**Pandas模組(將資料結構化轉換)**

* 為第三方模組需安裝 : ANACONDA/Environments/open terminal/key入”pip install pandas”
* 在LINUX上安裝 :
* **Ubuntu**

$ sudo apt install python3-pandas

* **CentOs**

$ pip3 install pandas

* 線上編輯器 : W3C School
* 可將資料轉成類似表格的DataFrame
* **import pandas as pd**
* *宣告一個DataFrame*

1. df = pd.DataFrame(columns = ['欄一', '欄二', '欄三'])

df >>>不用print

1. df = pd.DataFrame({'欄一': (“甲”,”乙”,”丙”),'欄二': (“a”,”b”,”c”)})

* *備份DataFrame*
* df2 = df.copy()
* *插入row*

1. df.loc[0] = ['Allen','22','175'] >>>0代表第一列



1. df.loc[0] = '0' >>>整row都變0

* *查詢*

1. 第1 ROW的”B”欄

df.loc[1,’B’]

1. 第0和1 ROW

df.loc[[0, 1]] >>>一定要有兩個括號!!!

1. 第0個ROW第1個column

df.iloc[0,1] >>>iloc只能取index

= df.loc[0][1] >>>loc就不能用上面的用法

* *插入column*

1. df['欄四'] = ['75','80','60']

>>>資料量一定要等於ROW的量

1. df.column = ['75','80','60']

* *刪除row*

df = df.drop(2)

>>> 一定要定義” df= ”不然不會永久刪除

* *刪除column*

df = df.drop(['欄四'], axis=1)

* *修改value*

df['欄二'][1] = '新value'

* *直接將資料插入DataFrame*

columns = ['欄一', '欄二', '欄三']

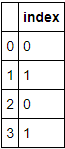
data = [['Ted','24','177'],['Judy','27','183']]

new\_df = pd.DataFrame(data=data, columns=columns)

df = df.append(new\_df) >>> 此時index會錯亂



df.reset\_index() >>> 此時index新舊都會顯示



df = df.reset\_index(drop=**True**)

* *將函數應用於DataFrame*

1. 先定義函數(比如可以做出過濾器把表格不想要的字去乾淨)

for c in df:

df[c] = df[c].apply(函數名) >>> 一定要定義df[c]=

1. (較推!)或用lambda,直接在apply後定義

* *其他功能*

1. 加總

Df.sum(axis=0或1)

1. 看各個row的相關統計(ex:平均)

df.describe()

1. 列出這欄幾個row

df.欄四.shape

1. 看某欄的前五個row

df.欄四.head()

1. 照順序排

df.sort\_values("欄四", inplace = True)

1. 找內容=指定值的欄位

df.欄四.eq(指定值)

1. 找值在某list裡面的欄位

db.欄四.isin(list)

1. 多重條件

df[~df.欄二.isin(list1) & df.欄三.isin(list2)]

>>> “ ~ ”代表不等於

1. 查值有無重複

BB=df["欄四"].duplicated() >>> 顯示TRUE/FALSE

>>> 可用[~BB]移除重複值!

1. 列出所有row且重複的不顯示

df.欄四.drop\_duplicates()

df.欄四.unique()

1. 列出所有不是空值的row

df.欄四.notnull()

1. Print要顯示幾個row

pd.options.display.max\_rows = 數字

* *將DataFrame存成CSV*

df.to\_csv('檔名', index=0, encoding='utf-8-sig', header=**None**)

>>> index=0和header=none是指不要存index和欄位標題

* *讀取CSV檔或EXCEL檔*

1. df = pd.read\_csv('檔名', header=**None**)

>>> 沒有欄位標題才要加header=None,欄位標題會變index

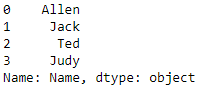
1. df = pd.read\_excel('檔名', sheet\_name=[“sheet1”,”sheet2”]) >>> 如果只想打開特定sheet的話
2. df = pd.read\_csv('檔名',dtype={'A欄': 'int'})

>>> 指定某欄格式轉換成int

1. df.to\_string() >>> 整個表格轉換成str

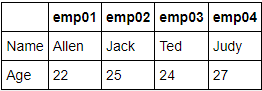
* DataFrame由Series組成, Series就像是一欄
* *宣告Series*

sName = pd.Series(data=['Allen', 'Jack', 'Ted', 'Judy'], name='Name', index=['0', '1', '2', '3'])

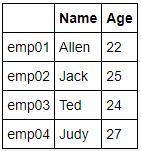


* *把Series放入DataFrame*

1. df.append(pd.Series([name, age]))
2. pd.DataFrame(data=[sName, sAge]) >>> 此時可能行列錯置了



pd.DataFrame(data=[sName, sAge]).T



**--ETL（extract-transform-load）--**

**處理JSON格式資料**

將dataframe讀成JSON時要考慮用哪種orient,會有不同的呈現 :

1. df.to\_json(orient='split')

>>> '{"columns":["col 1","col 2"],"index":["row 1","row 2"],"data":[["a","b"],["c","d"]]}'

1. df.to\_json(orient='index')

>>> '{"row 1":{"col 1":"a","col 2":"b"},"row 2":{"col 1":"c","col 2":"d"}}'

1. df.to\_json(orient='records')

>>> '[{"col 1":"a","col 2":"b"},{"col 1":"c","col 2":"d"}]'

1. df.to\_json(orient='table')

>>> '{"schema":{"fields":[{"name":"index","type":"string"},{"name":"col 1","type":"string"},{"name":"col 2","type":"string"}],"primaryKey":["index"],"pandas\_version":"0.20.0"},"data":[{"index":"row 1","col 1":"a","col 2":"b"},{"index":"row 2","col 1":"c","col 2":"d"}]}'

1. 讀回來 : pd.read\_json(路徑 , orient='???') >>>只有records的方法會少掉index

**處理XML格式資料**

import xml.etree.ElementTree as etree

tree = etree.parse('檔名')

root = tree.getroot()

datatframe = pd.DataFrame(columns = ["name", "email", "age"])

for n in root :

name = n.attrib.get("name")

mail = n.find("email").text if n is not None else None

age = n.find("age").text if n is not None else None

datatframe = datatframe.append(pd.Series([name, mail, age], index = columns), ignore\_index = True)

**處理SQLite格式資料**

from sqlalchemy import create\_engine

engine = create\_engine('sqlite://', echo=False)

df.to\_sql('schema名稱', con=engine) >>>將df寫入SQLite

READ=pd.read\_sql\_table(' schema名稱', con=engine)

或 READ=pd.read\_sql\_query('SELECT \* FROM schema名稱', con=engine)

READ=result.drop('index',axis=1) >>>捨去index

**製作篩選器**

1. fliter= (employees["Gender"] == “Female")

employees[fliter]

1. mask = employees["Team"].isin(["Legal","Scales","Product"])
2. mask = employees["Team"].notnull()
3. employees[employees["Start Date"].between("2009-11-10","2010-01-20")]

**處理文字格式**

1. 轉換成string :

* 物件.astype("string")
* s=pd.Series(['a', 'b', 'c'], dtype="string") >>> Series原是object

1. 把series內用”\_”連在一起的文字分開 : s.str.split('\_')
2. 把series內每row的文字用”,”合在一起,且空值顯示為”-“ : s.str.cat(sep=',' , na\_rep='-')

**處理時間格式**

1. friday = pd.Timestamp('2018-01-05')

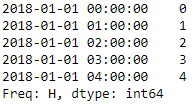
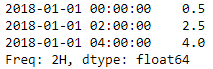
saturday = friday + pd.Timedelta('1 day')

saturday.day\_name() >>> 'Saturday'

1. idx = pd.date\_range('2018-01-01', periods=5, freq='H')

ts = pd.Series(range(len(idx)), index=idx)

ts.resample('2H').mean()

 >>> 

1. pd.Timestamp(datetime.datetime(2012, 5, 1))

>>> Timestamp('2012-05-01 00:00:00')

1. pd.Period('2012-05', freq='D') >>> Period('2012-05-01', 'D')
2. pd.to\_datetime(欄位) >>>轉為’2012-05-01’形式

**遺失值處理**

Pandas認為None和NaN(Not a number)都是遺失值

偵測 : df.isnull( ) >>> 

填補 :

1. df.fillna(填補值) 或 df.fillna(method ='pad’, inplace = True)

* ‘pad’ : 用前一個值來填補遺失值 / ‘bfill’ : 用後一個值來填補遺失值
* inplace = True代表直接更改原表格

1. df.replace(to\_replace= 要替換的原值, value = 替換成的值)
2. df.interpolate(method ='linear', limit\_direction='forward')

>>>內插法 : 用前後值的平均來填補

移除 :

df.dropna(inplace = True) >>>移除帶有空值的整row

**離群值(極端值)處理**

**離群值(outlier)判準：**

1. 在3rdQuartier 之上和1stQuartier 之下1.5 倍interquartile range (IQR)

(IQR=3rdQuartier –1stQuartier)

1. 與平均值差超過3個標準差

**離群值偵測：**

outliers=[ ]

def detect\_outlier(df) :

threshold=3 >>>3個標準差

mean\_1 = np.mean(df)

std\_1 =np.std(df)

for y in df :

z\_score= (y -mean\_1)/std\_1

if np.abs(z\_score) > threshold : >>>取絕對值

outliers.append(y)

return outliers

**離群值移除：**

z\_scores= stats.zscore(df) >>>將整個DF取z-score

abs\_z\_scores= np.abs(z\_scores)

filtered\_entries= (abs\_z\_scores< 3).all(axis=1)

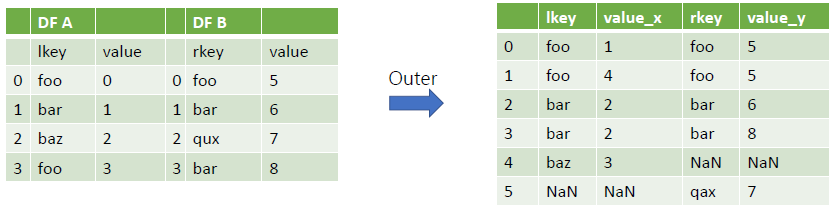
new\_df= df[filtered\_entries]

print(new\_df)

**資料合併與重塑**

1. **Merge(以column為依據做合併)**

pd.merge( [DF\_A ,DF\_B] ,left\_on='lkey' , right\_on='rkey' ,how='outer' )



1. **Join(以key column/index為依據做合併)**

DataFrame.join([A,B] , on='key')



1. **Concat(以index為依據做合併):**

**#Outer Join, Axis = 0 (內建)**

Result=pd.concat([df1, df2, df3], keys=['x', 'y', 'z’]) >>>把三個df用xyz的index標記

Result.loc['y'] >>>只顯示某個df的部分



**#Outer Join, Axis = 1**

pd.concat([df1, df4], axis=1)



**#Inner Join, Axis = 1**

pd.concat([df1, df4], axis=1, join='inner')



1. **Stack(把Columns堆疊成Row (index))/Unstack(把Row (index)展開成Columns)**

df.stack()

****

**Numpy套件-ND array**

* 是一種由資料所構成的N維容器,可用來做矩陣運算
* 只能存放同樣資料類別（dtype）的資料



* 在LINUX上安裝 :
* **Ubuntu**

$ sudo apt install python3-numpy

* **CentOs**

$ sudo yum install numpy scipy

* **基本用法**

**import numpy as np**

* + 1. **x = np.array([[1, 2, 3], [4, 5, 6]])**

x >>>array([[1, 2, 3],

[4, 5, 6]])

**x.shape** >>(2, 3) >>>因為是2 x 3的矩陣

**x.ndim**  >>>看維度

**x.size** >>>看有幾筆資料

**x.itemsize** >>>每個元素的大小(byte)

**x.dtype** >>>看資料型別

※ **np.dtype('i4') :** 查詢i4是什麼dtype >>>int32

* + 1. **x = np.array([1, 2, 3],ndmin=5)** >>>指定維度

>>> [[[[[1, 2, 3]]]]]

* + 1. **取值**
* **x[0,2]** >>>第”0”row的第”2”個column >>> 3
* **Fancy Indexing :** 用index來呼叫矩陣裡的內容(類似序列)

x = np.array([1, 2, 3, -1, 4])

ind = [2,4]

x[ind] >>> array([3,4])

* **rows = np.array(要取哪幾ROW)**

**cols = np.array(要取哪幾個COLUMN)**

**arr2 = arr1[rows,cols]**

* **迭代器**

**for x in np.nditer(arr,order='C'or'F')**

**或 for x in arr.flat :**

**print (x)**

>>> C:從ROW1開始 / F:從COLUMN1開始

* + 1. **填入值 : x.fill(0)** >>>全部值變1
    2. **偵測遺失值 : np.isnan(x)** >>>array([True])
    3. **Masked Array : 把特定位置的值mask掉,就可在運算時不列入計算**

import numpy.ma as ma

x = np.array([1, 2, 3, -1, 4])

mx = ma.masked\_array(x, mask=[0, 0, 0, 1, 0]) >>>0或1可改成False, True

mx.mean() =2.5 >>> 由布林值(0或1)構成的Array,1不列入計算

np.ma.masked\_invalid(x) >>>可把遺失值變成”-- ”

* + 1. **np.sum(x , axis = 0)** >>>垂直相加 >>> array([5, 7, 9])
    2. **轉換型別 : x.astype(型別)**
    3. **自定義型別**

**from numpy import \***

**dt = dtype([('name',str\_,20),('age',int8),('salary',float32)])**

>>>代表array的每個column的index分別是'name','age','salary'，型別分別是'str','int8','float32'

**arr = array([('a',1,2.4)],dtype=dt)** >>賦予dtype

* + 1. **做出簡易矩陣**
       - **x =np.arange (10)** >>> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

**x.reshape (2,5)**  >>>重塑成2x5的矩陣

**= x.reshape (2,-1)**  >>>不想自己算就打”-1”

* **x = np.arange(10,2,-1)** >>> array([10, 9, 8, 7, 6, 5, 4, 3])

\*注意這裡( )內是”, ”不是”: ”

x[1:7:2] >>>用index取值 >>>array([9, 7, 5])

x [np.array([[1,1],[2,3]])] >>> array([[9, 9],

[8, 7]]) >>>這樣也可以重塑

* + - * **值為0的矩陣 : np.zeros((row,column), dtype = int)**
      * **值為1的矩陣 : np.ones((row,column),dtype=int)**
      * **等差數列 : np.linspace(起始值,最末值,總共幾個數)**
      * **等比數列 : np.logspace(起始指數,最末指數,總共幾個數,base=底數)**
    1. **產生亂數**
* **rand = np.random.RandomState(隨便一個數字)**

**x = rand.randint(隨便一個數字, size=想產生幾個數字)**

* **np.random.normal(平均值, 標準差, 要產生幾個數字)**
* **np.random.randn(列,行) for \_ in range(要產生幾組矩陣)**
* **np.empty((列,行), dtype = 型態)**
  + 1. A = np.array([[a,b],[c,d]])

B = np.array([[1,2],[3,4]])

**A\*B (element wise multiplication)**  >>> array([[a1,b2],

[c3,d4]])

**A@B (matrix multiplication) = A.dot(B)**

>>> array([[a1+b3,a2+b4],

[c1+d3,c2+d4]])

* + 1. **Broadcasting(純量在與矩陣運算時,被視為等同向量)**
       1. 乘法 : 若b=2,在跟矩陣相乘時,可被視為二維的array([2, 2])
       2. 加法 : a = array([[ 0, 0, 0],

[10, 10, 10],

b = array([ 1, 2, 3])

**a + b** >>>array([[ 1, 2, 3],

[11, 12, 13],

* + 1. **合併(concatenate)**  >>> ex : 2x2+2x2變4x2

**垂直合併**

**np.concatenate((array1, array2) ,axis=0) = np.vstack((array1, array2))**

**水平合併**

**np.concatenate((array1,array2),axis=1) = np.hstack((array1, array2))**

>>>axis一定要小於矩陣的維數,不然無法成立

>>>所以一維的矩陣不能水平合併

* + 1. **疊加（stack）** >>> 沿著axis的方向去擴增,維度增加

array1 = np.array([1, 2, 3])>>>一維

array2 = np.array([2, 3, 4]) >>>一維

**np.stack((array1, array2), axis=0)** >>>array([[1, 2, 3],

[2, 3, 4]]) >>>二維

**np.stack((array1, array2), axis=-1或1) >>>**array([[1, 2],

[2, 3],

[3, 4]]) >>>翻轉

**np.concatenate((array1, array2),axis=0) >>>**array([1, 2, 3, 2, 3, 4])

>>>一維垂直延伸會變二維,不符合concatenate原則,所以只能水平延伸

* + 1. **攤平成一維**
* **arr.flatten()** >>>但只會產生copy,原矩陣不會變
* **arr.ravel()** >>>會更改原本的矩陣
  + 1. **翻轉**
* **np.rollaxis(arr,要翻轉的軸,0)**
* **np.swapaxes(arr,要對調的第一個軸,要對調的另一個軸)**
  + 1. **統計**
* **最小值 :** np.min(array名)
* **最大值 :** np.max(array名)
* **平均值 :** np.mean(array名)
* **中位數 :** np.median(array名)
* **眾數 :** np.mode(array名)[0]
* **標準差 :** np.std(array名)
* **最大值-最小值 :** np.ptp(array名)

**Matplotlib 套件-(繪圖)**

* **DataFrame直接轉為圖表**

import matplotlib.pyplot as plt

* df.plot() #折線圖

df.plot(kind='hist') #柱狀圖

df.plot(kind='scatter',x='X軸名',y='Y軸名') #散佈圖

* plt.show()
* plt.savefig('檔名') #存檔
* **直線圖**

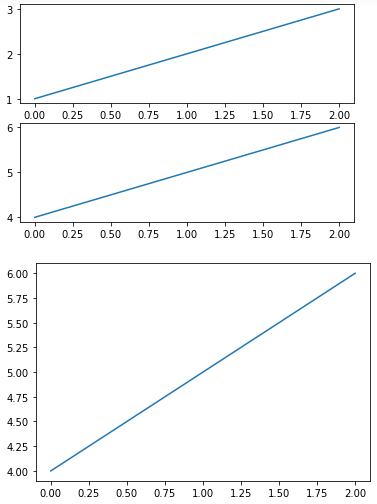
import matplotlib.pyplot as plt

plt.figure(1) # 創造第1個figure

plt.subplot(211) # 創造subplot # 默認值 (111)

#三個數字代表(垂直顯示幾個圖,平行顯示幾個圖,圖的序號)

1. plt.plot([1, 2, 3]) # y軸 #x軸默認值([0, 1, 2])



1. x = range(-8, 8)

y = [i\*\*3 +2\*i\*\*2+i - 1 for i in x]

pl.plot(x, y, 'ob--', label=u'y=x^3+2x^2+x-1')

#o是圈圈 b是藍色 --是虛線 # label代表圖標文字

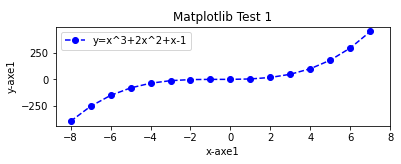
pl.legend() #創造圖標

pl.xticks([-8, -6, -4, -2, 0, 2, 4, 6, 8]) #x軸數字

pl.title('標題')

pl.xlabel('x軸名')

pl.ylabel('y軸名')



* **圓餅圖**

import matplotlib.pyplot as plt

labels = 'Frogs', 'Hogs', 'Dogs', 'Logs' #派名

sizes = [15, 30, 45, 10] #占比

explode = (0, 0.1, 0, 0) #第二個會凸顯出來

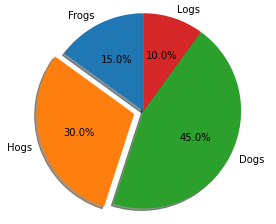
fig1, ax1 = plt.subplots()

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',

shadow=True, startangle=90) #逆時針繪製

ax1.axis('equal') # 確保能變圓形

plt.show()



* **Scatter Chart**

import matplotlib.pyplot as plt

np.random.seed(任意數字)

x = ~~~

y = ~~~

area = (~~~)\*\*2 #圖的面積

c = np.sqrt(area) #產生面積為area的圖

r = np.sqrt(x \*\* 2 + y \*\* 2)

r0 = 0.6 #想要以0.6為半徑畫一條區隔線

area1 = np.ma.masked\_where(r < r0, area) #遮掉r<0.6的area

area2 = np.ma.masked\_where(r >= r0, area) #遮掉r>=0.6的area

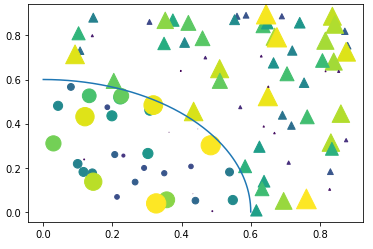
plt.scatter(x, y, s=area1, marker='^', c=c) #在area把點畫成三角形,散佈在area1

plt.scatter(x, y, s=area2, marker='o', c=c) #在area把點畫成圓形,散佈在area2

theta = np.arange(0, np.pi / 2, 0.01)

plt.plot(r0 \* np.cos(theta), r0 \* np.sin(theta)) #畫出以0.6為半徑的區隔線

plt.show()



* **Histogram**

import matplotlib.pyplot as plt

from matplotlib import colors

from matplotlib.ticker import PercentFormatter

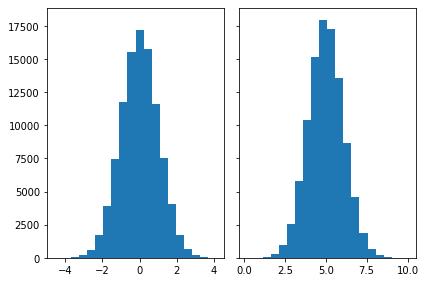
x = ~~~

y = ~~~

fig, axs = plt.subplots(1, 2, sharey=True, tight\_layout=True)

axs[0].hist(x, bins=要幾條柱) >>>第一個圖

axs[1].hist(y, bins=要幾條柱) >>>第二個圖



* **BAR CHART**

import matplotlib.pyplot as plt

menMeans = (20, 35, 30, 35, 27)

womenMeans = (25, 32, 34, 20, 25)

menStd = (2, 3, 4, 1, 2)

womenStd = (3, 5, 2, 3, 3)

index = np.arange(5) # 有五條bar

width = 0.35 # bar寬

p1 = plt.bar(index, menMeans, width, yerr=menStd)

p2 = plt.bar(index, womenMeans, width, bottom=menMeans, yerr=womenStd)

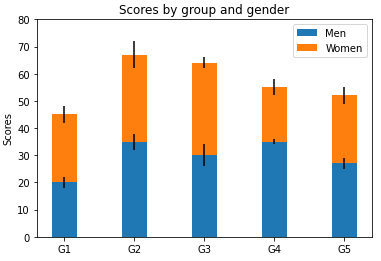
#底部疊在menMeans上

plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))

plt.yticks(np.arange(0, 81, 10))

plt.legend((p1[0], p2[0]), ('Men', 'Women')) #圖標

plt.show()



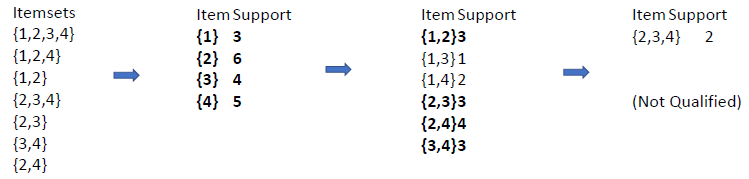
**關聯（規則）分析**

****

* **Support :** 前因後果同時發生的機率
* **Confidence :** 前因對後果發生影響的機率
* **Lift :** 前因後果共同和獨立發生的機率差異,>1表示有影響

**做關聯分析的演算法**

1. **Apriori（先驗）演算法 :** 一層一層迭代,較費時費空間



1. **FP-growth演算法 :** 透過FP-Tree查找,效率較高

****