
MATPLOTLIB AND PANDAS

Overview of Matplotlib:

→Matplotlib is a widely-used Python library for creating static, interactive, and animated visualizations. It provides a comprehensive environment for creating a wide variety of plots and graphs.

→Matplotlib is highly customizable and is often used in conjunction with other libraries like NumPy and Pandas for handling and visualizing data. It serves as the foundation for many higher-level plotting libraries, such as Seaborn and Plotly.

→Matplotlib is a powerful plotting library in Python used for creating static, animated, and interactive visualizations.

→Matplotlib's primary purpose is to provide users with the tools and functionality to represent data graphically, making it easier to analyze and understand.

→It was originally developed by John D. Hunter in 2003 and is now maintained by a large community of developers.

→Matplotlib is a comprehensive library for creating static, animated, and interactive plots in Python.

→It provides flexibility for visualizing data in multiple formats such as line charts, bar plots, histograms, pie charts, and scatter plots.

→Primary Module: pyplot is the most used module in Matplotlib for easy plotting, functioning similarly to MATLAB.

Key Features of Matplotlib

- **Highly Customizable:** Control every element of a plot (titles, axes, legends, ticks, etc.).
- **Supports Multiple Plot Types:** Includes line plots, scatter plots, bar charts, histograms, pie charts, and more.
- **Interactive and Animated Plots:** Supports toolkits like mpl_toolkits for 3D plots and interactive backends for notebooks.
- **Integration:** Works seamlessly with other libraries such as Pandas and NumPy.

Why and When should choose Matplotlib for Data Visualization?

Matplotlib is popular due to its ease of use, extensive documentation, and wide range of plotting capabilities. It offers flexibility in customization, supports various plot types, and integrates well with other Python libraries like NumPy and Pandas.

Matplotlib is a suitable choice for various data visualization tasks, including exploratory data analysis, scientific plotting, and creating publication-quality plots. It excels in scenarios where users require fine-grained control over plot customization and need to create complex or specialized visualizations.

Advantages of Matplotlib

Matplotlib is a widely used plotting library in Python that provides a variety of plotting tools and capabilities:

1. **Versatility:** Matplotlib can create a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more.
2. **Customization:** It offers extensive customization options to control every aspect of the plot, such as line styles, colors, markers, labels, and annotations.
3. **Integration with NumPy:** Matplotlib integrates seamlessly with NumPy, making it easy to plot data arrays directly.
4. **Publication Quality:** Matplotlib produces high-quality plots suitable for publication with fine-grained control over the plot aesthetics.

Disadvantages of Matplotlib

While Matplotlib is a powerful and versatile plotting library, it also has some disadvantages that users might encounter:

1. **Steep Learning Curve:** For beginners, Matplotlib can have a steep learning curve due to its extensive customization options and sometimes complex syntax.
2. **Verbose Syntax:** Matplotlib's syntax can be verbose and less intuitive compared to other plotting libraries like Seaborn or Plotly, making it more time-consuming to create and customize plots.
3. **Default Aesthetics:** The default plot aesthetics in Matplotlib are often considered less visually appealing compared to other libraries, requiring more effort to make plots visually attractive.

4. **Limited Interactivity:** While Matplotlib does support interactive plotting to some extent, it does not offer as many interactive features and options as other libraries like Plotly.

Library Overview:

- 1.**Core Module:** pyplot, which offers functions to create different types of plots.
- 2.**Customization:** Supports customization of labels, axes, titles, and legend positioning.
3. **Integration:** Works well with Jupyter notebooks and other interactive environments.
- 4.**Performance:** Suitable for creating detailed, publication-quality plots.

Common Use Cases:

- 1.Data exploration
2. Publishing static visualizations
- 3.Customizing plots for more granular control

Different Types of Graph in Matplotlib:

Matplotlib offers a wide range of plot types to suit various data visualization needs. Here are some of the most commonly used types of plots in Matplotlib:

- Line Graph
- Stem Plot
- Bar chart
- Histograms
- Scatter Plot
- Stack Plot
- Box Plot
- Pie Chart
- Error Plot
- Violin Plot
- 3D Plots

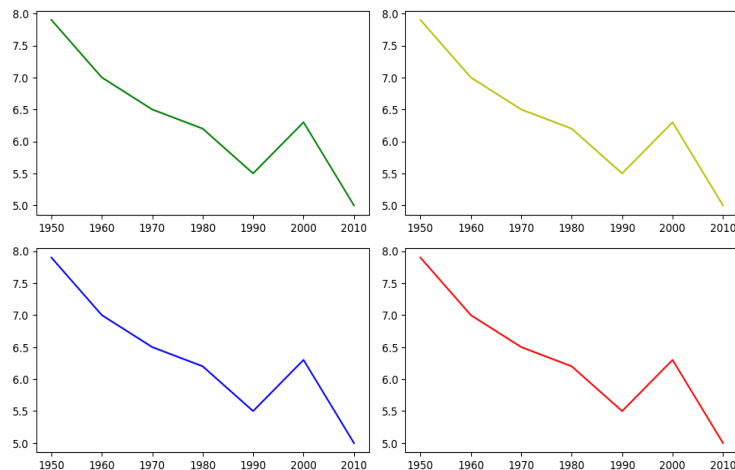
1. Line Plot

- **Use Case:** Showing trends or changes over time.
- **How it works:** Plots data points connected by straight lines.

Example:

```
import matplotlib.pyplot as plt  
x = [1, 2, 3, 4] y = [10, 20, 15,  
30] plt.plot(x, y, marker='o')  
plt.title("Line Plot") plt.show()
```

Output:



2. Scatter Plot

- **Use Case:** Displaying relationships between two variables.
- **How it works:** Plots individual data points without connecting them.

Example:

```
X=np.array([5,7,8,7,2,17,2,9,4,11,1  
2,9,6])
```

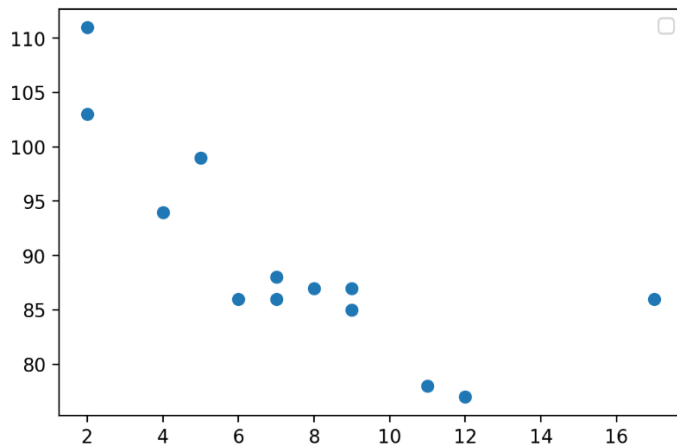
```
Y=np.array([99,86,87,88,111,86,10  
3,87,94,78,77,85,86])
```

```
plt.scatter(x, y)
```

```
plt.legend()
```

```
plt.show()
```

Output:



3. Bar Plot

- **Use Case:** Comparing discrete categories.
- **How it works:** Uses rectangular bars to represent data values.

Example:

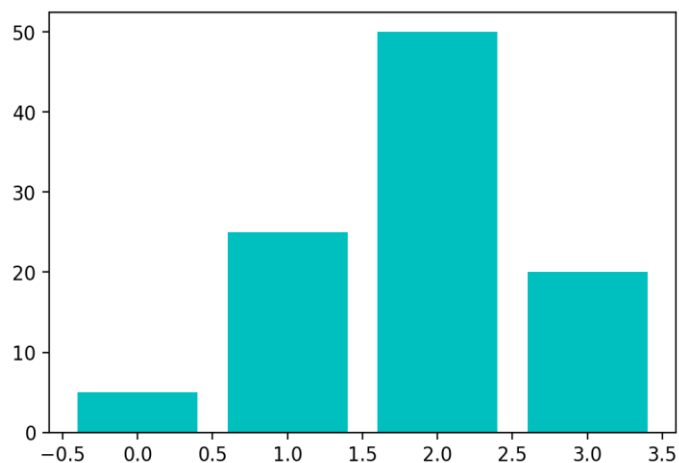
#define array

```
data= [5. , 25. , 50. , 20.]
```

```
plt.bar(range(len(data)), data,color='c')
```

```
plt.show()
```

Output:



4. Histogram

- **Use Case:** Showing frequency distributions.
- **How it works:** Groups data into bins and shows the count of items in each bin.

Example:

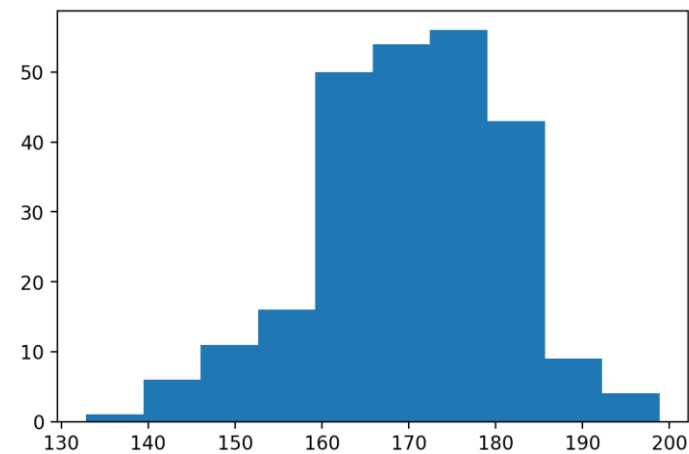
#draw random samples from random distributions.

```
x = np.random.normal(170, 10, 250)
```

#plot histograms

```
plt.hist(x) plt.show()
```

Output:



5. Pie Chart

- **Use Case:** Showing proportions of a whole.
- **How it works:** Displays data as a circle divided into slices.

Example:

#define the figure size

```
plt.figure(figsize=(7,7))
```

```
x = [25,30,45,10]
```

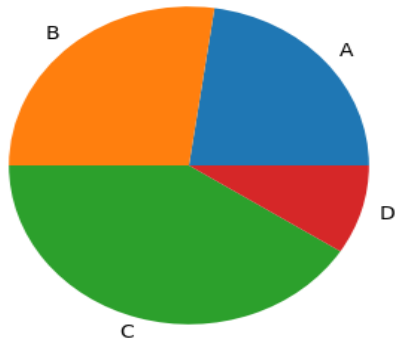
#labels of the pie chart

```
labels = ['A','B','C','D']
```

```
plt.pie(x, labels=labels)
```

```
plt.show()
```

Output:



6. Box Plot (or Whisker Plot)

- **Use Case:** Summarizing data distribution and detecting outliers.
- **How it works:** Shows quartiles, medians, and outliers of a dataset.

Example:

```
#create the random values by using numpy
```

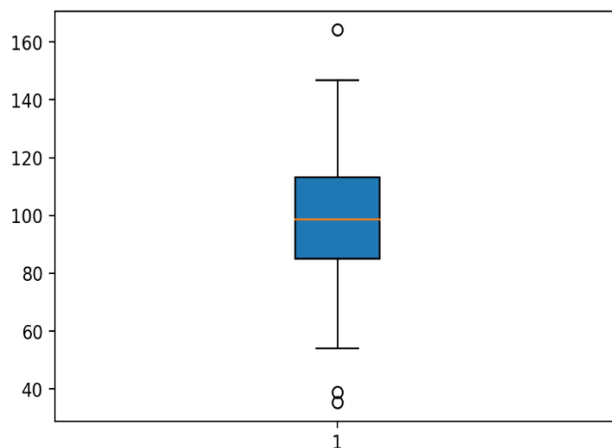
```
values= np.random.normal(100, 20, 300)
```

```
#creating the plot by boxplot() function which is available in matplotlib
```

```
plt.boxplot(values,patch_artist=True,vert=True)
```

```
plt.show()
```

Output:



7. Heatmap

- **Use Case:** Visualizing matrix-like data (e.g., correlations).
- **How it works:** Displays values as color-shaded cells.

Example:

```
import numpy as np
data = np.random.rand(3, 3)
plt.imshow(data, cmap='hot', interpolation='nearest')
plt.title("Heatmap")
plt.show()
```

8. Stacked Bar Plot

- **Use Case:** Comparing parts of different categories.
- **How it works:** Stacks multiple bars in a single bar

Example:

```
x = ['A', 'B', 'C']
y1 = [5, 7, 3]
y2 = [3, 8, 5]
plt.bar(x, y1, label='Set 1')
plt.bar(x, y2, bottom=y1, label='Set 2')
plt.legend()
plt.title("Stacked Bar Plot")
plt.show()
```

9. Area Plot

- **Use Case:** Similar to line plots, but emphasizes the magnitude under the line.
- **How it works:** Fills the area below the line plot.

Example:

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



```
y = [2, 7, 14, 17, 20, 27, 30, 38,
25, 18, 6, 1]

plt.plot(np.arange(12),y,
color="blue", alpha=0.6,
linewidth=2)

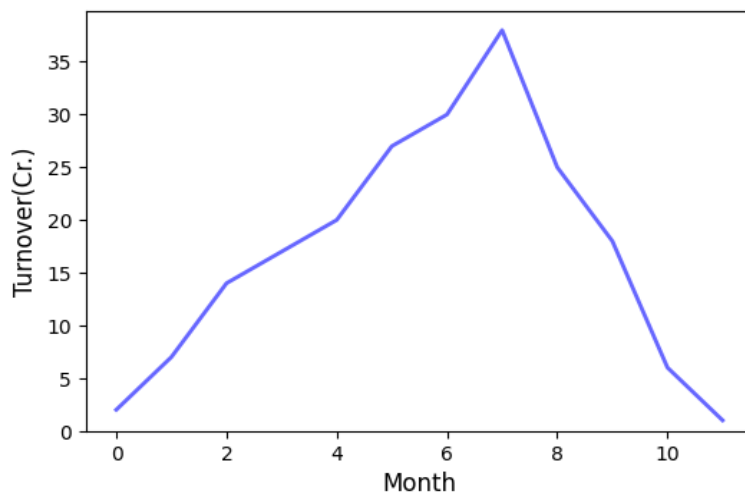
plt.xlabel('Month', size=12)

plt.ylabel('Turnover(Cr.)', size=12)

plt.ylim(bottom=0)

plt.show()
```

Output:



10.3D Graph

Now that you have seen some simple graphs, it's time to check some complex ones, i.e., 3-D graphs. Initially, Matplotlib was built for 2-dimensional graphs, but later, 3-D graphs were added. Let's check how you can plot a 3-D graph in Matplotlib.

Example:

```
ax = plt.axes(projection='3d')

# Data for a three-dimensional line

zline = np.linspace(0, 15, 1000)

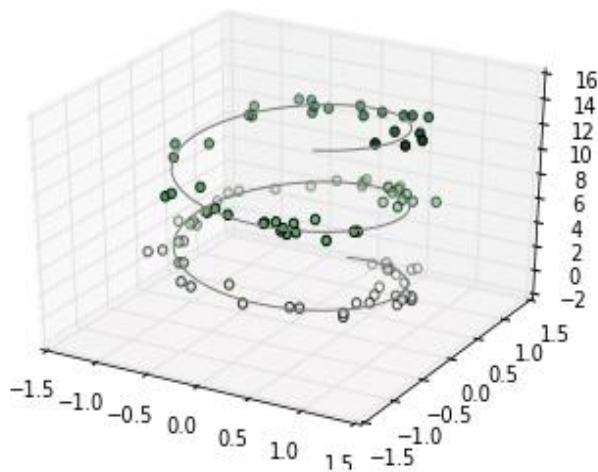
xline = np.sin(zline)

yline = np.cos(zline)

ax.plot3D(xline, yline, zline, 'gray')
```

```
# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 *
np.random.randn(100)
ydata = np.cos(zdata) + 0.1 *
np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata, c=zdata,
cmap='Greens');
```

Output:



Resources:

[Official Matplotlib Documentation](#)

[Matplotlib Tutorial \(Python Graphing\)](#)

[Matplotlib Examples and Tutorials](#)

Overview of Pandas:

- Pandas is a powerful data analysis and manipulation library for Python.
- It primarily handles structured data (such as dataframes) and offers tools for cleaning, transforming, and analyzing data efficiently.
- While it is not primarily a plotting library, Pandas integrates with Matplotlib to quickly generate plots from dataframes, making it ideal for exploratory data analysis (EDA).
- Pandas is an open-source data analysis and manipulation tool, highly regarded for its ease of use and flexibility.
- It is built on top of the Python programming language and offers powerful data structures for efficiently handling and analyzing large datasets.
- pandas (styled as pandas) is a [software library](#) written for the [Python programming language](#) for data manipulation and [analysis](#).
- In particular, it offers [data structures](#) and operations for manipulating numerical tables and [time series](#).
- It is [free software](#) released under the [three-clause BSD license](#).

Key Features of Pandas

Pandas provides several key features that make it an indispensable tool for data scientists and analysts working with Python:

- **Handling of Missing Data:** Pandas is adept at handling missing data and provides mechanisms for representing data as NaN, NA, or NaT without compromising the dataset's integrity.
- **Size Mutability:** It allows for dynamic modification of data structures, enabling the insertion and deletion of columns in DataFrames.
- **Data Alignment:** Pandas can automatically align data in computations or allow users to explicitly align datasets to a set of labels.
- **Group By Functionality:** It offers robust capabilities for performing splitapply-combine operations on datasets, which is essential for both aggregating and transforming data.
- **Data Conversion:** The library simplifies the conversion of disparate data types into uniform DataFrames.
- **Data Slicing and Indexing:** Pandas provides intuitive methods for slicing, indexing, and subsetting large datasets.

What is Pandas Library in Python?

Pandas is a powerful and versatile library that simplifies the tasks of data manipulation in [Python](#). Pandas is well-suited for working with tabular data, such as spreadsheets or SQL tables.

The Pandas library is an essential tool for data analysts, scientists, and engineers working with structured data in Python.

What is Python Pandas used for?

The Pandas library is generally used for data science, but have you wondered why? This is because the Pandas library is used in conjunction with other libraries that are used for data science.

It is built on top of the [NumPy library](#) which means that a lot of the structures of NumPy are used or replicated in Pandas.

The data produced by Pandas is often used as input for plotting functions in [Matplotlib](#), statistical analysis in [SciPy](#), and [machine learning algorithms](#) in [Scikit-learn](#).

You must be wondering, Why should you use the Pandas Library. Python's Pandas library is the best tool to analyze, clean, and manipulate data.

Here is a list of things that we can do using Pandas.

- Data set cleaning, merging, and joining.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
- Columns can be inserted and deleted from DataFrame and higherdimensional objects.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

Library Overview:

1.Data Frame: Two-dimensional, size-mutable, and heterogeneous tabular data structure.

2.Series: One-dimensional array-like object that can hold any data type.

3.Data Handling: Offers a rich set of functions for data manipulation, such as filtering, grouping, and aggregating.

4.Integration: Can generate simple plots using `.plot()` methods that leverage Matplotlib behind the scenes.

Common Use Cases:

- 1.Data cleaning and preparation
- 2.Fast exploratory data analysis (EDA)
- 3.Generating simple plots directly from Data Frames

Advantages of Pandas

1. Easy-to-Use Data Structures
2. Data Handling and Cleaning
3. Flexible Input and Output
4. Fast and Efficient
5. Powerful Data Aggregation and Grouping
6. Time Series Support
7. Visualization
8. Community Support and Documentation

Disadvantages of Pandas

1. Performance Issues with Large Datasets
2. Memory Consumption
3. Steep Learning Curve for Beginners
4. Inconsistent Performance in Some Operations
5. Chained Indexing Issues
6. Limited Parallelization Support
7. Error Handling Can Be Tricky

Resources:

Official Pandas Documentation

Pandas Plotting Documentation

Pandas Quick Start Guide

Different types of Graph in Pandas:

1. Line Plot
2. Bar Plot (Vertical and Horizontal)
3. Scatter Plot
4. Histogram
5. Pie Chart
6. Box Plot (Whisker Plot)
7. Area Plot
8. Hexbin Plot
9. Density Plot (KDE)
10. Multiple Subplots
11. Stacked Bar Plot
12. Lag Plot

1. Line Plot

- **Use Case:** Showing trends over time or sequential data.
- **Method:** plot.line()

Example

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'Year': [2015, 2016, 2017, 2018, 2019, 2020],
    'Sales': [200, 250, 300, 350, 400, 450],
    'Profit': [20, 30, 50, 70, 90, 110]
}

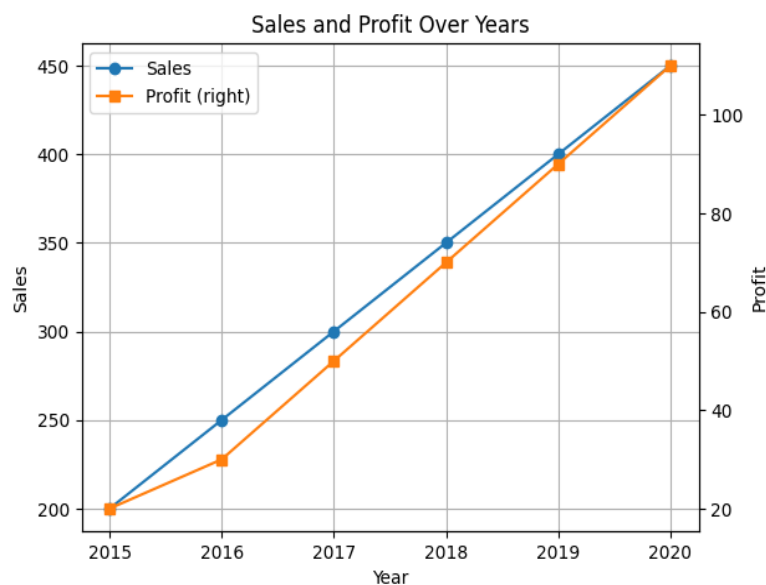
df = pd.DataFrame(data)
```

```

ax = df.plot(x='Year', y='Sales', kind='line', marker='o', title='Sales and
Profit Over Years')
df.plot(x='Year', y='Profit', kind='line', marker='s', ax=ax,
secondary_y=True)
ax.set_ylabel('Sales')
ax.right_ax.set_ylabel('Profit')
ax.grid(True)
plt.show()

```

Output:



2. Bar Plot (Vertical and Horizontal)

- **Use Case:** Comparing categorical data.

Vertical Bar Plot:

```
df.plot(x='Year', y='Sales', kind='bar', title='Sales by Year')
```

Horizontal Bar Plot:

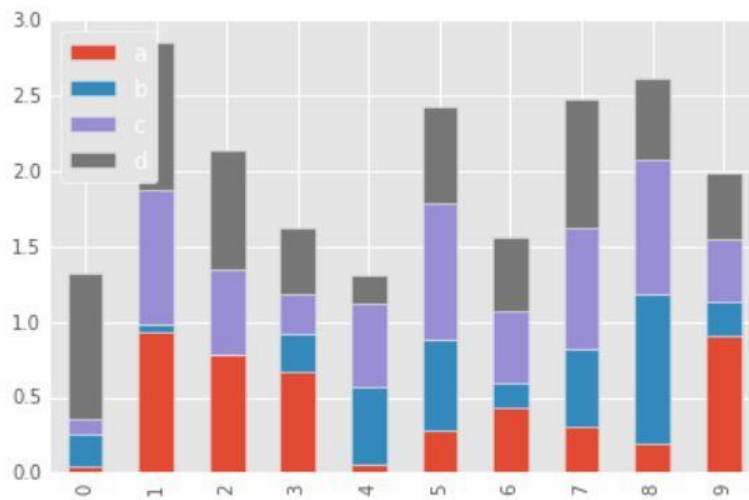
```
df.plot(x='Year', y='Sales', kind='barh', title='Sales by Year (Horizontal)')
```

Example:

```
df.plot.bar(stacked=True)
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12657bb38>
```



3. Scatter Plot

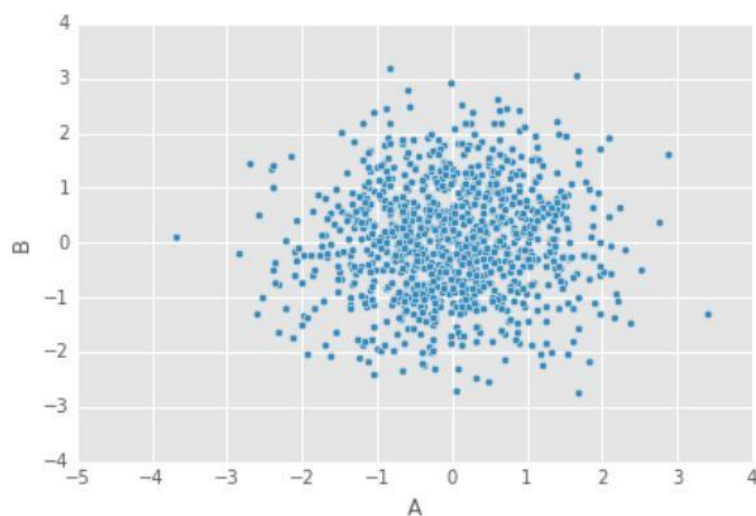
- **Use Case:** Visualizing relationships between two variables.
- **Method:** `plot.scatter()`

Example

```
df.plot.scatter(x='A', y='B')
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x126b90fd0>
```



4. Histogram

- **Use Case:** Displaying the frequency distribution of a dataset.
- **Method:** `plot.hist()`

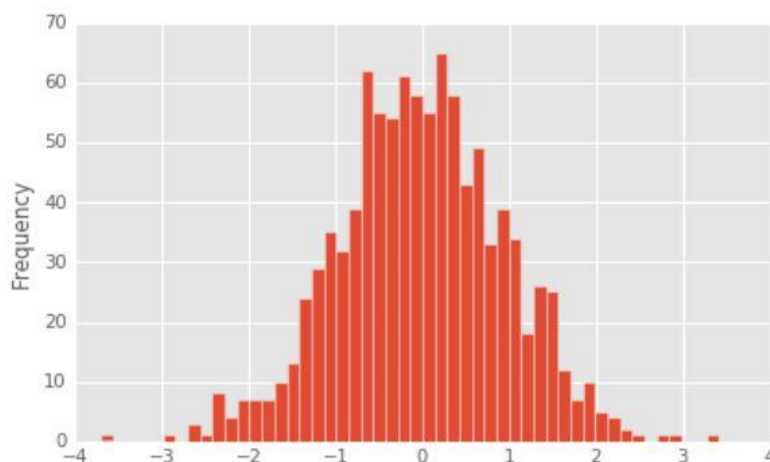
Example

```
data = pd.Series([1, 2, 2, 3, 3, 3, 4, 5, 5, 6])
```

```
data.plot(kind='hist', bins=5, title='Histogram of Values')
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12685e9b0>
```



5. Pie Chart

- **Use Case:** Showing proportions of a whole.
- **Method:** `plot.pie()`

Example

```
data = pd.Series([30, 40, 20, 10],
```

```
index=['A', 'B', 'C', 'D'])
```

```
data.plot(kind='pie', autopct='%1.1f%%', title='Category Distribution')
```

6. Box Plot (Whisker Plot)

- **Use Case:** Summarizing data distribution and identifying outliers.
- **Method:** `plot.box()`

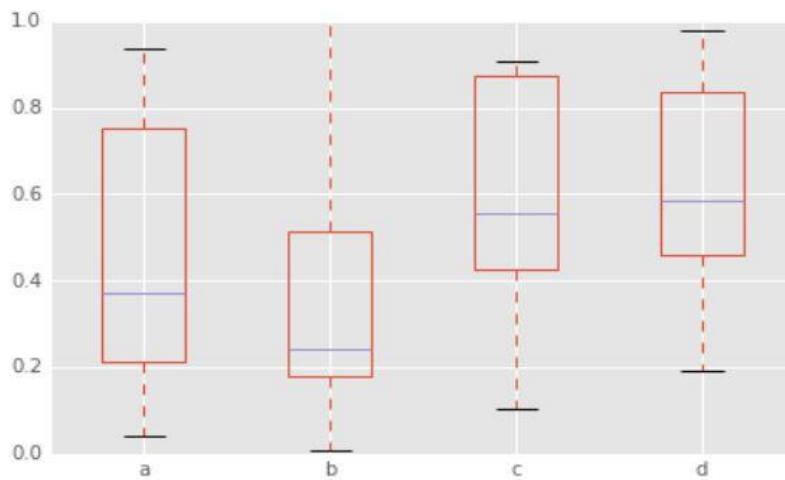
Example

```
df = pd.DataFrame({'A': [5, 7, 8, 5, 6], 'B': [10, 15, 8, 7, 9]})
```

```
df.plot(kind='box', title='Box Plot of Data')
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1271fa198>
```



7. Area Plot

- **Use Case:** Visualizing the magnitude of values over time.
- **Method:** `plot.area()`

Example

```
df.plot.area(alpha=0.4)
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x126222978>
```



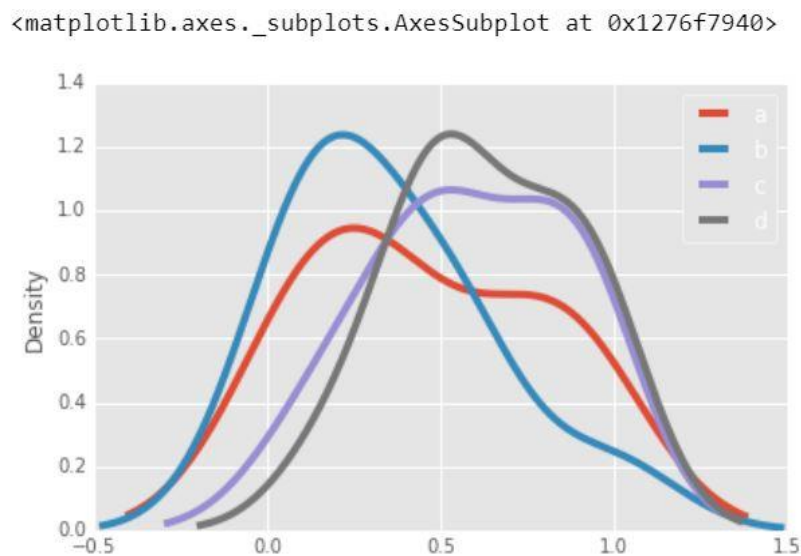
8. Density Plot (KDE)

- **Use Case:** Showing the probability density of a continuous variable.
- **Method:** `plot.kde()`

Example

```
data = pd.Series([1, 2, 2, 3, 3, 4, 5, 6])  
data.plot(kind='kde', title='Density Plot')
```

Output:



9. Multiple Subplots

- **Use Case:** Displaying multiple graphs within a single figure.
- **Method:** `subplots=True`

Example

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})  
df.plot(subplots=True, layout=(1, 2),  
figsize=(8, 4), title='Multiple Subplots')
```

10. Stacked Bar Plot

- **Use Case:** Comparing parts of multiple groups.
- **Method:** `plot.bar(stacked=True)`

Example

```
df = pd.DataFrame({'A': [3, 2, 5], 'B': [1, 4, 6]},  
index=['X', 'Y', 'Z'])  
df.plot(kind='bar', stacked=True, title='Stacked Bar Plot')
```

11. Lag Plot

- **Use Case:** Visualizing the relationship between a time series and a lagged version of itself.
- **Method:** Uses Pandas along with Matplotlib.

Example

```
from pandas.plotting import lag_plot  
data = pd.Series([1, 2, 3, 4, 3, 2, 1])  
lag_plot(data) plt.title('Lag Plot')  
plt.show()
```

Comparison: Matplotlib vs Pandas

Aspect	Matplotlib	Pandas
Purpose	A general-purpose plotting library for creating custom visualizations.	Primarily a data manipulation library with basic plotting capabilities.
Ease of Use	Requires more code and customization for plotting.	Easier to generate plots directly from Data Frames and Series.
Complexity	Suitable for complex, highly customized visualizations.	Best for quick, simple visualizations during data analysis.
Integration	Can integrate with Pandas, Seaborn, and other libraries for advanced features.	Uses Matplotlib as its backend for plotting functions.
Plot Types	Supports a wide range of plots (e.g., 3D plots, polar plots, animations).	Supports basic plots like line, bar, pie, scatter, and histogram.
Customization	Highly customizable (control over axes, labels, grids, etc.)	Limited customization options.
Performance	Requires more effort and code to optimize performance.	Faster for small to medium datasets but lacks scalability for large datasets.

