Pandas 中的算術運算特性

簡報閱讀 範例與作業 問題討論 學習心得(完成)

重要知識點

- 知道如何使用 DataFrame 中的運算
- 了解 DataFrame 運算的特性
- 知道 DataFrame 中統計與字串的操作
- 了解Pandas的統計函式操作

算術運算是指一般數字間的加、減、乘、除或次方等等的運算， NumPy 陣列的運算有「對齊」、「廣播」和「遮罩」三種向量的運算特性。

統計函式

在生活中常聽到以下情況

1. 台灣平均薪資為XXX
2. 今年指考最高分為XXX
3. 今年台大最低入取分數為XXX
4. 6個標準差的良率

因為數據很多的情況下時常使用敘述統計量來描述數據的分佈與統計量，在資料分析中常拿來對資料做初步的了解。

接下來我們以 pandas 的 DataFrame 資料來做統計函式的介紹

相同欄位的算術運算

在 DataFrame 的算術運算也有「對齊」的特性：

```
1  import pandas as pd
2
3  df1 = pd.DataFrame([[1, 2, 3]])
4  df2 = pd.DataFrame([[1, 1, 1]])
5
6  print(df1 + df2)
7  #     0  1  2
8  # 0  2  3  4
```

在 DataFrame 的「對齊」的特性很嚴格，欄位對不上會產生錯誤的結果：

```
import pandas as pd

df1 = pd.DataFrame([[1, 2, 3]], columns=['a',
df2 = pd.DataFrame([[1, 1, 1]], columns=['c',

print(df1 + df2)
#        a       b      c     d       e
# 0 NaN NaN     4 NaN NaN
```

補充：NaN 是 Not a Number 的縮寫，在程式中泛指無法定義的數值。

DataFrame 也有廣播的特性

在 DataFrame 的算術運算也有「廣播」的特性：

```
import pandas as pd

df1 = pd.DataFrame([[1, 2, 3]])

print(df1 + 1)
#    0  1  2
# 0  2  3  4
```

DataFrame 和 Array 有點不一樣

在 DataFrame 的「廣播」的特性也比較嚴格，只有支援常數的廣播：

```
df = pd.DataFrame([[1, 2, 3]])

print(df + 1)
#    0  1  2
# 0  2  3  4
print(df + pd.DataFrame([1]))
#    0   1   2
# 0  2 NaN NaN
```

```
import numpy as np

a = np.array([[1, 2, 3]])

print(a + 1)
# [[2 3 4]]

print(a + np.array([1]))
# [[2 3 4]]
```

比較和邏輯運算

比較運算是用來判斷數值之間的比較關係，邏輯運算適用於布林值的組合。在 NumPy 當中，我們會利用比較/邏輯運算所產生的布林陣列作為遮罩的條件，從陣列中篩選出滿足條件的元素。

DataFrame 也有遮罩運算

DataFrame 也有沿用遮罩的特性，也可以用來做資料篩選：

```
# a  False  False  True
# b   True   True  True

print(df[df > 2])
#      A    B  C
# a  NaN  NaN  3
# b  4.0  5.0  6
```

```
print(df['A'] > 2)
# a    False
# b     True
# Name: A, dtype: bool

print(df[df['A'] > 2])
#    A  B  C
# b  4  5  6
```

DataFrame 中的排序

除了常見的運算之外，在 DataFrame 當中更多了一些資料的操作方法。第一種我們常用的是排序方法：

```
import pandas as pd

df = pd.DataFrame({
    'col1': ['A', 'a', 'B', 'b'],
    'col2': [2, 1, 9, 8],
})
```

	col1	col2
0	A	2
1	a	1
2	B	9
3	b	8

```
1 df.sort_values(by=['col1'])
```

	col1	col2
0	A	2
2	B	9
1	a	1
3	b	8

```
1 df.sort_values(by=['col1', 'col2'])
```

	col1	col2
0	A	2
2	B	9
1	a	1
3	b	8

---

	col1	col2
2	B	9
3	b	8
0	A	2
1	a	1

DataFrame 的統計方法

df.count()	#非空元素計算
df.min()	#最小值
df.max()	#最大值
df.idxmin()	#最小值的位置
df.idxmax()	#最大值的位置
df.quantile(0.1)	#10%分位數
df.sum()	#求和
df.mean()	#均值
df.median()	#中位數
df.mode()	#眾數
df.var()	#方差

df.std()	#標準差
df.mad()	#平均絕對偏差
df.skew()	#偏度
df.kurt()	#峰度
df.describe()	#一次性輸出多個描述性統計指標

DataFrame 的字串方法

df.str.contains()	對所有欄位的文字進行包含子字串的檢查
df.str.count()	對所有欄位的文字進行計數操作
df.str.split()	對所有欄位的文字進行分割操作

統計函式 平均值mean()

今天都以班上學生國文、英文、數學分數的資料(右表)為例子介紹各個統計函數。
首先是最常使用到的平均值 mean()，pandas 可針對指定欄位算平均值，如果沒指定會對全部欄位算平均值。

```
score_df.math_score.mean()
```

```
score_df.mean()
```

```
60.7
```

```
[70] #全欄位算平均
     score_df.mean()
```

```
math_score      60.7
english_score   62.8
chinese_score   63.5
dtype: float64
```

```
[71] #學生平均分數
     score_df.mean(axis=1)
```

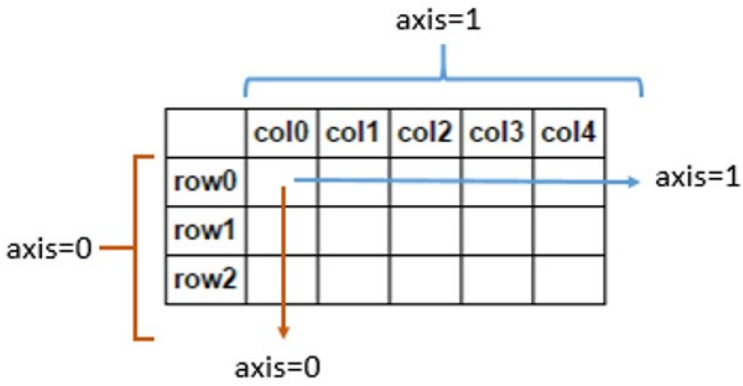
```
student_id
1     66.666667
2     51.666667
3     65.333333
4     76.000000
5     65.000000
6     61.000000
7     64.000000
8     69.333333
9     47.333333
10    57.000000
dtype: float64
```

```
score_df
```

student_id	math_score	english_score	chinese_score
1	50	80	70
2	60	45	50
3	98	43	55
4	70	69	89
5	56	79	60
6	60	68	55
7	45	70	77
8	55	77	76
9	25	57	60
10	88	40	43

如果今天想要算每個學生的總平均分數怎麼辦？

函式。




```
1  score_df.mean(axis=1)
```

```
[69] #指定欄位算平均
     score_df.math_score.mean()
```

```
60.7
```

```
[70] #全欄位算平均
     score_df.mean()
```

```
math_score      60.7
english_score   62.8
chinese_score   63.5
dtype: float64
```

```
[71] #學生平均分數
     score_df.mean(axis=1)
```

```
student_id
1     66.666667
2     51.666667
3     65.333333
4     76.000000
5     65.000000
6     61.000000
7     64.000000
8     69.333333
9     47.333333
10    57.000000
dtype: float64
```

統計函式 加總sum() 個數count()

以下利用加總算出學生3科總分，利用各數計算出應考人數

```
score_df.sum()
```

```
score_df.count()
```

```
[73] #學生3科總分數
     score_df.sum(axis=1)

     student_id
     1     200
     2     155
     3     196
     4     228
     5     195
     6     183
     7     192
     8     208
     9     142
     10    171
     dtype: int64
```

```
[74] #本次各科考試人數
     score_df.count()

     math_score       10
     english_score    10
     chinese_score    10
     dtype: int64
```

統計函式 中位數median()

- 中位數通常使用在有否贏過50%的數據，假如薪資中為數為4萬，超過4萬即為贏過50%的人，反之亦然。
- 中位數：通過把所有觀察值高低排序後找出正中間的一個作為中位數。 如果觀察值有偶數個，則中位數不唯一，通常取最中間的兩個數值的平均數作爲中位數。

以說我數學贏過了全班一半的同學。

```
score_df.median()
```

```
[75] #各科中位數分佈
     score_df.median()

     math_score      58.0
     english_score   68.5
     chinese_score   60.0
     dtype: float64
```

統計函式 百分位數quantile()

- 百分位數使用在觀察數據百分比，最常運用到的是升學分數的百分位數。
- 百分位數：將一組數據從小到大排序，並計算相應的累計百分位，則某一百分位所對應數據的值就稱為這一百分位的百分位數。如果百分位數設定在50%即為中位數。
- 以下計算75%的百分位數，如果我今天國文分數為75分，我可以說我的國文贏過班上75%的同學

```
score_df.quantile(0.75)
```

```
[77] #各科百分位數分佈(75%)
     score_df.quantile(0.75)

     math_score      67.50
     english_score   75.25
     chinese_score   74.50
     Name: 0.75, dtype: float64
```

統計函式 最大值max() 最小值min()

- 其中最小值常常拿來當通過門檻，例如:大學入取分數最低幾分。
- 以下計算全班各科最高與最低分

```
1  score_df.min()
```


```
1  score_df.min()
```


```
[78] #各科最大值
     score_df.max()

     math_score      98
     english_score   80
     chinese_score   89
     dtype: int64
```

```
[79] #各科最小值
     score_df.min()

     math_score      25
     english_score   40
     chinese_score   43
     dtype: int64
```

統計函式 標準差std() 變異數var()

- 標準差：在機率統計中最常使用作為測量一組數值的離散程度之用。一個較大的標準差，代表大部分的數值和其平均值之間差異較大；一個較小的標準差，代表這些數值較接近平均值。
- 變異數：為標準差平方
- 以下計算出標準差，可以發現國文分數標準差比數學分數標準差來的小，所以國文的分散程度比較小，也可以說國文分數較為集中

```
1  score_df.std()
```

```
1  score_df.var()
```

```
math_score      20.854256
english_score   15.418603
chinese_score   14.151953
dtype: float64
```

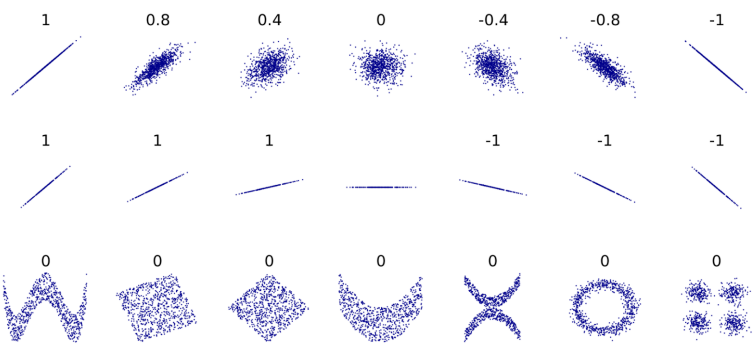
```
[81] #各科變異數
     score_df.var()
```

```
math_score      434.900000
english_score   237.733333
chinese_score   200.277778
dtype: float64
```

統計函式 相關係數corr()

相關係數：皮爾遜積矩相關係數（Pearson product-moment correlation coefficient）用於度量兩個變數X和Y之間的相關程度（線性相依）。在自然科學領域中，該係數廣泛用於度量兩個變數之間的線性相依程度。相關係數的值介於–1與+1之間，即–1≤r≤+1。其性質如下：

1. 當r>0時，表示兩變數正相關，r<0時，兩變數為負相關，r=0時，表示兩變數間無線性相關關係。
2. 一般可按三級劃分：|r|<0.4為低度線性相關；0.4≤|r|<0.7為顯著性相關；0.7≤|r|<1為高度線性相關。



此班學生數學越高分英文越低分，另外國文相對英文相關係數為0.68為正向高度相關，說明此班學生英文越高分國文越高分

```
score_df.corr()
```

```
[82] #各科之間的相關係數
     score_df.corr()
```

	math_score	english_score	chinese_score
math_score	1.000000	-0.532708	-0.314552
english_score	-0.532708	1.000000	0.682340
chinese_score	-0.314552	0.682340	1.000000

自訂義的行或列函式應用 apply()

你有時候可能會覺得說前面的統計函式不足以表達資料的特性，此時你可以使用 apply 做自定義的函式。

像是學校最常使用的加分方式為開根號乘以十，例如：我考 49 分加分過後 √49 x 10= 70，這種方程式沒辦法在統計函式中算出來，需要藉由 apply 中 lambda 的函式達成。

其中 lambda *x*相當於數學式中的 $f(x) = \sqrt{x} \times 10$

```
score_df.apply(lambda x : x**(0.5)*10)
```

student_id	math_score	english_score	chinese_score
1	70.710678	89.442719	83.666003
2	77.459667	67.082039	70.710678
3	98.994949	65.574385	74.161985
4	83.666003	83.066239	94.339811
5	74.833148	88.881944	77.459667
6	77.459667	82.462113	74.161985
7	67.082039	83.666003	87.749644
8	74.161985	87.749644	87.177979
9	50.000000	75.498344	77.459667
10	93.808315	63.245553	65.574385

apply 也適用先前統計函式，可以用下列程式碼看出兩個計算邏輯是等價的。

```
1 score_df.apply(sum,axis=1)
```

```
1 score_df.sum(axis=1)
```

```
student_id
1     200
2     155
3     196
4     228
5     195
6     183
7     192
8     208
9     142
10    171
dtype: int64
```

```
[87] #各科加總
     score_df.sum(axis=1)
```

```
student_id
1     200
2     155
3     196
4     228
5     195
6     183
7     192
8     208
9     142
10    171
dtype: int64
```

知識點回顧

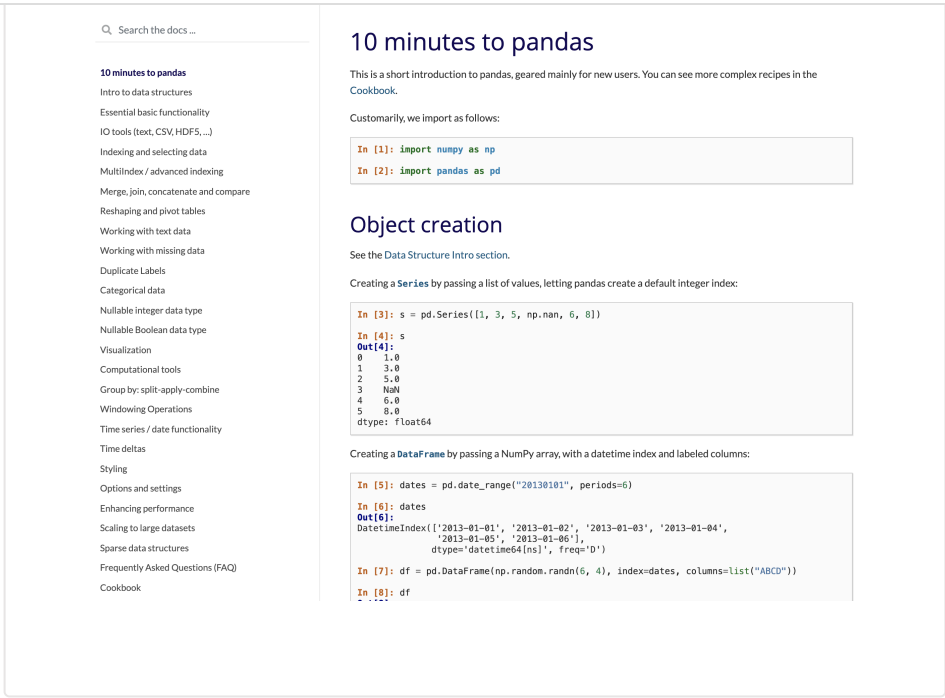
- 知道如何使用 DataFrame 中的運算
- 了解 DataFrame 運算的特性
- 知道 DataFrame 中統計與字串的操作
- 了解Pandas的統計函式操作

參考資料

10 minutes to pandas

網站：pandas

官方所推出的 10 分鐘帶你認識 Pandas 經典入門教材中，提供大量的範例與實作教學。



Search the docs ...

10 minutes to pandas
Intro to data structures
Essential basic functionality
IO tools (text, CSV, HDF5, ...)
Indexing and selecting data
MultiIndex / advanced indexing
Merge, join, concatenate and compare
Reshaping and pivot tables
Working with text data
Working with missing data
Duplicate Labels
Categorical data
Nullable integer data type
Nullable Boolean data type
Visualization
Computational tools
Group by: split-apply-combine
Windowing Operations
Time series / date functionality
Time deltas
Styling
Options and settings
Enhancing performance
Scaling to large datasets
Sparse data structures
Frequently Asked Questions (FAQ)
Cookbook

10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the Cookbook.

Customarily, we import as follows:

```
In [1]: import numpy as np

In [2]: import pandas as pd
```

Object creation

See the Data Structure Intro section.

Creating a Series by passing a list of values, letting pandas create a default integer index:

```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])

In [4]: s
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Creating a DataFrame by passing a NumPy array, with a datetime index and labeled columns:

```
In [5]: dates = pd.date_range("20130101", periods=6)

In [6]: dates
Out[6]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [7]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))

In [8]: df
```

[下一步：閱讀範例與完成作業]