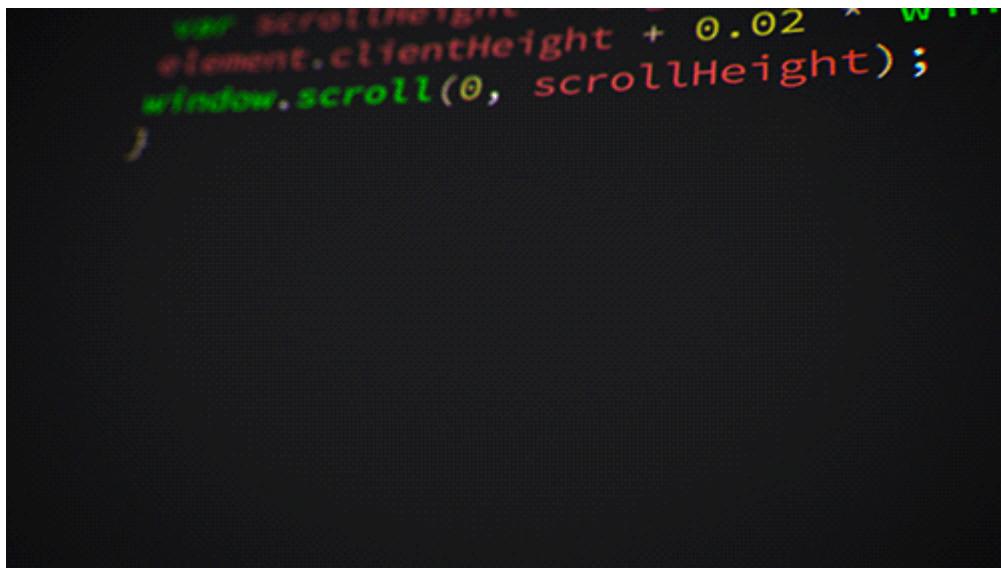


# Datenanalyse mit Python

Parameter	Kursinformationen
Veranstaltung:	<u>Prozedurale Programmierung / Einführung in die Informatik / Erhebung, Analyse und Visualisierung digitaler Daten</u>
Semester	Wintersemester 2025/26
Hochschule:	Technische Universität Freiberg
Inhalte:	Datenanalyse mit dem Python Paket Pandas
Link auf Repository:	<a href="https://github.com/TUBAF-IfI-LiaScript/VL_EAVD/blob/master/10_DatenAnalyse.md">https://github.com/TUBAF-IfI-LiaScript/VL_EAVD/blob/master/10_DatenAnalyse.md</a>
Autoren	Sebastian Zug & André Dietrich & Galina Rudolf & Bernhard Jung



Fragen an die heutige Veranstaltung ...

- Wie unterstützt das **numpy** Paket das Arbeiten mit großen wissenschaftlichen Datensätzen?
- Welche Datenformate sind für den Daten Austausch zwischen Mikrocontroller und Python Script üblich?
- Wie unterstützt das **pandas** Paket die wissenschaftliche Analyse von Datensätzen?
- Wie ändert sich der Analyseprozess verglichen mit der Verwendung einer Tabellenkalkulation?

## NumPy

NumPy (Numerical Python) ist eine grundlegende Bibliothek im Python-Ökosystem.

NumPy bietet leistungsstarke Werkzeuge für die effiziente Arbeit mit Arrays und Matrizen.

Stellen Sie es sich als Pythons Methode vor, mathematische Operationen auf großen Datensätzen schnell und effektiv durchzuführen.

NumPy wird insbesondere bei der Datenanalyse, beim wissenschaftlichem Rechnen, der Bildverarbeitung und maschinellem Lernen verwendet, oft im Zusammenspiel mit weiteren Bibliotheken wie `pandas` und `matplotlib`.

NumPy ist nicht Teil von Pythons Standardbibliothek und muss daher nachinstalliert werden. Hierfür gibt es verschiedene Methoden, die in den Übungen näher erklärt werden.

## NumPy: Effiziente, n-dimensionale Arrays

Ein wesentlicher Grund für die Existenz von NumPy liegt in der Unterstützung effizienter mathematischer Operationen auf n-dimensionalen Arrays (1D-Vektoren, 2D-Matrizen, nD-Tensoren).

- *N-dimensionale Arrays (ndarray):* NumPy Arrays sind wesentlich effizienter als Python-Listen bei der Durchführung mathematischer Operationen auf großen Datenmengen. Python-Listen können beliebige Datentypen enthalten, wodurch sie für numerische Berechnungen weniger optimiert sind.
- *Effiziente Implementierung in C:* NumPy ist in C implementiert und nutzt auch C-Datentypen für Zahlen wie `uint8` oder `float32`. In Python sind `int` und `float` dagegen keine primitiven Datentypen sondern vollwertige Objekte, was sie oft komfortabler, aber auch weniger effizient macht.
- *Elementweise Operationen:* NumPy ermöglicht mathematische Operation auf ganzen Arrays, ohne dass explizit Schleifen durchlaufen werden müssen (z.B. `for`-Schleifen). Dies nennt man auch *Vektorisierung*.

## numpy\_arrays.py



```
1 import numpy as np
2
3 # Ansatz mit Python Listen (langsam)
4 list1 = [1, 2, 3]
5 list2 = [4, 5, 6]
6 result_list = []
7 for i in range(len(list1)):
8     result_list.append(list1[i] + list2[i])
9 print(result_list) # Output: [5, 7, 9]
10
11 # Ansatz mit NumPy Arrays (schnell)
12 array1 = np.array([1, 2, 3])
13 array2 = np.array([4, 5, 6])
14 result_array = array1 + array2 # Elementweises Addieren!
15 print(result_array) # Output: [5 7 9]
```

```
[5, 7, 9]
[5 7 9]
[5, 7, 9]
[5 7 9]
```

NumPy wird näher in zwei einführenden *Jupyter Notebooks* erläutert (im Verzeichnis notebooks):

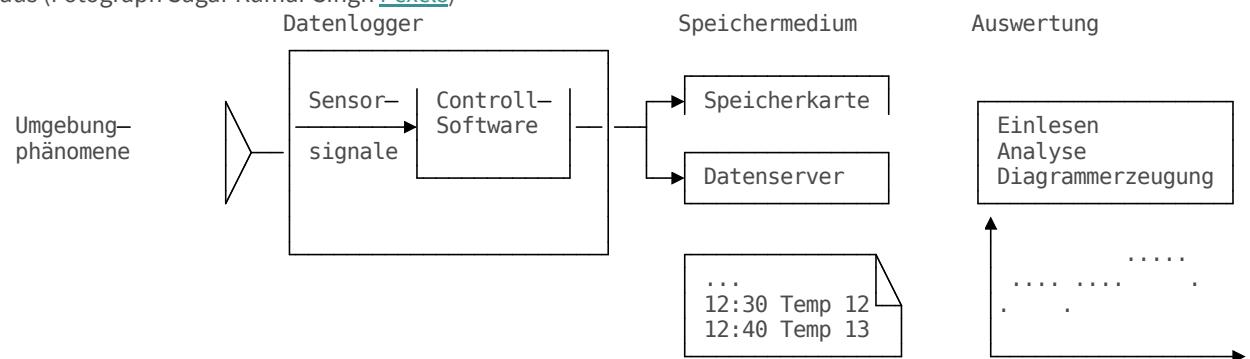
- Einführung in Numpy [numpy\\_01\\_intro.ipynb](#)
- Bildverarbeitung mit Numpy [numpy\\_02\\_image\\_processing.ipynb](#)

## Motivation: Datenanalyse mit Python / Pandas

Aufgabe: Dokumentieren Sie die zeitliche Verteilung des Erscheinen von Vögeln an einer Futterstelle. Zu welcher Tages / Nachtzeit ist die Aktivität am größten?



Vogelhaus (Photograph Sagar Kumar Singh [Pexels](#))



Und der Code für den Datenlogger? Wir werten den Beschleunigungssensor unseres Controllerboards aus.

```

#include "Sensor.h"
#include "RGB_LED.h"

DevI2C i2c(D14,D15);
LSM6DSLSensor sensor(i2c,D4,D5);
int xAxeData[3];

void setup() {
    Serial.begin(115200);           //Baudrate der seriellen Schnittstelle
    sensor.init(NULL);            //Start des Sensors
    sensor.enableAccelerator();   //Aktivierung des Beschleunigungssensors
}

void loop() {
    sensor.getXAxe(xAxeData);     //Lesen der Daten
    Serial.printf(";%d;%d;%d\n",xAxeData[0],xAxeData[1],xAxeData[2]); //Ausgabe der Daten via Serial
}
  
```

Aufgabe: Bewerten Sie die Implementierung! Welche Nachteile sehen Sie?

Für die Konfiguration des Zeitstempels im Visual Studio Code wurde der Parameter *Arduino: Change Timestamp Format* `%T.%L` [strftime Format](#) gesetzt.

Die Daten liegen als sogenannten *Comma-separated values* in einer csv Datei vor. Sie sind eine bequeme Möglichkeit, Daten aus Tabellenkalkulationen und Datenbanken zu exportieren und sie in andere Programme zu importieren oder zu verwenden. CSV-Dateien lassen sich sehr einfach programmatisch bearbeiten. Jede Sprache, die die Eingabe von Textdateien und die Manipulation von Zeichenketten unterstützt (wie Python), kann direkt mit CSV-Dateien arbeiten. Nachteilig ist, dass alle Inhalte als Text angelegt werden und damit verschwenderisch mit dem Speicher umgehen.

Als Trenner wurde hier das ; verwendet.

09:28:52.419; -7; -8; 1016  
09:28:52.430; -9; -8; 1017  
09:28:52.441; -9; -8; 1017  
09:28:52.452; -9; -8; 1017  
09:28:52.463; -16; -2; 1006  
09:28:52.474; -69; -160; 1057  
09:28:52.485; 58; 136; 984  
09:28:52.496; -10; -10; 1019  
09:28:52.507; -11; -6; 1012  
09:28:52.518; -5; 0; 1016  
09:28:52.528; -9; -15; 1013  
09:28:52.539; -9; -8; 1018  
09:28:52.551; -8; -9; 1016  
09:28:52.562; -8; -9; 1019

- Wie groß ist das normale Rauschen der Messwerte?
  - Wann wurde die größte Änderung der Beschleunigung gemessen?
  - Stellen Sie die Verlauf in einem Diagramm dar, benennen Sie die Achsen, erzeugen Sie ein Gitter.

**MEMO!** Arbeiten Sie bei der Analyse immer auf Kopien der eigentlichen Daten. Im Fall einer "Kompromitierung" durch eine einfache Schreiboperation haben Sie immer noch den Originaldatensatz zur Verfügung.

## Lösungsansatz 1: Office Tabellenkalkulation

```
data.csv
```

```
timestamp;X;Y;Z
09:28:52.419; -7; -8; 1016
09:28:52.430; -9; -8; 1017
09:28:52.441; -9; -8; 1017
09:28:52.452; -9; -8; 1017
```

Nutzen Sie die Importfunktion für csv-Dateien Ihres Office-Programms (Excel, LibreOffice Calc, ...)

## Lösungsansatz 2: Python nativ

Python kann die Textdateien umittelbar einlesen

1. Öffnen der Datei für das Lesen
2. Zeilenweises einlesen der Daten
  - Erfassen der Spaltennamen aus der ersten Zeile
  - Zerlegen anhand des **delimiter** (hier **;**)
  - Ablegen in einer vorbereiteten Datenstruktur
3. Schließen der Datei
  - durch **open()** ... **close()**
  - oder besser durch Verwendung von **with open() as**, was Datei automatisch schließt, siehe Beispiel unten.
4. Analyse der Daten

Diese Schrittfolge können wir mit dem Standardpaket [csv](#) etwas vereinfachen.

## data.csv

```
1 timestamp;X;Y;Z
2 09:28:52.419; -7; -8; 1016
3 09:28:52.430; -9; -8; 1017
4 09:28:52.441; -9; -8; 1017
5 09:28:52.452; -9; -8; 1017
```

## readCSV.py

```
1 import csv
2
3 # Einlesen der Daten
4 with open('data.csv', mode='r') as csv_file:
5     csv_reader = csv.DictReader(csv_file, delimiter=';')
6     list_of_dict = list(csv_reader)
7
8 # "Analyse" und Ausgabe
9 print(f"{len(list_of_dict)} Datensätze gelesen!")
10 for row in list_of_dict:
11     print(row)
```

```
4 Datensätze gelesen!
{'timestamp': '09:28:52.419', 'X': '-7', 'Y': '-8', 'Z': '1016'}
{'timestamp': '09:28:52.430', 'X': '-9', 'Y': '-8', 'Z': '1017'}
{'timestamp': '09:28:52.441', 'X': '-9', 'Y': '-8', 'Z': '1017'}
{'timestamp': '09:28:52.452', 'X': '-9', 'Y': '-8', 'Z': '1017'}
4 Datensätze gelesen!
{'timestamp': '09:28:52.419', 'X': '-7', 'Y': '-8', 'Z': '1016'}
{'timestamp': '09:28:52.430', 'X': '-9', 'Y': '-8', 'Z': '1017'}
{'timestamp': '09:28:52.441', 'X': '-9', 'Y': '-8', 'Z': '1017'}
{'timestamp': '09:28:52.452', 'X': '-9', 'Y': '-8', 'Z': '1017'}
```

**Aufgabe:** Bestimmen Sie die vorkommenden Maxima pro Spalte oder berechnen Sie die Differenz zwischen zwei benachbarten Werten einer Beschleunigungsachse.

## Lösungsansatz 3: Python mit Pandas

[pandas](#) ist eine für die Programmiersprache Python geschriebene Softwarebibliothek zur Datenmanipulation und -analyse, die insbesondere Datenstrukturen und Operationen zur Manipulation von numerischen Tabellen und Zeitreihen bietet. Es handelt sich um freie Software.

Der Name leitet sich von dem Begriff "*panel data*" ab, einem Begriff aus der Ökonometrie für Datensätze, die Beobachtungen über mehrere Zeiträume für dieselben Personen enthalten.

Der Code zum Paket kann unter [Link](#) eingesehen und bearbeitet werden.

**Achtung:** Mit der Verwendung von pandas ändert sich unser Blick auf den Code. Bislang haben wir Prozedural oder Objektorientiert programmiert. Jetzt ändert sich unser Blick - wir denken in Datenstrukturen und wenden Methoden darauf an.

## Pandas Grundlagen

Pandas kennt zwei grundsätzliche Datentypen [Series](#) und [DataFrame](#)

	<b>Pandas Series</b>	<b>Pandas DataFrame</b>
Format	Eindimensional	Zweidimensional (Tabellen)
Datentypen	Homogen - Reihenelemente müssen vom gleichen Datentyp sein.	Heterogen - DataFrame-Elemente können unterschiedliche Datentypen haben.
Zahl der Einträge	Größenunveränderlich - Einmal erstellt, kann die Größe nicht geändert werden.	Größenveränderlich - Elemente können in einem bestehenden DataFrame gelöscht oder hinzugefügt werden.

Wir betrachten zunächst die grundsätzliche Arbeitsweise für *Series* Daten.

## PandasSeries.py



```
1 import pandas as pd
2 import numpy as np
3
4 # Zufallszahlen mit dem Paket "numpy"
5 s_1 = pd.Series(np.random.randn(5))
6 print(s_1)
7
8 # Zufallszahlen und individuelle Indizes
9 s_2 = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
10 print(s_2)
11
12 # Für unseren Datensatz: Z Beschleunigungsdaten mit Zeitstempel als Index
13 data = {"09:28:52.419": 1016, "09:28:52.430": 1017, "09:28:52.441": 1018}
14 s_3 = pd.Series(data)
15 print(s_3)
```

```
0    -0.192189
1     0.769777
2     1.100136
3    -0.759419
4     0.252551
dtype: float64
a     0.949466
b     0.167207
c     0.591860
d     1.017129
e    -0.891083
dtype: float64
09:28:52.419    1016
09:28:52.430    1017
09:28:52.441    1017
dtype: int64
0    -0.117929
1    -0.349630
2     0.733030
3    -0.955494
4     0.114388
dtype: float64
a     0.609544
b     0.242113
c     0.398882
d    -1.987568
e    -0.511243
dtype: float64
09:28:52.419    1016
09:28:52.430    1017
09:28:52.441    1017
dtype: int64
```

**Achtung:** Im letztgenannten Beispiel [s\\_3](#) werden die Indizes nicht als Datum interpretiert sondern als Text. Realistisch wäre hier noch eine Transformation notwendig!

## PandasDataFrame.py



```
1 import pandas as pd
2 import numpy as np
3
4 # Multidimensionales DataFrame mit identischen Datentypen
5 df_1 = pd.DataFrame(np.random.randn(6, 4))
6 print(df_1)
7 print()
8
9 # Variables Set von Daten unterschiedlicher Typen
10 df_2 = pd.DataFrame(
11     {
12         "A": True,
13         "B": pd.date_range("20260101", periods=4),
14         "C": pd.Series(np.random.randn(4)),
15         "D": np.random.randint(16, size=4),
16         "E": pd.Categorical(["A", "A", "B", "C"]),
17         "F": "foo",
18     }
19 )
20 print(df_2)
```

```
      0          1          2          3
0  0.606212  0.372750  0.413261 -0.712787
1 -0.150516 -1.190214 -0.071595 -2.328077
2 -0.060530  0.747691 -1.165252  0.043770
3  0.774113  0.834756 -0.621202 -0.281807
4 -0.319336 -1.647127 -0.435454 -1.666276
5 -1.060656  1.999831  0.110499 -1.949763
```

```
      A          B          C          D          E          F
0  True 2026-01-01  0.809146    1    A    foo
1  True 2026-01-02  0.222920    2    A    foo
2  True 2026-01-03 -0.906289   15    B    foo
3  True 2026-01-04  0.854938   12    C    foo
```

```
      0          1          2          3
0  1.468204 -0.479060  0.492325  0.353875
1 -0.881489 -0.432173 -0.025121 -0.429988
2 -0.004018  0.923986  0.043741 -0.520779
3 -0.305170 -1.255613  0.340038  0.549451
4  0.556423  0.430142  0.639272 -0.431911
5  0.236408  1.577504  0.386692  0.096407
```

```
      A          B          C          D          E          F
0  True 2026-01-01 -0.376396    0    A    foo
1  True 2026-01-02 -0.256262    0    A    foo
2  True 2026-01-03  0.894212    7    B    foo
3  True 2026-01-04 -0.381825   10    C    foo
```

Aufgabe: Evaluieren Sie mittels `print(df_2.dtypes)` die realisierten Datentypen für `df_2`. Worüber "stolpern" Sie?

## Arbeiten mit Dataframes

Welche Aufgaben lassen sich nun mit Hilfe von Pandas über den Daten realisieren?

### Einblicke in die Struktur von Dataframes

## data.csv



```
1 timestamp;X;Y;Z
2 09:28:52.353; -8; -9; 1016
3 09:28:52.364; -9; -8; 1017
4 09:28:52.375; -9; -8; 1017
5 09:28:52.386; -8; -8; 1016
6 09:28:52.397; -9; -8; 1017
7 09:28:52.408; -9; -8; 1018
8 09:28:52.419; -9; -8; 1016
9 09:28:52.430; -9; -8; 1017
10 09:28:52.441; -9; -8; 1017
11 09:28:52.452; -9; -8; 1017
```

## df\_structure.py



```
1 import pandas as pd
2
3 # CSV-Datei einlesen
4 # Header-Info in erster Zeile (0 ist auch Default)
5 df = pd.read_csv('data.csv', header = 0, sep=";")
6
7 print("-- Anzahl Zeilen und Spalten:", df.shape)
8 print("-- Spaltennamen:", df.keys().values)
9 print("-- Datentypen der Spalten:\n", df.dtypes)
10 print("-- Erste 5 Zeilen:\n", df.head(5))
11 # print("-- Letzte 5 Zeilen:\n", df.tail(5))
12 # print(df) # Ausgabe des gesamten DataFrames
13 ..... # (kann bei grossen Dateien unübersichtlich sein)
```

```
-- Anzahl Zeilen und Spalten: (10, 4)
-- Spaltennamen: ['timestamp' 'X' 'Y' 'Z']
-- Datentypen der Spalten:
    timestamp      object
    X              int64
    Y              int64
    Z              int64
dtype: object
-- Erste 5 Zeilen:
    timestamp   X   Y     Z
0  09:28:52.353 -8  -9  1016
1  09:28:52.364 -9  -8  1017
2  09:28:52.375 -9  -8  1017
3  09:28:52.386 -8  -8  1016
4  09:28:52.397 -9  -8  1017
-- Anzahl Zeilen und Spalten: (10, 4)
-- Spaltennamen: ['timestamp' 'X' 'Y' 'Z']
-- Datentypen der Spalten:
    timestamp      object
    X              int64
    Y              int64
    Z              int64
dtype: object
-- Erste 5 Zeilen:
    timestamp   X   Y     Z
0  09:28:52.353 -8  -9  1016
1  09:28:52.364 -9  -8  1017
2  09:28:52.375 -9  -8  1017
3  09:28:52.386 -8  -8  1016
4  09:28:52.397 -9  -8  1017
```

## Indizierung

## data.csv



```
1 timestamp;X;Y;Z
2 09:28:52.353; -8; -9; 1016
3 09:28:52.364; -9; -8; 1017
4 09:28:52.375; -9; -8; 1017
5 09:28:52.386; -8; -8; 1016
6 09:28:52.397; -9; -8; 1017
7 09:28:52.408; -9; -8; 1018
8 09:28:52.419; -9; -8; 1016
9 09:28:52.430; -9; -8; 1017
10 09:28:52.441; -9; -8; 1017
11 09:28:52.452; -9; -8; 1017
```

## index.py



```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv', header = 0, sep=";")
4
5 # Auswahl von Spalten
6 s: pd.Series = df.timestamp # oder: s = df['timestamp']
7 print(s)
8 print("-- timestep 9:", df.timestamp[9])
9 print("-- Slice von timestamps:")
10 print(df.timestamp[5:8])
11
12 # Auswahl von Zeilen
13 print("-- Auswahl von Zeilen")
14 print(df[2:4])
```

```
0    09:28:52.353
1    09:28:52.364
2    09:28:52.375
3    09:28:52.386
4    09:28:52.397
5    09:28:52.408
6    09:28:52.419
7    09:28:52.430
8    09:28:52.441
9    09:28:52.452
Name: timestamp, dtype: object
-- timestep 9: 09:28:52.452
-- Slice von timestamps:
5    09:28:52.408
6    09:28:52.419
7    09:28:52.430
Name: timestamp, dtype: object
-- Auswahl von Zeilen
   timestamp    X    Y    Z
2  09:28:52.375  -9   -8  1017
3  09:28:52.386  -8   -8  1016
```

## Filtern

data.csv	
1	timestamp;X;Y;Z
2	09:28:52.353; -8; -9; 1016
3	09:28:52.364; -9; -8; 1017
4	09:28:52.375; -9; -8; 1017
5	09:28:52.386; -8; -8; 1016
6	09:28:52.397; -9; -8; 1017
7	09:28:52.408; -9; -8; 1018
8	09:28:52.419; -9; -8; 1016
9	09:28:52.430; -9; -8; 1017
10	09:28:52.441; -9; -8; 1017
11	09:28:52.452; -9; -8; 1017

## filter.py



```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv', header = 0, sep=";")
4 print(df, '\n')
5
6 # select columns by name
7 df2 = df.filter(items=["timestamp", "Z"])
8 print(df2, '\n')
9
10 # select rows where 'Z' > 1016
11 df3 = df['Z'] > 1016
12 print(df3, '\n')
13
14 above_1016 = df[df["Z"] > 1016]
15 print(above_1016, '\n')
```

```
    timestamp  X  Y      Z
0 09:28:52.353 -8 -9  1016
1 09:28:52.364 -9 -8  1017
2 09:28:52.375 -9 -8  1017
3 09:28:52.386 -8 -8  1016
4 09:28:52.397 -9 -8  1017
5 09:28:52.408 -9 -8  1018
6 09:28:52.419 -9 -8  1016
7 09:28:52.430 -9 -8  1017
8 09:28:52.441 -9 -8  1017
9 09:28:52.452 -9 -8  1017
```

```
    timestamp      Z
0 09:28:52.353 1016
1 09:28:52.364 1017
2 09:28:52.375 1017
3 09:28:52.386 1016
4 09:28:52.397 1017
5 09:28:52.408 1018
6 09:28:52.419 1016
7 09:28:52.430 1017
8 09:28:52.441 1017
9 09:28:52.452 1017
```

```
0    False
1     True
2     True
3    False
4     True
5     True
6    False
7     True
8     True
9     True
Name: Z, dtype: bool
```

```
    timestamp  X  Y      Z
1 09:28:52.364 -9 -8  1017
2 09:28:52.375 -9 -8  1017
4 09:28:52.397 -9 -8  1017
5 09:28:52.408 -9 -8  1018
7 09:28:52.430 -9 -8  1017
8 09:28:52.441 -9 -8  1017
```

```
9 09:28:52.452 -9 -8 1017
```

```
    timestamp  X  Y      Z
0 09:28:52.353 -8 -9 1016
1 09:28:52.364 -9 -8 1017
2 09:28:52.375 -9 -8 1017
3 09:28:52.386 -8 -8 1016
4 09:28:52.397 -9 -8 1017
5 09:28:52.408 -9 -8 1018
6 09:28:52.419 -9 -8 1016
7 09:28:52.430 -9 -8 1017
8 09:28:52.441 -9 -8 1017
9 09:28:52.452 -9 -8 1017
```

```
    timestamp      Z
0 09:28:52.353 1016
1 09:28:52.364 1017
2 09:28:52.375 1017
3 09:28:52.386 1016
4 09:28:52.397 1017
5 09:28:52.408 1018
6 09:28:52.419 1016
7 09:28:52.430 1017
8 09:28:52.441 1017
9 09:28:52.452 1017
```

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

```
Name: Z, dtype: bool
```

```
    timestamp  X  Y      Z
1 09:28:52.364 -9 -8 1017
2 09:28:52.375 -9 -8 1017
4 09:28:52.397 -9 -8 1017
5 09:28:52.408 -9 -8 1018
```

```
7 09:28:52.430 -9 -8 1017
8 09:28:52.441 -9 -8 1017
9 09:28:52.452 -9 -8 1017
```

## Statistische Beschreibung

Die Erzeugung deskriptiver Statistiken ist oft ein erster Schritt bei der Datenanalyse.

### data.csv



```
1 timestamp;X;Y;Z
2 09:28:52.353; -8; -9; 1016
3 09:28:52.364; -9; -8; 1017
4 09:28:52.375; -9; -8; 1017
5 09:28:52.386; -8; -8; 1016
6 09:28:52.397; -9; -8; 1017
7 09:28:52.408; -9; -8; 1018
8 09:28:52.419; -9; -8; 1016
9 09:28:52.430; -9; -8; 1017
10 09:28:52.441; -9; -8; 1017
11 09:28:52.452; -9; -8; 1017
```

### describe.py



```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv', header = 0, sep=";")
4 print(df.describe())
```

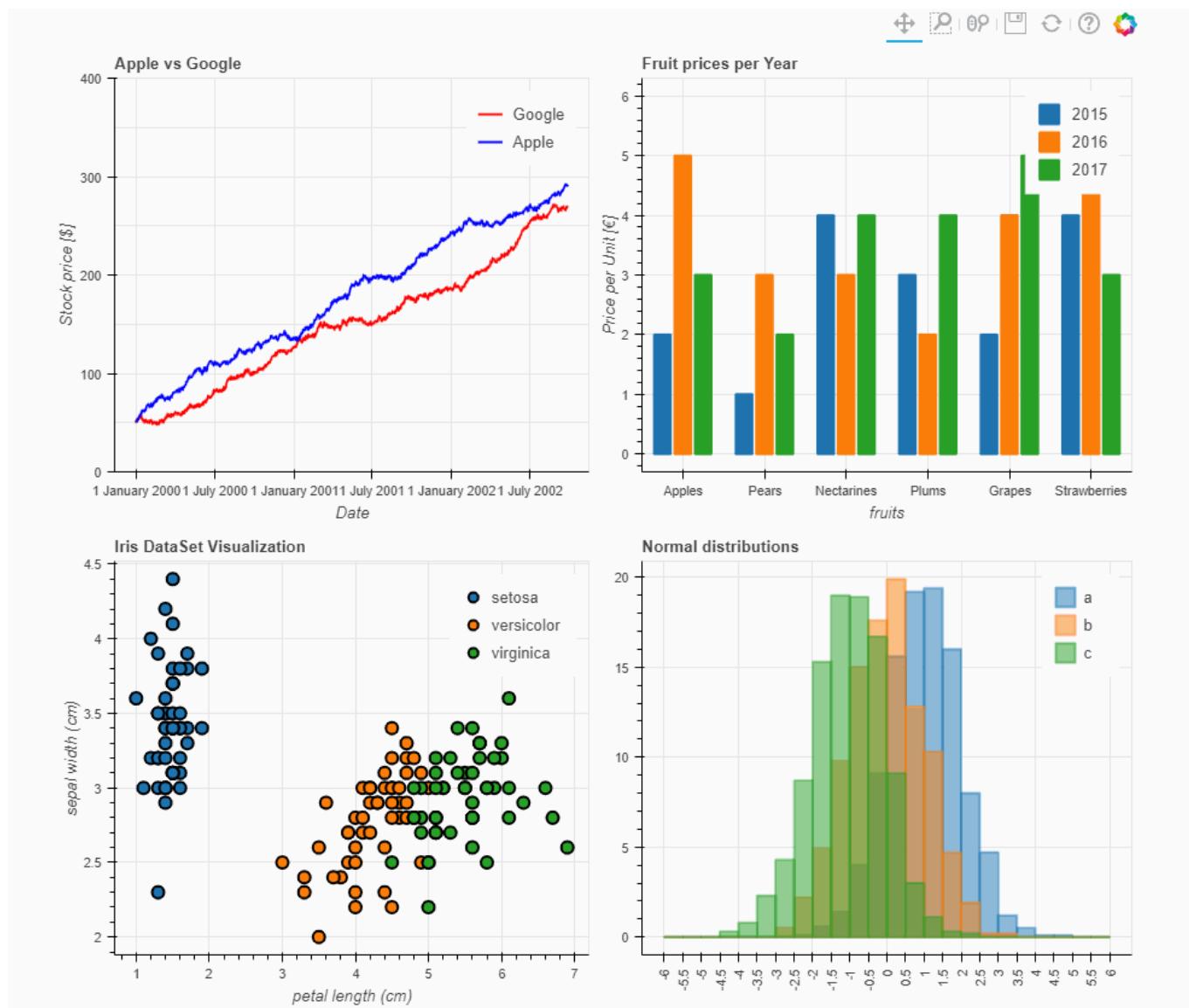
	X	Y	Z
count	10.000000	10.000000	10.000000
mean	-8.800000	-8.100000	1016.800000
std	0.421637	0.316228	0.632456
min	-9.000000	-9.000000	1016.000000
25%	-9.000000	-8.000000	1016.250000
50%	-9.000000	-8.000000	1017.000000
75%	-9.000000	-8.000000	1017.000000
max	-8.000000	-8.000000	1018.000000

	X	Y	Z
count	10.000000	10.000000	10.000000
mean	-8.800000	-8.100000	1016.800000
std	0.421637	0.316228	0.632456
min	-9.000000	-9.000000	1016.000000
25%	-9.000000	-8.000000	1016.250000
50%	-9.000000	-8.000000	1017.000000
75%	-9.000000	-8.000000	1017.000000
max	-8.000000	-8.000000	1018.000000

## Visualisierung mit pandas

Pandas ist unmittelbar mit der Bibliothek matplotlib verknüpft. Damit können wir die matplotlib-Methoden nahtlos nutzen.



Beispiele der Visualisierung von Pandas 'PatrikHlobil' [Link](#)

## data.csv



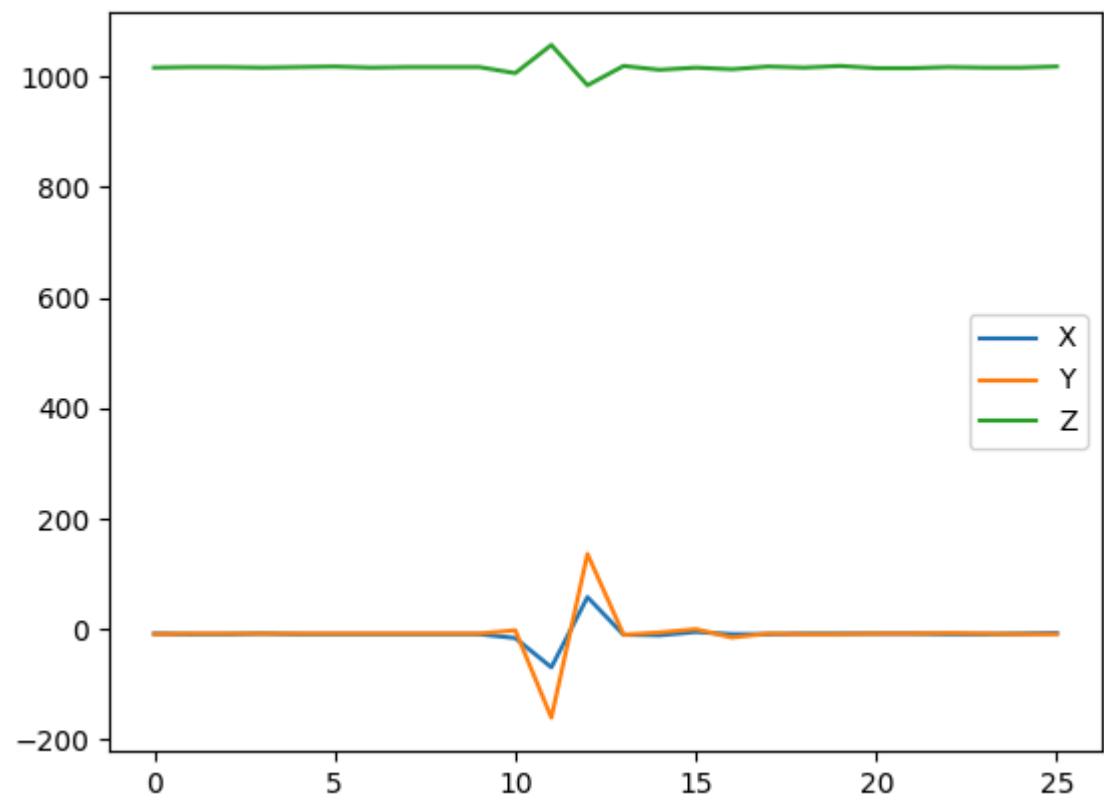
```
1 timestamp;X;Y;Z
2 09:28:52.353; -8; -9; 1016
3 09:28:52.364; -9; -8; 1017
4 09:28:52.375; -9; -8; 1017
5 09:28:52.386; -8; -8; 1016
6 09:28:52.397; -9; -8; 1017
7 09:28:52.408; -9; -8; 1018
8 09:28:52.419; -9; -8; 1016
9 09:28:52.430; -9; -8; 1017
10 09:28:52.441; -9; -8; 1017
11 09:28:52.452; -9; -8; 1017
12 09:28:52.463; -16; -2; 1006
13 09:28:52.474; -69; -160; 1057
14 09:28:52.485; 58; 136; 984
15 09:28:52.496; -10; -10; 1019
16 09:28:52.507; -11; -6; 1012
17 09:28:52.518; -5; 0; 1016
18 09:28:52.528; -9; -15; 1013
19 09:28:52.539; -9; -8; 1018
20 09:28:52.551; -8; -9; 1016
21 09:28:52.562; -8; -9; 1019
22 09:28:52.572; -8; -8; 1015
23 09:28:52.583; -8; -8; 1015
24 09:28:52.595; -9; -7; 1017
25 09:28:52.606; -9; -8; 1016
26 09:28:52.617; -8; -9; 1016
27 09:28:52.628; -7; -9; 1018
```

## readCSV.py

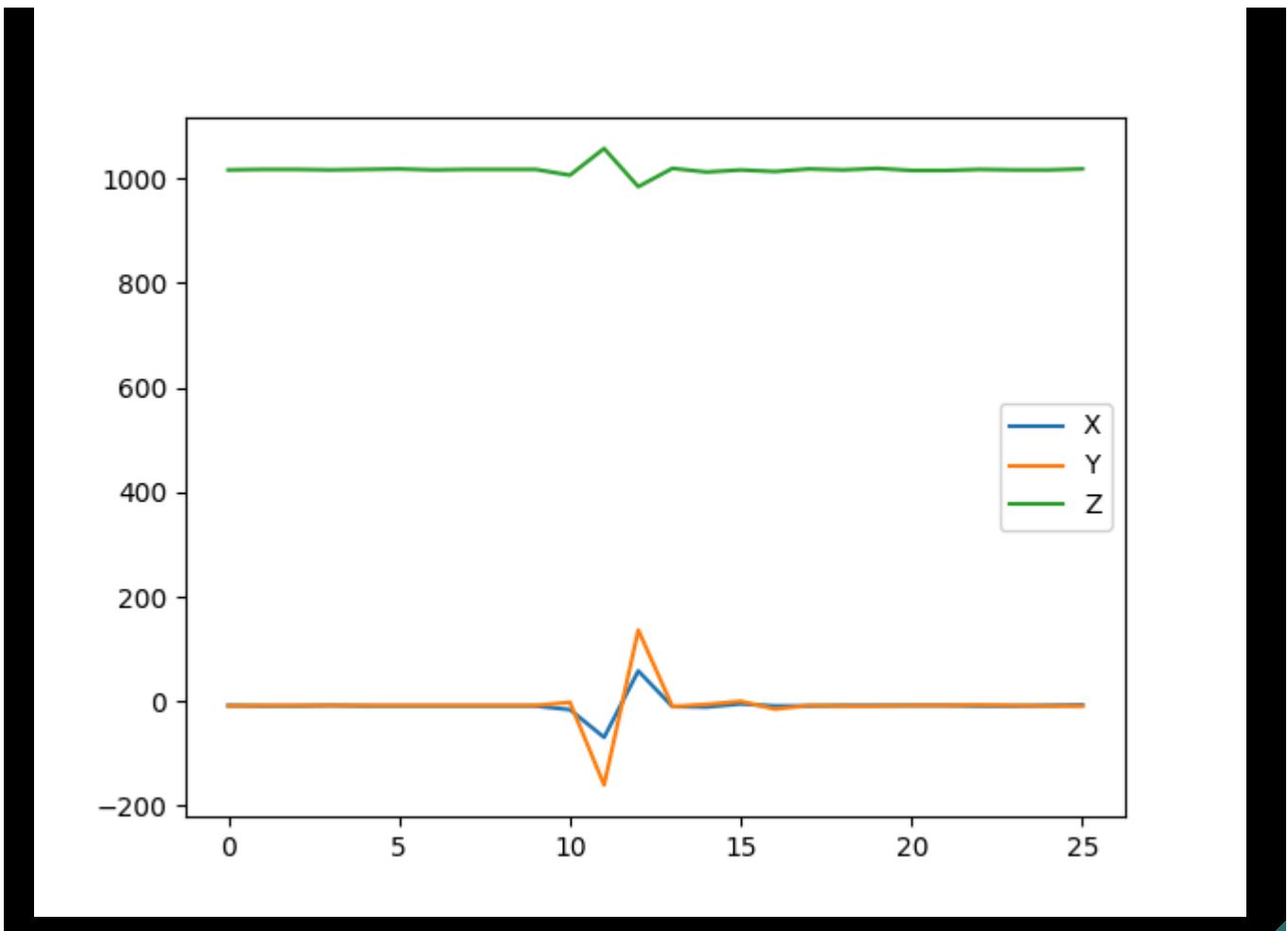


```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 df = pd.read_csv('data.csv', header = 0, sep=";")
5 df.plot()
6 plt.savefig('foo.png')
```

foo.png



foo.png



Anpassung	API	
Linientyp der Datendarstellung	<a href="#">pyplot.plot</a>	<code>plt.plot(a, b, 'ro:')</code>
Achsenlabel hinzufügen	<a href="#">pyplot.xlabel</a>	<code>plt.xlabel('my data', fontsize=14, color='red')</code>
Titel einfügen	<a href="#">pyplot.title</a>	<code>plt.title(r'\$\sigma_i=15\$')</code>
Gitter einfügen	<a href="#">pyplot.grid</a>	<code>plt.grid()</code>
Legende	<a href="#">pyplot.legend</a>	<code>plt.plot(a, b, 'ro:', label="Data")</code>
		<code>plt.legend()</code>
Speichern	<a href="#">pyplot.savefig</a>	<code>plt.savefig('foo.png')</code>

Die Bearbeitung der folgenden Aufgaben ist leichter, nachdem Sie sich mit den Inhalten der folgenden Vorlesung [Link L11](#) vertraut gemacht haben (bzw. noch eine Woche auf die Vorlesung warten 😊 ).

Aufgabe 1: Weisen Sie grafisch nach, dass es einen starken Zusammenhang zwischen den 3 Beschleunigungsdaten gibt!

Aufgabe 2: Geben Sie die Daten einer Achse in einem Histogramm aus! Schreiben Sie als Text den maximalen und den Minimalen Wert in die Mitte des Diagrams.

data.csv	
1	timestamp;X;Y;Z
2	09:28:52.353; -8; -9; 1016
3	09:28:52.364; -9; -8; 1017
4	09:28:52.375; -9; -8; 1017
5	09:28:52.386; -8; -8; 1016
6	09:28:52.397; -9; -8; 1017
7	09:28:52.408; -9; -8; 1018
8	09:28:52.419; -9; -8; 1016
9	09:28:52.430; -9; -8; 1017
10	09:28:52.441; -9; -8; 1017
11	09:28:52.452; -9; -8; 1017
12	09:28:52.463; -16; -2; 1006
13	09:28:52.474; -69; -160; 1057
14	09:28:52.485; 58; 136; 984
15	09:28:52.496; -10; -10; 1019
16	09:28:52.507; -11; -6; 1012
17	09:28:52.518; -5; 0; 1016
18	09:28:52.528; -9; -15; 1013
19	09:28:52.539; -9; -8; 1018
20	09:28:52.551; -8; -9; 1016
21	09:28:52.562; -8; -9; 1019
22	09:28:52.572; -8; -8; 1015
23	09:28:52.583; -8; -8; 1015
24	09:28:52.595; -9; -7; 1017
25	09:28:52.606; -9; -8; 1016
26	09:28:52.617; -8; -9; 1016
27	09:28:52.628; -7; -9; 1018

## ScatterPlot.py



```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 df = pd.read_csv('data.csv', header = 0, sep=";")
5 # Hier sind sie gefragt ...
6 plt.savefig('foo.png')
```

foo.png



foo.png

## Noch immer von Excel überzeugt?

- **Skalierbarkeit** - Pandas ist nur durch die Hardware begrenzt und kann größere Datenmengen verarbeiten. Excel ist aktuell auf 1.048.576 Zeilen und 16.384 Spalten beschränkt.
- **Geschwindigkeit** - Pandas arbeitet viel schneller als eine Tabellenkalkulation, was sich besonders bei der Arbeit mit größeren Datenmengen bemerkbar macht.
- **Automatisierung** - Viele der Aufgaben, die mit Pandas durchgeführt werden können, sind extrem einfach zu automatisieren, wodurch die Anzahl der langweiligen und sich wiederholenden Aufgaben, die täglich durchgeführt werden müssen, reduziert wird.
- **Interpretierbarkeit** - Eine Codesequenz aus Pandas ist übersichtlich und einfach zu interpretieren, da Tabellenkalkulationen Berechnungen pro Zelle ausführen, sind Fehler schwieriger zu identifizieren und zu beheben.
- **Erweiterte Funktionen** - Die Durchführung erweiterter statistischer Analysen und die Erstellung komplexer Visualisierungen ist sehr einfach.
- **Kompatibilität mit Excel-Dateien** - `xs/x`-Dateien können mit `read_excel()` eingelesen werden (d.h. ein Zwischenschritt mit Konvertierung zu CSV ist nicht notwendig).

<b>pandas</b>	<b>Tabellenkalkulation</b>
<code>DataFrame</code>	<i>worksheet</i>
<code>Series</code>	<i>column</i>
<code>Index</code>	<i>row headings</i>
<code>row</code>	<i>row</i>
<code>NaN</code>	empty cell

```
# Einlesen einer Excel Datei in Pandas
df = pd.read_excel("./myExcelFile.xlsx", index_col=0)

# Schreiben einer Excel Datei aus Pandas
df.to_excel("./myExcelFile.xlsx")
```



## Beispiel der Woche

Der Deutsche Wetterdienst bietet auf seinen [Webseiten](#) eine Vielzahl von historischen Datensätzen. Wir wollen unsere Pandas-Kenntnisse nutzen, um uns darin zu orientieren und dann "Licht in den Nebel bringen".

Die historischen Aufzeichnungen zu verschiedenen Stationen in Deutschland finden sich in der [Datenbank](#).

Aufgabe 1: Finden Sie die Stationsnummern von sächsischen Stationen in der Übersicht der Wetterstationen.

Den [Originaldatensatz](#) des deutschen Wetterdienstes können wir nicht verwenden - dieser ist als csv nicht ohne größeren Aufwand zu lesen. Daher wurde diese Datei aus didaktischen Gründen angepasst und liegt im Repository unter der angegebenen URL bereit.

## findweatherstations.py



```
1 import pandas as pd
2
3 url="https://raw.githubusercontent.com/" + \
4     "TUBAF-IfI-LiaScript/VL_ProzeduraleProgrammierung/" + \
5     "master/examples/11_DatenAnalyse/" + \
6     "Wetterdaten/wetter_tageswerte_Beschreibung_Stationen.txt"
7
8 # Datensatz laden
9 df=pd.read_csv(url, sep=';', header = 0)
10 #df=pd.read_csv("wetter_tageswerte_Beschreibung_Stationen.txt", sep='\
11 # Datensatz säubern
12 # Leerzeichen um Stationsnamen (39 Zeichen im Datensatz) entfernen, z
13 # ' Freiberg'                                ' -> 'Freiberg'
14 df['Stationsname'] = df['Stationsname'].str.strip()
15 df['Bundesland'] = df['Bundesland'].str.strip()
16
17 # Datensatz filtern (nur Sachsen) und ausgeben
18 df_sachsen = df[df.Bundesland == "Sachsen"] # nur Land Sachsen
19 print(df_sachsen)
```

	Stations_ID	Von_Datum	...	Stationsname
Bundesland				
16	131	20041101	...	Geringswalde-Altgeringswalde
Sachsen				
24	169	19470101	...	Annaberg-Buchholz
Sachsen				
41	222	19770101	...	Aue
Sachsen				
55	314	19450606	...	Kubschütz, Kr. Bautzen
Sachsen				
152	833	19720501	...	Neuhausen/Erzgeb.-Rauschenbach
Sachsen				
153	840	19900401	...	Carlsfeld
Sachsen				
157	853	18820101	...	Chemnitz
Sachsen				
161	876	19460801	...	Collmberg
Sachsen				
164	888	19810101	...	Crimmitschau
Sachsen				
176	965	19500301	...	Diesbar-Seußlitz
Sachsen				
181	991	19540901	...	Dippoldiswalde-Reinberg
Sachsen				
185	1003	19470101	...	Döbeln
Sachsen				
189	1047	18820101	...	Dresden (Mitte)
Sachsen				
190	1048	19340101	...	Dresden-Klotzsche
Sachsen				
191	1050	19490101	...	Dresden-Hosterwitz
Sachsen				
192	1051	19360101	...	Dresden-Strehlen
Sachsen				
218	1207	19480601	...	Elster, Bad-Sohl
Sachsen				
248	1358	18900801	...	Fichtelberg
Sachsen				
265	1441	19450701	...	Freiberg
Sachsen				
286	1583	19470101	...	Geisingberg
Sachsen				
303	1684	18670730	...	Görlitz

Sachsen				
307	1709	19520101	...	Gottleuba, Bad
Sachsen				
367	2166	19770101	...	Herrnhut
Sachsen				
377	2225	19220101	...	Hinterhermsdorf
Sachsen				
382	2252	19941201	...	Bertsdorf-Hörnitz
Sachsen				
448	2641	19941001	...	Klitzschen bei Torgau
Sachsen				
492	2928	18631201	...	Leipzig-Holzhausen
Sachsen				
493	2931	19570801	...	Leipzig-Mockau
Sachsen				
494	2932	19340101	...	Leipzig/Halle
Sachsen				
502	2985	19910102	...	Lichtenhain-Mittelndorf
Sachsen				
541	3166	19550801	...	Marienberg
Sachsen				
553	3234	19560601	...	Klipphausen-Garsebach
Sachsen				
587	3426	19360518	...	Muskau, Bad
Sachsen				
644	3788	19600601	...	Olbersdorf
Sachsen				
647	3811	19411001	...	Oschatz
Sachsen				
657	3883	19700301	...	Pausa-Unterreichenau
Sachsen				
666	3946	18820101	...	Plauen
Sachsen				
690	4149	19470101	...	Marienberg-Reitzenhain
Sachsen				
701	4222	19630101	...	Rodewisch
Sachsen				
762	4505	19510101	...	Schnarrtanne-Vogelsgrün
Sachsen				
782	4612	19360101	...	Schwarzenberg
Sachsen				
840	5068	19360101	...	Torgau
Sachsen				

875	5282	19170101	...	Wahnsdorf bei Dresden
Sachsen				
953	5779	19710101	...	Zinnwald-Georgenfeld
Sachsen				
954	5781	19390301	...	Zittau
Sachsen				
956	5797	18631204	...	Lichtentanne
Sachsen				
978	6095	19980501	...	Taubenheim-Seeligstadt
Sachsen				
983	6129	19990501	...	Sohland/Spree
Sachsen				
1009	6314	19490301	...	Nossen
Sachsen				
1025	7329	20051101	...	Treuen
Sachsen				
1028	7343	20060401	...	Deutschneudorf-Brüderwiese
Sachsen				
1048	13876	19220101	...	Altenberg-Raupennest
Sachsen				

[52 rows x 8 columns]

Bundesland	Stations_ID	Von_Datum	...	Stationsname
Sachsen	16	20041101	...	Geringswalde-Altgeringswalde
Sachsen	24	19470101	...	Annaberg-Buchholz
Sachsen	41	19770101	...	Aue
Sachsen	55	19450606	...	Kubschütz, Kr. Bautzen
Sachsen	152	19720501	...	Neuhausen/Erzgeb.-Rauschenbach
Sachsen	153	19900401	...	Carlsfeld
Sachsen	157	18820101	...	Chemnitz
Sachsen	161	19460801	...	Collmberg
Sachsen	164	19810101	...	Crimmitschau
Sachsen	176	19500301	...	Diesbar-Seußlitz

Sachsen				
181	991	19540901	...	Dippoldiswalde-Reinberg
Sachsen				
185	1003	19470101	...	Döbeln
Sachsen				
189	1047	18820101	...	Dresden (Mitte)
Sachsen				
190	1048	19340101	...	Dresden-Klotzsche
Sachsen				
191	1050	19490101	...	Dresden-Hosterwitz
Sachsen				
192	1051	19360101	...	Dresden-Strehlen
Sachsen				
218	1207	19480601	...	Elster, Bad-Sohl
Sachsen				
248	1358	18900801	...	Fichtelberg
Sachsen				
265	1441	19450701	...	Freiberg
Sachsen				
286	1583	19470101	...	Geisingberg
Sachsen				
303	1684	18670730	...	Görlitz
Sachsen				
307	1709	19520101	...	Gottleuba, Bad
Sachsen				
367	2166	19770101	...	Herrnhut
Sachsen				
377	2225	19220101	...	Hinterhermsdorf
Sachsen				
382	2252	19941201	...	Bertsdorf-Hörnitz
Sachsen				
448	2641	19941001	...	Klitzschen bei Torgau
Sachsen				
492	2928	18631201	...	Leipzig-Holzhausen
Sachsen				
493	2931	19570801	...	Leipzig-Mockau
Sachsen				
494	2932	19340101	...	Leipzig/Halle
Sachsen				
502	2985	19910102	...	Lichtenhain-Mittelndorf
Sachsen				
541	3166	19550801	...	Marienberg
Sachsen				

553	3234	19560601	...	Klipphausen-Garsebach
Sachsen				
587	3426	19360518	...	Muskau, Bad
Sachsen				
644	3788	19600601	...	Olbersdorf
Sachsen				
647	3811	19411001	...	Oschatz
Sachsen				
657	3883	19700301	...	Pausa-Unterreichenau
Sachsen				
666	3946	18820101	...	Plauen
Sachsen				
690	4149	19470101	...	Marienberg-Reitzenhain
Sachsen				
701	4222	19630101	...	Rodewisch
Sachsen				
762	4505	19510101	...	Schnarrtanne-Vogelsgrün
Sachsen				
782	4612	19360101	...	Schwarzenberg
Sachsen				
840	5068	19360101	...	Torgau
Sachsen				
875	5282	19170101	...	Wahnsdorf bei Dresden
Sachsen				
953	5779	19710101	...	Zinnwald-Georgenfeld
Sachsen				
954	5781	19390301	...	Zittau
Sachsen				
956	5797	18631204	...	Lichtentanne
Sachsen				
978	6095	19980501	...	Taubenheim-Seeligstadt
Sachsen				
983	6129	19990501	...	Sohland/Spree
Sachsen				
1009	6314	19490301	...	Nossen
Sachsen				
1025	7329	20051101	...	Treuen
Sachsen				
1028	7343	20060401	...	Deutschneudorf-Brüderwiese
Sachsen				
1048	13876	19220101	...	Altenberg-Raupennest

Sachsen

[52 rows x 8 columns]

Leider reicht der Freiberger Datensatz nur über wenige Jahre. Wir entscheiden uns für die weitere Untersuchung für die Daten aus Görlitz.

Aufgabe 2: Laden Sie den Görlitzer Datensatz in einen Dataframe und zählen Sie die Nebeltage. Visualisieren Sie das Ergebnis.

Der avisierte Datensatz für die Station "1684" [heruntergeladen](#). Die Datei

`wetter_tageswerte_01684_18800101_20181231_hist.zip` liefert als gepackter Ordner mehrere Datensets. Neben der eigentlichen Datendatei werden auch Stationskerndaten und Erhebungstechnik dokumentiert.

### extractFogObservations.py



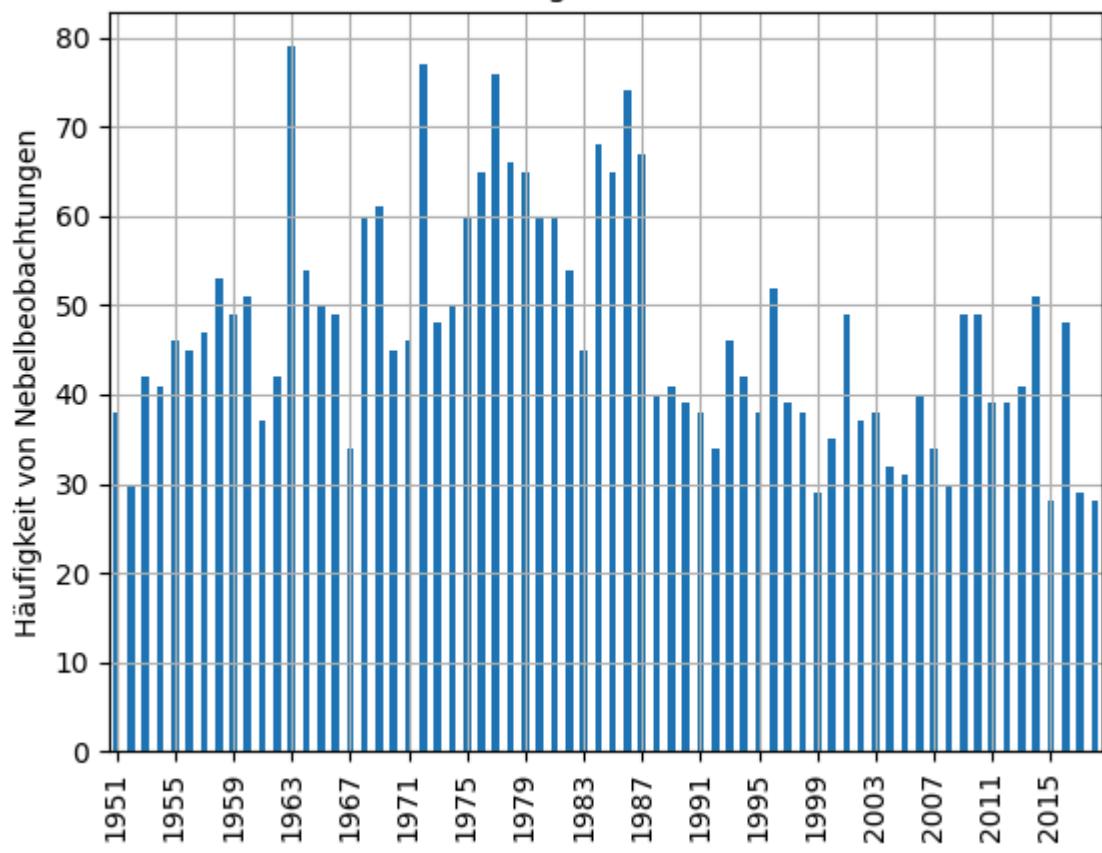
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 url="https://raw.githubusercontent.com/" + \
5     "TUBAF-IfI-LiaScript/VL_ProzeduraleProgrammierung/" + \
6     "master/examples/11_DatenAnalyse/" + \
7     "Wetterdaten/" + \
8     "wetter_tageswerte_01684_18800101_20181231_hist/" + \
9     "produkt_wetter_tag_18800101_20181231_01684.txt"
10
11 df=pd.read_csv(url, sep=';', header = 0)
12
13 df["JAHR"] = df["MESS_DATUM"] / 10000
14 df["JAHR"] = df["JAHR"].astype('int')
15
16 df_fog = df[df.NEBEL != -999]
17
18 print(f"\{df_fog.NEBEL.count()\} Tage im Datensatz, \{df_fog.NEBEL.sum()\} Nebel.")
19
20 ax = plt.axes()
21 df_fog.groupby("JAHR").NEBEL.sum().plot.bar(ax=ax)
22 ax.xaxis.set_major_locator(plt.MaxNLocator(20))
23 plt.ylabel("Häufigkeit von Nebelbeobachtungen")
24 plt.title("Nebeltage über Görlitz")
25 plt.grid()
26
27 #plt.show()
28 plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

24837 Tage im Datensatz, 3202 mit Nebel.

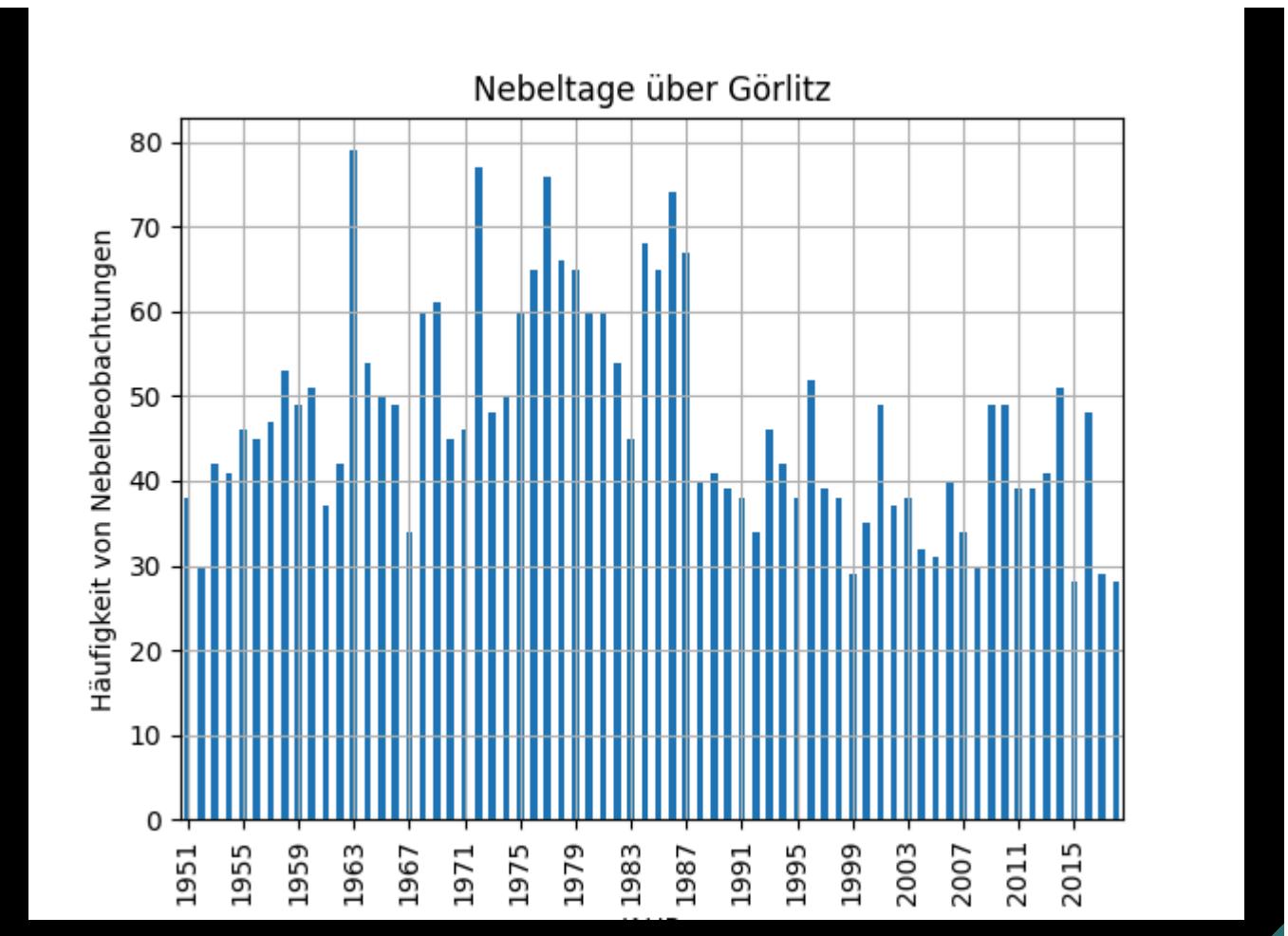
24837 Tage im Datensatz, 3202 mit Nebel.

foo.png

Nebeltage über Görlitz



foo.png



Aufgabe: Evaluieren Sie, welche Parameter sich in den vergangenen Jahren signifikant verändert haben.

## Quiz

### CSV-Dateien

Wofür steht CSV?

- Common System Variables
- Colorful Systematic Visualization
- Comma-separated values
- Critical Signal Version

## Python nativ

Wie lautet die Ausgabe des folgenden Programms, das die Daten aus der Datei data.csv einliest?

**data.csv**

```
timestamp;X;Y;Z  
09:28:52.419;-7;-8;1016  
09:28:52.430;-9;-8;1017  
09:28:52.441;-9;-8;1017  
09:28:52.452;-9;-8;1017
```

```
import csv
```

```
with open('data.csv', mode='r') as csv_file:  
    csv_reader = csv.DictReader(csv_file, delimiter=';')  
    list_of_dict = list(csv_reader)  
  
for row in list_of_dict:  
    print(row['X'], end=",")
```

## Pandas Grundlagen

Ordnen Sie *Pandas Series* und *Pandas Dataframes* die richtigen Eigenschaften zu.

<i>Pandas Series</i>	<i>Pandas Dataframe</i>	
<input type="radio"/>	<input type="radio"/>	Eindimensional
<input type="radio"/>	<input type="radio"/>	Zweidimensional
<input type="radio"/>	<input type="radio"/>	Heterogene Datentypen
<input type="radio"/>	<input type="radio"/>	Homogene Datentypen
<input type="radio"/>	<input type="radio"/>	Größenunveränderlich
<input type="radio"/>	<input type="radio"/>	Größenveränderlich

## Arbeit mit Dataframes

Wie lautet die Ausgabe dieses Programms?

```
import pandas as pd
a = [5, 7, 2, 7, 6]
s_1 = pd.Series(a, index=["a", "b", "c", "d", "e"])
print(s_1["c"])
```

Wie lautet die Ausgabe dieses Programms?

```
import pandas as pd
```

```
d = {  
    'A': [1,4,2,5],  
    'B': [2,5,3,6],  
    'C': [3,6,4,7],  
    'D': [4,7,5,8]  
}  
  
s_1 = pd.Series(d)  
print(s_1['C'][2])
```

Wie lautet die Ausgabe dieses Programms?

**data.csv**



```
title,FSK  
Toy Story,0  
Jumanji,12  
Grumpier Old Men,6  
Waiting to Exhale,12  
Father of the Bride Part II,0  
Heat,16  
Sabrina,6  
Tom and Huck,6  
Sudden Death,16  
GoldenEye,16  
The Amerian President,6  
Dracula: Dead and Loving It,12  
Balto,0  
Nixon,12  
Cutthroat Island,12
```

```
import pandas as pd  
  
df = pd.read_csv('data.csv')  
print(df['title'][9])
```

