

1

Einführung und Motivation

2

Grundlagen der Programmierung mit Python

3

Datenmanipulation und Datenanalyse mit pandas

4

Ausblick

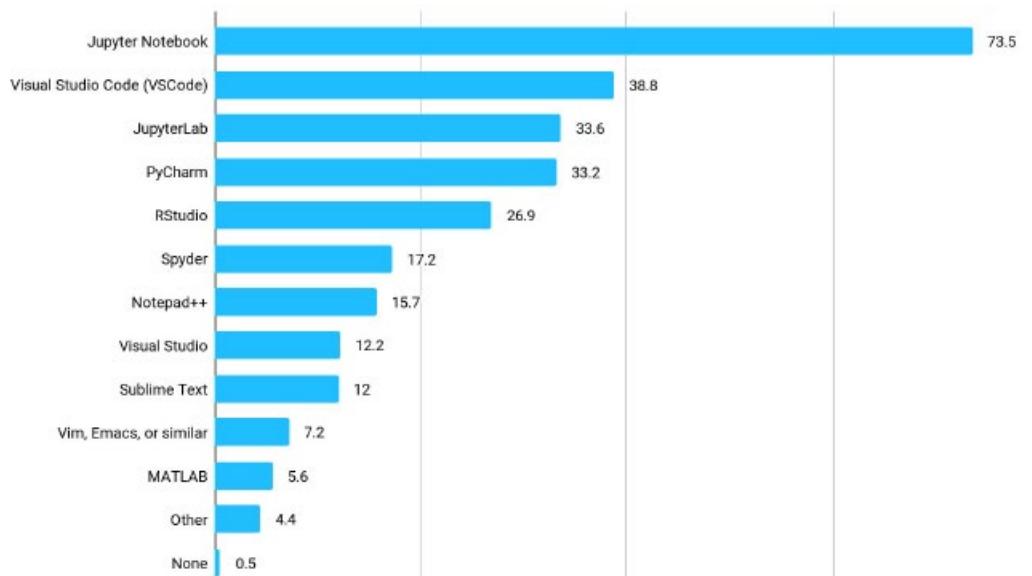
## 2 Grundlagen der Programmierung mit Python

### 2.1 Skriptsprachen für Data Science

*What programming languages  
do you use on a regular basis?*



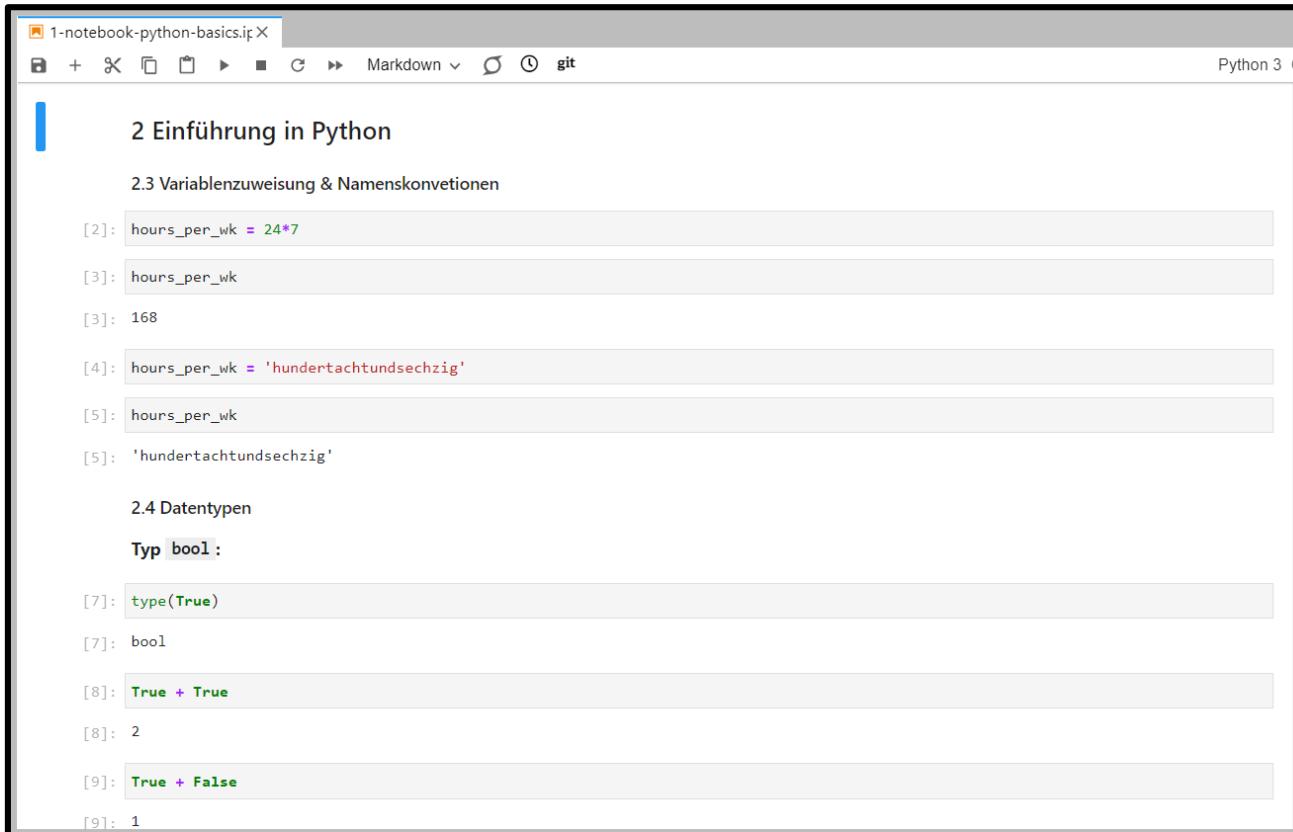
*Which of the integrated development environments  
(IDE's) do you use on a regular basis?*



Quelle: [2021 Kaggle Machine Learning & Data Science Survey](#)

# 2 Grundlagen der Programmierung mit Python

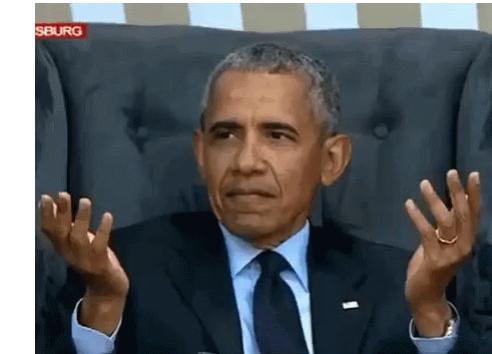
## 2.2 Das Jupyter-Notebook



The screenshot shows a Jupyter Notebook interface with the following content:

```
[2]: hours_per_wk = 24*7
[3]: hours_per_wk
[3]: 168
[4]: hours_per_wk = 'hundertachtundsechzig'
[5]: hours_per_wk
[5]: 'hundertachtundsechzig'

2.4 Datentypen
Typ bool:
[7]: type(True)
[7]: bool
[8]: True + True
[8]: 2
[9]: True + False
[9]: 1
```



Ein **Jupyter Notebook** ist ein interaktives Dokument, in dem Code, Code Output, Text, Grafiken/Plots und andere Elemente integriert werden können.

**Features:**  
Interactivity  
Iteration  
Sharing & Communication  
Transparency  
Reproducibility

## 2 Grundlagen der Programmierung mit Python

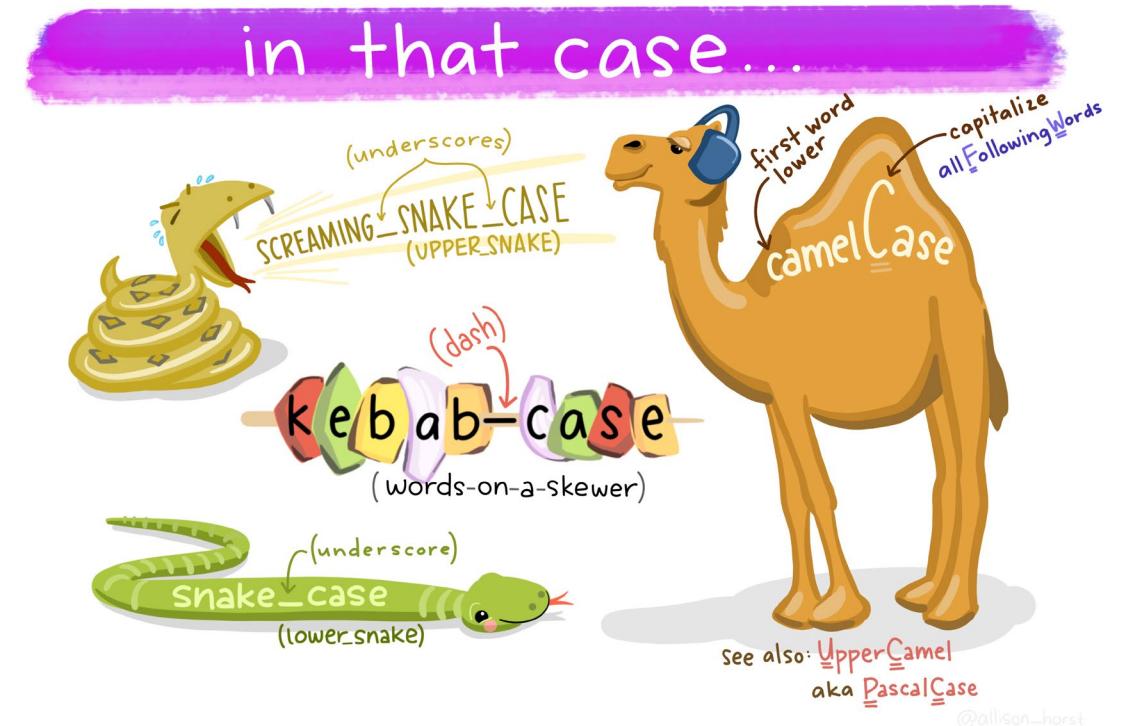
### 2.3 Variablenzuweisung & Namenkonventionen

**hours\_per\_wk = 24\*7**

Variablen-Name

Expression

- » Die Variablenzuweisung (ASSIGN STATEMENT) ordnet einem Wert (oder Objekt) einen Namen zu
- » Das „=“ dient als Zuweisungsoperator
- » Der Wert kann später über diesen Variablen-Namen abgerufen werden (REFERENCING)
- » Namen können durch erneute Zuweisung überschrieben werden
- » Namen sollten eindeutig und sprechend sein (Achtung: Leerzeichen können zu Problemen führen, Groß-/Kleinschreibung zählt!)

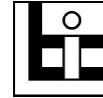


Quelle: [Alison Horst](#)

# 2 Grundlagen der Programmierung mit Python

## 2.4 Datentypen

---



Forschungsteam  
Berens

### Typ Boolean

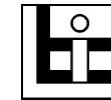
- » `bool`: Wahrheitswert (binär), entweder `True` oder `False`
- » Für Berechnungen mit Wahrheitswerten wird `True` (1) und `False` (0) angenommen.

### Typ Numerisch

- » `int`: Ganzzahl (besitzt niemals eine Nachkommastelle, z.B. 10 oder -10)
- » `float`: Dezimalzahl (besitzt immer mindestens eine Nachkommastelle, z.B. 9.98 oder 10.0)
- » Der Typ `float` besitzt limitierte Genauigkeit (~ 15-16 Nachkommastellen), danach kann es zu Rundungsdifferenzen kommen!

# 2 Grundlagen der Programmierung mit Python

## 2.4 Datentypen



### Typ String

» `str`: Ein *string* enthält eine beliebige Folge an Zeichen, die einen Text ergeben

» Strings sind umrahmt von Apostrophen: '`text`' oder "`text`"

» Achtung bei Vermischung von einfachen und doppelten Apostrophen:

`"Come to an end. I don't have a clue!"`

`"She said: 'This is a dead end!'"`

» Multi-line-strings können über mehrere Zeilen gehen:

```
s = """
```

```
This is a longer string that  
spans multiple lines
```

```
"""
```

» Auf die einzelnen Zeichen eines Strings kann mittels **SLICING** zugegriffen werden: `s[17:23]`

» Mehrere Strings können durch ein + zusammengefügt werden.

# 2 Grundlagen der Programmierung mit Python

## 2.4 Datentypen

### Slicing / Indexing

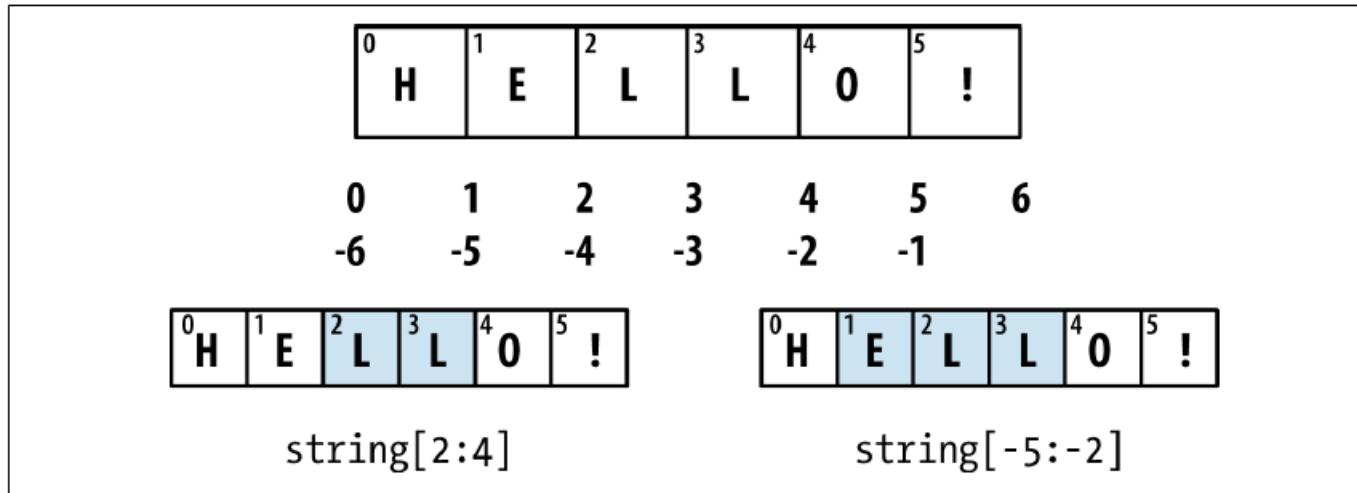


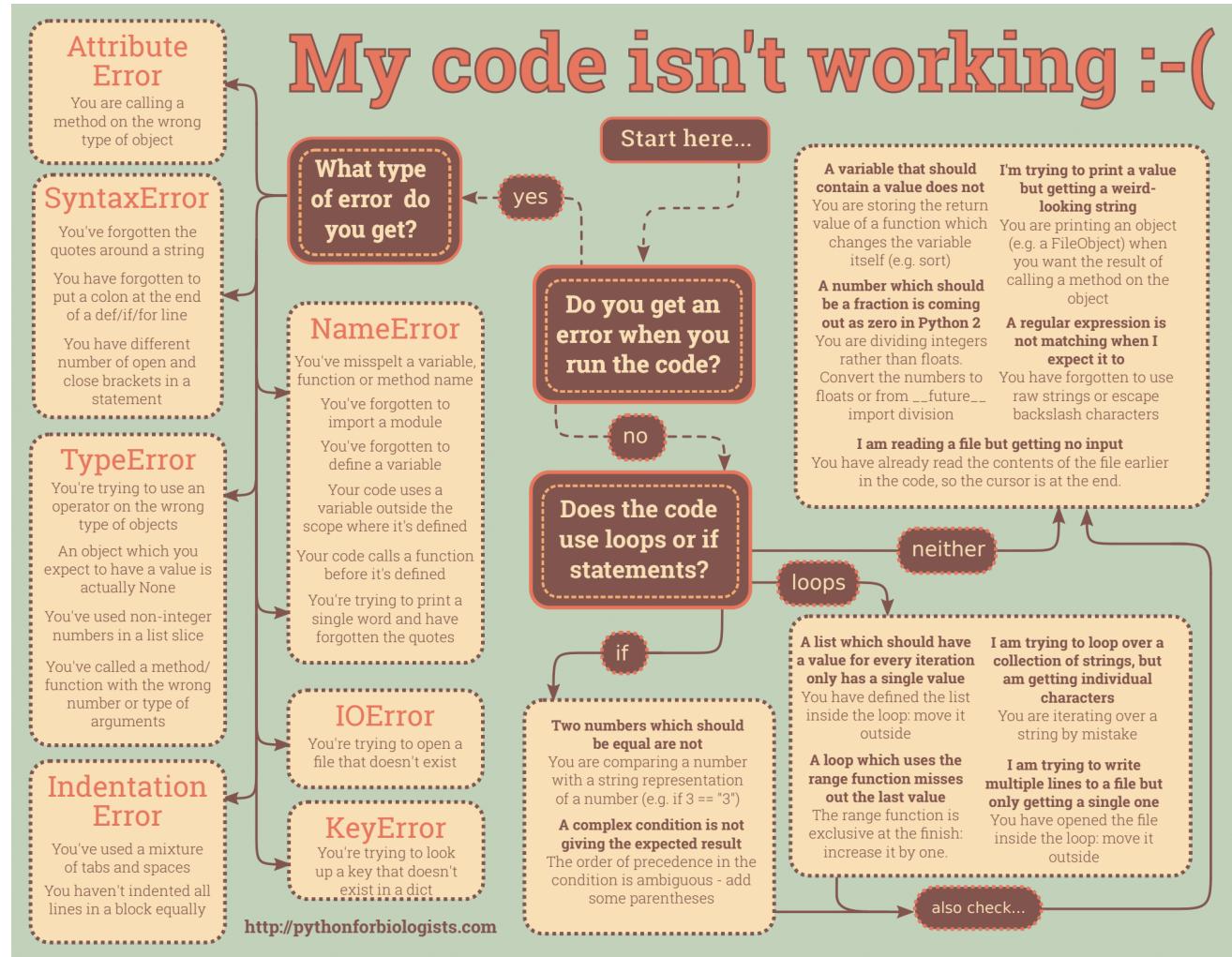
Figure 3-1. Illustration of Python slicing conventions

Zwei Stolpersteine bzw. Eigenheiten von Python:

- » Die Indizierung startet mit 0
- » Der Befehl `string[2:4]` gibt die Zeichen mit Index 2 und 3 zurück (die „obere Grenze“ ist exklusive)

## 2 Grundlagen der Programmierung mit Python

### Exkurs: Error Type Cheat Sheet

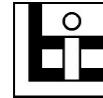


<http://pythonforbiologists.com>

Quelle:  
[pythonforbiologists.com](http://pythonforbiologists.com)

# 2 Grundlagen der Programmierung mit Python

## 2.4 Datentypen



### Typekonvertierung

- » Die Prüfung des Datentyps erfolgt mittels `type()`
- » Der Typ einer Variable hängt von dem Typ des zugewiesenen Wertes oder Objektes ab:

```
x = 2  
type(x)
```

- » Eine Typekonvertierung (TYPE CASTING) verändert den Datentyp eines Wertes:

- » String zu numerisch:

```
int('12')      float('1.2')
```

- » Numerisch zu String:

```
str(12)       str(1.2)
```

- » Numerisch zu numerisch:

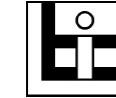
```
float(1)      int(1.2) (Achtung: Informationsverlust)
```

- » Numerisch zu boolean:

```
bool(1)       bool(0)       bool(5.4)
```

# 2 Grundlagen der Programmierung mit Python

## 2.5 Vergleichsoperatoren



Der Vergleich zweier Werte oder Variablen resultiert stets in einem Wahrheitswert (True oder False):

x = 1      ! Zuweisungsoperator

y = 5

x > 1

x >= 1      (äquivalent zu (x > 1) or (x == 1))

x > y

x == y      ! Vergleichsoperator

x != y

x+4 == y

x < 3 < y      (äquivalent zu (x < 3) and (y > 3))

1 + 0 + 1      == 2

sum([1, 0, 1])      == 2

sum([True, False, True])      == 2

## 2 Grundlagen der Programmierung mit Python

### Exkurs: CEO turnover and dismissal dataset



Received: 15 June 2020

Revised: 28 February 2021

Accepted: 1 March 2021

Published on: 17 March 2021

DOI: 10.1002/smj.3278

**RESEARCH ARTICLE**



**WILEY**

## A database of CEO turnover and dismissal in S&P 1500 firms, 2000–2018

Richard J. Gentry<sup>1</sup> | Joseph S. Harrison<sup>2</sup> |  
Timothy J. Quigley<sup>3</sup> | Steven Boivie<sup>4</sup>

<sup>1</sup>University of Mississippi, University, Mississippi

<sup>2</sup>Texas Christian University, Fort Worth, Texas

<sup>3</sup>University of Georgia, Athens, Georgia

<sup>4</sup>Texas A&M University, College Station, Texas

Quelle: [Gentry et al. \(2020\)](#), Link to [data](#)

## 2 Grundlagen der Programmierung mit Python

### Exkurs: CEO turnover and dismissal dataset (DATA DICTIONARY)

Variable	Type	Description
<i>dismissal_dataset_id</i>	int	The primary key. This will change from one version to the next. gvkey-year is also a unique identifier.
<i>coname</i>	str30	The Compustat Company Name.
<i>gvkey</i>	int	The Compustat Company identifier.
<i>fyear</i>	int	The fiscal year in which the event occurred.
<i>co_per_id</i>	int	The executive/company identifier from Execucomp.
<i>exec_fullname</i>	str50	The executive full name as listed in Execucomp.
<i>departure_code</i>	int	The departure reason coded from criteria above.
<i>ceo_dismissal</i>	bool	A dummy code for involuntary, non-health related turnover (Codes 3 & 4).
<i>interim_coceo</i>	str7	A descriptor of whether the CEO was listed as co-CEO or as an interim CEO (sometimes interim positions last a couple years).
<i>tenure_no_ceodb</i>	int	For CEOs who return, this value should capture whether this is the first or second time in office.
<i>max_tenure_ceodb</i>	int	For this CEO, how many times did s/he serve as CEO.
<i>fyear_gone</i>	int	An attempt to determine the fiscal year of the CEO's effective departure date. Occasionally, looking at departures on Execucomp does not agree with the leftofc date that we have. They apparently try to balance between the CEO serving one month in the fiscal year against documenting who was CEO on the date of record. I would stick to the Execucomp's fiscal year, departure indication for consistency with prior work.
<i>leftofc</i>	date	Left office of CEO, modified occasionally from Execucomp but same interpretation. The date of effective departure from the office of CEO.
<i>still_there</i>	str9	A date that indicates the last time we checked to see if the CEO was in office. <u>If no date, then it looks like the CEO is still in office but we are in the process of checking.</u>
<i>notes</i>	str	Long-form description and justification for the coding scheme assignment.
<i>sources</i>	str	URL(s) of relevant sources from internet or library sources.

## 2 Grundlagen der Programmierung mit Python

### Exkurs: CEO turnover and dismissal dataset (CODING SCHEME)

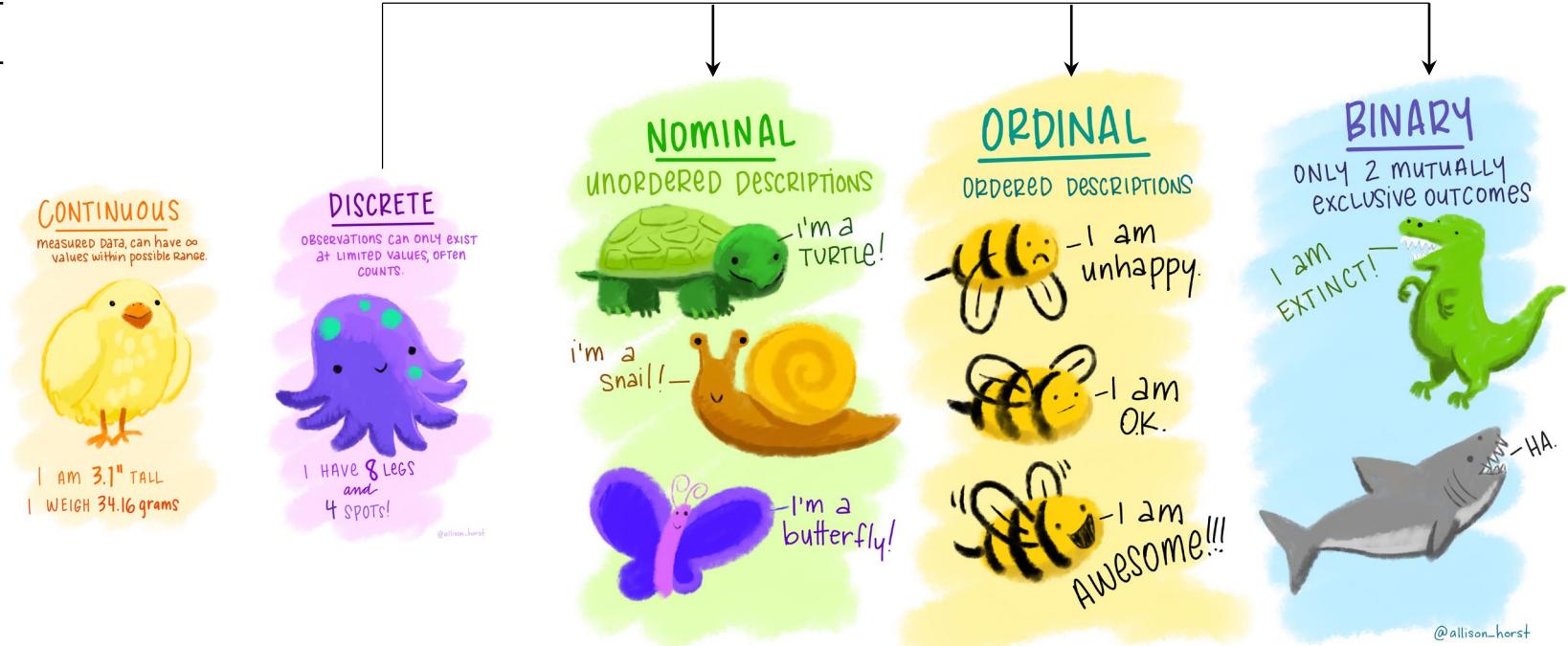
TABLE 2 CEO departure reasons and definitions

Code	Title	Brief description
1	Involuntary—CEO death	The CEO died while in office and did not have an opportunity to resign before health failed
2	Involuntary—CEO illness	Required announcement that the CEO was leaving for health concerns rather than removed during a health crisis
3	Involuntary—CEO dismissed for job performance	The CEO stepped down for reasons related to job performance. This included situations where the CEO was immediately terminated as well as when the CEO was given some transition period, but the media coverage was negative. Often the media cited financial performance or some other failing of CEO job performance (e.g., leadership deficiencies, innovation weaknesses, etc.)
4	Involuntary—CEO dismissed for personal issues	The CEO was terminated for behavioral or policy-related problems. The CEO's departure was almost always immediate, and the announcement cited an instance where the CEO violated company HR policy, expense account cheating, and so forth
5	Voluntary—CEO retired	Voluntary retirement based on how the turnover was reported in the media. Here, the departure did not sound forced, and the CEO often had a voice or comment in the succession announcement. Media coverage of voluntary turnover was more valedictory than critical. Firms use different mandatory retirement ages, so we could not use 65 or older and facing mandatory retirement as a cut off. We examined coverage around the event and subsequent coverage of the CEO's career when it sounded unclear
6	Voluntary—New opportunity	The CEO left to pursue a new venture or to work at another company. This frequently occurred in startup firms and for founders
7	Other	Interim CEOs, CEO departure following a merger or acquisition, company ceased to exist, company changed key identifiers so it is not an actual turnover, and CEO may or may not have taken over the new company
8	Missing	Despite attempts to collect information, there was not sufficient data to assign a code to the turnover event. These will remain the subject of further investigation and expansion

## 2 Grundlagen der Programmierung mit Python

### Exkurs: Datentypen & Variablen

Variable name	Type
dismissal_dataset_id	int
coname	str30
gvkey	int
fyear	int
co_per_rol	int
exec_fullname	str50
departure_code	int
ceo_dismissal	bool
interim_coceo	str7
tenure_no_ceodb	int
max_tenure_ceodb	int
fyear_gone	int
leftofc	date
still_there	str9
notes	str
sources	str



Quelle: [Alison Horst](#)

- » Jede Spalte einer Tabelle sollte Werte vom gleichen Datentyp enthalten.
- » Stetige Variablen sind immer numerisch. Diskrete Variable können vom Typ `int`, `str` oder `bool` sein.
- » Der Typ `date` ist ein Spezialfall. Er ist numerisch, aber speziell dargestellt (Stichwort: [unixtime](#)).

# 2 Grundlagen der Programmierung mit Python

## 2.6 Ausgewählte Datenstrukturen in Python



### Sequenzen

» `list`: [5, 7, 28, -5, 'python', 'data', 'python', True]

» Eine `list` ist eine ungeordnete Sequenz an Elementen (eine Art Aufzählung).

» `tuple`: (25, '2020'), ((True, True, False), (23, 'python'))

» Ein `tuple` ist eine Sequenz an Elementen, getrennt durch ein Komma (eine Art Koordinate).

» Im Gegensatz zu einer `list` kann ein `tuple` nicht modifiziert werden!

### Sets

» `set`: {'ceo\_death', 'ceo\_illness', 'ceo\_dismissed', ... , 'missing'}

» Ein `set` ist eine ungeordnete Sequenz an einzigartigen Elementen (eine Aufzählung ohne Duplikate).

### Mappings

» `dict`: {1: 'ceo\_death', 2: 'ceo\_illness', ... , 8: 'missing'}

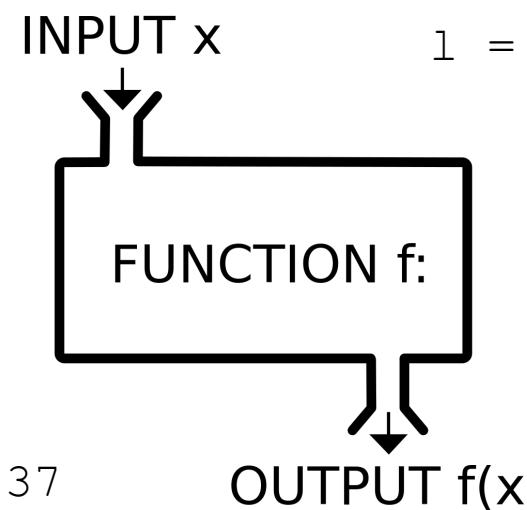
» Ein `dict` enthält sog. *key-value-pairs*, wobei Keys einzigartig sein müssen (eine Art Nachschlagewerk).

# 2 Grundlagen der Programmierung mit Python

## 2.7 Funktionen

"Hello World, it's  
beautiful out here!"

`len()`



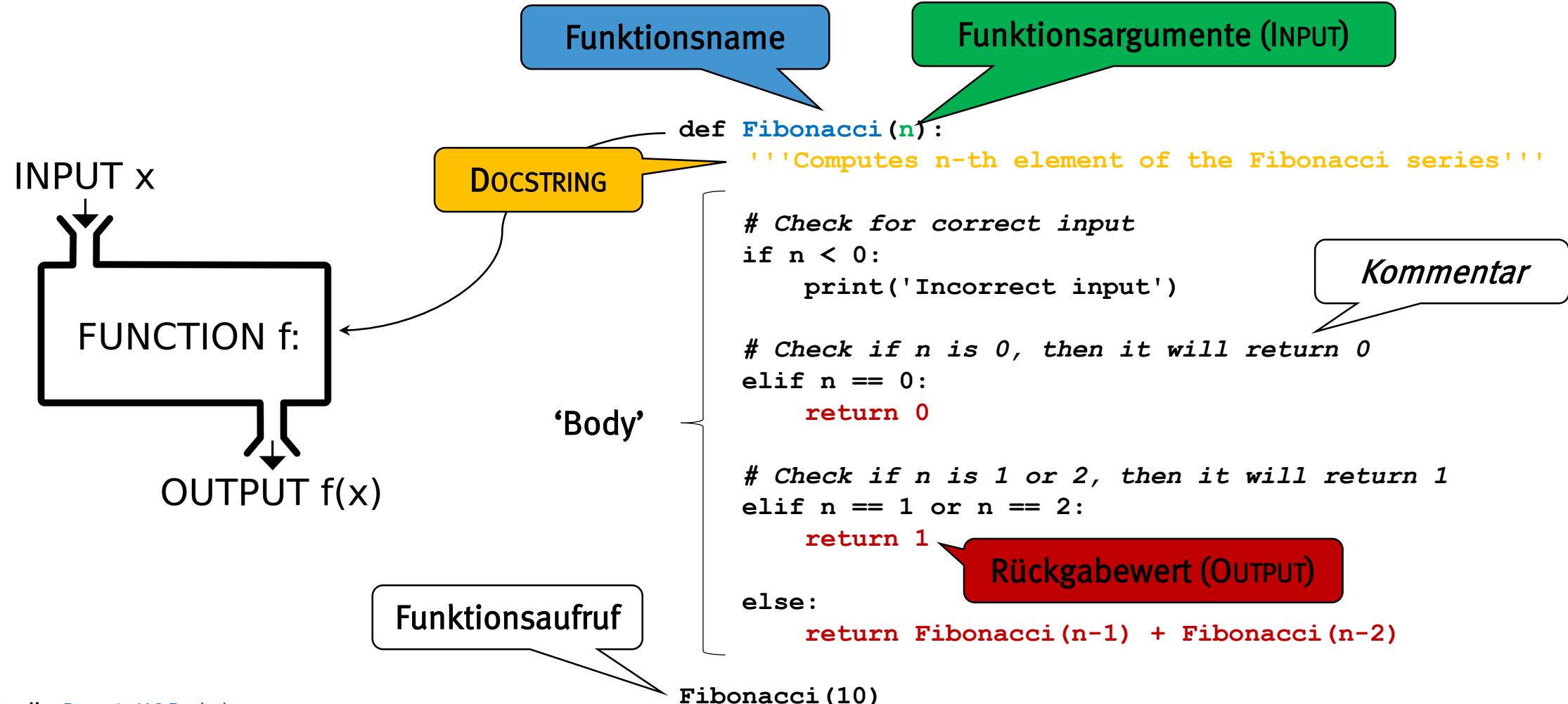
`l = [13, 24, 6, 7, -15, 59]`

`l.sort()`

`[-15, 6, 7, 13, 24, 59]`

# 2 Grundlagen der Programmierung mit Python

## 2.7 Funktionen



Quelle: [Data 8, UC Berkeley](#)

# 2 Grundlagen der Programmierung mit Python

## 2.7 Funktionen



```
def Funktionsname(arg1, arg2, arg3=5, arg4='mean', ...):
    '''Docstring'''
    ...
    Body
    ...
    return Rückgabewert
```

The code snippet illustrates the structure of a Python function definition. It starts with `def`, followed by the function name `Funktionsname`. The arguments are listed as `arg1, arg2, arg3=5, arg4='mean', ...`. A docstring is enclosed in triple quotes. The body of the function follows, containing ellipses and the `return` statement with the value `Rückgabewert`.

- » Funktionen erlauben eigene Funktionalitäten in Python zu integrieren.
- » Funktionen erleichtern es Programmcode zu strukturieren. Wenn wiederholt der gleiche Code ausgeführt wird, sollte dieser in einer Funktion gesammelt werden (Rule-of-thumb:  $n \geq 2$ ).
- » Eine Funktion kann eine beliebige Anzahl an Funktionsargumenten besitzen. Zunächst werden Positionale Argumente (`args`), anschließend Schlüssel-Wert Argumente (`kwargs`) eingegeben.
- » Argumente können *default*-Werte besitzen (z.B. `arg3=5`) und müssen so nicht vom User gesetzt werden (der Regelfall für `kwargs`).
- » Python unterstützt sogenannte LAMBDA FUNCTIONS: `Funktionsname = lambda x: x / 5`



**Übungsaufgaben:**  
30 Minuten



1

Einführung und Motivation

2

Grundlagen der Programmierung mit Python

3

Datenmanipulation und Datenanalyse mit pandas

4

Ausblick

# 3 Datenmanipulation und Datenanalyse mit pandas

## 3.1 Datenmodalitäten: Tabellarische Daten (TABULAR DATA)

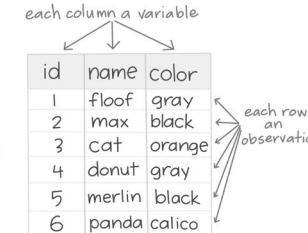
Quelle: [Alison Horst](#)

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

dismissal_dataset_id	coname	Variable	gvkey	year	co_per_role	exec_fullname	Departure Code	ceo_dismissal
1	AAR CORP	Observation	1004	1995	5622	Ira A. Eichner	5	0
2	AAR CORP		1004	2017	5623	David P. Storch	5	0
3	AAR CORP		1004	2018	51547	John McClain Holmes, III		
4	ADC TELECOMMUNICATIONS INC		1013	2000	2611	William J. Cadogan	6	
5	ADC TELECOMMUNICATIONS INC		1013	2003	23275	Richard R. Roscitt	6	0
6	ADC TELECOMMUNICATIONS INC		1013	2010	8741	Robert E. Switz	7	0
7	ALPHARMA INC -CLA		1034	1993	5628	Jeffrey E. Smith	5	0
...	...		...	...	...	...	...	...

Missing Value  
(NA, NaN, None)

Measurement

# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Verschiedene Arten von Missing Values (NAs)

**Strukturelle NAs:** Fehlende Werte sind strukturell motiviert

- » z.B. Anzahl der Geburten pro Mann, Berufserfahrung in Jahren für Kinder
- » z.B. Verwaltungs- und Vertriebskosten von Unternehmen, die nach GKV bilanzieren

**Zufällige NAs :** Fehlende Werte treten vollkommen zufällig auf

- » z.B. durch Unkonzentriertheiten bei der Datenerfassung, kurzzeitiger Ausfall von Sensoren, Abbruch der Internetverbindung

**Nicht zufällige NAs :** Fehlende Werte hängen von der untersuchten Variable oder anderen Variablen im Datensatz ab

- » z.B. Produktrezensionen: Unzufriedene Kund:innen neigen dazu, eher eine Rezension zu schreiben (missing values bei zufriedenen Kund:innen)
- » z.B. Rauchverhalten: Proband:innen neigen dazu, gesellschaftlich erwünschtes Verhalten zu berichten (DESIRABILITY BIAS) (missing values bei Raucher:innen)

**Explizite NAs:** Vorliegen eines Missing Values wird explizit berücksichtigt

Unternehmen	Jahr	Umsatz
A	2020	120
B	2020	NA
B	2021	292

**Implizite NAs:** Vorliegen eines Missing Values ergibt sich implizit aus Datenstruktur

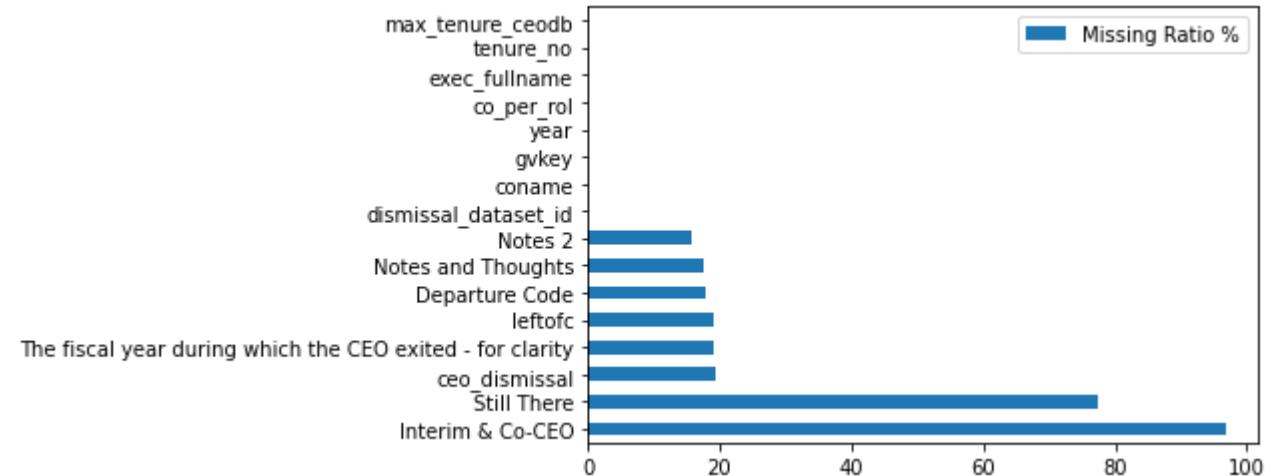
Unternehmen	Jahr	Umsatz
A	2020	120
B	2020	256
B	2021	292

# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Analyse von Missing Values

### Gängige Notationen von Missing Values:

- » Leere Zellen (bzw. leere Strings “ ”)
- » “NA”, “N/A”, “NaN” oder “-9999”
- » “oooooo” (z.B. verschiedene SAP-Systeme)

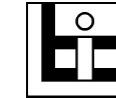


### Umgang mit Missing Values:

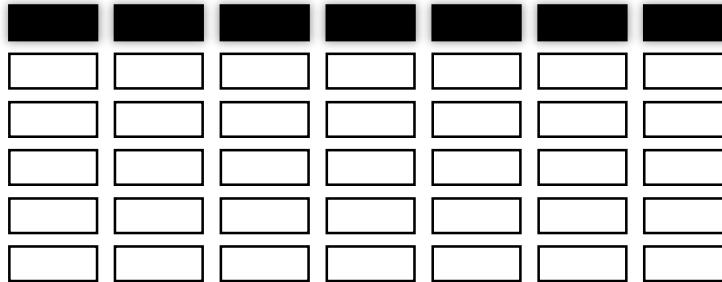
- » Entfernen der betroffenen Zeilen (sobald ein NA vorliegt)
- » Entfernen der betroffenen Variablen/Spalten (bei hohem Missing Ratio)
- » IMPUTATION (d.h. Ersetzen von Missing Values)

# 3 Datenmanipulation und Datenanalyse mit pandas

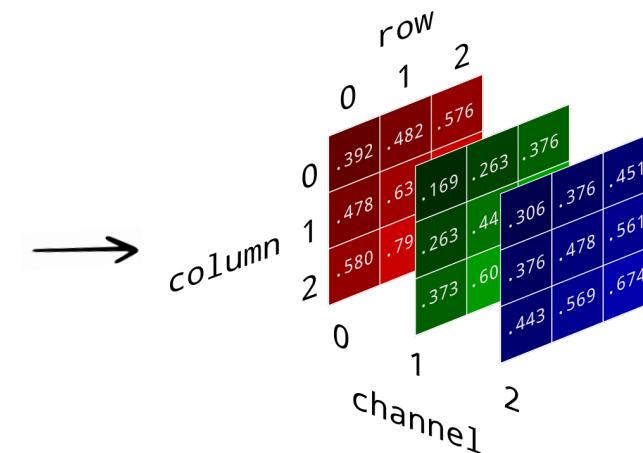
## 3.1 Datenmodalitäten: Text, Bild, Audio



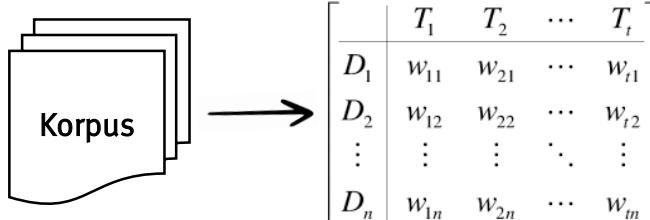
### Tabellarische Daten



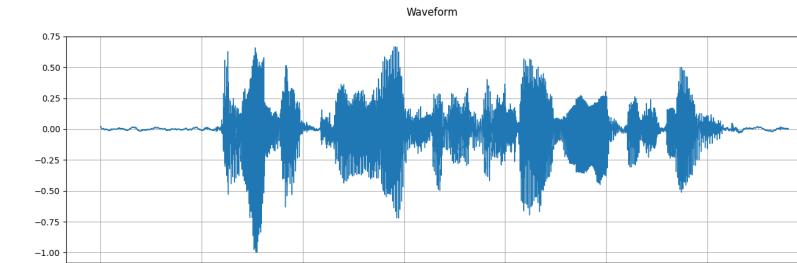
### Bild Daten



### Text Daten



### Audio Daten



Innerhalb dieser verschiedenen **Modalitäten** unterscheiden wir zwischen **strukturierten Daten** (insb. tabellarische Daten) und **unstrukturierten Daten** (z.B. Text, Bild, Audio, Video).

### 3 Datenmanipulation und Datenanalyse mit pandas

#### Exkurs: Module/Packages/Libraries

- » Die standardmäßige Python Installation bringt eine Vielzahl an eingebauten Funktionen mit, die sog. [Python Standard Library](#).
- » Programmiersprachen lassen sich durch verschiedene Bibliotheken (auch *packages*, *libraries* oder *modules* genannt) erweitern. Jede Bibliothek bringt Funktionen für bestimmte Zwecke mit (z.B. pandas für die Arbeit mit tabellarischen Daten, spacy für Textverarbeitung oder pytorch für Deep Learning).
- » Bibliotheken werden zu Beginn eines Python-Programmes geladen:

```
import <Modulname> (as <Alias>)  
(Import des gesamten Moduls)
```

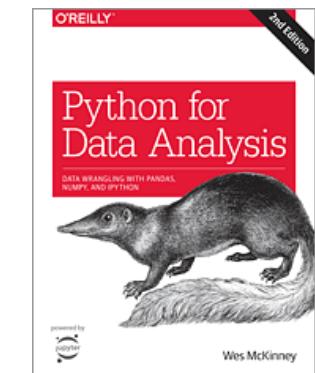
**Bsp:** import pandas as pd

```
from <Modulname> import <Funktion/Objekt>  
(Import einzelner Funktionen oder Objekte)
```

**Bsp:** from pandas import DataFrame



Von: [Wes McKinney](#)



# 3 Datenmanipulation und Datenanalyse mit pandas

## 3.2 pandas Datenstrukturen



**pd.Series()**: Eine 1D-Sequenz an Elementen (*values*) mit zugehörigen labels (*index*). Sie wird verwendet, um eine Spalte (d.h. Variable) einer Tabelle abzubilden.

```
data = [1995, 2017, 2018, 2000]  
pd.Series(data, name='year')
```

Index                          Values  
0    1995  
1    2017  
2    2018  
3    2000  
Name: year, dtype: int64

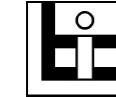
**pd.DataFrame()**: Eine 2D-Datentabelle, bestehend aus einer Folge von Spalten, die jeweils einen eigenen Datentyp (z.B. int, float, bool, str) aber den gleichen Index besitzen.

```
data = {'year': [1995, 2017, 2018, 2000],  
       'exec_fullname': ['Ira A. Eichner', 'David P. Storch',  
                         'John McClain Holmes, III', 'William J. Cadogan'],  
       'Departure Code': [5, 5, None, 6]}  
  
pd.DataFrame(data)
```

	year	exec_fullname	Departure Code
0	1995	Ira A. Eichner	5.0
1	2017	David P. Storch	5.0
2	2018	John McClain Holmes, III	NaN
3	2000	William J. Cadogan	6.0

# 3 Datenmanipulation und Datenanalyse mit pandas

## 3.3 Importieren von Daten



```
url = 'https://zenodo.org/.../CEO%20Dismissal%20Data%202021.02.03.xlsx?download=1'  
df = pd.read_excel(url, 'Sheet1', engine='openpyxl')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 9390 entries, 0 to 9389  
Data columns (total 16 columns):  
 #   Column           Non-Null Count  Dtype     
 ---    
 0   dismissal_dataset_id  9390 non-null   int64    
 1   coname            9390 non-null   object    
 2   gvkey             9390 non-null   int64    
 3   year              9390 non-null   int64    
 4   co_per_rol        9390 non-null   int64    
 5   exec_fullname    9390 non-null   object    
 6   Departure Code   7721 non-null   float64   
 7   ceo_dismissal    7578 non-null   float64   
 8   Interim & Co-CEO 294 non-null   object    
 9   tenure_no         9390 non-null   int64    
 10  max_tenure_ceodb 9390 non-null   int64    
 11  The fiscal year during which the CEO exited - for clarity 7587 non-null   float64   
 12  leftofc           7587 non-null   object    
 13  Still There       2123 non-null   object    
 14  Notes and Thoughts 7743 non-null   object    
 15  Notes 2            7910 non-null   object    
dtypes: float64(3), int64(6), object(7)  
memory usage: 1.1+ MB
```

### Weitere pd. read-Funktionen:

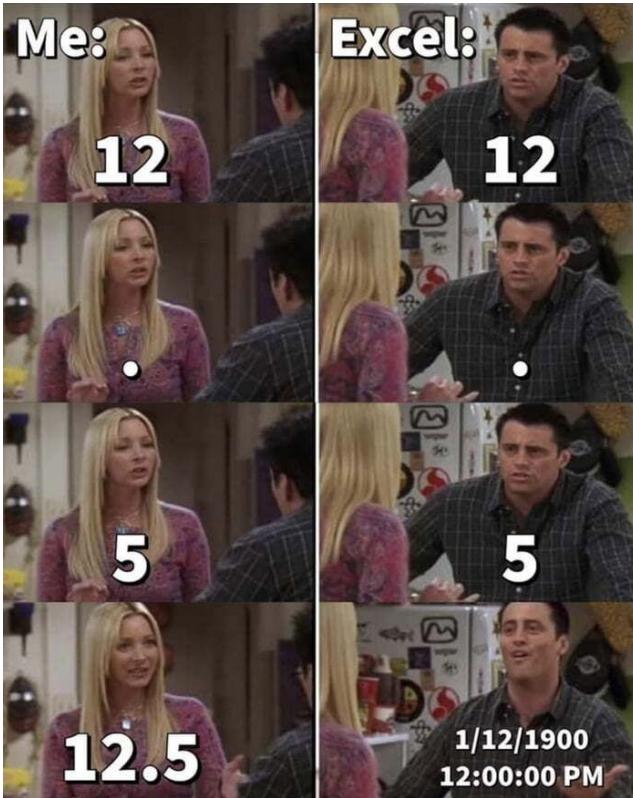
- » `read_csv`: comma-/semicolon-delimited
- » `read_table`: general delimited files
- » `read_excel`, `read_json`, `read_html`, etc.

Alle `read`-Funktionen besitzen auch zugehörige `write`-Funktionen zum Speichern von Daten (z.B. `to_csv` oder `to_json`).

# 3 Datenmanipulation und Datenanalyse mit pandas

## 3.3 Importieren von Daten

Achtung: CSV/Excel-Dateien speichern Datentypen nicht! Hier gibt es potenziell andere Speicherformate, die Abhilfe schaffen, z.B. HDF5 oder feather.



Hilfe? pandas besitzt eine sehr, sehr umfangreiche [Dokumentation](#).

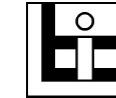
### pandas.read\_csv

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None,  
header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=None,  
prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None, engine=None,  
converters=None, true_values=None, false_values=None, skipinitialspace=False,  
skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True,  
na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=None,  
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False,  
cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None,  
decimal=',', lineterminator=None, quotechar='"', quoting=0, doublequote=True,  
escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None,  
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,  
low_memory=True, memory_map=False, float_precision=None, storage_options=None) [source]
```

Neben der Erklärung von Funktionsargumenten sind häufig auch Anwendungsbeispiele aufgeführt.

# 3 Datenmanipulation und Datenanalyse

## 3.4 Operationen auf Zeilen



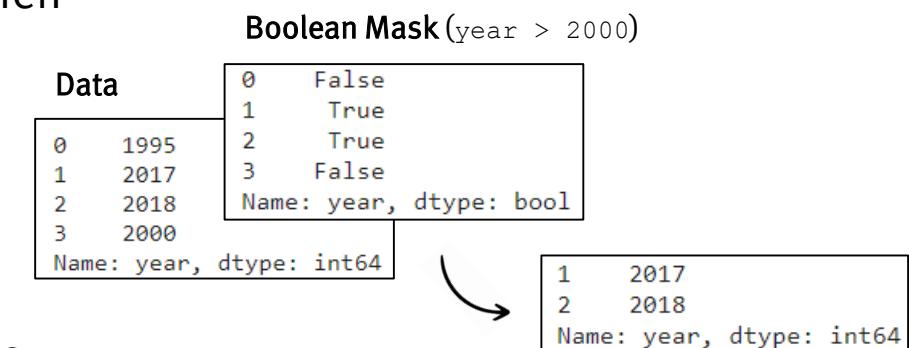
### Indexing / Slicing: Filtern auf Basis eines Indexes

- » `df.iloc[i]` gibt die  $i$ -te Zeile (d.h. Beobachtung) des Datensatzes zurück
- » `df.loc['label']` gibt alle Zeilen mit dem Zeilenindex `label` zurück
- » `df.drop('label', axis=0)` entfernt alle Zeilen mit dem Zeilenindex `label` \*

\* Zeilen: `axis=0` oder 'rows'  
Spalten: `axis=1` oder 'columns'

### Filtern (BOOLEAN MASKING): Filtern auf Basis von Vergleichsoperationen

- » `df[mask]` gibt alle Zeilen zurück, für die ein Vergleich `True` ergibt



### Sortieren: Veränderung der Reihenfolge

- » `df.sort_index(axis=0)` sortiert Zeilen anhand des Indexes
- » `df.sort_index(axis=0, ascending=False)` sortiert Zeilen in absteigender Reihenfolge
- » `df.sort_values(by='column')` sortiert Zeilen anhand der Spalte `column`

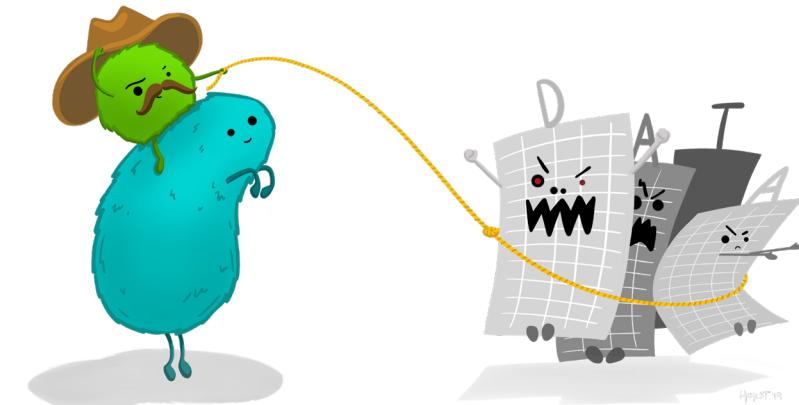


### Sampling: Generierung einer Stichprobe

- » `df.head(n)` und `df.tail(n)` gibt die ersten bzw. letzten n Zeilen des Datensatzes aus (DETERMINISTIC SAMPLING)
- » `df.sample(n, frac, replace=False)` gibt n zufällige Zeilen des Datensatzes aus (RANDOM SAMPLING)

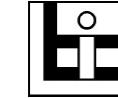
### Transformieren: Veränderung bestehender Zeilen

- » `df.dropna(axis=0, how='any')` entfernt Zeilen, die mindestens ein NA enthalten
- » `df.fillna({'col1': val1, 'col2': val2})` ersetzt NA Werte in Spalte col1 und col2
- » `df.replace(to_replace, value)` ersetzt beliebigen Wert (`to_replace`) durch `value`



# 3 Datenmanipulation und Datenanalyse

## 3.4 Operationen auf Zeilen: Übungsaufgaben



Forschungsteam  
Berens

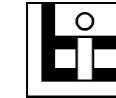
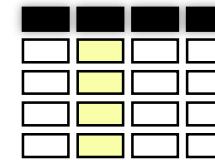


**Übungsaufgaben:**  
30 Minuten



# 3 Datenmanipulation und Datenanalyse

## 3.5 Operationen auf Spalten



### Indexing / Slicing: Filtern auf Basis eines Indexes

- » `df.iloc[:, i]` gibt die `i`-te Spalte des Datensatzes zurück (+ alle Zeilen)
- » `df.loc[:, 'col_name']` gibt die Spalte `col_name` zurück (+ alle Zeilen)
  - ↳ Best practice: `df['col_name']`
- » `df.drop('col_name', axis=1)` entfernt Spalte `col_name`\*

\* Zeilen: `axis=0` oder `'rows'`  
Spalten: `axis=1` oder `'columns'`

### Umbenennen: Veränderung von Spalten- bzw. Variablen-Namen

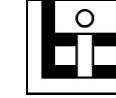
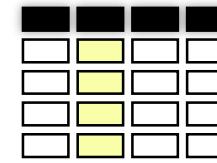
- » `df.rename(columns={'old_name': 'new_name'})`

### Sortieren: Veränderung der Reihenfolge

- » `df.reindex(columns=['col3', 'col1', 'col2'])` (Auflistung aller Spalten nötig)
- » `df[['col_name3', 'col_name1', 'col_name2']]` (Auflistung ausgewählter Spalten möglich)

# 3 Datenmanipulation und Datenanalyse

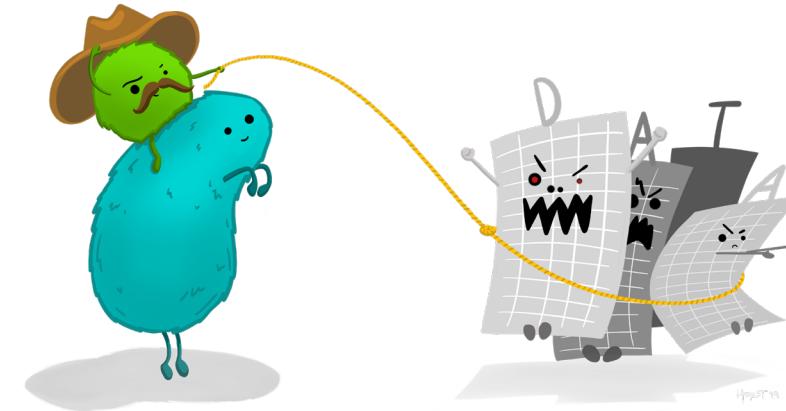
## 3.5 Operationen auf Spalten



Forschungsteam  
Berens

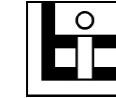
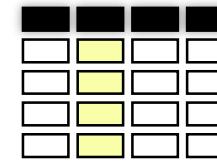
**Transformieren:** Erschaffung neuer Spalten oder Veränderung bestehender Spalten

- » `df['col_name'] = pd.Series([data])`
- » Sofern `col_name` noch nicht existiert, wird eine neue Spalte an das Ende des DataFrame angehangen.
- » Existiert `col_name` bereits, so wird die bestehende Spalte überschrieben.



# 3 Datenmanipulation und Datenanalyse

## 3.5 Operationen auf Spalten: Übungsaufgaben



Forschungsteam  
Berens

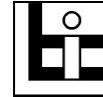
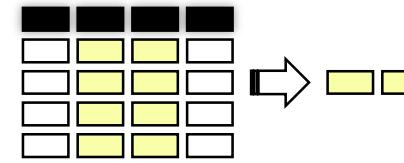


Übungsaufgaben:  
30 Minuten



# 3 Datenmanipulation und Datenanalyse

## 3.6 Datenaggregation



**Aggregation:** Reduktion von Spalten in deskriptive Statistiken (SUMMARY STATISTICS)

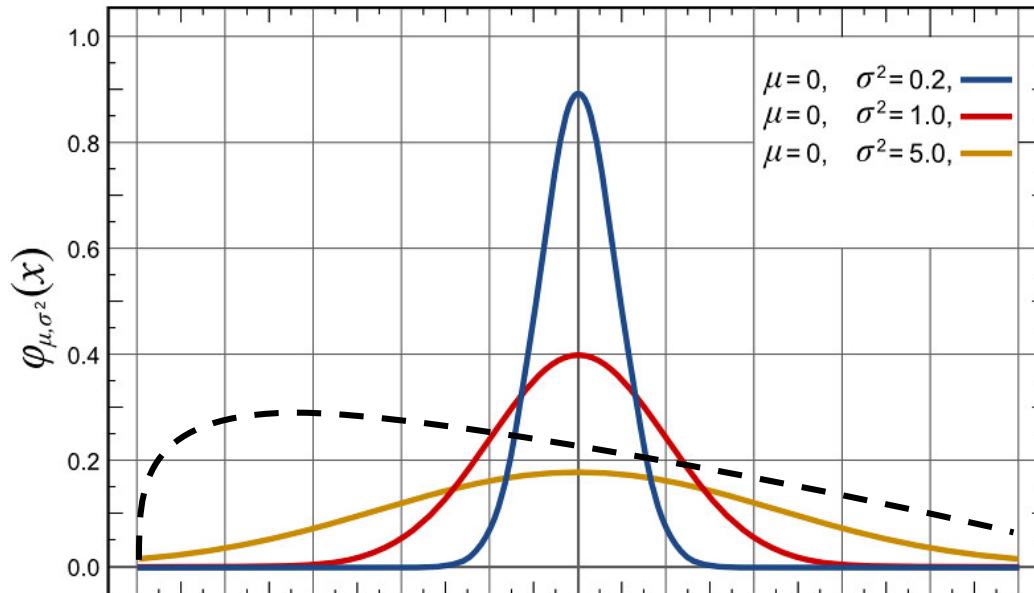
- » Berechnung von **univariaten Statistiken** via `value_counts()`, `sum()`, `cumsum()`, `mean()`, `median()`, `var()`, `quantile()`, etc.
  - » Anwendung auf einzelne Spalten: `df['col1'].sum()`
  - » Anwendung auf mehrere Spalten: `df[['col1', 'col2']].sum()`
- » Berechnung von **bivariaten Statistiken** via `correlate()`
  - » Paarweise Korrelationen zwischen Variablen: `df['col1'].corr(df['col2'])`
  - » Korrelationsmatrix für den gesamten Datensatz: `df.corr(numeric_only=True)`
- » Ermittlung von **einzigartigen Werten** je Spalte
  - » Einzigartige Werte: `df['col'].unique()`
  - » Einzigartige Werte, inkl. Anzahl: `df['col'].value_counts()`
- » pandas erlaubt uns sogar schnell eine Gesamtübersicht aller deskriptiven Statistiken zu erstellen via `df.describe()` (Achtung: Output der Funktion hängt vom `dtype` der jeweiligen Spalte ab!)

### 3 Datenmanipulation und Datenanalyse mit pandas

#### Exkurs: The Flaw of Averages

“The average European drinks 1L of beer per day.”

Quelle: [School of Data](#)



Alle vier Verteilungen weisen den gleichen Mittelwert auf. Aber welche Verteilung bildet die obige Aussage am besten ab?

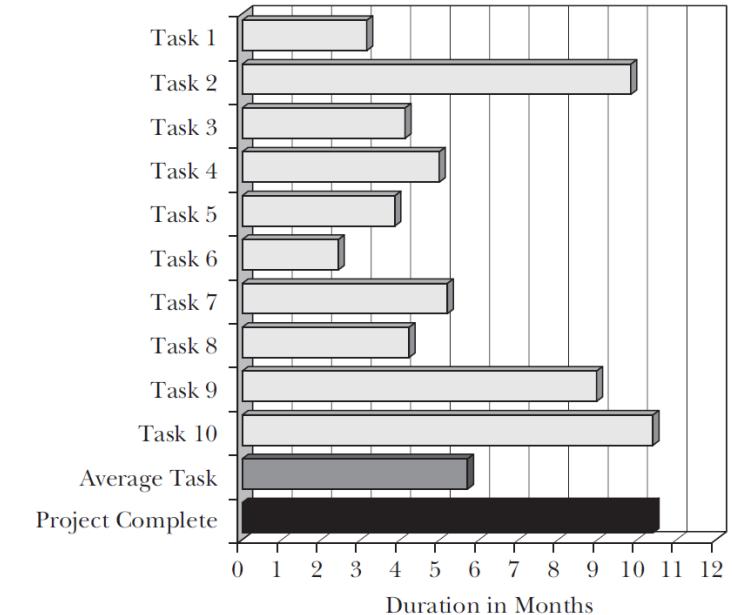
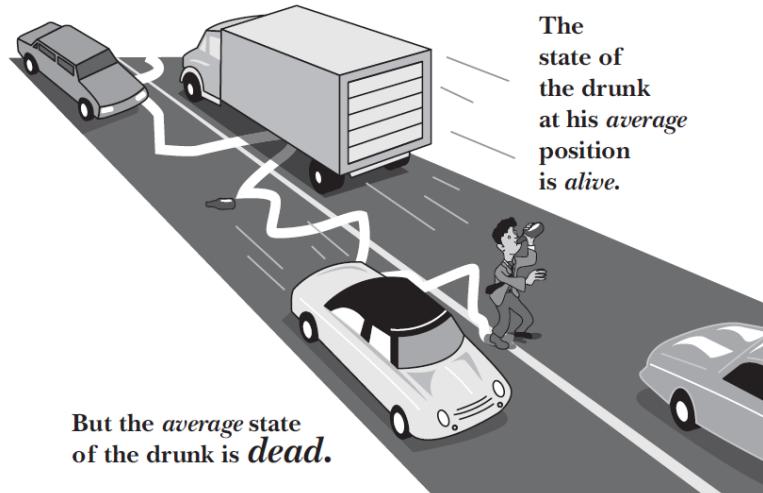
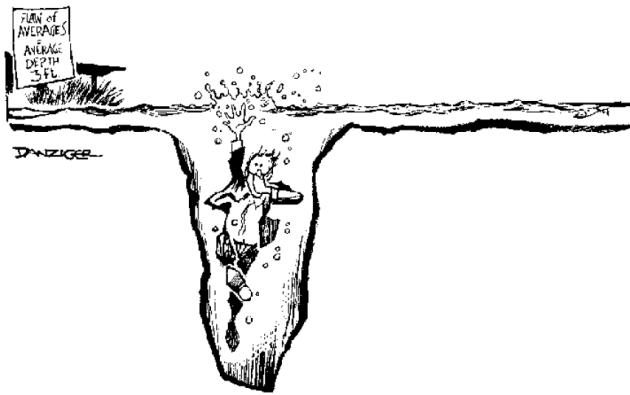
Hypothetische  
Verteilungen Europäischer  
Bierkonsument:innen

- » Die Zusammenfassung einer Verteilung in einer einzelnen Kennzahl führt zu einem starken **Informationsverlust** und **ignoriert Heterogenität!**
- » Eine Kennzahl sollte niemals ohne den zugehörigen **Kontext** (z.B. Median oder Varianz) berichtet und interpretiert werden!

### 3 Datenmanipulation und Datenanalyse mit pandas

#### Exkurs: The Flaw of Averages

**The Flaw of Average:** “Plans based on *average* assumptions are wrong on *average*.”

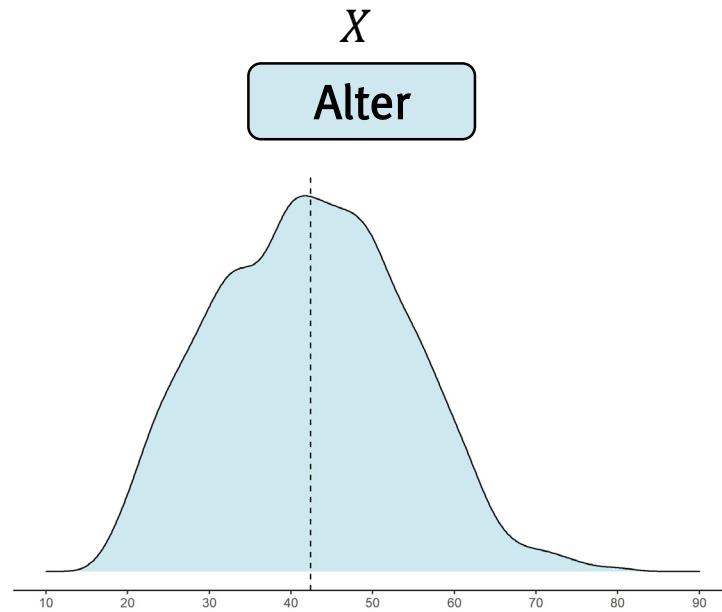


Quelle: Savage, S. L. *The Flaw of Averages. Why We Underestimate Risk in the Face of Uncertainty*. Hoboken: John Wiley & Sons, Inc., 2009.

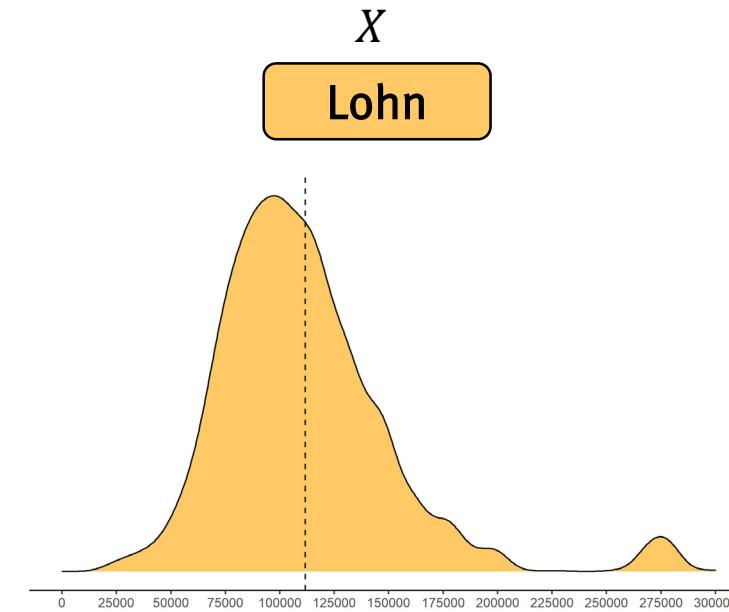
# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Standardisierung

Die Standardisierung macht Variablen, die auf unterschiedlichen Skalen gemessen werden, vergleichbar.



» gemessen in Jahren



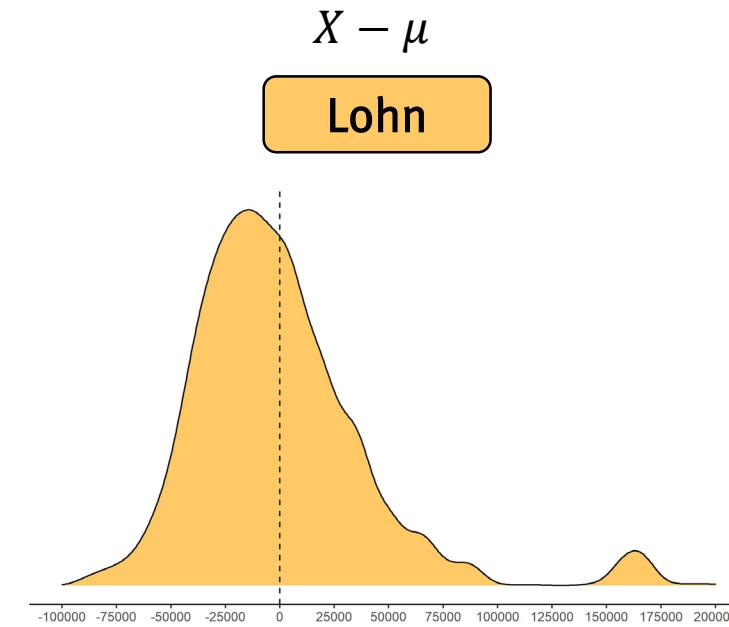
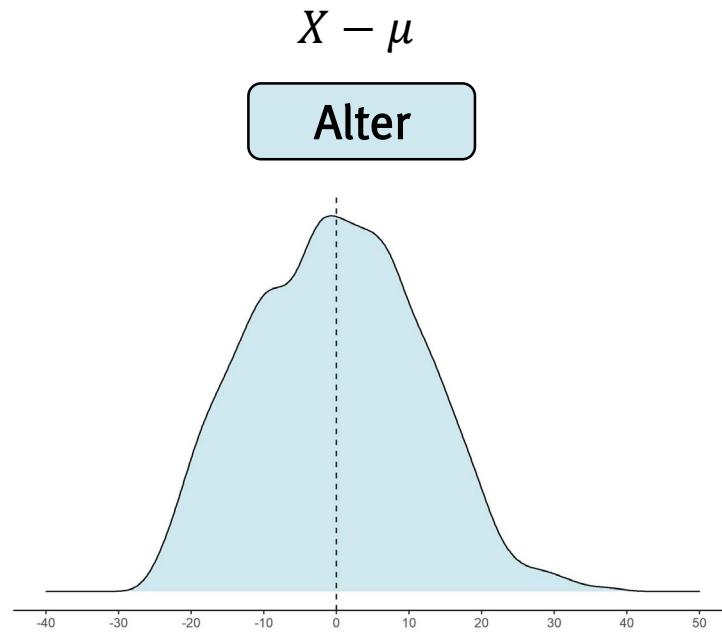
» gemessen in Euro

# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Standardisierung

Die Standardisierung macht Variablen, die auf unterschiedlichen Skalen gemessen werden, vergleichbar.

1. Schritt: Subtraktion des  
Mittelwertes  
(CENTERING)



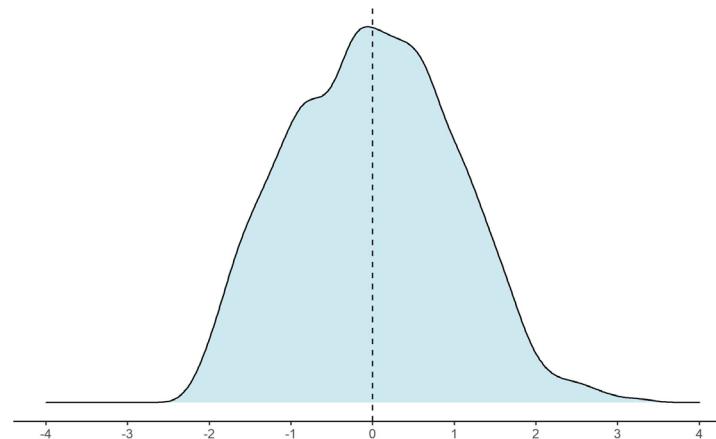
# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Standardisierung

Die Standardisierung macht Variablen, die auf unterschiedlichen Skalen gemessen werden, vergleichbar.

$$\frac{X - \mu}{\sigma}$$

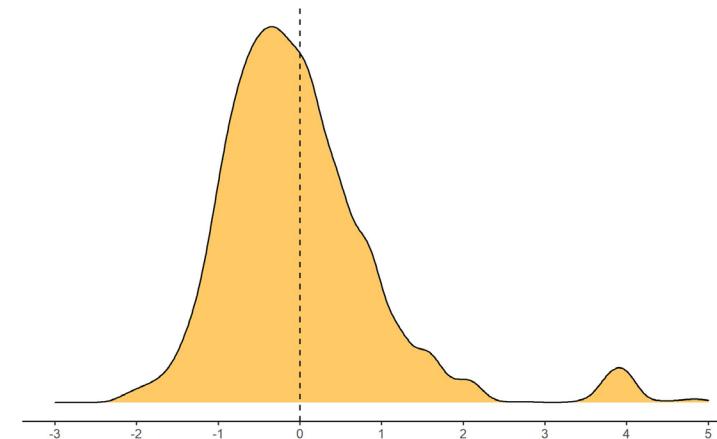
Alter



2. Schritt: Teilen durch die Standardabweichung  
(SCALING)

$$\frac{X - \mu}{\sigma}$$

Lohn

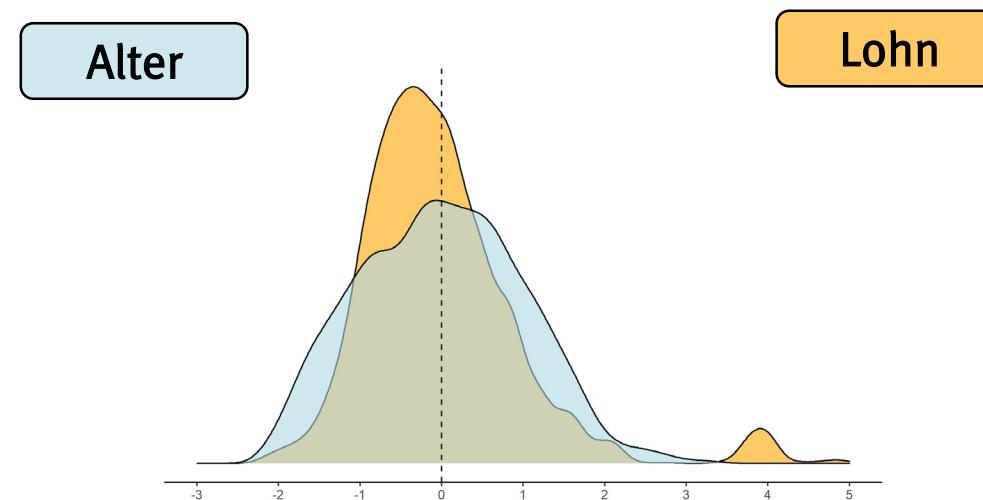


# 3 Datenmanipulation und Datenanalyse mit pandas

## Exkurs: Standardisierung

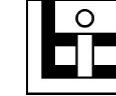
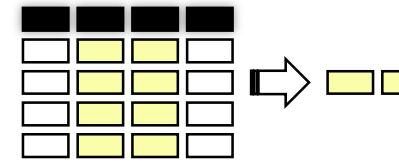
Die Standardisierung macht Variablen, die auf unterschiedlichen Skalen gemessen werden, vergleichbar:

$$Z = \frac{X - \mu}{\sigma}$$



# 3 Datenmanipulation und Datenanalyse

## 3.6 Datenaggregation: Übungsaufgaben



Forschungsteam  
Berens

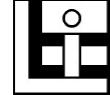
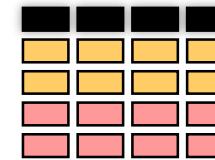


Übungsaufgaben:  
20 Minuten

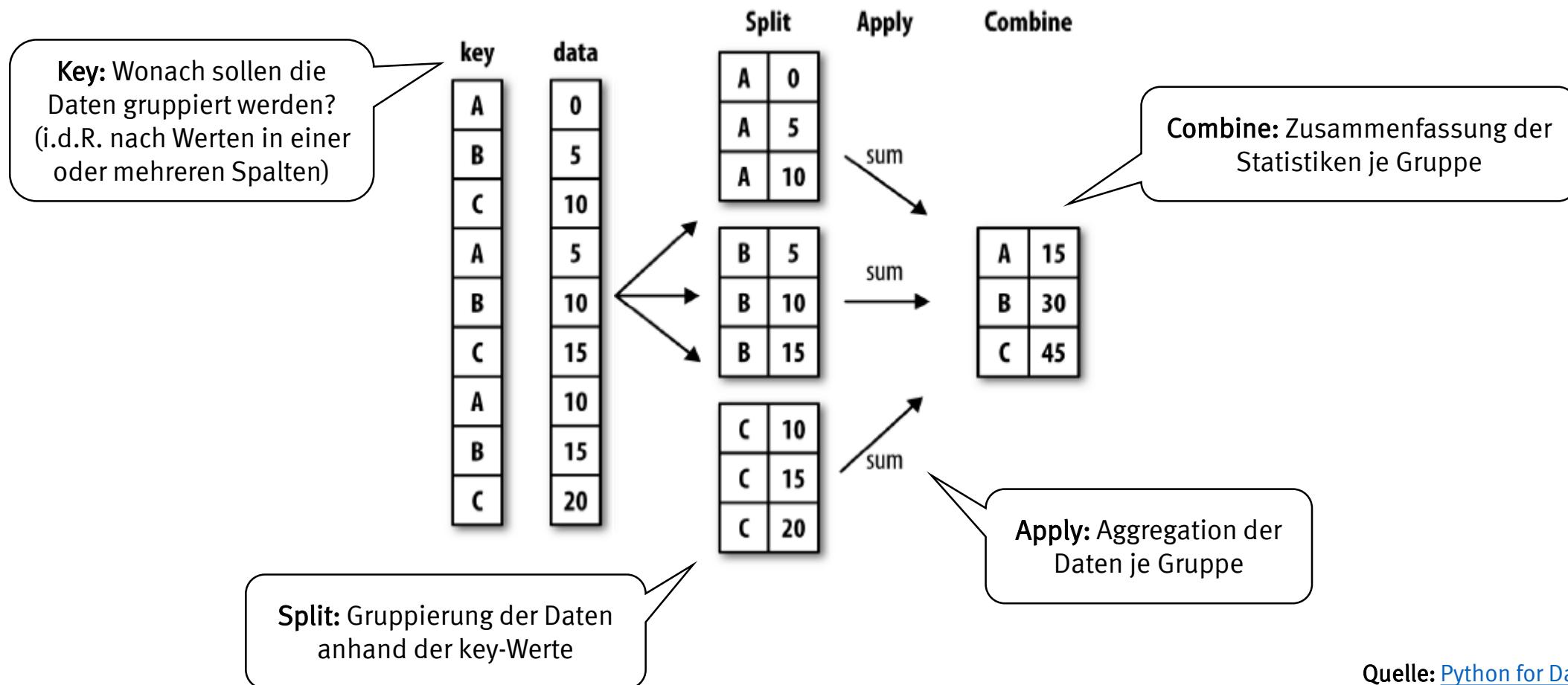


# 3 Datenmanipulation und Datenanalyse

## 3.7 Operationen auf Gruppen



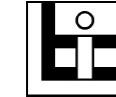
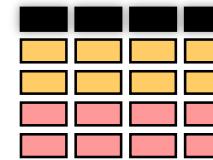
### Das „SPLIT-APPLY-COMBINE PARADIGM“:



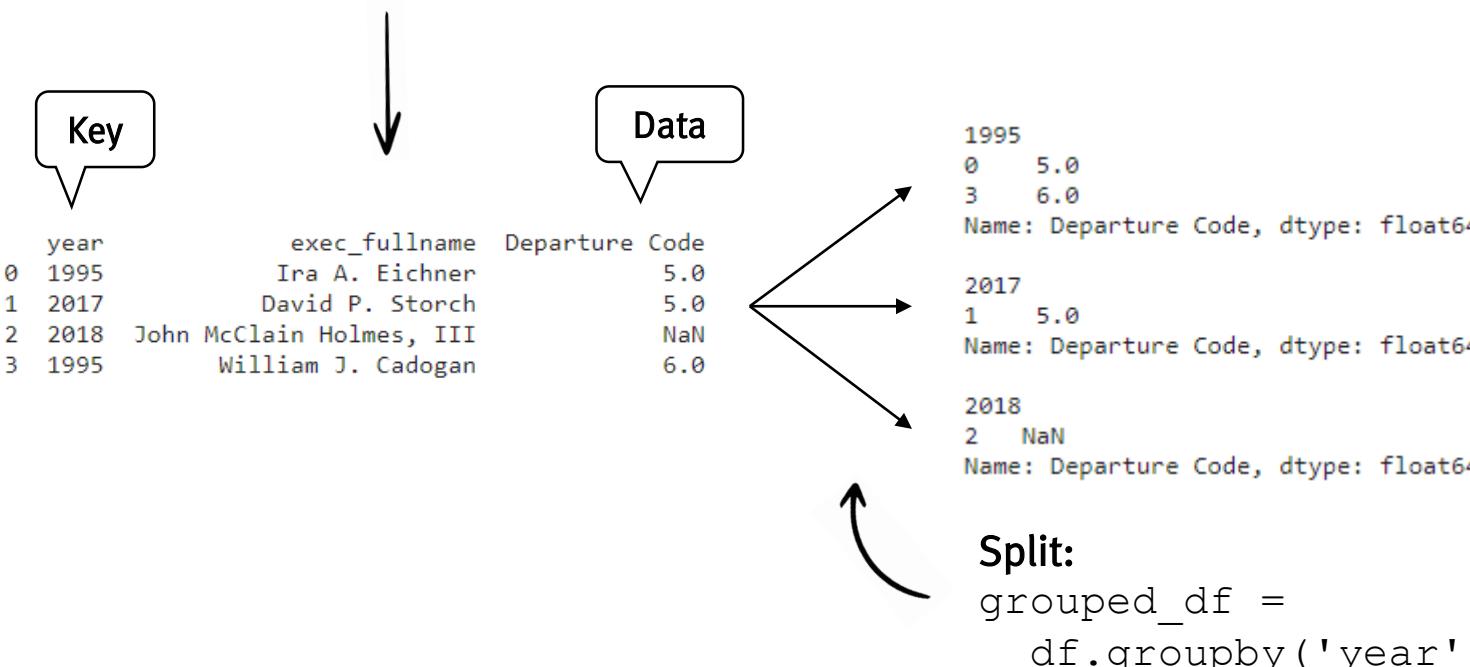
Quelle: [Python for Data Analysis](#)

# 3 Datenmanipulation und Datenanalyse

## 3.7 Operationen auf Gruppen



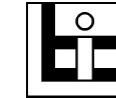
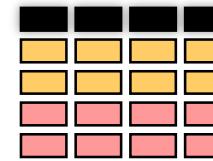
Spalten, die weder als Key noch für Berechnungen genutzt werden, werden ausgelassen.



\* df.groupby erstellt ein groupby Objekt. Dabei finden noch keine Berechnungen statt, es wird nur „gespeichert“ welche Beobachtungen zu welcher Gruppen gehören.

# 3 Datenmanipulation und Datenanalyse

## 3.7 Operationen auf Gruppen



### Apply & Combine:

```
grouped_df.sum()
```

```
1995  
0    5.0  
3    6.0  
Name: Departure Code, dtype: float64  
  
2017  
1    5.0  
Name: Departure Code, dtype: float64  
  
2018  
2    NaN  
Name: Departure Code, dtype: float64
```

```
year  
1995    11.0  
2017     5.0  
2018     0.0  
Name: Departure Code, dtype: float64
```

### Apply & Combine:

```
grouped_df.mean()
```

```
year  
1995    5.5  
2017     5.0  
2018     NaN  
Name: Departure Code, dtype: float64
```

### Apply & Combine:

```
grouped_df.agg  
(['sum', 'mean'])
```

year	sum	mean	tot	avg
1995	11.0	5.5	11.0	5.5
2017	5.0	5.0	5.0	5.0
2018	0.0	Nan	0.0	Nan

oder

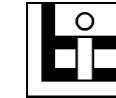
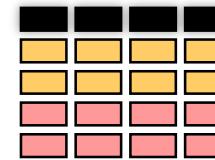
year	tot	avg
1995	11.0	5.5
2017	5.0	5.0
2018	0.0	Nan

### Apply & Combine:

```
grouped_df.agg([('tot', 'sum'), ('avg', 'mean')])
```

# 3 Datenmanipulation und Datenanalyse

## 3.7 Operationen auf Gruppen

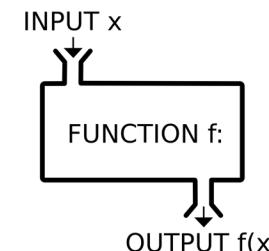


Für Fortgeschrittene: `df.groupby().apply()` erlaubt es uns beliebige Funktionen je Gruppe anzuwenden (z.B. Filtern oder Datenmanipulation innerhalb einer Gruppe).

```
year      exec_fullname  Departure Code
0  1995          Ira A. Eichner      5.0
1  2018        David P. Storch      5.0
2  2018  John McClain Holmes, III     NaN
3  1995    William J. Cadogan      6.0
```

```
df.groupby('year').apply(
    lambda g: g[g['Departure Code'] == 5]
)
```

```
year      exec_fullname  Departure Code
year
1995 0  1995          Ira A. Eichner      5.0
2017 1  2017        David P. Storch      5.0
```



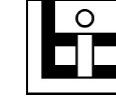
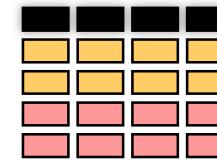
```
year      exec_fullname  Departure Code
0  1995          Ira A. Eichner      5.0
1  2018        David P. Storch      5.0
2  2018  John McClain Holmes, III     NaN
3  1995    William J. Cadogan      6.0
```

```
df.groupby('year').apply(
    lambda g: g['Departure Code'].fillna(
        g['Departure Code'].mean()
    )
)
```

```
year      exec_fullname  Departure Code
year
1995 0  1995          Ira A. Eichner      5.0
      3  1995    William J. Cadogan      6.0
2018 1  2018        David P. Storch      5.0
      2  2018  John McClain Holmes, III     5.0
```

# 3 Datenmanipulation und Datenanalyse

## 3.7 Operationen auf Gruppen: Übungsaufgaben



Forschungsteam  
Berens

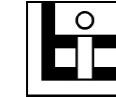
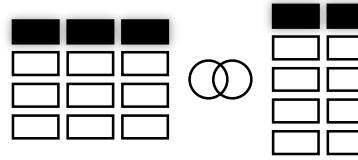


**Übungsaufgaben:**  
10 Minuten



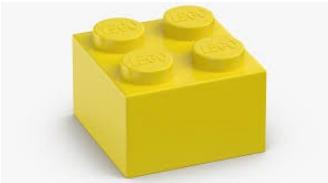
# 3 Datenmanipulation und Datenanalyse

## 3.8 Joinen von Datensätzen

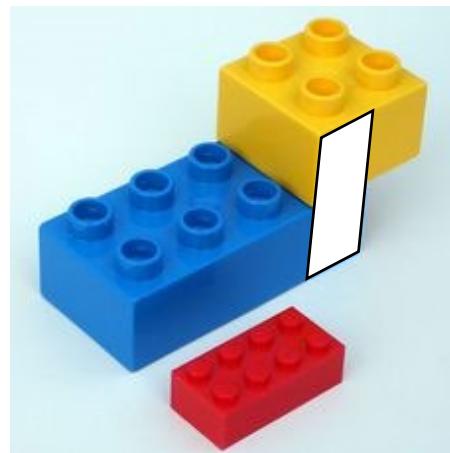


Forschungsteam  
Berens

Firms



CEOs



„Kleinster  
gemeinsamer Nenner“  
beider Tabellen

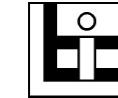
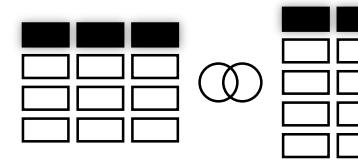
Quelle: [Data 8, UC Berkeley](#)



Quelle: [r/MemeTemplatesOfficial](#)

# 3 Datenmanipulation und Datenanalyse

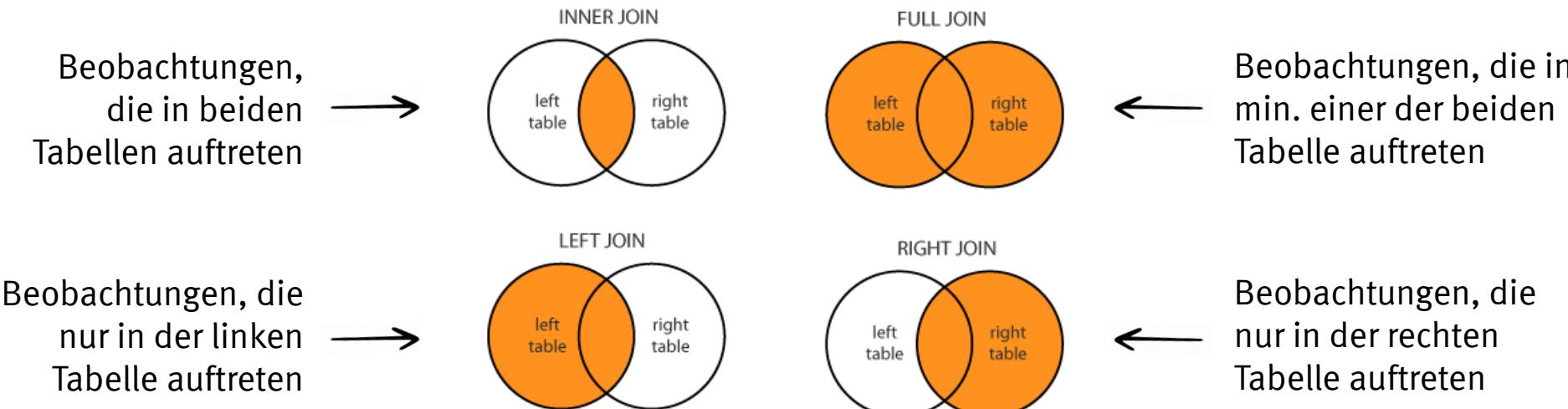
## 3.8 Joinen von Datensätzen



Forschungsteam  
Berens

**Merge/Join:** Kombinieren von Datensätzen anhand eines oder mehrerer *keys*. Joins sind ein integraler Bestandteil von relationalen Datenbanken und die Anwendungslogik in pandas ist stark an SQL angelehnt.

```
df1.merge(df2, how='inner', on='key', suffixes=('_x', '_y'))
```



**Concatenation:** Spalten-/Zeilenweises Aneinanderhängen von Datensätzen (BINDING oder STACKING).

```
pd.concat([df1, df2], axis=0)
```

Quelle: [Stackoverflow](#)

1

Einführung und Motivation

2

Grundlagen der Programmierung mit Python

3

Datenmanipulation und Datenanalyse mit pandas

4

Ausblick

# 4 Ausblick

## 4.1 Warum Python statt Excel und VBA?

---

- » **Skalierbarkeit:** Python erlaubt den Import und die effiziente Verarbeitung von sehr großen Datenmengen und/oder vielen verschiedenen Datensätzen gleichzeitig.
- » **Modularität und Ökosystem:** Python bietet ein riesiges Ökosystem an Erweiterungen, die Funktionalitäten liefern, die weit über Excel hinausgehen (z.B. Analyse von Text-, Bild-, und Audiodaten, interaktive Datenvisualisierung, Web Scraping, Entwicklung von Web Apps, Machine Learning, etc.)
- » **Automation und Replicability:** Ein Python-Programm integriert alle Datenmanipulations- und Datenanalyse-Schritte. Diese sind transparent nachvollziehbar und lassen sich 1-zu-1 auf einen neuen Datensatz mit gleicher Struktur anwenden. Gleiches gilt für die Wiederholung einer Datenanalyse. Analysen in Excel müssen u.U. jedes Mal aufs neue Verformelt werden, wobei die Formeln mitunter in riesigen Excel-Dateien versteckt sind. Das macht die Analyse schwer nachvollziehbar.
- » **Open Source:** Python ist Open Source. Somit muss im Gegensatz zu Excel keine Lizenz erworben werden.
- » **Wissenstransfer:** Wer Daten in Excel analysieren kann, kann nicht auch automatisch Daten in Python, R oder Stata analysieren. Wer hingegen die Datenanalyse mittels einer Programmiersprache beherrscht, kann das Erlernte einfacher auf neue Sprachen oder GUI-basierte Anwendungen übertragen.
- » **Die Macht der Gewohnheit:** Excel wirkt wie das einfacherer, intuitivere Tool nicht nur aufgrund seiner grafischen Nutzeroberfläche (GUI), sondern auch weil wir Excel schon früh gelernt haben. Welches Tool würde den Vergleich entscheiden, wenn der erste Kontakt in der Schule mit Python gewesen wäre?

# 4 Ausblick

## 4.2 Ein Blick über den Tellerrand

Datenvisualisierung mit Python:

 seaborn  matplotlib



Textverarbeitung mit Python:

 spaCy  NLTK

Machine and Deep Learning:

 scikit-learn  PyTorch

Web Scraping:

  Scrapy

Vorbereitung auf die  
Abschlussklausur:

- » Wiederholung der Übungsaufgaben, insb. Experimentieren mit den verfügbaren Notebooks
- » Optional: pandas [Short Course](#) by deeplearning.ai

Weiterführende Angebote:

- » [Programmierung mit Python](#) (WS 23/24); [Python ZIV-Kurs](#); MOOC, z.B. [Coursera](#)
- » Empirische Bachelorarbeit
- » Auf uns zukommen