Chapter 1: Introduction to Matplotlib

Created	@May 7, 2024 4:58 PM
Class	Introduction to Data Visualization with Matplotlib

Introduction to Data Visualization with Matplotlib

Introduction

1. Importance of data visualization
    - A picture is worth a thousand words
    - Allows deriving insights from data
    - Enables effective communication about data
2. Example: Animated visualization of Ebola outbreak history in West Africa
    - Showcases the power of complex visualizations created with Matplotlib

Matplotlib: A Python Library for Data Visualization

1. Overview of Matplotlib
    - Widely used Python library for visualizing data
    - Provides complete control over plot properties
    - Allows customization and precise control of visualizations
2. Course objectives

- Learn to control visualizations
- Create programs to automatically generate visualizations based on data using object-oriented interface called `pyplot`

Matplotlib's Object-Oriented Interface

1. Using the `pyplot` submodule
   - Import the submodule and conventionally name it `plt`

```
import matplotlib.pyplot as plt
```

2. `plt.subplots()` function
   - Creates a Figure object (container holding everything on the page)
   - Creates an Axes object (canvas for drawing and visualizing data)

Adding Data to the Plot

1. Loading data
   - Example: DataFrame containing weather data for Seattle
     - Columns: "MONTH" (three-letter month names), "monthly average normal temperature" (in Fahrenheit)
2. Plotting data using `Axes` methods
   - `plot()` method to add data to the Axes

```
fig, ax = plt.subplots()
ax.plot(seattle_data["MONTH"], seattle_data["monthly_avera
ge_normal_temperature"])
plt.show()
```

   - Horizontal dimension represents months
   - Vertical dimension represents temperature values
3. Adding multiple datasets

- Example: Adding temperature data for Austin, Texas

```
fig, ax = plt.subplots()
ax.plot(seattle_data["MONTH"], seattle_data["monthly_avera
ge_normal_temperature"])
ax.plot(austin_data["MONTH"], austin_data["monthly_average
_normal_temperature"])
plt.show()
```

Customizing Plots in Matplotlib

Customizing Data Appearance

1. Indicating discrete data points
    - Use `marker` argument in `plot()` method to add markers
        - Example: `marker="o"` for circles, `marker="v"` for downward triangles
    - Markers show where data exists, lines connect data points
    - Matplotlib marker styles documentation
2. Changing line styles
    - Use `linestyle` argument in `plot()` method
        - Example: `linestyle="--"` for dashed lines
    - Matplotlib line styles documentation
    - Can remove lines entirely by passing `linestyle="None"`
3. Changing data color
    - Use color abbreviations or names as arguments
        - Example: `color="r"` for red

Axis Labels

1. Importance of labeling axes

- Necessary for effective communication through visualizations
- Often neglected but crucial

2. Setting axis labels
   - Use `set_xlabel()` and `set_ylabel()` methods of Axes object

```
ax.set_xlabel("Month")
ax.set_ylabel("Average Temperature (Fahrenheit)")
```

   - Capitalize axis labels like sentences (first word and proper nouns)

Plot Title

1. Adding a title to the plot
   - Use `set_title()` method of Axes object

```
ax.set_title("Average Monthly Temperatures in Seattle")
```

   - Provides context and additional information about the data

Using Small Multiples in Matplotlib

Avoiding Cluttered Plots

1. Adding too much data can make plots busy and obscure patterns
   - Example: Plotting average precipitation, 25th percentile, and 75th percentile for Seattle and Austin
2. Solution: Using small multiples
   - Multiple small plots showing similar data across different conditions
   - Facilitates direct comparison between datasets

Creating Subplots in Matplotlib

1. Subplots as small multiples

- Function `plt.subplots()` creates subplots

2. One subplot (previous usage)

```
fig, ax = plt.subplots()
```

3. Multiple subplots in a grid

```
fig, ax = plt.subplots(nrows=3, ncols=2)
```

- `ax` is now an array of Axes objects (3 rows, 2 columns)
- Access individual Axes using indexing: `ax[row, col]`

4. One-dimensional case
   - If `nrows` or `ncols` is 1, `ax` is a 1D array
   - Access individual Axes using a single index: `ax[index]`

Example: Precipitation Data for Seattle and Austin

1. Creating subplots

```
fig, ax = plt.subplots(nrows=2, ncols=1, sharey=True)
```

- `sharey=True` ensures same y-axis range across subplots

2. Plotting data in subplots

```
ax[0].plot(seattle_data["MONTH"], seattle_data["precipitat
ion"])
ax[1].plot(austin_data["MONTH"], austin_data["precipitatio
n"])
```

3. Labeling axes

```
ax[0].set_ylabel("Precipitation (inches)")
ax[1].set_ylabel("Precipitation (inches)")
```

```
ax[1].set_xlabel("Month")
```

4. Displaying the figure

```
plt.show()
```