

MSC IN FINANCE

PRE-TERM COURSE WEEK3:

Data Analysis with Pandas

Tutor: Yuan Lu
CUHK Department of Finance
yuan.lu@link.cuhk.edu.hk

July 25, 2023

Agenda:

1. Pandas

- Data Structures
- Import and Export Data
- Methods
- Datetime
- Indexing
- Combine DataFrame
- Handling Missing Value
- GroupBy
- Lambda and Apply

2. Matplotlib

Why Pandas?

- Pandas is the most widely used python library in data science. The name is derived from “panel data”, an econometrics term used to describe datasets that include observations for the different individuals over multiple time periods. It is a high-performance tool to extract, clean, transform and analyze data.
- [https://en.wikipedia.org/wiki/Pandas_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))



Why Pandas? (Cont.) *Data analysis*

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
 - Ordered and unordered (not necessarily fixed-frequency) time series data.
 - Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
 - Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure
-
- - https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html

Why Pandas? (Cont.) *Connection with NumPy*

- NumPy provides essential features for working with clean, well-organized data. However, in real world we rarely come across such data. If NumPy lacked the flexibility of lists, Pandas core data structure called “DataFrame” provides just that.
- Built on top of the NumPy package, Pandas provides an efficient implementation of multidimensional arrays with row and column labels and allows heterogeneous data types as well as missing data.
- Pandas also provides efficient solution to those instances where element-wise broadcasting is not applicable such as grouping, pivots, etc. For brief summary on Pandas package, type “pd?” in a jupyter cell.

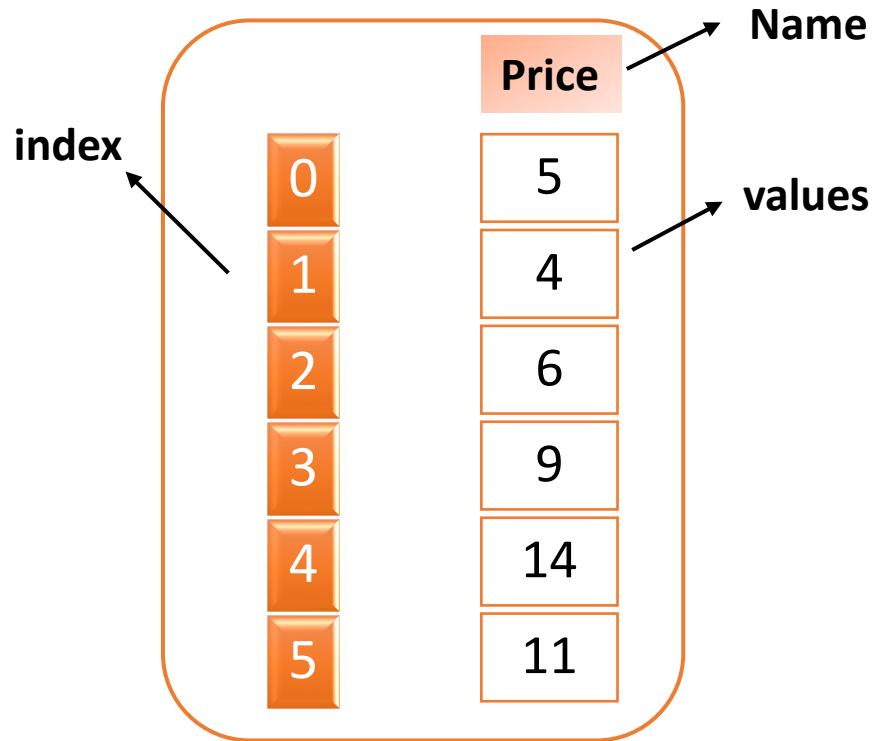


CUHK Business School
The Chinese University of Hong Kong

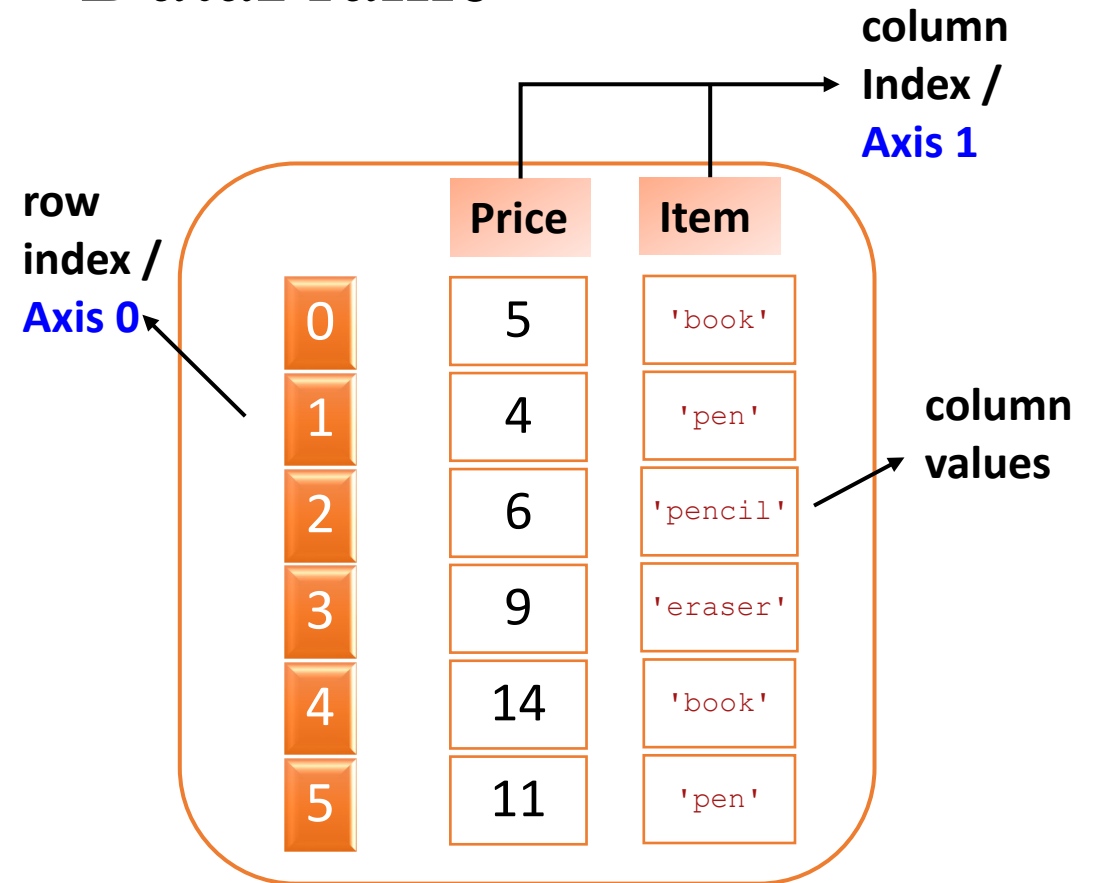
DATA STRUCTURES

Data Structures

Series



DataFrame



IMPORT AND EXPORT DATA

Read CSV

1. Download Historical Data of S&P 500 Index:

<https://finance.yahoo.com/quote/%5ESPX/history/>

S&P 500 INDEX (^SPX)

Chicago Options - Chicago Options Delayed Price. Currency in USD

☆ Follow

4,582.23 +44.82 (+0.99%)

At close: July 28 04:59PM EDT

Summary Chart **Historical Data** Options Components

Time Period: Jul 30, 2022 - Jul 30, 2023 ▾

Show: Historical Prices ▾

Frequency: Daily ▾

Apply

Currency in USD

Download

Date	Open	High	Low	Close*	Adj Close**	Volume
Jul 28, 2023	4,565.75	4,590.16	4,564.01	4,582.23	4,582.23	3,981,010,000
Jul 27, 2023	4,598.26	4,607.07	4,528.56	4,537.41	4,537.41	4,553,210,000
Jul 26, 2023	4,558.96	4,582.47	4,547.58	4,566.75	4,566.75	3,990,290,000
Jul 25, 2023	4,555.19	4,580.62	4,552.42	4,567.46	4,567.46	3,812,470,000
Jul 24, 2023	4,543.39	4,563.41	4,541.29	4,554.64	4,554.64	3,856,250,000
Jul 21, 2023	4,550.16	4,555.00	4,535.79	4,536.34	4,536.34	3,570,190,000
Jul 20, 2023	4,554.38	4,564.74	4,527.56	4,534.87	4,534.87	3,761,770,000
Jul 19, 2023	4,563.87	4,578.43	4,557.48	4,565.72	4,565.72	4,115,670,000

2. Import Data by pd.read_csv()

```
import pandas as pd
```

```
path='./data/' # could be any path that you store your data. Please change the  
directory when running your code
```

```
SPXdaily=pd.read_csv(path+'SPXdaily.csv')  
SPXdaily
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-08-01	4112.379883	4144.950195	4096.020020	4118.629883	4118.629883	4202810000
1	2022-08-02	4104.209961	4140.470215	4079.810059	4091.189941	4091.189941	4727710000
2	2022-08-03	4107.959961	4167.660156	4107.959961	4155.169922	4155.169922	4351760000
3	2022-08-04	4154.850098	4161.290039	4135.419922	4151.939941	4151.939941	4283320000
4	2022-08-05	4115.870117	4151.580078	4107.310059	4145.189941	4145.189941	4085940000
...
245	2023-07-24	4543.390137	4563.410156	4541.290039	4554.640137	4554.640137	3856250000
246	2023-07-25	4555.189941	4580.620117	4552.419922	4567.459961	4567.459961	3812470000
247	2023-07-26	4558.959961	4582.470215	4547.580078	4566.750000	4566.750000	3990290000
248	2023-07-27	4598.259766	4607.069824	4528.560059	4537.410156	4537.410156	4553210000
249	2023-07-28	4565.750000	4590.160156	4564.009766	4582.229980	4582.229980	3981010000

250 rows × 7 columns

Read CSV (Cont.) *Path*

- If you put your dataset on certain directory such as Desktop, you need declare the path (where you store your data).

```
path='C:/Users/User/Desktop/' # where you store your data  
pd.read_csv(path+'SPXdaily.csv')
```

- Get your current working directory

```
import os  
os.getcwd()
```

- If you put your dataset on your current working directory, you don't need to add path in front of your filename.

```
pd.read_csv('SPXdaily.csv')
```

- If you put your dataset on the sub folder of your current working directory, you could declare the path by './subfolder/' for short.

```
path='./data/' # where you store your data, subfolder in your current working dictory  
pd.read_csv(path+'SPXdaily.csv')
```

Read CSV (Cont.) *Parameters*

- Documentation of pandas.read_csv

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

```
pd.read_csv(path+'SPXdaily.csv', header=0, index_col=None, usecols=['Date','Close'], nrows=10)
```

File Path +File Name

Row number(s) to use
as the column names

When header=0, the
column names are
inferred from the first
line of the file

Column(s) to use
as the row labels
of the DataFrame

Return a subset
of the columns.

Number of rows of
file to read. Useful for
reading pieces of
large files.

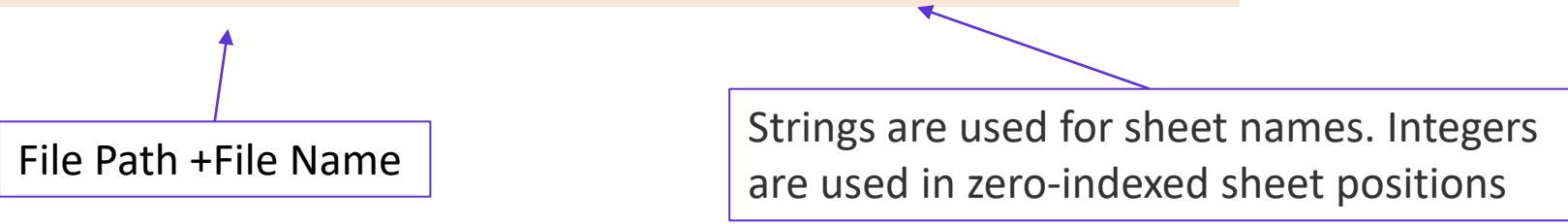
Read Excel (xlsx)

- Documentation of pandas.read_excel

https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html

```
pd.read_excel(path+'SPXdailyexcel.xlsx',sheet_name='Sheet1')
```

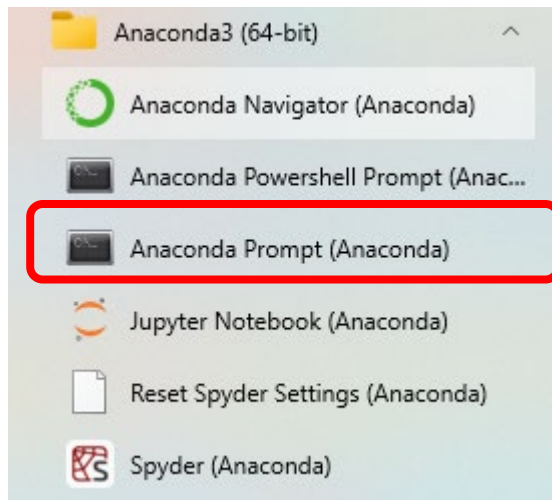
File Path +File Name



Strings are used for sheet names. Integers are used in zero-indexed sheet positions

Yfinance

- Yahoo Finance is one of the most popular sources of stock market data available for retail traders. [yfinance](#) is a popular open source library developed by [Ran Aroussi](#) as a means to access the financial data available on Yahoo Finance.
- Documentation:
<https://pypi.org/project/yfinance/>
- Install yfinance: open ananconda prompt and copy the pip install yfinance command.



```
Anaconda Prompt (Anaconda)  
(base) C:\Users\User>pip install yfinance
```

Yfinance (Cont.)

- **Get Historical Data:**

```
import yfinance as yf
```

```
yf.Ticker("APPL").history(period='max',interval='1mo',start="2022-07-30",end="2023-07-30",auto_adjust=False)
```

- **Parameters**

- **period:** As seen before, especially useful is the value “max”. The following are the valid values:
1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max.
- **interval:** Defines the size of each bar. Smaller bar sizes have more strict limitations, and only 7 days of 1-minute data can be retrieved. The following are the valid values:
1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,1mo,3mo
- **start:** Start date. The server expects a string formatted as YYYY-MM-DD.
- **end:** End date. The server expects a string formatted as YYYY-MM-DD.
- **auto_adjust:** whether to adjust prices to stock splits and dividend payments. The default value is *True*.

Web Data-Reader

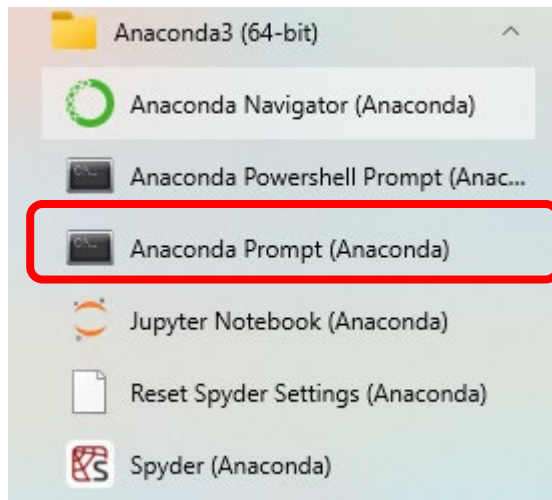
- Documentation:

https://pandas-datareader.readthedocs.io/en/latest/remote_data.html

- Functions from `pandas_datareader.data` extract data from various Internet sources into a pandas DataFrame.

Currently the following sources are supported:

- Tiingo; IEX; Alpha Vantage; Econdb; Enigma; Quandl; St.Louis FED (FRED); Kenneth French's data library; World Bank; OECD; Eurostat; Thrift Savings Plan; Nasdaq Trader symbol definitions; Stooq; MOEX; Naver Finance; Yahoo Finance
- Install Datareader: open ananconda prompt and copy the conda or pip install command.



Anaconda Prompt (Anaconda)

```
(base) C:\Users\User>pip install pandas-datareader
```

Or

Anaconda Prompt (Anaconda)

```
(base) C:\Users\User>conda install -c anaconda pandas-datareader
```

Web Data-Reader (Cont.) *Yahoo Finance*

```
from pandas_datareader import data as pdr

import yfinance as yf
yf.pdr_override() # <== that's all it takes :-)

# download dataframe
SPXdaily_pdr = pdr.get_data_yahoo("^SPX", start="2022-07-30", end="2023-07-30")
SPXdaily_pdr
```

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

	Open	High	Low	Close	Adj Close	Volume
Date						
2022-08-01	4112.379883	4144.950195	4096.020020	4118.629883	4118.629883	4202810000
2022-08-02	4104.209961	4140.470215	4079.810059	4091.189941	4091.189941	4727710000
2022-08-03	4107.959961	4167.660156	4107.959961	4155.169922	4155.169922	4351760000
2022-08-04	4154.850098	4161.290039	4135.419922	4151.939941	4151.939941	4283320000
2022-08-05	4115.870117	4151.580078	4107.310059	4145.189941	4145.189941	4085940000
...
2023-07-24	4543.390137	4563.410156	4541.290039	4554.640137	4554.640137	3856250000
2023-07-25	4555.189941	4580.620117	4552.419922	4567.459961	4567.459961	3812470000
2023-07-26	4558.959961	4582.470215	4547.580078	4566.750000	4566.750000	3990290000
2023-07-27	4598.259766	4607.069824	4528.560059	4537.410156	4537.410156	4553210000
2023-07-28	4565.750000	4590.160156	4564.009766	4582.229980	4582.229980	3981010000

250 rows × 6 columns

Note: Because of the version compatibility issue, when using data reader to access Yahoo finance data, we still need to import yfinance and use pdr_override to solve the issue.

Export to CSV

- Documentation of pandas.DataFrame.to_csv
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

```
path='./data/'  
SPXdaily.to_csv(path+'/SPXdaily_out0.csv')  
SPXdaily.to_csv(path+'/SPXdaily_out1.csv',header=True,index=False,columns=['Date','Close'])
```

SPXdaily_out0.csv

	A	B	C	D	E	F	G	H	I
1		Date	Open	High	Low	Close	Adj Close	Volume	
2	0	2022/8/1	4112.38	4144.95	4096.02	4118.63	4118.63	4.2E+09	
3	1	2022/8/2	4104.21	4140.47	4079.81	4091.19	4091.19	4.73E+09	
4	2	2022/8/3	4107.96	4167.66	4107.96	4155.17	4155.17	4.35E+09	
5	3	2022/8/4	4154.85	4161.29	4135.42	4151.94	4151.94	4.28E+09	
6	4	2022/8/5	4115.87	4151.58	4107.31	4145.19	4145.19	4.09E+09	
7	5	2022/8/8	4155.93	4186.62	4128.97	4140.06	4140.06	4.22E+09	
8	6	2022/8/9	4133.11	4137.3	4112.09	4122.47	4122.47	3.91E+09	
9	7	2022/8/10	4181.02	4211.03	4177.26	4210.24	4210.24	4.55E+09	
10	8	2022/8/11	4227.4	4257.91	4201.41	4207.27	4207.27	4.63E+09	
11	9	2022/8/12	4225.02	4280.47	4219.78	4280.15	4280.15	3.79E+09	
12	10	2022/8/15	4269.37	4301.79	4256.9	4297.14	4297.14	3.7E+09	
13	11	2022/8/16	4290.46	4325.28	4277.77	4305.2	4305.2	4.33E+09	
14	12	2022/8/17	4280.4	4302.18	4253.08	4274.04	4274.04	3.89E+09	
15	13	2022/8/18	4273.13	4292.53	4261.98	4283.74	4283.74	3.34E+09	
16	14	2022/8/19	4266.31	4266.31	4218.7	4228.48	4228.48	3.76E+09	

SPXdaily_out0.csv

	A	B	C
1	Date	Close	
2	2022/8/1	4118.63	
3	2022/8/2	4091.19	
4	2022/8/3	4155.17	
5	2022/8/4	4151.94	
6	2022/8/5	4145.19	
7	2022/8/8	4140.06	
8	2022/8/9	4122.47	
9	2022/8/10	4210.24	
10	2022/8/11	4207.27	
11	2022/8/12	4280.15	
12	2022/8/15	4297.14	
13	2022/8/16	4305.2	
14	2022/8/17	4274.04	
15	2022/8/18	4283.74	
16	2022/8/19	4228.48	

Export to Excel(xlsx)

- Documentation of pandas.DataFrame.to_excel

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

```
path='./data/'
SPXdaily_sheet1.to_excel(path+'/SPXdaily_outexcel.xlsx',sheet_name='Sheet_out1')
```

```
▼ # ExcelWriter can also be used to append to an existing Excel file:
▼ with pd.ExcelWriter(path+'/SPXdaily_outexcel.xlsx', engine='openpyxl',mode='a') as writer:
    SPXdaily_sheet2.to_excel(writer, sheet_name='Sheet_out2')
```

```
▼ # If you wish to write to more than one sheet in the workbook, it is necessary to specify an ExcelWriter object:
▼ with pd.ExcelWriter(path+'SPXdaily_outexcel1.xlsx') as writer:
    SPXdaily_sheet1.to_excel(writer, sheet_name='Sheet_out1')
    SPXdaily_sheet2.to_excel(writer, sheet_name='Sheet_out2')
```

ATTRIBUTES AND METHODS

Methods

- .info()

row
labels

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

```
RangeIndex: 250 entries, 0 to 249
```

```
Data columns (total 8 columns):
```

#	Column	Non-Null Count	Dtype
---	--------	----------------	-------

---	-----	-----	-----
-----	-------	-------	-------

0	Date	250 non-null	object
1	Open	250 non-null	float64
2	High	250 non-null	float64
3	Low	250 non-null	float64
4	Close	250 non-null	float64
5	Adj Close	250 non-null	float64
6	Volume	250 non-null	int64
7	diff	250 non-null	float64

column
names

no. of
values in
a column

data types
for each
column

```
dtypes: float64(6), int64(1), object(1)
```

```
memory usage: 15.8+ KB
```

Attributes and Methods (Cont.)

• .shape

(250, 8)

- **Axis 0**
 - Refers to no. of rows
- **Axis 1**
 - Refers to no. of columns

.ndim

2

- Refers to. of dimensions

.describe ()

- Get statistical summary such as average, standard deviation, minimum, maximum and quantile values of each numeric column

.round (2)

- Round results to different decimal places

Methods (Cont.)

Methods can be applied row-wise or column-wise by specifying the axis.

```
▼ # create a dataframe based on two series  
ab = pd.DataFrame( {'col_a':a, 'col_b':b} )  
ab |
```

	col_a	col_b
0	10	22
1	20	33
2	30	44
3	40	55

```
ab.sum(axis=0)
```

```
col_a    100  
col_b    154  
dtype: int64
```

```
ab.sum(axis=1)
```

```
0     32  
1     53  
2     74  
3     95  
dtype: int64
```

Methods (Cont.)

- “.drop” method can be used to remove rows or columns from `DataFrame`.
 - Multiple rows or columns can be removed by placing the index labels in a list.
 - “inplace = True” argument changes the original “DataFrame”.

```
SPXdaily.drop('Open', axis=1)
```

	Date	High	Low	Close	Adj Close	Volume
0	2022-08-01	4144.950195	4096.020020	4118.629883	4118.629883	4202810000
1	2022-08-02	4140.470215	4079.810059	4091.189941	4091.189941	4727710000
2	2022-08-03	4167.660156	4107.959961	4155.169922	4155.169922	4351760000
3	2022-08-04	4161.290039	4135.419922	4151.939941	4151.939941	4283320000
4	2022-08-05	4151.580078	4107.310059	4145.189941	4145.189941	4085940000
...
245	2023-07-24	4563.410156	4541.290039	4554.640137	4554.640137	3856250000
246	2023-07-25	4580.620117	4552.419922	4567.459961	4567.459961	3812470000
247	2023-07-26	4582.470215	4547.580078	4566.750000	4566.750000	3990290000
248	2023-07-27	4607.069824	4528.560059	4537.410156	4537.410156	4553210000
249	2023-07-28	4590.160156	4564.009766	4582.229980	4582.229980	3981010000

250 rows × 6 columns

```
SPXdaily.drop(['High', 'Low'], axis=1, inplace=True)  
SPXdaily
```

	Date	Open	Close	Adj Close	Volume
0	2022-08-01	4112.379883	4118.629883	4118.629883	4202810000
1	2022-08-02	4104.209961	4091.189941	4091.189941	4727710000
2	2022-08-03	4107.959961	4155.169922	4155.169922	4351760000
3	2022-08-04	4154.850098	4151.939941	4151.939941	4283320000
4	2022-08-05	4115.870117	4145.189941	4145.189941	4085940000
...
245	2023-07-24	4543.390137	4554.640137	4554.640137	3856250000
246	2023-07-25	4555.189941	4567.459961	4567.459961	3812470000
247	2023-07-26	4558.959961	4566.750000	4566.750000	3990290000
248	2023-07-27	4598.259766	4537.410156	4537.410156	4553210000
249	2023-07-28	4565.750000	4582.229980	4582.229980	3981010000

250 rows × 5 columns

Methods (Cont.) *Moving Averages*

Simple model that predict today's price as the average of the price of the last **2** days

DataFrame
(df) for
daily price
of a stock

5.49	}	$(5.49 + 5.65) / 2$	How do we do this in Pandas?
5.65			
5.87			
6.32			
6.59			
6.85			
7.56			
8.56			
9.25			
9.99			
N- days			

.rolling()

```
df['Price'].rolling(window=2).mean().shift()
```

NaN
5.57
5.76
6.10
6.45
6.72
7.20
8.06
8.90
9.62
N-1 days

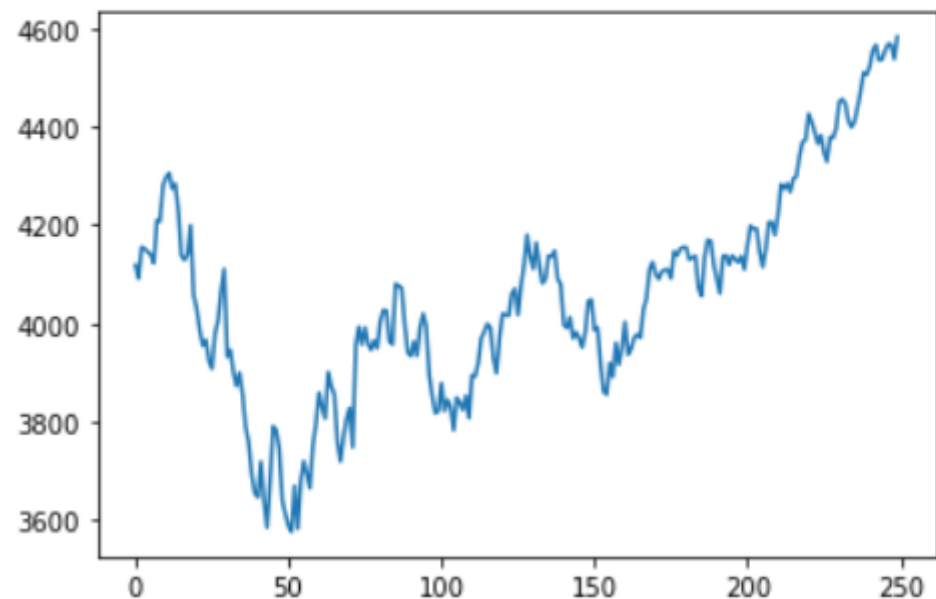
.shift()

NaN
NaN
5.57
5.76
6.10
6.45
6.72
7.20
8.06
8.90
N-1 days

Methods (Cont.) *Plot and histogram*

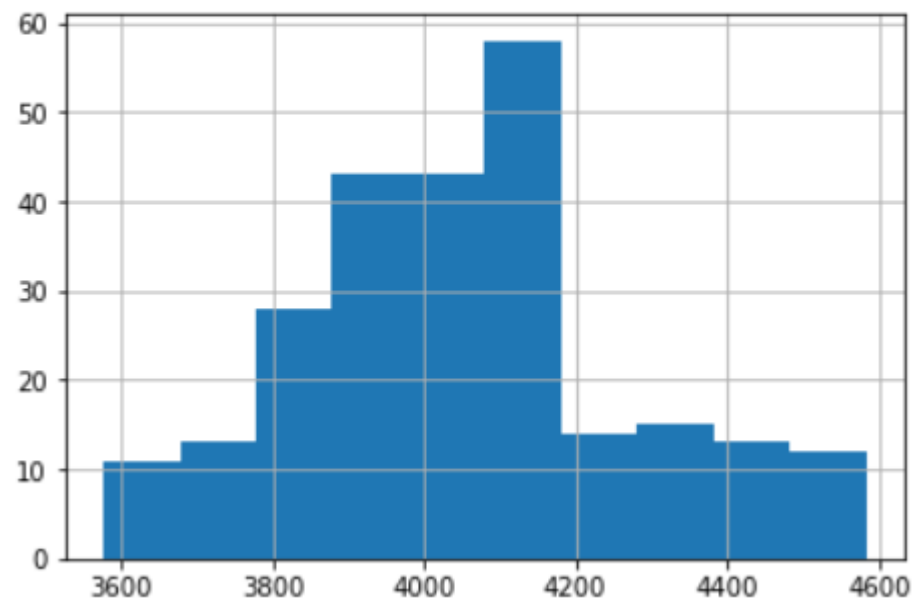
```
SPXdaily['Adj Close'].plot()
```

<AxesSubplot:>



```
SPXdaily['Adj Close'].hist()
```

<AxesSubplot:>





CUHK Business School
The Chinese University of Hong Kong

DATETIME

Datetime

- Python's “**datetime**” module is designed to easily store and manipulate date and time information.

```
import datetime as dt
```

- Example:

```
today = dt.datetime.now()
```

```
today.year
```

```
today.month
```

```
today.day
```

```
today.hour
```

```
import datetime as dt
today = dt.datetime.now()
display(today, type(today))
```

```
datetime.datetime(2023, 7, 31, 13, 18, 28, 440824)
```

```
datetime.datetime
```

- **Timestamp** data types come with numerous methods for easy extraction of date elements.

```
display(today.year,
        today.month,
        today.day,
        today.hour)
```

```
2023
```

```
7
```

```
31
```

```
13
```

Datetime (Cont.) *datetime.strptime*

- Certain elements of datetime timestamp objects can be extracted as strings by specifying the desired format in “datetime.strptime” method.
- For references on datetime format, visit - <https://strftime.org/>

```
today
```

```
datetime.datetime(2023, 7, 31, 13, 18, 28, 440824)
```

```
today.strftime('%Y-%m')
```

```
'2023-07'
```

```
today.strftime('%Y-%m-%w')
```

```
'2023-07-1'
```

```
today.strftime('%Y-%b-%d')
```

```
'2023-Jul-31'
```

```
today.strftime('%Y-%b-%d %A')
```

```
'2023-Jul-31 Monday'
```

Datetime (Cont.) *to_datetime*

- `.to_datetime()`

```
pd.to_datetime('2020-01-01 8:45am')  
→ Timestamp('2020-01-01 08:45:00')
```

A single point in time is represented as a
Timestamp in Pandas

Automatically infers a date/time format based
on the input

```
dates = pd.Series(['2019-01-01', '2019-02-01', '2019-03-01', '2019-04-01'])  
dates = pd.to_datetime(dates)  
dates
```

```
0    2019-01-01  
1    2019-02-01  
2    2019-03-01  
3    2019-04-01  
dtype: datetime64[ns]
```

Datetime (Cont.) *example: AAPL*

```
# Read AAPL from yahoo finance
from pandas_datareader import data as pdr

import yfinance as yf
yf.pdr_override() # <= that's all it takes :-)

aapl = pdr.get_data_yahoo("AAPL", start="2022-07-30", end="2023-07-30")
aapl
```

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

	Open	High	Low	Close	Adj Close	Volume
Date						
2022-08-01	161.009995	163.589996	160.889999	161.509995	160.551300	67829400
2022-08-02	160.100006	162.410004	159.630005	160.009995	159.060196	59907000
2022-08-03	160.839996	166.589996	160.750000	166.130005	165.143890	82507500
2022-08-04	166.009995	167.190002	164.429993	165.809998	164.825775	55474100
2022-08-05	163.210007	165.850006	163.000000	165.350006	164.596848	56697000
...
2023-07-24	193.410004	194.910004	192.250000	192.750000	192.750000	45377800
2023-07-25	193.330002	194.440002	192.919998	193.619995	193.619995	37283200
2023-07-26	193.669998	195.639999	193.320007	194.500000	194.500000	47471900
2023-07-27	196.020004	197.199997	192.550003	193.220001	193.220001	47460200
2023-07-28	194.669998	196.630005	194.139999	195.830002	195.830002	48254600

250 rows × 6 columns

- Get the day number of the week

```
# Create a new column of DayofWeek
aapl['DayofWeek'] = aapl.index.weekday
aapl
```

	Open	High	Low	Close	Adj Close	Volume	DayofWeek
Date							
2022-08-01	161.009995	163.589996	160.889999	161.509995	160.551300	67829400	0
2022-08-02	160.100006	162.410004	159.630005	160.009995	159.060196	59907000	1
2022-08-03	160.839996	166.589996	160.750000	166.130005	165.143906	82507500	2
2022-08-04	166.009995	167.190002	164.429993	165.809998	164.825790	55474100	3
2022-08-05	163.210007	165.850006	163.000000	165.350006	164.596848	56697000	4
...
2023-07-24	193.410004	194.910004	192.250000	192.750000	192.750000	45377800	0
2023-07-25	193.330002	194.440002	192.919998	193.619995	193.619995	37283200	1
2023-07-26	193.669998	195.639999	193.320007	194.500000	194.500000	47471900	2
2023-07-27	196.020004	197.199997	192.550003	193.220001	193.220001	47460200	3
2023-07-28	194.669998	196.630005	194.139999	195.830002	195.830002	48254600	4

250 rows × 7 columns

INDEXING PANDAS DATA STRUCTURES

Indexing and slicing

- Pandas dataframe allows extractions of elements from both:
 - rows
 - columns
- Two methods are available to extract elements:
 - `.iloc()`
 - `.loc()`
- Extracting elements based on conditions is also possible

Indexing

INDEX POSITION	index	ROW LABEL / COLUMN NAME
df[position] <i>(for series only)</i>	row	df.loc[row label]
df[start : stop positions]	rows	df.loc[start : stop row labels] df.loc[list of row labels]
Not Possible <i>(See below)</i>	column	df[column name]
	columns	df[list of column names]
df.iloc[start : stop positions , start : stop positions]	rows + columns	df.loc [start : stop / list of row labels , start : stop / list of column names]

Indexing (Cont.)

INDEX POSITION

`X[4]` *(for series only)*

`df[0 : 4]`

	x	y
a	5	'book'
b	4	'pen'
c	6	'pencil'
d	9	'eraser'
e	14	'book'
f	11	'pen'

df

ROW LABEL / COLUMN NAME

`df.loc[' f ']`

`df.loc[' a ', ' b ', ' c ', ' d ']`

`df.loc[' a ' : ' d ']`

Indexing (Cont.)

INDEX POSITION

Not Possible (*See below*)

	X	Y
a	5	'book'
b	4	'pen'
c	6	'pencil'
d	9	'eraser'
e	14	'book'
f	11	'pen'

df

ROW LABEL / COLUMN NAME

```
df['X']
```

```
df[['X', 'Y']]
```

Indexing (Cont.)

INDEX POSITION

```
df.iloc[ 2:5 , 0:2 ]
```

	x	y
a	5	'book'
b	4	'pen'
c	6	'pencil'
d	9	'eraser'
e	14	'book'
f	11	'pen'

df

ROW LABEL / COLUMN NAME

```
df.loc[
```

```
'c':'e',  
'X':'Y']
```

```
df.loc[
```

```
['c','d','e'],  
['X','Y']]
```

Indexing (Cont.)

CONDITIONAL INDEXING

Index labels can also be used to filter based on conditions.

Multiple conditions can also be specified.

	X	Y
a	5	'book'
b	4	'pen'
c	6	'pencil'
d	9	'eraser'
e	14	'book'
f	11	'pen'

df

CONDITIONAL INDEXING

```
df.loc[ df['Y'] == 'book' ]
```

```
df.loc[ df['X'] >= 10, 'Y' ]
```

```
df.loc[  
    ( df['X'] < 5 ) &  
    ( df['Y'] == 'pen' ),  
    ['X':'Y']]
```

Indexing (Cont.) Example : AAPL

- Indexing Series using index numbers
- Indexing DataFrame using index numbers
- Indexing DataFrame using index labels

```
aseries = aapl['Open']  
aseries[1]
```

160.10000610351562

```
aseries[1:5]
```

Date
2022-08-02 160.100006
2022-08-03 160.839996
2022-08-04 166.009995
2022-08-05 163.210007
Name: Open, dtype: float64

```
aapl.iloc[1:10:2, 1:5:2]
```

	High	Close
Date		
2022-08-02	162.410004	160.009995
2022-08-04	167.190002	165.809998
2022-08-08	167.809998	164.869995
2022-08-10	169.339996	169.240005
2022-08-12	172.169998	172.100006

```
aapl.loc['2022-08-01':'2022-08-05']
```

See more in the [Week3-SourceCode.ipynb](#)

	Open	High	Low	Close	Adj Close	Volume
Date						
2022-08-01	161.009995	163.589996	160.889999	161.509995	160.551315	67829400
2022-08-02	160.100006	162.410004	159.630005	160.009995	159.060196	59907000
2022-08-03	160.839996	166.589996	160.750000	166.130005	165.143890	82507500
2022-08-04	166.009995	167.190002	164.429993	165.809998	164.825790	55474100
2022-08-05	163.210007	165.850006	163.000000	165.350006	164.596848	56697000



CUHK Business School
The Chinese University of Hong Kong

COMBINE DATAFRAME

Concat

Documentation of pandas.concat

<https://pandas.pydata.org/docs/reference/api/pandas.concat.html>

- Simple illustration in Excel: Concatenating Two DataFrames: df1 and df2

df1					df2		
	1	2	3			2	b
	4	5	6			1	a
	7	8	9			4	d
	10	11	12			3	c

- Concatenating along the rows (axis=0)
- `pd.concat([df1,df2])` #default axis=0
- Concatenating along the columns (axis=1)
- `pd.concat([df1,df2], axis=1)`

pd.concat([df1, df2])			
	1	2	3
	4	5	6
	7	8	9
	10	11	12
	2	b	NaN
	1	a	NaN
	4	d	NaN
	3	c	NaN

pd.concat([df1, df2],axis=1)					
	1	2	3	2	b
	4	5	6	1	a
	7	8	9	4	d
	10	11	12	3	c

See more in the [Week3-SourceCode.ipynb](#)

Merge

Documentation of pandas.merge

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>

- Simple illustration in Excel:

Merge Two DataFrames: df1 and df2

based on specific column(s) or index

```
pd.merge(df1,df2,how='inner',left_on=0,ri  
ght_on=0,suffixes=['_1','_2'])
```

See more in the [Week3-SourceCode.ipynb](#)

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J
1										
2		df1						df2		
3			1	2	3			2	b	
4			4	5	6			1	a	
5			7	8	9			4	d	
6			10	11	12			3	c	
7										
8										
9										
10			pd.merge(df1,df2,how='inner',left_on=0,right_on=0,suffixes=['_1','_2'])							
11			1	2	3	a		a		
12			4	5	6	b		d		
13								#N/A		
14								#N/A		
15										
16										

The formula bar at the top shows: `=VLOOKUP(C11,H3:I6,2,FALSE)`

Join

Documentation of pandas.merge

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html>

- Simple illustration in Excel:

Join df2 to df1:

```
df1.join(df2, lsuffix='_1', rsuffix='_2')
```

See more in the [Week3-SourceCode.ipynb](#)

df1					df2		
	1	2	3			2	b
	4	5	6			1	a
	7	8	9			4	d
	10	11	12			3	c
df1.join(df2, lsuffix='_1', rsuffix='_2')							
	1	2	3	2	b		
	4	5	6	1	a		
	7	8	9	4	d		
	10	11	12	3	c		



CUHK Business School
The Chinese University of Hong Kong

HANDLING NANS

Handling NaNs

- NaNs (Not a Number) are different from Python's None type which also represent missing data in Python objects.
- Most of the calculation by default will ignore rows with NaN values.
- However, often you may have to specify in your operation, how you want the NaNs values to be dealt with.
- In other cases, you have to remove them from the dataset.

```
.isna()
```

```
#boolean mask for NaN values
```

```
.isnull().sum()
```

```
# count of no. of nan values by columns
```

```
.dropna(axis=0)
```

```
# remove NaN values by rows
```

```
.dropna(axis=1)
```

```
# remove NaN values by columns
```

```
.fillna(mu)
```

```
# replace NA values with specific number mu
```

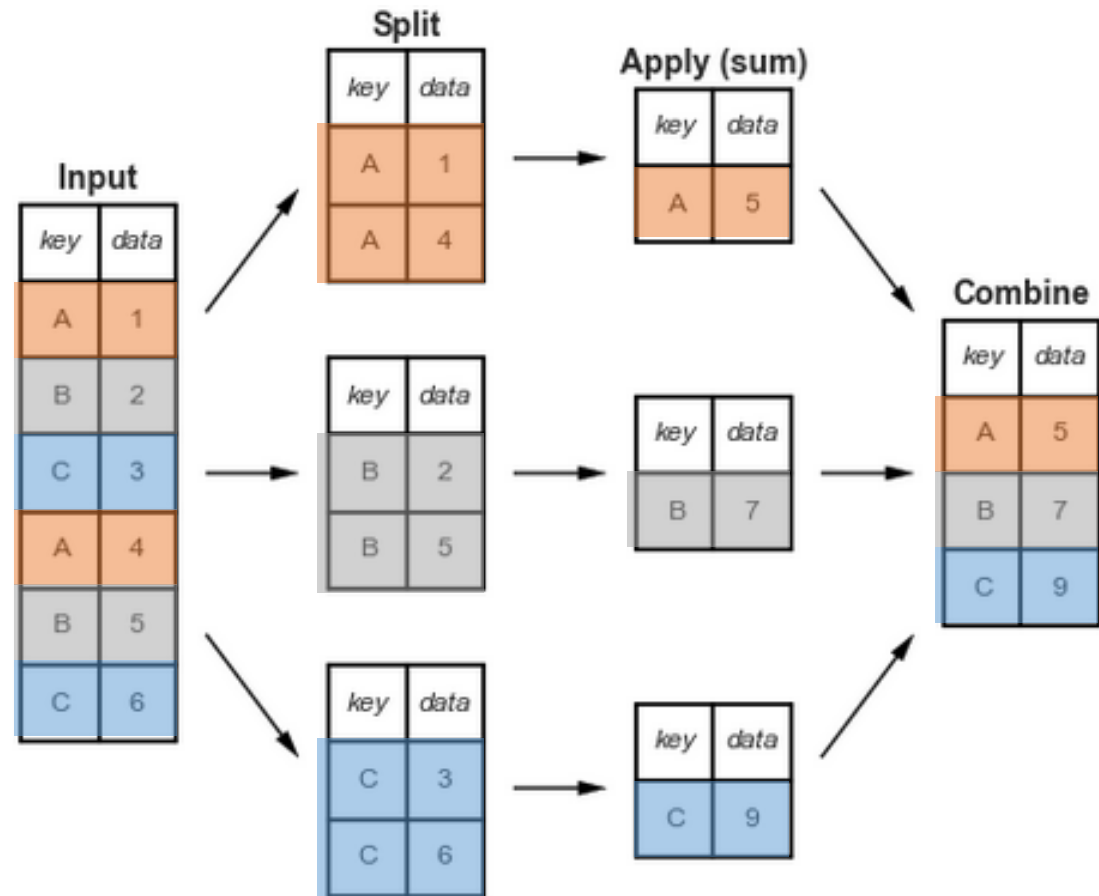


CUHK Business School
The Chinese University of Hong Kong

GROUPBY

GroupBy

- split the data based on some group
- apply a function to each of these groups and
- combine the results from each group together.



GroupBy *Example: Monthly Average*

- Each Month, calculate the average of daily prices and volumes for AAPL and SPY.

```
mean_df = concat_df.groupby( ['Name', 'Year', 'Month'] ).mean()  
mean_df
```

Name	Year	Month	Open	High	Low	Close	Adj Close	Volume
AAPL	2022	8	166.956522	168.589566	165.533913	166.885218	166.085829	6.566259e+07
		9	153.284286	155.308573	150.893333	153.002856	152.305925	9.927251e+07
		10	144.105716	146.800478	142.480476	145.013333	144.352798	8.895903e+07

See more in the [Week3-SourceCode.ipynb](#)

Name	Year	Month	Open	High	Low	Close	Adj Close	Volume
AAPL	2022	8	166.956522	168.589566	165.533913	166.885218	166.085829	6.566259e+07
		9	153.284286	155.308573	150.893333	153.002856	152.305925	9.927251e+07
		10	144.105716	146.800478	142.480476	145.013333	144.352798	8.895903e+07
		11	146.057143	148.032855	143.508094	145.843331	145.385676	8.213560e+07
		12	138.735716	140.166667	136.277144	137.876666	137.476318	7.979672e+07
	2023	1	135.126000	137.271498	133.737001	135.778999	135.384741	7.218262e+07
		2	150.226316	152.296842	149.011581	150.968421	150.674966	6.879994e+07
		3	154.455218	156.363044	153.241304	154.964782	154.750734	6.609855e+07
		4	164.437370	165.925790	163.552632	165.045790	164.817817	5.103735e+07
		5	172.510456	173.752728	171.444545	172.622728	172.526194	5.796161e+07
		6	183.679048	185.438095	182.615714	184.283333	184.283333	6.176672e+07
		7	192.319474	193.792106	191.006842	192.198948	192.198948	5.037924e+07
SPY	2022	8	415.512606	418.129130	412.880870	415.121306	408.451383	6.275628e+07
		9	385.002379	388.166187	380.890002	384.234288	378.848980	9.518612e+07
		10	370.311904	375.305238	366.714283	371.559524	367.091306	9.641581e+07
		11	390.998567	393.993336	387.656667	391.115239	386.411854	8.314216e+07
		12	391.231905	393.636665	387.524288	390.379047	386.509270	8.266541e+07
	2023	1	393.562001	396.882501	391.125002	394.695999	391.740210	7.877250e+07
		2	406.607367	409.583160	404.198949	407.072633	404.024159	8.437341e+07
		3	395.691303	398.815657	392.859999	396.023913	393.777914	1.093911e+08
		4	410.380522	412.383159	408.561051	410.921053	409.400296	7.345700e+07
		5	414.127727	415.745454	411.808183	413.880909	412.349199	8.094116e+07
		6	432.527143	435.110947	431.288096	433.684762	432.849303	8.332167e+07
		7	448.721583	450.590526	447.274207	449.013685	449.013685	6.908051e+07



CUHK Business School
The Chinese University of Hong Kong

LAMBDA AND APPLY

Lambda and apply

- Apply user-defined functions to Pandas DataFrame

1. A regular Python function



```
def twosum(x):  
    return x+10
```

2. A lambda function



```
twosum = lambda x:x+10
```

3. A lambda function applied to the entire DataFrame in Pandas



```
pd.apply( lambda x:x+10 )
```



CUHK Business School
The Chinese University of Hong Kong

MATPLOTLIB

Why Matplotlib?

- Most popular plotting library in python.
- It can easily create plots and render in IPython environments such as jupyter notebook.
- It is built on top of NumPy and designed to work with SciPy stack.

Matplotlib

Matplotlib has two main components:

1. FIGURE

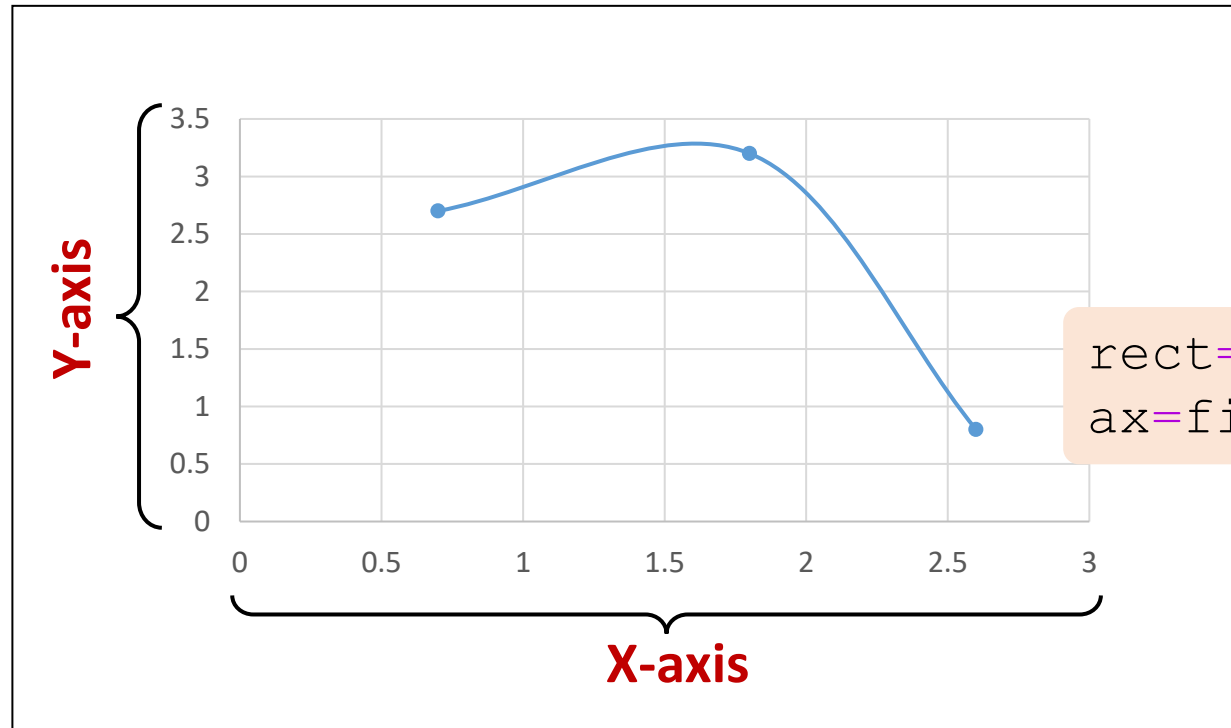
area where things
can be drawn

```
fig=plt.figure()
```

2. AXES

area where the data
can be plotted

```
rect=[0.1,0.2,0.8,0.9]  
ax=fig.add_axes(rect)
```



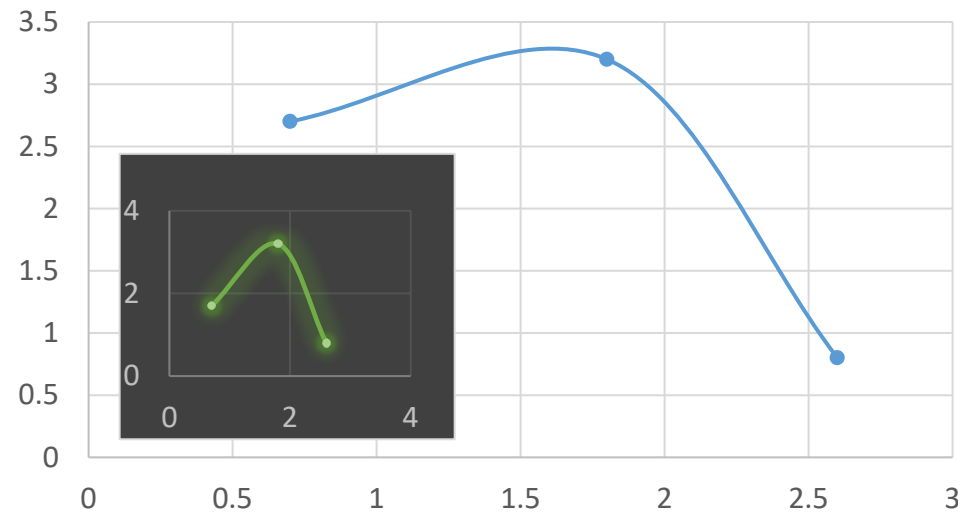
Matplotlib (Cont.)

A figure can contain multiple figures

```
fig=plt.figure()
```

```
rect=[0.1,0.2,0.8,0.9]  
ax1=fig.add_axes(rect)
```

```
rect=[0.2,0.3,0.3,0.5]  
ax2=fig.add_axes(rect)
```



Matplotlib (Cont.)

A figure can contain multiple plots

```
fig, ax = plt.subplots(2,2)
```

Method to
create
subplot

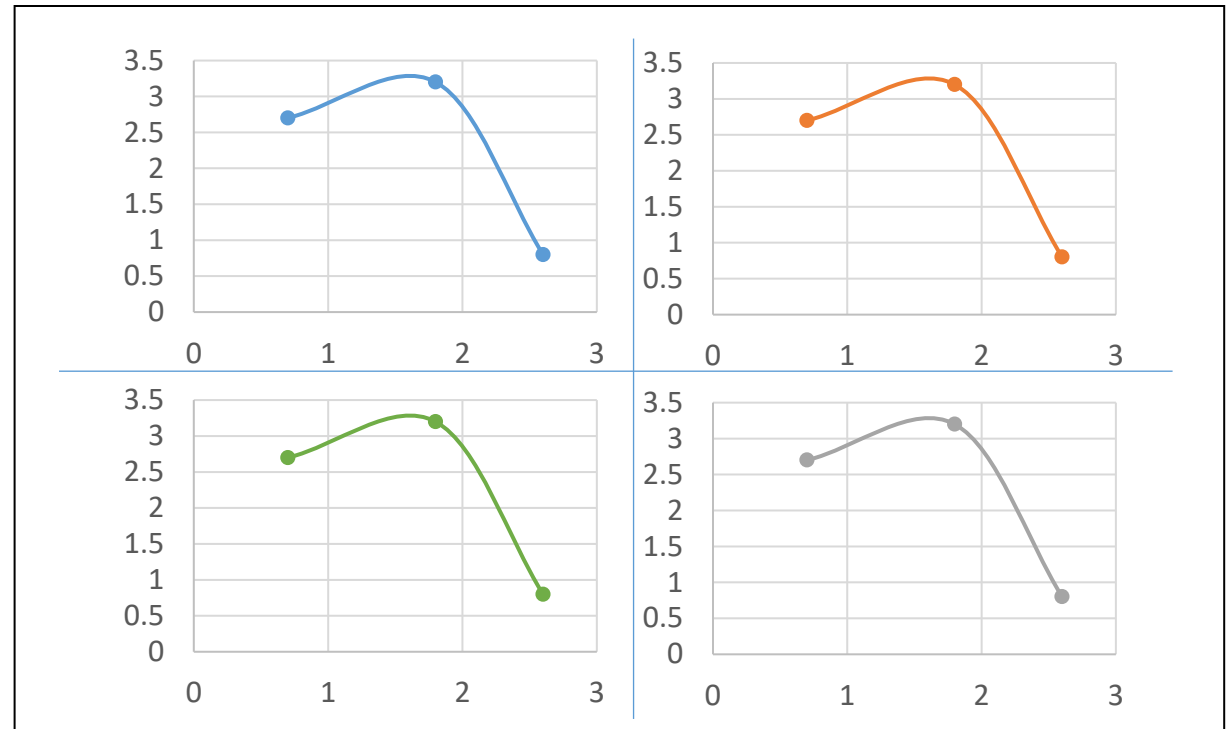
two rows and
two columns

```
ax[0,0].plot()
```

```
ax[0,1].plot()
```

```
ax[1,0].plot()
```

```
ax[1,1].plot()
```



Matplotlib (Cont.)

- Matplotlib supports
 - Line graphs
 - Histograms
 - Bar Charts
 - Pie Charts
 - Scatter plots
 - Paths
 - 3-D plotting
 - Contours

<https://matplotlib.org/stable/>

This Week

1. Pandas:

- - Importing and Exporting Datasets
- - Data Structures
- - Methods
- - Datetime
- - Indexing Pandas Data Structures
- - Combine Dataframe
- - Handling NaNs
- - Groupby
- - Apply and Lambda Functions

2. Matplotlib

Next Weeks

- 1. Statistical Tools and Linear Algebra in Python**
- 2. Python Projects**
- 3. ...**