Tamkang University
淡江大學

Tamkang University

Social Computing and Big Data Analytics
社群運算與大數據分析
Finance Big Data Analytics with Pandas in Python
(Python Pandas 財務大數據分析)

1052SCBDA06
MIS MBA (M2226) (8606)
Wed, 8,9, (15:10-17:00) (L206)

Min-Yuh Day
戴敏育
Assistant Professor
專任助理教授
Dept. of Information Management, Tamkang University
淡江大學 資訊管理學系
http://mail. tku.edu.tw/myday/

2017-03-22

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
1	2017/02/15	Course Orientation for Social Computing and Big Data Analytics (社群運算與大數據分析課程介紹)
2	2017/02/22	Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data (資料科學與大數據分析：探索、分析、視覺化與呈現資料)
3	2017/03/01	Fundamental Big Data: MapReduce Paradigm, Hadoop and Spark Ecosystem (大數據基礎：MapReduce典範、Hadoop與Spark生態系統)

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
4	2017/03/08	Big Data Processing Platforms with SMACK: Spark, Mesos, Akka, Cassandra and Kafka (大數據處理平台SMACK：Spark, Mesos, Akka, Cassandra, Kafka)
5	2017/03/15	Big Data Analytics with Numpy in Python (Python Numpy 大數據分析)
6	2017/03/22	Finance Big Data Analytics with Pandas in Python (Python Pandas 財務大數據分析)
7	2017/03/29	Text Mining Techniques and Natural Language Processing (文字探勘分析技術與自然語言處理)
8	2017/04/05	Off-campus study (教學行政觀摩日)

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
9	2017/04/12	Social Media Marketing Analytics (社群媒體行銷分析)
10	2017/04/19	期中報告 (Midterm Project Report)
11	2017/04/26	Deep Learning with Theano and Keras in Python (Python Theano 和 Keras 深度學習)
12	2017/05/03	Deep Learning with Google TensorFlow (Google TensorFlow 深度學習)
13	2017/05/10	Sentiment Analysis on Social Media with Deep Learning (深度學習社群媒體情感分析)

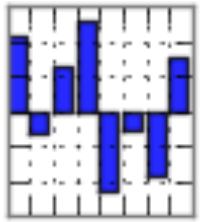
課程大綱 (Syllabus)

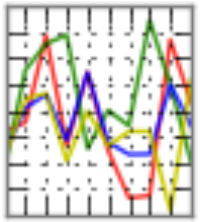
週次 (Week)	日期 (Date)	內容 (Subject/Topics)
14	2017/05/17	Social Network Analysis (社會網絡分析)
15	2017/05/24	Measurements of Social Network (社會網絡量測)
16	2017/05/31	Tools of Social Network Analysis (社會網絡分析工具)
17	2017/06/07	Final Project Presentation I (期末報告 I)
18	2017/06/14	Final Project Presentation II (期末報告 II)

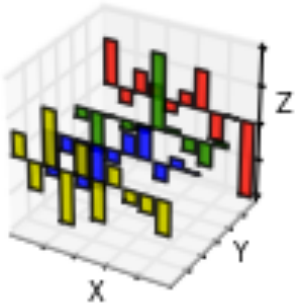
python™

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

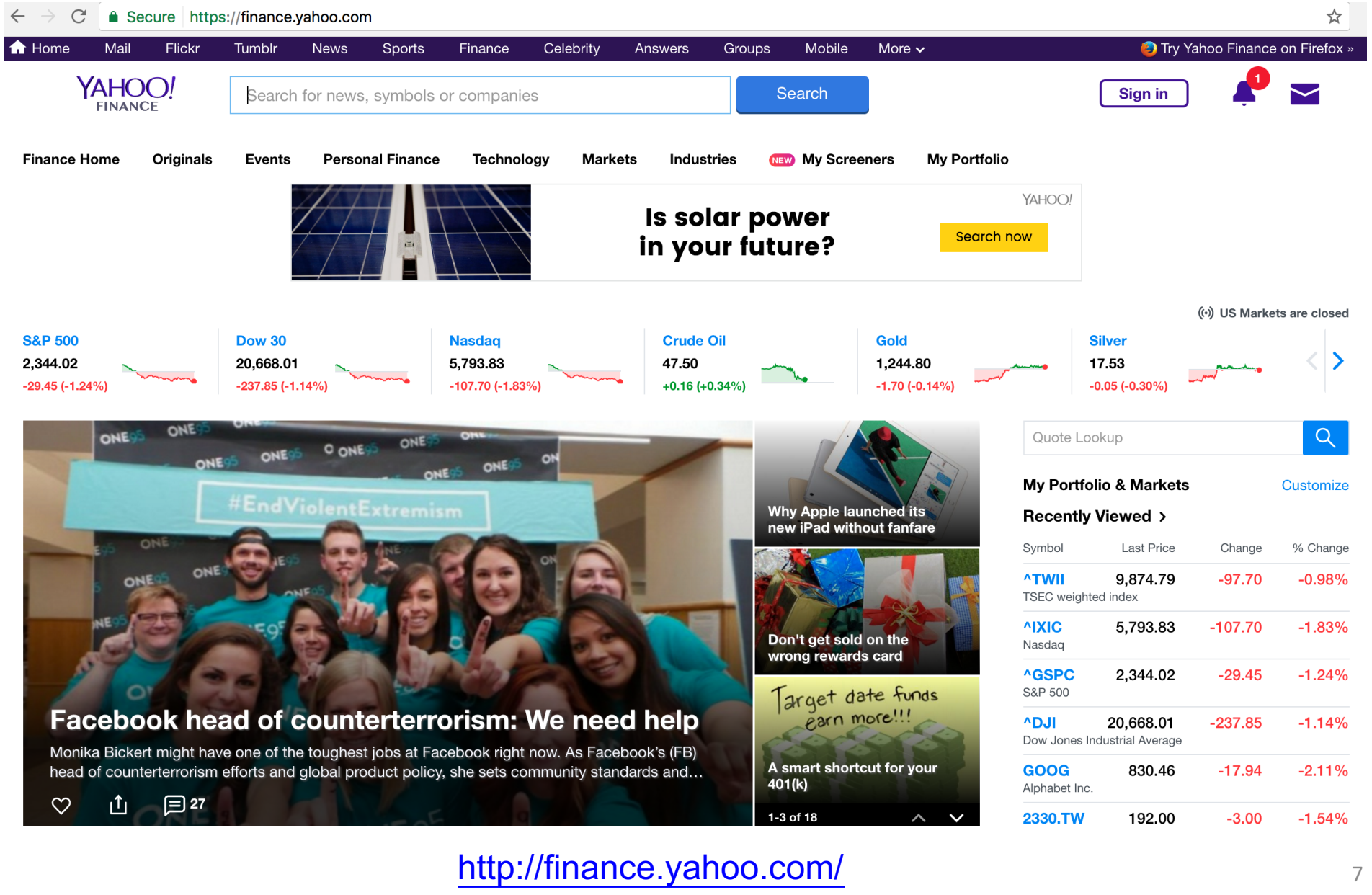






Source: https://www.python.org/community/logos/
http://pandas.pydata.org/

Yahoo Finance



http://finance.yahoo.com/

Apple Inc. (AAPL) -NasdaqGS

YAHOO! FINANCE

Search for news, symbols or companies | Search

Apple Inc. (AAPL)

NasdaqGS - NasdaqGS Delayed Price. Currency in USD

Add to watchlist

139.84 -1.62 (-1.15%)
At close: 4:00PM EDT

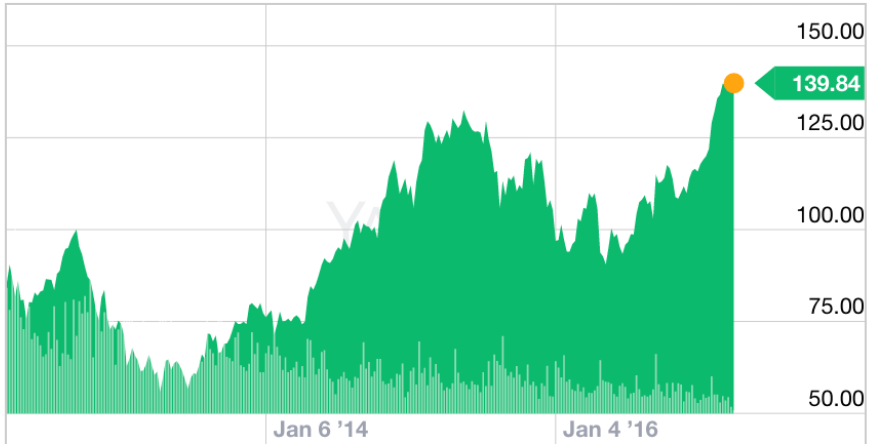
139.35 -0.49 (-0.35%)
After hours: 7:59PM EDT

Summary | Conversations | Statistics | Profile | Financials | Options | Holders | Historical Data | Analysts

Previous Close	141.46	Market Cap	733.68B
Open	142.11	Beta	1.45
Bid	139.31 x 100	PE Ratio (TTM)	16.79
Ask	139.40 x 1300	EPS (TTM)	8.33
Day's Range	139.73 - 142.80	Earnings Date	Apr 24, 2017 - Apr 28, 2017
52 Week Range	89.47 - 142.80	Dividend & Yield	2.28 (1.63%)
Volume	39,529,912	Ex-Dividend Date	N/A
Avg. Volume	26,889,183	1y Target Est	143.29

Trade prices are not sourced from all markets

1D 5D 1M 6M YTD 1Y 2Y **5Y** 10Y MAX | Interactive chart



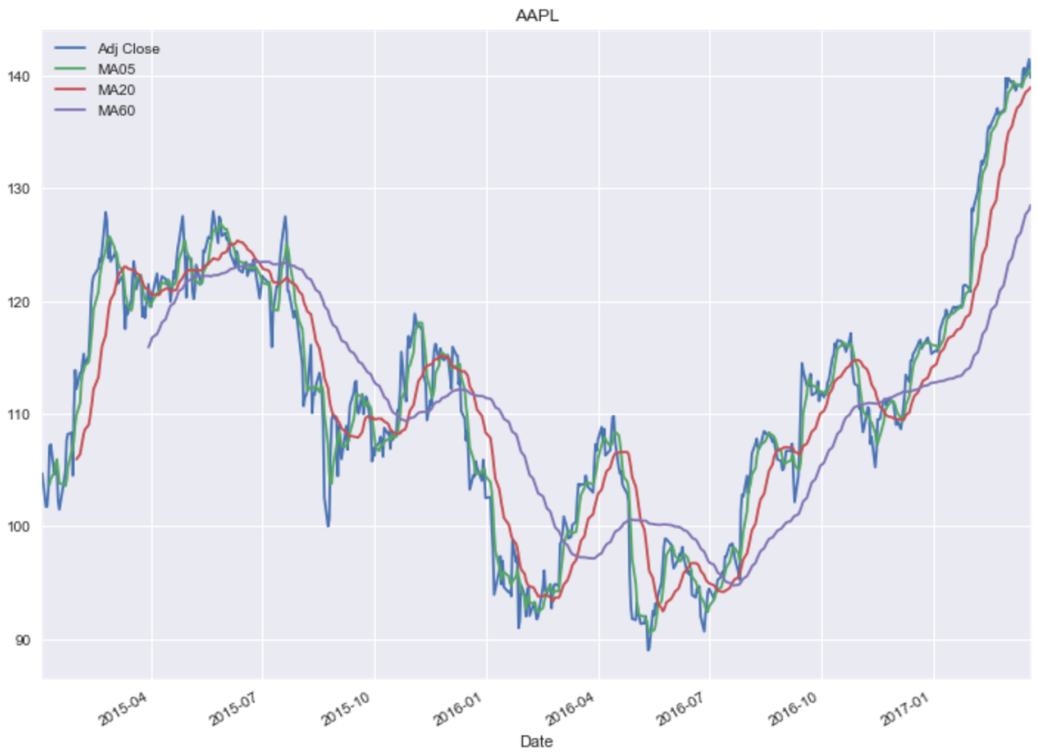
http://finance.yahoo.com/quote/AAPL?p=AAPL

Yahoo Finance Charts: Apple Inc. (AAPL)

http://finance.yahoo.com/chart/AAPL

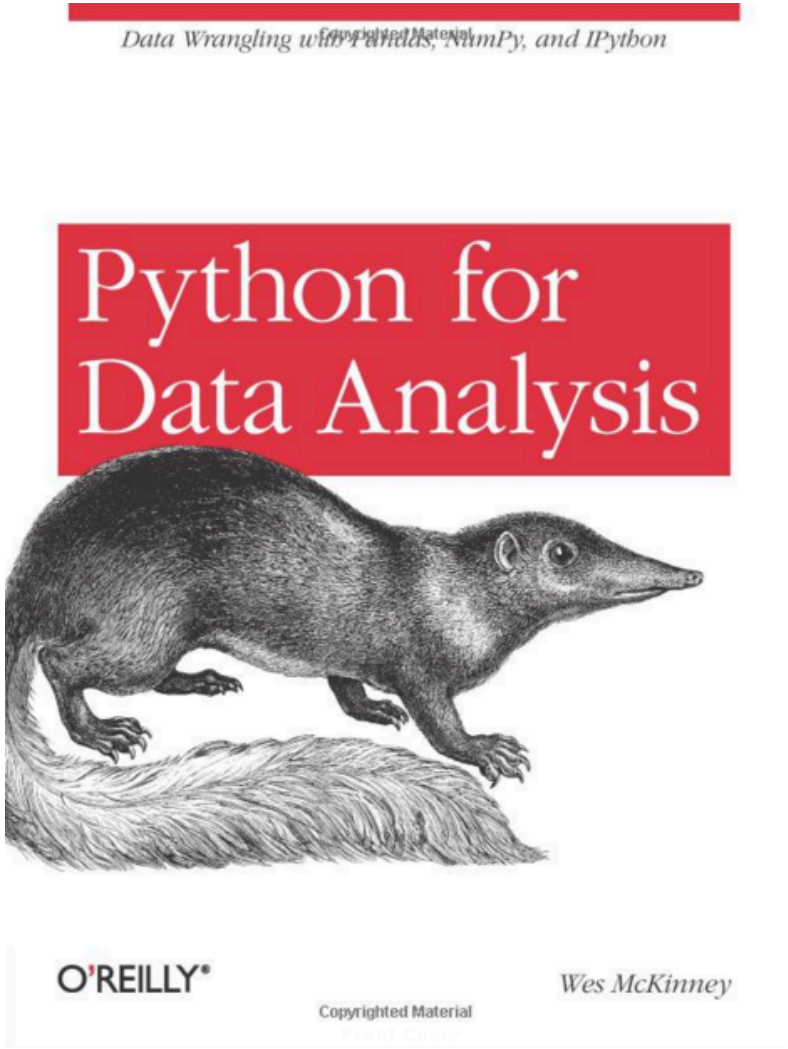
Python Pandas for Finance

```
# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days

df2 = pd.DataFrame({'Adj Close': df['Adj Close'],'MA05': df['MA05'],'MA20': df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
```



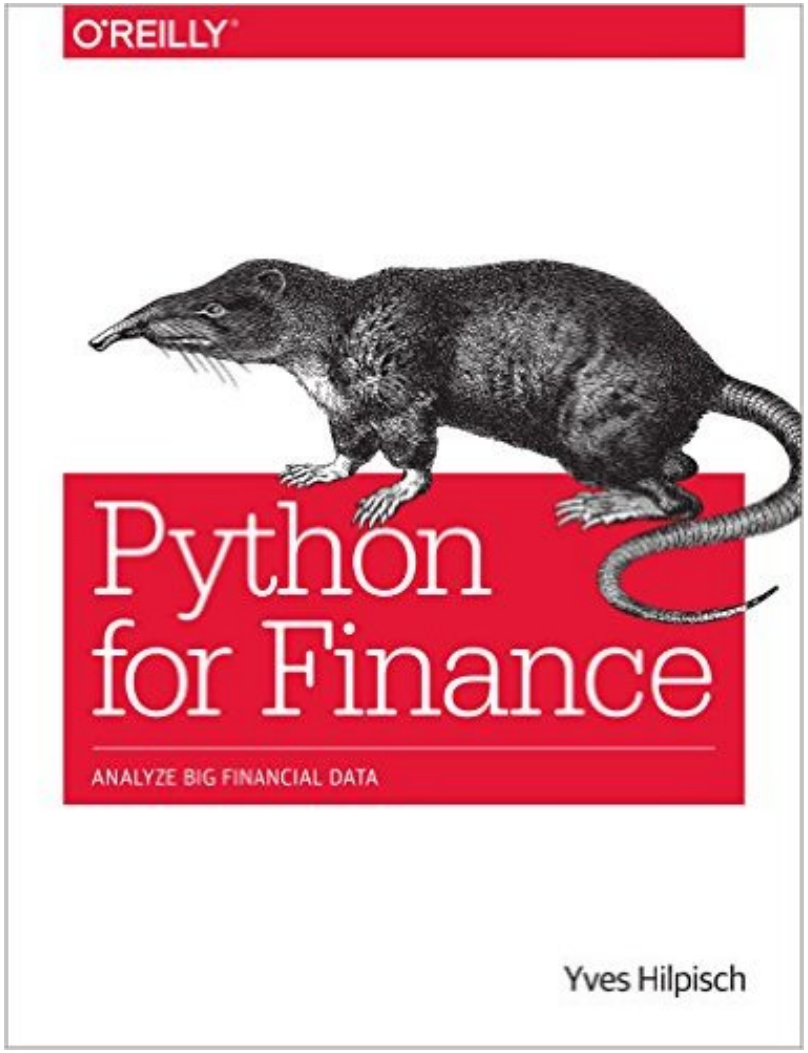
Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Wes McKinney (2012), Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media



Source: http://www.amazon.com/Python-Data-Analysis-Wrangling-IPython/dp/1449319793/

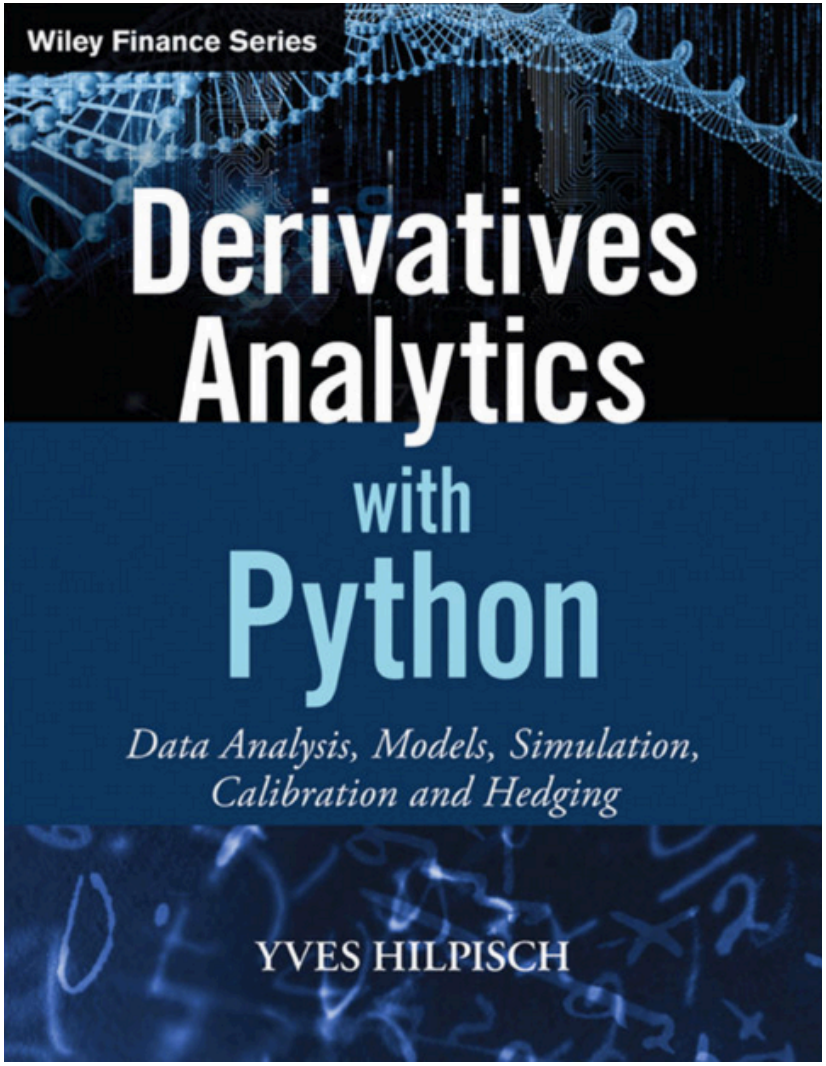
Yves Hilpisch, Python for Finance: Analyze Big Financial Data, O'Reilly, 2014



Source: http://www.amazon.com/Python-Finance-Analyze-Financial-Data/dp/1491945281

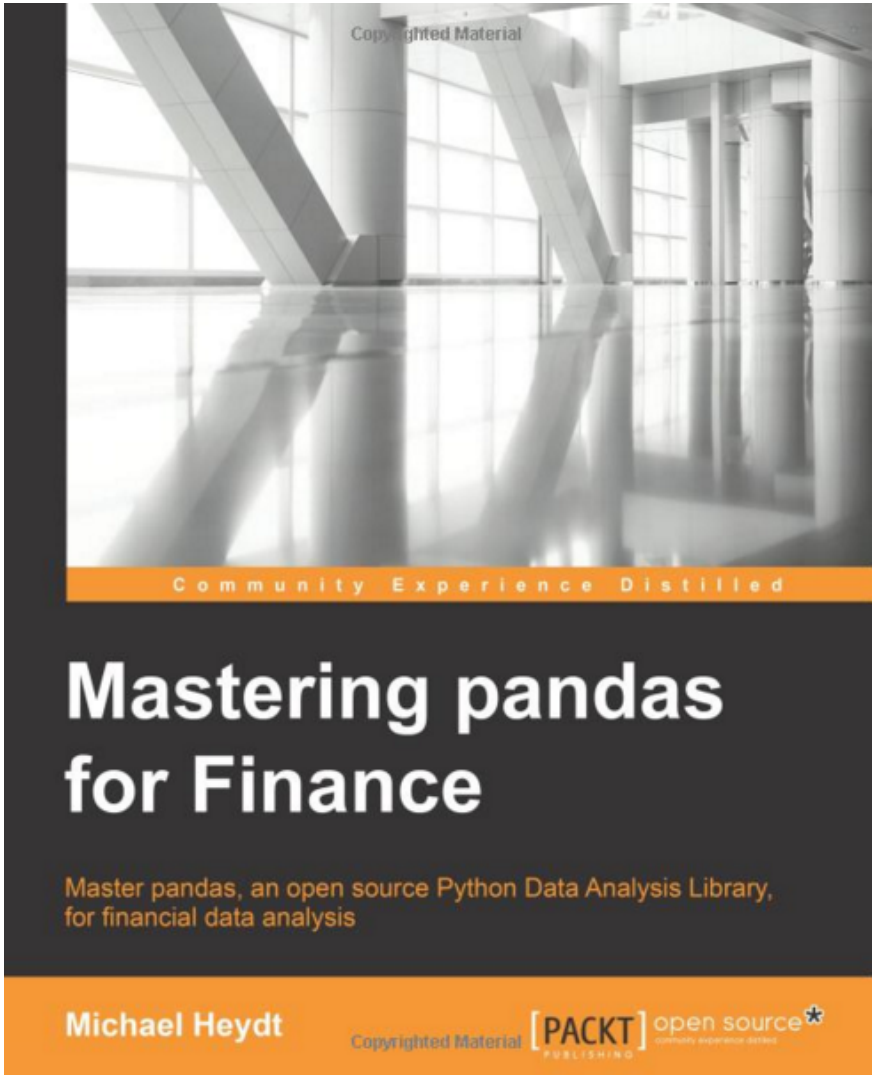
Yves Hilpisch (2015), Derivatives Analytics with Python:

Data Analysis, Models, Simulation, Calibration and Hedging, Wiley



Source: http://www.amazon.com/Derivatives-Analytics-Python-Simulation-Calibration/dp/1119037999/

Michael Heydt ,
Mastering Pandas for Finance,
Packt Publishing, 2015



Source: http://www.amazon.com/Mastering-Pandas-Finance-Michael-Heydt/dp/1783985100

Anaconda

ANACONDA
Powered by Continuum Analytics

Log In Get Support Search Contact

PRODUCTS COMMUNITY CONSULTING TRAINING ABOUT RESOURCES

ACCELERATE

Time-to-Value

CONNECT

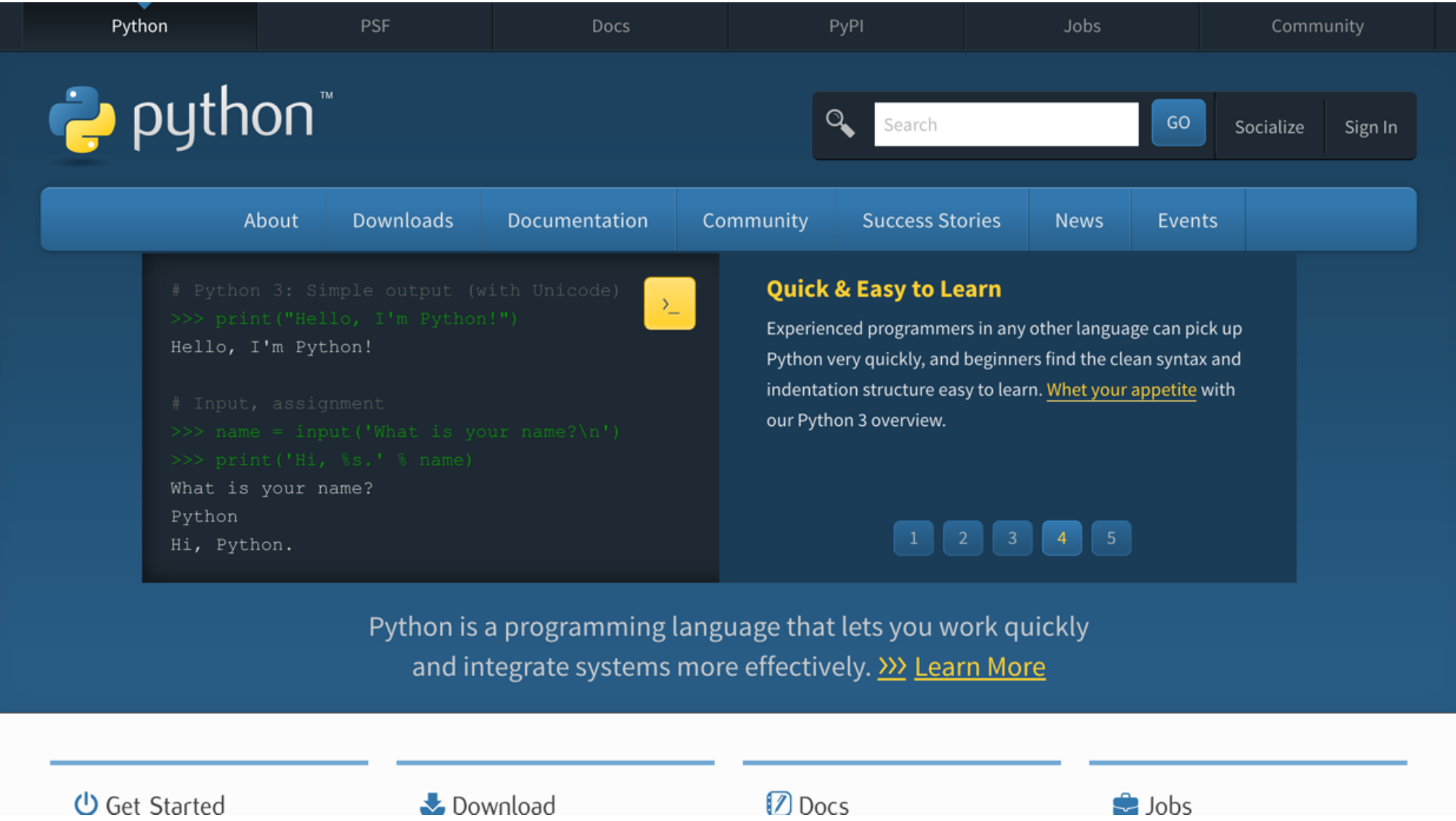
Data, Analytics & Compute

EMPOWER

Data Science Teams

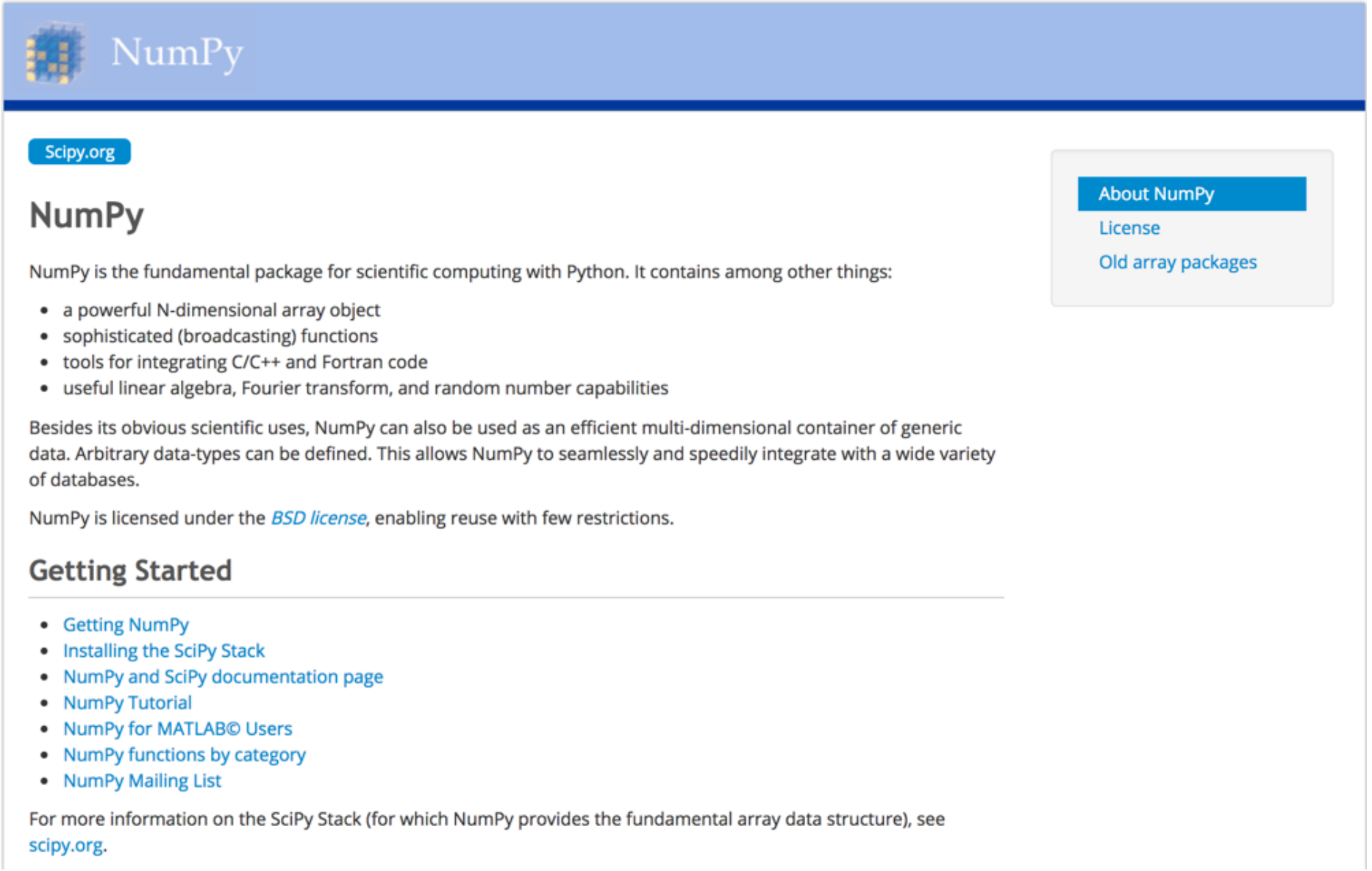
https://www.continuum.io/

Python



https://www.python.org/

NumPy

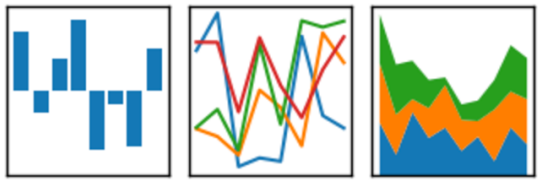


http://www.numpy.org/

pandas

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project.

A Fiscally Sponsored Project of
NUMFOCUS
OPEN CODE = BETTER SCIENCE

0.19.2 Final (December 24, 2016)

This is a minor bug-fix release in the 0.19.x series and includes some small regression fixes, bug fixes and performance improvements.

Highlights include:

- Compatibility with Python 3.6

VERSIONS

Release
0.19.2 - December 2016
download // docs // pdf

Development
0.20.0 - 2017
github // docs

Previous Releases
0.19.1 - download // docs // pdf
0.19.0 - download // docs // pdf
0.18.1 - download // docs // pdf
0.18.0 - download // docs // pdf
0.17.1 - download // docs // pdf
0.17.0 - download // docs // pdf
0.16.2 - download // docs // pdf
0.16.1 - download // docs // pdf
0.16.0 - download // docs // pdf
0.15.2 - download // docs // pdf
0.15.1 - download // docs // pdf
0.15.0 - download // docs // pdf
0.14.1 - download // docs // pdf

http://pandas.pydata.org/

pandas

Python Data Analysis Library

providing high-performance, easy-to-use
data structures and data analysis tools
for the Python programming language.

Source: http://pandas.pydata.org/

pandas Ecosystem

- Statistics and Machine Learning
  - Statsmodels
  - sklearn-pandas
- Visualization
  - Bokeh
  - yhat/ggplot
  - Seaborn
  - Vincent
  - IPython Vega
  - Plotly
  - Pandas-Qt
- IDE
  - IPython
  - quantopian/qgrid
  - Spyder
- API
  - pandas-datareader
  - quandl/Python
  - pydatastream
  - pandaSDMX
  - fredapi
- Domain Specific
  - Geopandas
  - xarray
- Out-of-core
  - Dask
  - Blaze
  - Odo

Source: http://pandas.pydata.org/pandas-docs/stable/index.html

pandas: powerful Python data analysis toolkit

pandas 0.19.2 documentation »

Table Of Contents

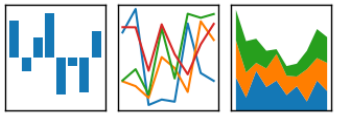
pandas: powerful Python data analysis toolkit

PDF Version

Zipped HTML

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Date: Dec 24, 2016 **Version**: 0.19.2

Binary Installers: http://pypi.python.org/pypi/pandas

Source Repository: http://github.com/pydata/pandas

Issues & Ideas: https://github.com/pydata/pandas/issues

Q&A Support: http://stackoverflow.com/questions/tagged/pandas

Developer Mailing List: http://groups.google.com/group/pydata

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

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 actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, `Series` (1-dimensional) and `DataFrame` (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, `DataFrame` provides everything that R's `data.frame` provides and much more. pandas is built on top of NumPy and is

http://pandas.pydata.org/pandas-docs/stable/

pandas:
powerful Python data analysis toolkit

- 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 actually need not be labeled at all to be placed into a pandas data structure

Source: http://pandas.pydata.org/pandas-docs/stable/

Series
DataFrame

- Primary data structures of pandas
  - Series (1-dimensional)
  - DataFrame (2-dimensional)
- Handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

Source: http://pandas.pydata.org/pandas-docs/stable/

pandas DataFrame

- DataFrame provides everything that R's data.frame provides and much more.
- pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

pandas
Comparison with SAS

pandas	SAS
DataFrame	data set
column	variable
row	observation
groupby	BY-group
NaN	.

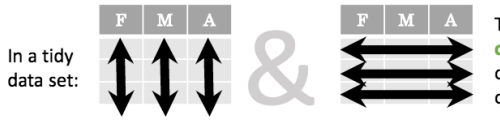
Source: http://pandas.pydata.org/pandas-docs/stable/comparison_with_sas.html

Python Pandas Cheat Sheet

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

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

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```

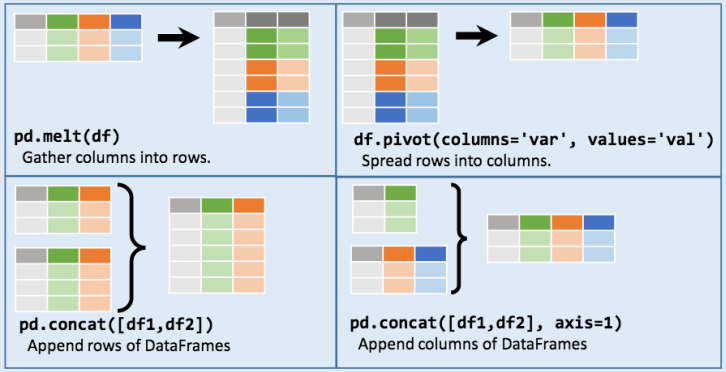
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set



pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df=df.sort_values('mpg')
Order rows by values of a column (low to high).

df=df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

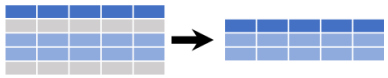
df=df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df=df.sort_index()
Sort the index of a DataFrame

df=df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df=df.drop(['Length','Height'], axis=1)
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

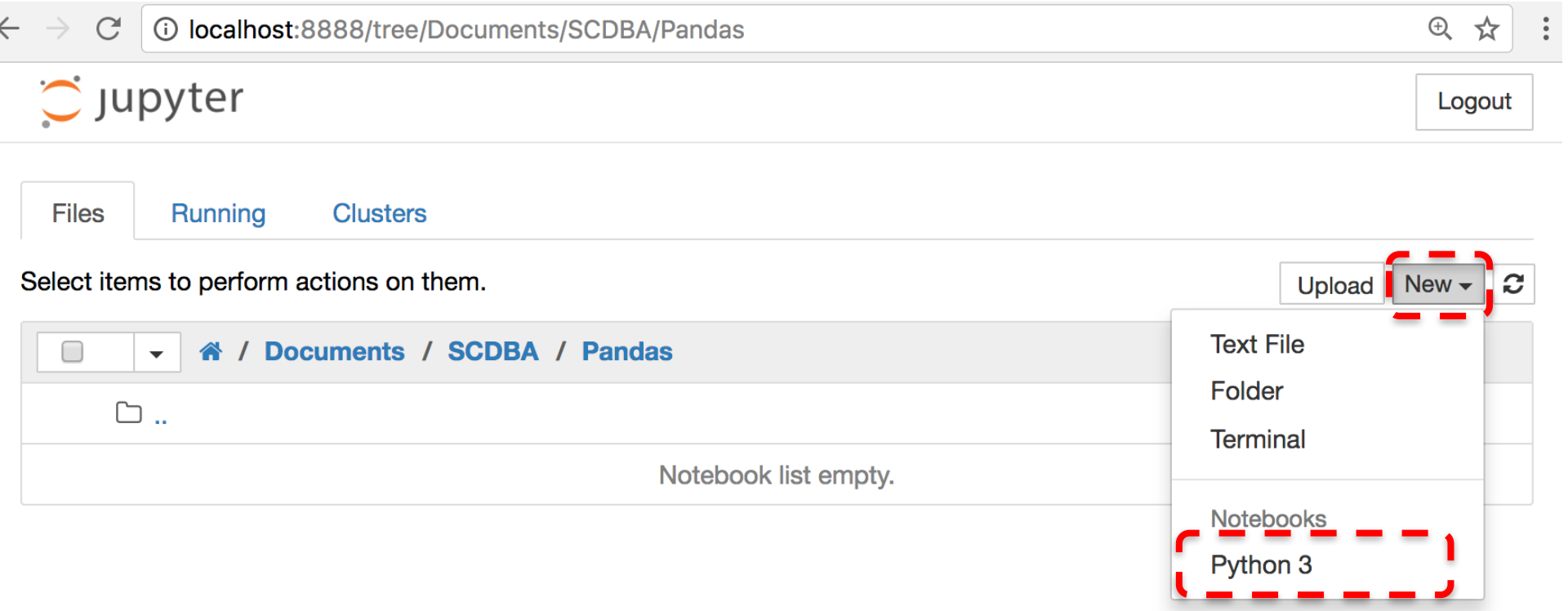
df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Source: https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

Jupyter Notebook New Python 3



Creating pd.DataFrame

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

In [1]:
```
import numpy as np
import pandas as pd
df = pd.DataFrame({"a": [4, 5, 6],
                   "b": [7, 8, 9],
                   "c": [10, 11, 12]},
                  index = [1, 2, 3])
df
```

Out[1]:

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

```
df = pd.DataFrame({"a": [4, 5, 6],
                   "b": [7, 8, 9],
                   "c": [10, 11, 12]},
                  index = [1, 2, 3])
```

Source: https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

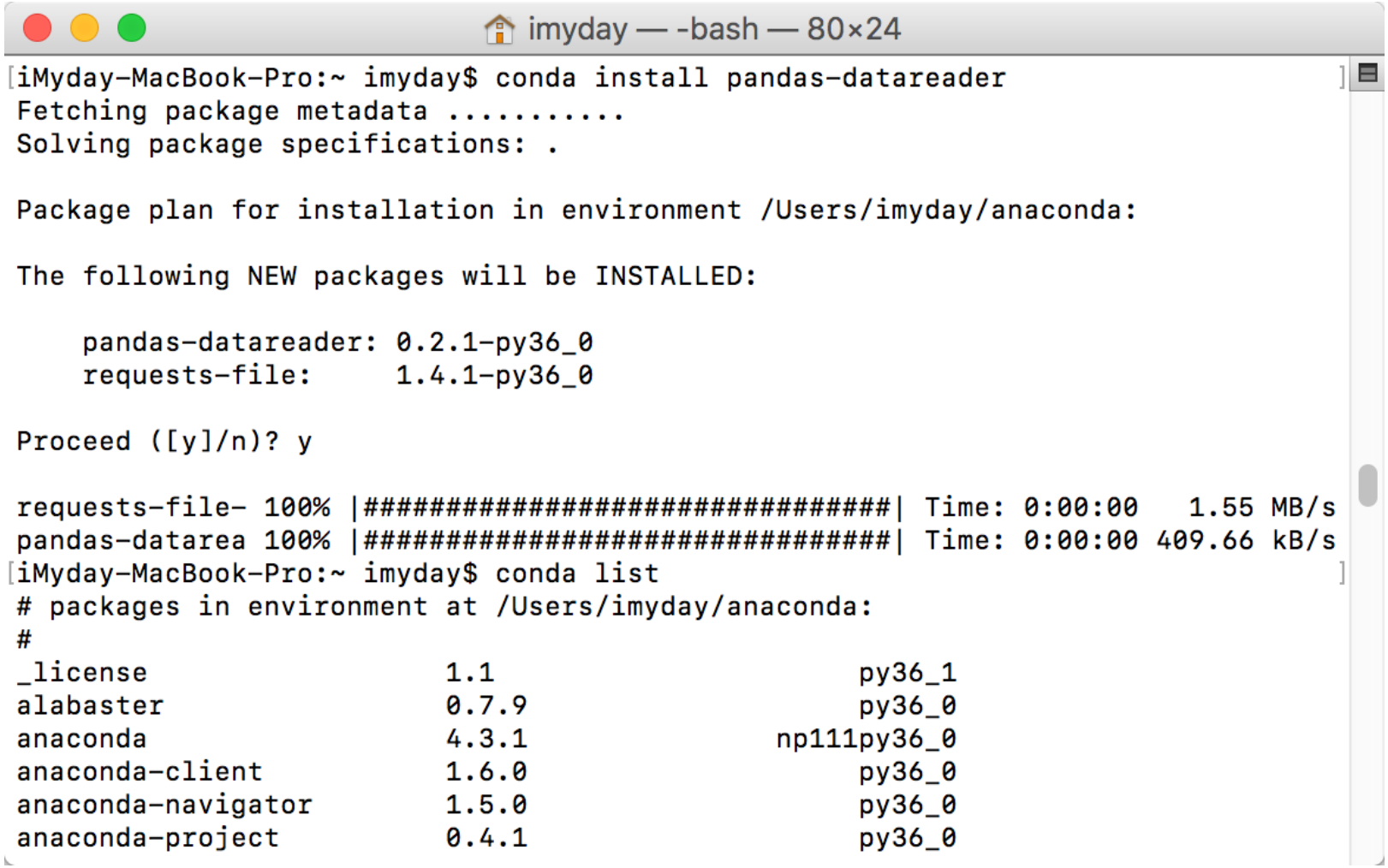
Pandas DataFrame

```
type(df)
```

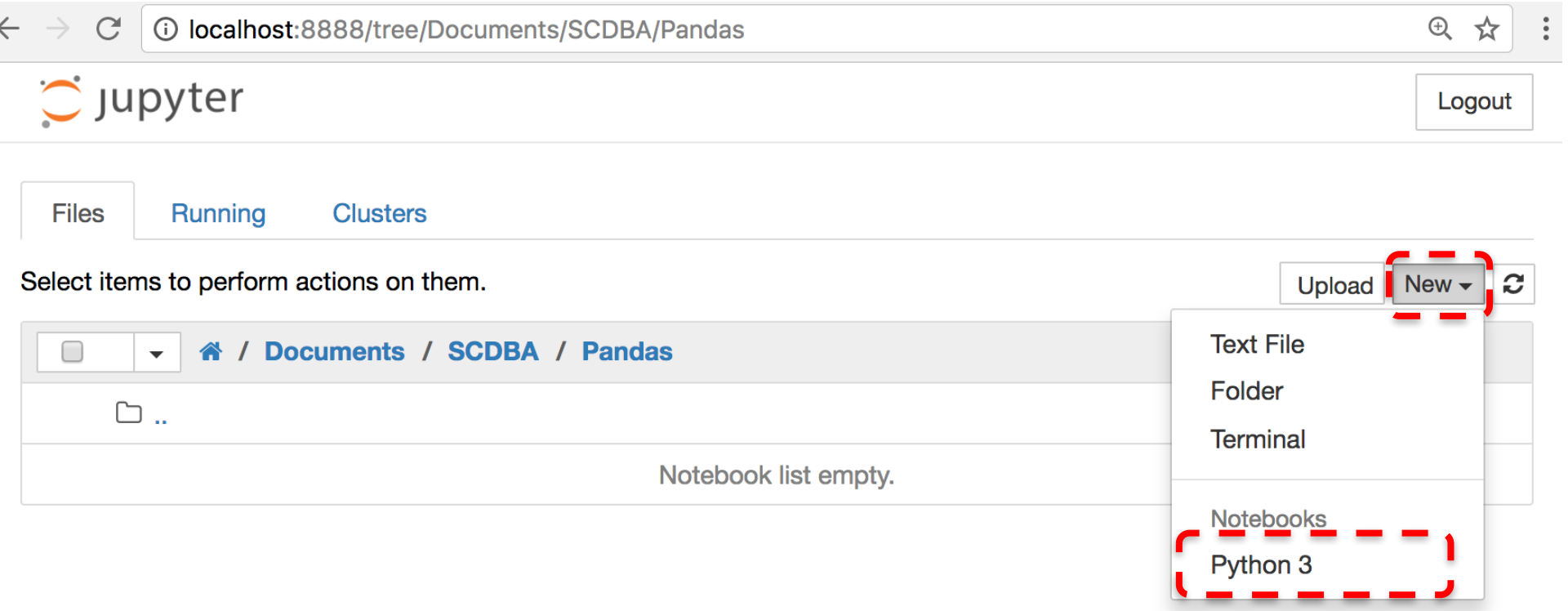
```
type(df)
```

```
pandas.core.frame.DataFrame
```

conda install pandas-datareader



Jupyter Notebook New Python 3



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
print('Hello Pandas')
```

```
s = pd.Series([1,3,5,np.nan,6,8])
s
```

```
dates = pd.date_range('20170301',
periods=6)
dates
```

Source: http://pandas.pydata.org/pandas-docs/stable/10min.html

localhost:8888/notebooks/Documents/SCDBA/python/PythonPandasFinance.ipynb

jupyter PythonPandasFinance (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Python 3

Code CellToolbar

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
print('Hello Pandas')
```

```
Hello Pandas
```

In [2]:
```
s = pd.Series([1,3,5,np.nan,6,8])
s
```

Out[2]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

In [3]:
```
dates = pd.date_range('20170301', periods=6)
dates
```

Out[3]:
```
DatetimeIndex(['2017-03-01', '2017-03-02', '2017-03-03', '2017-03-04',
               '2017-03-05', '2017-03-06'],
              dtype='datetime64[ns]', freq='D')
```

```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

	A	B	C	D
2017-03-01	-1.613934	0.168983	0.686730	1.520077
2017-03-02	0.656124	-1.155197	-1.969431	0.444569
2017-03-03	-0.242393	-0.372972	-0.310283	1.102838
2017-03-04	-0.034627	-0.422640	0.489834	1.498832
2017-03-05	-1.022516	-0.841598	-0.136151	-0.767695
2017-03-06	1.208003	3.114586	-0.451570	0.579479

```
df = pd.DataFrame(np.random.randn(4,6), index=['student1','student2','student3','student4'], columns=list('ABCDEF'))
df
```

	A	B	C	D	E	F
student1	-0.420406	0.829262	-0.326521	-0.037699	-1.350555	-0.617676
student2	0.310825	-0.356479	0.149704	-0.685609	-0.744307	-0.488782
student3	-1.295312	0.765656	1.701502	-0.415809	-0.454114	0.397702
student4	0.979525	0.367767	1.869465	0.988012	0.916746	0.165911

```
df2 = pd.DataFrame({ 'A' : 1.,
'B' : pd.Timestamp('20170322'),
'C' : pd.Series(2.5,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' })
df2
```

```
df2 = pd.DataFrame({ 'A' : 1.,
'B' : pd.Timestamp('20170322'),
'C' : pd.Series(2.5,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' })
df2
```

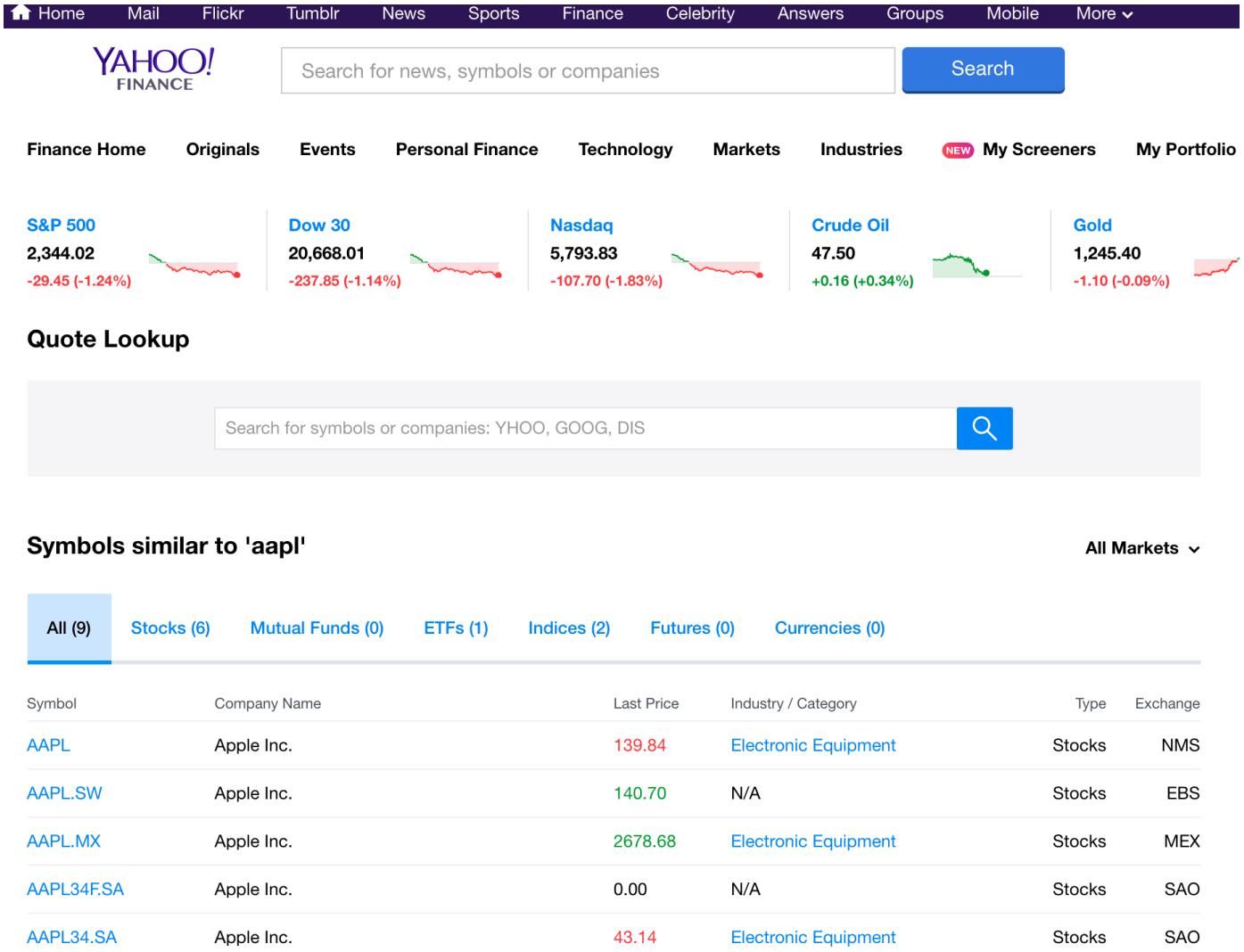
	A	B	C	D	E	F
0	1.0	2017-03-22	2.5	3	test	foo
1	1.0	2017-03-22	2.5	3	train	foo
2	1.0	2017-03-22	2.5	3	test	foo
3	1.0	2017-03-22	2.5	3	train	foo

df2.dtypes

```
df2.dtypes
```

```
A              float64
B    datetime64[ns]
C              float32
D                int32
E             category
F               object
dtype: object
```

Yahoo Finance Symbols: AAPL Apple Inc. (AAPL)



http://finance.yahoo.com/q?s=AAPL

Apple Inc. (AAPL) -NasdaqGS

YAHOO! FINANCE

Search for news, symbols or companies | Search

Apple Inc. (AAPL)

NasdaqGS - NasdaqGS Delayed Price. Currency in USD

Add to watchlist

139.84 -1.62 (-1.15%)
At close: 4:00PM EDT

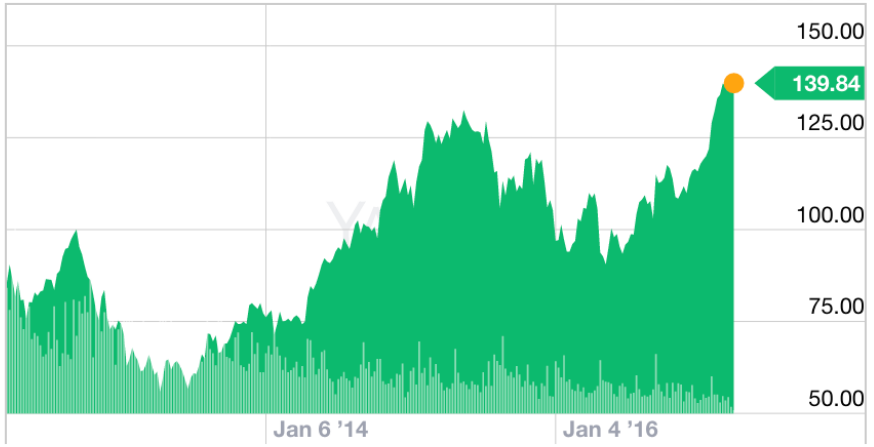
139.35 -0.49 (-0.35%)
After hours: 7:59PM EDT

Summary | Conversations | Statistics | Profile | Financials | Options | Holders | Historical Data | Analysts

Previous Close	141.46	Market Cap	733.68B
Open	142.11	Beta	1.45
Bid	139.31 x 100	PE Ratio (TTM)	16.79
Ask	139.40 x 1300	EPS (TTM)	8.33
Day's Range	139.73 - 142.80	Earnings Date	Apr 24, 2017 - Apr 28, 2017
52 Week Range	89.47 - 142.80	Dividend & Yield	2.28 (1.63%)
Volume	39,529,912	Ex-Dividend Date	N/A
Avg. Volume	26,889,183	1y Target Est	143.29

Trade prices are not sourced from all markets

1D 5D 1M 6M YTD 1Y 2Y **5Y** 10Y MAX | Interactive chart

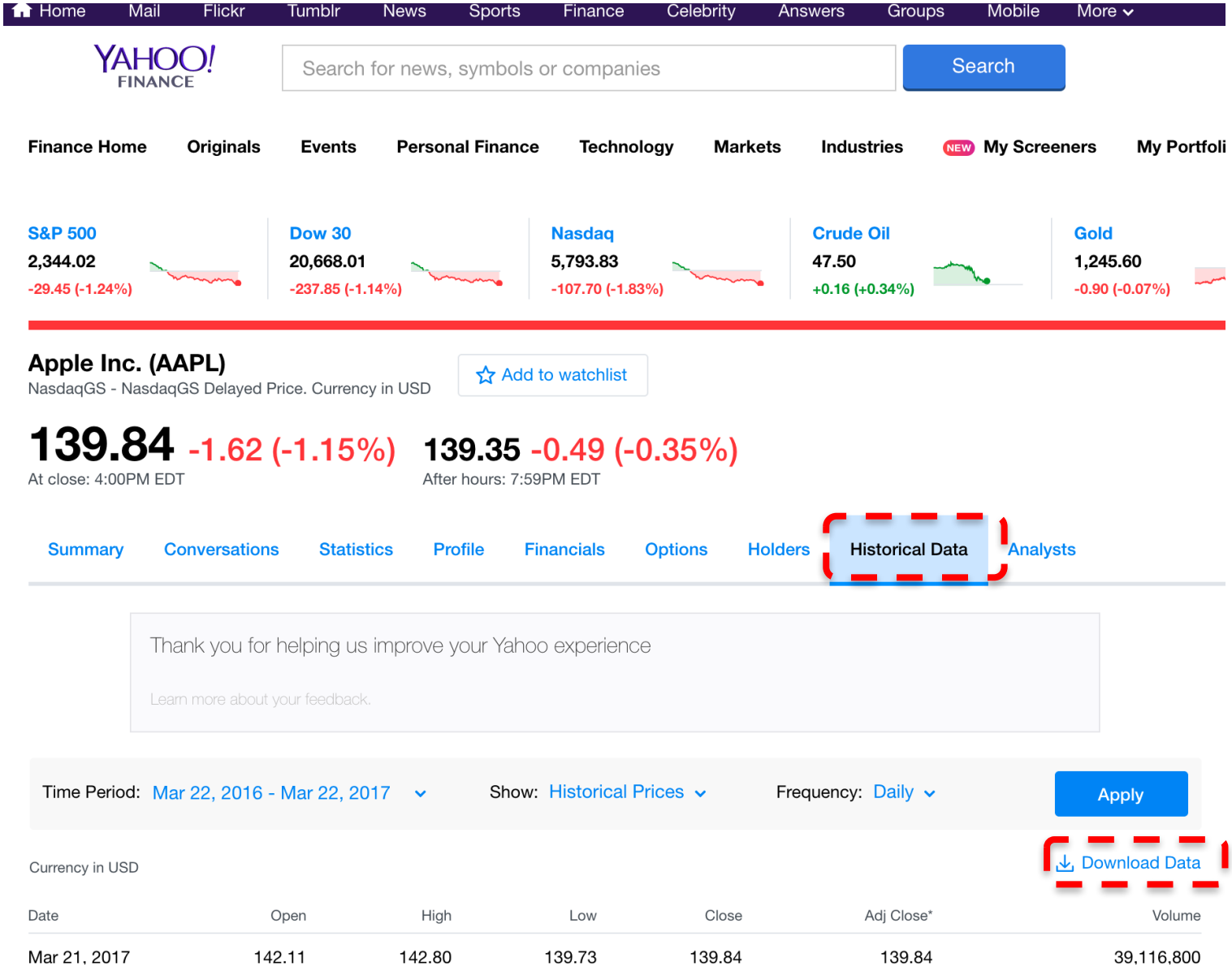


http://finance.yahoo.com/quote/AAPL?p=AAPL

Yahoo Finance Charts: Apple Inc. (AAPL)

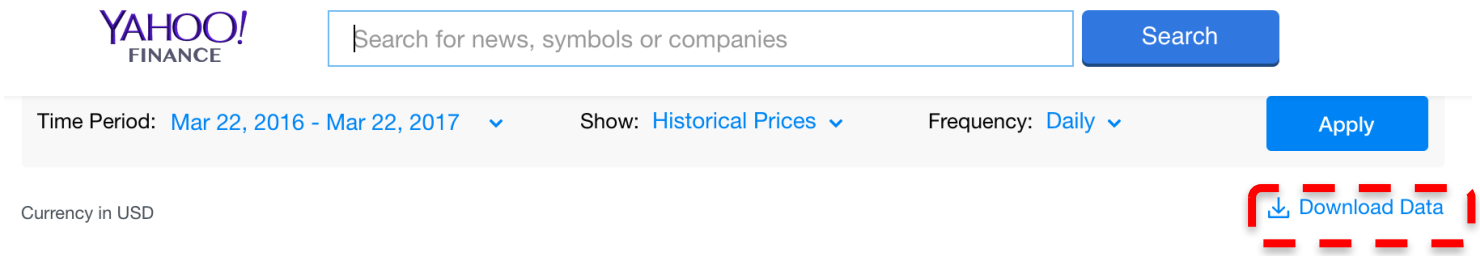
http://finance.yahoo.com/chart/AAPL

Apple Inc. (AAPL) Historical Data



http://finance.yahoo.com/q/hp?s=AAPL+Historical+Prices

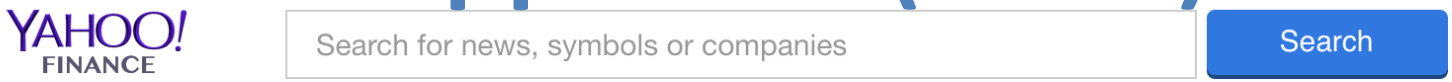
Yahoo Finance Historical Prices Apple Inc. (AAPL)



Date	Open	High	Low	Close	Adj Close*	Volume
Mar 21, 2017	142.11	142.80	139.73	139.84	139.84	39,116,800
Mar 20, 2017	140.40	141.50	140.23	141.46	141.46	21,542,000
Mar 17, 2017	141.00	141.00	139.89	139.99	139.99	43,885,000
Mar 16, 2017	140.72	141.02	140.26	140.69	140.69	19,232,000
Mar 15, 2017	139.41	140.75	139.03	140.46	140.46	25,691,800
Mar 14, 2017	139.30	139.65	138.84	138.99	138.99	15,309,100
Mar 13, 2017	138.85	139.43	138.82	139.20	139.20	17,421,700
Mar 10, 2017	139.25	139.36	138.64	139.14	139.14	19,612,800
Mar 09, 2017	138.74	138.79	137.05	138.68	138.68	22,155,900
Mar 08, 2017	138.95	139.80	138.82	139.00	139.00	18,707,200
Mar 07, 2017	139.06	139.98	138.79	139.52	139.52	17,446,300
Mar 06, 2017	139.37	139.77	138.60	139.34	139.34	21,750,000
Mar 03, 2017	138.78	139.83	138.59	139.78	139.78	21,108,100

http://finance.yahoo.com/quote/AAPL/history

Yahoo Finance Historical Prices
Apple Inc. (AAPL)



Time Period: Dec 12, 1980 - Mar 22, 2017 Show: Historical Prices Frequency: Daily Apply

1D 5D 3M 6M
YTD 1Y 5Y MAX

Start Date 12/12/1980 End Date 3/22/2017

Done Cancel

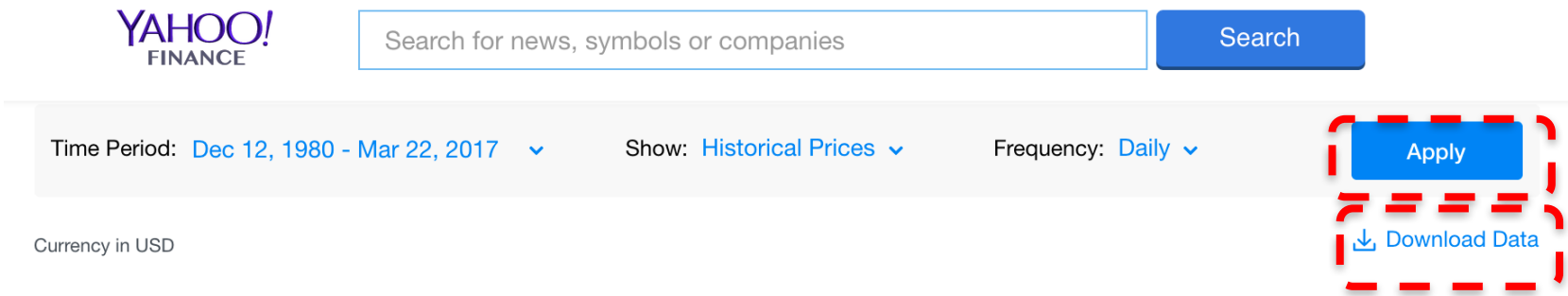
Currency i

Download Data

Date	Open	High	Low	Close	Adj Close*	Volume
Mar 21, 2		42.80	139.73	139.84	139.84	39,116,800
Mar 20, 2		41.50	140.23	141.46	141.46	21,542,000
Mar 17, 2		41.00	139.89	139.99	139.99	43,885,000
Mar 16, 2017	140.72	141.02	140.26	140.69	140.69	19,232,000
Mar 15, 2017	139.41	140.75	139.03	140.46	140.46	25,691,800
Mar 14, 2017	139.30	139.65	138.84	138.99	138.99	15,309,100
Mar 13, 2017	138.85	139.43	138.82	139.20	139.20	17,421,700
Mar 10, 2017	139.25	139.36	138.64	139.14	139.14	19,612,800
Mar 09, 2017	138.74	138.79	137.05	138.68	138.68	22,155,900
Mar 08, 2017	138.95	139.80	138.82	139.00	139.00	18,707,200

http://finance.yahoo.com/quote/AAPL/history?period1=345398400&period2=1490112000&interval=1d&filter=history&frequency=1d

Yahoo Finance Historical Prices
Apple Inc. (AAPL)



Currency in USD

Date	Open	High	Low	Close	Adj Close*	Volume
Mar 21, 2017	142.11	142.80	139.73	139.84	139.84	39,116,800
Mar 20, 2017	140.40	141.50	140.23	141.46	141.46	21,542,000
Mar 17, 2017	141.00	141.00	139.89	139.99	139.99	43,885,000
Mar 16, 2017	140.72	141.02	140.26	140.69	140.69	19,232,000
Mar 15, 2017	139.41	140.75	139.03	140.46	140.46	25,691,800
Mar 14, 2017	139.30	139.65	138.84	138.99	138.99	15,309,100
Mar 13, 2017	138.85	139.43	138.82	139.20	139.20	17,421,700
Mar 10, 2017	139.25	139.36	138.64	139.14	139.14	19,612,800
Mar 09, 2017	138.74	138.79	137.05	138.68	138.68	22,155,900
Mar 08, 2017	138.95	139.80	138.82	139.00	139.00	18,707,200

http://finance.yahoo.com/quote/AAPL/history?period1=345398400&period2=1490112000&interval=1d&filter=history&frequency=1d

Yahoo Finance Historical Prices

http://ichart.finance.yahoo.com/table.csv?s=AAPL

table.csv

```
Date,Open,High,Low,Close,Volume,Adj Close
2017-03-21,142.110001,142.800003,139.729996,139.839996,39116800,139.839996
2017-03-20,140.399994,141.50,140.229996,141.460007,20213100,141.460007
2017-03-17,141.00,141.00,139.889999,139.990005,43597400,139.990005
2017-03-16,140.720001,141.020004,140.259995,140.690002,19132500,140.690002
2017-03-15,139.410004,140.75,139.029999,140.460007,25566800,140.460007
2017-03-14,139.300003,139.649994,138.839996,138.990005,15189700,138.990005
2017-03-13,138.850006,139.429993,138.820007,139.199997,17042400,139.199997
2017-03-10,139.25,139.360001,138.639999,139.139999,19488000,139.139999
2017-03-09,138.740005,138.789993,137.050003,138.679993,22065200,138.679993
2017-03-08,138.949997,139.800003,138.820007,139.00,18681800,139.00
2017-03-07,139.059998,139.979996,138.789993,139.520004,17267500,139.520004
2017-03-06,139.369995,139.770004,138.600006,139.339996,21155300,139.339996
2017-03-03,138.779999,139.830002,138.589996,139.779999,21108100,139.779999
2017-03-02,140.00,140.279999,138.759995,138.960007,26153300,138.960007
2017-03-01,137.889999,140.149994,137.600006,139.789993,36272400,139.789993
2017-02-28,137.080002,137.440002,136.699997,136.990005,23403500,136.990005
2017-02-27,137.139999,137.440002,136.279999,136.929993,20196400,136.929993
2017-02-24,135.910004,136.660004,135.279999,136.660004,21690900,136.660004
2017-02-23,137.380005,137.479996,136.300003,136.529999,20704100,136.529999
2017-02-22,136.429993,137.119995,136.110001,137.110001,20745300,137.110001
```

Yahoo Finance Charts Alphabet Inc. (GOOG)

http://finance.yahoo.com/echarts?s=GOOG+Interactive#{"showArea":false,"showLine":false,"showCandle":true,"lineType":"candle","range":"5y","allowChartStacking":true}

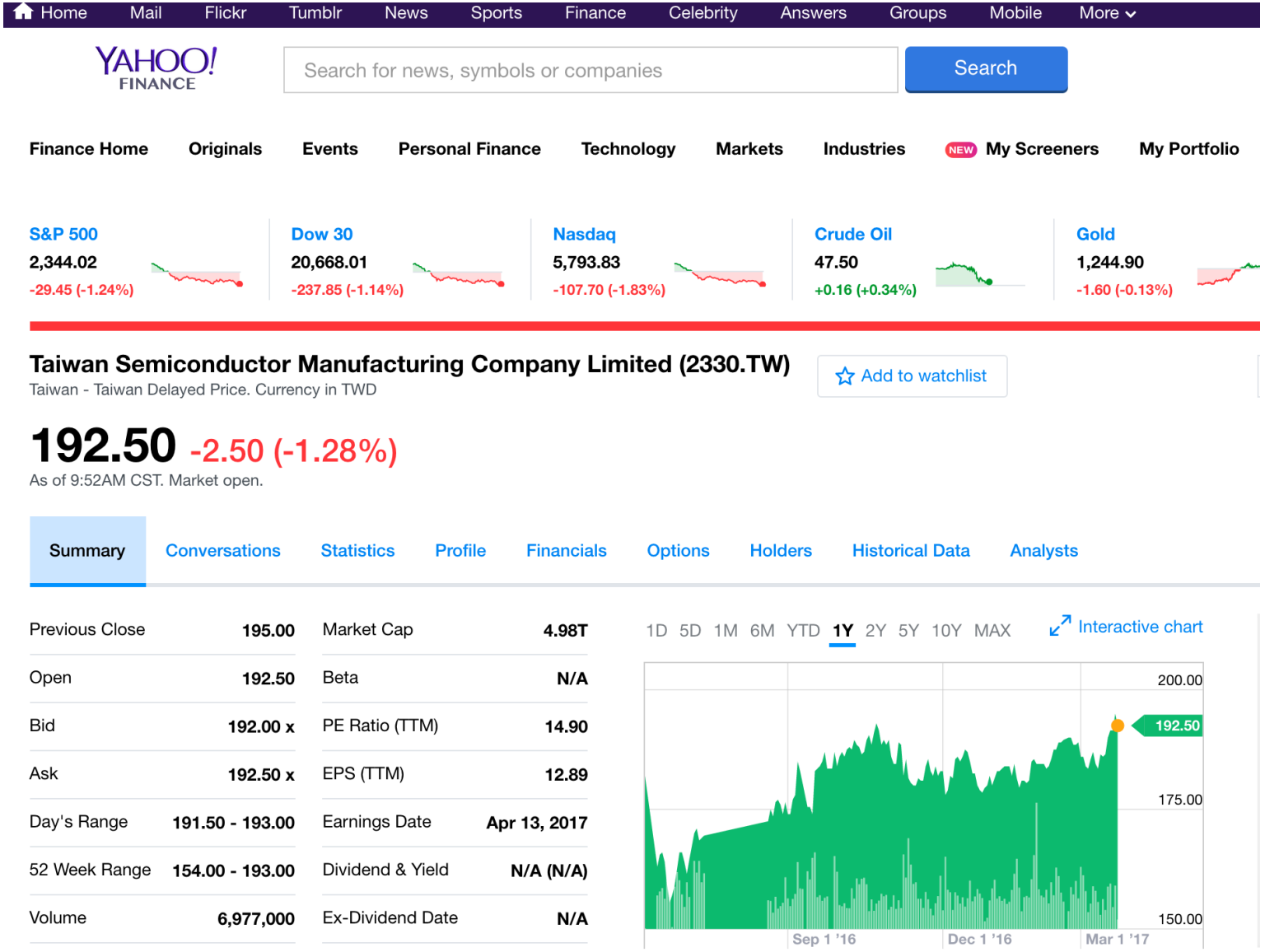
Dow Jones Industrial Average (^DJI)

http://finance.yahoo.com/chart/^DJI

TSEC weighted index (^TWII) - Taiwan

http://finance.yahoo.com/chart/^DJI

Taiwan Semiconductor Manufacturing Company Limited (2330.TW)



http://finance.yahoo.com/q?s=2330.TW

Yahoo Finance Charts
TSMC (2330.TW)

http://finance.yahoo.com/chart/2330.TW

```
import pandas as pd
import pandas_datareader.data as web
df = web.DataReader('AAPL', data_source='yahoo',
start='1/1/2010', end='3/21/2017')
df.to_csv('AAPL.csv')
df.tail()
```

```
import pandas as pd
import pandas_datareader.data as web
#df = web.DataReader('AAPL', 'yahoo')
df = web.DataReader('AAPL', data_source='yahoo', start='1/1/2010', end='3/21/2017')
#df = web.DataReader('AAPL', data_source='google', start='1/1/2010', end='3/21/2017')
df.to_csv('AAPL.csv')
df.tail()
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-15	139.410004	140.750000	139.029999	140.460007	25566800	140.460007
2017-03-16	140.720001	141.020004	140.259995	140.690002	19132500	140.690002
2017-03-17	141.000000	141.000000	139.889999	139.990005	43597400	139.990005
2017-03-20	140.399994	141.500000	140.229996	141.460007	20213100	141.460007
2017-03-21	142.110001	142.800003	139.729996	139.839996	39116800	139.839996

```
df = web.DataReader('GOOG',
data_source='yahoo', start='1/1/1980',
end='3/21/2017')
df.head(10)
```

```
df = web.DataReader('GOOG', data_source='yahoo', start='1/1/1980', end='3/21/2017')
df.head(10)
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2004-08-19	100.000168	104.060182	95.960165	100.340176	44871300	50.119968
2004-08-20	101.010175	109.080187	100.500174	108.310183	22942800	54.100990
2004-08-23	110.750191	113.480193	109.050183	109.400185	18342800	54.645447
2004-08-24	111.240189	111.600192	103.570177	104.870176	15319700	52.382705
2004-08-25	104.960181	108.000187	103.880180	106.000184	9232100	52.947145
2004-08-26	104.950180	107.950188	104.660179	107.910182	7128600	53.901190
2004-08-27	108.100185	108.620186	105.690180	106.150181	6241200	53.022069
2004-08-30	105.280178	105.490184	102.010172	102.010172	5221400	50.954132
2004-08-31	102.300173	103.710180	102.160177	102.370175	4941200	51.133953
2004-09-01	102.700174	102.970180	99.670169	100.250171	9181600	50.075011

df.tail(10)

```
df.tail(10)
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-08	833.510010	838.150024	831.789978	835.369995	988700	835.369995
2017-03-09	836.000000	842.000000	834.210022	838.679993	1259900	838.679993
2017-03-10	843.280029	844.909973	839.500000	843.250000	1701100	843.250000
2017-03-13	844.000000	848.684998	843.250000	845.539978	1149500	845.539978
2017-03-14	843.640015	847.239990	840.799988	845.619995	779900	845.619995
2017-03-15	847.590027	848.630005	840.770020	847.200012	1379600	847.200012
2017-03-16	849.030029	850.849976	846.130005	848.780029	970400	848.780029
2017-03-17	851.609985	853.400024	847.109985	852.119995	1712300	852.119995
2017-03-20	850.010010	850.219971	845.150024	848.400024	1190300	848.400024
2017-03-21	851.400024	853.500000	829.020020	830.460022	2442900	830.460022

df.count()

```
df.count()
```

```
Open          3169
High          3169
Low           3169
Close         3169
Volume        3169
Adj Close     3169
dtype: int64
```

df.ix['2015-12-31']

```
df.ix['2015-12-31']
```

```
Open          7.695000e+02
High          7.695000e+02
Low           7.583400e+02
Close         7.588800e+02
Volume        1.489600e+06
Adj Close     7.588800e+02
Name: 2015-12-31 00:00:00, dtype: float64
```

```
df.to_csv('2330.TW.Yahoo.Finance.Data.csv')
```

2330.TW.Yahoo.Finance.Data.csv

```
Date,Open,High,Low,Close,Volume,Adj Close
2010-01-01,64.5,64.5,64.5,64.5,0,52.8308
2010-01-04,65.0,65.0,64.0,64.9,39407000,53.1584
2010-01-05,65.0,65.1,63.9,64.5,37138000,52.8308
2010-01-06,64.5,64.9,63.7,64.9,49261000,53.1584
2010-01-07,64.9,65.0,64.2,64.2,42134000,52.5851
2010-01-08,63.5,64.3,63.5,64.0,46076000,52.4213
2010-01-11,64.0,64.9,63.5,64.5,36799000,52.8308
2010-01-12,64.4,64.4,63.3,63.6,49853000,52.0936
2010-01-13,63.0,63.1,62.6,62.8,47976000,51.4384
2010-01-14,63.6,63.6,63.0,63.2,36149000,51.766
2010-01-15,62.9,63.5,62.8,63.5,47852000,52.0117
2010-01-18,62.8,63.1,62.8,62.9,30136000,51.5203
2010-01-19,63.0,63.2,62.0,62.5,47202000,51.1926
2010-01-20,62.9,63.2,62.2,63.0,52281000,51.6022
```

Python Pandas for Finance

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline
```

```
#Python for Stocks: 1
#Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/
#Import Packages Required
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
#Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2015, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()
```

```
#Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2015, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-15	139.410004	140.750000	139.029999	140.460007	25566800	140.460007
2017-03-16	140.720001	141.020004	140.259995	140.690002	19132500	140.690002
2017-03-17	141.000000	141.000000	139.889999	139.990005	43597400	139.990005
2017-03-20	140.399994	141.500000	140.229996	141.460007	20213100	141.460007
2017-03-21	142.110001	142.800003	139.729996	139.839996	39116800	139.839996

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
```

```
df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
<matplotlib.axes._subplots.AxesSubplot at 0x1150bac88>
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
plt.figure(figsize=(12,9))
top = plt.subplot2grid((12,9), (0, 0),
rowspan=10, colspan=9)
bottom = plt.subplot2grid((12,9), (10,0),
rowspan=2, colspan=9)
top.plot(df.index, df['Adj Close'],
color='blue') #df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
<matplotlib.text.Text at 0x115630860>
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
plt.figure(figsize=(12,9))
top = plt.subplot2grid((12,9), (0, 0), rowspan=10, colspan=9)
bottom = plt.subplot2grid((12,9), (10,0), rowspan=2, colspan=9)
top.plot(df.index, df['Adj Close'], color='blue') #df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')
```

<matplotlib.text.Text at 0x115630860>

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

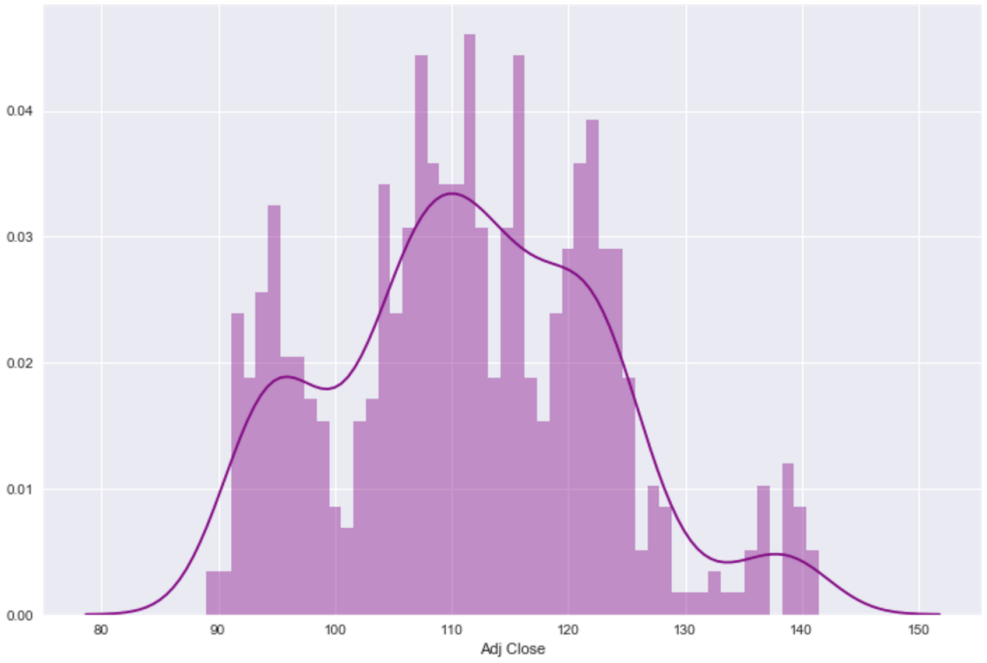
Python Pandas for Finance

```
plt.figure(figsize=(12,9))
sns.distplot(df['Adj Close'].dropna(),
bins=50, color='purple')
```

```
plt.figure(figsize=(12,8))
sns.distplot(df['Adj Close'].dropna(), bins=50, color='purple')
```

```
/Users/imyday/anaconda/lib/python3.6/site-packages/statsmodels/nonparametric/kdetools
g: using a non-integer number instead of an integer will result in an error in the fu
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x116309780>
```



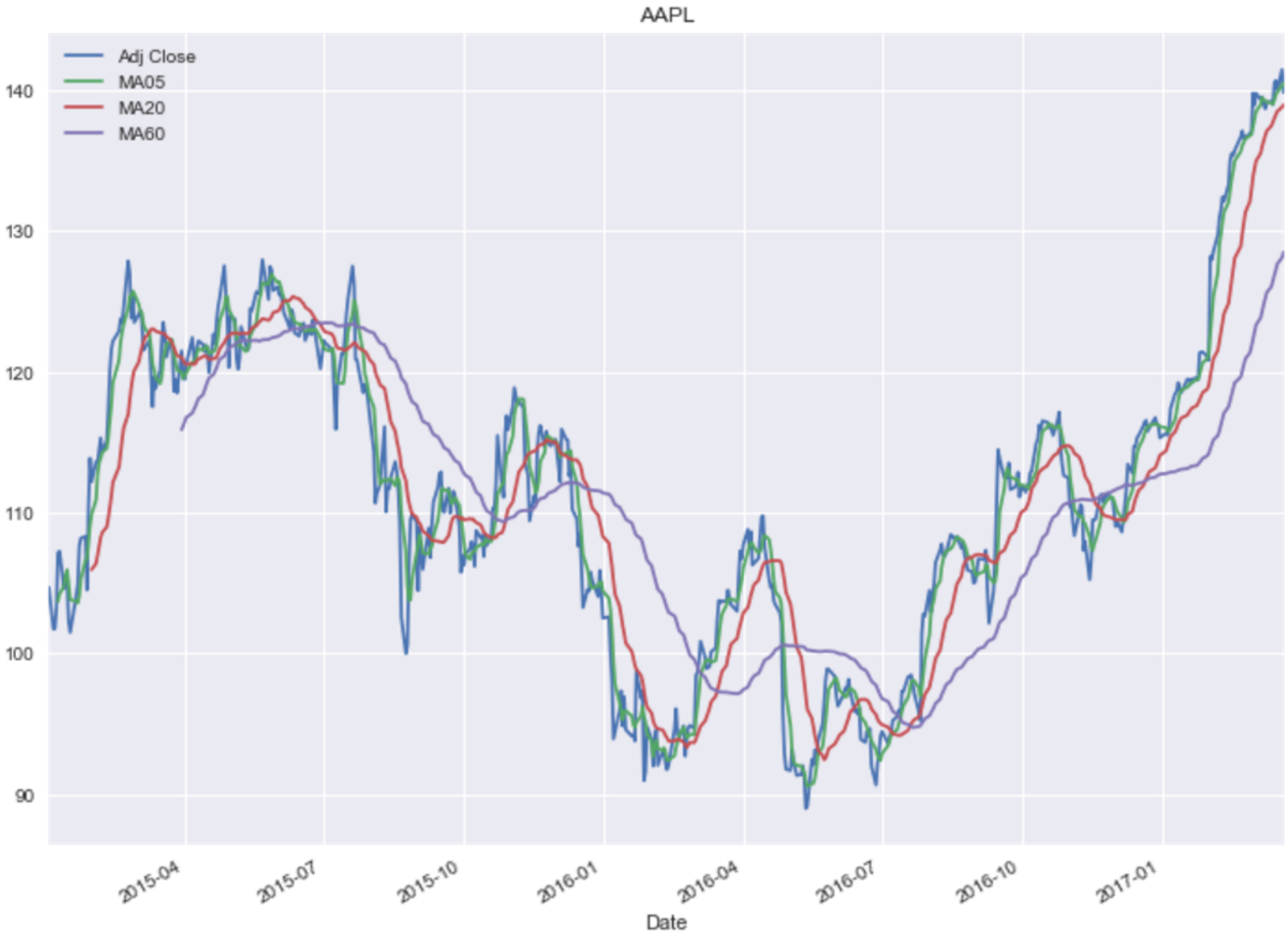
Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean()
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days

df2 = pd.DataFrame({'Adj Close': df['Adj
Close'],'MA05': df['MA05'],'MA20':
df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True,
title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
plt.show()
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

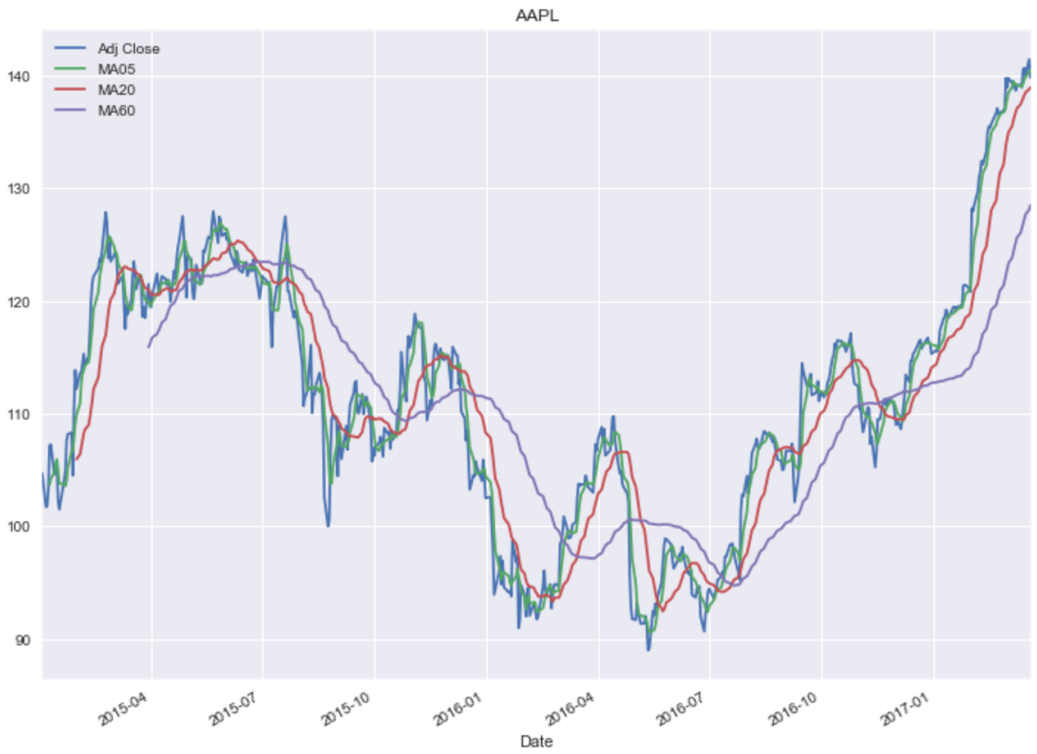
Python Pandas for Finance



Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days

df2 = pd.DataFrame({'Adj Close': df['Adj Close'],'MA05': df['MA05'],'MA20': df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
```



Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

```
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline

#Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2015, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()

df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
plt.figure(figsize=(12,9))
top = plt.subplot2grid((12,9), (0, 0), rowspan=10, colspan=9)
bottom = plt.subplot2grid((12,9), (10,0), rowspan=2, colspan=9)
top.plot(df.index, df['Adj Close'], color='blue') #df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')

plt.figure(figsize=(12,9))
sns.distplot(df['Adj Close'].dropna(), bins=50, color='purple')

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days
df2 = pd.DataFrame({'Adj Close': df['Adj Close'],'MA05': df['MA05'],'MA20': df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
plt.show()
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Examples: Python Pandas for Finance

In [11]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import pandas.io.data as web
import random

import datetime
import time
import timeit

import io
import os

import re
import codecs
import requests
get_ipython().magic('matplotlib inline')

from scipy import stats

#pd.set_option('display.notebook_repr_html', False)
pd.set_option('display.max_columns', 15)
pd.set_option('display.max_rows', 10)
pd.set_option('precision', 3)

def getDateTimeNow():
    strnow = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    return strnow

print('Hello Pandas')
print(getDateTimeNow())
```

```
Hello Pandas
20160323_145708
```

```
sSymbol = "AAPL"
#sSymbol = "GOOG"
#sSymbol = "IBM"
#sSymbol = "MSFT"
#sSymbol = "^TWII"
#sSymbol = "000001.SS"
#sSymbol = "2330.TW"
#sSymbol = "2317.TW"

# sURL = "http://ichart.finance.yahoo.com/table.csv?s=AAPL"
# sBaseURL = "http://ichart.finance.yahoo.com/table.csv?s="
sURL = "http://ichart.finance.yahoo.com/table.csv?s=" + sSymbol
#req = requests.get("http://ichart.finance.yahoo.com/table.csv?s=2330.TW")
#req = requests.get("http://ichart.finance.yahoo.com/table.csv?s=AAPL")
req = requests.get(sURL)

sText = req.text
#print(sText)
#df = web.DataReader(sSymbol, 'yahoo', starttime, endtime)
#df = web.DataReader("2330.TW", 'yahoo')

sPath = "data/"
sPathFilename = sPath + sSymbol + ".csv"
print(sPathFilename)

f = open(sPathFilename, 'w')
f.write(sText)
f.close()
sIOdata = io.StringIO(sText)
df = pd.DataFrame.from_csv(sIOdata)
df.head(5)
```

In [13]:
```
starttime = datetime.datetime(2000, 1, 1)
endtime = datetime.datetime(2015, 12, 31)
sSymbol = "AAPL"
#sSymbol = "GOOG"
#sSymbol = "IBM"
#sSymbol = "MSFT"
#sSymbol = "^TWII"
#sSymbol = "000001.SS"
#sSymbol = "2330.TW"
#sSymbol = "2317.TW"
#"^TWII"
#"AAPL"
#SHA:000016"
#"600000.SS"
#"2330.TW"
df = web.DataReader(sSymbol, 'yahoo', starttime, endtime)
#df_01 = web.DataReader("2330.TW", 'yahoo')
sSymbol = sSymbol.replace(":","_")
sSymbol = sSymbol.replace("^","_")
sPath = "data/financedata/"
#sPath = "/users/imyday/SCDBA/data/"  #Mac OS X
#sPath = "C:\data\"                   #Windows

sPathFilename = sPath + sSymbol + "_Yahoo_Finance.csv"
print(sPathFilename)
df.to_csv(sPathFilename)
df.head(5)
```

```
data/financedata/AAPL_Yahoo_Finance.csv
```

Out[13]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2000-01-03	104.875	112.500	101.688	111.938	133949200	3.702
2000-01-04	108.250	110.625	101.188	102.500	128094400	3.390

df.tail(5)

```
In [14]: df.tail(5)
```

Out[14]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2015-12-24	109.00	109.00	107.95	108.03	13596700	107.447
2015-12-28	107.59	107.69	106.18	106.82	26704200	106.243
2015-12-29	106.96	109.43	106.86	108.74	30931200	108.153
2015-12-30	108.58	108.70	107.18	107.32	25213800	106.741
2015-12-31	107.01	107.03	104.82	105.26	40912300	104.692

```
sSymbol = "AAPL”

# sURL = "http://ichart.finance.yahoo.com/table.csv?s=AAPL"
sURL = "http://ichart.finance.yahoo.com/table.csv?s=" + sSymbol
#req = requests.get("http://ichart.finance.yahoo.com/table.csv?s=AAPL")
req = requests.get(sURL)

sText = req.text
#print(sText)

sPath = "data/"
sPathFilename = sPath + sSymbol + ".csv"
print(sPathFilename)

f = open(sPathFilename, 'w')
f.write(sText)
f.close()
sIOdata = io.StringIO(sText)
df = pd.DataFrame.from_csv(sIOdata)
df.head(5)
```

In [15]:
```
sSymbol = "AAPL"
#sSymbol = "GOOG"
#sSymbol = "IBM"
#sSymbol = "MSFT"
#sSymbol = "^TWII"
#sSymbol = "000001.SS"
#sSymbol = "2330.TW"
#sSymbol = "2317.TW"

# sURL = "http://ichart.finance.yahoo.com/table.csv?s=AAPL"
# sBaseURL = "http://ichart.finance.yahoo.com/table.csv?s="
sURL = "http://ichart.finance.yahoo.com/table.csv?s=" + sSymbol
#req = requests.get("http://ichart.finance.yahoo.com/table.csv?s=2330.TW")
#req = requests.get("http://ichart.finance.yahoo.com/table.csv?s=AAPL")
req = requests.get(sURL)

sText = req.text
#print(sText)
#df = web.DataReader(sSymbol, 'yahoo', starttime, endtime)
#df = web.DataReader("2330.TW", 'yahoo')

sPath = "data/"
sPathFilename = sPath + sSymbol + ".csv"
print(sPathFilename)

f = open(sPathFilename, 'w')
f.write(sText)
f.close()
sIOdata = io.StringIO(sText)
df = pd.DataFrame.from_csv(sIOdata)
df.head(5)
```

```
data/AAPL.csv
```

```
data/AAPL.csv
```

Out[15]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2016-03-22	105.25	107.29	105.21	106.72	32232600	106.72
2016-03-21	105.93	107.65	105.14	105.91	35180800	105.91
2016-03-18	106.34	106.50	105.19	105.92	43402300	105.92
2016-03-17	105.52	106.47	104.96	105.80	34244600	105.80
2016-03-16	104.61	106.31	104.59	105.97	37893800	105.97

```
def getYahooFinanceData(sSymbol, starttime, endtime, sDir):
    #GetMarketFinanceData_From_YahooFinance
    #"^TWII"
    #"000001.SS"
    #"AAPL"
    #SHA:000016"
    #"600000.SS"
    #"2330.TW"
    #sSymbol = "^TWII"
    starttime = datetime.datetime(2000, 1, 1)
    endtime = datetime.datetime(2015, 12, 31)
    sPath = sDir
    #sPath = "data/financedata/"
    df_YahooFinance = web.DataReader(sSymbol, 'yahoo', starttime, endtime)
    #df_01 = web.DataReader("2330.TW", 'yahoo')
    sSymbol = sSymbol.replace(":","_")
    sSymbol = sSymbol.replace("^","_")
    sPathFilename = sPath + sSymbol + "_Yahoo_Finance.csv"
    df_YahooFinance.to_csv(sPathFilename)
    #df_YahooFinance.head(5)
    return sPathFilename
    #End def getYahooFinanceData(sSymbol, starttime, endtime, sDir):
```

In [16]:
```
def getYahooFinanceData(sSymbol, starttime, endtime, sDir):
    #GetMarketFinanceData_From_YahooFinance
    #"^TWII"
    #"000001.SS"
    #"AAPL"
    #SHA:000016"
    #"600000.SS"
    #"2330.TW"
    #sSymbol = "^TWII"
    starttime = datetime.datetime(2000, 1, 1)
    endtime = datetime.datetime(2015, 12, 31)
    sPath = sDir
    #sPath = "data/financedata/"
    df_YahooFinance = web.DataReader(sSymbol, 'yahoo', starttime, endtime)
    #df_01 = web.DataReader("2330.TW", 'yahoo')
    sSymbol = sSymbol.replace(":","_")
    sSymbol = sSymbol.replace("^","_")
    sPathFilename = sPath + sSymbol + "_Yahoo_Finance.csv"
    df_YahooFinance.to_csv(sPathFilename)
    #df_YahooFinance.head(5)
    return sPathFilename
    #End def getYahooFinanceData(sSymbol, starttime, endtime, sDir):
```

```
sSymbol = "AAPL”
starttime = datetime.datetime(2000, 1, 1)
endtime = datetime.datetime(2015, 12, 31)
sDir = "data/financedata/"

sPathFilename = getYahooFinanceData(sSymbol, starttime, endtime,
sDir)
print(sPathFilename)
```

In [17]:
```
sSymbol = "AAPL"
#sSymbol = "GOOG"
#sSymbol = "IBM"
#sSymbol = "MSFT"
#sSymbol = "^TWII"
#sSymbol = "000001.SS"
#sSymbol = "2330.TW"
#sSymbol = "2317.TW"
starttime = datetime.datetime(2000, 1, 1)
endtime = datetime.datetime(2015, 12, 31)
sDir = "data/financedata/"

sPathFilename = getYahooFinanceData(sSymbol, starttime, endtime, sDir)
print(sPathFilename)
```

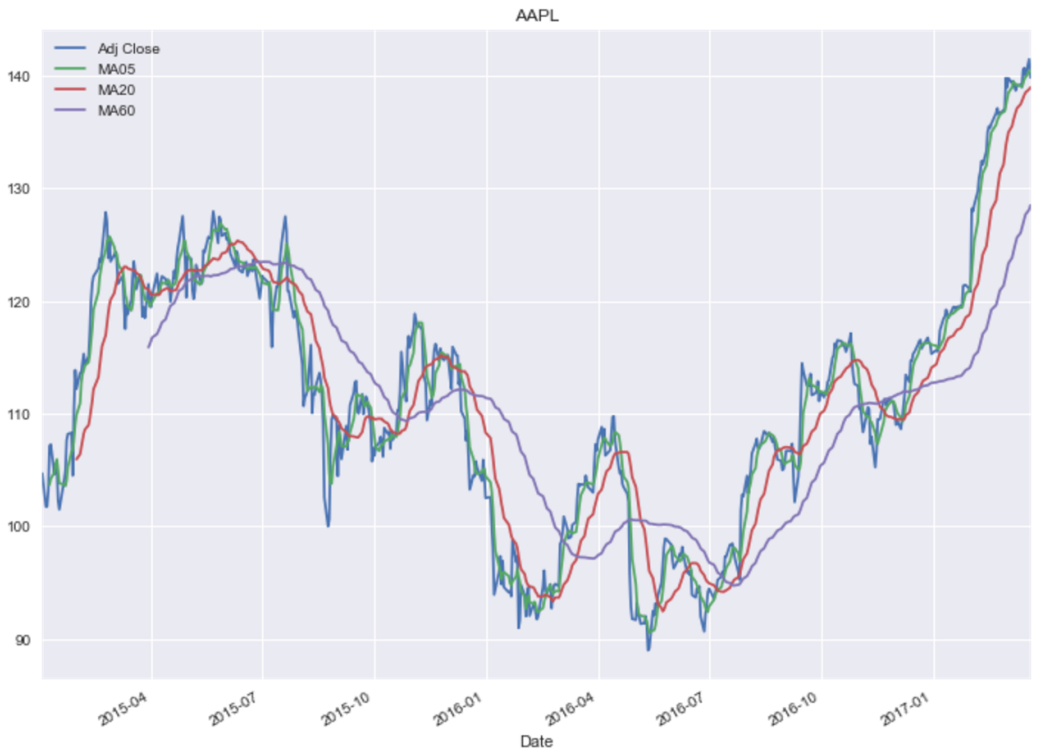
```
data/financedata/AAPL_Yahoo_Finance.csv
```

```
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline

#Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2015, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()

df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
plt.figure(figsize=(12,9))
top = plt.subplot2grid((12,9), (0, 0), rowspan=10, colspan=9)
bottom = plt.subplot2grid((12,9), (10,0), rowspan=2, colspan=9)
top.plot(df.index, df['Adj Close'], color='blue') #df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')

plt.figure(figsize=(12,9))
sns.distplot(df['Adj Close'].dropna(), bins=50, color='purple')

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days
df2 = pd.DataFrame({'Adj Close': df['Adj Close'],'MA05': df['MA05'],'MA20': df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
plt.show()
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

Python Pandas for Finance

```
# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days

df2 = pd.DataFrame({'Adj Close': df['Adj Close'],'MA05': df['MA05'],'MA20': df['MA20'], 'MA60': df['MA60']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')
fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)
```



Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/

References

- Wes McKinney (2012), Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media
- Yves Hilpisch (2014), Python for Finance: Analyze Big Financial Data, O'Reilly
- Yves Hilpisch (2015), Derivatives Analytics with Python: Data Analysis, Models, Simulation, Calibration and Hedging, Wiley
- Michael Heydt (2015) , Mastering Pandas for Finance, Packt Publishing
- Michael Heydt (2015), Learning Pandas - Python Data Discovery and Analysis Made Easy, Packt Publishing
- James Ma Weiming (2015), Mastering Python for Finance, Packt Publishing
- Fabio Nelli (2015), Python Data Analytics: Data Analysis and Science using PANDAs, matplotlib and the Python Programming Language, Apress
- Wes McKinney (2013), 10-minute tour of pandas, https://vimeo.com/59324550
- Jason Wirth (2015), A Visual Guide To Pandas, https://www.youtube.com/watch?v=9d5-Ti6onew
- Edward Schofield (2013), Modern scientific computing and big data analytics in Python, PyCon Australia, https://www.youtube.com/watch?v=hqOsfS3dP9w
- Python Programming, https://pythonprogramming.net/