖縣 2 2 2 2 ... 2 2 2 2

嘉義市 1 1 1 1 ... 1 1 1 1

[19 rows x 25 columns]

發病日 個案研判日 通報日 ... 內政部居住鄉鎮代碼 內政部感染縣市代碼 內政部感染鄉鎮代碼

是否境外移入 ...

否 10 10 10 ... 10 10 10

是 335 335 335 ... 335 335 335

[2 rows x 25 columns]

發病日 個案研判日 通報日 性別 ... 內政部居住縣市代碼 內政部居住鄉鎮代碼 內政部感染縣市代碼 內政部感染鄉鎮代碼

感染國家 ...

越南 94 94 94 94 ... 94 94 94 94

菲律賓 50 50 50 50 ... 50 50 50 50

泰國 44 44 44 44 ... 44 44 44 44

馬來西亞 33 33 33 33 ... 33 33 33 33

緬甸 29 29 29 29 ... 29 29 29 29

[5 rows x 25 columns]

發病日 個案研判日 通報日 性別 ... 內政部居住縣市代碼 內政部居住鄉鎮代碼 內政部感染縣市代碼 內政部感染鄉鎮代碼

居住鄉鎮 ...

中山區 8 8 8 8 ... 8 8 8 8

中正區 10 10 10 10 ... 10 10 10 10

信義區 7 7 7 7 ... 7 7 7 7

內湖區 10 10 10 10 ... 10 10 10 10

北投區 1 1 1 1 ... 1 1 1 1

南港區 7 7 7 7 ... 7 7 7 7

士林區 7 7 7 7 ... 7 7 7 7

大同區 5 5 5 5 ... 5 5 5 5

大安區 10 10 10 10 ... 10 10 10 10

文山區 2 2 2 2 ... 2 2 2 2

松山區 3 3 3 3 ... 3 3 3 3

萬華區 6 6 6 6 ... 6 6 6 6

[12 rows x 25 columns]

2018/06/25

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #網路抓取開放資料進行處理樣本資料做常態分配檢定  import pandas as pd  df1 = pd.read\_csv('Dengue\_Daily\_last12m.csv',encoding="utf-8", sep=",",header=0)  df\_county=df1.groupby("居住縣市").count()  print(df\_county.sort\_values("內政部居住鄉鎮代碼", ascending=False))  print(df1.groupby("是否境外移入").count())  df\_country=df1.groupby("感染國家").count()  print(df\_country.sort\_values("內政部居住鄉鎮代碼", ascending=False).head(5))  df\_taipei=df1[df1.*居住縣市=="台北市"*]  print(df\_taipei.groupby("居住鄉鎮").count())  print(df\_taipei.發病日.max()) |

#### ICON2參考解答程式說明

* 2行import pandas以進行資料分析。
* 3行運用pandas的read\_csv方法由' Dengue\_Daily\_last12m.csv'讀入資料，編碼方式採用"utf-8",檔案中以","分隔，設定為df1物件，為DataFrame資料架構。
* 4行運用pandas的groupby方法依照"居住縣市"分群，然後以count方法分群計算各群數目，設定給df\_county物件。
* 5行將df\_county依照"內政部居住鄉鎮代碼"由大到小排序後列印輸出。
* 6行運用pandas的groupby方法依照"是否境外移入"分群，然後以count方法分群計算各群數目後列印輸出。
* 7行運用pandas的groupby方法依照"感染國家"分群，然後以count方法分群計算各群數目，設定給df\_country物件。
* 8行將df\_country依照"內政部居住鄉鎮代碼"由大到小排序後列印輸出前五位。
* 9行運用選取方法從df1中選出居住縣市=="台北市"，設定為df\_taipei物件。
* 10行運用pandas的groupby方法依照"居住鄉鎮"分群，然後以count方法分群計算各群數目後列印輸出。
* 11行列印台北市病例中最近的發病日。

### Whack_Notepad_++ 綜合範例7

請撰寫程式，讀取'集保戶股權分散表0050.csv' (此為台灣集保公司集保戶股權分散表中0050的資料，主要欄位：資料日期、證券代號、持股分級、人數、股數、佔集保庫存數比例%，持股分級的定義，說明如下：

1.第1級至第15級，係持股為1:1-999 、2:1,000-5,000、3:5,001-10,000、4:10,001-15,000、5:15,001-20,000、6:20,001-30,000、7:30,001-40,000、8:40,001-50,000、9:50,001-100,000、10:100,001-200,000、11:200,001-400,000、12:400,001-600,000、13:600,001-800,000、14:800,001-1,000,000、15:1,000,001以上等15個級距。第16欄差異數調整。.第17欄為合計欄。)，請找出下列：  
A.0050股東集保戶總人數

B.0050股東集保戶總股數

C.0050股東集保戶分級最低持有總股數。

#### bestpractices 提示

需要import numpy以進行資料分析。

#### ICON2輸入與輸出樣本

輸入

20180622,0050,1,15897,3821737,0.45

20180622,0050,2,33908,64668313,7.72

20180622,0050,3,3611,28296918,3.37

20180622,0050,4,940,12101788,1.44

20180622,0050,5,585,10747613,1.28

20180622,0050,6,505,12947960,1.54

20180622,0050,7,201,7169748,0.85

20180622,0050,8,140,6448016,0.76

20180622,0050,9,234,16639220,1.98

20180622,0050,10,87,12309041,1.46

20180622,0050,11,49,14103616,1.68

20180622,0050,12,14,7071165,0.84

20180622,0050,13,4,2785000,0.33

20180622,0050,14,4,3715000,0.44

20180622,0050,15,55,634676865,75.78

20180622,0050,16,1,2000,0.00

20180622,0050,17,56234,837500000,100.00

輸出

0050股東集保戶總人數112469.0

0050股東集保戶總股數1675004000.0

0050股東集保戶分級最低持有總股數2000.0

#### 參考解答



|  |  |
| --- | --- |
| 1  2  3  4  5 | import numpy as np  f = np.genfromtxt('集保戶股權分散表0050.csv', delimiter=',',encoding = 'utf8')  print("0050股東集保戶總人數"+str(f[:,3].sum(axis=0)))  print("0050股東集保戶總股數"+str(f[:,4].sum(axis=0)))  print("0050股東集保戶分級最低持有總股數"+str(f[:,4].min(axis=0))) |

#### ICON2參考解答程式說明

* 1行import所需套件
* 2行運用numpy的genfromtxt讀取'集保戶股權分散表0050.csv'檔案，編碼方式採用"utf-8",檔案中以","分隔，設定為f物件。
* 3行運用sum方法計算與列印0050股東集保戶總人數。
* 4行運用sum方法計算與列印0050股東集保戶總股數
* 5行運用min方法計算與列印0050股東集保戶分級最低持有總股數。

本章作業

1. 請撰寫程式讀取下面資料：  
   datas = [[75,62,85,73,60], [91,53,56,63,65], [71,88,51,69,87],[69,53,87,74,70] ]  
   indexs = ["小林", "小黃", "小陳", "小美"]  
   columns = ["國語", "數學", "英文", "自然", "社會"]  
   利用Pandas的DataFrame資料架構，完成下面要求：  
   A.印出行標題為科目，列題標為個人的所有學生成績  
   B.印出後二位的成績  
   C.以自然遞減排序後列印自然成績  
   D.將"小黃"的"英文"改為80後，列印'小黃的成績')  
   E.將"小林"改為 "小張"；"自然"改為"理化"後，列印全體成績

bestpractices 提示：

本題需要import pandas進行資料分析。

ICON2輸入與輸出樣本

輸入

datas = [[75,62,85,73,60], [91,53,56,63,65], [71,88,51,69,87],[69,53,87,74,70] ]  
indexs = ["小林", "小黃", "小陳", "小美"]  
columns = ["國語", "數學", "英文", "自然", "社會"]

輸出

行標題為科目，列題標為個人的所有學生成績

國語 數學 英文 自然 社會

小林 75 62 85 73 60

小黃 91 53 56 63 65

小陳 71 88 51 69 87

小美 69 53 87 74 70

後二位的成績

國語 數學 英文 自然 社會

小陳 71 88 51 69 87

小美 69 53 87 74 70

以自然遞減排序

小美 74

小林 73

小陳 69

小黃 63

Name: 自然, dtype: int64

小黃的成績

國語 91

數學 53

英文 80

自然 63

社會 65

Name: 小黃, dtype: int64

全體成績

國語 數學 英文 理化 社會

小張 75 62 85 73 60

小黃 91 53 80 63 65

小陳 71 88 51 69 87

小美 69 53 87 74 70

1. 請撰寫程式，利用numpy模組，完成下列：  
   A.在5-16間產生15個隨機正整數。  
   B.將此陣列reshape成3\*5陣列X後，列印X陣列內容  
   C.列印其中最大值/最小值/總和/平均。  
   D.列印四個角落元素  
   E.將之存檔成'EX3-2.txt'  
   F.在5-16間再產生15個隨機正整數，將此陣列reshape成3\*5陣列Y後，列印Y陣列內容。  
   G.將X陣列與Y陣列相加產生Z陣列，將之列印出來。

bestpractices 提示：

本題需要import numpy進行資料分析。

ICON2輸入與輸出樣本



輸入

由Numpy隨機產生的數。

輸出

隨機正整數：[14 14 8 12 11 15 13 14 9 15 7 10 6 13 7]

X矩陣內容：

[[14 14 8 12 11]

[15 13 14 9 15]

[ 7 10 6 13 7]]

最大：15

最小：6

總和：168

平均：11.2

四個角落元素：[[14 11]

[ 7 7]]

Y矩陣內容：

[[14 5 6 11 12]

[10 7 6 14 9]

[15 10 8 8 13]]

Z矩陣內容：

[[28 19 14 23 23]

[25 20 20 23 24]

[22 20 14 21 20]]