Visualisierung mit Matplotlib

Vortrag

Robin Klimczak

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2022-25-05

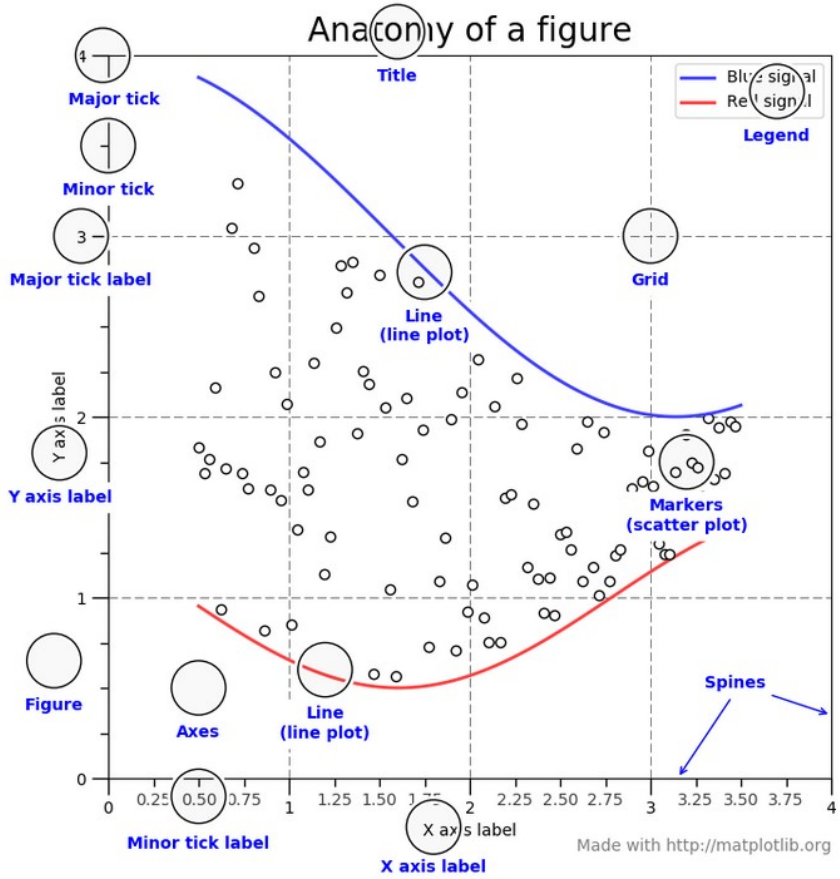
Gliederung (Agenda)

Überblick

- "Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python"
- Programmbibliothek für Python
- ermöglicht Datenvisualisierung
- erste Version 2003
- Open Source

Komponenten

Übersicht Komponenten [0]



Komponente

- figure: Die Abbildung als Grundbaustein
- axes: Region des Bildes mit Daten
- axis: es existieren 2 oder 3 axis-Objekte [0]
- weitere Komponente sind: label, line, grid, legend, title

```
1 x = [1, 2, 3]
2 fig, ax = plt.subplots()
3 ax.plot(x, x, label='linear')
```

Listing 1: Figure / Axes erstellen

Erste Schritte

- Installation Terminal:
- pip = Paketverwaltungsprogramm für Python-Pakete

```
1 pip install matplotlib
```

Listing 2: Programmcode SetUp

- import von:
  - matplotlib (numerisches Python)
  - numpy (numerisches Python)
  - Pakete von Drittanbietern (z.B. panda, conda)
- Zusatzbefehl in Jupyter Notebook notwendig

Erste Schritte

```
1 %matplotlib notebook
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import pandas as p
```

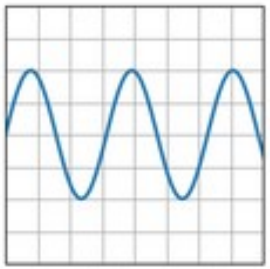
Listing 3: Programmcode SetUp

Erste Schritte

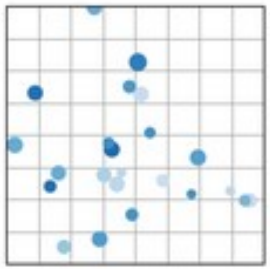
- gut gepflegte Dokumentation (https://matplotlib.org) [1]
- API Reference = aktuelles und vollständiges Nachschlagewerk aller:
  - Objekte
  - Methoden
- Beispiele und Tutorials
- Auflistung aller Plot types

Plot types

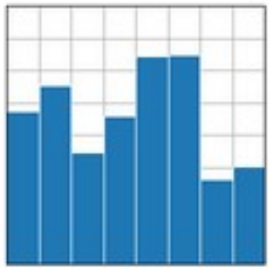
Basic: "Basic plot types, usually y versus x."[2]



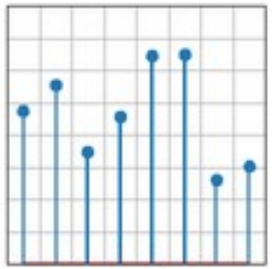
plot(x, y)



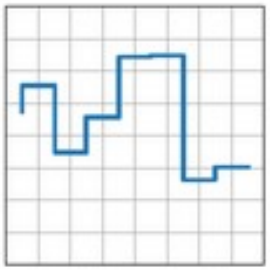
scatter(x, y)



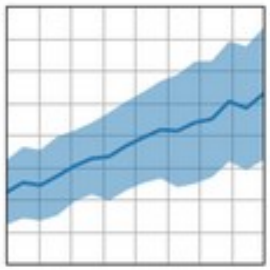
bar(x, height) / barh(y, width)



stem(x, y)



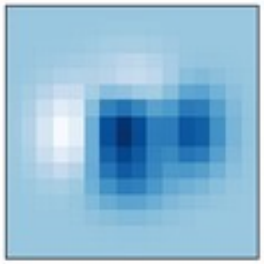
step(x, y)



fill_between(x, y1, y2)

Plot types

Plots of arrays and fields: "Plotting for arrays of data Z(x, y) and fields U(x, y), V(x, y)."[2]

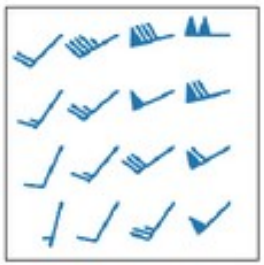


imshow(Z)

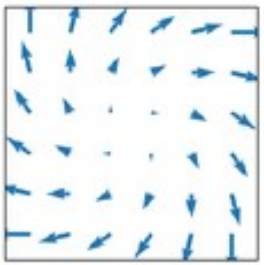
pcolormesh(X, Y, Z)

contour(X, Y, Z)

contourf(X, Y, Z)



barbs(X, Y, U, V)

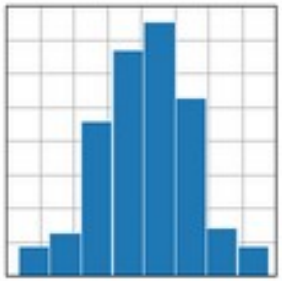


quiver(X, Y, U, V)

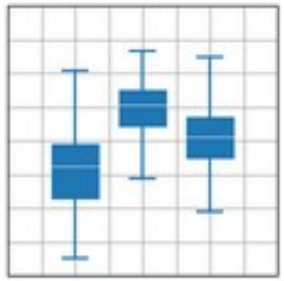
streamplot(X, Y, U, V)

Plot types

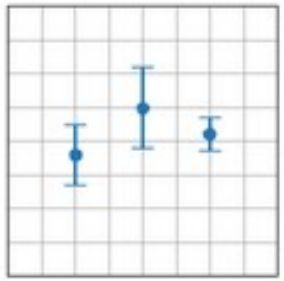
Statistics plots: "Plots for statistical analysis."[2]



hist(x)

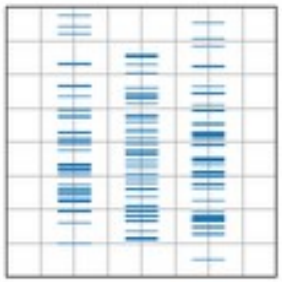


boxplot(X)



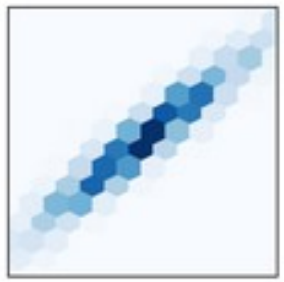
errorbar(x, y, yerr, xerr)

violinplot(D)



eventplot(D)

hist2d(x, y)

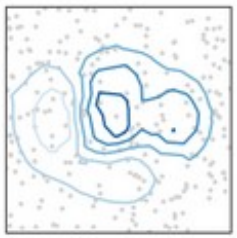


hexbin(x, y, C)

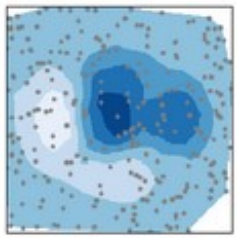
pie(x)

Plot types

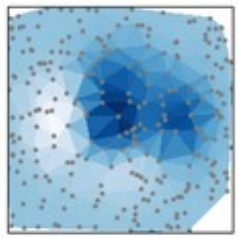
Unstructured coordinates: SSometimes we collect data z at coordinates (x,y) and want to visualize as a contour. Instead of gridding the data and then using contour, we can use a triangulation algorithm and fill the triangles.."[2]



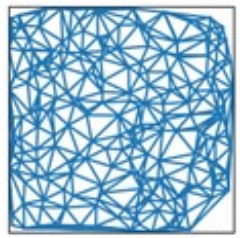
tricontour(x, y, z)



tricontourf(x, y, z)

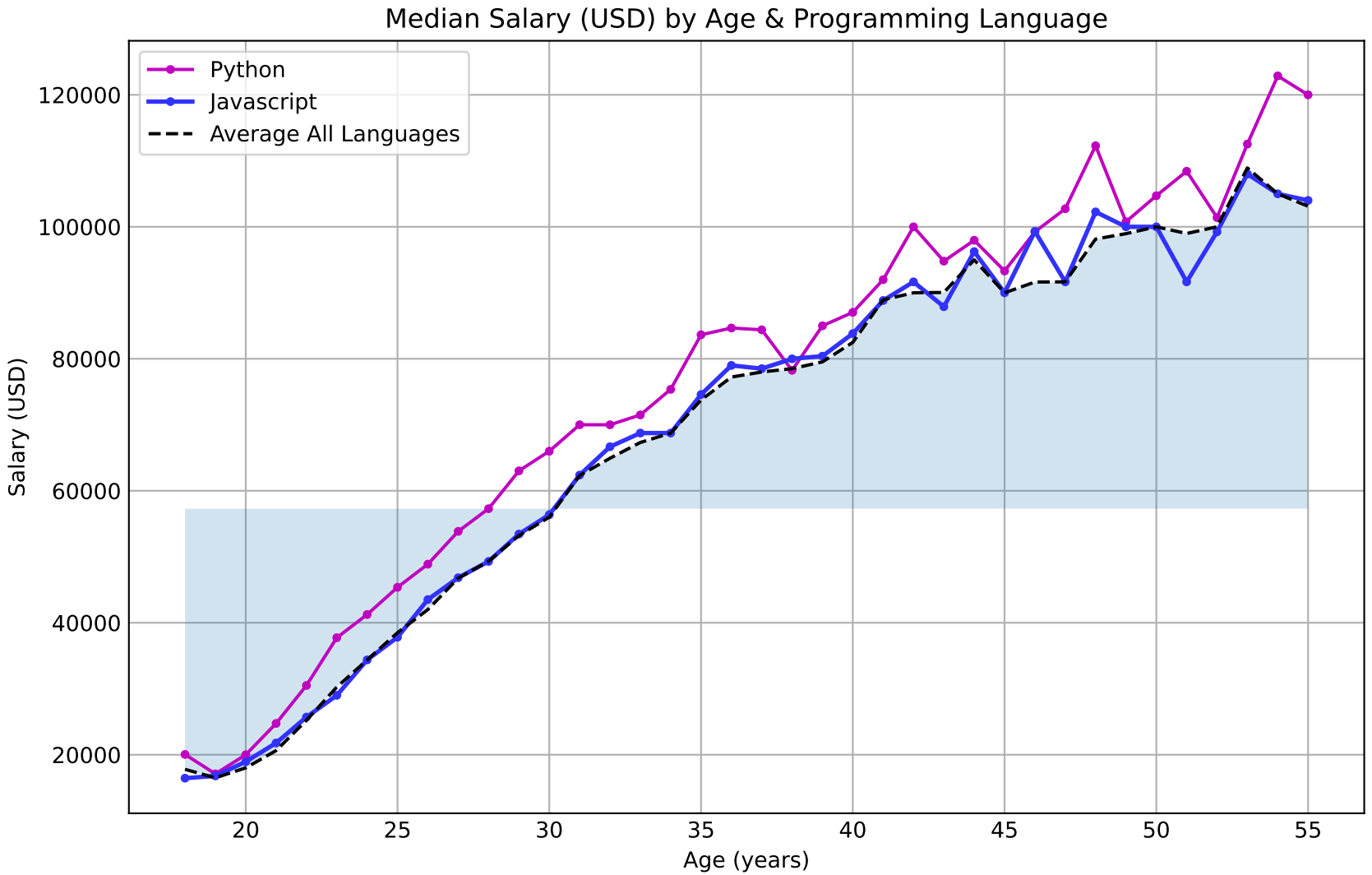


tripcolor(x, y, z)



triplot(x, y)

Implementationsbeispiele



Implementationsbeispiele

```
1  from matplotlib import pyplot as plt
2
3  fig = plt.figure()
4  fig.set_figheight(5)
5  fig.set_figwidth(8)
6
7  ages_x = [18, 19, ... , 54, 55]
8  py_dev_y = [20046, 17100, ... , 122870, 120000]
9  js_dev_y = [16446, 16791, ... , 105000, 104000]
10 dev_y = [17784, 16500, ... , 105000, 103117]
11
12 overall_median = 57287
```

Listing 4: Liniendiagramm 1

Implementationsbeispiele

```
1 ax = fig.add_subplot()
2 ax.plot(ages_x, py_dev_y, marker='.', color='m',
      ↪ label='Python')          #'o-m'
3 ax.plot(ages_x, js_dev_y, marker='.',
      ↪ color='#3232FF', linewidth='2',
      ↪ label='Javascript')
4 ax.plot(ages_x, dev_y, color='k', linestyle='--',
      ↪ label='Average All Languages')
5
6 ax.set_title('Median Salary (USD) by Age &
      ↪ Programming Language')
7 ax.set_ylabel('Salary (USD)')
8 ax.set_xlabel('Age (years)')
9 ax.legend()
```

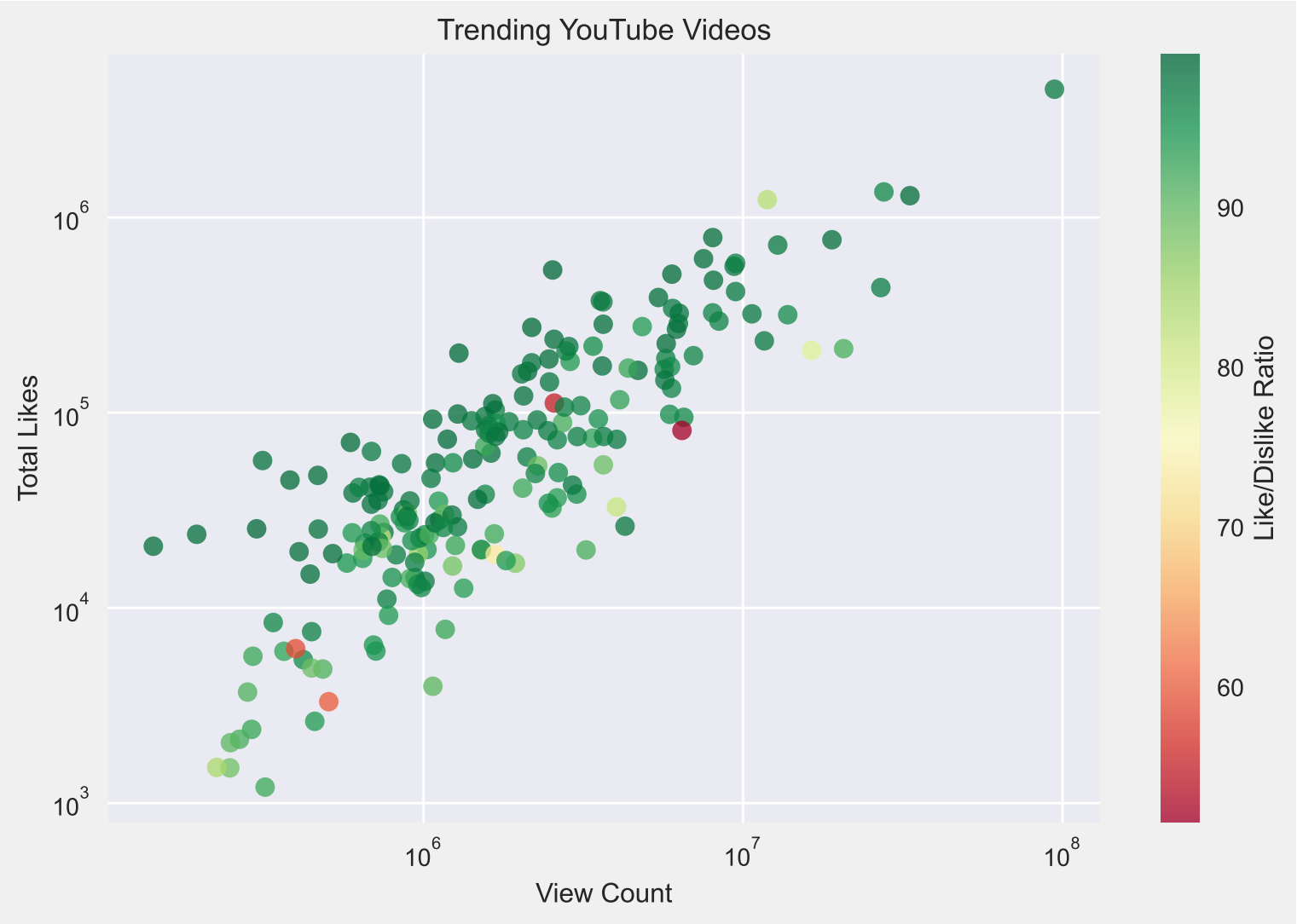
Listing 5: Liniendiagramm 2

Implementationsbeispiele

```
1 plt.fill_between(ages_x, dev_y, overall_median,
      ↪ interpolate=True, alpha=0.2)
2
3 plt.savefig('diagram.pdf', bbox_inches='tight',
      ↪ dpi=150)
4 plt.show()
```

Listing 6: Liniendiagramm 3

Implementationsbeispiele



Implementationsbeispiele

```
1  import pandas as pd
2  from matplotlib import pyplot as plt
3
4  plt.style.use('seaborn')
5
6  data =
       ↪ pd.read_csv('vollstaendigerPfad\Top200VideosTrend.csv
7  view_count = data['view_count']
8  likes = data['likes']
9  ratio = data['ratio']
10
11 fig, ax = plt.subplots()
12 ax.scatter(view_count, likes, c=ratio, cmap='RdYlGn',
       ↪ edgecolor='black', linewidth=1, alpha=0.75)
```

Listing 7: Streudiagram 1

Implementationsbeispiele

```
1  cbar = plt.colorbar()
2  cbar.set_label('Like/Dislike Ratio')
3
4  ax.set_xscale('log')
5  ax.set_yscale('log')
6
7  ax.set_title('Trending YouTube Videos')
8  ax.set_xlabel('View Count')
9  ax.set_ylabel('Total Likes')
10
11 plt.savefig('diagram2.pdf', bbox_inches='tight',
       ↪ dpi=150)
12
13 plt.show()
```

Listing 8: Streudiagram 2

Implementationsbeispiele

Dynamische Darstellungen

- Darstellung von Live-Daten
- Darstellung von Entwicklungen in Zeitabhängigkeit
- Jeder Datensatz kann dynamisch dargestellt werden

Zusammenfassung

- leistungsfähige Pyhton Programmbibliothek
- viele verschiedene Visualisierungstypen
- hohe Verbreitung
- gute Dokumentation
- Darstellung von statischen und dynamischen Grafiken

Literatur

[0] https://matplotlib.org/stable/tutorials/introductory/usage.html
[1] https://matplotlib.org/
[2] https://matplotlib.org/stable/plottypes/index