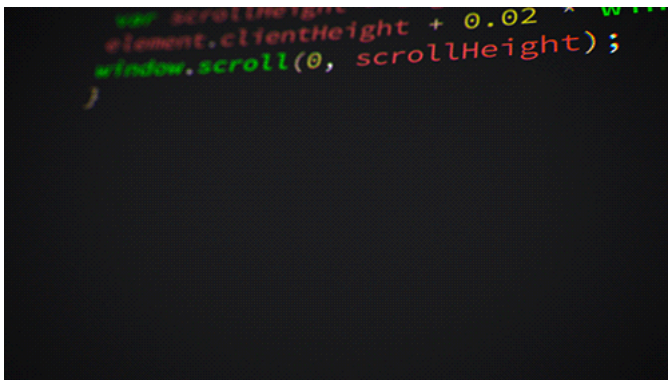


## Datenanalyse mit Python

Parameter	Kursinformationen
Veranstaltung:	Prozedurale Programmierung / Einführung in die Informatik
Semester	Wintersemester 2022/23
Hochschule:	Technische Universität Freiberg
Inhalte:	Datenanalyse mit dem Python Paket Pandas
Link auf Repository:	<a href="https://github.com/TUBAF-lfi-LiaScript/VL_ProzeduraleProgrammierung/blob/master/11_DatenAnalyse.md">https://github.com/TUBAF-lfi-LiaScript/VL_ProzeduraleProgrammierung/blob/master/11_DatenAnalyse.md</a>
Autoren	Sebastian Zug & André Dietrich & Galina Rudolf



### Fragen an die heutige Veranstaltung ...

- 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?

## Organisatorisches

Praktische Zusatzaufgabe für die Hörerinnen und Hörer der Einführung in die Informatik

- Entwickeln Sie eine Forschungsfrage, die sie mit Hilfe der in der Vorlesung vermittelten Programmierkenntnisse evaluieren wollen. Das Thema darf sich gern aus Ihrem Fachgebiet ergeben.
  - Reine Datenanalyse mit Python
  - Messdatenerhebung mit dem Mikrocontroller
  - Kombination aus Mikrocontrolleranwendung und Python Datenanalyse
- Fassen Sie Ihre Ergebnisse in einem kurzen Bericht zusammen. Dieser sollte
  - die Forschungsfrage motivieren
  - die Lösungsidee skizzieren
  - den Code enthalten
  - das Ergebnis dokumentieren
- Senden Sie uns das Dokument bis spätestens 31. März.

## Motivation

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



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

## Lösungsansatz 2: Python nativ

Python kann die Textdateien unmittelbar einlesen

1. Öffnen der Datei für das Lesen
2. Zeilenweises einlesen der Daten
  - Erfassen der Spaltennamen aus der ersten Zeile
  - Zerlegen anhand des `delimiters` (hier `;`)
  - Ablegen in einer vorbereiteten Datenstruktur
3. Schließen der Datei
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'}
```

**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](#)

	Pandas Series	Pandas DataFrame
Format	Eindimensional	Zweidimensional
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 und die Z Beschleunigungsdaten
13 data = {"09:28:52.419": 1016, "09:28:52.430": 1017, "09:28:52.441": 1017}
14 s_3 = pd.Series(data)
15 print(s_3)
```

```
0    0.752101
1   -0.227179
2   -0.646796
3    0.180126
4   -0.700289
dtype: float64
a   -2.057072
b    1.022910
c    0.828263
d   -2.544107
e    1.562343
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 sonder 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("20230101", 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 -0.083884 -1.064866  0.522014  0.560038
1 -0.104425 -0.710833  1.278256 -1.652493
2 -0.249366  0.910408  1.086682  0.780340
3  3.309983 -0.816514 -1.035340 -1.628887
4  1.470745 -1.476876  1.433273  2.415463
5 -0.869039  1.674031  2.029746 -0.769306

```

```

      A      B      C  D  E  F
0  True 2023-01-01 -0.812196  8  A  foo
1  True 2023-01-02 -0.115551  6  A  foo
2  True 2023-01-03 -0.132752  2  B  foo
3  True 2023-01-04  0.435237  4  C  foo

```

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

## Arbeit mit Dataframes

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

### 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 print(df)

```

```

      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

```

### 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)
```

```
   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
```

### Statistische Beschreibung

#### 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
```

### Nutzung

#### 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)
```

```
   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
```

## Visualisierung mit pandas

Pandas ist unmittelbar mit der schon bekannten Bibliothek matplotlib verknüpft. Damit können wir unsere bereits bekannten Methoden nahtlos nutzen.

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



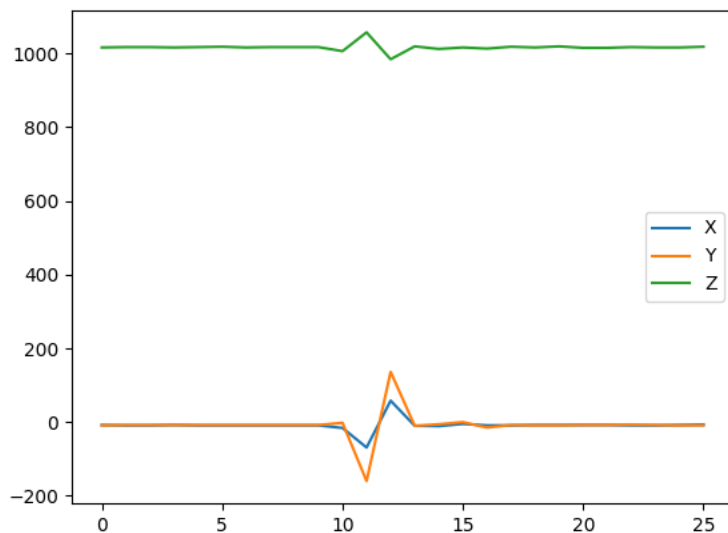
## data.csv

```
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



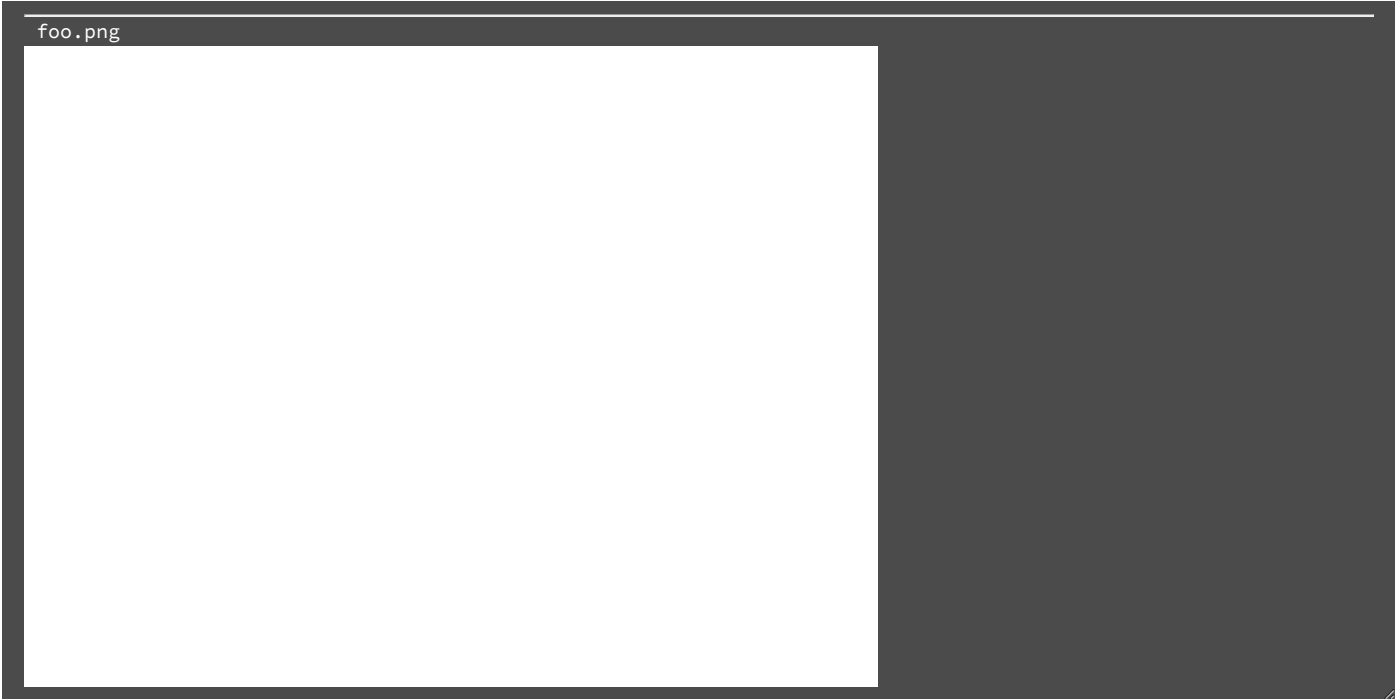
Anpassung	API	
Linientyp der Datendarstellung	<a href="#">pyplot.plot</a>	<code>plt.plot(a, b, 'ro:')</code>
Achsenlabel hinzufügen	<a href="#">pyplot.xlabel</a>	<code>plt.xlabel('my data', fontsize=14, color='red')</code>
Titel einfügen	<a href="#">pyplot.title</a>	<code>plt.title(r'\$\sigma_i=15\$')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

Merke: Mit dem zusätzlichen Parameter `style='o:'` können Sie die Konfiguration der Darstellung anpassen.

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



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.

pandas	Tabellenkalkulation
DataFrame	worksheet
Series	column
Index	row headings
row	row
NaN	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 df=pd.read_csv(url, sep=';', header = 0)
9 #df=pd.read_csv("wetter_tageswerte_Beschreibung_Stationen.txt", sep=';',
10     header = 0)
11 df['Bundesland'] = df['Bundesland'].str.strip()
12 df['Stationsname'] = df['Stationsname'].str.strip()
13 df_sachsen = df[df.Bundesland == "Sachsen"]
14 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]

Leider reicht der Freiburger 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

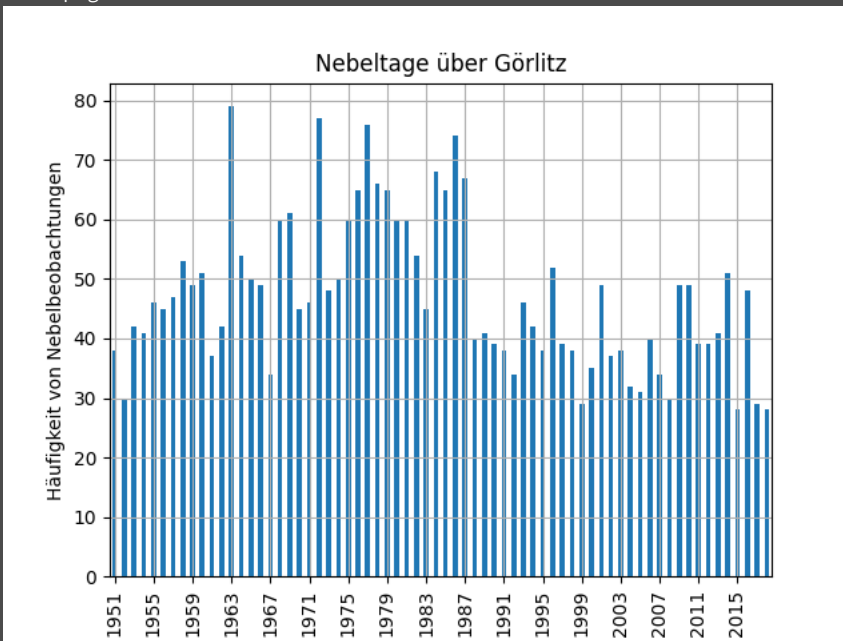
```

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()} mit
19     Nebel.")
20
21 ax = plt.axes()
22 df_fog.groupby("JAHR").NEBEL.sum().plot.bar(ax=ax)
23 ax.xaxis.set_major_locator(plt.MaxNLocator(20))
24 plt.ylabel("Häufigkeit von Nebelbeobachtungen")
25 plt.title("Nebeltage über Görlitz")
26 plt.grid()
27
28 #plt.show()
29 plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript

```

24837 Tage im Datensatz, 3202 mit Nebel.

foo.png



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