使用Python进行股票数据分析（一）

内容

[前言 2](#_Toc10524166)

[获取并可视化股票数据 2](#_Toc10524167)

[从quandl获取数据 2](#_Toc10524168)

[可视化股票数据 3](#_Toc10524169)

本译文主要介绍如何使用Python进行股票数据的分析，省略了原文简介部分。原文网址为：<https://ntguardian.wordpress.com/2018/07/17/stock-data-analysis-python-v2/>

# 前言

我将使用quandl和pandas\_datareader两个工具包，Anaconda并没有安装这两个包。运行下面的两个命令来安装它们：

conda install quandl

conda install pandas-datareader

# 获取并可视化股票数据

## 从quandl获取数据

在分析股票数据之前，我们需要先获得数据并转化成可用的格式。可以从[雅虎财经](https://finance.yahoo.com/)，[谷歌财经](http://finance.google.com/)或许多其他的地方获得股票数据。目前，我推荐使用quandl， 它是社区维护的经融和经济数据提供商。（[雅虎财经](https://finance.yahoo.com/)曾经提供高质量的股票数据，但是从2017年起，其API便不再维护，也不再能从它获得可信的数据了，详情请参考<https://quant.stackexchange.com/questions/35019/is-yahoo-finance-data-good-or-bad-now>）

默认情况下，quandl的get函数会返回一个包含所请求数据的Pandas DataFrame。

import pandas as pd

import quandl

import datetime

# We will look at stock prices over the past year, starting at January 1, 2016

start = datetime.datetime(2016,1,1)

end = datetime.date.today()

# Let's get Apple stock data; Apple's ticker symbol is AAPL

# First argument is the series we want, second is the source ("yahoo" for Yahoo! Finance), third is the start date, fourth is the end date

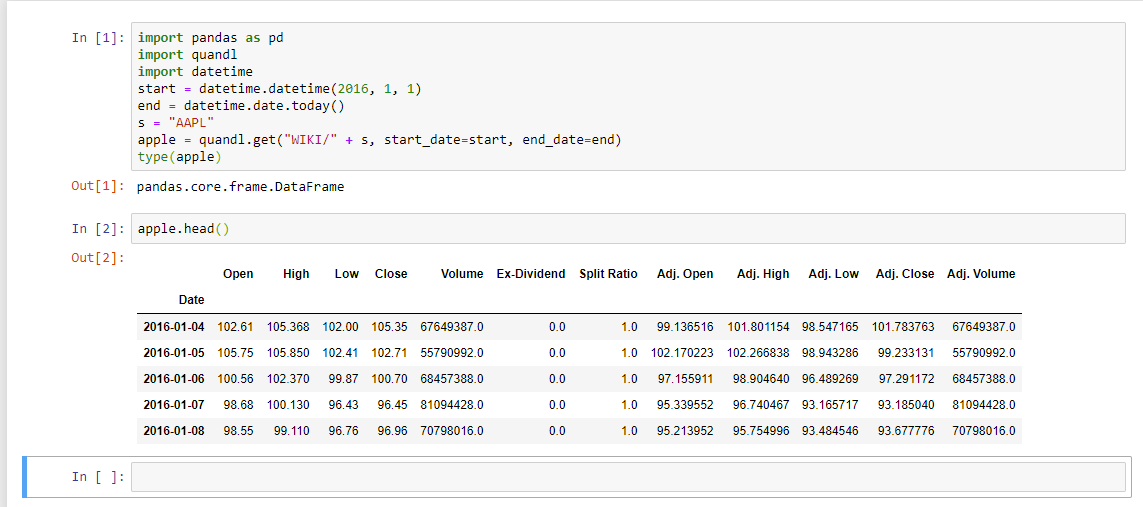
s = "AAPL"

apple = quandl.get("WIKI/" + s, start\_date=start, end\_date=end)

type(apple)

apple.head()

（译者注：上述程序在Jupyter Notebook中的运行情况如下图）

我们简单的讨论一下，Open是交易日开始时的股票价格（不需要是上一个交易日的收盘价），High是交易日中的最高股价，Low是交易日中的最低股价，Close是交易日结束时的股价。Volume指有多少交易了股票。Adjusted Prices（例如调整后的收盘价）是因为公司行为而调整后的股价。虽然认为股价主要由交易员设定，但股票拆分（公司使每只现存股票分成两个并使价格减半）和分红（支付公司的每股利润）也会影响股价，也需要加以考虑。

## 可视化股票数据

现在我们获得了想要可视化的股票数据了。首先，我将演示怎么使用matplotlib包来实现。注意到apple DataFrame对象有一个方便的方法，plot()，使得绘图更加方便。

import matplotlib.pyplot as plt # Import matplotlib

# This line is necessary for the plot to appear in a Jupyter notebook

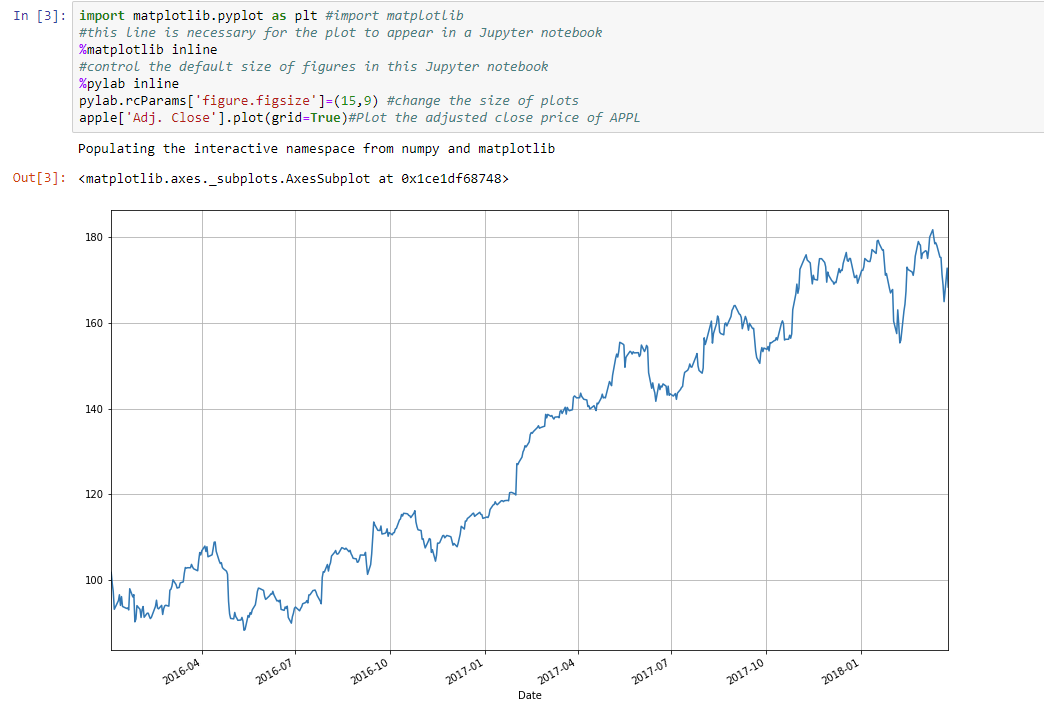
%matplotlib inline

# Control the default size of figures in this Jupyter notebook

%pylab inline

pylab.rcParams['figure.figsize'] = (15, 9) # Change the size of plots

apple["Adj. Close"].plot(grid = True) # Plot the adjusted closing price of AAPL

运行结果如下一个折线图是可以的，但是每天至少涉及4个变量（开盘价，收盘价，最高价，最低价），我们希望有某种可视化方法能够查看4个变量而不需要绘制4个单独的折线。金融数据通常绘制成日本蜡烛图，之所以这么叫是因为它是18世纪的日本大米交易者创建的。虽然需要一些努力，但是这种图可以用matplotlib创建。

我创建了一个函数并使用它来绘制我们的股票数据，也欢迎你使用它从pandas DataFrame中更方便地绘制蜡烛图。（代码是基于这个[例子](http://matplotlib.org/examples/pylab_examples/finance_demo.html) ， 你也可以在[这里](http://matplotlib.org/api/finance_api.html)查看相关函数的文档）[译者注：mpl\_finance包已经从matplotlib移除，所以需要单独安装，请到<https://github.com/matplotlib/mpl_finance> 下载并在下载目录下运行python setup.py install 命令来安装]

from matplotlib.dates import DateFormatter, WeekdayLocator,\

    DayLocator, MONDAY

from mpl\_finance import candlestick\_ohlc

def pandas\_candlestick\_ohlc(dat, stick="day", adj=False, otherseries=None):

    """

    :param dat: pandas DataFrame object with datetime64 index, and float columns "Open", "High", "Low", and "Close", likely created via DataReader from "yahoo"

    :param stick: A string or number indicating the period of time covered by a single candlestick. Valid string inputs include "day", "week", "month", and "year", ("day" default), and any numeric input indicates the number of trading days included in a period

    :param adj: A boolean indicating whether to use adjusted prices

    :param otherseries: An iterable that will be coerced into a list, containing the columns of dat that hold other series to be plotted as lines

    This will show a Japanese candlestick plot for stock data stored in dat, also plotting other series if passed.

    """

    mondays = WeekdayLocator(MONDAY)        # major ticks on the mondays

    alldays = DayLocator()                  # minor ticks on the days

    dayFormatter = DateFormatter('%d')      # e.g., 12

    # Create a new DataFrame which includes OHLC data for each period specified

    # by stick input

    fields = ["Open", "High", "Low", "Close"]

    if adj:

        fields = ["Adj. " + s for s in fields]

    transdat = dat.loc[:,fields]

    transdat.columns = pd.Index(["Open", "High", "Low", "Close"])

    if (type(stick) == str):

        if stick == "day":

            plotdat = transdat

            stick = 1 # Used for plotting

        elif stick in ["week", "month", "year"]:

            if stick == "week":

                transdat["week"] = pd.to\_datetime(transdat.index).map(lambda x: x.isocalendar()[1]) # Identify weeks

            elif stick == "month":

                transdat["month"] = pd.to\_datetime(transdat.index).map(lambda x: x.month) # Identify months

            transdat["year"] = pd.to\_datetime(transdat.index).map(lambda x: x.isocalendar()[0]) # Identify years

            grouped = transdat.groupby(list(set(["year",stick]))) # Group by year and other appropriate variable

            plotdat = pd.DataFrame({"Open": [], "High": [], "Low": [], "Close": []}) # Create empty data frame containing what will be plotted

            for name, group in grouped:

                plotdat = plotdat.append(pd.DataFrame({"Open": group.iloc[0,0],

                                            "High": max(group.High),

                                            "Low": min(group.Low),

                                            "Close": group.iloc[-1,3]},

                                           index = [group.index[0]]))

            if stick == "week": stick = 5

            elif stick == "month": stick = 30

            elif stick == "year": stick = 365

    elif (type(stick) == int and stick >= 1):

        transdat["stick"] = [np.floor(i / stick) for i in range(len(transdat.index))]

        grouped = transdat.groupby("stick")

        plotdat = pd.DataFrame({"Open": [], "High": [], "Low": [], "Close": []}) # Create empty data frame containing what will be plotted

        for name, group in grouped:

            plotdat = plotdat.append(pd.DataFrame({"Open": group.iloc[0,0],

                                        "High": max(group.High),

                                        "Low": min(group.Low),

                                        "Close": group.iloc[-1,3]},

                                       index = [group.index[0]]))

    else:

        raise ValueError('Valid inputs to argument "stick" include the strings "day", "week", "month", "year", or a positive integer')

    # Set plot parameters, including the axis object ax used for plotting

    fig, ax = plt.subplots()

    fig.subplots\_adjust(bottom=0.2)

    if plotdat.index[-1] - plotdat.index[0] < pd.Timedelta('730 days'):

        weekFormatter = DateFormatter('%b %d')  # e.g., Jan 12

        ax.xaxis.set\_major\_locator(mondays)

        ax.xaxis.set\_minor\_locator(alldays)

    else:

        weekFormatter = DateFormatter('%b %d, %Y')

    ax.xaxis.set\_major\_formatter(weekFormatter)

    ax.grid(True)

    # Create the candelstick chart

    candlestick\_ohlc(ax, list(zip(list(date2num(plotdat.index.tolist())), plotdat["Open"].tolist(), plotdat["High"].tolist(),

                      plotdat["Low"].tolist(), plotdat["Close"].tolist())),

                      colorup = "black", colordown = "red", width = stick \* .4)

    # Plot other series (such as moving averages) as lines

    if otherseries != None:

        if type(otherseries) != list:

            otherseries = [otherseries]

        dat.loc[:,otherseries].plot(ax = ax, lw = 1.3, grid = True)

    ax.xaxis\_date()

    ax.autoscale\_view()

    plt.setp(plt.gca().get\_xticklabels(), rotation=45, horizontalalignment='right')

    plt.show()

pandas\_candlestick\_ohlc(apple, adj=True, stick="month")



在蜡烛图中，黑色的蜡烛图标识这天的收盘价高于开盘价（赢利），而红色蜡烛表示这天的开盘价高于收盘价（亏损）。上影线和下影线表示最高价和最低价，主体表示开盘价和收盘价（颜色决定哪端是开盘价，哪端是收盘价）。蜡烛图在金融领域很流行而且[技术分析](https://en.wikipedia.org/wiki/Technical_analysis)中的一些策略根据蜡烛的形状，颜色，位置来做交易决定。今天我不会讲解这些策略。

我们可能想要把多个金融工具绘制到一起；我们可能想要比较股票，把它们和市场比较，或者查看其他证券例如[ETFs](https://en.wikipedia.org/wiki/Exchange-traded_fund)。后面我们也会查看怎样针对某个指标，例如移动平均线（moving average）去绘制一个金融工具。对于这种，你应该使用折线图而非蜡烛图。（你怎么才能绘制多个蜡烛图，一个在另一个上面而不弄乱图表?）

下面，我获取了其他几个技术公司的股票数据并且把他们调整后的收盘价绘制在了一起。

microsoft, google = (quandl.get("WIKI/" + s, start\_date=start, end\_date=end) for s in ["MSFT", "GOOG"])

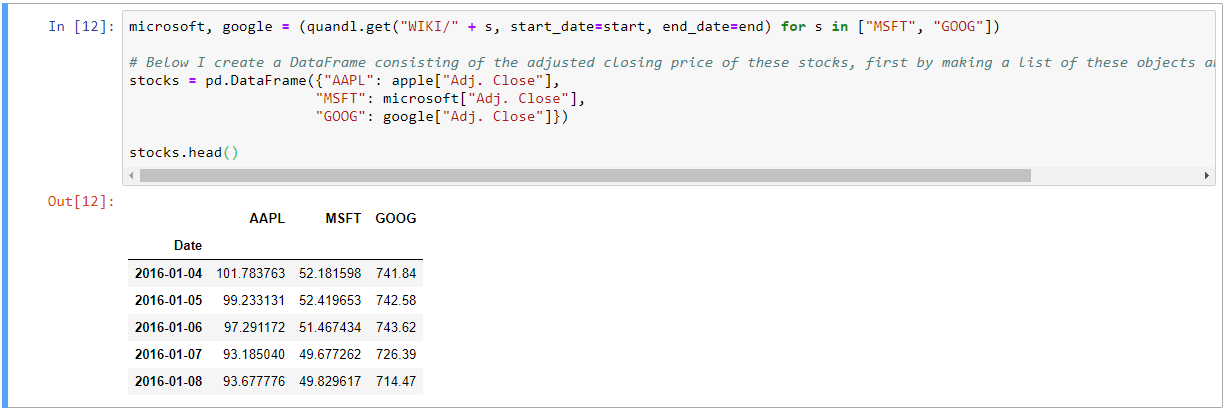
# Below I create a DataFrame consisting of the adjusted closing price of these stocks, first by making a list of these objects and using the join method

stocks = pd.DataFrame({"AAPL": apple["Adj. Close"],

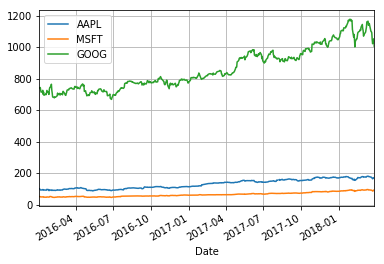
                      "MSFT": microsoft["Adj. Close"],

                      "GOOG": google["Adj. Close"]})

stocks.head()



stocks.plot(grid = True)



这幅图有什么问题呢？虽然绝对价格很重要（高价股很难购买，这不仅影响它们的波动性而且影响你交易股票的能力），但交易时，我们更关注资产的相对变化而不是它的绝对价格。谷歌的股票比微软或苹果的股票贵得多，而这种差异使得苹果和微软股票的波动性远低于它们实际的波动性（也就是说，他们的价格看着没有太多偏差）。

一种解决方案是在绘图时使用两种不同的比例；一种比例用于苹果和微软，另一种用于谷歌。

stocks.plot(secondary\_y = ["AAPL", "MSFT"], grid = True)



然而，更好的方案是绘制我们正在想要的信息：股票的收益。这涉及到把数据转换成某种对我们的目标更有用的东西。有多种转换可供我们使用。

一种转换是考虑自利息周期开始以来股票的收益。也就是说，我们如下绘图：

G:\latex.php.png

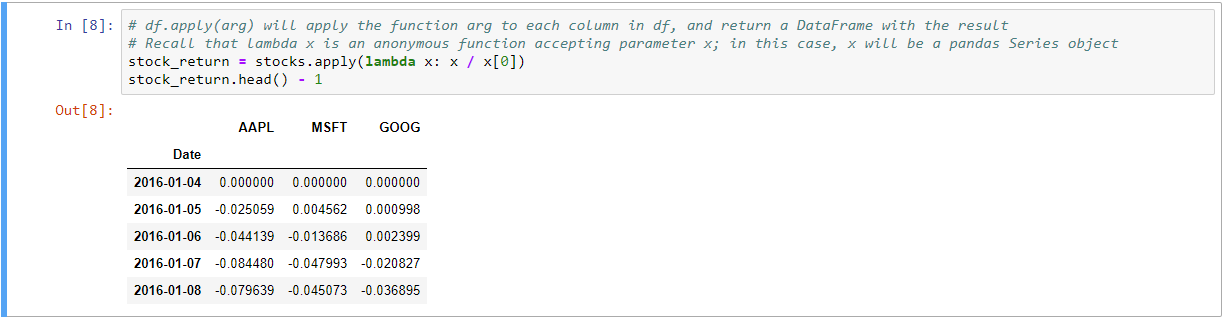
这就要求转换stocks对象中的数据，我接下来就会这样做。请注意，我使用了lamda函数，它允许我把一个定义好的小函数当做参数快速传递到另一个函数或者方法中（你可以从[这里](https://docs.python.org/3/reference/expressions.html#lambda)阅读更多关于lamda函数的信息）。

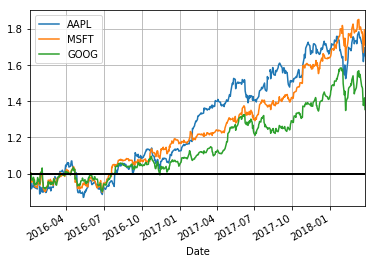
# df.apply(arg) will apply the function arg to each column in df, and return a DataFrame with the result

# Recall that lambda x is an anonymous function accepting parameter x; in this case, x will be a pandas Series object

stock\_return = stocks.apply(lambda x: x / x[0])

stock\_return.head() - 1

stock\_return.plot(grid = True).axhline(y = 1, color = "black", lw = 2)



这个图表更加有用。我们现在能够看到自此周期开始算起每只股票的盈利情况。此外，我们看到这些股票高度相关；他们通常朝同一方向移动，这一事实很难在其他图表中看到。

或者，我们可以绘制每只股票每天的变化情况。一种方法是在比较第t天和第t + 1天时，使用下面的公式绘制一只股票的增长率：

G:\latex.php1.png

但，变化可能以以下不同的方式理解：

G:\latex.php2.png

这些公式不同并可能导致不同的结论，但是有另一种方法来模拟一只股票的增长：使用对数差异。

G:\latex.php4.png

（这里log是自然对数，并且我们的定义不强依赖于是使用G:\latex.php5.png还是G:\latex.php6.png）使用对数差的优势是这种差可以被解释为股票的百分比变化，但不依赖于分数的分母。此外，对数差具有理想的性质：对数差的总和可以解释为汇总时间段的总变化（以百分比）（其他公式没有这种性质；他们会高估增长）。对数差也更加清晰地对应于股票价格在连续时间内的建模方式。

我们可以如下获得和绘制stocks中数据的对数差：

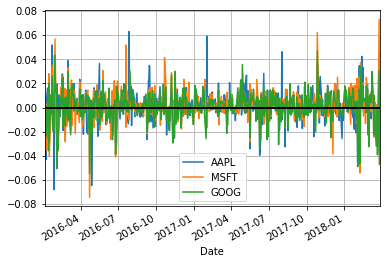
import numpy as np

stock\_change = stocks.apply(lambda x: np.log(x) - np.log(x.shift(1))) # shift moves dates back by 1.

stock\_change.head()



stock\_change.plot(grid = True).axhline(y = 0, color = "black", lw = 2)



你更喜欢哪种转换？看起来，自周期开始计算收益的方式使得问题中股票的整体趋势更加明显。但是，每天之间的变化是在对股票行为进行建模时实际考虑的更高级方法。所以他们不应该被忽视。我们经常想将股票的表现与整体市场的表现进行比较。[SPY](https://finance.yahoo.com/quote/SPY/)，是SPDR标准普尔500指数交易型开放式指数基金（ETF）的股票代码，是一只试图模仿标准普尔500股票指数构成的基金，因而代表了“市场”中的价值。

SPY数据不能从quandl免费获取，所以我将从雅虎财经中获取数据。（我没有选择。）

下面是我获得SPY数据并将它的表现跟我们的股票表现相比较。

#import pandas\_datareader.data as web    # Going to get SPY from Yahoo! (I know I said you shouldn't but I didn't have a choice)

#spyder = web.DataReader("SPY", "yahoo", start, end)    # Didn't work

#spyder = web.DataReader("SPY", "google", start, end)    # Didn't work either

# If all else fails, read from a file, obtained from here: http://www.nasdaq.com/symbol/spy/historical

spyderdat = pd.read\_csv("/home/curtis/Downloads/HistoricalQuotes.csv")    # Obviously specific to my system; set to

                                                                          # location on your machine

spyderdat = pd.DataFrame(spyderdat.loc[:, ["open", "high", "low", "close", "close"]].iloc[1:].as\_matrix(),

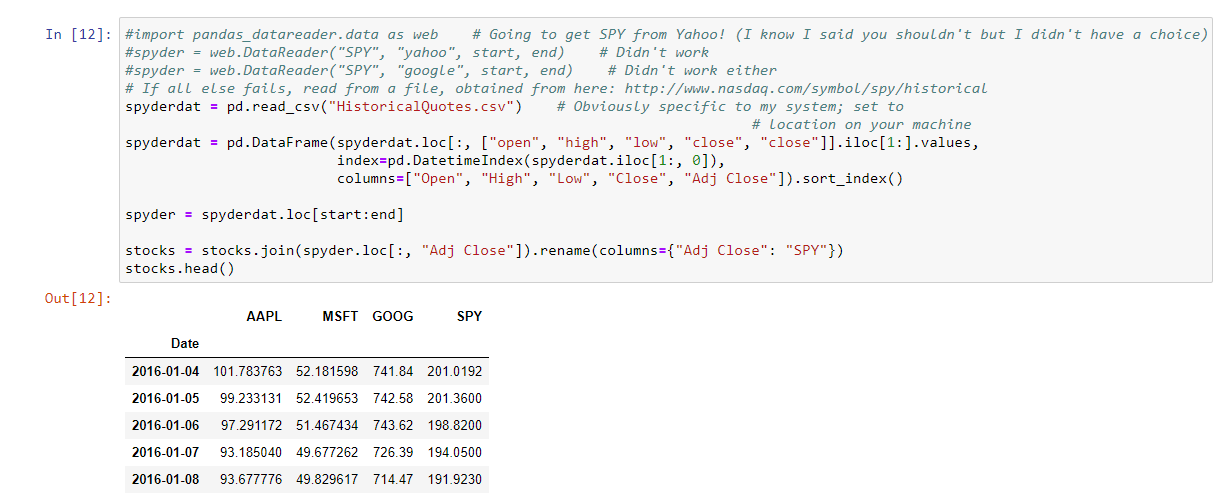
                         index=pd.DatetimeIndex(spyderdat.iloc[1:, 0]),

                         columns=["Open", "High", "Low", "Close", "Adj Close"]).sort\_index()

spyder = spyderdat.loc[start:end]

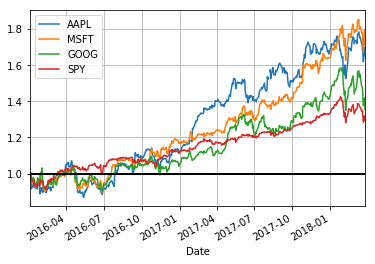
stocks = stocks.join(spyder.loc[:, "Adj Close"]).rename(columns={"Adj Close": "SPY"})

stocks.head()



stock\_return = stocks.apply(lambda x: x / x[0])

stock\_return.plot(grid = True).axhline(y = 1, color = "black", lw = 2)



stock\_change = stocks.apply(lambda x: np.log(x) - np.log(x.shift(1)))

stock\_change.plot(grid=True).axhline(y = 0, color = "black", lw = 2)

