Introduction to Plotting with Matplotlib in Python

This tutorial demonstrates how to use Matplotlib, a powerful data visualization library in Python, to create line, bar, and scatter plots with stock market data.

Updated May 2023 · 25 min read

CONTENTS

SHARE

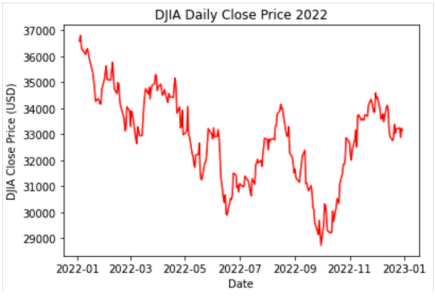
Matplotlib is a powerful and very popular data visualization library in Python. In this tutorial, we will discuss how to create line plots, bar plots, and scatter plots in Matplotlib using stock market data in 2022. These are the foundational plots that will allow you to start understanding, visualizing, and telling stories about data. Data visualization is an essential skill for all data analysts and Matplotlib is one of the most popular libraries for creating visualizations.

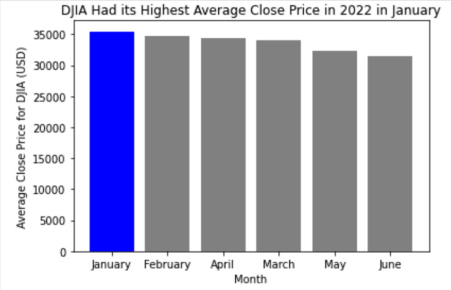
This tutorial expects some basic prior knowledge in NumPy arrays and pandas dataframes. When we use those libraries, we will quickly explain what we are doing. The main focus of this tutorial is Matplotlib, which works on top of these data structures to create visualizations.

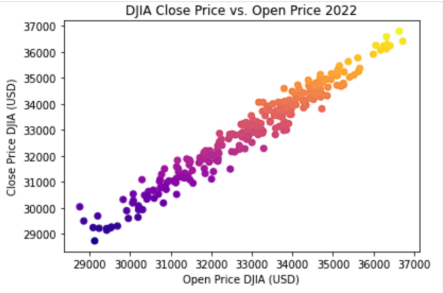
Matplotlib is very flexible and customizable for creating plots. It does require a lot of code to make more basic plots with little customizations. When working in a setting where exploratory data analysis is the main goal, requiring many quickly drawn plots without as much emphasis on aesthetics, the library seaborn is a great option as it builds on top of Matplotlib to create visualizations more quickly. Please see our Python Seaborn Tutorial For Beginners instead if exploratory data analysis or quick and easy graph creation is your main priority.

Matplotlib Examples

By the end of this tutorial, you will be able to make great-looking visualizations in Matplotlib. We will focus on creating line plots, bar plots, and scatter plots. We will also focus on how to make customization decisions, such as the use of color, how to label plots, and how to organize them in a clear way to tell a compelling story.







The Dataset

Matplotlib is designed to work with NumPy arrays and pandas dataframes. The library makes it easy to make graphs from tabular data. For this tutorial, we will use the Dow Jones Industrial Average (DJIA) index's historical prices from 2022-01-01 to 2022-12-31 (found here). You can set the date range on the page and then click the "download a spreadsheet" button.

We will load in the csv file, named `HistoricalPrices.csv` using the `pandas` library and view the first rows using the `.head()` method.

```
import pandas as pd

djia_data = pd.read_csv('HistoricalPrices.csv')
djia_data.head()
```

Explain code

Powered by OpenAI

	Date	Open	High	Low	Close
0	12/30/22	33121.61	33152.55	32847.82	33147.25
1	12/29/22	33021.43	33293.42	33020.35	33220.80
2	12/28/22	33264.76	33379.55	32869.15	32875.71
3	12/27/22	33224.23	33387.72	33069.58	33241.56
4	12/23/22	32961.06	33226.14	32814.02	33203.93

We see the data include 4 columns, a Date, Open, High, Low, and Close. The latter 4 are related to the price of the index during the trading day. Below is a brief explanation of each variable.

- **Date:** The day that the stock price information represents.
- **Open:** The price of the DJIA at 9:30 AM ET when the stock market opens.
- **High:** The highest price the DJIA reached during the day.
- **Low:** The lowest price the DJIA reached during the day.
- **Close:** The price of the DJIA when the market stopped trading at 4:00 PM ET.

As a quick clean up step, we will also need to use the `rename()` method in `pandas` as the dataset we downloaded has an extra space in the column names.

```
djia_data = djia_data.rename(columns = {' Open': 'Open', ' High': 'High', '
```

Explain code

Powered by OpenAI

We will also ensure that the Date variable is a datetime variable and sort in ascending order by the date.

```
djia_data['Date'] = pd.to_datetime(djia_data['Date'])
djia_data = djia_data.sort_values(by = 'Date')
```

Explain code Powered by OpenAI

Loading Matplotlib

Next, we will load the `pyplot` submodule of Matplotlib so that we can draw our plots. The `pyplot` module contains all of the relevant methods we will need to create plots and style them. We will use the conventional alias `plt`. We will also load in pandas, numpy, and datetime for future parts of this tutorial.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import datetime
```

Explain code Powered by OpenAI

Drawing Line Plots

The first plot we will create will be a line plot. Line plots are a very important plot type as they do a great job of displaying time series data. It is often important to visualize how KPIs change over time to understand patterns in data that can be actioned on.

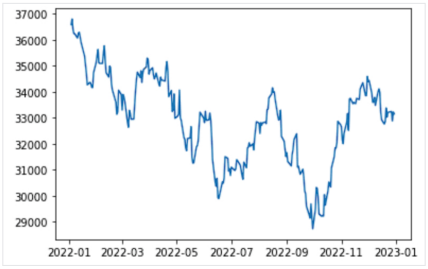
Line Plots with a Single Line

- Show how to draw a simple line plot with a single line.
  - Make sure to emphasize the use of plt.show() so the plot actually displays.
- Provide brief commentary on the plot, including interpretation.

We can create a line plot in matplotlib using the `plt.plot()` method where the first argument is the x variable and the second argument is the y variable in our line plot. Whenever we create a plot, we need to make sure to call `plt.show()` to ensure we see the graph we have created. We will visualize the close price over time of the DJIA.

```
plt.plot(djia_data['Date'], djia_data['Close'])
plt.show()
```

Explain code Powered by OpenAI



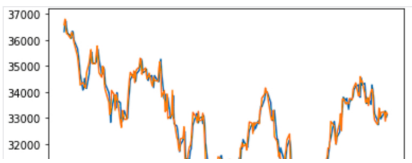
We can see that over the course of the year, the index price started at its highest value followed by some fluctuations up and down throughout the year. We see the price was lowest around October followed by a strong end of the year increase in price.

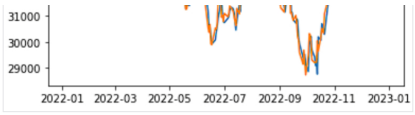
Line Plots with Multiple Lines

We can visualize multiple lines on the same plot by adding another `plt.plot()` call before the `plt.show()` function.

```
plt.plot(djia_data['Date'], djia_data['Open'])
plt.plot(djia_data['Date'], djia_data['Close'])


plt.show()
```

Explain code Powered by OpenAI





Over the course of the year, we see that the open and close prices of the DJIA were relatively close to each other for each given day with no clear pattern of one always being above or below the other.

Adding a Legend

If we want to distinguish which line represents which column, we can add a legend. This will create a color coded label in the corner of the graph. We can do this using `plt.legend()` and adding label parameters to each `plt.plot()` call.

```
plt.plot(djia_data['Date'], djia_data['Open'], label = 'Open')
plt.plot(djia_data['Date'], djia_data['Close'], label = 'Close')
plt.legend()
plt.show()
```

Explain code

Powered by OpenAI

We now see a legend with the specified labels appear in the default location in the top right (location can be specified using the `loc` argument in `plt.legend()`).

Drawing Bar Plots

Bar plots are very useful for comparing numerical values across categories. They are particularly helpful for finding the largest and smallest categories.

For this section we will aggregate the data into monthly averages using pandas so that we can compare monthly performance during 2022 for the DJIA. We will also use the first 6 months to make the data easier to visualize.

```
# Import the calendar package
from calendar import month_name

# Order by months by chronological order
djia_data['Month'] = pd.Categorical(djia_data['Date'].dt.month_name(), month_name

# Group metrics by monthly averages
djia_monthly_mean = djia_data \
    .groupby('Month') \
    .mean() \
    .reset_index()

djia_monthly_mean.head(6)
```

Explain code

Powered by OpenAI

	Month	Open	High	Low	Close
0	January	35498.1305	35740.9105	35145.955	35456.145
1	February	34687.5163157895	34906.2042105263	34362.2436842105	34648.4805263158
2	March	34007.4982608696	34270.8908695652	33752.9634782609	34029.7404347826
3	April	34392.0945	34640.3675	34078.481	34314.99
4	May	32364.3271428571	32668.0280952381	31996.5128571429	32379.4628571429
5	June	31526.0333333333	31755.6371428571	31188.7880952381	31446.7128571429

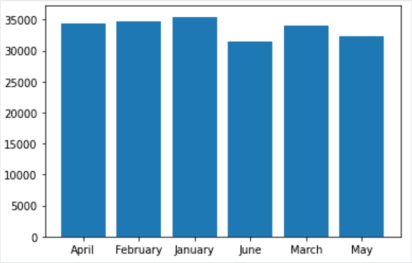
Vertical Bar Plots

We will start by creating a bar chart with vertical bars. This can be done using the `plt.bar()` method with the first argument being the x-axis variable (Month) and the `height` parameter being the y-axis (Close). We then want to make sure to call `plt.show()` to show our plot.

```
plt.bar(djia_monthly_mean['Month'], height = djia_monthly_mean['Close'])
plt.show()
```

Explain code

Powered by OpenAI



We see that most of the close prices of the DJIA were close to each other with the lowest average close value being in June and the highest average close value being in January.

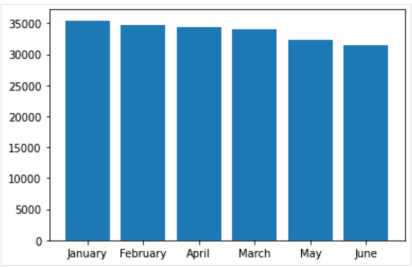
Reordering Bars in Bar Plots

If we want to show these bars in order of highest to lowest Monthly average close price, we can sort the bars using the `sort_values()` method in pandas and then using the same plt.bar() method.

```
djia_monthly_mean_srtd = djia_monthly_mean.sort_values(by = 'Close', ascend =

plt.bar(djia_monthly_mean_srtd['Month'], height = djia_monthly_mean_srtd['Close']
plt.show()
```

Explain code

Powered by OpenAI



As you can see, it is significantly easier to see which months had the highest average DJIA close price and which months had the lower averages. It is also easier to compare across months and rank the months.

Horizontal Bar Plots

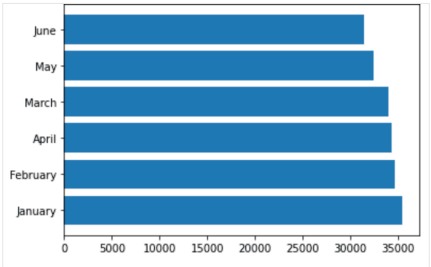
- Show how to swap the axes, so the bars are horizontal.
- Provide brief commentary on the plot, including interpretation.

It is sometimes easier to interpret bar charts and read the labels when we make the bar plot with horizontal bars. We can do this using the `plt.hbar()` method.

```
plt.barh(djia_monthly_mean_srtd['Month'], height = djia_monthly_mean_srtd['C e'
plt.show()
```

Explain code

Powered by OpenAI



As you can see, the labels of each category (month) are easier to read than when the bars

were vertical. We can still easily compare across groups. This horizontal bar chart is especially useful when there are a lot of categories.

Drawing Scatter Plots

Scatterplots are very useful for identifying relationships between 2 numeric variables. This can give you a sense of what to expect in a variable when the other variable changes and can also be very informative in your decision to use different modeling techniques such as linear or non-linear regression.

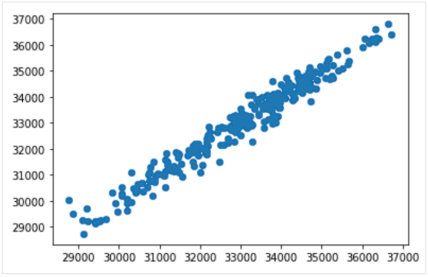
Scatter Plots

Similar to the other plots, a scatter plot can be created using `pyplot.scatter()` where the first argument is the x-axis variable and the second argument is the y-axis variable. In this example, we will look at the relationship between the open and close price of the DJIA.

```
plt.scatter(djia_data['Open'], djia_data['Close'])
plt.show()
```

Explain code

Powered by OpenAI



On the x-axis we have the open price of the DJIA and on the y-axis we have the close price. As we would expect, as the open price increases, we see a strong relationship in the close price increasing as well.

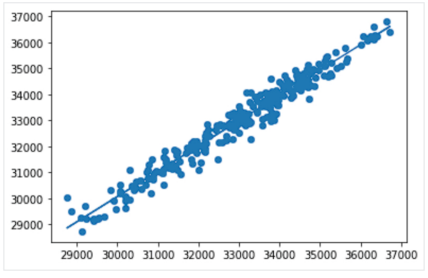
Scatter Plots with a Trend Line

Next, we will add a trend line to the graph to show the linear relationship between the open and close variables more explicitly. To do this, we will use the numpy `polyfit()` method and `poly1d()`. The first method will give us a least squares polynomial fit where the first argument is the x variable, the second variable is the y variable, and the third variable is the degrees of the fit (1 for linear). The second method will give us a one-dimensional polynomial class that we can use to create a trend line using plt.plot().

```
z = np.polyfit(djia_data['Open'], djia_data['Close'], 1)
p = np.poly1d(z)


plt.scatter(djia_data['Open'], djia_data['Close'])
plt.plot(djia_data['Open'], p(djia_data['Open']))
plt.show()
```

Explain code

Powered by OpenAI



As we can see, the line in the background of the graph follows the trend of the scatterplot closely as the relationship between open and close price is strongly linear. We see that as the open price increases, the close price generally increases at a similar and linear rate.

Setting the Plot Title and Axis Labels

Plot titles and axis labels make it significantly easier to understand a visualization and allow the viewer to quickly understand what they are looking at more clearly. We can do this by adding more layers using `plt.xtitle()`, `plt.ylabel()` and `plt.xlabel()` which we will demonstrate with the scatterplot we made in the previous section.

demonstrate with the scatterplot we made in the previous section.

```
plt.scatter(djia_data['Open'], djia_data['Close'])
plt.show()
```

Explain code

Powered by OpenAI

Changing Colors

Color can be a powerful tool in data visualizations for emphasizing certain points or telling a consistent story with consistent colors for a certain idea. In Matplotlib, we can change colors using named colors (e.g. "red", "blue", etc.), hex code ("#f4db9a", "#383c4a", etc.), and red-green-blue tuples (e.g. (125, 100, 37), (30, 54, 121), etc.).

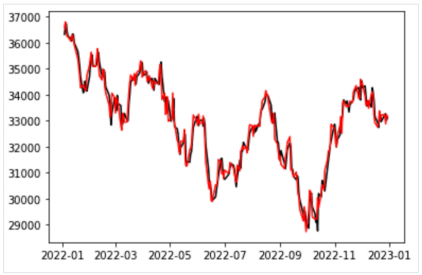
Lines

For a line plot, we can change a color using the color attribute in plt.plot(). Below, we change the color of our open price line to “black” and our close price line to “red.”

```
plt.plot(djia_data['Date'], djia_data['Open'], color = 'black')
plt.plot(djia_data['Date'], djia_data['Close'], color = 'red')
plt.show()
```

Explain code

Powered by OpenAI



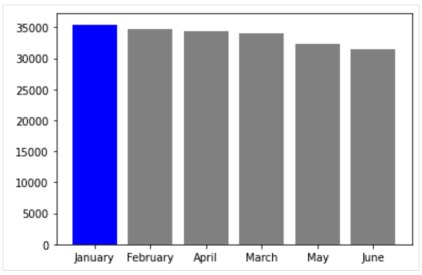
Bars

For bars, we can pass a list into the color attribute to specify the color of each line. Let’s say we want to highlight the average price in January for a point we are trying to make about how strong the average close price was. We can do this by giving that bar a unique color to draw attention to it.

```
plt.bar(djia_monthly_mean_srtd['Month'], height = djia_monthly_mean_srtd['C']
plt.show()
```

Explain code

Powered by OpenAI



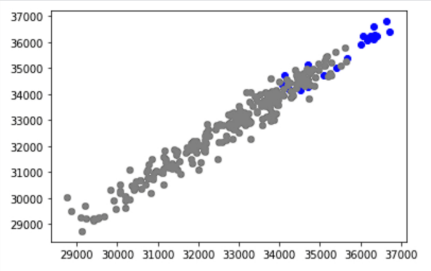
Points

Finally, for scatter plots, we can change the color using the color attribute of `plt.scatter()`. We will color all points in January as blue and all other points as gray to show a similar story

as in the above visualization.

```
plt.scatter(djia_data[djia_data['Month'] == 'January']['Open'], djia_data[d da

plt.scatter(djia_data[djia_data['Month'] != 'January']['Open'], djia_data[djia_da

plt.show()
```

Explain code Powered by OpenAI



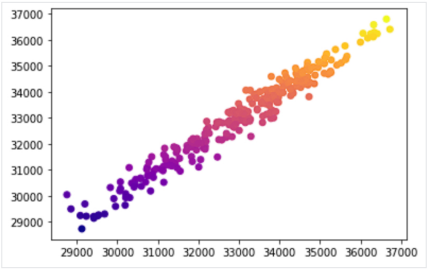
Using Colormaps

Colormaps are built-in Matplotlib colors that scale based on the magnitude of the value (documentation here). The colormaps generally aesthetically look good together and help tell a story in the increasing values.

We see in the below example, we use a colormap by passing the close price (y-variable) to the `c` attribute, and the plasma colormap through `cmap`. We see that as the values increase, the associated color gets brighter and more yellow while the lower end of the values is purple and darker.

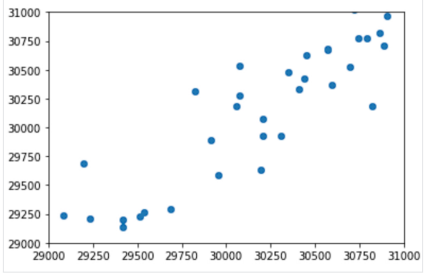
```
plt.scatter(djia_data['Open'], djia_data['Close'], c=djia_data['Close'], cm p

plt.show()
```

Explain code Powered by OpenAI



Setting Axis Limits

Sometimes, it is helpful to look at a specific range of values in a plot. For example, if the DJIA is currently trading around $30,000, we may only care about behavior around that price. We can pass a tuple into the `plt.xlim()` and `plt.ylim()` to set x and y limits respectively. The first value in the tuple is the lower limit, and the second value in the tuple is the upper limit.



Saving Plots

Finally, we can save plots that we create in matplotlib using the `plt.savefig()` method. We can save the file in many different file formats including 'png,' 'pdf,' and 'svg'. The first argument is the filename. The format is inferred from the file extension (or you can override this with the `format` argument).

```
plt.scatter(djia_data['Open'], djia_data['Close'])
plt.savefig('DJIA 2022 Scatterplot Open vs. Close.png')
```

Explain code

Powered by OpenAI

Take it to the Next Level

We have covered the basics of Matplotlib in this tutorial and you can now make basic line graphs, bar graphs, and scatter plots. Matplotlib is an advanced library with a lot of great features for creating aesthetically pleasing visualizations. If you would like to take your Matplotlib skills to the next level, take our **Introduction to Data Visualization with Matplotlib** course. You can also download our **Matplotlib Cheat Sheet: Plotting in Python** for reference as you start creating your own visualizations.

Matplotlib FAQs

What is Matplotlib in Python?

Matplotlib is a popular data visualization library in Python. It's often used for creating static, interactive, and animated visualizations in Python. Matplotlib allows you to generate plots, histograms, bar charts, scatter plots, etc., with just a few lines of code.

Why should I use Matplotlib for data visualization?

How do I install Matplotlib?

How do I create a basic plot in Matplotlib?

AUTHOR

Kevin Babitz

Data Science Blogger | Technical Analyst at WayFair | MSE in Data Science at UPenn

TOPICS

Python Data Visualization

Kevin Babitz

Data Science Blogger | Technical Analyst at WayFair | MSE in Data Science at UPenn

TOPICS

Python

Data Visualization

RELATED

10 Essential Python Skills All Data Scientists Should Master

The 7 Best Python Certifications For All Levels

Textacy: An Introduction to Text Data Cleaning and Normalization in Python

Types of Data Plots and How to Create Them in Python

Learn more about Python

CERTIFICATION AVAILABLE

Introduction to Data Visualization with Matplotlib

Beginner 4 hr 158.1K

Learn how to create, customize, and share data visualizations using Matplotlib.

See Details → Start Course

Python for MATLAB Users

Beginner 4 hr 6.6K

Transition from MATLAB by learning some fundamental Python concepts, and diving into the NumPy and Matplotlib packages.

See Details → Start Course

Introduction to NumPy

Beginner 4 hr 29.9K

Master your skills in NumPy by learning how to create, sort, filter, and update arrays using NYC's tree census.

See Details → Start Course

See More →

Related

10 Essential Python Skills All Data Scientists Should Master

All data scientists need expertise in Python, but which skills are the most important for them to master? Find out th...

Thaylise Nakamoto • 9 min

The 7 Best Python Certifications For All Levels

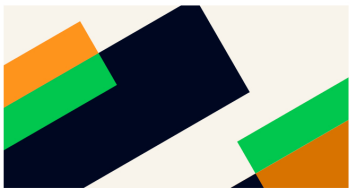
Find out whether a Python certification is right for you, what the best options are, and the alternatives on offer in this...

Matt Crabtree • 18 min

Textacy: An Introduction to Text Data Cleaning and...

Discover how Textacy, a Python library, simplifies text data preprocessing for machine learning. Learn about its unique...

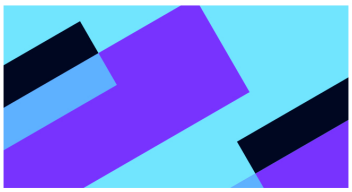
Mustafa El-Dalil • 5 min



Types of Data Plots and How to Create Them in Python

Explore various types of data plots—from the most common to advanced and unconventional ones—what they show,...

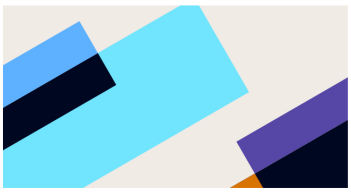
Elena Kosourova • 21 min



Coding Best Practices and Guidelines for Better Code

Learn coding best practices to improve your programming skills. Explore coding guidelines for collaboration, code structur...

Amberle McKee • 26 min



Pandas Profiling (ydata-profiling) in Python: A Guide fo...

Learn how to use the ydata-profiling library in Python to generate detailed reports for datasets with many features.

Satyam Tripathi • 9 min

See More →