國立高雄科技大學
National Kaohsiung University of Science and Technology

高雄科技大學資訊財務碩士學位學程

進階程式設計
– Pandas 概述(3/3)

郭建良
DavidCLKuo@nkust.edu.tw
2020.03.20

本日重點

- 基礎數值與統計 (完成上週)
- 類別資料的應用 (完成上週)
- 整理資料
- 結合 / 關聯與重塑資料
- 資料聚合
- 存取資料

本日使用到的初始設定

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import datetime
from datetime import datetime, date
```

基礎數值與統計

(把沒講完的補上)

資料重來: 重新載入兩個 csv 檔

```
sp500 = pd.read_csv("sp500.csv", index_col='Symbol',
usecols=[0, 2, 3, 7])
omh = pd.read_csv("omh.csv")
```

數值讀取和尋找

```
# 傳回索引或傳回結果
omh[ ['MSFT', 'AAPL'] ].min( )
omh[ ['MSFT', 'AAPL'] ].max( )
omh[ ['MSFT', 'AAPL'] ].idxmin( )
omh[ ['MSFT', 'AAPL'] ].idxmax( )

# 找出 N 個和累計
omh.nsmallest (4, ['MSFT']) ['MSFT']
omh.nlargest(4, ['MSFT']) ['MSFT']
pd.Series([1, 2, 3, 4]).cumsum( )
pd.Series( [1, 2, 3, 4] ).cumprod( )
```

變動百分比及平移運算

```
omh[ ['MSFT'] ].pct_change( )[3:10]


np.random.seed(1234)
s = pd.Series(np.random.randn(1000)).cumsum( )
s[:5]
s[0:100].plot( );


r = s.rolling(window = 3)
r
means = r.mean( )
means[:7]
```

變動百分比及平移運算(續)

```
means[4:10]

s[0:3].mean( )

s[1:4].mean( )

means[0:100].plot( );
```

隨機取樣

```
np.random.seed(1234)
df = pd.DataFrame(np.random.randn(50, 4))
df[:5]

# 利用 smaple( ) 指定要提取的樣本數
df.sample(n=3)
df.sample(frac=0.1)      # 取 10% 的列
df.sample(frac=0.1, replace=True)    # 取後是否放回
```

如何判別與處理資料遺漏

開始先建立 DataFrame

```
# 請注意如何形成 M x N 矩陣

df = pd.DataFrame(np.arange(0, 15).reshape(5, 3),
                  index=['a', 'b', 'c', 'd', 'e'],
                  columns=['X', 'Y', 'Z'])
df
```

	X	Y	Z
a	0	1	2
b	3	4	5
c	6	7	8
d	9	10	11
e	12	13	14

開始先建立 DataFrame (續)

```
# 把原矩陣擴大然後加上 NaN 的數值

df['W1'] = np.nan

df.loc['f'] = np.arange(15, 19)

df.loc['g'] = np.nan

df['W2'] = np.nan

df['W1']['a'] = 20

df
```

開始進行判斷 NaN 及相關操作

- df.isnull( )
- df.notnull( )
- df.count( )
- df.copy( )
- df.fillna( )　　# 有不同填補遺漏值的方式
- df.dropna( )　# 有額外參數設定 & recall 參數查法
- df.isnull( ).sum( )　# 進行複合功能的使用
- NumPy vs. Pandas 處理 NaN 的差異

df.isnull( )

df.isnull( ).sum( )

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

	X	:	Z
a			
b			
:			
:			
e			

X	:	Z	

df.count( )

len(df) - df.count( )

(len(df) - df.count( )).sum( )

兩者差在哪 (有無異取同工之處)

```
df.notnull( )

df.W1[df.W1.notnull(  )]

df.W1.dropna(  )

df.W1                              # 確認 dropna 是否對 df 有影響

df.dropna(    )                    # 請問發生何事

df.dropna(how = 'all')             # 反過來的思維

df.dropna(how='all', axis=1)       # 矩陣行列觀點
```

```
df2 = df.copy(   )                          #  以下純粹衍生練習

df2.loc['g'].X = 0

df2.loc['g'].Z = 0

df2

df2.dropna(how='any', axis=1)

df2.dropna(thresh=4, axis=1)        # Nan 達 4 個的門檻者
```

```
#  超級比一比看看發生何事

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

s = pd.Series(a)

a.mean( ), s.mean(  )

s = df.W1

s.sum( )

s.mean( )

s.cumsum(  )

df.W1 + 1
```

```
#  pandas 看到 NaN 時不是將之視為 0
就是忽略
#  NumPy 遇到 NaN 就會直接回傳 NaN
```

```
filled = df.fillna(0)
filled
df.mean( )                    #  再認真看一下
filled.mean( )

df.fillna(df.mean( ))
df.W1.fillna(method="ffill")      #  df.ffill( )
df.W1.fillna(method="bfill")      #  df.bfill( )
```

內插補值和資料處理

- pd.interpolate( )　　# 內插補值 (可設參數)
- pd.duplicated( )　　# 找出資料重複與否
- pd.drop_duplicates( )　　# 傳回移除重複列後的資料
- pd.map( )　　# 映射
- pd.replace( )
- pd.apply( )　　# 套用函數轉換資料

```
s = pd.Series([1, np.nan, np.nan, np.nan, 2])
s.interpolate(  )
s = pd.Series([0, np.nan, 100], index=[0, 1, 10])
s
s.interpolate(  )
s.interpolate(method="values")
```

```
ts = pd.Series([1, np.nan, 2],
           index=[datetime(2014, 1, 1),
                  datetime(2014, 2, 1),
                  datetime(2014, 4, 1)])
ts
ts.interpolate (  )
ts.interpolate(method="time")
```

```
data = pd.DataFrame({'a': ['x'] * 3 + ['y'] * 4,
                     'b': [1, 1, 2, 3, 3, 4, 4]})
data
data.duplicated( )
data.drop_duplicates( )
data.drop_duplicates( ).count( )
data.drop_duplicates(keep='last')
data['c'] = range(7)
data.duplicated( )
data.drop_duplicates(['a', 'b'])
```

```
x = pd.Series({"one": 1, "two": 2, "three": 3})
y = pd.Series({1: "a", 2: "b", 3: "c"})
x
y
x.map(y)
```

比比看差在哪

```
x = pd.Series({"one": 1, "two": 2, "three": 3})
y = pd.Series({1: "a", 2: "b"})
x.map(y)
```

```
s = pd.Series([0., 1., 2., 3., 2., 4.])
s
s.replace(2, 5)
s.replace([0, 1, 2, 3, 4], [4, 3, 2, 1, 0])
s.replace({0: 10, 1: 100})
```

lambda 再現江湖

```
s = pd.Series(np.arange(0, 5))
s.apply(lambda v: v * 2)

df = pd.DataFrame(np.arange(12).reshape(4, 3),
                  columns=['a', 'b', 'c'])
df
df.apply(lambda col: col.sum( ))
df
df.apply(lambda row: row.sum(  ), axis=1)
```

```
df['interim'] = df.apply(lambda r: r.a * r.b, axis=1)
df
df['result'] = df.apply(lambda r: r.interim + r.c, axis=1)
df
df.a = df.a + df.b + df.c
df

df = pd.DataFrame(np.arange(0, 15).reshape(3,5))
df.loc[1, 2] = np.nan
df
df.dropna( ).apply(lambda x: x.sum( ), axis=1)
df.applymap(lambda x: '%.2f' % x)
```

結合 / 關聯及重塑資料

(主要只是為了串DBMS概念)

- pd.concat( )
- pd.merge( )
- pd.join( )

資料初始化

```
df1 = pd.DataFrame(np.arange(9).reshape(3, 3),
                   columns=['a', 'b', 'c'])
df2 = pd.DataFrame(np.arange(9, 18).reshape(3, 3),
                   columns=['a', 'b', 'c'])
df3 = pd.DataFrame(np.arange(20, 26).reshape(3, 2),
                   columns=['a', 'd'],
                   index=[2, 3, 4])
df1
df2
df3
```

```
df1+df2

pd.concat([df1, df2])

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

pd.concat([df1, df3], axis=1, join='inner')
```

牛刀小試

```
customers = {'CustomerID': [10, 11],
             'Name': ['Mike', 'Marcia'],
             'Address': ['Address for Mike',
                         'Address for Marcia']}
customers = pd.DataFrame(customers)

orders = {'CustomerID': [10, 11, 10],
          'OrderDate': [date(2014, 12, 1),
                        date(2014, 12, 1),
                        date(2014, 12, 1)]}
orders = pd.DataFrame(orders)

customers
orders
customers.merge(orders)
```

資料初始化

```
left_data = {'key1': ['a', 'b', 'c'],
             'key2': ['x', 'y', 'z'],
             'lval1': [ 0, 1, 2]}
right_data = {'key1': ['a', 'b', 'c'],
              'key2': ['x', 'a', 'z'],
              'rval1': [ 6, 7, 8 ]}
left = pd.DataFrame(left_data, index=[0, 1, 2])
right = pd.DataFrame(right_data, index=[1, 2, 3])
left
right
```

```
left.merge(right)

left.merge(right, on='key1')

left.merge(right, on=['key1', 'key2'])

pd.merge(left, right, left_index=True, right_index=True)
```

```
left.merge(right, how='outer')

left.merge(right, how='left')

left.merge(right, how='right')

left.join(right, lsuffix='_left', rsuffix='_right')

left.join(right, lsuffix='_left', rsuffix='_right', how='inner')
```

資料聚合

SAC模式: split-apply-combine

- pd.groupby( )
- 許多聚合函數都直接內建再 GroupBy中
- gb.size( )
- gb.agg( )

```
sensor_data = pd.read_csv("sensors.csv")

sensor_data[:5]


grouped_by_sensor = sensor_data.groupby('sensor')

grouped_by_sensor

grouped_by_sensor.ngroups

grouped_by_sensor.groups
```

回顧許久未見的 def

```
def print_groups (group_object):
    for name, group in group_object:
        print (name)
        print (group[:5])
```

取分組(grouping)結果

```
print_groups(grouped_by_sensor)

grouped_by_sensor.size( )

grouped_by_sensor.count( )

grouped_by_sensor.get_group('accel')[:5]

grouped_by_sensor.head(3)

grouped_by_sensor.nth(6)

grouped_by_sensor.describe( )
```

延伸的分組模式

```
mcg = sensor_data.groupby(['sensor', 'axis'])
print_groups(mcg)

mi = sensor_data.copy(  )
mi = mi.set_index(['sensor', 'axis'])
mi

print_groups(mi.groupby(level=0))
print_groups(mi.groupby(level=['sensor', 'axis']))
```

```
sensor_axis_grouping = mi.groupby(level=['sensor', 'axis'])
sensor_axis_grouping.agg(np.mean)
sensor_data.groupby(['sensor', 'axis'],
as_index=False).agg(np.mean)
sensor_axis_grouping.mean(  )
sensor_axis_grouping.agg([np.sum, np.std])
sensor_axis_grouping.agg({'interval' : len,
                          'reading': np.mean})
```

轉換分組資料

```
transform_data = pd.DataFrame({ 'Label': ['A', 'C', 'B', 'A', 'C'],
                                'Values': [0, 1, 2, 3, 4],
                                'Values2': [5, 6, 7, 8, 9],
                                'Other': ['foo', 'bar', 'baz',
                                          'fiz', 'buz']},
                               index = list('VWXYZ'))
transform_data

grouped_by_label = transform_data.groupby('Label')
print_groups(grouped_by_label)
grouped_by_label.transform(lambda x: x + 10)
```

用平均值補足缺漏資料

```
df = pd.DataFrame({ 'Label': list("ABABAB"),
                    'Values': [10, 20, 11, np.nan, 12, 22]})
grouped = df.groupby('Label')
print_groups(grouped)

grouped.mean( )
filled_NaNs = grouped.transform(lambda x:
                    x.fillna(x.mean( )))
filled_NaNs
```

轉為Z分數(正規化)

```
np.random.seed(123456)
data = pd.Series(np.random.normal(0.5, 2, 365*3),
                 pd.date_range('2018-01-01', periods=365*3))
periods = 100
rolling = data.rolling(
    window=periods,
    min_periods=periods,
    center=False).mean(  ).dropna(  )
rolling[:5]
rolling.plot(  );
```

```
group_key = lambda x: x.year
groups = rolling.groupby(group_key)
groups.agg([np.mean, np.std])

z_score = lambda x: (x - x.mean()) / x.std()
normed = rolling.groupby(group_key).transform(z_score)
normed.groupby(group_key).agg([np.mean, np.std])
compared = pd.DataFrame({ 'Original': rolling,
                          'Normed': normed })
compared.plot(  );
```

過濾分組資料

```
df = pd.DataFrame({'Label': list('AABCCC'),
                   'Values': [1, 2, 3, 4, np.nan, 8]})
df

f = lambda x: x.Values.count( ) > 1
df.groupby('Label').filter(f)
f = lambda x: x.Values.isnull( ).sum( ) == 0
df.groupby('Label').filter(f)
grouped = df.groupby('Label')
group_mean = grouped.mean().mean()
f = lambda x: abs(x.Values.mean() - group_mean) > 2.0
df.groupby('Label').filter(f)
```

存取資料

- 溫故知新

廢話不多說: 快速瀏覽 CSV(recall 3/13 p.09)

```
msft = pd.read_csv("msft.csv")
msft[:5]

msft = pd.read_csv("msft.csv", index_col=0)
msft[:5]


msft.dtypes
msft = pd.read_csv("msft.csv", dtype = { 'Volume' : np.float64})
msft.dtypes
```

```
df = pd.read_csv("msft.csv", header=0,
        names=['date', 'open', 'high', 'low', 'close', 'volume'])
df[:5]

df2 = pd.read_csv("msft.csv", usecols=['Date', 'Close'],
index_col=['Date'])
df2[:5]

df = pd.read_csv("msft2.csv", skiprows=[0, 2, 3])
df[:5]
```

```
df2.to_csv("msft_modified.csv", index_label='date')
df2.head(  )
df2.dtypes

df.to_csv("msft_piped.txt", sep='|')
```

```
# 在 Anaconda 必須強制設定 (否則會有警告訊息)
df = pd.read_csv("msft_with_footer.csv", skipfooter=2,
engine='python')
df

pd.read_csv("msft.csv", nrows=3)

pd.read_csv("msft.csv", skiprows=100, nrows=5, header=0,
        names=['date', 'open', 'high', 'low', 'close', 'vol'])
```

XLS 再次見面

```
df = pd.read_excel("stocks.xlsx")
df[:5]
                                        # sheet_name vs. sheetname

aapl = pd.read_excel("stocks.xlsx", sheet_name='aapl')
aapl[:5]

df.to_excel("stocks2.xls")
df.to_excel("stocks_msft.xls", sheet_name='MSFT')

df.to_excel("msft2.xlsx")
```

```
# 懶要有專業~~
from pandas import ExcelWriter

with ExcelWriter("all_stocks.xls") as writer:
    aapl.to_excel(writer, sheet_name='AAPL')
    df.to_excel(writer, sheet_name='MSFT')


df = pd.read_excel("stocks.xlsx")
df.head(2).to_html("stocks.html")
```

FDIC 破產銀行資料

```
url =
"http://www.fdic.gov/bank/individual/failed/banklist.html"
banks = pd.read_html(url)
banks[0][0:5].iloc[:,0:2]
banks[0][51:55]
```

額外安裝

```
!pip install pandas_datareader

import pandas_datareader as pdr

from pandas_datareader import wb
```

```
# 取得世界銀行資料 (http://www.worldbank.org/)
# 取得國家完整清單
# 用 wb.download( ) 抓取個月預期壽命
countries = pdr.wb.get_countries(  )
countries.loc[0:5,['name', 'capitalCity', 'iso2c']]

le_data_all = pdr.wb.download(indicator="SP.DYN.LE00.IN",
start='1980', end='2019')
le_data_all.head(  )
le_data_all.index.levels[0]
```

```
le_data_all = wb.download(indicator="SP.DYN.LE00.IN", country = countries['iso2c'], start='1990', end='2019')
le_data_all

le_data = le_data_all.reset_index( ).pivot(index='country', columns='year')
le_data.iloc[:9, 15: 20]

test = le_data.iloc[:9, 15:20]
test.to_csv("tt.csv")
```

```
country_with_least_expectancy = le_data.idxmin(axis=0)
country_with_least_expectancy[:5]

expectancy_for_least_country = le_data.min(axis=0)
expectancy_for_least_country[:5]

least = pd.DataFrame(                                    # HW: 試著做出延伸的國家資料表
    data = {'Country': country_with_least_expectancy.values,
            'Expectancy': expectancy_for_least_country.values},
    index = country_with_least_expectancy.index.levels[1])
least[:5]
```

類別資料的應用

(同3/13講義)

建立統計上的類別變數

```
# 使用 .Categorical( )
lmh_values = ["low", "high", "medium", "medium", "high"]
lmh_cat = pd.Categorical(lmh_values)
lmh_cat

lmh_cat.categories
lmh_cat.get_values( )
lmh_cat.codes
```

建立統計上的類別變數(續)

```
# 重建編碼的合理順序
lmh_cat = pd.Categorical ( lmh_values,
                          categories=["low", "medium", " high"] )
lmh_cat
lmh_cat.codes
lmh_cat.get_values( )
lmh_cat.sort_values( )
```

建立統計上的類別變數(續)

```
# 利用 .astype 的做法 & 用 .cat 來讀取

cat_series = pd.Series(lmh_values, dtype="category")

cat_series


s = pd.Series(lmh_values)

asCat = s.astype("category")

asCat


cat_series.cat.categories
```

用其他pd函式產生類別物件

```
# 用 .cut( )
np.random.seed(12345)
values = np.random.randint(0, 100, 5)
bins = pd.DataFrame({ "Values": values} )
bins

bins['Group'] = pd.cut(values, range(0, 101, 10))
bins
bins.Group
```

用其他pd函式產生類別物件(續)

```
metal_values = ["bronze", "gold", "silver", "bronze"]
metal_categories = ["bronze", "silver", "gold"]
metals = pd.Categorical(metal_values,
categories=metal_categories, ordered = True)
metals

metals_reversed_values = pd.Categorical(metals.get_values()[::-1],
    categories = metals.categories, ordered=True)
metals_reversed_values
```

用其他pd函式產生類別物件(續)

```
metals <= metals_reversed_values

metals.codes

metals_reversed_values.codes


# 當指定不存在的類別時... 看看會出現啥情況

pd.Categorical(["bronze", "copper"], categories=metal_categories)
```

類別物件的敘述性統計資訊

metals.describe( )

metals.value_counts( )

(metals.min( ), metals.max( ), metals.mode( ))

小案例: 成績處理

```
np.random.seed(123)
names = ['Eve', 'Norris', 'Rose', 'Lane', 'Sky', 'Handsome', 'Dave', 'Katina', 'Alice', "Mark"]
grades = np.random.randint(50, 101, len(names))
scores = pd.DataFrame({'Name': names, 'Grade': grades})
scores

score_bins =    [ 0,  59,   62,  66,   69,   72,  76,   79,   82,  86,   89,  92,  99, 100]
letter_grades = ['F', 'D-', 'D', 'D+', 'C-', 'C', 'C+', 'B-', 'B', 'B+', 'A-', 'A', 'A+']
```

小案例: 成績處理(續)

```
letter_cats = pd.cut(scores.Grade, score_bins,
labels=letter_grades)
scores['Letter'] = letter_cats
scores

letter_cats.head( )
scores.Letter.value_counts( ).head( )
scores.sort_values(by=['Letter'], ascending=False)
scores.sort_values(by=['Letter'],
ascending=False).Letter.value_counts( ).head( )
```