Data Visualization: Beyond Matplotlib

```
$ echo "Data Sciences Institute"
```

Overview of this slide deck, we will:

- Learn about other packages to use for data visualization in Python, such as:
  - Seaborn
  - Plotly (for interactive viz)
  - Wordclouds and Venn Diagrams
- Discuss when to use (and when not to use) interactive data visualization

Seaborn

What is seaborn?

- A data viz package built on top of matplotlib
- REALLY SIMILAR to matplotlib (phew!) but takes care of some **semantic mapping** for us
- Recall: mapping is deciding which aspects of our dataset correspond to which visual elements on our plots

Install and load packages

- After installing seaborn, we can load all the packages we'll be using in this lesson

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy
import PIL
import requests
```

Load data

- We're going to work with the 'tips' sample dataset, which is available as part of the seaborn (sns) package

```
tips = sns.load_dataset("tips")
print(tips)
```

- This dataset contains information about tips at a restaurant, including meal time, party size, gender of customer, total bill amount, and tip amount

Basic plot

- Let's make a simple line plot of tip amount vs total bill

```
sns.lineplot(data=tips,       # choose our dataset
             x='total_bill',  # define our x variable
             y='tip')         # define our y variable
```

Use premade styles

- We can use premade styles to alter the appearance of our plot

```
sns.set_style('whitegrid')
```

- Try other preset styles like ‘darkgrid’, ‘whitegrid’, ‘dark’, ‘white’, ‘ticks’, then make our line plot again – how has it changed?

Add title and axis labels

- Seaborn can do many of the same things as matplotlib, like adding labels We do this using the **set.** function

```
tipgraph = sns.lineplot(data=tips,
                        x='total_bill',
                        y='tip')

tipgraph.set(title='Tips vs. Total Bill',
             xlabel='Total Bill ($)',
             ylabel='Tip Amount ($)')
```

Modify aesthetic elements

- Like in base matplotlib, we can use the subplots function to change figure size

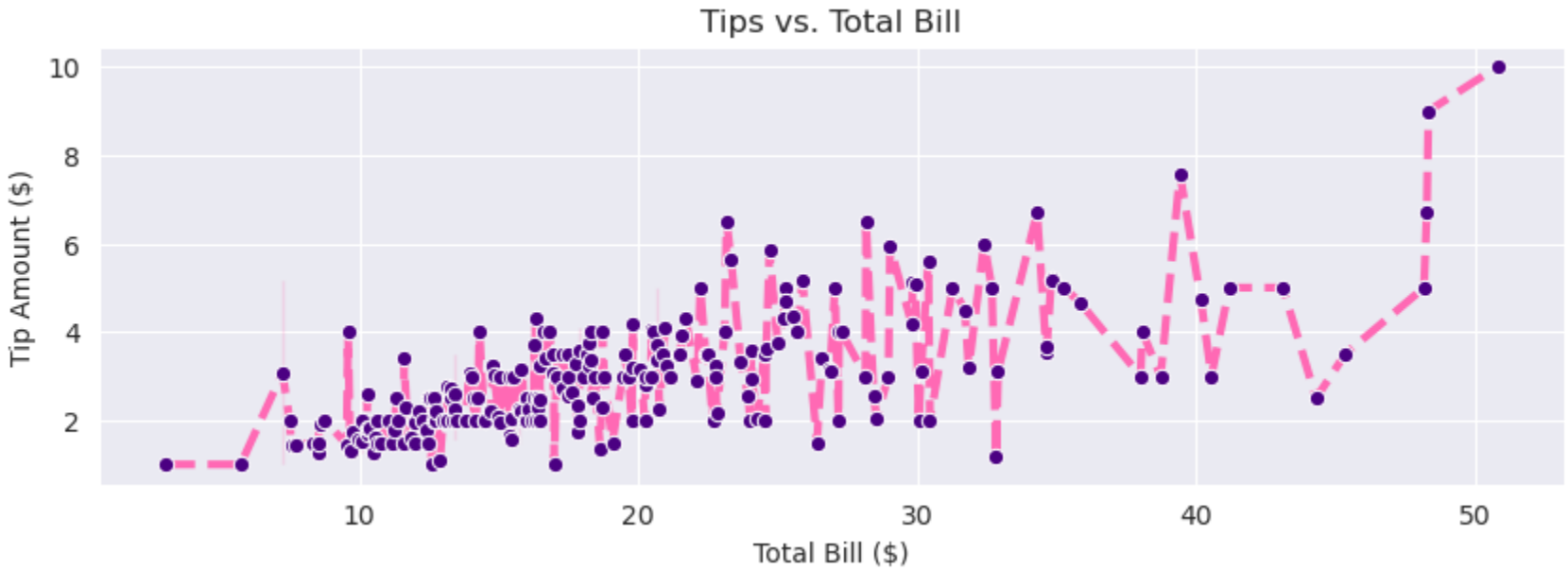
```
fig = plt.subplots(figsize=(10, 3))
```

Modify aesthetic elements

- We can also change colour, marker style, and line style

```
tipgraph = sns.lineplot(data=tips,
                        x='total_bill',
                        y='tip',
                        color = 'hotpink',
                        linestyle = '--',
                        linewidth = 3,
                        marker = 'o',
                        markerfacecolor = 'indigo')
```

Modify aesthetic elements



BUT why is seaborn actually an improvement?

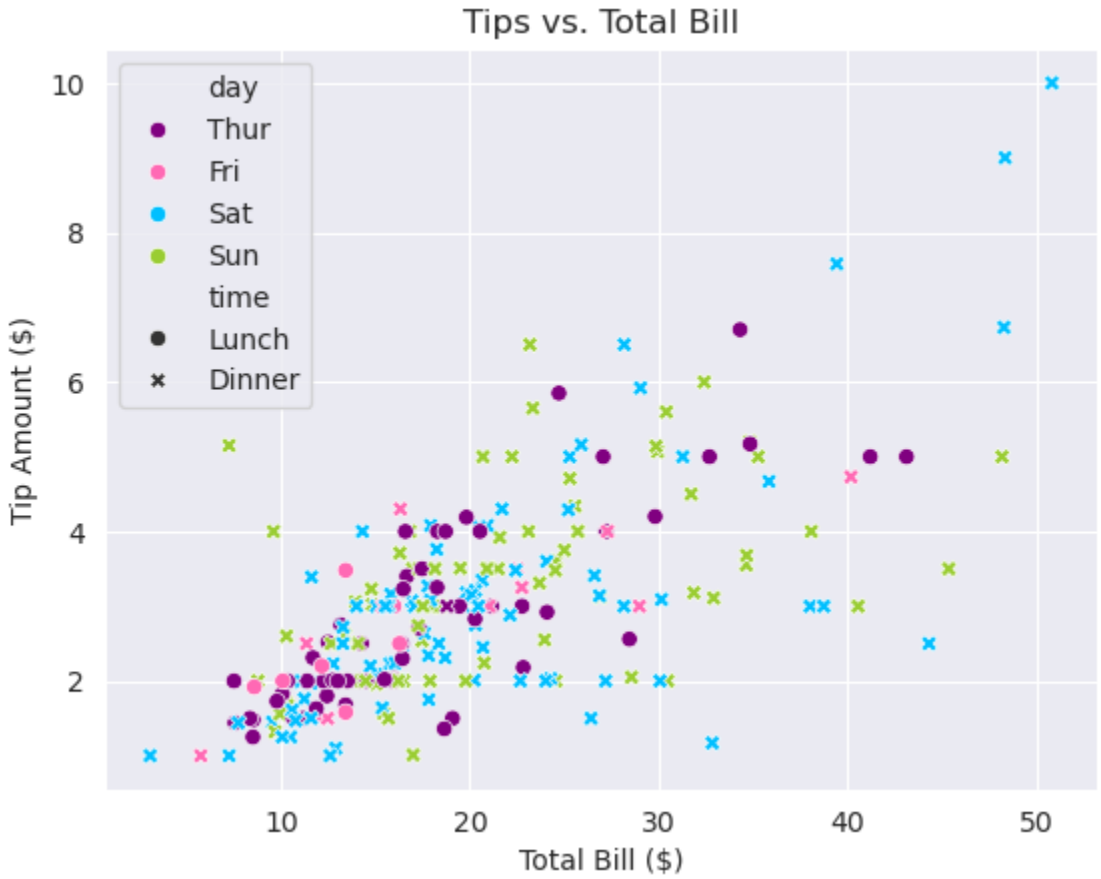
Add multiple variables

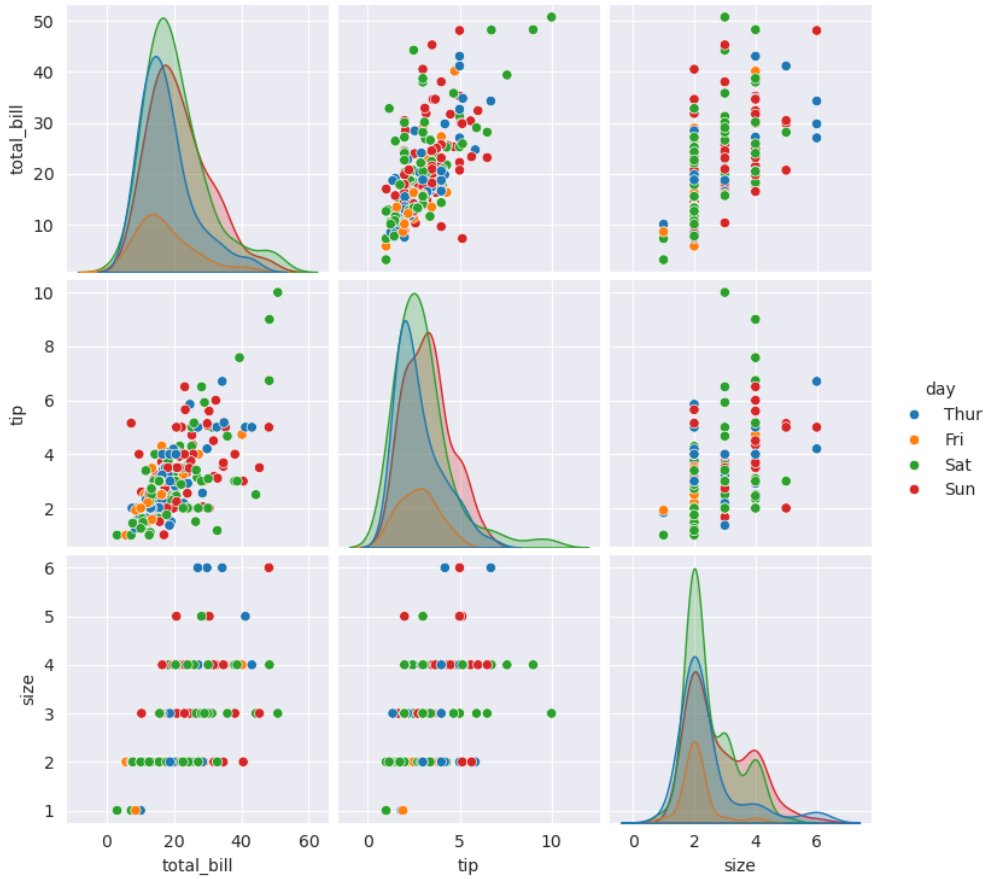
- Seaborn makes it very easy to represent multiple variables with different visual elements of our graph

```
tipgraph = sns.scatterplot(data=tips, x='total_bill',
                           y='tip', style = 'time', hue =
                           'day', palette = ['purple',
                           'hotpink', 'deepskyblue',
                           'yellowgreen'])

tipgraph.set(title='Tips vs. Total Bill',
             xlabel='Total Bill ($)',
             ylabel='Tip Amount ($)')
```

Add multiple variables

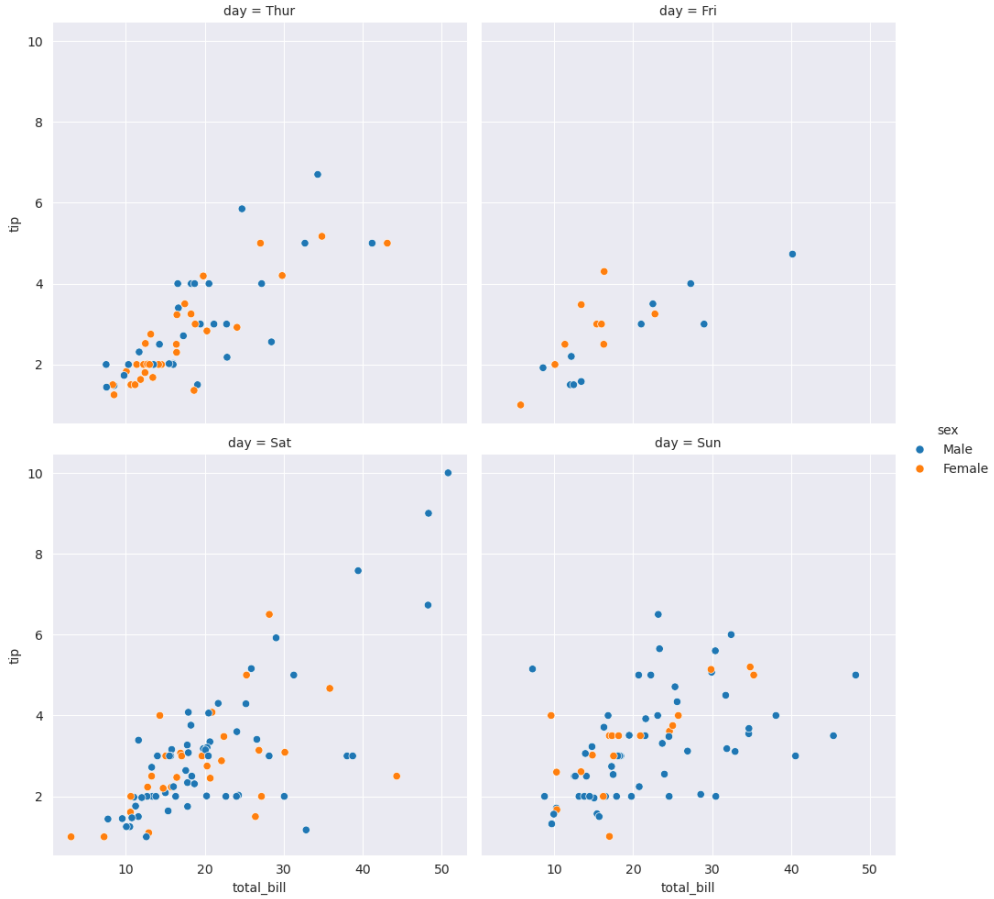




Pairplot

- **pairplot** is another convenient way to compare variables

```
sns.pairplot(
      data = tips,
      hue = 'day')
```



Relplot

- **relplot** also lets us explore specific levels within variables

```
daysplot = sns.relplot(
            data=tips,
            x="total_bill",
            y="tip",
            hue="sex",
            col="day",
            kind="scatter",
            col_wrap=2)
```

- **Activity**: Comment this snippet of code and describe what each new element is doing

Seaborn resources:

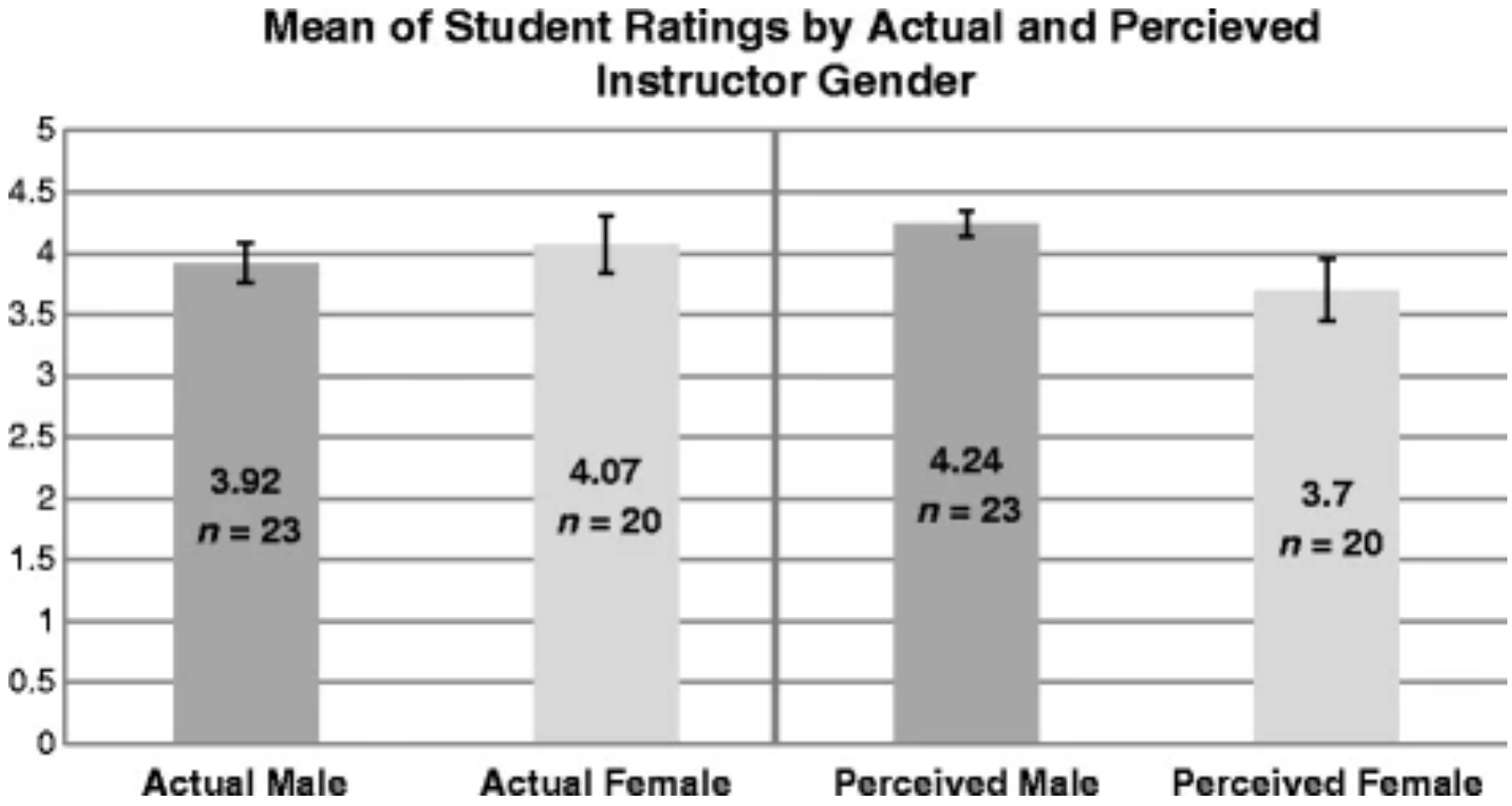
- Datacamp has a seaborn cheatsheet that covers the basics: https://images.datacamp.com/image/upload/v1676302629/Marketing/Blog/Seaborn_Cheat_Sheet.pdf
- Or look at the seaborn gallery: http://seaborn.pydata.org/examples/index.html

Case Study: Gender bias in teaching evaluations

How does faculty gender impact course evaluations?

- The gender of instructors (*perceived* or *actual*) influences how they are scored by students on teaching and course evaluations, with student evaluations tending to be significantly biased in favour of men and against women, even when all else is equal
- The bias in these student evaluations can adversely impact the ability of female scholars to be “full-time tenure-track, to hold tenured positions, to attain higher leadership roles in academia, and to earn the same salary as males in the same positions”

Mean of Student Ratings by Actual and Perceived Instructor Gender (MacNell et al., 2015)



Gendered Language in Teacher Reviews (Schmidt, 2015)

Gendered Language in Teacher Reviews

I've had trouble keeping this site up continuously during COVID. As of March 2021, I'm now trying a new strategy to cache common queries on the server even when the underlying database is down. If you find that many searches don't change the results, that's why.

This interactive chart lets you explore the words used to describe male and female teachers in about 14 million reviews from RateMyProfessor.com.

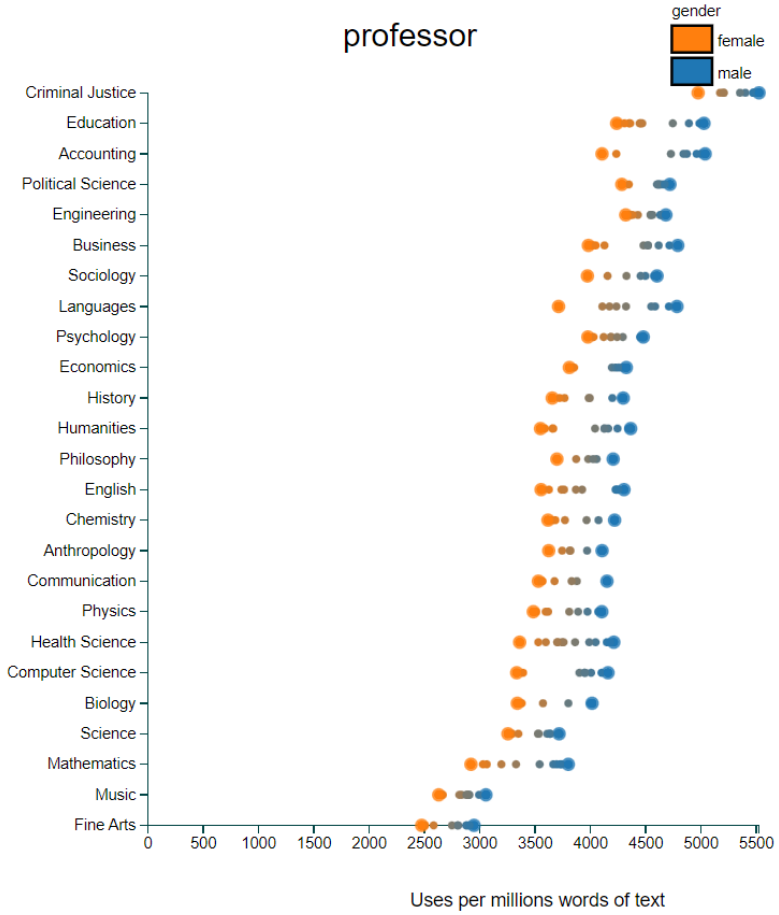
Not all words have gender splits, but a surprising number do. Even things like pronouns are used quite differently by gender.

Search term(s) (case-insensitive): use commas to aggregate multiple terms

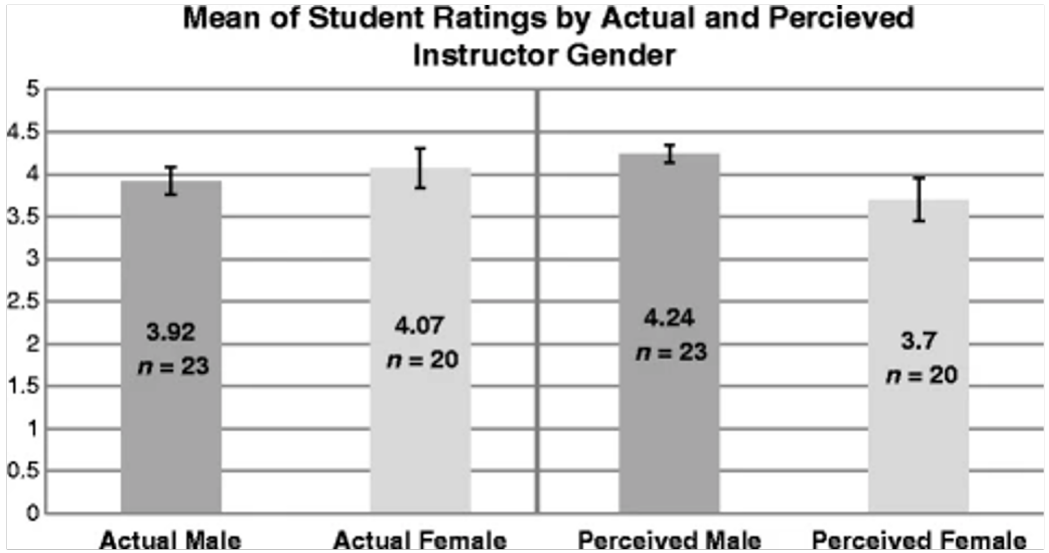
professor

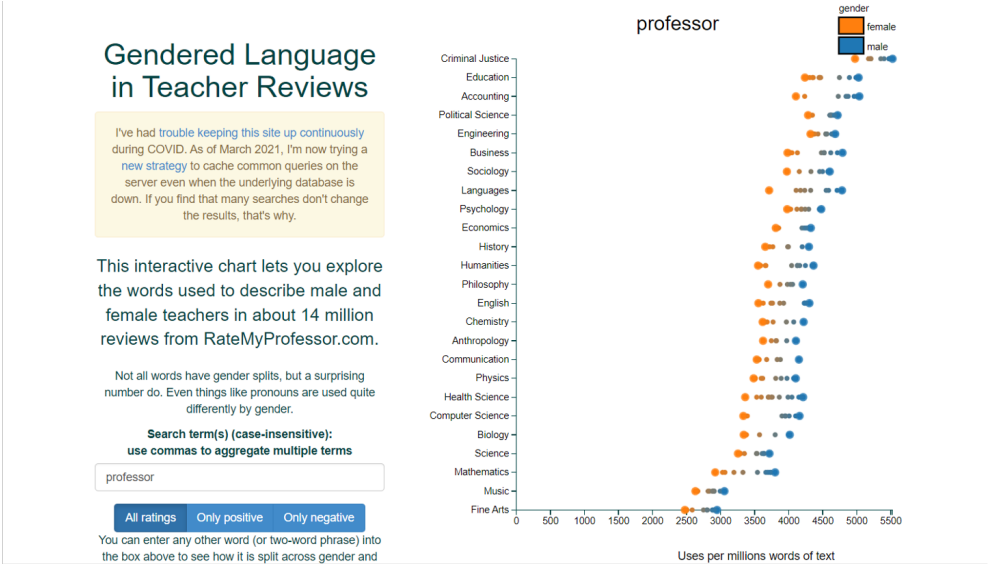
All ratings | Only positive | Only negative

You can enter any other word (or two-word phrase) into the box above to see how it is split across gender and



Activity: Comparing data visualizations





- **Let's discuss the two examples.**
  - How are they different? What does each visualization 'do'?
  - What are the pros and cons of each?

Static vs. dynamic data visualization

Defining static vs. dynamic data visualization

- **Static data visualization**
  - An image-based chart or infographic (think PDF, PNG, JPG)
  - A snapshot of data
  - Most of what we have seen so far in this course
- **Dynamic data visualization**
  - Interactive applications or web pages that allow users to modify or filter a data visualization
  - Multiple data stories in one

Benefits of dynamic data visualizations

- Dynamic data visualizations
  - Can provide information that cannot be obtained from static charts
  - Are useful for viewing individual-level data
  - Allow audiences to explore the data in-depth, supporting transparency and reproducibility
  - Can increase interest and engagement in research outputs

Costs of dynamic data visualizations

- Dynamic data visualizations
  - Make it more challenging to tell a single clear story or communicate a clear message
  - Can be confusing or overwhelming for audiences
  - Present access and sharing challenges (e.g. an image can be viewed in print, online, on mobile, but a dynamic viz might need to be hosted on a particular platform or software)

Designing dynamic data visualizations

Considerations

- Best practices for data visualization and accessibility still apply, but we also need to consider unique elements of dynamic data visualizations. For example:
  - Are the interactive elements easy to navigate?
  - Are the interactive elements accessible?
  - How much time will your audience have to interact with the data visualization?
  - **Important: Do interactive features actually help your data visualization serve your purpose?**

Elements of dynamic data visualizations

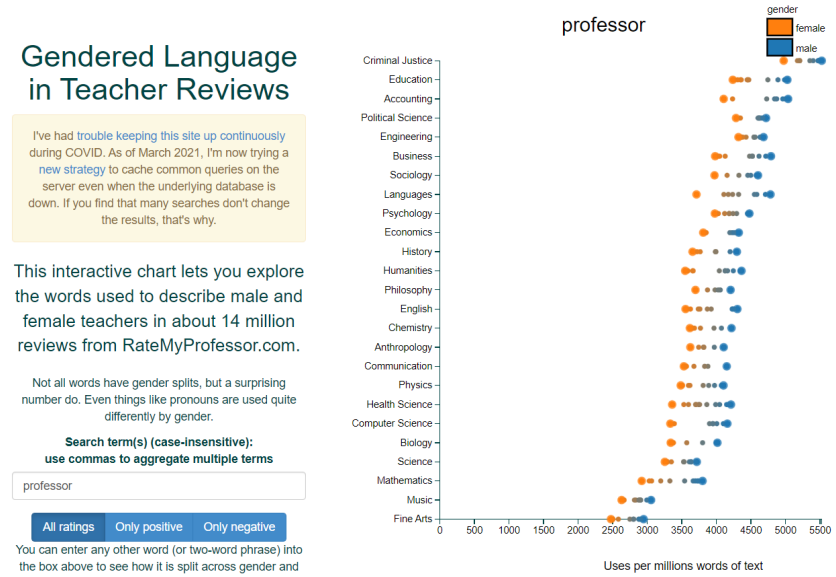
- We can conceptualize changes to our dynamic data visualizations in terms of how they affect two visual elements of our plots:
    - **Spatial elements** → Position and quantity (e.g. number of data points on a plot, scales of axes)
    - **Retinal elements** → Size, brightness, rotation, patterning, shape, and colour
- **We need to consider**: in our dynamic data visualizations, will spatial and retinal elements be fixed or mutable? Can they be created and deleted based on user interaction? Can their meanings change?

Types of changes

Identity-preserving	- Maintain associations between visual elements and underlying data - Some part of a representation stays constant (e.g., keeping the relative position of data points constant) - Important for comparing across time/snapshots
Transitional	- Maintain some associations, and limit changes to known values - New elements may be added or changed - Balance flexibility with the ability to compare over short periods of time/small changes between snapshots
Immediate	- Generally do not preserve associations - Create and delete elements, alter scales, remap variables every time a new ‘snapshot’ is created - Does not allow for comparison across time/snapshots

Activity: Types of changes

- Return to our earlier example of gendered language from RateMyProfessor.com
- What types of changes occur when we interact with this dynamic data visualization?
- Which spatial and retinal elements stay constant? Which change?
- If you were redesigning this viz, would you choose to make different changes? Why or why not?



Plotly

Setting up

- First, let's import our package and make some sample data

```
import plotly.graph_objects as go     # 'go' is 'graph objects'

x1 = np.array(["Luffy", "Zoro", "Nami", "Usopp", "Sanji"])
y1 = np.array([110, 180, 240, 99, 220])
```

Making our plot

- Plotly syntax is different from matplotlib/seaborn
- First, make our figure

```
graph = go.Figure()
```

- Next, choose our plot type (bar) and our data

```
graph.add_trace(go.Bar(x=x1, y=y1))
```

Making our plot

- Next, update our layout to include titles

```
graph.update_layout(
    title="Pirate Scores",
    xaxis_title="Pirates",
    yaxis_title="Score")
```

- Finally, show our plot!

```
graph.show()
```

Plotly features

- If we hover over the bars on our plot, we get automatic data labels
- If we click and drag over a portion of our plot, we can zoom in
    - Double click to zoom back out
- We can click and drag on our axis to scroll
- In the upper-right menu:
    - Save as PNG
    - Pan/zoom
    - Box/Lasso select

Exporting plotly graphs

- We can also save plotly graphs as HTML files and embed them in our webpages!

```
graph.write_html("../../folders/pirategraph.html")
```

Customizing plotly graphs

```
graph = go.Figure()
graph.add_trace(go.Scatter(x=x1, y=y1, mode='markers',    # we want points for a scatter plot
    marker=dict(
        size=15,                              # point size
        color='hotpink',                      # point colour
        opacity=1,                            # point transparency/alpha
        line=dict(width=5, color='purple')  # point outline
    )))

graph.update_layout(
    title='Interactive Pirate Plot',
    xaxis_title='Pirates',
    yaxis_title='Scores',
    width=500, height=500)
```

Wordclouds and venn diagrams

Why use code for qualitative viz?

Making wordclouds

- Import the **wordcloud** package and load our sample dataset of movie quotes

```
from wordcloud import WordCloud
df = pd.read_csv("https://raw.githubusercontent.com/prasertcbs/basic-dataset/master/movie_quotes.csv",
on_bad_lines='skip')
df
```

Making wordclouds

- We'll make a simple word cloud using the 'quote' variable

```
# join all our text from each row from our quote column into a string

text = " ".join(each for each in df.quote)

# generate our wordcloud image

wordcloud = WordCloud(background_color="white",
colormap = 'inferno').generate(text)

# use matplotlib syntax to put our image in a figure

fig, ax = plt.subplots(figsize=(7, 3))
ax.imshow(wordcloud,                  # remember 'imshow' from when we added pictures to our matplotlib axes
        interpolation='bilinear')   # this line helps smooth our image
ax.axis("off")
```

Making wordclouds

Making venn diagrams

- We can use the **matplotlib_venn** extension to make venn diagrams
- First import the package

```
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
```

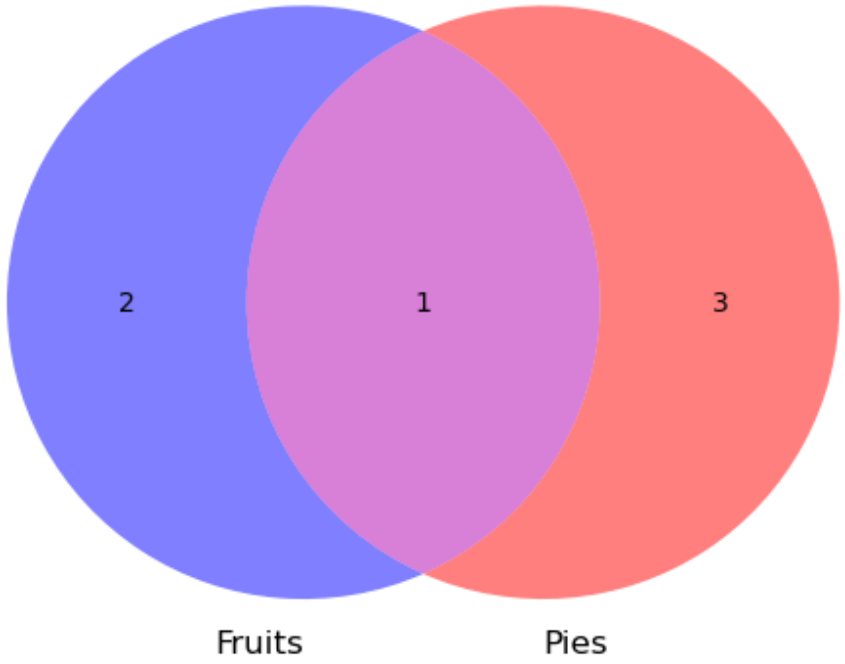
- Then define our sets

```
A = set(["apple", "banana", "watermelon"])
B = set(["pumpkin", "blueberry", "apple", "key lime"])
```

Making venn diagrams

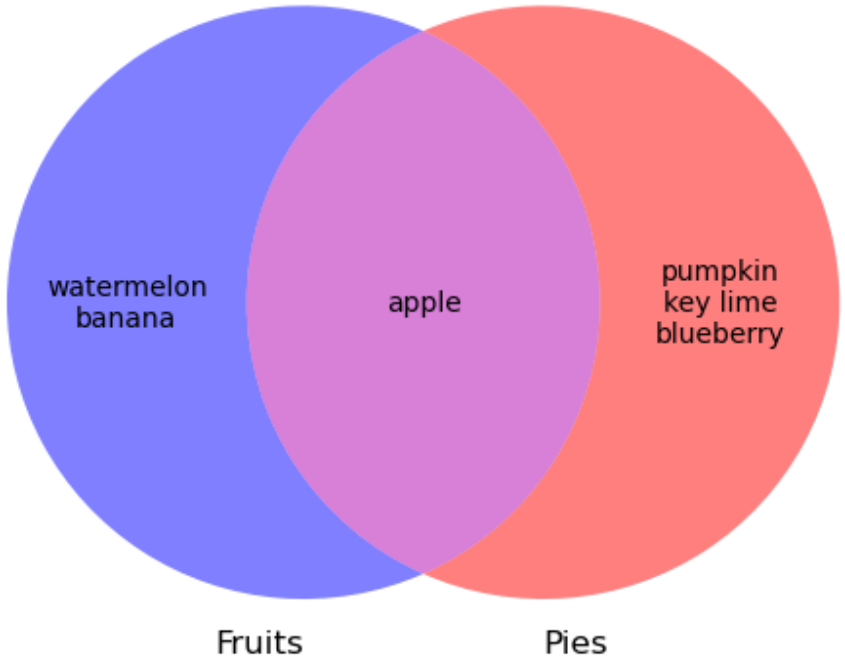
- Next, assign our sets to each circle and modify the appearance

```
diagram = venn2_unweighted([A, B],
                           set_labels = ('Fruits', 'Pies'),
                           set_colors=("blue", "red"),
                           alpha=0.5)
plt.show()
```



Making venn diagrams

- By default, our output will only show counts of how many items are in each set



Modifying venn diagrams

```
diagram.get_label_by_id("10")
    .set_text("n".join(A - B))

diagram.get_label_by_id("11")
    .set_text("n".join(A & B))

diagram.get_label_by_id("01")
    .set_text("n".join(B - A))
```

*The numbers in our brackets come from the documentation