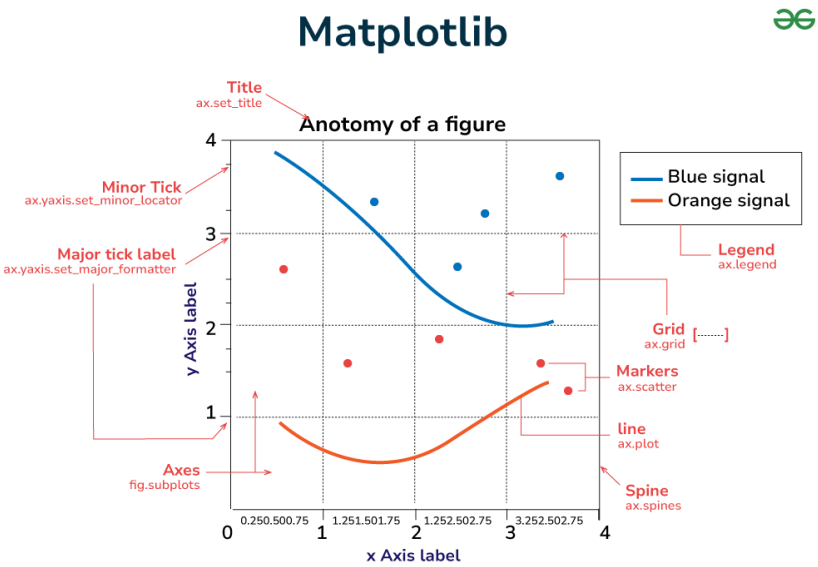
Matplotlib:

Matplotlib is a versatile and powerful visualization library in Python, widely appreciated for its simplicity and effectiveness. It is built upon NumPy arrays and designed to integrate seamlessly with the SciPy ecosystem. The library supports various plot types, such as line graphs, bar charts, scatter plots, histograms, and more.

Key Features of Matplotlib:

1. **Wide Variety of Plots:** Matplotlib supports numerous plot types, including line graphs, scatter plots, bar charts, histograms, pie charts, and more.
2. **Highly Customizable:** Users can control almost every aspect of the plot, such as line styles, colours, markers, labels, and annotations.
3. **Integration with NumPy:** The library works directly with NumPy arrays, making data plotting straightforward.
4. **Extensibility:** Matplotlib can be extended with add-ons like Seaborn, Pandas' plotting utilities, and Basemap for geographical visualizations.
5. **Cross-Platform Compatibility:** The library is platform-independent and functions smoothly on Windows, macOS, and Linux.
6. **Interactive Capabilities:** It supports interactive plotting with widgets and event handling, allowing dynamic data exploration.

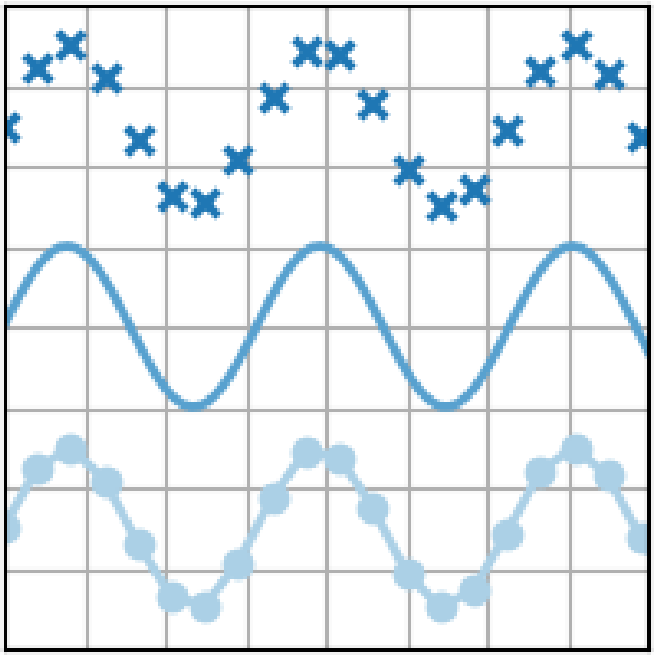
Understanding Matplotlib Figures: In Matplotlib, a figure acts as the main container that encompasses all the elements of a plot. It represents the canvas or window where the visualization is rendered.



Plot Types Based on Data Visualization:

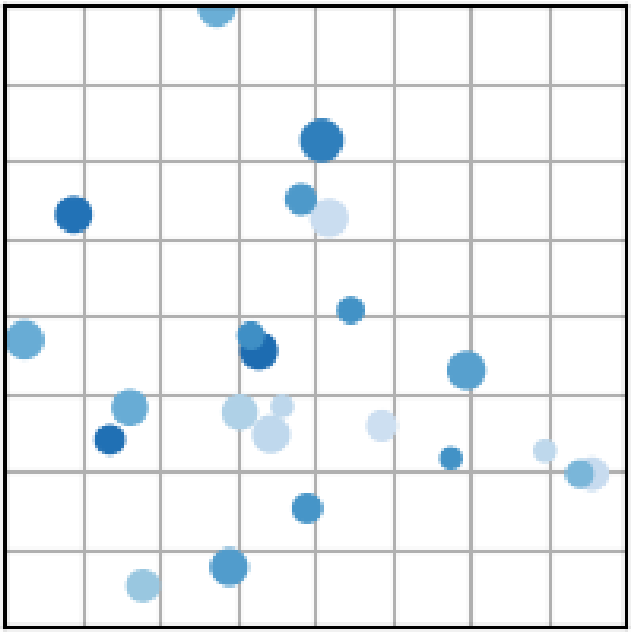
1. **Pairwise Data:**
    - `plot(x, y)`: Creates a simple line plot using x and y coordinates.

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

plt.style.use('_mpl-gallery')

# make data
x = np.linspace(0, 10, 100)
y = 4 + 1 * np.sin(2 * x)
x2 = np.linspace(0, 10, 25)
y2 = 4 + 1 * np.sin(2 * x2)

# plot
fig, ax = plt.subplots()

ax.plot(x2, y2 + 2.5, 'x', markeredgewidth=2)
ax.plot(x, y, linewidth=2.0)
ax.plot(x2, y2 - 2.5, 'o-', linewidth=2)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



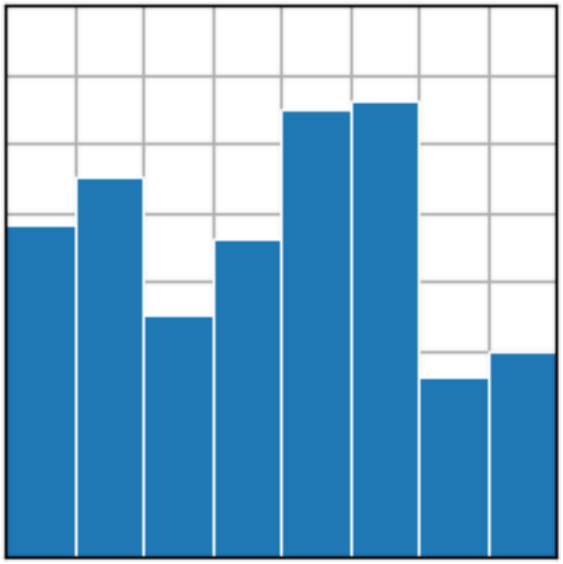
    - `scatter(x, y)`: Generates a scatter plot to depict pairwise data.

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

plt.style.use('_mpl-gallery')

# make the data
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
# size and color:
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# plot
fig, ax = plt.subplots()

ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



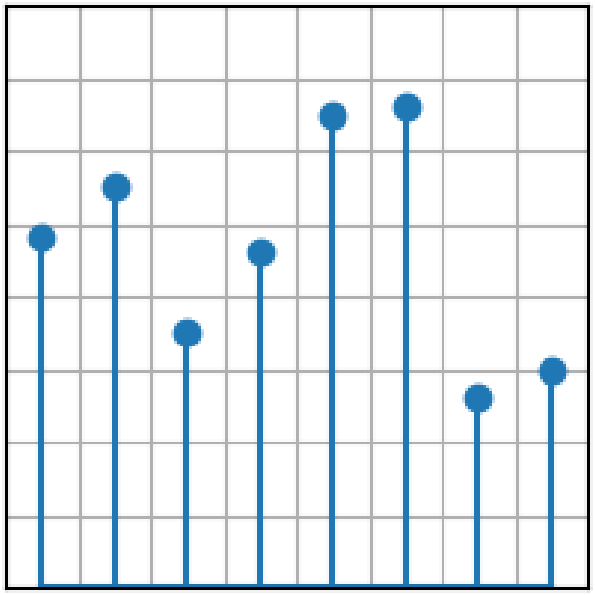
- `bar(x, height)`: Produces a bar chart for categorical data.

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

plt.style.use('_mpl-gallery')

# make data:
x = 0.5 + np.arange(8)
y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]

# plot
fig, ax = plt.subplots()

ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



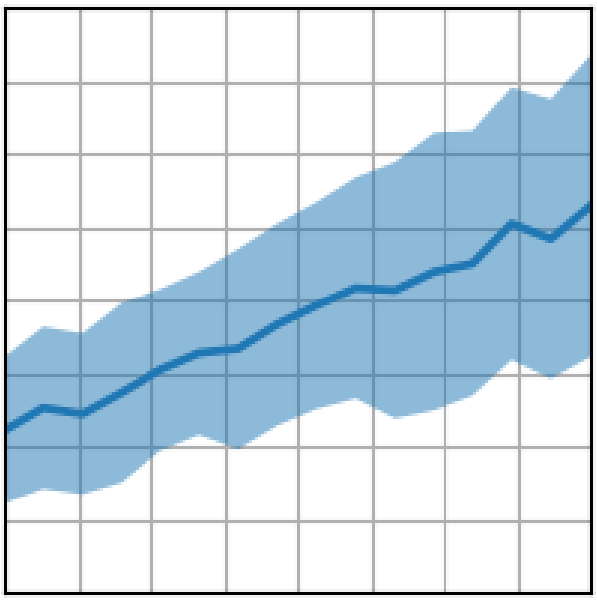
- `stem(x, y)`: Classifies variables as discrete or continuous.

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

plt.style.use('_mpl-gallery')

# make data
x = 0.5 + np.arange(8)
y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]

# plot
fig, ax = plt.subplots()

ax.stem(x, y)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



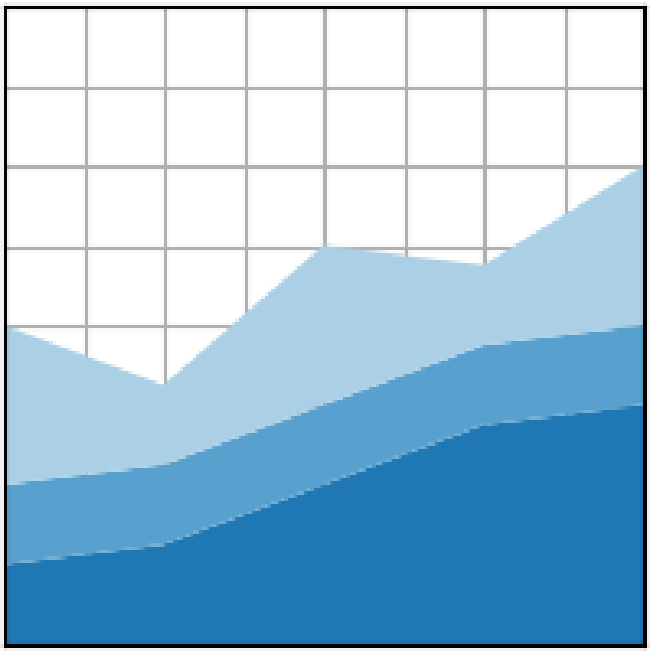
- `fill_between(x, y1, y2)`: Shades the area between specified ranges.

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

plt.style.use('_mpl-gallery')

# make data
np.random.seed(1)
x = np.linspace(0, 8, 16)
y1 = 3 + 4*x/8 + np.random.uniform(0.0, 0.5, len(x))
y2 = 1 + 2*x/8 + np.random.uniform(0.0, 0.5, len(x))

# plot
fig, ax = plt.subplots()

ax.fill_between(x, y1, y2, alpha=.5, linewidth=0)
ax.plot(x, (y1 + y2)/2, linewidth=2)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



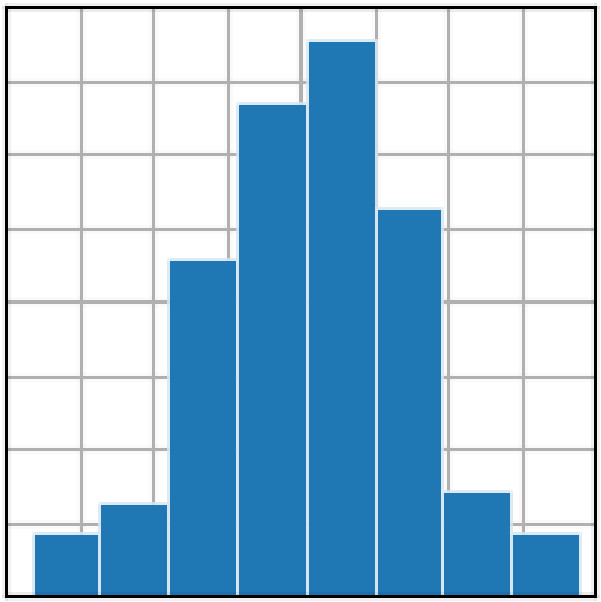
- `stackplot(x, y)`: Displays stacked data as a filled plot.

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

plt.style.use('_mpl-gallery')

# make data
x = np.arange(0, 10, 2)
ay = [1, 1.25, 2, 2.75, 3]
by = [1, 1, 1, 1, 1]
cy = [2, 1, 2, 1, 2]
y = np.vstack([ay, by, cy])

# plot
fig, ax = plt.subplots()

ax.stackplot(x, y)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



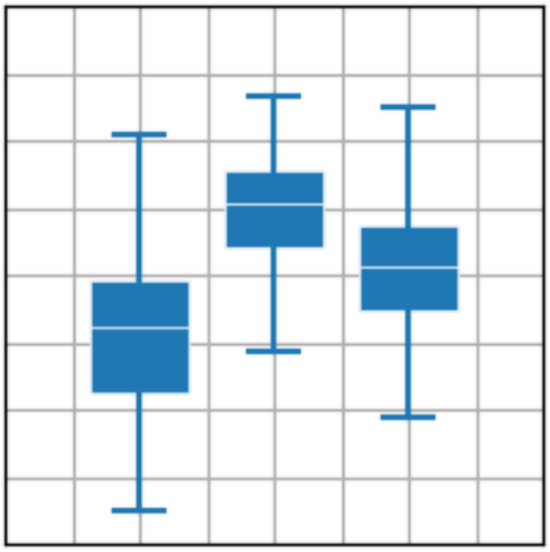
2. **Statistical Distributions:**
    - `hist(x)`: Displays numeric value distributions as bars.

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

plt.style.use('_mpl-gallery')

# make data
np.random.seed(1)
x = 4 + np.random.normal(0, 1.5, 200)

# plot:
fig, ax = plt.subplots()

ax.hist(x, bins=8, linewidth=0.5, edgecolor="white")

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 56), yticks=np.linspace(0, 56, 9))

plt.show()
```



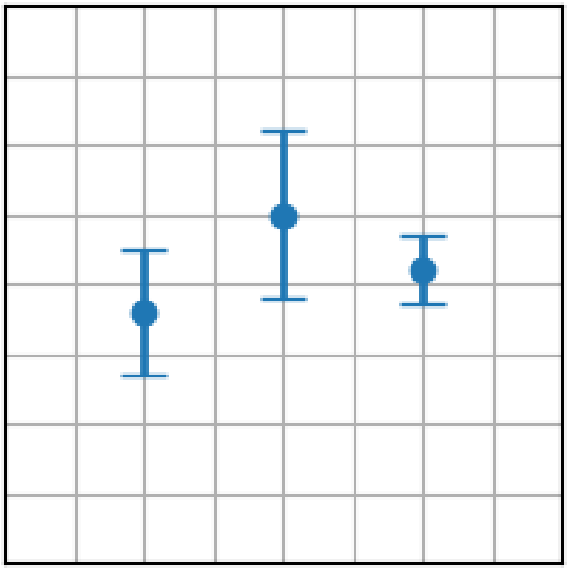
    - `boxplot(X)`: Visualizes data distribution using quartiles.

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

plt.style.use('_mpl-gallery')

# make data:
np.random.seed(10)
D = np.random.normal((3, 5, 4), (1.25, 1.00, 1.25), (100, 3))

# plot
fig, ax = plt.subplots()
VP = ax.boxplot(D, positions=[2, 4, 6], widths=1.5, patch_artist=True,
                showmeans=False, showfliers=False,
                medianprops={"color": "white", "linewidth": 0.5},
                boxprops={"facecolor": "C0", "edgecolor": "white",
                          "linewidth": 0.5},
                whiskerprops={"color": "C0", "linewidth": 1.5},
                capprops={"color": "C0", "linewidth": 1.5})

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



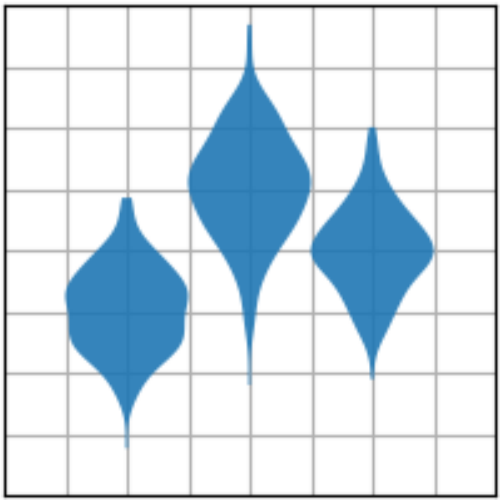
- `errorbar(x, y, yerr, xerr)`: Represents variability or uncertainty in data measurements.

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

plt.style.use('_mpl-gallery')

# make data:
np.random.seed(1)
x = [2, 4, 6]
y = [3.6, 5, 4.2]
yerr = [0.9, 1.2, 0.5]

# plot:
fig, ax = plt.subplots()

ax.errorbar(x, y, yerr, fmt='o', linewidth=2, capsize=6)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



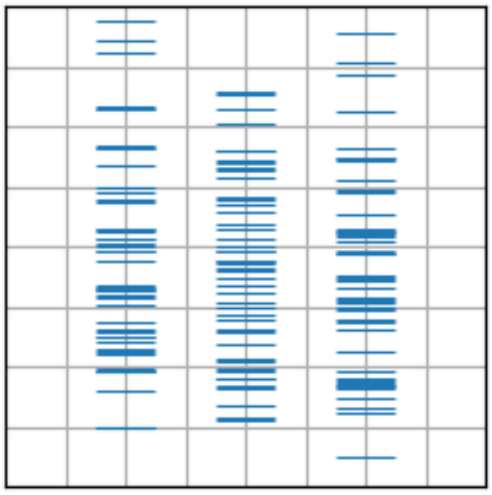
- `violinplot(D)`: Combines box plot and density plot characteristics.

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

plt.style.use('_mpl-gallery')

# make data:
np.random.seed(10)
D = np.random.normal((3, 5, 4), (0.75, 1.00, 0.75), (200, 3))

# plot:
fig, ax = plt.subplots()

vp = ax.violinplot(D, [2, 4, 6], widths=2,
                   showmeans=False, showmedians=False, showextrema=False)
# styling:
for body in vp['bodies']:
    body.set_alpha(0.9)
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



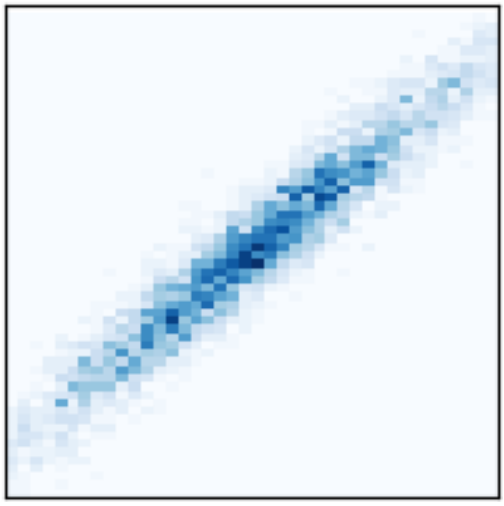
- `eventplot(D)`: Illustrates sequences of events over time.

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

plt.style.use('_mpl-gallery')

# make data:
np.random.seed(1)
x = [2, 4, 6]
D = np.random.gamma(4, size=(3, 50))

# plot:
fig, ax = plt.subplots()

ax.eventplot(D, orientation="vertical", lineoffsets=x, linewidth=0.75)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



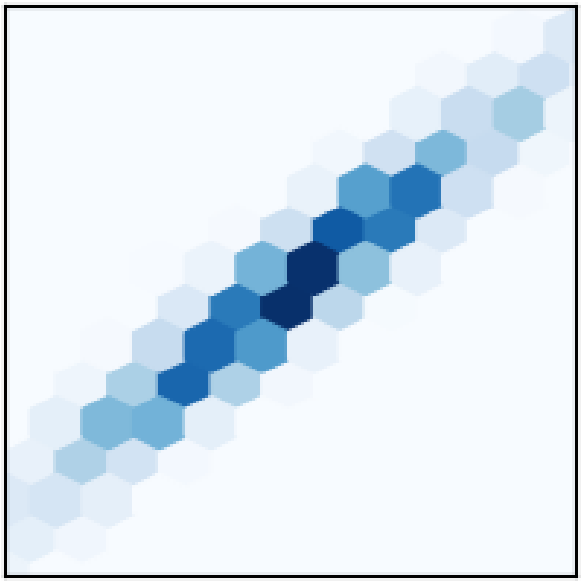
- `hist2d(x, y)`: Creates a 2D histogram.

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

plt.style.use('_mpl-gallery-nogrid')

# make data: correlated + noise
np.random.seed(1)
x = np.random.randn(5000)
y = 1.2 * x + np.random.randn(5000) / 3

# plot:
fig, ax = plt.subplots()

ax.hist2d(x, y, bins=(np.arange(-3, 3, 0.1), np.arange(-3, 3, 0.1)))

ax.set(xlim=(-2, 2), ylim=(-3, 3))

plt.show()
```



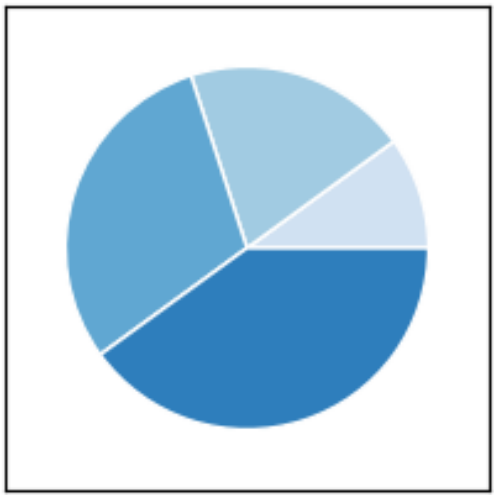
- `hexbin(x, y, c)`: Generates a hexagonal binning plot.

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

plt.style.use('_mpl-gallery-nogrid')

# make data: correlated + noise
np.random.seed(1)
x = np.random.randn(5000)
y = 1.2 * x + np.random.randn(5000) / 3

# plot:
fig, ax = plt.subplots()

ax.hexbin(x, y, gridsize=20)

ax.set(xlim=(-2, 2), ylim=(-3, 3))

plt.show()
```



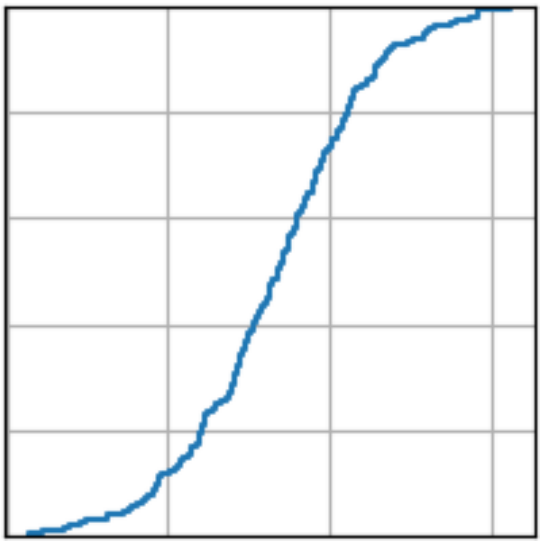
- `pie(x)`: Displays data as slices of a circular pie chart.

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

plt.style.use('_mpl-gallery-nogrid')


# make data
x = [1, 2, 3, 4]
colors = plt.get_cmap('Blues')(np.linspace(0.2, 0.7, len(x)))

# plot
fig, ax = plt.subplots()
ax.pie(x, colors=colors, radius=3, center=(4, 4),
       wedgeprops={"linewidth": 1, "edgecolor": "white"}, frame=True)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



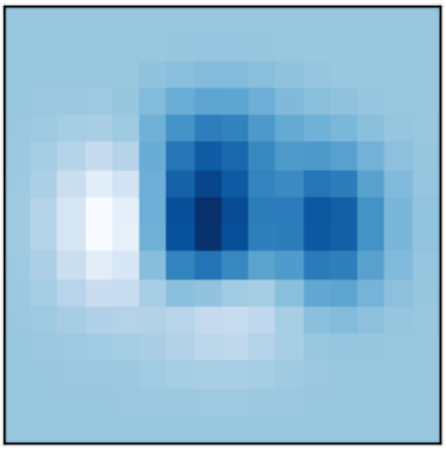
- `ecdf(x)`: Computes and plots the empirical cumulative distribution function.

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

plt.style.use('_mpl-gallery')

# make data
np.random.seed(1)
x = 4 + np.random.normal(0, 1.5, 200)

# plot:
fig, ax = plt.subplots()
ax.ecdf(x)
plt.show()
```



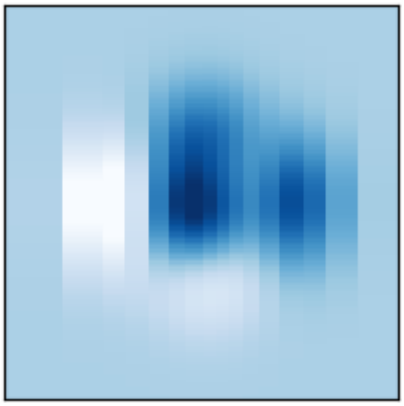
3. **Gridded Data:**
   - `imshow(Z)`: Renders a 2D array as an image.

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

plt.style.use('_mpl-gallery-nogrid')

# make data
X, Y = np.meshgrid(np.linspace(-3, 3, 16), np.linspace(-3, 3, 16))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)

# plot
fig, ax = plt.subplots()

ax.imshow(Z, origin='lower')

plt.show()
```



- `pcolormesh(X, Y, Z)`: Creates a pseudo-color plot with an irregular grid.

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

plt.style.use('_mpl-gallery-nogrid')

# make data with uneven sampling in x
x = [-3, -2, -1.6, -1.2, -.8, -.5, -.2, .1, .3, .5, .8, 1.1, 1.5, 1.9, 2.3, 3]
X, Y = np.meshgrid(x, np.linspace(-3, 3, 128))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)

# plot
fig, ax = plt.subplots()

ax.pcolormesh(X, Y, Z, vmin=-0.5, vmax=1.0)

plt.show()
```

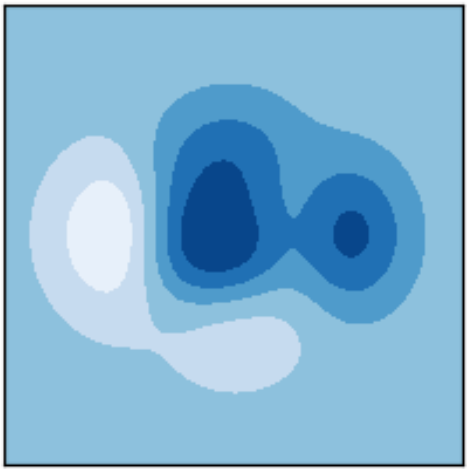


- `contour(X, Y, Z)`: Plots contour lines.

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

plt.style.use('_mpl-gallery-nogrid')

# make data
X, Y = np.meshgrid(np.linspace(-3, 3, 256), np.linspace(-3, 3, 256))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)
levels = np.linspace(np.min(Z), np.max(Z), 7)

# plot
fig, ax = plt.subplots()

ax.contour(X, Y, Z, levels=levels)

plt.show()
```

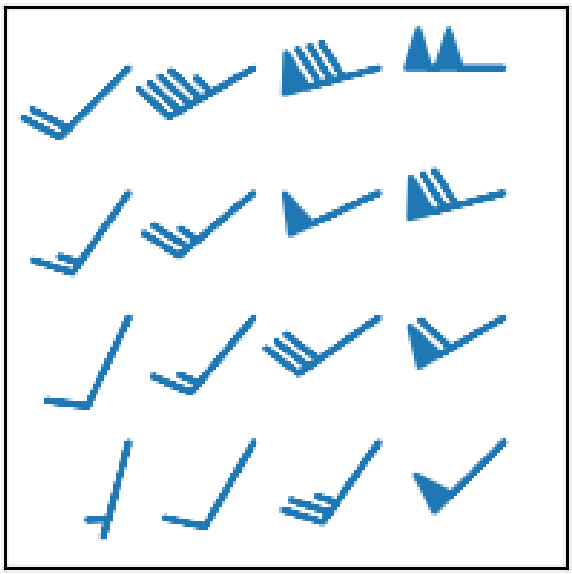
- `contourf(X, Y, Z)`: Generates filled contour plots.

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

plt.style.use('_mpl-gallery-nogrid')

# make data
X, Y = np.meshgrid(np.linspace(-3, 3, 256), np.linspace(-3, 3, 256))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)
levels = np.linspace(Z.min(), Z.max(), 7)

# plot
fig, ax = plt.subplots()

ax.contourf(X, Y, Z, levels=levels)

plt.show()
```



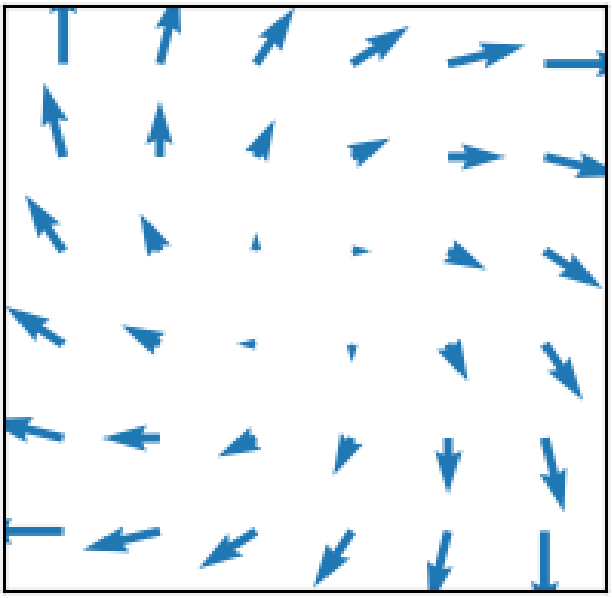
- `barbs(X, Y, U, V)`: Displays wind barb symbols.

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

plt.style.use('_mpl-gallery-nogrid')

# make data:
X, Y = np.meshgrid([1, 2, 3, 4], [1, 2, 3, 4])
angle = np.pi / 180 * np.array([[15., 30, 35, 45],
                                [25., 40, 55, 60],
                                [35., 50, 65, 75],
                                [45., 60, 75, 90]])
amplitude = np.array([[5, 10, 25, 50],
                      [10, 15, 30, 60],
                      [15, 26, 50, 70],
                      [20, 45, 80, 100]])
U = amplitude * np.sin(angle)
V = amplitude * np.cos(angle)

# plot:
fig, ax = plt.subplots()

ax.barbs(X, Y, U, V, barbcolor='C0', flagcolor='C0', length=7, linewidth=1.5)

ax.set(xlim=(0, 4.5), ylim=(0, 4.5))

plt.show()
```



- `quiver(X, Y, U, V)`: Plots a vector field.

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

plt.style.use('_mpl-gallery-nogrid')

# make data
x = np.linspace(-4, 4, 6)
y = np.linspace(-4, 4, 6)
X, Y = np.meshgrid(x, y)
U = X + Y
V = Y - X

# plot
fig, ax = plt.subplots()

ax.quiver(X, Y, U, V, color="C0", angles='xy',
          scale_units='xy', scale=5, width=.015)

ax.set(xlim=(-5, 5), ylim=(-5, 5))

plt.show()
```

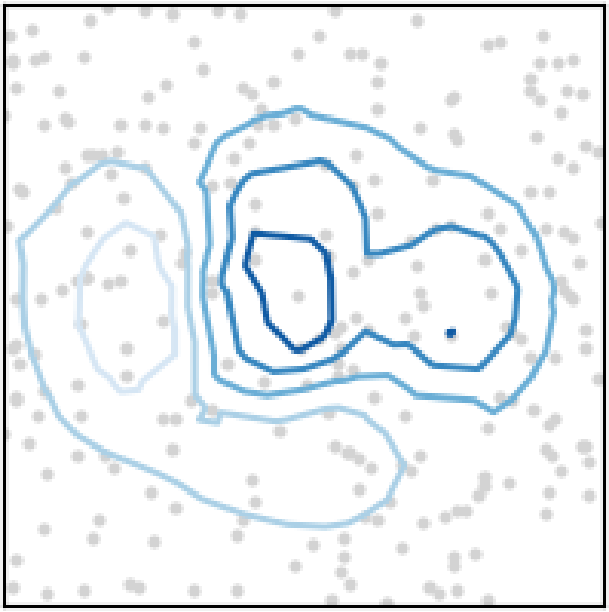


- `streamplot(X, Y, U, V)`: Draws streamlines of vector flows.

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

plt.style.use('_mpl-gallery-nogrid')

# make a stream function:
X, Y = np.meshgrid(np.linspace(-3, 3, 256), np.linspace(-3, 3, 256))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)
# make U and V out of the streamfunction:
V = np.diff(Z[1:, :], axis=1)
U = -np.diff(Z[:, 1:], axis=0)

# plot:
fig, ax = plt.subplots()

ax.streamplot(X[1:, 1:], Y[1:, 1:], U, V)

plt.show()
```

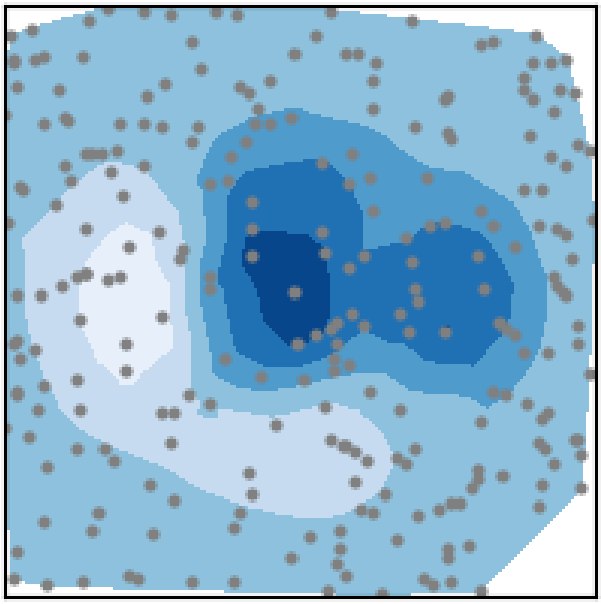
4. **Irregularly Gridded Data:**
   - `tricontour(x, y, z)`: Draws contour lines on unstructured triangular grids.

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

plt.style.use('_mpl-gallery-nogrid')

# make data:
np.random.seed(1)
x = np.random.uniform(-3, 3, 256)
y = np.random.uniform(-3, 3, 256)
z = (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)
levels = np.linspace(z.min(), z.max(), 7)

# plot:
fig, ax = plt.subplots()

ax.plot(x, y, 'o', markersize=2, color='lightgrey')
ax.tricontour(x, y, z, levels=levels)

ax.set(xlim=(-3, 3), ylim=(-3, 3))

plt.show()
```



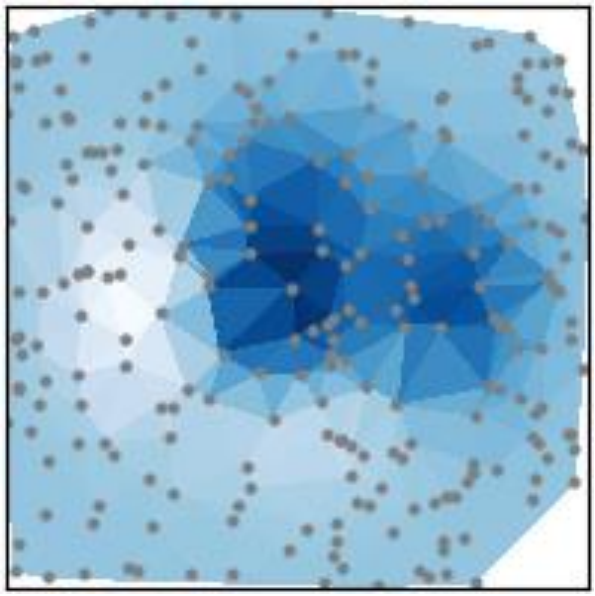
- `tricontourf(x, y, z)`: Creates filled contours on triangular grids.

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

plt.style.use('_mpl-gallery-nogrid')

# make data:
np.random.seed(1)
x = np.random.uniform(-3, 3, 256)
y = np.random.uniform(-3, 3, 256)
z = (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)
levels = np.linspace(z.min(), z.max(), 7)

# plot:
fig, ax = plt.subplots()

ax.plot(x, y, 'o', markersize=2, color='grey')
ax.tricontourf(x, y, z, levels=levels)

ax.set(xlim=(-3, 3), ylim=(-3, 3))

plt.show()
```



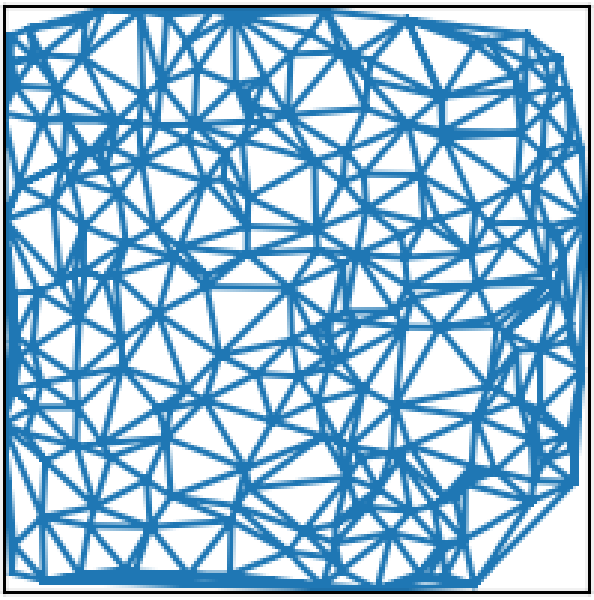
- `tripcolor(x, y, z)`: Displays pseudocolor plots of triangular grids.

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

plt.style.use('_mpl-gallery-nogrid')

# make data:
np.random.seed(1)
x = np.random.uniform(-3, 3, 256)
y = np.random.uniform(-3, 3, 256)
z = (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)

# plot:
fig, ax = plt.subplots()

ax.plot(x, y, 'o', markersize=2, color='grey')
ax.tripcolor(x, y, z)

ax.set(xlim=(-3, 3), ylim=(-3, 3))

plt.show()
```



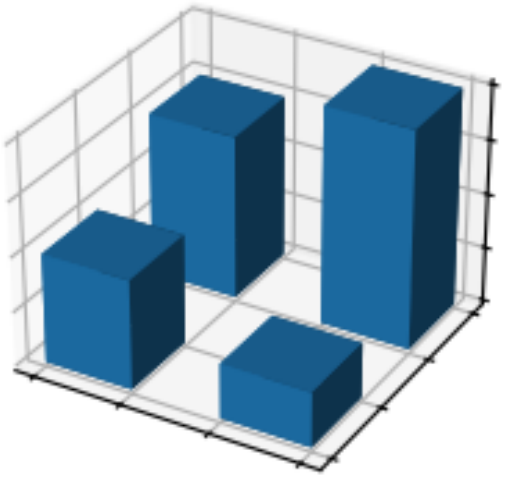
- `triplot(x, y)`: Renders triangular grids with markers and lines.

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

plt.style.use('_mpl-gallery-nogrid')

# make data:
np.random.seed(1)
x = np.random.uniform(-3, 3, 256)
y = np.random.uniform(-3, 3, 256)
z = (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)

# plot:
fig, ax = plt.subplots()

ax.triplot(x, y)

ax.set(xlim=(-3, 3), ylim=(-3, 3))

plt.show()
```



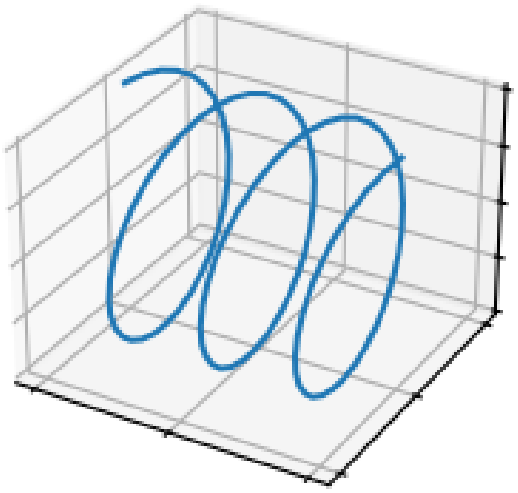
5. **3D and Volumetric Data:**
   - `barplot3d(x, y, z, dx, dy, dz)`: Creates 3D bar plots.

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

plt.style.use('_mpl-gallery')

# Make data
x = [1, 1, 2, 2]
y = [1, 2, 1, 2]
z = [0, 0, 0, 0]
dx = np.ones_like(x)*0.5
dy = np.ones_like(x)*0.5
dz = [2, 3, 1, 4]

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.bar3d(x, y, z, dx, dy, dz)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



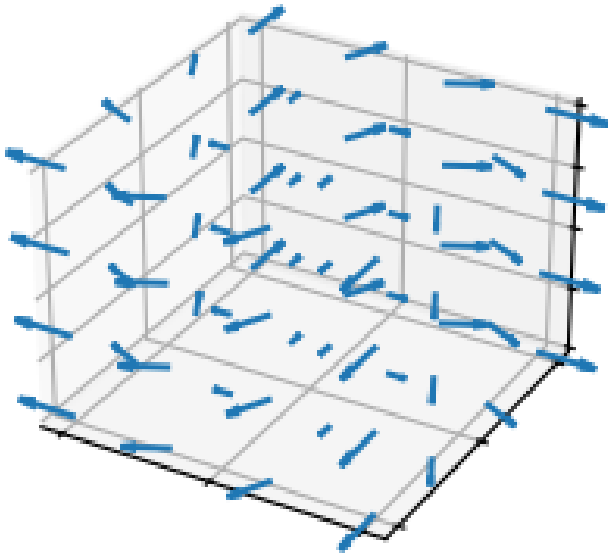
- `plot(xs, ys, zs)`: Produces 3D line plots.

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

plt.style.use('_mpl-gallery')

# Make data
n = 100
xs = np.linspace(0, 1, n)
ys = np.sin(xs * 6 * np.pi)
zs = np.cos(xs * 6 * np.pi)

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot(xs, ys, zs)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



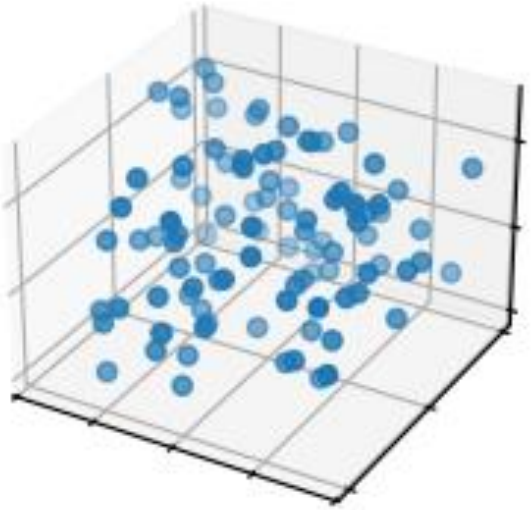
- `quiver(X, Y, Z, U, V, W)`: Plots 3D vector fields.

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

plt.style.use('_mpl-gallery')

# Make data
n = 4
x = np.linspace(-1, 1, n)
y = np.linspace(-1, 1, n)
z = np.linspace(-1, 1, n)
X, Y, Z = np.meshgrid(x, y, z)
U = (X + Y)/5
V = (Y - X)/5
W = Z*0


# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.quiver(X, Y, Z, U, V, W)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



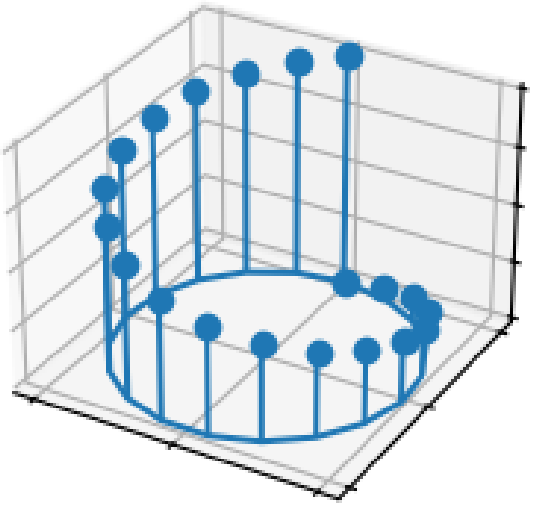
- `scatter(xs, ys, zs)`: Creates 3D scatter plots.

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

plt.style.use('_mpl-gallery')

# Make data
np.random.seed(19680801)
n = 100
rng = np.random.default_rng()
xs = rng.uniform(23, 32, n)
ys = rng.uniform(0, 100, n)
zs = rng.uniform(-50, -25, n)

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.scatter(xs, ys, zs)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



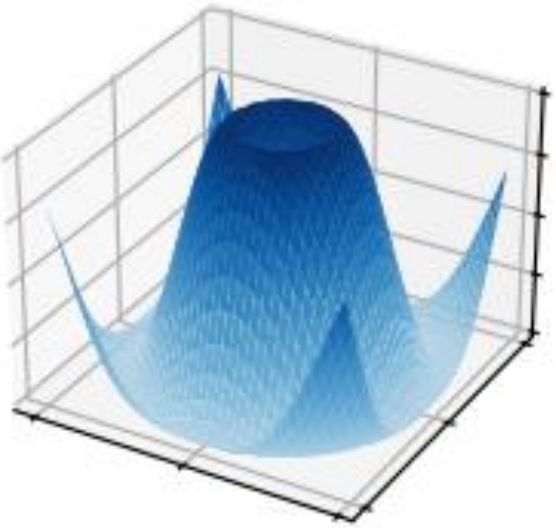
- `stem(x, y, z)`: Plots 3D stem graphs.

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

plt.style.use('_mpl-gallery')

# Make data
n = 20
x = np.sin(np.linspace(0, 2*np.pi, n))
y = np.cos(np.linspace(0, 2*np.pi, n))
z = np.linspace(0, 1, n)

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.stem(x, y, z)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



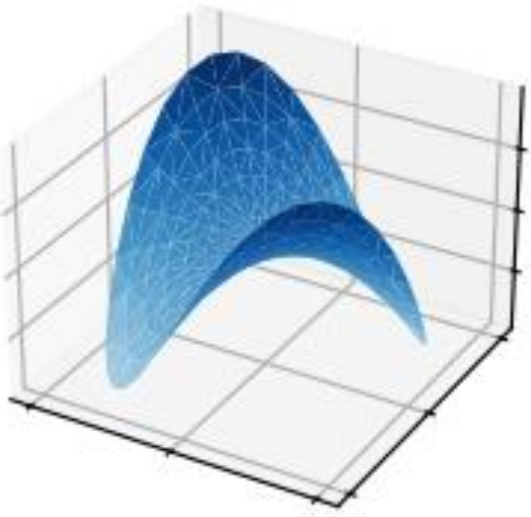
- `plot_surface(X, Y, Z)`: Displays 3D surface plots.

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

from matplotlib import cm

plt.style.use('_mpl-gallery')

# Make data
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

# Plot the surface
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X, Y, Z, vmin=Z.min() * 2, cmap=cm.Blues)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



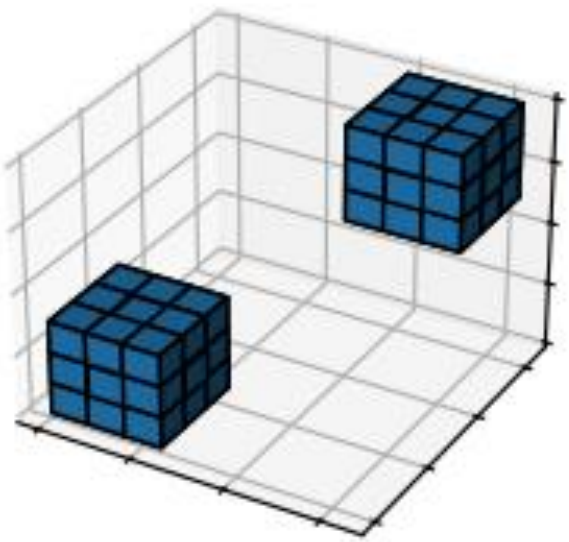
- `plot_trisurf(x, y, z)`: Draws triangulated surfaces.

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

from matplotlib import cm

plt.style.use('_mpl-gallery')

n_radii = 8
n_angles = 36

# Make radii and angles spaces
radii = np.linspace(0.125, 1.0, n_radii)
angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)[..., np.newaxis]

# Convert polar (radii, angles) coords to cartesian (x, y) coords.
x = np.append(0, (radii*np.cos(angles)).flatten())
y = np.append(0, (radii*np.sin(angles)).flatten())
z = np.sin(-x*y)

# Plot
fig, ax = plt.subplots(subplot_kw={'projection': '3d'})
ax.plot_trisurf(x, y, z, vmin=z.min() * 2, cmap=cm.Blues)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



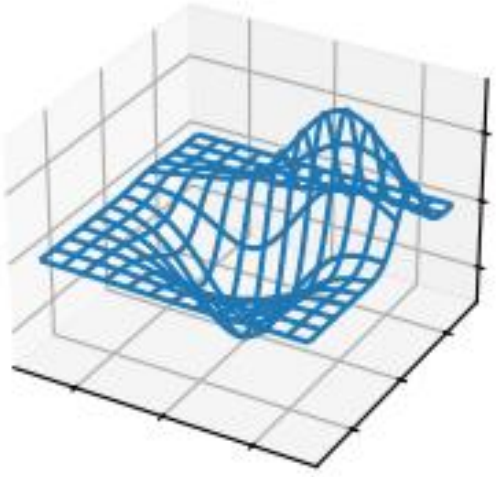
- `voxels([x, y, z], filled)`: Represents 3D pixel data (voxels).

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

plt.style.use('_mpl-gallery')

# Prepare some coordinates
x, y, z = np.indices((8, 8, 8))

# Draw cuboids in the top left and bottom right corners
cube1 = (x < 3) & (y < 3) & (z < 3)
cube2 = (x >= 5) & (y >= 5) & (z >= 5)

# Combine the objects into a single boolean array
voxelarray = cube1 | cube2

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.voxels(voxelarray, edgecolor='k')

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



- `plot_wireframe(X, Y, Z)`: Creates a wireframe of a surface using gridlines.

```
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import axes3d

plt.style.use('_mpl-gallery')

# Make data
X, Y, Z = axes3d.get_test_data(0.05)

# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



Pandas:

Pandas is a robust, open-source Python library designed for data manipulation and analysis. It offers specialized data structures and operations for handling structured data, such as rows and columns, making it an essential tool for data scientists.

Key Features for Data Visualization with Pandas:

1. **Diverse Plotting Options:** Includes line plots, bar plots, histograms, box plots, and scatter plots.
2. **Customizable Visuals:** Titles, labels, and styles can be added to enhance clarity.
3. **Missing Data Handling:** Handles missing values gracefully, ensuring accurate visualizations.
4. **Integration with Matplotlib:** Leverages Matplotlib's functionality for a wide range of plots.

Basic plots in Pandas are created using the built-in `plot()` method. For instance, you can use `df.plot(kind='hist')` to generate a histogram or replace 'hist' with other keywords such as 'box', 'barh', etc.

Common Plot Types in Pandas:

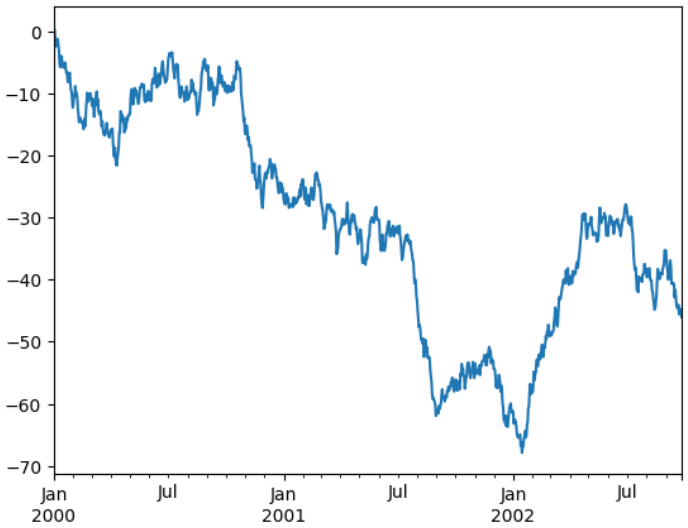
PLOT:

The "plot" method on Series and DataFrame is just a simple wrapper around "plt.

Let us look at a simple line plot in pandas:

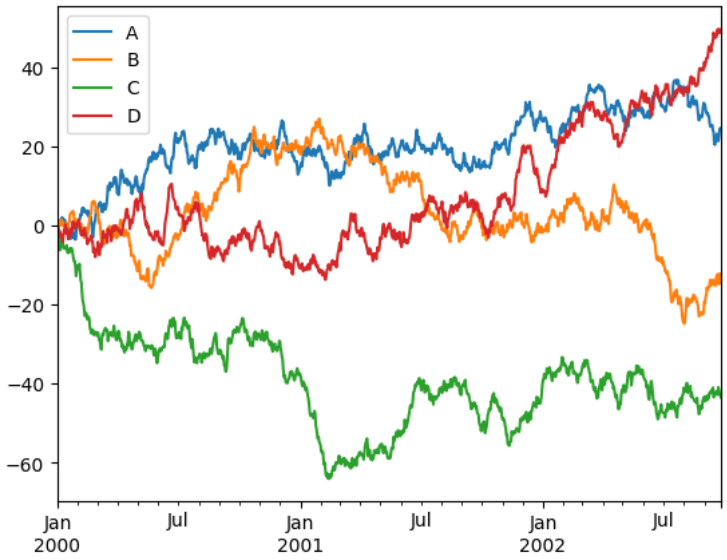
SINGLE LINE:

```
import pandas as pd
import numpy as np
np.random.seed(123456)
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000", periods=1000))
ts = ts.cumsum()
ts.plot();
```



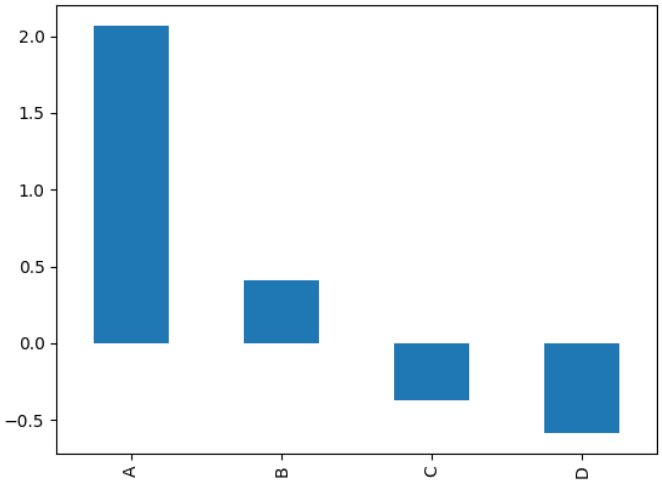
MULTI-LINE:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=list("ABCD"))
df = df.cumsum()
plt.figure();
df.plot();
```



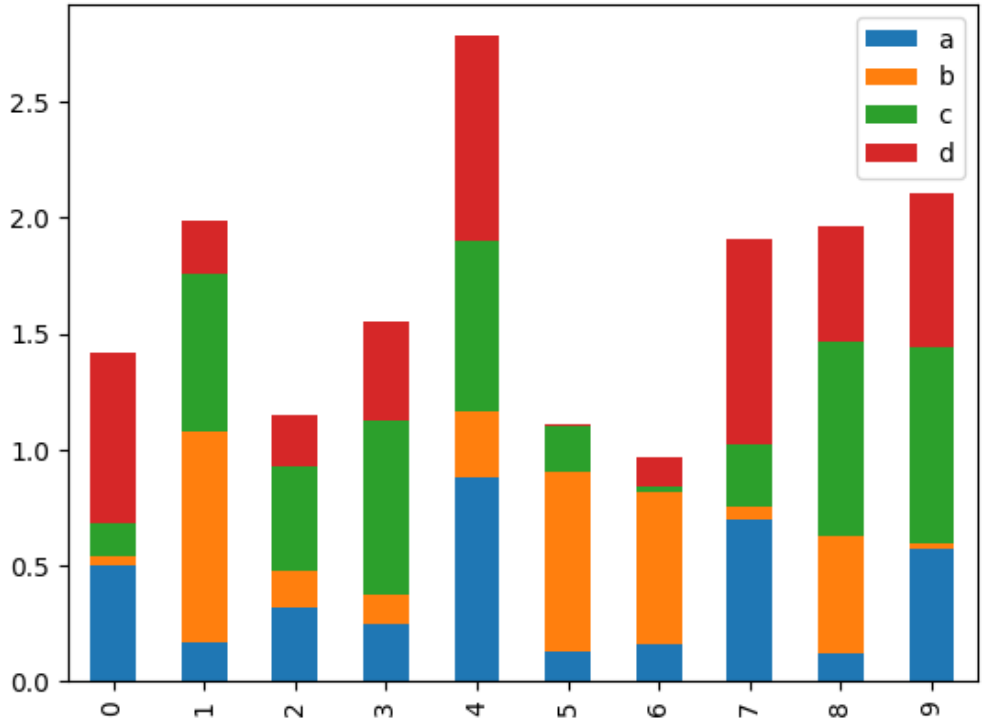
Barplot:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=list("ABCD"))
plt.figure();
df.iloc[5].plot(kind="bar");
```



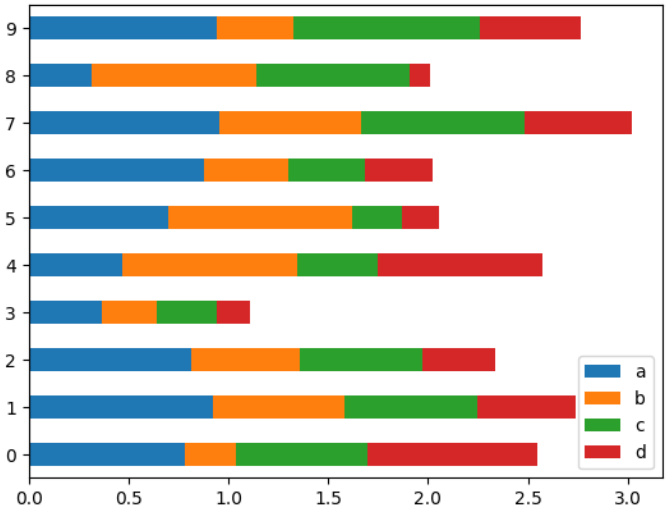
Stacked barplot:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df2 = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c", "d"])
df2.plot.bar(stacked=True);
```



Stacked Horizontal barplot:

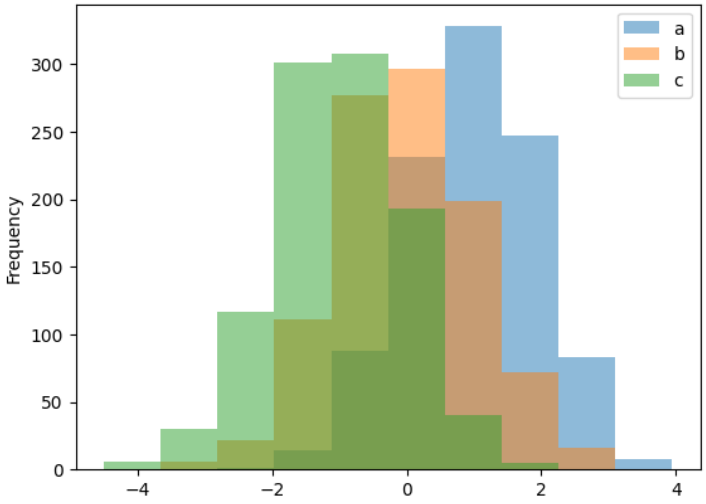
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df2 = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c", "d"])
df2.plot.barh(stacked=True);
```



Histogram:

A histogram is a type of chart that shows the frequency distribution of data points across a continuous range of numerical values. The values are grouped into bin or buckets that are arranged in consecutive order along the horizontal x-axis at the bottom of the chart. Each bin is represented by a vertical bar that sits on the x-axis and extends upward to indicate the number of data points within that bin.

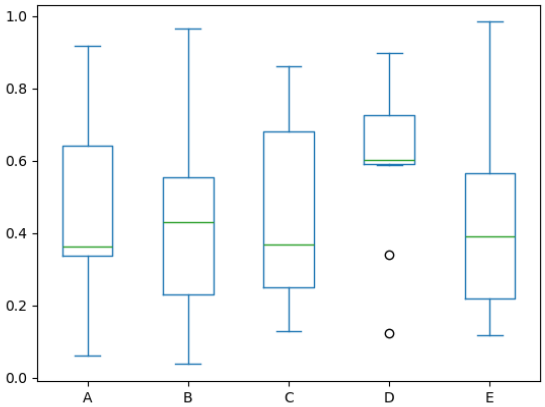
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df4 = pd.DataFrame(
    {
        "a": np.random.randn(1000) + 1,
        "b": np.random.randn(1000),
        "c": np.random.randn(1000) - 1,
    },
    columns=["a", "b", "c"],
)
plt.figure();
df4.plot.hist(alpha=0.5);
```



Box plot:

Box Plot is a graphical method to visualize data distribution for gaining insights and making informed decisions. Box plot is a type of chart that depicts a group of numerical data through their quartiles.

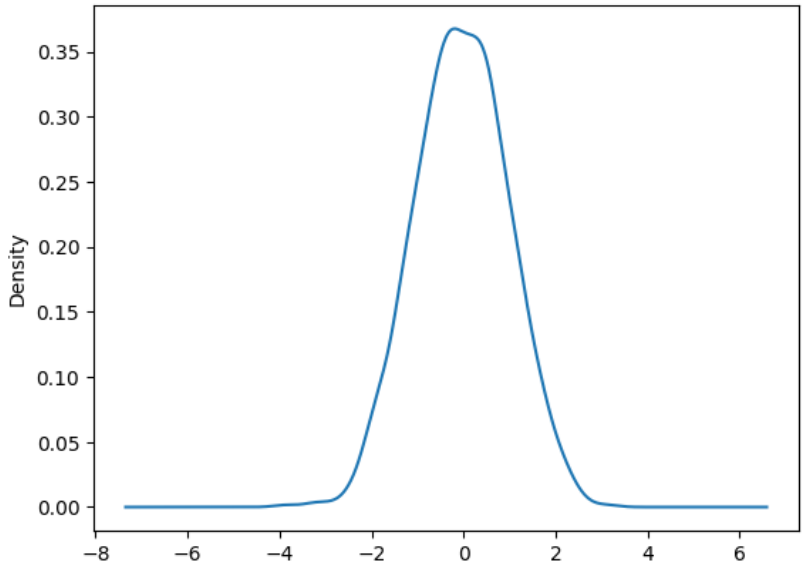
```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D", "E"])
df.plot.box();
```



kde or density plot:

A density plot is a representation of the distribution of a numeric variable. It uses a kernel density estimate to show the probability density function of the variable (see more). It is a smoothed version of the histogram and is used in the same concept.

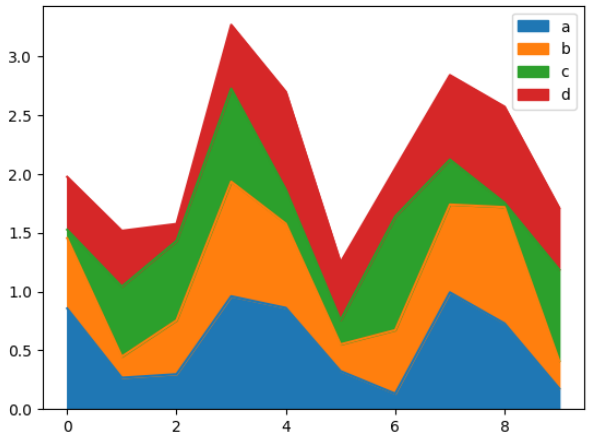
```
import pandas as pd
import numpy as np
ser = pd.Series(np.random.randn(1000))
ser.plot.kde();
```



Area Chart:

An area chart or area graph displays graphically quantitative data. It is based on the line chart. The area between axis and line are commonly emphasized with colors, textures and hatchings. Commonly one compares two or more quantities with an area chart.

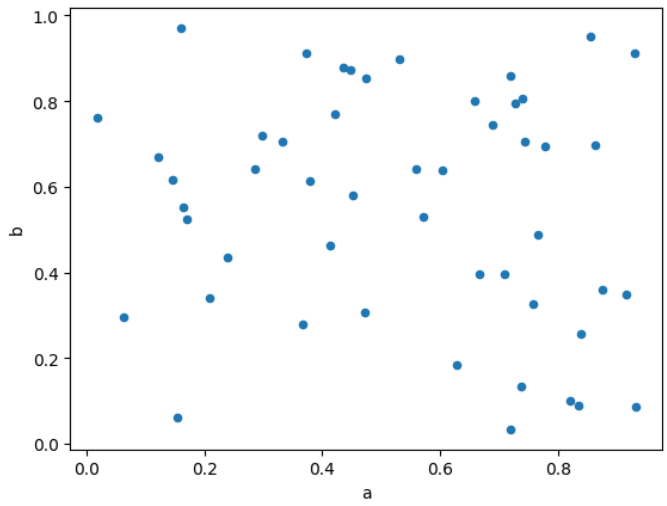
```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c", "d"])
df.plot.area();
```



Scatter Plot:

A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

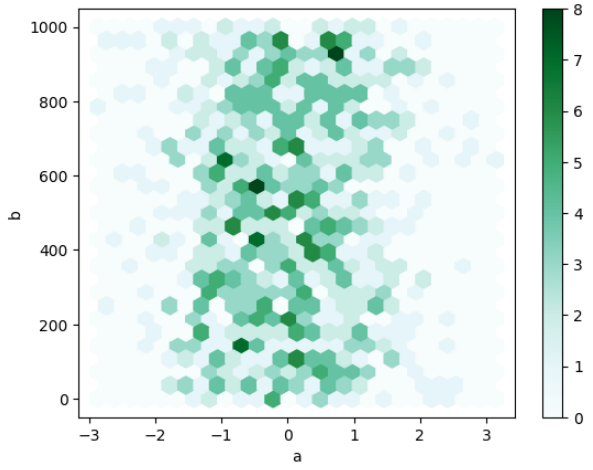
```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
df["species"] = pd.Categorical(
    ["setosa"] * 20 + ["versicolor"] * 20 + ["virginica"] * 10
)
df.plot.scatter(x="a", y="b");
```



Hexagonal Bin Plot:

A hexagonal bin plot is a way to visualize data by grouping points into hexagonal bins and coloring the bins based on the number of points in each bin.

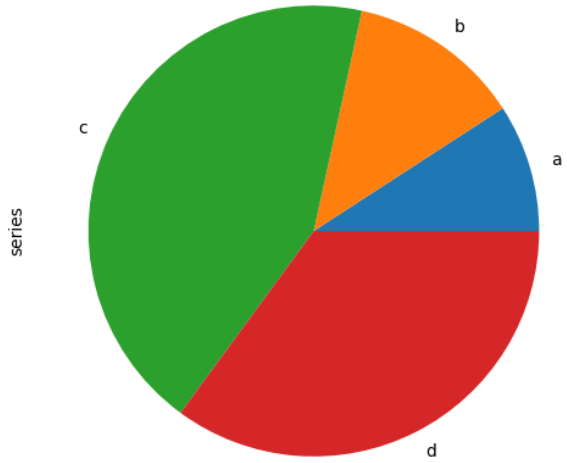
```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
df = pd.DataFrame(np.random.randn(1000, 2), columns=["a", "b"])
df["b"] = df["b"] + np.arange(1000)
df.plot.hexbin(x="a", y="b", gridsize=25);
```



Pie Chart:

A pie chart is a type of graph representing data in a circular form, with each slice of the circle representing a fraction or proportionate part of the whole. All slices of the pie add up to make the whole equaling 100 percent and 360 degrees.

```
import pandas as pd
import numpy as np
series = pd.Series(3 * np.random.rand(4), index=["a", "b", "c", "d"], name="series")
series.plot.pie(figsize=(6, 6));
```



MATPLOTLIB: Advantages and Disadvantages

Pros	Cons
Extensive Customization: Highly customizable, allowing detailed control over visual elements (colors, shapes, fonts, etc.).	**Complex Syntax:** Has a steep learning curve, especially for beginners, due to a complex syntax and many configurations.
Wide Usage & Support: Well-documented, widely used, and supported by a large community with many tutorials available.	**Not as High-Level:** Requires more code for basic plots compared to libraries like Seaborn and Plotly.
Static & Publication-Quality Plots: Produces high-quality, static images suitable for publications and presentations.	**Limited Interactivity:** Lacks built-in interactivity; requires additional libraries (e.g., Plotly) for interactive visualizations.
Versatile & Comprehensive: Supports a wide range of plot types, from basic charts to complex subplots and 3D graphics.	**Performance Limitations:** Can be slow when handling large datasets or complex plots.
Integrates Well with Other Libraries: Easily integrates with other Python libraries like Pandas, NumPy, and SciPy, making it versatile for various data science tasks.	**Outdated Aesthetic:** Default styles may look less modern, although styles can be customized with additional effort.
Cross-Platform Support: Works well across different operating systems and is compatible with Jupyter Notebooks.	**Overlapping Elements:** Requires manual adjustments to prevent overlapping elements, such as tick labels and legends.

PANDAS: Advantages and Disadvantages

Pros	Cons
Easy Data Handling: Simplifies data manipulation with powerful data structures (DataFrames, Series).	**Memory Intensive:** Consumes significant memory, which can limit handling of very large datasets.
Data Cleaning Tools: Offers robust tools for data cleaning, filtering, and aggregation.	**Performance Limitations with Large Data:** Not optimized for very large datasets, impacting speed and efficiency.
Integration with Other Libraries: Works seamlessly with libraries like NumPy, Matplotlib, and SciPy for data analysis and visualization.	**Learning Curve for Complex Operations:** Advanced operations and method chaining can be challenging for beginners.
Rich I/O Support: Supports reading and writing to multiple file formats (CSV, Excel, SQL, JSON, etc.).	**Limited Parallelism:** Lacks optimized multi-threaded processing, which can reduce efficiency for data-heavy tasks.
Built-In Time Series Support: Provides excellent functionality for time-based indexing and analysis.	**Performance Bottlenecks:** Certain operations can be inefficient, especially with row-wise operations.

Comparison Between Pandas and Matplotlib:

Feature	Pandas	Matplotlib
Primary Purpose	Data manipulation and analysis	Data visualization and plotting
Data Structure	Uses DataFrames and Series for data handling	Uses plots and figures for visual representation
Ease of Use	Simple for basic data tasks; complex operations have a steeper learning curve	Moderate learning curve for customization; extensive configuration options
Integration	Integrates well with libraries like NumPy and Matplotlib for visualization	Integrates with Pandas for plotting DataFrames directly
Output	Primarily tabular or text-based data representations	Creates static, publication-quality visuals suitable for presentations

Applications of Pandas:

1. **Data Cleaning:** Identifying and addressing missing or duplicate values.
2. **Data Aggregation:** Grouping and summarizing data.
3. **Time Series Analysis:** Analysing trends and patterns over time.
4. **Data Merging:** Combining datasets from various sources.
5. **Exploratory Data Analysis:** Generating insights from raw data.

Applications of Matplotlib:

1. **Data Visualization:** Creating detailed, high-quality charts.
2. **Trend Analysis:** Visualizing patterns and seasonality.
3. **Statistical Analysis:** Representing data distributions.
4. **Presentation Graphics:** Producing publication-ready visuals.
5. **Geospatial Mapping:** Plotting data points on geographical maps.
