Data Visualization and Plotting with Python

Jupyter notebook (Anaconda)

Class - II

Biomedical Data Science Initiative class @ NIA/NIH

Dr. Showkat Dar

PostDoc

Computational Genomics Unit (LGG)

Class Times: 12:30-2:00 pm Tuesday, June 11th and Thursday, June 13th, 2024

Class link: https://nih.zoomgov.com/j/1613089725?pwd=dE8reWkzRnBSaTRWNXB4Sk5XWElDQT09

Our Schedule -> Hands-On :: 2-way Learning

Day-I:

1. Background - Rules for better figures
2. Hands-on – Introduction to Python's plotting libraries (Matplotlib and Seaborn)
3. Basic plotting techniques (line graphs, bar charts, and histograms) and saving plots in different formats.

Day-II:

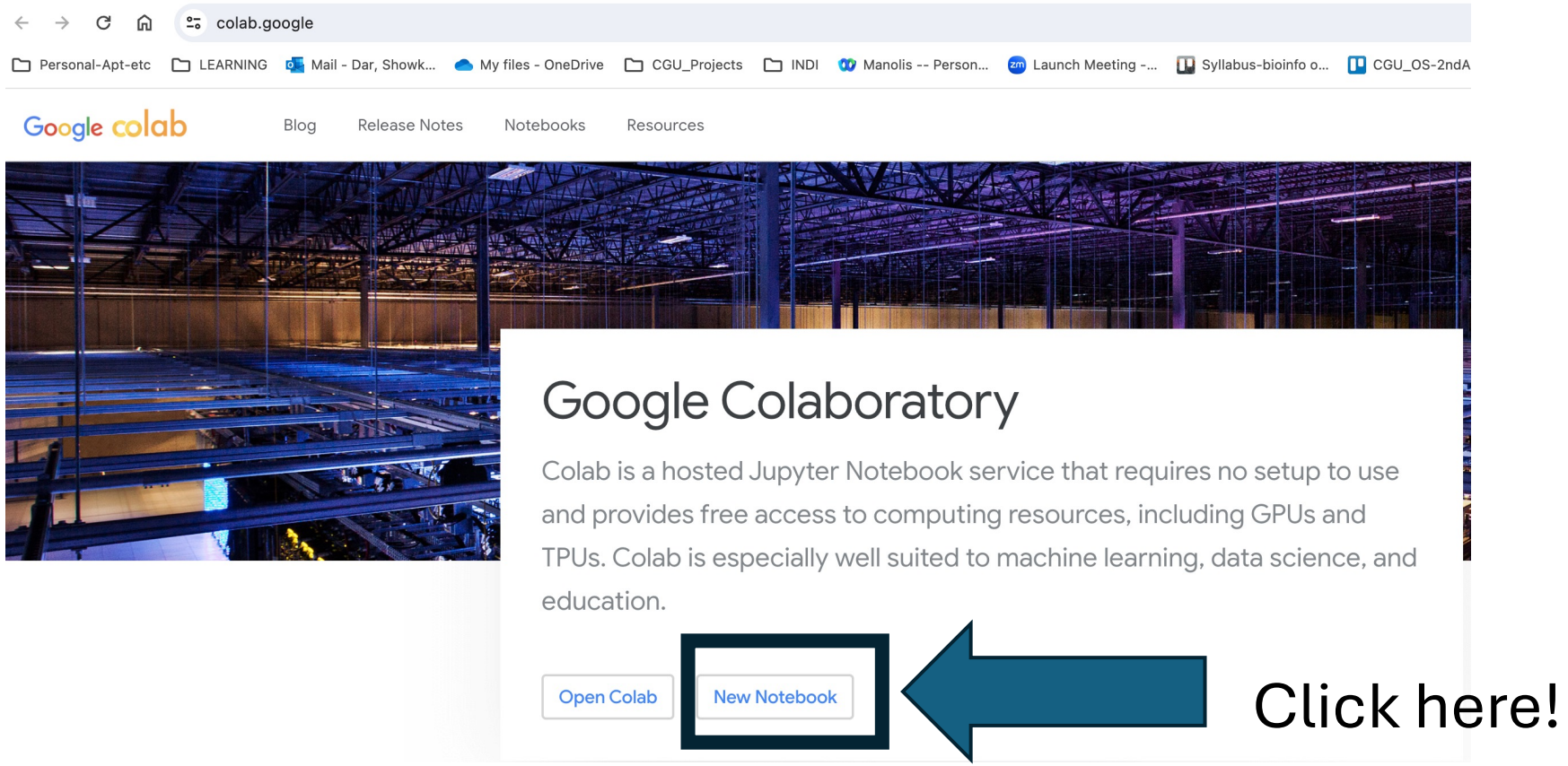
4. Advanced visualizations (heatmaps, pair plots, and time series visualization)
5. Best practices in data visualization.

➢ 10 Rules - Recap

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure to the Support Medium
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid “Chartjunk”
9. Message Trumps Beauty
10. Get the Right Tool

Online jupyter notebook options.

- Go to https://colab.google/



NAME.ipynb

File Edit View Insert Runtime Tools Help

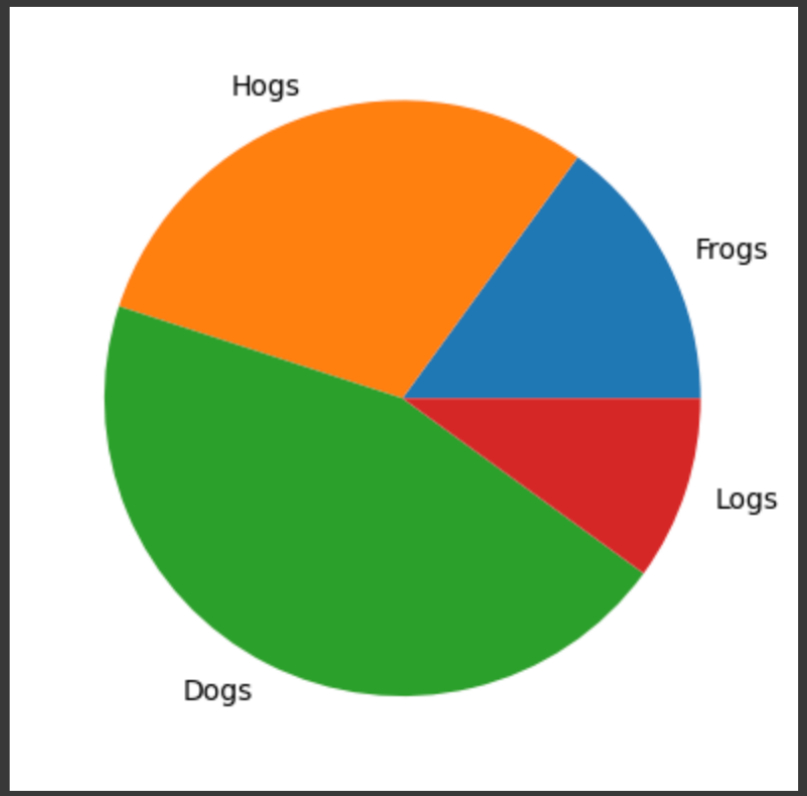
+ Code + Text

```
[2]  import matplotlib.pyplot as plt

     labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
     sizes = [15, 30, 45, 10]

     fig, ax = plt.subplots()
     ax.pie(sizes, labels=labels)
     plt.savefig("pieplot_labels.png")
```

1s



Day-II

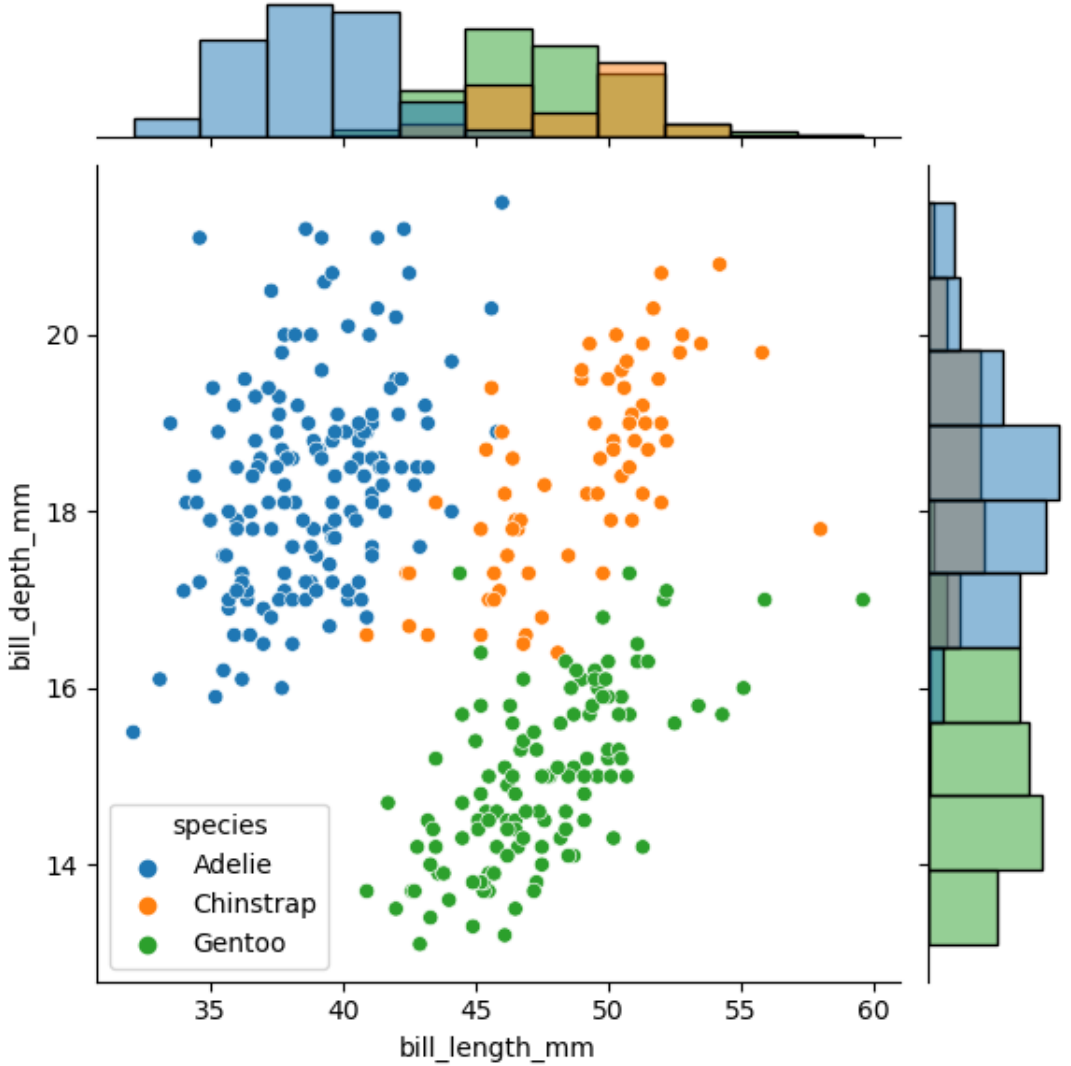
4. Advanced visualizations (heatmaps, pair plots, and time series visualization)

5. Best practices in data visualization.

seaborn : JointGrid

An object managing multiple subplots that correspond to joint and marginal axes for plotting a bivariate relationship or distribution.

class seaborn.**JointGrid**(*data=None, *, x=None, y=None, hue=None, height=6, ratio=5, space=0.2, palette=None, hue_order=None, hue_norm=None, dropna=False, xlim=None, ylim=None, marginal_ticks=False*)



https://seaborn.pydata.org/generated/seaborn.JointGrid.html

seaborn : JointGrid

Parameters: **data** : *`pandas.DataFrame`, `numpy.ndarray`, mapping, or sequence*

Input data structure. Either a long-form collection of vectors that can be assigned to named variables or a wide-form dataset that will be internally reshaped.

x, y : *vectors or keys in `data`*

Variables that specify positions on the x and y axes.

height : *number*

Size of each side of the figure in inches (it will be square).

ratio : *number*

Ratio of joint axes height to marginal axes height.

space : *number*

Space between the joint and marginal axes

dropna : *bool*

If True, remove missing observations before plotting.

{x, y}lim : *pairs of numbers*

Set axis limits to these values before plotting.

marginal_ticks : *bool*

If False, suppress ticks on the count/density axis of the marginal plots.

space : *number*

Space between the joint and marginal axes

dropna : *bool*

If True, remove missing observations before plotting.

{x, y}lim : *pairs of numbers*

Set axis limits to these values before plotting.

marginal_ticks : *bool*

If False, suppress ticks on the count/density axis of the marginal plots.

hue : *vector or key in `data`*

Semantic variable that is mapped to determine the color of plot elements. Note: unlike in `FacetGrid` or `PairGrid`, the axes-level functions must support `hue` to use it in `JointGrid`.

palette : *string, list, dict, or `matplotlib.colors.Colormap`*

Method for choosing the colors to use when mapping the `hue` semantic. String values are passed to `color_palette()`. List or dict values imply categorical mapping, while a colormap object implies numeric mapping.

hue_order : *vector of strings*

Specify the order of processing and plotting for categorical levels of the `hue` semantic.

hue_norm : *tuple or `matplotlib.colors.Normalize`*

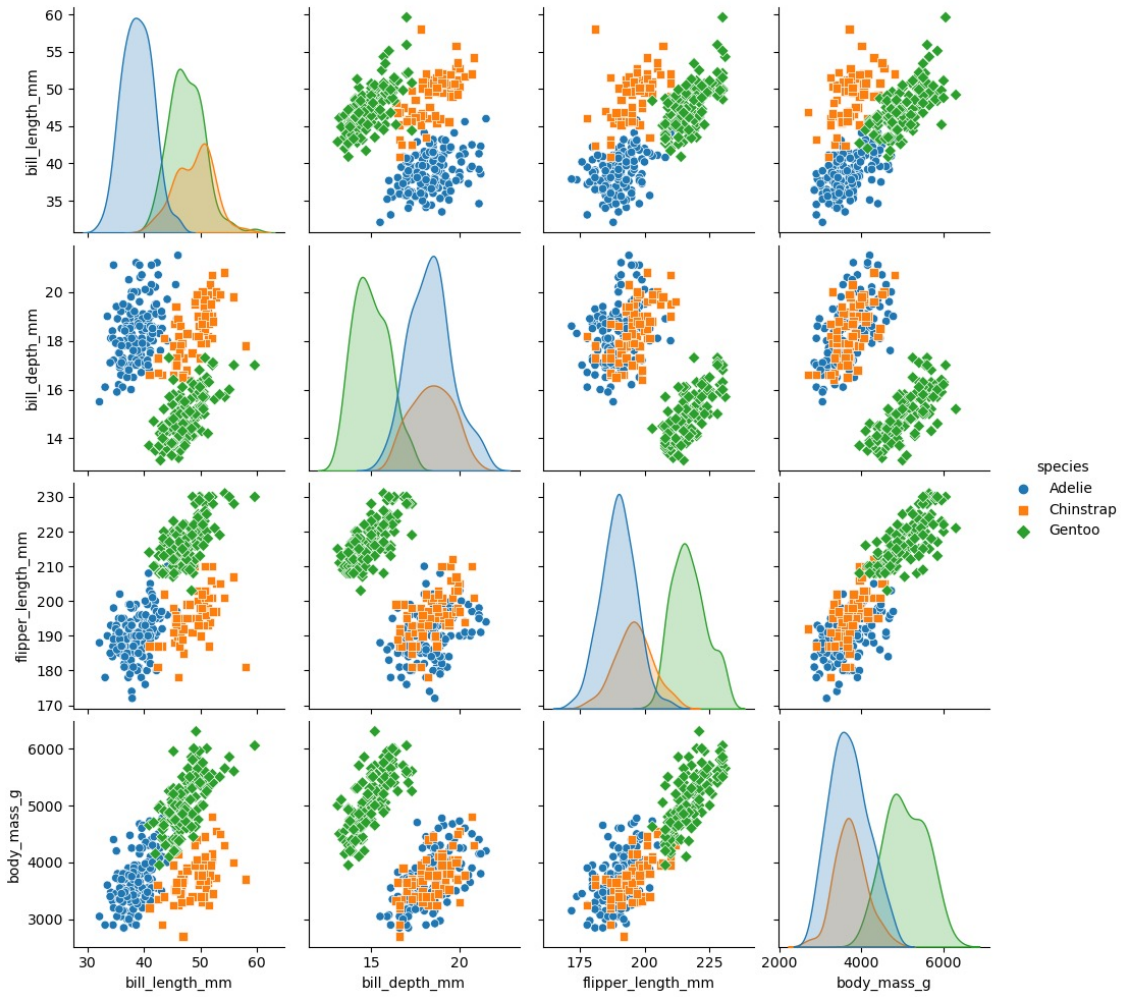
Either a pair of values that set the normalization range in data units or an object that will map from data units into a [0, 1] interval. Usage implies numeric mapping.

https://seaborn.pydata.org/generated/seaborn.JointGrid.html

seaborn : PairPlot

A pair plot, also known as a scatterplot matrix, is a matrix of graphs that enables the visualization of the relationship between each pair of variables in a dataset. It combines both histogram and scatter plots, providing a unique overview of the dataset's distributions and correlations.

seaborn.**pairplot**(*data*, *, *hue*=*None*, *hue_order*=*None*, *palette*=*None*, *vars*=*None*, *x_vars*=*None*, *y_vars*=*None*, *kind*='scatter', *diag_kind*='auto', *markers*=*None*, *height*=2.5, *aspect*=1, *corner*=*False*, *dropna*=*False*, *plot_kws*=*None*, *diag_kws*=*None*, *grid_kws*=*None*, *size*=*None*)



https://seaborn.pydata.org/generated/seaborn.pairplot.html

seaborn : Pairplot

Parameters: **data** : *pandas.DataFrame*

Tidy (long-form) dataframe where each column is a variable and each row is an observation.

hue : *name of variable in* `data`

Variable in `data` to map plot aspects to different colors.

hue_order : *list of strings*

Order for the levels of the hue variable in the palette

palette : *dict or seaborn color palette*

Set of colors for mapping the `hue` variable. If a dict, keys should be values in the `hue` variable.

vars : *list of variable names*

Variables within `data` to use, otherwise use every column with a numeric datatype.

{x, y}_vars : *lists of variable names*

Variables within `data` to use separately for the rows and columns of the figure; i.e. to make a non-square plot.

kind : *{'scatter', 'kde', 'hist', 'reg'}*

Kind of plot to make.

diag_kind : *{'auto', 'hist', 'kde', None}*

Kind of plot for the diagonal subplots. If 'auto', choose based on whether or not `hue` is used.

markers : *single matplotlib marker code or list*

Either the marker to use for all scatterplot points or a list of markers with a length the same as the number of levels in the hue variable so that differently colored points will also have different scatterplot markers.

height : *scalar*

Height (in inches) of each facet.

aspect : *scalar*

Aspect * height gives the width (in inches) of each facet.

corner : *bool*

If True, don't add axes to the upper (off-diagonal) triangle of the grid, making this a "corner" plot.

dropna : *boolean*

Drop missing values from the data before plotting.

{plot, diag, grid}_kws : *dicts*

Dictionaries of keyword arguments. `plot_kws` are passed to the bivariate plotting function, `diag_kws` are passed to the univariate plotting function, and `grid_kws` are passed to the `PairGrid` constructor.

grid : *PairGrid*

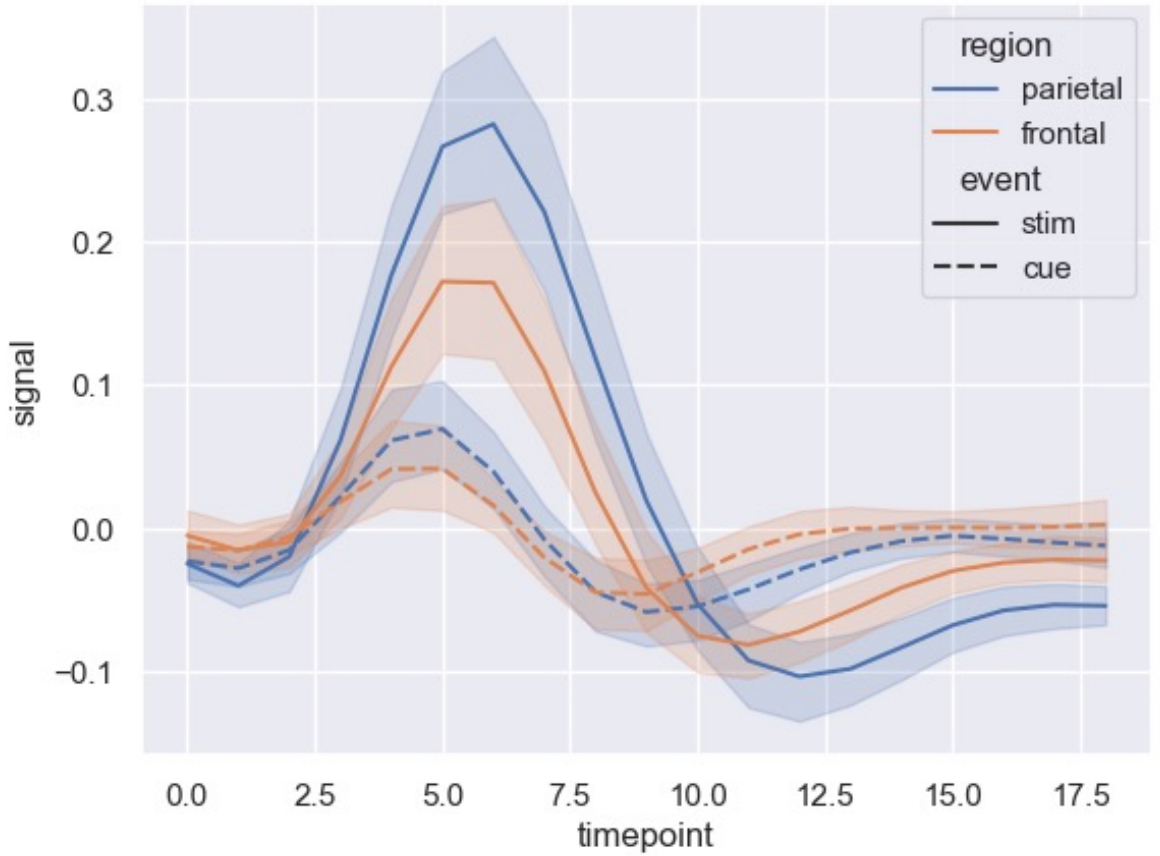
Returns the underlying `PairGrid` instance for further tweaking.

https://seaborn.pydata.org/generated/seaborn.pairplot.html

seaborn : timeseries

Time series graphs are created by plotting an aggregated value (either a count or a statistic, such as sum or average) on a time line.

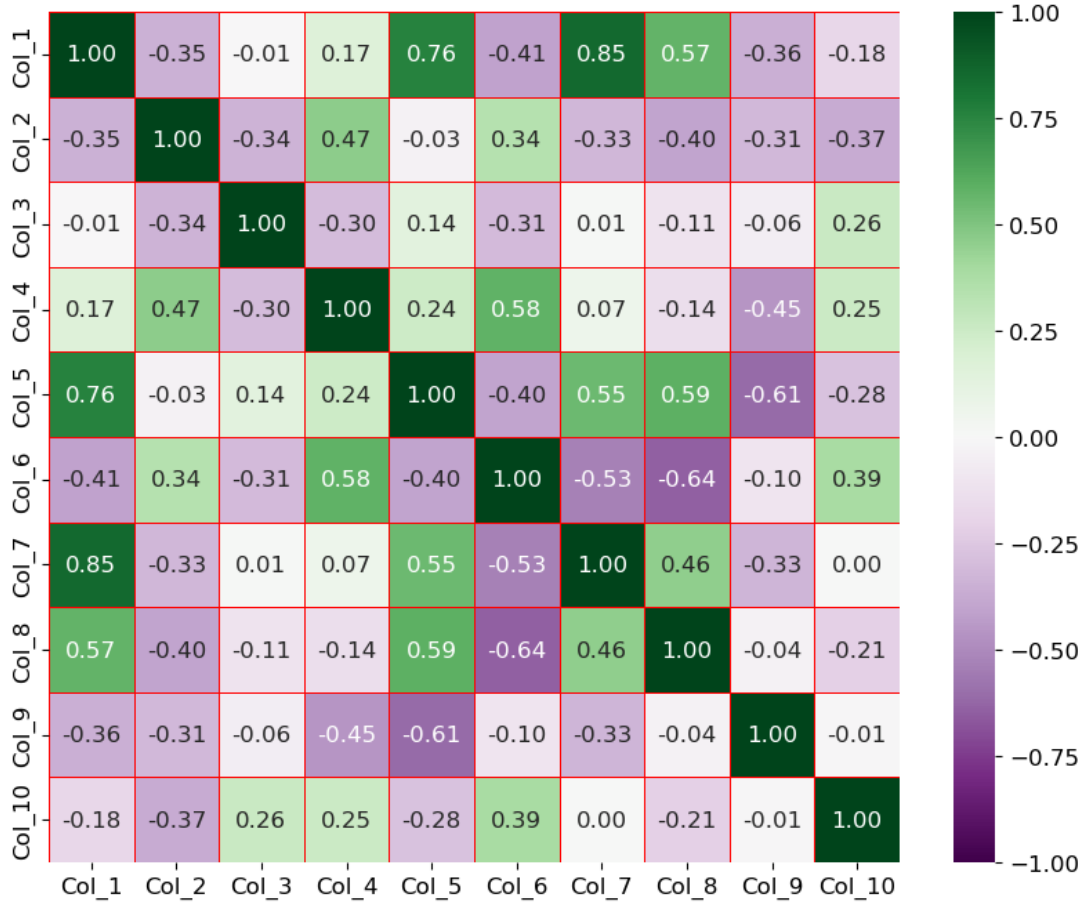
```
sns.lineplot(x="timepoint",
             y="signal",
             hue="region",
             style="event",
             data=fmri)
```



https://seaborn.pydata.org/examples/errorband_lineplots.html

seaborn : Heatmap

seaborn.**heatmap**(*data, *, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs*)



https://seaborn.pydata.org/generated/seaborn.heatmap.html

seaborn : Heatmap

Parameters:

data : *rectangular dataset*

2D dataset that can be coerced into an ndarray. If a Pandas DataFrame is provided, the index/column information will be used to label the columns and rows.

vmin, vmax : *floats, optional*

Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.

cmap : *matplotlib colormap name or object, or list of colors, optional*

The mapping from data values to color space. If not provided, the default will depend on whether `center` is set.

center : *float, optional*

The value at which to center the colormap when plotting divergent data. Using this parameter will change the default `cmap` if none is specified.

robust : *bool, optional*

If True and `vmin` or `vmax` are absent, the colormap range is computed with robust quantiles instead of the extreme values.

annot : *bool or rectangular dataset, optional*

If True, write the data value in each cell. If an array-like with the same shape as `data`, then use this to annotate the heatmap instead of the data. Note that DataFrames will match on position, not index.

fmt : *str, optional*

String formatting code to use when adding annotations.

annot_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.axes.Axes.text()` when `annot` is True.

linewidths : *float, optional*

Width of the lines that will divide each cell.

linecolor : *color, optional*

linecolor : *color, optional*

Color of the lines that will divide each cell.

cbar : *bool, optional*

Whether to draw a colorbar.

cbar_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.figure.Figure.colorbar()`.

cbar_ax : *matplotlib Axes, optional*

Axes in which to draw the colorbar, otherwise take space from the main Axes.

square : *bool, optional*

If True, set the Axes aspect to “equal” so each cell will be square-shaped.

xticklabels, yticklabels : *“auto”, bool, list-like, or int, optional*

If True, plot the column names of the dataframe. If False, don’t plot the column names. If list-like, plot these alternate labels as the xticklabels. If an integer, use the column names but plot only every n label. If “auto”, try to densely plot non-overlapping labels.

mask : *bool array or DataFrame, optional*

If passed, data will not be shown in cells where `mask` is True. Cells with missing values are automatically masked.

ax : *matplotlib Axes, optional*

Axes in which to draw the plot, otherwise use the currently-active Axes.

kwargs : *other keyword arguments*

All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

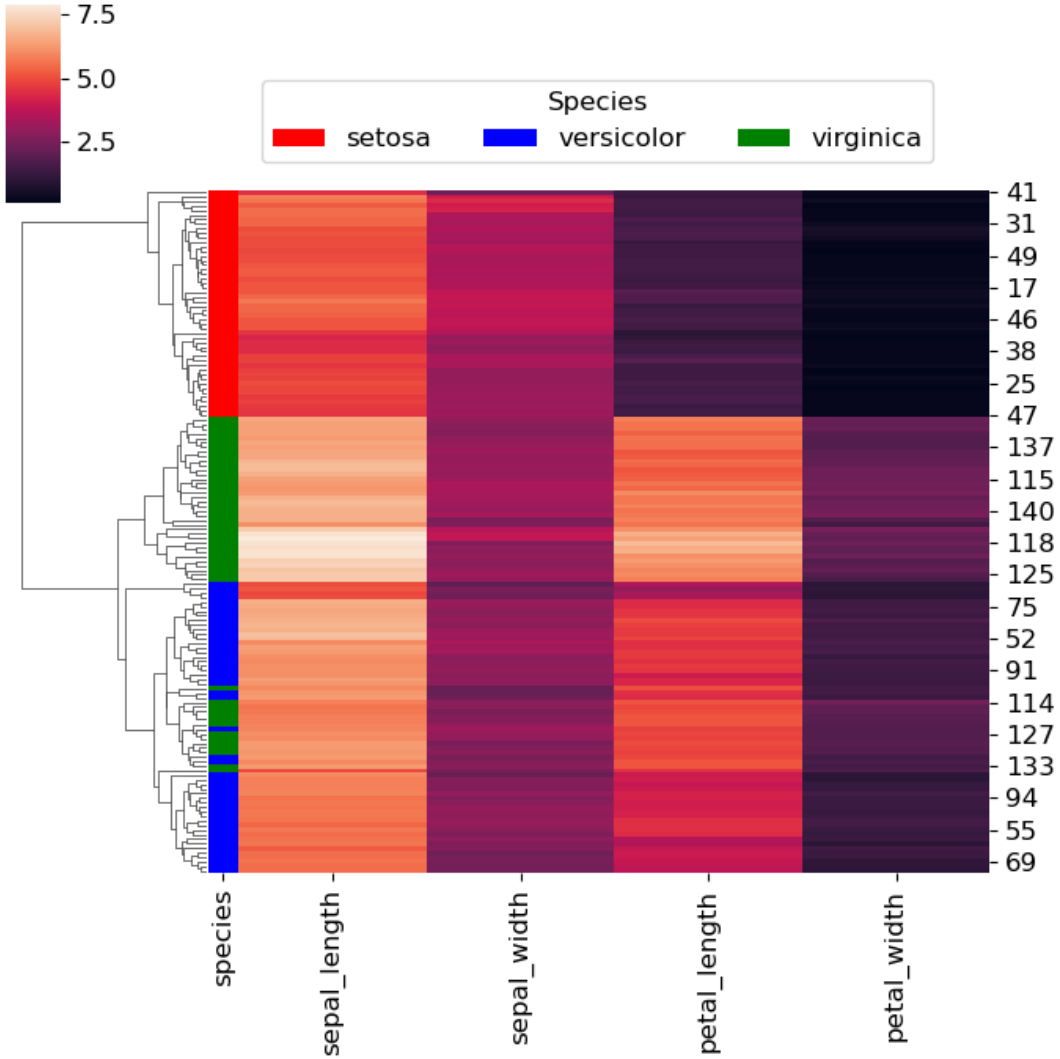
ax : *matplotlib Axes*

https://seaborn.pydata.org/generated/seaborn.heatmap.html

seaborn : clustermap

seaborn.clustermap

seaborn.**clustermap**(*data*, *, *pivot_kws=None*, *method='average'*, *metric='euclidean'*, *z_score=None*, *standard_scale=None*, *figsize=(10, 10)*, *cbar_kws=None*, *row_cluster=True*, *col_cluster=True*, *row_linkage=None*, *col_linkage=None*, *row_colors=None*, *col_colors=None*, *mask=None*, *dendrogram_ratio=0.2*, *colors_ratio=0.03*, *cbar_pos=(0.02, 0.8, 0.05, 0.18)*, *tree_kws=None*, ***kwargs*)



https://seaborn.pydata.org/generated/seaborn.clustermap.html

seaborn : clustermap

Parameters:

data : *2D array-like*

Rectangular data for clustering. Cannot contain NAs.

pivot_kws : *dict, optional*

If `data` is a tidy dataframe, can provide keyword arguments for pivot to create a rectangular dataframe.

method : *str, optional*

Linkage method to use for calculating clusters. See `scipy.cluster.hierarchy.linkage()` documentation for more information.

metric : *str, optional*

Distance metric to use for the data. See `scipy.spatial.distance.pdist()` documentation for more options. To use different metrics (or methods) for rows and columns, you may construct each linkage matrix yourself and provide them as `{row,col}_linkage`.

z_score : *int or None, optional*

Either 0 (rows) or 1 (columns). Whether or not to calculate z-scores for the rows or the columns. Z scores are: z = (x - mean)/std, so values in each row (column) will get the mean of the row (column) subtracted, then divided by the standard deviation of the row (column). This ensures that each row (column) has mean of 0 and variance of 1.

standard_scale : *int or None, optional*

Either 0 (rows) or 1 (columns). Whether or not to standardize that dimension, meaning for each row or column, subtract the minimum and divide each by its maximum.

figsize : *tuple of (width, height), optional*

Overall size of the figure.

cbar_kws : *dict, optional*

Keyword arguments to pass to `cbar_kws` in `heatmap()`, e.g. to add a label to the colorbar.

{row,col}_cluster : *bool, optional*

If `True`, cluster the {rows, columns}.

{row,col}_linkage : *`numpy.ndarray`, optional*

Precomputed linkage matrix for the rows or columns. See `scipy.cluster.hierarchy.linkage()` for specific formats.

{row,col}_colors : *list-like or pandas DataFrame/Series, optional*

List of colors to label for either the rows or columns. Useful to evaluate whether samples within a group are clustered together. Can use nested lists or DataFrame for multiple color levels of labeling. If given as a `pandas.DataFrame` or `pandas.Series`, labels for the colors are extracted from the DataFrames column names or from the name of the Series. DataFrame/Series colors are also matched to the data by their index, ensuring colors are drawn in the correct order.

mask : *bool array or DataFrame, optional*

If passed, data will not be shown in cells where `mask` is True. Cells with missing values are automatically masked. Only used for visualizing, not for calculating.

{dendrogram,colors}_ratio : *float, or pair of floats, optional*

Proportion of the figure size devoted to the two marginal elements. If a pair is given, they correspond to (row, col) ratios.

cbar_pos : *tuple of (left, bottom, width, height), optional*

Position of the colorbar axes in the figure. Setting to `None` will disable the colorbar.

tree_kws : *dict, optional*

Parameters for the `matplotlib.collections.LineCollection` that is used to plot the lines of the dendrogram tree.

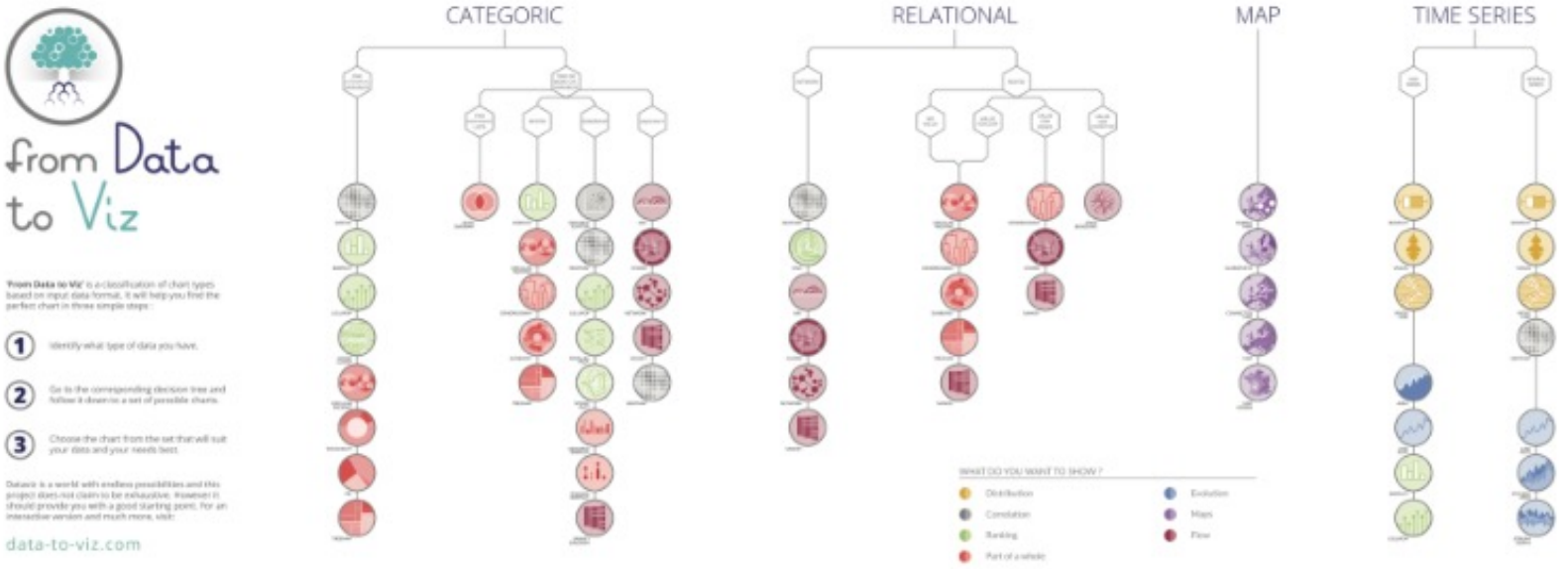
kwargs : *other keyword arguments*

Hands-on Examples … Self Practice

- **https://www.data-to-viz.com/**

Choose your own data type

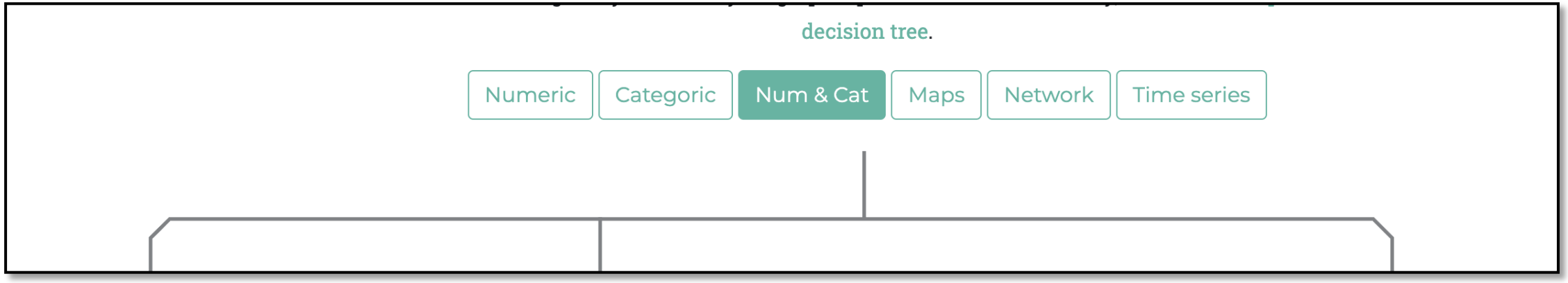
Explore the data types visualizations?



https://www.data-to-viz.com/

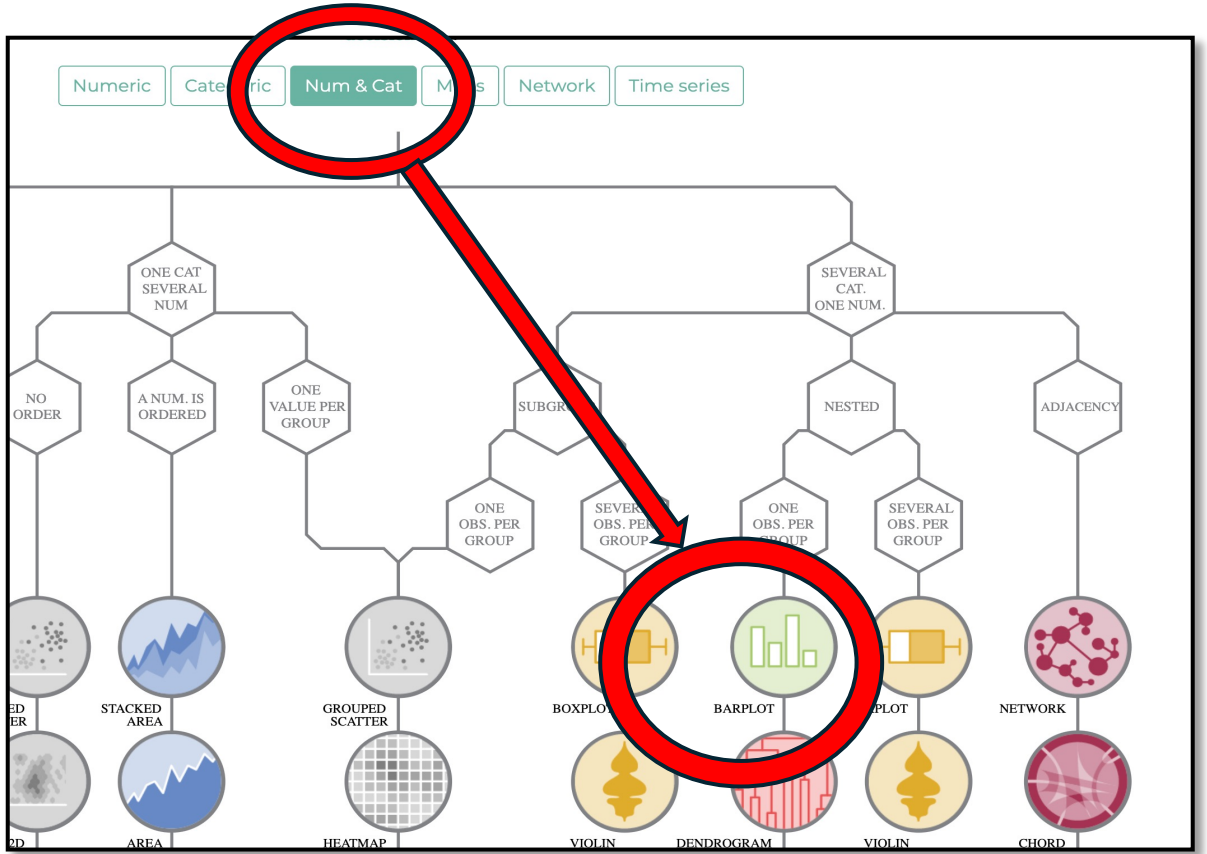
Steps:

1. Go to Website: https://www.data-to-viz.com/
2. Select the category of your choice…



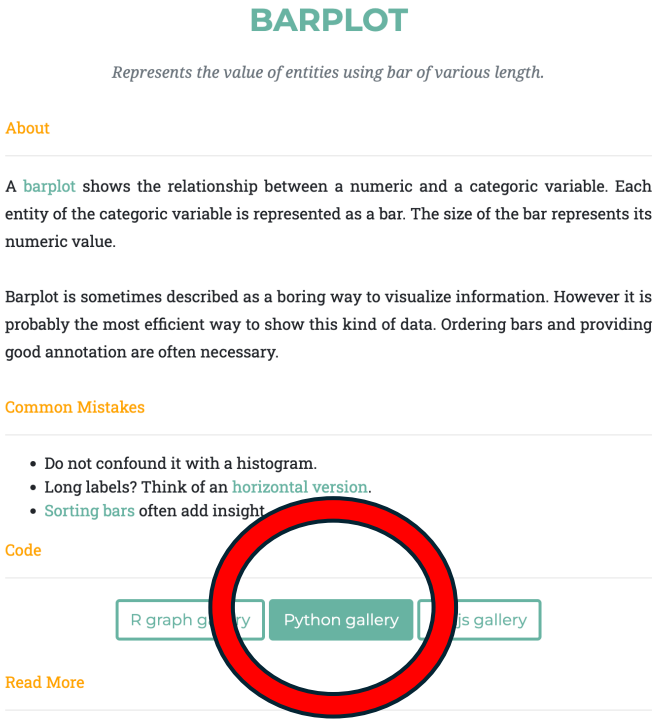
Steps:

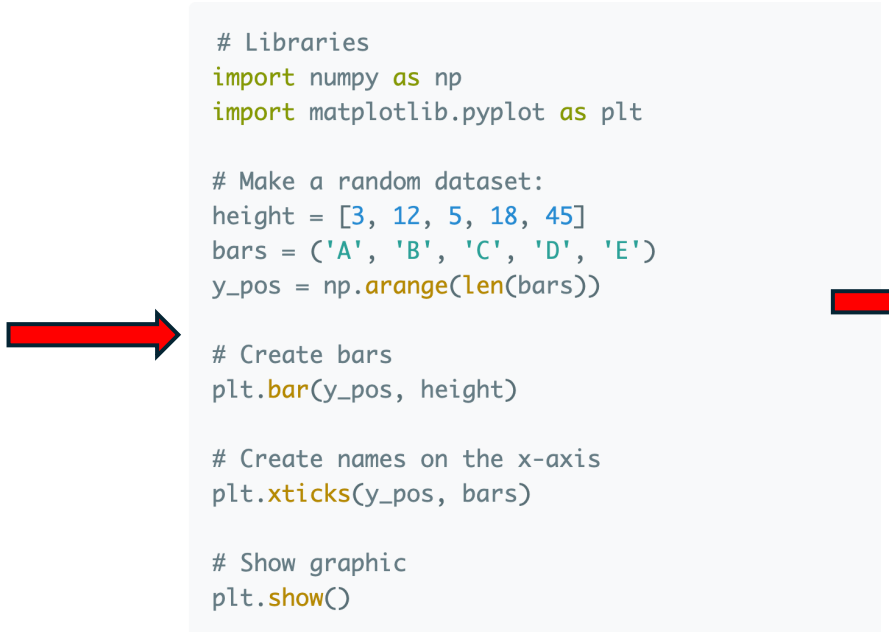
1. Go to Website: https://www.data-to-viz.com/
2. Select the category of your choice…
3. Select your plot e.g. **NUM & CAT → BARPLOT**



Steps:

1. Go to Website: https://www.data-to-viz.com/
2. Select the category of your choice…
3. Select your plot e.g. **NUM & CAT → BARPLOT**
4. Click on it → Go to code in python & replicate





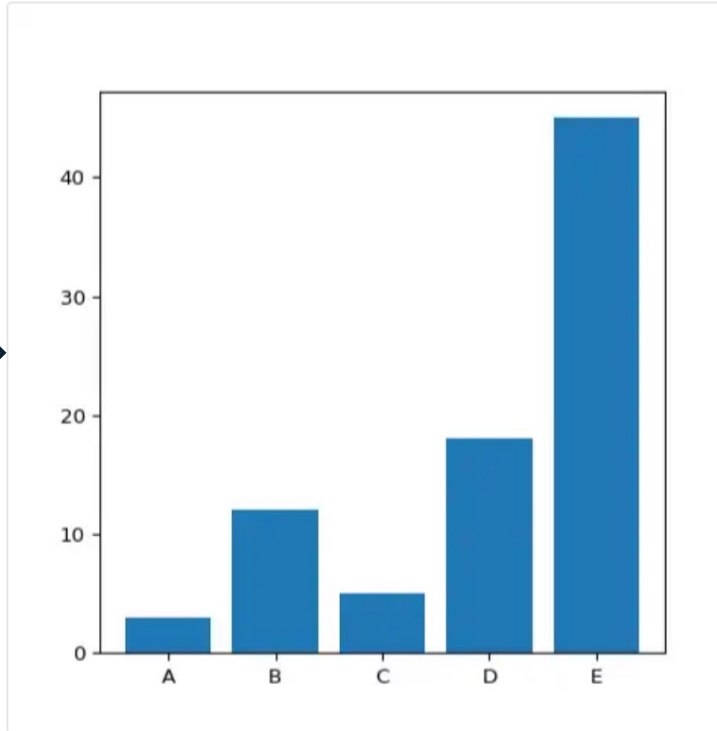
```
# Libraries
import numpy as np
import matplotlib.pyplot as plt

# Make a random dataset:
height = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
y_pos = np.arange(len(bars))

# Create bars
plt.bar(y_pos, height)

# Create names on the x-axis
plt.xticks(y_pos, bars)

# Show graphic
plt.show()
```



Additional sources: Best Practices of data visualization

- https://infogram.com/blog/good-data-visualization-examples/
- https://blog.csgsolutions.com/6-tips-for-creating-effective-data-visualizations
- https://www.maptive.com/data-visualization-examples/
- https://www.tableau.com/learn/articles/best-beautiful-data-visualization-examples
- https://www.polymersearch.com/blog/10-good-and-bad-examples-of-data-visualization
- https://www.analyticsvidhya.com/blog/2023/06/data-visualization-examples/

Thank you!

Reach me @

- Showkat.dar@nih.gov
- Showkat.dar19@gmail.com
- Mob: 443-248-8491
- B –Wing, 10th Floor, LGG, NIA.