vl-07

January 9, 2026

1 Programmierung für KI

1.0.1 Wintersemester 2025/26

2 Visualisierung von Daten in Python

2.1 Vorab...

In dieser Veranstaltung geht es darum, wie man mathematische Abbildungen (*Plots*) in mit der Bibliothek *Matplotlib* in Python Python umsetzt.

Es geht *nicht* (oder nur am Rande) darum, wie man gute Abbildungen gestaltet.

- Claus O. Wilke, *Fundamentals of Data Visualization*, O'Reilly, 2019.
- Shirin Elsinghorst, *The Good, the Bad and the Ugly: Analysen effektiv visualisieren und kommunizieren*, data2day Workshop, 2020

2.2 Matplotlib

- *matplotlib* ist eine Python Bibliothek zum Plotten von 2D Grafiken
- OpenSource Projekt seit 2002
- Der Funktionsumfang der Bibliothek ist sehr groß
- Für die Darstellung spezieller Graphen gibt es viele Beispiele in der Matplotlib Galerie

2.2.1 Matplotlib importieren

Genau wie bei Numpy mit `np` importieren wir standardmäßig Matplotlib unter dem Namen `mpl`

```
import matplotlib as mpl
```

2.2.2 PyPlot

- Pyplot ist ein Matplotlib-Modul, das eine MATLAB-ähnliche Schnittstelle bietet
- Funktioniert wie eine Kommandozeile
- Eine Abbildung kann erstellt und sukzessive verändert/erweitert werden
- Pyplot ist die Schnittstelle, die wir für die allermeisten Matplotlib-Funktionen verwenden

```
import matplotlib.pyplot as plt
```

[ ]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

2.2.3 Styles

- Ein *Style* legt allgemeine Eingeschften der Plots fest: Farbschema, Schriftgröße, Hintergrundfarbe, etc.
- Der `default` Style passt für die meisten Anwendungsfälle
- `dark_background` kann nützlich sein, wenn Sie Abbildungen für eine Website oder einen Foliensatz mit dunklem Hintergrund generieren wollen

[ ]:
```
#plt.style.use('dark_background')
plt.style.use('default')
mpl.rcParams['figure.figsize'] = (8, 3)
```

2.2.4 Plots anzeigen

- Wenn Sie Matplotlib in Jupyter Notebook Zellen verwenden, wird die Abbildung nach jeder Zelle *ausgewertet* und angezeigt
- In einem Python Skript funktioniert das nicht automatisch
- Mit `plt.show()` werden alle aktiven Grafiken ausgewertet und angezeigt

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.show()
```

Im Notebook erfolgt die Auswertung automatisch

[ ]:
```
import numpy as np
plt.figure()
x = np.linspace(0, 3*np.pi, 100)
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```

2.2.5 Grafiken speichern

- Eine praktische Eigenschaft der Matplotlib ist die Fähigkeit, Grafiken in verschiedenen Bildformaten abzuspeichern
- Man kann eine Grafik sehr einfach über das `savefig()` Kommando speichern
- Über den angegeben Dateinamen findet Matplotlib das gewünschte Format

[ ]:
```
plt.savefig('Sinus_und_Cosinus.png')
```

Nach diesem Kommando sollte es eine entsprechende Datei im aktuellen Verzeichnis geben

[ ]:
```
!ls -lh Sinus_und_Cosinus.png
```

Schauen wir uns die erzeugte Bilddatei an:

[ ]:
```
from IPython.display import Image
Image('Sinus_und_Cosinus.png')
```

Warum ist die Abbildung leer?

Nach einer Code-Zelle ist die bisherige Figure nicht mehr aktiv.

[ ]:
```
import numpy as np
x = np.linspace(0, 3*np.pi, 100)
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
plt.savefig('Sinus_und_Cosinus.png')
```

[ ]:
```
from IPython.display import Image
Image('Sinus_und_Cosinus.png')
```

`savefig()` kennt noch viele weitere Dateiformate. Eine vollständige Liste erhalten die ebenfalls von Matplotlib:

[ ]:
```
plt.gcf().canvas.get_supported_filetypes()
```

2.2.6 Die 2 Schnittstellen von Matplotlib

- Wie wir gesehen haben, kann das Verhalten von Matplotlib verwirrend ein
- Ein Grund ist, dass es 2 Verwendungsmöglichkeiten für PyPlot gibt:

1. **Die MATLAB-artige Schnittstelle**
2. **Die objektorientierte Schnittstelle**

Die MATLAB-artige Schnittstelle

- Matplotlib wurde ursprünglich als Python-Alternative für (die Plotting-Funktionen von) MATLAB etwickelt
- Viele MATLAB Funktionen zum Visualisieren lassen sich 1:1 mit `plt` umsetzen
- Beachten Sie, dass es in folgendem Beispiel keine *Variablen* gibt:

[ ]:
```
plt.figure()  # Abbildung erzeugen

# Erzeuge das erste (von 2) Koordinatensystem(en)
plt.subplot(2, 1, 1) # (Zeilen, Spalten, aktuelle Achse)
plt.plot(x, np.sin(x))

# Erzeuge das zweite Koordinatensystem
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

- Wir sehen hier keine Variablen, allerdings gibt es einen *versteckten Zustand*
- PyPlot verwaltet intern ein **current figure**, also eine aktuelle Abbildung und aktuelle Koordinatensysteme
- Alle `plt.`-Kommandos werden auf diese Objekte angewendet

- Normalerweise *sehen* Sie diese Objekte nicht
- Man kann Referenz auf diese internen Objekte beziehen
- `plt.gcf()` (*get current figure*) liefert eine Referenz auf das *Figure* Objekt
- `plt.gca()` (*get current axes*) liefert eine Referenz auf ein *Axes* Objekt
- Manche Funktionen müssen *über* diese Objekte aufgerufen werden

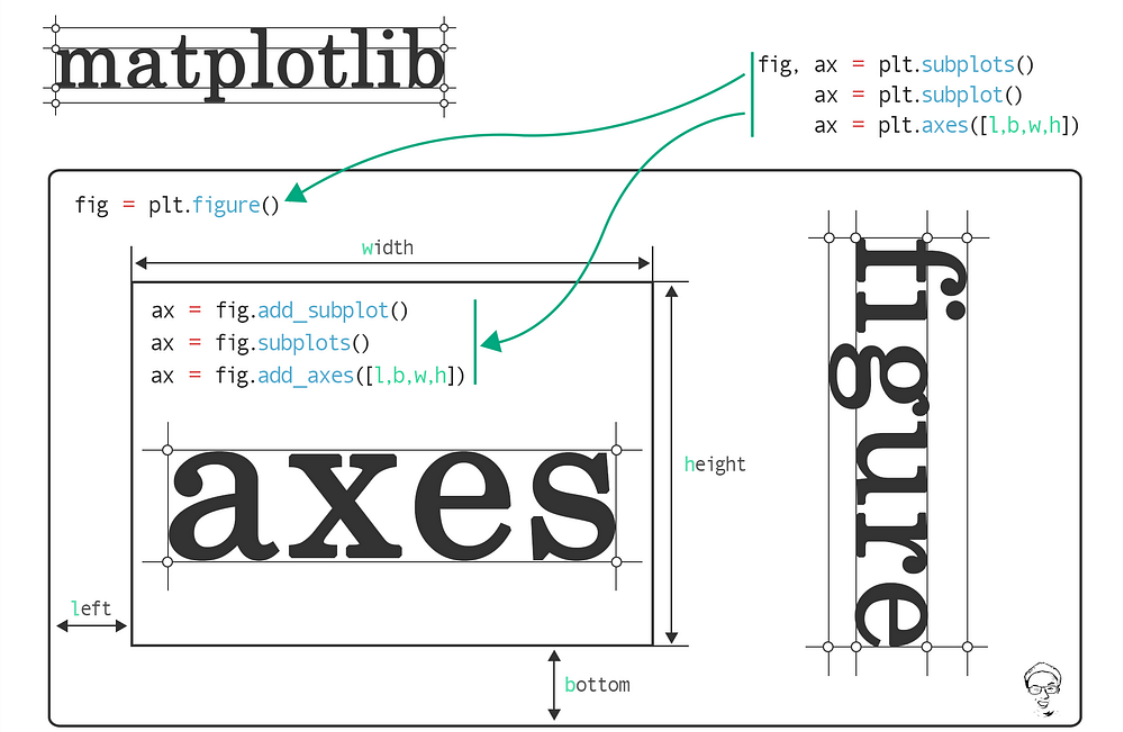
```
plt.gcf().canvas.get_supported_filetypes()
```

Die objektorientierte Schnittstelle

- Für komplexere Grafiken ist es sinnvoll, direkt die objektorientierte Schnittstelle zu verwenden
- `plt.subplots(N)` liefert ein 2-Tupel mit Referenzen auf die PyPlot Objekte
  1. Referenz auf die Abbildung (*Figure*)
  2. Eine Liste mit N Referenzen auf Koordinatensysteme (*Axes*)
- Normalerweise werden die Elemente des Tupels `fig` und `ax` genannt
- `fig, ax = plt.subplots(2)`
- Es zeugt von gutem Stil, sich an solche Konventionen zu halten

[ ]:
```
# Erstelle eine Abbildung mit Koordinatensystemen
fig, ax = plt.subplots(2)

# Plotte explizit in bestimmte Koordinatensysteme
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```



2.3 Kurvendiagramme

- Die vermutliche häufigste Diagramm-Art sind Kurvendiagramme
- Beispiel: Graph einer Funktion $y = f(x)$.

[ ]:
```
plt.style.use('seaborn-v0_8-whitegrid')
fig, ax = plt.subplots()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```

Wir können mehrere *Plots* in das selbe Koordinatensystem Zeichnen, indem wir die `plot`-Methode mehrfach aufrufen (bevor die Abbildung ausgewertet wird)

[ ]:
```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

2.3.1 Plots modifizieren: Linien

- Die `plot`-Methode besitzt diverse optionale Parameter
- Die Farbe der Kurve kann über den Parameter `color` eingestellt werden
- Auch hierfür gibt es verschiedene Möglichkeiten

[ ]:
```
plt.plot(x, x+1, color='blue')        # per Name
plt.plot(x, x+2, color='g')           # per Kurzname (rgbcmyk)
plt.plot(x, x+3, color='0.25')        # Graustufen zwischen 0 and 1
plt.plot(x, x+4, color='#FFDD44')     # Hex-Code (RRGGBB von 00 bis FF)
plt.plot(x, x+5, color=(1.0,0.2,0.3)) # RGB Tupel, Werte von 0 bis 1
plt.plot(x, x+6, color='chartreuse'); # HTML Farb-Namen
```

Wenn Sie keine `color` angeben, wechselt PyPlot durch das voreingestellte Farbschema.

[ ]:
```
plt.plot(x, x+1)
plt.plot(x, x+2)
plt.plot(x, x+3)
plt.plot(x, x+4)
plt.plot(x, x+5)
plt.plot(x, x+6)
```

Mit einem `plot`-Aufruf können ebenfalls mehrere Funktionen geplottet werden.

[ ]:
```
plt.plot(x, x+1, x, x+2, x, x+3, x, x+4, x, x+5, x, x+6);
```

Neben der Farbe kann auch die Linienform angepasst werden. Dies geschieht über den `linestyle` Parameter

[ ]:
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
#ODER:
```

```
plt.plot(x, x + 6, linestyle='-')   # solid
plt.plot(x, x + 7, linestyle='--')  # dashed
plt.plot(x, x + 8, linestyle='-.')  # dashdot
plt.plot(x, x + 9, linestyle=':');  # dotted
```

`linestyle` und `color` können in eine Art *Formatstring* zusammengefasst werden

[ ]:
```
plt.plot(x, x + 0, '-g')   # solid Grün
plt.plot(x, x + 1, '--c')  # dashed Cyan
plt.plot(x, x + 2, '-.k')  # dashdot Schwarz
plt.plot(x, x + 3, ':r');  # dotted Rot
```

2.3.2 Plots modifizieren: Grenzen der Koordinatenachsen

- Matplotlib leitet Grenzen für die Koordinatenachsen automatisch her
- Manchmal möchte man die Grenzen selbst einstellen
- Hierzu dienen die Methoden `plt.xlim()` und `plt.ylim()`

[ ]:
```
plt.figure()
plt.subplot(1,2,1)
plt.plot(x, np.sin(x))
plt.subplot(1,2,2)
plt.plot(x, np.sin(x))
plt.xlim(0, 10)
plt.ylim(1.5, -1.5);
```

Bei der objektorientierten Schnittstelle heißen die `xlim()` und `ylim()` Funktionen übrigens anders, nämlich `set_xlim()` und `set_ylim()`

(Habe ich schon erwähnt, dass die beiden Schnittstellen von PyPlot oft verwirren?)

[ ]:
```
fix, ax = plt.subplots(1,2)
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.sin(x))

ax[1].set_xlim(0, 10)
ax[1].set_ylim(-1, 1);
```

Mit `plt.axis()` können die `x` und `y` Grenzen gemeinsam gesetzt werden, als Liste von 4 Werten`[xmin, xmax, ymin, ymax]`

[ ]:
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```

2.3.3 Plots modifizieren: Titel von Graphen und Achsen

- Bei einer guten Abbildung sind immer die Achsen beschriftet
- Bei mehreren Kurven sollte es eine Legende geben
- Graphen können einen Titel besitzen

[ ]:
```
plt.plot(x, np.sin(x))
plt.title("Sinuskurve")
plt.xlabel("x", size=16)
plt.ylabel("sin(x)", size=16);
```

- Über den Parameter `label` der Methode `plot` kann einer Kurve ein Name verliehen werden
- Diese Namen werden in die Legende übernommen
- Eine Legende kann mit `plt.legend()` dem Graphen hinzugefügt werden
- Die Positionierung erfolgt automatisch oder über den Parameter `loc`

[ ]:
```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.legend(loc="lower left");
```

- Viele `.plt`-Funktionen der MATLAB-artigen Schnittstelle lassen sich genauso mit der objektorientierten Schnittstelle verwenden#
- Leider gilt das nicht für alle Funktionen, hier einige Unterschiede
- `plt.xlabel()` → `ax.set_xlabel()`
- `plt.ylabel()` → `ax.set_ylabel()`
- `plt.xlim()` → `ax.set_xlim()`
- `plt.ylim()` → `ax.set_ylim()`
- `plt.title()` → `ax.set_title()`

In der objektorientierten Variante können alle Parameter gleichzeitig über die `set()`-Funktion eingestellt werden:

[ ]:
```
fig, ax = plt.subplots()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2), xlabel='x', ylabel='sin(x)',
       title='Sinuskurve');
```

2.4 Streudiagramme

- Streudiagramme sind in der Statistik und zur Datenanalyse sehr verbreitet
- Simple Idee: Statt einer verbunden Kurve plotte nur die Punkte
- Da bei `plot` die Funktion als x und y Koordianten übergeben wird, kann `plot` auch Streudiagramme erzeugen

[ ]:
```
x = np.linspace(0, 10, 30)
y = np.sin(x)
#plt.plot(x, y, 'o', color='k');
plt.plot(x, np.sin(x), linestyle='', marker='o', color='k');
plt.plot(x, np.cos(x), 'o', color='r');
```

- Als *Marker* können verschiedene Symbole verwendet werden

[ ]:
```
import numpy.random as rng
plt.figure(figsize=(9,6))
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(3), rng.rand(3), marker, markersize=8,
             label=f"'{marker}'")
plt.legend(fontsize=14)
plt.xlim(0, 1.2);
```

"Echte" Streudiagramme mit `plt.scatter` Die bessere Alternative um Streudiagramme zu erzeugen ist die Funktion `plt.scatter`

[ ]:
```
x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.scatter(x, y, marker='o');
```

- `plt.scatter` hat mehr Einstellungsmöglichkeiten als `plt.plot`
- Jeder einzelne Punkt kann formatiert werden, bzw. die Daten geben die Formatierung vor
- So lassen sich drei (und mehr) Dimensionen in einem 2D-Plot darstellen

[ ]:
```
x = []
for i in range(4):
    x.append(rng.randn(100))
plt.scatter(x[0], x[1], c=x[2], s=1000*x[3], alpha=0.5, cmap='viridis')
plt.colorbar();  # Zeige die Farbskala
```

- Streudiagramme helfen bei der Datenanalyse, um zu verstehen, wie Datenpunkte anhand von *Merkmalen* angeordnet sind.
- Beispiel: Iris Datensatz mit Schwertlilienarten nach ihren Blütenblättern (*setal*) und Kronenblättern (*petal*) vermessen

[ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
merkmale = iris.data.T
art = iris.target
plt.scatter(merkmale[3], merkmale[1], alpha=0.6, marker='s', c=art,␣
 ↪cmap='viridis')
plt.xlabel(iris.feature_names[3])
plt.ylabel(iris.feature_names[1]);
```

2.5 Plots beschriften

[ ]:
```
x = np.linspace(-1.5, 1.5, 30)
px = 0.8
py = px**2

plt.plot(x, x**2, "b-", px, py, "ro")
plt.text(0, 1.5, "Quadratfunktion\n$y = x^2$", fontsize=20, color='blue',␣
 ↪horizontalalignment="center")
```

```
plt.text(px-0.05, py, "Ein Punkt", ha="right", weight="heavy")
plt.text(px, py, "x = %0.2f\n\ny = %0.2f"%(px, py), rotation=45, color='red')
```

2.6 3D-Plots

[ ]:
```
ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Greens');
```

[ ]:
```
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none');
```

2.7 GIFs erzeugen

[ ]:
```
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()
ax.axis([0, 4*np.pi, -1, 1]);
x = np.linspace(0,2*np.pi,50)

it = 63 # (2*PI)/0.1
p = 0

def my_plot(i):
    global p
    ax.clear()
    ax.plot(x, np.sin(x - p), color='r')
    p += 0.1
    return
```

```
anim = FuncAnimation(fig, my_plot, frames=np.arange(0, it), interval=50,␣
↪repeat=True)
anim.save('simple.gif', dpi=80, writer='pillow')
```

[ ]:
```
from IPython.display import Image
Image('simple.gif')
```

2.8 Beispiel: Covid-Daten visualisieren

- Beispiel für eine Visualisierung, bei der Matplotlib *nicht mehr ausreicht*
- Auch hier verwenden wir *Pandas* um die Daten zu verwalten
- Zusätzlich verwenden wir *Plotly* und *GeoJson*

[ ]:
```
import sys
!{sys.executable} -m pip install geojson
```

[ ]:
```
import numpy as np
import csv
import requests
import io
import pandas as pd
import plotly.express as px
import geojson
```

[ ]:
```
import requests
import os.path
import os

if not os.path.isfile("owid-covid-data.csv"):
    url = "https://covid.ourworldindata.org/data/owid-covid-data.csv"
    headers={'User-Agent': 'Mozilla/5.0'}
    r = requests.get(url, headers=headers)
    if r.status_code == 200:
        try:
            f = open("owid-covid-data.csv", 'wb')
            f.write(r.content)
        except:
            print("Irgendetwas ist schief gegangen!")
    else:
        print("Status code",  r.status_code)

sizeinbyte = os.path.getsize('owid-covid-data.csv')
print(f"Die Datei ist {sizeinbyte/10**6:.2f} MB groß")
```

[ ]:
```
import pandas as pd
df=pd.read_csv("owid-covid-data.csv")
```

```
print(f"Die Anzahl aller Einträge ist {df.size}")
```

Um sich einen ersten Eindruck von der Tabelle zu machen, kann man eine Reihe von Pandas-Methoden aufrufen: - `df.head(k)` zeigt die ersten `k` Einträge der Tabelle. Sie werden sehen, dass die Daten nach Ländern sortiert sind - `df.info()` zeigt Informationen zu den Spalten der Tabelle - `df.describe()` Gibt einige statistische Kennzahlen zu den Daten aus

[ ]:
```
df.head(10)
```

[ ]:
```
df.info()
```

Wenn Sie die Daten nach einer anderen Spalte sortieren wollen, geht das mit der `sort_by_values` Methode:

[ ]:
```
df_date = df.sort_values(by='date')
df_date
```

- Der Datensatz enthält nicht die in DE häufig verwendete Maß der *7-Tage Inzidenz*
- Wir können es allerdings aus den neuen Fällen pro Tag berechnen.
  - Um eine Normalisierung gemäß der Einwohnerzahlen zu erreichen, verwendenden wir die Spalte `new_cases_per_million`
  - Diese kann allerdings fehlende Werte enthalten, z.B. weil für einige Länder an bestimmten Tagen keine Daten vorlagen
  - Um diese fehlenden Werte zu *schätzen*, interpolieren wir. D.h. wir nehmen an, bei einer *Lücke* würden sie Werte linear fortlaufen. Also bei der folge `2, 3, NaN, 7, 8` würde das `NaN` durch `5` ersetzt.
- Weiteres Problem: Die Tabelle ist nach Daten sortiert, alle Länder stehen also vermischt in der Tabelle
- Wir verwenden die `groupby`-Methode um die Tabelle in Länder-Gruppen zu blocken

[ ]:
```
# Falsche Werte aussortieren
indexEntries = df_date[df_date['new_cases_per_million'] < 0 ].index
df_cleaned = df_date.drop(indexEntries)

def berechne_inzidenz(x):
    x.new_cases_per_million = x.new_cases_per_million.interpolate()
    x["Inzidenz"] = x.new_cases_per_million.rolling(7).sum()/10
    return x

df_cleaned = df_cleaned.groupby('iso_code').apply(berechne_inzidenz)
```

2.8.1 Inzidenzen Visualisieren

Für die Darstellung der weltweiten Inzidenz-Werte, eignet sich ein Plot als Weltkarte die wir z.B. mit der *Plotly* Methode `choropleth` erzeugen können.

[ ]:
```
def inzidenzen_als_karte(df):
    fig = px.choropleth(df, locations="iso_code",
                        color="Inzidenz",
```

```
                    #scope='europe',
                    range_color = [0,200],
                    hover_name="location",
                    animation_frame="date",
                    title = "Corvid: weltweite 7-Tages Inzidenz",
                    color_continuous_scale=px.colors.sequential.Jet)
    fig["layout"].pop("updatemenus")
    return fig
```

[ ]:
```
fig = inzidenzen_als_karte(df_cleaned)
fig.show()
```

Um bestimmte Zeilen eines DataFrames herauszufiltern, kann man bei der Auswahl der Spalten Bedingungen angeben. So können wir z.B. die Werte aus Deutschland aus der Tabelle herausfiltern:

[ ]:
```
df_de = df[df['iso_code']=='DEU']

print(f"Die Anzahl aller Einträge aus Deutschland ist {df_de.size}")
df_de.head()
```

Damit können wir die Inzidenz-Werte für Deutschland (`DEU`), Großbritannien (`GBR`) und USA (`USA`) in einen gemeinsamen Graphen plotten.

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

fig, axes = plt.subplots(1,1,figsize=(16,8))
for c in ['DEU', 'GBR', 'USA']:
    df = df_cleaned[df_cleaned['iso_code']==c]
    df = df.drop(df[df.date=="2021-10-31"].index)
    plt.plot(df.date.values, df.Inzidenz.values, label=c)
axes.xaxis.set_major_locator(MaxNLocator(10))
plt.xticks(rotation = 45, size=12)
plt.legend(prop={'size': 18})
plt.savefig("Inzidenzen.png")
```

2.9 Quellen

[1] Jake VanderPlas, *Python Data Science Handbook*, O'Reilly, 2016.

Die Covid-Daten stammen von *Our World in Data* und wurden dem Git-Repository https://github.com/owid/covid-19-data entnommen.

[2] Hasell, J., Mathieu, E., Beltekian, D. *et al.*. *A cross-country database of COVID-19 testing*, Sci Data 7, 345 (2020). https://doi.org/10.1038/s41597-020-00688-8