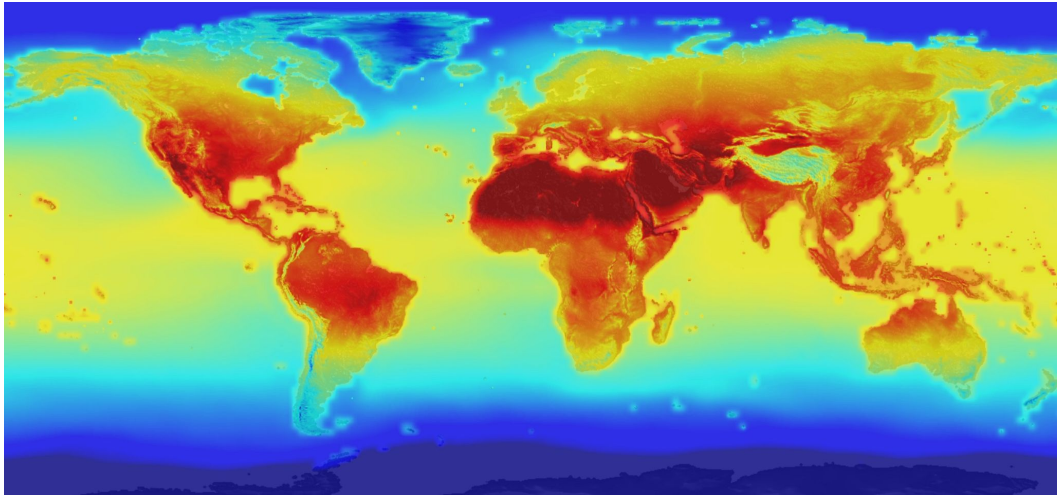
Data Visualization

Adapted from Data Carpentry's material:
https://datacarpentry.org/python-ecology-lesson/07-visualization-ggplot-python.html

LIBRARIES
COLORADO STATE UNIVERSITY

Why we need data visualizations

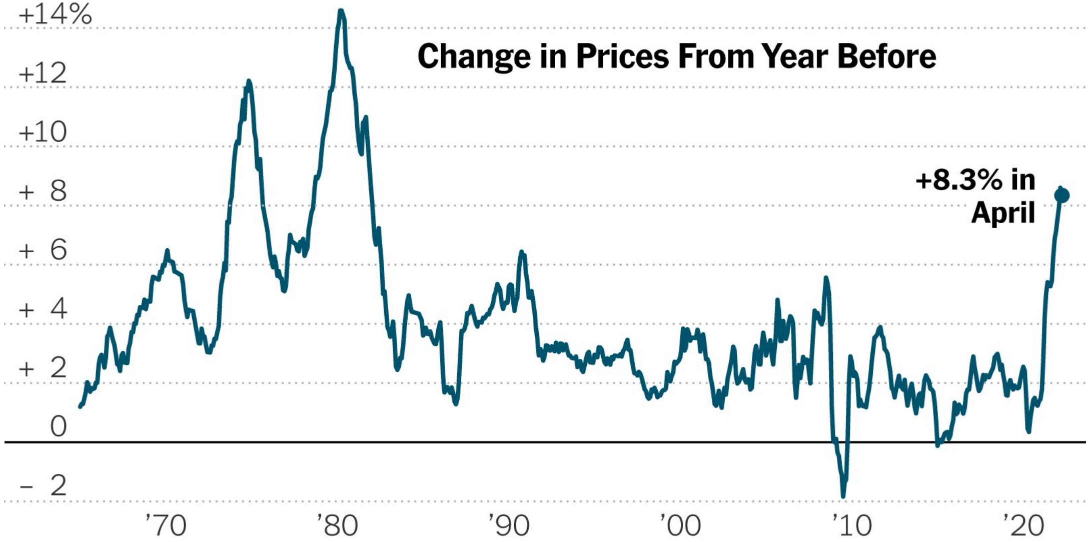
- To Explore, Monitor and Explain
  - Ref: https://spectrum.adobe.com/page/data-visualization-fundamentals/
- Data alone is often too complex for us and our audiences to understand
  - E.g generate heat maps to understand/analyze the global warming



Imagine just reading the heat data table!

Why we need data visualization continued

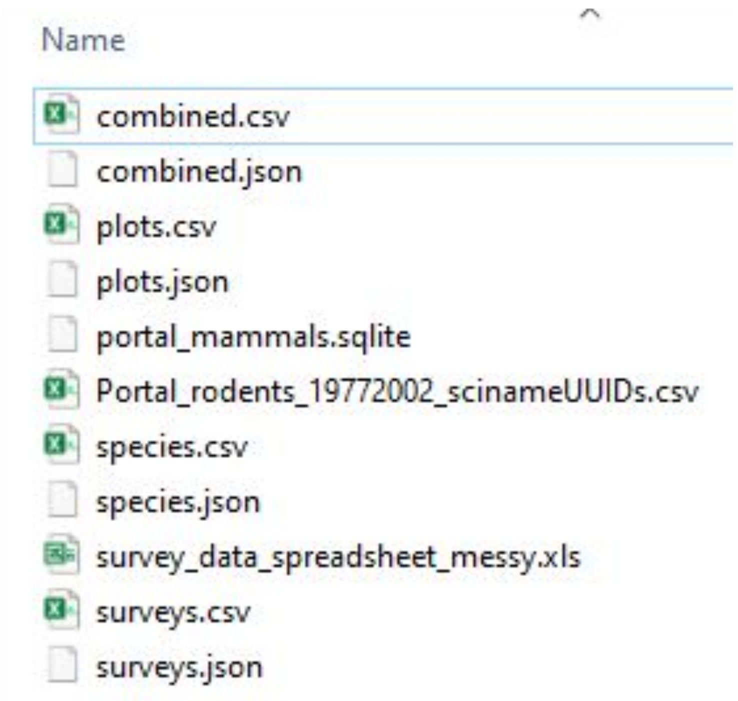
- Visualization can help:
  - Present the situation
  - e.g historical inflation rate for USA



Data is more explanatory when graphed

Exercise: Download the Data

- Go to: https://figshare.com/ndownloader/articles/1314459/versions/10
- Unzip the file
- Move all downloaded data into 'data' folder



Name

combined.csv
combined.json
plots.csv
plots.json
portal_mammals.sqlite
Portal_rodents_19772002_scinameUUIDs.csv
species.csv
species.json
survey_data_spreadsheet_messy.xls
surveys.csv
surveys.json

Making Data Visualization Easier

- Tools:
  - plotline - https://plotnine.readthedocs.io/en/stable/#
    - plotnine facilitates the creation of highly-informative plots
    - Based on the R implementation of ggplot
    - Built on Matplotlib (https://matplotlib.org/)
    - Interacts well with Pandas structured data
- Installation:
  - `Using Anaconda Navigator>Environments`
    - `Select "not installed" from the dropdown`
    - `Enter 'plotnine' into the search field`
    - `Click the checkbox next to plotnine in the list, then Apply`
  - `Alternatively use: conda install -y -c conda-forge plotnine` **within the Spyder Console**
- Test installation
  - `import plotnine as p9 #`**from python**

Exercise: Plotting with plotnine

- Create a graph step-by-step

    ```python
    import plotnine as p9
    import pandas as pd

    surveys_complete = pd.read_csv('data/surveys.csv')
    surveys_complete = surveys_complete.dropna()

    # plot the weight compared to the hindfood_length
    surveys_plot = p9.ggplot(data=surveys_complete, mapping=p9.aes(x='weight', y='hindfoot_length'))
    surveys_plot +  p9.geom_point() # creates the plot
    ```

Other aesthetics (aes) arguments: color, colour, fill, linetype, shape, size and stroke.

- Other common plots: *geom_bar, geom_box, geom_line, geom_smooth*
    - Full list: https://plotnine.readthedocs.io/en/stable/api.html

Exercise: Chaining elements with plotnine

- Use brackets and the '+' operator for adding elements to your plot
    - surveys_plot = p9.ggplot(data=surveys_complete, mapping=p9.aes(x='weight', y='hindfoot_length', color='species_id'))
    - (surveys_plot
    - + p9.geom_point()
    - **+ p9.xlab("Weight (g)")**
    - **+ p9.scale_x_log10()**
    - **+ p9.theme_bw()**
    - **+ p9.theme(text=p9.element_text(size=16)))**

- Change x or y labels for clarity
- log10 of the x-axis for better lower number interpretation
- Use theme_* to e.g. 'theme_bw' for changing background to white
- theme() to change additional parameters

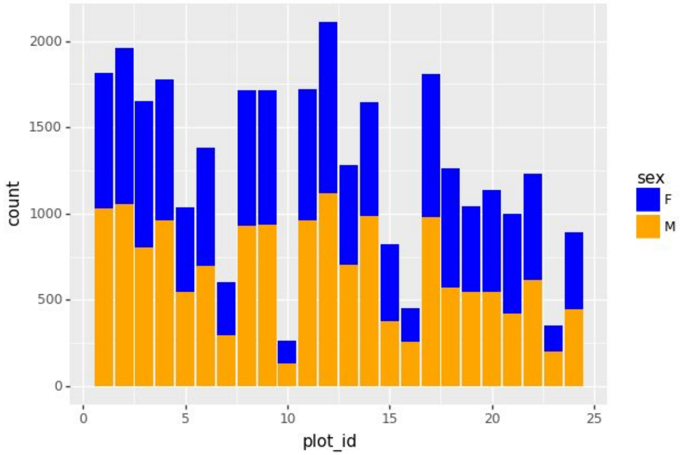
Exercise: Other plots with plotnine

- Boxplot
  - # visualize the distribution of weight within each species_id
  - `surveys_plot = p9.ggplot(data=surveys_complete, mapping=p9.aes(x='species_id', y='weight'))`
  - `surveys_plot + p9.geom_boxplot()`
- Time series line chart
  - *`#calculate number of counts per year for each species`*
  - `yearly_counts = surveys_complete.groupby(['year', 'species_id'])['species_id'].count()`
  - `yearly_counts = yearly_counts.reset_index(name='counts')`
    `# converts Series to Dataframe`
  - `surveys_plot = p9.ggplot(data=yearly_counts, mapping=p9.aes(x='year', y='counts',  color='species_id'))`
  - `surveys_plot + p9.geom_line()`

Challenge: Bar plot adaptations

Adapt the boxplot from the previous exercise and create a bar chart

- mapping the 'sex' variable to the color fill
- Change the scale of the color fill by providing the colors blue and orange manually (see API reference to find the appropriate function).



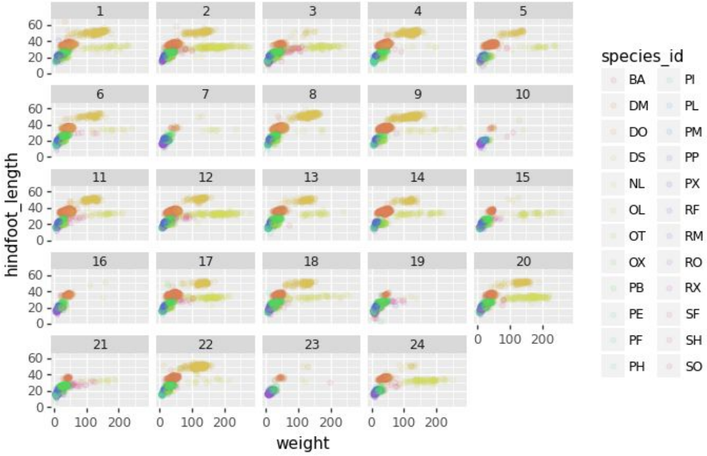
Exercise: Split plots

- Using facet_wrap
  - Extracts plots into an arbitrary number of dimensions to allow them to cleanly fit on one page

  -
    ```
    # plot the weight compared to the hindfood_length for each location
    ```

  -
    ```
    surveys_plot = p9.ggplot(data=surveys_complete,
    mapping=p9.aes(x='weight', y='hindfoot_length', color='species_id'))
    ```

  -
    ```
    surveys_plot + p9.geom_point(alpha=0.1) + p9.facet_wrap("plot_id")
    ```

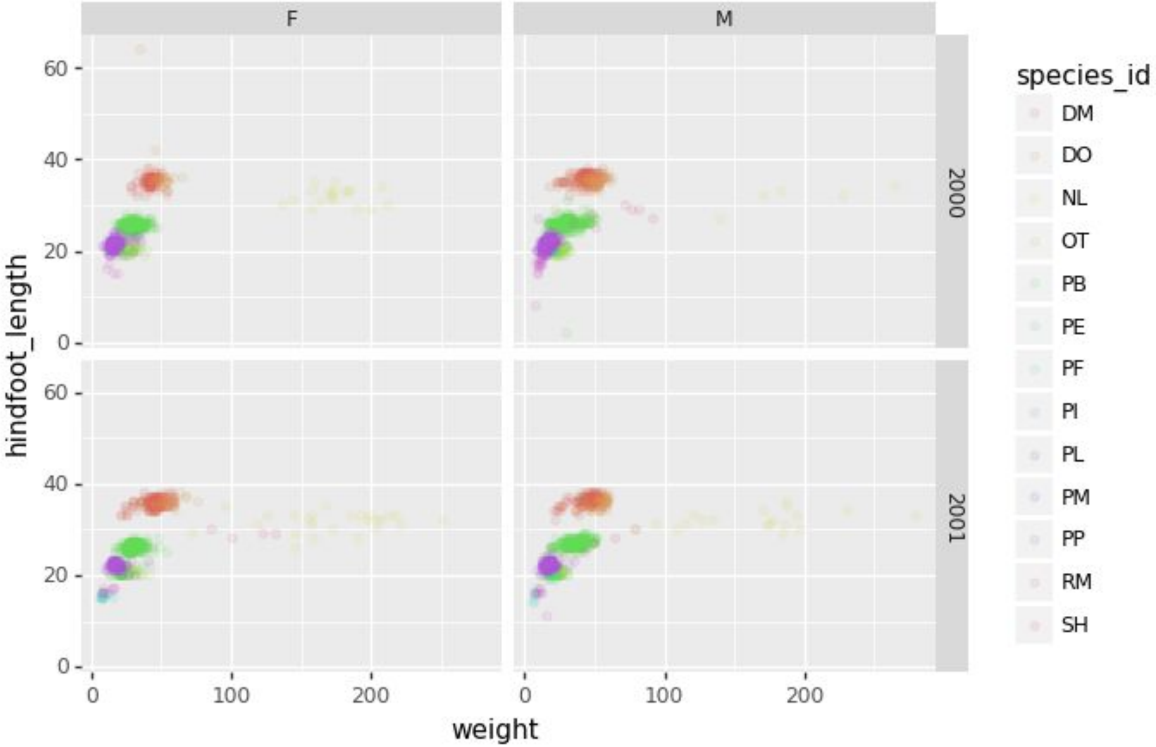


Exercise: Split plots

- Using facet_grid
    - To specify how you want your plots to be arranged
    - Uses formula notation (rows ~ columns)
    - A '.' can be used as a placeholder that indicates only one row or column) e.g **"year ~ ."**
    - `# select years 2001 and 2002 and plot weight vs hindfoot_length separated by year and sex`
    - `survey_2000_2001 = surveys_complete[surveys_complete["year"].isin([2000, 2001])]`
    - `surveys_plot =  p9.ggplot(data=survey_2000_2001, mapping=p9.aes(x='weight',  y='hindfoot_length', color='species_id'))`
    - `surveys_plot  + p9.geom_point(alpha=0.1) + p9.facet_grid("year ~ sex")`

Exercise: Split plots

- Using facet_grid



Further Customizations

- Change text angle
  - surveys_plot = p9.ggplot(data=surveys_complete, mapping=p9.aes(x='factor(year)'))
  - surveys_plot + p9.geom_bar()
  - surveys_plot + p9.geom_bar() + p9.theme_bw() + **p9.theme(axis_text_x = p9.element_text(angle=90)**)
- Use a custom theme and categorical variable with 'factor' function
  - *my_custom_theme = p9.theme(axis_text_x = p9.element_text(color="grey", size=10, angle=90, hjust=.5), axis_text_y = p9.element_text(color="grey", size=10))*
  - surveys_plot = p9.ggplot(data=surveys_complete, mapping=p9.aes(x='*factor*(year)'))
  - surveys_plot + p9.geom_bar() + *my_custom_theme*

Export the Plot

- Saving plots

  -
    ```
    my_plot = (p9.ggplot(data=surveys_complete,
    mapping=p9.aes(x='weight', y='hindfoot_length', color='species_id'))
    + p9.geom_point())
    ```

  -
    ```
    my_plot.save("my_bar_graph.png", width=10, height=10, dpi=300)
    ```

Data visualization using matplotlib

- Matplotlib is a well documented python library developed to emulate Matlab’s plotting commands
  - The plotting environment may seem friendlier if you are already experienced with Matlab

- Matplotlib can be installed to your conda environment as follows:

  -
    ```
    conda install -c conda-forge matplotlib
    ```

- You can test the installation by:

  -
    ```python
    import matplotlib as plt
    ```

- The detailed documentation is available at: https://matplotlib.org/

Scatterplot example using matplotlib

- For this we use the scatter() function from the pyplot sub-module

```
import matplotlib.pyplot as plt
import pandas as pd

surveys_complete = pd.read_csv('data/surveys.csv')
surveys_complete = surveys_complete.dropna()

x = surveys_complete.weight
y = surveys_complete.hindfoot_length
surveys_plot_plt = plt.scatter(x, y, s =10, c='black')
plt.show()
```

- The aesthetic arguments are passed directly to the scatter() function:
    - s -> size of the marker; c -> color of the marker
    - More on: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html

Plot element customization using matplotlib

- In order to add category-wise coloring we must extract the data we need represented in color:
  - `import numpy as np`
  - `labels, index = np.unique(surveys_complete.species_id, return_inverse=True)`
- Now we apply the indices to the data points:
  - `surveys_plot_plt = plt.scatter(x, y, s =10,  c=index)`
- You can let the legend() function to handle the coloring and specify where you want the legend to appear, and appearance of the legend box:
  - `plt.legend(surveys_plot_plt.legend_elements(num=None)[0], labels, ncol=6, loc='upper left', bbox_to_anchor=(-0.05, 1.15) )`
- Add other aspects such as x-label title, applying log scale to x-axis etc.
  - `plt.xlabel("Weight (g)")`
  - `plt.xscale("log")`
  - `plt.show()`

Plot element customization using matplotlib (contd.)

- Setting tick-label parameters:
  - `plt.xticks()` and `plt.yticks()` can be used to customize the tick-labels of x and y axes, respectively
  - E.g.: `plt.xticks(fontsize='25', rotation=30,horizontalalignment='right')`
  - More: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html
- Setting font type:
  - Matplotlib gives user control over the font type, size etc. through `plt.rcParams()`
  - E.g.: `plt.rcParams['font.family']`, `plt.rcParams['mathtext']`
  - `mathtext` refers to the ability of matplotlib to integrate symbols and equations to your plot's title or sub-titles or legend
  - Symbols and equations can be written in Latex E.g.: `$\lambda$` will yield λ
  - More on this and how to set your own graphing style, here: https://matplotlib.org/stable/tutorials/introductory/customizing.html

Even more control over the axes:

- So far we discussed using the plt object directly
- We can use `plt.subplots()` for increased customizability and even enabling graph grids
  - `fig, ax =plt.subplots()`
- Now if you want to have a second y-axis to represent more data corresponding to the same x-axis:
  - `ax2 = ax1.twinx()`
  - `ax1.plot(...)`
  - `ax2.plot(...)`
  - You can stack this to have as many extra y axis as needed
- Enabling multiple graphs to be in a 2x2 grid (for example):
  - `fig, ax = plt.subplots(nrows=2, ncols=2)`
- For easier control of each graph in the grid:
  - `fig, ((ax0, ax1),(ax2, ax3)) = plt.subplots(nrows=2, ncols=2)`

Boxplots with matplotlib

- For boxplots, matplotlib provides the `boxplot()` function
- The boxplot function is incredibly versatile in its customizability, but it only accepts a sequence of vectors or a matrix as its input
- So, unlike plotnine, matplotlib requires significantly more data wrangling
- To visualize the distribution of weight within each species_id:

```
data=[]
labels=[]
for element in np.unique(surveys_complete.species_id):
   data.append(surveys_complete.loc[surveys_complete['species_id'] == element, 'weight'].to_numpy())
   labels.append(element)

plt.boxplot(data, labels=labels) #additional arguments can be provided to control whisker and box width, marker size, shape, color, opacity etc.
plt.xlabel("Species ID")
plt.ylabel("weight distribution")

plt.show()
```

Combining matplotlib elements with plotnine

- We can use plotnine and its in-built functions for graphing, while using matplotlib for its customizability
- To do this we need to convert our plotnine graph to a matplotlib object:

```
myplot = (p9.ggplot(data=surveys_complete,
               mapping=p9.aes(x='hindfoot_length', y='weight')) +
               p9.geom_point())

plt_myplot = myplot.draw() #plotnine object converted to matplotlib object
p9_ax = plt_myplot.axes[0]  #This generates the “ax” parameters
```

- The p9_ax object can now be customized using matplotlib, as discussed:

```
p9_ax.set_xlabel("Hindfoot length")
p9_ax.tick_params(labelsize=16, pad=8)
p9_ax.set_title('Scatter plot of weight versus hindfoot length',
fontsize=15)
plt.show()
```

5 Minute Post Workshop Evaluation

https://forms.office.com/r/E1Yy7RNv3y