成為初級資料分析師 | Python 與資料科學應用

Pandas 101： 處理表格式資料的 Python 套件

郭耀仁

大綱

- pandas 解決了什麼問題
- pandas 基礎
- Series 與 DataFrame 的基礎操作
- 奧運獎牌排行
- DataFrame 的進階操作
- 美國普查

pandas 解決了什麼問題

隨堂練習：請計算註冊於開曼群島的上市公司股價中位數

https://tw.stock.yahoo.com/d/i/rank.php?t=pri&e=tse&n=100
(https://tw.stock.yahoo.com/d/i/rank.php?t=pri&e=tse&n=100)

In [2]:

```
stock_tickers, stock_names, prices = get_price_rank()
print(stock_tickers)
print(stock_names)
print(prices)
```

```
['3008', '6415', '6409', '2207', '5269', '6669', '1590', '2454', '3406', '147
6', '2059', '2327', '2912', '2330', '3533', '2395', '3563', '8464', '8454', '2
227', '3443', '2049', '6414', '2474', '2231', '8341', '4137', '2357', '2379',
'3665', '3034', '9921', '3661', '1256', '6452', '6230', '3653', '8462', '170
7', '9910', '4551', '8070', '6670', '6666', '2404', '9914', '2707', '2492', '4
943', '8016', '4968', '3413', '6491', '3130', '2345', '4766', '2439', '8422',
'1477', '1537', '4438', '2308', '6213', '2360', '8482', '8480', '5871', '495
8', '1558', '6451', '4572', '6504', '6271', '3044', '4935', '4536', '3376', '9
802', '1723', '1232', '2723', '2383', '5288', '2455', '2228', '8081', '3023',
'6176', '6533', '9941', '3045', '1536', '4912', '6269', '2412', '4763', '361
7', '6278', '6464', '4119']
['大立光', '矽力-KY', '旭隼', '和泰車', '祥碩', '緯穎', '亞德客-KY', '聯發科', '玉晶
光', '儒鴻', '川湖', '國巨', '統一超', '台積電', '嘉澤', '研華', '牧德', '億豐', '富邦
媒', '裕日車', '創意', '上銀', '樺漢', '可成', '為升', '日友', '麗豐-KY', '華碩', '瑞
昱', '貿聯-KY', '聯詠', '巨大', '世芯-KY', '鮮活果汁-KY', '康友-KY', '超眾', '健策',
'柏文', '葡萄王', '豐泰', '智伸科', '長華', '復盛應用', '羅麗芬-KY', '漢唐', '美利達',
'晶華', '華新科', '康控-KY', '矽創', '立積', '京鼎', '晶碩', '一零四', '智邦', '南
寶', '美律', '可寧衛', '聚陽', '廣隆', '廣越', '台達電', '聯茂', '致茂', '商億-KY',
'泰昇-KY', '中租-KY', '臻鼎-KY', '伸興', '訊芯-KY', '駐龍', '南六', '同欣電', '健
鼎', '茂林-KY', '拓凱', '新日興', '鈺齊-KY', '中碳', '大統益', '美食-KY', '台光電',
'豐祥-KY', '全新', '劍麟', '致新', '信邦', '瑞儀', '晶心科', '裕融', '台灣大', '和
大', '聯德控股-KY', '台郡', '中華電', '材料-KY', '碩天', '台表科', '台數科', '旭富']
[4435.0, 888.0, 702.0, 597.0, 567.0, 553.0, 441.0, 421.5, 408.0, 396.0, 356.0,
330.0, 305.5, 305.0, 303.5, 299.5, 290.5, 280.0, 280.0, 277.5, 266.5, 260.5, 2
56.5, 249.5, 247.0, 241.0, 239.0, 230.5, 230.5, 227.5, 223.5, 221.0, 219.0, 21
5.0, 213.5, 212.5, 195.0, 193.0, 191.5, 191.0, 181.0, 178.5, 178.5, 176.5, 17
5.5, 175.0, 170.0, 170.0, 170.0, 169.0, 168.0, 167.5, 165.0, 165.0, 159.0, 15
8.0, 156.5, 156.0, 152.0, 146.0, 145.5, 140.0, 139.5, 139.5, 139.0, 138.0, 13
6.5, 134.5, 134.0, 132.0, 130.5, 130.5, 127.0, 126.5, 125.5, 125.5, 125.0, 12
5.0, 123.5, 123.5, 122.5, 122.0, 121.5, 119.0, 117.5, 117.5, 117.0, 117.0, 11
6.0, 115.0, 114.5, 114.0, 114.0, 113.0, 113.0, 112.5, 112.0, 111.0, 110.0, 10
9.5]
```

In [3]:
```
from statistics import median

ky_prices = [price for stock_name, price in zip(stock_names, prices) if "KY" in st
ock_name]
print(median(ky_prices))
```

```
138.5
```

Python 一直以來都非常適合資料處理，但她的分析能力很薄弱，`pandas` 的開發有助於補足 Python 資料分析的需求，讓使用者能夠在 Python 中執行完整的資料分析流程，而無需切換到 data-centric 的特定語言，如 R。

In [4]:
```
import pandas as pd

df = pd.DataFrame()
df["ticker"] = stock_tickers
df["stock_name"] = stock_names
df["price"] = prices
df.head()
```

Out[4]:

	ticker	stock_name	price
0	3008	大立光	4435.0
1	6415	矽力-KY	888.0
2	6409	旭隼	702.0
3	2207	和泰車	597.0
4	5269	祥碩	567.0

In [5]:
```
df[df["stock_name"].str.contains("KY")]["price"].median()
```

Out[5]:
```
138.5
```

pandas 提供了新的資料結構

- Series 是具備索引的 ndarray
- DataFrame 能完美處理表格式資料（tabular data）
- Panel 能為 DataFrame 加入第三個維度，通常為時間（自 Pandas 0.20.0 版本之後取消了此類別）

pandas 主要的應用場景

- 表格式資料的讀取
- 豐富的資料清理與分析函數
- 視覺化：包裝了常用的 matplotlib.pyplot 圖形

pandas 基礎

什麼是 pandas?

Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more.

安裝與載入 pandas

- 安裝

```
# run in bash shell
pip install pandas
```

- 載入

```
# run in python console
import pandas as pd
```

pandas 的命名源自她的資料結構

- **Pan**el（自 Pandas 0.20.0 版本之後取消了此類別）
- **Da**taFrame
- **S**eries
- Index

pandas 中的 Series

使用 `pd.Series()` 函數創建 Series 類別，Series 從 ndarray 繼承了所有特性，並加上一組 Index。

In [6]:
```
import pandas as pd

movie_ratings = [8.0, 7.3, 8.5, 8.6]
ser = pd.Series(movie_ratings)
print(type(ser))
print(ser)
print(ser[3])
```

```
<class 'pandas.core.series.Series'>
0    8.0
1    7.3
2    8.5
3    8.6
dtype: float64
8.6
```

這使得她不只能夠透過絕對位置來索引，亦可以透過像操作 dict 一般，以鍵（Key）作為選擇索引依據

In [7]:
```
movie_ratings = [8.0, 7.3, 8.5, 8.6]
movie_titles = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity Wa
r", "Avengers: Endgame"]
ser = pd.Series(movie_ratings, index=movie_titles)
print(ser)
print(ser["Avengers: Endgame"])
```

```
The Avengers                8.0
Avengers: Age of Ultron     7.3
Avengers: Infinity War      8.5
Avengers: Endgame           8.6
dtype: float64
8.6
```

可以將 Series 視為一種較為泛用的 ndarray，同時具備 list 和 dict 的特性，以 .index 屬性與 .values 屬性可以將 Series 拆分為 Index 類別與 ndarray

In [8]:
```
movie_ratings = [8.0, 7.3, 8.5, 8.6]
movie_titles = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity Wa
r", "Avengers: Endgame"]
ser = pd.Series(movie_ratings, index=movie_titles)
print(ser.index)
print(ser.values)
print(type(ser.index))
print(type(ser.values))
```

```
Index(['The Avengers', 'Avengers: Age of Ultron', 'Avengers: Infinity War',
       'Avengers: Endgame'],
      dtype='object')
[8.  7.3 8.5 8.6]
<class 'pandas.core.indexes.base.Index'>
<class 'numpy.ndarray'>
```

pandas 中的 DataFrame

我們可以使用 `pd.DataFrame()` 函數創建 DataFrame 類別。

In [9]:
```
movie_ratings = [8.0, 7.3, 8.5, 8.6]
movie_titles = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity Wa
r", "Avengers: Endgame"]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
print(type(df))
df
```

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

Out[9]:

	title	rating
0	The Avengers	8.0
1	Avengers: Age of Ultron	7.3
2	Avengers: Infinity War	8.5
3	Avengers: Endgame	8.6

DataFrame 將多組共享 Index 的 Series 組合為一個具備列索引（row index）與欄標籤（column label）的資料集，我們可以進一步分拆成列索引、欄標籤與 Series

In [10]:
```
movie_ratings = [8.0, 7.3, 8.5, 8.6]
movie_titles = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity Wa
r", "Avengers: Endgame"]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
print(df.index)
print(df.columns)
print(df["title"])
print(df["rating"])
```

```
RangeIndex(start=0, stop=4, step=1)
Index(['title', 'rating'], dtype='object')
0                The Avengers
1    Avengers: Age of Ultron
2     Avengers: Infinity War
3          Avengers: Endgame
Name: title, dtype: object
0    8.0
1    7.3
2    8.5
3    8.6
Name: rating, dtype: float64
```

pandas 中的 Index

不論是 Series 或 DataFrame 物件都包含一個 Index 類別，作為萃取以及更新資料的根據，Index 可以被視為是一種結合了 tuple 的不可變（Immutable）特性以及 set 集合運算特性的資料結構類別，我們可以使用 `pd.Index()` 函數創建出下列的範例

In [11]:

```
pd_index = pd.Index([0, 2, 3, 4])
print(type(pd_index))
print(pd_index)
pd_index[0] = 1
```

```
<class 'pandas.core.indexes.numeric.Int64Index'>
Int64Index([0, 2, 3, 4], dtype='int64')

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-b5275c789408> in <module>
      2 print(type(pd_index))
      3 print(pd_index)
----> 4 pd_index[0] = 1

~/.local/lib/python3.7/site-packages/pandas/core/indexes/base.py in __setitem_
_(self, key, value)
   4258
   4259     def __setitem__(self, key, value):
-> 4260         raise TypeError("Index does not support mutable operations")
   4261
   4262     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

創建後不能更新，Index 也支援 Set 類別的集合運算，可以對兩組 Index 類別（如例子中的五個奇數、四個質數）使用交集、聯集與 XOR（Exclusive OR）

In [12]:
```
odds_index = pd.Index([1, 3, 5, 7, 9])
primes_index = pd.Index([2, 3, 5, 7])
print(odds_index & primes_index) # and
print(odds_index | primes_index) # or
print(odds_index ^ primes_index) # exclusive or
```

```
Int64Index([3, 5, 7], dtype='int64')
Int64Index([1, 2, 3, 5, 7, 9], dtype='int64')
Int64Index([1, 2, 9], dtype='int64')
```

Series 與 DataFrame 的基礎技巧

常見創建 Series 的方式是使用 `pd.Series()` 函數傳入一個「類似清單（list-like）」的物件，包含 list 、tuple 或 ndarray，如果沒有指定另外一個「類似清單」的物件作為索引，Series 會自動以類似 `range()` 函數設定對應長度的索引

In [13]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
ser = pd.Series(movie_ratings)
print(ser)
```

```
0    9.0
1    8.9
2    8.8
3    8.7
dtype: float64
```

在 pd.Series() 函數中加入 index 參數指定對應長度的索引

In [14]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
ser = pd.Series(movie_ratings, index=movie_titles)
print(ser.index)
print(ser.values)
print(ser)
```

```
Index(['The Dark Knight', 'Schindler's List', 'Forrest Gump', 'Inception'], dt
ype='object')
[9.  8.9 8.8 8.7]
The Dark Knight     9.0
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
```

Series 被設計成由一組索引與一組資料所搭建而成的資料結構，因此我們亦可以在 `pd.Series()` 函數中傳入 dict，如此一來字典中的鍵（Keys）會被記錄成為索引、字典中的值（Values）則會被記錄成為陣列中的資料

In [15]:
```
movie_dict = {
    "The Dark Knight": 9.0,
    "Schindler's List": 8.9,
    "Forrest Gump": 8.8,
    "Inception": 8.7
}
ser = pd.Series(movie_dict)
print(movie_dict.keys())
print(movie_dict.values())
print("\n")
print(ser.index)
print(ser.values)
print(ser)
```

```
dict_keys(['The Dark Knight', "Schindler's List", 'Forrest Gump', 'Inceptio
n'])
dict_values([9.0, 8.9, 8.8, 8.7])


Index(['The Dark Knight', 'Schindler's List', 'Forrest Gump', 'Inception'], dt
ype='object')
[9.  8.9 8.8 8.7]
The Dark Knight     9.0
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
```

常見創建 DataFrame 的方式是使用 `pd.DataFrame()` 函數傳入多個「類似清單（list-like）」的物件，包含 list、tuple 或 ndarray，並視需求指定變數名稱或索引；同樣地，若是在沒有指定索引的情況下，DataFrame 會自動以類似 `range()` 函數設定對應長度的索引

In [16]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df
```

Out[16]:

	title	rating
0	The Dark Knight	9.0
1	Schindler's List	8.9
2	Forrest Gump	8.8
3	Inception	8.7

與 Series 相同，DataFrame 被設計成由一組索引與多組類似清單資料所搭建而成的資料結構，因此我們亦可以在 `pd.DataFrame()` 函數中傳入 dict，如此一來字典中的鍵（Keys）會被記錄成為變數名稱、字典中的值（Values）則會被記錄成為陣列中的資料

In [17]:
```
movie_dict = {
    "title": ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"],
    "rating": [9.0, 8.9, 8.8, 8.7]
}
df = pd.DataFrame(movie_dict)
df
```

Out[17]:

	title	rating
0	The Dark Knight	9.0
1	Schindler's List	8.9
2	Forrest Gump	8.8
3	Inception	8.7

實務應用資料框物件時，多數情況都不會是手動輸入資料內容，而是由外部資料源載入

- 以逗號區隔變數的 CSV 文字檔
- 以 JSON 所組成的陣列文字檔
- 試算表
- 資料庫中的表格

使用 pd.read_csv() 函數讀入以逗號區隔變數的 CSV 文字檔

In [18]:
```
df = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/movies.csv")
df
```

Out[18]:

	title	rating
0	The Dark Knight	9.0
1	Schindler's List	8.9
2	Forrest Gump	8.8
3	Inception	8.7

使用 pd.read_json() 函數讀入以 JSON 所組成的陣列文字檔

In [19]:
```
df = pd.read_json("https://python4ds.s3-ap-northeast-1.amazonaws.com/movies.json")
df
```

Out[19]:

	title	rating
0	The Dark Knight	9.0
1	Schindler's List	8.9
2	Forrest Gump	8.8
3	Inception	8.7

使用 pd.read_excel() 函數讀入試算表

In [20]:
```
df = pd.read_excel("https://python4ds.s3-ap-northeast-1.amazonaws.com/movies.xlsx"
)
df
```

Out[20]:

	title	rating
0	The Dark Knight	9.0
1	Schindler's List	8.9
2	Forrest Gump	8.8
3	Inception	8.7

使用 pd.read_sql() 函數讀入資料庫中的表格

In [21]:
```
import sqlite3

# Creating a demo.db database in working directory
conn = sqlite3.connect('demo.db')
# Importing a table
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df.to_sql("movies", index=False, con=conn, if_exists='replace')
# Importing data from demo.movies
query_str = """
SELECT *
    FROM movies
    WHERE rating < 9.0;
"""
pd.read_sql(query_str, con=conn)
```

Out[21]:

	title	rating
0	Schindler's List	8.9
1	Forrest Gump	8.8
2	Inception	8.7

Series 的索引、切割與篩選

In [22]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
ser = pd.Series(movie_ratings, index=movie_titles)
print(ser)
print(ser[0])
print(ser["Forrest Gump"])
```

```
The Dark Knight     9.0
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
9.0
8.8
```

進行資料值的切割時，可以在中括號輸入所需資料的起點與終點，傳入絕對位置則與 Python 慣例一致：不包含終點；若是傳入索引值則會包含終點

In [23]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
ser = pd.Series(movie_ratings, index=movie_titles)
print(ser[1:4])
print(ser["Schindler's List":"Inception"])
```

```
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
```

Series 繼承了 ndarray 的所有特性，因此完全適用華麗索引與布林篩選這兩個便利的技法Series 的索引、切割與篩選

In [24]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
ser = pd.Series(movie_ratings, index=movie_titles)
# Fancy indexing
print(ser[[1, 2, 3]])
print(ser[["Schindler's List", "Forrest Gump", "Inception"]])
# Boolean filtering
print(ser < 9)
print(ser[ser < 9])
```

```
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
The Dark Knight     False
Schindler's List     True
Forrest Gump         True
Inception            True
dtype: bool
Schindler's List    8.9
Forrest Gump        8.8
Inception           8.7
dtype: float64
```

DataFrame 的選擇與篩選

利用 `[COLUMN]` 或 `.COLUMN` 能夠從資料框中選擇出單一或多個變數，成為一個 Series 或者變數欄位較少的資料框子集，實踐 SQL 語法中的 SELECT

In [25]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df["release_year"] = release_years
print(df["title"])
print(df.rating)
df[["title", "release_year"]]
```

```
0     The Dark Knight
1    Schindler's List
2         Forrest Gump
3            Inception
Name: title, dtype: object
0    9.0
1    8.9
2    8.8
3    8.7
Name: rating, dtype: float64
```

Out[25]:

	title	release_year
0	The Dark Knight	2008
1	Schindler's List	1993
2	Forrest Gump	1994
3	Inception	2010

對資料框直接應用布林篩選可以挑出符合條件（條件判斷結果為 True）的觀測值列數，實踐 SQL 語法中的 WHERE

In [26]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df["release_year"] = release_years
print(df["release_year"] > 2000)
df[df["release_year"] > 2000]
```

```
0     True
1    False
2    False
3     True
Name: release_year, dtype: bool
```

Out[26]:

	title	rating	release_year
0	The Dark Knight	9.0	2008
3	Inception	8.7	2010

資料框類別尚有兩個索引語法（Indexer）可以方便使用者在一次的函數呼叫中就完成選擇與篩選兩個任務

- `.loc[ROW_LABEL, COLUMN_LABEL]`：純粹以列、欄標籤為準進行選擇跟篩選
- `.iloc[ROW_INDEX, COLUMN_INDEX]`：純粹以資料的整數位置（integer location）為準進行選擇跟篩選

In [27]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["rating"] = movie_ratings
df["release_year"] = release_years
df.index = movie_titles
df
```

Out[27]:

	rating	release_year
The Dark Knight	9.0	2008
Schindler's List	8.9	1993
Forrest Gump	8.8	1994
Inception	8.7	2010

希望將上映年份在 2000 年之後的 "The Dark Knight" 與 "Inception" 利用索引語法選出來

- 使用 `.loc[]` 時必須傳入列標籤（電影名稱）與欄標籤（評等和上映年份）

In [28]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["rating"] = movie_ratings
df["release_year"] = release_years
df.index = movie_titles
df.loc[["The Dark Knight", "Inception"], ["rating", "release_year"]]
```

Out[28]:

	rating	release_year
The Dark Knight	9.0	2008
Inception	8.7	2010

希望將上映年份在 2000 年之後的 “The Dark Knight” 與 “Inception” 利用索引語法選出來

- 使用 `.iloc[]` 時必須傳入 “The Dark Knight” 與 “Inception” 的列整數位置（第 0 與 3 列）與欄整數位置（第 0 與 1 欄）

In [29]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["rating"] = movie_ratings
df["release_year"] = release_years
df.index = movie_titles
df.iloc[[0, 3], [0, 1]]
```

Out[29]:

	rating	release_year
The Dark Knight	9.0	2008
Inception	8.7	2010

排序

- `df.sort_index()`：依照資料框的列標籤遞增（預設）或遞減排序
- `df.sort_values()`：依照指定的資料框欄標籤遞增（預設）或遞減排序

In [30]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df["release_year"] = release_years
df.sort_index() # default: ascending
```

Out[30]:

	title	rating	release_year
0	The Dark Knight	9.0	2008
1	Schindler's List	8.9	1993
2	Forrest Gump	8.8	1994
3	Inception	8.7	2010

In [31]:
```
df.sort_index(ascending=False)
```

Out[31]:

	title	rating	release_year
3	Inception	8.7	2010
2	Forrest Gump	8.8	1994
1	Schindler's List	8.9	1993
0	The Dark Knight	9.0	2008

In [32]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
df = pd.DataFrame()
df["title"] = movie_titles
df["rating"] = movie_ratings
df["release_year"] = release_years
df.sort_values("title")
```

Out[32]:

	title	rating	release_year
2	Forrest Gump	8.8	1994
3	Inception	8.7	2010
1	Schindler's List	8.9	1993
0	The Dark Knight	9.0	2008

In [33]:
```
df.sort_values("release_year")
```

Out[33]:

	title	rating	release_year
1	Schindler's List	8.9	1993
2	Forrest Gump	8.8	1994
0	The Dark Knight	9.0	2008
3	Inception	8.7	2010

In [34]:
```
df.sort_values("release_year", ascending=False)
```

Out[34]:

	title	rating	release_year
3	Inception	8.7	2010
0	The Dark Knight	9.0	2008
2	Forrest Gump	8.8	1994
1	Schindler's List	8.9	1993

衍生變數

- 簡單運算
- 類別對應類別
- 數值對應類別
- 函數映射

簡單運算

透過 Series 從 ndarray 繼承而來的元素級別運算（element-wise operation）特性即可實踐，像是運用身高、體重這兩個欄位衍生出球員的 BMI

In [35]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
player_profile["bmi"] = player_profile["weightKilograms"] / player_profile["heightMeters"]**2
player_profile[["temporaryDisplayName", "bmi"]].head()
```

Out[35]:

	temporaryDisplayName	bmi
0	Adams, Steven	26.493861
1	Adebayo, Bam	26.742788
2	Adel, Deng	22.449939
3	Aldridge, LaMarcus	26.481885
4	Alexander, Kyle	22.416388

類別對應類別

透過 Series 的 `.map()` 方法來實踐，傳入 dict 作為對應的準則，字典的鍵（Key）為對應前的原始類別，字典的值（Value）為對應後的類別，例如將本來分類較細膩的鋒衛對應為較粗略的前場、後場

In [36]:
```
pos_dict = {
    "G": "Backcourt",
    "F": "Frontcourt",
    "C": "Frontcourt",
    "G-F": "Backcourt",
    "F-C": "Frontcourt",
    "F-G": "Frontcourt",
    "C-F": "Frontcourt"
}
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
print("Pos before mapping:")
player_profile["pos"].value_counts()
```

```
Pos before mapping:
```

Out[36]:
```
G      200
F      186
C       51
G-F     39
F-C     20
F-G     15
C-F     13
Name: pos, dtype: int64
```

In [37]:
```
print("Pos after mapping:")
player_profile["pos_recoded"] = player_profile["pos"].map(pos_dict)
player_profile["pos_recoded"].value_counts()
```

```
Pos after mapping:
```

Out[37]:
```
Frontcourt    285
Backcourt     239
Name: pos_recoded, dtype: int64
```

數值對應類別

透過 `pd.cut()` 函數將數值變數依照指定的門檻值或箱數切分成為類別變數，舉例來說將身高對應為小於等於 2 公尺以及超過 2 公尺兩個類別

In [38]:
```
import numpy as np

player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
player_profile["heightCategory"] = pd.cut(player_profile["heightMeters"], [0, 2, np.Inf], labels=["<= 2m", "> 2m"])
player_profile[["temporaryDisplayName", "heightMeters", "heightCategory"]].head(10)
```

Out[38]:

	temporaryDisplayName	heightMeters	heightCategory
0	Adams, Steven	2.13	> 2m
1	Adebayo, Bam	2.08	> 2m
2	Adel, Deng	2.01	> 2m
3	Aldridge, LaMarcus	2.11	> 2m
4	Alexander, Kyle	2.11	> 2m
5	Alexander-Walker, Nickeil	1.96	<= 2m
6	Allen, Grayson	1.96	<= 2m
7	Allen, Jarrett	2.11	> 2m
8	Allen, Kadeem	1.90	<= 2m
9	Aminu, Al-Farouq	2.06	> 2m

函數映射

透過 `.apply()` 方法來實踐，傳入函數或 lambda 表示式作為映射的準則，例如將本來分類較細膩的鋒衛對應為較粗略的 G、F 與 C

In [39]:
```
def recode_pos(x):
    if x[0] == 'G':
        return 'G'
    elif x[0] == 'F':
        return 'F'
    elif x[0] == 'C':
        return 'C'

player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
player_profile["pos_recoded"] = player_profile["pos"].apply(recode_pos)
print("Pos before applying:")
player_profile["pos"].value_counts()
```

```
Pos before applying:
```

Out[39]:
```
G      200
F      186
C       51
G-F     39
F-C     20
F-G     15
C-F     13
Name: pos, dtype: int64
```

In [40]:
```
print("Pos after applying:")
player_profile["pos_recoded"].value_counts()
```

```
Pos after applying:
```

Out[40]:
```
G    239
F    221
C     64
Name: pos_recoded, dtype: int64
```

摘要

對資料框呼叫常用的摘要方法

- `.count()` 計算列數
- `.mean()` 與 `.median()` 計算平均和中位數
- `.min()` 與 `.max()` 計算最小和最大值
- `.std()` 與 `.var()` 計算標準差和變異數
- `.prod()` 計算乘積
- `.sum()` 計算總和

摘要方法預設作用的維度是資料框的欄位，比方從上映年份、評等與片長三個變數中取出各自的最大值

In [41]:
```
movie_ratings = [9.0, 8.9, 8.8, 8.7]
movie_titles = ["The Dark Knight", "Schindler's List", "Forrest Gump", "Inception"
]
release_years = [2008, 1993, 1994, 2010]
movie_length_mins = [152, 195, 142, 148]
df = pd.DataFrame()
df["rating"] = movie_ratings
df["release_year"] = release_years
df["movie_length_mins"] = movie_length_mins
df.index = movie_titles
df.max()
print("\n")
df.max(axis=1)
```

Out[41]:
```
The Dark Knight     2008.0
Schindler's List    1993.0
Forrest Gump        1994.0
Inception           2010.0
dtype: float64
```

獲取資料的最大值最小值

- `df["col_name"].max()`
- `df["col_name"].min()`

獲取資料最大值最小值的索引值

- `df["col_name"].idxmax()`
- `df["col_name"].idxmin()`

隨堂練習：誰的背號數字最大?

In [42]:
```
import pandas as pd

numbers = [9, 23, 33, 91, 13]
players = ["Ron Harper", "Michael Jordan", "Scottie Pippen", "Dennis Rodman", "Luc Longley"]
df = pd.DataFrame()
df["number"] = numbers
df["player"] = players
df
```

Out[42]:

	number	player
0	9	Ron Harper
1	23	Michael Jordan
2	33	Scottie Pippen
3	91	Dennis Rodman
4	13	Luc Longley

In [43]:
```
max_number = df["number"].max()
df[df["number"] == max_number]["player"][3]
```

Out[43]:
```
'Dennis Rodman'
```

In [44]:
```
df.set_index("player")["number"].idxmax()
```

Out[44]:
```
'Dennis Rodman'
```

隨堂練習：誰的背號數字最小?

In [45]:
```
min_number = df["number"].min()
df[df["number"] == min_number]["player"][0]
```

Out[45]:
```
'Ron Harper'
```

In [46]:
```
df.set_index("player")["number"].idxmin()
```

Out[46]:
```
'Ron Harper'
```

分組摘要

更多的應用情境中我們會指定一些類別變數分組，在分組的前提下對資料框變數呼叫簡單的摘要方法，這樣的操作源自於 SQL 資料庫查詢語言的 GROUP BY 語法

In [47]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
groupby_object = player_profile.groupby("pos")
groupby_object["heightMeters"].mean() # Average height by pos
print("\n")
groupby_object["weightKilograms"].mean() # Average weight by pos
```

Out[47]:
```
pos
C      114.335294
C-F    110.676923
F      101.589730
F-C    110.855000
F-G     95.126667
G       88.896000
G-F     95.717949
Name: weightKilograms, dtype: float64
```

奧運獎牌排行

來源

Coursera (https://www.coursera.org/) 的 Introduction to Data Science in Python (https://www.coursera.org/learn/python-data-analysis/home/welcome) 課程作業

In [48]:
```
import pandas as pd

df = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/olympics.csv",
index_col=0)
```

In [49]:
```
df.head()
```

Out[49]:

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	Gold.2	Silver.2	Bronze.2	Co
Afghanistan	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
Algeria	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
Argentina	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
Armenia	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12
Australasia	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12

隨堂練習：哪個國家贏得的夏季奧運金牌數最多?

```
def answer_one(df):
    """
    這個函數應該回傳一個文字，國家名
    """
    return "答案"
```

In [51]:
```
answer_one(df)
```

Out[51]:
```
'United States'
```

隨堂練習：哪個國家夏季奧運與冬季奧運的金牌數差距數最大?

```
def answer_two(df):
    """
    這個函數應該回傳一個文字，國家名
    """
    return "答案"
```

In [53]:
```
answer_two(df)
```

Out[53]:
```
'United States'
```

隨堂練習：哪個國家夏季奧運與冬季奧運的金牌數差距除以總金牌數的比例最大？（僅考慮至少有一個夏季金牌與一個冬季金牌的國家）

$$\text{Ratio} = \frac{\text{Summer Gold} - \text{Winter Gold}}{\text{Total Gold}}$$

```
def answer_three(df):
    """
    這個函數應該回傳一個文字，國家名
    """
    return "答案"
```

In [55]:
```
answer_three(df)
```

Out[55]:
```
'Bulgaria'
```

隨堂練習：計算 146 個國家的獎牌點數，金牌 3 點、銀牌 2 點、銅牌 1 點。

```
def answer_four(df):
    """
    這個函數應該回傳一個 Series, 長度為 146
    """
    return "答案"
```

In [57]:
```
answer_four(df)[:10]
```

Out[57]:
```
Afghanistan      2
Algeria         27
Argentina      130
Armenia         16
Australasia     22
Australia      923
Austria        569
Azerbaijan      43
Bahamas         24
Bahrain          1
dtype: int64
```

DataFrame 的進階操作

不那麼基礎的 DataFrame 操作

- 處理遺漏值
- 多層索引值
- 轉置
- 合併

Pandas 常用於判斷、刪除和填補遺漏值的方法有四個：

- `.isnull()`
- `.notnull()`
- `.dropna()`
- `.fillna()`

.isnull() 方法能夠輸出一個布林陣列將遺漏值標記為 True，非遺漏值記錄為 False； 而 .notnull() 則是輸出與前者恰恰相反的布林陣列

In [58]:
```
ser = pd.Series([5, None, 6, np.NaN])
print(ser.isnull())
print("\n")
print(ser.notnull())
```

```
0    False
1     True
2    False
3     True
dtype: bool


0     True
1    False
2     True
3    False
dtype: bool
```

.dropna() 方法能夠將資料中遺漏值的部分捨棄，輸出非遺漏值的資料

In [59]:
```
ser = pd.Series([5, None, 6, np.NaN])
print(ser)
print("\n")
ser.dropna()
```

```
0    5.0
1    NaN
2    6.0
3    NaN
dtype: float64
```

Out[59]:
```
0    5.0
2    6.0
dtype: float64
```

對 Series 來說，.dropna() 方法運作的方式非常直觀；但是面對資料框我們無法捨棄單一個資料點，只能夠選擇捨棄一整個列（觀測值）或一整個欄（變數），這時可以傳入參數 axis=0 指定列（預設）、axis=1 指定欄

In [60]:
```
df = pd.DataFrame([
    [1,      np.nan, 7.],
    [2,      5,      8.],
    [np.nan, 6,      9.]
])
df
```

Out[60]:

	0	1	2
0	1.0	NaN	7.0
1	2.0	5.0	8.0
2	NaN	6.0	9.0

In [61]:
```
df.dropna() # default dropping rows with any NaN
```

Out[61]:

	0	1	2
1	2.0	5.0	8.0

In [62]:
```
df.dropna(axis=1) # dropping columns with any NaN
```

Out[62]:

	2
0	7.0
1	8.0
2	9.0

面對遺漏值我們會選擇填補而非捨棄，Pandas 提供了.fillna() 方法，輸出以指定值替代 NaN 的資料

In [63]:
```
ser = pd.Series([5, None, 6, np.NaN])
print(ser)
print("\n")
ser.fillna(5566)
```

```
0    5.0
1    NaN
2    6.0
3    NaN
dtype: float64
```

Out[63]:
```
0       5.0
1    5566.0
2       6.0
3    5566.0
dtype: float64
```

除了以指定值填補遺漏值以外，亦可使用參數 method='ffill' 規則為碰到遺漏值時用前一筆資料填補，同理參數 method='bfill' 規則恰好相反，碰到遺漏值時用後一筆資料填補

In [64]:
```
ser = pd.Series([5, None, 6, np.NaN, 7])
print(ser)
print("\n")
print(ser.fillna(method='ffill'))
print("\n")
print(ser.fillna(method='bfill'))
```

```
0    5.0
1    NaN
2    6.0
3    NaN
4    7.0
dtype: float64


0    5.0
1    5.0
2    6.0
3    6.0
4    7.0
dtype: float64


0    5.0
1    6.0
2    6.0
3    7.0
4    7.0
dtype: float64
```

對資料框應用「分組摘要」的技巧在一些特殊情況之下我們會得到一個索引值比較複雜的 Series 輸出，例如在 .groupby() 方法之中傳入兩個以上的類別變數作為分組依據，這時我們將會得到一種名為 MultiIndex 的類別，所謂的多層索引值

In [65]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
groupby_object = player_profile.groupby(["pos", "country"])
print(groupby_object["heightMeters"].mean()) # Average height by pos and country
print(type(groupby_object["heightMeters"].mean().index))
```

```
pos  country
C    Austria                  2.130000
     Bahamas                  2.160000
     Bosnia and Herzegovina   2.130000
     Canada                   2.060000
     Croatia                  2.135000
                                 ...
G-F  France                   2.006667
     Italy                    1.960000
     Japan                    2.060000
     Turkey                   2.010000
     USA                      1.993214
Name: heightMeters, Length: 88, dtype: float64
<class 'pandas.core.indexes.multi.MultiIndex'>
```

面對具有多層索引值的 Series，數值部分同樣使用 .values 屬性即可拆解，至於索引值的拆解較為複雜，必須像是面對多維度陣列的索引，運用 [m, n, ...] 的方式選取所需資料，例如想知道前述例子中，聯盟中的美國（USA）後衛（G）平均身高，就可以運用 ["G", "USA"] 取值；假如想知道聯盟中的美國（USA）搖擺人（G-F、F-G）平均身高，就運用 [["G-F", "F-G"]][:, "USA"] 取值

In [66]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
groupby_object = player_profile.groupby(["pos", "country"])
ser_w_multi_index = groupby_object["heightMeters"].mean() # Average height by pos and country
print(ser_w_multi_index.values) # values attribute of a multi-index series
print(ser_w_multi_index["G", "USA"]) # average height of USA's guards
print(ser_w_multi_index[["G-F", "F-G"]][:, "USA"]) # average heights of USA's swingmen
```

```
[2.13       2.16       2.13       2.06       2.135      2.06
 2.13333333 2.11       2.11       2.13       2.13       2.13
 2.2        2.17       2.135      2.18       2.08       2.11
 2.11961538 2.16       2.08       2.08       2.08       2.08
 2.13       2.09       2.06       2.08       2.055      2.06
 2.06       2.07       2.082      1.98       2.13       2.00666667
 2.06       2.07       2.06666667 2.06       2.11       2.07
 2.03       2.07       2.06       2.045      2.07       2.06
 2.085      2.06       2.06       2.055      2.03795455 1.995
 2.08       2.13       2.11       2.13       2.21       2.09933333
 2.02       2.03       2.01       1.98636364 1.92       1.93
 1.85       1.94142857 1.96       1.97       1.94       2.03
 1.93       1.98       1.9        1.93       1.9216763  2.03
 1.98       2.045      2.06       1.98       2.01       2.00666667
 1.96       2.06       2.01       1.99321429]
1.9216763005780348
pos
F-G    1.986364
G-F    1.993214
Name: heightMeters, dtype: float64
```

常見的轉置應用是寬表格（Wide Format）與長表格（Long Format）之間的互相轉換

寬表格是比較熟悉的資料框樣式，一列是獨立的觀測值，加入資訊是以增添欄位方式實踐，故得其名為寬表格；長表格是比較陌生的資料框樣式，具有以一欄 key 搭配一欄 value 來紀錄資料的項目與值，加入資訊是以增添列數方式實踐，故得其名為長表格

多數時候我們所使用的資料皆是寬表格的外觀，像是 NBA 球員的基本資料，一列是獨特的一名球員

In [67]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
wide_format = player_profile[["temporaryDisplayName", "heightMeters", "weightKilograms"]]
wide_format.head()
```

Out[67]:

	temporaryDisplayName	heightMeters	weightKilograms
0	Adams, Steven	2.13	120.2
1	Adebayo, Bam	2.08	115.7
2	Adel, Deng	2.01	90.7
3	Aldridge, LaMarcus	2.11	117.9
4	Alexander, Kyle	2.11	99.8

將寬表格的外觀轉換為長表格，表示以一個變數（Key）記錄身高或體重，再以一個變數（Value）記錄身高的高度與體重的重量，我們可以使用 pd.melt() 函數

In [68]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
wide_format = player_profile[["temporaryDisplayName", "heightMeters", "weightKilograms"]]
long_format = pd.melt(wide_format, id_vars="temporaryDisplayName", value_vars=["heightMeters", "weightKilograms"], var_name="key", value_name="value")
long_format.sort_values("temporaryDisplayName").head(10)
```

Out[68]:

	temporaryDisplayName	key	value
0	Adams, Steven	heightMeters	2.13
524	Adams, Steven	weightKilograms	120.20
525	Adebayo, Bam	weightKilograms	115.70
1	Adebayo, Bam	heightMeters	2.08
2	Adel, Deng	heightMeters	2.01
526	Adel, Deng	weightKilograms	90.70
3	Aldridge, LaMarcus	heightMeters	2.11
527	Aldridge, LaMarcus	weightKilograms	117.90
4	Alexander, Kyle	heightMeters	2.11
528	Alexander, Kyle	weightKilograms	99.80

In [69]:
```
long_format.sort_values("temporaryDisplayName").head(10)
```

Out[69]:

	temporaryDisplayName	key	value
0	Adams, Steven	heightMeters	2.13
524	Adams, Steven	weightKilograms	120.20
525	Adebayo, Bam	weightKilograms	115.70
1	Adebayo, Bam	heightMeters	2.08
2	Adel, Deng	heightMeters	2.01
526	Adel, Deng	weightKilograms	90.70
3	Aldridge, LaMarcus	heightMeters	2.11
527	Aldridge, LaMarcus	weightKilograms	117.90
4	Alexander, Kyle	heightMeters	2.11
528	Alexander, Kyle	weightKilograms	99.80

將長表格的外觀轉換為寬表格，會應用到類似分組的操作，以球員姓名作為分組依據，將數值資料樞紐回兩個變數，使用資料框的 .pivot() 方法

In [70]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
wide_format = player_profile[["temporaryDisplayName", "heightMeters", "weightKilograms"]]
long_format = pd.melt(wide_format, id_vars="temporaryDisplayName", value_vars=["heightMeters", "weightKilograms"], var_name="key", value_name="value")
long_format.pivot(index="temporaryDisplayName", columns="key", values="value").head()
```

Out[70]:

key	heightMeters	weightKilograms
temporaryDisplayName		
Adams, Steven	2.13	120.2
Adebayo, Bam	2.08	115.7
Adel, Deng	2.01	90.7
Aldridge, LaMarcus	2.11	117.9
Alexander, Kyle	2.11	99.8

最後稍微整理一下，利用 .reset_index() 以及刪除列索引的名稱，就能將樞紐後的表格回復成與原本一模一樣的寬表格

In [71]:
```
player_profile = pd.read_csv("https://python4ds.s3-ap-northeast-1.amazonaws.com/player_profile.csv")
wide_format = player_profile[["temporaryDisplayName", "heightMeters", "weightKilograms"]]
long_format = pd.melt(wide_format, id_vars="temporaryDisplayName", value_vars=["heightMeters", "weightKilograms"], var_name="key", value_name="value")
wide_format = long_format.pivot(index="temporaryDisplayName", columns="key", values="value").reset_index()
wide_format = wide_format.rename_axis(None, axis=1)
wide_format.head()
```

Out[71]:

	temporaryDisplayName	heightMeters	weightKilograms
0	Adams, Steven	2.13	120.2
1	Adebayo, Bam	2.08	115.7
2	Adel, Deng	2.01	90.7
3	Aldridge, LaMarcus	2.11	117.9
4	Alexander, Kyle	2.11	99.8

Pandas 套件有四種常用函數或方法能夠協助使用者合併不同資料源

- `pd.concat()`
- `df.append()`
- `pd.merge()`
- `df.join()`

簡単合併 pd.concat()

In [72]:
```
upper_df = pd.DataFrame()
lower_df = pd.DataFrame()
upper_df["character"] = ["Rachel Green", "Monica Geller", "Phoebe Buffay"]
upper_df["cast"] = ["Jennifer Aniston", "Courteney Cox", "Lisa Kudrow"]
lower_df["character"] = ["Joey Tribbiani", "Chandler Bing", "Ross Geller"]
lower_df["cast"] = ["Matt LeBlanc", "Matthew Perry", "David Schwimmer"]
```

In [73]:
```
print("Upper df:")
upper_df
print("Lower df:")
lower_df
print("Concatenated vertically:")
pd.concat([upper_df, lower_df]) # axis=0 as default
```

```
Upper df:
Lower df:
Concatenated vertically:
```

Out[73]:

	character	cast
0	Rachel Green	Jennifer Aniston
1	Monica Geller	Courteney Cox
2	Phoebe Buffay	Lisa Kudrow
0	Joey Tribbiani	Matt LeBlanc
1	Chandler Bing	Matthew Perry
2	Ross Geller	David Schwimmer

合併後的資料框具備了重複的列索引，如果希望可以重設列索引，可以在 pd.concat() 函數中加入參數 ignore_index=True

指定參數 axis=1 則為水平合併

In [74]:
```
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["character"] = ["Rachel Green", "Monica Geller", "Phoebe Buffay", "Joey Tr
ibbiani", "Chandler Bing", "Ross Geller"]
right_df["cast"] = ["Jennifer Aniston", "Courteney Cox", "Lisa Kudrow", "Matt LeBl
anc", "Matthew Perry", "David Schwimmer"]
```

In [75]:
```
print("Left df:")
left_df
print("Right df:")
right_df
print("Concatenated horizontally:")
pd.concat([left_df, right_df], axis=1)
```

```
Left df:
Right df:
Concatenated horizontally:
```

Out[75]:

	character	cast
0	Rachel Green	Jennifer Aniston
1	Monica Geller	Courteney Cox
2	Phoebe Buffay	Lisa Kudrow
3	Joey Tribbiani	Matt LeBlanc
4	Chandler Bing	Matthew Perry
5	Ross Geller	David Schwimmer

垂直合併 df.append()

In [76]:
```
upper_df = pd.DataFrame()
lower_df = pd.DataFrame()
upper_df["character"] = ["Rachel Green", "Monica Geller", "Phoebe Buffay"]
upper_df["cast"] = ["Jennifer Aniston", "Courteney Cox", "Lisa Kudrow"]
lower_df["character"] = ["Joey Tribbiani", "Chandler Bing", "Ross Geller"]
lower_df["cast"] = ["Matt LeBlanc", "Matthew Perry", "David Schwimmer"]
```

In [77]:
```
print("Upper df:")
upper_df
print("Lower df:")
lower_df
print("Concatenated vertically using append method:")
upper_df.append(lower_df)
```

```
Upper df:
Lower df:
Concatenated vertically using append method:
```

Out[77]:

	character	cast
0	Rachel Green	Jennifer Aniston
1	Monica Geller	Courteney Cox
2	Phoebe Buffay	Lisa Kudrow
0	Joey Tribbiani	Matt LeBlanc
1	Chandler Bing	Matthew Perry
2	Ross Geller	David Schwimmer

聯結 pd.merge()

在 Pandas 中若想要高效能操作類似關聯式資料庫表格聯結和合併，主要的實踐函數是 pd.merge() ，她沿用關聯式資料庫的正規法則 Relational Algebra，實踐正規法則所規範四種基礎聯結

- 一對一聯結（one-to-one）
- 一對多聯結（one-to-many）
- 多對一聯結（many-to-one）
- 多對多聯結（many-to-many）

In [78]:
```
#一對一聯結 (one-to-one)
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity
War", "Avengers: Endgame"]
left_df["release_year"] = [2012, 2015, 2018, 2019]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "The Avengers"
, "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 8.5, 7.3]
left_df
right_df
pd.merge(left_df, right_df)
```

Out[78]:

	title	release_year	rating
0	The Avengers	2012	8.5
1	Avengers: Age of Ultron	2015	7.3
2	Avengers: Infinity War	2018	8.5
3	Avengers: Endgame	2019	8.6

In [79]:
```
#一對多聯結 (one-to-many)
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers"]
left_df["release_year"] = [2012]
right_df["title"] = ["The Avengers", "The Avengers", "The Avengers"]
right_df["genre"] = ["Action", "Adventure", "Sci-Fi"]
left_df
right_df
pd.merge(left_df, right_df)
```

Out[79]:

	title	release_year	genre
0	The Avengers	2012	Action
1	The Avengers	2012	Adventure
2	The Avengers	2012	Sci-Fi

In [80]:
```
#多對一聯結 (many-to-one)
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "The Avengers", "The Avengers"]
left_df["genre"] = ["Action", "Adventure", "Sci-Fi"]
right_df["title"] = ["The Avengers"]
right_df["release_year"] = [2012]
left_df
right_df
pd.merge(left_df, right_df)
```

Out[80]:

	title	genre	release_year
0	The Avengers	Action	2012
1	The Avengers	Adventure	2012
2	The Avengers	Sci-Fi	2012

In [81]:
```
#多對多聯結（many-to-many）
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "The Avengers", "The Avengers"]
left_df["genre"] = ["Action", "Adventure", "Sci-Fi"]
right_df["title"] = ["The Avengers"]*6
right_df["avengers"] = ["Ironman", "Captain America", "The Hulk", "Thor", "Black W
idow", "Hawkeye"]
left_df
right_df
pd.merge(left_df, right_df)
```

Out[81]:

	title	genre	avengers
0	The Avengers	Action	Ironman
1	The Avengers	Action	Captain America
2	The Avengers	Action	The Hulk
3	The Avengers	Action	Thor
4	The Avengers	Action	Black Widow
5	The Avengers	Action	Hawkeye
6	The Avengers	Adventure	Ironman
7	The Avengers	Adventure	Captain America
8	The Avengers	Adventure	The Hulk
9	The Avengers	Adventure	Thor
10	The Avengers	Adventure	Black Widow
11	The Avengers	Adventure	Hawkeye
12	The Avengers	Sci-Fi	Ironman
13	The Avengers	Sci-Fi	Captain America
14	The Avengers	Sci-Fi	The Hulk
15	The Avengers	Sci-Fi	Thor
16	The Avengers	Sci-Fi	Black Widow
17	The Avengers	Sci-Fi	Hawkeye

加入 left_on 與 right_on 參數指定要用哪些變數進行聯結的對照依據

In [82]:
```
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity
War", "Avengers: Endgame"]
left_df["release_year"] = [2012, 2015, 2018, 2019]
right_df["movie_name"] = ["Avengers: Infinity War", "Avengers: Endgame", "The Aven
gers", "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 8.5, 7.3]
left_df
right_df
pd.merge(left_df, right_df, left_on="title", right_on="movie_name")
```

Out[82]:

	title	release_year	movie_name	rating
0	The Avengers	2012	The Avengers	8.5
1	Avengers: Age of Ultron	2015	Avengers: Age of Ultron	7.3
2	Avengers: Infinity War	2018	Avengers: Infinity War	8.5
3	Avengers: Endgame	2019	Avengers: Endgame	8.6

加入 how 參數則可以指定聯結後的資料框要採用交集（預設）、以左邊資料框存在的觀測值為主、以右邊資料框存在的觀測值為主或聯集

In [83]:
```
#交集（預設）
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity
War"]
left_df["release_year"] = [2012, 2015, 2018]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "Avengers: Age
of Ultron"]
right_df["rating"] = [8.5, 8.6, 7.3]
left_df
right_df
print("Inner join:")
pd.merge(left_df, right_df)
```

```
Inner join:
```

Out[83]:

	title	release_year	rating
0	Avengers: Age of Ultron	2015	7.3
1	Avengers: Infinity War	2018	8.5

In [84]:
```
#以左邊資料框存在的觀測值為主
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War"]
left_df["release_year"] = [2012, 2015, 2018]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 7.3]
left_df
right_df
print("Left join:")
pd.merge(left_df, right_df, how="left")
```

```
Left join:
```

Out[84]:

	title	release_year	rating
0	The Avengers	2012	NaN
1	Avengers: Age of Ultron	2015	7.3
2	Avengers: Infinity War	2018	8.5

In [85]:
```
#以右邊資料框存在的觀測值為主
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War"]
left_df["release_year"] = [2012, 2015, 2018]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 7.3]
left_df
right_df
print("Right join:")
pd.merge(left_df, right_df, how="right")
```

```
Right join:
```

Out[85]:

	title	release_year	rating
0	Avengers: Age of Ultron	2015.0	7.3
1	Avengers: Infinity War	2018.0	8.5
2	Avengers: Endgame	NaN	8.6

In [86]:
```
#聯集
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War"]
left_df["release_year"] = [2012, 2015, 2018]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 7.3]
left_df
right_df
print("Outer join:")
pd.merge(left_df, right_df, how="outer")
```

```
Outer join:
```

Out[86]:

	title	release_year	rating
0	The Avengers	2012.0	NaN
1	Avengers: Age of Ultron	2015.0	7.3
2	Avengers: Infinity War	2018.0	8.5
3	Avengers: Endgame	NaN	8.6

用列索引聯結 df.join()

In [87]:
```
left_df = pd.DataFrame()
right_df = pd.DataFrame()
left_df["title"] = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity
War", "Avengers: Endgame"]
left_df["release_year"] = [2012, 2015, 2018, 2019]
right_df["title"] = ["Avengers: Infinity War", "Avengers: Endgame", "The Avengers"
, "Avengers: Age of Ultron"]
right_df["rating"] = [8.5, 8.6, 8.5, 7.3]
left_df = left_df.set_index("title")
right_df = right_df.set_index("title")
left_df
right_df
left_df.join(right_df)
```

Out[87]:

	release_year	rating
title		
The Avengers	2012	8.5
Avengers: Age of Ultron	2015	7.3
Avengers: Infinity War	2018	8.5
Avengers: Endgame	2019	8.6

美國普查

來源

Coursera (https://www.coursera.org/) 的 Introduction to Data Science in Python (https://www.coursera.org/learn/python-data-analysis/home/welcome) 課程作業

In [88]:
```
import pandas as pd

census_df = pd.read_csv('https://storage.googleapis.com/py_ml_datasets/census.csv'
)
census_df.shape
```

Out[88]:
```
(3193, 100)
```

In [89]:
```
census_df.head()
```

Out[89]:

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATESBASE2010	POPESTIMATE20
0	40	3	6	1	0	Alabama	Alabama	4779736	4780127	4785161
1	50	3	6	1	1	Alabama	Autauga County	54571	54571	54660
2	50	3	6	1	3	Alabama	Baldwin County	182265	182265	183193
3	50	3	6	1	5	Alabama	Barbour County	27457	27457	27341
4	50	3	6	1	7	Alabama	Bibb County	22915	22919	22861

5 rows × 100 columns

隨堂練習：哪個州（state）的郡（county）數最多?

```
def answer_one(df):
    """
    這個函數應該回傳一個文字，州名
    """
    return "答案"
```

In [91]:
```
answer_one(census_df)
```

Out[91]:
```
'Texas'
```

隨堂練習：僅考慮每州（state）人口最多的三個郡（county）計算人口總和（CENSUS2010POP），哪三個州總和數最多？（請注意 SUMLEV 變數）

```
def answer_two(df):
    """
    這個函數應該回傳一個清單，三個州名
    """
    return "答案"
```

In [93]:
```
answer_two(census_df)
```

Out[93]:
```
['California', 'Texas', 'Illinois']
```

隨堂練習：哪個郡（county）在 2010-2015 期間人口改變數量最高？（POPESTIMATE2010:POPESTIMATE2015 這六個變數）

提示：如果 6 年的人口數分別為 120, 80, 105, 100, 130, 120 則人口改變數量為 130-80 = 50

```
def answer_three(df):
    """
    這個函數應該回傳一個文字，郡名
    """
    return "答案"
```

In [95]:
```
answer_three(census_df)
```

Out[95]:
```
'Harris County'
```

隨堂練習：篩選出屬於 REGION 1 或 2、開頭名稱為 Washington 並且 POPESTIMATE2015 大於 POPESTIMATE2014 的郡（county）

```
def answer_four():
    """
    這個函數應該回傳一個 DataFrame, 外型為 5x2,
    變數名稱為 ['STNAME', 'CTYNAME'],
    索引值由小到大排列
    """
    return "答案"
```

In [97]:
```
answer_four(census_df)
```

Out[97]:

	STNAME	CTYNAME
0	Iowa	Washington County
1	Minnesota	Washington County
2	Pennsylvania	Washington County
3	Rhode Island	Washington County
4	Wisconsin	Washington County

延伸閱讀

pandas: powerful Python data analysis toolkit (http://pandas.pydata.org/pandas-docs/stable/)

作業

擷取 Avengers: Endgame (2019) 的上映日期列表，改以 `pandas` 回答最多的上映日期為哪一天?

In [99]:
```
answer_one()
```

Out[99]:
```
release_date
12 July 2019          2
2 September 2019      1
22 April 2019         1
23 April 2019         1
24 April 2019        33
25 April 2019        22
26 April 2019        14
26 July 2019          1
28 April 2019         1
28 June 2019          3
29 April 2019         1
29 June 2019          1
4 July 2019           1
Name: country, dtype: int64
```

In [100]:
```
answer_one().idxmax()
```

Out[100]:
```
'24 April 2019'
```

擷取 Avengers: Endgame (2019) 的上映日期列表，改以 `pandas` 回答有幾個國家在那天上映?

In [102]:
```
answer_two()
```

Out[102]:
```
'共有 33 個國家在 24 April 2019 上映 Avengers: Endgame (2019)'
```