Wirtschaftsinformatik 2

Einstieg in die Datenanalyse mit Pandas

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

Hochschule Bochum

Wintersemester 2024/2025

Vorbemerkungen zum Block Pandas DataFrames

Pandas ist ein umfangreiches Python-Modul zur Datenanalyse. Zentral dafür sind die Datentypen `Series` und `DataFrame` als Darstellung für Daten. Im letzten Block ging es um die *Series* Elemente, mit denen Wertereihen dargestellt werden.

In diesem Block erweitern wir das Ganze um den Typ *DataFrame*, der eine Darstellung von Tabellen ermöglicht. Dabei kann auf jede Tabellen-Zelle und auf ganze Zeilen und Spalten zugegriffen werden. Für die Darstellung von einzelnen Zeilen und einzelnen Spalten wird in Pandas wieder der Typ *Series* benutzt.

Daten in tabellarischer Form kommen in einer Vielzahl von betriebswirtschaftlichen Anwendungen vor. Ein Beispiel ist die nachfolgende Tabelle mit Bestellungen aus dem Schoko-Kugel Shop, bei dem jede Zeile einer Bestellung entspricht.

Datum	BestellungNr	KundeNr	Anzahl_Gold	Anzahl_Rot	Anzahl_Blau	Anzahl_Gruen	Betrag
2023-10-12	77	78	1	0	0	0	0.050
2023-10-12	78	79	1	2	0	0	0.150
2023-10-12	79	80	1	1	1	0	0.150
2023-10-12	80	81	1	0	1	1	0.150
2023-10-12	81	82	1	1	1	0	0.150
2023-10-12	82	83	0	0	1	0	0.050

Die Tabelle enthält Spalten für die Bestellnummer, den Zeitpunkt der Bestellung, sowie die Anzahl der jeweils bestellten Produkte und den Gesamtbetrag der Bestellung.

Solche Tabellen können in Pandas einfach geladen und analysiert werden. Genau das ist das Ziel der restlichen Vorlesungen/Blöcke in *Wirtschaftsinformatik 2*.
Ein kleines Beispiel als *Schnellstart*:

```python
# Pandas importieren
import pandas as pd

# Tabelle einlesen
url = 'https://data.hsbo.de/Bestellungen.csv'
tabelle = pd.read_csv(url)

# Statistiken/Analysen/Ergebnisse berechnen
tabelle.describe()
```

Probieren Sie es im Notebook aus!

Im Folgenden werden wir uns im Detail mit `DataFrame`s und deren Konzepten und Benutzung auseinandersetzen...

Datenanalyse mit Pandas

DataFrame – Ein Datentyp für Tabellen

DataFrame als Teil von pandas

Wie der Typ *Series*, ist auch `DataFrame` ein Datentyp, der im Pandas Modul enthalten ist. Die folgenden Folien gehen davon aus, dass zunächst das Pandas Modul importiert wurde und *Series* und *DataFrame* noch explizit eingebunden wurden. Das geschieht mit den folgenden Zeilen:

```
# Pandas als 'pd' importieren
import pandas as pd

from pandas import Series, DataFrame
```

Datentyp für Tabellen: `DataFrame`

DataFrame Objekte repräsentieren Tabellen, d.h. in einer Variablen kann so eine ganze Tabelle gespeichert werden und auf die einzelnen Zeilen/Spalten/ Zellen der Tabelle zugegriffen werden.

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

- Besitzt Zeilen- und Spalten-Index
- Jede Zeile/jede Spalte als `Series` Objekt zugreifbar

Im abgebildeten Beispiel kann z.B. über den Namen "`a2`" auf die zweite Spalte zugegriffen werden.

Definition eines DataFrame

DataFrame Objekte werden typischerweise aus Dateien erzeugt, d.h. man liest Daten aus einer Datei in eine Tabelle. Sie lassen sich aber auch direkt aus z.B. einer Liste von Zeilen definieren:

```
z0 = [4,1,2,9]    # zeile 0
z1 = [5,1,3,6]    # zeile 1
z2 = [3,8,7,4]    # zeile 2

df = DataFrame([z0, z1, z2],
               columns=['a1', 'a2', 'a3', 'a4'])
```

Der Aufruf `DataFrame([..],..)` derzeugt eine DataFrame Tabelle und speichert sie in der Variablen `df`. Der Parameter `columns` gibt an, wie die Spalten der Tabelle heissen sollen. Wenn man den Parameter weglässt, werden die Spalten einfach durchnummeriert (0, 1,..).

Die Form eines DataFrame

Wenn man eine Tabelle erzeugt/eingelesen hat, ist es zunächst mal interessant zu wissen, wieviel Zeilen/Spalten die Tabelle bzw. der DataFrame enthält. Im Folgenden sei in der Variablen `df` ein DataFrame gespeichert.

df =

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

Über `df.shape` lässt sich die Form von `df` abfragen. Man bekommt damit ein Tupel der Art `(zeilen,spalten)`:

```
# 'shape' des obigen DataFrames  df

(zeilen, spalten) = df.shape   # zeilen=3, spalten=4
```

Einzelne Spalten sind `Series` Objekte

Häufig ist es erforderlich, auf Spalten einer Tabelle zuzugreifen und z.B. Berechnungen durchzuführen. Dazu lassen sich Spalten in einem DataFrame über

```
spalte = df['a1']
```

adressieren und z.B. in eine neue Variable schreiben. Die Spalten sind selbst wieder `Series` Objekte.

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

```
a2 = df['a2']  # Spalte 'a2' selektieren
type(a2)       # -> pandas.core.series.Series
```

Mit den Spalten-`Series`-Objekten rechnen

Natürlich kann mit den Series Objekten aus der Tabelle wieder ganz normal gerechnet werden. Auf diese Weise lassen sich z.B. zwei Spalten addieren oder eine Spalte multiplizieren:

```
summe_a1_a2 = df['a1'] + df['a2']
a1_verdoppelt = df['a1'] * 2
```

Die Ergebnisse können wieder als neue Spalten in die Tabelle / den DataFrame geschrieben werden:

```
df['summe_von_a1a2'] = summe_a1_a2
df['a1_doppelt'] = a1_verdoppelt
```

Erstellen Sie sich einen kleinen DataFrame und experimentieren Sie mit Spalten-Berechnungen herum!

Probieren Sie es im Notebook aus!

Wenn die Spalten `Series` Objekte sind...

... dann kann man mit mehreren Series-Objekten auch einen DataFrame bauen:

In diesem Beispiel erstellen wir ein paar Series-Objekte und einen leeren DataFrame und fügen die Series Objekte dann als Spalten in den leeren DataFrame ein:

```
s1 = Series([1,1,1])
s2 = Series([2,2,2])
s3 = Series([3,3,3])

d = DataFrame()
d['s1'] = s1
d['s2'] = s2
d['s3'] = s3
```

	s1	s2	s3
0	**1**	**2**	**3**
1	**1**	**2**	**3**
2	**1**	**2**	**3**

Mehr als eine Spalte selektieren

Häufig will man aus einer Tabelle ein paar Spalten auswählen, weil z.B. nur diese Spalten für die weitere Analyse gebracht werden. Dies funktioniert mit einem ähnlichen Zugriff:

```
df[LISTE_VON_SPALTEN_NAMEN]
```

In diesem Fall packt man eine Liste mit Spaltennamen in die eckigen Klammern, zum Beispiel die Spalten `a2` und `a3`:

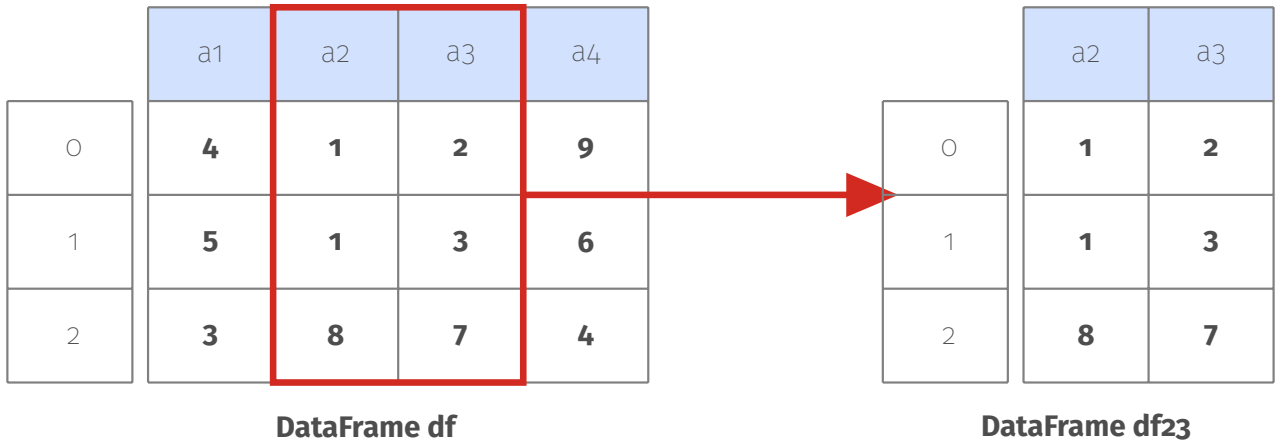
```
spalten2und3 = ['a2', 'a2']
kleineTabelle = df[spalten2und3]
```

Das Ergebnis ist in diesem Fall keine Liste von Series-Objekten, sondern wieder eine Tabelle, die als DataFrame zurückgeliefert wird!
Das lässt sich natürlich auch ohne die Extra-Variable `spalten2und3` in einem Schritt erledigen:

```
kleineTabelle = df[ ['a2','a3'] ]
```

Mehrere Spalten ergeben wieder einen DataFrame

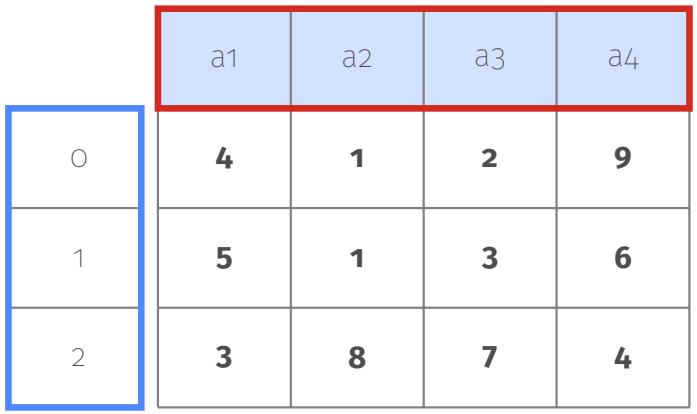
Sobald man mehrere Spalten (oder auch Zeilen) auswählt, passt das natürlich nicht mehr in ein Series Objekt. D.h. bei der Auswahl mehrerer Spalten ergibt das wieder einen (kleineren) DataFrame:




```
# z0,z1,z2 wie in Folie 7
df = DataFrame([z0, z1, z2],
               columns=['a1', 'a2', 'a3', 'a4'])
df23 = df[['a2','a3']]
```

Zeilen-/Spalten-Index eines DataFrames

Ein DataFrame hat zwei wichtige Komponenten: Den Zeilenindex `.index` und den Spaltenindex `.columns`. Die Abbildung zeigt die beiden Indizes, über die auf sämtliche Teile des DataFrames zugegriffen werden kann.



Mit `.loc[..]` kann über die Indizes auf den DataFrame zugegriffen werden.

Selektieren von Zeilen/Spalten

Über `.loc[..]` kann auf einzelne Zellen und auch größere Teile eines DataFrames/einer Tabelle zugegriffen werden. Dabei nutzt `.loc` den Index und die Spaltennamen. In beiden Fällen, kann man für einen DataFrame in der Variablen `df` Zeilen und Spalten angeben, die selektiert werden sollen.

```
df.loc[ ZEILEN , SPALTEN ]
```

Die Selektoren ZEILEN bzw. SPALTEN können Zahlen, Listen von Zahlen, oder auch Slices (vgl. Listen) sein. SPALTEN kann auch weggelassen werden – dann wird die komplette Zeile ausgewählt:

```
# Zelle in erster Zeile, dritter Spalte:
wertInZelle = df.loc[0,"a2"]

#  Die erste Zeile (als Series Objekt!)
zeile = df.loc[0,:]  # das Gleiche wie df.loc[0]

# Die erste Spalte:
spalte = df.loc[:,"a1"]
```

Einzelne Zeilen ergeben ebenfalls `Series` Objekte

Natürlich können auch Zeilen in einem DataFrame herausgepickt werden. Der einfachste Weg dazu ist `loc`. Damit lässt sich die Zeile an Position `x` abfragen:

```
zeile_x = df.loc[x]
```

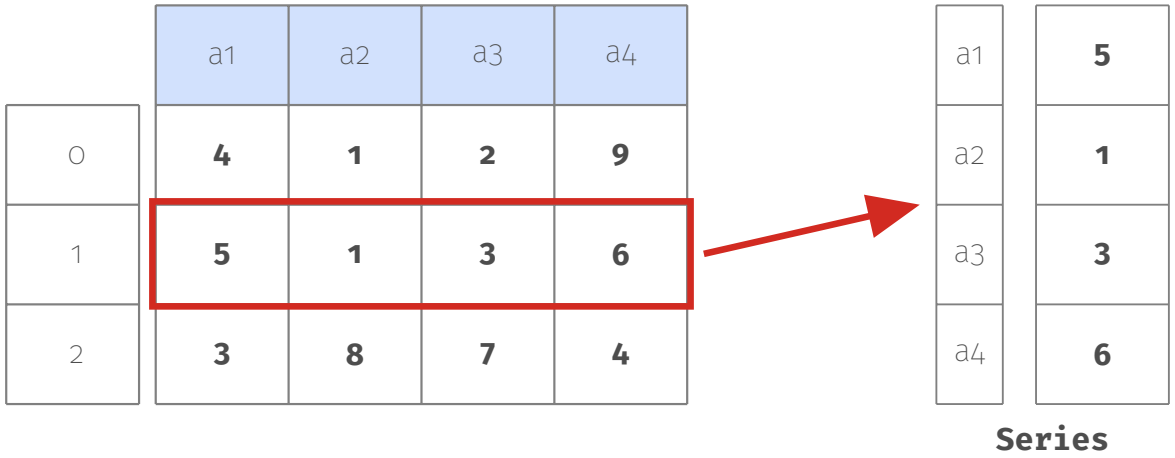
Dafür muss `x` ein Wert sein, der als **Zeilenindex** im DataFrame vorkommt. In der Darstellung der DataFrames haben wir den Zeilenindex immer als separate Spalte an den Anfang gestellt:

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

Zeilenindex

Einzelne Zeilen ergeben ebenfalls Series Objekte

Wenn man auf eine Zeile in einem DataFrame zugreift, bekommt man als Ergebnis ebenfalls ein Series Objekt. Diese Zeilen-Series Objekt hat als Index die Spaltennamen der Tabelle:




```
z2 = df.loc[1]    # Zeile 1 selektieren
type(z2)          # -> pandas.core.series.Series
```

Beispiel: Einzelne Zeilen im DataFrame

Natürlich können wir auch mit diesen Zeilen-Series Objekten wie mit ganz normalen Series-Objekten rechnen:

```
zeile2 = df.loc[1]
summe = df.loc[0] + df.loc[1] + df.loc[2]
```

Die Zeilensumme könnten wir auch als neue Zeile wieder zu unserem DataFrame hinzufügen:

```
df.loc['summe'] = df.loc[0] + df.loc[1] + df.loc[2]
```

Der Index kann verändert werden

Bisher haben wir DataFrames betrachtet, bei denen der Index von 0 an durchnummeriert wurde. Wie bei Series-Objekten, kann auch bei DataFrames der Index angepasst werden.
Als Beispiel wollen wir den Zeilen unseres DataFrames von zuvor die Buchstaben **A**, **B** und **C** zuordnen:

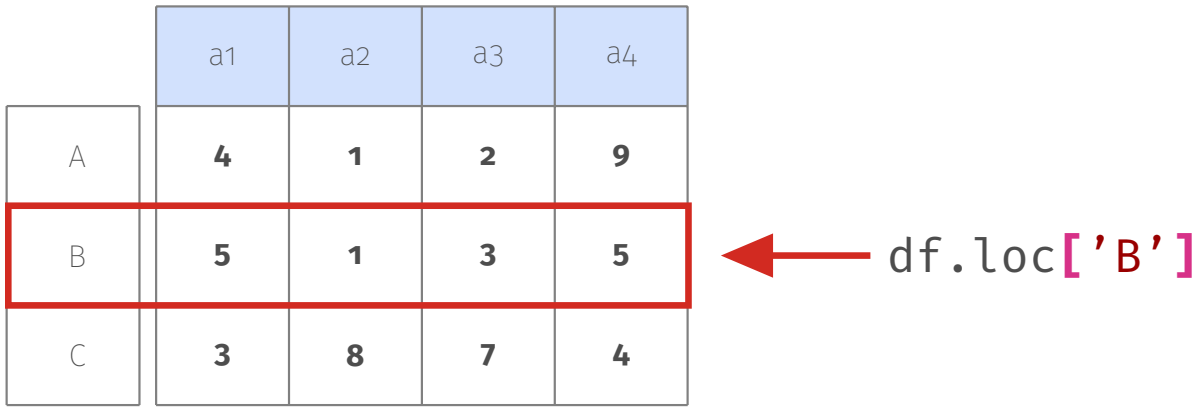
```
df.index = ['A','B','C']
```

	a1	a2	a3	a4
A	**4**	**1**	**2**	**9**
B	**5**	**1**	**3**	**5**
C	**3**	**8**	**7**	**4**

Der Index kann verändert werden

Bisher haben wir DataFrames betrachtet, bei denen der Index von 0 an durchnummeriert wurde. Wie bei Series-Objekten, kann auch bei DataFrames der Index angepasst werden.
Als Beispiel wollen wir den Zeilen unseres DataFrames von zuvor die Buchstaben A, B und C zuordnen:

```
df.index = ['A','B','C']
```



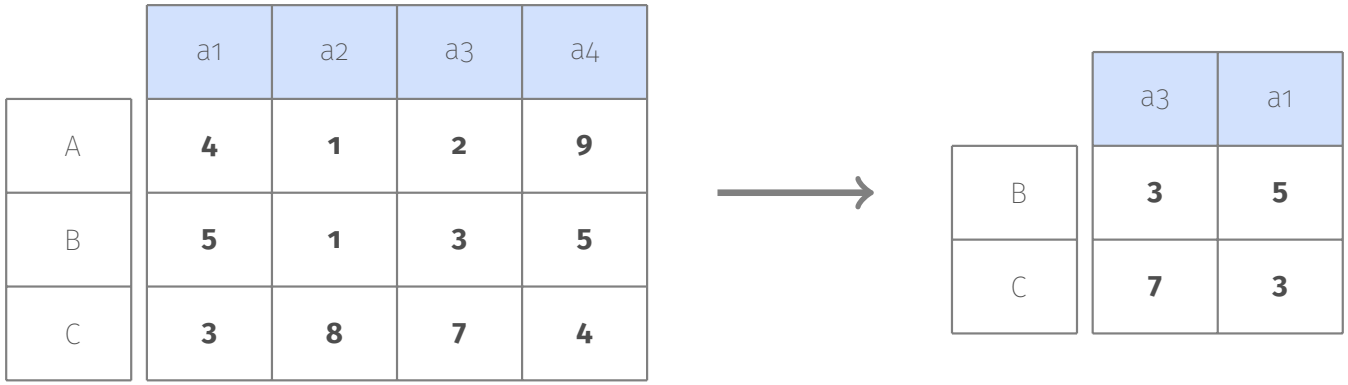

```
zeileB = df.loc['B']  # Zugriff per Index
```

Zeilen/Spalten mit Listen auswählen

Mit **Listen** für die Selektoren ZEILEN und SPALTEN können wir angeben, welche Zeilen und Spalten wir genau haben wollen, zum Beispiel:

```
zeilen = ['B','C']
spalten = ['a3','a1']
neueTabelle = df.loc[ zeilen, spalten ]
```

Die Variable `neueTabelle` enthält damit einen DataFrame, der die Zeilen mit dem Index **B** und **C** des originalen DataFrames `df` hat und die Spalten `a3` und `a1`.



Auswahl mit Bereichen START : ENDE

Pandas unterstützt auch *Bereiche* um eine Auswahl zu treffen. In `.loc[..]` kann dabei der Start-Index und der End-Index von den Zeilen (bzw. Spalten) angegeben werden, die ausgewählt werden sollen.

Als Beispiel wählen wir hier alle Zeilen von `'B'` bis `'C'` aus:

```
df.loc['B':'C']
```

	a1	a2	a3	a4
A	**4**	**1**	**2**	**9**
B	**5**	**1**	**3**	**5**
C	**3**	**8**	**7**	**4**

Auswahl mit Bereichen START : ENDE

Pandas unterstützt auch *Bereiche* um eine Auswahl zu treffen. In `.loc[..]` kann dabei der Start-Index und der End-Index von den Zeilen (bzw. Spalten) angegeben werden, die ausgewählt werden sollen.

Als Beispiel wählen wir hier alle Zeilen von `'B'` bis `'C'` aus:

```
df.loc['B':'C']
```

		a1	a2	a3	a4
	A	**4**	**1**	**2**	**9**
'B' ⟶	B	**5**	**1**	**3**	**5**
'C' ⟶	C	**3**	**8**	**7**	**4**

Beispiel: Zeilenindex und Bereiche

Wir betrachten nun einen DataFrame mit Klimadaten aus Deutschland, den wir in der Variablen `klima` gespeichert haben. Dabei betrachten wir einen Ausschnitt der ersten 4 Zeilen:

	Jahr	Temp
0	**1881**	**-5.360**
1	**1882**	**0.410**
2	**1883**	**-0.320**
3	**1884**	**2.860**

Beispiel: Zeilenindex und Bereiche

Wir betrachten nun einen DataFrame mit Klimadaten aus Deutschland, den wir in der Variablen `klima` gespeichert haben. Dabei betrachten wir einen Ausschnitt der ersten 4 Zeilen:

	Jahr	Temp
0	**1881**	**-5.360**
1	**1882**	**0.410**
2	**1883**	**-0.320**
3	**1884**	**2.860**

	Jahr	Temp
1881	**1881**	**-5.360**
1882	**1882**	**0.410**
1883	**1883**	**-0.320**
1884	**1884**	**2.860**

Wir ersetzen den Index [0,1,..] durch die Werte aus der Spalte `Jahr`:

```
klima.index = klima['Jahr']
```

Beispiel: Zeilenindex und Bereiche

Wir haben nun die Spalte mit den Jahreszahlen als Index in unserem DataFrame. Das ermöglicht uns mit *Bereichen* Teile des DataFrames leichter auszuwählen. Zum Beispiel, die Temperaturen der Jahre 2020 bis 2024:

```
klima.loc['2020':'2024']
```

	Jahr	Temp
2020	**2020**	**3.480**
2021	**2021**	**0.630**
2022	**2022**	**2.790**
2023	**2023**	**3.620**
2024	**2024**	**1.500**

Zwischenfazit: Was ist nun alles möglich?

Sie können mit den vorgestellten Methoden nun auf sämtliche Zeilen, Spalten und einzelne Zellen eines DataFrames zugreifen. D.h. Sie können Werte aus den Tabellen auslesen und in Variablen schreiben und auch Werte aus Variablen wieder in die Tabelle. Betrachen wir den folgenden DataFrame, der in der Variablen `df` gespeichert ist:

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

```
x = df.loc[2, 'a3']
df.loc[1, 'a2'] = 31
```

Welchen Wert hat `x`? Wie sieht der DataFrame nach diesen Anweisungen aus?

Daten Lesen und Explorieren

Pandas enthält Funktionen zum Lesen von DataFrames

Zum Abschluss betrachten wir die Daten, die wir in einen DataFrame einlesen können. Typischerweise werden Daten häufig in CSV- oder Excel-Dateien gespeichert. In Wirtschaftsinformatik 1 haben wir über Datenbanken gesprochen - auch hieraus kann Pandas Daten einlesen.

- `pd.read_csv` - Lesen aus CSV-Datei
- `pd.read_excel` - Lesen aus Excel-Datei

Der folgende Code zeigt Beispiele um Daten aus einer CSV Datei einzulesen - das kann eine lokal existierende Datei oder eine entfernte URL sein:

```
# Lesen aus der Datei 'meine-daten.csv'
df = pd.read_csv("meine-daten.csv")

# Funktioniert auch mit URLs
df = pd.read_csv("https://data.hsbo.de/iris.csv")
```

CSV ist weit verbreitetes Datenformat (*comma separated values*)

CSV-Dateien sind Dateien, die Tabellen speichern. Dazu wird in jeder Zeile der Datei eine Tabellenzeile abgelegt. Die Werte der Zellen in jeder Zeile werden durch Komma voneinander getrennt. Eine CSV-Datei kann eine Kopfzeile haben, die die Spaltennamen enthält, zum Beispiel:

```
NR,a1,a2,a3,a4,x1,"Art der Pflanze"
1,4,1,2,9,2.0,"setosa"
2,5,1,3,6,0.2,"versicolor"
3,3,8,7,4,13.0,"virginica"
```

Pandas unterstützt verschiedene Optionen

Mit der Pandas Funktion `read_csv(..)` lässt sich eine Quelle im CSV-Format in einen DataFrame einlesen. Als Trennzeichen zwischen den Zellen wird normalerweise das Komma benutzt - es kann aber auch ein Semikolon vorkommen. In diesem Fall, muss der Parameter `sep` angepasst werden:

```
pd.read_csv( "datei.csv", sep=",",
                header="infer", names=None)
```

sep	Trennzeichen der Spalten (z.B. ';')
header	Wie werden Spaltennamen bestimmt?
names	Liste mit Spaltennamen

Lesen einer CSV-Datei mit Semikolon, ohne Header:

```
columns = ["a1", "a2", "a3"]
df = pd.read_csv("datei.csv", sep=";", names=columns)
```

Nachdem Laden – Was ist drin, im DataFrame?

Wenn man Daten in einen DataFrame `df` geladen hat, geht es meist als nächstes an die *Datenexploration*, d.h. man ist interessiert an Fragen wie:

- Wie sieht der DataFrame aus? → `df.head(5)`
- Wieviele Zeilen/Spalten gibt es? → `df.shape`
- Welche Spalten/Datentypen? → `df.columns` / `df.dtypes`
- Wertebereiche der Spalten? → `df.describe()`

```python
import pandas as pd
df = pd.read_csv( "https://data.hsbo.de/iris.csv")

# Anfang anzeigen (ersten 5 Zeilen)
df.head(5)

# Spalten-Statistiken
df.describe()
```

Erkundung eines DataFrames

Der Datensatz unter `https://data.hsbo.de/iris.csv` enthält Daten über Schwertlilien (Pflanzen) – und zwar die Länge/Breite der Kelch- und Blütenblätter. Im Kapitel über *Series* wurde erläutert, dass ein Series Objekt Daten eines bestimmten Datentyps enthält. Ein *DataFrame* enthält mehrere Series Objekte als Spalten und so liefert `.dtypes` ein Series-Objekt, dass die Datentypen aller Spalten enthält:

```
import pandas as pd

iris = pd.read_csv('https://data.hsbo.de/iris.csv')

info = df.dtypes  # Information ueber die Datentypen
```

Spalten-Statistiken mit `describe()`

Eine hilfreiche Funktion von *DataFrame* ist die Methode `.describe()`. Diese Methode berechnet für alle numerischen Spalten des DataFrame automatisch eine ganze Reihe von Statistiken:

```
# Statistiken berechnen:
iris.describe()
```

```
# Und das Ergebnis
# von iris.describe()?
stats = iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150	150	150	150
mean	5.843	3.054	3.759	1.199
std	0.828	0.434	1.764	0.763
min	4.300	2	1	0.100
25%	5.100	2.800	1.600	0.300
50%	5.800	3	4.350	1.300
75%	6.400	3.300	5.100	1.800
max	7.900	4.400	6.900	2.500

Spalten-Statistiken mit `describe()`

Eine hilfreiche Funktion von *DataFrame* ist die Methode `.describe()`. Diese Methode berechnet für alle numerischen Spalten des DataFrame automatisch eine ganze Reihe von Statistiken:

```
# Statistiken berechnen:
iris.describe()
```

```
# Und das Ergebnis
# von iris.describe()?
stats = iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150	150	150	150
mean	5.843	3.054	3.759	1.199
std	0.828	0.434	1.764	0.763
min	4.300	2	1	0.100
25%	5.100	2.800	1.600	0.300
50%	5.800	3	4.350	1.300
75%	6.400	3.300	5.100	1.800
max	7.900	4.400	6.900	2.500

Das Ergebnis ist natürlich wieder ein DataFrame (voller Statistiken)