Python Visualization Libraries: Matplotlib & Pandas

Documentation Guide

Abstract

A comprehensive guide to data visualization using Python's Matplotlib and Pandas.

Souvik Dutta

Python Visualization Guide: Matplotlib & Pandas

1. Library Overview

- **Matplotlib**: A comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a foundational API that gives users granular control over every aspect of a plot. Matplotlib is ideal for creating publication-quality figures, detailed scientific plots, and custom visualizations where precise control is paramount.
- **Pandas**: A powerful data analysis and manipulation library that includes a high-level plotting interface built on top of Matplotlib. Its primary use case is for quick data exploration and creating visualizations directly from a DataFrame or Series with minimal code. It's a convenient tool for data scientists and analysts to quickly generate insights from their data.

2. Graph Types

Matplotlib

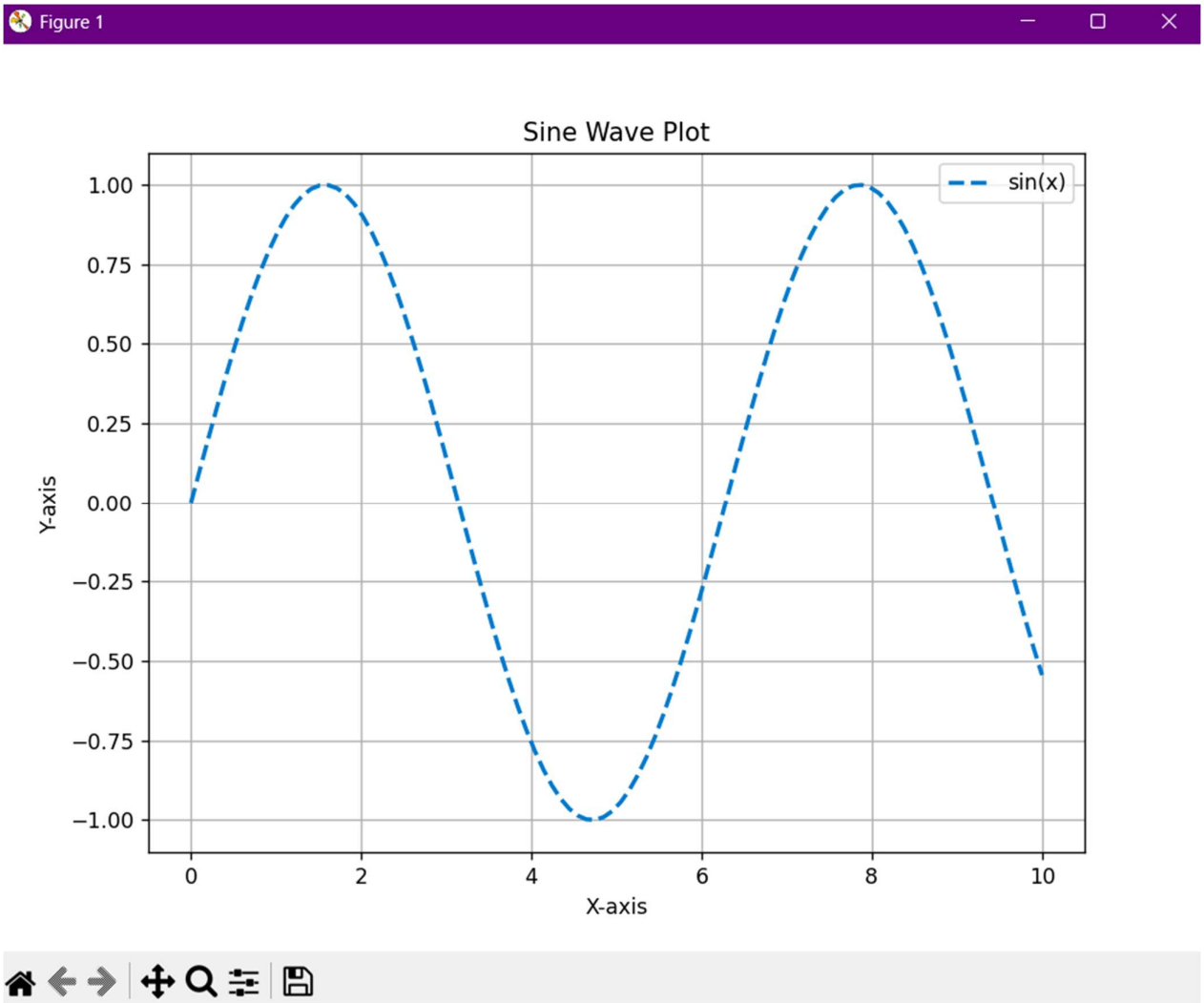
Matplotlib offers a state-based interface for creating a wide variety of plot types.

- plt.plot() - Line Plot
  - Description: Creates a 2D line plot, perfect for visualizing trends over a continuous range, such as time.
  - Use Case: Tracking stock prices, showing temperature changes over a year, or plotting mathematical functions.
  - Syntax and Key Parameters:
    - x: The horizontal coordinates.
    - y: The vertical coordinates.
    - format_string (optional): Shorthand for color, marker, and line style (e.g., 'r--o').
    - linewidth: The thickness of the line.
    - label: A string for the legend.

- **Example**

```
import matplotlib.pyplot as plt
import numpy as np

x_values = np.linspace(0, 10, 100)
y_values = np.sin(x_values)

plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, color='#007acc', linestyle='--', linewidth=2,
label='sin(x)')
plt.title('Sine Wave Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.show()
```

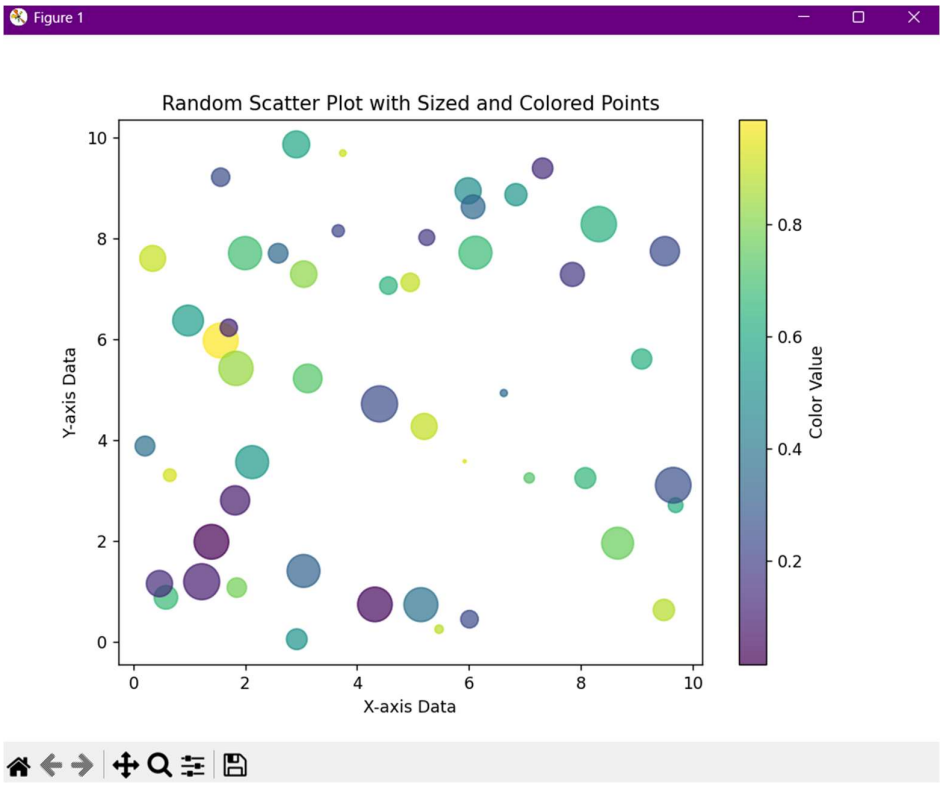
Output :



- **plt.scatter()** - Scatter Plot
  - **Description**: An ideal visualization for examining the relationship between two continuous variables, revealing clusters, correlations, or outliers.
  - **Use Case**: Plotting height vs. weight, visualizing customer demographics, or showing the relationship between advertising spend and sales.
  - **Syntax and Key Parameters**:
    - x: The data for the x-axis.
    - y: The data for the y-axis.
    - s (optional): The size of the markers.
    - c (optional): The color of the markers.
  - Example:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(42)
x = np.random.rand(50) * 10
y = np.random.rand(50) * 10
sizes = np.random.rand(50) * 500
colors = np.random.rand(50)

plt.figure(figsize=(8, 6))
plt.scatter(x, y, s=sizes, c=colors, cmap='viridis', alpha=0.7)
plt.title('Random Scatter Plot with Sized and Colored Points')
plt.xlabel('X-axis Data')
plt.ylabel('Y-axis Data')
plt.colorbar(label='Color Value')
plt.show()
```

Output :

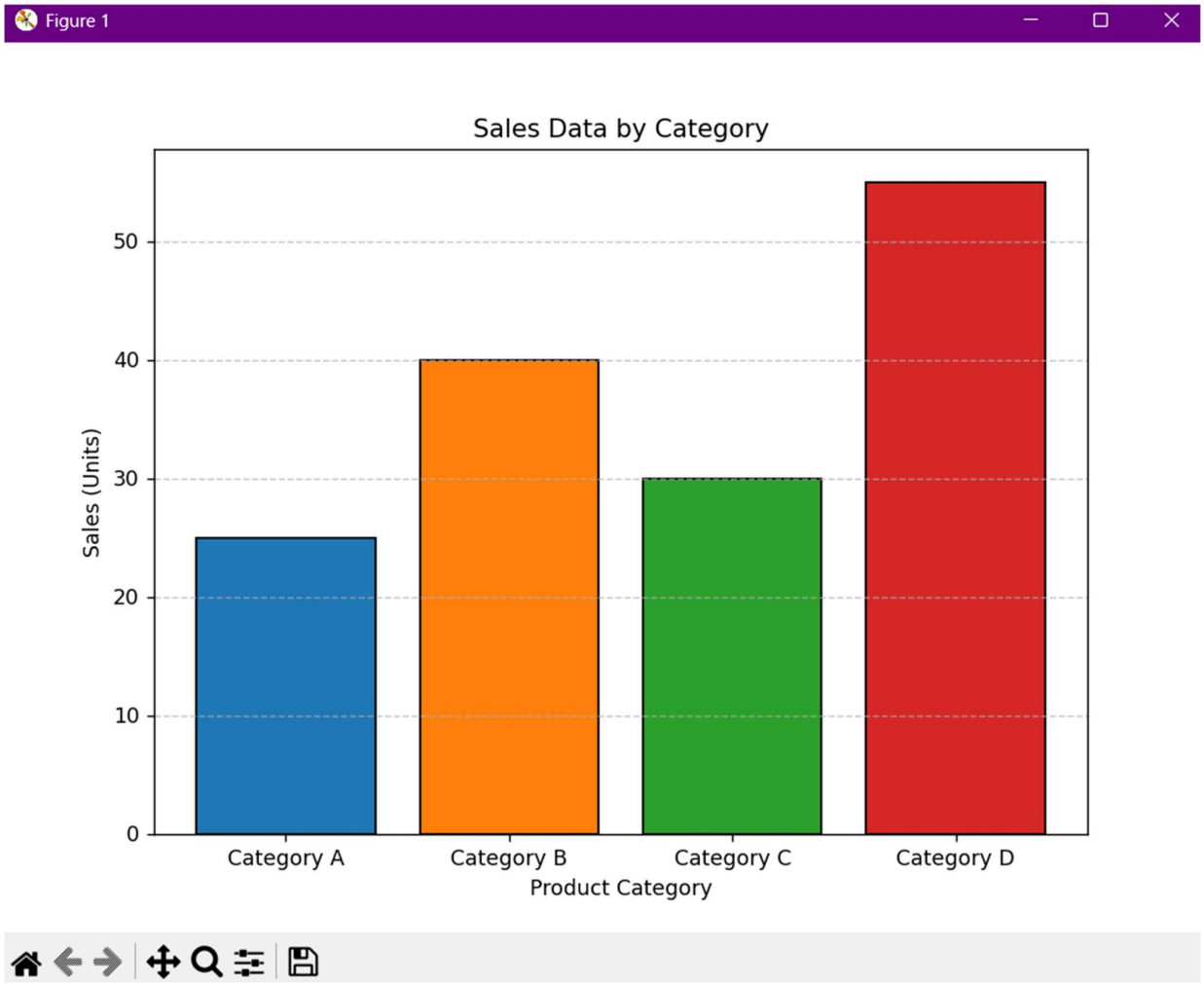


- plt.bar() - Bar Chart
  - **Description**: Generates a vertical bar chart, highly effective for comparing discrete, categorical data.
  - **Use Case**: Comparing sales by product category, visualizing survey results for different demographics, or showing website traffic by source.
  - **Syntax and Key Parameters**:
    - x: The x-coordinates of the bars.
    - **height**: The height of each bar.
    - **width**: The width of the bars.

- **Example**:

```
import matplotlib.pyplot as plt

categories = ['Category A', 'Category B', 'Category C', 'Category D']
values = [25, 40, 30, 55]

plt.figure(figsize=(8, 6))
plt.bar(categories, values, color=['#1f77b4', '#ff7f0e', '#2ca02c',
'#d62728'], edgecolor='black')
plt.title('Sales Data by Category')
plt.xlabel('Product Category')
plt.ylabel('Sales (Units)')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

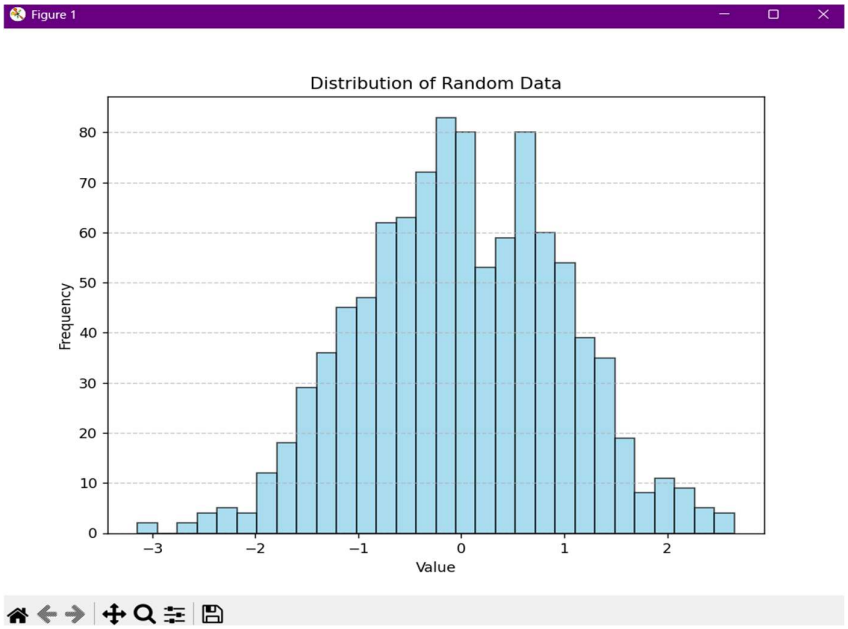
Output :



- plt.hist() - Histogram
  - **Description**: Visualizes the distribution of a single continuous variable by dividing the data into bins and counting their frequency.
  - **Use Case**: Analyzing the distribution of customer ages, checking for skewness in test scores, or understanding the frequency of product prices.
  - Syntax and Key Parameters:
    - x: The input data.
    - bins: The number of bins to use.
    - density: If True, the output is a probability density function.
  - Example:

```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.randn(1000)

plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Distribution of Random Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

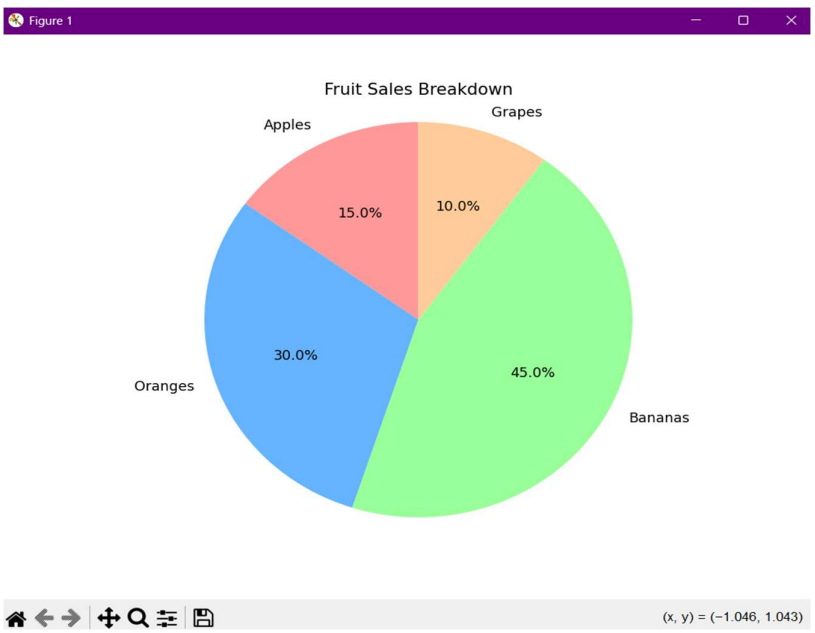
Output :



- plt.pie() - Pie Chart
  - **Description**: Creates a pie chart, useful for displaying proportional data, where each slice represents a category's proportion of the whole.
  - **Use Case**: Showing market share percentages, visualizing budget allocations, or displaying survey responses by category.
  - **Syntax and Key Parameters:**
    - x: The wedge sizes.
    - labels: A list of strings for each wedge.
    - autopct: A format string to display the percentage value.
  - **Example:**

```
import matplotlib.pyplot as plt

sizes = [15, 30, 45, 10]
labels = ['Apples', 'Oranges', 'Bananas', 'Grapes']
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=90)
plt.title('Fruit Sales Breakdown')
plt.axis('equal')
plt.show()
```

Output :



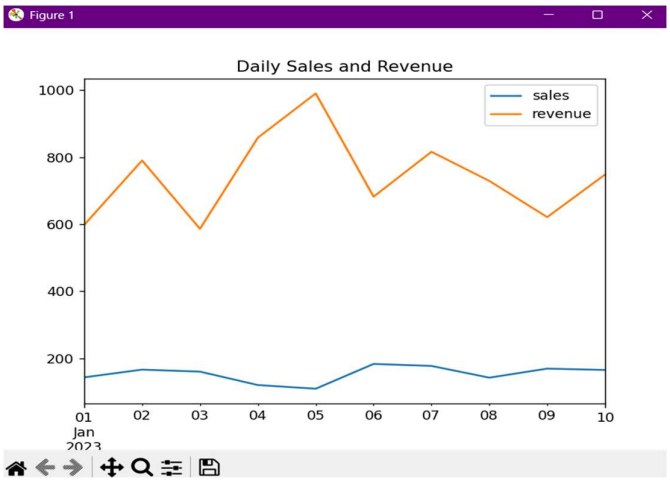
Pandas

Pandas provides a high-level plotting interface through the .plot() method.

- DataFrame.plot(kind='line') - Line Plot
  - Description: Plots the index of the DataFrame on the x-axis and each numeric column on the y-axis.
  - Use Case: Ideal for visualizing time-series data or trends, such as daily sales.
  - Syntax and Key Parameters:
    - x: The column for the x-axis.
    - y: The columns to plot on the y-axis.
  - Example:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

df = pd.DataFrame(
    {'sales': np.random.randint(100, 200, 10),
     'revenue': np.random.randint(500, 1000, 10)},
    index=pd.date_range('2023-01-01', periods=10)
)

df.plot(kind='line', title='Daily Sales and Revenue')

plt.show()
```

Output :



The DataFrame.plot(kind='bar') function in pandas is used to create bar charts directly from a DataFrame, making it a convenient tool for visualizing categorical data. This type of chart is especially useful when comparing values across different categories—for example, showing units sold by product or revenue by region. The x parameter specifies which column should be used for the labels on the x-axis, while the y parameter determines which column(s) will be plotted as the height of the bars. Additionally, setting stacked=True allows you to create a stacked bar chart, where multiple data series are layered on top of each other within the same bar, making it easier to compare cumulative totals. This method simplifies the process of turning tabular data into clear, interpretable visual insights.

- **Example:**

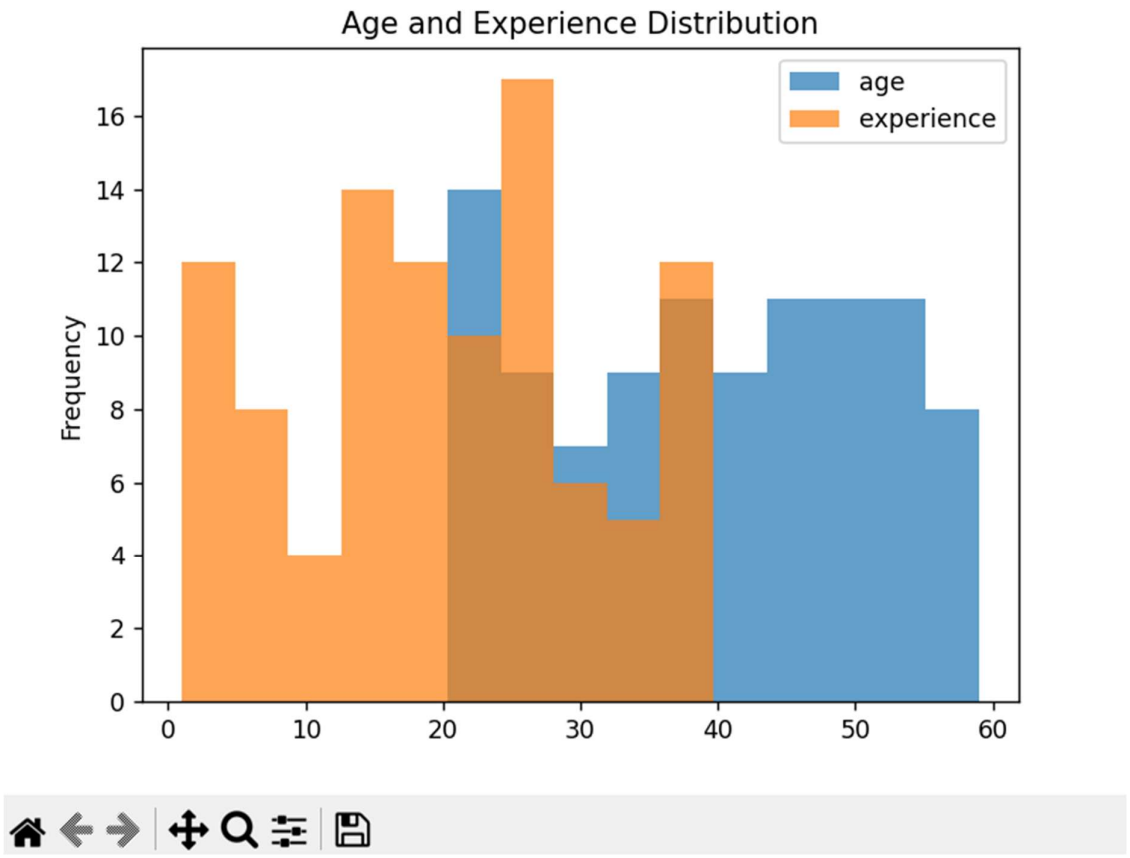
```
import pandas as pd

data = {'product': ['A', 'B', 'C', 'D'],
        'units_sold': [150, 200, 120, 180]}
df = pd.DataFrame(data)

df.plot(kind='bar', x='product', y='units_sold', title='Units Sold by
Product')
```

- Output :

- DataFrame.plot(kind='hist') - Histogram
  - Description: Creates a histogram for each numerical column in the DataFrame.
  - Use Case: A quick way to understand the distribution of a variable like age or experience.
  - Syntax and Key Parameters:
    - bins: The number of histogram bins.
  - Example:

```
import pandas as pd
import numpy as np

data = {'age': np.random.randint(20, 60, 100),
        'experience': np.random.randint(1, 40, 100)}
df = pd.DataFrame(data)

df[['age', 'experience']].plot(kind='hist', alpha=0.7, bins=15,
title='Age and Experience Distribution')
```

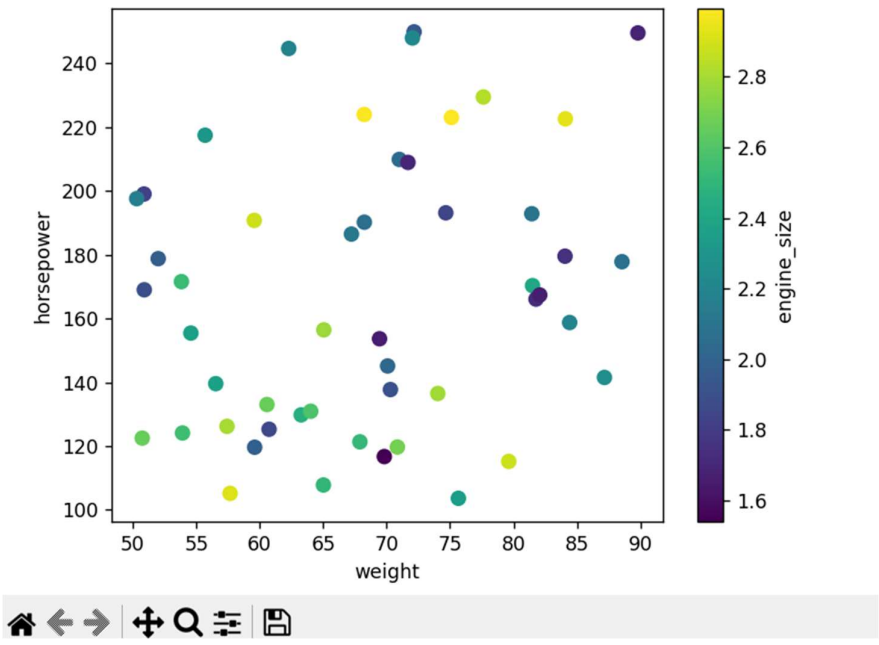
Output :



- DataFrame.plot(kind='scatter') - Scatter Plot
  - Description: Creates a scatter plot using two specified columns.
  - Use Case: Exploring potential relationships between variables, such as a car's weight and horsepower.
  - Syntax and Key Parameters:
    - x: The column for the x-axis.
    - y: The column for the y-axis.
    - s: The size of the markers, can be a column name.
    - c: The color of the markers, can be a column name.
  - Example:

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
    'weight': np.random.uniform(50, 90, 50),
    'horsepower': np.random.uniform(100, 250, 50),
    'engine_size': np.random.uniform(1.5, 3.0, 50)
})

df.plot(kind='scatter', x='weight', y='horsepower', c='engine_size',
colormap='viridis', s=50)
```

Output :



3. Comparison Graph

Feature	Matplotlib	Pandas
Ease of Use	Lower. Requires more boilerplate code and understanding of the API.	Higher. Simple, one-line commands for common plots directly from a DataFrame.
Customization	High. Provides fine-grained control over every element of the plot, including figure size, subplots, axes, and text.	Lower. Relies on Matplotlib's capabilities and may require a separate call to Matplotlib functions for advanced customization.
Interactivity	Limited native interactivity. Requires additional libraries (e.g., mpl_toolkits.mplot3d) for 3D plots.	No native interactivity. All interactive features would require a separate library like Plotly.
Large Datasets	Good performance. It is a mature library with efficient rendering capabilities for a wide range of data sizes.	Good performance. Its speed for plotting depends on the efficiency of its Matplotlib backend. As a data manipulation tool, Pandas is highly optimized for large datasets.
Primary Goal	Creating highly customized, publication-quality visualizations.	Quick, exploratory data analysis and visualization.