How to Easily Create Pie Charts from Pandas DataFrames

Authored by
stats writer

December 3, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Create Pie Charts from Pandas DataFrames*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=104406

ARABPSYCHOLOGY.COM

The Power of Visualization: Generating Pie Charts from Pandas

Data visualization is a critical step in any analytical workflow, transforming raw data into easily digestible insights. When dealing with categorical data and seeking to represent the proportional distribution of different categories, the pie chart remains one of the most effective tools. Fortunately, integrating data analysis with visualization is seamless when using the Pandas DataFrame in conjunction with its built-in plotting functionalities, which utilize the powerful Matplotlib library behind the scenes.

The process of generating a pie chart from a structured Pandas DataFrame is exceptionally straightforward. It primarily involves ensuring your data is aggregated correctly and then invoking the dedicated plotting function. This article provides a comprehensive guide, walking through the necessary steps from initial data preparation to advanced customization, allowing you to create clear, informative, and visually appealing visualizations that accurately reflect your data distributions.

Before plotting, it is vital to remember that a pie chart visualizes how different segments contribute to a whole. Therefore, the data must be grouped and aggregated (usually summed or counted) based on the categories you wish to display. The core of this technique relies on the sequential application of the .groupby() method, followed by an aggregation function like `.sum()`, culminating in the `.plot(kind='pie')` command.

Essential Syntax for Pie Chart Generation

To successfully generate a pie chart, the data must be prepared so that one column represents the categories (the index after grouping) and the other column represents the values or magnitudes associated with those categories. The standard structure leverages the efficiency of method chaining within Pandas.

The primary requirement is to aggregate the dataset by the column that defines the categorical groups. Once aggregated, the resulting Series or DataFrame contains the necessary totals for plotting. The `.plot()` method, when passed the argument `kind='pie'`, handles the conversion of these aggregate values into proportional slices.

You can use the following basic syntax to create a pie chart from a Pandas DataFrame, ensuring you replace the placeholder names with your actual column identifiers:

df.groupby().sum().plot(kind='pie', y='value_column')

In this syntax, `group_column` is the categorical feature used for segmentation, and `value_column` is the numerical feature whose aggregated values determine the size of the slices. The beauty of

ARABPSYCHOLOGY.COM

this approach lies in its conciseness and readability, allowing developers to create complex visualizations with minimal code. The following examples show how to use this syntax in practice, beginning with a straightforward implementation.

Example 1: Setting Up and Plotting a Basic Pie Chart

For our first example, we will simulate a scenario where we track points scored across different teams. This scenario is ideal for a pie chart because we want to see the proportional contribution of each team's score to the total points accumulated across all teams. We must first initialize our Pandas DataFrame, which serves as the foundation for our analysis and visualization.

Suppose we have the following sample data structure. Notice that team names are repeated, indicating multiple data entries per category, which necessitates the use of the .groupby() method before plotting. We rely on the `pandas` library, which is typically imported using the standard alias `pd`.

Here is the necessary code to create and display the initial DataFrame:

import pandas as pd

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

#view DataFrame
print(df)

team points
0 A 25
1 A 12
2 B 25
3 B 14
4 B 19
5 B 53
6 C 25
7 C 29
```

As shown in the output, the DataFrame contains eight records detailing individual point contributions. To plot this as a proportional chart, we must consolidate the total points for Team A, Team B, and Team C separately. This aggregation ensures that the slices accurately reflect the accumulated scores rather than individual entries.

ARABPSYCHOLOGY.COM

Implementing the Aggregation and Plotting Command

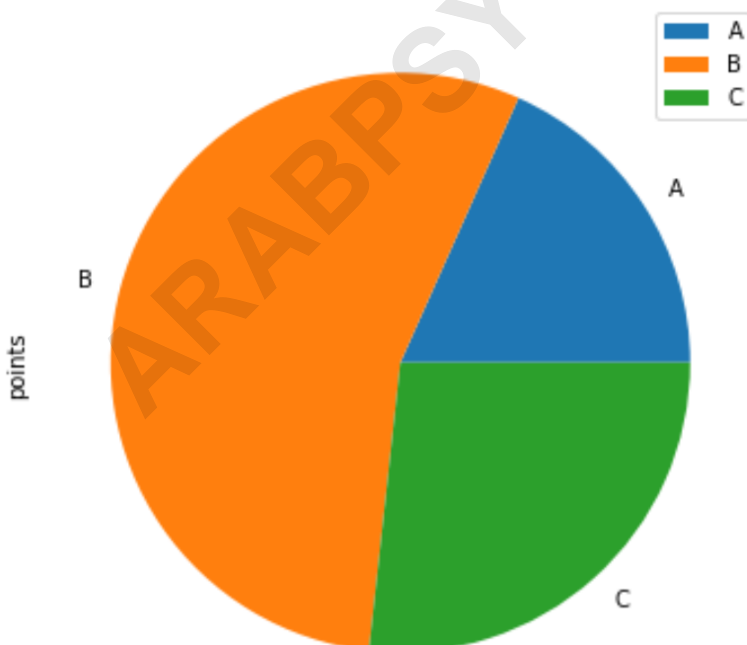
With the data prepared, the next step is to apply the method chain to generate the visualization. We utilize the .groupby() method on the `'team'` column to group all related scores, and then use `.sum()` to calculate the total points for each team. The resulting aggregated data structure is then passed directly to the `.plot()` function.

The key parameter within the `.plot()` call is `y='points'`. This explicitly tells Pandas which numerical column in the aggregated result should be used to determine the size of the pie slices. The index of the aggregated data (which is now the 'team' column) automatically provides the labels for each slice.

We can use the following concise syntax to create a pie chart that displays the portion of total points scored by each team:

df.groupby().sum().plot(kind='pie', y='points')

Executing this command generates the visual representation of the data distribution. The chart immediately provides a clear picture of which team contributed the largest percentage of points. This basic visualization is essential for quick data exploration and reporting, but often requires further refinement for professional use.



ARABPSYCHOLOGY.COM

Advanced Customization Techniques

While the basic chart is functional, customization is crucial for improving clarity, readability, and aesthetic appeal. Pandas plotting functions, inheriting from Matplotlib, offer extensive parameters to refine the visualization. These parameters allow us to add titles, specify custom color schemes, and, most importantly for pie charts, display the actual percentage value on each slice.

The following key arguments provide the most common and powerful ways to customize the appearance and information density of your pie chart:

autopct: This argument stands for automatic percentage formatting. It allows you to display the calculated percentage value directly on the slice, significantly enhancing the chart's informational value.
colors: This allows the user to specify a list of colors (using names, hex codes, or RGB tuples) that Matplotlib should cycle through when coloring the slices. This is essential for aligning the visualization with brand standards or improving contrast.
title: A descriptive title is mandatory for any formal visualization, providing immediate context to the viewer regarding the data being represented.

By implementing these arguments, we move beyond a raw visualization toward a publication-ready figure. Understanding the order of slices is also paramount; they are typically plotted based on the order of the categories in the underlying aggregated DataFrame index.

Example 2: Implementing Custom Styles and Percentages

We will now demonstrate how to apply these customization arguments to the existing team points data. The goal is to make the chart more informative by displaying percentages (using `autopct`) and more visually distinct by assigning specific colors. Furthermore, we will include a descriptive `title` to clearly define the plot's content.

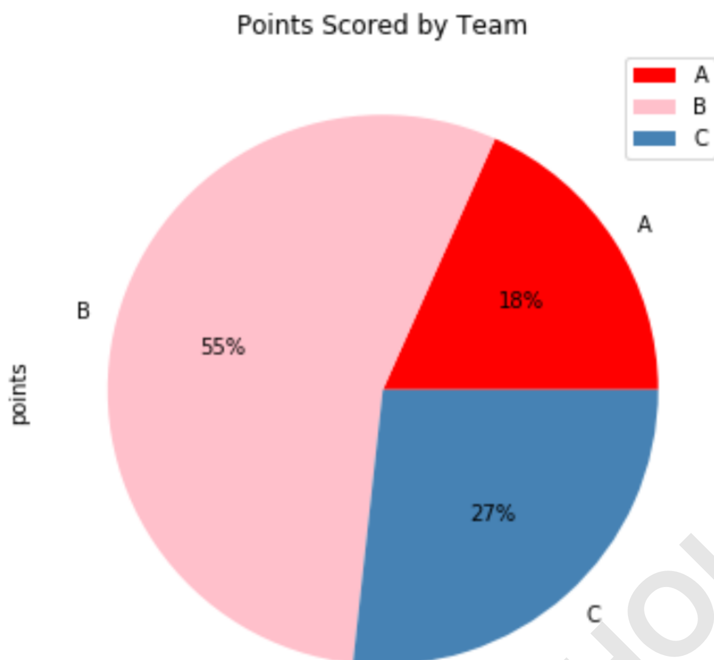
The `autopct` argument requires a format string, similar to those used in C-style programming or Python's `%` formatting. The format `'%1.0f%%'`, for instance, means display the percentage as a floating-point number rounded to zero decimal places, followed by the literal percentage symbol (`%%`). Custom colors are provided as a list, matching the number of categories being plotted.

The following code shows how to use these arguments in practice, chaining them directly within the `.plot()` call for efficiency:

df.groupby().sum().plot(kind='pie', y='points', autopct='%1.0f%%',
colors = ,
title='Points Scored by Team'))

ARABPSYCHOLOGY.COM

Execution of this code produces a much richer visualization, incorporating both textual information (percentages) and visual enhancements (color differentiation). This level of detail ensures that the audience can quickly glean the necessary insights without needing to refer back to the raw data table.



ARABPSYCHOLOGY.COM

Controlling Slice Ordering and Color Mapping

A key consideration when assigning custom colors is the relationship between the color list and the categorical labels. Matplotlib, through Pandas, assigns colors based on the internal order of the categories as they appear in the grouped result. If the underlying Pandas DataFrame has categories sorted alphabetically, the colors will be assigned alphabetically as well.

It is important to understand that the colors will be assigned to the categories as they appear in the aggregated DataFrame index. For example, if the teams are ordered A, B, C (which is the default alphabetical sort unless explicitly changed):

Team 'A' appears first in the index, which is why it received the color 'red' in the pie chart based on the provided list .
Team 'B' is second and receives 'pink'.
Team 'C' is third and receives 'steelblue'.

If you require a specific order for the slices (e.g., ordering by size or a custom hierarchy), you must sort the DataFrame **before** calling the `.plot()` method. This ensures predictable mapping between the custom color list and the visualization segments. Alternatively, for very complex

custom coloring, mapping dictionaries might be used with Matplotlib directly, although the Pandas built-in functionality handles most standard needs.

Saving and Exporting the Visualization

After generating and customizing the pie chart, the final step in the workflow is often saving the graphic for use in reports, presentations, or websites. Since the Pandas `.plot()` method relies on Matplotlib, you gain access to Matplotlib's powerful saving functionality. This usually requires importing Matplotlib's pyplot module (conventionally as `plt`) if you haven't already.

Once the plot object is created, you can save it using the standard .savefig() method. This method allows you to specify the filename, file format (such as PNG, JPEG, SVG, or PDF), and parameters like resolution (DPI) for high-quality output.

A typical sequence to save the output involves storing the plot result in an object (often an `Axes` object), and then calling `plt.savefig()`. For example, if the plot command were executed, the saving command would follow, typically requiring an explicit Matplotlib import:

```
import matplotlib.pyplot as plt
# plot generation code executed here, potentially assigned to ax
plt.savefig('points_pie_chart.png', dpi=300)
plt.close() # Clean up memory resources
```

Using a high DPI (dots per inch) setting, such as 300 or 600, ensures that the exported image maintains sharpness and clarity, which is crucial for professional documentation. The ability to export in vector formats like SVG or PDF is particularly beneficial for preserving scalability without loss of quality.

Conclusion and Further Explorations

Creating a compelling pie chart from a Pandas DataFrame is a streamlined process that leverages powerful aggregation and plotting capabilities. By chaining the .groupby() method and the `.plot(kind='pie')` function, developers can quickly generate accurate proportional visualizations.

Mastering the customization arguments, particularly `autopct` and `colors`, allows for the creation of charts that are not only accurate but also highly informative and visually appealing. Remember that the success of a pie chart depends on correctly aggregating the underlying data to represent parts of a meaningful whole.

For those interested in expanding their visualization toolkit, Pandas and Matplotlib offer functions

ARABPSYCHOLOGY.COM

for generating a wide variety of other common plots necessary for data analysis. These tutorials explain how to create other common plots using a pandas DataFrame:

ARABPSYCHOLOGY.COM