Berlin School of Economics and Law
Prof. Dr. Natalie Packham

Summer term 2021
LV-Nr. 425093

Computational Finance and FinTech

Getting Started

Contents

0 Getting started

0.1 Introduction

Introduction

- This course introduces students to computational methods used in the Finance and FinTech industry.
- Students will learn how to write programming code in **Python**.
- Along with developing programming skills, a variety of **computational and numerical aspects of finance** will be introduced, including
  - statistical methods in finance,
  - financial time series,
  - option pricing,
  - Monte Carlo methods.
- Some methods from machine learning will be treated as well.

Introduction

- A lot of important information about the course can be found in the **syllabus** on the course's Moodle webpage.
- The lecture script mainly follows the textbook **Python for Finance (2nd edition)** by Yves Hilpisch.
- The book is available as an **ebook** from the HWR library (link).
- I am grateful to **Yves Hilipsch** (The Python Quants Group) for letting me use his code and material.
- References to the book are abbreviated as **Py4Fi**.
- For example, along with this introductory chapter you may want to consult **Py4Fi, Ch. 1**.

Background

- Python is an open-source programming language that can be downloaded and used for free.
- Python was created by Guido van Rossum and first published in 1991.
- Today the language is largely developed by the Python Software Foundation, a nonprofit organization.
- It is named after the British comedy group "Monty Python".

Features

- Open source
- Interpreted, i.e., code is compiled at runtime
- Multiparadigm (supports e.g. object orientation)

- Multipurpose, i.e., it is used for rapid prototyping as well as production systems
- Cross-platform (runs on Windows, Linux, MacOS, ...)
- Dynamically typed (types are inferred by the Python interpreter)
- Indentation aware (indentation is used instead of braces to mark code blocks)
- Garbage collection (automated memory management)

[1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Many examples and extended information can be found on the following websites:

- Beginners' Guide
- Python.org
- Scipy Lectures
- The Hitchhiker's Guide to Python

Pros

- **Universal**: Python runs on any operating system.
- **Easy to learn**: Although Python is highly versatile (e.g. can be used for scientific computing), it is relatively easy to learn.
- **Readable code**: Python is a high-level programming language, making it easy to read and work with.
- **General purpose**: The language can be applied to solve different problems at hand.

Cons

- **Speed**: While Python is not slow, it cannot keep up with compiled languages such as C, C++, Fortran, COBOL, etc.

0.2 Installing Python and Jupyter Notebook

Setting up Python for this class

- We will use **Python 3.7** in this class.
- Easiest method to get started:

  - Install Anaconda; this will setup up **Python** as well as the **Jupyter Notebook** environment that we are going to use in class.
  - Make sure to install a version of Ananconda that installs Python 3.7 (as opposed to the current version 3.8).
  - For this, go to the archive https://repo.anaconda.com/archive/ and install the version named Anaconda3-2020.02 appropriate for your operating system.

Setting up Github for this class

- The easiest way to access code for this class is to install **GitHub**:
  - Getting started with GitHub
  - You may also just download a zip-file with the code from there.
- The course's repository is here.
- The **Py4Fi** book's repository is here.

Jupyter Notebook

- **Jupyter Notebook** is a browser-based application used for creating and sharing documents, containing live code, visualizations, equations, plain text, and many other features.
- Launch Jupyter Notebook:
  - from Anaconda or
  - from the command line (`Terminal` in MacOs or `cmd` in Windows) using the command

```
jupyter notebook
```

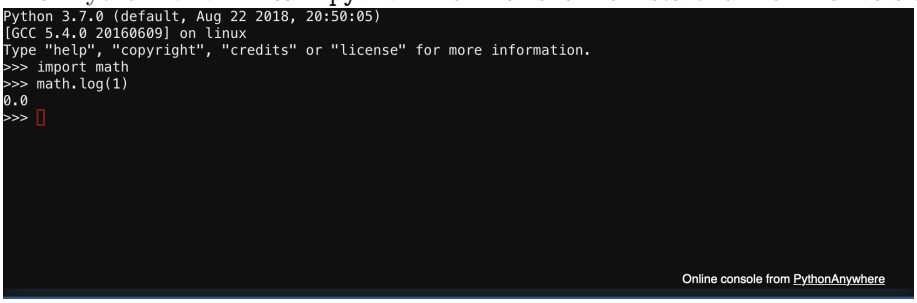
- Jupyter notebooks run in a local webserver.

Jupyter Notebook

- Before launching Jupyter Notbook from the command line you may wish to navigate to the directory where the notebooks are stored or where you want to store them.
- To open a new notebook click on "New" and "Python 3" in the Jupyter Notebook main window.
- To learn to use Jupyter Notebook before, click on "Help" and "User Interface Tour".

Other ways to run Python

- The Python **shell**: call `python` from a shell or start an online version at online shell



Other ways to run Python

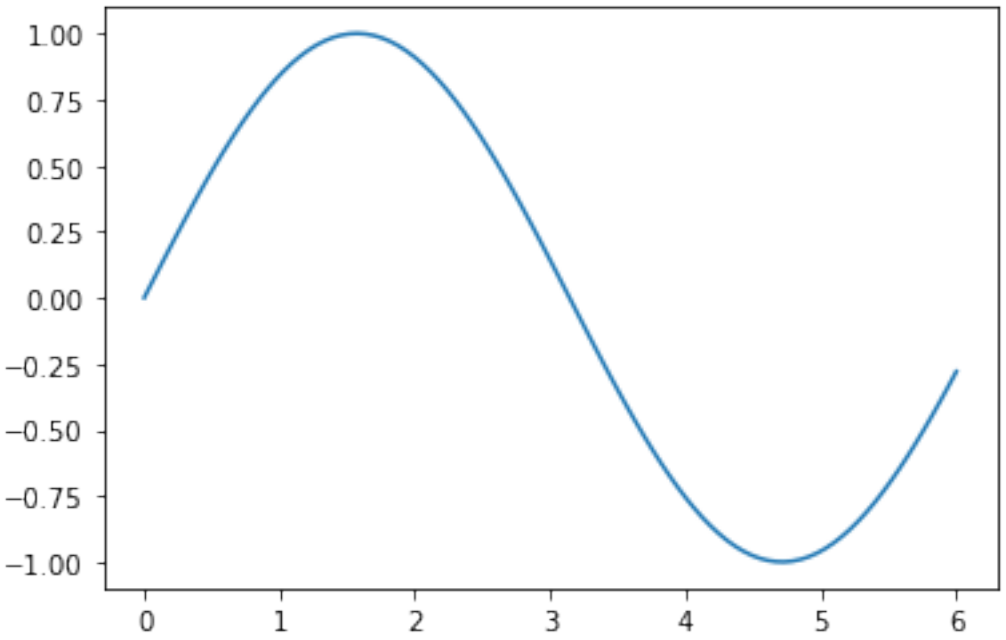
- Run Python scripts (suffix `.py`) from a command line using the `python` command, e.g. `python file.py`.
- Python's **IDLE** (Integrated Development and Learning Environment) is another basic shell to run Python commands.
- An **IDE** (Integrated Development Environment) is an application that integrates programming, running code, debugging, etc.
- **IPython**: Interactive Python shell.

The Python Ecosystem

- Aside from the programming language, there is a large number of packages, modules and other tools available to support specific tasks.
- For example, various plotting libraries are available and can be readily used using `import`.

The Python Ecosystem

[2]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
x = np.linspace(0, 6, 100); y = np.sin(x);
plt.plot(x, y);
```



Popular packages ("The scientific stack")

- NumPy: multidimensional array objects
- SciPy: functionality often needed in science or finance
- matplotlib: plotting
- pandas: times series and tabular data
- scikit-learn: machine learning package
- PyTables: data storage package

0.3 A first example

Example loading and plotting financial data

- The following code imports the packages that will be used and sets up the plotting library.

[3]:
```
import numpy as np
import pandas as pd
from pylab import plt, mpl
plt.style.use('seaborn')
mpl.rcParams['font.family'] = 'serif'
```

```
%matplotlib inline
```

[4]:
```
import numpy as np
import pandas as pd
from pylab import plt, mpl

plt.style.use('seaborn')
mpl.rcParams['font.family'] = 'serif'
%matplotlib inline
```

Example loading and plotting financial data

- The following code imports a time series of S&P 500 index data from a `csv` file (`csv`=comma separated values).
- It also shows information about the time series.

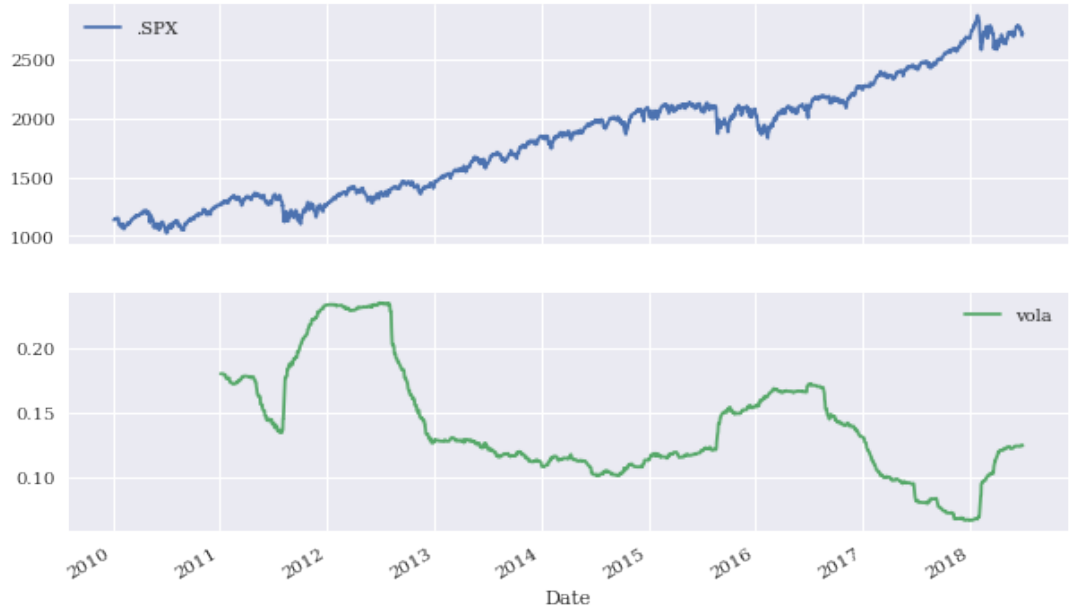
[5]:
```
data = pd.read_csv('data/tr_eikon_eod_data.csv', index_col=0, parse_dates=True)
data = pd.DataFrame(data['.SPX'])
data.dropna(inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2138 entries, 2010-01-04 to 2018-06-29
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   .SPX    2138 non-null   float64
dtypes: float64(1)
memory usage: 33.4 KB
```

Example loading and plotting financial data

- Transform the index level data to log-returns, estimate volatility (=standard deviation of returns) and produce plots of index and volatility.

[6]:
```
data['rets'] = np.log(data / data.shift(1))
data['vola'] = data['rets'].rolling(252).std() * np.sqrt(252)
data[['.SPX', 'vola']].plot(subplots=True, figsize=(10, 6));
```


.SPX
2500
2000
1500
1000
vola
0.20
0.15
0.10
2010
2011
2012
2013
2014
2015
2016
2017
2018
Date