8. Pandas

章節內容

- 介紹
- 安裝
- 數據幀 DataFrame
- 匯入和匯出數據
- 數據幀的過濾和選擇 Filtering & Selection of a DataFrame
- 缺失數據處理
- 重塑 DataFrame
- 與資料庫交互
- Df.groupby
- 添加行數據
- Df.sort_values
- Df.set_index，df.reset_index
- 更改列順序
- Series.unique、series.nunique
- pd.read_csv，pd.read_json
- 時間序列 Time Series
- 分類數據類型 Categorical Data Type
- df.rolling
- df.where、df.mask

介紹

- Pandas 這個名字來源於 Panel Data 和 Python Data Analysis
- 世界級的開源專案
- 為數據操作和分析而編寫的庫
- DataFrame 中用於整合操作的優勢
- 採用 Numpy 風格陣列計算的重要部分
- 與 Numpy、SciPy、Scikit-learn、Matplotlib 等相關聯

安裝

在 iPython 或 Jupyter 或終端機 Terminal 上

```
pip install pandas
```

檢查版本

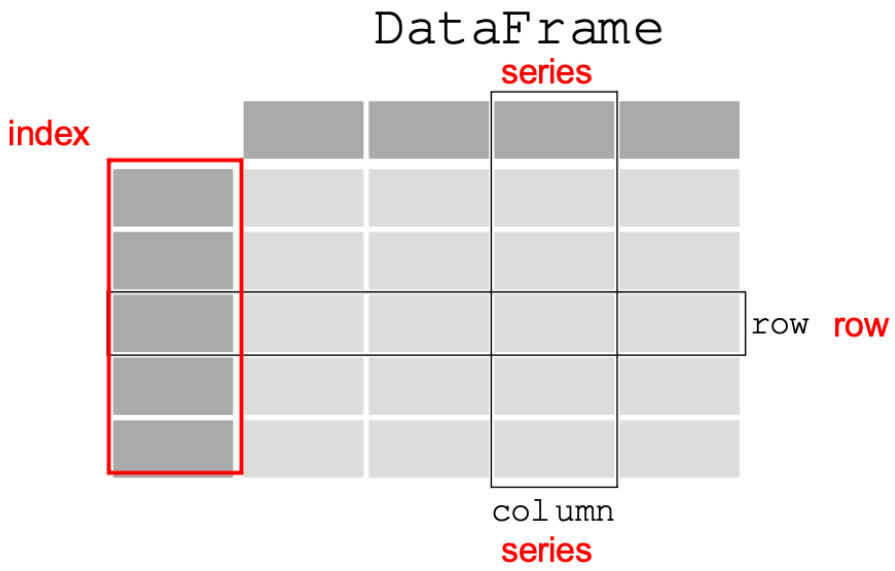
```
import pandas as pd
pd.__version__
```

```
'1.5.2'
```

通用術語翻譯

General terminology translation

pandas	Excel
`DataFrame`	worksheet
`Series`	column
`Index`	row headings
row	row
`NaN`	empty cell



Series 是一個類似 np.array 的一維物件，包含一系列值和關聯的數據標籤陣列（索引）。

```
obj = pd.Series([3.4, 2.5, 6.8, 4.9, 5.8])
obj
```

```
0    3.4
1    2.5
2    6.8
3    4.9
4    5.8
dtype: float64
```

DataFrame 是二維的、大小可變的、可能異構的表格數據。

In [2]:
```
df = pd.DataFrame(
    {
        "Name": ["Alice", "Bobby", "Charlie", "Dorathy" ],
        "Gender": ["female", "male", "male", "female"],
        "Age": [25, 30, 58, 42],
        "Height": [1.68, 1.72, 1.78, 1.63]
    }
)
df
```

Out[2]:

	Name	Gender	Age	Height
0	Alice	female	25	1.68
1	Bobby	male	30	1.72
2	Charlie	male	58	1.78
3	Dorathy	female	42	1.63

簡單的 DataFrame(DF)操作

```
In [3]: df["Name"]
```

```
Out[3]: 0       Alice
        1       Bobby
        2     Charlie
        3     Dorathy
        Name: Name, dtype: object
```

```
In [4]: df["Height"].max()
```

```
Out[4]: 1.78
```

```
In [5]: df["Age"].mean()
```

```
Out[5]: 38.75
```

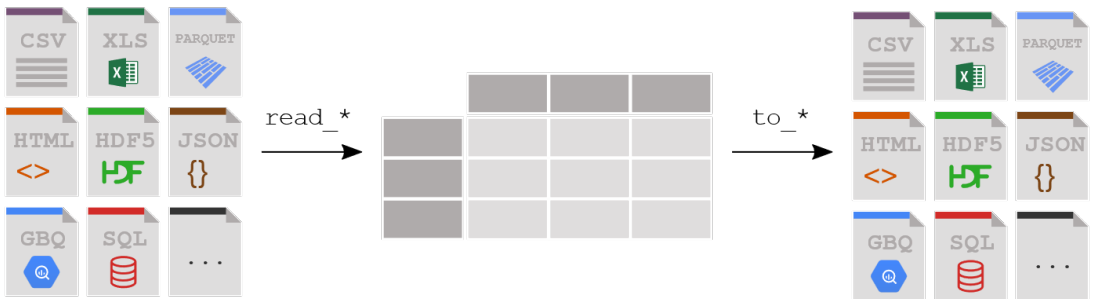
```
In [6]: df.describe()
```

Out[6]:

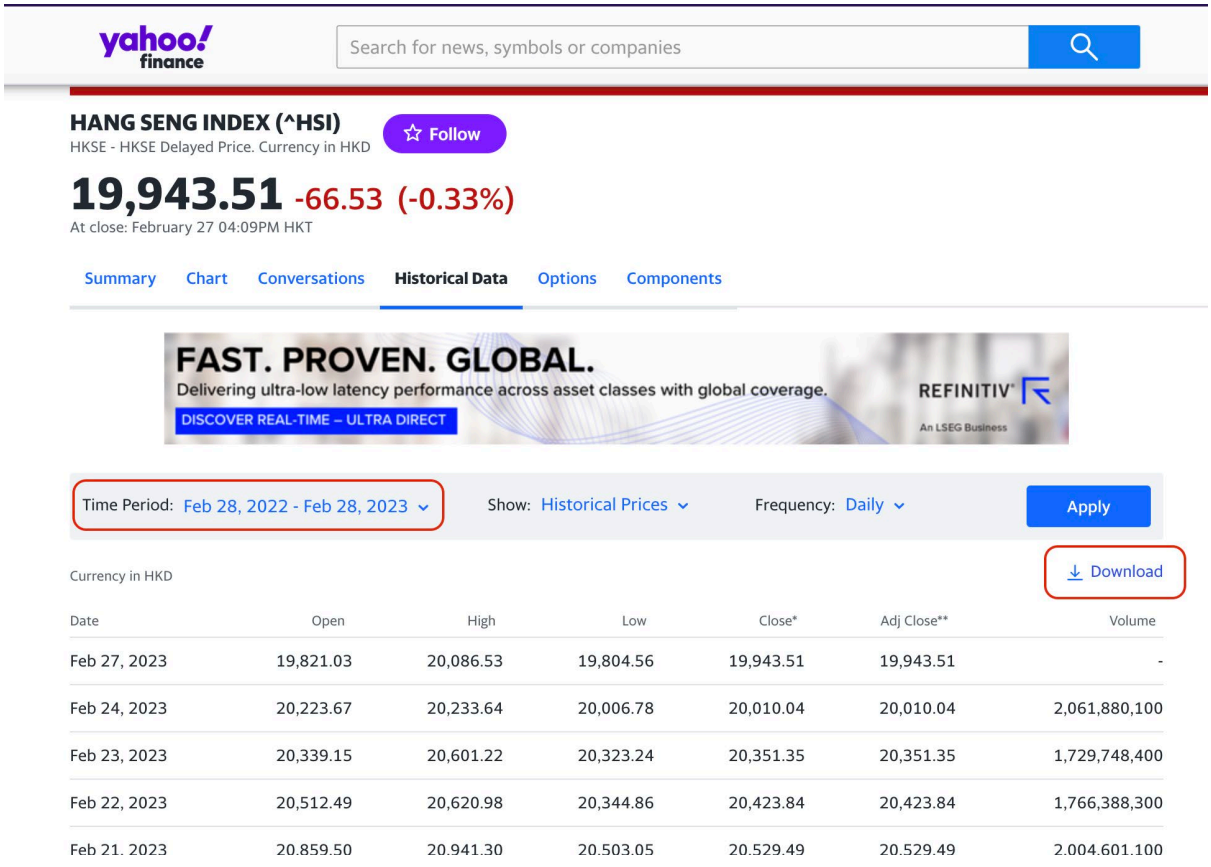
	Age	Height
count	4.000000	4.000000
mean	38.750000	1.702500
std	14.682756	0.063443
min	25.000000	1.630000
25%	28.750000	1.667500
50%	36.000000	1.700000
75%	46.000000	1.735000
max	58.000000	1.780000

匯入和匯出數據

- 在大多數情況下，我們導入數據以從現有數據集構建 DataFrame
- 出於學習目的，我們經常使用.csv



準備 csv 數據集



Date	Open	High	Low	Close*	Adj Close**	Volume
Feb 27, 2023	19,821.03	20,086.53	19,804.56	19,943.51	19,943.51	-
Feb 24, 2023	20,223.67	20,233.64	20,006.78	20,010.04	20,010.04	2,061,880,100
Feb 23, 2023	20,339.15	20,601.22	20,323.24	20,351.35	20,351.35	1,729,748,400
Feb 22, 2023	20,512.49	20,620.98	20,344.86	20,423.84	20,423.84	1,766,388,300
Feb 21, 2023	20,859.50	20,941.30	20,503.05	20,529.49	20,529.49	2,004,601,100

將.csv 文件導入 Pandas DataFrame

- 將.csv 保存在筆記本 .ipynb 的同一資料夾中

```
In [7]: df_hsi = pd.read_csv("^HSI.csv")
        df_hsi
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-01-03	23510.539063	23605.029297	23193.189453	23274.750000	23274.750000	734331100
1	2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200
2	2022-01-05	23323.769531	23323.769531	22851.500000	22907.250000	22907.250000	2768859000
3	2022-01-06	22843.199219	23082.949219	22709.599609	23072.859375	23072.859375	1765786100
4	2022-01-07	23318.919922	23497.500000	23162.849609	23493.380859	23493.380859	2602290600
...	...	...	...	...	...	...	...
241	2022-12-22	19537.449219	19735.000000	19475.679688	19679.220703	19679.220703	1939795100
242	2022-12-23	19382.230469	19686.769531	19380.470703	19593.060547	19593.060547	1363741800
243	2022-12-28	19787.939453	20099.769531	19787.939453	19898.910156	19898.910156	2823780600
244	2022-12-29	19648.400391	19764.519531	19539.839844	19741.140625	19741.140625	2902362900
245	2022-12-30	20030.849609	20073.919922	19781.410156	19781.410156	19781.410156	1747706800

246 rows × 7 columns

從 yfinance 導入數據

或者，我們通過 yfinance 包下載數據集。

請注意，yfinance 不是雅虎的附屬公司，而是熱心極客的作品

```
pip install yfinance
```

```
Collecting yfinance
  Downloading yfinance-0.2.12-py2.py3-none-any.whl (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.2/59.2 kB 54
Requirement already satisfied: numpy>=1.16.5 in /Users/honcyl
rom yfinance) (1.23.1)
Requirement already satisfied: pandas>=1.3.0 in /Users/honcyl
rom yfinance) (1.5.2)
Collecting multitasking>=0.0.7
```

In [9]:
```
import yfinance as yf
df_HSI = yf.download("^HSI", start="2022-01-01", end="2023-01-01")
```

```
[**********************100%**********************]  1 of 1 completed
```

In [10]:
```
df_HSI
```

Out[10]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-03	23510.539062	23605.029297	23193.189453	23274.750000	23274.750000	734331100
2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200
2022-01-05	23323.769531	23323.769531	22851.500000	22907.250000	22907.250000	2768859000
2022-01-06	22843.199219	23082.949219	22709.599609	23072.859375	23072.859375	1765786100
2022-01-07	23318.919922	23497.500000	23162.849609	23493.380859	23493.380859	2602290600
...	...	...	...	...	...	...
2022-12-22	19537.449219	19735.000000	19475.679688	19679.220703	19679.220703	1939795100
2022-12-23	19382.230469	19686.769531	19380.470703	19593.060547	19593.060547	1363741800
2022-12-28	19787.939453	20099.769531	19787.939453	19898.910156	19898.910156	2823780600
2022-12-29	19648.400391	19764.519531	19539.839844	19741.140625	19741.140625	2902362900
2022-12-30	20030.849609	20073.919922	19781.410156	19781.410156	19781.410156	1747706800

246 rows × 6 columns

In [11]:
```
print(type(df_hsi))
print(type(df_HSI))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

用於啟動 DataFrame 的更多基本工具

DF.head( )

DF.tail( )

DF.shape

DF.info( )

DF.to_datetime( )

DF.set_index( )

DF.reset_index( )

DF.describe( )

DF.isna( )

DF.isna( ).values.any( )

您能猜測嗎，為什麼我們需要一個索引嗎？

數據幀的過濾和選擇 Filtering & Selection of a DataFrame

Type	Notes
`df[val]`	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
`df.loc[val]`	Selects single row or subset of rows from the DataFrame by label
`df.loc[:, val]`	Selects single column or subset of columns by label
`df.loc[val1, val2]`	Select both rows and columns by label
`df.iloc[where]`	Selects single row or subset of rows from the DataFrame by integer position

```
df_hsi['關閉']     # 選擇為 pd 系列
df_hsi[['Close']]    # 選擇 pd DF
df_hsi[['Close'，'Volume']]   # 選擇兩列作為 pd DF
df_hsi.loc[df_hsi['關閉'] > df_hsi['打開']]   # 有條件選擇
df_hsi.loc[df_hsi.index >= '2022-03-01']    # 按索引定位
df_hsi.loc[df_hsi.index == '2022-07-04']    # 過濾特定日期
df_hsi.iloc[0：5]     # 基於整數位置的索引，用於按位置選擇
```

數據幀的過濾和選擇 Filtering & Selection of a DataFrame

> REMEMBER
>
> - When selecting subsets of data, square brackets `[]` are used.
> - Inside these brackets, you can use a single column/row label, a list of column/row labels, a slice of labels, a conditional expression or a colon.
> - Select specific rows and/or columns using `loc` when using the row and column names.
> - Select specific rows and/or columns using `iloc` when using the positions in the table.
> - You can assign new values to a selection based on `loc`/`iloc`.

Object Type	Indexers
Series	`s.loc[indexer]`
DataFrame	`df.loc[row_indexer,column_indexer]`

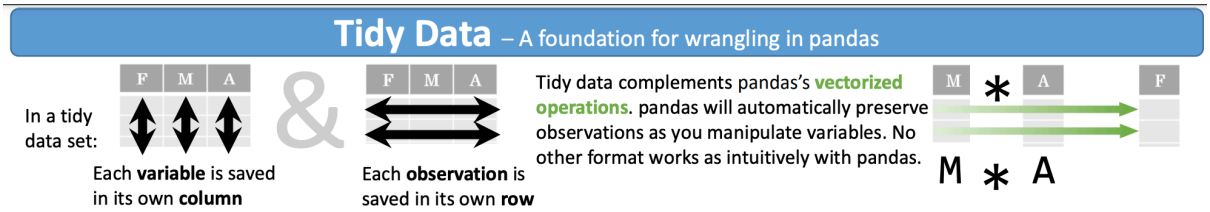
如果只想訪問標量值，最快的方法是使用 **.at** 和 **.iat** 方法，這些方法在所有數據結構上都實現。

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-03	23510.539063	23605.029297	23193.189453	23274.750000	23274.750000	734331100
2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200
2022-01-05	23323.769531	23323.769531	22851.500000	22907.250000	22907.250000	2768859000
2022-01-06	22843.199219	23082.949219	22709.599609	23072.859375	23072.859375	1765786100
2022-01-07	23318.919922	23497.500000	23162.849609	23493.380859	23493.380859	2602290600

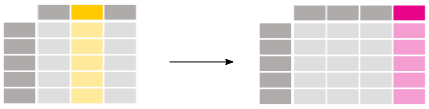
```
In [34]: df_hsi.at[df_hsi.index[4], 'Open']
Out[34]: 23318.919922

In [35]: df_hsi.iat[4, 4]
Out[35]: 23493.380859
```

添加新列

在幾乎每一次分析中，我們都需要通過計算創建額外的列，在 Pandas 中，邏輯非常簡單、整潔和快速。感謝矢量化數據管理！



新列中的值的計算是 按元素計算的，您不需要創建迴圈。只需使用 Pandas 內置功能即可。



```
In [34]: df_hsi["Return"] = df_hsi["Adj Close"].pct_change(1)   # percentage change from 1 day before
         df_hsi
```

Out[34]:

Date	Open	High	Low	Close	Adj Close	Volume	Return
2022-01-03	23510.539063	23605.029297	23193.189453	23274.750000	23274.750000	734331100	NaN
2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200	0.000648
2022-01-05	23323.769531	23323.769531	22851.500000	22907.250000	22907.250000	2768859000	-0.016427
2022-01-06	22843.199219	23082.949219	22709.599609	23072.859375	23072.859375	1765786100	0.007230
2022-01-07	23318.919922	23497.500000	23162.849609	23493.380859	23493.380859	2602290600	0.018226

```
In [35]: # add 10,20,50 days moving average
         for period in [10,20,50]:
             df_hsi[f"MA{period}"] = df_hsi["Adj Close"].rolling(period).mean()
         df_hsi.tail(5)
```

Out[35]:

Date	Open	High	Low	Close	Adj Close	Volume	Return	MA10	MA20	MA50
2022-12-22	19537.449219	19735.000000	19475.679688	19679.220703	19679.220703	1939795100	0.027073	19474.073047	19052.523438	17540.685566
2022-12-23	19382.230469	19686.769531	19380.470703	19593.060547	19593.060547	1363741800	-0.004378	19443.292187	19153.497461	17600.792988
2022-12-28	19787.939453	20099.769531	19787.939453	19898.910156	19898.910156	2823780600	0.015610	19486.820117	19283.545996	17666.513184
2022-12-29	19648.400391	19764.519531	19539.839844	19741.140625	19741.140625	2902362900	-0.007929	19501.314258	19360.369043	17723.044395
2022-12-30	20030.849609	20073.919922	19781.410156	19781.410156	19781.410156	1747706800	0.002040	19512.110351	19419.578027	17788.447012

刪除列或行

DataFrame.drop（*labels=None*， *，*axis=0*， *index=None*， *columns=None*，*level=None*， *inplace=False*， *errors='raise'*）

```
In [38]: # Drop columns
         df_hsi = df_hsi.drop(['MA20'], axis=1)
         df_hsi
```

Out[38]:

Date	Open	High	Low	Close	Adj Close	Volume	Return	MA10	MA50
2022-01-03	23510.539063	23605.029297	23193.189453	23274.750000	23274.750000	734331100	NaN	NaN	NaN
2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200	0.000648	NaN	NaN

```
In [37]: # Drop row
         df_hsi = df_hsi.drop(index=('2022-01-03'))
         df_hsi
```

Out[37]:

Date	Open	High	Low	Close	Adj Close	Volume	Return	MA10	MA50
2022-01-04	23400.619141	23439.300781	23146.890625	23289.839844	23289.839844	1760141200	0.000648	NaN	NaN
2022-01-05	23323.769531	23323.769531	22851.500000	22907.250000	22907.250000	2768859000	-0.016427	NaN	NaN

缺少數據處理

在百萬行的資料庫中，總會有一些不規則的數據，

例如 Null、NaN、None、Empty 或不是定義的 Series 的類型。

根據研究的需要，我們選擇以下其中一種來處理。

- DF.isna（ ）# 表示值是否為 NA。NA 值，例如 None 或 numpy。南
- DF.dropna（ ）# 包含 NA 的下拉行
- DF.fillna（method=“ffill”）# 用上行數據填充 NA
- DF.fillna（0）# 用零填充 NA
- DF.fillna（value=some_values）# 用特定值填充 NA，例如平均值或中位數
- DF['xxx'].interpolate（ ）# 用介於上下之間的值填充 NA

沒有硬性規定來處理 NA，做出合理的選擇。

更多觀察工具

- DF.nlargest（n， 'column'）# 選擇並排序前 n 個條目
- DF.nsmallest（n， 'column'）# 選擇並排序前 n 個條目
- DF.sample（n=5）# 隨機選擇 5 行
- DF.iloc[[0]] # 選擇 DF 上的第一行
- DF.iloc[[-1]] # 選擇 DF 上的最後一行
- DF.loc[（DF.index>？？） & （DF.index<？？）] # 有條件選擇

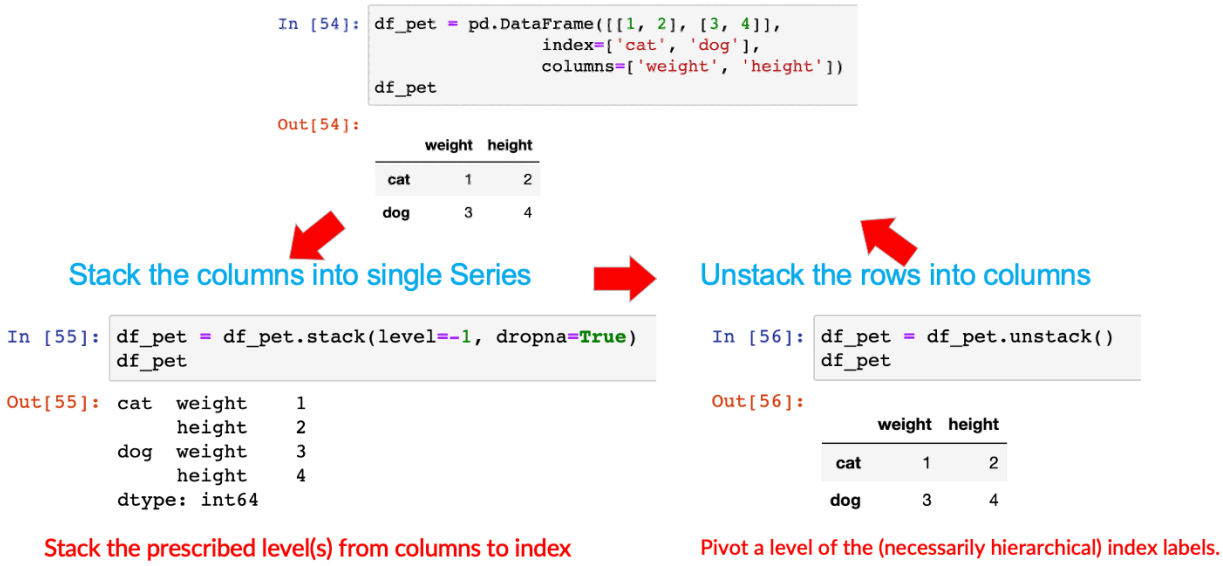
注意： 如有必要，請將過濾后的數據存儲在新 DF 或原始 DF 中。

重塑 DF

根據我們擁有的原始資料庫，有時我們需要從列到行進行重塑，反之亦然。

DF.stack（ ）

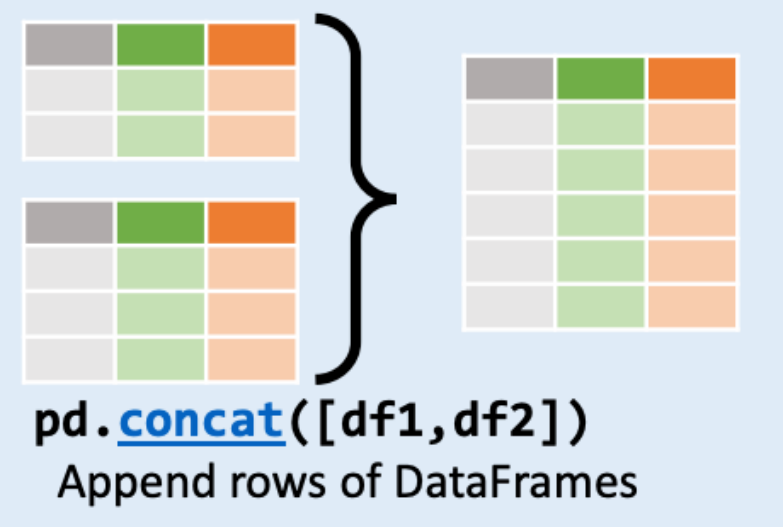
DF.unstack（ ）



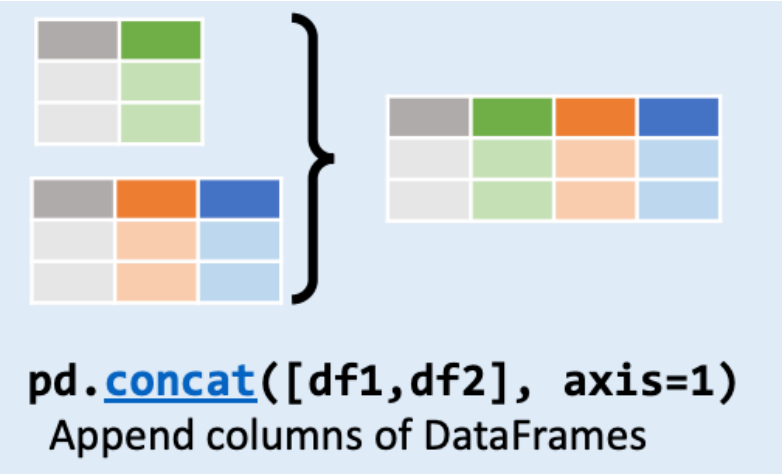
我們也可以用 stack()建立 3D 數據，但為操作簡便 2D 數據較優勝。

連接 DF

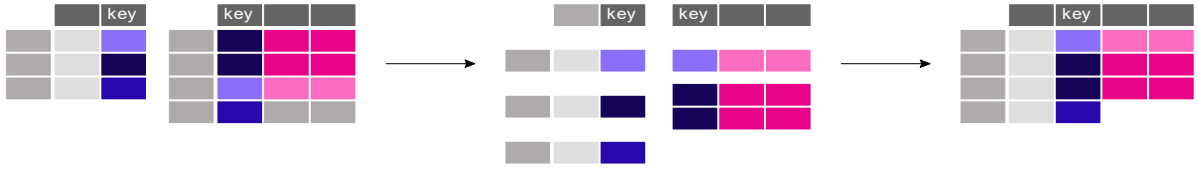
- 追加行



- 追加列或系列



將 DF 與 KEY 合併



df_merge = pd.merge（df1， df2， how='left'， on='鍵'）

In [64]:
```
df1 = pd.DataFrame({'Name': ['Alex', 'Bob', 'Chris'], 'ID': [1, 2, 3]})
df1
```

Out[64]:

	Name	ID
0	Alex	1
1	Bob	2
2	Chris	3

In [65]:
```
df2 = pd.DataFrame({'ID':[1,3],'Score':[65,60]})
df2
```

Out[65]:

	ID	Score
0	1	65
1	3	60

In [66]:
```
df = pd.merge(df1,df2, how='left', on='ID')
df
```

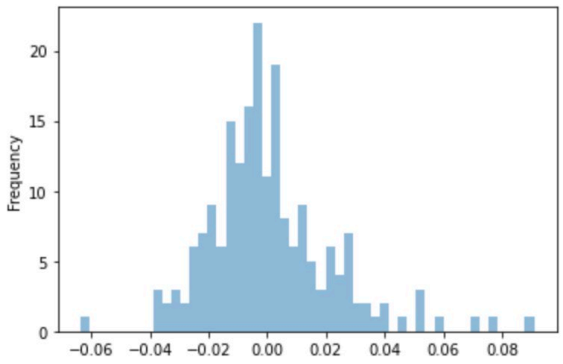
Out[66]:

	Name	ID	Score
0	Alex	1	65.0
1	Bob	2	NaN
2	Chris	3	60.0

繪製 DataFrame – 直方分佈圖 Histogram

Pandas 預設圖表庫是 Matplotlib。我們可以用簡單的代碼繪製一個快速圖表。

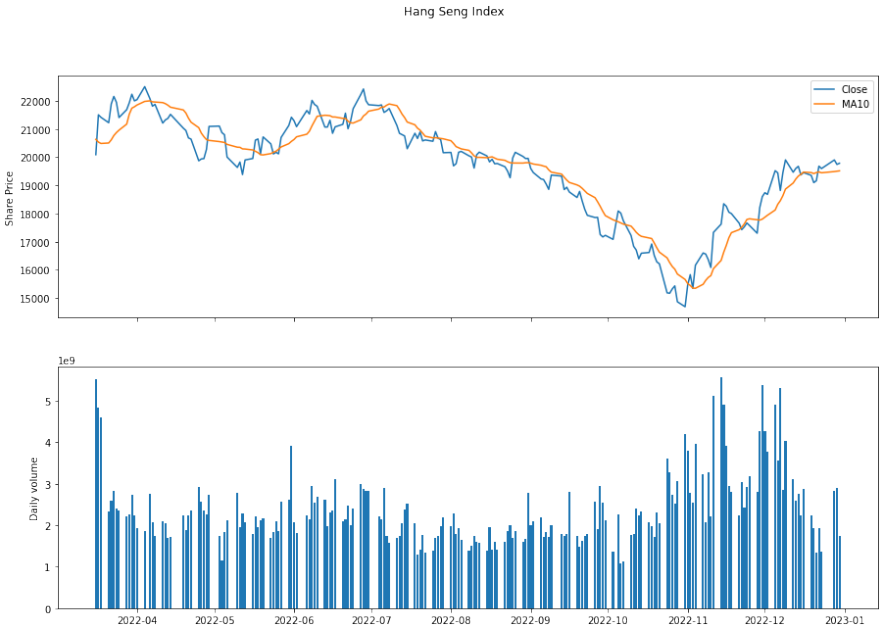
```
# Easy plot with Pandas plot.hist
ax = df_hsi['Return'].plot.hist(bins=50, alpha=0.5)
```



我們可以用 Matplotlib 繪製一個更好的圖表

```
# pip install matplotlib
```

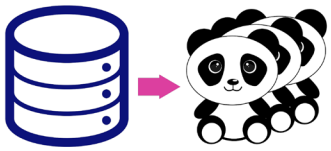
```
import matplotlib.pyplot as plt
fig, (ax1,ax2) = plt.subplots(2,1, sharex=True, figsize=(15, 10))

ax1.plot( df_hsi.index, df_hsi['Close'], df_hsi['MA10'])
ax1.set_ylabel('Share Price')
ax1.legend(['Close', 'MA10'])

ax2.bar( df_hsi.index, df_hsi['Volume'] )
ax2.set_ylabel('Daily volume')

fig.suptitle('Hang Seng Index')
```



與資料庫交互 Interacting with Database

在實際場景中，數據存儲在基於 SQL 的關係資料庫中，例如 SQL Server、PostgreSQL Server 和 MySQL。從基於 SQL 的伺服器載入資料非常簡單。我們以簡單的本地 sqlite3 資料庫為例。

首先，我們創建一個示例表。



```
In [71]: import sqlite3
```

```
In [79]: query = """
         CREATE TABLE sales(
         firstname VARCHAR(20),
         lastname VARCHAR(20),
         age INTEGER,
         sales REAL
         );
         """
```

連接資料庫並執行查詢

```
In [80]: con = sqlite3.connect('salesdata.sqlite')
         con.execute(query)   # you can create a table once
Out[80]: <sqlite3.Cursor at 0x11e70df10>
```

```
In [81]: con.commit()
```

```
In [82]: data = [('David', 'Chan', 35, 68000.50),
                 ('Edgar', 'Doe', 45, 79000.60),
                 ('Fion', 'Elle', 30, 88650.30)]
         stmt = "INSERT INTO sales VALUES(?, ?, ?, ?)"
         con.executemany(stmt, data)
Out[82]: <sqlite3.Cursor at 0x11e70db20>
```

```
In [83]: con.commit()
```

拉取數據並導入到 Pandas DF

In [84]:
```
cursor = con.execute('select * from sales')
rows = cursor.fetchall()
rows
```

Out[84]:
```
[('David', 'Chan', 35, 68000.5),
 ('Edgar', 'Doe', 45, 79000.6),
 ('Fion', 'Elle', 30, 88650.3)]
```

In [85]:
```
cursor.description
```

Out[85]:
```
(('firstname', None, None, None, None, None, None),
 ('lastname', None, None, None, None, None, None),
 ('age', None, None, None, None, None, None),
 ('sales', None, None, None, None, None, None))
```

In [88]:
```
df_sales = pd.DataFrame(rows,
        columns=[x[0] for x in cursor.description])
df_sales
```

Out[88]:

	firstname	lastname	age	sales
0	David	Chan	35	68000.5
1	Edgar	Doe	45	79000.6
2	Fion	Elle	30	88650.3

讀寫 HDF5

分層數據格式 Hierarchical Data Format（**HDF**）是一種檔格式（**HDF4**、**HDF5**），旨在存儲和組織大量數據。HDF5 格式旨在解決 HDF4 庫的一些局限性，並滿足現代系統和應用程式的當前和預期要求。

銀行數據可能是其中的一部分，一個檔的大小可能超過 10GB。

- 寫

  df.to_hdf（“data.h5”，“df”）

- 從 H5 讀取

  pd.read_hdf（“data.h5”，“df”）

DF.groupby

除了處理來自不同 DF 的數據外，通常還在同一 DF 中分析類別中的數據。

GroupBy 物件由 groupby 調用返回：df.groupby（），series.groupby（）

g = df.groupby（ column ） 表的分組表示形式

- 可以遍歷組
- 可以使用 agg（） 函數聚合每個組內的值以獲得匯總統計數據， ['mean'， 'max'， 'count'， 'sum']

Team	Score
A	8.1
A	8.3
A	9.2
B	6.5
B	7.1
B	8.6
B	7.3

max

A	9.2
B	8.6

Assume that we have a DF on the right and want to compare Company A and B's top Mile-Per-Gallon.

	Company	Model	Year	Transmission	EngineSize	MPG
0	A	A1	2019	Manual	1.4	55.4
1	A	A2	2020	Automatic	2.0	67.3
2	A	A3	2021	Automatic	1.4	58.9
3	B	B1	2018	Manual	1.5	52.3
4	B	B2	2019	Automatic	2.0	64.2
5	B	B3	2020	Automatic	1.5	68.9
6	B	B4	2021	Manual	1.5	83.1

```
# groupby columns on Col1 and estimate the
# maximum value of column Col2 for each group
# df.groupby([Col1])[Col2].max()
df.groupby(["Company"])["MPG"].max()
```

```
Company
A    67.3
B    83.1
Name: MPG, dtype: float64
```

df.groupby will NOT re-arrange the original sequence of rows. Whereas Pivoting will.

您可以將 'mean'、'max'、'count'、'sum' 等傳遞給 agg（ ） 函數。

在指定軸上使用一個或多個操作進行聚合。

```
# alternatively, you can pass 'max' to the agg() function
df.groupby(["Company"])["MPG"].agg('max')
```

```
Company
A    67.3
B    83.1
Name: MPG, dtype: float64
```

```
df.groupby(["Company"])["MPG"].agg(['mean','count','std'])
```

Company	mean	count	std
A	60.533333	3	6.115826
B	67.125000	4	12.736921

添加新的數據行

使用 pd.concat（[df， new_df]， ignore_index=True） 添加新的數據行。

許多網站使用 df.append（new_df），但很快就會被停用 depreciated。

```
df = pd.concat([df, pd.DataFrame({
                                    "Company":["C"],
                                    "Model":["C1"],
                                    "Year":[2023],
                                    "Transmission":["Automatic"],
                                    "EngineSize":[1.8] ,
                                    "MPG":[70],
                                    })
                ], ignore_index=True)
```

	Company	Model	Year	Transmission	EngineSize	MPG
0	A	A1	2019	Manual	1.4	55.4
1	A	A2	2020	Automatic	2.0	67.3
2	A	A3	2021	Automatic	1.4	58.9
3	B	B1	2018	Manual	1.5	52.3
4	B	B2	2019	Automatic	2.0	64.2
5	B	B3	2020	Automatic	1.5	68.9
6	B	B4	2021	Manual	1.5	83.1
7	C	C1	2023	Automatic	1.8	70.0

```
df = df.append(pd.DataFrame({
    "Company":["C"],
    "Model":["C1"],
    "Year":[2023],
    "Transmission":["Automatic"],
    "EngineSize":[1.8] ,
    "MPG":[70],
}))
```

```
/var/folders/cq/_ztw83fd0vsg5vkyf8hn0zd40000gn/T/ipykernel_52155/842031746.py:1: FutureWarning: The frame.append meth
od is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df = df.append(pd.DataFrame({
```

訪問 Access DF

存取行的最快方法是按其索引。 **df.loc[ ]** 很方便; **df.iloc[ ]** 速度很快。

	loc	iloc
	Label-based (Location)	***Integer position-based (iLocation)***
A Value	A single label `e.g.) df.loc['x1']`	A single integer `e.g.) df.iloc[0]`
A List or Array	A list of labels `e.g.) df.loc[['x1', 'x2']]`	A list of integers `e.g.) df.iloc[[0, 1]]`
A Slice Object	A slice object with labels 'x1' and 'x2' are included `e.g.) df.loc['x1':'x2']`	A slice object with inetgers i is included, j is not included `e.g.) df.iloc[i:j]`

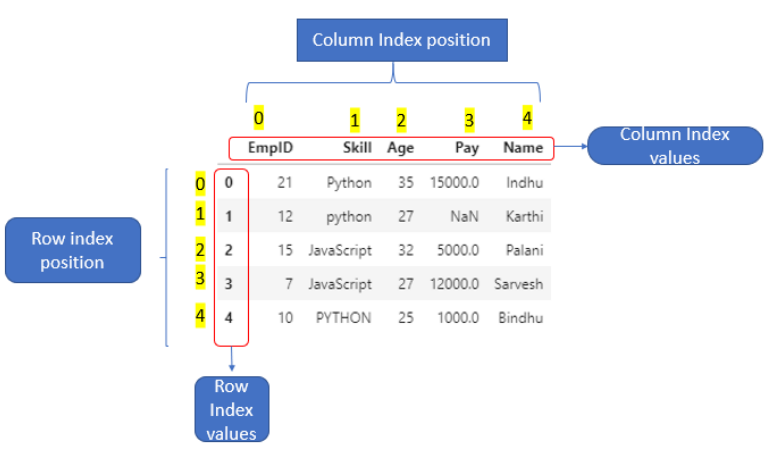
Conditions or Boolean Array	Conditional that returns a boolean Series
	`e.g.) df.loc[df['x1'] > 10]`
	`df.loc[[True, False, False]]`
		A boolean array
		`e.g.) df.iloc[[True, False, False]]`

A Callable Function	A callable function `e.g.) df.loc[lambda df: df['x1'] > 10]`	A callable function `e.g.) df.iloc[lambda x: x.index %2 == 0]`

DataFrame index 索引

在 Pandas DF 中，索引定義了數據的訪問速度。將唯一且相同的列標記為索引，或將其保留為預設的增量整數。典型的索引可以是 datetime 或 id。有 4 種方法可以處理 DF 的索引。

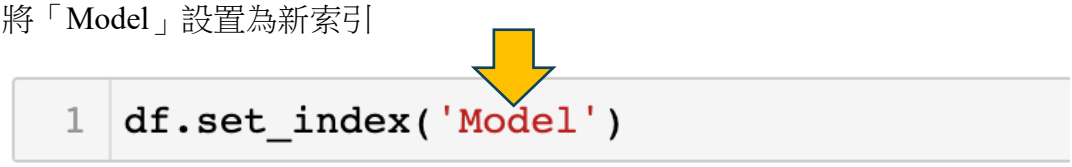
DataFrame.set_index
使用現有列設置 DataFrame 索引。
DataFrame.reset_index
與 set_index 相反。
DataFrame.reindex
更改為新索引或擴展索引。
DataFrame.reindex_like
更改為與其他 DataFrame 相同的索引。



DF.set_index

	Company	Model	Year	Transmission	EngineSize	MPG
0	A	A1	2019	Manual	1.4	55.4
1	A	A2	2020	Automatic	2.0	67.3
2	A	A3	2021	Automatic	1.4	58.9
3	B	B1	2018	Manual	1.5	52.3
4	B	B2	2019	Automatic	2.0	64.2
5	B	B3	2020	Automatic	1.5	68.9
6	B	B4	2021	Manual	1.5	83.1
7	C	C1	2023	Automatic	1.8	70.0

將「Model」設置為新索引



```
1  df.set_index('Model')
```

Model	Company	Year	Transmission	EngineSize	MPG
A1	A	2019	Manual	1.4	55.4
A2	A	2020	Automatic	2.0	67.3
A3	A	2021	Automatic	1.4	58.9
B1	B	2018	Manual	1.5	52.3
B2	B	2019	Automatic	2.0	64.2
B3	B	2020	Automatic	1.5	68.9
B4	B	2021	Manual	1.5	83.1
C1	C	2023	Automatic	1.8	70.0

Df.sort_values – 按行順序排序 sorting rows order

- 按任一軸上的值排序。

```
df.sort_values(by="Year", ascending=True)
```

	Company	Model	Year	Transmission	EngineSize	MPG
3	B	B1	2018	Manual	1.5	52.3
0	A	A1	2019	Manual	1.4	55.4
4	B	B2	2019	Automatic	2.0	64.2
1	A	A2	2020	Automatic	2.0	67.3
5	B	B3	2020	Automatic	1.5	68.9
2	A	A3	2021	Automatic	1.4	58.9
6	B	B4	2021	Manual	1.5	83.1
7	C	C1	2023	Automatic	1.8	70.0

按多個系列對行進行排序 Sorting rows by multiple series

- 請注意，整數索引也已重新排序。我們可以重置索引值。

```
df1 = df.sort_values(by=["Company","Year"], ascending=[True,False])
df1
```

	Company	Model	Year	Transmission	EngineSize	MPG
2	A	A3	2021	Automatic	1.4	58.9
1	A	A2	2020	Automatic	2.0	67.3
0	A	A1	2019	Manual	1.4	55.4
6	B	B4	2021	Manual	1.5	83.1
5	B	B3	2020	Automatic	1.5	68.9
4	B	B2	2019	Automatic	2.0	64.2
3	B	B1	2018	Manual	1.5	52.3
7	C	C1	2023	Automatic	1.8	70.0

更改列/序列順序 Alter column/series order

更改列順序相對簡單。

```
df[["MPG","EngineSize","Transmission","Company","Model","Year"]]
```

	MPG	EngineSize	Transmission	Company	Model	Year
0	55.4	1.4	Manual	A	A1	2019
1	67.3	2.0	Automatic	A	A2	2020
2	58.9	1.4	Automatic	A	A3	2021
3	52.3	1.5	Manual	B	B1	2018
4	64.2	2.0	Automatic	B	B2	2019
5	68.9	1.5	Automatic	B	B3	2020
6	83.1	1.5	Manual	B	B4	2021
7	70.0	1.8	Automatic	C	C1	2023
8	70.0	1.8	Automatic	C	C1	2023

使用 .iloc[ ] 是最快的方法

```
df.iloc[:,[5,4,3,0,1,2]]
```

	MPG	EngineSize	Transmission	Company	Model	Year
0	55.4	1.4	Manual	A	A1	2019
1	67.3	2.0	Automatic	A	A2	2020
2	58.9	1.4	Automatic	A	A3	2021
3	52.3	1.5	Manual	B	B1	2018
4	64.2	2.0	Automatic	B	B2	2019
5	68.9	1.5	Automatic	B	B3	2020
6	83.1	1.5	Manual	B	B4	2021
7	70.0	1.8	Automatic	C	C1	2023
8	70.0	1.8	Automatic	C	C1	2023

series.unique

Series.unique（）
返回 Series 物件的唯一值。ndarray 或 ExtensionArray。作為 NumPy 陣列返回的唯一值。
唯一值按出現順序返回。基於 hash table-based 的唯一，因此不排序。
Index.unique（） 提供相同的服務

```
df["EngineSize"].unique()
```

```
array([1.4, 2. , 1.5, 1.8])
```

```
df["Company"].unique()
```

```
array(['A', 'B', 'C'], dtype=object)
```

```
df["Year"].unique()
```

```
array([2019, 2020, 2021, 2018, 2023])
```

series.nunique

Series.**nunique**（*dropna=True*）
返回物件中唯一元素的數目。
默認情況下排除 NA 值。
dropna=True 表示預設情況下 NaN 不計數。
Index.nunique（） 提供相同的服務

```
df["EngineSize"].nunique()
```

```
4
```

```
df["Year"].nunique()
```

```
5
```

```
df["Transmission"].nunique()
```

```
2
```

Pd.read_csv 和 pd.read_json

從 json 讀取文件很簡單，就像 csv 一樣。

JSON（JavaScript Object Notation 物件表示法）是一種羽量級的數據交換格式。人類很容易閱讀和寫作。

API 代表應用程式程式設計介面。意思是，應用程式相互通信的協定。就像我們從 HKO API 中提取的數據一樣，有許多可用的開放數據，並支援 json 格式。

這次我們來導入香港房地產銷售數據。

Domestic Sales (from 2002)

RATING AND VALUATION DEPARTMENT | Housing | XLS | API Available

LAST UPDATED ON: **2023-03-06**
UPDATE FREQUENCY: **MONTHLY, EXCEPT FOR STOCK, VACANCY AND TAKE-UP TO BE UPDATED ANNUALLY**

Domestic Sales (from 2002)

- 首先從 data.gov.hk 找到 json 鏈接，然後 pd.read_json

```
link = "https://api.data.gov.hk/v2/filter?q=%7B%
data = pd.read_json(link)
```

- 使用 df.rename（columns={old：new}）更改列名
- “年”和“月”列應合併為相同的索引。

```
# change the column name
data.rename(columns = {'年':'Year','月':'Month','數目 No.':'Case',
    '總值  (百萬元) Consideration  ($ million)':'Amount_mil'}, inplace = True)
```

```
data
```

	Year	Month	Case	Amount_mil
0	2002	5	7325	16940
1	2002	6	7195	16129
2	2002	7	4961	10272
3	2002	8	4881	9867
4	2002	9	6278	12113
...	...	...	...	...
193	2018	6	6713	68023
194	2018	7	6091	65237
195	2018	8	4822	46765
196	2018	9	3500	37083
197	2018	10	4243	38571

198 rows × 4 columns

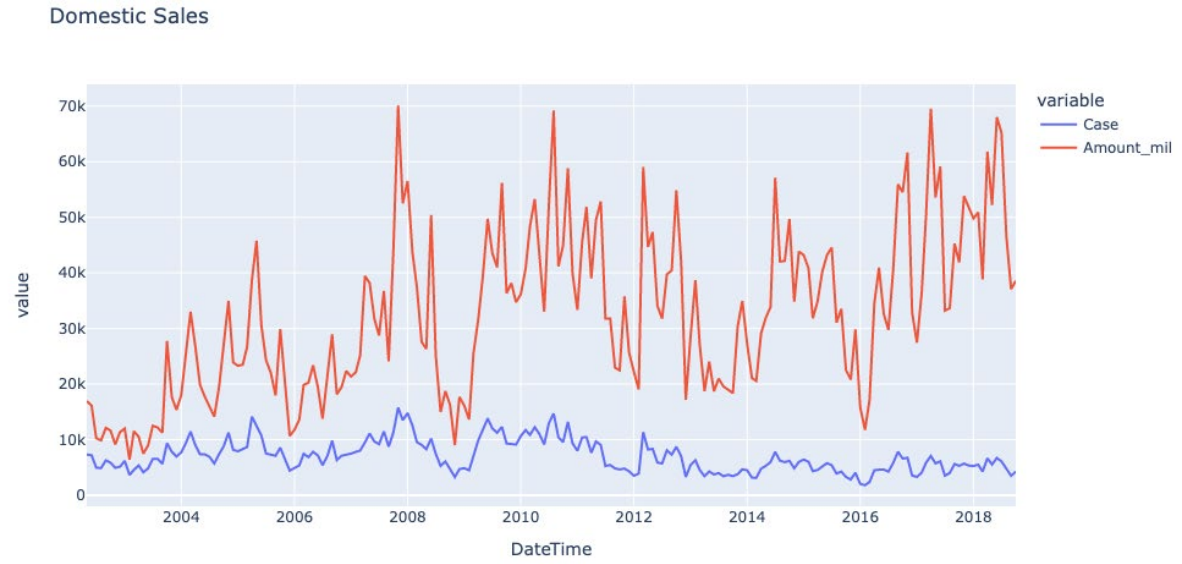
```python
# join Year and Month as new column
data["DateTime"] = data["Year"].astype(str) + "-" + data["Month"].astype(str)
# drop Year and Month
data = data.drop(['Year','Month'], axis=1)
# set DateTime as index
data = data.set_index('DateTime')
data
```

DateTime	Case	Amount_mil
2002-5	7325	16940
2002-6	7195	16129
2002-7	4961	10272
2002-8	4881	9867
2002-9	6278	12113

- Join Year and Month column as DateTime
- Drop Year and Month column
- Set index as DateTime

讓我們用 Plotly 繪製一個快速圖表。我們將很快詳細討論 Plotly 的用法。

```python
import plotly.express as px
fig = px.line(data, x=data.index, y=["Case","Amount_mil"], title='Domestic Sales')
fig.show()
```



Df.corr

df.corr（）， series.corr（）

計算列的成對相關性，不包括 NA/null 值。

```
1 data.corr()
```

	Case	Amount_mil
Case	1.000000	0.484761
Amount_mil	0.484761	1.000000

```
1 data["Case"].corr(data["Amount_mil"])
```

```
0.48476064701593535
```

注意：係數的值始終介於 −1 和 1 之間。

- -1 表示完全負線性相關。
- +1 表示完全正線性相關。
- 0 表示變數之間沒有線性依賴關係。

Df.T

DateTime	Case	Amount_mil
2002-5	7325	16940
2002-6	7195	16129
2002-7	4961	10272
2002-8	4881	9867
2002-9	6278	12113
...	...	...
2018-6	6713	68023
2018-7	6091	65237
2018-8	4822	46765
2018-9	3500	37083
2018-10	4243	38571

198 rows × 2 columns

df.T or df.transpose

- DataFrame 的轉置。

```
1 data.T
```

DateTime	2002-5	2002-6	2002-7	2002-8	2002-9	2002-10	2002-11	2002-12	2003-1	2003-2	...
Case	7325	7195	4961	4881	6278	5863	4941	5129	6187	3649	...
Amount_mil	16940	16129	10272	9867	12113	11658	9122	11352	11997	6405	...

2 rows × 198 columns

Df.to_numpy

df.to_numpy，series.to_numpy

將 DataFrame 轉換為 NumPy 陣列。默認情況下，返回陣列的 dtype 將是 DataFrame 中所有類型的通用 NumPy dtype。

```
1 data["Case"].to_numpy()
```

```
array([ 7325,  7195,  4961,  4881,  6278,  5863,  4941,  5129,  6187,
        3649,  4550,  5373,  4130,  4833,  6525,  6559,  5632,  9360,
        7811,  6967,  7726,  9449, 11449,  8994,  7380,  7362,  6911,
        5716,  7385,  8811, 11281,  8166,  7909,  8260,  8673, 14124,
       12463, 10750,  7497,  7298,  7100,  8554,  6308,  4426,  4899,
        5369,  7456,  6849,  7812,  7150,  5398,  7032,  9811,  6335,
```

```
1 data.to_numpy()
```

```
array([[ 7325, 16940],
       [ 7195, 16129],
       [ 4961, 10272],
       [ 4881,  9867],
       [ 6278, 12113],
       [ 5863, 11658],
       [ 4941,  9122],
       [ 5129, 11352],
       [ 6187, 11997],
       [ 3649,  6405],
       [ 4550, 11520],
```

時間序列 Time Series

時間序列是在連續時間獲得的量的一系列值，它們之間的間隔通常相等。在數學中，**時間序列**是按時間順序索引（或列出或繪製）的一系列數據點。最常見的是，時間序列是在連續等距的時間點上拍攝的序列。

幾個月的國內銷售物業數據、庫存數據、多年來的天氣數據、即時交通流量數據，都是很好的例子。

要處理時間序列分析，首先要檢查的是索引。它必須設置為日期時間索引。

我們以雅虎財經的 csv 格式的美國財政部數據為例。
Df.index 顯示索引為範圍整數。


```
1 tnx = pd.read_csv("^TNX.csv")
2 tnx.tail(5)
```

	Date	Open	High	Low	Close	Adj Close	Volume
7057	2022-12-26	NaN	NaN	NaN	NaN	NaN	NaN
7058	2022-12-27	3.787	3.862	3.787	3.860	3.860	0.0
7059	2022-12-28	3.818	3.890	3.815	3.887	3.887	0.0
7060	2022-12-29	3.868	3.886	3.818	3.835	3.835	0.0
7061	2022-12-30	3.869	3.905	3.831	3.879	3.879	0.0

```
1 tnx.index
```

```
RangeIndex(start=0, stop=7062, step=1)
```

df.info（）顯示所有系列 dtypes 和資訊

```
1 tnx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7062 entries, 0 to 7061
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       7062 non-null   object
 1   Open       5780 non-null   float64
 2   High       5780 non-null   float64
 3   Low        5780 non-null   float64
 4   Close      5780 non-null   float64
 5   Adj Close  5780 non-null   float64
 6   Volume     5780 non-null   float64
dtypes: float64(6), object(1)
memory usage: 386.3+ KB
```

pd.to_datetime

將 series 轉換為 datetime。 此函數將標量、類似數位、 Series 或 DataFrame/dict 轉換為 pandas datetime 物件。

```
tnx['Date'] = pd.to_datetime(tnx['Date'],format="%Y-%m-%d")
tnx.Date
```

```
0       2000-01-03
1       2000-01-04
2       2000-01-05
3       2000-01-06
4       2000-01-07
           ...
7057    2022-12-26
7058    2022-12-27
7059    2022-12-28
7060    2022-12-29
7061    2022-12-30
Name: Date, Length: 7062, dtype: datetime64[ns]
```

設置日期時間索引

```
tnx = tnx.set_index('Date')
tnx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 7062 entries, 2000-01-03 to 2022-12-30
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       5780 non-null   float64
 1   High       5780 non-null   float64
 2   Low        5780 non-null   float64
 3   Close      5780 non-null   float64
 4   Adj Close  5780 non-null   float64
 5   Volume     5780 non-null   float64
dtypes: float64(6)
memory usage: 386.2 KB
```

```
tnx.tail(3)
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-12-28	3.818	3.890	3.815	3.887	3.887	0.0
2022-12-29	3.868	3.886	3.818	3.835	3.835	0.0
2022-12-30	3.869	3.905	3.831	3.879	3.879	0.0

對日期時間的列進行分組 Grouping columns for datetime

在某些數據集中，年、月、日列在不同的列中是分開的。您可以將它們分組為 pd.to_datetime（df_dt[['年'，'月'，'日']]）

```
df_dt = pd.DataFrame({'year': [2023, 2023],
                      'month': [1, 1],
                      'day': [1, 2],
                      'sales': [3750, 3900]})
df_dt
```

	year	month	day	sales
0	2023	1	1	3750
1	2023	1	2	3900

```
df_dt['date'] = pd.to_datetime(df_dt[['year','month','day']])
df_dt = df_dt.drop(['year','month','day'], axis=1)
df_dt = df_dt.set_index('date')
df_dt
```

	sales
date	
2023-01-01	3750
2023-01-02	3900

Unix epoch time 紀元時間

Unix is enterprise server OS, the time format is a series of number. It can convert to Pandas datetime as well.

```
pd.to_datetime(1681038343, unit='s')
```

```
Timestamp('2023-04-09 11:05:43')
```

```
pd.to_datetime(1681038343433502912, unit='ns')
```

```
Timestamp('2023-04-09 11:05:43.433502912')
```

Unix Epoch Clock

Unix time across midnight into 17 September 2004 (no leap second)

TAI (17 September 2004)	UTC (16 to 17 September 2004)	Unix time
2004-09-17T00:00:30.75	2004-09-16T23:59:58.75	1 095 379 198.75
2004-09-17T00:00:31.00	2004-09-16T23:59:59.00	1 095 379 199.00
2004-09-17T00:00:31.25	2004-09-16T23:59:59.25	1 095 379 199.25

Categorical Data Type 分類數據類型

Pandas 支援分類數據類型，類似於 R 語言。

分類是對應於統計中的分類變數的 pandas 數據類型。分類變數 具有有限且通常固定的可能值（類別;R 中的級別）的數量。例如性別、社會階層、血型、國家隸屬關係、觀察時間或李克特量表的評級。

```
1 animal = pd.Series(["Bird","Cat", "Dog","Elephant"], dtype="category")
```

```
1 animal.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 4 entries, 0 to 3
Series name: None
Non-Null Count  Dtype
--------------  -----
4 non-null      category
dtypes: category(1)
memory usage: 336.0 bytes
```

有時我們需要在分類數據中對標量數據進行分組。

例如，我們有一個評分表，需要將它們分為幾個類別。

```
1 import numpy as np
2 df_exam = pd.DataFrame({'Score': np.random.randint(1, 101, size=40)})
```

```
1 df_exam.head(6)
```

	Score
0	6
1	8
2	41
3	60
4	93
5	35

np.where

使用 np.where 反覆運算條件語句。嵌套(nested)的 np.where 可以應用。

```
df_exam['Result'] = pd.Series(np.where(df_exam['Score']>70, 'Distiction',
                        np.where(df_exam['Score']>50, 'Pass', 'Fail')
                    )).astype('category')
```

```
df_exam.sample(5)
```

	Score	Result
31	51	Pass
24	50	Fail
6	18	Fail
36	84	Distiction
39	59	Pass

處理類別數據 category data 比字串變數快得多，因為 Pandas 將其視為單個專案。

```
df_exam.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype   
---  ------  --------------  -----   
 0   Score   40 non-null     int64   
 1   Result  40 non-null     category
dtypes: category(1), int64(1)
memory usage: 620.0 bytes
```

Pandas rolling window

Pandas **dataframe.rolling（）** 函數提供了滾動窗口計算的功能。滾動窗口計算的概念主要用於信號處理和時間序列數據。

語法：

數據幀。**rolling**（*window*， *min_periods=None*， *center=False*， *win_type=None*， *on=None*， *axis=0*， *closed=None*， *step=None*， *method='single'*）
通過 yfinance 準備恒生指數的時間序列數據。

從 2020-01-01 到 2023-01-01

```
import numpy as np
import pandas as pd
import yfinance as yf
```

```
hsi = yf.download("^HSI", start="2020-01-01", end="2023-01-01")
hsi.head(3)
```

```
[**********************100%**********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2020-01-02	28249.369141	28543.519531	28245.970703	28543.519531	28543.519531	1262732800
2020-01-03	28828.359375	28883.300781	28428.169922	28451.500000	28451.500000	1797904800
2020-01-06	28326.500000	28367.869141	28054.289062	28226.189453	28226.189453	1793426600

移動平均線 Moving Average（MA）- 滾動平均值 rolling mean

準備新的列“MA10”，這是“調整后收盘價”的移動平均線 10 天平均值

```
hsi['MA10'] = hsi['Adj Close'].rolling(10).mean()
hsi.tail(5)
```

Date	Open	High	Low	Close	Adj Close	Volume	MA30	MA10
2022-12-22	19537.449219	19735.000000	19475.679688	19679.220703	19679.220703	1939795100	19474.073047	19474.073047
2022-12-23	19382.230469	19686.769531	19380.470703	19593.060547	19593.060547	1363741800	19443.292187	19443.292187
2022-12-28	19787.939453	20099.769531	19787.939453	19898.910156	19898.910156	2823780600	19486.820117	19486.820117
2022-12-29	19648.400391	19764.519531	19539.839844	19741.140625	19741.140625	2902362900	19501.314258	19501.314258
2022-12-30	20030.849609	20073.919922	19781.410156	19781.410156	19781.410156	1747706800	19512.110352	19512.110352

指數加權移動平均線 Exponentially Weighted Moving Average（EMA）

在新列「EMA10」中準備「調整平倉價」的 10 天 EMA 數據

語法：

Series.ewm（com=None， span=None， halflife=None， alpha=None， min_periods=0， adjust=True， ignore_na=False， axis=0， times=None， method='single'）

期初調整係數衰減

$$y_t = \frac{x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^t x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots + (1-\alpha)^t}$$

```
hsi['EMA10'] = hsi['Adj Close'].ewm(span=10).mean()
hsi.tail(5)
```

Date	Open	High	Low	Close	Adj Close	Volume	MA30	MA10	EMA10
2022-12-22	19537.449219	19735.000000	19475.679688	19679.220703	19679.220703	1939795100	19474.073047	19474.073047	19329.083741
2022-12-23	19382.230469	19686.769531	19380.470703	19593.060547	19593.060547	1363741800	19443.292187	19443.292187	19377.079524
2022-12-28	19787.939453	20099.769531	19787.939453	19898.910156	19898.910156	2823780600	19486.820117	19486.820117	19471.957821
2022-12-29	19648.400391	19764.519531	19539.839844	19741.140625	19741.140625	2902362900	19501.314258	19501.314258	19520.900149
2022-12-30	20030.849609	20073.919922	19781.410156	19781.410156	19781.410156	1747706800	19512.110352	19512.110352	19568.265605

獲得特定時期 MA/EMA

Use slicing series and get the last row of data

```
hsi['Adj Close'][-10:].rolling(10).mean()[-1]
```

```
19512.1103515625
```

```
hsi['Adj Close'][-10:].ewm(span=10).mean()[-1]
```

```
19618.33963667069
```

捲動標準差 Rolling Standard Deviation （STD）

使用標準差衡量波動性和風險。讓使用 250 天性病

```
hsi['250Std'] = hsi['Adj Close'].rolling(250).std()
hsi.tail(5)
```

Date	Adj Close	MA30	MA10	EMA10	Kurt50	Skew50	250Max	250Min	250Std
2022-12-22	19679.220703	19474.073047	19474.073047	19329.083741	-1.232598	-0.145976	24965.550781	14687.019531	2365.320875
2022-12-23	19593.060547	19443.292187	19443.292187	19377.079524	-1.244126	-0.208008	24965.550781	14687.019531	2360.480438
2022-12-28	19898.910156	19486.820117	19486.820117	19471.957821	-1.239929	-0.258532	24965.550781	14687.019531	2354.616331
2022-12-29	19741.140625	19501.314258	19501.314258	19520.900149	-1.247615	-0.310313	24965.550781	14687.019531	2348.397469
2022-12-30	19781.410156	19512.110352	19512.110352	19568.265605	-1.217817	-0.375461	24965.550781	14687.019531	2341.891306

滯後 1 和滯後 2 週期的自相關 Autocorrelation of lag 1 and lag 2 period

自相關是指序列與其前 n 項的相關性。通常使用滯後 1 或滯後 2。

分別使用滯後 1 天和 2 天找到 HSI 自相關。

```
df_all['HSI'].autocorr(lag=1)
```

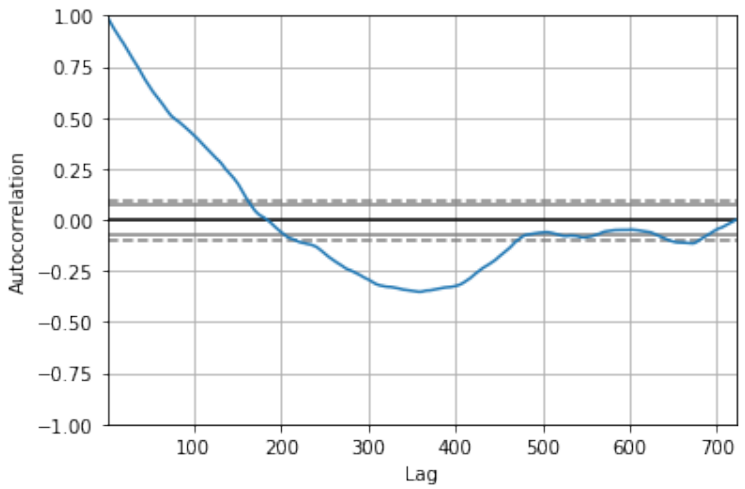
```
0.9941400668946364
```

```
df_all['HSI'].autocorr(lag=2)
```

```
0.9885254205676482
```

```
pd.plotting.autocorrelation_plot(df_all['HSI'])
```

```
<AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>
```



對於機器學習方法，分析師可能會使用AR、ARMA和ARIMA

Series.where（）和 Series.mask（）

```
p = pd.Series([1,2,3,4,5])
p
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
# Replace values where the condition is False
p.where(p<=3, 'where p<=3 is false')
```

```
0                        1
1                        2
2                        3
3    where p<=3 is false
4    where p<=3 is false
dtype: object
```

```
# Replace values where the condition is True
p.mask(p<=3, 'where p<=3 is true')
```

```
0    where p<=3 is true
1    where p<=3 is true
2    where p<=3 is true
3                     4
4                     5
dtype: object
```

本章小結

我們已經學會了所有基本的pandas df工作。是時候研究一些真實的數據和專案了。你嘗試的越多，你就越成功。不要害怕bugs和錯誤，高級程式師也每天都在經歷它們。由於 Pandas、NumPy、SciPy 和 Python 正在快速發展，一些在線資訊可能不會更新。官方文件始終值得信賴。

引用

官方網站：

- https://pandas.pydata.org/docs/

參考教材：

- Python for Data Analysis, 2nd edition, Wes McKinney, O'Reilly
- Python Data Science Handbook, Jake VanderPlas, O'Reilly
- Community Backup: https://pandas.pydata.org/docs/getting_started/tutorials.html