# Daten importieren

by Woche 7

Wir haben gerade erst DataFrames kennengelernt und müssen noch so einige Dinge über deren Handhabung lernen. Gleichzeitig dürfte es vielen schon in den Fingern kribbeln, endlich mal mit “echten” Daten zu arbeiten. Denn natürlich werden in realen Projekte die Daten meist nicht in Python erzeugt, sondern aus bereits vorhandenen Quellen importiert. Somit wollen wir lernen Daten aus zumindest einigen gängigen dieser Quellen zu importieren.

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

## CSV-Dateien

CSV-Dateien sind wohl die am häufigsten verwendete Art von Dateien, um tabellarische Daten zu speichern. CSV steht für “Comma Separated Values” und bedeutet, dass die Werte in der Datei durch Kommas getrennt sind. Vorteile von CSV sind, dass sie einfach zu erstellen und zu lesen sind und, dass sie von vielen Programmen unterstützt werden.

So lässt sich also beispielsweise folgende Tabelle…

| Land | Gericht | Kurzbeschreibung |
|---|---|---|
| Vietnam | Bún chả | Gegrilltes Schweinefleisch mit Reisnudeln und Kräutern |
| Italien | Pizza Napoli | Pizza mit Kapern, Sardellen und Oliven |
| Japan | Sushi | Reisbällchen oder -rollen mit Fisch oder Gemüse |
| Mexiko | Chiles en nogada | Gefüllte Poblano-Paprika mit Walnusssauce und Granatapfelkernen |

…so als CSV-Datei speichern:

```
Land,Gericht,Kurzbeschreibung
Vietnam,Bún chả,"Gegrilltes Schweinefleisch mit Reisnudeln und Kräutern"
Italien,Pizza Napoli,"Pizza mit Kapern, Sardellen und Oliven"
Japan,Sushi,"Reisbällchen oder -rollen mit Fisch oder Gemüse"
Mexiko,Chiles en nogada,"Gefüllte Poblano-Paprika mit Walnusssauce und
Granatapfelkernen"
```

In der Regel sind also die Spaltennamen in der ersten Zeile und die Werte in den folgenden Zeilen. Die Werte sind durch Kommas getrennt und Texte, die Kommas enthalten, werden in Anführungszeichen gesetzt.

Tatsächlich ist es leider nicht immer genau so, da es keine feste Norm gibt was eine CSV-Datei ist. Erstellt ihr beispielsweise eine Tabelle in Microsoft Excel und speichert sie dann als `.csv` Datei ab, so seht ihr schon beim Speichern, dass dort als Auswahlmöglichkeit steht `CSV (Trennzeichen-getrennt) (*.csv)`, also nicht **Komma-getrennt**, sondern **Trennzeichen-getrennt**. Öffnet ihr dann die gespeicherte Datei in einem Texteditor, so seht ihr, dass dort tatsächlich kein Komma, sondern ein Semikolon als Trennzeichen verwendet wird. Auch das gilt also als CSV-Datei und eben diese fehlende Standardisierung ist ein Nachteil von CSV-Dateien.

# Import

Zum Importieren von CSV-Dateien in einen DataFrame verwenden wir die Funktion `pd.read_csv()`. Diese Funktion hat viele Argumente, aber das wichtigste ist das erste: `filepath_or_buffer`. Hier gebt ihr den Pfad zur CSV-Datei an, die ihr importieren wollt. Um den Einstieg möglichst leicht zu machen, wollen wir hier keinen lokalen Pfad, sondern eine URL angeben. Das heißt, dass der Pfad nicht zu einer Datei führt, die auf meinem oder eurem Computer liegt, sondern zu einer CSV-Datei, die online verfügbar ist. So könnt auch ihr diesen Code sofort laufen lassen - gegeben ihr habt eine funktionierende Internetverbindung. Ihr könnt übrigens den Link, der hierunter in die Variable `pfad_zu_csv_datei` gespeichert wird, auch tatsächlich in eurem Browser öffnen und die Daten sehen.

```
pfad_zu_csv_datei = 'https://raw.githubusercontent.com/SchmidtPaul/
ExampleData/main/plant_growth/PlantGrowth.csv'
df = pd.read_csv(pfad_zu_csv_datei)
df
```

```
    rownames  weight group
0          1    4.17  ctrl
1          2    5.58  ctrl
2          3    5.18  ctrl
3          4    6.11  ctrl
4          5    4.50  ctrl
5          6    4.61  ctrl
6          7    5.17  ctrl
7          8    4.53  ctrl
8          9    5.33  ctrl
9         10    5.14  ctrl
10        11    4.81  trt1
11        12    4.17  trt1
```

```
12         13    4.41  trt1
13         14    3.59  trt1
14         15    5.87  trt1
15         16    3.83  trt1
16         17    6.03  trt1
17         18    4.89  trt1
18         19    4.32  trt1
19         20    4.69  trt1
20         21    6.31  trt2
21         22    5.12  trt2
22         23    5.54  trt2
23         24    5.50  trt2
24         25    5.37  trt2
25         26    5.29  trt2
26         27    4.92  trt2
27         28    6.15  trt2
28         29    5.80  trt2
29         30    5.26  trt2
```

Natürlich kann man die URL auch direkt in `pd.read_csv()` einfügen, ohne sie vorher in einer Variable (`pfad_zu_csv_datei`) zu speichern.

Da wir ab jetzt öfter Tabellen anschauen, die mehr als nur eine Handvoll Zeilen haben, ist es in der Regel effizienter, sich nur die ersten paar Zeilen anzeigen zu lassen. Das geht mit der Funktion `.head()`, welche standardmäßig nur die ersten 5 Zeilen anzeigt. Ihr könnt aber auch eine andere Zahl angeben, um mehr oder weniger Zeilen anzuzeigen. So könnt ihr beispielsweise die ersten 10 Zeilen anzeigen lassen, indem ihr `df.head(10)` schreibt. Und da man so quasi den “Kopf” der Daten betrachtet, gibt es auch eine Funktion `.tail()`, die den “Schwanz” der Daten anzeigt, also die letzten Zeilen.

```
df.head()
```

```
   rownames  weight group
0         1    4.17  ctrl
1         2    5.58  ctrl
2         3    5.18  ctrl
3         4    6.11  ctrl
4         5    4.50  ctrl
```

```
df.tail(3)
```

```
    rownames  weight group
27        28    6.15  trt2
```

```
28        29    5.80  trt2
29        30    5.26  trt2
```

Wie schon gesagt, gibt es viele Argumente, die ihr `pd.read_csv()` übergeben könnt. Hier ein Screenshot der Online-Dokumentation:


*Quelle: pandas.pydata.org*

Da wir eben erfolgreich eine CSV-Datei importiert haben, waren also die Standardwerte/ Defaults für die Argumente ausreichend. Nun wollen wir aber eine zweite Version der Daten importieren, die zwar ebenfalls in einer CSV-Datei gespeichert sind, aber unüblich formatiert sind. Zum Vergleich sind hier die ersten 6 Zeilen der beiden CSV-Dateien - links die erste, rechts die zweite:

```
rownames,weight,group
1,4.17,ctrl
2,5.58,ctrl
3,5.18,ctrl
4,6.11,ctrl
5,4.5,ctrl
```

```
Hier sind meine Daten
rownamesWUFFweightWUFFgroup
1WUFF4,17WUFFctrl
```

```
2WUFF5,58WUFFctrl
3WUFF5,18WUFFctrl
4WUFF6,11WUFFctrl
```

Würden wir wieder versuchen die Datei mit `pd.read_csv()` und ohne weitere Argumente zu importieren, so würden wir eine Fehlermeldung erhalten. Das liegt daran, dass die Datei nicht dem Standardformat entspricht. Wir müssen also die Argumente von `pd.read_csv()` so anpassen, dass sie zu den Daten passen. In diesem Fall sollten wir mindestens drei Dinge tun:

- `skiprows=1`: Die erste Zeile enthält keine Daten, sondern nur den Text “Hier sind meine Daten”. Wir wollen also, dass `pd.read_csv()` diese eine, erste Zeile überspringt.
- `sep='WUFF'`: Die Werte sind nicht durch Kommas, sondern durch “WUFF” getrennt. Das können wir `pd.read_csv()` mitteilen, indem wir das Argument `sep` (Separator) verwenden.
- `decimal=','`: In der zweiten Datei werden die Dezimalzahlen mit einem Komma statt einem Punkt geschrieben, also `4,17` anstatt `4.17`. Das ist im deutschsprachigen Raum die Norm, aber weltweit gesehen eher unüblich.

```
# URL zur unübliche formatierten CSV-Datei
pfad_2 = 'https://raw.githubusercontent.com/SchmidtPaul/ExampleData/main/
plant_growth/PlantGrowth2.csv'
```

```
pd.read_csv(pfad_2)
```

```
pandas.errors.ParserError: Error tokenizing data. C error: Expected 1 fields
in line 3, saw 2
```

```
df2 = pd.read_csv(
  pfad_2,
  skiprows=1,
  sep='WUFF',
  decimal=','
)

df2.head()
```

```
   rownames  weight group
0         1    4.17  ctrl
1         2    5.58  ctrl
2         3    5.18  ctrl
```

```
3         4    6.11  ctrl
4         5    4.50  ctrl
```

Nun wird man `WUFF` wohl eher nicht als Trennzeichen in einer CSV-Datei finden, aber es gibt mehr Trennzeichen als man glaubt. Oben bereits erwähnt ist das Semikolon `;` und eine ebenfalls gängiges Dateiformat ist die Tabulator-getrennte Text-Datei (`.txt`), bei der die Werte durch Tabs getrennt sind. Das Argument `sep` kann also auch ein Tabulatorzeichen sein, das man mit `\t` angibt.

`sep=';'`

```
rownames;weight;group
1;4.17;ctrl
2;5.58;ctrl
3;5.18;ctrl
4;6.11;ctrl
5;4.5;ctrl
```

`sep='\t'`

```
rownames    weight  group
1   4.17    ctrl
2   5.58    ctrl
3   5.18    ctrl
4   6.11    ctrl
5   4.5 ctrl
```

Letztendlich ist es auch bzgl. anderer Abweichungen von der Norm gut zu wissen, dass `pd.read_csv()` so flexibel ist. Die eigentliche Datei vorher/außerhalb von Python reparieren zu müssen ist also in den meisten Fällen nicht notwendig.

## Lokale Dateien

---

Natürlich kann und muss man auch lokale Dateien importieren, also Dateien die auf der Festplatte des eigenen Computers gespeichert sind. Prinzipiell ist hier alles gleich - bis auf den Pfad. Ein Pfad zu einer Datei namens `datei.csv`, die auf dem Desktop liegt, könnte je nach Betriebssystem so aussehen:

- Windows: `C:\Users\Benutzername\Desktop\datei.csv`
- macOS: `/Users/Benutzername/Desktop/datei.csv`
- Linux: `/home/Benutzername/Desktop/datei.csv`

Und tatsächlich muss man dann statt der URL lediglich diesen Pfad in `pd.read_csv()` einfügen und fertig. Das Problem bei diesen Pfaden ist aber, dass sie nicht portabel

sind. Das heißt, dass sie nur auf dem Computer funktionieren, auf dem sie erstellt wurden. Wenn ihr also euren Code an jemand anderen weitergebt, dann wird dieser Pfad nicht funktionieren. Ebenso wird der Beispielcode oben nicht bei euch funktionieren selbst wenn ihr eine `datei.csv` auf eurem Desktop gespeichert habt und den Pfad für das korrekte Betriebssystem kopiert, weil ihr wahrscheinlich nicht als “Benutzername” angemeldet seid.

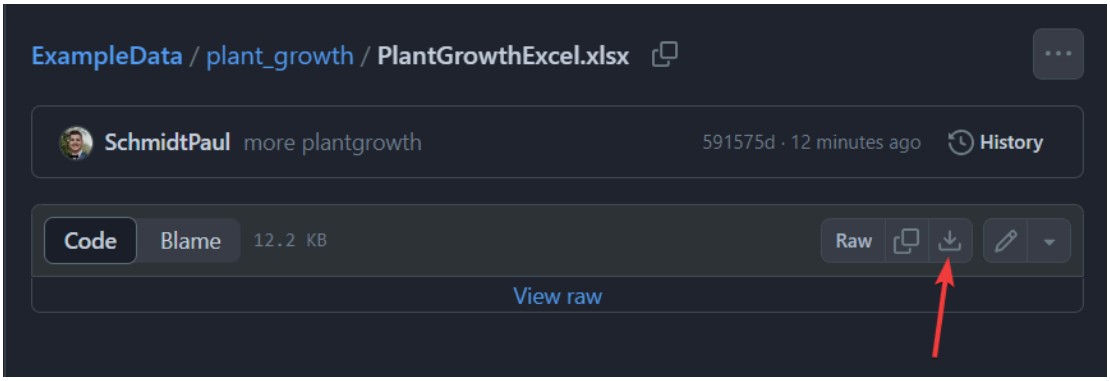
In diesem Video wird gezeigt wie man eine lokale Datei in Python einliest:

https://www.youtube.com/embed/xp76FTNkOdQ?si=bu_TOq-GS7m0DsVO

> ℹ Dateipfad auf Mac kopieren
>
> Um einen Dateipfad auf dem Mac zu kopieren, könnt ihr laut dieser Quelle wie folgt vorgehen: *“Öffne im ersten Schritt ein Finder-Fenster, indem du im Dock auf das Finder-Symbol klicken. Navigiere anschließend zu dem gewünschten Ordner oder der Datei und halte dort die Steuertaste (ctrl/control) gedrückt, während du den Ordner oder die Datei anklickst. Alternativ kannst du auch einfach einen Rechtsklick vornehmen, um das Kontextmenü aufzurufen. Drücke jetzt die Optionstaste (alt) auf der Tastatur. Dies ändert einige Optionen des Kontextmenüs und erlaubt dir damit den Datei- oder Ordnerpfad über die Option „„Datei-/Ordnername“ als Pfadname kopieren“ zu kopieren. Der Pfadname wird in die Zwischenablage übernommen und kann mit die Tastenkombination Befehlstaste (cmd) + (v) wieder eingefügt werden.”*

# Excel

Excel-Dateien sind ebenfalls eine häufige Art von Dateien, um tabellarische Daten zu speichern. Excel ist ein sehr mächtiges Programm und kann viele verschiedene Arten von Daten speichern. Die einfachste Art von Excel-Dateien sind `.xls` und `.xlsx` Dateien. `.xls` Dateien sind die ältere Version und `.xlsx` Dateien sind die neuere Version. Ein Grund für die Popularität von Excel-Dateien ist, dass sie von sehr vielen Anwendern und Anwenderinnen verwendet werden - also auch von denen, die keine Ahnung von Programmierung haben. In einer Excel-Datei wird im Vergleich zu einer CSV-Datei natürlich noch mehr Information gespeichert, wie beispielsweise die Schriftart, die Farbe, die Größe und die Ausrichtung des Textes. Allein schon die Möglichkeit mehrere Tabellenblätter in einer Datei zu haben, sollte zeigen, dass Excel-Dateien komplexer sind als CSV-Dateien. Dennoch ist es natürlich notwendig und möglich, Daten aus Excel-Dateien in Python zu importieren und zwar z.B. mit der Funktion `pd.read_excel()`. Allerdings funktioniert der Import von Excel-Dateien nicht so reibungslos direkt über eine URL wie bei CSV-Dateien. Ihr müsst also die Datei herunterladen und lokal speichern, um sie dann zu importieren. Die Datei mit der wir hier arbeiten, kann hier heruntergeladen werdem.

> **Download einzelner Dateien von GitHub**
>
> GitHub lernen wir später noch genauer kennen. Sehr vereinfacht ausgedrückt ist es eine Plattform, auf der man u.a. Dateien speichern und teilen kann. Der oben genannte Link führt euch zu einer Datei auf GitHub, nämlich `PlantGrowthExcel.xlsx`. Diese liegt im Ordner `plant_growth` innerhalb des Repositories `ExampleData` des GitHub Accounts `SchmidtPaul`.
>
> Um die Datei herunterzuladen,
>
> - klickt auf das Download-Symbol rechts
> - **oder** klickt auf die drei Punkte `...` oben rechts und dann auf `Download`
> - **oder** drückt `Strg` + `Shift` + `s`
>
> 
> 

Speichert die Datei in einem Ordner eurer Wahl und gebt den Pfad zu dieser Datei in `pd.read_excel()` an. Passend zum obigen Video ist der Pfad **bei mir** wieder wie folgt:

```
pfad_zu_excel_datei = 'C:/Users/PythonKurs/Desktop/Python Kurs/02 JupyterLab/
PlantGrowthExcel.xlsx'
```

Wenn wir die die Datei also nun importieren, so erhalten wir folgendes Ergebnis:

```
df = pd.read_excel(pfad_zu_excel_datei)
df
```

```
                                          Hallo!
0  Die Daten befinden sich auf dem Tabellenblatt ...
```

Das sieht eigenartig aus. Der Grund ist, dass die Daten auf dem zweiten Tabellenblatt namens “Daten” liegen, standardmäßig aber das erste Tabellenblatt importiert wird.

Öffnet an dieser Stelle ruhig mal die Excel-Datei in Excel um euch die Tabellenblätter selbst anzuschauen. Schließt die Datei danach aber wieder - es kann vorkommen, dass Python die Datei nicht öffnen kann, wenn sie bereits in einem anderen Programm geöffnet ist.

Wir versuchen es also erneut und geben diesmal explizit mit dem Argument `sheet_name=` an, dass wir das Tabellenblatt mit dem Namen “Daten” importieren wollen und dann klappt es auch:

```
df = pd.read_excel(pfad_zu_excel_datei, sheet_name='Daten')
df
```

```
    rownames  weight group
0          1    4.17  ctrl
1          2    5.58  ctrl
2          3    5.18  ctrl
3          4    6.11  ctrl
4          5    4.50  ctrl
5          6    4.61  ctrl
6          7    5.17  ctrl
7          8    4.53  ctrl
8          9    5.33  ctrl
9         10    5.14  ctrl
10        11    4.81  trt1
11        12    4.17  trt1
12        13    4.41  trt1
13        14    3.59  trt1
14        15    5.87  trt1
15        16    3.83  trt1
16        17    6.03  trt1
17        18    4.89  trt1
18        19    4.32  trt1
19        20    4.69  trt1
20        21    6.31  trt2
21        22    5.12  trt2
22        23    5.54  trt2
23        24    5.50  trt2
24        25    5.37  trt2
25        26    5.29  trt2
26        27    4.92  trt2
27        28    6.15  trt2
28        29    5.80  trt2
29        30    5.26  trt2
```

# Relative Pfade

Wie schon erwähnt, ist es nicht der beste Ansatz, die lokalen Pfade so wie oben anzugeben. Solche Pfade heißen absolute Pfade, weil sie den genauen Pfad von der Wurzel des Dateisystems bis zur Datei angeben. Die Alternative sind relative Pfade. Ein relativer Pfad ist ein Pfad, der relativ zu einem anderen Pfad ist und wenn ihr in einem Jupyter Notebook arbeitet, dann sind die Pfade relativ zum Speicherort/Pfad des Notebooks. Ihr könnt also den Pfad zur Datei relativ zum Pfad des Notebooks angeben.

- Liegt eine Datei `datei.csv` also im gleichen Ordner wie das Notebook, dann könnt ihr einfach den Dateinamen angeben: `pd.read_csv('datei.csv')`.
- Liegt eine Datei `datei.csv` in einem Unterordner namens `Datenordner`, dann könnt ihr den Pfad zum Unterordner und den Dateinamen angeben: `pd.read_csv('Datenordner/datei.csv')`
- Liegt eine Datei `datei.csv` im übergeordneten Ordner, dann könnt ihr `../` verwenden, um einen Ordner nach oben zu gehen: `pd.read_csv('../datei.csv')`

Relative Pfade sind demnach portabler und flexibler als absolute Pfade. Ihr könnt euren Code also an jemand anderen weitergeben und er/sie kann ihn ohne Änderungen ausführen, solange die Dateien im gleichen Verzeichnis bzw. an den gleichen Positionen relativ zueinander liegen.

# Übungen

Man kann anstelle des Tabellenblatt-Namen (als String) auch die Nummer/den Index des Tabellenblatts (als Zahl) angeben. Welche Nummer müsste ich demnach angeben um ebenfalls das “Daten” Tabellenblatt aus der oben genannten Excel-Datei zu importieren? (Hinweis: Auch hier beginnt Pyhton bei 0 zu zählen.)

```
pd.read_excel(pfad_zu_excel_datei, sheet_name= _ )
```

Probiere aus was passiert, wenn du jeweils eins der folgenden Argumente zusätzlich in `pd.read_excel(pfad_zu_excel_datei, sheet_name='Daten')` verwendest:

- `names=['Var1', 'xxxx']`
- `usecols='B:C'`
- `skiprows=[0,1], header=None`
- (A) Geschafft