COMP 1023 Introduction to Python Programming

Matplotlib

Dr. Cecia Chan, Prof. SC Cheung, Dr. Alex Lam, Dr. Desmond Tsoi

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

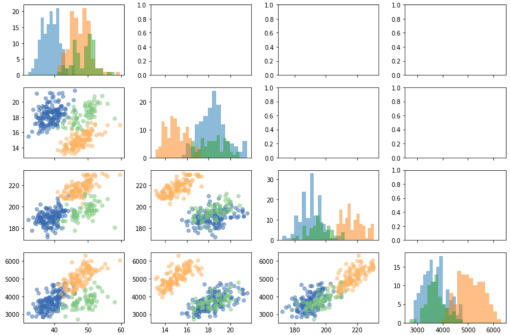
Introduction

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used for data visualization across various fields, including data science, engineering, and research.
- Key Features:
  - Static Visualizations: Generate high-quality static plots, such as line graphs, bar charts, and scatter plots.
  - Animated Visualizations: Create dynamic visualizations that illustrate changes over time or other variables.
  - Interactive Visualizations: Build interactive plots that allow users to explore data through zooming, panning, and real-time updates.

In this course, we will use version 3.10.3, and we will focus on static visualizations.

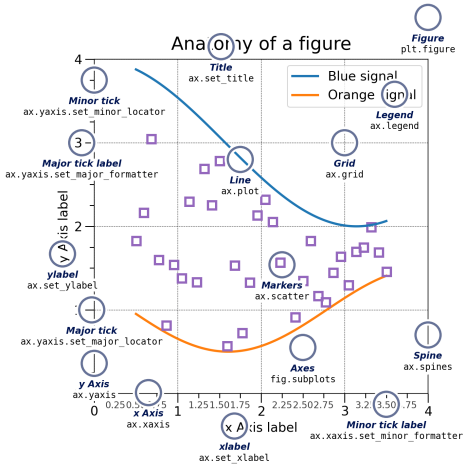
Why Matplotlib?

- Wide Adoption: Matplotlib is one of the most widely used libraries for data visualization in the Python ecosystem.
- Customization: It offers extensive options for customizing visual elements, including colors, labels, and styles.
- Integration: Matplotlib easily integrates with other libraries such as NumPy, Pandas, and Seaborn (a statistical data visualization library) for enhanced data analysis and visualization.



Key Components of a Matplotlib Figure

- A Matplotlib figure is composed of several key elements:
  - Figure: This serves as the primary container for all visual elements, functioning as the canvas for the entire plot.
  - Axes: These are the specific regions within the figure where data visualization occurs; a single figure can incorporate multiple axes.
  - Axis: The axes define the horizontal (x-axis) and vertical (y-axis) dimensions, including their limits, tick marks, and labels essential for data interpretation.
  - Lines and Markers: Lines are utilized to connect data points, illustrating trends, while markers highlight individual data points, particularly in scatter plots.
  - Title and Labels: The title of the plot provides overarching context, while axis labels clarify the data being represented on each respective axis.



Introduction to Matplotlib Pyplot

- Pyplot is a module of Matplotlib designed for creating static, interactive, and animated visualizations in Python.
- To effectively utilize Pyplot, follow these steps:
  1. Import the Module: Begin by importing the module using `import matplotlib.pyplot as plt`.
  2. Prepare Data: Organize your data into lists or arrays for plotting.
  3. Create the Plot: Generate the plot by calling `plt.plot()` with your data.
  4. Enhance the Visualization: Customize the plot by adding titles, axis labels, and other features using `plt.title()`, `plt.xlabel()`, and `plt.ylabel()`.
  5. Display the Plot: Finally, render the plot on the screen with `plt.show()`.

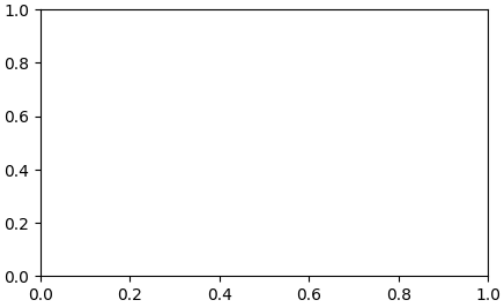
Figures and Axes

- The Figure object contains all the plots.
- You can have multiple plots inside the Figure object.
- You can also save the figure as an image (e.g., JPEG, PNG, etc.)
- You can have one or more Axes inside the figure object, and each Axes object corresponds to a plot.
- Each Axes has an x_axis and an y_axis that represent the data that you want to plot.

Creating a New Figure with a Plot

- To create a Figure, use the `subplots()` function from the Matplotlib library.
- The `subplots()` function returns a new Figure and its Axes object.

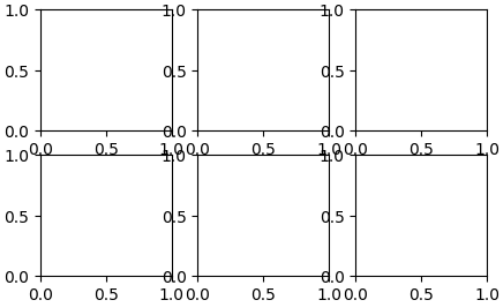
```python
# Filename: single_plot_create.py
import matplotlib.pyplot as plt

# The width and height are 5 and 3 inches
fig, ax = plt.subplots(figsize=(5, 3))
plt.show()
```



Creating a New Figure with Multiple Plots

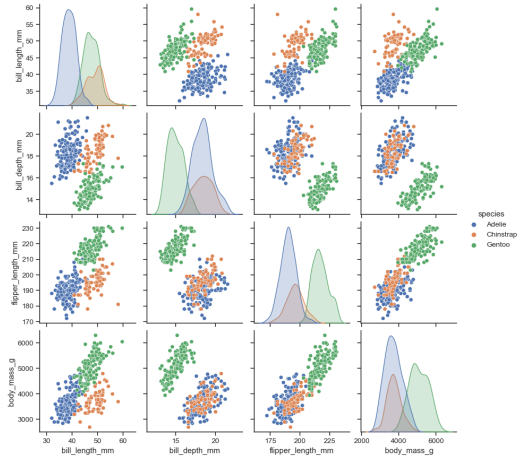
- To create multiple plots, specify the number of rows and columns in the parameters.
- The first two parameters of `plt.subplots()` define the number of plots as rows and columns. It returns the Figure object and the Axes as a NumPy array, allowing you to access each chart using NumPy indexing methods.

```python
# Filename: multiple_plots_create.py
import matplotlib.pyplot as plt

# Multiple plots with 2 rows and 3 columns
fig, ax = plt.subplots(2, 3, figsize=(5, 3))
plt.show()
```



Part I

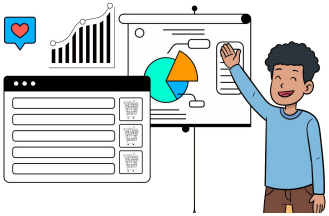
Pairwise Data Visualization



Pairwise Data Visualization

- Visualization techniques for pairwise data include plots of (x,y) coordinates, tabular data (var_0, ..., var_n), and functional relationships of the form $f(x) = y$.
  - `plot(x, y)`: Creates a line plot connecting the data points.
  - `scatter(x, y)`: Generates a scatter plot to display individual data points.
  - `bar(x, height)`: Produces a bar chart representing categorical data.
  - `stem(x, y)`†: Generates a stem plot for discrete data representation.
  - `fill_between(x, y1, y2)`†: Fills the area between two curves.
  - `stackplot(x, y)`†: Creates a stacked area plot to visualize cumulative data.
  - `stairs(values)`†: Generates a step plot for visualizing changes in data.

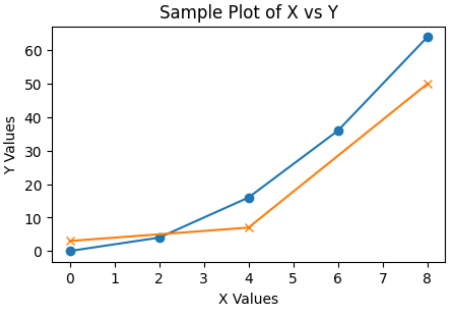
†: Self-explore



Drawing a Line Curve

- To draw a line curve, use the `plot()` method of the Axes object (`ax.plot`). This method takes two lists or NumPy arrays as input: the first list contains the x-coordinates, and the second contains the y-coordinates of the points in the line.
- You can customize the appearance of the plot by specifying the marker type using the marker attribute, and by adding a legend for the data points with the label attribute.
- You can specify the labels (i.e., names) of the x and y axes using `set_xlabel()` and `set_ylabel()`, respectively, and add a title to the plot, use the `set_title()` method.

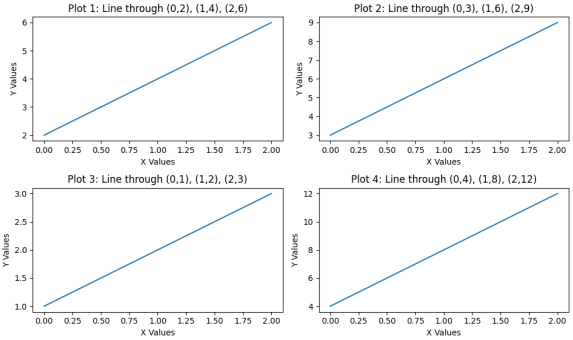
```
import matplotlib.pyplot as plt  # Filename: line_curve_draw.py

fig, ax = plt.subplots(figsize=(5, 3))
# Line through (x,y) -> (0,0), (2,4), (4,16), (6,36), (8, 64)
ax.plot([0, 2, 4, 6, 8], [0, 4, 16, 36, 64],
        marker='o', label="Data Points 1")
# Another line through (x,y) -> (0,3), (4,7), (8,50)
ax.plot([0, 4, 8], [3, 7, 50],
        marker='x', label="Data Points 2")

ax.set_xlabel("X Values")
ax.set_ylabel("Y Values")
ax.set_title("Sample Plot of X vs Y")
plt.show()
```



Drawing a Line Plot in Multiple Rows and Multiple Columns

- Each subplot can represent different datasets.
- x and y axis labels can be added using `set_xlabel()` and `set_ylabel()`.
- Titles for each subplot are set using the `set_title()` method.

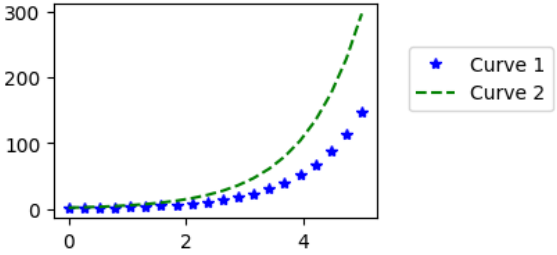
```
import matplotlib.pyplot as plt  # Filename: line_plot_draw_multiples.py

fig, ax = plt.subplots(2, 2, figsize=(10, 6))

ax[0, 0].plot([0, 1, 2], [2, 4, 6])  # First subplot
ax[0, 0].set_xlabel("X Values")
ax[0, 0].set_ylabel("Y Values")
ax[0, 0].set_title("Plot 1: Line through (0,2), (1,4), (2,6)")

ax[0, 1].plot([0, 1, 2], [3, 6, 9])  # Second subplot
ax[0, 1].set_xlabel("X Values")
ax[0, 1].set_ylabel("Y Values")
ax[0, 1].set_title("Plot 2: Line through (0,3), (1,6), (2,9)")

ax[1, 0].plot([0, 1, 2], [1, 2, 3])  # Third subplot
ax[1, 0].set_xlabel("X Values")
ax[1, 0].set_ylabel("Y Values")
ax[1, 0].set_title("Plot 3: Line through (0,1), (1,2), (2,3)")

ax[1, 1].plot([0, 1, 2], [4, 8, 12]) # Fourth subplot
ax[1, 1].set_xlabel("X Values")
ax[1, 1].set_ylabel("Y Values")
ax[1, 1].set_title("Plot 4: Line through (0,4), (1,8), (2,12)")

plt.tight_layout()
plt.show()
```



Drawing a Sequence of Points/Segments

- The `plt.plot()` function also allows you to display a sequence of points or segments that share the same properties (e.g., size, color, width).

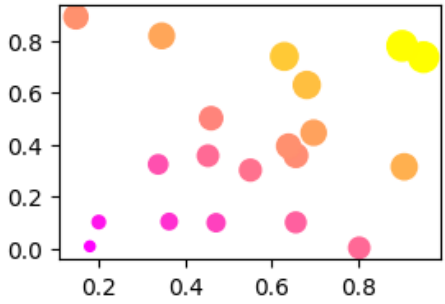
```
# Filename: sequence_points_segments_draw.py
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 5, 20)
y = np.exp(x)

fig, ax = plt.subplots(figsize=(3, 2))

ax.plot(x, y, c="blue", linestyle="",
        marker='*', label="Curve 1")
ax.plot(x, 2*y, c="green",
        linestyle="--", label="Curve 2")

# Specify the legend position with relative position
ax.legend(loc=(1.1, 0.5))
plt.show()
```



Scatter Plot: Displaying a Set of Points With Colormap

- To show a set of points and assign them custom properties (e.g., color, size), use the `ax.scatter()` method.
- You need to specify the lists of x and y coordinates of all your points as parameters (as NumPy arrays).
- You can also specify a colormap using the `cmap` parameter, and the size of the points using the `s` parameter. The size is expressed as the area in square points (dpi).

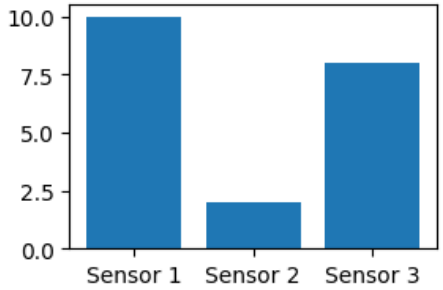
```
import numpy as np  # Filename: scatterplot.py
import matplotlib.pyplot as plt

x = np.random.rand(20)
y = np.random.rand(20)

# Color as a function of the positions of the points
colors = x + y
# Size as a function of the positions of the points
area = 100 * (x + y)
fig, ax = plt.subplots(figsize=(3, 2))

# Specify the colormap as "spring"
ax.scatter(x, y, c=colors, cmap="spring", s=area)
plt.show()
```



Bar Chart: Displaying a Sequence of Numbers as Bars

- To plot vertical or horizontal bars, use the `ax.bar()` method. You can specify the position of each bar on the x-axis as a list, and the height of each bar as a corresponding list (both lists should have the same size).
- You can assign text labels to the ticks on the horizontal axis using the `tick_label` parameter.

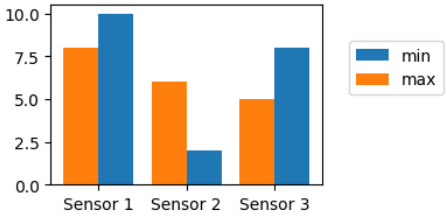
```python
# Filename: barchart_single.py
import numpy as np
import matplotlib.pyplot as plt

# Position of the bars on the x-axis
x = [1, 2, 3]
# The height of each bar
height = [10, 2, 8]

labels = ["Sensor 1", "Sensor 2", "Sensor 3"]
fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x, height, tick_label=labels)
plt.show()
```



Bar Chart: Displaying Multiples Bars

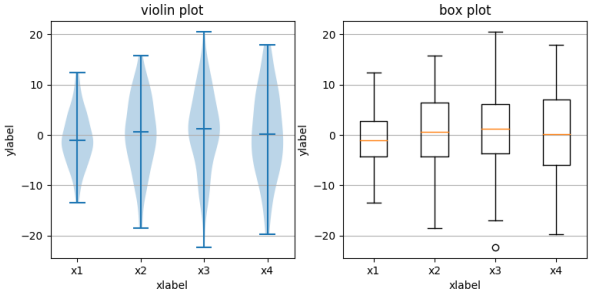
- You can group multiple bars side-by-side.
- Position the two bar plots at $x + \frac{\text{width}}{2}$ and $x - \frac{\text{width}}{2}$.

```python
# Filename: barchart_multiples.py
import numpy as np
import matplotlib.pyplot as plt
heightMin = [10, 2, 8]
heightMax = [8, 6, 5]
x = np.arange(3)
width = 0.4
labels = ["Sensor 1", "Sensor 2", "Sensor 3"]

fig, ax = plt.subplots(figsize=(3, 2))
# Blue bars
ax.bar(x + width / 2, heightMin, width=width, label="Min")
# Orange bars
ax.bar(x - width / 2, heightMax, width=width, label="Max")

ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc=(1.1, 0.5))
plt.show()
```



Part II

Statistical Distributions



Statistical Distributions

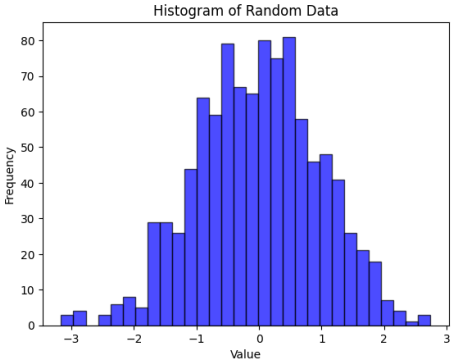
- Visualization techniques for depicting the distribution of one or more variables within a dataset.
- Many of these methods also provide calculations of the distributions.
  - `hist(x)`: Creates a histogram to visualize the frequency distribution of a single variable.
  - `boxplot(X)`: Generates a box plot to summarize the distribution of data through their quartiles.
  - `errorbar(x, y, yerr, xerr)`†: Displays data points with error bars indicating variability.
  - `violinplot(D)` †: Combines a box plot and a density plot to show the distribution of data.
  - `eventplot(D)`†: Visualizes events along an axis, useful for time series data.
  - `hist2d(x, y)`†: Creates a 2D histogram to display the joint distribution of two variables.
  - `hexbin(x, y, C)`†: Generates a hexagonal bin plot for bivariate data visualization.
  - `pie(x)`†: Creates a pie chart to represent proportions of categorical data.
  - `ecdf(x)`†: Computes and plots the empirical cumulative distribution function.

†: Self-explore

Creating a Histogram

- The `hist(x)` function is used to create a histogram, which visualizes the distribution of a dataset. `x` is a one-dimensional array or list containing numerical data.
- You can customize the number of bins to adjust the granularity of the histogram using the `bins` parameter.
- Additional options such as `color`, `alpha`, and `edgecolor` can enhance the visual appeal of the histogram.

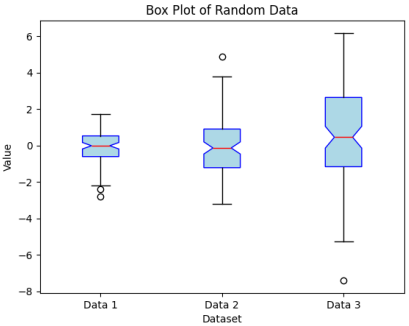
```python
# Filename: histogram.py
import matplotlib.pyplot as plt
import numpy as np

data = np.random.randn(1000) # Generate random data

plt.hist(data, bins=30, color='blue',
         alpha=0.7, edgecolor='black')
plt.title("Histogram of Random Data")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



Creating a Box Plot

- The `boxplot(X)` function creates a box plot, which visualizes the distribution of a dataset through its quartiles. `X` is a one-dimensional array or a two-dimensional array (for multiple box plots).
- Box plots provide a summary of the central tendency, variability, and potential outliers in the data.
- You can customize the appearance of the box plot using parameters such as `color`, `notch`, and `vert` (to control orientation).

```
import matplotlib.pyplot as plt  # Filename: boxplot.py
import numpy as np

data = [np.random.normal(0, std, 100) for std in range(1, 4)]

plt.boxplot(data, notch=True, patch_artist=True,
            boxprops=dict(facecolor='lightblue',
            color='blue'), medianprops=dict(color='red'))
plt.title("Box Plot of Random Data")
plt.xlabel("Dataset"); plt.ylabel("Value")
plt.xticks([1, 2, 3], ['Data 1', 'Data 2', 'Data 3'])
plt.show()
```



Save a Plot to File

- Generated figures can be saved to files in various formats (e.g., JPEG, PNG, PDF, EPS, etc.).
- Use the `fig.savefig()` method to save the figure.

```python
# Filename: save_plot.py
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(3, 2))

ax.plot([0, 1, 2], [2, 4, 6])
ax.plot([0, 1, 2], [3, 6, 9])

fig.savefig("test.png") # or .jpg, .eps, .pdf
```

A Lot More to Explore

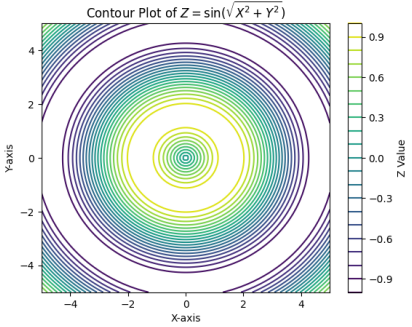
- Gridded Data
  - `imshow(Z)`†
  - `pcolormesh(X, Y, Z)`†
  - `contour(X, Y, Z)`
  - `contourf(X, Y, Z)`†
  - `barbs(X, Y, U, V)`†
  - `quiver(X, Y, U, V)`†
  - `streamplot(X, Y, U, V)`†
- Irregularly Gridded Data
  - `tricontour(x, y, z)`
  - `tricontourf(x, y, z)`†
  - `tripcolor(x, y, z)`†
  - `triplot(x, y)`†
- 3D and Volumetric Data
  - `bar3d(x, y, z, dx, dy, dz)`†
  - `fill_between(x1, y1, z1, x2, y2, z2)`
  - `plot(xs, ys, zs)`†
  - `quiver(X, Y, Z, U, V, W)`†
  - `scatter(xs, ys, zs)`†
  - `streamplot(x, y, u, v)`†
  - `plot_surface(X, Y, Z)`†
  - `plot_trisurf(x, y, z)`†
  - `voxels([x, y, z], filled)`†
  - `plot_wireframe(X, Y, Z)`†

†: Self-explore

Creating Contour Plots with `contour()`

- The `contour()` function is used to create contour plots, which represent 3D data in two dimensions using contour lines. A grid of data points represented by 2D arrays for the x and y coordinates and a corresponding z value.
- You can customize the contour levels using the `levels` parameter to specify the number of contour lines or specific z-values. Additional parameters such as `cmap` allow you to choose color maps for better visualization.

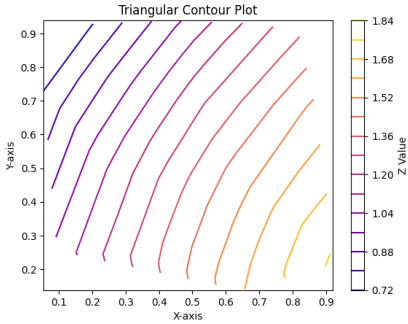
```python
# Filename: contour.py
import numpy as np
import matplotlib.pyplot as plt

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))

plt.contour(X, Y, Z, levels=20, cmap='viridis')
plt.colorbar(label='Z Value')
plt.title("Contour Plot of $Z = \sin(\sqrt{X^2 + Y^2})$")
plt.xlabel("X-axis"); plt.ylabel("Y-axis")
plt.show()
```



Creating Triangular Contour Plots with `tricontour()`

- The `tricontour()` function is used to create contour plots for irregularly spaced data defined by triangles. Three 1D arrays representing the x and y coordinates of the points, and a corresponding z value for each point.
- You can specify the contour levels using the `levels` parameter to control the number of contour lines or specific z-values. The `triangulate` parameter allows you to define the triangulation of the data for better visualization.

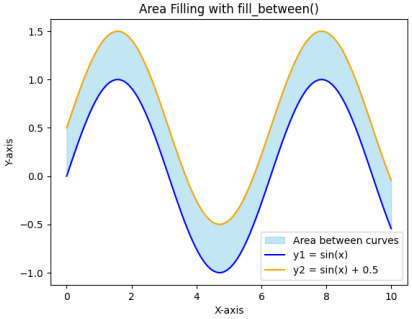
```python
# Filename: tricontour.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.tri import Triangulation

x = np.random.rand(30); y = np.random.rand(30)
z = np.sin(x) + np.cos(y)
triang = Triangulation(x, y) # Create triangulation

plt.tricontour(triang, z, levels=14, cmap='plasma')
plt.colorbar(label='Z Value')
plt.title("Triangular Contour Plot")
plt.xlabel("X-axis"); plt.ylabel("Y-axis")
plt.show()
```



Using `fill_between()` for Area Filling

- The `fill_between()` function is used to fill the area between two horizontal curves, useful for highlighting regions in a plot. x-coordinates and two sets of y-coordinates are defined the boundaries of the filled area.
- You can customize the appearance of the filled area using parameters like `color`, `alpha` (transparency), and `label` for legends.

```python
# Filename: area_filling.py
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y1 = np.sin(x); y2 = np.sin(x) + 0.5

plt.fill_between(x, y1, y2, color='skyblue',
                 alpha=0.5, label='Area between curves')
plt.plot(x, y1, label='y1 = sin(x)', color='blue')
plt.plot(x, y2, label='y2 = sin(x) + 0.5', color='orange')
plt.title("Area Filling with fill_between()")
plt.xlabel("X-axis"); plt.ylabel("Y-axis")
plt.legend()
plt.show()
```



Key Terms

- Animated Visualizations
- Area Filling
- Axes
- Axis
- Bar Chart
- Box Plot
- Color Map
- Contour Plot
- Data Visualization
- Error Bars
- Figure
- fill_between
- Gridded Data
- Histogram
- Interactive Visualizations
- Irregularly Gridded Data
- Matplotlib
- Plot
- Pyplot
- Scatter Plot
- Static Visualizations
- Triangular Contour Plot

Review Questions

Fill in the blanks in each of the following sentences about the Python environment.

1. Matplotlib is a comprehensive library for creating _____, animated, and interactive visualizations in Python.
2. The _____ module is designed for creating static, interactive, and animated visualizations.
3. A _____ serves as the primary container for all visual elements in a plot.
4. The _____ defines the horizontal (x-axis) and vertical (y-axis) dimensions of a plot.
5. The _____ function is used to create a histogram, which visualizes the distribution of a dataset.
6. To create a _____ plot, you can use the `boxplot()` function.

Answer: 1. static plots, 2. Pyplot, 3. figure, 4. axis, 5. hist(), 6. box.

Review Questions

Fill in the blanks in each of the following sentences about the Python environment.

7. The `scatter()` method is used to show a set of _____ on a plot.
8. You can fill the area between two curves using the _____ function.
9. The `contour()` function is used to create _____ plots that represent 3D data in two dimensions.
10. To create multiple plots in one figure, use the `subplots()` function with specified _____ and _____.
11. The _____ parameter in the `plot()` method allows you to customize the appearance of the plot.
12. You can save a figure to a file using the `fig.savefig()` method, which allows saving in various _____.

Answer: 7. points, 8. fill_between, 9. contour, 10. rows; columns, 11. marker, 12. formats.

That's all!

Any question?