01.Python語言簡介

大數據分析

- R/Python/Julia/SQL程式設計與應用 (R/Python/Julia/SQL Programming and Application)
- 資料視覺化 (Data Visualization)
- 機器學習 (Machine Learning)
- 統計品管 (Statistical Quality Control)
- 最佳化 (Optimization)

李明昌博士
https://www.youtube.com/@alan9956
http://rwepa.blogspot.com/
alan9956@gmail.com

大綱

- 1.RWEPA簡介
- 2.資料分析暨視覺化的心法
- 3.Python 簡介
- 4.Anaconda 簡介
- 5.資料型態與四大資料物件
- 6.pandas 資料處理

1.RWEPA簡介

RWEPA簡介 http://rwepa.blogspot.com/

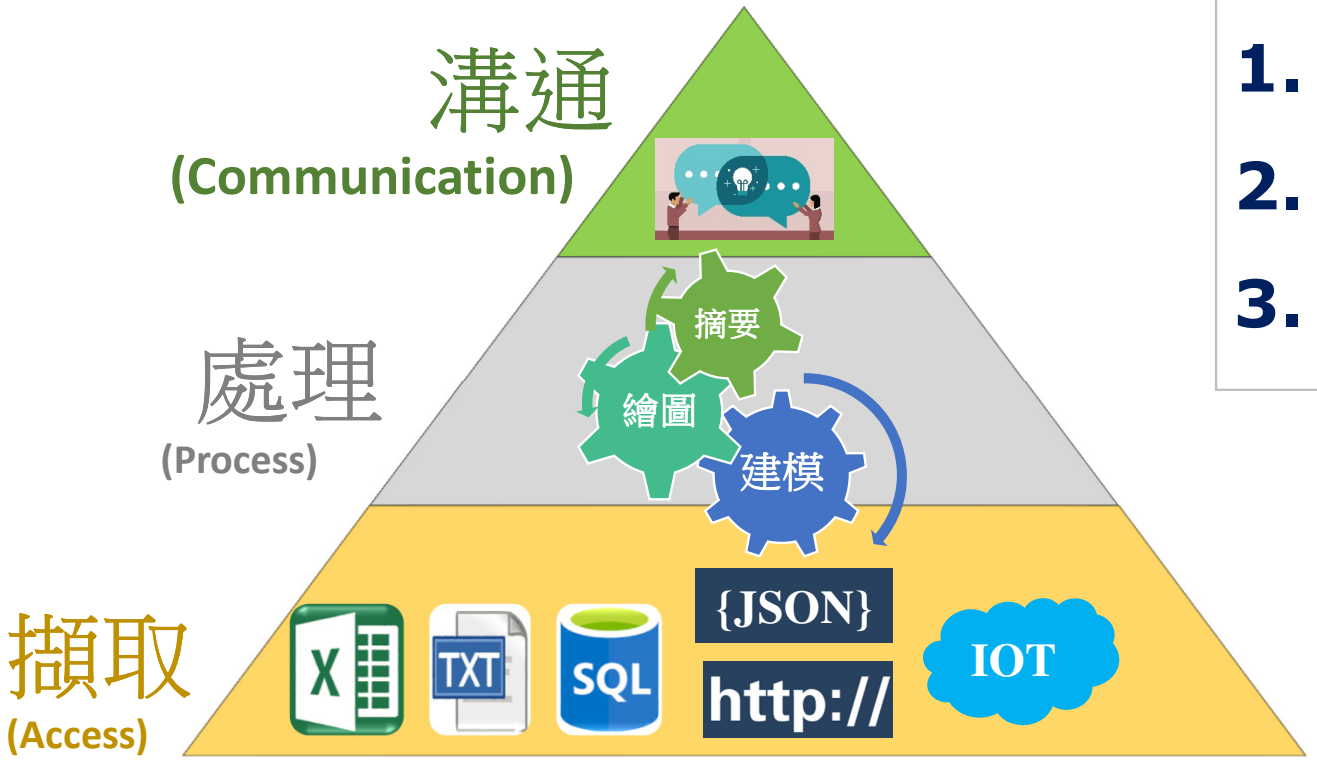
- 姓名：李明昌 (ALAN LEE)
- 現職：中華R軟體學會 常務理事
  臺灣資料科學與商業應用協會 常務理事
- 學歷：中原大學 工業與系統工程所 博士
- 經歷：
  - 育達科技大學 資訊管理系(所) 專任助理教授
  - 佛光大學 兼任教師
  - 國立台北商業大學 兼任教師
  - 東吳大學 兼任教師
  - 崇友實業 行銷企劃專員
  - 國航船務代理股份有限公司 海運市場運籌管理員

- 大專院校、資策會、工業技術研究院、國家發展委員會、中央氣象局、公平交易委員會、各縣市政府與日本名古屋產業大學等公民營單位演講達348餘場，3213小時以上。
- 連絡資訊：*alan9956@gmail.com*

- **iPAS 巨量資料分析師 證照推廣**
- **iPAS 營運智慧分析師 證照推廣**

2.資料分析暨視覺化的心法

★★★資料分析架構→APC方法



1.

2.

3.

資料分析暨視覺化工具

- R - http://rwepa.blogspot.com/ 【免費】
- Python - http://rwepa.blogspot.com/2020/02/pythonprogramminglee.html 【免費】
- Julia - https://julialang.org/ 【免費】
- PowerBI- https://powerbi.microsoft.com/zh-tw/ 【免費/付費】
- Tableau - https://www.tableau.com/ 【14天版免費/付費】
- Excel 【付費】

大數據分析工具

- Microsoft Excel 2019: 104萬餘筆資料限制

	A	B	C	D	E	F	G
1	WEEK_END_DATE	STORE_NUM	UPC	UNITS	VISITS	HHS	SPEND
1048572	14-Jan-09	367	1111009477	13	13	13	18.07
1048573	14-Jan-09	367	1111009497	20	18	18	27.8
1048574	14-Jan-09	367	1111009507	14	14	14	19.32
1048575	14-Jan-09	367	1111035398	4	3	3	14
1048576	14-Jan-09	367	1111038078	3	3	3	7.5

1,048,576筆資料限制

- 免費: 核心程式 + 套件(模組) + IDE

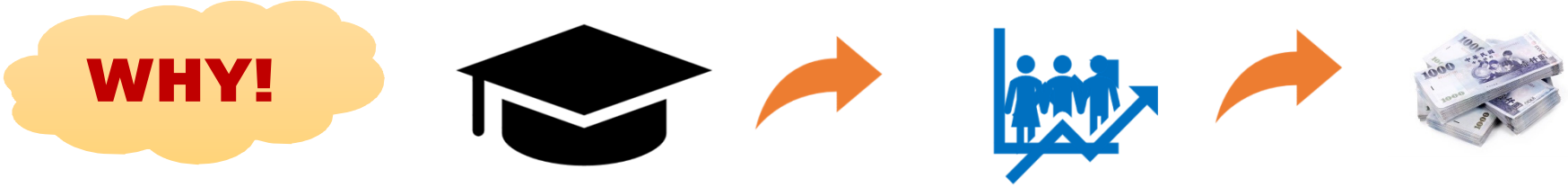
大數據分析免費工具

軟體	Python	R	Julia
Released	1991	2000	2012
用途	程式語言 系統結合	統計,繪圖,視覺化 程式語言	科學計算 程式語言
版本	自由軟體 物件導向	自由軟體 物件導向	自由軟體 物件導向
附加功能	免費模組	免費套件	免費模組
使用者	工科+ 商管	商管+ 工科	商管+ 工科

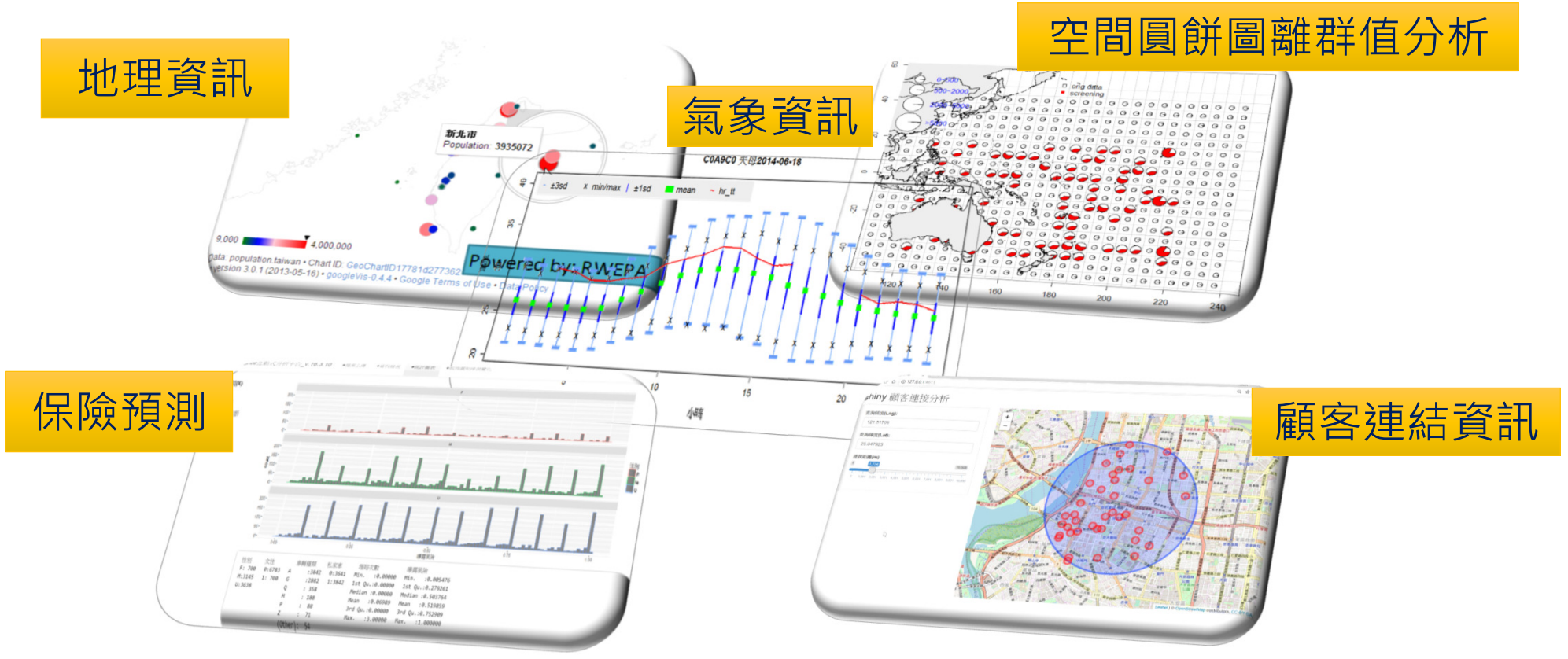
如何學習 Python?

- 熟悉教材內容
- 將教材內容的資料集改為工作資料集(企業, 學術)
- 遇到問題時, 想辦法尋找答案
- 掌握 APC方法
- 掌握 ①摘要 ②繪圖 ③建模
- 參考網路應用文章 (進階) & 學術論文

```
尋找答案 = {"方法1": "同事,同學,朋友等",
          "方法2": "Google",
          "方法3": "alan9956@gmail.com"}
```

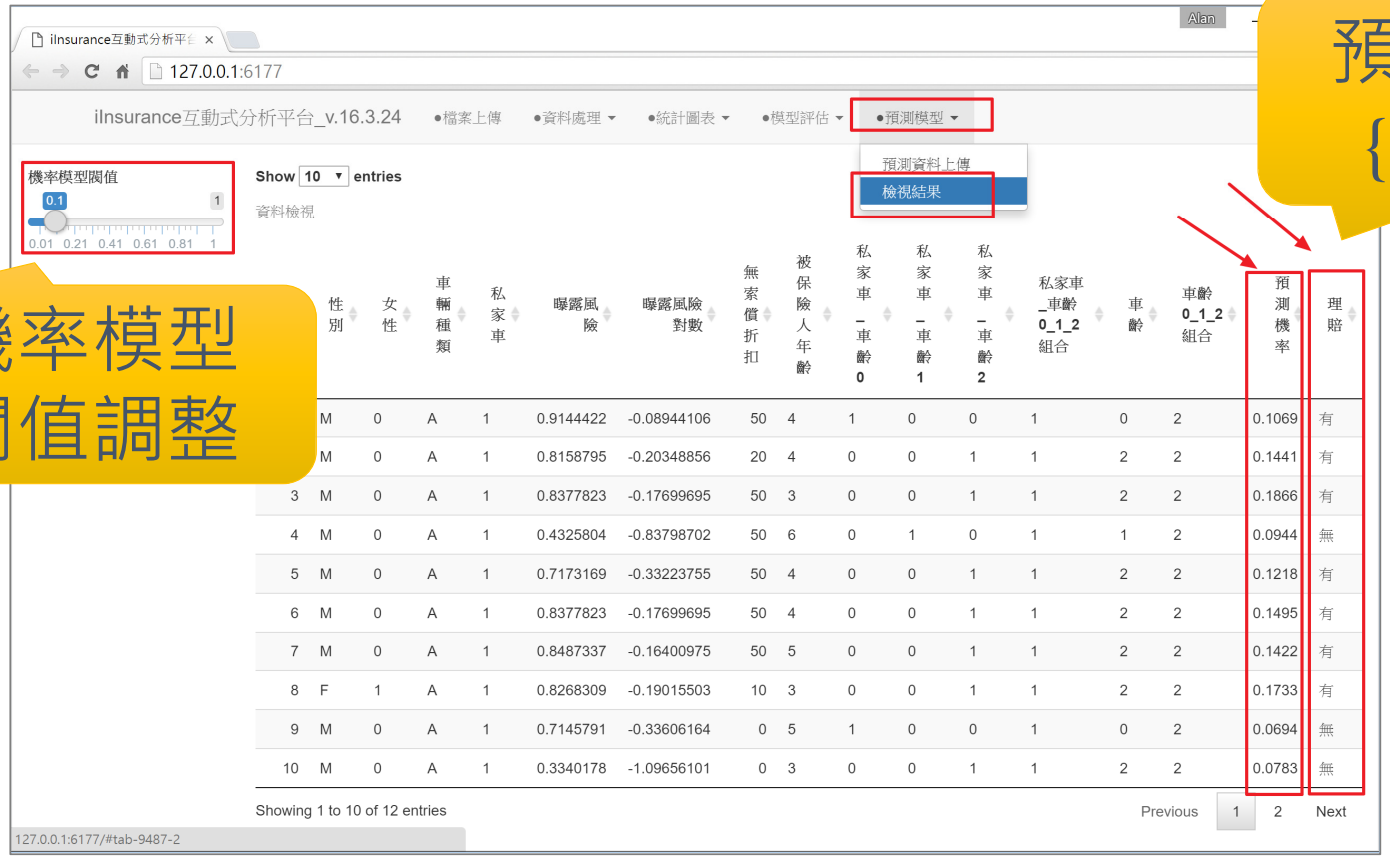


R+Shiny, Python+Streamlit 互動式平台

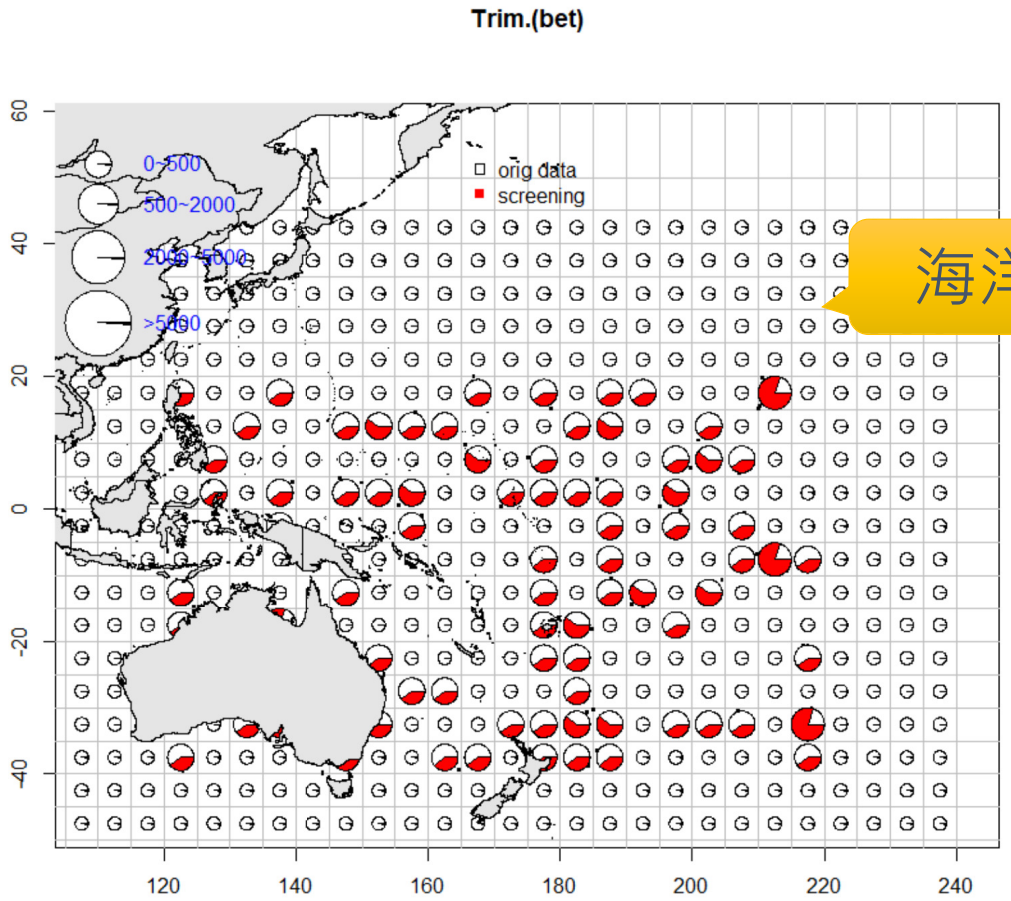


中央氣象局 1,600萬筆資料(14328個檔案)

保險預測模型

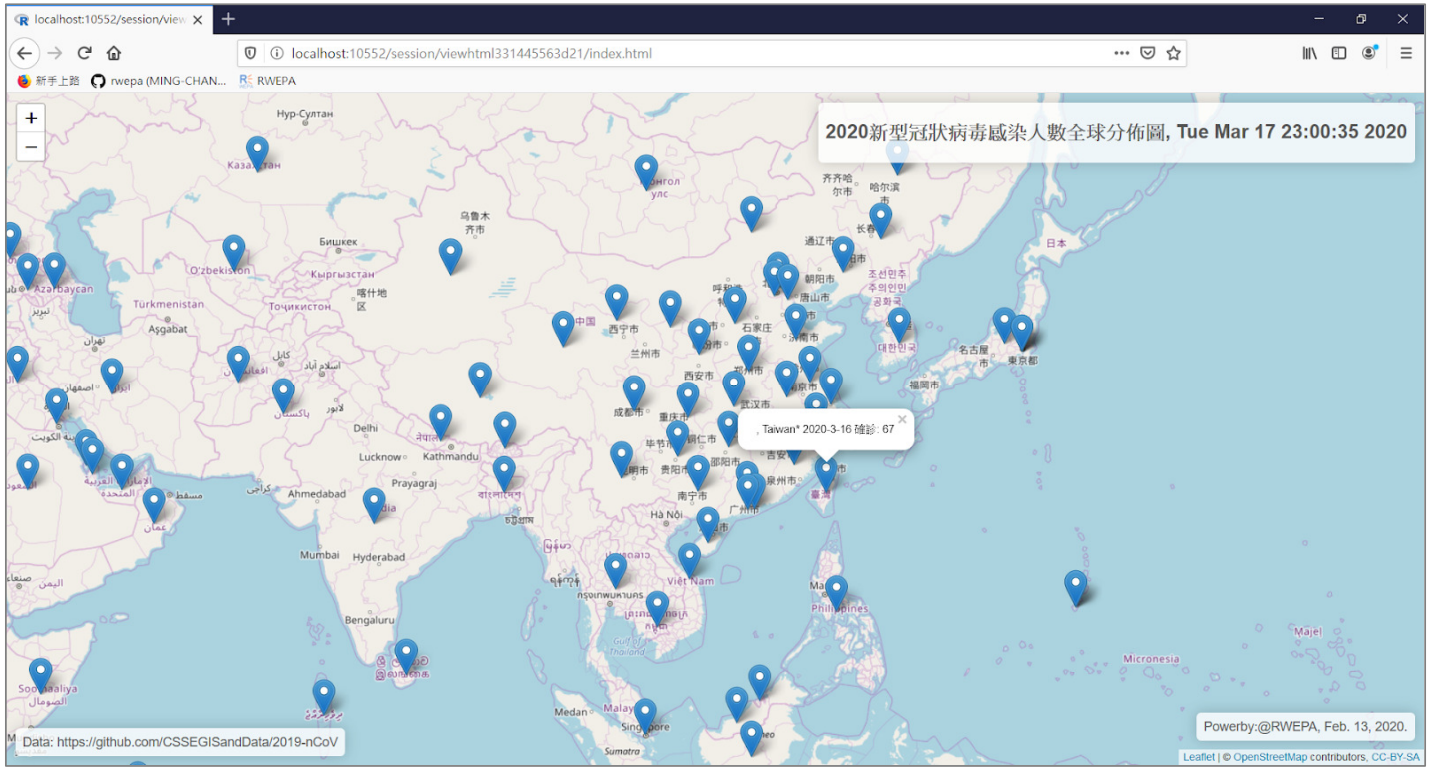

Alan
iInsurance互動式分析平台
127.0.0.1:6177
iInsurance互動式分析平台_v.16.3.24
●檔案上傳
●資料處理
●統計圖表
●模型評估
●預測模型
預測資料上傳
檢視結果
機率模型閥值
0.1
1
0.01 0.21 0.41 0.61 0.81 1
Show 10 entries
資料檢視
預測結果{有,無}
機率模型閥值調整
性別 女性 車輛種類 私家車 曝露風險 曝露風險對數 無索償折扣 被保險人年齡 私家車_車齡0 私家車_車齡1 私家車_車齡2 私家車_車齡0_1_2組合 車齡 車齡0_1_2組合 預測機率 理賠
1 M 0 A 1 0.9144422 -0.08944106 50 4 1 0 0 1 0 2 0.1069 有
2 M 0 A 1 0.8158795 -0.20348856 20 4 0 0 1 1 2 2 0.1441 有
3 M 0 A 1 0.8377823 -0.17699695 50 3 0 0 1 1 2 2 0.1866 有
4 M 0 A 1 0.4325804 -0.83798702 50 6 0 1 0 1 1 2 0.0944 無
5 M 0 A 1 0.7173169 -0.33223755 50 4 0 0 1 1 2 2 0.1218 有
6 M 0 A 1 0.8377823 -0.17699695 50 4 0 0 1 1 2 2 0.1495 有
7 M 0 A 1 0.8487337 -0.16400975 50 5 0 0 1 1 2 2 0.1422 有
8 F 1 A 1 0.8268309 -0.19015503 10 3 0 0 1 1 2 2 0.1733 有
9 M 0 A 1 0.7145791 -0.33606164 0 5 1 0 0 1 0 2 0.0694 無
10 M 0 A 1 0.3340178 -1.09656101 0 3 0 0 1 1 2 2 0.0783 無
Showing 1 to 10 of 12 entries
Previous 1 2 Next
127.0.0.1:6177/#tab-9487-2

空間圓餅圖離群值分析



海洋生態視覺化

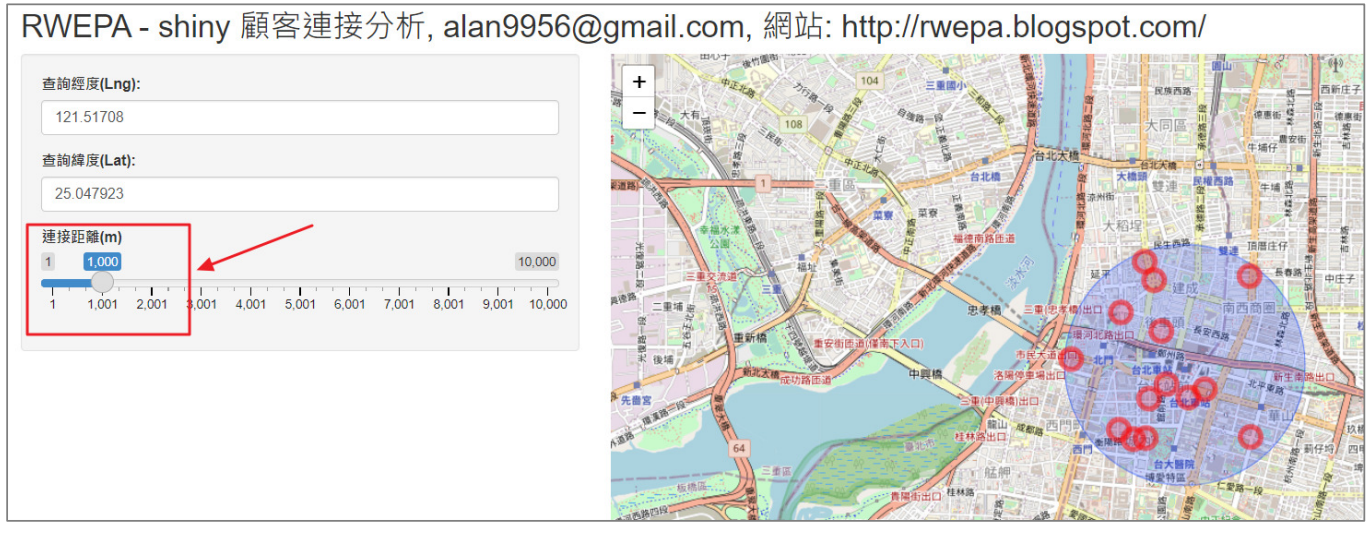
2020新型冠狀病毒視覺化



http://rwepa.blogspot.com/2020/02/2019nCoV.html

shiny 顧客連接分析

- https://rwepa.shinyapps.io/shinyCustomerConnect/

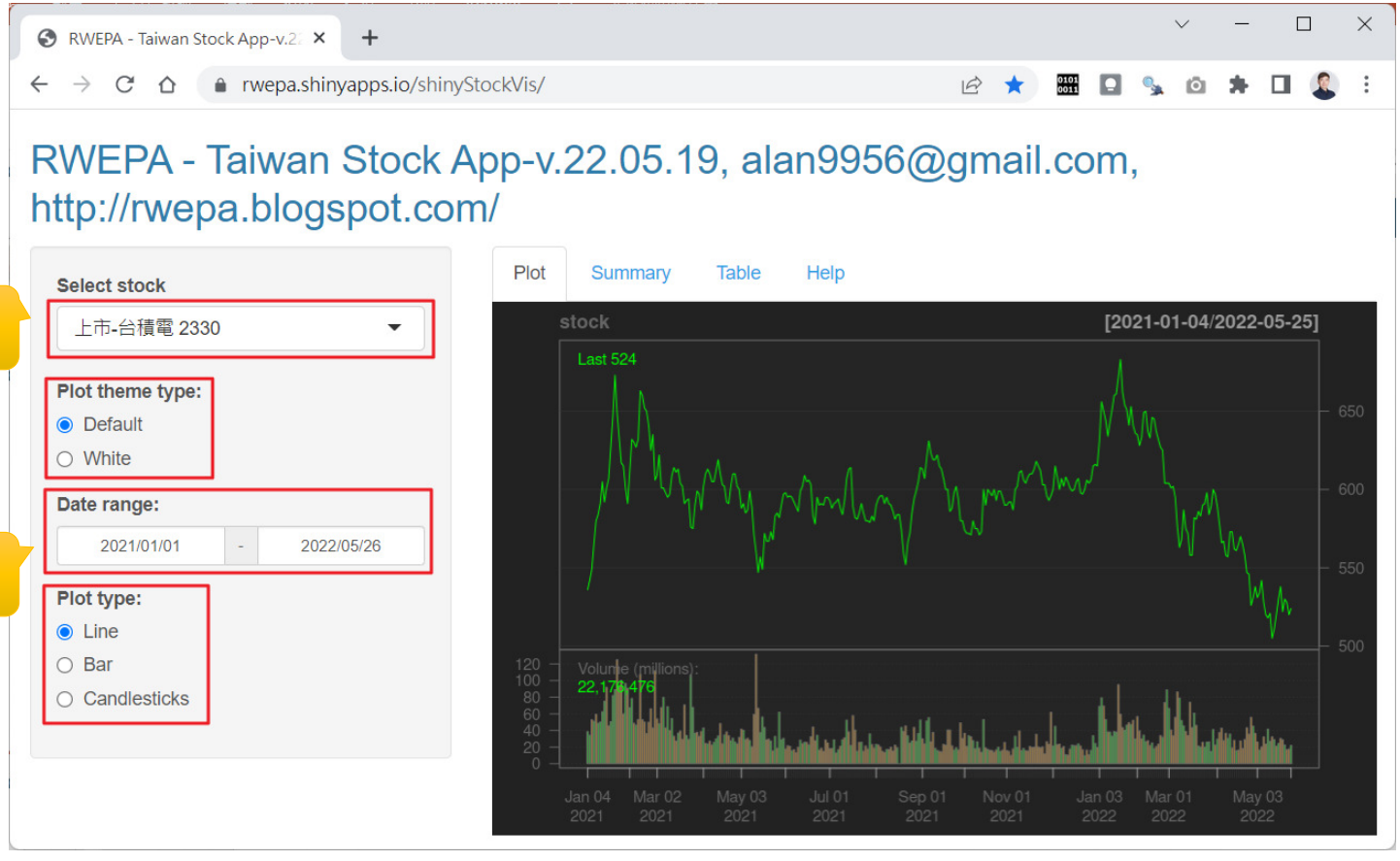


品質管制圖(quality control chart)應用

- 說明: http://rwepa.blogspot.com/2021/10/r-shiny-quality-control-chart.html
- 資料1: https://github.com/rwepa/shiny_spc/blob/main/data/spc_wafer_with_header.csv
- 資料2: https://github.com/rwepa/shiny_spc/blob/main/data/spc_pistonrings_without_header.csv
- 線上示範: https://rwepa.shinyapps.io/shiny_spc/

Taiwan Stock App

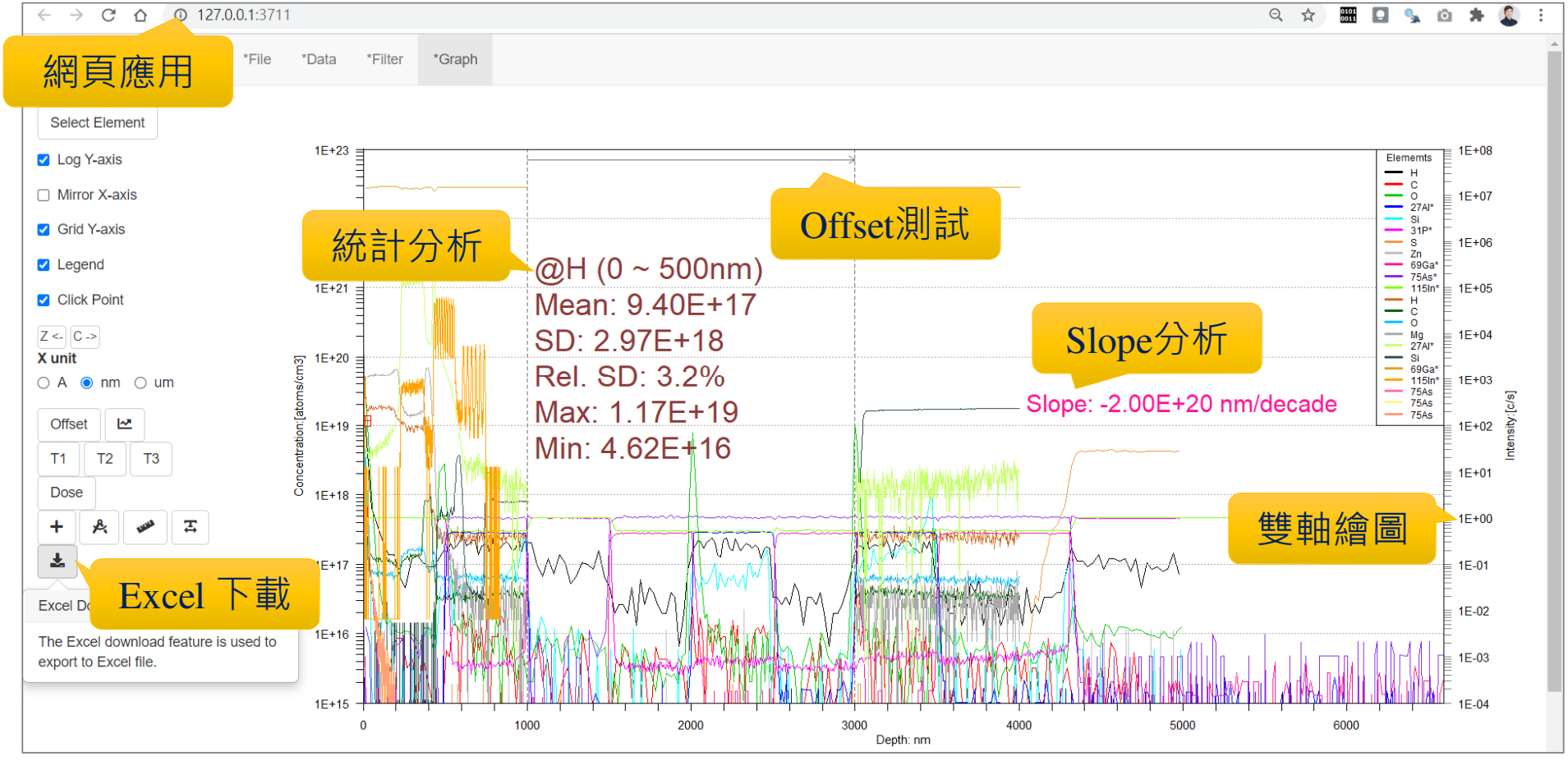
- https://rwepa.shinyapps.io/shinyStockVis/



下拉式選單

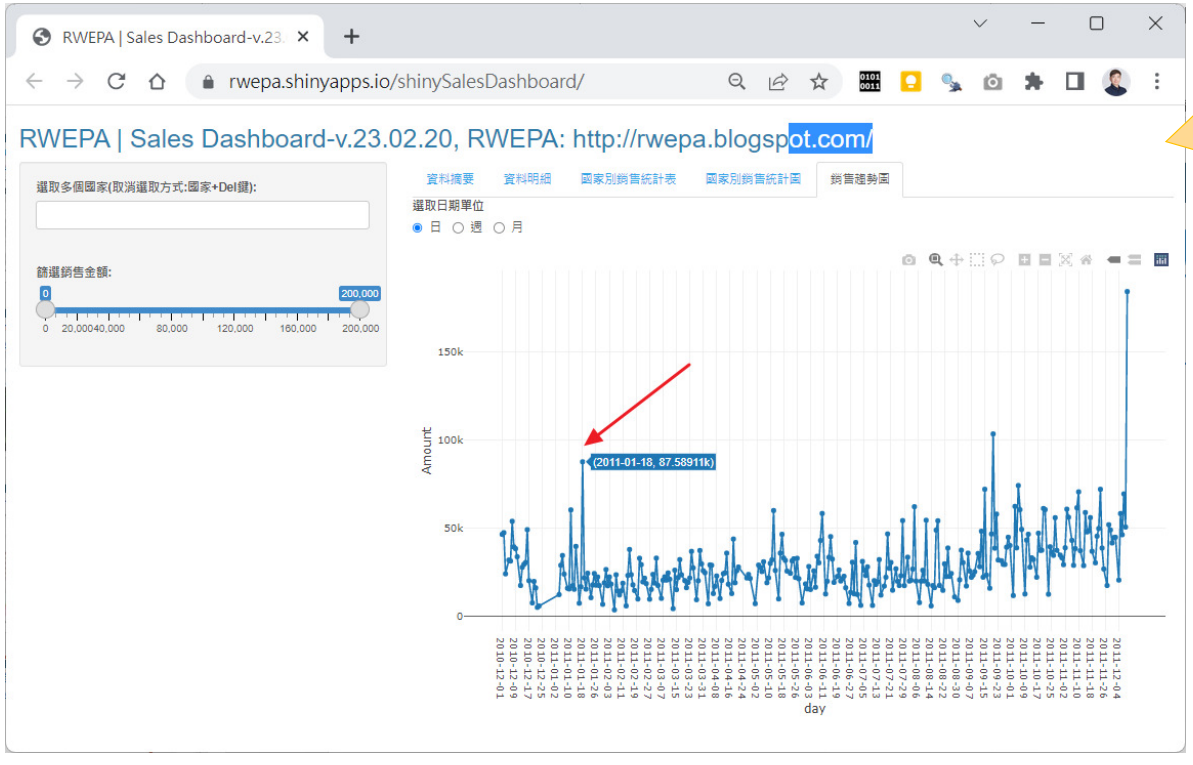
日期範圍

離子資料分析與視覺化應用



shiny銷售儀表板

- Shiny: https://rwepa.shinyapps.io/shinySalesDashboard/
- YouTube: https://youtu.be/4GgZlf8heQk



謝謝 ^_^

訂閱、讚、開啟小鈴鐺

shiny企業實務應用 第6集-小明算命師(下) - 第1季完結篇

- Ubuntu Shiny Server: https://shiny.rwepa.net/shiny-hr-teller/
- YouTube: https://youtu.be/rrD6KV3eV-w

下載 HTML, Word

Power BI 進行RFM分析

- 🌸YouTube：https://youtu.be/Lkr9HmzLTtg
- 🌸http://rwepa.blogspot.com/2023/07/rwepa-rfm-analysis-using-power-bi.html

Tableau - Superstore

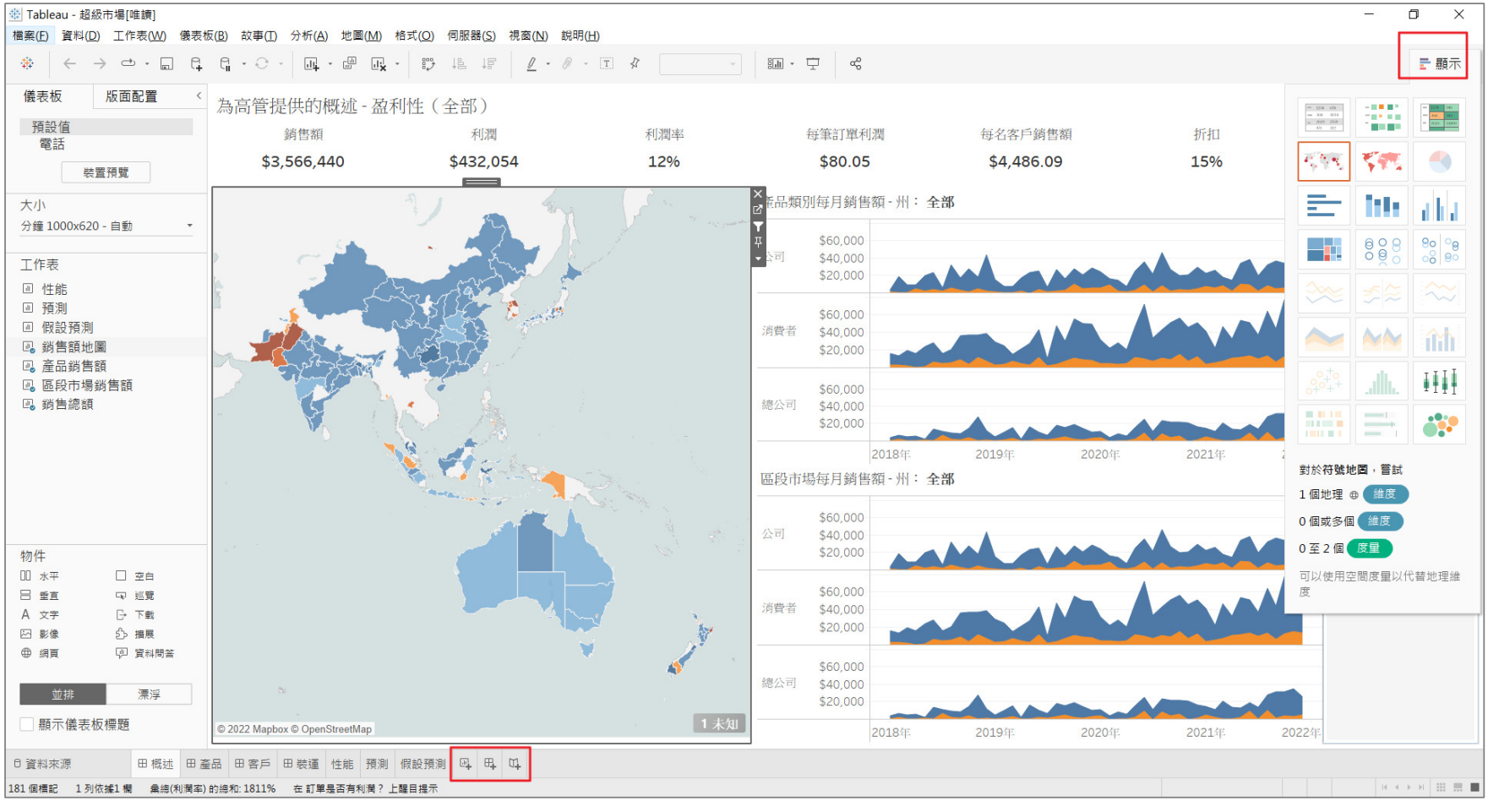
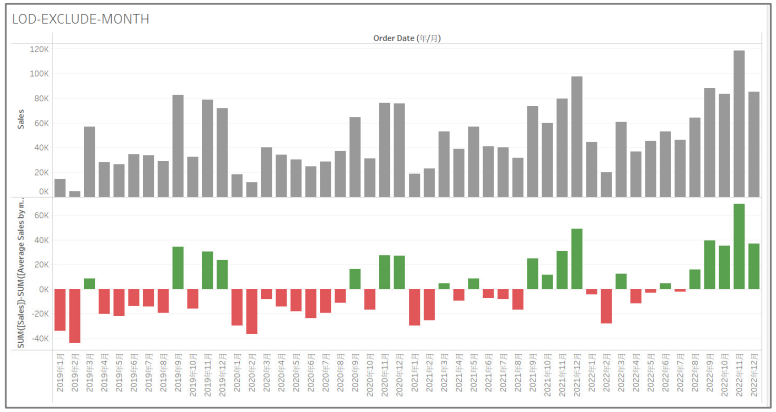
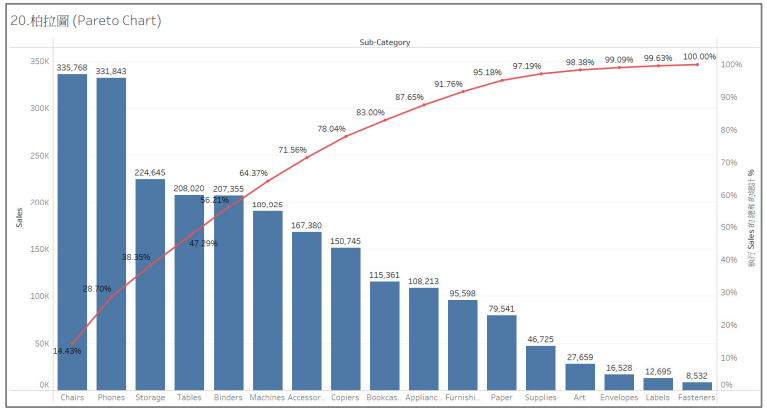
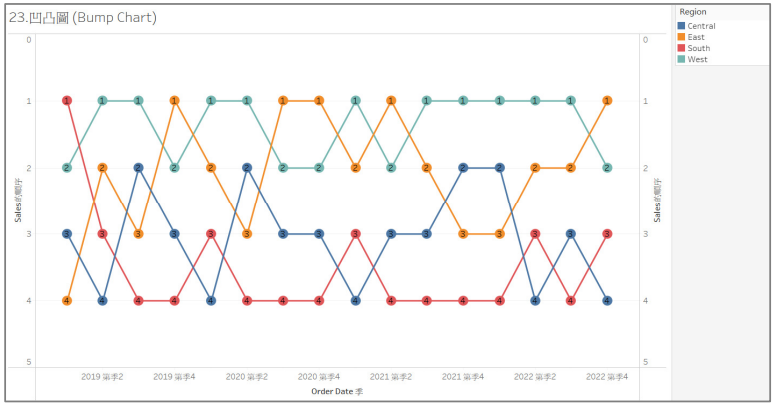


Tableau - 智慧製造應用

- https://github.com/rwepa/Talks
- https://public.tableau.com/app/profile/ming.chang.lee/vizzes







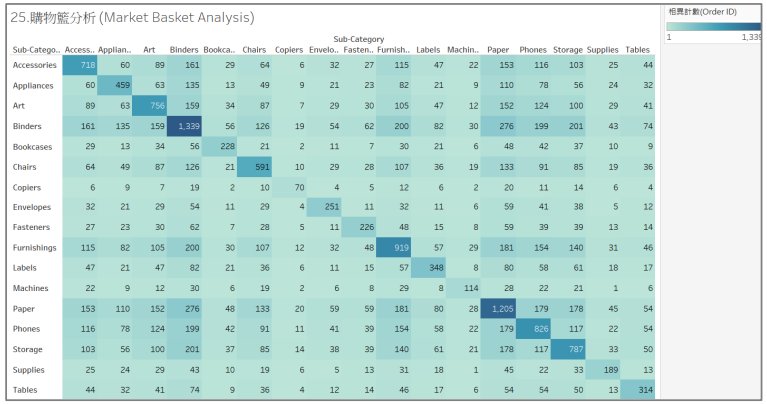
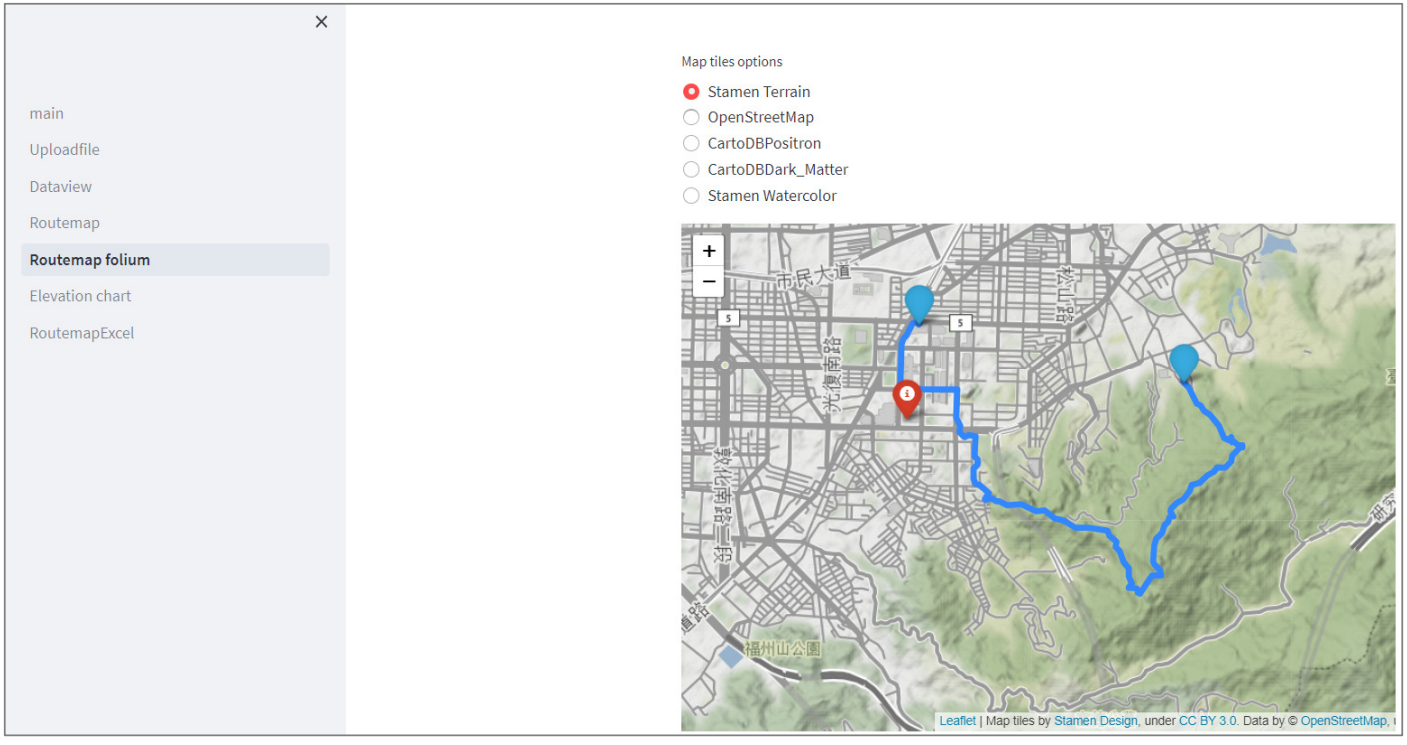


Tableau 教學

- Tableau資料分析與視覺化工具實作教師工作坊(初階)
  - https://github.com/rwepa/Talks/blob/main/tableau_tutorial_basic.pdf
- Tableau資料分析與視覺化工具實作教師工作坊(進階)
  - https://github.com/rwepa/Talks/blob/main/tableau_tutorial_advanced.pdf
- Tableau與R語言實務應用
  - https://github.com/rwepa/Talks/blob/main/tableau_r.pdf
- Tableau與MySQL資料庫實務應用
  - https://github.com/rwepa/Talks/blob/main/tableau_mysql.pdf

登山路線視覺化分析平台 (Python + Streamlit)

- YouTube：https://youtu.be/-_zghs2qrIg
- 系統展示 https://rwepa-climb.streamlit.app/



R 入門資料分析與視覺化應用(7小時28分鐘)

- https://mastertalks.tw/products/r?ref=MCLEE

課程提供教學範例的原始程式檔案與資料集
+中文字幕

- 主題

1. R, RStudio簡介與套件使用
2. 認識資料物件
3. 資料處理與分析
4. 資料視覺化應用

- 特色

1. 資料分析的**關鍵八步**
2. 提供必備**ggplot2**套件的應用知識與使用情境
3. 提供日期時間**zoo, xts套件**的整合應用操作
4. 提供**人力資源**資料與**銷售資料**，強化**實務資料**操作能力

R 商業預測應用(8小時53分鐘)

- https://mastertalks.tw/products/r-2?ref=MCLEE

課程提供教學範例的原始程式檔案與資料集
+中文字幕

- 主題

1. R，RStudio工具操作
2. 非監督式學習業預測
3. 監督式學習商業預測
4. 財金資料預測應用

- 特色

1. 採用**最有效率**方式學習大數據R語言，並應用於**職場資料分析**與**商業預測應用**
2. 提供**多元線性迴歸**的必備知識
3. 提供**財金資料商業預測應用**的基礎與進階必學技能
4. 提供學員人力資源資料與**台指期tick資料**預測演練

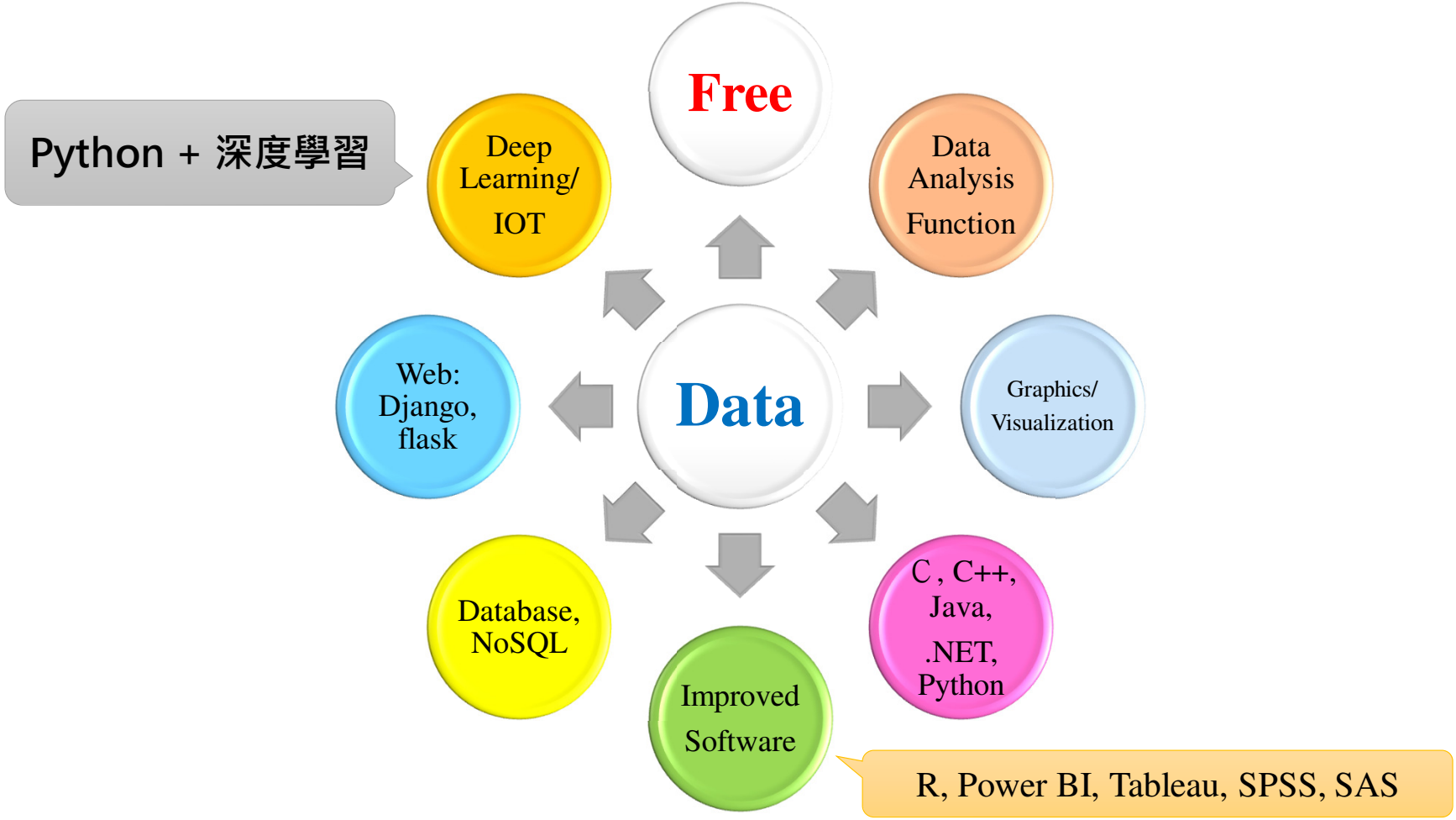
學習目標

3.Python 簡介

Python 簡介

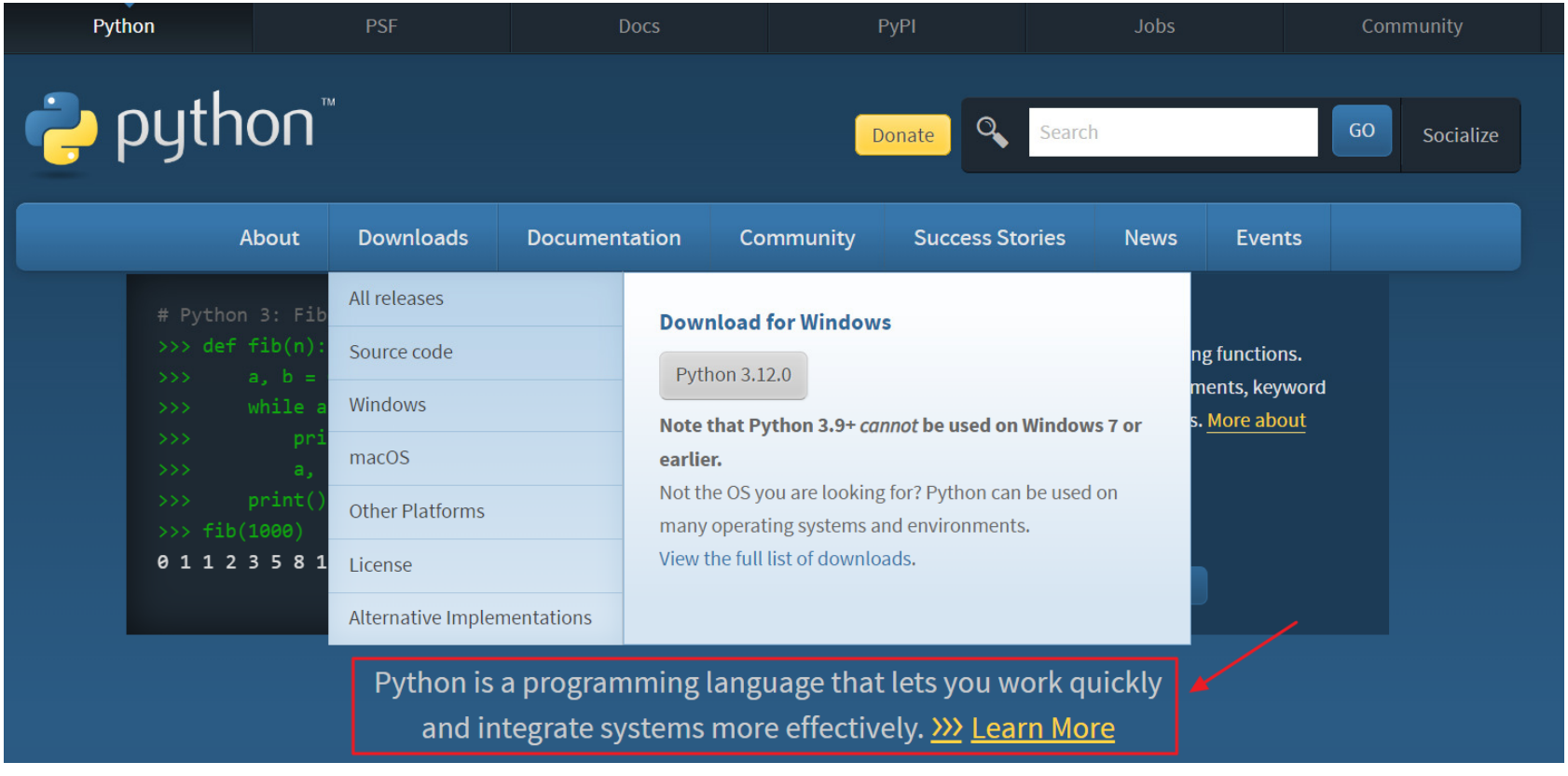
- 吉多·范羅蘇姆 (Guido van Rossum) 在1989年的聖誕節期間研發 Python 語言。
  - https://en.wikipedia.org/wiki/Guido_van_Rossum
  - Python 3.12.0 - 2023年10月15日
- 特性:
  - 跨平台
  - 開放性
  - 易讀性 (使用 : 符號)
  - 動態語言
  - 直譯語言
  - 豐富套件(模組)
  - 其他語言結合(C, C++), 例: Cython 編譯成執行檔(.exe)
  - 物件導向程式語言

Python-八大功能


Python + 深度學習
Free
Deep Learning/ IOT
Data Analysis Function
Web: Django, flask
Data
Graphics/ Visualization
Database, NoSQL
C , C++, Java, .NET, Python
Improved Software
R, Power BI, Tableau, SPSS, SAS

Python 下載

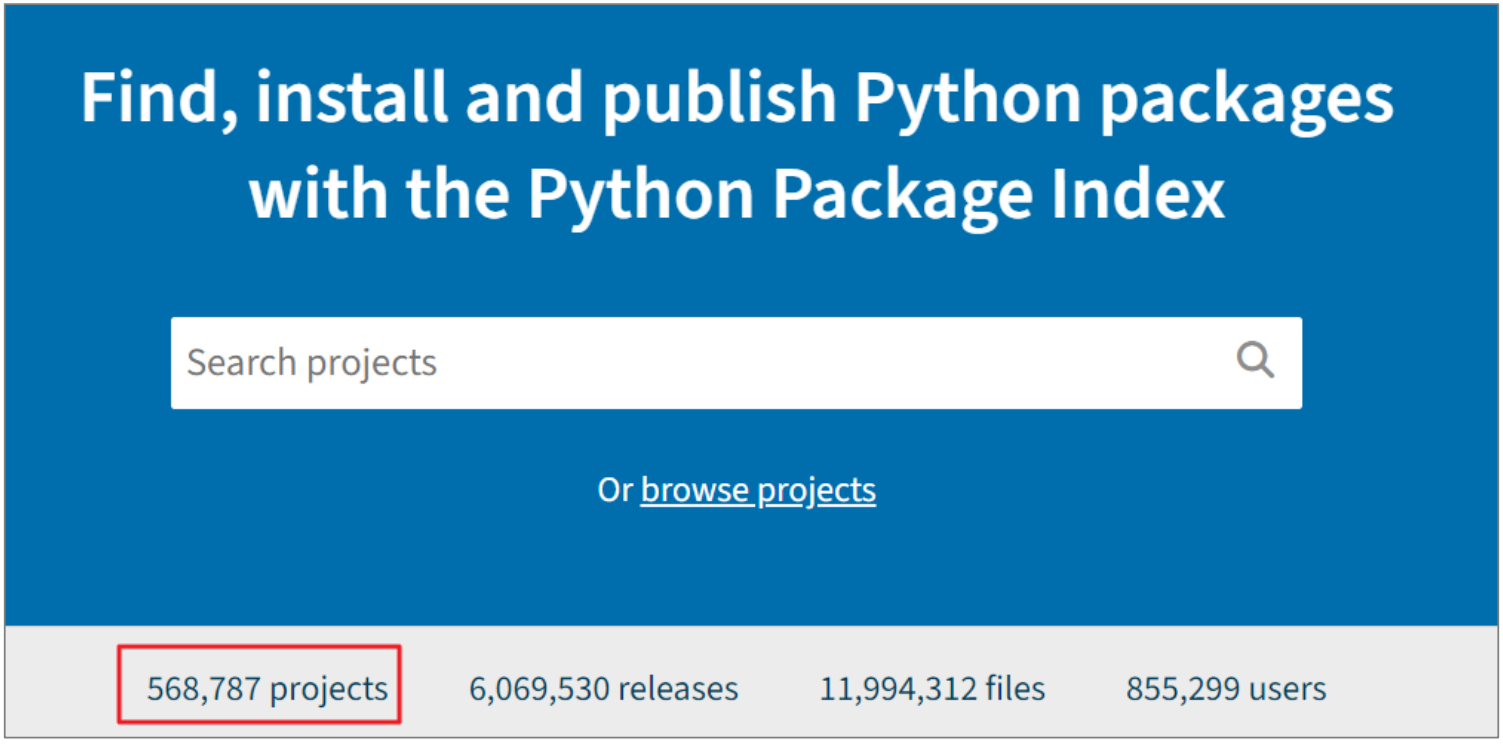
- https://www.python.org/



PyPI (Python Package Index)

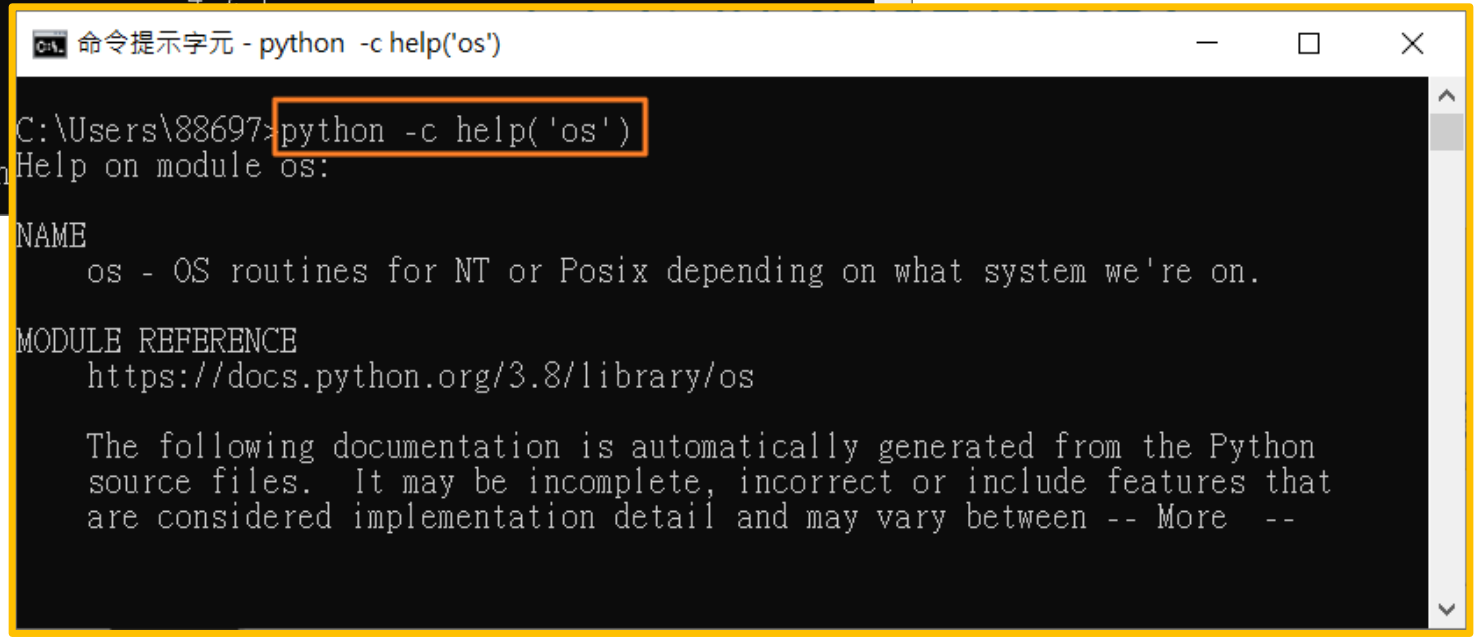
- https://pypi.org/
- 56萬專案總表

(2024.9.13)



已安裝模組(pip)

- **pip list**
- 模組說明
  python -c help('模組')
- 按 **Q** 關閉說明



Python IDE

- Anaconda, 包括 Spyder
  - https://www.anaconda.com/
- PyCharm
  - https://www.jetbrains.com/pycharm/
- RStudio
  - https://www.rstudio.com/products/rstudio/
- Visual Studio Code
  - https://code.visualstudio.com/docs/python/python-tutorial
- Google Colab (全名為「Colaboratory」)
  - https://colab.research.google.com/
- Positron

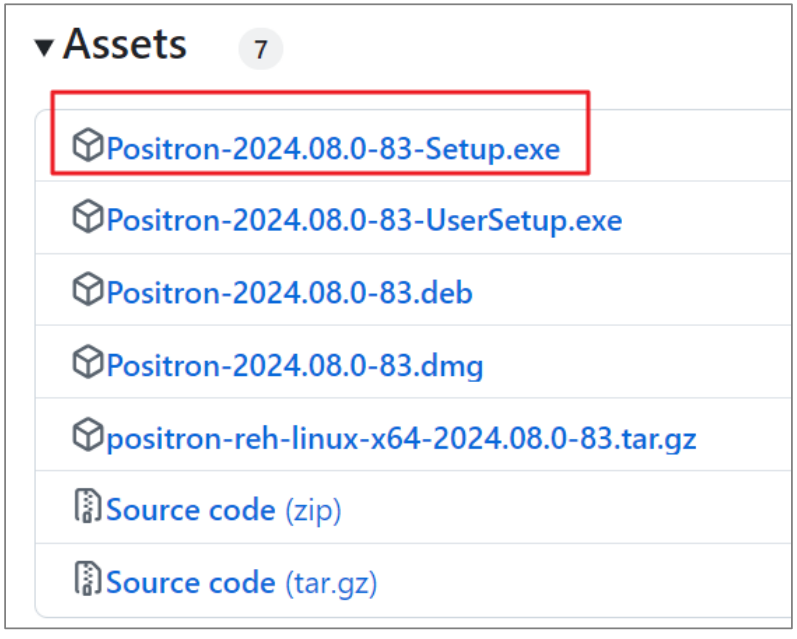
上課軟體 (Spyder)

IDE 整合開發環境-
Integrated
Development
Environment

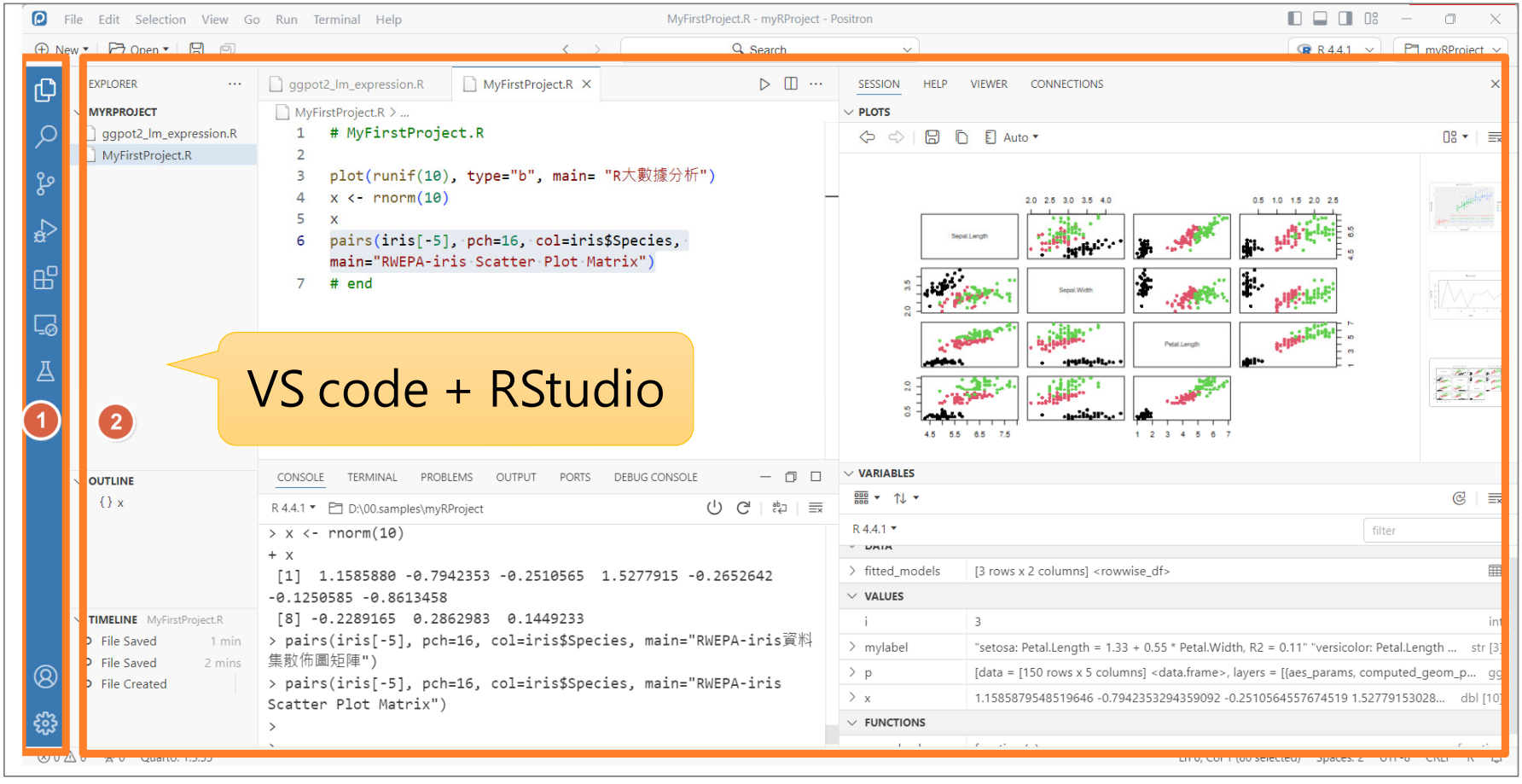
Positron (整合 R, Python, Jupyter)

Positron IDE, Since 2024.6

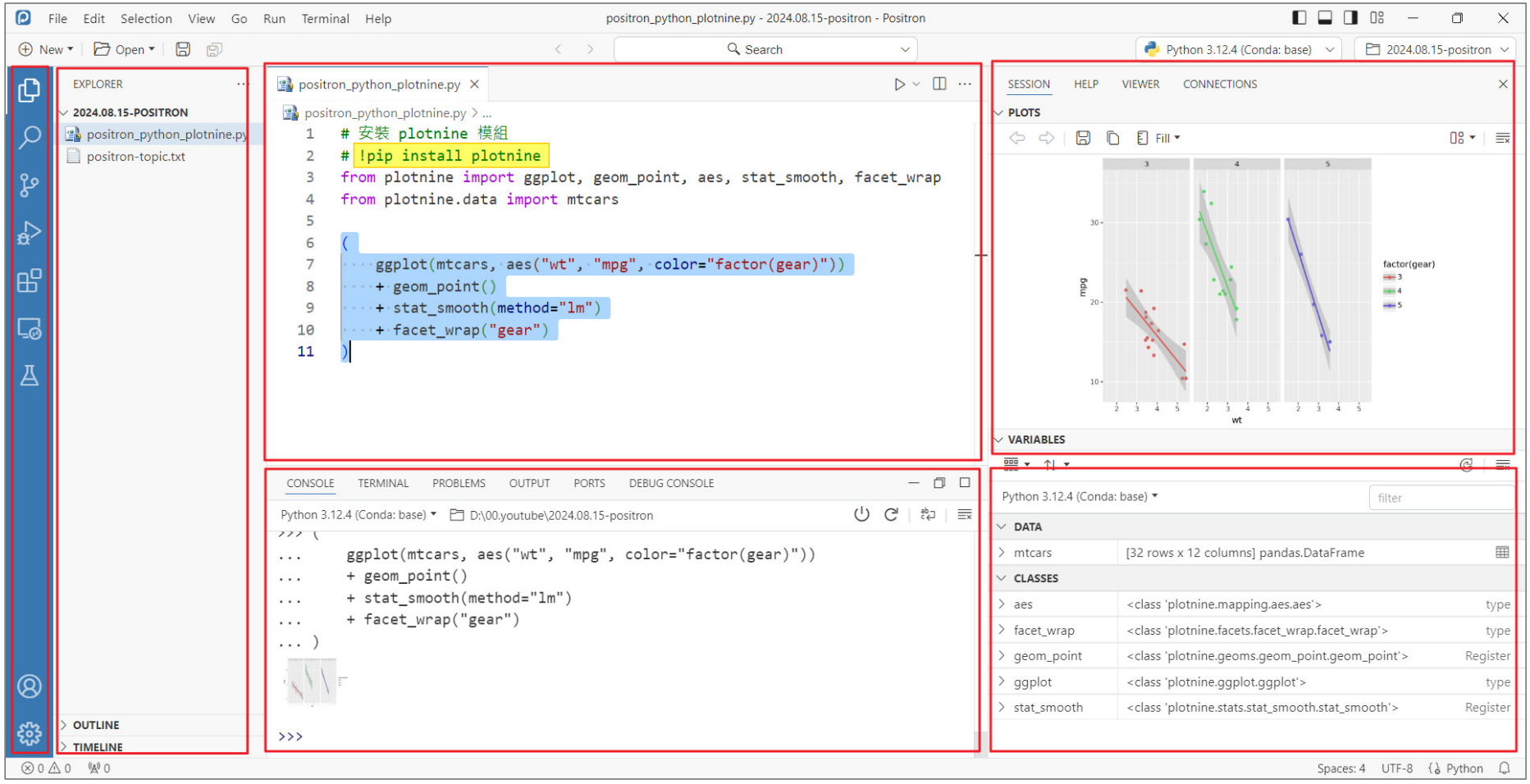
- 首頁: https://github.com/posit-dev/positron
- 下載: https://github.com/posit-dev/positron/releases



Positron+R 執行畫面



Positron+Python 執行畫面



4.Anaconda 簡介

Anaconda 特性

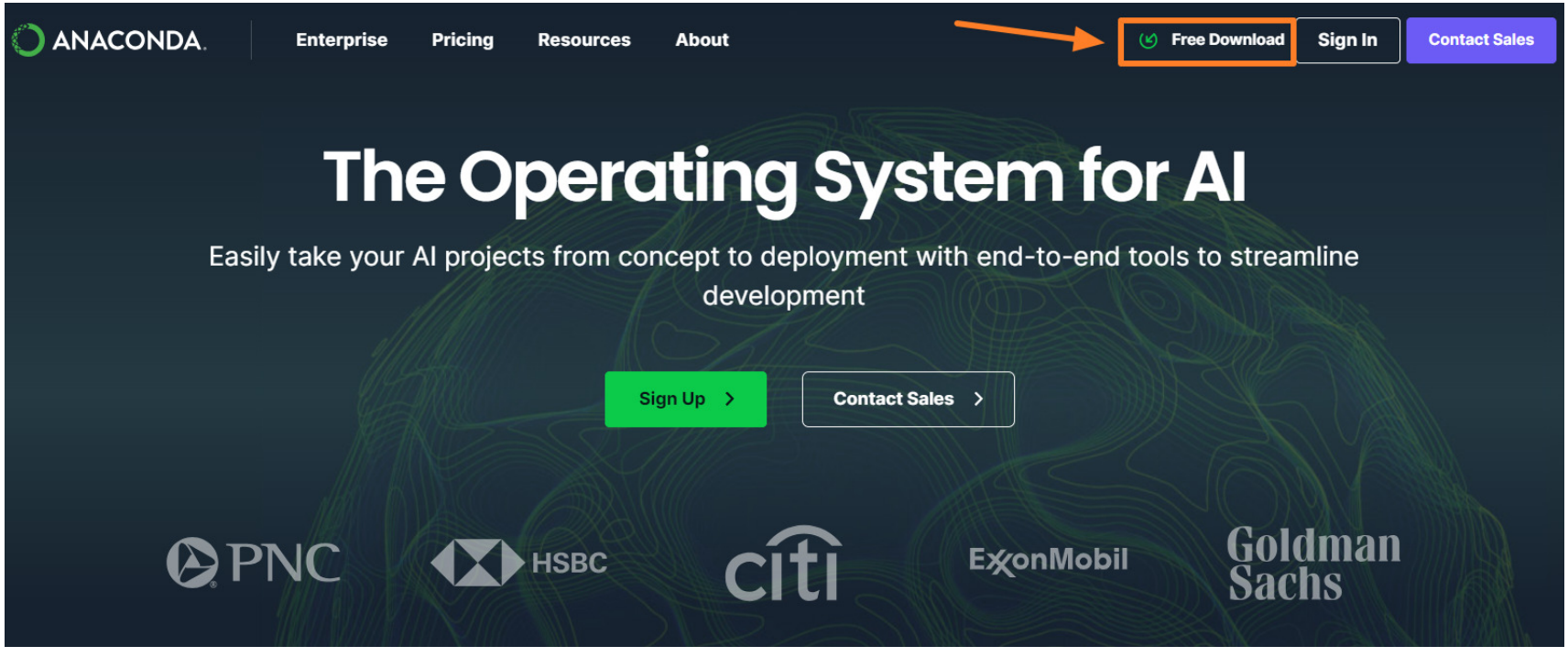
- Anaconda是一個免費、易於安裝/管理並支援Python語言
- 支援7500個以上的資料科學模組 (package)
- 支援模組的下載, 安裝, 更新
- 支援 Spyder (支援 Python IDE)
- 支援 jupyter notebook
- 支援 Windows、macOS, Linux

Anaconda 特性 (續)

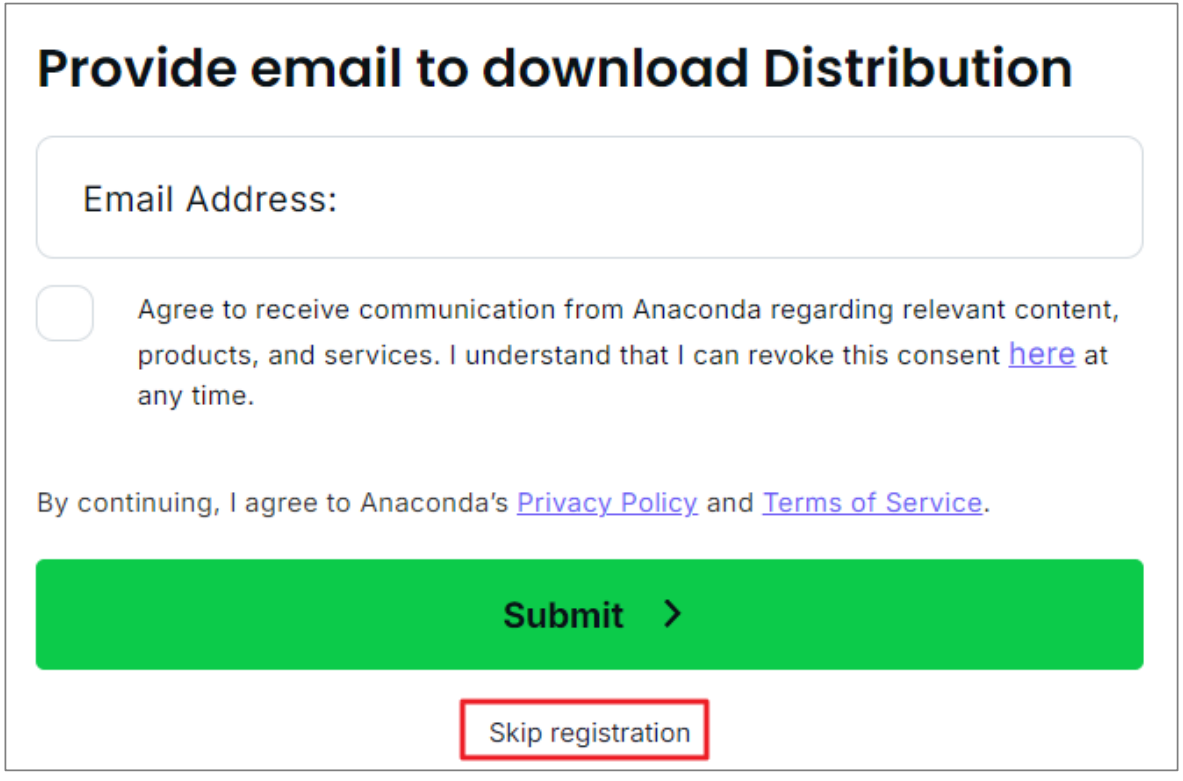
source: https://www.anaconda.com/distribution/

Anaconda 下載

- https://www.anaconda.com/

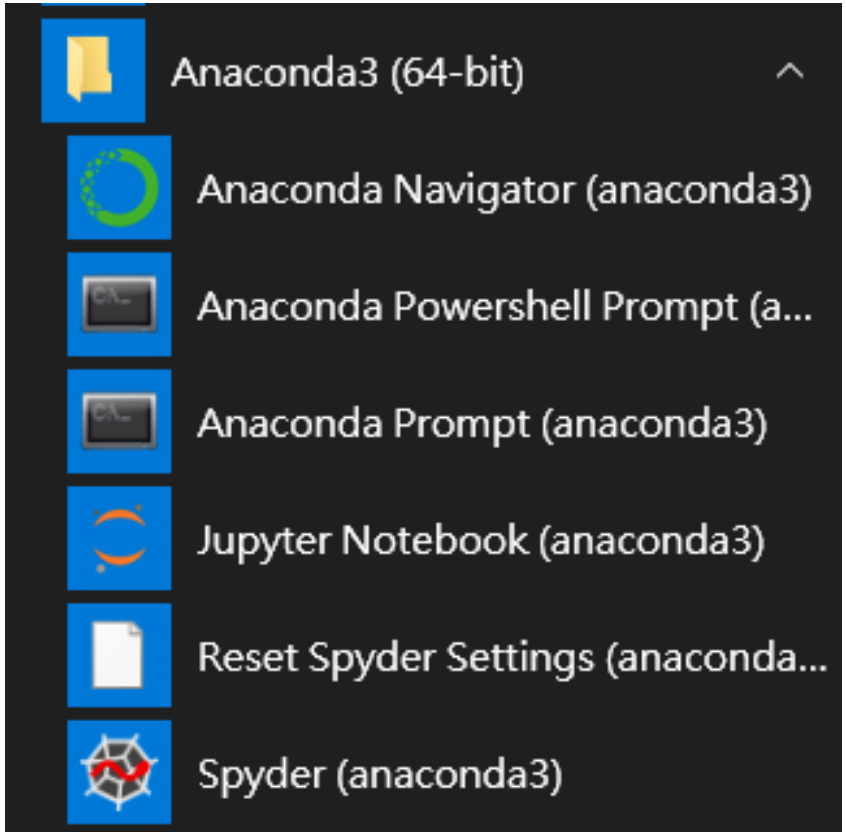


Anaconda 下載 (續)



Anaconda 下載 (續)

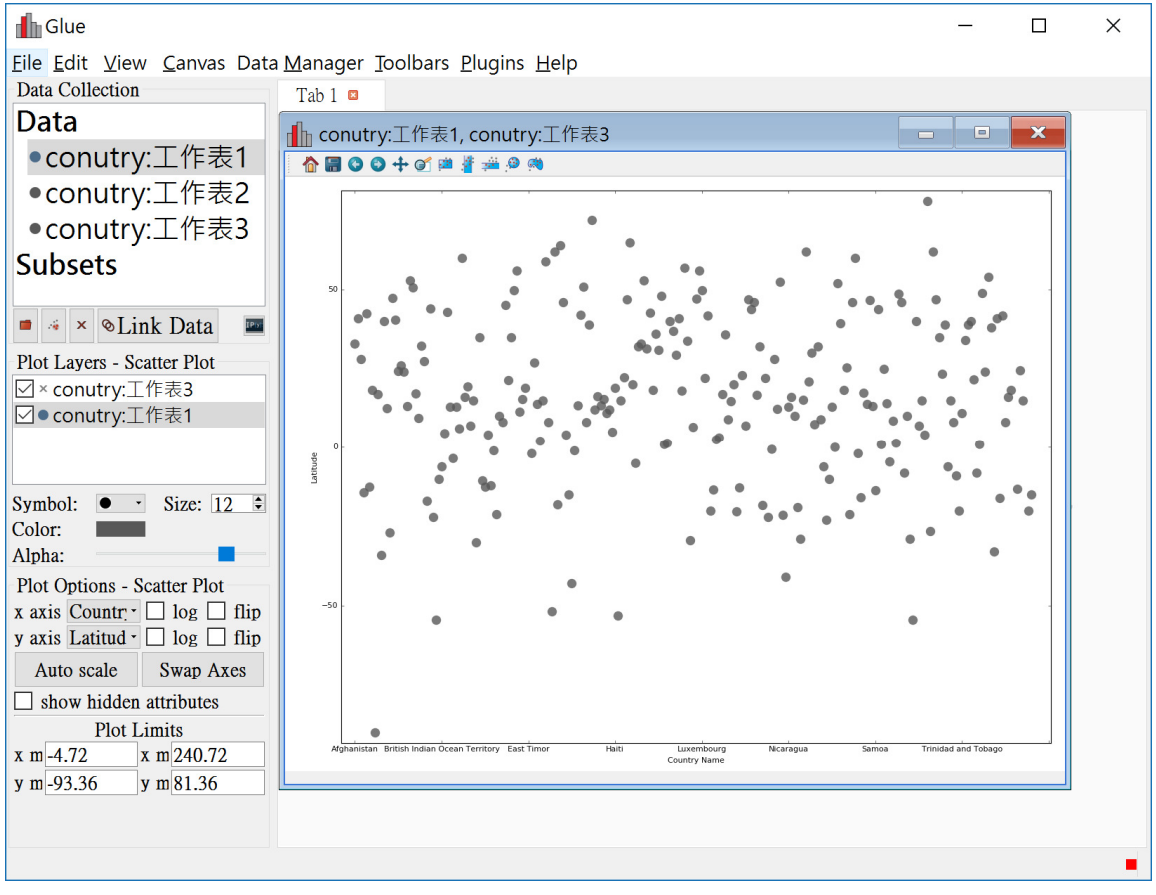
Anaconda3 (64-bit)



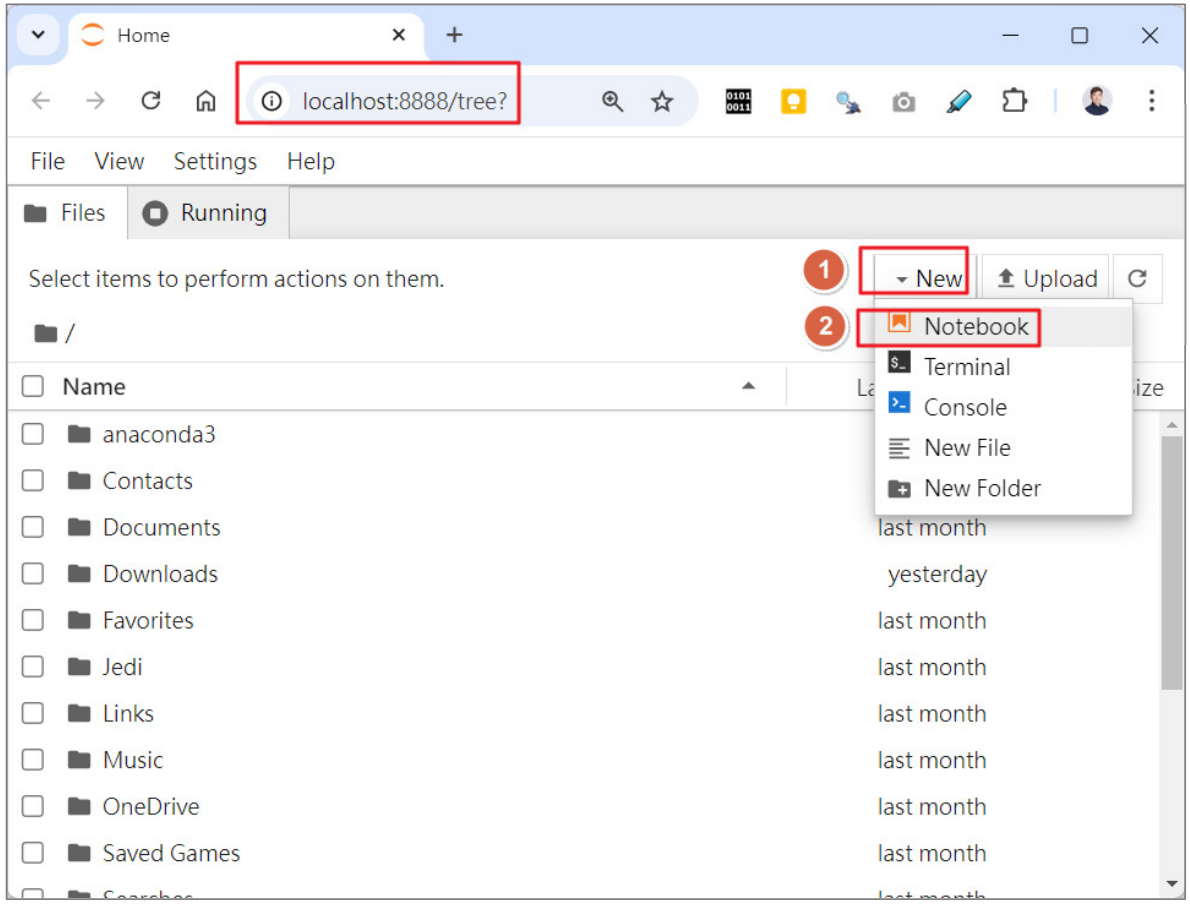
已安裝
6個元件

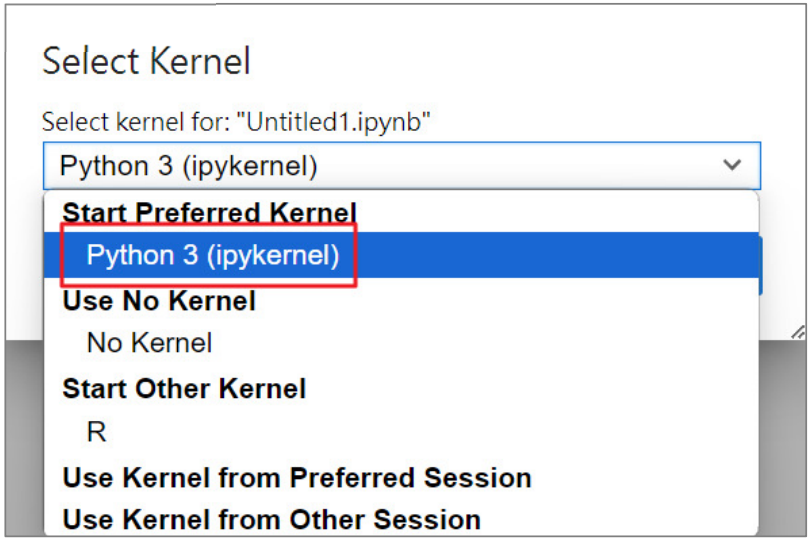
Anaconda Navigator

glueviz 視覺化


Glue
File Edit View Canvas Data Manager Toolbars Plugins Help
Data Collection
Data
conutry:工作表1
conutry:工作表2
conutry:工作表3
Subsets
Link Data
Plot Layers - Scatter Plot
conutry:工作表3
conutry:工作表1
Symbol:
Size: 12
Color:
Alpha:
Plot Options - Scatter Plot
x axis Countr log flip
y axis Latitud log flip
Auto scale
Swap Axes
show hidden attributes
Plot Limits
x m -4.72
x m 240.72
y m -93.36
y m 81.36
Tab 1
conutry:工作表1, conutry:工作表3
Latitude
50
0
-50
Afghanistan
British Indian Ocean Territory
East Timor
Haiti
Luxembourg
Nicaragua
Samoa
Trinidad and Tobago
Country Name

jupyter notebook \ New \ Python3



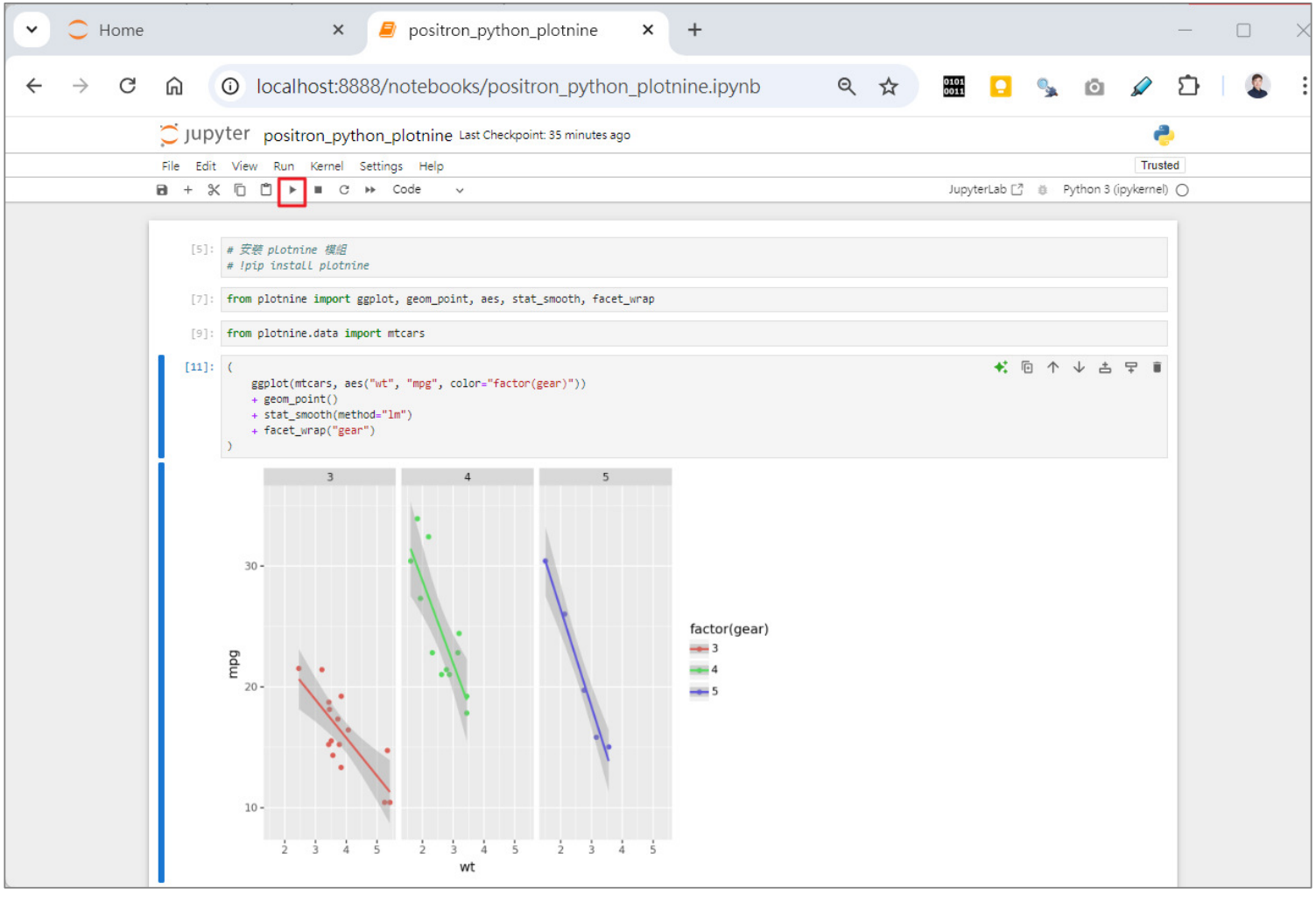


positron_python_plotnine.ipynb

```
# 安裝 plotnine 模組
# !pip install plotnine

from plotnine import ggplot, geom_point, aes, stat_smooth, facet_wrap

from plotnine.data import mtcars

(
    ggplot(mtcars, aes("wt", "mpg", color="factor(gear)"))
    + geom_point()
    + stat_smooth(method="lm")
    + facet_wrap("gear")
)
```

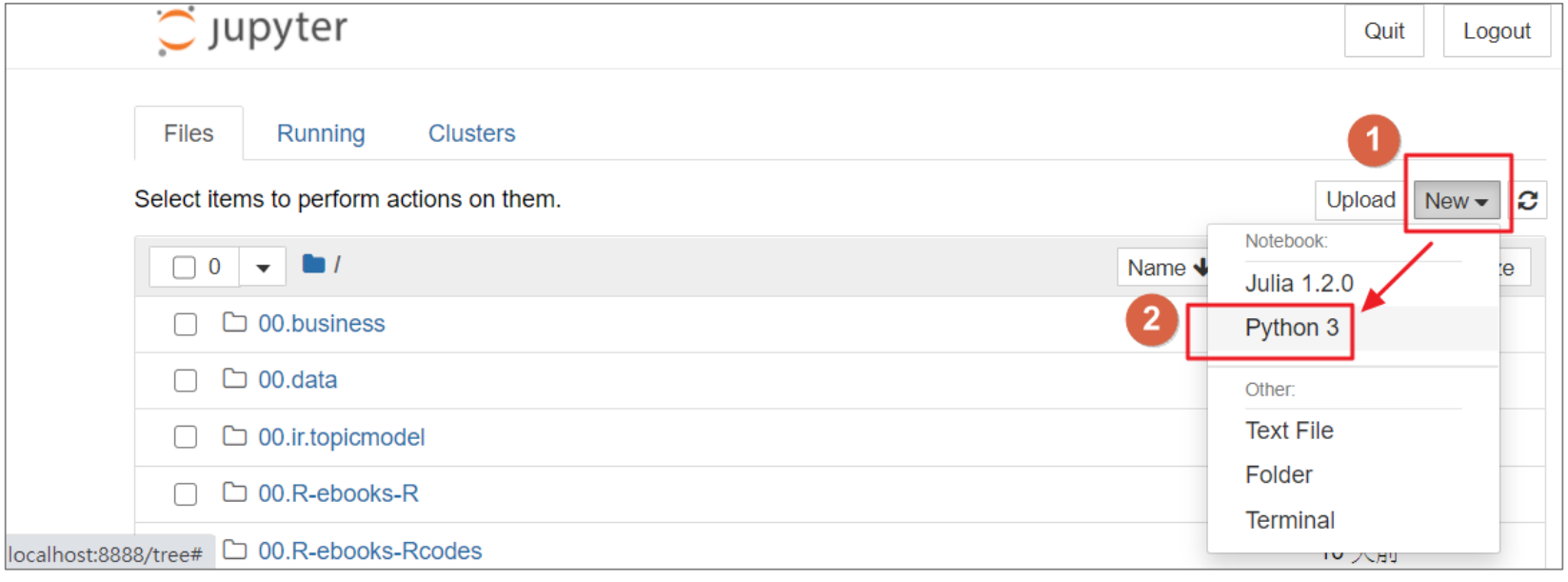
實作練習

jupyter notebook – 完成版



jupyter notebook – 更改預設目錄

- cd C:\
- jupyter-notebook



jupyter Notebook 快速鍵

- 按 [Esc] cell旁邊為藍色:
  - 按下x：刪除當前選擇的cell
  - 按下a：在當前選擇的上方新增一個cell
  - 按下b：在當前選擇的下方新增一個cell
  - 按下Shift + Enter：執行當前的cell並且選到下一個cell
  - 按下Ctrl + Enter：執行當前cell
  - 按下M：轉成markerdown模式，可以看到紅色框框內容從code變成markerdown

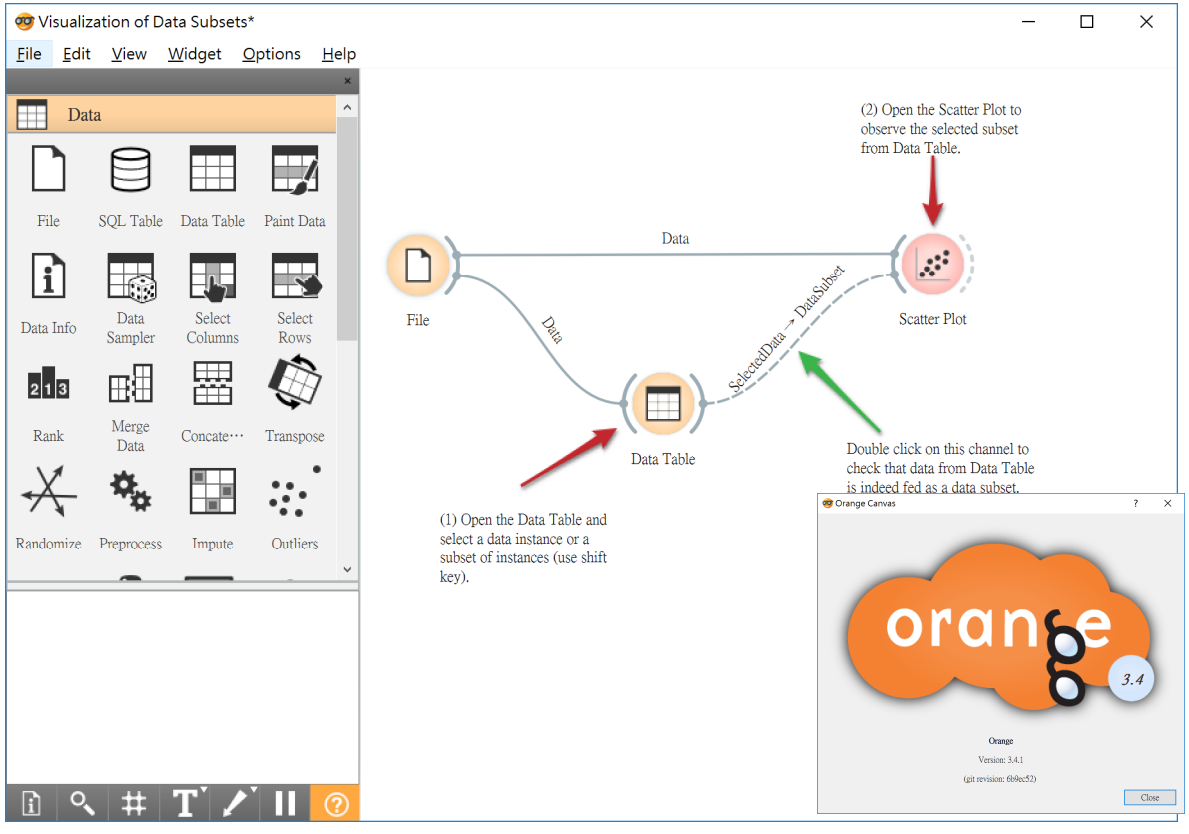
實作練習

開啟 → Python程式設計-李明昌.ipynb, 數學式主題

- http://rwepa.blogspot.com/2020/02/pythonprogramminglee.html
- PDF P.19

Orange

- 使用命令提示列 開啟 Orange: python -m Orange.canvas

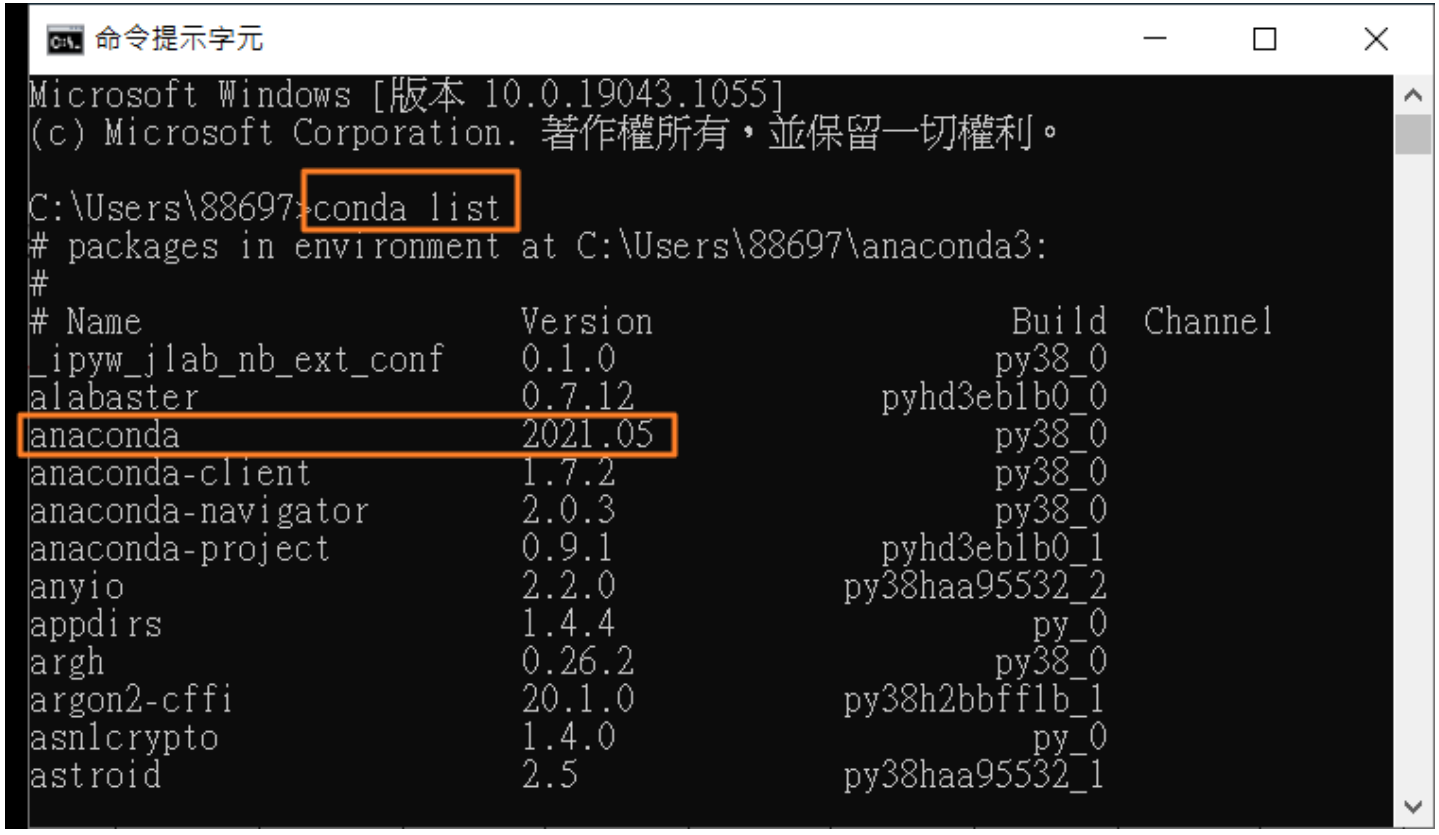


Orange - Scatter Plot

- Python Orange - 關聯規則教學
  - YouTube: https://youtu.be/rh5GxJamtNg
  - https://rwepa.blogspot.com/2022/07/python-orange-associate-tutorial.html

Anaconda 模組管理

- 顯示已安裝模組
  conda list

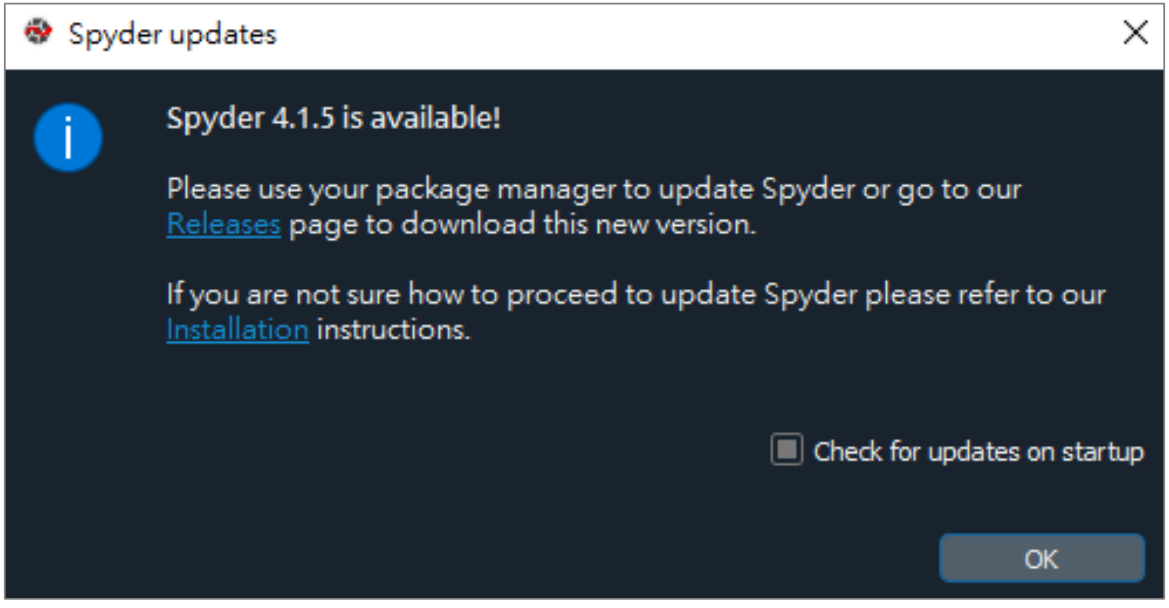


Anaconda 模組管理(續)

- 尋找官網套件
  conda search matplotlib
- 安裝模組
  conda install 模組名稱
- 更新模組
  conda update 模組名稱

Spyder 軟體簡介

Spyder 更新



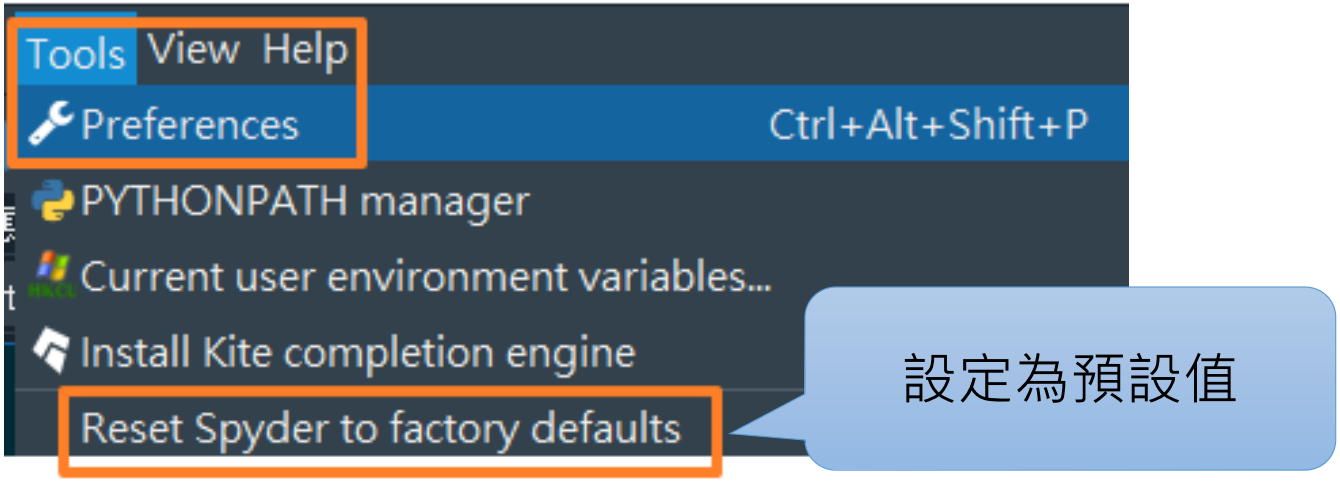
(最新版 5.5.1)

- Anaconda 整體更新　　conda update anaconda
- Spyder 更新　　conda update spyder

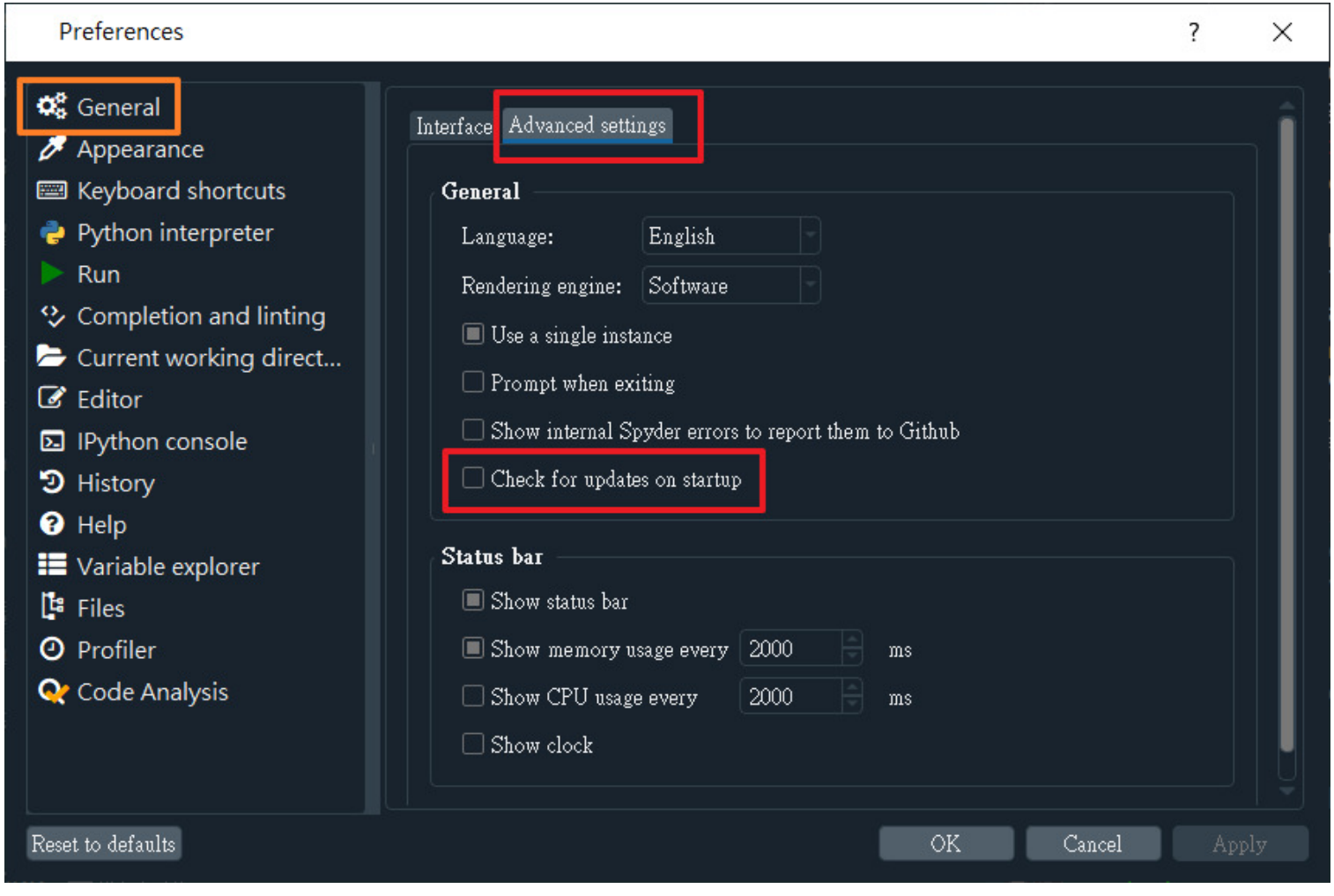
Spyder 畫面


功能表列
工具列
切換窗格
Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Tools View Help
C:\pythondata
Editor - C:\pythondata\python.sklearn.讀我.py
python.sklearn.讀我.py
type.operations.py
1
程式碼
物件瀏覽
2
Object inspector
Source Console Object
Object inspector Variable explorer File explorer
IPython console
Console 1/A
Python 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul 5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 4.2.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
IPython
3
Console History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 8 Column: 4 Memory: 40 %

喜好設定 Tools\Preferences

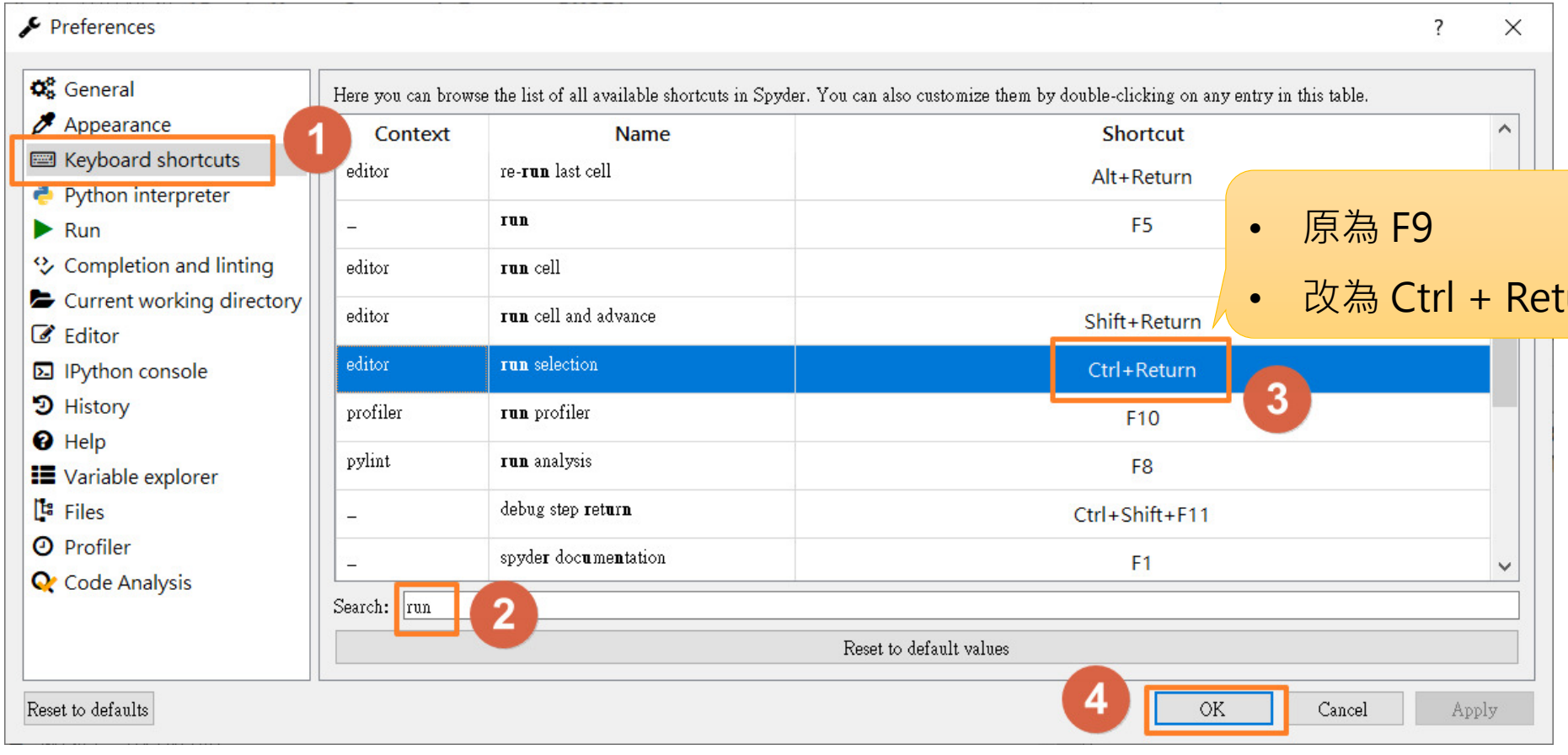


Preferences – General – Advanced setting



Preferences – Appearance

Preferences – Keyboard shortcuts

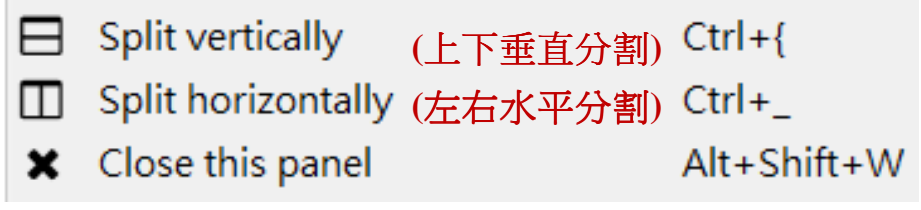


Spyder 好用的快速鍵

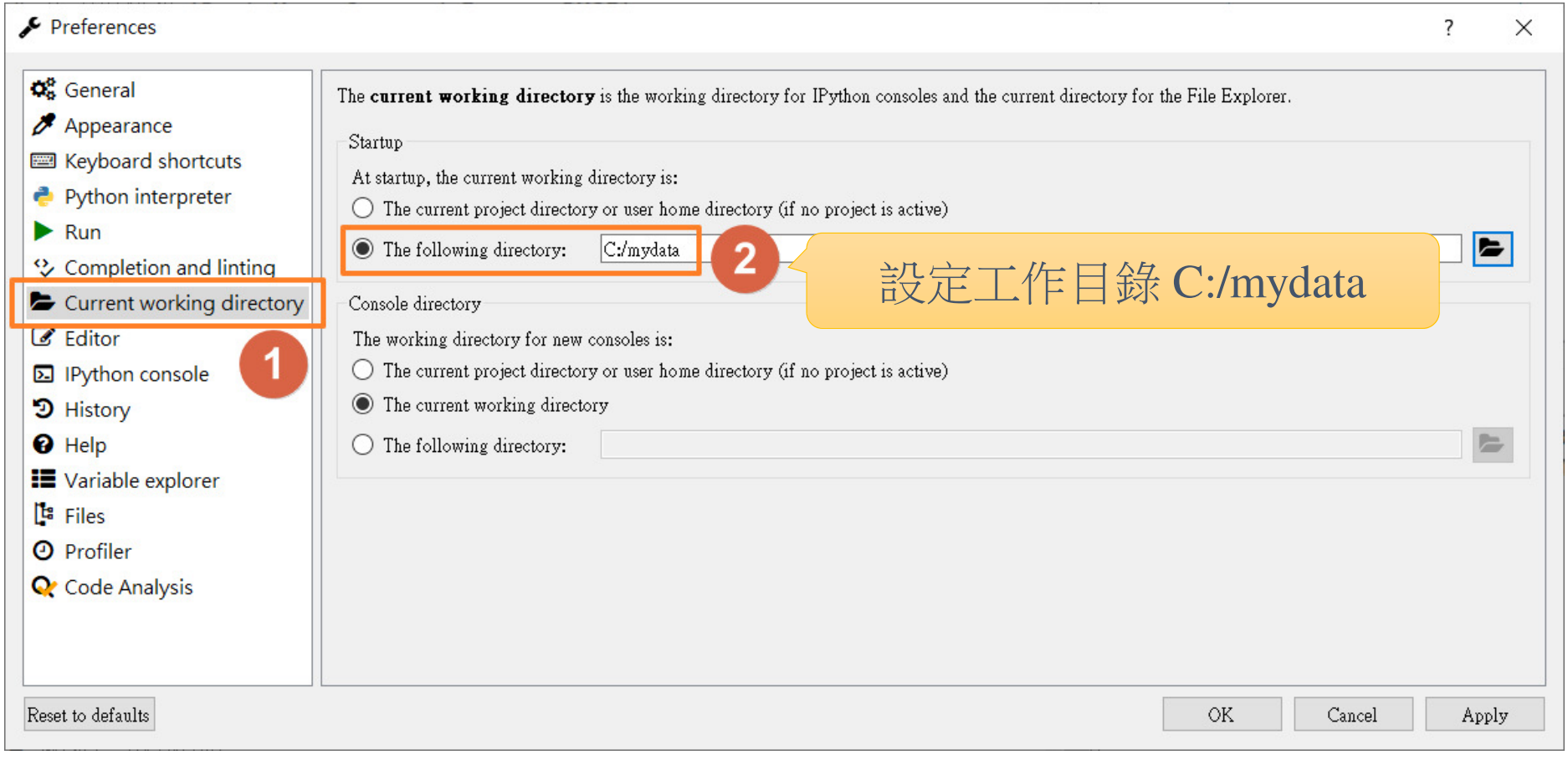
執行選取程式碼	**Ctrl + Enter (原為 F9)**
註解切換	**Ctrl + 1**
尋找	Ctrl + F
取代	Ctrl + R
切換至編輯視窗	Ctrl + Shift + E
切換至 Ipython Console	Ctrl + Shift + I
檔案切換	Ctrl + P
重新啟動 Ipython Console	**Ctrl + .**
檢視放大	Ctrl + “+”
檢視縮小	Ctrl + “-”
重新啟動 Spyder	Alt + Shift + R

Ipython Console 視窗

- 清空: Ctrl + L
- 清空: %clear

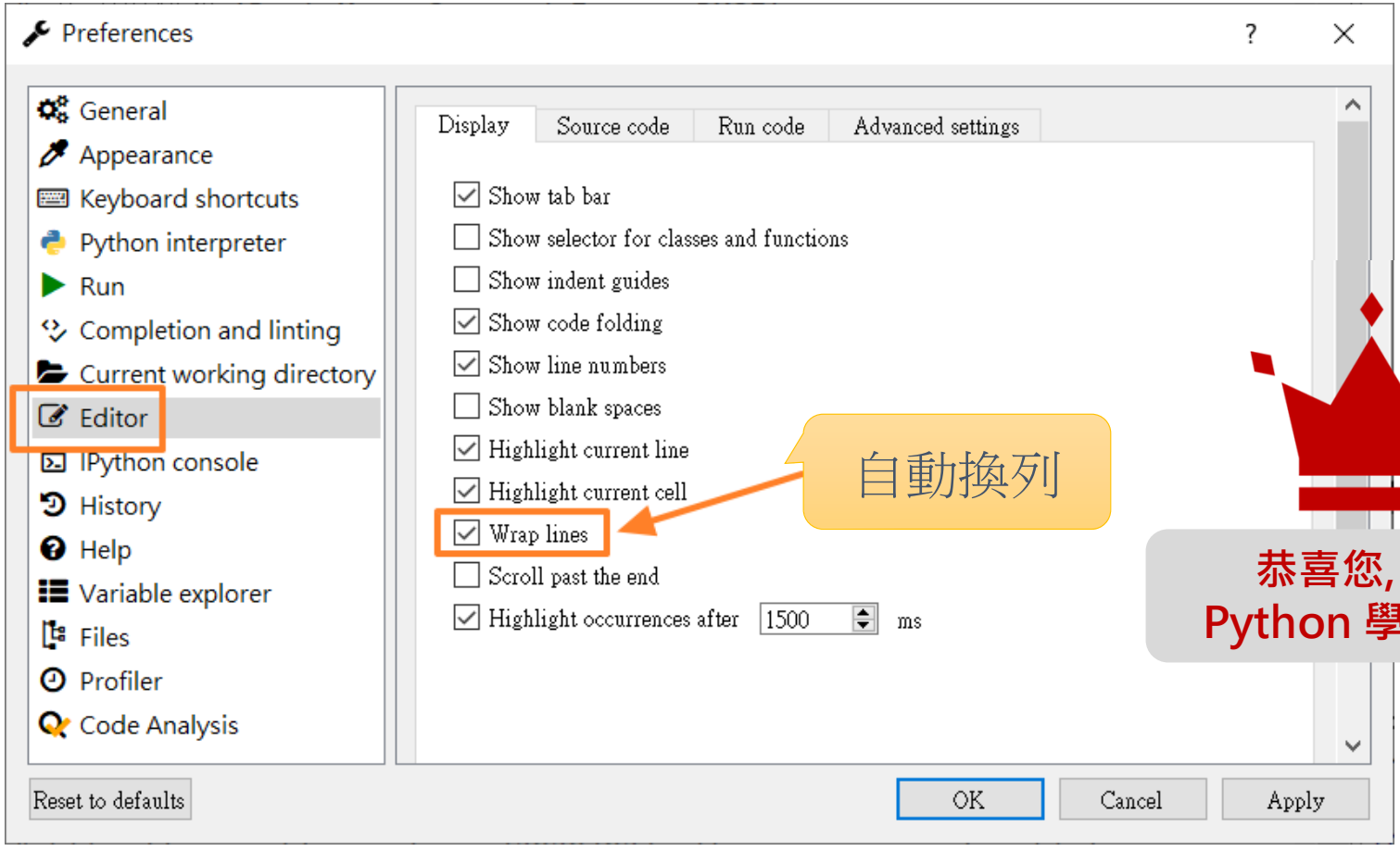


Preferences – Current working directory



實作練習

Preferences – Editor – Wrap lines



恭喜您, 開啟人生
Python 學習之旅 ^_^

Python 執行-命令提示列

- 建立 C:\mydata\helloworld.py

C:\mydata\helloworld.py

helloworld.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jun 22 23:06:49 2021
4
5  @author: rwepa
6  """
7
8  print("大數據Python應用")
```

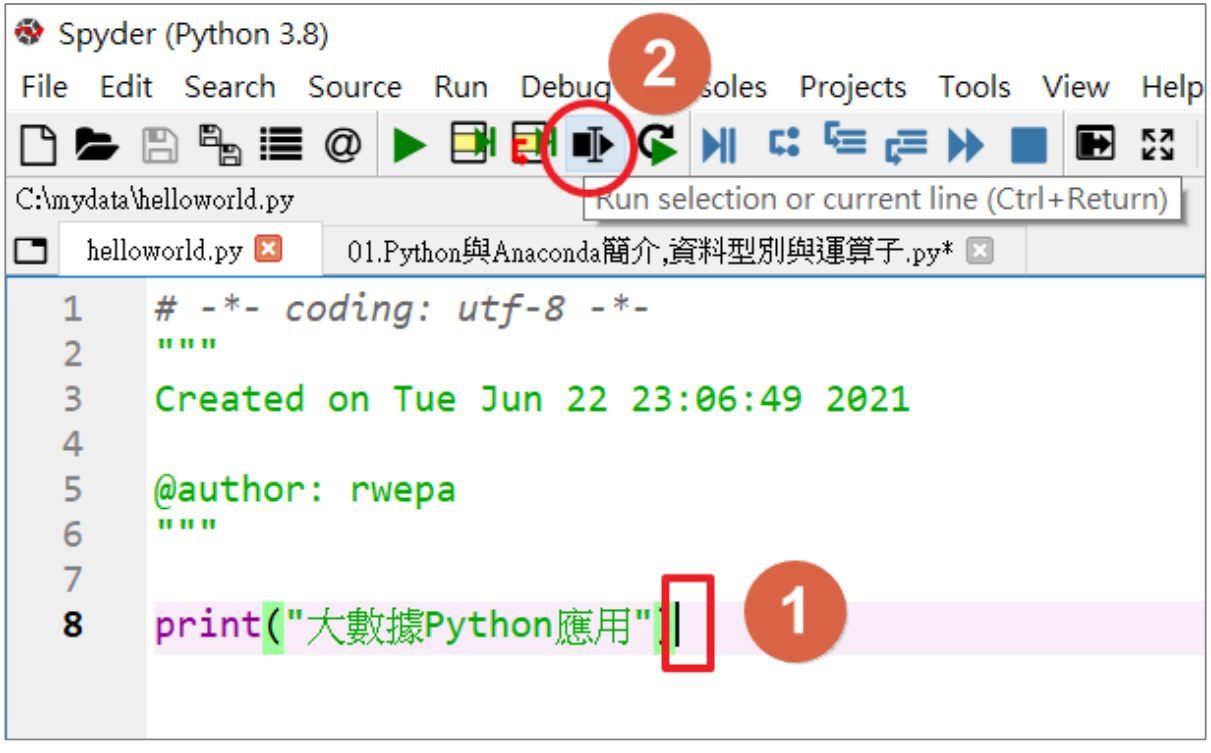
- python helloworld.py

命令提示字元

```
C:\mydata>python --version
Python 3.8.8

C:\mydata>dir
 磁碟區 C 中的磁碟是 WIN10
 磁碟區序號:  E428-7C96

 C:\mydata 的目錄

2021/06/22  下午 11:10    <DIR>          .
2021/06/22  下午 11:10    <DIR>          ..
2021/06/22  下午 11:08               122 helloworld.py
               1 個檔案              122 位元組
               2 個目錄  62,354,751,488 位元組可用

C:\mydata>python helloworld.py
大數據Python應用

C:\mydata>
```

Python 執行-使用 Spyder

- 選取資料列
- 按 [Run selection or current line]



變數

合法的變數名稱

- 必須以字母或下底線字符開頭
- 不能以數字開頭
- 只能包含字母、數字和下底線（A-z、0-9 和 _ ）
- 區分大小寫
  - 例: customer、Customer 和 CUSTOMER 是不同的變數
- 雙下底線開頭並結尾的名稱已經由Python保留, 例: __init__

合法變數 vs. 不合法變數

```
# 合法變數
大數據 = 1 # 中文亦可,建議不要使用
CustomerSaleReport = 1
_CustomerSaleReport = 1
Customer_Sale_Report = 1
customer_sale_report = 1

# 不合法變數
$CustomerSaleReport = 1 # SyntaxError: invalid syntax
2020_sale = 100 # SyntaxError: invalid decimal literal
break = 123 # SyntaxError: invalid syntax
```

- 命名為中文, OK?
- 命名為Python保留字, OK?

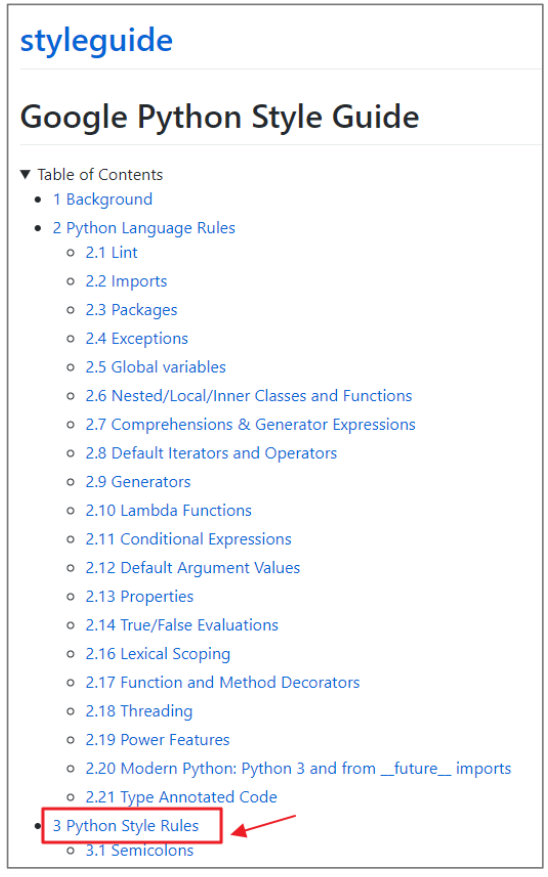
```
# 內建保留字
dir(__builtins__)
len(dir(__builtins__)) # 159
```

指派多個變數

```
In [1]:
   ...: x, y, z = "台北", "台中", "高雄"
   ...: print(x,y,z)
台北 台中 高雄

In [2]: type(x) # str
Out[2]: str

In [3]: address = ["台北", "台中", "高雄"]
   ...: x, y, z = address

In [4]: print(x)
台北

In [5]: print(y)
台中

In [6]: print(z)
高雄
```

Python Style Rules

- https://google.github.io/styleguide/pyguide.html



styleguide

Google Python Style Guide

▼ Table of Contents

Python Style Rules (續)

3 Python Style Rules

3.1 Semicolons

Do not terminate your lines with semicolons, and do not use semicolons to put two statements on the same line.

3.2 Line length

Maximum line length is *80 characters*.

Explicit exceptions to the 80 character limit:

Python 註解

- 使用一個 #　　→ 用於1行註解
- 使用二個 """　　→ 用於超過1行註解或函數之說明文件

```
"""
file     : 01.Python與Anaconda簡介,資料型別與運算子.py
author   : Ming-Chang Lee
email    : alan9956@gmail.com
RWEPA    : http://rwepa.blogspot.tw/
GitHub   : https://github.com/rwepa
Encoding : UTF-8
"""
```

內縮4個空白鍵之語法

注意：with 結尾加冒號

```
with open('sampleinput1.txt', 'r') as f:
    doc = [x.strip() for x in f.readlines()] # 使用 strip 刪除換行符號
    docname = []
    corpus = []
    for index in range(len(doc)):
        if index == 0:
```

內縮4個空白鍵

5.資料型態與四大資料物件

資料型態

資料型別

- **數值型別**
  - 整數　int
  - 長整數　long
  - 浮點數　float
  - 複數　complex
- **布林值 bool**
  - True
  - False
- **空值 None**　→ **類似 NULL**
- **字串 (String)** → 參考 字串處理

參考: https://docs.python.org/3/library/stdtypes.html

資料型別 – 範例

```
# 整數 int
x1 = 1
type(x1)

# 浮點數 float
x2 = 1.234
type(x2)

# 複數  complex
x3 = 1+2j
type(x3)

# 布林值 (Boolean)
x4 = True
type(x4)
x4 + 10
```

- 轉換為整數　　int
- 轉換為浮點數　　float
- 轉換為複數　　complex

None值

```
In [1]: import numpy as np
   ...: None == False
Out[1]: False

In [2]: None == 0
Out[2]: False

In [3]: False == 0
Out[3]: True

In [4]: True == 1
Out[4]: True

In [5]: None == np.nan
Out[5]: False

In [6]: None == None
Out[6]: True
```

整數亂數

```
In [1]: import random
   ...: random.seed(168)
   ...: myrandom = random.randrange(1, 100)
   ...: myrandom
Out[1]: 96

In [2]: random.seed(168)
   ...: myrandom = random.randrange(1, 100)
   ...: myrandom
Out[2]: 96
```

亂數種子 random.seed()

基本運算

運算子

運算子	功能	
**	次方	
*	乘法	
/	除法	
//	整除	
%	餘數	
+	加法	
-	減法	
|	或運算子	OR
^	互斥運算子	XOR
&	且運算子	AND
<<	左移運算子	
>>	右移運算子	

- Excel: =2^3
- R: 2^3
- Python: 2**3
- SQL: SELECT POWER(2, 3)

運算子 – 範例

```
In [1]:
   ...: 3 + 5
Out[1]: 8

In [2]: 3 + (5 * 4)
Out[2]: 23

In [3]: 3 ** 2
Out[3]: 9

In [4]: "Hello" + "World"
Out[4]: 'HelloWorld'

In [5]: 1 + 1.234
Out[5]: 2.234
```

```
In [6]: 7 / 2
Out[6]: 3.5

In [7]: 7 // 2
Out[7]: 3

In [8]: 7 % 2
Out[8]: 1

In [9]: 2 ** 10
Out[9]: 1024

In [10]: 1.234e3 - 1000
Out[10]: 234.0

In [11]: x5 = 1 == 2

In [12]: x5
Out[12]: False

In [13]: x5 + 10
Out[13]: 10
```

位移運算子

```
# 位移運算子: << 向左位移
# 位移運算子: >> 向右位移
a = 4 << 3 # 0100 --> 0100000, 32 16 8 4 2 1
print(a)

b = a * 4.5
print(b)

c = (a+b)/2.5
```

指派運算子

指派運算子	範例	結果
=	x = 9	x = 9
+=	x += 2	x = x + 2
-=	x -= 3	x = x - 3
*=	x *= 4	x = x * 4
/=	x /= 5	x = x / 5
%=	x %= 6	x = x % 6
//=	x //= 7	x = x // 7
**=	x **= 8	x = x ** 8
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```
In [1]:
   ...:
   ...: x = 9
   ...: x+=2
   ...: print(x)
11
```

資料物件

- 容器型別 (Container type)
  - tuple → ( ) 表示, 資料不可修改 (序列/元組)
  - list → [ ] 表示, 資料可以修改 (串列/清單)
  - set → { } 表示, 執行集合運算 (集合)
  - dict → {'key1': value1, ... } 表示, 可藉由key查value (字典)

Tuple

Tuple

- 建立序列

```
f = (2,3,4,5)              # A tuple of integers
g = ()                     # An empty tuple
h = (2, [3,4], (10,11,12)) # A tuple containing  mixed objects
```

- Tuples操作

```
x = f[1]                   # Element access. x = 3
y = f[1:3]                 # Slices (切片) y = (3,4)
z = h[1][1]                # Nesting. z = 4
```

取出 y的第1個索引到 4-1=3

- 特色
  - 與list類似，最大的不同tuple是一種唯讀且不可變更的資料結構
  - 不可取代tuple中的任意一個元素，因為它是唯讀不可變更的
  - Tuple 是具有 ordered 特性
  - Python 的**索引(指標)從0開始**

基本型態 – Tuple

```
In [1]: xy = (2, 3)
   ...: xy
Out[1]: (2, 3)

In [2]: personal = ('Hannah',14,5*12+6)
   ...: personal
Out[2]: ('Hannah', 14, 66)

In [3]: singleton = ("hello",)
   ...: singleton
Out[3]: ('hello',)

In [4]: type(singleton) # tuple
Out[4]: tuple

In [5]: singleton1 = ("hello")
   ...: singleton1
Out[5]: 'hello'

In [6]: type(singleton1) # str
Out[6]: str
```

- 有加上「,」?
- 不加上「,」之資料型別為何?

基本型態 – Tuple (續)

```
In [1]: f = (2,3,4,5)

In [2]: f[0]
Out[2]: 2

In [3]: f[-1]
Out[3]: 5

In [4]: f[-2]
Out[4]: 4

In [5]: f[len(f)-1]
Out[5]: 5

In [6]: t=((1,2), (2,"Hi"), (3,"RWEPA"), 2+3j, 6E23)

In [7]: t[2]
Out[7]: (3, 'RWEPA')

In [8]: t[:3]
Out[8]: ((1, 2), (2, 'Hi'), (3, 'RWEPA'))

In [9]: t[3:]
Out[9]: ((2+3j), 6e+23)

In [10]: t[-1]
Out[10]: 6e+23

In [11]: t[-3:]
Out[11]: ((3, 'RWEPA'), (2+3j), 6e+23)
```

索引 [-1] 表示倒數第1個元素

```
# tuple 長度
len(t) # 5
```

Tuple 建構子

type(employeeGender) # tuple

```
# tuple 建構子, 使用 tuple(( ... )) 或 tuple([ ... ])
employeeGender = tuple(("男", "女", "女"))
employeeGender

# tuple unpacking - 將元素指派至變數
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

type(green) # str

Tuple unpacking - 使用萬用字元*

開箱

```
In [1]:
   ...: fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
   ...: (green, yellow, *red) = fruits
   ...: print(green)
   ...: print(yellow)
   ...: print(red)
apple
banana
['cherry', 'strawberry', 'raspberry']
```

Tuple - loop 處理

```
# tuple - loop 處理
fruits = ("apple", "banana", "cherry")

# 方法1. tuple - 取出元素, 使用for
for x in fruits:
  print(x)

# 方法2. tuple - 取出元素, 使用while
i = 0
while i < len(fruits):
  print(fruits[i])
  i = i + 1

# 方法3. tuple - 取出元素, 使用指標 range, len
for i in range(len(fruits)):
  print(fruits[i])
```

長度 len

- 三個方法結果皆相同,何者較快?

```
In [3]:
   ...: for i in range(len(fruits)):
   ...:   print(fruits[i])
apple
banana
cherry
```

Tuple - join 結合, 重複

```
In [1]: # tuple - join 結合

In [2]: tuple1 = ("台北", "台中", "高雄")
   ...: tuple2 = ("男", "女", "女")
   ...: tuple3 = tuple1 + tuple2
   ...: print(tuple3)
('台北', '台中', '高雄', '男', '女', '女')

In [3]: # tuple - 重複

In [4]: tuple1*3
Out[4]: ('台北', '台中', '高雄', '台北', '台中', '高雄', '台北', '台中', '高雄')

In [5]: 3*tuple1
Out[5]: ('台北', '台中', '高雄', '台北', '台中', '高雄', '台北', '台中', '高雄')
```

使用 + 號

使用 * 號

Tuple - count 次數統計

```
In [6]: # count 次數統計

In [7]: tuple = ("男", "女", "女", "男", "女")

In [8]: tuple.count("男") # 2
Out[8]: 2

In [9]: tuple.count("女") # 3
Out[9]: 3
```

TRY:
tuple.count("A")

序列 – 方法

方法	功能
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

List

基本型態 – 串列(List)

- 建立串列

```
In [1]: a = [2, 3, 4]                  # 整數串列
   ...: b = [2, 7, 3.5, "Hello"] # 混合資料串列
   ...: c = []                         # 空串列
   ...: d = [2, [a, b]]                # 巢狀串列

In [2]: a
Out[2]: [2, 3, 4]

In [3]: b
Out[3]: [2, 7, 3.5, 'Hello']

In [4]: c
Out[4]: []

In [5]: d
Out[5]: [2, [[2, 3, 4], [2, 7, 3.5, 'Hello']]]
```

1 2

串列操作

```
In [7]: a
Out[7]: [2, 3, 4]

In [8]: a[1]          # 取得第2個元素
Out[8]: 3

In [9]: a[-1]         # 取得最後一個元素
Out[9]: 4

In [10]: b[1:3]           # 串列篩選
Out[10]: [7, 3.5]

In [11]: d[1][0][2] # 巢狀串列操作
Out[11]: 4

In [12]: b[0]         # 2
Out[12]: 2

In [13]: b[0] = 42    # 修改元素值

In [14]: b[0]         # 42
Out[14]: 42
```

```
a
[2, 3, 4]

b
[2, 7, 3.5, 'Hello']

d
[2, [[2, 3, 4], [2, 7, 3.5, 'Hello']]]
```

b[1:3] 相當於 b[1], b[2]

串列 slice format

```
In [1]: t=[1, 2, (3,"Hi"), [4,"RWEPA"], 2+3j, 6E7]

In [2]: t
Out[2]: [1, 2, (3, 'Hi'), [4, 'RWEPA'], (2+3j), 60000000.0]
         0  1       2          3          4        5
In [3]: t[2]
Out[3]: (3, 'Hi')

In [4]: t[:3]
Out[4]: [1, 2, (3, 'Hi')]

In [5]: t[3:]
Out[5]: [[4, 'RWEPA'], (2+3j), 60000000.0]

In [6]: t[-1]
Out[6]: 60000000.0

In [7]: t[-3:]
Out[7]: [[4, 'RWEPA'], (2+3j), 60000000.0]

In [8]: # 串列長度

In [9]: len(t)
Out[9]: 6
```

[:3] 開始~指標3-1=2

[3:] 指標3 ~ 最後

串列建構子, 使用 list((...)) 或 list([...])

```
In [1]: mylist1 = list(("男", "女", "女"))

In [2]: mylist1
Out[2]: ['男', '女', '女']

In [3]: mylist2 = list(["男", "女", "女"])

In [4]: mylist2
Out[4]: ['男', '女', '女']

In [5]: mylist1 == mylist2
Out[5]: True
```

串列 unpacking - 將元素指派至變數

```
In [1]:
   ...: fruits = ["apple", "banana", "cherry"]
   ...: green, yellow, red = fruits

In [2]: print(green)
apple

In [3]: print(yellow)
banana

In [4]: print(red)
cherry

In [5]: type(green) # str
Out[5]: str
```

串列 unpacking - 使用萬用字元*

```
In [1]: fruits = ["apple", "banana", "cherry", "strawberry", "raspberry"]

In [2]: green, yellow, *red = fruits

In [3]: print(green)
apple

In [4]: print(yellow)
banana

In [5]: print(red)
['cherry', 'strawberry', 'raspberry']

In [6]: type(green) # str
Out[6]: str
```

串列 - loop 處理

```
# list - loop 處理
mylist = [1, 2, 3, [4, 5], ["A", "B", "C"]]
# 練習 loop 方法

# 方法1. list - 取出元素, 使用for
for x in mylist:
  print(x)

# 方法2. list - 取出元素, 使用while
i = 0
while i < len(mylist):
  print(mylist[i])
  i = i + 1

# 方法3. list - 取出元素, 使用指標 range, len
for i in range(len(mylist)):
  print(mylist[i])

# 方法4. list - 取出元素, 使用串列包含法 (List Comprehension)
[print(x) for x in mylist]
```

```
In [2]: for x in mylist:
   ...:   print(x)
1
2
3
[4, 5]
['A', 'B', 'C']
```

串列包含法應用

```
In [1]: # for 資料篩選-包括字母 a

In [2]: codes = ["Python", "R", "SQL", "Julia", ".NET", "Java", "JavaScript"]
   ...: newlist = []
   ...: for x in codes:
   ...:     if "a" in x:
   ...:         newlist.append(x)
   ...: print(newlist)
['Julia', 'Java', 'JavaScript']
```

串列包含法應用1

- 串列包含法亦可用於序列, 集合, 字典等可反覆運算物件 (可迭代物件, iterable object)

```
In [1]: # 串列包含法應用1

In [2]: # 亦可用於序列, 集合, 字典等可反覆運算物件(可迭代物件, iterable object)

In [3]: codes = ["Python", "R", "SQL", "Julia", ".NET", "Java", "JavaScript"]
   ...: newlist = [x for x in codes if "a" in x]
   ...: print(newlist)
['Julia', 'Java', 'JavaScript']
```

串列包含法應用2

```
In [4]: # 串列包含法應用2

In [5]: newlist = [x.upper() for x in codes]
   ...: print(newlist)
['PYTHON', 'R', 'SQL', 'JULIA', '.NET', 'JAVA', 'JAVASCRIPT']

In [6]: codes.upper() # AttributeError: 'list' object has no attribute 'upper'
Traceback (most recent call last):

  File "<ipython-input-6-19ae796b0f51>", line 1, in <module>
    codes.upper() # AttributeError: 'list' object has no attribute 'upper'

AttributeError: 'list' object has no attribute 'upper'
```

串列包含法應用3

```
In [7]: # 串列包含法應用3

In [8]: newlist = ['RWEPA' for x in codes]
   ...: print(newlist)
['RWEPA', 'RWEPA', 'RWEPA', 'RWEPA', 'RWEPA', 'RWEPA', 'RWEPA']
```

串列-結合, 重複

```
In [14]: # 串列 join 結合

In [15]: e = a + b  # Join two lists

In [16]: e
Out[16]: [2, 3, 4, 2, 7, 3.5, 'Hello']

In [17]: # 串列 repeat 重複

In [18]: f1 = a*3    # repeat lists

In [19]: f1
Out[19]: [2, 3, 4, 2, 3, 4, 2, 3, 4]

In [20]: f2 = 3*a

In [21]: f2
Out[21]: [2, 3, 4, 2, 3, 4, 2, 3, 4]
```

串列-結合: +

串列-重複: *

串列 – 排序 sort

```
In [1]: # 串列排序-預設為遞增排序, 英文字母先大寫, 再小寫

In [2]: codes = ["python", "R", "SQL", "Julia", ".NET", "java", "JavaScript"]
   ...: codes.sort()
   ...: print(codes)
['.NET', 'JavaScript', 'Julia', 'R', 'SQL', 'java', 'python']

In [3]: # 串列排序-先全部小寫, 再排序

In [4]: codes = ["python", "R", "SQL", "Julia", ".NET", "java", "JavaScript"]

In [5]: codes.sort(key = str.lower)

In [6]: print(codes)
['.NET', 'java', 'JavaScript', 'Julia', 'python', 'R', 'SQL']
```

串列-遞減排序

```
In [1]: # 串列排序-遞減排序

In [2]: codes = ["python", "R", "SQL", "Julia", ".NET", "java", "JavaScript"]

In [3]: codes.sort(reverse =True)

In [4]: print(codes)
['python', 'java', 'SQL', 'R', 'Julia', 'JavaScript', '.NET']

In [5]: # 串列反序

In [6]: codes = ["python", "R", "SQL", "Julia", ".NET", "java", "JavaScript"]

In [7]: codes.reverse()

In [8]: print(codes)
['JavaScript', 'java', '.NET', 'Julia', 'SQL', 'R', 'python']
```

串列複製 – 使用等號

```
In [1]: # 串列複製,等號會建立參考物件

In [2]: a = [1, 2, 3]

In [3]: a
Out[3]: [1, 2, 3]

In [4]: b = a

In [5]: b[0] = 999 # 修改b,亦會修改a

In [6]: b
Out[6]: [999, 2, 3]

In [7]: a # a已經更新
Out[7]: [999, 2, 3]
```

串列複製 – 使用 copy

```
In [8]: # 串列複製-使用 copy

In [9]: a = [1, 2, 3]

In [10]: b = a.copy()

In [11]: b
Out[11]: [1, 2, 3]

In [12]: b[0] = 999

In [13]: b
Out[13]: [999, 2, 3]

In [14]: a # a保持不變
Out[14]: [1, 2, 3]
```

串列複製-使用 list

```
In [1]: # 串列複製-使用 list

In [2]: a = [1, 2, 3]

In [3]: c = list(a)

In [4]: c
Out[4]: [1, 2, 3]

In [5]: c[0] = 123

In [6]: c
Out[6]: [123, 2, 3]

In [7]: a # a保持不變
Out[7]: [1, 2, 3]
```

串列附加 append, 延伸 extend

```
In [1]: # 附加元素 append

In [2]: a = [1, 2, 3]

In [3]: a.append(['BigData', 'SQL']) # 新增1個元素

In [4]: a
Out[4]: [1, 2, 3, ['BigData', 'SQL']]

In [5]: # 延伸元素 extend

In [6]: a.extend(['Python', 'R', "Julia"]) # 新增一個串列

In [7]: a
Out[7]: [1, 2, 3, ['BigData', 'SQL'], 'Python', 'R', 'Julia']
```

附加為1個值

延伸為3個值

串列延伸 – tuple, list, set, dict

```
In [8]: # 延伸元素 extend - 加入tuple,list,set,dict

In [9]: a = [1, 2, 3]

In [10]: a.extend(('4', '5', 'RWEPA')) # 延伸一個序列

In [11]: a
Out[11]: [1, 2, 3, '4', '5', 'RWEPA']

In [12]: a.extend({'8', '8', '10'}) # 延伸一個集合

In [13]: a
Out[13]: [1, 2, 3, '4', '5', 'RWEPA', '8', '10']

In [14]: a.extend({'a':'R', 'b':'Python'}) # 延伸一個字典-ONLY KEY, NO VALUE

In [15]: a
Out[15]: [1, 2, 3, '4', '5', 'RWEPA', '8', '10', 'a', 'b']
```

串列 – insert

```
In [1]: # 插入元素

In [2]: a = list(range(5))

In [3]: a
Out[3]: [0, 1, 2, 3, 4]

In [4]: a.insert(2, 999) # 在指標為2的位置,插入新元素
                  1   2

In [5]: a
Out[5]: [0, 1, 999, 2, 3, 4]
```

串列 – remove, pop, del

```
In [6]: # 刪除指定元素

In [7]: a.remove(999)

In [8]: a
Out[8]: [0, 1, 2, 3, 4]

In [9]: # 刪除指定指標元素

In [10]: a.pop(1)
Out[10]: 1

In [11]: a
Out[11]: [0, 2, 3, 4]

In [12]: # 刪除指定指標元素

In [13]: del a[1]

In [14]: a
Out[14]: [0, 3, 4]
```

串列 – pop

```
In [14]: a
Out[14]: [0, 3, 4]

In [15]: # 刪除第一個元素

In [16]: a.pop(0)
Out[16]: 0

In [17]: a
Out[17]: [3, 4]

In [18]: # 刪除最後一個元素

In [19]: a.pop()
Out[19]: 4

In [20]: a
Out[20]: [3]
```

串列 – clear, del

```
# 清空物件元素, 物件仍存在記憶體
a.clear()
a

# 刪除物件, 物件不存在記憶體
del a
print(a) # NameError: name 'a' is not defined
```

串列 - zip 應用

```
In [1]: # zip 應用

In [2]: a = ("x1", "x2", "x3")

In [3]: b = ("y1", "y2", "y3")

In [4]: c = (1, 2, 3)

In [5]: x = zip(a, b, c)

In [6]: x
Out[6]: <zip at 0x19620369740>

In [7]: list(x)
Out[7]: [('x1', 'y1', 1), ('x2', 'y2', 2), ('x3', 'y3', 3)]
```

串列 – 方法

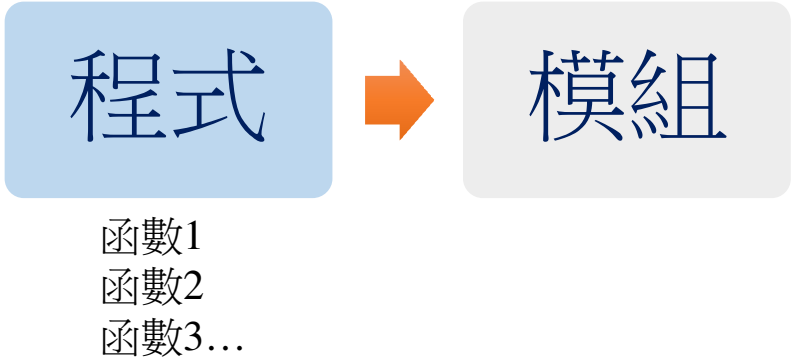
方法	功能
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python demo - set, dict

模組 (Modules)

模組

- Python 可以將一個主程式，分割成多個檔案，此分割的檔案可以形成模組(module)。
- 主檔名表示模組的名稱，即檔案名稱 =「**模組名稱 .py** 」 。
- 模組中包含 Python 函數和語法的檔案，可由其他專案使用。
- 模組中的函數可以被 import 到其他模組中，或是被 import 至Python程式。
- 在模組中，模組的名稱（字串）是全域變數 __name__ 的值。
- 第1次執行時，原始程式會編譯成「.pyc檔案」之位元碼(bytecode)，以增進執行效率。
- 模組下載：
  - pip install moduleName
  - conda install moduleName



使用模組

- 匯入模組的所有函數
  - import 模組名稱
  - import 模組名稱 as 別名
- 匯入特定函數
  - from 模組名稱 import 函數名稱1,...
  - from 模組名稱 import *
- 使用模組內的特定函數
  - 模組名稱.函數( )

```
In [1]: import math

In [2]: math.sqrt(9)
Out[2]: 3.0

In [3]: from math import sqrt

In [4]: sqrt(9)
Out[4]: 3.0
```

切換工作目錄

- os.getcwd()
- os.chdir(“C:/”)

```
# 切換工作目錄
import os
os.getcwd() # 讀取工作目錄
os.chdir("C:/") # 變更工作目錄
os.getcwd()
os.listdir(os.getcwd()) # 顯示檔案清單
```

模組的搜尋路徑

```
In [1]: import sys

In [2]: sys.path
Out[2]:
['C:\\Users\\88697\\anaconda3\\python38.zip',
 'C:\\Users\\88697\\anaconda3\\DLLs',
 'C:\\Users\\88697\\anaconda3\\lib',
 'C:\\Users\\88697\\anaconda3',
 '',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages\\locket-0.2.1-py3.8.egg',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages\\win32',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\88697\\anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\88697\\.ipython']
```

模組預設安裝資料夾

6.pandas 資料處理

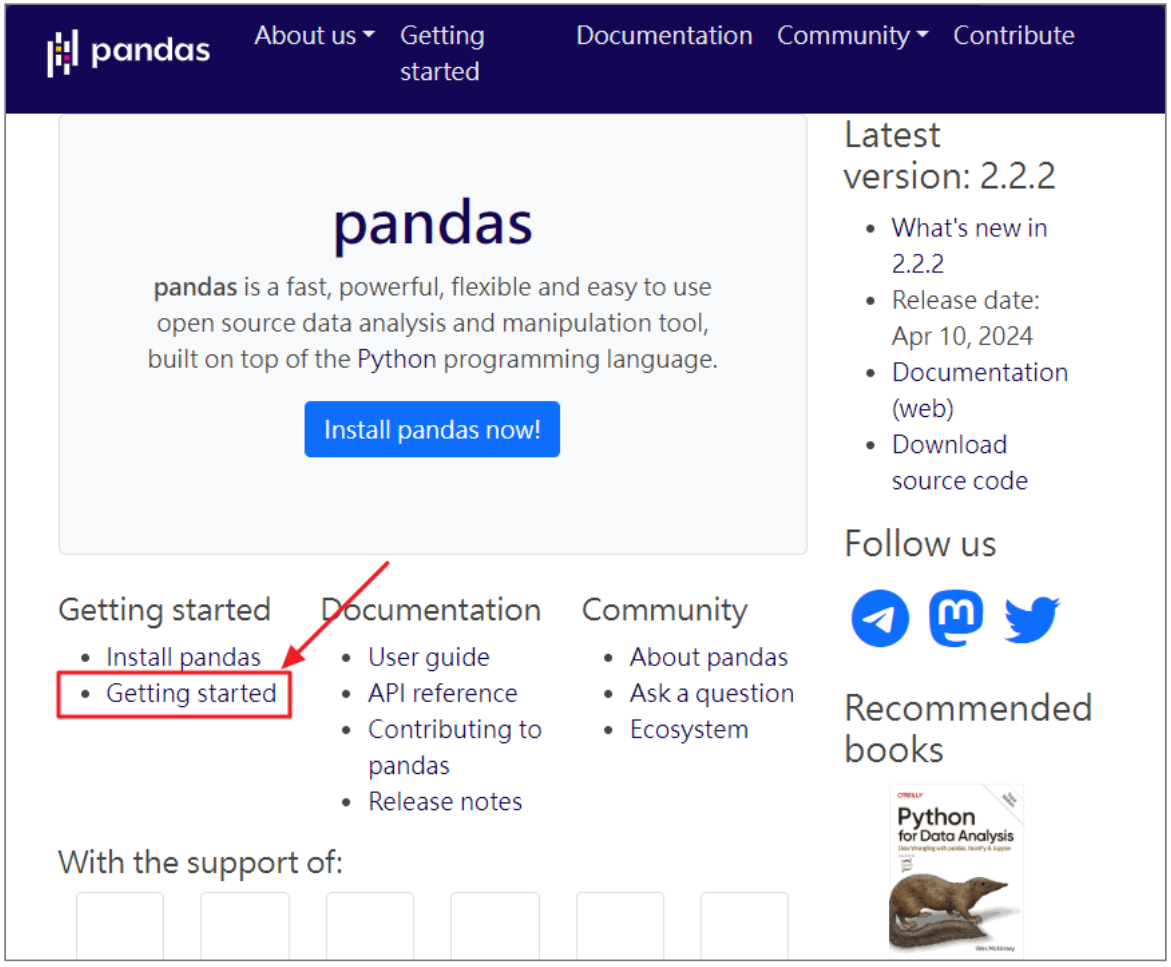
Pandas 資料結構

- Series 物件 → 相當於 Excel ?
- DataFrame 物件

pandas 模組

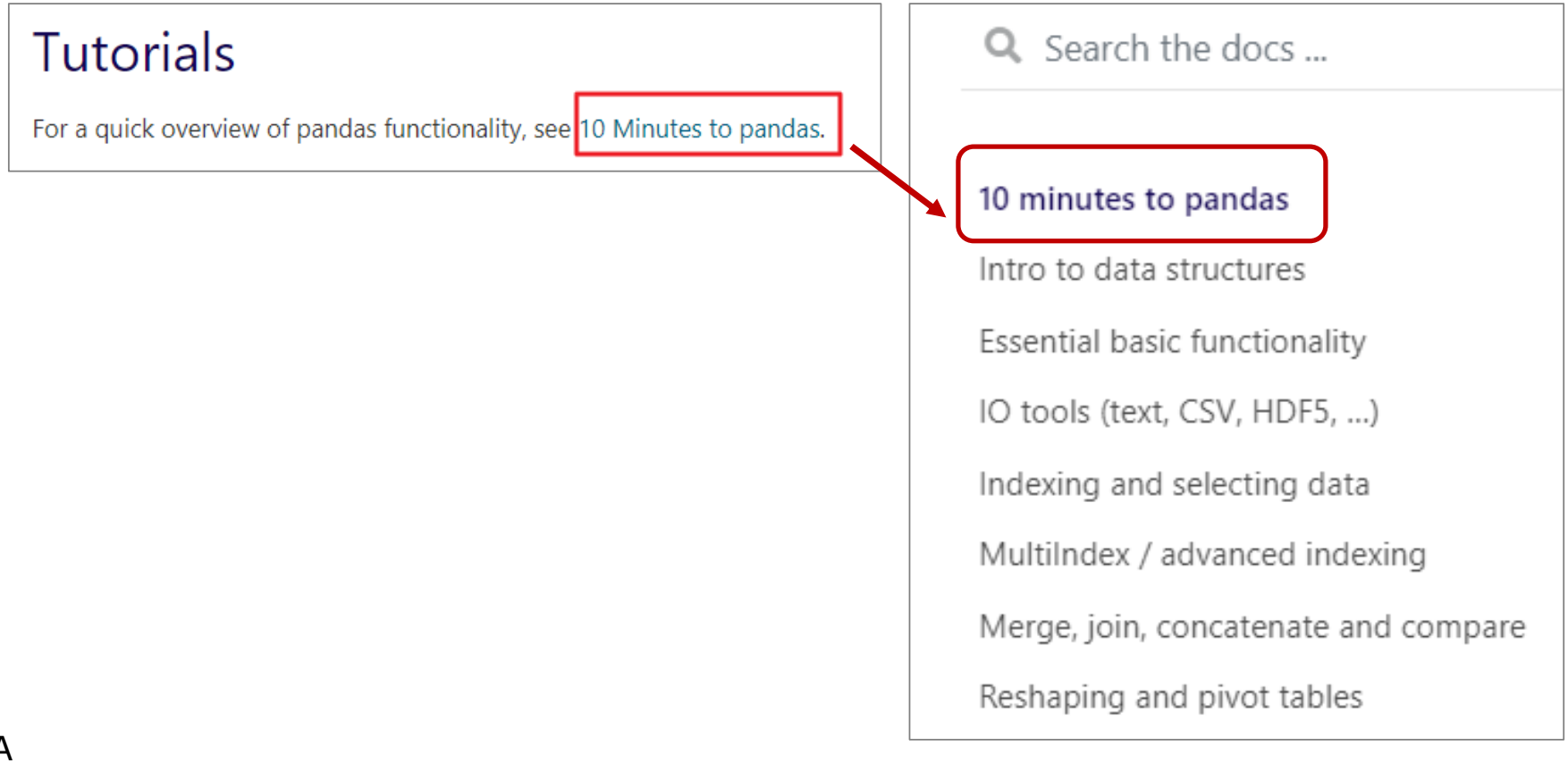
- pandas 取名自 pan(el)-da(ta)-s
- pandas 提供資料讀取，資料整理，統計分析，繪圖等功能。
- pandas是一套使用Python語言開發的Python套件，完整包含NumPy、Scipy和Matplotlab套件的功能.

https://pandas.pydata.org/



10 minutes to pandas

- https://pandas.pydata.org/docs/user_guide/10min.html



pandas 資料結構

- **Series 物件：**
  - 類似一維陣列的物件，擁有索引(index)與值的一維陣列.
  - 可以將Series視為是2個陣列的組合，一個是類似索引的標籤，另一個是實際資料值
  - s = pd.Series(data, index=index)
    data = array-like, Iterable, dict, scalar value
- **DataFrame 物件：**
  - 類似Excel 試算表的表格資料
  - 具有索引的二維陣列
  - 一個可以任易更改結構的表格，每一欄允許儲存不同資料型態的資料.

Series 物件

- import numpy as np # Python Scientific Computing Library
- import pandas as pd # Python Data Analysis Library

(1).Series - 使用 ndarray

```
# s = pd.Series(data, index=index)
# data 包括使用 array, Iterable, dict, scalar value
# 序列包括指標(Index) 與值(Value), 指標採用預設整數型態指標 0,1,2,...

# (1).Series - 使用 ndarray
s = pd.Series(data = np.random.randn(5), index=["a", "b", "c", "d", "e"])
s
# a   -0.492604
# b   -0.073386
# c   -0.063632
# d    0.197128
# e    0.178333
# dtype: float64
type(s) # pandas.core.series.Series
```

結果類似 Excel 的一行資料

(2).Series -使用 Iterable - 序列(tuple)

```
In [2]: # (2).Series -使用 Iterable - 序列(tuple)

In [3]: s1 = pd.Series((1,3,5,np.nan,6,8))
   ...: s1
Out[3]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

(3).Series - 使用 Iterable - 串列(List)

```
In [4]: # (3).Series - 使用 Iterable - 串列(List)

In [5]: s2 = pd.Series([1,3,5,np.nan,6,8])
   ...: s2
Out[5]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

相等(==) vs. 相同(is)

```
In [6]: s1 == s2 # equality 相等, 比較每個元素是否相同, 大部分使用此功能.
Out[6]:
0     True
1     True
2     True
3    False
4     True
5     True
dtype: bool
```

== 運算結果是 **False**

```
In [7]: s1 is s2 # identity 相同, 比較二物件是否指向同一個記憶體
Out[7]: False

In [8]: id(s1)
Out[8]: 2824957766672

In [9]: id(s2) # 與id(s1) 不相等
Out[9]: 2824958104144
```

equality vs. identity

- identity - 使用 id 函數, 查看說明 help(id). 相同程式 id 結果,每次不一定相同.
- https://realpython.com/python-is-identity-vs-equality/

```
In [1]: a = 'Hello world'

In [2]: b = 'Hello world'

In [3]: a == b
Out[3]: True

In [4]: a is b
Out[4]: False

In [5]: id(a)
Out[5]: 2166585399024

In [6]: id(b)
Out[6]: 2166589303664
```

```
In [1]: # 整數 [-5 ~ 256] 會使用相同記憶體位址功能

In [2]: a = 256
   ...: b = 256
   ...: a == b   # True
   ...: a is b   # True
   ...: id(a)
Out[2]: 140734466311952

In [3]: id(b)
Out[3]: 140734466311952

In [4]: a = 1000
   ...: b = 1000
   ...: a == b   # True
   ...: a is b   # False
   ...: id(a)
Out[4]: 2085585143472

In [5]: id(b)
Out[5]: 2085585143312
```

```
In [2]: x1 = np.nan

In [3]: x2 = np.nan

In [4]: id(x1)
Out[4]: 2355752046768

In [5]: id(x2) # 與上面結果相同
Out[5]: 2355752046768

In [6]: x1 == x2 # False
Out[6]: False

In [7]: x1 is x2 # True
Out[7]: True
```

(4).Series - 使用 Iterable - 字典(Dict)

```
In [2]: # (4).Series - 使用 Iterable - 字典(Dict)

In [3]: # 在 pandas 模組之中, NaN 表示為 "not a number"

In [4]: x = {"x1": 1, "x2": 2, "a": np.nan, "b": 3, "c": 4}
   ...: c = pd.Series(x)
   ...: c
Out[4]:
x1    1.0
x2    2.0
a     NaN
b     3.0
c     4.0
dtype: float64
```

(5).Series - 使用 scalar value

```
In [5]: # (5).Series - 使用 scalar value

In [6]: pd.Series(999.0, index=["a", "b", "c", "d", "e"])
Out[6]:
a    999.0
b    999.0
c    999.0
d    999.0
e    999.0
dtype: float64
```

可快速建立預設值

Series 使用 ndarray-like 操作

```
c
# x1     1.0
# x2     2.0
# a      NaN
# b      3.0
# c      4.0
# dtype: float64

c[0]                  # 1.0
c[1]                  # 2.0
c[-1]                 # 4.0
c[:3]
# x1     1.0
# x2     2.0
# a      NaN
# dtype: float64
```

```
In [15]: c[c > c.median()]
Out[15]:
b    3.0
c    4.0
dtype: float64

In [16]: c[[1, 3, 2]]
Out[16]:
x2    2.0
b     3.0
a     NaN
dtype: float64

In [17]: np.exp(c)
Out[17]:
x1      2.718282
x2      7.389056
a            NaN
b      20.085537
c      54.598150
dtype: float64
```

Series.array vs. ExtensionArray

```
In [20]: c.array      # 將 series 轉換為 PandasArray
Out[20]:
<PandasArray>
[1.0, 2.0, nan, 3.0, 4.0]
Length: 5, dtype: float64

In [21]: c1 = c.to_numpy() # 將 series 轉換為 NumPy ndarray

In [22]: c1
Out[22]: array([ 1.,  2., nan,  3.,  4.])

In [23]: c2 = c.to_numpy

In [24]: c2
Out[24]:
<bound method IndexOpsMixin.to_numpy of x1    1.0
x2    2.0
a     NaN
b     3.0
c     4.0
dtype: float64>

In [25]: c1 == c2
Out[25]: array([False, False, False, False, False])

In [26]: c1 is c2
Out[26]: False
```

- Series.array 是 pandasExtensionArray.
- ExtensionArray 是包括一個或多個 numpy.ndarray 的 thin wrapper類別

Series 使用 dict-like 操作

```
In [28]: c
Out[28]:
x1    1.0
x2    2.0
a     NaN
b     3.0
c     4.0
dtype: float64
```

```
In [29]: c['x1']
Out[29]: 1.0

In [30]: c['a'] = np.pi

In [31]: 'x1' in c
Out[31]: True

In [32]: c.get("a")
Out[32]: 3.141592653589793

In [33]: c.get("e") # None
```

pandas 資料框(DataFrame)物件

方法1.建立指標與值,再合併為資料框

```
In [2]: # 方法1.建立指標與值,再合併為資料框

In [3]: # 步驟1-建立 DatetimeIndex 物件

In [4]: dates = pd.date_range('20210801', periods=6) # 日期指標

In [5]: dates
Out[5]:
DatetimeIndex(['2021-08-01', '2021-08-02', '2021-08-03', '2021-08-04',
               '2021-08-05', '2021-08-06'],
              dtype='datetime64[ns]', freq='D')

In [6]: type(dates)
Out[6]: pandas.core.indexes.datetimes.DatetimeIndex
```

pd.DataFrame( )

```
In [7]: # 步驟2-建立 DataFrame

In [8]: # 欄位名稱: A, B, C, D

In [9]: df1 = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
   ...: df1
Out[9]:
                   A         B         C         D
2021-08-01  0.338935  1.610452 -1.534346 -1.008859
2021-08-02 -0.378642 -0.832767 -1.289297  1.327141
2021-08-03  0.706616 -1.376127  0.695725  0.446339
2021-08-04  0.267140  1.105310  0.072173  0.942306
2021-08-05  0.376187  0.220437  0.511613  0.828549
2021-08-06  0.414686  1.277009  0.614274 -0.324311

In [10]: type(df1)
Out[10]: pandas.core.frame.DataFrame
```

1 2 3

方法2.使用字典建立資料框

```
In [11]: df2 = pd.DataFrame({ 'A' : 1.,
    ...:      'B' : pd.Timestamp('20210801'),
    ...:      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
    ...:      'D' : np.array([3] * 4,dtype='int32'),
    ...:      'E' : pd.Categorical(["test","train","test","train"]),
    ...:      'F' : 'foo' })
    ...: df2
Out[11]:
     A          B    C  D      E    F
0  1.0 2021-08-01  1.0  3   test  foo
1  1.0 2021-08-01  1.0  3  train  foo
2  1.0 2021-08-01  1.0  3   test  foo
3  1.0 2021-08-01  1.0  3  train  foo

In [12]:
    ...: df2.dtypes
Out[12]:
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

方法3.使用 list of dicts 建立資料框

```
In [13]: # 預設指標

In [14]: mydata = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
    ...: df3 = pd.DataFrame(mydata)
    ...: df3
Out[14]:
   a   b     c
0  1   2   NaN
1  5  10  20.0

In [15]: # 客製化指標

In [16]: df4 = pd.DataFrame(mydata, index=["first", "second"])
    ...: df4
Out[16]:
        a   b     c
first   1   2   NaN
second  5  10  20.0
```

方法4.使用 dict of tuples 建立資料框

- 使用 tuples dictionary, 可建立 MultiIndexed dataframe (階層式指標資料框)

```
In [17]: df5 = pd.DataFrame(
    ...:     {
    ...:         ("a", "b"): {("A", "B"): 1, ("A", "C"): 2},
    ...:         ("a", "a"): {("A", "C"): 3, ("A", "B"): 4},
    ...:         ("a", "c"): {("A", "B"): 5, ("A", "C"): 6},
    ...:         ("b", "a"): {("A", "C"): 7, ("A", "B"): 8},
    ...:         ("b", "b"): {("A", "D"): 10, ("A", "B"): 11},
    ...:     }
    ...: )
    ...: df5
Out[17]:
        a              b
        b    a    c    a     b
A B   1.0  4.0  5.0  8.0  11.0
  C   2.0  3.0  6.0  7.0   NaN
  D   NaN  NaN  NaN  NaN  10.0
```

方法5.使用 list of dataclasses 建立資料框

- list of dataclasses 類似於 list of dictionaries

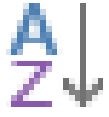
```
In [1]: import numpy as np  # Python Scientific Computing Library

In [2]: import pandas as pd # Python Data Analysis Library

In [3]: from dataclasses import make_dataclass

In [4]: Mydata = make_dataclass("Stations", [("x", int), ("y", int)])

In [5]: Mydata
Out[5]: types.Stations

In [6]: df6 = pd.DataFrame([Mydata(0, 0), Mydata(0, 3), Mydata(2, 3), Mydata(1, 2)])

In [7]: df6
Out[7]:
   x  y
0  0  0
1  0  3
2  2  3
3  1  2
```

pandas 1.1.0 新增功能
資料類別 (Data Classes)

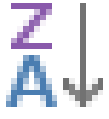
1

2

排序



從 A 到 Z 排序(S)　遞增(預設值)



從 Z 到 A 排序(O)　遞減

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html

(1).排序 sort_index

- 當對資料 "列" 進行排序時，axis必須設置為0.
- df.sort(["A"]) 新版不支援 sort, 改用 sort_values 或 sort_index
- ascending =FALSE, 即遞增是FALSE, 表示遞減是TRUE, 結果為D,C,B,A

```
In [10]: df1.sort_index(axis=1, ascending=False)
Out[10]:
                   D         C         B         A
2021-08-01 -0.928378  1.885510  1.521155  0.695556
2021-08-02 -1.394514  1.209076 -0.733466  0.740674
2021-08-03 -0.237016 -0.642413  0.538803  0.903706
2021-08-04  0.649434 -0.741611  0.932843 -0.146590
2021-08-05 -0.899328 -0.933120 -0.800698 -0.347963
2021-08-06 -1.790927  0.159388  0.827578  0.080708
```

- 0表示橫列指標
- 1表示直行指標

(2).排序 sort_values

- 依照 B 欄大小, 由小至大排序 (預設值是遞增)

```
In [12]: df1.sort_values(by='B')
Out[12]:
                   A         B         C         D
2021-08-05 -0.347963 -0.800698 -0.933120 -0.899328
2021-08-02  0.740674 -0.733466  1.209076 -1.394514
2021-08-03  0.903706  0.538803 -0.642413 -0.237016
2021-08-06  0.080708  0.827578  0.159388 -1.790927
2021-08-04 -0.146590  0.932843 -0.741611  0.649434
2021-08-01  0.695556  1.521155  1.885510 -0.928378
```

(2).排序 sort_values (續)

- 依照 B 欄大小, 改為由大至小排序 (遞減)

```
In [13]: df1.sort_values(by='B', ascending = False)
Out[13]:
                   A         B         C         D
2021-08-01  0.695556  1.521155  1.885510 -0.928378
2021-08-04 -0.146590  0.932843 -0.741611  0.649434
2021-08-06  0.080708  0.827578  0.159388 -1.790927
2021-08-03  0.903706  0.538803 -0.642413 -0.237016
2021-08-02  0.740674 -0.733466  1.209076 -1.394514
2021-08-05 -0.347963 -0.800698 -0.933120 -0.899328
```

(2).排序 sort_values - nan

- 依照 B 欄大小, 將 nan 排在最前面

```
In [25]: df1.sort_values(by='A')
Out[25]:
                   A         B         C         D
2021-08-05 -0.347963 -0.800698 -0.933120 -0.899328
2021-08-04 -0.146590  0.932843 -0.741611  0.649434
2021-08-06  0.080708  0.827578  0.159388 -1.790927
2021-08-01  0.695556  1.521155  1.885510 -0.928378
2021-08-02  0.740674 -0.733466  1.209076 -1.394514
2021-08-03       NaN  0.538803 -0.642413 -0.237016

In [26]: df1.sort_values(by='A', na_position = 'first')
Out[26]:
                   A         B         C         D
2021-08-03       NaN  0.538803 -0.642413 -0.237016
2021-08-05 -0.347963 -0.800698 -0.933120 -0.899328
2021-08-04 -0.146590  0.932843 -0.741611  0.649434
2021-08-06  0.080708  0.827578  0.159388 -1.790927
2021-08-01  0.695556  1.521155  1.885510 -0.928378
2021-08-02  0.740674 -0.733466  1.209076 -1.394514
```

實作練習

pandas 排序

- 使用 mydf 進行A欄位遞增, B欄位遞減排序
- 排序後, 須一併更改 mydf 內容

	A	B	C
0	1	10	aa
1	2	24	bb
2	2	26	cc
3	4	9	dd
4	2	29	aa

	A	B	C
0	1	10	aa
4	2	29	aa
2	2	26	cc
1	2	24	bb
3	4	9	dd

資料列,行選取

選取行

```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: np.random.seed(123)
   ...: dates = pd.date_range('20210801', periods=6) # 日期指標
   ...: df1 = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
   ...: df1
Out[1]:
                   A         B         C         D
2021-08-01 -1.085631  0.997345  0.282978 -1.506295
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351
2021-08-05  2.205930  2.186786  1.004054  0.386186
2021-08-06  0.737369  1.490732 -0.935834  1.175829

In [2]:
   ...: df1['A']
   ...: df1.A
Out[2]:
2021-08-01   -1.085631
2021-08-02   -0.578600
2021-08-03    1.265936
2021-08-04    1.491390
2021-08-05    2.205930
2021-08-06    0.737369
Freq: D, Name: A, dtype: float64
```

二種方法-

- df1['A'], df1[['A', 'B']]
- df1.A

選取列

- 選取列, df[1:4]選取第1至第3列(4-1=3), 此功能與 R 不同.
- R: df[1:4] 表示選取第1至第4行

```
In [3]:
   ...: df1
   ...: df1[1:4]
Out[3]:
                   A         B         C         D
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351
```

```
           A         B         C         D
0 2021-08-01 -1.085631  0.997345  0.282978 -1.506295
1 2021-08-02 -0.578600  1.651437 -2.426679 -0.428913
2 2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
3 2021-08-04  1.491390 -0.638902 -0.443982 -0.434351
4 2021-08-05  2.205930  2.186786  1.004054  0.386186
5 2021-08-06  0.737369  1.490732 -0.935834  1.175829
```

使用 loc, iloc

```
In [4]: # 使用 loc

In [5]: df1.loc[:, ['A','B']]
Out[5]:
                   A         B
2021-08-01 -1.085631  0.997345
2021-08-02 -0.578600  1.651437
2021-08-03  1.265936 -0.866740
2021-08-04  1.491390 -0.638902
2021-08-05  2.205930  2.186786
2021-08-06  0.737369  1.490732

In [6]: # 使用 iloc

In [7]: df1.iloc[2] # 指標為第2列
Out[7]:
A    1.265936
B   -0.866740
C   -0.678886
D   -0.094709
Name: 2021-08-03 00:00:00, dtype: float64
```

- loc[列, 行]
- 如果列的位置是 : 表示選取所有列

iloc[,] iloc[, :]

```
In [8]: df1.iloc[2:4,]
Out[8]:
                   A         B         C         D
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351

In [9]: df1.iloc[2:4, :]
Out[9]:
                   A         B         C         D
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351
```

iloc[:,] 逗號旁加上冒號:

```
In [11]: df1.iloc[:, 2]    # OK
Out[11]:
2021-08-01    0.282978
2021-08-02   -2.426679
2021-08-03   -0.678886
2021-08-04   -0.443982
2021-08-05    1.004054
2021-08-06   -0.935834
Freq: D, Name: C, dtype: float64

In [12]: df1.iloc[:, 2:4] # OK
Out[12]:
                   C         D
2021-08-01  0.282978 -1.506295
2021-08-02 -2.426679 -0.428913
2021-08-03 -0.678886 -0.094709
2021-08-04 -0.443982 -0.434351
2021-08-05  1.004054  0.386186
2021-08-06 -0.935834  1.175829
```

```
                   A         B         C         D
0 2021-08-01 -1.085631  0.997345  0.282978 -1.506295
1 2021-08-02 -0.578600  1.651437 -2.426679 -0.428913
2 2021-08-03  1.265936 -0.866740 -0.678886 -0.094709
3 2021-08-04  1.491390 -0.638902 -0.443982 -0.434351
4 2021-08-05  2.205930  2.186786  1.004054  0.386186
5 2021-08-06  0.737369  1.490732 -0.935834  1.175829
```

少了冒號

```
In [10]: df1.iloc[, 2]    # ERROR
  File "<ipython-input-10-5ff903a3b603>", line 1
    df1.iloc[, 2]    # ERROR
             ^
SyntaxError: invalid syntax
```

Boolean Indexing 邏輯值(條件式)資料選取

```
In [2]: df1.loc[dates[2]]
Out[2]:
A    1.265936
B   -0.866740
C   -0.678886
D   -0.094709
Name: 2021-08-03 00:00:00, dtype: float64

In [3]: df1.loc['20210803']
Out[3]:
A    1.265936
B   -0.866740
C   -0.678886
D   -0.094709
Name: 2021-08-03 00:00:00, dtype: float64
```

邏輯值(條件式)資料選取 (續)

```
In [4]: df1.loc['20210803', ['A', 'B']]
Out[4]:
A    1.265936
B   -0.866740
Name: 2021-08-03 00:00:00, dtype: float64

In [5]: df1.loc['20210802':'20210804', ['A', 'B']]
Out[5]:
                   A         B
2021-08-02 -0.578600  1.651437
2021-08-03  1.265936 -0.866740
2021-08-04  1.491390 -0.638902

In [6]: df1.iloc[[1,2,4],[0,2]] # 選取不連續範圍
Out[6]:
                   A         C
2021-08-02 -0.578600 -2.426679
2021-08-03  1.265936 -0.678886
2021-08-05  2.205930  1.004054
```

邏輯值(條件式)資料選取 (續)

```
In [7]: df1.iloc[2,2]
Out[7]: -0.6788861516220543

In [8]: df1.iat[2,2]
Out[8]: -0.6788861516220543

In [9]: df1[df1 > 1.5]
Out[9]:
                   A         B   C   D
2021-08-01       NaN       NaN NaN NaN
2021-08-02       NaN  1.651437 NaN NaN
2021-08-03       NaN       NaN NaN NaN
2021-08-04       NaN       NaN NaN NaN
2021-08-05   2.20593  2.186786 NaN NaN
2021-08-06       NaN       NaN NaN NaN

In [10]: df1[df1.A > 1.5]
Out[10]:
                   A         B         C         D
2021-08-05   2.20593  2.186786  1.004054  0.386186
```

使用 .isin -範例1

```
In [11]: # 使用 .isin - 範例1

In [12]: df1[df1.index.isin(['2021-08-02', '2021-08-04'])]
Out[12]:
                   A         B         C         D
2021-08-02 -0.57860  1.651437 -2.426679 -0.428913
2021-08-04  1.49139 -0.638902 -0.443982 -0.434351
```

使用 .isin -範例2

```
In [13]: # 使用 .isin - 範例2

In [14]: df2 = df1.copy()

In [15]: df2['E'] = ['one', 'one','two','three','four','three']

In [16]: df2
Out[16]:
                   A         B         C         D      E
2021-08-01 -1.085631  0.997345  0.282978 -1.506295    one
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913    one
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709    two
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351  three
2021-08-05  2.205930  2.186786  1.004054  0.386186   four
2021-08-06  0.737369  1.490732 -0.935834  1.175829  three

In [17]: df2[df2['E'].isin(['two','four'])]
Out[17]:
                   A         B         C         D     E
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709   two
2021-08-05  2.205930  2.186786  1.004054  0.386186  four
```

Missing Data 遺漏值 NaN

Missing Data 遺漏值

- Python: np.NaN (np.nan)
- R: NA

```
In [20]: df3
Out[20]:
                   A         B         C         D    E
2021-08-01 -1.085631  0.997345  0.282978 -1.506295  1.0
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913  1.0
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709  NaN
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351  NaN
```

```
In [21]: pd.isnull(df3)
Out[21]:
                A      B      C      D      E
2021-08-01  False  False  False  False  False
2021-08-02  False  False  False  False  False
2021-08-03  False  False  False  False   True
2021-08-04  False  False  False  False   True
```

```
In [22]: df3.dropna(how='any')
Out[22]:
                   A         B         C         D    E
2021-08-01 -1.085631  0.997345  0.282978 -1.506295  1.0
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913  1.0
```

```
In [23]: df3.fillna(value=999)
Out[23]:
                   A         B         C         D      E
2021-08-01 -1.085631  0.997345  0.282978 -1.506295    1.0
2021-08-02 -0.578600  1.651437 -2.426679 -0.428913    1.0
2021-08-03  1.265936 -0.866740 -0.678886 -0.094709  999.0
2021-08-04  1.491390 -0.638902 -0.443982 -0.434351  999.0
```

群組 Grouping

Python -群組

- https://github.com/rwepa/DataDemo/blob/master/Cars93.csv

```
In [1]:
   ...: import pandas as pd
   ...: df = pd.read_csv('C:/rdata/Cars93.csv')
   ...:
   ...: df = df[['Manufacturer', 'Price', 'AirBags', 'Horsepower', 'Origin']]

In [2]: df.head()
Out[2]:
  Manufacturer  Price              AirBags  Horsepower   Origin
0        Acura   15.9                 None         140  non-USA
1        Acura   33.9  Driver & Passenger         200  non-USA
2         Audi   29.1         Driver only         172  non-USA
3         Audi   37.7  Driver & Passenger         172  non-USA
4          BMW   30.0         Driver only         208  non-USA
```

Python – groupby {pandas}

```
In [3]: df_AirBags = df.groupby('AirBags')
   ...: type(df_AirBags)
Out[3]: pandas.core.groupby.generic.DataFrameGroupBy

In [4]: print(df_AirBags.groups)
{'Driver & Passenger': [1, 3, 10, 13, 19, 20, 29, 40, 42, 49, 50, 51, 58, 74, 76, 92],
'Driver only': [2, 4, 5, 6, 7, 8, 9, 12, 17, 18, 21, 23, 24, 25, 26, 27, 33, 34, 35,
36, 37, 39, 41, 47, 48, 54, 56, 57, 59, 62, 63, 64, 66, 68, 70, 77, 78, 81, 83, 84,
85, 86, 91], 'None': [0, 11, 14, 15, 16, 22, 28, 30, 31, 32, 38, 43, 44, 45, 46, 52,
53, 55, 60, 61, 65, 67, 69, 71, 72, 73, 75, 79, 80, 82, 87, 88, 89, 90]}
```

Python - 群組 2個維度

```
In [5]:
   ...: df_AirBagsOrigin = df.groupby(['AirBags', 'Origin'])
   ...:
   ...: # 群組大小
   ...: df_AirBagsOrigin.size()
Out[5]:
AirBags              Origin
Driver & Passenger   USA          9
                     non-USA      7
Driver only          USA         23
                     non-USA     20
None                 USA         16
                     non-USA     18
dtype: int64
```

群組大小 size()

```
In [6]:
   ...: df_AirBagsOrigin.size()
Out[6]:
AirBags                Origin
Driver & Passenger     USA           9
                       non-USA       7
Driver only            USA          23
                       non-USA      20
None                   USA          16
                       non-USA      18
dtype: int64
```

篩選群組 get_group()

```
In [7]:
   ...: df_AirBags.get_group('Driver & Passenger')
Out[7]:
     Manufacturer  Price             AirBags  Horsepower   Origin
1           Acura   33.9  Driver & Passenger         200  non-USA
3            Audi   37.7  Driver & Passenger         172  non-USA
10       Cadillac   40.1  Driver & Passenger         295      USA
13      Chevrolet   15.1  Driver & Passenger         160      USA
19       Chrylser   18.4  Driver & Passenger         153      USA
20       Chrysler   15.8  Driver & Passenger         141      USA
29          Eagle   19.3  Driver & Passenger         214      USA
40          Honda   19.8  Driver & Passenger         160  non-USA
42          Honda   17.5  Driver & Passenger         140  non-USA
49          Lexus   35.2  Driver & Passenger         225  non-USA
50        Lincoln   34.3  Driver & Passenger         160      USA
51        Lincoln   36.1  Driver & Passenger         210      USA
58  Mercedes-Benz   61.9  Driver & Passenger         217  non-USA
74        Pontiac   17.7  Driver & Passenger         160      USA
76        Pontiac   24.4  Driver & Passenger         170      USA
92          Volvo   26.7  Driver & Passenger         168  non-USA
```

群組 總和 sum, 平均值 mean

```
In [8]:
   ...: df_AirBags.sum()
Out[8]:
                    Price  Horsepower
AirBags
Driver & Passenger  453.9        2945
Driver only         912.6        6527
None                447.9        3904

In [9]:
   ...: df_AirBags.mean()
Out[9]:
                        Price  Horsepower
AirBags
Driver & Passenger  28.368750  184.062500
Driver only         21.223256  151.790698
None                13.173529  114.823529
```

群組 計算 agg()

```
In [10]:
    ...: df_AirBags.agg('min')
Out[10]:
                    Manufacturer  Price  Horsepower Origin
AirBags
Driver & Passenger         Acura   15.1         140    USA
Driver only                 Audi    9.8          82    USA
None                       Acura    7.4          55    USA

In [11]:
    ...: df_AirBags.agg('max')
Out[11]:
                    Manufacturer  Price  Horsepower   Origin
AirBags
Driver & Passenger         Volvo   61.9         295  non-USA
Driver only                Volvo   47.9         300  non-USA
None                  Volkswagen   23.3         200  non-USA
```

摘要

Python – describe {pandas}

```
In [1]: import pandas as pd
   ...: df = pd.read_csv('C:/mydata/Cars93.csv')
   ...: df
Out[1]:
   Manufacturer    Model     Type  ...  Weight   Origin                Make
0         Acura  Integra    Small  ...    2705  non-USA       Acura Integra
1         Acura   Legend  Midsize  ...    3560  non-USA        Acura Legend
2          Audi       90  Compact  ...    3375  non-USA             Audi 90
3          Audi      100  Midsize  ...    3405  non-USA            Audi 100
4           BMW     535i  Midsize  ...    3640  non-USA            BMW 535i
..          ...      ...      ...  ...     ...      ...                 ...
88   Volkswagen  Eurovan      Van  ...    3960  non-USA  Volkswagen Eurovan
89   Volkswagen   Passat  Compact  ...    2985  non-USA   Volkswagen Passat
90   Volkswagen  Corrado   Sporty  ...    2810  non-USA  Volkswagen Corrado
91        Volvo      240  Compact  ...    2985  non-USA           Volvo 240
92        Volvo      850  Midsize  ...    3245  non-USA           Volvo 850

[93 rows x 27 columns]
```

df.dtypes

```
In [2]: df.dtypes # object: 字串, float64: 含小數點數值
Out[2]:
Manufacturer          object
Model                 object
Type                  object
Min.Price            float64
Price                float64
Max.Price            float64
MPG.city               int64
MPG.highway            int64
AirBags               object
DriveTrain            object
Cylinders             object
EngineSize           float64
Horsepower             int64
RPM                    int64
Rev.per.mile           int64
Man.trans.avail       object
Fuel.tank.capacity   float64
Passengers             int64
Length                 int64
Wheelbase              int64
Width                  int64
Turn.circle            int64
Rear.seat.room       float64
Luggage.room         float64
Weight                 int64
Origin                object
Make                  object
dtype: object
```

df.describe(include='all')

```
In [3]: df.describe() # 無法顯示所有欄位
Out[3]:
       Min.Price      Price  ...  Luggage.room       Weight
count  93.000000  93.000000  ...     82.000000    93.000000
mean   17.125806  19.509677  ...     13.890244  3072.903226
std     8.746029   9.659430  ...      2.997967   589.896510
min     6.700000   7.400000  ...      6.000000  1695.000000
25%    10.800000  12.200000  ...     12.000000  2620.000000
50%    14.700000  17.700000  ...     14.000000  3040.000000
75%    20.300000  23.300000  ...     15.000000  3525.000000
max    45.400000  61.900000  ...     22.000000  4105.000000

[8 rows x 18 columns]

In [4]: df.describe(include='all') # 顯示所有欄位
Out[4]:
       Manufacturer   Model     Type  ...       Weight  Origin           Make
count            93      93       93  ...    93.000000      93             93
unique           32      93        6  ...          NaN       2             93
top       Chevrolet  Spirit  Midsize  ...          NaN     USA  Mercury Capri
freq              8       1       22  ...          NaN      48              1
mean            NaN     NaN      NaN  ...  3072.903226     NaN            NaN
std             NaN     NaN      NaN  ...   589.896510     NaN            NaN
min             NaN     NaN      NaN  ...  1695.000000     NaN            NaN
25%             NaN     NaN      NaN  ...  2620.000000     NaN            NaN
50%             NaN     NaN      NaN  ...  3040.000000     NaN            NaN
75%             NaN     NaN      NaN  ...  3525.000000     NaN            NaN
max             NaN     NaN      NaN  ...  4105.000000     NaN            NaN

[11 rows x 27 columns]
```

None 沒有限制

```
In [5]: pd.set_option('display.max_rows', None, 'display.max_columns', None) # None 沒有限制

In [6]: df.describe()
Out[6]:
       Min.Price      Price  Max.Price   MPG.city  MPG.highway  EngineSize  \
count  93.000000  93.000000  93.000000  93.000000    93.000000   93.000000
mean   17.125806  19.509677  21.898925  22.365591    29.086022    2.667742
std     8.746029   9.659430  11.030457   5.619812     5.331726    1.037363
min     6.700000   7.400000   7.900000  15.000000    20.000000    1.000000
25%    10.800000  12.200000  14.700000  18.000000    26.000000    1.800000
50%    14.700000  17.700000  19.600000  21.000000    28.000000    2.400000
75%    20.300000  23.300000  25.300000  25.000000    31.000000    3.300000
max    45.400000  61.900000  80.000000  46.000000    50.000000    5.700000

       Horsepower          RPM  Rev.per.mile  Fuel.tank.capacity  Passengers  \
count   93.000000    93.000000     93.000000           93.000000   93.000000
mean   143.827957  5280.645161   2332.204301           16.664516    5.086022
std     52.374410   596.731690    496.506525            3.279370    1.038979
min     55.000000  3800.000000   1320.000000            9.200000    2.000000
25%    103.000000  4800.000000   1985.000000           14.500000    4.000000
50%    140.000000  5200.000000   2340.000000           16.400000    5.000000
75%    170.000000  5750.000000   2565.000000           18.800000    6.000000
max    300.000000  6500.000000   3755.000000           27.000000    8.000000

           Length   Wheelbase      Width  Turn.circle  Rear.seat.room  \
count   93.000000   93.000000  93.000000    93.000000       91.000000
mean   183.204301  103.946237  69.376344    38.956989       27.829670
```

顯示所有資料(全部列, 全部行)

```
In [7]: df
Out[7]:
     Manufacturer           Model     Type  Min.Price  Price  Max.Price  \
0           Acura         Integra    Small       12.9   15.9       18.8
1           Acura          Legend  Midsize       29.2   33.9       38.7
2            Audi              90  Compact       25.9   29.1       32.3
3            Audi             100  Midsize       30.8   37.7       44.6
4             BMW            535i  Midsize       23.7   30.0       36.2
5           Buick         Century  Midsize       14.2   15.7       17.3
6           Buick          LeSabre    Large       19.9   20.8       21.7
7           Buick       Roadmaster    Large       22.6   23.7       24.9
8           Buick          Riviera  Midsize       26.3   26.3       26.3
9        Cadillac          DeVille    Large       33.0   34.7       36.3
10       Cadillac          Seville  Midsize       37.5   40.1       42.7
```

檔案匯入 pandas

pandas - IO tools

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata (統計)	read_stata	to_stata
binary	SAS (統計)	read_sas	
binary	SPSS (統計)	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL (資料庫)	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

讀取 Excel

pd.read_excel()

```
# 匯入 Excel 檔案, 全國訂單明細.xlsx
# https://github.com/rwepa/DataDemo/blob/master/全國訂單明細.xlsx

sales = pd.read_excel(io = 'C:/mydata/全國訂單明細.xlsx', sheet_name = '全國訂單明細')
sales # 8568*19
sales.head()
```

```
In [6]: sales.head()
Out[6]:
   訂單號       訂單日期 顧客姓名 訂單等級  ...   產品包箱       運送日期  Unnamed: 19 Unnamed: 20
0    3 2010-10-13   李鵬晨    低級  ...   大型箱子 2010-10-20          NaN          NaN
1    6 2012-02-20   王勇民    其它  ...   小型包裹 2012-02-21          NaN          NaN
2   32 2011-07-15   姚文文    高級  ...   中型箱子 2011-07-17          NaN          NaN
3   32 2011-07-15   姚文文    高級  ...   巨型紙箱 2011-07-16          NaN          NaN
4   32 2011-07-15   姚文文    高級  ...   中型箱子 2011-07-17          NaN          NaN

[5 rows x 21 columns]
```

補充篇

- **sales['產品包箱'].value_counts()**
- **sales['產品包箱'].value_counts(dropna=False)**

資料匯出

to_csv( )

```
# 資料匯出
df = pd.DataFrame({'姓名': ['ALAN', 'LEE'],
                   '地址': ['台北市', '新北市'],
                   '年資': [10, 20]})
df
#      姓名    地址   年資
# 0  ALAN   台北市   10
# 1   LEE   新北市   20

df.to_csv('data/df.csv', index = False)
```

Excel有亂碼

df.csv

```
1 姓名,地址,年資
2 ALAN,台北市,10
3 LEE,新北市,20
```

	A	B	C
1	憪 ?	?啣?	撟湔?
2	ALAN	?啣?擃?10	
3	LEE	?啣?擃?20	

encoding = 'utf_8_sig'

```
df.to_csv('data/df.csv', index = False, encoding = 'utf_8_sig') # OK
```

	A	B	C
1	姓名	地址	年資
2	ALAN	台北市	10
3	LEE	新北市	20

Excel 中文正常

Python 常用模組

模組	功能
Numpy	Large, multi-dimensional arrays and matrices
Scipy	Optimization, linear algebra, integration, FFT, signal
Pandas	DataFrame object for data manipulation
Matplotlib	Static, animated, and interactive visualizations
Statsmodels	Statistical models
Scikit-learn	Machine learning library
Tensorflow, PyTorch	Deep learning
Biopython	Biological computation
Scanpy	Single-cell analysis
Django, Flask, Streamlit	Web
Plotly, dash, bokeh	Interactive visualization

參考資料

- RWEPA
  - http://rwepa.blogspot.com/
- Python 程式設計-李明昌 <免費電子書>
  - http://rwepa.blogspot.com/2020/02/pythonprogramminglee.html
- iPAS Python programming <免費教材>
  - https://github.com/rwepa/ipas_bda/blob/main/ipas-python-program.py
- RWEPA | 登山路線視覺化分析平台 (Python + Streamlit) 【中文字幕】
  - YouTube: https://youtu.be/-_zghs2qrIg
  - Link: https://rwepa.blogspot.com/2023/08/visualization-climbing-routes-with.html
- 多看看 RWEPA 網站免費資源 + YouTube 影片
  - https://www.youtube.com/@alan9956

謝謝您的聆聽

Q & A

李明昌

alan9956@gmail.com

http://rwepa.blogspot.tw/