

TSRI-11

July 3, 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*-Merkmal 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
```

```
[2]:
```

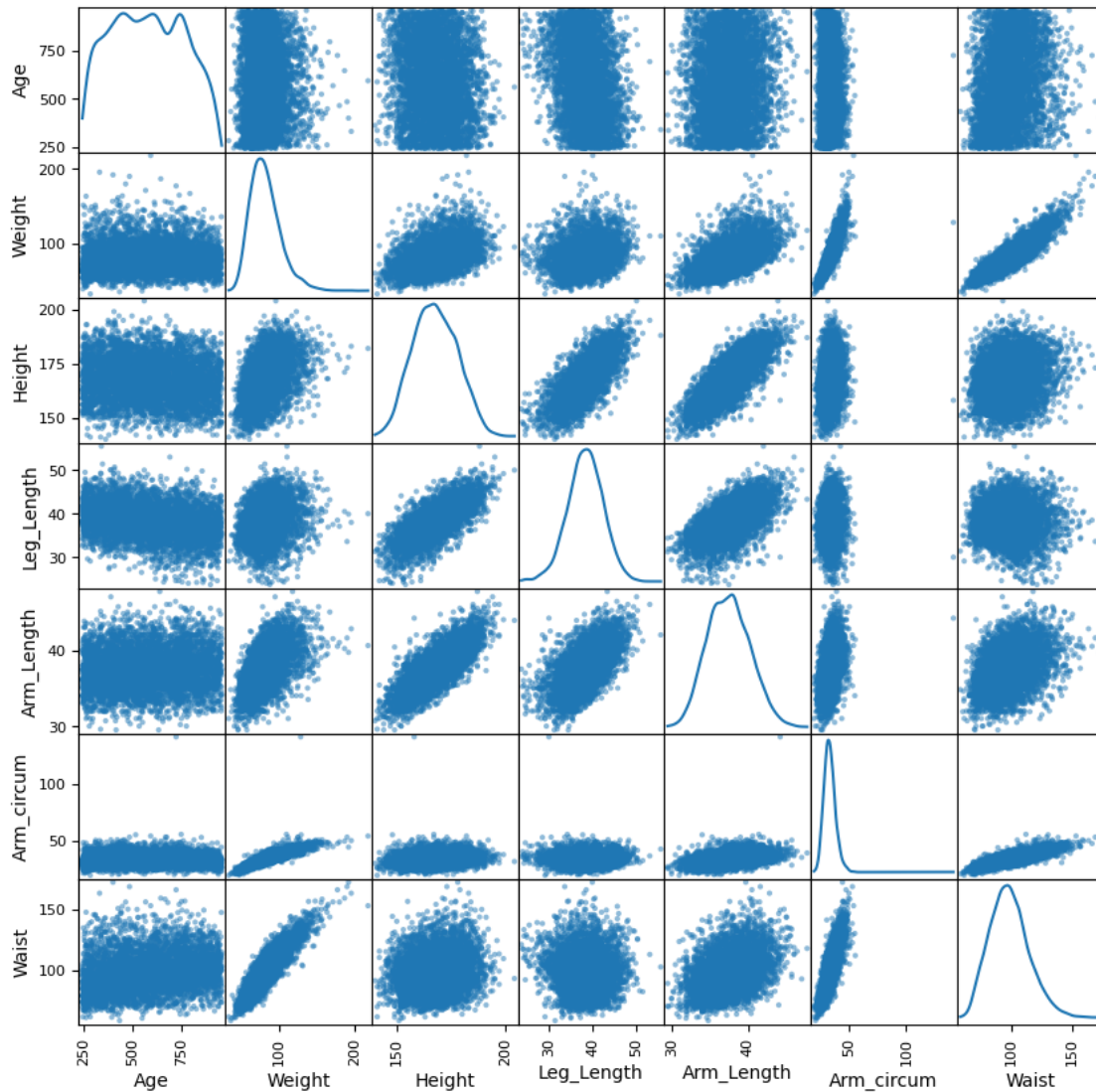
	Gender	Age	Weight	Height	Leg_Length	Arm_Length	Arm_circum	Waist
0	0	241	64.7	163	34.2	36.2	29.0	89.6
1	0	241	54.0	153	37.2	34.0	26.1	85.5
2	1	241	61.4	165	37.7	35.0	31.4	70.1
3	0	241	74.0	171	37.9	36.2	29.8	91.1
4	0	241	63.6	159	38.1	34.0	29.2	74.3
...
4974	1	958	89.5	184	43.0	42.8	32.2	112.8
4975	0	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.

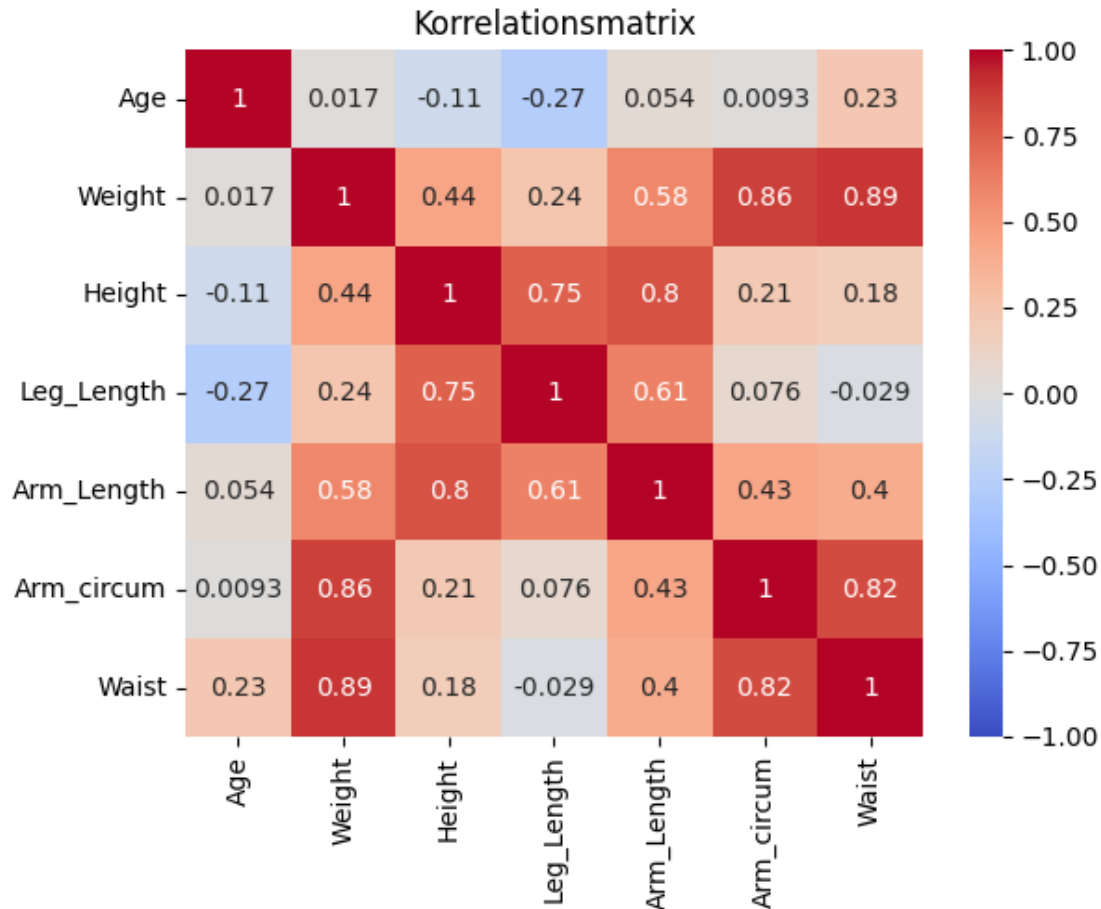
```
[3]: pd.plotting.scatter_matrix(df.loc[:, "Age:"], figsize=(10, 10), diagonal="kde")
plt.show()
```



```
[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

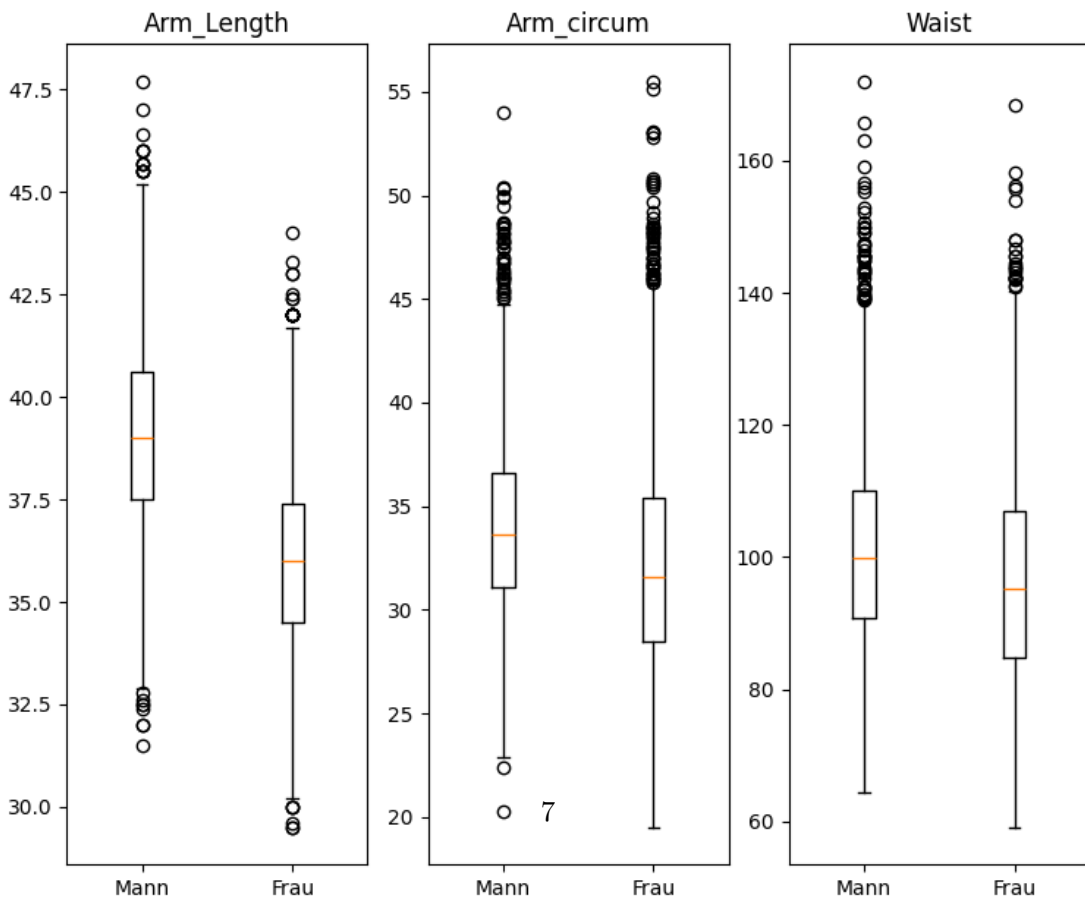
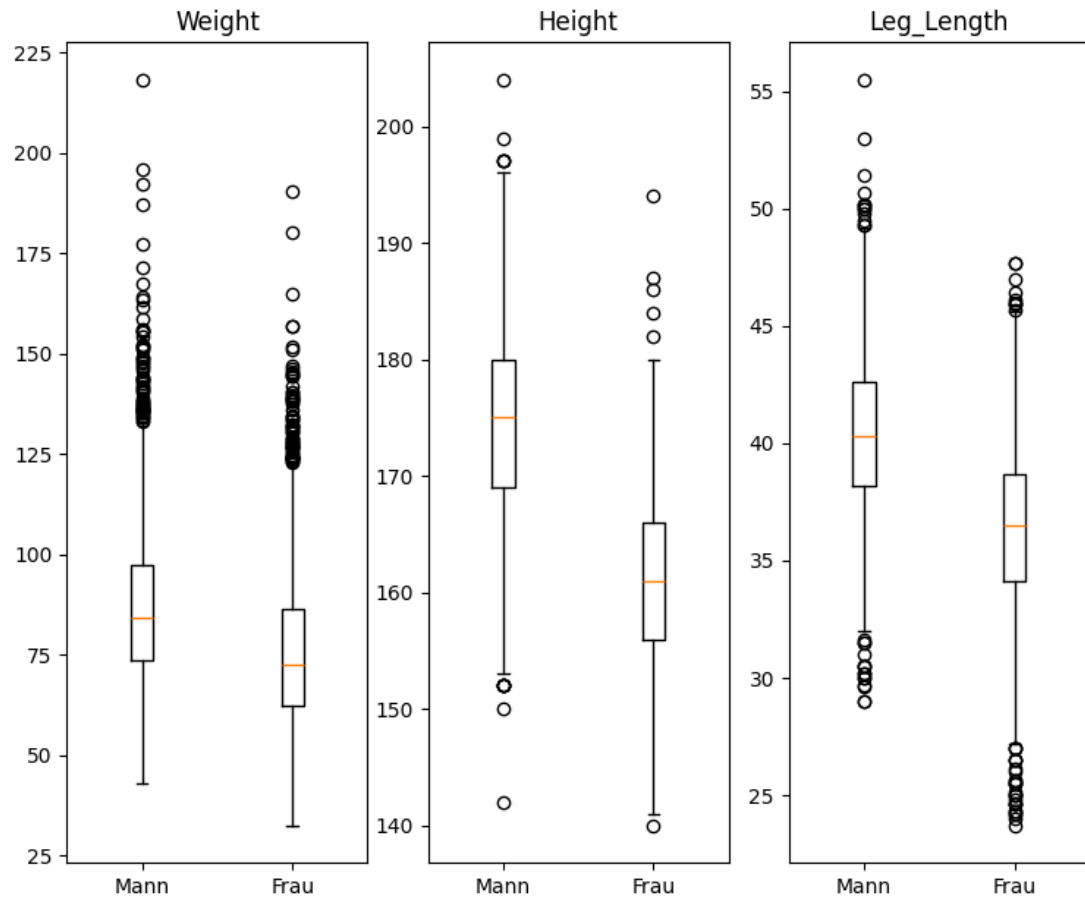
	Arm_Length	Arm_circum	Waist
count	2452.000000	2452.000000	2452.000000

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.

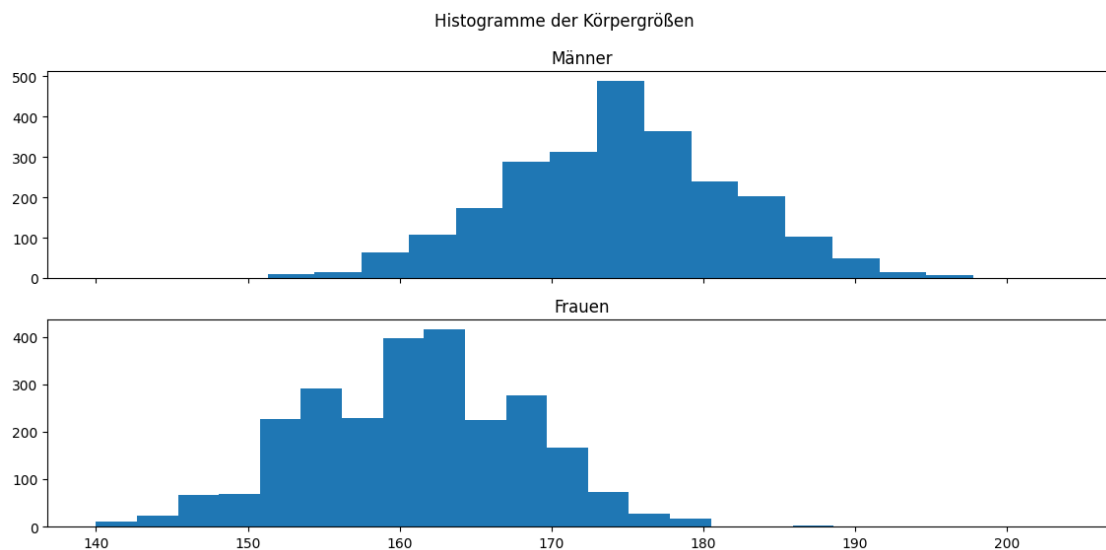
Im nachfolgenden Schritt gebe ich Ihnen ein Merkmal vor, welches Sie sich im Folgenden anschauen werden. Ihre Arbeit in Schritt (4) war aber nicht vergebens. Wir kommen darauf später noch zurück.

- (5) Erzeugen Sie ein **Histogramm**, welches die Verteilung der Körpergrößen von Männern und von Frauen darstellt.

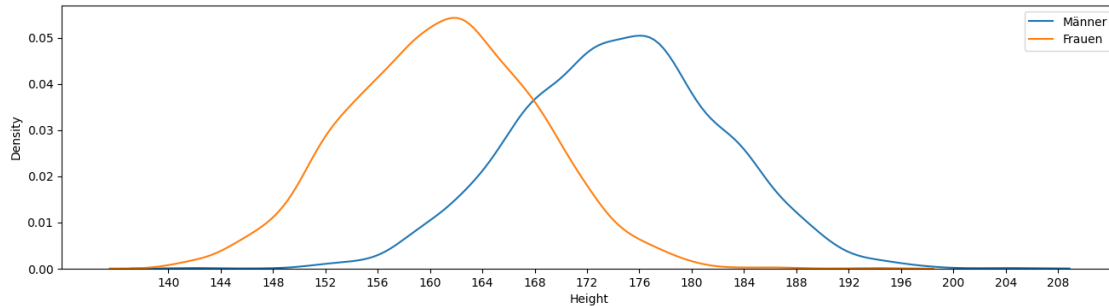
```
[9]: fig, axs = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

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, 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, 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 # Jouden 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.

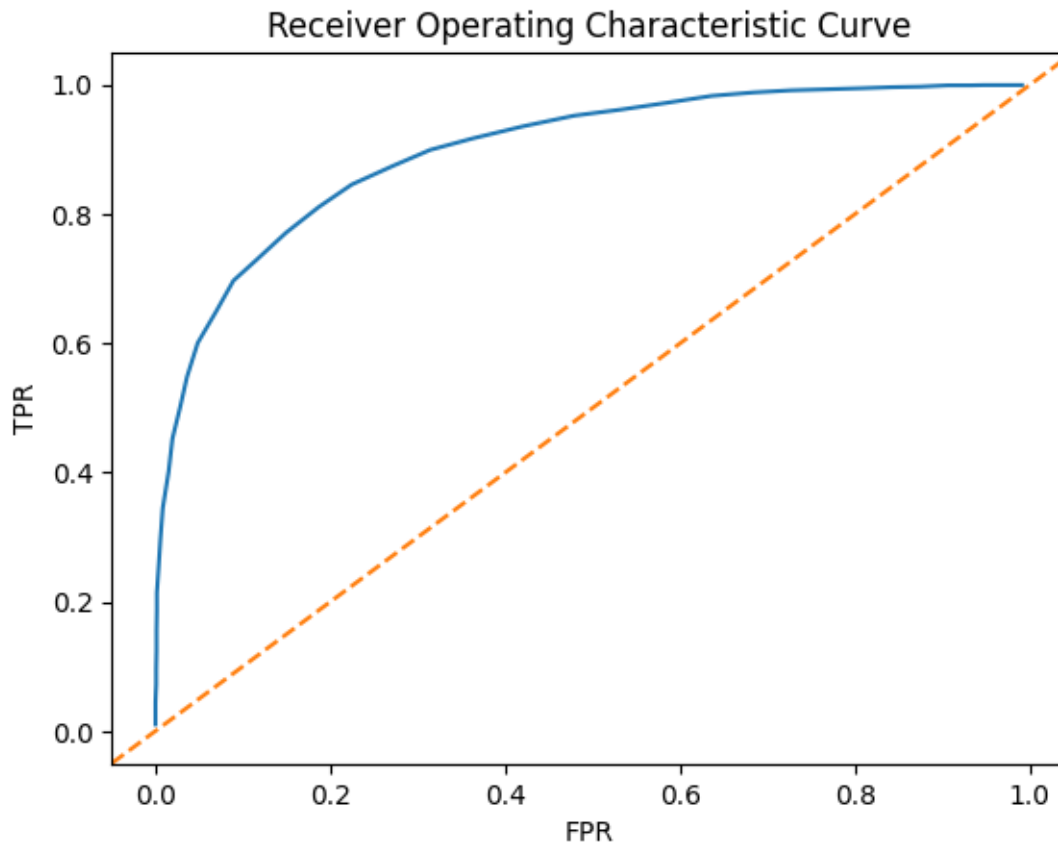
- (8) Wir wollen nun untersuchen, wie gut sich die Körpergröße für die Unterscheidung zwischen Männern und Frauen eignet, und welcher Schwellwert hier optimal ist. Dazu werden Sie für eine Menge von Schwellwerten die zugehörigen TPR und FPR-Werte bestimmen. Erstellen Sie zunächst eine Menge von Schwellwerten. 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.

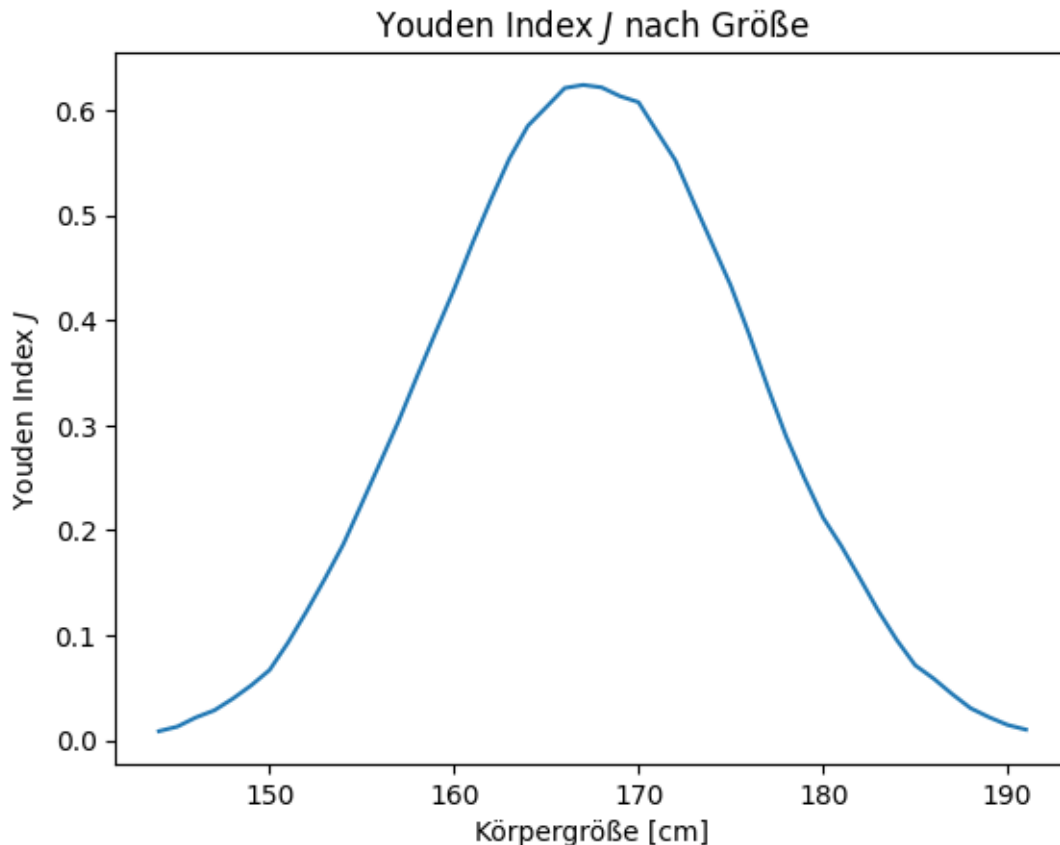
- (10) Schlagen Sie den Youden-Index in der Vorlesung nach und bestimmen Sie damit den optimalen Schwellwert zur Unterscheidung zwischen Frauen und Männern mithilfe der Körpergröße. Vergleichen Sie den erhaltenen Wert mit dem Wert, den Sie in Schritt (6) erhalten haben. (1-2 Sätze)

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]: 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.trapz:
def calc_AUC(roc: np.ndarray) -> float:
    roc = np.flipud(roc)
    return np.trapz(y=roc[:, 1], x=roc[:, 0], dx=1/roc[:, 1].shape[0])
calc_AUC(roc)
```

[17]: 0.8887302074735961

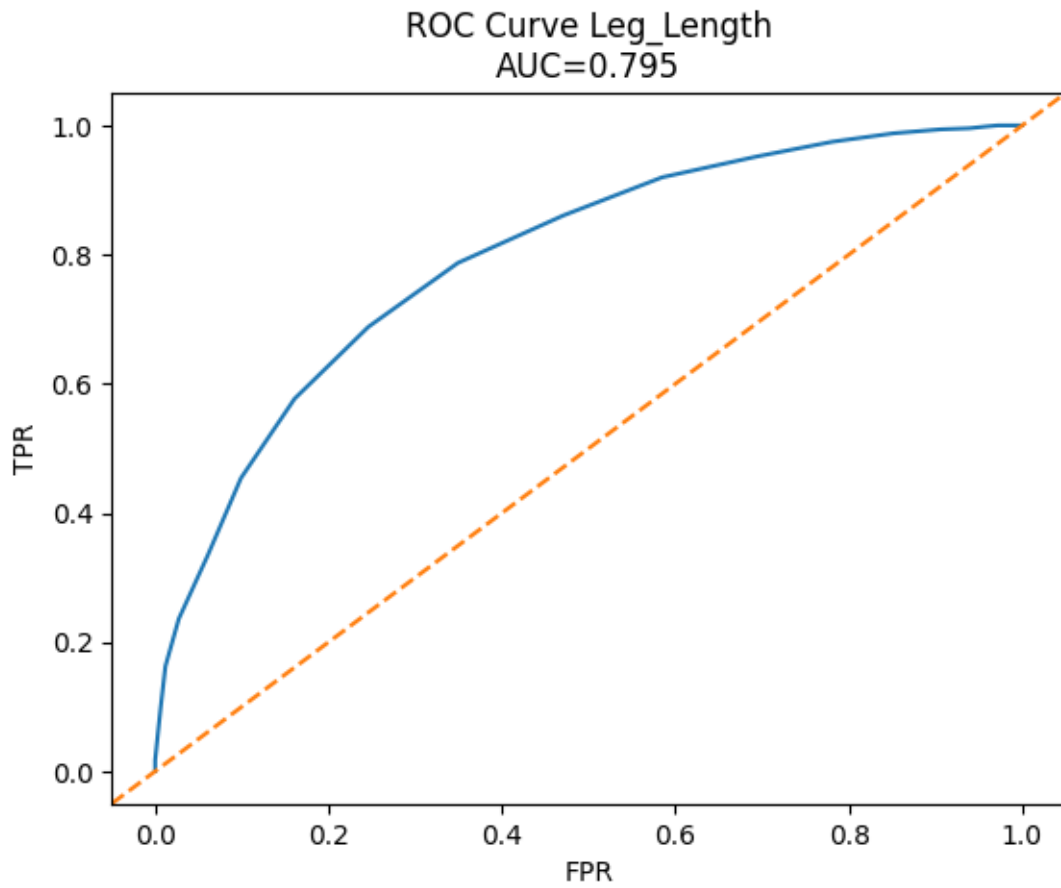
Ein Zufallsklassifikator würde eine AUC von 0.5 liefern, während ein perfekter Prädiktor/Klassifikator eine AUC von 1.0 liefern würde.

- (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?

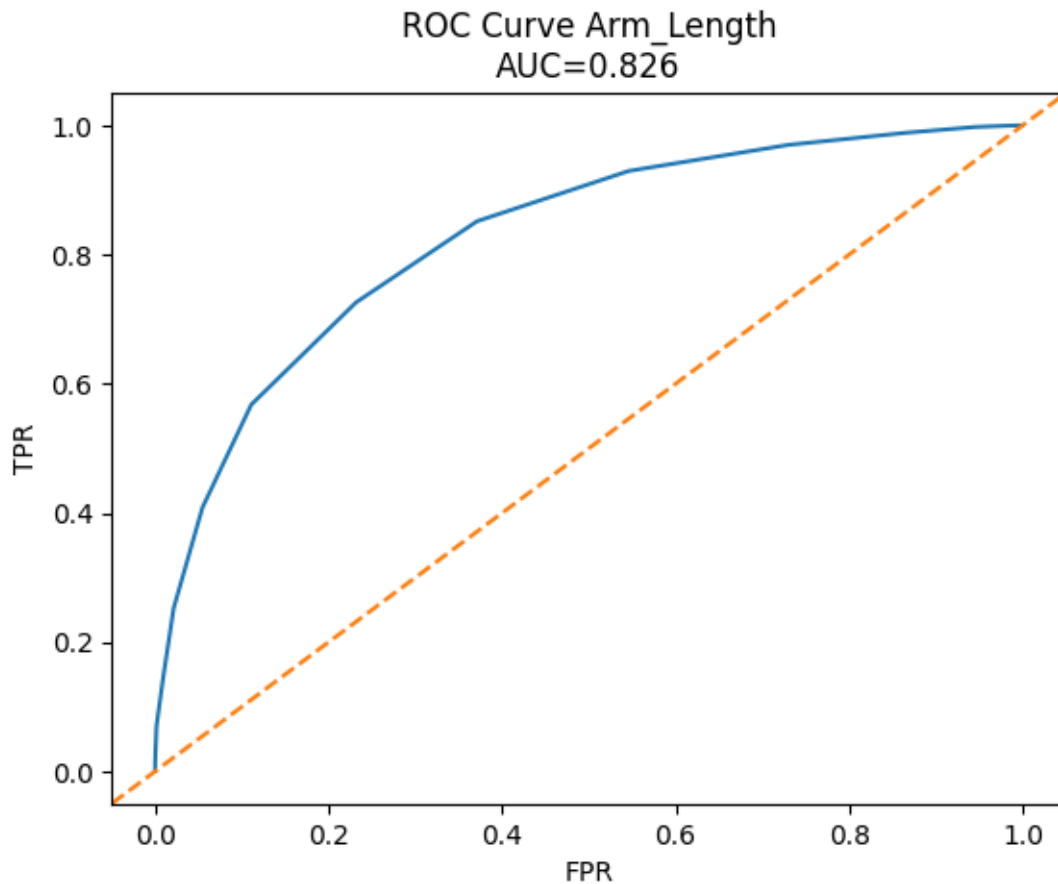
```
[18]: for merkmal in ["Leg_Length", "Arm_Length"]:
    schwellwerte: np.ndarray = np.arange(df[merkmal].min(), df[merkmal].max(), 1)

    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 =
    {calc_AUC(roc)}")
```



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 Epilepsietyp) 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

```

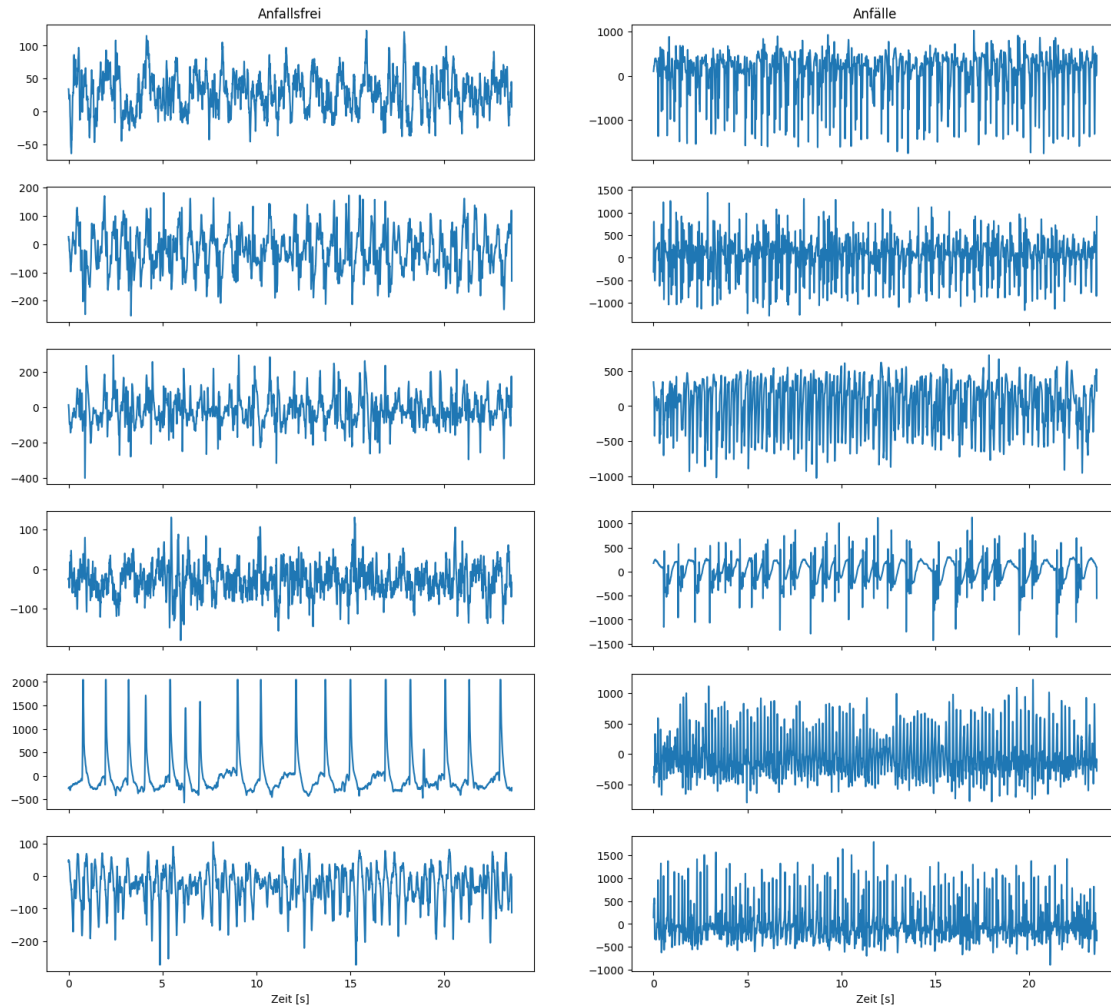
[4097 rows x 100 columns]

```

[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()

```



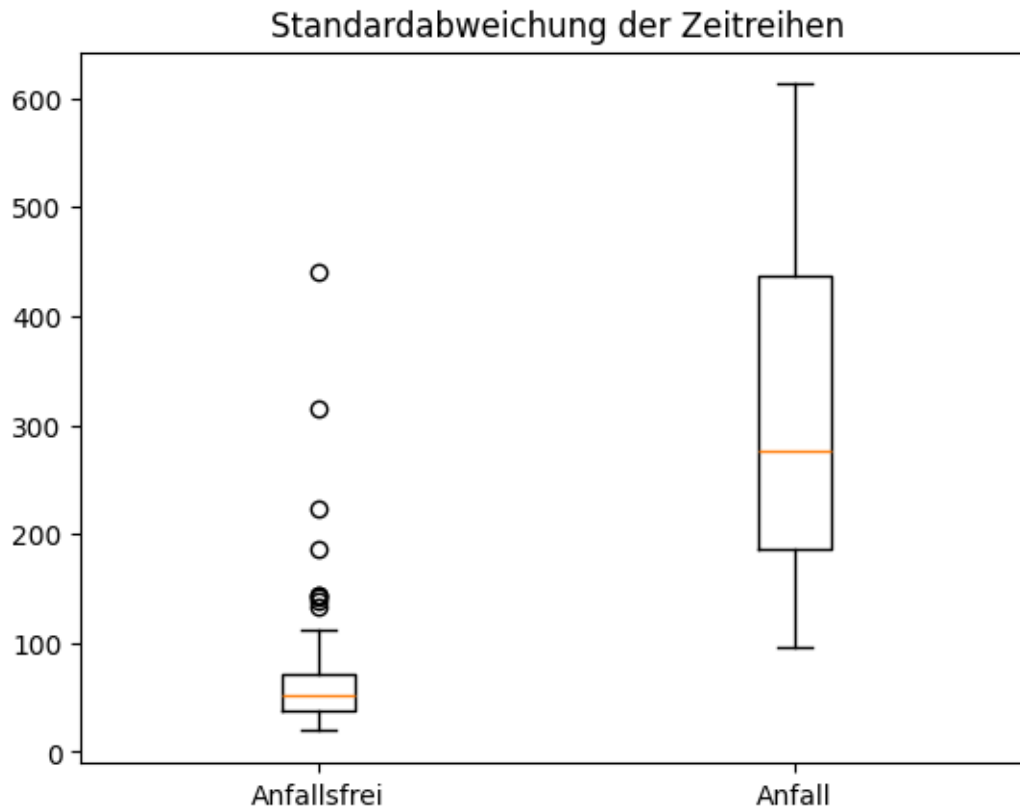
- (2) Normieren Sie alle Zeitreihen, sodass jede Zeitreihe den Mittelwert 0 aufweist. Auf diesen Daten werden Sie von nun an weiterarbeiten.

```
[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.

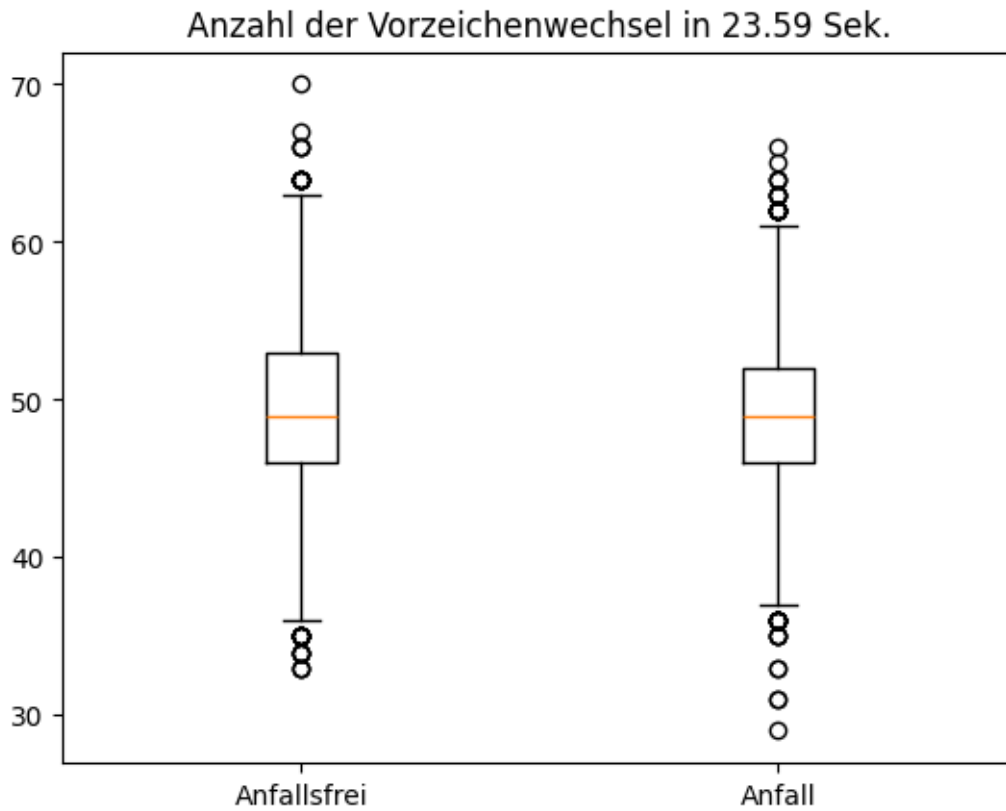
- Nutzen Sie Boxplots für eine schnelle Einschätzung, ob sich die Verteilung eines Merkmals für Datensatz D von der Verteilung desselben Merkmals für Datensatz E unterscheidet.
- Tipp: Probieren Sie mehrere Ideen aus aber halten Sie es einfach.

```
[24]: plt.title("Standardabweichung der Zeitreihen")
      plt.boxplot([df_D.std(), df_E.std()], tick_labels=["Anfallsfrei", "Anfall"])
      plt.show()
```

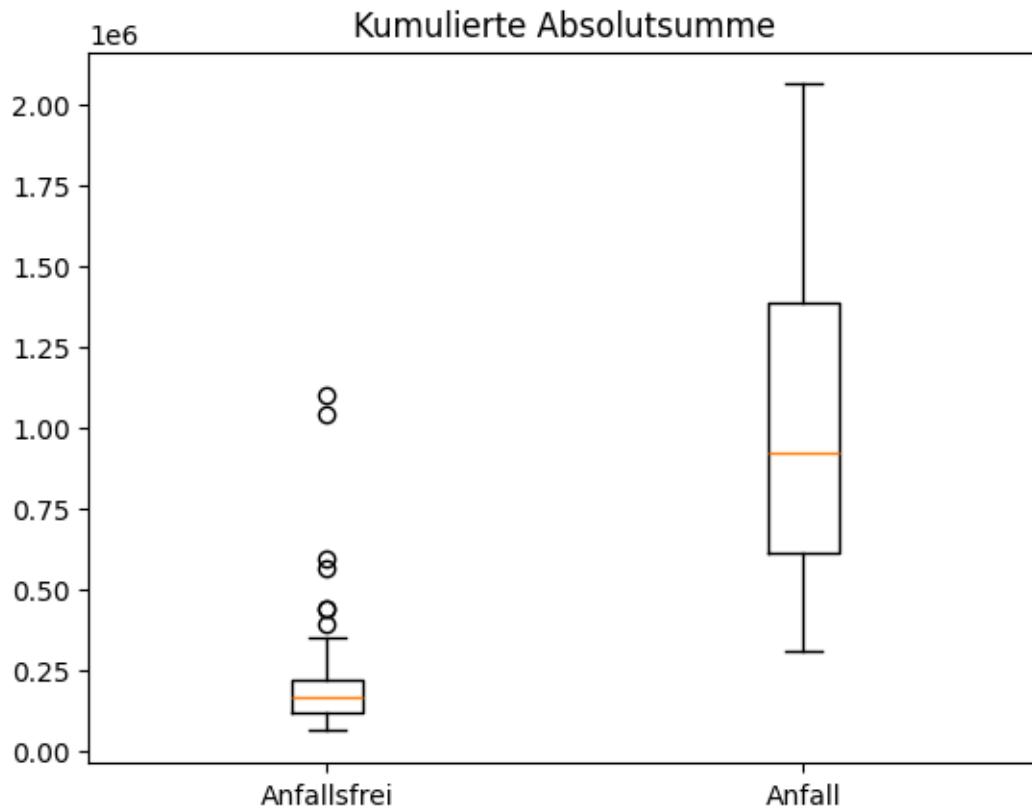


```
[25]: plt.title(f"Anzahl der Vorzeichenwechsel in {4096/173.61:.2f} Sek.")
def vzw(x: pd.Series) -> float:
    count = 0
    for i in range(len(x) - 1):
        if (x.iloc[i] < 0 < x.iloc[i + 1]) or (x.iloc[i] > 0 > x.iloc[i + 1]):
            count += 1
    return count

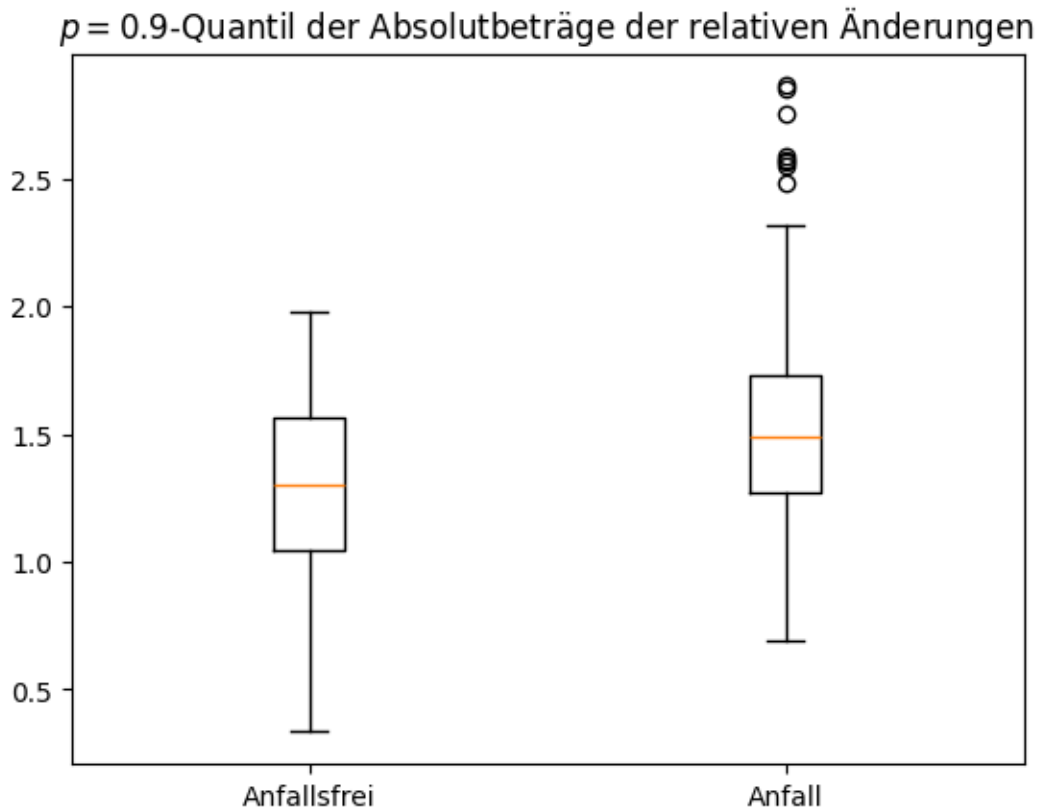
plt.boxplot([df_D.agg(vzw, axis="columns"), df_E.agg(vzw, axis="columns")],
            tick_labels=["Anfallsfrei", "Anfall"])
plt.show()
```



```
[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()
```



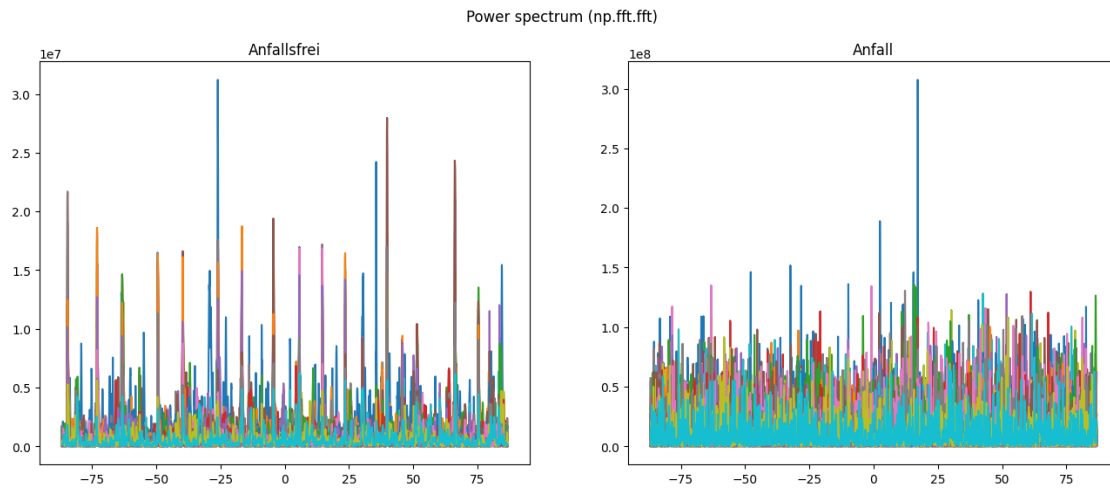
```
[28]: freqs = np.fft.fftfreq(4096, 1/173.61)
      idx = np.argsort(freqs)

      ps_D = np.abs(np.fft.fft(df_D)) ** 2
      ps_E = np.abs(np.fft.fft(df_E)) ** 2

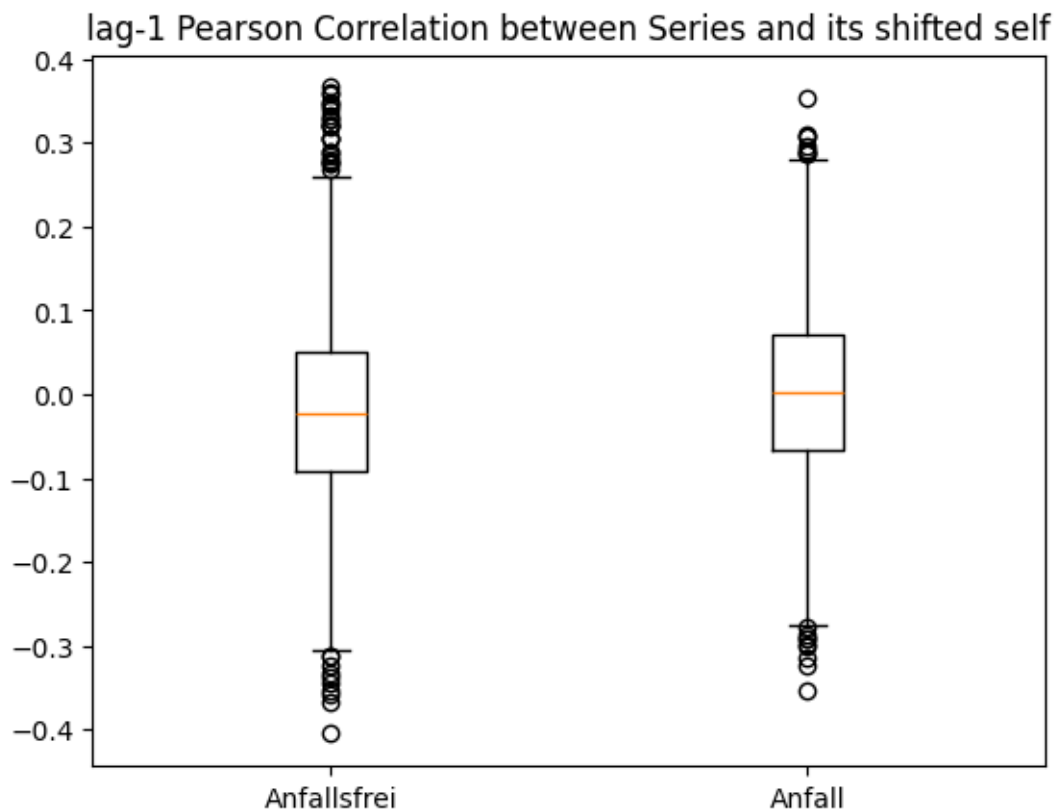
      fig, axs = plt.subplots(1, 2, figsize=(16, 6))

      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()
```

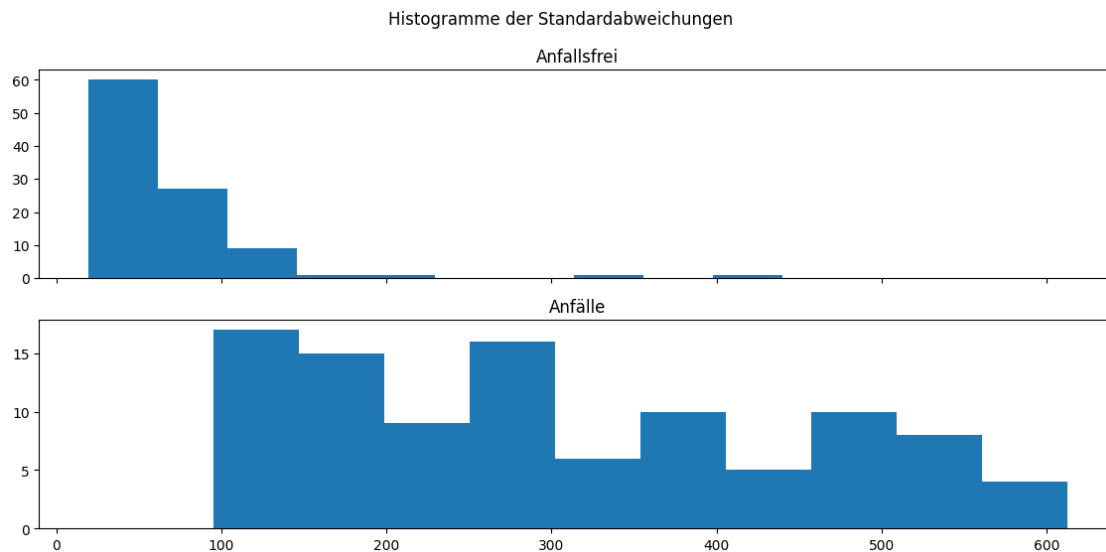


- (4) Visualisieren Sie in einem Histogramm die beiden Verteilungen des Merkmals, für das Sie sich in Schritt 3 entschieden haben.

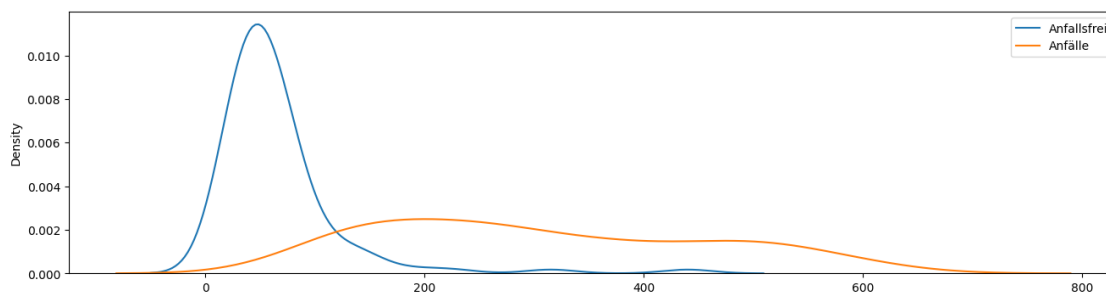
```
[30]: fig, axs = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

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()
```



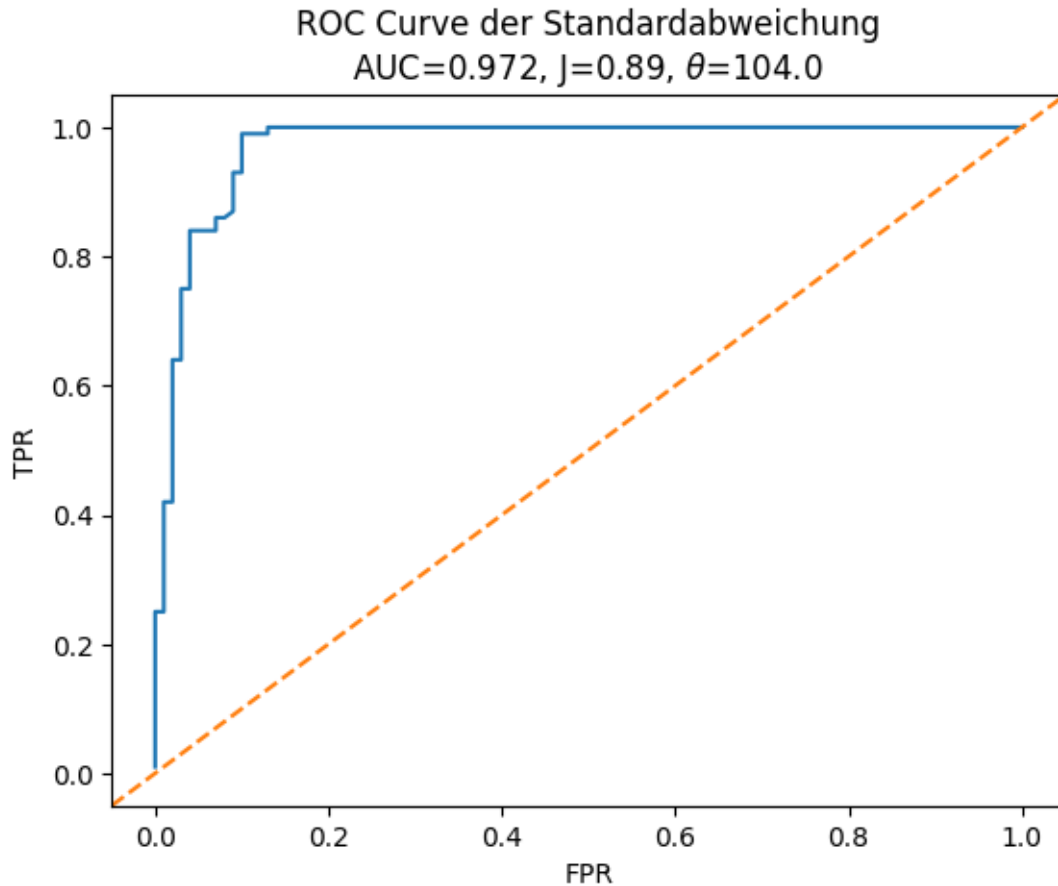
- (5) Visualisieren Sie die ROC-Kurve für Ihr Merkmal. Erstellen Sie zunächst eine Liste mit Schwellwerten, die Sie ausprobieren wollen. Nutzen Sie dann Ihren Code aus Aufgabe 11.1, um die ROC-Kurve zu erzeugen und zu visualisieren. Ermitteln Sie mithilfe des Youden-Index den optimalen Schwellwert für 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 =\n
    ↪{calc_AUC(roc)}")
print(f"Der optimale Schwellenwert liegt dafür bei\n
    ↪{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 $P(f) = \frac{|\underline{X}(f)|^2}{2}$ mit der Fouriertransformierten Zeitreihe $\underline{X}(f)$ - die Effektive Bandbreite $B_{\text{eff}} = \frac{\text{PM}}{T \cdot \text{FM}^2}$ - 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)

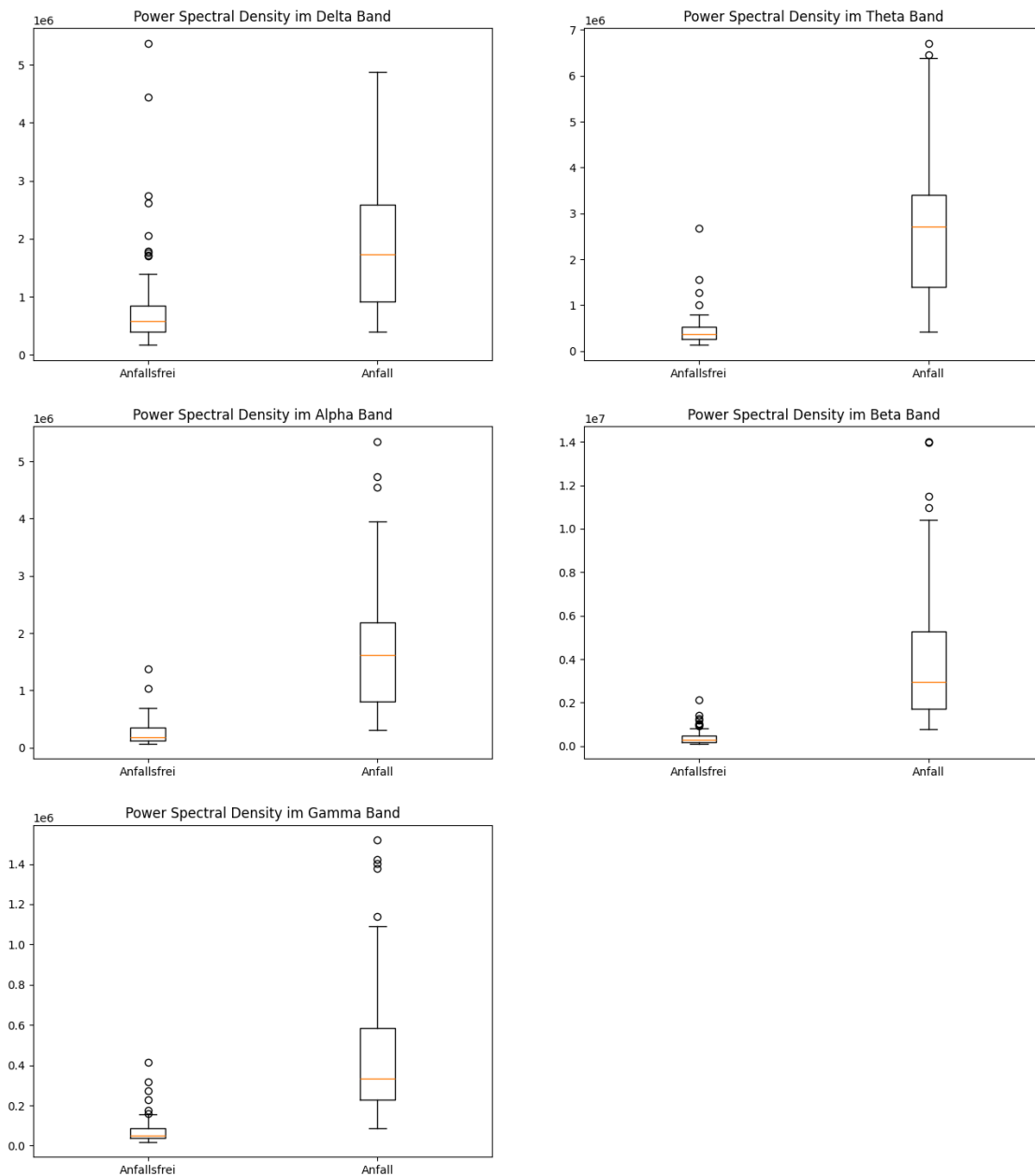
          # Frequenzbänder
          freq_bands = {
              "delta": (0, 4),
              "theta": (4, 8),
              "alpha": (8, 12),
              "beta": (12, 30),
              "gamma": (30, 45),
          }
          df_powers = pd.DataFrame()
          for band, (a, b) in freq_bands.items():
              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

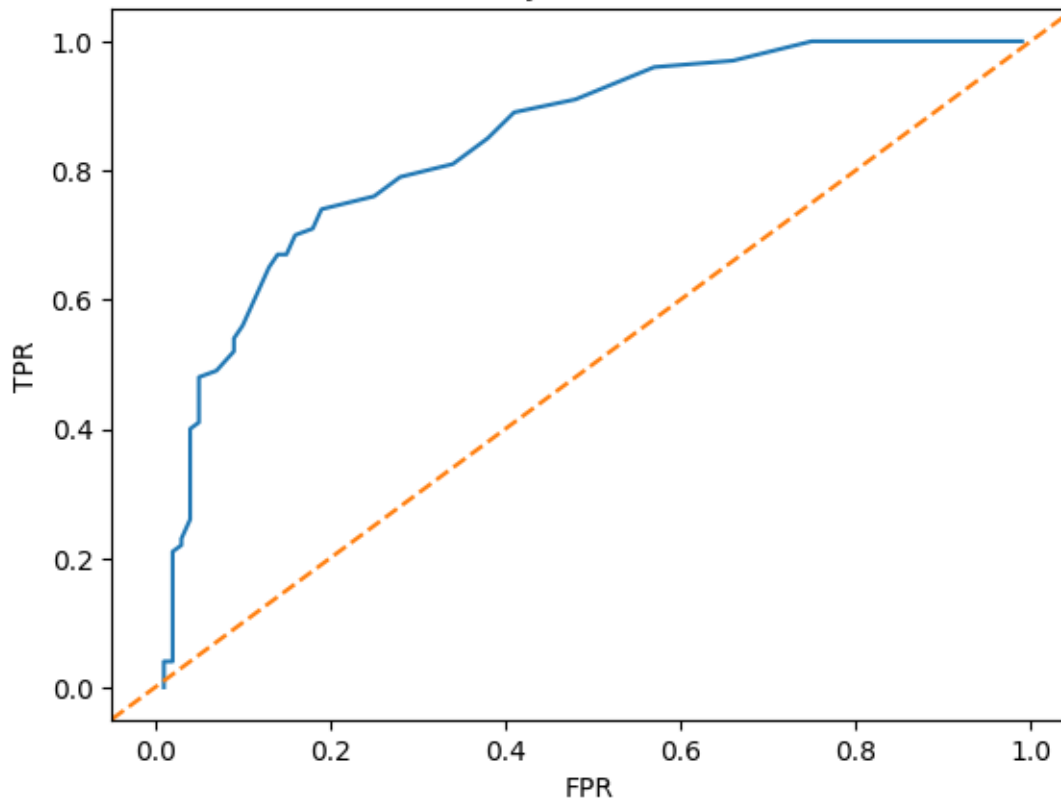
        roc = np.array(roc)
        phi_tilde_idx = np.argmax(youdens)

        plt.plot(roc[:, 0], roc[:, 1])
        plt.title(f"ROC Curve: {classifier}\nAUC={calc_AUC(roc):.3f},
        ↪J={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()

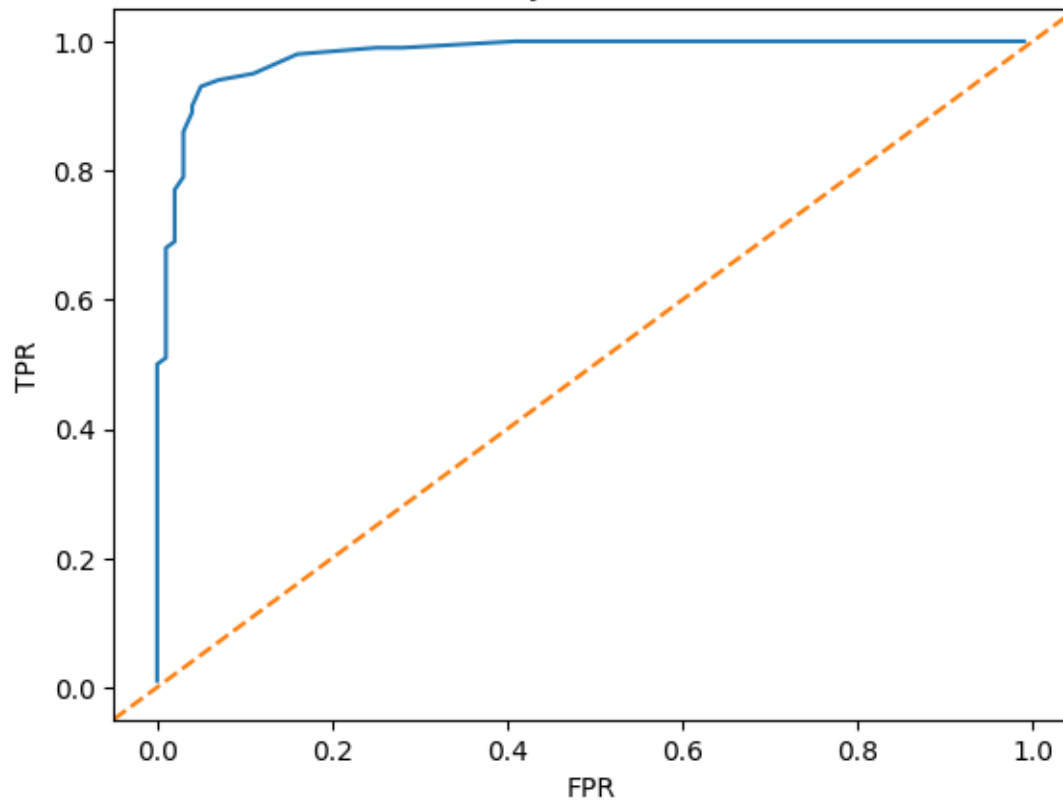
    for i, col in enumerate(df_Dpow.columns):
        roc_plot(df_Epow[col], df_Dpow[col], f"PSD auf dem {col.capitalize()}-Band")

```

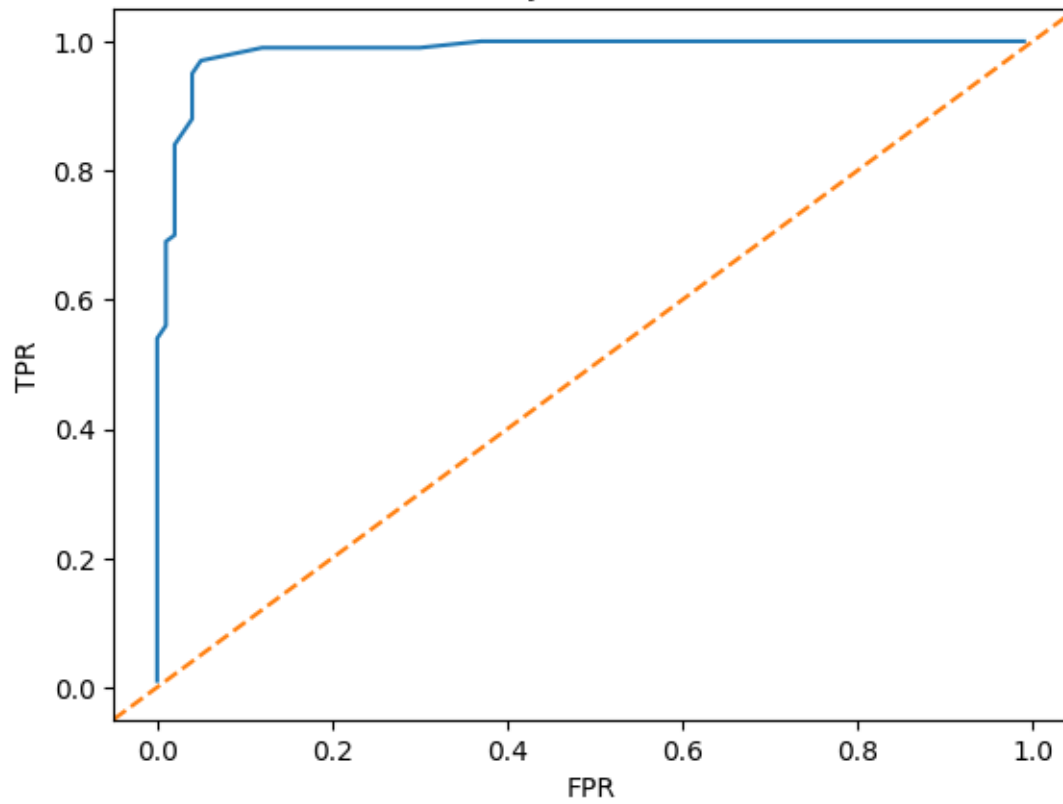
ROC Curve: PSD auf dem Delta-Band
AUC=0.832, $J=0.55$, $\theta=954272.9$



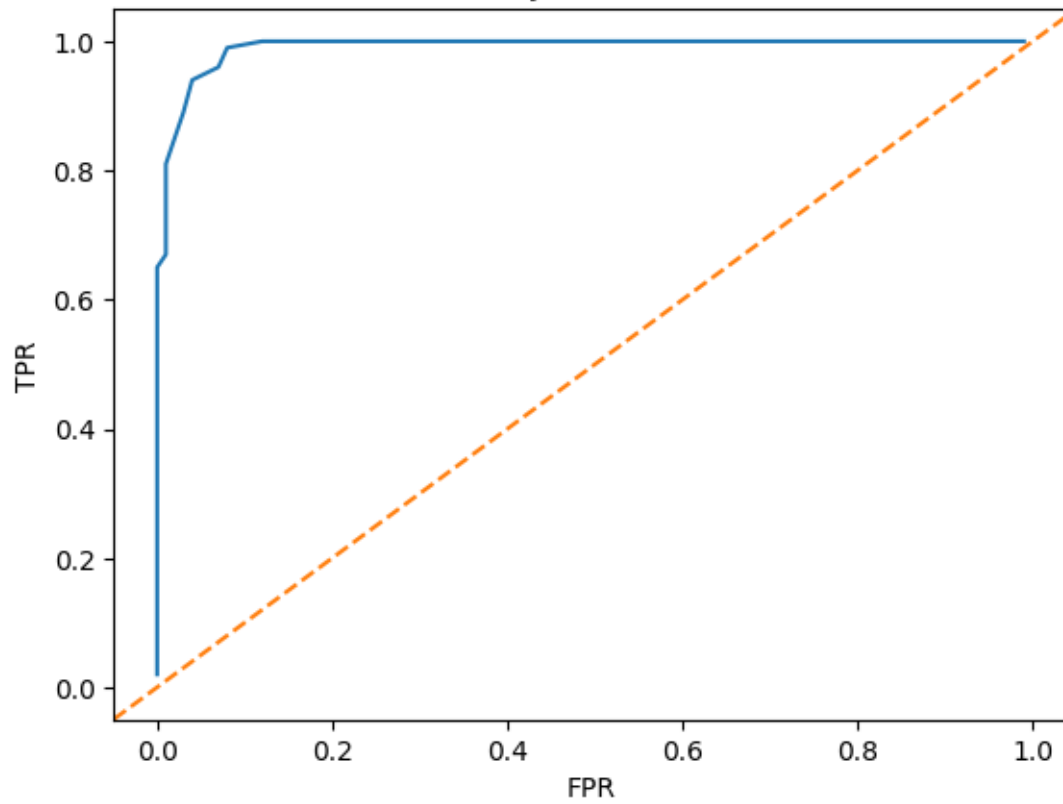
ROC Curve: PSD auf dem Theta-Band
AUC=0.970, $J=0.88$, $\theta=792258.9$

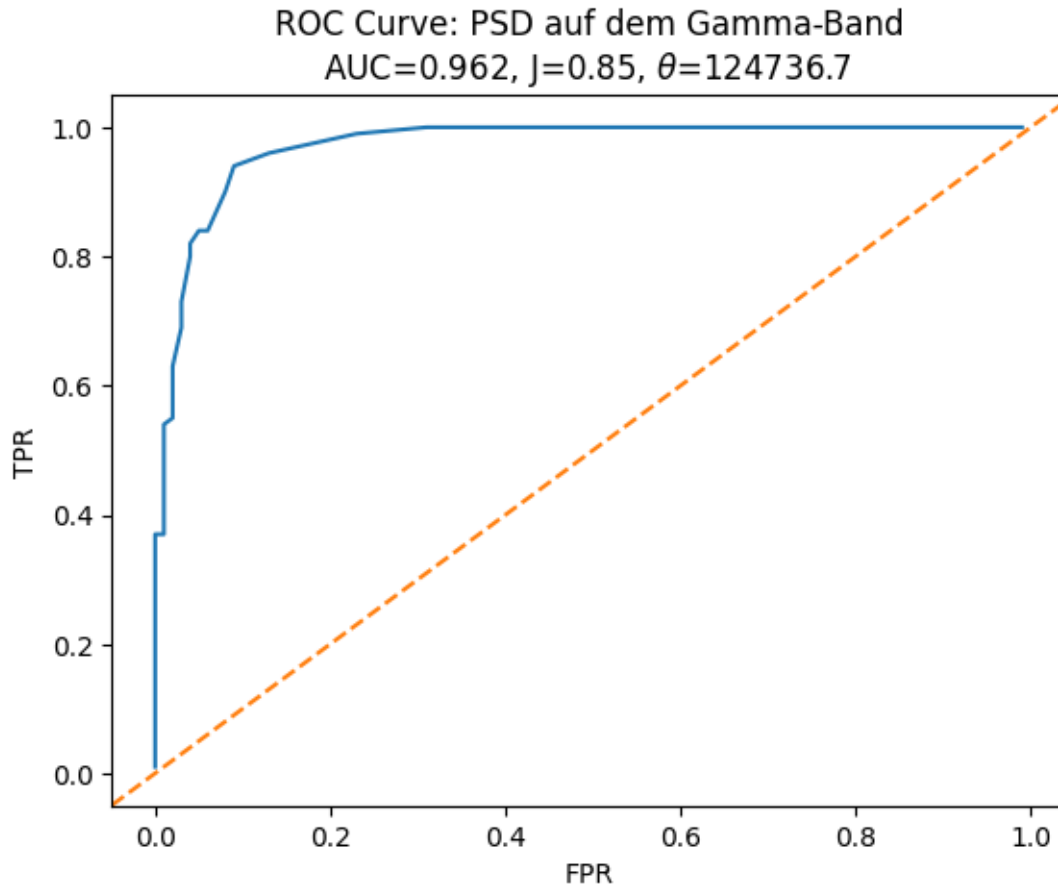


ROC Curve: PSD auf dem Alpha-Band
AUC=0.976, $J=0.92$, $\theta=497822.9$



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.

```
[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)
```

Ihre Aufgaben

- (1) Importieren Sie die Daten, indem Sie die obere Code-Zelle ausführen.

```
[38]: #!wget https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data
```

- (2) Vergewenwärtigen 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?

```
[39]: df
```

```
[39]:
```

	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	
..	
173	3	13.71	5.65	2.45	20.5	95	
174	3	13.40	3.91	2.48	23.0	102	

175	3	13.27	4.28	2.26	20.0	120
176	3	13.17	2.59	2.37	20.0	120
177	3	14.13	4.10	2.74	24.5	96

	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	
..	
173	1.68	0.61	0.52	1.06	
174	1.80	0.75	0.43	1.41	
175	1.59	0.69	0.43	1.35	
176	1.65	0.68	0.53	1.46	
177	2.05	0.76	0.56	1.35	

	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
..
173	7.70	0.64	1.74	740
174	7.30	0.70	1.56	750
175	10.20	0.59	1.56	835
176	9.30	0.60	1.62	840
177	9.20	0.61	1.60	560

[178 rows x 14 columns]

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` angekommen 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):

```
[40]: 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))])

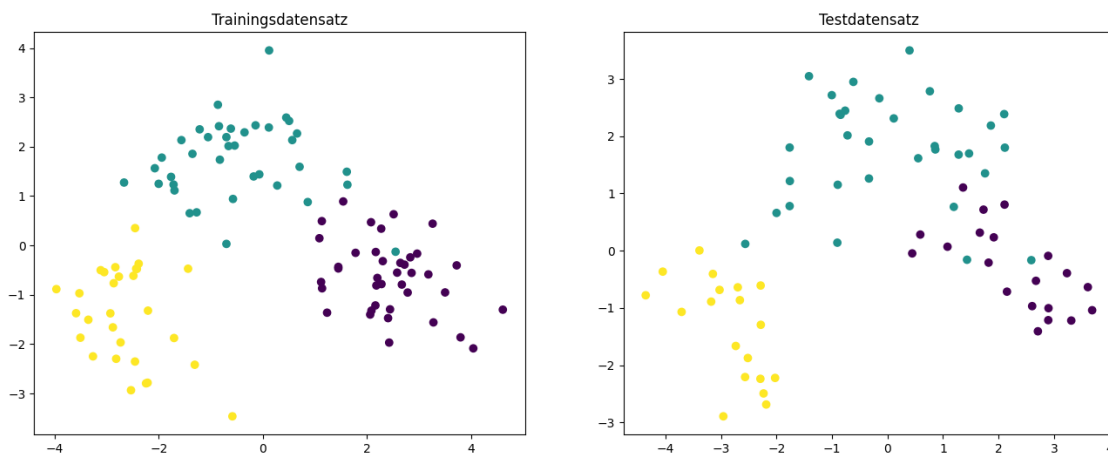
      X_train_p = pipe.fit_transform(X_train)
      X_test_p = pipe.transform(X_test)
```

- (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.

```
[41]: 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 `MN`, 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.

```
[42]: 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`.

```
[43]: 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.

```
[44]: (y_pred == y_test).sum() / y_pred.size
```

```
[44]: 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.

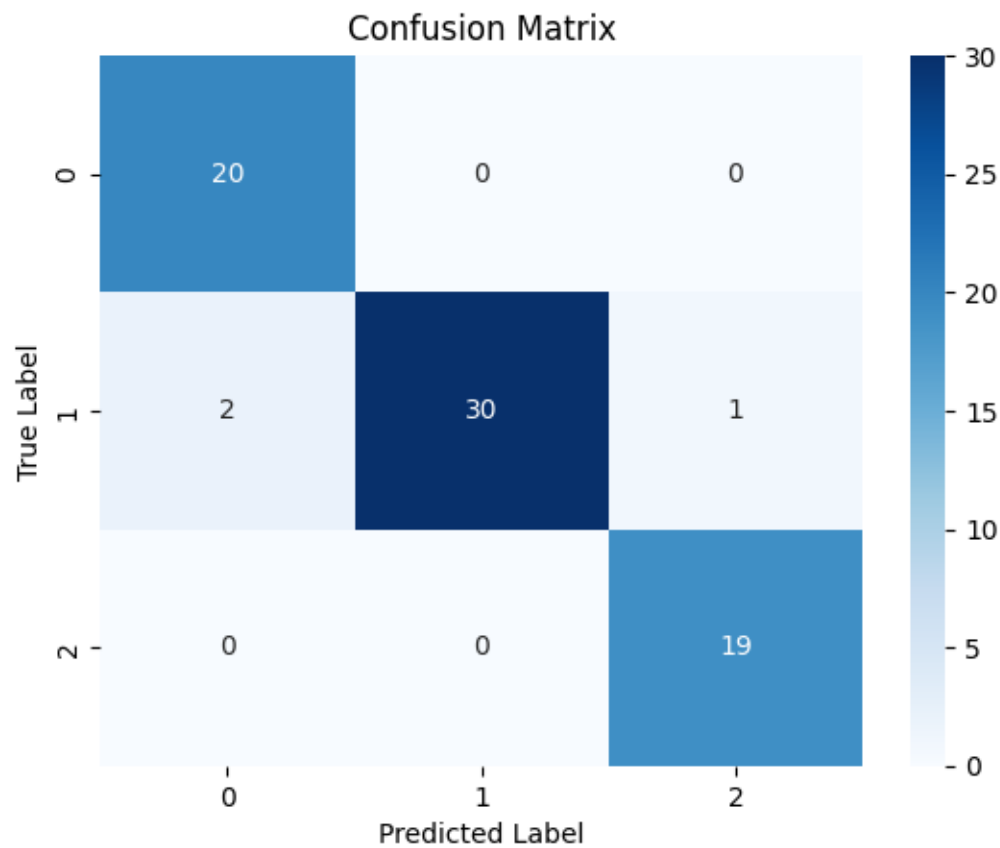
```
[45]: def confusion_matrix(y: np.ndarray, y_pred: np.ndarray):
      classes: np.ndarray = np.unique(y)
      array = []
      for true_label in classes:
          row = []
          for predicted_label in classes:
              row.append(((y == true_label) & (y_pred == predicted_label)).sum())
          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 Fehlzusordnungen passieren.