Create a Candlestick Chart Using Matplotlib in Python

Authored by

Mohammed looti

November 2, 2025

A candlestick chart is a fundamental tool in financial analysis, providing a visual representation of price action for securities, derivatives, or currencies over a specified period. These charts are essential for technical traders as they display four crucial data points: the open, close, high, and low prices.

Unlike simple line charts, candlesticks convey market sentiment and volatility instantly. They transform raw price data into intuitive shapes that reveal whether buyers or sellers were in control during a trading session. Mastering their creation is a key skill for any data scientist working with financial time-series data.

Introduction to Financial Candlestick Charts

The structure of a candlestick is designed to condense complex price movements into a single visual unit. Each candle represents the trading activity for one interval (e.g., a day, hour, or minute). The wide part of the candle is known as the "real body," which indicates the range between the opening and closing prices.

The thin lines extending above and below the real body are called "shadows" or "wicks." The upper shadow shows the highest price reached, while the lower shadow reveals the lowest price reached during that period. The color of the real body is what immediately signals the market direction.

In standard charting, a green (or sometimes white) body indicates a bullish session where the closing price was higher than the opening price. Conversely, a red (or sometimes black) body indicates a bearish session where the closing price was lower than the opening price. This visual differentiation is central to quick market interpretation.

Prerequisites: Preparing Data with Pandas and Python

To generate a reliable candlestick chart, we rely on Python's powerful data science ecosystem. Specifically, we utilize the Pandas library for efficient data handling and organization, which is optimized for managing time-series data in a DataFrame structure.

Before plotting, the data must be organized in the standard OHLC format (Open, High, Low, Close). The example below simulates a dataset containing the daily price movements of a particular stock over an eight-day period, indexing the data appropriately for chronological plotting.

The following code snippet demonstrates how to initialize the Pandas DataFrame, ensuring our price movements are correctly structured and indexed by date:

import pandas as pd

#create DataFrame

prices = pd.DataFrame({'open': ,
'close': ,
'high': ,
'low': },
index=pd.date_range("2021-01-01", periods=8, freq="d"))

#display DataFrame
print(prices)

open close high low
2021-01-01 25 24 28 22
2021-01-02 22 20 27 16
2021-01-03 21 17 29 14
2021-01-04 19 23 25 17
2021-01-05 23 22 24 19
2021-01-06 21 25 26 18
2021-01-07 25 29 31 22
2021-01-08 29 31 37 26

Implementing the Candlestick Logic Using Matplotlib

While specialized financial libraries exist (like `mplfinance`), generating a candlestick chart using core Matplotlib requires a clever workaround. Since Matplotlib does not include a native candlestick function in its standard toolkit, we must construct the candles manually by layering multiple bar charts.

The core of this technique involves calculating the height and position of two distinct elements for each day: the main body (Open to Close range) and the wicks (High/Low range). Crucially, we must first segment the data into "up" days (where Close >= Open) and "down" days (where Close < Open) to assign the appropriate colors.

The following comprehensive code block demonstrates this step-by-step construction. Notice the definition of two widths: `width` for the body and `width2` for the thin wicks, which ensures the visual integrity of the chart structure.

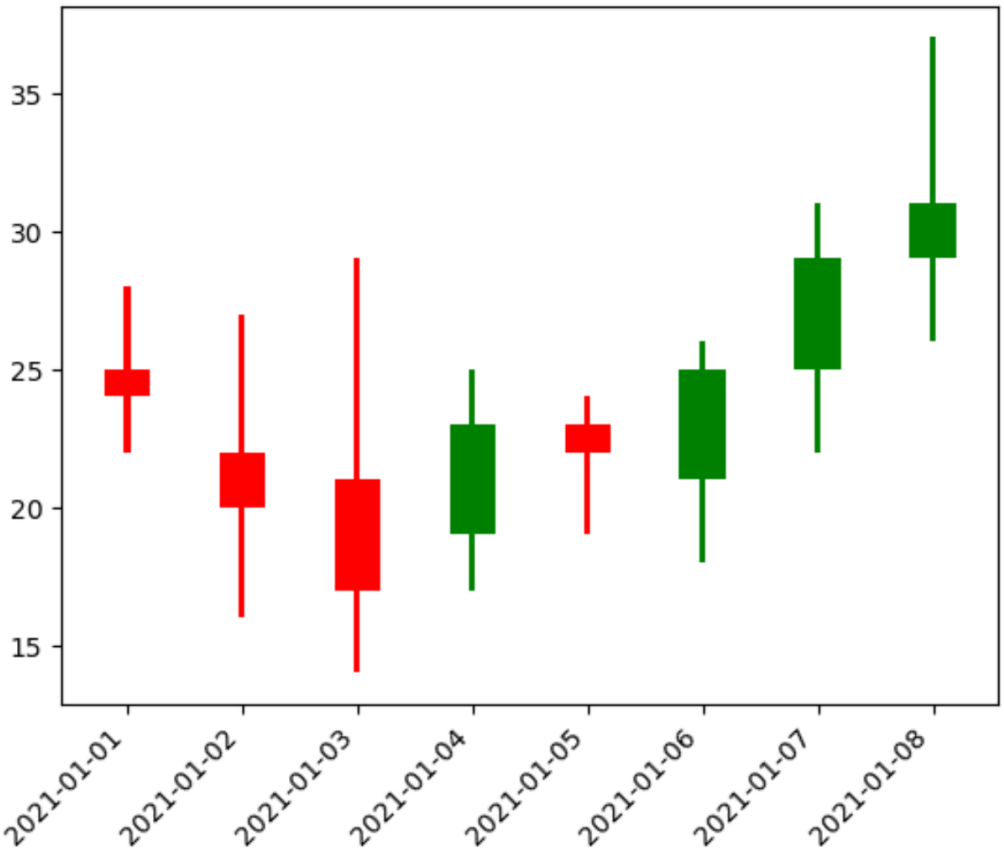
import matplotlib.pyplot as plt

#create figure
plt.figure()

#define width of candlestick elements

```
width = .4
width2 = .05

#define up and down prices
up = prices
down = prices

#define colors to use
col1 = 'green'
col2 = 'red'

#plot up prices
plt.bar(up.index,up.close-up.open,width,bottom=up.open,color=col1)
plt.bar(up.index,up.high-up.close,width2,bottom=up.close,color=col1)
plt.bar(up.index,up.low-up.open,width2,bottom=up.open,color=col1)

#plot down prices
plt.bar(down.index,down.close-down.open,width,bottom=down.open,color=col2)
plt.bar(down.index,down.high-down.open,width2,bottom=down.open,color=col2)
plt.bar(down.index,down.low-down.close,width2,bottom=down.close,color=col2)

#rotate x-axis tick labels
plt.xticks(rotation=45, ha='right')

#display candlestick chart
plt.show()
```

Executing the code above produces the initial candlestick visualization:



Interpreting the Generated Visualization

Each individual candlestick on the resulting chart encapsulates the full spectrum of price movement for its corresponding day. The visual signal provided by the color is immediate and crucial for interpreting market momentum.

A green candlestick signifies a day where the closing price was higher than the opening price, indicating a net gain and bullish sentiment for that session. Conversely, a red candlestick indicates that the closing price was lower than the opening price, suggesting bearish pressure and a net loss for the trading interval.

Furthermore, the shadows or wicks demonstrate the volatility and the range of extremes the price reached. Long wicks suggest significant price discovery occurred during the day, even if the eventual closing price was near the open. Short or absent wicks suggest that most of the trading occurred within the range of the main body.

Customizing the Chart Aesthetics

A significant advantage of building the chart manually with Matplotlib's `plt.bar()` function is the

high degree of control over the visual aesthetics. You are free to adjust parameters like the thickness of the candles and the color scheme to suit corporate branding or personal preference.

The variables `width` and `width2` directly control the visual proportions of the body and the wicks, respectively. By decreasing these values, we can make the chart appear less cluttered and more focused, which is often preferred when plotting a large volume of data points. Additionally, changing `col1` and `col2` allows for unique color mapping.

For instance, we can modify the parameters to use skinnier candles and employ a different color palette, swapping the standard green/red for a black/steelblue combination. This demonstrates the flexibility of the Matplotlib approach:

import matplotlib.pyplot as plt

#create figure
plt.figure()

#define width of candlestick elements
width = .2
width2 = .02

#define up and down prices
up = prices
down = prices

#define colors to use
col1 = 'black'
col2 = 'steelblue'

#plot up prices
plt.bar(up.index,up.close-up.open,width,bottom=up.open,color=col1)
plt.bar(up.index,up.high-up.close,width2,bottom=up.close,color=col1)
plt.bar(up.index,up.low-up.open,width2,bottom=up.open,color=col1)

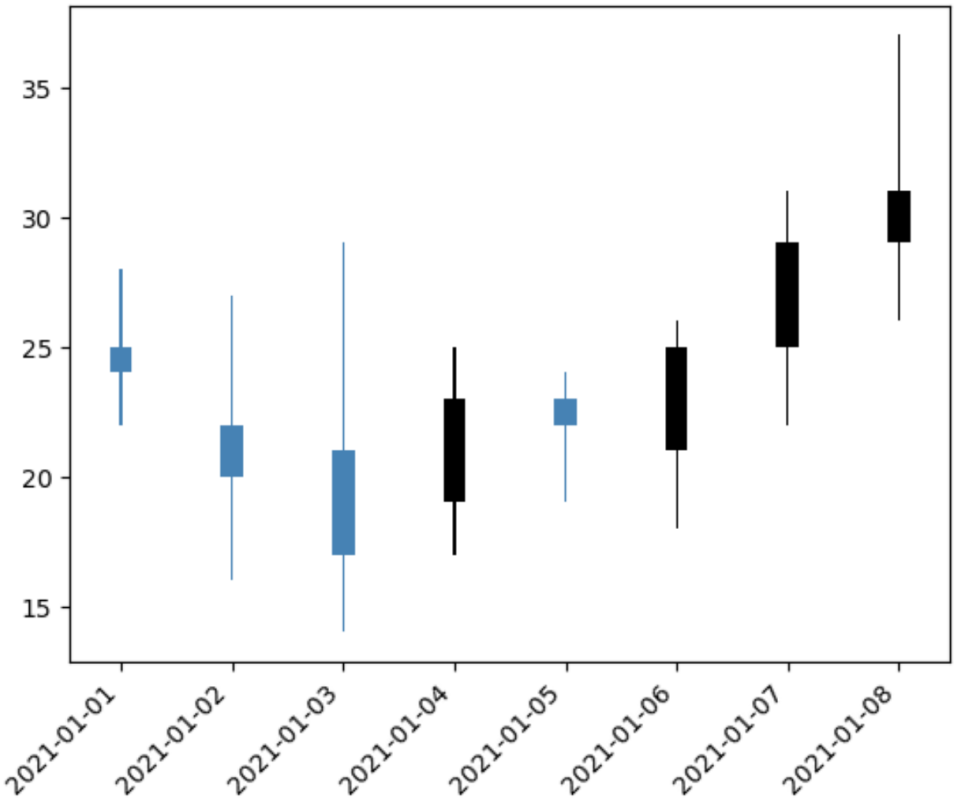
#plot down prices
plt.bar(down.index,down.close-down.open,width,bottom=down.open,color=col2)
plt.bar(down.index,down.high-down.open,width2,bottom=down.open,color=col2)
plt.bar(down.index,down.low-down.close,width2,bottom=down.close,color=col2)

#rotate x-axis tick labels
plt.xticks(rotation=45, ha='right')

#display candlestick chart

```
plt.show()
```

The resulting chart, using the new, thinner dimensions and customized colors, provides a cleaner visual display:



Conclusion and Next Steps

Creating functional and informative candlestick charts using Matplotlib and Pandas is a powerful technique for financial data visualization. While the implementation involves manually layering bar charts, this method grants maximum control over the final aesthetic and layout of the plot.

For high-frequency trading data or more complex technical analysis overlays, developers may consider dedicated libraries like `mplfinance` (a Matplotlib extension) which simplify the process considerably. However, understanding the underlying mechanism demonstrated here provides a strong foundation in data visualization principles.

Additional Resources

The following tutorials explain how to create other common charts in Python: