Wirtschaftsinformatik 2

Einstieg in die Datenanalyse mit Pandas

Prof. Dr. Christian Bockermann, Prof. Dr. Volker Klingspor

Hochschule Bochum

Sommersemester 2026

Vorbemerkungen zum Block Datenanalyse mit Pandas

Pandas ist ein umfangreiches Python-Modul zur Datenanalyse. Zentral dafür sind die Datentypen *Series* und *DataFrame* als Darstellung für Daten.

Die Vorlesung fokussiert sich auf die wesentlichen Konzepte, wobei der Fokus zunächst auf *Series* liegt. Zu Beginn kann es sein, dass einige Stellen nicht sofort verständlich sind (z.B. `read_csv`), das wird sich aber dann im nächsten Block aufklären.

Das Buch *Datenanalyse mit Pandas* ist ein detailiertes Nachschlagewerk, das an einigen Stellen sehr stark in die Tiefe geht (z.B. `numpy` Arrays in Kap. 4). Mit diesem Foliensatz zum Selbststudium beschränken wir uns auf den Teil, der für einen ersten Einstieg in die Datenanalyse (und diese Vorlesung) wichtig ist.

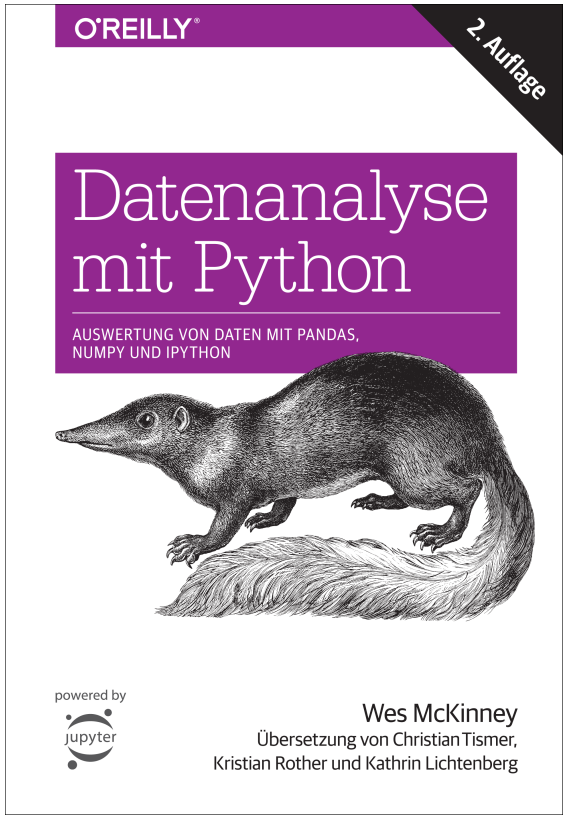
Das Pandas Modul

- Pandas ist ein Python-Modul für die Datenanalyse
- Ursprünglich entwickelt von Wes McKinney
- Buch: **Datenanalyse mit Python** (Wes McKinney)

Weitere Dokumentation:

`https://pandas.pydata.org`

- Als Online-Buch in Hochschul Bib erhältlich!!



Lernblock 4 behandelt Teile von Kapitel 5.1 (Series, bis Seite 132)

pandas ist ein zentrales Modul für die Datenverarbeitung in Python

- Datentypen für Tabellen (`DataFrame`) und Zeitreihen (`Series`)
- Funktionen zum Lesen/Transformieren von Daten
- Unterstützung einer Vielzahl von Formaten: CSV, Excel, Datenbanken, usw.

```python
import pandas as pd

# Lesen von Excel-Daten
df = pd.read_excel('meine_Daten.xlsx')
```

Einbinden des Moduls in ein Python Program

```
import pandas as pd

# optional dazu noch:
from pandas import Series, DataFrame
```

- Importiert Pandas als Modul **pd**
- **pd** ist eine allgemein verbreitete Abkürzung für Pandas
- Häufig werden **Series** und **DataFrame** noch direkt importiert um sich das `pd.` davor zu sparen (optional)

Pandas bietet Datentypen und Funktionen

- `pandas.read_csv`
- `pandas.read_excel`
- `pandas.to_csv`
- `Series.plot`
- `Series + Series`
- `DataFrame.plot`
- `DataFrame + DataFrame`
- `...`

Funktionen

0	**4**
1	**5**
2	**8**
3	**2**

Series

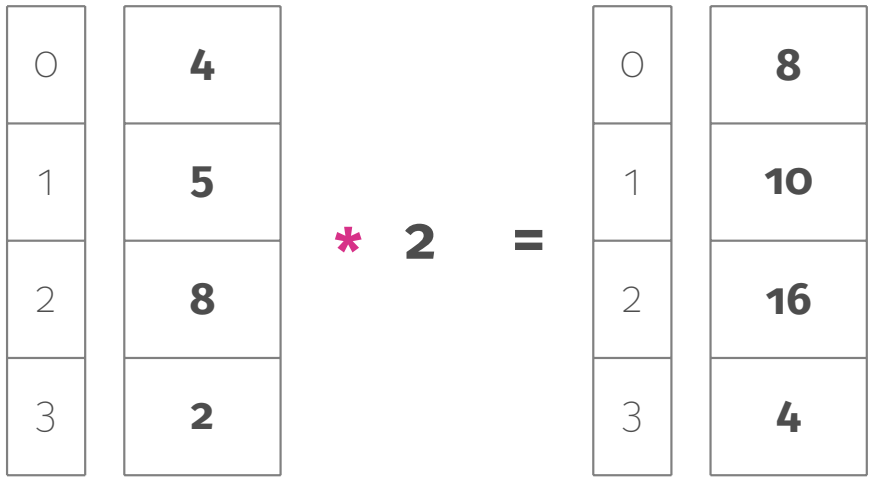
	a1	a2	a3	a4
0	**4**	**1**	**2**	**9**
1	**5**	**1**	**3**	**6**
2	**3**	**8**	**7**	**4**

DataFrame

Die rechte Seite zeigt die Darstellungen von Series und DataFrame Objekten, wie sie in den folgenden Folien stets graphisch dargestellt werden.

Pandas Datentypen unterstützen Rechenoperationen

Zum Beispiel:

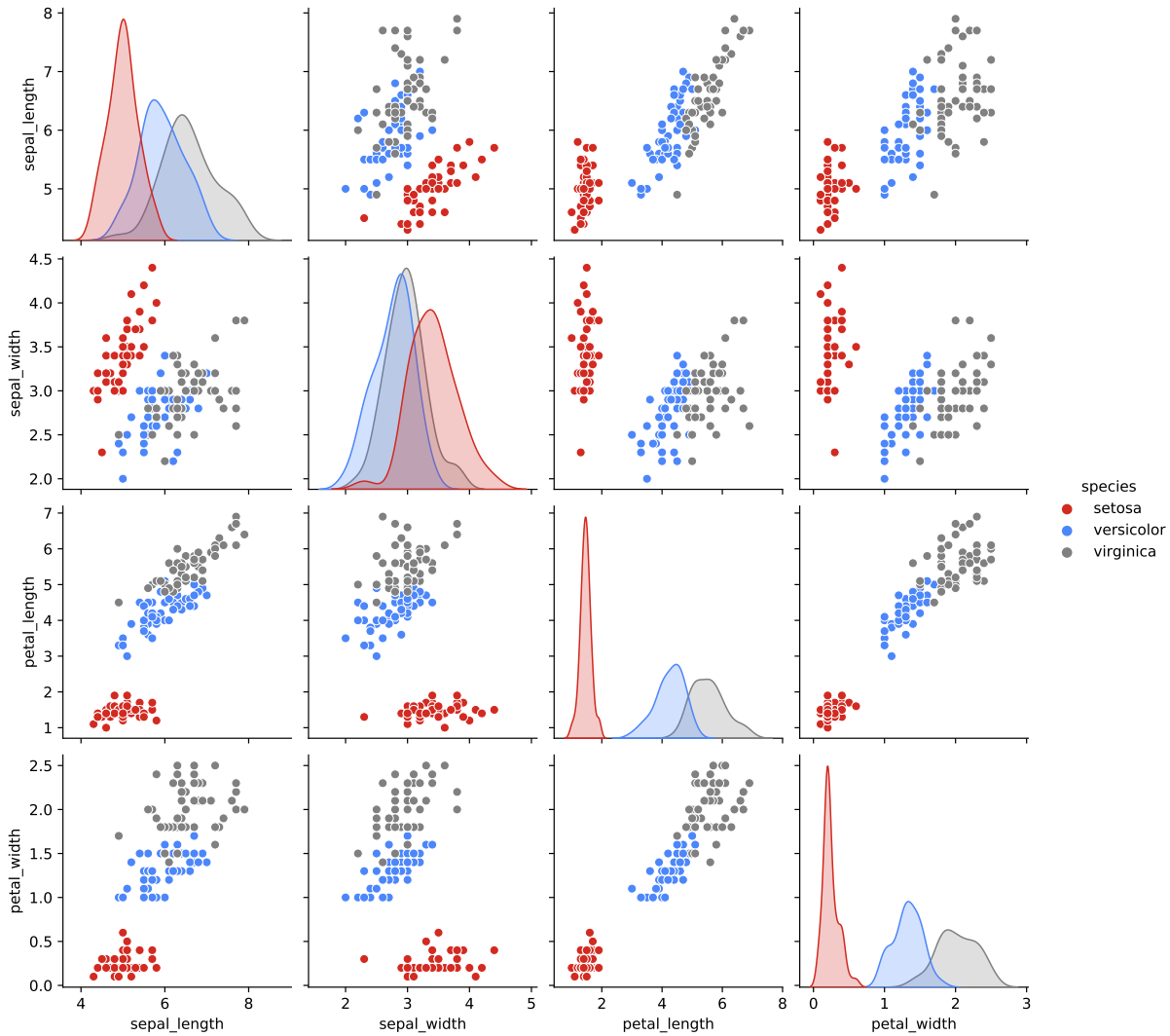



```
a = Series([4,5,8,2])
b = a * 2
```

Kompatibilität zu anderen Modulen

- Pandas baut auf *NumPy* Strukturen auf
- Das Modul *SciKit-Learn* enthält viele Algorithmen für maschinelles Lernen mit Pandas Datentypen
- Mit dem Modul *Seaborn* können Pandas Objekte grafisch geplottet werden

Pandas + Seaborn



Datenanalyse mit Pandas

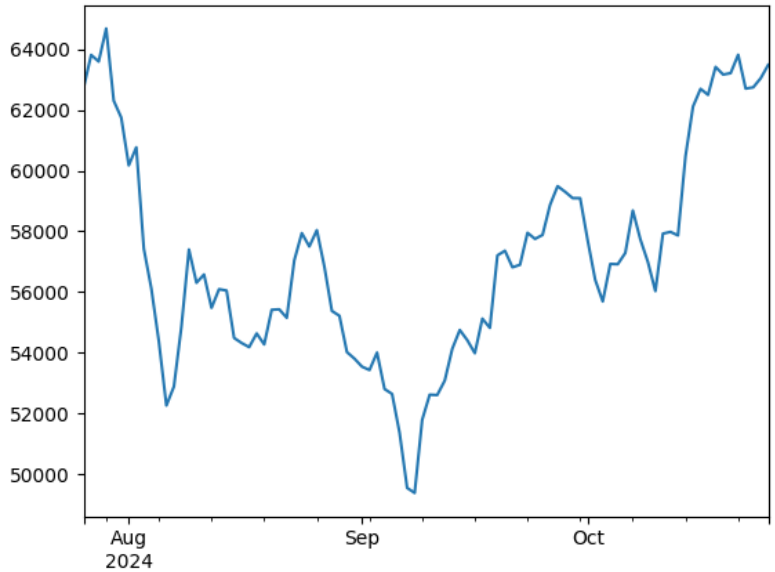
Series - Ein Datentyp für Messreihen

Datentyp: Series

Für die Analyse von Geschäftsprozessen oder betriebswirtschaftlichen Daten werden häufig Messreihen untersucht. Das können beispielsweise Börsenkurse, Preisdaten oder Bestellumsätze in einem Online-Shop sein.

Die folgende Abbildung zeigt den Kursverlauf der Kryptowährung BitCoin über die letzten drei Monate und die zugrunde liegenden Daten:

Datum	Höchstpreis
2024-10-25	63490.0
2024-10-24	63050.0
2024-10-23	62748.0
2024-10-22	62707.0
2024-10-21	63823.0



Um derartige Daten zu erfassen, darzustellen und zu analysieren bietet das Pandas Modul den Datentyp `Series` an, den wir im Folgenden betrachten.

Datentyp für Reihe von Meßwerten: **`Series`**

Index	**Werte**
0	4
1	5
2	8
3	2

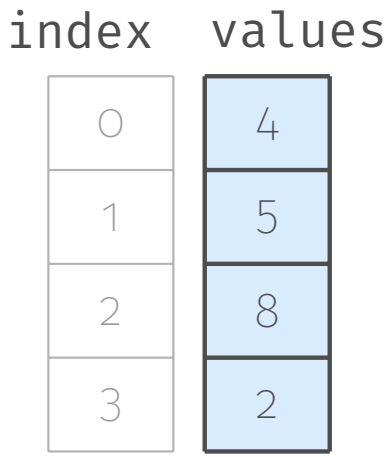
Ein `Series` Objekt lässt sich aus einer Liste von Werten erzeugen:

```
series1 = Series([4,5,8,2])
```

Datentyp: Series

Ein Objekt vom Typ `Series` besteht aus einem **`index`** und einem **`values`** Teil. Der **`values`** Teil ist wie eine Liste der Werte einer Series:

```
series1 = Series([4,5,8,2])
```



Zugriff auf die Werte 4, 5, 8,.. der Variable **`series1`**:

```
werte = series1.values    # quasi eine Liste [4, 5,..]
index = series1.index     # entspricht etwa [0, 1, ..]
```

Werte und Index von Series

Mit Schleifen kann auf die beiden Teiles eines `Series`-Objektes zugegriffen werden, z.B. durch:

```
series1 = Series([4, 5, 8, 2])

for wert in series1.values:
    print(wert)

for position in series1.index:
    print(position)
```

Probieren Sie es im Notebook aus!

Series Objekt im Jupyter Notebook:

Pandas-Series.ipynb

Code Python 3

```
[1]: from pandas import Series

[2]: s1 = Series([4,5,8,2], name="S1")

[3]: print(s1)
```

```
0    4
1    5
2    8
3    2
Name: S1, dtype: int64
```

(Hier mit optionalem Parameter **name**.)

Series Objekte haben häufig Werte eines Typs

Aber auch Series mit unterschiedlichen Werte-Typen sind möglich:

```
s1 = Series([4,5,8,2])
print(s1.dtypes)   # ergibt: int64

s2 = Series([1,2,'a','b'])
print(s2.dtypes)   # ergibt: object
```

Bisher haben wir die Datentypen `int`, `float` und `str` kennengelernt. Pandas benutzt leicht andere Typen für Werte, die aber ähnlich sind. `int64` entspricht `int` und `float64` dem bekannten `float`. Für andere allgemeine Dinge (z.B. Strings) verwendet Pandas den Datentyp `object`.

Zugriff auf Werte einer Series

Der Zugriff auf Elemente aus einer `Series` erfolgt ähnlich wie bei Listen über den Index:

```
s1 = Series([4,5,8,2])
x = s1[2]
print(x)

s1[2] = 7     # Veraendern von Werten
```

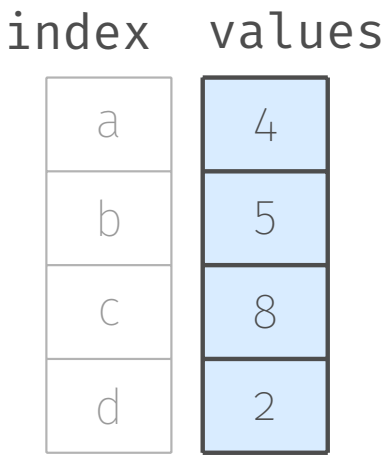
Wie siehen die Werte von `s1` aus, wenn der obige Code ausgeführt wurde?

Probieren Sie es im Notebook aus!

Series erlaubt eigene Index-Werte

Mit dem normalen Index kann auf die einzelnen Werte einer Series wie bei einer Liste zugegriffen werden. Es lassen sich aber auch Series-Objekte mit einem anderen Index erzeugen:

```
reihe = Series([4,5,8,2], index=['a', 'b', 'c', 'd'])
```




```
x = reihe['b']    # x ist jetzt 5
```

Index ändern

Der Index eines Series-Objektes kann auch nachträglich verändert werden, z.B.:

```
meineSeries = Series([9, 4, 2, 1, 7])

meineSeries.index = [10, 20, 30, 40, 50]
```

Wie sieht das Series-Objekt **meineSeries** aus,
wenn der obige Code ausgeführt wurde?

◀ Probieren Sie es im Notebook aus!

Index ändern

Was passiert bei dem folgenden Code? Und warum?

```
meineSeries = Series([9, 4, 2, 1, 7])

meineSeries.index = [1, 2, 3]
```

Probieren Sie es im Notebook aus!

Slicing von Series Objekten

Wie bei Listen unterstützt `Series` das sogenannte *Slicing* um auf Teile einer Series zuzugreifen - wie im folgenden Beispiel:

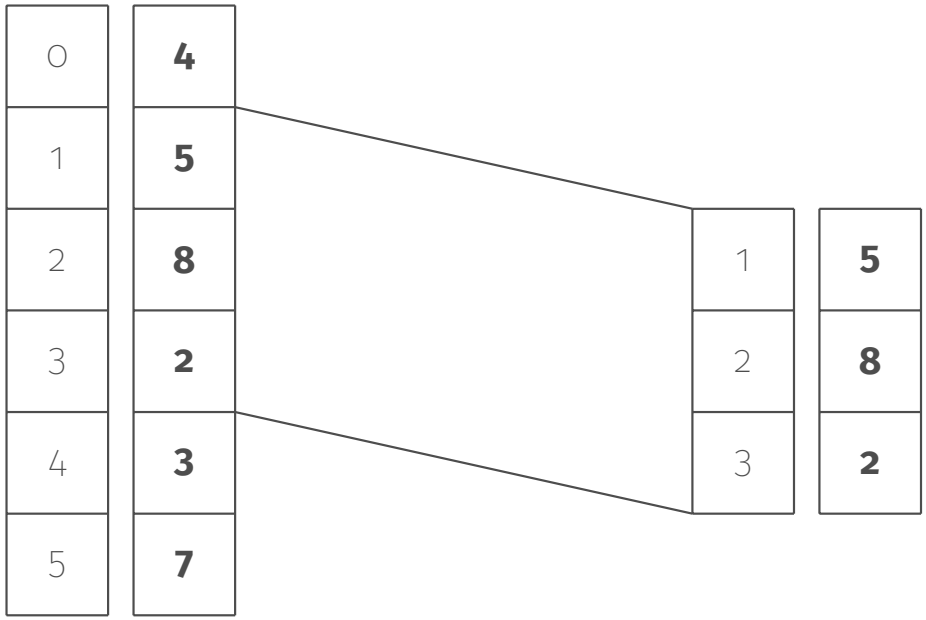
```
s1 = Series([4,5,8,2])

s2 = s1[2:4]
# s2 ist nun ein Teil von s1
```

Welche Index-Werte hat die neue Teil Serie?

Probieren Sie es im Notebook aus!

Was passiert beim Slicing?




```
s1 = Series([...])
```

```
s2 = s1[1:4]
```

Beim Slicing wird ein sogenannter View erzeugt, d.h. die Daten werden nicht in eine neue Series kopiert, sondern die neue Series **`s2`** "zeigt" auf den Teilbereich der bereits bestehenden Series **`s1`**.

Was passiert beim Verändern?

```
s1 = Series([4,5,8,2,3,7])

s2 = s1[1:4]
s2[2] = -1
```

- Wie wirkt sich die Zuweisung auf s1 aus?
- Was ist der Wert von **`s2[0]`** ?
- Welche Index-Werte hat **`s2`** ?

◀ Probieren Sie es im Notebook aus!

Rechnen mit Series Objekten

Einfache Statistiken mit Series

Mit den bisherigen Python-Kenntnissen lassen sich über `Series` Messreihen nun einfache Statistiken berechnen. Zum Beispiel der Durchschnitt:

```
s = Series([ 4, 5, 8, 2])

summe = 0
count = 0
for wert in s.values:
  summe = summe + wert
  count = count + 1

durchschnitt = summe / count
```

Genau für Statistiken und Datenanalyse wurde Pandas allerdings entwickelt und so haben `Series` Objekte diese Dinge schon sehr einfach integriert:

```
durchschnitt = s.mean()  # Durchschnitt berechnen
```

Einfache Statistiken mit Series

Neben `.mean()` gibt es natürlich noch eine Reihe weiterer eingebauter Statistik-Funktionen in `Series` Objekten:

```
s = Series( [4, 5, 8, 2] )

s.min()
s.max()
s.median()
...
```

Sie bekommen detailierte Informationen über die einzelnen Statistik-Funktionen über die Pandas Dokumentation[1] oder, in dem Sie `help(..)` aufrufen:

```
help(Series.min)
```

[1] https://pandas.pydata.org/docs/reference/series.html#computations-descriptive-stats

`Series` unterstützt eine Reihe von Operationen

Mit Series-Objekten lassen sich die Grundrechenarten durchführen (Addition, Multiplikation, usw.). Als Ergebnis kommt in der Regel wieder ein `Series` Objekt heraus.

```
a = Series([4,5,8,2])
b = Series([1,3,0,2])

s = a + b     # ergibt s = [5, 8, 8 , 4]
```

Operationen auf `Series` nutzen Index für "Join"

Was passiert im folgenden Fall?

0	**4**
1	**5**
2	**8**
3	**2**

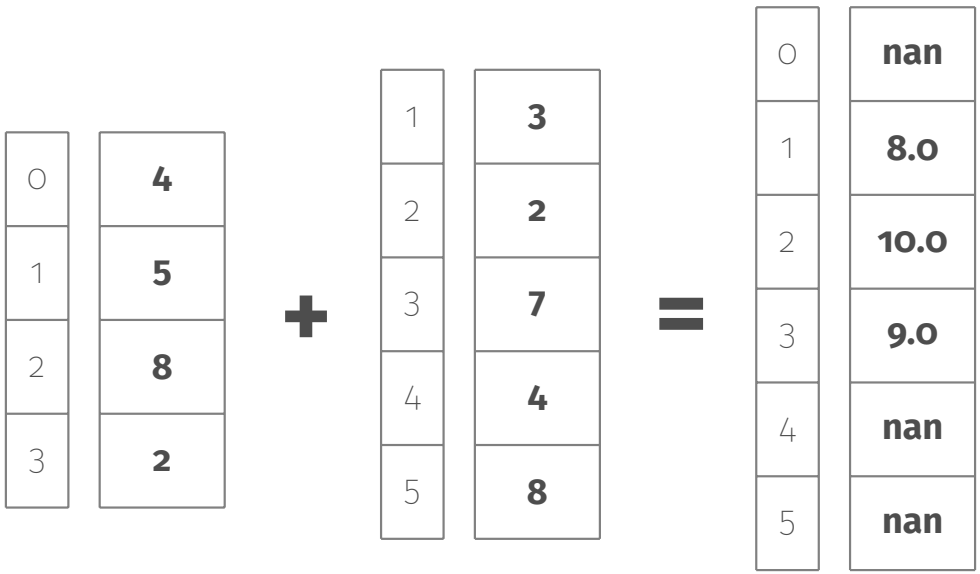
+

1	**3**
2	**2**
3	**7**
4	**4**
5	**8**

= ?

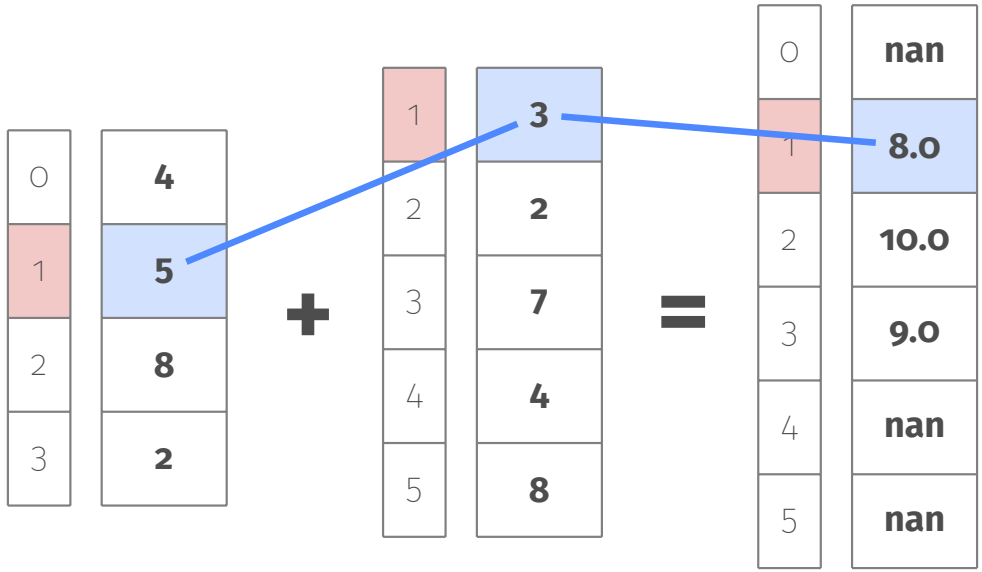
Operationen auf `Series` nutzen Index für "Join"

Was passiert im folgenden Fall?



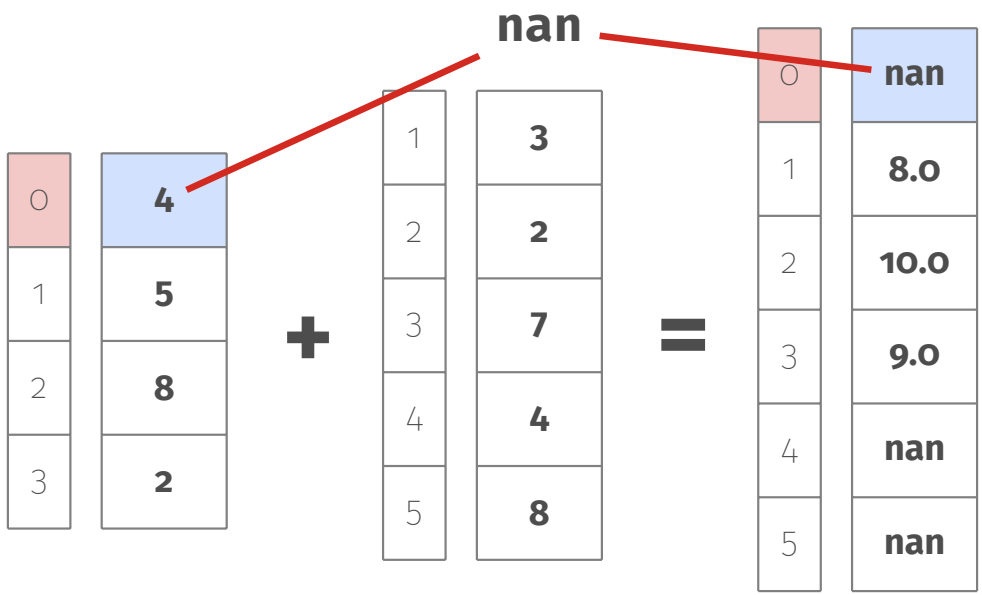
Operationen auf `Series` nutzen Index für "Join"

Was passiert im folgenden Fall?



Operationen auf `Series` nutzen Index für "Join"

Was passiert im folgenden Fall?



Fehlende Werte werden als
nan (not-a-number) behandelt!

Skalare Operationen

Auch Skalare lassen sich mit Series-Objekten mit den Grundrechenarten verknüpfen. Zum Beispiel über die Multiplikation oder die Addition:

```
s1 = Series( [4, 5, 8, 2])

s2 = 3 * s1
s3 = s1 + 17
```

Welche Werte haben `s2` und `s3`?

Probieren Sie es im Notebook aus!

Filtern von Series-Daten

Filtern von `Series` Objekten

Die Vergleichsoperator `>` beispielsweise kann auf alle Elemente einer Series angewendet werden. Dabei wird jedes Element verglichen und liefert entweder `True` oder `False`:

```
a = Series([4,5,8,2])

c = a > 4
```

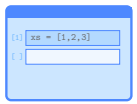
Filtern von `Series` Objekten

Die Vergleichsoperator `>` beispielsweise kann auf alle Elemente einer Series angewendet werden. Dabei wird jedes Element verglichen und liefert entweder `True` oder `False`:

```
a = Series([4,5,8,2])

c = a > 4
```

Wie sieht das Ergebnis **c** aus?



◀ Probieren Sie es im Notebook aus!

Filtern funktioniert mit `True`/`False` Sequenzen

Wir können mit einer Liste von `True`/`False` Werten auswählen, welche Elemente eines Series-Objektes wir auswählen oder abwählen wollen:

```
s = Series([4,5,8])

anfang = s[ [True,True,False] ]
```

Mit der **<** Operation passiert quasi das Gleiche:

```
filter = s < 8    # filter ist ~ [True,True,False]

anfang = s[filter]    # Auswaehlen
```

Probieren Sie es im Notebook aus!

Einfache Filter-Syntax

Wenn man die beiden letzten Folien betrachtet, haben wir dort zunächst einen Filter gebaut (Liste aus `True`/`False` Werten) und diesen Filter dann ähnlich wie beim Slicing in die Series eingesetzt.

Das funktioniert auch in einem Schritt:

```
s = Series([ 4, 5, 8, 2, 9, 3, 1, 7])

gefiltert = s[ s < 5 ]
```

Welche Werte erwarten Sie in `gefiltert`?

◀ Probieren Sie es im Notebook aus!

Einfache Plots mit Series

Series-Daten Plotten

Daten in Series Objekten lassen sich auf einfache Weise sehr schnell visualisieren. Dazu bietet `Series` die **`plot`** Methode:

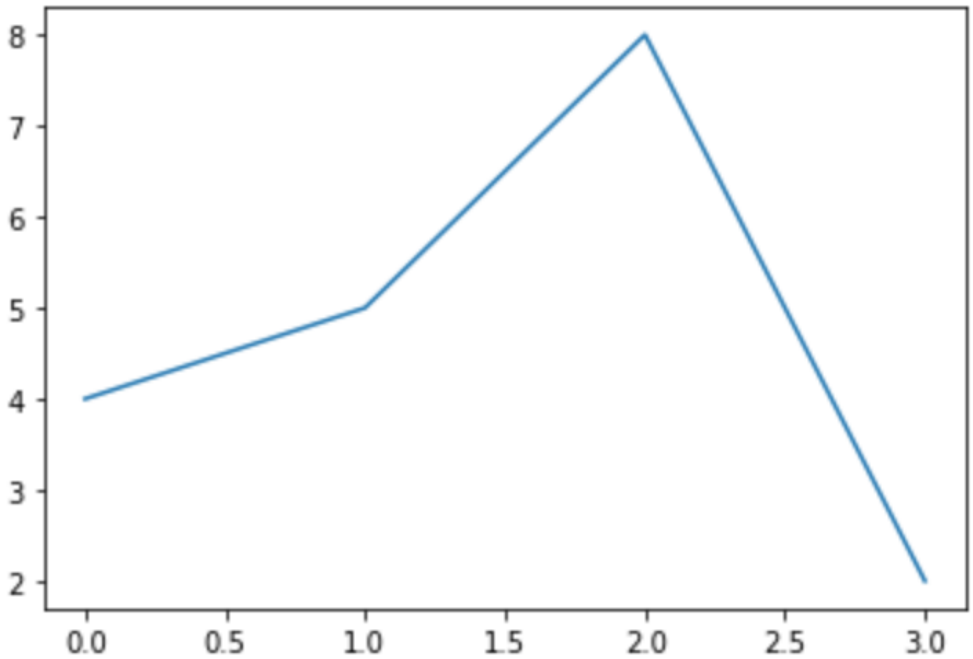
```
s1 = Series([4,5,8,2])
s1.plot()
```

Die Plot-Methode hat eine Vielzahl von Parameter-Möglichkeiten um die Plots entsprechend anzupassen[2]. Natürlich läßt sich das sehr leicht mit all den vorangegangenen Funktionen und Operationen verknüpfen.

[2] `https://pandas.pydata.org/docs/reference/series.html#plotting`

```
s1 = Series([4,5,8,2])
s1.plot()
```

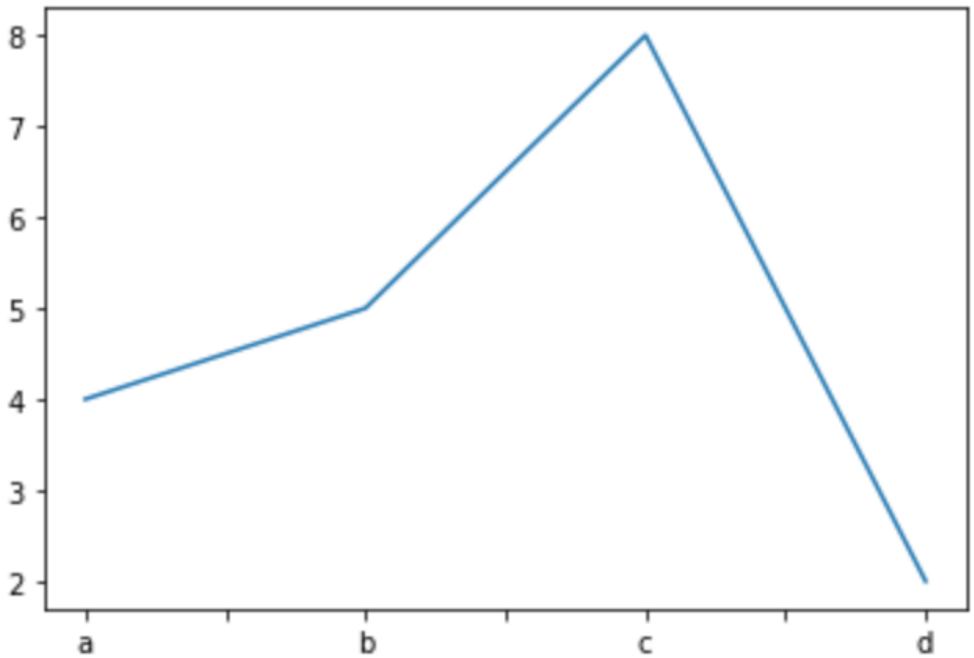
Out[3]: <AxesSubplot:>



Auch `Series` Objekte mit einem eigenen Index lassen sich plotten:

```
s1 = Series([4,5,8,2], index=['a', 'b', 'c', 'd'])
s1.plot()
```

```
Out[5]: <AxesSubplot:>
```



Exkursion - Daten einlesen

Exkursion

Als letzten Punkt greifen wir in diesem Skript ein wenig vor. Das Einlesen von Daten erfolgt bei Pandas mit der Funktion `read_csv`. Damit können Daten in eine Tabelle eingelesen werden – allerdings erzeugt das ein `DataFrame` Objekt und keine `Series`.

Das werden wir in der kommenden Woche genauer erläutern. Für den Moment reicht es aber aus zu wissen, dass mit dem folgenden Code-Fragement eine CSV-Datei eingelesen werden kann. Das Ergebnis ist jedoch kein einfaches `Series` Objekt, sondern eine komplette Tabelle, die als `DataFrame` dargestellt wird. Den Typ DataFrame werden wir in der übernächsten Woche behandeln.

Es lässt sich aus einem `DataFrame` (=Tabelle) Objekt allerdings eine Spalte als `Series` Objekt herausnehmen. Dies passiert in dem folgenden Code-Stück:

```
import pandas as pd

url = 'https://datascience.hs-bochum.de/data/StromDE.csv'
data = pd.read_csv(url)
data.index = pd.to_datetime(data['Datum'])

# Extrahieren der Messwerte fuer produzierten Solarstrom
solarStrom = data['Solarstrom']

# Extrahieren der Messwerte fuer den Gesamtstrom Bedarf
stromBedarf = data['Verbrauch']
```

Nach diesen Zeilen enthält **`solarStrom`** die Messreihe der Stromerzeugung durch Photovoltaik-Anlagen in Deutschland pro Tag und **`stromBedarf`** den tatsächlichen Verbrauch je Tag (beides in MWh).

Beispiel Notebook

Auf der Seite

`https://datascience.hs-bochum.de`

finden Sie unter

Vorlesungen → *Semester* → *Wirtschaftsinformatik 2*

ein Jupyter-Notebook **`winf2-05-wochenaufgabe.ipynb`**, das Sie als Startpunkt für die Wochenaufgabe nutzen können.