

Python 資料整理

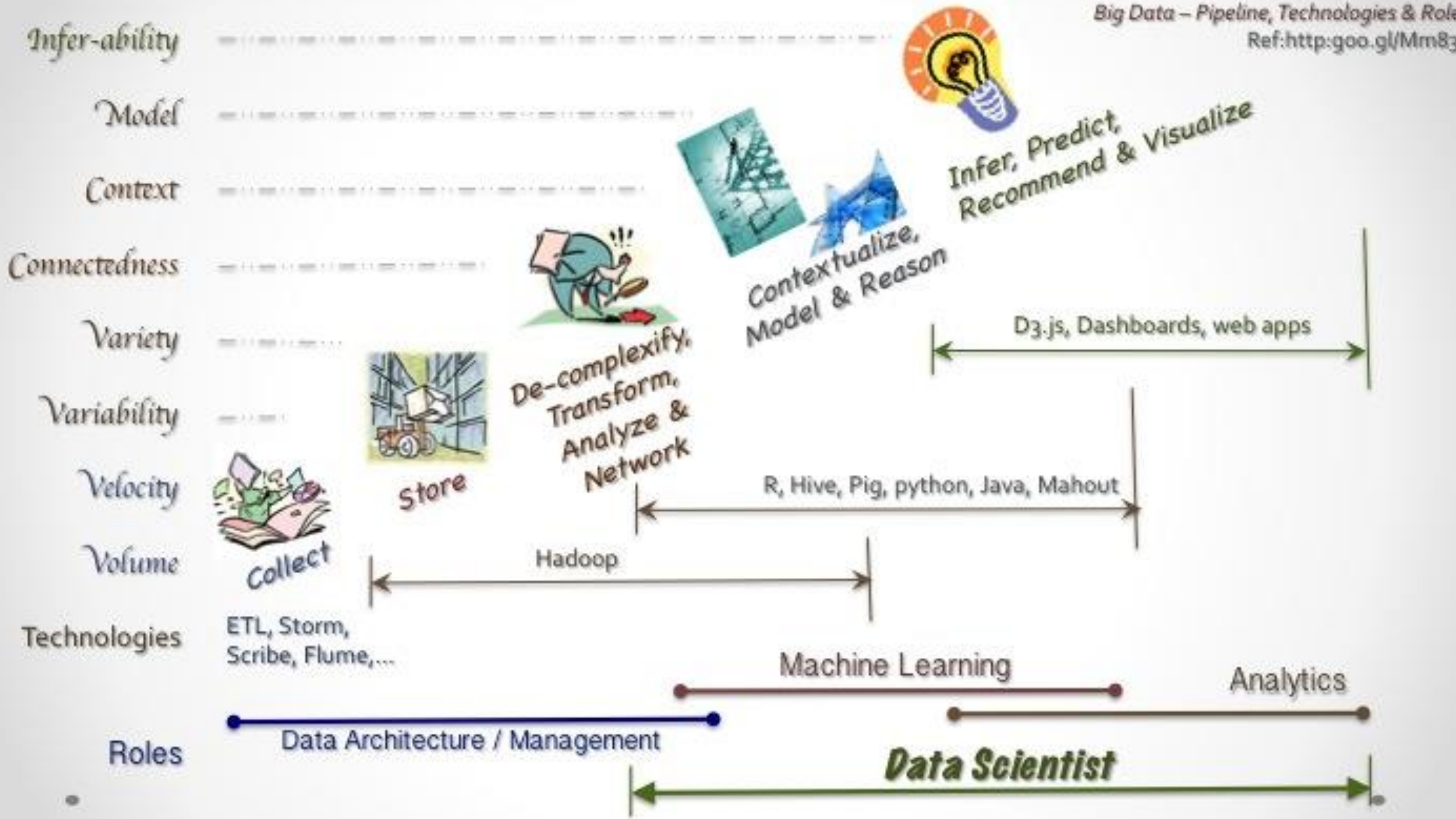
David Chiu

課程資料

■ 所有課程補充資料、投影片皆位於

□ <https://github.com/ywchiu/ctbcpy>

資料整理實務



資料科學家主要工作內容

“80% 都在做加總與平均”

工作內容

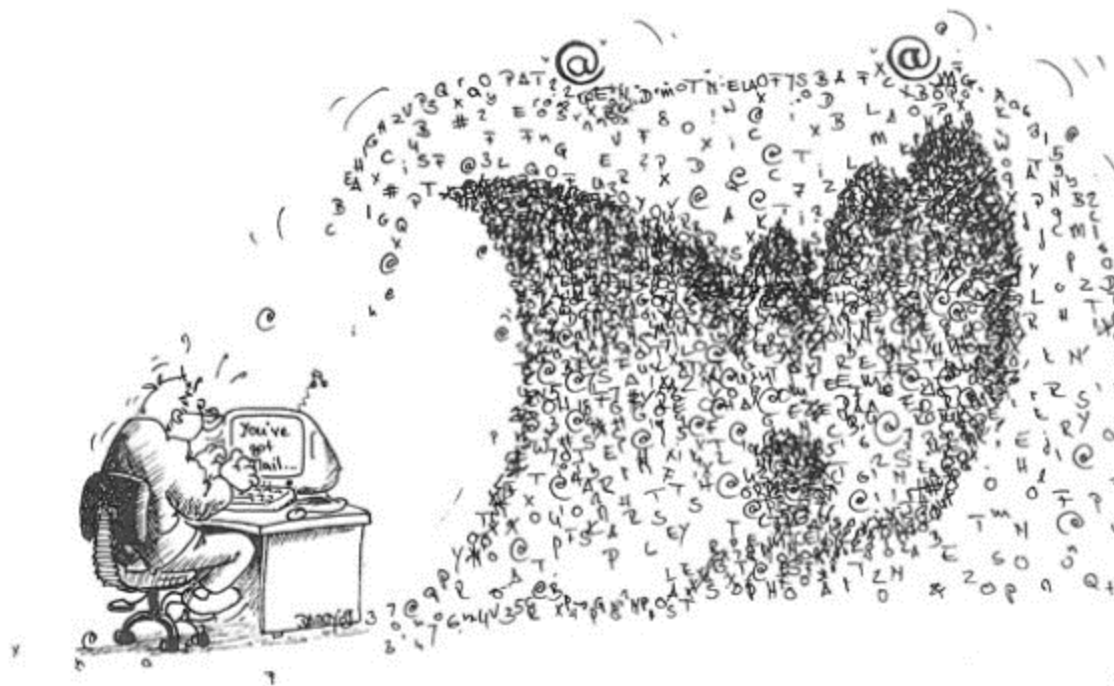
1. 資料處理 (Data Munging)
2. 資料分析 (Data Analysis)
3. 詮釋結果 (Interpret Result)

80% 時間會花在資料清理上

資料整理的步驟

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
- ...



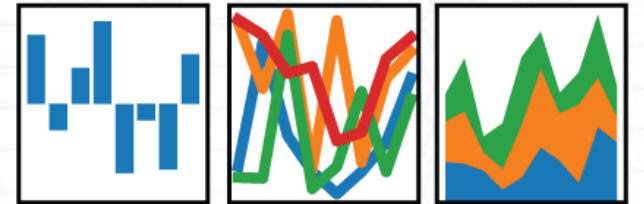
資料整理與計算工具

Numpy



Pandas

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Numpy

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



建立陣列

#建立一維陣列

```
import numpy as np  
my_list = [1,2,3]  
np.array(my_list)
```

#建立二維陣列

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

內建方法

#產生數列

```
np.arange(0,10)
```

```
np.arange(0,11,2)
```

#產生0與1

```
np.zeros(3)
```

```
np.zeros((5,5))
```

```
np.ones(3)
```

```
np.ones((3,3))
```

#根據給定範圍產生數列

```
np.linspace(0,10,3)
```

```
np.linspace(0,10,50)
```

#建立對角矩陣

```
np.eye(4)
```

#產生隨機數列

```
np.random.rand(2)
```

```
np.random.rand(5,5)
```

```
np.random.randn(2)
```

```
np.random.randn(5,5)
```

```
np.random.randint(1,100)
```

```
np.random.randint(1,100,10)
```

進階方法

#重塑陣列

```
arr = np.arange(25)  
ranarr =  
np.random.randint(0,50,10)  
arr.reshape(5,5)
```

#取得極值

```
ranarr.max()  
ranarr.argmax()  
ranarr.min()  
ranarr.argmin()
```

#取得陣列維度

```
arr.shape  
arr.reshape(1,25)  
arr.reshape(1,25).shape  
arr.reshape(25,1)  
arr.reshape(25,1).shape
```

#取得資料型態

```
arr.dtype
```


Numpy 與資料篩選

#像List 一樣利用[]篩選資料

```
arr = np.arange(0,11)
```

```
arr[8]
```

```
arr[1:5]
```

#廣播 (Broadcasting)

```
arr = np.arange(0,11)
```

```
slice_of_arr = arr[0:6]
```

```
slice_of_arr[:] = 99
```

```
slice_of_arr
```

#二維資料取值

```
arr_2d = np.array([[5,10,15],[20,25,30],[35,40,45]])
```

```
arr_2d[1]
```

```
arr_2d[1][0]
```

```
arr_2d[1,0]
```

```
arr_2d[2]
```

```
arr_2d[2,:]
```

#用條件篩選資料

```
arr = np.arange(1,11)
```

```
arr > 4
```

```
bool_arr = arr > 4
```

```
arr[bool_arr]
```

```
arr[arr > 2]
```

Numpy 與資料篩選

Numpy 計算

```
arr = np.arange(0,10)
```

```
arr + arr
```

```
arr * arr
```

```
arr - arr
```

```
arr/arr
```

```
1/arr
```

```
arr**3
```

Numpy 內建計算函數

```
np.sqrt(arr)
```

```
np.exp(arr)
```

```
np.max(arr)
```

```
np.sin(arr)
```

```
np.log(arr)
```

Pandas

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

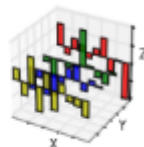
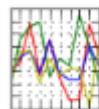
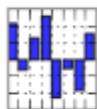
Pandas

■ 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

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



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 |

Numpy 與 Pandas

■ Numpy

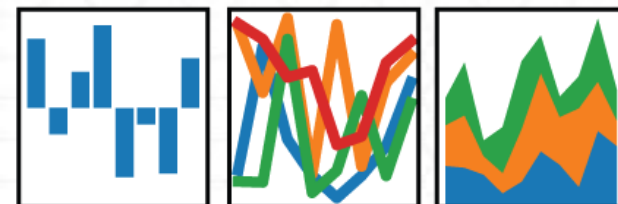
- ▣ 1D Array 如同R 的Vector
- ▣ 2D Array 如同R 的Matrix
- ▣ 只允許同一資料型態



■ Pandas

- ▣ 如同R 的DataFrame
- ▣ 可以使用欄位或列名取得資料
- ▣ 允許混雜不同資料型態

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$





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)

建立Series

```
import numpy as np
import pandas as pd
```

從List 建立Series

```
labels = ['a','b','c']
my_list = [10,20,30]
```

```
pd.Series(data=my_list)
pd.Series(data=my_list,index=labels)
pd.Series(my_list,labels)
```

從Numpy Arrays 建立 Series

```
arr = np.array([10,20,30])
pd.Series(arr)
pd.Series(arr,labels)
```

從Dictionary 建立 Series

```
d = {'a':10,'b':20,'c':30}
pd.Series(d)
```

使用索引

```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
```

```
ser2 = pd.Series([1,2,5,4],index = ['USA', 'Germany','Italy', 'Japan'])
```

```
ser1['USA']
```

```
ser1 + ser2
```

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/删除欄位

■ 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)
```

■ 删除第六列

```
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([\
```

可以使用np.nan 代表缺失值

```
    ['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
```

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

- 捨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

讀取資料

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/rent_591.csv',
index_col=0)
df
```

| | search_date | title | address | floor_info | price | layout | building_area | building_use |
|----|-------------|---------------------|--------------|------------|-----------|-----------|---------------|--------------|
| 5 | 2018-05-12 | 近捷運，採光佳，有景觀，全新家電裝潢 | 新北市淡水區民族路 | 9F/11F | 22,000元/月 | 2房1廳1衛1陽臺 | 24.00 | 電梯大樓/整層住家 |
| 9 | 2018-05-14 | 大衛營優質社區管理好，交通便利、裝潢佳 | 基隆市安樂區樂利三街 | 14F/18F | 17,000元/月 | 2房1廳2衛2陽臺 | 32.00 | 電梯大樓/整層住家 |
| 13 | 2018-05-14 | 家樂福夜市附近，雅房出租2房2廳 | 嘉義市西區興業西路 | 7F/8F | 7,500元/月 | 2房2廳1衛1陽臺 | 24.00 | 電梯大樓/整層住家 |
| 15 | 2018-05-14 | 台北市北投新民路捷運溫泉78坪雙拼美廈 | 台北市北投區新民路 | 2F/9F | 25,000元/月 | 4房3廳3衛3陽臺 | 78.00 | 電梯大樓/整層住家 |
| 16 | 2018-05-13 | 出海口海景高樓層溫馨房 | 新北市淡水區中正東路一段 | 19F/21F | 15,000元/月 | 1房1廳1衛0陽臺 | 18.01 | 電梯大樓/整層住家 |

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(',', '').replace('元/月',''))  
    return res  
df['price'].map(normalizePrice)
```

■ 使用匿名函式

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

Apply

```
df = pd.DataFrame(\n    [\n        [60,70,50],\n        [80,79,68],\n        [63,66,82]], \n    columns = ['First', 'Second', 'Third'])\n\ndf.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('2019-04-21', '%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(1555860118)
```

使用 pandas 轉換時間

■ 使用pandas 轉換時間

```
import pandas
```

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


虛擬變量 (DUMMY VARIABLE)

虛擬變量 (Dummy Variable)

| | search_date | title | address | floor_info | price | layout | building_area | building_use |
|----|-------------|---------------------|--------------|------------|-----------|-----------|---------------|--------------|
| 5 | 2018-05-12 | 近捷運，採光佳，有景觀，全新家電裝潢 | 新北市淡水區民族路 | 9F/11F | 22,000元/月 | 2房1廳1衛1陽臺 | 24.00 | 電梯大樓/整層住家 |
| 9 | 2018-05-14 | 大衛營優質社區管理好，交通便利、裝潢佳 | 基隆市安樂區樂利三街 | 14F/18F | 17,000元/月 | 2房1廳2衛2陽臺 | 32.00 | 電梯大樓/整層住家 |
| 13 | 2018-05-14 | 家樂福夜市附近，雅房出租2房2廳 | 嘉義市西區興業西路 | 7F/8F | 7,500元/月 | 2房2廳1衛1陽臺 | 24.00 | 電梯大樓/整層住家 |
| 15 | 2018-05-14 | 台北市北投新民路捷運溫泉78坪雙拼美廈 | 台北市北投區新民路 | 2F/9F | 25,000元/月 | 4房3廳3衛3陽臺 | 78.00 | 電梯大樓/整層住家 |
| 16 | 2018-05-13 | 出海口海景高樓層溫馨房 | 新北市淡水區中正東路一段 | 19F/21F | 15,000元/月 | 1房1廳1衛0陽臺 | 18.01 | 電梯大樓/整層住家 |

如何創造虛擬變數?

虛擬變數 (Dummy Variable)

#建立虛擬變數

```
pd.get_dummies(df['building_use'])
```

#合併虛擬變數與原DataFrame

```
df = pd.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



Pandas Aggregation

GroupBy

#使用SQL 計算平均價格

```
select building_use, mean(price) from df group by building_use;
```

#使用Pandas 計算平均價格

```
df.groupby('building_use')['price'].mean()
```

Multiple Aggregation

#根據search_date, building_use 計算平均價格

```
df.groupby(['search_date','building_use'])['price'].mean()
```

Sum, Mean, Std

#SUM

```
df.groupby('building_use')['price'].sum()
```

#MEAN

```
df.groupby('building_use')['price'].mean()
```

#STD

```
df.groupby('building_use')['price'].std()
```


資料合併

建立資料集

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],  
                    'B': ['B0', 'B1', 'B2', 'B3'],  
                    'C': ['C0', 'C1', 'C2', 'C3'],  
                    'D': ['D0', 'D1', 'D2', 'D3']},  
                    index=[0, 1, 2, 3])
```

```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],  
                    'B': ['B4', 'B5', 'B6', 'B7'],  
                    'C': ['C4', 'C5', 'C6', 'C7'],  
                    'D': ['D4', 'D5', 'D6', 'D7']},  
                    index=[4, 5, 6, 7])
```

```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],  
                    'B': ['B8', 'B9', 'B10', 'B11'],  
                    'C': ['C8', 'C9', 'C10', 'C11'],  
                    'D': ['D8', 'D9', 'D10', 'D11']},  
                    index=[8, 9, 10, 11])
```

Concatenation

#根據列合併

```
pd.concat([df1,df2,df3])
```

#根據行合併

```
pd.concat([df1,df2,df3],axis=1)
```

Merge

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],  
                    'A': ['A0', 'A1', 'A2', 'A3'],  
                    'B': ['B0', 'B1', 'B2', 'B3']})
```

```
right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],  
                    'C': ['C0', 'C1', 'C2', 'C3'],  
                    'D': ['D0', 'D1', 'D2', 'D3']})
```

```
pd.merge(left, right, how='inner', on='key')
```


Merge (Con'd)

```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],  
                     'key2': ['K0', 'K1', 'K0', 'K1'],  
                     'A': ['A0', 'A1', 'A2', 'A3'],  
                     'B': ['B0', 'B1', 'B2', 'B3']})
```

```
right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],  
                      'key2': ['K0', 'K0', 'K0', 'K0'],  
                      'C': ['C0', 'C1', 'C2', 'C3'],  
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

```
pd.merge(left, right, how='inner', on=['key1', 'key2'])
```

Join

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],  
                    'B': ['B0', 'B1', 'B2']},  
                    index=['K0', 'K1', 'K2'])
```

```
right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],  
                    'D': ['D0', 'D2', 'D3']},  
                    index=['K0', 'K2', 'K3'])
```

```
left.join(right)
```

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

$\wedge [A-Z] \setminus d \{9\} \$$

- 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', 'balcony']] = df['layout'].str.extract('(\d+)房(\d+)廳(\d+)衛(\d+)陽臺',  
expand=False)
```

```
df2[['layout', 'bedroom', 'living_room', 'bathroom', 'balcony']].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

讀取房屋資料

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/rent_591.csv',
index_col=0)
df
```

寫入文字資料

■ 寫進csv 檔案

```
df.to_csv('rent.csv')
```

■ 寫進Excel 檔案

```
df.to_excel('rent.xlsx')
```


轉換成JSON 資料

■ 直接轉換成JSON API

`df.to_json()`

從剪貼簿貼入DataFrame

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

借貸俱樂部資料清理

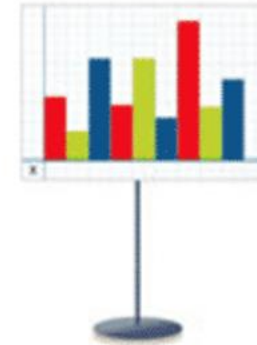
Lending Club



Borrowers apply for loans.
Investors open an account.



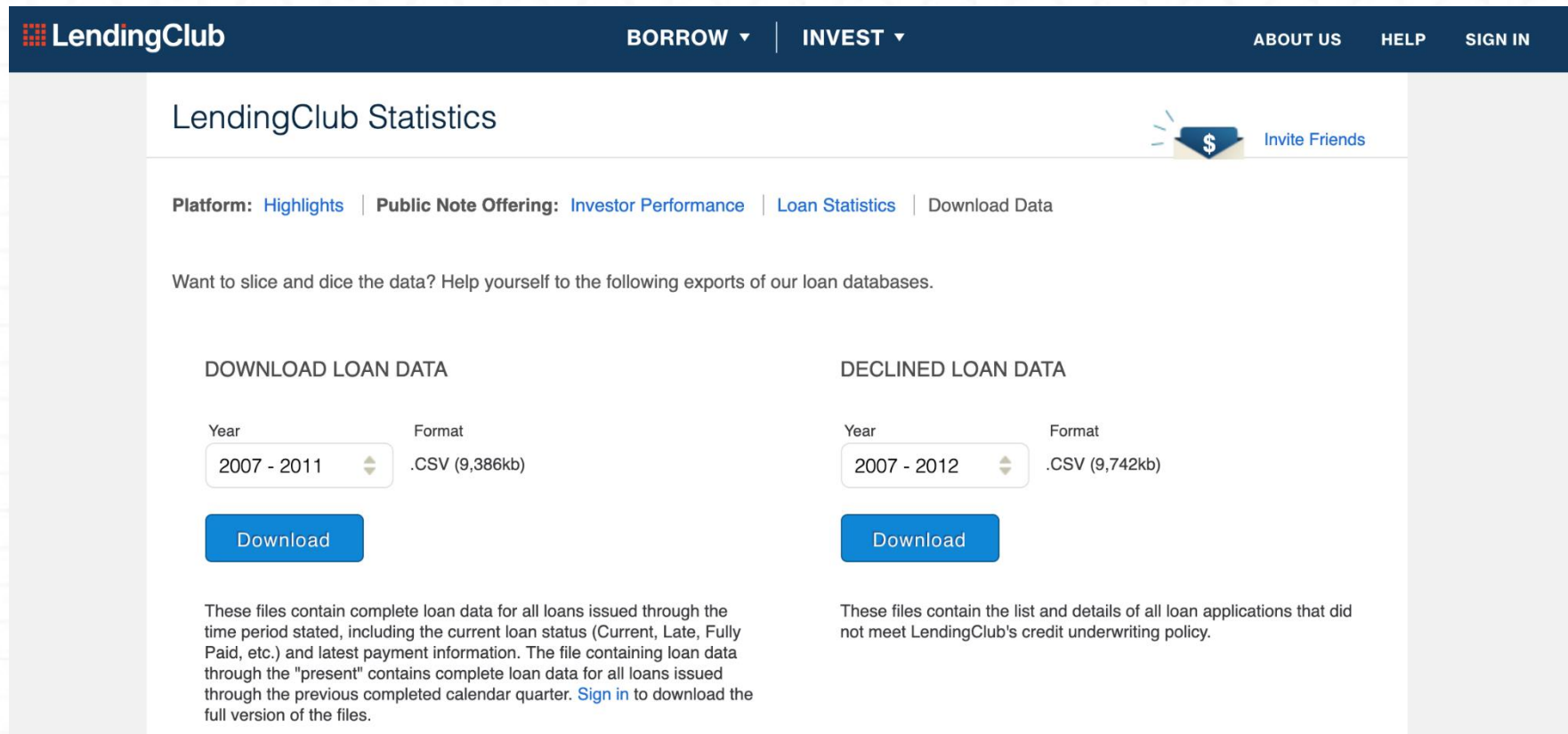
Borrowers get funded.
Investors build a portfolio.



Borrowers repay automatically.
Investors earn & reinvest.

借貸俱樂部資料

■ <https://www.lendingclub.com/info/download-data.action>



The screenshot shows the LendingClub website's 'Statistics' section. The header includes the LendingClub logo, navigation links for 'BORROW' and 'INVEST', and links for 'ABOUT US', 'HELP', and 'SIGN IN'. The main heading is 'LendingClub Statistics'. Below this, there are links for 'Platform: Highlights', 'Public Note Offering: Investor Performance', 'Loan Statistics', and 'Download Data'. A sub-header reads: 'Want to slice and dice the data? Help yourself to the following exports of our loan databases.'

There are two main sections for downloading data:

- DOWNLOAD LOAN DATA**: Includes a 'Year' dropdown set to '2007 - 2011' and a 'Format' dropdown set to '.CSV (9,386kb)'. A blue 'Download' button is below.
- DECLINED LOAN DATA**: Includes a 'Year' dropdown set to '2007 - 2012' and a 'Format' dropdown set to '.CSV (9,742kb)'. A blue 'Download' button is below.

Below the download buttons, there is explanatory text for each section. For 'DOWNLOAD LOAN DATA', it states: 'These files contain complete loan data for all loans issued through the time period stated, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. The file containing loan data through the "present" contains complete loan data for all loans issued through the previous completed calendar quarter. [Sign in](#) to download the full version of the files.' For 'DECLINED LOAN DATA', it states: 'These files contain the list and details of all loan applications that did not meet LendingClub's credit underwriting policy.'

資料清理

#讀取資料

```
dataset = pd.read_csv('LoanStats.csv')
```

#移除空白欄位

```
dataset = dataset.iloc[:,2:111]
```

```
empty_cols = [i for i in range(45,72)]
```

```
dataset = dataset.drop(dataset.columns[empty_cols],axis=1)
```

```
data_with_loanstatus_sliced = dataset[(dataset['loan_status']=="Fully Paid") | (dataset['loan_status']=="Charged Off")]
```

#轉換目標編碼

```
di = {"Fully Paid":0, "Charged Off":1}
```

```
Dataset_withBoolTarget= data_with_loanstatus_sliced.replace({"loan_status": di})
```

```
Dataset_withBoolTarget['loan_status'].value_counts()
```

```
print("Current shape of dataset :",Dataset_withBoolTarget.shape)
```

```
Dataset_withBoolTarget.head(3)
```

資料清理

#移除空白列

```
dataset=Dataset_withBoolTarget.dropna(thresh = 340000,axis=1)
print("Current shape of dataset :",dataset.shape)
```

#移除欄位

```
del_col_names = ["delinq_2yrs", "last_pymnt_d", "chargeoff_within_12_mths","delinq_amnt","emp_title", "term", "emp_title",
"pymnt_plan","purpose","title", "zip_code", "verification_status", "dti","earliest_cr_line", "initial_list_status", "out_prncp",
"pymnt_plan", "num_tl_90g_dpd_24m", "num_tl_30dpd", "num_tl_120dpd_2m", "num_accts_ever_120_pd", "delinq_amnt",
"chargeoff_within_12_mths", "total_rec_late_fee", "out_prncp_inv", "issue_d"] #deleting some more columns
dataset = dataset.drop(labels = del_col_names, axis = 1)
```

#篩選欄位

```
features = ['funded_amnt','emp_length','annual_inc','home_ownership','grade',
            "last_pymnt_amnt", "mort_acc", "pub_rec", "int_rate", "open_acc","num_actv_rev_tl",
            "mo_sin_rcnt_rev_tl_op","mo_sin_old_rev_tl_op","bc_util","bc_open_to_buy",
            "avg_cur_bal","acc_open_past_24mths",'loan_status'] #sub_grade' #selecting final features #'addr_state'tax_liens',
Final_data = dataset[features] #19 features with target var
Final_data["int_rate"] = Final_data["int_rate"].apply(lambda x:float(x[:-1])) #reomving % sign, conv to float - int_rate column
Final_data= Final_data.reset_index(drop=True)
print("Current shape of dataset :",Final_data.shape)
```

資料轉換

#資料編碼

```
Final_data['grade'] = Final_data['grade'].map({'A':7,'B':6,'C':5,'D':4,'E':3,'F':2,'G':1})
Final_data["home_ownership"] =
Final_data["home_ownership"].map({"MORTGAGE":6,"RENT":5,"OWN":4,"OTHER":3,"NONE":
2,"ANY":1})
Final_data["emp_length"] = Final_data["emp_length"].replace({'years':"','year':"','<':"','\+':"','n/a':'0'}, regex = True)
Final_data["emp_length"] = Final_data["emp_length"].apply(lambda x:int(x))
print("Current shape of dataset :",Final_data.shape)
Final_data.head()
```


#移除空值

```
Final_data.fillna(Final_data.mean(),inplace = True)
```

#資料標準化

```
scl = preprocessing.StandardScaler() #instance of preprocessing
```

```
fields = Final_data.columns.values[:-1]
```

```
data_clean = pd.DataFrame(scl.fit_transform(Final_data[fields]),  
columns = fields)
```

```
data_clean['loan_status'] = Final_data['loan_status']
```

```
data_clean['loan_status'].value_counts()
```

移除空值與資料標準化

#移除空值

```
Final_data.fillna(Final_data.mean(),inplace = True)
```

#資料標準化

```
scl = preprocessing.StandardScaler() #instance of preprocessing
```

```
fields = Final_data.columns.values[:-1]
```

```
data_clean = pd.DataFrame(scl.fit_transform(Final_data[fields]),  
columns = fields)
```

```
data_clean['loan_status'] = Final_data['loan_status']
```

```
data_clean['loan_status'].value_counts()
```

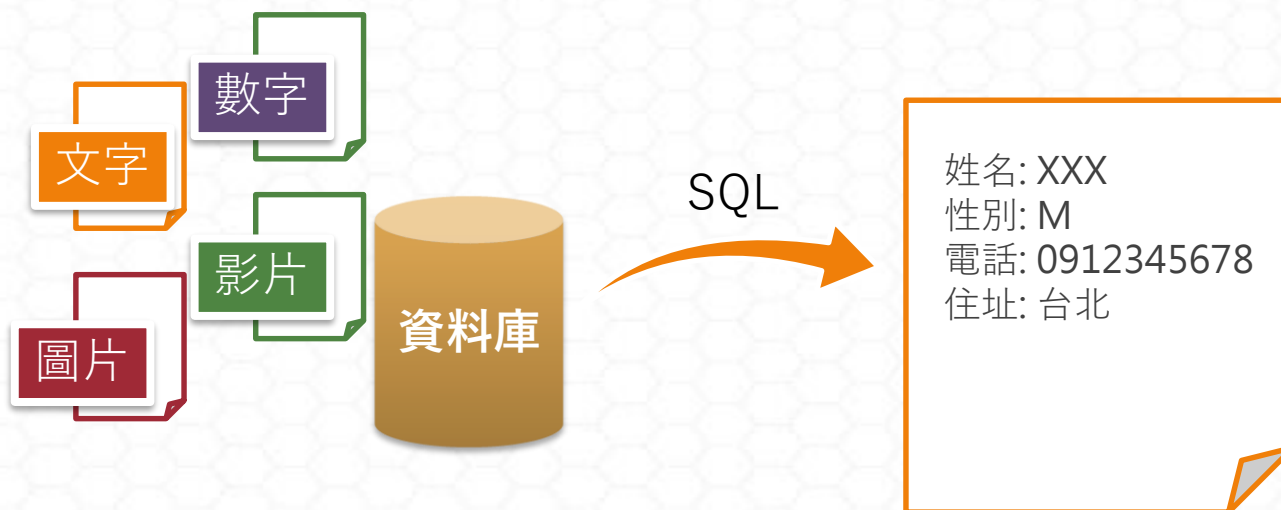
移除空值與資料標準化

```
loanstatus_0 = data_clean[data_clean["loan_status"]==0]
loanstatus_1 = data_clean[data_clean["loan_status"]==1]
subset_of_loanstatus_0 = loanstatus_0.sample(n=5500)
subset_of_loanstatus_1 = loanstatus_1.sample(n=5500)
data_clean = pd.concat([subset_of_loanstatus_1,
subset_of_loanstatus_0])
data_clean = data_clean.sample(frac=1).reset_index(drop=True)
```

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



資料庫的ACID

- 不可分割性 (**A**tomicity)
 - 交易必須全部完成或全部不完成
 - (e.g. 轉帳)
- 一致性 (**C**onsistency)
 - 交易開始到結束，資料完整性都符合既設規則與限制
 - (e.g. 帳號)
- 隔離性 (**I**solation)
 - 並行的交易不會引響彼此
 - (e.g. 餘額查詢)
- 持久性 (**D**urability)
 - 進行完交易後，對資料庫的變更會永久保留在資料庫
 - (e.g. 系統毀損)

SQLite

■ 特性

- self-contained
- serverless
- zero-configuration
- Transactional

■ ACID 資料庫

■ 支援 SQL 92 語法

■ 開源



使用Python 連結資料庫

```
import sqlite3 as lite
con = lite.connect('test.sqlite')
cur = con.cursor()
cur.execute('SELECT SQLITE_VERSION()')
data = cur.fetchone()
print(data)
con.close()
```

可以搭配 with 語句

透過SQLite 做資料新增、查詢

```
import sqlite3 as lite
with lite.connect("test.sqlite") as con:
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS PhoneAddress")
    cur.execute("CREATE TABLE PhoneAddress(phone CHAR(10) PRIMARY KEY, address TEXT, name TEXT unique,
age INT NOT NULL)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0912173381','United State','Jhon Doe',53)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0928375018','Tokyo Japan','MuMu Cat',6)")
    cur.execute("INSERT INTO PhoneAddress VALUES('0957209108','Taipei','Richard',29)")
    cur.execute("SELECT phone,address FROM PhoneAddress")
    data = cur.fetchall()
    for rec in data:
        print(rec[0], rec[1])
```

fetchone v.s. fetchall

■ fetchone

```
data = cur.fetchone()  
print(data[0], data[1])
```

■ fetchall

```
rows = cur.fetchall()  
for row in rows:  
    print(row)
```

使用Pandas 儲存資料

■ 建立DataFrame

```
import sqlite3 as lite
import pandas
employee = [{'name': 'Mary', 'age': 23, 'gender': 'F'}, {'name': 'John',
'age': 33, 'gender': 'M'}]
df = pandas.DataFrame(employee)
```

■ 使用Pandas 儲存資料

```
with lite.connect('test.sqlite') as db:
    df.to_sql(name='employee', index=False, con=db,
if_exists='replace')
```


儲存實例

讀取房屋資料

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/ywchiu/ctbcpy/master/data/rent_591.csv',
index_col=0)
df
```

資料轉換

■ 轉換金額

```
def normalizePrice(ele):  
    res = int(ele.replace(',', '').replace('元/月',''))  
    return res
```

```
df['price'] = df['price'].map(normalizePrice)
```

存儲資料到資料庫

```
import sqlite3 as lite
import pandas
with lite.connect('house.sqlite') as db:
    df.to_sql('rent_591', con = db, if_exists='replace', index=None)
```


篩選數據

■ 讀取表格

```
select * from rent_591;
```

■ 篩選行

```
select * from rent_591 where price > 10000;
```

■ 篩選欄位

```
select title, building_area, price from rent_591;
```

排序數據

■ 排序數據

```
select title, building_area, price from rent_591 order by price desc;
```

■ 取得前三排行的資料

```
select title, building_area, price from rent_591 order by price desc  
limit 3 ;
```

聚合數據

■ 使用Group By 聚合數據

```
select building_use, avg(price) from rent_591 group by  
building_use;
```

■ 使用Having 挑選出平均價格為10,000以上的建物類型

```
select building_use, avg(price) from rent_591 group by building_use  
having avg(price) >= 10000;
```

The background features a light blue hexagonal grid pattern. Overlaid on this is a large, faint, circular graphic composed of several concentric rings. These rings are made of segments in various shades of blue and white, creating a sense of depth and movement, similar to a stylized eye or a target. The text "THANK YOU" is centered horizontally and vertically over this graphic.

THANK YOU