Published in Towards Data Science

Stefano Basurto

Jan 7, 2020 · 7 min read ★ · Listen

TRADING-TOOLBOX

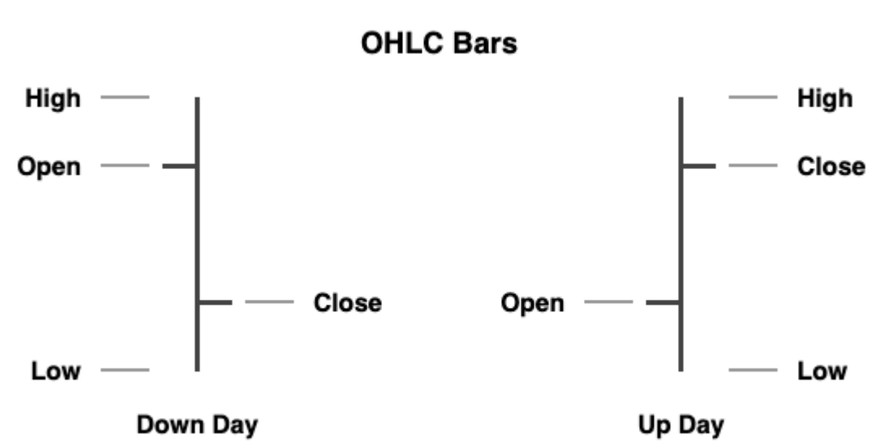
Python Trading Toolbox: introducing OHLC charts with Matplotlib

Unleashing the Power of OHLC data

Photo by Aditya Vyas on Unsplash

In other posts, we started exploring how to compute some basic indicators based on price (simple moving averages and other moving averages) and how to plot them on a chart together with the price. In those examples, we considered daily price data and used the closing price to represent each day of trading. Quite obviously, financial instruments trade throughout the whole day generating more than one price. The closing price is one of the most relevant prices but does not tell the whole story of what happened during the trading day.

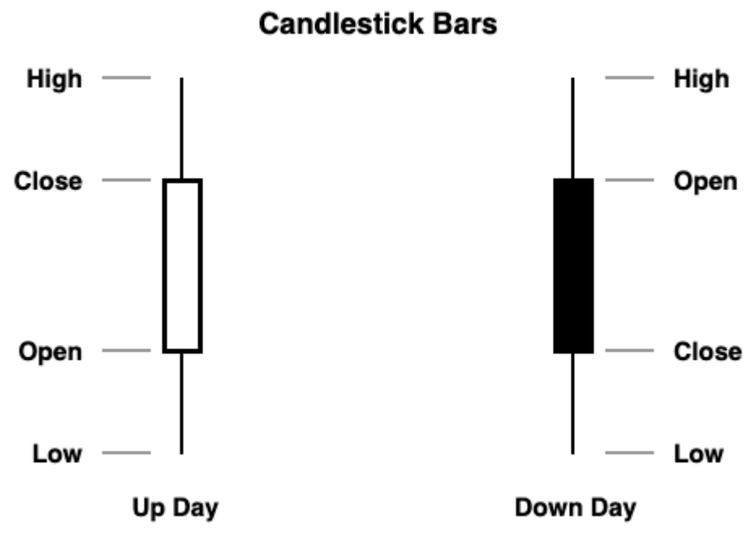
OHLC bars and bar charts are a traditional way to capture the range of prices of a financial instrument generated during the entire day of trading: for each single day, four prices are recorded: the opening price (**Open**), the highest price (**High**), the lowest price (**Low**), and the closing price (**Close**).



Bar charts are not limited to daily prices: weekly and monthly charts can be constructed using open, high, low and close for each period. They can be applied to intraday charts as well by using hourly bars, or bars for any desired interval (e.g. 30 minutes, 10 minutes, down to 1 minute).

Similarly to bar charts, candlestick charts are based on the Open, High, Low, and Close for each day, but use a different visual representation. The range between the Open and the Close is represented by a '*candle body*' — which takes different

colors depending on whether the Close is higher than the Open or the other way round (usually white and black). Highs and Lows are represented by '*candle wicks*' (called *shadows*), placed above and below the body respectively. The use of candlestick charting originates from Japan and is associated with a kind of analysis based on patterns.



Creating OHLC Bar Charts with Python

There are several good visualization resources that enable us to create bar and candlestick charts in Python. Two of the best are **Plot.ly** and **Bokeh**. Both solutions allow creating professionally looking **interactive charts**. On the other hand, **Matplotlib** focuses on **static charts** and is capable of producing beautiful publication-quality figures. That is usually my first port of call when I have to produce static charts.

While the Matplotlib library is one of those elements that make Python a great environment for data visualization, when it comes to OHLC financial charts it has so far performed below its true potential. The package that handles the drawing of OHLC and candlestick charts within Matplotlib is called *mpl-finance*, a module that used to be part of the main Matplotlib distribution until it was

declared deprecated and became available only as a separate package. That happened, I believe, for a good reason: *mpl-finance* is not particularly well integrated with *pandas* nor as easy to use as other plotting features of Matplotlib. More recently, it has found a new maintainer, Daniel Goldfarb, who is working at creating a new API for *mpl-finance* to make it more usable and consistent with *pandas* dataframes. The new version should be released at some point during 2020.

A glimpse into the near future

We can have a first look at how the upcoming version of *mpl-finance* is going to work and look like. To preview the new version (a pre-release, at the time of writing), you just need to run:

```
pip install mplfinance
```

Note the spelling of the upcoming version: there is no '-' nor '_' in the name, while the current version is installed as `mpl-finance` and imported (quite confusingly) as `mpl_finance`. The new version will put an end to this name mixup.

We can now create our first price bar charts, using the same data for the **SPY** *ETF* used in the first post of the series. You can download the CSV file here.

Typing:

```
import pandas as pd


datafile = 'SPY.csv'
data = pd.read_csv(datafile, index_col = 'Date')
data.index = pd.to_datetime(data.index) # Converting the dates from
string to datetime format


data
```

shows our OHLC price dataframe:

Date	Open	High	Low	Close	Adj Close	Volume
2014-08-20	198.119995	199.160004	198.080002	198.919998	180.141846	72763000
2014-08-21	199.089996	199.759995	198.929993	199.500000	180.667160	67791000
2014-08-22	199.339996	199.690002	198.740005	199.190002	180.386368	76107000
2014-08-25	200.139999	200.589996	199.149994	200.199997	181.301010	63855000
2014-08-26	200.330002	200.820007	200.279999	200.330002	181.418716	47298000
...	...	...	...	...	...	...
2019-08-13	287.739990	294.149994	287.359985	292.549988	292.549988	94299800
2019-08-14	288.070007	288.739990	283.760010	283.899994	283.899994	135622100
2019-08-15	284.880005	285.640015	282.390015	284.649994	284.649994	99556600
2019-08-16	286.480011	289.329987	284.709991	288.850006	288.850006	83018300
2019-08-19	292.190002	293.079987	291.440002	292.329987	292.329987	53571800

1258 rows × 6 columns

We can now import the newly installed *mpl-finance* library:

```
import mplfinance as mpf
```

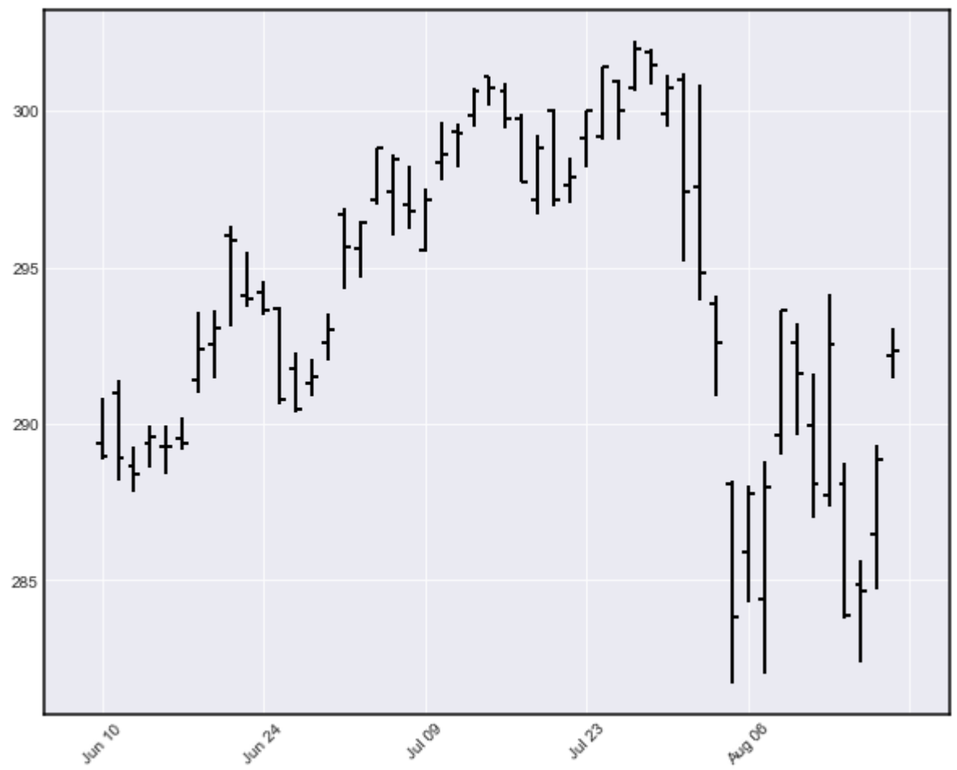
Creating a price bar chart (for the last 50 days of data) is as easy as:

```
mpf.plot(data[-50:], no_xgaps = True)
```

If you are not using Jupyter, do not forget to add the following line to visualize this and the next charts:

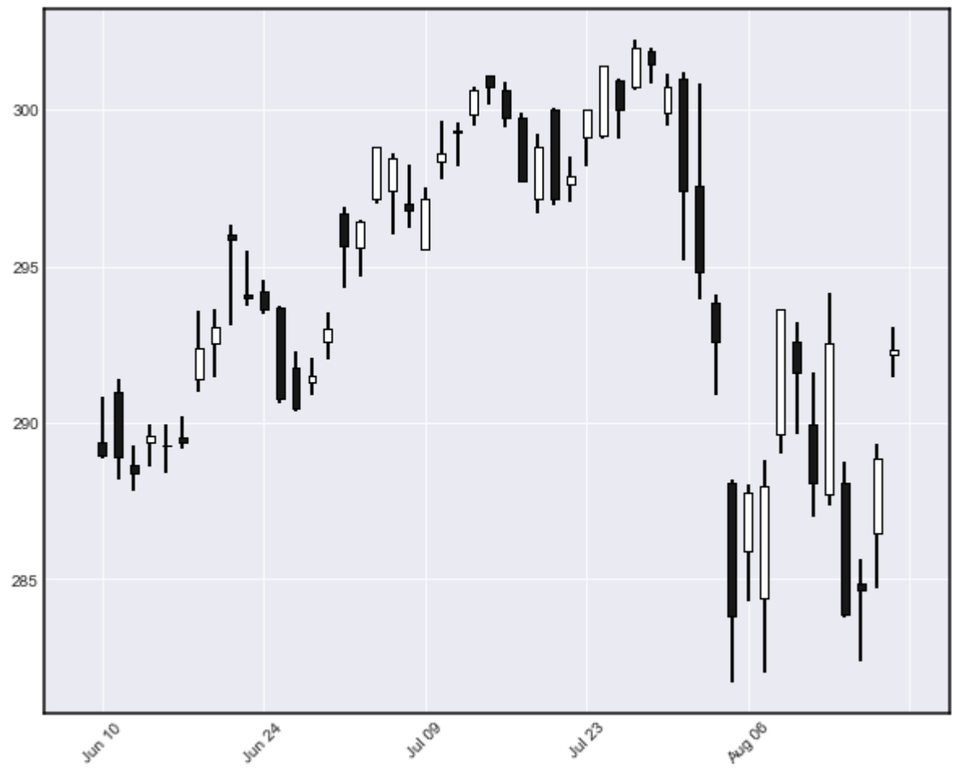
```
plt.show()
```

This is the result:



Creating a candlestick chart is equally straightforward:

```
mpf.plot(data[-50:], type='candlestick', no_xgaps = True)
```



The visual results look appealing. The `no_xgaps` option is a nifty feature that eliminates the gaps usually generated by days with no trading data (e.g. weekends and public holidays).

The current mpl-finance library

The current version of *mpl-finance* can be installed using:

```
pip install mpl-finance
```

or

```
conda install mpl-finance
```

if you are using *Conda* as a package manager. Compared to the version to be released, the current *mpl-finance* library requires some data manipulation to create a simple OHLC or candlestick chart. In particular:

- We need to present the data as a sequence of OHLC price sequences.
- The time and dates need to be expressly converted to a format that Matplotlib understands.

Luckily, all those operations will become obsolete once the new version is released, hopefully soon during 2020. For now, I just present my code to perform those tasks without going too much into detail. You can use it as it is:

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Loading data into dataframe:
datafile = 'SPY.csv'
data = pd.read_csv(datafile, index_col = 'Date')
# Converting the dates from string to datetime format:
data.index = pd.to_datetime(data.index)

# We need to exctract the OHLC prices into a list of lists:
dvalues = data[['Open', 'High', 'Low', 'Close']].values.tolist()

# Dates in our index column are in datetime format, we need to comvert them
# to Matplotlib date format (see https://matplotlib.org/3.1.1/api/dates_api.html):
pdates = mdates.date2num(data.index)

# If dates in our index column are strings instead of datetime objects, we should use:
# pdates = mpl.dates.datestr2num(data.index)

# We prepare a list of lists where each single list is a [date, open, high, low, close] sequence:
ohlc = [ [pdates[i]] + dvalues[i] for i in range(len(pdates)) ]
```

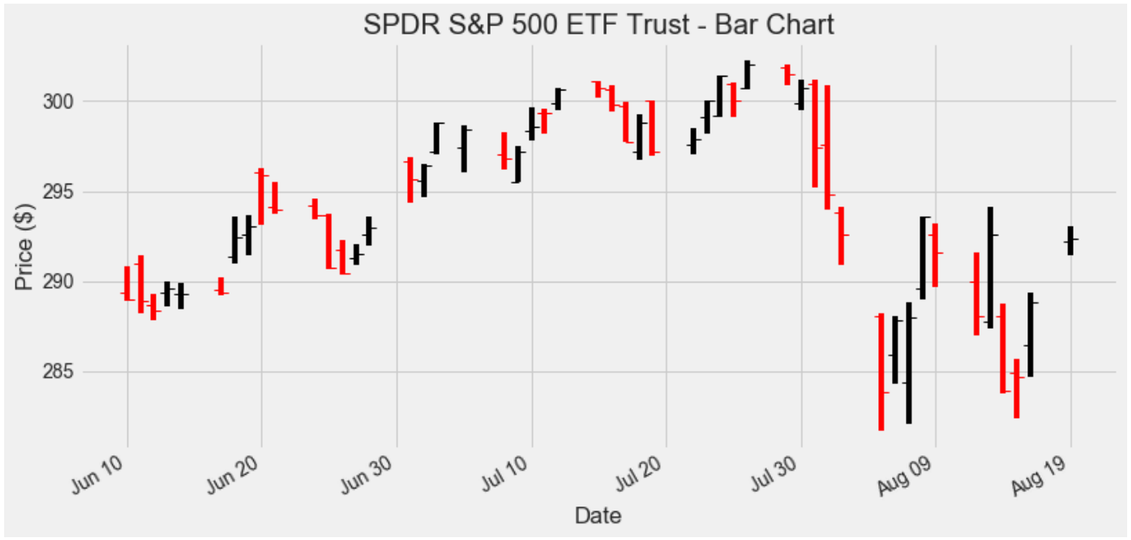
TTB03-ohlc_data.py hosted with ❤ by GitHub view raw

We can now use the data to plot a bar chart:

```
import mpl_finance as mpf # This is the old mpl-finance library - note the '_' in the library name

# We can now feed the ohlc matrix into mpl-finance to create a candle stick chart:

plt.style.use('fivethirtyeight')
fig, ax = plt.subplots(figsize = (12,6))

mpf.plot_day_summary_ohlc(ax, ohlc[-50:], ticksize = 5)

ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('SPDR S&P 500 ETF Trust - Bar Chart')

# Choosing to display the dates as "Month Day":
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d'))

# This is to automatically arrange the date labels in a readable way:
fig.autofmt_xdate()

# plt.show() # add this if you're not using Jupyter Notebook
```

TTB03-bar_chart.py hosted with ❤ by GitHub view raw

Which shows:



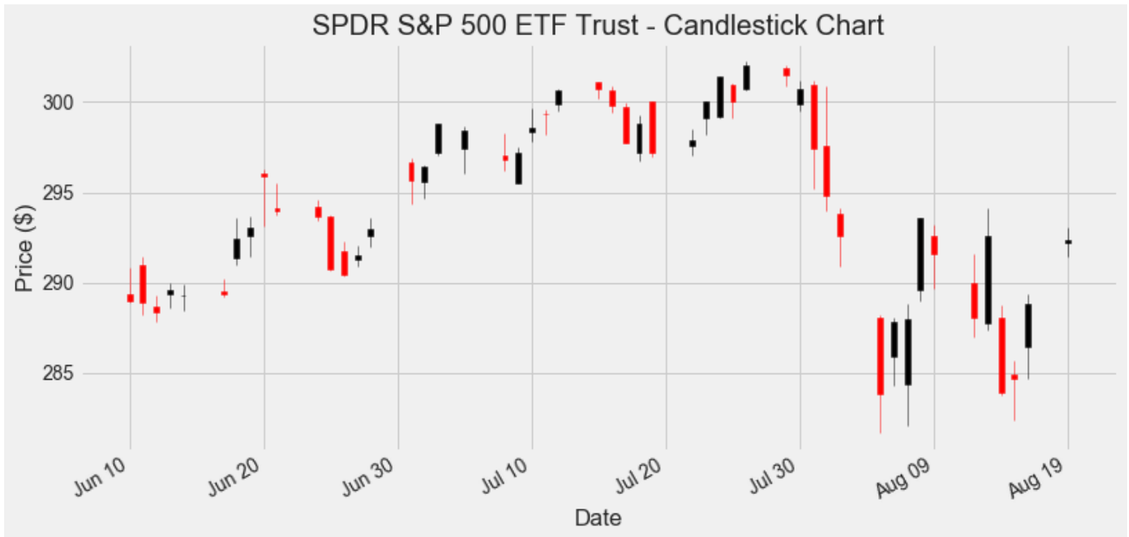
By default, bars with a Close higher than the Open are colored in black, while bars with a Close below the Open are colored in red. There are some visible horizontal gaps in the chart: they are generated by non-trading days (weekends and public holidays). Removing them requires some extra non-trivial filtering.

Similarly, a candlestick chart can be generated in the following way:

```
fig, ax = plt.subplots(figsize = (12,6))

mpf.candlestick_ohlc(ax, ohlc[-50:], width=0.4)
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('SPDR S&P 500 ETF Trust - Candlestick Chart')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d'))

fig.autofmt_xdate()

# plt.show() # add this if you're not using Jupyter Notebook
```

TTB03-candlesticks.py hosted with ❤ by GitHub view raw

Which produces:



Unleashing the Power of OHLC data

Using OHLC prices instead of just one single series as the Closing Price opens up a new world of possibilities: we can assess the range that prices covered during each trading day, observe the relationship of the Close versus the Open, check whether prices have risen above previous highs or fallen below previous lows and so on.

Here I will present a fairly simple example of a chart that makes use of High and Low prices as well as Close prices. This chart comes from a trading technique known as **Camelback Technique**. I am not interested in describing the trading technique in detail (plenty of information can be found on the web): we are just using the chart associated with it as a useful tool to spot existing trends.

We overlay the following moving averages on a daily price bar chart:

- 40-Day Simple Moving Average of the **Highs.**
- 40-Day Simple Moving Average of the **Lows.**
- 15-Day Exponential Moving Average of the **Closing** prices.

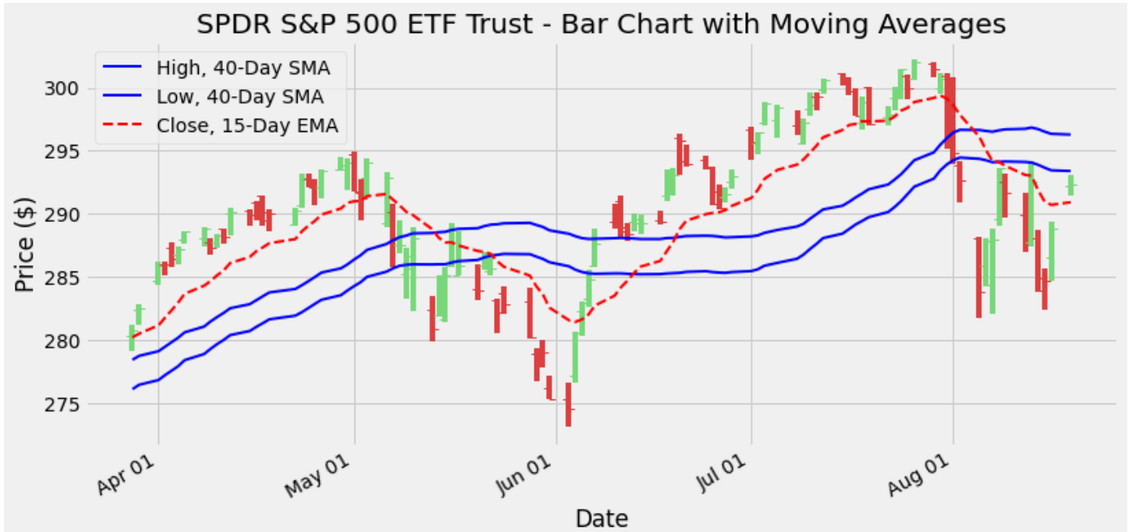
This can be coded as follows:

```
hsma40 = data['High'].rolling(40).mean()
lsma40 = data['Low'].rolling(40).mean()
ema15 = data['Close'].ewm(span=15).mean()

fig, ax = plt.subplots(figsize = (12,6))

mpf.plot_day_summary_ohlc(ax, ohlc[-100:], ticksize = 4, colorup='#77d879', colordown='#db3f3f')
ax.plot(hsma40[-100:], color = 'blue', linewidth = 2, label='High, 40-Day SMA')
ax.plot(lsma40[-100:], color = 'blue', linewidth = 2, label='Low, 40-Day SMA')
ax.plot(ema15[-100:], color = 'red', linestyle='--', linewidth = 2, label='Close, 15-Day EMA')

ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('SPDR S&P 500 ETF Trust - Bar Chart with Moving Averages')
ax.legend()
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d'))
fig.autofmt_xdate()
```


```
18
19  # plt.show() # add this if you're not using Jupyter Notebook
```

TTB03-camelback.py hosted with ❤ by GitHub view raw

Which gives:



The two simple moving averages drawn in blue create a **channel**: compared to a single moving average, we now have a *grey area* when prices are neither above nor below the channel.

1. Enter *long positions* (buying) only when the price bars are completely **above the higher 40-Day SMA**.
2. Enter *short positions* (selling) only when the price bars are completely **below the lower 40-Day SMA**.
3. We do not enter any position (we keep *flat* on the market) when the prices are **between the two 40-Day SMAs**, or the last bar is crossing any of them.

Another example could be to:

Stefano Basurto
371 Followers
Passionate about exploring new ways to apply data science techniques to make better investment decisions and develop insights in markets.

Follow

Related


Creating a Simple Rule-Based Chatbot with Python


Uploading python packages to be used by anyone with python


Getting Started With Pandas
Everything you will ever ...


Exploring Data of Indian Premier League with Python from Kaggle...

Help Status Writers Blog Careers
Privacy Terms About Knowable

1. Enter *long positions* only when the 15-Day EMA is above the higher 40-Day SMA.
2. Enter *short positions* only when the 15-Day EMA is below the lower 40-Day SMA.
3. Stay *flat* elsewhere, i.e. when the 15-Day EMA is inside the channel created by the two SMAs.

At this stage, we should be asking whether those rules (or any of the methods mentioned in the article on moving averages) can be used to build a profitable trading system. In other words, we should be asking whether those ideas will help us generate profits instead of making losses, and how to choose the best set of rules. I will try to address this in the next post. We will learn how to *backtest* a trading system to calculate the profits or losses based on historical data.