



## Datenanalyse und Einführung in Maschinelles Lernen WS 2025/26

### Datenanalyse mit Python

Dozentin: Grit Behrens  
mailto: [grit.behrens@hsbi.de](mailto:grit.behrens@hsbi.de)



# Lehrinhalte

1. Einstieg zu Datenanalyse und Einführung in Maschinelles Lernen
- 2. Datenanalyse mit Python**











# Datenanalyse mit Python

1. Python für Datenanalyse
2. Datenvisualisierung mit Matplotlib
3. Pandas Datenstrukturen
4. Pandas DataFrame
5. Pandas Dateiverarbeitung
6. Statistische Maße und Analysen
7. Datenvisualisierung mit Pandas
8. Zeit und Datum, Zeitreihen

# Datenanalyse mit Python

- 1. Python für Datenanalyse**
2. Datenvisualisierung mit Matplotlib
3. Pandas Datenstrukturen
4. Pandas DataFrame
5. Pandas Dateiverarbeitung
6. Statistische Maße und Analysen
7. Datenvisualisierung mit Pandas
8. Zeit und Datum, Zeitreihen

# Python für die Datenanalyse

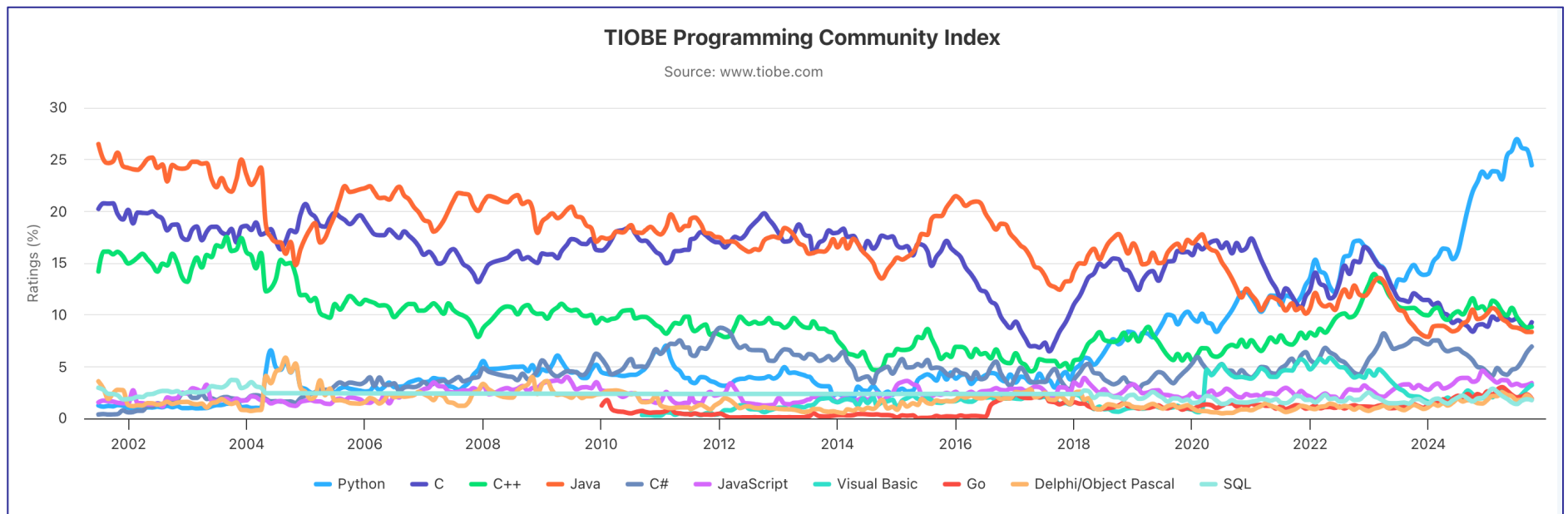
| Oct 2025 | Oct 2024 | Change | Programming Language  |                      | Ratings | Change |
|----------|----------|--------|---|----------------------|---------|--------|
| 1        | 1        |        |    | Python               | 24.45%  | +2.55% |
| 2        | 4        | ▲      |    | C                    | 9.29%   | +0.91% |
| 3        | 2        | ▼      |    | C++                  | 8.84%   | -2.77% |
| 4        | 3        | ▼      |    | Java                 | 8.35%   | -2.15% |
| 5        | 5        |        |    | C#                   | 6.94%   | +1.32% |
| 6        | 6        |        |    | JavaScript           | 3.41%   | -0.13% |
| 7        | 7        |        |    | Visual Basic         | 3.22%   | +0.87% |
| 8        | 8        |        |    | Go                   | 1.92%   | -0.10% |
| 9        | 10       | ▲      |  | Delphi/Object Pascal | 1.86%   | +0.19% |
| 10       | 11       | ▲      |  | SQL                  | 1.77%   | +0.13% |

Python ist **weltweit die populärste Programmiersprache**.

Der Erfolg beruht vor allem auf Anwendungen in Big Data-Datenanalyse und im Maschinellen Lernen.

*Oktober 2025 [www.tiobe.com/tiobe-index](http://www.tiobe.com/tiobe-index)*

# Python für die Datenanalyse



Oktober 2025 [www.tiobe.com/tiobe-index](http://www.tiobe.com/tiobe-index)

# Anaconda und Jupyter- Notebook für die Datenanalyse

## Empfehlung

*Jupyter Notebook* ist eine Web-Integrierte Entwicklungsumgebung – Web Integrated Development Environment (WIDE).

- interaktive Entwicklungsumgebung
- nutzbar auch für Visualisierungen und Präsentationen
- in der **Anaconda-Distribution** werden fundamentale Bibliotheken für Data-Mining mit ausgeliefert.





# Fundamentale Python Libraries für die Datenanalyse

## **NumPy – Numeric Computation** ([www.scipy.org](http://www.scipy.org))

Effiziente Datenstruktur in Form von multidimensionalen Arrays mit grundlegenden Operatoren und algebraischen Funktionen



## **SciPy- Scientific Computation** ([www.scipy.org](http://www.scipy.org))

Numerische Algorithmen und spezielle Toolboxes wie z.B. Signalverarbeitung, Optimierung, Statistik.

**Matplotlib** – zur Datenvisualisierung setzt auf SciPy auf



## **Scikit-Learn – Machine Learning in Python** ([www.scipy.org](http://www.scipy.org))

nutzt NumPy, SciPy und Matplotlib

enthält Algorithmen für z.B. Vorverarbeitung, Klassifikation, Regression, Clustering, Dimensionsreduktion



## **Pandas- Python Data Analysis** ([www.pandas.pydata.org](http://www.pandas.pydata.org))

hoch-performante Datenstrukturen und Datenanalyse-Tools

„DataFrame“-Objekt für Datenmanipulation mit integrierter Indexierung  
Import und Export von Datenformaten wie CSV, Text, Excel, SQL



## **Matplotlib- Visulization** (<https://matplotlib.org/>)



## **KERAS – Deep Learning** (<https://matplotlib.org/>)



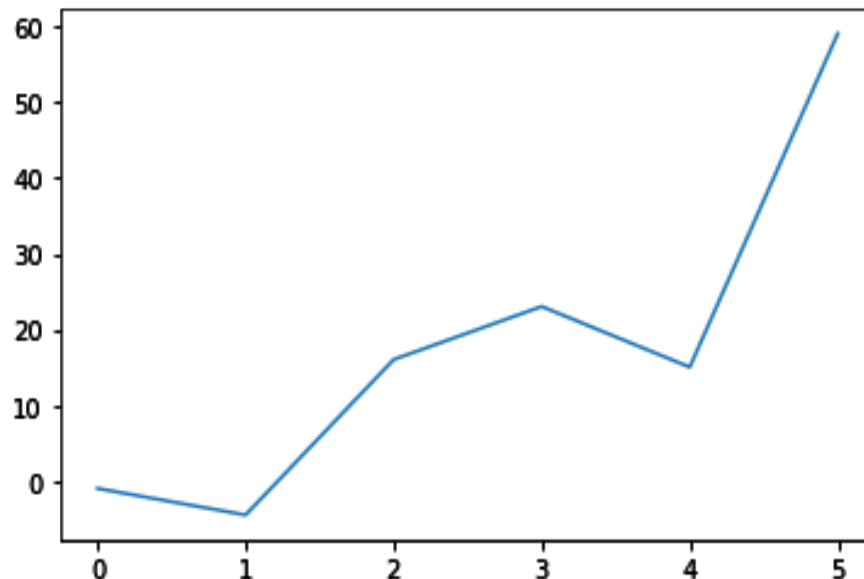
# Datenanalyse mit Python

1. Python für Datenanalyse
- 2. Datenvisualisierung mit Matplotlib**
3. Pandas Datenstrukturen
4. Pandas DataFrame
5. Pandas Dateiverarbeitung
6. Datenvisualisierung mit Pandas
7. Zeit und Datum, Zeitreihen

- Modul `pyplot` stellt prozedurale Schnittstelle zur objektorientierten Plot-Bibliothek von Matplotlib zur Verfügung
- Kommandos und Namen sind nach Vorbild von Matlab

```
#einfachstes Beispiel pyplot
import matplotlib.pyplot as plt

plt.plot([-1, -4.5, 16, 23, 15, 59])
plt.show()
```

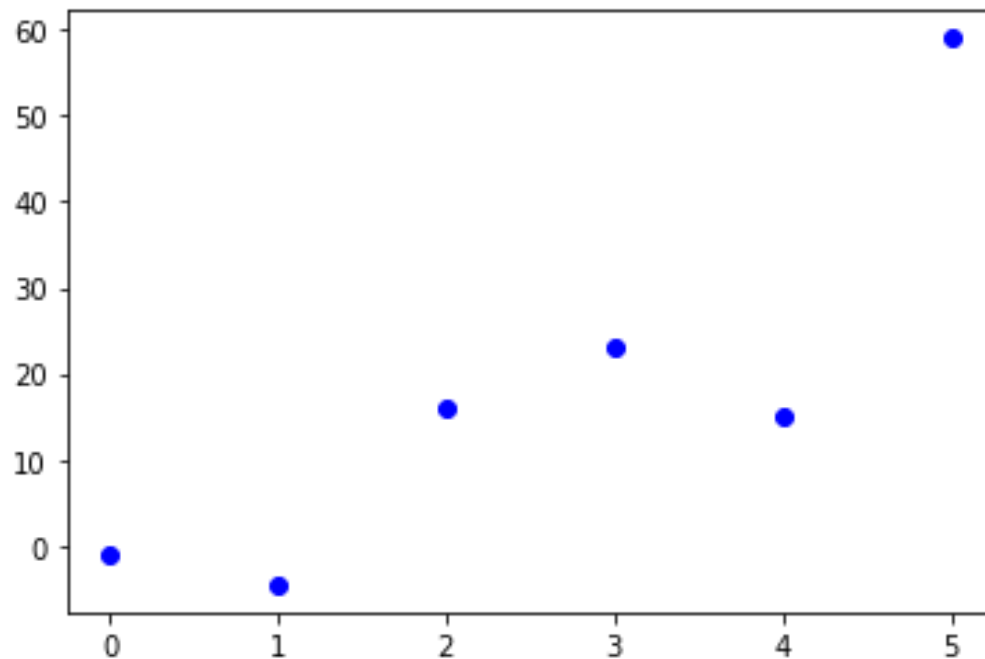


*Indizes der Liste -> Werte für x-Achse  
Listenwerte -> y-Achse*

- **Formatstring Syntax:** *„Linienstil oder Darstellung der diskreten Werte ,Farbe“*
  - *Defaultwert ist ,b-‘ - blaue Linie*

```
#Beispiel pyplot mit Formatübergabe 11-2
import matplotlib.pyplot as plt

plt.plot([-1, -4.5, 16, 23, 15, 59], "ob" )
plt.show()
```



*Formatstring „ob“ – blaue Vollkreise*

*z.B. :*

'-' (Bindestrich) durchgezogene Linie  
'--' (zwei Bindestriche) gestrichelte Linie  
'-.' Strichpunkt-Linie  
'o' Kreis-Marker  
'v' Dreiecks-Marker, Spitze nach unten....  
...  
'b' blau  
'g' grün  
'r' rot  
'y' gelb  
'k' schwarz  
'w' weiß

# Datenvisualisierung mit Matplotlib



```
#Beispiel pyplot mit Übergabe der x-Werte an plot
```

```
import matplotlib.pyplot as plt
```

```
# die X-Werte:
```

```
days = list(range(0, 22, 3))
```

```
print(days)
```

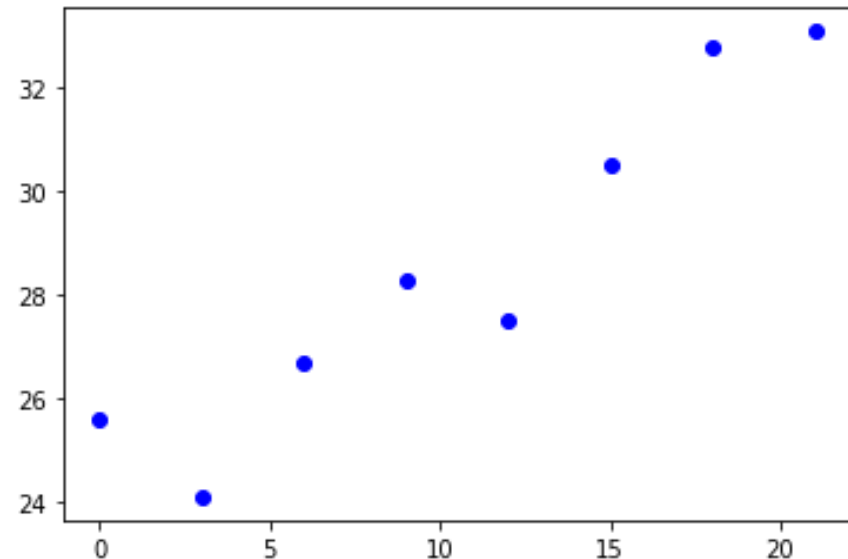
```
# die Y-Werte:
```

```
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
```

```
plt.plot(days, celsius_values, "ob")
```

```
plt.show()
```

Ausgabe: [0, 3, 6, 9, 12, 15, 18, 21]



# Datenvisualisierung mit Matplotlib



#Beispiel pyplot mit **Bezeichnung für die Achsen**

```
import matplotlib.pyplot as plt
```

```
days = list(range(1,9))
```

```
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8,  
33.1]
```

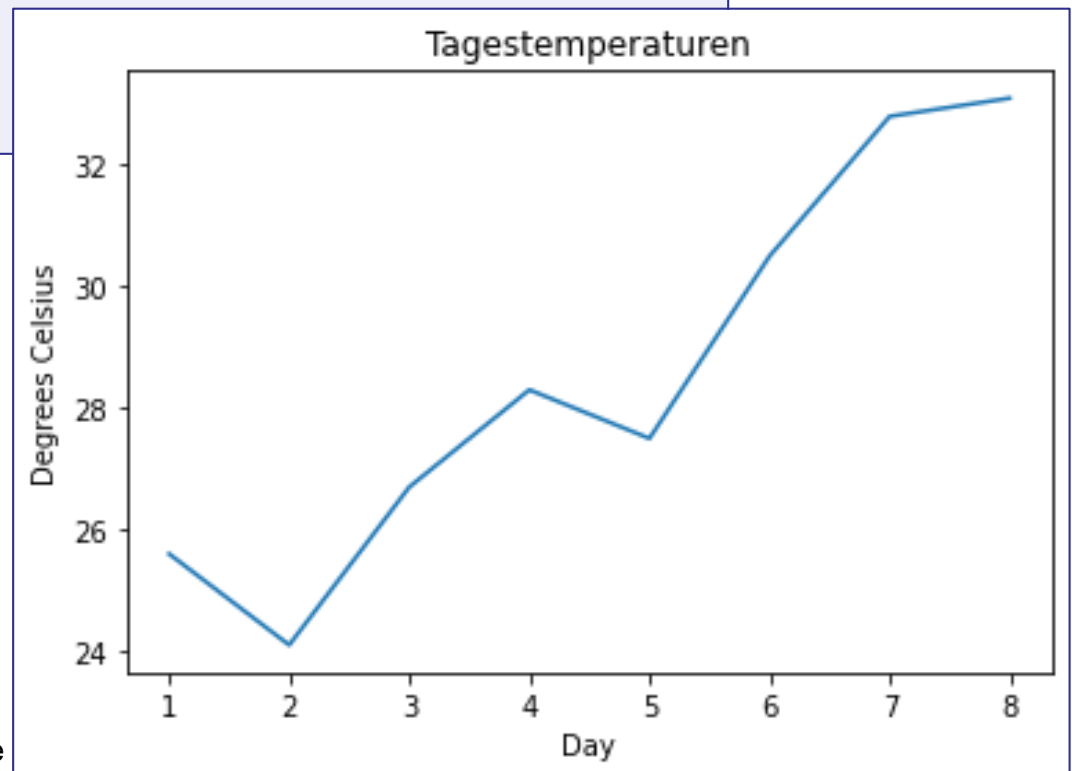
```
plt.plot(days, celsius_values)
```

```
plt.xlabel('Day')
```

```
plt.ylabel('Degrees Celsius')
```

```
plt.title('Tagestemperaturen')
```

```
plt.show()
```



# Datenvisualisierung mit Matplotlib



## #Achsen abfragen und manipulieren, mehrere Datenreihen für y-Achse

```
import matplotlib.pyplot as plt
```

```
days = list(range(1,9))
```

```
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
```

```
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
```

```
plt.xlabel('Day')
```

```
plt.ylabel('Degrees Celsius')
```

```
plt.title('Tagestemperaturen')
```

```
plt.plot(days, celsius_min,  
         days, celsius_min, "oy",  
         days, celsius_max,  
         days, celsius_max, "or")
```

```
print("The current limits for the axes are:")
```

```
print(plt.axis())
```

```
print("We set the axes to the following values:")
```

```
xmin, xmax, ymin, ymax = 0, 10, 14, 45
```

```
print(xmin, xmax, ymin, ymax)
```

```
plt.axis([xmin, xmax, ymin, ymax])
```

```
plt.show()
```

### Ausgabe:

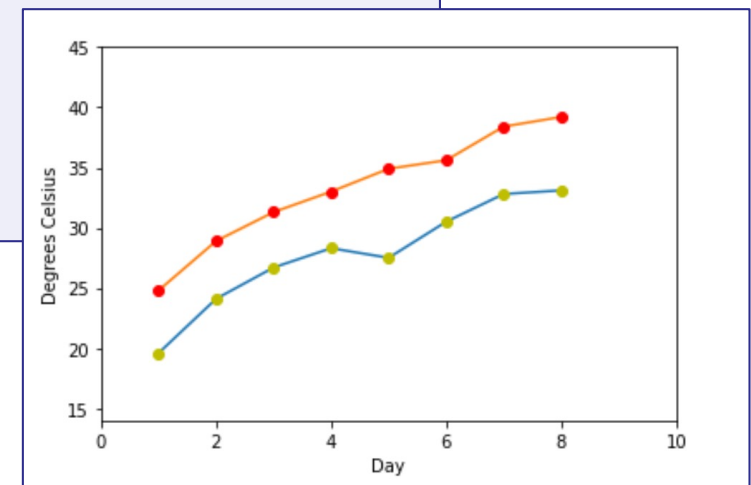
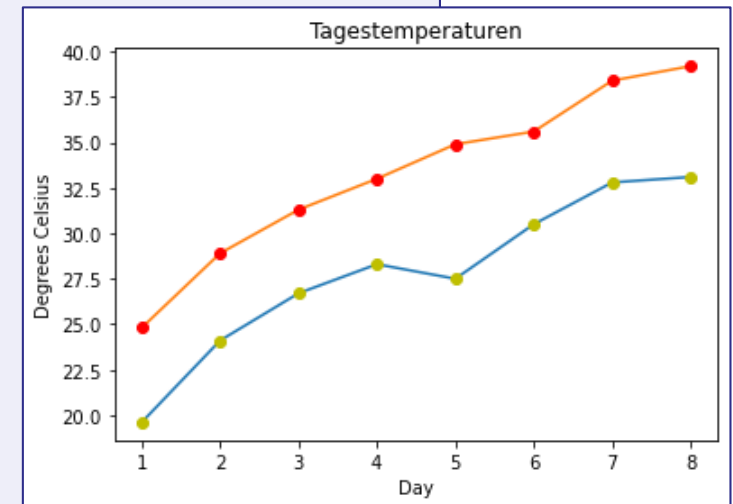
The current limits for the axes are:

(0.6499999999999999, 8.35, 18.62, 40.18)

We set the axes to the following values:

0 10 14 45

t Python



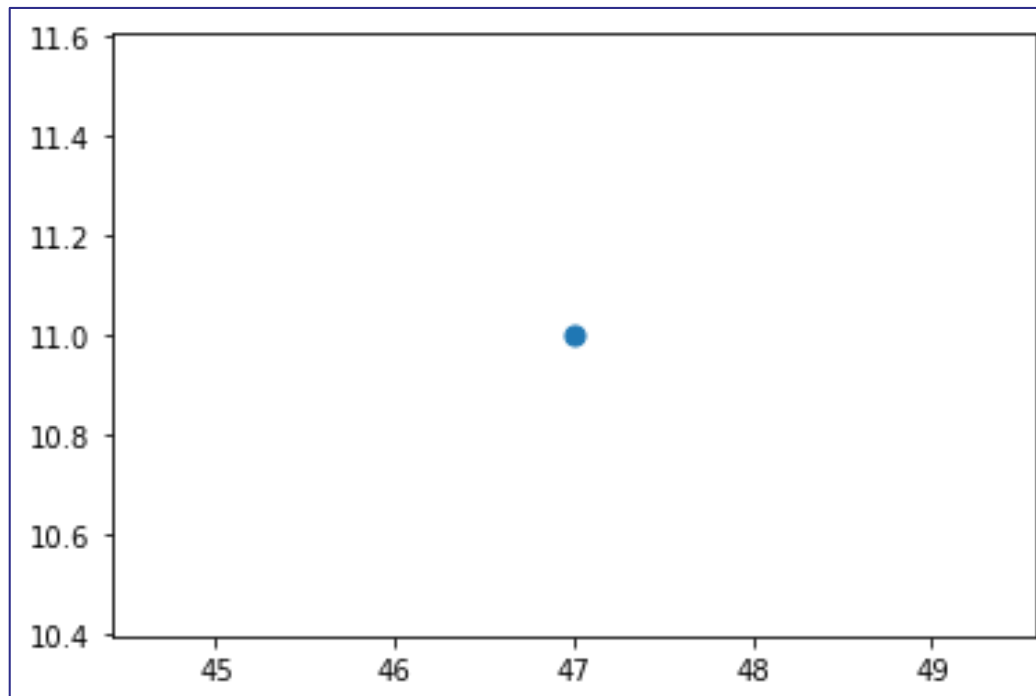
# Streudiagramme I/II

## #Streudiagramme

```
import matplotlib.pyplot as plt
```

```
plt.scatter(47,11, s=50)    #s-Durchmesser des Kreises
```

```
plt.show()
```





# Streudiagramme II/II

## #Streudiagramme

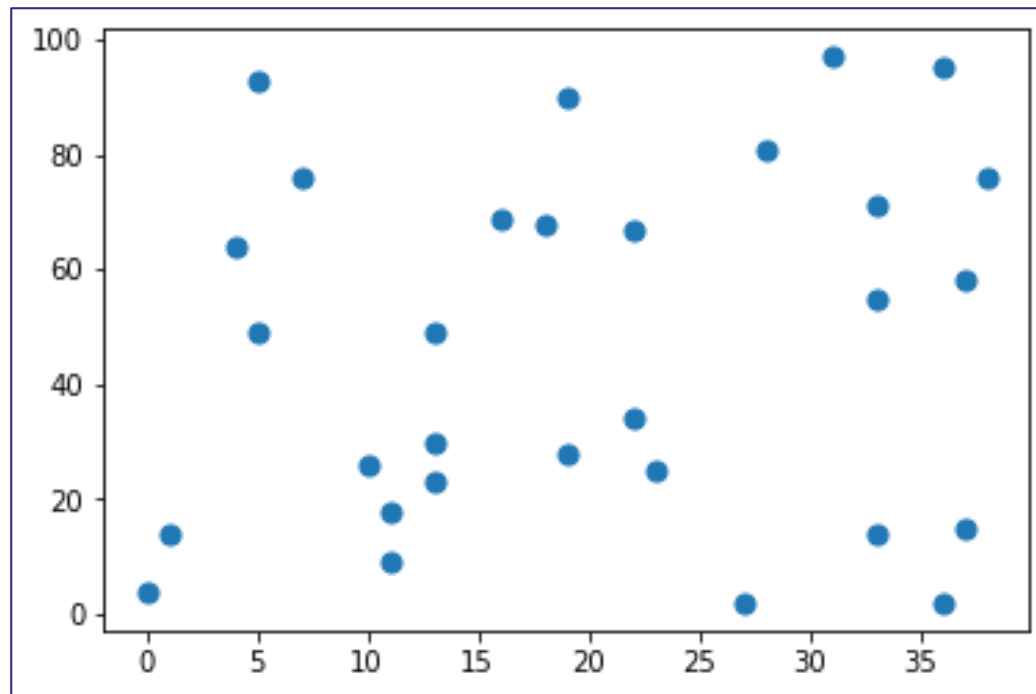
```
import matplotlib.pyplot as plt  
import numpy as np
```

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

```
X=np.random.randint(0, 40, (30,))# 30 Zufallszahlen zw. 0 und 40
```

```
Y=np.random.randint(0,100, (30,))# 30 Zufallszahlen zw. 0 und 100
```

```
ax.scatter(X,Y, s=50)
```



# Legenden einfügen

## #Legenden einfügen

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

```
y1 = np.sin(x)
```

```
y2 = np.cos(x)
```

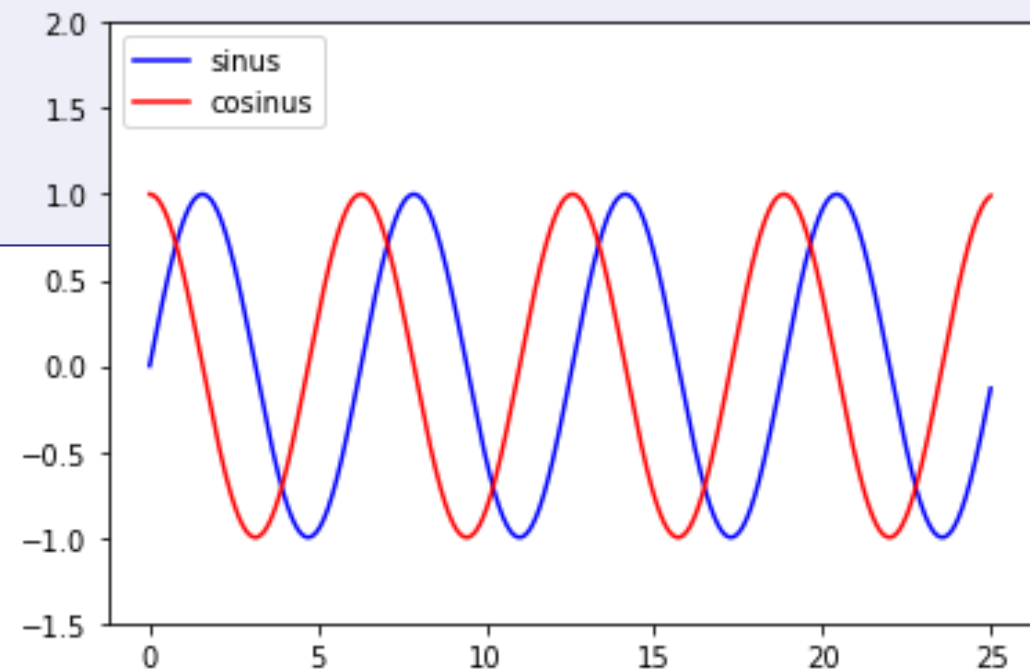
```
plt.plot(x, y1, '-b', label='sinus')
```

```
plt.plot(x, y2, '-r', label='cosinus')
```

```
plt.legend(loc='upper left')
```

```
plt.ylim(-1.5, 2.0)
```

```
plt.show()
```



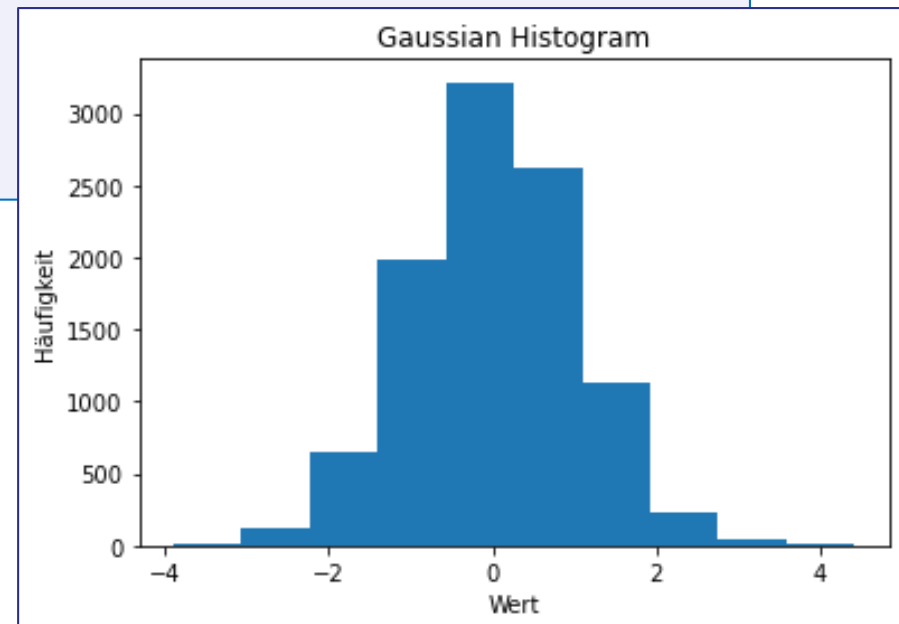
# Histogramme I/II

Def.: Ein Histogramm besitzt auf der x-Achse **gleich große benachbarte Intervallbereiche** (engl. Bins). Die Daten sind entsprechend ihrer Werte auf die Intervalle verteilt. Für jedes Intervall sind Rechtecke gezeichnet, deren **Höhe** (bzw. auch Flächeninhalt) der **Häufigkeit** (Anzahl der Werte) in diesem Intervall entspricht.

```
# Histogramm zeichnen mit matplotlib.pyplot.hist
import matplotlib.pyplot as plt
import numpy as np

gaussian_numbers = np.random.normal(size=10000) #10.000 Zufallszahlen
plt.hist(gaussian_numbers)
plt.title("Gaussian Histogram")
plt.xlabel("Wert")
plt.ylabel("Häufigkeit")
plt.show()
```

Funktion **hist** gibt Tupel mit drei Objekten zurück (**n**, **bins**, **patches**)



# Histogramme II/II

## Rückgabewerte von hist

**n[i]** – Anzahl der Werte innerhalb der Grenzen **bins[i]** und **bins[i+1]**

**bins[i]** – Intervallgrenzen

**patches** – Liste mit Rechtecken und ihren Eigenschaften

```
# Beispiel Histogramm weiter
```

```
n, bins, patches = plt.hist(gaussian_numbers)
```

```
print("Die ersten drei 'bins': ", bins[0:3])
```

**Ausgabe: Die ersten drei 'bins':**[-3.896383 -3.06540235 -2.23442171]

```
print("n: ", n, sum(n))
```

**Ausgabe: n:** [ 8. 114. 653. 1992. 3216. 2616. 1129. 231. 36. 5.] 10000.0

```
print("patches: ", patches)
```

```
for i in range(10):
```

```
    print(patches[i])
```

**Ausgabe:**

patches: <BarContainer object of 10 artists>

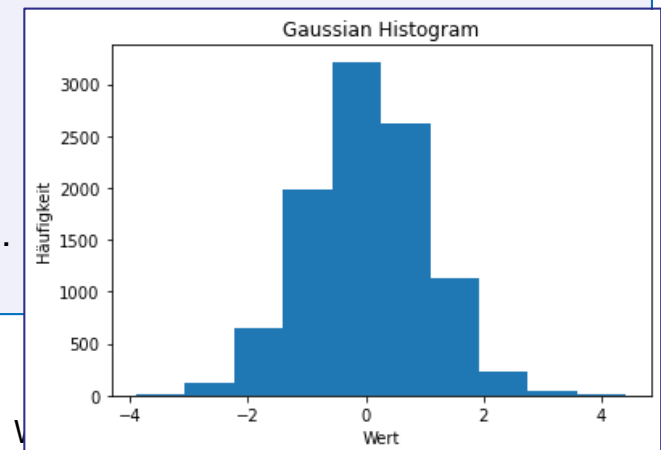
Rectangle(xy=(-3.89638, 0), width=0.830981, height=8, angle=0)

Rectangle(xy=(-3.0654, 0), width=0.830981, height=114, angle=0)

Rectangle(xy=(-2.23442, 0), width=0.830981, height=653, angle=0)

Rectangle(xy=(-1.40344, 0), width=0.830981, height=1992, angle=0) ...

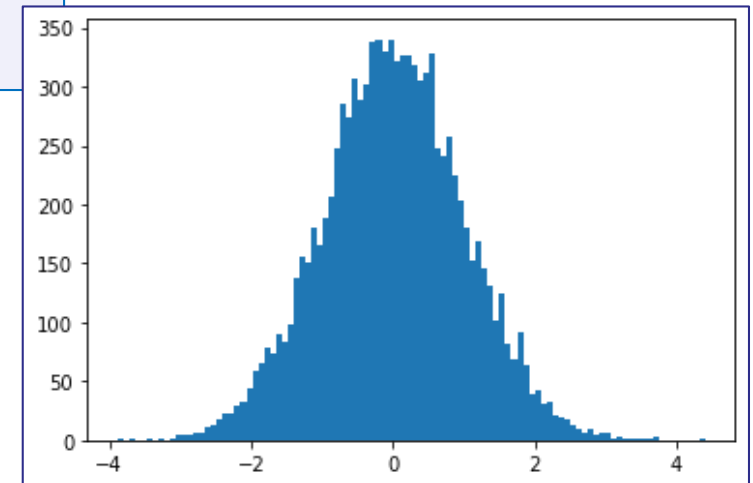
...Rectangle(xy=(3.58244, 0), width=0.830981, height=5, angle=0)



# Histogramme-Parameter

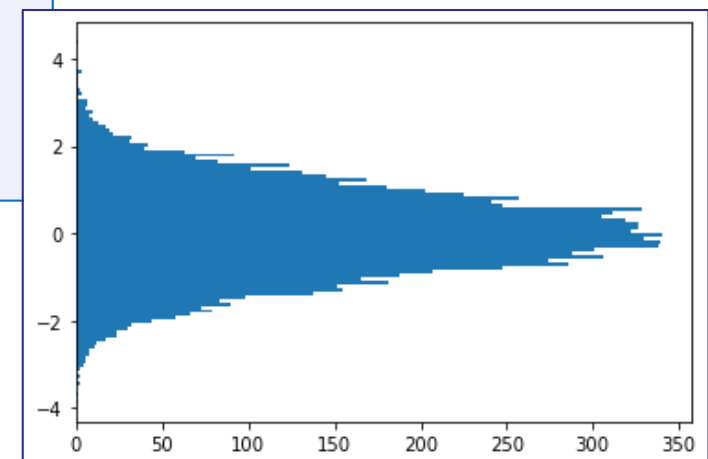
# Beispiel Histogramm weiter, **Anzahl der Bins**

```
plt.hist(gaussian_numbers, bins=100)  
plt.show()
```



# Beispiel Histogramm weiter, **horizontal zeichnen**

```
plt.hist(gaussian_numbers, bins=100,  
orientation="horizontal")  
plt.show()
```



# Säulendiagramm

# Beispiel Säulendiagramm

```
import matplotlib.pyplot as plt
```

```
subjects = ["Gw", "Sport", "Recht", "Ma_Na", "HM", "Agr, VM", "Ing", "Ku",  
"Sonstige"]
```

```
beginners = (88454, 8171, 311267, 91518, 42771, 16065, 217161, 26890, 3406)
```

```
bar_width = 0.9
```

```
fig, ax = plt.subplots() # subplots für Achsenmanipulation und Diagramm extra
```

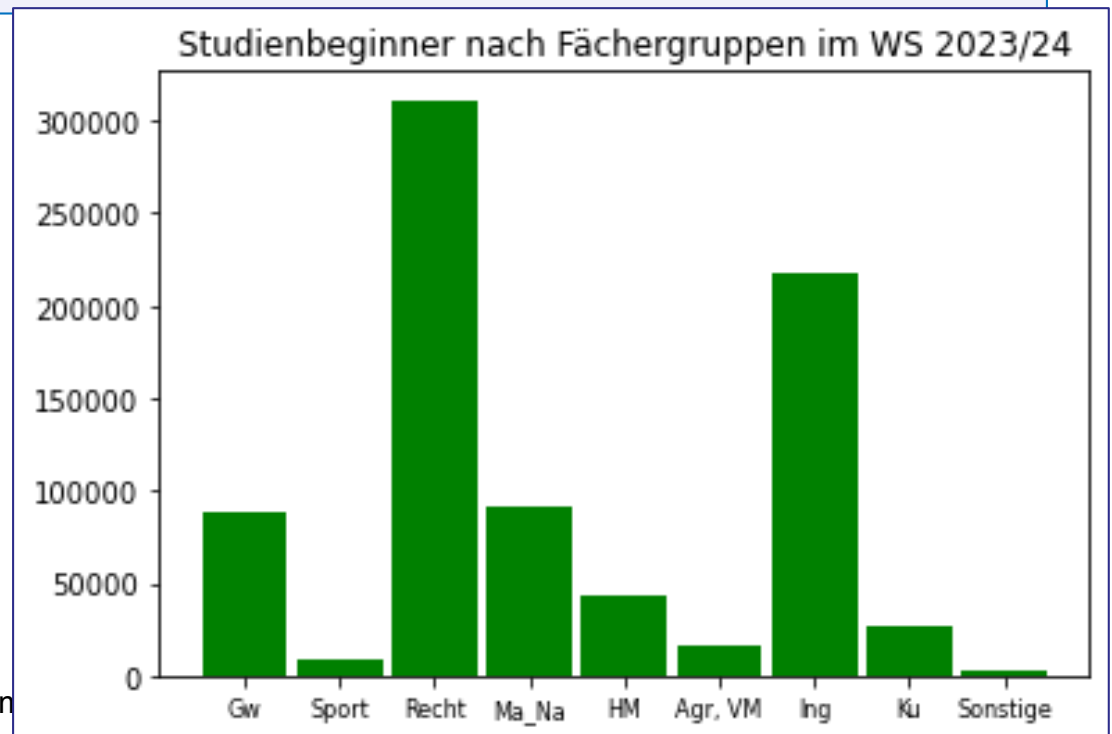
```
plt.title("Studienbeginner nach Fächergruppen im WS 2023/24")
```

```
ax.bar(subjects, beginners, bar_width, color="green")
```

```
ax.tick_params(axis='x', labelsiz=8)
```

```
ax.set_xticks(subjects) #label zentrieren
```

**Def.: Ein Säulendiagramm** setzt sich aus **senkrecht auf der x-Achse stehenden Rechtecken** zusammen, die sich wie Säulen aufrichten. Bei sehr schmalen Rechtecken redet man auch von „**Stabdiagrammen**“. Die Breite der Rechtecke hat keine mathematische Bedeutung.



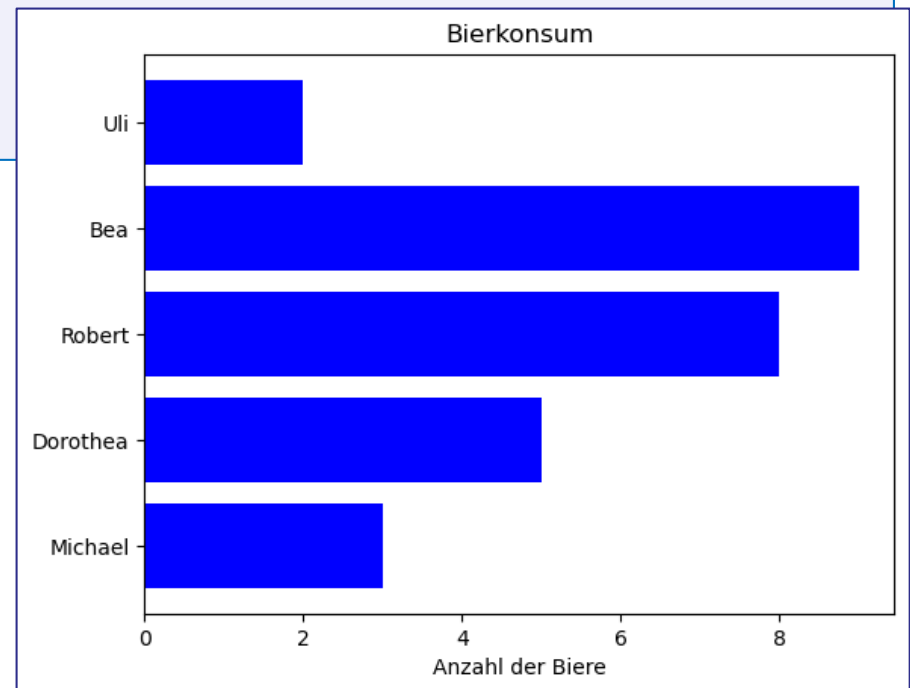
```
# Beispiel Balkendiagramm
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()
studies = ('Michael', 'Dorothea', 'Robert', 'Bea', 'Uli')
y_pos = np.arange(len(studies))
Bierkonsum = (3, 5, 8, 9, 2)

ax.barh(y_pos, Bierkonsum, align='center', color='blue')
ax.set_yticks(y_pos)
ax.set_yticklabels(studies)
ax.invert_yaxis() # labels von oben nach unten
ax.set_xlabel('Anzahl der Biere')
ax.set_title('Bierkonsum')
plt.show()
```

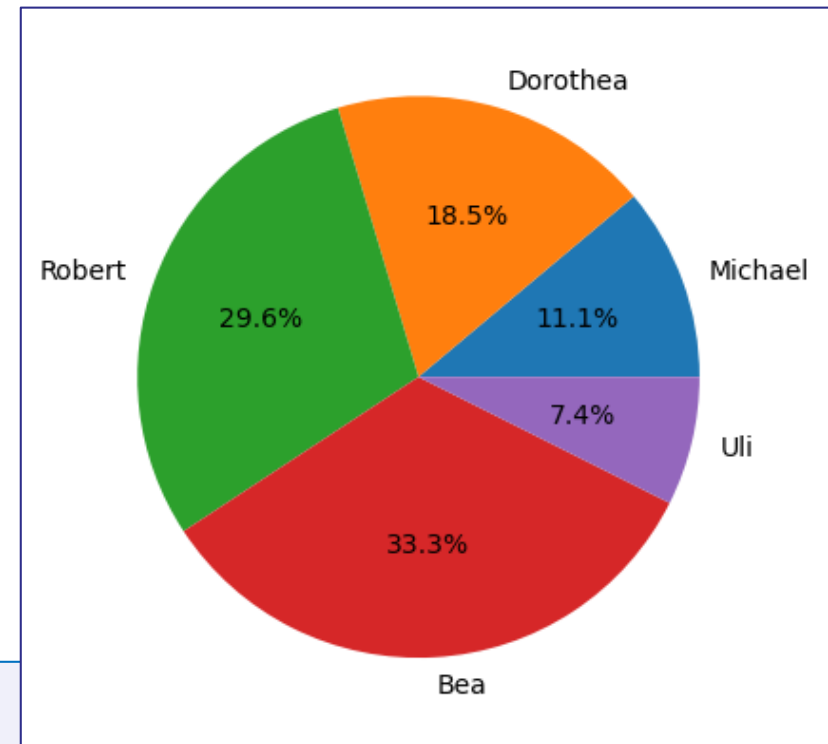
## Def.: Bei einem Balkendiagramm

sind im Vergleich zum  
Säulendiagramm die Rechtecke  
waagerecht orientiert.



# Tortendiagramm

Def.: Ein Tortendiagramm (engl. „pie chart“) ist kreisförmig. Jede Datenkategorie wird als ein Tortenstück dargestellt, dessen Größe proportional zum Wert dieser Datenkategorie ist.



## # Beispiel Tortendiagramm

```
import matplotlib.pyplot as plt
```

```
studies= 'Michael', 'Dorothea', 'Robert', 'Bea', 'Uli'
```

```
Bierkonsum = [3, 5, 8, 9, 2]
```

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

```
ax.pie(Bierkonsum, labels=studies, autopct='%1.1f%%' )
```

```
# Parameter labelt die Tortenstücke mit den Werten : autopct='%1.1f%%'
```



# Datenanalyse mit Python

1. Python für Datenanalyse
2. Datenvisualisierung mit Matplotlib
- 3. Pandas Datenstrukturen**
4. Pandas DataFrame
5. Pandas Dateiverarbeitung
6. Statistische Maße und Analysen
7. Datenvisualisierung mit Pandas
8. Zeit und Datum, Zeitreihen

# Pandas Datenstrukturen



## ➤ Pandas

- ist ein Acronym für „**panel data**“ → Begriff für Datensätze mit und ohne zeitliche Dimension
- kann direkt CSV und Excel einlesen
- ermöglicht Datenanalyse und –manipulation , Datensäuberung, Preprocessing (Vorbereitung )
  - z.B. Auffüllen von fehlenden Werten,
  - Gruppieren von Daten
  - Zusammenführen von Datensätzen
  - Transformation
  - Aggregation von Daten
- Visualisierungsmöglichkeiten
- Alles anwendbar auf sehr großen Datenmengen
- Pandas baut auf NumPy, SciPy und Matplotlib auf und nutzt deren Funktionen
- Wichtigste Datenstrukturen sind **Series** und **DataFrame**

# Series



## ➤ Series

- ist ein **eindimensionales Array-ähnliches Objekt mit einem Index**
- Index und Werte der Series müssen jeweils einen einheitlichen Datentyp aufweisen (Integer, Float, String, etc. )

# Beispiel Initialisierung einer Series

```
import pandas as pd
```

```
S = pd.Series([11, 28, 72, 3, 5, 8])
```

```
print(S) # Ausgabe:
```

|       |   |    |  |
|-------|---|----|--|
|       | 0 | 11 |  |
|       | 1 | 28 |  |
|       | 2 | 72 |  |
| Index | 3 | 3  |  |
|       | 4 | 5  |  |
|       | 5 | 8  |  |

← Werte der Series

Wenn kein Index definiert wird,  
benutzt Pandas einen Default-Index

# Beispiel Series weiter

```
print(S.index)
```

```
print(S.values)
```

**Ausgabe:** RangeIndex(start=0, stop=6, step=1) # default-Index  
[11 28 72 3 5 8]

# Series



## # Beispiel Series mit individuellem Indize, z.B. Strings for Fruits

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)
print(S)
```

### Ausgabe:

```
apples    20
oranges   33
Cherries  52
pears     10
```

## # Addition von Series mit gleichen Indizes

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits)
print(S + S2)
```

### Ausgabe:

```
apples    37
oranges   46
cherries  83
pears     42
```

## # Addition von Series mit ungleichen Indizes

```
fruits = ['peaches', 'oranges', 'cherries', 'pears']
fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits2)
print(S + S2)
```

### Ausgabe:

```
cherries    83.0
oranges     46.0
peaches      NaN
pears       42.0
raspberries  NaN
```

Der **resultierende Index** ist eine „**Vereinigungsmenge**“ beider Indizes. Wenn ein Index nicht in beiden Series vorkommt, wird der entsprechende Wert auf **NaN** gesetzt.

# Indizierung von Series



**# Zugriff auf einzelne Werte in Series , vorheriges Beispiel weiter**

```
print(S['apples'])
```

**Ausgabe:**

20

**# Zugriff auf mehrere Werte gleichzeitig in Series durch Übergabe eines  
# Array-ähnlichen Objekts**

```
print(S[['apples', 'oranges', 'cherries']])
```

**Ausgabe:**

```
apples    37
oranges   46
cherries  83
```

**# Filterung mit Booleschem Array**

```
S[S>30]
```

**Ausgabe:**

```
oranges  33
cherries 52
```

**# Operationen mit Skalaren oder die Anwendung von mathematischen  
# Funktionen**

```
import numpy as np
print((S + 3) * 4)
print("=====")
print(np.sin(S))
```

**Ausgabe:**

```
apples    92
oranges   144
cherries  220
pears     52
=====
apples    0.912945
oranges   0.999912
cherries  0.986628
pears    -0.544021
```

# Pandas.Series.apply- Funktion



## ➤ Series.apply

- Wendet eine Funktion **func** auf das Series-Objekt an und liefert ein Series- oder DataFrame-Objekt zurück.

**# Series.Apply-Funktion anwenden, vorheriges Beispiel weiter**

```
print(S.apply(np.log)) # Natürlicher Logarithmus
```

**Ausgabe:**

|          |          |
|----------|----------|
| apples   | 2.995732 |
| oranges  | 3.496508 |
| cherries | 3.951244 |
| pears    | 2.302585 |

**# Series.Apply-Funktion anwenden, Beispiel mit Lambda-Funktion**

```
# war: ([20, 33, 52, 10])
```

```
print(S.apply(lambda x: x if x > 50 else x + 10))
```

**Ausgabe:**

|          |    |
|----------|----|
| apples   | 30 |
| Oranges  | 43 |
| Cherries | 52 |
| pears    | 20 |

**#dictionary übergeben und in Series umwandeln mit Schlüssel als Index**

```
cities = {"London": 8615246, "Berlin": 3562166,  
"Madrid": 3165235, "Rome": 2874038, "Paris": 2273305,  
"Vienna": 1805681, "Bucharest": 1803425, "Hamburg": 1760433,  
"Budapest": 1754000, "Warsaw": 1740119, "Barcelona": 1602386,  
"Munich": 1493900, "Milan": 1350680}  
city_series = pd.Series(cities)  
print(city_series)
```

**Ausgabe:**

|        |         |
|--------|---------|
| London | 8615246 |
| Berlin | 3562166 |
| Madrid | 3165235 |
| ...    |         |
| Milan  | 1350680 |

# Wenn Daten fehlen NaN



- Ein häufiges Problem bei Messwerten ist, dass Daten fehlen.

Warum? Beispiele?

**# vorheriges Beispiel weiter, Index ist nicht vollständig -> NaN**

```
my_cities = ["London", "Paris", "Zurich", "Berlin",  
            "Stuttgart", "Hamburg"] // Array das für den Index übergeben wird, ist  
                                   // nicht vollständig in cities-dictionary  
my_city_series = pd.Series(cities, index=my_cities)  
print(my_city_series)
```

**Ausgabe:**

|           |           |
|-----------|-----------|
| London    | 8615246.0 |
| Paris     | 2273305.0 |
| Zurich    | NaN       |
| Berlin    | 3562166.0 |
| Stuttgart | NaN       |
| Hamburg   | 1760433.0 |

*Städten, die im Dictionary fehlen, aber im zugewiesenen Index sind, wird der Wert NaN zugewiesen.*

- Methoden `isnull()` und `notnull()` prüfen auf fehlende Werte.

**# vorheriges Beispiel weiter, Series auf fehlende Werte prüfen**

```
my_cities = ["London", "Paris", "Zurich", "Berlin",  
            "Stuttgart", "Hamburg"]  
my_city_series = pd.Series(cities, index=my_cities)  
print(my_city_series.isnull())
```

**Ausgabe:**

|           |       |
|-----------|-------|
| London    | False |
| Paris     | False |
| Zurich    | True  |
| Berlin    | False |
| Stuttgart | True  |
| Hamburg   | False |

# Fehlende Daten heraus filtern



- Mit der Methode `dropna()` können fehlende Werte aus einem Series-Objekt herausgefiltert werden.

**# vorheriges Beispiel weiter, NaN heraus filtern**

```
print("Vorher:\n")
print(my_city_series)
print("\nNachher:\n")
print(my_city_series.dropna())
```

## Ausgabe:

Vorher:

|           |           |
|-----------|-----------|
| London    | 8615246.0 |
| Paris     | 2273305.0 |
| Zurich    | NaN       |
| Berlin    | 3562166.0 |
| Stuttgart | NaN       |
| Hamburg   | 1760433.0 |

Nachher:

|         |           |
|---------|-----------|
| London  | 8615246.0 |
| Paris   | 2273305.0 |
| Berlin  | 3562166.0 |
| Hamburg | 1760433.0 |



# Fehlende Daten auffüllen



- Mit der Methode `fillna()` können fehlende Werte in einem Series-Objekt aufgefüllt werden.

**# vorheriges Beispiel weiter, NaN mit 0 auffüllen**

```
print(my_city_series.fillna(0))
```

**Ausgabe:**

|                  |            |
|------------------|------------|
| London           | 8615246.0  |
| Paris            | 2273305.0  |
| <b>Zurich</b>    | <b>0.0</b> |
| Berlin           | 3562166.0  |
| <b>Stuttgart</b> | <b>0.0</b> |
| Hamburg          | 1760433.0  |

**# vorheriges Beispiel weiter, NaN mit neuen Werten aus Dictionary auffüllen**

```
missing_cities = {"Stuttgart": 597939, "Zurich": 378884}
my_city_series.fillna(missing_cities, inplace=True) #parameter „inplace“=True für Einfügen
print(my_city_series)
```

**Ausgabe:**

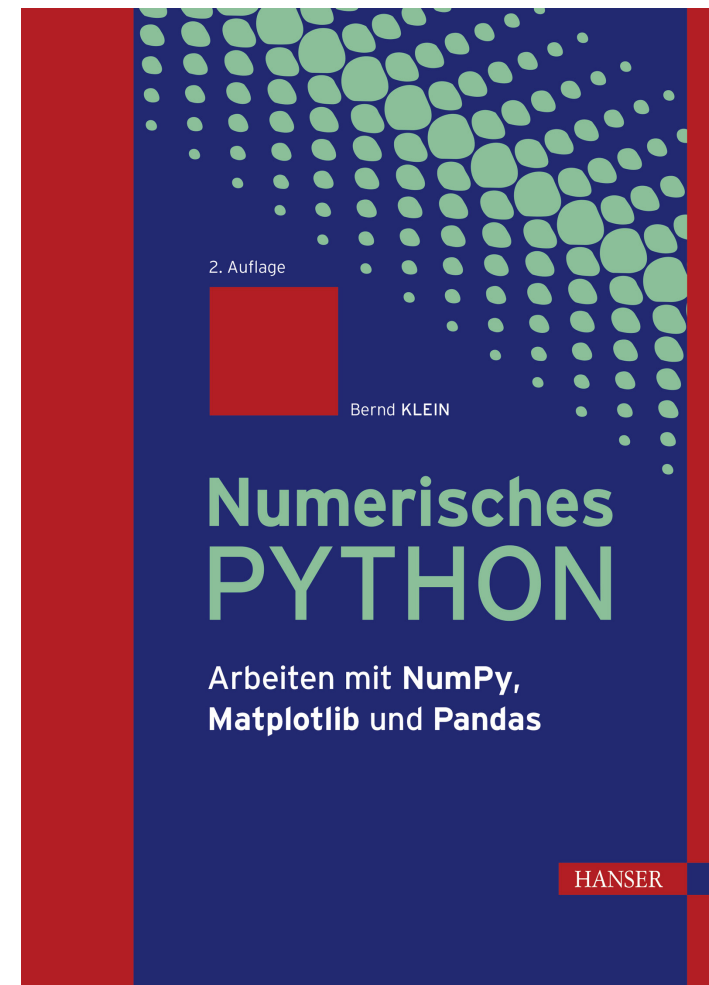
|                  |               |
|------------------|---------------|
| London           | 8615246.0     |
| Paris            | 2273305.0     |
| <b>Zurich</b>    | <b>378884</b> |
| Berlin           | 3562166.0     |
| <b>Stuttgart</b> | <b>597939</b> |
| Hamburg          | 1760433.0     |

**# alle werte in Integer umwandeln (Bevölkerungszahl)**

```
my_city_series = my_city_series.astype(int)
print(my_city_series) # dann Ausgabe in Integer
                      (ohne Bild)
```

# Literaturangaben und Quellen

KLEIN, Bernd, 2023. *Numerisches Python: Arbeiten mit NumPy, Matplotlib und Pandas*. 2., aktualisierte und erweiterte Auflage. München: Hanser. ISBN 3446471707



- Bis hierhin Blatt 1 Praktikum

# Ende VL 2, Blatt 1