TITLE: VISUALIZATION

NAME: Seajal Uniyal

INTERNSHIP: ShadowFox Data Science Intern

VISUALIZATION

Data visualization is the process of turning raw data into visual formats like charts and graphs, making it easier to understand and interpret. It helps quickly identify trends, patterns, and outliers.

In data science, it is important because it simplifies complex data, supports clear communication of insights, and aids in making data-driven decisions.

In python, data visualization has multiple libraries like matplotlib, seaborn, plotly, pandas etc.

Let us dive deeper into matplotlib and pandas.

MATPLOTLIB

A Matplotlib plot is made up of the following components:

1. **Figure**
2. **Axes**
3. **Axis**
4. **Artists**

Figure
The Figure is the overall container that holds everything you see in a plot. It can include one plot or multiple subplots.

Axes
Axes represent the actual plotting area inside the Figure. A plot can have two or three axes. Functions like set_xlabel() and set_ylabel() are used to label the x-axis and y-axis.

Axis
An Axis handles the scaling and generates the ticks and tick labels on the plot.

Artists
Everything visible in a Figure, such as lines, text, and shapes, is considered an Artist in Matplotlib.

Pyplot

Pyplot is a submodule of Matplotlib that provides functions to easily create different types of plots like bar charts, scatter plots, pie charts, histograms, and area charts. Typically, it is imported as:

```
[ ]: import matplotlib.pyplot as plt
     import numpy as np
```

Some Plots in Matplotlib:

Line Chart

Shows data points connected by lines, useful for observing trends or changes over time (e.g., stock prices, temperature variations).

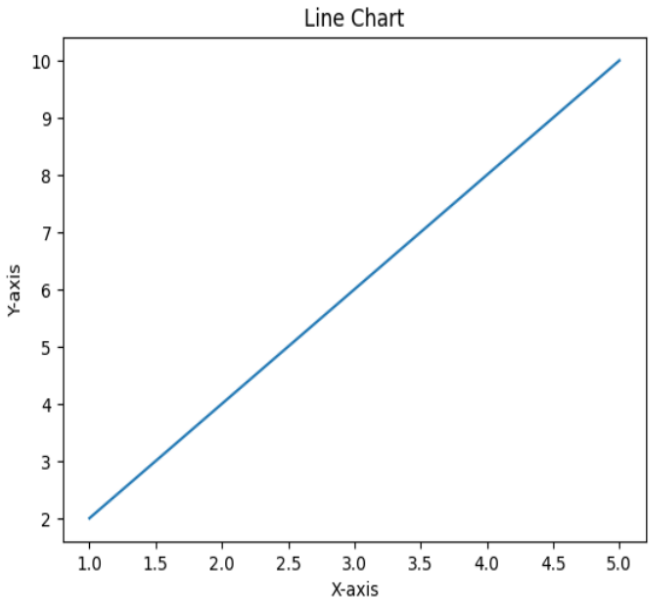
Code Snippet:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

Output:



Description

- x and y define data points (x → 1 to 5, y → double of x).
- plt.plot() draws a continuous line connecting those points.
- plt.title(), plt.xlabel(), plt.ylabel() add labels for clarity.
- plt.show() displays the plot.

Bar Chart

Represents data with rectangular bars, used for comparing categories or groups (e.g., sales of different products).

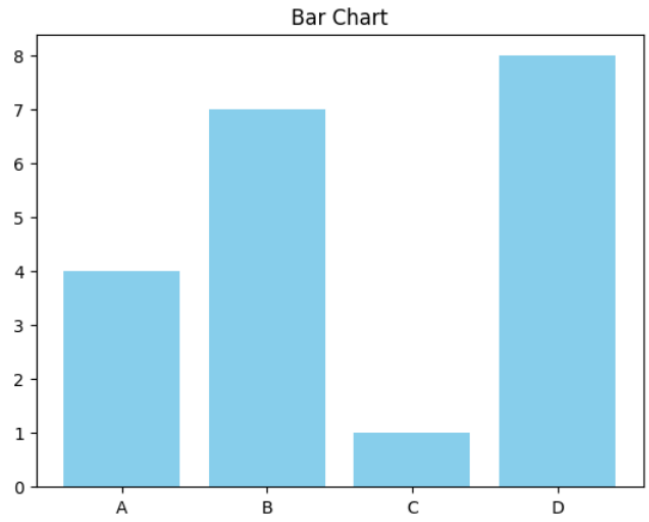
Code Snippet:

```
import matplotlib.pyplot as plt

categories = ['A','B','C','D']
values = [4,7,1,8]

plt.bar(categories, values, color='skyblue')
plt.title('Bar Chart')
plt.show()
```

Output:



Description

- categories is a list of names, values are numerical data for each category.
- plt.bar() creates vertical bars.
- color='skyblue' gives bars a light blue color.
- plt.title() adds a heading.

Histogram

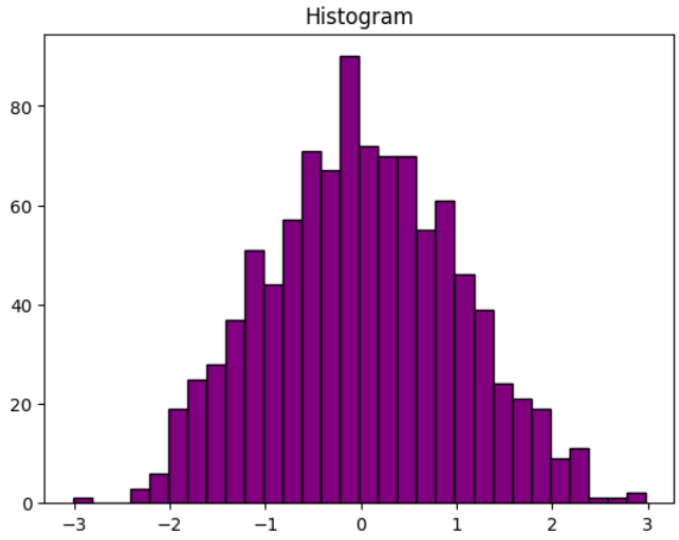
Displays the frequency distribution of data, helpful for understanding data spread (e.g., exam score distribution).

Code snippet:

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

data = np.random.randn(1000)  # random numbers
plt.hist(data, bins=30, color='purple', edgecolor='black')
plt.title('Histogram')
plt.show()
```

Output:



Description

- np.random.randn(1000) generates 1000 random numbers (bell curve style).
- plt.hist() divides values into 30 bins and shows how many values fall in each bin.
- edgecolor='black' gives each bar a black border.

Pie Chart

Represents parts of a whole as slices of a circle, good for showing percentage breakdowns (e.g., market share).

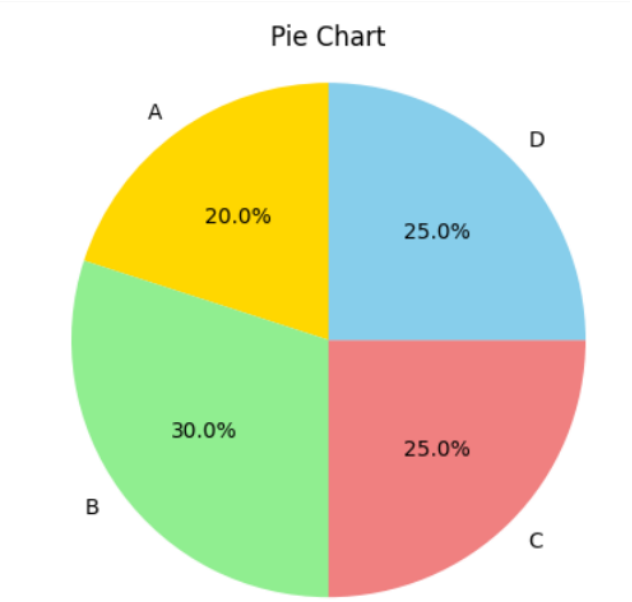
Code snippet:

```
import matplotlib.pyplot as plt

sizes = [20, 30, 25, 25]
labels = ['A', 'B', 'C', 'D']
colors = ['gold', 'lightgreen', 'lightcoral', 'skyblue']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart')
plt.axis('equal')  # makes it a circle
plt.show()
```

Output:



Description

- sizes = proportions for each category.
- labels = names of each category.
- autopct='%1.1f%%' shows percentages on the chart.
- startangle=90 rotates the chart to start from vertical.

Scatter Plot

Shows relationships between two variables, used for identifying correlations or patterns (e.g., height vs. weight analysis).

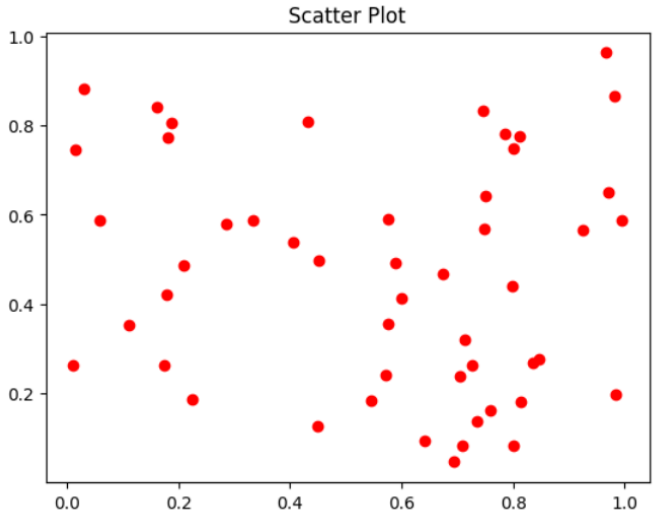
Code snippet:

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

x = np.random.rand(50)
y = np.random.rand(50)

plt.scatter(x, y, color='red')
plt.title('Scatter Plot')
plt.show()
```

Output:



Description

- np.random.rand(50) generates 50 random x and y values.
- plt.scatter() plots individual points (no connecting lines).
- color='red' makes points red**.**

Stack plot

Visualizes data changing over time, showing contribution of each part (e.g., energy consumption by source).

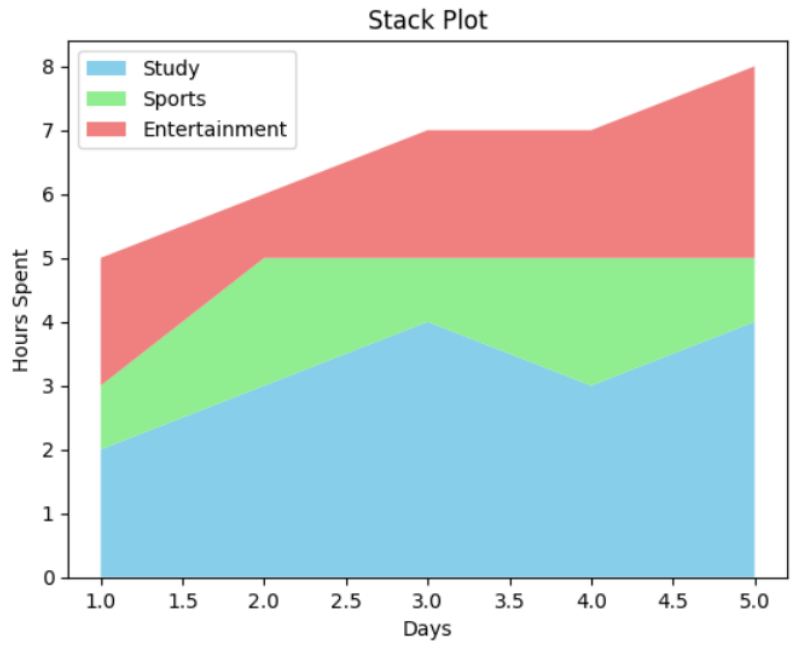
Code snippet:

```
import matplotlib.pyplot as plt

days = [1, 2, 3, 4, 5]
study = [2, 3, 4, 3, 4]
sports = [1, 2, 1, 2, 1]
entertainment = [2, 1, 2, 2, 3]

plt.stackplot(days, study, sports, entertainment,
              labels=['Study','Sports','Entertainment'],
              colors=['skyblue','lightgreen','lightcoral'])
plt.legend(loc='upper left')
plt.title('Stack Plot')
plt.xlabel('Days')
plt.ylabel('Hours Spent')
plt.show()
```

Output:



Description

- Imports data for days and three activities.
- Uses plt.stackplot() to create stacked areas.
- Adds legend, title, axis labels.
- Displays plot with plt.show().
- Stacked areas showing hours spent on Study, Sports, and Entertainment across days.

Area Plot

Similar to a line plot but with filled area, used to emphasize magnitude (e.g., cumulative rainfall over months).

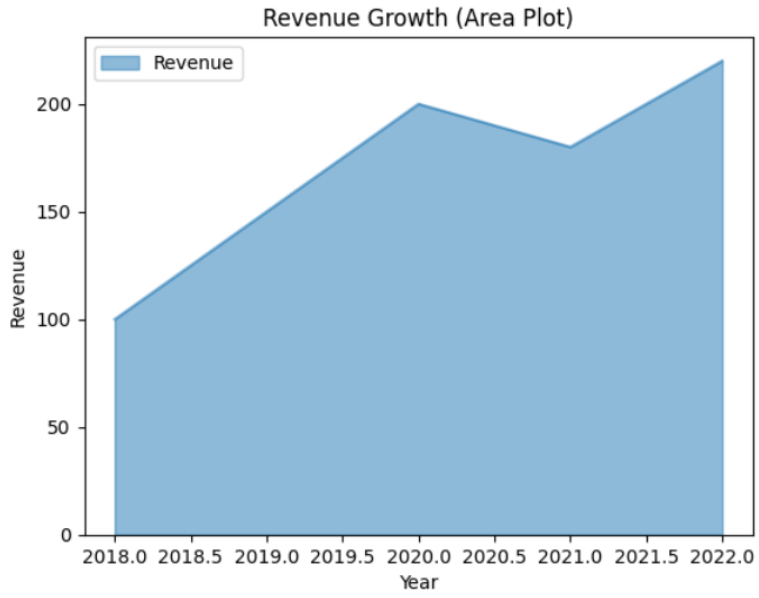
Code snippet:

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

data = {'Year':[2018,2019,2020,2021,2022],
        'Revenue':[100,150,200,180,220]}
df = pd.DataFrame(data)

df.plot(x='Year', y='Revenue', kind='area', alpha=0.5)
plt.title('Revenue Growth (Area Plot)')
plt.ylabel('Revenue')
plt.show()
```

Output:



Description

- Creates DataFrame with Year and Revenue.
- Uses df.plot(kind='area') for filled area chart.
- Area under revenue curve filled, showing revenue growth over years.

Stem Plot

Displays discrete data points with stems, useful for signal processing or sequence data.

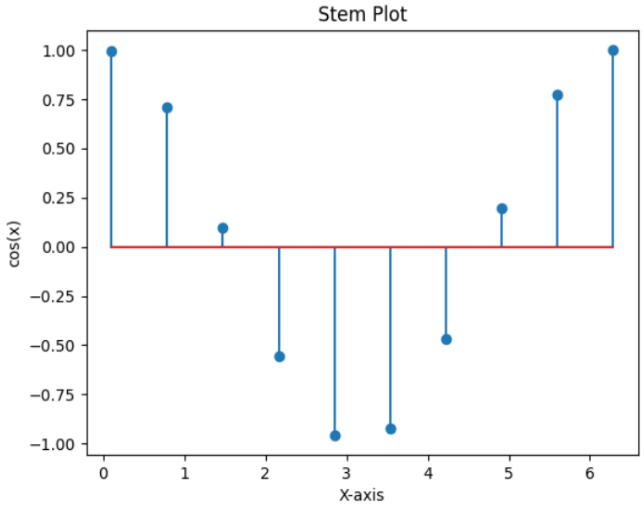
Code snippet:

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

x = np.linspace(0.1, 2 * np.pi, 10)
y = np.cos(x)

plt.stem(x, y)
plt.title('Stem Plot')
plt.xlabel('X-axis')
plt.ylabel('cos(x)')
plt.show()
```

Output:



Description:

- Generates 10 evenly spaced x values.
- Computes cosine values.
- Uses plt.stem() to draw stems from baseline.
- Discrete vertical lines with dots representing cosine values.

3D Plot

Visualizes data in three dimensions, helpful for spatial data or advanced scientific analysis.

Code snippet:

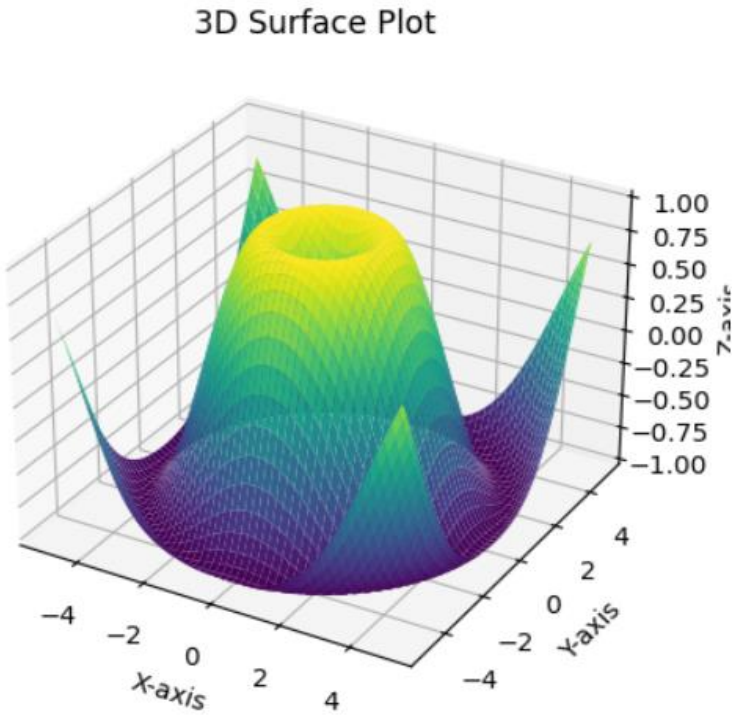
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

ax.plot_surface(X, Y, Z, cmap='viridis')
ax.set_title('3D Surface Plot')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
plt.show()
```

Output:



PANDAS

Pandas is a powerful data manipulation and analysis library in Python, built on top of NumPy. It is widely used for handling structured data like tables and time series. Pandas provides flexible data structures such as **Series** (1D) and **DataFrame** (2D), making it easier to clean, filter, transform, and analyze data efficiently. It also offers built-in methods for quick visualization, mainly relying on Matplotlib in the backend.

Common Plot Types in Pandas and Their Use Cases:

1. **Line Plot** – DataFrame.plot.line()
   Used for visualizing trends over time or continuous data (e.g., stock prices, temperature changes).
2. **Bar Plot** – DataFrame.plot.bar()
   Helps compare different categories or groups (e.g., sales by product).
3. **Histogram** – DataFrame.plot.hist()
   Displays frequency distribution of numerical data (e.g., age distribution).
4. **Box Plot** – DataFrame.plot.box()
   Shows data spread and helps detect outliers (e.g., salary distribution).
5. **Area Plot** – DataFrame.plot.area()
   Highlights magnitude of change over time (e.g., monthly revenue growth).

6. **Scatter Plot** – DataFrame.plot.scatter(x, y)
   Used for finding relationships between two variables (e.g., marketing spend vs. sales).
7. **Pie Chart** – DataFrame.plot.pie()
   Shows proportions of categories (e.g., market share of companies).
8. **Hexbin Plot** – DataFrame.plot.hexbin(x, y)
   Useful for visualizing the density of points in large datasets (e.g., customer location clustering).

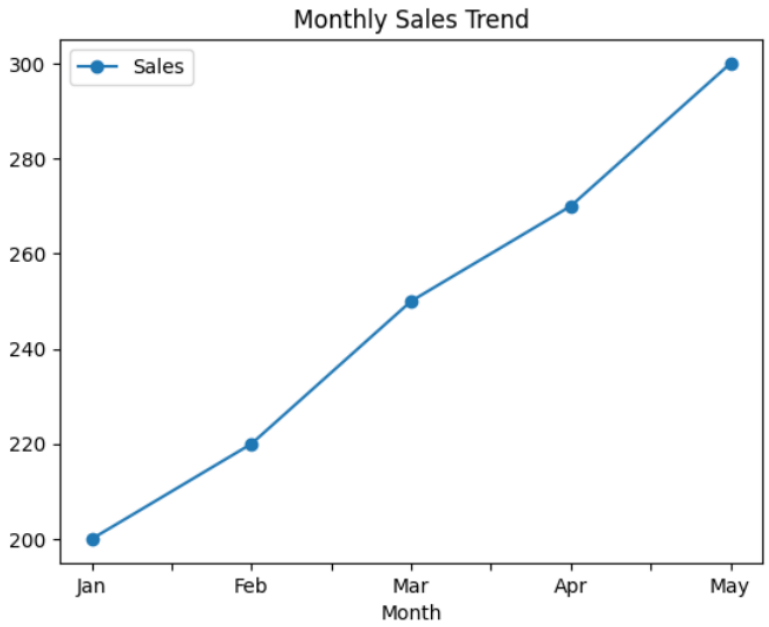
Line Plot

Code snippet:

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

df = pd.DataFrame({'Month':['Jan','Feb','Mar','Apr','May'],
                   'Sales':[200, 220, 250, 270, 300]})
df.plot.line(x='Month', y='Sales', marker='o')
plt.title('Monthly Sales Trend')
plt.show()
```

Output:



Description:

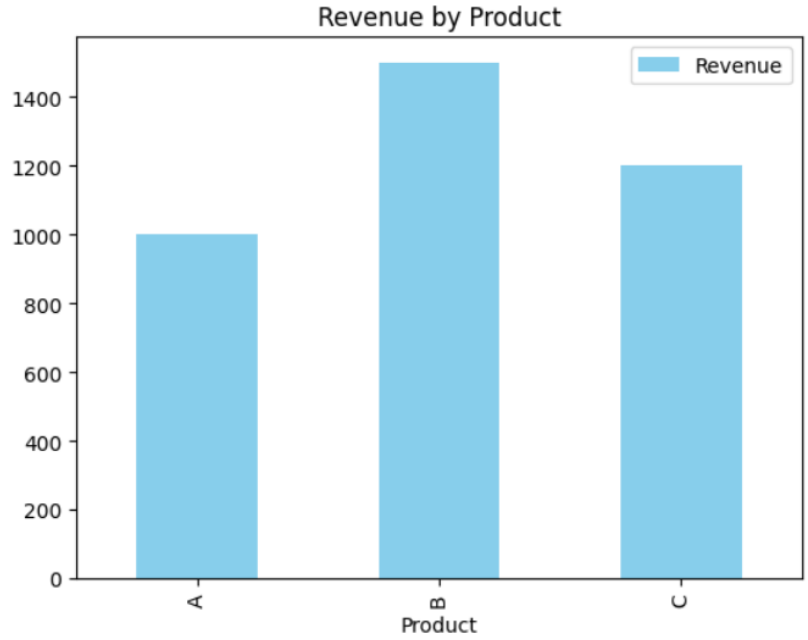
- Creates a DataFrame with months and sales.
- Uses plot.line() to draw a line connecting sales values.
- A line graph showing how sales increased from January to May.

Bar Plot

Code snippet:

```
df = pd.DataFrame({'Product':['A','B','C'],
                   'Revenue':[1000, 1500, 1200]})
df.plot.bar(x='Product', y='Revenue', color='skyblue')
plt.title('Revenue by Product')
plt.show()
```

Output:



Description:

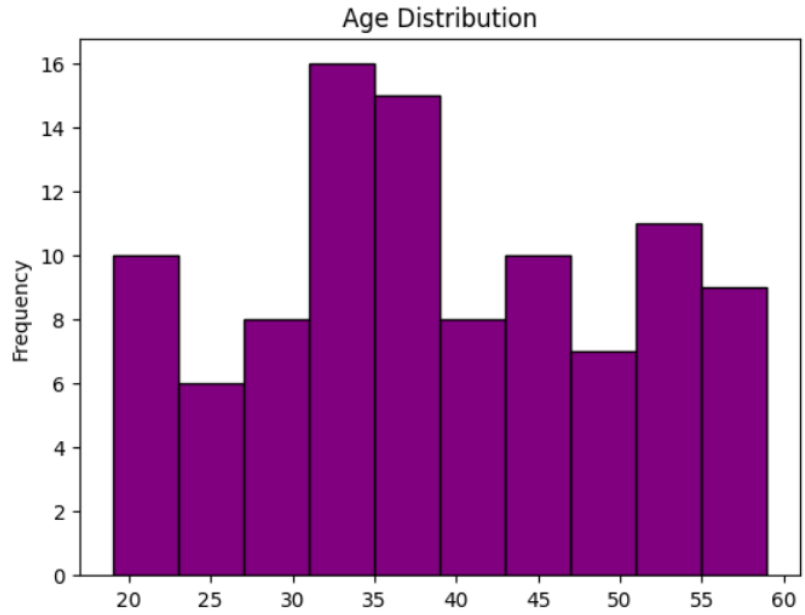
- Creates a DataFrame with products and revenue.
- Uses plot.bar() to draw vertical bars for each product.
- A bar chart comparing revenue among products A, B, and C.

Histogram

Code snippet:

```
import numpy as np
df = pd.DataFrame({'Age': np.random.randint(18, 60, 100)})
df['Age'].plot.hist(bins=10, color='purple', edgecolor='black')
plt.title('Age Distribution')
plt.show()
```

Output:



Description:

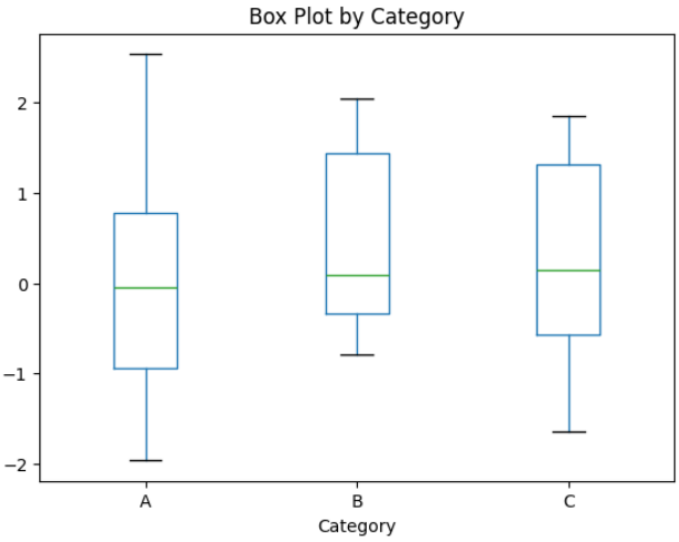
- Generates 100 random ages between 18 and 60.
- Uses plot.hist() to create frequency distribution with 10 bins.
- Histogram showing how many people fall into each age group.

Box Plot

Code snippet:

```
import numpy as np
df = pd.DataFrame({'Category': ['A']*10 + ['B']*10 + ['C']*10,
                   'Values': np.random.randn(30)})
df.boxplot(by='Category', column=['Values'], grid=False)
plt.title('Box Plot by Category')
plt.suptitle('')
plt.show()
```

Output:



Description:

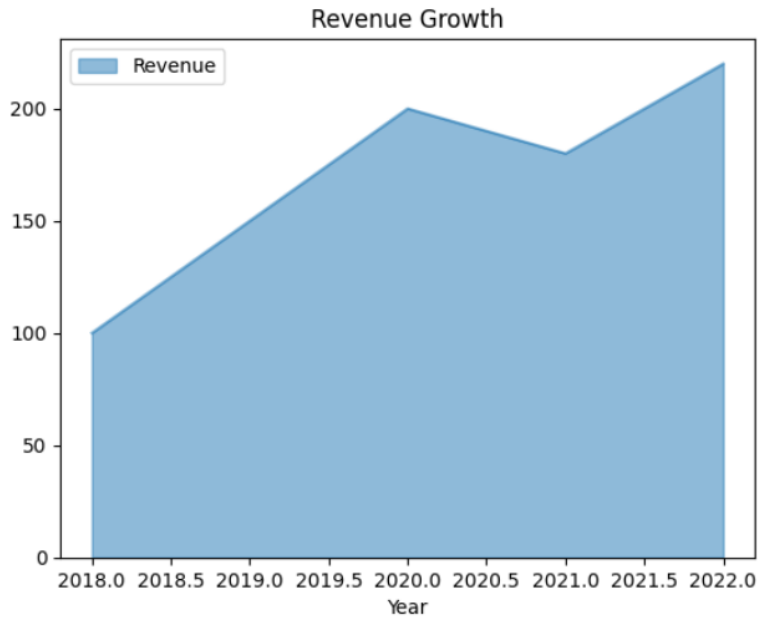
- Creates a DataFrame with three categories and random values.
- Uses boxplot() grouped by category to show spread and outliers.
- Three box plots showing data spread, quartiles, and any outliers for each category.

Area Plot

Code snippet:

```
df = pd.DataFrame({'Year':[2018,2019,2020,2021,2022],
                   'Revenue':[100,150,200,180,220]})
df.plot.area(x='Year', y='Revenue', alpha=0.5)
plt.title('Revenue Growth')
plt.show()
```

Output:



Description:

- Creates a DataFrame with yearly revenue.
- Uses plot.area() to fill the area under the line.
- A shaded area chart showing how revenue changed over the years.

Scatter Plot

Code snippet:

```
df = pd.DataFrame({'Marketing Spend':[10,20,30,40,50],
                   'Sales':[15,25,35,50,60]})
df.plot.scatter(x='Marketing Spend', y='Sales', color='red')
plt.title('Marketing Spend vs Sales')
plt.show()
```

Output:

Description:

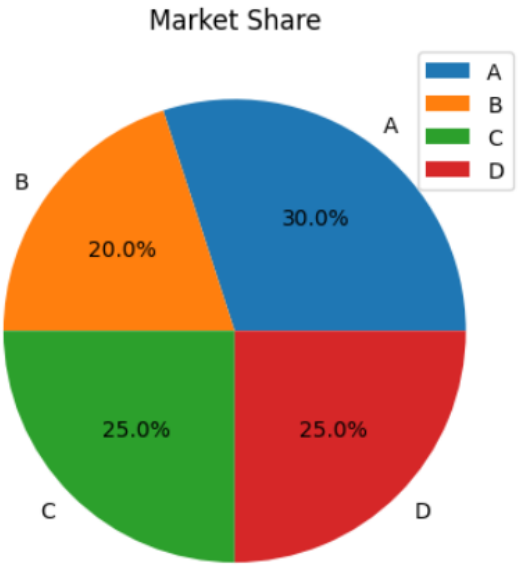
- Creates a DataFrame with marketing spend and sales.
- Uses plot.scatter() to plot points.
- A scatter plot showing positive correlation between marketing spend and sales.

Pie Chart

Code snippet:

```
df = pd.DataFrame({'Company':['A','B','C','D'],
                   'Market Share':[30, 20, 25, 25]})
df.set_index('Company').plot.pie(y='Market Share', autopct='%1.1f%%')
plt.title('Market Share')
plt.ylabel('')
plt.show()
```

Output:



Description:

- Creates a DataFrame with company market shares.
- Uses plot.pie() to draw a pie chart with percentage labels.
- A circular pie chart showing proportion of each company's market share.

Hexbin Plot

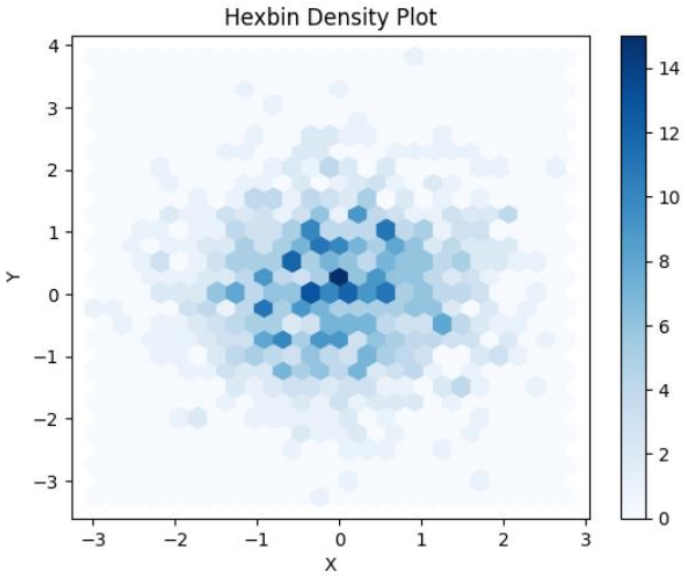
Code snippet:

python

CopyEdit

```python
df = pd.DataFrame({'X': np.random.randn(1000),
                   'Y': np.random.randn(1000)})
df.plot.hexbin(x='X', y='Y', gridsize=25, cmap='Blues')
plt.title('Hexbin Density Plot')
plt.show()
```

Output:



Description:

- Creates a DataFrame with 1000 random X and Y values.
- Uses plot.hexbin() to show density of points using hexagonal bins.
- A density map where darker blue areas indicate more points in that region.

COMPARISON OF MATPLOTLIB AND PANDAS

Matplotlib:

- **Core Library:** Matplotlib is a foundational Python library for creating static, animated, and interactive visualizations. It offers low-level control, allowing users to design any type of plot with detailed customization.

- **Flexibility:** Capable of producing a wide range of plots including 2D, 3D, and even interactive visualizations. Users have full control over every element of the plot (colors, styles, axes, labels, etc.).
- **Usage:** Often used when there is a need for customized, publication-quality graphics or complex visualizations beyond basic data plotting.
- **Integration:** Works seamlessly with NumPy, Pandas, SciPy, and other Python scientific libraries.

Advantages of Matplotlib:

- High customization and control.
- Wide variety of plot types.
- Mature ecosystem with strong community support.
- Suitable for complex and publication-quality visuals**.**

Pandas (Plotting):

- **High-Level Interface:** Pandas is primarily a data manipulation library, but it includes built-in plotting functionality that relies on Matplotlib under the hood. It provides an easy way to visualize data directly from Pandas DataFrames and Series.
- **Ease of Use:** Focuses on simplicity and speed, ideal for quick exploratory data analysis without needing detailed customization.
- **Plot Types:** Supports common plot types like line, bar, histogram, box, scatter, and pie charts directly from a DataFrame or Series with simple syntax.
- **Integration**: Naturally integrated with Pandas data structures, making it convenient for quick visualization of tabular data.

Advantages of Pandas Plotting:

- Quick and easy plotting directly from DataFrames.
- Minimal code required for basic visualizations.
- Great for exploratory data analysis.
- Built-in support for handling time series and tabular data plots.

Overall:

Matplotlib is best for detailed, highly customizable, and complex visualizations, while Pandas plotting is designed for quick, simple visualizations directly from tabular data. Many users use Pandas for initial exploratory plots and switch to Matplotlib for advanced or publication-quality graphics.