University of Zurich^{UZH}

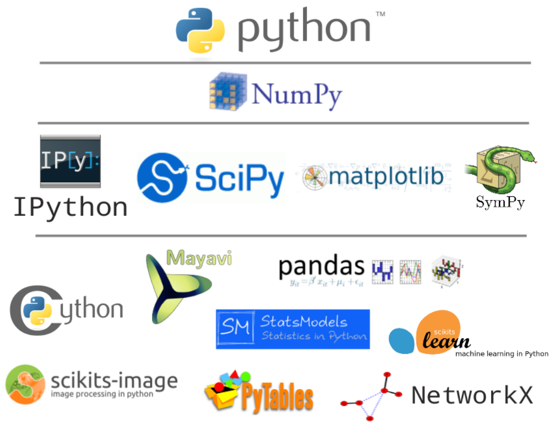
Department of Physics

Data Visualization and more

Scientific Programming with Python

Andreas Weiden

The Ecosystem of Homo Python Scientificus

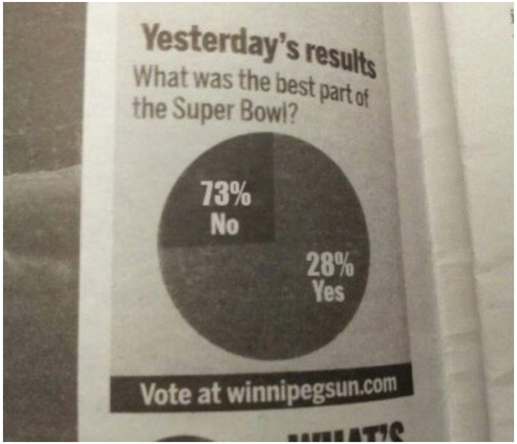

python™
NumPy
IP[y]: IPython
SciPy
matplotlib
SymPy
Mayavi
pandas
Cython
StatsModels Statistics in Python
scikits learn machine learning in Python
scikits-image image processing in python
PyTables
NetworkX

[Ondřej Čertík/LANL]

Table of Contents

Visualisation

Visualization as well as Content Matters



Visualization Options in Python

Matplotlib

- Started as emulation for MATLAB
- Basic plotting also in more than one dimension

Seaborn

- Collection of more complex plots
- Based on Matplotlib

bokeh

- Web publishable graphics
- Large variety of usable interactions

Folium

- Python interface to leaflet (maps)
- Plotting of geo data

Color

Color is a double-edged sword:

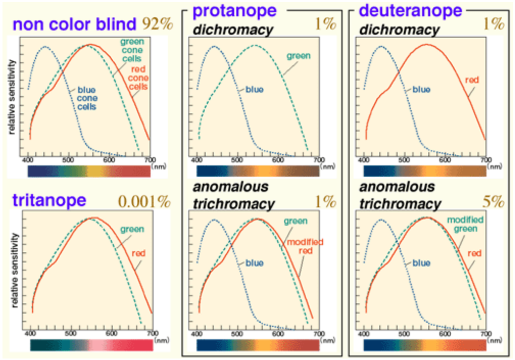
- Color can convey a lot of information
- But there are many forms of Color-blindness
- Many people will print your paper in black & white (for many reasons)

Two (non-exclusive) ways to deal with this:

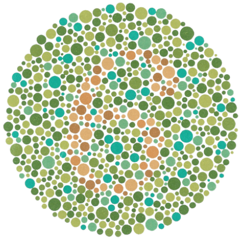
- Use Colors that are differentiable for all people and also in black & and white

Colorblindness

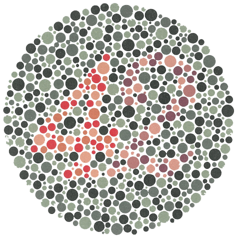
Colorblindness is not a total loss of color vision. Colorblind people can recognize a wide ranges of colors. But certain ranges of colors are hard to distinguish.



Colorblindness



6

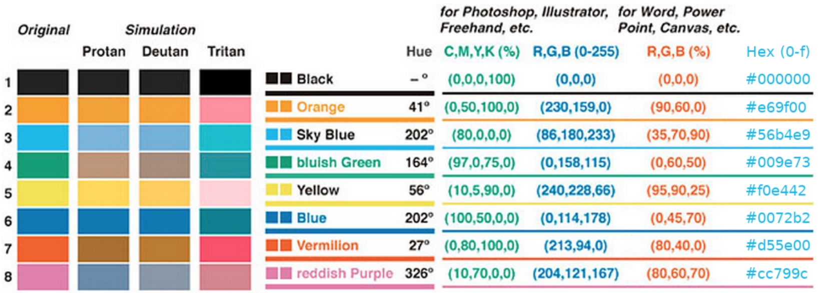


42

8% of Caucasian, 5% of Asian, and (4%) of African males are so-called "red-green" Colorblind.
Chance to have at least one Colorblind reviewer out of three is up to $1 - (1 - 0.92)^3 = 22\%$!

Colorblindness

The way to deal with Colorblindess is to use redundant encoding of information
Most Colorblind people might not be able to distinguish certain colors, but are usually able to distinguish different brightness



	Original	Simulation					for Photoshop, Illustrator, Freehand, etc.		for Word, Power Point, Canvas, etc.	
		Protan	Deutan	Tritan		Hue	C,M,Y,K (%)	R,G,B (0-255)	R,G,B (%)	Hex (0-f)
1	■	■	■	■	■■ Black	– °	(0,0,0,100)	(0,0,0)	(0,0,0)	#000000
2	■	■	■	■	■■ Orange	41°	(0,50,100,0)	(230,159,0)	(90,60,0)	#e69f00
3	■	■	■	■	■■ Sky Blue	202°	(80,0,0,0)	(86,180,233)	(35,70,90)	#56b4e9
4	■	■	■	■	■■ bluish Green	164°	(97,0,75,0)	(0,158,115)	(0,60,50)	#009e73
5	■	■	■	■	■■ Yellow	56°	(10,5,90,0)	(240,228,66)	(95,90,25)	#f0e442
6	■	■	■	■	■■ Blue	202°	(100,50,0,0)	(0,114,178)	(0,45,70)	#0072b2
7	■	■	■	■	■■ Vermilion	27°	(0,80,100,0)	(213,94,0)	(80,40,0)	#d55e00
8	■	■	■	■	■■ reddish Purple	326°	(10,70,0,0)	(204,121,167)	(80,60,70)	#cc799c

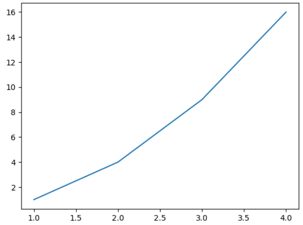
Use a color-palette taking advantage of this (either built-in or self-defined)

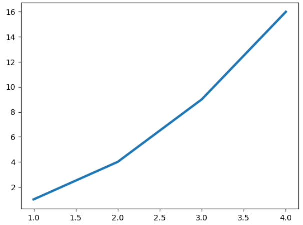
Texture

Use redundant coding. Not only Color, but also texture/patterns:

- Different markers
- Different line-styles
- Different filling-styles

Make plots visible enough using thick enough lines:

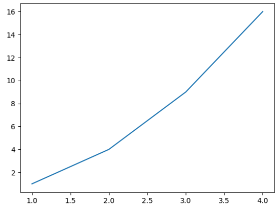




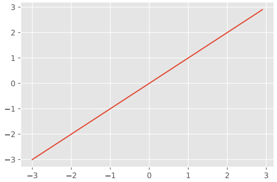
Custom styles

Matplotlib allows changing the style globally using an `.mplstyle` file.
In this file you can define almost everything, from frame line width, fonts, background color and grid, up to default figure size:

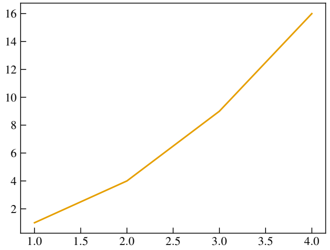
Default style



ggplot



custom style for LHCb



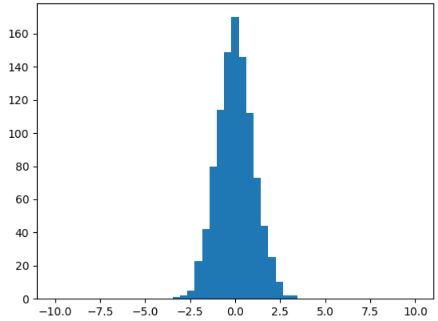
An example file with all options can be found at
https://matplotlib.org/tutorials/introductory/customizing.html

1D data

Histograms

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(1000)
bins = np.linspace(-10, 10)
plt.hist(x, bins=bins)
plt.show()
```

- See the distribution of a variable
- Can pass number of bins, range of bins or bin edges
- Set `density=True` for normalization

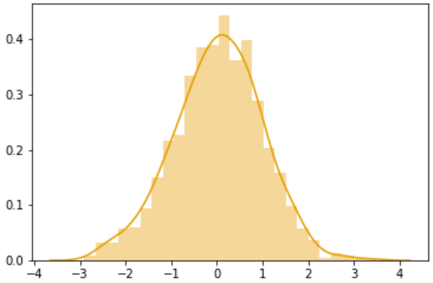


1D data

Kernel Density Estimation

```
import seaborn as sns
sns.distplot(x)
```

- Smooth estimation of a distribution
- Processes each datapoint as a gaussian centered at the point with given width (called bandwidth)
- Use `sns.kdeplot` for only the KDE
- `kdeplot` can take `cumulative=True`

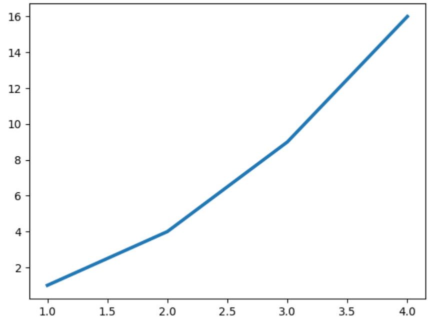


2D data

Line plots

```
x = np.array([1, 2, 3, 4])
y = x**2
plt.plot(x, y, linewidth=3)

import pandas as pd
df = pd.DataFrame({"x": x, "y":y})
df.plot("x", "y")
```

- Basic drawing command
- Can also be directly called from `pandas`

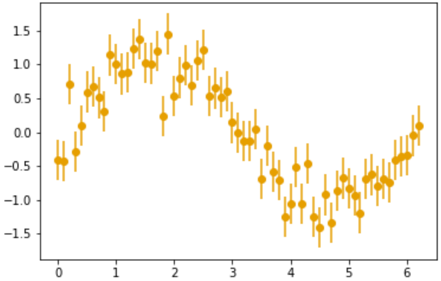


2D data

Error bars

```
x = np.arange(0, 2*np.pi, 0.1)
yerr = 0.3
noise = yerr * np.random.randn(*x.shape)
y = np.sin(x) + noise
plt.errorbar(x, y, yerr=yerr, fmt="o")
```

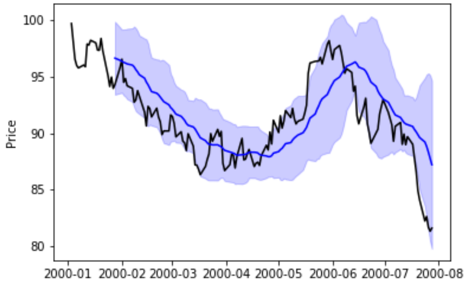
- Uncertainties are very important in science
- Can optionally take `xerr` and `yerr`
- `yerr` can be an array or a 2-tuple of arrays for asymmetric uncertainties



2D data

Filling areas

```
from numpy.random import randn

t = pd.date_range("2000-1-1", periods=150,
                  freq="B")
price = pd.Series(100+randn(150).cumsum(),
                  index=t)
avg = price.rolling(20).mean()
std = price.rolling(20).std()
plt.plot(price.index, price, "k")
plt.plot(avg.index, avg, "b")
plt.fill_between(std.index, avg-2*std,
                 avg+2*std, color="b",
                 alpha=0.2)
plt.ylabel("Price")
```



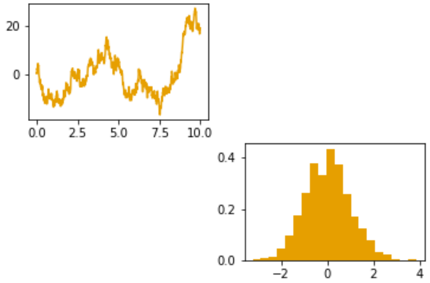
- Useful for errorbands

Intermezzo - Multiple graphs

Subplot

```
np.random.seed(42)
x = np.arange(0, 10, 0.01)
y = np.random.randn(len(x)).cumsum()
d = np.diff(y)

plt.subplot(2, 2, 1)
plt.plot(x, y)
plt.subplot(224)
plt.hist(d, bins=20, density=True)
```

- Useful for independent plots
- Use `sharex` and `sharey` if neccessary

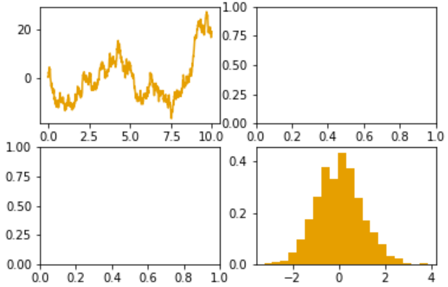


Intermezzo - Multiple graphs

Subplots

```
fig, axes = plt.subplots(2, 2)
axes[0,0].plot(x, y)
axes[1,1].hist(d, bins=20, density=True)
```

- Useful for grid of plots
- Use `sharex` and `sharey` if neccessary
- Access the axes as a `numpy` array

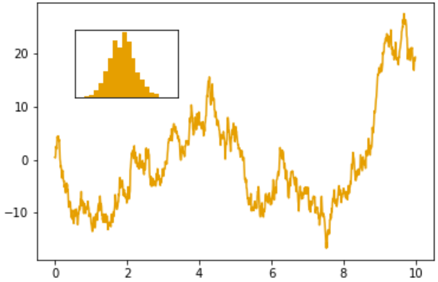


Intermezzo - Multiple graphs

Plot-in-plot

```
plt.plot(x, y)
plt.axes([0.2, .6, .2, .2])
plt.hist(d, bins=20, density=True)
plt.xticks([])
plt.yticks([])
```

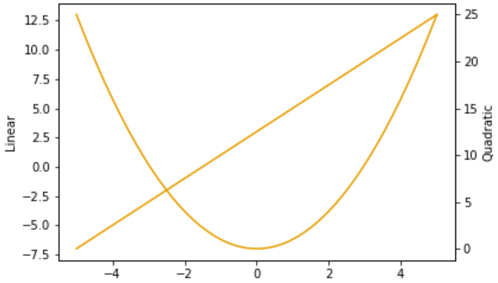
- Ideal for summary plot or zoomed version
- Can turn off the axes
- Freely placeable



Intermezzo - Multiple graphs

Twin axes

```
plt.figure()
x = np.linspace(-5, 5)
y = 2*x + 3
y2 = x**2

ax1 = plt.gca()
ax1.plot(x, y)
ax1.set_ylabel("Linear")
ax2 = ax1.twinx()
ax2.plot(x, y2)
ax2.set_ylabel("Quadratic")
```



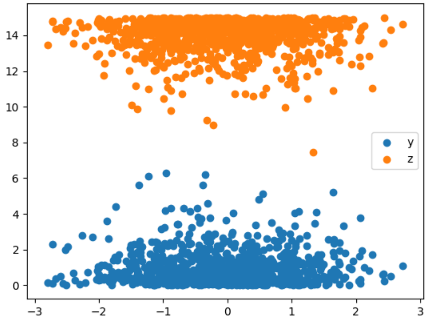
- Two completely independent axes
- Use `plt.twiny` for an additional x-axis
- Have to build legend manually

2D data

Scatter

```
from numpy.random import normal
from numpy.random import exponential

x = randn(1000)
y = exponential(1, 1000)
z = 15 - exponential(1, 1000)
plt.scatter(x, y, label="y")
plt.scatter(x, z, label="z")
plt.legend()
plt.savefig("figs/plt_scatter.png")
```

- Good at getting a feel for the data
- Bad for many datapoints

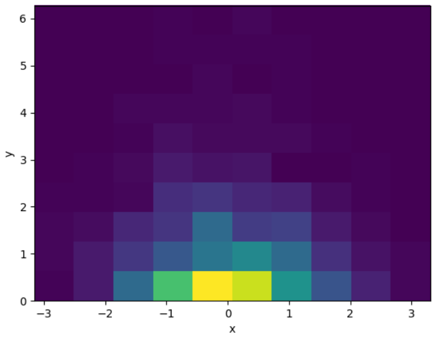


2D data

2D histogram

```
x = randn(1000)
y = exponential(size=1000)
plt.hist2d(x, y)
plt.xlabel("x")
plt.ylabel("y")
```

- Can take arbitrary binning like in 1D
- Also works for lots of data

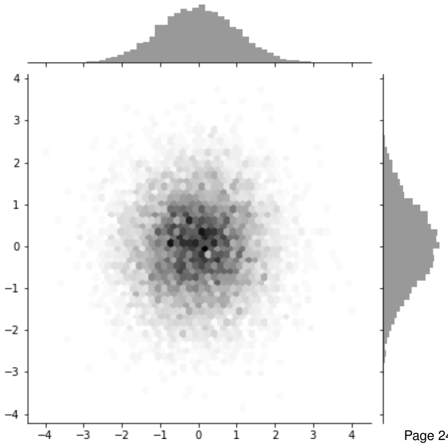


2D data

2D hexagonal histogram

```
x, y = randn(2, 1000)
sns.jointplot(x, y, kind="hex", color="k")
```

- Sometimes nicer to look at than square bins
- Has marginal distributions by default
- `matplotlib` also has a simpler version, `plt.hexbin`

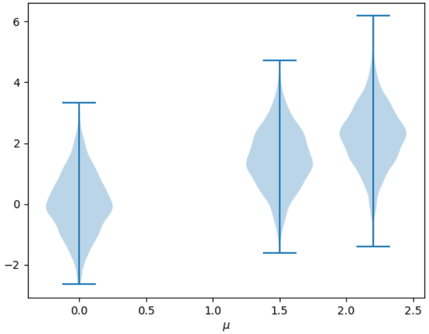


2D data

Violin plots

```
mus = 0, 1.5, 2.2
data = [normal(mu, 1, 1000) for mu in mus]
plt.violinplot(data, positions=mus)
plt.xlabel(r"$\mu$")
```

- More information than just plotting mean vs μ
- Can add plotting of individual data points, quantiles, etc

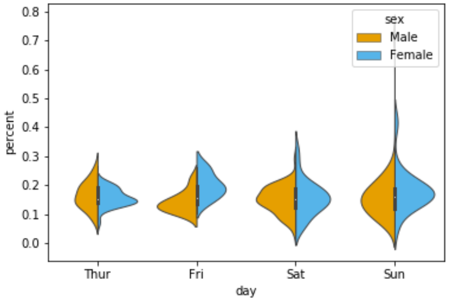


(2+1)D data

Split violin plots

```
import seaborn as sns
tips = sns.load_dataset("tips")

tips["percent"] = tips.tip/tips.total_bill
sns.violinplot("day", "percent", "sex",
               data=tips, split=True)
```

- Allows one more distinction via the two halves or more by putting them next to each other
- Good for additional category with few states

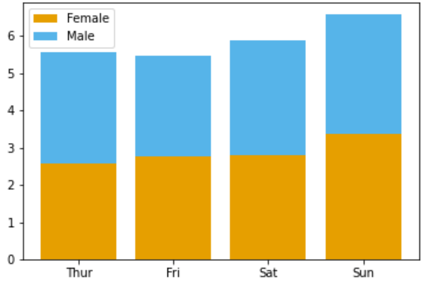


2D data

(Stacked) bar charts

```
bottom = 0
for sex, df in tips.groupby("sex"):
    df = df.groupby("day").tip.mean()\
           .reset_index()
    plt.bar(df["day"], df["tip"], 0.8,
            bottom, label=sex)
    bottom = df["tip"]
plt.legend()
```

- Good for one or two categories also with multiple states
- Shows the composition well, but not the development of each component

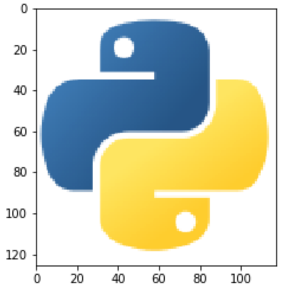


2D data

Images

```
path = "figs/python.png"
img = plt.imread(path)
fig1 = plt.imshow(img)
```

- `scipy.ndimage.imread` now deprecated
- Internally stored as a (2+1)D `numpy` array, so you can use fancy indexing on/with it

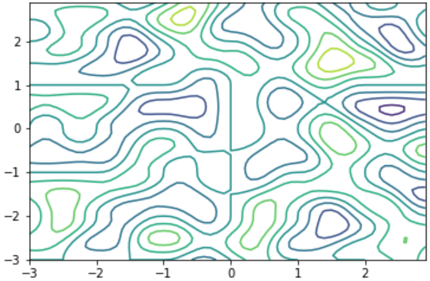


3D data

Contour plots

```
import noise
pnoise2 = np.vectorize(noise.pnoise2)
x = np.arange(-3, 3, 0.1)
y = np.arange(-3, 3, 0.1)
X, Y = np.meshgrid(x, y)
z = pnoise2(X, Y)
plt.contour(X, Y, z)
```

- Lines show fixed values, encoding in color
- Suitable for printing (no fancy gradients)

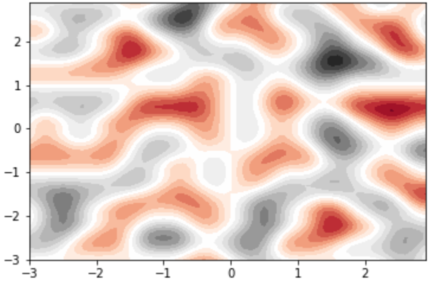


3D data

Filled contours

```
plt.contourf(X, Y, z, 20, cmap='RdGy')
```

- Contains more information than height lines
- Can use any colormap

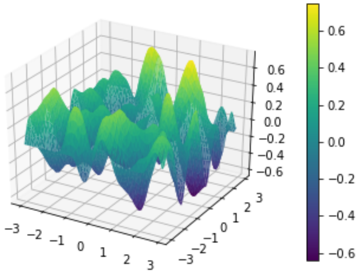


3D data

Surface plots

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

cmap = plt.cm.viridis
x, y, z = X.flatten(), Y.flatten(), z.flatten()
surf = ax.plot_trisurf(x, y, z, cmap=cmap)
plt.colorbar(surf)
```

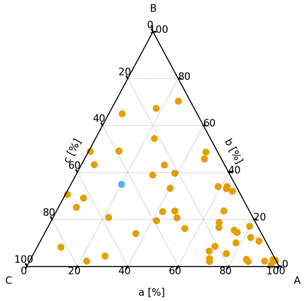
- Easy to immediately grasp
- Can zoom/rotate in interactive environments



3D data

Ternary plots

```
import ternary
fig, tax = ternary.figure(scale=100)
fig.set_size_inches(5, 5)
a=100*rand(50);b=(100-a)*rand(50);c=100-a-b
tax.scatter(np.array([[a, b, c]]))
tax.scatter([[20, 35, 45]])
tax.right_corner_label("A")
tax.top_corner_label("B")
tax.left_corner_label("C")
tax.bottom_axis_label("a [%]")
...
tax.gridlines(multiple=20, color="gray")
tax.ticks(axis='lbr', multiple=20)
tax.boundary(linewidth=1)
tax.get_axes().axis('off')
```



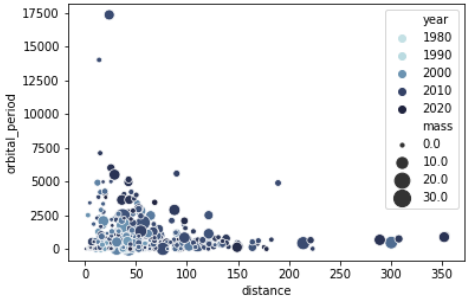
- For analyzing composition with three components that sum to a constant
- Need `python-ternary` for this

4D data

Enhanced scatter plot

```
planets = sns.load_dataset("planets")
cmap = sns.cubehelix_palette(rot=-.2,
                             as_cmap=True)
ax = sns.scatterplot(x="distance",
                     y="orbital_period",
                     hue="year",
                     size="mass",
                     palette=cmap,
                     sizes=(10, 200),
                     data=planets)
```

- Sometimes you can encode information in color and size of markers

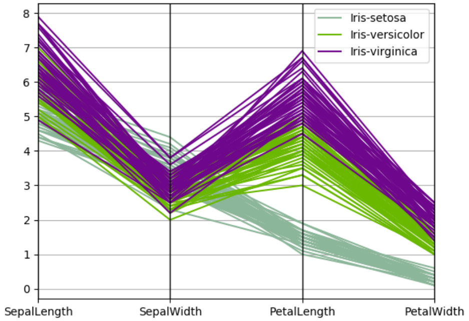


ND data

Parallel coordinates

```
from pandas.plotting import parallel_coordinates

iris = pd.read_csv("data/iris.csv")
parallel_coordinates(data, "Name")
```

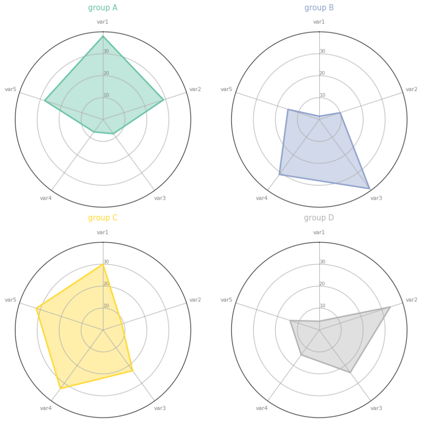
- Works for an arbitrary number of dimensions
- Results may vary according to order of dimensions
- Can only see broad features
- Best for distinguishing groups in multiple dimensions



ND data

Radar plot

- Can be used to compare few examples in many dimensions
- No easy implementation available
- Find this example here

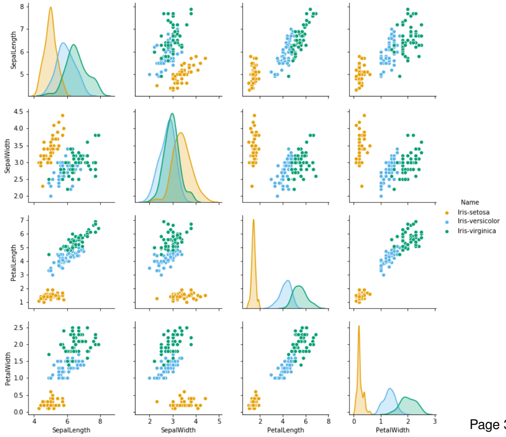


ND data

Pairplot

```
sns.pairplot(iris, diag_kind="kde",
             hue="Name")
```

- Plots each variables correlation with each other variable
- Can be used to find correlations between two variables out of many
- Easy to find a simple cut for classification
- Can even add automatic linear regression

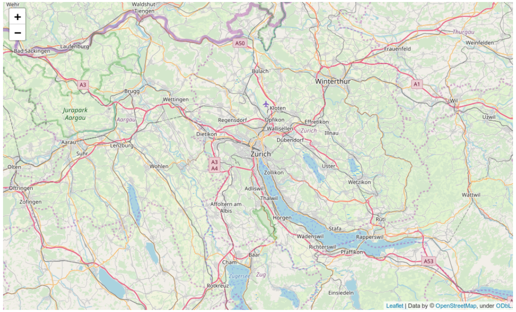


Geospatial data

Folium

```
import folium
m = folium.Map(location=[47.3686, 8.5391])
```

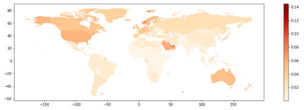
- Takes data from OpenStreetMap
- Interactive visualization via javascript in the browser
- No easy way to save the resulting map



Geospatial data

Geopandas

```
import geopandas
fig = plt.figure(figsize=(20, 5))
ax = fig.gca()
world = geopandas.read_file(
    geopandas.datasets.get_path(
        'naturalearth_lowres'))
world = world[(world.pop_est>0)
            & (world.name!="Antarctica")]
world['gdp_per_cap'] = world.gdp_md_est \
                     / world.pop_est
world.plot(column='gdp_per_cap', ax=ax,
         legend=True, cmap="OrRd")
```



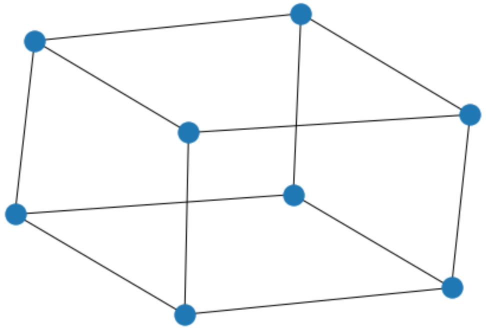
- Has a low-res version of all countries included
- Can read shapefiles in many common formats
- Combines them with `pandas` dataframes

Networks

Networkx

```
import networkx as nx
g = nx.cubical_graph()
nx.draw(g)
```

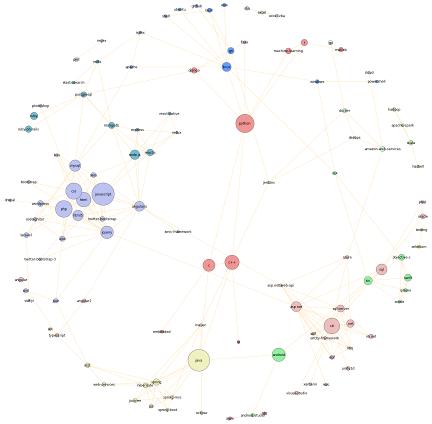
- Automatically positions the nodes according to the weights on the nodes
- Many common graphs included
- Many customizations possible



Networks

Networkx

- Automatically positions the nodes according to the weights on the nodes
- Useful for text analysis



Resources

- Pyplot tutorial: https://matplotlib.org/users/pyplot_tutorial.html
- Matplotlib documentation: https://matplotlib.org/api/pyplot_summary.html
- Custom style-sheets: https://matplotlib.org/users/customizing.html
- Pandas plotting documentation: https://pandas.pydata.org/pandas-docs/stable/visualization.html
- Seaborn documentation: https://seaborn.pydata.org/

More tools

Argparse

Easy parsing of commandline options using argparse.

```
import argparse

parser = argparse.ArgumentParser(description="Process some integers.")
parser.add_argument("integers", metavar="N", type=int, nargs="+",
                    help="an integer for the accumulator")
parser.add_argument("--sum", dest="accumulate", action="store_const",
                    const=sum, default=max,
                    help="sum the integers (default: find the max)")

args = parser.parse_args()
print(args.accumulate(args.integers))

$ script.py --sum 1 2 3 4
10
```

Webscraping

Requests

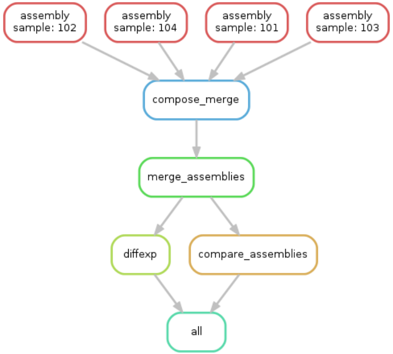
`requests` perform web-requests, both GET and POST (and more) to interact with anything reachable over the internet.
`BeautifulSoup` parses XML/HTML documents.

```
import requests
from bs4 import BeautifulSoup

# Re-use the connection to the server
session = requests.Session()
# Get the webpage
response = session.get(url)
# Fail early if unexpected response
response.raise_for_status()
# Read it into a datastructure that is easy to query
soup = BeautifulSoup(response.text, "lxml")
links = [a['href'] for a in soup.select("a.internal")]
```

Snakemake

Automate your analysis flow using snakemake.

```
rule targets:
    input:
        "plots/dataset1.pdf",
        "plots/dataset2.pdf"

rule plot:
    input:
        "raw/{dataset}.csv"
    output:
        "plots/{dataset}.pdf"
    shell:
        "somecommand {input} {output}"
```



Subprocess

Sometimes you need to run external commands, for which no Python module exists. This can be done with the `subprocess` module.
It has recently (Python 3.7) been simplified a lot:

```
import subprocess

result = subprocess.run(["du", "-h", "."], capture_output=True)
print(result.stdout)
print(result.stderr)
# ...

result2 = subprocess.run(["cat"], capture_output=True, input=b"test")
print(result2.stdout)
# b'test'
```

Frameworks

Some fields have even created their own toolkits:

- Computational biology: https://biopython.org/
- Astronomy: http://www.astropy.org/
- High-energy particle physics: https://github.com/scikit-hep