Astroinformatics II (Semester 2 2024)

Introduction & Course Logistics
Advanced Data Visualization

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

September 3, 2024

What you will learn in this class

Astronomers spend about 100 % of their time working with a computer.

This course will **prepare you** for:

- How to use Python efficient to do research
- How to write programs for typical research tasks
- How to use other programming languages than Python.

What you will learn in this class

Astronomers spend about 100 % of their time working with a computer.

This course will **prepare you** for:

- How to use Python efficient to do research
- How to write programs for typical research tasks
- How to use other programming languages than Python.

What this course does **not** cover:

- machine learning
- usage of LaTeX

(You might find some of that in other courses offered.)

Course Logistics

course content:

- lecture: Tuesday 8-10am (changes to 2-4pm)
- tutorial: Wednesday 7:30-8:30am (changes to 1-2pm)
- graded project practice
- graded project presentation

Course Logistics

course content:

- lecture: Tuesday 8-10am (changes to 2-4pm)
- tutorial: Wednesday 7:30-8:30am (changes to 1-2pm)
- graded project practice
- graded project presentation

grading:

- 4 graded project practices (60% of total grade)
  content: solving programming tasks by applying what has been learned in class, report in English to produce in one week
  evaluation: capacity to solve a problem and describing the solution
- participation (10 % of total grade)
- project presentation (30 % of total grade)
  content: summarizing what was done in the project practices

Course Logistics

course content:

- lecture: Tuesday 8-10am (changes to 2-4pm)
- tutorial: Wednesday 7:30-8:30am (changes to 1-2pm)
- graded project practice
- graded project presentation

grading:

- 4 graded project practices (60% of total grade)
  content: solving programming tasks by applying what has been learned in class, report in English to produce in one week
  evaluation: capacity to solve a problem and describing the solution
- participation (10 % of total grade)
- project presentation (30 % of total grade)
  content: summarizing what was done in the project practices

contact and course material:

- e-mail: `nina.hernitschek@uantof.cl`
- github: `https://github.com/ninahernitschek/astroinformatica_II_2024_2`

Course Logistics

September 3 Lecture 1: Intro, Advanced Visualization, **September 4** Tutorial 1

September 10 Lecture 2: Writing Efficient Code in Python **September 11** Tutorial 2

September 24 Lecture 3: GPU & Parallelization, **September 25** Tutorial 3

October 1 Graded Project Practice 1

October 8 Lecture 4: Introduction to C/C++ (I), **October 9** Tutorial 4

October 15 Lecture 5: Introduction to C/C++ (II), **October 16** Tutorial 5

October 21 Graded Project Practice 2

October 29 Lecture 6: Object-Oriented Programming, **October 30** Tutorial 6

November 5 Graded Project Practice 3

November 19 Lecture 7: Integrating C++ and Python, **November 20** Tutorial 7

November 26 Graded Project Practice 4

December 3 Project Presentation

Rules for Coding, Presentations, Report

graded project practices:

How the project report should look like:

- code
- a description of your solution, written with LaTeX in a PDF file, including plots ($\sim$ 1-2 pages)

Rules for Coding, Presentations, Report

coding:

- use meaningful variable names
- use comments and documentation
- If you have a question when something doesn't work, summarize what you tried - often this will even lead to the solution.

Rules for Coding, Presentations, Report

coding:

- use meaningful variable names
- use comments and documentation
- If you have a question when something doesn't work, summarize what you tried - often this will even lead to the solution.

project practice reports and project presentation:

- use LaTeX
- figures: all own figures should be in vectorized pdf format
- for all data: data description (incl. citation)
- own work: properly cite what is not your own work; discuss how the previous work is similar to or different from your own work
- implementation: medium-level implementation description with libraries/ software frameworks (incl. citation)
- discussion: reflect your approach (strengths, weaknesses, limitations), lessions learned
- bibliography

Textbooks

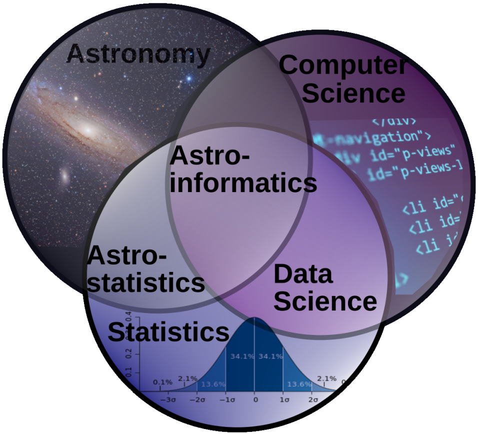
I can recommmend the following textbooks and online resources:

Python cookbook; Beazley, D. & Jones, B.; O'Reilly Media.

A Beginner's Guide to Working with Astronomical Data. M. Pössel, The Open Journal of Astrophysics, vol. 3, issue 1, id. 2, 2020.

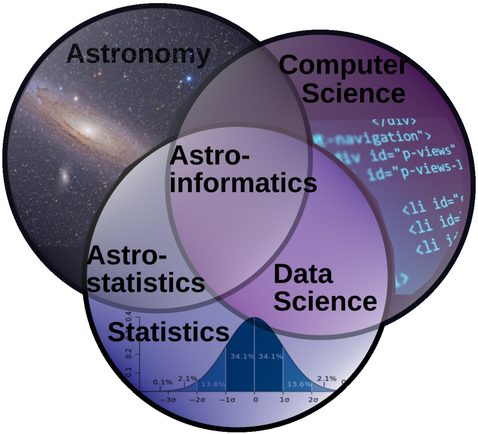
What is Astroinformatics?

Astroinformatics is an **interdisciplinary** field:



What is Astroinformatics?

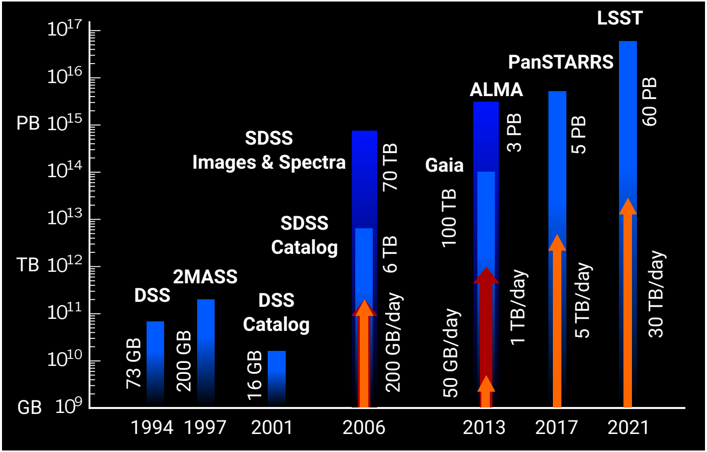
Astroinformatics is an **interdisciplinary** field:



Astroinformatics is primarily focused on developing the tools, methods, and applications for research in data-heavy astronomy.

Why we need Astroinformatics?

Modern All-Sky Surveys come with an increasing data volume:



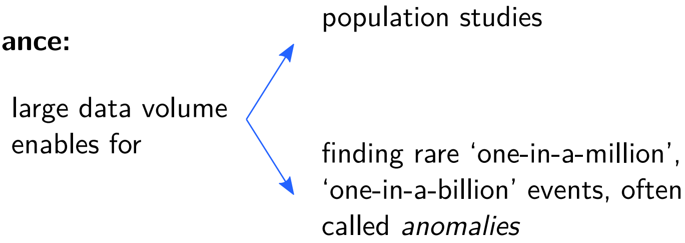
Why we need Astroinformatics?

upcoming large all-sky surveys like (but not only) LSST

Challenge:

- enormous data volume, > 60 PB total, 15 - 30 TB/ night
- follow-up opportunities should be identified immediately

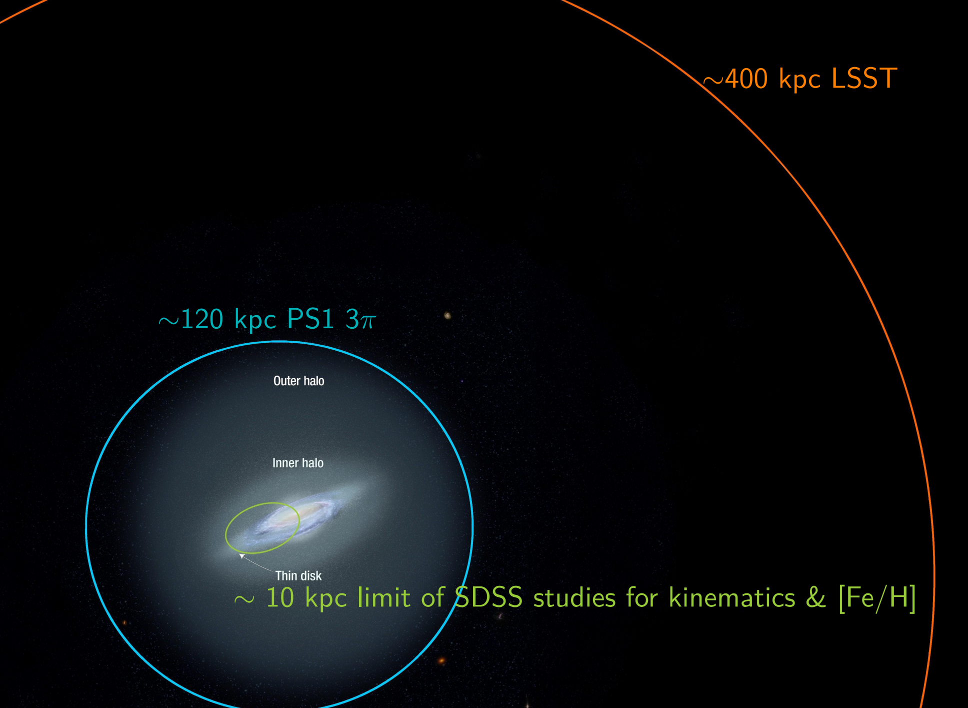
Chance:



large data volume enables for

population studies

finding rare ‘one-in-a-million’, ‘one-in-a-billion’ events, often called *anomalies*



∼400 kpc LSST

∼120 kpc PS1 3π

∼ 10 kpc limit of SDSS studies for kinematics & [Fe/H]

image based on NASA, ESA, and A. Feild (STScI)

Why we need Astroinformatics?

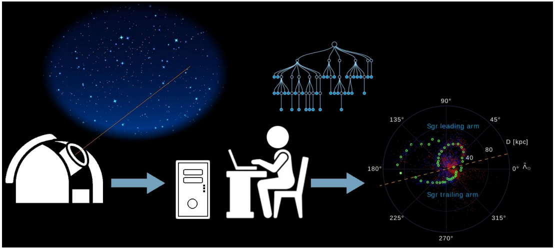
Astronomers need to solve equations, sometimes symbolic, sometimes numerically. Those are for example:

- Equations of movement of celestial bodies. (Compare to: The discovery of Neptune was made by solving equations by hand.)
- Computation of the stellar interior.
- Hydrodynamical simulations of which some now contain billions of particles.
- Machine learning for data from large all-sky surveys, for such as classification.
- Survey strategy planning, e.g. LSST survey strategy uses machine learning.
- Astroengineering problems, e.g. adaptive optics.

Example Use Cases

how I use computers/ computer programs in research

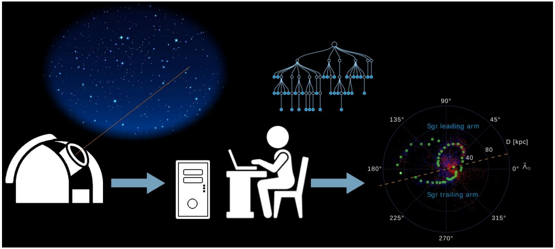
- planning observations



Example Use Cases

how I use computers/ computer programs in research

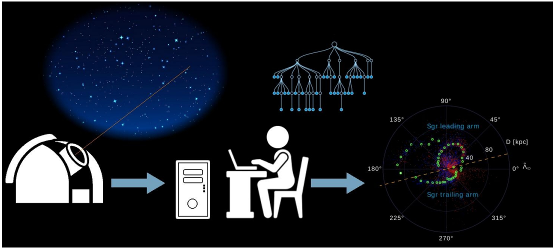
- planning observations
- downloading, plotting, preprocessing, merging data



Example Use Cases

how I use computers/ computer programs in research

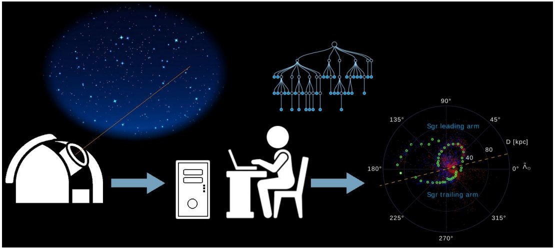
- planning observations
- downloading, plotting, preprocessing, merging data
- classifying data (machine learning)



Example Use Cases

how I use computers/ computer programs in research

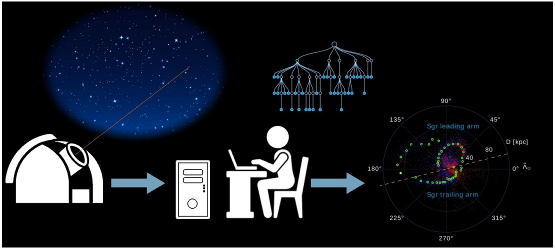
- planning observations
- downloading, plotting, preprocessing, merging data
- classifying data (machine learning)
- creating complex visualizations



Example Use Cases

how I use computers/ computer programs in research

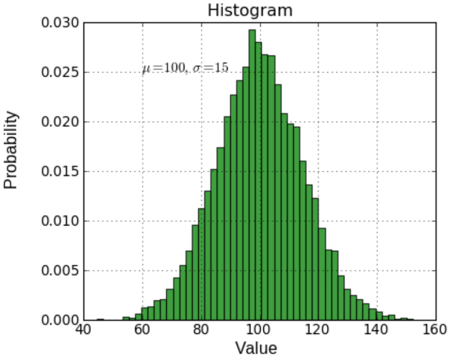
- planning observations
- downloading, plotting, preprocessing, merging data
- classifying data (machine learning)
- creating complex visualizations
- developing a LSST broker software



Advanced Data Visualization with Python

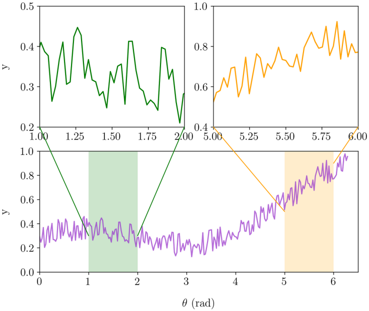
So far (Astroinformatica I):

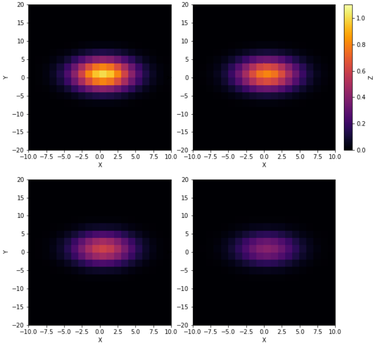
standard plots that can be made with matplotlib



Advanced Data Visualization with Python

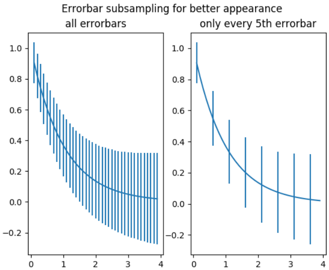
Mathplotlib allows for many ways to control the appearance of such plots, like in the following examples:

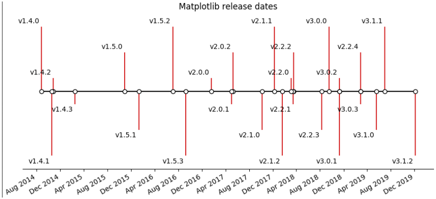




Advanced Data Visualization with Python

Based on this, even more complex plots can be created:

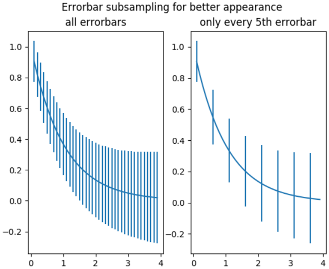


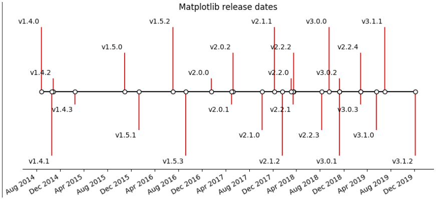


Source: matplotlib Gallery
`https://matplotlib.org/3.1.1/gallery/index.html`

Advanced Data Visualization with Python

Based on this, even more complex plots can be created:





Source: matplotlib Gallery
`https://matplotlib.org/3.1.1/gallery/index.html`

⇨ We will see an overview here and we will practice making such plots in the tutorial session this week.

Subplots

Subplots allow for creating multiple plots within a single figure.

Purpose:
This can be useful for comparing different datasets or different values (columns) within a dataset.

How to do it:
Subplots can be created in Matplotlib by using the `subplot()` function, which takes three arguments: the number of rows, the number of columns, and the index of the subplot to be created.

example: to create a 2×2 grid of subplots, we can use the following code:

```
fig, ax = plt.subplots(2, 2)
ax[0, 0].plot(x, y1)
ax[0, 1].plot(x, y2)
ax[1, 0].plot(x, y3)
ax[1, 1].plot(x, y4)
```

Subplots

In Matplotlib, **axes** should not be confused with coordinate axis:
Axes are the individual plots within a figure. By default, Matplotlib creates a single axes for each figure. However, you can create multiple axes within a single figure. This can be useful for plotting multiple datasets on one graph or creating a **side-by-side comparison of different datasets**.

How to do it:
Axes can be created using the `axes()` function, which takes four arguments: the x and y position of the left edge of the axes, and the width and height of the axes.

For example, to create two axes side-by-side, we can use the following code:

```
fig, ax = plt.subplots(2, 2)
ax1 = plt.axes([0.1, 0.1, 0.4, 0.4])
ax2 = plt.axes([0.5, 0.1, 0.4, 0.4])
ax1.plot(x, y1)
ax2.plot(x, y2)
```

Fine-tuning Figure Size and Layout

For making visually appealing and easily legible plots by on optimizing figure size and layout. Take the following factors into account:

Figure Size: The figure size should ensure that your plot is clearly visible without overwhelming the overall layout. Both oversized and too small figures can make it challenging interpreting the data.

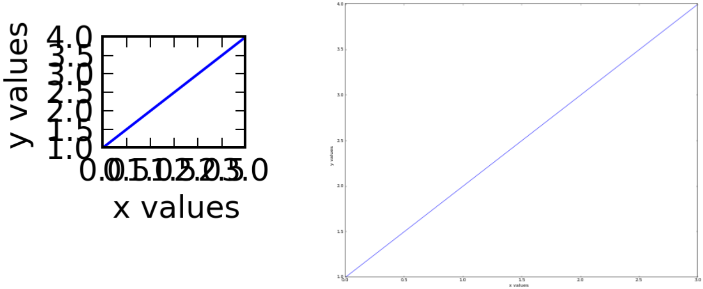


Fig: Two plots where the figure size is not optimal.

Figure Size

The following plot has the right size for half-page width plots:

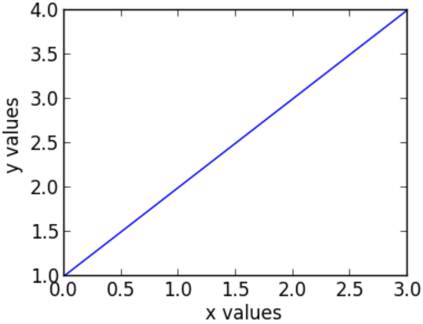
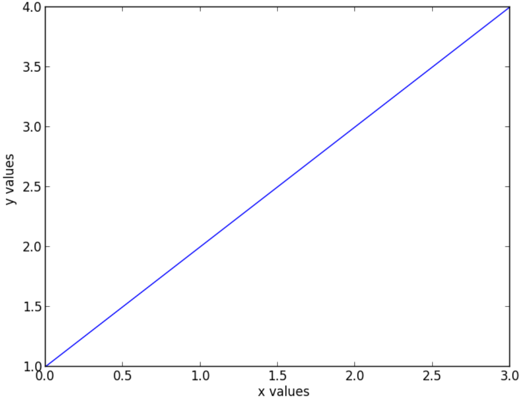


Figure Size

… and for full-page width plots:

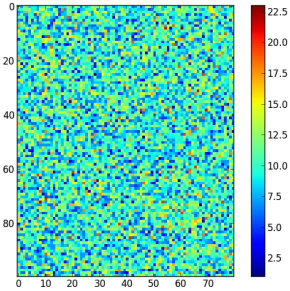


Colorbars

Aspect Ratio: Tailor the aspect ratio of your figure to the nature of your data. For instance, when dealing with time series data, a wider aspect ratio can effectively show trends over time.

Spaciousness and Margins: Enhance readability by adjusting space between different plot elements such as titles, color bars, axis labels.

Adding a **colorbar** in Matplotlib automatically takes space from the axes to which it is attached:



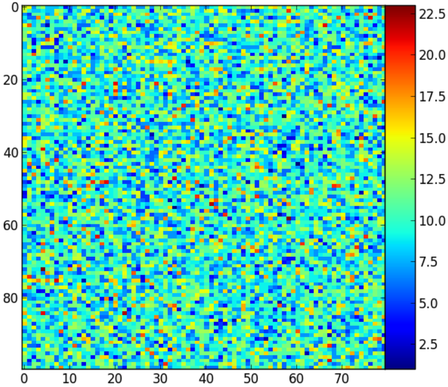
Colorbars

If you want to **customize exactly** where the colorbar appears, you can define a set of axes, and pass it to colorbar using the cax argument and setting aspect=’auto’:

```
fig = plt.figure()
ax = fig.add_axes([0.1,0.1,0.6,0.8])
image = np.random.poisson(10., (100, 80))
i = ax.imshow(image, aspect='auto', interpolation='nearest')

colorbar_ax = fig.add_axes([0.7, 0.1, 0.05, 0.8])
fig.colorbar(i, cax=colorbar_ax)
```

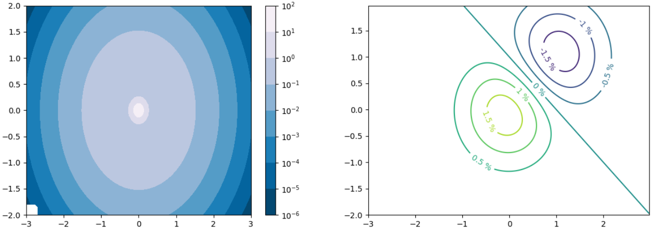
Colorbars

This is the resulting plot:



Contour Plots

Matplotlib allows for creating contour plots to visualize 3D data. There are various styles available, e.g.:

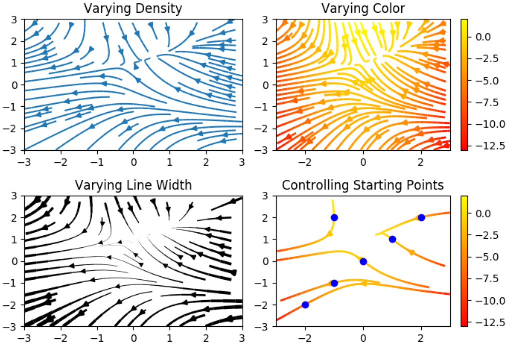


Source: `https://matplotlib.org/3.1.1/gallery/index.html`

Stream Plots

Motivation
Course Logistics
What is Astroinformatics?
Advanced Data Visualization
Workflow
Which Plot to Use?
Summary & Outlook

The Matplotlib `streamplot()` function is designed to plot vector fields. In addition to simply plotting the streamlines, it allows to map the colors and/or line widths of streamlines to a separate parameter, such as the speed or local intensity of the vector field.



Varying Density

Varying Color

Varying Line Width

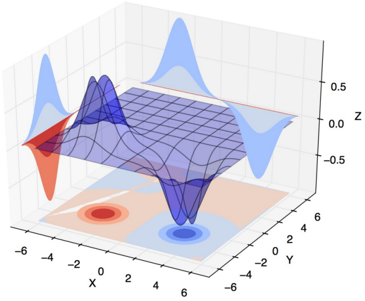
Controlling Starting Points

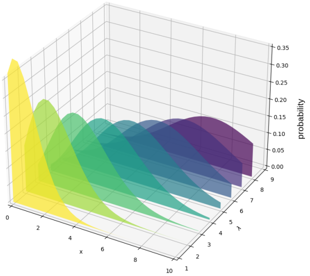
Source: `https://matplotlib.org/3.1.1/gallery/images_contours_and_fields/plot_streamplot.html`

3D Plots

The mplot3d Toolkit in Matplotlib allows for creating a variety of 3D plots, including such that show projections onto lower dimensions (marginal distributions).

Because of the broad topic, it is strongly recommended to check out the documentation.





Source:
`https://matplotlib.org/3.1.1/tutorials/toolkits/mplot3d.html`

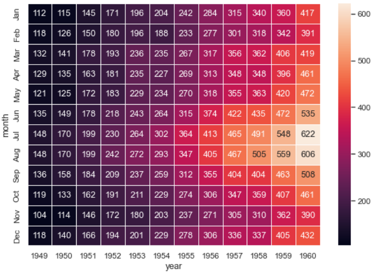
seaborn Python package

Seaborn is a Python data visualization library based on matplotlib.

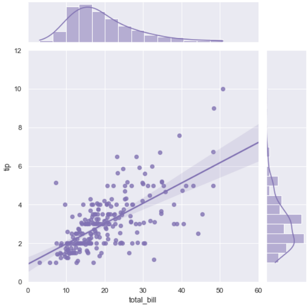
Seaborn allows to create visually appealing plots with minimal code, while Matplotlib offers detailed customization for every aspect of a plot.

Documentation and the source code of example plots such as the plots below are available at `https://seaborn.pydata.org/`.

Annotated heatmap



Linear regression with marginal distributions



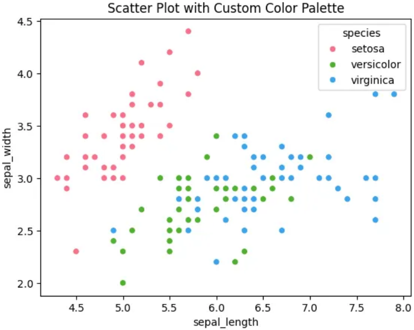
seaborn Python package

example: Using seaborn to plot with a custom color palette.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Setting a color palette
sns.set_palette("husl")

# Sample data
data = sns.load_dataset("iris")

# Scatter plot with custom color palette
sns.scatterplot(data=data, x="sepal_length", y="sepal_width",
               hue="species", palette="husl")
plt.title("Scatter Plot with Custom Color Palette")
plt.show()
```

seaborn Python package

This is the resulting plot:

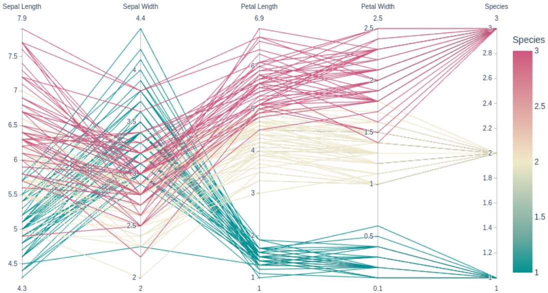


plotly Python package

More advanced plots including *interactive plots* can be made using Plotly, an open-source library building on the JavaScript library plotly.js.

example: Representations of multivariate data.

In this **parallel coordinates plot** of the iris data set, each row of the DataFrame is represented by a polyline mark which traverses a set of parallel axes, one for each of the dimensions.

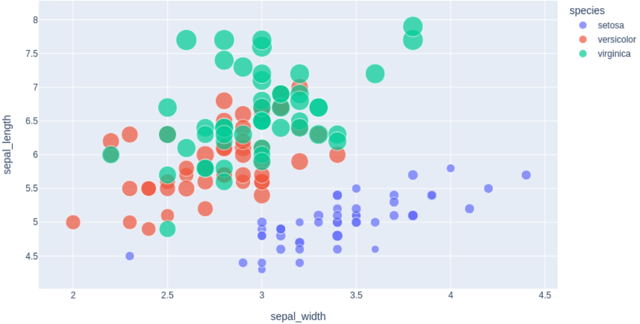


Source: `https://plotly.com/python/`

plotly Python package

Imagine assigning different sizes to points based on a third variable, and coloring them based on a fourth variable. Suddenly, multiple dimensions converge in a single plot, unveiling hidden patterns.

Scatter plots with variable-sized circular markers are often known as **bubble charts**.

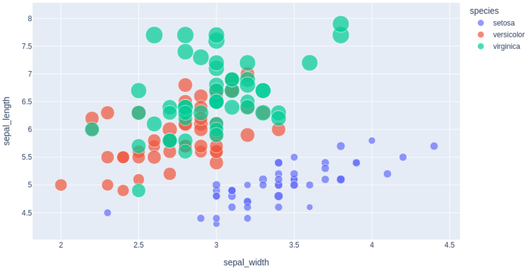


plotly Python package

They can easily be produced in plotly:

```
import plotly.express as px

df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length",
                 color="species", size='petal_length',
                 hover_data=['petal_width'])
```

Caution: The plot will show up in the web browser.



plotly Python package

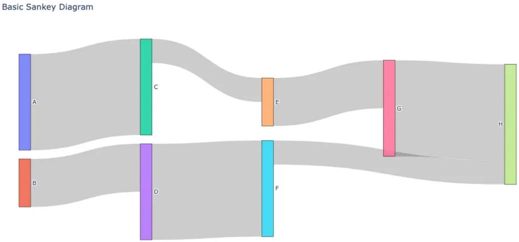
Plotly makes it possible to create **3D and animated visualizations**. They can be used to reflect changes over time or in response to various variables. They are thus a valuable tool for such as presentations and websites.

```
import plotly.graph_objs as go
import numpy as np

# Sample data
t = np.linspace(0, 10, 50)
x = np.sin(t)
y = np.cos(t)
z = t

# Creating the animated scatter plot
fig = go.Figure(data=[go.Scatter3d(x=x, y=y, z=z, mode='markers')])

fig.update_layout(scene=dict(
    xaxis_title='X',
    yaxis_title='Y',
    zaxis_title='Z'),
    title='3D Scatter Plot')

fig.show()
```

plotly Python package

Plotly makes it possible to create **Sankey Diagrams**.
Sankey diagrams capture the movement and connections within datasets. The width of the flows corresponds to the quantity being represented, making it easy to identify the major flows and their relative magnitudes. These diagrams enable analysts to grasp intricate relationships and patterns, providing a comprehensive visual understanding of how data elements are interconnected.

The functionality can be accessed with `plotly.graph_objects.Sankey()`.



Workflow

general advice: **Separating computations and plotting**

If you are doing calculations prior to plotting, and these take a while to get carried out, it is a good idea to separate the computational part of scripts from the plotting part (i.e. have a dedicated plotting script). Save the information from the computation routine, and then read this in to a plotting program.

The **advantage** of doing this is that it is easier to tweak the plotting script without re-running the computation every time. Also other mistakes (e.g.: plotting routine failing) can be prevented.

Workflow

general advice: **Separating computations and plotting**

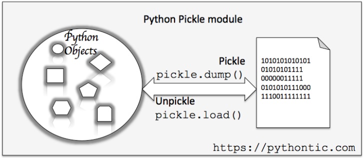
If you are doing calculations prior to plotting, and these take a while to get carried out, it is a good idea to separate the computational part of scripts from the plotting part (i.e. have a dedicated plotting script). Save the information from the computation routine, and then read this in to a plotting program.

The **advantage** of doing this is that it is easier to tweak the plotting script without re-running the computation every time. Also other mistakes (e.g.: plotting routine failing) can be prevented.

What to do with the data?
You can save them in a text file, or you can **serialize** them.

Workflow

The Python pickle module implements binary protocols for serializing and de-serializing a Python object structure.



The process of dumping objects from memory to binary file with pickle:

```
import pickle

pickle.dump(object, object.pkl, other_params)
```

The loading process from binary pickle file to memory is just as simple:

```
import pickle

object = pickle.load(object.pkl)
```

Which Plot to Use?

Which plot type is the right for your data?

It depends on your data and what you want to show.

A **good overview** guiding you through the decision process can be found here:

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

Which Plot to Use?

The Python Graph Gallery is a collection of hundreds of charts made with Python available on `https://python-graph-gallery.com/`

Graphs are given in about 40 sections following the data-to-viz classification. There are also sections dedicated to more general topics like matplotlib or seaborn.

Each **example** is accompanied by its corresponding reproducible code along with comprehensive explanations.

Tips and Tricks

- Remember to label your axes, so viewers know what the data represents.
- Add a legend to your graph, so viewers know which lines or points represent which datasets.
- Consider adding a title to your graph, so viewers know what the graph is about.
- Use color to draw attention to important points, or to separate different datasets.
- Check your graph for errors, such as incorrect axes labels or incorrect data points.
- After computing something, first save the results before plotting them.
- Save your graph as an image file, so you can share it with others.
- Use different plotting techniques, such as subplots, axes, and 3D graphs, to create the best possible visualization.

Summary of Resources

https://python4astronomers.github.io/plotting/advanced.html

https://python4astronomers.github.io/

https://matplotlib.org/stable/gallery/color/named_colors.html

https://python-graph-gallery.com/

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

https://matplotlib.org/3.1.1/gallery/index.html

An Outlook: Efficient Programming in Python

We have seen a variety of advanced plots, including resources on where to get example code for an easy start.

In the next lecture we will see how to **write more efficient code** in Python.