上海立信会计金融学院
SHANGHAI LIXIN UNIVERSITY OF ACCOUNTING AND FINANCE

《Python金融数据分析》

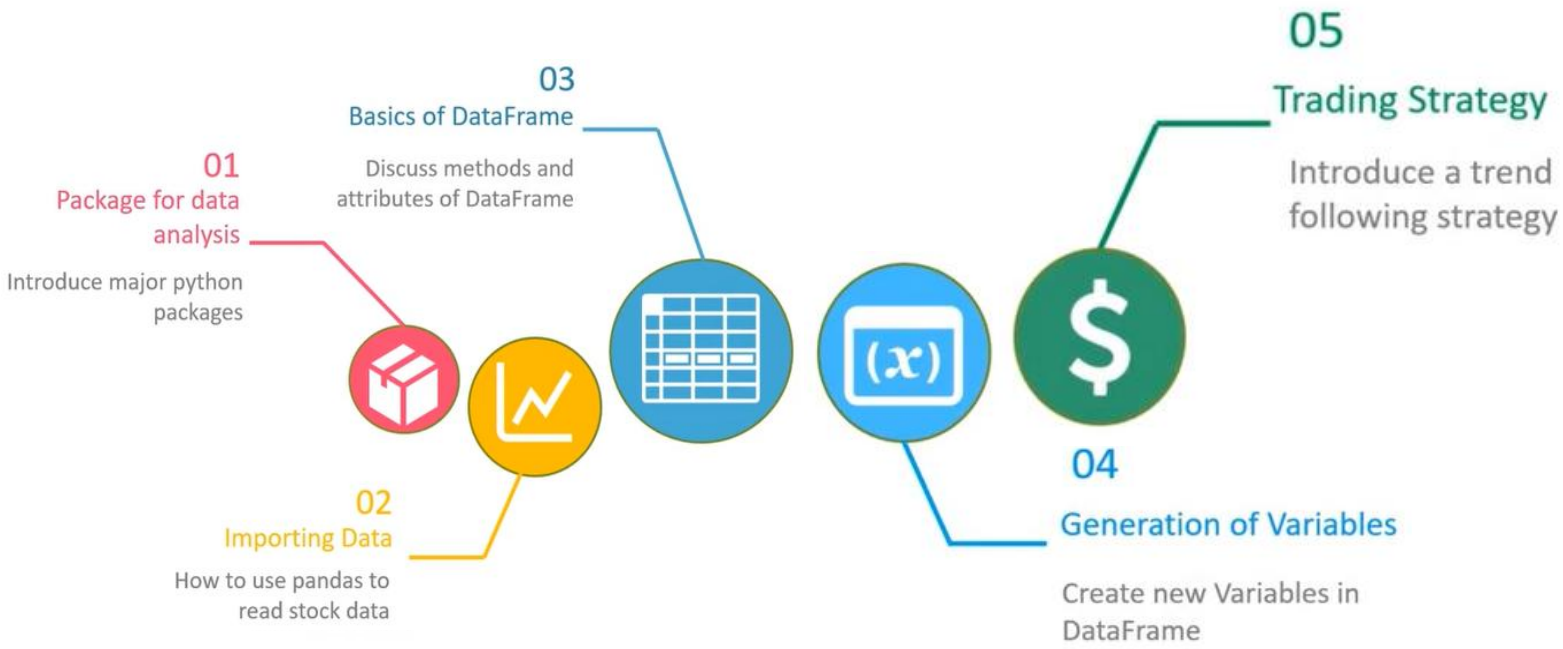
Hong Cheng（程宏）

School of Statistics and Mathematics

Shanghai LiXin University of Accounting and Finance

March 2022

通过以下步骤介绍Python教程



希望你能够在第一个Topic之后灵活运用股票数据
以实现你的各种奇妙想法。

03 Basics of DataFrame

We will present basic attributes and methods of DataFrame, which we will use a lot in this course.

Then we will discuss one of the most important skills for beginners, **how to select some portion of data**.

To begin with, let's first take a look at what DataFrame looks like.

print serveral rows at the top DataFrame, fb which is historical data for Facebook.

Basic structure of a DataFrame

In [3]
```
fb.head()
```

Out [3]

	Open	High	Low	Close	Adj Close	Volume
Date						
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

In [3]
```
fb.head()
```

Out [3]

	Open	High	Low	Close	Adj Close	Volume
Date						
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

Pandas DataFrame 是什么数据结构？？？

Pandas DataFrame is a tabular structure of data.

Attributes of DataFrame

```
In  [1]   fb.index
          fb.index[0]   #the first index
          fb.index[-1]   #the last index
          fb.columns
```

Another important attribute of a DataFrame is the **size**. **The size of DataFrame can be described by the number of rows and columns**.

```
In  [2]   fb.shape   #number of rows and columns
```

```
Out [2]   (786, 6)
```

Number of observations

```
Out [2]   (786, 6)
```

Number of variables

As methods, **head** and **tail** are often **used to check whether data is correct or to check contents of index and columns**.

Methods of DataFrame

```
In [3]  fb.tail()
```

out [3]

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-30	241.110001	246.419998	238.410004	242.720001	242.720001	14270800
2018-01-31	245.770004	249.270004	244.449997	245.800003	245.800003	11964400
2018-02-01	238.520004	246.899994	238.059998	240.500000	240.500000	12980600
2018-02-02	237.000000	237.970001	231.169998	233.520004	233.520004	17961600
2018-02-05	227.000000	233.229996	205.000000	213.699997	213.699997	28869000

With tail, you can get the last five rows.

There is **another method called describe**, which can give you some summary statistics for each column.

Methods of DataFrame

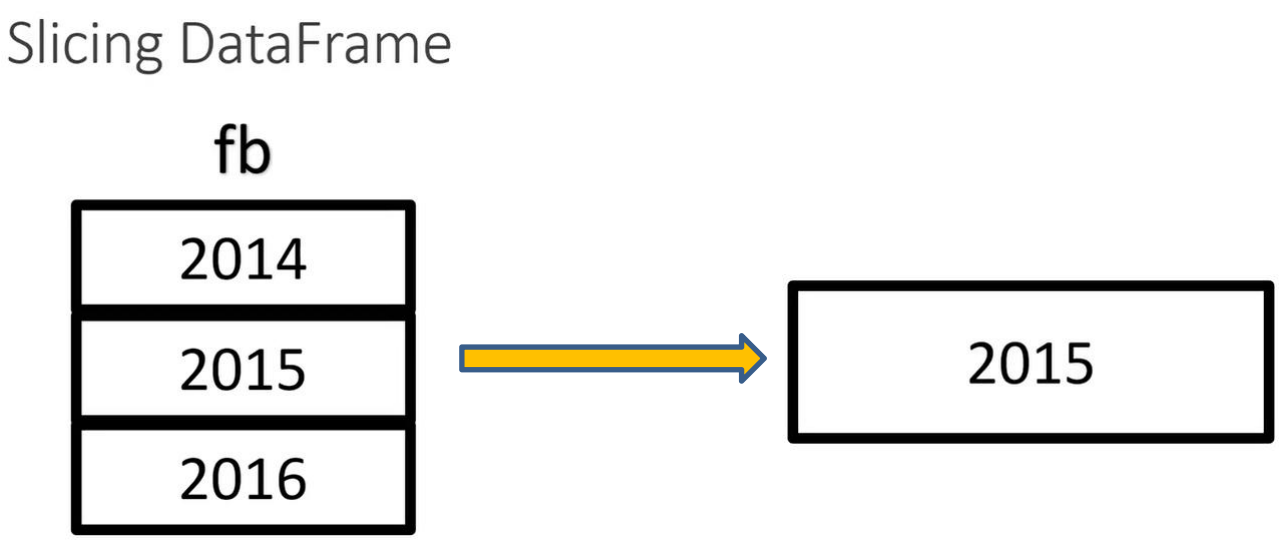
```
In [3]  fb.describe()
```

Out [3]

	Open	High	Low	Close	Adj Close	Volume
count	780.000000	780.000000	780.000000	780.000000	780.000000	7.800000e+02
mean	80.212705	81.285654	79.022397	80.264897	79.914215	1.204453e+07
std	64.226121	65.048907	63.190963	64.198375	64.327846	8.221848e+06
min	19.250000	19.500000	18.940001	19.139999	18.576082	1.311200e+06
25%	25.525000	26.085000	24.845000	25.475000	25.134513	7.215200e+06
50%	53.379999	54.034999	52.930000	53.420000	53.035403	9.728700e+06
75%	113.322502	115.779999	110.297499	113.702501	113.261238	1.408885e+07
max	245.770004	249.270004	244.449997	246.850006	246.850006	9.232320e+07

Next, we will discuss selection of data from DataFrame.

For example, select only close price of 2015. **How to do that.**



There are **two ways to slice a DataFrame**, selection **by label** and selection **by position.**

Slicing DataFrame

- Selection by label
  - .loc
- Selection by position
  - .iloc

2015

For example, if you want the close price on the first day of 2015, you can use a method **loc** along with labels of index and the column.

The first entry is the index label. **The second entry** of label is a column name.

Slicing DataFrame

```
In [3]  fb.loc['2015-01-02', 'Close']
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

Alternatively, you also can select by position using **iloc**.

The first entry stands for row number. **The second** is a column number.
The position starts with zero.

Slicing DataFrame

(3)

```
In [3]  fb.iloc[1, 3]
```

(1)

	Open	High	Low	Close	Adj Close	Volume
Date						
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

You can select multiple rows. For example, we can get close price for the whole year of 2015. **This selects close price from 2015, January 1 to 2015, December 31.**

Slicing DataFrame

```
In  [4]  fb.loc['2015-01-01':'2015-12-31', 'Close']

Out [4]  Date
         2015-01-02    20.129999
         2015-01-05    19.790001
         2015-01-06    19.139999
                        ....
         2015-12-31    32.959999
         Name: Close, Length: 252, dtype: float64
```

We also can **select multiple columns**. The colon sign in the second entry of iloc means all columns.

Slicing DataFrame

```
In  [5]  fb.iloc[624: , : ]
```

Out [5]

Date	Open	High	Low	Close	Adj Close	Volume
2017-06-23	158.679993	159.320007	153.220001	153.830002	153.587982	27214700
2017-06-26	155.160004	156.600006	148.330002	152.149994	151.910614	26599000
2017-06-27	151.440002	151.789993	146.350006	146.580002	146.349380	24987300
2017-06-28	149.320007	151.940002	145.750000	151.750000	151.511246	24873700
2017-06-29	150.600006	150.720001	144.080002	146.679993	146.449219	26610600

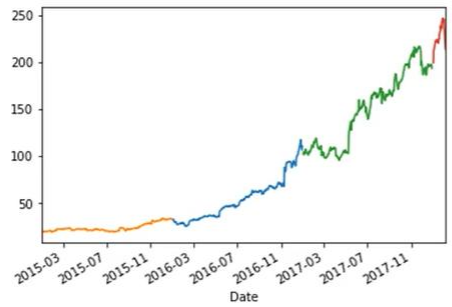
DataFrame has built in method of plot, which **means you can plot data directly without importing matplotlib.**

There is a method called plot in pandas DataFrame. You can easily visualize the close price with this method. You even can slice close price into different years, and visualize them one by one.

Visualizing stock price

In [6]:
```
fb.loc['2015-01-01':'2015-12-31','Close'].plot()# 2015
fb.loc['2016-01-01':'2016-12-31','Close'].plot() # 2016
fb.loc['2017-01-01':'2017-12-31','Close'].plot()# 2017
fb.loc['2018-01-01':'2018-12-31','Close'].plot()# 2018
```

Out [6]:



Lab 1: Please, get familiar with this part, and **do some practice in Jupyter Notebook**. We will use these skills a lot in our formal analysis of financial data.

Instructions

In this Jupyter Notebook, you will practice the following basics of DataFrame:

1. Import stock data (in csv format) into a new DataFrame

2. Display the size of a DataFrame using ".shape"

3. Display the summary statistics of a DataFrame using ".describe()"

4. Slice row(s) of data of a DataFrame using "Selection by label" - loc and "Selection of position - iloc"

5. Plot the data of a DataFrame

1. Import stock data (in csv format) into a new DataFrame

In [1]:
```
#import the packages "Pandas" and "MatPlotLib" into Jupyter Notebook
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
#import Facebook's stock data
fb = pd.DataFrame.from_csv('../data/facebook.csv')
```

In [3]:
```
print(fb.head())
```

```
                Open       High        Low      Close  Adj Close   Volume
Date
2014-12-31  20.400000  20.510000  19.990000  20.049999  19.459270  4157500
2015-01-02  20.129999  20.280001  19.809999  20.129999  19.536913  2842000
2015-01-05  20.129999  20.190001  19.700001  19.790001  19.206934  4948800
2015-01-06  19.820000  19.840000  19.170000  19.190001  18.624611  4944100
2015-01-07  19.330000  19.500000  19.080000  19.139999  18.576082  8045200
```

In [2]:
```
#It is your turn to import Microsoft's stock data - "microsoft.csv", which is located in the same folder of facebook.csv
#Replace "None" with your code
ms = None
```

In [5]:
```
# print head of ms, 1 line
```

Expected Output:

Date	Open	High	Low	Close	Adj Close	Volume
2014-12-31	46.730000	47.439999	46.450001	46.450001	42.848763	21552500
2015-01-02	46.660000	47.419998	46.540001	46.759998	43.134731	27913900
2015-01-05	46.369999	46.730000	46.250000	46.330002	42.738068	39673900
2015-01-06	46.380001	46.750000	45.540001	45.650002	42.110783	36447900
2015-01-07	45.980000	46.459999	45.490002	46.230000	42.645817	29114100

2. Display the size of a DataFrame using ".shape"

```
In [6]: print(fb.shape)
(780, 6)

In [7]: # print the shape of ms, 1 line
```

3. Display the summary statistics of a DataFrame using ".describe()"

```
In [8]: # print summary statistics of Facebook
print(fb.describe())
```

	Open	High	Low	Close	Adj Close
count	780.000000	780.000000	780.000000	780.000000	780.000000
mean	80.212705	81.285654	79.022397	80.264897	79.914215
std	64.226121	65.048907	63.190963	64.198375	64.327846
min	19.250000	19.500000	18.940001	19.139999	18.576082
25%	25.525000	26.085000	24.845000	25.475000	25.134513
50%	53.379999	54.034999	52.930000	53.420000	53.035403
75%	113.322502	115.779999	110.297499	113.702501	113.261238
max	245.770004	249.270004	244.449997	246.850006	246.850006

	Volume
count	7.800000e+02
mean	1.204453e+07
std	8.221848e+06
min	1.311200e+06
25%	7.215200e+06
50%	9.728700e+06
75%	1.408885e+07
max	9.232320e+07

```
In [21]: # print summary statistics of Microsoft
```

4. Slice row(s) of data of a DataFrame using "Selection by label" - loc and "Selection of position - iloc"

```
In [10]: # select all the price information of Facebook in 2016.
         fb_2015 = fb.loc['2015-01-01':'2015-12-31']

In [11]: # print the price of Facebook on '2015-03-16'
         print(fb_2015.loc['2015-03-16'])

Open         2.288000e+01
High         2.311000e+01
Low          2.273000e+01
Close        2.297000e+01
Adj Close    2.237908e+01
Volume       5.923900e+06
Name: 2015-03-16 00:00:00, dtype: float64

In [12]: # select all the price information of Microsoft in 2016.

In [8]: # print the price of Microsoft on '2016-03-16'

Out[8]: Open         5.345000e+01
        High         5.460000e+01
        Low          5.340000e+01
        Close        5.435000e+01
        Adj Close    5.187095e+01
        Volume       3.169170e+07
        Name: 2016-03-16 00:00:00, dtype: float64
```

Expected Output:

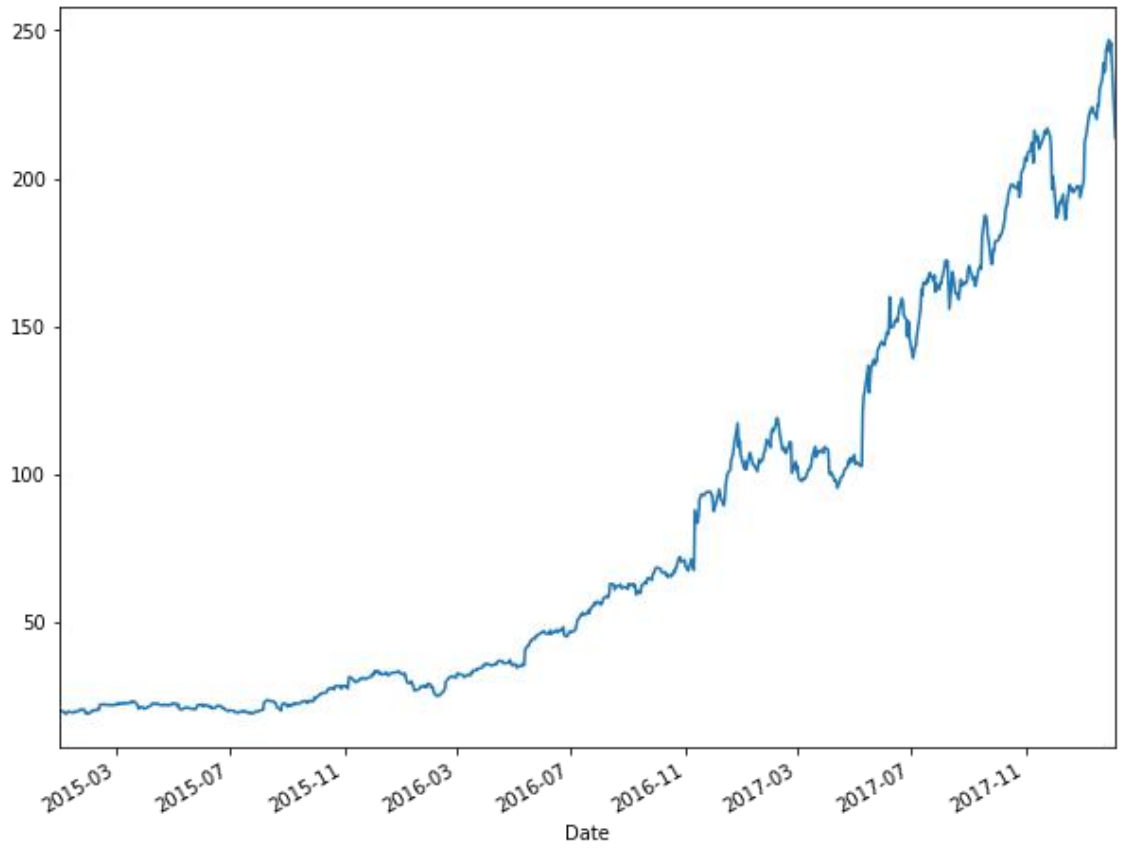
```
Open         5.345000e+01
High         5.460000e+01
Low          5.340000e+01
Close        5.435000e+01
Adj Close    5.187095e+01
Volume       3.169170e+07
```

```
In [14]: # print the opening price of the first row
         print(fb.iloc[0, 0])

20.4

In [15]: # print the opening price of the last row
```

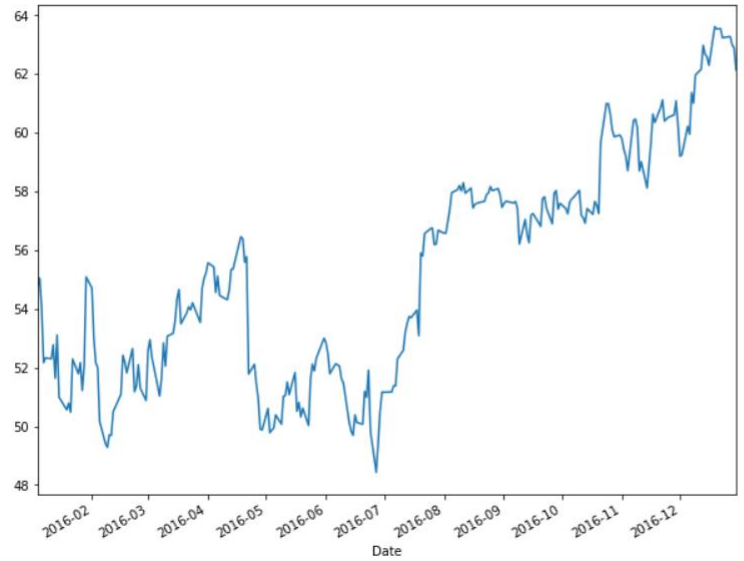
Expected Output: 90.559998

5. Plot the data of a DataFrame

In [16]:
```
plt.figure(figsize=(10, 8))
fb['Close'].plot()
plt.show()
```



In [34]:
```
plt.figure(figsize=(10, 8))
# plot only the Close price of 2016 of Microsoft, 1 line
```



Data和DataFrame.ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

Jupyternote Book课堂练习
十五分钟

we will learn more methods which **help us to create new variables**.

To create new variables for other columns, we will learn a new method of DataFrame slicing, which is a special one.

It is to select particular columns.

For example, we want to select column of Close price.

Select a single column from a DataFrame

```
In [3]    fb['Close']
```

Date	Open	High	Low	Close	Adj Close	Volume
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

If you want to select multiple columns for example both Open and Close prices, you can **use lists of names by putting all names in a square bracket**, and give this names list an entry to over 10 multiple columns.

Select multiple columns from a DataFrame

```
In [4]    fb[['Open', 'Close']]
```

Date	Open	High	Low	Close	Adj Close	Volume
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200

If you want to **create a new column** called Price1 which is Close price of tomorrow, you can do this.

The fb now has a new column. **What is the right side of this code?**

Create a new column in a DataFrame

Close price of tomorrow

```
In [3]  fb['Price1'] = fb['Close'].shift(-1)
```

Out [3]

Date	Open	High	Low	Close	Adj Close	Volume	Price1
2014-12-31	20.400000	20.510000	19.990000	20.049999	19.459270	4157500	20.129999
2015-01-02	20.129999	20.280001	19.809999	20.129999	19.536913	2842000	19.790001
2015-01-05	20.129999	20.190001	19.700001	19.790001	19.206934	4948800	19.190001
2015-01-06	19.820000	19.840000	19.170000	19.190001	18.624611	4944100	19.139999
2015-01-07	19.330000	19.500000	19.080000	19.139999	18.576082	8045200	19.860001

Here's a demonstration of a shift(-1). It shifts a whole column upwards by one row.

What .shift() does?

Date	Close
2014-12-31	20.12999
2015-01-02	19.79001
2015-01-05	19.19001
2015-01-06	19.13999

```
fb['close'].shift(-1)
```

Next, we **create a new variable called PriceDiff** which is the price change between tomorrow and today.

Create a "Price difference" column

PriceDiff = (Close price of tomorrow
- Close price of today)

We do not need to compute a price difference one day by one day. **Instead**, we can take this job nicely **with only one line of code**. It is very intuitive and clean. That is why a lot of practitioners like to use Python to pre-process data. We can check the updated DataFrame fb, the new column is generated.

Price difference

In [3]:
```
fb['PriceDiff'] = fb['Price1'] - fb['Close']
fb.head()
```

Out [3]:

Date	Open	High	Low	Close	Volume	Price1	PriceDiff
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002

Similarly, we can calculate the daily return. We know daily return should be PriceDiff divided by Close, as we can see from the formula.

Create a "Daily return" column

$$\text{Daily return} = \frac{\text{Price difference}}{\text{Close price of today}}$$

Daily return

In [3]
```
fb['Return'] = fb['PriceDiff']/fb['Close']
```

Out [3]

Date	Open	High	Low	Close	Volume	Price1	PriceDiff
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002

We double check if the new variable is created.

Daily return

```
In [3]  fb.head()
```

Out [3]

Date	Open	High	Low	Close	Volume	Price1	PriceDiff	Return
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000	0.003990
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998	-0.016890
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000	-0.030318
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002	-0.002606
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002	0.037618

Next, we will create a new variable direction.

Create a "Direction" column

Rules

PriceDiff > 0	➡	Up	1
PriceDiff <= 0	➡	Down	-1

Here's the code for the List comprehension.

Direction

Condition

List comprehension

```
In [3]  fb['Direction'] = [1 if fb.loc[ei, 'PriceDiff'] > 0 else -1
                           for ei in fb.index]
```

Out [3]

Date	Open	High	Low	Close	Volume	Price1	PriceDiff	Return	
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000	0.003990	1
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998	-0.016890	-1
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000	-0.030318	-1
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002	-0.002606	-1
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002	0.037618	

List comprehension is a very important tool for creation of new variables following very complicated rules.

A Moving average is a widely used signal in stock trading, which is loading average price over a defined number of days.

Random fluctuations are very common in stock price. Taking an average price in a period can smooth out noise.

For example, we want to calculate the moving average of close price over three days, which is the average of Close price of today, yesterday, and the day before yesterday.

Create a "Moving average" column

In [3]
```
fb['Average3'] = (fb['Close'] + fb['Close'].shift(1)
                  + fb['Close'].shift(2))/3
```

Out [3]

Date	Open	High	Low	Close	Volume	Price1	PriceDiff	Return
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000	0.003990
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998	-0.016890
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000	-0.030318
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002	-0.002606
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002	0.037618

Let us take a close look, **what is a shift(1).**

What .shift(1) does?

Date	Close
2014-12-31	20.04999
2015-01-02	20.12999
2015-01-05	19.79001
2015-01-06	19.19001

```
fb['close'].shift(1)
```

What .shift(1) does?

Date	Close
2014-12-31	NaN
2015-01-02	20.04999
2015-01-05	20.12999
2015-01-06	19.79001

```
fb['close'].shift(1)
```

What is shift(2)? It shifts the column of Close price downward by two rows, which get the Close price two days ago.

What .shift(2) does?

Date	Close
2014-12-31	20.04999
2015-01-02	20.12999
2015-01-05	19.79001
2015-01-06	19.19001

```
fb['close'].shift(2)
```

Date	Close
2014-12-31	NaN
2015-01-02	NaN
2015-01-05	20.04999
2015-01-06	20.12999

```
fb['close'].shift(2)
```

Check the new update fb.

Moving average

```
In [3]  fb.head()
```

Out [3]

Date	Open	High	Low	Close	Volume	Price1	PriceDiff	Return	Average3
2014-12-31	20.400000	20.510000	19.990000	20.049999	4157500	20.129999	0.080000	0.003990	NaN
2015-01-02	20.129999	20.280001	19.809999	20.129999	2842000	19.790001	-0.339998	-0.016890	NaN
2015-01-05	20.129999	20.190001	19.700001	19.790001	4948800	19.190001	-0.600000	-0.030318	19.990000
2015-01-06	19.820000	19.840000	19.170000	19.190001	4944100	19.139999	-0.050002	-0.002606	19.703334
2015-01-07	19.330000	19.500000	19.080000	19.139999	8045200	19.860001	0.720002	0.037618	19.373334

In Python, **if** some valuable data is missing, we **call it is missing values**. It will also show **NaN value** if read into DataFrame.

DataFrame has very nice methods to handle NaN values.

In DataFrame, it has building method to compute moving average over any number of days.

MA40 is a moving average Close price over 40 days, and **MA200** is a moving average price over 200 days.

Rolling is a very nice method to provide us rolling window calculation.

Calculate moving average using .rolling()

In [3]:
```
fb['MA40'] = fb['Close'].rolling(40).mean()
fb['MA200'] = fb['Close'].rolling(200).mean()
```

Now, let us plot these moving averages and compare them with Close price.

Plot moving average

In [4]:
```
fb['Close'].plot()
fb['MA40'].plot()
fb['MA200'].plot()
```

Out [4]:

Fast signal

Slow signal

we call moving average 40 is a fast signal and **MA200 is a slow signal** which reflects the price over a long history.

If MA40 is above MA200, some traders who we call **trend-following traders**, they believe the stock price will move upwards for a while.

Plot moving average

In [4]:
```
fb['Close'].plot()
fb['MA40'].plot()
fb['MA200'].plot()
```

Out [4]:

Fast signal

Slow signal

MA40>MA200

What have learnt so far…

- Building new variables with column-wise operations
  - ✓Price difference
  - ✓Daily return
  - ✓Direction
  - ✓Moving average
- List comprehension

Next Lesson, we will build a simple strategy for stock trading based on our discussion about slow and fast signals.

Lab 2: Create features and columns in DataFrame

Instructions

In this Jupyter Notebook, you will practice the following codes to create new features/columns in a DataFrame

1. Create new columns in the DataFrame by arithmetic calculation - Price Difference and Daily Return

2. Create a new column using List Comprehension - Direction

3. Create a new column using Rolling Window Calculation - Any days of Moving Average

1. Create new columns in the DataFrame by arithmetic calculation - Price Difference and Daily Return

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
fb = pd.DataFrame.from_csv('../data/facebook.csv')
ms = pd.DataFrame.from_csv('../data/microsoft.csv')
```

In [3]:
```
#Create a new column PriceDiff in the DataFrame fb
fb['PriceDiff'] = fb['Close'].shift(-1) - fb['Close']
```

In [5]:
```
#Your turn to create PriceDiff in the DataFrame ms
```

In [7]:
```
#Run this code to display the price difference of Microsoft on 2015-01-05
print(ms['PriceDiff'].loc['2015-01-05'])
```

Expected Output: -0.68

Daily Return is calcuated as PriceDiff/Close

In [ ]:
```
#Create a new column Return in the DataFrame fb
fb['Return'] = fb['PriceDiff'] /fb['Close']
```

In [ ]:
```
#Your turn to create a new column Return in the DataFrame MS
ms['Return'] = None
```

In [ ]:
```
#Run this code to print the return on 2015-01-05
print(ms['Return'].loc['2015-01-05'])
```

Expected Output: -0.0146773142811

2. Create a new column using List Comprehension - Direction

In [ ]:
```
#Create a new column Direction.
#The List Comprehension means : if the price difference is larger than 0, denote as 1, otherwise, denote as 0,
#for every record in the DataFrame - fb

fb['Direction'] = [1 if fb['PriceDiff'].loc[ei] > 0 else 0 for ei in fb.index ]
```

In [ ]:
```
# Your turn to create a new column Direction for MS

ms['Direction'] = None
```

In [ ]:
```
# Run the following code to show the price difference on 2015-01-05
print('Price difference on {} is {}. direction is {}'.format('2015-01-05', ms['PriceDiff'].loc['2015-01-05'], ms['Direction'].loc['2015-01-05'])
```

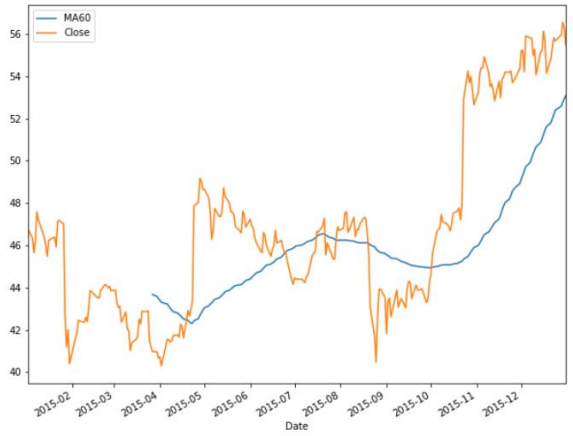
Expected Output: Price difference on 2015-01-05 is -0.6799999999999997. direction is 0

3. Create a new column using Rolling Window Calculation - Any days of Moving Average

In [ ]:
```
fb['ma50'] = fb['Close'].rolling(50).mean()

#plot the moving average
plt.figure(figsize=(10, 8))
fb['ma50'].loc['2015-01-01':'2015-12-31'].plot(label='MA50')
fb['Close'].loc['2015-01-01':'2015-12-31'].plot(label='Close')
plt.legend()
plt.show()
```

In [ ]:
```
# You can use .rolling() to calculate any numbers of days' Moving Average. This is your turn to calculate "60 days"
# moving average of Microsoft, rename it as "ma60". And follow the codes above in plotting a graph

ms['ma60'] = None

#plot the moving average
plt.figure(figsize=(10, 8))
ms['ma60'].loc['2015-01-01':'2015-12-31'].plot(label='MA60')
ms['Close'].loc['2015-01-01':'2015-12-31'].plot(label='Close')
plt.legend()
plt.show()
```

Expected Output:



上海立信会计金融学院
SHANGHAI LIXIN UNIVERSITY OF ACCOUNTING AND FINANCE

Data和Create new features and columns in DataFrame.ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

Jupyternote Book课堂练习
十五分钟

05
Trading Strategy

You will learn how to **build** **a simple strategy of stock trading**.

We hope with this **money-making case**, you are highly **motivated to** learn Python we covered so far, and get ready for the next several topics.

Fast Signal?? Slow Signal??

Moving Average of short period is more closely associated with recent change of stock price, which we call **Fast Signal**.

Moving Average over long period reflects the price change over long-term history, which we call **slow signal**.

We **create MA10 and MA50**, which are fast signal and slow signal respectively.

Strategy – Fast signal and slow signal

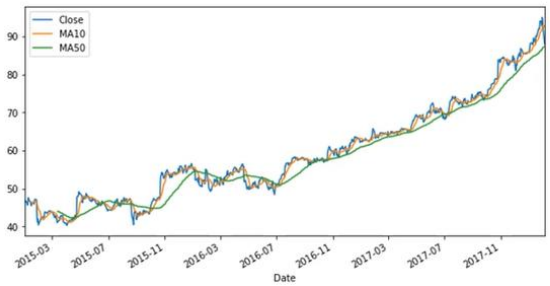
```
In [3]: ms=pd.DataFrame.from_csv('data/microsoft.csv')
        ms['MA10']=ms['Close'].rolling(10).mean()   # fast signal
        ms['MA50']=ms['Close'].rolling(50).mean()   # slow signal
```

Then, we plot close price MA10 and MA50.

Plot the stock price with two moving averages

```
In [3]: ms['Close'].plot(legend=True)
        ms['MA10'].plot(legend=True) # fast signal
        ms['MA50'].plot(legend=True) # slow signal
```

Out[3]:

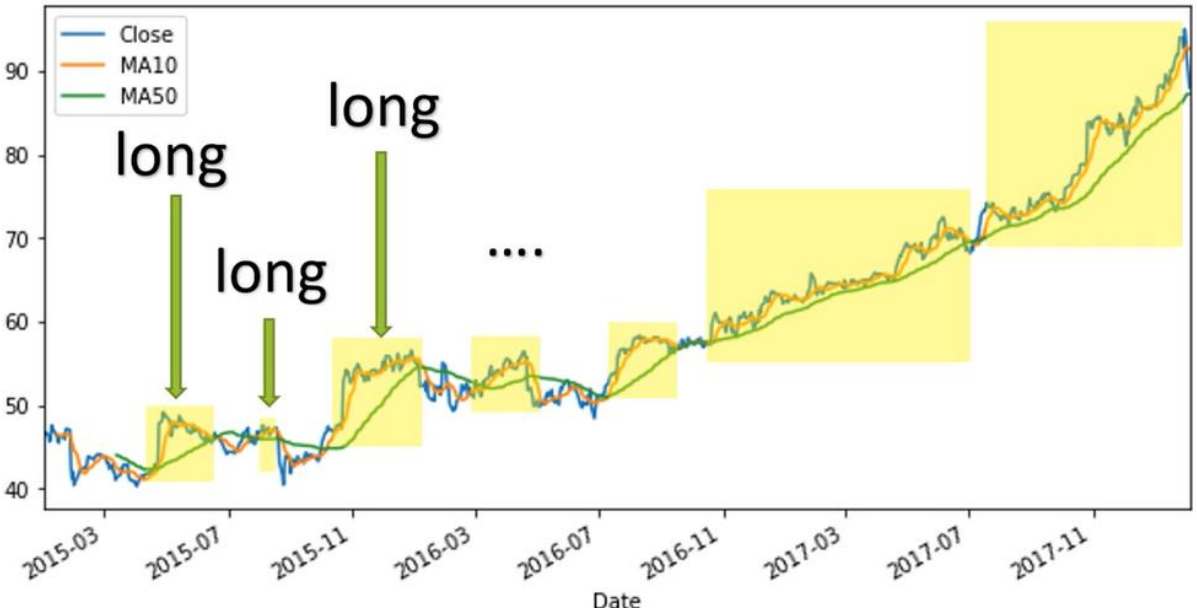


If MA10 is larger than MA50, the stock price is believed by some traders, that it goes up in the next several days. Otherwise, the price will decrease.

Our strategy is, if MA10 is larger than MA50, we will buy and hold one share of stock. Alternately speaking, we will long one share of stock. **This yellow area labels the days on which we buy and hold one share of stock.**

Plot the stock price with two moving averages

MA10 > MA50 → Buy and hold one share of stock



We can create a new variable called shares, to denote whether we long or not. It is created **using list comprehension**.

Long or not?

```
In [3]: ms['Shares']= [ 1 if ms.loc[ei,'MA10']>ms.loc[ei,'MA50'] else 0
                        for ei in ms.index]
```

Next, we will compute **daily profit.**

Daily profit

```
In [4]: ms['Close1']=ms['Close'].shift(-1)
        ms.iloc[500:505,:]
```

Out[4]:

Date	Open	High	Low	Close	Adj Close	Volume	MA10	MA50	Shares	Close1
2016-12-23	63.450001	63.540001	62.799999	63.240002	61.569557	12398000	63.020000	60.4170	1	63.279999
2016-12-27	63.209999	64.070000	63.209999	63.279999	61.608490	11763200	63.131000	60.5342	1	62.990002
2016-12-28	63.400002	63.400002	62.830002	62.990002	61.326157	14653300	63.132000	60.6496	1	62.900002
2016-12-29	62.860001	63.200001	62.730000	62.900002	61.238541	10181600	63.154001	60.7544	1	62.139999
2016-12-30	62.959999	62.990002	62.029999	62.139999	60.498604	25579900	63.110000	60.8466	1	62.580002

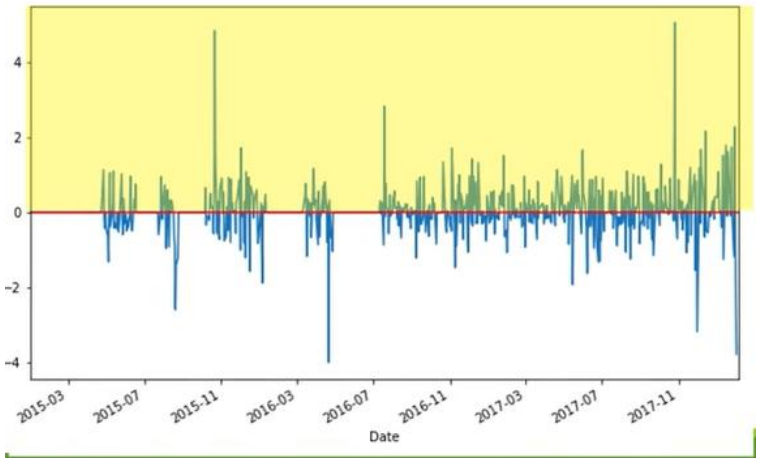
First, we **create variable Close1**, which is the close price of tomorrow.

Then, we'll create a new variable called **Profit**, which indeed is the daily profit.

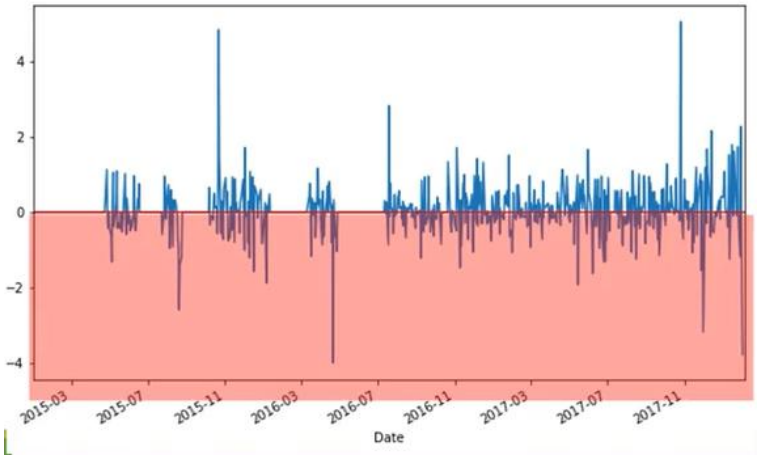
Daily profit

In [11]:

```
ms['Profit']=[ms.loc[ei,'Close1']-ms.loc[ei,'Close']
              if ms.loc[ei,'Shares']==1
              else 0 for ei in ms.index]
ms['Profit'].plot()
plt.axhline(y=0,color='red')
```



Make money



Lose money

We can compute the **cumulative wealth**.

Using DataFrame method, Cumsum to compute cumulative sum and create a new variable Wealth.

Then, we **check tail part of DataFrame**, and **check whether we make money or lose money**

Cumulative wealth

In [12]
```
ms['wealth']=ms['Profit'].cumsum()
ms.tail()
```

Out[12]

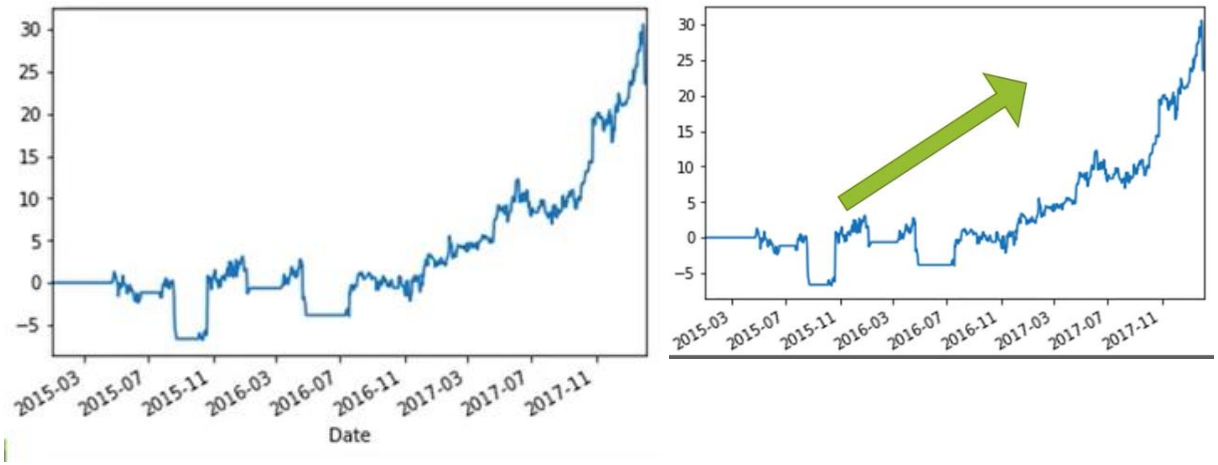
	Open	High	Low	Close	Adj Close	Volume	MA10	MA50	Shares	Close1	Profit	wealth
Date												
2018-01-30	93.300003	93.660004	92.099998	92.739998	92.739998	38635100	91.862	86.5244	1	95.010002	2.270004	30.540009
2018-01-31	93.750000	95.400002	93.510002	95.010002	95.010002	48756300	92.349	86.7606	1	94.260002	-0.750000	29.790009
2018-02-01	94.790001	96.070000	93.580002	94.260002	94.260002	47227900	92.765	86.9978	1	91.779999	-2.480003	27.310006
2018-02-02	93.639999	93.970001	91.500000	91.779999	91.779999	47867800	92.943	87.1828	1	88.000000	-3.779999	23.530007
2018-02-05	90.559998	93.239998	88.000000	88.000000	88.000000	50354600	92.582	87.2684	1	NaN	NaN	NaN

ms.index[-2]

Then, we print out wealth, which in fact is a sum of profit. To realize this profit, we need to buy one share of stock initially. So, our **initial investment is a stock price on the first day**.

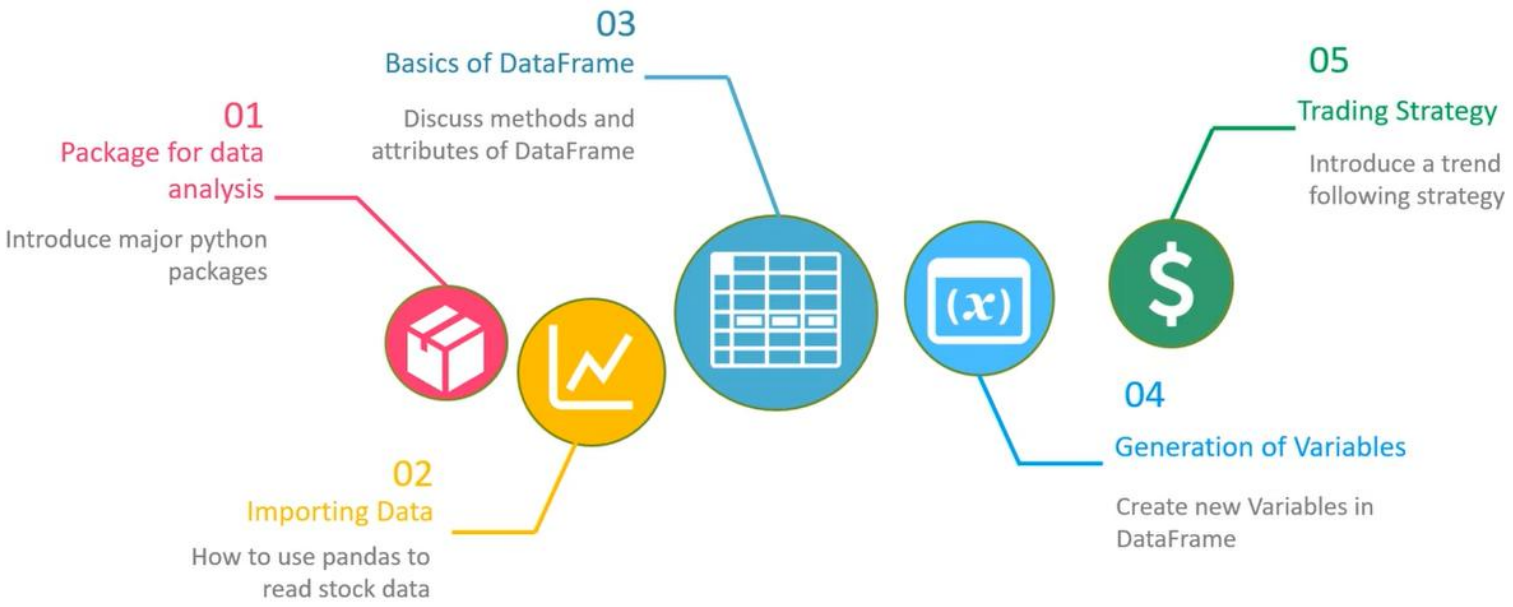
Cumulative wealth

```
In [12]  print("Total money you win is ", ms.loc[ms.index[-2],'wealth'])
         print("Total money you spent is ", ms.loc[ms.index[0],'Close'])
         ms['wealth'].plot()

Out[12]  Total money you win is 23.530007
         Total money you spent is 46.450001
```



This strategy seems to be very promising, **but** there are lots of questions if you want to implement it in real market.

Summary



You should know **how to read the data into DataFrame**, **explore basic attributes** and methods of DataFrame. Most importantly, know **how to create new variables** using column-wise transformation or list comprehension.

Lab 3: Build a simple trading strategy

Instructions

In this Jupyter Notebook, you are going to **mimic the steps illustrated** from the lecture in **creating your first trading strategy - Trend following trading strategy**.

Build a simple trading strategy

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Munging the stock data and add two columns - MA10 and MA50

In [4]:
```
#import FB's stock data, add two columns - MA10 and MA50
#use dropna to remove any "Not a Number" data
fb = pd.DataFrame.from_csv('../data/facebook.csv')
fb['MA10'] = fb['Close'].rolling(10).mean()
fb['MA50'] = fb['Close'].rolling(50).mean()
fb = fb.dropna()
fb.head()
```

Out[4]:

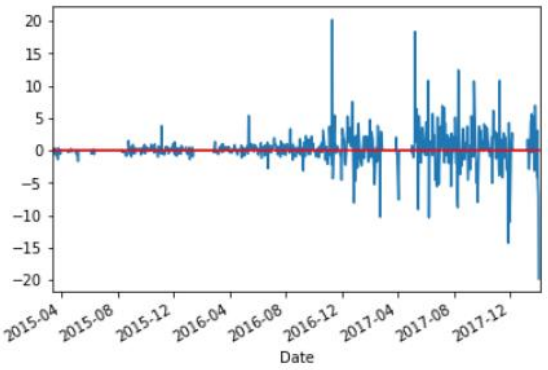
Date	Open	High	Low	Close	Adj Close	Volume	MA10	MA50
2015-03-13	22.559999	22.760000	22.250000	22.700001	22.116024	8982200	22.648	21.0174
2015-03-16	22.879999	23.110001	22.730000	22.969999	22.379078	5923900	22.685	21.0758
2015-03-17	22.920000	23.260000	22.760000	23.250000	22.651876	7497500	22.792	21.1382
2015-03-18	23.250000	23.370001	22.660000	22.870001	22.281652	10337600	22.836	21.1998
2015-03-19	22.950001	23.299999	22.780001	23.219999	22.622650	7768900	22.872	21.2804

2. Add "Shares" column to make decisions base on the strategy

In [6]:
```
#Add a new column "Shares", if MA10>MA50, denote as 1 (long one share of stock), otherwise, denote as 0 (do nothing)

fb['Shares'] = [1 if fb.loc[ei, 'MA10']>fb.loc[ei, 'MA50'] else 0 for ei in fb.index]
```

In [7]:
```
#Add a new column "Profit" using List Comprehension, for any rows in fb, if Shares=1, the profit is calculated as the close price of
#tomorrow - the close price of today. Otherwise the profit is 0.

#Plot a graph to show the Profit/Loss

fb['Close1'] = fb['Close'].shift(-1)
fb['Profit'] = [fb.loc[ei, 'Close1'] - fb.loc[ei, 'Close'] if fb.loc[ei, 'Shares']==1 else 0 for ei in fb.index]
fb['Profit'].plot()
plt.axhline(y=0, color='red')
```

Out[7]: <matplotlib.lines.Line2D at 0x7f0c99862128>



3. Use .cumsum() to display our model's performance if we follow the strategy

In [8]:
```
#Use .cumsum() to calculate the accumulated wealth over the period

fb['wealth'] = fb['Profit'].cumsum()
fb.tail()
```

Out[8]:

Date	Open	High	Low	Close	Adj Close	Volume	MA10	MA50	Shares	Close1	Profit	wealth
2018-01-30	241.110001	246.419998	238.410004	242.720001	242.720001	14270800	235.692003	210.030001	1	245.800003	3.080002	177.820011
2018-01-31	245.770004	249.270004	244.449997	245.800003	245.800003	11964400	237.800003	210.713801	1	240.500000	-5.300003	172.520008
2018-02-01	238.520004	246.899994	238.059998	240.500000	240.500000	12980600	239.406003	211.296601	1	233.520004	-6.979996	165.540012
2018-02-02	237.000000	237.970001	231.169998	233.520004	233.520004	17961600	239.747003	211.685401	1	213.699997	-19.820007	145.720005
2018-02-05	227.000000	233.229996	205.000000	213.699997	213.699997	28869000	237.748003	211.638401	1	NaN	NaN	NaN

In [12]:
```
#plot the wealth to show the growth of profit over the period

fb['wealth'].plot()
plt.title('Total money you win is {}'.format(fb.loc[fb.index[-2], 'wealth']))
```

Out[12]: <matplotlib.text.Text at 0x7f0c93b9e400>

You can create your own simple trading strategy by copying the codes above and modify the codes accordingly **using the data of Microsoft (microsoft.csv)**.

Data和CBuild a simple trading strategy.ipynb在Github中下载

https://github.com/cloudy-sfu/QUN-Data-Analysis-in-Finance/tree/main/Labs

Jupyternote Book课堂练习
二十分钟

Quiz 1

1. Which of the following library has DataFrame object?

☐ Pandas

☐ Numpy

☐ Matplotlib

☐ Statsmodels

Quiz 1

2. Which of the following is the correct way to import a library, eg Pandas?

☐ import pandas as pd

☐ #include <pandas>

☐ pandas import

☐ pandas

Quiz 1

3. What is the method of DataFrame object to import a csv file?

☐ import_csv()

☐ read_csv()

☐ sv()

☐ from_csv()

Quiz 1

4. Which of the following attributes of a DataFrame return a list of column names of this DataFrame?

☐ columns

☐ shape

☐ dtype

☐ column

Quiz 1

5. Which of the following can slice ‘Close’ from ‘2015-01-01’ to ‘2016-12-31’ from data, which is a DataFrame object?

☐ data.iloc[‘2015-01-01’:’2016-12-31’, ‘Close’]

☐ data.loc[‘2015-01-01’:’2016-12-31’, ‘Close’]

Quiz 1

6. What is the method of DataFrame to plot a line chart?

☐ plot_graph()

☐ scatter()

☐ plot()

☐ axhline()

Quiz 1

7. Suppose you have a DataFrame - data, which contains columns 'Open', 'High', 'Low', 'Close', 'Adj Close' and 'Volume'. What does data[['Open', 'Low']] return?

☐ Columns 'Open' and 'Low'

☐ No results are shown

☐ The first row of data which contains only columns 'Open' and 'High'

☐ All columns of data except 'Open' and 'High'

Quiz 1

8. Suppose you have a DataFrame ms , which contains the daily data of 'Open', 'High', 'Low', 'Close', 'AdjClose' and 'Volume' of Microsoft's stock.

Which of the following syntax calculates the Price difference, (ie 'Close' of tomorrow –'Close' of today)?

☐ ms['Close'].shift(1) – ms['Close'].shift(1)

☐ ms['Close'].shift(-1) – ms['Close']

☐ ms['Close'].shift(1) – ms['Close']

☐ ms['Close'].shift(-1) – ms['Close'].shift(-1)

Quiz 1

9. Suppose you have a DataFrame - ms , which contains the daily data of ‘Open’, ‘High’, ‘Low’, ‘Close’, ‘AdjClose’ and ‘Volumn’ of Microsoft's stock.

What is the method of DataFrame to calculate the 60 days moving average?

☐ rolling(60).median()

☐ rolling().mean(60)

☐ moving_average(60)

☐ rolling(60).mean()

Quiz 1

10. Which of the following idea(s) is/are correct to the simple trading strategy that we introduced in the lecture?

☐ If fast signal is larger than slow signal, this indicates an upward trend at the current moment

☐ We short one share of stocks if fast signal is larger than slow signal

☐ Use longer moving average as slow signal and shorter moving average as fast signal

Thank You