DEGGENDORF INSTITUTE OF TECHNOLOGY

Data Visualization

Prof. Dr. Phillipp Torkler

Data Visualization Provides Data Exploration and Communication

Data Visualization serves **two** major purposes:

1. **Data Exploration:**
   - Familiarize with a data set
   - Look for patterns in data
   - Patterns or regularities in the data set are expected but not known in advance
   - Data exploration can guide the scientific process
   - New scientific ideas can emerge from visualizations from (laboratory) data. Thus, data visualization is often more than ‘just showing data’.

2. **Communication / Presentation:**
   - Interesting findings have been made and need to get communicated clearly to readers
   - Focus to tell a clear message
   - A reader does not need to get through the same process as a researcher that tries to find patterns during data exploration. In contrast, key findings should be communicated without overwhelming a reader with unnecessary data

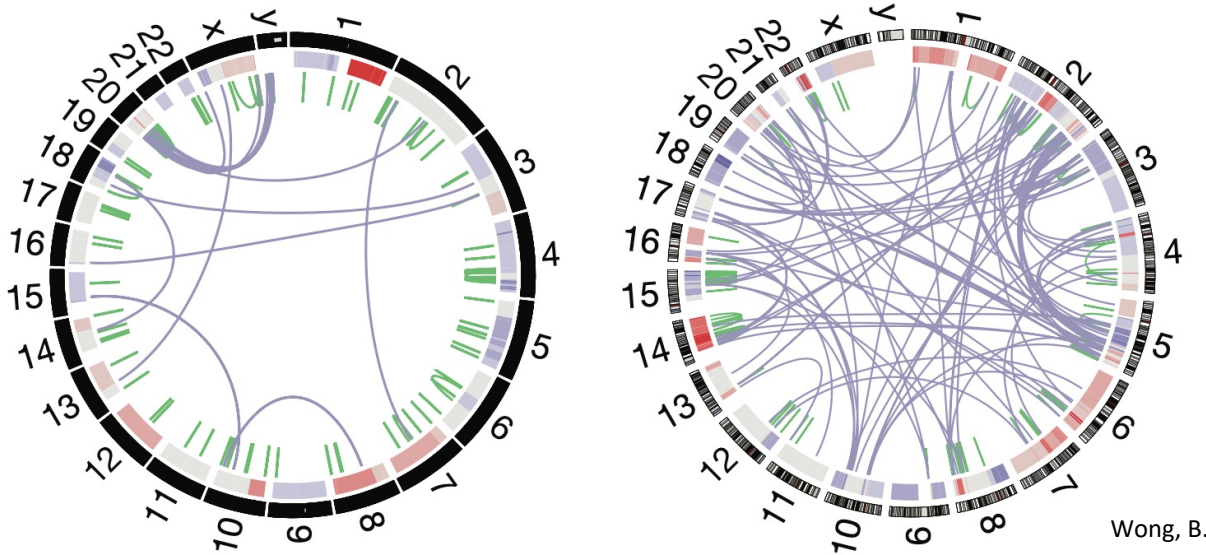
In either case, the purpose of any figure is to transport a message!

Why Do We Need Exploration and Communication?

Today, the challenge for researchers is to take benefit from large data sets without getting drawn in too much data. Thus, both exploration and communication is key for finding and sharing new insights.

Goals of data visualization:

- enable researches to explore and explain their data through (interactive) visualizations
- take advantage of the human's ability to recognize patterns
- data types and research questions evolve rapidly in the scientific community. Likewise, data visualizations need to adapt to new techniques to provide insights.
- Visualization as a complement for algorithmic approaches to provide a mental image of what happens (see example)



Wong, B. "Visualzing biological data" *Nature Methods* (2012)

Technical and Design Aspects of Data Visualization

A figure can be **any** visual representation: graphs, photos, drawings, schematics, cartoons, maps etc.. Despite their variety, the purpose of any figure is to support a message. **Good figures will show the data AND transport the message** you want to tell!

Figure creation consists of two major building blocks:

1. The ’technical’ generation of a figure (the doing):
   - e.g. use of a programming language and corresponding graphic library
   - choice and usage of graphics software
   - photography etc…

2. The ‘design’ of a figure (the theory and idea):
   - use graphic design principles
   - biology of the human visual system
   - psychology
   - statistics (in case of scientific visualizations)

In this course we want to train ’the doing’ and introduce a couple of design principles to generate meaningful and clear figures.

Why Do We Need Design Aspects And Statistics?

TECHNISCHE HOCHSCHULE DEGGENDORF THD

- Today, an unprecedented amount of data is available. In addition, the generation of visualizations has been made easy. In other words, everyone has data, wants to get knowledge out of data and visualizations can be made without much effort.

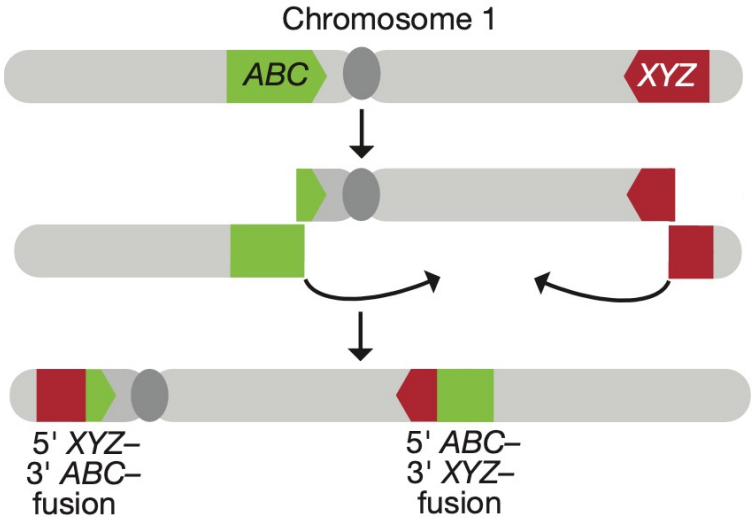
- However, information can be lost or hidden in bad visualizations leading to misleading information. As a consequence, knowledge about the visual reception of humans is needed to generate precise and non-misleading visualizations. Similarly, a background in statistics is required to generate (and understand) data driven visualizations.

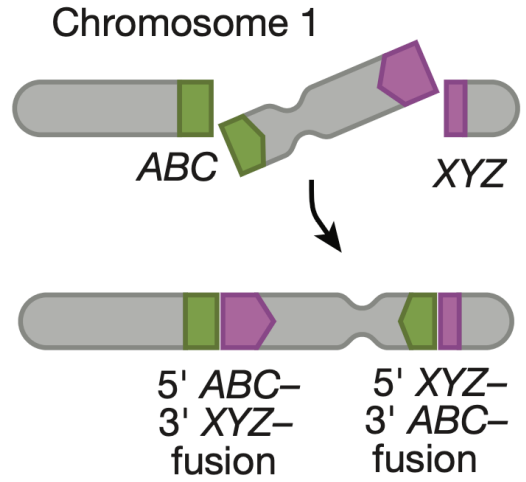
- Visualization is an important part of any data related project.

- Even if visualization is helpful in many situations, it is not needed in every situation and should not be produced because visualizations are easy to generate.

Examples For Clearly Structured Figures (1)

Visualizing chromosomal inversion that results in two fusion genes:

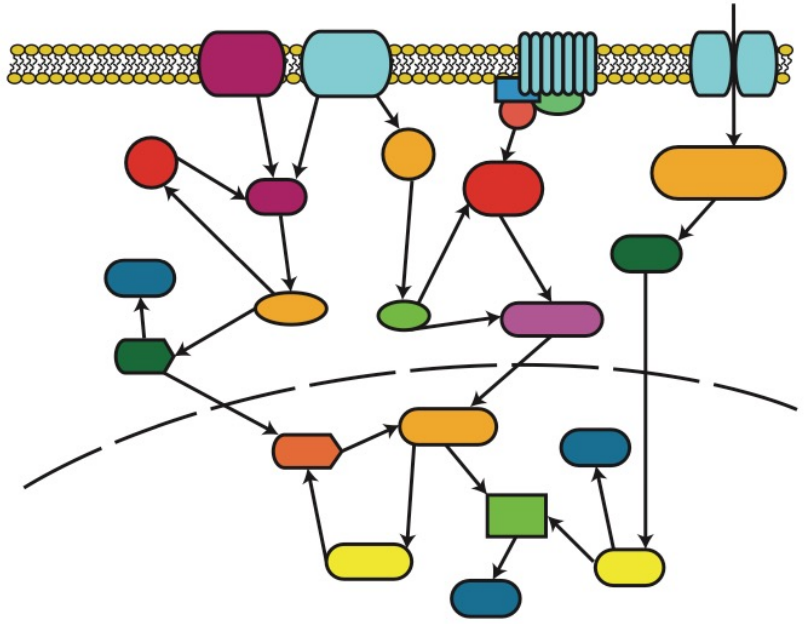


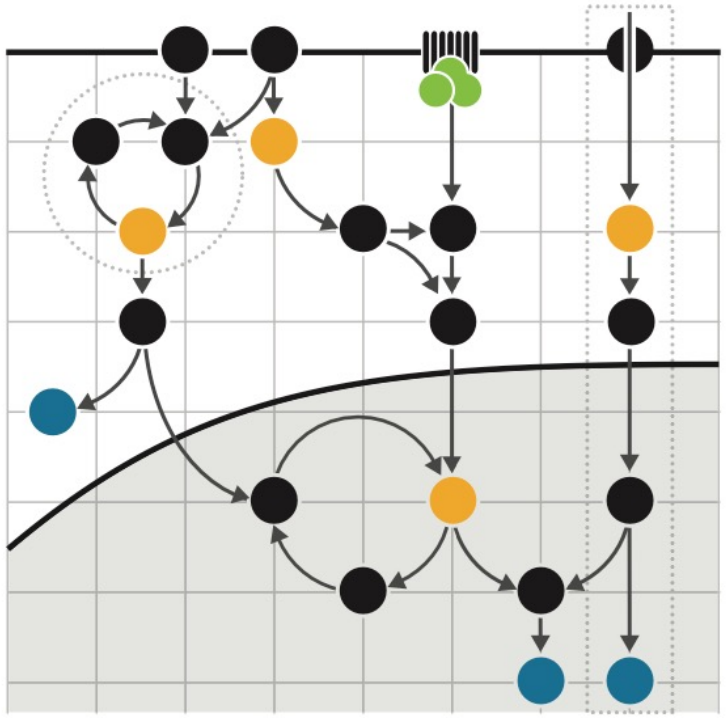


Wong, B. “Simplify to clarify” *Nature Methods* (2011)

Examples For Clearly Structured Figures (2)

Examples for pathway diagrams





- redundant visual encodings removed and main points emphasized by visual grouping
- Color and shape variations have been removed except for those highlighting a molecule of interest (orange), the products of the pathway (blue)

Wong, B. “Pathways” *Nature Methods* (2016)

Reading Exercise

Salience to relevance

- What is salience?
- Why do we need to consider the concept of salience when designing (scientific) visualizations?

Figure Generation

Vector vs. Raster Graphics

https://inkscape.org/~ozant/★art-bot

Vector vs. Raster Graphics

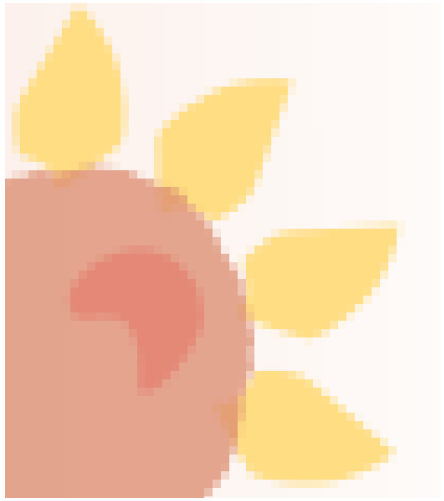
https://inkscape.org/~ozant/★art-bot

Raster Graphic

Vector Graphic

Vector vs. Raster Graphics

https://inkscape.org/~ozant/★art-bot



A raster graphic of width n and height m is described by an mn array where each position i, j stores the color information of the corresponding pixel of the graphic.

In contrast, vector graphics are described by graphical primitives like lines (described by two points) and circles (described by center and radius). Primitives can be described by mathematical functions, and they can be combined to build complex objects.

Objects in a vector graphic are fully described by their primitives and as a consequence of that vector graphics can be scaled without loss in quality.

Take home message: schematic drawings or data visualizations are ideally generated in a vector format.

Tools For Creating and Editing Vector Graphics

TECHNISCHE HOCHSCHULE DEGGENDORF THD

There are many tools available, but two prominent examples are Adobe Illustrator (if you have money and want to pay) and Inkscape (free and open-source). Both programs can be used to create or edit vector graphics.

https://inkscape.org

https://www.adobe.com/products/illustrator.html

Knowledge in these tools can be quite helpful even as a computer scientist. Composing bigger figures from multiple single plots, quickly adjust or align colors, add explanations or explanatory drawings can help to create better figures. Depending on the scenario, compositions can be done more easily in these programs rather than doing these edits via programming.

Figure Generation via Programming

Of course, as computer scientists we want to generate figures from data automatically via programming languages. In the first part of the course, you have already been introduced into *base R* and *ggplot2.* For the sake of comparison, let's look at a few others:

Base R [https://cran.r-project.org]

matplotlib (python) [https://matplotlib.org]

D3 (JavaScript) [https://d3js.org]

ggplot2 (R) [https://ggplot2.tidyverse.org]

seaborn (Python) [https://seaborn.pydata.org/examples/index.html]

plotly (R, Python, JavaScript) [https://plotly.com]

bokeh (Python) [https://docs.bokeh.org/en/latest/index.html]

and so on…. there are so many…

If There Are So Many, Which Should I Learn?

TECHNISCHE HOCHSCHULE DEGGENDORF THD

Typically, it makes sense to focus on a plotting library that belongs to the language that is used in the project. Since most data science is performed via R or/and Python it is beneficial to have experience in both. For R good starting points are base R and ggplot2 for Python matplotlib is very popular.

Comparing the left and right list, what do you think is the difference between them?

Base R [https://cran.r-project.org]

matplotlib (python) [https://matplotlib.org]

D3 (JavaScript) [https://d3js.org]

ggplot2 (R) [https://ggplot2.tidyverse.org]

seaborn (Python) [https://seaborn.pydata.org/examples/index.html]

plotly (R, Python, JavaScript) [https://plotly.com]

bokeh (Python) [https://docs.bokeh.org/en/latest/index.html]

It’s hard to draw a clear line, but some libraries are more low-level (e.g. base R, D3, matplotlib), whereas others are more high-level (e.g. seaborn, ggplot2). It’s a trade-off. The more high-level a library is, the more you need to stick to the decisions and the offered functionality. Extending or changing is often cumbersome. Low-level offers more flexibility at the cost of spending more time to create visual pleasing plots.

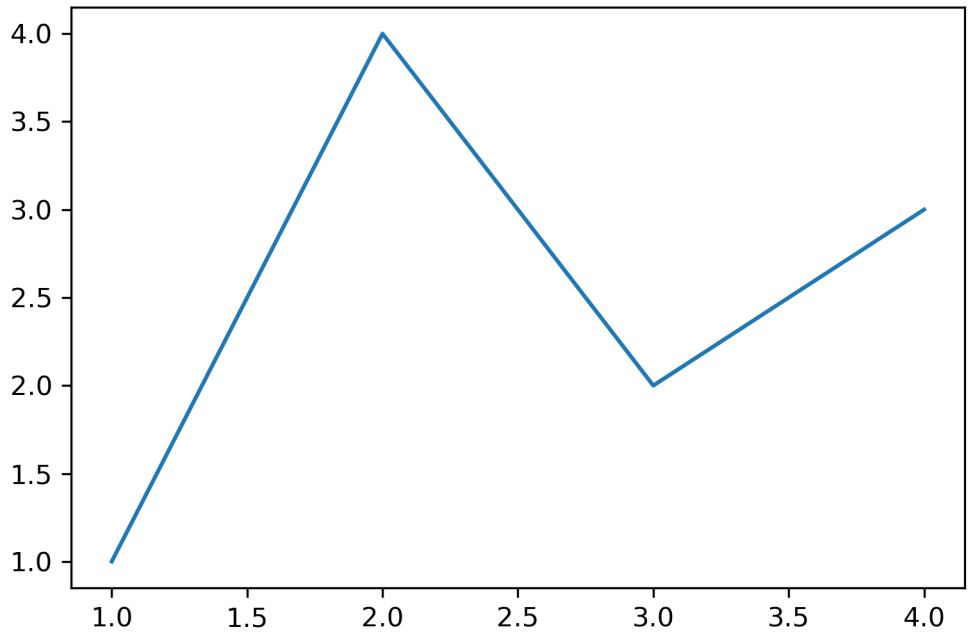
DEGGENDORF INSTITUTE OF TECHNOLOGY

Matplotlib

Matplotlib

TECHNISCHE HOCHSCHULE DEGGENDORF THD

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

fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```



matplotlib user guide:

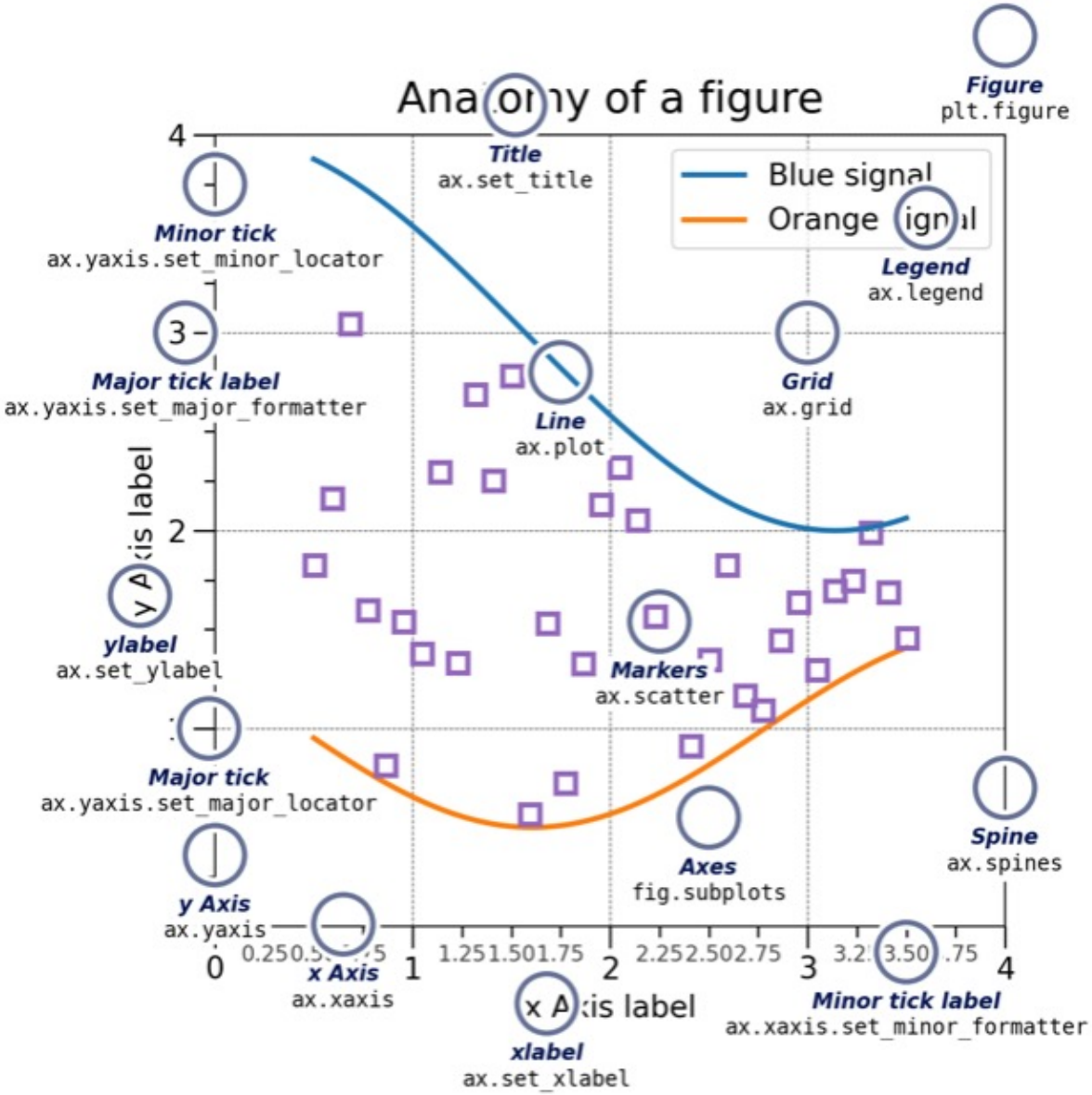
https://matplotlib.org/stable/tutorials/introductory/usage.html

Matplotlib

Multiple ways exist to use Matplotlib. The most supported and in my opinion most useful way to use Matplotlib is by using the suggested object-oriented (OO) style.

A Matplotlib figure is an object that is composed of several other objects.

Each of these objects can be manipulated by using the object's functions.



https://matplotlib.org/stable/tutorials/introductory/usage.html

Figures and Axes

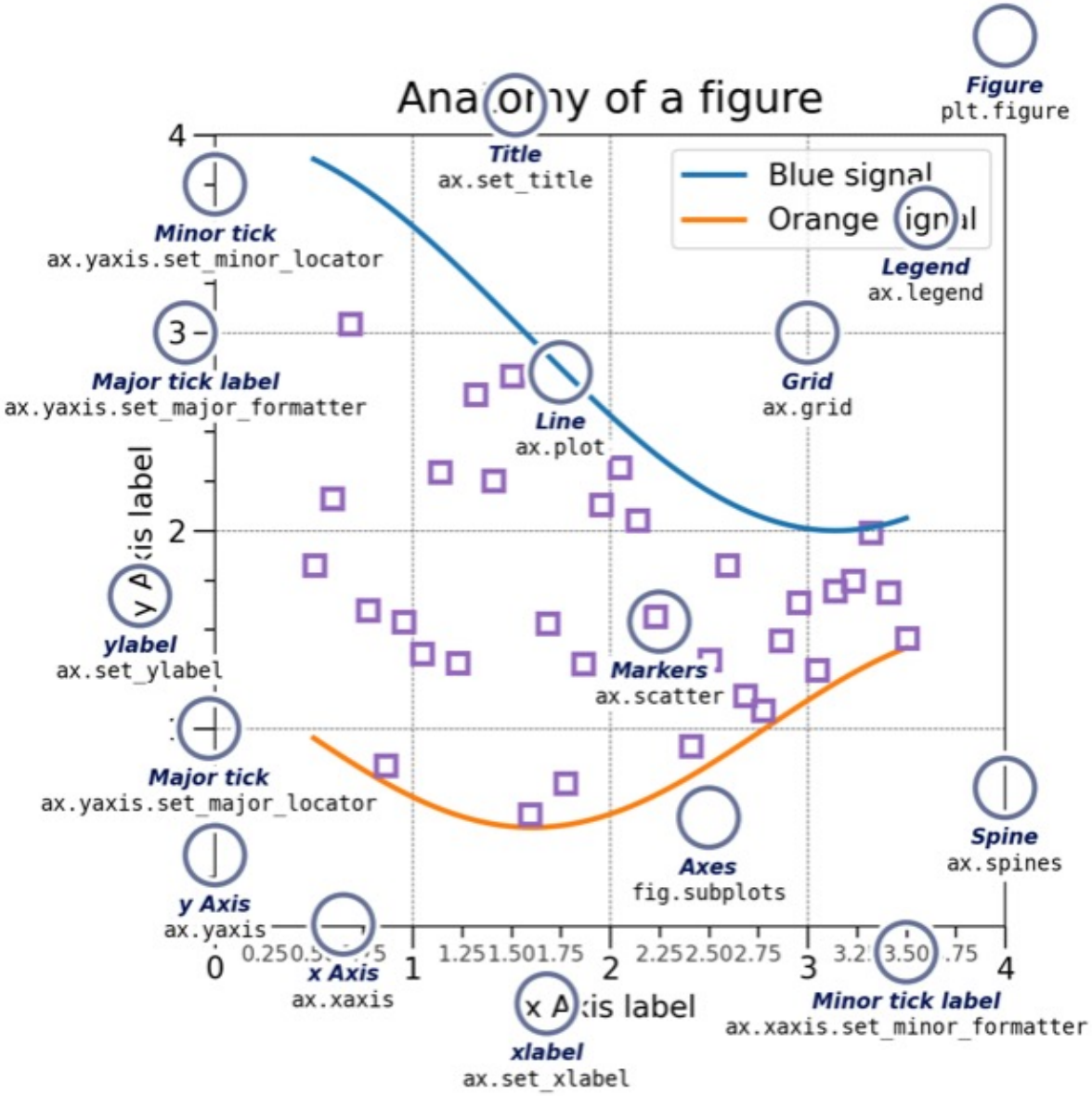
A **Figure** object contains everything that is displayed. The area where graphical objects are drawn is an **Axes**. Consequently, **Figures** consists of **Axes** and thus a **Figure** can have multiple **Axes**.

The distinction between a **Figure** and an **Axes** is different compared to some other plotting libraries.

Typically, **Figure** and **Axes** objects are created together.

Axes are the major object and entry point to work in the OO-style and are used to actually draw elements.

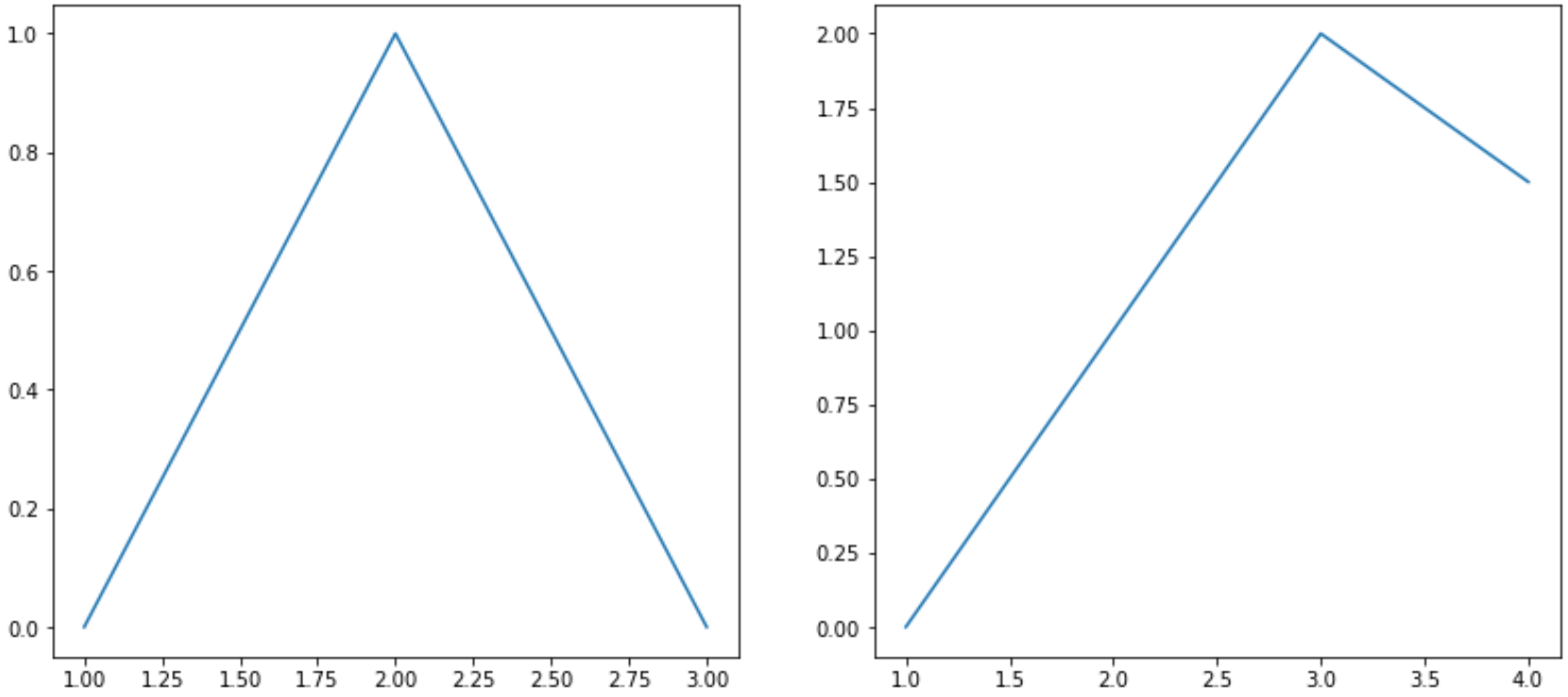
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```



https://matplotlib.org/stable/tutorials/introductory/usage.html

Figures and Axes

```
fig, axs = plt.subplots(1,2)
fig.set_size_inches(13.5, 6)
axs[0].plot([1,2,3], [0,1,0])
axs[1].plot([1,2,3,4], [0,1,2,1.5]);
```



In this example a **Figure** objects has two **Axes** objects.

Figures and Axes

Generate random data with numpy to practice plotting:

```
np.random.seed(2)
x1 = np.random.normal(loc=5, scale=2, size=100)
noise1 = np.random.normal(loc=0, scale=2, size=100)
y1 = 3 +  1.5*x1 + noise1

x2 = np.random.normal(loc=5, scale=2, size=100)
noise2 = np.random.normal(loc=0, scale=2, size=100)
y2 = 3 + -1.2*x2 + noise2
```

Figures and Axes

```
fig, ax = plt.subplots()
fig.set_size_inches(6,6)

color_A = "#009e73"
color_B = "#56b3e9"

ax.set_xlim([-1,10]) #set axis limits manually
ax.set_ylim([-11,23])

ax.scatter(x1, y1, s=50, color=color_A, label="Data A") #draw points
ax.scatter(x2, y2, s=50, color=color_B, label="Data B")

#draw lines
x1_line = np.array([-1,11])
y1_line = 3 +  1.5*x1_line
ax.plot(x1_line, y1_line, color=color_A, linewidth=2)
ax.plot(x1_line, y1_line+3, color=color_A, linestyle="dotted")
ax.plot(x1_line, y1_line-3, color=color_A, linestyle="dotted")

x2_line = np.array([-1,11])
y2_line = 3 + -1.2*x2_line
ax.plot(x2_line, y2_line, color=color_B, linewidth=2)
ax.plot(x2_line, y2_line+3, color=color_B, linestyle="dotted")
ax.plot(x2_line, y2_line-3, color=color_B, linestyle="dotted")

#add text
ax.text(2,16, r'$y=3+1.5x$', fontsize=13, color=color_A)

#set labels, title and grid
ax.grid()
ax.set_xlabel("X")
ax.set_ylabel("y")
ax.legend()
ax.set_title("My Figure");
fig.savefig('01_example_scatter.png', format='png', dpi=96)
```

Helper Functions

To create your own plots, it is common to generate custom functions that take an **Axes** object as well as the data that is sought to be plotted as input and that performs the actual plotting. Thus, the plotting function can be used on any new data.

```
def plot_function(ax, data1, data2):
    #do necessary transformations/computations
    #ax.scatter(..)
    #ax.plot(...)..
    ;
```

Alternatively, functions can be written that perform the complete plot and that return the **Figure** and **Axes** to allow for custom changes/additions and or export of a plot.

```
def make_plot(lots_of_data):
    fig, ax = plt.subplots()
    #do all you need to do
    #create complete figure
    return fig, ax
```

Programming Exercise

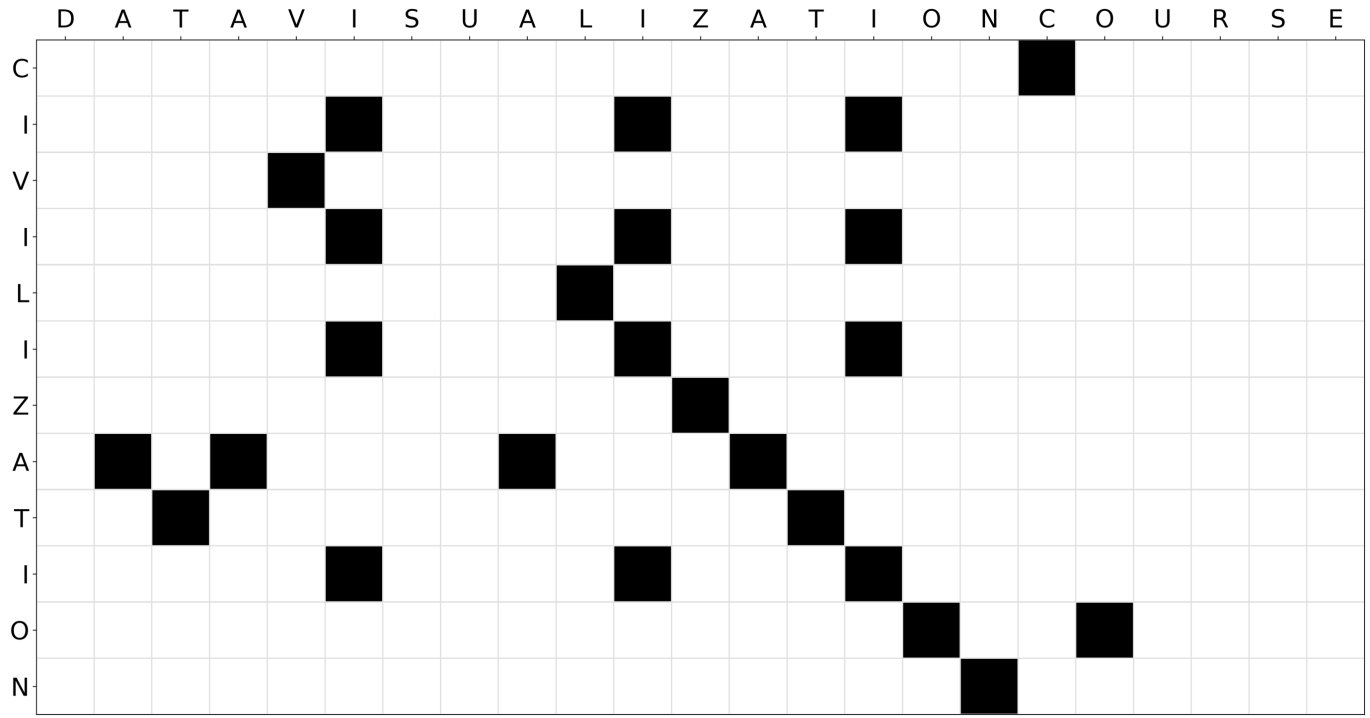
- Create a dot plot from scratch using Python/matplotlib

A Dot Plot Visualizes Sequence Similarity

A simple and visual way of comparing two sequences s_1 and s_2 of length n and m with each other is a dot plot. The sequences s_1 and s_2 span a $n \times m$ matrix M where a dot is plotted for $M_{i,j}$ if $s_1[i] = s_2[j]$.

$$M_{i,j} = \begin{cases} 1, \text{if } s_1[i] = s_2[j] \\ 0, \text{ if } s_1[i] \neq s_2[j] \end{cases}$$

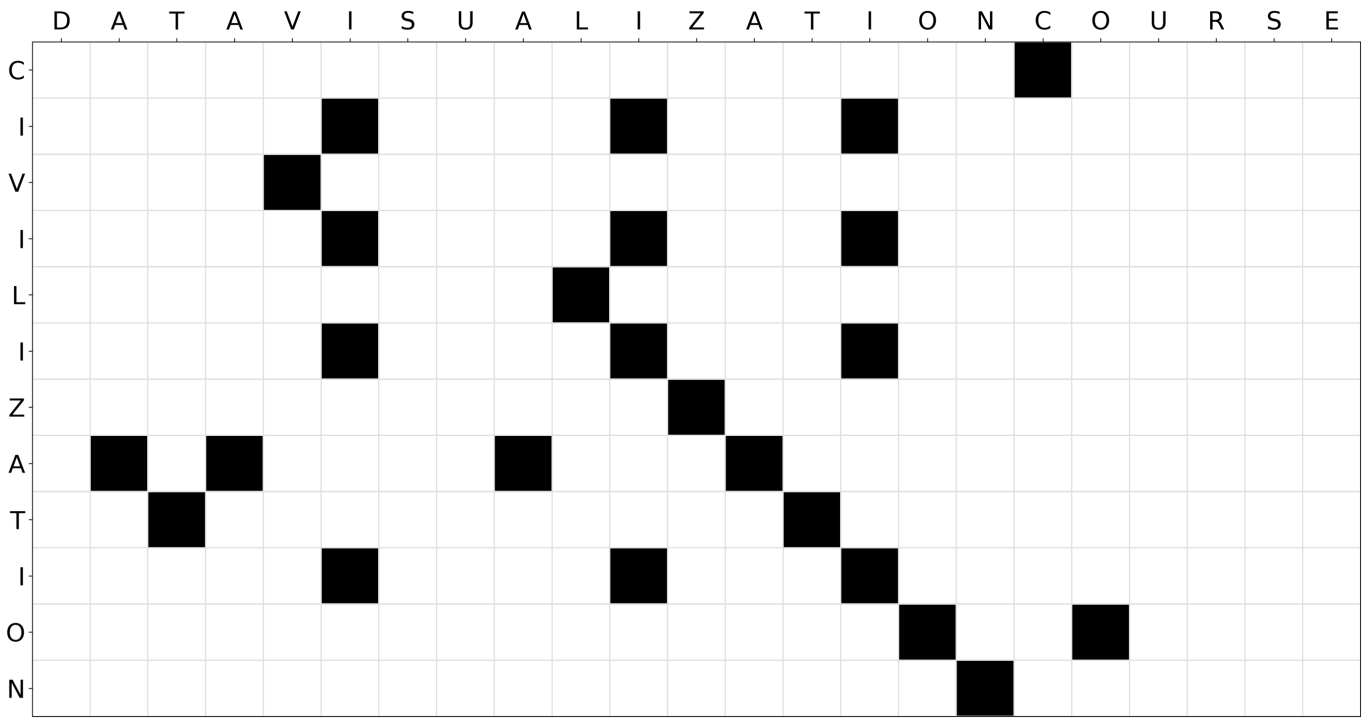
The figure below shows a dot plot for $s_1 = CIVILIZATION$ and $s_2 = DATAVISUALIZATIONCOURSE$.

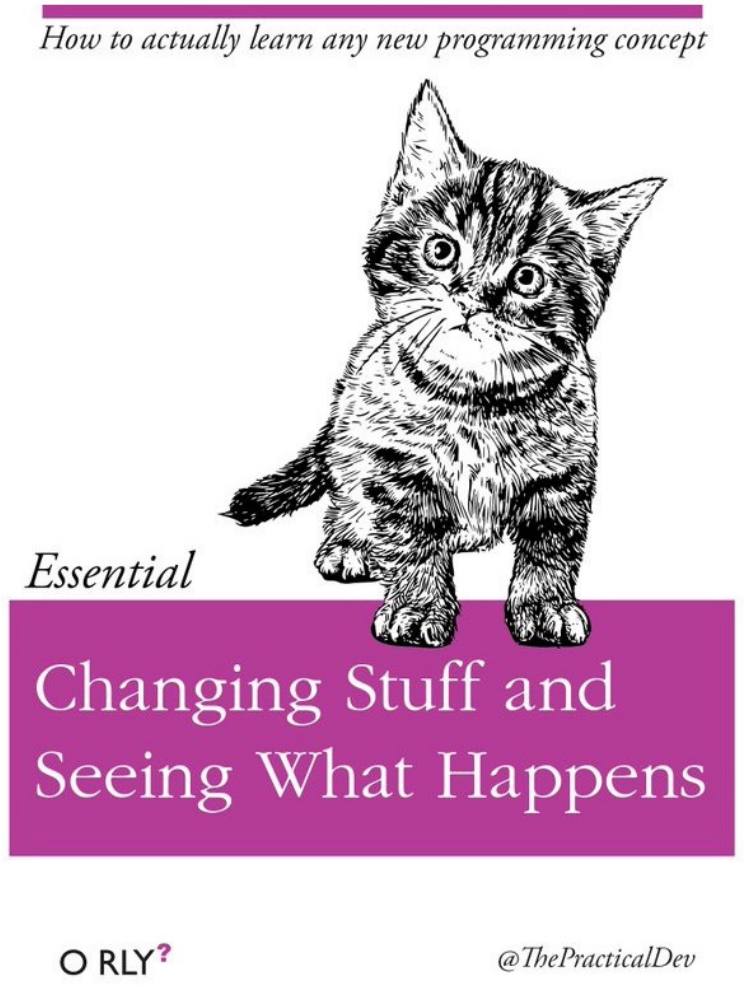


A Dot Plot Visualizes Sequence Similarity

Matplotlib user guide:

https://matplotlib.org/stable/tutorials/introductory/usage.html





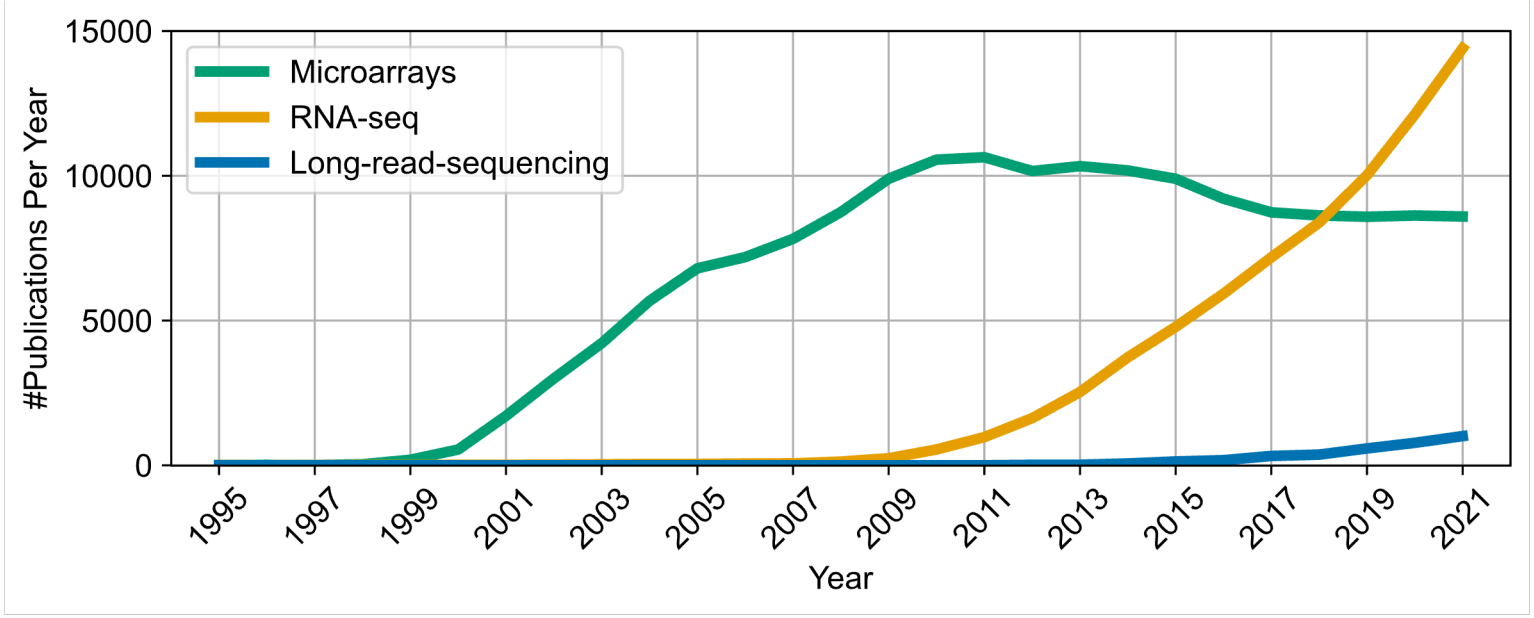
https://twitter.com/ThePracticalDev/status/720257210161311744

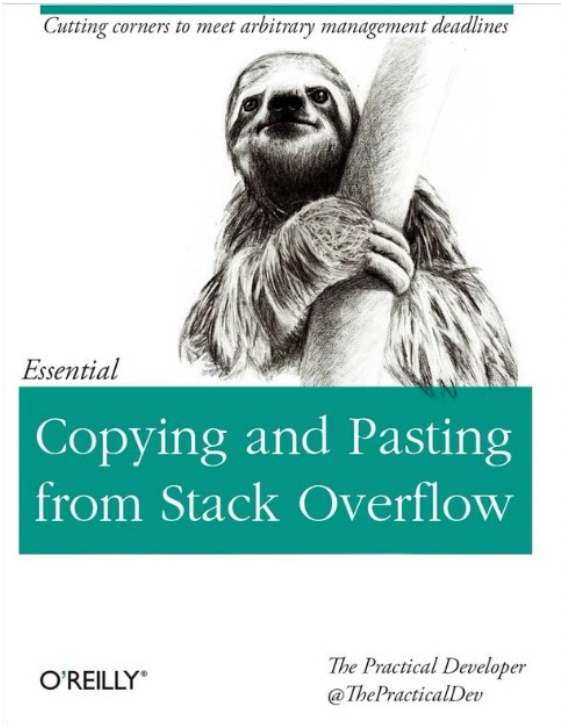
Programming Exercise

- Create a line plot to visualize publications per year indexed in the pubmed/medline database

Line Plot to Visualize Publication Counts

- Download data from iLearn
- Find a solution to read in the data
- Write a function that produces a plot like shown below





https://github.com/thepracticaldev/orly-full-res

DEGGENDORF INSTITUTE OF TECHNOLOGY DIT

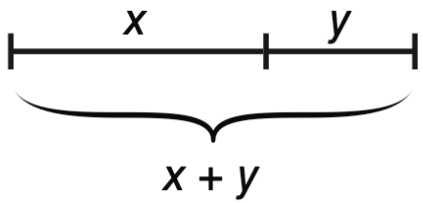
Layout and White Space

Layout and the Golden Ratio

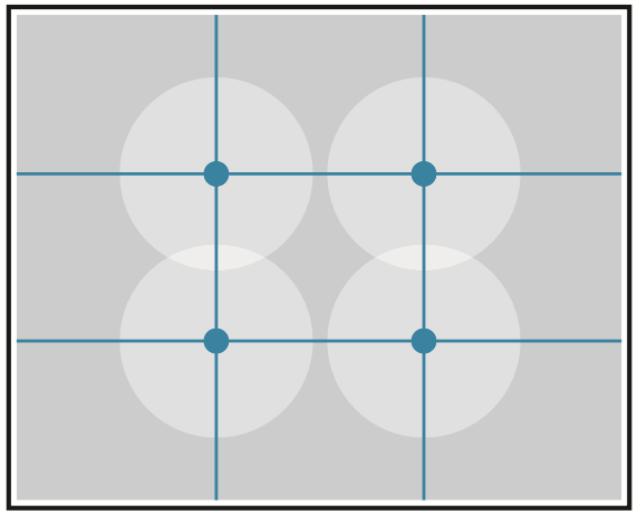
- What is layout? Layout is the act of arranging text and images on the page/slide/poster according to an overall aesthetic scheme and for the purpose of clarifying a presentation.

- Why do we need layout? Well-structured content can guide readers through complex information and unstructed material can confuse or, worse, it needs more time for the reader arrange the items into correct order rather than focus on the message.

- A helpful guidedance for the arrangement of elements in a figure is the ‘rule of thirds’ also known as the ‘golden ratio’ that became popular during artists of the renaissance (and remains popular since).



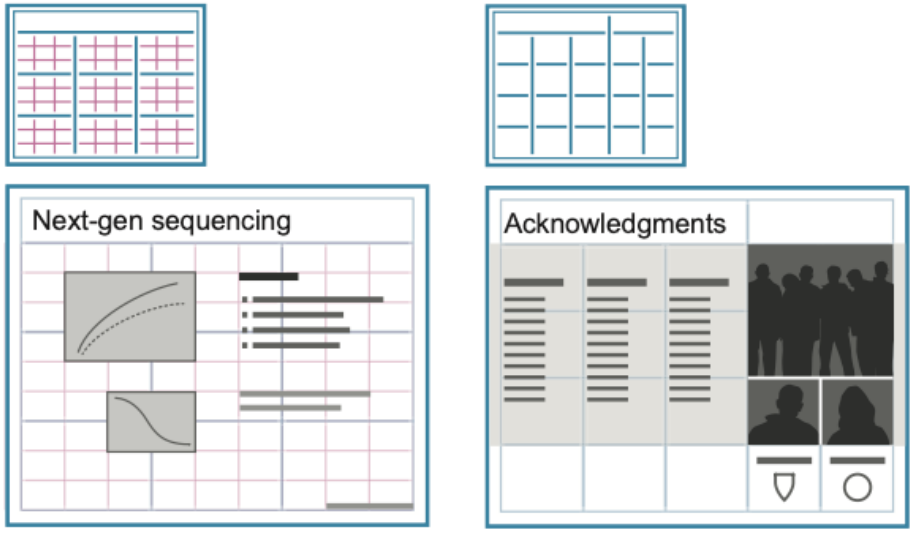
The golden section is a line segment divided by the golden ratio 13:8 such that ($x + y$) is to x as x is to y.



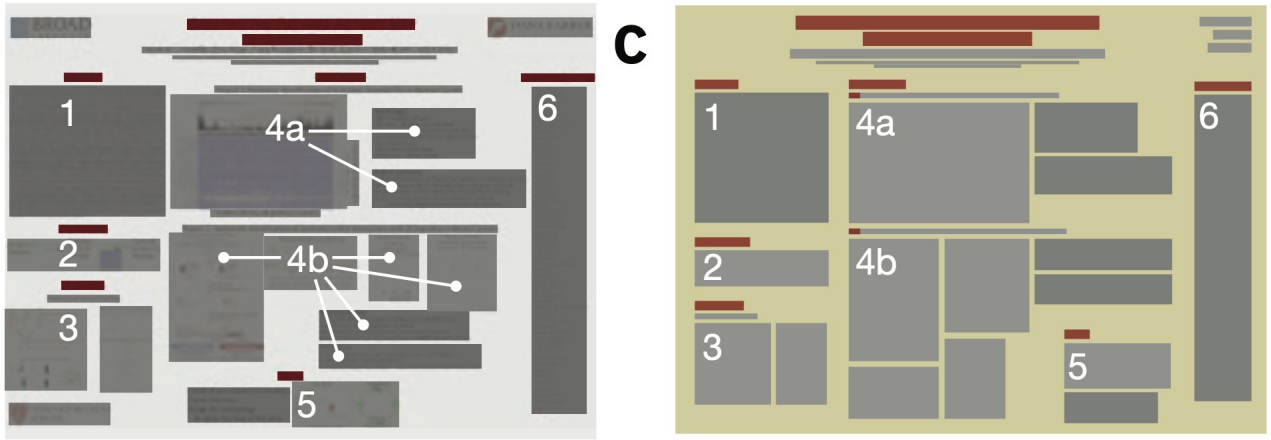
Wong, B. “Layout” *Nature Methods* (2011)

Using Grid Lines Helps To Arrange And Structure Content

- Grids help you to anchor elements and to align elements properly.
- Gestalt principles can help you to group elements and give structure.
- White spaces are an important tool to properly structure content.



Wong, B. “Layout” *Nature Methods* (2011)



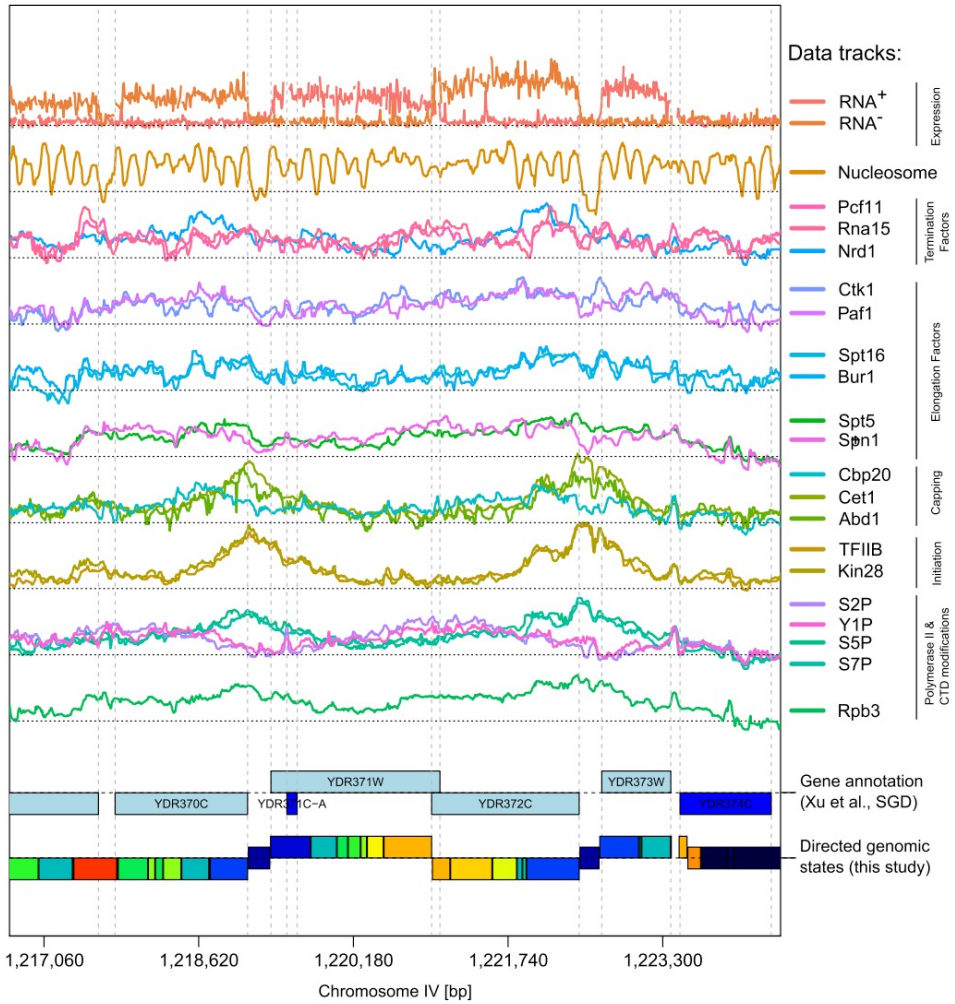
Wong, B. “Negative space” *Nature Methods* (2011)

Reading Exercise:

Managing deep data in genome browsers

1. Why is data visualization challenging in modern life science research?
2. What can be done to reduce visual complexity and what are potential drawbacks?

Summarization Example



Chromatin immunoprecipitation in conjunction with hidden Markov models (HMMs) segment the genome into discrete states that can be related to DNA-associated protein complexes.

Zacher,B et al. "Annotation of genomics data using bidirectional hidden Markov models unveils variations in Pol II transcription cycle." *Molecular systems biology* 10.12 (2014): 768.

DEGGENDORF INSTITUTE OF TECHNOLOGY

Programming Exercise

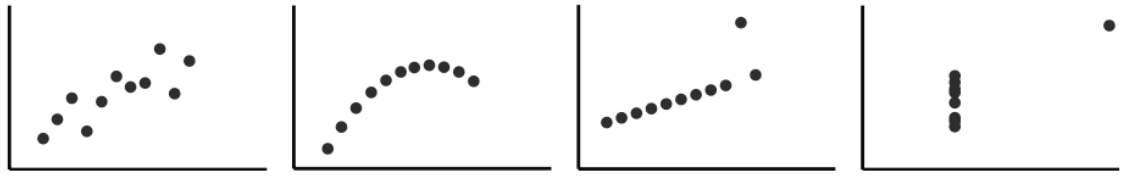
Read And Plot Data

TECHNISCHE HOCHSCHULE DEGGENDORF THD

- Go to iLearn and download ‘04_data.csv’
- Load the data into a Jupyter notebook
- Use y=0 for the x1 column and y=1 for the x2 column and generate a scatter plot the data.
  - Do you spot a difference between x1 and x2?
- Calculate the means from x1 and x2 and compare them. Do they appear to be different?
- Go to https://seaborn.pydata.org/installing.html and install the seaborn package.
- Create a boxplot of x1 and x2 using seaborn. Do the distributions appear to be different?
  - Check https://seaborn.pydata.org/generated/seaborn.boxplot.html#seaborn.boxplot for further guidance
- Create a violin or kde plot of x1 and x2 using seaborn. What’s the difference compared to the previously produced boxplot?

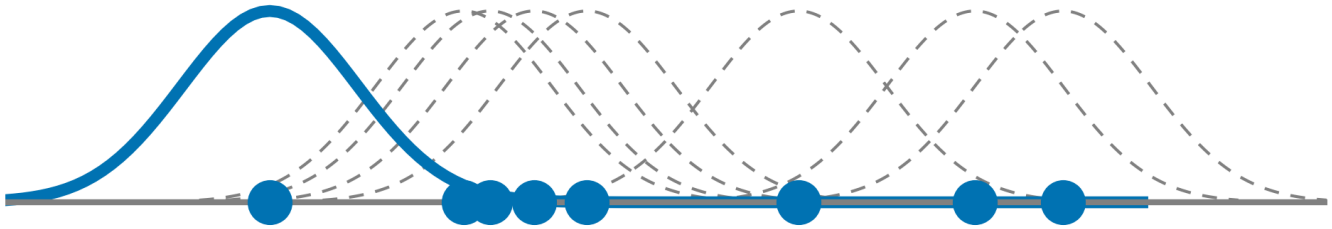
Ancombe’s Quartet

I		II		III		IV	
x	*y*	*x*	*y*	*x*	*y*	*x*	*y*
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.10	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.10	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89



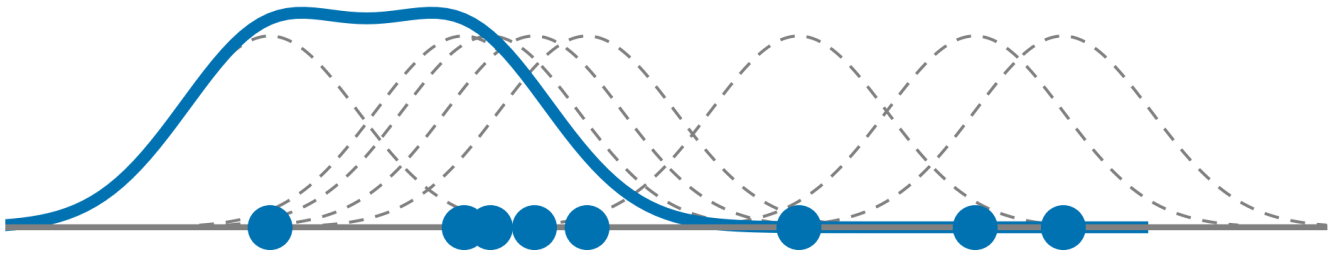
Raw data as well as summary statistics can be misleading if you rely exlcusively on these. The shown four data sets known as Ancombe’s quartet have the same means, standard deviations, correlation coefficients, regression lines etc. but still the data sets are fundamentally different as can be seen by visualization. During data exploration don’t rely on a single perspective.

Kernel Density Estimation In A Nutshell



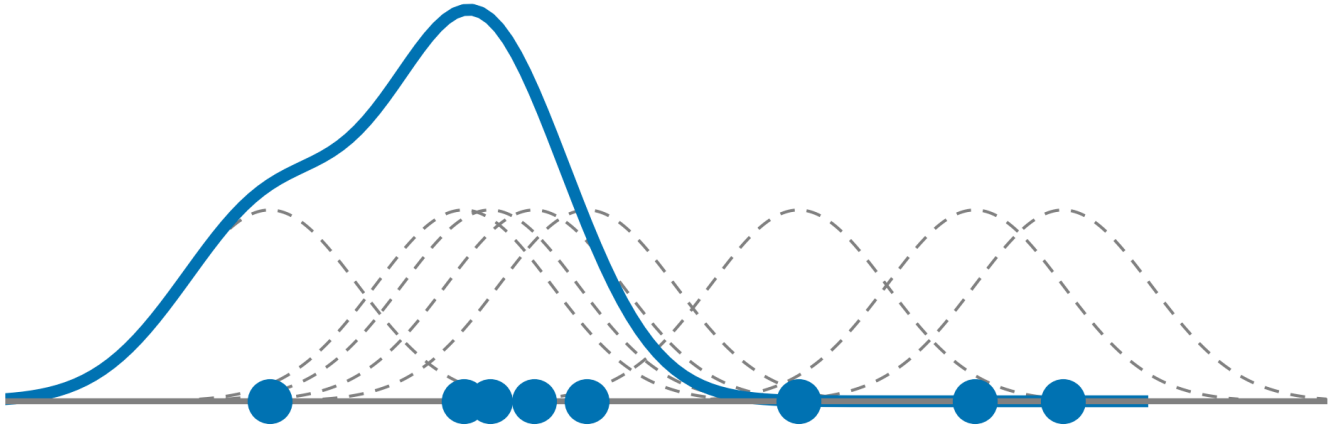
The density estimate at each point is the average contribution from each of the kernels at that point.

Kernel Density Estimation In A Nutshell



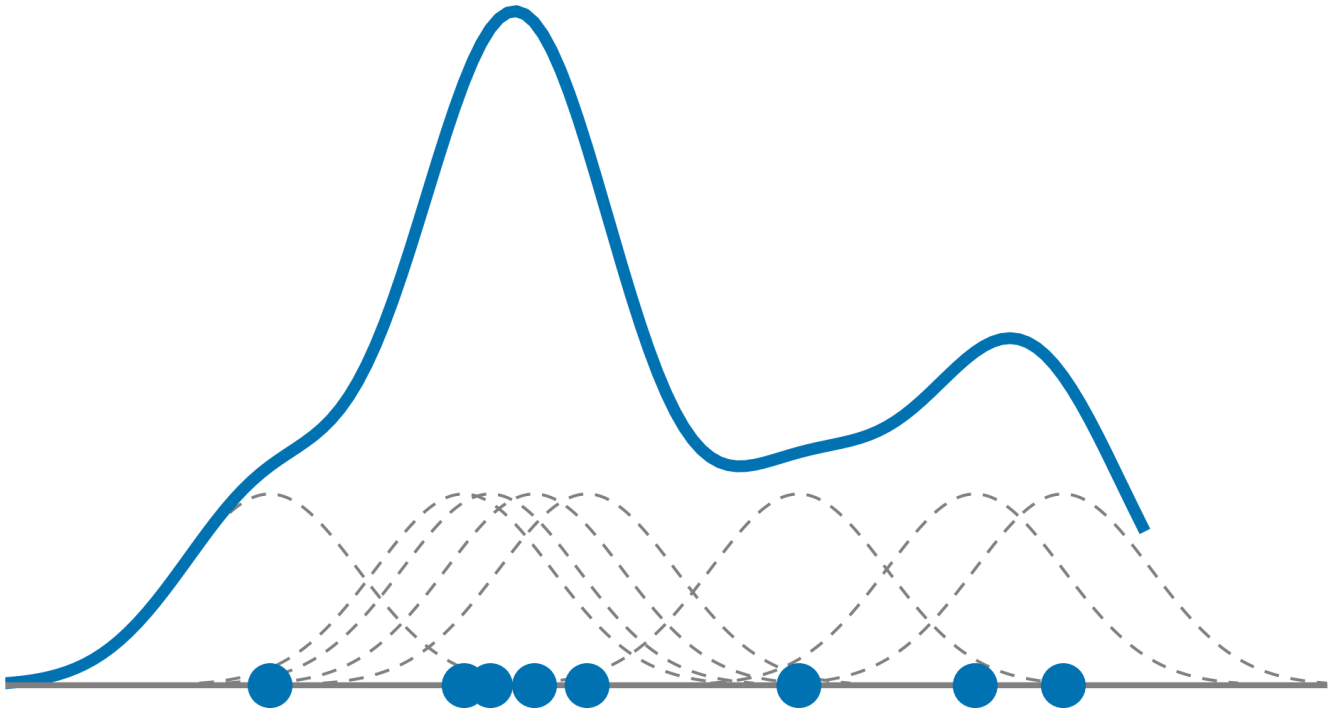
The density estimate at each point is the average contribution from each of the kernels at that point.

Kernel Density Estimation In A Nutshell



The density estimate at each point is the average contribution from each of the kernels at that point.

Kernel Density Estimation In A Nutshell



The density estimate at each point is the average contribution from each of the kernels at that point.

Read And Plot Data

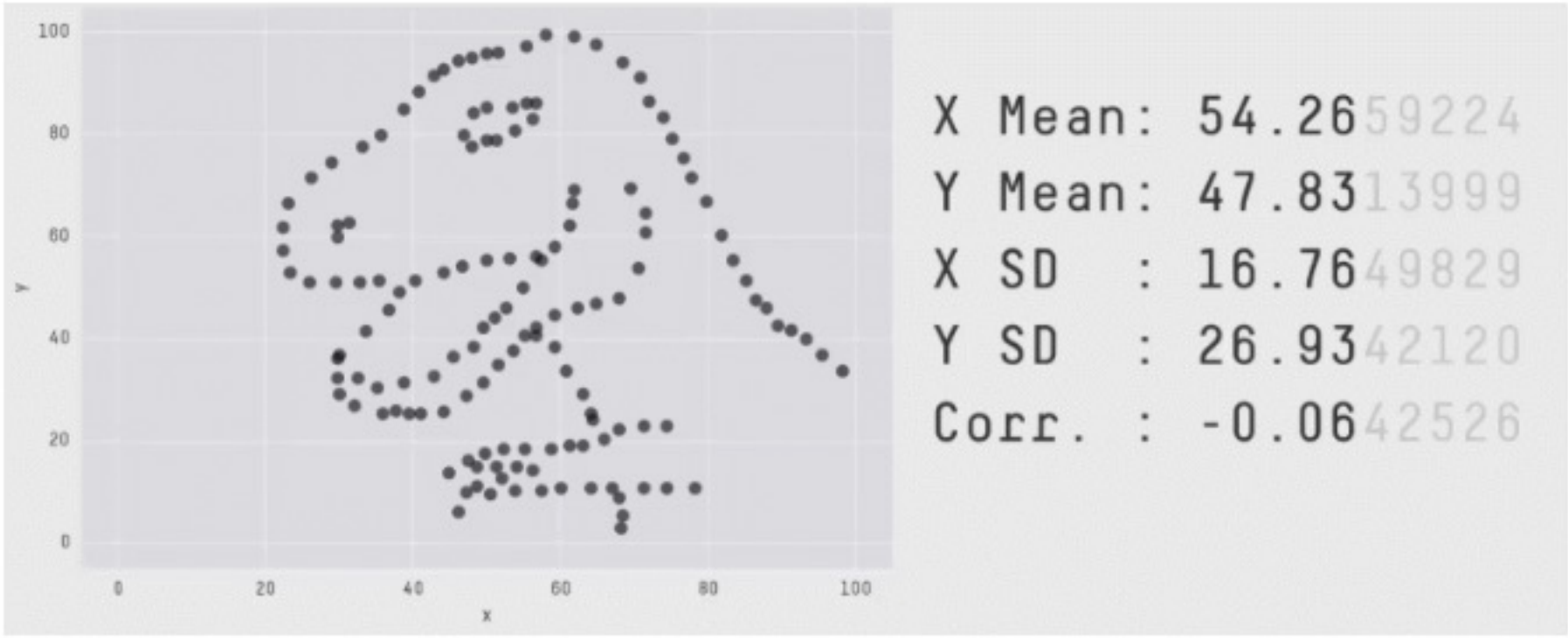
TECHNISCHE HOCHSCHULE DEGGENDORF THD

- Go to iLearn and download ‘04_data.csv’
- Load the data into a Jupyter notebook
- Plot the data in 1D. Use y=0 for the x1 column and y=1 for the x2 column and plot the data.
  - Do you spot a difference between x1 and x2?
- Calculate the means from x1 and x2 and compare them. Do they appear to be different?
- Go to https://seaborn.pydata.org/installing.html and install the seaborn package.
- Create a boxplot of x1 and x2 using seaborn. Do the distributions appear to be different?
  - Check https://seaborn.pydata.org/generated/seaborn.boxplot.html#seaborn.boxplot for further guidance
- Create a violin or kde plot of x1 and x2 using seaborn. What’s the difference compared to the previously produced boxplot?

Never Trust Summary Statistics Alone

https://www.autodesk.com/research/publications/same-stats-different-graphs

Never Trust Summary Statistics Alone



https://www.autodesk.com/research/publications/same-stats-different-graphs

Reading Exercise

Elements of visual style

1. What is the connection between writing and figure generation?

2. List and explain the shown examples of effective written communication applied to the process of figure creation.

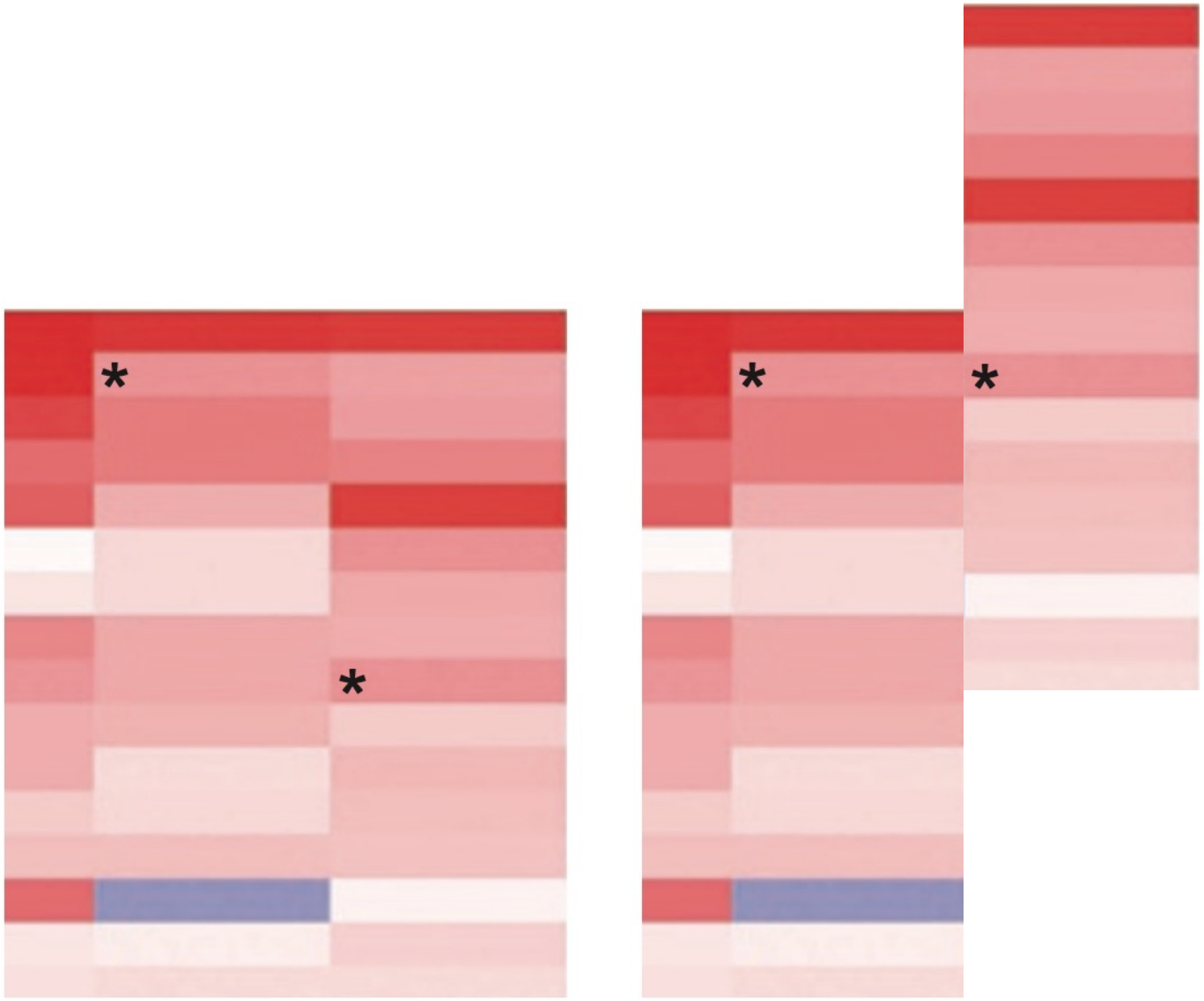
DEGGENDORF INSTITUTE OF TECHNOLOGY

Color

Color

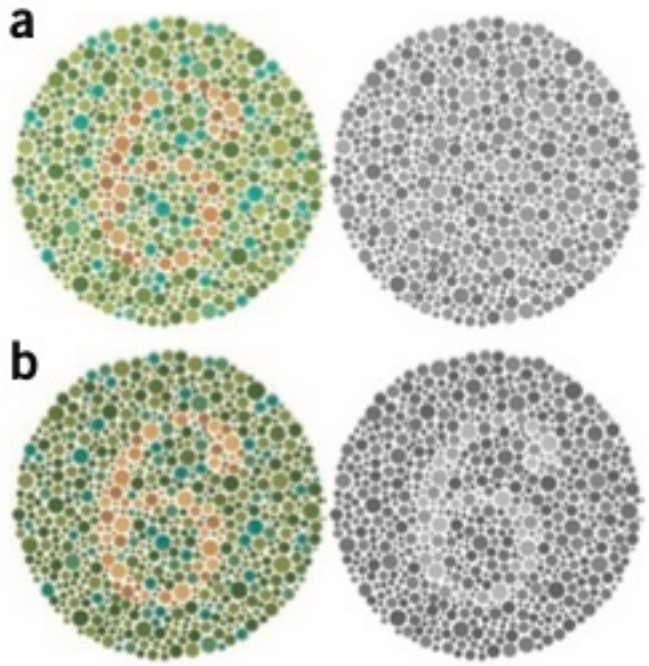
TECHNISCHE HOCHSCHULE DEGGENDORF THD

Color



Wong, B. “Color coding” *Nature Methods* (2010)

Color Blindness



Wong, B. “Color blindness” *Nature Methods* (2011)

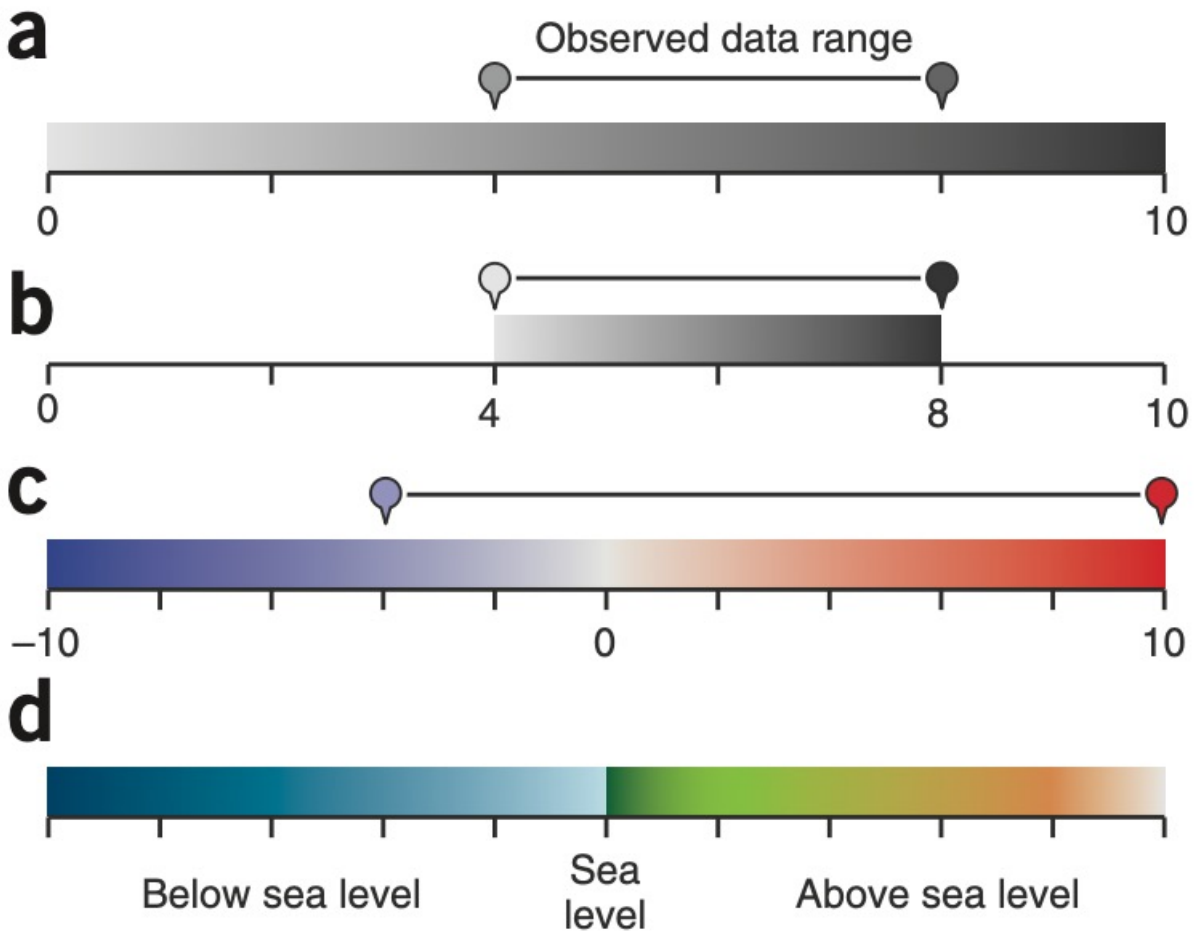
Ishihara color vision test plates. **a)** colors are only different in hue. **b)** colors can be separated easier if the colors also change in lightness and saturation

Color	Color name	RGB (1–255)	CMYK (%)	P	D
	Black	0, 0, 0	0, 0, 0, 100		
	Orange	230, 159, 0	0, 50, 100, 0		
	Sky blue	86, 180, 233	80, 0, 0, 0		
	Bluish green	0, 158, 115	97, 0, 75, 0		
	Yellow	240, 228, 66	10, 5, 90, 0		
	Blue	0, 114, 178	100, 50, 0, 0		
	Vermillion	213, 94, 0	0, 80, 100, 0		
	Reddish purple	204, 121, 167	10, 70, 0, 0		

Wong, B. “Color blindness” *Nature Methods* (2011)

Colors optimized for color-blind individuals. P and D indicate simulated colors as seen by individuals with protanopia and deuteranopia.

Mapping quantitative data to color



(a) Plotting data with only positive or negative values, an intuitive encoding is a sequential color map that varies only the lightness from 10% to 90% black.

(b) Rescaling the color map to the observed minimum and maximum. This is applicable if the theoretical range (e.g. 0,10) is not of interest.

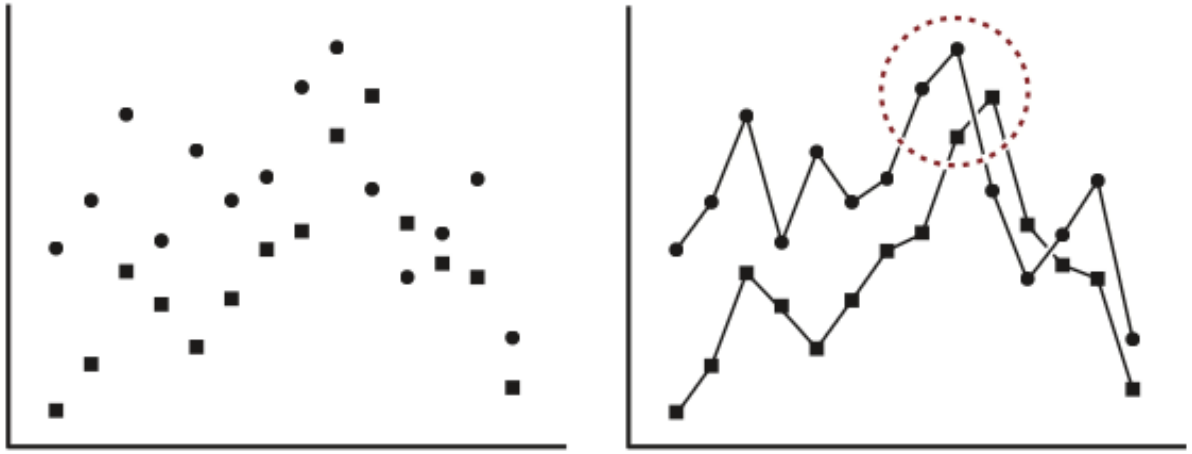
(c) Scenario where data containing both positive and negative values and in which the lower and upper ends of the distribution as well as zero need to be distinguished. A diverging (or bipolar) color schema that employs both color hue and color saturation is effective. Color hue makes a distinction between positive and negative values (for example, red and blue) and color saturation indicates the relative scale and no saturation represents zero. (d) The interpretation of zero or other key values can further influence the choice of color keys.

Gehlenborg & Wong “Mapping quantitative data to color” *Nature Methods* (2012)

DEGGENDORF INSTITUTE OF TECHNOLOGY

Programming Exercise

Creating Multiple Subplots



Wong “Gestalt principles (Part 1)” *Nature Methods* (2010)

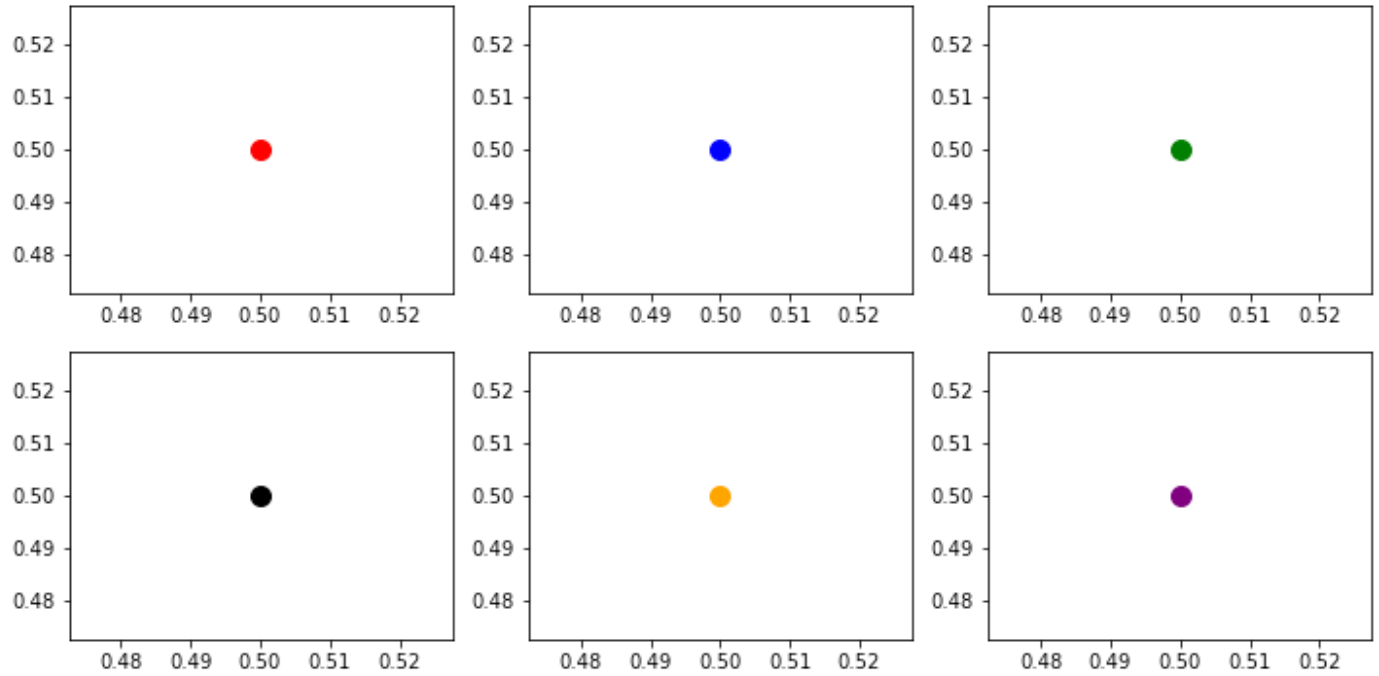
Creating Multiple Subplots

columns

rows

```
fig, axs = plt.subplots(2,3)

fig.set_size_inches(12,6)
axs[0,0].scatter(0.5,0.5,s=100,c='red')
axs[0,1].scatter(0.5,0.5,s=100,c='blue')
axs[0,2].scatter(0.5,0.5,s=100,c='green')

axs[1,0].scatter(0.5,0.5,s=100,c='black')
axs[1,1].scatter(0.5,0.5,s=100,c='orange')
axs[1,2].scatter(0.5,0.5,s=100,c='purple')
```

```
<matplotlib.collections.PathCollection at 0x7fd0c329d8b0>
```



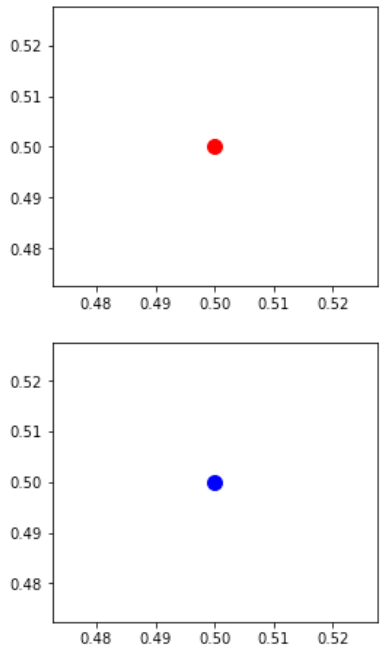
Creating Multiple Subplots

TECHNISCHE HOCHSCHULE DEGGENDORF THD

Vertical

```
fig, axs = plt.subplots(3,1)

fig.set_size_inches(4,12)
axs[0].scatter(0.5,0.5,s=100,c='red')
axs[1].scatter(0.5,0.5,s=100,c='blue')
axs[2].scatter(0.5,0.5,s=100,c='green')
```

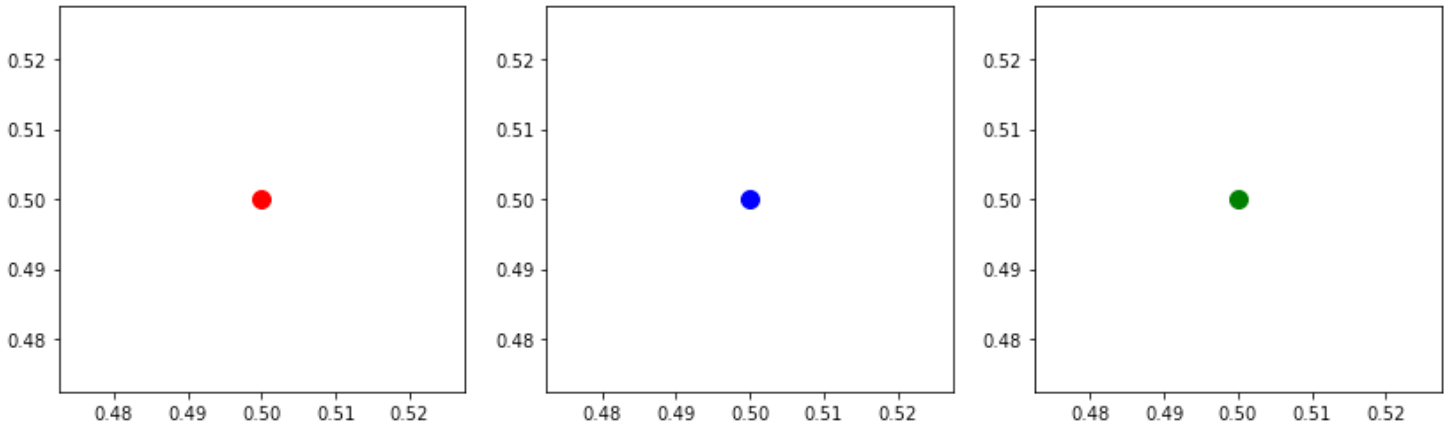
```
<matplotlib.collections.PathCollection at 0x7fd0d7b0a5b0>
```



Horizontal

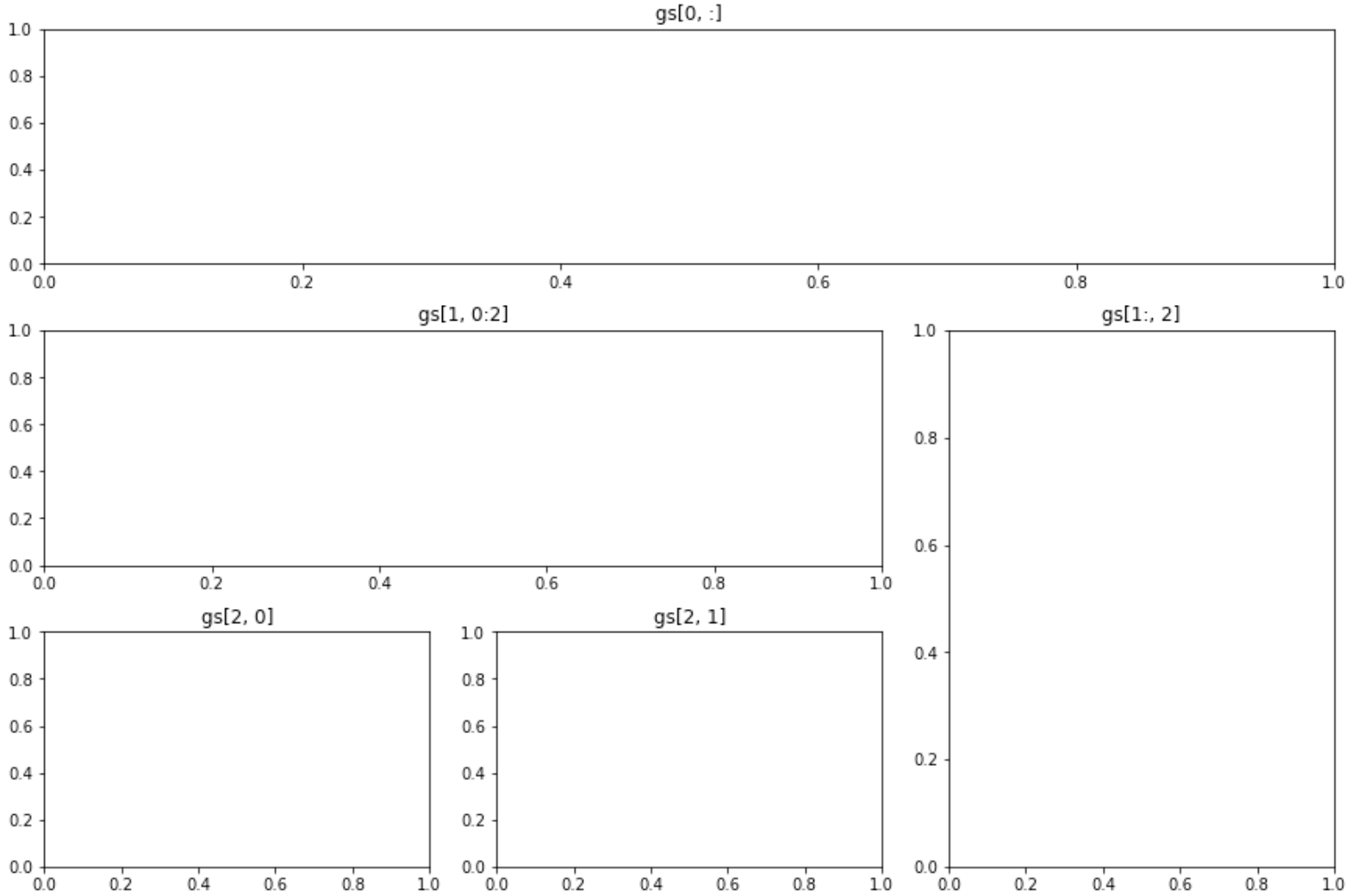
```
fig, axs = plt.subplots(1,3)

fig.set_size_inches(14,4)
axs[0].scatter(0.5,0.5,s=100,c='red')
axs[1].scatter(0.5,0.5,s=100,c='blue')
axs[2].scatter(0.5,0.5,s=100,c='green')
```

```
<matplotlib.collections.PathCollection at 0x7fd0b28fc160>
```



Use GridSpec for Custom Layouts

```
import matplotlib.gridspec as gridspec
fig = plt.figure(constrained_layout=True)
fig.set_size_inches(12,8)
gs = fig.add_gridspec(3, 3)

f_ax1 = fig.add_subplot(gs[0, :])
f_ax1.set_title('gs[0, :]')
f_ax2 = fig.add_subplot(gs[1, 0:2])
f_ax2.set_title('gs[1, 0:2]')
f_ax3 = fig.add_subplot(gs[1:, 2])
f_ax3.set_title('gs[1:, 2]')
f_ax4 = fig.add_subplot(gs[2, 0])
f_ax4.set_title('gs[2, 0]')
f_ax5 = fig.add_subplot(gs[2, 1])
f_ax5.set_title('gs[2, 1]')
```

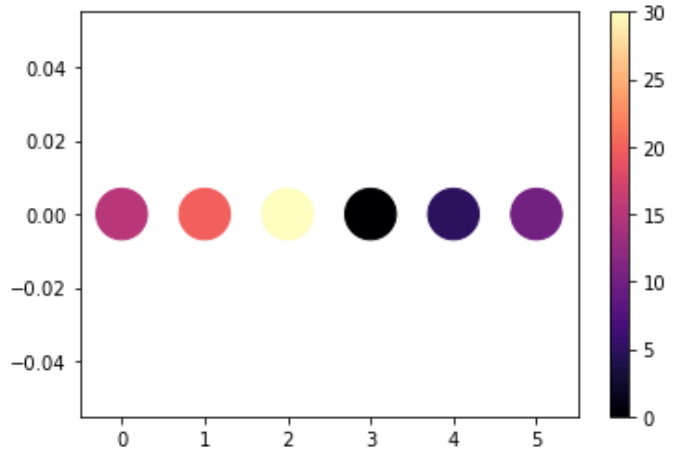


according to:
https://matplotlib.org/stable/tutorials/intermediate/gridspec.html

Colormaps

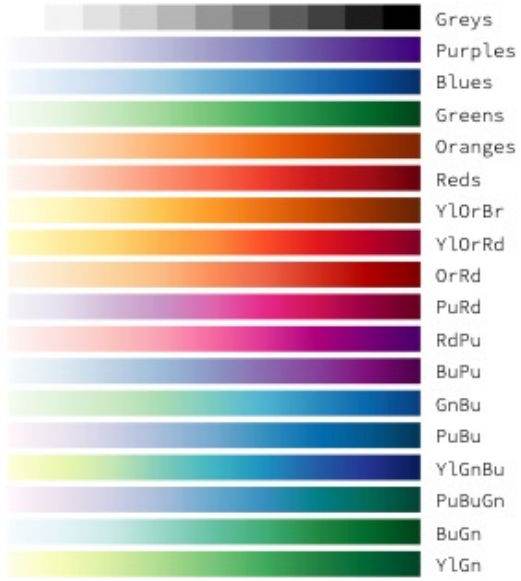
```
fig, ax = plt.subplots()
ax.set_xlim([-.5,5.5])
points = ax.scatter(np.arange(0,6,1),np.zeros(6), c=[15,20,30,0,5,10],
           s=750, cmap=plt.get_cmap('magma'), vmin=0, vmax=30)
fig.colorbar(points, ax=ax)
```

```
<matplotlib.colorbar.Colorbar at 0x7fece0ef2100>
```

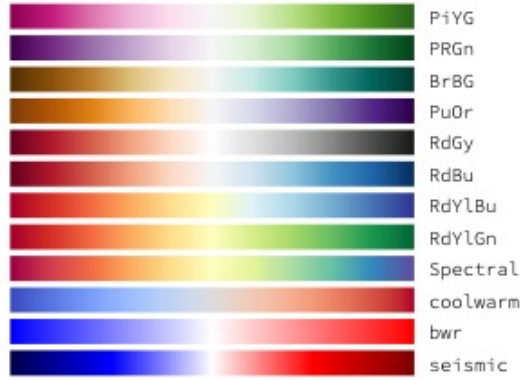


Uniform colormaps

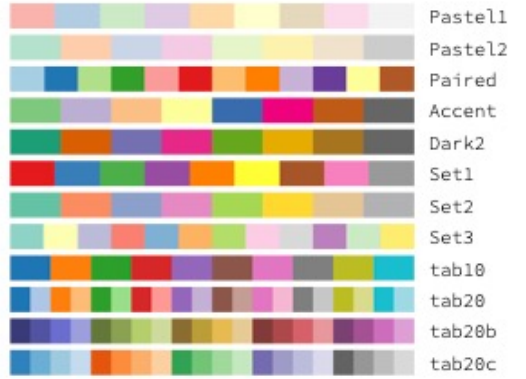
Sequential colormaps



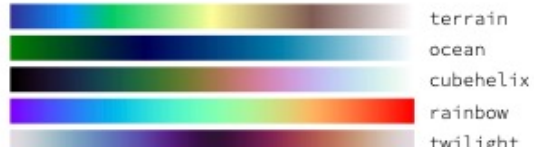
Diverging colormaps



Qualitative colormaps



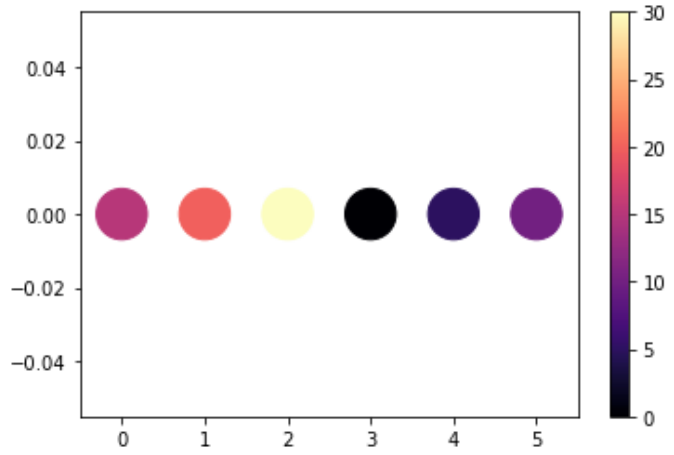
Miscellaneous colormaps

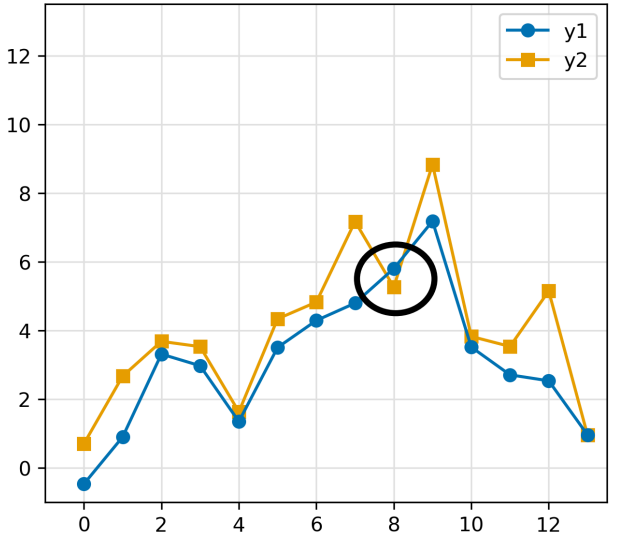


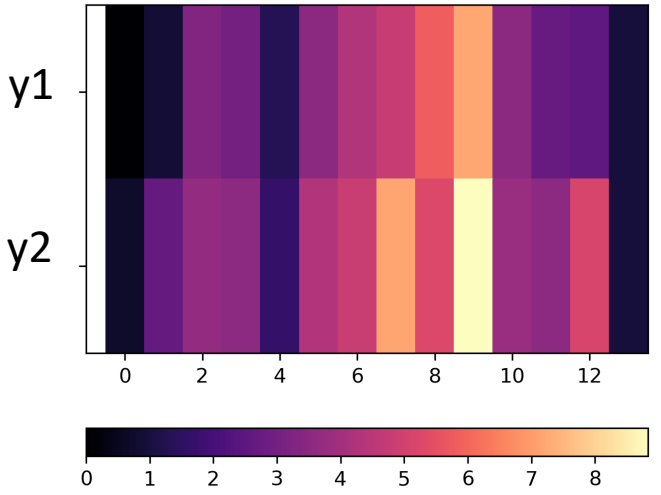
Colormaps

```
fig, ax = plt.subplots()
ax.set_xlim([-.5,5.5])
points = ax.scatter(np.arange(0,6,1),np.zeros(6), c=[15,20,30,0,5,10],
           s=750, cmap=plt.get_cmap('magma'), vmin=0, vmax=30)
fig.colorbar(points, ax=ax)
```

```
<matplotlib.colorbar.Colorbar at 0x7fece0ef2100>
```







- (go to iLearn and download ’05_data.csv’)
- create the shown figure with 2 subplots
- familiarize with the *pcolormesh()* function

Matplotlib cheatsheets and hand-outs:
https://github.com/matplotlib/cheatsheets#cheatsheets

Creating Multiple Subplots

- go to iLearn and download ’05_data.csv’
- create the shown figure with 4 subplots
- figure out how to draw circles with Matplotlib

Matplotlib cheatsheets and hand-outs:
https://github.com/matplotlib/cheatsheets#cheatsheets