

Data Visualizatinn

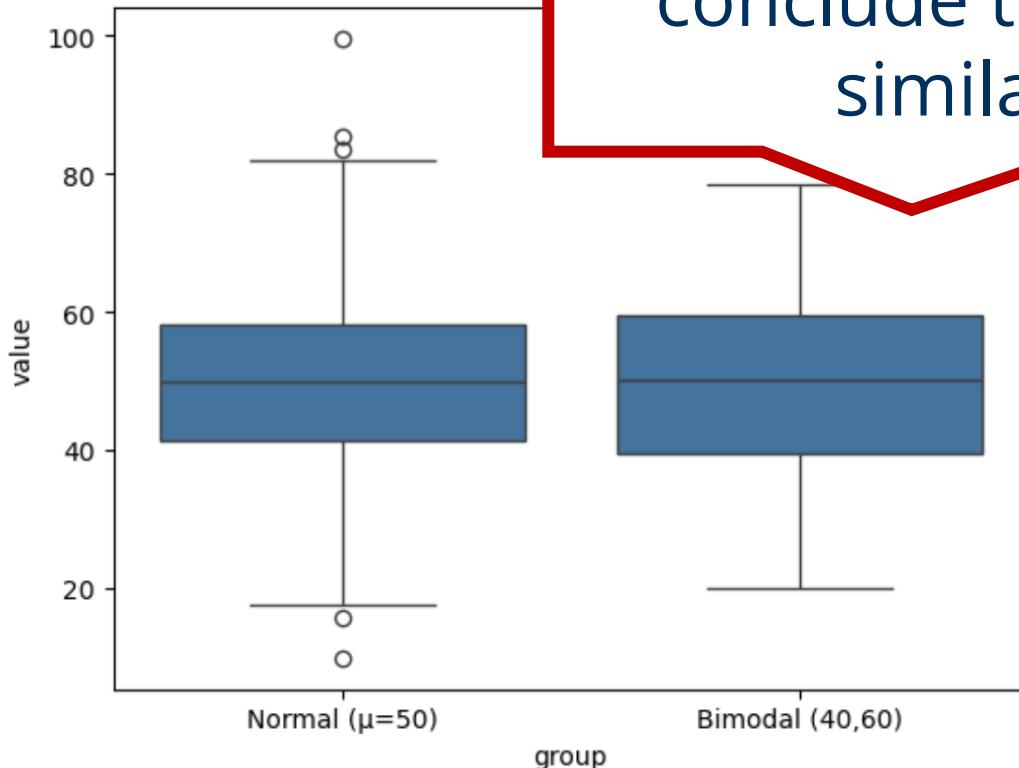
Robert Haase

Reusing Materials from Jan Ewald and Mara Lampert (ScaDS.AI, Uni Leipzig)

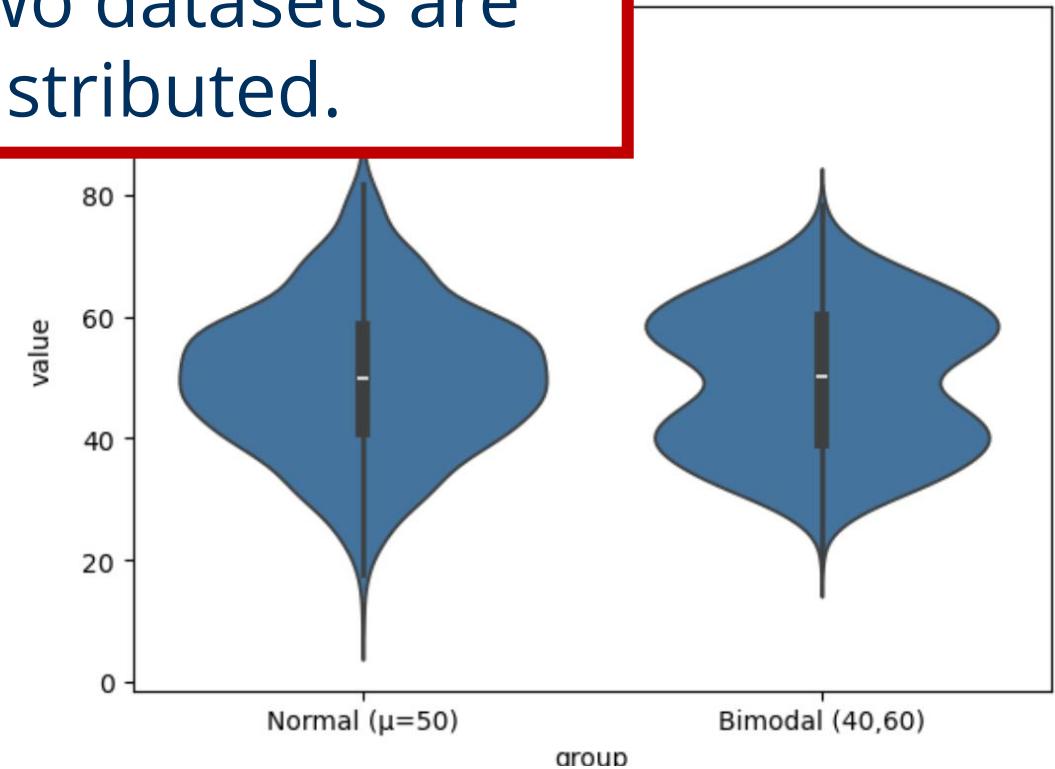
These slides can be reused under the terms of the [CC-BY 4.0](#) license unless mentioned otherwise.

Take home message: choose plots wisely

From such a visualization you may conclude that two datasets are similarly distributed.

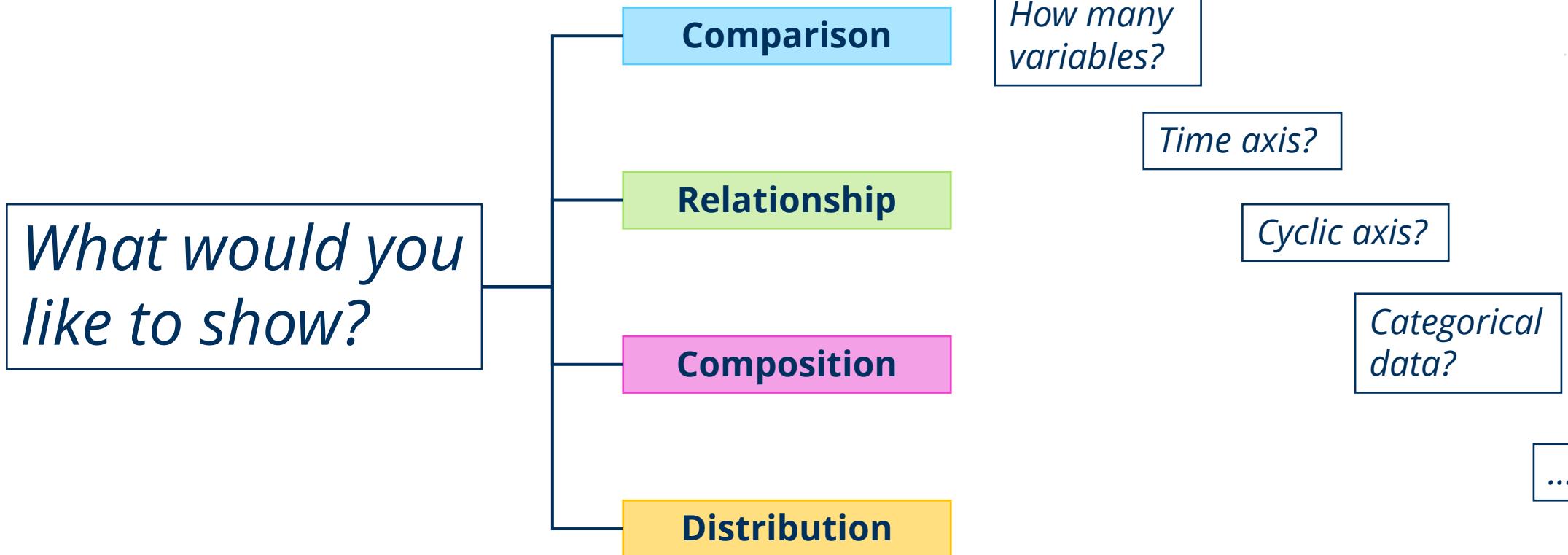


```
sns.boxplot(data=df, x="group", y="value")
```

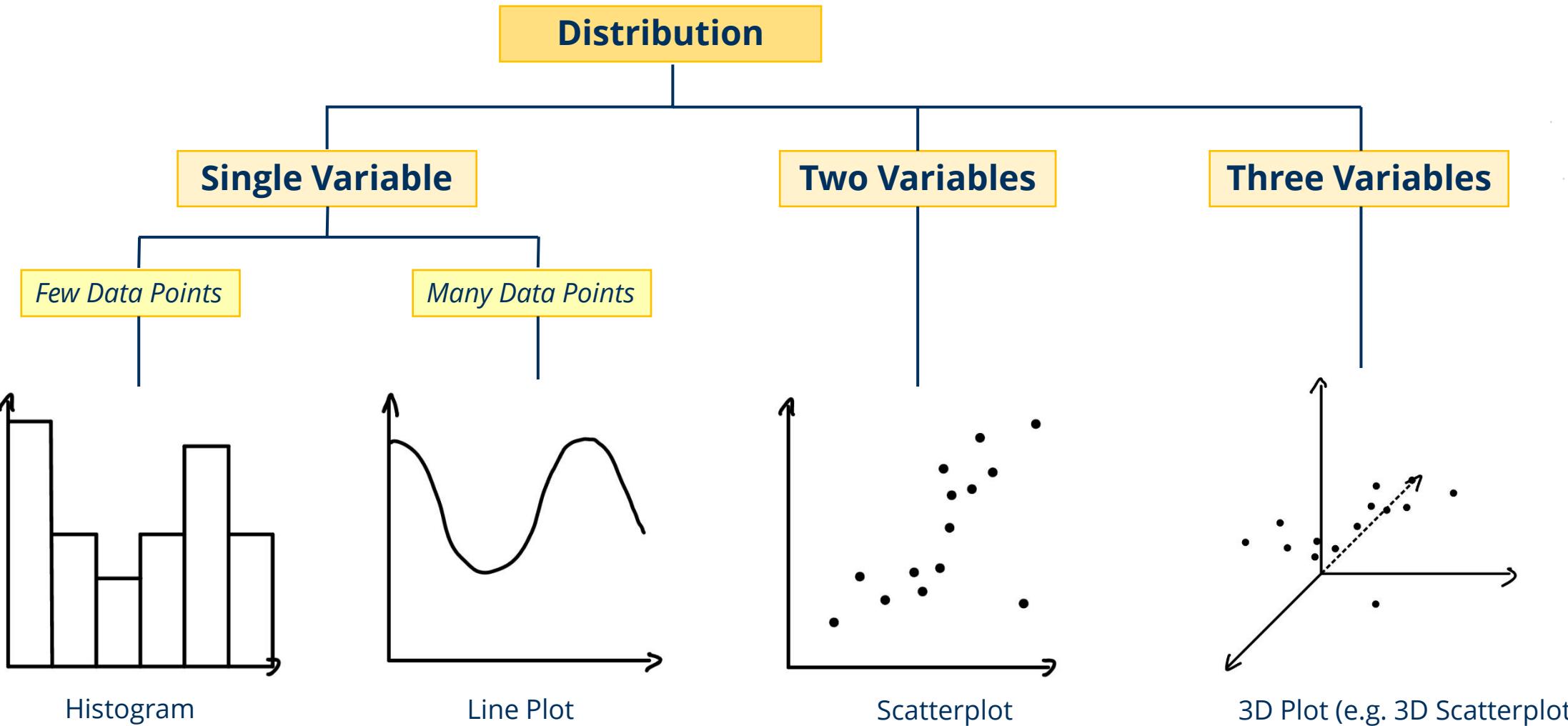


```
sns.violinplot(data=df, x="group", y="value")
```

First things first.



Distribution of data



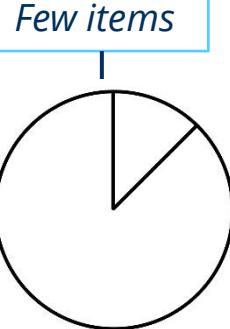
Comparing data

Comparison

Among items

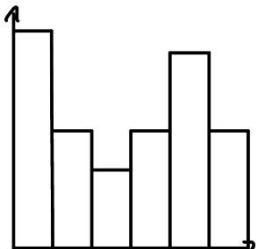
One variable

Few categories



Pie Chart

Many categories



Bar Plot

Two variables

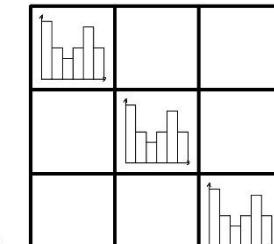
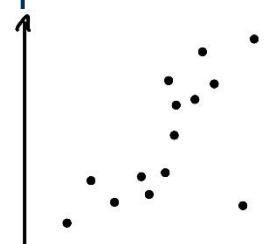


Table (with Plots)

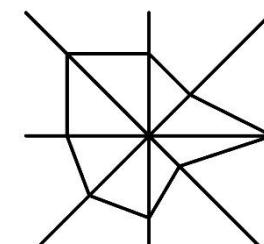


Scatterplot

Over time

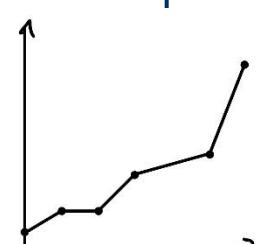
Many timepoints

Cyclical Data



Spiderplot

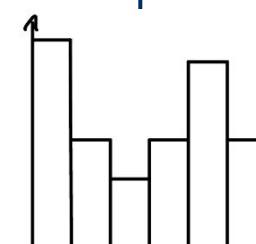
Non-Cyclical Data



Line Plot

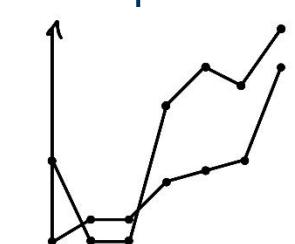
Few timepoints

few categories



Bar Plot

Many categories



Line Plot

Composition of data

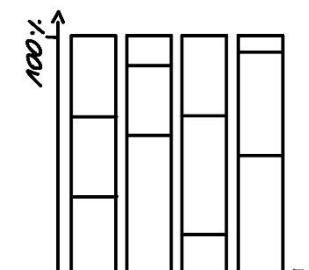
Composition

Changing Over Time

Few Periods

Only Relative Differences Matter

Relative and Absolute Differences Matter

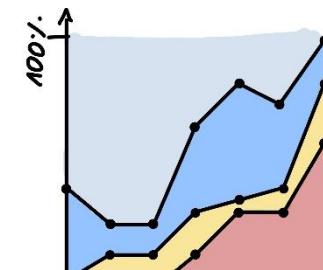


Stacked 100% Column Chart

Many Periods

Only Relative Differences Matter

Relative and Absolute Differences Matter



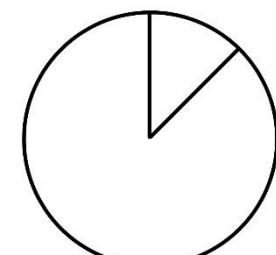
Stacked 100% Area Chart

Static

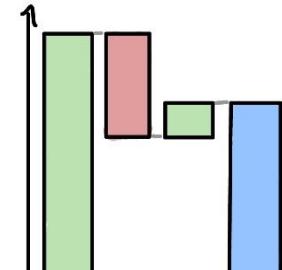
Simple Share of Total

Accumulation or Subtraction to Total

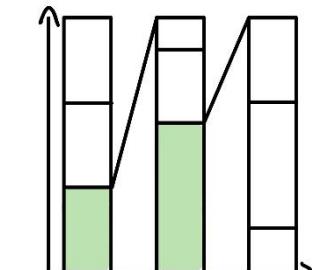
Components of Components



Pie Chart

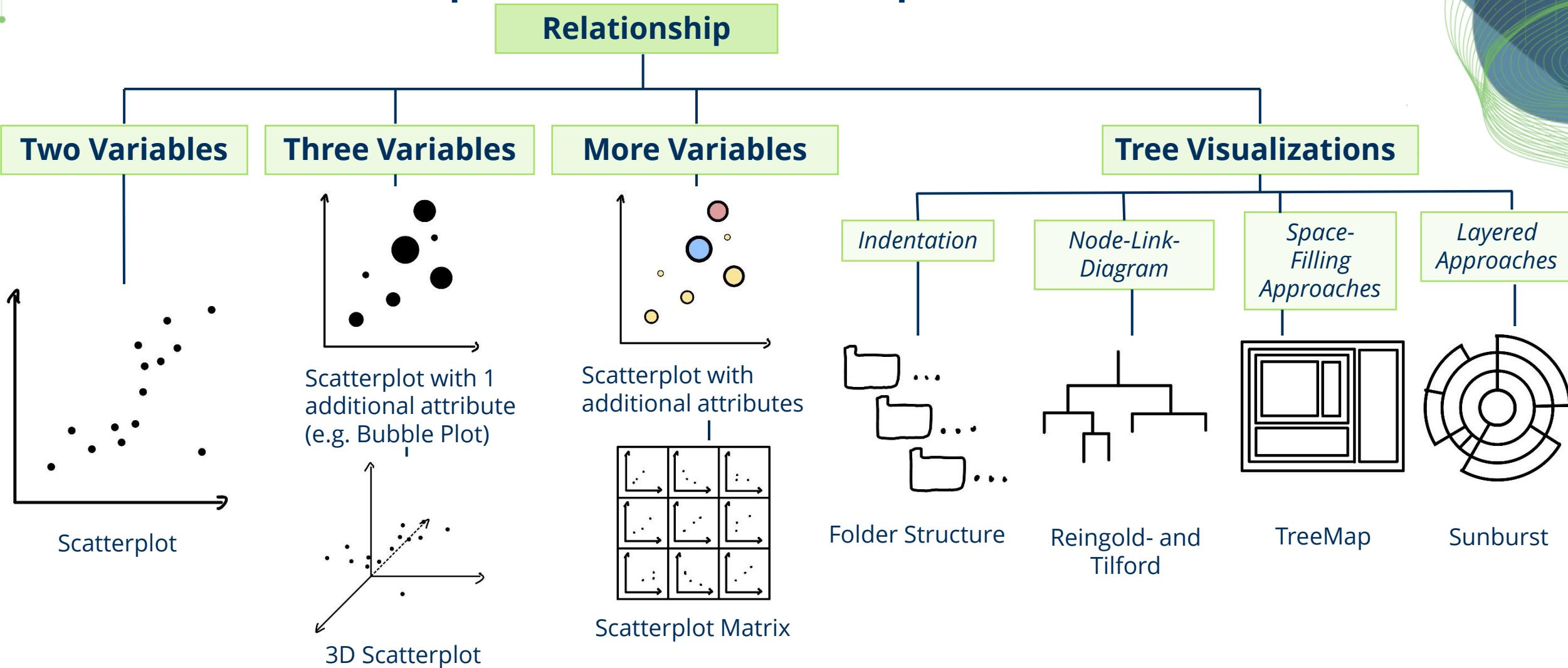


Waterflow Chart



Stacked 100% Column Chart with Subcomponents

Relationships between aspects of data

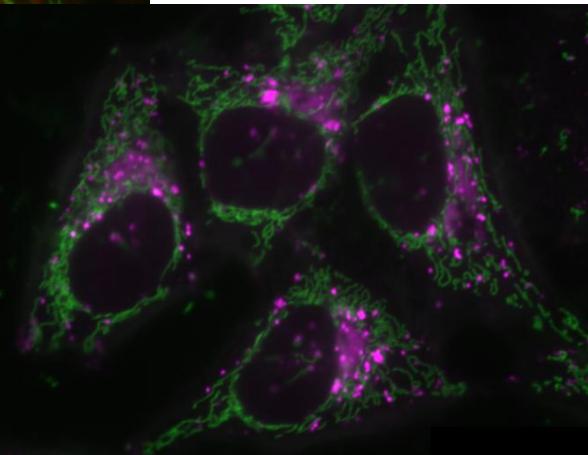
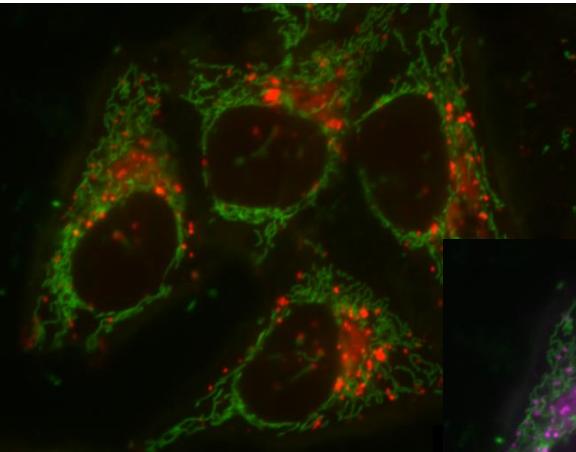


Color maps / lookup tables

Choose visualization of your color tables wisely!

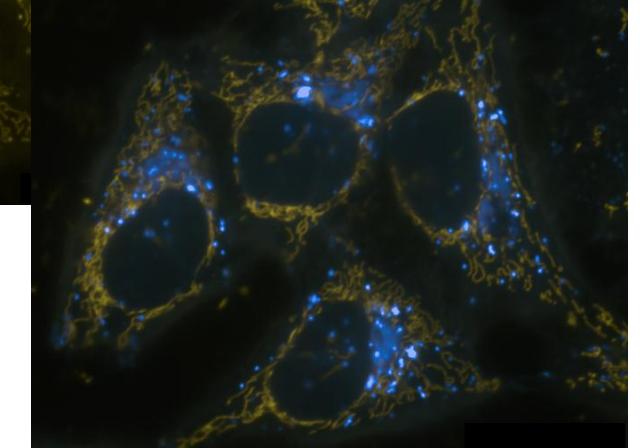
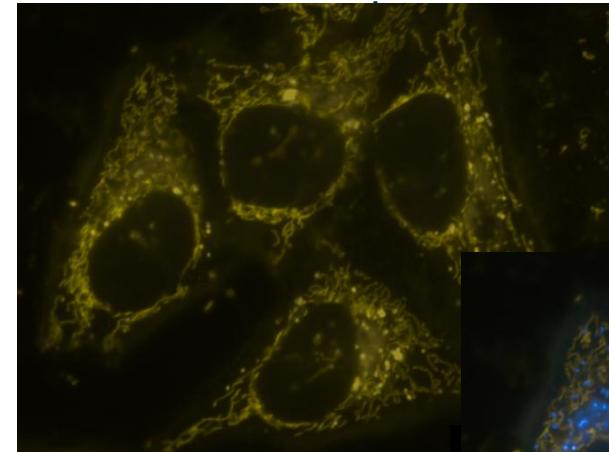
Think of people with red/green blindness!

Default view



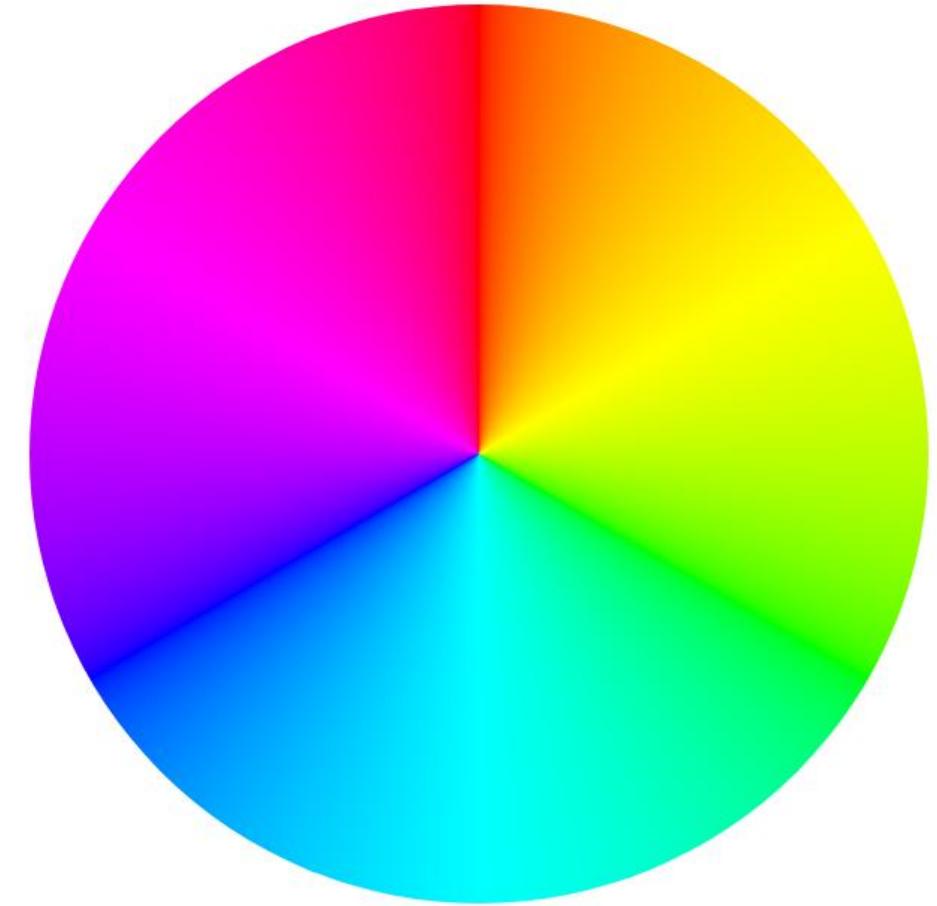
Replace red
with magenta!

Red/green blind people see it like



Colour theory

- Humans may not see the difference between adjacent colors properly
- Use opposite colours instead.



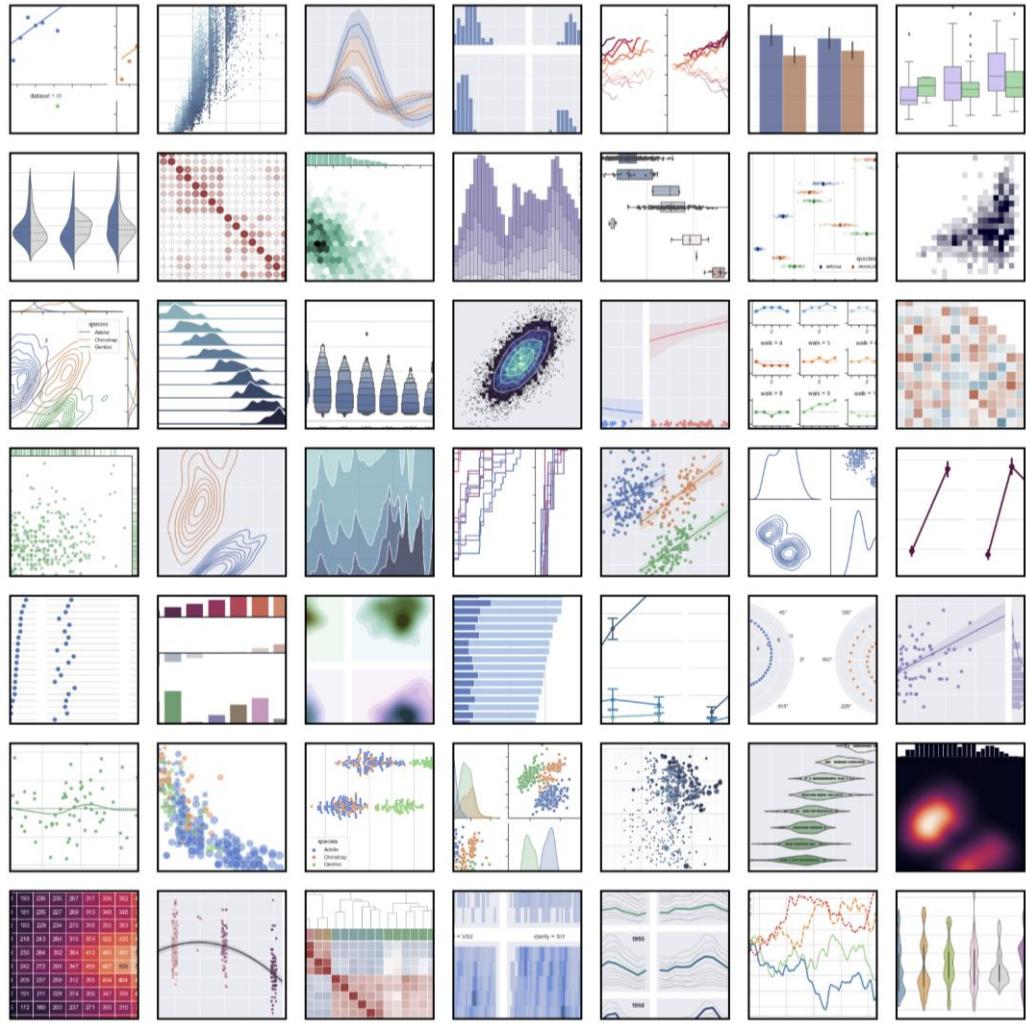
Data visualization using seaborn

Seaborn is a community standard library for advanced data visualization, based on matplotlib.

We strongly discourage data scientists from using plain matplotlib.

<https://seaborn.pydata.org/tutorial/introduction.html>

<https://seaborn.pydata.org/examples/index.html>

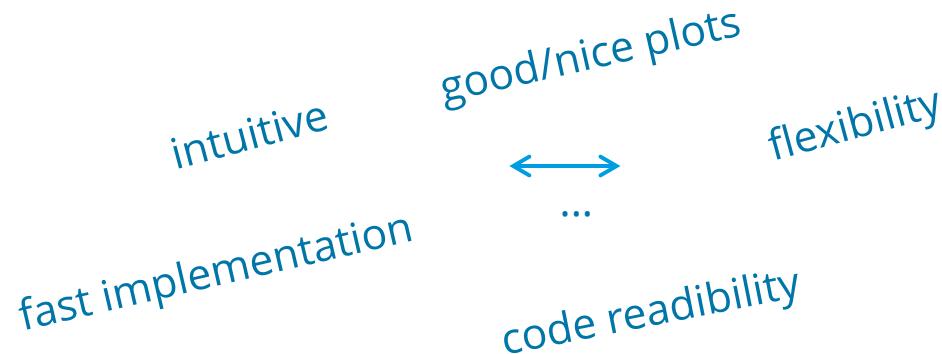


API comparison & philosophies



seaborn

seaborn.objects



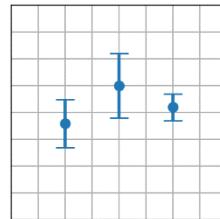
Plot type driven ← → **Data and geometry driven**

API comparison & philosophies

Data:

| | | |
|--|----------|----------|
| | <i>a</i> | <i>b</i> |
| | | |
| | | |
| | | |

Plot:



```
x=data[‘a’]  
y=data[‘b’]  
yerr = sd(y)  
  
errorbar(x,y, yerr)
```



seaborn

```
pointplot(  
    data,  
    x=‘a’, y=‘b’,  
    errorbar=‘sd’  
)
```

seaborn.objects

```
so.Plot(data,x=‘a’, y=‘b’)  
.add( so.Dot(),  
      so.Agg())  
.add( so.Range(),  
      so.Est(errorbar=‘sd’))
```

- 1) Choose plot type
- 2) Extract or calculate variables
- 3) Stuff into plot API

- 1) Choose basic type
- 2) Define data and variables

- 1) Define data and variables
- 2) Choose plot geometries
- 3) Define statistics

Plot type driven ←

→ **Data and geometry driven**

Concepts behind seaborn

Figure-level



Axes-level

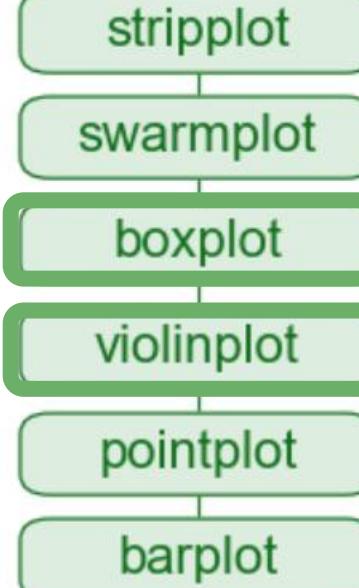
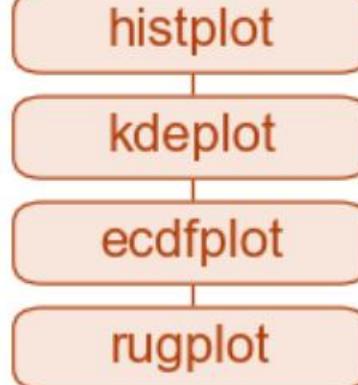
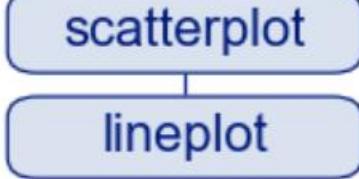
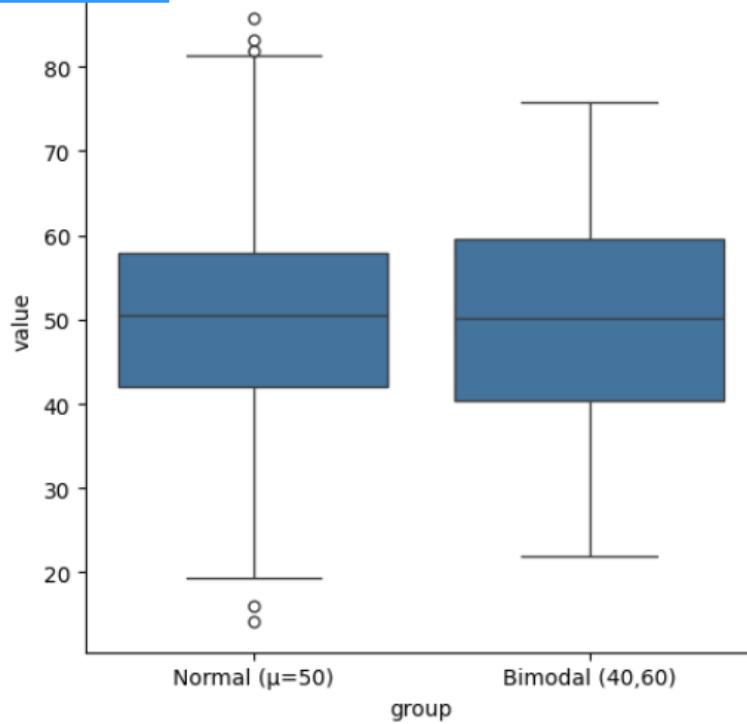


Figure level versus axes level

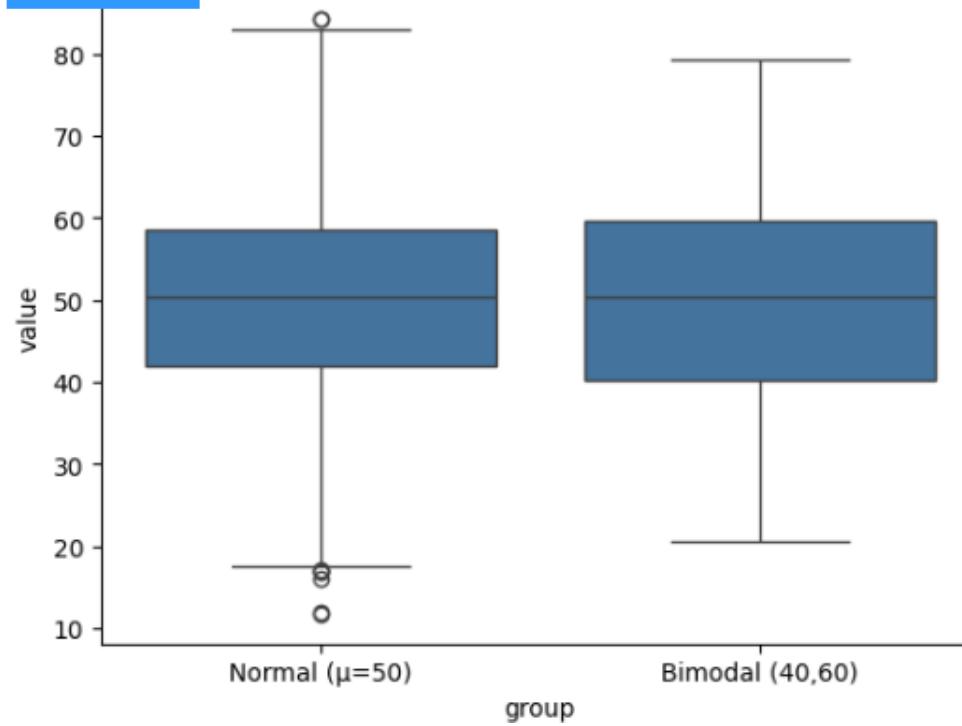
```
sns.catplot(data=df,  
            x="group",  
            y="value",  
            kind="box");
```

Figure
level



```
sns.boxplot(data=df,  
            x="group",  
            y="value");
```

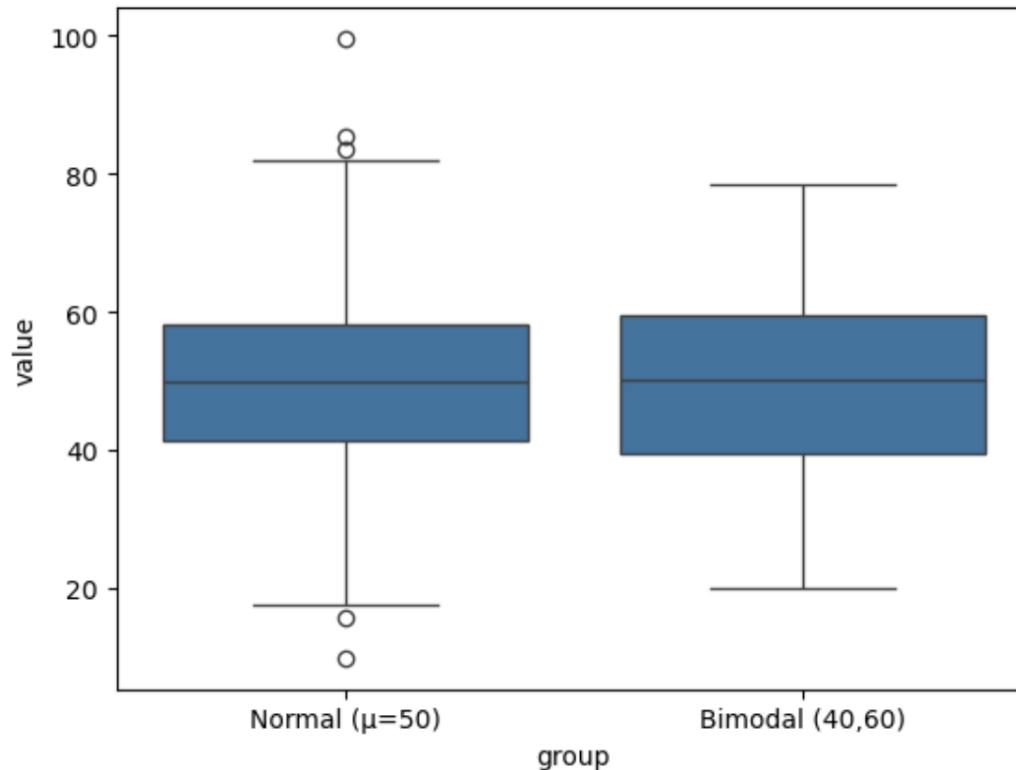
Axes
level



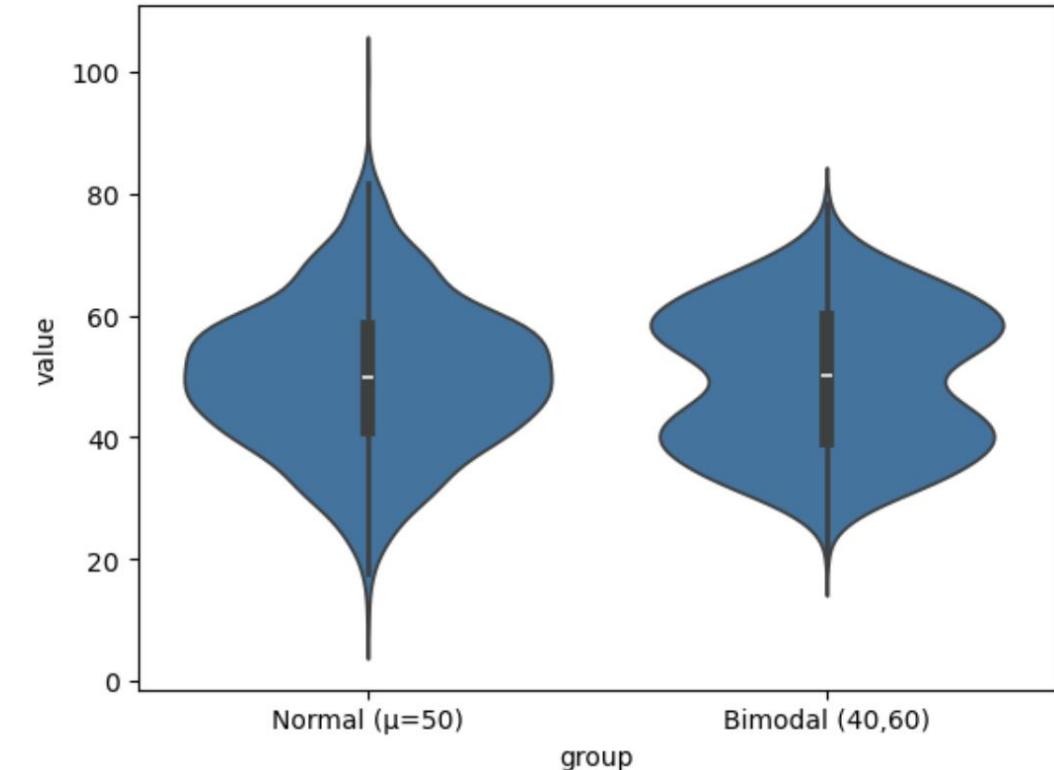
Comparison based on categories

Easy to switch plot type

~~sns.boxplot(data=df, x="group", y="value")~~



sns.violinplot(data=df, x="group", y="value")



Concepts behind seaborn

Figure-level

relplot
(relational)

scatterplot

lineplot

displot
(distributions)

histplot

kdeplot

ecdfplot

rugplot

catplot
(categorical)

stripplot

swarmplot

boxplot

violinplot

pointplot

barplot

Axes-level

Distributions

Figure
level

```
sns.histplot(data=data, x='value', hue='series')
```

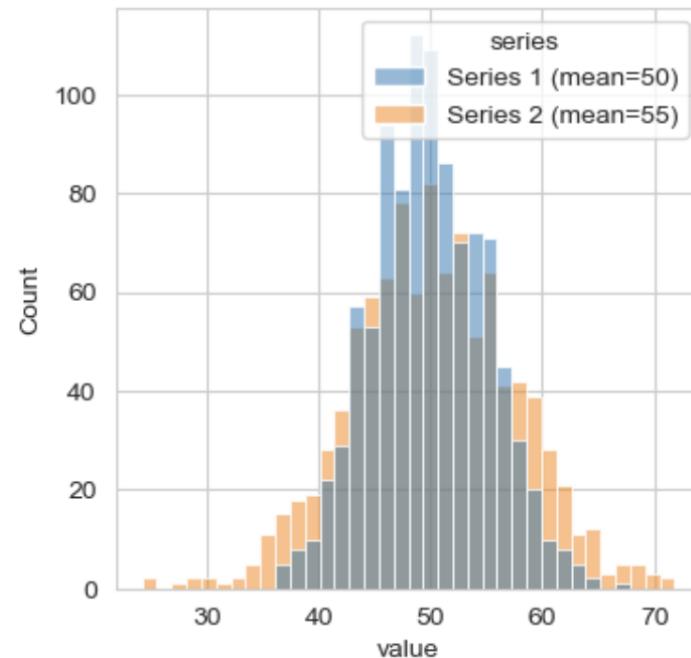
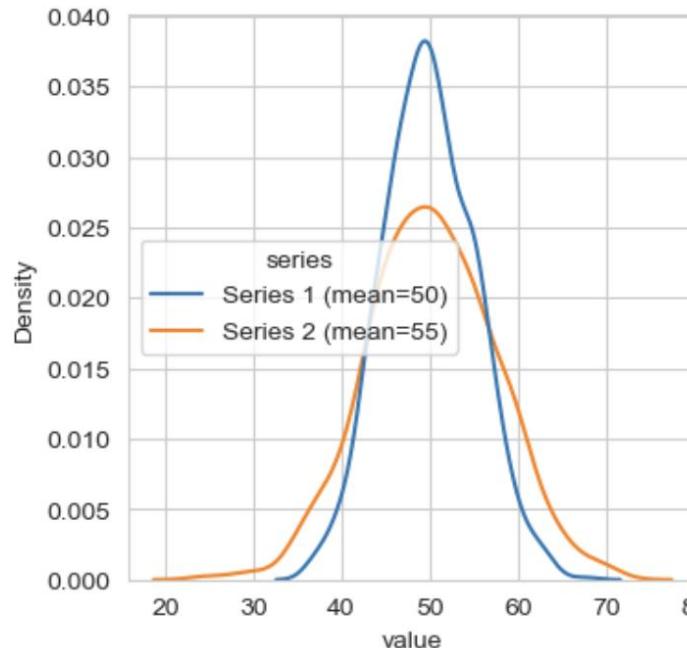
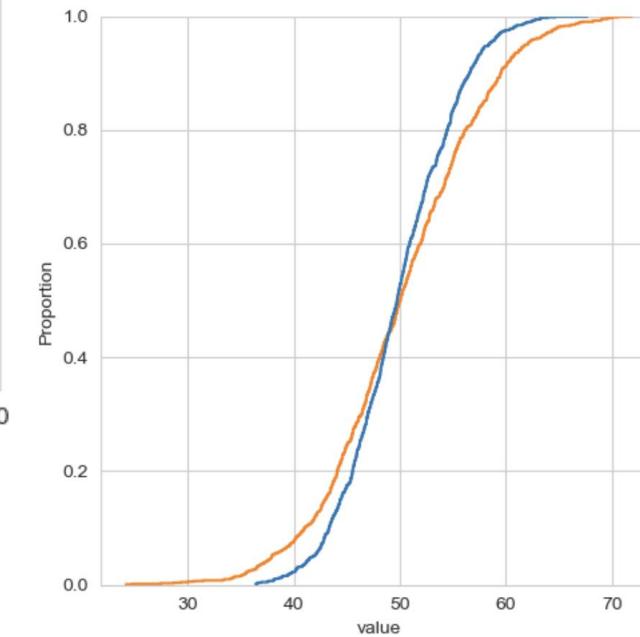


Figure
level

```
sns.kdeplot(data=data, x='value', hue='series')
```



```
sns.displot(data=data, x='value',  
hue='series', kind="ecdf")
```



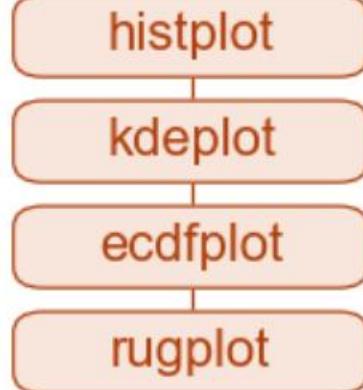
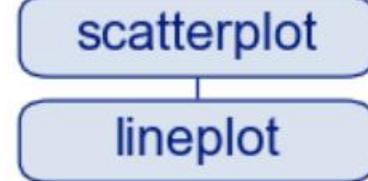
Axes
level

Concepts behind seaborn

Figure-level

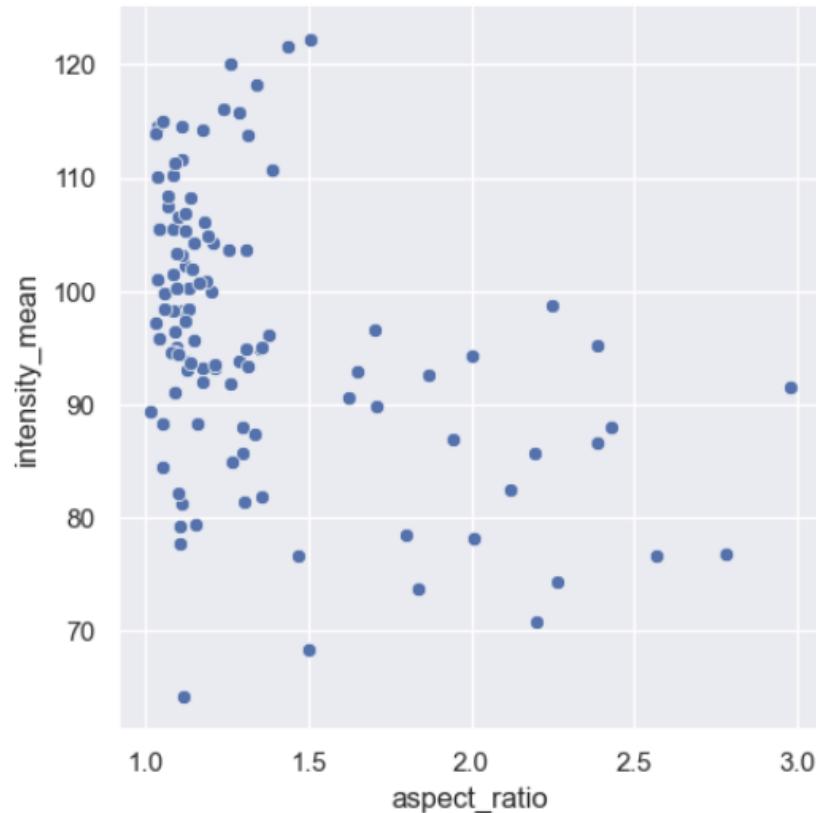


Axes-level



Relationships

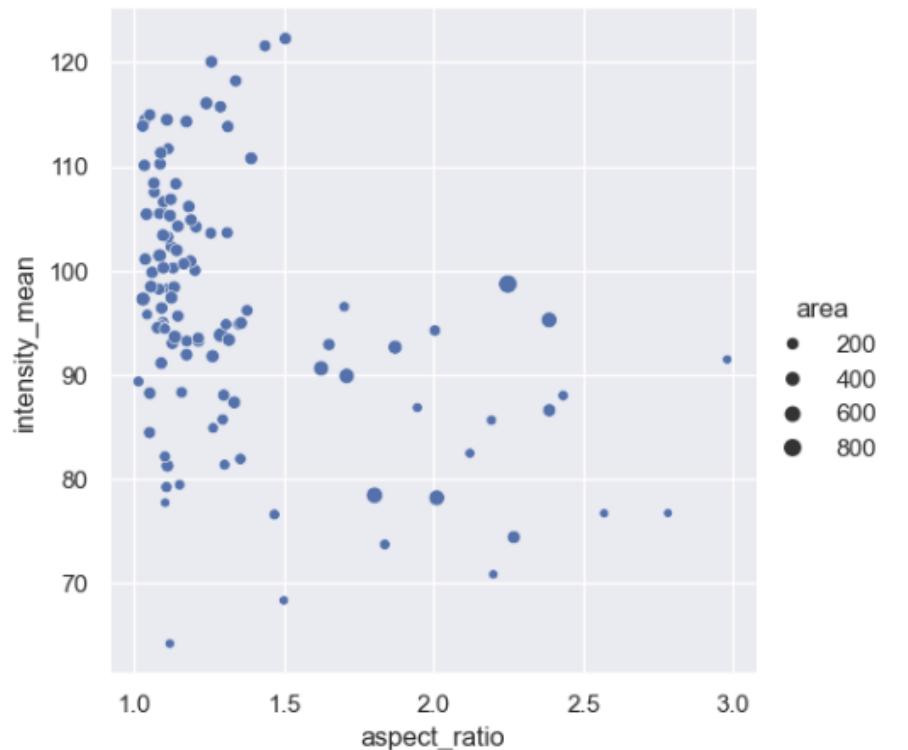
```
sns.relplot(data=df,  
            x="aspect_ratio",  
            y="intensity_mean");
```



| area | intensity_mean | major_axis_length | minor_axis_length | aspect_ratio | file_name |
|------|----------------|-------------------|-------------------|--------------|--------------------|
| 139 | 96.546763 | 17.504104 | 10.292770 | 1.700621 | 20P1_POS0010_D_1UL |
| 360 | 86.613889 | 35.746808 | 14.983124 | 2.385805 | 20P1_POS0010_D_1UL |
| 43 | 91.488372 | 12.967884 | 4.351573 | 2.980045 | 20P1_POS0010_D_1UL |
| 140 | 73.742857 | 18.940508 | 10.314404 | 1.836316 | 20P1_POS0010_D_1UL |
| 144 | 89.375000 | 13.639308 | 13.458532 | 1.013432 | 20P1_POS0010_D_1UL |

Relationships

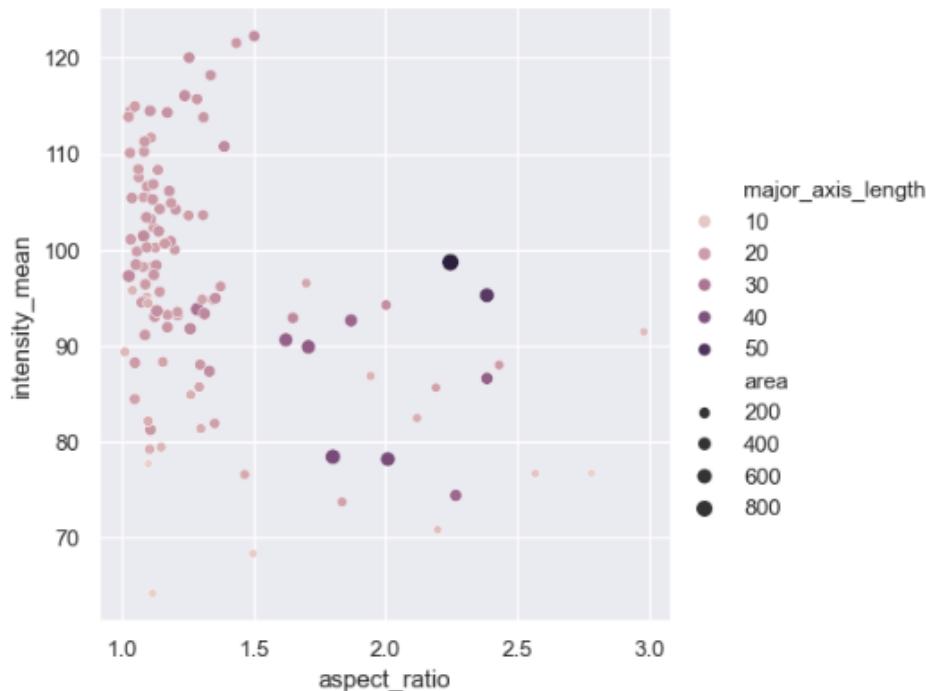
```
sns.relplot(data=df,  
            x="aspect_ratio",  
            y="intensity_mean",  
            size="area");
```



| area | intensity_mean | major_axis_length | minor_axis_length | aspect_ratio | file_name |
|------|----------------|-------------------|-------------------|--------------|--------------------|
| 139 | 96.546763 | 17.504104 | 10.292770 | 1.700621 | 20P1_POS0010_D_1UL |
| 360 | 86.613889 | 35.746808 | 14.983124 | 2.385805 | 20P1_POS0010_D_1UL |
| 43 | 91.488372 | 12.967884 | 4.351573 | 2.980045 | 20P1_POS0010_D_1UL |
| 140 | 73.742857 | 18.940508 | 10.314404 | 1.836316 | 20P1_POS0010_D_1UL |
| 144 | 89.375000 | 13.639308 | 13.458532 | 1.013432 | 20P1_POS0010_D_1UL |

Relationships

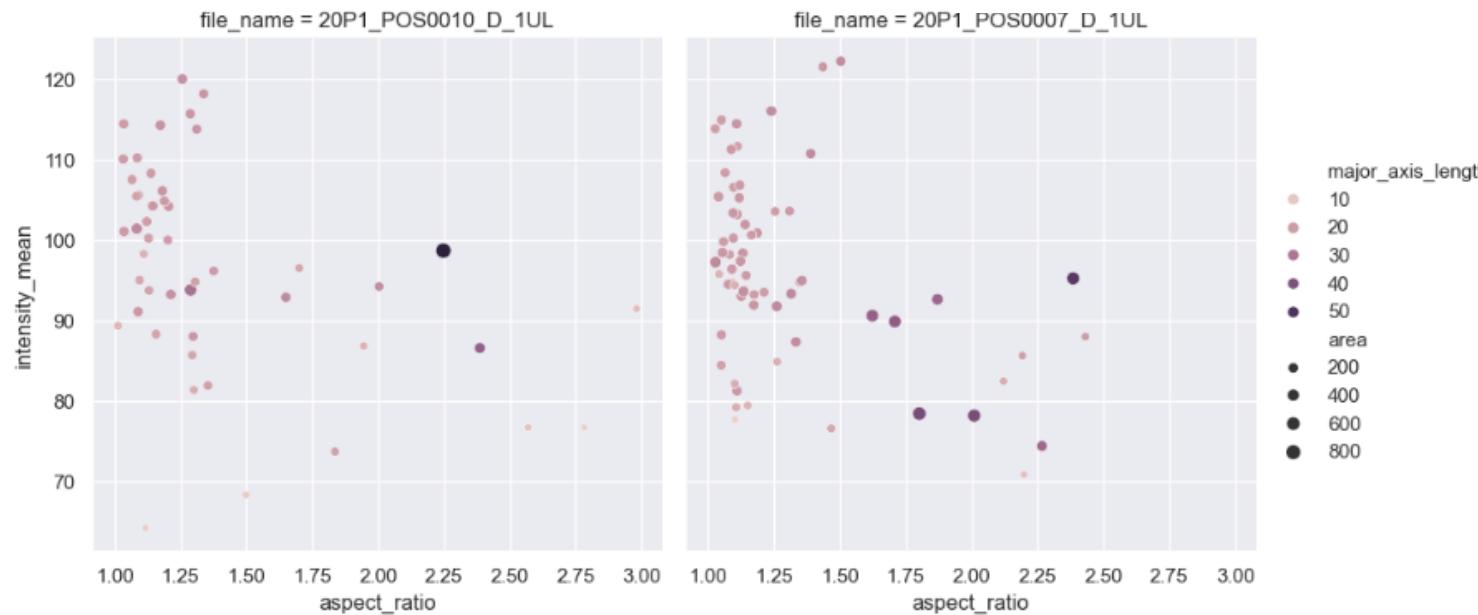
```
sns.relplot(data=df,  
            x="aspect_ratio",  
            y="intensity_mean",  
            size="area",  
            hue="major_axis_length");
```



| area | intensity_mean | major_axis_length | minor_axis_length | aspect_ratio | file_name |
|------|----------------|-------------------|-------------------|--------------|--------------------|
| 139 | 96.546763 | 17.504104 | 10.292770 | 1.700621 | 20P1_POS0010_D_1UL |
| 360 | 86.613889 | 35.746808 | 14.983124 | 2.385805 | 20P1_POS0010_D_1UL |
| 43 | 91.488372 | 12.967884 | 4.351573 | 2.980045 | 20P1_POS0010_D_1UL |
| 140 | 73.742857 | 18.940508 | 10.314404 | 1.836316 | 20P1_POS0010_D_1UL |
| 144 | 89.375000 | 13.639308 | 13.458532 | 1.013432 | 20P1_POS0010_D_1UL |

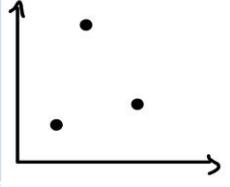
Relationships

```
sns.relplot(data=df,  
            x="aspect_ratio",  
            y="intensity_mean",  
            size="area",  
            hue="major_axis_length",  
            col="file_name");
```



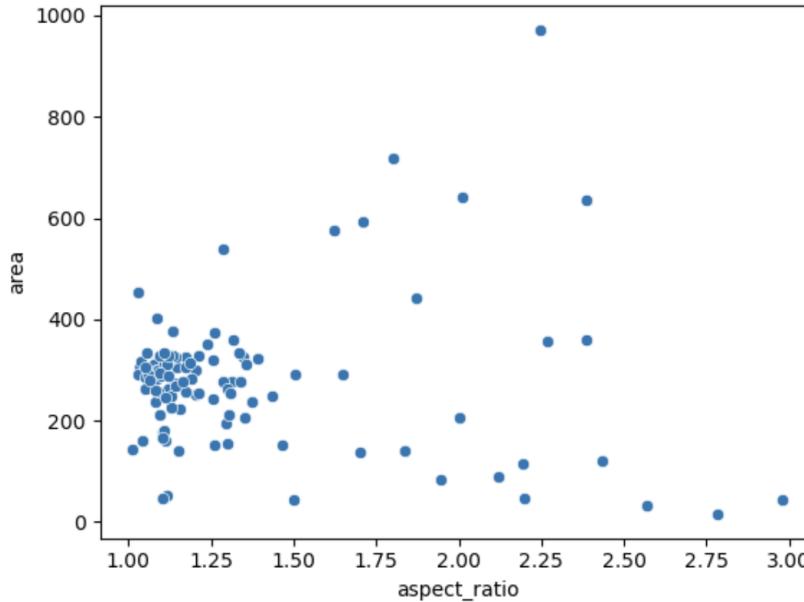
| area | intensity_mean | major_axis_length | minor_axis_length | aspect_ratio | file_name |
|------|----------------|-------------------|-------------------|--------------|--------------------|
| 139 | 96.546763 | 17.504104 | 10.292770 | 1.700621 | 20P1_POS0010_D_1UL |
| 360 | 86.613889 | 35.746808 | 14.983124 | 2.385805 | 20P1_POS0010_D_1UL |
| 43 | 91.488372 | 12.967884 | 4.351573 | 2.980045 | 20P1_POS0010_D_1UL |
| 140 | 73.742857 | 18.940508 | 10.314404 | 1.836316 | 20P1_POS0010_D_1UL |
| 144 | 89.375000 | 13.639308 | 13.458532 | 1.013432 | 20P1_POS0010_D_1UL |

Basic elements of data visualization

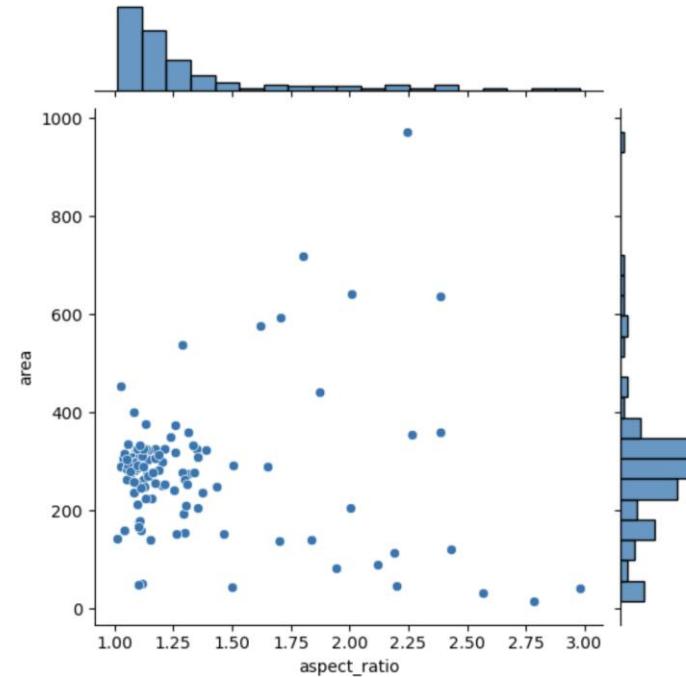
| Visual Variables | | Characteristics | | | | | |
|------------------|----------|---|-------------|--------------|-----------|----------|-------------------------------------|
| | | Selective | Associative | Quantitative | Order | Length | |
| | | yes | yes | yes | yes | infinite | |
| Position | Position |  | yes | yes | yes | yes | |
| | Size |  | yes | no | partially | yes | Selection: ~ 5 Distinction: ~ 20 |

Relationships

```
sns.scatterplot(data=df, x="aspect_ratio", y="area");
```



```
sns.jointplot(data=df, x="aspect_ratio", y="area");
```

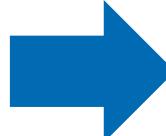


Relationships

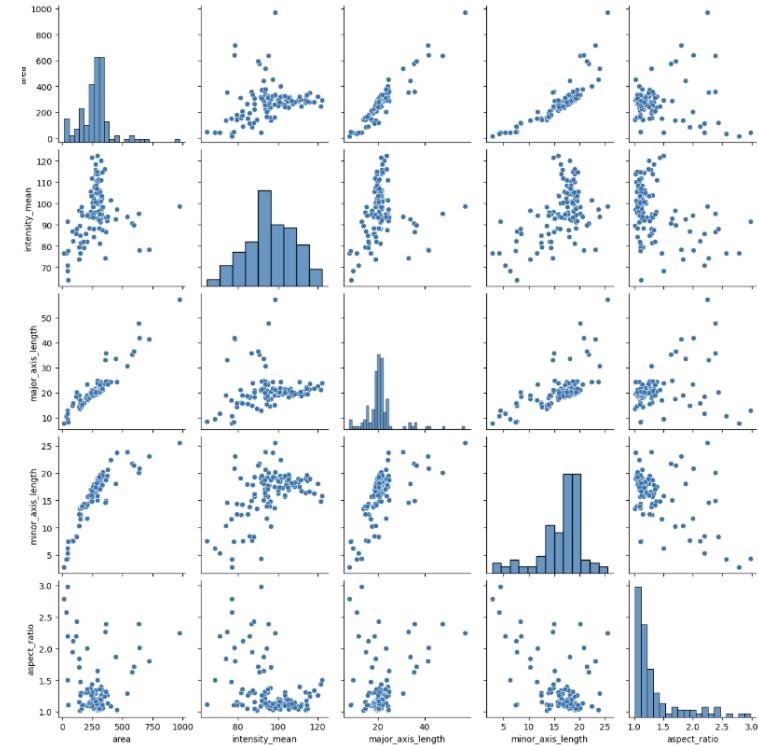
Pairplots are great for getting an overview

| | area | intensity_mean | major_axis_length | minor_axis_length | aspect_ratio | file_name |
|-----|------|----------------|-------------------|-------------------|--------------|--------------------|
| 0 | 139 | 96.546763 | 17.504104 | 10.292770 | 1.700621 | 20P1_POS0010_D_1UL |
| 1 | 360 | 86.613889 | 35.746808 | 14.983124 | 2.385805 | 20P1_POS0010_D_1UL |
| 2 | 43 | 91.488372 | 12.967884 | 4.351573 | 2.980045 | 20P1_POS0010_D_1UL |
| 3 | 140 | 73.742857 | 18.940508 | 10.314404 | 1.836316 | 20P1_POS0010_D_1UL |
| 4 | 144 | 89.375000 | 13.639308 | 13.458532 | 1.013432 | 20P1_POS0010_D_1UL |
| ... | ... | ... | ... | ... | ... | ... |
| 106 | 305 | 88.252459 | 20.226532 | 19.244210 | 1.051045 | 20P1_POS0007_D_1UL |
| 107 | 593 | 89.905565 | 36.508370 | 21.365394 | 1.708762 | 20P1_POS0007_D_1UL |
| 108 | 289 | 106.851211 | 20.427809 | 18.221452 | 1.121086 | 20P1_POS0007_D_1UL |
| 109 | 277 | 100.664260 | 20.307965 | 17.432920 | 1.164920 | 20P1_POS0007_D_1UL |
| 110 | 46 | 70.869565 | 11.648895 | 5.298003 | 2.198733 | 20P1_POS0007_D_1UL |

111 rows × 6 columns



```
sns.pairplot(data=df);
```

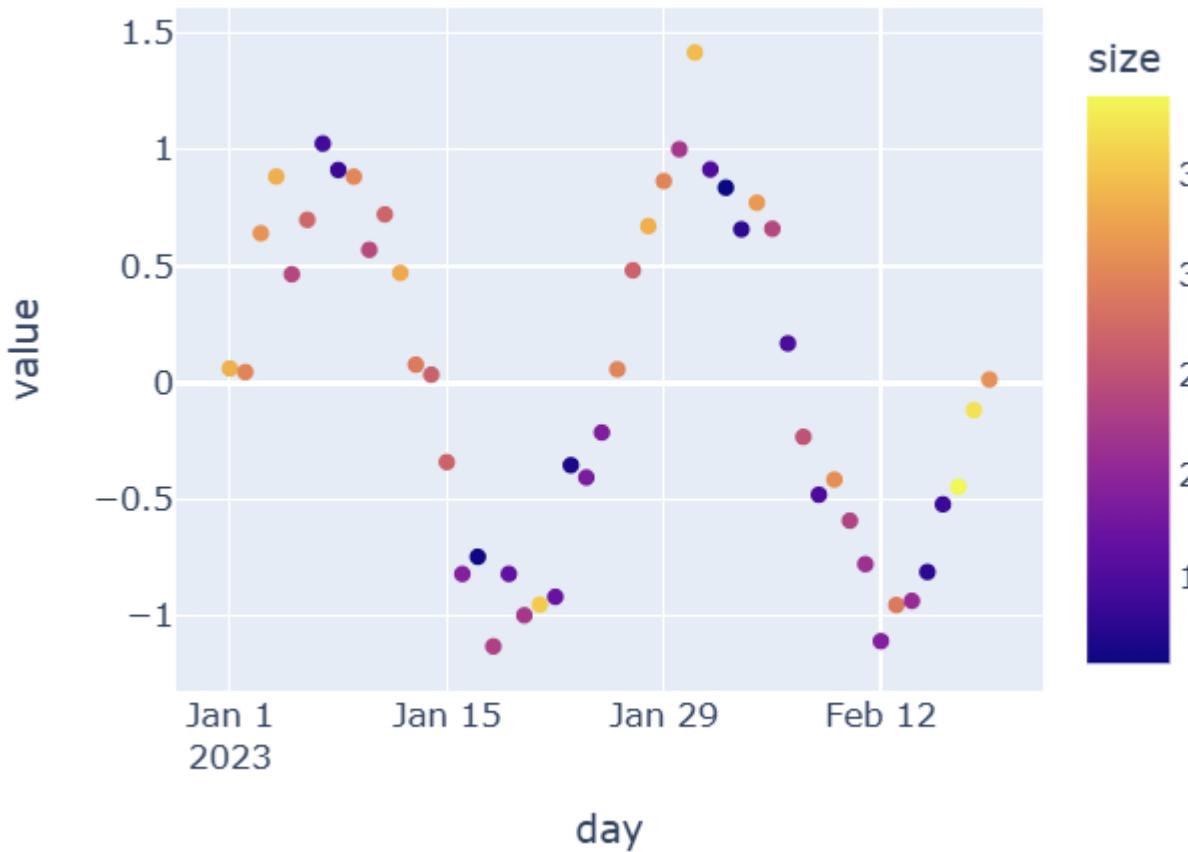


Interactive plotting

The plotly library aims at a similar API like seaborn and brings interactive plots.

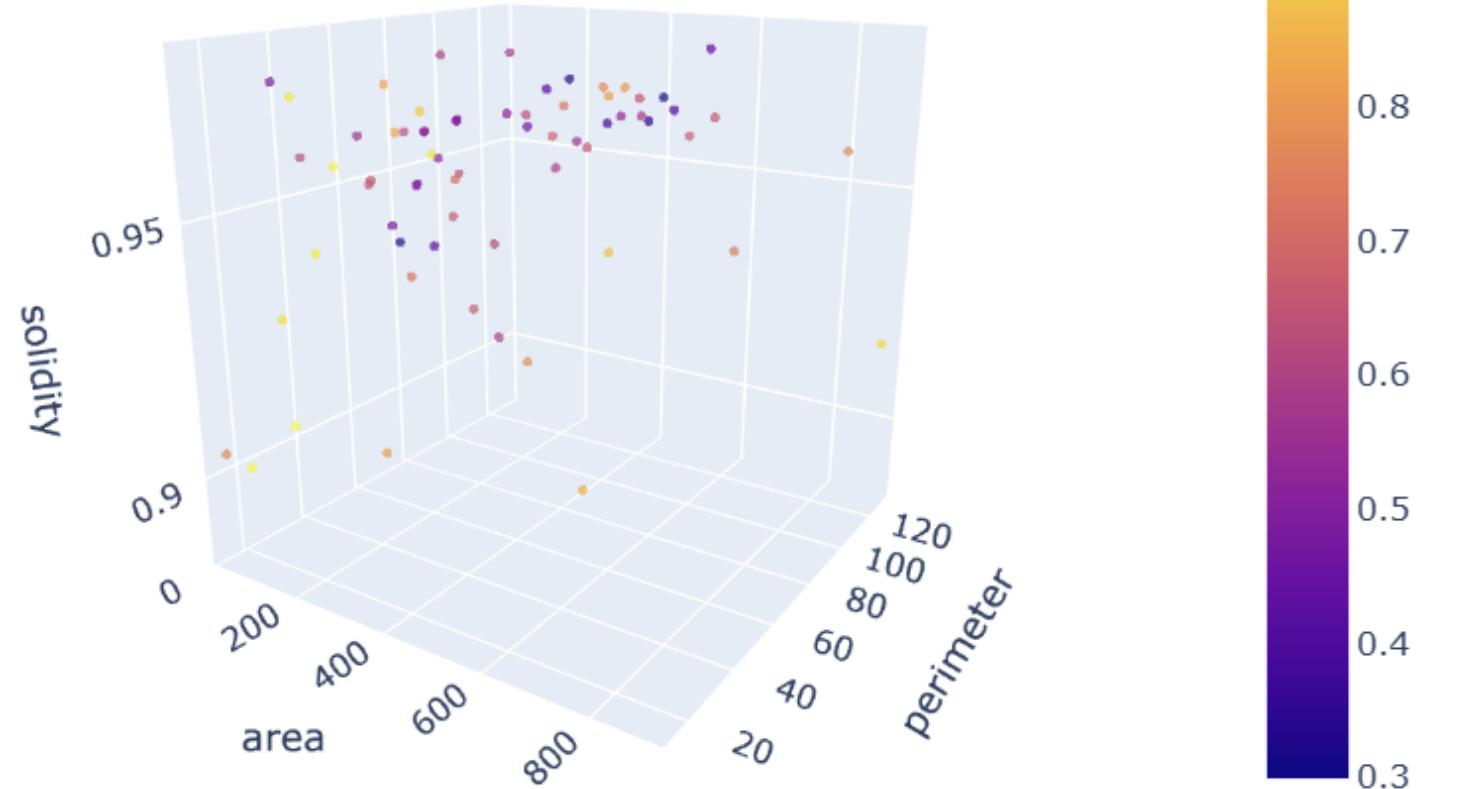
```
import plotly.express as px

px.scatter(
    df,
    x="day",
    y="value",
    color="size"
)
```



Plotting in 3D with plotly

```
import plotly.express as px  
  
fig = px.scatter_3d(df,  
                     x="area",  
                     y="perimeter",  
                     z="solidity",  
                     color="eccentricity")
```



Plotting higher dimensional data

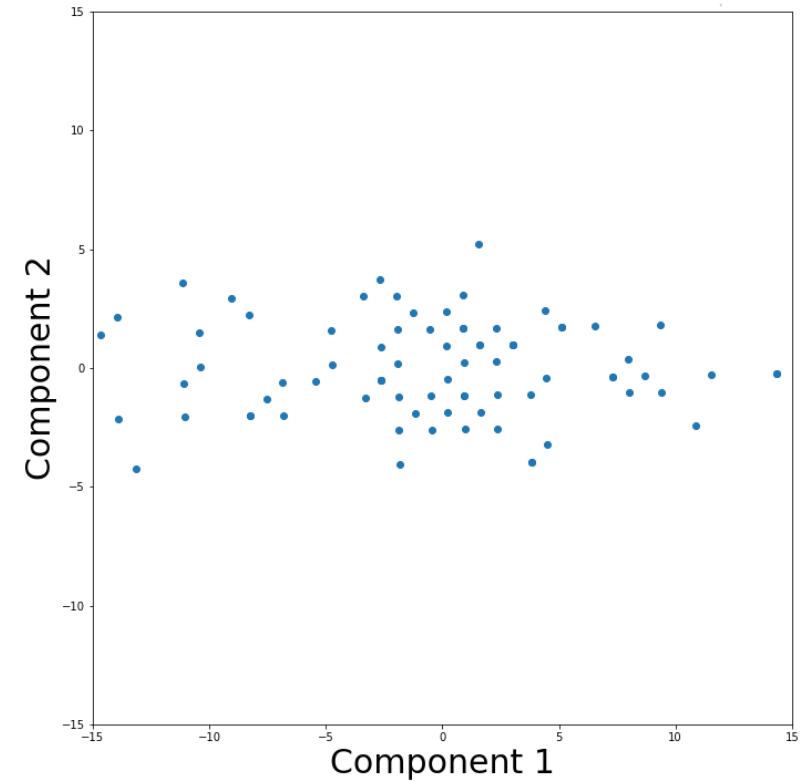
Using Dimensionality reduction (a.k.a. unsupervised machine learning)

-> Embeddings

| | count | mean | std |
|------------------------------|-------|------------|------------|
| label | 44.0 | 22.500000 | 12.845233 |
| area | 44.0 | 401.863636 | 202.852288 |
| bbox_area | 44.0 | 542.750000 | 295.106376 |
| equivalent_diameter | 44.0 | 21.781085 | 6.174086 |
| convex_area | 44.0 | 423.295455 | 216.613747 |
| max_intensity | 44.0 | 234.909091 | 17.517856 |
| mean_intensity | 44.0 | 190.116971 | 15.034153 |
| min_intensity | 44.0 | 128.000000 | 0.000000 |
| extent | 44.0 | 0.758804 | 0.063276 |
| local_centroid-0 | 44.0 | 11.439824 | 4.126230 |
| local_centroid-1 | 44.0 | 10.138666 | 3.491815 |
| solidity | 44.0 | 0.953153 | 0.024749 |
| feret_diameter_max | 44.0 | 26.382434 | 8.915046 |
| major_axis_length | 44.0 | 25.876797 | 9.591558 |
| minor_axis_length | 44.0 | 18.872898 | 5.158791 |
| orientation | 44.0 | 0.053057 | 0.691430 |
| eccentricity | 44.0 | 0.600434 | 0.165688 |
| standard_deviation_intensity | 44.0 | 29.556705 | 5.507399 |
| aspect_ratio | 44.0 | 1.374342 | 0.397611 |
| roundness | 44.0 | 0.762889 | 0.156695 |
| circularity | 44.0 | 0.918858 | 0.133288 |

Many dimensions

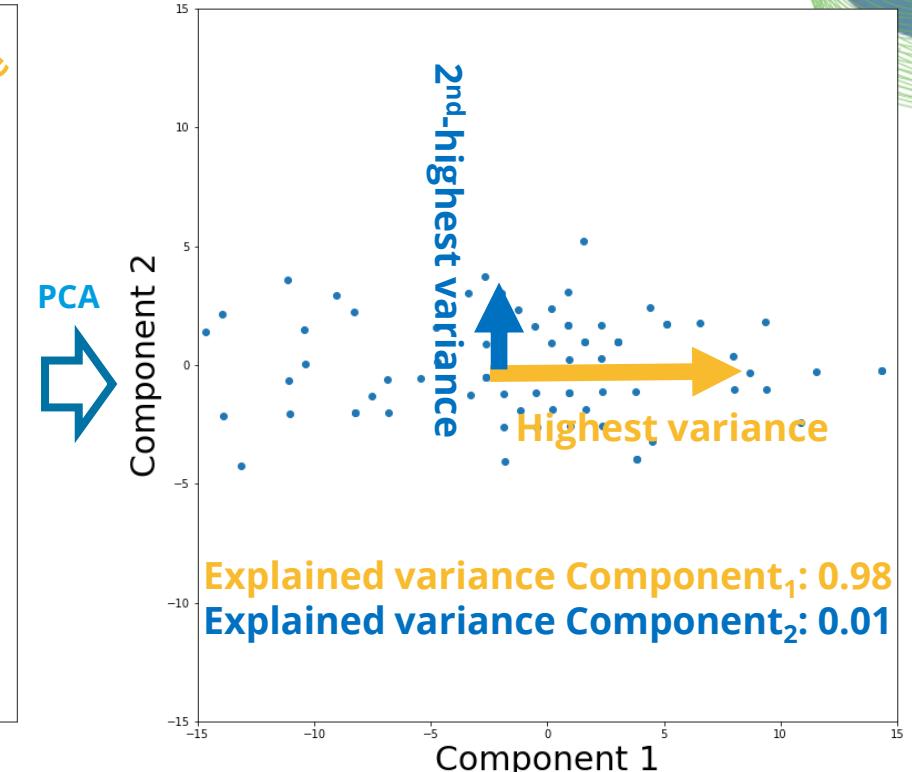
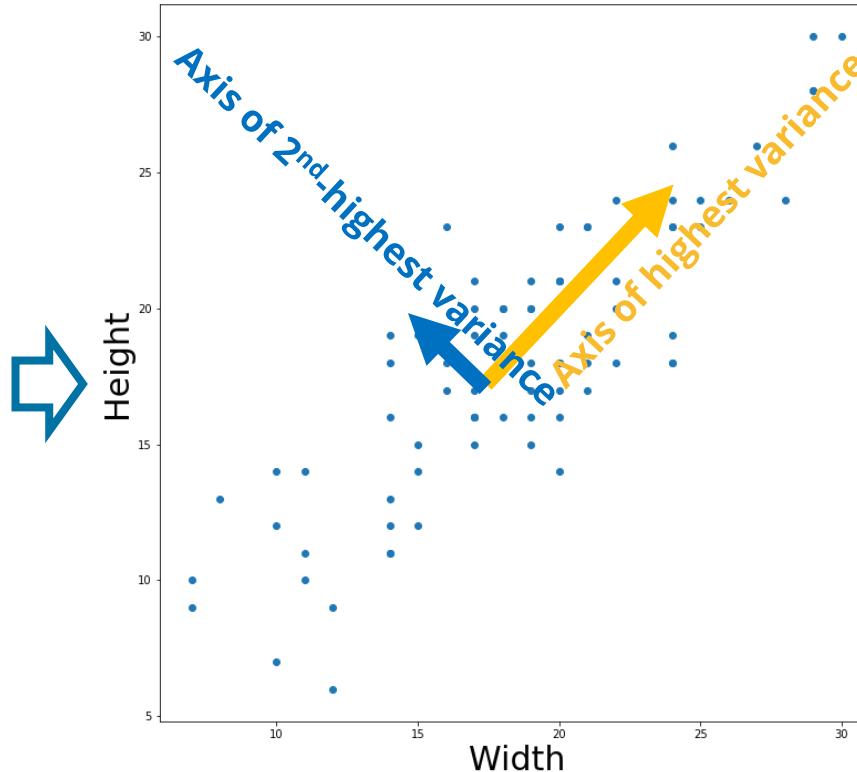
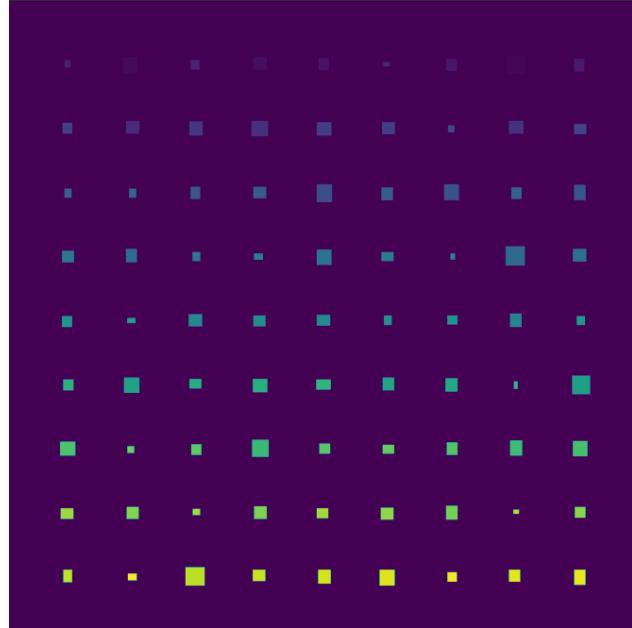
Dimensionality
reduction



PCA: Principal Component Analysis

Decomposes data into linear combinations of features that explain the highest variance

Example: Squares of different size

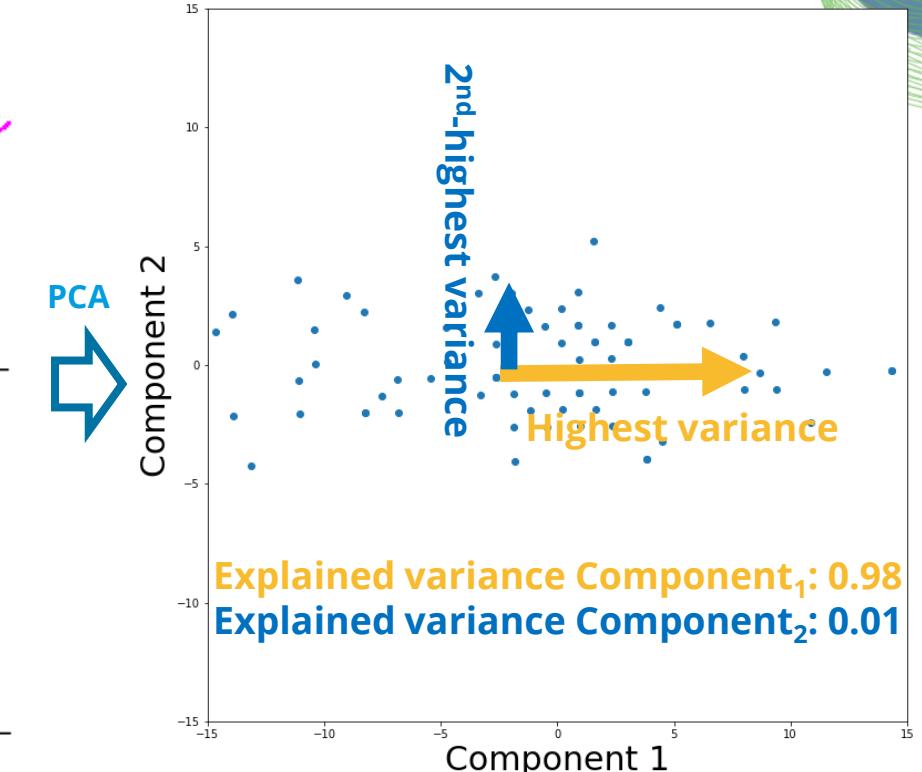
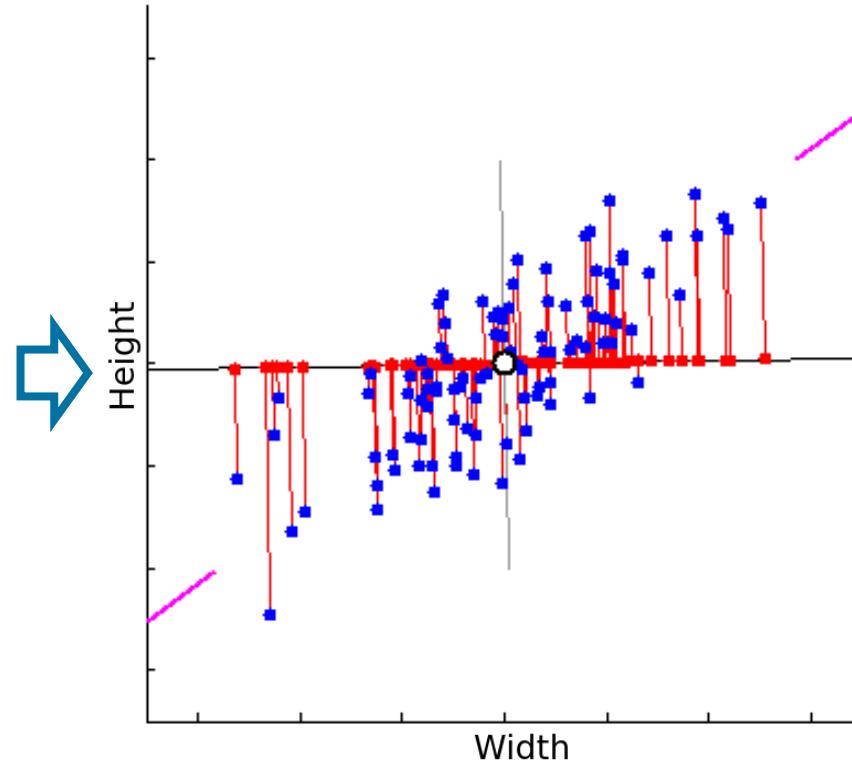
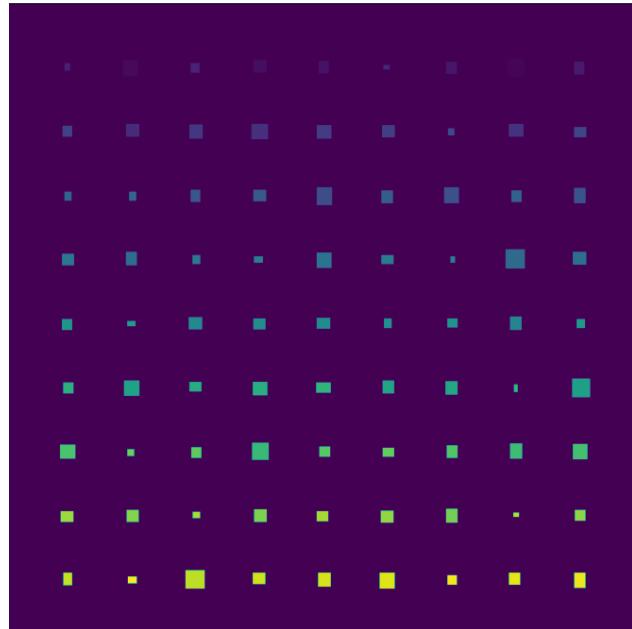


→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

PCA: Principal Component Analysis

Decomposes data into linear combinations of features that explain the highest variance

Example: Squares of different size



→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

PCA in Python: sklearn.decomposition.PCA

- Import package

```
from sklearn.decomposition import PCA
```

- Apply PCA

```
pca = PCA(n_components=2)  
pca.fit(standardized_data)
```

- Transform data into new coordinate system

```
transformed_data = pca.transform(data)
```

Important!

Always check the explained variance along the PCA component axes!

```
pca.explained_variance_ratio_
```

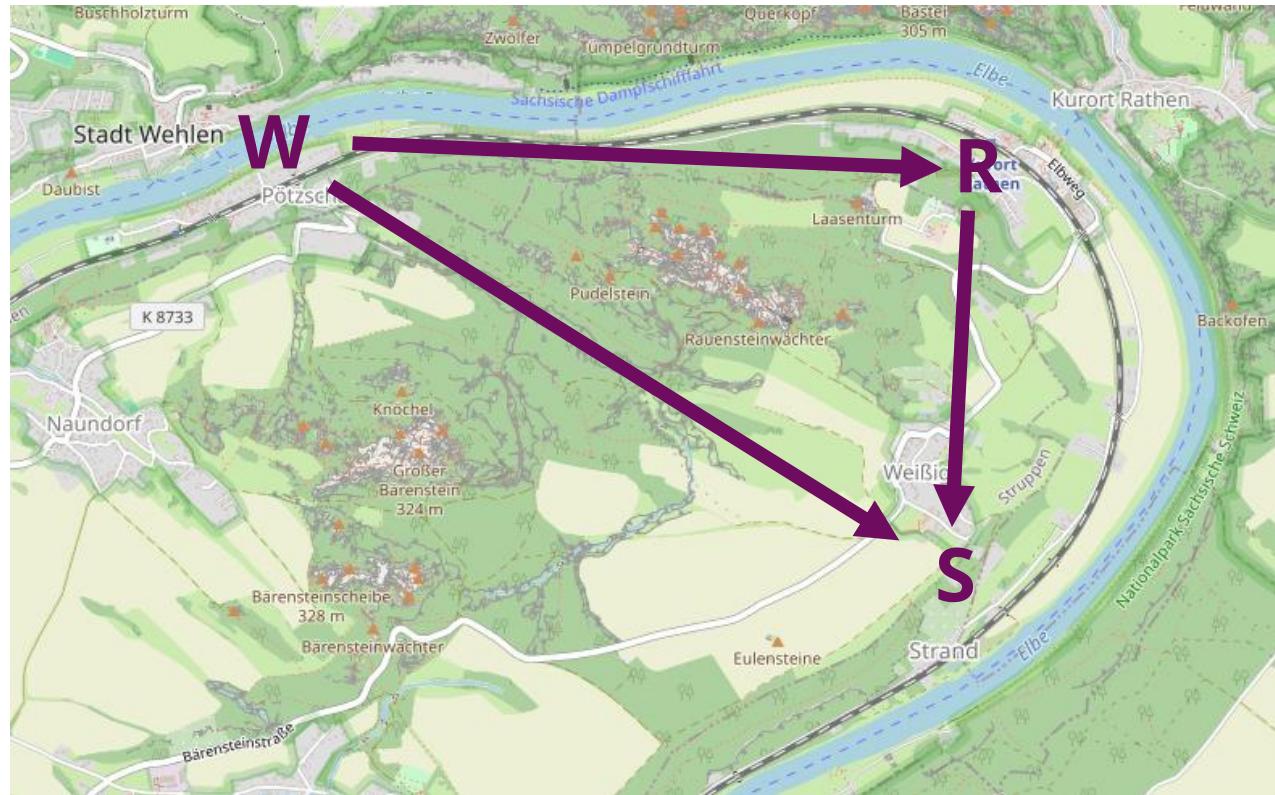
```
array([0.98773142, 0.01226858])
```

The screenshot shows the official scikit-learn website. At the top, there's a navigation bar with links for Install, User Guide, API, Examples, Community, and More. Below the header, the text "scikit-learn Machine Learning in Python" is displayed. There are six main sections with examples and descriptions:

- Classification:** Identifying which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: SVM, nearest neighbors, random forest, and more... Example image: A grid of 2x3 subplots showing handwritten digits.
- Regression:** Predicting a continuous-valued attribute associated with an object. Applications: Drug response, Stock prices. Algorithms: SVR, nearest neighbors, random forest, and more... Example image: A line plot titled "Boosted Decision Tree Regression" showing target values over data points.
- Clustering:** Automatic grouping of similar objects into sets. Applications: Customer segmentation, Grouping experiment outcomes. Algorithms: k-Means, spectral clustering, mean-shift, and more... Example image: A scatter plot of digits labeled with different colors representing clusters.
- Dimensionality reduction:** Reducing the number of random variables to consider. Applications: Visualization, Increased efficiency. Example image: A 2D scatter plot showing data points and their projections onto a lower-dimensional space.
- Model selection:** Comparing, validating and choosing parameters and models. Applications: Improved accuracy via parameter tuning. Example image: A line plot showing cross-validation scores for different model parameters.
- Preprocessing:** Feature extraction and normalization. Applications: Transforming input data such as text for use with machine learning algorithms. Example image: A 2D scatter plot showing text data points being transformed into numerical features.

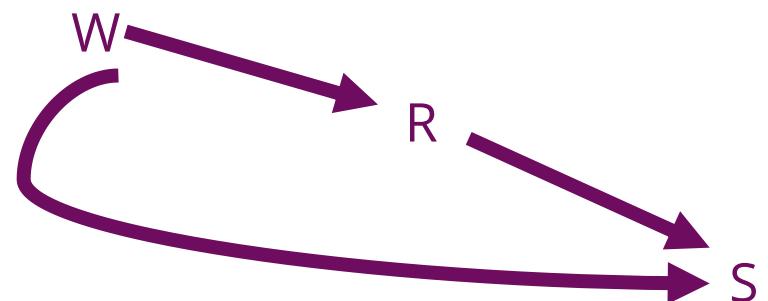
Non-Euclidian spaces

Not all features might be distances in space



Use travel time between W and S as metric for distance

→ Travelling from **Wehlen** to **Strand** by bike is probably faster if you make a detour through **Rathen**



Uniform Manifold Approximation Projection (UMAP)

Structural, hierarchical, **non-linear** transformation

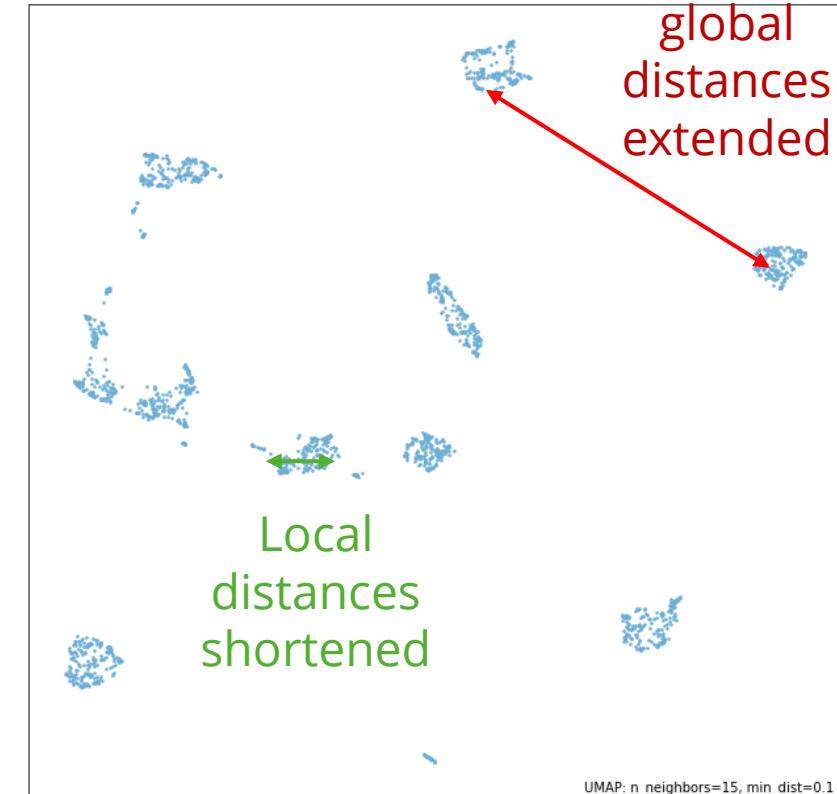
Modifies density of data points.

| | count | mean | std |
|------------------------------|-------|------------|------------|
| label | 44.0 | 22.500000 | 12.845233 |
| area | 44.0 | 401.863636 | 202.852288 |
| bbox_area | 44.0 | 542.750000 | 295.106376 |
| equivalent_diameter | 44.0 | 21.781085 | 6.174086 |
| convex_area | 44.0 | 423.295455 | 216.613747 |
| max_intensity | 44.0 | 234.909091 | 17.517856 |
| mean_intensity | 44.0 | 190.116971 | 15.034153 |
| min_intensity | 44.0 | 128.000000 | 0.000000 |
| extent | 44.0 | 0.758804 | 0.063276 |
| local_centroid-0 | 44.0 | 11.439824 | 4.126230 |
| local_centroid-1 | 44.0 | 10.138666 | 3.491815 |
| solidity | 44.0 | 0.953153 | 0.024749 |
| feret_diameter_max | 44.0 | 26.382434 | 8.915046 |
| major_axis_length | 44.0 | 25.876797 | 9.591558 |
| minor_axis_length | 44.0 | 18.872898 | 5.158791 |
| orientation | 44.0 | 0.053057 | 0.691430 |
| eccentricity | 44.0 | 0.600434 | 0.165688 |
| standard_deviation_intensity | 44.0 | 29.556705 | 5.507399 |
| aspect_ratio | 44.0 | 1.374342 | 0.397611 |
| roundness | 44.0 | 0.762889 | 0.156695 |
| circularity | 44.0 | 0.918858 | 0.133288 |

Many dimensions



UMAP 2



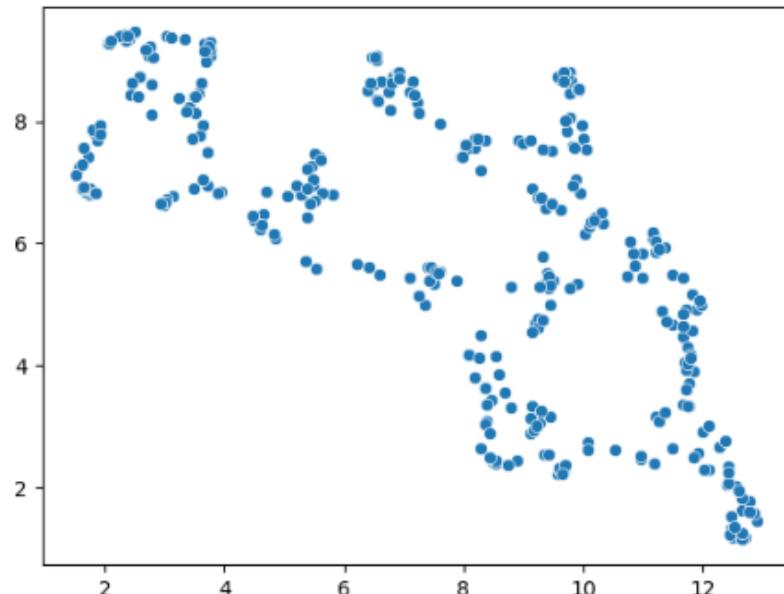
UMAP 1

Uniform Manifold Approximation Projection (UMAP)

Non-deterministic algorithm: You execute it twice, you get different results.

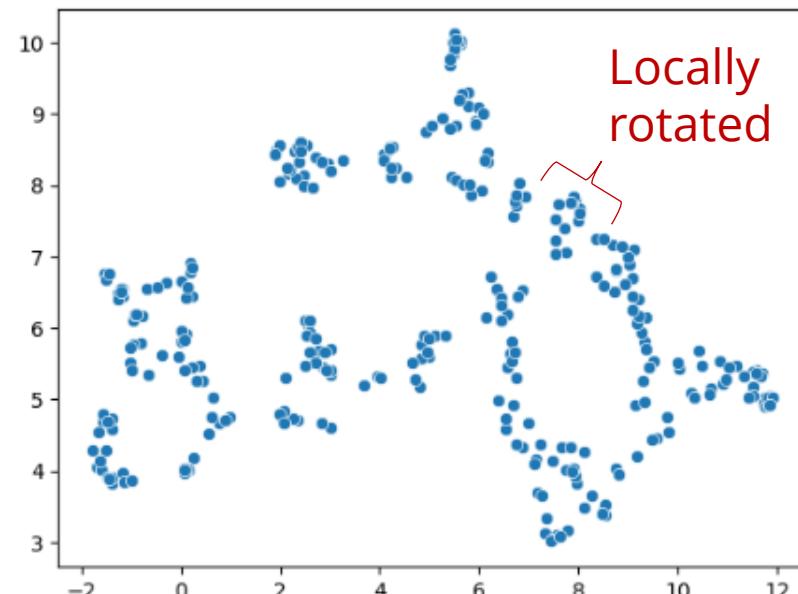
```
[11]: reducer = umap.UMAP()  
embedding2 = reducer.fit_transform(scaled_statistics)  
  
seaborn.scatterplot(x=embedding2[:, 0],  
                     y=embedding2[:, 1])
```

```
[11]: <AxesSubplot: >
```



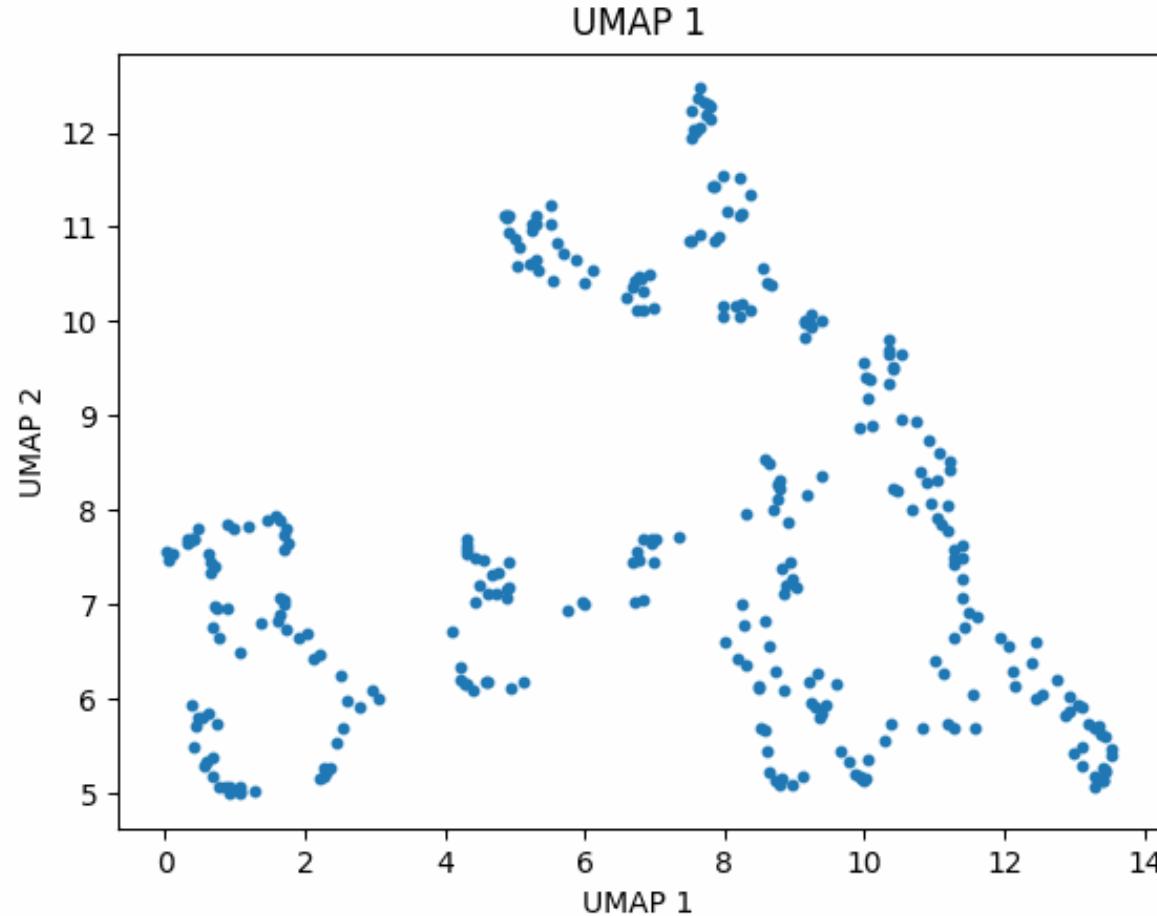
```
[12]: reducer = umap.UMAP()  
embedding2 = reducer.fit_transform(scaled_statistics)  
  
seaborn.scatterplot(x=embedding2[:, 0],  
                     y=embedding2[:, 1])
```

```
[12]: <AxesSubplot: >
```



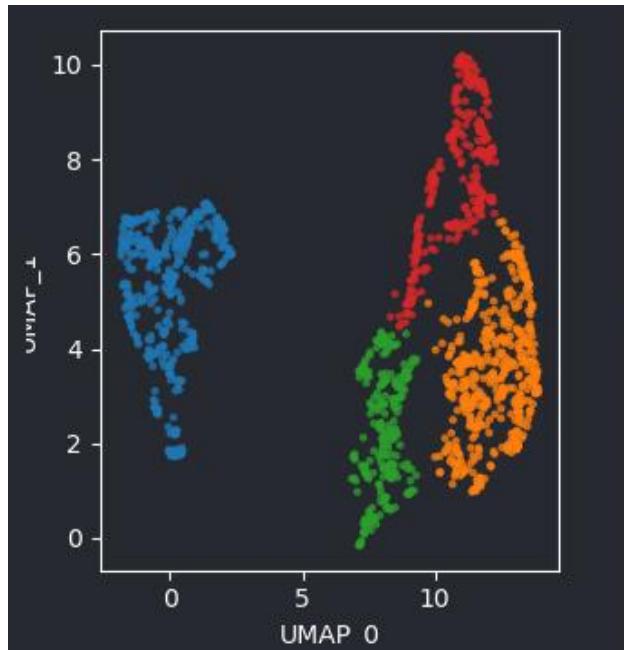
Uniform Manifold Approximation Projection (UMAP)

Non-deterministic algorithm: You execute it twice, you get different results.

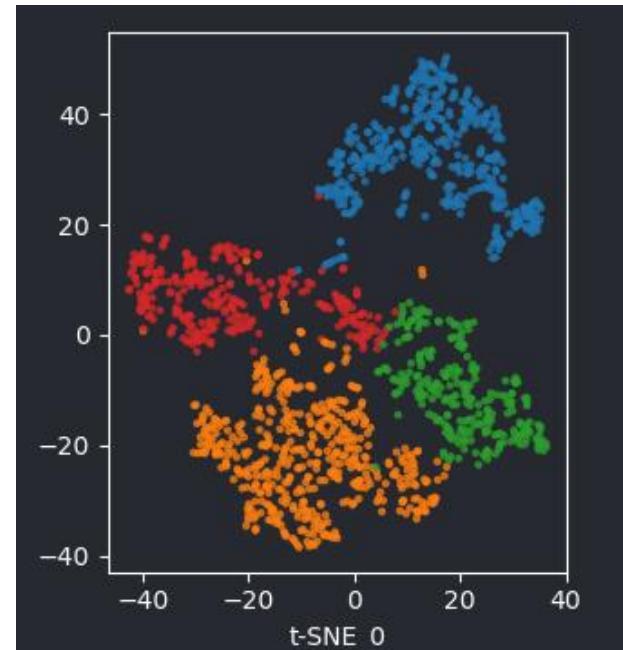


Dimensionality reduction

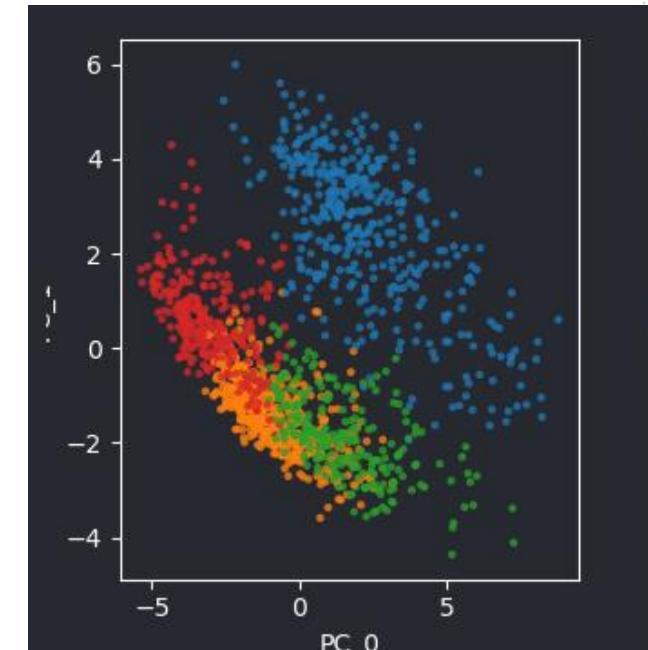
Visual comparison of different methods



Uniform manifold approximation and
projection (UMAP)



t-distributed stochastic neighbor
embedding (t-SNE)



Principal component analysis (PCA)

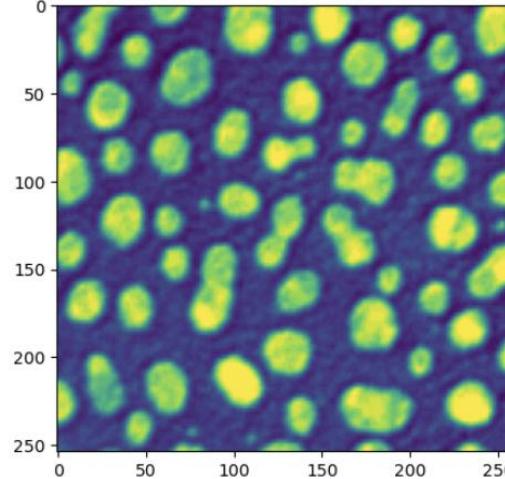
Visualizing images

`imshow` originates from Matlab and has been implemented many times.

matplotlib

```
[3]: plt.imshow(image)
```

```
[3]: <matplotlib.image.AxesImage at 0x1fa5b5145d0>
```

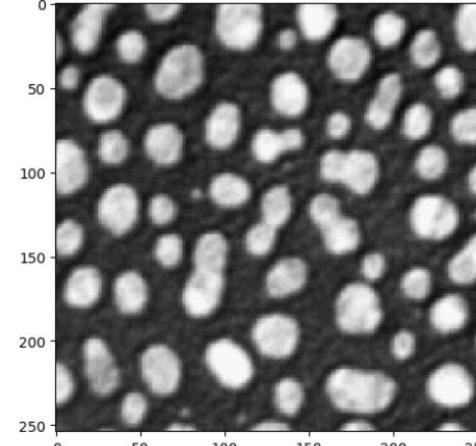


scikit-image

```
[4]: io.imshow(image)
```

```
C:\Users\rober\AppData\Local\Temp\ipykernel_46732\2038164919.py:1:  
FutureWarning: `imshow` is deprecated since version 0.25 and will be removed in version 0.27. Please use `matplotlib`, `napari`, etc. to visualize images.  
io.imshow(image)
```

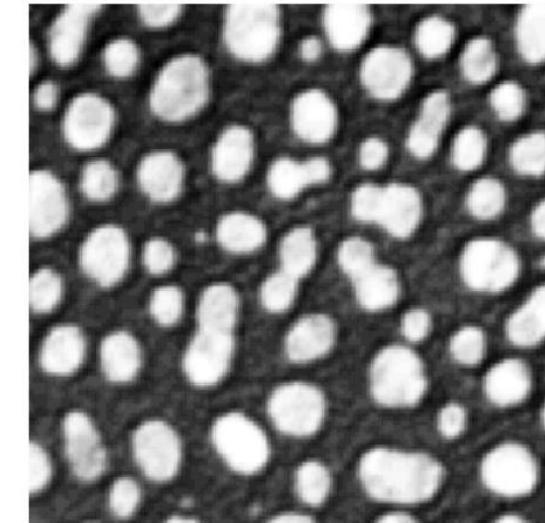
```
[4]: <matplotlib.image.AxesImage at 0x1fa580cbc50>
```



```
[1]: import matplotlib.pyplot as plt  
import stackview  
from skimage import io
```

stackview

```
[5]: stackview.imshow(image)
```



Visualizing images

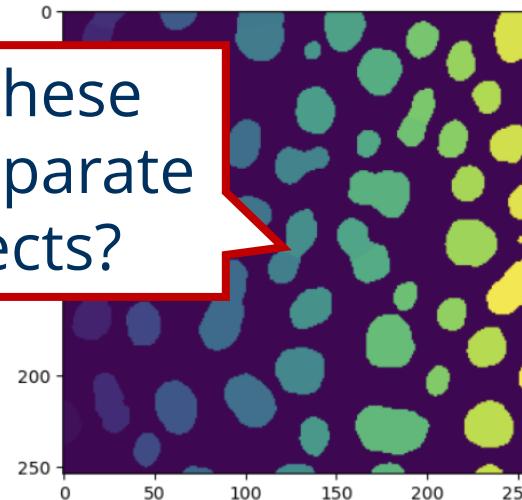
imshow originates from Matlab and has been implemented many times.

matplotlib

```
[6]: plt.imshow(labels)
```

```
[6]: <matplotlib.image.AxesImage at 0x1fa5b6df210>
```

Are these two separate objects?

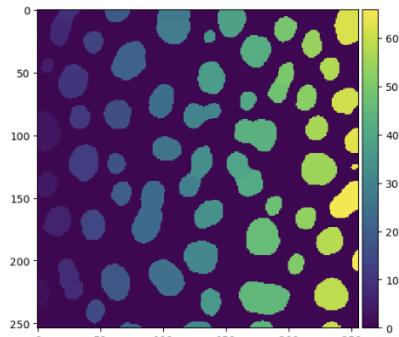


scikit-image

```
[7]: io.imshow(labels)
```

```
C:\Users\rober\AppData\Local\Temp\ipykernel_46732\28416559.py:1: FutureWarning: 'imshow' is deprecated since version 0.25 and will be removed in version 0.27. Please use `matplotlib`, `napari`, etc. to visualize images.  
io.imshow(labels)  
C:\Users\rober\miniforge3\envs\bob_env\Lib\site-packages\skimage\io\_plugins\matplotlib_plugin.py:15  
8: UserWarning: Low image data range; displaying image with stretched contrast.  
lo, hi, cmap = _get_display_range(image)
```

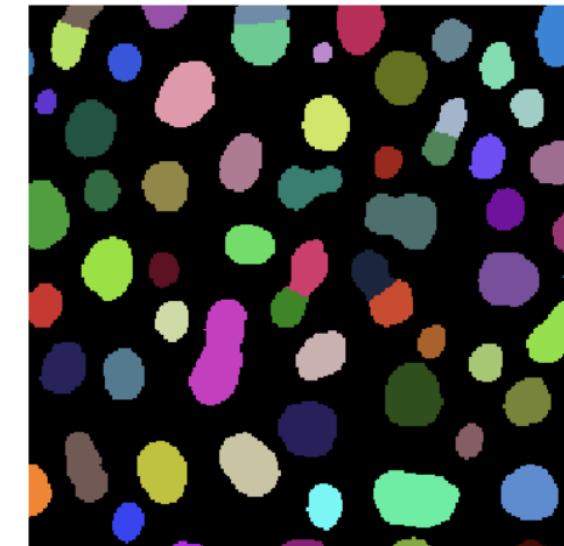
```
[7]: <matplotlib.image.AxesImage at 0x1fa5b67b410>
```



```
[1]: import matplotlib.pyplot as plt  
import stackview  
from skimage import io
```

stackview

```
[8]: stackview.imshow(labels)
```



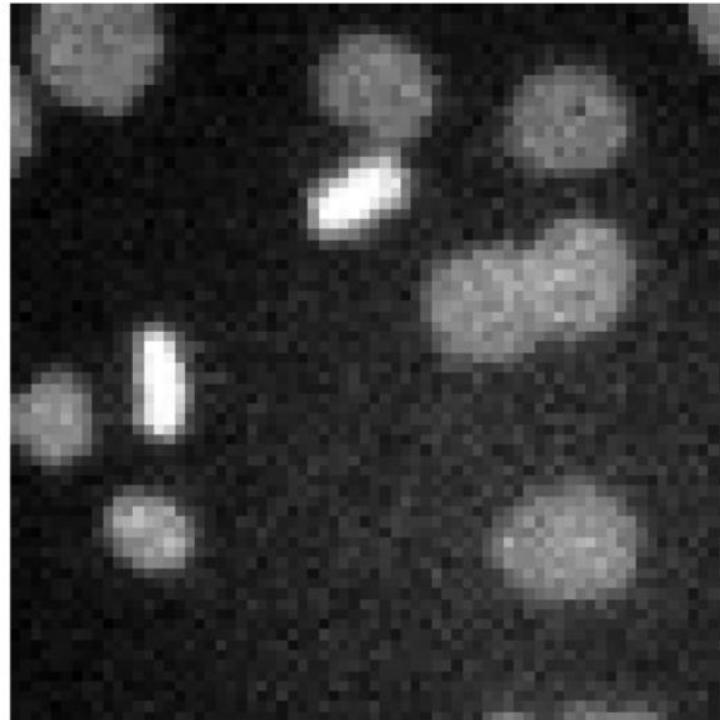
Visualizing images

... using *stackview*

Disclosure: I maintain this

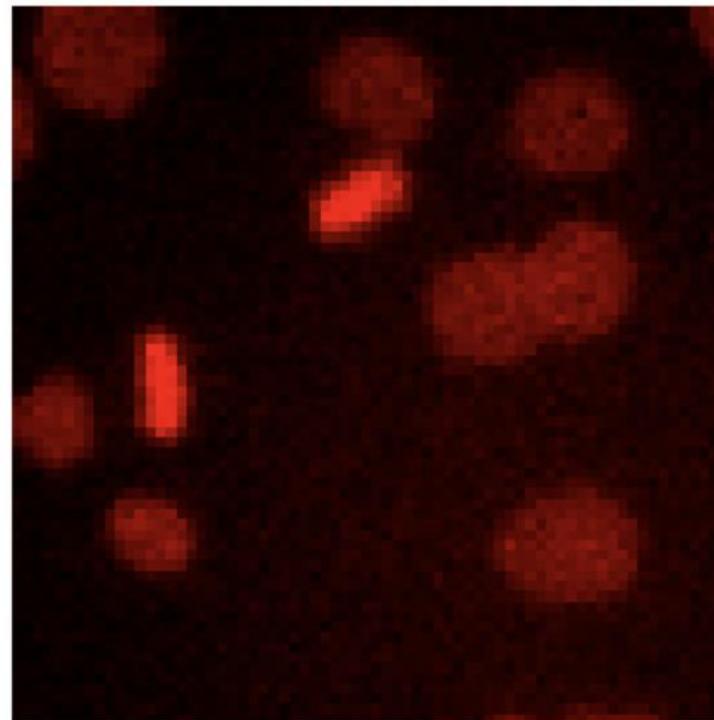
[3]:

```
import stackview  
stackview.imshow(image)
```



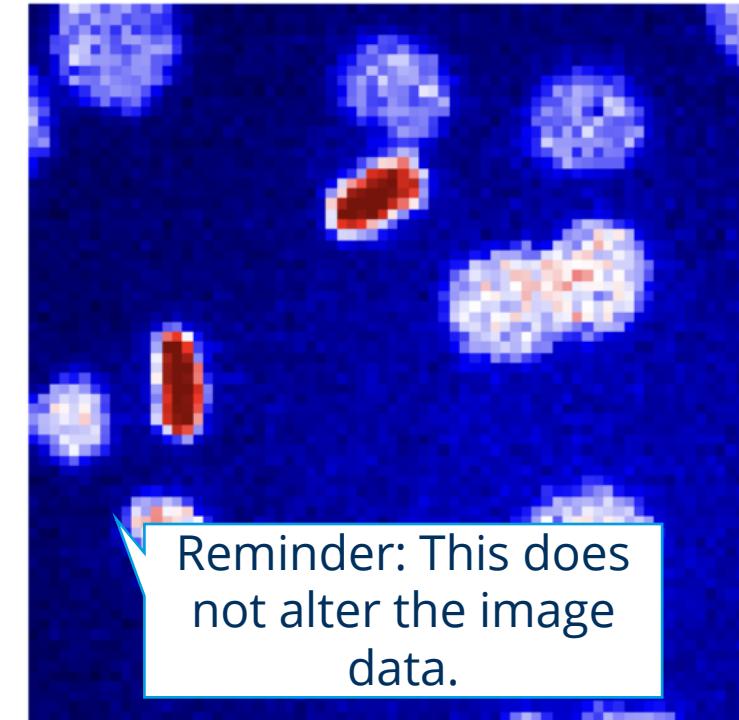
[5]:

```
stackview.imshow(image, colormap="pure_red")
```



[6]:

```
stackview.imshow(image, colormap="seismic")
```

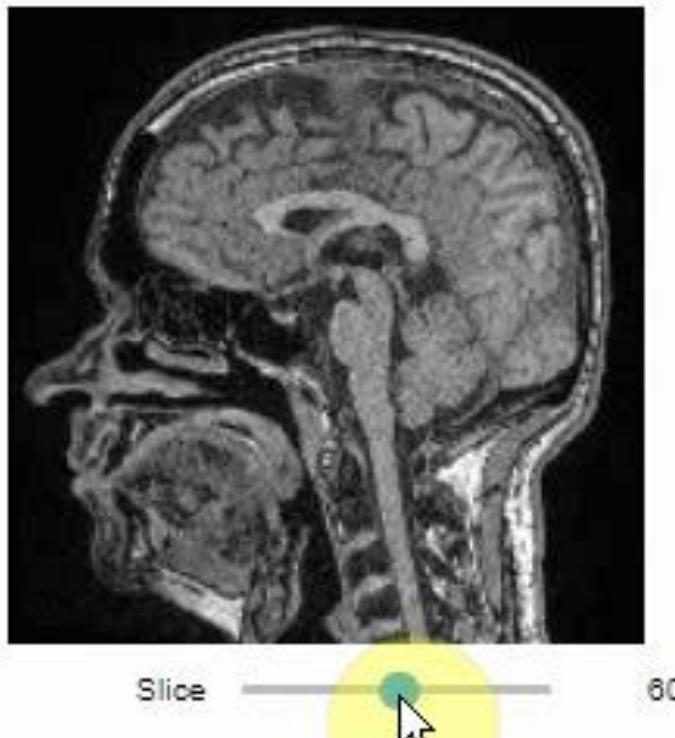


Reminder: This does
not alter the image
data.

Visualizing images

... using *stackview*

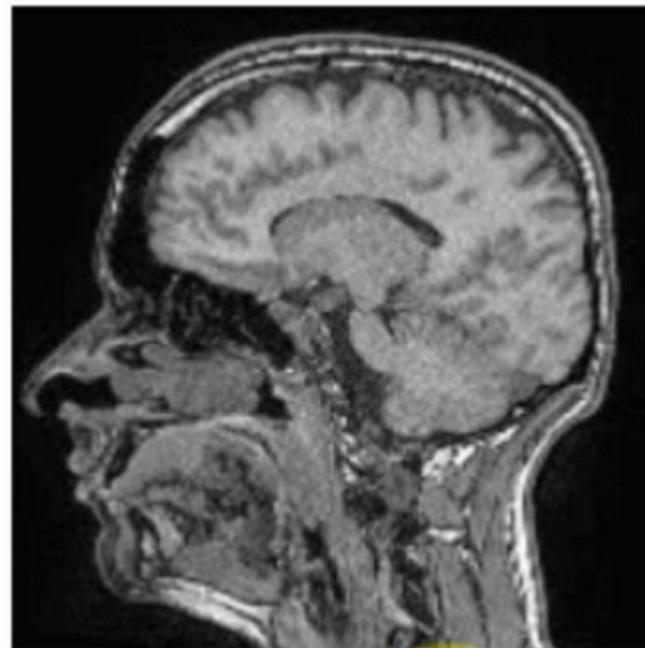
```
[6]: stackview.slice(mri_image)
```



Visualizing images

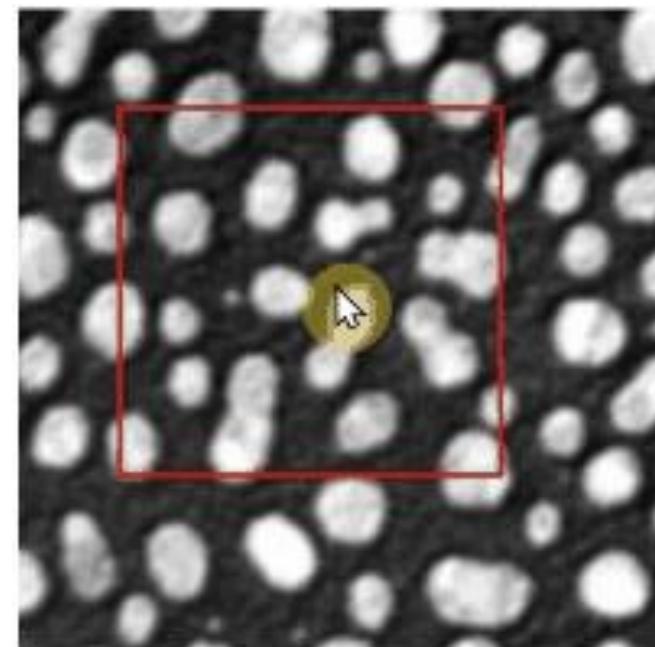
... using *stackview*

```
[6]: stackview.slice(mri_image)
```



Slice 69

```
[8]: stackview.histogram(image)
```

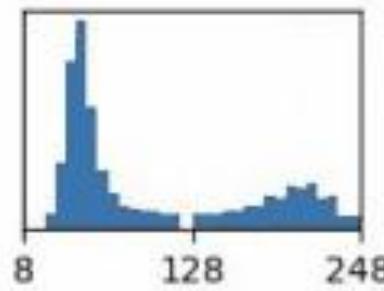


slice (... , 38:184, 39:190)

dtype uint8

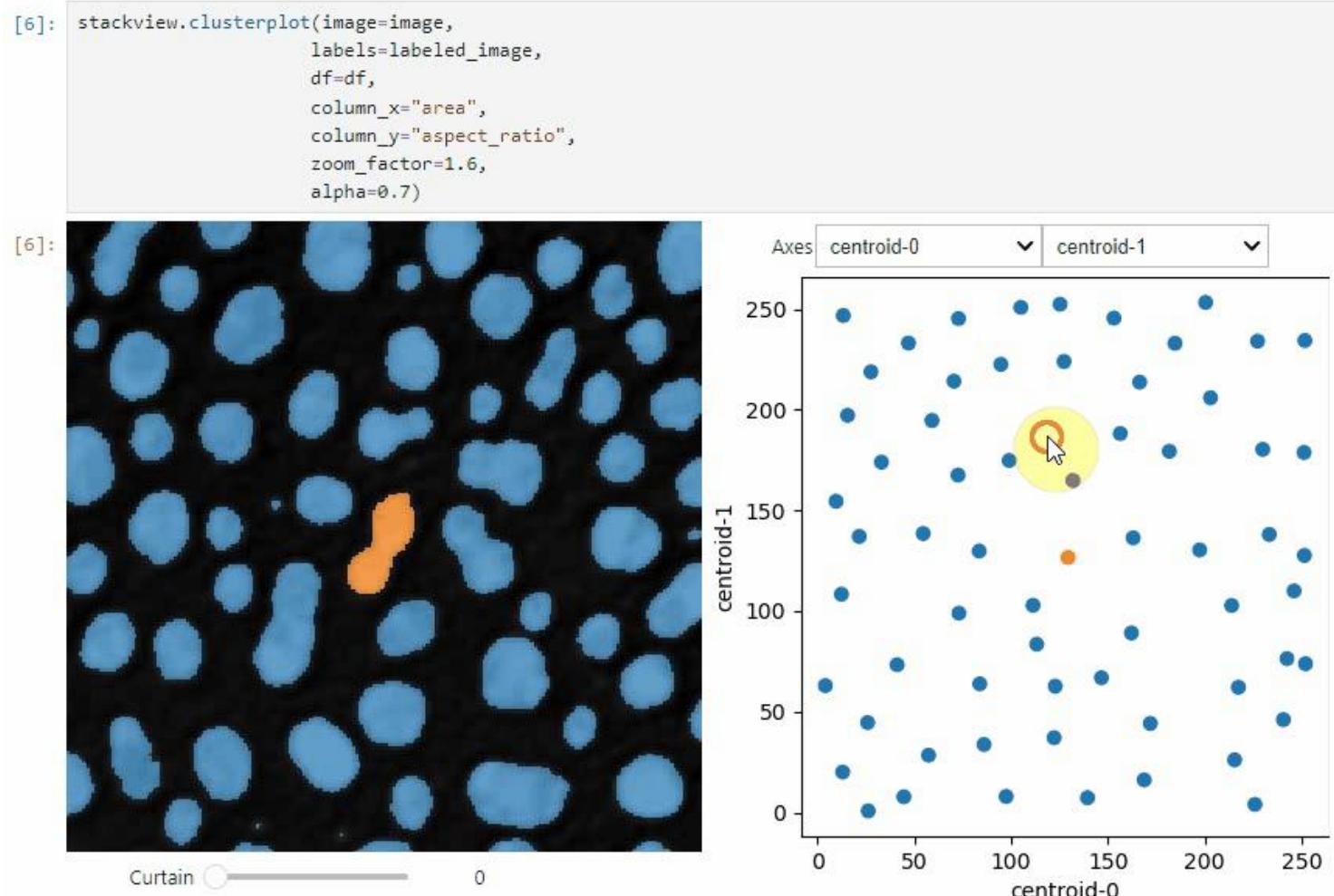
min 8

max 248



Selecting objects according to their properties

Understanding what certain measurements *mean* may require interactive user interfaces

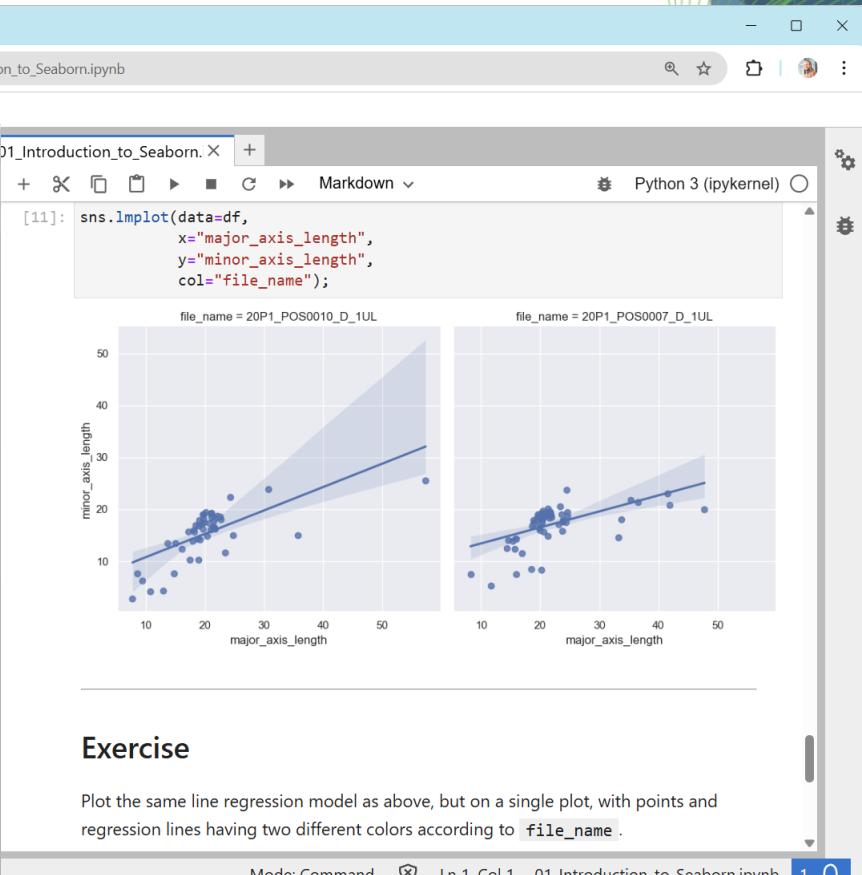
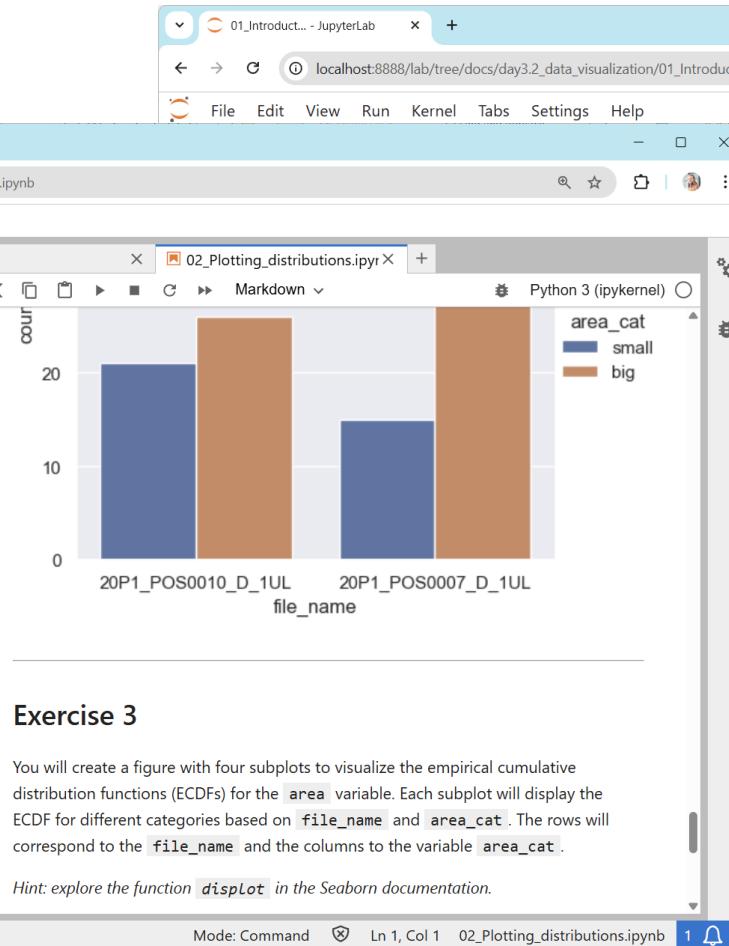
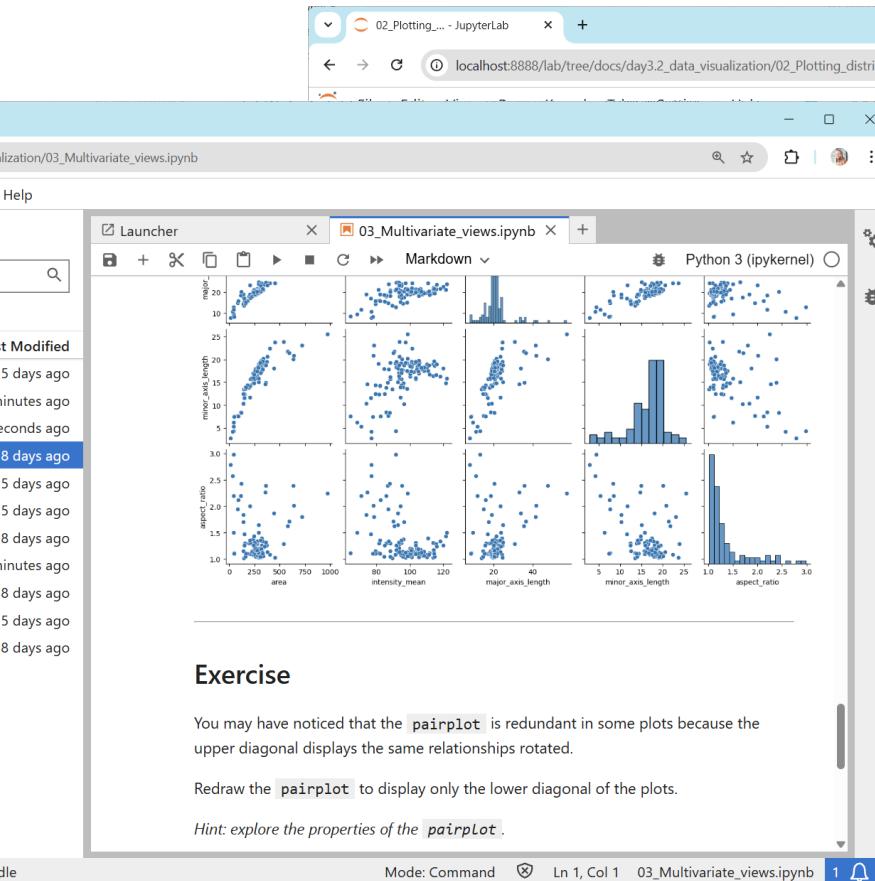


Exercises

Robert Haase

Plotting: seaborn

Explore the seaborn API



Plotting: plotly

Explore interactive plotting using plotly

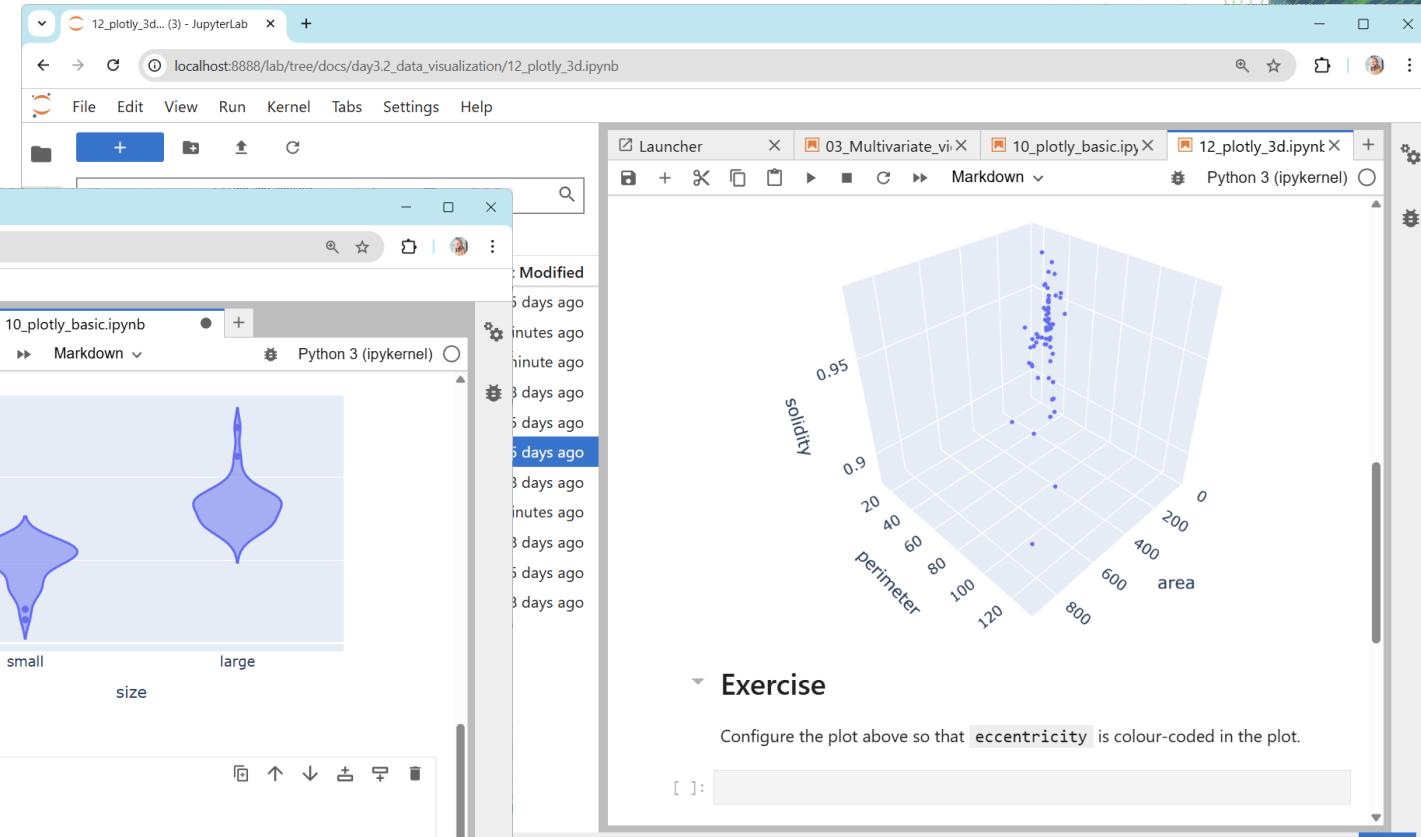
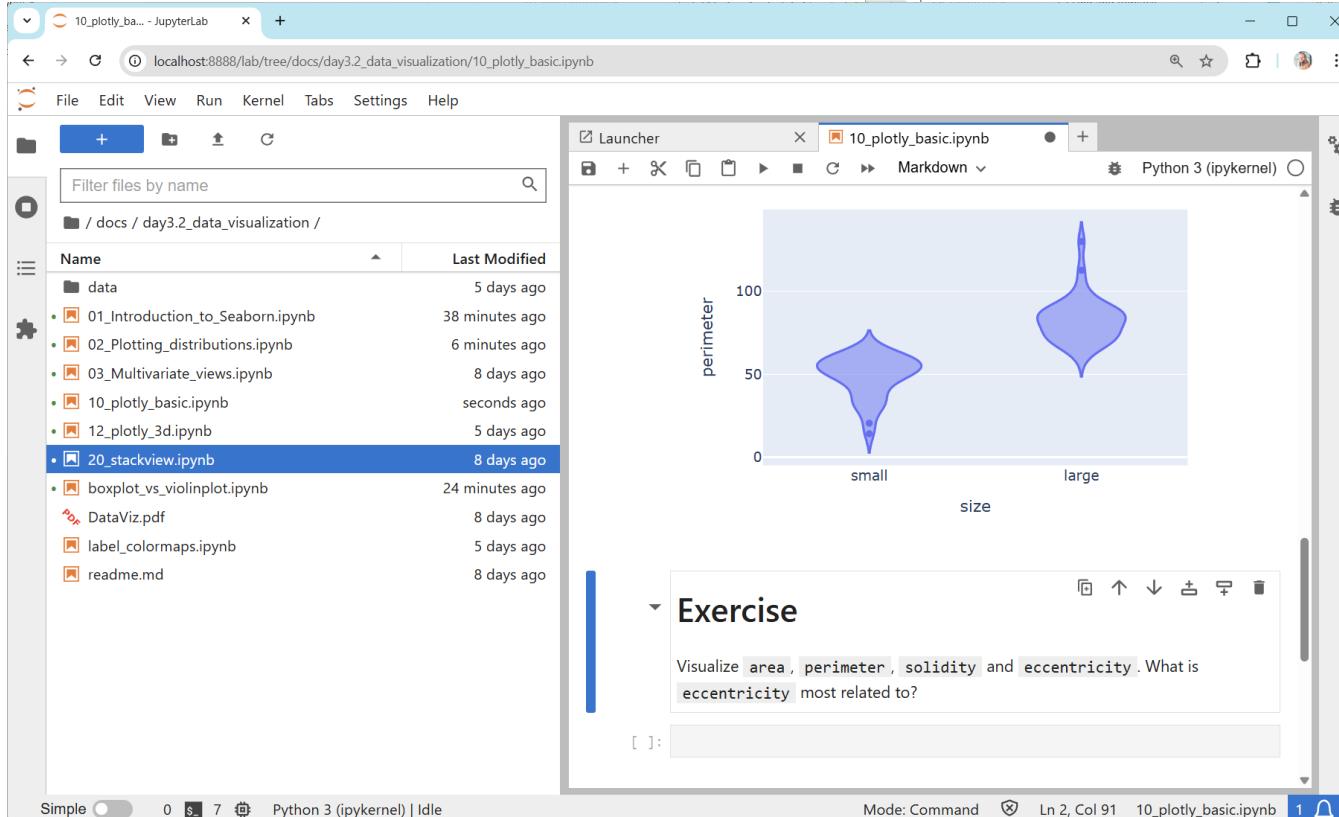


Image visualization

Note: This will only work on the jupyter4nfdi system.

You will need to install stackview by executing this from a code cell:

```
!pip install stackview
```

