A picture containing text, outdoor, alcohol

Description automatically generated

Gangmin Li

  Cambridge, UK 2023.02.10

**金融风险分析方法**

重庆理工大学

操作手册 1

内容提要：

本手册是风险分析方法三个实验操作的第一部分。 主要包括：

1. 实验环境的搭建：我将使用非常流行的Jupyter notebook； Jupyter Notebook是一个开源的Web应用程序，它允许您创建和共享文档，其中包括代码、方程、可视化图像和解释性文本。Jupyter Notebook支持交互执行代码。您可以单独执行代码块，以及多次执行代码并查看结果。Jupyter Notebook支持可视化和共享。您轻松创建和查看图形、图表和其他可视化元素。您可以通过将Notebook文件上传到GitHub、共享到JupyterHub或通过共享Notebook URL来与他人共享您的工作。
2. 收悉用于数据分析的Python基本语言和工具包；数据分析流程和结果汇报方法。
3. 本手册将贯穿 “learning by doing”通过具体实例来实现教学目的。学生需要根据操作步骤自行独立动手。从而获得所学知识和成功的满足。这也是模拟实际工作的最佳方式。

如果你遇到问题：

1. 使用Google 或者ChatGPT
2. 使用stackoverflow （<https://stackoverflow.com/>）
3. 和同学讨论
4. 询问老师

前 言

本手册将为您提供基本的数据分析工具（Python——当今最流行的数据科学和机器学习语言），并将其用于财务风险和回报分析。

Python 是一种高级编程语言，以简单易用称。 它具有广泛的应用，包括金融风险管理。 Python 提供了定义函数的功能，这是可扩展编程的核心 [https://www.python.org/]。 您可以在 Python 代码中使用强制和可选参数、关键字参数，甚至任意参数列表。 在本教程中，我们将主要使用 Pandas、NumPy 和 Matplotlib（Python 库）来帮助您入门。 在学习了基本方法和过程之后，您可以学习和使用其他高级 Python 库。

如果你从来没有使用过Python语言。 那么你可以跟随本手册使用 Jupyter notebook. Jupyter Notebook 是一个基于 Web 的交互式计算平台，允许您创建和共享包含实时代码、方程式、可视化和叙述文本的文档。 它是一种流行的数据分析、科学计算和机器学习工具，它支持包括 Python 在内的多种编程语言。

有多种方式可以在您的电脑上安装 Jupyter Notebook。 你可以根据自己的电脑类型自行安装。

1.使用Anaconda安装Jupyter Notebook

Anaconda 是一种流行的 Python 发行版，预装了许多用于数据科学和机器学习的库和工具，包括 Jupyter Notebook。 以下是使用 Anaconda 安装 Jupyter Notebook 的方法。

2.使用pip安装Jupyter Notebook

pip 是 Python 的包安装器，可以用来安装 Jupyter Notebook。 使用 pip，您可以从 Python 包索引 (PyPI) 或其他安装包索引来安装您需要的安装包。 pip 可以轻松管理 Python 的包，包括安装、升级和卸载，而无需手动下载和安装包。

# How to get the most out of this course?

To get the most out of this course you should do the following.

1. Download Example Notebook （\*.ipynb）open in your Jupyter Notebook (see how to get it below).
2. Watch videos if it has one.
3. Try run it line-by-line by yourself in Jupyter Notebook
4. Add your interpretation or understanding of each cell of code. Some of them have been done for you.

Notice that this tutorial is edited from a Rune tutorial “learn Python with Rune”. You can learn more from the original course website.

C O U R S E R E S O U R C E S

▸ Lesson Post: [https://www.learnpythonwithrune.org/p...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbXBndHc0RGZiMDhFc0JoSm9LLTNrX0tnZjE1Z3xBQ3Jtc0tuR0JTbm4teGFCWlVUX1VOS1B0VFJqaFBjSUtSZ0RqZkU3UTR3ajM4SlFTNDlqWXVpbVA1eVAyT3JxTVhBSlczck1XdEtheWZJSGR3dG5iVk9pT3BmaDhObnhNSjJrSXloZTVhY1F4dl9sNjYtYkZxNA&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Fpandas-for-financial-stock-analysis%2F&v=m8ahf_c9hEc)

▸ Jupyter Notebooks: [https://github.com/LearnPythonWithRun...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa3FRVGtRR29jQ0hkNUMtU0Q1WmFaU2JEMUN1Z3xBQ3Jtc0ttVUxSNFYzTktmeEJzTzhySl81RWJSN2NLRnlTQTRjRVl5cVB2RWEtdGVtaVZ6TlplSUhHdlpuekFNalo3WlJVZ3RUeVE5OUFxeE9BdDV3ZW5wRzNEQ1BGcUp1Q0twcDJ5N2ZBVW1NU0t1U2lscXQyQQ&q=https%3A%2F%2Fgithub.com%2FLearnPythonWithRune%2FFinancialDataAnalysisWithPython&v=m8ahf_c9hEc)

▸ Full video playlist [https://youtube.com/playlist?list=PLv...](https://www.youtube.com/playlist?list=PLvMRWNpDTNwQF6t_Tq7aVX0AI6H1avSpv)

## 第一节. 安装Jupyter Notebook

前面说过有两种安装Jupyter notebook 的方法：

## 使用Anaconda 安装Jupyter Notebook

Anaconda 是一种流行的 Python 发行版，预装了许多用于数据科学和机器学习的库和工具，包括 Jupyter Notebook。 以下是使用 Anaconda 安装 Jupyter Notebook 的方法和人步骤。

# Go to [Anaconda](https://www.anaconda.com/products/individual) (<https://www.anaconda.com/products/individual>) and download the individual edition.

Graphical user interface, text, application, email, website

Description automatically generated

# It will install Python and [Jupyter](https://jupyter-notebook.readthedocs.io/) notebook.

Graphical user interface, text, application

Description automatically generated

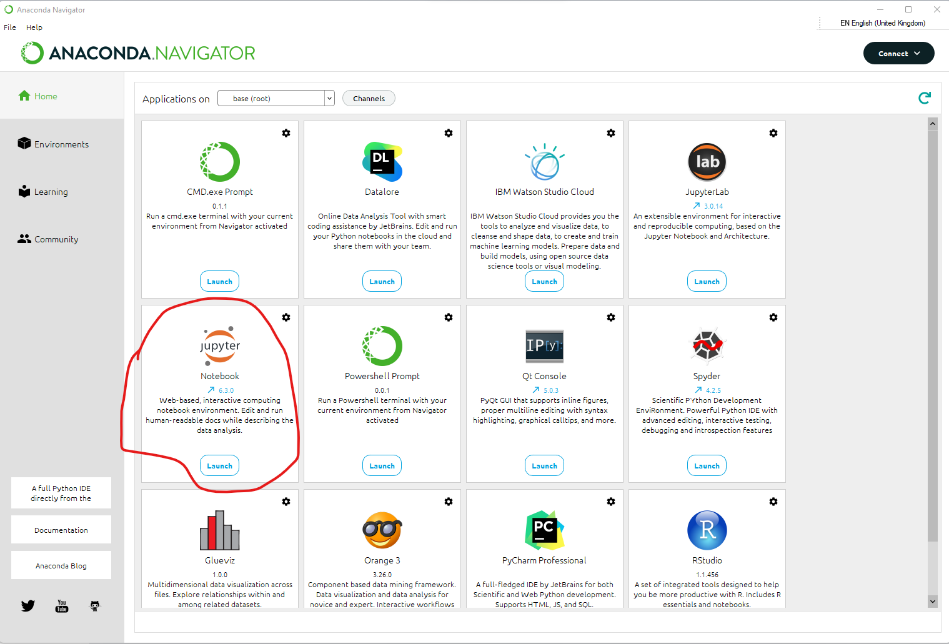
# Launch Anaconda.

Graphical user interface, application, Teams

Description automatically generated

# Launch Jupyter Notebook：

Lunch Jupyter notebook in anaconda navigator, by click lunch.



After you lunched the Jupyter notebook, You will need to access a web browser and goto:

**localhost:8888/**

You will see your local directory are listed in the browser as shown in the following figure.

Table

Description automatically generated

至此你的Jupyter notebook就安装完毕可以开始使用了。

## 使用pip 安装Jupyter Notebook

To use Jupyter Notebook with Python, you will need to have Python installed on your system. As I mentioned earlier, if you are using Windows, you can install Python through the Microsoft Store. Once you have installed Python, you will typically already have pip installed. Pip is a package-management system written in Python and is used to install and manage software packages.

To install Jupyter Notebook using pip, follow these steps:

1. Open a terminal or command prompt.
2. Run the following command:

pip install jupyter

jupyter notebook

After the installation is complete, you can start Jupyter Notebook by running the following command:

1. Type “Jupyter notebook” in anaconda PowerShell or,
2. In Windows CMD or PowerShell

Shape, rectangle

Description automatically generated

Now you should have jupyter notebook up and running.

## 使用云端 clouds

* If you don’t have or cannot install Jupyter Notebook. Alternatively, you can use some online Python running environment such as:
* **Google Colab:** Google Colab is a free cloud-based platform that provides a Jupyter Notebook environment that runs on Google's cloud servers. You can use it to write, run and share code, and you can even use free GPU and TPU resources provided by Google.
* **Microsoft Azure:** Azure provides a wide range of cloud services, including a Jupyter Notebook service. You can launch a Jupyter Notebook server on Azure, either as a virtual machine or as a container, and access it through a web browser.
* **Amazon Web Services (AWS):** AWS offers a cloud-based Jupyter Notebook service called Amazon SageMaker. You can launch a Jupyter Notebook server on SageMaker, which is preconfigured with popular Python libraries and machine learning frameworks.

We will use google Colab as an example to show the steps that you need to take to run your jupyter notebook.

1. Go to the Google Colab website at <https://colab.research.google.com/>. If you're not already signed into your Google account, you'll be prompted to sign in.
2. Once you're signed in, you'll be taken to the Colab home page. From there, you can create a new notebook by clicking on the "New notebook" button.
3. In the new notebook, you can write your Python code, add text and visualizations, and perform other operations that you would typically do in a Jupyter Notebook.
4. To save your notebook, you can either download it as a .ipynb file or save it to your Google Drive account.
5. If you need to install additional Python packages or libraries, you can do so using the "!pip install" command within a code cell.
6. Google Colab provides free GPU and TPU resources, so if you need to perform computationally intensive tasks, you can take advantage of these resources to speed up your computations.

Here is my Colab. It has advantages of linked with github, Google drive so you can run notebooks on these places or upload form your local drive.

**Colab** ([https://colab.research.google.com](https://colab.research.google.com)/)

**Another good python running server is Mybinder.**

[**https://mybinder.org/**](https://mybinder.org/)

you can simple put your github repo URL into a box and it will take you to your notebooks and provide a python running environment for you.

Graphical user interface, text, application

Description automatically generated

**Mybinder** [**https://mybinder.org/v2/gh/GangminLi/Basics-Financial-Risk-Management/HEAD**](https://mybinder.org/v2/gh/GangminLi/Basics-Financial-Risk-Management/HEAD)

Lastly,

Kaggle also provide a python running environment

[**https://www.kaggle.com/**](https://www.kaggle.com/)

**Kaggle code** [**https://www.kaggle.com/garygli/notebook62a25ca0ae/edit**](https://www.kaggle.com/garygli/notebook62a25ca0ae/edit)

## 第二节. Download and run tutorial Jupyter Notebooks

## Download, lunch and working on supplied Jupyter notebooks

You can Download Tutorial Jupyter notebooks from **My GitHub repository**:

<https://github.com/GangminLi/Financial-Risk-Management-with-Python/>

Graphical user interface, text, application, email

Description automatically generated

or **QQ 群文件夹**

## Open Jupyter notebook files

There are two ways to open your Jupyter notebooks:

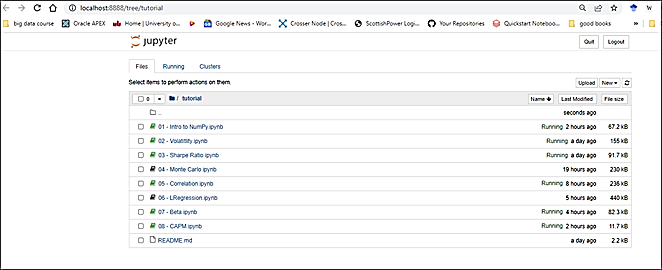
* 1. **Jupyter Notebook** **Environment**

In Jupyter Notebook Environment,

Navigate to the downloaded Notebooks

All the provided tutorials are in **XXXXX.ipynb** files. You should open it in **Jupyter notebook** and follow instructions.

Goto: localhost:8888/tree/tutorial



1. In windows command prompt or PowerShell change directory to your download directory and type “Jupyter notebook”

To open Jupyter notebook in your (downloaded) notebook directory. Using:



Here, **path/to/your/directory** with the actual path to the directory you want to set as the default.

For example, I have downloaded all the tutorial notebook into “C:\Users\gli46\OneDrive\Documents\Work\Teach\BDA for Risk Management\tutorial “

I will use,



to open notebook.

## 第三节. Actual tutorial

In this first tutorial there are totally 8 lessons:

[**Lesson 1**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-1): Get to know **pandas** with Python – how to get historical stock price data.

[**Lesson 2**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-2): Learn about [**Series**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html) from **pandas** – how to make calculations with the data.

[**Lesson 3**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-3): Learn about [**DataFrames**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html) from **pandas** – add, remove and enrich the data.

[**Lesson 4**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-4): Start visualize data with [**Matplotlib**](http://matplotlib.org/) – the best way to understand price data.

[**Lesson 5**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-5): Read data from **API**s – read data directly from pages like [**Yahoo! Finance**](http://finance.yahoo.com/) the right way.

[**Lesson 6**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-6): Calculate the [**Volatility**](https://www.investopedia.com/terms/v/volatility.asp) and [**Moving Average**](https://www.investopedia.com/terms/m/movingaverage.asp) of a stock.

[**Lesson 7**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-7): Technical indicators: [**MACD**](https://www.investopedia.com/terms/m/macd.asp) and [**Stochastic Oscillator**](https://www.investopedia.com/terms/s/stochasticoscillator.asp) – easy with **Pandas**.

[**Lesson 8**](https://www.learnpythonwithrune.org/start-python-with-pandas-for-financial-analysis/#lesson-8): Export it all into **Excel** – in multiple sheets with colour formatted cells and charts.

## Lesson 1. Get to know Pandas

In this part we will get familiar to work with [DataFrames](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html) – the primary data structure in Pandas. We will learn how to read a historical stock price data from [Yahoo! Finance](http://finance.yahoo.com/) and load it into a DataFrame. This will be done by exporting a CSV file from Yahoo! Finance and load the data. Later we will learn how to read the data directly from the Yahoo! Finance API.

A DataFrame is similar to an Excel sheet. DataFrame can contain data in a similar way as we will see in this lesson.

Then we will learn how to use the index of the dates. This will be necessary later when we make calculations later on.

The first part of the tutorial will give the foundation of what you need to know about DataFrames for financial analysis.

### File name: 01 - Introduction to Pandas

### Video: <https://youtu.be/m8ahf_c9hEc>

In the first tutorial you will download historical stock prices from [**Yahoo! Finance**](http://finance.yahoo.com/) (<https://finance.yahoo.com/>) as [**CSV** file](https://en.wikipedia.org/wiki/Comma-separated_values) (AAPL.csv as provided) and import them into our **Jupyter** notebook environment in a **DataFrame (called data)**.

If you are new to **CSV** files and **DataFrames**. Don’t worry, that is what we will cover here.

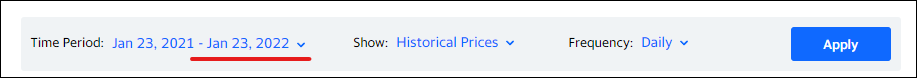
Let’s start by going to **Yahoo! Finance** and download the **CVS** file. In this course we have used Apple, but feel free to make similar calculation on a stock of your choice.

Go to **Yahoo! Finance** write **AAPL** (ticker for Apple) and press [**Historical Data**](https://finance.yahoo.com/quote/AAPL/history?p=AAPL) and download the **CSV** data file.

Graphical user interface

Description automatically generated

The default data are one year ending today.



You can change to any other date. But for now, let us use default values. And click “download” download the file.

The **CSV** data file named “**AAPL.CSV**” will contain Comma Separated Values (CSV) similar to this.

Table

Description automatically generated

The first line shows the column names (Date, Open, High, Low, Close, Adj Close, Volume). Then each line contains a data entry for a given day.

Now. let us load this file into Jupyter notebook and explore it with some basic Python tools.

If you want to build your own jupyter notebook, simple copy and past the code in this tutorial into your notebook.

In **Jupyter Notebook**, if we want to explore/analysis data in this file, we firstly need to import the **Pandas** library. This is needed in order to load the data into a **DataFrame**.

import pandas as pd

### Read data from CSV

Use the .read\_csv() function to read in the data from a CSV file. This function takes the path to the CSV file as its argument. The following code assume that “AAPL.CSV” file is in the currently directory.

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

The**read\_csv(…)** read the **CSV** file **AAPL.csv**.

The **AAPL.csv** file is the one you downloaded from**Yahoo! Finance** (or from the zip-file downloaded above) and needs to be located in the same folder you are working from in your **Jupyter notebook**.

The arguments in **read\_csv(…)** are the following.

* **index\_col=0** this sets the first column of the **CSV** file to be the index. In this case, it is the Date column.
* **parse\_dates=True** this ensures that dates in the CSV file are interpreted as dates. This is important if you want to take advantage of the index being a time.

### Understand data

When you get a new dataset, you store it in a DataFrame. The first thing you need to do is explore it and see what it contains. There are several useful methods and attributes for this.

* print() shows the entir dataset.
* .head() returns the first few rows (the “head” of the DataFrame).
* .tail() returns the first few rows (the “head” of the DataFrame).
* .info(). shows information on each of the columns, such as the data type and number of missing values.
* .shape returns the number of rows and columns of the DataFrame. and
* .describe().

The code

print(data)

will print out entire DataFrame.

Open High Low Close Adj Close Volume

Date

2020-01-27 77.514999 77.942497 76.220001 77.237503 76.576187 161940000

2020-01-28 78.150002 79.599998 78.047501 79.422501 78.742477 162234000

2020-01-29 81.112503 81.962502 80.345001 81.084999 80.390747 216229200

2020-01-30 80.135002 81.022499 79.687500 80.967499 80.274246 126743200

2020-01-31 80.232498 80.669998 77.072502 77.377502 76.714989 199588400

... ... ... ... ... ... ...

[253 rows x 6 columns]

data.head(10)

It simply outputs the first 10 lines in the DataFrame,

Table

Description automatically generated

Notice that “Date” is listed BELOW the other attributes like “Open”, “High”, “low” ..., and “Volume”, it means that the DataFrame is indexed by “Date”.

**.head** is commonly used after load data into a DataFrame to check if the data is correctly loaded.

data.tail()

Table

Description automatically generated

data.info()

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

DatetimeIndex: 253 entries, 2020-01-27 to 2021-01-26

Data columns (total 6 columns):

# Column Non-Null Count Dtype

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

0 Open 253 non-null float64

1 High 253 non-null float64

2 Low 253 non-null float64

3 Close 253 non-null float64

4 Adj Close 253 non-null float64

5 Volume 253 non-null int64

dtypes: float64(5), int64(1)

memory usage: 21.9 KB

data.shape

(253, 6)

data.describe()

Graphical user interface, text, table, Excel

Description automatically generated

After we load the CSV file into data, we can check its type,

data.dtypes

Run the code you will see,

Open float64

High float64

Low float64

Close float64

Adj Close float64

Volume int64

dtype: object

“data” type is “object”, which is DataFrame and each individual column’s type. Here, they are all typed “float64”. You can get yourself familiar with other types.

### Understand Index and use it quick access data

The index is an important part of a pandas DataFrame because it provides a way to easily identify and select rows based on the labels in the index, and it helps with the alignment of data in operations between DataFrames.

data.index

outputs,

DatetimeIndex(['2020-01-27', '2020-01-28', '2020-01-29', '2020-01-30',

'2020-01-31', '2020-02-03', '2020-02-04', '2020-02-05',

'2020-02-06', '2020-02-07',

...

'2021-01-12', '2021-01-13', '2021-01-14', '2021-01-15',

'2021-01-19', '2021-01-20', '2021-01-21', '2021-01-22',

'2021-01-25', '2021-01-26'],

dtype='datetime64[ns]', name='Date', length=253, freq=None)

We can use **loc** to lookup an index with a date and individual column or samples.

data.loc['2020-01-27']

this will give us,

Open 7.751500e+01

High 7.794250e+01

Low 7.622000e+01

Close 7.723750e+01

Adj Close 7.657619e+01

Volume 1.619400e+08

Name: 2020-01-27 00:00:00, dtype: float64

A more advanced option is to use an interval (or slice as it is called). Slicing with **loc** on a DataFrame is done by using a starting and ending index **.loc[start:end]**or an open ended index **.loc[start:]**, which will take data beginning from start to the last data.

data.loc['2021-01-01':]

This will give all the data starting from 2020-01-01. Notice, that there is no data on January 1st, but since the index is interpreted as a **datetime**, it can figure out the first date after.

Graphical user interface, application, table

Description automatically generated

data.loc[:'2020-07-01']

Table

Description automatically generated

Another important way to index into DataFrames is by **iloc[]**, which does it with index.

data.iloc[0]

data.iloc[-1]

Where you can index from the start with index **0, 1, 2, 3, …**Or from the end **-1, -2, -3, -4, …**

So, data.iloc[0] will output the first day (2020-01-27) data,

Open 7.751500e+01

High 7.794250e+01

Low 7.622000e+01

Close 7.723750e+01

Adj Close 7.657619e+01

Volume 1.619400e+08

Name: 2020-01-27 00:00:00, dtype: float64

and data.iloc[-1] will output the last day (2021-01-26) data,

Open 1.436000e+02

High 1.443000e+02

Low 1.413700e+02

Close 1.420800e+02

Adj Close 1.420800e+02

Volume 5.038856e+07

Name: 2021-01-26 00:00:00, dtype: float64

## Lesson 2: Learn about columns and Series

In the first lesson we learnt how to load data into a DataFrame. This part will show how to work with each column in the DataFrame. The columns are represented by a different data type, called [**Series**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html).

### In this lesson you will learn

In this lesson we will learn how to make calculations on the columns. The columns are represented by a data type called **Series**.

Each column in a **DataFrame** is a **Series** and can be easily accessed. Also, it is easy to calculate new **Series** of data. This is similar to calculate now columns of data in an **Excel** sheet.

### File name: 02 - Series

### Video: https://youtu.be/CG6rMc8FU9I

We will start by importing the data like in lesson 1.

import pandas as pd

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

Then we investigate the data type of the columns of the **DataFrame** **data**.

data.dtypes

Which results in the following.

Open float64

High float64

Low float64

Close float64

Adj Close float64

Volume int64

dtype: object

This means shows that each column has one data type. Here **Open** is float64. This is one difference from Excel sheets, where each cell has a data type. The advantage of restricting a data type per column is speed.

The data type of **data** is **DataFrame**.

The build in function **type(…)**gives you the type. It is handy to use it when exploring data. For example, the data type of **data** is **DataFrame**.

type(data)

You will see,

pandas.core.frame.DataFrame

Notice that it is given by a long string **pandas.core.frame.DataFrame**, this is the structure of the library **Pandas**.

The data type of a column in a DataFrame can be found by.

type(data['Close'])

Where **data[‘Close’]** gives access to column **Close** in the **DataFrame** **data**.

pandas.core.series.Series

Where we see a column is represented as a **Series**. The is similar to a DataFrame that it has an index. E.g. the **Series data[‘Close’]** has the same index as the **DataFrame data**.

Having the same index, it will help to calculate the daily difference from open and close. We can do,

daily\_chg = data['Open'] - data['Close']

This calculates a **Series daily\_chg** which is the *opening price minus the closing price*.

We can also calculate the daily change in percentage,

daily\_pct\_chg = (data['Close'] - data['Open'])/data['Open']\*100

In the calculation above we have limited us to only use data on the same rows (same dates). Later we will learn how to do it with data from previous day (the row above).

Now we will **normalize** the data by using the **iloc** we learned about in section 1.

norm = data['Close']/data['Close'].iloc[0]

The above statements calculates a **Series norm** where the **Close** price is normalized by dividing by the first available **Close** price, accessed by using **iloc[0]**.

This results in that **norm.iloc[0]** will be**1.0000**and **norm.iloc[-1]** we show the **return of this particular stock** if invested in on day 1 (index 0) and sold on the day of the last index (index -1), in the case of the video: **1.839521**.

## Lesson 3: Add, Remove and Calculate with columns in DataFrames

In the last lesson we learned about Series and how to make simple calculations using them. Also, how to normalize data.

### In this lesson you will learn

In this lession we will see how to add new columns calculated from values in other columns in our **DataFrame**. This is similar to calculate in Excel on data from different columns.

Then we will demonstrate some useful function when working with financial data.

Finally, we will show how to remove (or drop) columns from our **DataFrame**.

### File name: 03 - Work with DataFrames

### Video: <https://youtu.be/DMIQYZwBTeY>

As usual we need to load the data into our DataFrame.

import pandas as pd

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

It is always a good habit to inspect the data with**data.head()** (see the video lesson or the in the Notebook link below the video for expected output).

To create a new column in our data set simply write as follows.

data['Daily chg'] = data['Close'] - data['Open']

The above statement will create a new column named **Daily chg** with the difference between column **Close** and **Open**.

Similarly, you can create a column with the normalized data as follows.

data['Normalized'] = data['Close'] / data['Close'].iloc[0]

To find the minimum of a column.

data['Close'].min()

This will find the minimal value of the column **Close**.

To find the index of the minimum value use the following.

data['Close'].argmin()

You can do similar things as the following shows.

data['Normalized'].min()

data['Normalized'].argmin()

data['Close'].max()

data['Close'].argmax()

To get the mean value of a column, simply use **mean()**.

data['Close'].mean()

It is always good practice to remove the columns of data we do not intend to use anymore. This can be done by using **drop**().

data.drop(labels=['High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)

Where we use the following arguments.

* **labels=[‘High’, ‘Low’, ‘Adj Close’, ‘Volume’]** sets the labels of the columns we want to remove.
* **axis=1** sets the axis of the labels. Default is 0, and will look for the labels on the index. While axis 1 is the column names.
* **inplace=True** says it should actually remove the columns on the DataFrame we work on. Otherwise it will return a new DataFrame without the columns.

## Lesson 4: Use Matplotlib on historical stock prices to Visualize

In the last lesson we learned how to create new columns calculated by the values of the existing data. Also, how to remove columns we do not need.

### In this lesson you will learn

In the lesson we will learn about how to use **Matplotlib** (Visualization) with **DataFrames** on time series data (stock price data).

We will explore the object and functional way to use **Matplotlib** and how to create multiple charts in one figure. Also, we will see how to make bar charts among things.

### File name: 04 - Visualize DataFrames - Matplotlib

### Video: <https://youtu.be/2ywUfs0rgtU>

We start by reading the data into a DataFrame.

import pandas as pd

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

In the next lesson we will learn how to read data directly from an API.

You can visualize data directly form the DataFrame by using **data.plot()**, but as you can see in the Video or the Notebook link below the video, the result is not satisfactory.

To get a better experience we need to import **Matplotlib**. **Matplotlib** is a data visualization library used in data science in python (you may need to install **Matplotlib first**, pip install matplotlib).

import matplotlib.pyplot as plt

%matplotlib notebook

The **%matplotlib notebook** is important to tell Jupyter notebook how to visualize the results from our plots.

A simple plot can now be done as follows.

data.plot()



Can you find the problem with the graph?

Where are the open, high, low, close and Adj Close?

Ignore the obvious challenge here, that there are too much data and the scales are not similar (Volume is larger than the other).

It is okay to visualise a series but not a DataFrame, Let us try to plot “close” value in this way.

data['Close'].plot()

what have you seen? Nothing! What happened? First, you’ll think nothing. But you are actually wrong. If you run code “data.plot()” again, and pay attention to the horizontal line at the bottom, what is its colour?

Now you run “data['Close'].plot()”, and look the color of the same line, has it changed?

That was what has happened!

The plot actually paints the “lose” line In the original chart.

Now, let us try different thing. We plot a time serious with 4 columns and their value are randomly generated.

Here is the code,

# A randomly generated 4 columns of random values indexed with a date range to form a time series data.

index=pd.date\_range("1/1/2000", periods=1000)

df = pd.DataFrame(np.random.randn(1000, 4), index=index, columns=list("ABCD"))

df = df.cumsum()

df.plot()

You will see this,

Chart, line chart, scatter chart

Description automatically generated

It is very nice, isn’t it?

Now let us plot the 4 columns of "Open", "High", "Low", "Close" without “volume”. We need to assign the data in different DataFrame, let us call it data2.

# Plot a subdata with also 4 columns from the original data

data2 = data[["Open", "High", "Low", "Close"]]

data2.plot()

now you will see,

Chart, line chart, histogram

Description automatically generated

Now let us investigate the figure 1. You will notice that there is a line under the curve, that are the five columns are painted. Because the volume (the last column has significant scale in comparison with other 5 columns (the straight line underneath the curve; the other 5 has very small differences. So you see one straight line.

You can also separate subplots by using subplots = True argument.

# use subplots = True to seperate them

data.plot(subplots = True)

This will produce,

Chart, line chart

Description automatically generated

We can plot a single column without plot the whole DataFrame.

# plot a single column (series) like "Close"

data['Close'].plot()

Chart, line chart, scatter chart

Description automatically generated

### Use Matplotlib is in an objective way

To solve multiple plots in one figure problem, we can use Matplotlib in an object-oriented way.

The idea is we create objects first and do the plotting with their methods. Methods are the functions that come with the object.

Generally, we create both a **figure** and an **axes** object with plt.subplots(). Then we use the ax.plot() method from our axes object to create the plot. We also use two more methods, ax.set\_xlabel() and ax.set\_ylabel() to label our axes. We can aslo use ax.set\_title() to add a title if we want.

fig, ax = plt.subplots()

data['Close'].plot(ax=ax)

ax.set\_ylabel("Price")

ax.set\_title("AAPL")

This looks more involved at first. But let’s break it down.

* **fig, ax = plt.subplots()** Creates a new figure and axis to draw on. By default there will be one figure and one axis. The figure is the “picture”, the axis is the chart area.
* **data[‘Close’].plot(ax=ax)** The argument **ax=ax** tells **Pandas** to use the axis given from the above statement.
* **ax.set\_ylabel(“Price”)** This is actually not needed but just to create a label for the y axis.
* **ax.set\_title(“AAPL”)** The same here, it just creates a title.

The result would be something similar to this.

Chart, line chart

Description automatically generated

You can have multiple axis in one figure as follows.

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

data['Open'].plot(ax=ax[0, 0], title="Open")

data['High'].plot(ax=ax[0, 1], title="High")

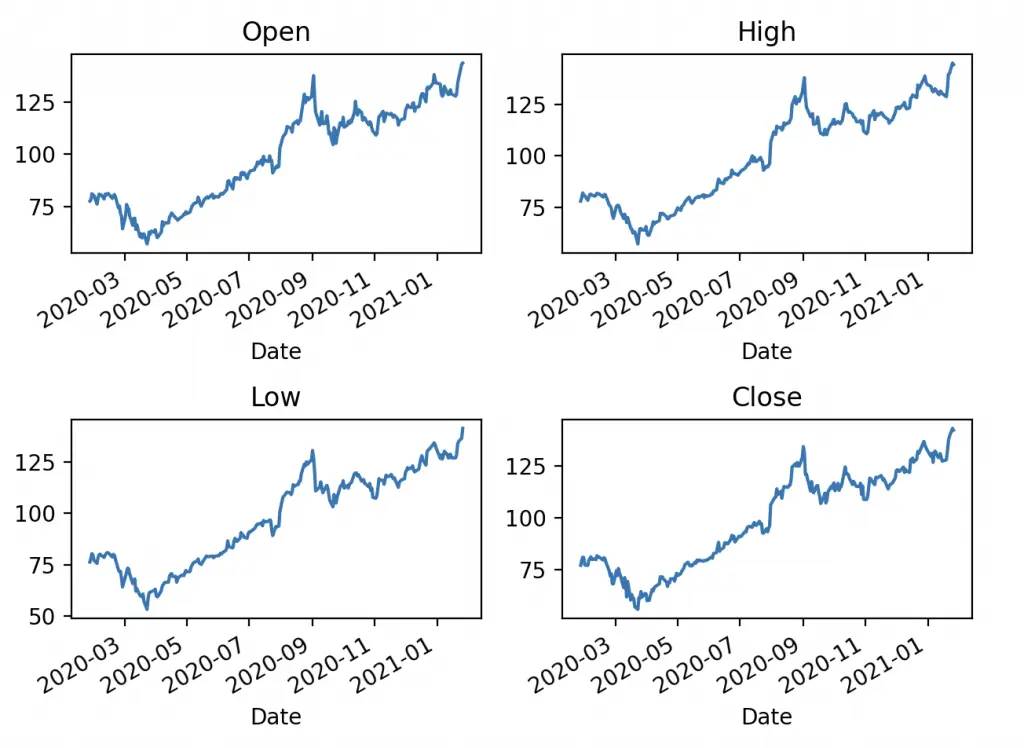
data['Low'].plot(ax=ax[1, 0], title="Low")

data['Close'].plot(ax=ax[1, 1], title="Close")

plt.tight\_layout()

The **plt.subplots(2, 2)** creates a 2-by-2 grid of axis. You access the axis in **ax** by **ax[0, 0]** and similar for the other axis.

It will result in the following output (or something similar).



## Horizontal bar plot

A bar plot can be created as follows, where we also make a time interval of the data we want by using .loc[].

fig, ax = plt.subplots()

data['Volume'].loc['2020-07-01':'2020-08-15'].plot.barh(ax=ax)

Chart

Description automatically generated

## Lesson 5: Use Pandas Datareader to read data directly from API

In the last lesson we learned about how to visualize our financial data.

### In this lesson you will learn

How to use Pandas Datareader to read historical stock prices from API. Then how to list all Nasdaq ticker symbols and explore the world bank data and more.

### File: 05 - Pandas Datareader

### Video: <https://youtu.be/sgndYho8RyI>

We will start differently this time. We will use the [**Pandas Datareader**](http://pandas-datareader.readthedocs.io/) to read data directly from the free open **Yahoo! Finance API.**

This can be done as follows.

import pandas\_datareader as pdr

import datetime as dt

ticker = "AAPL"

start = dt.datetime(2019, 1, 1)

end = dt.datetime(2020, 12, 31)

data = pdr.get\_data\_yahoo(ticker, start, end)

Here we first import two libraries.

* **pandas\_datareader** The Pandas Datareader. If you do not have it installed already in your Jupyter Notebook you can do that by entering this in a cell  **!pip install pandas\_datareader** and execute it.
* **datetime** This is a default library and represents a date and time. We only use it for the date aspects.

The following lines.

* **ticker = “AAPL”** The ticker we want data from. You can use any ticker you want. In this course we have used the ticker for [**Apple**](http://apple.com/) (**AAPL**).
* **start = dt.datetime(2019, 1, 1)** Is the starting day we want historic stock price data.
* **end = dt.datetime(2020, 12, 31)** The end day.
* **data = pdr.get\_data\_yahoo(ticker, start, end)** This is the magic that uses**Pandas Datareader** (**pdr**) to get data from the **Yahoo! Finance API**. It returns a DataFrame as we know it from previous lessons.

You can explore the advantage of using Pandas Datareader.

To test if we have the “pandas-datareader” and if it works fine. We do,

# Install using pip

# pip install pandas-datareader

If you have no “pandas-datareader” installed, you need to uncomment the two lines above. Then you can run the following code. It reads 5-years of 10-year constant maturity yields on U.S. government bonds.

import pandas\_datareader as pdr

#This example reads 5-years of 10-year constant maturity yields on U.S. government bonds.

data = pdr.get\_data\_fred('GS10')

We can then check if our data read is successful by suing head() and tail() functions

data.head()

data.tail()

Now,

### Read data from Yahoo Finance

There are two ways you get data from Yahoo finance:

1. Use pandas\_datareader.

import pandas\_datareader.data as web

df = web.DataReader('AAPL', 'yahoo', start='2019-09-10', end='2019-10-09')

This code uses pandas\_datareader.data function read data from Yahoo finance service API.

It takes 4 parameters:

* 1. Ticker: AAPL for Apple;
  2. Data source: Yahoo;
  3. Start date: '2019-09-10'. Notice the format.
  4. Ending date: '2019-10-09'.

1. Use pandas\_datareader.get\_data\_yahoo. pandas\_datareader.get\_data\_yahoo() is a function from the pandas-datareader library used to retrieve historical price data for a specific stock or index from Yahoo Finance. It returns a pandas dataframe object containing stock data such as open, high, low, close, and volume data.

import datetime as dt

# Ticker is the symbol of a company on the Stock market. AAPL is Apple

ticker = "AAPL"

start = dt.datetime(2019, 1, 1)

end = dt.datetime(2020, 12, 31)

data = pdr.get\_data\_yahoo(ticker, start, end)

Remember get data from internet may have many problems such as connection problem, source service is down, wrong API parameters, session expired, etc.

The solution is to use alternative services.

## Alternative source

Let us use yfinance as an alternative to get AAPL data.

import yfinance as yf

ticker = "AAPL"

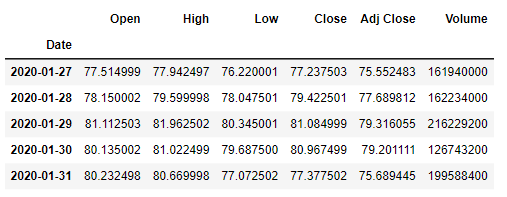
start\_date = "2020-01-27"

end\_date = "2021-01-27"

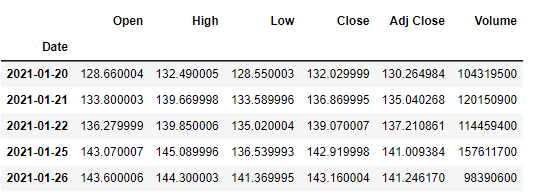
data = yf.download(ticker, start=start\_date, end=end\_date)

Notice that the data is now hold “AAPL” market data in DataFrame.

data.head()



data.tail()

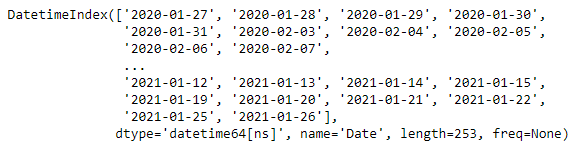


You can see the contents are the same with what we have in our CSV file.

You can run data.index to check the time.

data.index

Will show that the index is already a **DatetimeIndex**.



data.dtypes

Open float64

High float64

Low float64

Close float64

Adj Close float64

Volume int64

dtype: object

## Read historic stock prices data from Stooq

Another open source you can get historic stock prices is from **Stooq**. Stooq is a website that provides historical financial data including OHLCV (Open, High, Low, Close, Volume) data on over 21,000 global securities and ETFs, data on global indices, commodities, and bonds, 1980 currency pairs and 132 cryptocurrencies. the website is managed by a Polish firm, historical stock prices in US dollars can be downloaded with symbols familiar to US stock traders and analysts.

Let import data form Stooq.

data2 = pdr.get\_data\_stooq(ticker, start)

There is no error. We can check the data we get from the source. The most common function we can use are:

**.head(), .info(), .shape, .describe(), .columns, .index, .values**

A nice feature is to collect all the **Nasdaq** symbols.

nasdaq\_sym = pdr.get\_nasdaq\_symbols()

Now nasdaq\_sym holds all the symbols from Nasdaq. It is a DataFrame. We can explore it in a bit detail.

# shape returns the number of rows and columns of the DataFrame

nasdaq\_sym.shape

There are 11717 symbols at the time of writing (**11717 rows and 11 columns).** It is indexed by **Symbol.** So, we can access individual rows by its index (symbol).

nasdaq\_sym.loc['AAPL'] could display the details of Apple company. However, how can we find apple company symbol?

We can use string comparison function provided by Pandas. The following code does that,

# find company Apple's symbol using many str functions such as .str.contains("xxxx")

Name = nasdaq\_sym["Security Name"].str.contains("Apple")

nasdaq\_sym[Name]

Now, nasdaq\_sym[Name] will hold sub- DataFrame that only contains “Apple” in "Security Name".

A screenshot of a computer

Description automatically generated with medium confidence

We can see there are two records have Apple. If want to be specific, we can do.

Name = nasdaq\_sym["Security Name"].str.contains("Apple Inc")

nasdaq\_sym[Name]

Now the result will be only one with symbol, “AAPL”.

We can use this symbol as index to get apple’s Nasdaq records.

nasdaq\_sym.loc['AAPL']

Graphical user interface, text, application

Description automatically generated

## Lesson 6: Calculate Volatility and Moving Average

In the last lesson we learned how to read directly from **Yahoo! Finance API** and other **API**s with **Pandas Datareader**.

### In this lesson you will learn

How to do calculations and how to visualize some calculation results. In Finance Risk analysing,, like many other data analyses, we need to calculate many financial terms like **daily returns** and **daily log returns**, **volatility,** **moving average**: **Simple Moving Average (MA)** and **Exponential Moving Average (EMA)**, etc.

### File: 06 - Simple Calculations - Volatility, SMA, and EMA

### Video: <https://youtu.be/bceaTUXoqY8>

In this lesson we will use the **CSV**-file approach to get historic stock prices. Feel free to use the **Pandas Datareader** we explored in the last lesson.

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib notebook

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

Notice that we also import Matplotlib as we will use it to visualize our findings.

In this lesson we explore the difference between daily simple returns and daily log returns. Shortly explained, the log returns have the advantage that you can add them together, while this is not the case for simple returns. Therefore, the log returns are used in most financial analysis.

Here are some basic calculations you suppose to know. If you don’t, [investopdia i](https://www.investopedia.com/financial-term-dictionary-4769738)s a good place to know the terms.

We will just mention a few that they are needed.

* + - 1. **Pct change** (Pct change) represents the degree of change over time. Increase = New Number - Original Number

% increase = Increase ÷ Original Number × 100.

* + - 1. **Single-period Return**:  , where:

= final value, including dividends and interest

= initial value

A return *R* over a period of time *t*  corresponds to a rate of return *r*:



A compound rate of return *r*:



* + - 1. **Log return**

The **logarithmic return** or [**continuously compounded return**](https://www.wikiwand.com/en/Continuous_compounding), also known as [force of interest](https://www.wikiwand.com/en/Compound_interest#Force_of_interest), is:



,

where:

 = logarithmic rate of return

t = length of time period

The difference between daily simple returns and daily log returns is the log returns have the advantage that you can add them together, while this is not the case for simple returns. Therefore, the log returns are used in most financial analysis.

* + - 1. The **simple moving average** **(SMA).**

The moving average is calculated by adding a stock's prices over a certain period and dividing the sum by the total number of periods.

The past five closing prices of stock ABC are $25.25, $25.50, $25.00, $24.90 and $26.80.

Therefore, the SMA is $25.49.

Because,

* + - 1. The **exponential moving average** **(EMA)** focuses more on recent prices than on a long series of data points, as the simple moving average required. To Calculate an EMA

**Current EMA = ((Price(current) - previous EMA) X multiplier) + previous EMA**

The market **volatility** is the variation (Change of the return). It is normally calculated by the **standard deviation**.

Text

Description automatically generated

We start with calculate the Log return on “close”. To calculate the **daily log returns** we need the **NumPy** library. For the purpose here, we will not explore the depths of **NumPy**, all we need is to apply the **log**-function on a full column in our **DataFrame**.

import numpy as np

data['Log returns'] = np.log(data['Close']/data['Close'].shift())

The above code creates a column called **Log returns** with the daily **log return** of the **Close** price. This also creates a column called **Log returns** with the daily log return of the **Close** price. It can be seen by,

# check data

data.head()

Table

Description automatically generated

## Calculate market volatility

The market volatility is the variation (Change of the return). It is normally calculated by the standard deviation.

data['Log returns'].std()

It gives us, 0.029763804543855122

The above gives the **daily standard deviation**. The volatility is generally defined as the **annualized standard deviation**. Using the above formula we can calculate it as follows.

volatility = data['Log returns'].std()\*252\*\*.5

print("volatility:", volatility)

This will print:

volatility: 0.47248574936504945

We can visualize Apple (AAPL ticket number) stock volatility,

str\_vol = str(round(volatility, 4)\*100)

fig, ax = plt.subplots()

data['Log returns'].hist(ax=ax, bins=50, alpha=0.6, color='b')

ax.set\_xlabel("Log return")

ax.set\_ylabel("Freq of log return")

existential("AAPL volatility: " + str\_vol + "%")

We will see,

Chart, histogram

Description automatically generated

## Understand price trend in the market

The **simple moving average (SMA)** and **The exponential moving average (EMA)** are the two major measurements used to measure market price trend direction over a period of time.

The **simple moving average** (SMA) can be calculated by using a method **rolling(…)**.

data['SMA10'] = data['Close'].rolling(10).mean()

Where **rolling(10).mean()** will take the last 10 entries (rows) and apply **mean()**on it. This calculates the simple moving average with period 10.

An **exponential moving average** (EMA) can be calculated as follows.

data['EMA10'] = data['Close'].ewm(span=10, adjust=False).mean()

You can check above code calculation by,

data.head(20)

Thinking why I used 20 in the head function?

## Visualize market volatility

Both moving average and exponential moving average can be visualized as follows.

fig, ax = plt.subplots()

data[['MA10', 'EMA10']].loc['2020-12-01':].plot(ax=ax)

data['Close'].loc['2020-12-01':].plot(ax=ax, alpha=0.25)

Chart, line chart

Description automatically generated

What is the third line represent?

## Lesson 7: Calculate MACD and Stochastic Oscillator

In the last lesson we learned how to calculate the volatility of a stock. Also, the Simple Moving Average (MA) and Exponential Moving Average (EMA).

### In this lesson you will learn

How to calculate and visualize the MACD and the Stochastic Oscillator.

### File: 07 - MACD and Stochastic

### Video: https://youtu.be/jddmdKDq\_VQ

### MACD

I suppose you understand the meaning of **MACD.** If you don’t know please read from here:

[MACD Indicator Explained, with Formula, Examples, and Limitations (investopedia.com)](https://www.investopedia.com/terms/m/macd.asp)

Moving average convergence/divergence (MACD, or MAC-D) is a [trend-following](https://www.investopedia.com/terms/t/trendtrading.asp) [momentum](https://www.investopedia.com/terms/m/momentum.asp) indicator that shows the relationship between two [exponential moving averages (EMAs)](https://www.investopedia.com/terms/e/ema.asp) of a [security’s](https://www.investopedia.com/terms/s/security.asp) price. The MACD line is calculated by subtracting the 26-period EMA from the 12-period EMA.

## MACD Formula

**MACD=12-Period EMA − 26-Period EMA**

subtracting the long-term EMA (26 periods) from the short-term EMA (12 periods).

**Plot**

The result of that calculation is the MACD line. A nine-day EMA of the MACD line is called the signal line, which is then plotted on top of the MACD line, which can function as a trigger for buy or sell signals. Traders may buy the security when the MACD line crosses above the signal line and sell—or short—the security when the MACD line crosses below the signal line.

Let’s get started.

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib notebook

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

First we want to calcite the [MACD](https://www.investopedia.com/terms/m/macd.asp).

The calculation (12-26-9 MACD (default)) is defined as follows.

* MACD=12-Period EMA − 26-Period EMA
* Singal line 9-Perioed EMA of MACD

Where EMA is the Exponential Moving Average we learned about in the last lesson.

exp1 = data['Close'].ewm(span=12, adjust=False).mean()

exp2 = data['Close'].ewm(span=26, adjust=False).mean()

data['MACD'] = exp1 - exp2

data['Signal line'] = data['MACD'].ewm(span=9, adjust=False).mean()

The above code will add two extra columns into the data DataFrame.

Now the data will look like,

Table

Description automatically generated

We can plot them into a graph,

fig, ax = plt.subplots()

data[['MACD', 'Signal line']].plot(ax=ax)

data['Close'].plot(ax=ax, alpha=0.25, secondary\_y=True)

### Stochastic oscillator

Again, it is assumed that you understand the concept of the **stochastic oscillator.** A **stochastic oscillator** is a momentum indicator comparing a particular [closing price](https://www.investopedia.com/terms/c/closingprice.asp) of a security to a range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a [moving average](https://www.investopedia.com/terms/m/movingaverage.asp) of the result. It is used to generate [overbought](https://www.investopedia.com/terms/o/overbought.asp) and oversold trading signals, utilizing a 0–100 bounded range of values.

* Lagging indicator
* <https://www.investopedia.com/terms/s/stochasticoscillator.asp>

## Formula for the Stochastic Oscillator

Text

Description automatically generated

**Calculations**

* 14-high (H14): Maximum of last 14 trading days
* 14-low (L14): Minimum of last 14 trading days
* %K: (Last close - 14-low)\*100 / (14-high - 14-low)
* %D: Simple Moving Average of %K

Here is the code,

high14 = data['High'].rolling(14).max()

low14 = data['Low'].rolling(14).min()

data['%K'] = (data['Close'] - low14)\*100/(high14 - low14)

data['%D'] = data['%K'].rolling(3).mean()

The code above will add two extra columns %K and %D in the DataFrame, they can be checked,

data.tail()

here are the results,

Table

Description automatically generated

We can plot them into a graph,

fig, ax = plt.subplots()

data[['%K', '%D']].loc['2020-11-01':].plot(ax=ax)

ax.axhline(80, c='r', alpha=0.3)

ax.axhline(20, c='r', alpha=0.3)

data['Close'].loc['2020-11-01':].plot(ax=ax, alpha=0.3, secondary\_y=True)

here are the results.

Chart, line chart, histogram

Description automatically generated

Can you explain what are two red lines mean?

## Lesson 8: Export it all to an Excel sheet automatically with Python

In the last lesson we learned how to calculate and visualize the **MACD** and **Stochastic Oscillator**.

### In this lesson you will learn

In this lesson we will learn how to export all the data from Python into an Excel sheet. It will be in multiple sheets with coloured rows and charts. And of course, all automated from Python.

### File: 08 - Export to Excel

### Video: https://youtu.be/Ytbhysfp4Js

Apart from plotting results in graphs, many occasions you need to export your data to other system or share with colleagues or stick holders. The common way is to export data into CSV file in data projects. In Finance and Risk analysis the results can be exports to Excel spread sheet.

To demonstrate the process, we need some data. This data generally are manipulations (calculations) from an original data. We will use “AAPL.csv” file as the original data and our calculations including Moving Average (MA10), MACD, Single Line, %K and %D.

### Read original data

We use the following code to read data.

import pandas as pd

data = pd.read\_csv("AAPL.csv", index\_col=0, parse\_dates=True)

data.head()

### Moving Average

We generate **moving average** and add it into data.

data['MA10'] **=** data['Close'].rolling(10).mean()

data.tail()

Table

Description automatically generated

### MACD

We calculate MACD and Single line add them into data.

exp1 **=** data['Close'].ewm(span**=**12, adjust**=False**).mean()

exp2 **=** data['Close'].ewm(span**=**26, adjust**=False**).mean()

data['MACD'] **=** macd **=** exp1 **-** exp2

data['Signal line'] **=** exp3 **=** macd.ewm(span**=**9, adjust**=False**).mean()

data.tail()

### Table Description automatically generated

### Stochastic Oscillator

high14 **=** data['High'].rolling(14).max()

low14 **=** data['Low'].rolling(14).min()

data['%K'] **=** pct\_k **=** (data['Close'] **-** low14)**\***100**/**(high14 **-** low14)

data['%D'] **=** pct\_d **=** data['%K'].rolling(3).mean()

data.tail()

### 

### Time period

data **=** data.loc['2020-01-01':]

data **=** data.iloc[::**-**1]

data.head()

Adjust the time period we need. This is needed as many calculations does not have results (will not be available (**NaN**) or incorrect) for the first a few lines (like the first ten for MA10).

data **=** data.loc['2020-01-01':]

slides data from 2020-01-01 afterwards and data.iloc[::**-**1] reverses the data so to have the most recent data on the top of our Excel sheet.

### Generating Excel

To generating our Excel sheet we need the [**XlsxWriter**](https://xlsxwriter.readthedocs.io/).

If you don’t have it installed already you can install it by running this in a cell: **!pip install XlsxWriter**

The code that generated the Excel sheet as,

writer = pd.ExcelWriter("technical.xlsx",

engine='xlsxwriter',

date\_format = 'yyyy-mm-dd',

datetime\_format='yyyy-mm-dd')

workbook = writer.book

# Create a format for a green cell

green\_cell = workbook.add\_format({

'bg\_color': '#C6EFCE',

'font\_color': '#006100'

})

# Create a format for a red cell

red\_cell = workbook.add\_format({

'bg\_color': '#FFC7CE',

'font\_color': '#9C0006'

})

# \*\*

# \*\* MA

# \*\*

sheet\_name = 'MA10'

data[['Close', 'MA10']].to\_excel(writer, sheet\_name=sheet\_name)

worksheet = writer.sheets[sheet\_name]

# Set column width of Date

worksheet.set\_column(0, 0, 15)

for col in range(1, 3):

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=B2>=C2',

'format': green\_cell

})

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=B2<C2',

'format': red\_cell

})

# Create a new chart object.

chart1 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart1.add\_series({

'name': 'AAPL',

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 1, len(data), 1],

})

# Create a new chart object.

chart2 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart2.add\_series({

'name': sheet\_name,

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 2, len(data), 2],

})

# Combine and insert title, axis names

chart1.combine(chart2)

chart1.set\_title({'name': sheet\_name + " AAPL"})

chart1.set\_x\_axis({'name': 'Date'})

chart1.set\_y\_axis({'name': 'Price'})

# Insert the chart into the worksheet.

worksheet.insert\_chart('E2', chart1)

# \*\*

# \*\* MACD

# \*\*

sheet\_name = 'MACD'

data[['Close', 'MACD', 'Signal line']].to\_excel(writer, sheet\_name=sheet\_name)

worksheet = writer.sheets[sheet\_name]

# Set column width of Date

worksheet.set\_column(0, 0, 15)

for col in range(1, 4):

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=C2>=D2',

'format': green\_cell

})

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=C2<D2',

'format': red\_cell

})

# Create a new chart object.

chart1 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart1.add\_series({

'name': 'MACD',

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 2, len(data), 2],

})

# Create a new chart object.

chart2 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart2.add\_series({

'name': 'Signal line',

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 3, len(data), 3],

})

# Combine and insert title, axis names

chart1.combine(chart2)

chart1.set\_title({'name': sheet\_name + " AAPL"})

chart1.set\_x\_axis({'name': 'Date'})

chart1.set\_y\_axis({'name': 'Value'})

# To set the labels on x axis not on 0

chart1.set\_x\_axis({

'label\_position': 'low',

'num\_font': {'rotation': 45}

})

# Insert the chart into the worksheet.

worksheet.insert\_chart('F2', chart1)

# \*\*

# \*\* Stochastic

# \*\*

sheet\_name = 'Stochastic'

data[['Close', '%K', '%D']].to\_excel(writer, sheet\_name=sheet\_name)

worksheet = writer.sheets[sheet\_name]

# Set column width of Date

worksheet.set\_column(0, 0, 15)

for col in range(1, 4):

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=C2>=D2',

'format': green\_cell

})

# Create a conditional formatted of type formula

worksheet.conditional\_format(1, col, len(data), col, {

'type': 'formula',

'criteria': '=C2<D2',

'format': red\_cell

})

# Create a new chart object.

chart1 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart1.add\_series({

'name': '%K',

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 2, len(data), 2],

})

# Create a new chart object.

chart2 = workbook.add\_chart({'type': 'line'})

# Add a series to the chart.

chart2.add\_series({

'name': '%D',

'categories': [sheet\_name, 1, 0, len(data), 0],

'values': [sheet\_name, 1, 3, len(data), 3],

})

# Combine and insert title, axis names

chart1.combine(chart2)

chart1.set\_title({'name': sheet\_name + " AAPL"})

chart1.set\_x\_axis({'name': 'Date'})

chart1.set\_y\_axis({'name': 'Value'})

# To set the labels on x axis not on 0

chart1.set\_x\_axis({

'label\_position': 'low',

'num\_font': {'rotation': 45}

})

# Insert the chart into the worksheet.

worksheet.insert\_chart('F2', chart1)

# End of sheets

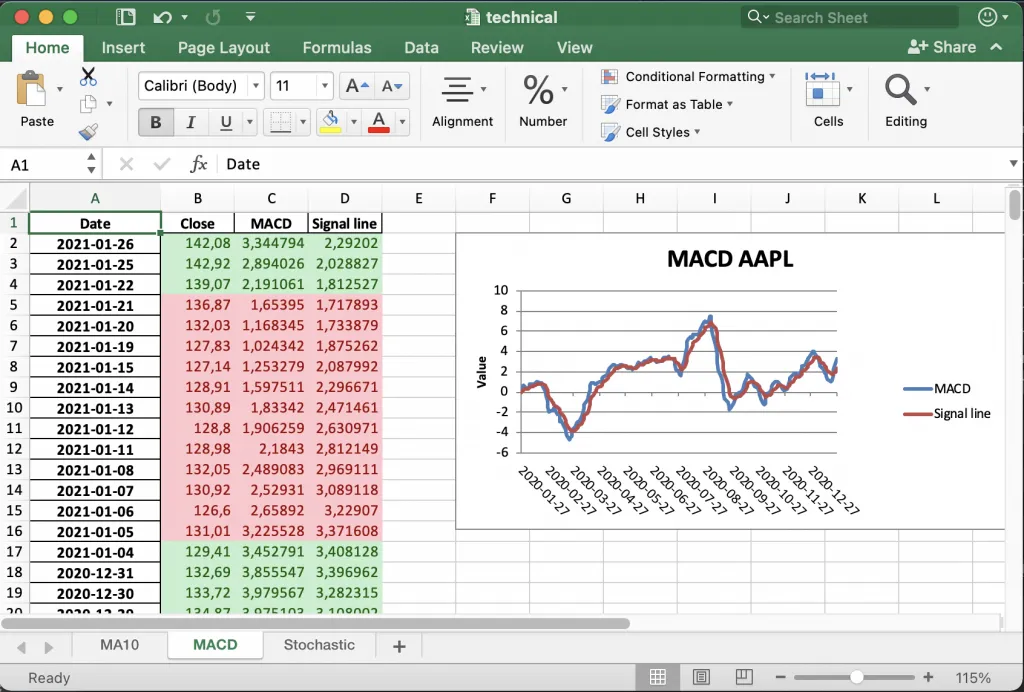
# Close

writer.close()

For a walkthrough of the code, please see the video to this lesson.

This will generate an Excel sheet in called **technical.xlsx**. It will contain 3 sheets (MA10, MACD, Stochastic Oscillator).

A sheet will look similar to this.



## 后 记

## 本实践手册简单介绍了用于风险分析和管理的最新技术手段。那就是Python语言和Python语言的相关包。Python语言是一个极其强大计算及高级语言。全面掌握它需要很长的时间。可是有谁在学习他用不到的语言和软件包？几乎所有科技工作者和数据分析师都是在需要的时候才学习所需工具。这就是本手册的目的。

## 其次， 本手册只是简单的介绍了使用Python进行数据分析的流程和基本工具。数据处理和分析的每一项工作都有很多不同的工具或者软件包来支持。举例来说，我们简单介绍了原始数据的导入。 那就是从一个CSV文件中读入并存储到Pandas的DataFrame中。其实， 就这样一个数据导入， 有很多数据源，CSV， JSON， XML，Excel， 数据库，… 还有网络挖掘等， 数据的存储除过DataFrame还有Array（矩阵）， dectinary（数据字典）， list（列表）等等。 不同的方式都有现成的工具包，函数来实现。所以， 我们的介绍只是一个抛砖引玉的过程。绝不是全部。

## 有了这个基本的训练，希望同学们不在对Python语言感到陌生和畏惧。 并希望它能够给您带来自信和快乐。从而激发您探索更多的方法和工具。

## 比如， 数据可视化Python有很多健全且完美的工具包。希望同学自己去探索。

## 2023. 2. 14英国剑桥