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 DataFrame

- **DataFrame** ist ein Datentyp von Pandas, der logisch gesehen ganzen Tabellen entspricht:
  - Er besteht aus einer ***geordneten Sequenz von Spalten.***
  - ***Jede Spalte*** hat einen ***eindeutigen Datentyp*** (wie eine Series).
  - Verschiedene Spalten können verschiedene Datentypen haben.
  - Besitzt einen Zeilen- und einen Spaltenindex

? Anwendungs-beispiele nennen ?

```
# Beispiel Konkatenation von Series zu einem DataFrame
import pandas as pd
years = range(2020,2024)
shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index=years)
shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index=years)
shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index=years)
#Series in Dataframe einfügen mit axis=1, (sonst entsteht eine lange Series von drei Sequenzen hintereinander)
shops_df = pd.concat([shop1, shop2, shop3], axis=1)
print(shops_df)
```

Output ist ein DataFrame:

	0	1	2
2020	2409.14	1203.45	3412.12
2021	2941.01	3441.62	3491.16
2022	3496.83	3007.83	3457.19
2023	3119.55	3619.53	1963.10

? Nennen Sie den Zeilen- und den Spaltenindex?

Durch Konkatenation von **`Series-Objekten`** kann ein **`DataFrame-Objekt`** erzeugt werden.

DataFrame- Spalten

Die ***Spaltennamen*** und ***Werte*** im **`DataFrame-Objekt`** können mit den Attributen **`columns`** und **`columns.values`** abgefragt werden, sie können auch manipuliert werden.

```
# Spaltennamen und values des DataFrame abfragen (Beispiel weiter)
shops_df.columns
```

Output : `RangeIndex(start=0, stop=3, step=1)`

```
shops_df.columns.values
```

Output: `array([0, 1, 2])`

```
#  Spaltennamen setzen / manipulieren
shops_namen = ["shop1", "shop2", "shop3"]
shops_df.columns = shops_namen
print(shops_df)
```

Output:

	shop1	shop2	shop3
2020	2409.14	1203.45	3412.12
2021	2941.01	3441.62	3491.16
2022	3496.83	3007.83	3457.19
2023	3119.55	3619.53	1963.10

```
#Zugriff auf eine Spalte mit Indizierung
print(shops_df["shop1"])
```

```
#Zugriff auf eine Spalte mit Spaltennamen
print(shops_df.shop1)
```

Output(Series-Objekt dieser Spalte, beide Zugriffsarten):

```
2020    2409.14
2021    2941.01
2022    3496.83
2023    3119.55
Name: shop1, dtype: float64
```

```
2020    2409.14
2021    2941.01
2022    3496.83
2023    3119.55
Name: shop1, dtype: float64
```

DataFrames aus Dictionaries

Man kann aus Dictionaries (Spaltenname = Key, Spaltenwerte = Values der Seriesobjekte) direkt einen DataFrame erzeugen

```
# DataFrame aus Dictionaries erzeugen
cities = {"city": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1740119, 1602386,
                         1493900, 1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"]}
city_frame = pd.DataFrame(cities)
print(city_frame)
```

Output:

	city	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

DataFrame – Index ändern

Der Index für 0,1,2 wird dem DataFrame automatisch zugewiesen. Man kann den Index manipulieren über das Attribut **`Index.`**

```
# Dem DataFrame einen anderen Index zuweisen
ordinals = ["first", "second", "third", "fourth",
          "fifth", "sixth", "seventh", "eigth",
         "ninth", "tenth", "eleventh", "twelvth",
 "thirteenth"]

city_frame = pd.DataFrame(cities, index=ordinals)
print(city_frame)
```

Output:

	city	population	country
first	London	8615246	England
second	Berlin	3562166	Germany
third	Madrid	3165235	Spain
fourth	Rome	2874038	Italy
fifth	Paris	2273305	France
sixth	Vienna	1805681	Austria
seventh	Bucharest	1803425	Romania
eigth	Hamburg	1760433	Germany
ninth	Budapest	1754000	Hungary
tenth	Warsaw	1740119	Poland
eleventh	Barcelona	1602386	Spain
twelvth	Munich	1493900	Germany
thirteenth	Milan	1350680	Italy

DataFrame – Reihenfolge der Spalten

Mit dem Parameter **columns** kann die Reihenfolge der Spalten im DataFrame festgelegt werden.

```
# im DataFrame die Spalten umsortieren
city_frame = pd.DataFrame(cities,
                          columns = ["city",
                                     "country",
                                     "population"])
print(city_frame)
```

Output:

	city	country	population
0	London	England	8615246
1	Berlin	Germany	3562166
2	Madrid	Spain	3165235
3	Rome	Italy	2874038
4	Paris	France	2273305
5	Vienna	Austria	1805681
6	Bucharest	Romania	1803425
7	Hamburg	Germany	1760433
8	Budapest	Hungary	1754000
9	Warsaw	Poland	1740119
10	Barcelona	Spain	1602386
11	Munich	Germany	1493900
12	Milan	Italy	1350680

DataFrame – Zeilen selektieren mit loc I/II

Auf die ***Zeilen*** eines DataFrame kann man selektiv mit `loc` und `iloc` zugreifen.

```
# Im DataFrame den Index country setzten, statt Durchnummerierung
city_frame = pd.DataFrame(cities,
                          columns=("city", "population"),
                          index=cities["country"])
print(city_frame)
```

Output:

	city	population
England	London	8615246
Germany	Berlin	3562166
Spain	Madrid	3165235
Italy	Rome	2874038
France	Paris	2273305
Austria	Vienna	1805681
Romania	Bucharest	1803425
Germany	Hamburg	1760433
Hungary	Budapest	1754000
Poland	Warsaw	1740119
Spain	Barcelona	1602386
Germany	Munich	1493900
Italy	Milan	1350680

```
# mit loc Zeilen selekieren (z.B. nur für Index : Country = Germany)
print(city_frame.loc["Germany"])
```

Output:

	city	population
Germany	Berlin	3562166
Germany	Hamburg	1760433
Germany	Munich	1493900

```
# Liste von Indexwerten an loc übergeben, z.B. Germany und Poland
print(city_frame.loc[["Germany", "Poland"]])
```

Output:

	city	population
Germany	Berlin	3562166
Germany	Hamburg	1760433
Germany	Munich	1493900
Poland	Warsaw	1740119

DataFrame – Zeilen selektieren mit loc II/II

Auf die ***Zeilen*** eines DataFrame kann man selektiv mit `loc` und `iloc` zugreifen.

```
# Bedingungen für Spaltenwerte angeben
print(city_frame.loc[city_frame.population >4000000])
                              Output:
```

```
            city  population
England   London     8615246
```

```
print(city_frame.loc[city_frame.city=="London"] )
                              Output:
```

```
            city  population
England   London     8615246
```

```
# mehrere Bedingungen verknüpfen
cond1 = city_frame.population > 2000000
cond2 = city_frame.city.str.contains('P')
print(city_frame[cond1 & cond2]) //logisches Und
                              Output:
```

```
           city  population
France   Paris     2273305
```

```
print(city_frame[cond1 | cond2]) // logisches Oder
```

```
                                    Output:
```

```
            city  population
England   London     8615246
Germany   Berlin     3562166
Spain     Madrid     3165235
Italy       Rome     2874038
France     Paris     2273305
```

DataFrame – Zeilen selektieren mit query

Auf die ***Zeilen*** eines DataFrame kann man selektiv mit **query** zugreifen.

```
# Zeilen selektieren aus DataFrame mit query
print(city_frame.query("population > 2000000"))
```

Output:

	city	population
England	London	8615246
Germany	Berlin	3562166
Spain	Madrid	3165235
Italy	Rome	2874038
France	Paris	2273305

```
# Verwenden von Variablen mit @ im Query-Ausdruck
min_population = 4000000
print(city_frame.query("population > @min_population"))
```

Output:

	city	population
England	London	8615246

```
# Verwenden des Query-Ausdrucks für den Index des DataFrames
print(city_frame.query("index == 'Germany' or index == 'Poland'"))
```

Output:

	city	population
Germany	Berlin	3562166
Germany	Hamburg	1760433
Poland	Warsaw	1740119
Germany	Munich	1493900

DataFrame – Modifikationen I/III

In einen ***DataFrame können Spalten eingefügt werden*** mit verschiedenen Befehlen:
`index, loc, insert` und **`assign`**.

```
#  DataFrame verbrauch initialisieren
import pandas as pd
verbrauch = pd.DataFrame({'Kaffee':[3,0,2,8],
                          'Tee':[0,8,2,0]},
                         index = ['Ralph', 'Grit', 'Sarah', 'Alex'])
verbrauch.index.name = 'Team'
print(verbrauch)
```

Output:

Team	Kaffee	Tee
Ralph	3	0
Grit	0	8
Sarah	2	2
Alex	8	0

```
#  Spalten Kakao  und Kekse einfügen mit Index-Schreibweise
data =[[1,10],
       [0,12],
       [2,10],
       [0,12]]
verbrauch[['Kakao', 'Kekse']] = data
print(verbrauch)
```

Output:

Team	Kaffee	Tee	Kakao	Kekse
Ralph	3	0	1	10
Grit	0	8	0	12
Sarah	2	2	2	10
Alex	8	0	0	12

DataFrame – Modifikationen II/III

```
#  DataFrame verbauch mit insert um zwei Spalten + Werte erweitern.
import pandas as pd
verbrauch = pd.DataFrame({'Kaffee':[3,0,2,8],
                          'Tee':[0,8,2,0]},
                         index = ['Ralph', 'Grit', 'Sarah', 'Alex'])
verbrauch.index.name = 'Team'

verbrauch.insert(1, 'Kekse', [10,12,10,12])
verbrauch.insert(1, 'Kakao', [1, 0, 2, 0])
print(verbrauch)
```

Output:

Team	Kaffee	Kakao	Kekse	Tee
Ralph	3	1	10	0
Grit	0	0	12	8
Sarah	2	2	10	2
Alex	8	0	12	0

```
#  Spalten und Zeilen einfügen mit loc
verbrauch = pd.DataFrame({'Kaffee':[3,0,2,8],
                          'Tee':[0,8,2,0],
                          'Kakao': [1,0,2,0]},
                         index = ['Ralph', 'Grit', 'Sarah', 'Alex'])
verbrauch.index.name = 'Team'

# Einfügen einer neuen Spalte Kekse
verbrauch.loc[:, 'Kekse'] = [10,12,10,12]

# Einfügen einer neuen Zeile Andy
verbrauch.loc['Andy'] = [5,4,3,1]
print(verbrauch)
```

Output:

Team	Kaffee	Tee	Kakao	Kekse
Ralph	3	0	1	10
Grit	0	8	0	12
Sarah	2	2	2	10
Alex	8	0	0	12
Andy	5	4	3	1

DataFrame – Modifikationen III/III

```
# Im DataFrame verbrauch einer Spalte neue Werte zuweisen.
verbrauch = pd.DataFrame({'Kaffee':[3,0,2,8],
                          'Tee':[0,8,2,0],
                          'Kakao': [1,0,2,0]},
                         index = ['Ralph', 'Grit', 'Sarah', 'Alex'])
verbrauch.index.name = 'Team'

verbrauch['Kakao']=[1,0,3,1]   //Index-Schreibweise
verbrauch.Kaffee = [2,0,3,10]  //Schreibweise mit Punktoperator
print(verbrauch)               Output:

# Zeilenwerte austauschen mit loc
verbrauch.loc['Sarah'] = [3,3,5]
print(verbrauch)               Output:
```

Team	Kaffee	Tee	Kakao
Ralph	2	0	1
Grit	0	8	0
Sarah	3	2	3
Alex	10	0	1

Team	Kaffee	Tee	Kakao
Ralph	2	0	1
Grit	0	8	0
Sarah	3	3	5
Alex	10	0	1

DataFrame – einzelne Werte ändern

```
#  Einzelne Werte ändern mit at
verbrauch = pd.DataFrame({'Kaffee':[3,0,2,8],
                          'Tee':[0,8,2,0],
                          'Kakao': [1,0,2,0]},
                         index = ['Ralph', 'Grit', 'Sarah', 'Alex'])
verbrauch.index.name = 'Team'

#Grit's Teetassen um 2 erhöhen
verbrauch.at['Grit', 'Tee']+=2
print(verbrauch)
```

Output:

Team	Kaffee	Tee	Kakao
Ralph	3	0	1
Grit	0	10	0
Sarah	2	2	2
Alex	8	0	0

```
#  Zugriff mit iat  auf Zeilen-Spalten-Paare durch Integer-Position,  zählt immer ab 0 in der Wertematrix
print(verbrauch.iat[1, 1])     #1.Zeile,1.Spalte
```

Output: 10

DataFrame – Index ändern

```
# Index ändern
import pandas as pd

cities = ['Bielefeld', 'Minden', 'Herford']
data = {'City': cities,
        'Max_Temp':[22,23,23],
        'Min_Temp': [0,-1,0]}
df = pd.DataFrame(data)
print(df)
```

Output:

	City	Max_Temp	Min_Temp
0	Bielefeld	22	0
1	Minden	23	-1
2	Herford	23	0

```
# besser gleich index und columns angeben
df = pd.DataFrame(data,
                  columns=['Min_Temp', 'Max_Temp'],
                  index=cities)
print(df)
```

Output:

	Min_Temp	Max_Temp
Bielefeld	0	22
Minden	-1	23
Herford	0	23

Daten von https://www.wetter24.de/vorhersage/klima/deutschland/minden/18222996/

DataFrame – Umsortierung der Spalten und Zeilen mit `reindex`

```
#  Reihenfolge der Zeilen (Minden zuerst)
df2 = df.reindex(['Minden', 'Bielefeld', 'Herford'])
print(df2)
```

Output:

	Min_Temp	Max_Temp
Minden	-1	23
Bielefeld	0	22
Herford	0	23

```
#  Zusätzliche Zeile angeben, ggf Werte NaN
df3= df.reindex(['Minden', 'Bielefeld', 'Herford', 'Lübbecke'])
print(df3)
```

Output:

	Min_Temp	Max_Temp
Minden	-1.0	23.0
Bielefeld	0.0	22.0
Herford	0.0	23.0
Lübbecke	NaN	NaN

```
# Spaltenreihenfolge ändern mit columns + Zeilen für Minden und Bielefeld aussuchen
df4 = df.reindex(['Minden', 'Bielefeld'],
                 columns=['Max_Temp', 'Min_Temp'])
print(df4)
```

Output:

	Max_Temp	Min_Temp
Minden	23	-1
Bielefeld	22	0

DataFrame – Spalten umbenennen

Mit der DataFrame-Methode **rename** können wir Spalten umbennen. Den Parameter **inplace** setzt man auf **True**, wenn man das DataFrame-Objekt direkt ändern will und kein neues erzeugen möchte.

```
# gib ursprüngliches DataFrame aus
print(df)
```

Output:

	Min_Temp	Max_Temp
Bielefeld	0	22
Minden	-1	23
Herford	0	23

```
# Spalten umbennen
df.rename(columns={"Min_Temp": "Minimale Temperatur",
                   "Max_Temp": "Maximale Temperatur"},
                     inplace = True)
print(df)
```

Output:

	Minimale Temperatur	Maximale Temperatur
Bielefeld	0	22
Minden	-1	23
Herford	0	23

DataFrame – Spalte als neuen Index setzen

Mit der DataFrame-Methode **set_index** können wir eine Spalte als neuen Index setzen. Den Parameter **inplace** setzt man auf **True**, wenn man das DataFrame-Objekt direkt ändern will und kein neues erzeugen möchte.

```
# schon vorhandenes DataFrame nochmal, Index wird automatisch erzeugt mit 0, 1,2,3,...
cities = ['Bielefeld', 'Minden', 'Herford']
data = {'City': cities,
        'Max_Temp':[22,23,23],
        'Min_Temp': [0,-1,0]}
df = pd.DataFrame(data)
print(df)
```

Output:

	City	Max_Temp	Min_Temp
0	Bielefeld	22	0
1	Minden	23	-1
2	Herford	23	0

```
#  Index neu setzen mit set_index, inplace = true
df.set_index('City', inplace =True)
print(df)
)
```

Output:

City	Max_Temp	Min_Temp
Bielefeld	22	0
Minden	23	-1
Herford	23	0

DataFrame – Sortieren mit `sort_values`

Mit der DataFrame-Methode `sort_values` können wir den DataFrame nach den Werten einer Spalte (ggf auch für den Index) sortieren.

```
# vorheriges Beispiel des city_frame (wdh)
cities = {"city": ["London", "Berlin", "Madrid", "Rom
                   "Paris", "Vienna", "Bucharest",
                   "Budapest", "Warsaw", "Barcelon
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2
                         2273305, 1805681, 1803425
                         1754000, 1740119, 1602386
                         1493900, 1350680],
          "country": ["England", "Germany", "Spain",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland",
                      "Germany", "Italy"]}
city_frame = pd.DataFrame(cities)
print(city_frame)
```

Output:

	city	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

```
# city als Index setzen
city_frame.set_index('city', inplace =True)

# Sortieren nach den Werten einer Spalte
city_frame= city_frame.sort_values(by="city")
print(city_frame)
```

Output:

city	population	country
Barcelona	1602386	Spain
Berlin	3562166	Germany
Bucharest	1803425	Romania
Budapest	1754000	Hungary
Hamburg	1760433	Germany
London	8615246	England
Madrid	3165235	Spain
Milan	1350680	Italy
Munich	1493900	Germany
Paris	2273305	France
Rome	2874038	Italy
Vienna	1805681	Austria
Warsaw	1740119	Poland

DataFrame – verschachtelte Dictionaries I/II

Verschachtelte Dictionaries können an einen DataFrame übergeben werden. Die ***Indizes des äußeren Dictionarys*** entsprechen ***den Spalten*** des DataFrame; die ***inneren Schlüssel der Dictionarys*** entsprechen den ***Indizes der Zeilen*** im DataFrame.

```
# Beispiel verschachteltes Dictionary : Werte des Wachstums des BIP einiger EU-Länder 2018-
# 2021, Angabe in % ;
import pandas as pd

growth = {"Belgien": {"2018": 1.8,
                      "2019": 2.1,
                      "2020": -5.7,
                      "2021": 6.2},
          "Deutschland": {"2018": 1.0,
                          "2019": 1.1,
                          "2020": -3.7,
                          "2021": 2.6},
           "Frankreich:  -"-   # so ähnliche Daten wie oben
           "Griechenland": -"-
            "Italien": -"-
growth_frame = pd.DataFrame(growth)
print(growth_frame)
```

Datenquelle: https://ec.europa.eu/eurostat/databrowser/view/tec00115/default/table?lang=en

Output:

	Belgien	Deutschland	Frankreich	Griechenland	Italien
2018	1.8	1.0	1.9	1.7	0.9
2019	2.1	1.1	1.8	1.8	0.5
2020	-5.7	-3.7	-7.8	-9.0	-9.0
2021	6.2	2.6	6.8	8.3	6.7

DataFrame – verschachtelte Dictionaries II/II

Vertauschung von Zeilen und Spalten mittels `transpose`

```
# Vertauschen der Zeilen und Spalten , Beispiel weiter
print(growth_frame.transpose())  Output:

# alternative Property- Schreibweise geht auch genauso:
print(growth_frame.T)
```

	2018	2019	2020	2021
Belgien	1.8	2.1	-5.7	6.2
Deutschland	1.0	1.1	-3.7	2.6
Frankreich	1.9	1.8	-7.8	6.8
Griechenland	1.7	1.8	-9.0	8.3
Italien	0.9	0.5	-9.0	6.7

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 Dateiverarbeitung

Pandas bietet **Highlevel-Funktionalitäten** zum Einlesen und Schreiben von Daten aus verschiedenen Datenformaten wie z.B. ***JSON, HTML, SQL, MS EXCEL, CSV,...***

CSV - steht dabei für ***„Comma Separated Values“.***
- Pandas akzeptiert aber auch andere Trennzeichen.

exchange_rates_eu_us_tr_ch

year	CHF	TRY	USD	EUR
2000	1.688843	0.625219	1	1.082705
2001	1.687615	1.225588	1	1.116533
2002	1.558607	1.507226	1	1.057559
2003	1.346651	1.500885	1	0.884048
2004	1.243496	1.425537	1	0.803922
2005	1.245177	1.343583	1	0.8038
2006	1.253843	1.428453	1	0.796433
2007	1.200366	1.302931	1	0.729672
2008	1.08309	1.301522	1	0.679923
2009	1.088142	1.54996	1	0.716958
2010	1.042906	1.502849	1	0.754309
2011	0.888042	1.674955	1	0.718414
2012	0.937684	1.796001	1	0.778338
2013	0.926904	1.903768	1	0.752945
2014	0.916151	2.188542	1	0.752728
2015	0.962381	2.720009	1	0.901296
2016	0.985394	3.020135	1	0.903421
2017	0.984692	3.648133	1	0.885206
2018	0.977892	4.82837	1	0.846773
2019	0.993709	5.673819	1	0.893276
2020	0.938965	7.008605	1	0.875506
2021	0.913846	8.850408	1	0.845494
2022	0.95494	16.57	1	0.949624

Einlesen von CSV- Dateien I/II

Pandas ermöglicht das Einlesen von CSV-Dateien mit der Methode **read_csv.**

```
# csv-Datei einlesen
import pandas as pd

fname = "data/exchange_rates_eu_us_tr_ch.csv"
exchange_rates = pd.read_csv(fname, sep="\t") # tabulator für Separatoren -> leserlicher
print(exchange_rates.tail(7)) # nur die letzten 7 Zeilen ausgeben
```

Output:

	year	CHF	TRY	USD	EUR
16	2016	0.985394	3.020135	1	0.903421
17	2017	0.984692	3.648133	1	0.885206
18	2018	0.977892	4.828370	1	0.846773
19	2019	0.993709	5.673819	1	0.893276
20	2020	0.938965	7.008605	1	0.875506
21	2021	0.913846	8.850408	1	0.845494
22	2022	0.954940	16.570000	1	0.949624

exchange_rates_eu_us_tr_ch.csv — Bearbeitet

year	CHF	TRY	USD	EUR
2000	1.688843	0.625219	1	1.082705
2001	1.687615	1.225588	1	1.116533
2002	1.558607	1.507226	1	1.057559
2003	1.346651	1.500885	1	0.884048
2004	1.243496	1.425537	1	0.803922
2005	1.245177	1.343583	1	0.8038
2006	1.253843	1.428453	1	0.796433
2007	1.200366	1.302931	1	0.729672
2008	1.08309	1.301522	1	0.679923
2009	1.088142	1.54996	1	0.716958
2010	1.042906	1.502849	1	0.754309
2011	0.888042	1.674955	1	0.718414
2012	0.937684	1.796001	1	0.778338
2013	0.926904	1.903768	1	0.752945
2014	0.916151	2.188542	1	0.752728
2015	0.962381	2.720009	1	0.901296
2016	0.985394	3.020135	1	0.903421
2017	0.984692	3.648133	1	0.885206
2018	0.977892	4.82837	1	0.846773
2019	0.993709	5.673819	1	0.893276
2020	0.938965	7.008605	1	0.875506
2021	0.913846	8.850408	1	0.845494
2022	0.95494	16.57	1	0.949624

Methode **read_csv** nutzt erste Zeile automatisch als Spaltennamen. Sie öffnet und schließt die Datei auch automatisch.

Einlesen von CSV- Dateien II/II

Methode `read_csv:`

- erste Zeile überspringen mit Parameter `header=0`
- Erste Spalte zum Index machen mit Parameter `index_col=0`

```
# csv-Datei einlesen, erste Zeile überspringen und eigene Spaltennamen übergeben, Index setzen
auf erste Spalte (Jahre)
import pandas as pd

fname = "data/exchange_rates_eu_us_tr_ch.csv"
column_names = ['year', 'CH', 'TR', 'US', 'EU']
exchange_rates = pd.read_csv(fname,
                             sep="\t",
                             header=0,
                             index_col=0,
                             names=column_names)
print(exchange_rates.head())
```

Output:

year	CH	TR	US	EU
2000	1.688843	0.625219	1	1.082705
2001	1.687615	1.225588	1	1.116533
2002	1.558607	1.507226	1	1.057559
2003	1.346651	1.500885	1	0.884048
2004	1.243496	1.425537	1	0.803922

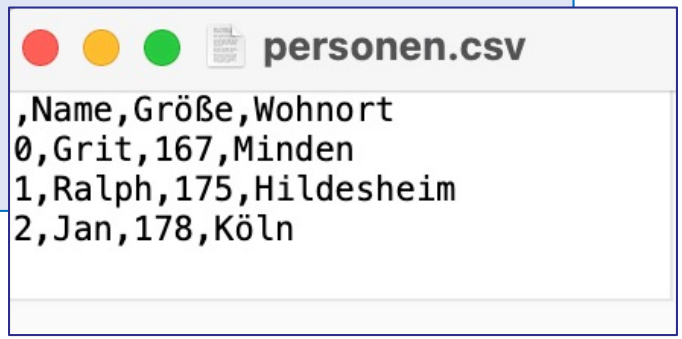
Schreiben von CSV- Dateien: Einfaches Beispiel

Mit der Methode `to_csv` kann man einen DataFrame in einer csv-Datei speichern.

```
# Schreiben von csv-Dateien - einfaches Beispiel
import pandas as pd

data = [('Grit', 167, 'Minden'),
        ('Ralph', 175, 'Hildesheim'),
        ('Jan', 178, 'Köln')]

df = pd.DataFrame(data)
df.columns = ['Name', 'Größe', 'Wohnort']
df.to_csv("data/personen.csv")
```



erzeugte CSV-Datei „personen.csv“:

- Header-Zeile beginnt mit Komma
- Datenzeilen werden automatisch nummeriert (0,1,...)

➔ Beim neuen Einlesen den index_col = 0 setzen, um zu verhindern, dass automatisch generierter Index dann wieder als neue Spalte interpretiert wird:

```
df = pd.read_csv('data/personen.csv', index_col=0)
```

-> Ggf. kann auch gleich beim Schreiben der CSV verhindert werden, dass ein neuer Index erzeugt wird:

```
df.to_csv("data/personen_2.csv", index=False)
```

Einlesen von zwei CSV- Dateien und Erzeugen einer neuen CSV-Datei

Beispiel:

- Einlesen der beiden Dateien `countries_male_population.csv` und `countries_female_population.csv`. (enthalten jeweils die männlichen und weiblichen Bevölkerungszahlen von Ländern).
- Addieren der Bevölkerungszahlen zur Gesamtbevölkerungszahl pro Land
- Schreiben einer neuen CSV-Datei `countries_total_population1.csv`

```
import pandas as pd

column_names = ["Country"] + list(range(2003, 2013))
male_pop = pd.read_csv("data/countries_male_population.csv",
                       header=None, index_col=0, names=column_names)
female_pop = pd.read_csv("data/countries_female_population.csv",
                         header=None,index_col=0,names=column_names)
population = male_pop + female_pop
print(population.head())
```

Output:

```
                2003      2004      2005      2006      2007      2008  \
Country
Australia   19872646  20091504  20339759  20605488  21015042  21431781
Austria      8067289   8140122   8206524   8265925   8298923   8331930
Belgium     10355844  10396421  10445852  10511382  10584534  10666866
Canada      31361611  31372587  31989454  32299496  32649482  32927372
Czech Republic  10203269  10211455  10220577  10251079  10287189  10381130

                2009      2010      2011      2012
Country
Australia   21874920  22342398  22620554  22683573
Austria      8355260   8375290   8404252   8443018
Belgium     10753080  10839905  10366843  11035958
Canada      33327337  33334414  33927935  34492645
Czech Republic  10467542  10506813  10532770  10505445
```

```
population.to_csv("data/countries_total_population1.csv")
```

DataFrames : Konkatenation von DataFrames

Beispiel weiter:

- Die drei DataFrames „population“ (mit Summe fem./male) „female“ und „male“ konkatenieren

```
# Konkatenation der drei DataFrames und Erzeugen des dataFrames pop_complete

pop_complete = pd.concat([population,
                          male_pop,
                          female_pop],
                         keys=["total", "male", "female"])
print(pop_complete.iloc[[0, 1, 2, 29, 30, 31, 32, 59, 60, 61, 62]])
                                                  # iloc selektiert die gewünschten Zeilen
```

- hierarchischer Index aus **keys** und **country**
- **keys** ist primärer Index

		2003	2004	2005	2006	2007
	Country					
total	Australia	19872646	20091504	20339759	20605488	21015042
	Austria	8067289	8140122	8206524	8265925	8298923
	Belgium	10355844	10396421	10445852	10511382	10584534
	United States	288774226	290810719	294442683	297308143	300184434
male	Australia	9873447	9990513	10121438	10257418	10444622
	Austria	3909120	3949825	3986296	4019354	4037171
	Belgium	5066885	5087176	5111325	5143821	5181408
	United States	141957038	143037260	144947584	146394361	147922764
female	Australia	9999199	10100991	10218321	10348070	10570420
	Austria	4158169	4190297	4220228	4246571	4261752
	Belgium	5288959	5309245	5334527	5367561	5403126
	Canada	15829173	15834015	16146667	16303914	16478759

		2008	2009	2010	2011	2012
	Country					
total	Australia	21431781	21874920	22342398	22620554	22683573
	Austria	8331930	8355260	8375290	8404252	8443018
	Belgium	10666866	10753080	10839905	10366843	11035958
	United States	304846731	305127551	307756577	309989078	312232049
male	Australia	10660917	10888385	11124254	11260747	11280804
	Austria	4054214	4068047	4079093	4095337	4118035
	Belgium	5224309	5268651	5312221	5370234	5413801
	United States	150336755	150464902	151840906	152449134	153596908
female	Australia	10770864	10986535	11218144	11359807	11402769
	Austria	4277716	4287213	4296197	4308915	4324983
	Belgium	5442557	5484429	5527684	4996609	5622157
	Canada	16601231	16802833	16807738	17101813	17379104

DataFrames : Hierarchischen Index swappen

Beispiel weiter:

- Konkateniertes DataFrame aufbereiten
  - IndexLevel umdrehen (Land als Index in der ersten Spalte)
  - Index sortieren

```
# Hierarchien umdrehen (Länder als erste Spalte)
df = pop_complete.swaplevel()
# Länder als index alpahbetisch sortieren
df.sort_index(inplace=True)
print(df.head(5))
```

Output:

```
                      2003      2004      2005      2006      2007      2008  \
Country
Australia female   9999199  10100991  10218321  10348070  10570420  10770864
          male     9873447   9990513  10121438  10257418  10444622  10660917
          total   19872646  20091504  20339759  20605488  21015042  21431781
Austria   female   4158169   4190297   4220228   4246571   4261752   4277716
          male     3909120   3949825   3986296   4019354   4037171   4054214

                      2009      2010      2011      2012
Country
Australia female   10986535  11218144  11359807  11402769
          male     10888385  11124254  11260747  11280804
          total    21874920  22342398  22620554  22683573
Austria   female    4287213   4296197   4308915   4324983
          male      4068047   4079093   4095337   4118035
```

```
df.to_csv("data/countries_total_population2.csv")
```

Lesen und Schreiben von JSON-Dateien

Das Lesen und Schreiben von JSON-Dateien geht analog zur Bearbeitung von CSV-Dateien mit den Methoden `to_json` und `read_json.`

```
# Lesen und Schreiben von json

import pandas as pd
shoppings = pd.DataFrame({'Einkauf_1': {'apples': 44, 'oranges': 61},
                          'Einkauf_2': {'apples': 32, 'oranges': 89},
                         'Einkauf_3': {'apples': 93, 'oranges': 17}})

# Abspeichern in JSON-Datei:
shoppings.to_json('shoppings.json')
```

Geschriebene CVS-Datei:

shoppings.json

```
{"Einkauf_1":
{"apples":44,"oranges":61},"Einkauf_2":
{"apples":32,"oranges":89},"Einkauf_3":
{"apples":93,"oranges":17}}
```

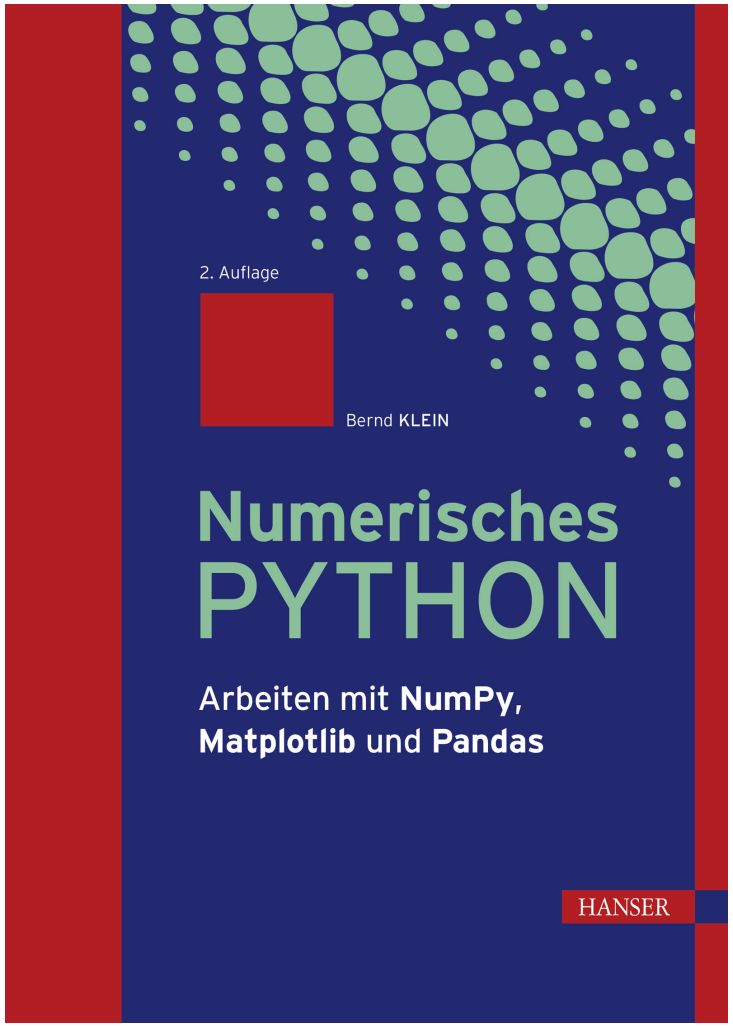
```
# Einlesen der eben geschriebenen Datei:
shoppings2 = pd.read_json('shoppings.json')
print(shoppings2)
```

Output:

```
         Einkauf_1  Einkauf_2  Einkauf_3
apples          44         32         93
oranges         61         89         17
```

Literaturangaben und Quellen

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



So geht es weiter : Datenanalyse mit Python

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