# Python 資料清理與儲存實務

David Chiu

Big Data – Pipeline, Technologies & Role
Ref:http:goo.gl/Mm8

Infer-ability
Model
Context
Connectedness
Variety
Variability
Velocity
Volume

資料儲存與清理

Infer, Predict, Recommend & Visualize

Contextualize, Model & Reason

De-complexify, Transform, Analyze & Network

D3.js, Dashboards, web apps

Store

R, Hive, Pig, python, Java, Mahout

Collect

Hadoop

Technologies

ETL, Storm, Scribe, Flume,...

Machine Learning

Analytics

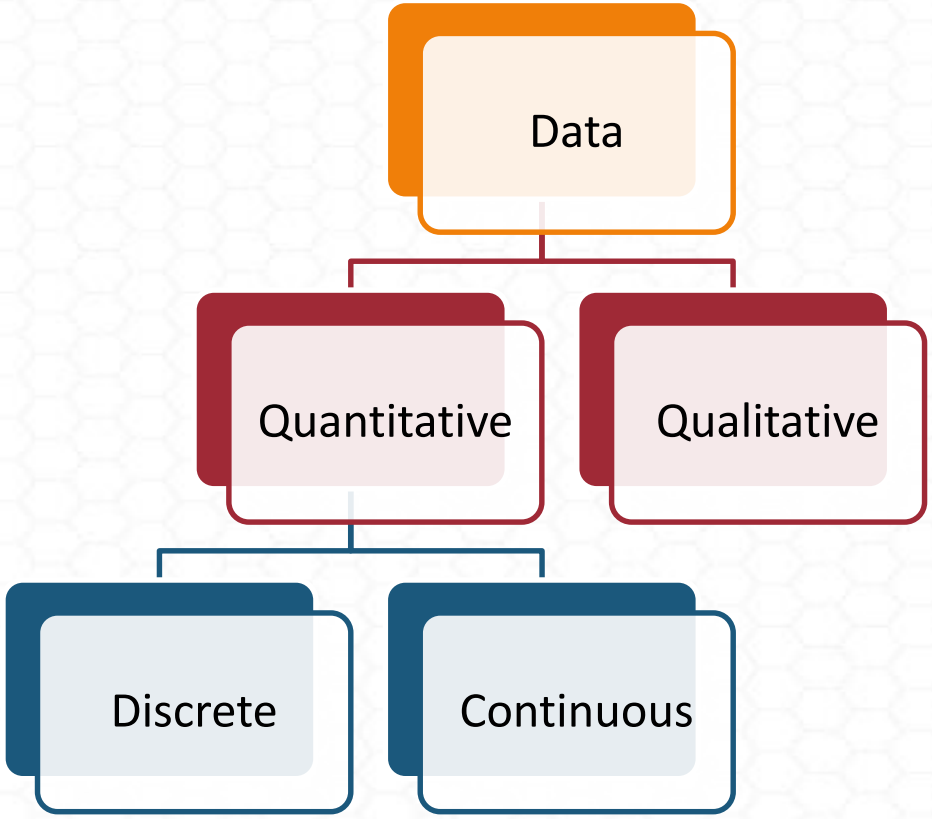
Roles

Data Architecture / Management

**Data Scientist**

# 數據的種類

# 數據的種類



# 數據型態

- 定性資料 (Qualitative or Categorical Data)
  - 敘述特性或種類
  - e.g. 住在哪一區，哪個種族的人
- 定量資料 (Quantitative or Numerical Data)
  - 可以被計數或測量
  - e.g. 身高、消費金額

# 定量資料類型

- 離散數據 (Discrete Data)
  - 只能用自然數或整數單位計算
  - 只能按計量單位數計數，可由一般計數方法取得
  - e.g. 員工人數
- 連續資料 (Continuous Data)
  - 一定區間內可以任意取值的數據，其數值是連續不斷的，相鄰兩個數值可取無限個數值
  - 其數值只能用測量或計量的方法取得
  - e.g. 零件的規格尺寸

# 結構化vs半結構化vs非結構化數據

- 結構化資料
  - 每筆資料都有固定的欄位、固定的格式，方便程式進行後續取用與分析
  - 例如：資料庫
- 半結構化資料
  - 資料介於結構化資料與非結構化資料之間
  - 資料具有欄位，也可以依據欄位來進行查找，使用方便，但每筆資料的欄位可能不一致
  - 例如：XML, JSON
- 非結構化資料
  - 沒有固定的格式，必須整理以後才能存取
  - 沒有格式的文字、網頁數據

# 結構化資料

- 資料有固定的欄位與格式
  - 例如：資料庫表格中所存放的資料

| id | title | content | time | view_cnt | category |
|---|---|---|---|---|---|
| 56 | Uni-girls成立經紀公司　將選拔5位… | 統一7-ELEVEn獅隊旗下Uni… | 2016-06-17 16:11:00 | 0 | 體育 |
| 57 | 販售工業用劣質油　盞好企業判6… | 盞好企業負責人吳容合被… | 2016-06-17 16:10:00 | 0 | 社會 |
| 58 | 義大回嗆中職會長：聯盟應深切檢討 | 中華職棒會長吳志揚今天… | 2016-06-17 16:09:00 | 0 | 體育 |
| 59 | 【就職滿月】蔡英文滿意度4成7　… | 總統蔡英文就職即將滿月… | 2016-06-17 16:08:00 | 0 | 政治 |
| 60 | 【有片】素珠之亂有續集　洪家賣… | (更新：新增影片)自稱「… | 2016-06-17 16:08:00 | 36312 | 社會 |
| 61 | 昔日老毛鬥爭工具　「中央專案組… | 香港銅鑼灣書店店長林榮… | 2016-06-17 16:08:00 | 0 | 國際 |
| 62 | 【有片】北市府給雪隧建議　林聰… | (更新:新增動新聞)台北市… | 2016-06-17 16:07:00 | 5067 | 政治 |

- 可以下SQL 處理與撈取資料
  - select title, content from newsmain;

# 半結構化資料 - XML

- 可以使用欄位存取資料內容
- 欄位不固定，例如Mary 就少了age的欄位
- 可以彈性的存放各種欄位格式的資料

```
<users>
  <user>
    <name>QOO</name>
    <gender>M</gender>
    <age>12</age>
  </user>
  <user>
    <name>Mary</name>
    <gender>F</gender>
  </user>
</users>
```

# 半結構化資料 - JSON

- 如同XML可以使用欄位存取資料內容
- 使用Key:Value存放資料
- 不用宣告欄位的結尾，可以比XML更快更有效傳輸資料

```
[
    user:{
        name:QOO,
        gender:M,
        age:12,
    },
    user:{
        name:Mary,
        gender:F
    }
]
```

# 資料清理實務

# 資料科學家主要工作內容

***“80% 都在做加總與平均”***

## 工作內容

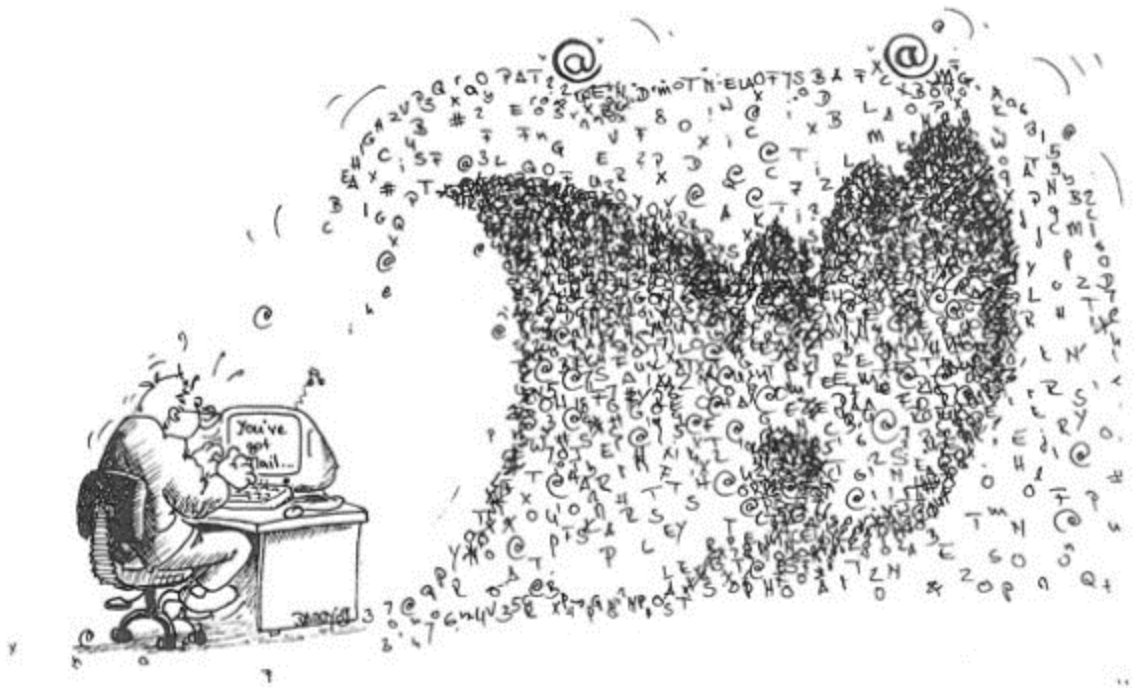
1. 資料處理 (Data Munging)
2. 資料分析 (Data Analysis)
3. 詮釋結果 (Interpret Result)

80% 時間會花在資料清理上

# 資料是雜亂無章的

**80% 的時間都在清理資料**

- 資料篩選
- 偵測遺失值
- 補齊遺失值
- 資料轉換
- 處理時間格式資料
- 重塑資料
- 學習正規運算式
- …



# 讓兩個List 元素進行相乘?

```
a = [1, 3, 5, 7, 9]
b = [2, 4, 6, 8, 10]

print zip(a, b)
[(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]

c = [i*j for i, j in zip(a, b)]
print c
[2, 12, 30, 56, 90]
```

如何讓兩個List 裡面的元素相乘?

# 可以使用 NumPy

```
import numpy as np
na = np.array(a)
nb = np.array(b)

nc = na * nb

print nc
[ 2 12 30 56 90]
```

# Numpy

- Numeric Python
  - Python 數學運算套件
  - N維陣列物件
  - 多種數學運算函式
    - (linear algebra, Fourier transform … etc)
  - 可整合 C/C++ 和 Fortran

# Pandas

- Python for Data Analysis
  - 源自於R
  - Table-Like 格式
  - 提供高效能、簡易使用的資料格式(Data Frame)讓使用者可以快速操作及分析資料

# 一維陣列

```
na = np.array([1, 2, 3, 4, 5])
na
array([1, 2, 3, 4, 5])

na[0]
1

na[1:3]
array([2, 3])

na[na > 3]
array([4, 5])
```

**跟一般List 操作類似**

# 二維陣列

```
na = np.array([[1, 2, 3, 4, 5],
[6, 7, 8, 9, 10]])
na
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])

na[0, 0]
1

na[0, :]
array([1, 2, 3, 4, 5])
```

```
na[:, 0]
array([1, 6])

na.T
array([[ 1,  6],
       [ 2,  7],
       [ 3,  8],
       [ 4,  9],
       [ 5, 10]])
```

# 如何增添資料的列與行名?

```
na = np.array([['frank', 'M', 29], ['mary', 'F', 23], ['tom', 'M',
35], ['ted', 'M', 33], ['jean', 'F', 21], ['lisa', 'F', 20]])
na
```

```
array([['frank', 'M', '29'],
       ['mary', 'F', '23'],
       ['tom', 'M', '35'],
       ['ted', 'M', '33'],
       ['jean', 'F', '21'],
       ['lisa', 'F', '20']],
      dtype='|S5')
```

# 產生結構化資訊

```
import numpy as np
na = np.array([['name', 'gender', 'age'], ['frank', 'M', 29], ['mary', 'F', 23], ['tom', 'M', 35],
['ted', 'M', 33], ['jean', 'F', 21], ['lisa', 'F', 20]])
na
```

```
array([['name', 'gender', 'age'],
       ['frank', 'M', '29'],
       ['mary', 'F', '23'],
       ['tom', 'M', '35'],
       ['ted', 'M', '33'],
       ['jean', 'F', '21'],
       ['lisa', 'F', '20']],
      dtype='|S6')
```

看起來好像有點麻煩

# 產生Pandas DataFrame

```
import pandas as pd
df = pd.DataFrame([['frank', 'M', 29], ['mary', 'F', 23], ['tom', 'M', 35],
['ted', 'M', 33], ['jean', 'F', 21], ['lisa', 'F', 20]])
df
```

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | frank | M | 29 |
| 1 | mary | F | 23 |
| 2 | tom | M | 35 |
| 3 | ted | M | 33 |
| 4 | jean | F | 21 |
| 5 | lisa | F | 20 |

# 新增欄位名稱

```
df.columns=['name', 'gender', 'age']
df
```

| | name | gender | age |
|---|---|---|---|
| 0 | frank | M | 29 |
| 1 | mary | F | 23 |
| 2 | tom | M | 35 |
| 3 | ted | M | 33 |
| 4 | jean | F | 21 |
| 5 | lisa | F | 20 |

# Pandas Series

# 序列(Series)

- 類似Array, List 的一維物件
- 每個Series 都可以透過其索引(Index) 進行存取
- 預設Series 會以 0 到 Series 長度做為索引編號

# 序列(Series)

```
s = pd.Series([21000, 18900, 18000])
s
```

```
0    21000
1    18900
2    18000
dtype: int64
```

# 自訂序列索引

```
pd.Series([21000, 18900, 18000], index = ['Iphone', 'Edge S7', 'HTC M1'])
```

```
Iphone      21000
Edge S7     18900
HTC M1      18000
dtype: int64
```

# 利用字典建立序列

```
s = pd.Series({'Iphone':21000,'Edge S7':18900,'HTC M1':18000})
```

```
Edge S7     18900
HTC M1      18000
Iphone      21000
dtype: int64
```

# 篩選序列 (1/2)

- 根據位置篩選

s[2]

- 篩選多筆資料

s[1:3]

```
HTC M1     14400
Iphone     20000
dtype: int64
```

# 篩選序列 (2/2)

■ 篩選出Iphone

```
s['Iphone']
```

■ 篩選出Iphone 與 Edge S7

```
s[['Iphone', 'Edge S7']]
```

■ 根據價格篩選

```
s[s<20000]
```

# 篩選是如何完成的?

- 產生Boolean真假表

```
cheap = s < 20000
cheap
```

```
Edge S7      True
HTC M1       True
Iphone      False
dtype: bool
```

```
s[cheap]
```

```
Edge S7    18900
HTC M1     18000
dtype: int64
```

# 可以修改序列中的值

■ 修改單一值

```
s['Iphone'] = 20000
s['Iphone']
```

■ 修改符合條件的值 (20000 元以下打八折)

```
s[s<20000] = s[s<20000] * 0.8
```

# 檢查序列中是否有該值存在

- 檢查是否有Iphone

```
'Iphone' in s
```

- 檢查是否有小米

```
'mi' in s
```

# 數學計算

■ 打八折價格

```
s * 0.8
```

■ 使用numpy 做進階計算(e.g. 開根號)

```
import numpy as np
np.sqrt(s)
```

# 產生簡易統計

- 取最大值

```
s.max()
```

- 取平均值

```
s.mean()
```

- 取敘述性統計

```
s.describe()
```

# 序列相加

- 將兩個序列相加

```
S2 = s[['Iphone', 'HTC M1']] + s[['mi', 'HTC M1']]
```

```
HTC M1     28800
Iphone       NaN
mi           NaN
dtype: float64
```

數值加NaN 會變成 NaN

# 檢查序列是否有空值

- 檢查非空值資料

s2.notnull()

- 檢查空值資料

s2.isnull()

# Pandas DataFrame

# DataFrame

- 類似Excel 的表格結構(Tabular Data Structure)
- 包含欄與列的資料，可以根據欄與列運算元據
- 類似R 的 DataFrame
- 可以想像是序列(Series)的集合

# 建立DataFrame (1/2)

```
df = pd.DataFrame([['frank', 'M', 29], ['mary', 'F', 23], ['tom', 'M', 35],
['ted', 'M', 33], ['jean', 'F', 21], ['lisa', 'F', 20]])
df.columns = ['name', 'gender', 'age']
df
```

| | name | gender | age |
|---|---|---|---|
| 0 | frank | M | 29 |
| 1 | mary | F | 23 |
| 2 | tom | M | 35 |
| 3 | ted | M | 33 |
| 4 | jean | F | 21 |
| 5 | lisa | F | 20 |

6 rows × 3 columns

# 建立DataFrame (2/2)

■ 在DataFrame 裡面增加Columns 描述

```
df = pd.DataFrame([['frank', 'M', 29], ['mary', 'F', 23], ['tom', 'M', 35], ['ted',
'M', 33], ['jean', 'F', 21], ['lisa', 'F', 20]], columns = ['name', 'gender', 'age'])
```

■ 使用字典建立DataFrame

```
df = pd.DataFrame([{'name':'frank', 'gender':'M', 'age':29}, \
                   {'name':'mary',  'gender':'F', 'age':23}, \
                   {'name':'tom',   'gender':'M', 'age':35}, \
                   {'name':'ted',   'gender':'M', 'age':33}, \
                   {'name':'jean',  'gender':'F', 'age':21}, \
                   {'name':'lisa',  'gender':'F', 'age':20}])
```

# 取樣前/後數筆資料

■ 取前幾筆資料

df.head()

■ 取後幾筆資料

df.tail()

# 取得DataFrame 基本資訊

■ 取得基本敘述

df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 5
Data columns (total 3 columns):
age       6 non-null int64
gender    6 non-null object
name      6 non-null object
dtypes: int64(1), object(2)
memory usage: 192.0+ bytes
```

■ 取得基本統計

df.describe()

| | age |
|---|---|
| count | 6.000000 |
| mean | 26.833333 |
| std | 6.400521 |
| min | 20.000000 |
| 25% | 21.500000 |
| 50% | 26.000000 |
| 75% | 32.000000 |
| max | 35.000000 |

8 rows × 1 columns

■ 取得基本型態

df.dtypes

```
age         int64
gender     object
name       object
dtype: object
```

# 存取元素與切割 (Indexing & Slicing)

df['name']

```
0    frank
1     mary
2      tom
3      ted
4     jean
5     lisa
Name: name, dtype: object
```

df[['name', 'age']]

| | name | age |
|---|---|---|
| 0 | frank | 29 |
| 1 | mary | 23 |
| 2 | tom | 35 |
| 3 | ted | 33 |
| 4 | jean | 21 |
| 5 | lisa | 20 |

6 rows × 2 columns

# 存取元素與切割 (Indexing & Slicing)

df['gender'] == 'M'

```
0     True
1    False
2     True
3     True
4    False
5    False
Name: gender, dtype: bool
```

df[df['gender'] == 'M']

| | name | gender | age |
|---|---|---|---|
| 0 | frank | M | 29 |
| 2 | tom | M | 35 |
| 3 | ted | M | 33 |

3 rows × 3 columns

# 存取元素與切割 (Indexing & Slicing)

- 使用 & 取條件交集

```
df[(df['gender'] == 'M') & (df['age'] >= 30) ]
```

- 使用 | 取條件聯集

```
df[(df['gender'] == 'M') | (df['age'] >= 30) ]
```

# 新增/刪除欄位

■ 新增欄位

```
df['employee'] = True
```

| | age | gender | name | employee |
|---|---|---|---|---|
| 0 | 29 | M | frank | True |
| 1 | 23 | F | mary | True |
| 2 | 35 | M | tom | True |
| 3 | 33 | M | ted | True |
| 4 | 21 | F | jean | True |
| 5 | 20 | F | lisa | True |

■ 刪除欄位

```
del df['employee']
```

OR

```
df = df.drop('employee', 1)
```

# 新增/刪除欄位

■ 新增第六列

```
df.loc[6] = {'age':20,'gender':'F','name':'qoo'}
```

OR

```
df.append(pd.DataFrame([{'age':20,'gender':'F','name':'qoo'}]),
ignore_index=True)
```

■ 刪除第六列

```
df.drop(6)
```

# 設定新的索引

```
df['userid'] = range(101,107)
df.set_index('userid', inplace=True)
```

| | age | gender | name |
|---|---|---|---|
| **userid** | | | |
| **101** | 29 | M | frank |
| **102** | 23 | F | mary |
| **103** | 35 | M | tom |
| **104** | 33 | M | ted |
| **105** | 21 | F | jean |
| **106** | 20 | F | lisa |

# 根據位置取值

- iloc 可以根據位置取值

df.iloc[1]

```
age          23
gender        F
name       mary
Name: 102, dtype: object
```

df.iloc[[1,3,5]]

| | age | gender | name |
|---|---|---|---|
| userid | | | |
| 102 | 23 | F | mary |
| 104 | 33 | M | ted |
| 106 | 20 | F | lisa |

# 處理缺失值

# 缺失值 (Missing Value)

- 資料中有特定或一個範圍的值是不完全的
- 缺失值可能會導致資料分析時產生偏誤的推論
- 缺失值可能來自機械的缺失或是人為的缺失
  - 機械缺失 e.g. 機械故障，導致資料無法被完整保存
  - 人為缺失 e.g. 受訪者拒絕透露部分資訊

# 建立一含有缺失值的Data Frame

```
import pandas as pd
import numpy  as np
df = pd.DataFrame([\
                   ['frank', 'M',    np.nan], \
                   ['mary' , np.nan, np.nan], \
                   ['tom'  , 'M',    35], \
                   ['ted'  , 'M',    33], \
                   ['jean' , np.nan, 21], \
                   ['lisa' , 'F',    20]])
df.columns = ['name', 'gender', 'age']
df
```

可以使用np.nan 代表缺失值

# 檢查序列是否有缺失值

- 檢查非缺失值資料

```
df['gender'].notnull()
```

- 檢查缺失值資料

```
df['gender'].isnull()
```

# 檢查欄位或Data Frame 是否含有缺失值

■ 檢查欄位是否含有缺失值

```
df.name.isnull().values.any()
```

■ 檢查Data Frame 是否還有缺失值

```
df.isnull().values.any()
```

# 計算缺失值的數量

- 檢查欄位缺失值的數量

```
df.isnull().sum()
```

- 計算所有缺失值的數量

```
df.isnull().sum().sum()
```

# 處理缺失值

- 捨棄缺失值
  - 當缺失值占資料比例很低時
- 使用平均數、中位數、眾數等敘述性統計補齊缺失值
- 使用內插法補齊缺失值
  - 如果欄位資料成線性規律

# 捨棄缺失值

■ 舍棄含有任意缺失值的行

```
df.dropna()
```

■ 捨棄所有欄位都含有缺失值的行

```
df.dropna(how= 'all')
```

■ 捨棄超過兩欄缺失值的行

```
df.dropna(thresh=2)
```

# 捨棄含有缺失值的列

- 增加一包含缺失值的列

```
df['employee'] = np.nan
```

- 捨棄皆為缺失值的列

```
df.dropna(axis = 1, how='all')
```

# 填補缺失值

■ 用0填補缺失值

```
df.fillna(0)
```

■ 用平均數缺失值

```
df['age'].fillna(df["age"].mean())
```

■ 用各性別年齡平均填缺失值

```
df['age'].fillna(df.groupby("gender")["age"].transform(
"mean"))
```

# 向前/向後填值

■ 向後填補缺失值

```
df.fillna(method='pad')
```

■ 向前填補缺失值

```
df.fillna(method='bfill', limit=2)
```

| 參數 | 動作 |
| --- | --- |
| pad / ffill | 往後填值 |
| bfill / backfill | 往前填值 |

# 使用內插法填補缺失值

```
df2 = pd.DataFrame([[1, 870],\
                    [2, 900],\
                    [np.nan, np.nan],\
                    [4, 950],\
                    [5,1080],\
                    [6,1200]])
df2.columns = ['time', 'val']
df2.interpolate()
```

■ 可參閱

https://docs.scipy.org/doc/scipy/reference/interpolate.html#univariate-interpolation

# 數據轉換

# 數據轉換

*“如何清洗、轉換數據”*

| | detail_url | search_date | title | address | floor_info | price | layout | building_area | building_use | latitude | longitude |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | https://rent.591.com.tw/rent-detail-1050934.html | 2017-05-14 | 套房出租-近東方學院(六~八月可入住) | 高雄市湖內區民生街 | 3F/3F | 4,500元/月 | NaN | 6.0 | 透天厝/獨立套房 | 22.878436 | 120.250810 |
| 1 | https://rent.591.com.tw/rent-detail-1008046.html | 2017-05-14 | 好房子【聰明的你、妳快來租吧】 | 新北市蘆洲區長安街 | 4F/5F | 7,500元/月 | NaN | 10.0 | 透天厝/獨立套房 | 25.081558 | 121.461868 |
| 2 | https://rent.591.com.tw/rent-detail-1586266.html | 2017-05-12 | 陽台大套房(獨立洗衣機)近頂溪捷運站 | 新北市永和區中山路一段 | 8F/12F | 11,000元/月 | NaN | 8.0 | 電梯大樓/分租套房 | 25.010293 | 121.511328 |

# 讀取資料

```
import pandas
df = pandas.read_csv('rent_591_sample.csv', encoding='utf-8',
index_col=0)
df.head(3)
```

觀看前三行資料

| | detail_url | search_date | title | address | floor_info | price | layout | building_area | building_use | latitude | longitude |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | https://rent.591.com.tw/rent-detail-1050934.html | 2017-05-14 | 套房出租-近東方學院(六~八月可入住) | 高雄市湖內區民生街 | 3F/3F | 4,500元/月 | NaN | 6.0 | 透天厝/獨立套房 | 22.878436 | 120.250810 |
| 1 | https://rent.591.com.tw/rent-detail-1008046.html | 2017-05-14 | 好房子【聰明的你、妳快來租吧】 | 新北市蘆洲區長安街 | 4F/5F | 7,500元/月 | NaN | 10.0 | 透天厝/獨立套房 | 25.081558 | 121.461868 |
| 2 | https://rent.591.com.tw/rent-detail-1586266.html | 2017-05-12 | 陽台大套房(獨立洗衣機)近頂溪捷運站 | 新北市永和區中山路一段 | 8F/12F | 11,000元/月 | NaN | 8.0 | 電梯大樓/分租套房 | 25.010293 | 121.511328 |

# 向量化計算

■ 計算新價格

```
df['building_area'] / 0.3025
```

■ 使用Numpy 計算新價格

```
import numpy as np
np.sqrt(df['building_area'])
```

■ 合併兩字串

```
df['address'] +  '-' + df['price']
```

■ 將新計算的均價存入Data Frame

```
df['square_feet'] = df['building_area'] / 0.3025
```

# Apply, Map, ApplyMap

- Map
  將函數套用到Series 上的每個元素
- Apply
  將函數套用到DataFrame 上的行與列
- ApplyMap
  將函式套用到DataFrame上的每個元素(elementwise)

# Map

■ 移除價格中的,

```
def normalizePrice(ele):
    res = int(ele.replace(',', ''))
    return res
df['price'].map(normalizePrice)
```

■ 使用匿名函式

```
df['price'].map(lambda e: int(e.replace(',', '')))
```

# Apply

```
df = pandas.DataFrame(\
[\
   [60,70,50],\
   [80,79,68],\
   [63,66,82]], \
  columns = ['First', 'Second', 'Third'])

df.apply(lambda e: e.max() - e.min(), axis=1)
```

根據行 axis = 0
根據列 axis = 1

# ApplyMap

- 將所有缺失值(NaN)替代成 ‘-’

```
import numpy as np
df.applymap(lambda e: '-' if pandas.isnull(e)
else e)
```

# 處理時間格式資料

# 處理時間格式資料

■ 列印出現在時間

```
from datetime import datetime
current_time = datetime.now()
```

■ 將時間轉換成字串

```
current_time.strftime('%Y-%m-%d')
```

■ 將字串轉換為時間

```
datetime.strptime('2017-08-21', '%Y-%m-%d')
```

可參考
https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

# 時間回溯

■ 往前回溯一天

```
from datetime import timedelta
current_time - timedelta(days = 1)
```

■ 往前回溯10天

```
for i in range(1,10):
    dt = current_time - timedelta(days = i)
    print(dt.strftime('%Y-%m-%d'))
```

# 轉換UNIX 時間

■ 將datetime 轉換為 UNIX timestamp

```
from time import mktime
mktime(current_time.timetuple())
```

■ 將 UNIX timestamp 轉換為 datetime

```
datetime.fromtimestamp(1492859823)
```

# 在pandas 轉換時間

在pandas 轉換時間

```
import pandas
df['search_date'] =
pandas.to_datetime(df['search_date'], format = '%Y-%m-
%d')
```

# 虛擬變量 (DUMMY VARIABLE)

# 虛擬變量 (Dummy Variable)

| | title | floor_info | price | layout | building_area | building_use |
|---|---|---|---|---|---|---|
| 0 | 套房出租-近東方學院(六~八月可入住) | 3F/3F | 4,500元/月 | NaN | 6.0 | 透天厝/獨立套房 |
| 1 | 好房子【聰明的你、妳快來租吧】 | 4F/5F | 7,500元/月 | NaN | 10.0 | 透天厝/獨立套房 |
| 2 | 陽台大套房(獨立洗衣機)近頂溪捷運站 | 8F/12F | 11,000元/月 | NaN | 8.0 | 電梯大樓/分租套房 |

如何創造虛擬變數?

# 虛擬變數 (Dummy Variable)

```
#建立虛擬變數
pandas.get_dummies(df['building_use'])

#合併虛擬變數與原DataFrame
df = pandas.concat([df, pandas.get_dummies(df['building_use'])], axis=1)

#捨棄原有欄位
df.drop('building_use', axis=1)
```

# 建立樞紐分析表(pivot_table)

```
df2 = df.pivot_table(index='search_date', columns='building_use',
values='price', aggfunc=sum)
df2.head()

df3 = df.pivot_table(index='building_use', columns='search_date',
values='price', aggfunc=sum)
df3.head()
```

也可以使用df2.T 做轉置

# 長寬表格轉換 (stack & unstack)

■ 建立多索引的樞紐分析表

```
df['btype'] = df['building_use'].map(lambda e: str(e).split('/')[0])
df['buse']  = df['building_use'].map(lambda e: str(e).split('/')[1] if len(str(e).split('/'))> 1 else None)
df_multi_idx = df.pivot_table(index=['btype', 'buse'], columns='search_date', values='price', aggfunc=sum)
```

■ 轉換為寬表格

```
df_wide = df_multi_idx.unstack()
df_wide.head()
```

■ 轉換為長表格

```
df_long = df_wide.stack()
df_long.head()
```

# 正規運算式

# 正規運算式

起始　大寫字母A到Z　數字　數字位元數　結束

^[A-Z] \d {9}$

- 代表連續

或

[0-9]

# 正規運算式 (符號與意義)

| 符號 | 意義 |
| --- | --- |
| . | 比對除換行外的任意字元 |
| ^ | 比對字串開始 |
| $ | 比對字串結尾 |
| * | 比對0個或多個由前面正規運算式定義的片段，貪婪方式 |
| + | 比對1個或多個由前面的正規運算式定義的片段，貪婪方式 |
| ? | 比對0個或1個由前面的規則運算式定義的片段，貪婪方式 |
| *? , +?, ?? | 非貪婪版本的 *, +, 和 ? (盡可能少的比對) |
| [...] | 比對方括弧內內的字元集中的任意一個字元 |
| (...) | 比對括號內的運算式，也表示一個群組 |
| (?P<id>...) | 類似 (...), 但該組同時得到一個 id,可以在後面的模式中引用 |

# 正規運算式範例 (1)

```
import re
m = re.match(r"(\w+)@(\w+)", "david@largidata.com")
print(m.groups())

m = re.match(r"(\w+)@([a-z.]+)",
"david@largitdata.com")
print(m.groups())

m = re.match(r"(\d+)\.(\d+)", "1999.5")
print(m.groups())
```

# 正規運算式範例(2)

剖析姓名

```
m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "David Chiu")
print(m.group('first_name'), m.group('last_name'))
```

剖析Linux 指令

```
str1 = 'scp file.txt root@10.0.0.1:./'
m=re.search('^scp ([\w\.]+) (\w+)@([\w\.]+):(.+)',str1)
if m:
    print(m.group(1), m.group(2), m.group(3), m.group(4))
```

# 在DataFrame 上使用正規表達法

■ 從layout用正規表達法抽取資訊

```
df2 = df[df['layout'].notnull()]
df2[['bedroom', 'living_room', 'bathroom']] = df['layout'].str.extract('(\d+)房(\d+)廳(\d+)衛', expand=False)
df2[['layout','bedroom', 'living_room', 'bathroom']].head()
```

| | layout | bedroom | living_room | bathroom |
|---|---|---|---|---|
| 0 | 4房2廳2衛 | 4 | 2 | 2 |
| 1 | 2房1廳1衛 | 2 | 1 | 1 |
| 2 | 1房0廳1衛 | 1 | 0 | 1 |
| 3 | 3房2廳2衛 | 3 | 2 | 2 |
| 4 | 3房2廳2衛 | 3 | 2 | 2 |

# PANDAS IO

# 使用Pandas 寫入與讀取檔案

- 可以使用Pandas 的內建函式讀取/寫入資料
  - csv
  - Excel
  - Clipboard
  - sql
  - json
  - Html

# 讀取文字檔案

```
m_cols = ['Time','Action','User','Product', 'Quantity','Price']
orders = pd.read_csv('dataset/purchase_order.tab', sep='\t',
          parse_dates={'Dates': [0]},names=m_cols, encoding='utf-8')
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54772 entries, 0 to 54771
Data columns (total 6 columns):
Dates       54772 non-null datetime64[ns]
Action      54772 non-null object
User        54772 non-null object
Product     54772 non-null object
Quantity    54772 non-null int64
Price       54721 non-null float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(3)
memory usage: 2.9+ MB
```

# 寫入文字資料

■ 寫進csv 檔案

```
orders.to_csv('orders.csv')
```

■ 寫進Excel 檔案

```
orders.to_excel('orders.xlsx')
```

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | | Dates | Action | User | Product | Quantity | Price |
| 2 | 0 | 2015-07-01 00:00:01 | order | U3126227 | P00069445 | 1 | 1069 |
| 3 | 1 | 2015-07-01 00:00:03 | order | U2390123 | P00060180 | 1 | 1680 |
| 4 | 2 | 2015-07-01 00:00:19 | order | U1000769 | P00022679 | 1 | 285 |
| 5 | 3 | 2015-07-01 00:01:10 | order | U2963285 | P00161442 | 1 | 550 |
| 6 | 4 | 2015-07-01 00:01:36 | order | U3008845 | P00145169 | 1 | 249 |
| 7 | 5 | 2015-07-01 00:01:48 | order | U4510503 | P00041342 | 1 | 1780 |

# 轉換成JSON 資料

- 直接轉換成JSON API

orders.to_json()


```
'{"Dates":{"0":1435708801000,"1":1435708803000,"2":1435708819000,"3":1435708870
000,"4":1435708896000},"Action":{"0":"order","1":"order","2":"order","3":"orde
r","4":"order"},"User":{"0":"U312622727","1":"U239012343","2":"U1000769737
3","3":"U296328517","4":"U300884570"},"Product":{"0":"P0006944501","1":"P000601
8073","2":"P0002267974","3":"P0016144236","4":"P0014516980122"},"Quantity":
{"0":1,"1":1,"2":1,"3":1,"4":1},"Price":{"0":1069.0,"1":1680.0,"2":285.0,"3":55
0.0,"4":249.0}}'
```

# 從剪貼簿貼入DataFrame

```
df = pd.read_clipboard()
df
```

| B | C | D | E | F | G |
|---|---|---|---|---|---|
| Dates | Action | User | Product | Quantity | Price |
| 2015-07-01 00:00:01 | order | U3126227 | P0006944 | 1 | 1069 |
| 2015-07-01 00:00:03 | order | U2390123 | P0006018 | 1 | 1680 |
| 2015-07-01 00:00:19 | order | U1000769 | P0002267 | 1 | 285 |
| 2015-07-01 00:01:10 | order | U2963285 | P0016144 | 1 | 550 |
| 2015-07-01 00:01:36 | order | U3008845 | P0014516 | 1 | 249 |
| 2015-07-01 00:01:48 | order | U4510503 | P0004134 | 1 | 1780 |
| 2015-07-01 00:01:58 | order | U4651240 | P0012662 | 1 | 488 |
| 2015-07-01 00:02:18 | order | U4030013 | P0004401 | 1 | 276 |
| 2015-07-01 00:03:12 | order | U4651232 | P0000319 | 1 | 2199 |
| 2015-07-01 00:03:45 | order | U3118085 | P0006944 | 1 | 1069 |

# Pandas Aggregation

# 計算每個產品購買均價

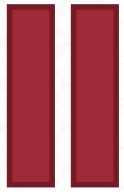
- 計算產品均價

```
orders[orders['Product'] == 'P0000143511']['Price'].mean()
```

如果今天產品有很多個?
是否要根據不同產品不斷取平均？

# 使用SQL統計資料

```
SELECT Product, AVERAGE(Price) FROM orders
GROUP BY Product
```



```
orders.groupby('Product')['Price'].mean().head()
```

# 將產品價格排序後取前三名

```
orders.groupby('Product')['Price'].mean().sort_values(ascending=False)[0:3]
```

```
Product
P0000143511    438888
P0000143500    438888
P0006584093    320000
Name: Price, dtype: float64
```

# 抓出購買商品大戶

```
orders['Total_price'] = orders['Quantity'] * orders['Price']
orders.groupby('User')['Total_price'].sum().sort_values(ascending=False)[0:3]
```

```
User
U166708333      2942744
U10120098943    1451117
U142809250       747550
Name: Total_price, dtype: float64
```

# 讀取Views檔案

```
m_cols = ['Time','Action','User','Product']
views= pd.read_csv('dataset/purchase_view.tab', sep='\t',
          parse_dates={'Dates': [0]},names=m_cols, encoding='utf-8')
views.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1191486 entries, 0 to 1191485
Data columns (total 4 columns):
Dates      1191486 non-null datetime64[ns]
Action     1191486 non-null object
User       1191486 non-null object
Product    1191486 non-null object
dtypes: datetime64[ns](1), object(3)
memory usage: 45.5+ MB
```

# 使用Merge合併Views 與 Orders

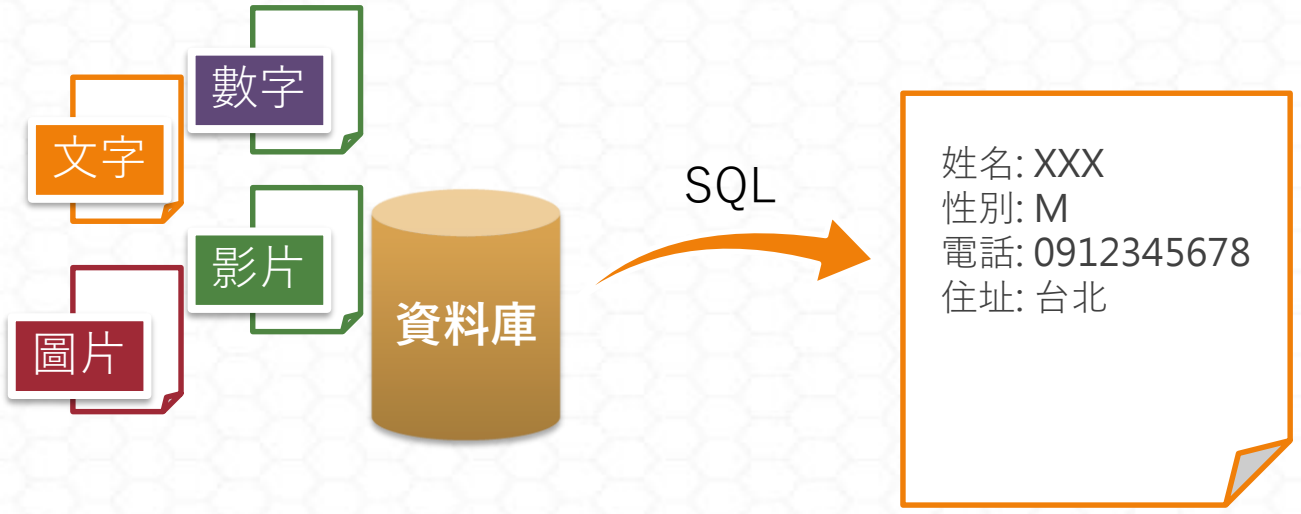
```
orders_cnt = orders.groupby(['User','Product'])['Product'].size().reset_index(name='buys')
views_cnt = views.groupby(['User','Product'])['Product'].size().reset_index(name='views')
merge_df = pd.merge(orders_cnt, views_cnt, on=['User','Product'], how='right')
```

| | User | Product | buys | views |
|---|---|---|---|---|
| 0 | U10000044910 | P0018800250 | 1 | 2 |
| 1 | U10000056953 | P0013834251 | 1 | 7 |
| 2 | U10000065461 | P0023597022 | 1 | 1 |
| 3 | U10000092625 | P0023729451 | 1 | 14 |
| 4 | U10000092625 | P0024236730 | 1 | 4 |

# 資料儲存

# 資料庫

- 將資料以結構化方式做存儲，讓使用者可以透過結構化查詢語言(Structured Query Language, 簡稱SQL)快速取用及維護資料



# 關聯式資料庫

- 安全存儲、管理資料
  - 有效管理磁片上的資料
- 保持資料的一致性
  - ACID 四原則
- 可以透過標準模型整合資料
  - 使用SQL 運算資料

# 使用Python 連結Teradata

```
import teradata

udaExec = teradata.UdaExec (appName="HelloWorld", version="1.0",
        logConsole=False)

session = udaExec.connect(method="odbc", system="tdprod",
        username="xxx", password="xxx");

for row in session.execute("SELECT GetQueryBand()"):
    print(row)
```

需要先安裝 Teradata
! pip install teradata

# 透過SQLite 做資料查詢

```
import teradata

udaExec = teradata.UdaExec ()

with udaExec.connect("${dataSourceName}") as session:
    for row in session.execute("SELECT * FROM ${table}"):
        print(row)
```

# 使用 Cursor

```
import teradata
udaExec = teradata.UdaExec()

with udaExec.connect("${dataSourceName}") as session:
    with session.cursor() as cursor:
    for row in cursor.execute("SELECT * from ${tableName}"):
            session.execute("DELETE FROM ${tableName} WHERE id = ?", (row.id, )):
```

# Execute v.s. Excutemany

- execute

```
session.execute("""INSERT INTO employee (id, firstName, lastName, dob)
                   VALUES (?, ?, ?, ?)""", (1,"James", "Kirk", "2233-03-22"))
```

- executemany

```
session.executemany("""INSERT INTO employee (id, firstName, lastName, dob)
                       VALUES (?, ?, ?, ?)""",
                    ((1,"James", "Kirk", "2233-03-22"),
                     (2,"Jean-Luc", "Picard", "2305-07-13")),
                    batch=True)
```

# Stored Procedures

```
results = session.callproc("MyProcedure",
(teradata.InOutParam("inputValue", "inoutVar1"),
teradata.OutParam(), teradata.OutParam("outVar2",
dataType="PERIOD")))
print(results.inoutVar1)
print(results.outVar1)
```

# Transactions

```
import teradata
udaExec = teradata.UdaExec()
with udaExec.connect("${dataSourceName}", autoCommit=False)
as session:
    session.execute("CREATE TABLE ${tableName} (${columns})")
    session.commit()
```

# Sqlalchemy-teradata

```
from sqlalchemy import create_engine
td_engine = create_engine('teradata://dbuser:passw@host.com')
sql = 'select * from dbc.usersV'
result = td_engine.execute(sql)
```

需要先安裝 sqlalchemy-teradata

```
! pip install sqlalchemy-teradata
```

# THANK YOU