Plot exportieren

by Woche 11

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Nach all den Visualisierungen, die wir bisher erstellt haben, wird es Zeit, sich mit dem Exportieren von Plots zu beschäftigen. Oftmals reicht es nicht aus, die Plots nur in Jupyter Notebooks oder interaktiven Umgebungen zu betrachten. Stattdessen möchte man sie oft in Präsentationen, Berichten oder wissenschaftlichen Arbeiten verwenden. Dazu müssen die Plots in verschiedene Dateiformate exportiert werden.

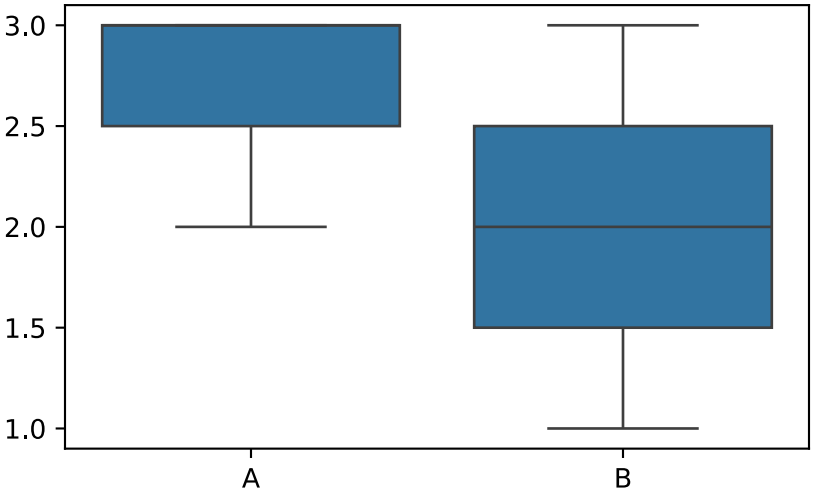
Größe des Plots

Bevor wir uns mit dem Exportieren von Plots beschäftigen, sollten wir uns damit auseinandersetzen, wie man die Größe der Plots festlegt. Dies kann mit der Funktion `plt.figure(figsize=(width, height))` erreicht werden, wobei `width` die Breite und `height` und Höhe der Abbildung in Inch/Zoll angeben. Bisher haben wir `plt.figure()` ohne Argumente verwendet, was die Standardgröße von 6.4x4.8 Zoll ergibt[1]. Hier zwei Beispiele mit unterschiedlichen Größen:

```
x = np.array(['A', 'A', 'A', 'B', 'B', 'B'])
y = np.array([2, 3, 3, 1, 2, 3])
```

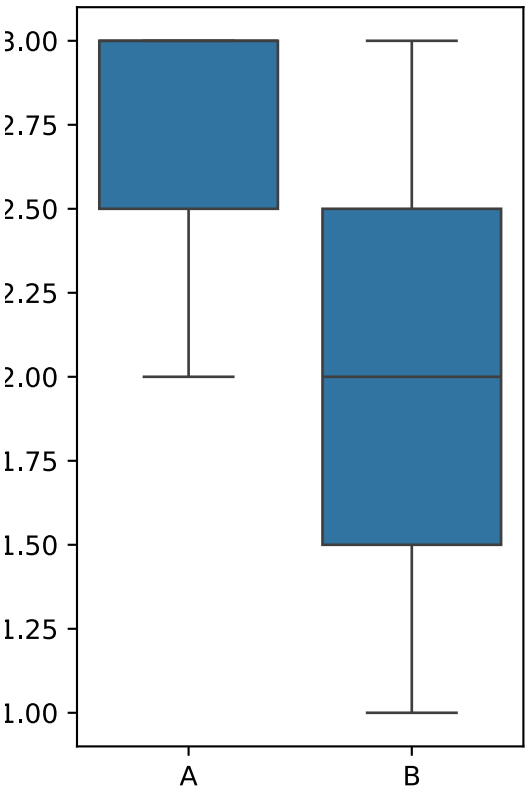
```
plt.figure(figsize=(5, 3))
sns.boxplot(x=x, y=y)
plt.show()
```

[1] Eine Übersicht aller Standardwerte gibt es hier oder auch hier.




```
plt.figure(figsize=(3, 5))
sns.boxplot(x=x, y=y)
plt.show()
```

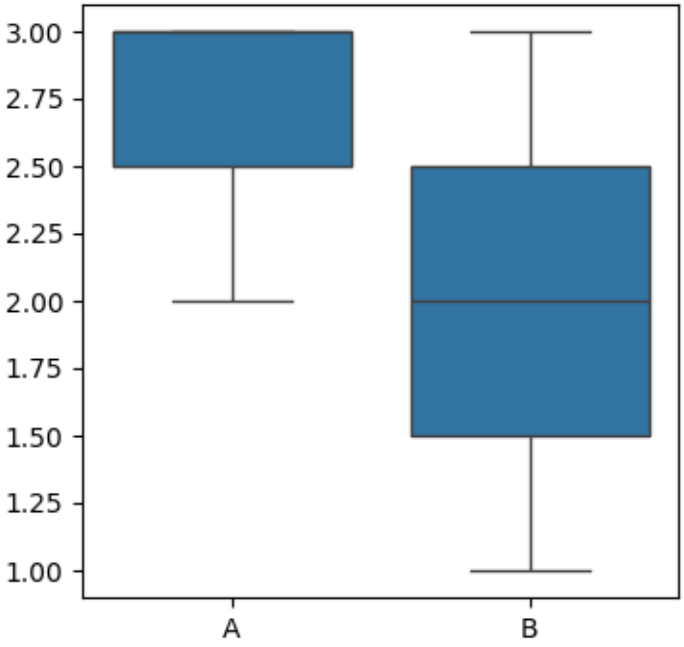
2



Exportieren

Um einen Plot als Datei zu exportieren, können wir die Funktion `plt.savefig()` anstelle von `plt.show()` verwenden. Diese Funktion erwartet mindestens den Dateinamen und das Dateiformat als Argumente. Beides kann einfach als String angegeben wie z.B. `'bild.png'` oder `'abbildung.jpeg'` angegeben werden. Die Datei wird dann im aktuellen Arbeitsverzeichnis gespeichert. Das bedeutet standardmäßig, dass die exportierte Datei im selben Ordner liegt wie das Jupyter Notebook, mit dessen Code sie erstellt wurde.

```
plt.figure(figsize=(4, 4))
sns.boxplot(x=x, y=y)
plt.savefig('bild1.png')
```



Wird die Datei also so in Jupyter Notebooks/Jupyter Lab exportiert, wird die Abbildung aber trotzdem auch direkt unter dem Befehl angezeigt. Ist dies nicht gewünscht, so kann `plt.close()` nach dem Speichern der Datei aufgerufen werden, sodass die Abbildung geschlossen wird.

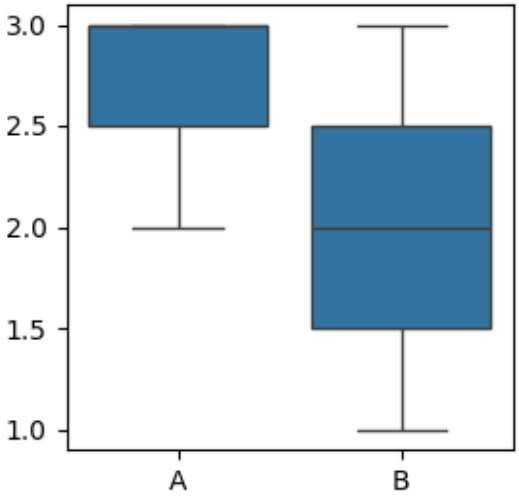
Andersherum ist es in Jupyter Notebooks/Jupyter Lab auch möglich eine bereits vorhandene bzw. früher exportierte Abbildung anzeigen zu lassen, indem wir die Funktionen `Image()` und `display()` aus dem Modul `IPython.display` wie folgt verwenden:

Exportiere Bild ohne es anzuzeigen

```
plt.figure(figsize=(3, 3))
sns.boxplot(x=x, y=y)
plt.savefig('bild2.png')
plt.close()
```

Zeige bereits vorhandenes Bild an

```
from IPython.display import Image, display
display(Image(filename='bild2.png'))
```



i Anzeigen via Markdown

Schließlich sei noch ergänzt, dass man eine Abbildung in einem Jupyter Notebook (`.ipynb`) auch direkt in Markdown-Zellen, also ohne Python-Code, anzeigen lassen kann. Dazu würde man lediglich `![](bild2.png)` in einer Markdown-Zelle schreiben und auch so wird dann das bereits vorhandene Bild angezeigt.

Dateiformate und DPI

Nun haben wir bereits zwei Abbildungen als PNG-Datei exportiert. Wir haben aber auch die Möglichkeit, die Abbildungen in anderen Formaten zu exportieren. Die gängigsten Formate sind PNG, JPEG, PDF und SVG. Jedes Format hat seine eigenen Vor- und Nachteile. PNG ist beispielsweise verlustfrei[2] und eignet sich gut für Plots, die in Präsentationen oder Berichten verwendet werden. PDF und SVG sind vektorbasierte[3] Formate, die sich gut für Drucke eignen, da sie skalierbar sind. Um alle verfügbaren Formate zu sehen, kann folgender Befehl verwendet werden:

```
formate = plt.gcf().canvas.get_supported_filetypes()
pd.DataFrame(formate.items())
```

[2] Verlustfrei bedeutet, dass die Qualität des Bildes nicht beeinträchtigt wird, wenn es mehrmals gespeichert wird, weil jedes Mal gilt, dass die komprimierte Datei wieder in die Originaldatei rücktransformiert werden kann.

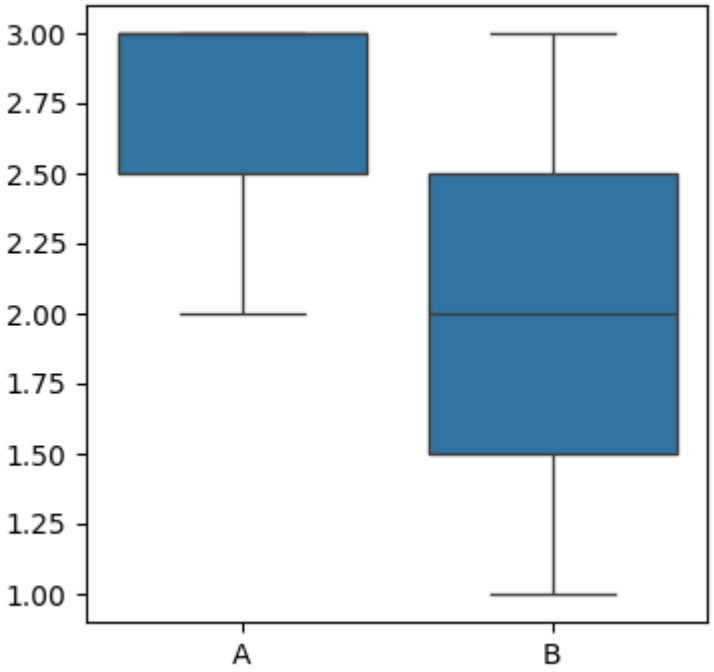
[3] Vektorbasierte Formate speichern sozusagen nicht als eine Menge Pixel, sondern als durch Formeln definierte Linien und Flächen. Das hat den Vorteil, dass man die Abbildung beliebig vergrößern kann, also unendlich hineinzoomen kann, ohne dass sie an Qualität verliert bzw. “pixelig” wird.

```
       0                                1
0    eps           Encapsulated Postscript
1    jpg  Joint Photographic Experts Group
2   jpeg  Joint Photographic Experts Group
3    pdf          Portable Document Format
4    pgf                PGF code for LaTeX
5    png         Portable Network Graphics
6     ps                        Postscript
7    raw                   Raw RGBA bitmap
8   rgba                   Raw RGBA bitmap
9    svg          Scalable Vector Graphics
10  svgz          Scalable Vector Graphics
11   tif          Tagged Image File Format
12  tiff          Tagged Image File Format
13  webp                 WebP Image Format
```

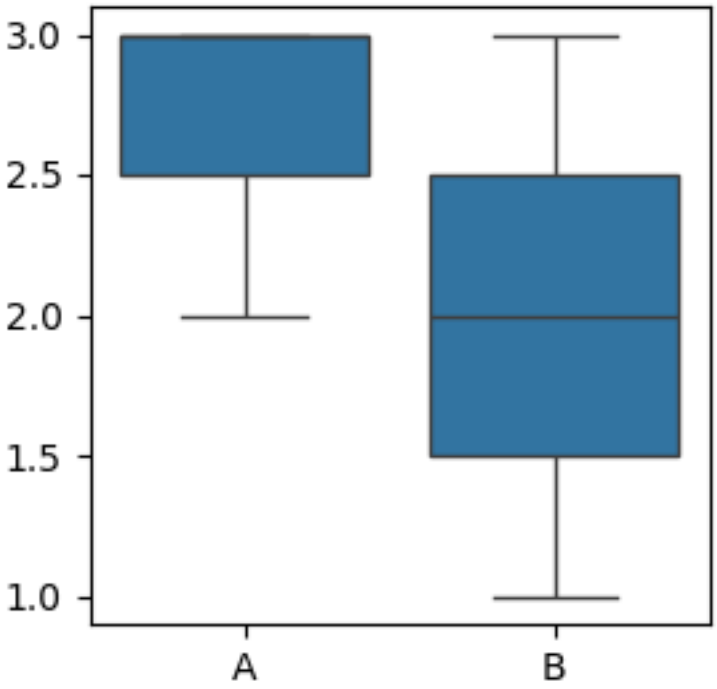
Um einen anderen Dateitypen zu erhalten, können wir also einfach den Dateinamen entsprechend ändern. Bei nicht-vektorbasierten Formaten (= PNG und JPEG) können wir auch die Auflösung des Bildes festlegen. Die Auflösung wird in DPI (Dots per Inch) gemessen und gibt an, wie viele Pixel pro Zoll in einem Bild vorhanden sind. Eine höhere Auflösung bedeutet, dass das Bild schärfer ist, aber auch mehr Speicherplatz benötigt. Die Standardauflösung beträgt 100 DPI. Die Auflösung kann mit dem Argument `dpi` in der Funktion `plt.savefig()` festgelegt werden. Dabei muss klar sein, dass selbst identische `figsize` bei unterschiedlichen `dpi`-Werten unterschiedlich große Dateien ergeben. Normalerweise wird dann ein weniger gut aufgelöstes Bild auch kleiner

dargestellt - auch in Jupyter Notebooks/Jupyer Labs. Hier werden beide Bilder aber trotz unterschiedlicher DPI ausnahmesweise mal auf dieselbe Höhe und Breite forciert um den Unterschied deutlich zu machen:

```
plt.figure(figsize=(4, 4))
sns.boxplot(x=x, y=y)
plt.savefig('bild1.png', dpi=100)
```




```
plt.figure(figsize=(4, 4))
sns.boxplot(x=x, y=y)
plt.savefig('out/bild3.png', dpi=30)
```



> **i** DPI bereits in `plt.figure()` angeben
>
> Übrigens kann der DPI-Wert auch bereits in der `plt.figure()`-Funktion - ebenfalls via `dpi=` - angegeben werden. Tatsächlich ist auch dort der besagte Standard `dpi=100` während der Standard in `plt.savefig()` gleich `dpi='figure'` ist. Letzteres bedeutet, dass der DPI-Wert standardmäßig dem Wert entspricht, der in der `plt.figure()`-Funktion angegeben wurde.

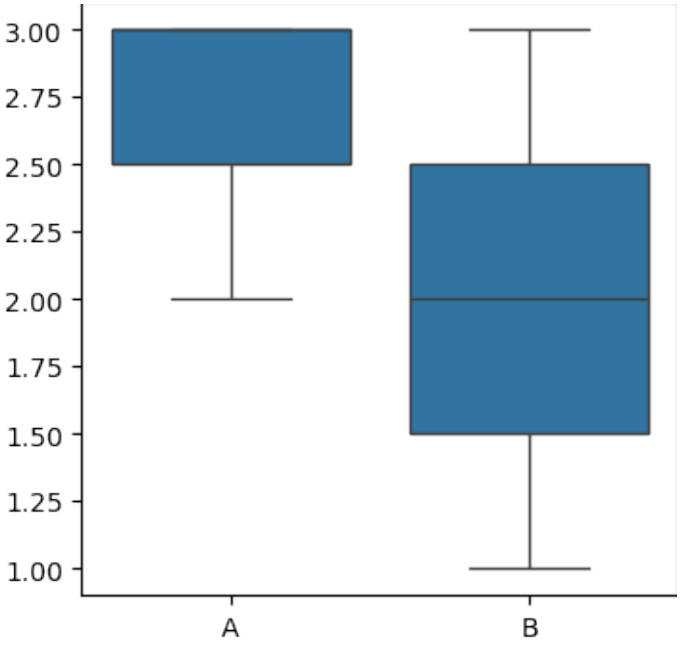
weitere Optionen

Neben dem Dateiformat und der Auflösung gibt es noch weitere Optionen, die in der Funktion `plt.savefig()` festgelegt werden können. Beispielsweise ist der Hintergrund des Plots standardmäßig weiß, erkann jedoch mittels `transparent=True` auch transparent gemacht werden.

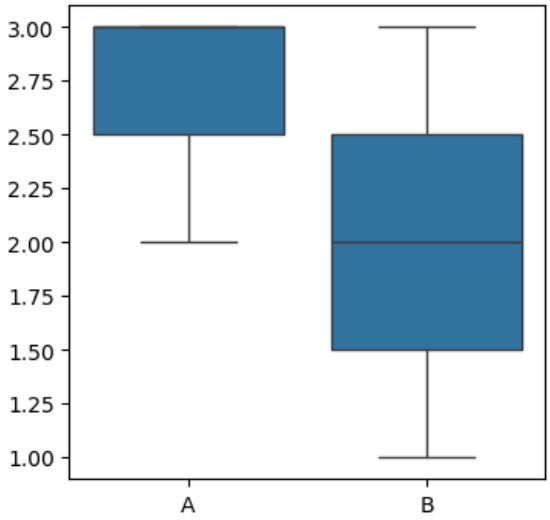
Außerdem können die Funktionen `bbox_inches='tight'` und `pad_inches=0.1` verwendet werden, um den Plot so zu beschneiden, dass er genau in die Abbildung passt. `bbox_inches='tight'` sorgt dafür, dass der Plot so zugeschnitten wird, dass alle Elemente im Plot sichtbar sind. `pad_inches` legt fest, wie viel Platz zwischen den Elementen und dem Rand des Plots bleiben soll. Genau dieser Befehl bietet eine

schnelle Lösung, falls mal Text am Rand einer Abbildung abgeschnitten erscheint. Hier ein Beispiel, wobei ein Hintergrundbild (Quelle: unsplash.com) verwendet wird um die Transparenz kenntlich zu machen:

```
plt.figure(figsize=(4, 4))
sns.boxplot(x=x, y=y)
plt.savefig(
  'out/bild4.png',
  bbox_inches='tight',
  pad_inches=0,
  transparent=True
)
plt.close()
```




```
plt.figure(figsize=(4, 4))
sns.boxplot(x=x, y=y)
plt.savefig(
  'out/bild5.png',
  bbox_inches='tight',
  pad_inches=2,
  transparent=False
)
plt.close()
```



Datei direkt öffnen

Hier sei noch ein Tipp erwähnt, mit dem man die Exportierten Dateien auch direkt mit seinem Betriebssystem, also quasi außerhalb von Python/Jupyter Notebooks/Jupyter Labs, öffnen kann. Dazu je nach Betriebssystem folgender Befehl verwendet werden:

Windows

```
import os

os.startfile('bild5.png')
```

macOS

```
import os
import subprocess as sp
sp.run(['open', 'bild5.png'])
```

Linux

```
import os
import subprocess as sp
sp.run(['xdg-open', 'bild5.png'])
```

Es sei angemerkt, dass die Datei dann also im Prinzip so geöffnet wird, wie als wenn man auf sie doppelklicken würde. Das bedeutet, dass sie in dem Standardprogramm geöffnet wird, das auf eurem Betriebssystem für das jeweilige Dateiformat zuständig ist. Es ist gut möglich, dass einige von euch für bestimmte Dateiformate noch gar kein Standardprogramm festgelegt haben. Beispielsweise würde dann in Windows beim Doppelklick auf eine solche Datei ein Dialog erscheinen, der fragt, mit welchem Programm die Datei geöffnet werden soll. Falls das der Fall ist, dann wird auch der oben genannte Befehl nicht funktionieren. Man muss also erst dafür sorgen, dass ein Standardprogramm festgelegt ist.

Wenn es aber erstmal funktioniert, dann ist dies eine sehr schnelle Möglichkeit, um die exportierten Dateien zu überprüfen, ohne sie erst in einem Dateiexplorer suchen und anklicken zu müssen. Ich selbst arbeite oft mit dieser Methode, wenn es eine relevante Abbildung ist, die bereits größtenteils fertig ist und ich nur noch die Feinheiten überprüfen möchte. Ich betrachte dann die Änderungen, die ich an der Abbildung vornehme, also gar nicht mehr in Python/Jupyter Notebooks/Jupyter Labs, sondern exportiere jedes Mal direkt die Abbildung und öffne sie auotmatisch mit einem entsprechenden Programm (auf meinem zweiten Bildschirm).

Weitere Ressourcen

- Beheben von Matplotlib savefig, das Labels abschneidet: Eine umfassende Anleitung [bis zur Überschrift *Alternative zu Matplotlib: Daten mit PyGWalker visualisieren*]

Übungen

Exportiere eine Abbildung deiner Wahl einmal in jedes der vier Dateiformate PNG, JPEG, PDF und SVG. Verwende dabei die Auflösung von 300 DPI für die Dateiformate, bei denen dies sinnvoll ist.

- (A) Geschafft