Heatmaps und mehr matplotlib

by Woche 3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
np.random.seed(1) # damit die Zufallszahlen reproduzierbar sind
```

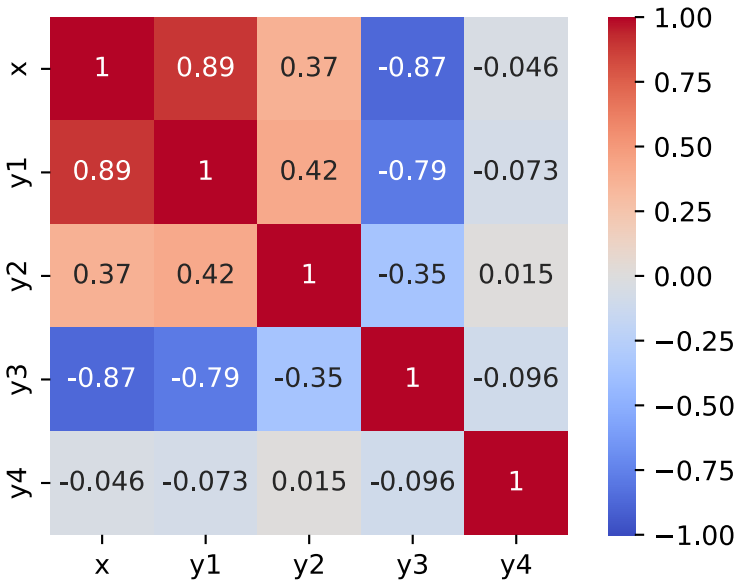
Im vorangegangenen Kapitel hatten wir ein schönes Beispiel dafür, wie man mit seaborn einfacher und schneller ans Ziel kommt und gleichzeitig einen gut aussehenden Plot erhält. Diesen wollen wir nun auch ohne seaborn nachbauen um ein etwas besseres Verständnis von matplotlib zu bekommen.

seaborn Heatmap ohne seaborn

Da wir aber mittlerweise wissen, dass seaborn nur ein *wrapper* für matplotlib ist, ist es eine gute Übung zu versuchen diesen Plot nur mit matplotlib nachzubauen. Am Ende können wir vergleichen wie viel umständlicher es ist dieselbe Heatmap nur mit matplotlib zu erstellen.

```
# Erzeugen korrelierter Daten
n = 100
x  =      np.random.normal(0, 1, n)   # 100 Zufallszahlen aus Normalverteilung
y1 =  x + np.random.normal(0, 0.5, n) # Starke positive Korrelation
y2 =  x + np.random.normal(0, 2, n)   # Schwache positive Korrelation
y3 = -x + np.random.normal(0, 0.5, n) # Starke negative Korrelation
y4 =      np.random.normal(0, 1, n)   # Keine Korrelation

# In DataFrame speichern
df = pd.DataFrame({'x': x,'y1': y1,'y2': y2,'y3': y3,'y4': y4})

# Korrelationsmatrix berechnen
correlation_matrix = df.corr()
```

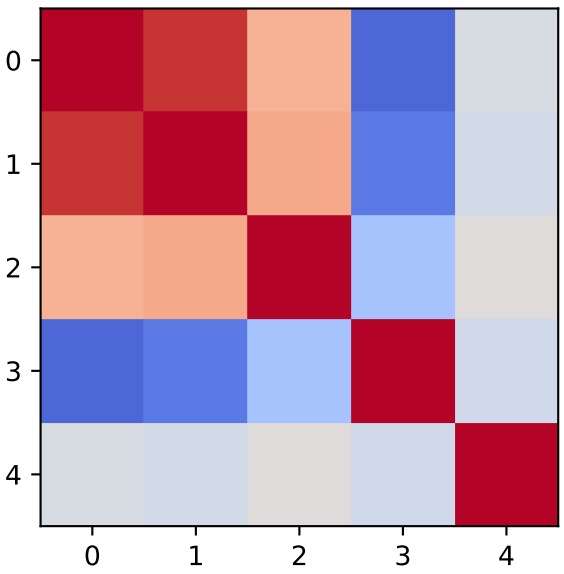
```
# Heatmap mit seaborn
fig, ax = plt.subplots()

sns.heatmap(
    data=correlation_matrix,
    square=True,     # forciert eine quadratische Form der Tiles
    annot=True,      # Schreibt Wert auf jedes Tile
    cmap='coolwarm', # Name einer der Farbpaletten
    vmin=-1,         # Legende Start - Standard: Kleinster Wert in Daten
    vmax=1,          # Legende Ende  - Standard: Größter Wert in Daten
    center=0,        # Legende Mitte
    ax=ax            # Explizit angeben, dass wir unsere ax verwenden wollen
)

plt.show()
```



Heatmap via ax.imshow()

Die matplotlib-eigene Heatmap-Funktion ist `imshow()` und von den Argumente, die wir in `sns.heatmap()` genutzt haben, können wir `cmap`, `vmin` und `vmax` auch hier verwenden. Außerdem gibt es zwar kein `square=True` um die Zellen quadratisch zu machen, aber wir können `aspect='equal'` verwenden um das gleiche Ergebnis zu erzielen. Das Resultat sieht schon ganz gut aus:

```
fig, ax = plt.subplots()
ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')
plt.show()
```



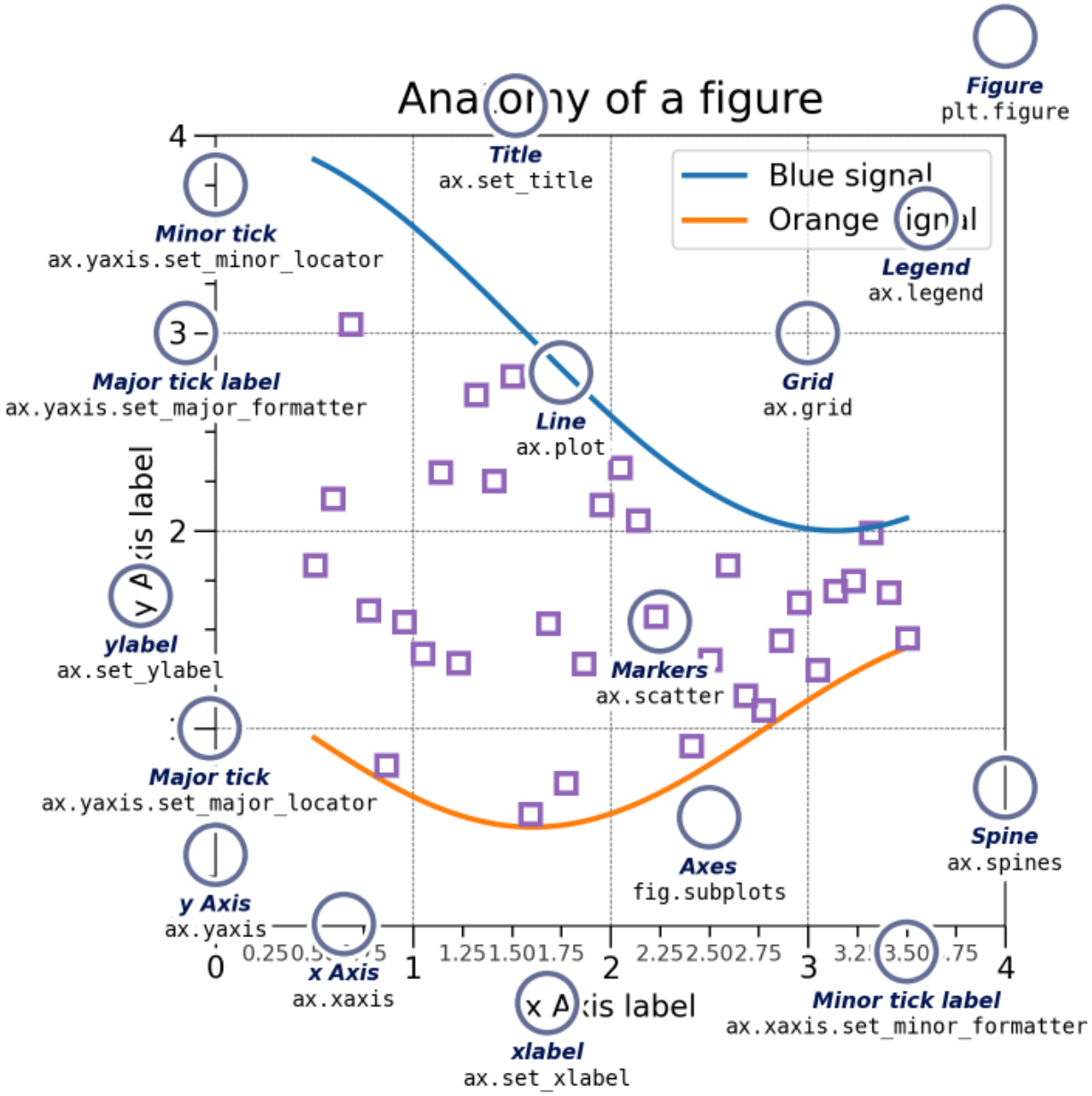
Wir haben also eine Heatmap mit der korrekte Farbpalette und den korrekten Werten. Was uns aber fehlt sind

- Kein schwarzer Rand um die Heatmap
- Achsenbeschriftungen
- Legende

- Labels für die einzelnen Zellen

Schwarzer Rand

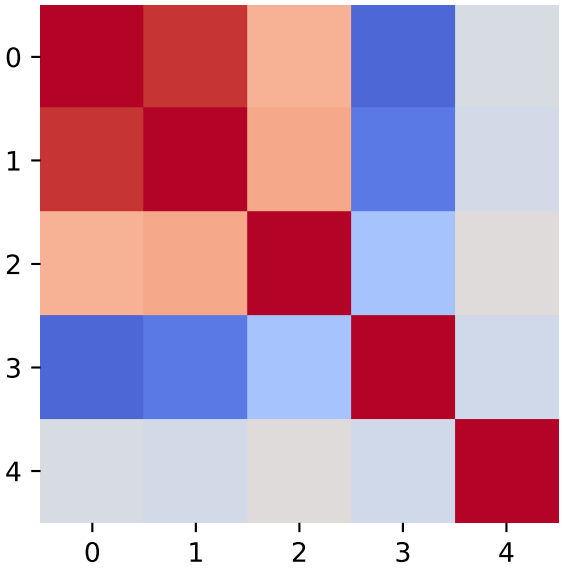
Den Punkt mit dem schwarzen Rand können wir am schnellsten und deshalb zuerst beheben. Beim schwarzen Rand handelt es sich laut matplotlib um die *Spines* der Achsen, die standardmäßig sichtbar sind. Solche Bezeichnungen muss man natürlich erst kennenlernen oder zumindest wissen, dass es sie gibt, um sie dann in der Dokumentation nachzuschlagen. Diese Abbildung stammt direkt aus der matplotlib-Dokumentation und ist schon mal ein guter Anfang:



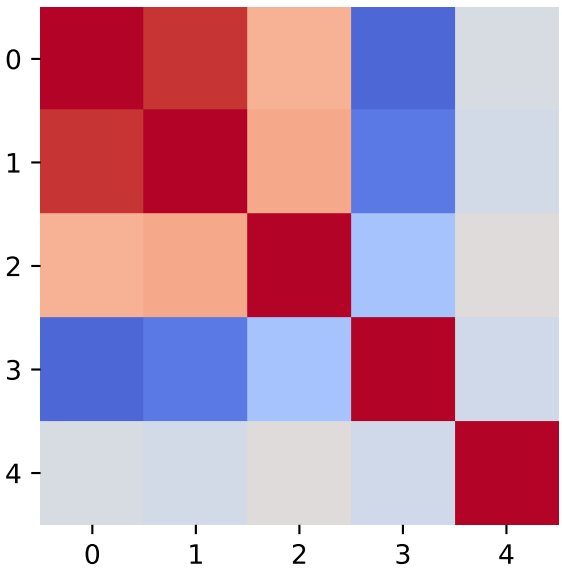
Um eine der Spines unsichtbar zu machen, können wir die `set_visible()` Methode verwenden. Hier sind zwei Möglichkeiten das für alle vier Spines umzuetzen. Da der Ansatz mit der Schleife eleganter ist, werden wir den auch weiter verwenden.

```
fig, ax = plt.subplots()
ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
```

```
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)

plt.show()
```



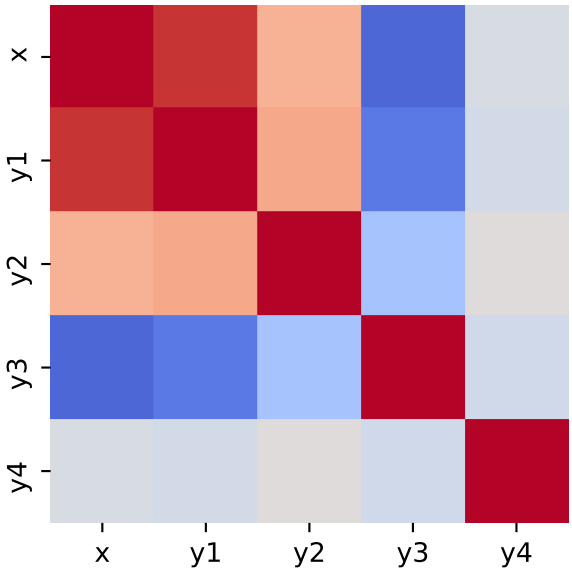

```
fig, ax = plt.subplots()
ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)



plt.show()
```



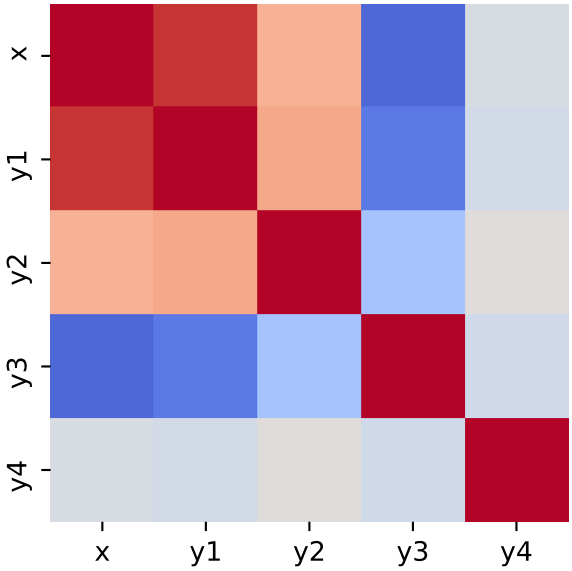
Achsenbeschriftungen

Wie man der *Anatomy of a Figure* entnehmen kann, unterscheidet man bei der Achsenbeschriftung in matplotlib zwischen dem “label” und dem “tick label”. Letzteres sollten wir in unserer Heatmap noch anpassen, da dort aktuell nur die Zahlen von 0 bis 4 stehen. Erstmal sollen die Tick Labels der y-Achse um 90° gedreht werden, was via `ax.tick_params(axis='y', rotation=90)` erreicht werden kann. Um die gewünschten Ticks so zu labeln, wie wir es wollen, müssen wir erst die Ticks setzen (`set_xticks()` / `set_yticks()`) und dann die Labels setzen (`set_xticklabels()` / `set_yticklabels()`. Hier wieder zwei Möglichkeiten, wobei die zweite Variante wieder eleganter ist.

```
fig, ax = plt.subplots()
ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks([0, 1, 2, 3, 4])
ax.set_yticks([0, 1, 2, 3, 4])
ax.set_xticklabels(['x', 'y1', 'y2', 'y3', 'y4'])
ax.set_yticklabels(['x', 'y1', 'y2', 'y3', 'y4'])
ax.tick_params(axis='y', rotation=90)

plt.show()
```




```
fig, ax = plt.subplots()
ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

plt.show()
```

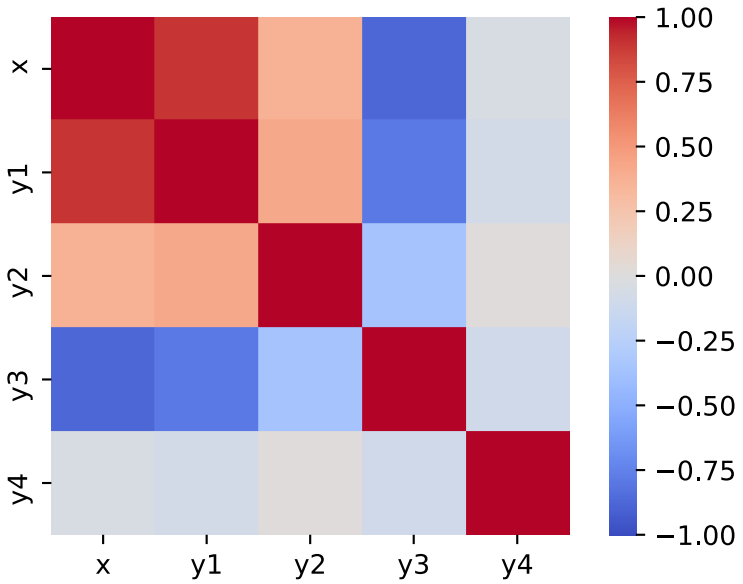


Das setzen der Ticks im ersten Schritt mag in genau diesem Beispiel unnötig vorkommen, da ja schon standardmäßig 0-4 an den Ticks steht, es muss aber klar sein, dass man ja nicht immer genau an all den Stellen Ticks haben möchte, an denen sie standardmäßig eingefügt wurden.

Farbgradient-Legende

Es gibt in matplotlib zwar den Befehl `ax.legend()`, welcher beispielsweise automatisch eine Legende erstellt für die verschiedenfarbigen Linien in einem Plot pro Kategorie. Wir haben hier aber keine Kategorien mit zugehörigen Farben, sondern eine Farbskala für die Werte in der Heatmap. Aus diesem Grund gehen wir hier einen anderen Weg und nutzen die `colorbar()` Funktion, welche eine Farbskala erstellt, die wir dann zu unserem Plot hinzufügen können. Damit das funktioniert, müssen wir allerdings die `imshow()` Funktion etwas anders aufrufen: Wir speichern das Ergebnis in einer Variable `im`, die wir dann an die `colorbar()` Funktion übergeben.

```
fig, ax = plt.subplots()
im = ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

cbar = fig.colorbar(mappable=im, ax=ax) # Farbskala hinzufügen
cbar.outline.set_visible(False) # Schwarzen Rand um Farbskala entfernen

plt.show()
```



Moment mal - was ist denn dieses `im`? Wir dachten, wir hätten mit `fig` und `ax` bereits alles verstanden, und jetzt kommt plötzlich noch ein weiteres Objekt hinzu? Ja, aber es handelt sich hierbei immerhin nur um einen weiteren logischen Schritt in der Hierarchie von matplotlib - **ein grafisches Element** innerhalb einer `ax`:

- `fig` repräsentiert die gesamte Abbildung (Figure), also das Gesamtcontainer-Objekt für alle Visualisierungselemente
- `ax` repräsentiert einen Plotbereich (Axes) mit eigenem Koordinatensystem innerhalb der Figure
- `im` repräsentiert ein einzelnes grafisches Element (hier ein *AxesImage*-Objekt), das durch eine Plotting-Methode (hier *ax.imshow()*) erzeugt wurde und spezifische Eigenschaften wie Farbzuordnung und Datenwerte enthält.

Andere Plotting-Funktionen erzeugen ebenfalls solche grafischen Elemente, die wir dann für weitere Anpassungen oder Ergänzungen verwenden können. Beispielsweise erzeugt `ax.scatter()` ein *PathCollection*-Objekt, das wir für die Anpassung von Größe und Farbe der Punkte verwenden können und `ax.plot()` erzeugt ein *Line2D*-Objekt, das wir für die Anpassung von Linienstärke und -farbe verwenden können. Wir haben diese aber bisher noch nie in ein Objekt gespeichert, sondern immer direkt geplottet. Nun brauchen wir aber genau dieses Objekt, um damit die Farbskala hinzuzufügen und wir nennen es `im`, weil es eben ein Axes**Im**age-Objekt ist.

Beim Erzeugen der Farbskale übergeben wir dann dieses `im`-Objekt an die `colorbar()` Funktion, sodass direkt entnommen werden kann um welche Farben und Werte es sich handelt. Außerdem legen wir mit `ax=ax` fest, dass die Farbskala auf der gleichen Achse wie unser Plot erscheinen soll[1].

Schließlich - weil wir ja genau sein wollen - müssen wir auch noch den schwarzen Rand um die Farbskala entfernen, der standardmäßig hinzugefügt wird. Das machen wir mit `cbar.outline.set_visible(False)` - also mit derselben Funktion wie oben bei den Spines. Und das geht wieder mal, da wir vorher das `cbar`-Objekt speichern, sodass wir dann für weitere Anpassungen verwenden können.

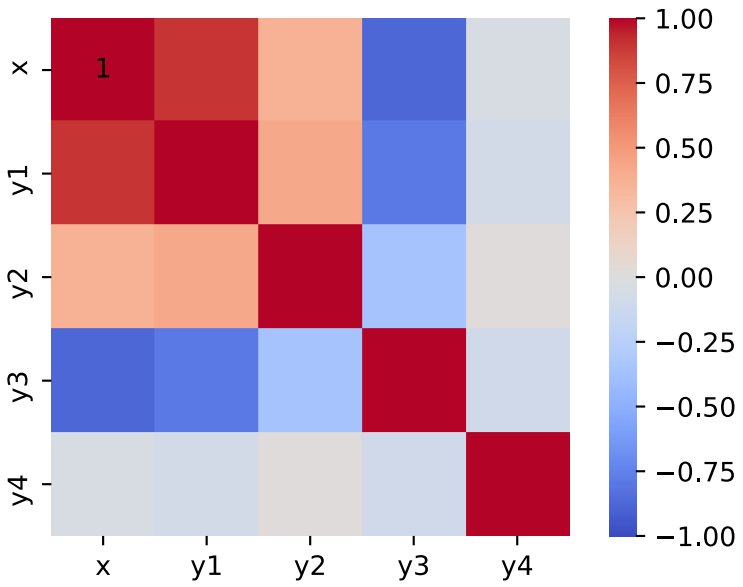
Labels für die Zellen

Die letzte Anpassung, die wir vornehmen wollen, ist das Hinzufügen der Werte in die Zellen. Wie man Text in eine Abbildung einfügt, haben wir bereits in einem früheren Kapitel gesehen. Hier

[1] In der Dokumentation von `colorbar()` wird es übrigens ausgedrückt als *von welcher Achse Platz für die Farbskala gestohlen werden soll.*

müssen wir also nur die Positionen der Texte in den Zellen bestimmen und dann den Text hinzufügen. Für einen Tile geht das auch relativ einfach, da wir nur die Mitte des Tiles bestimmen müssen. Beispielsweise muss in den Tile ganz oben links sowieso eine 1. Die genaue Position zu bestimmen ist sehr einfach, wenn wir das Koordinatensystem von `ax` (statt dem von `fig`) verwenden: x=0 und y=0. Allerdings ist der Text ja standardmäßig dann an diesem Punkt linkszentriert. Wir stellen also mit `ha="center"` (horizotnal) und `va="center"` (vertikal) sicher, dass die 1 genau mittig auf diese Koordinate geschrieben wird:

```
fig, ax = plt.subplots()
im = ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

cbar = fig.colorbar(mappable=im, ax=ax)
cbar.outline.set_visible(False)

ax.text(0, 0, "1", ha="center", va="center", color="black")

plt.show()
```

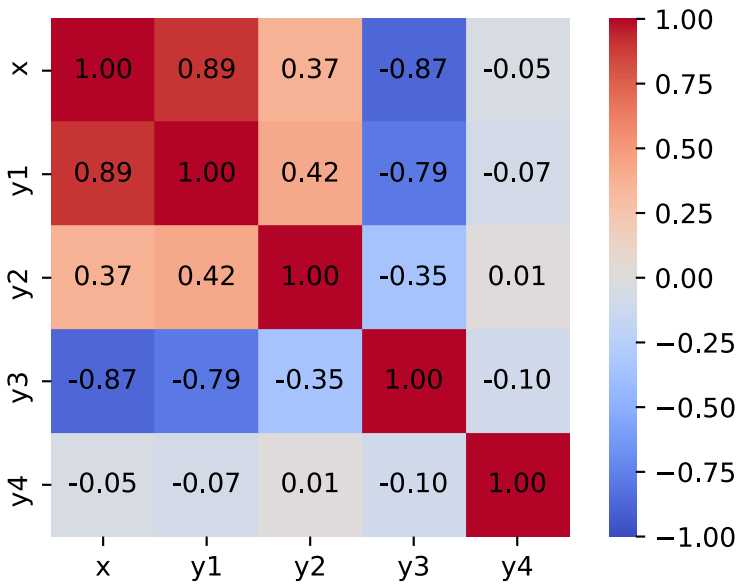


Das klappt bestens. Nun müssen wir das nur noch für alle Tiles machen und den Wert schreiben wir natürlich nicht manuell ab, sondern verwenden die Werte aus der `correlation_matrix`. Hier wieder zwei Ansätze wobei sich spätestens hier der Ansatz mit der eleganteren Schleife deutlich lohnt:

```
fig, ax = plt.subplots()
im = ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
```

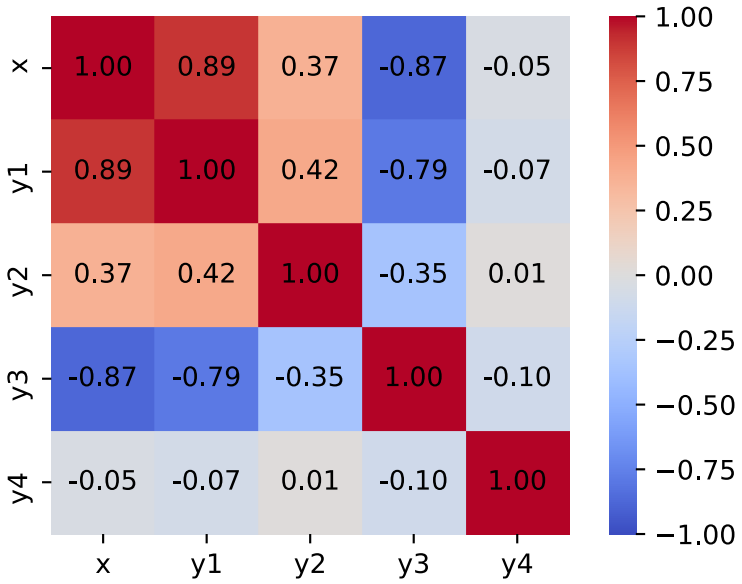
```
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

cbar = fig.colorbar(mappable=im, ax=ax)
cbar.outline.set_visible(False)

# Erste Zeile
ax.text(0, 0, f"{correlation_matrix.iloc[0, 0]:.2f}", ha="center", va="center",
color="black")
ax.text(1, 0, f"{correlation_matrix.iloc[0, 1]:.2f}", ha="center", va="center",
color="black")
ax.text(2, 0, f"{correlation_matrix.iloc[0, 2]:.2f}", ha="center", va="center",
color="black")
ax.text(3, 0, f"{correlation_matrix.iloc[0, 3]:.2f}", ha="center", va="center",
color="black")
ax.text(4, 0, f"{correlation_matrix.iloc[0, 4]:.2f}", ha="center", va="center",
color="black")

# Zweite Zeile
ax.text(0, 1, f"{correlation_matrix.iloc[1, 0]:.2f}", ha="center", va="center",
color="black")
ax.text(1, 1, f"{correlation_matrix.iloc[1, 1]:.2f}", ha="center", va="center",
color="black")
ax.text(2, 1, f"{correlation_matrix.iloc[1, 2]:.2f}", ha="center", va="center",
color="black")
ax.text(3, 1, f"{correlation_matrix.iloc[1, 3]:.2f}", ha="center", va="center",
color="black")
ax.text(4, 1, f"{correlation_matrix.iloc[1, 4]:.2f}", ha="center", va="center",
color="black")

# Dritte Zeile
ax.text(0, 2, f"{correlation_matrix.iloc[2, 0]:.2f}", ha="center", va="center",
color="black")
ax.text(1, 2, f"{correlation_matrix.iloc[2, 1]:.2f}", ha="center", va="center",
color="black")
ax.text(2, 2, f"{correlation_matrix.iloc[2, 2]:.2f}", ha="center", va="center",
color="black")
ax.text(3, 2, f"{correlation_matrix.iloc[2, 3]:.2f}", ha="center", va="center",
color="black")
ax.text(4, 2, f"{correlation_matrix.iloc[2, 4]:.2f}", ha="center", va="center",
color="black")

# Vierte Zeile
ax.text(0, 3, f"{correlation_matrix.iloc[3, 0]:.2f}", ha="center", va="center",
color="black")
ax.text(1, 3, f"{correlation_matrix.iloc[3, 1]:.2f}", ha="center", va="center",
color="black")
ax.text(2, 3, f"{correlation_matrix.iloc[3, 2]:.2f}", ha="center", va="center",
color="black")
ax.text(3, 3, f"{correlation_matrix.iloc[3, 3]:.2f}", ha="center", va="center",
color="black")
ax.text(4, 3, f"{correlation_matrix.iloc[3, 4]:.2f}", ha="center", va="center",
color="black")

# Fünfte Zeile
```

```
ax.text(0, 4, f"{correlation_matrix.iloc[4, 0]:.2f}", ha="center", va="center",
color="black")
ax.text(1, 4, f"{correlation_matrix.iloc[4, 1]:.2f}", ha="center", va="center",
color="black")
ax.text(2, 4, f"{correlation_matrix.iloc[4, 2]:.2f}", ha="center", va="center",
color="black")
ax.text(3, 4, f"{correlation_matrix.iloc[4, 3]:.2f}", ha="center", va="center",
color="black")
ax.text(4, 4, f"{correlation_matrix.iloc[4, 4]:.2f}", ha="center", va="center",
color="black")

plt.show()
```




```
fig, ax = plt.subplots()
im = ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

cbar = fig.colorbar(mappable=im, ax=ax)
cbar.outline.set_visible(False)

for i in range(len(correlation_matrix)):
    for j in range(len(correlation_matrix)):
        text = ax.text(j, i, f"{correlation_matrix.iloc[i, j]:.2f}",
                       ha="center", va="center", color="black")
```

```
plt.show()
```



Fazit

Und so haben wir es fast geschafft den seaborn-Plot nur mit matplotlib nachzubauen. Auf dem Weg hierhin haben wir nebenbei folgendes über matplotlib gelernt:

- Es gibt spezifische Begriffe für die verschiedenen Teile einer Abbildung - die *Anatomy of a Figure*
- Wir können Dinge wie die *Spines* der Achsen aber auch die *cbar.outline* unsichtbar machen, um den schwarzen Rand zu entfernen
- Wir können die Ticks und Labels der Achsen anpassen

- Wir können einen Farbgradienten als Legende hinzufügen
- Selbst Befehle wie `ax.imshow()`/`ax.scater()`/`ax.plot()` geben uns ein Objekt zurück, das wir für weitere Anpassungen verwenden können, wenn wir es denn speichern
- Wir haben 3 verschiedene Beispiele gesehen warum man immer wieder Schleifen innerhalb von matplotlib-code sieht - es ist einfach eleganter, kürzer und weniger fehleranfällig

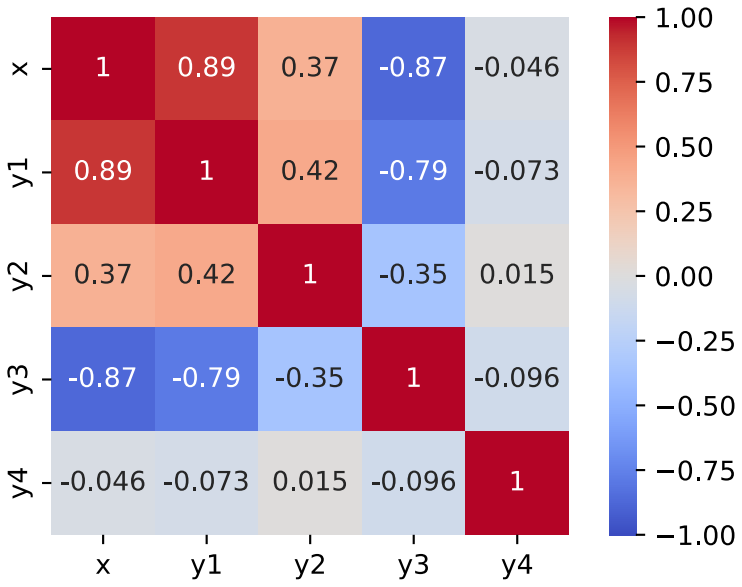
Allerdings haben wir es eben nur **fast** geschafft den Plot zu reproduzieren, da nämlich bei genaueren Hinsehen klar wird, dass die Tile-Labels noch ein paar vorteilhafte Feinheiten aufweisen:

- Die 1en auf der Diagonal sind auf keine Nachkommastelle gerundet
- Einige Werte sind nicht schwarz, sondern weiß, um besser auf dem dunklen Hintergrund zu erscheinen
- Einige Werte sind nicht auf zwei, sondern drei Nachkomma-Stellen gerundet

Aber das soll die Übungsaufgabe für dieses Kapitel sein:

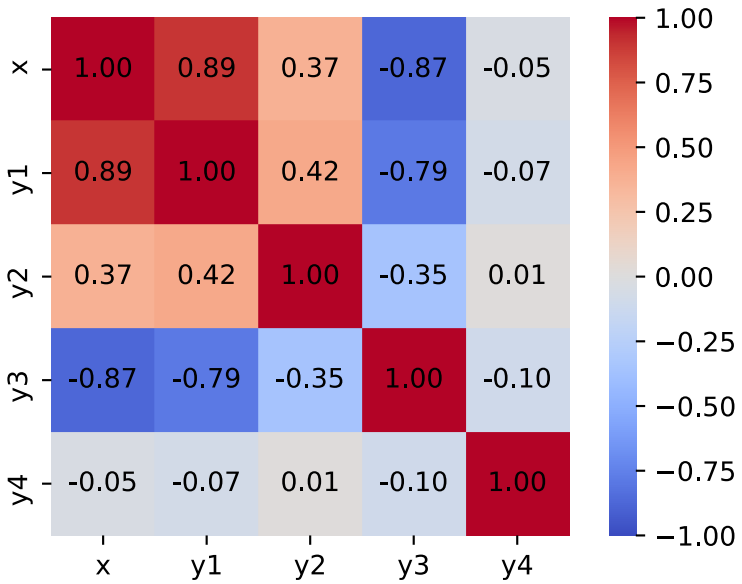
seaborn original

```
fig, ax = plt.subplots()

sns.heatmap(
    data=correlation_matrix,
    square=True,
    annot=True,
    cmap='coolwarm',
    vmin=-1,
    vmax=1,
    center=0,
    ax=ax
)

plt.show()
```



unser matplotlib Nachbau

```
fig, ax = plt.subplots()
im = ax.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='equal')

for spine in ax.spines.values():
    spine.set_visible(False)

ax.set_xticks(np.arange(len(correlation_matrix)))
ax.set_yticks(np.arange(len(correlation_matrix)))
ax.set_xticklabels(correlation_matrix.columns)
ax.set_yticklabels(correlation_matrix.index)
ax.tick_params(axis='y', rotation=90)

cbar = fig.colorbar(mappable=im, ax=ax)
cbar.outline.set_visible(False)

for i in range(len(correlation_matrix)):
    for j in range(len(correlation_matrix)):
        text = ax.text(j, i, f"{correlation_matrix.iloc[i, j]:.2f}",
                       ha="center", va="center", color="black")

plt.show()
```



Übungen

Übung 1

Vervollständige den matplotlib Nachbau der seaborn Heatmap, indem du folgende Feinheiten einbaust:

1. Die 1en auf der Diagonale sollten ohne Nachkommastellen angezeigt werden (also “1” statt “1.00”)
2. Einige der Werte sollen weiß statt schwarz angezeigt werden (siehe seaborn-Plot)
3. Einige der Werte sollen auf drei statt zwei Nachkommastellen gerundet werden (siehe seaborn-Plot)

Hinweis: Es reicht, wenn durch deine Anpassungen/Bedingungen dein Plot für genau diese Heatmap aussieht wie die von seaborn generierte Heatmap. Es muss also nicht für alle möglichen Heatmaps funktionieren.

- (A) Geschafft

Hinweis: Lösungen zu den meisten Übungsaufgaben findet ihr, indem ihr ganz oben rechts im Kapitel auf den `</> Code` Button klickt und dann entsprechend nach ganz unten scrollt. Es ist Absicht, dass dies etwas umständlich ist.