✏️

Chapter 4: Sharing visualizations with others

Created	@May 7, 2024 5:00 PM
Class	Introduction to Data Visualization with Matplotlib

Creating Visualizations for Sharing and Automation

1. Customizing Figure Styles

- Changing the overall style of a figure
  - Using styles like `ggplot` from R
- Effects of style changes
  - Changes multiple elements (colors, fonts, background grid, etc.)
- Applying styles to all figures in a session
- Reverting to the default style
  - Code example:

    ```
    plt.style.use("default")
    ```

2. Available Styles in Matplotlib

- Matplotlib provides several predefined styles
  - Examples: `"bmh"`, `"seaborn-colorblind"`, styles from Seaborn library

3. Choosing an Appropriate Style

- Consider the audience and communication goals
- Avoid dark backgrounds for better visibility
- Use colorblind-friendly styles
  - Examples: `"seaborn-colorblind"`, `"tableau-colorblind10"`
  - Approximately 1 out of 20 individuals is colorblind
- Considerations for web vs. printed figures
  - For printed figures, use less ink and avoid colored backgrounds
  - Use `"grayscale"` style for black-and-white printing

Key Details:

- The chapter focuses on creating visualizations that can be shared with others and incorporated into automated data analysis pipelines.
- It covers customizing figure styles in Matplotlib, available styles, and considerations for choosing an appropriate style based on the audience and communication goals.
- Styles affect multiple elements of the figure, such as colors, fonts, and background grid.
- Matplotlib provides several predefined styles, including those inspired by other libraries like Seaborn.
- Colorblind-friendly styles should be considered to ensure color differences are visible to all viewers.
- Styles can be chosen based on whether the figure is intended for web or print, considering factors like ink usage and visibility.
- The `"grayscale"` style is recommended for black-and-white printing.

Code Examples:

```
# Applying the 'ggplot' style
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
# Creating a figure# ... (plotting code) ...# Reverting to the default style
plt.style.use('default')
```

Saving Visualizations for Sharing

1. Saving Figures as Files

- Replacing `plt.show()` with `savefig()`
  - Saves the figure as a file instead of displaying it
  - Provide a filename as input
  - Code example:

    ```
    fig.savefig("gold_medals.png")
    ```

2. File Formats

- PNG (Portable Network Graphics)
  - Lossless compression, high quality
  - Larger file size
- JPG (Joint Photographic Experts Group)
  - Lossy compression, smaller file size
  - Control quality with `quality` parameter (1-100)
  - Don’t set `qualtiy` valur over 95 as the compression will no longer be effective
  - Code example:

```
fig.savefig("image.jpg", quality=80)
```

- SVG (Scalable Vector Graphics)
  - Vector graphics format
  - Allows editing individual elements
  - Good for post-production editing
  - Code example:

    ```
    fig.savefig("image.svg")
    ```

3. Image Quality

- DPI (Dots Per Inch)
  - Higher DPI means higher resolution
  - Larger file size for higher DPI
  - Code example:

    ```
    fig.savefig("image.png", dpi=300)
    ```

4. Figure Size

- Control figure size with `set_size_inches()`
  - Specify width and height in inches
  - Determines aspect ratio
    - Code example:

    ```
    fig.set_size_inches([8, 6])
    ```

Key Details:

- The chapter focuses on saving visualizations as files for sharing and further processing.
- Figures can be saved using `savefig()` instead of `plt.show()`, providing a filename and format.
- Common file formats include PNG (lossless compression), JPG (lossy compression), and SVG (vector graphics).
- Image quality can be controlled using parameters like `quality` for JPG and `dpi` for resolution.
- Figure size and aspect ratio can be set using `set_size_inches()`.

Automating Figure Creation Based on Data

1. Flexibility of Matplotlib

- Matplotlib can adapt to the inputs provided
- Allows writing functions and programs that adjust behavior based on input data

2. Benefits of Automation

- Makes it easier to do more
- Allows faster execution
- Provides flexibility and robustness
- Ensures reproducible behavior across different runs

3. Example: Visualizing Height of Athletes Across Sports

- Input data: DataFrame with "Sport" column and athlete heights
- Identify unique sports using `unique` method on "Sport" column

- Loop over unique sports
    - In each iteration:
        - Select rows for the current sport
        - Create a bar for the sport
            - Height: Mean of "Height" column
            - Error bar: Standard deviation of "Height" column
- Set y-label and x-tick labels based on sports
- Rotate x-tick labels for better visibility

4. Advantages of Automated Visualization

- No need to know the number of sports in advance
- Code automatically adjusts the number of bars based on input data
- Provides flexibility and adaptability to different datasets

Key Details

- Matplotlib's flexibility allows writing programs that adjust behavior based on input data.
- Automation offers benefits like ease of use, speed, flexibility, robustness, and reproducibility.
- The example demonstrates visualizing athlete heights across sports by looping over unique sports in the data.
- The code automatically adjusts the number of bars based on the input data, without needing to know the number of sports in advance.
- Automated visualization provides flexibility and adaptability to different datasets.