Python Pandas I

Programmierkurs 2 Data Science WS23/24

https://www.scmp.com/news/people-culture/environment/article/3164671/why-are-pandas-so-chonky-despite-their-vegan-diet

Leonard Traeger
M. Sc. Information Systems
leonard.traeger@fh-dortmund.de

Disclaimer

Slides are mainly based on

- https://pandas.pydata.org/docs/index.html and
- https://www.w3schools.com/python/pandas/pandas_intro.asp

→ Find everything you need to know there!

Official Pandas cheat sheet:

- https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

More beginner-friendly Pandas cheat sheet by Dataquest:

- https://drive.google.com/file/d/1UHK8wtWbADvHKXFC937IS6MTnlSZC_zB/view

Learning Goals Python Pandas I

- **Explain** how I/O integrates into the Pandas library and **list** a few different supported data formats.
- **Explain** the core data structures of Pandas and relate these to regular Python containers.
- **Demonstrate** Pandas I/O, Data Inspection, Indexing, Selection, and Deletion.
- **Analyse** on what attributes to apply Data Reduction given an example.
- **Apply** Data Masking and **demonstrate** how to insert new records and change values of rows or columns of a DataFrame.
- **Identify** potential cleaning functions, data transformations and visualisations and **justify**.

name	age	height
Biden	78	1.83
Trump	70	1.9
Obama	47	1.87

NumPy Structured Arrays (Recap)

```
name = ['Biden', 'Trump', 'Obama']
age = [78, 70, 47]
height = [1.83, 1.90, 1.87]
```

We have constructed a relational table!

```
presidents = np.zeros(3, dtype={'names':('name', 'age', 'height'),
                                'formats':('U10', 'i4', 'f8')})
print(presidents.dtype)
# Prints "[('name', '<U10'), ('age', '<i4'), ('height', '<f8')]"
```

- Fill the array with our lists of values.

```
presidents['name'] = name
presidents['age'] = age
presidents['height'] = height
print(presidents)
# Prints "[('Biden', 78, 1.83) ('Trump', 70, 1.9 ) ('Obama', 47, 1.87)]"
```

- Data is now conveniently arranged in one structured array.

What is Pandas? Why use it?

"**Pan**el **Da**ta" or "Python Data Analysis" (Wes McKinney 2008)

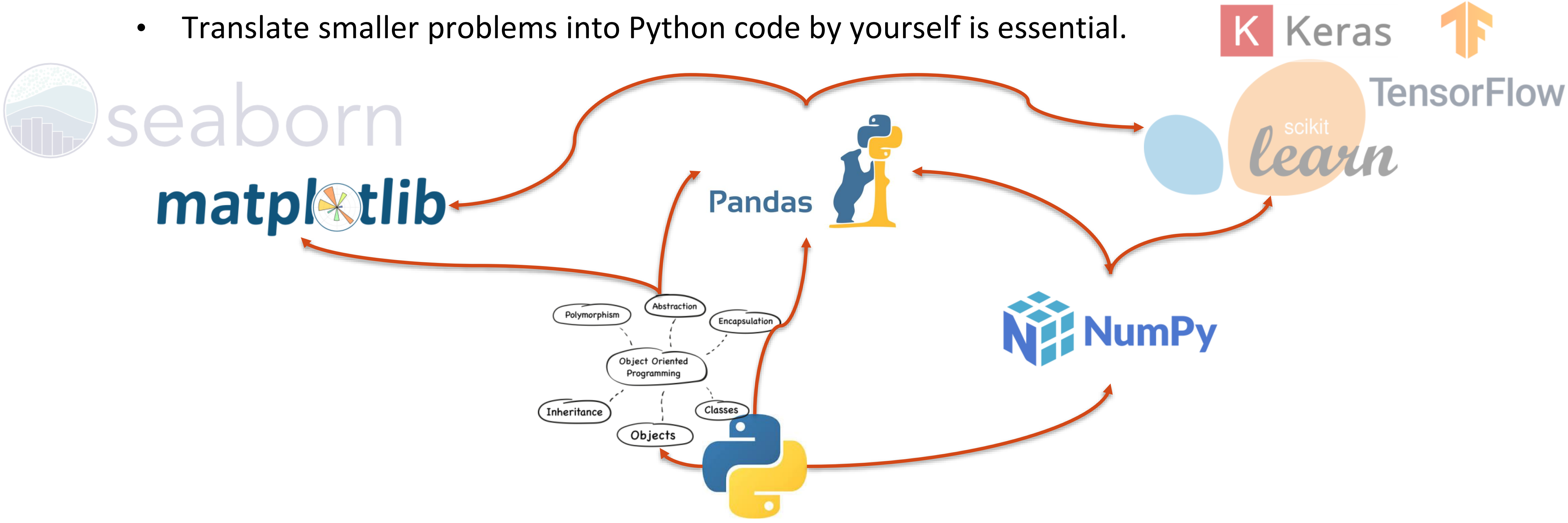
- **Open-source** library in Python.
- **High-performance** tool for **data structures** and **analytics**.

- Pandas helps us to **import**, **clean**, **explore**, **manipulate**, and **analyze data** and **make conclusions** based on statistical theories.
- Use as soon **tabular data** comes up e.g., **spreadsheets, databases, …** assuming not so large files.

```
import numpy as np
import pandas as pd
```

How to master Python Libraries?

- Narrow down your focus by learning the **basics first (Python I,II,III, NumPy).**
- Translate smaller problems into Python code by yourself is essential.



What can I do with Pandas?

- View your data.
- Get a quick idea of what you are dealing with.
- Get statistic summary.
- View and cast data types.
- Group, select, and apply functions…

… and much more!

→ Excel, SPSS and the power of Python combined!

Series Object

- **One-dimensional** labeled array.
- Created from Python **lists, tuples or dictionaries** (ordered or indexed).
- Can have indices
  - Dictionaries return key as index with value
  - If none, an index will be created having the values [0, ..., len(data) - 1].

		Ordered	Changeable	Indexed	Duplicates
List	[ ]	Yes	Yes	Yes	Yes
Tuple	( )	Yes	No	Yes	Yes
~~**Set**~~	~~{ }~~	~~No~~	~~Yes~~	~~No~~	~~No~~
Dictionary	{“_:_“}	No	Yes	Yes	No

pandas.Series

Series Object (cont.)

```
t = ('Zuckerberg', 42, 'Gates', 'Bezos', 'Musk')
print(pd.Series(data=t))
print(type(pd.Series(data=t)[0])) # Prints "<class 'str'>"
print(type(pd.Series(data=t)[1])) # Prints "<class 'int'>"

s = pd.Series(data=t, index=['a', 'b', 'c', 'd', 'e'])
print(s)
print(s['a'])                      # Prints "Zuckerberg"

d = {'person': 2, 'cat': 4, 'spider': 8}
print(pd.Series(data=d))
```

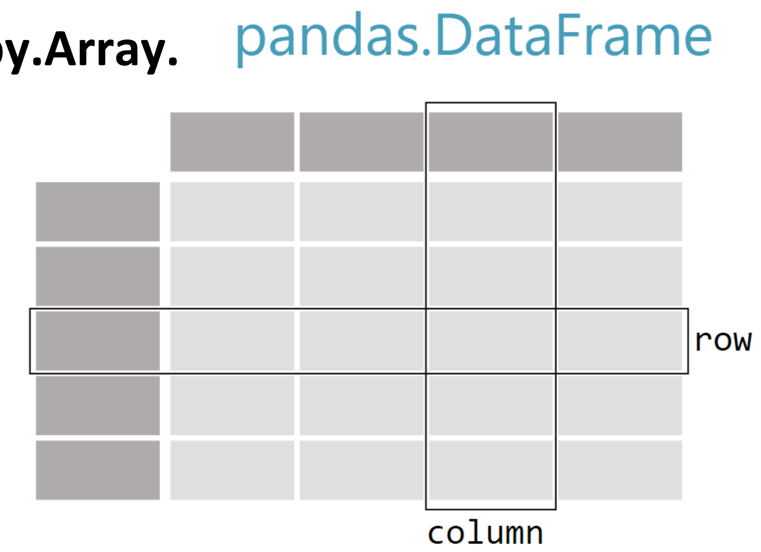
```
0    Zuckerberg
1            42
2         Gates
3         Bezos
4          Musk
dtype: object
```

```
a    Zuckerberg
b            42
c         Gates
d         Bezos
e          Musk
dtype: object
```

```
person    2
cat       4
spider    8
dtype: int64
```

Data Frame

- Combine Series Objects and you get DataFrames.
- **Two-dimensional** labeled data structure with **columns** of **different types**.
- Has columns and optional indices (row keys).
- Think of it like a spreadsheet or SQL table, or a dictionary of Series objects.
- Created from Python **arrays**, **dictionaries, pandas.Series or numpy.Array.**



Slides may use interchangeably
- Column, attribute, feature, and object.
- Row, record, values, instance.

Data Frame (cont.)

```
animal = ['person', 'cat', 'spider', 'panda']
arms = [2,4,8,2]
pd.DataFrame(data={'animal': animal, 'arms': pd.Series(arms)})
```

or

```
pd.DataFrame(np.array([['person', 2], ['cat',4], ['spider',8], ['panda',2]]),
             columns=['animal', 'arms'])
```

	animal	arms
0	person	2
1	cat	4
2	spider	8
3	panda	2

Avoid print(DataFrame) as Jypiter has a great DataFrame interpreter

Colab Data Frame (cont.)

	animal	arms
0	person	2
1	cat	4
2	spider	8
3	panda	2

1 to 4 of 4 entries Filter

index: to

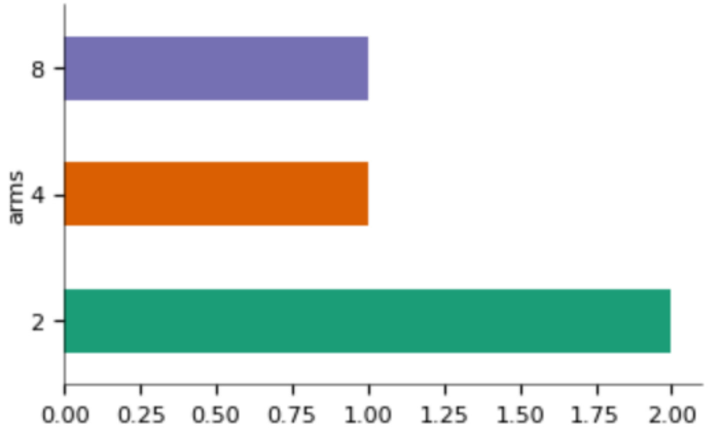
animal:

arms:

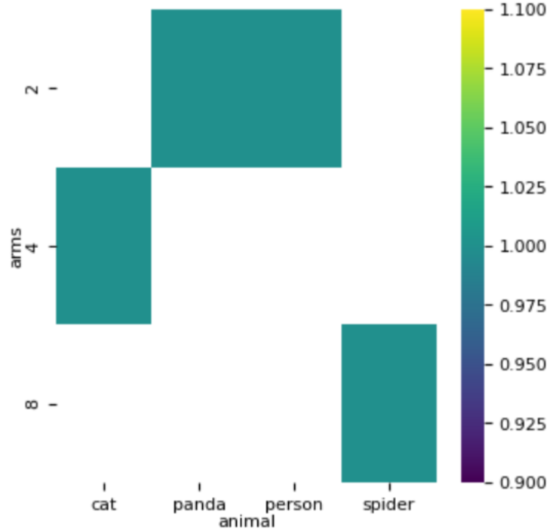
Search by all fields:

index	animal	arms
0	person	2
1	cat	4
2	spider	8
3	panda	2

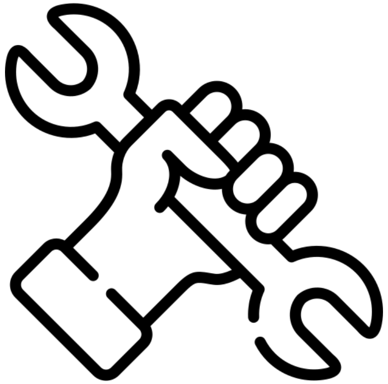
Show 25 per page



2-d categorical distributions



```python
import numpy as np
import pandas as pd
```



Training #1

- Create a Pandas DataFrame based on the following data:

name	#games	age
Roman Bürki	200	31
Marwin Hitz	60	34
Gregor Kobel	30	23
Marco Reus	372	34

- Explore the  and  function. What do you think about them?

“A Bright Future, Not without Challenges”

Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. ***2023****. How Large Language Models Will Disrupt Data Management. Proc. VLDB Endow. 16, 11 (July 2023), 3302–3309. https://doi.org/10.14778/3611479.3611527*

ETL Nightmare

- More automated data from A to B pipelines will lead to more complexity.

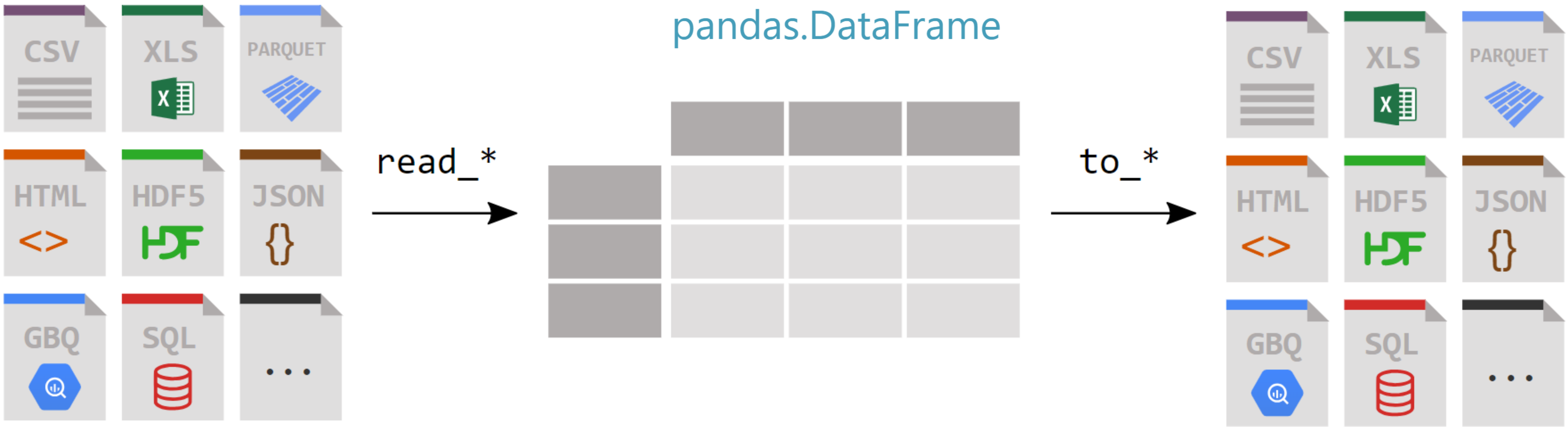
Trust

- Increased ability to explain and justify answers will be needed (privacy, attribution, and value).

Lack of data

- Open Internet is full of rich data sources which is not always true in enterprises.

Pandas **Integration**



I/O API

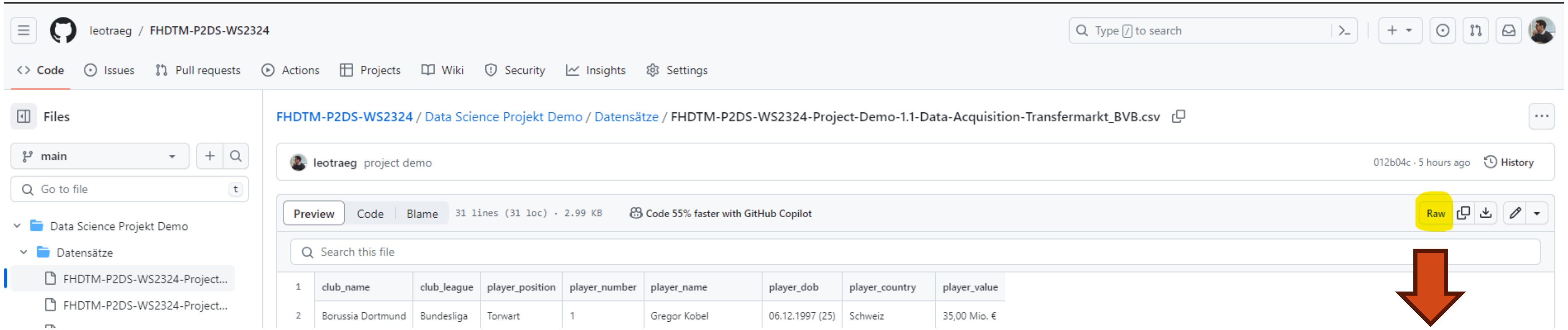
- Pandas **Reader function** (…to read some file) e.g.,
  - `pd.read_csv("someFolder/.../someFile.csv")`
  - generally **returns** a pandas **DataFrame** object.

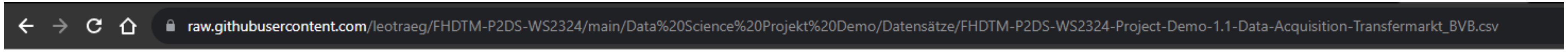
- Pandas **Writer function** (reverse operation) e.g.,
  - `DataFrame.to_csv("Folder/.../FileVer2.csv")`
  - transforms a pandas DataFrame to the desired format file and uploads it to the path.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	to_orc
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

https://pandas.pydata.org/docs/user_guide/io.html

I/O API Example

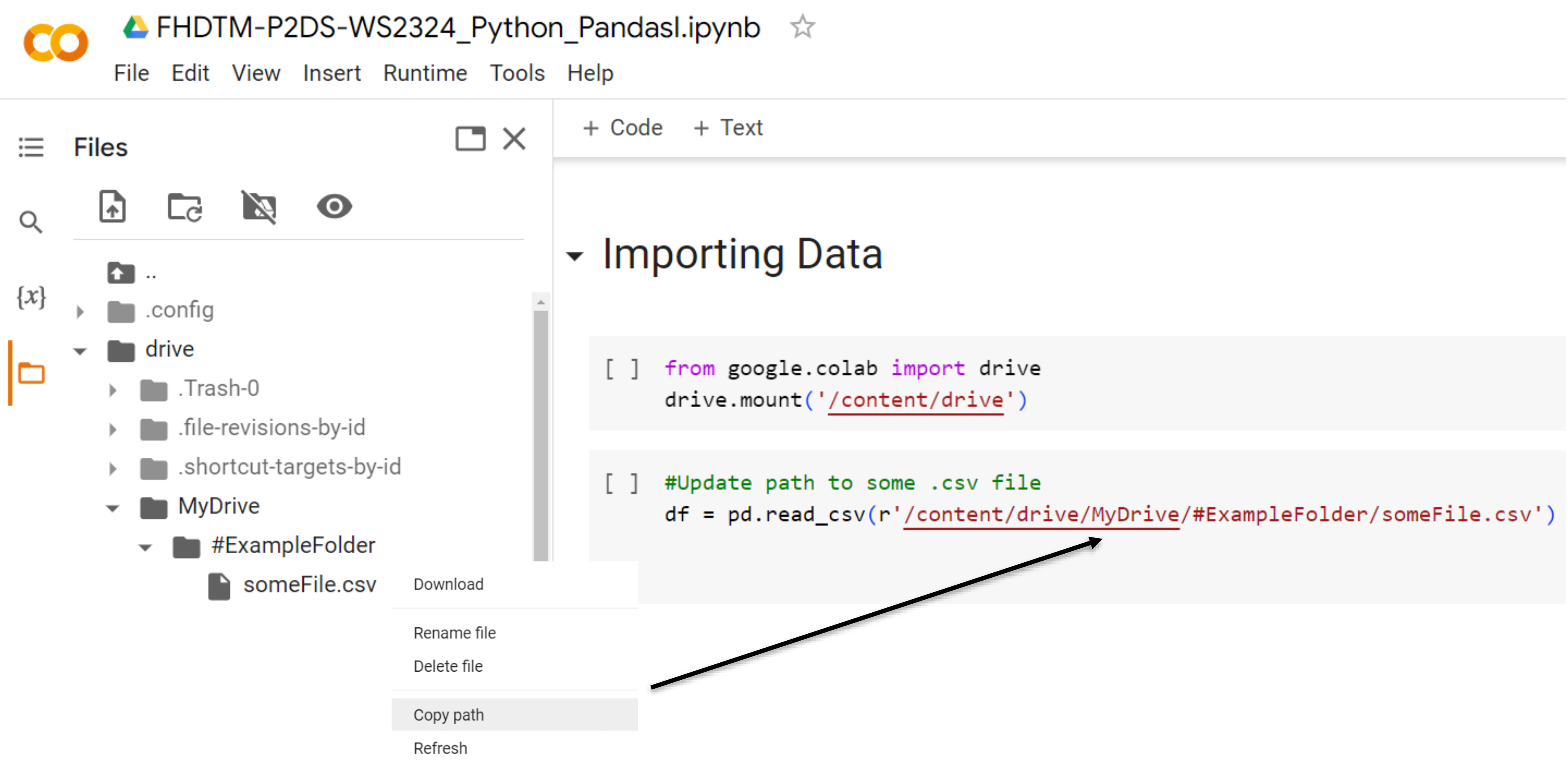





```
club_name,club_league,player_position,player_number,player_name,player_dob,player_country,player_value
Borussia Dortmund,Bundesliga,Torwart,1,Gregor Kobel,06.12.1997 (25),Schweiz,"35,00 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,35,Marcel Lotka,25.05.2001 (22),Deutschland,"1,50 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,33,Alexander Meyer,13.04.1991 (32),Deutschland,"1,00 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,31,Silas Ostrzinski,19.11.2003 (19),Deutschland,150 Tsd. €
Borussia Dortmund,Bundesliga,Abwehr,4,Nico Schlotterbeck,01.12.1999 (23),Deutschland,"40,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,25,Niklas Süle,03.09.1995 (27),Deutschland,"35,00 Mio. €"
```

```python
url = "https://raw.githubusercontent.com/leotraeg/FHDTM-P2DS-WS2324/main/Data%20
df_bvb_player = pd.read_csv(url)
```

I/O with Google Colab + Drive



Technology Arts Sciences
TH Köln

Data Inspection

- `df.columns` or `df.keys` returns a **list** with **colum names**.

```
df_bvb_player.columns
```

```
Index(['club_name', 'club_league', 'player_position', 'player_number',
       'player_name', 'player_dob', 'player_country', 'player_value'],
      dtype='object')
```

- `df.shape` returns **number** of **instances** and number of **columns**.

```
df_bvb_player.shape
```

```
(30, 8)
```

- `df.size` returns **total number** of **cells**.

```
df_bvb_player.size
```

```
240
```

no function, no ( ) !

Data Inspection (cont.)

- `df.index` returns the **range** of the **index** of the frame.

```
df_bvb_player.index
```

```
RangeIndex(start=0, stop=30, step=1)
```

- `df.s.value_counts()` returns a dictionary of a series (attribute) with key(**distinct row value elements**) and value(**count**).

```
df_bvb_player.player_position.value_counts()
```

```
Abwehr        10
Mittelfeld     9
Sturm          7
Torwart        4
Name: player_position, dtype: int64
```

Let‘s see what makes Python superior to Excel

Example: Power of Pandas+Python

```python
for position, count in df_bvb_player.player_position.value_counts().items():
  percentage = round(count/len(df_bvb_player), 2) * 100
  print(position, percentage)
```

```
Abwehr 33.0
Mittelfeld 30.0
Sturm 23.0
Torwart 13.0
```

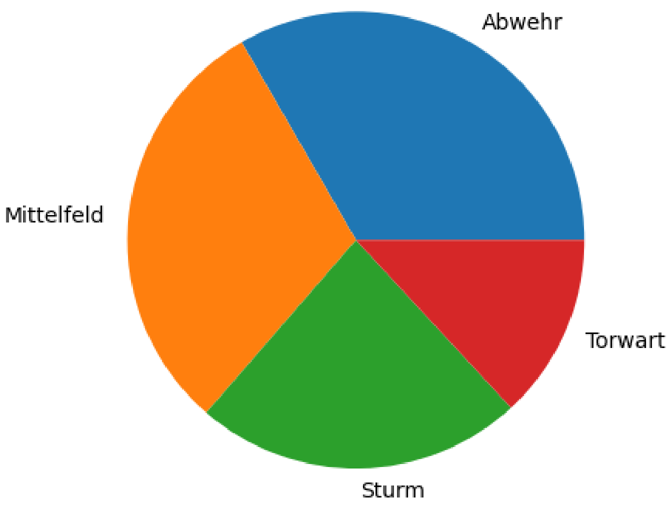
Example: Power of Pandas+Python

```
position_plot = []
value_plot = []

for position, count in df_bvb_player.player_position.value_counts().items():
  percentage = round(count/len(df_bvb_player), 2) * 100
  #print(position, percentage)
  position_plot.append(position)
  value_plot.append(percentage)
```

Example: Power of Pandas+Python

```
position_plot = []
value_plot = []

for position, count in df_bvb_player.player_position.value_counts().items():
  percentage = round(count/len(df_bvb_player), 2) * 100
  #print(position, percentage)
  position_plot.append(position)
  value_plot.append(percentage)
```

```
import matplotlib.pyplot as plt
fig1, ax1 = plt.subplots()
ax1.pie(value_plot, labels=position_plot)
plt.show()
```



We will learn about this library soon!

Viewing Data

- `df.head(n)` returns **n** rows from the **top** of the **frame**.

```
df_bvb_player.head(2)
```

	club_name	club_league	player_position	player_number	player_name	player_dob
0	Borussia Dortmund	Bundesliga	Torwart	1	Gregor Kobel	06.12.1997 (25)
1	Borussia Dortmund	Bundesliga	Torwart	35	Marcel Lotka	25.05.2001 (22)

- `df.tail(n)` returns **n** rows from the **bottom** of the **frame**.

```
df_bvb_player.tail(2)
```

	club_name	club_league	player_position	player_number	player_name	player_do
28	Borussia Dortmund	Bundesliga	Sturm	9	Sébastien Haller	22.06.199 (2
29	Borussia Dortmund	Bundesliga	Sturm	18	Youssoufa Moukoko	20.11.200 (1

Data Indexing

- `df[start:end]` returns frame with **inbound implicit integer index.**

```
df_bvb_player[1:3]
```

	club_name	club_league	player_position	player_number	player_name	player_dob
1	Borussia Dortmund	Bundesliga	Torwart	35	Marcel Lotka	25.05.2001 (22
2	Borussia Dortmund	Bundesliga	Torwart	33	Alexander Meyer	13.04.1991 (32

- `df.set_index([column_a])` returns **copy** of a frame with **reassigned index** based on **column.**

```
df_bvb_player.set_index(['player_number'])
```

player_number	club_name	club_league	player_position	player_name	player_dob
1	Borussia Dortmund	Bundesliga	Torwart	Gregor Kobel	06.12.1997 (25)
35	Borussia Dortmund	Bundesliga	Torwart	Marcel Lotka	25.05.2001 (22)

Copy behavior of DataFrame.methods

- DataFrame methods usually behave default-wise with `inplace=False`.
- This means that the method is executed on a copy of the DataFrame and returned.

```
df_bvb_player.set_index(['player_number'])
```

- The original DataFrame variable remains the same. If you want to actually change it:

```
df_bvb_player = df_bvb_player.set_index(['player_number'])
```

or

```
df_bvb_player.set_index(['player_number'], inplace=True)
```

- The newly assigned column becomes the index and is dropped as a column.

Data Selection

https://www.bundesliga.com/en/news/Bundesliga/marco-reus-still-getting-better-borussia-dortmund-position-517380.jsp

- `df.loc[index]` returns row object of **explicit integer index**.

```
df_bvb_player.loc[11]
```

```
club_name              Borussia Dortmund
club_league                   Bundesliga
player_position               Mittelfeld
player_name                   Marco Reus
player_dob             31.05.1989 (34)
player_country               Deutschland
player_value              7,00 Mio. €
```

- `df.iloc[index]` returns row object of **implicit integer index**.

```
df_bvb_player.iloc[11]
```

```
club_name              Borussia Dortmund
club_league                   Bundesliga
player_position                  Abwehr
player_name                  Marius Wolf
player_dob             27.05.1995 (28)
player_country               Deutschland
player_value             10,00 Mio. €
```

`.loc` and `.iloc` also work for Series

Data Selection (cont.)

- `df[column]` returns **specified attribute dictionary** of frame.

```
df_bvb_player["club_name"]
```

```
player_number
1      Borussia Dortmund
35     Borussia Dortmund
33     Borussia Dortmund
31     Borussia Dortmund
```

- `df[[column_a, column_b, ...]]` returns **frame** of **specified attributes**.

```
df_bvb_player[["club_name","player_name","player_dob"]]
```

	club_name	player_name	player_dob
player_number			
1	Borussia Dortmund	Gregor Kobel	06.12.1997 (25)
35	Borussia Dortmund	Marcel Lotka	25.05.2001 (22)
33	Borussia Dortmund	Alexander Meyer	13.04.1991 (32)

Data Deletion / Reduction / Subset Selection

- `df.drop(index=[index_a, index_b, ...]` returns frame copy with **remaining records.**

```
df_bvb_player.drop(index=[1])
```

player_number	club_name	club_league	player_position	player_name	player_dob
35	Borussia Dortmund	Bundesliga	Torwart	Marcel Lotka	25.05.2001 (22)
33	Borussia Dortmund	Bundesliga	Torwart	Alexander Meyer	13.04.1991 (32)

- `df.drop(columns=[column_a, column_b, ...]` returns frame copy with **remaining columns.**

```
df_bvb_player.drop(columns=["club_league"]).head()
```

player_number	club_name	player_position	player_name	player_dob	player_country
1	Borussia Dortmund	Torwart	Gregor Kobel	06.12.1997 (25)	Schweiz
35	Borussia Dortmund	Torwart	Marcel Lotka	25.05.2001 (22)	Deutschland
33	Borussia Dortmund	Torwart	Alexander Meyer	13.04.1991 (32)	Deutschland

When to do **Data Reduction**?

Redundant attributes

- Duplicate much or all of the information contained in one or more other attributes,
- E.g., **date of birth** and **age** (if snapshot time of age computation is today).

Irrelevant attributes

- Contain no information that becomes more valueable for the analysis than without.
- E.g., **club name** or **club league** as they have the same value for all player records.
  - Different if more players from different clubs and leagues join analysis!

Disclaimer: Machine Learning, particularly Neural Networks, do this automatically ☺

DataFrame as Array

…forget about player_number as the new index

- View the DataFrame as an enhanced **two-dimensional array** using the `values` attribute:

```
df_bvb_player.values
```

```
array([['Borussia Dortmund', 'Bundesliga', 'Torwart', 1, 'Gregor Kobel',
        '06.12.1997 (25)', 'Schweiz', '35,00 Mio. €'],
       ['Borussia Dortmund', 'Bundesliga', 'Torwart', 35, 'Marcel Lotka',
        '25.05.2001 (22)', 'Deutschland', '1,50 Mio. €'],
       ['Borussia Dortmund', 'Bundesliga', 'Torwart', 33,
        'Alexander Meyer', '13.04.1991 (32)', 'Deutschland',
        '1,00 Mio. €'],
```

All array functions can be used like masking, Ufuncs up to comprehension lists…

- We can **transpose** the full DataFrame to swap rows and columns using the `T` attribute:

```
df_bvb_player.T
```

	0	1	2	3	4
club_name	Borussia Dortmund	Borussia Dortmund	Borussia Dortmund	Borussia Dortmund	Borussia Dortmund
club_league	Bundesliga	Bundesliga	Bundesliga	Bundesliga	Bundesliga
player_position	Torwart	Torwart	Torwart	Torwart	Abwehr

Preview: **Data Transformation**

- We can easily modify the DataFrame and **filter** for **rows** and **conditions**:

```
df_bvb_player.player_number %2 == 0
```

```
0     False
1     False
2     False
3     False
4      True
```

- Using **Masking**, we can view the DataFrame with rows satisfying the conditions:

```
df_bvb_player[df_bvb_player.player_number %2 == 0]
```

	club_name	club_league	player_position	player_number	player_name	player_dc
4	Borussia Dortmund	Bundesliga	Abwehr	4	Nico Schlotterbeck	01.12.199 (2
7	Borussia Dortmund	Bundesliga	Abwehr	44	Soumaïla Coulibaly	14.10.200 (1
10	Borussia Dortmund	Bundesliga	Abwehr	26	Julian Ryerson	17.11.199 (2

Preview: **Data Transformation** (cont.)

- We can also easily modify the DataFrame, .e.g., adding a new column:

```
df_bvb_player.player_number %2 == 0
```

```
0     False
1     False
2     False
3     False
4      True
```

```
df_bvb_player['is_even'] = df_bvb_player.player_number %2 == 0
```

```
df_bvb_player.head()
```

player_number	player_name	player_dob	player_country	player_value	is_even
1	Gregor Kobel	06.12.1997 (25)	Schweiz	35,00 Mio. €	False
35	Marcel Lotka	25.05.2001 (22)	Deutschland	1,50 Mio. €	False
33	Alexander Meyer	13.04.1991 (32)	Deutschland	1,00 Mio. €	False
31	Silas Ostrzinski	19.11.2003 (19)	Deutschland	150 Tsd. €	False
4	Nico Schlotterbeck	01.12.1999 (23)	Deutschland	40,00 Mio. €	True

Data Masking

- `df.column operator value` returns **index mask**.

```
df_bvb_player.player_number > 40
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7      True
```

- `df[mask]` returns **dataframe** with rows based on mask.

```
df_bvb_player[df_bvb_player.player_number > 40]
```

	club_name	club_league	player_position	player_number	player_name	player_d
7	Borussia Dortmund	Bundesliga	Abwehr	44	Soumaïla Coulibaly	14.10.20 (1
8	Borussia Dortmund	Bundesliga	Abwehr	47	Antonios Papadopoulos	10.09.19 (2
24	Borussia Dortmund	Bundesliga	Sturm	43	Jamie Bynoe-Gittens	08.08.20 (1

Data Masking (cont.)

Use & and |
instead of `and` and `or`
for multiple masking conditions

- Masks can become as complex as you want via **Boolean logic**.

```
df_bvb_player[(df_bvb_player.player_number > 40) &
              (df_bvb_player.player_country == "Deutschland")]
```

	club_name	club_league	player_position	player_number	player_name	player_do
8	Borussia Dortmund	Bundesliga	Abwehr	47	Antonios Papadopoulos	10.09.199 (23

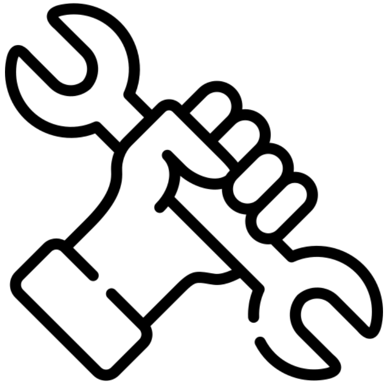
- `df.loc[mask, column]` returns the **row object(s)**, and the attribute value of a specified column capable to changes.

```
df_bvb_player.loc[(df_bvb_player.player_number > 40) &
                  (df_bvb_player.player_country == "Deutschland"),
                  "player_value"] = "1,00 Mio. €"
```

```
df_bvb_player.loc[(df_bvb_player.player_number > 40) &
                  (df_bvb_player.player_country == "Deutschland"),
                  "player_value"]
```

```
8    1,00 Mio. €
Name: player_value, dtype: object
```

Training #2



Open a blank .ipynb file and import the .csv file

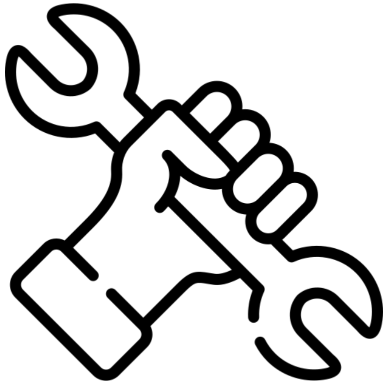
```
url = "https://raw.githubusercontent.com/leotraeg/FHDTM-P2DS-WS2324/main/Data%20Science%20Projekt%20Demo/Datens%C3%A4tze/FHDTM-P2DS-WS2324-Project-Demo-1.1-Data-Acquisition-Transfermarkt_BVB.csv"
```

as a pandas data frame `pd.read_csv(url)`.

1. Delete all players from Norway „Norwegen“.
   - Use masking to get the index of relevant players, e.g., `df.drop(mask.index, inplace=True)`
2. “Hanna Muster” joined the BVB club this season. You can add a new record to the DataFrame using a `dictionary { }` and `pd.Series` object by `df.loc[len(df)] = pd.Series(data=dictionary)`.
   - Assign the unknown attributes with numpy’s `np.NaN` value.
   - You can compute Hanna‘s age by yourself or use the `get age` method from the Python III class.

```
from datetime import date, timedelta, datetime
```

Training #2



Check:

1)
```
df_bvb_player[df_bvb_player.player_country == "Norwegen"]
```

club_name	club_league	player_position	player_number	player_name	player_dob	player_country	playe

2)
```
df_bvb_player.tail()
```

	club_name	club_league	player_position	player_number	player_name	player_dob	player_country	player_value
26	Borussia Dortmund	Bundesliga	Sturm	21.0	Donyell Malen	19.01.1999 (24)	Niederlande	28,00 Mio. €
27	Borussia Dortmund	Bundesliga	Sturm	16.0	Julien Duranville	05.05.2006 (17)	Belgien	8,50 Mio. €
28	Borussia Dortmund	Bundesliga	Sturm	9.0	Sébastien Haller	22.06.1994 (29)	Elfenbeinküste	30,00 Mio. €
29	Borussia Dortmund	Bundesliga	Sturm	18.0	Youssoufa Moukoko	20.11.2004 (18)	Deutschland	30,00 Mio. €
30	Borussia Dortmund	Bundesliga	NaN	NaN	Hanna Muster	17.07.2000 (23)	Deutschland	NaN

Getting to know your data

It is pretty **hard** to work, analyze and apply statistical methods on data...

...**if you do not know anything about your data!**

An **initial exploration** of your data can help you decide how to proceed.

Getting to know your data (cont.)

Take the **time** to **open** up your data **file** and have a look.

You might **be surprised** at what you find!

You may **notice obvious issues** with the data, e.g.:

- Duplicate records
- Duplicate attributes
- Nonsensical values
- Useless attributes
- Incomplete data formatting during I/O ☺

Too much data to inspect manually? Take a sample!

Viewing Meta Data

- `df.info()` returns **meta data** about the **frame**.

```
df_bvb_player.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   club_name        30 non-null     object
 1   club_league      30 non-null     object
 2   player_position  30 non-null     object
 3   player_number    30 non-null     int64
 4   player_name      30 non-null     object
 5   player_dob       30 non-null     object
 6   player_country   30 non-null     object
 7   player_value     30 non-null     object
dtypes: int64(1), object(7)
memory usage: 2.0+ KB
```

What did go unfavorable during the I/O process?

	club_name	club_league	player_position	player_number	player_name	player_dob	player_country	player_value
0	Borussia Dortmund	Bundesliga	Torwart	1	Gregor Kobel	06.12.1997 (25)	Schweiz	35,00 Mio. €
1	Borussia Dortmund	Bundesliga	Torwart	35	Marcel Lotka	25.05.2001 (22)	Deutschland	1,50 Mio. €

Viewing Meta Data (cont.)

- `df.describe()` returns **summary statistics** for **all numerical attributes** in the **frame**.

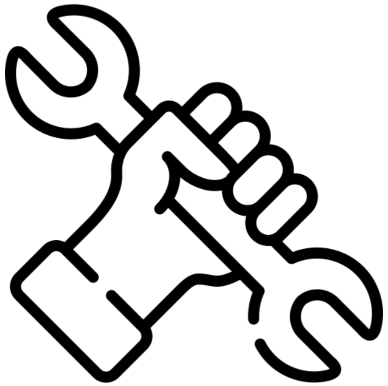
```
df_bvb_player.describe()
```

	player_number
count	30.000000
mean	20.300000
std	12.809345
min	1.000000
25%	9.250000
50%	19.500000
75%	29.250000
max	47.000000

What statistics would be more interesting?

	club_name	club_league	player_position	player_number	player_name	player_dob	player_country	player_value
0	Borussia Dortmund	Bundesliga	Torwart	1	Gregor Kobel	06.12.1997 (25)	Schweiz	35,00 Mio. €
1	Borussia Dortmund	Bundesliga	Torwart	35	Marcel Lotka	25.05.2001 (22)	Deutschland	1,50 Mio. €

```python
import numpy as np
import pandas as pd
```



Think-Pair-Share #1

Continue working on your Training#1 .ipynb file with the imported.csv file

```
url = "https://raw.githubusercontent.com/leotraeg/FHDTM-P2DS-WS2324/main/Data%20Science%20Projekt%20Demo/Datens%C3%A4tze/FHDTM-P2DS-WS2324-Project-Demo-1.1-Data-Acquisition-Transfermarkt_BVB.csv"
```

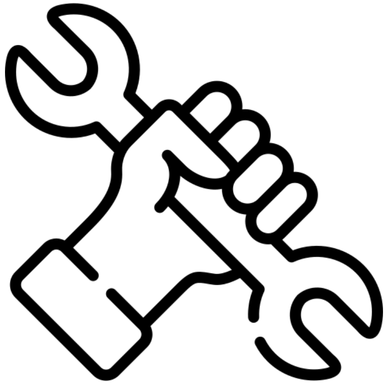
as a pandas data frame `pd.read_csv(url)`.

Explore the data frame and discuss with your peers **potential preprocessing steps**:

- Cleaning functions
- Data transformations, e.g., column generation, grouping, aggregations
- You can get creative, e.g., merge different data

…but always **explain why**, e.g., relevant to visualize.

Think-Pair-Share #1



Cleaning functions

Data transformations

Visualizations

Other

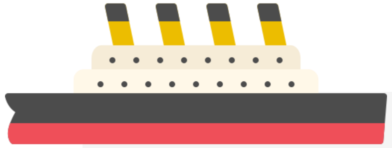
Getting to know your data (cont.)

Simple visualization tools and summary statistics are very useful!

- Make some plots
- Calculate summary statistics

...and think:

- Is the distribution consistent with the background knowledge? (You may need to consult domain experts)
- Any obvious outliers?
- Are some attributes heavily correlated with each other?



	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

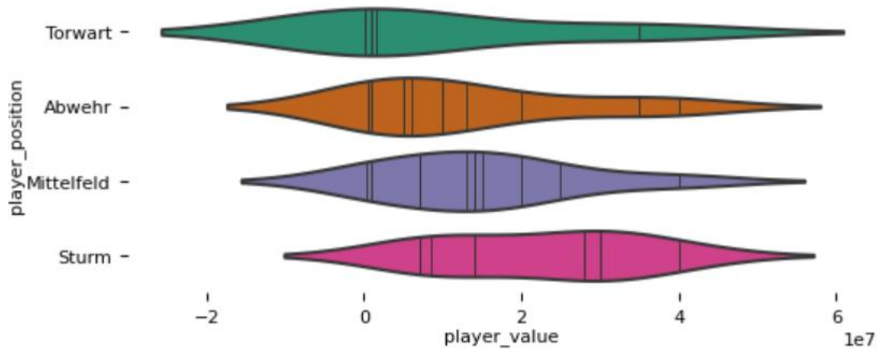
Takeaways

- Python and Pandas provide powerful toolkits for data science processes.
- Manual inspection and data preprocessing remains relevant.
- It is pretty hard to work, analyze and apply statistical methods on data **if you do not know anything about your data!**
- Take the time to open up your data file and have a look.
- You may **notice obvious** and **less obvious issues** with the data!

Outlook

- In the next week, we will dive deep into Preprocessing with Python Pandas.
- Similar to the functional approach of NumPy ufuncs, we can deploy functions for data reduction, data cleaning, data integration, and data transformations!

- In week 9 we will dive deep into Python Matplotlib to plot data:



Technology
Arts Sciences
TH Köln

See you again next week in person.

Questions?