Learning to Plot Data: A Guide to Visualizing Two Columns from a Pandas DataFrame

Authored by

Mohammed looti

June 7, 2026

Introduction to Data Visualization with Pandas

Effective Data Visualization is crucial for interpreting complex datasets. When working with tabular data in Python, the Pandas DataFrame structure serves as the industry standard for storage and manipulation. A frequent requirement in data analysis is plotting the relationship or trend between two specific columns within this structure. Fortunately, Pandas integrates seamlessly with powerful plotting libraries, offering two primary, straightforward methods for visualizing column data: using a **Scatter Plot** for correlation analysis or employing a **Line Chart** to illustrate sequential trends. Understanding these two foundational techniques is essential for any data scientist leveraging the Python ecosystem.

The choice between these methods depends entirely on the nature of the data and the underlying question being asked. If you are examining how two independent variables relate to each other--such as player points versus assists--a **Scatter Plot** is the ideal choice. Conversely, if your data is sequential (e.g., performance over time or across games), the ability of a **Line Chart** to connect data points in order provides a clearer representation of change and progression. We will explore both of these common approaches in detail, providing practical code examples using real-world basketball statistics.

Essential Libraries: Pandas and Matplotlib

Before diving into the plotting mechanics, it is vital to acknowledge the two core libraries enabling this functionality. First, Pandas provides the high-performance, easy-to-use data structures, most notably the **DataFrame**. Second, Matplotlib is the foundational plotting library in Python. While Pandas offers a convenient `.plot()` wrapper method, this method internally relies on Matplotlib's functionality. Therefore, many advanced customization options require direct interaction with the underlying Matplotlib API, specifically the `pyplot` module.

The two basic methodologies for plotting two columns from a Pandas DataFrame are executed as follows. Note the fundamental difference: Method 1 uses the Matplotlib interface directly, while Method 2 leverages the built-in Pandas plotting convenience function.

Method 1: Visualizing Relationships with Scatter Plots

A Scatter Plot is primarily used to observe and display the relationship between two numerical variables. Each row in the **DataFrame** is represented as a single point, plotted according to its values in the specified X and Y columns. This visualization is excellent for identifying correlations, clusters, or outliers within the data. To implement this, we typically import the `pyplot` module from Matplotlib and call the `scatter()` function, passing the desired columns as arguments for the X and Y axes.

The syntax is straightforward, referencing the **DataFrame** (`df`) and selecting the columns using bracket notation (e.g., `df`). This approach gives the user maximum control over the plotting environment, allowing for detailed customization of markers, colors, and plot annotations using standard Matplotlib functions.

import matplotlib.pyplot as plt

plt.scatter(df, df)

This initial code snippet immediately generates the basic plot, but in real-world applications, always remember to add informative axis labels and a title using Matplotlib functions like `plt.xlabel()`, `plt.ylabel()`, and `plt.title()` for proper interpretability.

Detailed Example: Creating a Scatter Plot

To demonstrate the practical application of a Scatter Plot, let us consider a sample Pandas DataFrame containing basketball player statistics. This dataset includes the team affiliation, total points scored, and the number of assists recorded by several players. We aim to visualize the relationship between points scored (X-axis) and assists (Y-axis) to see if high scorers also tend to accumulate a high number of assists.

First, the Pandas DataFrame must be constructed. This step ensures that our data is properly structured before visualization can begin. Note the clean, tabular representation of the data, which is the cornerstone of effective data analysis in Python.

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ,
'points': ,
'assists': })

#view DataFrame
print(df)

team points assists
0 A 18 5
1 B 22 7
2 C 19 7
3 D 14 9
4 E 14 12

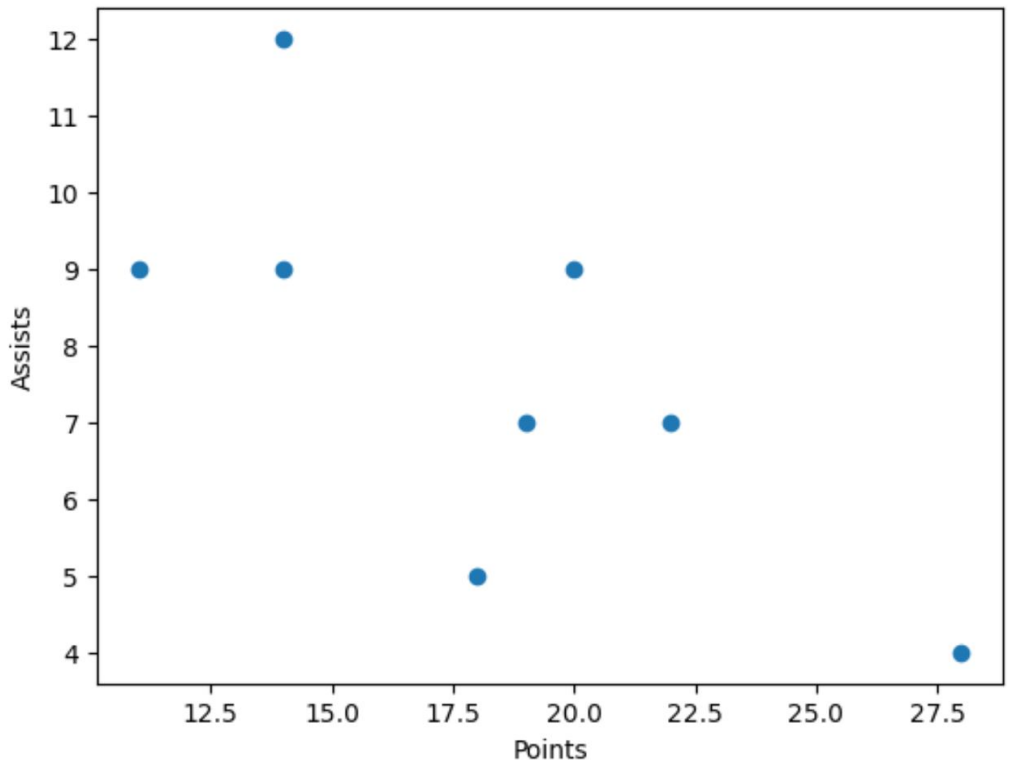
5 F 11 9
6 G 20 9
7 H 28 4

Once the data is prepared, we execute the plotting command. We explicitly map the **points** column to the independent X-axis and the **assists** column to the dependent Y-axis. This visualization is essential for quickly identifying potential linear or non-linear relationships between these two performance metrics.

import matplotlib.pyplot as plt

#create scatter plot
plt.scatter(df, df)

#add axis labels
plt.xlabel('Points')
plt.ylabel('Assists')



As visible in the output, the X-axis is now scaled according to the values in the **points** column, and the Y-axis reflects the values from the **assists** column. Analyzing the resulting Scatter Plot allows us to visually determine the strength and direction of the correlation between these two variables

across the observed players.

Method 2: Tracking Trends with Line Charts

When the X-axis represents a sequence, such as time, game number, or ordered indices, a Line Chart is the most appropriate visualization tool. Unlike scatter plots, line charts connect consecutive data points, emphasizing continuity and trend over time. This method is particularly useful for time series analysis or tracking performance metrics across ordered experimental stages.

The most efficient way to generate a Line Chart directly from a **DataFrame** is by utilizing the built-in Pandas `.plot()` method. This method acts as a wrapper around Matplotlib, defaulting to a line plot unless specified otherwise. Critically, the `.plot()` method allows for plotting multiple columns simultaneously against a single designated X-axis column, making comparison easy. We specify the X-axis column and provide a list of Y-axis columns we wish to track.

df.plot(x='column1', y=)

This approach is highly idiomatic in the Pandas DataFrame environment, streamlining the process of plotting complex comparisons. When multiple Y columns are provided, Pandas automatically generates a legend to distinguish the lines, a feature often requiring more explicit code when using pure Matplotlib calls.

Detailed Example: Generating a Multi-Column Line Chart

For our second example, we will track the performance of a basketball team across six sequential games. We are interested in visualizing two metrics: **points_for** (scored by the team) and **points_against** (scored by the opponent). The game number (**game** column) serves as the sequential X-axis. This comparison helps us quickly identify games where the team performed strongly (points for > points against) versus games where they struggled.

We begin by setting up the sequential dataset in a Pandas DataFrame. Notice that the index order (0 through 5) corresponds naturally to the game sequence, although we explicitly use the designated 'game' column for the X-axis definition, which is best practice.

import pandas as pd

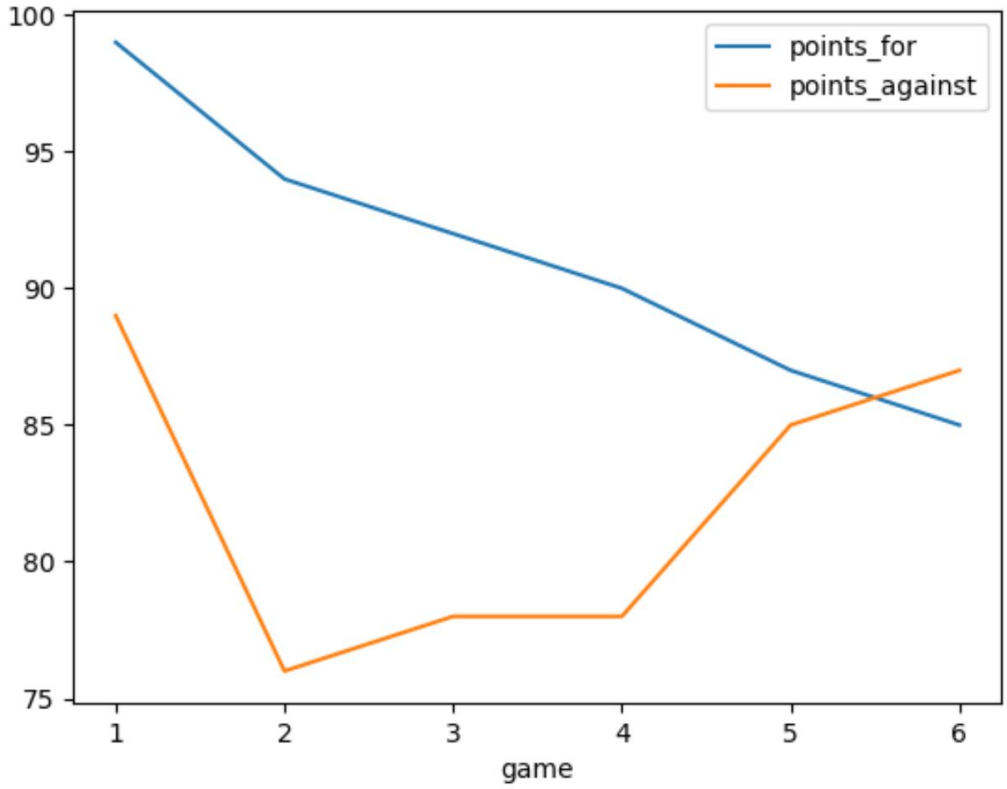
#create DataFrame
df = pd.DataFrame({'game': ,
'points_for': ,
'points_against': })

#view DataFrame
print(df)

game points_for points_against
0 1 99 89
1 2 94 76
2 3 92 78
3 4 90 78
4 5 87 85
5 6 85 87

To visualize the performance trend, we invoke the `.plot()` method, specifying 'game' as the X-axis and passing a list containing the two quantitative columns to the Y parameter. This command efficiently generates two distinct lines on the same plot, allowing for direct comparison of the scores across the sequence of games.

#plot points_for and points_against columns on same y-axis
df.plot(x='game', y=)



The resulting Line Chart clearly shows the performance trajectory. The blue line represents the

values for the **points_for** column in each game, indicating a downward trend in scoring. Simultaneously, the orange line illustrates the **points_against** column, showing that opposing teams' scores remained relatively stable or slightly increased towards the end of the observed sequence. This visual comparison immediately highlights the diminishing margin of victory (or increasing margin of loss) over the six games.

Summary and Further Resources

Mastering the art of plotting two columns from a Pandas DataFrame is a foundational skill in data science. By choosing between a **Scatter Plot** (for correlation) and a **Line Chart** (for trends), and leveraging the power of Matplotlib, analysts can quickly transform raw data into meaningful visual insights. Remember that the key distinction lies in whether the relationship is independent (scatter) or sequential/ordered (line).

The techniques demonstrated here--using `plt.scatter()` for fine-grained control and `df.plot()` for streamlined sequential visualization--cover the vast majority of two-variable plotting tasks encountered in data analysis workflows. These methods ensure that your Data Visualization efforts are both accurate and compelling, leading to better decision-making based on robust analysis.

Additional Resources

The following tutorials explain how to perform other common tasks in pandas and data visualization:

The official Pandas visualization documentation offers deep dives into various plot types supported by the DataFrame structure.

Reviewing comprehensive guides on Data Visualization best practices helps ensure plots are readable and informative.

Understanding advanced Matplotlib customization techniques is essential for creating publication-quality graphics.