Design for an Audience: Takeaways

Syntax

- Creating a bar plot using matplotlib's OO approach:

```
fig, ax = plt.subplots(figsize)
ax.bar(x, height, width, color)
plt.show()
```

- Removing all the four spines of a plot named ax:

```
for location in ['left', 'right', 'bottom', 'top']:
    ax.spines[location].set_visible(False)
```

- Hiding and modifying ticks:

```
ax.tick_params(top=False, left=False)
ax.tick_params(axis='x', colors='grey')
```

- Moving the x-ticks on top:

```
ax.xaxis.tick_top()
```

- Configuring ticks and tick labels:

```
ax.set_xticks([0, 150000, 300000])
ax.set_xticklabels(['0', '150,000', '300,000'])
```

- Adding text on a plot:

```
ax.text(x, y, s, ymin, color, size, weight)
```

- Adding a vertical line on a plot:

```
ax.axvline(x, ymin, c, alpha)
```

Concepts

- Depending on the graph's audience, we have two kinds of data visualization:
    - Exploratory data visualization: we create graphs for ourselves to better understand and explore data
    - Explanatory data visualization: we create graphs for others to inform, make a point, or tell a story
- Design principles help us in two ways:
    - They generate design options
    - They help us choose among those options
- The familiarity principle says that we should choose what our audience is most familiar with.
- The device your readers are using to see your graph is important — if you expect them to use a mobile device, then your graph should have mobile-friendly proportions.
- If we know that a large part of our audience will read the article on a mobile. This means that our graph needs to have mobile-friendly proportions.

- Generally, a graph has three elements:
    - Data elements: the numbers and categories represented and the relationships between them.
    - Structural elements: the axes, the ticks, the legend, the grid, etc.
    - Decorations: extra colors, shapes, artistic drawings etc.
- From the total amount of ink used for printing a graph, some of the ink goes to show the data — that is the data-ink.
- Maximizing the data-ink ratio enables us to build graphs where readers focus more on the data. To maximize data-ink, we can do the following:
    - Erase non-data ink
    - Erase redundant data-ink
- Be mindful of people's reading direction. When it comes to a graph, people usually start looking from top left and follow a zigzag pattern until they reach bottom right.
- Generally, the title must be data ink. If we need to give structural explanations in text, we can use the subtitle. That's because the title is always so noticeable, and we need to leverage that to show more data (and also maximize the data-ink ratio).

Resources

- The Visual Display of Quantitative Information — Edward Tufte
- The Lifecycle of a Plot
- Matplotlib Gallery

Storytelling Data Visualization: Takeaways

Syntax

- Creating a grid chart of four plots:

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4, ncols=1)
```

- Modifying all Axes objects using a for loop:

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4, ncols=1)
axes = [ax1, ax2, ax3, ax4]
for ax in axes:
    ax.plot(x_coordinates, y_coordinates)
    ax.set_yticklabels([])
plt.show()
```

- Adding a horizontal line:

```
ax.axhline(y)
```

- Adding a horizontal line with various parameters:

```
ax1.axhline(y=1600, xmin=0.5, xmax=0.8,
            linewidth=6, color='#af0b1e', alpha=0.1)
```

Concepts

- In a broad sense, a story is a sequence of events: something happens, then something else happens, and so on. A graph that only shows numerical facts isn't a story.
- Another story element is change: something or someone changes throughout the story. A static graph that doesn't show any element of change isn't a story.
- To create a data story, we need to wrap numerical facts into events that show change.
- Matplotlib can be very powerful if you're a little imaginative. You'll often find yourself wanting to do X and search for a specific function to do X. This approach won't always work because the function you want may not exist.
- However, you can often combine what Matplotlib already has to get what you want. To do that, identify the basic parts of what you want to create. Then, try to create the basic parts using Matplotlib.

Resources

- The Visual Display of Quantitative Information — Edward Tufte
- Matplotlib Gallery
- Is Your Data Story Actually a Story — Joshua Smith

Gestalt Principles and Pre-Attentive Attributes: Takeaways

Concepts

- The overarching idea behind Gestalt principles is that humans generally perceive patterns rather than individual objects. From a practical point of view, Gestalt principles tell us what sort of pattern we can expect people to see when we show them our data visualizations.
- When we see distinct objects close to each other, we perceive them as a group — this is the principle of proximity.
- When we see distinct objects that are similar to one another, we perceive them as a group — this the principle of similarity.
- Similarity can apply to color, shape, size, or other visual qualities.
- When we see distinct elements enclosed inside a visual form, we perceive them as part of the same group — this is the principle of enclosure.
- When we see distinct objects connected by some kind of a visual form (usually a line), we perceive them as part of the same group — this the principle of connection.
- Some of the Gestalt principles are stronger than others, and they create a visual hierarchy. We need to create data visualization with visual hierarchy in mind — if connection cancels out similarity without intention, we can communicate incorrect information.
- Connection and enclosure typically have similar strengths, and they are both stronger than proximity and similarity.
- Thicker lines and stronger color can mean a stronger connection and a weaker enclosure. Dotted lines along with a strong-colored enclosing form can mean stronger enclosure and weaker connection.
- Similarity can be stronger than proximity in some cases, and vice-versa.
- A visual object that is different from the rest stands out and signals where to look. We can use this visual effect to guide our audience's attention. If people look where we want them to, we can more easily deliver our information.
- Our brain typically becomes aware of different objects before we consciously direct our attention toward them. Because they come before conscious attention, we call them pre-attentive.
- Pre-attentive attributes can take many forms: color, size, shape, enclosure, width, length, etc.
- Pre-attentive attributes can become inefficient if we overuse them, and although pre-attentive attributes can be useful, they aren't essential. If you don't need one for your graph, then don't add it.

Resources

- Pre-attentive processing
- Gestalt psychology

Matplotlib Styles: FiveThirtyEight Case Study: Takeaways

Syntax

- Using a specific Matplotlib style:

```
import matplotlib.style as style
style.use('fivethirtyeight')
plt.plot([1, 2, 3], [5, 2, 7])
plt.show()
```

- Checking Matplotlib's styles:

```
style.available
```

- Moving the x-coordinate of the left sides of the bars by using the left parameter:

```
ax.barh(left)
```

- Adding a signature bar with name and data source:

```
ax.text(x, y, 'Creator' + ' ' * 90 + 'Source',
        color,
        backgroundcolor)
```

Concepts

- Matplotlib's pre-defined styles change the default visual properties of graphs.
- We must call the `style.use()` function before we create the graph.
- Once we use a certain style, all subsequent graphs will inherit that style.
- To return to the default settings, use `style.use('default')` .
- If you want to switch between different styles, use style.use('default') between each change — some of the styles can interfere with one another.

Resources

- Blog Post: FiveThirtyEight Graphs Tutorial