Python (Semester 2 2024)

Data Visualization

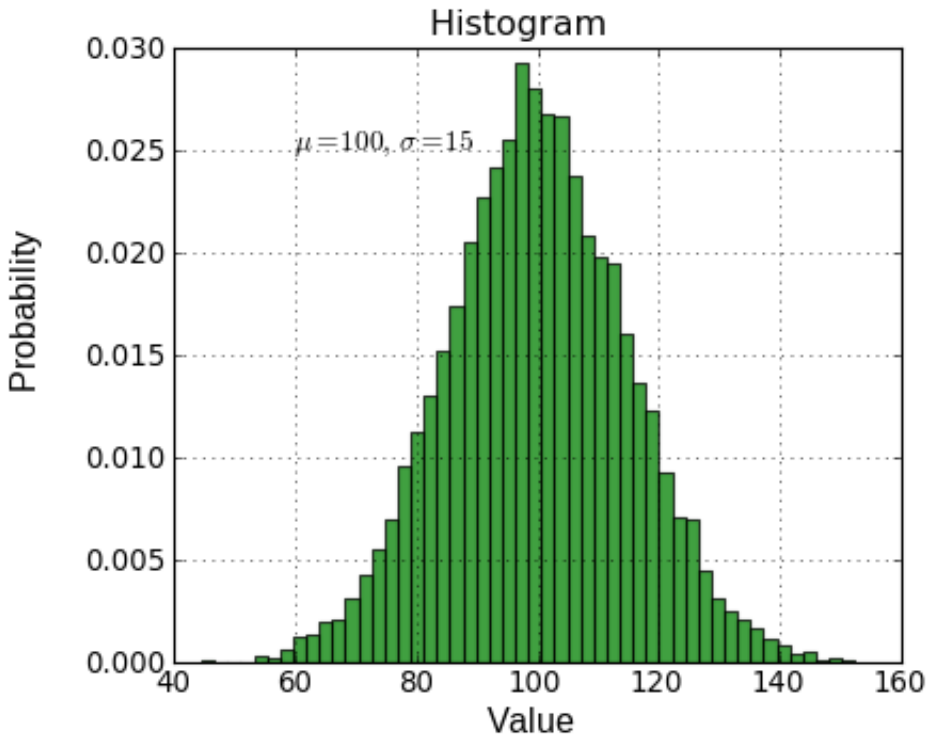
Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

September 23, 2024

Advanced Data Visualization with Python

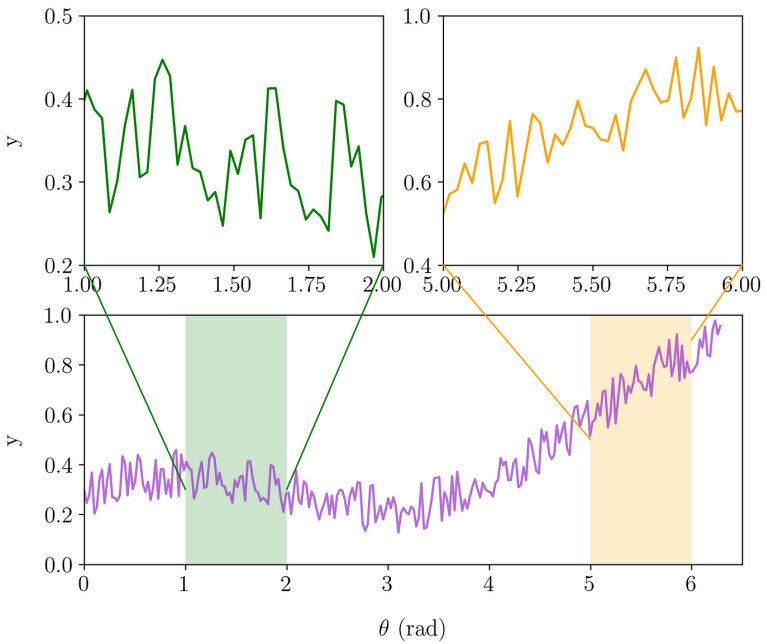
So far (Astroinformatica I):

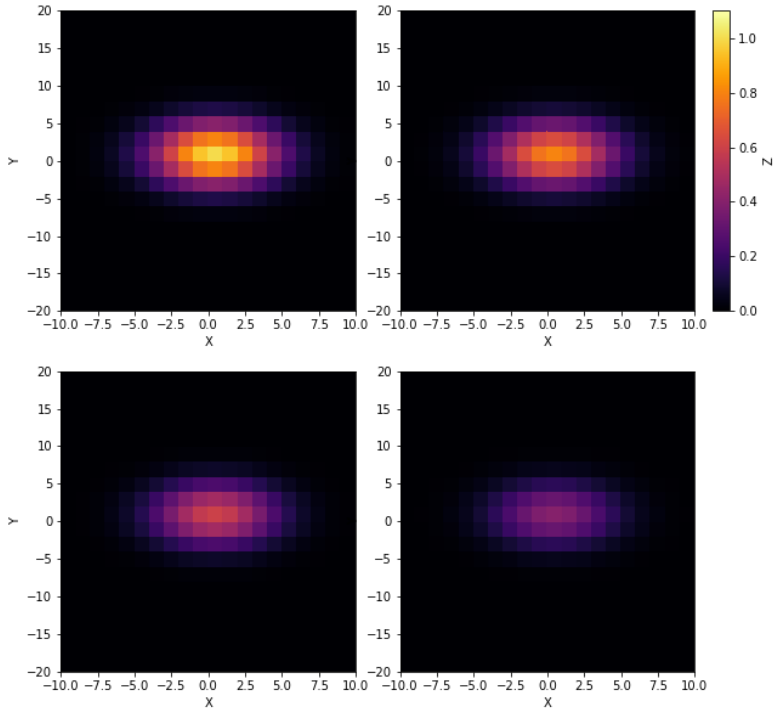
standard plots that can be made with matplotlib



Advanced Data Visualization with Python

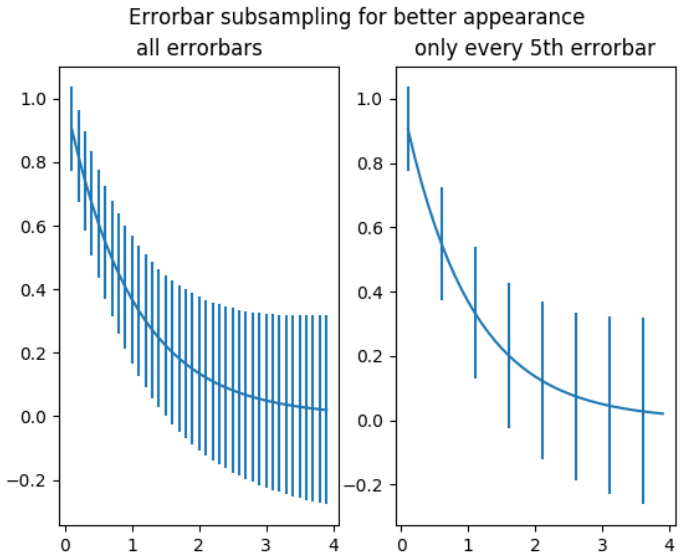
Mathplotlib allows for many ways to control the appearance of such plots, like in the following examples:

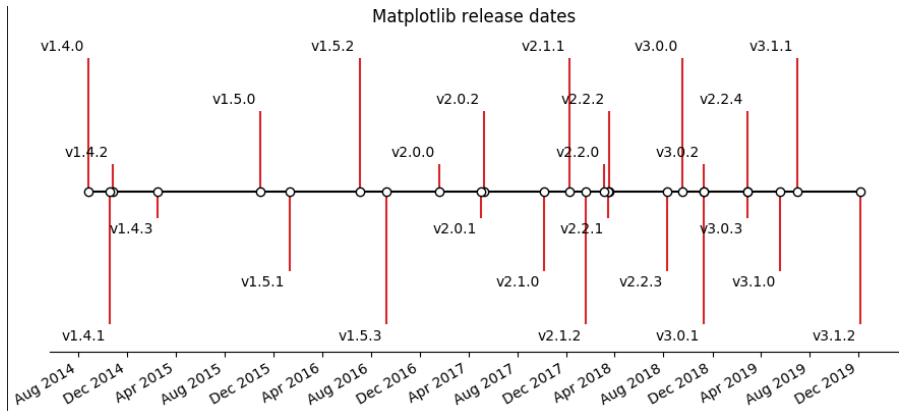




Advanced Data Visualization with Python

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

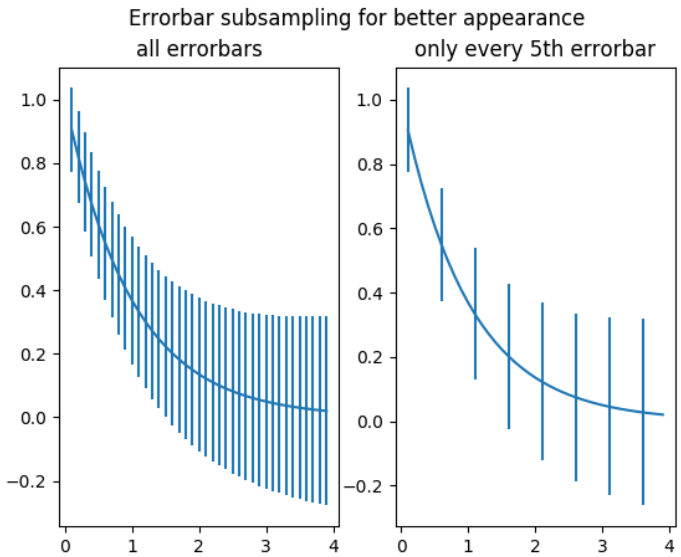


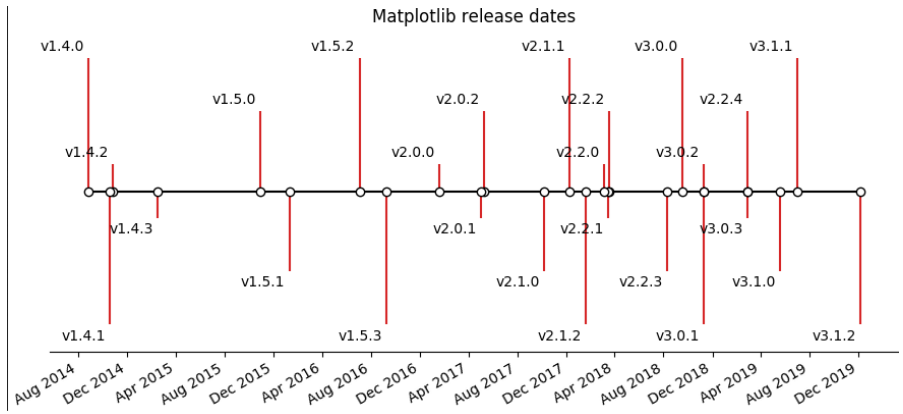


Source: matplotlib Gallery
`https://matplotlib.org/3.1.1/gallery/index.html`

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.

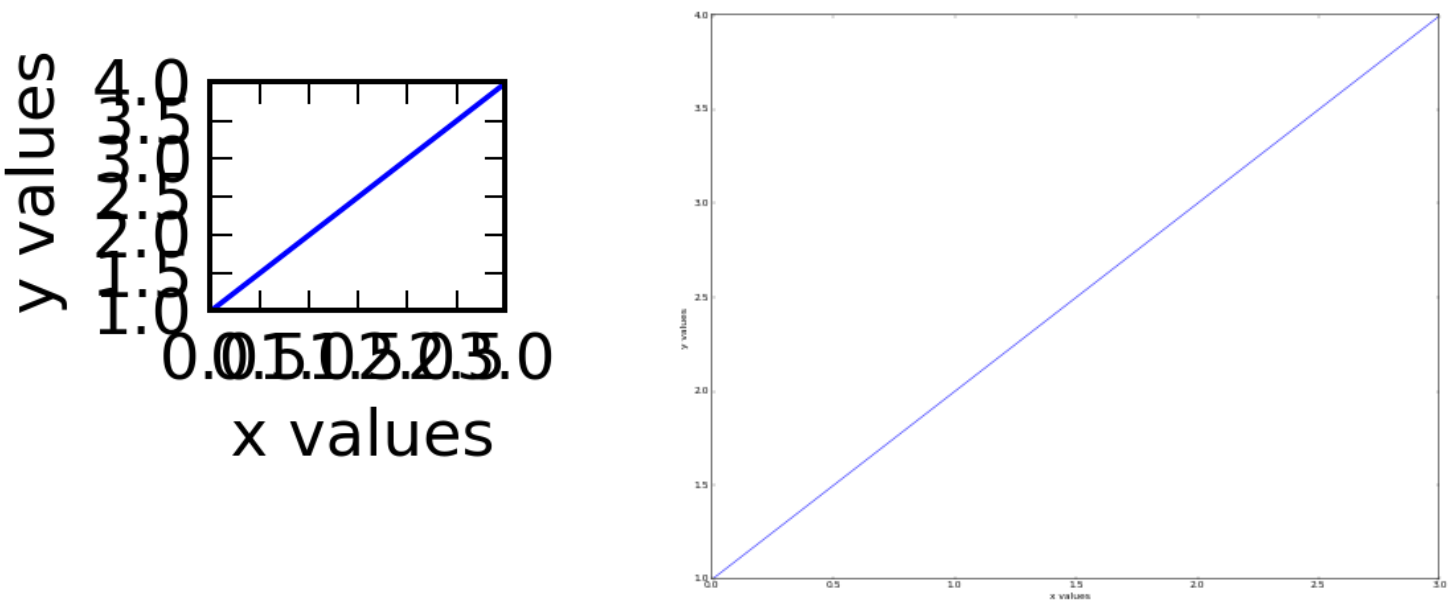


Fig: Two plots where the figure size is not optimal.

Figure Size

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

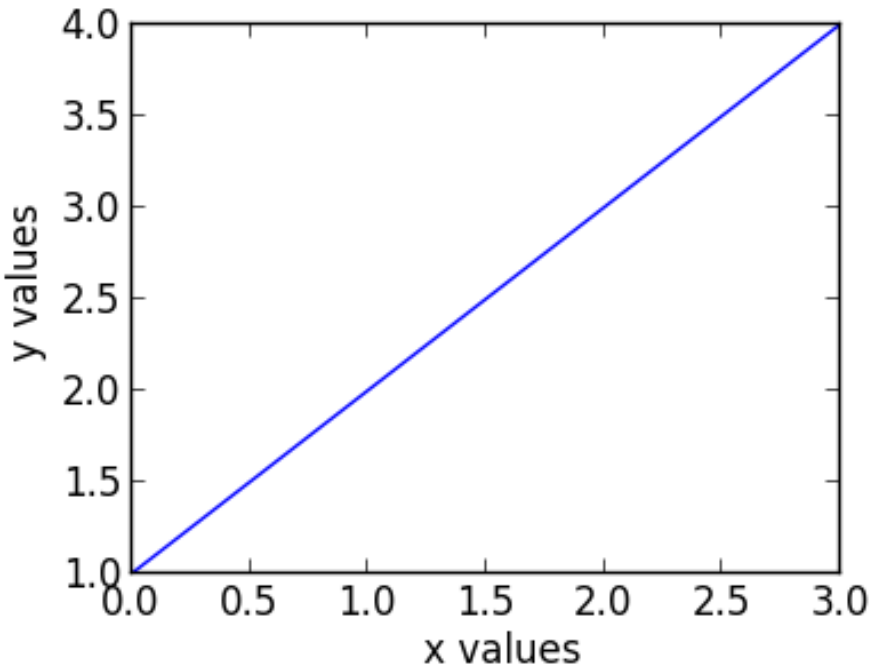
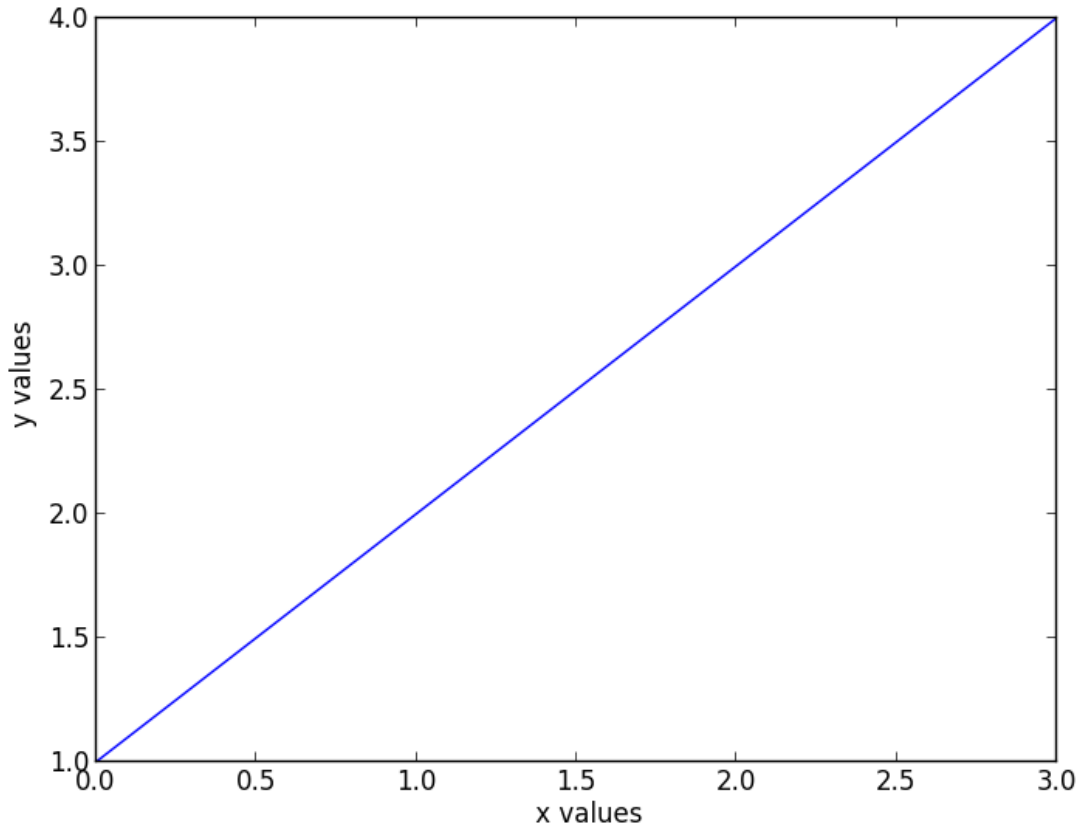


Figure Size

... and for full-page width plots:



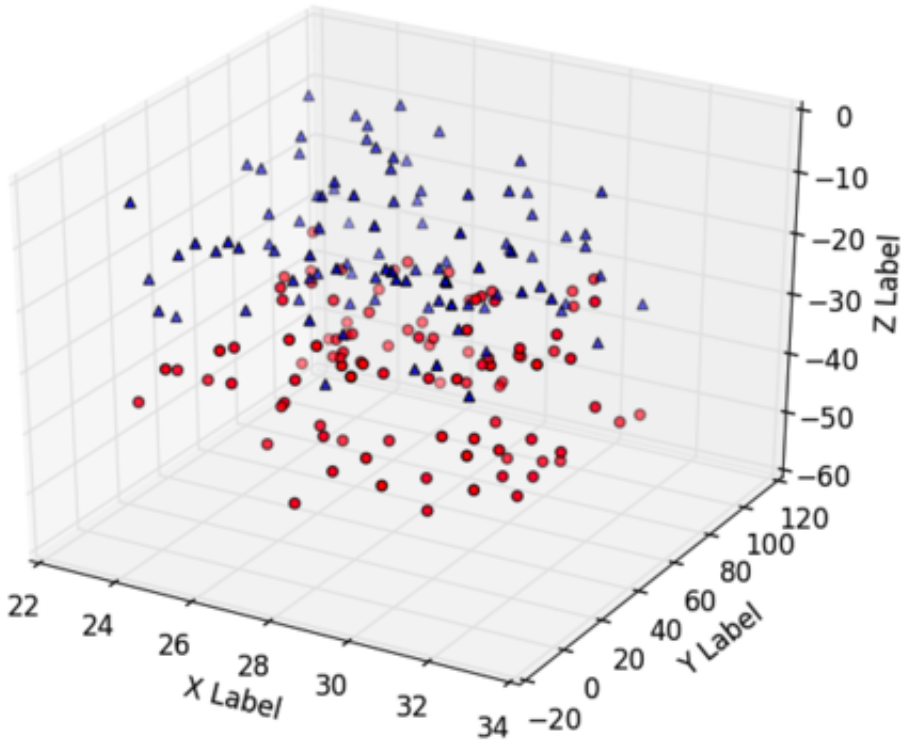
Visualizing 3D Data

Matplotlib supports different ways for **plotting 3D data**.
Here is an example using the `mpl_toolkits.mplot3d` module:

```
# Example: plotting 3D data

from mpl_toolkits.mplot3d import Axes3D

def randrange(n, vmin, vmax):
    return (vmax-vmin)*np.random.rand(n) + vmin

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
for c, m, zl, zh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zl, zh)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
```

Visualizing 3D Data

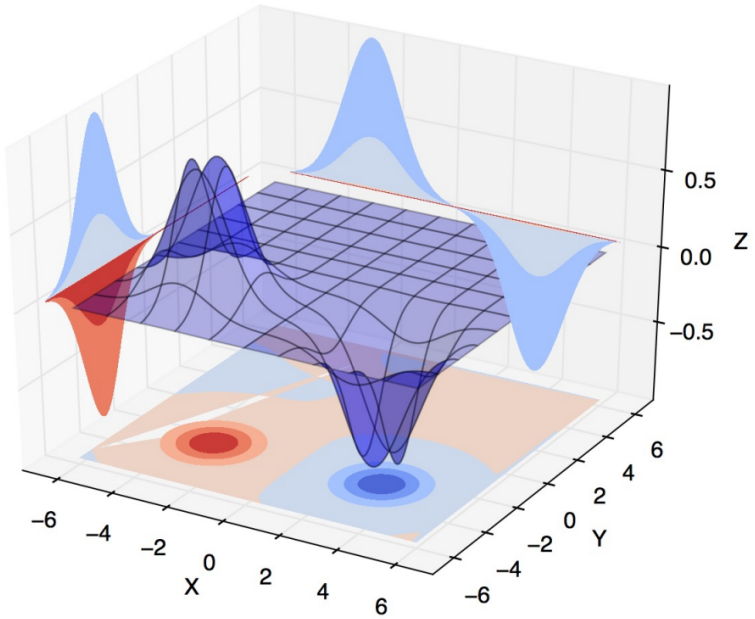
The resulting plot is the following:

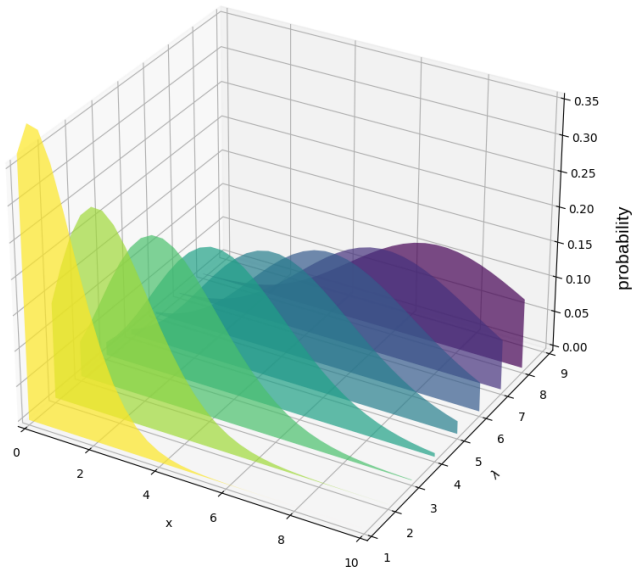


3D Plots

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.



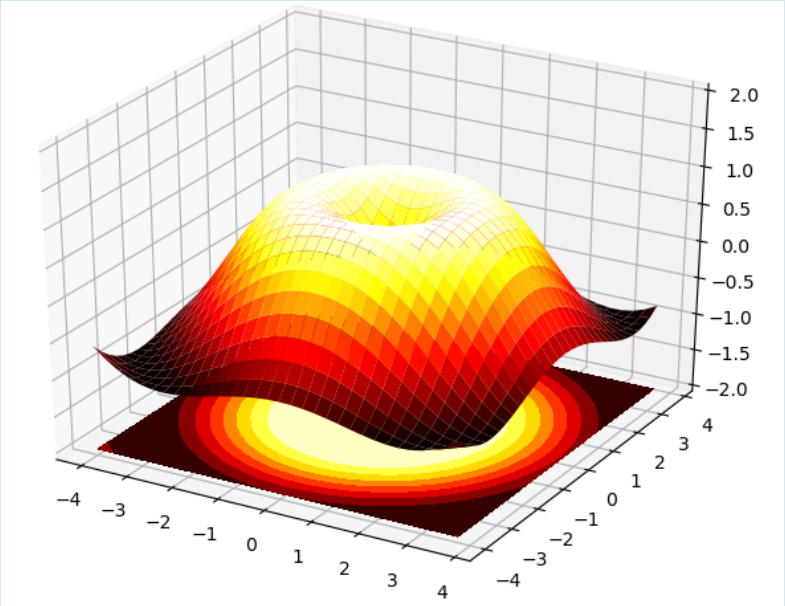


Source:
`https://matplotlib.org/3.1.1/tutorials/toolkits/mplot3d.html`

3D Surface Plots

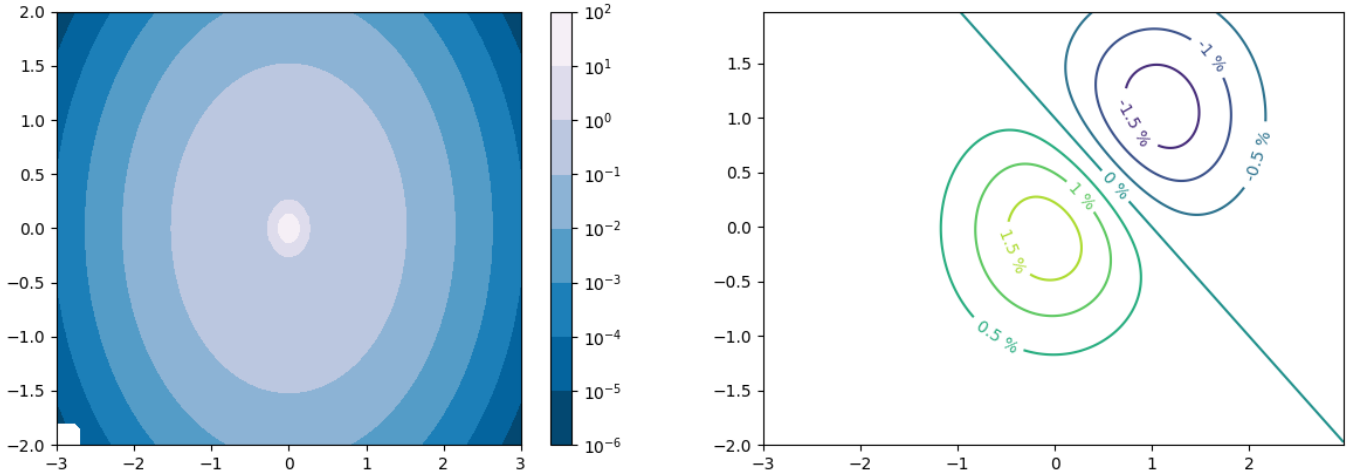
In addition, Matplotlib allows for generating **3D surface plots** of mathematical functions. Here is an example, along with the resulting plot:

```
# example: 3D surface plot

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X ** 2 + Y ** 2)
Z = np.sin(R)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)
ax.set_zlim(-2, 2)

plt.show()
```



Contour Plots

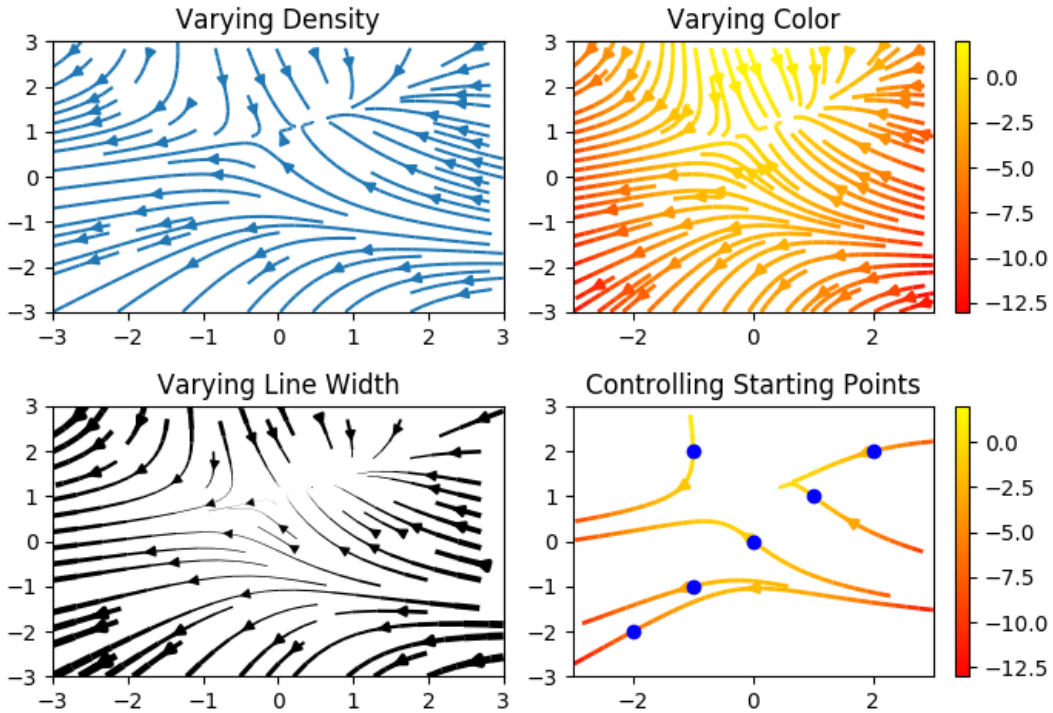
Matplotlib allows for creating contour plots to visualize 3D data. There are various styles available, e.g.:



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

Stream Plots

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.



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

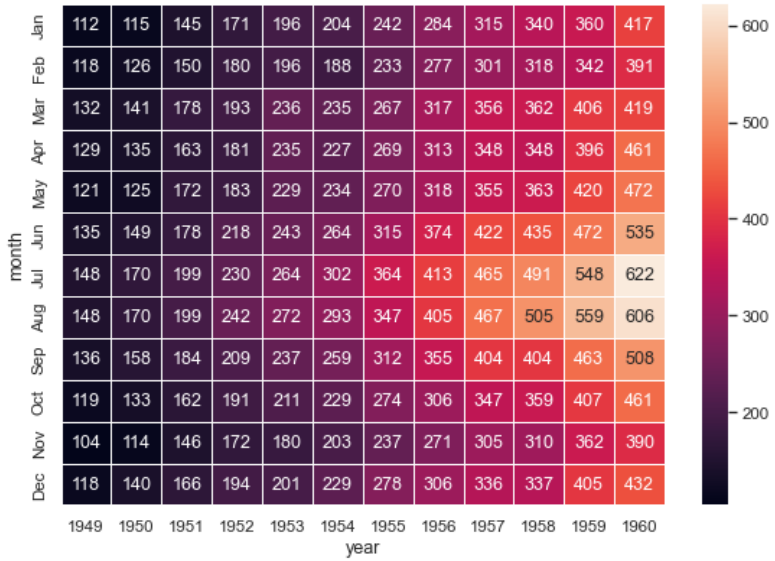
seaborn Python package

Seaborn is a Python data visualization library based on matplotlib.

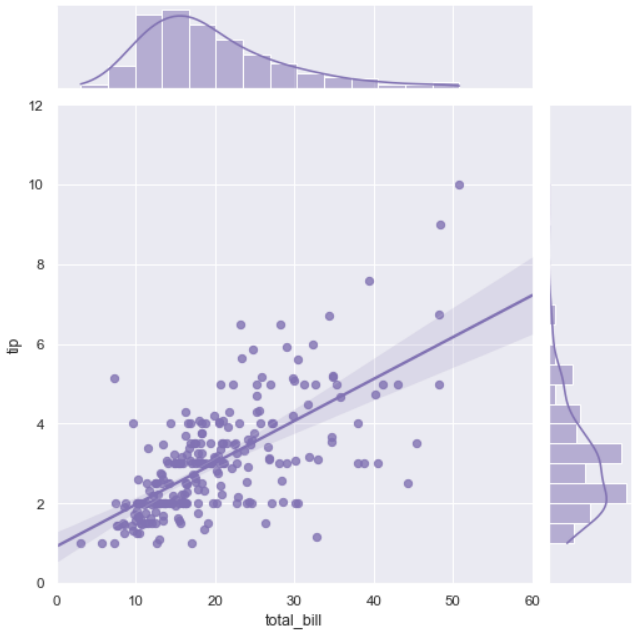
Seaborn allows to create visually appealing plots with minimal code, while Matplotlib offers detailed customization for every aspect of a plot.

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



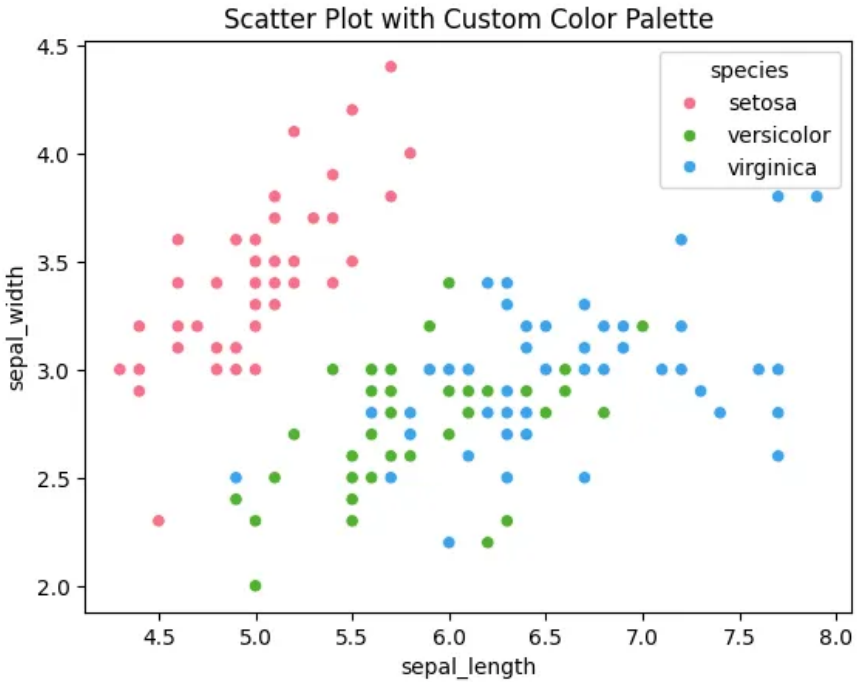
seaborn Python package

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

```
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:

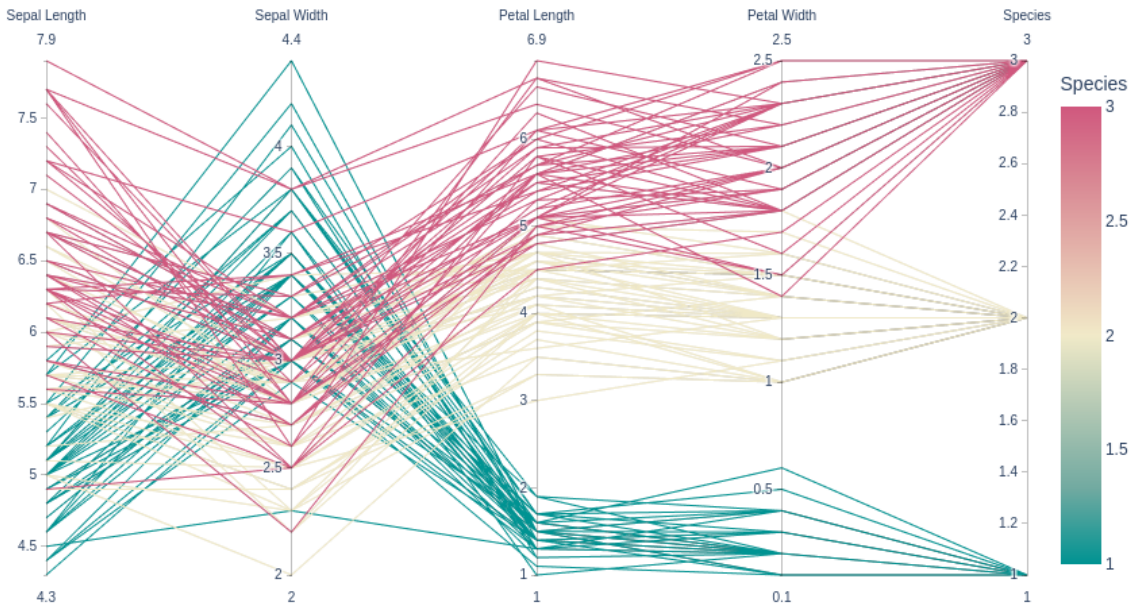


plotly Python package

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

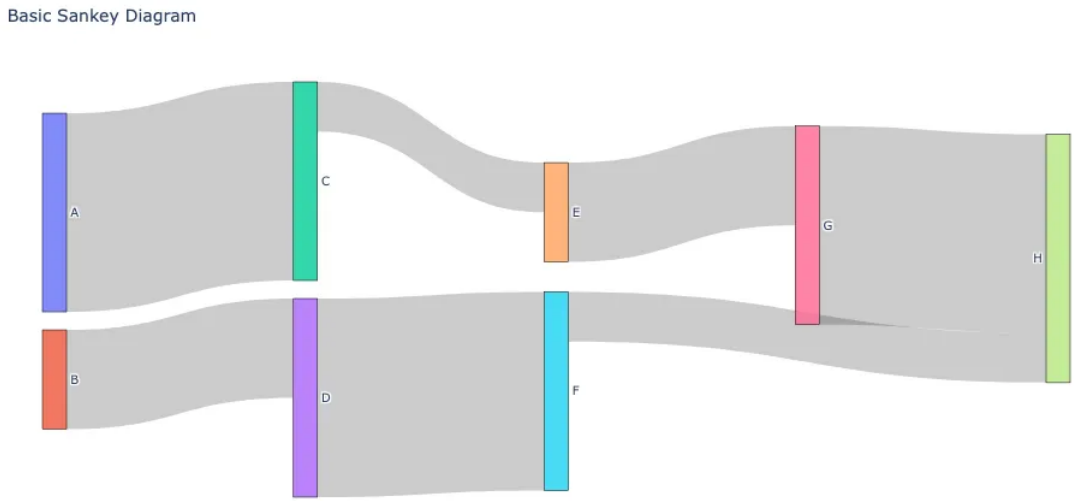
Plotly makes it possible to create **3D and animated visualizations**. They can be used to reflect changes over time or in response to various variables. They are thus a valuable tool for such as presentations and websites.

```
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.

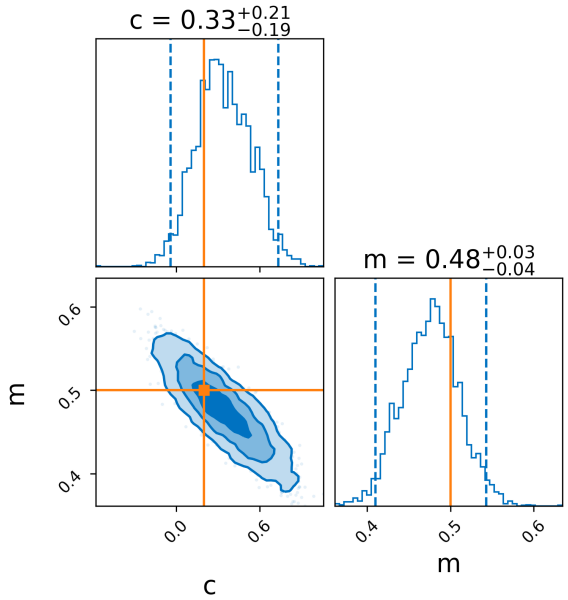
The functionality can be accessed with `plotly.graph_objects.Sankey()`.

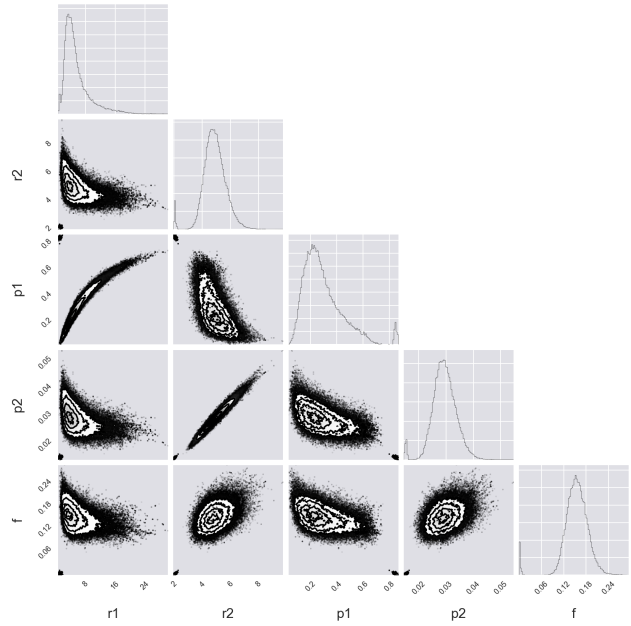


Corner Plots

Finally, beyond plots provided within `matplotlib` and related packages, it is important to highlight that there are specialized plotting libraries for specific purposes.

Corner Plots combine scatter plots, heatmaps and histograms in order visualize multidimensional samples. In these visualizations, each one- and two-dimensional projection of the sample is plotted to reveal covariances. Originally designed to display the results of Markov Chain Monte Carlo simulations, corner plots can be used for displaying multidimensional samples in general.





Corner Plots

The most common package for plotting corner plots is `corner.py`, available at

`https://corner.readthedocs.io/en/latest/`

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**

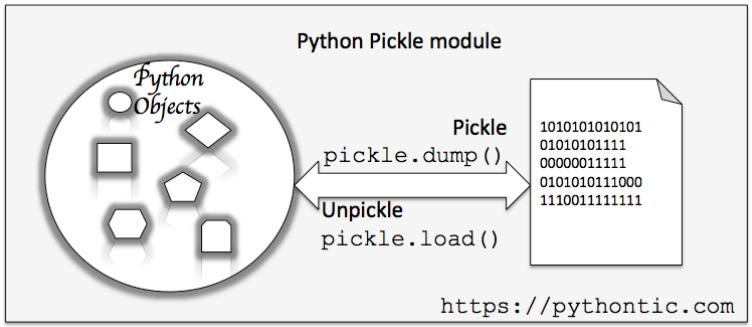
If you are doing calculations prior to plotting, and these take a while to get carried out, it is a good idea to separate the computational part of scripts from the plotting part (i.e. have a dedicated plotting script). Save the information from the computation routine, and then read this in to a plotting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

Here we have seen how to visualize a variety of data in a meaningful way to convey scientific information.

For seeing specific example plots along with the code behind them, I can strongly recommend taking a look at the **Matplotlib gallery** (`https://matplotlib.org/stable/gallery/index.html`).

An Outlook: Efficient Code

Here we have seen how to visualize a variety of data in a meaningful way to convey scientific information.

For seeing specific example plots along with the code behind them, I can strongly recommend taking a look at the **Matplotlib gallery** (`https://matplotlib.org/stable/gallery/index.html`).

In the next lecture, we will see how we can **write more efficient code**.