

TSRI-2

July 23, 2024

1 Übung 2.1

Gruppenname: **TSRI**

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.1 2.1.1 Shapes und Co.

Bei vielen Data Science Projekten werden Daten zunächst zugänglich gemacht, sodass sie als Numpy Arrays vorliegen. Das Verständnis von *Shapes* solcher Arrays ist eine wichtige Voraussetzung für eine produktive Arbeit mit diesen Datenstrukturen.

Nehmen Sie die Folien zur heutigen Vorlesung zur Hand.

Ihre Aufgaben

- (1) Betrachten Sie den untenstehenden Code. Beantworten Sie: Welchen **Shape** hat das Numpy Array **a**?

```
[1]: import numpy as np  
a = np.array([5, 10, 15, 20])
```

```
[2]: print(f"Die Shape von `a` ist {a.shape}")
```

Die Shape von `a` ist (4,)

- (2) Lässt sich das Array **a** als (mathematischen) Vektor interpretieren? Wenn **a** ein Vektor ist, können Sie ihn [transponieren](#)?

In der Mathematik ist ein Vektor ein Element eines Vektorraums und ein Vektorraum eine algebraische Struktur, in der die Vektorraumaxiome erfüllt sind. Das NumPy Array **a** lässt sich als Vektor in einem Vektorraum über den Körper K interpretieren, wobei Addition **+** und Multiplikation ***** in K sowie die Vektoraddition **+** und die Skalarmultiplikation ***** sich wie zu erwarten verhalten.

Auch das euklidische Skalarprodukt mit **np.dot** und eine **elementweise** Vektor-Vektor-Multiplikation mit ***** sind definiert, jedoch lässt sich **a** nicht transponieren, da es sich um ein 1-dimensionales Array der Shape (4,0) handelt und die Unterscheidung zwischen Zeilen und Spalten bei nur einer Dimension nicht sinnvoll ist. Zur Transposition ist daher eine Umwandlung

in ein (zumindest) 2-dimensionales Array der Shape (4,1) notwendig. Dies lässt sich z.B. mit `a.reshape((4,1))` bewerkstelligen, aber besser noch so:

```
[3]: a_vec = a[:, np.newaxis]
      print(f"{{a_vec.shape}} und {{a_vec.T.shape}}")
```

```
a_vec.shape=(4, 1) und a_vec.T.shape=(1, 4)
```

- (3) Betrachten Sie das unten stehende Array `b`. Erzeugen Sie aus `b` bitte zwei neue Arrays `c` und `d`: `c` soll ein Spaltenvektor sein, und `d` soll ein Zeilenvektor sein. Wie gehen Sie vor?

```
[4]: b = np.array([2, 4, 6, 8, 10])
      c, d = b[np.newaxis, :], b[:, np.newaxis]
```

Lässt sich `c` bzw. `d` transponieren? Warum?

Ja, weil es sich um 2-dimensionale Arrays handelt.

- (4) Betrachten Sie die dritte Zeile des unten stehenden Codes: Hat sich durch die Zeile `k[0, 1] = 10` das Array `m` geändert? Falls ja, warum? Falls nein, warum hat es sich nicht geändert?

```
[5]: k = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      m = k[:2, 1:3]
      k[0, 1] = 10
```

Ja, denn `m` ist eine Referenz (genannt *view*) auf einen Ausschnitt von `k` und das Element `k[0, 1]` ist in dem Ausschnitt unter der Referenz `m[0, 0]` enthalten. (basic indexing)

- (5) Betrachten Sie die dritte Zeile des unten stehenden Codes: Hat sich durch die Änderung des Arrays `p` der Inhalt von `q` geändert? Falls ja, warum? Falls nein, warum hat es sich nicht geändert?

```
[6]: p = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      q = p[[1, 2], 0]
      p[1, 0] = 25
```

Nein, der Inhalt von `q` hat sich nicht geändert, da beim *advanced indexing* immer eine Kopie der Daten erzeugt wird.

1.2 2.1.2 Broadcasting - Teil 1

Nehmen Sie sich die Folien zur heutigen Vorlesung zur Hand. Unten sehen Sie eine Operation mit zwei Numpy Arrays `a` und `b`.

Ihre Aufgaben

- (1) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array `c` erhalten?
- Tragen Sie Ihre Voraussage in die Variable `shape` ein.
 - Begründen Sie Ihre Vorhersage mit den drei Broadcasting Regeln aus der Vorlesung.
 - Berechnen Sie auf Papier das zu erwartende Ergebnis. Tragen Sie es als Numpy Array in die Variable `ergebnis` ein.
 - Führen Sie dann jeweils die Zelle aus, um zu schauen, ob Sie richtig lagen.

```
[7]: import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = a + b

# Ihre Vorhersagen
shape = (3,)
ergebnis = np.array([5, 7, 9])

print('{}'.format('Shape korrekt.' if np.all(shape==c.shape) else 'Shape nicht\u2192korrekt.'))
print('{}'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else 'Ergebnis nicht korrekt.'))

```

Shape korrekt.

Ergebnis korrekt.

- (2) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array **c** erhalten?

```
[8]: a = np.array([1, 2, 3])
b = np.array([4, 5, 6])[:, np.newaxis]

c = a + b

shape = (3, 3) # Ihre Vorhersage
ergebnis = np.array([
    [5, 6, 7],
    [6, 7, 8],
    [7, 8, 9],
])

print('{}'.format('Shape Korrekt.' if np.all(shape==c.shape) else 'Shape nicht\u2192korrekt.'))
print('{}'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else 'Ergebnis nicht korrekt.'))

```

Shape Korrekt.

Ergebnis korrekt.

- (3) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array **c** erhalten?

```
[9]: a = np.array([[1, 0, 0], [0, 0, 2]])
b = np.ones(12).reshape((4, 1, 3))

c = a * b
```

```

shape = (4, 2, 3) # Ihre Vorhersage

# Regel 1 (padding with ones on left side): a(2,3) -> a(1,2,3)
a_1 = np.array([
    [
        [1, 0, 0], [0, 0, 2]
    ]
])
print(f"a {a.shape} -> {a_1.shape}")

# Regel 2 (stretching from 1 to match the other shape) a(1,2,3) -> a(4,2,3)
a_2 = np.array([
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]]
])
print(f"a {a_1.shape} -> {a_2.shape}")

# Regel 2 (stretching from 1 to match the other shape) b(4,1,3) -> b(4,2,3)
b_1 = np.array([
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]],
])
print(f"b {b.shape} -> {b_1.shape}")

# Elementweise Multiplikation.
ergebnis = a_2

print('{}.'.format('Shape korrekt.' if np.all(shape==c.shape) else 'Shape nicht korrekt.'))
print('{}.'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else 'Ergebnis nicht korrekt.'))

```

```

a (2, 3) -> (1, 2, 3)
a (1, 2, 3) -> (4, 2, 3)
b (4, 1, 3) -> (4, 2, 3)
Shape korrekt.
Ergebnis korrekt.

```

1.3 2.1.3 Broadcasting Teil 2

Ein häufiger Vorverarbeitungsschritt, der Ihnen in verschiedenen Data Science Projekten begegnet wird, ist die Normierung von Daten. Häufig werden dabei Datensätze, z.B. Zeitreihen, auf Mittelwert 0 und Standardabweichung 1 normiert. Dieser Vorgang wird auch *z-scoring* (oder:

Standardisierung) genannt:

Sei x_i der i -te Datenpunkt einer Zeitreihe. Dann ist der z-score dieses Datenpunkts definiert als

$$\tilde{x}_i = \frac{x_i - \bar{x}}{\sigma},$$

wobei \bar{x} der Mittelwert sowie σ die Standardabweichung der Zeitreihe bezeichnen.

Das in der unten stehenden Code-Zelle definierte Numpy Array X enthält 5 Zeitreihen (Spalten) mit je 30 Einträgen (Zeilen).

- Nutzen Sie Broadcasting sowie `np.mean` und `np.std`, um die Zeitreihen auf Mittelwert 0 und Varianz 1 zu normieren. Die transformierten Zeitreihen sollen im Numpy Array Y gespeichert werden. Dabei schreiben Sie nur eine Zeile Code, um die Zeitreihen zu transformieren.
- Beachten Sie das `axis` Argument bei `np.mean` und `np.std`.

```
[10]: import numpy as np
np.random.seed(0)

X = np.random.random((30, 5))

# Ihr Ergebnis:
assert np.mean(X, axis=0).shape == (5,), "5 Zeitreihen sollten 5 Mittelwerte haben"
Y = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
```

1.4 2.1.4 Advanced Indexing

Nehmen Sie sich die Folien der heutigen Vorlesung zur Hand. In diesem Übungsteil geht es darum, Sie mit *Advanced Indexing* in Numpy Arrays vertraut zu machen.

Ihre Aufgaben

(1) Betrachten Sie den unten stehenden Code-Zellen.

- Tragen Sie in der Variable `vorhersage` ein, welches Array Sie in Variable y erwarten.
- Prüfen Sie durch Ausführen des Codes, ob Ihre Vorhersage richtig war.
- Beantworten Sie die Frage: Wodurch wird der Shape des Arrays y bestimmt?

```
[11]: import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]

vorhersage = np.array([1, 4, 5]) # Ihre Vorhersage
print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else
'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.

(2) Betrachten Sie den unten stehenden Code-Zellen.

```
[12]: x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])

rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]

vorhersage = np.array([
    [0, 2], [9, 11]
]) # Ihre Vorhersage

print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else
    'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.

(3) Betrachten Sie den unten stehenden Code-Zellen.

```
[13]: x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])

y = x[1:4,[1,2]] # Kombination von Slicing und Advanced Indexing

vorhersage = np.array([
    [4, 5],
    [7, 8],
    [10, 11]
]) # Ihre Vorhersage

print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else
    'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.

TSRI-3

July 23, 2024

1 Übung 3

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden Sie explorative Datenanalyse (EDA) kennenlernen und Prinzipien guter Visualisierung einsetzen. Nebenbei werden Sie mit den Software-Bibliotheken vertrauter, die Sie für Ihre Data Science Arbeiten nutzen.

1.0.2 3.1 Summary Statistics

Sie haben in der Vorlesung gesehen, dass *Summary Statistics* hilfreich sind, um erste Informationen über einen Datensatz zu erhalten und zusammenzufassen. In dieser Aufgabe werden Sie diese Techniken anwenden und untersuchen.

- Arbeiten Sie mit der Bibliothek *pandas*, die Sie in der letzten Übung kennengelernt haben. Dabei wird es auch darum gehen, Ihre Pandas Fertigkeiten zu festigen und zu vertiefen. Nutzen Sie auch Suchmaschinen und die Dokumentation von Pandas online, um die Befehle für die verschiedenen unten geforderten Arbeitsschritte herauszufinden.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

- (1) Importieren Sie die oben angegebene Datei mithilfe von Pandas. Interpretieren Sie dabei die erste Spalte der CSV-Datei als Index. **Bitte visualisieren Sie die Daten zunächst nicht!** Zur Visualisierung kommen Sie noch im Schritt (6) dieser Übung.

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("mysterious_data.csv", index_col="dataset")
df.head()
```

```
[1]:          x      y
dataset
mystery  55.3846  97.1795
mystery  51.5385  96.0256
mystery  46.1538  94.4872
mystery  42.8205  91.4103
mystery  40.7692  88.3333
```

- (2) Ihr Index enthält die Bezeichnung verschiedener Datensätze. Wie viele verschiedenen Datensätze enthalten Ihre Daten und wie heißen diese? (kurze Angabe bzw. Ausgabe genügt)

```
[2]: print(f"Es sind {df.value_counts('dataset').size} verschiedene Datensätze enthalten:")
print(df.value_counts("dataset"))
```

Es sind 13 verschiedene Datensätze enthalten:

```
dataset
away           142
bullseye       142
circle          142
dots            142
h_lines          142
high_lines       142
mystery          142
slant_down       142
slant_up          142
star             142
v_lines           142
wide_lines        142
x_shape           142
Name: count, dtype: int64
```

- (3) Wie heißen Ihre Spalten?

Die Spalten heißen `dataset`, `x` und `y`.

- (4) Bestimmen Sie die Summary Statistics “Mittelwert” und “Standardabweichung” für jede Spalte eines jeden Datensatzes einzeln sowie den Korrelationskoeffizienten zwischen den beiden Spalten für jeden Datensatz und geben Sie diese drei Summary Statistics auf zwei Nachkommastellen genau an.

```
[3]: from IPython.display import Markdown, display

def ausgabe(name, data):
    pd.options.display.float_format = "{:.2f}".format
    display(Markdown(f"## {name}:\n{data}\n##\n"))

df_grouped_1 = df.groupby("dataset").agg(["mean", "std"])
df_grouped_2 = df.groupby("dataset").corr().reset_index()
```

```

df_grouped_1["corr"] = df_grouped_2[df_grouped_2["level_1"] == "x"] .
    ↪set_index("dataset")["y"]
ausgabe("Summary Statistics", df_grouped_1)

```

1.1 Summary Statistics:

dataset	x	y	corr		
	mean	std	mean	std	
away	54.27	16.77	47.83	26.94	-0.06
bullseye	54.27	16.77	47.83	26.94	-0.07
circle	54.27	16.76	47.84	26.93	-0.07
dots	54.26	16.77	47.84	26.93	-0.06
h_lines	54.26	16.77	47.83	26.94	-0.06
high_lines	54.27	16.77	47.84	26.94	-0.07
mystery	54.26	16.77	47.83	26.94	-0.06
slant_down	54.27	16.77	47.84	26.94	-0.07
slant_up	54.27	16.77	47.83	26.94	-0.07
star	54.27	16.77	47.84	26.93	-0.06
v_lines	54.27	16.77	47.84	26.94	-0.07
wide_lines	54.27	16.77	47.83	26.94	-0.07
x_shape	54.26	16.77	47.84	26.93	-0.07

- (5) Wie stark unterscheiden sich die Datensätze in den Summary Statistics (zwei Nachkommastellen) aus Schritt (4)? Es reicht aus, wenn Sie die Unterschiede/Ähnlichkeiten in Worten beschreiben.

Die Summary Statistics der Datensätze sind sehr ähnlich und unterscheiden sich höchstens mal in der zweiten Nachkommastelle um 0,01.

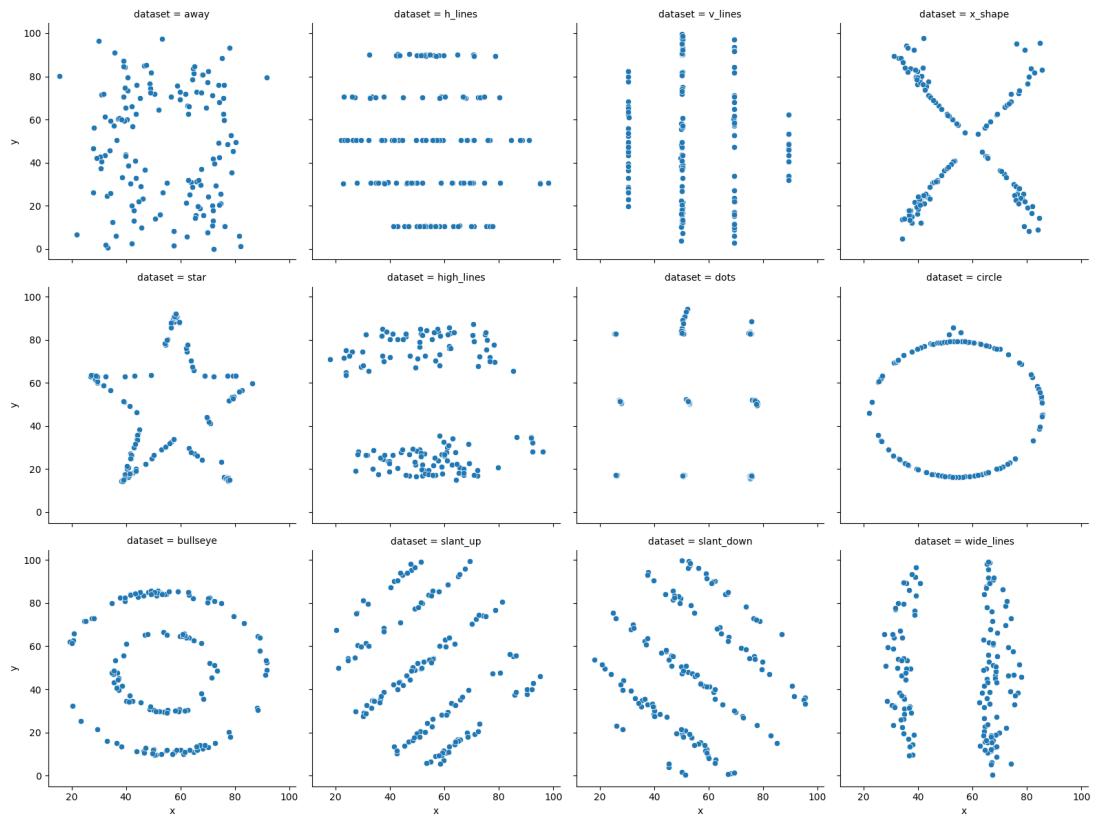
- (6) Visualisierung: Erstellen Sie für jeden Datensatz **außer “mystery”** einen Scatter-Plot. Unterscheiden sich die Datensätze? Falls ja, gibt es große oder kleine Unterschiede? (1-5 Worte)

```

[4]: import seaborn as sns
df_without_mystery = df.loc[~df.index.isin(["mystery"])]


# pandas-only:
#df_without_mystery.groupby("dataset").plot(x="x", y="y", subplots=True, ↪
#    layout=(3, 4), sharex=True, sharey=True, kind="scatter")
# Seaborn:
sns.relplot(kind='scatter', data=df_without_mystery, x='x', y='y', ↪
    col='dataset', col_wrap=4, height=4)
plt.show()

```



Die Datensätze unterscheiden sich erheblich.

- (7) Nehmen Sie sich die Folien zur heutigen Vorlesung zur Hand: Benennen Sie nun den Datensatz aus der Vorlesung, bei dem Sie ein ähnliches Phänomen beobachtet haben. (2 Worte)

Die 1971 vom englischen Statistiker Francis Anscombe erzeugten Daten, welche unter dem Namen *Anscombes Quartett* bekannt sind, unterscheiden sich ebenfalls kaum in ihren Summary Statistics, weisen aber große Unterschiede bei ihrer Visualisierung auf.

- (8) Visualisieren Sie nun den Datensatz **mystery** in einem Scatter-Plot.

```
[5]: df[df.index == "mystery"].plot(x="x", y="y", kind="scatter", c=["#488211"],  
    title="dataset = mystery")  
plt.show()
```



Damit darf ich Ihnen gratulieren. Sie haben gerade einen bekannten Datensatz kennengelernt.

Herzlichen Glückwunsch. Sie haben soeben den “DataSaurus” entdeckt. Dieses Geschöpf stammt ursprünglich von Alberto Cairo, der an der University of Miami *Visuellen Journalismus* lehrt, und tauchte zum ersten Mal in einem Tweet von Cairo auf: <https://twitter.com/albertocairo/status/765167969139765250/photo/1>

Seine Message ist klar: **Don't trust summary statistics. Always visualize your data first.**

Der DataSaurus hat es inzwischen zu kleiner Berühmtheit erlangt. Hinter dem nachfolgenden Link können Sie sehen, wie der Datensatz, den Sie gerade analysiert haben, entstanden ist: <https://www.autodeskresearch.com/publications/samestats>

1.1.1 3.2 Visualisierung Teil 1 (Weltgesundheit)

Die Visualisierung von Daten gehört zu den Tätigkeiten, die Sie als Data Scientist sehr häufig ausführen werden. Das Erstellen aussagekräftiger Abbildungen ist zeitintensiv, wird Ihnen aber mit wachsender Übung immer leichter fallen.

In dieser Übung werden Sie einen Datensatz visualisieren und interpretieren, der in den letzten Jahren oft diskutiert wurde. Sie werden untersuchen, wie es um die Gesundheit und den Wohlstand der Weltbevölkerung (zurzeit etwa 7,6 Milliarden Menschen) steht. Die Daten stammen aus

unterschiedlichen Quellen und wurden von der gemeinnützigen Gapminder-Stiftung zusammengeführt, die es sich zum Ziel gemacht hat, Menschen mit Statistiken über den Zustand der Welt aufzuklären.

Ihre Aufgaben

- (1) Führen Sie die unten angeführte Code-Zelle aus, um die Daten zu importieren.

```
[6]: #!pip install gapminder
from gapminder import gapminder
gapminder.head()
```

```
[6]:      country continent  year  lifeExp      pop  gdpPercap
0  Afghanistan      Asia  1952    28.80  8425333    779.45
1  Afghanistan      Asia  1957    30.33  9240934    820.85
2  Afghanistan      Asia  1962    32.00 10267083    853.10
3  Afghanistan      Asia  1967    34.02 11537966    836.20
4  Afghanistan      Asia  1972    36.09 13079460    739.98
```

- (2) Welche Spalten sind in Ihrem Datensatz enthalten? (kurze Liste der Spalten)

Enthalten sind die Spalten `country`, `continent`, `year`, `lifeExp`, `pop`, `gdpPercap`.

- (3) Machen Sie eine Kurzrecherche und klären Sie für sich: Was ist “GDP”? Was bedeutet “per capita”? (1-2 Sätze)

→ `gdpPercap` ist das inflationsbereinigte Bruttoinlandsprodukt pro Einwohner, welches in PPP Dollar angegeben ist.

Aus der Gapminder Dokumentation heißt es dazu:

GDP per capita in constant PPP dollars

GDP per capita measures the value of everything produced in a country during a year, divided by the number of people. The unit is constant dollars adjusted for inflation in 2017’s prices. As the cost of living varies across countries, we use a currency called “international dollars”, which is adjusted for Purchasing Power Parity (PPP). This is a virtual currency that enables better comparisons. Such a dollar would buy in each country a comparable amount of goods and services as a U.S. dollar would buy in the United States. GDP per capita is the GDP divided by the population of the country, which gives a rough estimate of the average annual income of the citizens.

[Gapminder Documentation](#)

- (4) Erstellen Sie eine Abbildung (Scatter-Plot), auf der Sie die Lebenserwartung (y-Achse) gegen GDP per capita (x-Achse) für jedes Land auftragen - *und zwar für das im Datensatz enthaltene neueste Jahr*. Skalieren Sie die x-Achse logarithmisch. Skalieren Sie die Größe der Punkte proportional zur Population des Landes.

```
[7]: df_1: pd.DataFrame = gapminder.sort_values("year", ascending=False).
    ↪groupby("country").first()

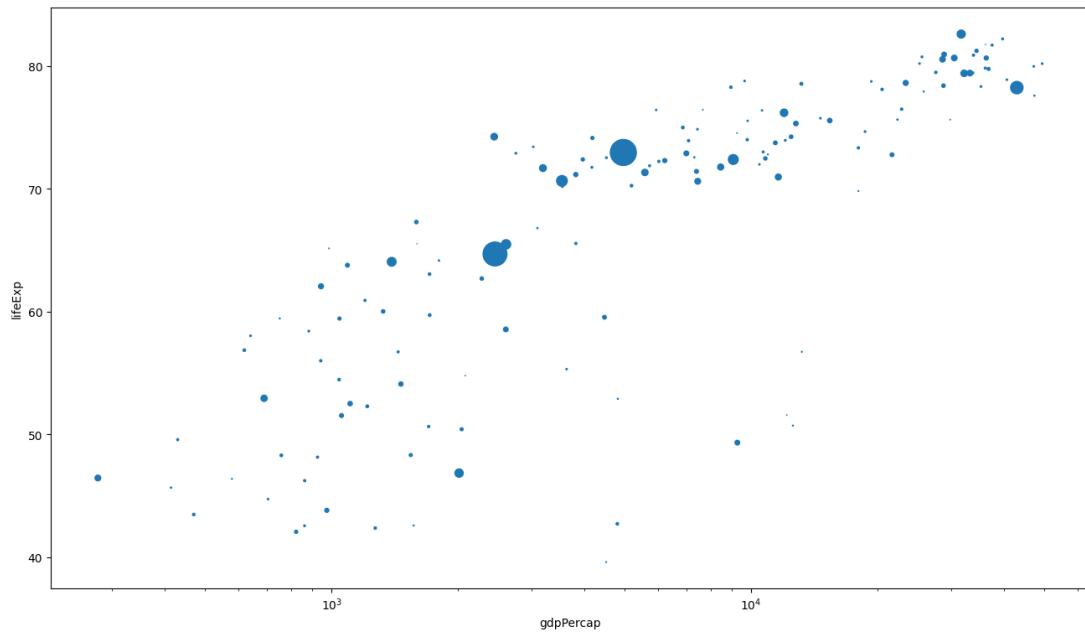
# Normalisieren
```

```

df_1["pop_norm"] = df_1["pop"] / df_1["pop"].max()

df_1.plot.scatter(x="gdpPercap", y="lifeExp", s=df_1["pop_norm"] * 500, □
    ↪logx=True, figsize=(16, 9))
plt.show()

```



- (5) Interpretieren Sie Ihre Abbildung. Sehen Sie Zusammenhänge zwischen Lebenserwartung und GDP per Capita? (1-2 Sätze)

Bis auf wenige Ausreißer, bei denen trotz hohem GDP per Capita eine Lebenserwartung von < 60 Jahren vorliegt, scheint eine exponentielle Korrelation vorzuliegen. Je höher das GDP per Capita, desto höher auch die Lebenserwartung.

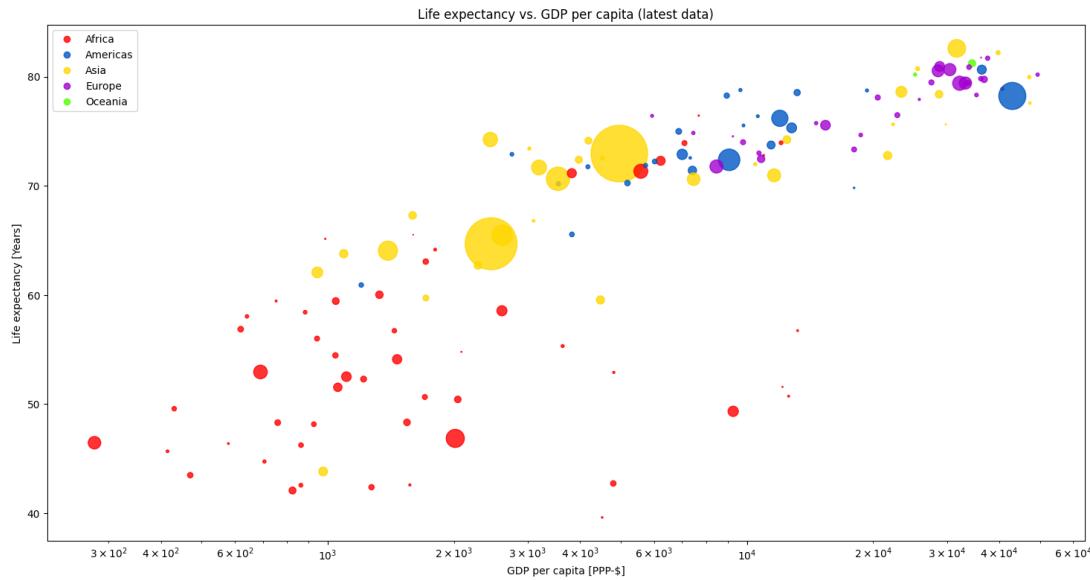
- (6) Erweitern Sie Ihre Abbildung aus Schritt (4), in dem Sie die Informationen über die Kontinente integrieren: Identifizieren Sie zunächst die Kontinente, die im Datensatz angeführt werden. Färben Sie die Punkte der Länder gemäß Ihrer Kontinentzugehörigkeit ein.

```
[8]: from matplotlib.ticker import LogFormatterSciNotation
fig, ax = plt.subplots(figsize=(18, 9))
labels, index = np.unique(df_1["continent"], return_inverse=True)
sc = ax.scatter(df_1["gdpPercap"], df_1["lifeExp"], marker='o', □
    ↪s=df_1["pop_norm"] * 3000, c=index, cmap="prism", alpha=0.8)
ax.legend(sc.legend_elements()[0], labels)
plt.title("Life expectancy vs. GDP per capita (latest data)")
plt.xscale("log")
plt.xlabel("GDP per capita [PPP-$]")
plt.ylabel("Life expectancy [Years]")
```

```

plt.tick_params(axis='x', which='minor')
ax.xaxis.set_minor_formatter(LogFormatterSciNotation(labelOnlyBase=False,
    minor_thresholds=(10, 2)))
plt.show()

```



- (7) Interpretieren Sie Ihre Abbildung aus Schritt (6). Gibt es Tendenzen bezüglich der Kontinentzugehörigkeit der Länder? Falls ja, welche? (1-3 Sätze).

Es sind einige Tendenzen zu erkennen, so haben die Länder auf dem afrikanischen Kontinent viel häufiger eine geringere Lebenserwartung und einen viel geringeren GDP per Capita.

- (8) Erzeugen Sie eine neue Abbildung, die dieselben Informationen wie Ihre Abbildung in Schritt (6) zeigt, nur aus dem Jahr 1967. Vergleichen Sie diese Abbildung mit der aus Schritt (6). Was hat sich verändert und wie interpretieren Sie dies? (1-3 Sätze)

```

[9]: df_2= pd.DataFrame = gapminder[gapminder["year"] == 1967].copy()
df_2["pop_norm"] = df_2["pop"] / df_2["pop"].max()

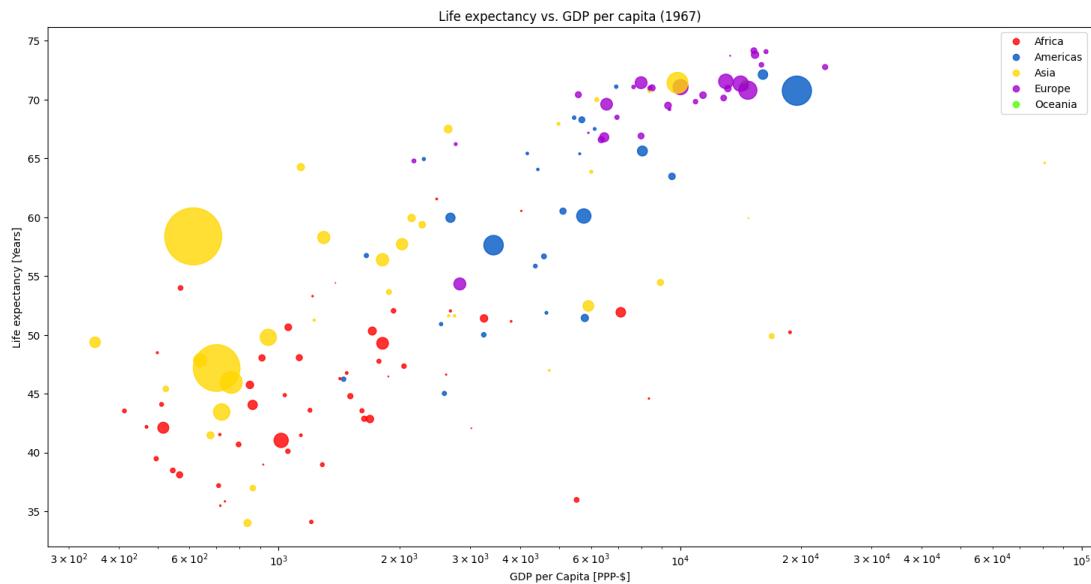
fig, ax = plt.subplots(figsize=(18, 9))
labels, index = np.unique(df_2["continent"], return_inverse=True)
sc = ax.scatter(df_2["gdpPercap"], df_2["lifeExp"], marker='o',
    s=df_2["pop_norm"] * 3000, c=index, cmap="prism", alpha=0.8)
ax.legend(sc.legend_elements()[0], labels)
plt.title("Life expectancy vs. GDP per capita (1967)")
plt.xscale("log")
plt.xlabel("GDP per Capita [PPP-$]")
plt.ylabel("Life expectancy [Years]")
plt.tick_params(axis='x', which='minor')

```

```

ax.xaxis.set_minor_formatter(LogFormatterSciNotation(labelOnlyBase=False,
    minor_thresholds=(10, 2)))
plt.show()

```



Die Lebenserwartung einiger Länder, besonders in Asien, ist hochgegangen, während ihr pro Kopfeinkommen ebenfalls gestiegen ist.

- (9) [Optional] Machen Sie Ihre Abbildung interaktiv. Nutzen Sie die `interact` Funktion, um eine Abbildung zu erzeugen, in der Sie mithilfe eines Reglers das Jahr einstellen können, für das die Abbildung erzeugt werden soll. Wie verändern sich Lebenserwartung und GDP per Capita über die Jahre hinweg? (1-3 Sätze)
- Hinweis: Um diese Teilaufgabe umzusetzen, recherchieren Sie im Netz und lesen Sie unter anderem [hier](#) nach, wie Sie `interact` nutzen können.

```
[10]: from ipywidgets import interact
import ipywidgets as widgets

years = np.unique(gapminder["year"])

def plot_year(year: int, save_output: bool = False) -> None:
    df_3: pd.DataFrame = gapminder[gapminder["year"] == year].copy()
    df_3["pop_norm"] = df_3["pop"] / df_3["pop"].max()
    fig, ax = plt.subplots(figsize=(14, 8.4))
    labels, index = np.unique(df_3["continent"], return_inverse=True)
    sc = ax.scatter(df_3["gdpPerCap"], df_3["lifeExp"], marker='o', s=df_3["pop_norm"] * 3000, c=index, cmap="viridis", alpha=0.8)
    ax.legend(sc.legend_elements()[0], labels, loc="lower right")
```

```

plt.title(f"Life expectancy vs. GDP per capita ({year})")
plt.xscale("log")
plt.xlabel("GDP per Capita [PPP-$]")
plt.ylabel("Life expectancy [Years]")
plt.grid(visible=True, which='both', axis='x', alpha=0.05)
plt.grid(visible=True, which='major', axis='y', alpha=0.08)
plt.xlim((3e2, 6e4))
plt.ylim((25, 90))
plt.tick_params(axis='x', which='minor')
ax.xaxis.set_minor_formatter(LogFormatterSciNotation(labelOnlyBase=False,
                                                     minor_thresholds=(10, 1)))
if save_output:
    plt.savefig(f"life_expectancy/{year}.png")
else:
    plt.show()

#interact(plot_year, year=widgets.IntSlider(min=1952, max=2007, step=5))

```

1.1.2 3.3 Visualisierung Teil 2 (Geburtenraten)

In dieser Aufgabe werden Sie Ihre Fähigkeiten, Daten zu visualisieren, weiter verfeinern. Wir werden uns wieder Daten der Gapminder Stiftung anschauen.

- Da diese Übung an die obere Übung anschließt, gehe ich davon aus, dass Sie nun vertrauter mit der Visualisierung von Daten sind. Daher sind die Anweisungen in dieser Übungsaufgabe etwas freier gehalten.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#) (Datensatz 1) und [hier](#) (Datensatz 2).

Randbemerkung

Im Rahmen dieser Aufgabe wollen wir unter dem Begriff “Geburtenrate” die Gesamtfertilitätsrate (*total fertility rate*) verstehen, wie Sie sie direkt im “Datensatz 2” angegeben finden. Dies ist die durchschnittliche Anzahl von Kindern, die eine gebärfähige Person im Laufe ihres Lebens bekommt. Die Forschung unterscheidet allerdings zwischen verschiedenen Fertilitätsindikatoren (z.B. zwischen sogenannten Geburtenziffern, Fertilitätsraten und Reproduktionsraten). Falls Sie also einmal mit einem Bevölkerungswissenschaftler sprechen sollten: Sie untersuchen hier im Rahmen dieser Aufgabe die Gesamtfertilitätsrate.

Ihre Aufgaben

- (1) Importieren Sie die oben aufgeführten Datensätze.

```
[11]: df_4 = pd.read_csv("child_mortality_0_5_year olds_dying_per_1000_born.csv")
df_5 = pd.read_csv("children_per_woman_total_fertility.csv")
```

- (2) Interpretieren Sie die Dateinamen und schauen Sie in die Datensätze: Was enthalten Ihre Datensätze für Daten?

```
[12]: print(df_4.columns, df_4.shape)
print(df_5.columns, df_5.shape)

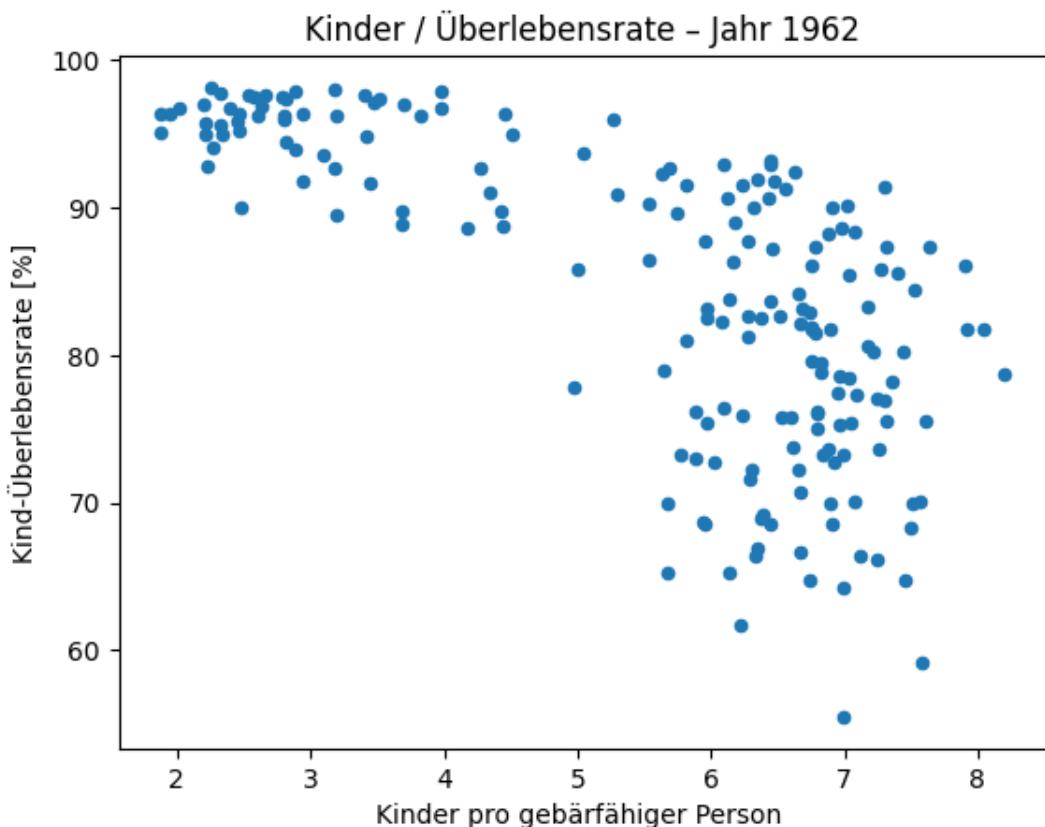
Index(['country', '1800', '1801', '1802', '1803', '1804', '1805', '1806',
       '1807', '1808',
       ..
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017',
       '2018'],
      dtype='object', length=220) (193, 220)
Index(['country', '1800', '1801', '1802', '1803', '1804', '1805', '1806',
       '1807', '1808',
       ..
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017',
       '2018'],
      dtype='object', length=220) (184, 220)
```

Die Datensätze enthalten für 193 respektive 184 Länder Zahlen zu der Kindersterblichkeit in den Jahren von 1800 bis 2018. Dabei scheint der erste Datensatz die Sterblichkeit von Kindern vor dem 5. Lebensjahr pro 1.000 (lebend) Geburten zu enthalten und der zweite Datensatz die Gesamtfertilitätsrate. (*The number of children that would be born to each woman with prevailing age-specific fertility rates.*)

- (3) Erstellen Sie die erste Abbildung: Tragen Sie in einem Scatter-Plot die Kind-Überlebensrate (in Prozent) (y-Achse) gegen die Kinder pro gebärfähiger Person (x-Achse) für jedes aufgeführte Land für das Jahr 1962 ein.

```
[13]: df = df_4[df_4["1962"].notna()][["country", "1962"]].set_index("country").
      join(df_5[df_5["1962"].notna()][["country", "1962"]].set_index("country"), 
            lsuffix="_mort", rsuffix="_fert")

# Überlebensrate in Prozent = 1 - Sterberate in Prozent
df["1962_surv"] = (1 - (df["1962_mort"] / 1000)) * 100
df.plot(x="1962_fert", y="1962_surv", kind="scatter")
plt.title("Kinder / Überlebensrate - Jahr 1962")
plt.xlabel("Kinder pro gebärfähiger Person ")
plt.ylabel("Kind-Überlebensrate [%]")
plt.show()
```



- (4) Erweitern Sie die Abbildung aus Schritt (3), indem Sie die Punkte des Scatterplots proportional zur Populationsgröße des jeweiligen Landes skalieren. Dazu werden Sie Informationen aus Übung 3.2 verwenden müssen.

```
[14]: def show_for_year(year, df_4, df_5):
    df = df_4[df_4[year].notna()][["country", year]].set_index("country").
    ↪join(df_5[df_5[year].notna()][["country", year]].set_index("country"), ↪
    ↪lsuffix="_mort", rsuffix="_fert")

    df[f"year_surv"] = 1 - (df[f"year_mort"] / 1000)

    df_5: pd.DataFrame = gapminder[gapminder["year"] == int(year)].copy()
    df_5["pop_norm"] = df_5["pop"] / df_5["pop"].std()

    df_6 = df.join(df_5.set_index("country"))
    df_7 = df_6[~df_6["lifeExp"].isna()]

    print(f"Es wurden zu den {df.shape[0]} Ländern aus dem Kinder-Geburten/
    ↪Sterberaten-Datensatz nur {df_7.shape[0]} Populationsgrößen aus dem
    ↪Gapminder Datensatz zugeordnet.")
```

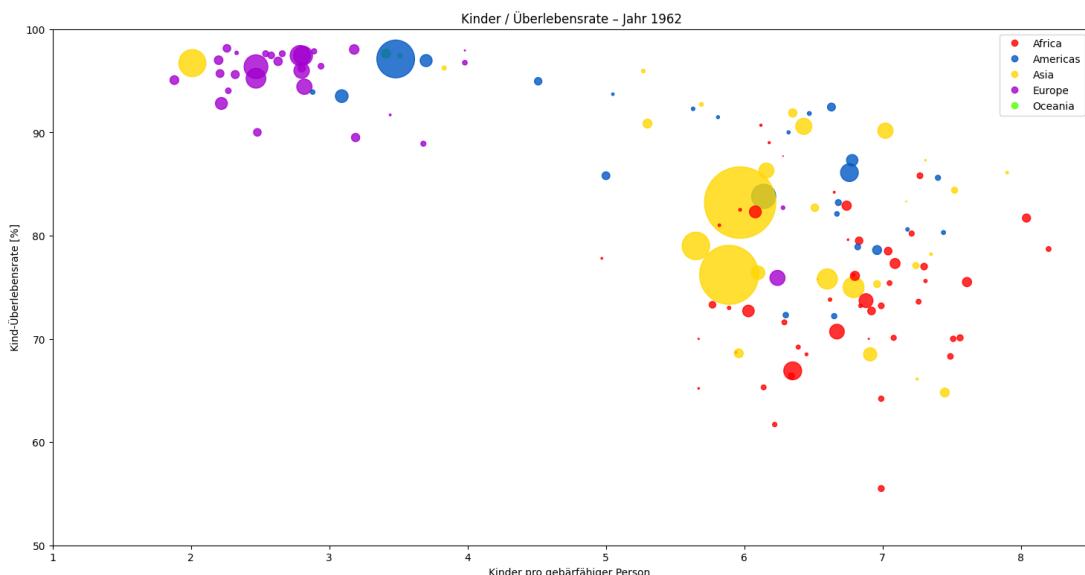
```

fig, ax = plt.subplots(figsize=(18, 9))
labels, index = np.unique(df_7["continent"], return_inverse=True)
sc = ax.scatter(df_7[f"{year}_fert"], df_7[f"{year}_surv"] * 100, c=index, marker='o', s=df_7["pop_norm"] * 500, cmap="prism", alpha=0.8)
ax.legend(sc.legend_elements()[0], labels)
plt.title(f"Kinder / Überlebensrate - Jahr {year}")
plt.xlabel("Kinder pro gebärfähiger Person")
plt.ylabel("Kind-Überlebensrate [%]")
plt.ylim((50, 100))
plt.xlim((1, 8.5))
plt.show()

show_for_year("1962", df_4, df_5)

```

Es wurden zu den 186 Ländern aus dem Kinder-Geburten/Sterberaten-Datensatz nur 134 Populationsgrößen aus dem Gapminder Datensatz zugeordnet.



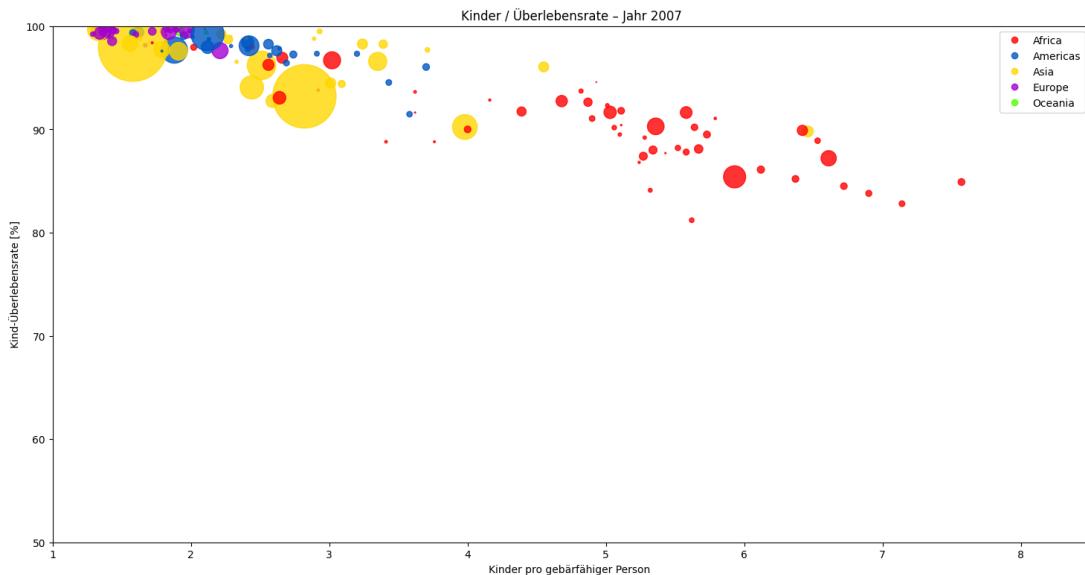
- (5) Interpretieren Sie die Abbildung aus Schritt (4). Können Sie verschiedene Cluster von Ländern erkennen? (1-5 Sätze)

Es lassen sich zwei Cluster ausmachen. Das eine ist im Bereich von 2-4 Kindern pro Person mit Überlebensraten von über 90% und das andere Cluster liegt im Bereich von 5.5-8 Kindern pro Person, wobei die Überlebensraten hier zwischen 60 und 90% verteilt sind.

- (6) Erstellen Sie eine weitere Abbildung wie in Schritt (4), allerdings für das Jahr 2007.

[15]: show_for_year("2007", df_4, df_5)

Es wurden zu den 193 Ländern aus dem Kinder-Geburten/Sterberaten-Datensatz nur 134 Populationsgrößen aus dem Gapminder Datensatz zugeordnet.



- (7) Vergleichen Sie Ihre Abbildungen aus Schritt (4) und Schritt (6): Wie haben sich die Daten verändert? Können Sie für das Jahr 2007 weiterhin Cluster erkennen? (1-5 Sätze)

Im Bereich von 1,2 bis 2,5 Kindern pro Person gibt es ein Cluster bei einer Überlebensrate von über 97 %. Insgesamt ist sowohl die Kindersterblichkeit, als auch die Geburtenrate, bei den meisten Ländern stark gesunken. Die Überlebensrate ist 2007 in jedem Land über 85 %, was ein großer Fortschritt im Vergleich zu 1962 ist, jedoch lässt sich auch eine Separation der Länder Afrikas feststellen, bei denen die Überlebensraten zwar auch gestiegen und die Geburtenraten gesunken sind, aber bei weitem nicht im gleichen Maße wie auf anderen Kontinenten.

- (8) Ziehen Sie ein Fazit: Beschreiben Sie in wenigen Sätzen, was Sie aus den beiden Abbildungen lernen.

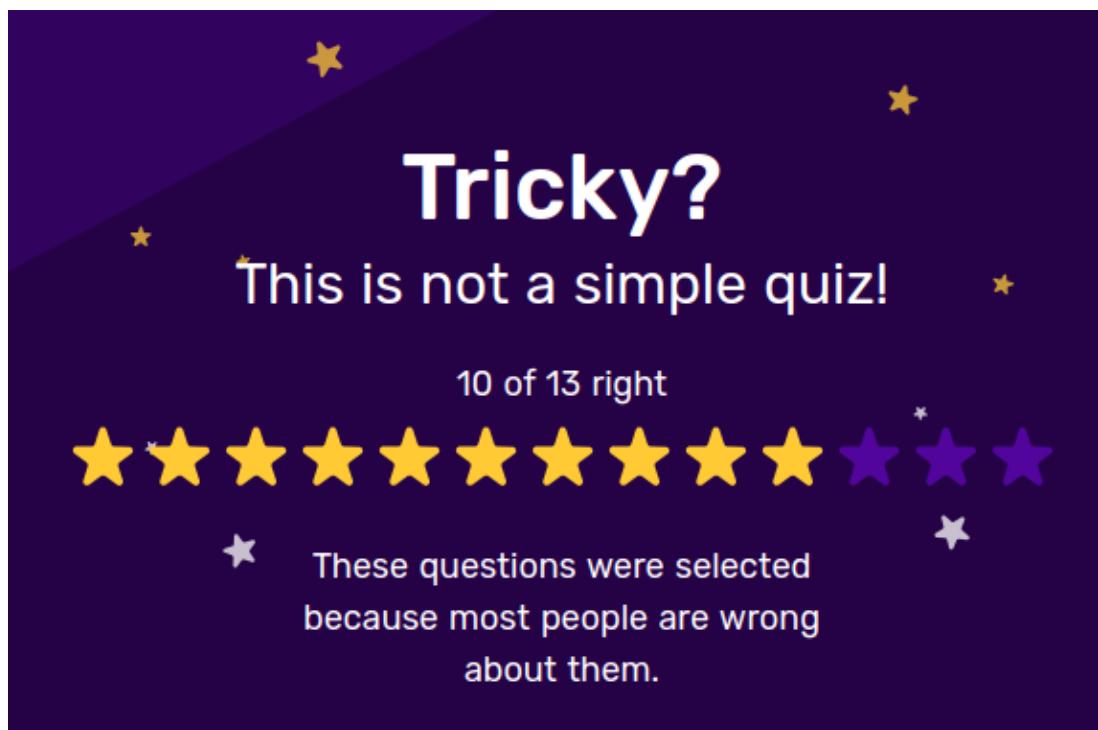
Je kleiner die Kindersterblichkeit ist, desto geringer wird mit einem zeitlichen Versatz auch die Geburtenrate.

1.1.3 3.4 Visualisierung Teil 3

Sie haben in den letzten beiden Aufgaben Daten der Gapminder Stiftung visualisiert. In dieser Aufgabe geht es darum, Daten selbstständig zu analysieren, zu visualisieren und Ihre Erkenntnisse aufzubereiten.

Ihre Aufgaben

- (1) Sofern noch nicht geschehen: Tun Sie sich mit einem Kollegen/einer Kollegin zusammen.
- (2) Testen Sie Ihr Vorwissen: Absolvieren Sie gemeinsam den Test der Gapminder-Stiftung, der [hier](#) hinterlegt ist. Wie viele der Fragen konnten Sie richtig beantworten?



- (3) Stöbern Sie gemeinsam in den Datenquellen der Gapminder Stiftung unter <https://www.gapminder.org/data/>. (Scrollen Sie auf der verlinkten Seite weiter nach unten, um die Datensammlung zu sehen: "List of indicators in Gapminder Tools").
- (4) Identifizieren Sie Datensätze, die Sie interessant finden. Führen Sie eine explorative Analyse durch, d.h. laden und importieren Sie die Daten, stellen Sie Fragen auf, und erstellen Sie entsprechende Visualisierungen, interpretieren Sie Ihre Befunde.

```
[16]: # Suicide woman/men per 100_000 people
# https://www.who.int/violence_injury_prevention/en/
suicides_men = pd.read_csv("suicide_men_per_100000_people.csv")
suicides_women = pd.read_csv("suicide_women_per_100000_people.csv")
```

```
# Individuals using the Internet (% of population)
# https://data.worldbank.org/indicator/IT.NET.USER.ZS
internet_users = pd.read_csv("internet_users.csv")
```

```
[17]: print(suicides_men.isnull().sum().values)
print(suicides_women.isnull().sum().values)
```

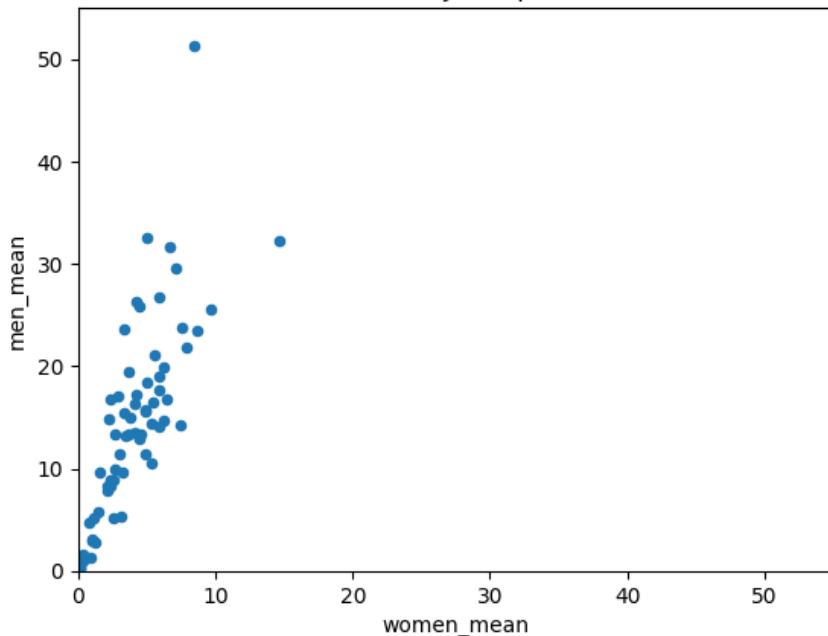
0	108	101	98	97	96	87	87	86	84	84	82	78	80	76	78	80	79
74	74	73	76	74	75	74	74	73	73	74	71	71	65	56	52	65	62
40	47	47	50	52	49	55	53	52	52	55	61	62	61	62	63	63	65
68	68	69	73	66	65	67	68	63	67	68	68	78	104				
0	101	94	91	90	89	80	80	79	78	77	76	73	74	70	72	73	72
69	67	68	70	69	68	69	70	68	69	68	67	68	63	51	48	59	57
38	41	42	43	45	44	49	46	46	46	49	54	55	54	56	57	56	58

```
61 61 63 66 59 58 60 61 59 61 61 62 71 97]
```

```
[18]: women_mean = suicides_women.set_index("country").loc[:, "2005":"2015"].
    ↪mean(axis=1, skipna=True).dropna()
men_mean = suicides_men.set_index("country").loc[:, "2005":"2015"].mean(axis=1,
    ↪skipna=True).dropna()

suicides = pd.concat([women_mean.rename("women_mean"), men_mean.
    ↪rename("men_mean")], axis="columns", join="inner")
suicides.plot(x="women_mean", y="men_mean", kind="scatter")
plt.title("Mittlere Anzahl an Suizide der letzten 10 Jahre pro 100k Einwohner nach Geschlecht")
plt.xlim((0, 55))
plt.ylim((0, 55))
plt.show()
```

Mittlere Anzahl an Suizide der letzten 10 Jahre pro 100k Einwohner nach Geschlecht



```
[19]: internet_users_max = internet_users.set_index("country").loc[:, "2005":"2015"].
    ↪max(axis=1).dropna()
```

```
[20]: internet_users_max.name = "max_internet_users_percent"
```

```
[21]: suicides = suicides.join(internet_users_max)
```

```
[22]: suicides["suicides_per_100k"] = suicides["women_mean"] + suicides["men_mean"]
```

```
[23]: suicides
```

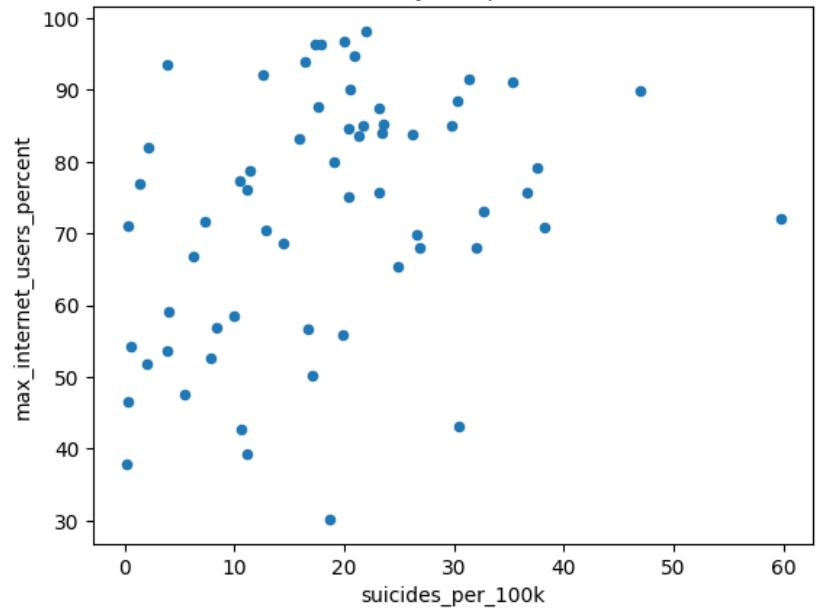
```
[23]:      women_mean  men_mean  max_internet_users_percent \
country
Albania          3.08      5.30           56.90
Armenia          0.99      3.06           59.10
Australia        4.87     15.54           84.60
Austria          5.01     18.40           83.90
Azerbaijan      0.29      1.06           77.00
...
Tunisia          ...       ...           ...
Turkey           0.12      0.26           46.50
USA              0.96      2.88           53.70
Uzbekistan      4.15     16.30           75.00
South Africa    2.34      8.22           42.80
South Africa    0.38      1.65           51.90

      suicides_per_100k
country
Albania          8.38
Armenia          4.06
Australia        20.41
Austria          23.41
Azerbaijan      1.35
...
Tunisia          ...       ...
Turkey           0.38
USA              3.84
Uzbekistan      20.45
South Africa    10.56
South Africa    2.03
```

[66 rows x 4 columns]

```
[24]: suicides.plot(x="suicides_per_100k", y="max_internet_users_percent", ↴
    ↴kind="scatter")
plt.title("Mittlere Anzahl an Suizide der letzten 10 Jahre pro 100k Einwohner ↴
    ↴nach Internetzugang")
plt.show()
```

Mittlere Anzahl an Suizide der letzten 10 Jahre pro 100k Einwohner nach Internetzugang



TSRI-4

July 23, 2024

1 Übung 4

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

werden Sie verschiedene Datensätze explorieren und dabei Ihre EDA-Fertigkeiten trainieren.

1.0.2 4.1 Visualisierung Teil 1 (Lebenserwartung)

In dieser Übung geht es darum, dass Sie Ihre Pandas- und Visualisierungs-Fähigkeiten weiter verfeinern. Wir werden die Entwicklung der Lebenserwartung der Weltbevölkerung in den letzten 200 Jahren untersuchen.

Ihre Daten

Die Daten stammen aus unterschiedlichen Datenquellen (unter anderem den United Nations, dem Institute for Health Metrics and Evaluation, der Human Mortality Database des Max Plank Instituts für demografische Forschung und der University of California Berkeley), die von der Gapminder Stiftung zusammengefasst wurden. Sie finden die Daten hier:

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

- (1) Welche Erwartungshaltung haben Sie - ohne dass Sie vorher recherchieren: Was war die durchschnittliche Lebenserwartung eines Menschen zu Beginn des 19. Jahrhunderts (also 18XX) und im Jahr 2018? (1-3 Sätze)

35 Jahre

- (2) Importieren und untersuchen Sie den hinterlegten Datensatz: Für welche Zeitspanne sind Daten hinterlegt?

```
[1]: import pandas as pd  
life_expectancy = pd.read_csv("life_expectancy_years.csv")
```

```
[2]: life_expectancy.set_index("country", inplace=True)
print("Zeitspanne: vom Jahr", life_expectancy.columns.min(), "bis zum Jahr", life_expectancy.columns.max())
life_expectancy.head(3)
```

Zeitspanne: vom Jahr 1800 bis zum Jahr 2018

```
[2]:          1800  1801  1802  1803  1804  1805  1806  1807  1808  1809  ... \
country
Afghanistan  28.2  28.2  28.2  28.2  28.2  28.2  28.1  28.1  28.1  28.1 ...
Albania      35.4  35.4  35.4  35.4  35.4  35.4  35.4  35.4  35.4  35.4 ...
Algeria       28.8  28.8  28.8  28.8  28.8  28.8  28.8  28.8  28.8  28.8 ...

          2009  2010  2011  2012  2013  2014  2015  2016  2017  2018
country
Afghanistan  55.7  56.2  56.7  57.2  57.7  57.8  57.9  58.0  58.4  58.7
Albania      75.9  76.3  76.7  77.0  77.2  77.4  77.6  77.7  77.9  78.0
Algeria       76.3  76.5  76.7  76.8  77.0  77.1  77.3  77.4  77.6  77.9
```

[3 rows x 219 columns]

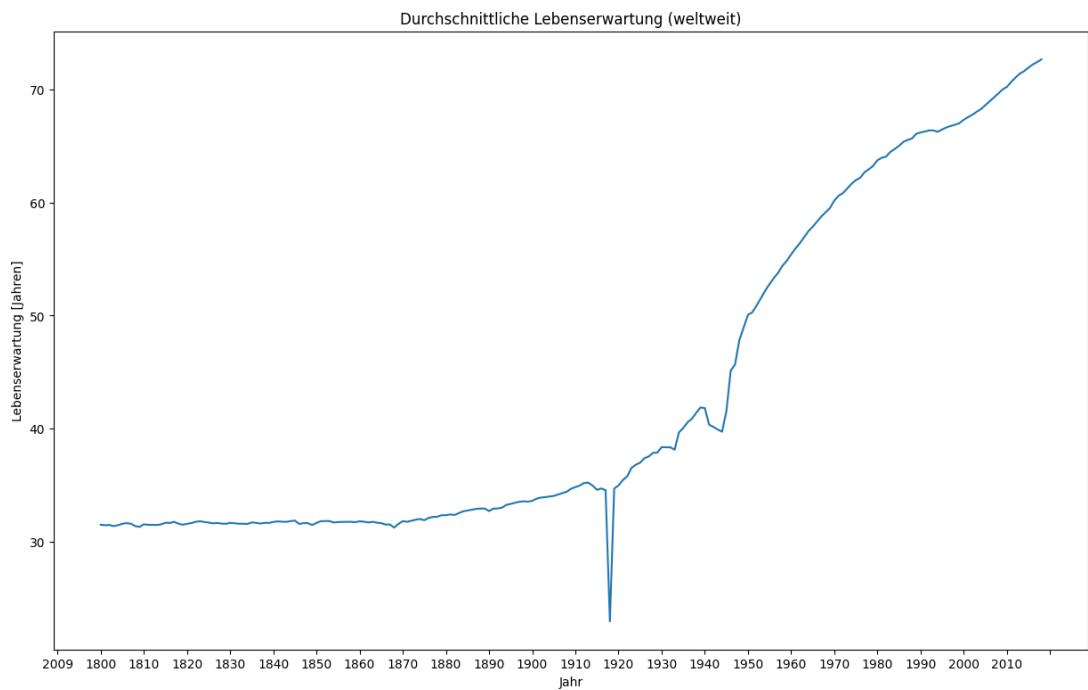
Für den Zeitraum von 1800 bis 2018 in Jahresschritten.

- (3) Erstellen Sie eine Visualisierung der weltweiten Lebenserwartung (y-Achse) über die Jahre (x-Achse) hinweg. Halten Sie sich an die Regeln guter Visualisierung. Achten Sie auf Achsenbeschriftungen.
- Hinweis: Zur Bearbeitung dieser Aufgabe werden Sie im Netz recherchieren müssen.

```
[3]: import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Durchschnittliche Lebenserwartung (weltweit)")
plt.xlabel("Jahr")
plt.ylabel("Lebenserwartung [Jahren]")

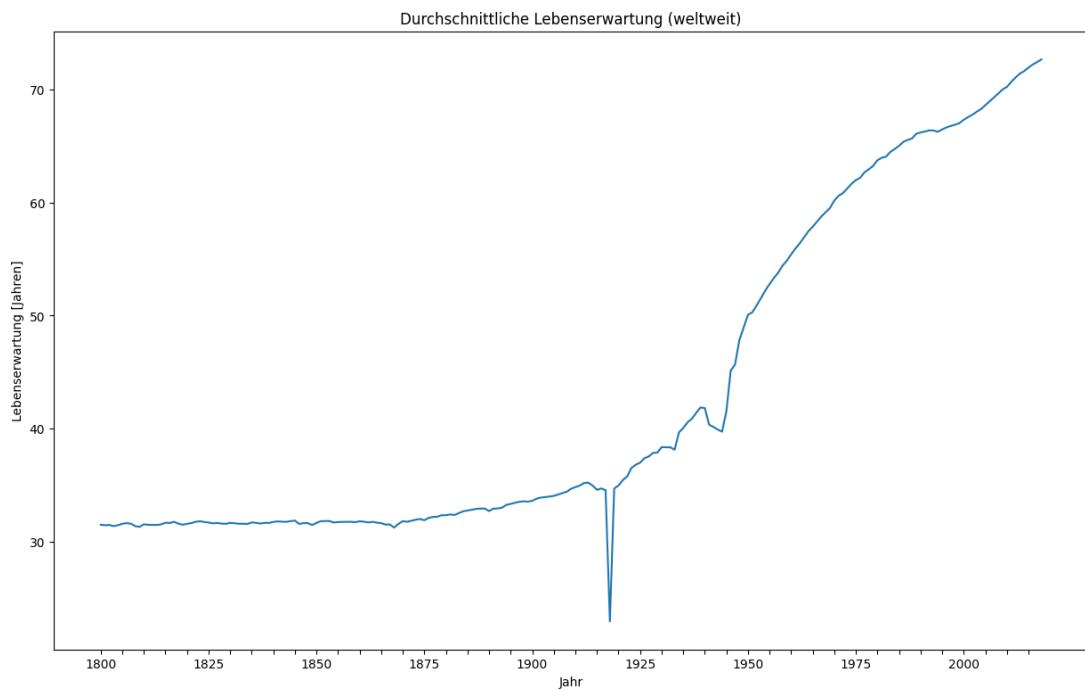
plt.tick_params(axis='x', which='both')
plt.gca().xaxis.set_major_locator(plt.MultipleLocator(10))
life_expectancy.mean().plot()
plt.show()
```



```
[4]: import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Durchschnittliche Lebenserwartung (weltweit)")
plt.xlabel("Jahr")
plt.ylabel("Lebenserwartung [Jahren]")

y = life_expectancy.mean().values
plt.plot(y)
plt.xticks(np.arange(0, 217, 5), [i if i % 25 == 0 else "" for i in np.
    arange(1800, 2017, 5)])
plt.show()
```



[5]: `life_expectancy.mean()`

```
[5]: 1800    31.502717
1801    31.461957
1802    31.478804
1803    31.383152
1804    31.459239
...
2014    71.622995
2015    71.933690
2016    72.206952
2017    72.422283
2018    72.658152
Length: 219, dtype: float64
```

- (4) Vergleichen Sie Ihre Abbildung mit Ihrer Erwartungshaltung aus Schritt (1). Wurden Sie überrascht? Falls ja, inwiefern? (1-3 Sätze)

Lag nur um 4 Jahre daneben.

- (5) Führen Sie eine kurze Recherche durch: Welche Ereignisse könnten die Einbrüche in der weltweiten Lebenserwartung, die Sie zu bestimmten Zeiten beobachteten, erklären?

Angenommen hätte ich, dass die größten Einbrüche zum Zeitpunkt der beiden Weltkriege stattgefunden hätten, jedoch ist der größte Einbruch der durchschnittlichen Lebenserwartung im Jahre 1918 auf Grund der spanischen Grippe passiert.

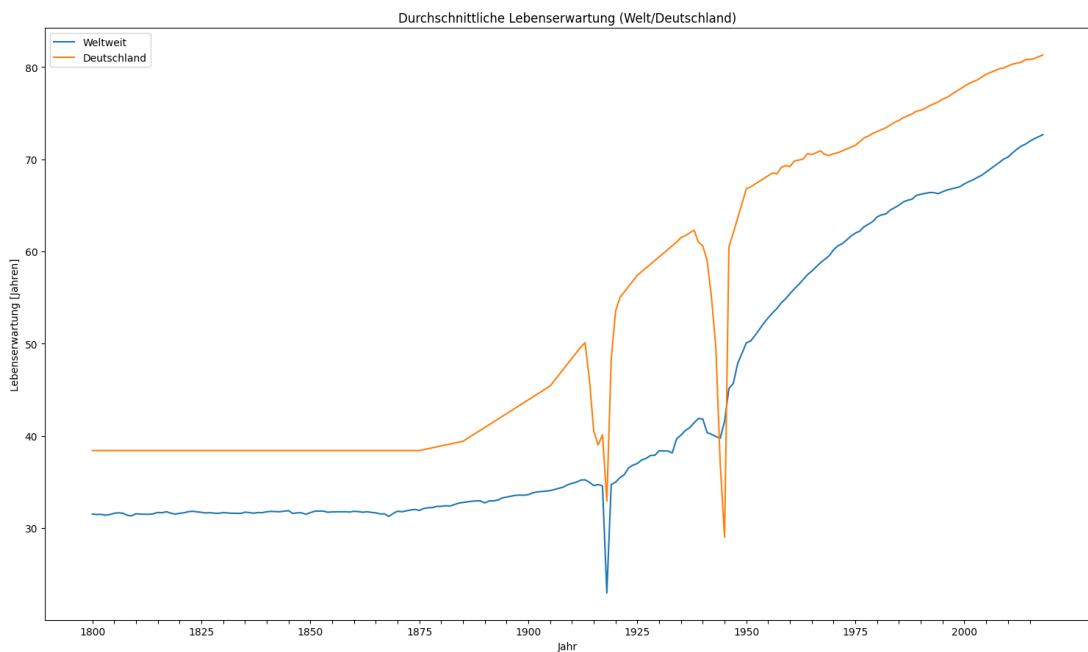
Relevante Ereignisse:

- 1918: Die Spanische Grippe, die laut den Archiven der Weltgesundheitsorganisation (WHO) schätzungsweise 20 bis 40 Millionen Menschen das Leben kostete (Quelle: [Weltgesundheitsorganisation \(WHO\)](#)).
- 1914-1918: Der Erste Weltkrieg, der mehr als 16,5 Millionen Menschen das Leben kostete (Quelle: [History.com](#))
- 1939-1945: Der Zweite Weltkrieg, der laut Britannica zwischen 35 und 60 Millionen Menschen das Leben kostete (Quelle: [Britannica](#)).
- 1943: Die Bengalische Hungersnot, die drei Millionen Todesopfer forderte. (Quelle: [NCBI](#))

- (6) [Optional]: Vergleichen Sie die weltweite Lebenserwartung mit der von Deutschland: Erstellen Sie eine Visualisierung, die beide Graphen zeigt, und interpretieren Sie diese. (1-3 Sätze).

```
[6]: fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Durchschnittliche Lebenserwartung (Welt/Deutschland)")
plt.xlabel("Jahr")
plt.ylabel("Lebenserwartung [Jahren]")

y_1 = life_expectancy.mean().values
y_2 = life_expectancy.loc[life_expectancy.index == "Germany", "1800":].values.
    ↪transpose()
plt.plot(y_1, label="Weltweit")
plt.plot(y_2, label="Deutschland")
plt.xticks(np.arange(0, 220, 5), [i if i % 25 == 0 else "" for i in np.
    ↪arange(1800, 2017, 5)])
plt.legend()
plt.tight_layout()
plt.show()
```



Die durchschnittliche Lebenserwartung lag in Deutschland lange ca. acht Jahre über dem weltweiten Durchschnitt. Zwischen 1940 und 1945 ist jedoch ein tiefer Einschnitt zu entdecken, der mit dem 2. Weltkrieg und dem NS Regime zusammenhängen dürfte.

1.0.3 4.2 Visualisierung Teil 2 (Geburtenraten)

Dies ist eine Fortsetzung von vorherigen Übungsaufgaben. Wir kommen noch einmal auf Daten der Gapminder Stiftung zurück, um Geburtenraten weltweit zu untersuchen.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Randbemerkung

Im Rahmen dieser Aufgabe wollen wir unter dem Begriff “Geburtenrate” die Gesamtfertilitätsrate (*total fertility rate*) verstehen, wie Sie sie direkt im Datensatz angegeben finden. Dies ist die durchschnittliche Anzahl von Kindern, die eine Frau im Laufe ihres Lebens bekommt. Die Forschung unterscheidet allerdings zwischen verschiedenen Fertilitätsindikatoren (z.B. zwischen so genannten Geburtenziffern, Fertilitätsraten und Reproduktionsraten). Falls Sie also einmal mit einem Bevölkerungswissenschaftler sprechen sollten: Sie untersuchen hier im Rahmen dieser Aufgabe die Gesamtfertilitätsrate.

Ihre Aufgaben

- (1) Importieren Sie den oben aufgeführten Datensatz.

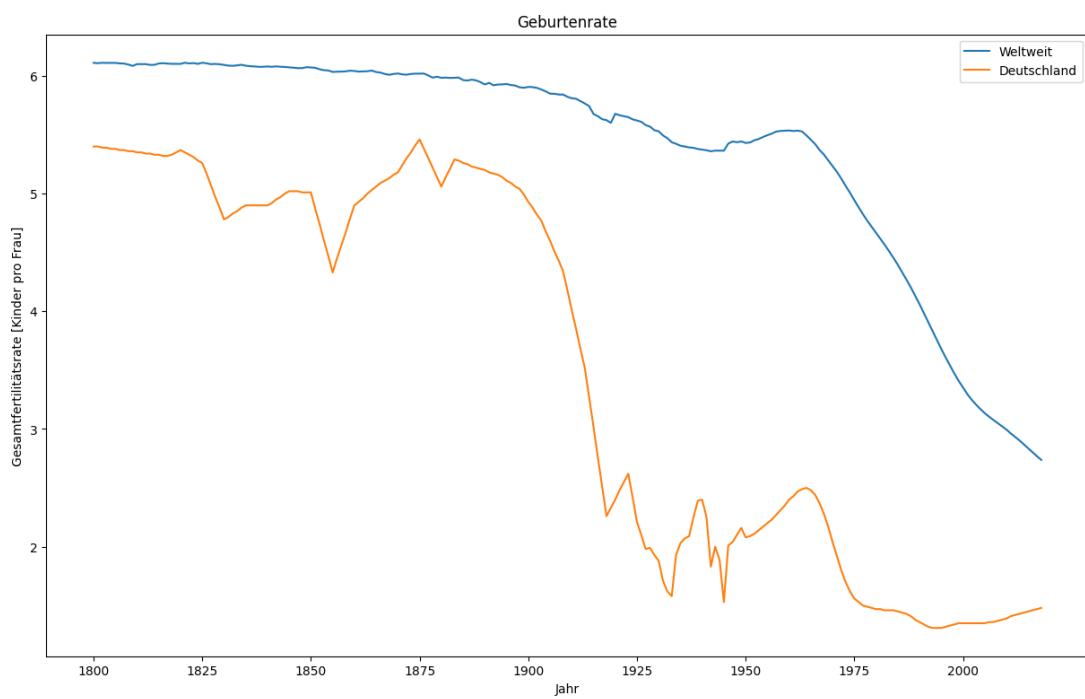
```
[7]: #!wget 'https://data.bialonski.de/ds/children_per_woman_total_fertility.csv'
fertility = pd.read_csv('children_per_woman_total_fertility.csv')
fertility.head()
```

```
[7]:      country  1800  1801  1802  1803  1804  1805  1806  1807  1808  \
0      Afghanistan  7.00  7.00  7.00  7.00  7.00  7.00  7.00  7.00  7.00
1          Albania  4.60  4.60  4.60  4.60  4.60  4.60  4.60  4.60  4.60
2          Algeria  6.99  6.99  6.99  6.99  6.99  6.99  6.99  6.99  6.99
3          Angola  6.93  6.93  6.93  6.93  6.93  6.93  6.93  6.94  6.94
4 Antigua and Barbuda  5.00  5.00  4.99  4.99  4.99  4.98  4.98  4.97  4.97
...        2009   2010   2011   2012   2013   2014   2015   2016   2017   2018
0 ...    6.04   5.82   5.60   5.38   5.17   4.98   4.80   4.64   4.48   4.33
1 ...    1.65   1.65   1.67   1.69   1.70   1.71   1.71   1.71   1.71   1.71
2 ...    2.83   2.89   2.93   2.94   2.92   2.89   2.84   2.78   2.71   2.64
3 ...    6.24   6.16   6.08   6.00   5.92   5.84   5.77   5.69   5.62   5.55
4 ...    2.15   2.13   2.12   2.10   2.09   2.08   2.06   2.05   2.04   2.03
```

[5 rows x 220 columns]

- (2) Visualisieren Sie die weltweite Geburtenrate (y-Achse) als Funktion der Zeit (Jahre, x-Achse). Achten Sie auf die Regeln guter Visualisierung, wie Sie sie in der Vorlesung kennengelernt haben. Beschriften Sie alle Achsen.

```
[8]: fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Geburtenrate")
y_welt: pd.DataFrame = fertility.loc[:, "1800":].mean()
y_de: pd.DataFrame = fertility.loc[fertility["country"] == "Germany", "1800":].mean()
plt.plot(y_welt.values, label="Weltweit")
plt.plot(y_de.values, label="Deutschland")
plt.xlabel("Jahr")
plt.ylabel("Gesamtfertilitätsrate [Kinder pro Frau]")
plt.legend()
plt.xticks(np.arange(0, 218, 25), fertility.columns[1::25])
plt.show()
```



- (3) Ab wann geht die Geburtenrate weltweit besonders stark zurück? (ungefähre Jahreszahl). Welche Hypothesen haben Sie, warum dieser Rückgang zu beobachten ist?

Ab 1963 ist ein stetiger Rückgang zu beobachten. In Europa könnte dies auf das Ende des Baby Booms nach dem 2. Weltkrieg zurückzuführen sein. Global dürften aber die sinkende Kindersterblichkeit die größte Rolle gespielt haben.

- (4) Ermitteln Sie für jedes Jahr das Land, in dem die Geburtenrate am meisten zurückging. (Ein *Pandas Series* Objekt).

```
[9]: fertility_change: pd.DataFrame = fertility.set_index("country") - fertility.set_index("country").shift(1, axis=1)
```

```
# äquivalent zu
# fertility_change: pd.DataFrame = fertility.set_index("country").
    ↪diff(periods=1, axis=1)
```

fertility_change

```
[9]:
```

	1800	1801	1802	1803	1804	1805	1806	1807	1808	\
country										
Afghanistan	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Albania	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Algeria	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Angola	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.01	0.00	
Antigua and Barbuda	NaN	0.0	-0.01	0.0	0.0	-0.01	0.0	-0.01	0.00	
...	
Venezuela	NaN	0.0	0.01	0.0	0.0	0.01	0.0	0.00	0.01	
Vietnam	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Yemen	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Zambia	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
Zimbabwe	NaN	0.0	0.00	0.0	0.0	0.00	0.0	0.00	0.00	
	1809	...	2009	2010	2011	2012	2013	2014	2015	\
country										
Afghanistan	0.0	...	-0.21	-0.22	-0.22	-0.22	-0.21	-0.19	-0.18	
Albania	0.0	...	0.00	0.00	0.02	0.02	0.01	0.01	0.00	
Algeria	0.0	...	0.08	0.06	0.04	0.01	-0.02	-0.03	-0.05	
Angola	0.0	...	-0.07	-0.08	-0.08	-0.08	-0.08	-0.08	-0.07	
Antigua and Barbuda	0.0	...	-0.01	-0.02	-0.01	-0.02	-0.01	-0.01	-0.02	
...	
Venezuela	0.0	...	-0.03	-0.03	-0.03	-0.02	-0.03	-0.02	-0.03	
Vietnam	0.0	...	0.02	0.01	0.00	0.01	0.00	0.00	0.00	
Yemen	0.0	...	-0.14	-0.13	-0.12	-0.11	-0.11	-0.11	-0.12	
Zambia	0.0	...	-0.08	-0.08	-0.08	-0.08	-0.07	-0.07	-0.06	
Zimbabwe	0.0	...	0.01	0.01	-0.01	-0.02	-0.04	-0.06	-0.06	
	2016	2017	2018							
country										
Afghanistan	-0.16	-0.16	-0.15							
Albania	0.00	0.00	0.00							
Algeria	-0.06	-0.07	-0.07							
Angola	-0.08	-0.07	-0.07							
Antigua and Barbuda	-0.01	-0.01	-0.01							
...							
Venezuela	-0.02	-0.03	-0.02							
Vietnam	-0.01	0.00	0.00							
Yemen	-0.10	-0.11	-0.10							
Zambia	-0.06	-0.05	-0.06							
Zimbabwe	-0.08	-0.08	-0.07							

[184 rows x 219 columns]

```
[10]: for year in fertility_change.columns[1:]:
    print(f'{year} {fertility_change[year].idxmin():<25} ↴
          {fertility_change[year].min():1.4f}"
```

1801	United Kingdom	-0.3700
1802	Norway	-0.1600
1803	Finland	-0.4500
1804	Norway	-0.2600
1805	United Kingdom	-0.1000
1806	Finland	-0.3700
1807	Norway	-0.1200
1808	Finland	-0.8100
1809	Norway	-0.7900
1810	Philippines	-0.0700
1811	Finland	-0.4400
1812	Sweden	-0.2500
1813	Sweden	-0.5400
1814	Norway	-0.2300
1815	Barbados	-0.0300
1816	United Kingdom	-0.2900
1817	Norway	-0.3800
1818	Norway	-0.2400
1819	Finland	-0.2700
1820	Poland	-0.2700
1821	Germany	-0.0200
1822	Finland	-0.7500
1823	Poland	-0.2700
1824	Finland	-0.3700
1825	United Kingdom	-0.0400
1826	Poland	-0.6700
1827	Sweden	-0.5000
1828	Germany	-0.1000
1829	United Kingdom	-0.3800
1830	Sweden	-0.2700
1831	Sweden	-0.3500
1832	Norway	-0.1600
1833	Finland	-0.5100
1834	Philippines	-0.1400
1835	Poland	-0.9300
1836	Norway	-0.4700
1837	Sweden	-0.1500
1838	Poland	-0.5400
1839	Norway	-0.5200
1840	Philippines	-0.0600
1841	Poland	-0.4100

1842	Belgium	-0.0600
1843	Finland	-0.1900
1844	Poland	-0.2700
1845	Poland	-0.6700
1846	Finland	-0.3700
1847	Poland	-0.4000
1848	Poland	-0.2700
1849	Belgium	-0.0600
1850	Sweden	-0.2100
1851	Greece	-0.2200
1852	Finland	-0.3800
1853	Chile	-0.2500
1854	Poland	-0.2700
1855	Poland	-0.5400
1856	Greece	-0.2200
1857	Finland	-0.3800
1858	Chile	-0.4300
1859	Greece	-0.2200
1860	Slovenia	-0.5200
1861	Chile	-1.0700
1862	New Zealand	-0.3200
1863	Poland	-0.4000
1864	Hungary	-0.3800
1865	Chile	-0.8500
1866	Romania	-0.4500
1867	Hungary	-0.3800
1868	Finland	-1.0700
1869	Australia	-0.2400
1870	Switzerland	-0.1500
1871	Poland	-0.4000
1872	Hungary	-0.2700
1873	Japan	-0.1500
1874	Slovenia	-0.1300
1875	Luxembourg	-0.2900
1876	Chile	-0.2700
1877	Hungary	-0.3200
1878	Romania	-0.5200
1879	Greece	-0.3200
1880	Hungary	-0.3500
1881	New Zealand	-0.3700
1882	Chile	-0.5800
1883	Luxembourg	-0.1600
1884	Uruguay	-0.5000
1885	Chile	-1.9400
1886	Bulgaria	-0.6700
1887	Croatia	-0.1700
1888	Philippines	-0.4700
1889	Sri Lanka	-0.8600

1890 Hungary	-0.4200
1891 Chile	-0.4400
1892 Russia	-0.8700
1893 Uruguay	-0.2700
1894 Portugal	-0.2900
1895 Chile	-0.4400
1896 Argentina	-0.5200
1897 Uruguay	-0.7700
1898 Romania	-0.8300
1899 Sri Lanka	-1.1000
1900 Romania	-0.4300
1901 Sri Lanka	-0.3100
1902 Guyana	-0.2900
1903 Poland	-0.2700
1904 Philippines	-1.2700
1905 Sri Lanka	-0.9000
1906 Philippines	-0.6200
1907 Sri Lanka	-0.4400
1908 Malta	-0.1300
1909 Philippines	-0.6300
1910 Switzerland	-0.3200
1911 Slovenia	-0.2400
1912 Sri Lanka	-0.7100
1913 Slovak Republic	-0.4300
1914 Slovak Republic	-0.4300
1915 Russia	-3.5200
1916 Slovenia	-0.7800
1917 Romania	-0.5900
1918 Romania	-0.5800
1919 Russia	-2.2800
1920 Switzerland	-0.1500
1921 Russia	-2.0000
1922 Slovak Republic	-0.2900
1923 Greece	-0.3300
1924 Slovak Republic	-0.3000
1925 Norway	-0.2400
1926 Mauritius	-0.3000
1927 Portugal	-0.3300
1928 Mauritius	-0.3000
1929 Sri Lanka	-0.5500
1930 Russia	-0.4000
1931 Greece	-0.3600
1932 Russia	-0.5600
1933 Russia	-1.0400
1934 Russia	-0.4400
1935 Sri Lanka	-0.4300
1936 Japan	-0.2500
1937 Ireland	-0.3500

1938	Japan	-0.5800
1939	Albania	-0.8600
1940	Russia	-0.6800
1941	Armenia	-0.9600
1942	Russia	-1.6400
1943	Armenia	-1.5200
1944	North Korea	-0.5800
1945	Japan	-0.8300
1946	North Korea	-0.5800
1947	North Korea	-0.5800
1948	North Korea	-0.5800
1949	South Korea	-0.8100
1950	South Korea	-0.8200
1951	Guinea-Bissau	-0.3100
1952	Iraq	-0.4400
1953	Iraq	-0.3700
1954	Iraq	-0.3100
1955	Iraq	-0.2400
1956	China	-0.3700
1957	Romania	-0.1300
1958	China	-0.7100
1959	China	-1.2300
1960	China	-0.2800
1961	China	-0.7100
1962	Singapore	-0.2900
1963	Singapore	-0.3000
1964	China	-1.2900
1965	Grenada	-0.3100
1966	Mauritius	-0.3200
1967	China	-0.9600
1968	Mauritius	-0.3300
1969	China	-0.7000
1970	Mauritius	-0.2600
1971	China	-0.3500
1972	China	-0.4800
1973	China	-0.4100
1974	China	-0.3600
1975	China	-0.5700
1976	China	-0.3300
1977	China	-0.3800
1978	Mexico	-0.2700
1979	Mexico	-0.2600
1980	China	-0.4800
1981	Libya	-0.2000
1982	Algeria	-0.1900
1983	Libya	-0.2000
1984	Kuwait	-0.2200
1985	Kuwait	-0.2500

1986	Kuwait	-0.2900
1987	Kuwait	-0.2900
1988	Kuwait	-0.3000
1989	Iran	-0.3200
1990	Iran	-0.3300
1991	Iran	-0.3400
1992	Oman	-0.3600
1993	Oman	-0.3700
1994	Oman	-0.3800
1995	Armenia	-0.3800
1996	Oman	-0.3700
1997	Oman	-0.3500
1998	Oman	-0.3300
1999	Oman	-0.3100
2000	Oman	-0.2700
2001	Oman	-0.2400
2002	Oman	-0.1900
2003	Ethiopia	-0.1800
2004	Yemen	-0.1800
2005	Ethiopia	-0.1800
2006	Afghanistan	-0.1800
2007	Afghanistan	-0.1900
2008	Afghanistan	-0.2100
2009	Afghanistan	-0.2100
2010	Afghanistan	-0.2200
2011	Afghanistan	-0.2200
2012	Afghanistan	-0.2200
2013	Afghanistan	-0.2100
2014	Afghanistan	-0.1900
2015	Afghanistan	-0.1800
2016	Afghanistan	-0.1600
2017	Afghanistan	-0.1600
2018	Afghanistan	-0.1500

(5) Ermitteln Sie mit Ihrem Ergebnis aus Schritt (4) das Land, dessen Geburtenrate Ende der 80er und Anfang der 90er Jahre am stärksten zurückging. Um welches Land handelt es sich?

```
[11]: fertility_change.loc[:, "1988":"1991"].idxmin(axis=0)
```

```
[11]: 1988    Kuwait
      1989    Iran
      1990    Iran
      1991    Iran
      dtype: object
```

Im Zeitraum von 1989 bis 1991 war der größte jährliche Geburtenrückgang im Iran. Eine Veränderung um -0.99 Kinder pro Frau in nur drei Jahren.

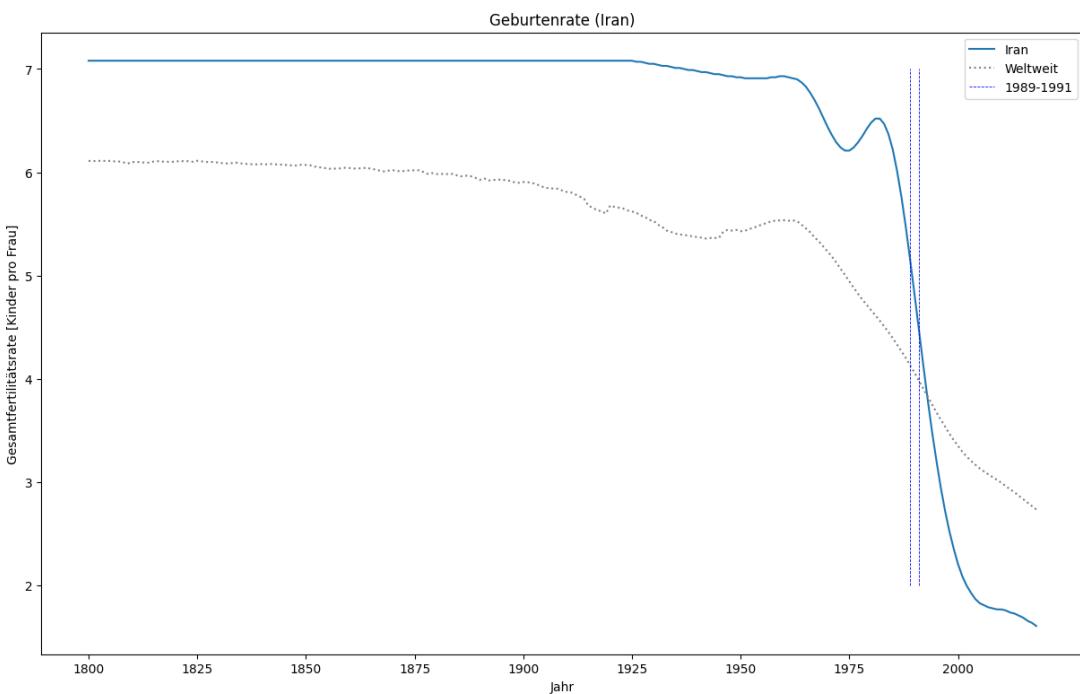
(6) Visualisieren Sie den Zeitverlauf der Geburtenraten für das Land, welches Sie in Schritt (5)

identifiziert haben. Achten Sie auf die Regeln guter Visualisierung, die Sie in der Vorlesung kennengelernt haben. Was Sie beobachten, zählt zu den schnellsten Geburtenrückgängen der Weltgeschichte.

```
[12]: fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Geburtenrate (Iran)")
y_welt: pd.DataFrame = fertility.loc[:, "1800":].mean()
y_sg: pd.DataFrame = fertility.loc[fertility["country"] == "Singapore", "1800":].mean()
y_tl: pd.DataFrame = fertility.loc[fertility["country"] == "Timor-Leste", "1800":].mean()
y_ir: pd.DataFrame = fertility.loc[fertility["country"] == "Iran", "1800":].mean()

plt.plot(y_ir.values, label="Iran")
plt.plot(y_welt.values, linestyle=":", color="grey", label="Weltweit")
plt.xlabel("Jahr")
plt.ylabel("Gesamtfertilitätsrate [Kinder pro Frau]")
plt.vlines(x=[189, 191], ymin=2, ymax=7, colors='blue', linestyles='--', linewidth=0.6, label="1989-1991")

plt.legend()
plt.xticks(np.arange(0, 218, 25), fertility.columns[1::25])
plt.show()
```

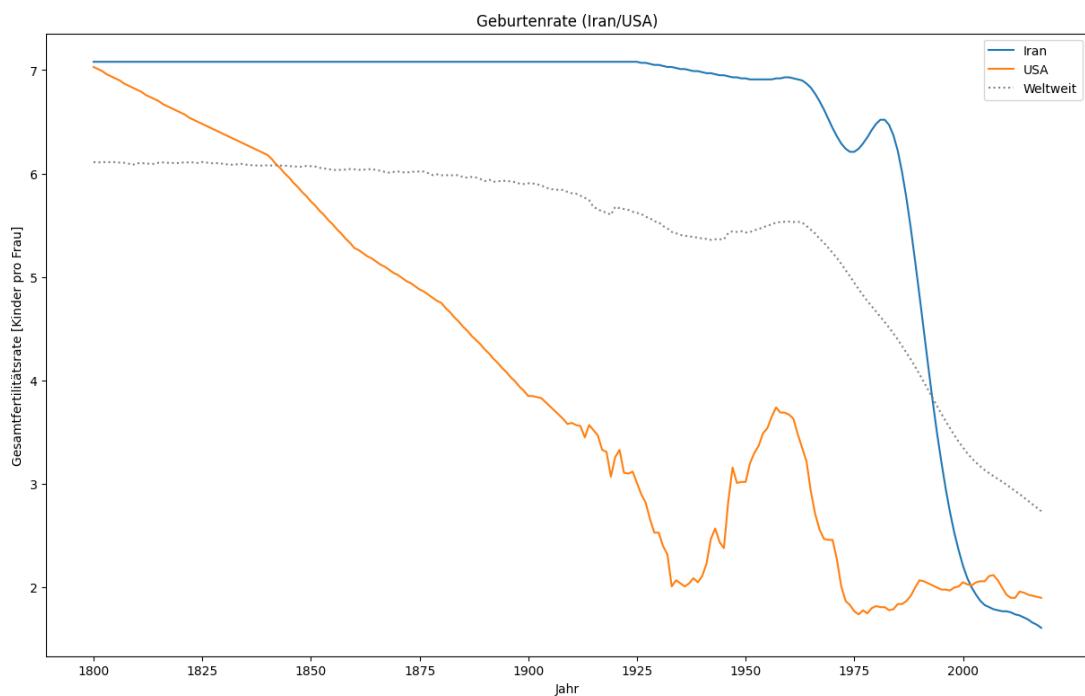


- (7) Fügen Sie der Abbildung aus Schritt (6) noch die Geburtenrate der USA als zusätzlichen Graphen hinzu. Beschreiben Sie grob den qualitativen Unterschied, den Sie feststellen (1-3 Sätze).

```
[13]: fig, ax = plt.subplots(figsize=(15, 9))
plt.title("Geburtenrate (Iran/USA)")
y_welt: pd.DataFrame = fertility.loc[:, "1800":].mean()
y_us: pd.DataFrame = fertility.loc[fertility["country"] == "United States", "1800":].mean()
y_ir: pd.DataFrame = fertility.loc[fertility["country"] == "Iran", "1800":].mean()

plt.plot(y_ir.values, label="Iran")
plt.plot(y_us.values, label="USA")
plt.plot(y_welt.values, linestyle=":", color="grey", label="Weltweit")
plt.xlabel("Jahr")
plt.ylabel("Gesamtfertilitätsrate [Kinder pro Frau]")

plt.legend()
plt.xticks(np.arange(0, 218, 25), fertility.columns[1::25])
plt.show()
```



Die Geburtenraten in den USA unterliegen ab 1910 schärferen Änderungen, während die Kurve vom Iran sehr glatt ist. Eine mögliche Erklärung könnte in der Erhebungsfrequenz der Daten liegen – vielleicht wurden fehlende Daten für den Iran interpoliert.

Beschreibung der Geburtenrateentwicklung in Iran und in den Vereinigten Staaten von 1800 bis 2000: - Wir sehen, dass im Jahre 1800 sowohl Iran als auch die Vereinigten Staaten eine ähnliche Geburtenrate hatten, nämlich 7 Kinder pro Frau im Jahr 1800, die bis zum Jahr 2000 auf ungefähr 2 Kinder pro Frau sank. - Der Hauptunterschied besteht darin, wie schnell sich diese Rate entwickelt hat. In den Vereinigten Staaten gab es eine allmähliche lineare Senkung der Geburtenrate, von etwa 1 Kind pro Frau alle 50 Jahre zwischen 1800 und 1900. Während dieses Zeitraums blieb die Geburtenrate im Iran stabil bei 7 Kindern pro Frau. In den 1950er Jahren stieg die Geburtenrate in den Vereinigten Staaten von 2 auf fast 4 Kinder pro Frau, während sie im Iran von 7 auf 6 Kinder senkte. - Aber plötzlich in den 1980er Jahren gab es einen dramatischen Rückgang der Geburtenrate im Iran, von 7 Kindern pro Frau im Jahr 1980 auf 3 Kinder pro Frau im Jahr 2000 also ein Rückgang von 4 Kinder pro Frau in weniger als 30 Jahren. Im Gegensatz dazu gab es in den Vereinigten Staaten einen langsameren Rückgang der Geburtenrate von 4 Kindern pro Frau in den 1950er Jahren auf 2 Kinder pro Frau im Jahr 1990.

Schlussbemerkung *Der Iran verbesserte Ende des 20. Jahrhunderts die medizinische Versorgung und die Bildung der Bürger deutlich. Eine obligatorische Sexualaufklärung, einfacher Zugang zu Verhütungsmitteln (unter anderem zur Pille) sowie eine im Iran befindliche Kondomfabrik (die in den 90er Jahren größte Kondomfabrik weltweit) werden zu den verschiedenen Faktoren gezählt, die den Geburtenrückgang erklären.*

1.0.4 4.3 Multivariate explorative Analyse (Mietspiegel)

In vielen Städten und Gemeinden werden sogenannte Mietspiegel erstellt, die eine Marktübersicht zu Miethöhen bereitstellen. Diese werden gerne von Sachverständigen, Vermietern und Mietinteressenten zurate gezogen. In §558, Absatz (2) des bürgerlichen Gesetzbuchs ist die ortsübliche Vergleichsmiete definiert:

„Die ortsübliche Vergleichsmiete wird gebildet aus den üblichen Entgelten, die in der Gemeinde oder einer vergleichbaren Gemeinde für Wohnraum vergleichbarer Art, Größe, Ausstattung, Beschaffenheit und Lage einschließlich der energetischen Ausstattung und Beschaffenheit in den letzten vier Jahren vereinbart oder, von Erhöhungen nach §560 abgesehen, geändert worden sind.“

Das bedeutet für die Nettomiete, dass ihr Durchschnittswert in Abhängigkeit von Merkmalen wie Art, Größe, Ausstattung, Beschaffenheit und Lage der Wohnung zu bestimmen bzw. zu schätzen ist.

Im Rahmen dieser Übung werden Sie einen Mietspiegel aus München untersuchen, einer Gegend, die bekannt für außerordentlich hohe Mieten ist.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

- (1) Importieren Sie die Daten.

```
[14]: #!wget -nc 'https://data.bialonski.de/ds/mietspiegel2015.txt'
mietspiegel: pd.DataFrame = pd.read_csv('mietspiegel2015.txt', sep=' ')
```

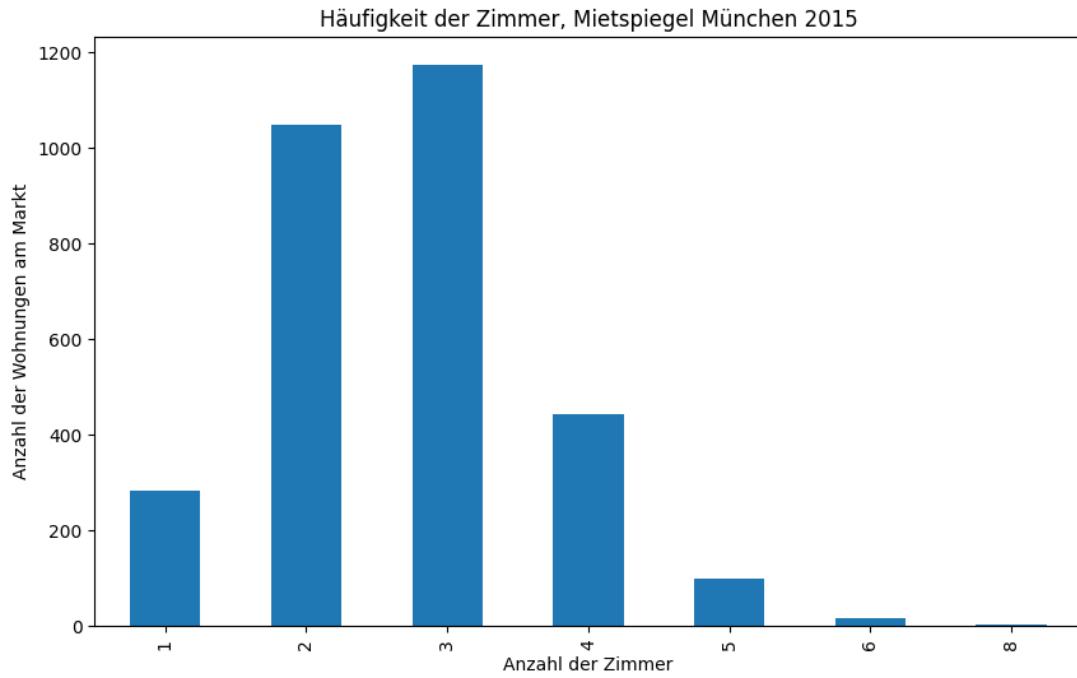
```
[14]:
```

	nm	nmqm	wfl	rooms	bj	bez	wohngut	\
0	608.40	12.67	48	2	1957.5	Untergiesing	0	
1	780.00	13.00	60	2	1983.0	Bogenhausen	1	
2	822.60	7.48	110	5	1957.5	Obergiesing	0	
3	500.00	8.62	58	2	1957.5	Schwanthalerhöhe	0	
4	595.00	8.50	70	3	1972.0	Aubing...	0	
...	
3060	549.00	8.19	67	3	1957.5	Fledmoching-Hasenbergel	0	
3061	873.50	11.80	74	3	1972.0	Bogenhausen	0	
3062	1130.00	13.95	81	3	1972.0	Bogenhausen	0	
3063	440.00	16.30	27	1	1972.0	Schwabing West	1	
3064	625.43	7.36	85	4	1957.5	Schwabing West	1	
	wohnbest	ww0	zh0	badkach0	badextra	kueche		
0	0	0	0	1	0	0		
1	0	0	0	1	0	1		
2	0	0	1	1	1	0		
3	0	0	0	1	0	1		
4	0	0	0	0	0	0		
...		
3060	0	0	0	1	0	0		
3061	1	0	0	1	1	0		
3062	1	0	0	1	1	0		
3063	0	0	0	1	0	1		
3064	0	0	0	1	0	0		

[3065 rows x 13 columns]

(2) Erzeugen Sie ein Säulendiagramm, dass die Häufigkeiten der Zimmeranzahl (rooms) zeigt.

```
[15]: fig, ax = plt.subplots(figsize=(10, 6))
plt.title("Häufigkeit der Zimmer, Mietspiegel München 2015")
mietspiegel['rooms'].value_counts().sort_index().plot(kind="bar")
plt.xlabel("Anzahl der Zimmer")
plt.ylabel("Anzahl der Wohnungen am Markt")
plt.show()
```



- (3) Erstellen Sie eine 5-Number-Summary (Fünf-Punkte-Zusammenfassung) für die Variable Baujahr (bj).

```
[16]: mietspiegel['bj'].describe().loc[['min', '25%', '50%', '75%', 'max']]
```

```
[16]: min    1918.0
      25%    1957.5
      50%    1957.5
      75%    1983.0
      max    2012.5
Name: bj, dtype: float64
```

- (4) Berechnen Sie die 5-Number-Summary für die Variablen Nettomiete (nm) und Nettomiete/Quadratmeter (nmqm), jeweils getrennt für Wohnungen, die vor bzw. nach dem Jahr 1958 gebaut wurden. Erstellen Sie ebenfalls die dazugehörigen Boxplots. Interpretieren Sie Ihre Ergebnisse. (1-3 Sätze)

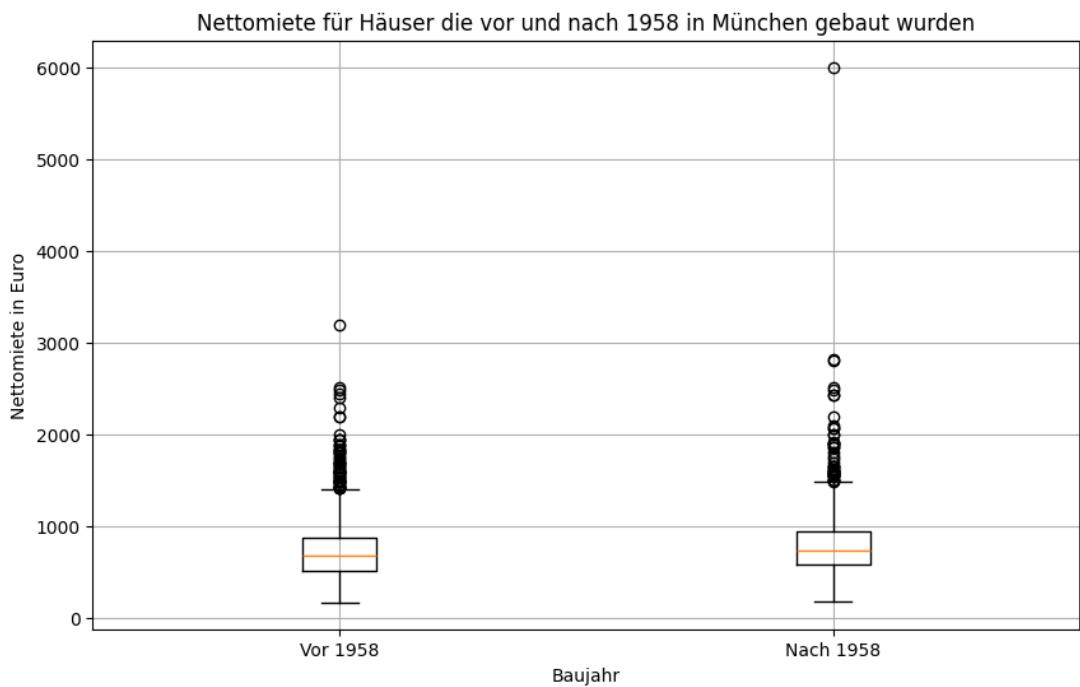
```
[17]: print(mietspiegel.groupby(mietspiegel['bj'] >= 1958).describe().loc[:, ['nm', ↵'nmqm']].loc(axis=1)[pd.IndexSlice[:, 'min':'max']])
```

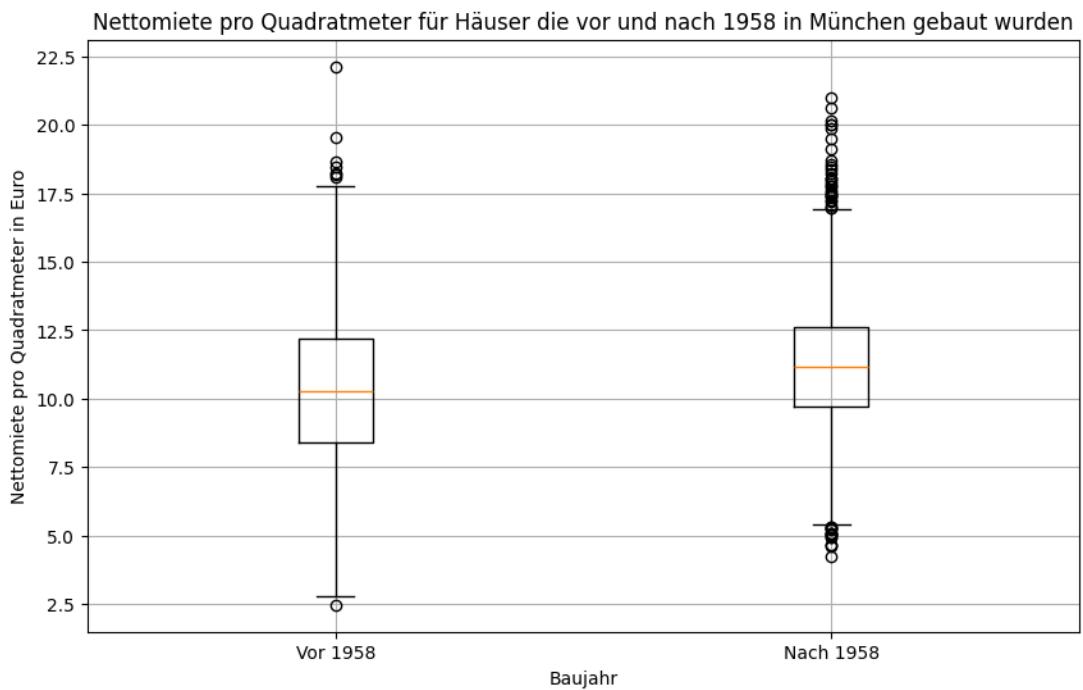
bj	nm				nmqm					
	min	25%	50%	75%	max	min	25%	50%	75%	
False	174.75	511.0	677.36	872.0	3200.0	2.47	8.380	10.28	12.19	22.13
True	187.63	590.0	735.00	950.0	6000.0	4.24	9.735	11.17	12.62	21.01

```
[18]: # Subsetting
miete_baujahr_vor_1958 = mietspiegel[mietspiegel['bj'] < 1958][['nm', 'nmqm']]
miete_baujahr_nach_1958 = mietspiegel[mietspiegel['bj'] >= 1958][['nm', 'nmqm']]

# Plotting
plt.figure(figsize=(10, 6))
plt.boxplot([miete_baujahr_vor_1958['nm'], miete_baujahr_nach_1958['nm']], 
            tick_labels=['Vor 1958', 'Nach 1958'])
plt.title('Nettomiete für Häuser die vor und nach 1958 in München gebaut wurden')
plt.ylabel('Nettomiete in Euro')
plt.xlabel('Baujahr')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.boxplot([miete_baujahr_vor_1958['nmqm'], miete_baujahr_nach_1958['nmqm']], 
            tick_labels=['Vor 1958', 'Nach 1958'])
plt.title('Nettomiete pro Quadratmeter für Häuser die vor und nach 1958 in München gebaut wurden')
plt.ylabel('Nettomiete pro Quadratmeter in Euro')
plt.xlabel('Baujahr')
plt.grid(True)
plt.show()
```





Die Nettomiete und Nettomiete/Quadratmeter sind bei Wohnungen nach 1958 tendenziell höher als bei Wohnungen vor 1958. Die maximale Nettomiete hat sich sogar fast verdoppelt, allerdings ist die maximale Nettomiete pro Quadratmeter gesunken.

- (5) Führen Sie die Analyse aus Schritt (4) noch einmal durch, allerdings schichten Sie hier Ihre Analyse nicht mehr nach “Baujahr vor oder nach 1958” sondern nach der Anzahl der Räume (rooms). Das bedeutet, dass Sie Boxplots der Nettomiete (nm), der Nettomiete/Quadratmeter (nmqm) in Abhängigkeit der Raumanzahl erstellen. Interpretieren Sie Ihre Ergebnisse. (1-3 Sätze).

```
[19]: # Subsetting
miete_rooms_netto_miete = mietspiegel.set_index('rooms')[['nm']]
miete_rooms_netto_miete_pro_quadratmeter = mietspiegel.
    ↪set_index('rooms')[['nmqm']]

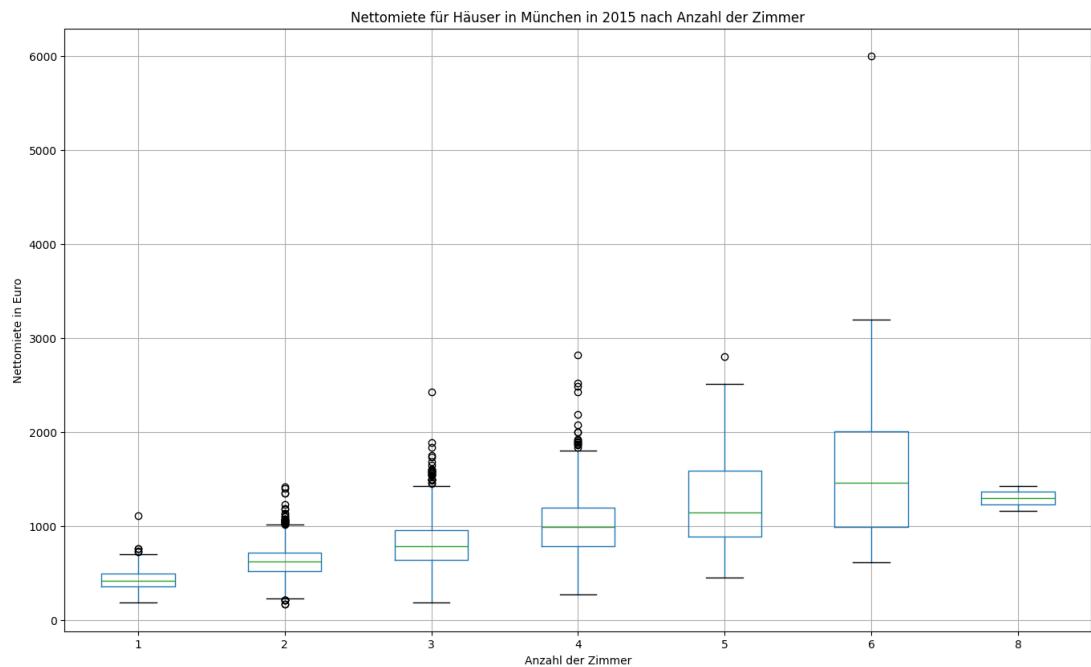
# Plotting netto_miete
miete_rooms_netto_miete.boxplot(by='rooms', figsize=(16, 10))
plt.suptitle('') # remove the automatic title
plt.title('Nettomiete für Häuser in München in 2015 nach Anzahl der Zimmer')
plt.ylabel('Nettomiete in Euro')
plt.xlabel('Anzahl der Zimmer')
plt.grid(True)
plt.show()

# Plotting netto_miete_pro_quadratmeter
```

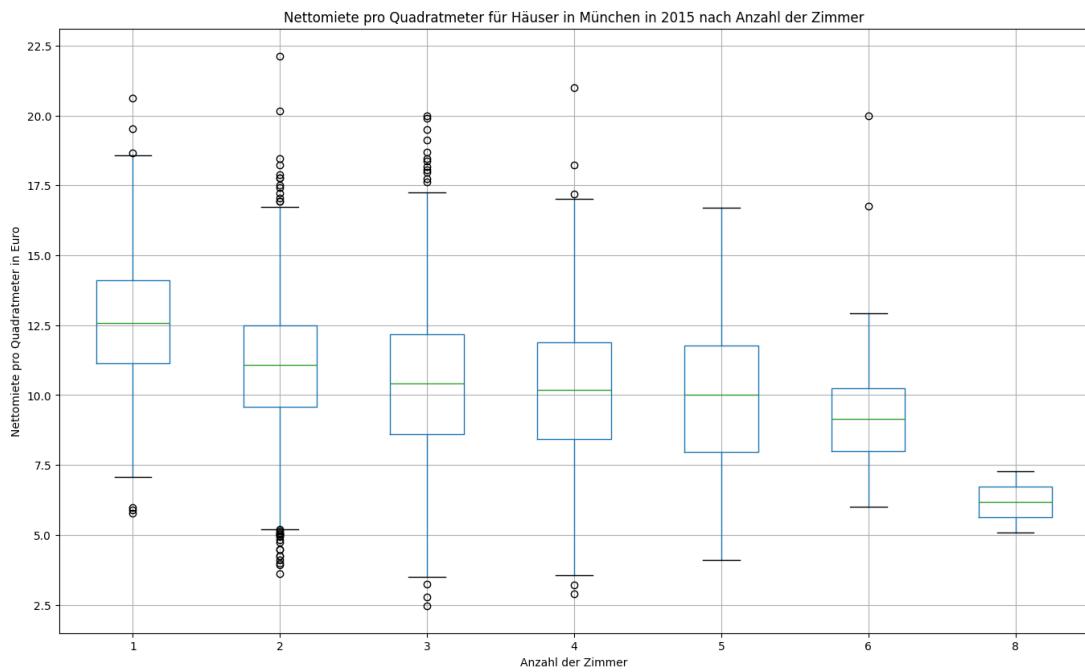
```

plt.figure(figsize=(16, 10))
miete_rooms_netto_miete_pro_quadratmeter.boxplot(by='rooms', figsize=(16, 10))
plt.suptitle('') # remove the automatic title
plt.title('Nettomiete pro Quadratmeter für Häuser in München in 2015 nach  
Anzahl der Zimmer')
plt.ylabel('Nettomiete pro Quadratmeter in Euro')
plt.xlabel('Anzahl der Zimmer')
plt.grid(True)
plt.show()

```



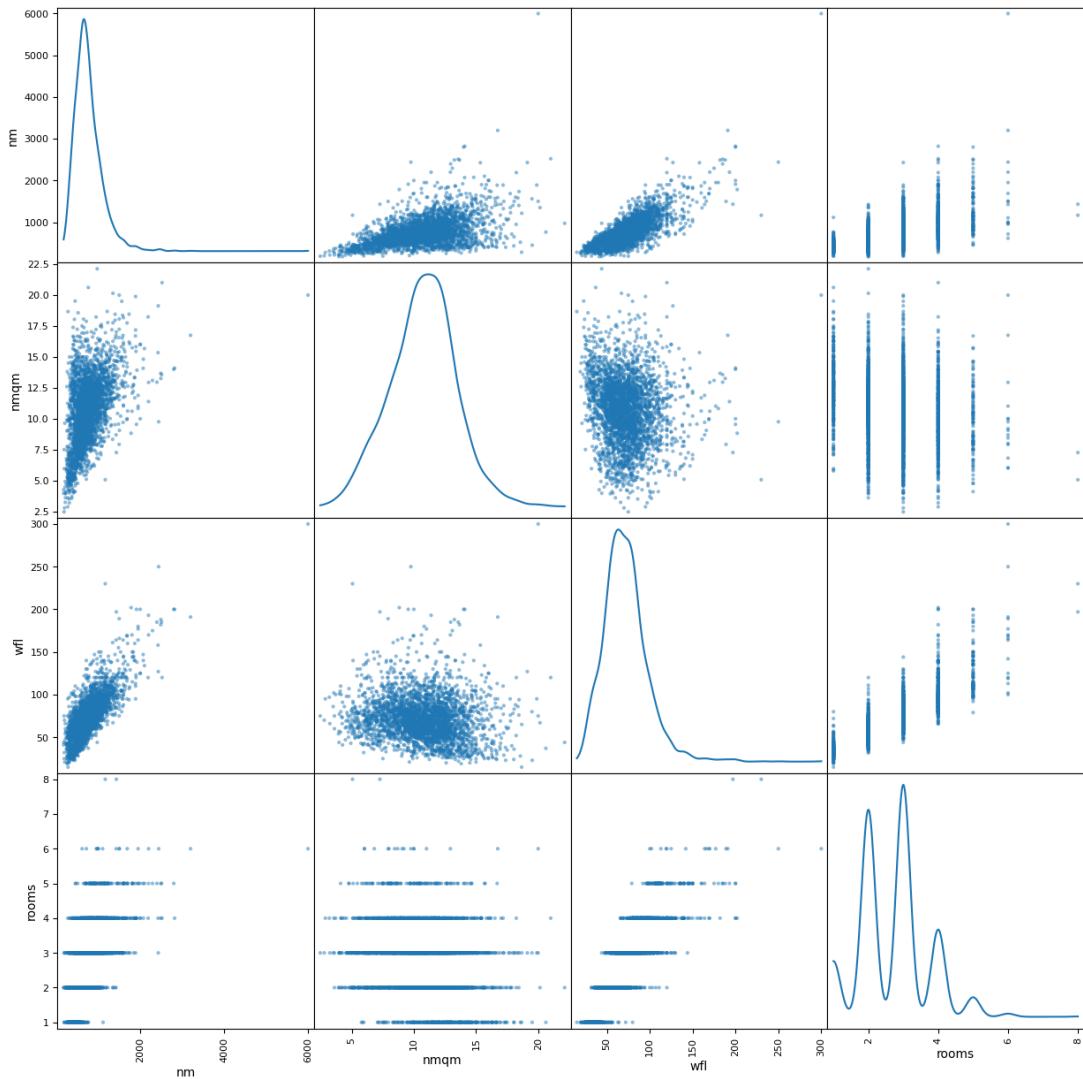
<Figure size 1600x1000 with 0 Axes>



Interpretation: - Wir sehen, dass die Nettomiete pro Zimmeranzahl fast linear zwischen 1 und 6 Zimmern ansteigt, mit Ausnahme von 8 Mietobjekten. Die Nettomiete pro Quadratmeter hingegen sinkt je nach Zimmeranzahl. Für Wohnungen/Häuser mit 1 bis 3 Zimmern sinkt sie, zwischen 3 und 5 Zimmern bleibt sie konstant und sinkt deutlich für 8 Zimmer. - Je mehr Zimmer, desto höher die absolute Miete, jedoch zahlt man pro Zimmer deutlich weniger, wenn man sich in einem Mietobjekt mit 6 Zimmern oder mehr befindet. - Daraus lässt sich interpretieren, dass es für Studenten wahrscheinlich günstiger ist, ein Zimmer in einem Haus/Wohnung mit mindestens 6 Zimmern zu mieten, aber besonders teuer ist, eine Einzimmerwohnung zu mieten.

- (6) Erkunden Sie nun etwaige Zusammenhänge zwischen den Variablen Nettomiete (nm), Nettomiete/Quadratmeter (nmqm), Wohnfläche (wfl) und Anzahl der Räume (rooms), indem Sie einen [Scatter-Matrix-Plot](#) erstellen.

```
[20]: pd.plotting.scatter_matrix(mietspiegel.loc[:, ['nm', 'nmqm', 'wfl', 'rooms']],   
                                figsize=(15, 15), diagonal='kde')  
plt.show()
```



- (7) Interpretieren Sie den Ihren Plot aus Schritt 6: Sehen Sie Zusammenhänge zwischen einzelnen Variablen? (1-3 Sätze)

Zwischen der Nettomiete und der Wohnfläche besteht ein starker Zusammenhang, während bei der Nettomiete pro Quadratmeter und der Wohnfläche kaum ein Zusammenhang zu erkennen ist. Die Anzahl der Zimmer und die Wohnfläche sind erwartungsgemäß ebenfalls stark korreliert.

- (8) Berechnen Sie Pearsons Korrelationskoeffizienten für alle Paare von Variablen aus Schritt (6). Schätzen Sie ein, welche Variablen miteinander schwach, mittel oder stark korreliert sind. Interpretieren Sie Ihre Ergebnisse (1-3 Sätze). Achten Sie dabei unter anderem auf Ihren Befund zwischen Zimmerzahl und Nettomiete/Quadratmeter sowie Nettomiete und Wohnfläche.

```
[21]: mietspiegel.loc[:, ['nm', 'nmqm', 'wfl', 'rooms']].corr(method='pearson')
```

```
[21]:      nm      nmqm      wfl      rooms
nm    1.000000  0.462052  0.783243  0.587658
nmqm   0.462052  1.000000 -0.136143 -0.216358
wfl    0.783243 -0.136143  1.000000  0.838881
rooms  0.587658 -0.216358  0.838881  1.000000
```

Zimmeranzahl und Wohnfläche sind erwartungsgemäß stark positiv korreliert. Auch die Nettomiete und die Wohnfläche sind positiv korreliert, wenn auch etwas schwächer (mittlere Korrelation). Bei der Nettomiete pro Quadratmeter und der Wohnfläche liegt kaum eine Korrelation vor.

- (9) Berechnen Sie Spearmans Korrelationskoeffizient für alle Paare von Variablen aus Schritt (6). Vergleichen Sie die Werte, die Sie erhalten, mit den Werten von Pearsons Korrelationskoeffizienten aus Schritt (8). Für welches Feature-Paar sehen Sie in beiden Koeffizienten die größten Unterschiede? Wie interpretieren Sie diesen Unterschied?
- Der **Pearsons Korrelationskoeffizient** charakterisiert der Stärke linearer Zusammenhänge.
 - Der **Spearman Korrelationskoeffizient** charakterisiert der Stärke monotoner Zusammenhänge.

```
[22]: mietspiegel.loc[:, ['nm', 'nmqm', 'wfl', 'rooms']].corr(method='spearman')
```

```
[22]:      nm      nmqm      wfl      rooms
nm    1.000000  0.437574  0.750244  0.607100
nmqm   0.437574  1.000000 -0.177474 -0.224491
wfl    0.750244 -0.177474  1.000000  0.859069
rooms  0.607100 -0.224491  0.859069  1.000000
```

Der stärkste Unterschied liegt bei dem Zusammenhang der Nettomiete pro Quadratmeter und der Wohnfläche vor.

1.0.5 4.4 Pearsons Korrelationskoeffizient und Invarianz

Betrachten Sie die beiden Merkmale X und Y sowie ihre linearen Transformationen

$$\tilde{X} = a_X X + b_X, \quad a_X \neq 0$$

und

$$\tilde{Y} = a_Y Y + b_Y, \quad a_Y \neq 0$$

Ihre Aufgaben

- (1) Schlagen Sie in der Vorlesung die Definition von Pearsons Korrelationskoeffizienten nach.

Pearsons Korrelationskoeffizient (Charakterisierung der Stärke linearer Zusammenhänge) r ist der Quotient aus der Kovarianz \tilde{s}_{XY} und der Standardabweichung der Stichproben $\tilde{s}_X \tilde{s}_Y$.

$$r = \frac{\tilde{s}_{XY}}{\tilde{s}_X \tilde{s}_Y}$$

Kovarianz:

$$\tilde{s}_{XY} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Standardabweichung der Stichprobe X :

$$\tilde{s}_X = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Er misst die Stärke des linearen Zusammenhangs. Je näher die Messwerte an einer Geraden liegen, desto näher liegt r bei 1 (positive Steigung) oder bei -1 (negative Steigung). Bei $|r| < 0.5$ spricht man von einer schwachen Korrelation, bei $0.5 \leq |r| < 0.8$ von einer mittleren Korrelation und bei $|r| \geq 0.8$ von einer starken Korrelation.

(2) Zeigen Sie, dass für Pearsons Korrelationskoeffizient r_{XY} gilt:

$$|r_{\tilde{X}\tilde{Y}}| = |r_{XY}|$$

Diese Invarianz unter linearen Transformationen wird auch *Maßstabsunabhängigkeit* genannt.

(1) Die Definition des Pearson-Korrelationskoeffizienten r_{XY} zwischen den Merkmalen X und Y ist gegeben durch:

$$r_{XY} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

wobei $\text{cov}(X, Y)$ die Kovarianz von X und Y und σ_X, σ_Y die Standardabweichungen von X und Y sind.

(2) Wir betrachten nun die linearen Transformationen \tilde{X} und \tilde{Y} :

$$\begin{aligned}\tilde{X} &= a_X X + b_X, \quad a_X \neq 0 \\ \tilde{Y} &= a_Y Y + b_Y, \quad a_Y \neq 0\end{aligned}$$

Um die Kovarianz von \tilde{X} und \tilde{Y} zu berechnen, verwenden wir die Eigenschaften der Kovarianz:

$$\text{cov}(\tilde{X}, \tilde{Y}) = \text{cov}(a_X X + b_X, a_Y Y + b_Y)$$

Da die Kovarianz linear ist, können wir sie ausdrücken als:

$$\text{cov}(\tilde{X}, \tilde{Y}) = a_X a_Y \text{cov}(X, Y)$$

Nun betrachten wir die Standardabweichungen von \tilde{X} und \tilde{Y} :

$$\begin{aligned}\sigma_{\tilde{X}} &= |a_X| \sigma_X \\ \sigma_{\tilde{Y}} &= |a_Y| \sigma_Y\end{aligned}$$

Jetzt können wir den Pearson-Korrelationskoeffizienten zwischen \tilde{X} und \tilde{Y} berechnen:

$$r_{\tilde{X}\tilde{Y}} = \frac{\text{cov}(\tilde{X}, \tilde{Y})}{\sigma_{\tilde{X}}\sigma_{\tilde{Y}}} = \frac{a_X a_Y \text{cov}(X, Y)}{|a_X|\sigma_X|a_Y|\sigma_Y} = \frac{\text{cov}(X, Y)}{\sigma_X\sigma_Y} = r_{XY}$$

Also haben wir gezeigt, dass $|r_{\tilde{X}\tilde{Y}}| = |r_{XY}|$. Dies bestätigt die Invarianz des Pearson-Korrelationskoeffizienten unter linearen Transformationen, auch bekannt als Maßstabsunabhängigkeit.

TSRI-5

July 23, 2024

1 Übung 5

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Rimbault @Marine Rimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

werden Sie einen echten Datenfall bearbeiten, der die Firma Tesla betrifft. In diesem Fall werden Sie sich mit dem Thema *Datenaufbereitung* und dessen Konsequenzen beschäftigen. Nach dieser Aufgabe werden Sie sich noch einmal mit explorativer Analyse und Zusammenhangsmaßen (wie beispielsweise der *Mutual Information*) beschäftigen.

1.0.2 5.1 Der Fall Tesla: Autopilot

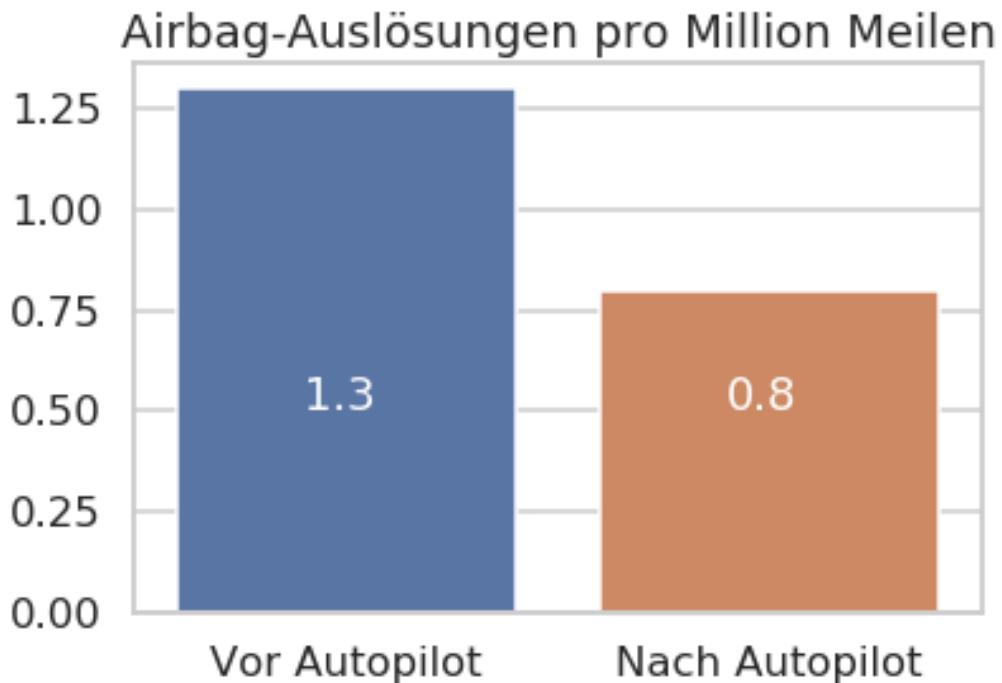
- In diesem Datenfall üben Sie Ihre Fähigkeiten, Datensätze zu erschließen, aufzubereiten und auf Plausibilität zu prüfen.

Seit wenigen Jahren verstärken verschiedene Unternehmen und Startups ihre Bemühungen, autonome Fahrzeuge zu entwickeln. Solche selbststeuerenden Fahrzeuge versprechen eine Vielzahl neuer Anwendungen und haben das Potenzial, unser Verhältnis zu Autos zu verändern. Manche Visionäre sprechen bereits von “mobility as a service”, bei der Autos nicht mehr besessen, sondern per App für eine Fahrt geliehen werden und selbstständig fahren.

Bei der Entwicklung von autonomen Fahrzeugen verfolgen verschiedene Hersteller eine schrittweise Strategie, nach der Assistenzfunktionen zum Parken oder für das Halten der Fahrspur nach und nach den Fahrern zur Verfügung gestellt werden. Tesla stellt den sogenannten *Autopilot* bereit, der das Feature *Autosteer* zum automatischen Halten der Fahrspur enthält.

Im Jahr 2016 verunglückte Joshua Brown in Florida in einem Tesla Model S, während das Autosteer Feature aktiviert war. Er war damit der erste (soweit bekannte) Todesfall, der in einem selbstfahrenden Auto eingetreten ist.

Die NHTSA, eine Bundesbehörde der USA im Bereich des Verkehrsministeriums, untersuchte den Fall und forderte Zahlen von Tesla an, die die Sicherheit des Autosteering Features belegen sollten. Anhand dieser Zahlen konnte die Behörde im Jahr 2017 folgende Feststellung machen:



Die Anzahl der Airbag-Auslösungen dient dabei als ein Anhaltspunkt für die Anzahl der Unfälle. Dieser Auswertung zufolge reduziert *Autosteer* die Unfallwahrscheinlichkeit um 40%!

Der dieser Statistik zugrundeliegende Datensatz wurde aufgrund eines Antrags basierend auf dem Informationsfreiheitsgesetz der USA (Freedom of Information Act) verfügbar gemacht und steht Ihnen in dieser Übung zur Verfügung.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Wir werden die folgenden Eigenschaften [1-6] untersuchen:

1. Kilometerstand (in Meilen) bevor der Installation von Autosteer (“1l Previous Mileage before Autosteer Install”).
 - Dieser Wert wurde abgelesen vor Installation von Autosteer.
2. Kilometerstand (in Meilen) nach der Installation von Autosteer (“1l Next Mileage after Autosteer Install”)
 - Dieser Wert wurde abgelesen nach Installation von Autosteer.
3. Gefahrene Meilen vor Autosteer-Installation (“Miles before Autosteer”)
 - Dieser Wert wurde von Tesla so angegeben.
4. Gefahrene Meilen nach Autosteer Installation (“Miles after Autosteer”)
 - Dieser Wert wurde von Tesla so angegeben.
5. Airbag-Auslösungen vor Autosteer-Installation (“Airbag events before Autosteer”)

- Dieser Wert wurde von Tesla so angegeben.
6. Airbag-Auslösungen nach Autosteering-Installation (“Airbag events after Autosteering”)
- Dieser Wert wurde von Tesla so angegeben.

Ihre Aufgaben

Für die nachfolgenden Aufgaben benötigen Sie Ihre Neugier und etwas detektivisches Gespür.

- (1) Betrachten Sie zunächst mit einer Tabellenkalkulation den oben hinterlegten Datensatz. Suchen Sie nach den oben erwähnten sechs Eigenschaften.
 - Importieren Sie nun mithilfe von Pandas die oben sechs erwähnten Eigenschaften aus der Excel-Datei. Dabei kann es hilfreich sein, das entsprechende Sheet der Excel-Datei in Pandas anzugeben, aus dem Sie die Daten importieren müssen.

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
tesla: pd.DataFrame = pd.read_excel("tesla.xlsx", ↴
sheet_name="PE16_007_PRODUCTION DATA", usecols=['11 Previous Mileage before Autosteer Install', '11 Next Mileage after Autosteer Install', 'Miles before Autosteer', 'Miles after Autosteer', 'Airbag events before Autosteer', 'Airbag events after Autosteer'], skipfooter=4).dropna(how="all")
tesla.shape
```

[1]: (43781, 6)

- (2) Von wie vielen Fahrzeugen wurden Daten erhoben? Nennen Sie die Anzahl der Fahrzeuge, indem Sie sie aus Ihren Daten aus Teilaufgabe (1) bestimmen.

43.781 Autos

- (3) Reproduzieren Sie die beiden Zahlen der NHTSA Behörde der USA, d.h. berechnen Sie die Anzahl der Airbag-Auslösungen vor Autosteering-Installation pro 1 Million gefahrener Meilen und nach Autosteering-Installation pro 1 Million gefahrener Meilen und geben Sie sie an.

```
[2]: airbag_events_before = tesla['Airbag events before Autosteer'].sum() / ↴
(tesla['Miles before Autosteer'].sum() / 1e6)
airbag_events_after = tesla['Airbag events after Autosteer'].sum() / ↴
(tesla['Miles after Autosteer'].sum() / 1e6)

print(f"Anzahl der Airbag-Auslösungen vor Autosteering-Installation: ↴
{airbag_events_before:.2f} pro 1 Million gefahrener Meilen.")
print(f"Anzahl der Airbag-Auslösungen nach Autosteering-Installation: ↴
{airbag_events_after:.1f} pro 1 Million gefahrener Meilen.")
```

Anzahl der Airbag-Auslösungen vor Autosteering-Installation: 1.3274 pro 1 Million gefahrener Meilen.

Anzahl der Airbag-Auslösungen nach Autosteering-Installation: 0.8140 pro 1 Million gefahrener Meilen.

(4) Bis hierher stimmen unsere Analysen mit der US-Behörde überein. Doch es gibt Ungereimtheiten in den Daten. Betrachten Sie dazu die Datenreihen 1 und 2, d.h. die Kilometerstände (in Meilen) der Fahrzeuge vor und nach der Installation von Autosteer. Vor der Installation von Autosteer wird der Kilometerstand erhoben; nach der Installation von Autosteer wird der Kilometerstand erhoben:

1. Welche Erwartungshaltung haben Sie an die beiden Datensätze? (1 Satz)
2. Explorieren Sie diese beiden Datensätze (Stichwort: Explorative Analyse). Was fällt Ihnen auf? (1 Satz)

Wenn Sie partout nicht mehr weiter kommen, gebe ich Ihnen einen Tipp: Wenn Sie vor der Installation von Autosteer den Kilometerstand abschreiben würden, und nach der Installation ebenfalls den Kilometerstand notieren, würden sich die beiden Kilometerstände unterscheiden oder sollten sie gleich sein?

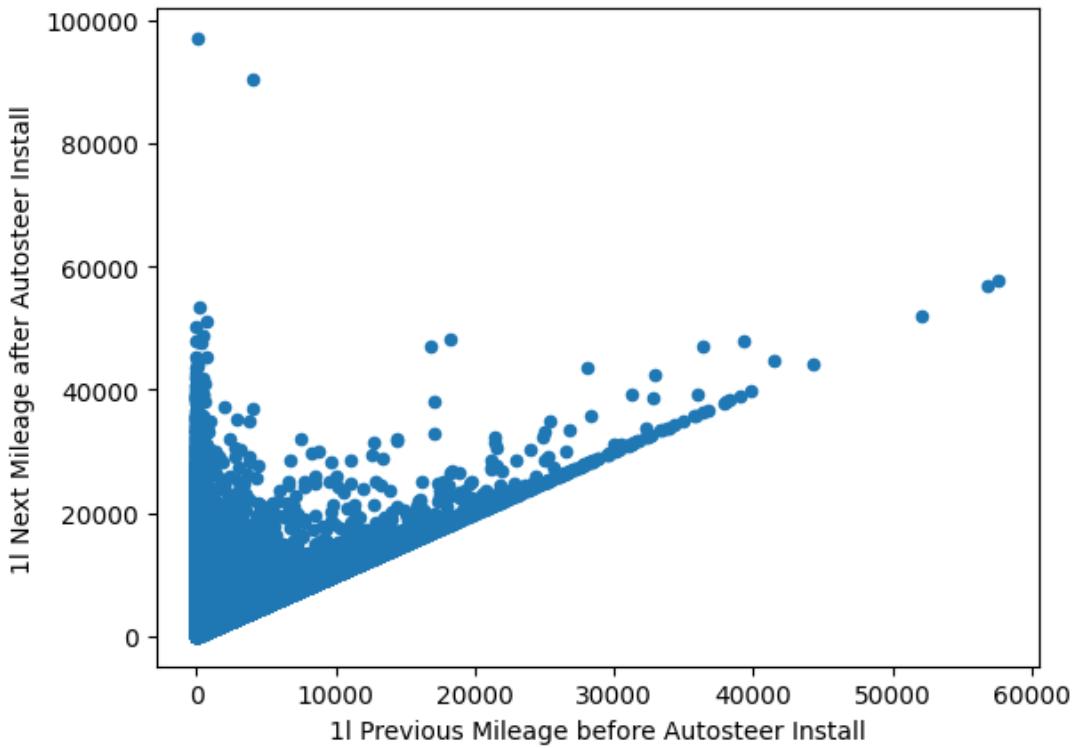
Die Erwartung ist, - dass die Installation von Autosteer nicht oder nicht viel an der Fahrzeugnutzung Meilen/Zeit ändert. - dass der Kilometerstand sich durch die Installation von Autosteer in der Werkstatt nicht ändert.

```
[3]: tesla[['11 Previous Mileage before Autosteer Install', '11 Next Mileage after Autosteer Install']].describe()
```

```
[3]:      11 Previous Mileage before Autosteer Install \
count                  43699.000000
mean                   1482.599991
std                    3698.902472
min                     0.000000
25%                     0.000000
50%                     0.000000
75%                     245.000000
max                    57627.000000

      11 Next Mileage after Autosteer Install
count                  43556.000000
mean                   4555.421917
std                    6164.083570
min                     0.000000
25%                     0.000000
50%                     2130.000000
75%                     7052.000000
max                    97066.000000
```

```
[4]: tesla.plot(x='11 Previous Mileage before Autosteer Install', y='11 Next Mileage after Autosteer Install', kind="scatter")
plt.show()
```



Eigentlich würden wir hier eine Gerade erwarten, da nicht davon auszugehen ist, dass sich der Kilometerstand bei der Installation ändert.

- (5) Problem 1: Identifizieren Sie anhand Ihrer Beobachtungen aus Schritt (4) die Fahrzeuge, bei denen Sie genau feststellen konnten, zu welchem Kilometerstand Autosteer tatsächlich installiert worden sein muss. Betrachten Sie **nur** die Fahrzeuge aus dieser Menge, für die tatsächlich gefahrenen Kilometer (mehr als 0) vor der Autosteer-Installation ausgewiesen wurden.
- Wie viele Fahrzeuge erfüllen die gerade genannten Kriterien?
 - Bestimmen Sie die Anzahl der Airbag-Auslösungen pro gefahrenen 1 Million Meilen vor und nach der Installation von Autosteer. Geben Sie diese Zahlen an. Was stellen Sie fest?
 - Vergleichen Sie Ihre Zahlen mit den oben angegebenen Zahlen der NHTSA. Sehen Sie gleiche oder unterschiedliche Tendenzen?

```
[5]: subset = tesla[(tesla['Miles before Autosteer'] > 0.0) & (tesla['1l Previous Mileage before Autosteer Install'] == tesla['1l Next Mileage after Autosteer Install'])]
print(f"Fahrzeuge, bei denen die Kriterien übereinstimmen: {len(subset)}")

events_before_autosteer_per_mile = subset["Airbag events before Autosteer"] . sum() / (subset["Miles before Autosteer"] . sum() / 1e6)
```

```

events_after_autosteer_per_mile = subset["Airbag events after Autosteer"].sum() / (subset["Miles after Autosteer"].sum() / 1e6)

print(f"Airbag-Auslösungen vor Autosteer: {events_before_autosteer_per_mile}")
print(f"Airbag-Auslösungen nach Autosteer: {events_after_autosteer_per_mile}")

```

Fahrzeuge, bei denen die Kriterien übereinstimmen: 5719

Airbag-Auslösungen vor Autosteer: 0.7607940388363484

Airbag-Auslösungen nach Autosteer: 1.2504243627681144

- (6) Problem 2: Identifizieren Sie anhand Ihrer Beobachtungen aus Schritt (4) die Fahrzeuge, bei denen Sie genau feststellen konnten, zu welchem Kilometerstand Autosteer tatsächlich installiert worden sein muss. Betrachten Sie nun alle Fahrzeuge aus dieser Menge, die **keine gefahrenen Kilometer vor der Autosteer-Installation** aufweisen.

- Wie viele Fahrzeuge erfüllen die gerade genannten Kriterien?
- Wie viele Airbag-Auslösungen vor der Installation von Autosteer beobachten Sie für diese Gruppe von Fahrzeugen?
- Welche Auswirkung hat Ihre Beobachtung auf die Größe der Anzahl an Airbag-Auslösungen pro 1 Million gefahrener Meilen, wie sie die NHTSA berechnet hat?

```

[6]: subset = tesla[(tesla['Miles before Autosteer'] == 0.0) & (tesla['11 Previous Mileage before Autosteer Install'] == tesla['11 Next Mileage after Autosteer Install'])]

print(f"Fahrzeuge, die keine gefahrenen Kilometer vor der Autosteer-Installation aufweisen: {len(subset)}")

events_before_autosteer = subset["Airbag events before Autosteer"].sum()

print(f"Anzahl der Airbag-Auslösungen vor Autosteer dieser Fahrzeuge: {events_before_autosteer}")

```

Fahrzeuge, die keine gefahrenen Kilometer vor der Autosteer-Installation aufweisen: 14689

Anzahl der Airbag-Auslösungen vor Autosteer dieser Fahrzeuge: 1.0

- (7) Problem 3: Identifizieren Sie anhand Ihrer Beobachtungen aus Schritt (4) die Fahrzeuge, bei denen Sie **nicht** genau feststellen können, zu welchem Kilometerstand Autosteer tatsächlich installiert wurde. Betrachten Sie **nur** die Fahrzeuge aus dieser Menge, für die tatsächlich mehr als 0 gefahrene Kilometer vor der Autosteer-Installation ausgewiesen wurden.

1. Wie viele Autos sind in dieser Menge enthalten? Wie viele Airbag-Auslösungen sind in dieser Menge enthalten?
2. Bestimmen Sie die Gesamtanzahl der gefahrenen Meilen über alle Fahrzeuge dieser Menge, die zwischen dem Kilometerstand vor Autosteer-Installation und nach Autoinstallation gefahren wurden. Dies sind die "Gap"-Meilen, bei denen wir nicht wissen, zu welcher exakten Kilometerstand Autosteer installiert wurde.
3. Bestimmen Sie die Gesamtanzahl aller Meilen dieser Menge, die vor der Autosteer-Installation gefahren wurden.
4. Vergleichen Sie die Zahlen aus Schritt 7.2 und 7.3: Ist die gefahrene Gesamtmeilenzahl

vor Autosteer-Installation deutlich größer oder kleiner als die Gesamtmeilenzahl der “Gap”-Meilen?

Welchen Schluss ziehen Sie für die Validität der NHTSA Untersuchung aus Ihren Beobachtungen aus diesem Schritt? (1-3 Sätze).

```
[7]: subset = tesla[(tesla['Miles before Autosteer'] > 0.0) & (tesla['1l Previous Mileage before Autosteer Install'] != tesla['1l Next Mileage after Autosteer Install'])]

print(f"Es sind {len(subset)} Fahrzeuge mehr als 0 Meilen vor der Autosteer Installation gefahren, bei denen sich nicht feststellen ließ, wann Autosteer tatsächlich installiert wurde.")

gap_miles = subset["1l Next Mileage after Autosteer Install"] - subset["1l Previous Mileage before Autosteer Install"]
print(f"Gap Miles = {gap_miles.sum():23.1f}")
print(f"Miles before Autosteer = {subset['Miles before Autosteer'].sum()}")
```

Es sind 8920 Fahrzeuge mehr als 0 Meilen vor der Autosteer Installation gefahren, bei denen sich nicht feststellen ließ, wann Autosteer tatsächlich installiert wurde.

Gap Miles = 57955464.0
Miles before Autosteer = 22726819.0

Die Anzahl der “Gap”-Meilen ist mehr als doppelt so groß wie die Anzahl der in die NHTSA Rechnung einbezogenen Meilen vor der Autosteer Installation.

- (8) Problem 4: Identifizieren Sie anhand Ihrer Beobachtungen aus Schritt (4) die Fahrzeuge, bei denen Sie **nicht** genau feststellen können, zu welchem Kilometerstand Autosteer tatsächlich installiert wurde. Betrachten Sie **nur** die Fahrzeuge aus dieser Menge, für die nur 0 gefahrene Kilometer vor der Autosteer-Installation ausgewiesen wurden.
 1. Wie viele Autos sind in dieser Menge enthalten?
 2. Wie viele Airbag-Auslösungen können Sie vor und nach der Autosteer-Installation feststellen?

Wie deuten Sie Ihre Beobachtungen hinsichtlich der Validität der NHTSA Untersuchung? (1-3 Sätze)

```
[8]: subset = tesla[(tesla['Miles before Autosteer'] == 0.0) & (tesla['1l Previous Mileage before Autosteer Install'] != tesla['1l Next Mileage after Autosteer Install'])]

print(f"Fahrzeuge, die keine gefahrenen Kilometer vor der Autosteer-Installation aufweisen und bei denen der Installationszeitpunkt nicht genau festgestellt werden kann: {len(subset)}")

events_before_autosteer = subset["Airbag events before Autosteer"].sum()
events_after_autosteer = subset["Airbag events after Autosteer"].sum()

print(f"Anzahl der Airbag-Auslösungen vor Autosteer dieser Fahrzeuge: {events_before_autosteer}")
```

```
print(f"Anzahl der Airbag-Auslösungen nach Autosteer dieser Fahrzeuge:\n"
      f"{events_after_autosteer}")
```

Fahrzeuge, die keine gefahrenen Kilometer vor der Autosteer-Installation aufweisen und bei denen der Installationszeitpunkt nicht genau festgestellt werden kann: 14452

Anzahl der Airbag-Auslösungen vor Autosteer dieser Fahrzeuge: 17.0

Anzahl der Airbag-Auslösungen nach Autosteer dieser Fahrzeuge: 48.0

- (9) Abschluss: Betrachten Sie Ihre Ergebnisse zu Problemen 1-4. Lässt sich aufgrund der vorliegenden Zahlen die Aussage treffen, dass durch Autosteer die Anzahl der Airbag-Auslösungen pro gefahrene 1 Million Meilen zurückgegangen ist und damit das Fahren sicherer geworden ist? Argumentieren Sie in wenigen Sätzen.

<https://www.heise.de/autos/artikel/Rechenfehler-Tesla-Autosteer-senkt-Unfallrate-nicht-4303949.html>

1.1 5.2 Multivariate explorative Analyse II (Palmer Pinguine & Parallel Coordinates Plots)

In dieser Übung untersuchen Sie Daten verschiedener Pinguin-Gattungen, die während einer Forschungsmission in der Antarktis an der [Palmer-Station](#) gesammelt wurden. Der Datensatz stammt aus einer [Publikation](#), die das Futtersuchverhalten der Tiere untersuchte. Sie werden die Daten mithilfe von [Parallel Coordinates Plots](#) (PCPs, ||-Plots) untersuchen. PCPs sind wirkungsvolle Instrumente für eine Exploration höherdimensionaler Räume, die Sie im Rahmen dieser Übung kennenlernen werden. Sie spielen in vielen explorativen Analysen eine Rolle, beispielsweise auch bei der [Parameterexploration](#) zum Training tiefer neuronaler Netze.

Es begrüßen Sie:

Hinweis

- Für diese Übung müssen Sie die Bibliotheken [hiplot](#) und [seaborn](#) installieren, sollten Sie sie noch nicht installiert haben.

- (1) Importieren Sie den Datensatz in einen Pandas DataFrame.

- URL des Datensatzes: <https://data.bialonski.de/ds/palmerpenguins.csv>

```
[9]: penguins: pd.DataFrame = pd.read_csv('palmerpenguins.csv')
penguins
```

```
[9]:    species     island  bill_length_mm  bill_depth_mm  flipper_length_mm \
0       Adelie  Torgersen        39.1          18.7           181.0
1       Adelie  Torgersen        39.5          17.4           186.0
2       Adelie  Torgersen        40.3          18.0           195.0
3       Adelie  Torgersen         NaN            NaN            NaN
4       Adelie  Torgersen        36.7          19.3           193.0
..        ...
339  Chinstrap     Dream        55.8          19.8           207.0
340  Chinstrap     Dream        43.5          18.1           202.0
```

```

341  Chinstrap      Dream        49.6       18.2       193.0
342  Chinstrap      Dream        50.8       19.0       210.0
343  Chinstrap      Dream        50.2       18.7       198.0

      body_mass_g
0            3750.0
1            3800.0
2            3250.0
3              NaN
4            3450.0
..
339          ...
340          4000.0
341          3400.0
342          3775.0
342          4100.0
343          3775.0

```

[344 rows x 6 columns]

(2) Untersuchen Sie den Datensatz und beantworten Sie bitte folgende Fragen:

- Welche Features sind enthalten?
- Wie viele Datenpunkte gibt es?
- Wie viele Klassen (`species`) gibt es?
- Wie viele Datenpunkte gibt es pro Klasse?

```
[10]: print(f"Features: {list(penguins.columns)}")
print(f"Datenpunkte: {penguins.shape[0]}")
print(f"Verschiedene Spezies: {penguins['species'].unique()}")
print(f"Datenpunkte pro Spezies:\n{penguins.groupby('species').size()}")
```

```

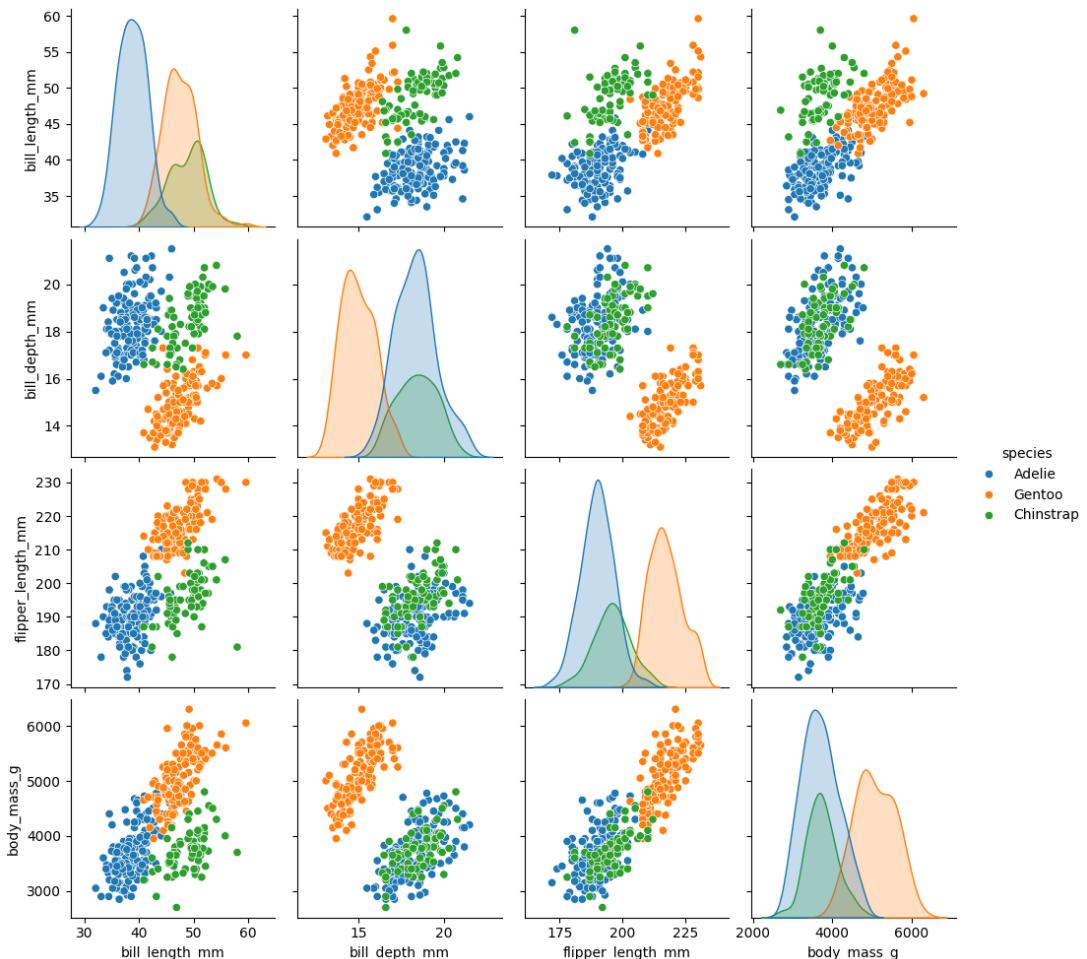
Features: ['species', 'island', 'bill_length_mm', 'bill_depth_mm',
'flipper_length_mm', 'body_mass_g']
Datenpunkte: 344
Verschiedene Spezies: ['Adelie' 'Gentoo' 'Chinstrap']
Datenpunkte pro Spezies:
species
Adelie      152
Chinstrap     68
Gentoo      124
dtype: int64

```

(3) Erstellen Sie einen Scatter-Matrix-Plot des oben beschriebenen Datensatzes, **wobei jeder Datenpunkt durch eine Farbe entsprechend der Spezies gekennzeichnet wird**. Verwenden Sie dazu die Bibliothek `seaborn` und die Funktion `pairplot`.

Hinweis: `pairplot` interpretiert die numerischen Spalten des Datensatzes automatisch als Features. Aus diesem Grund erscheinen nicht zwingend alle Spalten im Scatter-Matrix-Plot.

```
[11]: import seaborn as sns
sns.pairplot(penguins, hue='species')
plt.show()
```



Falls Sie wissen möchten, was es mit dem sogenannten “Bill” auf sich hat:

- (4) Die drei Pinguingattungen erscheinen im Scatter-Matrix-Plot der vorherigen Teilaufgabe als “Punktewolken” (mehr dazu in einer späteren Vorlesungseinheit zum Thema *Clustering*).
 - Versuchen Sie nun, die drei Gattungen mithilfe sogenannter “Bounding Boxes” in den Plots zu identifizieren. Suchen Sie dazu nach Featurepaaren, in denen sich die Klassen gut voneinander unterscheiden lassen. Für jede “Bounding Box” benötigen Sie jeweils ein Intervall für die x- und y-Achse, welche die Punktewolke einer Gattung möglichst gut einschließt.

Fiktives Beispiel: - Gattung A: $\text{body_mass_g} \in [3000, 5000]$ und $\text{flipper_length_mm} \in [170, 200]$

- Gattung Adelie: $\text{bill_depth_mm} \in [15, 22]$ und $\text{bill_length_mm} \in [30, 45]$
- Gattung Gentoo: $\text{bill_depth_mm} \in [12, 18]$ und $\text{bill_length_mm} \in [40, 55]$
- Gattung Gentoo: $\text{bill_depth_mm} \in [12, 18]$ und $\text{flipper_length_mm} \in [200, 235]$

- Gattung Chinstrap: `bill_depth_mm` $\in [16, 21]$ und `bill_length_mm` $\in [40, 55]$
- (5) Erstellen Sie einen interaktiven Parallel Coordinates Plot (||-Plot) des oben beschriebenen Datensatzes.
- Sollten Sie keinen interaktiven Plot erstellen können, bitten wir Sie, für diese Übung in einen anderen Editor (wie Jupyter Notebook oder Jupyter Lab) zu wechseln.
1. Erstellen Sie einen ||-Plot mit der Bibliothek `hiplot` und den Funktionen `Experiment.from_dataframe` sowie `Experiment.display`.
 2. Nutzen Sie die Methode `colorby`, um das Feature `species` für die Farbkodierung auszuwählen. Die Linien Ihres Plots werden dann gemäß der Pinguingattung eingefärbt.
 3. **Wichtig:** Sie müssen sich mit der (nicht direkt ersichtlichen) interaktiven Bedienung des Plots vertraut machen. Klicken Sie dazu im Plot oben auf “Help” und durchlaufen Sie das dort hinterlegte Tutorial.

```
[12]: import hiplot as hip

experiment = hip.Experiment.from_dataframe(penguins)
experiment.colorby = "species"
experiment.display()
```

```
<IPython.core.display.Javascript object>
<IPython.core.display.HTML object>

[12]: <hiplot.ipynotebook.IPythonExperimentDisplayed at 0x7f75c540eea0>
```

- (6) Versuchen Sie, mithilfe von geschicktem Slicing der verschiedenen Features im Parallel Coordinates Plot die jeweiligen Gattungen zu identifizieren. Orientieren Sie sich gerne an Ihren Ergebnissen bzw. Intervallen aus Teilaufgabe (4).
- (7) Versuchen Sie nun, bessere “Bounding Boxes” zu finden, um die Gattungen besser voneinander zu trennen. Betrachten Sie dabei insbesondere Features, die in der vorherigen Aufgabe nicht verwendet wurden. (*Zur Erinnerung: bisher wurden nur numerische Features betrachtet.*)
- Gattung Adelie: `island` $\in \{Biscoe; Dream; Torgersen\}$ und `bill_length_mm` $\in [34, 43]$
 - Gattung Gentoo: `island` $\in \{Biscoe\}$ und `bill_length_mm` $\in [45, 56]$
 - Gattung Chinstrap: `island` $\in \{Dream\}$ und `bill_length_mm` $\in [45, 53]$
- (8) Vergleichen Sie die beiden Visualisierungen und diskutieren Sie die Vor- und Nachteile der beiden Ansätze.

Die Visualisierung durch den Pairplot ermöglicht es, die Beziehung verschiedener Kombinationen von Feature-Paaren grafisch darzustellen. Auf der Diagonalen ist zudem eine anschauliche Darstellung der Verteilung der einzelnen Features in den verschiedenen Spezies abgebildet. Ein Nachteil ist jedoch, dass die Visualisierung auf numerische Features beschränkt ist.

Der ||-Plot bietet hingegen die Möglichkeit, auch nicht-numerische Features darzustellen und näher zu untersuchen. Im Gegensatz zum Pairplot lassen sich mit dem ||-Plot jedoch nicht auf einen Blick der Zusammenhang eines Features in Kombination mit verschiedenen anderen Features darstellen. Die Reihenfolge der Achsen bestimmt hier die Feature-Paarungen, sodass die Betrachtung verschiedener Anordnungen notwendig ist.

1.1.1 5.3 Zusammenhangsmaße (Teil 1): Spearman vs Pearson

In dieser Übung werden wir synthetische Daten generieren und damit Spearmans- und Pearsons Korrelationskoeffizienten untersuchen.

Ihre Daten

Bei den Daten handelt es sich um die berühmte Fibonacci-Folge, eine unendliche Folge, die vom italienischen Mathematiker Leonardo Fibonacci im 12. Jahrhundert zur Beschreibung der Entwicklung einer Kaninchenpopulation aufgestellt wurde.

Sei y_n das n-te Glied der Folge und seien $y_1 = 0$ und $y_2 = 1$. Dann ist y_n für $n \geq 3$ definiert als

$$y_n = y_{n-1} + y_{n-2}.$$

Ihre Aufgaben

Nutzen Sie hauptsächlich Numpy (und ein wenig Pandas) zur Bearbeitung dieser Aufgaben.

- (1) Schreiben eine Funktion, die für einen gegebenen Wert $n \in \mathbb{N}$ das Glied der Fibonacci-Folge zurückgibt.

```
[13]: def fibonacci(n: int) -> np.ndarray:  
    fibonacci_nums: list = [0, 1]  
    while len(fibonacci_nums) < n:  
        fibonacci_nums.append(fibonacci_nums[-1] + fibonacci_nums[-2])  
    return np.array(fibonacci_nums)  
  
fibonacci(10)[-1]
```

```
[13]: np.int64(34)
```

- (2) Erzeugen Sie Ihren Datensatz. Sei $n = \{1, \dots, 30\}$. Erstellen Sie mit Ihrer Funktion aus Schritt (1) die Folge y_n für alle n . Sie erhalten damit Wertepaare (n, y_n) , die Sie z.B. in einem Numpy Array hinterlegen können.

```
[14]: data = np.array(list(zip(np.arange(1, 31), fibonacci(30))))
```

- (3) Schlagen Sie die Definition des Pearson Korrelationskoeffizienten in der Vorlesung nach und implementieren Sie eine Funktion, die zwei Datenreihen entgegennimmt und Pearsons Korrelationskoeffizient zurückgibt.

Pearsons Korrelationskoeffizient (Charakterisierung der Stärke linearer Zusammenhänge) r ist der Quotient aus der Kovarianz \tilde{s}_{XY} und der Standardabweichung der Stichproben $\tilde{s}_X \tilde{s}_Y$.

$$r = \frac{\tilde{s}_{XY}}{\tilde{s}_X \tilde{s}_Y}$$

Kovarianz:

$$\tilde{s}_{XY} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Standardabweichung der Stichprobe X :

$$\tilde{s}_X = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

```
[15]: def pearsons(X: np.ndarray, Y: np.ndarray) -> float:
    assert X.shape == Y.shape, "X und Y müssen die gleiche Shape haben"
    n: int = X.shape[0]
    X_diff, Y_diff = X - X.mean(), Y - Y.mean()
    covariance = np.multiply(X_diff, Y_diff).sum() / (n-1)
    sigma_x = np.sqrt(np.power(X_diff, 2).sum() / (n-1))
    sigma_y = np.sqrt(np.power(Y_diff, 2).sum() / (n-1))
    return covariance / (sigma_x * sigma_y)
```

- (4) Bestimmen Sie dann den Pearson Korrelationskoeffizienten zwischen der Datenreihe n und y_n mithilfe Ihrer Funktion aus (3) und notieren Sie sich den Wert.

```
[16]: pearsons(data[:, 0], data[:, 1])
```

```
[16]: np.float64(0.6034880493240204)
```

- (5) Schlagen Sie in den Vorlesungsfolien die Definition von Spearmans Korrelationskoeffizienten nach. Kopieren Sie sich den Code aus Schritt (3) und erzeugen Sie daraus eine neue Funktion, die Spearmans Korrelationskoeffizienten implementiert. Um die Ränge zu berechnen, können Sie Pandas `rank` Funktion nutzen.

```
[17]: def spearman(X: np.ndarray, Y: np.ndarray) -> float:
    return pearsons(pd.Series.rank(pd.DataFrame(X)).to_numpy(), pd.Series.
                    rank(pd.DataFrame(Y)).to_numpy())
```

- (6) Nutzen Sie Ihre Funktion aus Schritt (5), um Spearmans Korrelationskoeffizient zwischen n und y_n zu berechnen, die Sie in Schritt (2) erzeugt hatten.

```
[18]: spearman(data[:, 0], data[:, 1])
```

```
[18]: np.float64(0.9998887591075041)
```

- (7) Vergleichen Sie den Wert von Pearsons Korrelationskoeffizienten (aus Schritt 4) mit dem Wert von Spearmans Korrelationskoeffizienten (aus Schritt 6). Was fällt Ihnen auf? (1 Satz). Was ist die Ursache für den Unterschied zwischen beiden Werten, den Sie feststellen? (1-3 Sätze)

Der berechnete Korrelationskoeffizient nach Spearman ist größer. Der Zusammenhang zwischen n und y_n ist zwar monoton steigend, aber nicht linear, weshalb Pearsons Korrelationskoeffizient nur auf eine mittlere Korrelation hindeutet, wohingegen Spearmans Koeffizient auf Basis der Ränge eine hohe Monotonie zwischen den Daten misst.

1.1.2 5.4 Zusammenhangsmaße (Teil 2): Mutual Information, Pearson, Spearman

In dieser Übung werden Sie Zusammenhänge zwischen verschiedenen Datensätzen mithilfe der Zusammenhangsmaße untersuchen, die Sie in der Vorlesung kennengelernt haben. Daneben werden

Sie vertraut mit dem Importieren von Daten aus Excel-Tabellen.

- Nutzen Sie Pandas und Numpy zum Bearbeiten der folgenden Aufgaben.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

(1) Importieren Sie die Daten mithilfe der [Funktionalität](#) von Pandas in einen DataFrame.

- Visualisieren Sie die Daten zunächst **nicht** (aus didaktischen Gründen). Sie werden die Daten erst im Teilschritt (5) visualisieren.

```
[19]: df_dependency = pd.read_excel('dependency_data.xlsx')
df_dependency.head()
```

```
[19]:   Unnamed: 0         X         y1         y2         y3         y4
0          0  0.097627  1.017967 -0.338989  9.902373 -0.965421
1          1  0.430379  0.102369 -0.446531  9.569621  0.133793
2          2  0.205527  0.823668  0.354986  9.794473  0.266585
3          3  0.089766  1.093290  0.316893  9.910234 -0.749156
4          4 -0.152690  0.635302  0.391532 10.152690  0.220924
```

(2) Wir betrachten im Folgenden Paare von Spalten, beispielsweise (X, y_1) , die wir *Datenreihenpaare* nennen werden.

Berechnen Sie mit Ihrem Code aus Übung 4 den Pearson- sowie den Spearman-Korrelationskoeffizienten für die Datenreihenpaare ...

1. (X, y_1)
2. (X, y_2)
3. (X, y_3)
4. (X, y_4)

```
[20]: print("    Pears.  Spear.")
for y in df_dependency.columns[2:]:
    print(f"{y} {pearsons(df_dependency['X'].to_numpy(), df_dependency[y].to_numpy()): 2.3f} {spearmen(df_dependency['X'].to_numpy(), df_dependency[y].to_numpy()): 2.3f}")
```

```
Pears.  Spear.
y1  0.018  0.042
y2  0.035  0.036
y3  0.329  -0.727
y4  0.018  0.019
```

(3) Beantworten Sie - ohne zu Visualisieren - anhand der Koeffizienten aus Teilaufgabe (2):

- Welche der Datenreihenpaare zeigen Korrelationen? Welche Datenreihenpaare sind unkorreliert?
- Gibt es Widersprüche zwischen den Korrelationskoeffizienten? Falls ja, für welche Daten?

Das Datenreihenpaar (X, y_3) zeigt eine schwach-positive Pearson Korrelation und eine mittlere, negative Spearmen Korrelation. Ein negativer monotonen Zusammenhang steht jedoch im Widerspruch zu einem schwach-positiven linearen Zusammenhang.

Die Übrigen Datenreihenpaare scheinen nicht korreliert zu sein.

- (4) Bestimmen Sie nun für die in Teilaufgabe (2) genannten Datenreihenpaare jeweils die Mutual Information. Ausgehend von Ihren Werten für die Mutual Information beantworten Sie bitte: Zwischen welchen Datenreihenpaaren gibt es Zusammenhänge, zwischen welchen Datenreihenpaare gibt es keine (großen) Zusammenhänge?
- Nutzen Sie für die Bestimmung der *Mutual Information* die [Implementierung](#) der scikit-learn Bibliothek. Hintergrund: Eine robuste Schätzung der Mutual Information ist anspruchsvoll und wurde von verschiedenen Forschungsgruppen untersucht. Einer der [weltweit bekanntesten Schätzer](#) der Mutual Information wurde von Forschern des Forschungszentrum Jülichs (am John von Neumann-Institut für Computing) im Jahr 2004 publiziert und wird auch von scikit-learn implementiert.

```
[21]: import math
from sklearn.feature_selection import mutual_info_regression
mutual_information = mutual_info_regression(df_dependency.loc[:, "y1":], df_dependency['X'])
print(mutual_information)
print(mutual_information[2], "nit")
print(mutual_information[2] * 1/math.log(2), "bit")
```

[0.84637242 0.88664582 2.54890351 0.02066885]
2.5489035129910853 nit
3.6772904578966963 bit

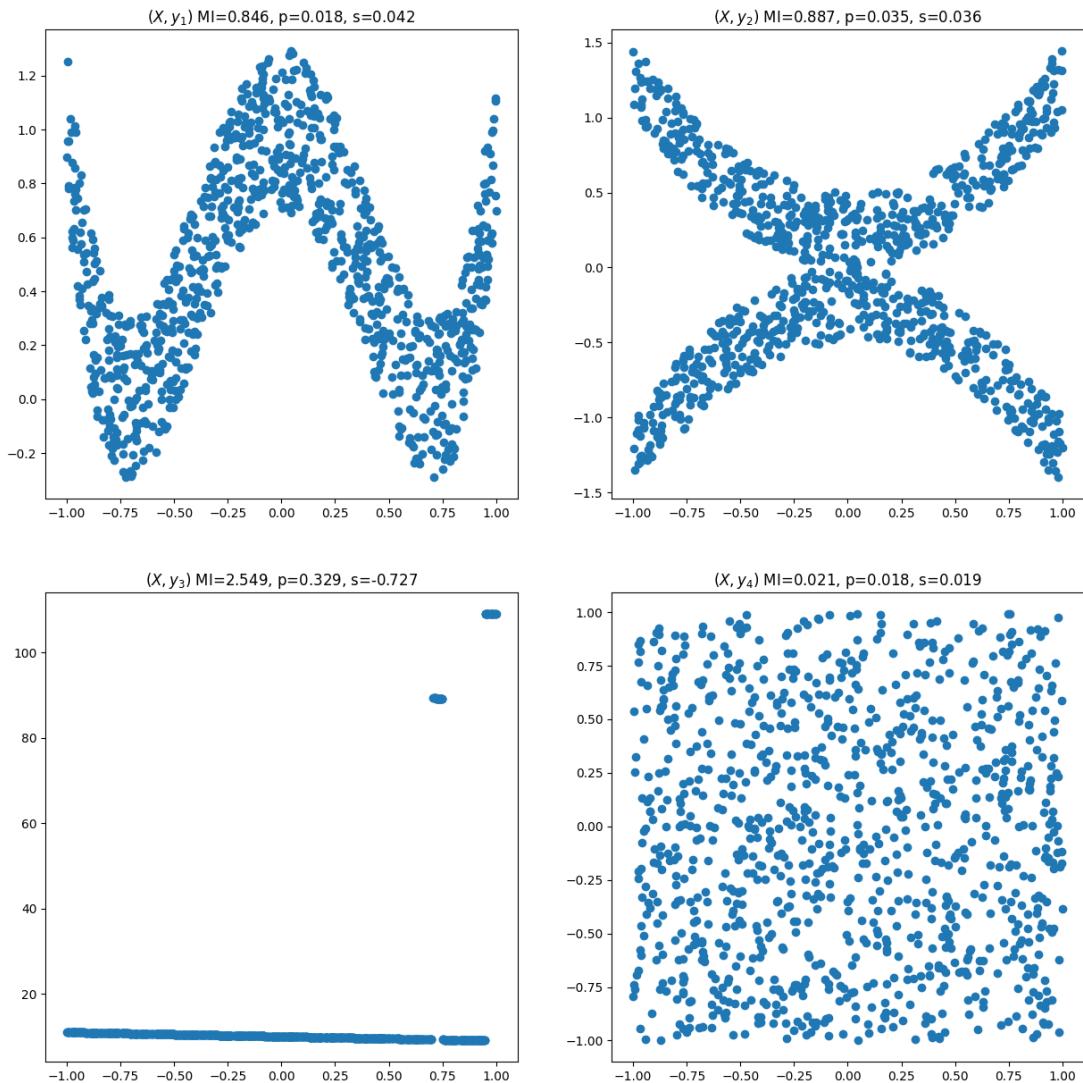
Die Mutual Information zwischen y_3 und X beträgt 2.5489 nat units oder 3.67729 bit. Dies deutet auf einen stärkeren Zusammenhang als bei den übrigen Paaren hin.

- (5) Visualisieren Sie nun alle Datenreihenpaare aus Teilaufgabe (2) in Scatterplots und beschriften Sie jeden Plot mit den entsprechenden Werten des Pearson-, Spearman-Korrelationskoeffizienten und dem Wert der Mutual Information.

```
[22]: fig, axs = plt.subplots(2, 2, figsize=(15, 15))

for i, y in enumerate(df_dependency.columns[2:]):
    axs.ravel()[i].scatter(df_dependency['X'], df_dependency[y])
    p = pearsons(df_dependency['X'].to_numpy(), df_dependency[y].to_numpy())
    s = spearmen(df_dependency['X'].to_numpy(), df_dependency[y].to_numpy())
    axs.ravel()[i].set_title(f"${X, {y[0]}-{y[-1]}}$ MI={mutual_information[i]:.1f}, p={p:.1f}, s={s:.1f})"

plt.show()
```



- (6) Begründen Sie nun für jedes Datenreihenpaar, warum Sie Korrelationen oder keine Korrelation durch Pearson, Spearman und die Mutual Information angezeigt bekommen.
- Begründen Sie zusätzlich, warum Sie für Datenreihenpaar (X, y_3) so unterschiedliche Ergebnisse für den Pearson- und den Spearman-Korrelationskoeffizienten erhalten haben.
 - (X, y_1) : Keine Korrelation, denn obwohl ein Zusammenhang zwischen X und y_1 besteht, ist dieser nicht monoton und die Symmetrie des Graphen führt dazu, dass die Korrelationskoeffizienten nahe 0 sind.
 - (X, y_2) : Keine Korrelation, denn obwohl ein Zusammenhang zwischen X und y_1 besteht, ist dieser nicht monoton und die Symmetrie des Graphen führt dazu, dass die Korrelationskoeffizienten nahe 0 sind.
 - (X, y_3) : Die Datenpunkte liegen größtenteils auf einer Geraden mit einer sehr kleinen negativen Steigung. Durch die starken Ausreißer nach oben war der Wert des Pearson-Koeffizienten

jedoch positiv und wies nur auf eine leichte Korrelation hin. Da der Spearman-Koeffizient robuster gegenüber Ausreißern ist, blieb hier sowohl das negative Vorzeichen als auch die Stärke der Korrelation erhalten.

- (X, y_4) : Keine Korrelation, da kein erkennbarer Zusammenhang zwischen der Verteilung von X und y_4 vorliegt.

TSRI-6

July 23, 2024

1 Übung 6

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden wir intensiv mit Dimensionsreduktion mittels Hauptkomponentenzerlegung (PCA) beschäftigen. Wir werden die PCA implementieren und damit verschiedene Datensätze untersuchen.

1.0.2 6.1 Eigengesichter (Bildkompression mittels PCA)

In den 90er Jahren haben Forscher des Olivetti Research Laboratory in Cambridge, basierend auf der PCA, eines der frühen Verfahren für Gesichtserkennung entwickelt. Das Olivetti Research Laboratory wurde wenige Jahre später von AT&T übernommen, und der [Datensatz](#), auf dem die Arbeiten beruhten, wurde unter dem Namen *Olivetti Faces Dataset* weltweit bekannt.

Wir werden uns in dieser Übung mit dem Aspekt der Dimensionsreduktion mithilfe von PCA beschäftigen, und wie sie zur Datenkompression von Gesichtsbildern eingesetzt werden kann. Die PCA-Richtungen (Hauptkomponenten) eines Datensatzes von Gesichtsbildern werden auch *Eigen gesichter* bzw. *Eigenfaces* genannt.

Ihre Daten

Die Olivetti Faces bestehen aus 400 Graustufenbildern (64x64 Pixel), die von 40 Personen stammen. Von jeder Person wurden 10 Gesichtsbilder angefertigt, wobei die Personen gebeten wurden, unterschiedliche Gesichtsausdrücke zu zeigen. Gleichzeitig wurde die Beleuchtung während der Fotoaufnahmen variiert.

- Importieren Sie den Datensatz durch das Ausführen der untenstehenden Code-Zelle.

```
[1]: import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import fetch_olivetti_faces

# Import data
```

```
data: np.ndarray  
targets: np.ndarray  
data, targets = fetch_olivetti_faces(data_home=". /", return_X_y=True)
```

Ihre Aufgaben

- Bei der Implementierung der PCA in den nachfolgenden Teilaufgaben nutzen Sie bitte ausschließlich *Numpy* und keine Funktionen von *sklearn*.

(1) Untersuchen Sie die Daten in `data`:

- Nennen Sie die Anzahl der Samples (Gesichter) in diesem Datensatz.
- Nennen Sie die Anzahl der Features (Merkmale, Pixel) jedes Gesichts.
- Prüfen Sie, in welchem Wertebereich die Pixelwerte variieren und nennen Sie den Wertebereich.

```
[2]: print(f"data.shape={}\n{data.max()=} bis {data.min()=}, {data.mean()=}")
```

```
data.shape=(400, 4096)  
data.max()=np.float32(1.0) bis data.min()=np.float32(0.0),  
data.mean()=np.float32(0.5470426)
```

Im Datensatz sind 400 Samples mit je 4096 Features. Die Pixelwerte variieren um 0.547 im Wertebereich von [0, 1].

(2) Nutzen Sie Ihre Erkenntnisse aus (1) und visualisieren Sie das Gesicht 296 (0-basierte Zählung).

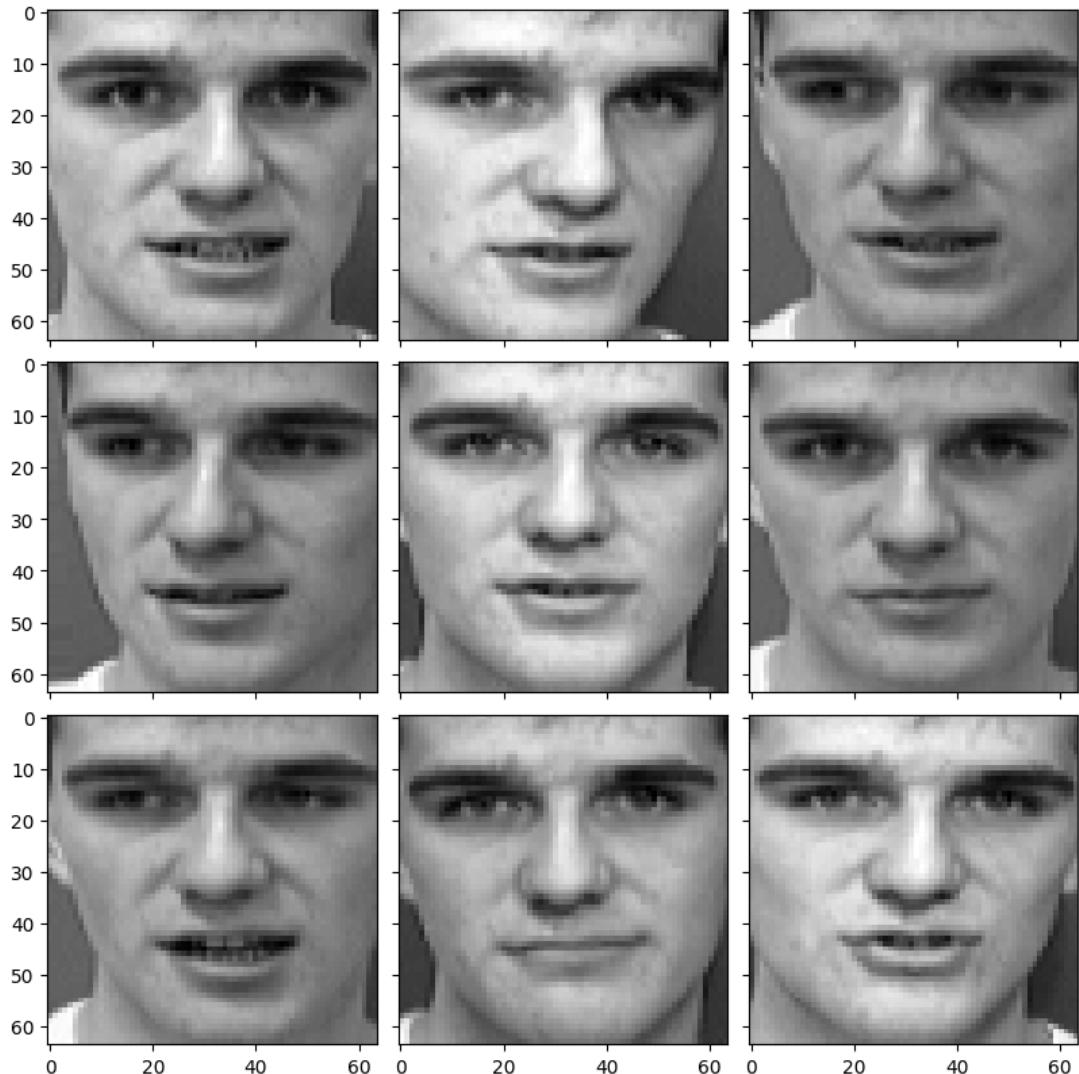
- Dabei kann Ihnen [dieser Befehl](#) und [jener Befehl](#) hilfreich sein. Wenn Sie `cmap=gray` verwenden, dann wird das Gesicht in Graustufen dargestellt.

```
[3]: plt.imshow(data[296].reshape((64, 64)), cmap="gray")  
plt.axis("off")  
plt.show()
```



(3) Visualisieren Sie in einem **3x3 Plot** die Gesichter 291 bis 299.

```
[4]: fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, constrained_layout=True, u
    ↪figsize=(8, 8))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(data[291 + i].reshape((64, 64)), cmap="gray")
plt.show()
```



- (4) Nehmen Sie die Folien zur heutigen Vorlesung zur Hand und schlagen Sie nach, wie die PCA definiert ist.
- Überlegen Sie sich die Dimensionen der Kovarianzmatrix des Datensatzes und nennen Sie sie hier.

Die Dimension D der Kovarianzmatrix S ist die Anzahl der Features und nicht die Anzahl der Featurevektoren (Samples). $S \in \mathbb{R}^{4096 \times 4096}$

- (5) Bestimmen Sie die **Kovarianzmatrix** `cov_mat` des Datensatzes. Überprüfen Sie, ob die Dimensionen der Kovarianz mit Ihrer Erwartung aus Schritt (4) übereinstimmt.
- Hinweis: Eine typische Fehlerquelle besteht bei diesem Schritt darin, dass die Kovarianzmatrix nicht korrekt berechnet wird. Es ist daher wichtig, dass Sie wissen und überprüfen, ob die Dimensionen der Kovarianzmatrix stimmen.

```
[5]: cov_mat = np.cov(data, rowvar=False)
print(f"cov_mat.shape={cov_mat.shape}")
```

```
cov_mat.shape=(4096, 4096)
```

- (6) Bestimmen Sie nun die Eigenwerte `eig_vals` und Eigenvektoren `eig_vecs` Ihrer Kovarianzmatrix.

```
[6]: eig_vals, eig_vecs = np.linalg.eigh(np.cov(data, rowvar=False))
```

- (7) Ein typischer Fehler ist es, anzunehmen, dass Ihre Eigenwerte sortiert vorliegen. Dies ist im Allgemeinen *nicht* der Fall. Daher sortieren Sie bitte die Eigenwerte in absteigender Größe. Nutzen Sie dafür [diesen Befehl](#), um die Sortierung in eine separate Variable `idx` zu speichern, mit der Sie dann die Sortierung der Eigenwerte vornehmen.

- Beachten Sie, dass der Ihnen angegebene Befehl nicht in absteigender Reihenfolge (*descending order*) sortieren kann. Sortieren Sie also zunächst aufsteigend und kehren Sie dann die Reihenfolge im resultierenden Array um.
- Falls Sie glauben, dass Ihre Eigenwerte schon sortiert vorliegen, ohne dass Sie sortieren müssen, prüfen Sie dies nach, indem Sie sich die Variable `idx` anschauen.

```
[7]: idx = np.argsort(eig_vals)[::-1]
```

- (8) Sortieren Sie nun die assoziierten Eigenvektoren, die Sie im Array `eig_vecs` gespeichert hatten.
- Beachten Sie: Die Eigenvektoren liegen als Spalten in `eigen_vecs` vor, *nicht* als Zeilen. Dies können Sie auch in der [Dokumentation](#) nachschlagen. Das Verwechseln von Zeilen und Spalten ist eine typische Fehlerquelle.
 - Nutzen Sie `idx`, um die Spalten (also die Eigenvektoren) so zu sortieren, dass sie zu Ihren sortierten Eigenwerten passen. Dazu können Sie [Integer Array Indexing](#) einsetzen. Für Integer Array Indexing müssen Sie Ihr Array `idx` in eine Python Liste umwandeln.

```
[8]: # eig_vals mit den Zeilen Indizes idx sortieren
# eig_vecs mit idx als Spalten Indizes sortieren und danach erst transponieren
eig_vals, eig_vecs = eig_vals[idx], eig_vecs[:, idx]
```

- (9) Schlagen Sie in der Vorlesung nach, wie die *Proportion of Variance Explained* (PVE) definiert ist.
- Bestimmen Sie die PVE als Funktion der PCA-Komponenten und visualisieren Sie sie.
 - Visualisieren Sie ebenfalls die kumulative PVE als Funktion der PCA-Komponenten.
 - Beschreiben Sie kurz den Verlauf der PVE in Ihren beiden Abbildungen. Wie interpretieren Sie diesen Verlauf? (1-2 Sätze)

```
[9]: pve = eig_vals / eig_vals.sum()

fig, ax = plt.subplots(2, 2, figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title("PVE als Funktion der PCA-Komponenten")
```

```

plt.plot(np.arange(len(idx)) + 1, pve)
plt.ylabel("PVE")
plt.xlabel("PCA Komponente")

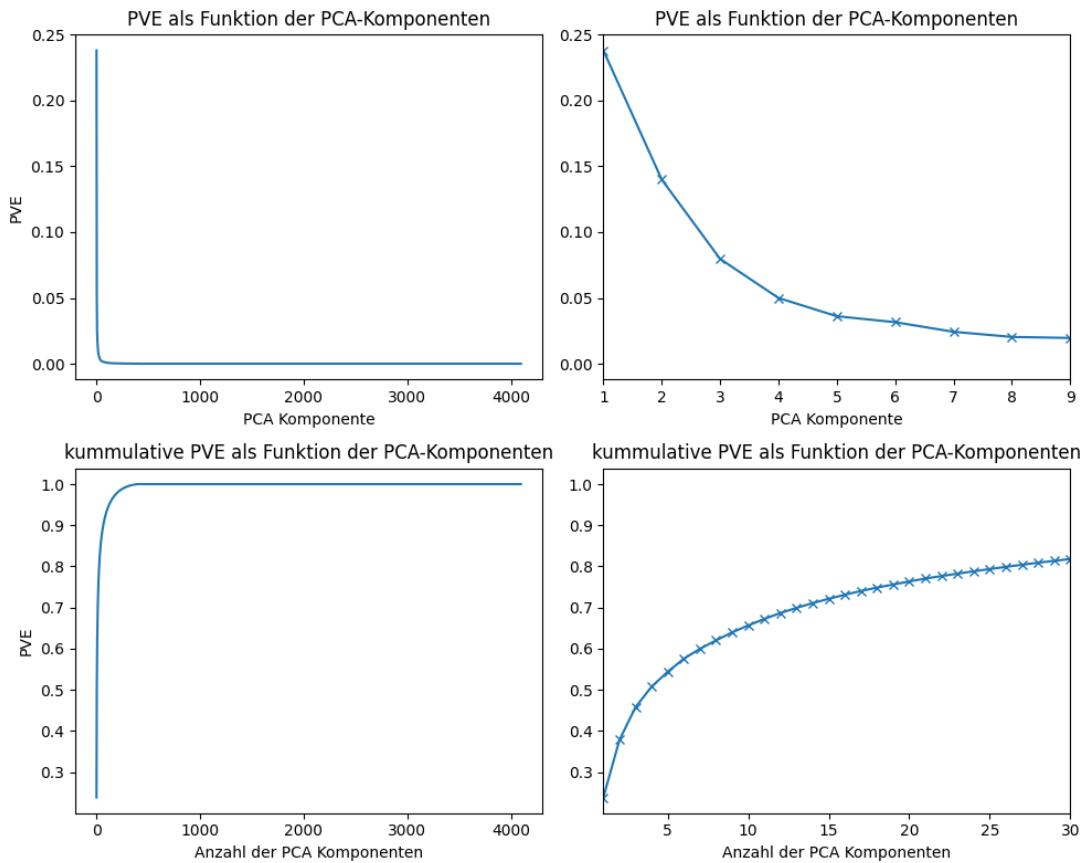
plt.subplot(2, 2, 2)
plt.title("PVE als Funktion der PCA-Komponenten")
plt.plot(np.arange(len(idx)) + 1, pve, marker="x")
plt.xlim(1, 9)
plt.xlabel("PCA Komponente")

plt.subplot(2, 2, 3)
plt.title("kummulative PVE als Funktion der PCA-Komponenten")
plt.plot(np.arange(len(idx)) + 1, np.cumsum(pve))
plt.ylabel("PVE")
plt.xlabel("Anzahl der PCA Komponenten")

plt.subplot(2, 2, 4)
plt.title("kummulative PVE als Funktion der PCA-Komponenten")
plt.plot(np.arange(len(idx)) + 1, np.cumsum(pve), marker="x")
plt.xlim(1, 30)
plt.xlabel("Anzahl der PCA Komponenten")

fig.tight_layout()
plt.show()

```



Die Proportion of Variance Explained (PVE) ist der Bruchteil der Gesamtvarianz der Merkmale, die über die PCA-Komponenten repräsentiert wird.

$$PVE(j) = \frac{\text{Var}(Z_j)}{\text{Var}_{\text{total}}} = \frac{\lambda_j}{\sum_{i=1}^D \lambda_i}$$

Die PVE der ersten PCA Komponente (nach Sortierung) ist sehr hoch und fällt dann rasch ab.

(10) Wir untersuchen nun die Hauptkomponenten der PCA.

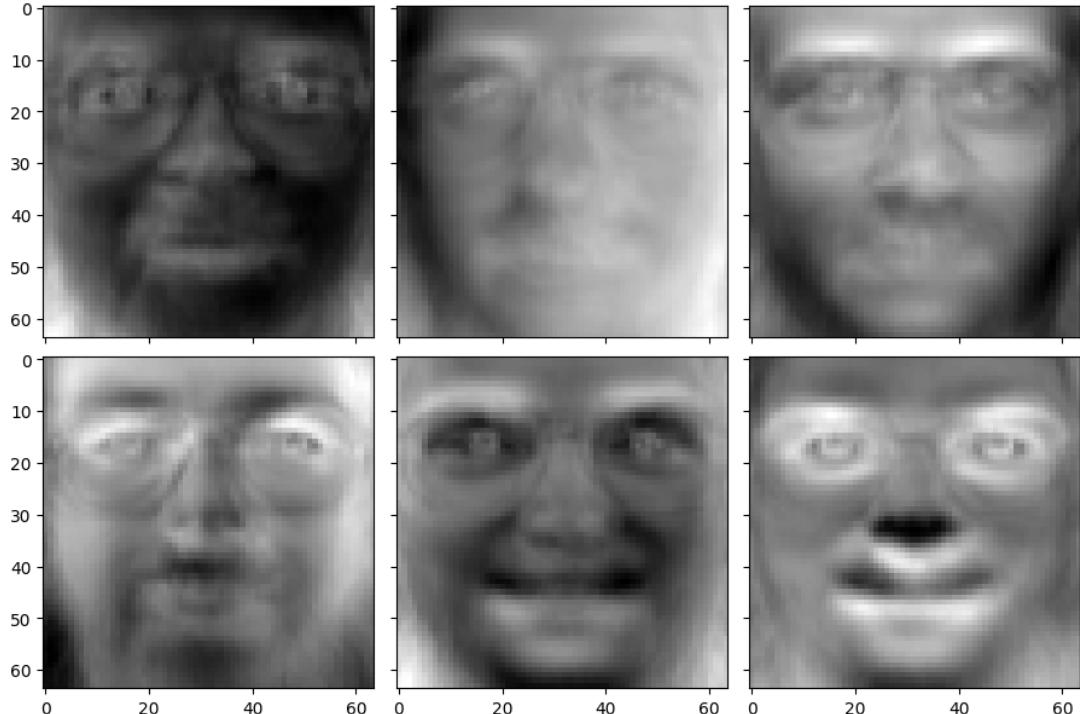
- Visualisieren Sie das erste Eigengesicht (*eigenface*). Dies entspricht der 1. PCA-Komponente, also dem Eigenvektor der Kovarianzmatrix, der dem größten Eigenwert zugeordnet ist. Dieser Eigenvektor wird auch als “erster Eigenvektor” bezeichnet.
- Visualisieren Sie ebenfalls die Eigengesichter 2 bis 5.

```
[10]: fig, ax = plt.subplots(2, 3, sharex=True, sharey=True, constrained_layout=True, figsize=(9, 6))

for i in range(6):
    plt.subplot(2, 3, i + 1)
```

```
eigenface = eig_vecs[:, i]
plt.imshow(eigenface.reshape((64, 64)), cmap="gray")
```

```
plt.show()
```



- (11) Die PCA-Komponenten (Eigengesichter) stellen eine Basis dar, mit der sich die Gesichter im Olivetti-Datensatz darstellen lassen. Ihre Erkenntnisse aus Teilaufgabe (9) deuten darauf hin, dass zur Darstellung der Gesichter nicht alle Basisvektoren notwendig sein werden. Dies werden Sie hier untersuchen.

Seien $\vec{f}_1, \dots, \vec{f}_k$ die PCA-Komponenten der k -größten Eigenwerte, und bezeichne \vec{g} der Vektor, der das Gesicht 296 (0-basierte Zählung) darstellt. Projizieren Sie das Gesicht \vec{g} auf die ersten 25 PCA-Komponenten (Eigengesichter). Sie erhalten dadurch 25 Koeffizienten c_i ,

$$c_i = \vec{f}_i^T(\vec{g} - \vec{m}),$$

wobei \vec{m} der Vektor der Mittelwerte über die Merkmale des gesamten Datensatzes ist. Denn: Vergessen Sie nicht, dass Sie Ihre Daten *vor* der Projektion zentrieren müssen (schlagen Sie die entsprechende Folie in der Vorlesung nach). Dies bedeutet, dass Sie von den Merkmalen (Pixeln) des Gesichts 296 jeweils die Mittelwerte (ermittelt über den ganzen Datensatz) abziehen müssen.

Bemerkung:

- Sie haben hier das Gesicht 296 mithilfe von 25 Koeffizienten beschrieben und damit mit einem Vektor in einem 25-dimensionalen Raum. Dieser Raum ist deutlich kleiner als der ursprüngliche Raum (Dimensionsreduktion!).

```
[11]: data_centered = (data - data.mean(axis=0))
data_projected = data_centered[296][:, np.newaxis].T @ eig_vecs[:, 0:25]
data_projected.shape
```

[11]: (1, 25)

- (12) Sie haben aus Teilaufgabe (11) 25 Koeffizienten vorliegen, mit denen Sie - mithilfe der Eigengesichter - nun das Gesicht rekonstruieren werden. Rekonstruieren Sie das Gesicht aus den 25 Koeffizienten mithilfe der Eigengesichter,

$$\vec{g}' = \left(\sum_{i=1}^k c_i \vec{f}_k \right) + \vec{m},$$

wobei Sie auch hier sich daran erinnern, dass Sie für die Rekonstruktion nun den Mittelwertsvektor \vec{m} , den Sie in Teilschritt (11) abgezogen hatten, wieder aufaddieren müssen.

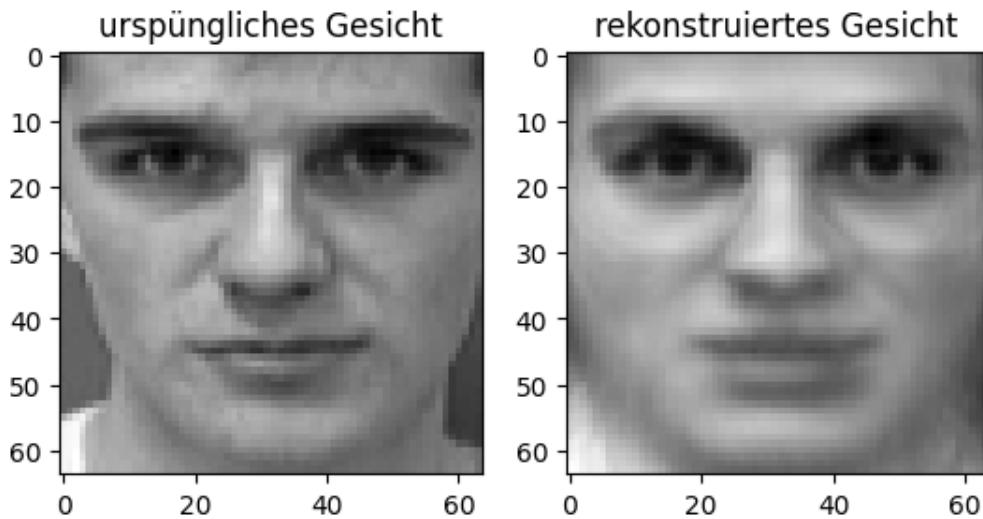
- Visualisieren Sie in einem Plot das ursprüngliche Gesicht (links) sowie das rekonstruierte Gesicht (rechts).

```
[12]: fig, ax = plt.subplots(1, 2)

# ursprüngliches Gesicht
ax[0].imshow(data[296].reshape((64, 64)), cmap="grey")
ax[0].set_title("ursprüngliches Gesicht")

# rekonstruiertes Gesicht
reconstructed_face = np.dot(data_projected, eig_vecs[:, 0:25].T) + data.
    mean(axis=0)
ax[1].imshow(reconstructed_face.reshape((64, 64)), cmap="grey")
ax[1].set_title("rekonstruiertes Gesicht")

plt.show()
```



- (13) Experimentieren Sie mit einer größeren Anzahl von PCA-Komponenten und untersuchen Sie, wie sich die Projektion und die Rekonstruktion verbessert, wenn Sie die ersten 100 oder 200 PCA-Komponenten benutzen.

- Was beobachten Sie, wenn Sie die Anzahl der PCA-Komponenten erhöhen? (1-2 Sätze)

```
[13]: komponente = [5, 10, 25, 30, 50, 75, 100, 150, 200, 300, 400]
```

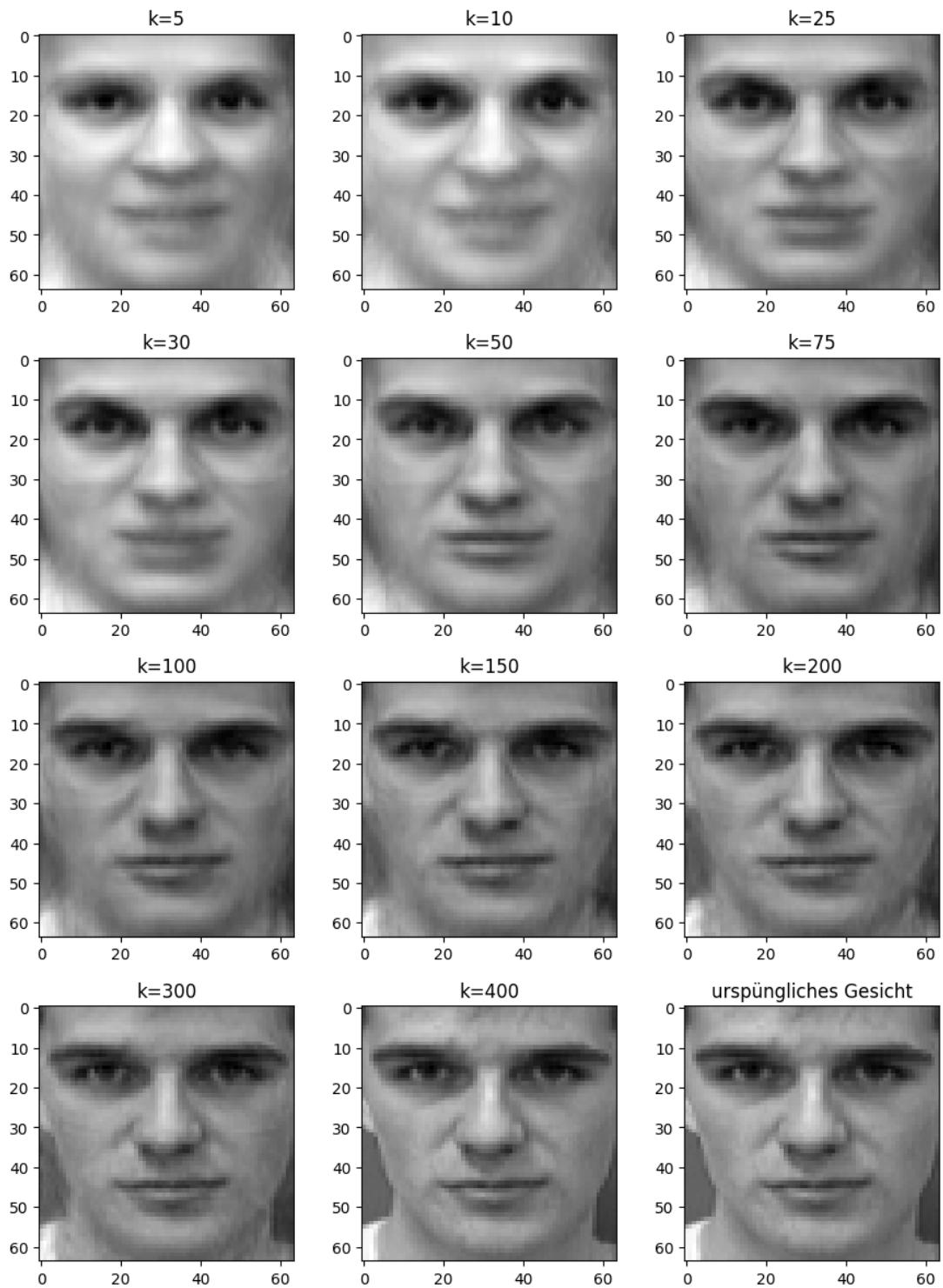
```
fig, ax = plt.subplots(4, 3, figsize=(9, 12))
fig.tight_layout()

for i, k in enumerate(komponente):
    data_projected = data_centered[296][:, np.newaxis].T @ eig_vecs[:, 0:k]
    reconstructed_face = np.dot(data_projected, eig_vecs[:, 0:k].T) + data.mean(axis=0)

    plt.subplot(4, 3, i+1)
    plt.imshow(reconstructed_face.reshape((64, 64)), cmap="grey")
    plt.title(f"k={k}")

plt.subplot(4, 3, 12)
plt.imshow(data[296].reshape((64, 64)), cmap="grey")
plt.title("ursprüngliches Gesicht")

plt.show()
```



Das Gesicht wird immer besser erkennbar.

(14) Nehmen Sie an, wir würden PCA als Bildkompressionsmethode einsetzen. Überdenken Sie folgendes Gedankenexperiment: Wir projizieren jedes Gesicht auf die 200 größten PCA-Komponenten (Eigengesichter) und erhalten jeweils 200 Koeffizienten pro Gesicht. Wir speichern nun diese Koeffizienten sowie die 200 PCA-Komponentenvektoren in einer Datei ab.

- Wie groß (in Megabytes) ist der Originaldatensatz, wenn wir zur Speicherung jedes Pixels 8 Bytes nutzen?
- Wie groß (in Megabytes) wäre der projizierte Datensatz, der lediglich die 200 Eigengesichter, die 200 Koeffizienten pro Gesicht sowie den Mittelwertsvektor (\vec{m}) enthält? Wir nehmen auch in diesem Fall an, dass jedes Pixel und jeder Koeffizient 8 Bytes belegt.
- Im Datensatz sind 400 Gesichter erfasst, welche mit jeweils 4096 Pixeln und 8 Byte pro Pixel eine Größe von $400 \cdot 4096 \cdot 8$ Byte = 13107200 Byte oder rund 13.1 MB (12.5 MiB) belegen würden.
- Nach Projektion werden nur noch 200 Koeffizienten pro Gesicht benötigt $400 \cdot 200 \cdot 8$ Byte = 640kB, was deutlich weniger ist, als zuvor. Jedoch werden auch die 200 Eigengesichter und einen Mittelwertvektor benötigt $(200 + 1) \cdot 4096 \cdot 8 = 6586368$ Byte, sodass insgesamt 7226368 Byte oder ungefähr 7.23 MB benötigt werden.

Dies entspricht einer Kompressionsrate von 55 %, bei einem relativ geringen Qualitätsverlust (s.o.).

(15) [Optional] Nutzen Sie [ipython Widgets](#), um eine interaktive Visualisierung der Gesichter des Datensatzes sowie ihrer Rekonstruktion zu erstellen: Legen Sie zwei Slider an: Mit dem ersten Slider können sie die Nummer des zu visualisierenden Gesichtes im Datensatz einstellen. Mit dem zweiten Slider können Sie die Anzahl der PCA-Komponenten für die Projektion und anschließende Rekonstruktion einstellen. Das Widget soll das ursprüngliche Gesicht sowie das rekonstruierte Gesicht darstellen.

```
[14]: from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

def show_face(face: int, pca_komponenten: int):
    data_projected = data_centered[face][:, np.newaxis].T @ eig_vecs[:, 0:pca_komponenten]
    reconstructed_face = np.dot(data_projected, eig_vecs[:, 0:pca_komponenten].T) + data.mean(axis=0)

    plt.subplot(121)
    plt.imshow(data[face].reshape((64, 64)), cmap="grey")
    plt.title("ursprüngliches Gesicht")

    plt.subplot(122)
    plt.imshow(reconstructed_face.reshape((64, 64)), cmap="grey")
    plt.title(f"k={pca_komponenten}")
    plt.show()

interact(show_face, face=widgets.IntSlider(min=0, max=data.shape[0] - 1, step=1, value=296), pca_komponenten=widgets.IntSlider(min=1, max=500, step=1, value=10))
```

```
interactive(children=(IntSlider(value=296, description='face', max=399), IntSlider(value=10, description='pca_...'))
```

[14]: <function __main__.show_face(face: int, pca_komponenten: int)>

1.0.3 6.2 Kriminalitätsraten (EDA mithilfe der PCA)

In dieser Übung werden wir einen klassischen Datensatz untersuchen, der aus den USA stammt. Es handelt sich um eine Zusammenstellung der Kriminalitätsraten in verschiedenen Staaten der USA sowie um den Prozentsatz der Bevölkerung, der in urbanen Regionen lebt (*UrbanProp*) aus dem Jahr 1975. Bei den Kriminalitätsraten interessieren uns Morde (*murder*), Körperverletzungen (*assault*) sowie Vergewaltigungen (*rape*) pro 100000 Einwohner, aufgeschlüsselt nach US-Staaten. Der Datensatz stammt aus dem *World Almanac and Book of facts 1975* (Kriminalitätsraten) und den *Statistical Abstracts of the United States 1975* (Urbane Regionen).

Ihre Daten

- Sie finden den Datensatz, den Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

- (1) Importieren Sie den Datensatz und verschaffen Sie sich einen Überblick darüber, welche Merkmale (Features) Ihr Datensatz aufweist.

[15]:

```
import pandas as pd
df = pd.read_csv("USArrests.csv")
print(df.head())
```

	Unnamed: 0	Murder	Assault	UrbanPop	Rape
0	Alabama	13.2	236	58	21.2
1	Alaska	10.0	263	48	44.5
2	Arizona	8.1	294	80	31.0
3	Arkansas	8.8	190	50	19.5
4	California	9.0	276	91	40.6

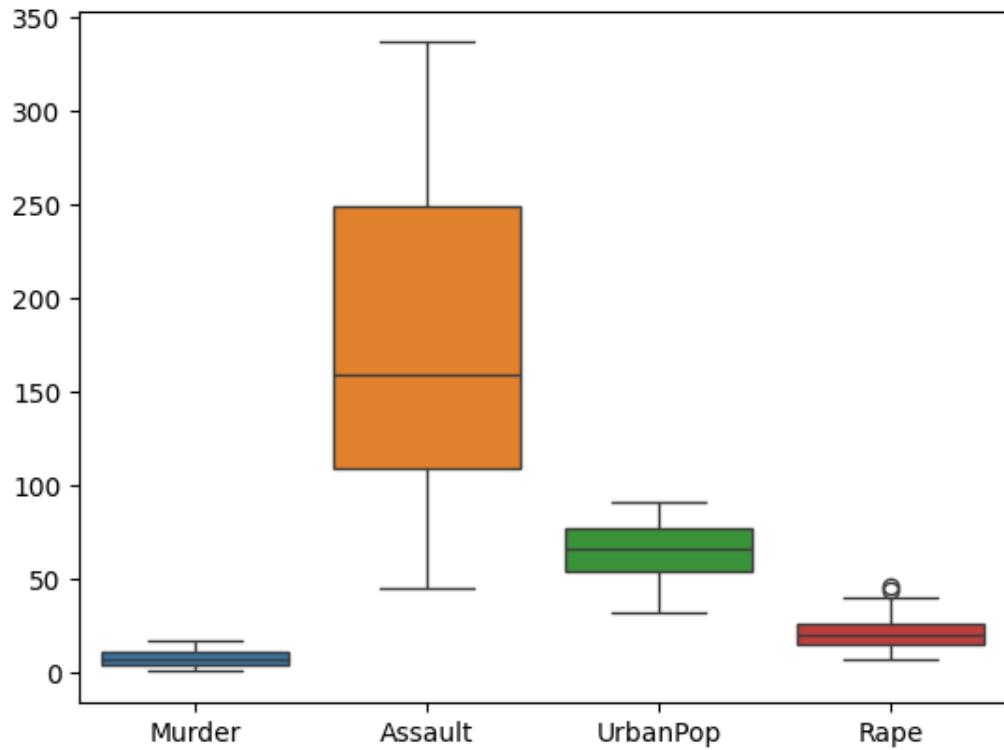
- (2) Führen Sie eine explorative Analyse durch (mit Mitteln der deskriptiven Statistik, Visualisierung und Werkzeugen zur Ermittlung von Zusammenhängen (Korrelationen)). Schreiben Sie (1-3 Sätze) Ihre Befunde auf.

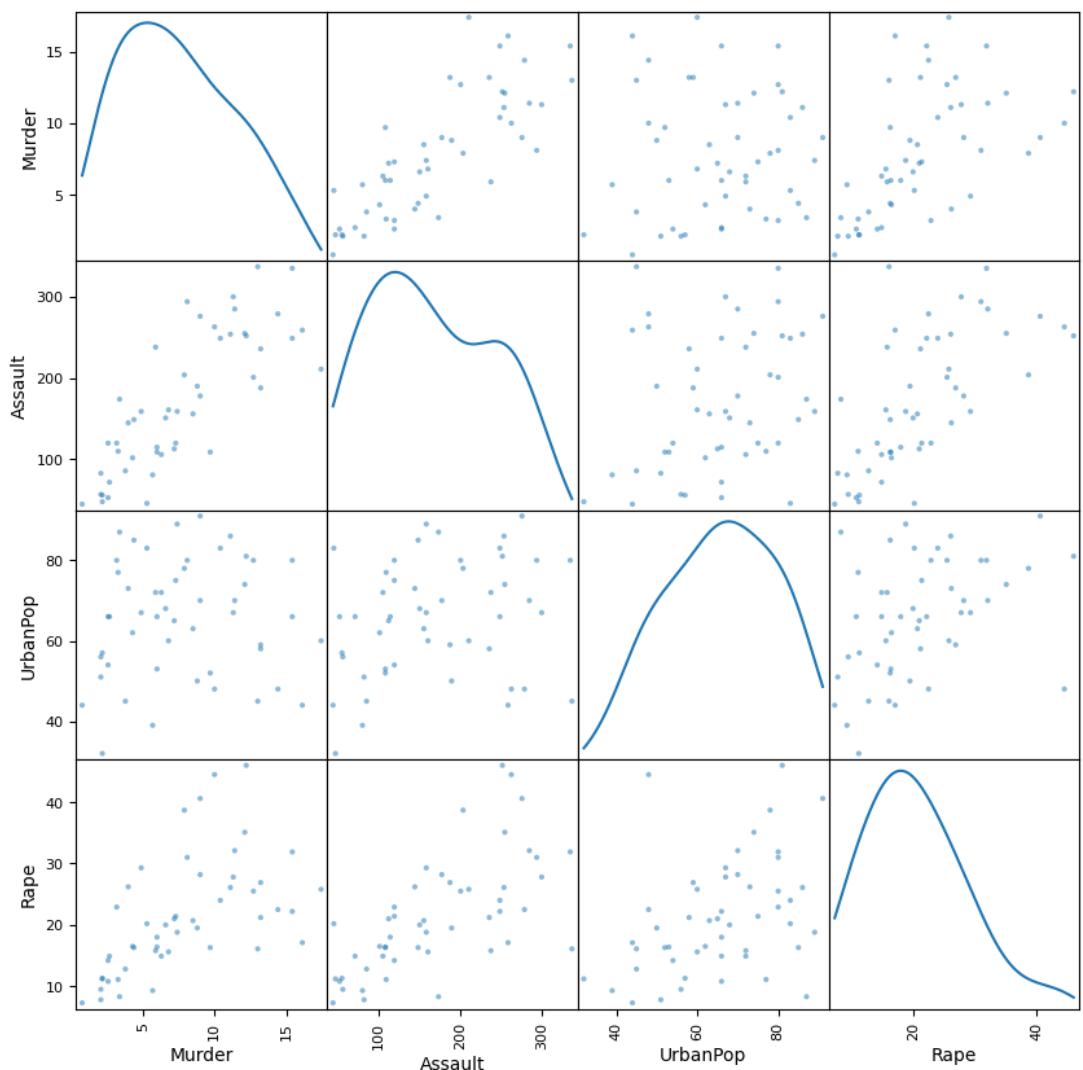
[16]:

```
df.describe()
```

	Murder	Assault	UrbanPop	Rape
count	50.00000	50.00000	50.00000	50.00000
mean	7.78800	170.76000	65.54000	21.23200
std	4.35551	83.337661	14.474763	9.366385
min	0.80000	45.00000	32.00000	7.30000
25%	4.07500	109.00000	54.50000	15.07500
50%	7.25000	159.00000	66.00000	20.10000
75%	11.25000	249.00000	77.75000	26.17500
max	17.40000	337.00000	91.00000	46.00000

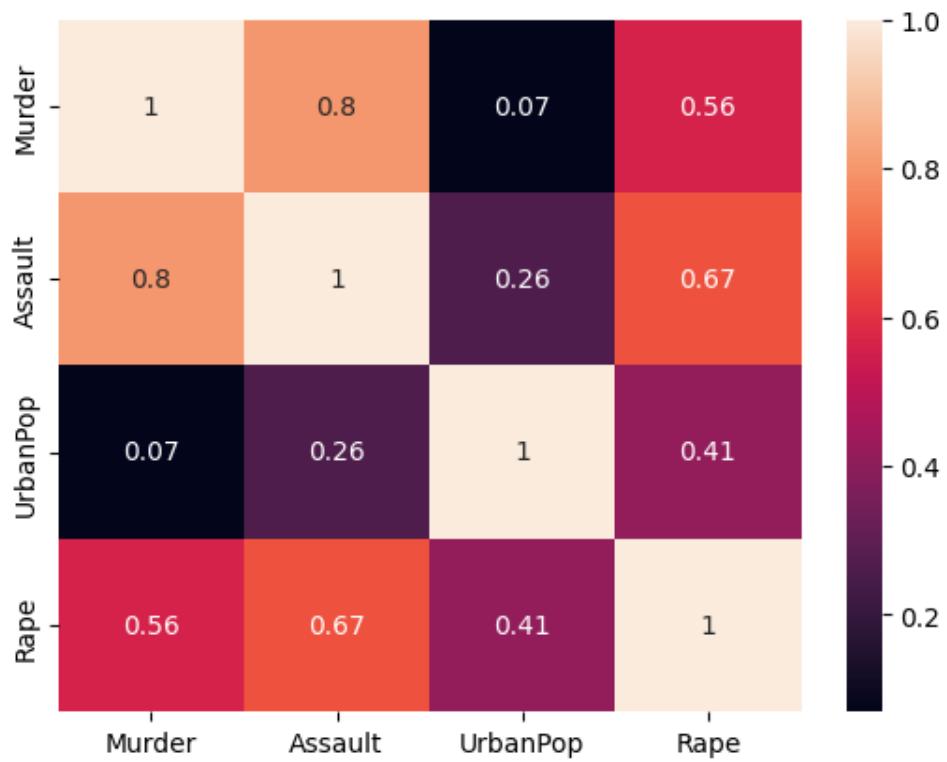
```
[17]: import seaborn as sns  
sns.boxplot(df)  
plt.show()  
  
pd.plotting.scatter_matrix(df, figsize=(10, 10), diagonal="kde")  
plt.show()
```





```
[18]: sns.heatmap(df.corr(numeric_only=True), annot=True, )
```

```
[18]: <Axes: >
```



```
[19]: df[['Murder', 'Assault', 'UrbanPop', 'Rape']].corr()
```

```
[19]:      Murder   Assault  UrbanPop     Rape
Murder  1.000000  0.801873  0.069573  0.563579
Assault  0.801873  1.000000  0.258872  0.665241
UrbanPop  0.069573  0.258872  1.000000  0.411341
Rape    0.563579  0.665241  0.411341  1.000000
```

Die Features *Murder* und *Assault* sind stark linear korreliert (Pearsons Korrelationskoeffizient $r \geq 0.8$). Bei *Rape* besteht eine lineare Korrelation zu *Assault* und *Murder* von mittlerer Stärke.

(3) Ermitteln Sie die Varianz der Merkmale. Welche Unterschiede stellen Sie fest? (1-2 Sätze).

```
[20]: df.describe().loc['std']
```

```
[20]: Murder      4.355510
Assault     83.337661
UrbanPop    14.474763
Rape       9.366385
Name: std, dtype: float64
```

Das Merkmal *Assault* hat eine deutlich höhere Varianz als die übrigen Merkmale.

(4) Wir bereiten uns jetzt für eine Dimensionsreduktion per PCA vor. Klären Sie für sich

zunächst: In welchen Skalen wurden die Merkmale gemessen?

Die Merkmale *Assault*, *Rape*, *Murder* sind in Fälle pro 100k Einwohner angegeben und *UrbanPop* ist in Prozent angegeben.

- (5) Welche Vorbereitungsmaßnahme ergibt sich für Sie aus Punkt (4)? Bereiten Sie Ihre Daten entsprechend vor.
- Wie sollten Sie Ihre Daten vor einer PCA transformieren?

```
[21]: df_z = df.set_index('Unnamed: 0')
# Standardisieren der Daten
df_z = (df_z - df_z.mean()) / df_z.std()
```

- (6) Nutzen Sie Ihre Implementierung der PCA aus Aufgabe 6.1 und führen Sie die PCA durch. Ihr Produkt sind die Merkmale in den neuen PCA-Koordinaten.

```
[22]: eig_vals, eig_vecs = np.linalg.eigh(np.cov(df_z, rowvar=False))
idx = np.argsort(eig_vals)[::-1]
eig_vals, eig_vecs = eig_vals[idx], eig_vecs[:, idx]
```

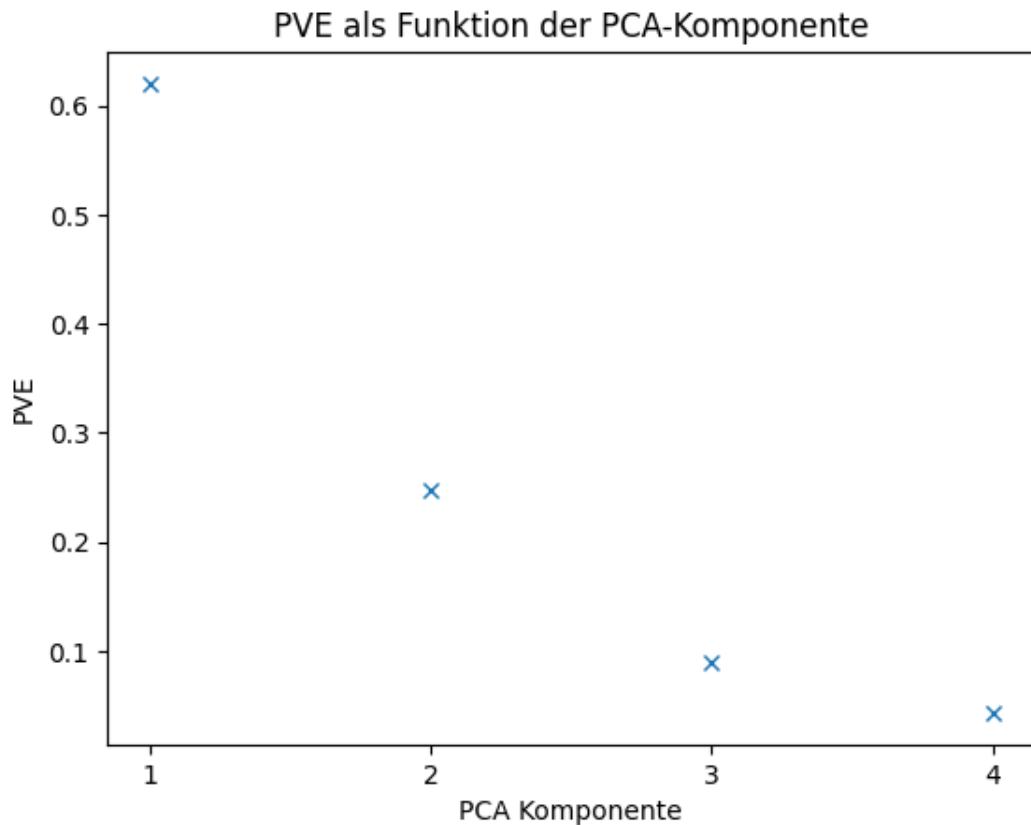
- (7) Implementieren Sie eine Funktion, die Ihnen die *Proportion of Variance Explained* (PVE) für ein gegebenes Merkmal berechnet. Schlagen Sie dazu in der Vorlesung nach, wie die PVE definiert ist.

```
[23]: pve = eig_vals / eig_vals.sum()
pve
```

```
[23]: array([0.62006039, 0.24744129, 0.0891408 , 0.04335752])
```

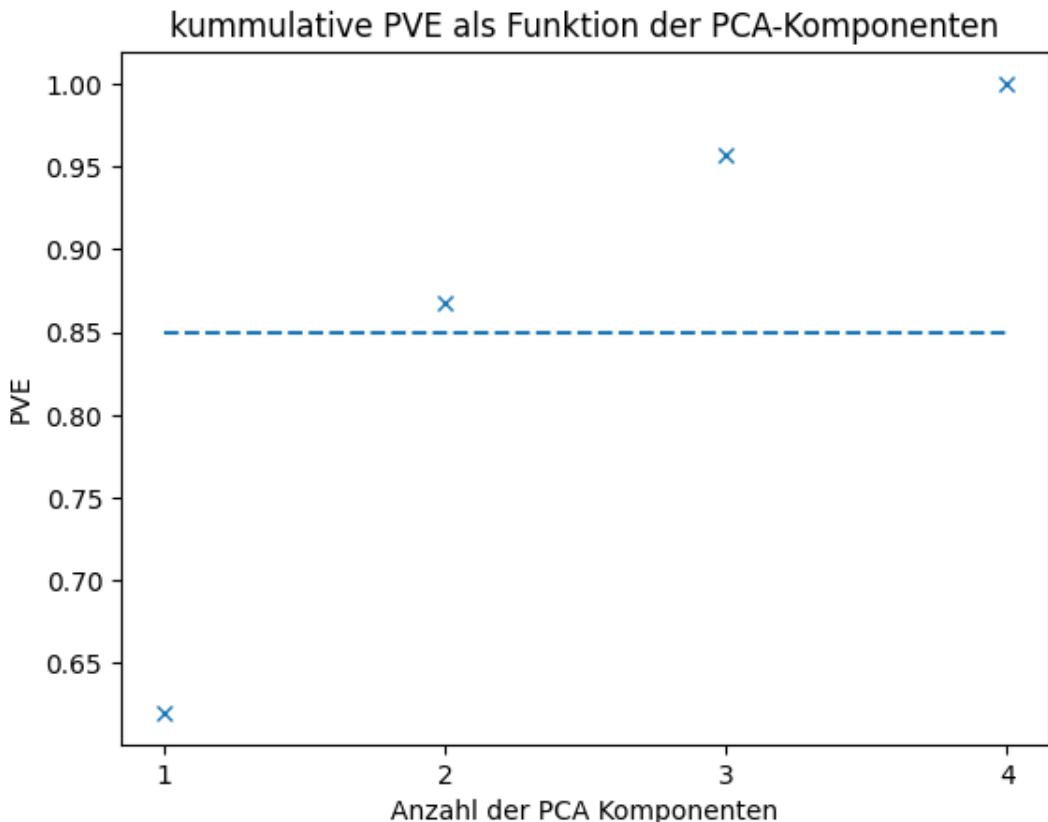
- (8) Visualisieren Sie die PVE als Funktion der PCA-Komponente: Erstellen Sie einen Plot, der die PVE der ersten, zweiten, usw. Komponente der PCA anzeigt.

```
[24]: plt.title("PVE als Funktion der PCA-Komponente")
plt.plot(np.arange(len(idx)) + 1, pve, "x")
plt.ylabel("PVE")
plt.xlabel("PCA Komponente")
plt.xticks(np.arange(len(idx)) + 1)
plt.show()
```



- (9) Ermitteln Sie die Anzahl der ersten PCA-Komponenten, mit deren Hilfe Sie etwas mehr als 85% der Gesamtvarianz erklären können. Notieren Sie sich diese Zahl.

```
[25]: plt.title("kummulative PVE als Funktion der PCA-Komponenten")
plt.plot(np.arange(len(idx)) + 1, np.cumsum(pve), "x")
plt.ylabel("PVE")
plt.xlabel("Anzahl der PCA Komponenten")
plt.xticks(np.arange(len(idx)) + 1)
plt.hlines(y=0.85, xmin=1, xmax=4, linestyles="--")
plt.show()
```



- (10) Für die aus Schritt (9) ermittelte Anzahl an PCA Komponenten betrachten Sie nun die assoziierten Richtungsvektoren: Listen Sie die Komponenten der Richtungsvektoren in einer Tabelle (z.B. einem DataFrame), in der jede Spalte einen Richtungsvektor enthält und die Zeilen den Richtungsvektorkomponenten entsprechen. Beschriften Sie die Zeilen mit den Namen der Merkmale Ihres Datensatzes.

```
[26]: pd.DataFrame(eig_vecs[:, 0:2], df_z.columns, columns=["u1", "u2"])
```

```
[26]:
```

	u1	u2
Murder	-0.535899	0.418181
Assault	-0.583184	0.187986
UrbanPop	-0.278191	-0.872806
Rape	-0.543432	-0.167319

- (11) Betrachten Sie nun Ihre Tabelle aus Schritt 10: Die Einträge der Richtungsvektoren entsprechen dem Gewicht, dem die PCA-Transformation dem entsprechenden Merkmal bei der Projektion auf die PCA-Richtung zuweist. Interpretieren Sie die Gewichte der 1. PCA Komponente: Welche Merkmale scheinen für die 1. PCA-Komponente wichtiger (bzw. gleich wichtig) zu sein, welche weniger wichtig? (1-3 Sätze)

Die Merkmale **Murder**, **Assault** und **Rape** spielen bei der Projektion auf die 1. PCA Komponente

eine fast gleichermaßen wichtige Rolle, während das Merkmal `UrbanPop` eine geringere Bedeutung hat.

- (12) Interpretieren Sie die Gewichte der 2. PCA Komponente: Welche Merkmale scheinen für die 2. PCA-Komponente wichtiger (bzw. gleich wichtig) zu sein, welche weniger wichtig? (1-3 Sätze)

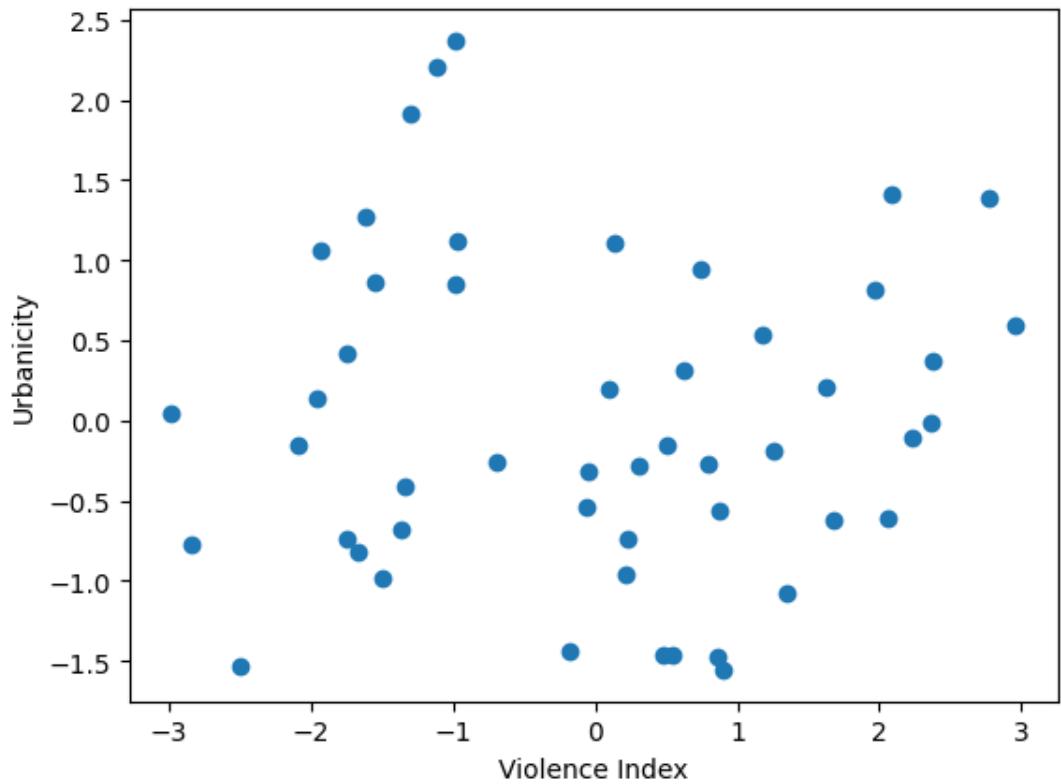
Bei der Projektion auf die 2. PCA Komponente ist besonders das Merkmal `UrbanPop` ausschlaggebend, während das Merkmale `Murder` eine kleine und die Merkmale `Assault` und `Rape` eine noch geringere Bedeutung haben.

- (13) In Anbetracht Ihrer Beobachtungen aus Schritten (11) und (12): Welche Eigenschaften fasst die 1. PCA Komponente zusammen, welche die 2. PCA-Komponente? Welche ungefähren Namen würden Sie den PCA-Komponenten geben?

Die erste Komponente könnte man als `Violence Index` bezeichnen, da hier die `Assault`, `Rape` und `Murder` in einer sehr ähnlichen Gewichtung einfließen. Die zweite Komponente könnte man als `Urbanicity` bezeichnen, da hier `UrbanPop` die Hauptrolle spielt, auch wenn nach der Projektion nicht mehr der reine prozentuale Anteil dargestellt ist.

- (14) Visualisieren Sie die Daten in ihren in Schritt (9) ausgewählten PCA-Komponenten in einem Scatterplot. Beschriften Sie die Achsen mit den provisorischen Namen, die Sie Ihnen in Schritt 13 gegeben haben.

```
[27]: df_projected = df_z @ eig_vecs[:, 0:2]
plt.scatter(df_projected[0], df_projected[1])
plt.xlabel("Violence Index")
plt.ylabel("Urbanicity")
plt.show()
```



1.0.4 6.3 z-scoring und PCA

In diesem Übungsteil werden wir uns noch einmal mit der PCA beschäftigen, allerdings dieses Mal auf konzeptioneller Ebene.

In der Vorlesung hatten wir die Kovarianzmatrix S wie folgt definiert,

$$S = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T,$$

wobei \mathbf{x}_n der Featurevektor des n -ten Datenpunktes und $\bar{\mathbf{x}}$ den Mittelwertsvektor über alle Datenpunkte bezeichnet.

Ihre Aufgaben

- (1) Nutzen Sie die obige Definition von S und berechnen Sie S_{ij} .

$$\begin{aligned}
S &= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \\
S_{ij} &= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_{n,i} - \bar{\mathbf{x}}_i)(\mathbf{x}_{n,j} - \bar{\mathbf{x}}_j) \\
&= \begin{cases} \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_{n,i} - \bar{\mathbf{x}}_i)^2 & \text{für } i = j \\ \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_{n,i} - \bar{\mathbf{x}}_i)(\mathbf{x}_{n,j} - \bar{\mathbf{x}}_j) & \text{für } i \neq j \end{cases} \\
&= \begin{cases} \text{Var}(\mathbf{x}_i) & \text{für } i = j \\ \text{Cov}(\mathbf{x}_i, \mathbf{x}_j) & \text{für } i \neq j \end{cases}
\end{aligned}$$

- (2) Zeigen Sie, dass S symmetrisch ist, also das $S^T = S$. Diese Eigenschaft hatten wir im Beweis der PCA in der Vorlesung genutzt.

Da die Multiplikation invariant bzgl. Vertauschung der Argumente ist (Kommutativgesetz), gilt

$$\begin{aligned}
S_{ij} &= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_{n,i} - \bar{\mathbf{x}}_i)(\mathbf{x}_{n,j} - \bar{\mathbf{x}}_j) \\
&= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_{n,j} - \bar{\mathbf{x}}_j)(\mathbf{x}_{n,i} - \bar{\mathbf{x}}_i) \\
&= S_{ji}
\end{aligned}$$

und damit ist auch $S^T = S$.

- (3) Wir verschieben nun den Mittelwert der n Datenpunkte \mathbf{x}_n , in dem wir jeweils den Vektor $\mathbf{a} \neq \mathbf{0}$ auf alle Datenpunkte addiert, also:

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{a}.$$

Bezeichne $S(\mathbf{x})$ die oben definierte Kovarianzmatrix und $S(\tilde{\mathbf{x}})$ die Kovarianzmatrix der transformierten Merkmale $\tilde{\mathbf{x}}_n$. Sind beide Matrizen unterschiedlich? Falls ja, warum? Falls nein, warum nicht?

Die Kovarianzmatrix ist invariant bzgl. Verschiebungen des Mittelwerts:

$$\begin{aligned}
S(\tilde{\mathbf{x}}) &= \frac{1}{N} \sum_{n=1}^N (\tilde{\mathbf{x}}_n - \bar{\tilde{\mathbf{x}}})(\tilde{\mathbf{x}}_n - \bar{\tilde{\mathbf{x}}})^T \\
&= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n + \mathbf{a} - (\bar{\mathbf{x}} + \mathbf{a}))(\mathbf{x}_n + \mathbf{a} - (\bar{\mathbf{x}} + \mathbf{a}))^T \\
&= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \\
&= S(\mathbf{x})
\end{aligned}$$

- (4) Welche Konsequenzen hat eine Verschiebung der Mittelwerte auf die Richtungen der PCA? Argumentieren Sie ausgehend von Ihren Beobachtungen in Schritt (3). (1-3 Sätze)

Die Verschiebung der Mittelwerte um den Vektor \mathbf{a} hat keine Auswirkung auf die Kovarianzmatirx und damit auch nicht auf die Richtungen der PCA, jedoch haben die Daten nach PCA Transformation einen Offset, wenn sie zuvor nicht zentriert wurden.

- (5) In der Vorlesung wurde empfohlen, die Daten vor einer PCA zu zentrieren, sodass alle Merkmale Mittelwert 0 haben. Welche Konsequenzen hat dies für die Richtungen der PCA? Orientieren Sie sich an Ihrer Antwort in Schritt (4).

Die Zentierung hat keine Auswirkung auf die Richtungen der PCA.

- (6) Betrachten Sie zwei Szenarien: (a) Sie zentrieren Ihre Datenpunkte vor der Projektion auf die Richtungen der PCA. (b) Sie zentrieren Ihre Datenpunkte *nicht* vor der Projektion auf die Richtungen der PCA. Werden Sie in beiden Fällen dieselben PCA-Koordinaten Ihrer Datenpunkte erhalten? Falls ja, warum? Falls nein, warum nicht?

Die PCA-Koordinaten werden typischerweise einen Offset haben, wenn die Datenpunkte vor der Projektion nicht zentriert wurden.

- (7) Nehmen wir an, dass Sie Ihre Merkmale standardisieren (also ein z-scoring durchführen). Betrachten Sie nun die Kovarianzmatrix S , wie sie zu Beginn der Aufgabe definiert ist. Diese Matrix S entspricht einer weiteren Matrix, die Sie bereits kennen. Um welche Matrix handelt es sich?

Nach Standardisierung der Kovarianzmatrix S entspricht diese der Pearson-Korrelationsmatrix.

TSRI-7

July 23, 2024

1 Übung 7

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden Sie zunächst Multidimensionales Skalieren (MDS) implementieren und anwenden (Übung 7.1). MDS wird dann der Ausgangspunkt sein, dass Sie eine nichtlineare Dimensionsreduktion mit Isomap (Übung 7.2) durchführen können.

1.0.2 7.1 Städte in Deutschland (Multidimensionales Skalieren)

In dieser Übung werden Sie Multidimensionales Skalieren implementieren und damit die Daten einer Distanzmatrix visualisieren: Sie erhalten die Entfernung (in Autobahnkilometer) zwischen deutschen Großstädten und finden mithilfe von MDS Koordinaten, mit denen Sie die Position dieser Städte visualisieren können.

Ihre Aufgaben

Bearbeiten Sie bitte alle Aufgaben mit *Numpy* und die Visualisierung später mit *Matplotlib*.

(1) Importieren Sie Ihre Daten, indem Sie die unten stehende Code-Zelle ausführen.

```
[1]: import numpy as np
from matplotlib import pyplot as plt

# Matrix der Distanzen (in Autobahnkilometern)
M: np.ndarray = np.array([[0, 548, 289, 576, 586],
                         [548, 0, 493, 195, 392],
                         [289, 493, 0, 427, 776],
                         [576, 195, 427, 0, 577],
                         [586, 392, 776, 577, 0]])
```



```
# Zugehörige Labels
labels = ['Berlin', 'Frankfurt', 'Hamburg', 'Köln', 'München']
```

- (2) Schlagen Sie in den Vorlesungsfolien nach, wie die Multidimensionale Skalierung in einzelnen Schritten durchgeführt wird.

Die Matrix M enthält Distanzen. Aber die Multidimensionale Skalierung erwartet eine Distanzmatrix D, in der die Distanzen quadriert vorliegen. Erzeugen Sie nun die Distanzmatrix D.

[2]: D: np.ndarray = np.multiply(M, M)
D

[2]: array([[0, 300304, 83521, 331776, 343396],
[300304, 0, 243049, 38025, 153664],
[83521, 243049, 0, 182329, 602176],
[331776, 38025, 182329, 0, 332929],
[343396, 153664, 602176, 332929, 0]])

- (3) Erzeugen Sie die Matrix B mithilfe der Matrix D. Dieser Schritt wird auch *Double Centering* (Doppelzentrierung) genannt.

[3]: N: int = D.shape[0]
H: np.ndarray = np.identity(N) - 1/N * np.ones((N, N))
B: np.ndarray = -0.5 * H @ D @ H
B

[3]: array([[107352.64, -75194.86, 70799.94, -75929.16, -27028.56],
[-75194.86, 42561.64, -41359.56, 38550.84, 35441.94],
[70799.94, -41359.56, 117768.24, 4002.14, -151210.76],
[-75929.16, 38550.84, 4002.14, 72565.04, -39188.86],
[-27028.56, 35441.94, -151210.76, -39188.86, 181986.24]])

- (4) Führen Sie nun eine **Eigenwertzerlegung** der Matrix B durch.

- Sortieren Sie die Eigenwerte in absteigender Reihenfolge.
- Sortieren Sie die Eigenvektoren in der Reihenfolge der Eigenwerte. Beachten Sie, dass die Eigenvektoren den Spalten der von Numpy zurückgegebenen Matrix entsprechen.

[4]: eig_vals, eig_vecs = np.linalg.eigh(B)
idx = np.argsort(eig_vals)[::-1]
eig_vals, eig_vecs = eig_vals[idx], eig_vecs[:, idx]

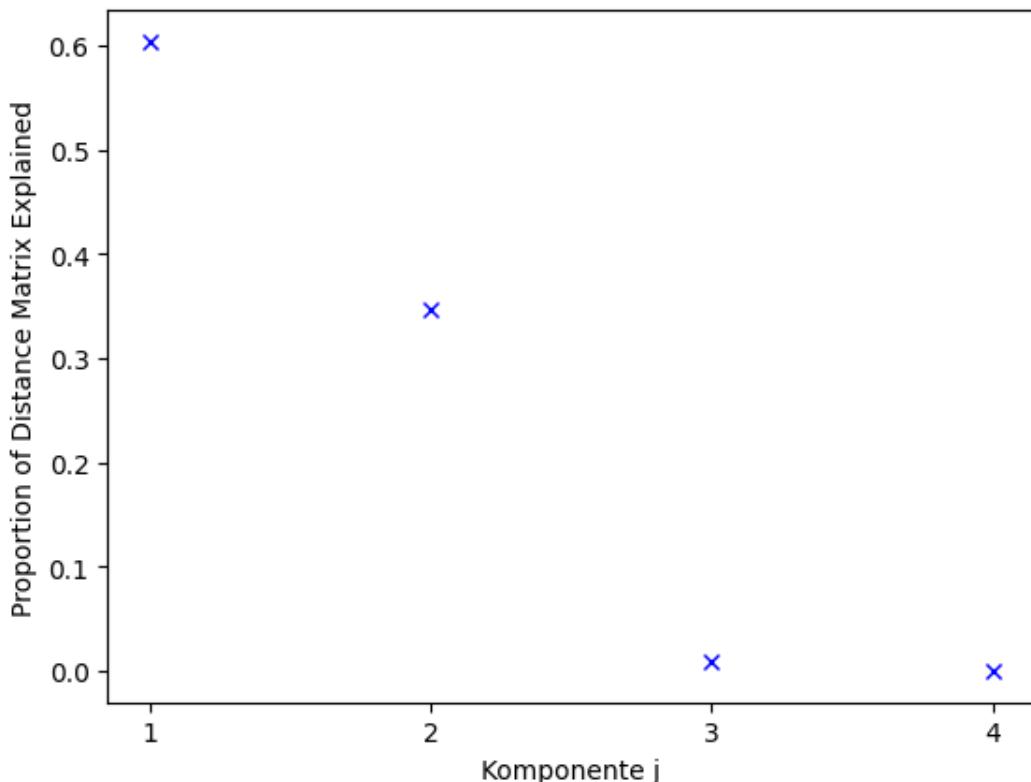
- (5) Berechnen Sie die *Proportion of Distance Matrix Explained* (PDME).

- Können Sie einen Ellbogenverlauf erkennen?
- Wie viele Dimensionen (Komponenten) sollte jeder Datenpunkt haben, dessen Koordinaten Sie rekonstruieren werden?

Proportion of Distance Matrix Explained

$$\text{PDME}(j) = \frac{\lambda_j}{\sum_{i=1}^N |\lambda_i|} \quad \forall j \text{ mit } \lambda_j \geq 0$$

```
[5]: pdme = eig_vals / np.abs(eig_vals).sum()
pdme = pdme[pdme >= 0]
plt.plot(np.arange(len(pdme)) + 1, pdme, "bx")
plt.xticks(np.arange(len(pdme)) + 1)
plt.xlabel("Komponente j")
plt.ylabel("Proportion of Distance Matrix Explained")
plt.show()
```



Es lässt sich ein Ellbogenverlauf erkennen und für die Anzahl der Dimensionen lässt sich klar ablesen, dass diese 2 betragen sollte.

(6) Eliminierung negativer Eigenwerte:

- Haben Sie in Schritt 4 negative Eigenwerte beobachtet? Verwerfen Sie diese mitsamt den assoziierten Eigenvektoren. Am Ende der Übung erkläre ich Ihnen, warum Sie negative Eigenwerte beobachten könnten.

```
[6]: # Reihenfolge beachten!
eig_vecs = eig_vecs[:, eig_vals >= 0]
eig_vals = eig_vals[eig_vals >= 0]
```

(7) Konstruieren Sie mit den Eigenvektoren und Eigenwerten aus Schritt (6) die Datenmatrix \mathbf{X} . Schlagen Sie dazu in der Vorlesung nach, wie dies geht. Wählen Sie für die Konstruktion so

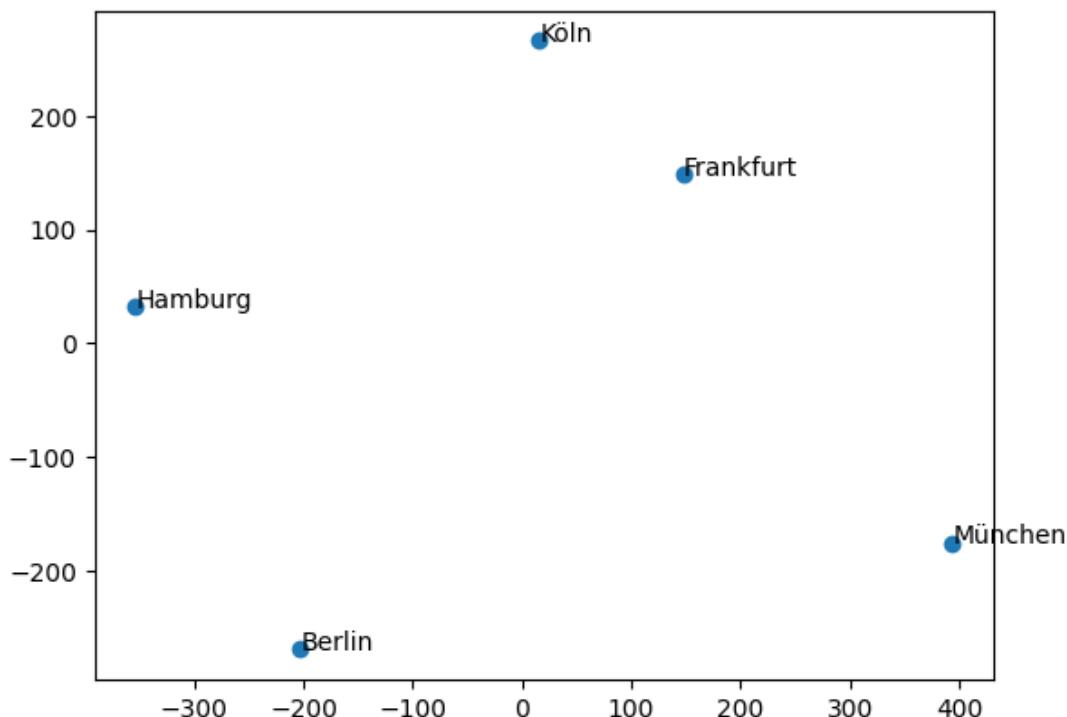
viele Features (Dimensionen) aus, wie Sie in Schritt (5) über die PDME ermittelt haben.

```
[7]: Q: int = 2 # Anzahl der Dimensionen nach Dimensionsreduktion  
X: np.ndarray = np.diag(np.sqrt(eig_vals[:Q])) @ eig_vecs[:, :Q].T  
X
```

```
[7]: array([[-203.18600393, 147.7359528 , -353.99640444, 15.78611067,  
         393.66034489],  
        [-269.50533465, 148.06345119, 32.03109021, 265.94192159,  
         -176.53112834]])
```

- (8) Visualisieren Sie die Daten in den von Ihnen in Schritt (7) ermittelten Koordinaten. Beschriften Sie jeden Datenpunkt mit dem Namen der jeweiligen Stadt, die er repräsentiert.

```
[8]: plt.scatter(*X)  
for i in range(X.shape[1]):  
    plt.annotate(labels[i], X[:, i])  
plt.show()
```



- (9) Ähnelt Ihre Abbildung aus Schritt (8) der geografischen Lage dieser deutschen Städte? Falls Sie Unterschiede bemerken, woran könnten diese Unterschiede liegen? (1-3 Sätze)

Die Abbildung zeigt die Abstände und relative Lage der Städte zueinander (Distanzen invariant unter orthogonalen Transformationen (Rotation, Spiegelung) sowie Koordinatenursprungsverschiebung).

```
[9]: im = plt.imread("germany.png", ) # Quelle: https://upload.wikimedia.org/wikipedia/commons/2/26/EU-Germany.svg
plt.imshow(np.flipud(im), origin="lower")

phi = 1.5 * np.pi
rotation = np.array([
    [np.cos(phi), -np.sin(phi)],
    [np.sin(phi), np.cos(phi)]
])
spiegelung = np.array([
    [-1, 0],
    [0, 1]
])

X_adjusted = X.T * 0.54
X_rotated = (X_adjusted @ spiegelung @ rotation + np.array([[240, 320]])).T

plt.scatter(*X_rotated)
for i in range(X_rotated.shape[1]):
    plt.annotate(labels[i], X_rotated[:, i])
plt.show()
```



Anmerkung zum Auftreten negativer Eigenwerte

Wenn die Matrix B aus euklidischen Distanzen hergeleitet wurde, gilt $B = XX^T$. In diesem Fall ist die Matrix B positiv-semidefinit:

$$\mathbf{a}^T B \mathbf{a} = \mathbf{a}^T X X^T \mathbf{a} = (X\mathbf{a})^T (X\mathbf{a}) \geq 0 \quad \forall \mathbf{a}$$

Damit gilt insbesondere für alle ihre Eigenwerte: $\lambda_i \geq 0$.

Wenn Sie nun für eine Matrix B , die Sie aus einer Matrix D konstruiert haben, negative Eigenwerte beobachten, bedeutet dies, dass die Matrix D keine “euklidische Distanzmatrix” ist, d.h. keine euklidischen Distanzen zwischen Objekten enthält. Dies ist nicht weiter schlimm. In der Praxis werden negative Eigenwerte und assozierte Eigenvektoren verworfen. Daneben wird für die Berechnung der PDME im Nenner die Absolutbeträge der Eigenwerte verwendet.

1.0.3 7.2 Schweizer Rollkuchen (nichtlineare Dimensionsreduktion mit Isomap)

Die PCA erlaubt Ihnen die Extraktion von linearen Strukturen aus hochdimensionalen Räumen. Allerdings liegen in der Realität Daten auf nichtlinearen Mannigfaltigkeiten vor. In dieser Übung lernen Sie Isomap kennen, mithilfe dessen Sie genau solche nichtlineare Strukturen extrahieren können. Das Paper zu Isomap wurde seinerzeit im renommierten Magazin *Science* publiziert. Isomap ist eine Variante des Multidimensionalen Skalierens, allerdings werden die paarweisen Distanzen zwischen den Datenpunkten auf eine andere Art bestimmt als in der klassischen MDS.

Ihre Daten

Sie bearbeiten einen klassischen Datensatz: Den sogenannten *Schweizer Rollkuchen* (swiss roll). Wenn Sie wissen wollen, wie dieser Kuchen aussieht, schauen Sie doch zum Beispiel [hier](#) vorbei.

Ihre Aufgaben

- Nutzen Sie im Nachfolgenden Ihren Code aus Übung 7.1 (MDS), um Isomap zu implementieren.
- (1) Führen Sie die unten stehende Code-Zelle aus, um den Datensatz zu erzeugen und um Bibliotheken zu importieren.

```
[10]: import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.metrics import pairwise_distances
from scipy.sparse.csgraph import shortest_path
from sklearn.neighbors import kneighbors_graph
from mpl_toolkits.mplot3d import Axes3D

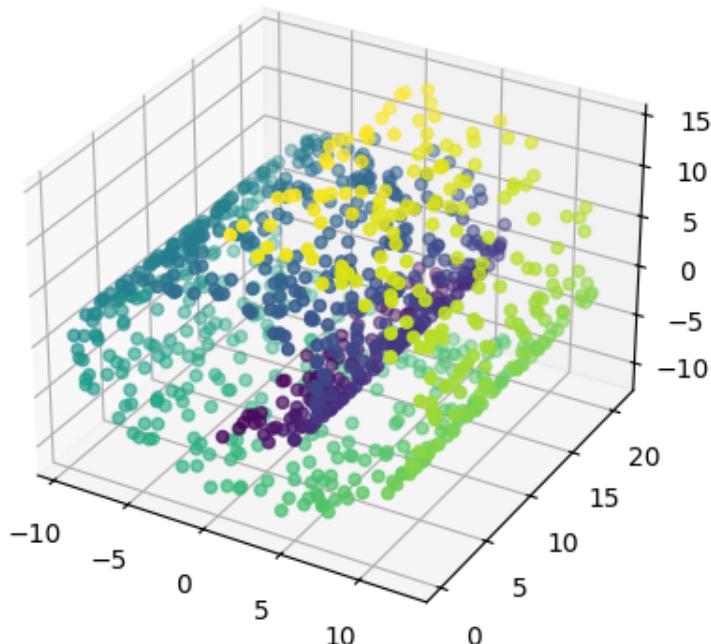
# Ihre Daten
X, t = make_swiss_roll(1000, random_state=100) # X: Koordinaten, t: Variable, die die Farbe jedes Datenpunktes kodiert
```

- (2) Visualisieren Sie zunächst die Daten in einem dreidimensionalen Scatterplot. Achten Sie darauf, dass Sie die Datenpunkte farbkodieren gemäß ihrer Variablen t .

```
[11]: %matplotlib inline
ax = plt.axes(projection='3d')
ax.scatter3D(*X.T, c=t)

plt.title("Schweizer Rollkuchen")
plt.show()
```

Schweizer Rollkuchen



(3) Schlagen Sie in der Vorlesung nach, wie Isomap in einzelnen Schritten durchgeführt wird.

- Bestimmen Sie zunächst eine Matrix $M = (m_{ij})$, wobei m_{ij} der euklidischen Distanz zwischen Datenpunkt i und j entspricht.

Wichtig: Hier sind die Distanzen euklidisch und noch nicht quadriert.

```
[12]: # Distanzmatrix M = (m_{ij}), wobei m_{ij} die euklidische Distanz (nicht ↴ quadriert) zwischen Datenpunkt i und j ist.
M = np.linalg.norm(X[:, np.newaxis] - X[np.newaxis, :], axis=-1)
```

(4) Erzeugen Sie aus M einen K-nächste-Nachbarn Graphen.

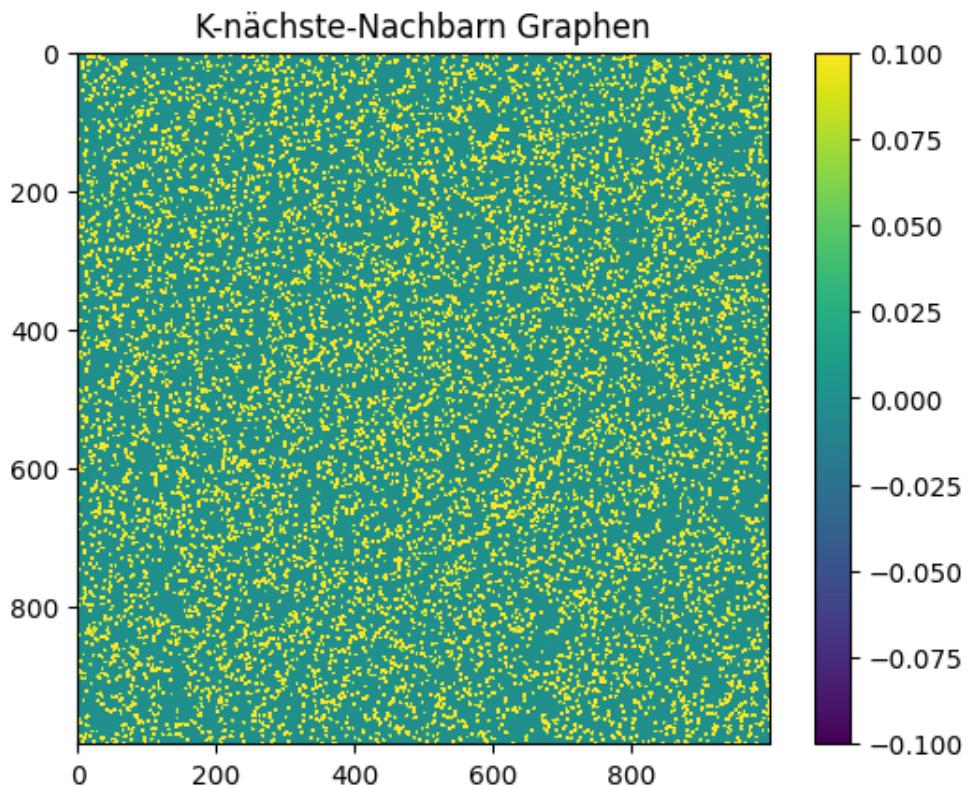
- Nutzen Sie dafür aus scikit-learn den Befehl `kneighbors_graph` und nutzen Sie für die Anzahl nächster Nachbarn zunächst $K = 10$.
- Achten Sie darauf, dass bei der Erstellung des Graphen die euklidischen Abstände zurückgegeben werden. Sie müssen dazu neben M und K einen **zusätzlichen Parameter** der

Funktion anpassen.

- Die entstehende Matrix ist *sparse*. Wenn Sie sich diese Matrix einmal visualisieren wollen, müssen Sie sie in eine gewöhnliche Form [transformieren](#).

```
[13]: from sklearn.neighbors import kneighbors_graph  
knn_graph = kneighbors_graph(M, n_neighbors=10, mode='distance')
```

```
[14]: from scipy.sparse import csr_matrix  
  
plt.imshow(csr_matrix.todense(knn_graph), norm='linear')  
  
plt.colorbar()  
plt.title("K-nächste-Nachbarn Graphen")  
plt.show()
```



(5) Ermittelten Sie die Matrix Y der kürzesten Graphdistanzen für alle Paare von Punkten.

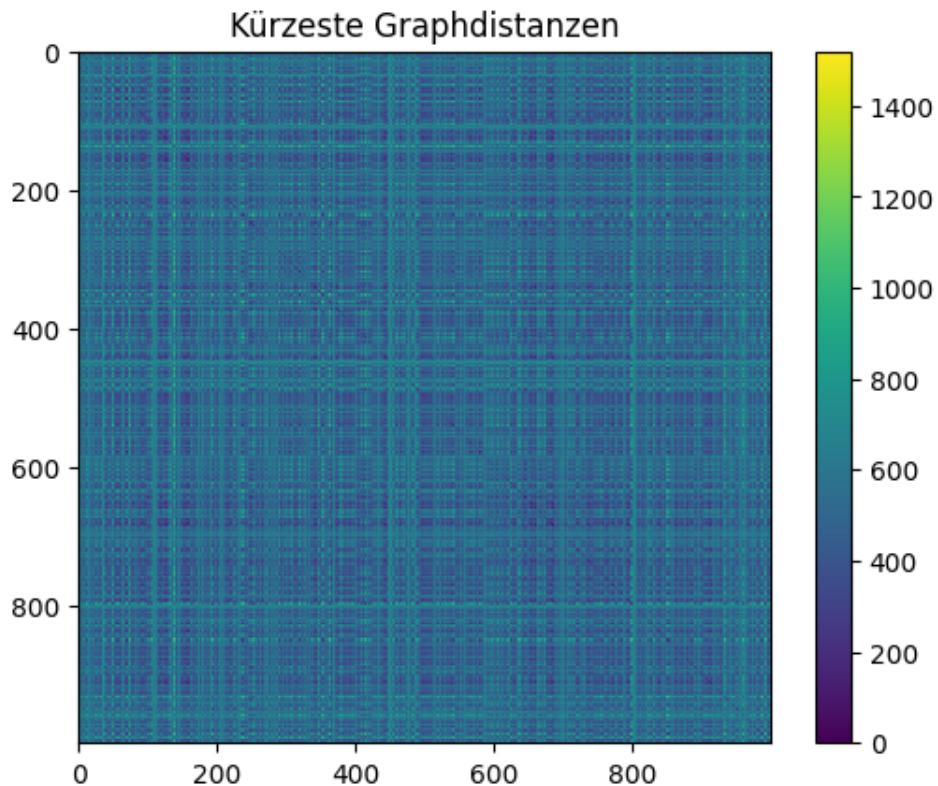
- Nutzen Sie dafür aus `scipy` den Befehl `shortest_path`.
- Wenn Sie wollen, können Sie sich die Matrix dieses Graphen einmal [visualisieren](#).

```
[15]: from scipy.sparse.csgraph import shortest_path  
Y = shortest_path(knn_graph)
```

```

plt.imshow(Y, norm='linear')
plt.colorbar()
plt.title("Kürzeste Graphdistanzen")
plt.show()

```



- (6) Bestimmen Sie für Y eine neue Datenmatrix (Koordinaten) Z mithilfe einer Multidimensionalen Skalierung (MDS). Nutzen Sie dafür Ihren Code aus Übung 7.1.
- Erinnern Sie sich: Der erste Schritt der MDS besteht darin, die Einträge einer Distanzmatrix zu quadrieren, bevor die Matrix B bestimmt wird. Vergessen Sie also nicht, die Einträge der Matrix Y zu quadrieren.

```
[16]: def multidimensional_scaling(M: np.ndarray, Q: int) -> tuple[np.ndarray, np.ndarray]:
    # Distanzmatrix quadrieren
    D: np.ndarray = np.multiply(M, M)

    # Double Centering
    N: int = D.shape[0]
    H: np.ndarray = np.identity(N) - 1/N * np.ones((N, N))
```

```

B: np.ndarray = -0.5 * H @ D @ H

# Eigenwertzerlegung
eig_vals, eig_vecs = np.linalg.eigh(B)
idx = np.argsort(eig_vals)[::-1]
eig_vals, eig_vecs = eig_vals[idx], eig_vecs[:, idx]

# Negative Eigenwerte eliminieren
eig_vecs = eig_vecs[:, eig_vals >= 0]
eig_vals = eig_vals[eig_vals >= 0]

# Neue Datenmatrix Z aus den Eigenwerten und Eigenvektoren mit Q
# Dimensionen konstruieren
Z: np.ndarray = np.diag(np.sqrt(eig_vals[:Q])) @ eig_vecs[:, :Q].T
return Z, eig_vals

Z, eig_vals = multidimensional_scaling(Y, 2)

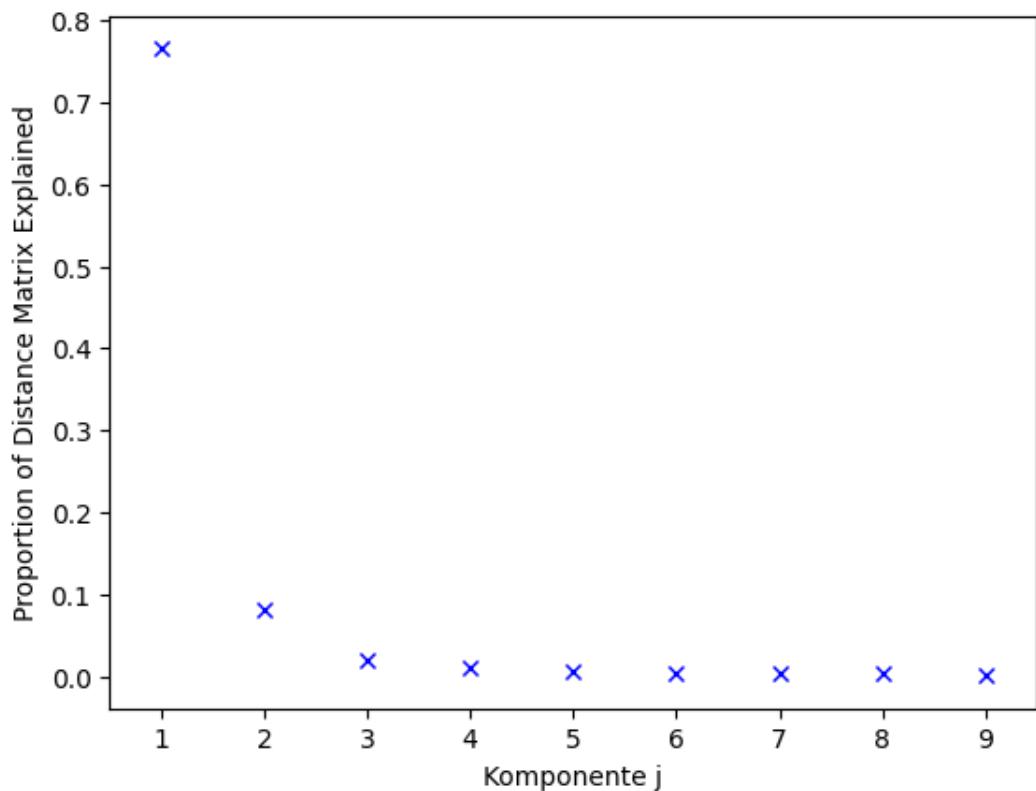
```

- (7) Untersuchen Sie die *Proportion of Distance Matrix Explained* (PDME): Wie hochdimensional wird die nichtlineare Struktur sein, die in den Daten vorhanden ist? (1-2 Worte)

```

[17]: pdme = eig_vals / np.abs(eig_vals).sum()
pdme = pdme[pdme >= 0]
plt.plot(np.arange(len(pdme)) + 1, pdme, "bx")
plt.xticks(np.arange(len(pdme)) + 1)
plt.xlim(0.5, 9.5)
plt.xlabel("Komponente j")
plt.ylabel("Proportion of Distance Matrix Explained")
plt.show()

```

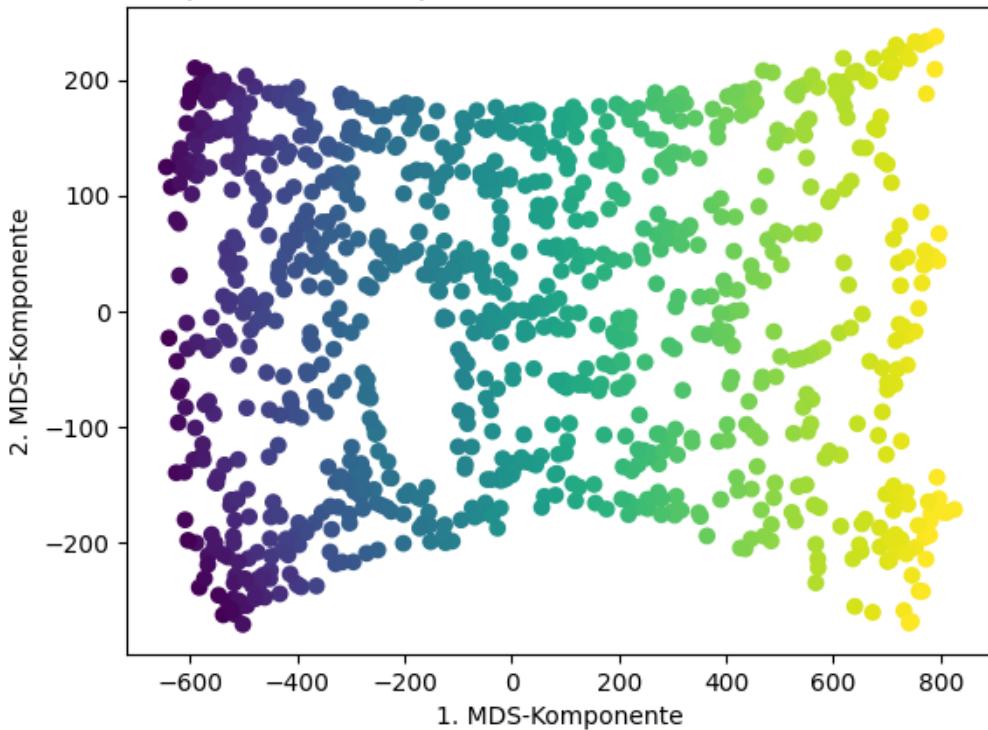


(8) Visualisieren Sie die Daten mithilfe der Koordinaten Z und farbkodieren Sie sie gemäß t .

```
[18]: plt.scatter(*Z, c=t)

plt.title(f"Z-Scatterplot nach Isomap mit 10 Nachbarn (MDS auf {Q} Dimensionen)")
plt.xlabel("1. MDS-Komponente")
plt.ylabel("2. MDS-Komponente")
plt.show()
```

Z-Scatterplot nach Isomap mit 10 Nachbarn (MDS auf 2 Dimensionen)

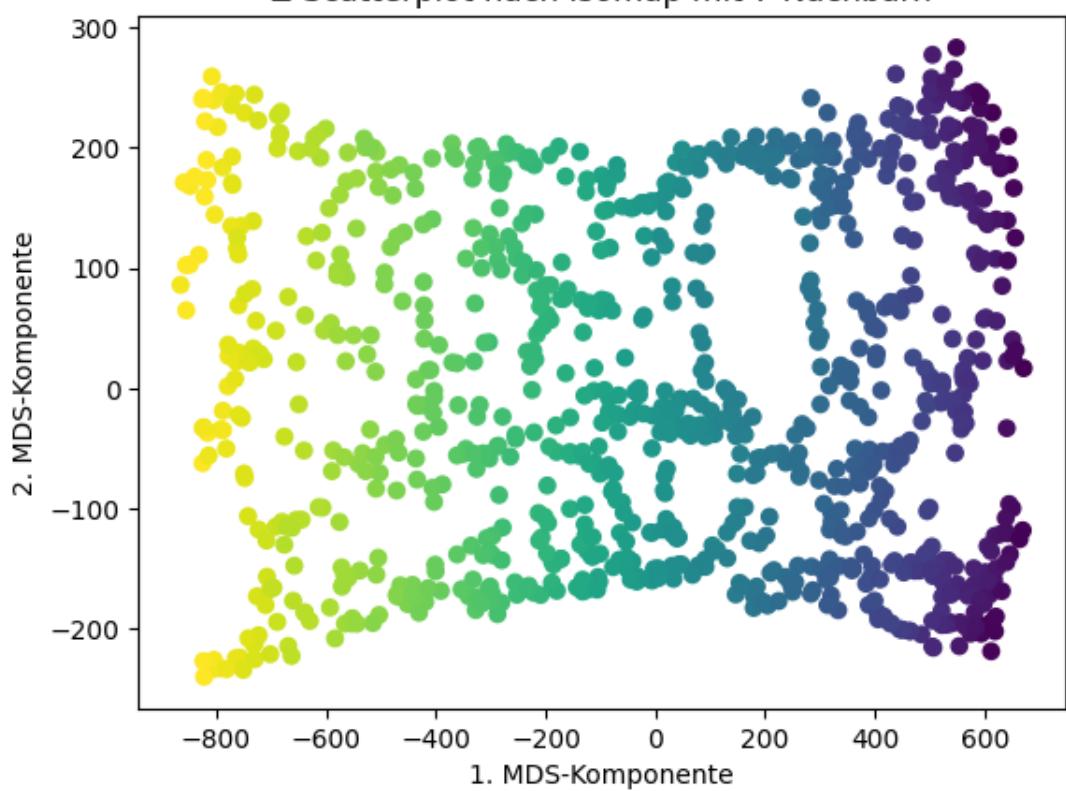


- (9) [Optional] Erzeugen Sie Abbildungen wie in Schritt (8), aber für verschiedene Werte von K für die Konstruktion des Nachbarschaftsgraphen. Was fällt Ihnen auf?

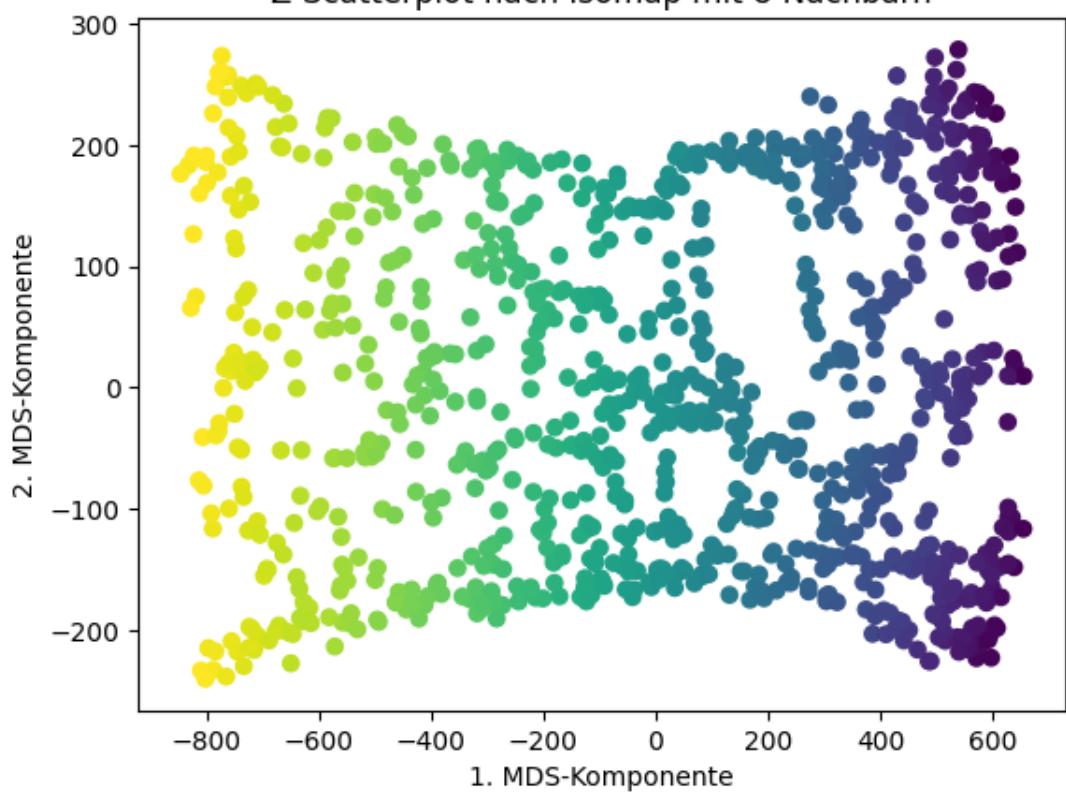
```
[19]: for K in [7, 8, 9, 10, 20, 30]:
    Y = shortest_path(kneighbors_graph(M, n_neighbors=K, mode='distance'))
    Z, _ = multidimensional_scaling(Y, 2)

    plt.scatter(*Z, c=t)
    plt.title(f"Z-Scatterplot nach Isomap mit {K} Nachbarn")
    plt.xlabel("1. MDS-Komponente")
    plt.ylabel("2. MDS-Komponente")
    plt.show()
```

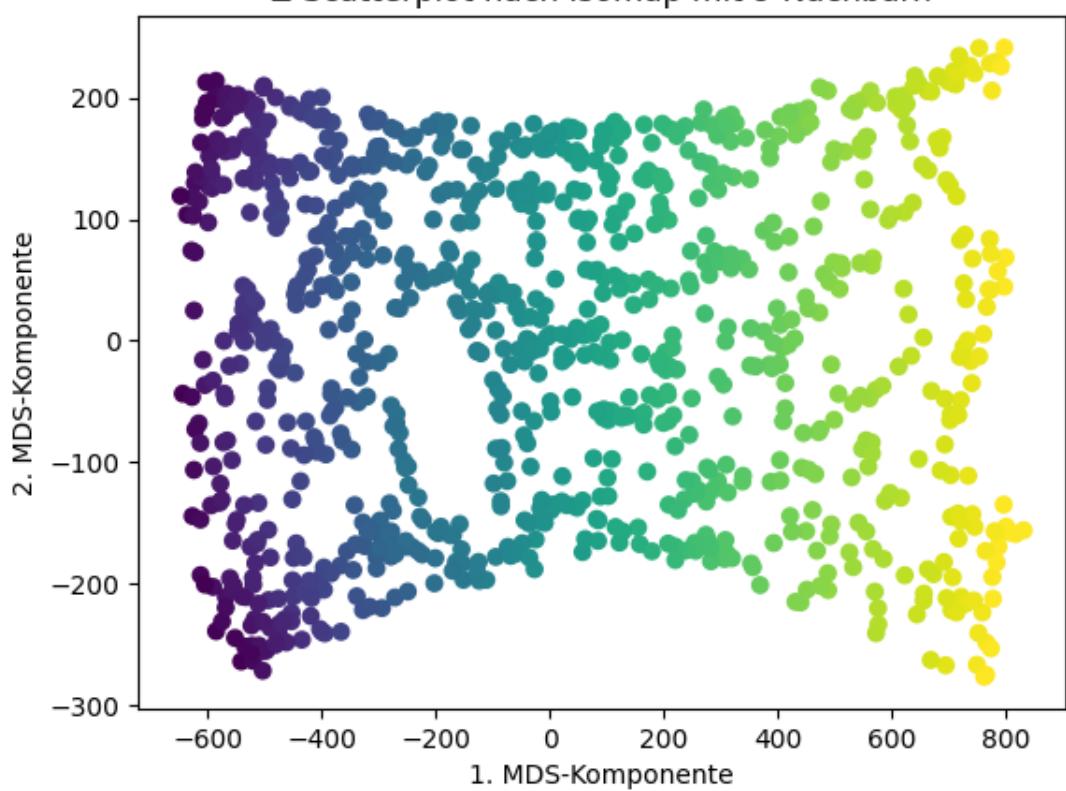
Z-Scatterplot nach Isomap mit 7 Nachbarn



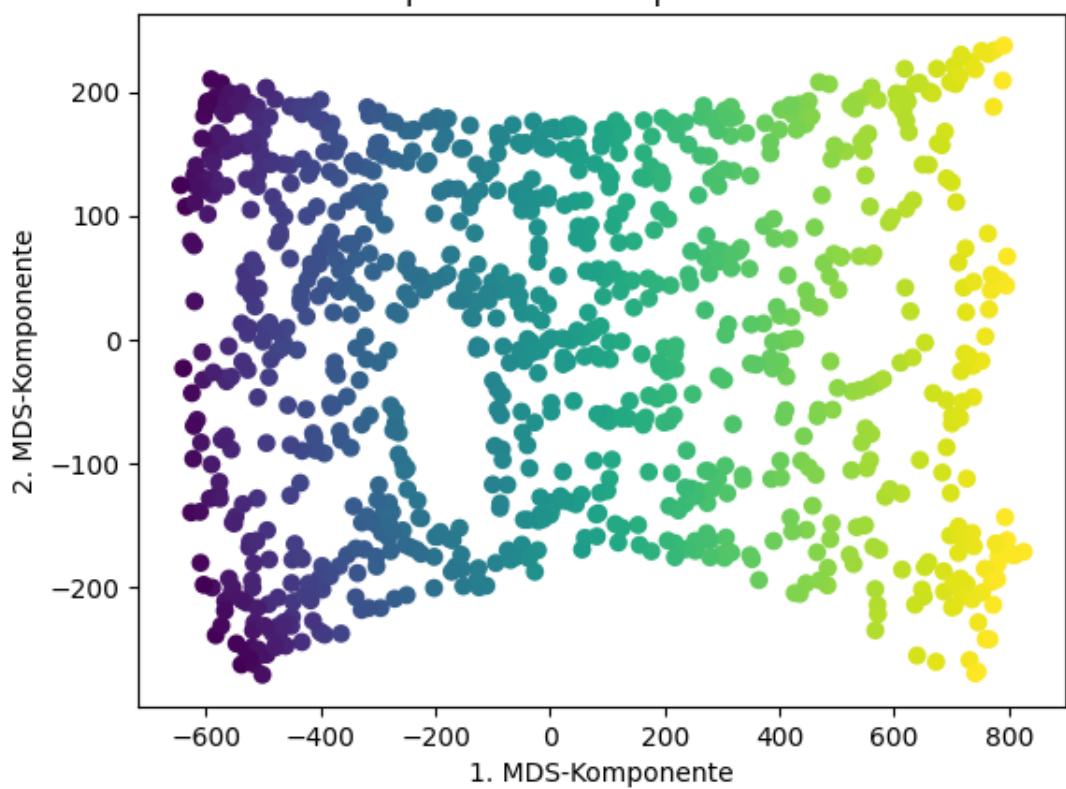
Z-Scatterplot nach Isomap mit 8 Nachbarn



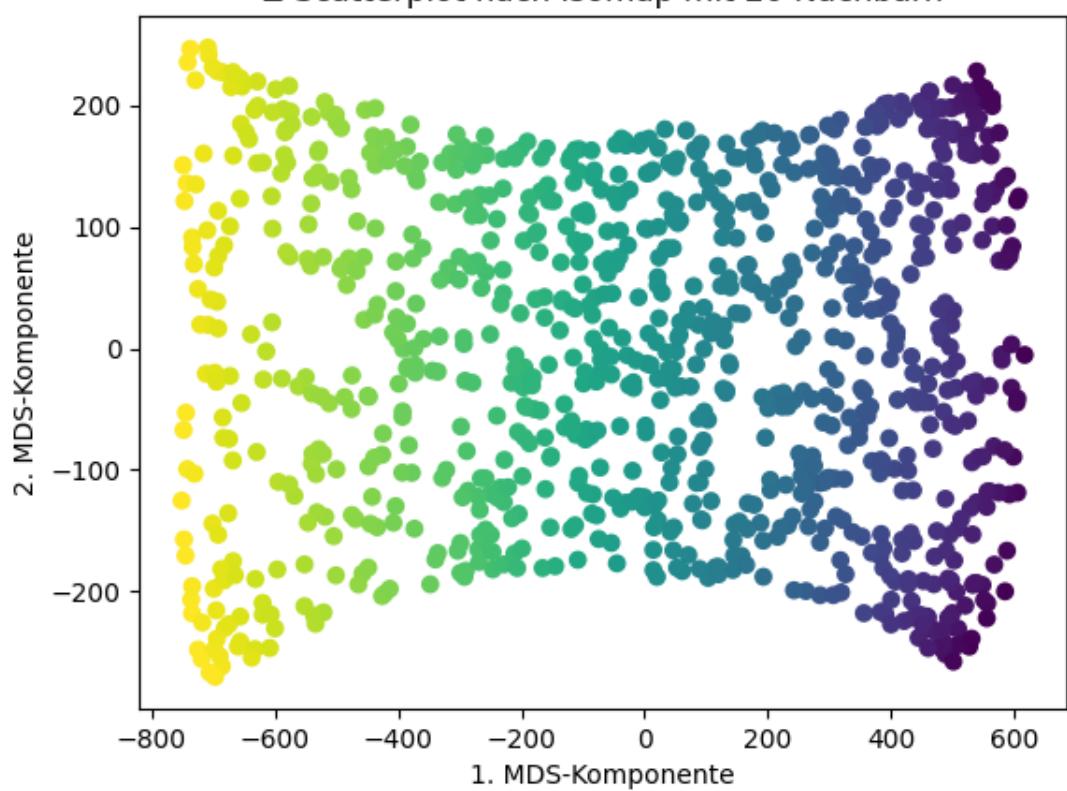
Z-Scatterplot nach Isomap mit 9 Nachbarn

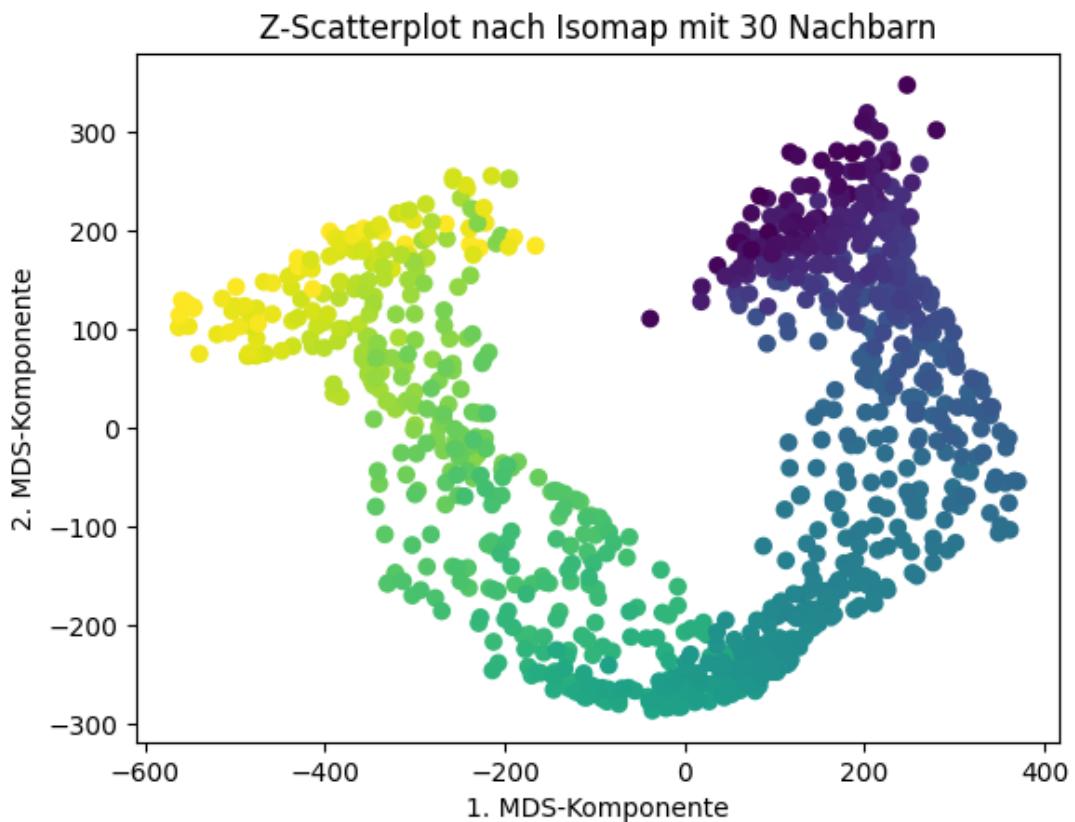


Z-Scatterplot nach Isomap mit 10 Nachbarn



Z-Scatterplot nach Isomap mit 20 Nachbarn

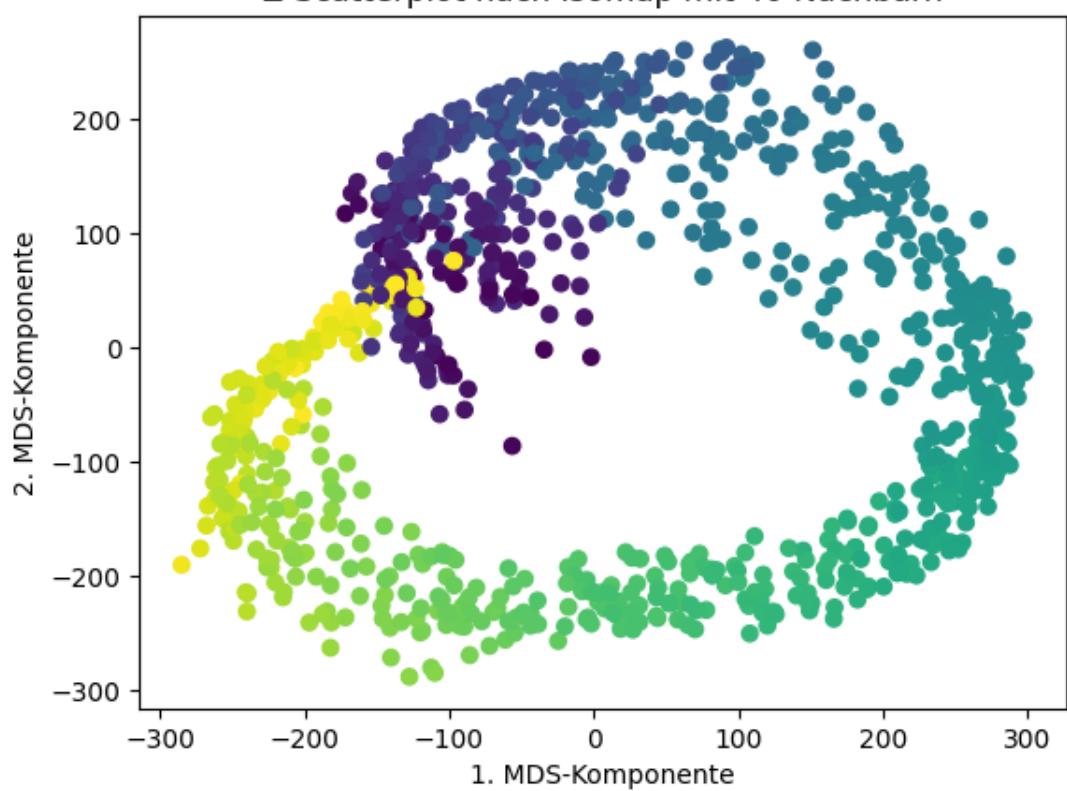




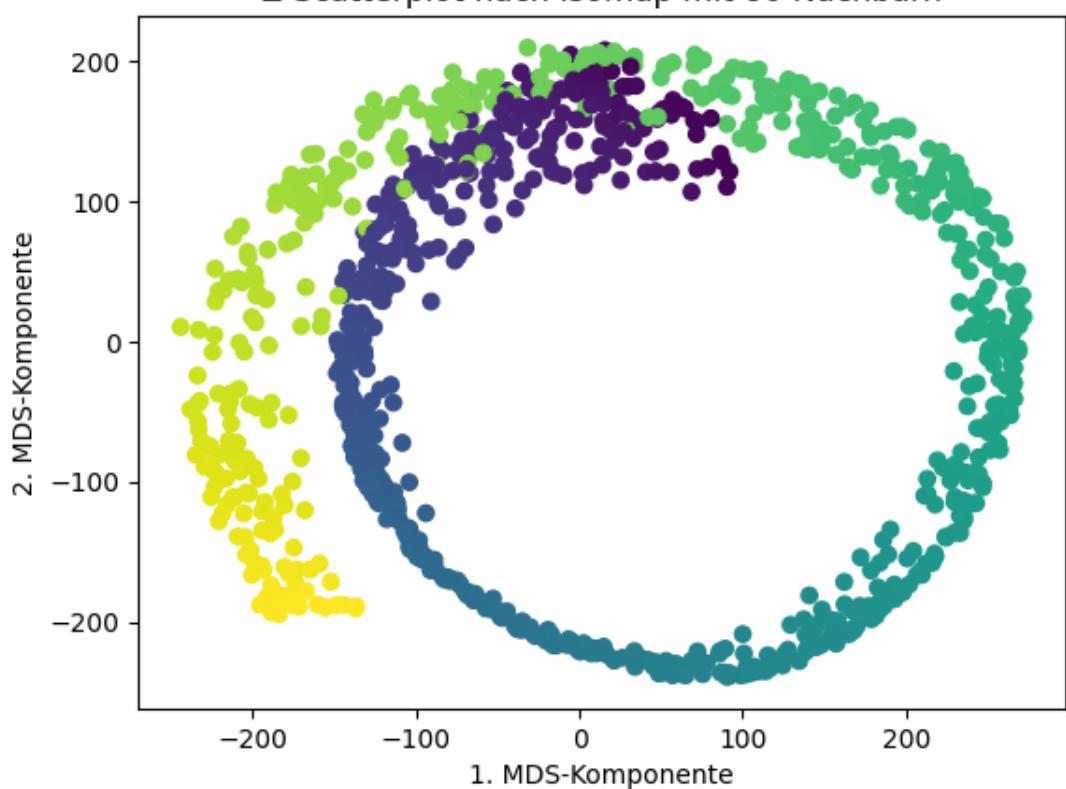
```
[20]: for K in [40, 80, 120, 260, 400, 600, 800]:
    Y = shortest_path(kneighbors_graph(M, n_neighbors=K, mode='distance'))
    Z, _ = multidimensional_scaling(Y, 2)

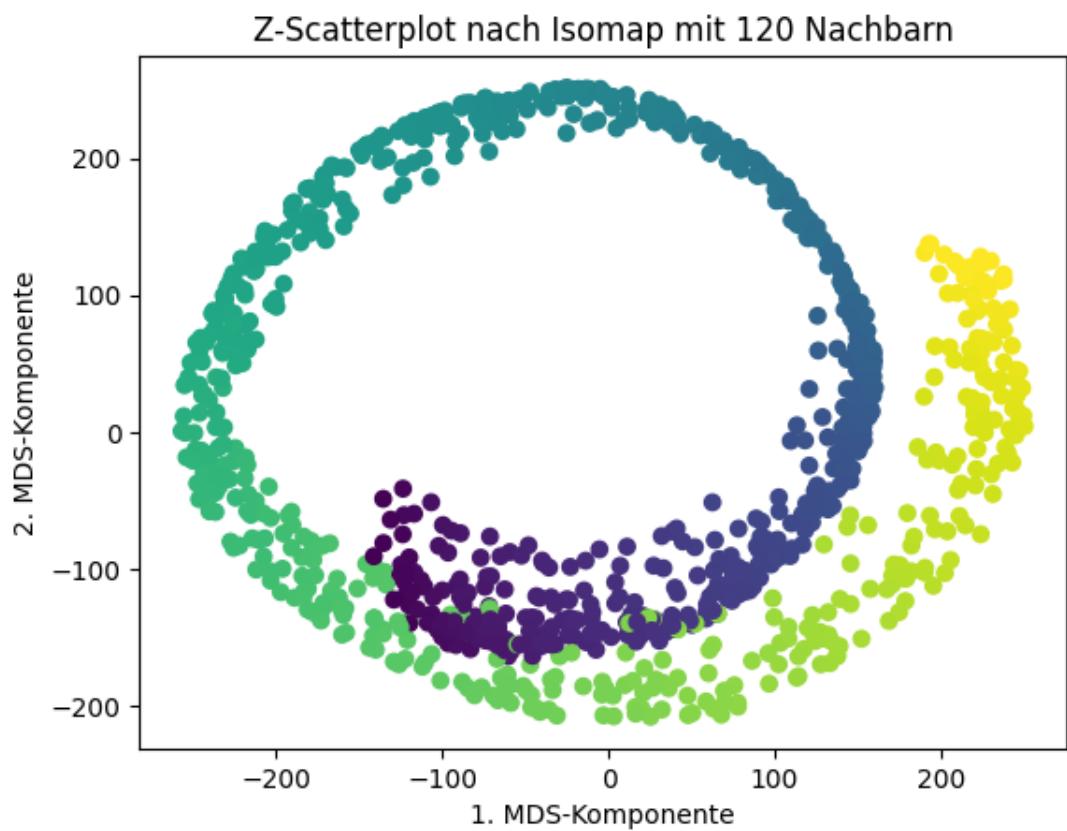
    plt.scatter(*Z, c=t)
    plt.title(f"Z-Scatterplot nach Isomap mit {K} Nachbarn")
    plt.xlabel("1. MDS-Komponente")
    plt.ylabel("2. MDS-Komponente")
    plt.show()
```

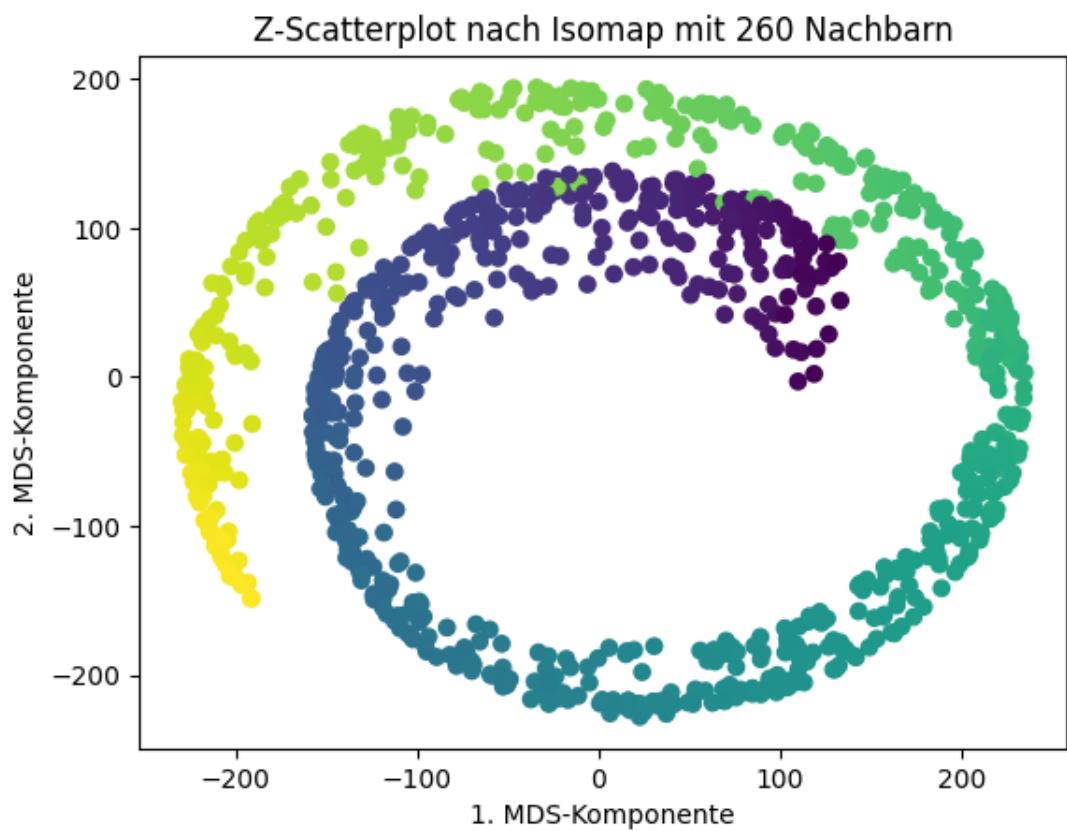
Z-Scatterplot nach Isomap mit 40 Nachbarn

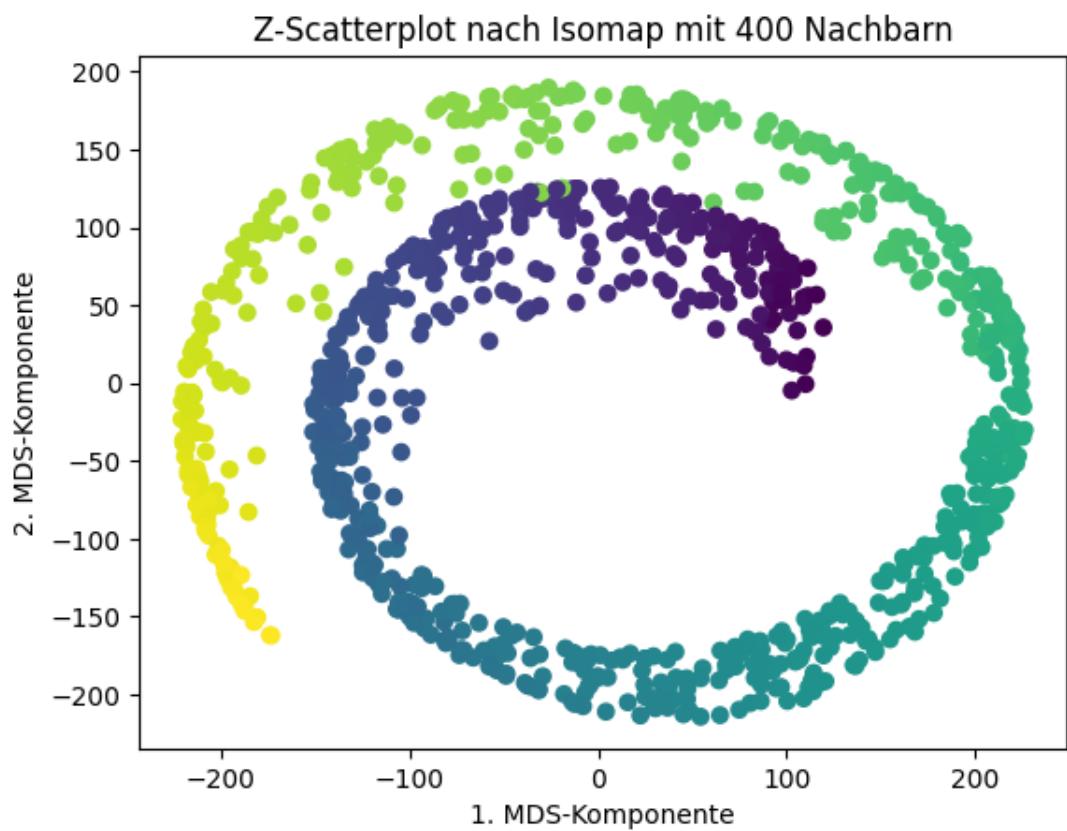


Z-Scatterplot nach Isomap mit 80 Nachbarn

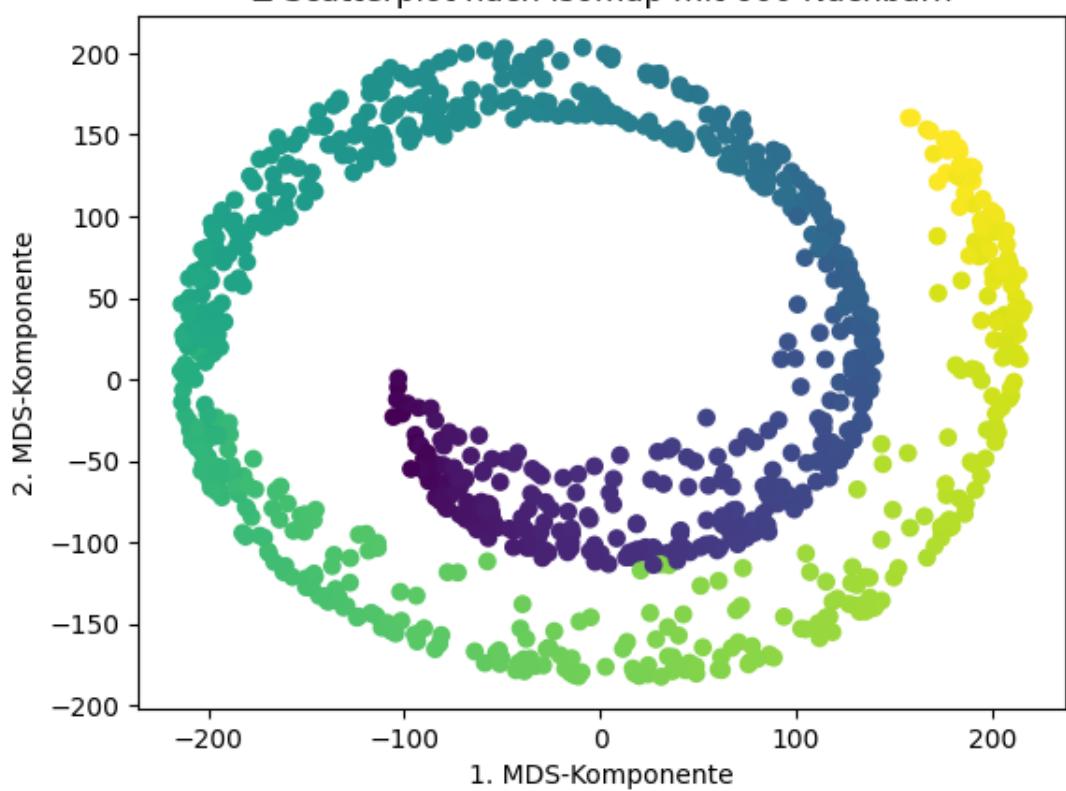




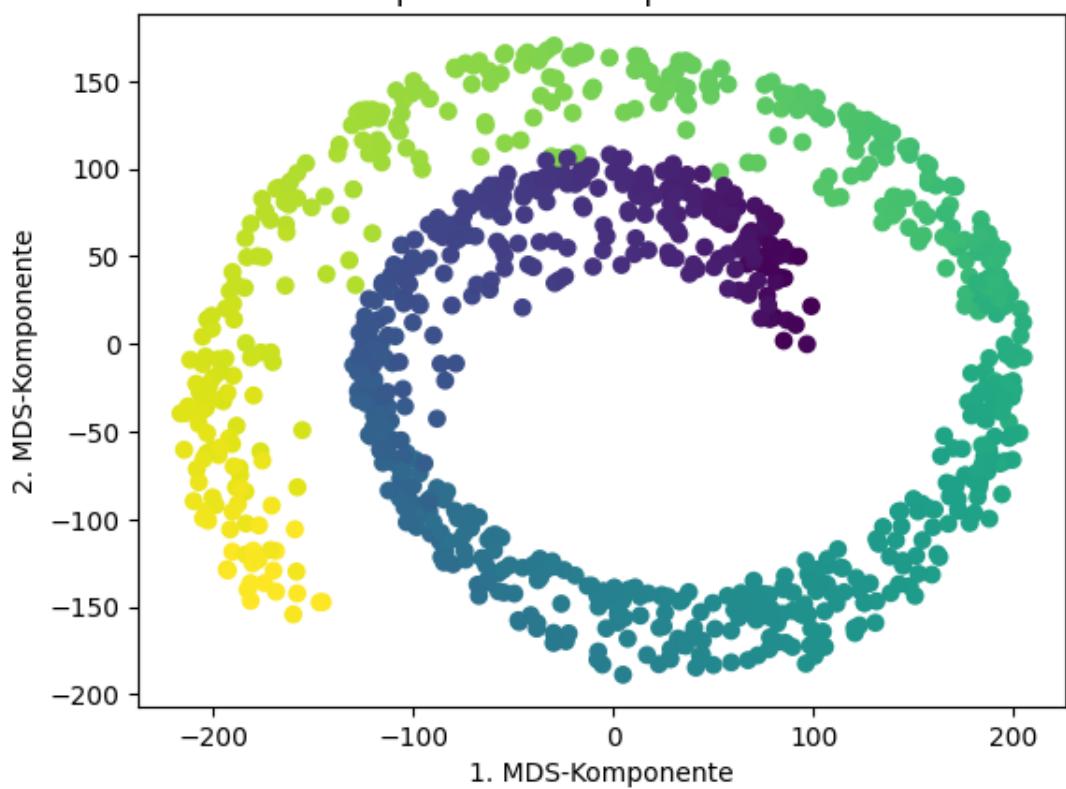




Z-Scatterplot nach Isomap mit 600 Nachbarn



Z-Scatterplot nach Isomap mit 800 Nachbarn



TSRI-8

July 23, 2024

1 Übung 8

Gruppenname: **TSRI**

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 Hinweise

- Nutzen Sie für die folgenden Aufgaben **nur** die Mittel, die Ihnen Numpy und Matplotlib zur Verfügung stellt.

1.0.2 In dieser Übung ...

... werden wir uns mit K-Means Clustering vertraut machen. Wir werden K-Means Clustern implementieren (Übung 8.1).

1.0.3 8.1 K-Means

In der Vorlesung haben Sie K-Means als eine klassische Clustering-Methode kennengelernt. Sie werden Ihr Wissen über den K-Means Algorithmus durch diese Übung vertiefen. Nutzen Sie für die Implementierung Befehle der Numpy-Bibliothek.

Ihre Aufgaben

- (1) Schlagen Sie in den Vorlesungsfolien den K-Means Algorithmus nach.
 1. Allen Datenpunkten zufällig zu den Klustern $1, \dots, K$ zuordnen.
 2. Iteriere, bis die Cluster sich nicht mehr ändern:
 1. Für jeden der K Cluster, berechne den geometrischen Schwerpunkt (centroid). Der Schwerpunkt ist der Featurevektor, der sich aus der Mittelung aller Featurevektoren der dem Cluster zugehörigen Datenpunkte ergibt.
 2. Weise jedem Datenpunkt dem Cluster zu, dessen Schwerpunkt dem Datenpunkt am nächsten liegt (im euklidischem Sinne).

$$\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{i,j}$$

2. Weise jedem Datenpunkt dem Cluster zu, dessen Schwerpunkt dem Datenpunkt am nächsten liegt (im euklidischem Sinne).

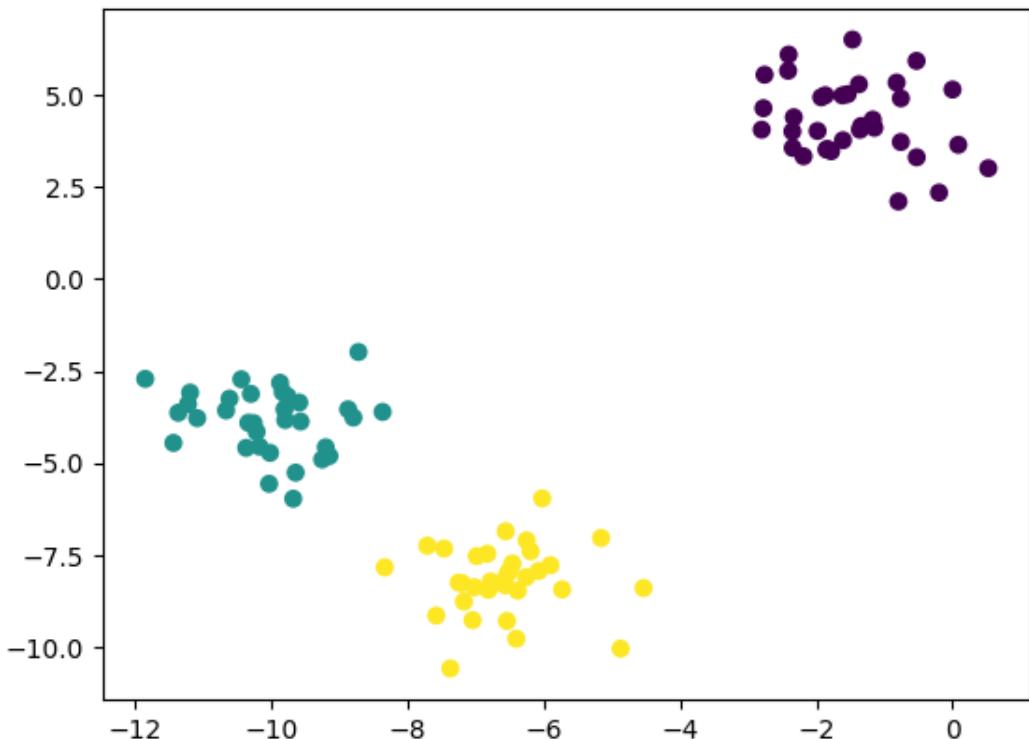
- (2) Ich habe Ihnen synthetische Daten zur Übung bereitgestellt. Bitte führen Sie die unten stehende Code-Zelle aus. Sie erzeugt 100 Datenpunkte mit je zwei Features (Merkmale, Array \mathbf{X}), organisiert in drei Cluster. Die Clusterzugehörigkeiten sind im Vektor \mathbf{y} kodiert.

```
[1]: import numpy as np
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

# generate data
X, y = make_blobs(n_samples=100, n_features=2, centers=3, random_state=1)
```

- (3) Visualisieren Sie die Daten in einem Scatterplot und färben Sie die Cluster gemäß ihrer Clusterzugehörigkeit ein.

```
[2]: plt.scatter(*X.T, c=y)
plt.show()
```



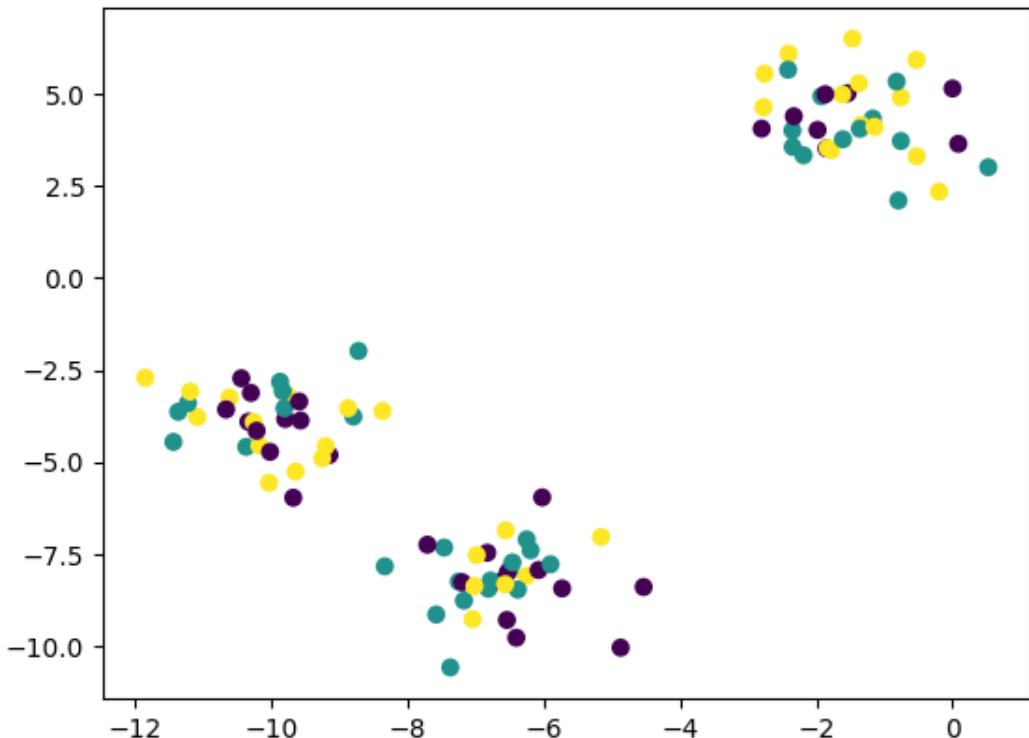
- (4) Die Anzahl der Cluster sei $K = 3$. Implementieren Sie Schritt 1 des K-Means Algorithmus, also die zufällige Zuweisung von Datenpunkten zu den K Clustern. Dies hier kann Ihnen dabei hilfreich sein.

```
[3]: # Sei  $C_i$  die Indexmenge der Datenpunkte aus Cluster  $i$ , und  $i \in [0, K]$ 
K = 3
```

```
y_random = np.array([np.random.randint(0, K) for i in range(X.shape[0])])
```

- (5) Visualisieren Sie in einem Scatterplot die Daten und färben Sie die Datenpunkte gemäß Ihrer aus Schritt (4) ermittelten zufälligen Clusterzugehörigkeit ein.

```
[4]: plt.scatter(*X.T, c=y_random)
plt.show()
```



- (6) Implementieren Sie nun die Iterationsschritte 2a und 2b, also die Bestimmung der Clusterzentren (geometrische Schwerpunkte) sowie die Neuzuordnung der Datenpunkte zu demjenigen Cluster, zu dem die Entfernung zum Schwerpunkt des Clusters am kleinsten ist. Nutzen Sie dabei als Distanzmaß den euklidischen Abstand. Eventuell können Ihnen diese Funktionen dabei hilfreich sein: `np.argmin`, `np.unique`.

- Prüfen Sie nach Schritt 2b, ob die Zuordnung von Datenpunkten zu Clustern tatsächlich alle Cluster enthält. Falls etwa ein Cluster keine Datenpunkte mehr enthält, starten Sie die Prozedur von Schritt 1 an neu. Wir werden später auf die “Leere-Cluster-Problematik” eingehen.
- Es empfiehlt sich, Ihren Code - sobald er funktioniert - in eine Funktion namens `kmeans` zu schreiben, die die Daten X sowie die Clusteranzahl K entgegennimmt, und das Clusteringergebnis y_{pred} ausgibt. Sie können dann im weiteren Verlauf dieser Übung einfach nur noch die Funktion `kmeans` aufrufen.

```
[5]: # Alternative für KMeans++
def initialize_centroids(X: np.ndarray, K: int) -> np.ndarray:
    n_samples, n_features = X.shape
    centroids = np.zeros((K, n_features))

    # Wähle das erste Zentrum zufällig aus den Datenpunkten aus
    centroids[0] = X[np.random.randint(0, n_samples)]

    # Wähle die restlichen K-1 Zentren aus
    for k in range(1, K):
        distances = np.min(distance.cdist(X, centroids[:k], 'euclidean'), axis=1)
        probabilities = distances / np.sum(distances)
        cumulative_probabilities = np.cumsum(probabilities)
        r = np.random.rand()

        for i, p in enumerate(cumulative_probabilities):
            if r < p:
                centroids[k] = X[i]
                break

    return centroids
```

```
[6]: from scipy.spatial import distance

def kmeans(X: np.ndarray, K: int, plusplus=False) -> np.ndarray:
    n_samples, n_features = X.shape

    # Schritt 1: Allen Datenpunkten zufällig zu den Klustern $1, \dots, K$ zuordnen.
    if plusplus:
        # Alternative Initialisierung für der Zentren für KMeans++
        centroids = initialize_centroids(X, K)
        y_pred = np.zeros(X.shape[0], dtype=int)
        x_clusterd_prev = np.ones(y_pred.shape) * -1
    else:
        centroids = np.zeros((K, n_features))
        y_pred = np.random.randint(0, K, size=X.shape[0])
        x_clusterd_prev = np.zeros(y_pred.shape)

    # Schritt 2a: Für jedes Cluster den geometrischen Schwerpunkt bestimmen
    while not np.array_equal(y_pred, x_clusterd_prev):
        print("Step")
        x_clusterd_prev = y_pred.copy()
```

```

for i in range(K):
    # C_i ist die Menge der Datenindizes, welche im Cluster i liegen
    C_i = [x_idx for x_idx in range(len(y_pred)) if y_pred[x_idx] == i]
    #centroids[:, i] = 1 / len(C_i) * np.sum(X[C_i], axis=0)
    if C_i: # Vermeide Division durch Null, falls ein Cluster leer ist
        centroids[i] = np.mean(X[C_i], axis=0)
    else:
        # Falls ein Cluster leer ist, initialisiere ein neues Zentrum
        ↪zufällig
        centroids[i] = X[np.random.randint(0, X.shape[0])]

    # Schritt 2b: Jeden Datenpunkt dem nächstgelegenen Cluster zuweisen
    distances = distance.cdist(X, centroids, metric='euclidean')
    y_pred = np.argmin(distances, axis=1)

    # Überprüfe ob die Zuordnung von Datenpunkten zu Clustern tatsächlich
    ↪alle Cluster enthält
    if np.unique(y_pred).shape[0] != K:
        print("Wiederhole von Schritt 1")
        y_pred = np.random.randint(0, K, size=X.shape[0])

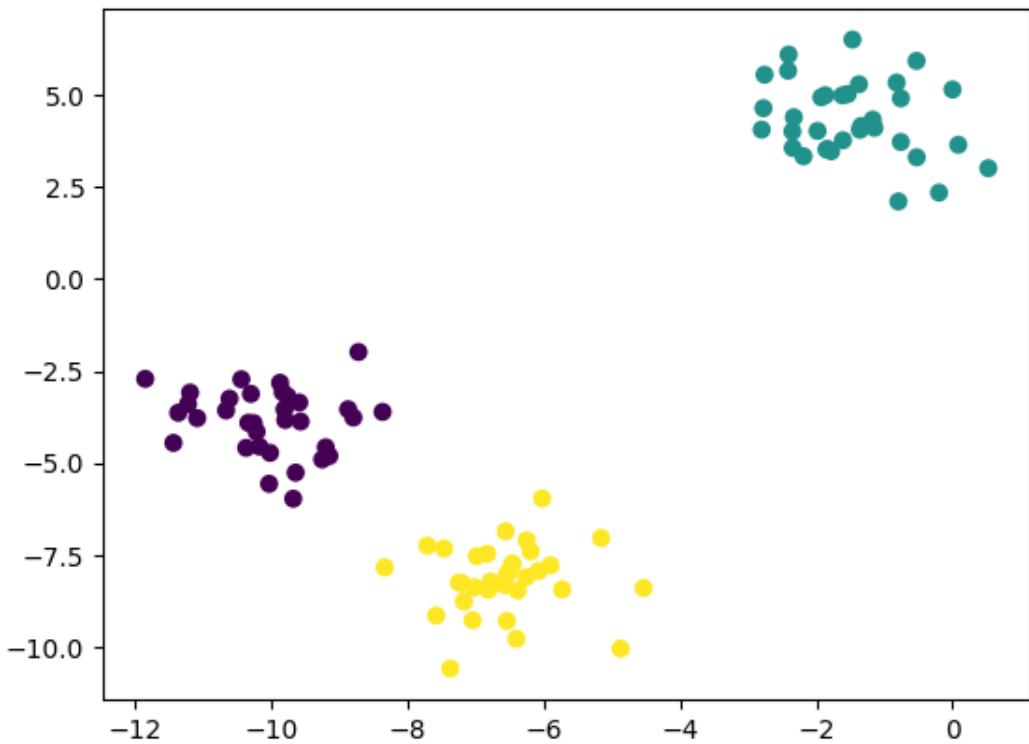
return y_pred

```

- (7) Führen Sie nun mit Ihrer Implementierung K-Means auf den Daten aus ($K = 3$). Visualisieren Sie die Daten in einem Scatterplot und färben Sie sie gemäß der durch K-Means ermittelten Clusterzugehörigkeiten ein. Vergleichen Sie Ihren Plot mit dem aus Schritt (5).

```
[7]: y_pred = kmeans(X, 3, plusplus=True)
plt.scatter(*X.T, c=y_pred)
plt.show()
```

Step
Step
Step



Sie haben im vorherigen Schritt gesehen, dass Sie mithilfe von K-Means Cluster im Datensatz identifizieren können. Nun geht es darum, Ihre Implementierung abzusichern, sodass Sie leeren Clustern, die während der Iterationen entstehen können, umgehen kann.

- (8) Implementieren Sie vor Schritt 2a (also zu Beginn Ihrer Iteration) ein Verfahren, das überprüft, ob es Cluster gibt, die keine Datenpunkte enthalten. In diesem Falle wählen Sie einen zufälligen Datenpunkt aus und weisen ihn dem leeren Cluster zu. Dieser [Befehl](#) könnte Ihnen bei diesem Unterfangen hilfreich sein.
- Wie kann es dazu kommen, dass wir während der Iteration leere Cluster erhalten? Schauen Sie sich zur Erklärung das [hier](#) verlinkte Beispiel an.
 - Wenn Sie Schritt (8) fertig bearbeitet haben, entfernen Sie einfach den Neustart-Mechanismus, den Sie in Schritt (6) implementiert hatten. Sie brauchen diesen nicht mehr.

Fällt während einer Iteration der Schwerpunkt des Clusters C_i so zwischen zwei umliegende Cluster C_a und C_b , dass alle Elemente $i \in C_i$ bei der nächsten Iteration eine geringere euklidische Distanz zu den Cluster Schwerpunkten C_a und C_b haben, dann *zerreißt* das Cluster C_i und wird leer.

```
[8]: from scipy.spatial import distance
import random

def kmeans(X: np.ndarray, K: int, plusplus=False) -> tuple[np.ndarray, np.
    ndarray]:
    n_samples, n_features = X.shape
```

```

assert n_samples > K, "Es können nicht mehr Cluster als Datenpunkte\u2191  

↪gebildet werden"

# Schritt 1:
if plusplus:
    # Alternative Initialisierung für der Zentren für KMeans++
    centroids = initialize_centroids(X, K)
    y_pred = np.zeros(n_samples, dtype=int)
    x_clusterd_prev = np.ones(y_pred.shape) * -1
else:
    # Allen Datenpunkten zufällig zu den Klustern $1, \dots, K$ zuordnen.
    centroids = np.zeros((K, n_features))
    y_pred = np.random.randint(0, K, size=n_samples)
    x_clusterd_prev = np.zeros(y_pred.shape)

while not np.array_equal(y_pred, x_clusterd_prev):
    x_clusterd_prev = y_pred.copy()

    # Überprüfe, ob einzelne Cluster leer sind
    if np.unique(y_pred).shape[0] != K:
        # Da auch mehrere Cluster leer sein können, müssen wir hier durch\u2191  

↪alle leeren Cluster iterieren.
        empty_clusters = set(range(K)).difference(y_pred)
        # print(f"Leere(s) Cluster {empty_clusters} gefunden.")
        # um sicherzustellen, dass nicht zufällig mehrfach der gleiche\u2191  

↪Datenpunkt ausgewählt wird
        replaced_idxs = set()
        for ec in empty_clusters:
            random_idx = random.choice(list(set(range(y_pred.shape[0])).  

↪difference(replaced_idxs)))
            replaced_idxs.add(random_idx)
            y_pred[random_idx] = ec

    # Schritt 2a: Für jedes Cluster den geometrischen Schwerpunkt bestimmen
    for i in range(K):
        # X_i ist die Menge der Daten, welche im Cluster i liegen
        X_i = X[y_pred == i]
        if len(X_i) > 0:
            centroids[i] = np.mean(X_i, axis=0)

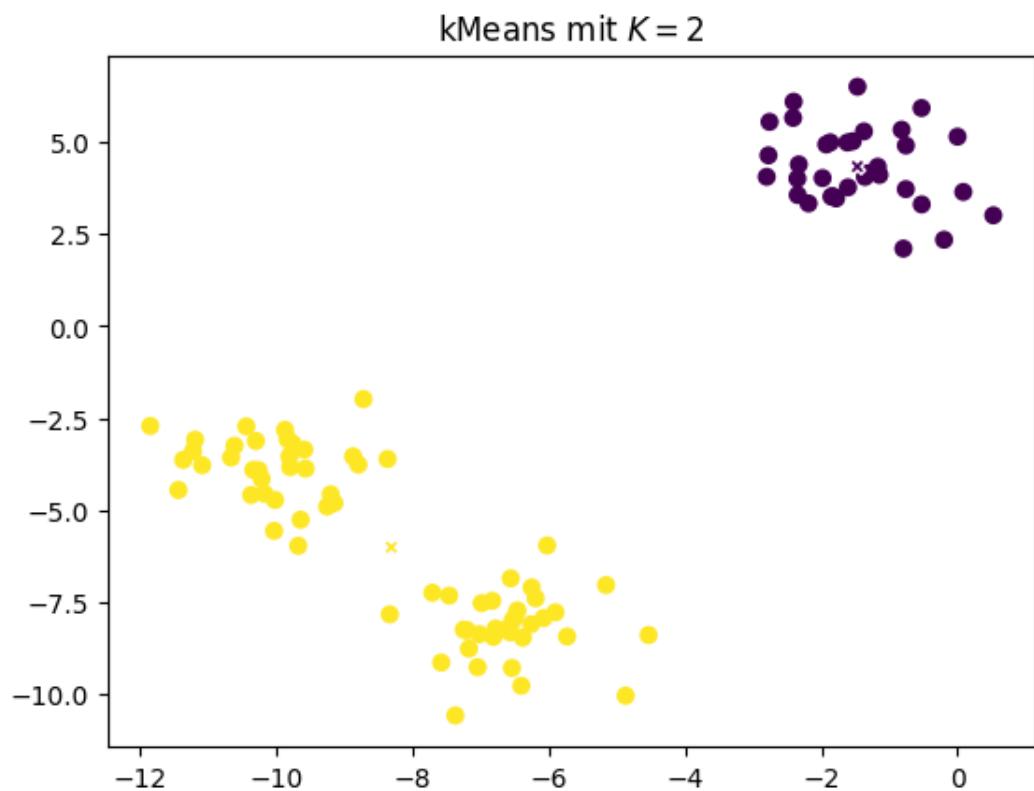
    # Schritt 2b: Jeden Datenpunkt dem nächstgelegenen Cluster zuweisen
    distances = distance.cdist(X, centroids, metric='euclidean')
    y_pred = np.argmax(distances, axis=1)

return y_pred, centroids

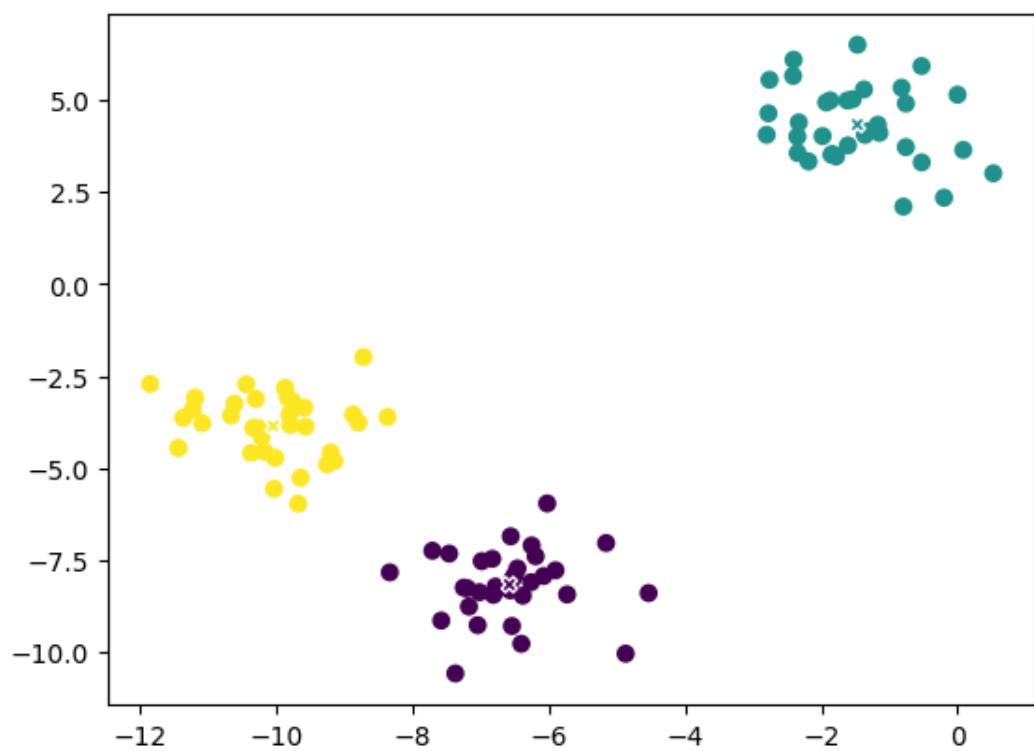
```

- (9) Ermitteln Sie Cluster für $K = 2, 3, 4, 5$ und visualisieren Sie sie (wie in Schritt 7) in Scatterplots. Was fällt Ihnen auf? (1-3 Sätze)

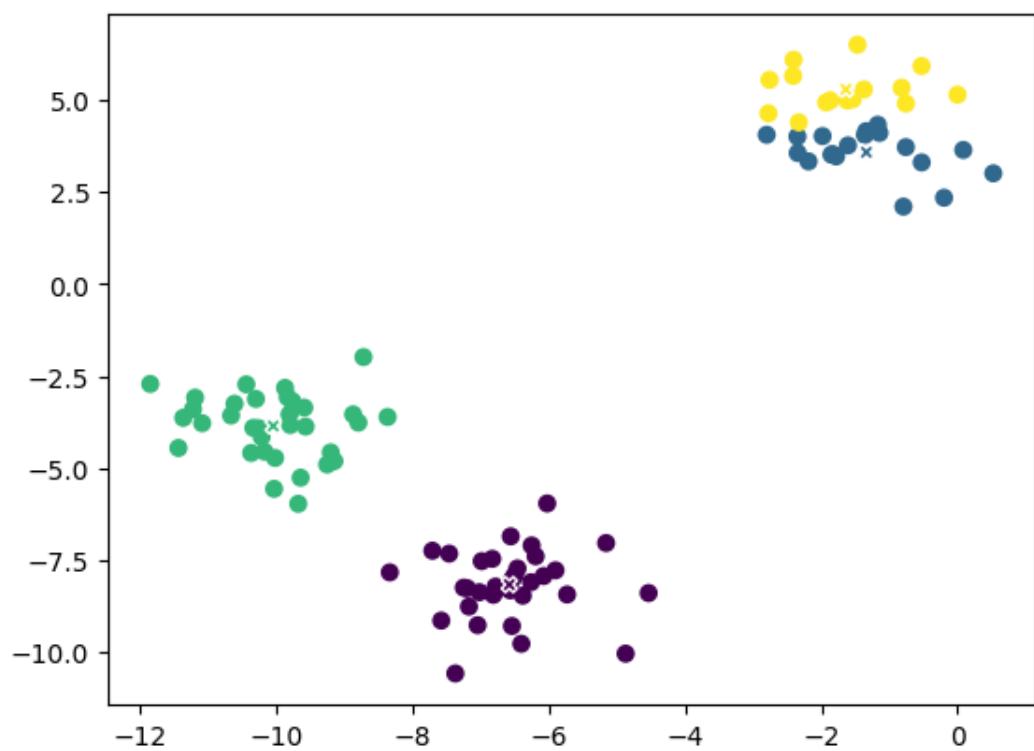
```
[9]: for K in 2, 3, 4, 5:  
    x_clusterd_new, x_centroids = kmeans(X, K)  
    plt.title(f"kMeans mit $K={K}$")  
    plt.scatter(X[:, 0], X[:, 1], c=x_clusterd_new)  
    plt.scatter(x_centroids[:, 0], x_centroids[:, 1], c=range(K),  
                edgecolors='white', marker="X")  
    plt.show()
```

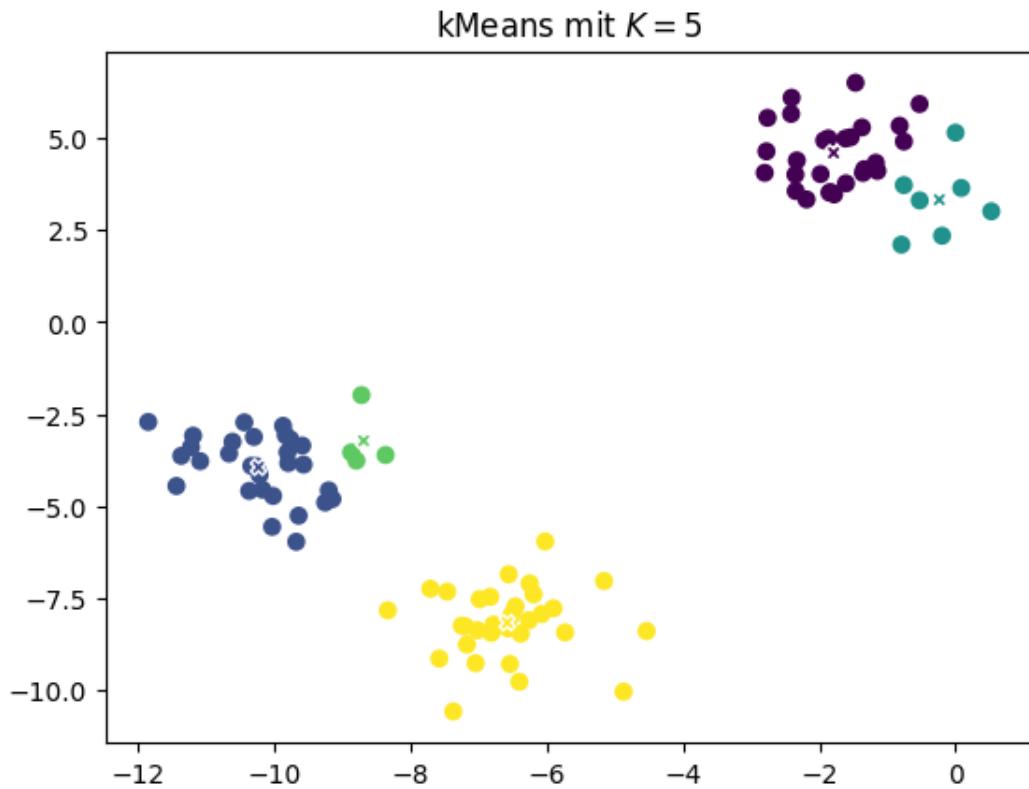


kMeans mit $K = 3$



kMeans mit $K = 4$





Die Bildung von 2 oder 3 Clustern erscheint noch sinnvoll, während für $K > 3$ die Unterteilung willkürlich erscheint.

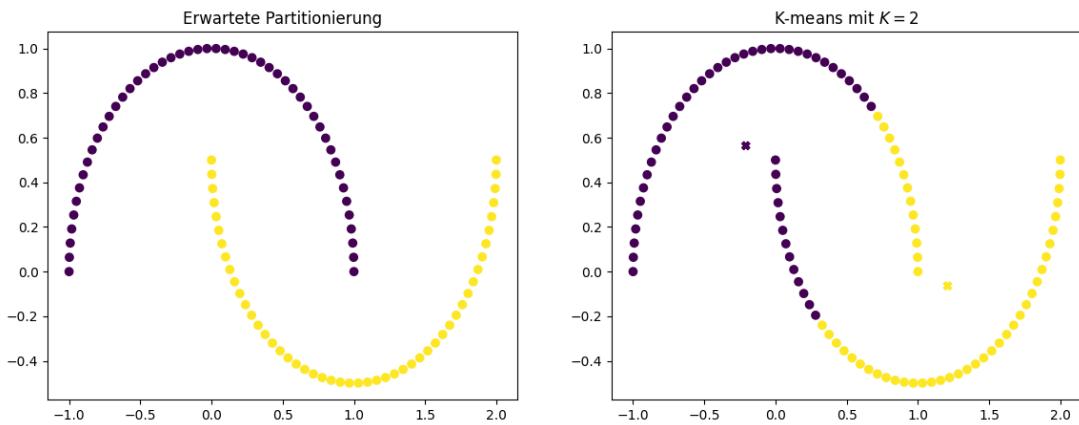
- (10) Ermitteln Sie für den unten erzeugten Datensatz mithilfe von K-Means $K = 2$ Cluster und visualisieren Sie das Clusterergebnis (vergessen Sie nicht, die Punkte gemäß ihrer Clusterzugehörigkeit einzufärben).

```
[10]: from sklearn.datasets import make_moons
[X2, y2] = make_moons(random_state=1)
```

```
[11]: y2_pred, y2_centroids = kmeans(X2, 2)

plt.subplots(1, 2, figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.title("Erwartete Partitionierung")
plt.scatter(X2[:, 0], X2[:, 1], c=y2)

plt.subplot(1, 2, 2)
plt.title(f"K-means mit $K=2$")
plt.scatter(X2[:, 0], X2[:, 1], c=y2_pred)
plt.scatter(y2_centroids[:, 0], y2_centroids[:, 1], c=range(2), marker="X")
plt.show()
```



- (11) Erklären Sie, warum Sie im vorherigen Schritt nicht die Cluster erhalten, wie Sie sie für die Daten erwarten würden. Mit welchem anderen Clusteransatz könnten Sie eventuell die korrekten Cluster ermitteln? (3-6 Sätze).

Wir verwenden den euklidischen Abstand zu den Centroids mit K-means als *dissimilarity measure*, jedoch lassen sich die Daten damit nicht gut ohne vorherige Transformation partitionieren. Für eine korrekte Partitionierung sollte hier der Abstand zwischen den einzelnen Datenpunkten zu ihren Nachbarn eine größere Rolle spielen als ihr Abstand zu den Cluster Schwerpunkten. Um bei diesem Datensatz Cluster zu bilden, bei denen die unmittelbare Distanz zu anliegenden Datenpunkten entscheidender sein soll, als der Abstand zu einem gemeinsamen Schwerpunkt, könnte eine **Featuretransformation** oder eine **hierarchische Clusteranalyse** vielleicht hilfreicher sein.

(Einschub) agglomeratives hierarchisches Clustern (HCA) mit der quadratischen Distanz als dissimilarity measure.

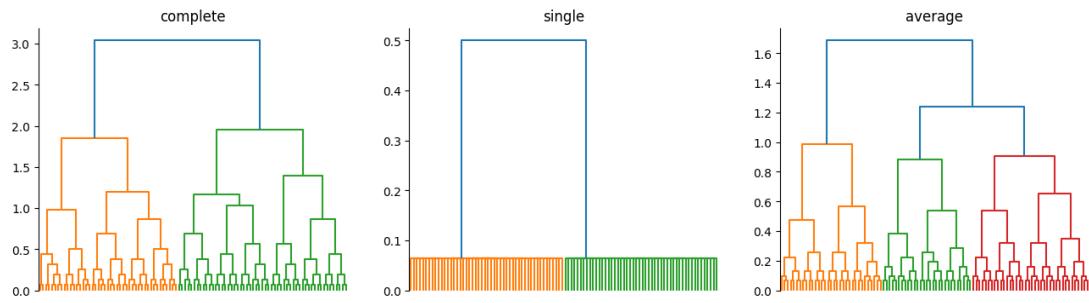
```
[12]: from scipy.cluster import hierarchy
Z = hierarchy.linkage(X2, 'single')

[13]: fig, axs = plt.subplots(1, 3, figsize=(16, 4))

for i, m in enumerate(['complete', 'single', 'average']):
    Z = hierarchy.linkage(X2, m)
    hierarchy.dendrogram(Z, truncate_mode='none', show_leaf_counts=True, no_labels=True, ax=axs[i])

    axs[i].set_title(m)
    axs[i].spines['top'].set_visible(False)
    axs[i].spines['right'].set_visible(False)
    axs[i].spines['bottom'].set_visible(False)

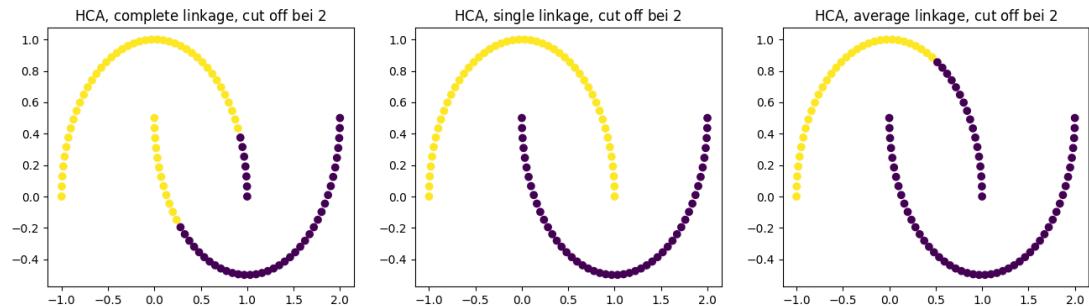
plt.show()
```



```
[14]: fig, axs = plt.subplots(1, 3, figsize=(16, 4))

for i, m in enumerate(['complete', 'single', 'average']):
    y2_hca_pred = hierarchy.cut_tree(hierarchy.linkage(X2, m), n_clusters=2)
    axs[i].set_title(f"HCA, {m} linkage, cut off bei $2$")
    axs[i].scatter(*X2.T, c=y2_hca_pred)

plt.show()
```



- (12) [Optional] Schlagen Sie in den Vorlesungsfolien nach, wie die summierte Intra-Cluster-Variation $\sum_{k=1}^K W(C_k)$ definiert ist. Implementieren Sie eine Funktion `icv`, die die Daten X sowie die Clusterzugehörigkeit y entgegennimmt und Ihnen die summierte Intra-Cluster-Variation zurückgibt.

Die **Intra-Cluster-Variation** $W(C_k)$ des Clusters C_k ist die Summe der quadratischen euklidischen Distanzen zwischen allen Datenpunkten.

$$\begin{aligned} W(C_k) &= \frac{1}{|C_K|} \sum_{i, i' \in C_K} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \\ &= 2 \sum_{i \in C_K} \sum_{j=1}^p (x_{ij} - \bar{x}_{kj})^2 \end{aligned}$$

mit Centroid:

$$\bar{x}_{kj} = \frac{1}{|C_K|} \sum_{i \in C_K} x_{ij}$$

Das Clustering ist gut, wenn die Intra-Cluster-Variation minimiert werden wird, also wir Cluster haben, in denen der euklidische Abstand zwischen den Datenpunkten möglichst klein ist.

```
[15]: from scipy.spatial import distance

def icv(X: np.ndarray, y:np.ndarray) -> float:
    summed_intra_cluster_variation = 0
    # äußere Summe, iteriere über alle Klassen k
    for k in np.unique(y):
        # Erstelle alle x_ij und x_i'j paare
        X_k = X[y == k]
        # innere Summe, Differenzenquadrate und Mittelwert
        distances = distance.cdist(X_k, X_k, metric='euclidean')
        intra_cluster_variation = np.sum(distances ** 2) / float(X_k.shape[0])
        summed_intra_cluster_variation += intra_cluster_variation
    return summed_intra_cluster_variation

print(f"icv = {icv(X2, y2_pred)}")
```

icv = 80.07819322206497

- (13) [Optional] Sie wissen aus der Vorlesung, dass K-Means Clustering-Ergebnisse liefern kann, die lokalen Minima in der Intra-Cluster-Variation entsprechen. Das erzielte Clustering hängt von der zufälligen Initialisierung der Clusterzugehörigkeiten in Schritt 1 des Algorithmus ab. Wir wollen nun erreichen, dass Ihre Implementierung von kmeans mehrfache ($n = 20$) Clustering-Versuche mit unterschiedlichen zufälligen initialen Clusterzugehörigkeiten unternimmt und das Clustering mit der niedrigsten Intra-Cluster-Variation zurückliefert. Ändern Sie dementsprechend Ihre Funktion ab und implementieren Sie diese Funktionalität.

```
[16]: def repeated_kmeans(X: np.ndarray, K: int, repeats=20) -> tuple[float, np.ndarray]:
    best_icv: float = float('inf')
    for i in range(repeats):
        y, centroids = kmeans(X, K)
        y_icv = icv(X, y)
        if y_icv < best_icv:
            best_icv, best_y, best_centroids = y_icv, y, centroids
    return best_icv, best_y, best_centroids

repeated_kmeans(X2, 8, 200)
```

```
[16]: (np.float64(10.461054974366206),
       array([4, 4, 5, 4, 3, 0, 5, 4, 1, 1, 3, 2, 7, 7, 7, 5, 5, 2, 4, 6, 6, 1,
              3, 7, 0, 5, 3, 5, 2, 2, 1, 3, 7, 3, 3, 2, 2, 7, 0, 5, 4, 7, 4, 0,
              4, 2, 2, 3, 5, 3, 2, 7, 0, 6, 7, 2, 0, 3, 6, 1, 5, 1, 1, 2, 2, 3,
```

```

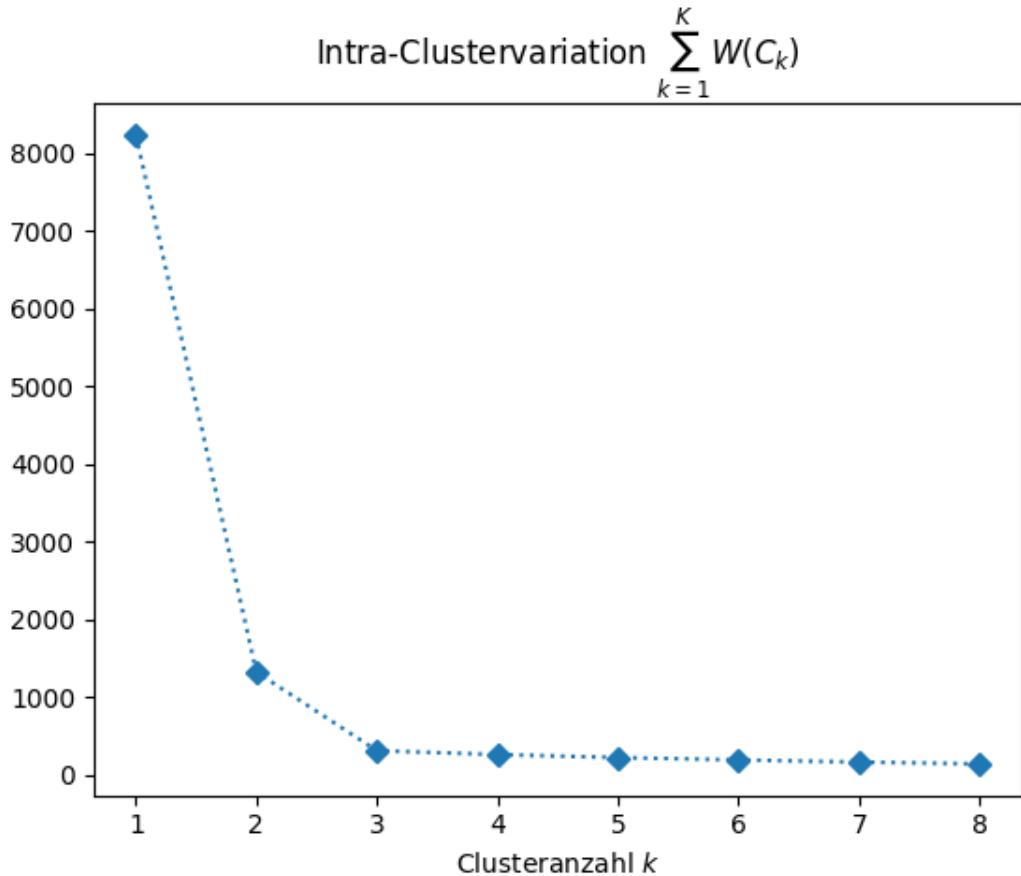
6, 7, 6, 3, 7, 0, 7, 1, 6, 3, 0, 1, 4, 0, 5, 5, 6, 0, 0, 7, 5, 6,
1, 4, 6, 0, 1, 4, 6, 4, 6, 1, 6, 2]),
array([[ 0.36614247,  0.90437898],
       [ 0.63385753, -0.40437898],
       [-0.90047281,  0.36456104],
       [ 1.90047281,  0.13543896],
       [ 1.36614247, -0.40437898],
       [-0.36614247,  0.90437898],
       [ 0.90047281,  0.36456104],
       [ 0.09952719,  0.13543896]]))

```

- (14) [Optional] Bestimmen Sie mit Ihrer Funktion `icv` aus Schritt (12) die Intra-Cluster-Variation für $K = 1, \dots, 8$ und tragen Sie diese Größe gegen K in einem Plot auf.

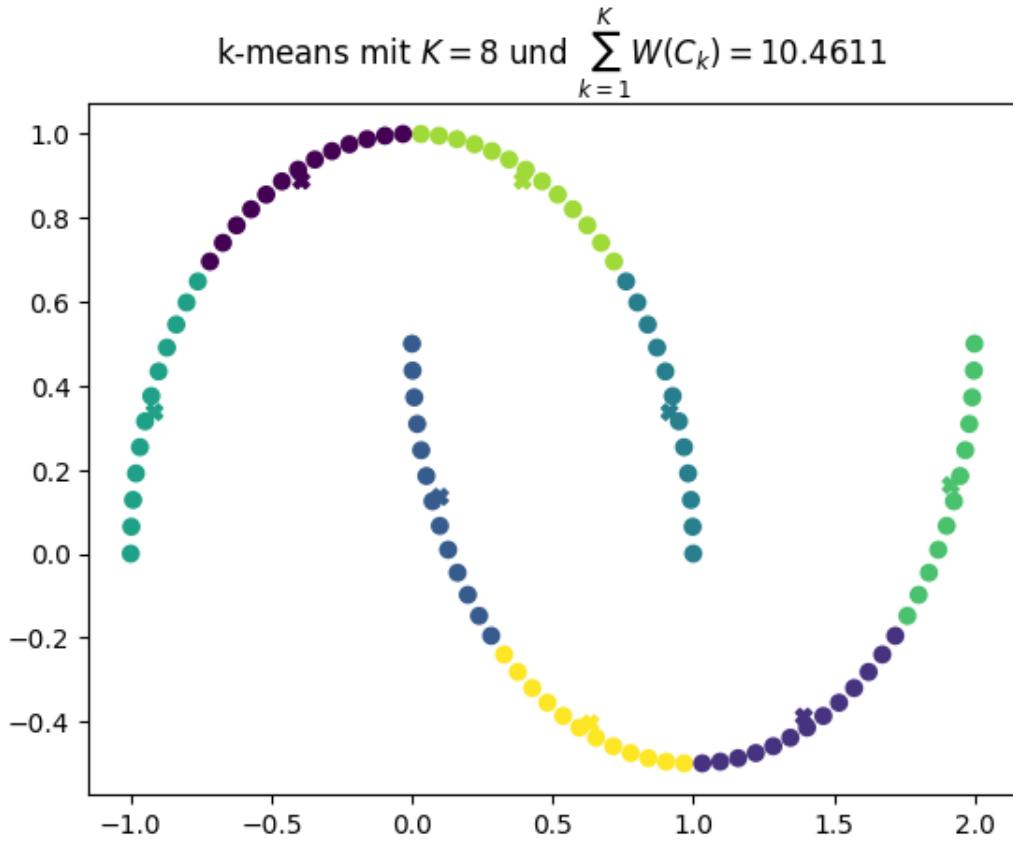
```
[17]: icvs: list[float] = []
for K in range(1, 8+1):
    icv_k, _, _ = repeated_kmeans(X, K, 20)
    icvs.append(icv_k)

plt.title("Intra-Clustervariation $\sum_{k=1}^K W(C_k)$")
plt.plot(range(1, 8+1), icvs, 'D:')
plt.xlabel("Clusteranzahl $k$")
plt.show()
```



- (15) [Optional] Die “korrekte” Anzahl an Clustern (sofern es sie überhaupt gibt) ist notorisch schwierig zu bestimmen und Methoden, diese zu bestimmen, ist aktueller Forschungsgegenstand. Ein pragmatischer Ansatz ist es, sich die Intra-Cluster-Variation (ICV) als Funktion der Clusteranzahl K zu plotten und das K auszuwählen, zu dem die ICV gerade stark abgesunken ist. Betrachten Sie dazu Ihren Plot aus Schritt (14): Gibt es ein K , ab der die summierte Intra-Cluster-Variation stark sinkt? Geben Sie diesen Wert für K an. Ist dieser Wert Ihrer Meinung nach sinnvoll?

```
[18]: K2 = 8
X2_icv, X2_clusterd, X2_centroids = repeated_kmeans(X2, K2, 30)
plt.title(f"k-means mit $K={K2}$ und ${\sum_{k=1}^K W(C_k)}$ = {X2_icv:.4f}")
plt.scatter(*X2.T, c=X2_clusterd)
plt.scatter(*X2_centroids.T, c=range(K2), marker="x")
plt.show()
```



Der ICV-Wert ist sowohl beim Übergang von $K = 1$ auf $K = 2$, als auch vom Übergang von $K = 2$ auf $K = 3$ stark abgesunken. Daher könnte man $K = 2$ oder $K = 3$ wählen. Möchte man jedoch die ICV generell einfach minimieren, wäre ein höheres K zu wählen.

Damit darf ich Ihnen gratulieren. Sie haben das K-Means-Clusteringverfahren intensiv durchdrungen. K-Means wird Ihnen in Data Science und Machine Learning Projekten oft begegnen.

1.0.4 8.2 Bildkompression (mittels K-Means)

In dieser Übung werden Sie lernen, wie Sie mithilfe von K-Means ein einfaches Verfahren etablieren können, um Bilder zu komprimieren.

Das Verfahren:

1. Sie starten mit einem Foto in 16 Millionen Farben (24 Bit). Die Farbe jedes Pixels wird durch einen Rot-, Grün- und Blauanteil kodiert (jeweils 8 Bit) und besteht daher aus einem Featurevektor der Länge 3.
2. Mithilfe von K-Means finden Sie in diesen Featurevektoren 16 Cluster und ihre 16 Clusterzentren.
3. Die Bildinformation lässt sich nun mithilfe der 16 Cluster sowie ihrer Clusterzentren darstellen. Für die Kodierung der Clusteridentität reichen 4 Bits ($2^4 = 16$ Cluster). Die assoziierten 16 verschiedenen Vektoren der Clusterzentren werden ebenfalls gespeichert.

Ihre Daten

- Foto: <https://data.bialonski.de/ds/eichhorn.png>

Ihre Aufgaben

- (1) Importieren Sie die Daten in das Array `im`. Welche Dimensionen hat das Array? In welchen Dimensionen wird was kodiert?

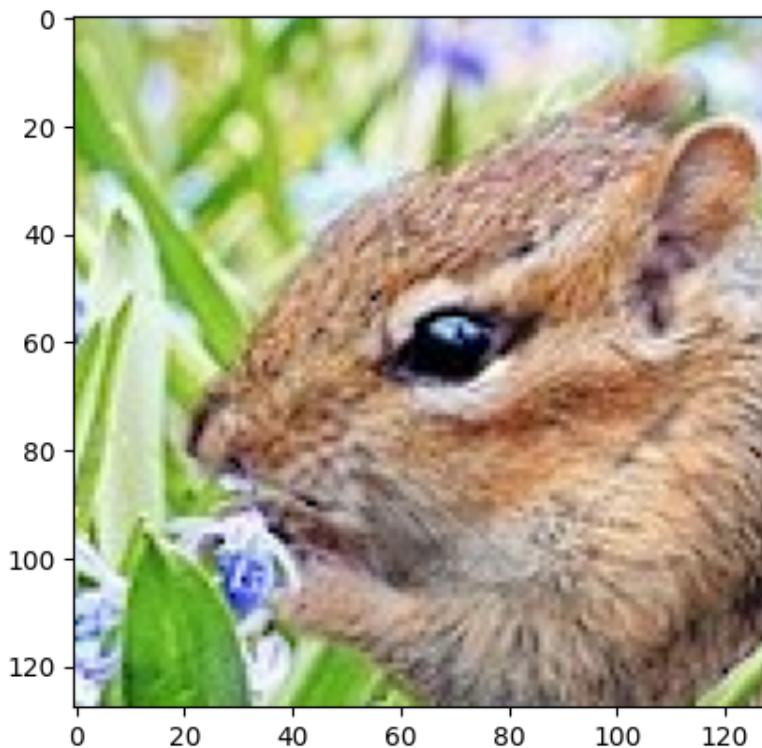
```
[19]: im = plt.imread("eichhorn.png")
im.shape
```

[19]: (128, 128, 3)

Die 128x128 Pixel des Bildes haben je einen Rot, Grün und Blau Wert.

- (2) Visualisieren Sie die Daten im Array `im`.

```
[20]: plt.imshow(im)
plt.show()
```



- (3) Transformieren Sie das Array `im` in ein neues Array `X` mit neuem *Shape*, sodass dieses Array nur noch 2 Dimensionen aufweist: In den Zeilen stehen die Datenpunkte (Pixel) und in den Spalten die Farbkodierung.

```
[21]: im = im.reshape((im.shape[0] * im.shape[1], im.shape[2]))  
im.shape
```

[21]: (16384, 3)

- (4) Wenden Sie Ihren K-Means Algorithmus aus Übung 8.1 an, um 16 Cluster im Array X zu finden. Geben Sie außerdem auch die assoziierten Clusterzentren zurück.

- **Wichtig:** Wenn Sie Schwierigkeiten mit Ihrer Implementierung von K-Means haben, dann können Sie auch die Implementierung von [scikit-learn](#) benutzen.

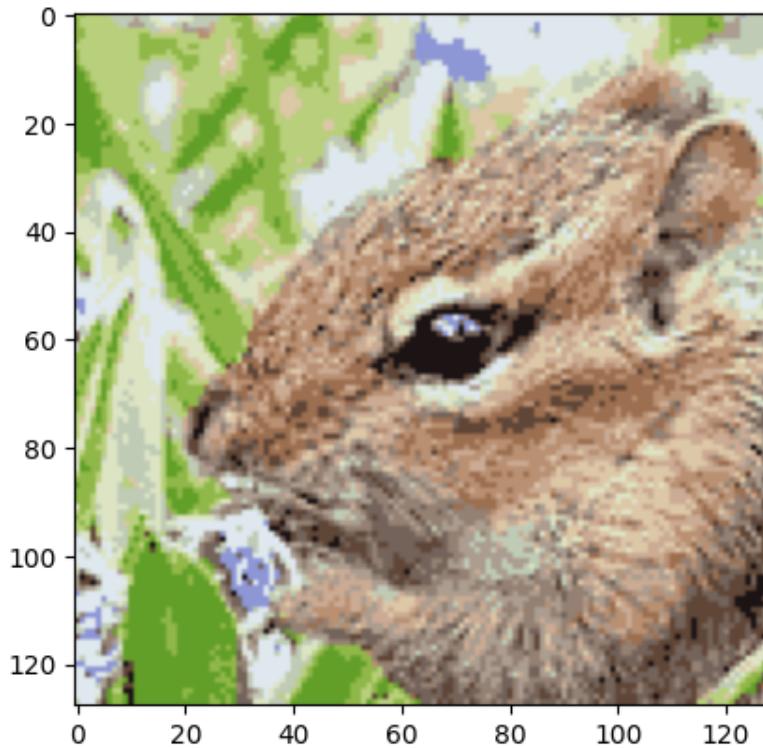
```
[22]: im_pred, im_centroids = kmeans(im, 16)
```

- (5) Wir erstellen nun das Foto in seiner komprimierten Form, um es in einem späteren Schritt visualisieren zu können. Dazu müssen wir ein Array erstellen, wo jedes Pixel nicht mehr durch seinen ursprünglichen Rot-, Grün-, Blau-Vektor dargestellt wird, sondern durch den assoziierten Clusterzentrumsvektor. Jedes Pixel wurde über K-Means einem von 16 Clustern und damit auch einem von 16 Clusterzentrumsvektoren zugeordnet.
- Erstellen Sie ein Array Z mit denselben Dimensionen wie X , wo Sie allerdings die Features (die Farbkodierung) durch die jeweiligen Clusterzentrumsvektoren ersetzt haben.

```
[23]: Z: np.ndarray = np.zeros(im.shape)  
for i, k in enumerate(im_pred):  
    Z[i, :] = im_centroids[k, :]
```

- (6) Geben Sie dem Array Z wieder die Form (Shape), die dem ursprünglichen Array X entspricht und visualisieren Sie dieses Array als Bild (siehe Schritt (2)).

```
[24]: plt.imshow(Z.reshape((128, 128, 3)))  
plt.show()
```



- (7) Vergleichen Sie die Bilder aus Schritt (2) und Schritt (6): Sehen Sie große Qualitätsunterschiede?

Der Qualitätsunterschied ist bemerkbar, aber außerordentlich gering, wenn man bedenkt, dass nur noch 16 geschickte Farben in dem Bild vorkommen.

- (8) [Optional] Wie viele Bytes benötigen Sie zum Speichern des ursprünglichen Bildes? Wie viele Bytes benötigen Sie, wenn Sie das Foto komprimiert mit K-Means abspeichern? Berechnen Sie die zu erwartenden Werte und geben Sie sie an.

```
[25]: print(np.size(im), np.size(im)*4, im.dtype)
```

49152 196608 float32

Das ursprüngliche Bild wurde mit $128^2 \cdot 3$ Farbwerten kodiert, welche als `float32` (also mit 4 Byte pro Farbwert) gespeichert wurden.

$$49152 \cdot 4 \text{ Byte} \approx 196.6 \text{ kByte}$$

In der komprimierten Form gibt es nur noch 16 verschiedene Farbwerte, zu deren Unterscheidung lediglich 4 bit benötigt werden.

$$49152 \cdot 4 \text{ bit} \approx 24.58 \text{ kByte}$$

Zusätzlich müssen allerdings noch die 16 Centroids abgespeichert werden, da diese für jedes Bild einzigartig sind.

$$16 \cdot 3 \cdot 4 \text{ Byte} = 192 \text{ Byte}$$

Damit liegt der Speicherbedarf für das Bild nach Kompression bei 24.768 kByte, was fast $\frac{1}{8}$ der ursprünglichen Größe ist.

HCA

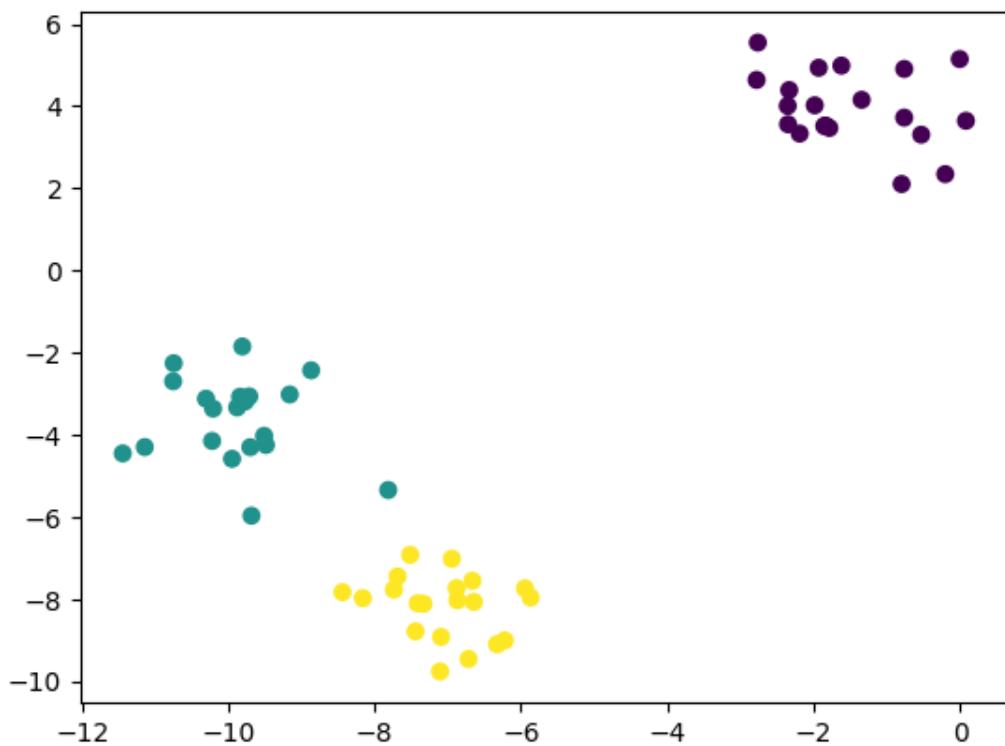
July 23, 2024

1 Hierarchische Clusteranalyse (HCA)

```
[1]: import numpy as np
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

# generate data
X, y = make_blobs(n_samples=60, n_features=2, centers=3, random_state=1)

[2]: plt.scatter(*X.T, c=y)
plt.show()
```



Relevante Größen für das HCA ist das - *dissimilarity measure*, ein Unterschiedlichkeitsmaß auf

Ebene der Datenpunkte (z.B. euklidische Distanz oder Korrelationskoeffizient) und - *linkage*, ein Unterschiedlichkeitsmaß auf Ebene der Cluster (z.B. complete, single, average).

Beim **Complete Linkage** (maximale Inter-Cluster-Unterschiedlichkeit) werden alle paarweisen Unterschiedlichkeiten zwischen Punkten aus dem ersten und Punkten aus dem zweiten Cluster bestimmt und das Maximum genommen.

Beim **Single Linkage** (minimale Inter-Cluster-Unterschiedlichkeit) wird die minimale paarweise Unterschiedlichkeit zwischen den Punkten aus den Clustern verwendet.

Beim **Average Linkage** (mittlere Inter-Cluster-Unterschiedlichkeit) wird der Mittelwert der paarweisen Unterschiedlichkeiten zwischen den Punkten aus den Clustern verwendet.

```
[3]: def dissimilarity(x1, x2) -> float:
    return np.linalg.norm(x1 - x2)

def linkage(C1, C2, type: str = 'average') -> float:
    diss_pairs: list[float] = []
    for c1 in C1:
        for c2 in C2:
            assert c1 != c2, "Hier ist ein Punkt doppelt!?"
            diss_pairs.append(dissimilarity(c1, c2))

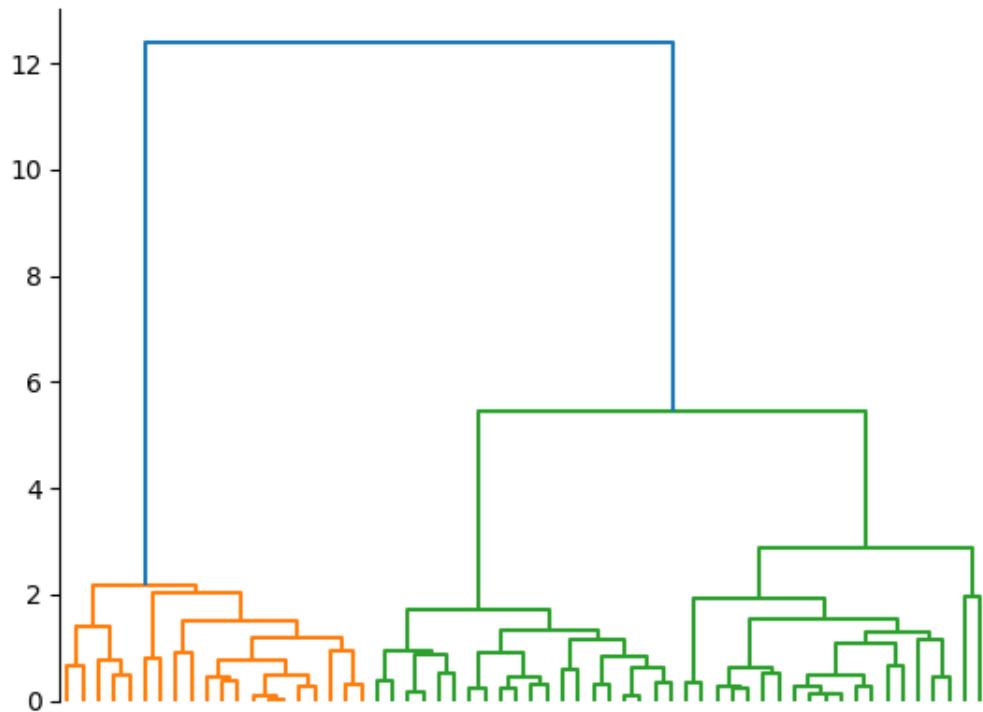
    diss_np: np.ndarray = np.array(diss_pairs)
    if type == 'average':
        return diss_np.mean()
    elif type == 'single':
        return diss_np.min()
    elif type == 'complete':
        return diss_np.max()
```

```
[4]: from scipy.cluster import hierarchy
Z = hierarchy.linkage(X, 'average')

fig, axs = plt.subplots(1, 1)
dn = hierarchy.dendrogram(Z, truncate_mode='none', show_leaf_counts=True, no_labels=True)
# plt.axis('off')

axs.spines['top'].set_visible(False)
axs.spines['right'].set_visible(False)
axs.spines['bottom'].set_visible(False)

plt.show()
```



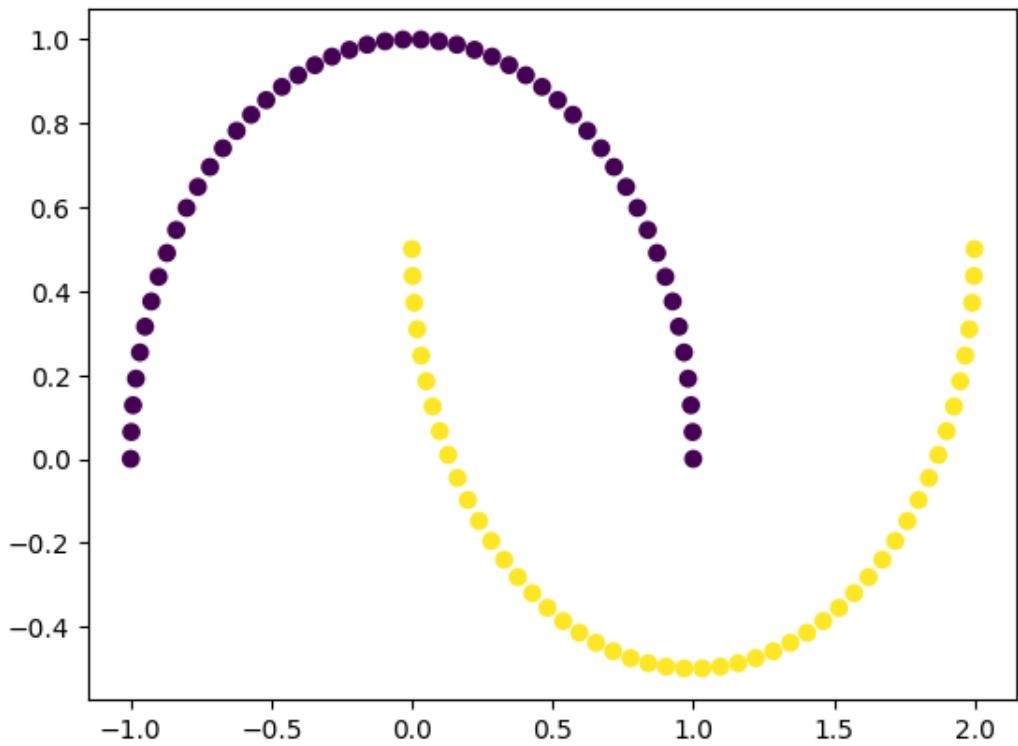
[5]:

```
[5]: array([[1.5000000e+01, 3.1000000e+01, 2.17280603e-02, 2.0000000e+00],  
[7.0000000e+00, 3.4000000e+01, 7.84081035e-02, 2.0000000e+00],  
[1.4000000e+01, 6.0000000e+01, 8.01486344e-02, 3.0000000e+00],  
[4.1000000e+01, 4.3000000e+01, 1.25870441e-01, 2.0000000e+00],  
[3.9000000e+01, 6.3000000e+01, 1.34498817e-01, 3.0000000e+00],  
[9.0000000e+00, 1.0000000e+01, 1.41502638e-01, 2.0000000e+00],  
[5.0000000e+01, 5.2000000e+01, 2.19544043e-01, 2.0000000e+00],  
[2.2000000e+01, 2.8000000e+01, 2.29017234e-01, 2.0000000e+00],  
[2.5000000e+01, 4.5000000e+01, 2.36786207e-01, 2.0000000e+00],  
[5.7000000e+01, 6.4000000e+01, 2.52907290e-01, 4.0000000e+00],  
[1.3000000e+01, 1.6000000e+01, 2.60230122e-01, 2.0000000e+00],  
[4.7000000e+01, 5.9000000e+01, 2.77771987e-01, 2.0000000e+00],  
[2.7000000e+01, 6.6000000e+01, 2.79279277e-01, 3.0000000e+00],  
[1.2000000e+01, 3.3000000e+01, 2.84149341e-01, 2.0000000e+00],  
[0.0000000e+00, 3.8000000e+01, 3.14084714e-01, 2.0000000e+00],  
[5.0000000e+00, 3.7000000e+01, 3.14507818e-01, 2.0000000e+00],  
[2.4000000e+01, 5.5000000e+01, 3.24447961e-01, 2.0000000e+00],  
[3.5000000e+01, 5.1000000e+01, 3.39150555e-01, 2.0000000e+00],  
[1.1000000e+01, 4.6000000e+01, 3.69461861e-01, 2.0000000e+00],  
[2.6000000e+01, 4.2000000e+01, 3.76273196e-01, 2.0000000e+00],  
[6.7000000e+01, 7.3000000e+01, 4.33347292e-01, 4.0000000e+00],
```

```
[1.7000000e+01, 1.8000000e+01, 4.39646057e-01, 2.0000000e+00],
[3.2000000e+01, 7.8000000e+01, 4.47076194e-01, 3.0000000e+00],
[6.2000000e+01, 7.1000000e+01, 4.67010940e-01, 5.0000000e+00],
[8.0000000e+00, 3.6000000e+01, 4.76907410e-01, 2.0000000e+00],
[6.9000000e+01, 7.0000000e+01, 4.91126811e-01, 6.0000000e+00],
[2.3000000e+01, 2.9000000e+01, 4.93573171e-01, 2.0000000e+00],
[2.1000000e+01, 4.4000000e+01, 5.14690628e-01, 2.0000000e+00],
[4.0000000e+00, 5.8000000e+01, 5.76830510e-01, 2.0000000e+00],
[7.2000000e+01, 8.7000000e+01, 6.11822772e-01, 5.0000000e+00],
[6.1000000e+01, 7.6000000e+01, 6.15691415e-01, 4.0000000e+00],
[3.0000000e+00, 4.8000000e+01, 6.43680235e-01, 2.0000000e+00],
[1.0000000e+00, 5.6000000e+01, 6.57439507e-01, 2.0000000e+00],
[8.2000000e+01, 8.3000000e+01, 7.49255371e-01, 8.0000000e+00],
[5.3000000e+01, 8.4000000e+01, 7.73065958e-01, 3.0000000e+00],
[2.0000000e+00, 5.4000000e+01, 7.95153274e-01, 2.0000000e+00],
[7.4000000e+01, 9.0000000e+01, 8.17698606e-01, 6.0000000e+00],
[6.5000000e+01, 8.6000000e+01, 8.46906210e-01, 4.0000000e+00],
[6.8000000e+01, 8.0000000e+01, 8.91659240e-01, 6.0000000e+00],
[3.0000000e+01, 4.9000000e+01, 9.10461022e-01, 2.0000000e+00],
[1.9000000e+01, 7.5000000e+01, 9.26479534e-01, 3.0000000e+00],
[7.9000000e+01, 9.7000000e+01, 9.47594946e-01, 6.0000000e+00],
[8.5000000e+01, 9.2000000e+01, 1.07425674e+00, 8.0000000e+00],
[4.0000000e+01, 8.1000000e+01, 1.14620027e+00, 3.0000000e+00],
[8.8000000e+01, 9.6000000e+01, 1.15873747e+00, 8.0000000e+00],
[9.3000000e+01, 1.0000000e+02, 1.18712115e+00, 1.1000000e+01],
[1.0200000e+02, 1.0300000e+02, 1.29296893e+00, 1.1000000e+01],
[9.8000000e+01, 1.0400000e+02, 1.32422272e+00, 1.4000000e+01],
[9.1000000e+01, 9.4000000e+01, 1.40895941e+00, 5.0000000e+00],
[9.9000000e+01, 1.0500000e+02, 1.49390755e+00, 1.3000000e+01],
[8.9000000e+01, 1.0600000e+02, 1.53631806e+00, 1.6000000e+01],
[1.0100000e+02, 1.0700000e+02, 1.69966602e+00, 2.0000000e+01],
[7.7000000e+01, 1.1000000e+02, 1.93718720e+00, 1.8000000e+01],
[6.0000000e+00, 2.0000000e+01, 1.97184784e+00, 2.0000000e+00],
[9.5000000e+01, 1.0900000e+02, 2.03221798e+00, 1.5000000e+01],
[1.0800000e+02, 1.1400000e+02, 2.15349804e+00, 2.0000000e+01],
[1.1200000e+02, 1.1300000e+02, 2.85779533e+00, 2.0000000e+01],
[1.1100000e+02, 1.1600000e+02, 5.44979660e+00, 4.0000000e+01],
[1.1500000e+02, 1.1700000e+02, 1.24091400e+01, 6.0000000e+01]])
```

```
[6]: from sklearn.datasets import make_moons
[X2, y2] = make_moons(random_state=1)

plt.scatter(*X2.T, c=y2)
plt.show()
```

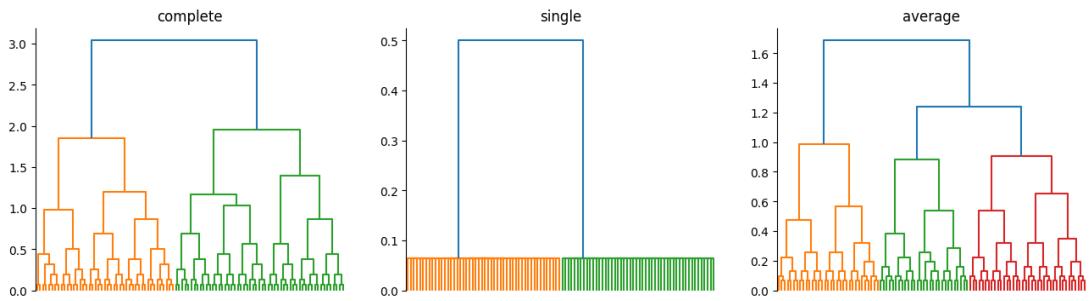


```
[7]: fig, axs = plt.subplots(1, 3, figsize=(16, 4))

for i, m in enumerate(['complete', 'single', 'average']):
    Z = hierarchy.linkage(X2, m)
    hierarchy.dendrogram(Z, truncate_mode='none', show_leaf_counts=True,
                         no_labels=True, ax=axs[i])

    axs[i].set_title(m)
    axs[i].spines['top'].set_visible(False)
    axs[i].spines['right'].set_visible(False)
    axs[i].spines['bottom'].set_visible(False)

plt.show()
```



Nun mit NumPY

```
[8]: from scipy.spatial.distance import cdist

def hca_numpy(X: np.ndarray, K: int, linkage_type: str = 'average'):
    # --- Distanzmatrix ---
    d = cdist(X, X)

    # --- Erstelle n Cluster ---
    y_pred = np.array(range(X.shape[0]))

    while True:
        # --- Performance ---
        C = np.unique(y_pred)

        # --- Wiederhole bis K-Cluster ---
        if C.shape[0] <= K:
            break

        if linkage_type == 'average':
            # Average Linkage
            linkage = np.array([[d[y_pred==i][:,y_pred==j].mean() if i!=j else
                np.inf for j in C] for i in C])
        elif linkage_type == 'single':
            # Single Linkage
            linkage = np.array([[d[y_pred==i][:,y_pred==j].min() if i!=j else
                np.inf for j in C] for i in C])
        elif linkage_type == 'complete':
            # Complete Linkage
            linkage = np.array([[d[y_pred==i][:,y_pred==j].max() if i!=j else
                np.inf for j in C] for i in C])

        # --- Kombiniere gleichartige Cluster ---
        idx = linkage.argmin()
        i = idx % C.shape[0]
```

```

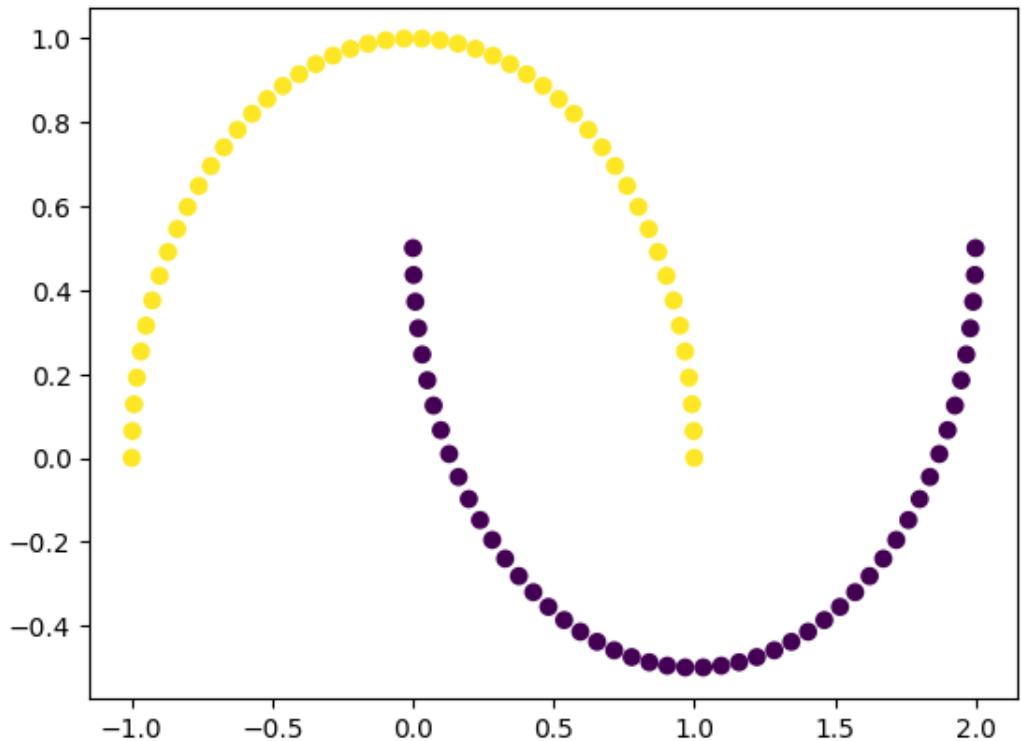
j = idx // C.shape[0]

y_pred[y_pred==C[i]] = C[j]

# --- Visualisierung ---
plt.scatter(*X.T, c=y_pred)
plt.show()

hca_numpy(X2, 2, linkage_type='single')

```



TSRI-9

July 23, 2024

1 Übung 9

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden Sie vertraut mit Clustervalidierungstechniken. Diese sind nützlich, um Parameter (wie beispielsweise die Anzahl der Cluster) einzustellen und die Qualität eines Clusterings zu bewerten.

1.0.2 9.1 Prediction Strength (Bestimmung der Clusteranzahl für K-Means)

Sie haben in der Vorlesung verschiedene Clustervalidierungsverfahren kennengelernt. Die *Prediction Strength* ist eine empfehlenswerte Validierungsmethode, die Sie im Rahmen dieser Übung implementieren und damit Ihr Verständnis vertiefen werden.

Hinweise

- Nutzen Sie die Implementierung des K-Means Algorithmus von Scikit-Learn.
- Für die Implementierung der *Prediction Strength* können Sie numpy, sklearn und scipy verwenden.

Ihre Aufgaben

(1) Schlagen Sie in den Vorlesungsfolien nach, wie die Prediction Strength definiert ist.

Der **Prediction Strength-Ansatz** geht davon aus, dass die Clusterqualität hoch ist, wenn Clusterzugehörigkeiten auf anderen Realisation der Daten zuverlässig vorhergesagt werden können. Dabei wird das Finden richtiger Clustering Parameter als *Model Selection Problem* (wie aus dem Supervised Learning) betrachtet. Dort werden beste Parameter über die Optimierung der Vorhersagen auf Out-of-Sample Daten (Maximierung der prediction strength bzw. Minimierung des Vorhersagefehlers E_{out}) im Rahmen einer Validierung gefunden.

Sei $\mathcal{C} = \{C_1, \dots, C_k\}$ ein Clustering von $\mathcal{D}_{\text{train}}$, deren Datenpunkte in den jeweiligen Regionen R_{C_i} des Merkmalsraum liegen, $\mathcal{A} = \{A_1, \dots, A_K\}$ ein Clustering von \mathcal{D}_{val} und M eine $N_{\text{val}} \times N_{\text{val}}$ -Matrix (sog. Ko-Mitgliedschaftsmatrix) mit

$$M_{ii'} := \begin{cases} 1 & \exists k \text{ mit } (i, i') \in R_{C_k} \\ 0 & \text{sonst} \end{cases}$$

ist. (Ein Matrixeintrag in M ist 1, wenn die jeweiligen zwei Datenpunkte aus dem Validierungsset zur selben Region eines Clusters k im Trainingset gehören.)

Dann ist die *prediction strength*

$$\text{ps}(K) = \min_{j=1, \dots, K} \frac{1}{|A_j|(|A_j| - 1)} \sum_{i, i' \in A_j, i \neq i'} M_{ii'}$$

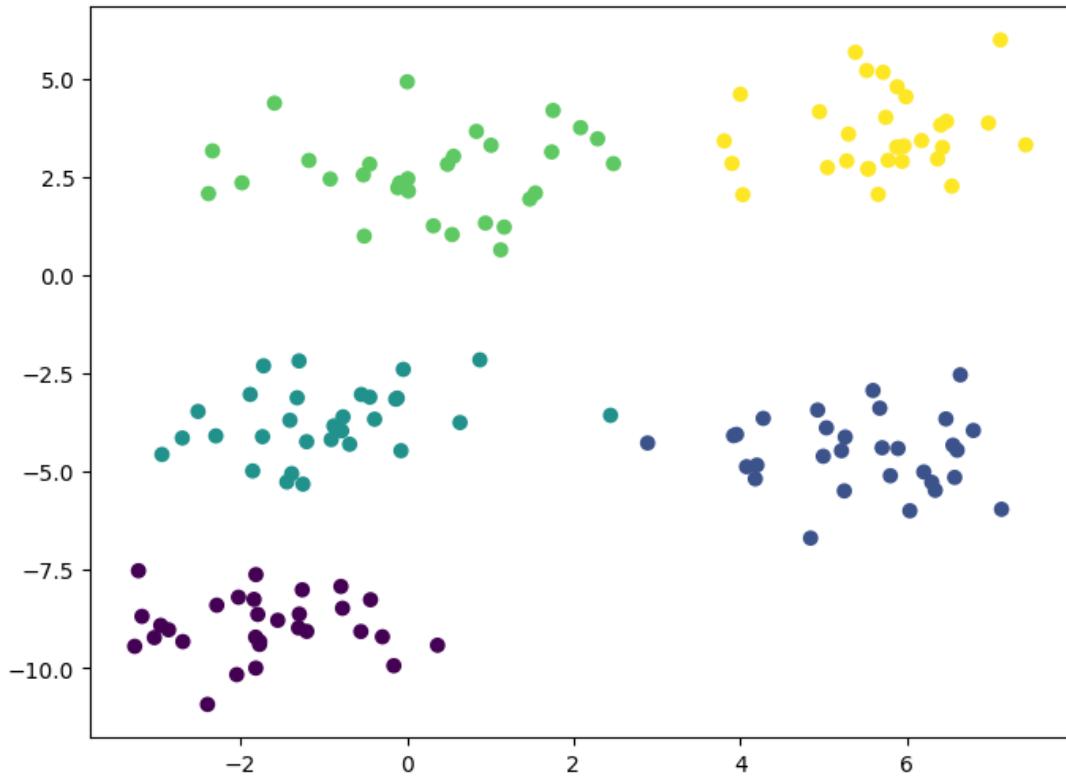
- (2) Ich habe Ihnen synthetische Daten zur Übung bereitgestellt. Bitte führen Sie die unten stehende Code-Zelle aus. Sie erzeugt 150 Datenpunkte mit je zwei Features (Merkmale, Array \mathbf{X}), organisiert in 5 Clustern. Die "wahren" Clusterzugehörigkeiten sind im Vektor \mathbf{y} kodiert.
- In der Praxis haben Sie selbstverständlich keine wahren Clusterzugehörigkeiten vorliegen. Sie wollen vielmehr im Rahmen Ihrer Clusteranalyse diese Clusterzugehörigkeiten erkennen. In dieser Übung nutzen wir \mathbf{y} lediglich für die Visualisierung, die die folgende Codezelle erzeugt, und löschen danach diese Variable wieder.

```
[1]: import numpy as np
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt, rcParams
rcParams['figure.figsize'] = (8, 6)

# generate data
X, y = make_blobs(n_samples=150, centers=5, cluster_std=1, random_state=40)

# visualize data
plt.scatter(x=X[:, 0], y=X[:, 1], c=y)

del y
```



(3) Teilen Sie die Daten zufällig in ein Trainings- und ein Validierungsset auf. Das Validierungsset soll dabei 20% der Gesamtdatenpunkte enthalten, das Trainingsset 80%.

- Ihre Datenvariablen heißen `X_train` und `X_valid`. (Scikit-Learn bezeichnet unser Validierungsset als Testset; lassen Sie sich dadurch nicht verwirren.)
- Nutzen Sie `random_state=40`, damit Sie Ihre Ergebnisse einfach mit denen Ihrer Kolleginnen und Kollegen vergleichen können.

[2] : `from sklearn import model_selection`

```
X_train, X_valid = model_selection.train_test_split(X, test_size=0.2, □
    ↪random_state=40)
```

(4) Für die nachfolgenden (Debugging-)Schritte ist es hilfreich, wenn Sie bereits jetzt mittels K-Means jeweils Cluster im Trainings- und Validierungsset bestimmen.

- Nutzen Sie bei den nachfolgenden Schritten jeweils `random_state=42`, um vergleichbare Ergebnisse wie Ihre Kolleginnen und Kollegen zu erhalten.
1. Ermitteln Sie mittels **K-Means** $K = 8$ Cluster im **Trainingsset** (setzen Sie den Parameter `n_init=10`, um die Warning zu deaktivieren). Speichern Sie die *Centroids* (auch `cluster_centers` genannt) der Cluster in der Variablen `train_centroids`.

2. Ermitteln Sie mittels eines **neuen** K-Means Modells ebenfalls $K = 8$ Cluster im **Validierungsset** (setzen Sie den Parameter `n_init=10`, um die Warning zu deaktivieren). Speichern Sie die Clusterzugehörigkeiten (*cluster labels*) der Daten im Validierungsset in der Variablen `valid_labels`.

```
[3]: from sklearn.cluster import KMeans
kmeans_train = KMeans(n_clusters=8, random_state=42, n_init=10).fit(X_train)
kmeans_valid = KMeans(n_clusters=8, random_state=42, n_init=10).fit(X_valid)
train_centroids = kmeans_train.cluster_centers_
valid_labels = kmeans_valid.labels_
```

- (5) Sie werden die Prediction Strength in mehreren Schritten implementieren. Implementieren Sie zunächst eine Funktion `get_comembership_matrix`, die die Co-Membership Matrix M (Ko-Mitgliedschaftsmatrix) aus den Daten bestimmt.

Hintergrund

Die Matrix M kodiert, welches Paar von Datenpunkten aus dem Validierungsset demselben Cluster im Trainingsset zugeordnet ist. *Damit enthält die Implementierung der Funktion `get_comembership_matrix` implizit unsere Annahmen darüber, wie ein Clustergebiet im Merkmalsraum aussehen sollte.* Verschiedene Clusterverfahren treffen unterschiedliche Annahmen. Daher unterscheidet sich die Implementierung von `get_comembership_matrix` je nach gewähltem Clusteringverfahren!

Wir wollen später Clusterings validieren, die wir mittels K-Means erzeugt haben. Bei K-Means werden Datenpunkte dem Cluster zugeordnet, dessen *Centroid* (Clusterzentrum) sie am nächsten liegen. Um zu bestimmen, ob zwei Datenpunkte aus dem Validierungsset demselben Cluster des Trainingssets angehören, müssen Sie also prüfen, ob Sie demselben Centroid aus dem Trainingsset am nächsten liegen.

Hinweise

- Die Funktion `get_comembership_matrix` nimmt die Daten `X_valid` sowie die `train_centroids` entgegen und gibt die Co-Membership Matrix M zurück.
- Bei der Implementierung können Ihnen Funktionen hilfreich sein, die Sie unterstützen bei (i) der Bestimmung **euklidischer Distanzen**, (ii) bei der Bestimmung des **Arrayindexes mit dem kleinsten Eintrag** sowie (iii) der **Bestimmung aller Indizes von Arrayeinträgen**, die eine Bedingung erfüllen.

```
[4]: from scipy.spatial import distance

# Co-Membership Matirx für K-Means
def get_comembership_matrix(X_valid: np.ndarray, train_centroids: np.ndarray):
    # Distanzmatrix zwischen Validierungsdaten und den Centroids aus dem
    # Trainingsdatenset
    dist_matrix = distance.cdist(X_valid, train_centroids)

    # Clusterzugehörigkeit ergibt sich durch die kleinste Entfernung
    pred_labels = np.argmin(dist_matrix, axis=1)
```

```

# und die Co-Membership-Matrix erhalten wir aus der paarweisen
↪Übereinstimmung der Clusterzugehörigkeiten
M: np.ndarray = (pred_labels[:, np.newaxis] == pred_labels[np.newaxis, :]).
↪astype(int)

return M # (N_val \times N_val)-Matrix

```

- (6) Implementieren Sie nun die Funktion `get_prediction_strength`, die eine Co-Membership Matrix M , die Clusterzugehörigkeiten auf dem Validierungsset `valid_labels` sowie die Anzahl K von Clustern entgegennimmt und die *Prediction Strength* ps zurückliefert.
- Rufen Sie sich dazu noch einmal mithilfe der Vorlesungsfolien in Erinnerung, wie die Prediction Strength definiert ist.
 - Bei der Implementierung kann Ihnen [dies hier](#) hilfreich sein.

$$ps(K) = \min_{j=1, \dots, K} \frac{1}{|A_j|(|A_j| - 1)} \sum_{i, i' \in A_j, i \neq i'} M_{ii'}$$

```
[5]: def get_prediction_strength(M: np.ndarray, valid_labels: np.ndarray, K: int) -> float:
    ps: float = np.inf

    for j in range(K):
        A_j = np.argwhere(valid_labels == j)[:, 0]

        # |A_j|
        N_A_j = A_j.size

        # Keine Division durch 0 oder negative Werte
        if N_A_j <= 1:
            continue

        # Submatrix für die aktuelle Clusterzuordnung
        M_j = M[A_j][:, A_j]

        # Vorhersagestärke für Cluster j:
        ps_j = (np.sum(M_j) - N_A_j) / (N_A_j * (N_A_j - 1))
        ps = min(ps, ps_j)

    return ps if ps != np.inf else 0.0

K = train_centroids.shape[0]
M = get_comembership_matrix(X_valid, train_centroids)
get_prediction_strength(M, valid_labels, K)
```

```
[5]: np.float64(0.3333333333333333)
```

(7) Visualisieren Sie die *Prediction Strength* als Funktion der Clusteranzahl K für $K \in \{1, \dots, 9\}$.

Hierzu müssen Sie für jeden Wert K ein Clustering auf den Trainingsdaten und auf den Validierungsdaten durchführen, die Co-Membership-Matrix M und anschließend die Prediction Strength bestimmen. Beschriften Sie die y-Achse mit “Prediction Strength” und die x-Achse mit “Clusteranzahl”.

```
[6]: def ps(X: np.ndarray, K: int) -> float:
    assert (K >= 1 and K <= X.size * 0.2), "Anzahl der Cluster muss >= 1 sein und darf die Anzahl der Datenpunkte (N_val) nicht übersteigen"

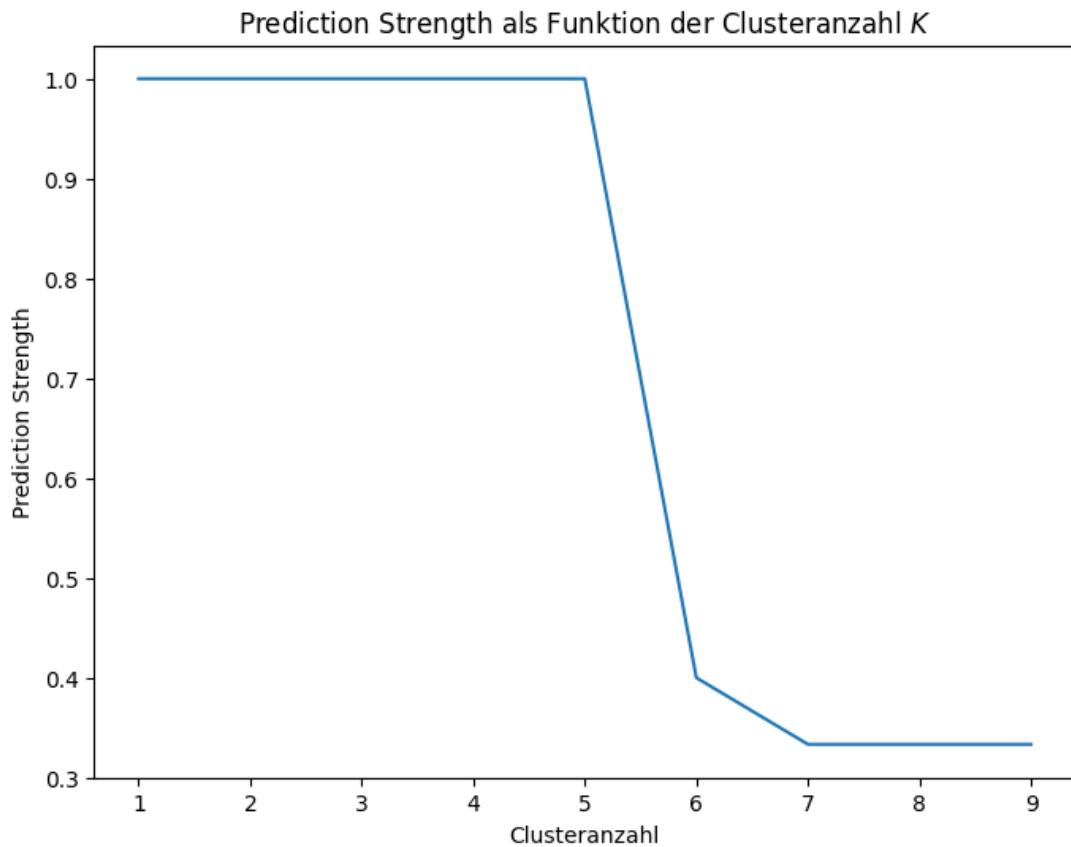
    # Train- / Valid-Split
    X_train, X_valid = model_selection.train_test_split(X, test_size=0.2, random_state=40)

    # Erstelle Clustering
    kmeans_train = KMeans(n_clusters=K, random_state=42, n_init=10).fit(X_train)
    valid_labels = KMeans(n_clusters=K, random_state=42, n_init=10).fit_predict(X_valid)
    train_centroids = kmeans_train.cluster_centers_

    # Berechne Co-Membership Matrix M
    M = get_comembership_matrix(X_valid, train_centroids)

    # Berechne Prediction Strength
    return get_prediction_strength(M, valid_labels, K)

k_range = range(1, 10)
plt.plot(k_range, [ps(X, K) for K in k_range])
plt.title("Prediction Strength als Funktion der Clusteranzahl $K$")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```



- (8) Lesen Sie die optimale Anzahl an Clustern anhand Ihrer Visualisierung aus Schritt (7) ab und geben Sie sie hier an. Die optimale Clusteranzahl ist die größte Zahl $K > 1$, für die die Prediction Strength noch den größten Wert annimmt.
- Stimmt die von Ihnen ermittelte Clusteranzahl mit der “wahren” Anzahl der Cluster (vgl. Schritt (1)) in den Daten überein?

Die optimale Clusteranzahl ist bei $K = 5$, was deckungsgleich mit der erwarteten Clusteranzahl aus der Abbildung ist.

- (9) [Optional] In der Praxis wird oft die Prediction Strength im Rahmen einer sogenannten Kreuzvalidierung ermittelt. Dabei werden alle Daten X in V disjunkte Mengen (sogenannte *Folds*) aufgespalten, wobei die Daten aus $(V - 1)$ Folds als Trainingsset und dem verbliebenen Fold als Validierungsset angesehen wird. Durch Durchtauschen der Zuordnungen der Folds zu Trainings- und Validierungsset lassen sich so V mal die Prediction Strength bestimmen (jedes Fold dient einmal als Validierungsdatensatz). Die V Werte für die Prediction Strength werden anschließend arithmetisch gemittelt.
- Bestimmen Sie für $K \in \{1, \dots, 9\}$ mittels 5-facher Kreuzvalidierung die Prediction Strength und visualisieren Sie sie. Dabei kann Ihnen ggf. [diese Funktion](#) hilfreich sein.

```
[7]: from sklearn.model_selection import KFold

def ps_KFold(X: np.ndarray, K: int, V: int, random_state=40) -> float:
    assert (K >= 1 and K <= X.size / 5), "Anzahl der Cluster muss >= 1 sein und darf die Anzahl der Datenpunkte (N_val) nicht übersteigen"

    kf = KFold(n_splits=V, shuffle=True, random_state=random_state)

    ps_sum: float = 0.0
    for train_index, test_index in kf.split(X):
        X_train, X_valid = X[train_index], X[test_index]

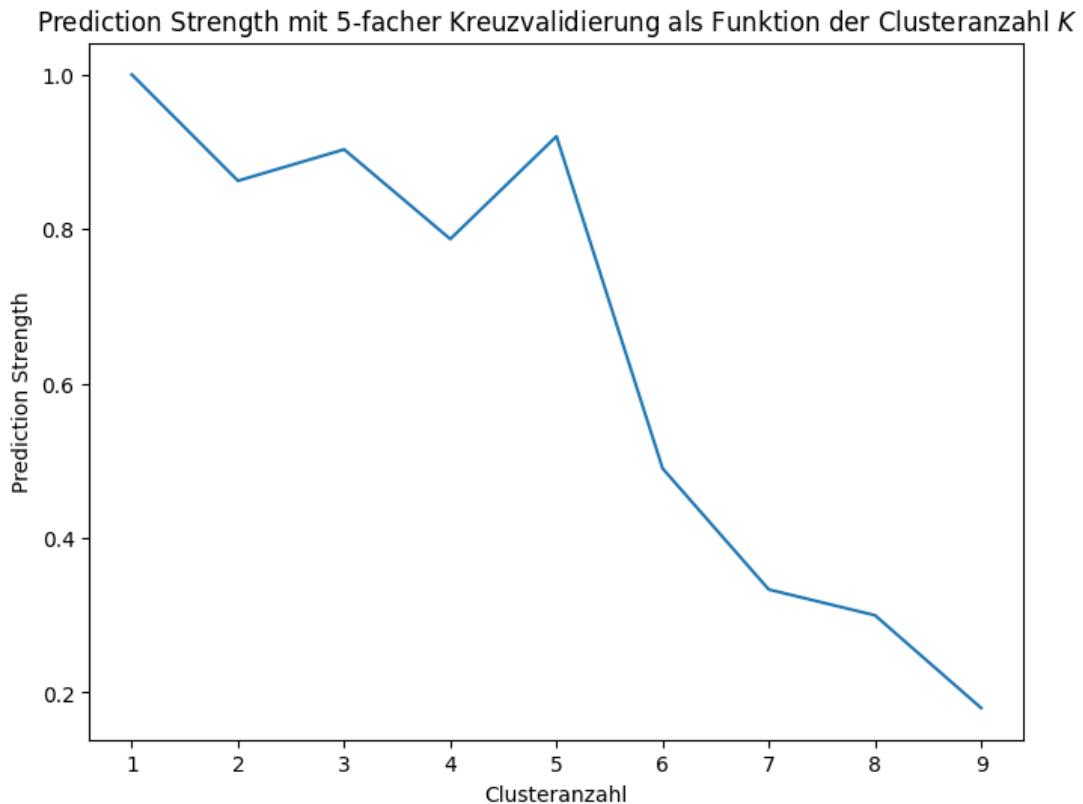
        # Erstelle Clustering
        kmeans_train = KMeans(n_clusters=K, random_state=42, n_init=10).fit(X_train)
        valid_labels = KMeans(n_clusters=K, random_state=42, n_init=10).fit_predict(X_valid)
        train_centroids = kmeans_train.cluster_centers_

        # Berechne Co-Membership Matrix M
        M = get_comembership_matrix(X_valid, train_centroids)

        # Berechne Prediction Strength
        ps_sum += get_prediction_strength(M, valid_labels, K)

    # Arithmetisches Mittel der Werte, die für die Prediction Strength ermittelt wurden
    return ps_sum / V

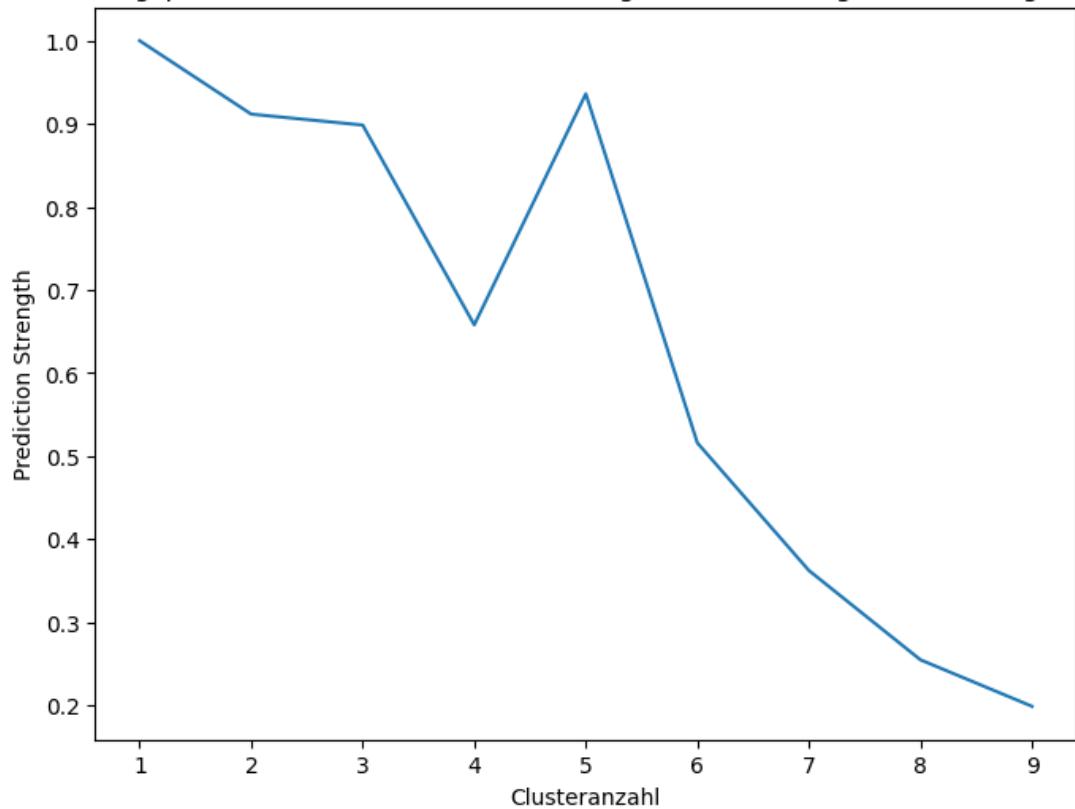
k_range = range(1, 10)
plt.plot(k_range, [ps_KFold(X, K, 5, random_state=35) for K in k_range])
plt.title("Prediction Strength mit 5-facher Kreuzvalidierung als Funktion der Clusteranzahl $K$")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```



Frage: Der Prediction Strength Graph der Kreuzvalidierung variiert stark. Wäre es da nicht sinnvoll, diese mehrfach mit zufälligem Split durchzuführen, um bessere Ergebnisse zu erhalten?

```
[8]: k_range = range(1, 10)
plt.plot(k_range, [np.array([ps_KFold(X, K, 5, random_state=i) for i in range(15)]).mean() for K in k_range])
plt.title("Avg. $ps(K)$ mit 5-facher Kreuzvalidierung nach 15 zufälligen Anordnungen")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```

Avg. $ps(K)$ mit 5-facher Kreuzvalidierung nach 15 zufälligen Anordnungen



TSRI-10

July 23, 2024

1 Übung 10

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden wir intensiv mit einem Datensatz arbeiten und uns mit ihm vertraut machen.

1.0.2 10.1 Datenaufbereitung (Corona Pandemie)

In dieser Übungsaufgabe wird es darum gehen, weitere Praxiserfahrung bei der Datenaufbereitung zu sammeln. Wie Sie im Rahmen der Veranstaltung gelernt haben, lässt sich Datenaufbereitung nur schwer auf Vorlesungsfolien vermitteln, sondern muss praktisch geübt werden. Sie werden Daten der Corona-Pandemie analysieren. Die Daten stammen vom Robert Koch Institut.

- Ziel dieser Übung ist es, den Datensatz des Robert Koch Instituts so aufzubereiten, dass Sie am Ende einen Pandas DataFrame vorliegen haben, der die Anzahl Neuinfizierter pro Tag und Bundesland enthält.

Ihre Aufgaben

- (1) Da der Datensatz des Robert Koch Instituts mittlerweile leider nicht mehr online verfügbar ist, haben wir Ihnen [hier](#) einen Export der Daten (aus dem März 2022) hinterlegt.
 - Die heruntergeladene Datei ist mehrere hundert Megabytes groß. Um Plattenplatz zu sparen, ist es empfehlenswert, die Datei in ihrem gezippten Format zu belassen. Die Pandas-Bibliothek, mit der wir die Daten später untersuchen werden, kann [gezippte Dateien direkt importieren](#).
- (2) Importieren Sie die Daten in einen Pandas DataFrame, den Sie `df_germany` nennen.

```
[1]: import pandas as pd
```

```
df_germany: pd.DataFrame = pd.read_csv("RKI_COVID-19.zip")
```

- (3) Untersuchen Sie den importierten DataFrame:
 - Welche Spalten weist der DataFrame auf?

- Welche Information enthält eine Zeile des DataFrames?
- In welcher Spalte ist die Information über die Anzahl der Neuinfizierten enthalten?

```
[2]: print(f'Der DataFrame hat die folgenden Spalten:\n{list(df_germany.columns)}')
print(f'und die Zeilen sind Meldungen an das RKI, wobei die Spalte `AnzahlFall` die
die Anzahl der Neuinfizierten seit der letzten Meldung enthält.')
```

Der DataFrame hat die folgenden Spalten:

['IdBundesland', 'Bundesland', 'IdLandkreis', 'Landkreis', 'Altersgruppe',
 'Altersgruppe2', 'Geschlecht', 'Meldedatum', 'Refdatum', 'IstErkrankungsbeginn',
 'NeuerFall', 'NeuerTodesfall', 'NeuGenesen', 'AnzahlFall', 'AnzahlTodesfall',
 'AnzahlGenesen', 'Datenstand', 'ObjectId']

und die Zeilen sind Meldungen an das RKI, wobei die Spalte `AnzahlFall` die Anzahl der Neuinfizierten seit der letzten Meldung enthält.

Die Fallzahlendaten enthalten die in der folgenden Tabelle abgebildeten Merkmale und deren Ausprägungen:

Merkmal	Ausprägung	Erläuterung
IdLandkreis	1001 bis 16077	Identifikationsnummer des Landkreises basierend auf dem Amtlichen Gemeindeschlüssel (AGS) zuzüglich der 12 Bezirke Berlins (11001 bis 11012); Gebietsstand: 30.06.2020 (2. Quartal)
Geschlecht	W, M, unbekannt	Geschlecht der Fallgruppe: weiblich (W), männlich (M) und (unbekannt)
Altersgruppe	A00-A04, A05-A14, A15-A34, A35-A59, A60-A79, A80+, unbekannt	Altersspanne der in der Gruppe enthaltenen Fälle, stratifiziert nach 0-4 Jahren, 5-14 Jahren, 15-34 Jahren, 35-59 Jahren, 60-79 Jahren, 80+ Jahren sowie unbekannt
Meldedatum	JJJJ-MM-TT	Datum, wann der Fall dem Gesundheitsamt bekannt geworden ist. JJJJ entspricht der Jahreszahl, MM dem Monat und TT dem Tag.
Refdatum	JJJJ-MM-TT	Datum des Erkrankungsbeginns. Wenn das nicht bekannt ist, das Meldedatum.
IstErkrankungsbeginn	0, 1	1: Refdatum ist der Erkrankungsbeginn 0: Refdatum ist das Meldedatum

Merkmal	Ausprägung	Erläuterung
AnzahlFall	Ganze Zahl	Anzahl der gemeldeten Fälle in der entsprechenden Fallgruppe Für NeuerFall = -1, ist die Anzahl negativ: Es handelt sich um eine Korrektur der Fallgruppe, die angibt, wie viele Infektionen zu viel gemeldet worden sind
AnzahlTodesfall	Ganze Zahl	Anzahl der gemeldeten Todesfälle in der entsprechenden Fallgruppe Für NeuerTodesfall = -1, ist die Anzahl negativ: Es handelt sich um eine Korrektur der Fallgruppe, die angibt, wie viele Todesfälle zu viel gemeldet worden sind
AnzahlGenesen	Ganze Zahl	Anzahl der genesenen Fälle in der entsprechenden Fallgruppe Für NeuGenesen = -1, ist die Anzahl negativ: Es handelt sich um eine Korrektur der Fallgruppe, die angibt, wie viele genesene Fälle zu viel gemeldet worden sind

Merkmal	Ausprägung	Erläuterung
NeuerFall, NeuerTodesfall, NeuGenesen	0, 1, -1	0 : Fälle der Gruppe sind in der Publikation für den aktuellen Tag und in der für den Vortag enthalten. Das bedeutet diese Fälle sind seit mehr als einem Tag bekannt. 1 : Fälle der Gruppe sind erstmals in der aktuellen Publikation enthalten. Das heißt, es sind für den Publikationstag neu übermittelte oder entsprechend neu bewertete Fälle. -1: Fälle der Gruppe sind in der Publikation des Vortags enthalten, werden jedoch nach dem aktuellen Tag aus den Fallzahlendaten entfernt. Das heißt, es sind Fälle die ab dem aktuellen Tag wegfallen. Eine solche Fallgruppe kann beispielsweise durch fälschliche Meldungen entstehen, die so als Korrektur angezeigt werden.
NeuerTodesfall, NeuGenesen	-9	Fälle in der Gruppe sind weder in der Publikation für den aktuellen Tag, noch in der Publikation des Vortags, als genesen (“NeuGenesen”) oder verstorben (“NeuerTodesfall”) gemeldet. Das bedeutet, dass zu den Fällen in der Gruppe keine Information über den Gesundheitsverlauf der Infektion bekannt ist. Das ist zum Beispiel häufig der Fall, wenn eine Fallgruppe gerade erst als infiziert gemeldet worden ist.

Quelle: Robert Koch-Institut (2023): SARS-CoV-2 Infektionen in Deutschland, Berlin: Zenodo. DOI:10.5281/zenodo.4681153.

Die Anzahl der Neuinfektionen ist also die `AnzahlFall` für die Fallgruppe in der `NeuerFall == 1` ist.

(4) Setzen Sie die Spalte `Meldedatum` auf den Typ datetime Index.

```
[3]: df_germany["Meldedatum"] = pd.to_datetime(df_germany["Meldedatum"])
df_germany.sort_values("Meldedatum", inplace=True)
df_germany[df_germany["NeuerFall"] == 1]
```

	IdBundesland	Bundesland	IdLandkreis	\
779469	5	Nordrhein-Westfalen	5162	
2326835	8	Baden-Württemberg	8336	
1947044	8	Baden-Württemberg	8115	
1954754	8	Baden-Württemberg	8115	
1999774	8	Baden-Württemberg	8118	
...	
1727733	7	Rheinland-Pfalz	7133	
4037517	14	Sachsen	14626	
1728505	7	Rheinland-Pfalz	7133	
37710	1	Schleswig-Holstein	1053	
4447963	16	Thüringen	16077	

	Landkreis	Altersgruppe	Altersgruppe2	Geschlecht	\
779469	LK Rhein-Kreis Neuss	A35-A59	Nicht übermittelt	W	
2326835	LK Lörrach	A35-A59	Nicht übermittelt	W	
1947044	LK Böblingen	A05-A14	Nicht übermittelt	M	
1954754	LK Böblingen	A35-A59	Nicht übermittelt	W	
1999774	LK Ludwigsburg	A15-A34	Nicht übermittelt	W	
...	
1727733	LK Bad Kreuznach	A80+	Nicht übermittelt	M	
4037517	LK Görlitz	A60-A79	Nicht übermittelt	M	
1728505	LK Bad Kreuznach	A60-A79	Nicht übermittelt	W	
37710	LK Herzogtum Lauenburg	A05-A14	Nicht übermittelt	W	
4447963	LK Altenburger Land	unbekannt	Nicht übermittelt	W	

	Meldedatum	Refdatum	\
779469	2020-03-28 00:00:00+00:00	2020/03/28 00:00:00+00	
2326835	2020-07-13 00:00:00+00:00	2020/07/06 00:00:00+00	
1947044	2020-07-18 00:00:00+00:00	2020/07/18 00:00:00+00	
1954754	2020-10-07 00:00:00+00:00	2020/10/07 00:00:00+00	
1999774	2020-10-10 00:00:00+00:00	2020/10/10 00:00:00+00	
...	
1727733	2022-03-21 00:00:00+00:00	2022/03/21 00:00:00+00	
4037517	2022-03-21 00:00:00+00:00	2022/03/21 00:00:00+00	
1728505	2022-03-21 00:00:00+00:00	2022/03/21 00:00:00+00	
37710	2022-03-21 00:00:00+00:00	2022/03/21 00:00:00+00	
4447963	2022-03-21 00:00:00+00:00	2022/03/21 00:00:00+00	

	IstErkrankungsbeginn	NeuerFall	NeuerTodesfall	NeuGenesen	\
779469	0	1	-9	1	

2326835	1	1	-9	1
1947044	0	1	-9	1
1954754	0	1	-9	1
1999774	0	1	-9	1
...
1727733	0	1	-9	-9
4037517	0	1	-9	-9
1728505	0	1	-9	-9
37710	0	1	-9	-9
4447963	0	1	-9	-9
AnzahlFall	AnzahlTodesfall	AnzahlGenesen	Datenstand	\
779469	1	0	1	22.03.2022, 00:00 Uhr
2326835	1	0	1	22.03.2022, 00:00 Uhr
1947044	1	0	1	22.03.2022, 00:00 Uhr
1954754	1	0	1	22.03.2022, 00:00 Uhr
1999774	1	0	1	22.03.2022, 00:00 Uhr
...
1727733	7	0	0	22.03.2022, 00:00 Uhr
4037517	2	0	0	22.03.2022, 00:00 Uhr
1728505	46	0	0	22.03.2022, 00:00 Uhr
37710	2	0	0	22.03.2022, 00:00 Uhr
4447963	1	0	0	22.03.2022, 00:00 Uhr
ObjectId				
779469	779470			
2326835	2326836			
1947044	1947045			
1954754	1954755			
1999774	1999775			
...	...			
1727733	1727734			
4037517	4037518			
1728505	1728506			
37710	37711			
4447963	4447964			

[17031 rows x 18 columns]

(5) Erstellen Sie aus dem alten DataFrame einen neuen DataFrame **covid**, der folgende Spalten enthält:

- Erste Spalte: **Meldedatum**
- 16 weitere Spalten, die jeweils nach einem Bundesland benannt sind
- letzte Spalte: **Deutschland**

Der DataFrame enthält also die Anzahl der Neuinfizierten pro Tag (Meldedatum) und Bundesland sowie für die gesamte Bundesrepublik (Spalte Deutschland).

- Beachten Sie: Vermutlich werden Sie für die Lösung dieser Teilaufgabe etwas mehr Zeit benötigen und im Internet in der Pandas Dokumentation recherchieren müssen.

```
[4]: bundeslaender = df_germany["Bundesland"].unique().tolist()
covid: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuerFall"] == 1],  
    values="AnzahlFall", index="Melde datum", columns="Bundesland", aggfunc="sum")
covid["Deutschland"] = covid[bundeslaender].sum(axis="columns")
covid.head(10)
```

Bundesland	Baden-Württemberg	Bayern	Berlin	Brandenburg	\
Melde datum					
2020-03-28 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-07-13 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-07-18 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-10-07 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-10-10 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-10-27 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-11-05 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-13 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-11-27 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-12-02 00:00:00+00:00	1.0	NaN	NaN	NaN	

Bundesland	Bremen	Hamburg	Hessen	Mecklenburg-Vorpommern	\
Melde datum					
2020-03-28 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-07-13 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-07-18 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-07 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-10 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-27 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-05 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-13 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-27 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-12-02 00:00:00+00:00	NaN	NaN	NaN	NaN	

Bundesland	Niedersachsen	Nordrhein-Westfalen	\
Melde datum			
2020-03-28 00:00:00+00:00	NaN	1.0	
2020-07-13 00:00:00+00:00	NaN	NaN	
2020-07-18 00:00:00+00:00	NaN	NaN	
2020-10-07 00:00:00+00:00	NaN	NaN	
2020-10-10 00:00:00+00:00	NaN	NaN	
2020-10-27 00:00:00+00:00	NaN	NaN	
2020-11-05 00:00:00+00:00	NaN	NaN	
2020-11-13 00:00:00+00:00	NaN	NaN	
2020-11-27 00:00:00+00:00	NaN	NaN	
2020-12-02 00:00:00+00:00	NaN	NaN	

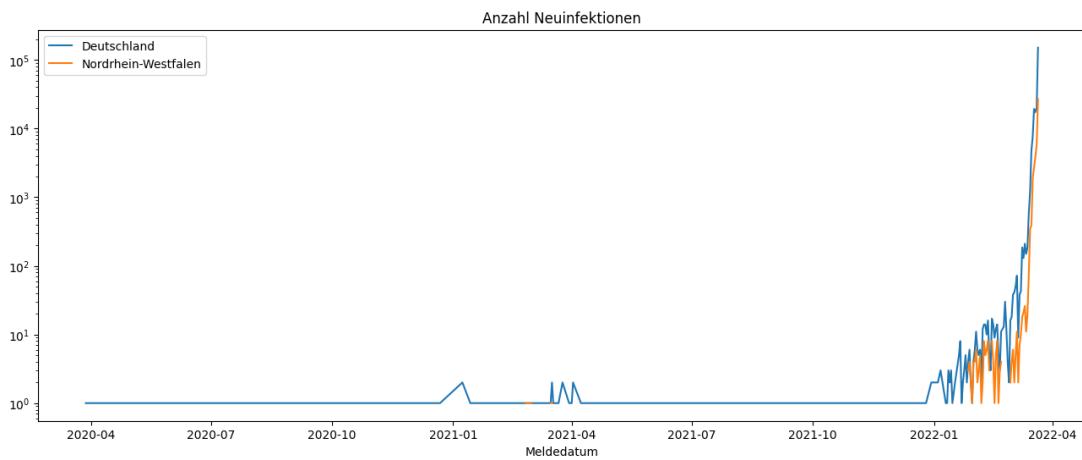
Bundesland	Rheinland-Pfalz	Saarland	Sachsen	Sachsen-Anhalt	\
Meldedatum					
2020-03-28 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-07-13 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-07-18 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-07 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-10 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-10-27 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-05 00:00:00+00:00	1.0	NaN	NaN	NaN	
2020-11-13 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-11-27 00:00:00+00:00	NaN	NaN	NaN	NaN	
2020-12-02 00:00:00+00:00	NaN	NaN	NaN	NaN	

Bundesland	Schleswig-Holstein	Thüringen	Deutschland
Meldedatum			
2020-03-28 00:00:00+00:00	NaN	NaN	1.0
2020-07-13 00:00:00+00:00	NaN	NaN	1.0
2020-07-18 00:00:00+00:00	NaN	NaN	1.0
2020-10-07 00:00:00+00:00	NaN	NaN	1.0
2020-10-10 00:00:00+00:00	NaN	NaN	1.0
2020-10-27 00:00:00+00:00	NaN	NaN	1.0
2020-11-05 00:00:00+00:00	NaN	NaN	1.0
2020-11-13 00:00:00+00:00	NaN	NaN	1.0
2020-11-27 00:00:00+00:00	NaN	NaN	1.0
2020-12-02 00:00:00+00:00	NaN	NaN	1.0

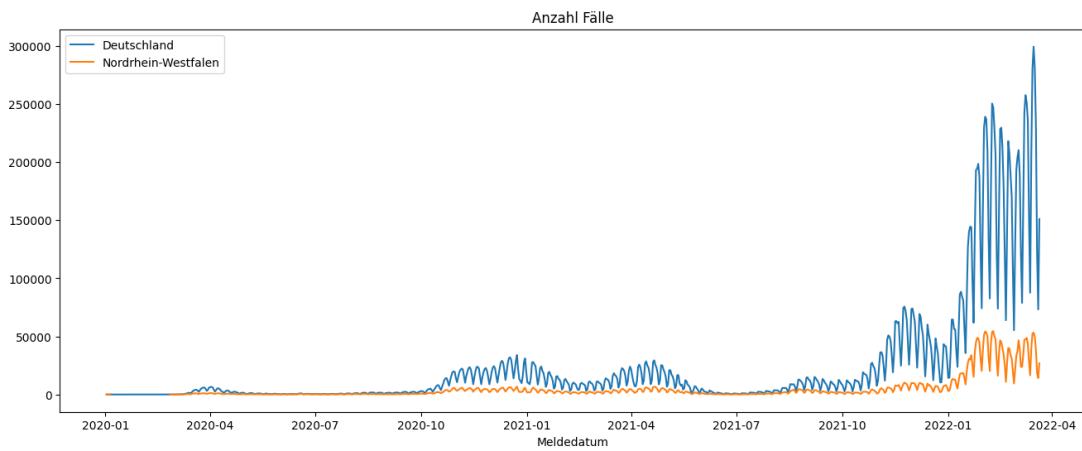
- (6) Visualisieren Sie die Anzahl der Neuinfizierten in Nordrhein-Westfalen sowie in der gesamten Bundesrepublik als Funktion der Zeit.

- Die x-Achse ist mit "Meldedatum" beschriftet.
- Auf der x-Achse stehen Datumsangaben (z.B: '2020-03-17'), keine bloßen Integers.
- Der Titel der Abbildung lautet "Anzahl Neuinfektionen".

```
[5]: import matplotlib.pyplot as plt
plt.figure(figsize=(16, 6))
plt.xlabel("Meldedatum")
plt.yscale("log")
plt.title("Anzahl Neuinfektionen")
plt.plot(covid["Deutschland"], label="Deutschland")
plt.plot(covid["Nordrhein-Westfalen"], label="Nordrhein-Westfalen")
plt.legend()
plt.show()
```



```
[6]: covid: pd.DataFrame = pd.pivot_table(df_germany, values="AnzahlFall",  
    ↪index="Meldedatum", columns="Bundesland", aggfunc="sum")  
covid["Deutschland"] = covid[bundeslaender].sum(axis="columns")  
  
plt.figure(figsize=(16, 6))  
plt.xlabel("Meldedatum")  
plt.title("Anzahl Fälle")  
plt.plot(covid["Deutschland"], label="Deutschland")  
plt.plot(covid["Nordrhein-Westfalen"], label="Nordrhein-Westfalen")  
plt.legend()  
plt.show()
```



1.0.3 10.2 Exploration (Corona Pandemie)

Sie werden die Daten aus der vorangegangenen Aufgabe in dieser Aufgabe analysieren.

- Diese Aufgabe ist freier gehalten als die vorangegangene Aufgabe.

Schritte

- (1) Importieren Sie die Daten.

```
[7]: df_germany: pd.DataFrame = pd.read_csv("RKI_COVID-19.zip")

df_germany["Meldedatum"] = pd.to_datetime(df_germany["Meldedatum"])
df_germany.set_index("Meldedatum", inplace=True)

[8]: # Quelle: https://github.com/owid/covid-19-data/tree/master/public/data
#!wget https://covid.ourworldindata.org/data/owid-covid-data.csv

[9]: df_owid: pd.DataFrame = pd.read_csv("owid-covid-data.csv")
df_owid["date"] = pd.to_datetime(df_owid["date"])

[10]: df_owid.columns
```

[10]: Index(['iso_code', 'continent', 'location', 'date', 'total_cases', 'new_cases', 'new_cases_smoothed', 'total_deaths', 'new_deaths', 'new_deaths_smoothed', 'total_cases_per_million', 'new_cases_per_million', 'new_cases_smoothed_per_million', 'total_deaths_per_million', 'new_deaths_per_million', 'new_deaths_smoothed_per_million', 'reproduction_rate', 'icu_patients', 'icu_patients_per_million', 'hosp_patients', 'hosp_patients_per_million', 'weekly_icu_admissions', 'weekly_icu_admissions_per_million', 'weekly_hosp_admissions', 'weekly_hosp_admissions_per_million', 'total_tests', 'new_tests', 'total_tests_per_thousand', 'new_tests_per_thousand', 'new_tests_smoothed', 'new_tests_smoothed_per_thousand', 'positive_rate', 'tests_per_case', 'tests_units', 'total_vaccinations', 'people_vaccinated', 'people_fully_vaccinated', 'total_boosters', 'new_vaccinations', 'new_vaccinations_smoothed', 'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred', 'total_boosters_per_hundred', 'new_vaccinations_smoothed_per_million', 'new_people_vaccinated_smoothed', 'new_people_vaccinated_smoothed_per_hundred', 'stringency_index', 'population_density', 'median_age', 'aged_65_older', 'aged_70_older', 'gdp_per_capita', 'extreme_poverty', 'cardiovasc_death_rate', 'diabetes_prevalence', 'female_smokers', 'male_smokers', 'handwashing_facilities', 'hospital_beds_per_thousand', 'life_expectancy', 'human_development_index', 'population', 'excess_mortality_cumulative_absolute', 'excess_mortality_cumulative', 'excess_mortality', 'excess_mortality_cumulative_per_million'],
dtype='object')

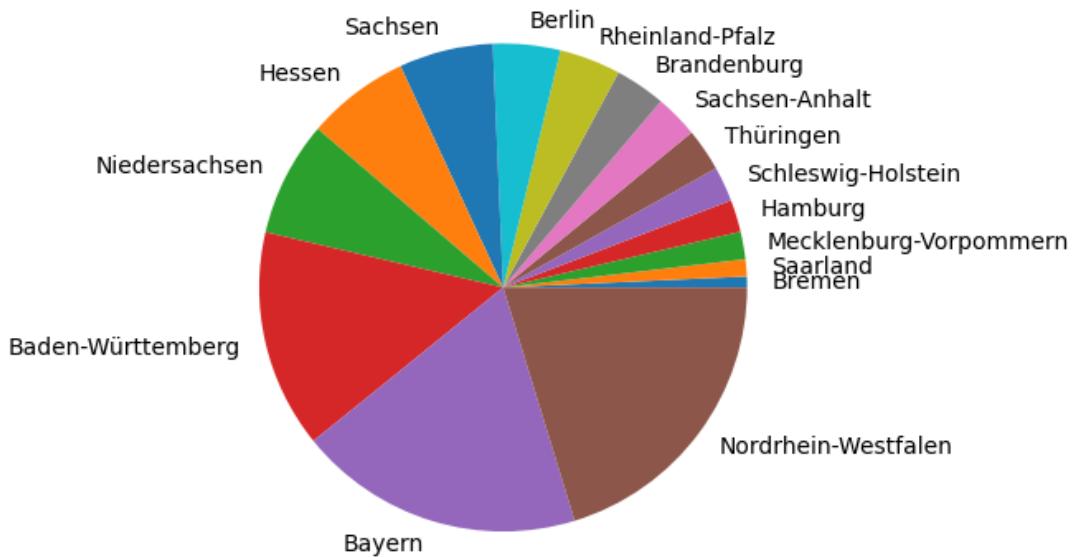
- (2) Untersuchen Sie die Daten: **Erzeugen Sie Fragen** an die Daten und notieren Sie sie hier.

- Welche Bundesländer hatten die meisten Fälle?
- Wie oft kam es zu Falschmeldungen / Korrekturen?
- Wie verhalten sich die Fallzahlen im Verlauf der Pandemie zu den Todeszahlen und den Zahlen der Genesenen?

(3) **Untersuchen** Sie Ihre Fragen mit den angegebenen Daten und **visualisieren** Sie bzw. **beschreiben** Sie Ihre Erkenntnisse.

- Hinweis: Ich lade Sie herzlich dazu ein, weitere Datenquellen aus dem Netz zu Ihren Untersuchungen hinzuziehen. Geben Sie in diesem Falle Ihre Datenquellen hier an.

```
[11]: covid.sum()[:-1].sort_values().plot(kind="pie")
plt.show()
```



```
[12]: print(df_germany[df_germany["NeuerFall"] == -1]["AnzahlFall"].sum())
print(df_germany[df_germany["NeuGenesen"] == -1]["AnzahlGenesen"].sum())
print(df_germany[df_germany["NeuerTodesfall"] == -1]["AnzahlTodesfall"].sum())
```

```
-1137
-661
-3
```

Die Menge der korrigierten Fälle ist überschaubar und fällt in der Gesamtbetrachtung kaum auf.

```
[13]: bundeslaender = df_germany["Bundesland"].unique().tolist()
```

```

covid_fall: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuerFall"] != 0], values="AnzahlFall", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_fall["Deutschland"] = covid_fall[bundeslaender].sum(axis="columns")

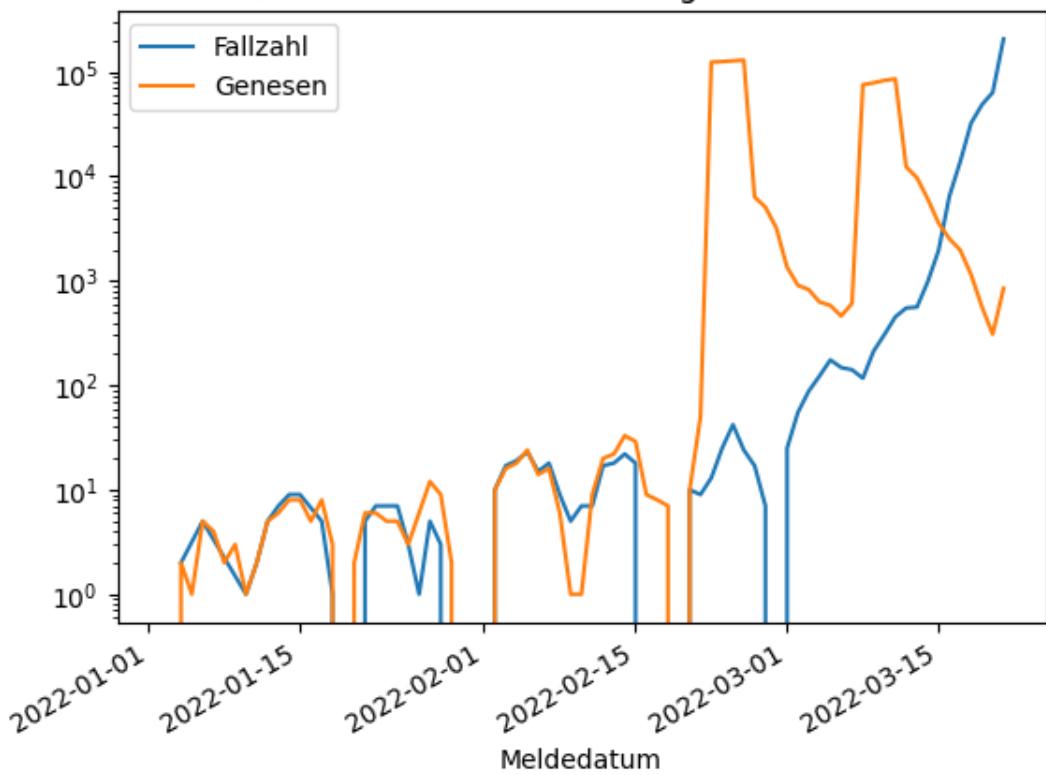
covid_tod: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuerTodesfall"] != 0], values="AnzahlTodesfall", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_tod["Deutschland"] = covid_tod[bundeslaender].sum(axis="columns")

covid_genesen: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuGenesen"] != 0], values="AnzahlGenesen", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_genesen["Deutschland"] = covid_genesen[bundeslaender].sum(axis="columns")

startdatum = "2022-01-01"
plt.title(f"Erkrankte/Genesene der letzten 4 Tage seit dem {startdatum}")
covid_fall[covid_fall.index > startdatum].rolling('4D')["Deutschland"].sum().plot(label="Fallzahl")
covid_genesen[covid_genesen.index > startdatum].rolling('4D')["Deutschland"].sum().plot(label="Genesen")
plt.yscale("log")
plt.legend()
plt.show()

```

Erkrankte/Genesene der letzten 4 Tage seit dem 2022-01-01



```
[14]: bundeslaender = df_germany["Bundesland"].unique().tolist()

covid_fall: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuerFall"] != 0], values="AnzahlFall", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_fall["Deutschland"] = covid_fall[bundeslaender].sum(axis="columns")

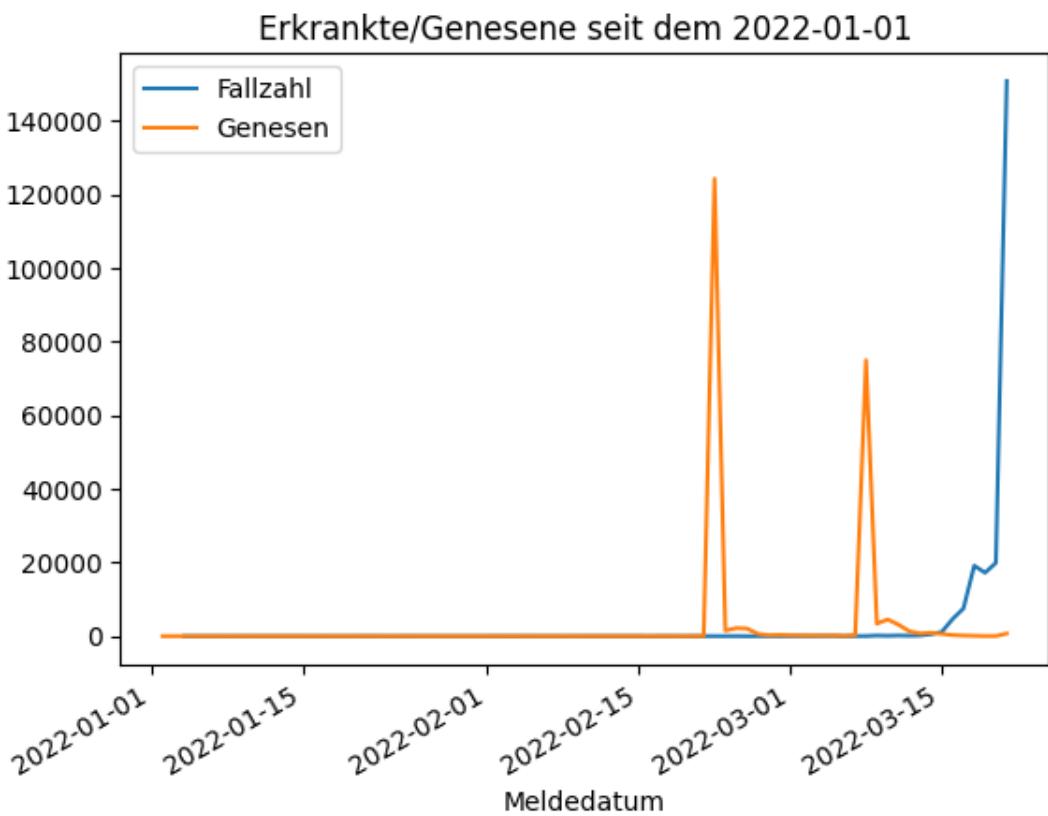
covid_tod: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuerTodesfall"] != 0], values="AnzahlTodesfall", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_tod["Deutschland"] = covid_tod[bundeslaender].sum(axis="columns")

covid_genesen: pd.DataFrame = pd.pivot_table(df_germany[df_germany["NeuGenesen"] != 0], values="AnzahlGenesen", index="Meldedatum", columns="Bundesland", aggfunc="sum")
covid_genesen["Deutschland"] = covid_genesen[bundeslaender].sum(axis="columns")
```

```

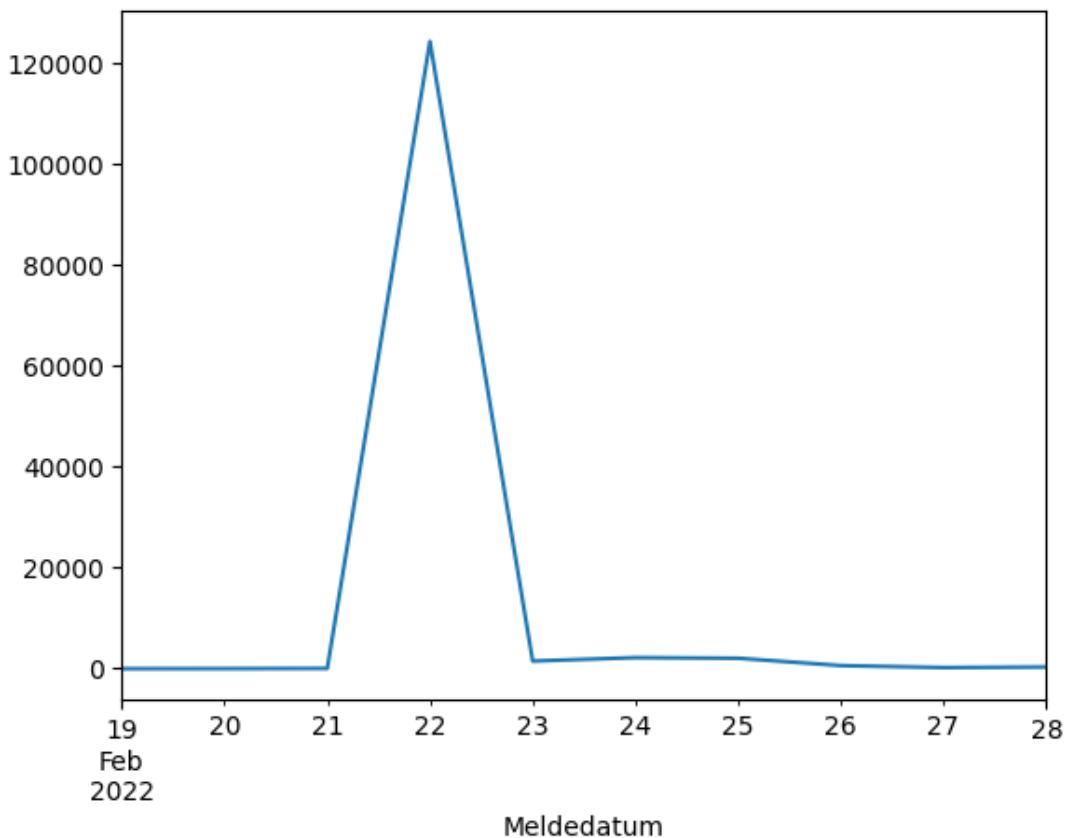
startdatum = "2022-01-01"
plt.title(f"Erkrankte/Genesene seit dem {startdatum}")
covid_fall[covid_fall.index > startdatum] ["Deutschland"].plot(label="Fallzahl")
covid_genesen[covid_genesen.index > startdatum] ["Deutschland"].
    plot(label="Genesen")
# plt.yscale("log")
plt.legend()
plt.show()

```



```
[15]: covid_genesen[("2022-03-01" > covid_genesen.index) & (covid_genesen.index >
    "2022-02-18")]["Deutschland"].plot(label="Genesen")
```

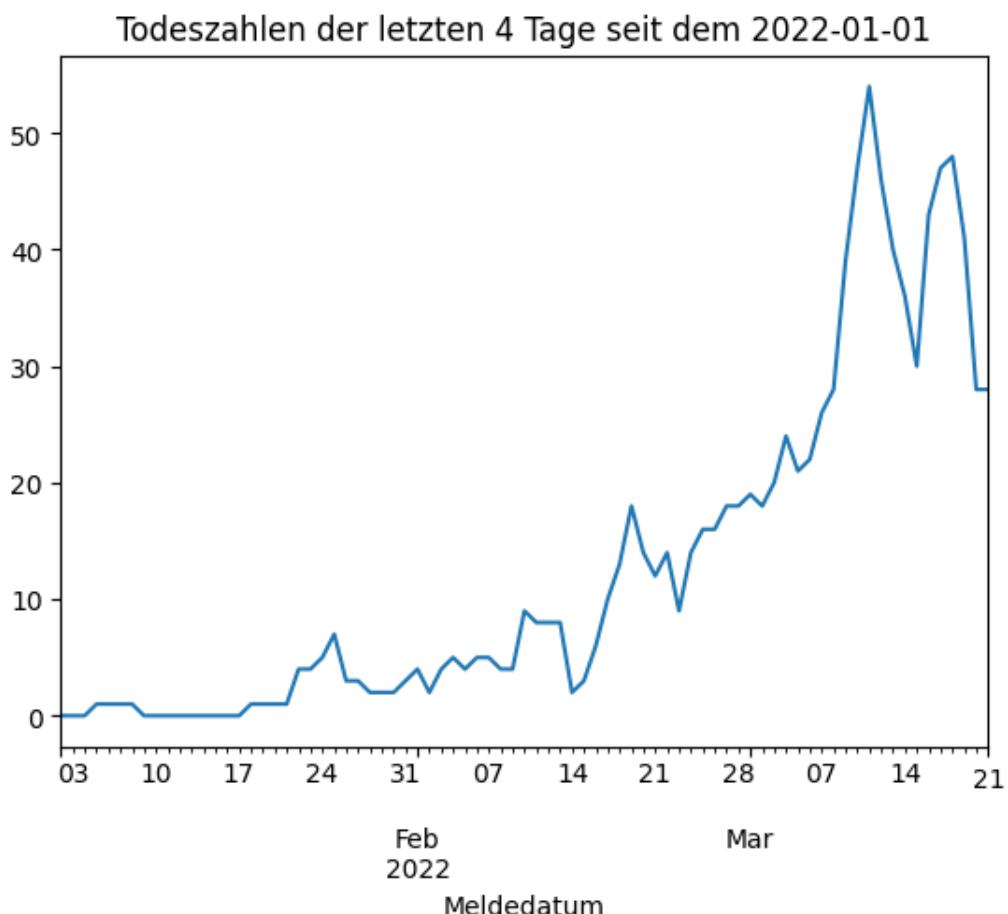
```
[15]: <Axes: xlabel='Meldedatum'>
```



Was passiert da?

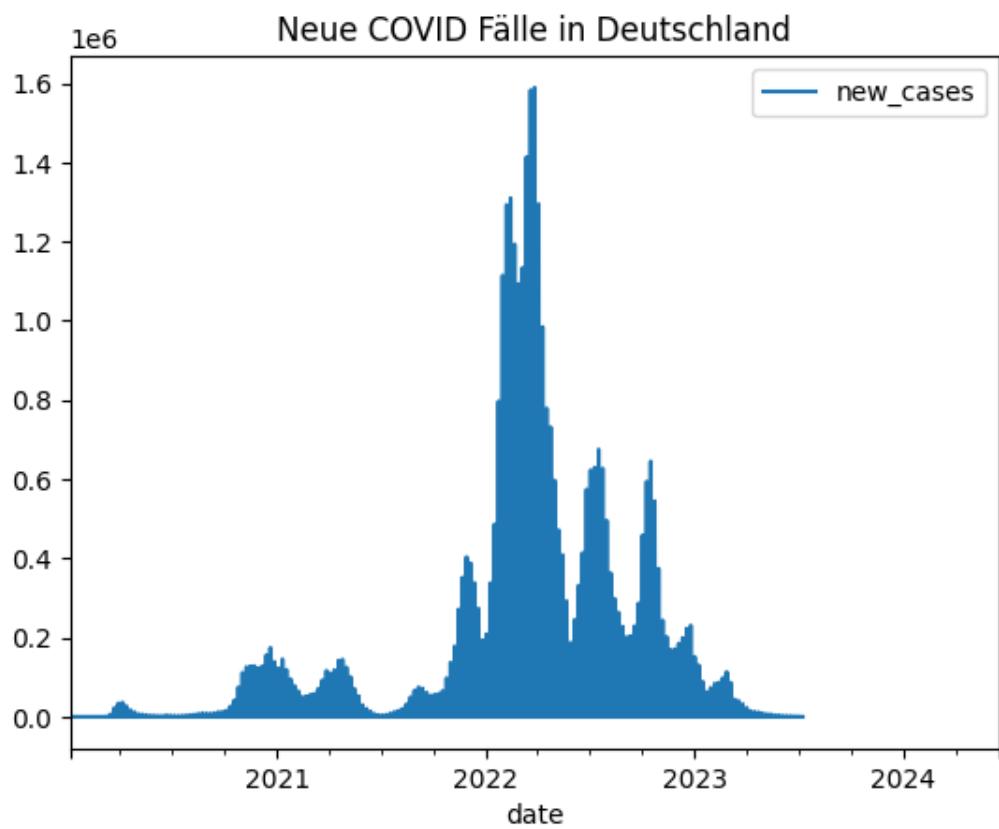
Eigentlich ist zu erwarten, dass die Anzahl der Genesenen einen sehr ähnlichen Verlauf hat, wie der gemeldeten Erkrankungen, jedoch mit einer Phasenverschiebung um 1-2 Wochen. Der rasante Anstieg der Genesenen, den wir im Zeitraum ab dem 22.02.2022 feststellen, muss demnach eine andere Ursache haben.

```
[16]: #covid_fall[covid_fall.index > startdatum]["Deutschland"].plot()
#covid_genesen[covid_genesen.index > startdatum]["Deutschland"].plot()
plt.title(f"Todeszahlen der letzten 4 Tage seit dem {startdatum}")
covid_tod[covid_tod.index > startdatum].rolling('4D')["Deutschland"].sum().
    plot()
plt.show()
```



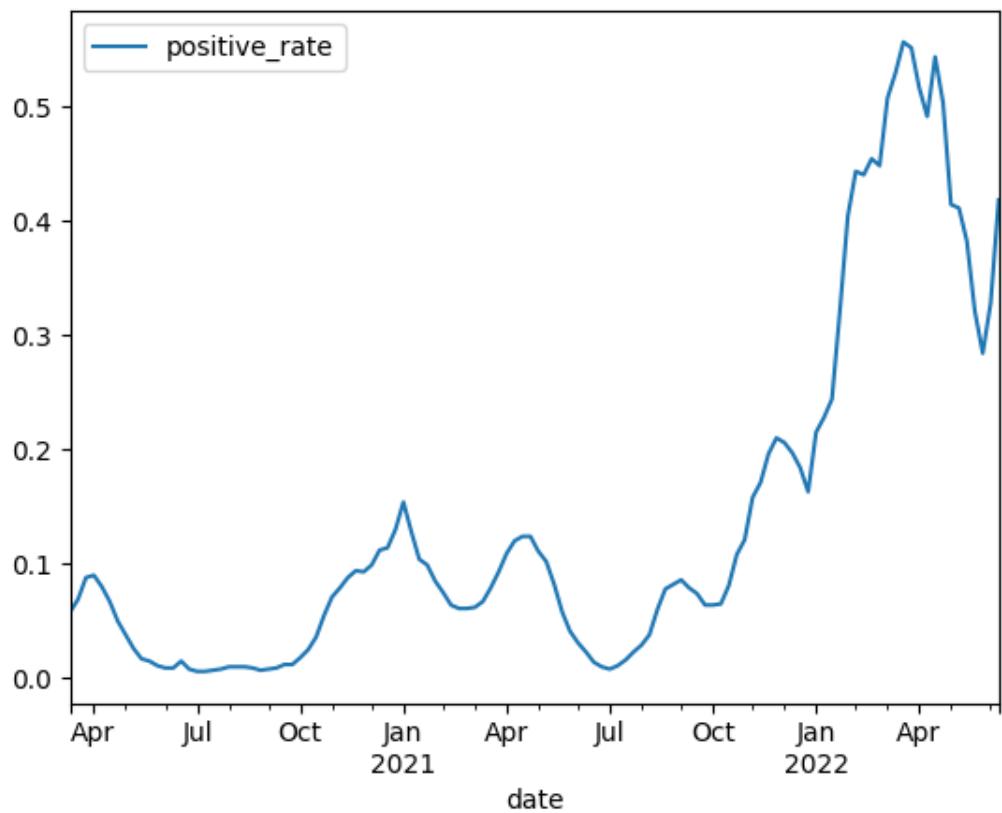
Wir betrachten nun die Daten vom Our World in Data.

```
[17]: df_owid[df_owid["location"] == "Germany"].plot(x="date", y="new_cases")
plt.title("Neue COVID Fälle in Deutschland")
plt.show()
```

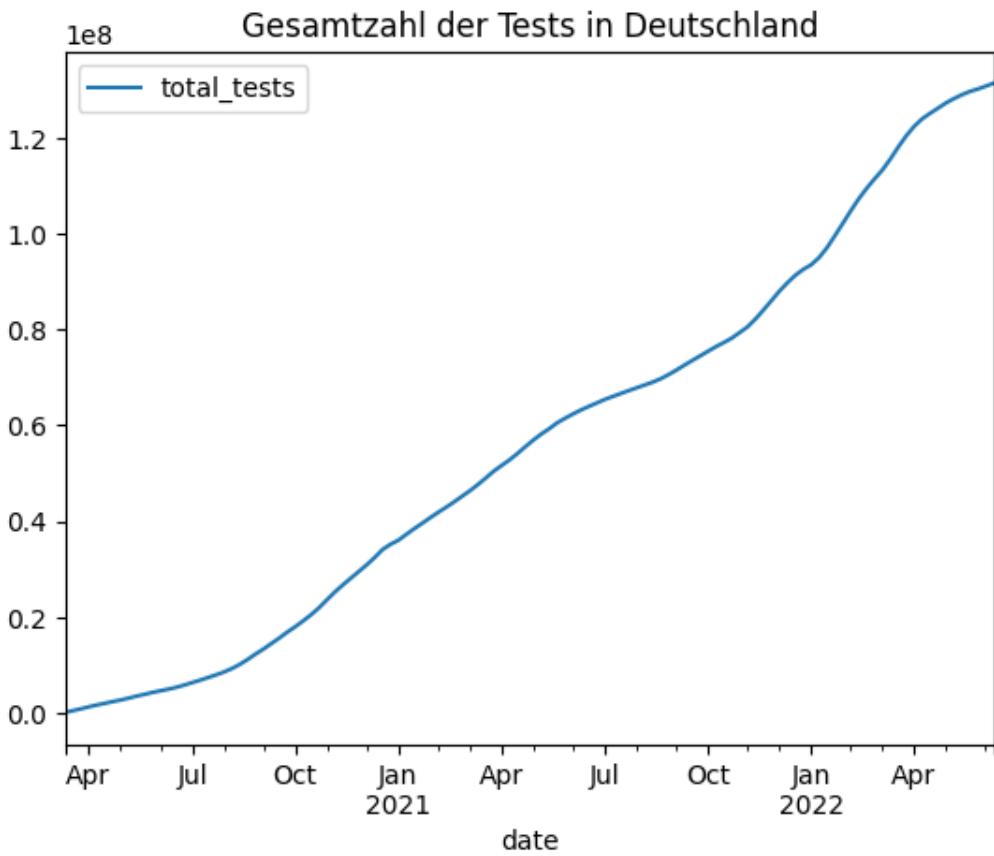


```
[18]: df_owid[df_owid["location"] == "Germany"][['date', 'positive_rate']].dropna().  
      plot(x='date', y='positive_rate')
```

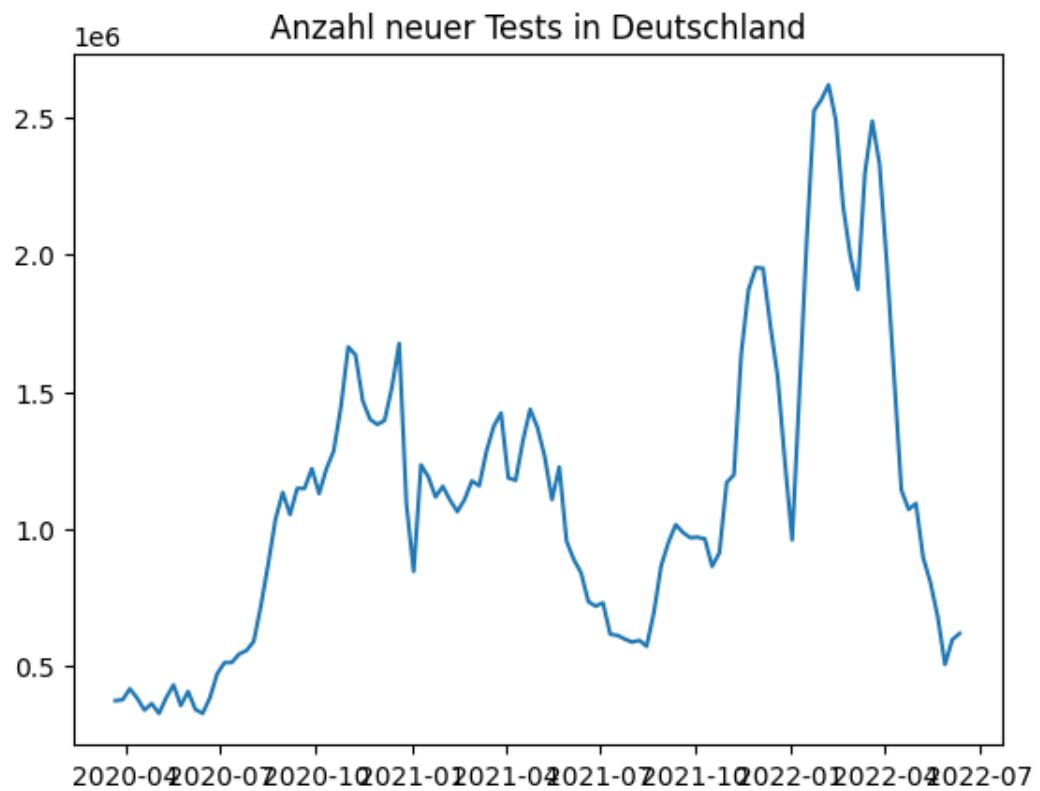
```
[18]: <Axes: xlabel='date'>
```



```
[19]: df_owid[df_owid["location"] == "Germany"][['date', 'total_tests']].dropna().  
    plot(x='date', y='total_tests')  
plt.title("Gesamtzahl der Tests in Deutschland")  
plt.show()
```



```
[20]: tests_in_de = df_owid[df_owid["location"] == "Germany"][['date',  
    ↴'total_tests']].dropna()  
tests_in_de.set_index("date", inplace=True)  
  
plt.title("Anzahl neuer Tests in Deutschland")  
plt.plot(tests_in_de - tests_in_de.shift(1, axis=0))  
plt.show()
```



Die Anzahl neuer Tests war im Jan. 2022 etwas eingebrochen und ist im Feb. 2020 wieder hochgeschnellt, aber ob das ausreichend ist, um die große Anzahl „plötzlich“ neu Genesener zu erklären, ist nicht ganz klar.

TSRI-11

July 23, 2024

1 Übung 11

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden wir uns mit Merkmalen (Features) und einfachen Modellen Klassifikatoren erzeugen und die Genauigkeit dieser Modelle mit verschiedenen Metriken bewerten. In Übung 11.1 und 11.2 werden wir uns mit Schwellwert-basierten Modellen beschäftigen. Übung 11.3 ist optional und behandelt die Klassifikation über eine einfache Form des Nächste Nachbarn Modells. Ziel aller Übungen ist es, Sie mit daten-getriebener Modellierung und der Evaluation von Klassifikatoren vertraut zu machen.

1.0.2 11.1 Frau und Mann (EDA, ROC, AUC)

In dieser Übung werden wir uns mit den Daten des [National Health and Nutrition Examination Survey](#) aus den Jahren 2009-2010 beschäftigen, die verschiedene Gesundheitsdaten der amerikanischen Bevölkerung umfassen. Wir werden zunächst in einer explorativen Datenanalyse (EDA) den Datensatz untersuchen. In einem zweiten Schritt werden wir daran arbeiten, mithilfe von Merkmalen (Features) Frauen und Männer zu unterscheiden, ohne dabei die Angabe des Geschlechts (Merkmal “*Gender*”) in den Daten zu nutzen. Das Resultat ist ein ganz einfaches Klassifikationsmodell, dessen Genauigkeit wir mithilfe verschiedener Metriken untersuchen werden.

Ihre Daten

- Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#).

Ihre Aufgaben

- (1) Importieren Sie die Daten und untersuchen Sie: Welche Merkmale (Features) enthält der Datensatz? (Hinweis: Das *Gender*-Merkmals kodiert, ob es sich um einen Mann (1) oder um eine Frau (0) handelt.)

```
[1]: #!wget https://data.bialonski.de/ds/nhanes-data.csv
```

```
import pandas as pd
```

```

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df: pd.DataFrame = pd.read_csv('nhanes-data.csv', delimiter='\t')

```

[2]: df

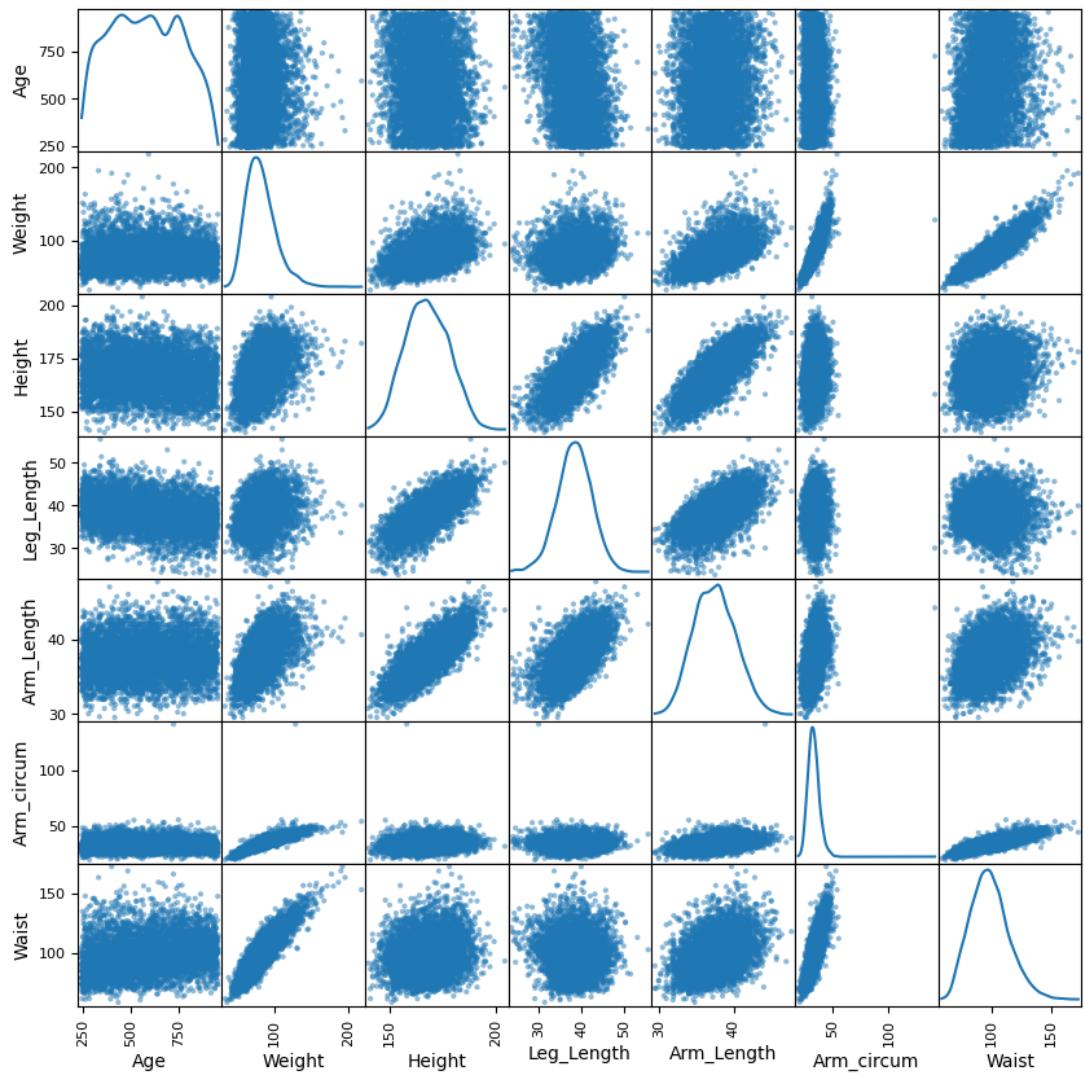
	Gender	Age	Weight	Height	Leg_Length	Arm_Length	Arm_circum	Waist
0	0	241	64.7	163	34.2	36.2	29.0	89.6
1	0	241	54.0	153	37.2	34.0	26.1	85.5
2	1	241	61.4	165	37.7	35.0	31.4	70.1
3	0	241	74.0	171	37.9	36.2	29.8	91.1
4	0	241	63.6	159	38.1	34.0	29.2	74.3
...
4974	1	958	89.5	184	43.0	42.8	32.2	112.8
4975	0	959	78.6	151	35.6	34.2	33.5	114.9
4976	1	959	86.5	175	38.6	41.5	32.8	100.2
4977	0	959	58.0	163	40.2	37.5	26.2	82.8
4978	1	959	76.2	168	40.8	39.5	29.8	103.5

[4979 rows x 8 columns]

Der Datensatz enthält die Features Gender, Age, Weight, Height, Leg_Length, Arm_Length, Arm_circum und Waist.

- (2) Führen Sie eine EDA auf den Daten (zunächst ohne das *Gender* Merkmal) durch. Erstellen Sie unter anderem eine **Scattermatrix** und erzeugen Sie auch paarweise **Korrelationskoeffizienten** (Pearson), um zu untersuchen, ob es Zusammenhänge zwischen den Merkmalen gibt. Notieren Sie hier Ihre Beobachtungen.

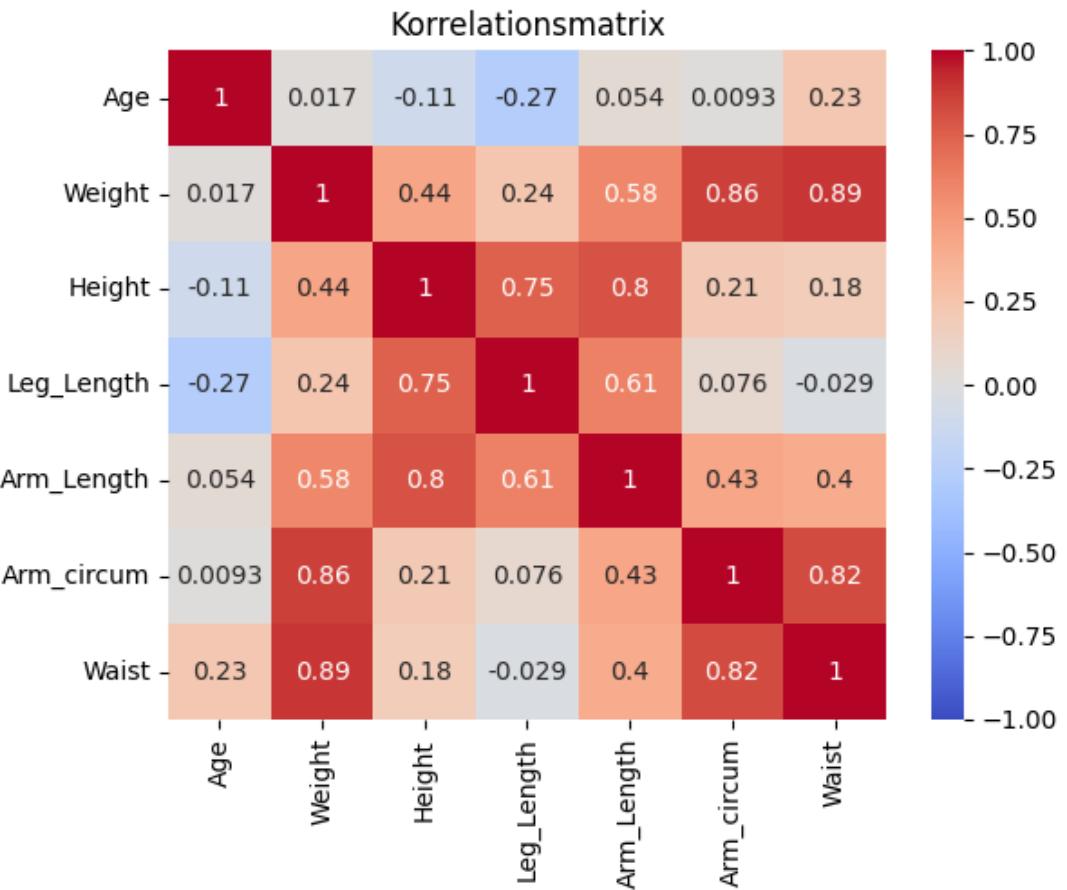
[3]: pd.plotting.scatter_matrix(df.loc[:, "Age":], figsize=(10, 10), diagonal="kde")
plt.show()



```
[4]: corr = df.loc[:, "Age":].corr(method='pearson', numeric_only=True)

sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

plt.title("Korrelationsmatrix")
plt.show()
```



Es bestehen stark positive Korrelation $r \geq 0.8$ zwischen den Merkmalen - `Arm_circum` und `Waist` - `Arm_circum` und `Weight` - `Weight` und `Waist` - `Arm_Length` und `Height`

und eine mittlere Korrelation $0.5 \leq |r| < 0.8$ zwischen den Merkmalen - `Weight` und `Arm_Length` - `Height` und `Leg_Length` - `Leg_Length` und `Arm_Length`

- (3) Als Fortsetzung zu Schritt (2): Welche Vermutungen haben Sie, die die beobachtete Korrelation zwischen Alter und Körpergröße sowie zwischen Gewicht und Taillenumfang erklären könnte?

Dass nur eine schwach negative Korrelation zwischen Alter und Körpergröße besteht, deutet darauf hin, dass Kinder, Jugendliche und Heranwachsende nicht Teil des Datensatzes sind.

Die starke Korrelation von Gewicht und Taillenumfang (sowie Armumfang) entspricht der Erwartung.

Sie haben bisher die Schritte einer Explorativen Datenanalyse (EDA) nachvollzogen, in der es darum geht, die Daten kennenzulernen (und ggf. interessante Hypothesen und Fragen zu generieren). Im nächsten Schritt wollen wir mit den Daten etwas erreichen: Wir möchten anhand der Merkmale entscheiden können, ob es sich um eine Frau oder einen Mann handelt (unser Ziel ist also eine binäre Klassifikation). Sobald wir mit einem konkreten Ziel an die Daten herantreten, ändert sich

unser Blick auf die Daten. Wir befinden uns in der *Feature Engineering* (Merkmalsgenerierung) Phase, in der wir nach Eigenschaften suchen, um unser Ziel zu erreichen.

- (4) Erzeugen Sie jeweils einen DataFrame für die Datensätze aller Frauen sowie für die Datensätze aller Männer. Untersuchen Sie, ob Sie Merkmale (außer *gender*) in den Daten finden, mit denen Sie zwischen den Geschlechtern unterscheiden können. Nutzen Sie dazu die Mittel der EDA, also Summary Statistics, Box-Plots, und so weiter. Notieren Sie sich vielversprechende Merkmale, die Ihnen die Unterscheidung zwischen den Geschlechtern erlauben könnten.

```
[5]: df_f: pd.DataFrame = df[df["Gender"] == 0].dropna()
df_m: pd.DataFrame = df[df["Gender"] == 1].dropna()
```

```
[6]: df_f.describe()
```

```
[6]:      Gender        Age       Weight      Height   Leg_Length \
count  2526.0  2526.000000  2526.000000  2526.000000  2526.000000
mean    0.0    583.213381   75.886382   160.982185   36.329612
std     0.0    196.841670   19.344734    7.164715   3.572567
min     0.0    241.000000   32.400000   140.000000   23.700000
25%    0.0    416.250000   62.100000   156.000000   34.100000
50%    0.0    576.000000   72.400000   161.000000   36.500000
75%    0.0    747.000000   86.375000   166.000000   38.700000
max    0.0    959.000000  190.200000   194.000000   47.700000
```

```
          Arm_Length   Arm_circum      Waist
count  2526.000000  2526.000000  2526.000000
mean   35.961006    32.305859   96.645606
std    2.182163    5.371440  15.887564
min   29.500000    19.500000  59.100000
25%  34.500000    28.500000  84.800000
50%  36.000000    31.600000  95.300000
75%  37.400000    35.400000 107.000000
max  44.000000    55.500000 168.400000
```

```
[7]: df_m.describe()
```

```
[7]:      Gender        Age       Weight      Height   Leg_Length \
count  2452.0  2452.000000  2452.000000  2452.000000  2452.000000
mean    1.0    585.944943   87.102896   174.413540   40.387235
std     0.0    197.574705   19.665853    7.744141   3.394613
min    1.0    241.000000   43.100000   142.000000   29.000000
25%    1.0    419.750000   73.475000   169.000000   38.200000
50%    1.0    588.000000   84.300000   175.000000   40.300000
75%    1.0    749.000000   97.300000   180.000000   42.600000
max    1.0    959.000000  218.200000   204.000000   55.500000
```

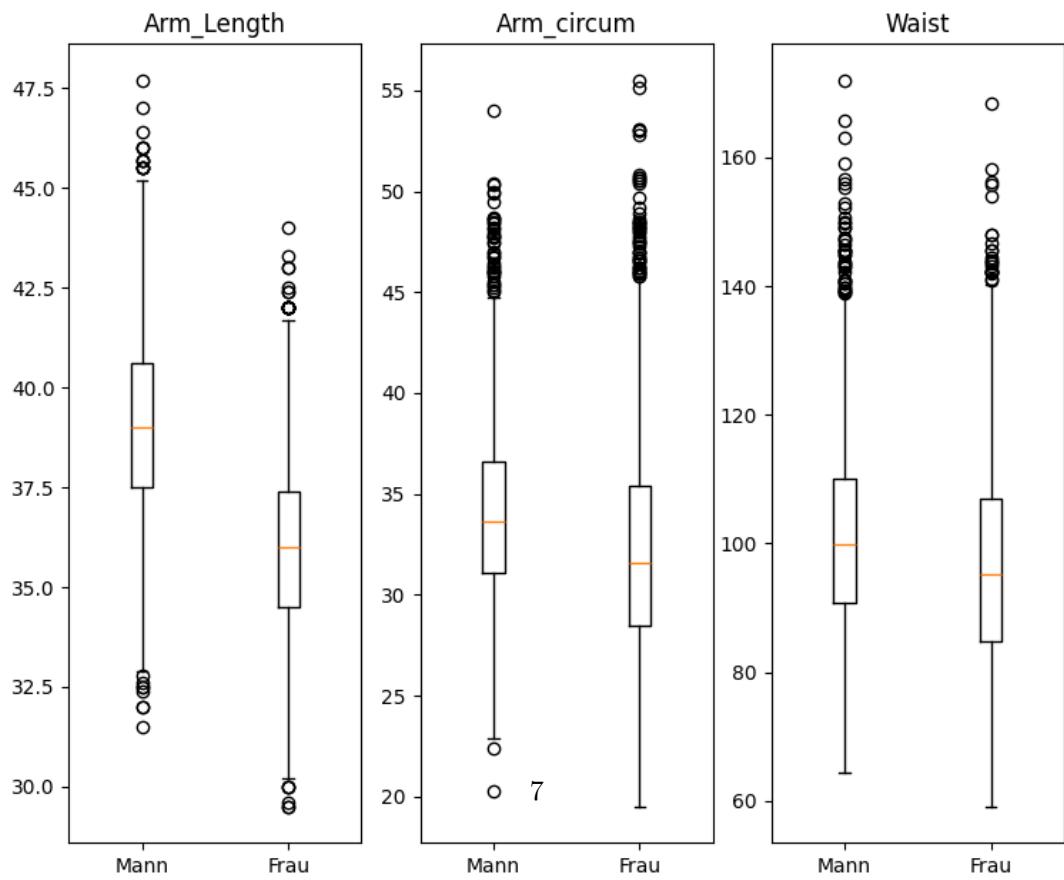
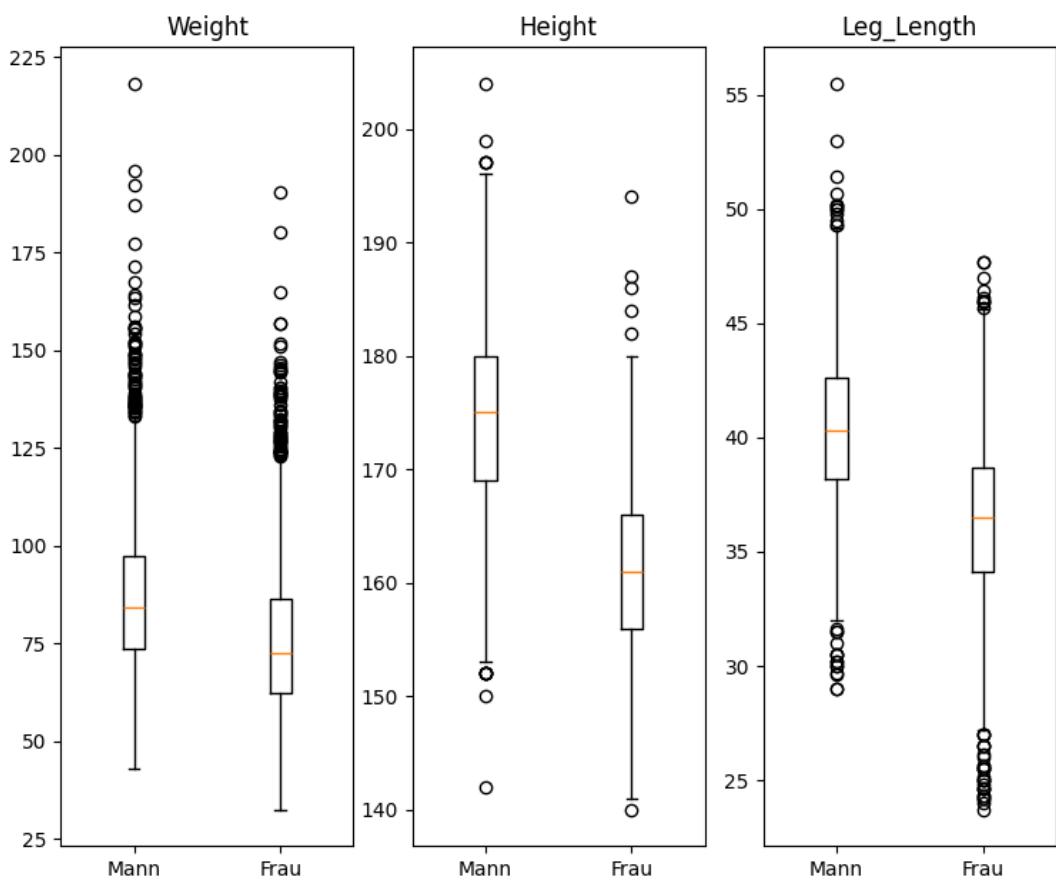
```
          Arm_Length   Arm_circum      Waist
count  2452.000000  2452.000000  2452.000000
```

```
mean      39.043515    34.013825    101.076794
std       2.380436     4.374639     15.460271
min      31.500000    20.300000    64.400000
25%      37.500000    31.100000    90.800000
50%      39.000000    33.600000    99.800000
75%      40.600000    36.600000   110.000000
max      47.700000    54.000000   172.000000
```

```
[8]: fig, axs = plt.subplots(2, 3, figsize=(9, 16))

for i, c in enumerate(df.loc[:, "Weight":].columns):
    axs.flat[i].set_title(c)
    axs.flat[i].boxplot([df_m[c], df_f[c]], tick_labels=["Mann", "Frau"])

plt.show()
```



Gut geeignet scheinen Height, Leg_Length und Arm_Length zu sein.

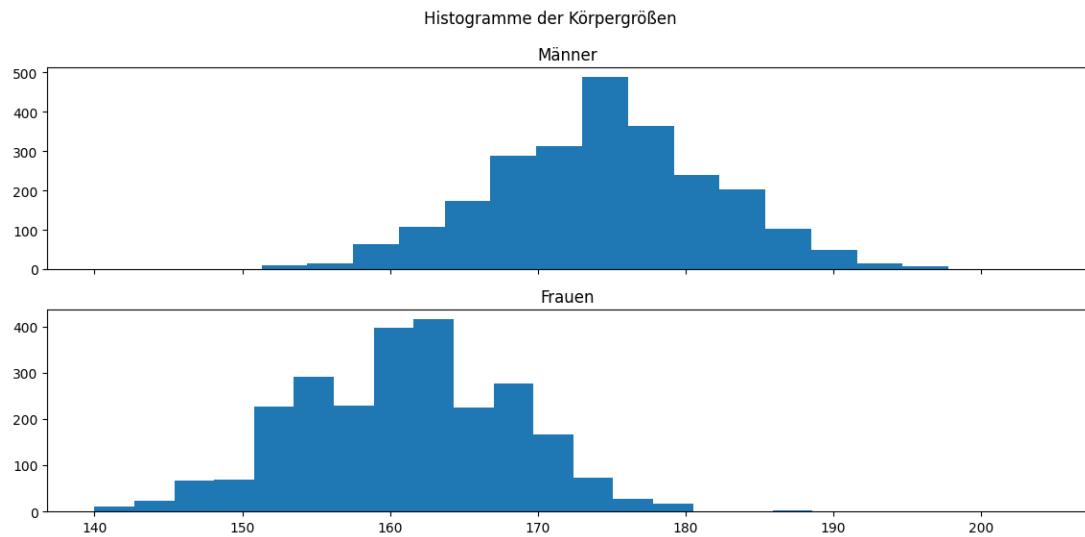
Im nachfolgenden Schritt gebe ich Ihnen ein Merkmal vor, welches Sie sich im Folgenden anschauen werden. Ihre Arbeit in Schritt (4) war aber nicht vergebens. Wir kommen darauf später noch zurück.

- (5) Erzeugen Sie ein [Histogramm](#), welches die Verteilung der Körpergrößen von Männern und von Frauen darstellt.

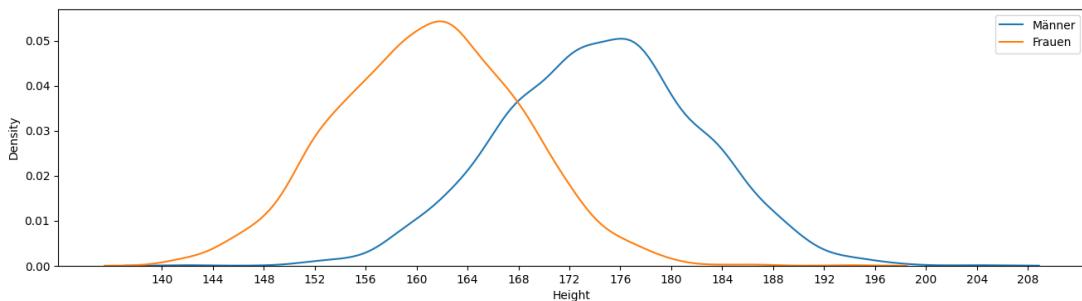
```
[9]: fig, axs = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

fig.suptitle("Histogramme der Körpergrößen")
axs[0].hist(df_m["Height"], bins=20)
axs[0].set_title("Männer")

axs[1].hist(df_f["Height"], bins=20)
axs[1].set_title("Frauen")
plt.show()
```



```
[10]: plt.figure(1, (16, 4))
sns.kdeplot(df_m["Height"], label="Männer")
sns.kdeplot(df_f["Height"], label="Frauen")
plt.xticks(np.arange(140, 210, 4))
plt.legend()
plt.show()
```



- (6) Wir werden ein einfaches Schwellwert-Modell konstruieren, um zwischen Männern und Frauen zu unterscheiden. Dieser sogenannte Klassifikator wird mithilfe eines Schwellwerts und eines Merkmals entscheiden, ob der Datensatz von einer Frau oder einem Mann stammt. Betrachten Sie die Abbildung aus Schritt (5). Notieren Sie sich hier den Schwellwert, den Sie nutzen würden, um alle Datensätze mit Körpergrößen größer als Ihr Schwellwert als Männer zu klassifizieren.

168cm

Der Schwellwert entscheidet darüber, wie gut oder wie schlecht ihr Klassifikator zwischen Frauen und Männern unterscheiden kann. Wir werden nun untersuchen, welche Genauigkeit unser Klassifikator hat und dabei auch einen Schwellwert finden, der die Genauigkeit des Klassifikators maximiert.

- (7) Sie haben in Schritt (5) beobachtet, dass Männer (zumindest in dem von Ihnen untersuchten US-amerikanischen Datensatz) tendenziell etwas größer sind als Frauen. Sei P die Anzahl der Männer und N die Anzahl der Frauen im Datensatz. Bestimmen Sie die Wahr-Positiv-Rate (*True Positive Rate*, TPR) sowie die Falsch-Positiv-Rate (*False Positive Rate*, FPR) für den Schwellwert, für den Sie sich in Schritt (6) entschieden haben. Für die Definition von TPR und FPR schauen Sie bitte in die Vorlesungsfolien. Interpretieren Sie kurz Ihre erhaltenen Werte. (1-2 Sätze).

Das **Precision** (oder auch **Positive Predictive Value**) betrachtet daher nur die Positiven Punkte (und „bestraft“ damit Falsch-Positive).

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Der **Recall** (oder auch die **True Positive Rate** oder **Sensitivity**) betrachtet das Verhältnis der Richtig-Positiven zu allen Positiven (und „bestraft“ damit Falsch-Negative).

$$\text{TPR} = \frac{\text{TP}}{P} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

```
[11]: def calc_tf_pn(df_p: np.array, df_n: np.array, cutoff: float) -> tuple[int, int, int]:
    P: int = df_p.size # Anzahl der Datenpunkte in der vorherzusagenden Klasse (Positiv)
```

```

N: int = df_n.size # Anzahl der Datenpunkte, die nicht der
↪vorherzusagenden Klasse entsprechen (Negativ)
TP: int = df_p[df_p > cutoff].size # Anzahl der korrekt vorhergesagten
↪Positiven
TN: int = df_n[df_n <= cutoff].size # Anzahl der korrekt vorhergesagten
↪Negativen
FN: int = df_p[df_p <= cutoff].size # Anzahl der falsch Negativen
FP: int = df_n[df_n > cutoff].size # Anzahl der falsch Positiven
return TP, TN, FN, FP

"""
Errechnet die True Positive Rate (TPR, Recall/Sensitivity) sowie den Positive
↪Predictive Value (PPV, Precision)
unter der Annahme, dass die positiven Daten P rechts vom Cutoff liegen (also
↪größer als der Cutoff sind).
"""

def calc_classifier_metrics(df_p: np.array, df_n: np.array, cutoff: float,
↪output=True) -> tuple[float, float, float, float, float]:
    TP, TN, FN, FP = calc_tf_pn(df_p, df_n, cutoff)
    PPV = TP/(TP + FP) # Positive Predictive Value (Precision)
    TPR = TP/(TP + FN) # True Positive Rate (Recall/Sensitivity)
    FPR = FP/(TN + FP) # False Positive Rate

    ACC = (TP + TN)/(TP + TN + FN + FP) # Accuracy
    if PPV+TPR == 0:
        F1 = 0
    else:
        F1 = 2*(PPV*TPR)/(PPV+TPR) # F1-Score
    J = TPR - FPR # Youden Index

    if output:
        print(f"True PR TPR = {TPR}")
        print(f"False PR FPR = {FPR}")
        print(f"Accuracy ACC = {ACC}")
        print(f"Precision PPV = {PPV}")
        print(f"F1-Score F_1 = {F1}")
        print(f"Youden I. J = {J}")

    return TPR, FPR, ACC, PPV, F1, J

```

[12]: calc_classifier_metrics(df_m["Height"].to_numpy(), df_f["Height"].to_numpy(),
↪168)

```

True PR TPR = 0.7712071778140294
False PR FPR = 0.1496437054631829
Accuracy ACC = 0.8113700281237445
Precision PPV = 0.8334067871308947

```

```
F1-Score  F_1 = 0.8011014615547554
Youden I.  J = 0.6215634723508465
```

```
[12]: (0.7712071778140294,
 0.1496437054631829,
 0.8113700281237445,
 0.8334067871308947,
 0.8011014615547554,
 0.6215634723508465)
```

Eine TPR = 77.12% bedeutet, dass 77.12% aller Männer anhand des Kriteriums der Körpergröße von 168cm richtig erkannt wurden.

Eine FPR = 14.96% bedeutet, dass mit diesem Kriterium fälschlicherweise auch 14.96% aller Frauen als Männer klassifiziert wurden.

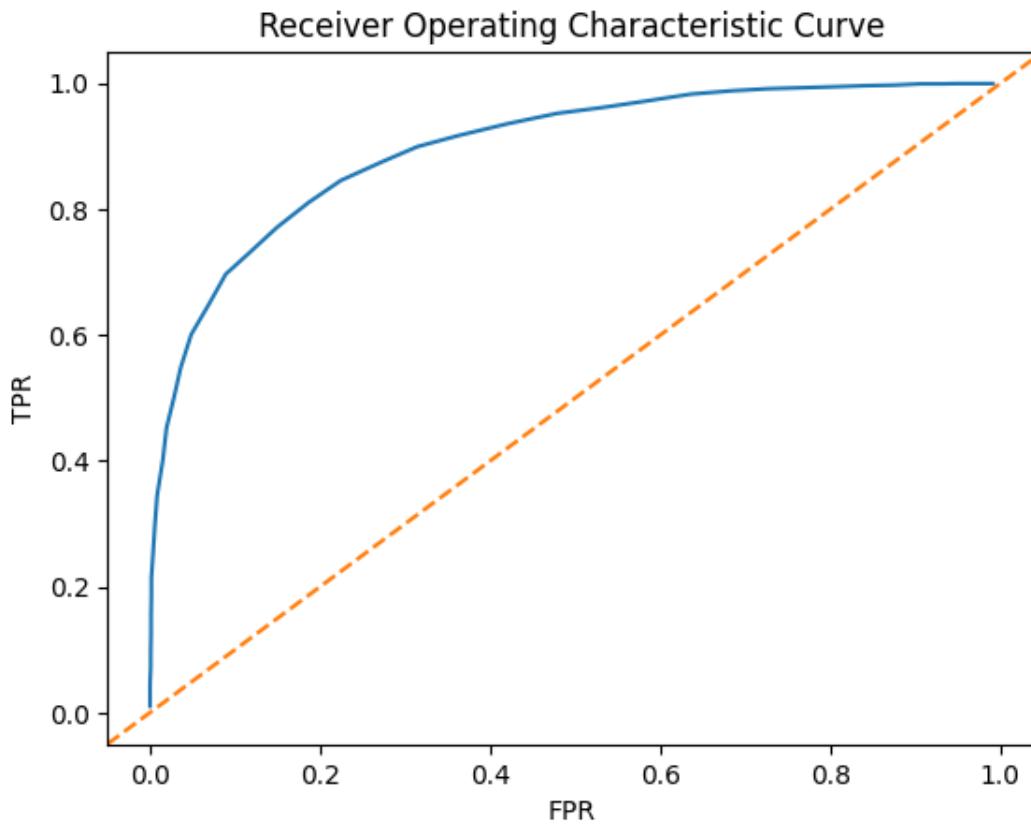
- (8) Wir wollen nun untersuchen, wie gut sich die Körpergröße für die Unterscheidung zwischen Männern und Frauen eignet, und welcher Schwellwert hier optimal ist. Dazu werden Sie für eine Menge von Schwellwerten die zugehörigen TPR und FPR-Werte bestimmen. Erstellen Sie zunächst eine Menge von Schwellwerten. Hinweis: Die Daten können Ihnen dabei helfen, Schwellwerte zu definieren.

```
[13]: schwellwerte: np.ndarray = np.arange(144, 192, 1)
```

- (9) Bestimmen Sie TPR und FPR für Ihre Schwellwerte aus Schritt (8). Tragen Sie in einem Plot die TPR-Werte (y-Achse) gegen die FPR-Werte (x-Achse) auf und zeichnen Sie zusätzlich die Raumdiagonale ein. Vergessen Sie nicht, Ihren Plot zu beschriften. Damit haben Sie Ihre *Receiver Operating Characteristic Curve* (ROC) erhalten. Ihre ROC-Kurve wird sich von der Raumdiagonalen unterscheiden. Was bedeutet dies? (1-2 Sätze).

```
[14]: plt.title("Receiver Operating Characteristic Curve")
roc = []
youdens = []
for s in schwellwerte:
    TPR, FPR, _, _, _, J = calc_classifier_metrics(df_m["Height"].to_numpy(), □
    ↪df_f["Height"].to_numpy(), s, output=False)
    roc.append([FPR, TPR])
    youdens.append(J)

roc = np.array(roc)
plt.plot(roc[:, 0], roc[:, 1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.axline((0, 0), slope=1, c="C1", linestyle="--")
plt.show()
```



Die Raumdiagonale entspricht einem Zufallsprädiktor und ein solcher Verlauf der ROC Kurve würde anzeigen, dass wir eine nicht-trennbare Verteilung vorliegen hätten.

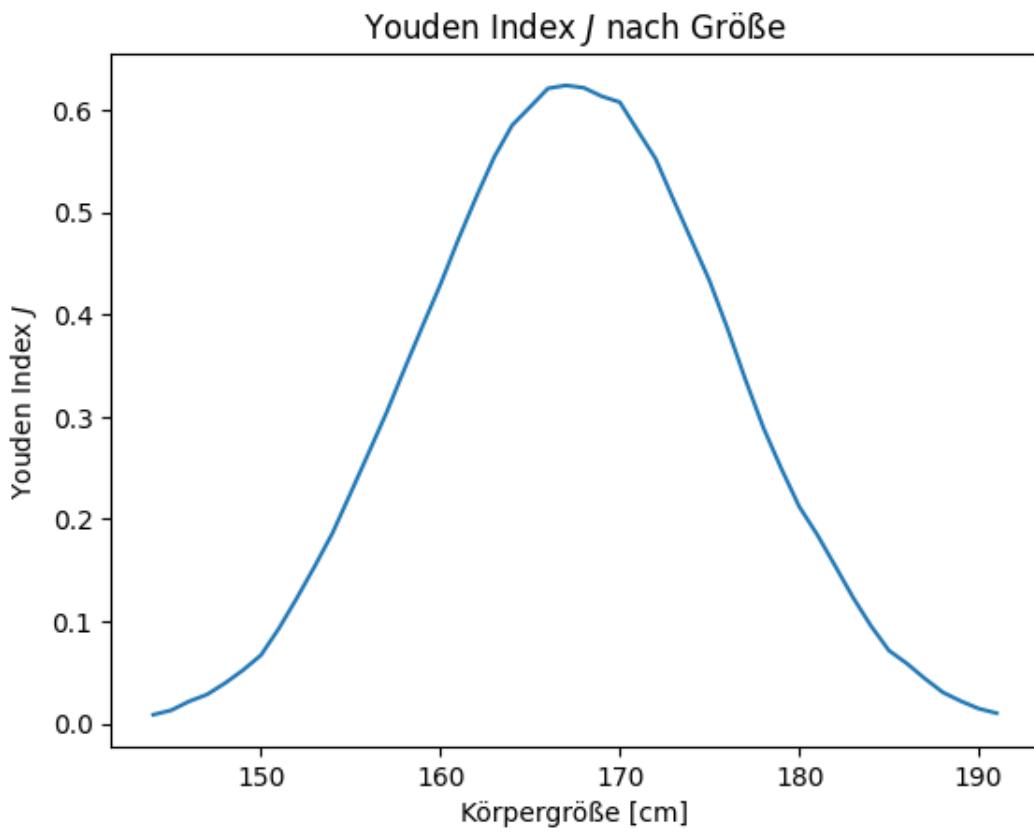
- (10) Schlagen Sie den Youden-Index in der Vorlesung nach und bestimmen Sie damit den optimalen Schwellwert zur Unterscheidung zwischen Frauen und Männern mithilfe der Körpergröße. Vergleichen Sie den erhaltenen Wert mit dem Wert, den Sie in Schritt (6) erhalten haben. (1-2 Sätze)

Der Youden Index J mit

$$J = \text{TPR} - \text{FPR}$$

wird an dem Punkt, welcher die größte Distanz zur Raumdiagonalen hat, maximal.

```
[15]: plt.title("Youden Index $J$ nach Größe")
plt.plot(schwellwerte, youdens)
plt.ylabel("Youden Index $J$")
plt.xlabel("Körpergröße [cm]")
plt.show()
```



```
[16]: schwellwerte[np.argmax(youdens)]
```

```
[16]: np.int64(167)
```

Der Optimale Wert liegt bei $\theta = 167\text{cm}$, was sehr nach an meiner Schätzung von 168cm liegt.

(11) Wir werden nun die ROC-Kurve durch eine skalare Größe charakterisieren, der *Area Under the Curve* (AUC). Nutzen Sie die [Sehnentrapezregel](#), um die Fläche unter Ihrer ROC-Kurve zu bestimmen und geben Sie den Wert der AUC aus.

- Bevor Sie die AUC bestimmen, stellen Sie sicher, dass die TPR und FPR Werte in der richtigen Reihenfolge vorliegen ([Dies](#) kann Ihnen dabei helfen).
- Welchen AUC-Wert würden Sie für einen Zufallsklassifikator (Zufallsprädiktor) erhalten?

```
[17]: def calc_AUC(roc: np.ndarray) -> float:
    roc = np.flipud(roc)
    auc: float = 0.0
    for i in range(len(roc)-1):
        auc += (roc[i+1, 0] - roc[i, 0]) * (roc[i, 1] + roc[i+1, 1]) / 2
    return auc
```

```
# oder mit np.trapezoid:
def calc_AUC(roc: np.ndarray) -> float:
    roc = np.flipud(roc)
    return np.trapezoid(y=roc[:, 1], x=roc[:, 0], dx=1/roc[:, 1].shape[0])

calc_AUC(roc)
```

[17]: `np.float64(0.8887302074735961)`

Ein Zufallsklassifikator würde eine AUC von 0.5 liefern, während eine perfekter Prädikator/Klassifikator eine AUC von 1.0 liefern würde.

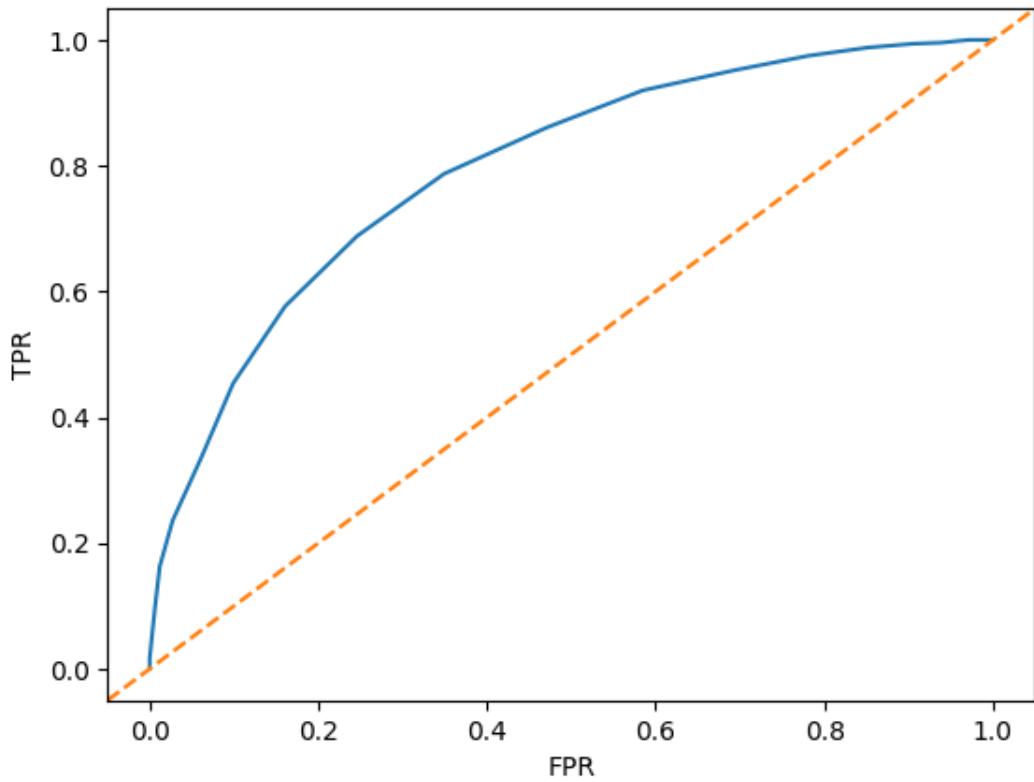
- (12) [Optional] Sie haben in Schritt (4) verschiedene Merkmale notiert, mithilfe derer Sie vermuteten, dass Sie Frauen und Männer voneinander unterscheiden könnten. Wählen Sie aus Schritt (4) ein oder zwei Merkmale heraus, die *nicht* der Körpergröße entsprechen. Bestimmen Sie für diese Merkmale die AUC. Sind Ihre Merkmale besser oder schlechter geeignet, um zwischen Männern und Frauen zu unterscheiden?

```
[18]: for merkmal in ["Leg_Length", "Arm_Length"]:
    schwellwerte: np.ndarray = np.arange(df[merkmal].min(), df[merkmal].max(), ↵
                                         1)

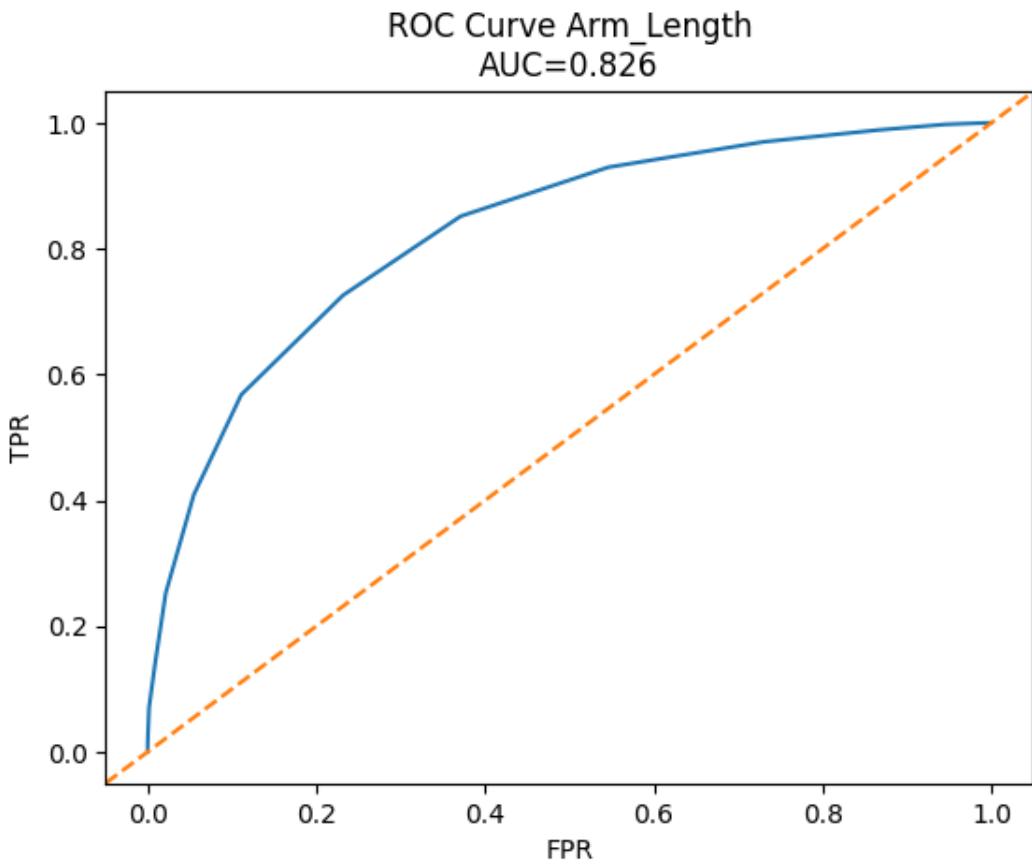
    roc = []
    for s in schwellwerte:
        TPR, FPR, _, _, _ = calc_classifier_metrics(df_m[merkmal].to_numpy(), df_f[merkmal].to_numpy(), s, output=False)
        roc.append([FPR, TPR])

    roc = np.array(roc)
    plt.plot(roc[:, 0], roc[:, 1])
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.axline((0, 0), slope=1, c="C1", linestyle="--")
    plt.title(f"ROC Curve {merkmal}\nAUC={calc_AUC(roc):.3f}")
    plt.show()
    print(f"Ein Klassifikator mit dem Merkmal {merkmal} liefert ein AUC ="
          f"\n{calc_AUC(roc)}")
```

ROC Curve Leg_Length
AUC=0.795



Ein Klassifikator mit dem Merkmal Leg_Length liefert ein AUC =
0.7948405102432259



Ein Klassifikator mit dem Merkmal Arm_Length liefert ein AUC =
0.8261527100213247

```
[19]: print("Ein Klassifikator mit dem Merkmal")
for merkmal in df.loc[:, "Weight":].columns:
    schwellwerte: np.ndarray = np.arange(df[merkmal].min() + 2, df[merkmal].
    ↪max() -2, 1)
    roc = []
    for s in schwellwerte:
        try:
            TPR, FPR, _, _, _, _ = calc_classifier_metrics(df_m[merkmal].
            ↪to_numpy(), df_f[merkmal].to_numpy(), s, output=False)
            roc.append([FPR, TPR])
        except ZeroDivisionError:
            continue

    roc = np.array(roc)
    print(f" - {merkmal:11s} liefert ein AUC von {calc_AUC(roc):.3f}")
```

Ein Klassifikator mit dem Merkmal

- Weight liefert ein AUC von 0.675
- Height liefert ein AUC von 0.893
- Leg_Length liefert ein AUC von 0.788
- Arm_Length liefert ein AUC von 0.811
- Arm_circum liefert ein AUC von 0.612
- Waist liefert ein AUC von 0.583

Somit scheint Height das beste Merkmal zur Unterscheidung von Mann und Frau zu sein, gefolgt von Arm_Length und Leg_Length.

1.0.3 11.2 Detektion epileptischer Anfälle (Feature Engineering, ROC, AUC)

1% der Weltbevölkerung leidet unter epileptischen Anfällen. Etwa 25% aller Patienten können mithilfe von [Antikonvulsiva](#) die Häufigkeit ihrer epileptischen Anfälle nicht zufriedenstellend senken. Für solche Patienten, wenn sie unter einem besonders hohen Leidensdruck stehen, werden epilepsiechirurgische Eingriffe angedacht, in Rahmen derer Teile des Gehirns entfernt werden, um Anfallsfreiheit zu erreichen. Nur wenige Kliniken führen solche Eingriffe durch. In diesem Zusammenhang zählt die [Klinik für Epileptologie Bonn](#) zu einem der wichtigsten europäischen Zentren. Diese Klinik hat auch im Jahr 2001 einen der ersten, öffentlich frei zugänglichen Datensätze geschaffen (den "Bonn Datensatz"), die wir im Rahmen dieser Übung untersuchen werden.

Bevor Patienten Teile des Gehirns operativ entfernt werden, wird untersucht - sehr vereinfacht gesprochen - wo im Gehirn welche Funktionen (beispielsweise Sprache, Muskelsteuerung und so weiter) implementiert sind und in welcher Region (je nach Epilepsietypr) epileptische Anfälle im Gehirn entstehen. Dazu werden den Patienten die Schädeldecke geöffnet, und es werden Elektroden implantiert, um ein sogenanntes intrakranielles EEG aufzuzeichnen. Nachfolgend finden Sie das Implantationsschema für die Daten, die Sie untersuchen werden:



Die linke Abbildung zeigt sogenannte Tiefenelektroden (Stabelektroden), die die Gehirndynamik des Hippocampus erfassen. Der Hippocampus ist Teil des limbischen Systems, das zum Beispiel Emotionen verarbeitet aber auch wichtig für die Gedächtnisbildung ist. Sogenannte Streifenelektroden erfassen die Hirndynamik an lateralen (mittlere Abbildung) und basalen (rechte Abbildung) Stellen des Neokortex. Das Paper von Andrzejak et al, das die Daten beschreibt, finden Sie [hier](#) - nur für den Fall, dass Sie neugierig sind (Andrzejak et al (2001). Indications of nonlinear deterministic and finite dimensional structures in time series of brain electrical activity: Dependence on

recording region and brain state, Phys. Rev. E, 64, 061907).

Ein Fernziel aktueller Forschung ist es, Methoden zu entwickeln, um epileptische Anfälle zuverlässig vorherzusagen, um damit die Patienten vor dem Auftreten ihrer Anfälle zu warnen. Die kleine Schwester dieser Forschungsrichtung ist die “Anfallsdetektion”, also die zuverlässige Detektion mithilfe von EEG-Aufzeichnungen. Eine zuverlässige Anfallsdetektion wird als wichtiger Zwischenschritt hin zur Anfallsvorhersage erachtet.

Ihre Daten * Sie finden die Daten, die Sie für diese Übung benötigen, [hier](#) (Datensatz 1) und [hier](#) (Datensatz 2).

Daten D und E stammen aus dem “Bonn Datensatz”. Datensatz D enthält 100 Zeitreihen von 5 Patienten, die während einer anfallsfreien Phase aufgezeichnet wurden. Datensatz E enthält 100 Zeitreihen von denselben Patienten, die während eines epileptischen Anfalls aufgezeichnet wurden. Alle Zeitreihen wurden mit einer Abtastrate von $f = 173.61$ Hz erfasst.

```
[20]: #!wget https://data.bialonski.de/ds/bonn_epi_dataset_D.bz2
#!wget https://data.bialonski.de/ds/bonn_epi_dataset_E.bz2
```

Ihre Aufgaben

Unser Ziel ist es, ein Merkmal (Feature) zu finden, mit dem wir die Zeitreihen der Anfälle (Daten E) von den “anfallsfreien” Zeitreihen (Daten D) unterscheiden können. Dies ist ein binäres Klassifikationsproblem. Wir gehen dabei ganz ähnlich wie in Aufgabe 11.1 vor.

(1) Importieren Sie die Daten und machen Sie sich mit Ihnen vertraut:

- Was sind die Spalten, was sind die Zeilen der DataFrames, die Sie importiert haben?
- Visualisieren Sie beispielhafte Zeitreihen aus den beiden Datensätzen.

```
[21]: df_D: pd.DataFrame = pd.read_csv("bonn_epi_dataset_D.bz2", index_col=0)
df_E: pd.DataFrame = pd.read_csv("bonn_epi_dataset_E.bz2", index_col=0)
df_D
```

```
[21]:      1     2     3     4     5     6     7     8     9    10 ...    91    92    93    94    95 \
0     34    60    26   -41    13   -15   -24    23  -263    59 ...    99   -131     1   -41   -39
1     33    47    16   -42     6    -2   -27    17  -263    52 ...   114   -153    -4   -41   -27
2     28    38    13   -48    -1     0   -23    10  -261    51 ...   122   -177    -6   -48   -16
3     22    29    12   -48   -13     2   -28    10  -258    46 ...   132   -194   -15   -48    -3
4     21    28    17   -48   -29    -2   -34     7  -258    43 ...   151   -204    -8   -44    17
...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
4092   40   209   113   -23   167   -52   -70    34  -314   109 ...    55   -38   -25     16   -30
4093   45   177   119   -28   175   -53   -67    42  -319   110 ...    75   -37   -13     19   -22
4094   39   149   114   -30   161   -44   -57    41  -316   109 ...    84   -41   -12     16   -34
4095   41   126    99   -23   129   -42   -33    39  -316   104 ...    83   -46   -10     12   -39
4096    7    42  -130   -13     1   -25   -52   -55  -254   -57 ...    54   -15    22    -8   -56

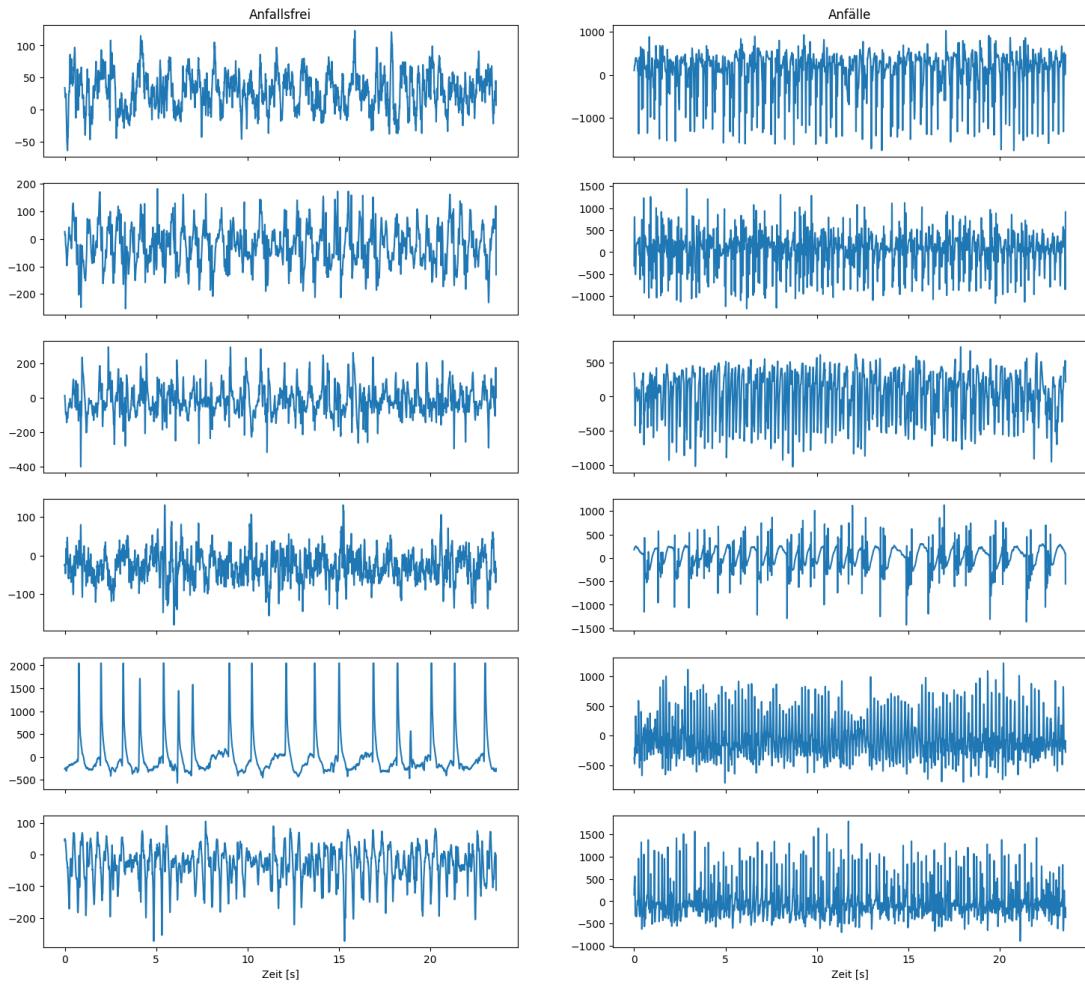
          96     97     98     99    100
0      45     75     67      5   -45
1      52     72     86      2   -53
2      79     72     99     -6   -51
```

```
3      117    80   109   -4   -52
4     146    81   115   -9   -54
...
4092    -9   -46    18   31   -24
4093     0   -56    16   39   -22
4094    18   -40    17   36   -26
4095    28   -43    10   36   -14
4096    35   106    26   -5   -28
```

[4097 rows x 100 columns]

```
[22]: fig, axs = plt.subplots(6, 2, figsize=(18, 16), sharex=True)
for y in range(6):
    axs[y, 0].plot(df_D.index / 173.61, df_D.iloc[:, y*2])
    axs[y, 1].plot(df_E.index / 173.61, df_E.iloc[:, y*2])

axs[0, 0].set_title("Anfallsfrei")
axs[0, 1].set_title("Anfälle")
axs[5, 0].set_xlabel("Zeit [s]")
axs[5, 1].set_xlabel("Zeit [s]")
plt.show()
```



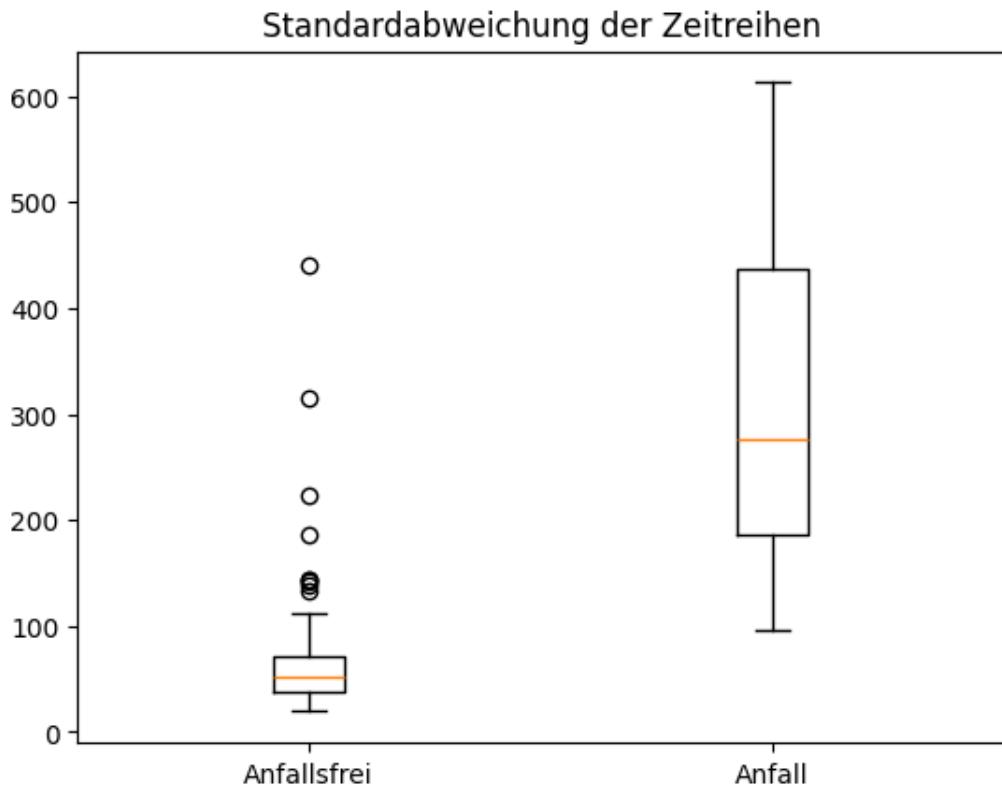
- (2) Normieren Sie alle Zeitreihen, sodass jede Zeitreihe den Mittelwert 0 aufweist. Auf diesen Daten werden Sie von nun an weiterarbeiten.

```
[23]: df_D = df_D - df_D.mean()
df_E = df_E - df_E.mean()
```

- (3) Willkommen im Feature Engineering: Untersuchen Sie mit Mitteln der Explorativen Analyse (EDA), mit welchen Merkmalen sich die beiden Datensätze voneinander gut unterscheiden lassen. Entscheiden Sie sich am Ende für ein Merkmal, mit dem Sie fortfahren wollen.

- Nutzen Sie Boxplots für eine schnelle Einschätzung, ob sich die Verteilung eines Merkmals für Datensatz D von der Verteilung desselben Merkmals für Datensatz E unterscheidet.
- Tipp: Probieren Sie mehrere Ideen aus aber halten Sie es einfach.

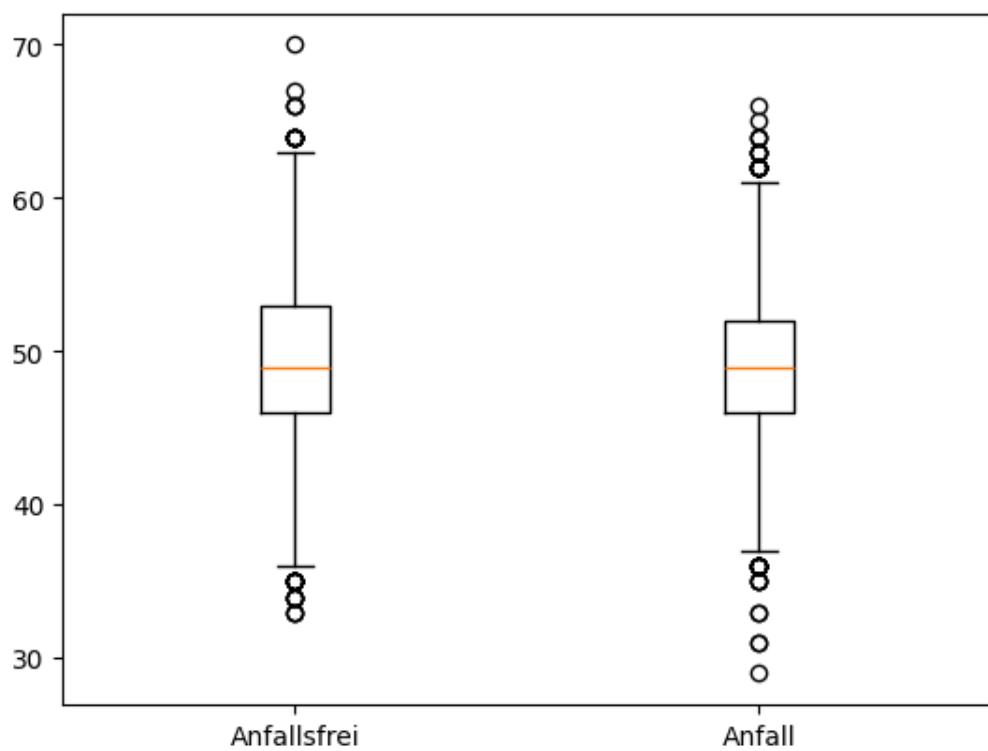
```
[24]: plt.title("Standardabweichung der Zeitreihen")
plt.boxplot([df_D.std(), df_E.std()], tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```



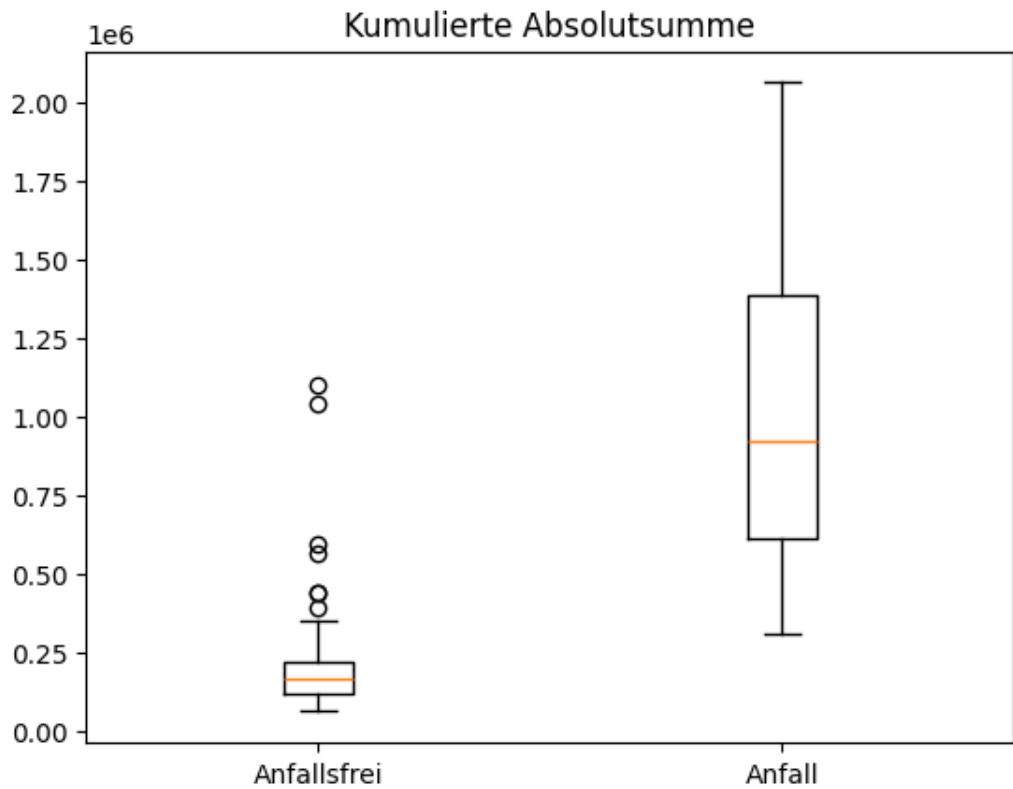
```
[25]: plt.title(f"Anzahl der Vorzeichenwechsel in {4096/173.61:.2f} Sek.")
def vzw(x: pd.Series) -> float:
    count = 0
    for i in range(len(x) - 1):
        if (x.iloc[i] < 0 < x.iloc[i + 1]) or (x.iloc[i] > 0 > x.iloc[i + 1]):
            count += 1
    return count

plt.boxplot([df_D.agg(vzw, axis="columns"), df_E.agg(vzw, axis="columns")], □
           □ tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```

Anzahl der Vorzeichenwechsel in 23.59 Sek.

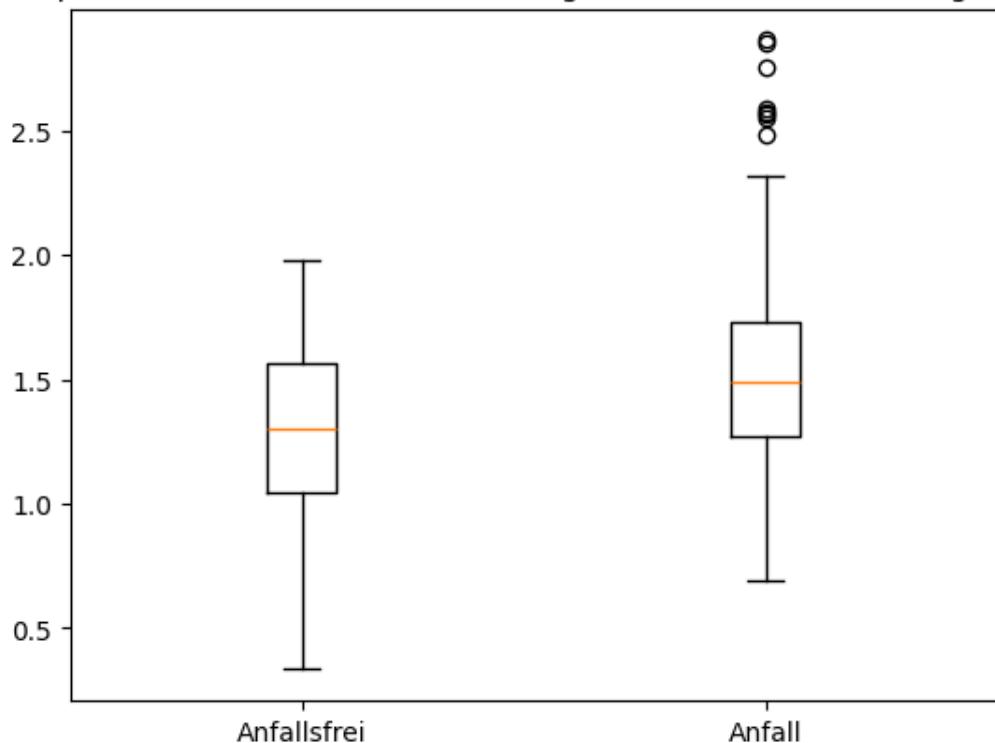


```
[26]: plt.title(f"Kumulierte Absolutsumme")
plt.boxplot([df_D.abs().sum(), df_E.abs().sum()], tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```



```
[27]: plt.title(f"$p=0.9$-Quantil der Absolutbeträge der relativen Änderungen")
plt.boxplot([df_D.pct_change().abs().quantile(0.9), df_E.pct_change().abs().
    quantile(0.9)], tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```

$p = 0.9$ -Quantil der Absolutbeträge der relativen Änderungen



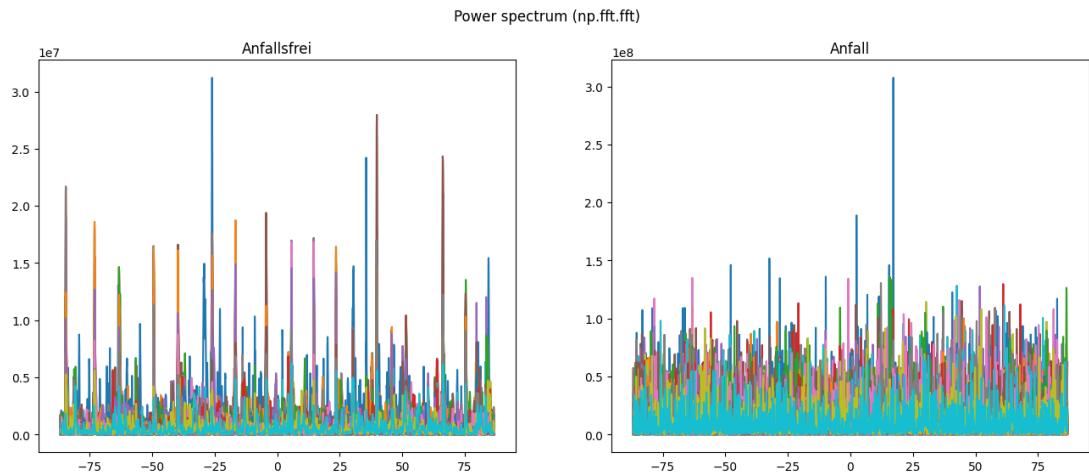
```
[28]: freqs = np.fft.fftfreq(4096, 1/173.61)
idx = np.argsort(freqs)

ps_D = np.abs(np.fft.fft(df_D)) ** 2
ps_E = np.abs(np.fft.fft(df_E)) ** 2

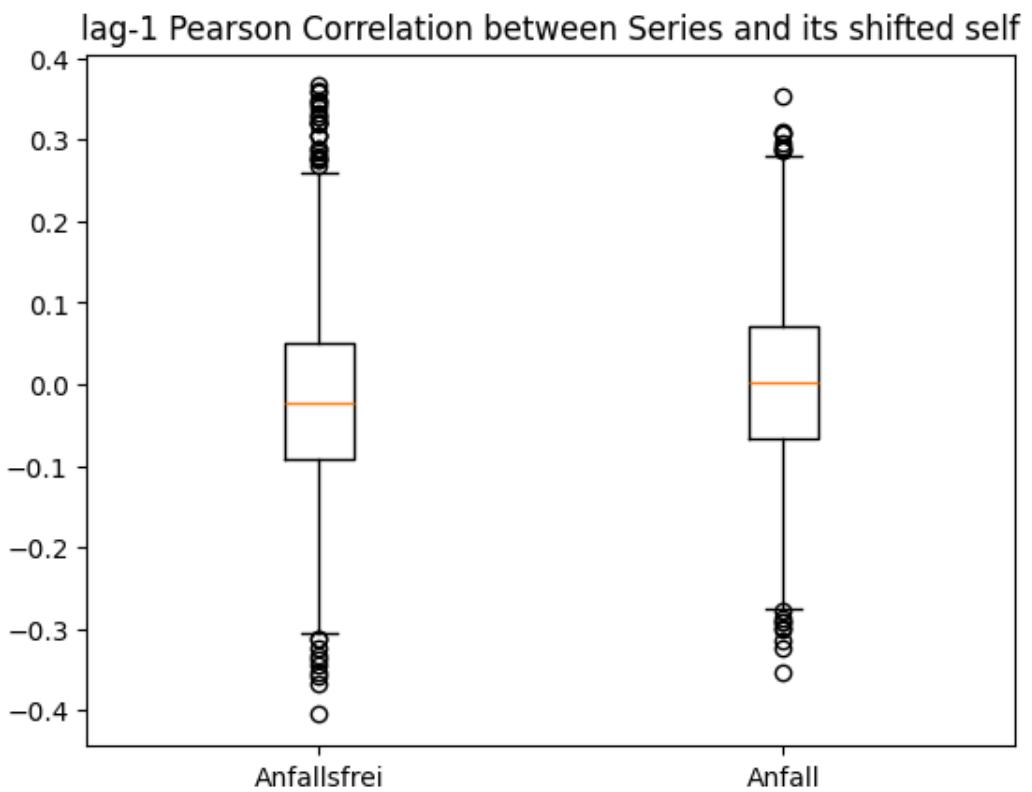
fig, axs = plt.subplots(1, 2, figsize=(16, 6))

axs[0].plot(freqs[idx], ps_D[idx])
axs[0].set_title("Anfallsfrei")
axs[1].plot(freqs[idx], ps_E[idx])
axs[1].set_title("Anfall")

fig.suptitle('Power spectrum (np.fft.fft)')
plt.show()
```



```
[29]: plt.title("lag-1 Pearson Correlation between Series and its shifted self")
plt.boxplot([df_D.agg(pd.Series.autocorr, axis="columns"), df_E.agg(pd.Series.
    ↪autocorr, axis="columns")], tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```

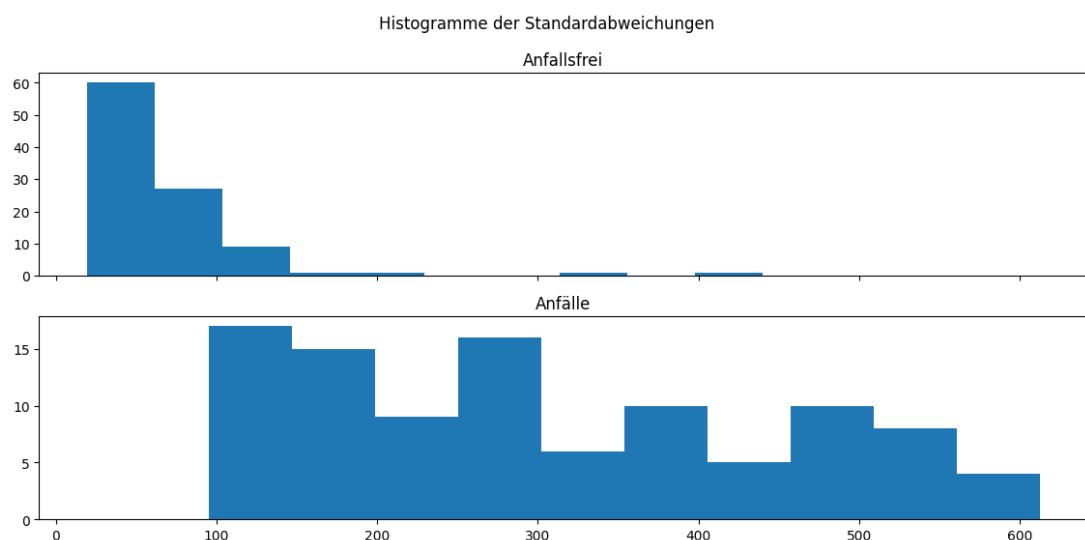


- (4) Visualisieren Sie in einem Histogramm die beiden Verteilungen des Merkmals, für das Sie sich in Schritt 3 entschieden haben.

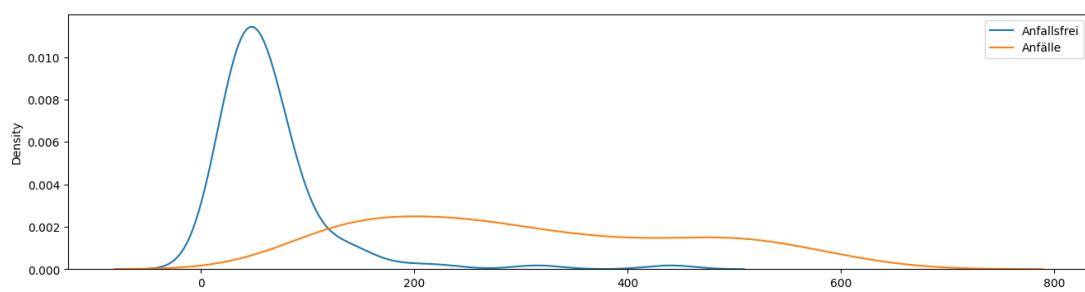
```
[30]: fig, axs = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

fig.suptitle("Histogramme der Standardabweichungen")
axs[0].hist(df_D.std())
axs[0].set_title("Anfallsfrei")

axs[1].hist(df_E.std())
axs[1].set_title("Anfälle")
plt.show()
```



```
[31]: plt.figure(1, (16, 4))
sns.kdeplot(df_D.std(), label="Anfallsfrei")
sns.kdeplot(df_E.std(), label="Anfälle")
plt.legend()
plt.show()
```



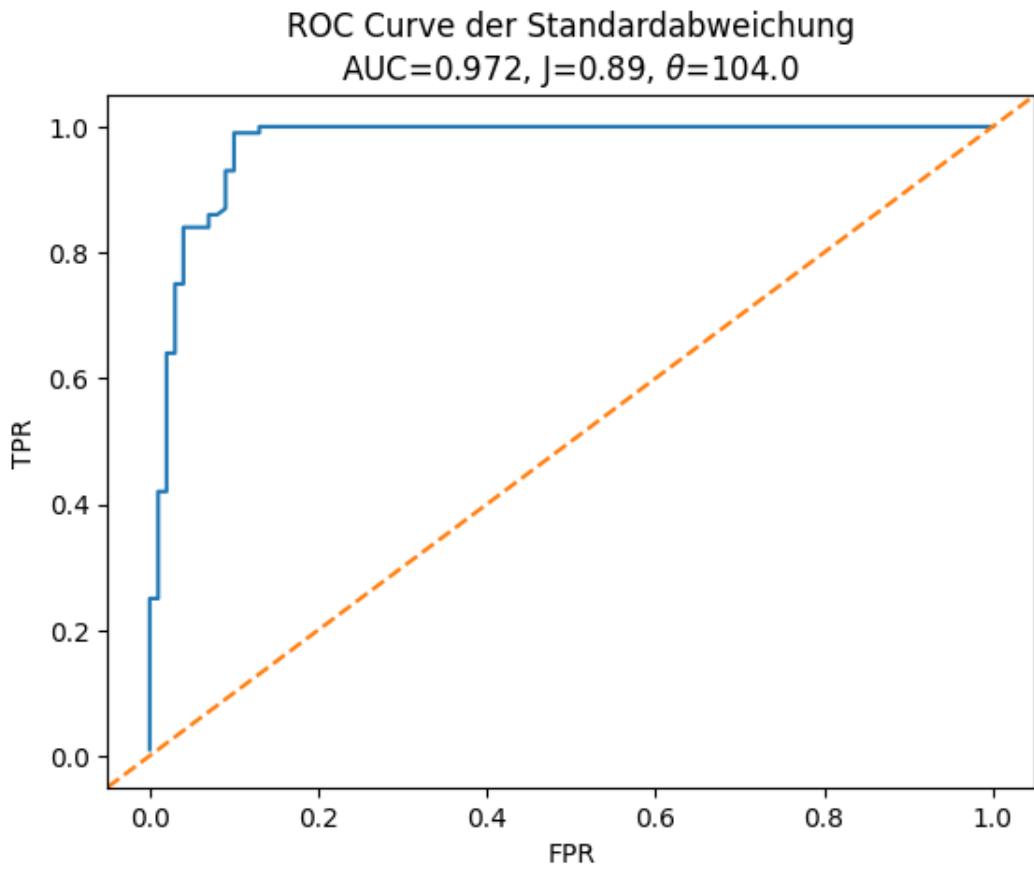
- (5) Visualisieren Sie die ROC-Kurve für Ihr Merkmal. Erstellen Sie zunächst eine Liste mit Schwellwerten, die Sie ausprobieren wollen. Nutzen Sie dann Ihren Code aus Aufgabe 11.1, um die ROC-Kurve zu erzeugen und zu visualisieren. Ermitteln Sie mithilfe des Youden-Index den optimalen Schwellwert für Ihr Klassifikationsproblem.

```
[32]: schwellwerte: np.ndarray = np.arange(-50, 600, 1)

roc = []
youdens = []
for s in schwellwerte:
    TPR, FPR, _, _, J = calc_classifier_metrics(df_E.std().to_numpy(), df_D.
    ↪std().to_numpy(), s, output=False)
    roc.append([FPR, TPR])
    youdens.append(J)

roc = np.array(roc)
plt.plot(roc[:, 0], roc[:, 1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.axline((0, 0), slope=1, c="C1", linestyle="--")
phi_tilde_idx = np.argmax(youdens)
plt.title(f"ROC Curve der Standardabweichung\nAUC={calc_AUC(roc):.3f},\n"
    ↪J={youdens[phi_tilde_idx]:.2f}, $\theta$={schwellwerte[phi_tilde_idx]:.1f}")
plt.show()

print(f"Ein Klassifikator mit der Standardabweichung liefert ein AUC ="
    ↪{calc_AUC(roc)}")
print(f"Der optimale Schwellenwert liegt dafür bei"
    ↪{schwellwerte[phi_tilde_idx]} mit J = {youdens[phi_tilde_idx]}")
```



Ein Klassifikator mit der Standardabweichung liefert ein AUC = 0.9720500000000001

Der optimale Schwellenwert liegt dafür bei 104 mit $J = 0.89$

- (6) Ermitteln Sie die AUC für das von Ihnen gewählte Merkmal und geben Sie sie an.

$$\text{AUC} = 0.972$$

Damit darf ich Ihnen gratulieren. Sie haben soeben einen Klassifikator erzeugt, der anzeigt, ob wir es mit einem epileptischen Anfall zu tun haben oder nicht. Daneben haben Sie die Qualität des Klassifikators und der Merkmale mithilfe der AUC bewertet.

- (7) [Optional] Diese Aufgabe richtet sich nur an die Tüftler und Bastler unter Ihnen, die schon alle anderen Aufgaben dieser Übung gelöst haben und noch Lust und Zeit auf weitere Analysen haben. Ihre Aufgabe ist wie folgt: Ermitteln Sie die AUC für Features, die der Gesamtpower in den EEG-Frequenzbändern Delta (0-4 Hz), Theta (4-8 Hz), Alpha (8-12 Hz), Beta (12-30 Hz) und Gamma (30-45 Hz) entsprechen. Wie bewerten Sie diese Merkmale hinsichtlich ihrer Klassifikationsleistung im Vergleich zu dem von Ihnen gewählten Merkmal aus Schritt (3)?
- Achtung: Sie verlassen in dieser Teilaufgabe den “geführten” Bereich. Das bedeutet: Eigene

Recherche, was eine Fouriertransformation und ein Powerspektrum ist und wie dieses berechnet werden kann. Die folgenden zwei Links können Ihnen dabei helfen: [rfft](#), [hamming](#).

Um die Gesamtpower in den Frequenzbändern zu berechnen, benötigen wir die *Spektrale Leistungsdichte* (oder auch Power Spectral Density, PSD genannt). Diese berechnet sich über

$$G_{xx}(f) = \frac{P(f)}{B_{\text{eff}}}$$

wobei - das Leistungsspektrum $P(f) = \frac{|X(f)|^2}{2}$ mit der Fouriertransformierten Zeitreihe $X(f)$ - die Effektive Bandbreite $B_{\text{eff}} = \frac{\text{PM}}{\text{FM}^2}$ - der Leistungsmittelwert $\text{PM} = \frac{1}{N} \sum_{k=0}^{N-1} w^2(k)$ - der Fenstermittelwert $\text{FM} = \frac{1}{N} \sum_{k=0}^{N-1} w(k)$ - und $w(n)$ die Hamming Funktion

ist. Für uns relevant ist jedoch nur $P(f)$.

Durch Anwendung der Fouriertransformation `np.fft.rfft` transformieren wir von der Zeit- in den Frequenzbereich.

```
[33]: def calc_power_of_freqbands(df: pd.DataFrame, sample_rate: float) -> pd.
    DataFrame:
    h = np.hamming(df.shape[0]).reshape(-1, 1)

    # Anwendung einer Fast Fourier Transformation (FFT) auf die Zeitreihen zur
    # Extraktion von Frequenzkomponenten
    fourier = np.fft.rfft(df * h, axis=0)
    freqs = np.fft.rfftfreq(df.shape[0], 1./sample_rate)

    # Power-Spektrum aus der FFT berechnen um die, in den verschiedenen
    # Frequenzbereichen enthaltene, Leistung zu berechnen
    psd = np.abs(fourier)

    frequency_bands = pd.DataFrame(np.abs(psd), freqs)

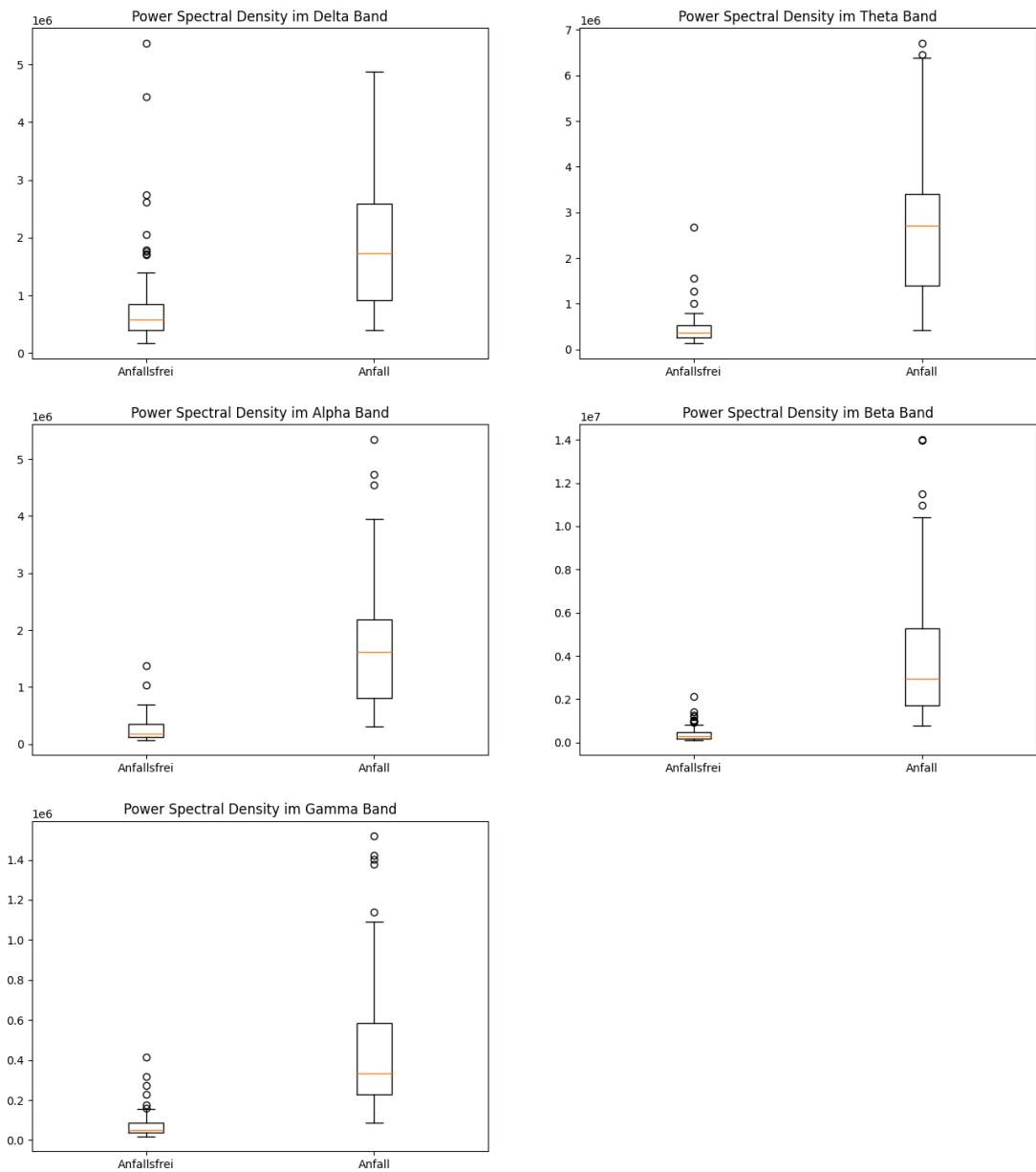
    # Frequenzbänder
    freq_bands = {
        "delta": (0, 4),
        "theta": (4, 8),
        "alpha": (8, 12),
        "beta": (12, 30),
        "gamma": (30, 45),
    }
    df_powers = pd.DataFrame()
    for band, (a, b) in freq_bands.items():
        df_powers[band] = frequency_bands.loc[a:b].sum()
    return df_powers
```

```
[34]: sample_rate = 173.61
df_Dpow = calc_power_of_freqbands(df_D, sample_rate)
df_Epow = calc_power_of_freqbands(df_E, sample_rate)
```

```
[35]: fig, axs = plt.subplots(3, 2, figsize=(16, 18))
for i, col in enumerate(df_Dpow.columns):
    axs.flat[i].set_title(f"Power Spectral Density im {col.capitalize()} Band")
    axs.flat[i].boxplot([df_Dpow[col], df_Epow[col]],  

    ↪ tick_labels=["Anfallsfrei", "Anfall"])

axs[-1, -1].axis('off')
plt.show()
```



```
[36]: def roc_plot(df_p, df_n, classifier: str):
    schwellwerte: np.ndarray = np.linspace(min(df_p.min(), df_n.min()), max(df_p.max(), df_n.max()), 100)

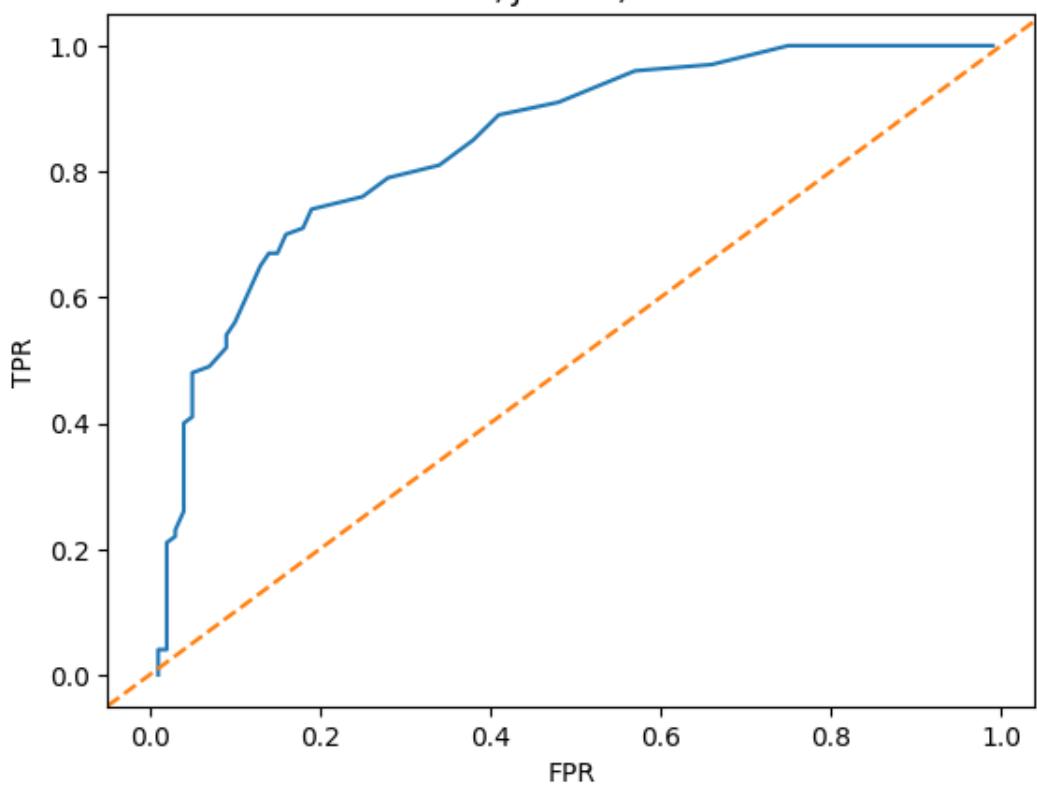
    roc = []
    youdens = []
    for s in schwellwerte:
        try:
            TPR, FPR, _, _, _ , J = calc_classifier_metrics(df_p.to_numpy(), df_n.to_numpy(), s, output=False)
            roc.append([FPR, TPR])
            youdens.append(J)
        except ZeroDivisionError:
            continue

    roc = np.array(roc)
    phi_tilde_idx = np.argmax(youdens)

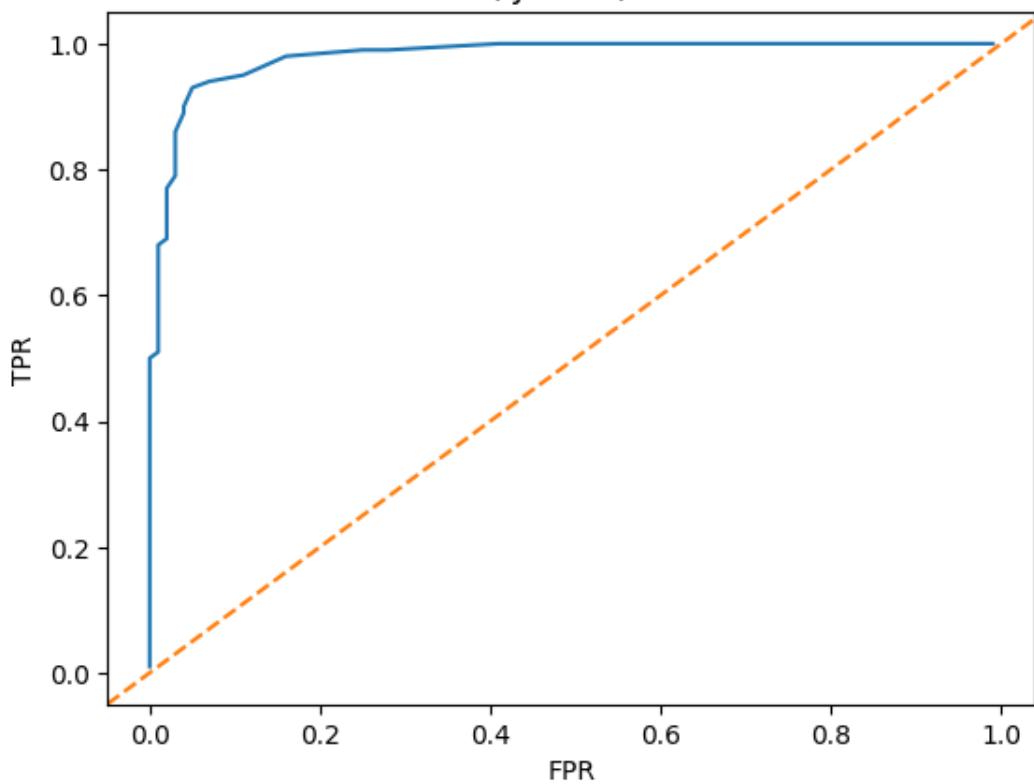
    plt.plot(roc[:, 0], roc[:, 1])
    plt.title(f"ROC Curve: {classifier}\nAUC={calc_AUC(roc):.3f}, \nJ={youdens[phi_tilde_idx]:.2f}, \\\theta=[schwellwerte[phi_tilde_idx]:.1f]")
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.axline((0, 0), slope=1, c="C1", linestyle="--")
    plt.show()

for i, col in enumerate(df_Dpow.columns):
    roc_plot(df_Epow[col], df_Dpow[col], f"PSD auf dem {col.capitalize()}-Band")
```

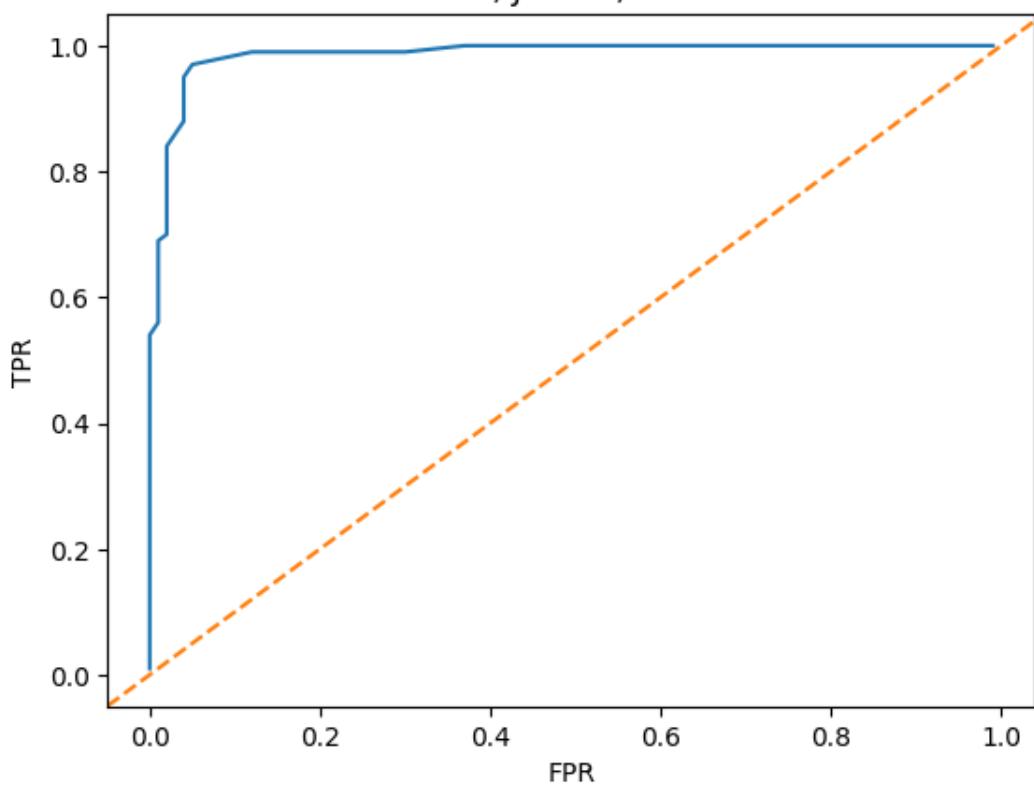
ROC Curve: PSD auf dem Delta-Band
AUC=0.832, J=0.55, $\theta=954272.9$



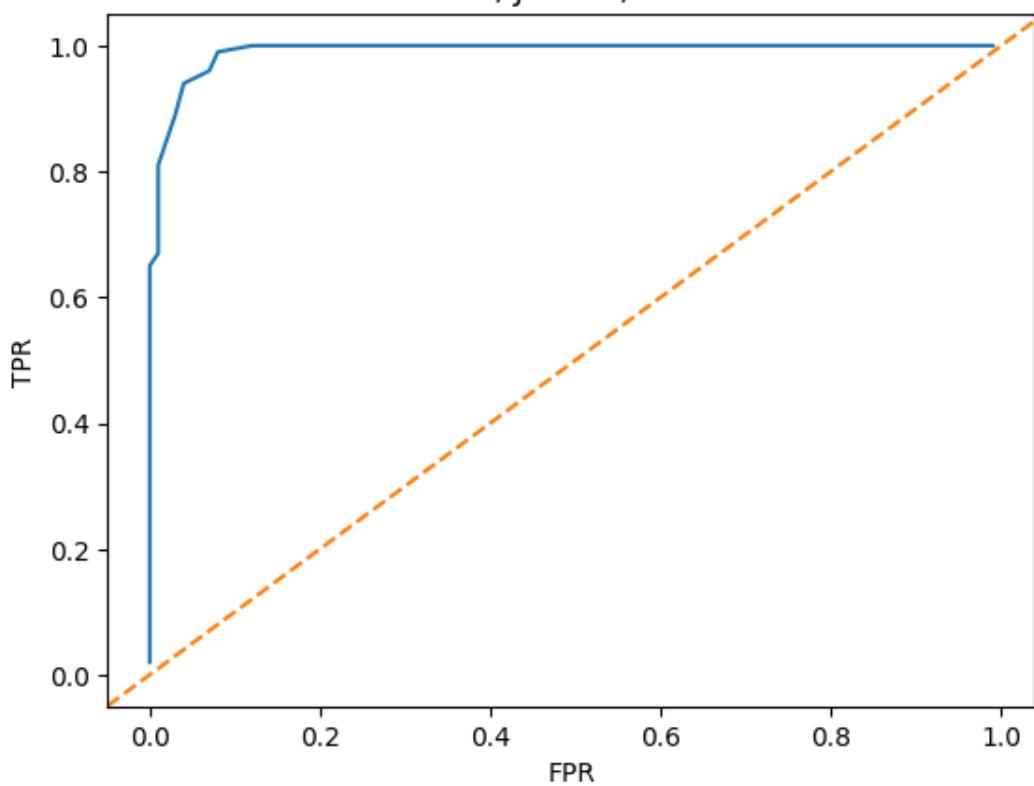
ROC Curve: PSD auf dem Theta-Band
AUC=0.970, J=0.88, $\theta=792258.9$

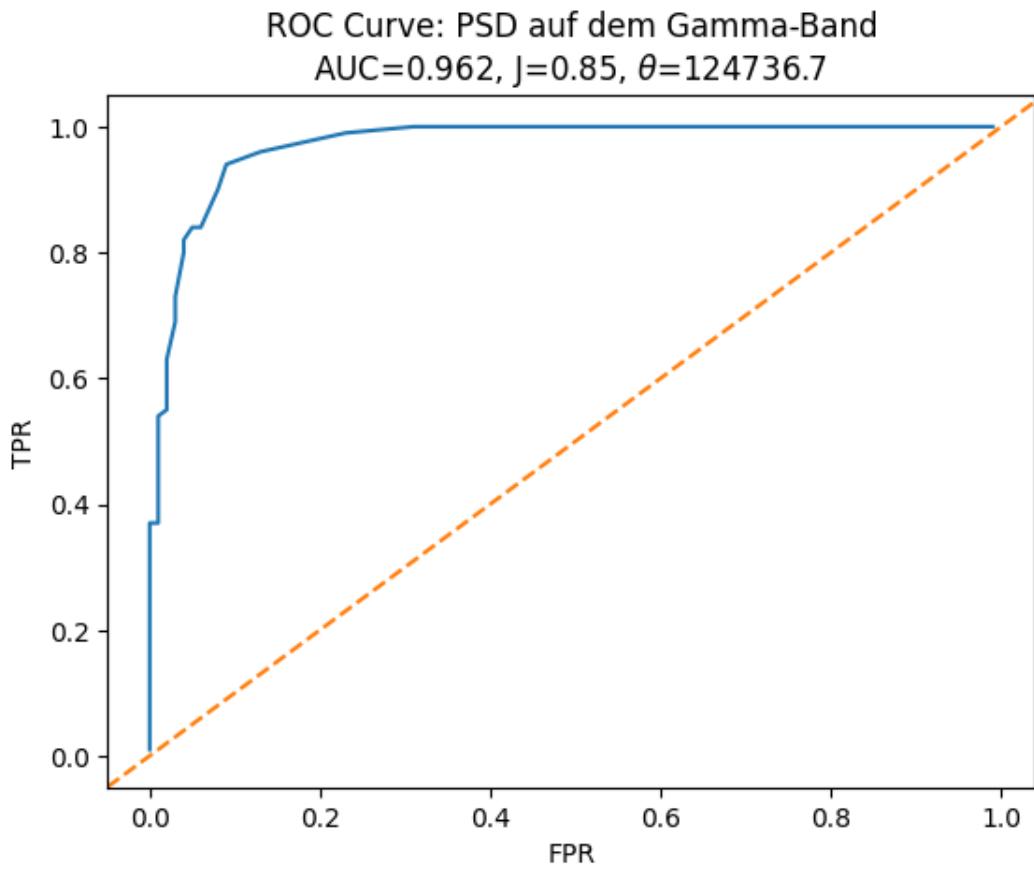


ROC Curve: PSD auf dem Alpha-Band
AUC=0.976, J=0.92, $\theta=497822.9$



ROC Curve: PSD auf dem Beta-Band
AUC=0.981, J=0.91, $\theta=820426.7$





1.0.4 11.3 [Optional] Weinqualitäten (Multiklassen-Klassifikation, Feature Engineering)

In dieser Übung werden wir ein Nächste Nachbarn (NN) Modell erstellen, mit dem wir die Klasse eines Weines (d.h. die Kultursorte) aus den Eigenschaften vorhersagen können. Wir werden mit PCA-transformierten Merkmalen arbeiten.

Ihre Daten

Zu Beginn der 90er Jahre wurden verschiedene Weinproben in einer Region Italiens untersucht. Die Weine stammen von drei verschiedenen Kultursorten. Diese Kultursorten werden im unten hinterlegten Datensatz als Klasse 1, 2 und 3 (*class labels*) bezeichnet. Unter den 13 untersuchten Merkmalen finden Sie neben chemischen Eigenschaften (Alkoholgehalt, Säuregehalt) auch physikalische Eigenschaften (Farbintensität, etc).

Ich habe Ihnen den Weindatensatz in zwei Teile geteilt: Der erste Datensatz ist der sogenannte Trainingsdatensatz, mithilfe dessen Sie ihr Modell bauen werden. Der zweite Datensatz ist der sogenannte Testdatensatz, auf dem Sie ihr Modell anwenden und testen werden.

Ihre Aufgaben

(1) Importieren Sie die Daten, indem Sie die folgende Code-Zelle ausführen.

```
[37]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from scipy.stats import zscore
from sklearn.model_selection import train_test_split

# import data
column_names = ['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']
df = pd.read_csv('wine.data', header=None)
df.columns = column_names

# preprocess data
X, y = df.iloc[:, 1:].values, df.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=10)

X_train, X_test = zscore(X_train), zscore(X_test)

# apply PCA
pca = PCA(n_components=2)
X_train_p = pca.fit_transform(X_train)
X_test_p = pca.transform(X_test)
```

(2) Vergegenwärtigen Sie sich die Eigenschaften des Wein-Datensatzes: Untersuchen Sie dazu das Pandas-Objekt. Welche Eigenschaften wurden für die Weine erfasst? Wie viele Weinproben wurden genommen?

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

```
[38]:   Class label  Alcohol  Malic acid  Ash  Alcalinity of ash  Magnesium \
0            1     14.23      1.71  2.43           15.6       127
1            1     13.20      1.78  2.14           11.2       100
2            1     13.16      2.36  2.67           18.6       101
3            1     14.37      1.95  2.50           16.8       113
4            1     13.24      2.59  2.87           21.0       118

   Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins \
0            2.80      3.06                  0.28          2.29
1            2.65      2.76                  0.26          1.28
2            2.80      3.24                  0.30          2.81
3            3.85      3.49                  0.24          2.18
```

4	2.80	2.69	0.39	1.82
	Color intensity	Hue OD280/OD315 of diluted wines	Proline	
0	5.64	1.04	3.92	1065
1	4.38	1.05	3.40	1050
2	5.68	1.03	3.17	1185
3	7.80	0.86	3.45	1480
4	4.32	1.04	2.93	735

Es wurden 13 verschiedene Eigenschaften (Class label nicht mitgezählt) für 178 Weinproben erfasst, wobei zwischen 3 Klassen unterschieden wird.

- (3) Lesen Sie den Code ab dem Kommentar `# preprocess data`. Was passiert in diesen Code-Zeilen? Welches Feature-Engineering findet statt, bis wir zu den Daten `X_train_p`, `X_test_p` angelkommen sind, mit denen Sie dann arbeiten werden? Wie viele Features (Merkmale) haben die transformierten Daten?

Das Dataframe wird in die 13 Features `X` und die Class Labels `y` aufgeteilt. Dann findet ein Split der Daten in das Trainingsset (60%) und das Testset (40%) statt.

Bei der nachfolgenden Normierung der Features werden allerdings die Trainingsdaten und die Testdaten separat voneinander normiert, was zu data leakage führt. Es sollte eine Normierung der Trainingsdaten stattfinden und dann die gleichen Normierungsparameter auf das Testset angewandt werden.

Nach der Normierung findet eine PCA auf $n = 2$ statt.

Um *Data Leakage* zu vermeiden, empfiehlt sich die Verwendung einer Pipeline in folgender Form (mögen es uns die Korrigierenden verzeihen xD):

```
[39]: from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# preprocess data
X, y = df.iloc[:, 1:].values, df.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, ↴
random_state=10)

pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA(n_components=2))])

X_train_p = pipe.fit_transform(X_train)
X_test_p = pipe.transform(X_test)
```

- (4) Visualisieren Sie in zwei Scatterplots den PCA-transformierten Trainingsdatensatz sowie den Testdatensatz. Färben Sie die Punkte in beiden Plots ein gemäß der Klassenzugehörigkeit, wie Sie in Ihrem Vektor `y_train` bzw. `y_test` kodiert ist. Die Einträge in `y_train` und `y_test` werden auch *Labels* genannt.

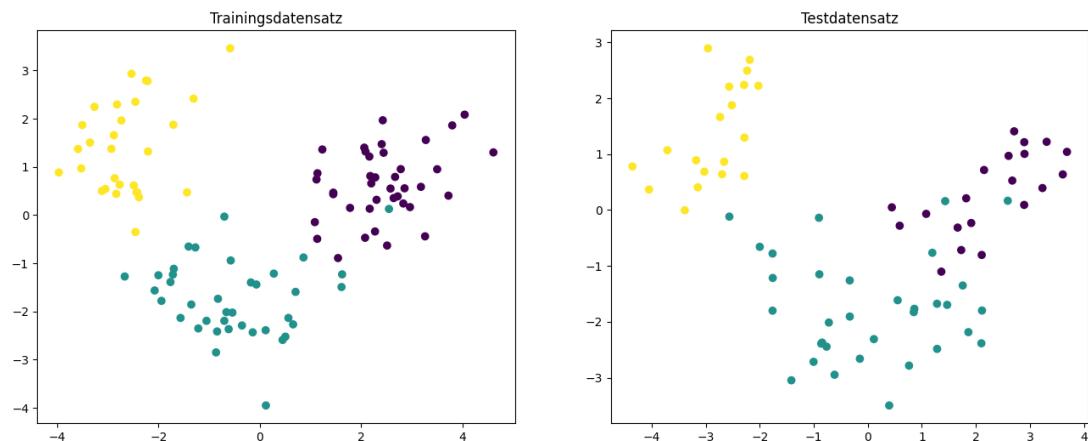
```
[40]: fig, axs = plt.subplots(1, 2, figsize=(16, 6))
```

```

axs[0].set_title("Trainingsdatensatz")
axs[0].scatter(X_train_p[:, 0], X_train_p[:, 1], c=y_train)
axs[1].set_title("Testdatensatz")
axs[1].scatter(X_test_p[:, 0], X_test_p[:, 1], c=y_test)

plt.show()

```



- (5) Wir werden jetzt ein NN-Modell (Nächste Nachbarn Modell) erstellen. Schreiben Sie dazu eine Funktion mit dem Namen `NN`, die die Trainingsdaten (`X_train_p` und `y_train_p`) sowie die Testdaten (`X_test_p`) entgegen nimmt und die vorhergesagten Klassen (Labels) für die Testdaten als Vektor (`y_pred`) zurückgibt. Schlagen Sie in den Vorlesungsfolien nach, wie ein NN Modell definiert ist und nutzen Sie für die Implementierung numpy Funktionen. Broadcasting kann Ihnen ebenfalls sehr hilfreich sein.

```
[41]: def NN(X_train: np.ndarray, y_train: np.ndarray, X: np.ndarray) -> np.ndarray:
    y2 = np.sum(X_train**2, axis=1)
    x2 = np.sum(X**2, axis=1).reshape(-1, 1)
    distances = np.sqrt(x2 - 2*(X @ X_train.T) + y2)
    return y_train[np.argmin(distances, axis=1)]
```

- (6) Sie haben in Schritt (5) ein einfaches Machine Learning Modell implementiert, einen Klassifikator. Sagen Sie mithilfe Ihrer Funktion die Labels (Klassen) der Weinproben des Testdatensatzes voraus und speichern Sie die Voraussage im Vektor `y_pred`.

```
[42]: y_pred: np.ndarray = NN(X_train_p, y_train, X_test_p)
```

- (7) Bestimmen Sie die *Accuracy* Ihrer Vorhersage, also den Anteil der korrekt vorhergesagten Klassen dividiert durch die Gesamtanzahl aller Vorhersagen. Beurteilen Sie anhand der *Accuracy*, ob Ihr Modell die Klassen des Testdatensatzes gut vorhersagen kann.

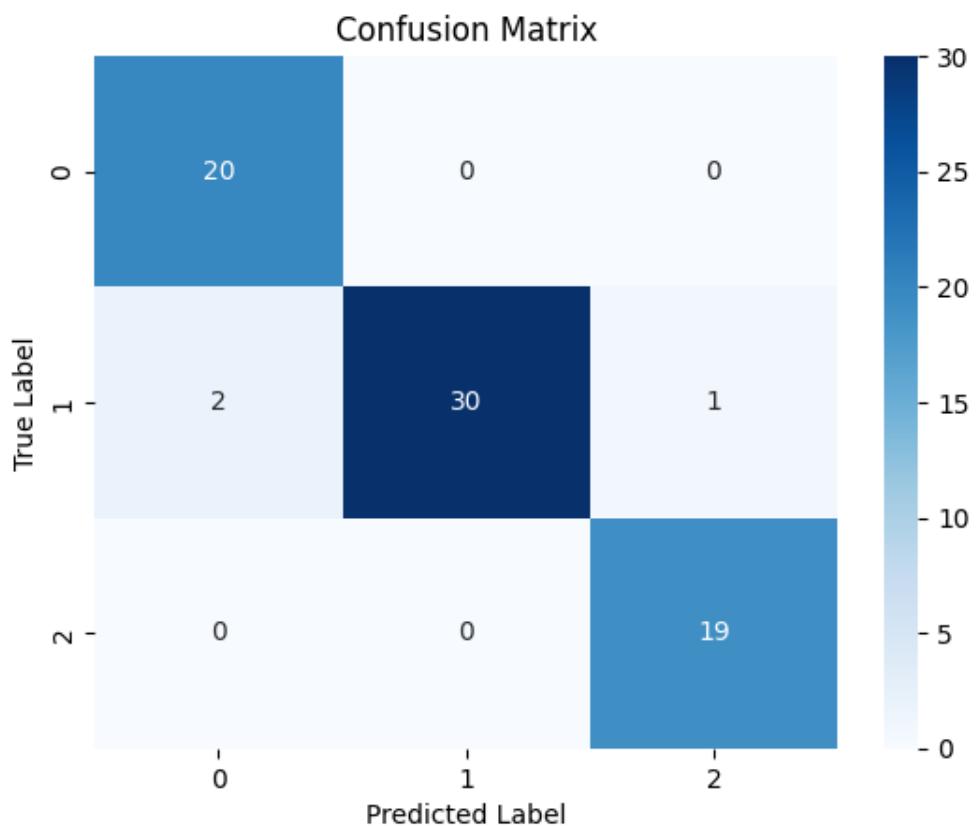
```
[43]: (y_pred == y_test).sum() / y_pred.size
```

```
[43]: np.float64(0.9583333333333334)
```

Es gelingt eine korrekte Vorhersage mit einer *Accuracy* von rund 95.83% (bei fehlerhafter Normierung der Testdaten 94.4%), sodass man sagen kann, dass unserem Nearest Neighbor Modell die Vorhersage gut gelingt.

- (8) Bestimmen Sie eine *Confusion Matrix*, um Ihren Klassifikator besser einschätzen zu können:
1. Schlagen Sie in den Folien der Vorlesung nach, wie eine Confusion Matrix aufgebaut wird.
 2. Bestimmen Sie die Confusion Matrix Ihres Klassifikators auf dem Testdatensatz.
 3. Visualisieren Sie die Confusion Matrix mithilfe der Matplotlib.

```
[44]: def confusion_matrix(y: np.ndarray, y_pred: np.ndarray):  
    classes: np.ndarray = np.unique(y)  
    array = []  
    for true_label in classes:  
        row = []  
        for predicted_label in classes:  
            row.append(((y == true_label) & (y_pred == predicted_label)).sum())  
        array.append(row)  
  
    confusion_matrix: np.ndarray = np.array(array)  
    sns.heatmap(confusion_matrix, annot=True, cmap='Blues')  
  
    plt.title("Confusion Matrix")  
    plt.ylabel("True Label")  
    plt.xlabel("Predicted Label")  
    plt.show()  
  
confusion_matrix(y_test, y_pred)
```



- (9) Interpretieren Sie die *Confusion Matrix*: Welche Klassen werden besser vorhergesagt, welche schlechter?

Bei den Klassen 0 und 2 gelingt die Vorhersage sehr zuverlässig, während bei der Klasse 1 drei Fehlzuordnungen passieren.

TSRI-12

July 23, 2024

1 Übung 12

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden wir uns mit daten-getriebener Modellierung mithilfe von Nächste Nachbarn Modellen beschäftigen. Nächste Nachbarn Modelle gehören zu den einfachsten Machine Learning Modellen. Viele weitere Modelle können Sie im Bachelor Kurs “Machine Learning” kennenlernen. Ich lade Sie herzlich dazu ein, sich für den Kurs Bachelorkurs “Machine Learning” anzumelden, sollten Sie Interesse haben.

1.0.2 12.1 Weinqualitäten (Multiklassen-Klassifikation, Feature Engineering)

- Diese Übung kennen Sie in abgewandelter Form aus der optionalen Übung 11.3. Falls Sie Übung 11.3 schon bearbeitet haben, können Sie sich von dort für eine Lösung hier inspirieren lassen.

In dieser Übung werden wir ein Nächste Nachbarn (NN) Modell erstellen, mit dem wir die Klasse eines Weines (d.h. die Kultursorte) aus den Eigenschaften vorhersagen können. Wir werden mit PCA-transformierten Merkmalen arbeiten.

Ihre Daten

Zu Beginn der 90er Jahre wurden verschiedene Weinproben in einer Region Italiens untersucht. Die Weine stammen von drei verschiedenen Kultursorten. Diese Kultursorten werden im unten hinterlegten Datensatz als Klasse 1, 2 und 3 (*class labels*) bezeichnet. Unter den 13 untersuchten Merkmalen finden Sie neben chemischen Eigenschaften (Alkoholgehalt, Säuregehalt) auch physikalische Eigenschaften (Farbintensität, etc).

Ich habe Ihnen den Weindatensatz in zwei Teile geteilt: Der erste Datensatz ist der sogenannte Trainingsdatensatz, mithilfe dessen Sie ihr Modell bauen werden. Der zweite Datensatz ist der sogenannte Testdatensatz, auf dem Sie ihr Modell anwenden und testen werden.

```
[1]: import numpy as np  
import pandas as pd
```

```

from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

# import data
column_names = ['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)
df.columns = column_names

# preprocess data
X, y = df.iloc[:, 1:].values, df.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=10)

# apply PCA
pca = PCA(n_components=2)
X_train_p = pca.fit_transform(X_train)
X_test_p = pca.transform(X_test)

```

Ihre Aufgaben

- (1) Importieren Sie die Daten, indem Sie die obere Code-Zelle ausführen.
- (2) Vergegenwärtigen Sie sich die Eigenschaften des Wein-Datensatzes: Untersuchen Sie dazu das Pandas-Objekt. Welche Eigenschaften wurden für die Weine erfasst? Wie viele Weinproben wurden genommen?

[2]: `print(f"Erfasste Eigenschaften von {df.shape[0]} Weinproben: {', '.join(list(df.columns[1:]))}")`

Erfasste Eigenschaften von 178 Weinproben: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, Proline

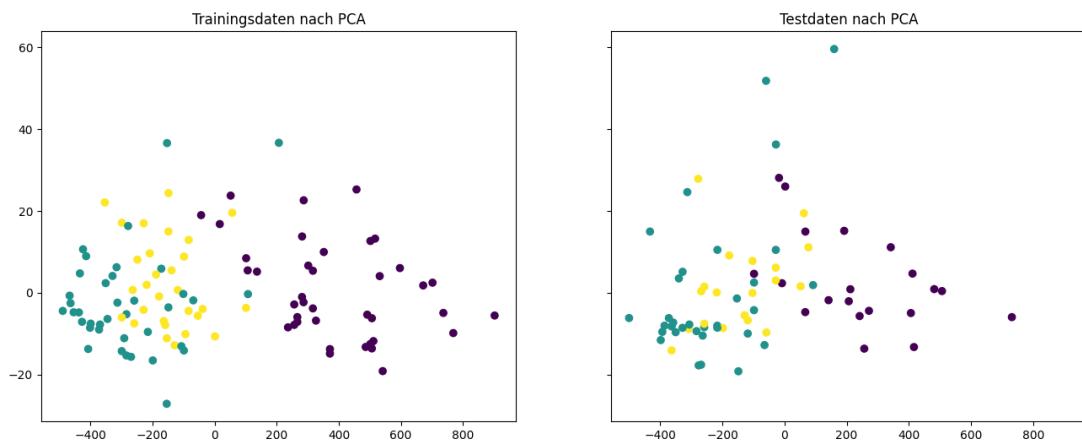
- (3) Lesen Sie den Code ab dem Kommentar `# preprocess data`. Was passiert in diesen Code-Zeilen? Welches Feature-Engineering findet statt, bis wir zu den Daten `X_train_p`, `X_test_p` angekommen sind, mit denen Sie dann arbeiten werden? Wie viele Features (Merkmale) haben die transformierten Daten?

Es wurde eine PCA der Daten von 14 auf 2 Dimensionen durchgeführt.

- (4) Visualisieren Sie in zwei Scatterplots den PCA-transformierten Trainingsdatensatz sowie den Testdatensatz. Färben Sie die Punkte in beiden Plots ein gemäß der Klassenzugehörigkeit, wie Sie in Ihrem Vektor `y_train` bzw. `y_test` kodiert ist. Die Einträge in `y_train` und `y_test` werden auch *Labels* genannt.

- Hinweis: Vielleicht wollen Sie sich an Ihrem Code aus früheren Übungen bedienen, wo Sie eine ähnliche Aufgabe schon einmal gelöst haben.

```
[3]: fix, axs = plt.subplots(1, 2, figsize=(16, 6), sharey=True)
axs[0].scatter(X_train_p[:, 0], X_train_p[:, 1], c=y_train)
axs[0].set_title("Trainingsdaten nach PCA")
axs[1].scatter(X_test_p[:, 0], X_test_p[:, 1], c=y_test)
axs[1].set_title("Testdaten nach PCA")
axs[1].set_xlim(*axs[0].get_xlim())
plt.show()
```



- (5) Wir werden jetzt ein NN-Modell (Nächste Nachbarn Modell) erstellen. Schreiben Sie dazu eine Funktion mit dem Namen `NN`, die die Trainingsdaten (`X_train_p` und `y_train`) sowie die Testdaten (`X_test_p`) entgegen nimmt und die vorhergesagten Klassen (Labels) für die Testdaten als Vektor (`y_pred`) zurückgibt. Schlagen Sie in den Vorlesungsfolien nach, wie ein NN Modell definiert ist und nutzen Sie für die Implementierung numpy Funktionen. Broadcasting kann Ihnen ebenfalls sehr hilfreich sein.

```
[4]: def NN(X_train: np.ndarray, y_train: np.ndarray, X: np.ndarray) -> np.ndarray:
    y2 = np.sum(X_train**2, axis=1)
    x2 = np.sum(X**2, axis=1).reshape(-1, 1)
    distances = np.sqrt(x2 - 2*(X @ X_train.T) + y2)
    return y_train[np.argmin(distances, axis=1)]
```

- (6) Sie haben in Schritt (5) ein einfaches Machine Learning Modell implementiert, einen Klassifikator. Sagen Sie mithilfe Ihrer Funktion die Labels (Klassen) der Weinproben des Testdatensatzes voraus und speichern Sie die Voraussage im Vektor `y_pred`.

```
[5]: y_pred: np.ndarray = NN(X_train_p, y_train, X_test_p)
```

- (7) Bestimmen Sie die *Accuracy* Ihrer Vorhersage, also den Anteil der korrekt vorhergesagten Klassen dividiert durch die Gesamtanzahl aller Vorhersagen. Beurteilen Sie anhand der *Accuracy*, ob Ihr Modell die Klassen des Testdatensatzes gut vorhersagen kann.

```
[6]: def Accuracy(y_test: np.ndarray, y_pred: np.ndarray) -> float:
    return (y_pred == y_test).sum() / y_pred.size

Accuracy(y_test, y_pred)
```

```
[6]: np.float64(0.6944444444444444)
```

Mit einer Accuracy von $\text{ACC} \approx 69,4\%$ ist die Vorhersage eher schlecht.

- (8) Wiederholen Sie Schritt (5), allerdings für Trainingsdaten, deren Features Sie auf Mittelwert 0 und Varianz 1 **normiert** haben:
- Normieren Sie also die ursprünglichen Trainingsdaten **vor der PCA**, führen Sie dann die PCA mit den normierten Daten durch und klassifizieren Sie die aus der PCA resultierenden Daten. Welche Genauigkeit (Accuracy) erhalten Sie nun auf den Trainingsdaten? Wie erklären Sie sich diese Veränderung im Vergleich zu dem Ergebnis aus Schritt (7)?
 - Normieren Sie anschließend auch die Testdaten, und zwar so, dass diese mit denselben Parametern wie die Trainingsdaten normiert werden. Dies bedeutet: Fitten Sie den StandardScaler auf die Trainingsdaten (`.fit`) und wenden Sie diese Transformation nun auch auf die Testdaten an (`.transform`).

```
[7]: from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

# preprocess data
X, y = df.iloc[:, 1:].values, df.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, ↴random_state=10)

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2))
])

X_train_p = pipe.fit_transform(X_train)
X_test_p = pipe.transform(X_test)

y_pred = NN(X_train_p, y_train, X_test_p)

Accuracy(y_test, y_pred)
```

```
[7]: np.float64(0.9583333333333334)
```

- (9) Wir werden nun untersuchen, wie sich die *Accuracy* mit der Anzahl k der nächsten Nachbarn ändert. Nutzen Sie die **Implementierung des kNN Modells von scikit learn**, um die Accuracy als Funktion von k zu ermitteln ($k \in \{1, \dots, 50\}$).

Hinweise

- Trainieren Sie Ihr Modell auf dem Trainingsset.
- Ermitteln Sie die Accuracies auf den Testsets.

```
[8]: from sklearn.neighbors import KNeighborsClassifier

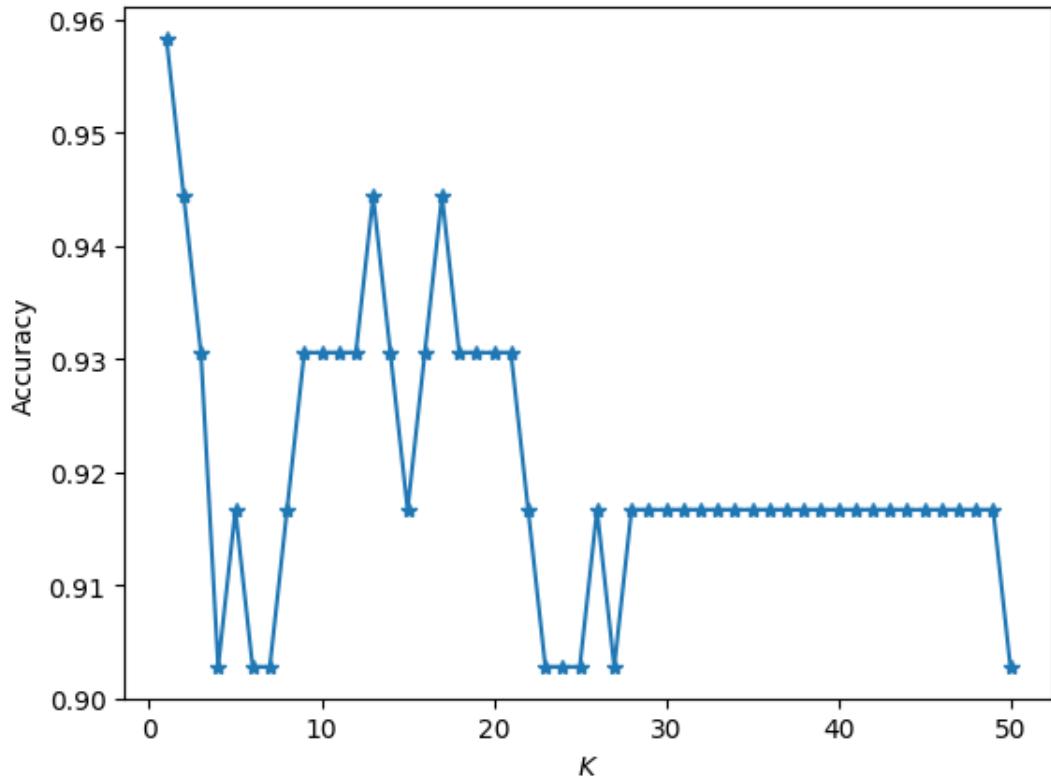
accuracies = []
for k in range(1, 51):
    knn_classifier = KNeighborsClassifier(n_neighbors=k).fit(X_train_p, y_train)
    y_pred = knn_classifier.predict(X_test_p)
    accuracy = (y_pred == y_test).sum() / y_pred.size
    accuracies.append(accuracy)
print(f"k={k:2d} ACC = {accuracy}")
```

k= 1 ACC = 0.9583333333333334
k= 2 ACC = 0.9444444444444444
k= 3 ACC = 0.9305555555555556
k= 4 ACC = 0.9027777777777778
k= 5 ACC = 0.9166666666666666
k= 6 ACC = 0.9027777777777778
k= 7 ACC = 0.9027777777777778
k= 8 ACC = 0.9166666666666666
k= 9 ACC = 0.9305555555555556
k=10 ACC = 0.9305555555555556
k=11 ACC = 0.9305555555555556
k=12 ACC = 0.9305555555555556
k=13 ACC = 0.9444444444444444
k=14 ACC = 0.9305555555555556
k=15 ACC = 0.9166666666666666
k=16 ACC = 0.9305555555555556
k=17 ACC = 0.9444444444444444
k=18 ACC = 0.9305555555555556
k=19 ACC = 0.9305555555555556
k=20 ACC = 0.9305555555555556
k=21 ACC = 0.9305555555555556
k=22 ACC = 0.9166666666666666
k=23 ACC = 0.9027777777777778
k=24 ACC = 0.9027777777777778
k=25 ACC = 0.9027777777777778
k=26 ACC = 0.9166666666666666
k=27 ACC = 0.9027777777777778
k=28 ACC = 0.9166666666666666
k=29 ACC = 0.9166666666666666
k=30 ACC = 0.9166666666666666
k=31 ACC = 0.9166666666666666
k=32 ACC = 0.9166666666666666
k=33 ACC = 0.9166666666666666
k=34 ACC = 0.9166666666666666

```
k=35 ACC = 0.9166666666666666  
k=36 ACC = 0.9166666666666666  
k=37 ACC = 0.9166666666666666  
k=38 ACC = 0.9166666666666666  
k=39 ACC = 0.9166666666666666  
k=40 ACC = 0.9166666666666666  
k=41 ACC = 0.9166666666666666  
k=42 ACC = 0.9166666666666666  
k=43 ACC = 0.9166666666666666  
k=44 ACC = 0.9166666666666666  
k=45 ACC = 0.9166666666666666  
k=46 ACC = 0.9166666666666666  
k=47 ACC = 0.9166666666666666  
k=48 ACC = 0.9166666666666666  
k=49 ACC = 0.9166666666666666  
k=50 ACC = 0.9027777777777778
```

(10) Visualisieren Sie die Accuracy als Funktion von k . Bestimmen Sie den besten Wert k .

```
[9]: plt.plot(range(1, 51), accuracies, "-*")  
plt.xlabel("$K$")  
plt.ylabel("Accuracy")  
plt.show()
```



Bei $k = 1$ haben wir die höchste Accuracy auf den Testdaten erzielt.

Damit darf ich Ihnen gratulieren. Sie haben ein k-Nächste Nachbarn Modell an einen Datensatz angepasst.

1.0.3 12.2 k-nächste-Nachbarn Klassifikation

In dieser kurzen Übungsaufgabe werden Sie sich weiter mit dem kNN-Klassifikator vertraut machen und die Wirkung des Parameters k untersuchen.

Ihre Aufgaben

- (1) Visualisieren Sie die weiter unten erzeugten synthetischen Daten.

```
[10]: import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline

# helper function to plot decision boundaries
def plot_decision_regions(X, y, classifier, axis=plt, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    axis.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)

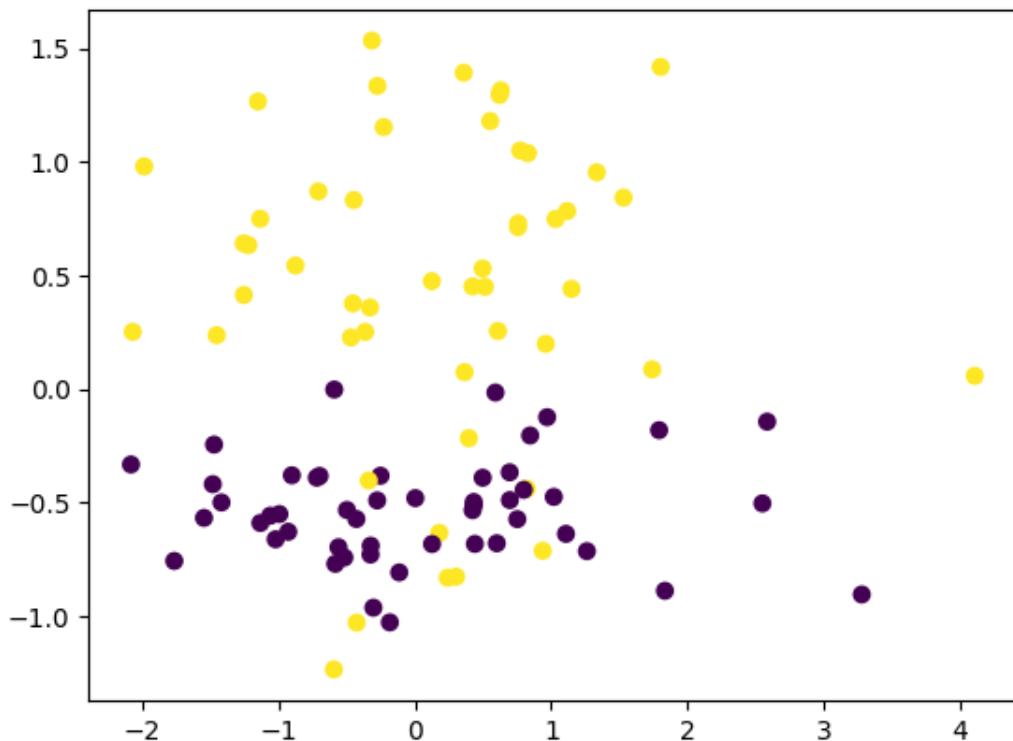
    if not hasattr(axis, "set_xlim"):
        axis = plt.gca()
    axis.set_xlim(xx1.min(), xx1.max())
    axis.set_ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        axis.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                     alpha=0.8, c=np.expand_dims(cmap(idx), 0),
                     marker=markers[idx], label=cl)
```

```
# generate 100 data points: features X, labels y
X, y = make_classification(n_features=2, n_redundant=0, n_informative=1,
                           n_clusters_per_class=1, class_sep=0.5, random_state=2)
```

[11]:

```
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



- (2) Fitten Sie einen **kNN-Klassifikator** an die Daten mit $k = 1$ und visualisieren Sie mit der Hilfsfunktion `plot_decision_regions` die entstandenen Entscheidungsflächen.

[12]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=1).fit(X, y)
plot_decision_regions(X, y, knn_classifier)
```

- (3) Wiederholen Sie Schritt (2) für einige wenige Werte von k . Variieren Sie dabei k zwischen 1 und 50. Wie verändern sich die Entscheidungsflächen? Wie verändert sich, Ihrer Meinung nach, die Komplexität des kNN-Modells mit dem Wert k ?

[13]:

```
k_list = [1, 2, 3, 10, 15, 20, 30, 40, 50]
fig, axs = plt.subplots(len(k_list) // 3, 3, figsize=(16, 16))
for i, k in enumerate(k_list):
    knn_classifier = KNeighborsClassifier(n_neighbors=k).fit(X, y)
```

```
    axs.flat[i].set_title(f"$k={k}$")
    plot_decision_regions(X, y, knn_classifier, axs.flat[i])
```

Je kleiner k ist, desto größer ist die Komplexität des kNN-Modells.

- (4) Machen Sie Ihre Visualisierung aus Schritt (3) interaktiv, indem Sie `ipywidgets` benutzen, und über einen Slider Werte für k auswählen können.

```
[14]: from ipywidgets import interact
import ipywidgets as widgets

def knn_instance(k: int):
    knn_classifier = KNeighborsClassifier(n_neighbors=k).fit(X, y)
    plot_decision_regions(X, y, knn_classifier)

interact(knn_instance, k=widgets.IntSlider(min=1, max=60, step=1))

interactive(children=(IntSlider(value=1, description='k', max=60, min=1), Output()), _dom_classes=('widget-int...')

[14]: <function __main__.knn_instance(k: int)>
```

DS-Probeklausur-Bearbeitung

July 23, 2024

1 Probeklausur “Einführung in Data Science”

1.0.1 FH Aachen - University of Applied Sciences

Professor Dr. Stephan Bialonski

1.1 Name und Matrikelnummer

- Ihr Vor- und Nachname: (bitte ausfüllen)
- Ihre Matrikelnummer: (bitte ausfüllen)

Beachten Sie: **Die Weitergabe dieser Probeklausur an Dritte ist untersagt.**

1.2 Klausurbearbeitung. Dies müssen Sie tun:

1. **Namen und Matrikelnummer eintragen.** Tragen Sie oben Ihren Namen sowie Ihre Matrikelnummer ein, indem Sie die Platzhalter “(bitte ausfüllen)” mit Ihren Angaben ersetzen.
2. **Notebook umbenennen.** Benennen Sie dieses Jupyter Notebook nach folgendem Schema um: **Nachname-Vorname-Matrikelnummer**, wobei Sie für die Platzhalter entsprechend Ihren Namen und Ihre Matrikelnummer eintragen. Sie können das Notebook umbenennen, indem Sie auf **File>Rename** klicken und dann den neuen Namen des Jupyter Notebooks eintragen, oder indem Sie oben auf den alten Namen dieses Notebooks klicken und dort den neuen Namen eintragen. Speichern Sie Ihr Notebook nach der Umbenennung (Shortkey **Strg+s** oder auf das Speichern-Symbol (Diskette) oben klicken).
3. **Klausur bearbeiten.** *Die Bearbeitungszeit beträgt 75 Minuten.* Achten Sie auf die Bearbeitungszeit und stellen Sie sich ggf. einen Countdown-Timer oder Wecker. **Planen Sie für die Klausurabgabe 5 Minuten Zeit ein. Die Klausurabgabe beginnt nach Ende der Bearbeitungszeit.** Sie finden direkt unter diesem Satz eine Anleitung (“Klausurabgabe. Das müssen Sie tun.”), wie Sie Ihre Klausur abgeben werden.

1.3 Klausurabgabe. Dies müssen Sie tun:

1. **Bearbeitungen abspeichern.** Das Wichtigste zuerst: Speichern Sie Ihre Bearbeitungen (Shortkey **Strg+s** oder auf das Speichern-Symbol (Diskette) klicken).
2. **Jupyter Notebook Datei abspeichern.** Speichern Sie Ihre Jupyter Notebook als Datei (im Folgenden “Klausur” genannt) lokal auf Ihrem Rechner ab. (**File > Download as > Notebook (.ipynb)**).

3. Sie haben eine Probeklausur bearbeitet. Daher geben Sie Ihre Klausur jetzt nicht ab. Die Probeklausur wird im Verlauf der Veranstaltung innerhalb Ihrer Teams besprochen. Jedes Team erstellt eine finale Klausurversion, die sie mitsamt der erarbeiteten Übungen wie üblich als Übungsabgabe einreicht. In der echten Klausur würden Sie Ihre Klausur direkt bei ILIAS einreichen.

1.4 Klausuraufgaben

1.4.1 Hinweise

- (1) **Überblick verschaffen.** Wie viele Punkte können Sie bei welchen Aufgaben erzielen?
- (2) **Sie müssen die Klausur persönlich und ohne fremde Hilfe bearbeiten.**
- (3) **Dies ist eine Kofferklausur.** Sie dürfen erlaubte Hilfsmittel nutzen.
- (4) Führen Sie die unten stehende Codezelle aus, um die Größe der von Ihnen zu erstellenden Abbildungen zu konfigurieren.

```
[1]: import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10, 7]
```

1.4.2 Aufgabe: Corona-Pandemie

Erreichbare Punktzahl: 40 Im Frühjahr 2020 begann sich das neuartiges Coronavirus SARS-CoV-2, welches die Lungenerkrankung Covid-19 verursachen kann, in Deutschland und Europa schlagartig zu verbreiten. In dieser Aufgabe analysieren und visualisieren Sie das Infektionsgeschehen in Deutschland.

Ihre Daten

Ein vereinfachter Datensatz des Robert Koch Instituts (RKI) steht Ihnen in dieser Aufgabe zur Verfügung.

Ihre Aufgaben

- (1) Führen Sie zunächst die unten stehende Code-Zelle aus, um den Datensatz und die Pandas-Bibliothek verfügbar zu machen.

```
[2]: import pandas as pd

# Daten des Robert Koch Instituts
RKI_data_simplified = 'RKI_data_simplified.csv'
```

- (2) Importieren Sie mithilfe von Pandas den Datensatz in einen DataFrame, den Sie df_covid nennen. Nutzen Sie dazu die Variable RKI_data_simplified. [/2] Punkte

```
[3]: # Ihr Code
df_covid: pd.DataFrame = pd.read_csv(RKI_data_simplified)
df_covid.head()
```

```
[3]: Meldedatum Deutschland Schleswig-Holstein Hamburg Niedersachsen \
0 2020-01-28 2 NaN NaN NaN
1 2020-01-29 2 NaN NaN NaN
2 2020-01-31 3 NaN NaN NaN
3 2020-02-03 1 NaN NaN NaN
4 2020-02-04 4 NaN NaN 1.0

Bremen Nordrhein-Westfalen Hessen Rheinland-Pfalz Baden-Württemberg \
0 NaN NaN NaN NaN NaN
1 NaN NaN NaN NaN NaN
2 NaN NaN NaN NaN NaN
3 NaN NaN NaN NaN NaN
4 NaN NaN NaN NaN NaN

Bayern Saarland Berlin Brandenburg Mecklenburg-Vorpommern Sachsen \
0 2.0 NaN NaN NaN NaN NaN
1 2.0 NaN NaN NaN NaN NaN
2 3.0 NaN NaN NaN NaN NaN
3 1.0 NaN NaN NaN NaN NaN
4 3.0 NaN NaN NaN NaN NaN

Sachsen-Anhalt Thüringen
0 NaN NaN
1 NaN NaN
2 NaN NaN
3 NaN NaN
4 NaN NaN
```

- (3) Der DataFrame, den Sie erhalten haben, enthält die Anzahl neu infizierter Personen für **Deutschland** sowie aufgelistet nach Bundesland, die zu einem Zeitpunkt (**Meldedatum**) gemeldet wurden. [/4] Punkte
- Wandeln Sie die Spalte **Meldedatum** in einen *datetime* Typen um. Sie können die Pandas Funktion **to_datetime** dafür nutzen. (2 Punkte)
 - Machen Sie anschließend **Meldedatum** zum Index Ihres DataFrames. Sie können die Pandas Funktion **set_index** dafür nutzen. (2 Punkte)

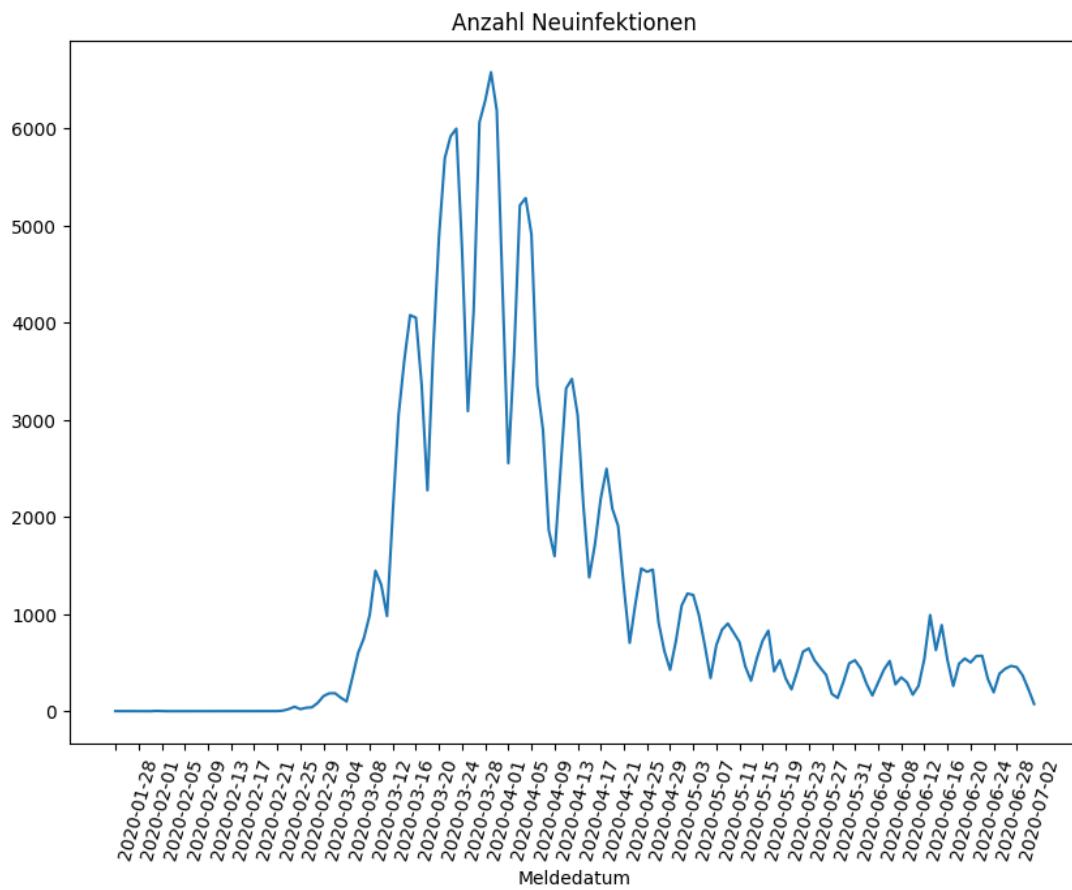
```
[4]: # Ihr Code
df_covid["Meldedatum"] = pd.to_datetime(df_covid["Meldedatum"])
df_covid.set_index("Meldedatum", inplace=True)
```

- (4) Visualisieren Sie die Anzahl neu infizierter Personen für Deutschland (Spalte: **Deutschland**) als Funktion des Meldedatums. Ihre Abbildung muss folgende Eigenschaften erfüllen: [/6] Punkte
- Die x-Achse ist mit “Meldedatum” beschriftet. (2 Punkte)
 - Auf der x-Achse stehen Datumsangaben (z.B: ‘2020-03-17’), keine bloßen Integers. (2 Punkte)
 - Der Titel der Abbildung lautet “Anzahl Neuinfektionen”. (2 Punkte)

```
[5]: # Ihr Code
plt.xlabel("Melde datum")
plt.title("Anzahl Neuinfektionen")
plt.plot(df_covid["Deutschland"])

xticks = pd.date_range(start=df_covid.index.min(), end=df_covid.index.max(), freq='4d')
plt.xticks(xticks, xticks.strftime("%Y-%m-%d"), rotation=75, ha="left")

plt.show()
```



- (5) Am 2. April 2020 begann die *Einführung in Data Science* Vorlesung in Aachen. Bestimmen Sie die Anzahl gemeldeter Neuinfizierter an diesem Tag und nennen Sie diese Anzahl. [/4] Punkte

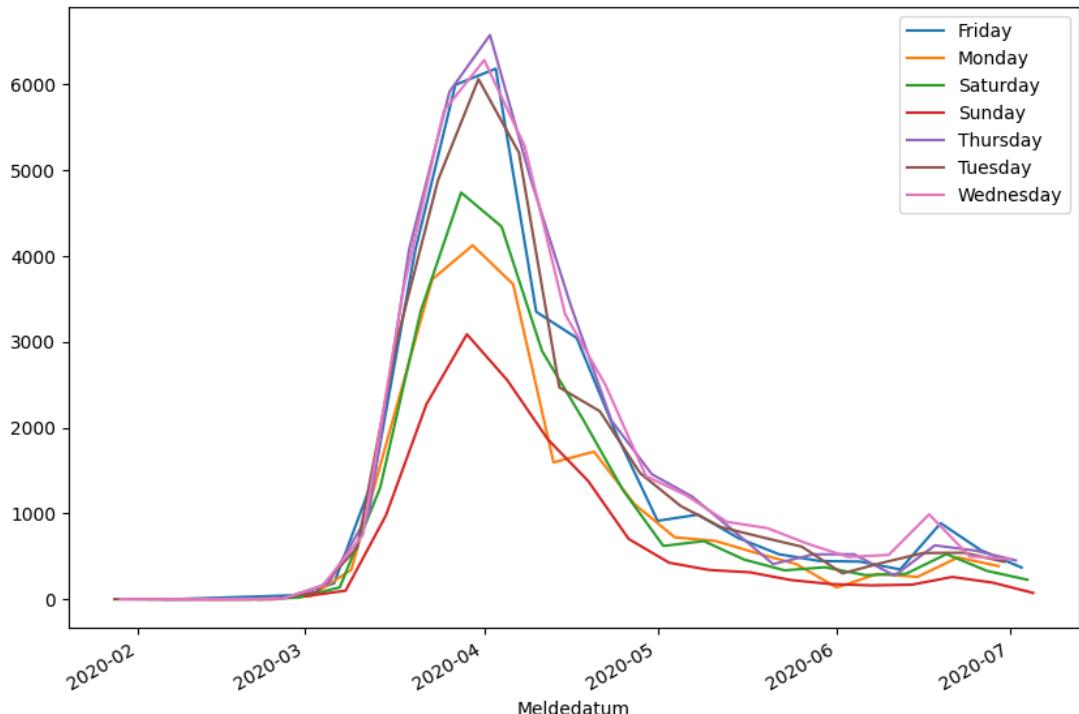
```
[6]: # Ihr Code
df_covid.loc["2020-04-02"]["Deutschland"]
```

```
[6]: np.float64(6574.0)
```

6574 gemeldete Neuinfektionen an dem Tag

- (6) Die Anzahl gemeldeter Neuinfizierter in Ihrer Visualisierung aus Schritt (4) zeigt periodische Schwingungen. Wie erklären Sie das Auftreten dieser Schwingungen? (Stichpunkte oder 1-3 Sätze) [/6] Punkte

```
[7]: df_covid["Weekday"] = df_covid.index.day_name()
df_covid.groupby("Weekday")["Deutschland"].plot()
plt.legend()
plt.show()
```



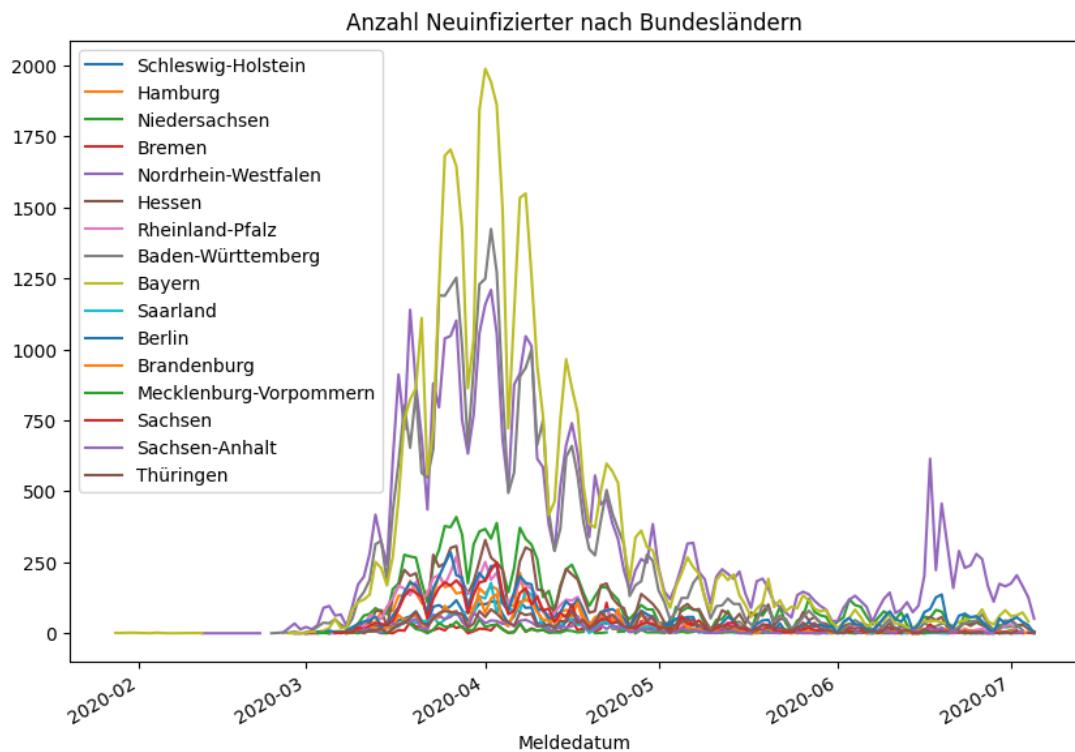
Die Schwankungen entstehen durch den Wochenverlauf. Sonntags werden nur sehr wenige Infektionen gemeldet, gefolgt von Montagen und Samstagen.

- (7) Die Anzahl Neuinfizierter steigt kurzzeitig im Juni 2020 etwas an und flacht dann wieder ab. Untersuchen Sie, auf welches Bundesland sich dieser Anstieg zurückführen lässt: [/8] Punkte
1. Visualisieren Sie die Anzahl gemeldeter Neuinfizierter als Funktion des Meldedatums *für alle Bundesländer* in derselben Abbildung. Ihre Abbildung erfüllt die folgenden Eigenschaften:
 - Die x-Achse ist mit "Meldedatum" beschriftet. (1 Punkt)
 - Auf der x-Achse stehen Datumsangaben (z.B: '2020-03-17'), keine bloßen Integers. (1 Punkt)
 - Der Titel der Abbildung lautet "Anzahl Neuinfizierter nach Bundesländern". (1 Punkt)
 - Eine Legende erlaubt eine farbkodierte Unterscheidung der dargestellten Graphen nach Bundesländern. (1 Punkt)

- Ihre Abbildung enthält **nicht** die Gesamtzahl aller Infizierter Deutschlands.

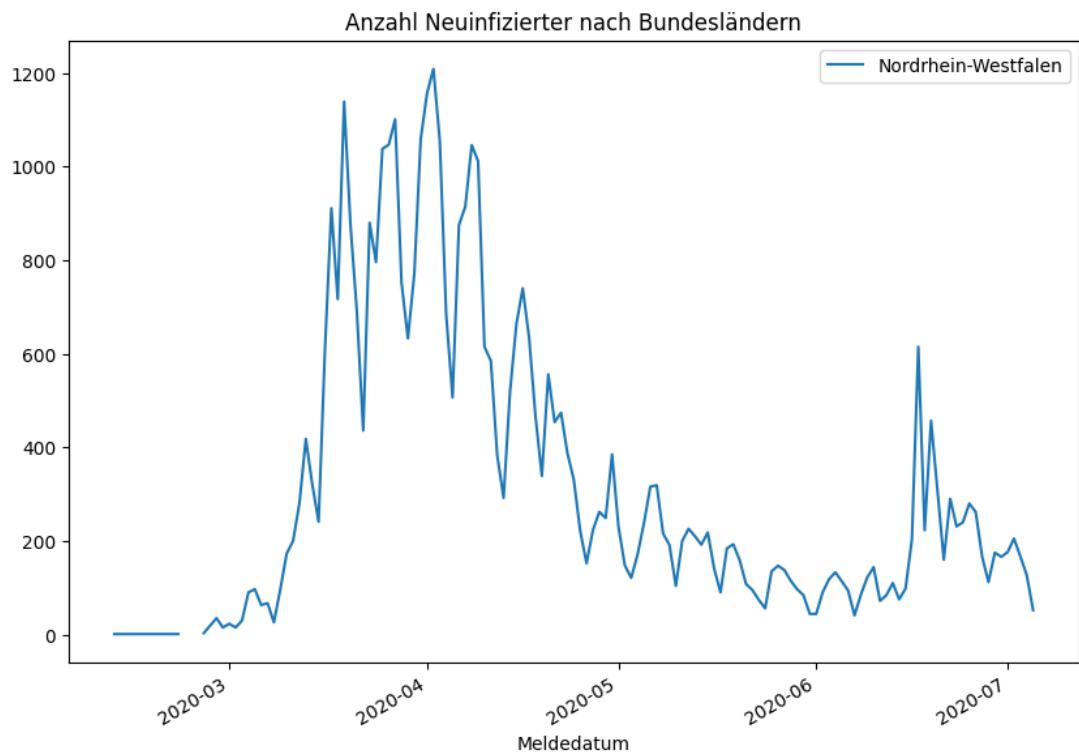
[8]: # Ihr Code

```
df_covid.iloc[:, 1:].plot()
plt.xlabel("Meldedatum")
plt.title("Anzahl Neuinfizierter nach Bundesländern")
plt.legend()
plt.show()
```



[9]: df_covid.loc[:, "Nordrhein-Westfalen"].plot()

```
plt.xlabel("Meldedatum")
plt.title("Anzahl Neuinfizierter nach Bundesländern")
plt.legend()
plt.show()
```



2. Identifizieren Sie über Ihre Abbildung das Bundesland, auf das sich der kurzzeitige Anstieg der Neuinfizierten im Juni 2020 zurückführen lässt, und nennen Sie dieses Bundesland. (2 Punkte)

- Bonus: Benennen Sie die Ursache dieser erhöhten Infektionszahlen. (+2 Punkte)

Nordrhein-Westfalen. In dem Kreis Gütersloh kam es im Juni 2020 zu dem bisher „größten Infektionsgeschehen“ in Deutschland. Corona-Ausbruch bei Tönnies – Lockdown in Gütersloh. In: tagesschau.de. ARD, 23. Juni 2020, abgerufen am 23. Juni 2020.

3. In welchem Bundesland Deutschlands wurden als erstes Corona-Infizierte gemeldet? (2 Punkte)

[10]: # Ihr Code

```
df_covid.sort_index()
df_covid
```

[10]:

Meldedatum	Deutschland	Schleswig-Holstein	Hamburg	Niedersachsen	Bremen	\
2020-01-28	2		NaN	NaN	NaN	NaN
2020-01-29	2		NaN	NaN	NaN	NaN
2020-01-31	3		NaN	NaN	NaN	NaN
2020-02-03	1		NaN	NaN	NaN	NaN
2020-02-04	4		NaN	NaN	1.0	NaN

...
2020-07-01	467		12.0	9.0	25.0	4.0
2020-07-02	454		5.0	2.0	22.0	4.0
2020-07-03	371		2.0	2.0	30.0	NaN
2020-07-04	229		2.0	NaN	NaN	3.0
2020-07-05	75		6.0	NaN	NaN	1.0

Meldedatum	Nordrhein-Westfalen	Hessen	Rheinland-Pfalz	Baden-Württemberg	\
2020-01-28	NaN	NaN	NaN	NaN	
2020-01-29	NaN	NaN	NaN	NaN	
2020-01-31	NaN	NaN	NaN	NaN	
2020-02-03	NaN	NaN	NaN	NaN	
2020-02-04	NaN	NaN	NaN	NaN	
...	
2020-07-01	176.0	49.0	30.0	42.0	
2020-07-02	205.0	29.0	8.0	30.0	
2020-07-03	167.0	31.0	6.0	12.0	
2020-07-04	127.0	16.0	2.0	NaN	
2020-07-05	52.0	7.0	NaN	NaN	

Meldedatum	Bayern	Saarland	Berlin	Brandenburg	Mecklenburg-Vorpommern	\
2020-01-28	2.0	NaN	NaN	NaN	NaN	
2020-01-29	2.0	NaN	NaN	NaN	NaN	
2020-01-31	3.0	NaN	NaN	NaN	NaN	
2020-02-03	1.0	NaN	NaN	NaN	NaN	
2020-02-04	3.0	NaN	NaN	NaN	NaN	
...	
2020-07-01	61.0	1.0	47.0	8.0	NaN	
2020-07-02	64.0	1.0	57.0	8.0	1.0	
2020-07-03	74.0	NaN	41.0	2.0	NaN	
2020-07-04	43.0	1.0	29.0	3.0	NaN	
2020-07-05	NaN	NaN	1.0	NaN	NaN	

Meldedatum	Sachsen	Sachsen-Anhalt	Thüringen	Weekday
2020-01-28	NaN	NaN	NaN	Tuesday
2020-01-29	NaN	NaN	NaN	Wednesday
2020-01-31	NaN	NaN	NaN	Friday
2020-02-03	NaN	NaN	NaN	Monday
2020-02-04	NaN	NaN	NaN	Tuesday
...
2020-07-01	1.0	1.0	1.0	Wednesday
2020-07-02	4.0	4.0	10.0	Thursday
2020-07-03	1.0	3.0	NaN	Friday
2020-07-04	NaN	NaN	3.0	Saturday

2020-07-05 NaN 6.0 2.0 Sunday

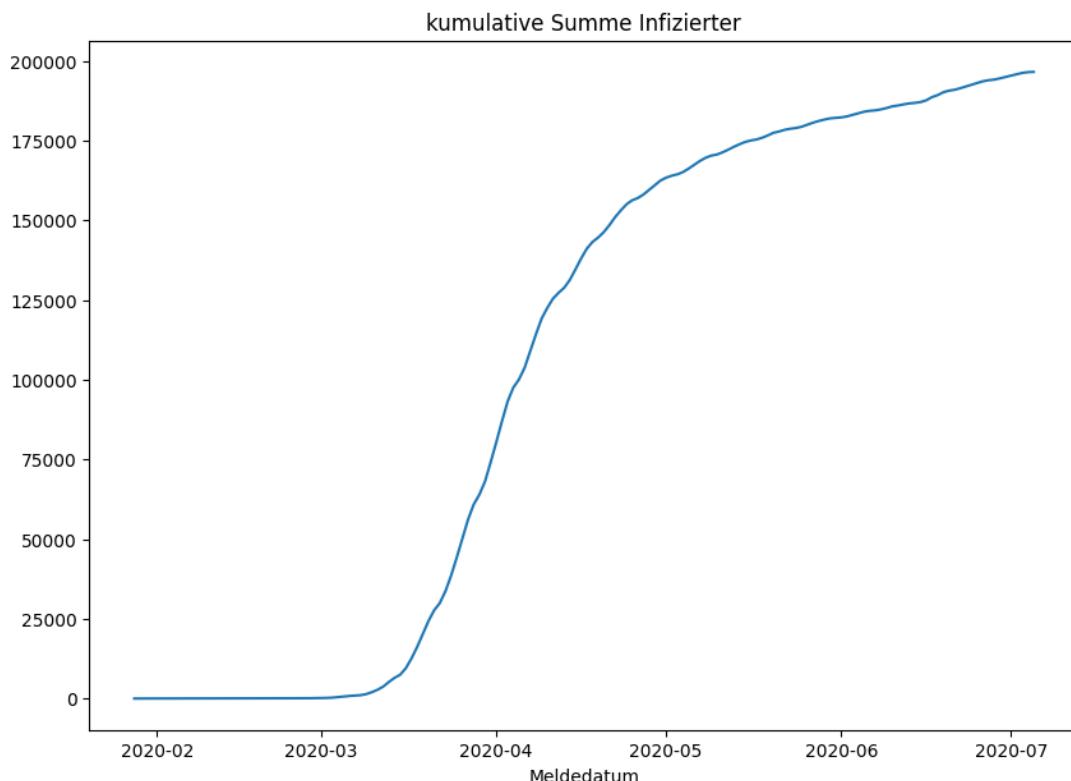
[144 rows x 18 columns]

Bayern

- (8) Visualisieren Sie die kumulative Summe der Neuinfizierten in Deutschland (Spalte: Deutschland) als Funktion des Meldedatums. [/4] Punkte
- Die x-Achse ist mit "Meldedatum" beschriftet. (1 Punkte)
 - Auf der x-Achse stehen Datumsangaben (z.B: '2020-03-17'), keine bloßen Integers. (1 Punkte)
 - Der Titel der Abbildung lautet "kumulative Summe Infizierter". (1 Punkte)

[11]: # Ihr Code

```
plt.xlabel("Meldedatum")
plt.title("kumulative Summe Infizierter")
plt.plot(df_covid["Deutschland"].cumsum())
plt.show()
```



- (9) Bestimmen Sie die Zahl aller Menschen, die in Deutschland bis zum 5. Juli 2020 als mit Covid-19 infiziert gemeldet wurden. [/6] Punkte

```
[12]: # Ihr Code  
df_covid["Deutschland"].cumsum().loc["2020-07-05"]
```

```
[12]: np.int64(196550)
```

196.550 Personen wurden bis zum 5. Juli 2020 infiziert.

- (10) *Bonus:* (Bearbeiten Sie diese Teilaufgabe am besten erst, wenn Sie bereits die anderen Aufgaben dieser Klausur bearbeitet haben.)

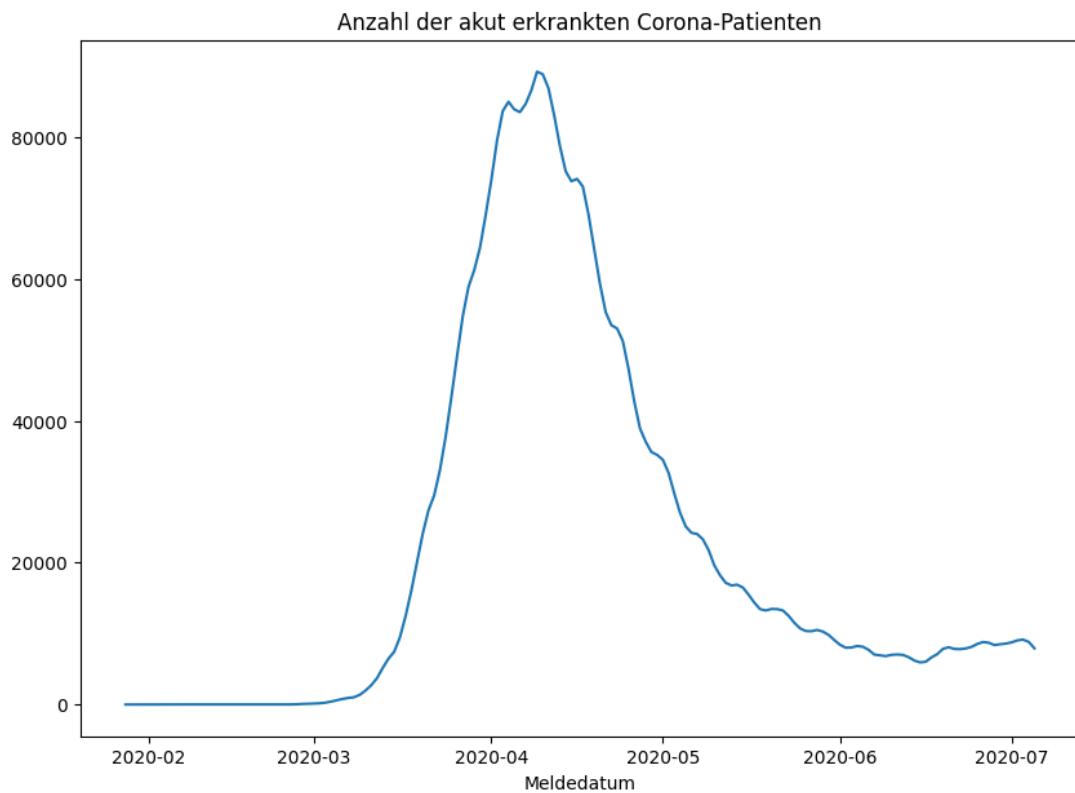
Visualisieren Sie die geschätzte Anzahl der akut erkrankten Coronapatienten ganz Deutschlands als Funktion des Meldedatums. [/8] Punkte

Für die Schätzung der akut erkrankten Patienten (5 Punkte) nehmen wir Folgendes an:

- Wir nehmen an, dass die Infektion 18 Tage nach dem Meldedatum vorbei ist und der Patient gesund geworden ist.
- Wir nehmen an, dass keine Menschen durch eine Corona-Infektion versterben.

Ihre Visualisierung erfüllt die folgenden Eigenschaften: - Die x-Achse ist mit "Meldedatum" beschriftet. (1 Punkt) - Auf der x-Achse stehen Datumsangaben (z.B: '2020-03-17'), keine bloßen Integers. (1 Punkt) - Der Titel der Abbildung lautet "Anzahl der akut erkrankten Corona-Patienten". (1 Punkt)

```
[13]: # Ihr Code  
plt.xlabel("Meldedatum")  
plt.title("Anzahl der akut erkrankten Corona-Patienten")  
plt.plot(df_covid.rolling('18D')["Deutschland"].sum())  
plt.show()
```



1.4.3 Aufgabe: Reproduktionszahl

Erreichbare Punktzahl: 28 Sie bestimmen in dieser Aufgabe die Reproduktionszahl R für die Covid-19 Pandemie in Deutschland anhand der Daten des Robert-Koch-Instituts. Die Reproduktionszahl R gibt an, wie viele Menschen von einer infektiösen Person durchschnittlich angesteckt werden, wenn kein Mitglied der Population gegenüber dem Erreger immun ist.

Sie werden eine vereinfachte Schätzung der R -Zahl durchführen. Die R -Zahl eines Tages t sei definiert als

$$R(t) := \frac{\sum_{x=t-3}^t I_x}{\sum_{x=t-7}^{t-4} I_x},$$

wobei I_x die Anzahl der Neuinfizierten eines Tages x bezeichnet.

Die R -Zahl entspricht also der Summe der Infizierten innerhalb eines 4-Tageszeitraums dividiert durch die Summe der Infizierten des davorliegenden 4-Tageszeitraums.

Ihre Daten

Ein vereinfachter Datensatz des Robert Koch Instituts (RKI) steht Ihnen in dieser Aufgabe zur Verfügung.

Ihre Aufgaben

- (1) Führen Sie zunächst die unten stehende Code-Zelle aus, um den Datensatz und die Pandas-Bibliothek verfügbar zu machen.

```
[14]: import pandas as pd
```

```
# Daten des Robert Koch Instituts
RKI_data_simplified = 'RKI_data_simplified.csv'
```

- (2) Importieren Sie mithilfe der Variablen `RKI_data_simplified` den Datensatz in einen Pandas DataFrame, den Sie `df_covid` nennen. [/4] Punkte
- Wandeln Sie die Spalte `Meldedatum` in einen *datetime* Typen um. Sie können die Pandas Funktion `to_datetime` dafür nutzen. (2 Punkte)
 - Machen Sie anschließend `Meldedatum` zum Index Ihres DataFrames. Sie können die Pandas Funktion `set_index` dafür nutzen. (2 Punkte)

```
[15]: # Ihr Code
```

```
df_covid: pd.DataFrame = pd.read_csv(RKI_data_simplified)
df_covid["Meldedatum"] = pd.to_datetime(df_covid["Meldedatum"])
df_covid.set_index("Meldedatum", inplace=True)
```

- (3) Bestimmen Sie nun die $R(t)$ -Zahl für Deutschland (Spalte: Deutschland) aus den Daten, die Sie im vorherigen Schritt verfügbar gemacht haben. [/12] Punkte

Gehen Sie dabei am besten in zwei Schritten vor:

1. Bestimmen Sie die rollierenden 4-Tagessummen der Infiziertenzahlen. Sie erhalten dadurch einen neuen DataFrame, der eine Zeitreihe enthält, deren Einträge jeweils einer Summe über die Anzahl der Infizierten über 4 aufeinanderfolgende Tage darstellt. Nutzen Sie für diesen Schritt am besten die Funktionalität von Pandas. (6 Punkte)
 - **Wichtig:** Die Summen in der Definition von $R(t)$ (siehe obige Gleichung) verlaufen jeweils über 4 direkt aufeinanderfolgenden Tagen.
2. Bestimmen Sie $R(t)$ gemäß der Definition zu Beginn dieser Aufgabe. Nutzen Sie aus, dass Sie die Zeitreihe (der rollierenden 4-Tagessummen) um 4 Tage mithilfe von Pandas verschieben können. (6 Punkte)

Wir führen einen Shift $r \rightarrow r_{\text{shft}}$ mit `r_shft = r.shift(4, axis=0)` durch, da

$$R(5) = \frac{r(5)}{r_{\text{shft}}(5)} = \frac{r(5)}{r(1)}$$

sein soll. Aus dieser Betrachtung geht hervor, warum der frühere Zeitraum (Nenner) verschoben werden muss und nicht der spätere.

```
[16]: df_covid['date'] = df_covid.index
```

```
r1: pd.DataFrame = df_covid.rolling('4D', on='date')[["Deutschland"]].sum()
```

```
# Nenner in die Zukunft shiftet bedeutet, dass bei index=5 r(5) = r1(5) / r1_shifted(5) und da r1_shifted(5) = r1(1) ist, passt das
r_t: pd.DataFrame = r1 / r1.shift(4, axis=0)
r_t.head(10)
```

[16]: Meldedatum

Meldedatum	R(t)
2020-01-28	NaN
2020-01-29	NaN
2020-01-31	NaN
2020-02-03	NaN
2020-02-04	2.500000
2020-02-06	1.500000
2020-02-07	0.857143
2020-02-11	0.500000
2020-02-12	0.800000
2020-02-20	0.166667

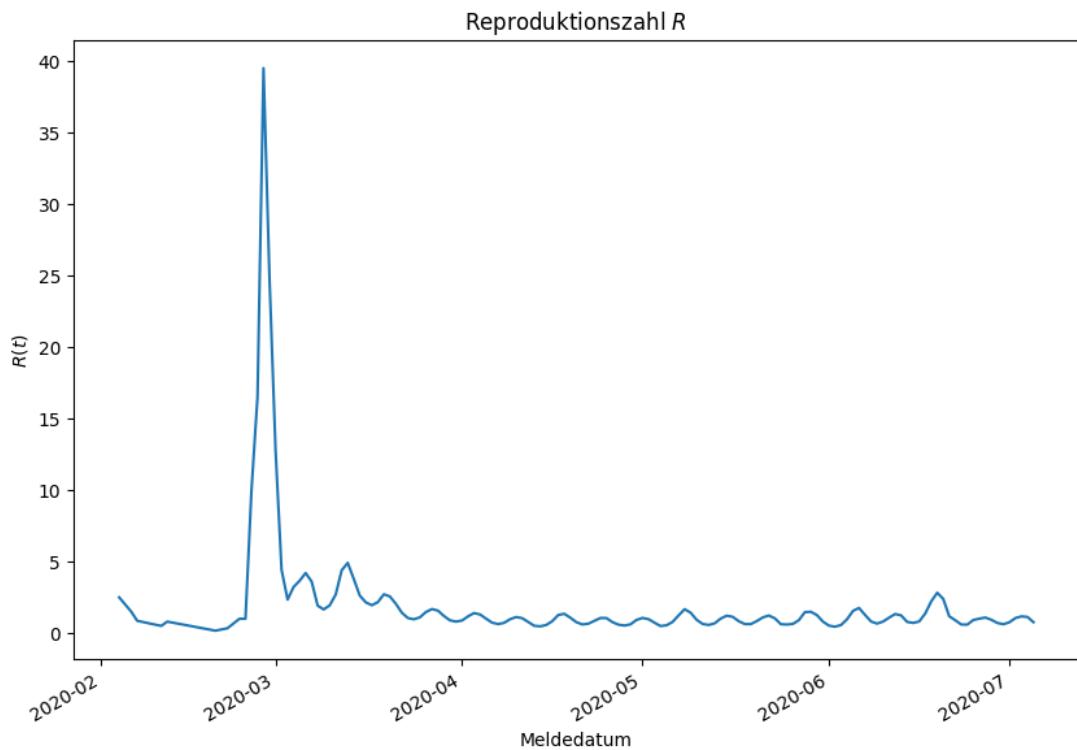
Name: Deutschland, dtype: float64

(4) Visualisieren Sie die $R(t)$ -Zahl als Funktion des Meldedatums t . [/4] Punkte

- Sie haben $R(t)$ als Funktion des Meldedatums in einer Abbildung visualisiert. (1 Punkt)
- Die x-Achse ist mit “Meldedatum” beschriftet. (1 Punkt)
- Auf der x-Achse stehen Datumsangaben (z.B: ‘2020-03-17’ oder ‘Apr 2020’), keine bloßen Integers. (1 Punkt)
- Der Titel der Abbildung lautet “Reproduktionszahl R”. (1 Punkt)

[17]: # Ihr Code

```
r_t.plot()
plt.xlabel("Meldedatum")
plt.ylabel("$R(t)$")
plt.title("Reproduktionszahl $R$")
plt.show()
```



- (5) An welchen Tagen im Juni 2020 können Sie Reproduktionszahlen beobachten, die größer sind als 2? [/8] Punkte

- Angabe der Tage im Juni 2020, deren Reproduktionszahlen größer sind als 2. (6 Punkte)
- Welcher Ausbruch steht mit den erhöhten Reproduktionszahlen im Juni im Zusammenhang stehen? (1-2 Sätze oder Stichworte) (2 Punkte)

```
[18]: # Ihr Code
r_t_juni = r_t.loc["2020-06-01":"2020-06-30"]
r_t_juni[r_t_juni > 2]
```

```
[18]: Meldedatum
2020-06-18    2.210959
2020-06-19    2.820370
2020-06-20    2.387097
Name: Deutschland, dtype: float64
```

18., 19. und am 20. Juni 2020

Outbreaks of COVID-19 have been reported in several federal states (including in institutions for asylum seekers and refugees, in connection with a religious events or in meat processing plants).

1.4.4 Aufgabe: Corona-Warn-App Analyse

Erreichbare Punktzahl: 20 Die Corona-Warn-App wurde im Juni 2020 in Deutschland veröffentlicht, um die Kontaktnachverfolgung infizierter Personen zu erleichtern. Die App wurde bis Anfang Juli 2020 bereits über 14 Millionen Mal heruntergeladen.

Über die App-Infrastruktur werden jeden Tag Menschen vor potentiellen Infektionen gewarnt. Aus den dabei benötigten täglichen Daten lassen sich die Anzahl der Menschen ermitteln, die pro Tag über die Warn-App als infiziert gemeldet wurden.

Sie werden in dieser Aufgabe untersuchen, wieviel Prozent aller neuinfizierten Fälle (laut Robert-Koch-Institut) bereits über die Corona-Warn-App gemeldet werden.

Ihre Daten

- Ein vereinfachter Datensatz des Robert Koch Instituts (RKI) steht Ihnen in dieser Aufgabe zur Verfügung.
- Corona-Warn-App Datensatz, der die Schätzung der Anzahl der als infiziert gemeldeten Menschen enthält

Ihre Aufgaben

- (1) Führen Sie zunächst die unten stehende Code-Zelle aus, um die Datensätze und die Pandas-Bibliothek verfügbar zu machen.

```
[19]: import pandas as pd

# Daten des Robert Koch Instituts
RKI_data_simplified = 'RKI_data_simplified.csv'

# Daten der Corona Warn App Infrastruktur
cwa_data = 'cwa_data.csv'
```

- (2) Importieren Sie mithilfe der Variablen `RKI_data_simplified` und `cwa_data` die Datensätze in Pandas DataFrames, die Sie `df_covid` bzw. `df_cwa` nennen. Für beide DataFrames machen Sie Folgendes: [/4] Punkte

- Wandeln Sie die Spalte `Meldedatum` in einen `datetime` Typen um. Sie können die Pandas Funktion `to_datetime` dafür nutzen. (2 Punkte)
- Machen Sie anschließend `Meldedatum` zum Index Ihres jeweiligen DataFrames. Sie können die Pandas Funktion `set_index` dafür nutzen. (2 Punkte)

```
[20]: # Ihr Code
df_covid= pd.DataFrame = pd.read_csv(RKI_data_simplified)
df_cwa= pd.DataFrame = pd.read_csv(cwa_data)

df_covid["Meldedatum"] = pd.to_datetime(df_covid["Meldedatum"])
df_covid.set_index("Meldedatum", inplace=True)

df_cwa["Meldedatum"] = pd.to_datetime(df_cwa["Meldedatum"])
df_cwa.set_index("Meldedatum", inplace=True)
```

```
df_cwa.head()
```

[20]:

	AnzahlFall
Meldedatum	
2020-06-22	0
2020-06-23	27
2020-06-24	21
2020-06-25	19
2020-06-26	23

- (3) Bestimmen Sie tagesgenau den prozentualen Anteil der Neuinfizierten, die durch die Corona-Warn-App gemeldet wurden, an der gesamten Anzahl der Neuinfizierten (Spalte "Deutschland"), die durch das Robert-Koch-Institut gemeldet wurden. [/6] Punkte

[21]:

```
# Ihr Code
df = pd.DataFrame = df_covid.join(df_cwa, on="Meldedatum")
prozentualAnteil = df["AnzahlFall"] / df["Deutschland"]
prozentualAnteil.dropna()
```

[21]:

Meldedatum	
2020-06-22	0.000000
2020-06-23	0.049724
2020-06-24	0.041833
2020-06-25	0.033451
2020-06-26	0.040280
2020-06-27	0.120482
2020-06-28	0.071795
2020-06-29	0.038760
2020-06-30	0.059361
2020-07-01	0.057816
2020-07-02	0.072687
2020-07-03	0.043127
2020-07-04	0.048035

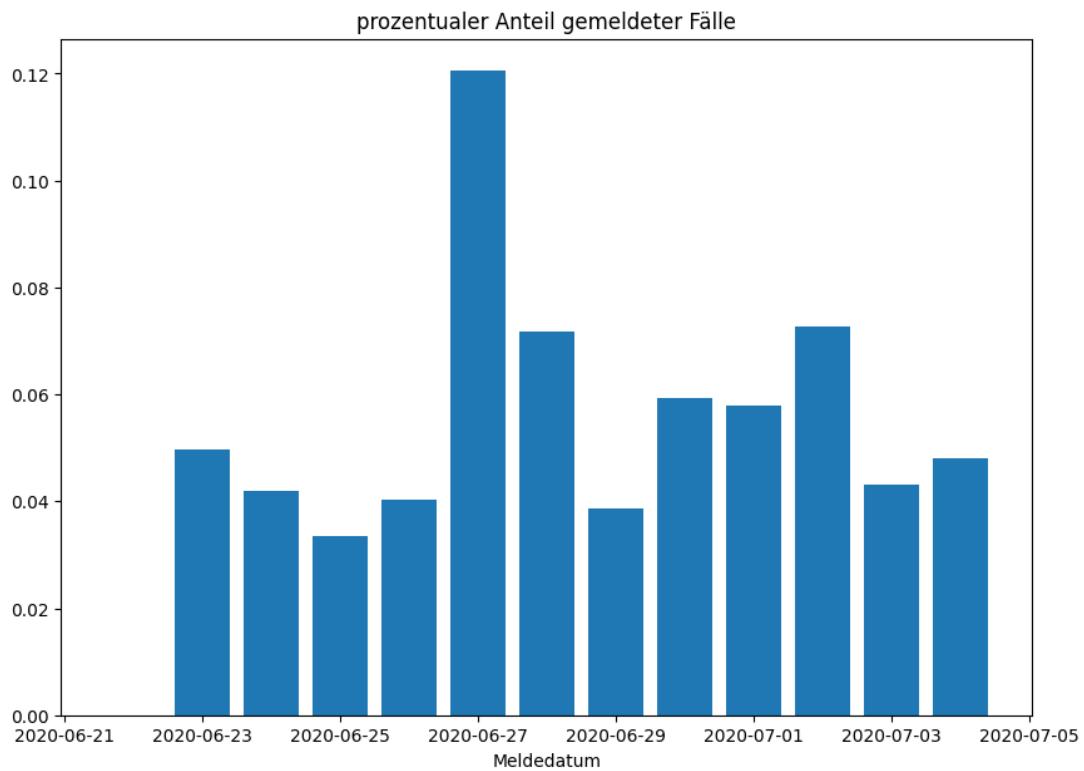
dtype: float64

- (4) Visualisieren Sie in einem Bar-Plot den prozentualen Anteil, den Sie in Schritt (3) ermittelt haben, als Funktion des Meldedatums. [/5] Punkte

- Die x-Achse ist mit "Meldedatum" beschriftet. (1 Punkte)
- Auf der x-Achse stehen Datumsangaben (z.B: '2020-03-17' oder 'Apr 2020'), keine bloßen Integers. (2 Punkte)
- Der Titel der Abbildung lautet "prozentualer Anteil gemeldeter Fälle". (1 Punkte)

[22]:

```
# Ihr Code
plt.bar(prozentualAnteil.index, prozentualAnteil)
plt.xlabel("Meldedatum")
plt.title("prozentualer Anteil gemeldeter Fälle")
plt.show()
```



- (5) Wieviel Prozent aller Neuinfizierten vom 2. Juli wurden durch die Corona-Warn-App gemeldet? Ermitteln Sie diese Prozentzahl und nennen Sie sie. [/5] Punkte

[23]: `# Ihr Code
prozentualAnteil.loc["2020-07-02"]`

[23]: `np.float64(0.07268722466960352)`

7,27 %

1.4.5 Aufgabe: Werbeindustrie

Erreichbare Punktzahl: 12 Der folgende Datenfall ist echt und wurde für die Klausurstellung anonymisiert:

Ein Data Scientist einer Firma, die im Bereich Onlinewerbung tätig ist, fragt um Rat. Die Firma möchte anhand von Beobachtungen verschiedene Personengruppen identifizieren. Es liegen Daten von $N = 700000$ Personen mit je $D = 100000$ Merkmalen (Features) vor.

Der Data Scientist hat eine explorative Datenanalyse (EDA) wie folgt vorgenommen:

1. Die Dimension des Feature-Raums wurde mithilfe einer PCA (Principal Component Analysis) von 100000 auf $D' = 3500$ Dimensionen reduziert.

2. Um eine Intuition über die in diesem dimensionsreduzierten Feature-Raum etwaig vorhandenen Cluster zu erhalten, wurden mit einem nichtlinearen Verfahren des Multidimensionalen Skalierens (MDS) die Daten in zwei Dimensionen dargestellt. Folgende Abbildung ergab sich:

Arbeiten Sie unter folgenden Annahmen:

- Das MDS-Verfahren funktioniert wahrheitsgetreu.
- Die PCA wurde korrekt implementiert.

Ihre Aufgaben

- (1) Untersuchen Sie die oben dargestellte Analysekette des Data Scientist: Welche Aspekte müssen Sie hinterfragen bzw. kritisieren? Notieren Sie stichwortartig Ihre Fragen bzw. Kritikpunkte. (Bei zwei Kritikpunkten bzw. Fragen sind Sie schon gut unterwegs). [/6] Punkte
 - Wie wurde die Anzahl der Komponenten gewählt, auf die der Featureraum über die PCA reduziert wurde? Wurde hier ein Ellbogenverlauf in der Proportion of Variance Explained (PVE) zu Rate gezogen?
 - Warum meint der Data Scientist, dass sich der Raum durch eine lineare Transformation dimensionsreduzieren lässt? Wäre es nicht angebracht, auch nach nichtlinearen Strukturen zu suchen?
- (2) Der Data Scientist hat die in der obigen Abbildung erkennbaren Clusterstrukturen untersucht. Der rot eingekreiste Bereich markiert einen Cluster, der tatsächlich einer Personengruppe entspricht, wie er herausgefunden hat. Solche Cluster möchte er über ein geeignetes Clusterverfahren aus dem Feature-Raum extrahieren. [/6] Punkte
 - Nennen Sie genau ein Clusterverfahren aus der Vorlesung *Data Science*, welches Sie ihm vorschlagen. (4 Punkte)

K-Means

- Begründen Sie Ihre Entscheidung für das von Ihnen vorgeschlagene Clusterverfahren und grenzen Sie es zu alternativen Verfahren ab. (1-5 Sätze) (2 Punkte)

Ich schlage K-Means vor, da hier nach der Vorverarbeitung ein sphärische Objekte im Feature-Space entstanden ist, welches über die euklidische Distanz zum Clusterschwerpunkt gut identifizierbar sein sollte. Ein hierarchisches Clustern könnte dann in Frage kommen, wenn wir hierarchisch organisierte Daten vorliegen haben. Dies könnte nach Art der Daten der Fall sein und das auftreten zusammenhängender Stränge würde auch dafür sprechen, jedoch ist der markierte Bereich spärlich.