

Modul Statistik und Machine Learning Modelle



Autoren

Dr. rer. nat. Joachim Krois

Leitender Datenwissenschaftler in der Abteilung ODDH2, Charité - Universitätsmedizin Berlin. Er arbeitet vorwiegend in den Bereichen Statistik, prädiktive Modellierung, maschinelles Lernen, Computer Vision, Epidemiologie und Versorgungsforschung. Ein Arbeitsschwerpunkt ist die Entwicklung konkreter Lösungen zur Verbesserung der zahnmedizinischen Diagnostik durch die Anwendung von Techniken des maschinellen Lernens. Joachim Krois ist ein Advokat des Paradigmas der Datenzahnmedizin. Hierbei wird der Komplexität der Entscheidungsfindung im medizinischen Bereich Rechnung getragen indem Kosten, Nutzen und Vertrauenswürdigkeit von Prognosesystemen kritisch evaluiert werden. Er ist Teil der ITU/WHO Focus Group Artificial Intelligence for Health (AI4H)

und verantwortet dort in der Themengruppe “Dental diagnostics and digital dentistry” kollaborativ und interdisziplinär am Benchmarking für KI-Algorithmen. Dadurch sollen die Generalisierbarkeit und Vertrauenswürdigkeit von KI-Systemen verbessert werden.

Mag. rer. nat. Jan Senekowitsch

Studium der theoretischen Physik an der Karl-Franzens Universität Graz und wissenschaftlicher Mitarbeiter an der Technischen Universität Graz im Arbeitsbereich Quantenchemie. Derzeit arbeitet er als Consultant und Freelancer.

Vorwort

Im Jahr 2000 wurde der Datenwissenschaftler als der “ Sexiest Job of the 21st Century” bezeichnet ([Harvard Business Review, 2012](#)). Diese mutige Prognose beruht auf der Vorahnung eines Paradigmenwechsels in Wissenschaft und Forschung, der mehr und mehr auch zu sozio-ökonomischen Veränderungen führen würde.

Bereits im 17. Jahrhundert sind die experimentellen und theoretischen Wissenschaften entstanden, die heute als die beiden grundlegenden Forschungsparadigmen für das Verständnis der Natur gelten. In den letzten Jahrzehnten sind Computersimulationen zu einem dritten Paradigma geworden. Heute, an der Schwelle zur Big-Data-Ära hat sich ein viertes Paradigma herausgebildet, welches auf datenintensiven Methoden beruht. Viele Branchen machen sich das datengetriebene Paradigma bereits zu eigen und nutzen Daten, um Wert zu erschaffen. Daten gelten als eine Schlüsselressource für moderne Gesellschaften und erfüllen eine Reihe von Merkmalen, um eine technologische Revolution auszulösen. Dazu gehören unter anderem, dass die Kosten für die Nutzung von Daten niedrig sind, Daten eine unerschöpfliche Ressource darstellen, da sie beliebig oft und von beliebig vielen Akteuren genutzt werden können, und eine intelligente Nutzung von Daten die Effizienz von Prozessen und Verfahren erhöht. Viele große Technologieunternehmen haben den Wert von Daten bereits in den 1990er und 2000er Jahren erkannt, in denen auch der Begriff “Data ist the new oil” geprägt wurde ([Economist 2017](#)).

Dabei sind statistische Methoden und der Einsatz von computer-gestützten Methoden essentiell für die Auswertung und Interpretation von Daten. Ziel dieses Moduls ist es die Grundlagen der deskriptiven und der schliessenden Statistik aber auch Programmierkenntnisse zu vermitteln, um diese zur Datenauswertung und Problemlösung einsetzen zu können. Die Verbindung eines

mathematischen Fachwissens mit Programmierkenntnissen sowie eine problembezogene und lösungsorientierte Anwendung derselben wird oft als "Data Science" beschrieben.

In den letzten Jahrzehnten erfreute sich die Programmiersprache Python wachsender Beliebtheit. Python hat sich zunehmend als Standardsprache in der Data Science etabliert da sie einer einfachen gut leserlichen Syntax folgt aber auch strukturierte, objektorientierte, aspektorientierte und funktionale Programmierstile ermöglicht. Ein weitere Vorzug von Python ist der einfache Umgang mit umfangreichen Bibliotheken wie [NumPy](#), [pandas](#) und [matplotlib](#).

In diesem Sinne wünschen Ihnen die Autoren viel Erfolg uns Spaß mit dem Erlernen von statistischen Methoden und deren Anwendung mithilfe der Programmiersprache Python.

Berlin, August 2022

Dr. rer. nat. Joachim Krois

Lernziele

Strukturierte Datensätze und deskriptive Statistik

- Der Umgang mit strukturierten Datensätzen wird anhand Panda Dataframes vorgestellt. Das Erstellen, Modifizieren und Selektieren von Daten wird demonstriert.
- Die Grundlagen der deskriptiven Statistik wie unterschiedliche Möglichkeiten die zentralen Tendenz von Daten zu bestimmen werden am Beispiel von Mittelwert, Modalwert, Median besprochen. Es werden anhand des [students](#) Beispieldatensatzes deren Vor- und Nachteile veranschaulicht.
- Weiters werden Streuungsmaße und deren Bedeutung in der deskriptiven Statistik diskutiert. Begriffe wie Varianz, Standardabweichung werden erklärt. Es wird durch numerische Simulation das Tschebyscheff-Theorem vorgestellt.
- Letztlich werden Maße für die Korrelation zwischen Variablen und deren Bedeutung in der deskriptiven Statistik besprochen. Die Studierenden lernen Kovarianz, Pearson- und Spearman Korrelation und Kontingenzkoeffizient kennen.
- Zusammengefasst werden folgende Begriffe und Methoden besprochen: Maße der zentralen Tendenz, Streuungsmaße, Positionsmaß, 5-Punkte-Zusammenfassung, Ausreißer und Boxplots, Maße für die Relation zwischen Variablen

Deskriptive Statistik

```
import pandas as pd
```

Angewandte Statistik lässt sich in zwei Bereiche unterteilen: [deskriptive Statistik](#) und [Inferenzstatistik](#).

Die deskriptive Statistik umfasst Methoden zur Organisation, Darstellung und Beschreibung von Daten mit Hilfe von Tabellen, Diagrammen und Streuungsmaßen. Im Gegensatz dazu besteht die Inferenzstatistik aus Methoden, die Stichprobenergebnisse verwenden, um Entscheidungen oder Vorhersagen über eine Grundgesamtheit zu treffen (s.1–151, 231–312, s.10, 12).

Das Wort [univariat](#) bezieht sich auf die Tatsache, dass nur eine Variable betrachtet wird. Der Hauptzweck der univariaten Statistik besteht darin, die Daten zu beschreiben und zusammenzufassen. Wenn zwei oder mehr Variablen analysiert werden, spricht man von [bivariater](#) oder [multivariater Analyse](#) bzw. Statistik. In diesem Fall sind wir in erster Linie an den Beziehungen zwischen und unter einer Reihe von Variablen interessiert.

Strukturierte Datensätze

Strukturierte Daten

Bei [Data Science](#) geht es um die Gewinnung von Wissen aus [Daten](#). Daten sind eine spezifische Form von [Informationen](#) und weisen verschiedene Abstraktions- und Strukturniveaus auf ([strukturiert](#), [halbstrukturiert](#) oder [unstrukturiert](#)).

Eine sehr verbreitete [Datenstruktur](#) ist ein Array. In verschiedenen Bereichen gibt es andere Bezeichnungen für einen solchen Datentyp, die synonym verwendet werden, z. B. Matrix () in der Mathematik, [Tabelle](#) in Datenbanken, [Tabellenkalkulation](#) und [Dataframe](#) , der eine grundlegende Python-Objektklasse ist z. B. : (Pandas [DataFrame](#)).

Daten eines solchen Typs bestehen aus Beobachtungen und entsprechenden Variablen, die oft als Merkmale bezeichnet werden.

id	Name	Age
1	John	26
2	Alice	20
3	Mike	21
4	Anne	25

In diesem Beispiel entsprechen die **Beobachtungen** (*Stichprobe* genannt) einer Anzahl von Personen. Jede beobachtete Person wird durch eine Reihe von **Variablen** (so genannte *Merkmale*) charakterisiert: Durch eine Identifikationsnummer (id), durch einen Namen und durch ein Alter. In unserem Beispiel ist es sehr einfach, sich durch einen Blick auf die Tabelle einen Gesamteindruck von den Daten selbst zu verschaffen. Wir erkennen sofort, dass es in unserer Stichprobe 4 Personen gibt, zwei Frauen und zwei Männer. Außerdem sehen wir sofort, dass die jüngste Person 20 Jahre alt ist und Alice heißt und die älteste Person 26 Jahre alt ist und John heißt.

Anwendungen in der realen Welt enthalten jedoch oft eine große Menge an Daten. Hunderte, Tausende, Millionen oder sogar Milliarden von Beobachtungen, kombiniert mit Tausenden von Variablen, können einen Datensatz bilden. Für den Menschen ist es unmöglich, allein durch die Betrachtung solcher Datensätze irgendwelche Schlussfolgerungen über die Daten zu ziehen. Daher reduzieren wir die Daten auf eine überschaubare Größe, indem wir Tabellen erstellen, Diagramme zeichnen oder zusammenfassende Maße wie Durchschnittswerte berechnen. Diese Art von statistischen Methoden wird als **deskriptive Statistik** bezeichnet (s.10).

Der **students** Datensatz

In diesem Abschnitt werden wir einen Datensatz namens **students** untersuchen. Zunächst laden wir den Datensatz, geben ihm einen geeigneten Namen und verschaffen uns einen Eindruck von seiner Struktur und Größe, indem wir die Methode **info()** auf den Datensatz anwenden.

```
# Lese Datei students.csv als Dataframe ein
df = pd.read_csv("../data/students.csv")
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8239 entries, 0 to 8238
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   stud_id           8239 non-null    int64  
 1   name              8239 non-null    object  
 2   gender             8239 non-null    object  
 3   age                8239 non-null    int64  
 4   height             8239 non-null    int64  
 5   weight             8239 non-null    float64 
 6   religion            8239 non-null    object  
 7   nc_score            8239 non-null    float64 
 8   semester            8239 non-null    object  
 9   major               8239 non-null    object  
 10  minor               8239 non-null    object  
 11  score1              4892 non-null    float64 
 12  score2              4892 non-null    float64 
 13  onlineTutorial      8239 non-null    int64  
 14  graduated            8239 non-null    int64  
 15  salary               1753 non-null    float64 
dtypes: float64(5), int64(5), object(6)
memory usage: 1.0+ MB

```

Der Studentendatensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id*, *name*, *gender*, *age*, *height*, *weight*, *religion*, *nc_score*, *semester*, *major*, *minor*, *score1*, *score2*, *onlineTutorial*, *graduated*, *salary*. Neben dem jeweiligen Variablenamen listet die Methode `info()` die `Klasse` jeder einzelnen Variablen auf. Alle Objekte in Python haben eine Klasse, z. B. `numerische` Datentypen, die in die Unterklassen `(int)` Ganzzahlen, `(float)` Gleitkommazahlen und `(imag)` komplexe Zahlen, eingeteilt werden.

In den nächsten Abschnitten werden wir die deskriptiven Statistiken des `students` Datensatzes genauer untersuchen.

Maße der zentralen Tendenz

```

import statistics as st
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

```

Um ein Gefühl für einen Datensatz zu bekommen, kann er durch numerische Streuungsmaße beschrieben werden. Es gibt drei Hauptmaße für die **zentrale Tendenz**: den **Mittelwert**, den **Median** und den **Modalwert**; es gibt jedoch auch andere Maße für die zentrale Tendenz, wie z. B. das **harmonische Mittel**, das **gewichtete Mittel** und das **geometrische Mittel**, um nur einige zu nennen.

Der Mittelwert

Der arithmetische Mittelwert

$$\text{Mittelwert} = \frac{\text{Summe aller Werte}}{\text{Anzahl aller Werte}}$$

Das für Stichprobendaten berechnete arithmetische Mittel wird mit \bar{x} (gelesen als “x strich”) bezeichnet, und das Mittel für Grundgesamtheitsdaten wird mit μ (griechischer Buchstabe mu) bezeichnet. Der Mittelwert kann also durch die folgenden Gleichungen ausgedrückt werden:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^n x_i$$

oder

$$\mu = \frac{1}{N} \cdot \sum_{i=1}^N x_i$$

wobei $\sum_{i=1}^n x_i$ die Summe aller Werte (x_1, x_2, \dots, x_n) , N der Umfang der Grundgesamtheit und n der Stichprobenumfang ist. Betrachten wir die Variable `height` im Datensatz `students` und berechnen wir ihr arithmetisches Mittel. Beachten Sie, dass es sich bei dem `students` Datensatz um eine *Stichprobe* und **nicht** um die Grundgesamtheit aller Studenten handelt. Daher berechnen wir \bar{x} , den Stichprobenmittelwert.

```
df = pd.read_csv("../data/students.csv", index_col=0)
# Wende Funktion mean() auf Spalte 'height' an und speicher Ergebnis in height_mean
height_mean = df["height"].mean()
print(f"\"Mittelwert der Körpergröße der Studenten: {height_mean}\"")
```

Mittelwert der Körperlänge der Studenten: 171.38075009103048

Das geometrische Mittel

Bei der Untersuchung von Phänomenen wie der Inflation oder der Bevölkerungsentwicklung, die mit periodischen Zu- oder Abnahmen einhergehen (so genannte Veränderungsraten), ist das **geometrische Mittel** besser geeignet, um die durchschnittliche Veränderung über den gesamten Untersuchungszeitraum zu ermitteln. Um das geometrische Mittel einer Folge von n Werten x_1, x_2, \dots, x_n zu berechnen, werden diese miteinander multipliziert und dann die n -te Wurzel aus diesem Produkt ermittelt.

$$\bar{x}_{geo} = \sqrt{x_1 \cdot x_2 \cdots x_n}$$

was umgeschrieben werden kann als

$$\bar{x}_{geo} = \sqrt{x_1 \cdot x_2 \cdots x_n} = \left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}} = \sqrt[n]{\prod_{i=1}^n x_i}$$

Lassen Sie uns das an einem Beispiel verdeutlichen:

Wir betrachten die jährlichen Wachstumsraten eines Schwarmes von Honigbienen über einen Zeitraum von 5 Jahren. Diese Veränderungsraten sind: 14%, 26%, 16%, -38%, -6%. Außerdem wissen wir, dass sich zu Beginn des Beobachtungszeitraums 5.000 Bienen im Schwarm befunden haben. Wir suchen nach der mittleren Rate der Populationsveränderung. Zunächst schreiben wir unsere Wachstumsraten in eine Liste.

```
bees = [14, 26, 16, -38, -6]
```

Nun wenden wir wieder besseres Wissen das arithmetische Mittel an:

```
# Wir verwenden die Funktion `mean()` im Paket `numpy`.
print(f"Die mittlere Änderungsrate der Bienenpopulation ist {np.mean(bees)} Prozent")
```

Die mittlere Änderungsrate der Bienenpopulation ist 2.4 Prozent

Das arithmetische Mittel zeigt, dass der Schwarm über den Zeitraum von fünf Jahren wächst! Wir sind skeptisch und berechnen das jährliche Wachstum des Bienenschwärms explizit. Zunächst wandeln wir die angegebenen Prozentsätze in relative Wachstumsraten um (`bees_growth_rel`), und dann berechnen wir einfach den Stand der Bienenpopulation, indem wir die Veränderungsraten nacheinander mit der Anzahl der Bienen multiplizieren, von der wir wissen, dass sie zu Beginn der Erhebung 5.000 betrug.

```
print("bees:      ", bees)
# Initialisiere leere Liste bees_growth_rel und Indexvariable i
bees_growth_rel = [np.round(x / 100 + 1, 2) for x in bees]
print("bees rates: ", bees_growth_rel)
```

```
bees:      [14, 26, 16, -38, -6]
bees rates: [np.float64(1.14), np.float64(1.26), np.float64(1.16), np.float64(0.62)]
```

```
pop = 5000
for rate in bees_growth_rel:
    pop = pop * rate
    print(
        f"Nach einem relativen Wachstum von {rate} erreicht die Population einen Wert von {pop:.0f}"
    )
```

```
Nach einem relativen Wachstum von 1.14 erreicht die Population einen Wert von 5700.0
Nach einem relativen Wachstum von 1.26 erreicht die Population einen Wert von 7182.0
Nach einem relativen Wachstum von 1.16 erreicht die Population einen Wert von 8331.0
Nach einem relativen Wachstum von 0.62 erreicht die Population einen Wert von 5165.0
Nach einem relativen Wachstum von 0.94 erreicht die Population einen Wert von 4855.0
```

Wow, was für eine Überraschung! Offensichtlich stimmt da etwas nicht! Wir hatten erwartet, dass der Schwarm im Laufe der Zeit im Durchschnitt zunehmen würde. Wir haben jedoch einen Rückgang der absoluten Zahl der Bienen berechnet!

Versuchen wir es mit dem geometrischen Mittel. Zum Glück gibt es in Python vordefinierte Funktionspakete für Berechnung von Streuungsmaßen. Diese können durch `import statistics` verwendet werden. Die Funktion für das geometrische Mittel heisst `geometric_mean()`

```
st.geometric_mean(bees_growth_rel)
```

```
0.9941469529781075
```

Großartig! Das geometrische Mittel zeigt, dass die Zahl der Arten im Laufe der Zeit mit einer durchschnittlichen Rate von 0,994 abnimmt, was $-0,006\%$ entspricht. Wir überprüfen dies, indem wir 5.000 Bienen (die anfängliche Anzahl der Bienen im Schwarm) mal 0,994 für jedes Jahr nehmen; dies ergibt 4.971 Bienen nach dem ersten Jahr, 4.942 nach dem zweiten Jahr, 4.913 nach dem dritten Jahr, 4.884 nach dem vierten Jahr und 4.855 nach dem fünften Jahr der Beobachtung. Eine perfekte Übereinstimmung! Im Gegensatz zum arithmetischen Mittel gibt das geometrische Mittel die Entwicklung im Vergleich zum Vorjahr nicht zu hoch an!

Der harmonische Mittelwert

Der **harmonische Mittelwert** eignet sich am besten für die Ermittlung des Durchschnitts von inversen Größen wie Geschwindigkeit (km/h) oder Bevölkerungsdichte (pop/km²).

Betrachten Sie das folgende Beispiel: Die Entfernung zwischen Ihrem Haus und dem nächsten See beträgt 40 km. Sie sind mit einer Geschwindigkeit von 20 km pro Stunde zum See gefahren und mit einer Geschwindigkeit von 80 km pro Stunde nach Hause zurückgekehrt. Wie hoch war die Durchschnittsgeschwindigkeit während der gesamten Fahrt? Berechnen wir zunächst das arithmetische Mittel.

```
# Initialisiere Liste speed_arithmetic
speed_arithmetic = [20, 80]

print(np.mean(speed_arithmetic))
```

```
50.0
```

Das arithmetische Mittel der beiden Geschwindigkeiten, mit denen Sie gefahren sind, beträgt 50 km pro Stunde. Dies ist jedoch nicht die richtige Durchschnittsgeschwindigkeit. Es lässt die Tatsache außer Acht, dass Sie mit 20 km/h viel länger gefahren sind als mit 80 km/h. Um die richtige

Durchschnittsgeschwindigkeit zu ermitteln, müssen wir stattdessen das harmonische Mittel berechnen.

Das harmonische Mittel \bar{x}_h für die positiven reellen Zahlen x_1, x_2, \dots, x_n ist definiert durch

$$\bar{x}_h = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}} \quad x_i > 0, \quad \forall i$$

Wir können die vordefinierte Funktion `harmonic_mean` aus dem Packet `statistics` verwenden:

```
speed = [20, 80]
st.harmonic_mean(speed)
```

```
32.0
```

Aber ist dieses Ergebnis korrekt? Wir versuchen das Ergebnis nachzuvollziehen. Im obigen Beispiel beträgt die Entfernung zwischen dem See und Ihrem Haus 40 km. Die Fahrt von A nach B mit einer Geschwindigkeit von 20 km/h dauert also 2 Stunden. Die Fahrt von B nach A mit einer Geschwindigkeit von 80 km/h dauert 0,5 Stunden. Die Gesamtzeit für die Rundstrecke von 2×40 km beträgt 2,5 Stunden. Die Durchschnittsgeschwindigkeit beträgt dann $\frac{80}{2,5} = 32$.

Der gewichtete Mittelwert

Es gibt Anwendungen, bei denen bestimmte Werte in einem Datensatz als wichtiger angesehen werden können als andere. Im Allgemeinen gilt für eine Folge von n Datenwerten x_1, x_2, \dots, x_n und ihren entsprechenden Gewichten w_1, w_2, \dots, w_n , ist das gewichtete (arithmetische) Mittel \bar{x}_w gegeben durch

$$\bar{x}_w = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$$

wobei $\sum_{i=1}^n w_i x_i$ durch Multiplikation jedes Datenwertes mit seinem Gewicht und anschließender Addition der Produkte ermittelt wird.

Um die Noten der Studenten in einem Kurs zu ermitteln, kann ein Dozent beispielsweise der

Abschlussprüfung eine dreimal so hohe Gewichtung zuweisen wie den anderen Prüfungen. Ermitteln wir den gewichteten Mittelwert für einen Studenten, der in den ersten beiden Prüfungen 45 und 68 Punkte und in der Abschlussprüfung 74 Punkte erzielt hat.

Wir berechnen das gewichtete Mittel indem wir die Funktion `np.average()` anwenden.

```
# Initialisiere Liste scores, weights_scores und leere Liste weighted
scores = [45, 68, 74]
weights_scores = [1, 1, 3]
np.average(scores, weights=weights_scores)
```

```
np.float64(67.0)
```

Verglichen mit dem nicht gewichteten Mittelwert ergibt sich ein anderes Ergebnis.

```
np.mean(scores)
```

```
np.float64(62.33333333333336)
```

Bitte beachten Sie, dass die Gewichtung der Eingabewerte ein Prinzip ist, das auch auf andere Mittelwertmaße anwendbar ist. Zum Beispiel können wir die Eingangsvariable für die Berechnung des geometrischen Mittels gewichten.

Der gewichtete geometrische Mittelwert

$$\bar{x}_{geo_w} = \left(\prod_{i=1}^n x_i^{w_i} \right)^{1/\sum_{i=1}^n w_i}$$

wobei x_1, x_2, \dots, x_n den Datenwerten und w_1, w_2, \dots, w_n den Gewichten entsprechen.

Wir greifen auf die Implementierung des gewichteten geometrischen Mittelwerts im [Scipy](#) Modul zurück.

```
from scipy.stats import gmean
```

Wir berechnen das Beispiel des Bienenschwärms aus dem obigen Abschnitt erneut. Wir erinnern uns, dass wir im obigen Beispiel einen Bienenschwarm über 5 Jahre beobachtet und die Veränderungsraten der Bienenpopulation notiert haben. Die jährlichen Veränderungsraten waren 1, 14, 1, 26, 1, 16, 0, 62, 0, 94, was x_1, x_2, \dots, x_n entspricht. Um das Ergebnis von oben zu reproduzieren, setzen wir alle Gewichte w_1, w_2, \dots, w_n auf 1.

```
weights_bees = [1, 1, 1, 1, 1]
gmean(bees_growth_rel, weights=weights_bees)
```

```
np.float64(0.9941469529781075)
```

```
gmean(bees_growth_rel, weights=weights_bees) == gmean(bees_growth_rel)
```

```
np.True_
```

Korrekt, wir erhalten das gleiche Ergebnis! Falls wir nun jedes konsekutive Element mehr gewichten wollen, können wir die Gewichte auf z.B. [1, 2, 3, 4] setzen.

```
weights_bees = [1, 2, 3, 4, 5]
gmean(bees_growth_rel, weights=weights_bees)
```

```
np.float64(0.9241626141253284)
```

Der gewichtete harmonische Mittelwert

Schließlich werden wir eine eigene gewichtete harmonische Mittelwertfunktion implementieren.

Das gewichtete harmonische Mittel \bar{x} für die positiven reellen Zahlen x_1, x_2, \dots, x_n ist durch die folgende Gleichung definiert:

$$\bar{x}_{hw} = \frac{w_1 + w_2 + \dots + w_n}{\frac{w_1}{x_1} + \frac{w_2}{x_2} + \dots + \frac{w_n}{x_n}} = \frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n \frac{w_i}{x_i}}, \quad x_i > 0, \quad \forall i$$

Implementieren wir die Funktion des gewichteten harmonischen Mittels in Python. Die Kodierung ist einfach, aber um die Funktionalität unserer Funktion des gewichteten harmonischen Mittels zu erweitern, fügen wir eine `if`-Anweisung ein. Diese `if`-Anweisung im Code normalisiert die Gewichte, wenn die Gewichte nicht in Proportionen angegeben sind. Wenn die `if`-Anweisung ausgeführt wird, geben wir ein Benutzer-Feedback aus, andernfalls erfolgt keine Rückmeldung.

```
# Definiere Funktion my_weighted_harmonic_mean
def my_weighted_harmonic_mean1(data: np.array, weights: np.array):
    if np.sum(weights) != 1:
        print("Die Gewichte werden normalisiert")
        weights = weights / np.sum(weights)
    return np.sum(weights) / sum(weights / data)
```

Lassen Sie uns unser `my_weighted_harmonic_mean` an einem recht komplexen Datensatz ausprobieren. Der Datensatz `cities` besteht aus allen Landeshauptstädten Deutschlands, ihrer Bevölkerungszahl und ihrer Fläche. Ziel ist es, die mittlere Bevölkerungsdichte für die Landeshauptstädte Deutschlands zu berechnen. Sie können die Datei `cities.csv` [hier](#) herunterladen. Die Daten werden von dieser [Website](#) abgerufen.

Zunächst laden wir den Datensatz, geben ihm einen passenden Namen und schauen ihn uns an.

```
df = pd.read_csv("../data/cities.csv")
df
```

	name	state	area_km2	pop.size
0	Berlin	Land Berlin	891.85	3415100
1	Bremen	Freie Hansestadt Bremen	325.42	546450
2	Dresden	Freistaat Sachsen	328.31	525100
3	Düsseldorf	Land Nordrhein-Westfalen	217.41	593682
4	Erfurt	Freistaat Thüringen	269.17	203480
5	Hamburg	Freie und Hansestadt Hamburg	755.26	1751780
6	Hannover	Land Niedersachsen	204.14	514130
7	Kiel	Land Schleswig-Holstein	118.60	239860
8	Magdeburg	Land Sachsen-Anhalt	200.97	229924

	name	state	area_km2	pop.size
9	Mainz	Land Rheinland-Pfalz	97.76	202750
10	München	Freistaat Bayern	310.71	1388300
11	Potsdam	Land Brandenburg	187.27	159450
12	Saarbrücken	Saarland	167.07	176990
13	Schwerin	Land Mecklenburg-Vorpommern	130.46	91260
14	Stuttgart	Land Baden-Württemberg	207.36	597939
15	Wiesbaden	Land Hessen	203.90	272630

Zweitens erstellen wir eine neue Spalte und berechnen die Bevölkerungsdichte (Einwohner pro Quadratkilometer) und speichern das Ergebniss in der Spalte `density`.

```
df["density"] = df["pop.size"] / df["area_km2"]
df
```

	name	state	area_km2	pop.size	density
0	Berlin	Land Berlin	891.85	3415100	3829.231373
1	Bremen	Freie Hansestadt Bremen	325.42	546450	1679.214554
2	Dresden	Freistaat Sachsen	328.31	525100	1599.403003
3	Düsseldorf	Land Nordrhein-Westfalen	217.41	593682	2730.702360
4	Erfurt	Freistaat Thüringen	269.17	203480	755.953487
5	Hamburg	Freie und Hansestadt Hamburg	755.26	1751780	2319.439663
6	Hannover	Land Niedersachsen	204.14	514130	2518.516704
7	Kiel	Land Schleswig-Holstein	118.60	239860	2022.428331
8	Magdeburg	Land Sachsen-Anhalt	200.97	229924	1144.071254
9	Mainz	Land Rheinland-Pfalz	97.76	202750	2073.956628
10	München	Freistaat Bayern	310.71	1388300	4468.153584
11	Potsdam	Land Brandenburg	187.27	159450	851.444439
12	Saarbrücken	Saarland	167.07	176990	1059.376309

	name		state	area_km2	pop.size	density
13	Schwerin	Land Mecklenburg-Vorpommern		130.46	91260	699.524759
14	Stuttgart	Land Baden-Württemberg		207.36	597939	2883.579282
15	Wiesbaden	Land Hessen		203.90	272630	1337.076999

Drittens berechnen wir das Gewicht für jede Stadt entsprechend der Bevölkerungszahl und speichern das Ergebniss in der Spalte `cities_pop_weight`.

```
df["cities_pop_weight"] = df["pop.size"] / df["pop.size"].sum()
df
```

	name	state	area_km2	pop.size	density	cities_pop_weight
0	Berlin	Land Berlin	891.85	3415100	3829.231373	0.313058
1	Bremen	Freie Hansestadt Bremen	325.42	546450	1679.214554	0.050092
2	Dresden	Freistaat Sachsen	328.31	525100	1599.403003	0.048135
3	Düsseldorf	Land Nordrhein-Westfalen	217.41	593682	2730.702360	0.054422
4	Erfurt	Freistaat Thüringen	269.17	203480	755.953487	0.018653
5	Hamburg	Freie und Hansestadt Hamburg	755.26	1751780	2319.439663	0.160584
6	Hannover	Land Niedersachsen	204.14	514130	2518.516704	0.047130
7	Kiel	Land Schleswig-Holstein	118.60	239860	2022.428331	0.021988
8	Magdeburg	Land Sachsen-Anhalt	200.97	229924	1144.071254	0.021077
9	Mainz	Land Rheinland-Pfalz	97.76	202750	2073.956628	0.018586
10	München	Freistaat Bayern	310.71	1388300	4468.153584	0.127264
11	Potsdam	Land Brandenburg	187.27	159450	851.444439	0.014617
12	Saarbrücken	Saarland	167.07	176990	1059.376309	0.016224
13	Schwerin	Land Mecklenburg-Vorpommern	130.46	91260	699.524759	0.008366
14	Stuttgart	Land Baden-Württemberg	207.36	597939	2883.579282	0.054812
15	Wiesbaden	Land Hessen	203.90	272630	1337.076999	0.024992

Nun wenden wir unsere Implementierung des gewichteten harmonischen Mittels

`my_weighted_harmonic_mean` an und vergleichen es mit dem arithmetischen Mittel.

```
# Wir verwenden die Spalte "cities_pop_weight" als Gewichte. Hier ist keine Normalisierung erforderlich
my_weighted_harmonic_mean1(df["density"], df["cities_pop_weight"])
```

```
np.float64(2363.437731548685)
```

Jetzt testen wir die Funktionalität unserer Funktion, indem wir `pop.size` als Eingabeparameter angeben.

```
# Wir verwenden die Spalte "pop.size" als Gewichte. Ein Normalisierung ist notwendig
my_weighted_harmonic_mean1(df["density"], df["pop.size"])
```

Die Gewichte werden normalisiert

```
np.float64(2363.437731548685)
```

Wunderbar, die Funktion funktioniert wie erwartet. Die Ergebnisse sind identisch. Wir können daraus schließen, dass die durchschnittliche Bevölkerungsdichte in den Landeshauptstädten Deutschlands etwa 2.363 Einwohner/km² beträgt.

Zum Vergleich können wir das arithmetische Mittel der Bevölkerungsdichte berechnen. Dieses würde uns hier ein weniger exaktes Ergebnis liefern.

```
df["density"].mean()
```

```
np.float64(1998.2545454902418)
```

Der Median

Ein weiteres sehr wichtiges Maß für die zentrale Tendenz ist der **Median**. Der Median von Messwerten ist der Wert des mittleren Terms in einem Datensatz, der in aufsteigender Reihenfolge geordnet wurde. Der Median unterteilt also einen geordneten Datensatz **in zwei gleiche Teile**. Die Berechnung des Medians besteht aus den folgenden zwei Schritten:

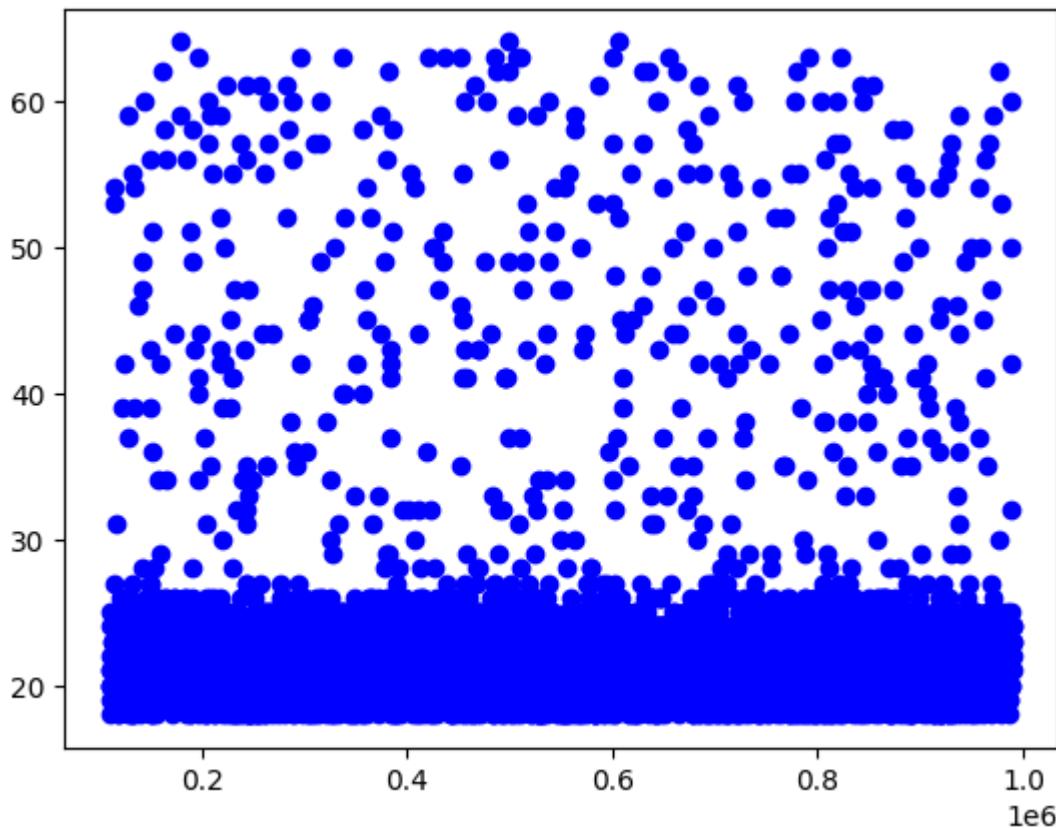
1. Ordnen Sie den Datensatz in aufsteigender Reihenfolge.
2. Finden Sie den mittleren Term. Der Wert dieses Terms ist der Median.

Wenn die Anzahl der Beobachtungen in einem Datensatz ungerade ist, wird der Median durch den Wert des mittleren Terms in den geordneten Daten bestimmt. Ist die Anzahl der Beobachtungen jedoch gerade, so wird der Median durch den Durchschnitt der Werte der beiden mittleren Terme bestimmt (s.52).

Lassen Sie uns den Median für die Variable `age` des `students` Datensatzes auswerten.

```
df = pd.read_csv("../data/students.csv", index_col=0)
plt.plot(df.index, df.age, "bo")
```

```
[<matplotlib.lines.Line2D at 0x7f35309ca800>]
```



Wenn wir die Variable `age` plotten, erkennen wir sofort, dass es einige Studenten gibt, die viel älter sind als der Rest der Studenten. Lassen Sie uns den Median berechnen...

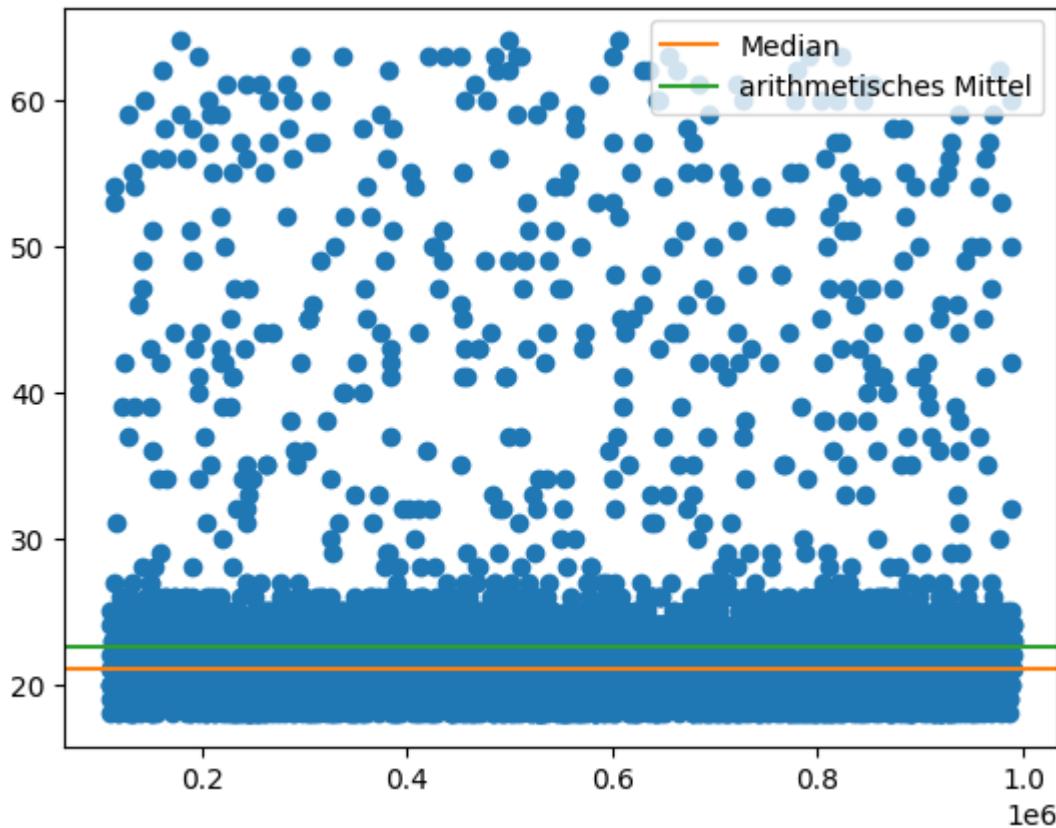
```
print(f"Median des Alters der Studenten: {np.median(df.age)}")
print(f"Mittelwert des Alters der Studenten: {np.mean(df.age)}")
```

```
Median des Alters der Studenten: 21.0
Mittelwert des Alters der Studenten: 22.541570578953756
```

Zur Veranschaulichung fügen wir nun den Median und das arithmetische Mittel in das Streudiagramm ein.

```
plt.plot(df.index, df.age, "o", color="C0")
plt.axhline(np.median(df.age), label="Median", color="C1")
plt.axhline(np.mean(df.age), label="arithmetisches Mittel", color="C2")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f35308adae0>
```

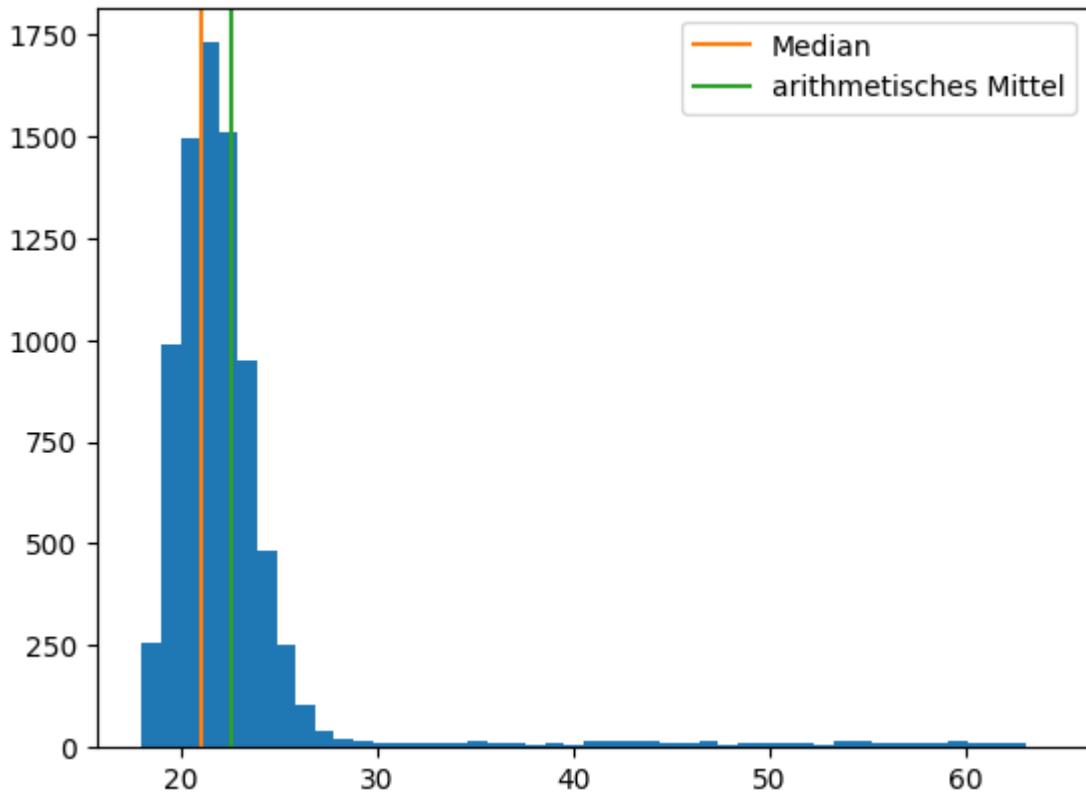


Noch deutlicher veranschaulicht ein Histogramm den Sachverhalt, dass das arithmetische Mittel stark durch Extremwerte und Ausreisser beeinflusst wird, der Median hingegen nicht. Daher wird der

Median als Maß für die zentrale Tendenz bei Datensätzen, die Ausreißer enthalten, dem Mittelwert vorgezogen ([]) s.54).

```
plt.hist(df.age, bins=df.age.unique())
plt.axvline(np.median(df.age), label="Median", color="C1")
plt.axvline(np.mean(df.age), label="arithmetisches Mittel", color="C2")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f3530968fa0>
```

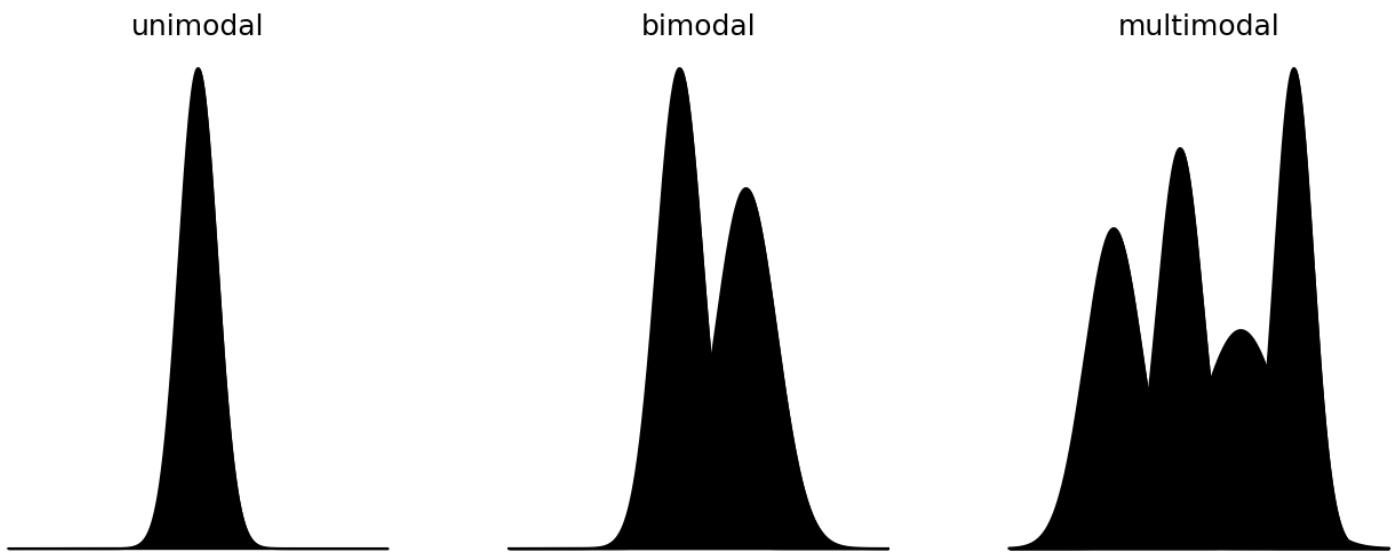


Der Modalwert

In der Statistik stellt der **Modus** den häufigsten Wert in einem Datensatz dar. Der Modalwert ist also der Wert, der in einem Datensatz mit der größten Häufigkeit auftritt ([]) s. 53). In Bezug auf die graphische Häufigkeitsverteilung entspricht der Modus dem Scheitelpunkt bzw. den Scheitelpunkten des Diagramms. Ein großes Manko des Modus ist, dass ein Datensatz keinen oder mehr als einen Modalwert haben kann, während er nur einen Mittelwert und nur einen Median hat. Ein Datensatz, bei dem jeder Wert nur einmal vorkommt, hat zum Beispiel keinen Modus. Ein Datensatz, bei dem nur ein Wert mit der größten Häufigkeit vorkommt, hat nur einen Modus. Der Datensatz wird in diesem Fall als **unimodal** bezeichnet. Ein Datensatz mit zwei Werten, die am häufigsten vorkommen, hat

zwei Modi. Die Verteilung wird in diesem Fall als **bimodal** bezeichnet. Wenn mehr als zwei Werte in einem Datensatz am häufigsten vorkommen, dann enthält der Datensatz mehr als zwei Modi und wird als **multimodal** bezeichnet ([] s. 81,83).

► Show code cell source



Anders als der Mittelwert und der Median kann der Modus für quantitative (numerische) und qualitative (kategoriale) Daten verwendet werden. Python verfügt über eine vordefinierte Funktion `mode()` zur Berechnung des Modus im Paket `statistics`. Wir verwenden diese um den Modalwert aller Spaltenwerte des `students` Datensatzes zu berechnen.

```
df = pd.read_csv("../data/students.csv")
for col in ["gender", "age", "religion", "nc_score", "semester", "height", "weight"]
    print(f"Der Modalwert der Spalte '{col}' ist '{st.mode(df[col])}'")
```

```
Der Modalwert der Spalte 'gender' ist 'Male'.
Der Modalwert der Spalte 'age' ist '21'.
Der Modalwert der Spalte 'religion' ist 'Catholic'.
Der Modalwert der Spalte 'nc_score' ist '1.18'.
Der Modalwert der Spalte 'semester' ist '1st'.
Der Modalwert der Spalte 'height' ist '174'.
Der Modalwert der Spalte 'weight' ist '67.1'.
```

Streuungsmaße

```
import random

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm, gaussian_kde
```

Die Maße der zentralen Tendenz, wie Mittelwert, Median und Modus, geben nicht das ganze Bild der Verteilung eines Datensatzes wieder. Zwei Datensätze mit identischem Mittelwert können völlig unterschiedliche Streuungen aufweisen. Die Streuung der Beobachtungswerte des einen Datensatzes kann viel größer oder kleiner sein als die des anderen Datensatzes. Daher ist der Mittelwert, Median oder Modus allein in der Regel kein ausreichendes Maß, um die Form der Verteilung eines Datensatzes aufzuzeigen. Wir benötigen auch ein Maß, das Informationen über die Variation zwischen den Datenwerten liefert. Diese Maße werden als **Streuungsmaße** bezeichnet. Die Maße der zentralen Tendenz und der Streuung ergeben zusammen genommen ein besseres Bild eines Datensatzes als die Maße der zentralen Tendenz allein (s.65).

Varianz und Standardabweichung

Die **Varianz** ist die Summe der quadrierten Abweichungen vom Mittelwert. Die Varianz für Populationsdaten wird mit σ^2 bezeichnet (gelesen als Sigma-Quadrat), und die für Stichprobendaten berechnete Varianz wird mit s^2 bezeichnet.

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \mu)^2}{N}$$

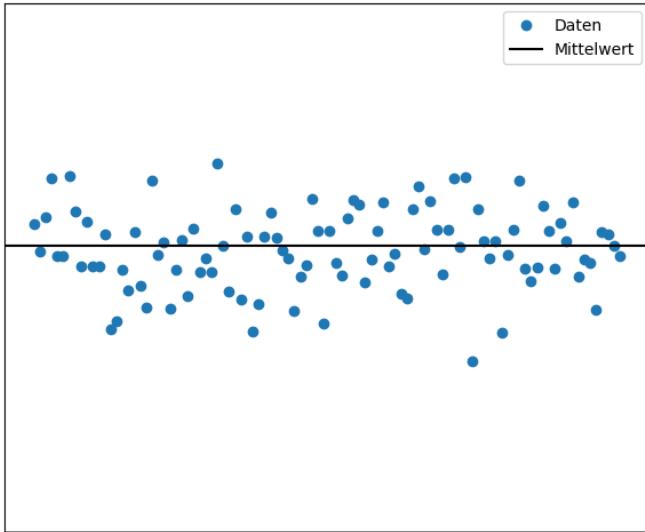
$$\text{und } s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$$

wobei σ^2 die Varianz der Grundgesamtheit und s^2 die Stichprobenvarianz ist. Die Größe $x_i - \mu$ oder $x_i - \bar{x}$ in den obigen Formeln wird als die Abweichung des x_i -Wertes (x_1, x_2, \dots, x_n) vom Mittelwert bezeichnet (s.64).

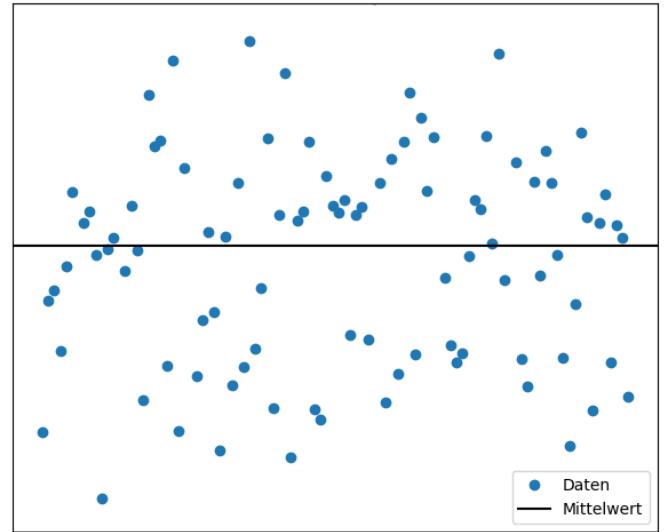
Die [Standardabweichung](#) ist das gebräuchlichste Maß für die Streuung. Der Wert der Standardabweichung gibt an, wie eng die Werte eines Datensatzes um den Mittelwert herum gestreut sind. Im Allgemeinen zeigt ein niedriger Wert der Standardabweichung für einen Datensatz an, dass die Werte dieses Datensatzes über einen relativ kleineren Bereich um den Mittelwert herum verteilt sind. Im Gegensatz dazu zeigt ein größerer Wert der Standardabweichung für einen Datensatz an, dass die Werte dieses Datensatzes über einen relativ größeren Bereich um den Mittelwert herum gestreut sind ([] s.65).

▶ Show code cell source

Geringe Abweichungen um den Mittelwert



Grosse Abweichungen um den Mittelwert



Die Standardabweichung erhält man durch Ziehen der Quadratwurzel aus der **Varianz**. Folglich wird die für Grundgesamtheitsdaten berechnete Standardabweichung mit σ und die für Stichprobendaten berechnete Standardabweichung mit s bezeichnet.

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)^2}{N}}$$

und

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

wobei σ die Standardabweichung der Grundgesamtheit und s die Standardabweichung der Stichprobe ist.

Als Übung berechnen wir für einige numerische Variablen, die im `students` Datensatz von Interesse sind, den Mittelwert, den Median, die Varianz und die Standardabweichung und stellen sie in einem Dataframe dar.

```
# Lesen der Datei students.csv als Dataframe; nur die Spalten "age", "weight", "height"
students = pd.read_csv(
    "../../data/students.csv",
    index_col=0,
    usecols=["age", "height", "weight", "nc_score"],
)
# Zeige die ersten 10 Werte
students.head(10)
```

	height	weight	nc_score
age			
19	160	64.8	1.91
19	172	73.0	1.56
22	168	70.6	1.24
19	183	79.7	1.37
21	175	71.4	1.46
19	189	85.8	1.34
21	156	65.9	1.11
21	167	65.7	2.03
18	195	94.4	1.29
18	165	66.0	1.19

Pandas verfügt über die Methode `agg`. Diese lässt uns sehr einfach verschiedene deskriptive Statistiken berechnen.

```
students.agg(["mean", "median", "var", "std"])
```

	height	weight	nc_score
mean	171.380750	72.998131	2.166481
median	171.000000	71.800000	2.040000
var	122.711652	74.566017	0.658610
std	11.077529	8.635162	0.811548

Verwendung der Standardabweichung

Mit Hilfe des **Mittelwerts** und der **Standardabweichung** lässt sich der Anteil oder Prozentsatz der Gesamtbeobachtungen ermitteln, die in ein bestimmtes Intervall um den Mittelwert fallen.

Tschebyscheff-Theorem

Die [Tschebyscheff Ungleichung](#) gibt eine untere Schranke für die Fläche unter einer Kurve zwischen zwei Punkten, die auf gegenüberliegenden Seiten des Mittelwerts und im gleichen Abstand vom Mittelwert liegen.

Für jede Zahl k , die größer als 1 ist, liegen mindestens $1 - \frac{1}{k^2}$ der Datenwerte innerhalb von k Standardabweichungen vom Mittelwert.

Lassen Sie uns Python verwenden, um ein Gefühl für den Tschebyscheff-Theorem zu bekommen.

```

k = np.arange(1, 4.1, 0.1)
value = np.round((1 - (1 / k**2)) * 100)
chebyshev = pd.DataFrame({"k": k, "Prozent": value})
chebyshev

```

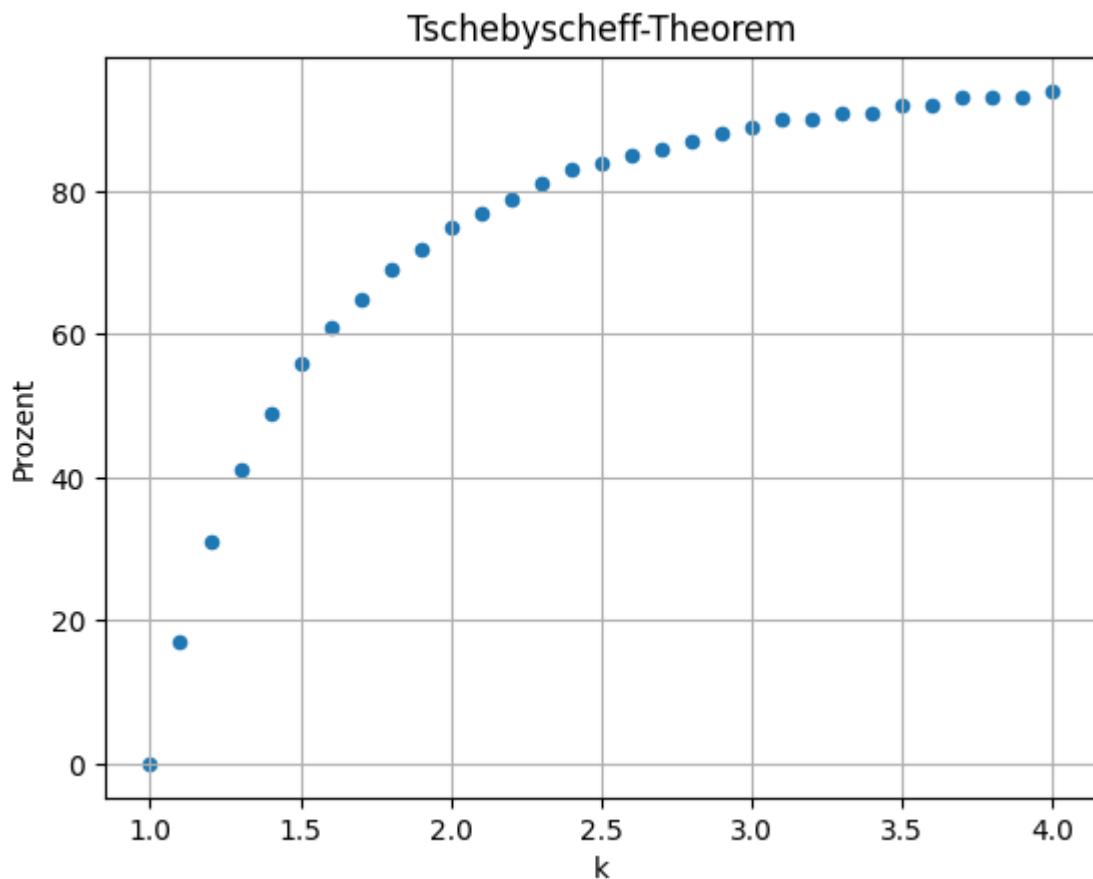
	k	Prozent
	0	1.0
	1	17.0
	2	31.0
	3	41.0
	4	49.0
	5	56.0
	6	61.0
	7	65.0
	8	69.0
	9	72.0
	10	75.0
	11	77.0
	12	79.0
	13	81.0
	14	83.0
	15	84.0
	16	85.0
	17	86.0
	18	87.0
	19	88.0
	20	89.0
	21	90.0
	22	90.0
	23	91.0
	24	91.0
	25	92.0
	26	92.0

	k	Prozent
27	3.7	93.0
28	3.8	93.0
29	3.9	93.0
30	4.0	94.0

Um es in Worte zu fassen: Für $k = 2$ bedeutet das, dass mindestens 75% der Datenwerte innerhalb von **2 Standardabweichungen** vom Mittelwert liegen.

Stellen wir das Tschebyscheff-Theorem mit Python dar:

```
fig, ax = plt.subplots()
chebyshev.plot.scatter(x="k", y="Prozent", ax=ax)
ax.set_title("Tschebyscheff-Theorem")
ax.grid()
```



Das Theorem gilt sowohl für Stichproben- als auch für Grundgesamtheitsdaten. Die Tschebyscheffsche Ungleichung gilt für Verteilungen beliebiger Form. Sie kann jedoch nur für $k > 1$

verwendet werden. Denn wenn $k = 1$ ist, ist der Wert von $1 - \frac{1}{k^2}$ Null, und wenn $k < 1$ ist, ist der Wert von $1 - \frac{1}{k^2}$ negativ (siehe s.304).

Empirische Regel

Während die Tschebyscheffsche Ungleichung auf jede Art von Verteilung anwendbar ist, gilt die **empirische Regel** nur für eine bestimmte Art von Verteilung, die so genannte **Gaußverteilung** oder **Normalverteilung**. Es gibt 3 Regeln:

Bei einer Normalverteilung sind

1. 68% der Beobachtungen innerhalb einer Standardabweichung des Mittelwerts.
2. 95% der Beobachtungen innerhalb von zwei Standardabweichungen des Mittelwerts.
3. 99,7% der Beobachtungen innerhalb von drei Standardabweichungen des Mittelwerts.

Da wir inzwischen über genügend Hacking-Power verfügen, werden wir versuchen zu testen, ob die drei Regeln gültig sind.

(1) Erstens werden wir die Funktion `random.normal()` in Python erforschen, um normalverteilte Daten zu erzeugen, und

(2) Zweitens werden wir zu unserem `students` Datensatz zurückkehren um diese Regeln an diesem Datensatz zu validieren.

Die Normalverteilung gehört zur Familie der [stetigen Verteilungen](#). In Python gibt es eine Vielzahl von Wahrscheinlichkeitsverteilungen ([hier](#)). Um Daten aus einer Normalverteilung zu erzeugen, kann man die Funktion `random.normal()` verwenden, die ein Zufallsvariablen generator für die Normalverteilung ist.

Mit der Funktion `np.random.normal(loc=0.0, scale=1.0)` können wir Zufallsvariablen aus einer Normalverteilung mit einem gegebenen Mittelwert (Standard ist 0) und einer Standardabweichung (Standard ist 1) entnehmen. Mit dem Argument `size` können wir die Anzahl der erzeugten Zufallsvariablen bestimmen.

```
np.random.normal(loc=0.0, scale=1.0, size=1)
```

```
array([0.35778736])
```

```
np.random.normal(loc=0.0, scale=1.0, size=1)
```

```
array([0.56078453])
```

```
np.random.normal(loc=0.0, scale=1.0, size=1)
```

```
array([1.08305124])
```

Wir können die Funktion ziemlich einfach bitten, hunderte oder tausende oder noch mehr (Pseudo-)Zufallszahlen zu ziehen:

```
np.random.normal(loc=0.0, scale=1.0, size=10)
```

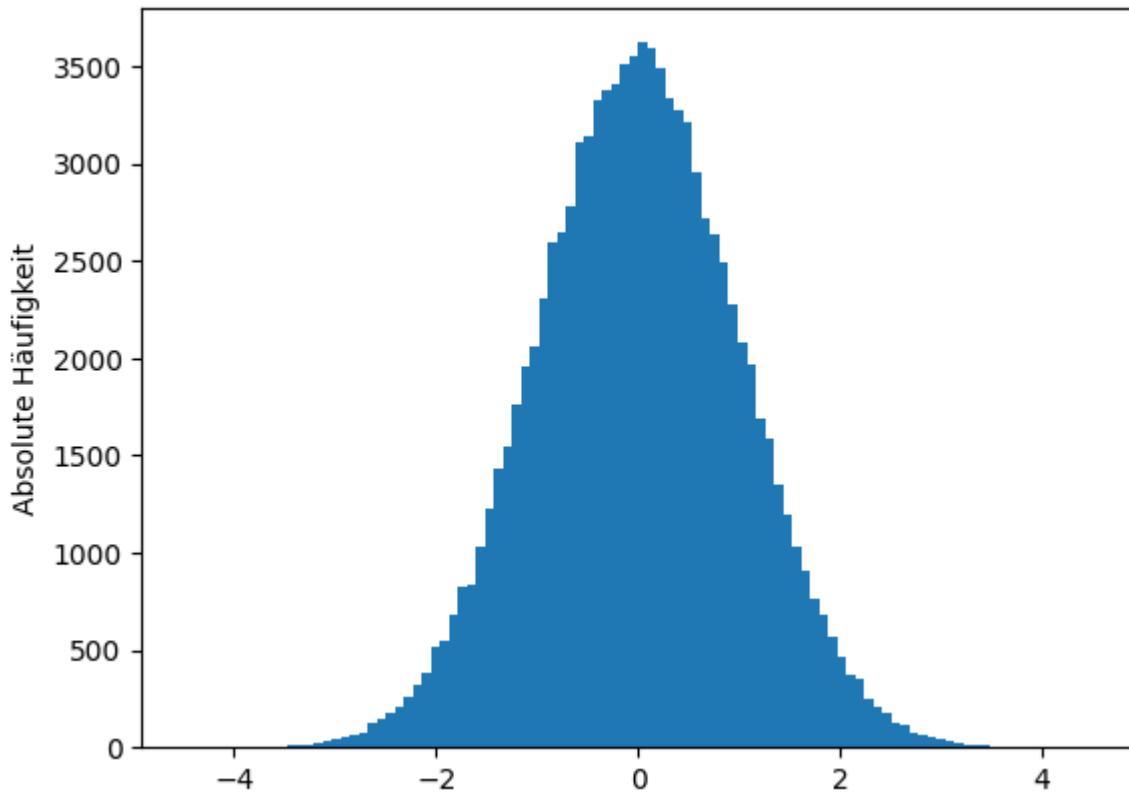
```
array([ 1.05380205, -1.37766937, -0.93782504,  0.51503527,  0.51378595,
       0.51504769,  3.85273149,  0.57089051,  1.13556564,  0.95400176])
```

```
np.random.normal(loc=0.0, scale=1.0, size=100)
```

```
array([ 0.65139125, -0.31526924,  0.75896922, -0.77282521, -0.23681861,
       -0.48536355,  0.08187414,  2.31465857, -1.86726519,  0.68626019,
      -1.61271587, -0.47193187,  1.0889506 ,  0.06428002, -1.07774478,
      -0.71530371,  0.67959775, -0.73036663,  0.21645859,  0.04557184,
      -0.65160035,  2.14394409,  0.63391902, -2.02514259,  0.18645431,
      -0.66178646,  0.85243333, -0.79252074, -0.11473644,  0.50498728,
      0.86575519, -1.20029641, -0.33450124, -0.47494531, -0.65332923,
      1.76545424,  0.40498171, -1.26088395,  0.91786195,  2.1221562 ,
      1.03246526, -1.51936997, -0.48423407,  1.26691115, -0.70766947,
      0.44381943,  0.77463405, -0.92693047, -0.05952536, -3.24126734,
      -1.02438764, -0.25256815, -1.24778318,  1.6324113 , -1.43014138,
      -0.44004449,  0.13074058,  1.44127329, -1.43586215,  1.16316375,
      0.01023306, -0.98150865,  0.46210347,  0.1990597 , -0.60021688,
      0.06980208, -0.3853136 ,  0.11351735,  0.66213067,  1.58601682,
      -1.2378155 ,  2.13303337, -1.9520878 , -0.1517851 ,  0.58831721,
      0.28099187, -0.62269952, -0.20812225, -0.49300093, -0.58936476,
      0.8496021 ,  0.35701549, -0.6929096 ,  0.89959988,  0.30729952,
      0.81286212,  0.62962884, -0.82899501, -0.56018104,  0.74729361,
      0.61037027, -0.02090159,  0.11732738,  1.2776649 , -0.59157139,
      0.54709738, -0.20219265, -0.2176812 ,  1.09877685,  0.82541635])
```

Wenn wir ein Histogramm dieser Zahlen erstellen, sehen wir die namensgebende glockenförmige Verteilung.

```
y_norm = np.random.normal(loc=0.0, scale=1.0, size=100000)
bins = int(len(y_norm) / 1000) # bestimmt die Anzahl der Bins
plt.hist(y_norm, bins=bins)
plt.ylabel("Absolute Häufigkeit")
plt.show()
```



Wir kennen bereits den Mittelwert und die Standardabweichung der Werte in `y_norm`, da wir die Funktion `np.random.normal()` explizit mit `mean=0` und `sd=1` aufgerufen haben. Wir müssen also nur die Zahlen in `y_norm` zählen, die größer als -1 bzw. kleiner als 1 und 2 bzw. -2 und 3 bzw. -3 sind, und sie zur Länge von `y_norm`, in unserem Fall 100.000 , in Beziehung setzen, um die drei oben genannten Regeln zu bestätigen.

```
# Berechne Anzahl der Werte < 1 - Anzahl der Werte > -1 durch Gesamtanzahl
sd1 = sum((y_norm > -1) & (y_norm < 1)) / len(y_norm)

# Berechne Anzahl der Werte < 2 - Anzahl der Werte > -2 durch Gesamtanzahl
sd2 = sum((y_norm > -2) & (y_norm < 2)) / len(y_norm)

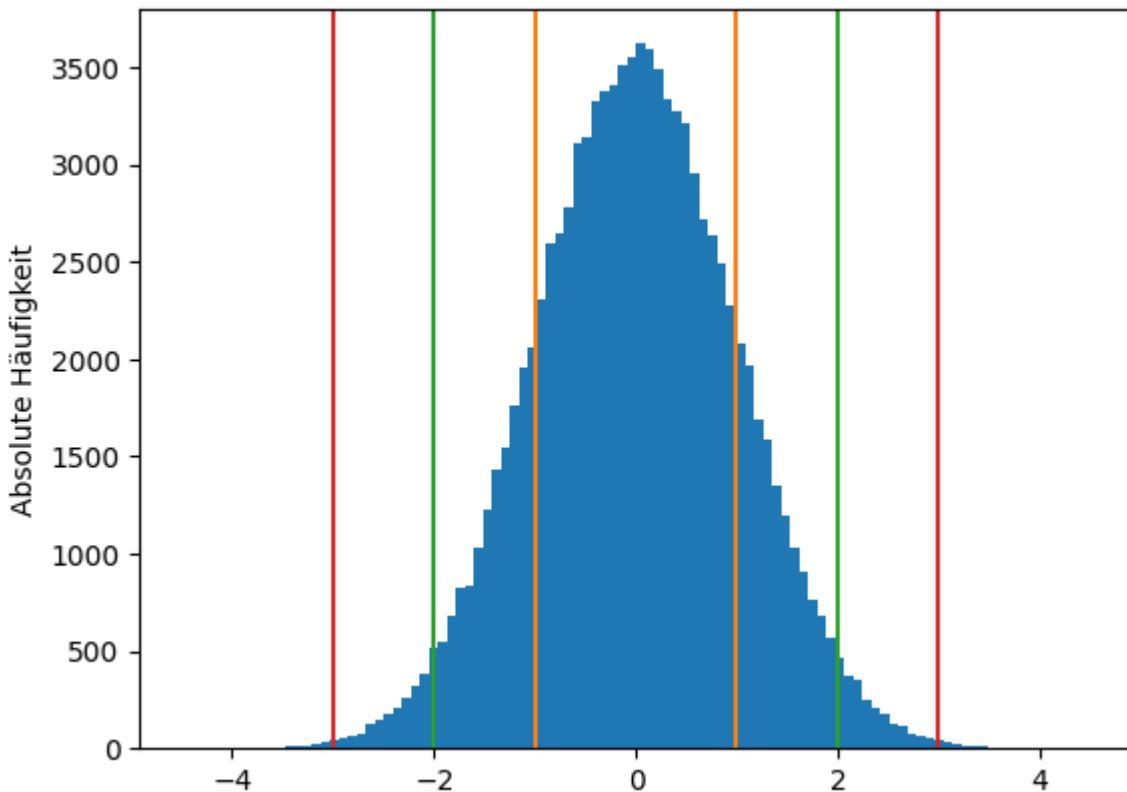
# Berechne Anzahl der Werte < 3 - Anzahl der Werte > -3 durch Gesamtanzahl
sd3 = sum((y_norm > -3) & (y_norm < 3)) / len(y_norm)

print("sd1 :", sd1)
print("sd2 :", sd2)
print("sd3 :", sd3)
```

```
sd1 : 0.68143
sd2 : 0.95444
sd3 : 0.99737
```

Perfekte Übereinstimmung! Die drei empirischen Regeln sind offensichtlich gültig. Um unsere Ergebnisse zu veranschaulichen, stellen wir das Histogramm erneut dar und fügen einige Anmerkungen hinzu. Bitte beachten Sie, dass wir in der `hist()`-Funktion das Argument `density=True` setzen. Dies hat zur Folge, dass das resultierende Histogramm nicht mehr die Zählungen auf der y-Achse anzeigt, sondern die **Dichtewerte** (normalisierte Zählung geteilt durch Bin-Breite), was bedeutet, dass sich die Balkenbereiche zu 1 summieren.

```
fig, ax = plt.subplots()
ax.hist(y_norm, bins=bins)
ax.set_ylabel("Absolute Häufigkeit")
for e, std in enumerate([(-1, 1), (-2, 2), (-3, 3)]):
    plt.axvline(std[0], color=f"C{e+1}")
    plt.axvline(std[1], color=f"C{e+1}")
```



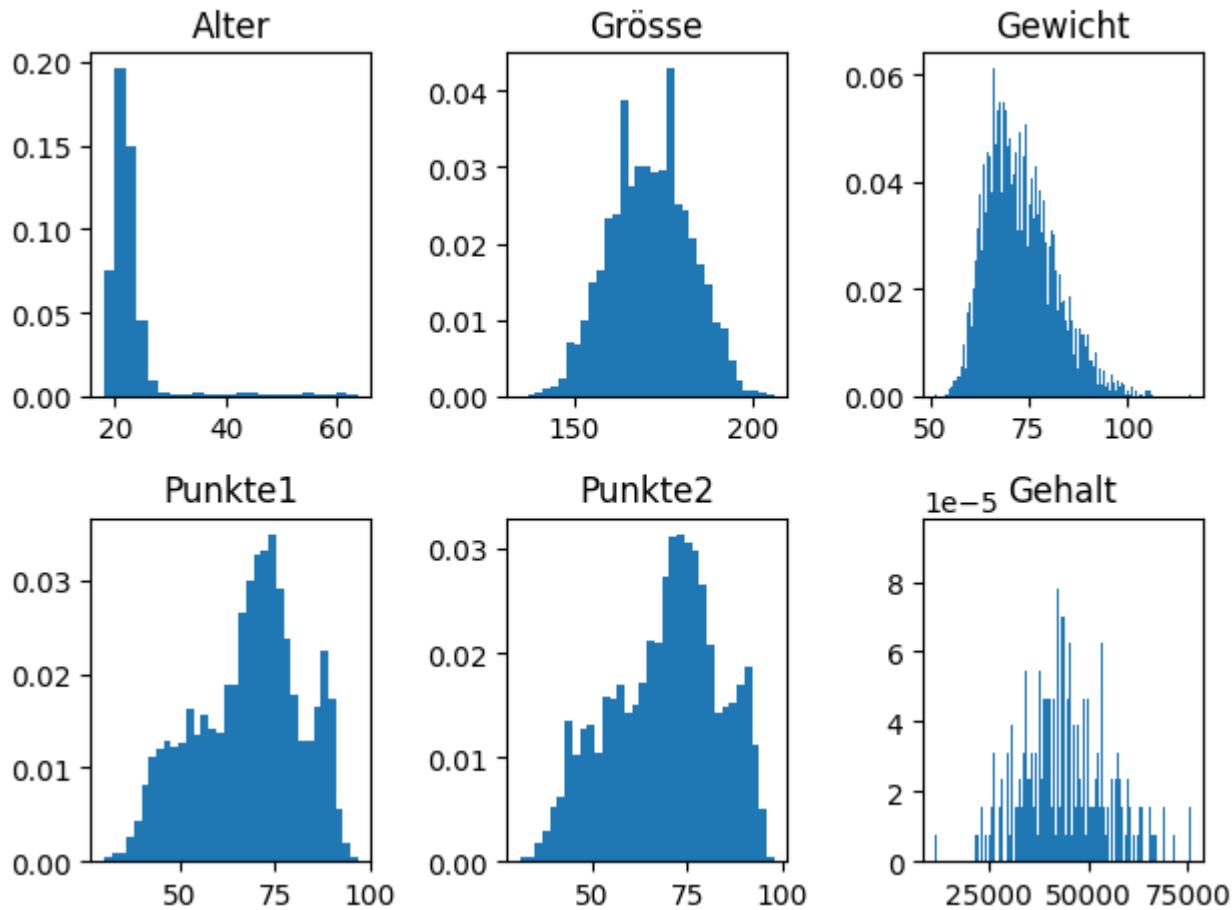
Nun, lassen Sie uns an der **zweiten** Aufgabe arbeiten: Überprüfen Sie die drei empirischen Regeln anhand des `students` Datensatzes. Dazu müssen wir überprüfen, ob eine der numerischen Variablen im Studentendatensatz normalverteilt ist. Wir beginnen mit der Extraktion numerischer Variablen von Interesse aus dem `students` Datensatz. Dann zeichnen wir Histogramme und beurteilen, ob die Variable normalverteilt ist oder nicht. Zunächst überprüfen wir jedoch den Datensatz, indem wir die Funktion `head()` aufrufen.

```
# Lesen der Datei students.csv als Dataframe;
students = pd.read_csv(
    "../../data/students.csv",
    usecols=["age", "height", "weight", "score1", "score2", "salary"],
)
students.head(10)
```

	age	height	weight	score1	score2	salary
0	19	160	64.8	NaN	NaN	NaN
1	19	172	73.0	NaN	NaN	NaN
2	22	168	70.6	45.0	46.0	NaN
3	19	183	79.7	NaN	NaN	NaN
4	21	175	71.4	NaN	NaN	NaN
5	19	189	85.8	NaN	NaN	NaN
6	21	156	65.9	NaN	NaN	NaN
7	21	167	65.7	58.0	62.0	NaN
8	18	195	94.4	57.0	67.0	NaN
9	18	165	66.0	NaN	NaN	NaN

Um einen Überblick über die Form der Verteilung der einzelnen Variablen zu erhalten, verwenden wir die Methode `hist()`.

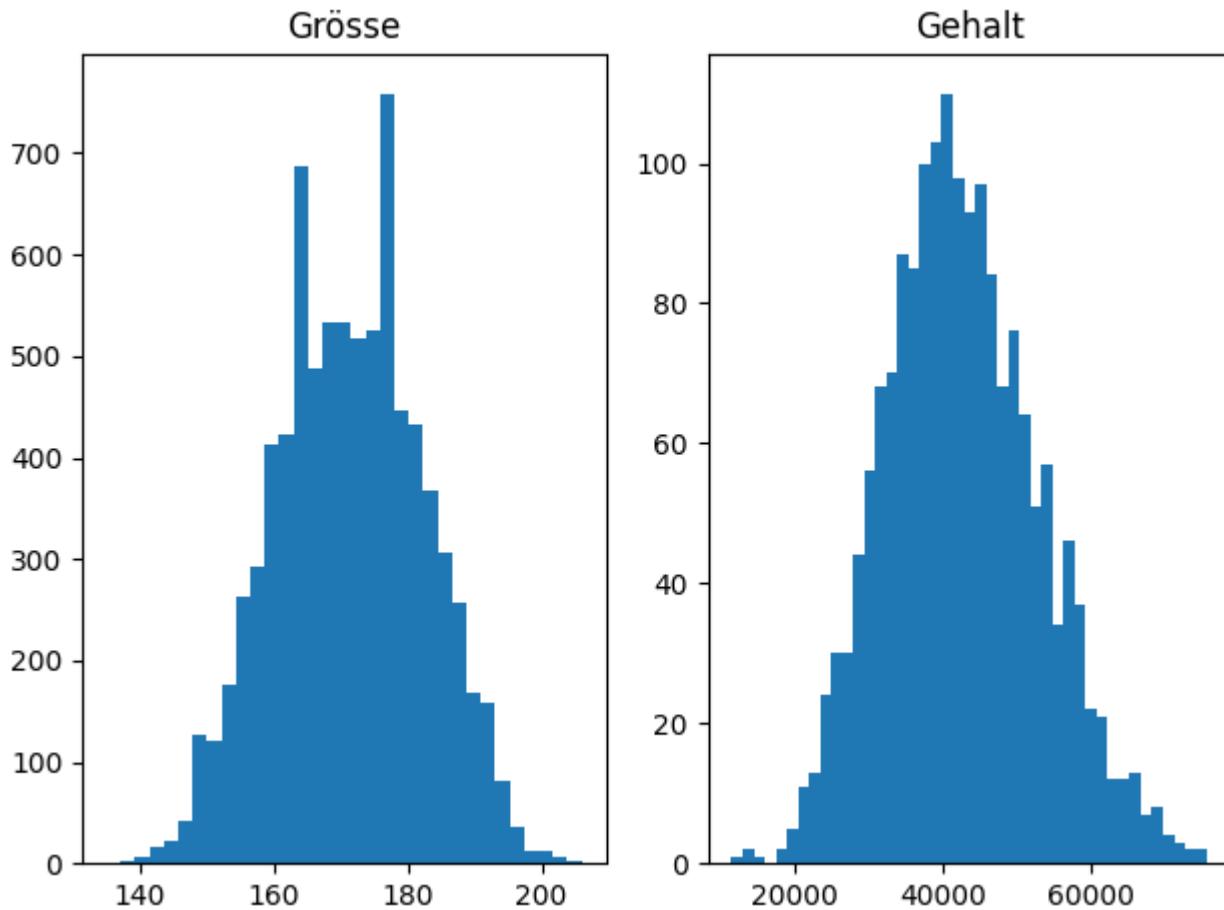
```
cols = students.columns
fig, ax = plt.subplots(ncols=int(len(cols) / 2), nrows=2)
ax = np.ravel(ax) # vereinfacht das iterieren durch die axes Objekte
titles = {
    "age": "Alter",
    "height": "Grösse",
    "weight": "Gewicht",
    "score1": "Punkte1",
    "score2": "Punkte2",
    "salary": "Gehalt",
}
for e, col in enumerate(cols):
    bins = int(students[col].nunique() / 2)
    ax[e].hist(students[col], bins=bins, density=True)
    ax[e].set_title(titles[col])
fig.tight_layout()
```



Wir stellen sofort fest, dass einige Variablen positiv verzerrt sind, also schließen wir sie aus und behalten diejenigen, die normal verteilt zu sein scheinen.

```
cols = ["height", "salary"]
fig, ax = plt.subplots(ncols=2)
ax = np.ravel(ax) # vereinfacht das iterieren durch die axes Objekte
bins_fact = {"height": 2, "salary": 40}

for e, col in enumerate(cols):
    bins = int(students[col].nunique() / bins_fact[col])
    ax[e].hist(students[col], bins=bins)
    ax[e].set_title(titles[col])
fig.tight_layout()
```



Nun, sowohl die Variable `height` als auch die Variable `salary` scheinen mehr oder weniger normalverteilt zu sein. Es ist also eine Frage des Geschmacks, welche Variable man für die weitere Analyse auswählt. Für den Moment bleiben wir bei der Gehaltsvariable und überprüfen, ob die drei oben genannten empirischen Regeln gültig sind. Wir wechseln zu Python und validieren diese Regeln, indem wir zunächst den Mittelwert und die Standardabweichungen berechnen. Bitte beachten Sie, dass die Gehaltsvariable Fehlwerte enthält, die mit `NA` gekennzeichnet sind. Daher schließen wir zunächst alle `NA`-Werte aus, indem wir die Funktion `dropna()` anwenden.

```
salary = students["salary"].dropna()
salary
```

```
10      45254.11
12      40552.79
13      27007.03
16      33969.16
27      50617.64
...
8228    33259.70
8233    41028.24
8234    36750.09
8237    40112.04
8238    45900.13
Name: salary, Length: 1753, dtype: float64
```

```
print(f"Mittelwert des Gehalts:           {salary.mean()}")
print(f"1 Standardabweichung des Gehalts: {salary.std()}")
print(f"2 Standardabweichungen des Gehalts: {2 * salary.std()}")
print(f"3 Standardabweichungen des Gehalts: {3 * salary.std()}"
```

```
Mittelwert des Gehalts:           42522.112253280095
1 Standardabweichung des Gehalts: 10333.139991647347
2 Standardabweichungen des Gehalts: 20666.279983294695
3 Standardabweichungen des Gehalts: 30999.419974942044
```

Wie in der obigen allgemeinen Beispielform zählen wir die Anzahl der Werte, die größer als $+1$ s.d. bzw. kleiner als -1 s.d. und $+2$ s.d. bzw. -2 s.d. und $+3$ s.d. bzw. -3 s.d. sind, und setzen sie in Beziehung zur Länge des Vektors, in unserem Fall 1753.

```

salary_mean = salary.mean()
salary_std = salary.std()

sd1 = (
    sum((salary > (salary_mean - salary_std)) & (salary < (salary_mean + salary_std))
    / salary.shape[0]
)

sd2 = (
    sum(
        (salary > (salary_mean - 2 * salary_std))
        & (salary < (salary_mean + 2 * salary_std))
    )
    / salary.shape[0]
)

sd3 = (
    sum(
        (salary > (salary_mean - 3 * salary_std))
        & (salary < (salary_mean + 3 * salary_std))
    )
    / salary.shape[0]
)

print("sd1 :", sd1)
print("sd2 :", sd2)
print("sd3 :", sd3)

```

```

sd1 : 0.6708499714774672
sd2 : 0.9560752994865944
sd3 : 0.9977181973759269

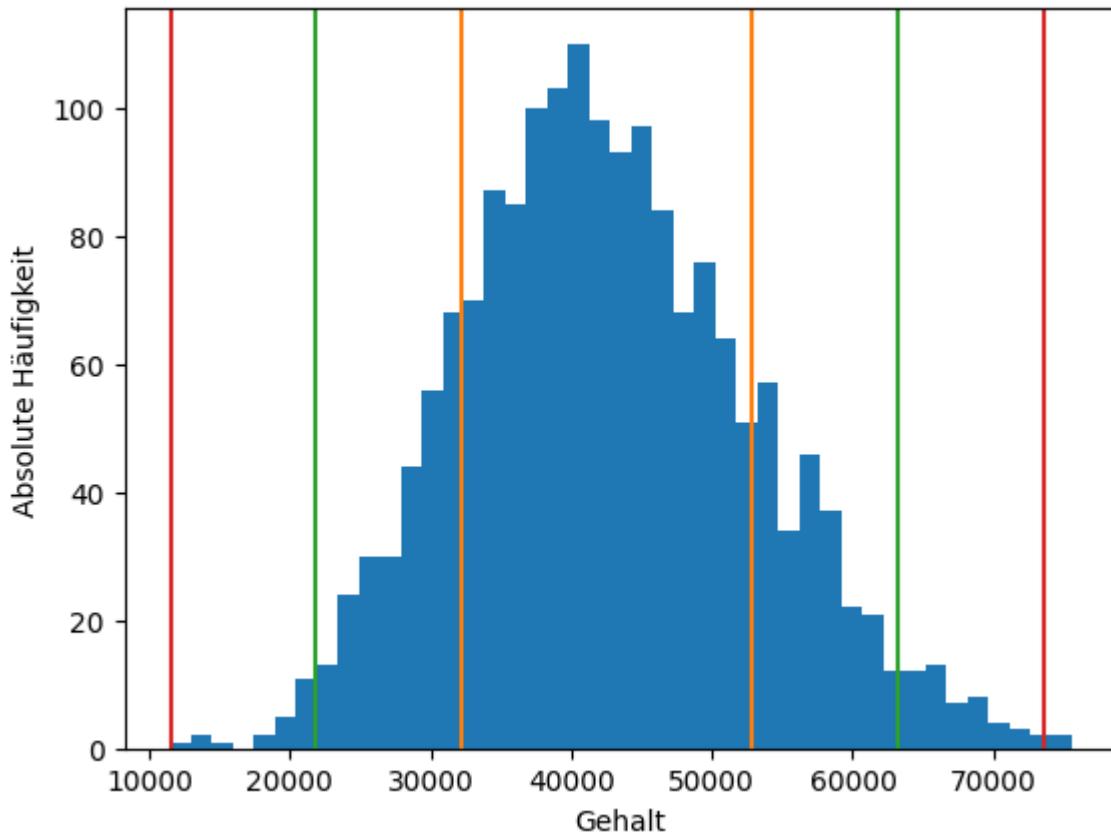
```

Wow, ziemlich nah dran! Offensichtlich zeigt die Gehaltsvariable eine starke Tendenz zur Unterstützung der so genannten empirischen Regel. Wir stellen das Histogramm für die Variable `salary` dar, um unseren Eindruck zu bestätigen.

```

fig, ax = plt.subplots()
bins = int(salary.nunique() / 40)
ax.hist(salary, bins=bins)
ax.set_ylabel("Absolute Häufigkeit")
ax.set_xlabel("Gehalt")
for e in range(3):
    ax.axvline(salary_mean + (e + 1) * salary_std, color=f"C{e+1}")
    ax.axvline(salary_mean - (e + 1) * salary_std, color=f"C{e+1}")

```



Wir können nun unseren Visualisierungsansatz erweitern, indem wir die **empirische Dichteschätzung** mit der Funktion `scipy_kernel.evaluate()` grafisch darstellen und ihre Form überprüfen. Wir stellen die empirische Dichteschätzung als gestrichelte Linie dar, indem wir das [\(Linientyp-Argument\)](#) `'-.'` und eine Linienbreite von 3 (Argument `linewidth=3.0`) setzen.

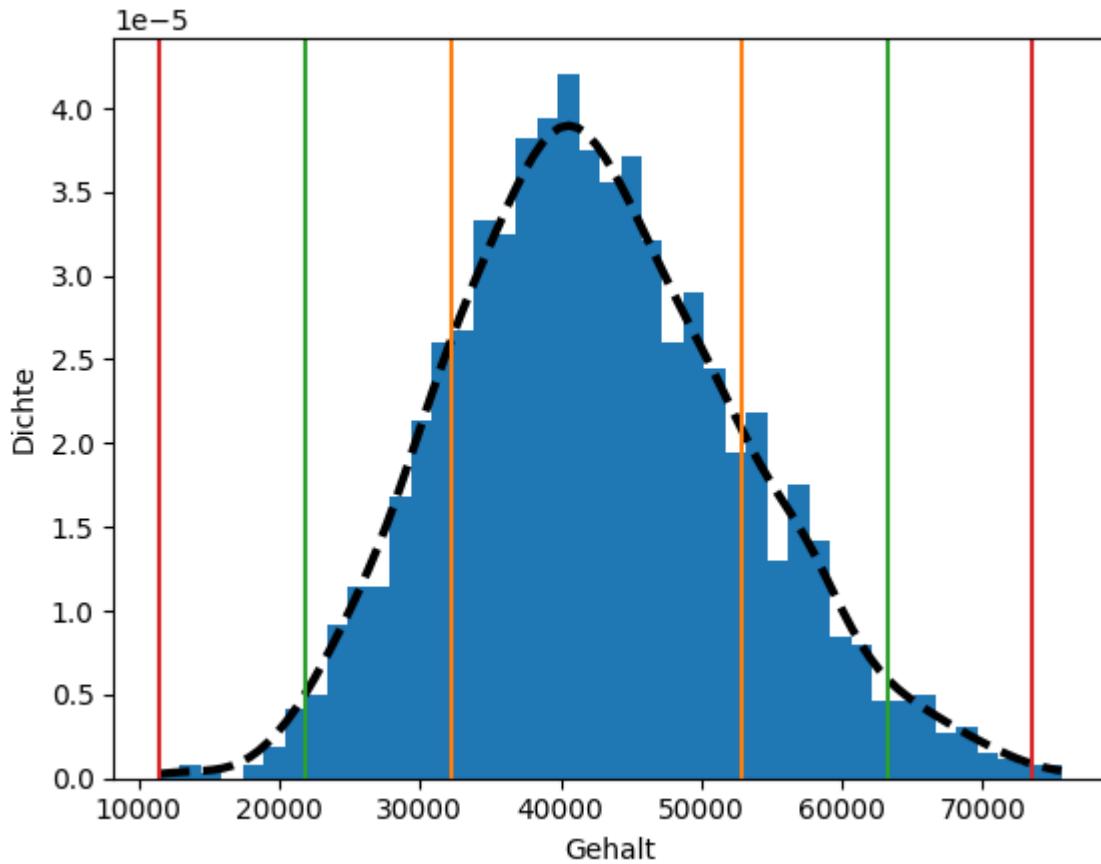
```

fig, ax = plt.subplots()
bins = int(salary.unique() / 40)
ax.hist(salary, bins=bins, density=True)
ax.set_ylabel("Dichte")
ax.set_xlabel("Gehalt")

# empirische Dichteschätzung
x_salary = np.linspace(salary.min(), salary.max(), 100)
scipy_kernel = gaussian_kde(salary)
dens_emp = scipy_kernel.evaluate(x_salary)
ax.plot(x_salary, dens_emp, color="k", linestyle="dashed", linewidth=3.0)

# Standardabweichungen
for e in range(3):
    ax.axvline(salary_mean + (e + 1) * salary_std, color=f"C{e+1}")
    ax.axvline(salary_mean - (e + 1) * salary_std, color=f"C{e+1}")

```

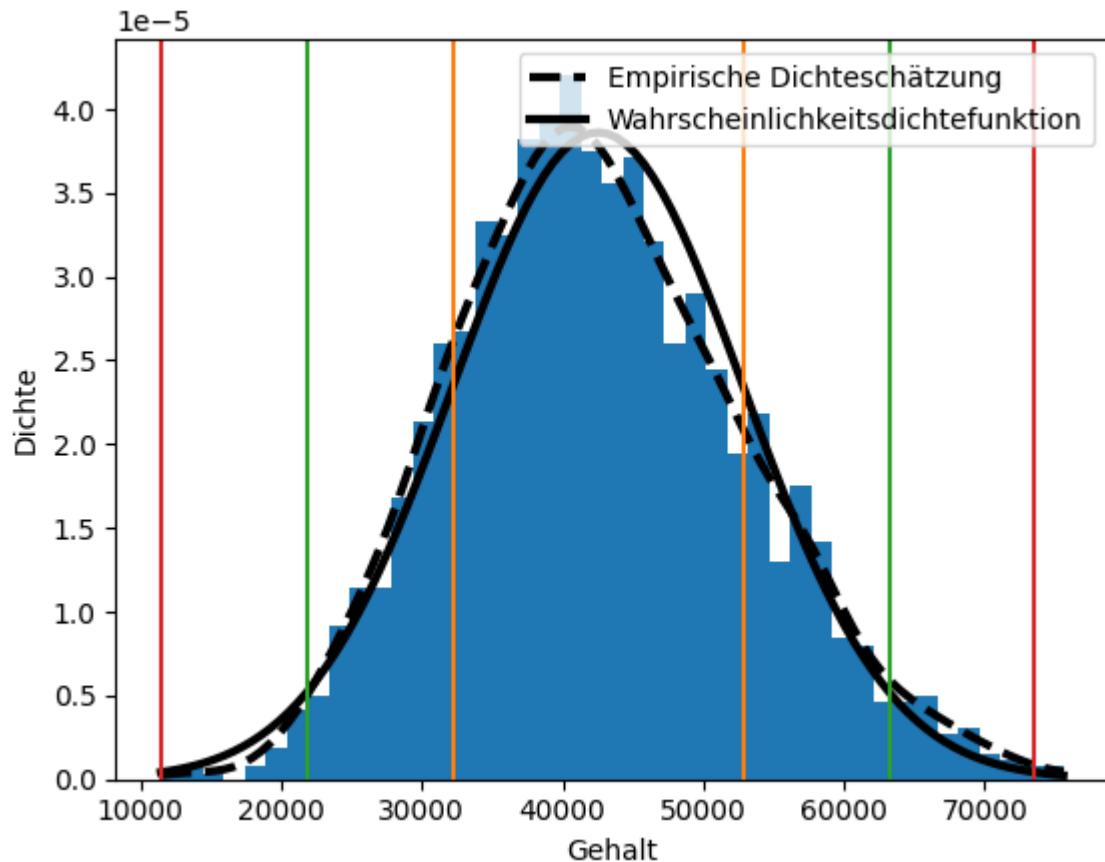


Schließlich vergleichen wir unsere **empirische Dichteschätzung** mit der theoretischen **Wahrscheinlichkeitsdichtefunktion**, die auf dem tatsächlichen Mittelwert und der Standardabweichung der Daten `salary` basiert. Für einen besseren visuellen Vergleich wechseln wir zurück zu einer nicht eingefärbten Histogramm-Darstellung.

```
fig, ax = plt.subplots()
bins = int(salary.unique() / 40)
ax.hist(salary, bins=bins, density=True)
ax.set_ylabel("Dichte")
ax.set_xlabel("Gehalt")

# empirische Dichteschätzung
x_salary = np.linspace(salary.min(), salary.max(), 100)
scipy_kernel = gaussian_kde(salary)
dens_emp = scipy_kernel.evaluate(x_salary)
ax.plot(
    x_salary,
    dens_emp,
    color="k",
    linestyle="dashed",
    linewidth=3.0,
    label="Empirische Dichteschätzung",
)
# Wahrscheinlichkeitsdichtefunktion
pdf = norm.pdf(x_salary, loc=salary.mean(), scale=salary.std())
ax.plot(
    x_salary, pdf, color="k", linewidth=3.0, label="Wahrscheinlichkeitsdichtefunktion"
)
# Standardabweichungen
for e in range(3):
    ax.axvline(salary_mean + (e + 1) * salary_std, color=f"C{e+1}")
    ax.axvline(salary_mean - (e + 1) * salary_std, color=f"C{e+1}")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7fd44fd9e0>
```



Wir können daraus schließen, dass `salary` im Datensatz der `students` ungefähr normalverteilt ist. Die Grafik zeigt jedoch, dass die Verteilung der Gehaltsvariablen leicht linksschief ist. Dies ist an der Abweichung zwischen der **empirischen Dichteschätzung** und der **Wahrscheinlichkeitsdichtefunktion** zu erkennen.

Die Spannweite

Die **Spannweite** als Maß für die Streuung ist einfach zu berechnen. Sie ergibt sich aus der Differenz zwischen dem größten und dem kleinsten Wert in einem Datensatz.

$$\text{Range} = \text{größter Wert} - \text{kleinster Wert}$$

Betrachten wir unseren `students` Datensatz. Wir unterteilen den Datensatz so, dass er nur numerische Daten enthält.

```
df = pd.read_csv("../data/students.csv")
df.sample(10)
```

	stud_id	name	gender	age	height	weight	religion	nc_score	semes
3599	255883	Morrison-Cook, Austin	Male	20	172	72.6	Other	1.67	
2081	686345	Lenahan, Kyle	Male	21	181	78.0	Other	1.38	
521	346731	Lee, Nicholas	Male	22	192	93.1	Catholic	1.27	
5603	963305	Kramer, David	Male	56	170	72.6	Orthodox	2.14	
3670	124687	Than, Jonathan	Male	22	172	72.4	Protestant	1.79	
5375	696262	Macasinag, Paulo	Male	20	197	94.3	Catholic	3.41	
2317	452190	Briseno, Alisha	Female	63	162	68.0	Catholic	2.19	
4839	955228	Thapa, Richard	Male	23	175	76.0	Other	2.48	
1285	854649	Duling, Joshua Ian	Male	61	170	70.3	Protestant	1.54	>
117	582922	Valenciano-Poffenbarger, Quincie	Female	19	175	76.8	Catholic	2.19	

Wir sind also an den Kategorien `age`, `height`, `weight` und `nc_score` interessiert. Wir verwenden die Methoden `min()` und `max()`, um das Minimum und Maximum der ausgewählten Variablen zu berechnen. Erneut greifen wir auf die `agg`-Methode zurück.

```
summary = df[["age", "height", "weight", "nc_score"]].agg(["min", "max"])
summary
```

	age	height	weight	nc_score
min	18	135	51.4	1.0

	age	height	weight	nc_score
max	64	206	116.0	4.0

Um nun die Spannweite für jede Variable zu berechnen, müssen wir nur eine Zeile von der anderen abziehen.

```
summary.loc["Spannweite"] = summary.loc["max"] - summary.loc["min"]
summary
```

	age	height	weight	nc_score
min	18.0	135.0	51.4	1.0
max	64.0	206.0	116.0	4.0
Spannweite	46.0	71.0	64.6	3.0

Die Spannweite hat, wie der Mittelwert, den Nachteil, dass sie durch Ausreißer beeinflusst wird.

Daher ist die Spannweite kein gutes Streuungsmaß für einen Datensatz, der Ausreißer enthält. Ein weiterer Nachteil der Verwendung der Spannweite als Streuungsmaß ist, dass ihre Berechnung nur auf zwei Werten basiert: Dem größten und dem kleinsten. Alle anderen Werte in einem Datensatz werden bei der Berechnung der Spanne ignoriert. Daher ist die Spannweite oftmals kein sehr zufriedenstellendes Maß für die Streuung (s.62).

Das Positionsmaß

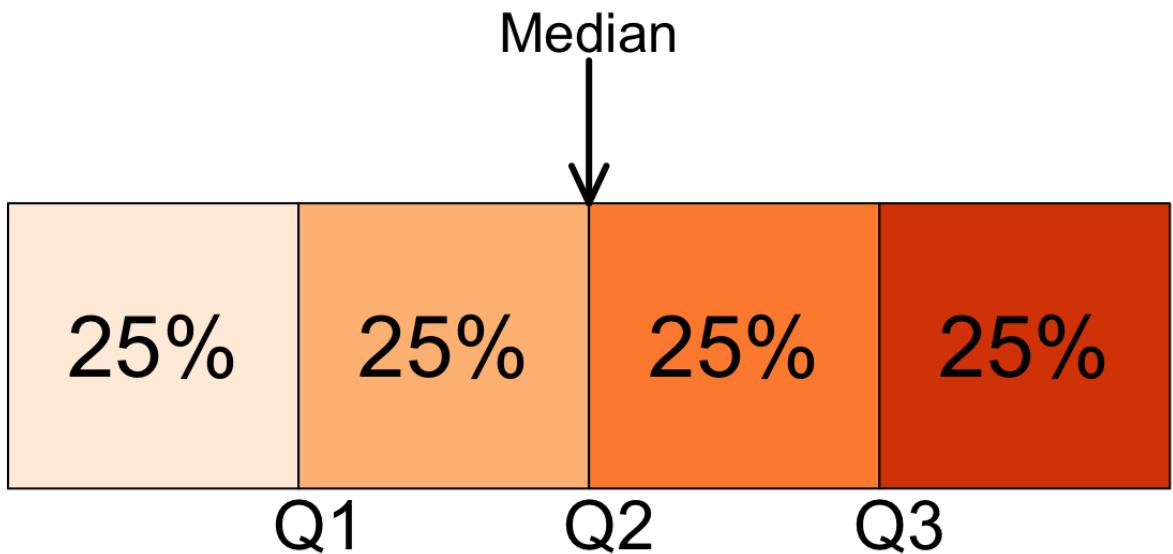
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stats
```

Ein Positionsmaß bestimmt die Position eines einzelnen Wertes im Verhältnis zu anderen Werten in einer Stichprobe oder einem Populationsdatensatz. Anders als der Mittelwert und die Standardabweichung sind auf **Quantilen** basierende deskriptive Maße nicht empfindlich gegenüber dem Einfluss einiger weniger extremer Beobachtungen. Aus diesem Grund werden deskriptive Maße, die auf Quantilen basieren, häufig denjenigen vorgezogen, die auf dem Mittelwert und der Standardabweichung basieren (s.60).

Quantile sind Punkte, die den Bereich der Daten in zusammenhängende Intervalle mit gleichen Wahrscheinlichkeiten unterteilen. Bestimmte [Quantile](#) sind besonders wichtig: Der **Median** eines Datensatzes unterteilt die Daten in zwei gleiche Teile: die unteren 50% und die oberen 50%. **Quartile** unterteilen die Daten in vier gleiche Teile und **Perzentile** unterteilen sie in Hundertstel oder 100 gleiche Teile. Beachten Sie, dass der Median auch das 50-te Perzentil ist. **Dezile** unterteilen einen Datensatz in Zehntel (10 gleiche Teile), und die **Quintile** unterteilen einen Datensatz in Fünftel (5 gleiche Teile). Es gibt immer ein Quantil weniger als die Anzahl der gebildeten Gruppen (z. B. gibt es **3** Quartile, die die Daten in **4** gleiche Teile unterteilen!).

Quartile und Interquartilsbereich

Quartile unterteilen einen geordneten Datensatz in **vier gleiche Teile**. Diese drei Maße werden als **erstes Quartil (Q_1)**, **zweites Quartil (Q_2)** und **drittes Quartil (Q_3)** bezeichnet. Das zweite Quartil ist dasselbe wie der Median eines Datensatzes. Das erste Quartil ist der Wert des mittleren Terms unter den Beobachtungen, die kleiner als der Median sind, und das dritte Quartil ist der Wert des mittleren Terms unter den Beobachtungen, die größer als der Median sind (s.59).



Ungefähr 25% der Werte in einem geordneten Datensatz sind kleiner als $Q1$ und etwa 75% sind größer als $Q1$. Das zweite Quartil, $Q2$, unterteilt einen geordneten Datensatz in zwei gleiche Teile; daher sind das zweite Quartil und der Median identisch. Etwa 75% der Datenwerte sind kleiner als $Q3$ und etwa 25% sind größer als $Q3$. Die Differenz zwischen dem dritten Quartil und dem ersten Quartil eines Datensatzes wird als **Interquartilsbereich (IQR)** bezeichnet (s.61).

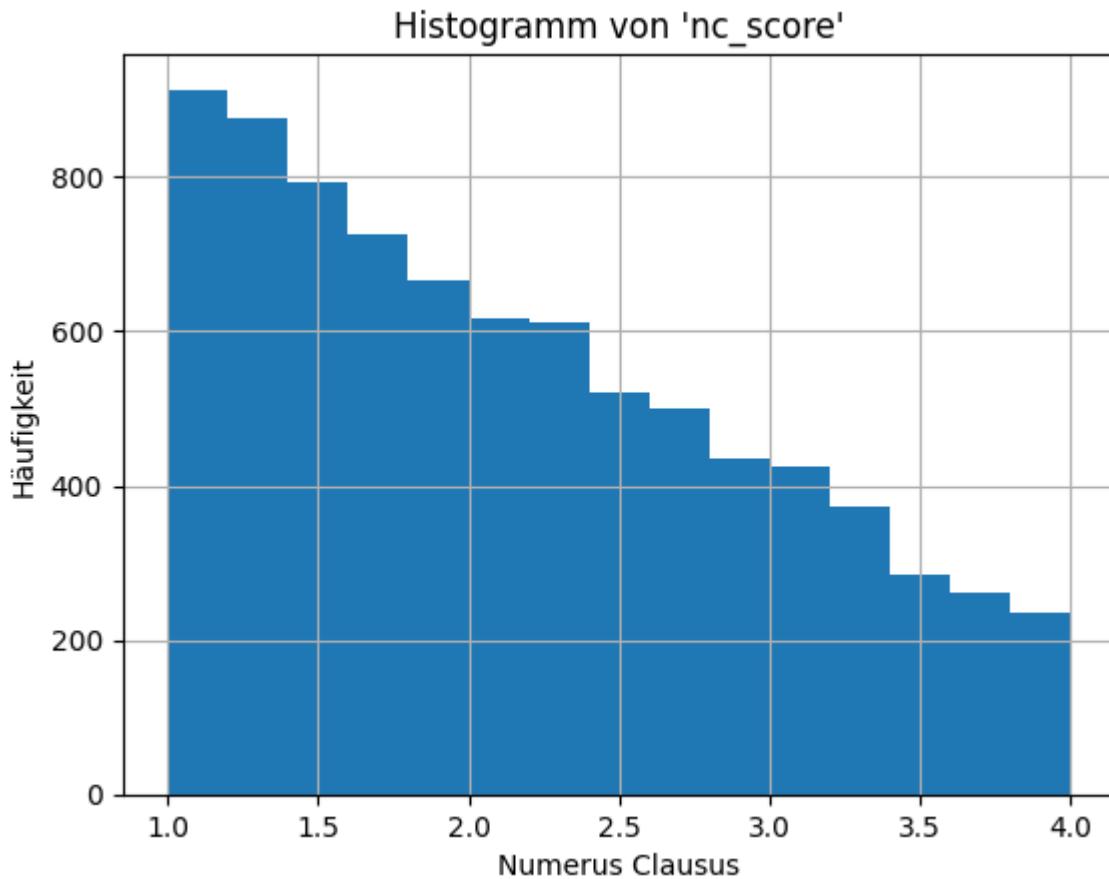
$$IQR = Q3 - Q1$$

Wechseln wir zu Python und testen wir seine Funktionalität zur Berechnung von Quantilen/Quartilen. Wir werden die `nc_score`-Variable des `students` Datensatzes verwenden, um Quartile und den IQR zu berechnen. Die `nc_score`-Variable entspricht der Numerus-Clausus-Punktzahl jedes Studenten.

Zunächst werden die Daten unterteilt und ein Histogramm erstellt, um die Verteilung der Variablen genauer zu untersuchen.

```
# Lese Daten ein
df = pd.read_csv("../data/students.csv")

# Histogramm
df["nc_score"].hist(bins=15)
plt.xlabel("Numerus Clausus")
plt.ylabel("Häufigkeit")
plt.title("Histogramm von 'nc_score'")
plt.show()
```



Um die Quartile für die Variable nc_score zu berechnen, wenden wir die Funktion `np.quantile` an.

Um die Quartile für die Variable `nc_score` zu berechnen, schreiben wir also einfach:

```
for e, q in enumerate([0.25, 0.5, 0.75, 1]):
    print(f"{e+1}. Quantil des nc_score: {np.quantile(df['nc_score'], q)}")
```

1. Quantil des nc_score: 1.46
2. Quantil des nc_score: 2.04
3. Quantil des nc_score: 2.78
4. Quantil des nc_score: 4.0

Hinweis: Nicht alle Statistiker definieren Quartile auf genau dieselbe Weise. Eine ausführliche Diskussion der verschiedenen Methoden zur Berechnung von Quantilen finden Sie in dem Online-Artikel "["Quartiles in Elementary Statistics" von E. Langford \(2006\)](#)".

Um den IQR für die Variable `nc_score` zu berechnen, schreiben wir entweder...

```
# Berechne Interquartil Abstand mit Funktion np.percentile()
q1, q3 = np.percentile(df["nc_score"], [25, 75])
iqr = q3 - q1
iqr
```

```
np.float64(1.3199999999999998)
```

...oder wir wenden die eingebaute Funktion `iqr` aus dem Modul `scipy.stats` an:

```
stats.iqr(df["nc_score"], rng=(25, 75))
```

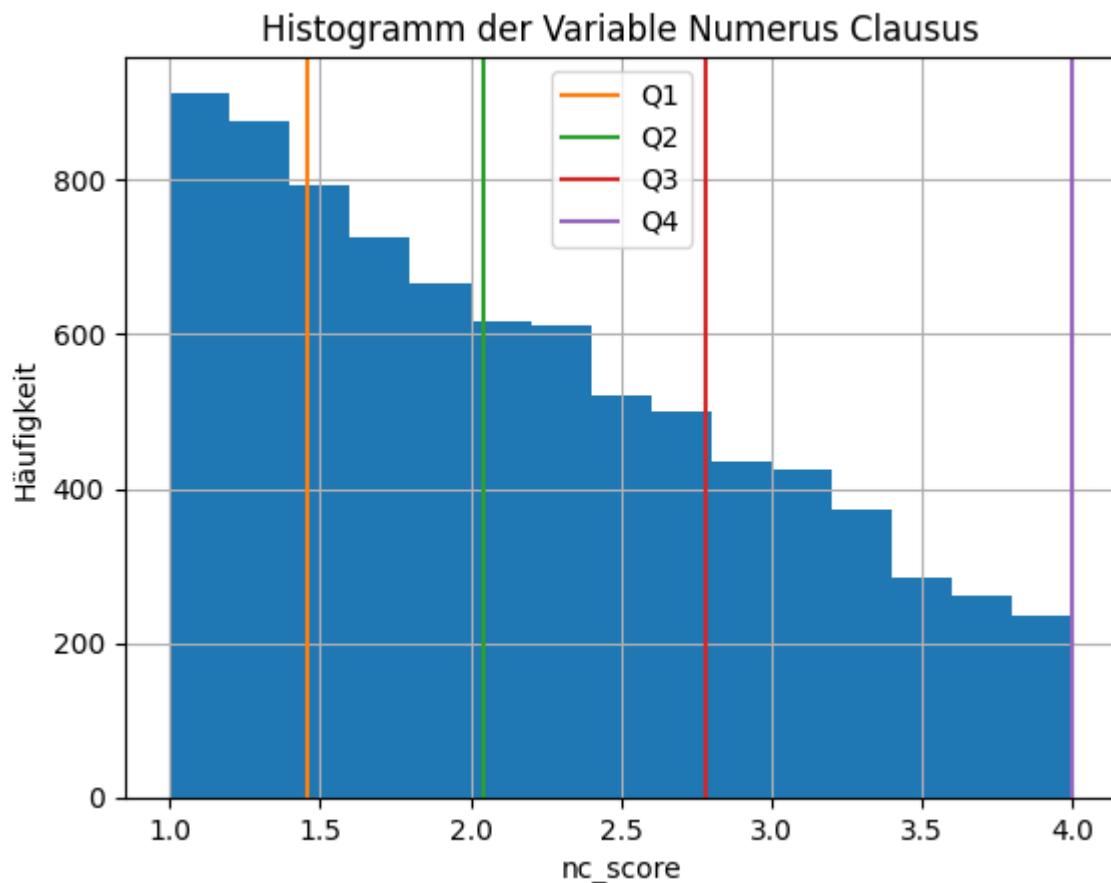
```
np.float64(1.3199999999999998)
```

Wir können die Aufteilung der Variablen `nc_score` in Quartile visualisieren, indem wir ein Histogramm erstellen und ein paar zusätzliche Codezeilen hinzufügen.

```
# Plotte die Liste als Histogramm mit Quartilen
df["nc_score"].hist(bins=15)
plt.xlabel("nc_score")
plt.ylabel("Häufigkeit")
plt.title("Histogramm der Variable Numerus Clausus")

# Zeichne vertikale Linien bei Q1, Q2, Q3, Q4
for e, q in enumerate([0.25, 0.5, 0.75, 1]):
    plt.axvline(x=np.quantile(df["nc_score"], q), color=f"C{e+1}", label=f"Q{e+1}")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa2607e9570>
```



Die 5-Punkte-Zusammenfassung

Aus den drei Quartilen ($Q1, Q2, Q3$) können wir ein Maß für die Lage der Mitte (den Median, $Q2$) und ein Maß für die Variation der beiden mittleren Quartale der Daten, $Q2 - Q1$ für die zweiten Quartile und $Q3 - Q2$ für die dritten Quartile ableiten. Die drei Quartile sagen jedoch nichts über die Variation der ersten und vierten Quartile aus.

Um diese Informationen zu erhalten, beziehen wir auch die Beobachtungen des Minimums und des Maximums mit ein. Die Variation der ersten Quartile kann als Differenz zwischen dem Minimum und der ersten Quartile, $Q1 - Min$, gemessen werden, und die Variation der vierten Quartile kann als Differenz zwischen der dritten Quartile und dem Maximum, $Max - Q3$, gemessen werden. Somit liefern das Minimum, das Maximum und die Quartile zusammen unter anderem Informationen über Zentrum und Variation (s.62).

Die so genannte **Tukey-Fünf-Punkte-Zusammenfassung** (nach dem Mathematiker [John Wilder Tukey](#)) eines Datensatzes besteht aus den Werten *Min*, *Q1*, *Q2*, *Q3* und *Max* des Datensatzes.

Die Fünf-Punkte-Zusammenfassung lässt sich in Python durch `np.percentile` und die `min()`, `max()` Funktionen berechnen. Zu Demonstrationszwecken berechnen wir die Fünf-Punkte-Zusammenfassung für die Variable `nc_score`

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
df = pd.read_csv("../data/students.csv", index_col=0)

# Berechne Fünf-Zahlen-Zusammenfassung
scores = df["nc_score"]

# Berechne die Quartile
q1, median, q3 = np.percentile(scores, [25, 50, 75])

# Berechne minimal/maximal Datenpunkte
data_min, data_max = min(scores), max(scores)

# Ausgabe der Daten
print(f"Min: {data_min}")
print(f"Q1: {q1}")
print(f"Median: {median}")
print(f"Q3: {q3}")
print(f"Max: {data_max}")
```

```
Min: 1.0
Q1: 1.46
Median: 2.04
Q3: 2.78
Max: 4.0
```

Diese Funktion liefert Minimum, untere Quartile, Median, obere Quartile und Maximum für die Eingabedaten.

In Python gibt es eine ähnliche Methode namens `describe()`, die ähnliche Statistiken liefert.

```
df.describe()
```

	age	height	weight	nc_score	score1	score2
count	8239.000000	8239.000000	8239.000000	8239.000000	4892.000000	4892.000000
mean	22.541571	171.380750	72.998131	2.166481	68.164963	69.494685
std	6.065111	11.077529	8.635162	0.811548	14.051762	14.395251
min	18.000000	135.000000	51.400000	1.000000	30.000000	31.000000
25%	20.000000	163.000000	66.500000	1.460000	58.000000	59.000000
50%	21.000000	171.000000	71.800000	2.040000	70.000000	71.000000
75%	23.000000	180.000000	78.500000	2.780000	78.000000	80.000000
max	64.000000	206.000000	116.000000	4.000000	97.000000	98.000000

Perzentile und Perzentilrang

Perzentile unterteilen einen geordneten Datensatz in 100 gleiche Teile. Jeder (geordnete) Datensatz hat 99 Perzentile, die ihn in 100 gleiche Teile unterteilen. Das **k-te** Perzentil wird mit P_k bezeichnet, wobei k eine ganze Zahl im Bereich von 1 bis 99 ist. Das 25-te Perzentil wird zum Beispiel mit P_{25} bezeichnet.

So kann das **k-te** Perzentil, P_k , als ein Wert in einem Datensatz definiert werden, bei dem etwa $k\%$ der Messungen kleiner als der Wert von P_k und etwa $(100 - k)$ der Messungen größer als der Wert von P_k sind

Der ungefähre Wert des **k-ten** Perzentils, bezeichnet mit P_k , ist

$$P_k = \frac{kn}{100}$$

wobei **k** die Nummer des Perzentils bezeichnet und **n** den Stichprobenumfang darstellt.

Als Übung berechnen wir das 38-te, das 50-te und das 73-te Perzentil der Variablen `nc_score` in Python. Zunächst berechnen wir das 38. Perzentil gemäß der oben angegebenen Gleichung. Dann wenden wir die `np.percentile()`-Funktion von Python an, um das 38-te, 50-te und 73-te Perzentil der Variablen `nc_score` zu ermitteln.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
df = pd.read_csv("../data/students.csv", index_col=0)

scores = df["nc_score"].sort_values()

# Berechne 38-te Perzentile
k = 38
n = len(scores)
print(f"Die {k}-te Perzentile ist {round((k*n)/100)}.")


```

Die 38-te Perzentile ist 3131.

scores

```
stud_id
427725    1.0
547711    1.0
174533    1.0
808854    1.0
340635    1.0
...
575545    4.0
941727    4.0
619443    4.0
595839    4.0
547558    4.0
Name: nc_score, Length: 8239, dtype: float64
```

Wir wählen den Wert anhand dieser Zahl in der geordneten Liste `nc_score` aus und vergleichen ihn mit den Perzentilwerten.

`scores.iloc[3131]`

`np.float64(1.74)`

```
quartiles = np.percentile(scores, [38, 50, 73])
print(quartiles)
```

```
[1.74 2.04 2.71]
```

Das hat gut funktioniert! Sie können prüfen, ob der Median der Variablen `nc_score` dem 50-te Perzentil (2,04) entspricht, wie oben berechnet.

Wir können auch den **Perzentilrang** für einen bestimmten Wert x_i eines Datensatzes mit der folgenden Gleichung berechnen:

$$\text{Perzentil Rang für } x_i = \frac{\text{Werte kleiner als } x_i}{\text{Gesamtanzahl der Werte im Datensatz}}$$

Der Perzentilrang von x_i gibt den Prozentsatz der Werte im Datensatz an, die kleiner als x_i sind.

In Python gibt es keine eingebaute Funktion zur Berechnung des Perzentilrangs. Es ist jedoch relativ einfach, eine solche Funktion selbst zu schreiben.

```
# Definiere Funktion my_percentile_rank
def my_percentile_rank(x, data: pd.Series):
    """Computes the percentile rank
    Args:
        x: int or float number
        data: Pandas Series
    """
    return sum(data < x) / len(data)
```

Jetzt berechnen wir zum Beispiel den Perzentilrang für einen Numerus clausus von 2,5.

```
my_percentile_rank(2.5, scores)
```

```
0.6627017841971113
```

Wenn wir das Ergebnis auf den nächsten ganzzahligen Wert runden, können wir feststellen, dass etwa 66% der Studenten in unserem Datensatz einen Numerus clausus von mehr als 2,5 hatten.

Ausreißer und Boxplots

Ausreißer

Bei der Datenanalyse ist die Identifizierung von Ausreißern und damit von Beobachtungen, die deutlich aus dem Gesamt muster der Daten herausfallen, sehr wichtig. Ein Ausreißer erfordert besondere Aufmerksamkeit. Er kann das Ergebnis eines Mess- oder Aufzeichnungsfehlers, einer Beobachtung aus einer anderen Population oder einer ungewöhnlich extremen Beobachtung sein. Beachten Sie, dass eine extreme Beobachtung nicht zwangsläufig ein Ausreißer sein muss, sondern auch ein Hinweis auf eine Schieflage sein kann (s.62).

Wenn wir einen Ausreißer beobachten, sollten wir versuchen, seine Ursache zu ermitteln. Ist ein Ausreißer auf einen Mess- oder Aufzeichnungsfehler zurückzuführen oder gehört er aus einem anderen Grund eindeutig nicht zum Datensatz, kann er einfach entfernt werden. Wenn jedoch keine Erklärung für einen Ausreißer ersichtlich ist, ist die Entscheidung, ob er im Datensatz verbleiben soll, eine schwierige Ermessensentscheidung.

Als Diagnoseinstrument zum Aufspüren von Beobachtungen, die Ausreißer sein könnten, können wir Quartile und den **IQR** verwenden. Daher definieren wir die **Untergrenze** und die **Obergrenze** eines Datensatzes. Die untere Grenze ist die Zahl, die $1,5 \times IQR$ unter dem ersten Quartil liegt; die obere Grenze ist die Zahl, die $1,5 \times IQR$ über dem dritten Quartil liegt. Beobachtungen, die unterhalb der Untergrenze oder oberhalb der Obergrenze liegen, sind potenzielle Ausreißer (s.61).

$$\text{Untere Grenze} = Q1 - 1,5 \times IQR$$

$$\text{Obere Grenze} = Q3 + 1,5 \times IQR$$

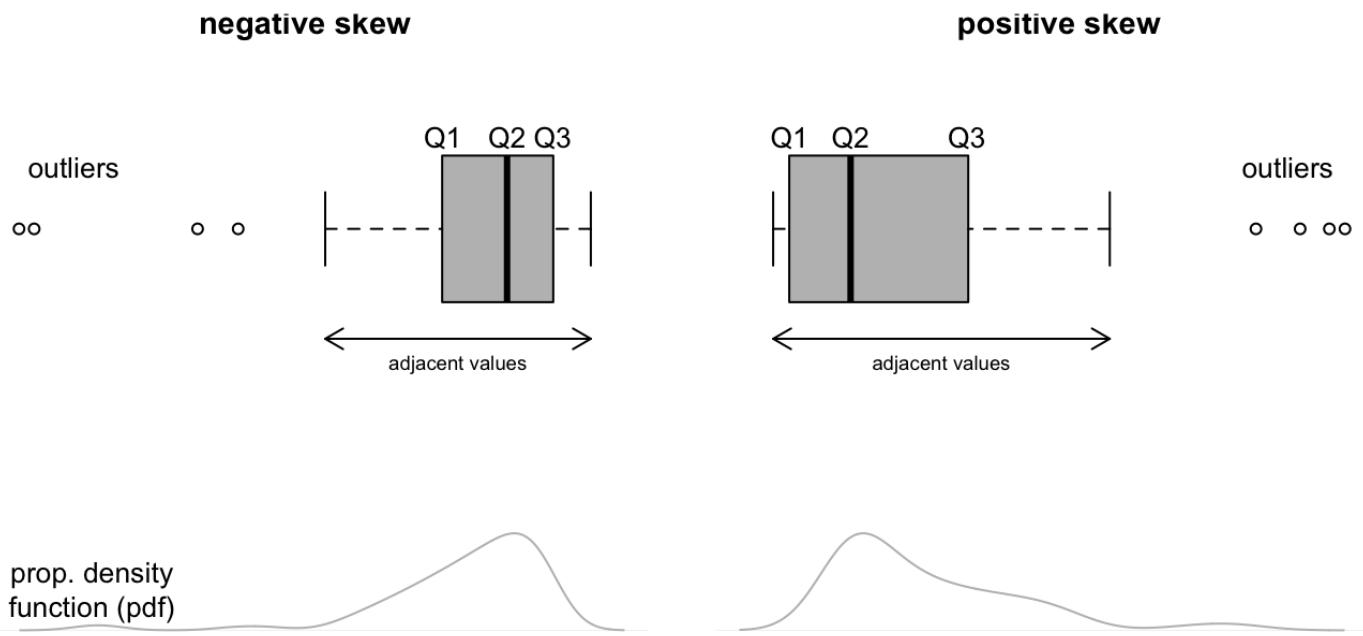
Boxplots

Das Boxplot-Diagramm, auch **Box-and-Whisker-Diagramm** genannt, basiert auf der Fünf-Punkte-Zusammenfassung (s.62) und kann zur grafischen Darstellung von Zentrum und Variation eines

Datensatzes verwendet werden. Diese Diagramme wurden von dem Mathematiker [John Wilder Tukey](#) erfunden. Es sind mehrere [Arten von Boxplots](#) gebräuchlich.

Box-and-Whisker-Diagramm bieten eine grafische Darstellung der Daten anhand von fünf Maßzahlen: dem Median, dem ersten Quartil, dem dritten Quartil sowie dem kleinsten und dem größten Wert des Datensatzes zwischen der unteren und der oberen Grenze. Der Abstand zwischen den verschiedenen Teilen des Kastens zeigt den Grad der Streuung und Schiefe der Daten an. Durch die Erstellung von Box-and-Whisker-Diagrammen können verschiedene Verteilungen miteinander verglichen werden. Es hilft auch, Ausreißer zu erkennen (s.62). Box-Plots können entweder horizontal oder vertikal gezeichnet werden.

Box-Whisker-Plot



Die Ränder der Box sind immer das erste und dritte Quartil, und der Bereich innerhalb der Box ist immer das zweite Quartil (der Median). Die von den Boxen ausgehenden Linien (Whisker) zeigen die Variabilität außerhalb des oberen und unteren Quartils an. Um einen Boxplot zu erstellen, benötigen wir auch das Konzept der benachbarten Werte. Die **angrenzenden Werte** eines Datensatzes sind die extremsten Beobachtungen, die noch innerhalb der unteren und oberen Grenzen liegen; sie sind die extremsten Beobachtungen, die keine potenziellen Ausreißer sind. Ausreißer können als einzelne

Punkte aufgetragen werden. Wenn es in einem Datensatz keine potenziellen Ausreißer gibt, sind die angrenzenden Werte lediglich das Minimum und das Maximum der Beobachtungen ().

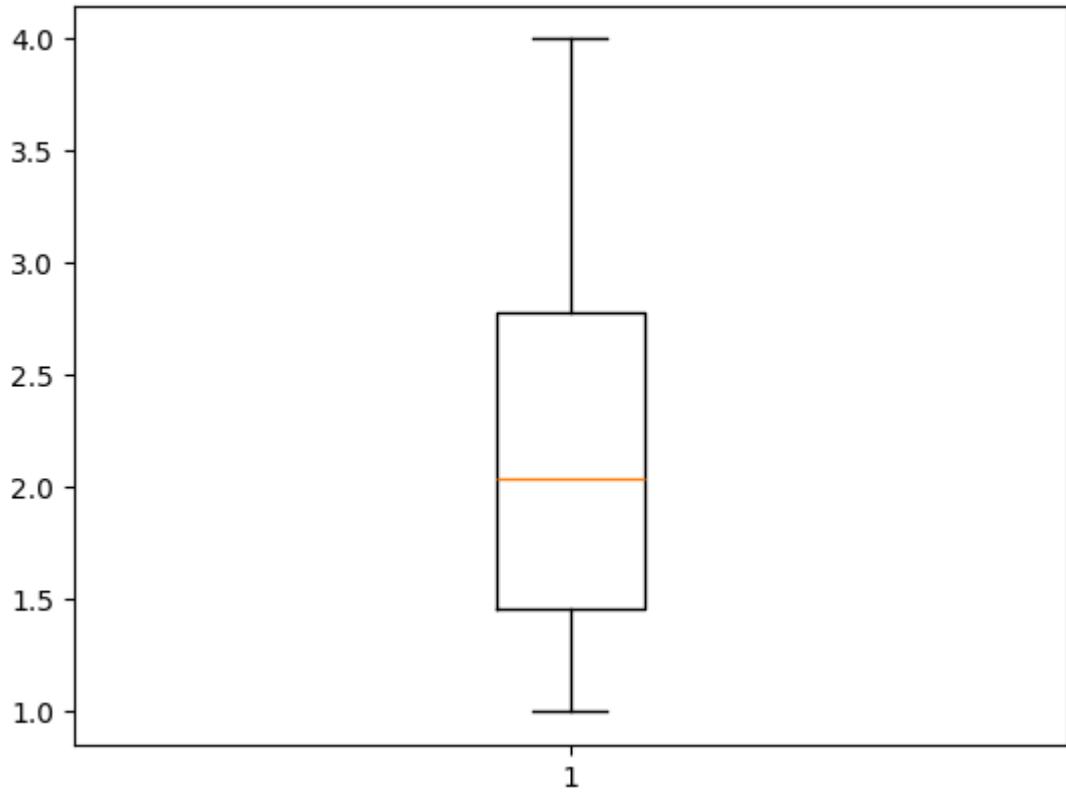
Lassen Sie uns nun eine Reihe von Boxplots erstellen, um den Datensatz `students` eingehender zu analysieren. Wir beginnen mit der Erstellung eines Boxplots für die Variable `nc_score`.

```
# Lese der Datei students.csv; nur Spalten 'nc_score', 'semester', 'gender', 'height' \v
students = pd.read_csv(
    "../../data/students.csv", usecols=["nc_score", "semester", "gender", "height"]
)
students
```

	gender	height	nc_score	semester
0	Female	160	1.91	1st
1	Female	172	1.56	2nd
2	Female	168	1.24	3rd
3	Male	183	1.37	2nd
4	Female	175	1.46	1st
...
8234	Male	181	2.91	6th
8235	Male	178	2.03	2nd
8236	Female	169	3.72	3rd
8237	Male	195	2.74	4th
8238	Female	170	3.29	>6th

8239 rows × 4 columns

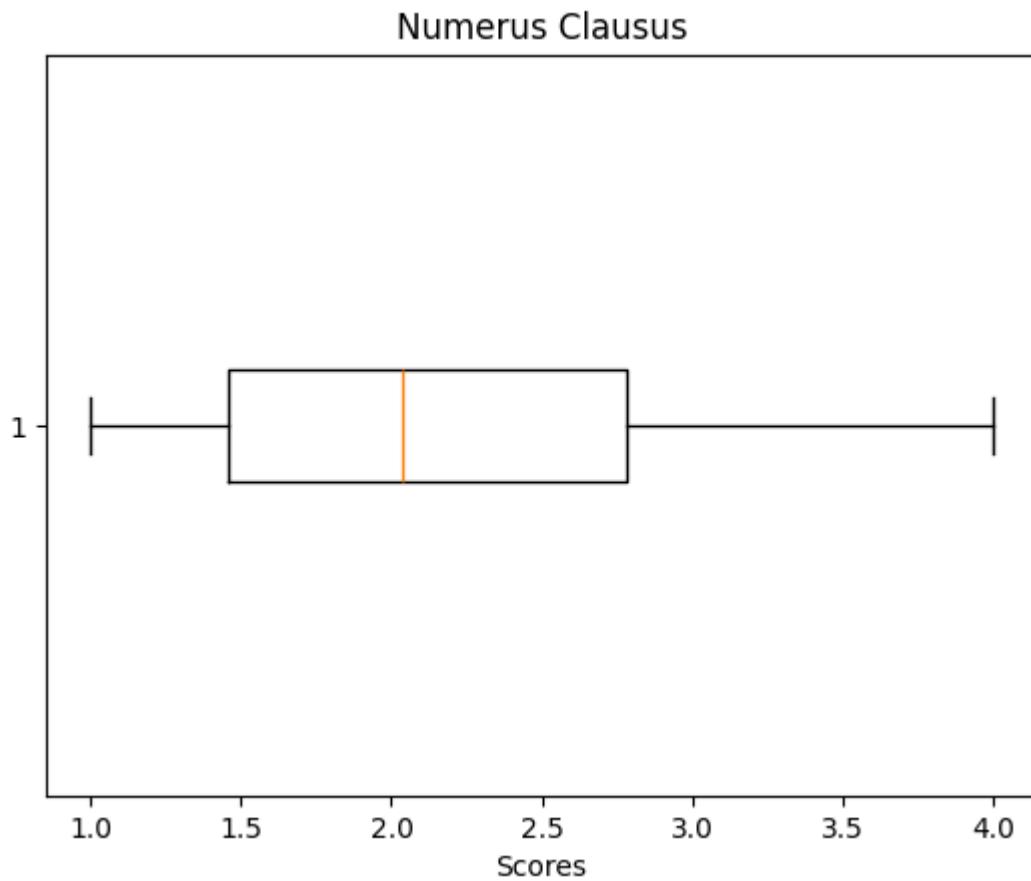
```
_ = plt.boxplot(students["nc_score"])
```



Wir erhalten sofort einen Eindruck von der Streuung und Schiefe der Daten. Durch Hinzufügen des Arguments `vert = False` zum Boxplot drehen wir den Boxplot um 90° .

```
fig, ax = plt.subplots()
ax.boxplot(students["nc_score"], vert=False)
ax.set_xlabel("Scores")
ax.set_title("Numerus Clausus")
```

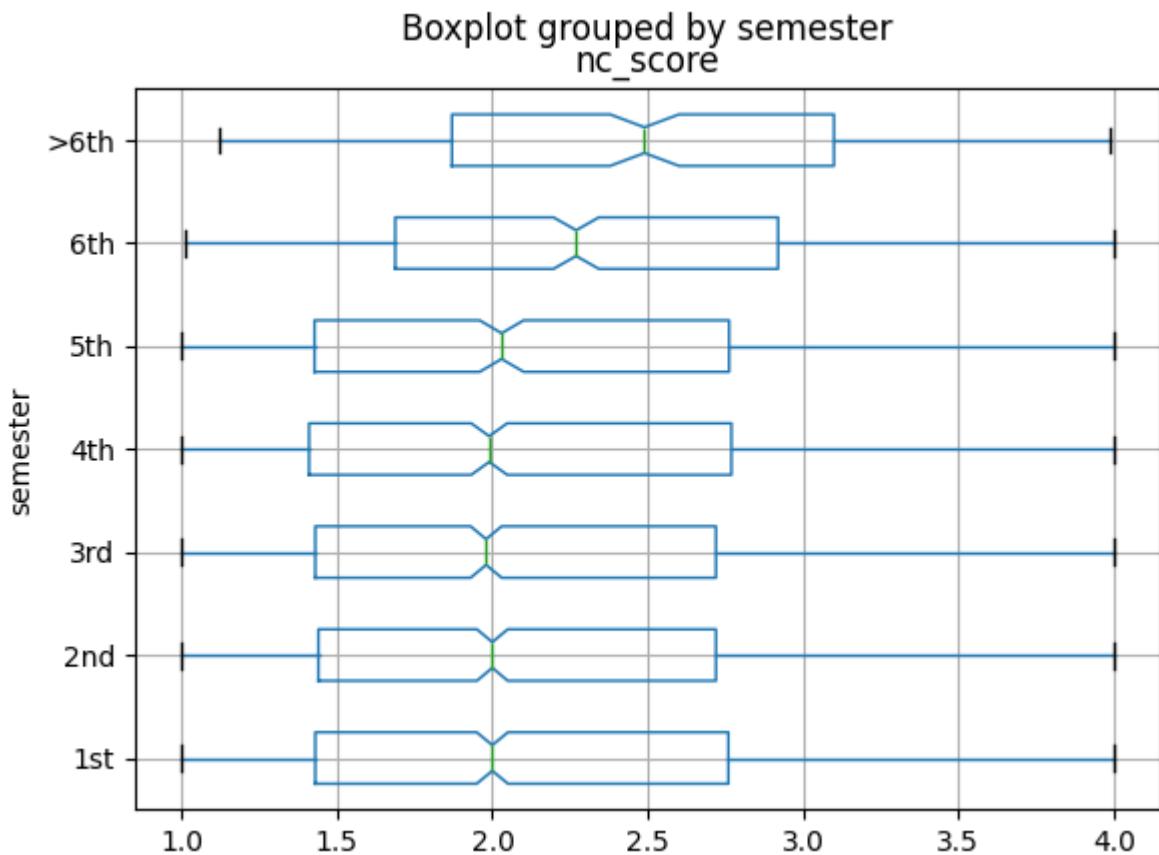
```
Text(0.5, 1.0, 'Numerus Clausus')
```



Boxplots sind eine sehr leistungsfähige Technik für die explorative Datenanalyse, da es sehr einfach ist, die Variable von Interesse, in unserem Fall die `nc_score`-Variable, auf andere Variablen zu beziehen und in ihrer Relation darstellen. In Python können wir das erreichen indem wir das Argument `by` in der `boxplot` Methode anwenden.

Zeichnen wir einen Boxplot der Variable `nc_score` in Abhängigkeit von der Variable `semester`. Die `semester`-Variable entspricht dem Semester, in dem der jeweilige Student studiert. Zu Ihrer Information: Die Mindeststudienzeit für die untersuchten Studiengänge ist auf 4 Semester festgelegt.

```
_ = students.boxplot(column="nc_score", by="semester", vert=False, notch=True)
```

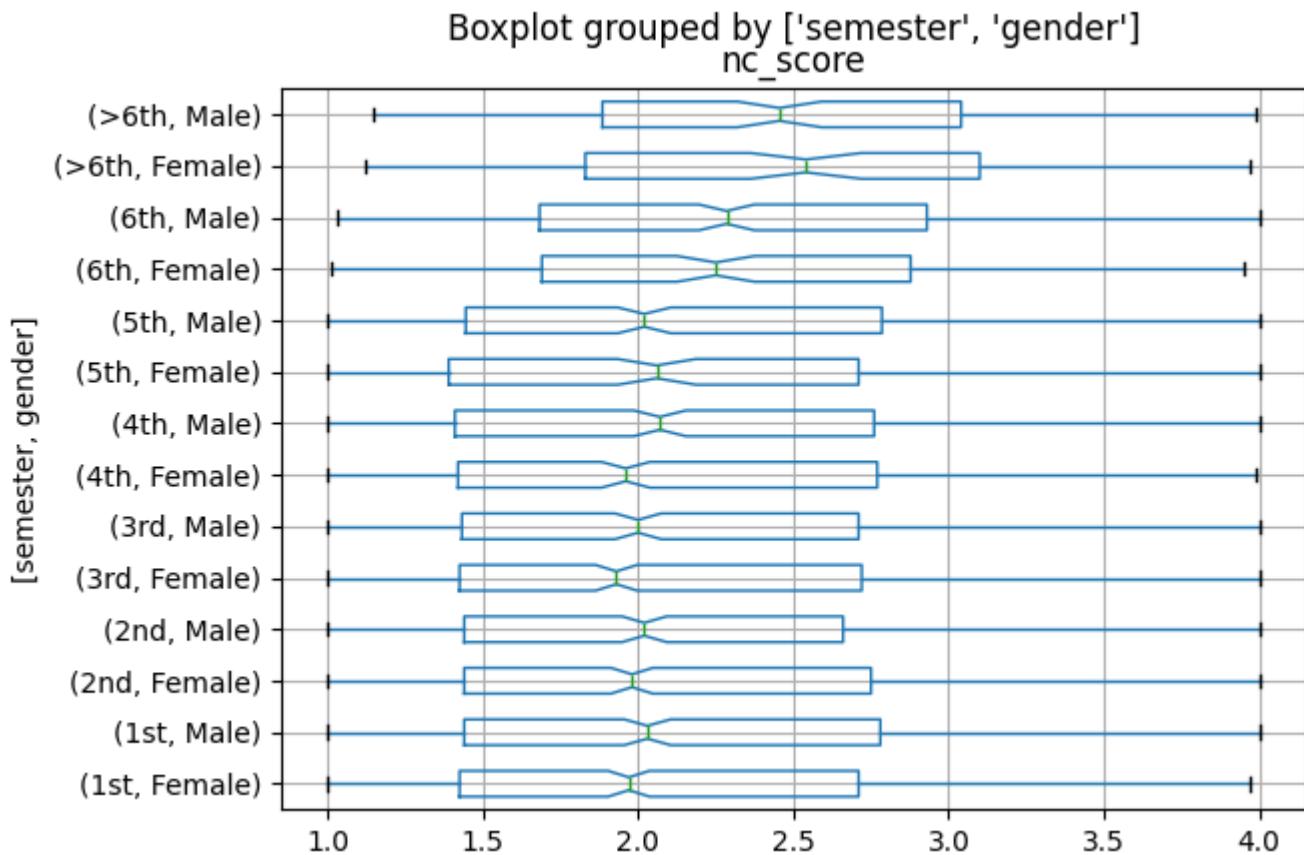


Interessant, nicht wahr? Die Grafik zeigt, dass Studierende höherer Semester (> 5 -tes Semester) beim *Numerus clausus* tendenziell schlechter abschneiden. Mit anderen Worten: Studierende, die ihr Studium innerhalb der Mindeststudienzeit abschließen, haben tendenziell eine höhere *Numerus clausus*-Punktzahl.

Doch damit sind wir noch nicht fertig. Wir wollen wissen, ob das Geschlecht einen Einfluss auf diese Beobachtung hat. Wir können ganz einfach eine Interaktionsvariable einfügen, indem wir die Variable nach den gruppiert werden soll in eckige Klammern setzen : `[var1, var2]`. Außerdem verwenden wir das Argument `notch` in der `boxplot` Methode. Wenn sich die notches zweier Diagramme nicht überschneiden, ist dies ein “starker Hinweis” dafür, dass sich die beiden Mediane unterscheiden [Chambers, et al. \(1983\): Graphical Methods for Data Analysis. Wadsworth & Brooks/Cole, S. 62](#).

Bitte beachten Sie, dass wir eine zusätzliche Zeile Code schreiben müssen, um ein schöneres *y*-Label zu erhalten.

```
_ = students.boxplot(
  column="nc_score", by=["semester", "gender"], vert=False, notch=True
)
```

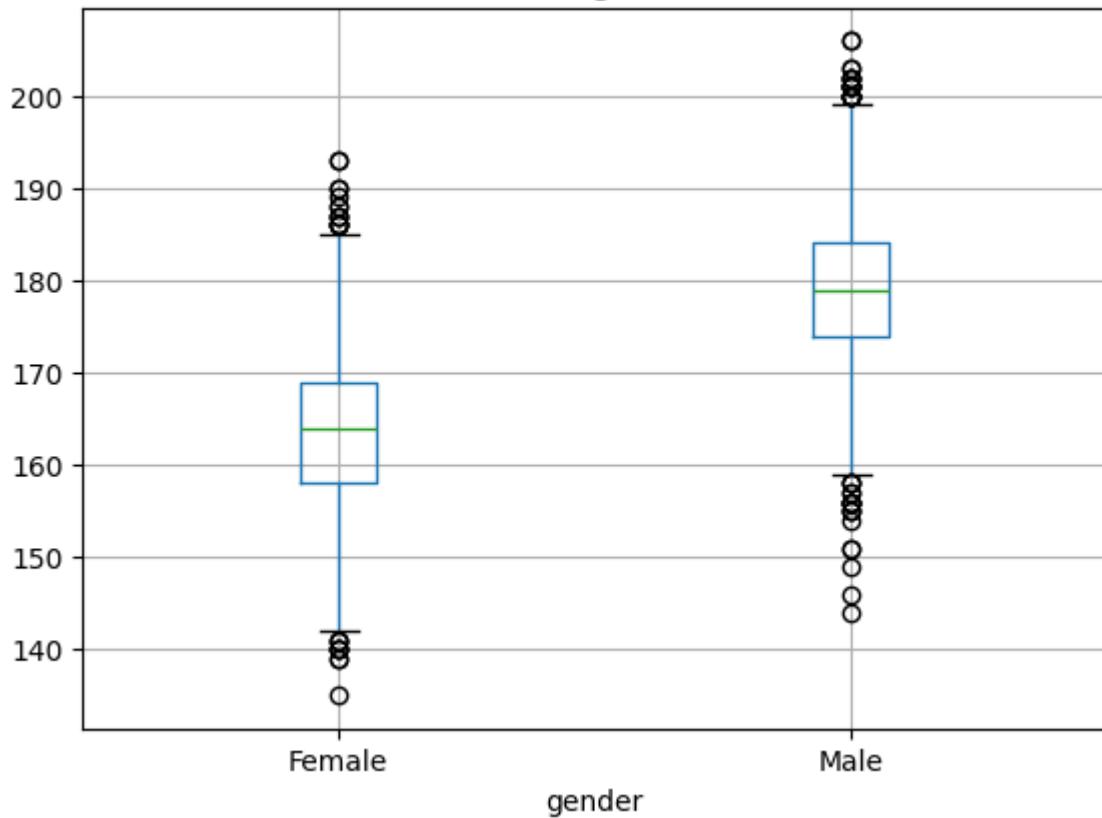


Dieses Diagramm ist nicht so einfach zu interpretieren. Es scheint jedoch, dass die oben gemachte Beobachtung bestätigt wird: Studierende aus höheren Semestern (> 5-tes) schneiden beim *Numerus clausus* tendenziell schlechter ab. Der Einfluss des Geschlechts auf die *Numerus-Clausus*-Werte ist jedoch nicht so eindeutig. Wir werden Methoden der **Inferenzstatistik** anwenden müssen, um zu beurteilen, ob diese Unterschiede *statistisch signifikant* sind oder ob diese Schwankungen um den Median auch rein zufällig sein könnten.

Zum Abschluss dieses Abschnitts und um auch einen Boxplot mit Ausreißern zu sehen, stellen wir die Variable `height` gegen die Variable `gender` dar.

```
_ = students.boxplot(column="height", by="gender")
```

Boxplot grouped by gender
height



Offensichtlich, und sicherlich nicht unerwartet, gibt es einen Unterschied in der Größe der Studenten zwischen den verschiedenen Gruppen (männlich oder weiblich). Weibliche Studenten sind tendenziell kleiner als männliche, aber wenn wir uns die Extreme ansehen, gibt es in beiden Gruppen große und kleine Personen. Wie bereits erwähnt, müssen wir jedoch zunächst die Daten auf *statistische Signifikanz* prüfen, um sicher zu sein, dass der beobachtete Größenunterschied nicht nur zufällig ist.

Maße für die Relation zwischen Variablen

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

In den vorangegangenen Abschnitten haben wir uns hauptsächlich auf eine einzelne Variable, die Beschreibung einer Verteilung und die Berechnung von zusammenfassenden Statistiken konzentriert. Statistiken, die eine einzelne Variable beschreiben, werden als univariate Statistiken bezeichnet. Untersucht man die Beziehung zwischen zwei Variablen, spricht man von bivariater Statistik. Wenn

die Beziehungen zwischen mehreren Variablen gleichzeitig untersucht werden, spricht man von [multivariaten Statistiken](#). **Maße für die Relation** zwischen Variablen bieten die Möglichkeit, die Größe der Verknüpfung zwischen zwei Variablen zusammenzufassen.

Kovarianz

Die [Kovarianz](#) ist ein Maß für die gemeinsame Variabilität von zwei Variablen. Die Kovarianz kann jeden Wert im Intervall $]-\infty, +\infty[$ annehmen.

Die Kovarianz ist positiv, wenn die größeren/kleineren Werte der einen Variablen hauptsächlich mit den größeren/kleineren Werten der anderen Variablen übereinstimmen. Die Kovarianz ist negativ, wenn die Variablen tendenziell ein entgegengesetztes Verhalten zeigen. Die Kovarianz ist negativ, wenn die größeren Werte der einen Variablen hauptsächlich mit den kleineren Werten der anderen Variablen übereinstimmen.

Die Kovarianz, s_{xy} wird durch die folgende Gleichung berechnet

$$s_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

Die **normalisierte Version der Kovarianz** wird **Korrelationskoeffizient** genannt, dessen Größe die Stärke einer linearen Beziehung zwischen zwei Variablen angibt.

Korrelation

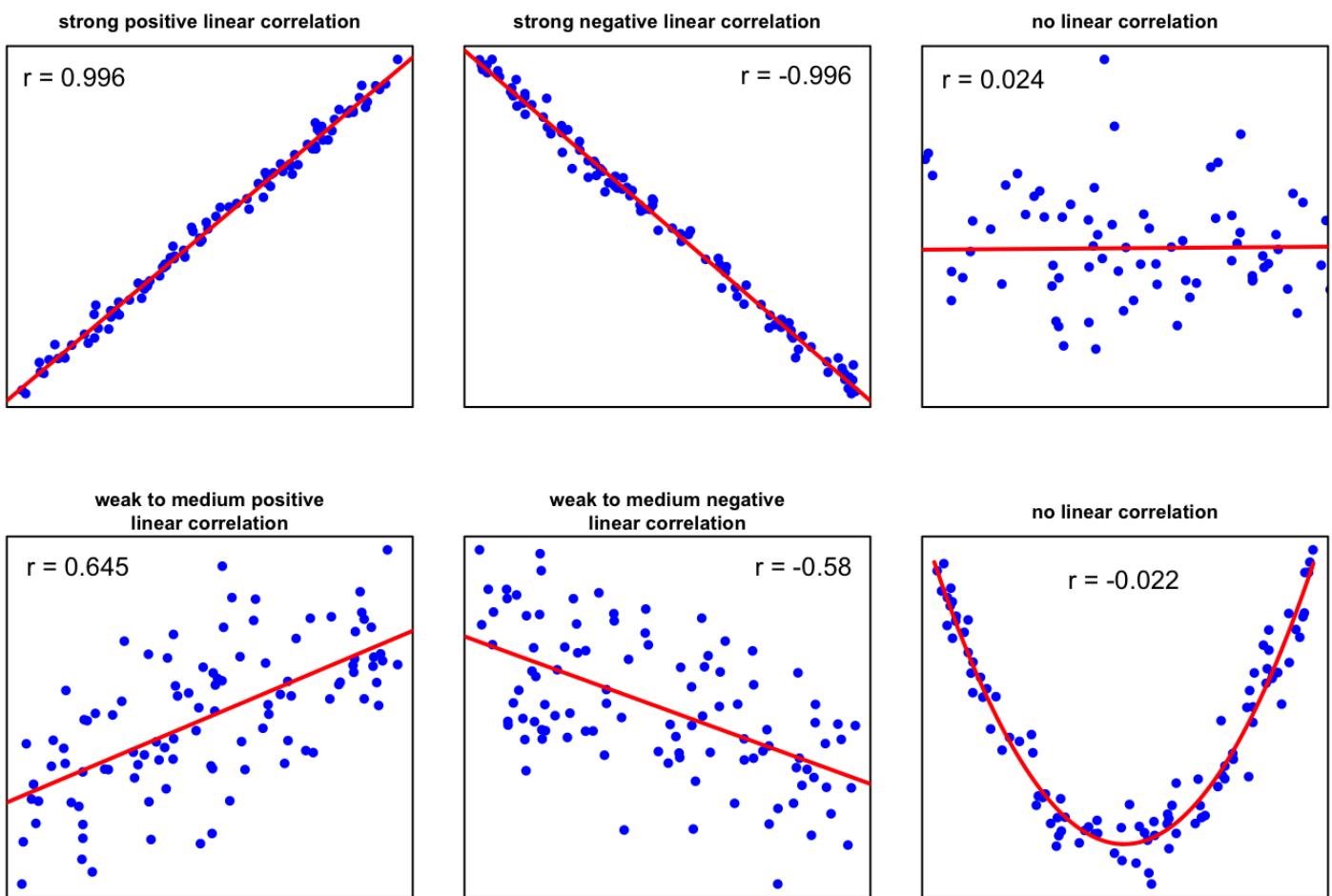
Pearson-Produkt-Moment-Korrelationskoeffizient

Die Korrelation ist eine weit verbreitete Methode zur Untersuchung der Beziehung zwischen **quantitativen Variablen**. Die am häufigsten verwendete Statistik ist der **lineare Korrelationskoeffizient r** , der zu Ehren seines Entwicklers, [Karl Pearson](#), auch als [Pearson-Produkt-Moment-Korrelationskoeffizient](#) bekannt ist. Er ist gegeben durch

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{s_{xy}}{s_x s_y}$$

wobei s_{xy} die Kovarianz von x und y und s_x und s_y die Standardabweichung von x bzw. y sind. Durch Division durch die Standardabweichungen der Stichprobe, s_x und s_y , wird der lineare Korrelationskoeffizient, r , skalenunabhängig und nimmt Werte zwischen -1 und 1 an.

Der lineare Korrelationskoeffizient misst die Stärke der linearen Beziehung zwischen zwei Variablen. Liegt r nahe bei ± 1 , sind die beiden Variablen stark korreliert, und in einem Streudiagramm liegen die Datenpunkte um eine Linie herum. Liegt r weit von ± 1 entfernt, sind die Datenpunkte weiter gestreut. Liegt r nahe bei 0 , sind die Datenpunkte im Wesentlichen um eine horizontale Linie gestreut, was darauf hindeutet, dass es fast keine lineare Beziehung zwischen den Variablen gibt.



Eine interessante Eigenschaft von r ist, dass sein Vorzeichen die Steigung der linearen Beziehung zwischen zwei Variablen widerspiegelt. Ein positiver Wert von r deutet darauf hin, dass die Variablen **positiv linear korreliert** sind, was bedeutet, dass y tendenziell linear zunimmt, wenn x zunimmt. Ein

negativer Wert von r deutet darauf hin, dass die Variablen **negativ linear korreliert** sind, was bedeutet, dass y tendenziell linear abnimmt, wenn x zunimmt.

Es gibt keine eindeutige Klassifizierungsregel für die Größe einer linearen Beziehung zwischen zwei Variablen. Die folgende Tabelle kann jedoch als Faustregel für den Umgang mit den numerischen Werten des Pearson-Produkt-Moment-Korrelationskoeffizienten dienen.

starke lineare Korrelation	$r > 0,9$
mittlere lineare Korrelation	$0,7 < r \leq 0,9$
schwache lineare Korrelation	$0,5 < r \leq 0,7$
keine oder zweifelhafte lineare Korrelation	$0 < r \leq 0,5$

Die Pearson-Korrelation setzt voraus, dass die Variablen annähernd normalverteilt sind, und sie ist bei Vorhandensein von Ausreißern nicht robust.

In einem späteren Abschnitt über lineare Regression wird das Bestimmtheitsmaß, R^2 ein anschauliches Maß für die Qualität von linearen Modellen. Es besteht eine enge Beziehung zwischen R^2 und dem linearen Korrelationskoeffizienten r . Das Bestimmtheitsmaß R^2 ist gleich dem Quadrat des linearen Korrelationskoeffizienten r .

$$\text{Determinationskoeffizient}(R^2) = r^2$$

Korrelationskoeffizient nach Pearson: Ein Beispiel

Um ein Gefühl dafür zu bekommen, berechnen wir den Pearson-Produkt-Moment-Korrelationskoeffizienten in einem Beispiel. Dazu laden wir den `students` Datensatz in unseren Arbeitsbereich (Sie können die Datei `students.csv` [hier](#) herunterladen).

```
# Einlesen der Daten als Dataframe
students = pd.read_csv("../data/students.csv", index_col=0)
```

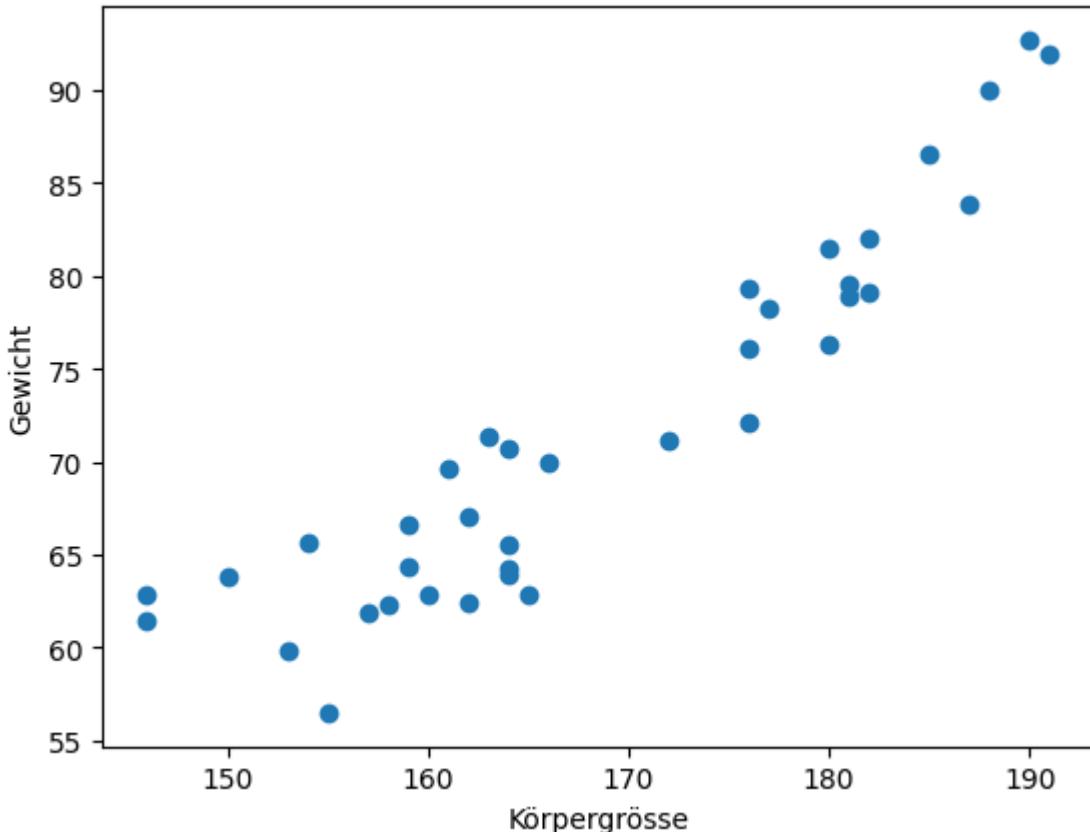
Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

In diesem Beispiel wird die **lineare Beziehung zwischen dem Gewicht und der Größe der Studenten** untersucht. Dazu wählen wir zufällig 37 Studenten aus und extrahieren die Variablen `height` und `weight` aus dem Datensatz.

```
# Wähle zufällige Stichprobe n=37 aus ohne doppelt auszuwählen
students_sample = students.sample(37, replace=False, random_state=1)

# Lese height und weight der Probe aus dem dataframe in Listen ein
height_sample = students_sample["height"]
weight_sample = students_sample["weight"]
```

```
plt.scatter(height_sample, weight_sample, marker="o")
plt.xlabel("Körpergrösse")
plt.ylabel("Gewicht")
plt.show()
```



Das Streudiagramm zeigt, dass eine lineare Beziehung zwischen den beiden betrachteten Variablen besteht. Für diese Übung berechnen wir den linearen Korrelationskoeffizienten zunächst von Hand und wenden dann die `np.corrcoef()`-Funktion in Python an. Wir erinnern uns an die Gleichung von oben

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{s_{xy}}{s_x s_y}$$

```
s_xy = sum(
    (height_sample - height_sample.mean()) * (weight_sample - weight_sample.mean())
)
sx_sy = np.sqrt(sum((height_sample - height_sample.mean()) ** 2)) * np.sqrt(
    sum((weight_sample - weight_sample.mean()) ** 2)
)

s_xy / sx_sy
```

```
np.float64(0.9295600715108283)
```

Zur Überprüfung des Ergebnisses berechnen wir das Verhältnis der Kovarianz von x und y und die Standardabweichung von x und y .

$$r = \frac{s_{xy}}{s_x s_y}$$

```
# Berechne Kovarianz zwischen height_sample und weight_sample
s_xy = np.cov(height_sample, weight_sample)[0, 1]

# Berechne Standardabweichung für height_sample und weight_sample
std_height_sample = np.std(height_sample, ddof=1)
std_weight_sample = np.std(weight_sample, ddof=1)

# Berechne Pearsonkorrelationskoeffizienten
s_xy / (std_height_sample * std_weight_sample)
```

```
np.float64(0.929560071510828)
```

Zum Schluss wenden wir die Funktion `np.corrcoef` an, die im `numpy` Modul enthalten ist.

```
np.corrcoef(weight_sample, height_sample)[0, 1]
```

```
np.float64(0.9295600715108281)
```

Perfekt! Die vier Berechnungen ergeben genau das gleiche Ergebnis! Der lineare Korrelationskoeffizient beträgt $r = 0,9296$. Daraus können wir schließen, dass es einen starken linearen Zusammenhang zwischen der Größe und dem Gewicht eines Studenten gibt.

Natürlich ist eine Korrelationsanalyse nicht auf zwei Variablen beschränkt. Dank statistischer Softwarepakete, wie Python, können wir eine paarweise Korrelationsanalyse für mehr als zwei Variablen durchführen. Lassen Sie uns zunächst den Datensatz vorbereiten. Für eine bessere Visualisierung ziehen wir 100 zufällig ausgewählte Studenten aus dem `students` Datensatz. Dann wählen wir eine numerische Variablen aus, um eine Korrelationsanalyse durchzuführen.

```
# Wähle zufällige Stichprobe n=100 aus ohne Zurücklegen auszuwählen
students_sample = students.sample(100, replace=False, random_state=42)
# Wähle nur numerische Variablen aus
students_sample = students_sample.select_dtypes("number")
students_sample.sample(10)
```

	age	height	weight	nc_score	score1	score2	online_tutorial	graduated		
stud_id										
163772	23	163	65.4	1.90	76.0	67.0	0	1	26	
580061	23	177	71.4	1.99	60.0	60.0	1	0		
210967	18	159	67.1	1.15	57.0	49.0	1	0		
855463	23	157	66.0	3.41	NaN	NaN	0	0		
523713	21	194	91.5	1.62	NaN	NaN	0	0		
973462	21	167	63.3	1.36	68.0	64.0	1	0		
112287	19	176	76.1	2.68	52.0	55.0	0	1	46	
133917	22	173	71.7	3.50	NaN	NaN	0	0		
567950	20	182	79.7	1.37	59.0	58.0	1	1	42	
591911	21	158	63.7	1.10	61.0	72.0	1	0		

Offensichtlich enthalten einige Variablen fehlende Werte, die als *NaN* bezeichnet werden. Wir können diese Werte aus der Analyse ausschließen, indem wir die Methode `dropna()` hinzufügen.

```
# Entferne Spalten die NaN enthalten
students_sample_clean = students_sample.dropna(axis=0, how="any")
students_sample_clean.sample(10)
```

	age	height	weight	nc_score	score1	score2	online_tutorial	graduated	
stud_id									
324623	19	157	66.1	1.75	70.0	69.0	1	1	37
112287	19	176	76.1	2.68	52.0	55.0	0	1	46
648227	24	187	91.0	1.94	51.0	50.0	0	1	41
729441	34	167	70.4	1.13	60.0	57.0	1	1	33
601421	24	197	99.9	2.72	64.0	70.0	1	1	40
111998	20	174	75.8	2.15	64.0	72.0	1	1	37
163772	23	163	65.4	1.90	76.0	67.0	0	1	28
193855	20	181	80.1	2.20	50.0	53.0	1	1	25
397699	23	172	69.4	1.83	69.0	67.0	0	1	37
462072	23	172	76.5	1.25	64.0	68.0	0	1	47

Die Methode `corr()` liefert eine schöne Tabelle, auch **Korrelationsmatrix** genannt, mit den paarweisen Pearson'schen Korrelationskoeffizienten.

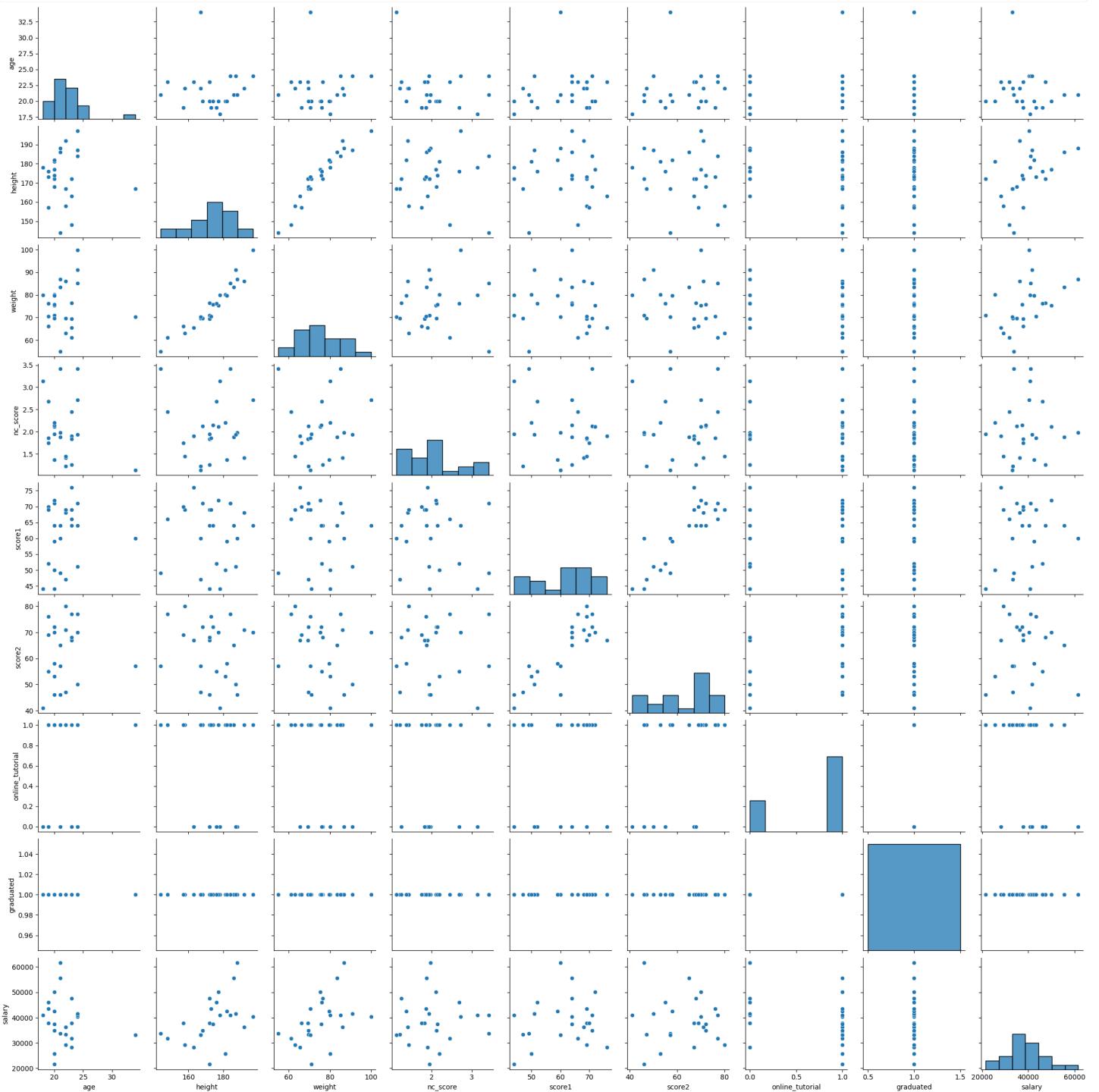
```
students_sample_clean.corr()
```

	age	height	weight	nc_score	score1	score2	online_tutorial
age	1.000000	-0.021219	0.046222	-0.277334	0.132991	0.073143	C
height	-0.021219	1.000000	0.955495	-0.031163	-0.041360	-0.181769	-C
weight	0.046222	0.955495	1.000000	0.073386	-0.078268	-0.176928	-C
nc_score	-0.277334	-0.031163	0.073386	1.000000	-0.185161	-0.037330	-C
score1	0.132991	-0.041360	-0.078268	-0.185161	1.000000	0.860207	C
score2	0.073143	-0.181769	-0.176928	-0.037330	0.860207	1.000000	C
online_tutorial	0.037136	-0.155009	-0.165820	-0.048404	0.131912	0.381767	1
graduated	NaN						
salary	-0.146958	0.467934	0.473708	0.037957	0.165573	-0.038099	-C

Eine Tabelle ist eine schöne Darstellung für eine Korrelationsanalyse, aber eine Abbildung würde natürlich die Interpretierbarkeit verbessern. Das Modul [seaborn](#) bietet die Funktion [pairplot](#) zur Darstellung von Korrelationsmatrizen.

```
# Erzeuge Pair Plot
sns.pairplot(students_sample_clean)
```

<seaborn.axisgrid.PairGrid at 0x79624dfb3d30>



Es fällt sofort auf, dass die Mehrzahl der Variablen nicht linear korreliert zu sein scheint. Im Gegensatz dazu scheinen die Variablenpaare `height` und `weight` sowie `score1` und `score2` positiv korreliert zu sein.

Rangkorrelationskoeffizient nach Spearman

Der Rangkorrelationskoeffizient nach Spearman ([] s.133), auch bekannt als Spearmans ρ , ist ein nichtparametrischer Rangkorrelationskoeffizient. Er wurde von [Charles Spearman](#) entwickelt und ist eine Alternative zum Produkt-Moment-Korrelationskoeffizienten von Pearson. Der Spearman ρ Rangkorrelationskoeffizient wird für Stichprobendaten mit r_s und für Grundgesamtheitsdaten mit ρ_s bezeichnet (Mann 2012). Der Korrelationskoeffizient bewertet die monotone Beziehung zwischen zwei Variablen und liegt zwischen -1 und 1 . Er beschreibt die lineare Korrelation zwischen den Rängen der Daten zu den Variablen x und y . Die Spearman-Korrelation ist hoch, wenn die Variablen einen ähnlichen Rang haben, und niedrig, wenn die Variablen einen ungleichen Rang haben.

Um r_s zu berechnen, werden die Daten für jede Variable, x und y , getrennt eingestuft. Die Differenz zwischen jedem Paar von Rängen bezeichnen wir mit d . Für eine gegebene bivariate Sequenz $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ ist Spearmans r_s gegeben durch

$$r_s = 1 - \frac{6 \sum_{i=1}^n (r_{xi} - r_{yi})^2}{n(n^2 - 1)}$$

wobei $r_{xi} = \text{Rank}(x_i)$, $r_{yi} = \text{Rank}(y_i)$, und n die Stichprobengröße ist.

Im Gegensatz zum linearen Korrelationskoeffizienten von Pearson ist der lineare Koeffizient von Spearman sowohl für quantitative als auch für ordinale Variablen geeignet. Außerdem sind rangbasierte Korrelationen nicht von der Normalverteilungsannahme abhängig und resistenter gegenüber Ausreißern ([] s.135).

Spearman's Rangkorrelationskoeffizient: Ein Beispiel

Betrachten wir ein Beispiel. Die Bevölkerung einer Gemeinde an einem Fluss ist der Meinung, dass der jüngste Anstieg der Abflussspitzen auf die Abholzung durch ein Holzunternehmen in den letzten Jahren zurückzuführen ist. Wir berechnen den Spearman'schen Rangkorrelationskoeffizienten, um festzustellen, ob es eine Korrelation zwischen dem Spitzenabfluss und dem Anteil der abgeholzten Fläche im Wassereinzugsgebiet gibt (Daten geändert nach [McCuen 2003, S. 112](#)).

Jahr	Abflussspitzen ($m^3 s^{-1}$)	abgeholzten Fläche (%)
1982	227	53
1983	249	56
1984	210	57
1985	190	58
1986	314	55
1987	345	54
1988	161	51
1989	266	50
1990	402	49
1991	215	47
1992	164	46
1993	405	44
1994	328	43
1995	294	42

Lassen Sie uns unsere Datenvektoren konstruieren. Wir ordnen die Abflusswerte der Variablen [Q](#) und die abgeholzte Fläche der Variablen [logged](#) zu.

```

# Erzeuge Listen Q, logged
q = [227, 249, 210, 190, 314, 345, 161, 266, 402, 215, 164, 405, 328, 294]
logged = [53, 56, 57, 58, 55, 54, 51, 50, 49, 47, 46, 44, 43, 42]

# Erzeuge Dataframe aus Q und logged
df = pd.DataFrame({"Q": q, "Logged": logged})
df

```

	Q	Logged
0	227	53
1	249	56
2	210	57
3	190	58
4	314	55
5	345	54
6	161	51
7	266	50
8	402	49
9	215	47
10	164	46
11	405	44
12	328	43
13	294	42

Zunächst berechnen wir den Spearman'schen Rangkorrelationskoeffizienten per Hand. Erinnern Sie sich an die Gleichung

$$r_s = 1 - \frac{6 \sum_{i=1}^n (r_{xi} - r_{yi})^2}{n(n^2 - 1)}$$

wobei $r_{xi} = \text{Rang}(x_i)$, $r_{yi} = \text{Rang}(y_i)$, und n der Stichprobenumfang ist. Wir verwenden die Methode `rank()`, um den Rang für die Werte jeder Variablen zu berechnen.

```
rs = 1 - (
  6
  * sum((df.rank()["Q"] - df.rank()["Logged"]) ** 2)
  / (df.shape[0] * (df.shape[0] ** 2 - 1)))
)
rs
```

```
-0.34065934065934056
```

Alternativ können wir auch die Funktion `corr()` in Python verwenden und das Argument `method = 'spearman'` hinzufügen.

```
df.corr(method="spearman")
```

	Q	Logged
Q	1.000000	-0.340659
Logged	-0.340659	1.000000

Der Rangkorrelationskoeffizient von Spearman ist nichts anderes als der lineare Korrelationskoeffizient von Pearson für die gerankten Daten. Das Ergebnis der folgenden Codezelle sollte mit den vorherigen Ergebnissen übereinstimmen.

```
df.rank().corr()
```

	Q	Logged
Q	1.000000	-0.340659
Logged	-0.340659	1.000000

Perfekt, wir erhielten das gleiche Ergebnis durch alle drei Berechnungen, die einen Spearman's Rangkorrelationskoeffizienten von $r_s = -0,34$ ergeben. Die Ergebnisse deuten darauf hin, dass es keine bis eine schwache negative Korrelation zwischen dem Spitzenabfluss und der Einschlagsfläche gibt. Mit anderen Worten, der Abfluss nimmt tendenziell ab, wenn die Abholzungsfläche zunimmt. Die Wahrnehmung der Bevölkerung kann also durch unsere statistische Analyse nicht bestätigt werden. Es wird empfohlen, einen statistischen Test durchzuführen, um festzustellen, ob das Ergebnis statistisch signifikant ist oder ob die Abweichung nur zufällig ist. Weitere Informationen finden Sie in den Abschnitten über *Hypothesentests*.

Kontingenzkoeffizient

Der **Kontingenzkoeffizient** C ist ein χ^2 -basiertes Maß für die Assoziation von kategorialen Daten. Er stützt sich auf den [\$\chi^2\$ -Test auf Unabhängigkeit](#). Die χ^2 Statistik ermöglicht es zu beurteilen, ob eine statistische Beziehung zwischen den Variablen in den [Kontingenztabellen](#) (auch bekannt als Kontingenztafeln oder Kreuztabellen) besteht oder nicht. In dieser Art von Tabellen wird die Verteilung der Variablen in einem Matrixformat dargestellt.

Für die Berechnung des **Kontingenzkoeffizienten** C zu berechnen, müssen wir die χ^2 -Statistik im Voraus bestimmen.

Berechnung der χ^2 -Statistik

Die χ^2 -Statistik ist gegeben durch $\chi^2 = \sum \frac{(O-E)^2}{E}$,

wobei O für die beobachtete Häufigkeit und E für die erwartete Häufigkeit steht. Bitte beachten Sie, dass $\frac{(O-E)^2}{E}$ für jede Zelle ausgewertet und dann aufsummiert wird.

Anhand eines Beispiels soll die Berechnung der χ^2 -Statistik auf der Grundlage kategorialer Beobachtungsdaten näher erläutert werden. Nehmen wir eine Prüfung am Ende des Semesters. Es gibt drei Gruppen von Studierenden: Die Studierenden haben entweder bestanden, nicht bestanden oder nicht an der Prüfung teilgenommen. Außerdem wird die Anzahl der Übungen, die jeder einzelne Studierende während des Semesters absolviert hat, in vier Gruppen eingeteilt: Keine, weniger als die Hälfte ($< 0,5$), mehr als die Hälfte ($\geq 0,5$), alle.

Die resultierende Kontingenztabelle sieht wie folgt aus:

	Keine	$< 0,5$	$\geq 0,5$	Alle
erfolgreich	12	13	24	14
nicht erfolgreich	22	11	8	6
nicht teilgenommen	11	14	6	7

Zunächst konstruieren wir ein `dataframe` Objekt und geben ihm den Namen `obs`, um uns daran zu erinnern, dass diese Daten der beobachteten Häufigkeit entsprechen:

```
# Erzeuge neues Dataframe obs
col1 = ["erfolgreich", "nicht erfolgreich", "nicht teilgenommen"]
col2 = [12, 22, 11]
col3 = [13, 11, 14]
col4 = [24, 8, 6]
col5 = [14, 6, 7]

obs = pd.DataFrame(
    {"Keine": col2, "<0.5": col3, ">0.5": col4, "Alle": col5}, index=col1
)
obs
```

	Keine	<0.5	>0.5	Alle
erfolgreich	12	13	24	14
nicht erfolgreich	22	11	8	6
nicht teilgenommen	11	14	6	7

Perfekt, jetzt haben wir eine ordentliche Darstellung unserer Daten in Python. Allerdings fehlt noch ein Teil, um eine Kontingenztabelle zu vervollständigen. Die Zeilensummen und Spaltensummen.

Es gibt mehrere Möglichkeiten, die Zeilen- und Spaltensummen in Python zu berechnen. Unter anderem kann man die Funktion `sum(df, axis=0,1)` verwenden. Die Funktion benötigt zwei Argumente, die Daten und eine Zahl, die angibt, ob die Daten zeilen- oder spaltenweise vorliegen.

```
# Berechne Zeilen/Spaltensumme
columnsum = obs.sum(axis=0).values
rowsum = obs.sum(axis=1).values

print(f"Spaltensummen: {columnsum}")
print(f"Zeilensummen: {rowsum}")
```

```
Spaltensummen: [45 38 38 27]
Zeilensummen: [63 47 38]
```

Setzt man alle Teile zusammen, sieht die Kontingenztabelle wie folgt aus:

	Keine	< 0,5	$\geq 0,5$	Alle	Zeilensumme
erfolgreich	12	13	24	14	63
nicht erfolgreich	22	11	8	6	47
nicht teilgenommen	11	14	6	7	38
Spaltensumme	45	38	38	27	

Toll, jetzt haben wir eine Tabelle mit den beobachteten Häufigkeiten. Im nächsten Schritt berechnen wir die erwarteten Häufigkeiten. Um die erwarteten Häufigkeiten (E) zu berechnen, wenden wir diese Gleichung an:

$$E = \frac{R \times C}{n},$$

wobei R die Zeilensumme, C die Spaltensumme und n der Stichprobenumfang ist. Bitte beachten Sie, dass wir die erwartete Häufigkeit für jeden einzelnen Tabelleneintrag berechnen müssen, d. h. wir müssen $3 \times 4 = 12$ Berechnungen durchführen. Auch hier bietet Python mehrere Möglichkeiten, diese Aufgabe zu lösen. Man kann eine verschachtelte for-Schleife bauen, um jede Zelle zu durchlaufen und die Berechnungen schrittweise durchzuführen, das ist definitiv in Ordnung! Wir können aber auch die vektorisierte und damit viel schnellere Funktion `outer()` in Kombination mit der Funktion `sum()` verwenden. Wir weisen das Ergebnis einer Variablen mit der Bezeichnung `exp` zu, um uns daran zu erinnern, dass diese Tabelle den erwarteten Häufigkeiten entspricht.

```
exp = np.outer(columnsum, rowsum) / sum(columnsum)
exp
```

```
array([[19.15540541, 14.29054054, 11.55405405],
       [16.17567568, 12.06756757, 9.75675676],
       [16.17567568, 12.06756757, 9.75675676],
       [11.49324324, 8.57432432, 6.93243243]])
```

Im nächsten Schritt erstellen wir aus den Ergebnissen einen Dataframe der dieselbe Spalten- und Zeilenanzahl, derselben Index und dieselben Spaltennamen hat wie der `obs` Dataframe.

```
exp = pd.DataFrame(np.transpose(exp), index=obs.index, columns=obs.columns)
exp
```

	Keine	<0.5	>0.5	Alle
erfolgreich	19.155405	16.175676	16.175676	11.493243
nicht erfolgreich	14.290541	12.067568	12.067568	8.574324
nicht teilgenommen	11.554054	9.756757	9.756757	6.932432

Im nächsten Schritt berechnen wir schließlich die χ^2 -Statistik. Erinnern Sie sich an die Gleichung:

$$\chi^2 = \sum \frac{(O - E)^2}{E},$$

wobei O für die beobachtete Häufigkeit und E für die erwartete Häufigkeit steht.

```
# Berechne chisqVal
chisqVal = np.sum(((obs.values - exp.values) ** 2) / exp.values)
chisqVal
```

```
np.float64(17.344387665138406)
```

Berechnung des Kontingenzkoeffizienten C

Der Kontingenzkoeffizient, bezeichnet als C^* , passt die χ^2 -Statistik um den Stichprobenumfang (n) an. Er kann wie folgt geschrieben werden

$$C^* = \sqrt{\frac{\chi^2}{n + \chi^2}},$$

wobei χ^2 der χ^2 -Statistik entspricht und n die Anzahl der Beobachtungen bezeichnet. Wenn keine Beziehung zwischen zwei Variablen besteht, ist $C^* = 0$. Der Kontingenzkoeffizient C^* kann Werte > 1 nicht überschreiten, aber der Kontingenzkoeffizient kann kleiner als 1 sein, selbst wenn zwei Variablen perfekt miteinander verbunden sind. Dies ist nicht so wünschenswert, daher wird C^* so

angepasst, dass er maximal 1 erreicht, wenn in einer Tabelle mit einer beliebigen Anzahl von Zeilen und Spalten ein vollständiger Zusammenhang besteht. Daher berechnen wir C_{max}^* , das ist

$$C_{max}^* = \sqrt{\frac{k-1}{k}},$$

wobei k die Anzahl der Zeilen oder die Anzahl der Spalten ist, je nachdem, welcher Wert kleiner ist, $k = \min(\text{Zeilen}, \text{Spalten})$. Dann wird der bereinigte Kontingenzkoeffizient wie folgt berechnet

$$C = \frac{C^*}{C_{max}^*} = \sqrt{\frac{k \cdot \chi^2}{(k-1)(n + \chi^2)}}$$

Im obigen Abschnitt wurde die χ^2 -Statistik der Variablen `chisqVal` zugeordnet und als 17,3443877 berechnet. Nun setzen wir diesen Wert in die Gleichung für den Kontingenzkoeffizienten C^* ein.

```
C_star = np.sqrt(chisqVal / (sum(columnsum) + chisqVal))  
C_star
```

```
np.float64(0.3238804670641156)
```

Der Kontingenzkoeffizient C^* beläuft sich auf 0,3238805.

Nun wenden wir die Gleichung für den bereinigten Kontingenzkoeffizienten, C .

```
k = [obs.shape[0], obs.shape[1]]  
k = min(k)  
k
```

3

```
C_star_max = np.sqrt((k - 1) / k)  
C = C_star / C_star_max  
C
```

```
np.float64(0.39667094098068806)
```

Der bereinigte Kontingenzkoeffizient C wird mit 0,3966709 bewertet. Zur Erinnerung: Der Kontingenzkoeffizient reicht von 0 bis 1. Ein Kontingenzkoeffizient von 0,4 deutet nicht auf einen starken Zusammenhang zwischen den Prüfungsergebnissen und der Bereitschaft der Studierenden hin, die Übungen während des Semesters zu bearbeiten.

Übungsaufgaben

Die Pandas Bibliothek

Die [Pandas-Bibliothek](#) wurde 2010 von [Wes McKinney](#) entwickelt. pandas bietet **Datenstrukturen** und **Funktionen** für die Manipulation, Verarbeitung, Bereinigung und Verwertung von Daten. Im Python-Ökosystem ist pandas das modernste Werkzeug für die Arbeit mit tabellarischen oder tabellenähnlichen Daten, bei denen jede Spalte von einem anderen Typ sein kann ([String](#), [numerisch](#), [Datum](#) oder andere). pandas bietet ausgereifte Indizierungsfunktionen, die das Umformen, Zerlegen, Aggregieren und Auswählen von Teilmengen von Daten erleichtern. pandas stützt sich auf andere Pakete, wie [NumPy](#) und [SciPy](#). Außerdem integriert pandas [matplotlib](#) zum Plotten.

Wenn Sie neu im Umgang mit pandas sind, empfehlen wir Ihnen dringend, die sehr gut geschriebenen [pandas-Tutorials](#) zu besuchen, die alle relevanten Abschnitte für neue Benutzer abdecken, um richtig loszulegen.

Nach der Installation (Details finden Sie in der [Dokumentation](#)) wird pandas mit dem kanonischen Alias [pd](#) importiert.

```
import pandas as pd
```

```
import numpy as np
```

Die Pandas-Bibliothek verfügt über zwei bewährte Datenstrukturen: **Series** und **DataFrame**.

- eindimensionales pd.Series-Objekt
- zweidimensionales pd.DataFrame-Objekt

Das `pd.Series` Objekt

Erzeugung von Daten

```
# importiere das random module von numpy
from numpy import random

# setze seed
random.seed(123)
# Erzeuge 26 Zufallszahlen zwischen -10 and 10
my_data = random.randint(low=-10, high=10, size=26)
# Ausgabe
my_data
```

```
array([ 3, -8, -8, -4,  7,  9,  0, -9, -10,  7,  5, -1, -10,
       4, -10,  5,  9,  4, -6, -10,  6, -6,  7, -7, -8, -3])
```

```
type(my_data)
```

```
numpy.ndarray
```

Eine Series ist ein eindimensionales Array-ähnliches Objekt, das ein Array mit Daten und ein zugehöriges Array mit Datenbeschriftungen, genannt Index, enthält. Wir erstellen ein

`pd.Series-Objekt`, indem wir die Funktion `pd.Series()` aufrufen.

```
# Entkommentieren für Dokumentation

# docstring
# ?pd.Series

# source
# ??pd.Series
```

```
# Erzeuge pd.Series Objekt
s = pd.Series(data=my_data)
s
```

```
0      3
1     -8
2     -8
3     -4
4      7
5      9
6      0
7     -9
8     -10
9      7
10     5
11     -1
12     -10
13      4
14     -10
15      5
16      9
17      4
18     -6
19     -10
20      6
21     -6
22      7
23     -7
24     -8
25     -3
dtype: int64
```

```
type(s)
```

```
pandas.core.series.Series
```

pd.Series -Attribute

Python-Objekte im Allgemeinen und die `pd.Series` im Besonderen bieten nützliche objektspezifische Attribute.

Attribut -> `OBJECT.attribute`

Beachten Sie, dass das Attribut ohne Klammern aufgerufen wird

```
s.dtypes
```

```
dtype('int64')
```

```
s.index
```

```
RangeIndex(start=0, stop=26, step=1)
```

Wir können das Attribut `index` verwenden, um einem `pd.Series-Objekt` einen Index zuzuweisen.

Betrachten wir die Buchstaben des Alphabets....

```
import string
letters = string.ascii_uppercase
letters
```

```
'ABCDEFGHIJKLMNPQRSTUVWXYZ'
```

```
s.index = list(letters)
s
```

```
A      3
B     -8
C     -8
D     -4
E      7
F      9
G      0
H     -9
I    -10
J      7
K      5
L     -1
M    -10
N      4
O    -10
P      5
Q      9
R      4
S     -6
T    -10
U      6
V     -6
W      7
X     -7
Y     -8
Z     -3
dtype: int64
```

pd.Series -Methoden

```
s.sum()
```

```
np.int64(-34)
```

```
s.mean()
```

```
np.float64(-1.3076923076923077)
```

```
s.max()
```

```
np.int64(9)
```

```
s.min()
```

```
np.int64(-10)
```

```
s.median()
```

```
np.float64(-2.0)
```

```
s.quantile(q=0.5)
```

```
np.float64(-2.0)
```

```
s.quantile(q=[0.25, 0.5, 0.75])
```

```
0.25    -8.0
0.50    -2.0
0.75     5.0
dtype: float64
```

Elementweise Arithmetik

Eine sehr nützliche Eigenschaft von `pd.Series`-Objekten ist, dass wir arithmetische Operationen *elementweise* anwenden können.

```
s + 10
# s*0.1
# 10/s
# s**2
# (2+s)**1**3
# s+s
```

```
A    13
B     2
C     2
D     6
E    17
F    19
G    10
H     1
I     0
J    17
K    15
L     9
M     0
N    14
O     0
P    15
Q    19
R    14
S     4
T     0
U    16
V     4
W    17
X     3
Y     2
Z     7
dtype: int64
```

Auswahl und Indizierung

Eine weitere wichtige Datenoperation ist die Indizierung und Auswahl bestimmter Teilmengen des Datenobjekts. pandas verfügt über einen [sehr umfangreichen Satz](#) von Methoden für diese Art von Aufgaben.

In der einfachsten Form indizieren wir eine Reihe numpy-ähnlich, indem wir den `[]` Operator verwenden, um einen bestimmten [Index](#) der Reihe auszuwählen.

```
s
```

```
A      3
B     -8
C     -8
D     -4
E      7
F      9
G      0
H     -9
I    -10
J      7
K      5
L     -1
M    -10
N      4
O    -10
P      5
Q      9
R      4
S     -6
T    -10
U      6
V     -6
W      7
X     -7
Y     -8
Z     -3
dtype: int64
```

```
s[3]
```

```
/tmp/ipykernel_17384/3172538273.py:1: FutureWarning: Series.__getitem__ treating key
s[3]
```

```
np.int64(-4)
```

```
s[2:6]
```

```
C     -8
D     -4
E      7
F      9
dtype: int64
```

```
s["C"]
```

```
np.int64(-8)
```

```
s["C":"K"]
```

```
C      -8
D      -4
E       7
F       9
G       0
H      -9
I     -10
J       7
K       5
dtype: int64
```

Das `pd.DataFrame`-Objekt

Die primäre Datenstruktur von Pandas ist der `DataFrame`. Es handelt sich um eine zweidimensionale, größenveränderliche, potenziell heterogene tabellarische Datenstruktur mit Zeilen- und Spaltenbeschriftungen. Arithmetische Operationen richten sich sowohl auf Zeilen- als auch auf Spaltenbeschriftungen aus. Grundsätzlich kann man sich den `DataFrame` als einen `dictionary`-artigen Container für Seriesobjekte vorstellen.

Erzeugen eines `DataFrame`-Objekts von Grund auf

pandas erleichtert den Import verschiedener Datentypen und -quellen, aber für dieses Tutorial erzeugen wir ein DataFrame-Objekt von Grund auf.

Quelle: <http://duelingdata.blogspot.de/2016/01/the-beatles.html>

```
df = pd.DataFrame(
    {
        "id": range(1, 5),
        "Name": ["John", "Paul", "George", "Ringo"],
        "Last Name": ["Lennon", "McCartney", "Harrison", "Star"],
        "dead": [True, False, True, False],
        "year_born": [1940, 1942, 1943, 1940],
        "no_of_songs": [62, 58, 24, 3],
    }
)
df
```

	id	Name	Last Name	dead	year_born	no_of_songs
0	1	John	Lennon	True	1940	62
1	2	Paul	McCartney	False	1942	58
2	3	George	Harrison	True	1943	24
3	4	Ringo	Star	False	1940	3

pd.DataFrame -Attribute

```
df.dtypes
```

```
id          int64
Name        object
Last Name   object
dead         bool
year_born   int64
no_of_songs int64
dtype: object
```

```
# Achse 0
df.columns
```

```
Index(['id', 'Name', 'Last Name', 'dead', 'year_born', 'no_of_songs'], dtype='object')
```

```
# Achse 1  
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

pd.DataFrame -Methoden

Verschaffen Sie sich einen schnellen Überblick über den Datensatz

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4 entries, 0 to 3  
Data columns (total 6 columns):  
 #   Column      Non-Null Count  Dtype     
 ---  --          --          --          --  
 0   id          4 non-null      int64    
 1   Name         4 non-null      object   
 2   Last Name   4 non-null      object   
 3   dead         4 non-null      bool     
 4   year_born   4 non-null      int64    
 5   no_of_songs  4 non-null      int64    
 dtypes: bool(1), int64(3), object(2)  
 memory usage: 292.0+ bytes
```

```
df.describe()
```

	id	year_born	no_of_songs
count	4.000000	4.00	4.000000
mean	2.500000	1941.25	36.750000
std	1.290994	1.50	28.229712
min	1.000000	1940.00	3.000000
25%	1.750000	1940.00	18.750000
50%	2.500000	1941.00	41.000000
75%	3.250000	1942.25	59.000000
max	4.000000	1943.00	62.000000

```
df.describe(include="all")
```

	id	Name	Last Name	dead	year_born	no_of_songs
count	4.000000	4	4	4	4.00	4.000000
unique	NaN	4	4	2	NaN	NaN
top	NaN	John	Lennon	True	NaN	NaN
freq	NaN	1	1	2	NaN	NaN
mean	2.500000	NaN	NaN	NaN	1941.25	36.750000
std	1.290994	NaN	NaN	NaN	1.50	28.229712
min	1.000000	NaN	NaN	NaN	1940.00	3.000000
25%	1.750000	NaN	NaN	NaN	1940.00	18.750000
50%	2.500000	NaN	NaN	NaN	1941.00	41.000000
75%	3.250000	NaN	NaN	NaN	1942.25	59.000000
max	4.000000	NaN	NaN	NaN	1943.00	62.000000

Index in die Variable **id** ändern

```
df
```

	id	Name	Last Name	dead	year_born	no_of_songs
0	1	John	Lennon	True	1940	62
1	2	Paul	McCartney	False	1942	58
2	3	George	Harrison	True	1943	24
3	4	Ringo	Star	False	1940	3

```
df.set_index("id")
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

```
df
```

	id	Name	Last Name	dead	year_born	no_of_songs
0	1	John	Lennon	True	1940	62
1	2	Paul	McCartney	False	1942	58
2	3	George	Harrison	True	1943	24
3	4	Ringo	Star	False	1940	3

Beachten Sie, dass sich nichts geändert hat!!

Aus Gründen der Speicher- und Berechnungseffizienz gibt [Pandas](#) eine Ansicht des Objekts zurück, keine Kopie. Wenn wir also eine dauerhafte Änderung vornehmen wollen, müssen wir das Objekt einer Variablen zuweisen/neu zuordnen:

```
df = df.set_index("id")
```

oder einige Methoden haben das Argument `inplace=True`:

```
df.set_index("id", inplace=True)
```

```
df = df.set_index("id")
```

```
df
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

Arithmetische Methoden

```
df
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

```
df.sum(axis=0)
```

```
Name          JohnPaulGeorgeRingo
Last Name    LennonMcCartneyHarrisonStar
dead                   2
year_born      7765
no_of_songs     147
dtype: object
```

```
df.sum(axis=1)
```



```
-----  
TypeError Traceback (most recent call last)  
Cell In[40], line 1  
----> 1 df.sum(axis=1)  
  
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa  
11661 @doc(make_doc("sum", ndim=2))  
11662 def sum(  
11663     self,  
11664     (...)  
11665     **kwargs,  
11666     ):  
> 11670     result = super().sum(axis, skipna, numeric_only, min_count, **kwargs)  
11671     return result.__finalize__(self, method="sum")  
  
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa  
12498 def sum(  
12499     self,  
12500     axis: Axis | None = 0,  
12501     (...)  
12502     **kwargs,  
12503     ):  
> 12506     return self._min_count_stat_function(  
12507         "sum", nanops.nansum, axis, skipna, numeric_only, min_count, **kwargs  
12508     )  
  
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa  
12486 elif axis is lib.no_default:  
12487     axis = 0  
> 12489 return self._reduce(  
12490     func,  
12491     name=name,  
12492     axis=axis,  
12493     skipna=skipna,  
12494     numeric_only=numeric_only,  
12495     min_count=min_count,  
12496     )  
  
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa  
11558     df = df.T  
11560 # After possibly _get_data and transposing, we are now in the  
11561 # simple case where we can use BlockManager.reduce  
> 11562 res = df._mgr.reduce(blk_func)  
11563 out = df._constructor_from_mngr(res, axes=res.axes).iloc[0]  
11564 if out.dtype is not None and out.dtype != "boolean":  
  
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa  
1498 res_blocks: list[Block] = []  
1499 for blk in self.blocks:  
> 1500     nbs = blk.reduce(func)  
1501     res_blocks.extend(nbs)  
1503 index = Index([None]) # placeholder
```

```

File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
398 @final
399 def reduce(self, func) -> list[Block]:
400     # We will apply the function and reshape the result into a single-row
401     # Block with the same mgr_locs; squeezing will be done at a higher level
402     assert self.ndim == 2
--> 404     result = func(self.values)
406     if self.values.ndim == 1:
407         res_values = result

File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
11479         return np.array([result])
11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
81     raise TypeError(
82         f"reduction operation '{f_name}' not allowed for this dtype"
83     )
84 try:
--> 85     return f(*args, **kwargs)
86 except ValueError as e:
87     # we want to transform an object array
88     # ValueError message to the more typical TypeError
89     # e.g. this is normally a disallowed function on
90     # object arrays that contain strings
91     if is_object_dtype(args[0]):


File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
401 if datetimelike and mask is None:
402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
406 if datetimelike:
407     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
474     results = [func(x, **kwargs) for x in arrs]
475     return np.array(results)
--> 477 return func(values, axis=axis, **kwargs)

File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
643 elif dtype.kind == "m":
644     dtype_sum = np.dtype(np.float64)
--> 646 the_sum = values.sum(axis, dtype=dtype_sum)
647 the_sum = _maybe_null_out(the_sum, axis, mask, values.shape, min_count=min_c
649 return the_sum

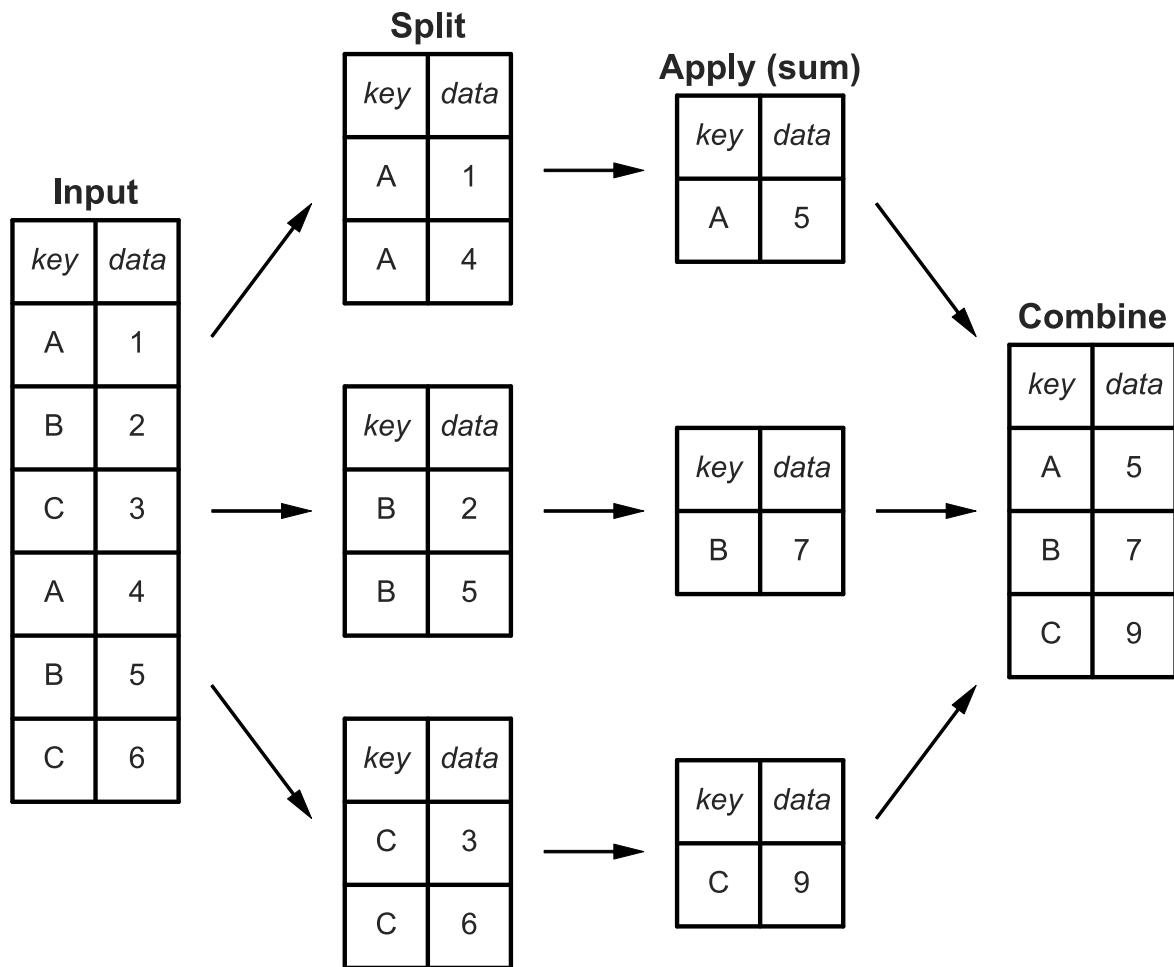
File ~/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10/site-pa
51 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
52         initial=_NoValue, where=True):
--> 53     return umr_sum(a, axis, dtype, out, keepdims, initial, where)

TypeError: can only concatenate str (not "bool") to str

```

Groupby -Methode

[Hadley Wickham 2011: The Split-Apply-Combine Strategy for Data Analysis, Journal of Statistical Software, 40\(1\)](#)



df

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

```
df.groupby("dead")
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fde143b30d0>
```

```
df.groupby("dead").sum()
```

	year_born	no_of_songs
dead		
False	3882	61
True	3883	86

```
df.groupby("dead")["no_of_songs"].sum()
```

```
dead
False    61
True     86
Name: no_of_songs, dtype: int64
```

```
df.groupby("dead")["no_of_songs"].mean()
```

```
dead
False    30.5
True     43.0
Name: no_of_songs, dtype: float64
```

```
df.groupby("dead")["no_of_songs"].agg(["mean", "max", "min", "sum"])
```

	mean	max	min	sum
dead				
False	30.5	58	3	61
True	43.0	62	24	86

Familie von `apply/map`-Methoden

- `apply` arbeitet zeilenweise (`axis=0`, Standard) / spaltenweise (`axis=1`) auf einem `DataFrame`
- `applymap` arbeitet elementweise auf einem `DataFrame`
- `map` arbeitet elementweise mit einer `Series`.

```
df
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

```
# (axis=0, default)
df[["Name", "Last Name"]].apply(lambda x: x.sum())
```

```
Name          JohnPaulGeorgeRingo
Last Name    LennonMcCartneyHarrisonStar
dtype: object
```

```
# (axis=1)
df[["Name", "Last Name"]].apply(lambda x: x.sum(), axis=1)
```

```
id
1      JohnLennon
2      PaulMcCartney
3      GeorgeHarrison
4      RingoStar
dtype: object
```

... vielleicht ein nützlicherer Fall ...

```
df.apply(lambda x: " ".join(x[["Name", "Last Name"]]), axis=1)
```

```
id
1      John Lennon
2      Paul McCartney
3      George Harrison
4      Ringo Star
dtype: object
```

Auswahl und Indizierung

Spaltenindex

```
df["Name"]
```

```
id
1      John
2      Paul
3      George
4      Ringo
Name: Name, dtype: object
```

```
df[["Name", "Last Name"]]
```

	Name	Last Name
id		
1	John	Lennon
2	Paul	McCartney
3	George	Harrison
4	Ringo	Star

```
df.dead
```

```
id
1    True
2    False
3    True
4    False
Name: dead, dtype: bool
```

Zeilenindex

Neben dem `[]`-Operator verfügt Pandas über weitere Indizierungsoperatoren wie `.loc[]` und `.iloc[]`, um nur einige zu nennen.

- `.loc[]` basiert hauptsächlich auf **Bezeichnungen**, kann aber auch mit einem booleschen Array verwendet werden.
- `.iloc[]` basiert in erster Linie auf **Ganzzahlpositionen** (von 0 bis Länge – 1 der Achse), kann aber auch mit einem booleschen Array verwendet werden.

```
df.head(2)
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58

```
df.loc[1]
```

```
Name      John
Last Name  Lennon
dead      True
year_born  1940
no_of_songs 62
Name: 1, dtype: object
```

```
df.iloc[1]
```

```
Name      Paul
Last Name  McCartney
dead      False
year_born  1942
no_of_songs 58
Name: 2, dtype: object
```

Zeilen- und Spaltenindizes

```
df.loc[row, col]
```

```
df.loc[1, "Last Name"]
```

```
'Lennon'
```

```
df.loc[2:4, ["Name", "dead"]]
```

	Name	dead
id		
2	Paul	False
3	George	True
4	Ringo	False

logisches Indizieren

```
df
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58
3	George	Harrison	True	1943	24
4	Ringo	Star	False	1940	3

```
df["no_of_songs"] > 50
```

```
id
1    True
2    True
3   False
4   False
Name: no_of_songs, dtype: bool
```

```
df.loc[df["no_of_songs"] > 50]
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	True	1940	62
2	Paul	McCartney	False	1942	58

```
df.loc[(df["no_of_songs"] > 50) & (df["year_born"] >= 1942)]
```

	Name	Last Name	dead	year_born	no_of_songs
id					
2	Paul	McCartney	False	1942	58

```
df.loc[(df["no_of_songs"] > 50) & (df["year_born"] >= 1942), ["Last Name", "Name"]]
```

	Last Name	Name
id		
2	McCartney	Paul

Manipulation von Spalten, Zeilen und bestimmten Einträgen

Hinzufügen einer Zeile zum Datensatz

```
from numpy import nan

df.loc[5] = ["Mickey", "Mouse", nan, 1928, nan]
df
```

```
/tmp/ipykernel_428947/1527406442.py:3: FutureWarning: Behavior when concatenating bc
df.loc[5] = ["Mickey", "Mouse", nan, 1928, nan]
```

	Name	Last Name	dead	year_born	no_of_songs
id					
1	John	Lennon	1.0	1940	62.0
2	Paul	McCartney	0.0	1942	58.0
3	George	Harrison	1.0	1943	24.0
4	Ringo	Star	0.0	1940	3.0
5	Mickey	Mouse	NaN	1928	NaN

```
df.dtypes
```

```
Name      object
Last Name  object
dead      float64
year_born   int64
no_of_songs float64
dtype: object
```

Beachten Sie, dass sich die Variable `dead` geändert hat. Ihre Werte änderten sich von `True` / `False` zu `1.0` / `0.0`. Folglich änderte sich ihr `dtype` von `bool` zu `float64`.

Hinzufügen einer Spalte zum Datensatz

```
from datetime import datetime
datetime.today()
```

```
datetime.datetime(2023, 4, 28, 8, 51, 24, 137990)
```

```
now = datetime.today().year
now
```

```
2023
```

```
df["age"] = now - df.year_born
df
```

	Name	Last Name	dead	year_born	no_of_songs	age
id						
1	John	Lennon	1.0	1940	62.0	83
2	Paul	McCartney	0.0	1942	58.0	81
3	George	Harrison	1.0	1943	24.0	80
4	Ringo	Star	0.0	1940	3.0	83
5	Mickey	Mouse	NaN	1928	NaN	95

Einen bestimmten Eintrag ändern

```
df.loc[5, "Name"] = "Minnie"
```

```
df
```

	Name	Last Name	dead	year_born	no_of_songs	age
id						
1	John	Lennon	1.0	1940	62.0	83
2	Paul	McCartney	0.0	1942	58.0	81
3	George	Harrison	1.0	1943	24.0	80
4	Ringo	Star	0.0	1940	3.0	83
5	Minnie	Mouse	NaN	1928	NaN	95

Plotten

Die Plotting-Funktionalität in Pandas basiert auf Matplotlib. Es ist recht praktisch, den Visualisierungsprozess mit der grundlegenden Pandas-Darstellung zu beginnen und zu matplotlib zu wechseln, um die Pandas-Visualisierung anzupassen.

plot -Methoden

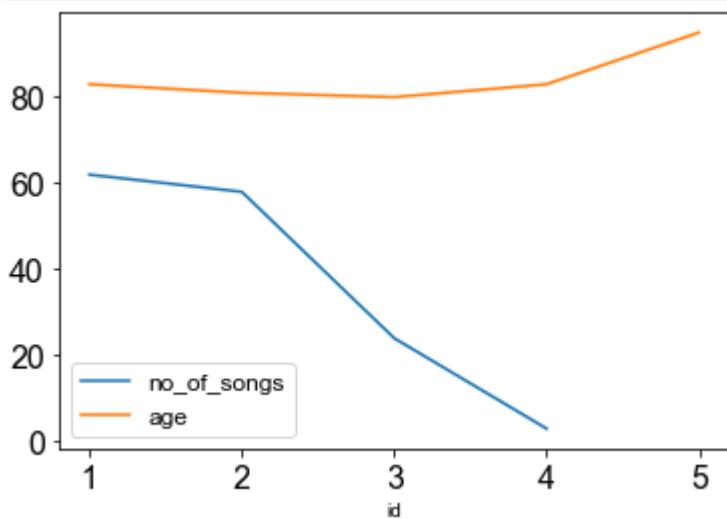
```
# dieser Aufruf bewirkt, dass die Zahlen unter den Codezellen eingezeichnet werden
%matplotlib inline
```

```
df
```

	Name	Last Name	dead	year_born	no_of_songs	age
id						
1	John	Lennon	1.0	1940	62.0	83
2	Paul	McCartney	0.0	1942	58.0	81
3	George	Harrison	1.0	1943	24.0	80
4	Ringo	Star	0.0	1940	3.0	83
5	Minnie	Mouse	NaN	1928	NaN	95

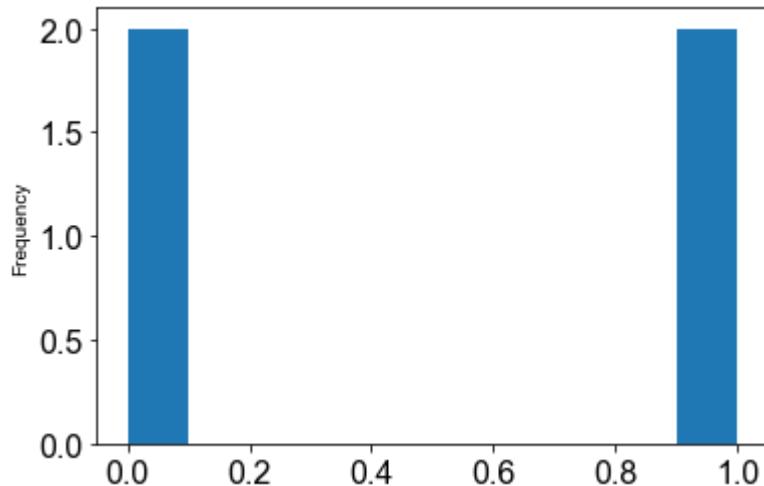
```
df[["no_of_songs", "age"]].plot()
```

```
<Axes: xlabel='id'>
```



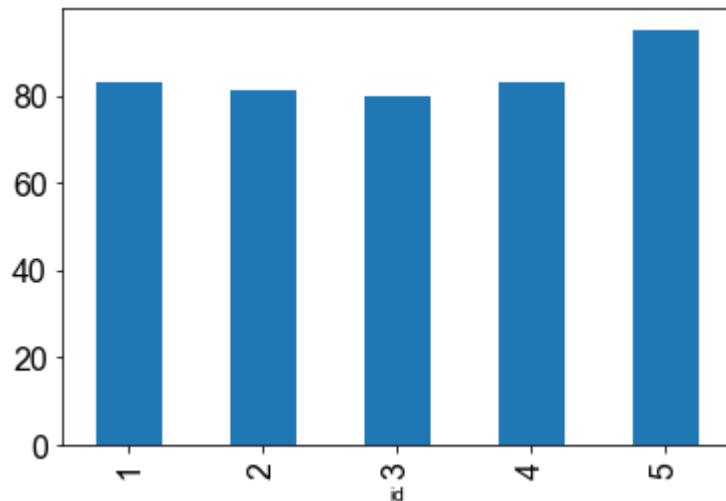
```
df["dead"].plot.hist()
```

```
<Axes: ylabel='Frequency'>
```



```
df["age"].plot.bar()
```

```
<Axes: xlabel='id'>
```



...einige Anmerkungen zum Plotten mit Python

Das Plotten ist ein wesentlicher Bestandteil der Datenanalyse. Die Welt der Python-Visualisierung kann jedoch ein frustrierender Ort sein. Es gibt viele verschiedene Optionen, und die Auswahl der richtigen ist eine Herausforderung. (Wenn Sie sich trauen, werfen Sie einen Blick auf die [Python-Visualisierungslandschaft](#)).

[matplotlib](#) ist wahrscheinlich die bekannteste Python-Bibliothek für 2D-Diagramme. Mit ihr lassen sich plattformübergreifend Zahlen in Publikationsqualität in einer Vielzahl von Formaten und interaktiven Umgebungen erstellen. Allerdings ist matplotlib aufgrund der komplexen Syntax und der Existenz zweier Schnittstellen, einer **MATLAB-ähnlichen Zustandsbasierten Schnittstelle** und einer **objektorientierten Schnittstelle**, schwer zugänglich. Daher gibt es **immer mehr als eine Möglichkeit, eine Visualisierung zu erstellen**. Eine weitere Quelle der Verwirrung ist die Tatsache, dass matplotlib gut in andere Python-Bibliotheken integriert ist, wie z. B. [pandas](#), [seaborn](#), [xarray](#) und andere. Daher gibt es Verwirrung darüber, wann man die reine matplotlib oder ein Tool, das auf matplotlib aufbaut, verwenden sollte.

Wir importieren die `matplotlib`-Bibliothek und das `pyplot`-Modul von matplotlib mit den folgenden kanonischen Befehlen

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

In Bezug auf die Terminologie von matplotlib ist es wichtig zu verstehen, dass die `Figure` das endgültige Bild ist, das eine oder mehrere `Axes` enthalten kann, und dass die `Axes` eine individuelle Darstellung repräsentieren.

Um ein `Figure`-Objekt zu erstellen, rufen wir

```
fig = plt.figure()
```

Ein bequemerer Weg, ein `Figure`-Objekt und ein `Axes`-Objekt auf einmal zu erstellen, ist jedoch der Aufruf

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

Dann können wir das `Axes`-Objekt verwenden, um Daten für die Darstellung hinzuzufügen.

```

import matplotlib.pyplot as plt

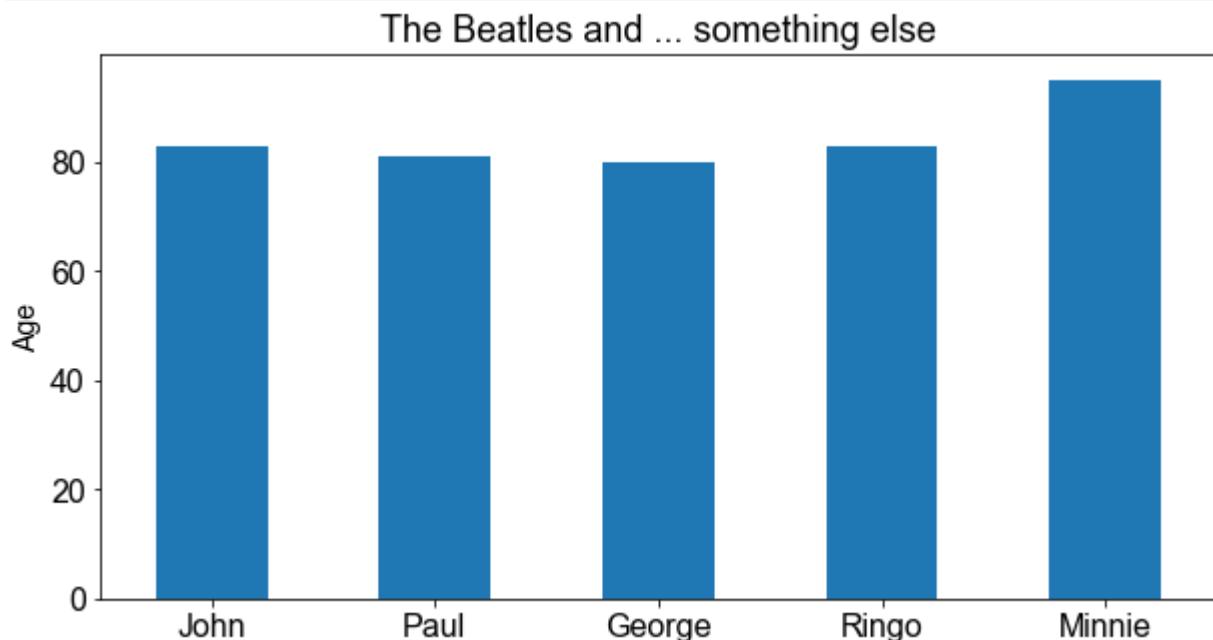
# Erzeuge Figure und Axes Objekt
fig, ax = plt.subplots(figsize=(10, 5))

# plot die Daten und referenzier das Axes Objekt
df["age"].plot.bar(ax=ax)

# Passe das Axes Objekt an
ax.set_xticklabels(df["Name"], rotation=0)
ax.set_xlabel("")
ax.set_ylabel("Age", size=14)
ax.set_title("The Beatles and ... something else", size=18)

```

Text(0.5, 1.0, 'The Beatles and ... something else')



Beachten Sie, dass wir nur an der Oberfläche der Plot-Möglichkeiten mit Pandas kratzen. In der Online-Dokumentation von Pandas ([hier](#)) finden Sie einen umfassenden Überblick.

Übung zu Pandas Dataframes

1. Erstellen Sie einen Pandas Dataframe (`df`) wie vorgegeben.
2. Fügen Sie die Spalte `nc_score` = [2.5, 3, 2.2, 1.0] zum Dataframe `df` hinzu.
3. Wählen Sie alle Daten aus, für die gilt `age` > 23 und speichern Sie die Werte in den neuen Dataframe `df2`.

id	Name	Age
1	John	26
2	Alice	20
3	Mike	21
4	Anne	25

Frage 1 ...

Frage 2 ...

Frage 3 ...

Lösungen

▶ Show code cell content

▶ Show code cell content

▶ Show code cell content

Maße der zentralen Tendenz, Streumaße und Fünf-Punkte Zusammenfassung

1. Berechnen Sie Mittelwert, Standardabweichung, Modalwert des Vektors `data`.

```
data = [10, 1, 17, 0, 14, 2, 11, 1, 4, 10, 5, 1, 1, 99, 47, 16, 3, 4, 9, 11]
```

2. Berechnen Sie die Fünf-Punkte-Zusammenfassung für `data`.

3. Generieren Sie mit der Funktion `np.linspace()` eine x -Achse mit Werten zwischen 0 und 1 und stellen Sie `data` als Streudiagramm dar.

4. Stellen Sie die Daten als Boxplot dar.

5. Bewerten Sie anhand der Ergebnisse mögliche Ausreißer.

Frage 1 ...

Frage 2 ...

Frage 3 ...

Frage 4 ...

Frage 5 ...

Lösungen

► Show code cell content

Arithmetisches Mittel

die die Schreiben Sie eine Funktion (`arithmetic_average`) die den Mittelwert einer Liste/eines Arrays berechnet. Vergleichen Sie Ihre Implementierung mit einer *built-in* Funktion von Python. Verwenden Sie die Funktion auf die folgenden Daten an.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = [-10, -8, -6, -4, -2, 0]
```

```
# Frage 1 ...
```

Lösungen

▶ Show code cell content

Lernziele

Diskrete Zufallsvariablen

- Die Studierenden werden mit diskreten Zufallsvariablen und der Berechnung deren Erwartungswerts und Standardabweichung vertraut gemacht
- Verschiedene Verteilungsarten von Zufallsvariablen wie Binomialverteilung, hypergeometrische Verteilung und Poisson-Verteilung werden erklärt
- Den Studierenden wird anhand von Beispielen die Konzepte von Permutationen und Kombinatorik erläutert
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : diskrete Zufallsvariablen, Erwartungswert und Standardabweichung von diskreten Zufallsvariablen, Binomialverteilung, hypergeometrische Verteilung, Poissonverteilung

Diskrete Zufallsvariablen und ihre Wahrscheinlichkeitsverteilungen

```
import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Eine **Zufallsvariable** ist eine Variable, deren Wert vom Zufall abhängt; dementsprechend ist ihr Wert mit einer Wahrscheinlichkeit verbunden. Generell sind Zufallsvariablen Zuordnungsvorschriften die möglichen Ergebnissen eines Zufallsexperiments einen Zahlenwert zuordnen. Eine **diskrete Zufallsvariable** weist den möglichen Ergebnissen diskrete, also abzählbare Werte zu. Eine **Wahrscheinlichkeitsverteilung** ist eine Auflistung der möglichen Werte und der entsprechenden Wahrscheinlichkeiten einer diskreten Zufallsvariablen, die häufig durch eine Formel dargestellt wird.

Ein Diagramm der Wahrscheinlichkeitsverteilung, das die Wahrscheinlichkeit jedes Wertes, dargestellt durch einen vertikalen Balken, dessen Höhe der Wahrscheinlichkeit entspricht, und die möglichen Werte einer diskreten Zufallsvariablen auf der horizontalen Achse anzeigt, wird **Wahrscheinlichkeitshistogramm** genannt.

Die **Summe der Wahrscheinlichkeiten einer diskreten Zufallsvariablen** für eine beliebige diskrete Zufallsvariable X wird geschrieben als

$$\sum_{i=1}^N P(X = x_i) = 1$$

Bei einer großen Anzahl an voneinander unabhängigen Beobachtungen einer Zufallsvariablen X wird das Wahrscheinlichkeitshistogramm eine Annäherung an die Wahrscheinlichkeitsverteilung für X darstellen ([] s.209-250).

Diskrete Zufallsvariablen - ein Beispiel

Lassen Sie uns das Konzept der **diskreten Zufallsvariablen** anhand eines Beispiels erläutern.

Unsere zu untersuchende Population besteht aus allen Studierenden, allen Dozenten und allen Verwaltungsmitarbeitern der [FU Berlin](#). Wir wählen zufällig eine dieser Personen aus und fragen sie nach der Anzahl ihrer Geschwister. Folglich ist die Antwort, die Anzahl der Geschwister einer zufällig ausgewählten Person, eine diskrete Zufallsvariable, bezeichnet als X . Der tatsächliche Wert (Anzahl der Geschwister) von X hängt vom Zufall ab, aber wir können trotzdem alle Werte von X auflisten, z.B. 0 Geschwister, 1 Geschwister, 2 Geschwister, usw. Zur Vereinfachung beschränken wir die Anzahl der Geschwister in dieser Übung auf 5.

Laut der [Website](#) der FU Berlin gibt es im WS 2021/2022 33.000 Studierende, 4.000 Doktoranden, 379 Professoren und 4.660 Mitarbeiter an der FU Berlin (bitte beachten Sie, dass sich die

tatsächlichen Zahlen im Laufe der Zeit ändern können).

Da wir keine Vorstellung von der damit verbundenen Wahrscheinlichkeit für eine bestimmte Anzahl von Geschwistern haben, starten wir einige Experimente:

Wir wählen **eine** zufällig ausgewählte Person aus und fragen nach der Anzahl der Geschwister.

Die Antwort lautet: 0

Wir wählen **zehn** zufällig ausgewählte Personen aus und befragen sie zu ihren Geschwistern.

Die Antworten lauten: 4, 0, 2, 0, 2, 2, 1, 2, 0, 3

Wir wählen **hundert** Personen aus und fragen nach Geschwistern.

Die Antworten lauten:

2, 0, 1, 2, 2, 0, 0, 0, 1, 3, 1, 2, 1, 0, 2, 0, 0, 2, 1, 1, 1, 2, 2, 1, 2, 2, 0, 1, 1, 2, 4, 0, 3, 2, 0, 1, 2

Sie sehen, die Form der Notation wird ziemlich schnell unübersichtlich, wenn wir die Anzahl der abgefragten Individuen erhöhen. Wir beschließen also, die **Häufigkeit** und die entsprechende **relative Häufigkeit** der Werte für die Klassen 0, 1, 2, 3, 4, 5 (um es deutlich zu sagen: die letzte Klasse entspricht 5 oder mehr Geschwistern) zu notieren und das Experiment in Form einer schön formatierten Tabelle zu präsentieren.

Wir wählen 1.000 Personen aus und befragen sie zu ihren Geschwistern.

Geschwister (x)	Absolute Häufigkeit (f)	Relative Häufigkeit
0	205	0,205
1	419	0,419
2	280	0,28
3	65	0,065
4	29	0,029
5	2	0,002
	1000	1

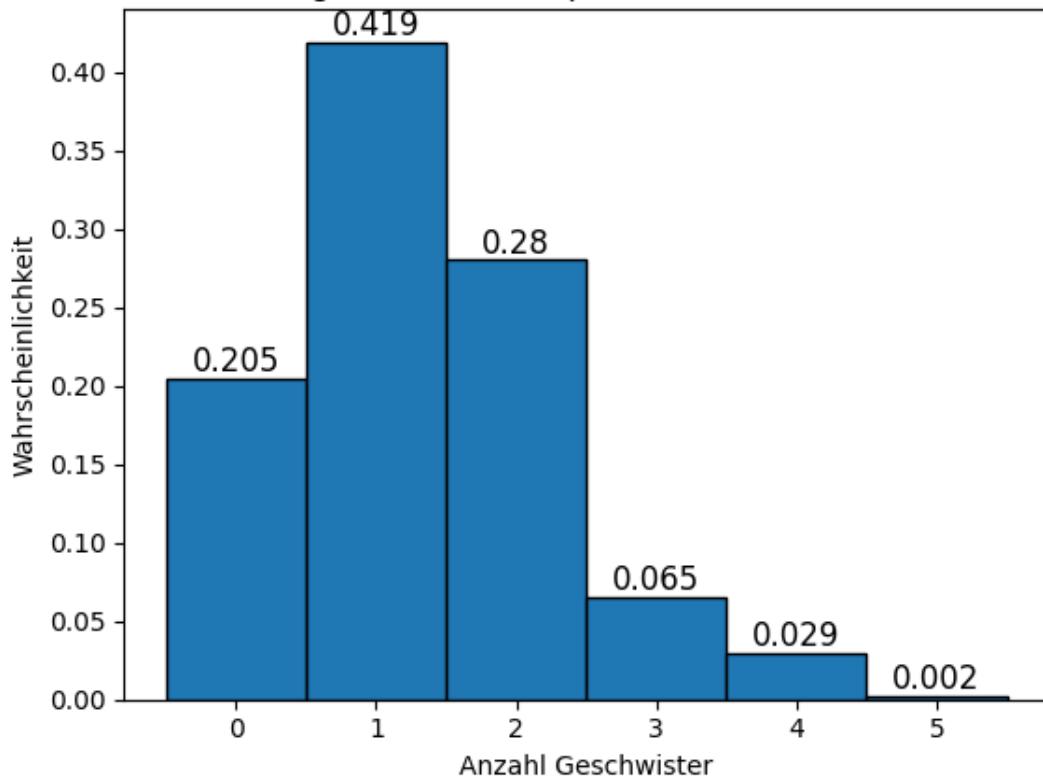
Nachdem wir alle möglichen Werte aufgelistet und die entsprechenden relativen Häufigkeiten berechnet haben, kennen wir immer noch nicht genau die Wahrscheinlichkeiten der diskreten Zufallsvariablen X für die gesamte Population von 40.961 Personen, die der FU Berlin zugeordnet sind. Nach Gesprächen mit 1.000 zufällig ausgewählten Personen sind wir jedoch recht zuversichtlich, dass eine so große Anzahl von Interviews - verglichen mit der Anzahl der Gesamtpopulation (40.961) - uns eine gute Annäherung an die Wahrscheinlichkeiten der diskreten Zufallsvariablen X (Anzahl der Geschwister) für die Gesamtpopulation liefern wird.

Im nächsten Schritt zeichnen wir ein **Wahrscheinlichkeitshistogramm** (der Stichprobe), das die möglichen Werte einer diskreten Zufallsvariablen X auf der horizontalen Achse und die Anteile dieser Werte auf der vertikalen Achse darstellt. Ein Verhältnishistogramm kann auch als Annäherung an die Wahrscheinlichkeitsverteilung dienen. Bitte beachten Sie, dass sowohl die **Summe der Wahrscheinlichkeiten** als auch die **Summe der Anteile** jeder diskreten Zufallsvariablen gleich 1 ist.

▶ Show code cell source

```
Text(0, 0.5, 'Wahrscheinlichkeit')
```

Wahrscheinlichkeitshistogramm der Zufallsvariablen X , die Anzahl der Geschwister zufällig ausgewählter Einzelpersonen der FU Berlin



Bei vielen Anwendungen im wirklichen Leben kennen wir die Wahrscheinlichkeitsverteilung der Grundgesamtheit nicht - **und werden sie auch nie kennen**. Das liegt vor allem daran, dass in vielen Anwendungen die Grundgesamtheit viel zu groß ist oder es keine Möglichkeit gibt, zuverlässige Daten zu erhalten, oder wir weder das Geld noch die Zeit für eine umfassende Datenerhebung haben. Erhöht man jedoch die Anzahl der unabhängigen Beobachtungen einer Zufallsvariablen X , so nähert sich das Wahrscheinlichkeitshistogramm der Stichprobe immer mehr dem Wahrscheinlichkeitshistogramm der Grundgesamtheit an. Um diese Behauptung zu beweisen, vergrößern wir unser Experiment:

Wir wählen nacheinander 10, 100 und 1.000 zufällig Personen aus, die mit der FU Berlin verbunden sind, und befragen sie nach der Anzahl der Geschwister. Wir werden jedes unserer drei Experimente aufzeichnen und schließlich mit der tatsächlichen/realen Wahrscheinlichkeitsverteilung vergleichen (Bitte beachten Sie, dass dieses Beispiel ein Übungsbeispiel ist und nicht die reale Anzahl der Geschwister in der Population der Personen an der FU Berlin darstellt; daher *kennen* die Dozenten des vorliegenden Skripts die Wahrscheinlichkeitsverteilung der Grundgesamtheit ;-))

```

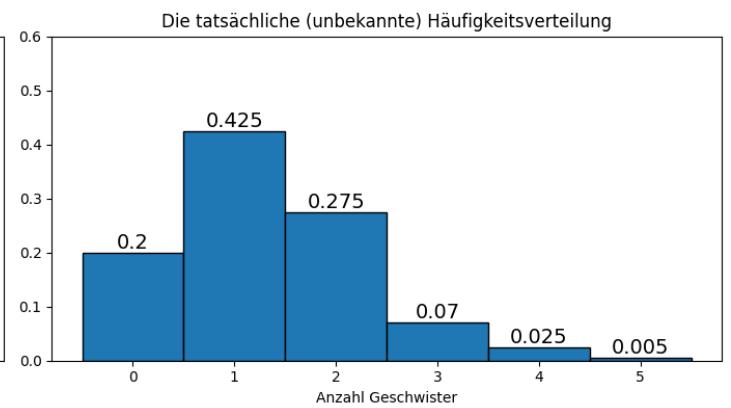
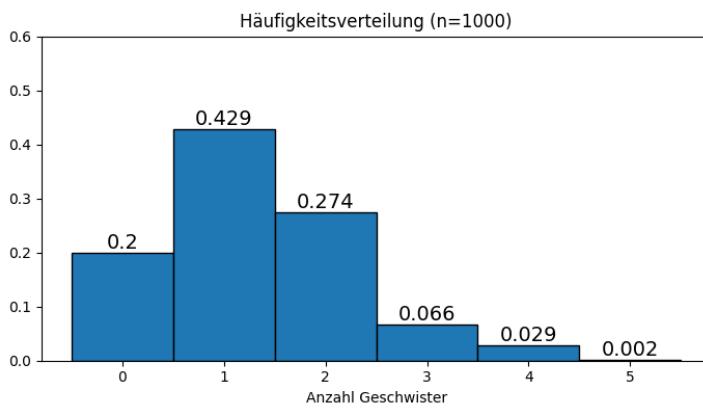
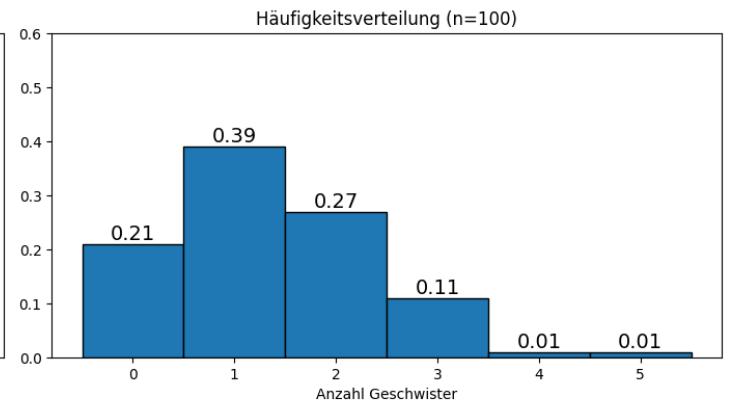
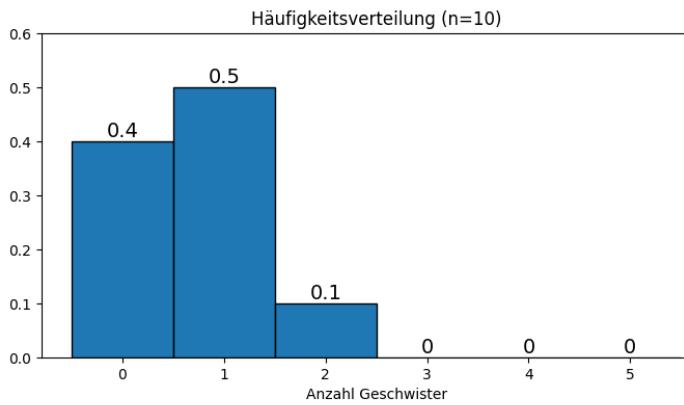
_prob_experiment = (0.2, 0.425, 0.275, 0.07, 0.025, 0.005)

trialsizes = [10, 100, 1000]
siblings = [0, 1, 2, 3, 4, 5]
np.random.seed(1)
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(14, 8))
ax = np.ravel(ax)
for e, n in enumerate(trialsizes):
    trial = np.random.choice(
        range(len(_prob_experiment)), size=n, replace=True, p=_prob_experiment
    )
    unique_elements, counts_elements = np.unique(trial, return_counts=True)
    counts = dict(zip(unique_elements, counts_elements / n))
    df = pd.DataFrame.from_dict(counts, orient="index", columns=["frequency"])
    df = df.reindex(siblings).fillna(0)
    ax[e].bar(
        df.index,
        df.frequency,
        width=1.0,
        edgecolor="k",
    )
    ax[e].bar_label(ax[e].containers[0], label_type="edge", size=14)
    ax[e].set_title(f"Häufigkeitsverteilung (n={n})")

ax[3].bar(
    siblings,
    _prob_experiment,
    width=1.0,
    edgecolor="k",
)
ax[3].bar_label(ax[3].containers[0], label_type="edge", size=14)
ax[3].set_title(f"Die tatsächliche (unbekannte) Häufigkeitsverteilung")

for _ax in ax:
    _ax.set_ylim(0, 0.6)
    _ax.set_xlabel("Anzahl Geschwister")
fig.tight_layout()

```



Die Diagramme bestätigen unsere Hypothese, dass sich das Histogramm der Stichprobe mit zunehmender Anzahl der Beobachtungen immer mehr dem Häufigkeitsverteilung der Grundgesamtheit annähert.

Der Mittelwert und die Standardabweichung einer diskreten Zufallsvariablen

Mittelwert einer diskreten Zufallsvariable

Der Mittelwert einer **diskreten Zufallsvariablen** X wird mit μ_X oder, wenn keine Verwechslung auftreten soll, einfach mit μ bezeichnet. Die Begriffe **Erwartungswert**, $E(X)$ und **Erwartung** werden üblicherweise anstelle des Begriffs Mittelwert verwendet.

$$E(X) = \sum_{i=1}^N x_i P(X = x_i)$$

Bei einer großen Anzahl unabhängiger Beobachtungen einer Zufallsvariablen X nähert sich $E(X)$ dieser Beobachtungen - der Stichprobe - dem Mittelwert μ der Grundgesamtheit an. Je größer die

Zahl der Beobachtungen ist, desto näher liegt $E(X)$ an μ (s.226).

Erinnern wir uns an unser Experiment aus dem vorherigen Abschnitt, als wir 1.000 Personen ausgewählt und nach der Anzahl der Geschwister gefragt haben. Werfen wir noch einmal einen Blick auf die Tabelle, die das Experiment zusammenfasst

Geschwister (x)	Absolute Häufigkeit (f)	Relative Häufigkeit
0	205	0,205
1	419	0,419
2	280	0,28
3	65	0,065
4	29	0,029
5	2	0,002
	1000	1

Berechnen wir den Erwartungswert (Mittelwert) für dieses Experiment.

$$\begin{aligned}
 E(X) &= \sum_{i=1}^N x_i P(X = x_i) \\
 &= 0 \cdot P(X = 0) + 1 \cdot P(X = 1) + 2 \cdot P(X = 2) + 3 \cdot P(X = 3) + 4 \cdot P(X = 4) + 5 \cdot \\
 &= 0 \cdot 0,205 + 1 \cdot 0,419 + 2 \cdot 0,28 + 3 \cdot 0,065 + 4 \cdot 0,029 + 5 \cdot 0,002 \\
 &= 1,3
 \end{aligned}$$

Der sich daraus ergebende Erwartungswert von 1,3 liegt nahe am Mittelwert μ , den wir anhand der Wahrscheinlichkeiten der Grundgesamtheit berechnen (die realen Wahrscheinlichkeiten sind der unteren rechten Abbildung im vorherigen Abschnitt entnommen).

$$\mu = 1 \cdot 0,2 + 2 \cdot 0,425 + 3 \cdot 0,275 + 4 \cdot 0,07 + 5 \cdot 0,025 = 2,28$$

Übung

Betrachten wir einen fairen sechsseitigen Würfel. Wir können den **Erwartungswert** $E(X)$ leicht mit Python berechnen. Der Begriff “fair” bedeutet, dass jede Zufallsvariable

$X = x_i$, $x \in \{1, 2, 3, 4, 5, 6\}$ mit gleicher Wahrscheinlichkeit auftritt. Daher ist $P(X = x_i) = \frac{1}{6}$.

$$E(X) = \sum_{i=1}^6 x_i P(X = x_i) = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3,5$$

In Python schreiben wir den folgenden Code:

```
p_die = 1 / 6
die = pd.Series([1, 2, 3, 4, 5, 6])
die = die * p_die
sum(die)
```

3.5

Was aber, wenn wir uns nicht sicher sind, ob die Würfel wirklich fair sind? Woher wissen wir, dass wir nicht betrogen werden? Oder anders ausgedrückt: Wie oft müssen wir würfeln, bevor wir mehr Vertrauen haben können?

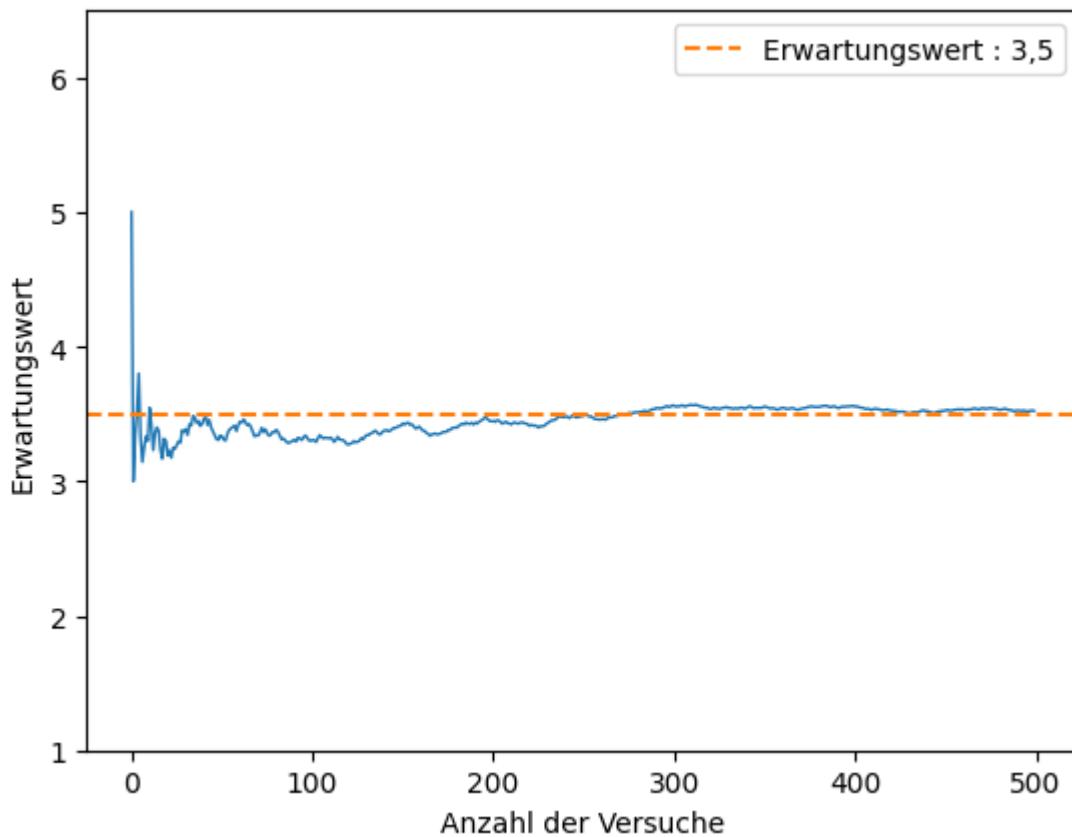
Führen wir ein Berechnungsexperiment durch: Wir wissen aus den obigen Überlegungen, dass der Erwartungswert eines 6-seitigen fairen Würfels 3,5 ist. Wir führen ein Experiment durch, indem wir einen Würfel immer und immer wieder werfen. Wir speichern das Ergebnis und bevor wir erneut würfeln, berechnen wir den Durchschnitt aller bisherigen Würfelwürfe. Um dieses kleine Experiment durchzuführen, schreiben wir eine for-Schleife in Python.

```

# Setze random seed für Reproduzierbarkeit
random.seed(10)
wuerfel = []
e_wert = []
x = np.arange(0, 500, 1)
# Simuliere Würfelwurf
for i in range(500):
    r = random.randint(1, 6)
    wuerfel.append(r)
    e_wert.append(np.mean(wuerfel))
# Plotten
fig, ax = plt.subplots()
ax.plot(x, e_wert, lw=1)
ax.axhline(y=3.5, color="C1", linestyle="dashed", label="Erwartungswert : 3,5")
ax.set_xlabel("Anzahl der Versuche")
ax.set_ylabel("Erwartungswert")
ax.set_ylim(1, 6.5)
ax.legend()

```

<matplotlib.legend.Legend at 0x725e00a9f5e0>



Das Diagramm zeigt, dass die Kurve nach anfänglichen Schwankungen schließlich abflacht und sich dem $E(X)$ von 3,5 annähert.

Standardabweichung einer diskreten Zufallsvariable

Die Standardabweichung einer diskreten Zufallsvariablen X wird mit σ_X oder, wenn keine Verwechslung auftreten soll, einfach mit σ bezeichnet. Sie ist definiert als

$$\sigma = \sqrt{\sum_{i=1}^N (x_i - \mu)^2 P(X = x_i)}$$

Die Binomialverteilung

```
import math
import random
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import binom
```

Viele Anwendungen der Wahrscheinlichkeitsrechnung und Statistik betreffen die Wiederholung eines Experiments. Wir nennen jede Wiederholung einen **Versuch**, und wir sind besonders an Fällen interessiert, in denen das Experiment (jeder Versuch) **nur zwei mögliche Ergebnisse** hat: Erfolg oder Misserfolg, wahr oder falsch, 0 oder 1.

Um wiederholte Versuche eines Experiments mit zwei möglichen Ergebnissen zu analysieren, benötigen wir Kenntnisse über **Fakultäten**, **Binomialkoeffizienten** und **Bernoulli-Versuche**.

Auf der Grundlage dieser Voraussetzungen lernen wir die **Binomialverteilung** kennen, die eine **Wahrscheinlichkeitsverteilung** für die Anzahl der Erfolge in einer Folge von **Bernoulli-Versuchen** ist ([] s.213,s.289).

Voraussetzungen

Fakultät

Das Produkt der ersten k positiven ganzen Zahlen wird als k -Fakultät bezeichnet und mit $k!$ abgekürzt (siehe s.186).

$$k! = k \times (k - 1) \times \dots \times 2 \times 1$$

Wir definieren auch $0! = 1$.

Betrachten wir ein einfaches Beispiel und berechnen die Fakultät von 6. Wenn wir die obige Gleichung einsetzen, erhalten wir folgendes Ergebnis

$$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$$

In Python können wir das entweder direkt berechnen...

```
6 * 5 * 4 * 3 * 2 * 1
```

```
720
```

...oder wir verwenden die eingebaute Funktion `factorial()`

```
math.factorial(6)
```

```
720
```

Binomialkoeffizienten

Wenn n eine positive ganze Zahl und k eine nicht negative ganze Zahl kleiner oder gleich n ist, dann ist der Binomialkoeffizient $\binom{n}{k}$ definiert als:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Die Binomialkoeffizienten $\binom{n}{k}$ werden oft als “ n über k ” gelesen, weil es $\binom{n}{k}$ Möglichkeiten gibt, k Elemente ohne Rücksicht auf ihre Reihenfolge aus einer Menge von n Elementen auszuwählen. Mit anderen Worten: Die Binomialkoeffizienten beziehen sich auf die Anzahl der Kombinationen von n Dingen, die k -mal ohne Wiederholung ausgewählt werden. Bitte beachten Sie, dass die Reihenfolge der Auswahl keine Rolle spielt.

Versuchen wir es mit einem Beispiel, um ein Gefühl dafür zu bekommen.

Nehmen wir ein einfaches Wort wie “dog”, das drei verschiedene Buchstaben enthält: d, o, g. Wie viele Möglichkeiten gibt es, genau einen Buchstaben aus diesen 3 Buchstaben zu ziehen? Mit Sicherheit gibt es 3 Möglichkeiten, genau einen Buchstaben zu zeichnen: “d”, “o”, oder “g”. Wir können also alle Kombinationen als $\binom{3}{1}$ schreiben. Was ist mit zwei Buchstaben? Wie viele Kombinationen gibt es, um genau zwei Buchstaben aus 3 Buchstaben zu ziehen? Die Kombinationen sind “do”, “dg” und “og” (Bitte beachten Sie, dass z.B. “og” und “go” nur als eine Kombination gezählt werden, da die Reihenfolge im Moment keine Rolle spielt). Dementsprechend ist die Antwort 3, die als $\binom{3}{2}$ geschrieben werden kann. Eine letzte Frage: Wie viele Kombinationen gibt es, um aus 3 Buchstaben genau drei Buchstaben zu ziehen? Fragen wir Python!

Zunächst wenden wir einen naiven Ansatz an, indem wir die obige Formel anwenden.

```
# n entspricht 3 Buchstaben von dog
n = 3
# k entspricht wir ziehen 3 Buchstaben
k = 3
math.factorial(n) / ((math.factorial(k) * math.factorial(n - k)))
```

Wir können aber auch die eingebaute Funktion `math.comb()` verwenden, um die Anzahl der Kombinationen zu berechnen. Wir machen das für $k = 1, 2, 3$.

```
# 3 über 1  
math.comb(3, 1)
```

3

```
# 3 über 2  
math.comb(3, 2)
```

3

```
# 3 über 3  
math.comb(3, 3)
```

1

Nun, da wir mit dem Konzept vertraut sind, wollen wir ein komplexeres Beispiel betrachten: Die SRH Hochschule bittet alle Absolventen, ihre 4 Lieblingskurse aus dem Studienplan zu wählen. Wie viele verschiedene Antworten könnten die Studierenden geben, wenn der Studienplan 24 Kurse zur Auswahl anbietet? Irgendwelche Vermutungen? Lassen Sie uns Python fragen!

```
math.comb(24, 4)
```

10626

Die Studenten können 10.626 verschiedene Antworten geben.

Bernoulli-Versuche

Wiederholte Versuche eines Experiments werden als **Bernoulli-Versuche** bezeichnet, wenn die folgenden drei Bedingungen erfüllt sind:

1. Das Experiment (jeder Versuch) hat zwei mögliche Ergebnisse, die mit s für Erfolg und f für Misserfolg bezeichnet werden.
2. Die Versuche sind unabhängig voneinander.
3. Die Wahrscheinlichkeit für einen Erfolg, die als Erfolgswahrscheinlichkeit bezeichnet wird und mit p bezeichnet wird, bleibt von Versuch zu Versuch gleich.

Die Binomialverteilung

Die Binomialverteilung ist die **Wahrscheinlichkeitsverteilung** für die Anzahl der Erfolge in einer Folge von **Bernoulli-Versuchen** ([]) s.350).

Die binomische Wahrscheinlichkeitsformel

Bei n Bernoulli-Versuchen ist die Anzahl der Ergebnisse, die genau x Erfolge enthalten, gleich dem Binomialkoeffizienten $\binom{n}{x}$ ([]) s.188).

X sei die Gesamtzahl der Erfolge in n Bernoulli-Versuchen mit der Erfolgswahrscheinlichkeit p . Dann ist die Wahrscheinlichkeitsverteilung der Zufallsvariablen X gegeben durch:

$$P(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}, \quad x = 0, 1, 2, \dots, n$$

Die Zufallsvariable X wird als **Binomialzufallsvariable** bezeichnet und folgt der Binomialverteilung mit den Parametern n und p ([]) s.356).

Betrachten wir ein Beispiel aus der realen Welt.

Langzeitstatistiken besagen, dass die Chance, die Abschlussprüfung in Statistik zu bestehen, bei 0,3 liegt. Ja, etwa 70% der Studenten fallen in der Statistikprüfung durch! Übrigens, wenn Sie dieses E-Learning-Modul absolvieren, werden Ihre Chancen, die Abschlussprüfung zu bestehen, definitiv steigen :-)

Betrachten wir eine Klasse mit 25 Studenten. Wie hoch ist die Wahrscheinlichkeit, dass genau 3 der Studenten dieser Klasse die Abschlussprüfung in Statistik bestehen werden? Oder anders ausgedrückt: $P(X = 3)$. Auch hier beginnen wir mit der Implementierung eines naiven Ansatzes in Python.

```

# Anzahl der Studenten
n = 25
# Erfolgswahrscheinlichkeit
p = 0.3
# Anzahl der Studenten die die Prüfung bestehen
k = 3

math.comb(n, k) * p ** (k) * (1 - p) ** (n - k)

```

0.02427998871170032

Wow, die Wahrscheinlichkeit, dass genau 3 von 25 Studierenden ($P(X = 3)$) die Abschlussprüfung in Statistik bestehen, ist ziemlich gering. Wie sieht es mit der Wahrscheinlichkeit aus, dass genau 15 von 25 Studierenden ($P(X = 15)$) die Abschlussprüfung in Statistik bestehen werden? Wir wenden uns an Python.

```

# Anzahl der Studenten
n = 25
# Erfolgswahrscheinlichkeit
p = 0.3
# Anzahl der Studenten die die Prüfung bestehen
k = 15

math.comb(n, k) * p ** (k) * (1 - p) ** (n - k)

```

0.001324897424235193

Die Wahrscheinlichkeit von $P(X = 15)$ beträgt etwa 0,1%. Wir könnten mit unseren Experimenten fortfahren, um alle Wahrscheinlichkeiten für **genau ein Ergebnis** für $k = 0, 1, 2, \dots, n$ herauszufinden. Bitte beachten Sie allerdings, dass es für unser spezielles Beispiel nicht sehr informativ ist, die Wahrscheinlichkeit für das Bestehen der Prüfung für genau eine bestimmte Anzahl (k) von Studenten zu kennen. Für uns ist es von größerem Informationsinteresse, wenn wir die Frage beantworten könnten, wie hoch die Wahrscheinlichkeit ist, dass k oder weniger Studierende ($P(X \leq k)$) die Prüfung bestehen, oder, was ebenso interessant ist, dass k oder mehr Studierende ($P(X \geq k)$) die Prüfung bestehen.

Als Übung wenden wir uns an Python und bestimmen die Wahrscheinlichkeit, dass 9 oder weniger Studierende die Abschlussprüfung in Statistik bestehen ($P(X \leq 9)$). Wir interessieren uns also für die Wahrscheinlichkeit, dass 0 von 25, 1 von 25, 2 von 25, ... oder 9 von 25 Studierenden die

Prüfung bestehen. Um diese Wahrscheinlichkeit zu berechnen, können wir unseren naiven Ansatz erweitern und Folgendes berechnen

$$P(X = 0) + P(X = 1) + P(X = 2) + \cdots + P(X = 9)$$

```
# Anzahl der Studenten
n = 25
# Erfolgswahrscheinlichkeit
p = 0.3
# Anzahl der maximalen Erfolge
k = 9

prob = 0
prob_total = 0

# Erzeuge for-loop
for i in range(k + 1):
    prob = math.comb(n, i) * p ** (i) * (1 - p) ** (n - i)
    prob_total += prob
prob_total
```

0.8105639764950532

Die Wahrscheinlichkeit, dass 9 oder weniger Studierende die Statistikprüfung bestehen ($P(X \leq 9)$), beträgt 81,1%. Das wiederum bedeutet, dass die Wahrscheinlichkeit, dass 10 oder mehr Studierende die Prüfung bestehen, $P(X > 9) = 1 - P(X \leq 9)$ oder nur 18,9% beträgt.

Neben der ehrlich gesagt recht unangenehmen Einsicht in die Statistik des Bestehens der Abschlussprüfung ist es klar, dass die naive Implementierung von oben ziemlich mühsam ist. Deshalb führen wir die eingebauten Funktionen `binom.pmf()` und `binom.cdf()` ein.

Die Verwendung von `binom.pmf()` ist wie folgt: `binom.pmf(k, n, p, loc)`. Wir lassen vorerst das Argument `loc` außer Acht. Somit vereinfacht sich die Funktion `binom.pmf()` zu `binom.pmf(k, n, p)`, wobei k den insgesamten Erfolgen, n der Gesamtanzahl und p der Wahrscheinlichkeit für das Bestehen der Prüfung entspricht. Um zu überprüfen, ob die Funktion `binom.pmf()` wie erwartet funktioniert, erinnern wir uns an unsere Testbeispiele von oben, nämlich $P(X = 3)$ und $P(X = 15)$.

```
binom.pmf(3, 25, 0.3)
```

```
np.float64(0.024279988711700378)
```

```
binom.pmf(15, 25, 0.3)
```

```
np.float64(0.0013248974242351943)
```

Vergleichen Sie diese Ergebnisse mit den Ergebnissen unserer naiven Implementierung (scrollen Sie nach oben). Die Zahlen sollten übereinstimmen.

Die Funktion `binom.pmf()` ist sehr praktisch, wenn wir Wahrscheinlichkeiten akkumulieren wollen, wie z. B. $P(X \leq x)$ oder $P(X > x)$, denn wir können eine for - Schleife für das Argument x angeben und anschließend summieren. Betrachten wir das Beispiel von oben ($P(X \leq 9)$).

```
P_le9 = []
for i in range(10):
    P_le9.append(binom.pmf(i, 25, 0.3))
P_le = sum(P_le9)
P_le
```

```
np.float64(0.8105639764950539)
```

Eine weitere eingebaute Funktion, die für die Beantwortung von $P(X \leq 9)$ geeignet ist, ist die Funktion `binom.pdf()`, die die Verteilungsfunktion zurückgibt. Die Funktion `binom.cdf()` berechnet die kumulative Wahrscheinlichkeitsverteilung und ist insofern praktisch, da wir auf den Aufruf der Summenfunktion verzichten können. Die Verwendung von `binom.cdf()` ist wie folgt:

`binom.cdf(k, n, p, loc)`. Auch hier lassen wir das Argument `loc` außer Acht und belassen es bei `binom.cdf(k, n, p)`. Zusätzlich zu den Argumenten der Funktion `binom.cdf()` gilt zu beachten, dass `binom.cdf()` die Wahrscheinlichkeiten $P(X \leq x)$ wiedergibt und `binom.sf(k, n, p)` $P(X > x)$ die Gegenwahrscheinlichkeit `1-binom.cdf()`.

Um dies zu verdeutlichen, lassen Sie uns $P(X \leq 9)$ neu berechnen. Das Ergebnis der obigen Berechnung war 0,81056. Nun wenden wir die Funktion `binom.cdf()` an, um das gleiche Ergebnis zu erhalten.

```
binom.cdf(9, 25, 0.3)
```

```
np.float64(0.8105639764950546)
```

Um $P(X > 9)$ zu erhalten, was dasselbe ist wie $1 - P(X \leq 9)$, setzen wir das Argument

```
binom.sf(9, 25, 0.3)
```

```
np.float64(0.18943602350494537)
```

```
1 - binom.cdf(9, 25, 0.3)
```

```
np.float64(0.18943602350494537)
```

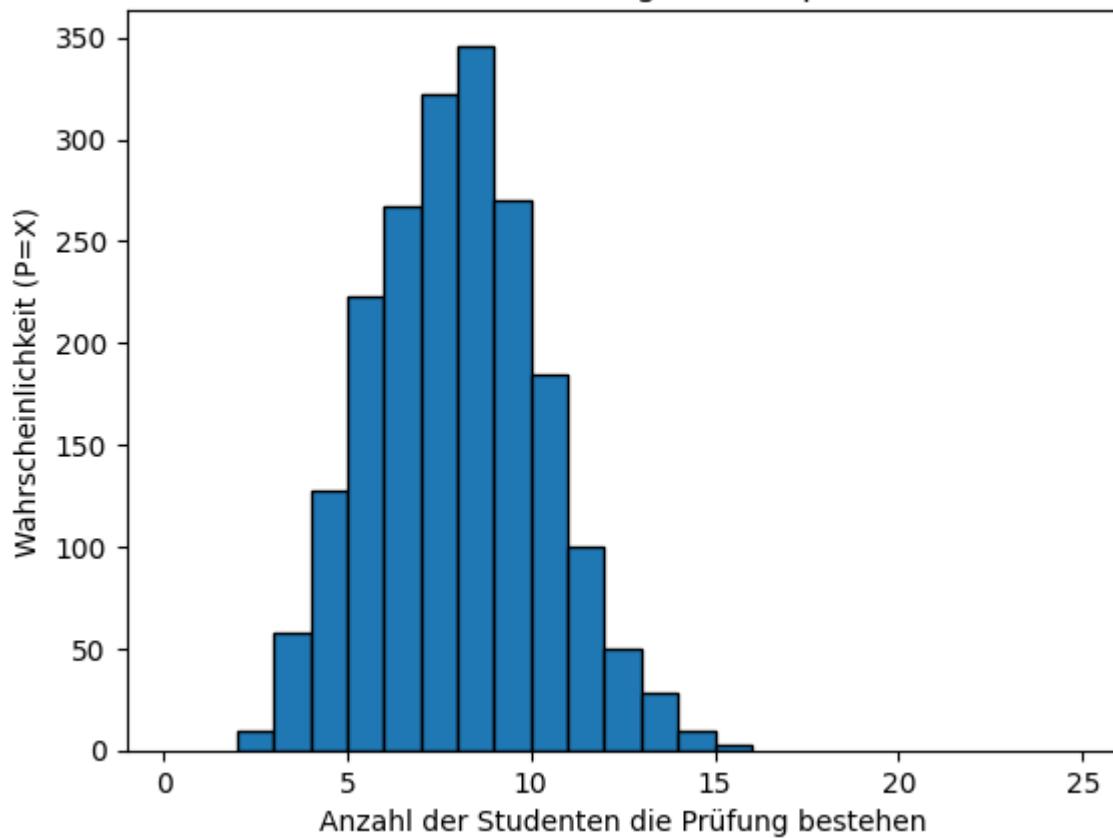
Um die **binomische Wahrscheinlichkeitsverteilung** zu visualisieren, verwenden wir die Funktion `binom.rvs()`, die Zufallsabweichungen für eine Binomialverteilung erzeugt, die durch ihre Größe n und die Erfolgs-/Misserfolgswahrscheinlichkeit p definiert ist.

```
# Setze random seed für Reproduzierbarkeit
np.random.seed(seed=42)

# Erzeuge y-Werte der Binomialverteilung
n = 25
p = 0.3
y = binom.rvs(n, p, size=2000)

# Erzeuge Plot
fig, ax = plt.subplots()
ax.set_xlim(-1, 26)
ax.set_title(f"Binomialverteilung: n={n} , p={p}")
ax.set_xlabel("Anzahl der Studenten die Prüfung bestehen")
ax.set_ylabel("Wahrscheinlichkeit (P=X)")
bins = max(y) - min(y)
_ = ax.hist(y, bins, edgecolor="k")
```

Binomialverteilung: $n=25$, $p=0.3$

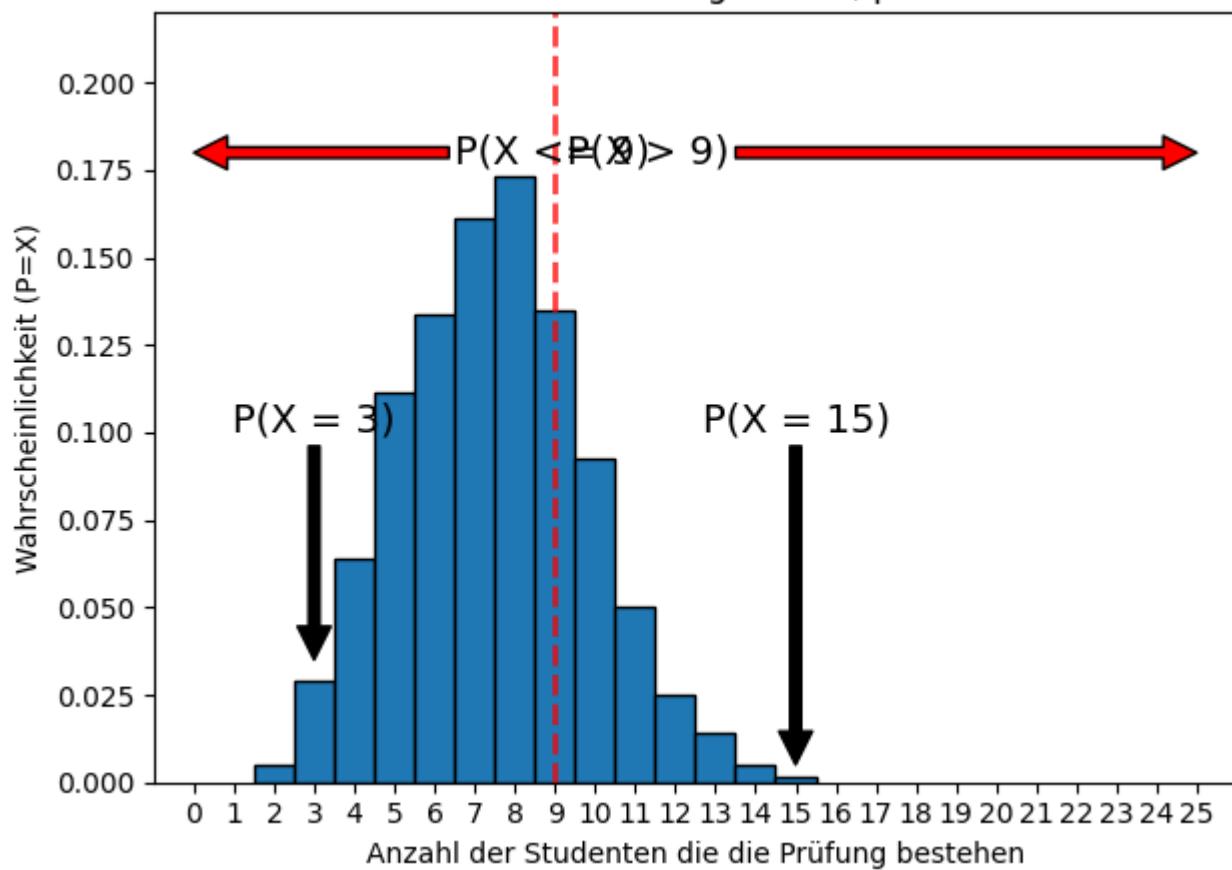


Zusätzlich visualisieren wir die oben diskutierten Wahrscheinlichkeiten:

$P(X = 3)$, $P(X = 15)$, $P(X \leq 9)$, $P(X > 9)$ und die entsprechende **kumulative binomiale Wahrscheinlichkeitsverteilungsfunktion**.

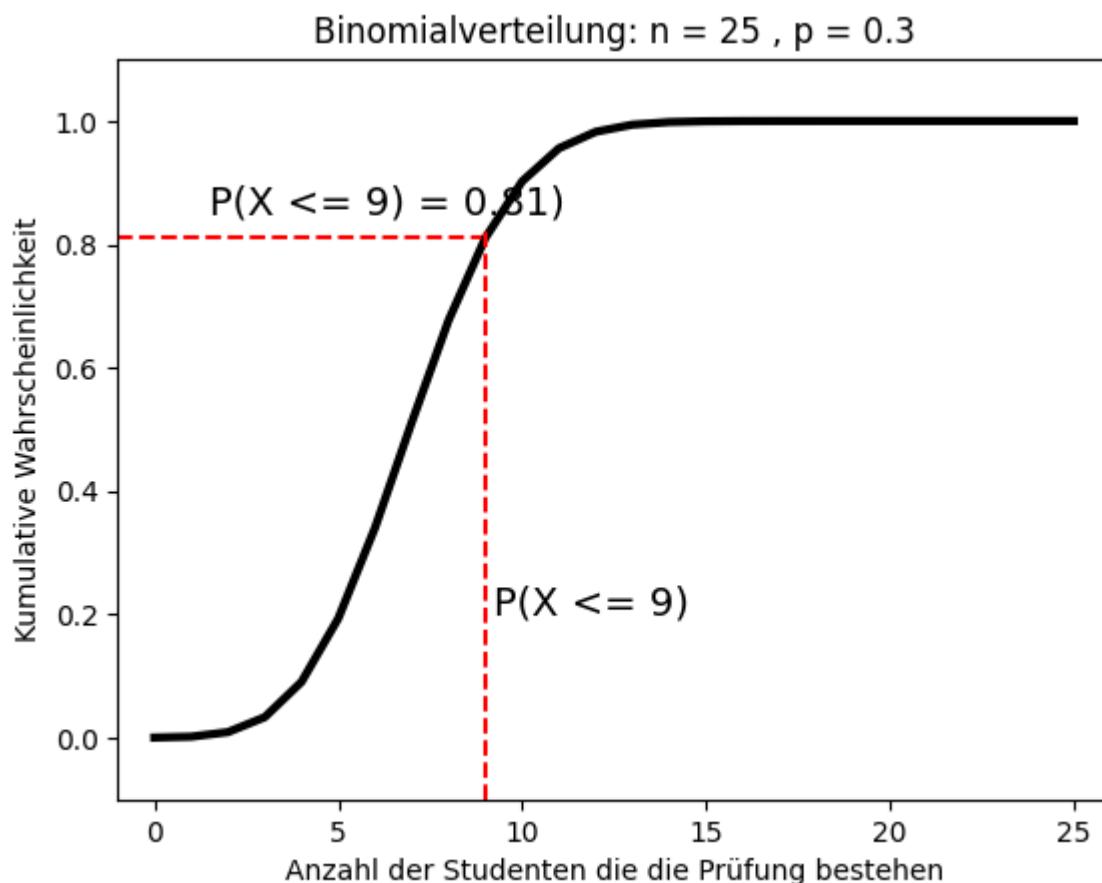
► Show code cell source

Binomialverteilung: $n=25$, $p=0.3$



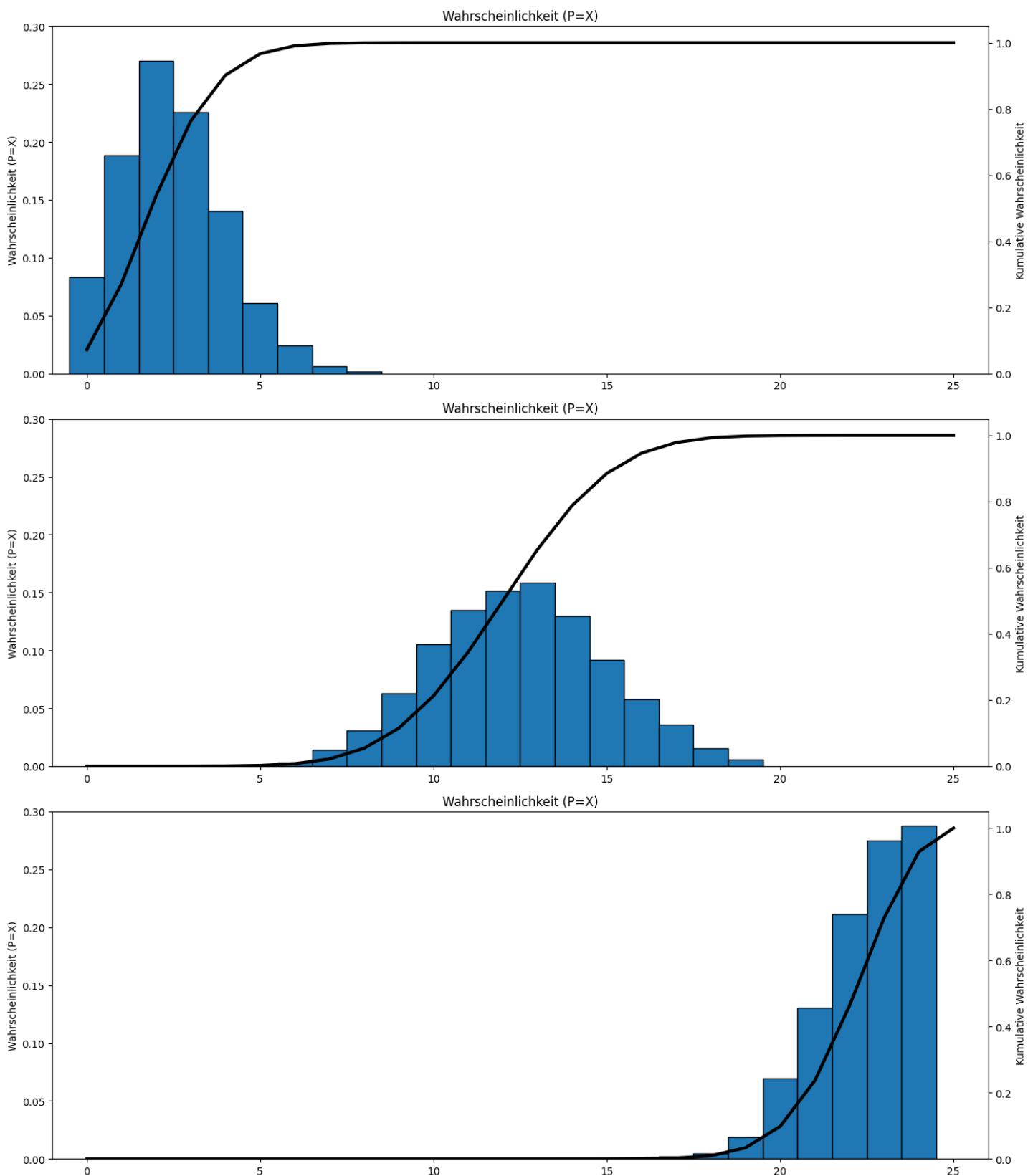
► Show code cell source

Text(1.5, 0.85, 'P(X <= 9) = 0.81')



Zum Abschluss dieses Abschnitts und um Ihnen ein Gefühl für die Formen der verschiedenen binomischen Wahrscheinlichkeitsverteilungen zu vermitteln, werden im Folgenden drei verschiedene binomische Wahrscheinlichkeitsverteilungen und die entsprechenden kumulativen binomischen Beweisbarkeitsverteilungen für $p = 0,1, p = 0,5$ und $p = 0,9$ angegeben.

```
fig, ax = plt.subplots(nrows=3, figsize=(14, 16))
x = np.linspace(0, 25, num=26)
for e, p in enumerate([0.1, 0.5, 0.9]):
    y = binom.rvs(n, p, size=2000)
    bins = max(y) - min(y)
    ax[e].hist(y, bins, edgecolor="k", density=True, align="left")
    ax[e].set_ylabel("Wahrscheinlichkeit (P=X)")
    ax[e].set_title("Wahrscheinlichkeit (P=X)")
    ax[e].set_xlim(-1, 26)
    ax[e].set_ylim(0, 0.30)
    # zweite Y-Achse
    ax2 = ax[e].twinx()
    cdf = binom.cdf(x, n, p)
    ax2.plot(x, cdf, color="k", linewidth=3)
    ax2.set_ylabel("Kumulative Wahrscheinlichkeit")
    ax2.set_ylim(0, 1.05)
fig.tight_layout()
```



Mittelwert und Standardabweichung einer binomialen Zufallsvariable

Der Mittelwert und die Standardabweichung einer binomialen Zufallsvariablen mit den Parametern n und p sind jeweils

$$\mu = np$$

und

$$\sigma = np(1 - p),$$

Erinnern wir uns an das Beispiel aus dem vorherigen Abschnitt. Die Wahrscheinlichkeit, die statistische Abschlussprüfung zu bestehen, beträgt 0,3. Wir betrachten eine Klasse mit 25 Studenten. Die Zufallsvariable X , die dem Erfolg in der Prüfung entspricht, ist eine binomiale Zufallsvariable und folgt einer Binomialverteilung mit den Parametern $n = 25$ und $p = 0,3$. Somit können der Mittelwert μ und die Standardabweichung σ wie folgt berechnet werden:

$$\mu = np = 25 \times 0,3 = 7,5$$

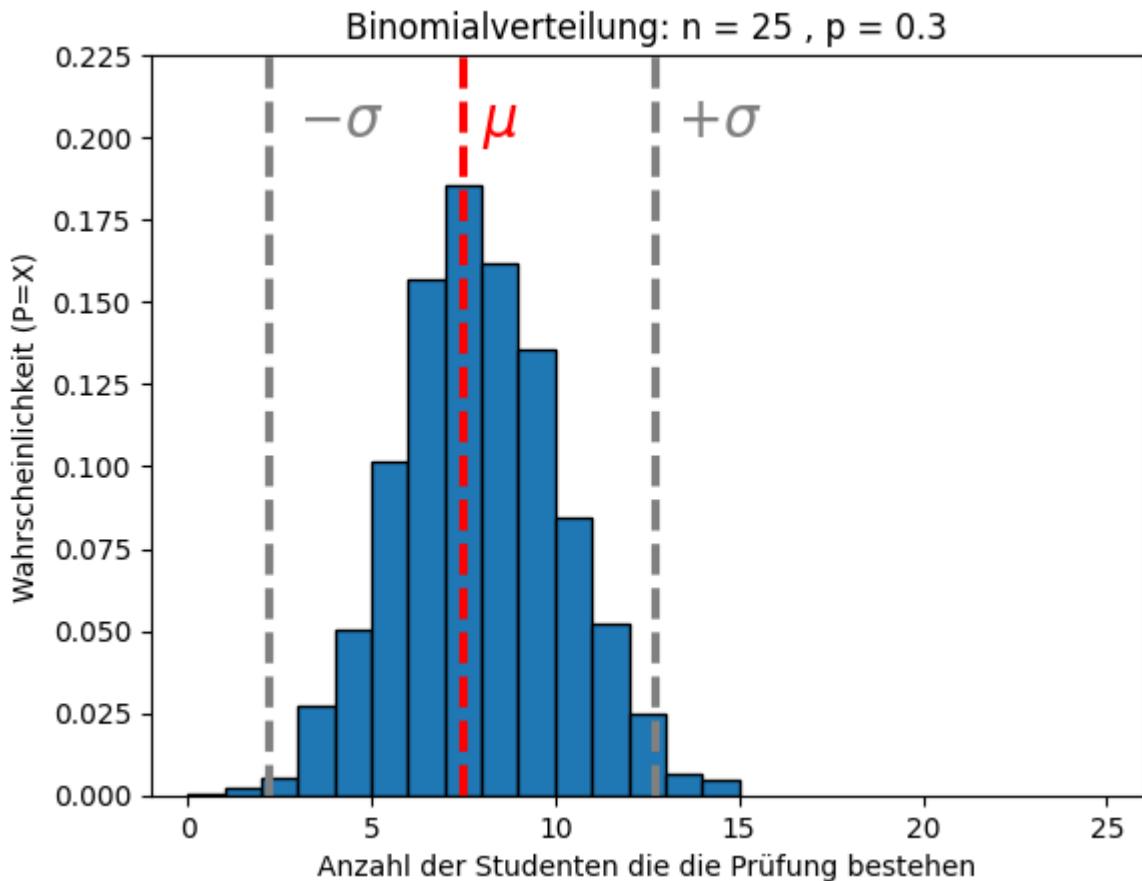
und

$$\sigma = np(1 - p) = 25 \times 0,3 \times (1 - 0,3) = 5,25$$

Die folgende Grafik zeigt die Binomialverteilung mit den Parametern $n = 25$ und $p = 0,3$ sowie deren Mittelwert μ und Standardabweichung σ .

► Show code cell source

Text(13.25, 0.2, '\$+\sigma\$')



Binomiale Approximation der hypergeometrische Verteilung

Häufig möchte man den Anteil (Prozentsatz) der Mitglieder einer endlichen Grundgesamtheit bestimmen, die ein bestimmtes Attribut aufweisen. Im Allgemeinen ist die betrachtete Grundgesamtheit zu groß, als dass der Grundgesamtheitsanteil durch eine Zählung ermittelt werden könnte. Nehmen wir an, dass eine einfache Zufallsstichprobe des Umfangs n aus einer Grundgesamtheit entnommen wird, in der der Anteil der Mitglieder, die ein bestimmtes Merkmal aufweisen, p ist. Dann ist eine Zufallsvariable von primärer Bedeutung für die Schätzung von p die Anzahl der Mitglieder der Stichprobe, die das bestimmte Merkmal aufweisen, die wir als X bezeichnen. Die genaue Wahrscheinlichkeitsverteilung von X hängt davon ab, ob die Stichprobe **mit** oder **ohne Ersetzung** durchgeführt wird. Wird die Stichprobe mit Ersetzung durchgeführt, handelt es sich um Bernoulli-Versuche: Jede Auswahl eines Mitglieds der Grundgesamtheit entspricht einem Versuch. Ein Versuch ist erfolgreich, wenn das in diesem Versuch ausgewählte Mitglied das

angegebene Attribut aufweist; andernfalls ist er nicht erfolgreich. Die Versuche sind **unabhängig**, da die Stichprobe mit Ersetzung durchgeführt wird. Die Erfolgswahrscheinlichkeit bleibt von Versuch zu Versuch gleich, sie ist immer gleich dem Anteil der Grundgesamtheit, der das angegebene Merkmal aufweist. Daher hat die Zufallsvariable X die **Binomialverteilung** mit den Parametern n (der Stichprobenumfang) und p (der Anteil der Grundgesamtheit) ([] s.356).

In der Realität werden die Stichproben jedoch in der Regel **ohne Ersetzung** durchgeführt. Unter diesen Umständen handelt es sich bei dem Stichprobenverfahren nicht um Bernoulli-Versuche, da die Versuche **nicht unabhängig** sind und die Erfolgswahrscheinlichkeit von Versuch zu Versuch variiert. Mit anderen Worten: Die Zufallsvariable X hat keine Binomialverteilung. Ihre Verteilung wird als hypergeometrische Verteilung bezeichnet ([] s.364).

In der Praxis kann eine hypergeometrische Verteilung jedoch in der Regel durch eine Binomialverteilung angenähert werden. Der Grund dafür ist, dass es bei einem **Stichprobenumfang von nicht mehr als 5 % der Grundgesamtheit** kaum einen Unterschied zwischen einer Stichprobe mit und ohne Ersatz gibt ([] s.365).

Stichproben und die Binomialverteilung

Angenommen, eine einfache Zufallsstichprobe des Umfangs n wird aus einer endlichen Grundgesamtheit entnommen, in der der Anteil der Mitglieder, die ein bestimmtes Merkmal aufweisen, p ist. Dann hat die Anzahl der in der Stichprobe enthaltenen Mitglieder, die das bestimmte Merkmal aufweisen

- eine exakte Binomialverteilung mit den Parametern n und p , wenn die Stichprobe mit Ersatz durchgeführt wird
- eine näherungsweise Binomialverteilung mit den Parametern n und p wenn die Stichprobenziehung ohne Ersetzung erfolgt und der Stichprobenumfang nicht mehr als 5% des Grundgesamtheitsumfangs beträgt ([] s.365).

Die Poisson-Verteilung

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import poisson
```

Eine weitere wichtige diskrete Wahrscheinlichkeitsverteilung ist die [Poisson-Verteilung](#), benannt zu Ehren des französischen Mathematikers und Physikers [Simeon D. Poisson](#) (1781-1840). Die Poisson-Verteilung wird häufig verwendet, um die Wahrscheinlichkeit zu beschreiben, dass eine Reihe von Ereignissen in einem bestimmten Zeit- oder Raumintervall eintritt, wobei die Wahrscheinlichkeit des Auftretens dieser Ereignisse sehr gering ist (s.367). Da die Anzahl der Versuche jedoch sehr groß ist, treten diese Ereignisse tatsächlich ein.

Die Zufallsvariable X , die als **Poisson-Zufallsvariable** bezeichnet wird, ist die Anzahl der Ereignisse (oder des Eintreffens) solcher Ereignisse in einem bestimmten Zeit- oder Raumintervall. Eine Poisson-Zufallsvariable hat unendlich viele mögliche Werte, nämlich alle ganzen Zahlen (s.242).

Unter der Annahme, dass λ der Erwartungswert solcher Ankünfte in einem Zeitintervall fester Länge ist, ist die Wahrscheinlichkeit, genau x Ereignisse zu beobachten, durch die Wahrscheinlichkeitsfunktion gegeben

$$P(X = x) = e^{-\lambda} \frac{\lambda^x}{x!}, \quad x = 0, 1, 2, \dots,$$

wobei λ eine positive reelle Zahl ist, die die durchschnittliche Anzahl der Ereignisse während eines festen Zeitintervalls darstellt, und $e \approx 2,7182818$ (die Eulersche Zahl). Somit wird jede bestimmte Poisson-Verteilung durch einen Parameter identifiziert, der gewöhnlich mit λ (dem griechischen Buchstaben Lambda) bezeichnet wird. Wenn das Ereignis beispielsweise durchschnittlich 10 Mal pro Sekunde auftritt, tritt es in 60 Sekunden durchschnittlich 600 Mal auf und $\lambda = 600$.

Die Poisson-Verteilung - ein Beispiel

Wenden wir die Poisson-Verteilung in Form eines Beispiels an. Wir konzentrieren uns auf das [Jahrhunderthochwasser](#), ein Konzept, das im Flussbau häufig zur Planung von Hochwasserschutzmaßnahmen verwendet wird.

Erinnern wir uns an die mathematische Notation einer **Poisson-Zufallsvariablen**:

$$P(X = x) = e^{-\lambda} \frac{\lambda^x}{x!}, \quad x = 0, 1, 2, \dots,$$

wobei λ eine positive reelle Zahl ist, die die durchschnittliche Anzahl der Ereignisse während eines festen Zeitintervalls darstellt, und $e \approx 2,7182818$.

Das *Jahrhunderthochwasser* ist eine Kurzbezeichnung für ein Hochwasser mit einer jährlichen Überschreitungswahrscheinlichkeit von 1% und einem durchschnittlichen Wiederholungsintervall von 100 Jahren. Der Begriff kann jedoch für Menschen irreführend sein, denn sie stellen sich vor, dass der Begriff Hochwasser beschreibt, die einmal alle 100 Jahre auftreten. Dies ist jedoch nicht der Fall. Ein Hochwasser mit einer jährlichen Überschreitungswahrscheinlichkeit von 1% bedeutet, dass in **jedem** einzelnen Jahr mit einer Wahrscheinlichkeit von 0,01 ein Hochwasser in einer Größenordnung auftritt, die einem Jahrhunderthochwasser entspricht.

Im Rahmen einer Poisson-Verteilung wird der Erwartungswert $E(x) = \lambda$ eines solchen Hochwassers während des festen Intervalls von 100 Jahren auf $\lambda = 100 \times 0,01 = 1$ gesetzt. Die Poisson-Zufallsvariable X ist also die Anzahl der Ereignisse, die natürlich je nach Fragestellung verschiedene Werte annehmen kann. Wir können uns für die Wahrscheinlichkeit interessieren, dass ein solches Hochwasserereignis während des 100-Jahres-Intervalls nicht auftritt, $P(x = 0)$, oder wir wollen die Wahrscheinlichkeit wissen, dass ein solches Hochwasserereignis genau einmal während des 100-Jahres-Intervalls auftritt, also $P(x = 1)$, oder wir wollen die Wahrscheinlichkeit wissen, dass zwei oder mehr solcher Hochwasserereignisse während des 100-Jahres-Intervalls auftreten, also $P(x \geq 2)$. Setzt man diese Werte in die obige Gleichung ein, so erhält man

$$\lambda = 1, x = 0, 1, 2, \dots, n$$

$$P(X = 0) = e^{-1} \frac{1 \times 0}{0!}, \quad \text{für } x = 0$$

$$P(X = 1) = e^{-1} \frac{1 \times 1}{1!}, \quad \text{für } x = 1$$

$$P(X \geq 2) = \sum_{i=2}^n e^{-1} \frac{1 \times x_i}{x_i!}, \quad \text{für } x_i = 2, 3, \dots, n$$

Wir wenden uns an Python, um die Berechnungen durchzuführen. Wir werden die Funktionen `poisson.pmf` und `poisson.cdf` verwenden.

```
x_0 = poisson.pmf(0, mu=1)
x_0
```

```
np.float64(0.36787944117144233)
```

```
x_1 = poisson.pmf(1, mu=1)
x_1
```

```
np.float64(0.36787944117144233)
```

$$P(X \geq 2) = 1 - P(X = 1) - P(X = 0)$$

```
xge2 = 1 - x_1 - x_0
xge2
```

```
np.float64(0.26424111765711533)
```

Alternativ können wir die `poisson.cdf` verwenden.

```
1 - poisson.cdf(1, mu=1)
```

```
np.float64(0.26424111765711533)
```

Die Ergebnisse zeigen, dass die Wahrscheinlichkeit, dass während eines Zeitraums von 100 Jahren kein Hochwasser $P(X = 0)$ in einer Größenordnung auftritt, die einem Jahrhunderthochwasser entspricht, 0,37 beträgt, was interessanterweise genauso wahrscheinlich ist wie das Auftreten von genau einem Hochwasser $P(X = 1)$. Die Wahrscheinlichkeit, dass zwei oder mehr $P(X \geq 2)$ solcher Hochwasserereignisse innerhalb des 100-Jahres-Intervalls auftreten, ist 0,26 und damit geringer. Beachten Sie jedoch, dass die Wahrscheinlichkeit, dass zwei oder mehr $P(X \geq 2)$ solcher Hochwasserereignisse während des 100-Jahres-Intervalls eintreten, etwa 26% beträgt!

Zur Überprüfung der Richtigkeit addieren wir die Wahrscheinlichkeiten $P(x = 0)$, $P(x = 1)$ und $P(x \geq 2)$, was 1 ergeben sollte,

```
x_0 + x_1 + xge2
```

```
np.float64(1.0)
```

Zur besseren Veranschaulichung stellen wir die Wahrscheinlichkeiten der Poisson-Zufallsvariablen $x = 0, 1, 2, 3, 4, \geq 5$ dar.

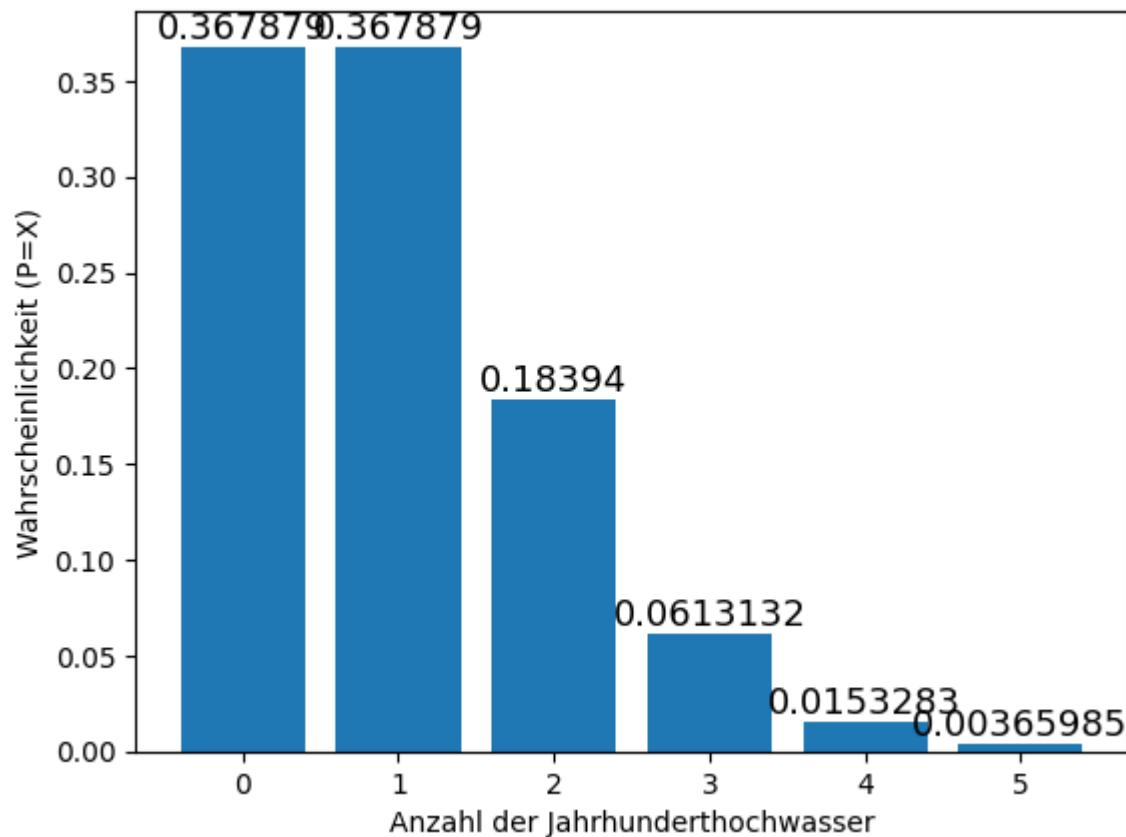
```
xs = [0, 1, 2, 3, 4, 5]
res = []

for x in xs:
    if x != 5:
        p = poisson.pmf(x, mu=1)
    else:
        p = 1 - poisson.cdf(4, mu=1)
    res.append(p)

fig, ax = plt.subplots()
ax.bar(xs, height=res)
_ = ax.bar_label(ax.containers[0], label_type="edge", size=13)

ax.set_xlabel("Anzahl der Jahrhunderthochwasser")
ax.set_ylabel("Wahrscheinlichkeit (P=X)")
```

Text(0, 0.5, 'Wahrscheinlichkeit (P=X)')



Form, Mittelwert und Standardabweichung einer Poisson-Verteilung

Alle Poisson-Verteilungen sind rechtsschief. Der Mittelwert μ und die Standardabweichung σ einer Poisson-Zufallsvariablen mit dem Parameter λ sind

$$\mu = \lambda$$

und

$$\sigma = \sqrt{\lambda}.$$

Poisson-Approximation an die Binomialverteilung

In Situationen, in denen n groß und p sehr klein ist, kann die Poisson-Verteilung zur Annäherung an die Binomialverteilung verwendet werden. Erinnern Sie sich an die binomische Wahrscheinlichkeitsverteilung:

$$P(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}, \quad x = 0, 1, 2, \dots, n$$

Wie im früheren Beispiel "Jahrhunderthochwasser" ist n eine große Zahl (100) und p eine kleine Zahl (0,01). Einsetzen in die Gleichung von oben $P(x = 1)$ ergibt

$$\begin{aligned} P(X = 1) &= \binom{100}{1} \times 0,01^1 \times (1 - 0,01)^{100-1} \\ &= 100 \times 0,01 \times 0,3697296 \\ &= 0,3697296 \end{aligned}$$

Das Ergebnis kommt dem oben ermittelten Ergebnis sehr nahe `poisson.pmf(1,1)` = 0,3678794. Die geeignete Poisson-Verteilung ist diejenige, deren Mittelwert gleich dem der Binomialverteilung ist, d. h. $\lambda = np$, was in unserem Beispiel $\lambda = 100 \times 0,01 = 1$ ist.

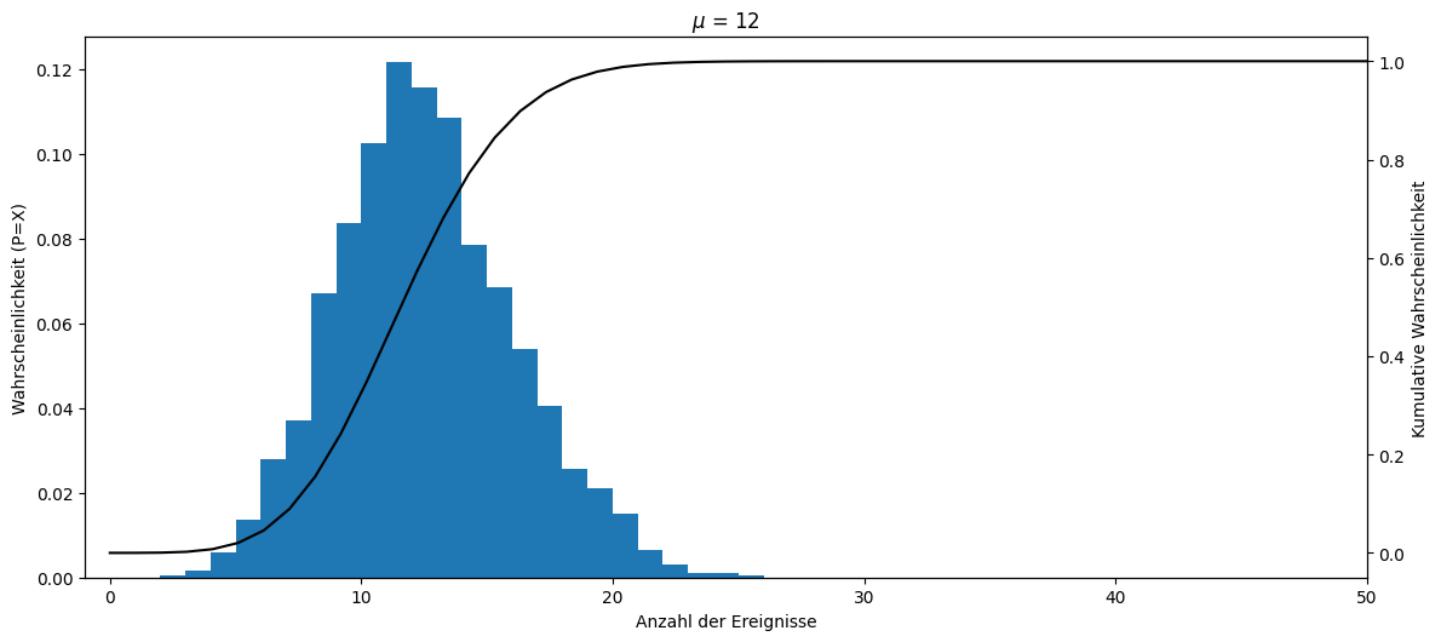
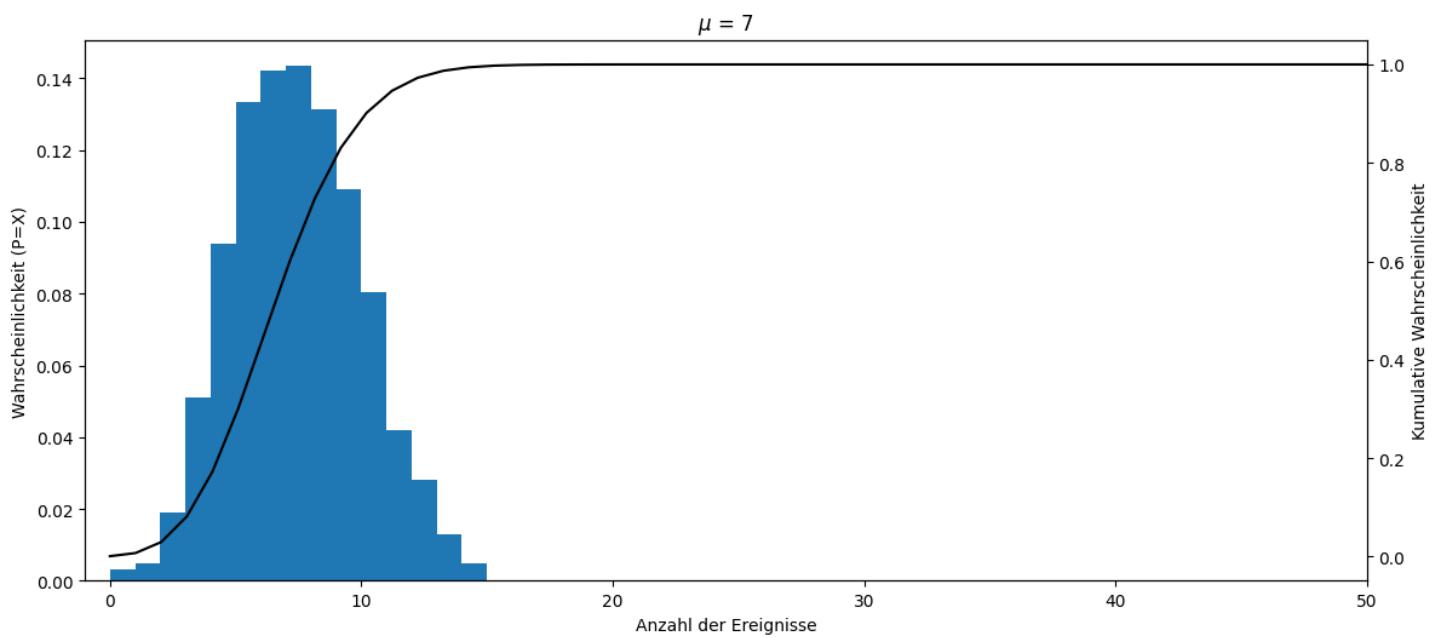
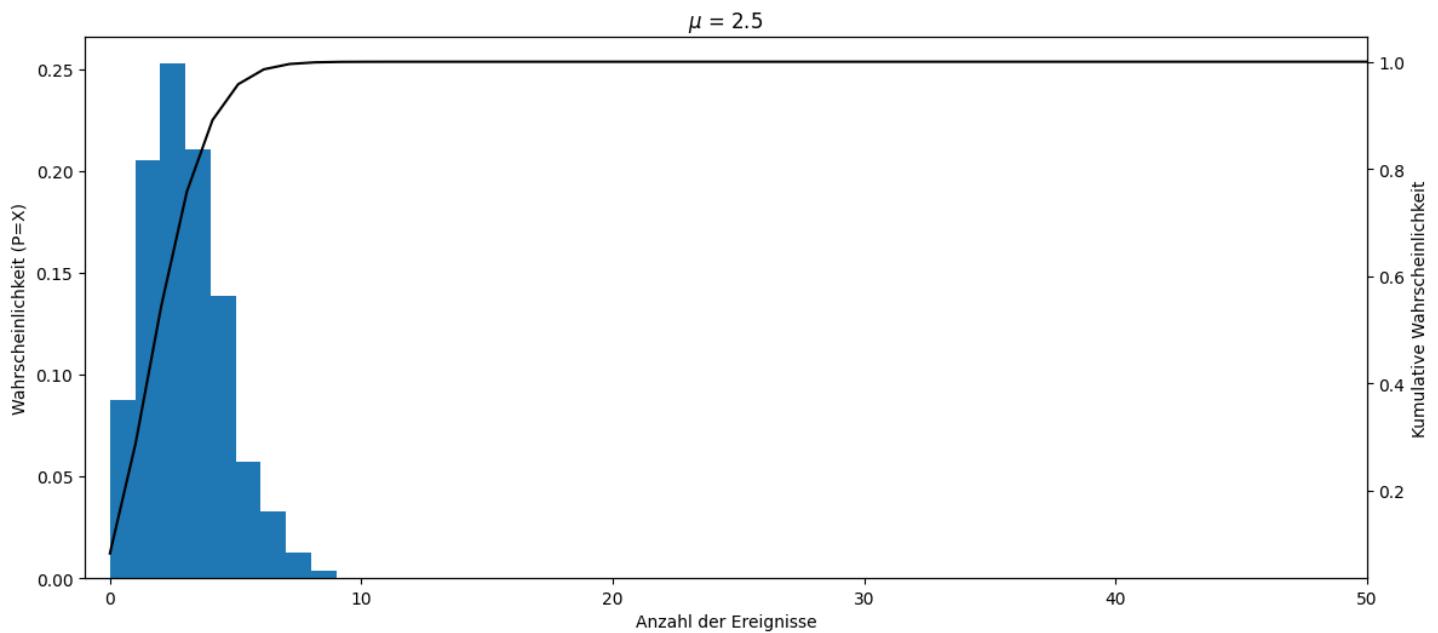
Zum Abschluss dieses Abschnitts und um Ihnen eine Vorstellung von den Formen der verschiedenen Poisson-Wahrscheinlichkeitsverteilungen zu geben, werden im Folgenden drei verschiedene Poisson-Wahrscheinlichkeitsverteilungen und die entsprechenden kumulativen Poisson-Wahrscheinlichkeitsverteilungen für $\lambda = 2, 5, \lambda = 7$ und $\lambda = 12$ angegeben.

```
lambdas = [2.5, 7, 12]

fig, ax = plt.subplots(nrows=len(lambdas), figsize=(12, 16))
for e, _lambda in enumerate(lambdas):
    x = poisson.rvs(_lambda, size=2000)
    bins = max(x) - min(x)
    ax[e].hist(x, bins, density=True)
    ax[e].set_xlabel("Anzahl der Ereignisse")
    ax[e].set_ylabel("Wahrscheinlichkeit (P=X)")
    ax[e].set_title(f"$\mu$ = {_lambda}")

    x = np.linspace(0, 50, 50)
    cdf = poisson.cdf(x, mu=_lambda)
    ax2 = ax[e].twinx()
    ax2.set_ylabel("Kumulative Wahrscheinlichkeit")
    ax2.plot(x, cdf, color="k")

    ax[e].set_xlim(-1, 50)
fig.tight_layout()
```



Übungsaufgaben

Häufigkeiten und Erwartungswert

Betrachten wir den Würfelwurf für einen fairen sechseitigen Würfel. Bei 10 Würfen werden die folgenden Zahlen gewürfelt: 1, 1, 2, 5, 6, 3, 4, 2, 4, 5

1. Erklären Sie anhand dieser Stichprobe relative und absolute Häufigkeiten.
2. Berechnen Sie den Erwartungswert der Stichprobe.
3. Berechnen Sie den Erwartungswert für einen Würfel unter der Annahme von Laplacewahrscheinlichkeit (gleicher Wahrscheinlichkeit für alle x_i von X)

Frage 1 ...

Frage 2 ...

Frage 3 ...

Lösungen

Frage 1

Absolute Häufigkeiten $h(a_j) = h_j$ entsprechen der Anzahl der Ereignisse/Elemente/Messergebnisse x_i für die gilt $x_i = a_j$. **Relative Häufigkeiten** entsprechen den **absoluten Häufigkeiten** geteilt durch die Gesamtanzahl n betrachteten Ereignisse/Elemente/Messergebnisse $f(a_j) = f_j = \frac{h_j}{n}$ ([] s.30).

Angewandt auf die Stichprobe ergeben sich die absoluten und relativen Häufigkeiten wie folgt:

X	Absolute Häufigkeit	Relative Häufigkeit
1	2	0,2
2	2	0,2
3	1	0,1
4	2	0,2
5	2	0,2
6	1	0,1

Frage 2

$$E(X) = \sum_{i=1}^6 P(x_i)x_i = 1 \cdot 0, 2 + 2 \cdot 0, 2 + 3 \cdot 0, 1 + 4 \cdot 0, 2 + 5 \cdot 0, 2 + 6 \cdot 0, 1 = 3, 3$$

Frage 3

Da von Laplacewahrscheinlichkeit ausgegangen werden kann gilt: $P(x_i) = \frac{1}{6} \approx 0,167$

$$E(X) = \sum_{i=1}^6 P(x_i)x_i = 1 \cdot 0,167 + 2 \cdot 0,167 + 3 \cdot 0,167 + 4 \cdot 0,167 + 5 \cdot 0,167 + 6 \cdot$$

Normierung

Eine Zufallsvariable X kann die Werte 1, 2, 3, 4 annehmen. Angenommen $f(x) = P(X = x)$ ist die dazugehörige Wahrscheinlichkeitsfunktion und ist gegeben durch:

$$f(x) = \begin{cases} \frac{x(b-x)}{10}, & x \in \{1, 2, 3, 4\} \\ 0, & \text{sonst} \end{cases}$$

Wie muß b gewählt werden um die Normierung der Wahrscheinlichkeitsfunktion zu gewährleisten ?

Frage 1 ...

Lösungen

Um die Wahrscheinlichkeitsfunktion zu normieren muß

$$\sum_{i=1}^n f(x_i) = 1$$

gelten.

$$\sum_{i=1}^4 f(x_i) = \frac{1 \cdot (b-1)}{10} + \frac{2 \cdot (b-2)}{10} + \frac{3 \cdot (b-3)}{10} + \frac{4 \cdot (b-4)}{10} = 1$$

$$(b-1) + (2b-4) + (3b-9) + (4b-16) = 10$$

$$b + 2b + 3b + 4b - 1 - 4 - 9 - 16 = 10$$

$$10b - 30 = 10$$

$$\Rightarrow b = 4$$

Binomialverteilung

In einer Fabrik werden serienmäßig Tübel mit einem Ausschussanteil von 4,5% hergestellt, d. h. unter 1000 hergestellten Tübel befinden sich im Mittel genau 45 unbrauchbare Tübel.

1. Mit welchen Wahrscheinlichkeiten finden wir in einer Zufallsstichprobe von 10 Tübel genau 0, 1, 3, 5, 8 unbrauchbare Tübel?

2. Mit welchen Wahrscheinlichkeiten finden wir in einer Zufallsstichprobe von 10 Tübel 3 oder mehr unbrauchbare Tübel?

Frage 1 ...

Frage 2 ...

Lösungen

Es handelt sich hier um ein Bernoulli-Experiment. Die Wahrscheinlichkeit, beim Ziehen eines Tübels einen unbrauchbaren Tübels zu erhalten, liegt bei $p = 0,04$. Die Zufallsvariable folgt der Binomialverteilung mit den Parametern $n = 10$ und $p = 0,4$.

Somit ergibt sich

$$P(X = x) = \binom{10}{x} \times 0,04^x \times (1 - 0,04)^{(10-x)} \quad (x = 0, 1, 3, 5, 8)$$

► Show code cell content

$$P(X \geq 3) = 1 - P(X = 0) - P(X = 1) - P(X = 2)$$

► Show code cell content

Lernziele

Stetige Zufallsvariablen

- Die Konzepte von diskreten Zufallsvariablen werden auf stetige Zufallsvariablen und ihre Wahrscheinlichkeitsverteilungen verallgemeinert
- Die Wahrscheinlichkeitsdichtefunktion und ihre Rolle in der Konstruktion von Wahrscheinlichkeitsverteilungen wird erläutert. Insbesonders werden Normalverteilung, die

Standard-Normalverteilung und die Umrechnung auf standardisierte Zufallsvariablen, kontinuierliche gleichmäßige Verteilung, Studentsche t-Verteilung, Chi-Quadrat-Verteilung und F-Verteilung besprochen

- Es wird erklärt wie diese Wahrscheinlichkeitsverteilungen berechnet werden können und deren Syntax und Anwendungen durch Beispiele in Python veranschaulicht
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : stetige Zufallsvariablen, Wahrscheinlichkeitsdichtefunktionen, Normalverteilung, Standard-Normalverteilung, kontinuierliche gleichmäßige Verteilung, Studentsche t-Verteilung, Chi-Quadrat-Verteilung, F-Verteilung

Stetige Zufallsvariablen und ihre Wahrscheinlichkeitsverteilungen

Eine [Zufallsvariable](#), deren Werte nicht abzählbar sind, nennt man eine **stetige Zufallsvariable**. D.h. eine kontinuierliche Zufallsvariable kann jeden Wert annehmen (z.B. reelle Zahlenwerte), der in einem oder mehreren Intervallen enthalten ist. Da die Anzahl der in einem Intervall enthaltenen Werte unendlich ist, ist auch die mögliche Anzahl der Werte, die eine kontinuierliche Zufallsvariable annehmen kann, unendlich ([] s.251).

Es gibt viel mehr [kontinuierliche Wahrscheinlichkeitsverteilungen](#), als wir hier besprechen können. Beachten Sie jedoch, dass in Python mittlerweile eine große Anzahl verschiedener diskreter und kontinuierlicher Wahrscheinlichkeitsverteilungen implementiert sind, siehe [hier](#).

In Python sind Wahrscheinlichkeitsfunktionen durch allgemeine Methoden wie `rvs`, `pdf`, `cdf` und `ppf` zugänglich. `rvs` ist das allgemeine Syntax für Zufallsvariablen generatoren wie `uniform.rvs()` für die Gleichverteilung oder `norm.rvs()` für die Normalverteilung. `pdf` ist Methode für die Wahrscheinlichkeitsdichtefunktion wie `uniform.pdf` und `norm.pdf()`. Das `cdf` ist der Syntax für die kumulative Dichtefunktion wie `uniform.cdf()` und `norm.cdf()`. Das `ppf` ist der allgemeine Syntax für die Quantilfunktion, wie `uniform.ppf()` und `norm.ppf()`. Behalten Sie das im Hinterkopf, wenn wir die Kapazitäten in Python weiter erforschen.

Wahrscheinlichkeitsdichtefunktionen

Die Form der Verteilung einer Zufallsvariablen kann durch eine glatte Kurve veranschaulicht werden. Solche Kurven, die die Verteilung von kontinuierlichen Variablen darstellen, werden

Wahrscheinlichkeitsdichtefunktionen (PDF) oder einfach **Dichtefunktionen** genannt.

Wahrscheinlichkeitsdichtefunktionen haben drei Haupteigenschaften ([] s.327):

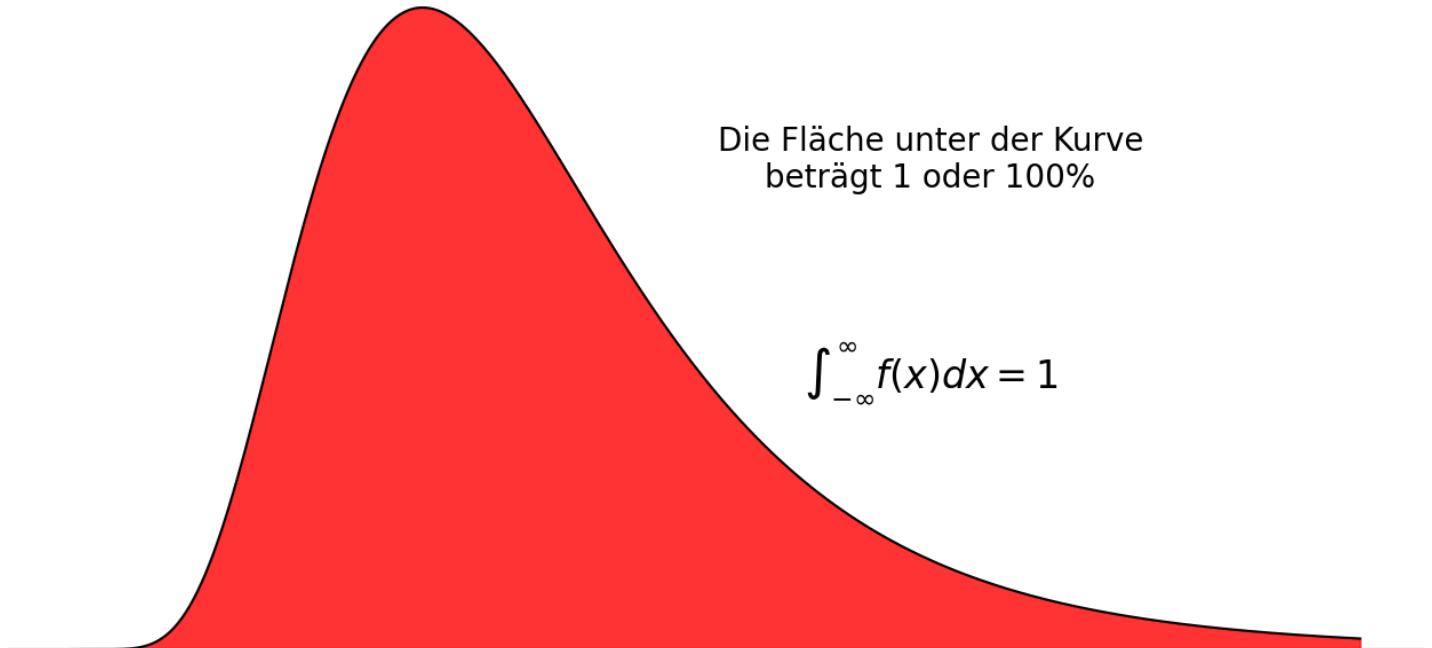
1. Eine PDF wird immer auf oder über der horizontalen Achse gezeichnet
2. Die Gesamtfläche zwischen einer PDF und der horizontalen Achse ist gleich 1 und somit liegt jeder Wert in jedem Teilintervall der PDF im Bereich von 0 bis 1
3. Alle möglichen Beobachtungen der Variablen, die innerhalb eines bestimmten Bereichs liegen, entsprechen der entsprechenden Fläche unter der Dichtefunktion und können als prozentueller Anteil ausgedrückt werden.

Die Fläche unter der Kurve wird durch das Integral des Wertes x von $-\infty$ bis $+\infty$ berechnet und in der Regel auf den Wert 1 normiert.

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

► Show code cell source

```
Text(2, 0.4, '$\\int_{-\\infty}^{\\infty} f(x)dx=1$')
```



Die Wahrscheinlichkeit, dass eine stetige Zufallsvariable x einen Wert innerhalb eines bestimmten Intervalls annimmt, ist durch die Fläche unter der Kurve zwischen den beiden Grenzen des Intervalls

gegeben. Die farbige Fläche unter der Kurve für das Intervall $]-\infty, a]$ (linkes Feld) und für das Intervall $[a, +\infty[$ (rechtes Feld) ist in der folgenden Abbildung dargestellt.

Die Wahrscheinlichkeit, dass x in das Intervall $]-\infty, a]$ fällt, ist

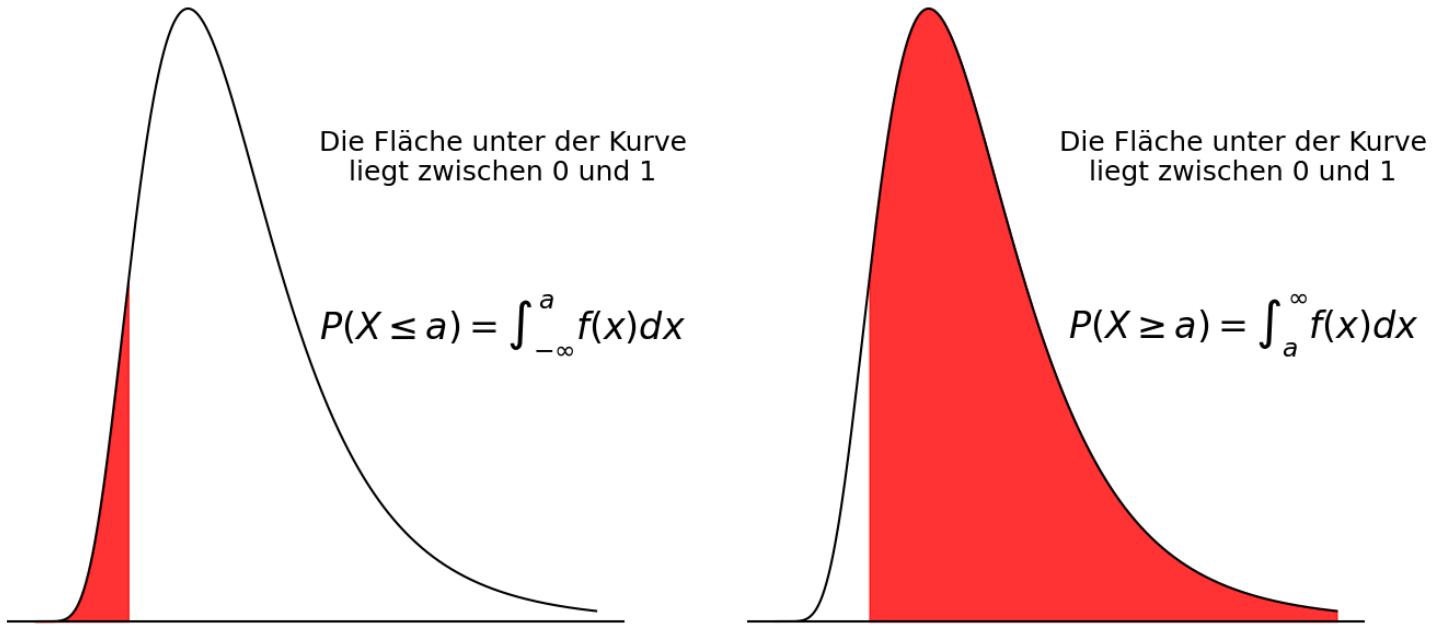
$$P(X \leq a) = \int_{-\infty}^a f(x)dx$$

und die Wahrscheinlichkeit, dass x in das Intervall $[a, +\infty[$ fällt, ist

$$P(X \geq a) = 1 - P(X \leq a) = \int_a^{\infty} f(x)dx$$

► Show code cell source

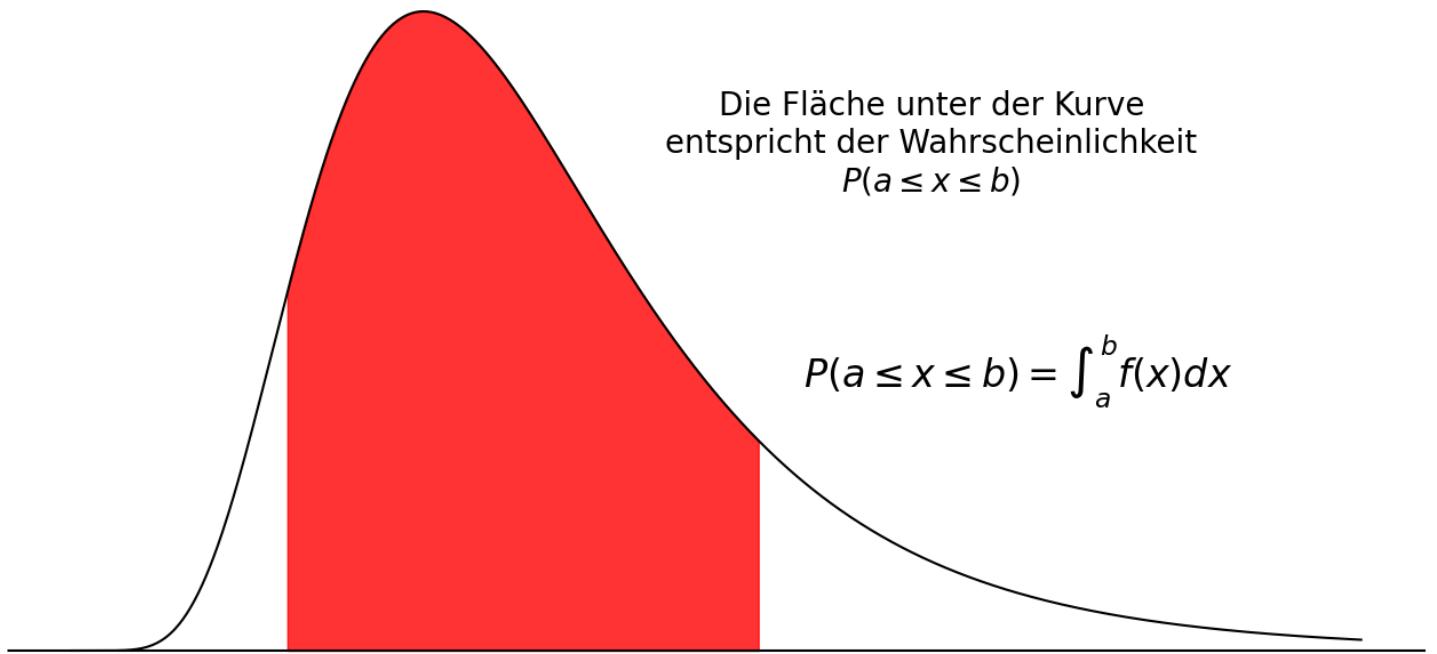
```
Text(2.5, 0.45, '$P(X \geq a) = \int_a^{\infty} f(x)dx$')
```



Die Wahrscheinlichkeit, dass eine kontinuierliche Zufallsvariable x einen Wert innerhalb eines bestimmten Intervalls annimmt, ist durch die Fläche unter der Kurve zwischen den beiden Grenzen des Intervalls gegeben. Der Wert der farbigen Fläche unter der Kurve von a bis b in der folgenden Abbildung gibt die Wahrscheinlichkeit an, dass x in das Intervall $[a, b]$ fällt.

► Show code cell source

Text(2.2, 0.4, '\$P(a \leq x \leq b) = \int_a^b f(x)dx\$')



$$P(a \leq x \leq b) = \int_a^b f(x)dx$$

$$= P(x \leq b) - P(x \leq a)$$

$$= \int_{-\infty}^b f(x)dx - \int_{-\infty}^a f(x)dx$$

Man beachte, dass das Intervall $a \leq x \leq b$ besagt, dass x größer oder gleich a , aber kleiner oder gleich b ist.

Bei einer kontinuierlichen Wahrscheinlichkeitsverteilung wird die Wahrscheinlichkeit immer für ein Intervall berechnet. **Die Wahrscheinlichkeit, dass eine kontinuierliche Zufallsvariable x einen einzigen Wert annimmt, ist immer Null.** Das liegt daran, dass die Wahrscheinlichkeit, genau einen Wert aus einer unendlichen Anzahl von Werten $\in \mathbb{R}$ zu wählen, gleich Null ist. Im geometrischen Sinne bedeutet dies, dass die Fläche einer Linie, die einen einzigen Punkt darstellt, Null ist.

$$P(x) = 0$$

Daraus lässt sich ableiten, dass für eine stetige Zufallsvariable gilt

$$P(a \leq x \leq b) = P(a < x < b)$$

Mit anderen Worten: Die Wahrscheinlichkeit, dass x einen Wert im Intervall a bis b annimmt, ist gleich groß, unabhängig davon, ob die Werte a und b im Intervall enthalten sind oder nicht.

Die Normalverteilung

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm
import statsmodels.api as smi
```

Die [Normalverteilung](#) wird in der Wahrscheinlichkeitstheorie, der Statistik sowie in den Natur- und Sozialwissenschaften häufig verwendet. Sie wird auch **Gauß-Verteilung** genannt, weil [Carl Friedrich Gauß](#) (1777 – 1855) einer der ersten war, der sie für die Analyse astronomischer Daten verwendete (s.83,s.271).

Die **Normalverteilung** oder die **Normalkurve** ist eine glockenförmige (symmetrische) Kurve. Ihr Mittelwert wird mit μ und ihre Standardabweichung mit σ bezeichnet. Eine kontinuierliche Zufallsvariable x , die eine Normalverteilung aufweist, wird als **normale Zufallsvariable** bezeichnet.

Die Notation für eine Normalverteilung lautet $X \sim N(\mu, \sigma)$. Die Wahrscheinlichkeitsdichtefunktion (PDF) wird geschrieben als

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

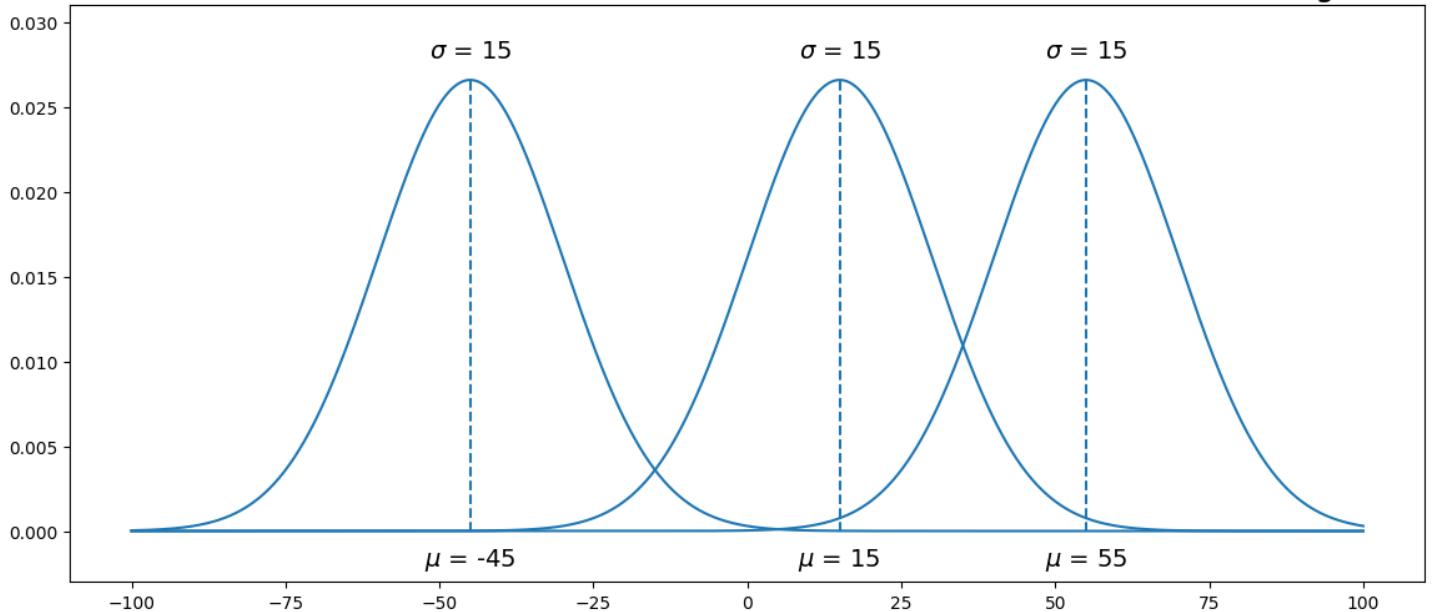
wobei $e \approx 2,71828$ und $\pi \approx 3,14159$. Die Wahrscheinlichkeitsdichtefunktion $f(x)$ gibt den vertikalen Abstand zwischen der horizontalen Achse und der Normalkurve im Punkt x an.

Die Normalverteilung wird durch zwei Parameter beschrieben, den Mittelwert μ und die Standardabweichung σ . Jeder unterschiedliche Satz von Werten für μ und σ ergibt eine andere Normalverteilung. Der Wert von μ bestimmt den Mittelpunkt einer Normalverteilungskurve auf der horizontalen Achse, und der Wert von σ gibt die Streuung der Werte um den Mittelpunkt an.

► Show code cell source

(-0.003, 0.031)

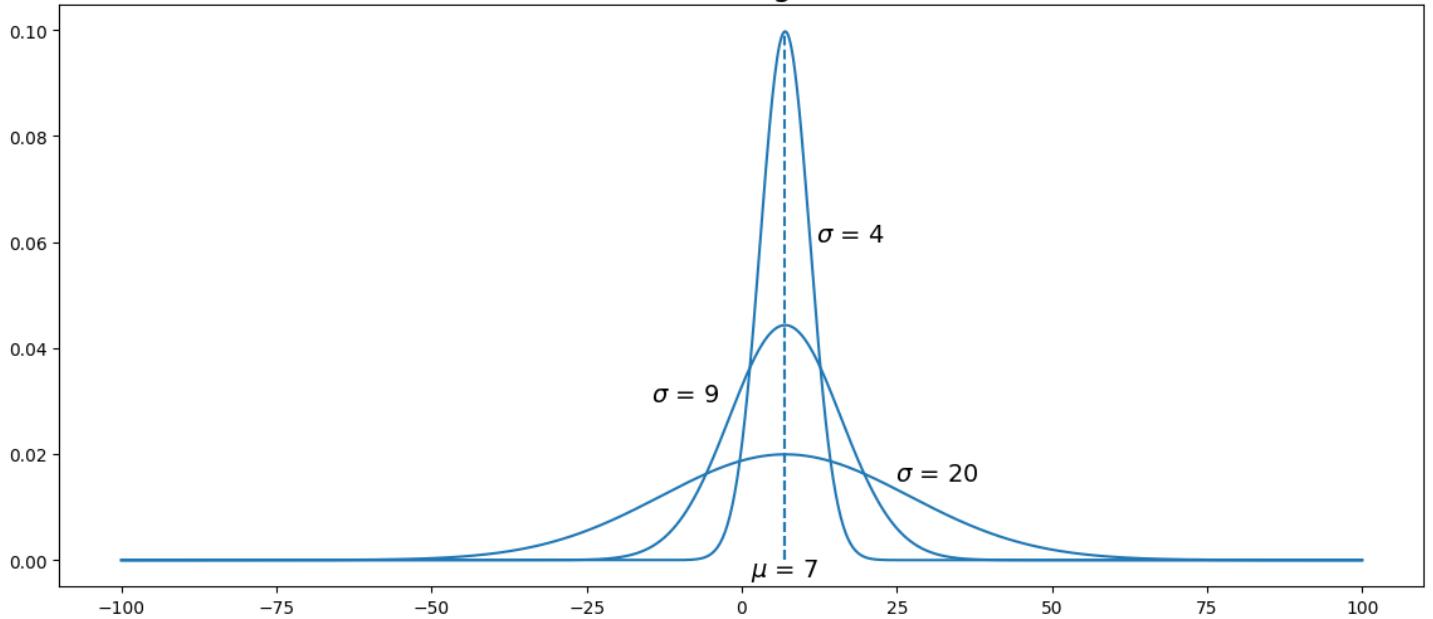
Drei Wahrscheinlichkeitsdichtefunktionen der Normalverteilung mit unterschiedlichen Mittelwerten aber mit identischen Standardabweichungen



► Show code cell source

```
Text(0.5, 1.0, 'Drei Wahrscheinlichkeitsdichtefunktionen der Normalverteilung\nmit unterschiedlichen Mittelwerten aber mit identischen Standardabweichungen')
```

Drei Wahrscheinlichkeitsdichtefunktionen der Normalverteilung mit unterschiedlichen Standardabweichungen aber mit identischen Mittelwerten

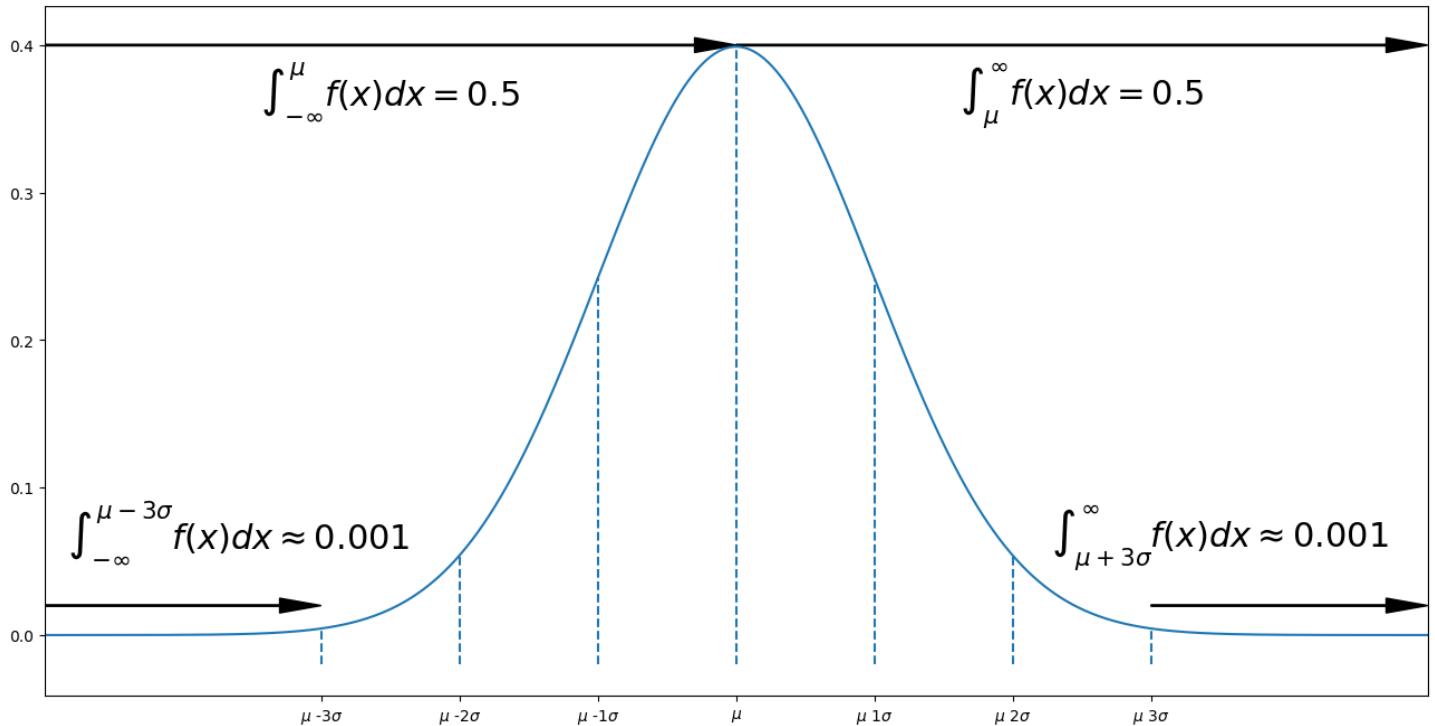


Eine Normalverteilung ist unter anderem durch die folgenden Merkmale gekennzeichnet (s.272):

1. Die Gesamtfläche unter einer Normalverteilungskurve beträgt 1,0, also 100%.
2. Eine Normalverteilungskurve ist symmetrisch um den Mittelwert. Folglich liegen 50% der Gesamtfläche unter einer Normalverteilungskurve auf der linken Seite des Mittelwerts und 50% liegen auf der rechten Seite des Mittelwerts.
3. Die Ausläufer einer Normalverteilungskurve erstrecken sich unendlich weit in beide Richtungen, ohne die horizontale Achse zu berühren oder zu kreuzen. Obwohl eine Normalverteilungskurve niemals die horizontale Achse berührt, kommt sie jenseits der Punkte, durch $\mu - 3\sigma$ und $\mu + 3\sigma$ dargestellten Punkte so nahe an diese Achse heran, dass die Fläche unter der Kurve jenseits dieser Punkte in beiden Richtungen praktisch als Null angenommen werden kann.

► Show code cell source

(-5.0, 5.0)



Die Standard-Normalverteilung

Die [Standardnormalverteilung](#) ist ein Spezialfall der Normalverteilung. Bei der Standardnormalverteilung ist der Wert des Mittelwerts gleich Null ($\mu = 0$) und der Wert der Standardabweichung gleich 1 ($\sigma = 1$).

Wenn man also $\mu = 0$ und $\sigma = 1$ in die PDF der Normalverteilung einsetzt, vereinfacht sich die Gleichung zu

$$\begin{aligned}
 f(x) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \\
 &= \frac{1}{1 \times \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-0}{1}\right)^2} \\
 &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}
 \end{aligned}$$

Die Zufallsvariable, die die Standardnormalverteilung erfüllt, wird mit z bezeichnet. Folglich werden die Einheiten für die Kurve der Standardnormalverteilung mit z bezeichnet und als **z -Werte**, **z -Scores** oder **z -Statistik** bezeichnet.

Die **kumulative Verteilungsfunktion (CDF)** der Standardnormalverteilung, die der Fläche unter der Kurve für das Intervall $]-\infty, z]$ entspricht und gewöhnlich mit dem griechischen Großbuchstaben ϕ bezeichnet wird, ist gegeben durch

$$F(x < z) = \phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z e^{-\frac{1}{2}x^2} dx$$

wobei $e \approx 2,71828$ und $\pi \approx 3,14159$.

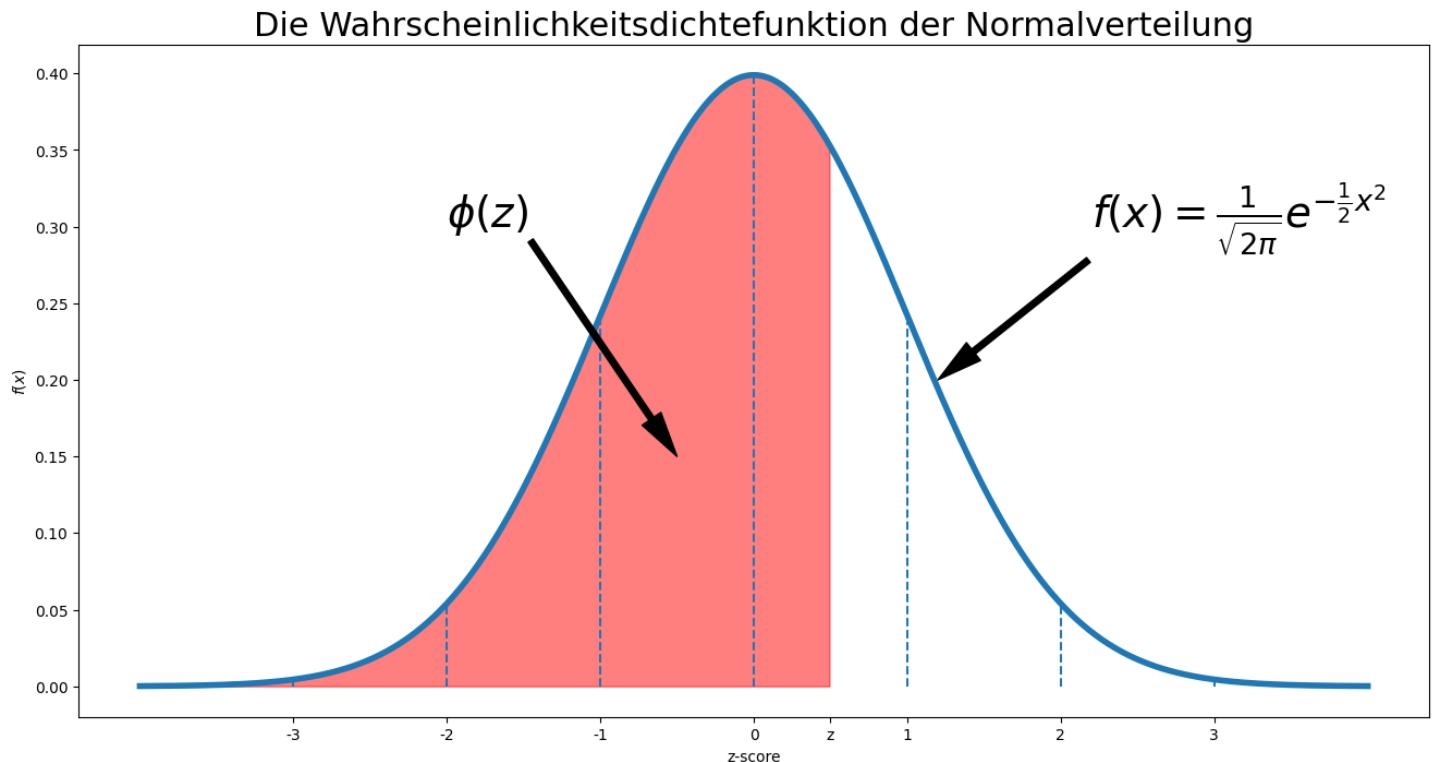
Grundlegende Eigenschaften der Standardnormalkurve

Die Standardnormalkurve ist ein Spezialfall der Normalverteilung und damit auch eine Wahrscheinlichkeitsverteilungskurve. Daher gelten die grundlegenden Eigenschaften der Normalverteilung auch für die Standardnormalkurve (s.85).

1. Die Gesamtfläche unter der Standardnormalkurve ist 1 (diese Eigenschaft ist allen Dichtekurven gemeinsam).
2. Die Standardnormalkurve erstreckt sich unendlich in beide Richtungen und nähert sich dabei der horizontalen Achse, berührt sie aber nie.
3. Die Standardnormalkurve ist glockenförmig, ihr Mittelpunkt liegt bei $z = 0$. Fast die gesamte Fläche unter der Standardnormalkurve liegt zwischen $z = -3$ und $z = 3$.

► Show code cell source

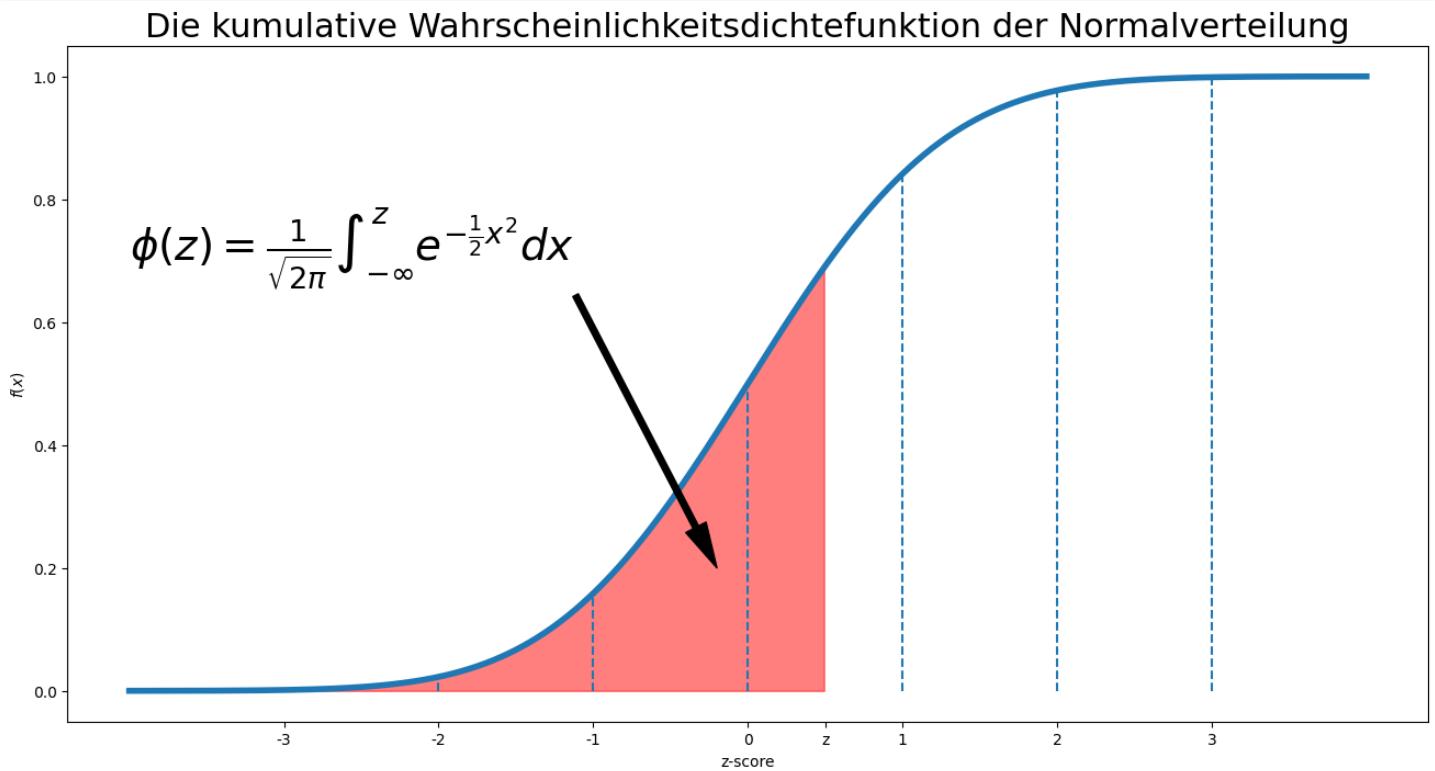
Text(0.5, 1.0, 'Die Wahrscheinlichkeitsdichtefunktion der Normalverteilung')



Die z -Werte auf der rechten Seite des Mittelwerts sind positiv und die auf der linken Seite sind negativ. Der z -Wert für einen Punkt auf der horizontalen Achse gibt den Abstand zwischen dem Mittelwert ($z = 0$) und diesem Punkt in Form der Standardabweichung an. Ein Punkt mit einem Wert von $z = 2$ liegt zum Beispiel zwei Standardabweichungen rechts vom Mittelwert. Ebenso liegt ein Punkt mit einem Wert von $z = -2$ zwei Standardabweichungen links vom Mittelwert.

► Show code cell source

Text(0.5, 1.0, 'Die kumulative Wahrscheinlichkeitsdichtefunktion der Normalverteilung')



Das Methode die Wahrscheinlichkeiten durch Berechnung der Fläche unter der Standardnormalkurve zu bestimmen, kommt häufig zur Anwendung. Aus diesem Grund gibt es [Wahrscheinlichkeitstabellen](#), um die Fläche für einen bestimmten z -Wert zu ermitteln. Python ist jedoch ein so leistungsfähiges Werkzeug, dass wir die Fläche unter der Kurve für einen bestimmten z -Wert berechnen können.

Um die Fläche unter der Kurve für eine Standardnormalverteilung zu berechnen, verwenden wir zunächst die Funktion `norm` aus dem `scipy.stats` Paket um eine Standardnormalverteilung zu generieren und wenden darauf die Methode `cdf` an um die kumulative Wahrscheinlichkeit zu berechnen. Die Funktion `norm` ist definiert als `norm(loc = Mittelwert, scale = Standardabweichung)`. Um die Standardwerte zu erhalten setzen wir den Mittelwert und die Standardabweichung jeweils auf 0 und 1 sind. Wenden wir die Methode `cdf` an bekommen wir die kumulative Wahrscheinlichkeit bis zum angegebenen Punkt. Wir berechnen die Fläche unter der Kurve für $z = -3, -2, -1, 0, 1, 2, 3$ oder formeller geschrieben:

$$P(x \leq z) \quad \forall z \in (-3, -2, -1, 0, 1, 2, 3)$$

```
norm.cdf(-3)
```

```
np.float64(0.0013498980316300933)
```

```
norm.cdf(-2)
```

```
np.float64(0.022750131948179195)
```

```
norm.cdf(-1)
```

```
np.float64(0.15865525393145707)
```

```
norm.cdf(0)
```

```
np.float64(0.5)
```

```
norm.cdf(1)
```

```
np.float64(0.8413447460685429)
```

```
norm.cdf(2)
```

```
np.float64(0.9772498680518208)
```

```
norm.cdf(3)
```

```
np.float64(0.9986501019683699)
```

Perfekt! Wir haben einige der oben genannten Eigenschaften einer Standardnormalkurve bestätigt. Wir erinnern uns, dass wir die Fläche unter der Kurve für das Intervall $]-\infty, z]$ berechnet haben.

Der Aufruf von `norm.cdf(-3)` ergibt eine sehr geringe Zahl. Nur etwa 0,1% der gesamten Fläche unter der Kurve befinden sich links von $z = -3$, was dem Abstand der dreifachen Standardabweichung vom Mittelwert entspricht. Außerdem ergibt `norm.cdf(0)` 50%. Fantastisch! Daraus schließen wir, dass die Fläche unter der Kurve für das Intervall $]-\infty, 0]$ die gleiche ist wie die Fläche unter der Kurve für das Intervall $[0, \infty[$ und dass die Fläche unter der Kurve sich zu 1 aufsummiert. Auch hier haben wir eine der oben genannten Eigenschaften einer Standardnormalkurve bestätigt. Und schließlich ergibt der Aufruf von `norm.cdf(3)` eine hohe Zahl nahe bei 1. Somit sind etwa 99,9% der Fläche unter der Kurve im Intervall $]-\infty, 3]$ zu finden. Für den Bereich jenseits von $z = 3$ bleibt nur wenig übrig.

Es sei daran erinnert, dass wir die Fläche unter der Kurve für jedes beliebige Intervall explizit berechnen können

$$P(a \leq z \leq b) = P(z \leq b) - P(z \leq a)$$

$$= \int_a^b f(z) dz$$

$$= \int_{-\infty}^b f(x) dx - \int_{-\infty}^a f(x) dx$$

Berechnen wir die Fläche unter der Kurve für die folgenden Intervalle: $[-1, 1]$, $[-2, 2]$, $[-3, 3]$. Oder in Worten: Bestimmen wir die Fläche unter der Kurve für ± 1 Standardabweichung, für ± 2 Standardabweichungen und für ± 3 Standardabweichungen.

`norm.cdf(1) - norm.cdf(-1)`

`np.float64(0.6826894921370859)`

`norm.cdf(2) - norm.cdf(-2)`

`np.float64(0.9544997361036416)`

`norm.cdf(3) - norm.cdf(-3)`

`np.float64(0.9973002039367398)`

Toll, wir haben soeben die Empirische Regel (s.86), auch bekannt als **68 – 95 – 99, 7-Regel**, bestätigt, die sich auf den [Tschebyscheffsche Ungleichung](#) bezieht. Für eine glockenförmige Verteilung sind die 3 Regeln dass ungefähr

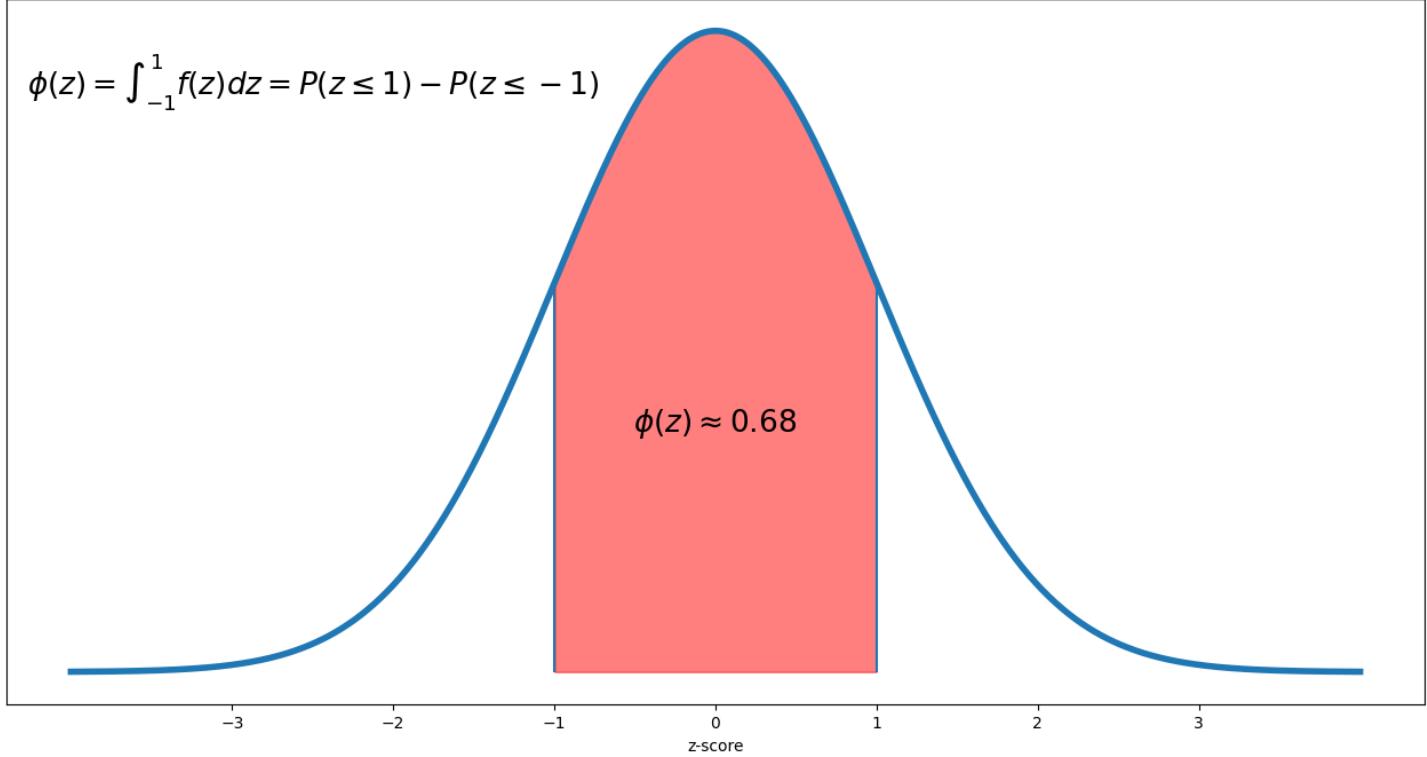
1. 68% der Beobachtungen liegen innerhalb einer Standardabweichung des Mittelwerts,
2. 95% der Beobachtungen liegen innerhalb von zwei Standardabweichungen des Mittelwerts, und
3. 99, 7% der Beobachtungen liegen innerhalb von drei Standardabweichungen des Mittelwerts.

Um unsere Intuition zu stärken, wird die empirische Regel im Folgenden veranschaulicht.

► Show code cell source

```
Text(0.5, 1.0, 'Die Fläche des Intervalls $z=[-1,1]$')
```

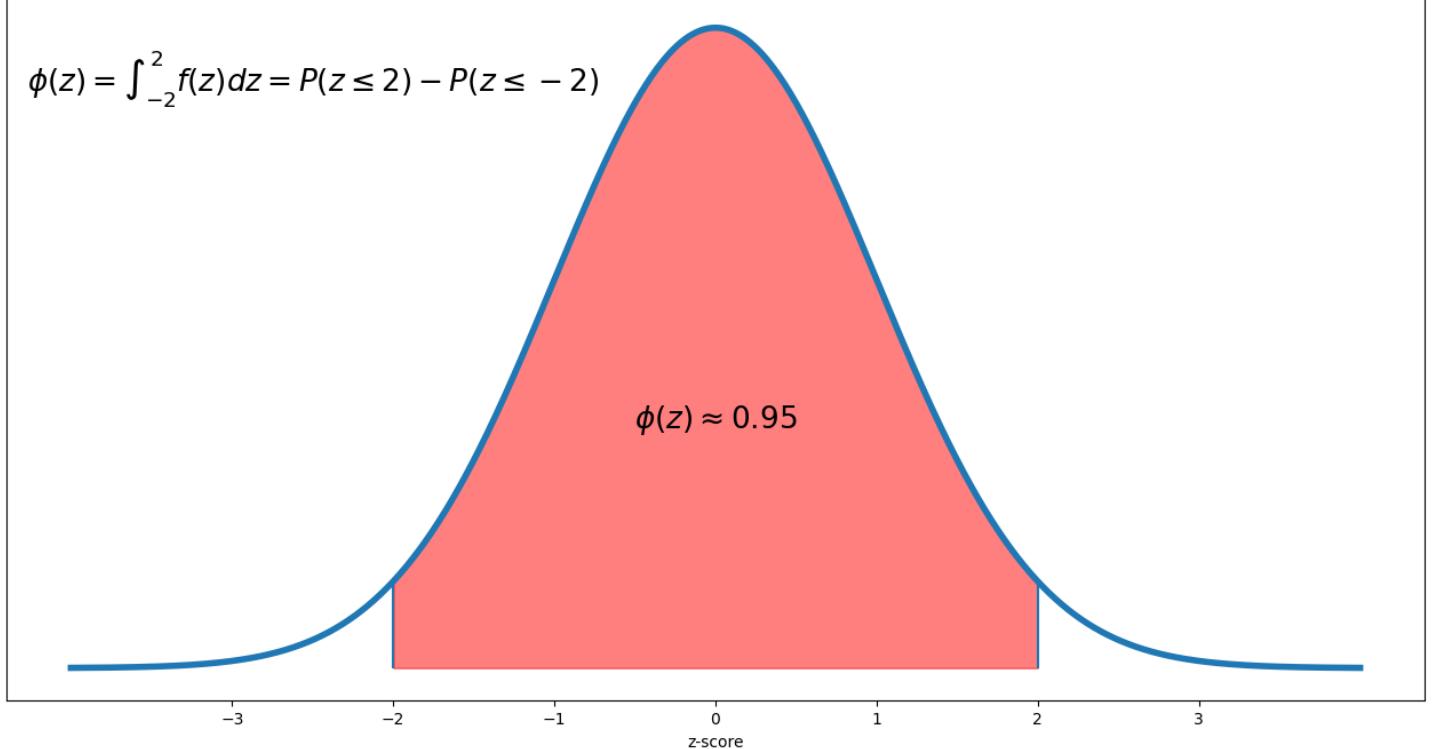
Die Fläche des Intervalls $z = [-1, 1]$



► Show code cell source

```
Text(0.5, 1.0, 'Die Fläche des Intervalls $z=[-2,2]$')
```

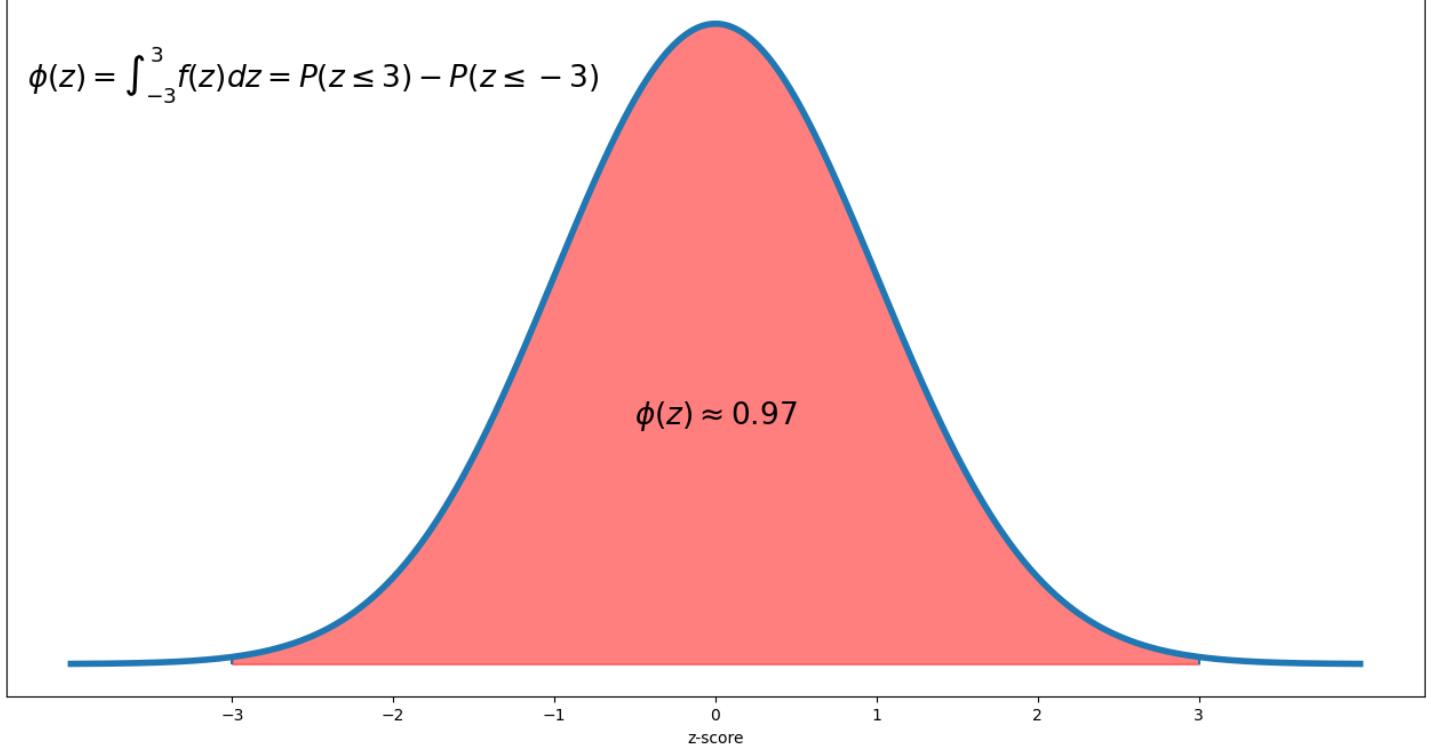
Die Fläche des Intervalls $z = [-2, 2]$



► Show code cell source

```
Text(0.5, 1.0, 'Die Fläche des Intervalls $z=[-3,3]$')
```

Die Fläche des Intervalls $z = [-3, 3]$



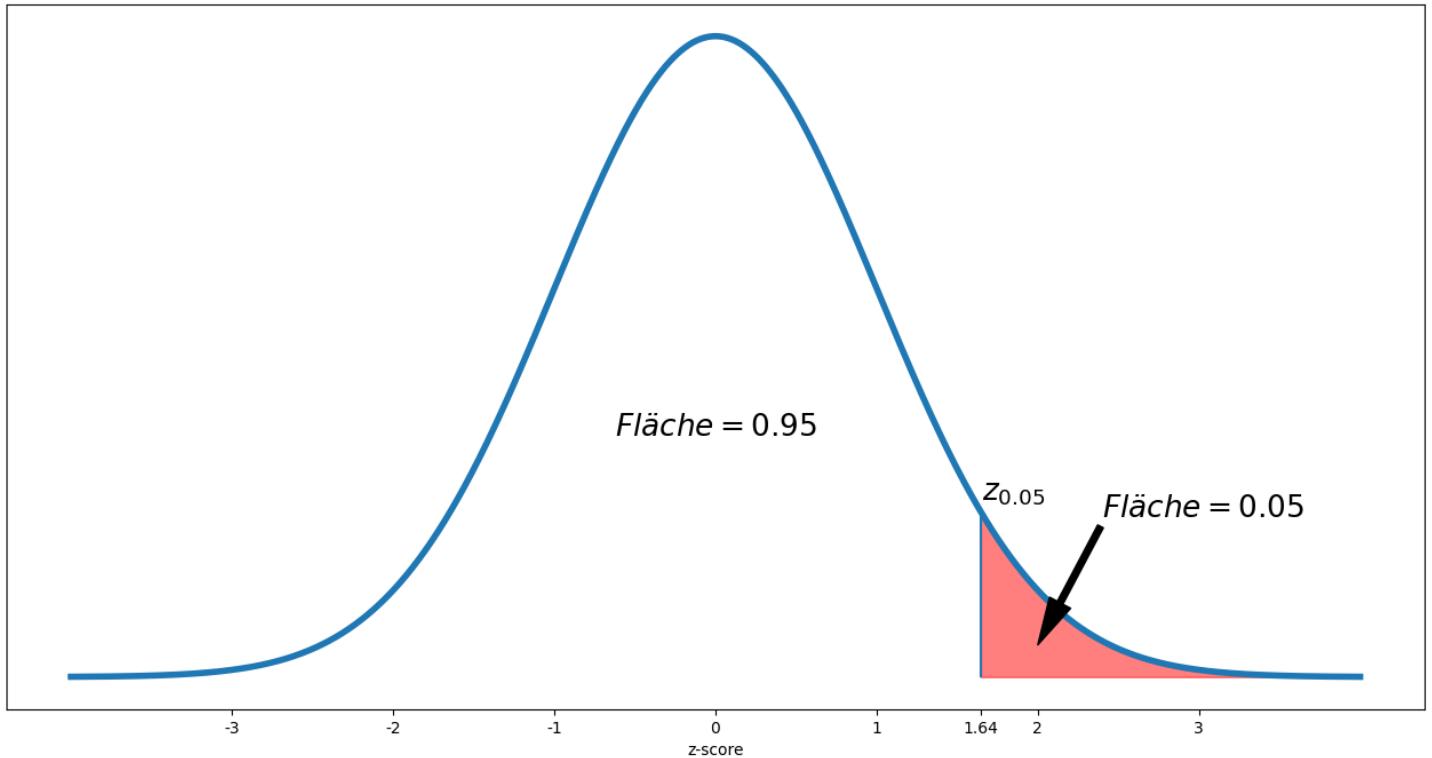
Bestimmung des z -Wertes, bei bekannter Fläche unter der Normalverteilungskurve

Bisher haben wir z -Scores verwendet, um die Fläche unter der Kurve zu berechnen. Jetzt machen wir es andersherum. Wir berechnen den oder die z -Score(s), die einer bestimmten Fläche unter der Standardnormalkurve entsprechen. Das Auffinden des z -Scores, der eine bestimmte Fläche hat, ist so häufig, dass es eine spezielle Notation gibt. Das Symbol z_α wird verwendet, um den z -Score zu bezeichnen, der eine Fläche von α (alpha) zu seiner Rechten unter der Standardnormalkurve aufweist.

Ermitteln wir $z_{0,05}$, den z -Wert, der unter der Standardnormalkurve eine Fläche von 0,05 zu seiner Rechten hat. Der Wert von α entspricht der Wahrscheinlichkeit, einen bestimmten Wert zu erhalten, der dem Intervall $[z, \infty]$ entspricht. Denn die Fläche rechts davon ist 0,05. Die Fläche links davon ist $1 - 0,05 = 0,95$, was dem Intervall $[-\alpha, z]$ (siehe Grafik unten).

► Show code cell source

```
[]
```



Um den entsprechenden z -Score zu erhalten, kann man ihn in einer [Wahrscheinlichkeitstabelle](#) nachschlagen oder Python verwenden. Daher wenden wir die Funktion `norm.ppf` an. Die `norm.ppf`-Funktion wird geschrieben als `norm.ppf(p, mean = 0, scale = 1, loc = 0)`. Wir behalten die Standardwerte für die Argumente `mean`, `sd` und `loc` bei. Allerdings müssen wir vorsichtig sein auf welchen Bereich der Fläche unter der Normalverteilung wir uns beziehen. Für `norm.ppf(p)` erhalten wir den z -Score, bei dem das p-Argument der Bereich links von z ist. Wenn wir dagegen `norm.ppf(1-p)` berechnen, erhalten wir den z -Score, bei dem das p-Argument der Bereich rechts von z ist. Wenden wir uns an Python um dies zu verdeutlichen.

```
norm.ppf(0.05)
```

```
np.float64(-1.6448536269514729)
```

```
norm.ppf(0.95)
```

```
np.float64(1.6448536269514722)
```

Es ist interessant zu erwähnen das die Perzentile Punkt Funktion `norm.ppf` die inverse Funktion der kumulativen Wahrscheinlichkeitsfunktion `norm.cdf` ist

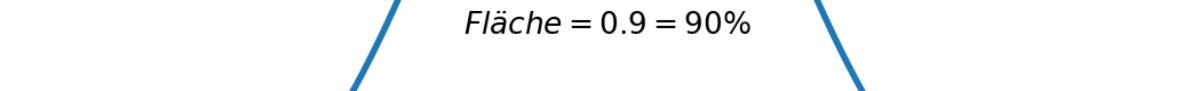
```
norm.cdf(norm.ppf(0.95))
```

```
np.float64(0.95)
```

Da die Standardnormalverteilung symmetrisch ist, erhalten wir zweimal die gleiche Zahl, aber mit einem anderen Vorzeichen. Das bedeutet, dass bei einem z-Wert von etwa 1,64 95% aller Werte links von $z_{0,05}$ und 5% aller Werte rechts davon liegen. Im Gegensatz dazu liegen für einen z-Wert von etwa -1,64 5% aller Werte links von $z_{0,05}$ und 95% aller Werte rechts davon. Kombiniert man diese, erhält man das Intervall $z \in [-1,64, 1,64]$, das 90% aller Werte abdeckt.

► Show code cell source

```
[]
```



Fläche = 0.9 = 90%

Fläche = 0.05

Fläche = 0.05

-3 -2 -1 0 1 2 3

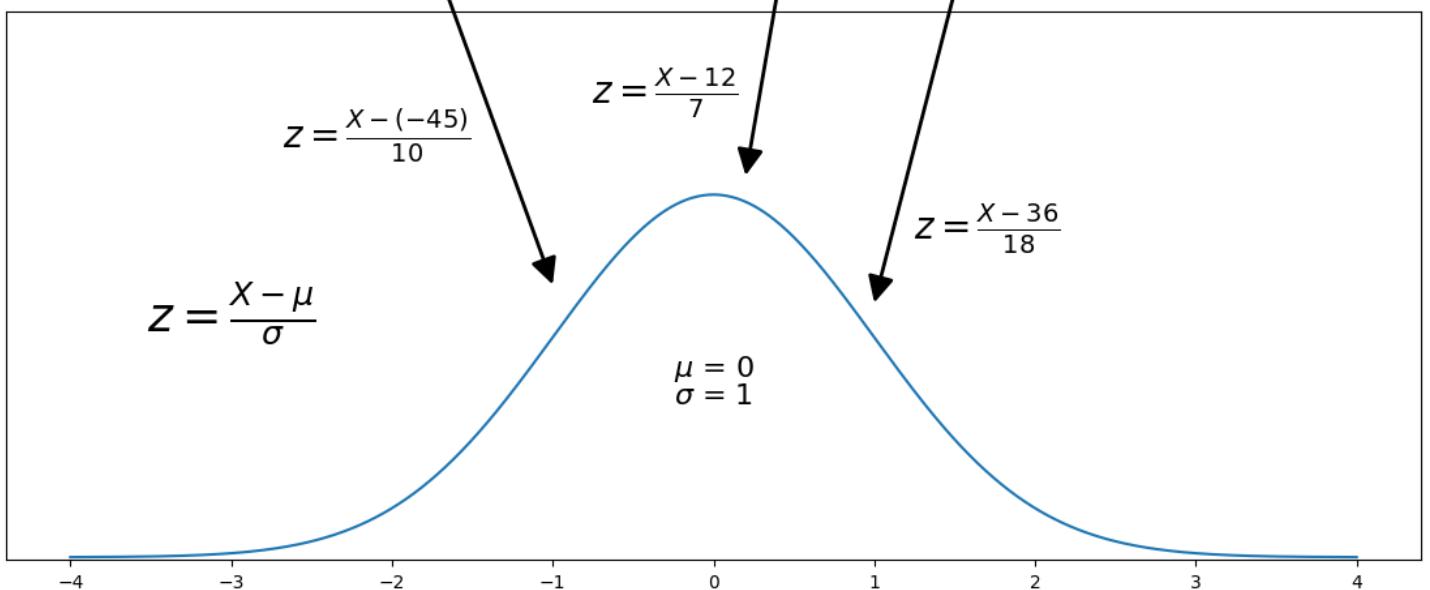
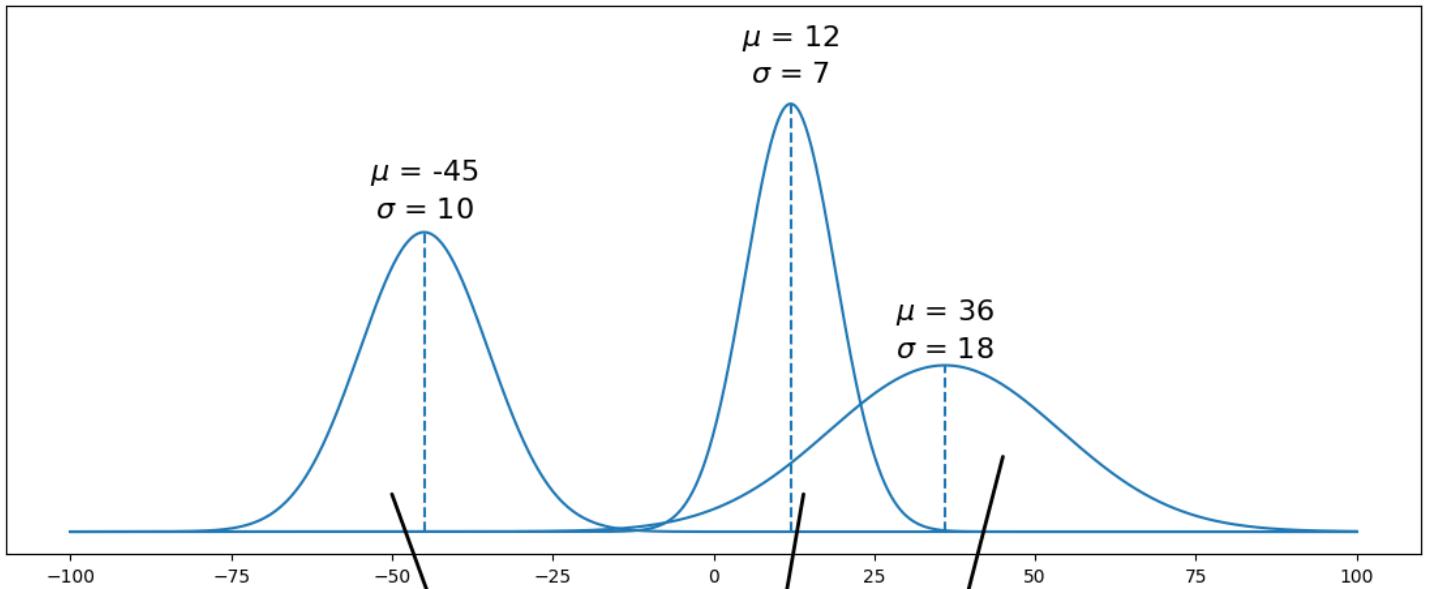
Standardisierung einer normalverteilten Variable

Bevor wir das Konzept der Standardnormalverteilung auf einen realen Datensatz anwenden können, müssen wir das Konzept der **Standardisierung einer Normalverteilung** diskutieren. Wir wissen, dass eine Normalverteilung durch zwei Parameter parametrisiert ist, ihren Mittelwert $\mu \in \mathbb{R} > 0$ und ihre Standardabweichung $\sigma \in \mathbb{R} > 0$, $X \sim N(\mu, \sigma)$. Der tatsächliche Wert dieser Parameter hängt von der Population und den zur Beschreibung ihrer Merkmale verwendeten Metriken ab. Um ein bestimmtes μ und σ , das sich auf eine bestimmte Zufallsvariable X bezieht, in $\mu = 0$ und $\sigma = 1$ umzuwandeln, müssen wir den x -Wert in einen z -Wert umwandeln, indem wir die folgende Gleichung anwenden.

$$z = \frac{x - \mu}{\sigma}$$

Als Ergebnis erhalten wir eine Standardnormalverteilung für eine bestimmte Normalverteilung. Dieses Verfahren ist unerlässlich, wenn Sie die z -Scores oder eine auf einen z -Score bezogene Wahrscheinlichkeit ($P(z)$) bestimmen müssen indem man sie in einer Tabelle nachschlägt. Wir werden später sehen, dass Python ein so mächtiges Werkzeug ist, dass der Schritt der Standardisierung überflüssig ist.

► Show code cell source



Die Standard-Normalverteilung: Ein Beispiel in Python

Vorbereitung der Daten

Jetzt sind wir bereit, einige Übungen zu machen. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Zuerst laden wir den Datensatz und geben ihm einen passenden Namen.

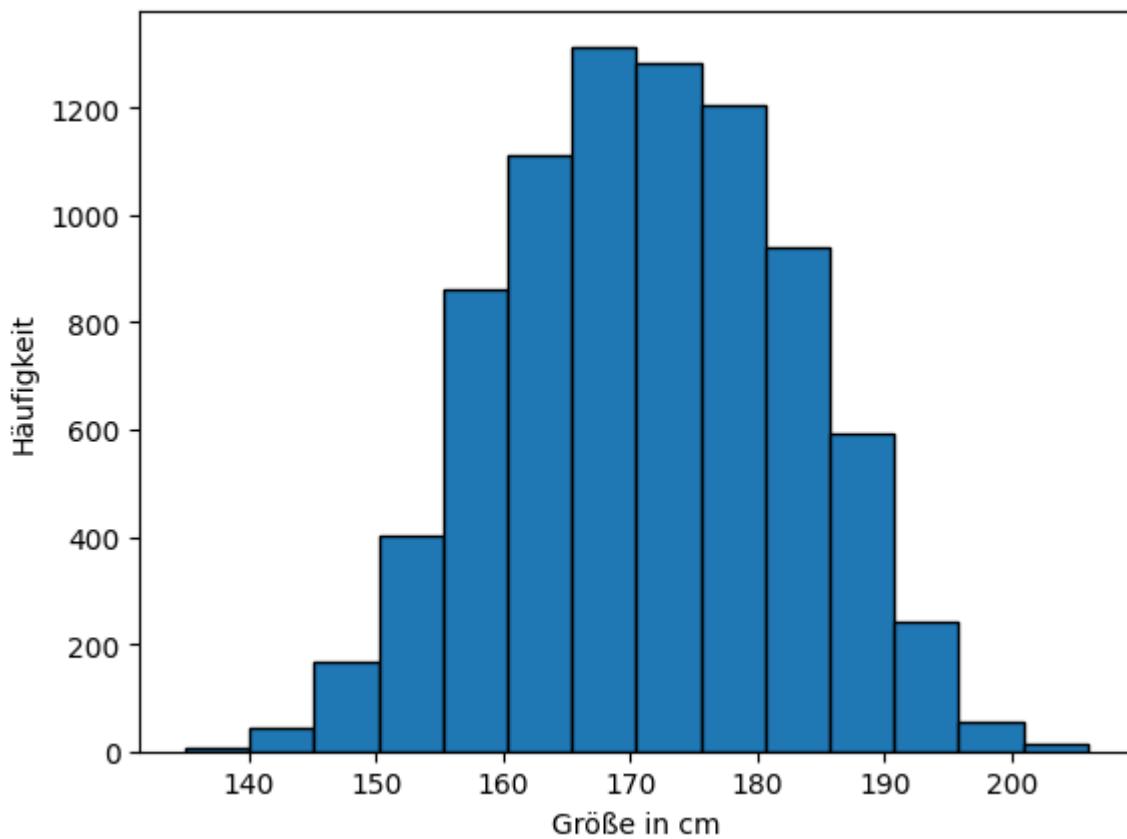
```
# Lese Datei students.csv als Dataframe ein
students = pd.read_csv("../data/students.csv")
# Lese Spalte 'height' ein
height = students["height"]
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id*, *Name*, *Geschlecht*, *Alter*, *Größe*, *Gewicht*, *Religion*, *nc_score*, *Semester*, *Hauptfach*, *Nebenfach*, *score1*, *score2*, *online_tutorial*, *graduated*, *salary*. In diesem Abschnitt verwenden wir die Variable `height`, um das bisher Besprochene zu üben.

Zunächst wollen wir sicherstellen, dass wir es mit normalverteilten Daten zu tun haben. Wenn eine Variable normalverteilt ist, sollte ein Histogramm der Beobachtungen bei einer großen Stichprobe in etwa die Form einer Glocke haben.

```
# Plotte die Werte als Histogramm
fig, ax = plt.subplots()
ax.hist(height, bins=14, edgecolor="k")
# Erzeuge Labels
ax.set_ylabel("Häufigkeit")
ax.set_xlabel("Größe in cm")
```

Text(0.5, 0, 'Größe in cm')



Aus dem Diagramm kann man schließen, dass die Variable `height` normalverteilt ist. Allerdings ist es vor allem bei kleinen Stichproben oft schwierig, eine klare Form in einem Histogramm festzustellen, insbesondere, ob sie glockenförmig ist. Daher ist eine empfindlichere grafische Technik zur Beurteilung der Normalität erforderlich. **Normal-Quantil-Plot** bieten eine solche Technik. Die Idee hinter einem Normal-Quantil-Plot oder kurz Q-Q Plot ist einfach: Man vergleicht die beobachteten Werte der Variablen mit den Beobachtungen, die für eine normalverteilte Variable erwartet werden. Genauer gesagt ist ein Q-Q Plot eine Darstellung der beobachteten Werte der Variablen im Vergleich zu den Werten, die für eine Variable mit der Standardnormalverteilung erwartet werden. Wenn die Variable normalverteilt ist, sollte der Q-Q Plot in etwa linear sein (d. h. in etwa auf einer Geraden liegen) (s.88).

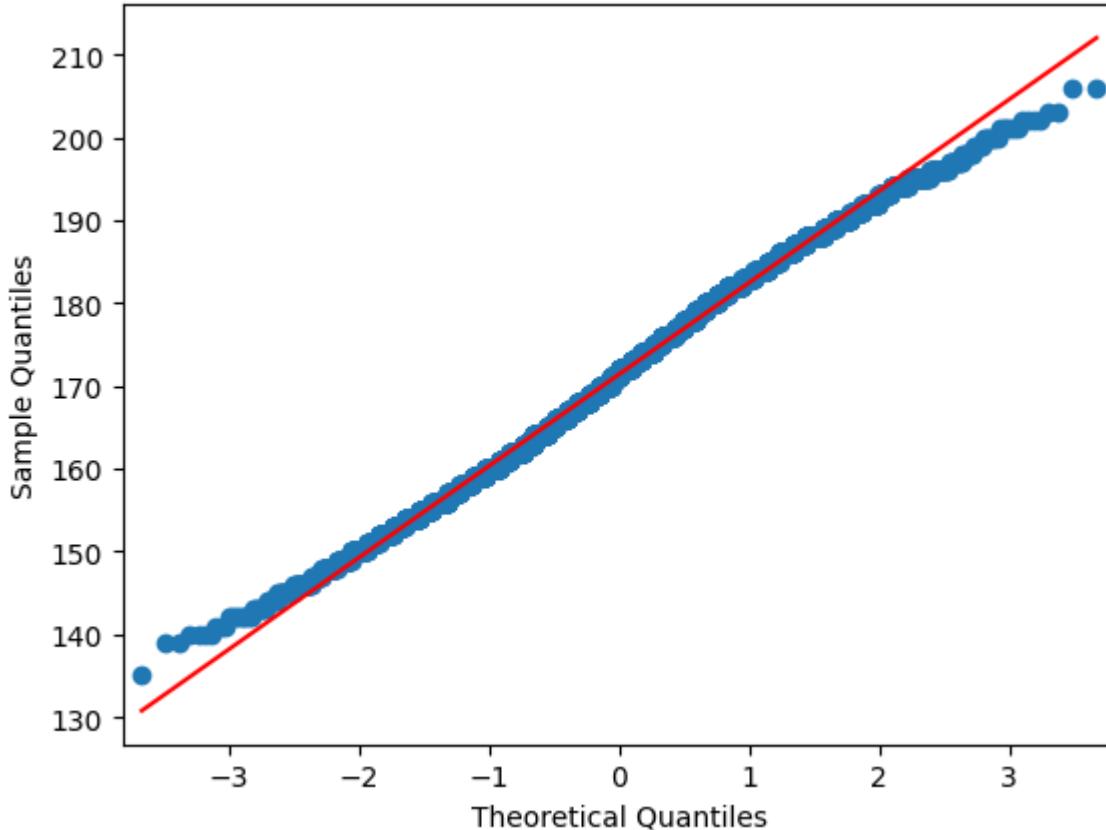
Bei der Verwendung eines Normal-Quantil-Plots zur Beurteilung der Normalität einer Variablen sind zwei Dinge zu beachten:

1. Die Entscheidung, ob eine normale Wahrscheinlichkeitsverteilung annähernd linear ist, ist eine subjektive Entscheidung, und

2. dass wir nur eine begrenzte Anzahl von Beobachtungen dieser bestimmten Variablen verwenden, um ein Urteil über alle möglichen Beobachtungen der Variablen zu fällen.

In Python können wir die Funktion `qqplot()` verwenden, um Normalwahrscheinlichkeitsplots zu erstellen, die auch als [Q-Q-Plots](#) bezeichnet werden.

```
# Erzeuge Q-Q Plot
_ = smi.qqplot(height, line="r")
```



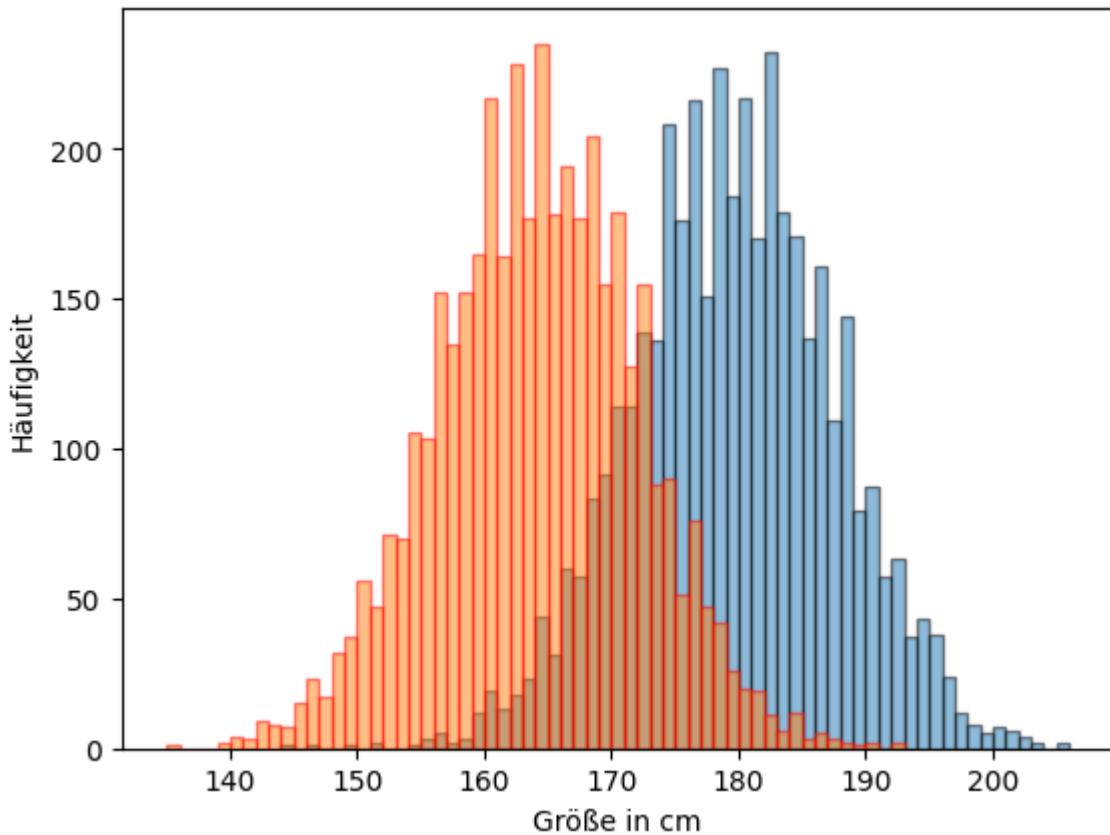
Bei der Betrachtung des Diagramms sehen wir, dass die Quantile der Stichprobe im Vergleich zu den theoretischen Quantilen am unteren und oberen Ende etwas abweichen. Dieser Tatsache muss etwas mehr Aufmerksamkeit geschenkt werden! Was könnte der Grund für die Abweichung am oberen und unteren Ende der Verteilung sein? Irgendeine Vermutung?

Was ist mit dem Geschlecht? Ehrlich gesagt scheint es natürlich zu sein, dass die durchschnittliche Körpergröße von Männern und Frauen unterschiedlich ist. Stellen wir ein Histogramm der Körpergröße von Männern und Frauen auf.

```
male_height = students.loc[students["gender"] == "Male", "height"]
female_height = students.loc[students["gender"] == "Female", "height"]
```

```
# Plotte die Werte als Histogramm
fig, ax = plt.subplots()
# Bestimme Anzahl Bins
bins_male = male_height.max() - male_height.min()
ax.hist(male_height, bins_male, edgecolor="k", alpha=0.5)
# Bestimme Anzahl Bins
bins_female = female_height.max() - female_height.min()
ax.hist(female_height, bins_female, edgecolor="r", alpha=0.5)
# Erzeuge Labels
ax.set_ylabel("Häufigkeit")
ax.set_xlabel("Größe in cm")
```

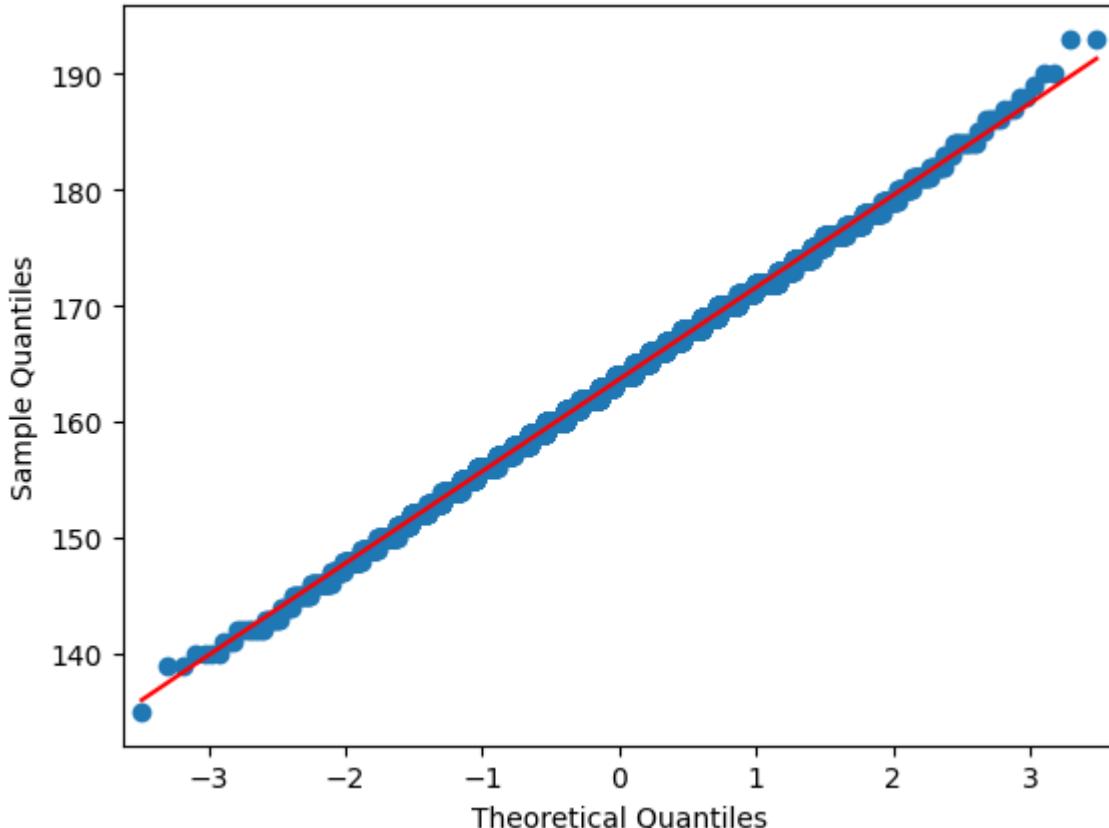
```
Text(0.5, 0, 'Größe in cm')
```



Das ist es! Offensichtlich haben die beiden Gruppen unterschiedliche Mittelwerte, so dass die Zusammenfassung zu einer Gruppe dazu führt, dass die linken und rechten Ausläufer der sich ergebenden Verteilung weiter reichen, als bei einer normalverteilten Variablen zu erwarten wäre. Um fortzufahren, betrachten wir also nur die Körpergröße der Studentinnen. Der Klarheit halber zeichnen

wir noch einmal den Normalwahrscheinlichkeitssplit der Größenvariablen, um sicherzustellen, dass unsere Zielvariablen normalverteilt sind.

```
# Erzeuge Q-Q Plot
_ = smi.qqplot(female_height, line="r")
```



Bevor wir mit den eigentlichen Übungen beginnen, berechnen wir zunächst den Mittelwert \bar{x} und die Standardabweichung s der Zielvariablen. Außerdem standardisieren wir die Variable, um eine Standardnormalverteilung mit $\bar{x} = 0$ und $s = 1$ zu erhalten, und weisen ihr einen geeigneten Variablenamen zu.

```
# Heights
height_mean = female_height.mean()
height_mean
```

```
np.float64(163.65328467153284)
```

```
height_sd = female_height.std()  
height_sd
```

```
np.float64(7.919725792052593)
```

```
height_z = (female_height - height_mean) / height_sd
```

Die Variable `height` hat einen Mittelwert von 163,7 cm und eine Standardabweichung von 7,9 cm.

Suche nach dem Bereich links von einem angegebenen z -Scores oder x -Wertes

Frage 1

Wie hoch ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Studentin aus dem `students` Datensatz eine Körpergröße von 168 cm oder weniger hat? Wir suchen also nach $P(x \leq 168)$.

Zunächst berechnen wir die Wahrscheinlichkeit für die standardisierte Variable. Dazu müssen wir den Wert, der uns interessiert (168 cm), in einen z -Score umwandeln.

$$z = \frac{x - \mu}{\sigma} = \frac{168 - 163,7}{7,9} = 0,55$$

```
height_z2 = (168 - height_mean) / height_sd  
height_z2
```

```
np.float64(0.5488466952768823)
```

Dann müssen wir die Fläche unter der Kurve links neben dem erhaltenen z -Wert berechnen. Zur Erinnerung: Die Fläche unter der Kurve einer normalverteilten Variablen kann mit Hilfe der Funktion `norm.cdf()` berechnet werden. Die `norm.cdf()`-Funktion wird als `norm.cdf(q, loc = 0, scale = 1)` geschrieben. Für dieses spezielle Beispiel können wir alle Standardargumente akzeptieren.

```
norm.cdf(height_z2)
```

```
np.float64(0.7084446690628331)
```

Genial, wir haben ein Ergebnis: $P(z \leq 0,55) \approx 0,71$

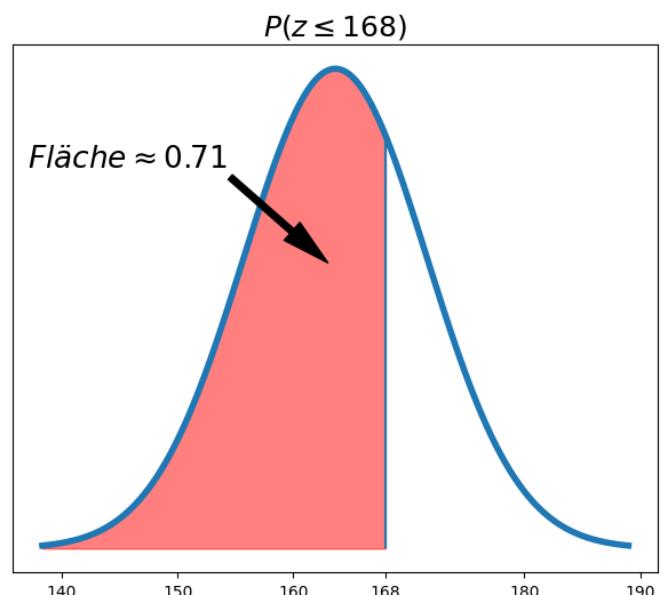
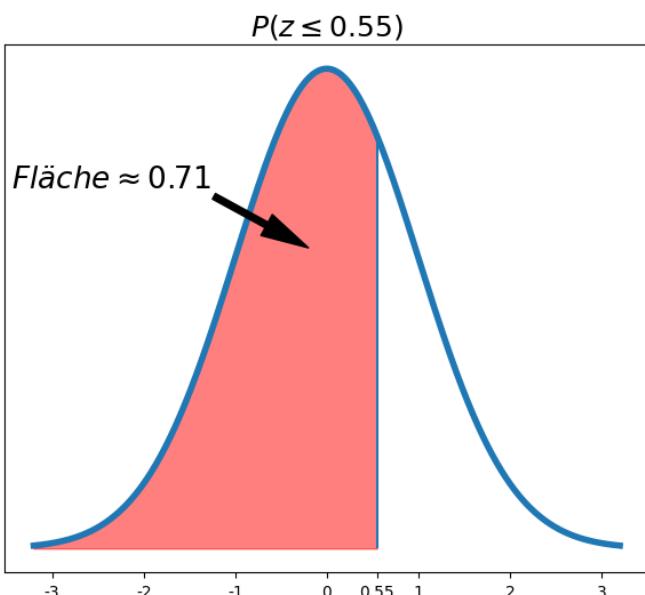
Nun führen wir die gleiche Berechnung durch, überspringen aber diesmal den Schritt der Standardisierung. Dank der Leistungsfähigkeit von Python müssen wir uns nicht auf Tabellen verlassen, sondern können den Stichprobenmittelwert \bar{x} und die Standardabweichung der Stichprobe, s , in die Funktion `stats.norm.cdf` eingeben.

```
x = 168
norm.cdf(x, loc=height_mean, scale=height_sd)
```

```
np.float64(0.7084446690628331)
```

Perfekt! Die Zahlen stimmen überein: $P(x \leq 168) \approx 0,71$. Um sicherzustellen, dass wir verstehen, was vor sich geht, werden unten sowohl die Fläche unter der Kurve für die standardisierte Variable in z -Werten (linkes Feld) als auch die Fläche für die nicht standardisierte Variable in cm (rechtes Feld) dargestellt.

► Show code cell source



Ermitteln der Fläche rechts von einem bestimmten x -Wert

Frage 2

Wie hoch ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Studentin aus dem `students` Datensatz eine Körpergröße von 185 cm oder mehr hat? Wir suchen also nach $P(x \geq 175)$. Um die Fläche unter der Kurve rechts vom interessierenden Wert zu erhalten, müssen wir in die Funktion `stats.norm.sf()` oder `1-stats.norm.cdf` verwenden.

```
x = 175 # height in cm
norm.sf(x, loc=height_mean, scale=height_sd)
```

```
np.float64(0.07596955321865)
```

```
x = 175 # height in cm
norm.cdf(x, loc=height_mean, scale=height_sd)
```

```
np.float64(0.92403044678135)
```

Antwort: $P(x \geq 175) \approx 0,08$

Ermitteln der Fläche zwischen zwei angegebenen x -Werten

Um die Fläche unter einer Kurve für ein Intervall $[a, b]$ zu bestimmen, verwenden wir die Gleichung

$$P(a \leq x \leq b) = \int_a^b f(x)dx = P(x \leq b) - P(x \leq a).$$

Frage 3

Wie hoch ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Studentin aus dem `students` Datensatz eine Körpergröße zwischen 155 und 165 cm hat, $P(155 \leq x \leq 165)$?

```

x_lower = 155 # height in cm
x_upper = 165 # height in cm

cdf_upper = norm.cdf(x_upper, loc=height_mean, scale=height_sd)
cdf_lower = norm.cdf(x_lower, loc=height_mean, scale=height_sd)
cdf_upper - cdf_lower

```

```
np.float64(0.4302335028797312)
```

Antwort: $P(155 \leq x \leq 165) \approx 0,43$

Frage 4

Wie hoch ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Studentin aus dem Studentendatensatz eine Körpergröße zwischen 170 und 180 cm hat, $P(170 \leq x \leq 180)$?

```

x_lower = 170 # height in cm
x_upper = 180 # height in cm

cdf_upper = norm.cdf(x_upper, loc=height_mean, scale=height_sd)
cdf_lower = norm.cdf(x_lower, loc=height_mean, scale=height_sd)
cdf_upper - cdf_lower

```

```
np.float64(0.19194918877717126)
```

Antwort: $P(170 \leq x \leq 180) \approx 0,19$

z_α finden

Frage 5

Wir möchten wissen, welche Körpergröße der Studentinnen in unserem `students` Datensatz mit einer Wahrscheinlichkeit von 0,60 übereinstimmt. Oder anders ausgedrückt: Wenn wir eine Anzahl von n Studenten aus dem `students` Datensatz zufällig auswählen, welche Größe teilt die Stichprobe in 60% der n Studierenden, die kleiner sind, und 40% der n Studentinnen, die größer als diese bestimmte Größe sind. Wir suchen also nach $P(X <?) = 0,60$.

Um $P(X < ?) = 0,60$ zu lösen, werden wir zwei Ansätze wählen. Der erste Ansatz verwendet den z -Score, und der zweite verwendet Python, um den Standardisierungsschritt überflüssig zu machen. . .

Für beide Ansätze verwenden wir die `norm.ppf()`-Funktion, die wie folgt geschrieben wird:

`norm.ppf(p, loc = 0, scale = 1)`.

Für den ersten Ansatz müssen wir die Gleichung für die Standardisierung von oben umstellen und sie für x lösen

$$z = \frac{x - \mu}{\sigma} \implies x = z\sigma + \mu$$

Für die Berechnung von x benötigen wir den Mittelwert (`height_mean`) und die Standardabweichung (`height_sd`) für die Variable `height`, die 163,7 cm bzw. 7,9 cm beträgt. Außerdem müssen wir einen z -Score für die gegebene Wahrscheinlichkeit von 0,60 erhalten. Wir können diesen z -Score in einer Tabelle nachschlagen oder die `norm.ppf()`-Funktion in Python anwenden. Wir wollen den z -Score ermitteln, bei dem der Bereich links von diesem z -Score 0,60 entspricht; erinnern Sie sich, dass wir nach $P(X < ?) = 0,60$ suchen.

```
z = norm.ppf(0.6, loc=0, scale=1)  
z
```

```
np.float64(0.2533471031357997)
```

Da wir nun z kennen, können wir in die Gleichung von oben einsetzen

$$x = z\sigma + \mu$$

$$= 0,25 \times 7,9 + 163,7$$

$$\approx 165,66$$

Perfekt, wir sind fertig: $P(X < 165,66) = 0,60$

Nun gehen wir den zweiten Ansatz durch, bei dem wir den Schritt der z -Berechnung überspringen.

Alles, was wir tun müssen, ist, die `norm.ppf()`-Funktion mit dem Mittelwert und der Standardabweichung unserer Variablen `height` zu füttern.

```
x = norm.ppf(0.6, loc=height_mean, scale=height_sd)  
x
```

```
np.float64(165.65972425857925)
```

Keine Überraschung, die Zahlen stimmen überein: $P(X < 165,66) = 0,60$.

Die kontinuierliche gleichmäßige Verteilung

```
import matplotlib.pyplot as plt  
import numpy as np  
from scipy.stats import uniform
```

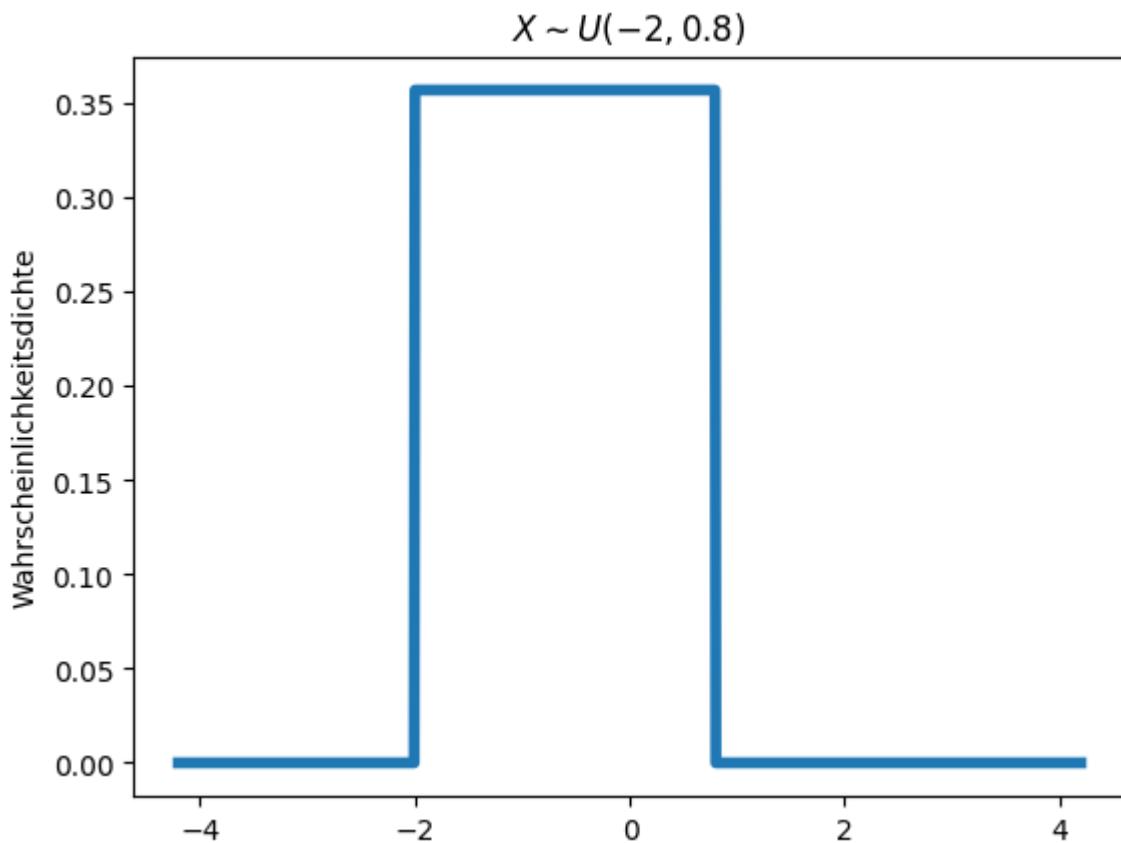
Die [Gleichverteilung](#) ist die einfachste Wahrscheinlichkeitsverteilung, aber sie spielt eine wichtige Rolle in der Statistik, da sie bei der Modellierung von Zufallsvariablen sehr nützlich ist. Die Gleichverteilung ist eine kontinuierliche Wahrscheinlichkeitsverteilung und befasst sich mit Ereignissen, deren Auftreten gleich wahrscheinlich ist. Die kontinuierliche Zufallsvariable X gilt als gleichmäßig verteilt oder hat eine rechteckige Verteilung auf dem Intervall $[a, b]$. Wir schreiben $X \sim U(a, b)$, wenn seine Wahrscheinlichkeitsdichtefunktion gleich $f(x) = \frac{1}{b-a}$, $x \in [a, b]$ und ansonsten gleich 0 ist (s.331).

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{wenn } a \leq x \leq b \\ 0, & \text{wenn } x < a \text{ oder } x > b \end{cases}$$

Die folgende Abbildung zeigt eine kontinuierliche Gleichverteilung $X \sim U(-2, 0, 8)$, also eine Verteilung, bei der alle Werte von x innerhalb des Intervalls $[-2, 0, 8]$ gleich $\frac{1}{b-a} (= \frac{1}{0,8-(-2)} \approx 0,36)$ sind, während alle anderen Werte von x gleich 0 sind.

```
x = np.linspace(-4.2, 4.2, num=1000)  
fig, ax = plt.subplots()  
ax.plot(x, uniform.pdf(x, -2, 2 + 0.8), linewidth=4)  
ax.set_title(r"$X \sim U(-2, 0.8)$")  
ax.set_ylabel("Wahrscheinlichkeitsdichte")
```

```
Text(0, 0.5, 'Wahrscheinlichkeitsdichte')
```



Der Mittelwert und der Median sind gegeben durch

$$\mu = \frac{a + b}{2}.$$

Die kumulative Dichtefunktion ist unten dargestellt und ergibt sich aus der Gleichung

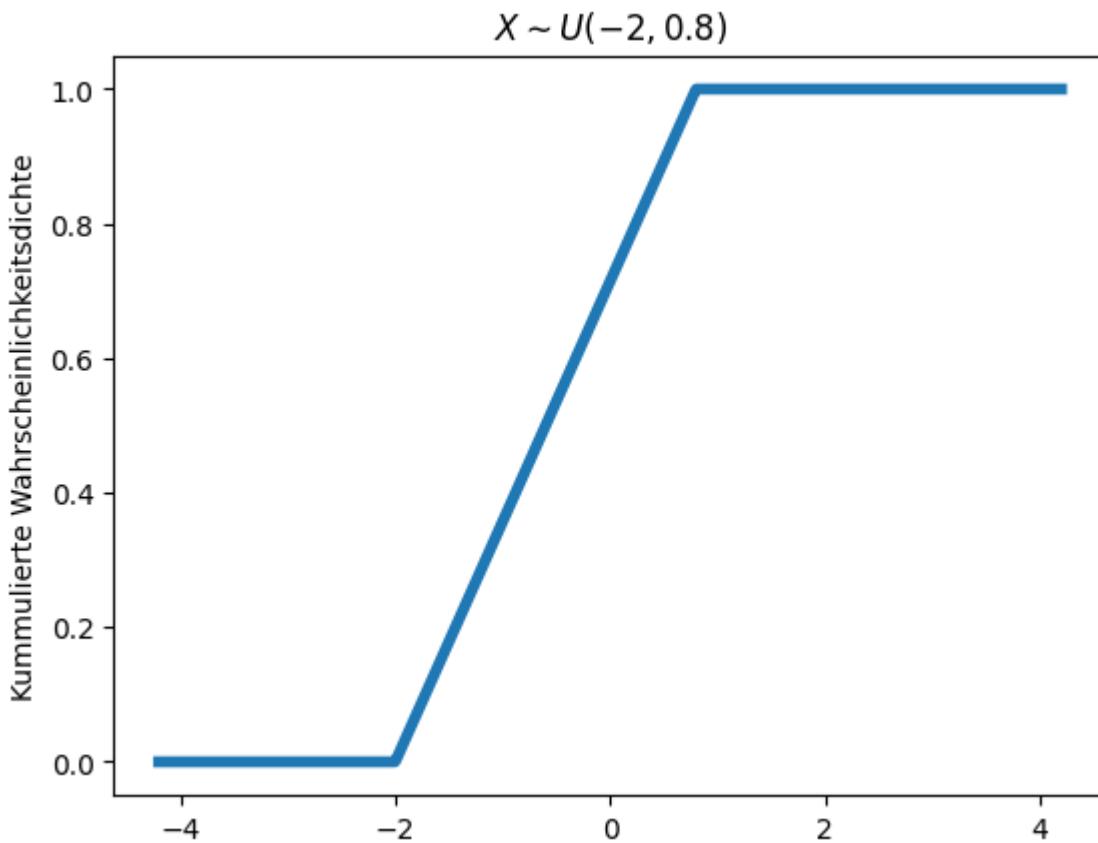
$$F(x) = \begin{cases} 0, & \text{for } x < a \\ \frac{x-a}{b-a}, & \text{for } x \in [a, b) \\ 1, & \text{for } x \geq b \end{cases}$$

```

x = np.linspace(-4.2, 4.2, num=1000)
fig, ax = plt.subplots()
ax.plot(x, uniform.cdf(x, -2, 2 + 0.8), linewidth=4)
ax.set_title(r"$X \sim U(-2, 0.8)$")
ax.set_ylabel("Kummulierte Wahrscheinlichkeitsdichte")

```

Text(0, 0.5, 'Kummulierte Wahrscheinlichkeitsdichte')



Die kontinuierliche gleichmäßige Verteilung in Python

Python ermöglicht den Zugriff auf die Gleichverteilung mit den Funktionen `uniform.pmf()`, `uniform.cdf()`, `uniform.ppf()` und `uniform.rvs()`. Wenden Sie die Funktion `dir()` auf diese Funktionen an, um weitere Informationen zu erhalten.

Die Funktion `uniform.rvs()` erzeugt Zufallsabweichungen der Gleichverteilung und wird als `uniform.rvs(loc, loc+scale, size)` geschrieben. Wir können auf einfache Weise n Zufallsstichproben innerhalb eines beliebigen Intervalls erzeugen indem wir die Zahlenwerte für minimalen (a) und maximalen Wert (b) in `random.uniform(a, b, size)` einsetzen.

```
u_rvs = np.random.uniform(-1, 1, size=40)
u_rvs
```

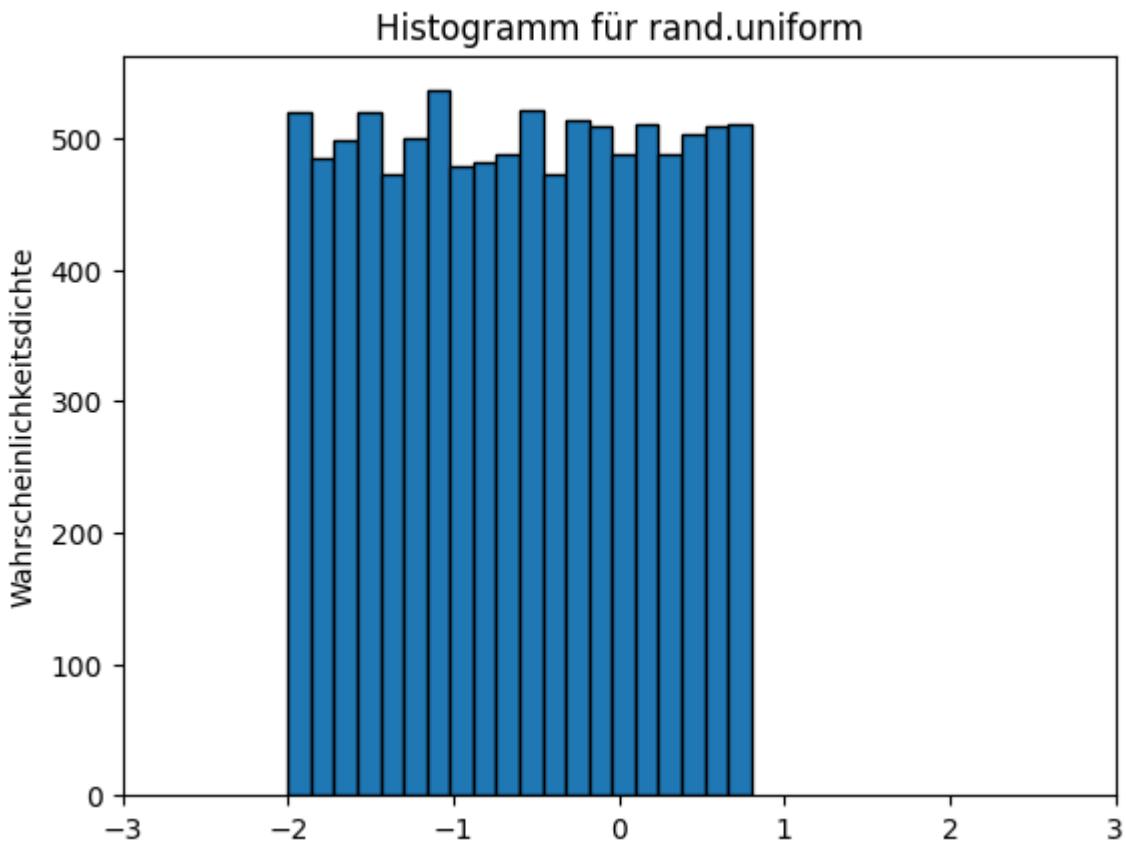
```
array([-0.0205954 ,  0.53221541,  0.51272364,  0.5140732 , -0.16786397,
       -0.73199718,  0.66645117,  0.63607862,  0.0391667 , -0.303326 ,
       -0.20869598, -0.89068677, -0.5118229 , -0.90877369,  0.73735285,
       -0.74602666, -0.30200711,  0.62887019, -0.02298375,  0.33260333,
       0.77519481, -0.41660946,  0.18846448,  0.08883272,  0.06678826,
       -0.65314687,  0.89611103, -0.36292166, -0.48991632, -0.87296698,
       0.58557429, -0.37248086, -0.3111459 ,  0.12273519,  0.68698454,
       -0.52506539, -0.93348557,  0.05244385,  0.70280723, -0.4530584 ])
```

Wir können die Dichtefunktion für $X \sim U(-2, 0, 8)$ mit Hilfe der Funktion `uniform.rvs()` approximieren und die Ergebnisse als Histogramm darstellen.

```
# Erzeuge gleichverteilte Werte
u_rvs = np.random.uniform(-2, 0.8, size=10000)

# Plotte Histogramm
fig, ax = plt.subplots()
ax.set_xlim(-3, 3)
ax.set_title("Histogramm für rand.uniform")
ax.set_ylabel("Wahrscheinlichkeitsdichte")
ax.hist(u_rvs, bins=20, edgecolor="k")
```

```
(array([520., 484., 498., 520., 473., 500., 536., 478., 481., 487., 521.,
       472., 513., 509., 487., 511., 487., 503., 509., 511.]),
array([-1.99985808, -1.85986735, -1.71987662, -1.57988589, -1.43989517,
       -1.29990444, -1.15991371, -1.01992298, -0.87993225, -0.73994153,
       -0.5999508 , -0.45996007, -0.31996934, -0.17997861, -0.03998788,
       0.10000284,  0.23999357,  0.3799843 ,  0.51997503,  0.65996576,
       0.79995649]),
<BarContainer object of 20 artists>)
```



Außerdem stellen wir sowohl das Dichtehistogramm von oben als auch die gleichmäßige Wahrscheinlichkeitsverteilung für das Intervall $[-2, 8]$ dar, indem wir die Funktion `uniform.pdf()` anwenden.

```

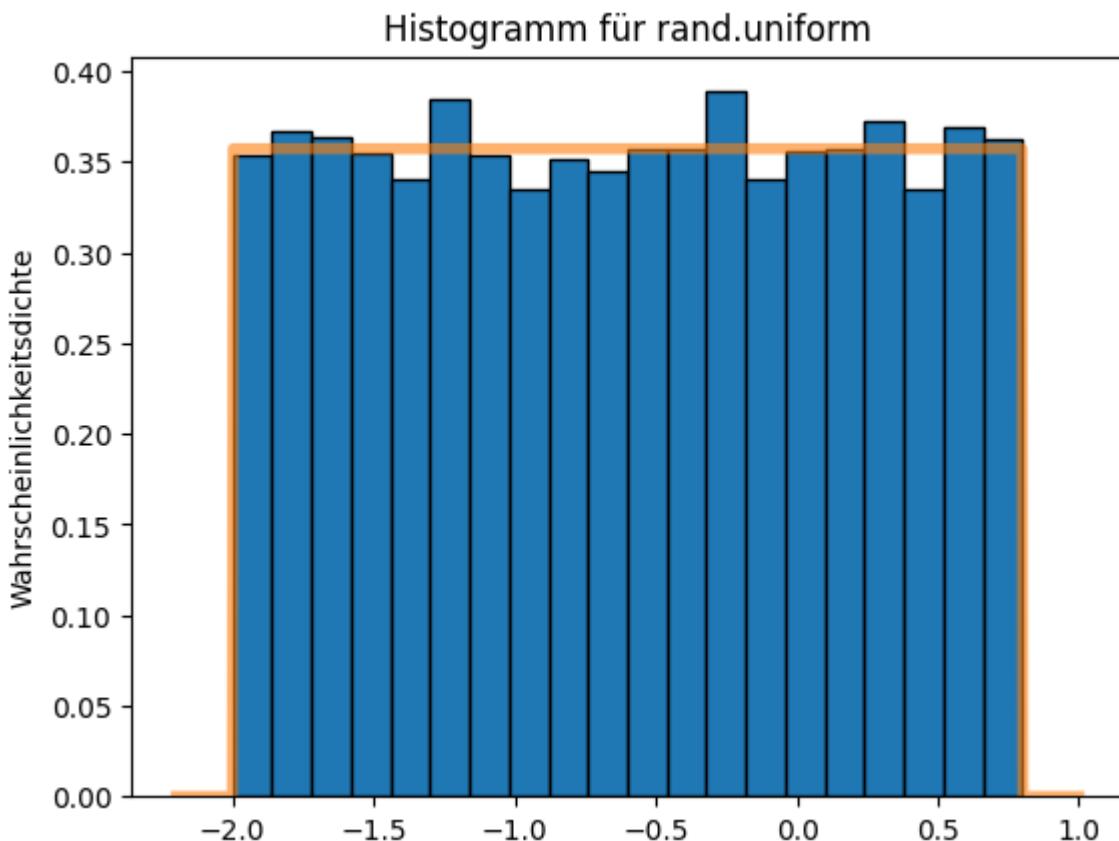
# Erzeuge x-werte
x = np.linspace(-2.2, 1, num=1000)

# Erzeuge gleichverteilte Werte
u_rvs = uniform.rvs(loc=-2, scale=2 + 0.8, size=10000)

# Plotte Histogramm und uniforme PDF
fig, ax = plt.subplots()
ax.set_title("Histogramm für rand.uniform")
ax.set_ylabel("Wahrscheinlichkeitsdichte")
ax.hist(u_rvs, bins=20, edgecolor="k", density=True)
ax.plot(x, uniform.pdf(x, -2, 2 + max(u_rvs)), linewidth=4, alpha=0.6)

```

[<matplotlib.lines.Line2D at 0x7ab4b5035c00>]



```

unif_mean = (-2 + 0.6) / 2
unif_mean

```

-0.7

Die Abbildung zeigt, dass unsere 10.000 Stichproben, die nach dem Zufallsprinzip aus einer Gleichverteilung gezogen wurden (Histogramm), sich der Gleichverteilung $X \sim U(-2, 0, 8)$ (Liniendiagramm).

Außerdem können wir die Funktion `uniform.cdf()` verwenden, um die Fläche unter der Kurve für einen bestimmten Schwellenwert zu berechnen, oder wir können die Funktion `uniform.ppf()` verwenden, um einen Schwellenwert für eine bestimmte Wahrscheinlichkeit zurückzugeben.

Übung

Betrachten wir die gleichmäßige Wahrscheinlichkeitsverteilung, die durch $X \sim U(-3, 5, 5)$ gegeben ist.

Frage 1

Wie lautet der Mittelwert μ für die gegebene Gleichverteilung.

```
unif_mean = (-3 + 5.5) / 2
unif_mean
```

1.25

Der Mittelwert μ für die durch $X \sim U(-3, 5, 5)$ gegebene gleichmäßige Wahrscheinlichkeitsverteilung beträgt 1,25.

Frage 2

Welcher Wert von x entspricht dem Wert, der die gegebene Gleichverteilung in zwei gleiche Teile teilt, oder anders ausgedrückt: $P(X < ?) = 0,5$.

```
p_50 = uniform.ppf(0.5, -3, 8.5)
p_50
```

`np.float64(1.25)`

Das ist überhaupt keine Überraschung. Der Wert von x , der die Gleichverteilung in zwei gleiche Teile teilt, beträgt 1,25 und ist somit gleich μ .

Frage 3

Angenommen, die obige Verteilung beschreibt ein physikalisches Phänomen. Wie groß ist die Wahrscheinlichkeit, dass bei einer Messung des physikalischen Prozesses, der das Phänomen bestimmt, ein Wert ≥ 4 gemessen wird, oder anders ausgedrückt: $P(X \geq 4)$. Aufgrund des Charakters einer Gleichverteilung ist die Messung eines beliebigen Wertes innerhalb des Intervalls $[-3, 5, 5]$ gleich wahrscheinlich ist.

Wir werden diese Frage auf zwei Arten lösen, numerisch und analytisch. Um die Frage numerisch zu lösen, müssen wir zunächst ein Experiment durchführen. Wir wiederholen unsere Messung eine große Anzahl von Malen und zählen dann, wie oft wir einen Wert ≥ 4 registriert haben. Dank der Leistungsfähigkeit von Python und dem integrierten Zufallszahlengenerator `uniform.rvs()` für gleichmäßig verteilte Daten) ist die Wiederholungsaufgabe sehr einfach, allerdings sollte man sich darüber im Klaren sein, dass in realen Anwendungen oft nur eine sehr begrenzte Anzahl von Messungen verfügbar ist.

```
# Erzeuge gleichverteilte Zufallsvariablen und Variable count
u_rvs = uniform.rvs(-3, 8.5, size=10000)

# Zähle Werte größer gleich 4
count = sum(u_rvs >= 4)

# Dividiere durch Gesamtanzahl der Werte
count = count / len(u_rvs)
count
```

```
np.float64(0.1773)
```

Die Ergebnisse zeigen, dass etwa 18% der Messungen Werte ≥ 4 ergeben.

Zweitens, um die Frage analytisch zu lösen, verwenden wir die kumulative Wahrscheinlichkeitsdichtefunktion, die in Python für gleichmäßige Verteilungen durch die Funktion `uniform.cdf()` implementiert ist. Wir interessieren uns also für die Fläche unter der Kurve bis zum Wert von $x = 4$.

```
1 - uniform.cdf(4, -3, 8.5)
```

```
np.float64(0.17647058823529416)
```

Der analytische Ansatz ergibt ein Ergebnis von $0,1764706$ oder anders ausgedrückt, mit einer Wahrscheinlichkeit von $17,65\%$ erhalten wir Werte ≥ 4 , also $P(X \geq 4) \approx 0,18$.

Es ist offensichtlich, dass beide Ansätze sehr ähnliche Ergebnisse liefern. Es ist jedoch zu beachten, dass das Ergebnis des numerischen Ansatzes eine Annäherung an das analytische Ergebnis darstellt. Bedenken Sie, dass die Qualität einer solchen Annäherung sehr stark von der Anzahl der Zufallsvariablen abhängt, aus denen die Stichprobe besteht, in unserem Fall von der Anzahl der Messungen.

Die Student t-Verteilung

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import t
```

Die Studentsche t -Verteilung ist nach [William Sealy Gosset](#) (1876 – 1937) benannt, der sie 1908 erstmals bestimmte. Gosset war einer der besten Oxford-Absolventen in Chemie und Mathematik seiner Generation. Im Jahr 1899 nahm er eine Stelle als Brauer bei [Arthur Guinness Son & Co, Ltd](#) in Dublin, Irland, an. Bei seiner Arbeit für die Guinness-Brauerei interessierte er sich für die Qualitätskontrolle anhand kleiner Proben in verschiedenen Stadien des Produktionsprozesses. Da Guinness seinen Angestellten die Veröffentlichung von Papieren untersagte, um die Weitergabe vertraulicher Informationen zu verhindern, hatte Gosset seine Arbeit unter dem Pseudonym "Student" veröffentlicht, und seine Identität war einige Zeit nach der Veröffentlichung seiner berühmtesten Errungenschaften nicht bekannt, so dass die Verteilung den Namen "Studentsche" oder " t -Distribution" erhielt, wodurch sein Name weniger bekannt wurde als seine wichtigen Ergebnisse in der Statistik ([] s.81).

Die t -Verteilungskurve ist, wie die Normalverteilungskurve, symmetrisch (glockenförmig) um den Mittelwert. Die t -Verteilungskurve ist jedoch flacher als die Standard-Normalverteilungskurve. Folglich hat die t -Verteilungskurve eine geringere Höhe und eine breitere Streuung als die Standardnormalverteilung.

Die t -Verteilung hat nur einen Parameter, die sogenannten **Freiheitsgrade** (df). Die Form einer bestimmten t -Verteilungskurve hängt von der Anzahl der Freiheitsgrade (df) ab. Die Anzahl der Freiheitsgrade für eine t -Verteilung ist gleich dem Stichprobenumfang minus eins, das heißt,

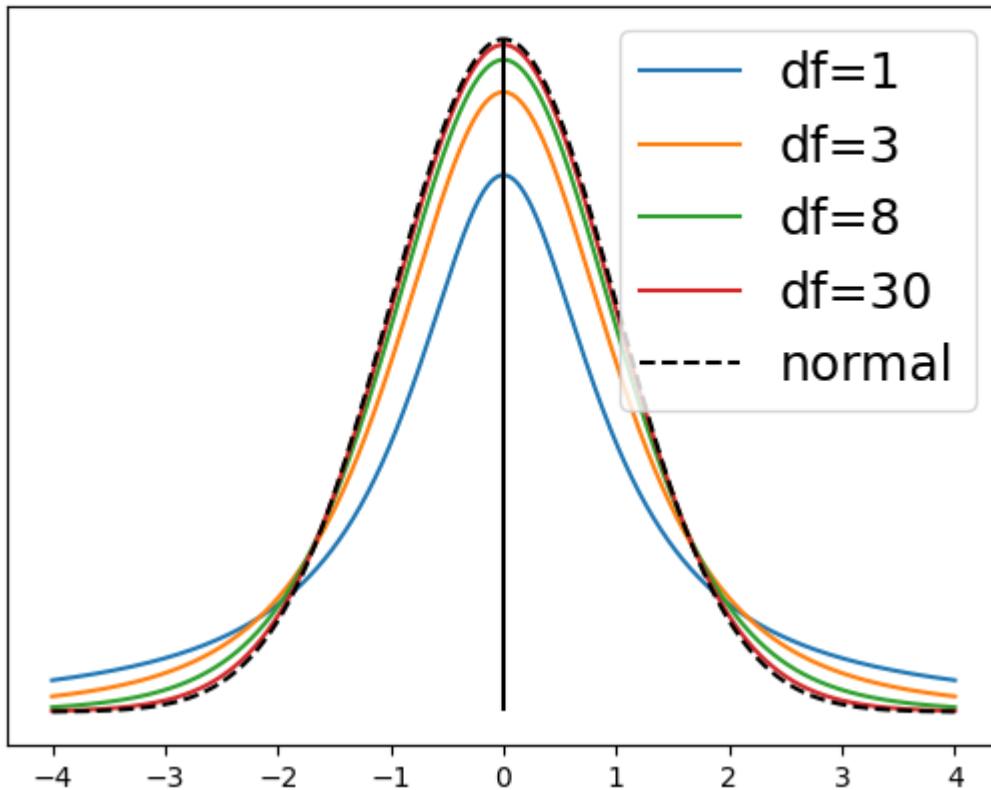
$$df = n - 1$$

Wenn der Stichprobenumfang n und damit df zunimmt, nähert sich die t -Verteilung der Standardnormalverteilung an. Die Einheiten einer t -Verteilung werden mit t bezeichnet. Der Mittelwert der t -Verteilung ist gleich 0, und ihre Standardabweichung beträgt $\sqrt{df/(df - 2)}$ (s.280,s.284).

▶ Show code cell source

```
<matplotlib.legend.Legend at 0x77cd8a47a470>
```

Vergleich der t-Wrscheinlichkeitsdichtefunktion
mit unterschiedlichen Freiheitsgraden (df) und
der Wahrscheinlichkeitsdichtefunktion der Normalverteilung



Grundlegende Eigenschaften von t-Kurven

- Die Gesamtfläche unter einer t -Kurve ist gleich 1.
- Eine t -Kurve erstreckt sich unendlich in beide Richtungen und nähert sich dabei der horizontalen Achse, berührt sie aber nie.
- Eine t -Kurve ist symmetrisch um 0.
- Mit zunehmender Anzahl von Freiheitsgraden ähneln t -Kurven immer mehr der Standard-Normalverteilung.

Die Studentsche-t-Verteilung in Python

Python ermöglicht den Zugriff auf die t -Verteilung mit den Funktionen `t.pdf()`, `t.cdf()`, `t.ppf()` und `t.rvs()`. Wenden Sie die Funktion `dir()` auf diese Funktionen an, um weitere Informationen zu erhalten.

Die Funktion `t.rvs()` erzeugt Zufallsabweichungen der t -Verteilung und wird als `t.rvs(df, loc, scale, size)` geschrieben. Wir können leicht eine Anzahl von n Zufallsstichproben erzeugen.

Erinnern Sie sich daran, dass die Anzahl der Freiheitsgrade für eine t -Verteilung gleich dem Stichprobenumfang minus eins ist, d.h.,

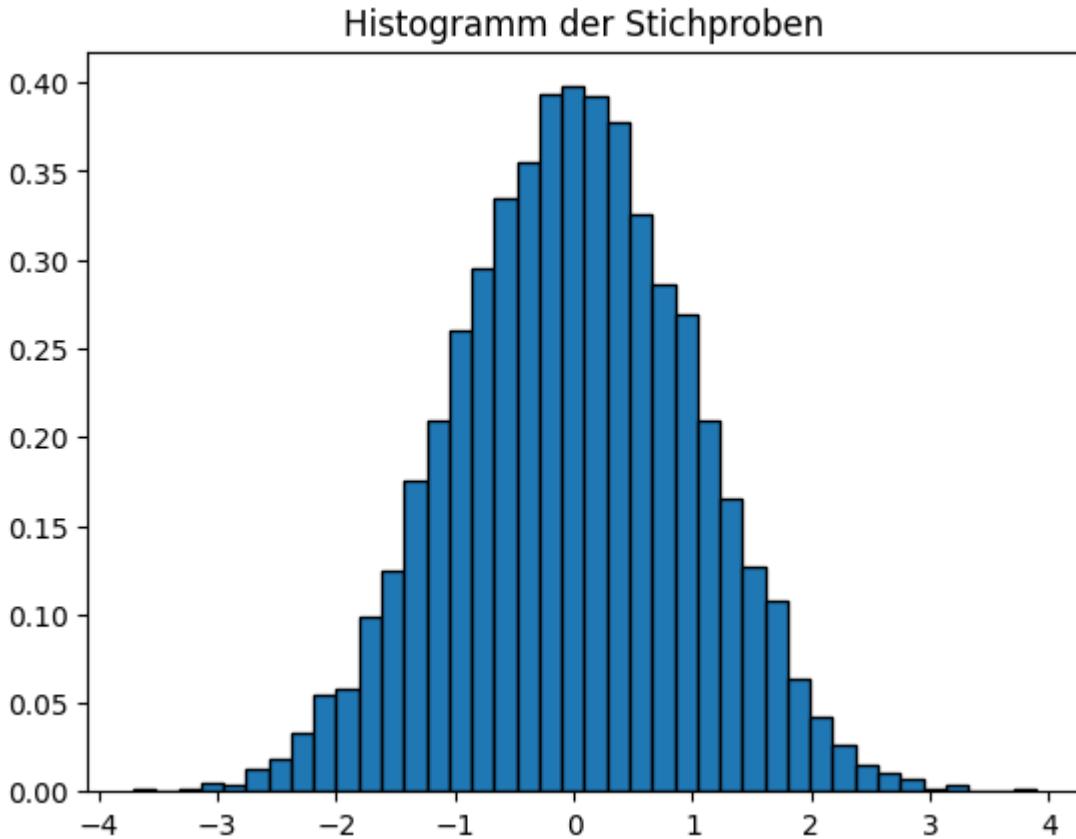
$$df = n - 1.$$

```
# Generiere Zufallswerte der t Verteilung mit df = 29 und Stichprobengrösse = 30
n = 30
t.rvs(df=n - 1, size=n)
```

```
array([-0.97604951, -0.74975019,  0.56615562, -0.32523933,  1.04330061,
       -0.55910938, -0.01034667, -1.73024539,  0.87060816,  1.45253269,
       0.69911955, -0.74722242, -0.22615058, -1.5157746 ,  0.43720184,
      -0.49392699,  0.64105893, -0.93552942,  1.35987327,  0.32839682,
      1.01838794, -1.01441186,  0.60564305, -2.22555967,  0.62273408,
      0.12606213,  0.10182999, -0.46277047, -1.19597291, -0.43818372])
```

Außerdem können wir eine sehr große Anzahl von Stichproben erzeugen und sie als Histogramm darstellen.

```
# Generiere Zufallswerte der t Verteilung mit df = 9999 und Stichprobengröße = 1000
n = 10000
y = t.rvs(df=n - 1, size=n)
fig, ax = plt.subplots()
ax.set_title("Histogramm der Stichproben")
_ = ax.hist(y, bins=40, density=True, edgecolor="k")
```

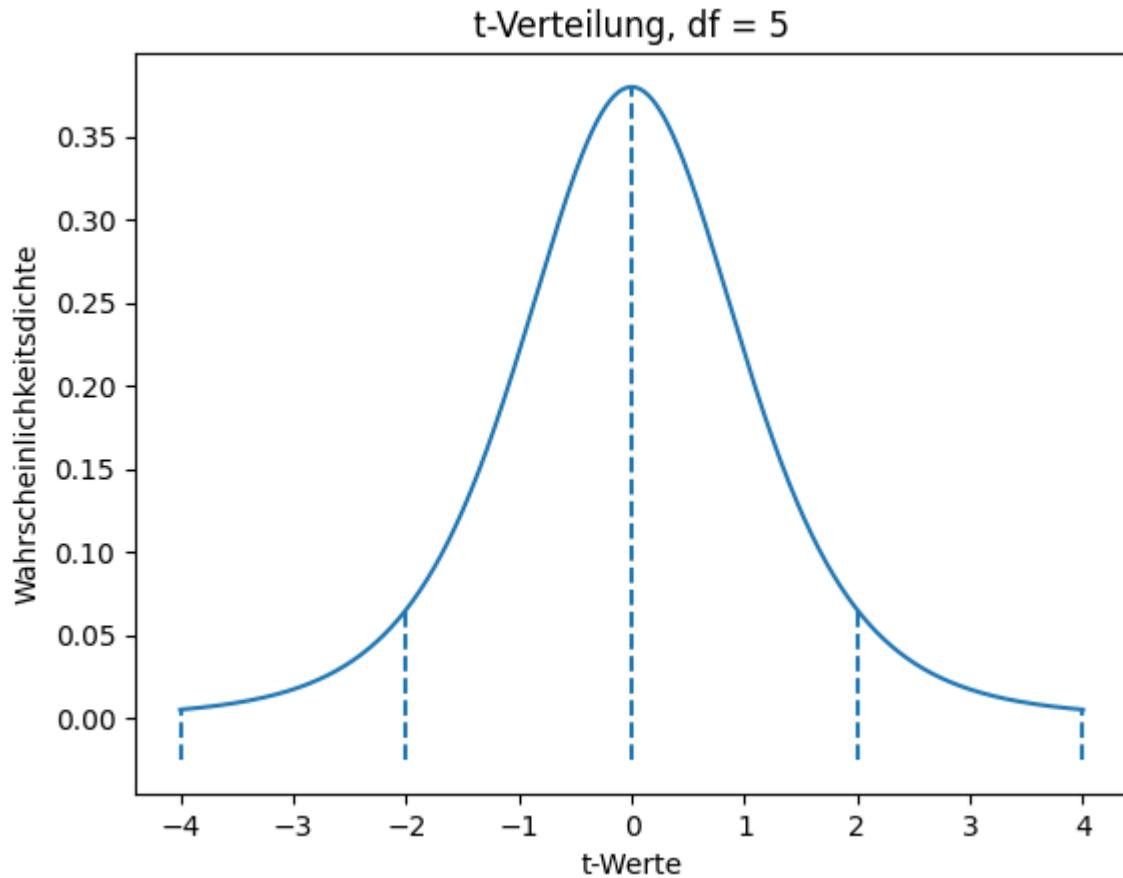


Mit der Funktion `t.pdf()` können wir die Wahrscheinlichkeitsdichtefunktion und damit den vertikalen Abstand zwischen der horizontalen Achse und der t -Kurve an jedem beliebigen Punkt berechnen. Zur Demonstration konstruieren wir eine t -Verteilung mit $df = 5$ und berechnen die Wahrscheinlichkeitsdichtefunktion bei $t = -4, -2, 0, 2, 4$.

```
x = [-4, -2, 0, 2, 4]
y_t = t.pdf(x, df=5)
y_t
```

```
array([0.00512373, 0.06509031, 0.37960669, 0.06509031, 0.00512373])
```

► Show code cell source

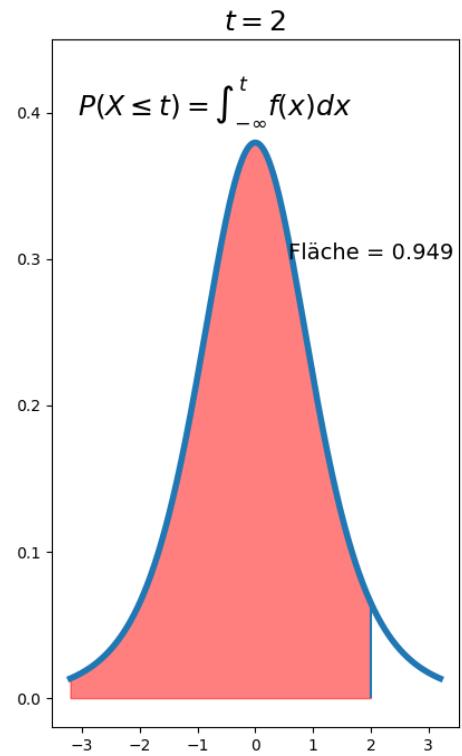
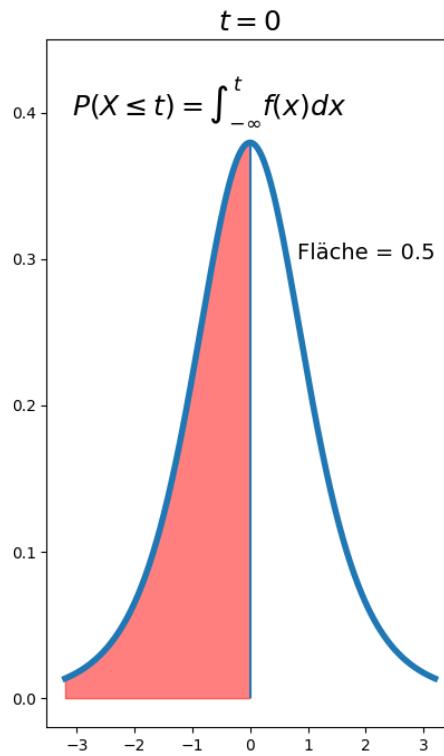
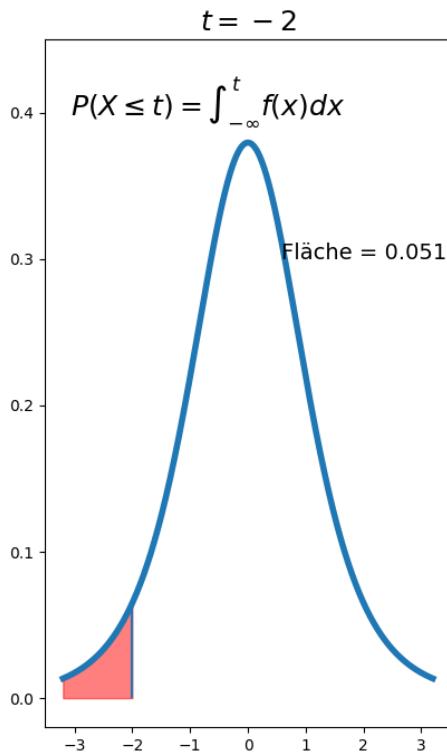


Eine weitere sehr nützliche Funktion ist die Funktion `t.cdf()`, die die Fläche unter der t -Kurve für ein beliebiges Intervall liefert. Berechnen wir die Fläche unter der Kurve für die Intervalle $j_i =] -\infty, -2]$, $] -\infty, 0]$, $] -\infty, 2]$ und $k_i = [-2, \infty[$, $[0, \infty[$, $[2, \infty[$ für eine Zufallsvariable mit einer t -Verteilung mit $df = 5$.

```
x_cdf_left = []
# Berechne kumulative Wahrscheinlichkeit links von Wert _t
for _t in [-2, 0, 2]:
    res = t.cdf(_t, df=5)
    print(f"Viert für Fläche links von {_t}: {res}")
    x_cdf_left.append(res)
```

```
Viert für Fläche links von -2: 0.05096973941492914
Viert für Fläche links von 0: 0.5
Viert für Fläche links von 2: 0.9490302605850709
```

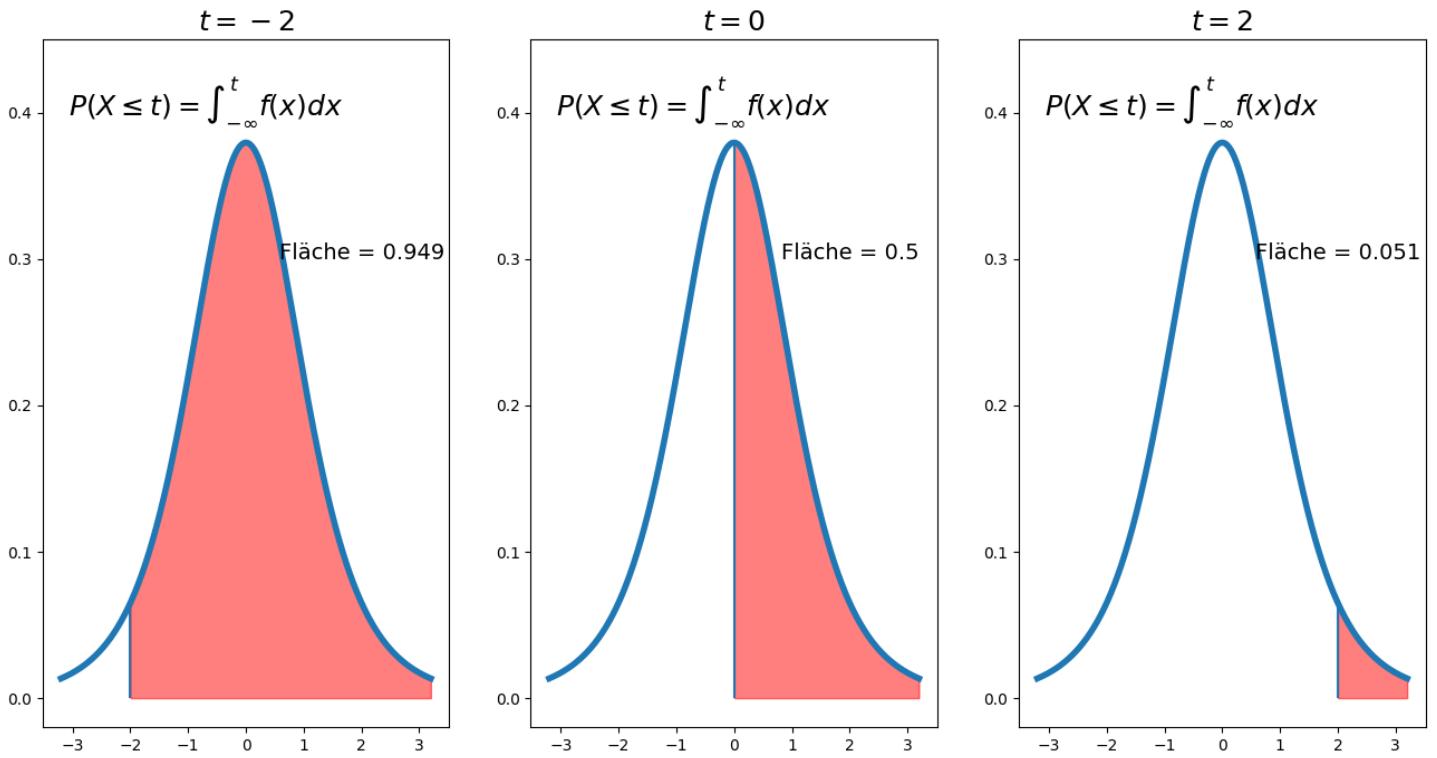
► Show code cell source



```
x_cdf_right = []
# Berechne kumulative Wahrscheinlichkeit rechts von Wert _t
for _t in [-2, 0, 2]:
    res = 1 - t.cdf(_t, df=5)
    print(f"Wert für Fläche rechts von {_t}: {res}")
    x_cdf_right.append(res)
```

Wert für Fläche rechts von -2: 0.9490302605850709
 Wert für Fläche rechts von 0: 0.5
 Wert für Fläche rechts von 2: 0.050969739414929105

► Show code cell source



Die Funktion `t.ppf()` liefert die Quantilfunktion und ist damit die Umkehrfunktion von `t.cdf()`. Für die Intervalle $j_i =] -\infty, -2]$, $] -\infty, 0]$, $] -\infty, 2]$ einer Zufallsvariablen, die einer t -Verteilung mit $df = 5$ folgt, liefert die Funktion `t.ppf()`...

`x_cdf_left`

```
[np.float64(0.05096973941492914),
 np.float64(0.5),
 np.float64(0.9490302605850709)]
```

```
for x in x_cdf_left:
    print(f"round(x,2): {round(t.ppf(x, df=5), 2)}")
```

```
0.05: -2.0
0.5: 0.0
0.95: 2.0
```

... und für die Intervalle $k_i = [-2, \infty[$, $[0, \infty[$, $[2, \infty[$ einer Zufallsvariablen, die einer t -Verteilung mit $df = 5$ folgt, liefert die Funktion `t.ppf`

```
for x in x_cdf_right:  
    print(f"round(x,2): {round(t.ppf(x, df=5), 2)}")
```

```
0.95: 2.0  
0.5: 0.0  
0.05: -2.0
```

Die Chi-Quadrat-Verteilung

```
import matplotlib.pyplot as plt  
import numpy as np  
from scipy.stats import chi2
```

Die Chi-Quadrat (χ^2) ist eine der wichtigsten kontinuierlichen Wahrscheinlichkeitsverteilungen mit vielen Anwendungen in der statistischen Theorie und Inferenz (s.441).

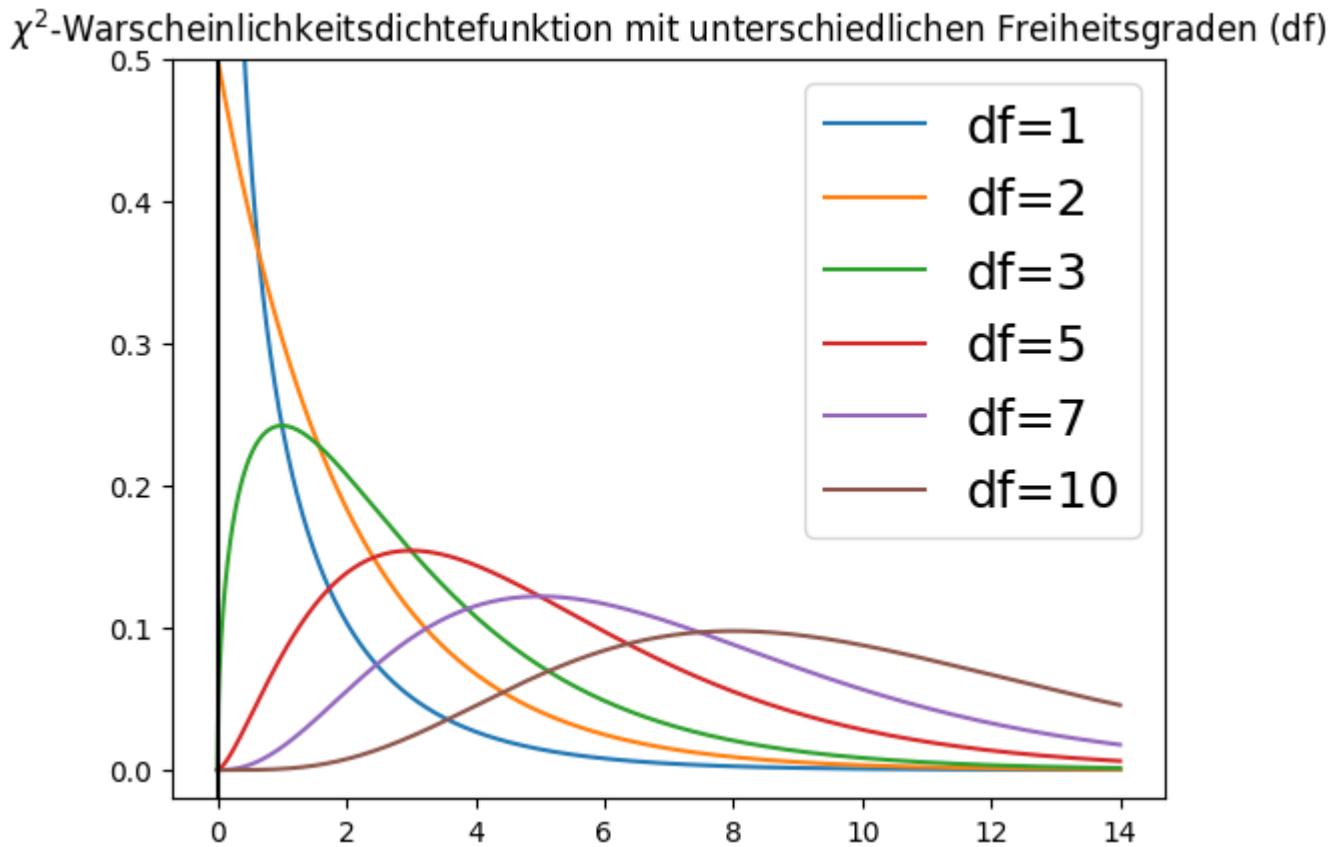
Sei $n > 0$ eine positive ganze Zahl. Für eine Zufallsvariable, die eine (χ^2)-Verteilung mit n Freiheitsgraden (df) hat, lautet die Wahrscheinlichkeitsdichtefunktion

$$f(x) = \begin{cases} 0 & \text{wenn } x \leq 0 \\ \frac{x^{(n/2-1)} e^{-x/2}}{2^{n/2} \Gamma(\frac{k}{2})} & \text{wenn } x > 0 \end{cases}$$

wobei Γ die [Gamma-Funktion](#) bezeichnet. Die (χ^2)-Verteilung (mit n Freiheitsgraden) ist gleich der Γ -Verteilung mit den Parametern $(n/2, 2)$, d. h. mit Mittelwert und Varianz gleich n bzw. $2n$.

▶ Show code cell source

```
<matplotlib.lines.Line2D at 0x74ee6fb824a0>
```



Grundlegende Eigenschaften von χ^2 -Kurven

Die Gesamtfläche unter einer χ^2

- Kurve ist gleich 1
- Eine χ^2 -Kurve beginnt bei 0 auf der horizontalen Achse und erstreckt sich unendlich weit nach rechts, wobei sie sich der horizontalen Achse nähert, diese aber nie berührt.
- Eine χ^2 -Kurve ist rechtsschief.
- Mit zunehmender Anzahl von Freiheitsgraden sehen χ^2 -Kurven zunehmend wie normalverteilt aus.

Die Chi-Quadrat-Verteilung in Python

Die wichtigsten Funktionen zur Interaktion mit der χ^2 -Verteilung sind `chi2.pdf()`, `chi2.cdf()`, `chi2.ppf()`, `chi2.rvs()`. Die Funktion `chi2.pdf()` liefert die Dichte, die Funktion `chi2.cdf()`

die Verteilungsfunktion, die Funktion `chi2.ppf()` die Quantilfunktion und die Funktion `chi2.rvs()` die Zufallsabweichungen.

Wir verwenden die Funktion `chi2.pdf()`, um die Dichte für die ganzzahligen Werte 4 bis 8 einer χ^2 -Kurve mit $df = 7$.

```
for i in range(4, 9):  
    print(chi2.pdf(i, df=7))
```

```
0.11518072856146785  
0.12204152134938738  
0.11676521599113947  
0.10411977480817192  
0.08817913751079275
```

Wir verwenden `chi2.cdf()`, um die Fläche unter der Kurve für das Intervall $[0, 6]$ und das Intervall $[6, \infty[$ einer χ^2 -Kurve mit $df = 7$ zu berechnen. Weiter fragen wir Python, ob die Summe der Intervalle $[0, 6]$ und $[6, \infty[$ den Wert 1 ergibt.

```
chi2_cdf1 = chi2.cdf(6, df=7)  
chi2_cdf1
```

```
np.float64(0.4602506496044429)
```

```
chi2_cdf2 = 1 - chi2.cdf(6, df=7)  
chi2_cdf2
```

```
np.float64(0.539749350395557)
```

```
chi2_cdf1 + chi2_cdf2
```

```
np.float64(1.0)
```

Wir verwenden `chi2.ppf()`, um das Quantil für eine bestimmte Fläche (= Wahrscheinlichkeit) unter der Kurve für eine χ^2 -Kurve mit $df = 7$ zu berechnen, die $q = 0, 25, 0, 5, 0, 75$ und $0, 999$

entspricht.

```
chi2.ppf(0.25, df=7)
```

```
np.float64(4.2548521835465145)
```

```
chi2.ppf(0.5, df=7)
```

```
np.float64(6.345811195521515)
```

```
chi2.ppf(0.75, df=7)
```

```
np.float64(9.037147547908143)
```

```
chi2.ppf(0.999, df=7)
```

```
np.float64(24.321886347856854)
```

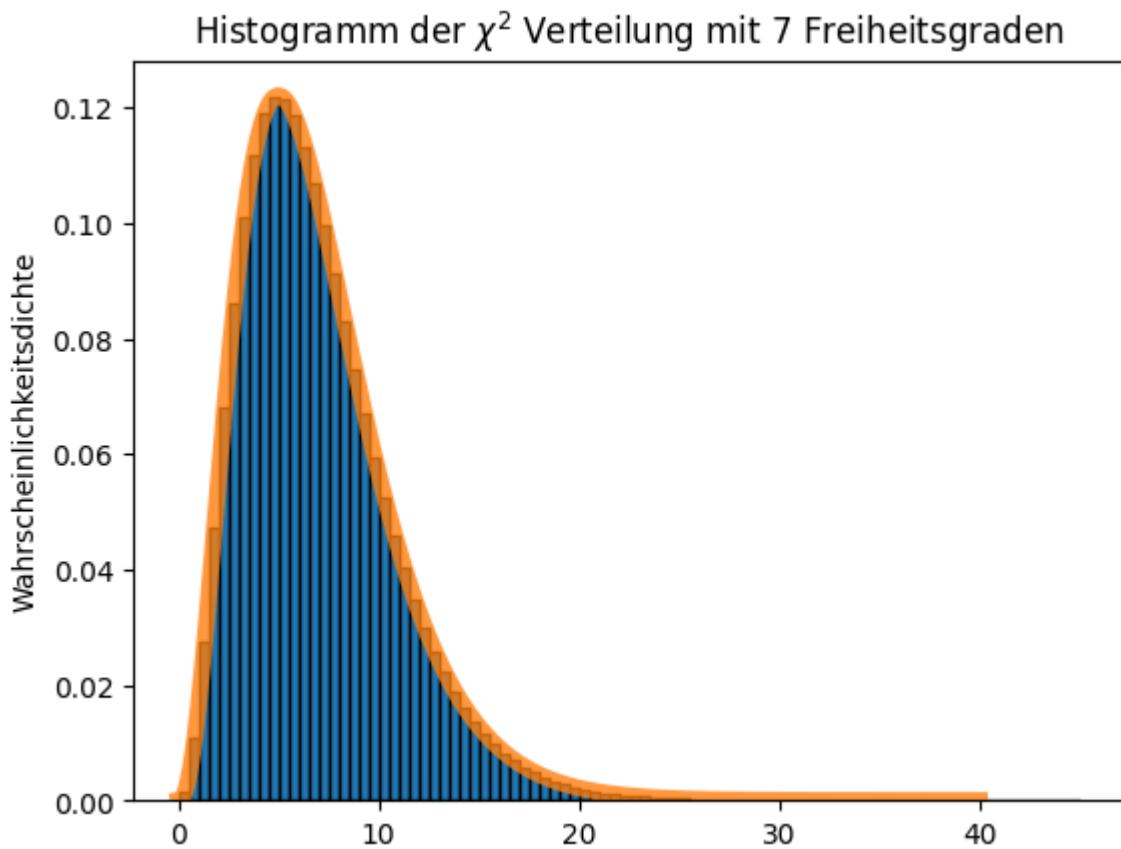
Wir verwenden die Funktion `chi2.rvs()`, um 100.000 Zufallswerte aus der χ^2 -Verteilung mit $df = 7$ zu erzeugen. Anschließend zeichnen wir ein Histogramm und vergleichen es mit der Wahrscheinlichkeitsdichtefunktion der χ^2 -Verteilung mit $df = 7$ (orangefarbene Linie).

```
# Erzeuge Chi^2 Werte
chi2_rvs = chi2.rvs(df=7, size=10000000)

# Erzeuge x-werte
x = np.linspace(0, 40, num=1000)

# Plotte chi2-Verteilung
fig, ax = plt.subplots()
ax.set_title("Histogramm der $\chi^2$ Verteilung mit 7 Freiheitsgraden")
ax.set_ylabel("Wahrscheinlichkeitsdichte")
ax.hist(chi2_rvs, bins=90, edgecolor="k", density=True)
ax.plot(x, chi2.pdf(x, df=7), linewidth=6, alpha=0.8)
```

```
[<matplotlib.lines.Line2D at 0x74ee6f96e8f0>]
```



Die F-Verteilung

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import f
```

Die Snedecor- F -Verteilung oder die Fisher-Snedecor-Verteilung (nach [Sir Ronald A. Fisher](#) und [George W. Snedecor](#)) oder kurz die **F -Verteilung** ist eine kontinuierliche Wahrscheinlichkeitsverteilung mit dem Bereich $[0, +\infty[$, abhängig von zwei Parametern, die mit v_1, v_2 bezeichnet werden ([]) s.281, [] s.179). In statistischen Anwendungen sind v_1, v_2 positive ganze Zahlen.

Seien Y_1 und Y_2 seien zwei unabhängige Zufallsvariablen, die **Chi-Quadrat**-verteilt sind, mit v_1 bzw. v_2 Freiheitsgraden. Dann wird die Verteilung des Verhältnisses (Z)

$$Z = \frac{Y_1/v_1}{Y_2/v_2}$$

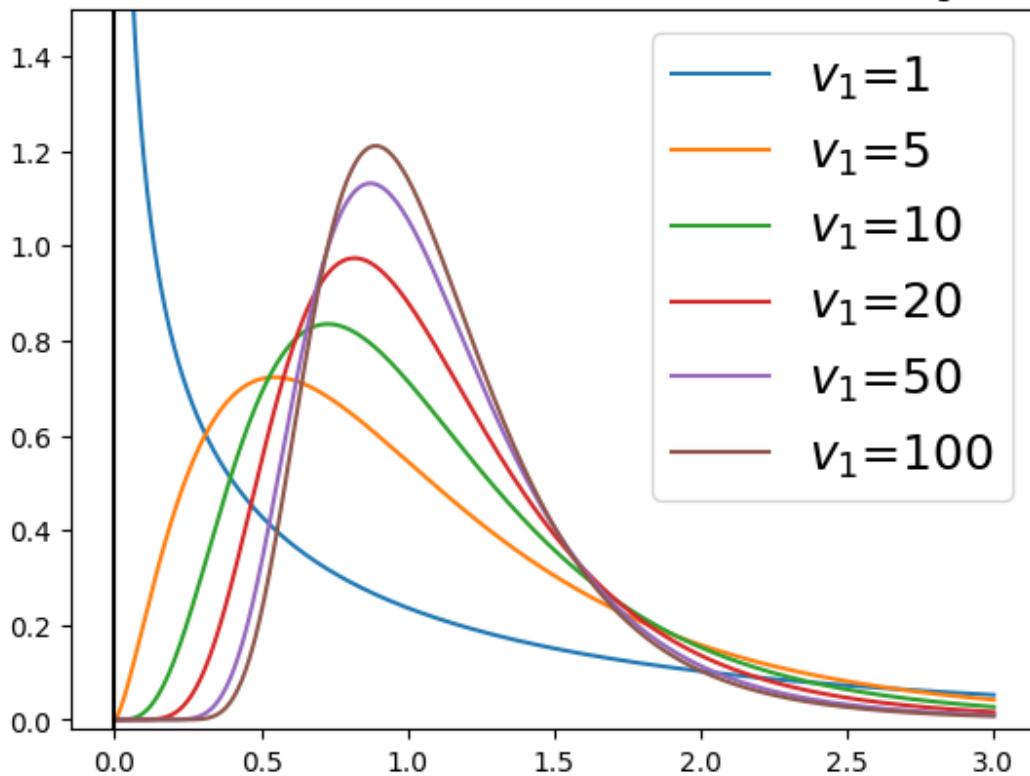
als F -Verteilung mit v_1 und v_2 Freiheitsgraden bezeichnet. Die F -Verteilung wird häufig auch als *Verteilung des Varianzverhältnisses* bezeichnet ([] s.281).

Eine F -Verteilung hat zwei Zahlen von Freiheitsgraden, v_1 und v_2 , die ihre Form bestimmen. Die erste Zahl der Freiheitsgrade, v_1 , wird als die **Freiheitsgrade des Zählers** und die zweite, v_2 , als die **Freiheitsgrade des Nenners** bezeichnet.

▶ Show code cell source

```
<matplotlib.lines.Line2D at 0x76f806f4e500>
```

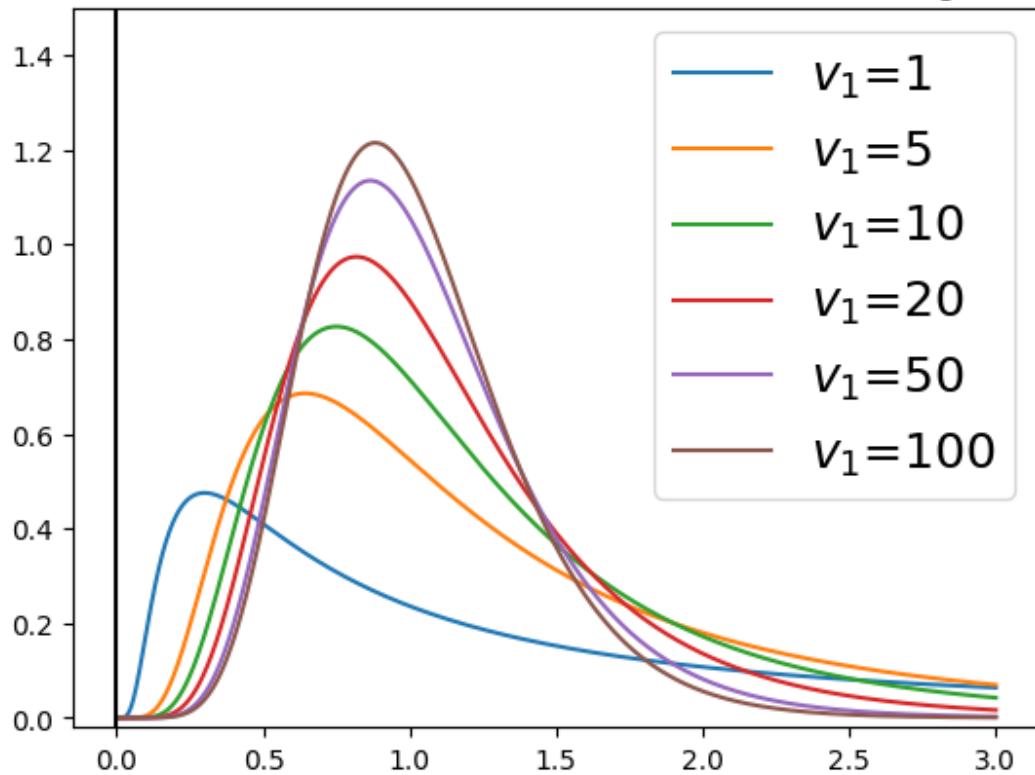
F-Wrscheinlichkeitsdichtefunktion mit unterschiedlichen Freiheitsgraden ($v_1, v_2 = 20$)



▶ Show code cell source

```
<matplotlib.lines.Line2D at 0x76f806d3ada0>
```

F-Wrscheinlichkeitsdichtefunktion mit unterschiedlichen Freiheitsgraden ($v_1 = 20, v_2$)



Grundlegende Eigenschaften von F-Kurven

- Die Gesamtfläche unter einer F -Kurve ist gleich 1. -Eine F -Kurve beginnt bei 0 auf der horizontalen Achse und erstreckt sich unendlich weit nach rechts, wobei sie sich der horizontalen Achse nähert, diese aber nie berührt.
- Eine F -Kurve ist rechtsschief.

Die F-Verteilung in Python

Die wichtigsten Funktionen zur Interaktion mit der F -Verteilung sind `f.pdf()`, `f.cdf()`, `f.ppf()`, `f.rvs()`. Die Funktion `f.pdf()` liefert die Dichte, die Funktion `f.cdf()` die Verteilungsfunktion, die Funktion `f.ppf()` die Quantilfunktion und die Funktion `f.rvs()` erzeugt Zufallsabweichungen.

Wir verwenden die Funktion `f.pdf()`, um die Dichte bei einem Wert von 1, 2 einer F -Kurve mit $v_1 = 10$ und $v_2 = 20$.

```
f.pdf(1.2, 10, 20)
```

```
np.float64(0.5626124566227062)
```

Wir verwenden `f.cdf()`, um die Fläche unter der Kurve für das Intervall $[0, 1, 5]$ und das Intervall $[1, 5, +\infty[$ einer F-Kurve mit $v_1 = 10$ und $v_2 = 20$ zu berechnen. Weiter fragen wir Python, ob die Summe der Intervalle $[0, 1, 5]$ und $[1, 5, +\infty[$ den Wert 1 ergibt.

```
f_cdf1 = f.cdf(1.5, 10, 20)  
f_cdf1
```

```
np.float64(0.7890535374813874)
```

```
f_cdf2 = 1 - f.cdf(1.5, 10, 20)  
f_cdf2
```

```
np.float64(0.2109464625186126)
```

```
f_cdf1 + f_cdf2
```

```
np.float64(1.0)
```

Wir verwenden `f.ppf()`, um das Quantil für eine bestimmte Fläche (= Wahrscheinlichkeit) unter der Kurve für eine F -Kurve mit $v_1 = 10$ und $v_2 = 20$ zu berechnen, die $q = 0, 25, 0, 5, 0, 75$ und $0, 999$ entspricht.

```
q = [0.25, 0.5, 0.75, 0.999]  
f.ppf(q, 10, 20)
```

```
array([0.65639363, 0.96626389, 1.39948744, 5.07524621])
```

Wir verwenden die Funktion `f.rvs()`, um 100.000 Zufallswerte aus der F -Verteilung mit $v_1 = 10$ und $v_2 = 20$ zu erzeugen. Anschließend zeichnen wir ein Histogramm und vergleichen es mit der Wahrscheinlichkeitsdichtefunktion der F -Verteilung mit $v_1 = 10$ und $v_2 = 20$ (orange Linie).

```
# Erzeuge Chi^2 Werte
f_rvs = f.rvs(10, 20, size=10000)

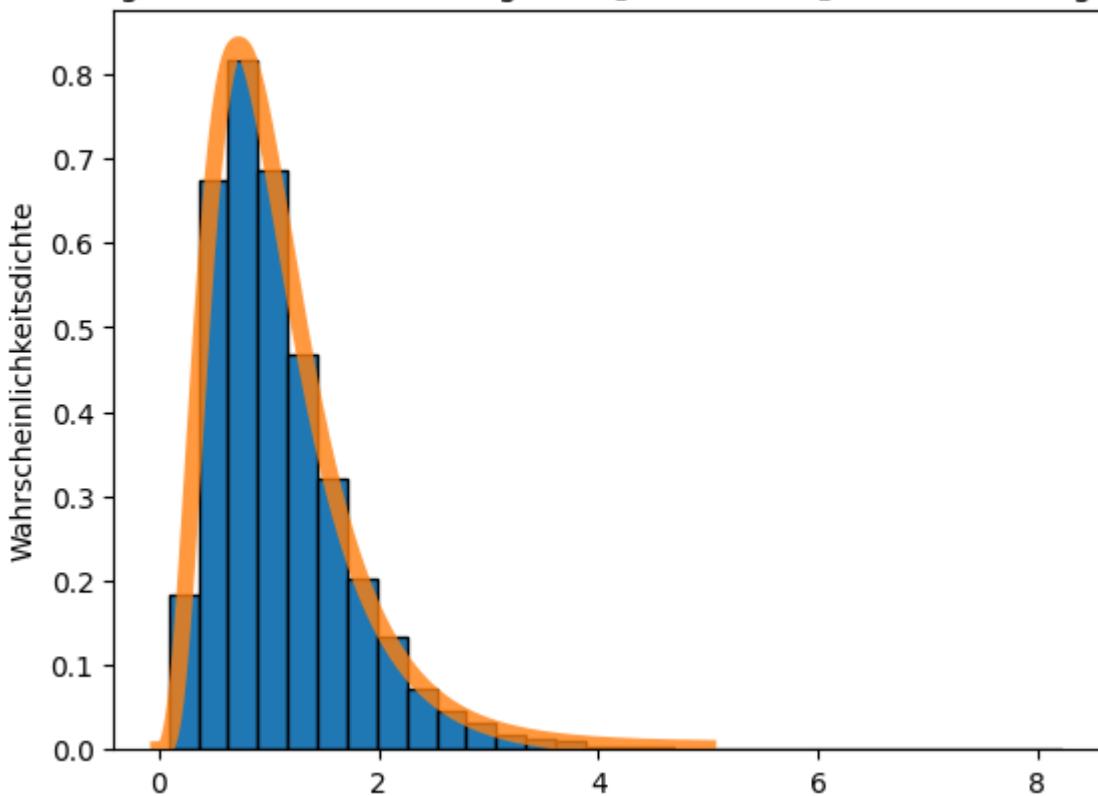
# Erzeuge x-werte
x = np.linspace(0, 5, num=1000)

# Plotte t-Verteilung
fig, ax = plt.subplots()
# plt.xlim(0,4.5)

ax.set_title("Histogramm der F-Verteilung mit $v_1=10$ und $v_2=20$ Freiheitsgraden")
ax.set_ylabel("Wahrscheinlichkeitsdichte")
ax.hist(f_rvs, bins=30, edgecolor="k", density=True)
ax.plot(x, f.pdf(x, 10, 20), linewidth=6, alpha=0.8)
```

[<matplotlib.lines.Line2D at 0x76f806d981f0>]

Histogramm der F-Verteilung mit $v_1 = 10$ und $v_2 = 20$ Freiheitsgraden



Übungsaufgaben

Wahrscheinlichkeitsdichtefunktion

1. Nennen Sie die 3 Haupteigenschaften von Wahrscheinlichkeitsdichtefunktionen (PDF).
 2. Wie kann das Flächenintegral über eine Wahrscheinlichkeitsdichtefunktion interpretiert werden?
 3. Mit welcher Python Funktion kann die Wahrscheinlichkeitsdichtefunktionen der Normaverteilung bestimmt werden?
-

Lösungen

Die 68-95-99,7-Regel

1. Was sind die Hauptaussagen der 68 – 95 – 99,7-Regel?
2. Berechnen Sie die entsprechenden Integrale nach der 68 – 95 – 99,7-Regel für die Normalverteilung $X \sim N(-2, 2)$

```
# Frage 2 ...
```

Lösungen

Frage 1

Frage 2

► Show code cell content

Die Normalverteilung

1. Welche Kenngrößen charakterisieren die Normalverteilung
2. Was ist die Standardnormalverteilung und in welchem Bezug steht sie zur Normalverteilung?
3. Generieren Sie 10.000 Zufallswerte für die Normalverteilung mit Mittelwert $\mu = 1$ und Standardabweichung $\sigma = 3$, unter Verwendung der Funktion `np.random.normal(loc, scale, size)`, und stellen Sie das Ergebnis als Histogramm dar.
4. Führen Sie eine z -Transformation für diese Werte durch und plotten Sie das Ergebnis.

```
# Frage 3 ...
```

```
# Frage 4 ...
```

Lösungen

► Show code cell content

► Show code cell content

Lernziele

Zentraler Grenzwertsatz

- Die Kernaussagen des zentralen Grenzwertsatzes werden vorgestellt und anhand von interaktiven Beispielen wird veranschaulicht, daß durch Entnahme von Stichproben geeigneten Umfangs, Aussagen über die zugrundeliegende Grundgesamtheit getroffen werden können
- Die Studierenden lernen, daß die entnommene Stichprobe als neue Zufallsvariable aufzufassen ist und können selbständig überprüfen, daß Folgen von Stichproben, unabhängig von ihrer ursprünglichen Verteilung, in ihrem Mittelwert, bei n -facher Wiederholung der Stichprobe, gegen eine Normalverteilung um denselben Mittelwert konvergieren
- Die Studierenden lernen den Standardfehler $\sigma_{\bar{x}}$ anhand des Beispiels des Standardfehlers des Mittelwerts $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$ kennen
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Zentraler Grenzwertsatz, Grundgesamtheit, Stichprobenverteilungen, Stichprobenfehler, Standardfehler

Der zentrale Grenzwertsatz

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm, uniform, beta, gamma
```

Der [zentrale Grenzwertsatz](#) ist eines der nützlichsten Konzepte der Statistik. Bei diesem Theorem geht es um die Ziehung von Stichproben einer endlichen Größe n aus einer Grundgesamtheit. Das Theorem besagt, dass, wenn man Stichproben mit einem ausreichend großen Stichprobenumfang n sammelt und den Mittelwert jeder Stichprobe berechnet, die Form des Histogramms dieser Mittelwerte sich einer Gauß-Verteilung annähert. Die Nützlichkeit des zentralen Grenzwertsatzes ergibt sich aus der Tatsache, dass **die Verteilung der Stichprobenmittelwerte unabhängig von der Verteilung der ursprünglichen Verteilung der Zufallsvariablen der Normalverteilung folgt** (s.436).

Die Grundgesamtheitsverteilung

Die **Grundgesamtheitsverteilung** ist die Wahrscheinlichkeitsverteilung, die sich aus der Kenntnis aller Elemente einer Grundgesamtheit ergibt ([] s.302). Wir wissen, dass die interessierende Zufallsvariable je nach der betrachteten Grundgesamtheit eine diskrete Variable sein kann, d. h. eine Variable, die zumindest im Prinzip abzählbar ist (abzählbar unendlich), oder die Zufallsvariable kann eine kontinuierliche Variable sein, d. h. eine Variable, die jeden Wert innerhalb eines bestimmten Intervalls annehmen kann (überabzählbar unendlich). Sowohl die diskrete als auch die kontinuierliche Wahrscheinlichkeitsverteilung kann durch statistische Parameter wie den Mittelwert, die Standardabweichung, den Median, den Modalwert und andere beschrieben werden. Diese Parameter, die die Grundgesamtheit beschreiben, sind jedoch **immer konstant**, da die Grundgesamtheit die Menge aller Elemente ist und sich somit die Grundgesamtheitsstatistik nicht ändert. So gibt es beispielsweise für jeden Populationsdatensatz **nur einen Wert** für den Populationsmittelwert, **einen Wert** für die Standardabweichung usw.

Grundgesamtheitsstatistiken und

Stichprobenstatistiken

Betrachten wir ein einfaches Beispiel für eine kleine diskrete Grundgesamtheit, die aus den ersten zehn ganzen Zahlen $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ besteht.

```
population = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
mean = np.mean(population)
std = np.std(population)

print(f"Mittelwert (Grundgesamtheit): {mean}")
print(f"Standartabweichung (Grundgesamtheit): {std}")
```

```
Mittelwert (Grundgesamtheit): 5.5
Standartabweichung (Grundgesamtheit): 2.8722813232690143
```

Der Populationsmittelwert μ , und die Populationsstandardabweichung σ beträgt 5,5 bzw. etwa 2,872. Es ist wichtig zu erkennen, dass sich diese Parameter, die Populationsparameter, nicht ändern! Sie sind durch die Grundgesamtheit festgelegt.

Nehmen wir nun eine Zufallsstichprobe ohne Ersetzung mit dem Umfang $n = 3$ aus dieser Grundgesamtheit.

```
np.random.seed(1)

my_sample = np.random.choice(population, size=3, replace=False)
my_sample
```

```
array([ 3, 10,  7])
```

Nun berechnen wir den Mittelwert und die Standardabweichung der gegebenen Stichprobe. Da wir uns aber auf eine bestimmte Stichprobe beziehen, nennen wir den statistischen Parameter diesmal **Stichprobenstatistik** oder, wenn wir uns auf die Verteilung der Werte (Elemente) beziehen, **Stichprobenverteilung**. Um dies zu verdeutlichen, wird der Stichprobenmittelwert mit \bar{x} und die Stichprobenstandardabweichung mit s bezeichnet.

```

x_bar = np.mean(my_sample)
s = np.std(my_sample, ddof=1)

print(f"Mittelwert (Stichprobe): {x_bar}")
print(f"Standartabweichung (Stichprobe): {s}")

```

```

Mittelwert (Stichprobe): 6.666666666666667
Standartabweichung (Stichprobe): 3.5118845842842465

```

Bitte beachten Sie, dass sich die Stichprobenstatistiken je nach den tatsächlichen Elementen in der Stichprobe von Stichprobe zu Stichprobe ändern.

Der Schätzfehler

Wir wiederholen die Stichprobe aus dem vorigen Abschnitt fünfmal und geben den Mittelwert \bar{x} für jede einzelne Stichprobe aus.

```

population = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(5):
    my_sample = np.random.choice(population, size=3, replace=False)
    mean = np.mean(my_sample)
    print(f"Die {i+1}. Stichprobe hat einen Mittelwert von {mean}")

```

```

Die 1. Stichprobe hat einen Mittelwert von 6.666666666666667
Die 2. Stichprobe hat einen Mittelwert von 6.333333333333333
Die 3. Stichprobe hat einen Mittelwert von 4.0
Die 4. Stichprobe hat einen Mittelwert von 6.666666666666667
Die 5. Stichprobe hat einen Mittelwert von 5.333333333333333

```

Es liegt auf der Hand, dass verschiedene Stichproben (mit derselben Grösse), die aus derselben Grundgesamtheit ausgewählt wurden, unterschiedliche Stichprobenstatistiken ergeben, da sie unterschiedliche Elemente enthalten. Darüber hinaus unterscheidet sich jede aus einer Stichprobe gewonnene Stichprobenstatistik, z. B. der Stichprobenmittelwert \bar{x} , von dem Ergebnis, das aus der entsprechenden Grundgesamtheit, dem Grundgesamtheitsmittelwert μ , gewonnen wird. Die Differenz zwischen dem Wert einer aus einer Stichprobe gewonnenen Statistik und dem Wert des entsprechenden, aus der Grundgesamtheit gewonnenen Parameters wird als Schätzfehler bezeichnet. Im Fall des Mittelwerts kann der Schätzfehler wie folgt geschrieben werden

$$\text{Schätzfehler} = \bar{x} - \mu$$

Aufgrund des Charakters von Zufallsstichproben und willkürlichen Ziehung einer Reihe von Werten aus der Grundgesamtheit ist der daraus resultierende Schätzfehler zufällig, oder anders gesagt, der Schätzfehler ist eine Zufallsvariable. Es ist jedoch zu beachten, dass es neben der beschriebenen Zufälligkeit noch andere Fehlerquellen gibt. Diese Fehler hängen oft mit dem Prozess der Datenerzeugung zusammen. Solche Fehler werden beispielsweise durch die menschliche Handhabung der Daten, Kalibrierungsfehler der Messgeräte etc. verursacht.

Um ein Gefühl für die Art des Schätzfehlers zu bekommen, führen wir ein Experiment durch. Bei diesem Experiment besteht die interessierende Grundgesamtheit aus den ersten 100 ganzen Zahlen $\{1, 2, 3, \dots, 100\}$. Wir wollen den Einfluss des Stichprobenumfangs n auf den Schätzfehler untersuchen. Der Einfachheit halber wählen wir den Stichprobenmittelwert als die interessierende Statistik. Für eine ausreichend große Anzahl von Versuchen (z.B. 5000 Versuche) berechnen wir den Schätzfehler für Stichproben mit dem Umfang $n = 10, 25, 50, 75$.

```
trial_size = 5000 # 5000 Versuche
sample_size = [10, 25, 50, 75] # Stichprobenumfang
population = range(1, 101)
mean_pop = np.mean(population)
for n in sample_size:
    error_sample = []
    for _ in range(trial_size):
        my_sample = np.random.choice(population, size=n, replace=False)
        mean = np.mean(my_sample)
        error_sample.append(abs(mean - mean_pop))
    print(f"Schätzfehler (n={n}):", np.round(np.mean(error_sample), 3))
```

Schätzfehler (n=10): 6.886

Schätzfehler (n=25): 4.034

Schätzfehler (n=50): 2.339
Schätzfehler (n=75): 1.359

Aus dem obigen Experiment können wir schließen, dass der Schätzfehler umso kleiner ist, je größer der Stichprobenumfang ist. Mit anderen Worten: Je größer der Stichprobenumfang ist, desto mehr

nähert sich der Stichprobenmittelwert \bar{x} dem Grundgesamtheitsmittelwert μ an. Dies ist eine wichtige Erkenntnis, die im Abschnitt über die *Inferenzstatistik* ausführlicher behandelt werden wird.

Die Stichprobenverteilung

Ausgehend von unserer Intuition der Zufälligkeit im Stichprobenprozess führen wir die **Stichprobenverteilung** ein. Die Stichprobenverteilung ist eine Verteilung einer Stichprobenstatistik (↗ s.337). Oft wird der Name der berechneten Statistik als Teil des Titels hinzugefügt. Wenn es sich bei der berechneten Statistik beispielsweise um den Stichprobenmittelwert handelt, würde die Stichprobenverteilung den Titel **Stichprobenverteilung des Stichprobenmittelwerts** tragen.

Erinnern wir uns an das einfache Beispiel aus dem vorigen Abschnitt, bei dem die Grundgesamtheit durch die ersten 100 ganzen Zahlen $\{1, 2, 3, \dots, 100\}$ repräsentiert wurde. Wenn wir wiederholt Stichproben aus dieser Grundgesamtheit ziehen und jedes Mal die Stichprobenstatistik (z. B. \bar{x} oder s, \dots) berechnen, wird **die resultierende Verteilung der Stichprobenstatistik als Stichprobenverteilung dieser Statistik** bezeichnet.

Aus dieser Grundgesamtheit nehmen wir wiederholt Zufallsstichproben (x) ohne Ersetzung mit der Größe $n = 30$. Die Zufallsstichproben könnten Mengen erzeugen, die wie folgt aussehen :

$\{19, 79, 33, 38, 14, 67, 7, 9, 12, 27, 4, 89, 34, 77, 78, 32, 65, 10, 84, 64, 90, 55, 88, 56, 11, 8, \dots\}$

oder

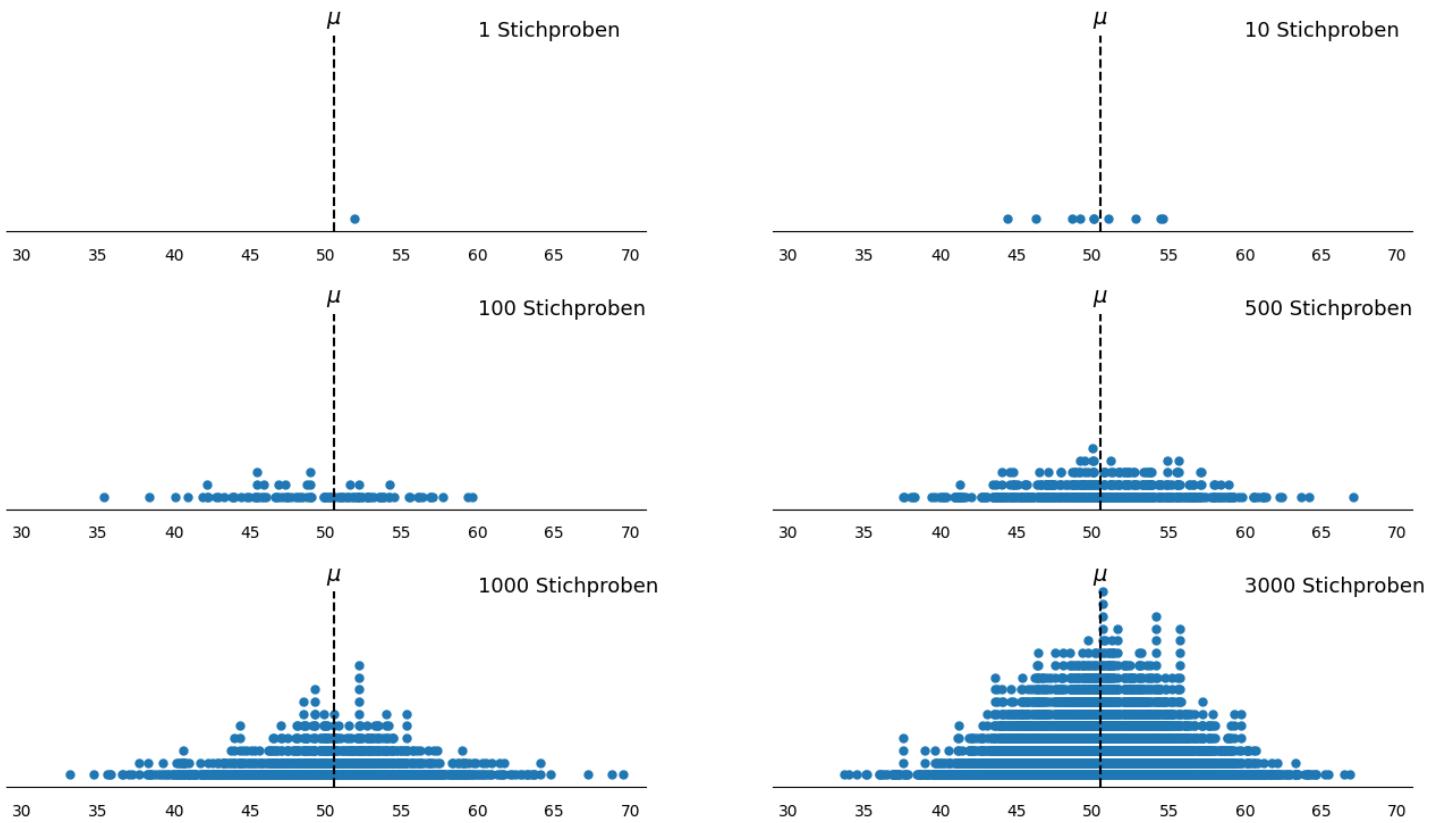
$\{43, 52, 56, 8, 65, 60, 46, 15, 64, 19, 82, 91, 88, 1, 5, 9, 4, 92, 67, 36, 72, 31, 50, 96, 87, 6, 9, \dots\}$

... etc.

Für jede Stichprobe berechnen wir eine Stichprobenstatistik. In diesem Beispiel nehmen wir den Mittelwert, \bar{x} , jeder Stichprobe. Beachten Sie jedoch, dass es sich bei der Stichprobenstatistik um eine beliebige deskriptive Statistik handeln kann, z. B. um den Median, die Standardabweichung, einen Anteil usw. Sobald wir die Stichprobenmittelwerte für alle Stichproben erhalten haben, listen wir alle ihre verschiedenen Werte und die Anzahl ihres Auftretens (Häufigkeiten) auf, um relative Häufigkeiten oder **empirische Wahrscheinlichkeiten** zu erhalten.

► Show code cell source

► Show code cell source



Je häufiger wir eine Stichprobe nehmen, desto besser nähert sich die relative Häufigkeitsverteilung der Stichprobenstatistik der Grundgesamtheitsverteilung an. Mit anderen Worten: Wenn die Anzahl der Stichproben gegen unendlich geht, nähert sich die resultierende Häufigkeitsverteilung der Grundgesamtheitsverteilung an. **Die Stichprobenverteilung einer Statistik** ist eine Wahrscheinlichkeitsverteilung dieser Statistik, die aus allen möglichen Stichproben mit demselben Umfang aus der Grundgesamtheit abgeleitet wird. Die Stichprobenverteilung sollte jedoch nicht mit einer Grundgesamtheitsverteilung verwechselt werden: Letztere beschreibt die Verteilung der Werte (Elemente) in der Grundgesamtheit.

Der Standardfehler

Ebenso wie die Verteilungen der Grundgesamtheit können auch die Stichprobenverteilungen mit Parametern beschrieben werden. Der Erwartungswert (Mittelwert) einer beliebigen Verteilung kann durch das Symbol μ dargestellt werden. Im Falle der Stichprobenverteilung wird der Mittelwert μ oft mit einem tiefgestellten Index geschrieben, um anzugeben, welche Stichprobenverteilung beschrieben wird. Der Erwartungswert der Stichprobenverteilung des Mittelwerts wird zum Beispiel

durch das Symbol $\mu_{\bar{x}}$ dargestellt. Der Wert von $\mu_{\bar{x}}$ kann als der theoretische Mittelwert der Verteilung der Stichprobenmittelwerte angesehen werden.

Wenn wir aus einer Grundgesamtheit eine ausreichend große Anzahl von Stichproben (mit gleichem Umfang) auswählen und deren Mittelwerte berechnen, dann nähert sich der Mittelwert ($\mu_{\bar{x}}$) all dieser Stichprobenmittelwerte dem Mittelwert (μ) der Grundgesamtheit an. Deshalb wird der Stichprobenmittelwert \bar{x} als Schätzer des Populationsmittelwertes μ bezeichnet. Somit ist der Mittelwert der Stichprobenverteilung gleich dem Mittelwert der Grundgesamtheit.

$$\mu_{\bar{x}} = \mu$$

Für die Standardabweichung einer Stichprobenverteilung gibt es eine besondere Bezeichnung, den **Standardfehler**. Der Standardfehler der Stichprobenverteilung einer Statistik, bezeichnet als $\sigma_{\bar{x}}$, beschreibt das Ausmaß, in dem die berechneten Statistiken erwartungsgemäß voneinander abweichen, wenn sie anhand einer Stichprobe ähnlichen Umfangs berechnet und aus ähnlichen Grundgesamtheitsmodellen ausgewählt werden. Je größer der Standardfehler einer bestimmten Statistik ist, desto größer sind die Unterschiede zwischen den berechneten Statistiken für die verschiedenen Stichproben ([] s.133).

Es ist jedoch zu beachten, dass der Standardfehler $\sigma_{\bar{x}}$ nicht gleich der Standardabweichung σ der Verteilung der Grundgesamtheit ist (es sei denn, $n = 1$). Der Standardfehler ist gleich der Standardabweichung der Grundgesamtheit geteilt durch die Quadratwurzel des Stichprobenumfangs :

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

Diese Gleichung gilt nur, wenn die Stichprobe entweder mit Ersatz aus einer endlichen Grundgesamtheit oder mit oder ohne Ersatz aus einer unendlichen Grundgesamtheit gezogen wird. Dies entspricht der Bedingung, dass der Stichprobenumfang (n) im Vergleich zum Grundgesamtheitsumfang (N) klein ist. Der Stichprobenumfang gilt als klein im Vergleich zum Umfang der Grundgesamtheit, wenn der Stichprobenumfang gleich oder weniger als 5% des Umfangs der Grundgesamtheit ist, d. h., wenn

$$\frac{n}{N} \leq 0,05$$

Wenn diese Bedingung nicht erfüllt ist, wird die folgende Gleichung zur Berechnung von $\sigma_{\bar{x}}$ verwendet :

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \sqrt{\frac{N-n}{N-1}}$$

In den meisten praktischen Anwendungen ist der Stichprobenumfang jedoch klein im Vergleich zur Grundgesamtheit.

Stichproben aus einer normalverteilten Grundgesamtheit

Form der Stichprobenverteilung

Die Form der Stichprobenverteilung bezieht sich auf die beiden folgenden Fälle :

1. Die Grundgesamtheit, aus der die Stichproben gezogen werden, ist normalverteilt.
2. Die Grundgesamtheit, aus der die Stichproben gezogen werden, ist nicht normalverteilt.

Stichproben aus einer normalverteilten Grundgesamtheit

Wenn die Grundgesamtheit, aus der die Stichproben gezogen werden, normalverteilt ist und ihr Mittelwert gleich μ und ihre Standardabweichung gleich σ ist, dann gilt :

1. Der Mittelwert der Stichprobenmittel, $\mu_{\bar{x}}$, ist gleich dem Mittelwert der Grundgesamtheit, μ
2. Die Standardabweichung der Stichprobenmittelwerte, $\sigma_{\bar{x}}$ ist gleich $\frac{\sigma}{\sqrt{n}}$, wobei $\frac{n}{N} \leq 0,05$ angenommen wird.
3. Die Form der Stichprobenverteilung der Stichprobenmittelwerte \bar{x} ist normal, unabhängig vom Wert von n .

Betrachten wir eine normalverteilte Grundgesamtheit. Der Einfachheit halber verwenden wir die Standardnormalverteilung, $N \sim (\mu, \sigma)$, mit $\mu = 0$ und $\sigma = 1$. Berechnen wir nun $\mu_{\bar{x}}$ und $\sigma_{\bar{x}}$ für Stichproben mit dem Stichprobenumfang $n = 5, 15, 30, 50$.

Es sei daran erinnert, dass für eine hinreichend große Anzahl wiederholter Stichproben $\mu_{\bar{x}} \approx \mu$. Somit $\mu_{\bar{x}}$ der verschiedenen betrachteten Stichprobenverteilungen :

$$\mu_{\bar{x}_{n=5}} = \mu_{\bar{x}_{n=15}} = \mu_{\bar{x}_{n=30}} = \mu_{\bar{x}_{n=50}} = \mu = 0$$

Wir erinnern uns an den Standardfehler der Stichprobenverteilung $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$. Wir können also $\sigma_{\bar{x}}$ für $n = 5, 15, 30, 50$ Elemente leicht berechnen. Die verschiedenen Stichprobenverteilungen werden im Folgenden visualisiert.

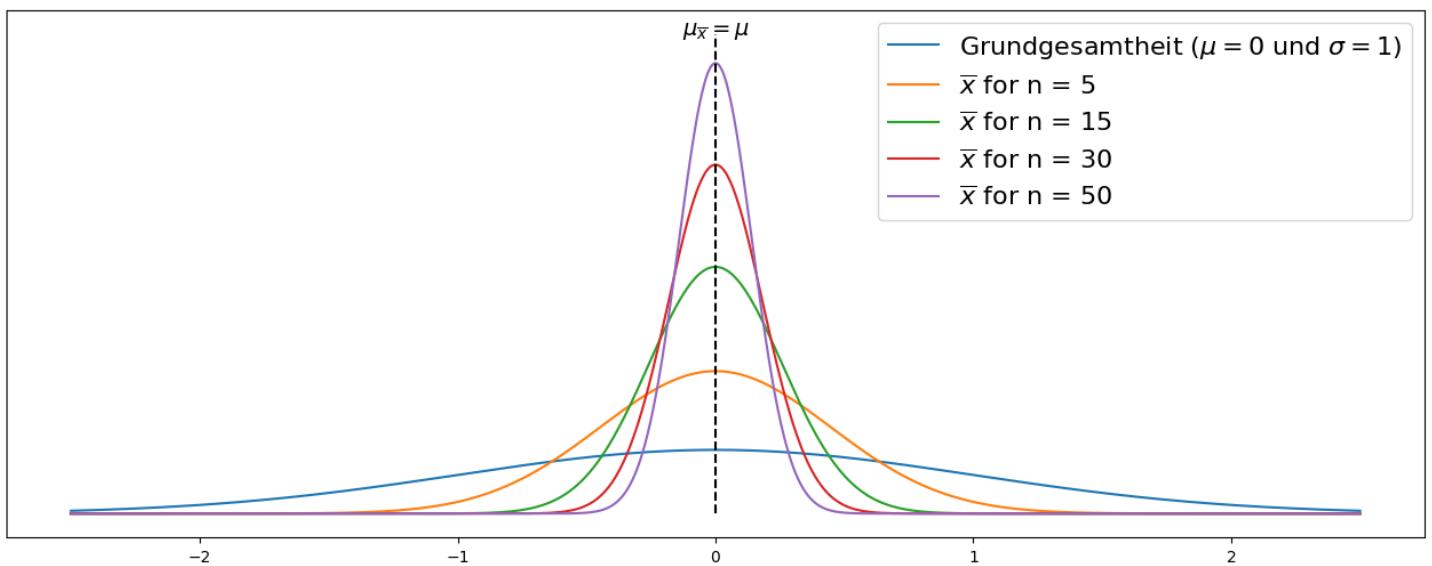
$$\sigma_{\bar{x}_{n=5}} = \frac{\sigma}{\sqrt{n}} = \frac{1}{\sqrt{5}} \approx 0,447$$

$$\sigma_{\bar{x}_{n=15}} = \frac{\sigma}{\sqrt{n}} = \frac{1}{\sqrt{15}} \approx 0,258$$

$$\sigma_{\bar{x}_{n=30}} = \frac{\sigma}{\sqrt{n}} = \frac{1}{\sqrt{30}} \approx 0,183$$

$$\sigma_{\bar{x}_{n=50}} = \frac{\sigma}{\sqrt{n}} = \frac{1}{\sqrt{50}} \approx 0,141$$

► Show code cell source

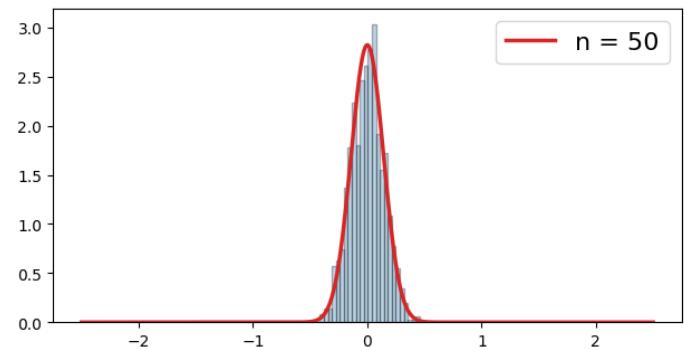
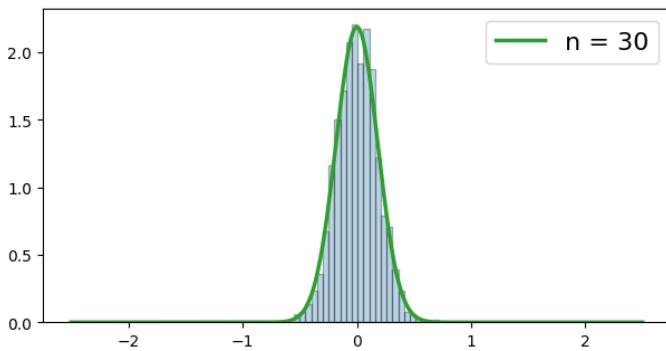
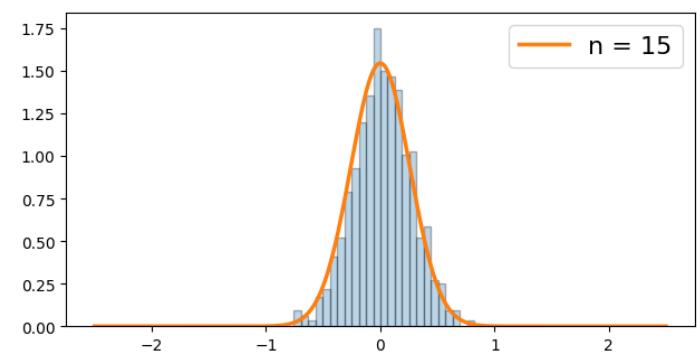
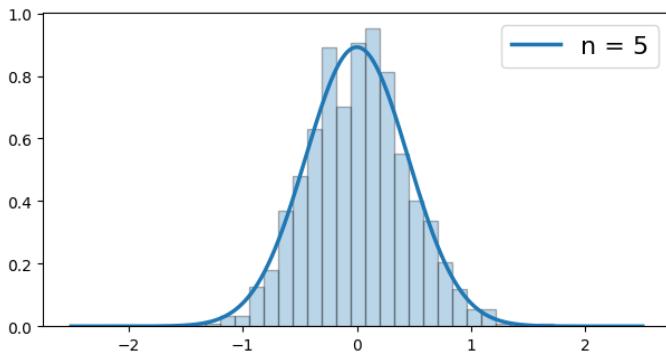


Es gibt zwei wichtige Beobachtungen bezüglich der Stichprobenverteilung von \bar{x}

1. Die Streuung der Stichprobenverteilung ist kleiner als die Streuung der entsprechenden Grundgesamtheitsverteilung. Mit anderen Worten: $\sigma_{\bar{x}} < \sigma$
2. Die Standardabweichung der Stichprobenverteilung nimmt mit zunehmendem Stichprobenumfang ab.

Um die 3. Behauptung von oben, dass die Form der Stichprobenverteilung von \bar{x} unabhängig vom Wert von n normal ist, zu überprüfen, führen wir eine numerische Simulation durch. Für eine ausreichend große Anzahl von Versuchen (Versuche = 1000) ziehen wir Stichproben aus der Standardnormalverteilung $N \sim (\mu = 0, \sigma = 1)$, wobei jede einzelne Stichprobe einen Stichprobenumfang von $n = 5, 15, 30, 50$ hat. Für jede Stichprobe berechnen wir den Stichprobenmittelwert \bar{x} und stellen die empirischen Wahrscheinlichkeiten dar. Anschließend vergleichen wir die empirische Verteilung dieser Wahrscheinlichkeiten mit den aus den obigen Gleichungen berechneten Stichprobenverteilungen.

► Show code cell source



Die Abbildung verifiziert die 3. Behauptung von oben: Die Form der Stichprobenverteilung von \bar{x} ist für jeden Wert von n normal.

Darüber hinaus zeigt die Abbildung, dass die Verteilung der empirischen Wahrscheinlichkeiten (Balken) gut mit der Stichprobenverteilung (farbige Linie) übereinstimmt und dass die

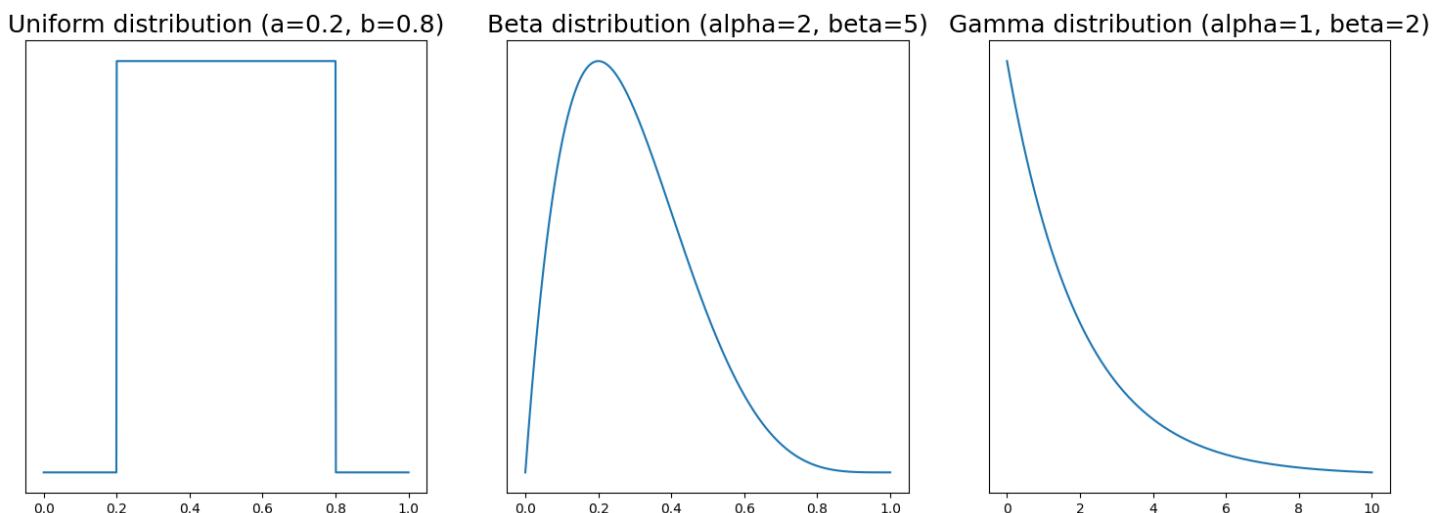
Standardabweichung der Stichprobenverteilung von \bar{x} mit zunehmendem Stichprobenumfang abnimmt. Es sei daran erinnert, dass die y -Achse die *Dichte* darstellt, d. h. die **Wahrscheinlichkeit pro Einheitswert** der Zufallsvariablen. Aus diesem Grund kann die Wahrscheinlichkeitsdichte einen Wert größer als 1 annehmen, aber nur über einen Bereich mit einer Größe kleiner als 1.

Stichproben aus einer nicht normalverteilten Grundgesamtheit

Im vorangegangenen Abschnitt haben wir die Form von Stichprobenverteilungen erörtert, wenn eine Stichprobe aus einer normalverteilten Grundgesamtheit gezogen wird. In realen Anwendungen kennen wir jedoch oft nicht die tatsächliche Form der Grundgesamtheit.

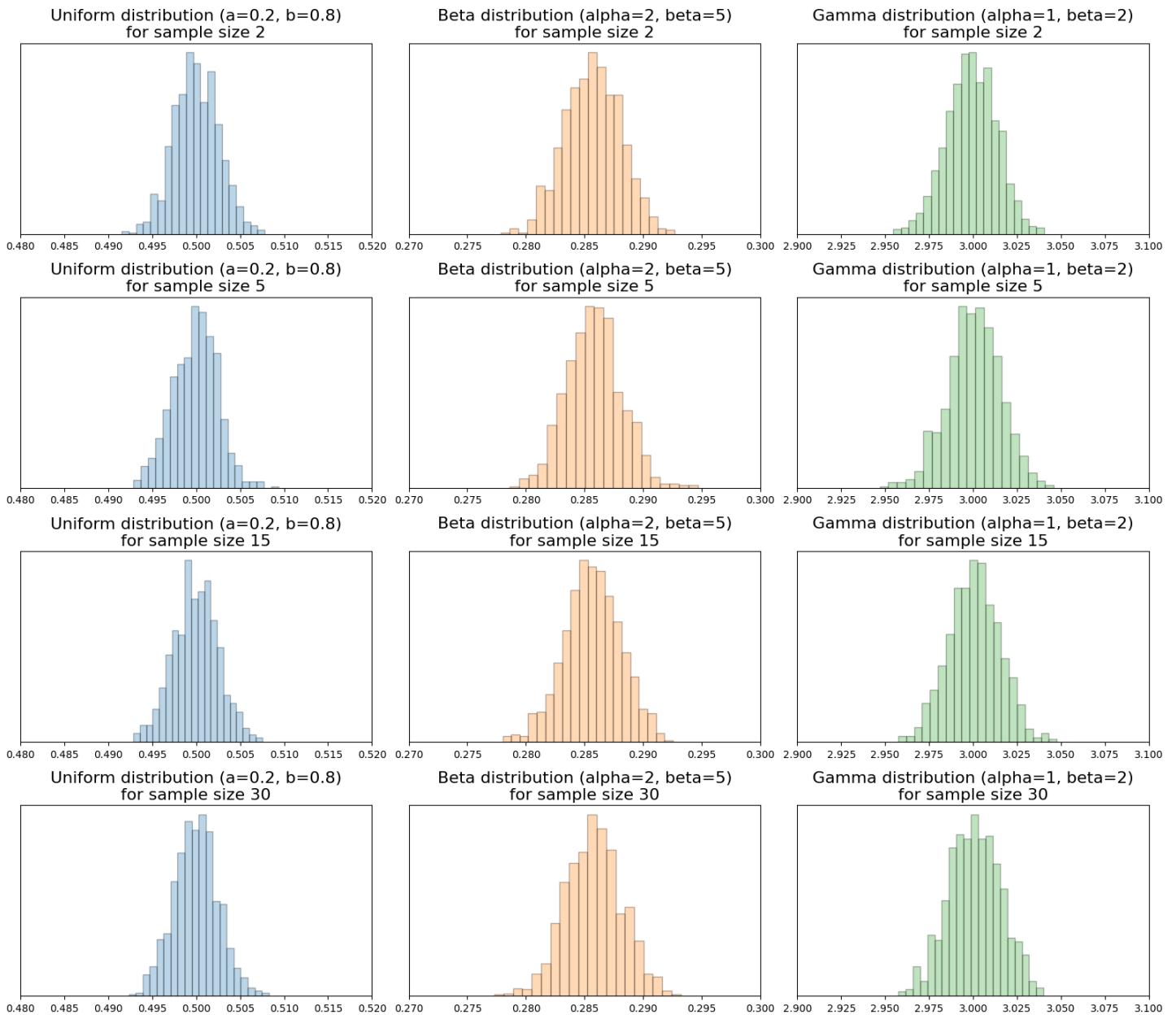
Um zu verstehen, wie die Form der Verteilung der interessierenden Grundgesamtheit die Form der Stichprobenverteilung beeinflusst, führen wir ein Experiment durch. Wir betrachten drei verschiedene kontinuierliche Wahrscheinlichkeitsdichtefunktionen: die [Gleichverteilung](#), die [Beta-Verteilung](#) und die [Gamma-Verteilung](#). Wir gehen hier nicht ins Detail, aber die folgende Abbildung zeigt, dass diese drei PDF's (Probability Density Functions) nicht normalverteilt sind.

► Show code cell source



Nun führen wir das gleiche Experiment wie im vorherigen Abschnitt durch. Für eine ausreichend große Anzahl von Versuchen (Versuche = 1000) ziehen wir aus jeder einzelnen Verteilung eine Stichprobe. Diesmal hat jedoch jede einzelne Stichprobe einen Stichprobenumfang $n = 2, 5, 15, 30$. Für jede Stichprobe berechnen wir den Stichprobenmittelwert \bar{x} und stellen die empirischen Wahrscheinlichkeiten nach 1000 Versuchen dar.

► Show code cell source



Die Abbildung zeigt, dass im Falle einer nicht normalverteilten Grundgesamtheit die Stichprobenverteilungen nicht normalverteilt sind, wenn $n < 30$. Allerdings nähern sich die Stichprobenverteilungen einer Normalverteilung an, wenn $n > 30$. Man sieht auch, dass die Streuung der Stichprobenverteilung mit zunehmendem Stichprobenumfang abnimmt.

Nach dem **zentralen Grenzwertsatz** ist die Stichprobenverteilung bei einem großen Stichprobenumfang ($n > 30$) annähernd normal, unabhängig von der Form der Grundgesamtheitsverteilung.

Der Mittelwert und die Standardabweichung der Stichprobenverteilung von \bar{x} sind jeweils,

$$\mu_{\bar{x}} = \mu \text{ und } \sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

Der Stichprobenumfang wird gewöhnlich als groß angesehen, wenn $n \geq 30$ ist.

Da die Stichprobenverteilung eine Normalverteilung approximiert, liefert die Fläche unter der Kurve der Stichprobenverteilung probabilistische Informationen über die Stichprobenstatistik.

Erinnern Sie sich an die **empirische Regel** (§ s.86), auch bekannt als die **68-95-99,7-Regel**. Auf die Stichprobenverteilung angewandt bedeutet die 68 – 95 – 99, 7-Regel folglich, dass

- etwa 68, 26% der Stichprobenmittelwerte innerhalb einer Standardabweichung des Populationsmittelwerts liegen werden,
- 95, 44% der Stichprobenmittelwerte innerhalb von zwei Standardabweichungen des Populationsmittelwertes liegen und
- etwa 99, 74% der Stichprobenmittelwerte innerhalb von drei Standardabweichungen des Mittelwerts der Grundgesamtheit liegen.

Übungsaufgaben

Der Zentrale Grenzwertsatz

Erklären Sie den Zentralen Grenzwertsatz.

Stichprobenfehler

Berechnen Sie aus der gegebenen Grundgesamtheit pop :

```
pop = [4, 10, 10, 17, 5, 1, 11, 3, 15, 8, 10, 2, 11, 10, 15, 5, 0, 14, 1, 12]
```

1. den Mittelwert der Grundgesamtheit
2. den Mittelwert und Stichprobenfehler einer Stichprobe mit Umfang $n = 10$

```
# Frage 1 ...
```

```
# Frage 2 ...
```

Lösungen

► Show code cell content

► Show code cell content

Würfelexperiment

1. Generieren Sie 100.000 (gleichwahrscheinliche) Würfe eines Würfels und berechnen Sie Mittelwert und Standardabweichung der Würfelsumme.
2. Wählen Sie aus den Würfelsummen 200 Stichproben mit einer Stichprobengröße von 50 aus. Berechnen Sie den Standardfehler mit

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

wobei $\sigma_{\bar{x}}$ als Standardfehler, σ als Standardabweichung der Stichprobe und \sqrt{n} als Wurzel aus der Stichprobengröße

3. Wiederholen Sie das Experiment für 10 Würfel

Hilfsfunktionen

```

import numpy as np

def dice_roll(nrolls: int, nsides: int = 6, seed=None) -> list:
    """Function to simulate a dice roll
    params:
        nrolls: number of rolls/dices
        nsides: number of sides
    """
    if seed is not None:
        np.random.seed(seed)

    return [np.random.randint(1, nsides + 1) for x in range(nrolls)]

```

Frage 1 ...

Frage 2 ...

Frage 3 ...

Lösungen

▶ Show code cell content

▶ Show code cell content

▶ Show code cell content

Test auf Normalverteilung

Generieren Sie in 100.000 (gleichwahrscheinliche) Würfe eines Würfels und berechnen Sie Mittelwert und Standardabweichung der Würfelsumme. Wählen Sie aus den Würfelwürfen 200 Stichproben mit unterschiedlichen Stichprobenumfängen. Ab welchem Stichprobenumfang (3, 5, 7, 10, 15, 20, 30, 50) können wir davon ausgehen, dass die Stichprobenverteilung des Mittelwertes normalverteilt ist. Nutzen sie zur Validierung der Hypothese den Wilk-Shapiro Test.

Hilfsfunktionen

```

import numpy as np
from scipy import stats

def test_for_normal_distribution(x, verbose=True):
    """Function to test if a sample is normally distributed.
    Therefore the Shapiro-Wilk test is employed. If the p-value is <0.05 we reject
    conclude that the data is not normally distributed for reference see
    https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html"""
    shapiro_test = stats.shapiro(x)
    pvalue = shapiro_test.pvalue
    if verbose:
        print(f"p-value: {pvalue}")
        if pvalue < 0.05:
            print(
                f"The null hypothesis is rejected, the data is NOT normally distributed"
            )
        else:
            print(
                f"Given the data the null hypothesis cannot be rejected, the data is"
            )
    return pvalue

def dice_roll(nrolls: int, nsides: int = 6, seed=None) -> list:
    """Function to simulate a dice roll
    params:
        nrolls: number of rolls/dices
        nsides: number of sides
    """
    if seed is not None:
        np.random.seed(seed)

    return [np.random.randint(1, nsides + 1) for x in range(nrolls)]

```

Frage 1 ...

► Show code cell content

Lernziele

Inferenzstatistik und Konfidenzintervalle

- Die Methodik der Inferenzstatistik wird theoretisch wie praktisch diskutiert
- Die Begriffe Konfidenzintervalle und Intervallschätzung und ihre Anwendung in der Schätzung von Grundgesamtheitsparametern wird veranschaulicht
- Die z -Verteilung und t -Verteilung werden sowohl theoretisch als auch durch Beispielen in Python beschrieben
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Inferenzstatistik, Konfidenzintervalle, Punkteschätzung, Intervall-Schätzung, z -Verteilung, t -Verteilung

Inferenzstatistik und Konfidenzintervalle

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm, t
```

Die [Inferenzstatistik](#) besteht aus Methoden, die Stichprobenergebnisse verwenden, um Entscheidungen oder Vorhersagen über eine Grundgesamtheit zu treffen ([]) s.12, [] s.237). Dieser Bereich der angewandten Statistik ist in allen Situationen von grundlegender Bedeutung, in denen das Wissen über die Grundgesamtheit begrenzt ist oder sogar gänzlich fehlt, was bei den meisten Anwendungen im wirklichen Leben der Fall ist. Außerdem ist es in vielen Fällen einfach zu teuer, sehr zeitaufwändig oder praktisch unmöglich, die Daten aller Mitglieder einer Grundgesamtheit zu erfassen. Daher wird eine Stichprobe aus der Grundgesamtheit gezogen und eine geeignete Stichprobenstatistik berechnet. Auf der Grundlage des Wertes der Stichprobenstatistik wird dann dem entsprechenden (unbekannten) Grundgesamtheitsparameter ein Wert zugewiesen.

Dieses Verfahren, bei dem Grundgesamtheitsparameter auf der Grundlage von Stichproben ein numerischer Wert zugewiesen wird, wird als [Schätzung](#) bezeichnet. Der numerische Wert wird als **Schätzung** des Grundgesamtheitsparameters bezeichnet. Die zur Schätzung eines Populationsparameters verwendete Stichprobenstatistik wird als **Schätzer** bezeichnet ([]) s.339).

Die Punkteschätzung

Bei einer Stichprobe ergibt der Wert der berechneten Stichprobenstatistik eine **Punktschätzung** (§ s.338) des entsprechenden Populationsparameters. Beispielsweise ist der Stichprobenmittelwert (\bar{x}) eine Punktschätzung des entsprechenden Populationsmittelwerts μ , oder die Stichprobenstandardabweichung s ist eine Punktschätzung für die Standardabweichung der Grundgesamtheit σ . Es ist jedoch zu beachten, dass jede zufällig ausgewählte Stichprobe aus einer Population voraussichtlich einen anderen Wert der Stichprobenstatistik ergibt. Mit anderen Worten, der Stichprobenmittelwert \bar{x} und die Stichprobenstandardabweichung s variieren von Stichprobe zu Stichprobe, während der Grundgesamtheitsmittelwert μ und die Grundgesamtheitsstandardabweichung σ fest sind. Folglich weicht der Punktschätzer fast immer vom wahren Wert der Grundgesamtheit ab. Daher sollte jeder Punktschätzung eine Information beigefügt werden, die die Genauigkeit dieser Schätzung angibt.

Die Intervall-Schätzung

Anstatt einem Grundgesamtheitsparameter einen einzelnen Wert zuzuordnen, gibt eine **Intervallschätzung** eine probabilistische Aussage, die das gegebene Intervall mit der Wahrscheinlichkeit in Beziehung setzt, dass dieses Intervall tatsächlich den wahren (unbekannten) Grundgesamtheitsparameter enthält.

Das **Konfidenzniveau** wird a priori gewählt und hängt somit von den Präferenzen des Nutzers ab. Es wird bezeichnet durch

$$100(1 - \alpha)\%$$

Obwohl jeder beliebige Wert für das Konfidenzniveau gewählt werden kann, sind die gebräuchlichsten Werte 90%, 95% und 99%. Wird das Konfidenzniveau als Wahrscheinlichkeit ausgedrückt, wird es als **Konfidenzkoeffizient** bezeichnet und mit $(1 - \alpha)$. Die gebräuchlichsten Vertrauenskoeffizienten sind 0, 90, 0, 95 bzw. 0, 99.

Ein $100(1 - \alpha)\%$ -iges Konfidenzintervall ist eine Intervallschätzung um einen Populationsparameter θ (hier ist der griechische Buchstabe θ ein Platzhalter für einen beliebigen Populationsparameter von Interesse, wie z. B. den Mittelwert μ oder die Standardabweichung σ),

von dem bei wiederholten Stichproben der Größe N erwartet wird, dass er den wahren Wert von θ in $100(1 - \alpha)\%$ der Fälle einschließt ([] s.358).

Die tatsächliche Zahl, die zur Punktschätzung addiert oder von ihr subtrahiert wird, wird als Fehlermarge bezeichnet.

$$CI : \text{Punktschätzung} \pm \text{Fehlermarge}$$

Die Fehlermarge besteht aus zwei Elementen. Zum einen aus dem so genannten **kritischen Wert** und zum anderen aus einem Maß für die Variabilität der [Stichprobenverteilung](#). Der kritische Wert ist ein numerischer Wert, der dem a priori festgelegten Vertrauensniveau entspricht. Er wird manchmal als z^* oder $z_{\alpha/2}^*$ bezeichnet. Das Maß für die Variabilität ist der [Standardfehler](#), der als $\frac{\sigma}{\sqrt{n}}$ bezeichnet wird.

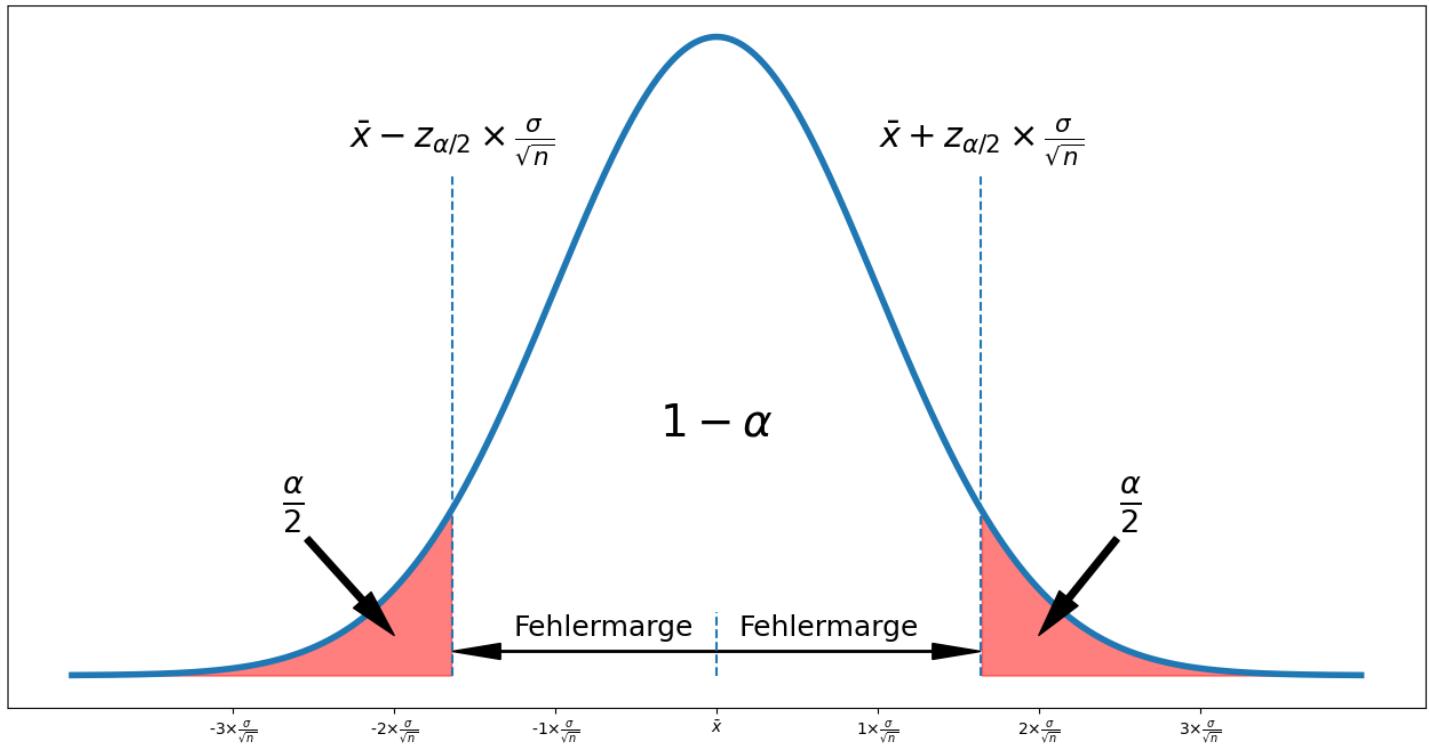
Die Fehlermarge (FM) wird also ausgedrückt als

$$FM = z_{\alpha/2}^* \times \frac{\sigma}{\sqrt{n}}$$

Schauen wir uns zum besseren Verständnis eine Abbildung an.

► Show code cell source

[]



Dementsprechend lautet die vollständige Gleichung für das Konfidenzintervall wie folgt

$$CI : \text{Punktschätzung} \pm z_{\alpha/2}^* \times \frac{\sigma}{\sqrt{n}}$$

Um den entsprechenden Wert für $z_{\alpha/2}^*$ zu erhalten, kann man ihn in einer [Tabelle](#) nachschlagen oder die Funktion `norm.ppf()` in Python verwenden. Lassen Sie uns zur Übung einige Konfidenzintervalle konstruieren.

Konfidenzniveau von 90% ($\alpha = 0,1$)

`norm.ppf(0.05)`

`np.float64(-1.6448536269514729)`

`norm.ppf(0.95)`

```
np.float64(1.6448536269514722)
```

Die untere und obere Grenze des Intervalls, das einen Bereich von 90% um den Mittelwert abdeckt, wird durch die z -Werte $-1,64$ bzw. $1,64$ angegeben.

Bei einem Konfidenzniveau von 90% ($\alpha = 0,1$) ergibt sich aus der obigen Gleichung

$$CI_{90\%} : \text{Punktschätzung} \pm 1,64 \times \frac{\sigma}{\sqrt{n}}$$

Konfidenzniveau von 95% ($\alpha = 0,05$)

```
norm.ppf(0.025)
```

```
np.float64(-1.9599639845400545)
```

```
norm.ppf(0.975)
```

```
np.float64(1.959963984540054)
```

Die untere und obere Grenze des Intervalls, das einen Bereich von 90% um den Mittelwert abdeckt, wird durch die z -Werte $-1,96$ bzw. $1,96$ angegeben.

Bei einem Konfidenzniveau von 95% ($\alpha = 0,05$) ergibt sich aus der obigen Gleichung

$$CI_{95\%} : \text{Punktschätzung} \pm 1,96 \times \frac{\sigma}{\sqrt{n}}$$

Konfidenzniveau von 99% ($\alpha = 0,01$)

```
norm.ppf(0.005)
```

```
np.float64(-2.575829303548901)
```

```
norm.ppf(0.995)
```

```
np.float64(2.5758293035489004)
```

Die untere und obere Grenze des Intervalls, das einen Bereich von 90% um den Mittelwert abdeckt, wird durch die z -Werte $-2,58$ bzw. $2,58$ angegeben.

Bei einem Konfidenzniveau von 99% ($\alpha = 0,01$) ergibt sich aus der obigen Gleichung

$$CI_{99\%} : \text{Punktschätzung} \pm 2,58 \times \frac{\sigma}{\sqrt{n}}$$

Die Breite eines Konfidenzintervalls und der Kompromiss zwischen Präzision und Genauigkeit

Ein Gedankenexperiment

Betrachten wir die durchschnittliche Tagestemperatur in Berlin an einem schönen Sommertag im Juni. Nehmen wir an, wir messen eine mittlere Tagestemperatur von 19°C . Nun geht es darum, die mittlere Temperatur von morgen zu schätzen.

Es gibt natürlich mehrere mehr oder weniger ausgeklügelte Ansätze, um diese Aufgabe zu lösen. Man kann in den Datenarchiven nachsehen und die Schätzung auf deskriptive Statistiken stützen, oder man kann sogar einen sehr ausgeklügelten Modellierungsansatz mit einer Unmenge von Modellparametern anwenden, oder man kann einfach eine Schätzung auf der Grundlage des gesunden Menschenverstands vornehmen. Unabhängig davon, welcher Ansatz gewählt wird, wird das Ergebnis immer eine Schätzung sein, die mit einem gewissen Grad an Unsicherheit verbunden ist.

Um dieser Unsicherheit Rechnung zu tragen, präsentieren wir keine Punktschätzung der morgigen Temperatur, sondern eine Intervallschätzung. Um eine hohe **Genauigkeit** unserer Schätzung zu erreichen, also, dass wir sehr sicher sein wollen, dass unser Intervall den tatsächlichen Wert enthält, wenden wir eine sehr große Fehlermarge an. Wir sagen zum Beispiel, dass die morgige Temperatur $19 \pm 20^\circ\text{C}$ beträgt. Nach gesundem Menschenverstand würden wir wahrscheinlich zustimmen, dass die durchschnittliche Tagestemperatur an einem Sommertag im Juni in Berlin zwischen -1 und 39°C liegt.

Obwohl das Intervall sehr groß ist und vielleicht sogar alle mittleren Tagestemperaturen im Juni für Berlin in der Geschichte der Wetterbeobachtung enthält, besteht immer noch eine kleine Chance, dass wir uns irren. Stellen Sie sich ein natürliches oder vom Menschen verursachtes katastrophales Ereignis wie einen gewaltigen Vulkanausbruch, einen Asteroideneinschlag oder einen Atomkrieg vor; in diesen glücklicherweise sehr unwahrscheinlichen Fällen kann selbst eine so große Fehlermarge nicht garantieren, dass die morgige Durchschnittstemperatur innerhalb des angegebenen Intervalls liegt. Dennoch ist es wichtig zu beachten, dass wir, **um eine hohe Genauigkeit zu erreichen, das Konfidenzniveau und damit die Breite des Konfidenzintervalls erhöhen**.

OK, wir sagen, dass die morgige Temperatur im Bereich von $19 \pm 20^\circ\text{C}$ liegt. Aber ist eine solche Aussage, obwohl sie sehr genau ist, überhaupt von Wert? Hilft uns diese Schätzung bei der Entscheidung, welche Kleidung wir morgen tragen sollen? Nein, ganz und gar nicht!

In vielen Anwendungen sind wir also nicht nur an der **Genauigkeit** interessiert, sondern auch an der **Präzision** einer Schätzung. Eine Schätzung der morgigen Temperatur mit höherer Genauigkeit wäre $19 \pm 2^\circ\text{C}$. Eine solche Vorhersage hilft uns zwar bei der Entscheidung, welche Kleidung wir anziehen sollen, aber die Wahrscheinlichkeit, dass wir uns irren, ist viel größer. Daher ist es wichtig, sich zu vergegenwärtigen, dass eine **Erhöhung der Genauigkeit die Breite des Konfidenzintervalls verkleinert und somit das Konfidenzniveau verringert**. Diese Abwägung zwischen Genauigkeit und Präzision bei der Auswahl eines geeigneten Konfidenzniveaus ist in der Praxis sehr wichtig.

Schätzung des Mittelwerts einer Grundgesamtheit - Die z -Verteilung

Die Schätzung des Mittelwerts einer Grundgesamtheit anhand einer Stichprobe ist eine sehr häufige Aufgabe. Wenn die Standardabweichung der Grundgesamtheit (σ) bekannt ist, basiert die Konstruktion eines Konfidenzintervalls für den Grundgesamtheitsmittelwert (μ) auf der

normalverteilten Stichprobenverteilung der Stichprobenmittelwerte (gewährleistet durch den [zentralen Grenzwertsatz](#)). Wenn die Grundgesamtheit, aus der die Stichprobe gezogen wird, nicht normalverteilt ist, sollte der Stichprobenumfang $n > 30$ sein.

Das $100(1 - \alpha)\%$ ige Konfidenzintervall für μ ist gegeben durch

$$CI : \bar{x} \pm z_{\alpha/2}^* \times \sigma_{\bar{x}}$$

wobei $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$

Der Wert von $z_{\alpha/2}^*$ entspricht dem kritischen Wert und wird aus der [Standardnormaltafel](#) entnommen oder mit der Funktion `norm.ppf()` in Python berechnet. Der kritische Wert ist eine Größe, die mit dem gewünschten Konfidenzniveau zusammenhängt. Typische Werte für $z_{\alpha/2}^*$ sind 1, 64, 1, 96 und 2, 58, was einem Konfidenzniveau von 90%, 95% und 99% entspricht. Dieser kritische Wert wird mit dem Standardfehler ($\sigma_{\bar{x}}$) multipliziert, um die Fehlermarge zu vergrößern oder zu verkleinern.

Der Standardfehler ($\sigma_{\bar{x}}$) ergibt sich aus dem Verhältnis zwischen der Standardabweichung der Grundgesamtheit (σ) und der Quadratwurzel des Stichprobenumfangs n . Er beschreibt das Ausmaß, in dem die berechnete Stichprobenstatistik von einer Stichprobe zur anderen abweichen kann. Das Produkt aus dem kritischen Wert und dem Standardfehler wird als Fehlermarge bezeichnet. Es ist die Größe, die vom Wert von \bar{x} subtrahiert und zu diesem addiert wird, um das Konfidenzintervall für μ zu erhalten.

Intervallschätzung für einen Mittelwert - Die z -Verteilung

Zum besseren Verständnis der Schätzung eines Populationsmittelwerts und der Konstruktion eines Konfidenzintervalls diskutieren wir das Verfahren anhand eines Datensatzes. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Zunächst importieren wir den Datensatz und geben ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein
students = pd.read_csv("../data/students.csv")
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

In diesem Abschnitt konzentrieren wir uns auf die Größe von Studentinnen. Wir gehen davon aus, dass die im Datensatz angegebenen Größenmessungen eine sehr gute Annäherung an die interessierende Population darstellen, nämlich die Größe der Studentinnen in cm. Die Variable `height` der Studentinnen ist annähernd normalverteilt - was durch die symmetrische Glockenform des Populationshistogramms bestätigt wird.

```
female = students.loc[students["gender"] == "Female", "height"]
```

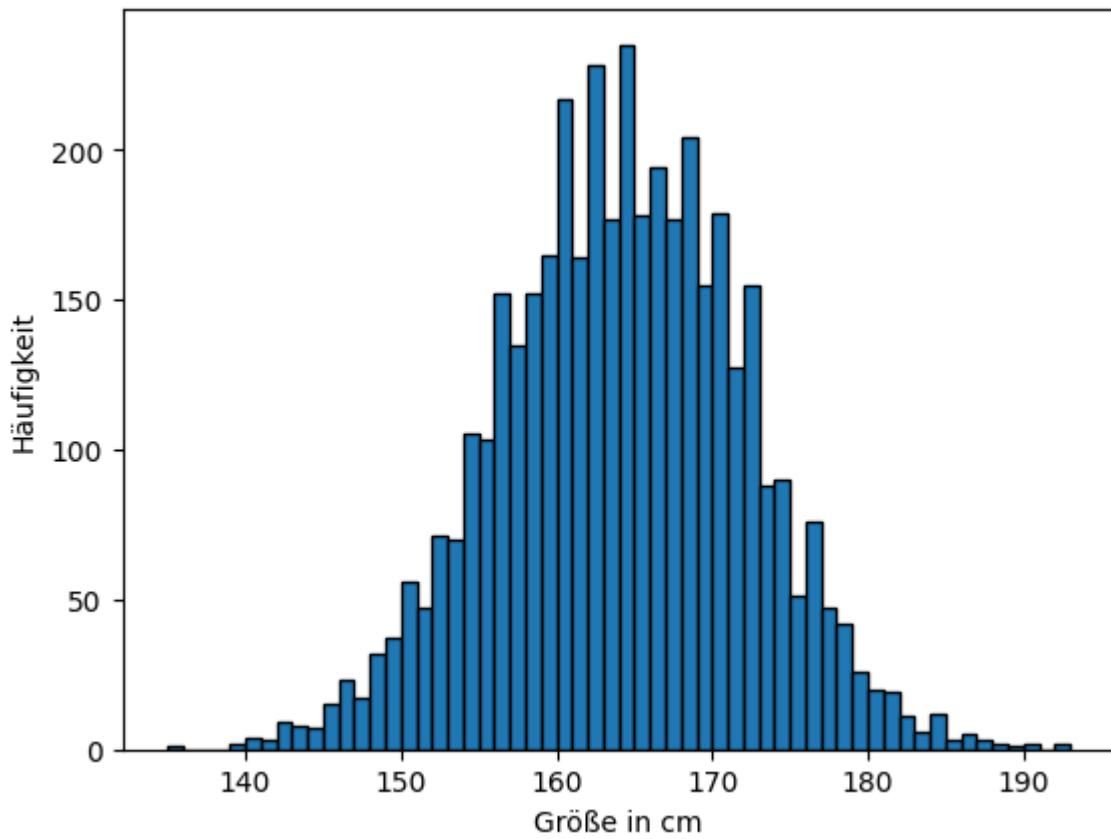
```
# Bestimme Anzahl Bins
bins = max(female) - min(female)

# Plotte die Werte als Histogramm
fig, ax = plt.subplots()

ax.hist(female, bins, edgecolor="k")

# Erzeuge Labels
ax.set_ylabel("Häufigkeit")
ax.set_xlabel("Größe in cm")
```

```
Text(0.5, 0, 'Größe in cm')
```



Auf der Grundlage der gegebenen Daten berechnen wir zunächst den Populationsmittelwert μ und die Populationsstandardabweichung σ .

```
f_mean = np.mean(female)  
f_mean
```

```
np.float64(163.65328467153284)
```

```
f_std = np.std(female)  
f_std
```

```
np.float64(7.918762263149209)
```

Anschließend konstruieren wir eine Wahrscheinlichkeitsverteilung, indem wir die Funktion `norm.pdf()` anwenden, die durch die zuvor berechneten Parameter μ und σ definiert ist, und

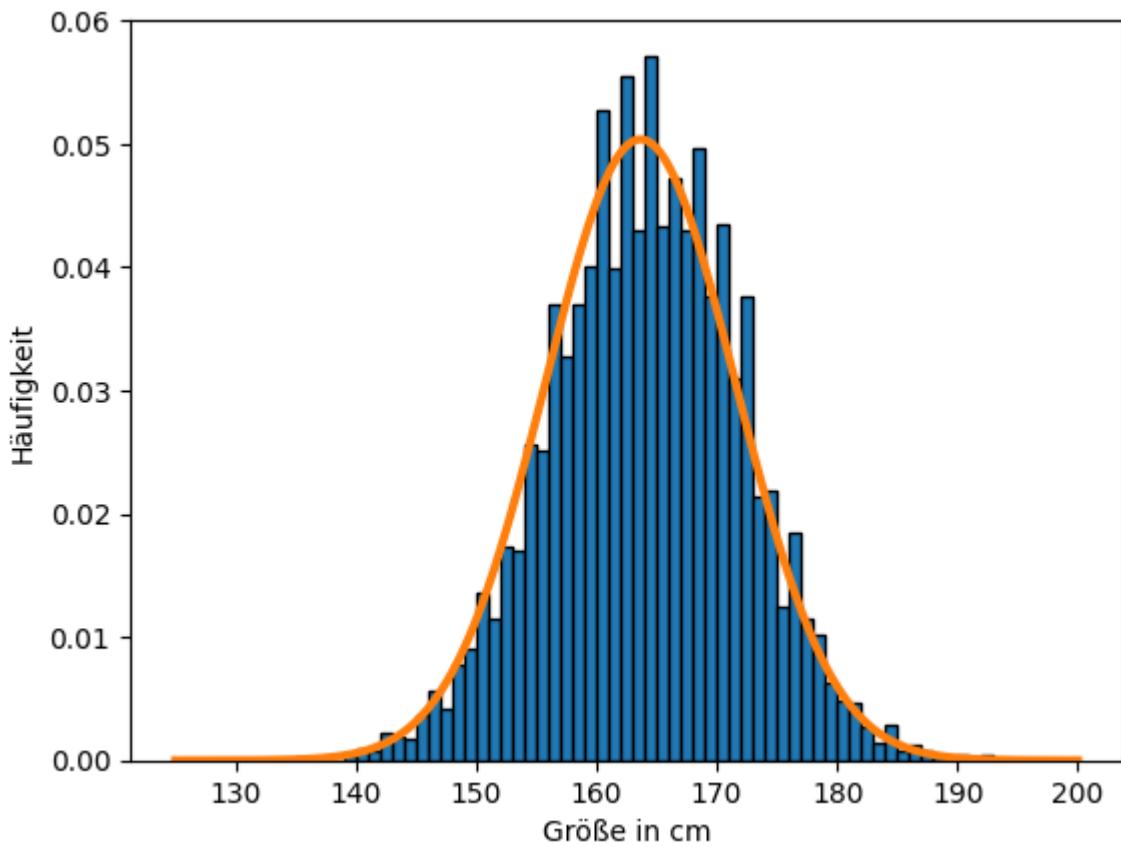
stellen sie über dem Histogramm dar.

```
x = np.linspace(125, 200, 500)
y = norm.pdf(x, f_mean, f_std)

# Bestimme Anzahl Bins
bins = max(female) - min(female)

# Plotte die Werte als Histogramm
fig, ax = plt.subplots()
ax.hist(female, bins=bins, edgecolor="k", density=True)
ax.plot(x, y, linewidth=3)
# Erzeuge Labels
ax.set_ylabel("Häufigkeit")
ax.set_xlabel("Größe in cm")
```

```
Text(0.5, 0, 'Größe in cm')
```



Eine gute Übereinstimmung!

Nun nehmen wir eine Zufallsstichprobe mit einem Stichprobenumfang von $n = 10$ aus der Wahrscheinlichkeitsverteilung, indem wir die Funktion `norm.rvs()` anwenden und den Stichprobenmittelwert \bar{x} berechnen.

```
# Grösse der Stichprobe
sample_size = 10

# Wähle normalverteilte Zufallszahlen mit Mittelwert f_mean und Standardabweichung
sample = norm.rvs(f_mean, f_std, sample_size, random_state=1)
sample
```

```
array([176.51608944, 158.80893107, 159.47081813, 155.15670124,
       170.50624195, 145.42794689, 177.47003423, 157.62546819,
       166.17967943, 161.67857995])
```

```
sample_mean = np.mean(sample)
sample_mean
```

```
np.float64(162.88404905120882)
```

Unsere Stichprobe ergibt einen Stichprobenmittelwert \bar{x} von etwa 162,884. Dies ist unser Punktschätzer für den interessierenden Grundgesamtheitsparameter, der in diesem Fall die durchschnittliche Körpergröße der Studentinnen (μ).

Wie genau ist unsere Punktschätzung? Wir fragen Python, ob unsere Schätzung mit dem wahren Grundgesamtheitsparameter übereinstimmt.

```
sample_mean == f_mean
```

```
np.False_
```

OK, das haben wir erwartet!

Berechnen wir nun einige Intervallschätzungen, indem wir die 90%-, 95%- und 99%-Konfidenzintervalle konstruieren. Erinnern Sie sich an die Gleichung für ein Konfidenzintervall.

$$CI : \text{Punktschätzung} \pm z_{\alpha/2}^* \times \frac{\sigma}{\sqrt{n}}$$

Der kritische Wert $z_{\alpha/2}^*$ beträgt 1, 64, 1, 96 und 2, 58 für Konfidenzniveaus von 90%, 95% bzw. 99%.

Angewandt auf unsere Daten ergibt die Gleichung

$$CI_{90\%} : 162,884 \pm 1,64 \times \frac{7,92}{\sqrt{10}} = 162,884 \pm 4,12$$

Wir können also mit 90%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Studenten (der Grundgesamtheitsparameter μ) zwischen 158, 764 und 167, 004 cm liegt.

$$CI_{95\%} : 162,884 \pm 1,96 \times \frac{7,92}{\sqrt{10}} = 162,884 \pm 4,91$$

Wir können also mit 95%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Studenten zwischen 157, 974 und 167, 794 cm liegt.

$$CI_{99\%} : 162,884 \pm 2,58 \times \frac{7,92}{\sqrt{10}} = 162,884 \pm 6,45$$

Wir können also mit 99%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Studenten zwischen 156, 434 und 169, 334 cm liegt.

Es liegt auf der Hand, **dass die Fehlerspanne größer wird, wenn man eine höhere Sicherheit haben will, dass der unbekannte Grundgesamtheitsparameter im Intervall enthalten ist.**

Zur Überprüfung der Richtigkeit wollen wir untersuchen, ob wir mit unseren Intervallschätzungen tatsächlich den wahren Wert der Grundgesamtheit erfasst haben. Es ist wichtig, sich daran zu erinnern, dass das Konfidenzintervall unserem Stichprobenmittelwert keine Wahrscheinlichkeit zuweist, sondern besagt, dass das Konfidenzintervall bei wiederholten Zufallsstichproben den Mittelwert der Grundgesamtheit in $100(1 - \alpha)\%$ der Fälle enthalten soll. Um diese Behauptung zu testen, schreiben wir selbst eine einfache Python-Funktion.

```

def CI_eval(pop_mean, sigma, n, estimate, alpha):
    """Funktion zur Evaluierung des Konfidenzintervalls"""
    out = {f"{int((1 - x) * 100)}%": {} for x in alpha}

    for e, _alpha in enumerate(alpha):
        key = f"{int((1 - _alpha) * 100)}%"
        out[key]["true value"] = pop_mean
        out[key]["estimate"] = estimate
        upper = estimate + norm.ppf(1 - _alpha / 2) * sigma / np.sqrt(n)
        out[key]["upper"] = upper
        lower = estimate - norm.ppf(1 - _alpha / 2) * sigma / np.sqrt(n)
        out[key]["lower"] = lower
        out[key]["test"] = pop_mean >= lower and pop_mean <= upper

    return out

```

Wenden wir nun unsere selbst erstellte Funktion `CI_eval()` auf unsere Daten an. Wir setzen `pop_mean = f_mean`, `sigma = f_std`, `n = sample_size`, `estimate = sample_mean`, `alpha = [0.1, 0.05, 0.01]`, um zu evaluieren, ob die drei oben konstruierten Konfidenzintervalle (90%, 95% und 99%) den Grundgesamtheitsmittelwert enthalten. Schließlich wandeln wir den resultierenden Vektor in ein `dataframe`-Objekt um, um die Lesbarkeit zu verbessern.

```

df = pd.DataFrame.from_dict(
    CI_eval(
        pop_mean=f_mean,
        sigma=f_std,
        n=sample_size,
        estimate=sample_mean,
        alpha=[0.1, 0.05, 0.01],
    )
)
df

```

	90%	95%	99%
true value	163.653285	163.653285	163.653285
estimate	162.884049	162.884049	162.884049
upper	167.00298	167.792059	169.334267
lower	158.765118	157.97604	156.433831
test	True	True	True

Ein interessantes Ergebnis. Der wahre Mittelwert der Grundgesamtheit ($f_mean = 163,65$) wird von allen drei Konfidenzintervallen erfasst. Der Mittelwert unserer Zufallsstichprobe ($sample_mean = 162,884$) war ein recht guter Schätzer.

Schätzung des Mittelwerts einer Grundgesamtheit - Die t -Verteilung

Bisher haben wir uns auf σ , die Standardabweichung der Grundgesamtheit, gestützt, um auf den Mittelwert der Grundgesamtheit zu schließen. Der Populationsparameter σ wird zur Berechnung des Standardfehlers ($SF = \frac{\sigma}{\sqrt{n}}$) verwendet, der ein Bestandteil der Fehlermarge ist. Was aber, wenn man die Standardabweichung der Grundgesamtheit nicht kennt, was in der Regel der Fall ist? Man kann die Standardabweichung der Stichprobe, die mit s bezeichnet wird, als Schätzwert für die Standardabweichung der Grundgesamtheit verwenden.

$$\text{wenn } s \approx \sigma \text{ dann } SF = \frac{s}{\sqrt{n}}$$

Es ist jedoch zu beachten, dass im Gegensatz zu σ die Standardabweichung der Stichprobe, s , von Stichprobe zu Stichprobe variiert und dass $s < \sigma$ ist. Man kann den Stichprobenumfang n erhöhen, und damit wird s zu einer besseren Schätzung für σ . Solange wir jedoch σ nicht kennen, müssen wir bei der Durchführung des Inferenzverfahrens in jedem Fall zwei Größen schätzen: sowohl den Mittelwert μ als auch die Standardabweichung σ . Aus diesem Grund führt die Verwendung von s als Schätzung für σ zu einer größeren Unsicherheit bei der Schätzung des Mittelwerts μ . Um dieser zusätzlichen Unsicherheit entgegenzuwirken, wenden wir die so genannte [\$t\$ -Verteilung](#) oder [Studentsche-Verteilung](#) an, um die Fehlermarge (FM) zu berechnen.

Das Verfahren zur Ermittlung eines Konfidenzintervalls für einen Grundgesamtheitsmittelwert, wenn die Standardabweichung σ der Grundgesamtheit nicht bekannt ist, ist im Wesentlichen dasselbe wie bei bekannter Standardabweichung der Grundgesamtheit, mit der Ausnahme, dass jetzt die t -Verteilung und die Standardabweichung s der Stichprobe anstelle der Standardnormalverteilung (z -Scores) bzw. der Standardabweichung σ der Grundgesamtheit herangezogen werden.

Erinnern Sie sich an die Konstruktion eines Konfidenzintervalls

$$CI : \text{Punktschätzung} \pm FM$$

Die Fehlermarge (FM) besteht aus dem kritischen Wert und einem Maß für die Variabilität der Stichprobenverteilung. Der kritische Wert ist $t_{df, \alpha/2}^*$ für das gegebene Konfidenzniveau und die Freiheitsgrade. Sein Wert ergibt sich aus einer t -Verteilungstabelle für $n - 1$ Freiheitsgrade oder wird in Python mit der Funktion `t.cdf()` berechnet. Das Maß für die Variabilität der Stichprobenverteilung ist der Standardfehler (SF). Da die Standardabweichung σ der Grundgesamtheit nicht bekannt ist, wird sie durch die Standardabweichung s der Stichprobe ersetzt, was zu $SF = \frac{s}{\sqrt{n}}$ führt.

Folglich ist das $100(1 - \alpha)\%$ ige Konfidenzintervall für μ gleich

$$CI : \bar{x} \pm t_{df, \alpha/2}^* \frac{s}{\sqrt{n}}$$

Lassen Sie uns zur Übung einige Konfidenzintervalle konstruieren. Für den Zweck dieser Übung wird df auf 12 gesetzt.

Konfidenzniveau von 90% ($\alpha = 0,1$)

```
t.ppf(0.05, df=12)
```

```
np.float64(-1.7822875556491593)
```

```
t.ppf(0.95, df=12)
```

```
np.float64(1.782287555649159)
```

Die untere und obere Grenze des Intervalls (bei $df = 12$), das einen Bereich von 90% um den Mittelwert abdeckt, entspricht den t -Werten $-1,78$ und $1,78$

Bei einem Konfidenzniveau von 90% ($\alpha = 0,1$) ergibt sich aus der obigen Gleichung

$$CI_{90\%} : \text{Punktschätzung} \pm 1,78 \times \frac{s}{\sqrt{n}}$$

Konfidenzniveau von 95% ($\alpha = 0,05$)

```
t.ppf(0.025, df=12)
```

```
np.float64(-2.178812829663418)
```

```
t.ppf(0.975, df=12)
```

```
np.float64(2.1788128296634177)
```

Die untere und obere Grenze des Intervalls (bei $df = 12$), das einen Bereich von 95% um den Mittelwert abdeckt, entspricht den t -Werten $-2,18$ und $2,18$

Bei einem Konfidenzniveau von 95% ($\alpha = 0,05$) ergibt sich aus der obigen Gleichung

$$CI_{95\%} : \text{Punktschätzung} \pm 2,18 \times \frac{s}{\sqrt{n}}$$

Konfidenzniveau von 99% ($\alpha = 0,01$)

```
t.ppf(0.005, df=12)
```

```
np.float64(-3.054539589392901)
```

```
t.ppf(0.995, df=12)
```

```
np.float64(3.0545395893929017)
```

Die untere und obere Grenze des Intervalls (bei $df = 12$), das einen Bereich von 99% um den Mittelwert abdeckt, entspricht den t -Werten $-3,05$ und $3,05$

Bei einem Konfidenzniveau von 99% ($\alpha = 0,01$) ergibt sich aus der obigen Gleichung

$$CI_{99\%} : \text{Punktschätzung} \pm 3,05 \times \frac{s}{\sqrt{n}}$$

Intervallschätzung für einen Mittelwert - Die *t*-Verteilung

Wir üben das Verfahren "The One-Mean t-Interval Procedure", um Konfidenzintervalle mit einem Datensatz zu konstruieren. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Zunächst importieren wir den Datensatz und geben ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id, name, gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary`.

In diesem Abschnitt konzentrieren wir uns wieder auf die Körpergröße von Studentinnen. Gehen wir davon aus, dass die im Datensatz angegebenen Größenmessungen eine sehr gute Annäherung an die interessierende Grundgesamtheit darstellen, so dass wir eine Stichprobe aus der Wahrscheinlichkeitsdichtefunktion auf der Grundlage des Mittelwerts und der Standardabweichung des Datensatzes ziehen.

```
female = students.loc[students["gender"] == "Female", "height"]
f_mean = np.mean(female)
f_std = np.std(female)
```

```
f_mean
```

```
np.float64(163.65328467153284)
```

```
f_std
```

```
np.float64(7.918762263149209)
```

Wir ziehen eine Zufallsstichprobe mit einem Stichprobenumfang von $n = 6$ aus der Wahrscheinlichkeitsverteilung, die durch den Mittelwert und die Standardabweichung der Höhenvariablen definiert ist, und berechnen den Stichprobenmittelwert \bar{x} und die Stichprobenstandardabweichung s .

```
sample_size = 6
sample = t.rvs(sample_size - 1, f_mean, f_std, sample_size, random_state=1)
sample_mean = np.mean(sample)
sample_mean
```

```
np.float64(161.44236164394297)
```

```
s = np.std(sample)
s
```

```
np.float64(10.09685230523463)
```

Unsere Zufallsstichprobe ergibt einen Stichprobenmittelwert \bar{x} von etwa 161,44 und eine Stichprobenstandardabweichung s von ungefähr 10,1. Dies sind unsere Punktschätzungen für die interessierende Grundgesamtheit, die in diesem Fall die Körpergröße der Studentinnen in unserem Datensatz ist.

Wie genau ist unsere Punktschätzung? Wir fragen Python, ob unsere Schätzungen mit dem wahren Grundgesamtheitsparameter übereinstimmen.

```
sample_mean == f_mean
```

```
np.False_
```

```
s == f_std
```

```
np.False_
```

```
s < f_std
```

```
np.False_
```

OK, das haben wir erwartet!

Berechnen wir nun die Intervallschätzungen, indem wir die 90%-, 95%- und 99%-Konfidenzintervalle konstruieren. Erinnern Sie sich an die Gleichung für ein Konfidenzintervall und wie man den Freiheitsgrad berechnet.

$$CI : \text{Punktschätzung} \pm t_{df, \alpha/2}^* \times \frac{s}{\sqrt{n}}$$

$$df = n - 1$$

Der kritische Wert $t_{5, \alpha/2}^*$ beträgt 2,02, 2,57 und 4,03 für Konfidenzniveaus von 90%, 95% bzw. 99%.

Angewandt auf unsere Höhendaten ergibt die Gleichung

$$CI_{90\%} : 161,44 \pm 2,02 \times \frac{10,1}{\sqrt{6}} = 161,44 \pm 8,32$$

Wir können also mit 90%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Studentinnen (der Populationsparameter μ) zwischen 153,12 und 169,76 cm liegt.

$$CI_{95\%} : 161,44 \pm 2,57 \times \frac{10,1}{\sqrt{6}} = 161,44 \pm 10,6$$

Wir können also mit 95%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Schülerinnen zwischen 150,84 und 172,04 cm liegt.

$$CI_{99\%} : 161,44 \pm 4,03 \times \frac{10,1}{\sqrt{6}} = 161,44 \pm 16,6$$

Wir können also mit 99%iger Sicherheit sagen, dass die durchschnittliche Körpergröße der Studentinnen zwischen 144,84 und 178,04 cm liegt.

Es liegt auf der Hand, **dass die Fehlerspanne größer ist, wenn wir eine höhere Sicherheit haben wollen, dass der unbekannte Grundgesamtheitsparameter im Intervall enthalten ist.**

Zur Überprüfung der Korrektheit untersuchen wir, ob wir mit unserer Intervallschätzung tatsächlich den wahren Grundgesamtheitswert erfasst haben. Es ist wichtig, sich daran zu erinnern, dass das Konfidenzintervall unserem Stichprobenmittelwert keine Wahrscheinlichkeit zuweist, sondern besagt, dass das Konfidenzintervall bei wiederholten Zufallsstichproben den Mittelwert der Grundgesamtheit in $100(1 - \alpha)\%$ der Fälle einschließen sollte. Um diese Behauptung zu testen, ändern wir die von uns im vorherigen Abschnitt geschriebene Python-Funktion leicht ab.

```
def CI_eval_t(pop_mean, sigma, n, estimate, alpha):
    """Funktion zur Evaluierung des Konfidenzintervalls"""
    out = {f"int(({1 - x} * 100)}%": {} for x in alpha}

    for e, _alpha in enumerate(alpha):
        key = f"int(({1 - _alpha} * 100)}%"
        out[key]["true value"] = pop_mean
        out[key]["estimate"] = estimate
        upper = estimate + t.ppf(1 - _alpha / 2, df=n - 1) * sigma / np.sqrt(n)
        out[key]["upper"] = upper
        lower = estimate - t.ppf(1 - _alpha / 2, df=n - 1) * sigma / np.sqrt(n)
        out[key]["lower"] = lower
        out[key]["test"] = pop_mean >= lower and pop_mean <= upper

    return out
```

Wenden wir nun unsere selbst erstellte Funktion `CI.eval.t()` auf unsere Daten an. Wir setzen `pop.mean = f_mean`, `sigma = s`, `n = sample.size`, `estimate = x.bar`, `alpha = [0.1, 0.05, 0.01]`, um zu evaluieren, ob die drei oben konstruierten Konfidenzintervalle (90%, 95%

und 99%) den Populationsmittelwert enthalten. Schließlich wandeln wir den resultierenden Vektor in ein `dataframe`-Objekt um, um die Lesbarkeit zu verbessern.

```
pd.DataFrame.from_dict(  
    CI_eval_t(  
        pop_mean=f_mean,  
        sigma=s,  
        n=sample_size,  
        estimate=sample_mean,  
        alpha=[0.1, 0.05, 0.01],  
    )  
)
```

	90%	95%	99%
true value	163.653285	163.653285	163.653285
estimate	161.442362	161.442362	161.442362
upper	169.748437	172.038358	178.062947
lower	153.136287	150.846365	144.821777
test	True	True	True

`f_mean`

```
np.float64(163.65328467153284)
```

`sample_mean`

```
np.float64(161.44236164394297)
```

Nun, ein interessantes Ergebnis. Der wahre Mittelwert der Population (`f_mean` = 163, 65) wird von allen drei Konfidenzintervallen erfasst. Der Mittelwert unserer Zufallsstichprobe (`sample_mean` = 161, 4) war ein recht guter Schätzer.

Übungsaufgaben

Konfidenzintervall

1. Berechnen Sie das 95% Konfidenzintervall umd den Mittelwert einer Normalverteilung $N(-2, 3)$
2. Stellen Sie die Normalverteilung inklusive des Konfidenzintervalls dar.

Frage 1 ...

Frage 2 ...

Lösung

► Show code cell content

► Show code cell content

Punktschätzungen bei unbekanntem σ

1. Welche Wahrscheinlichkeitsverteilung wird für die Berechnungen von Punktschätzungen bei unbekannter Standardabweichnung der Grundgesamtheit und kleiner Stichprobengrösse ($n < 30$) verwendet?
2. Wir simulieren eine Grundgesamtheit an Daten indem wir 100 Zufallszahlen zwischen -100 und 100 generieren. Berechnen Sie Mittelwert und Standardabweichung für diese Zufallsdaten.
3. Nehmen Sie eine Stichprobe vom Umfang $n = 10$. Berechnen Sie Mittelwert und Konfidenzintervall ($\alpha = 0,05$) für die Stichprobe unter der Verwendung einer geeigneten Wahrscheinlichkeitsverteilung und überprüfen Sie ob der Mittelwert der Grundgesamtheit innerhalb des Konfidenzintervalls liegt.

```
# Frage 2 ...
```

```
# Frage 3 ...
```

Lösungen

▶ Show code cell content

▶ Show code cell content

Punkt- und Intervallschätzungen

1. Erklären Sie was man unter Punktschätzung beziehungsweise Intervallschätzung versteht.
2. Wie viel größer muß die Stichprobe bei einer Intervallschätzung sein um die Länge der Konfidenzintervalle zu halbieren?

Lösung

Lernziele

Hypothesentests

- Die Grundlagen des Hypothesentests sowie das Aufstellen von Null- und Alternativhypothese, Wahl des Signifikanzniveaus und Umgang mit Fehlern 1ter und 2ter Art, kritischer Wert und p -Wert Ansatz wird den Studierenden vermittelt und anhand von Beispielen verdeutlicht
- Verschiedene Testverfahren wie : Einstichproben t -Test , Zweistichproben t -Test , Chi-Quadrat-Test auf Anpassungsgüte , Chi-Quadrat-Unabhängigkeitstest , t -Test für die Korrelation werden vorgestellt und exemplarisch in Python behandelt
- Die Studierenden sollen lernen schrittweise eigenständig Hypothesentests nachzuvollziehen und durchzuführen
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Hypothesentests , Formulierung der Nullhypothese und der Alternativhypothese , links-,rechts-,beidseitige Tests , Fehler 1ter und 2ter Art , Signifikanzniveau , kritische Wert und der p -Wert-Ansatz , Einstichproben t -Test , Zweistichproben t -Test , Chi-Quadrat-Test auf Anpassungsgüte , Chi-Quadrat-Unabhängigkeitstest , t -Test für die Korrelation

Hypothesentests

Ein sehr häufiges Problem, mit dem Wissenschaftler konfrontiert sind, ist die **Bewertung der Signifikanz** von verstreuten statistischen Daten. Aufgrund der begrenzten Verfügbarkeit von Beobachtungsdaten wenden Wissenschaftler inferenzstatistische Methoden an, um zu entscheiden, ob die beobachteten Daten signifikante Informationen enthalten oder ob die verstreuten Daten nichts weiter als die Manifestation der inhärent probabilistischen Natur des Datenerzeugungsprozesses sind.

Im Allgemeinen formuliert ein Wissenschaftler ein solches Problem wie folgt. Der Wissenschaftler erstellt ein Modell, das lediglich eine Vereinfachung des Datenerzeugungsprozesses darstellt, und betrachtet eine bestimmte Annahme - eine so genannte [Hypothese](#) - dieses Modells. Anhand von Daten will er diese vorläufige Hypothese bewerten.

Bei der Hypothesenprüfung geht es darum, **auf der Grundlage von Stichproben aus der Grundgesamtheit statistische Rückschlüsse auf diese Grundgesamtheit zu ziehen**. Eine Möglichkeit zur Schätzung eines Grundgesamtheitsparameters ist die Erstellung von Konfidenzintervallen. Eine andere Möglichkeit besteht darin, eine Entscheidung über einen Parameter in Form eines Tests zu treffen. Jeder Hypothesentest erfordert die Erhebung von Daten (Stichproben). Wenn die Hypothese als richtig angenommen wird, kann der Wissenschaftler die erwarteten Ergebnisse eines Experiments berechnen. Weichen die beobachteten Daten erheblich von den erwarteten Ergebnissen ab, so gilt die Annahme als falsch. Auf der Grundlage der beobachteten Daten trifft der Wissenschaftler also eine Entscheidung darüber, ob es aufgrund der Analyse der Daten genügend Beweise dafür gibt, dass das Modell - die Hypothese - verworfen werden sollte, oder ob es keine ausreichenden Beweise für die Verwerfung der angegebenen Hypothese gibt.

Einführung in Hypothesentests

In der **Inferenzstatistik** geht es darum, Entscheidungen oder Urteile über den Wert einer bestimmten Beobachtung oder Messung zu treffen. Eine der am häufigsten verwendeten Methoden, um solche Entscheidungen zu treffen, ist die Durchführung eines [Hypothesentests](#). Eine [Hypothese](#) ist ein

Erklärungsvorschlag für ein Phänomen. Im Zusammenhang mit statistischen Hypothesentests ist der Begriff Hypothese eine Aussage über etwas, von dem angenommen wird, dass es wahr ist.

Zu einem Hypothesentest gehören zwei Hypothesen: die **Nullhypothese** und die **Alternativhypothese**. Die Nullhypothese (H_0) ist eine zu prüfende Aussage. Die Alternativhypothese (H_A) ist eine Aussage, die als Alternative zur Nullhypothese betrachtet wird.

Mit dem Hypothesentest soll geprüft werden, ob die Nullhypothese zugunsten der Alternativhypothese verworfen werden sollte. Die grundlegende Logik eines Hypothesentests besteht darin, zwei statistische Datensätze zu vergleichen. Ein Datensatz wird durch eine Stichprobe gewonnen, der andere Datensatz stammt aus einem idealisierten Modell. Wenn die Stichprobendaten mit dem idealisierten Modell übereinstimmen, wird die Nullhypothese nicht verworfen; wenn die Stichprobendaten nicht mit dem idealisierten Modell übereinstimmen und somit eine Alternativhypothese unterstützen, wird die Nullhypothese zugunsten der Alternativhypothese verworfen.

Das Kriterium für die Entscheidung über die Ablehnung der Nullhypothese ist eine so genannte Teststatistik. Die Teststatistik ist eine Zahl, die aus dem Datensatz berechnet wird, der durch Messungen und Beobachtungen oder, allgemeiner, durch Stichproben gewonnen wird.

Formulierung der Hypothese

Jeder Hypothesentest beginnt mit der Formulierung der Nullhypothese und der Alternativhypothese. Dieser Abschnitt konzentriert sich auf Hypothesentests für einen Grundgesamtheitsmittelwert, μ jedoch gilt das allgemeine Verfahren für jeden Hypothesentest.

Die Nullhypothese für einen Hypothesentest für einen Mittelwert der Grundgesamtheit, μ wird ausgedrückt als

$$H_0 : \mu = \mu_0,$$

wobei μ_0 eine Zahl ist.

Die Formulierung der Alternativhypothese hängt vom Zweck des Hypothesentests ab. Es gibt drei Möglichkeiten, eine Alternativhypothese zu formulieren (s.369, s.97).

Wenn es bei dem Hypothesentest darum geht, zu entscheiden, ob ein Grundgesamtheitsmittelwert von dem angegebenen Wert μ_0 abweicht, wird die Alternativhypothese wie folgt formuliert

$$H_A : \mu \neq \mu_0.$$

Ein solcher Hypothesentest wird als **zweiseitiger Test** bezeichnet.

Wenn es bei dem Hypothesentest darum geht, zu entscheiden, ob der Mittelwert der Grundgesamtheit, μ kleiner ist als der angegebene Wert μ_0 , wird die Alternativhypothese wie folgt ausgedrückt

$$H_A : \mu < \mu_0.$$

Ein solcher Hypothesentest wird als **linksseitiger Test** bezeichnet.

Geht es bei dem Hypothesentest darum, zu entscheiden, ob der Mittelwert der Grundgesamtheit, μ größer als ein bestimmter Wert μ_0 ist, wird die Alternativhypothese wie folgt ausgedrückt

$$H_A : \mu > \mu_0.$$

Ein solcher Hypothesentest wird als **rechtsseitiger Test** bezeichnet.

Man beachte, dass ein Hypothesentest als **einseitiger Test** bezeichnet wird, wenn er entweder "linksseitig" oder "rechtsseitig" ist.

	zweiseitiger Test	linksseitiger Test	rechtsseitiger Test
Beziehung zwischen μ, μ_0 Ablehnung	\neq	<	>

Fehler vom Typ I, Fehler vom Typ II und Signifikanzniveau

Jede Entscheidung, die auf der Grundlage eines Hypothesentests getroffen wird, kann falsch sein. Im Rahmen von Hypothesentests gibt es zwei Arten von Fehlern: [Fehler vom Typ I und Fehler vom Typ II](#). Ein Fehler vom Typ I tritt auf, wenn eine wahre Nullhypothese abgelehnt wird (ein “falsches Positiv”), während ein Fehler vom Typ II auftritt, wenn eine falsche Nullhypothese nicht abgelehnt wird (ein “falsches Negativ”). Mit anderen Worten, ein Fehler vom Typ I besteht darin, dass ein Effekt festgestellt wird, der nicht vorhanden ist, während ein Fehler vom Typ II darin besteht, dass ein Effekt nicht festgestellt wird, der vorhanden ist.

	H_0 trifft zu	H_0 trifft nicht zu
H_0 nicht ablehnen	korrekte Entscheidung	Typ II Fehler
H_0 ablehnen	Typ I Fehler	korrekte Entscheidung

Wenn Sie sich nicht sicher sind, ob es sich um einen Fehler des Typs I oder des Typs II handelt, hilft Ihnen vielleicht eine Illustration ([hier](#)).

Die Durchführung eines Hypothesentests bedeutet immer, dass die Möglichkeit besteht, eine falsche Entscheidung zu treffen. Die Wahrscheinlichkeit eines Fehlers vom Typ I (eine wahre Nullhypothese wird abgelehnt) wird allgemein als [Signifikanzniveau](#) des Hypothesentests bezeichnet und mit α angegeben. Die Wahrscheinlichkeit eines Fehlers vom Typ II (eine falsche Nullhypothese wird nicht abgelehnt) wird mit β angegeben. Beachten Sie, dass bei einem festen Stichprobenumfang die Wahrscheinlichkeit β umso größer ist, je kleiner das Signifikanzniveau α ist, eine falsche Nullhypothese nicht zurückzuweisen (s.385).

Das Ergebnis eines Hypothesentests ist eine Aussage zu Gunsten der Nullhypothese oder zu Gunsten der Alternativhypothese. Wenn die Nullhypothese abgelehnt wird, liefern die Daten genügend Beweise, um die Alternativhypothese zu stützen. Wenn die Nullhypothese nicht verworfen wird, liefern die Daten keine ausreichenden Beweise für die Alternativhypothese. Wenn der Hypothesentest auf dem Signifikanzniveau α durchgeführt wird, kann man sagen, dass die Testergebnisse auf dem **α -Niveau statistisch signifikant sind**. Wenn die Nullhypothese auf dem Signifikanzniveau α nicht abgelehnt wird, kann man sagen, dass die Testergebnisse auf dem **α -Niveau statistisch nicht signifikant sind**.

Der kritische Wert und der p -Wert-Ansatz bei

der Hypothesenprüfung

Um zu entscheiden, ob die Nullhypothese abzulehnen ist, wird eine **Teststatistik** berechnet. Die Entscheidung wird auf der Grundlage des numerischen Wertes der Teststatistik getroffen. Es gibt zwei Ansätze, um zu dieser Entscheidung zu gelangen: Der Ansatz des **kritischen Wertes** und der Ansatz des **p-Wertes**.

Der Ansatz des kritischen Wertes

Mit dem Ansatz des **kritischen Wertes** wird festgestellt, ob die beobachtete Teststatistik zu stark von einem definiertem kritischen Wert abweicht oder nicht. Dazu wird die beobachtete Teststatistik (berechnet auf der Grundlage der Stichprobendaten) mit dem kritischen Wert, einer Art Grenzwert, verglichen. Wenn die Teststatistik extremer ist als der kritische Wert, wird die Nullhypothese abgelehnt. Wenn die Teststatistik nicht so extrem ist wie der kritische Wert, wird die Nullhypothese nicht verworfen. Der kritische Wert wird auf der Grundlage des vorgegebenen Signifikanzniveaus α und der Art der Wahrscheinlichkeitsverteilung des idealisierten Modells berechnet. Der kritische Wert teilt die Fläche unter der Wahrscheinlichkeitsverteilungskurve in die **Ablehnungsregion(en)** und in die **Nichtablehnungsregion**.

Die folgenden drei Abbildungen zeigen einen rechtsseitigen Test, einen linksseitigen Test und einen zweiseitigen Test. Das idealisierte Modell in den Abbildungen, und damit H_0 wird durch eine glockenförmige normale Wahrscheinlichkeitskurve beschrieben.

Bei einem **zweiseitigen** Test wird die Nullhypothese abgelehnt, wenn die Teststatistik entweder zu klein oder zu groß ist. Der Ablehnungsbereich für einen solchen Test besteht also aus zwei Teilen: einem links und einem rechts.

► Show code cell content

► Show code cell content

Bei einem **linksseitigen Test** wird die Nullhypothese abgelehnt, wenn die Teststatistik zu klein ist. Der Ablehnungsbereich für einen solchen Test besteht also aus einem Teil, der links von der Mitte liegt.

► Show code cell content

Bei einem **rechtsseitigen Test** wird die Nullhypothese abgelehnt, wenn die Teststatistik zu groß ist. Der Ablehnungsbereich für einen solchen Test besteht also aus einem Teil, der sich rechts von der

Mitte befindet.

► Show code cell content

Der p -Wert-Ansatz

Bei der p -Wert-Methode wird die Wahrscheinlichkeit (p -Wert) des numerischen Wertes der Teststatistik mit dem angegebenen Signifikanzniveau (α) des Hypothesentests verglichen.

Der p -Wert entspricht der Wahrscheinlichkeit, Stichprobendaten zu beobachten, die mindestens so extrem sind wie die tatsächlich erhaltene Teststatistik. Kleine p -Werte sind ein Beweis gegen die Nullhypothese. Je kleiner (näher an 0) der p -Wert ist, desto stärker ist der Beweis gegen die Nullhypothese.

Ist der p -Wert kleiner als oder gleich dem angegebenen Signifikanzniveau α ist, wird die Nullhypothese abgelehnt; andernfalls wird die Nullhypothese nicht abgelehnt. Mit anderen Worten: wenn $p \leq \alpha$, wird H_0 abgelehnt; andernfalls, wenn $p > \alpha$, wird H_0 nicht abgelehnt.

Folglich kann durch die Kenntnis des p -Wertes jedes gewünschte Signifikanzniveau bewertet werden. Beträgt der p -Wert eines Hypothesentests beispielsweise 0,01, kann die Nullhypothese auf jedem Signifikanzniveau größer oder gleich 0,01 abgelehnt werden. Bei einem Signifikanzniveau kleiner als 0,01 wird sie nicht verworfen. Daher wird der p -Wert in der Regel verwendet, um die Stärke des Beweises gegen die Nullhypothese ohne Bezug auf das Signifikanzniveau zu bewerten.

Die folgende Tabelle enthält Leitlinien für die Verwendung des p -Werts zur Bewertung der Beweise gegen die Nullhypothese (s.388).

p -Wert	Hinweise gegen H_0
$p > 0,10$	Schwache oder keine Hinweise
$0,05 < p \leq 0,10$	Mäßiger Hinweis
$0,01 < p \leq 0,05$	Starker Hinweis
$p \leq 0,01$	Sehr starker Hinweis

Hypothesentests für den Mittelwert einer Grundgesamtheit

Die einfachste Form eines Hypothesentests ist der **Test auf einen Grundgesamtheitsmittelwert**. Bei diesem Test vergleichen wir den aus der Stichprobe (Beobachtungen) gewonnenen Mittelwert μ mit einem angenommenen Grundgesamtheitsmittelwert μ_0 . Die Nullhypothese besagt, dass der Mittelwert und der Mittelwert der Grundgesamtheit gleich sind, und wird daher wie folgt geschrieben

$$H_0 : \mu = \mu_0.$$

Weichen die beobachteten Daten (μ) signifikant vom angenommenen Mittelwert der Grundgesamtheit (μ_0) ab, dann wird die Annahme H_0 zugunsten der Alternativhypothese H_A verworfen. Je nach der spezifischen Forschungsfrage wird die Alternativhypothese H_A wie folgt formuliert

$$H_A : \mu < \mu_0 \quad \text{oder} \quad \mu > \mu_0 \quad \text{oder} \quad \mu \neq \mu_0.$$

Wenn die Daten jedoch nicht genügend Beweise liefern, um die angegebene Hypothese (H_0) zu verwerfen, verwerfen wir H_0 nicht und schließen, dass die Daten nicht genügend Beweise liefern, um anzunehmen, dass sich der beobachtete Mittelwert μ vom angenommenen Grundgesamtheitsmittelwert (μ_0) unterscheidet. Die beobachtete Variabilität in den Daten wird auf die inhärent probabilistische Natur des Datenerzeugungsprozesses zurückgeführt, oder mit anderen Worten, die beobachtete Variabilität in den Daten ist auf den Zufall zurückzuführen.

In den nächsten Abschnitten wird gezeigt, dass für die eigentliche Berechnung der statistischen Signifikanz unser Wissen über die Grundgesamtheitsparameter von Bedeutung ist.

Hypothesentests für einen Grundgesamtheitsmittelwert bei bekannter Standardabweichung

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm
```

Wenn die Standardabweichung (σ) der Grundgesamtheit bekannt ist, wird ein Hypothesentest, der für einen Mittelwert der Grundgesamtheit durchgeführt wird, als **z -Test für einen Mittelwert** oder einfach als **z -Test** bezeichnet.

Ein z -Test ist ein Hypothesentest zum Testen eines Mittelwerts der Grundgesamtheit, μ , gegen einen angenommenen Mittelwert der Grundgesamtheit, μ_0 . Der z -Test setzt normalverteilte Variablen oder einen großen Stichprobenumfang voraus; dann garantiert der zentrale Grenzwertsatz eine normalverteilte Stichprobenverteilung. Darüber hinaus muss σ , die Standardabweichung der Grundgesamtheit, bekannt sein. In der Praxis ist diese Annahme fast nie erfüllt, so dass der z -Test nur selten angewendet wird. Er ist jedoch der einfachste Hypothesentest und daher ein guter Einstieg in das Thema.

Zur Durchführung des z -Tests gehen wir schrittweise vor, wie in der folgenden Tabelle dargestellt. Zunächst wird der **Ansatz des kritischen Werts** dargestellt, dann wird in einem zweiten Schritt die Analyse für den **Ansatz des p -Werts** wiederholt.

-
- | | |
|------------|--|
| Schritt 1 | Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an. |
| Schritt 2 | Legen Sie das Signifikanzniveau, α fest. |
| Schritt 3 | Berechnen Sie den Wert der Teststatistik. |
| Schritt 4a | Ansatz des kritischen Wertes: Bestimmung des kritischen Wertes. |
| Schritt 4b | P -Wert-Ansatz: Bestimmen Sie den p -Wert. |
| Schritt 5a | Ansatz des kritischen Werts: Wenn der Wert der Teststatistik in den |
| Schritt 5b | P -Wert-Ansatz: Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht able |
| Schritt 6 | Interpretieren Sie das Ergebnis des Hypothesentests. |
-

z -Test mit einem Mittelwert: Ein Beispiel

In diesem Abschnitt arbeiten wir mit dem [students](#) Datensatz. Sie können die Datei [students.csv](#) [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen geeigneten Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id*, *Name*, *Geschlecht*, *Alter*, *Größe*, *Gewicht*, *Religion*, *nc_score*, *Semester*, *Hauptfach*, *Nebenfach*, *score1*, *score2*, *online_tutorial*, *graduated*, *salary*.

Zur Veranschaulichung der Hypothesentests untersuchen wir das Durchschnittsgewicht der Studenten und vergleichen es mit dem Durchschnittsgewicht der erwachsenen Europäer. [Walpole et al. \(2012\)](#) veröffentlichten Daten über das durchschnittliche Körpergewicht (kg) pro Region, einschließlich Europa. Sie geben die durchschnittliche Körpermasse für die europäische erwachsene Bevölkerung mit 70,8 kg an. Wir setzen μ_0 , den Mittelwert der Bevölkerung, entsprechend fest, $\mu_0 = 70,8$. Aufgrund des methodischen Ansatzes von haben sie leider keine Standardabweichung (σ) der Gewichte europäischer Erwachsener angegeben. Zu Demonstrationszwecken gehen wir davon aus, dass die Gewichtsdaten aus dem Studentendatensatz eine gute Näherung für die interessierende Population darstellen. Daher setzen wir σ auf die Standardabweichung der Gewichtsvariablen im `students` Datensatz.

```
mu0 = 70.8
sigma_z = np.std(students.weight)
sigma_z
```

```
np.float64(8.634637630634796)
```

Außerdem nehmen wir eine Zufallsstichprobe mit einem Stichprobenumfang von $n = 14$. Die Stichprobe besteht aus den Gewichten in kg von 14 zufällig ausgewählten Studenten aus dem Studentendatensatz. Schließlich berechnen wir den Stichprobenmittelwert (\bar{x}), die Stichprobenstatistik unseres Interesses. Die Stichprobenstatistik wird der Variablen `x_bar` zugewiesen.

```
n = 14
x_weight = students.weight.sample(n, random_state=4)
x_bar = np.mean(x_weight)
x_bar
```

```
np.float64(77.02857142857144)
```

Hypothesentests: Der Ansatz des kritischen Werts

Schritt 1: Geben Sie die Nullhypothese (H_0) und die Alternativhypothese (H_A) an.

Die Nullhypothese besagt, dass das Durchschnittsgewicht der Studenten (μ) gleich dem Durchschnittsgewicht europäischer Erwachsener von 70,8 kg (μ_0) ist, wie von Walpole et al. (2012) berichtet. Mit anderen Worten: Es gibt keinen Unterschied zwischen dem Durchschnittsgewicht der Studenten und dem Durchschnittsgewicht der europäischen Erwachsenen.

$$H_0 : \mu = 70,8$$

Zur Veranschaulichung testen wir drei Alternativhypotesen.

Alternativhypothese 1: Das Durchschnittsgewicht der Studenten entspricht nicht dem Durchschnittsgewicht der europäischen Erwachsenen. Mit anderen Worten: Es gibt einen Unterschied zwischen dem Durchschnittsgewicht der Studenten und dem Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_1} : \mu \neq 70,8$$

Alternativhypothese 2: Das Durchschnittsgewicht der Studenten ist geringer als das Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_2} : \mu < 70,8$$

Alternativhypothese 3: Das Durchschnittsgewicht der Studenten ist höher als das Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_3} : \mu > 70,8$$

Schritt 2: Legen Sie das Signifikanzniveau α fest.

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3: Berechnen Sie den Wert der Teststatistik.

Die folgende Gleichung wird zur Berechnung der Teststatistik z verwendet.

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

- Berechnen Sie den Wert der Teststatistik

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} = \frac{77,02 - 70,8}{8,64/\sqrt{14}} \approx 2,699$$

```
z = (x_bar - mu0) / sigma_z * np.sqrt(n)  
z
```

```
np.float64(2.699033971278552)
```

Schritt 4a: Bestimmen Sie den kritischen Wert.

Um den kritischen Wert zu berechnen, wenden wir die Funktion `norm.ppf()` in Python an. Es sei daran erinnert, dass wir auf drei Alternativhypotesen (H_{A_1} , H_{A_2} und H_{A_3}) testen und daher auch drei kritische Werte berechnen müssen ($z_{A_1} = \pm z_{\alpha/2}$, $z_{A_2} = -z_{\alpha}$ und $z_{A_3} = +z_{\alpha}$).

```
z_HA_1 = norm.ppf(1 - alpha / 2)  
z_HA_2 = norm.ppf(1 - (1 - alpha))  
z_HA_3 = norm.ppf(1 - alpha)
```

```
print(f"z HA 1: {z_HA_1}")  
print(f"z HA 2: {z_HA_2}")  
print(f"z HA 2: {z_HA_3}")
```

```
z HA 1: 1.959963984540054
z HA 2: -1.6448536269514722
z HA 2: 1.6448536269514722
```

Die kritischen Werte sind $z_{A_1} \approx \pm 1,96$, $z_{A_2} \approx -1,64$ und $z_{A_3} \approx 1,64$.

Schritt 5a: Wenn der Wert der Teststatistik in den Verwerfungsbereich fällt, ist H_0 zu verwerfen; andernfalls ist H_0 nicht zu verwerfen.

Der Wert der in Schritt 3 ermittelten Teststatistik ist $z \approx 2,699$. Es sei daran erinnert, dass wir drei Alternativhypthesen (H_{A_1} , H_{A_2} und H_{A_3}) untersuchen. Daher bewerten wir den Ablehnungsbereich für jede einzelne Hypothese.

- **Alternativhypothese H_{A_1} :** $\mu \neq 70,8$

Erinnern Sie sich an die kritischen Werte für H_{A_1} .

$$z_{A_1} = \pm z_{\alpha/2} = \pm 1,96$$

Fällt die Teststatistik ($z \approx 2,699$) in den Ablehnungsbereich? Beachten Sie, dass es sich um einen zweiseitigen Test handelt, d. h. wir werten die obere und die untere Grenze aus.

- Obere Grenze

$$2,699 > 1,96$$

```
# Obere Grenze
# Ablehnen?
print(z > abs(z_HA_1))
```

True

- Untere Grenze

$$2,699 < -1,96$$

```
# Untere Grenze
# Ablehnung?
print(z < -abs(z_HA_1))
```

False

► Show code cell content

Aufgrund der numerischen und grafischen Auswertung fällt der Wert in den Verwerfungsbereich, so dass wir H_0 verwerfen. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant.

- **Alternativhypothese H_{A_2} :** $\mu < 70,8$

Erinnern Sie sich an den kritischen Wert für H_{A_2} .

$$z_{A_2} = -z_\alpha = -1,64$$

Fällt die Teststatistik ($z \approx 2,699$) in den Ablehnungsbereich?

$$2,699 < -1,64$$

```
# Ablehnung?
print(z < z_HA_2)
```

False

► Show code cell content

Aufgrund der numerischen und grafischen Auswertung fällt der Wert nicht in den Verwerfungsbereich, so dass wir H_0 nicht verwerfen. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant.

- **Alternativhypothese H_{A_3} :** $\mu > 70,8$

Erinnern Sie sich an den kritischen Wert für H_{A_3} .

$$z_{A_3} = +z_\alpha = 1,64$$

Fällt die Teststatistik ($z \approx 2,699$) in den Ablehnungsbereich?

$$2,699 > 1,64$$

```
# Ablehnen?  
print(z > z_HA_3)
```

True

► Show code cell content

Aufgrund der numerischen und grafischen Auswertung fällt der Wert in den Verwerfungsbereich, so dass wir H_0 verwerfen. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests.

- **Alternativhypothese H_{A_1} :** $\mu \neq 70,8$

Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass sich das Durchschnittsgewicht der Studenten vom Durchschnittsgewicht der europäischen Erwachsenen unterscheidet.

- **Alternativhypothese H_{A_2} :** $\mu < 70,8$

Bei einem Signifikanzniveau von 5% liefern die Daten keine ausreichenden Beweise für die Schlussfolgerung, dass das Durchschnittsgewicht der Studenten geringer ist als das Durchschnittsgewicht der europäischen Erwachsenen

- **Alternativhypothese H_{A_3} :** $\mu > 70,8$

Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass sich das Durchschnittsgewicht der Studenten vom Durchschnittsgewicht der europäischen Erwachsenen unterscheidet.

Hypothesentests: Der p -Wert-Ansatz

Schritt 1: Geben Sie die Nullhypothese (H_0) und die Alternativhypothese (H_A) an.

Die Nullhypothese besagt, dass das Durchschnittsgewicht der Studenten (μ) gleich dem Durchschnittsgewicht europäischer Erwachsener von 70,8 kg (μ_0) ist, wie von Walpole et al. (2012) berichtet. Mit anderen Worten: Es gibt keinen Unterschied zwischen dem Durchschnittsgewicht der Studenten und dem Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_1} : \mu \neq 70,8$$

Zur Veranschaulichung testen wir drei Alternativhypotesen.

Alternativhypothese 1: Das Durchschnittsgewicht der Studenten entspricht nicht dem Durchschnittsgewicht der europäischen Erwachsenen. Mit anderen Worten: Es gibt einen Unterschied zwischen dem Durchschnittsgewicht der Studenten und dem Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_1} : \mu \neq 70,8$$

Alternativhypothese 2: Das Durchschnittsgewicht der Studenten ist geringer als das Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_2} : \mu < 70,8$$

Alternativhypothese 3: Das Durchschnittsgewicht der Studenten ist höher als das Durchschnittsgewicht der europäischen Erwachsenen.

$$H_{A_3} : \mu > 70,8$$

Schritt 2: Legen Sie das Signifikanzniveau α fest.

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3: Berechnen Sie den Wert der Teststatistik.

Die folgende Gleichung wird zur Berechnung der Teststatistik z verwendet.

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

- Berechnen Sie den Wert der Teststatistik

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} = \frac{77,02 - 70,8}{8,64/\sqrt{14}} \approx 2,699$$

```
z = (x_bar - mu0) / sigma_z * np.sqrt(n)  
z
```

```
np.float64(2.699033971278552)
```

Schritt 4b: Bestimmen Sie den p -Wert.

Um den p -Wert zu berechnen, verwenden wir die Funktion `norm.cdf()` in Python. Erinnern Sie sich daran, dass wir auf drei Alternativhypthesen testen (H_{A_1} , H_{A_2} und H_{A_3}), daher berechnen wir auch drei p -Werte ($(P(z_{A_1}))$, $(P(z_{A_2}))$ und $(P(z_{A_3}))$).

- **Alternativhypothese H_{A_1}** : $\mu \neq 70,8$

```
# Die Wahrscheinlichkeit, einen z-Wert zu beobachten, der größer oder kleiner ist, v  
upper = 1 - norm.cdf(abs(z))  
lower = norm.cdf(-abs(z))  
p_z_1 = upper + lower  
p_z_1
```

```
np.float64(0.0069541077277850295)
```

Aus Schritt 3 ergibt sich der Wert der Teststatistik $z \approx 2,699$. Der Test ist zweiseitig, d. h. der p -Wert ist die Wahrscheinlichkeit, dass ein Wert z in der Größenordnung von $2,699$ oder mehr oder ein Wert z in der Größenordnung von $-2,699$ oder weniger beobachtet wird. Diese Wahrscheinlichkeit beträgt $0,006954$. Also $p \approx 0,007$.

- **Alternativhypothese H_{A_2} :** $\mu < 70,8$

```
# Die Wahrscheinlichkeit, einen z-Wert oder einen kleineren Wert zu beobachten, wenn
p_z_2 = norm.cdf(z)
p_z_2
```

```
np.float64(0.9965229461361075)
```

Aus Schritt 3 ergibt sich der Wert der Teststatistik $z \approx 2,699$. Der Test ist linksschief, d. h. der p -Wert ist die Wahrscheinlichkeit, einen Wert z von $2,699$ oder weniger zu beobachten. Diese Wahrscheinlichkeit ist ungefähr 1 . $p \approx 1$.

- **Alternativhypothese H_{A_3} :** $\mu > 70,8$

```
# Die Wahrscheinlichkeit, einen z-Wert oder einen größeren Wert zu beobachten, wenn
p_z_3 = 1 - norm.cdf(z)
p_z_3
```

```
np.float64(0.003477053863892521)
```

Aus Schritt 3 ergibt sich der Wert der Teststatistik $z = 2,699$. Der Test ist rechtsschief, d. h. der p -Wert ist die Wahrscheinlichkeit, einen Wert z von $2,699$ oder mehr zu beobachten. Diese Wahrscheinlichkeit ist ungefähr $0,003477$; d.h. $p \approx 0,004$.

Schritt 5b: Wenn $p \leq \alpha$, wird H_0 verworfen; andernfalls wird H_0 nicht verworfen.

Der p -Wert der in Schritt 4 ermittelten Teststatistik wird mit dem benutzerdefinierten Signifikanzniveau α von 5% verglichen. Es sei daran erinnert, dass wir drei Alternativhypotesen (H_{A_1} , H_{A_2} und H_{A_3}) untersuchen. Wir führen also Vergleiche für jede einzelne Hypothese durch.

- **Alternativhypothese H_{A_1} :** $\mu \neq 70,8$

$$0,007 \leq 0,05$$

```
# Ablehnen?  
p_z_1 <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,05 und wir verwerfen H_0 . Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant und liefern einen sehr starken Beweis gegen die Nullhypothese.

- **Alternativhypothese H_{A_2}** : $\mu < 70,8$

$$1 \leq 0,05$$

```
# Ablehnen?  
p_z_2 <= alpha
```

```
np.False_
```

Der p -Wert ist größer als das angegebene Signifikanzniveau von 0,05 und wir verwerfen H_0 nicht. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant und liefern keinen ausreichenden Beweis gegen die Nullhypothese.

- **Alternativhypothese H_{A_3}** : $\mu > 70,8$

$$0,004 \leq 0,05$$

```
# Ablehnen?  
p_z_3 <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,05 und wir verwerfen H_0 . Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant und liefern einen sehr starken Beweis

gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests.

- **Alternativhypothese** $H_{A_1} : \mu \neq 70,8$

$p \approx 0,007$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass sich das Durchschnittsgewicht der Studenten vom Durchschnittsgewicht der europäischen Erwachsenen unterscheidet.

- **Alternativhypothese** $H_{A_2} : \mu < 70,8$

$p \approx 1$. Bei einem Signifikanzniveau von 5% liefern die Daten keine hinreichenden Anhaltspunkte für die Schlussfolgerung, dass das Durchschnittsgewicht der Studenten geringer ist als das Durchschnittsgewicht der europäischen Erwachsenen.

- **Alternativhypothese** $H_{A_3} : \mu > 70,8$

$p \approx 0,004$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass das Durchschnittsgewicht der Studenten höher ist als das Durchschnittsgewicht der europäischen Erwachsenen.

Hypothesentests für den Mittelwert einer Grundgesamtheit, wenn σ unbekannt ist

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import t, ttest_1samp
```

Ein Hypothesentest für einen Grundgesamtheitsmittelwert, bei dem die Standardabweichung σ der Grundgesamtheit nicht bekannt ist, wird auf die gleiche Weise durchgeführt, wie wenn die Standardabweichung der Grundgesamtheit bekannt ist. Der einzige Unterschied besteht darin, dass die [t-Verteilung](#) und nicht die Standardnormalverteilung (z -Verteilung) verwendet wird.

Bei einem Test mit der Nullhypothese $H_0 : \mu = \mu_0$ wird die Teststatistik, t , wie folgt berechnet

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

Dieses Hypothesentestverfahren wird als **Einstichproben t -Test für einen Mittelwert** oder einfach als **t -Test** bezeichnet. Es sei daran erinnert, dass Hypothesentests einem schrittweisen Verfahren folgen, das wie folgt zusammengefasst wird

-
- | | |
|------------|---|
| Schritt 1 | Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an. |
| Schritt 2 | Legen Sie das Signifikanzniveau, α fest. |
| Schritt 3 | Berechnen Sie den Wert der Teststatistik. |
| Schritt 4a | Ansatz des kritischen Wertes: Bestimmung des kritischen Wertes. |
| Schritt 4b | P-Wert-Ansatz: Bestimmen Sie den p-Wert. |
| Schritt 5a | Ansatz des kritischen Werts: Wenn der Wert der Teststatistik in den |
| Schritt 5b | P-Wert-Ansatz: Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht able |
| Schritt 6 | Interpretieren Sie das Ergebnis des Hypothesentests. |
-

Ähnlich wie im vorangegangenen Abschnitt stellen wir zunächst den Ansatz des **kritischen Werts** vor und wiederholen dann in einem zweiten Schritt die Analyse für den **p -Wert-Ansatz**. Diesmal verpacken wir jedoch den Ansatz des kritischen Werts in eine selbst erstellte Funktion. Für den p -Wert-Ansatz werden wir die leistungsstarke Maschinerie von Python nutzen und die vorhandene Funktion `ttest_1samp()` anwenden.

Hypothesentests: Der Ansatz des kritischen Werts

Erstellen wir eine Funktion namens `simple_ttest()`. Die Funktion nimmt als Eingangsargumente einen Vektor `x` (Stichprobendaten), `mu0`, das μ_0 entspricht, den Grundgesamtheitsdurchschnitt, gegen den getestet werden soll, das angegebene Signifikanzniveau α , bezeichnet als `alpha`, und die Methode des t -Tests, `left` oder `right` oder der Standardwert `two-sided`, mit dem Funktionsargument namens `Methode`. Die Ausgabe der Funktion ist ein Boolescher Wert, `TRUE` oder `FALSE`. Wenn `TRUE`, wird H_0 abgelehnt, wenn `FALSE`, wird H_0 nicht abgelehnt.

```

def simple_ttest(x, mu0, alpha, method="two-sided"):
    n = len(x)
    xbar = np.mean(x)
    s = np.std(x, ddof=1)
    # Berechne Teststatistik
    tstat = (xbar - mu0) / (s / np.sqrt(n))

    # Berechne kritischer Wert
    df = n - 1
    # für den linksseitigen Test
    if method == "left":
        crit_val = t.ppf(q=alpha, df=df)
        # Werte Ablehnungsbereich aus
        if tstat < crit_val:
            reject = True
        else:
            reject = False
    # für den rechtsseitigen Test
    if method == "right":
        crit_val = t.ppf(q=1 - alpha, df=df)
        # Werte Ablehnungsbereich aus
        if tstat > crit_val:
            reject = True
        else:
            reject = False

    # für den zweiseitigen Test (default)
    if method == "two-sided":
        crit_val = t.ppf(q=1 - alpha / 2, df=df)
        # Werte Ablehnungsbereich aus
        if abs(tstat) > abs(crit_val) and -abs(tstat) < -abs(crit_val):
            reject = True
        else:
            reject = False

    print("Significance level:", alpha)
    print("Degrees of freedom:", df)
    print("Test statistic:", round(tstat, 4))
    print("Critical value:", round(crit_val, 4))
    print("Reject H0:", reject)

```

Ein tolles Stück Code :-)

t-Test mit einem Mittelwert: Ein Beispiel

Nun ist es an der Zeit, unsere Funktion `simple_ttest()` zu testen. Dazu wiederholen wir das Beispiel aus dem vorherigen Abschnitt. Wir verwenden den `students` Datensatz. Sie können die

Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id, name, gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary*.

Wir untersuchen das Durchschnittsgewicht einer Zufallsstichprobe von Studenten aus dem `students` Datensatz und vergleichen es mit dem Durchschnittsgewicht aller erwachsenen Europäer. [Walpole et al. \(2012\)](#) veröffentlichten Daten über das durchschnittliche Körpergewicht (kg) pro Region, einschließlich Europa. Sie geben die durchschnittliche Körpermasse für die europäische erwachsene Bevölkerung mit 70,8 kg an. Wir setzen daher μ_0 , den Mittelwert der Grundgesamtheit, entsprechend fest ($\mu_0 = 70,8$). Ferner ziehen wir eine Zufallsstichprobe (x) mit einem Umfang von $n = 9$. Die Stichprobe besteht aus den Gewichten in kg von 9 zufällig ausgewählten Studenten.

```
mu0 = 70.8
n = 9

x = students.weight.sample(n, random_state=1)
x.head(10)
```

```
stud_id
883146    56.5
677171    86.5
141515    62.9
159416    62.4
175825    64.4
804998    76.3
746386    65.7
949645    61.4
847488    66.6
Name: weight, dtype: float64
```

Überprüfen der Hypothesen

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass das Durchschnittsgewicht der Studenten dem Durchschnittsgewicht der europäischen Erwachsenen entspricht, wie von Walpole et al. (2012) berichtet. Mit anderen Worten: Es gibt keinen Unterschied zwischen dem Durchschnittsgewicht der Studenten und dem Durchschnittsgewicht der europäischen Erwachsenen.

$$H_0 : \mu = 70,8$$

Erinnern Sie sich daran, dass die Formulierung der Alternativhypothese vorgibt, ob wir einen zweiseitigen, einen links- oder einen rechtsseitigen Hypothesentest durchführen.

Alternativhypothese 1

$$H_{A_1} : \mu \neq 70,8$$

führt zu einem zweiseitigen Hypothesentest.

Alternativhypothese 2

$$H_{A_2} : \mu < 70,8$$

zu einem linksseitigen Hypothesentest und

Alternativhypothese 3

$$H_{A_3} : \mu > 70,8$$

zu einem rechtsseitigen Hypothesentest

Schritt 2: Legen Sie das Signifikanzniveau, α fest.

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3, 4 und 5: Berechnen Sie den Wert der Teststatistik, bestimmen Sie den kritischen Wert, und werten Sie den Wert der Teststatistik aus. Wenn er in den Verwerfungsbereich fällt, verwerfen Sie H_0 ; andernfalls verwerfen Sie H_0 nicht

Jetzt kommt unsere selbst erstellte Funktion `simple_ttest()` ins Spiel. Wir geben der Funktion eine Zufallsstichprobe in Form eines Vektors, einen Wert für μ_0 , ein Signifikanzniveau α und die Methode (`two-sided`, `left` oder `right`) an. Erinnern Sie sich daran, dass zweiseitig der Standardwert ist. Wenn wir also keine Methode angeben, wird die Funktion den zweiseitigen Hypothesentest anwenden.

```
simple_ttest(x, mu0, alpha)
```

```
Significance level: 0.05
Degrees of freedom: 8
Test statistic: -1.271
Critical value: 2.306
Reject H0: False
```

Cool! ;-)

Ein rechtsseitiger t -Test

```
simple_ttest(x, mu0, alpha, method="right")
```

```
Significance level: 0.05
Degrees of freedom: 8
Test statistic: -1.271
Critical value: 1.8595
Reject H0: False
```

Ein linksseitiger t -Test

```
simple_ttest(x, mu0, alpha, method="left")
```

```
Significance level: 0.05
Degrees of freedom: 8
Test statistic: -1.271
Critical value: -1.8595
Reject H0: False
```

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

Wenn die Teststatistik und damit der Stichprobenmittelwert so extrem ist, dass er über den kritischen Wert hinausgeht und damit in den Verwerfungsbereich fällt, schließen wir, dass die Daten bei einem Signifikanzniveau von 5% genügend Beweise liefern, um H_0 zu verwerfen. Fällt die Teststatistik und damit der Stichprobenmittelwert dagegen in den Nicht-Verwerfungsbereich, so schließen wir, dass die Daten keine Beweise für die Ablehnung von H_0 liefern.

Hypothesentests in Python: Der p -Wert Ansatz

Der zweite Ansatz basiert auf der Zuweisung einer Wahrscheinlichkeit für den Wert der Teststatistik. Wenn die Teststatistik sehr extrem ist und die Nullhypothese wahr ist, wird der Teststatistik eine geringe Wahrscheinlichkeit zugewiesen. Ist die Teststatistik dagegen überhaupt nicht extrem, so wird ihr eine viel höhere Wahrscheinlichkeit zugewiesen. Diese Wahrscheinlichkeit wird als p -Wert bezeichnet.

Um den genauen p -Wert für einen gegebenen numerischen Wert zu berechnen, sind wir auf Software angewiesen, da die “In-Tabellen-nachschlagen-Methode” etwas mühsam ist, da nicht alle numerischen Werte der Teststatistik in einer solchen Tabelle angegeben sind. Dies führt zu Rundungsfehlern. Die Programmiersprache Python bietet jedoch einen sehr leistungsfähigen Mechanismus zur Durchführung von t -Tests. Die generische Funktion heißt `ttest_1samp`. Sie können mehr Details erfahren, indem Sie `help(ttest_1samp)` in Ihre Konsole eingeben. Obwohl die Funktion verschiedene Argumente liefert, ist der spezielle t -Test, für den wir uns jetzt interessieren, nämlich der Test eines Grundgesamtheitsmittelwertes, wenn σ unbekannt ist, ist recht einfach.

Wir wiederholen das gleiche Problem wie im vorherigen Abschnitt, wobei wir die bereits definierten Variablen verwenden. Zur Erinnerung: Die Stichprobe ist ein Vektor von Gewichten (x), $\mu_0 = 70$, 8 , angegeben als `mu0`, $n = 9$, angegeben als `n`. Mit diesen Informationen wenden wir die Funktion

`ttest_1samp` für einen zweiseitigen t -Test an, indem wir `ttest_1samp(x = x, mu = mu0, alternative = 'two-sided')` in Python eingeben. Wir vergleichen zur Auswertung ob der p -Wert unter $\alpha = 0,05$ liegt und damit statistisch Signifikant ist.

```
ttest_1samp(x, mu0, alternative="two-sided")
```

```
TtestResult(statistic=np.float64(-1.2710092384447174), pvalue=np.float64(0.239435182
```

```
0.2394 < alpha
```

```
False
```

Wahnsinn! Python führt einen Hypothesentest in einer einzigen Codezeile durch und gibt zusätzliche Informationen kostenlos aus! Gehen wir die Ausgabe der Funktion `ttest_1samp()` durch.

Die erste Zeile listet die Ergebnisse der Berechnungen für die Teststatistik t und den p -Wert auf: $t = -1,271$, p -Wert = 0,2394. Wir können die Alternativhypothese, die wir verwenden durch das Argument `alternative = less`, `greater`, `two-sided` ändern. In unserem Fall war sie zweiseitig. Versuchen Sie selbst, das Attribut in `less` oder `greater` zu ändern, um eine andere Alternativhypothese anzuwenden.

Es ist erwähnenswert, dass die Funktion `ttest_1samp()` die Ergebnisse des t -Tests ausgibt. Wir können also die Ergebnisse der Funktion `ttest_1samp()` in einer Variablen speichern.

Um den p -Wert abzurufen, geben wir ein

```
statistics, pvalue = ttest_1samp(x, mu0, alternative="two-sided")
```

```
pvalue
```

```
np.float64(0.23943518262349406)
```

```
statistics
```

```
np.float64(-1.2710092384447174)
```

Hypothesentests für zwei Grundgesamtheitsmittelwerte

Bisher haben wir uns auf Hypothesentests für den Mittelwert einer Grundgesamtheit konzentriert. In vielen Anwendungen wollen wir jedoch die Mittelwerte von zwei oder mehr Grundgesamtheiten vergleichen. In den folgenden Abschnitten werden Schlussfolgerungsverfahren für den Vergleich der Mittelwerte von zwei Grundgesamtheiten erörtert. Daher müssen wir zunächst zwischen Stichproben aus zwei unabhängigen Populationen und Stichproben aus nicht unabhängigen Populationen unterscheiden, die als abhängige Stichproben bezeichnet werden.

In den folgenden Abschnitten werden die Parameter und die Statistiken von Population 1 und Population 2 mit dem Index 1 bzw. 2 bezeichnet. Somit sind μ_1 und σ_1 die Populationsparameter der Population 1 und μ_2 und σ_2 die der Population 2. Analog dazu sind \bar{x}_1 , s_1 und n_1 der Stichprobenmittelwert, die Stichprobenstandardabweichung und der Stichprobenumfang der Grundgesamtheit 1, während \bar{x}_2 , s_2 und n_2 der aus der Grundgesamtheit 2 gezogenen Stichprobe entsprechen.

Für unabhängige Stichproben mit dem Umfang n_1 und n_2 der Grundgesamtheit 1 und der Grundgesamtheit 2 ist der Mittelwert aller möglichen Unterschiede zwischen den beiden Stichprobenmittelwerten gleich dem Unterschied zwischen den beiden Grundgesamtheitsmittelwerten.

$$\mu_{\bar{x}_1 - \bar{x}_2} = \mu_1 - \mu_2$$

Außerdem ist die Standardabweichung aller möglichen Unterschiede zwischen den beiden Stichprobenmittelwerten gleich der Quadratwurzel aus der Summe der Varianzen der Grundgesamtheit, jeweils dividiert durch den entsprechenden Stichprobenumfang.

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_1^2}{n_2}}$$

Eine normalverteilte Variable oder ein ausreichend großer Stichprobenumfang (man denke an den [zentralen Grenzwertsatz](#)) führt dazu, dass die Differenz der Stichprobenmittelwerte ($\bar{x}_1 - \bar{x}_2$) ebenfalls normalverteilt ist.

Die Hypothesentestverfahren für zwei Grundgesamtheitsmittelwerte sind im Grunde dieselben wie für einen Grundgesamtheitsmittelwert. Bitte beachten Sie, dass wir uns in den folgenden Abschnitten auf den p -Wert-Ansatz konzentrieren und nicht mehr auf den kritischen Wert-Ansatz eingehen. Daher wird das Hypothesentestverfahren leicht überarbeitet. Das schrittweise Vorgehen bei Hypothesentests ist wie folgt zusammengefasst

-
- Schritt 1 Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an.
 - Schritt 2 Legen Sie das Signifikanzniveau, α fest.
 - Schritt 3 Berechnen Sie den Wert der Teststatistik.
 - Schritt 4 P-Wert-Ansatz: Bestimmen Sie den p-Wert.
 - Schritt 5b P-Wert-Ansatz: Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht ablehnen
 - Schritt 6 Interpretieren Sie das Ergebnis des Hypothesentests.
-

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
-----  
Working on the host: imarevic-pc
```

```
-----  
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

```
-----  
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

Inferenz für zwei Grundgesamtheitsmittelwerte bei unabhängigen Stichproben; $s_1 \approx s_2$

```
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from random import sample  
from scipy.stats import t, ttest_ind  
import statsmodels.api as smi
```

In diesem Abschnitt führen wir einen Hypothesentest für die Mittelwerte von zwei Grundgesamtheiten durch. Wir gehen davon aus, dass die Standardabweichungen (s_1, s_2) der beiden Grundgesamtheiten gleich, aber unbekannt sind. Wenn wir jedoch σ und die Differenz der Stichprobenmittelwerte ($\bar{x}_1 - \bar{x}_2$) kennen würden, könnte die Teststatistik wie folgt geschrieben werden.

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sigma \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

In fast allen realen Anwendungen kennen wir σ jedoch nicht. Daher müssen wir es im Voraus schätzen. Das geht am besten, wenn man die Stichprobenvarianzen s_1^2 und s_2^2 als zwei Schätzungen für σ^2 betrachtet. Durch Zusammenfassen der beiden Stichprobenabweichungen und Gewichtung nach dem Stichprobenumfang ergibt sich der Schätzwert für σ^2 wie folgt

$$s_g^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2},$$

und durch Ziehen der Quadratwurzel erhalten wir

$$s_g = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}},$$

Die Größe s_g wird als [gewichtete Stichprobenstandardabweichung](#) bezeichnet (engl. pooled), wobei der tiefgestellte Index g für **gewichtet** steht.

Die Ersetzung von σ in der obigen Gleichung durch seine Schätzung s_g ergibt

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Der Nenner der Gleichung $s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$ ist der Schätzer der Standardabweichung von $\bar{x}_1 - \bar{x}_2$, der wie folgt geschrieben werden kann

$$s_{\bar{x}_1 - \bar{x}_2} = s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Bitte beachten Sie, dass die Gleichung für die Teststatistik t einer t -Verteilung folgt. Die Freiheitsgrade (df) sind gegeben durch

$$df = n_1 + n_2 - 2$$

Intervall-Schätzung von $\mu_1 - \mu_2$

Das $100(1 - \alpha)\%$ -Konfidenzintervall für $\mu_1 - \mu_2$ ist gegeben durch

$$(\bar{x}_1 - \bar{x}_2) \pm t \times s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}},$$

wobei sich der Wert von t aus der t -Verteilung für das gegebene Konfidenzniveau und $n_1 + n_2 - 2$ Freiheitsgrade ergibt.

Der Zweistichproben- t -Test für unabhängige Stichproben (engl. pooled t -Test): Ein Beispiel

Python ermöglicht es uns, einen Zweistichproben- t -Test für unabhängige Stichproben durchzuführen, indem wir die Funktion `ttest_ind()` um das Argument `equal_var = True` erweitern.

Um den Zweistichproben- t -Test zu üben, laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und weisen Sie ihm einen geeigneten Namen zu.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `Name`, `Geschlecht`, `Alter`, `Größe`, `Gewicht`, `Religion`, `nc_score`, `Semester`, `Hauptfach`, `Nebenfach`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Um den Zweistichproben- t -Test für unabhängige Stichproben zu veranschaulichen, untersuchen wir das mittlere Jahresgehalt (in Euro) der Absolventen. Die erste Grundgesamtheit besteht aus männlichen Studenten und die zweite aus weiblichen Studenten. **Die Frage ist, ob es einen Unterschied im mittleren Jahresgehalt der Absolventen in Bezug auf das Geschlecht gibt?**

Vorbereitung der Daten

Wir beginnen mit der Datenaufbereitung.

- Wir unterteilen den Datensatz anhand der binären Variable `graduated`, die angibt, ob der Student seinen Abschluss bereits gemacht hat. Die ganze Zahl 1 steht für "abgeschlossen", 0 bedeutet, dass der Schüler seinen Abschluss noch nicht gemacht hat.
- Dann teilen wir den Datensatz nach Geschlecht auf (männlich und weiblich).
- Dann ziehen wir aus jeder Teilmenge 50 weibliche und 50 männliche Studenten und extrahieren die Variable von Interesse, das mittlere Jahresgehalt (in Euro), das in der Spalte Gehalt gespeichert ist. Diese beiden Vektoren ordnen wir den Variablen `male_sample` und `female_sample` zu.

```

n = 50

male = students.loc[(students["gender"] == "Male") & (students["graduated"] == 1)]
female = students.loc[(students["gender"] == "Female") & (students["graduated"] == 1)]

male_sample = male["salary"].sample(n=50, random_state=1)
female_sample = female["salary"].sample(n=50, random_state=1)

```

Außerdem prüfen wir ob die Daten normalverteilt sind, indem wir ein Normalwahrscheinlichkeitsdiagramm erstellen, das oft als [Q-Q-Diagramm](#) bezeichnet wird. Wenn die Variable normalverteilt ist, sollte das Q-Q-Diagramm ungefähr linear sein.

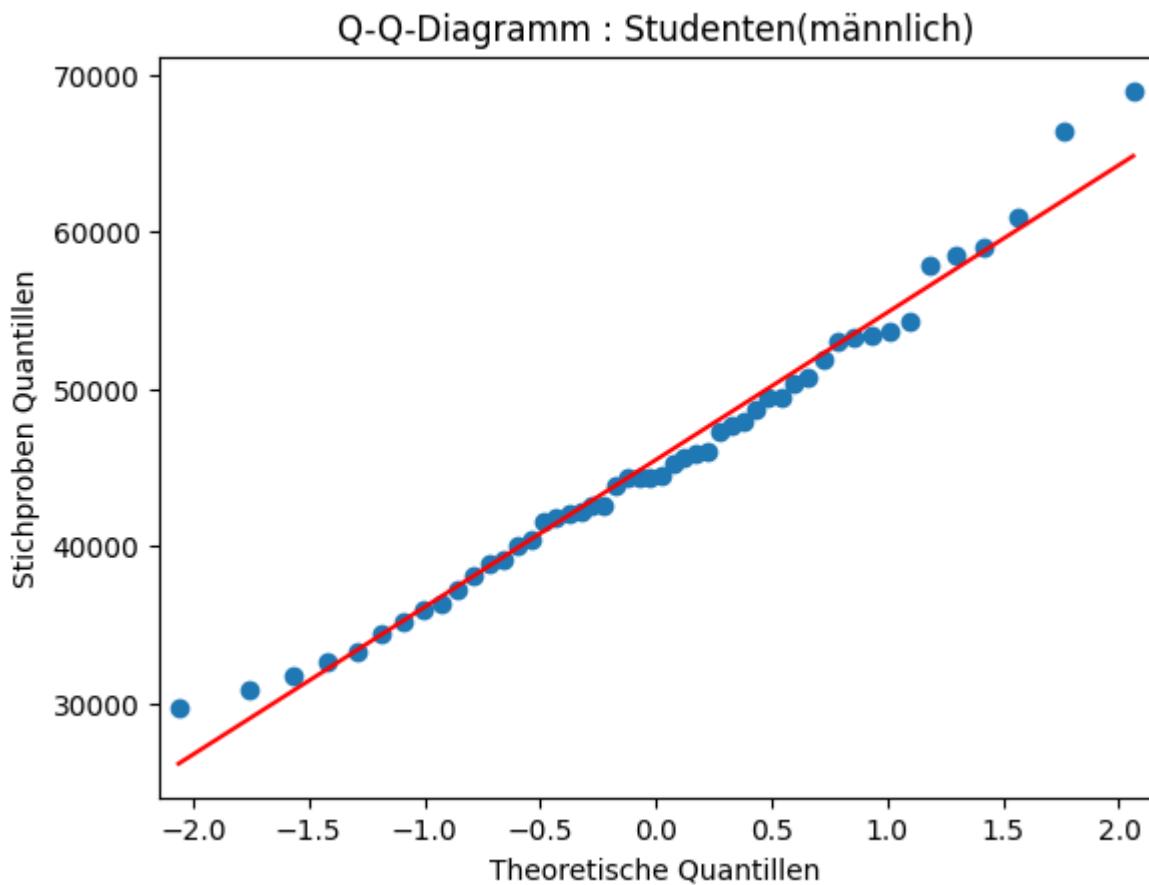
In Python können wir die Funktion `qqplot()` verwenden, um Q-Q-Plots zu erstellen.

```

# Erzeuge Q-Q Plot
qqp = smi.qqplot(male_sample, line="r")
ax = qqp.gca()
ax.set_title("Q-Q-Diagramm : Studenten(männlich)")
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")

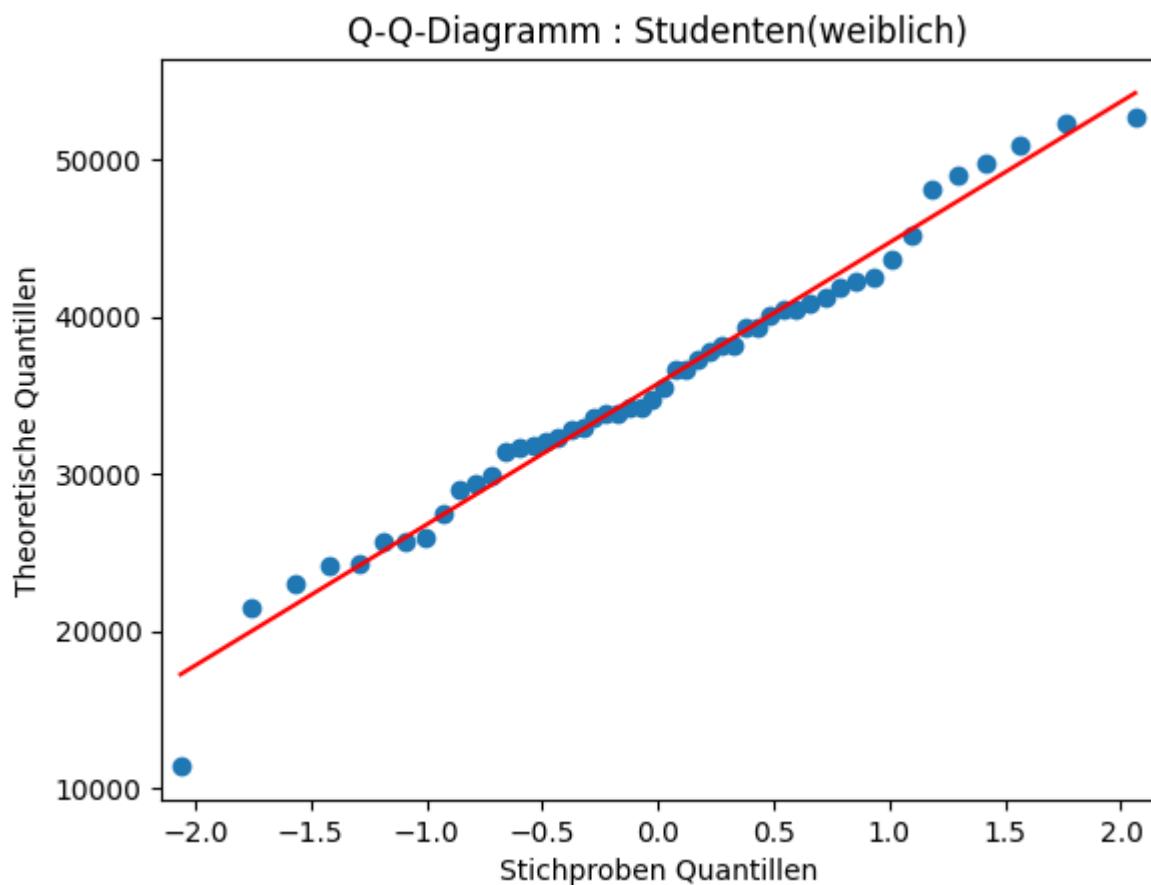
```

```
Text(0, 0.5, 'Stichproben Quantillen')
```



```
# Erzeuge Q-Q Plot
qqp = smp.qqplot(female_sample, line="r")
ax = qqp.gca()
ax.set_title("Q-Q-Diagramm : Studenten(weiblich)")
ax.set_xlabel("Stichproben Quantillen")
ax.set_ylabel("Theoretische Quantillen")
```

```
Text(0, 0.5, 'Theoretische Quantillen')
```



Wir sehen, dass die Stichprobendaten etwas Rauschen beinhalten sind, aber sie sind immer noch ungefähr normalverteilt. Die Abweichung von der Geraden im oberen und unteren Teil deutet darauf hin, dass die Wahrscheinlichkeitsverteilung leicht schief ist.

Außerdem prüfen wir, ob die Standardabweichungen der beiden Grundgesamtheiten ungefähr gleich sind. Als Faustregel gilt, dass die Bedingung gleicher Standardabweichungen der Grundgesamtheit erfüllt ist, wenn das Verhältnis der größeren zur kleineren Standardabweichung der Stichprobe kleiner als 2 ist. Gehen wir davon aus, dass die Daten des `students` Datensatzes eine gute Annäherung an die Grundgesamtheit darstellen.

```
# Berechne Standardabweichung
np.std(male["salary"])
```

```
np.float64(9653.18969182546)
```

```
# Berechne Standardabweichung
np.std(female["salary"])
```

```
np.float64(7723.490231798204)
```

```
# Berechne Verhältnis
np.std(male["salary"]) / np.std(female["salary"])
```

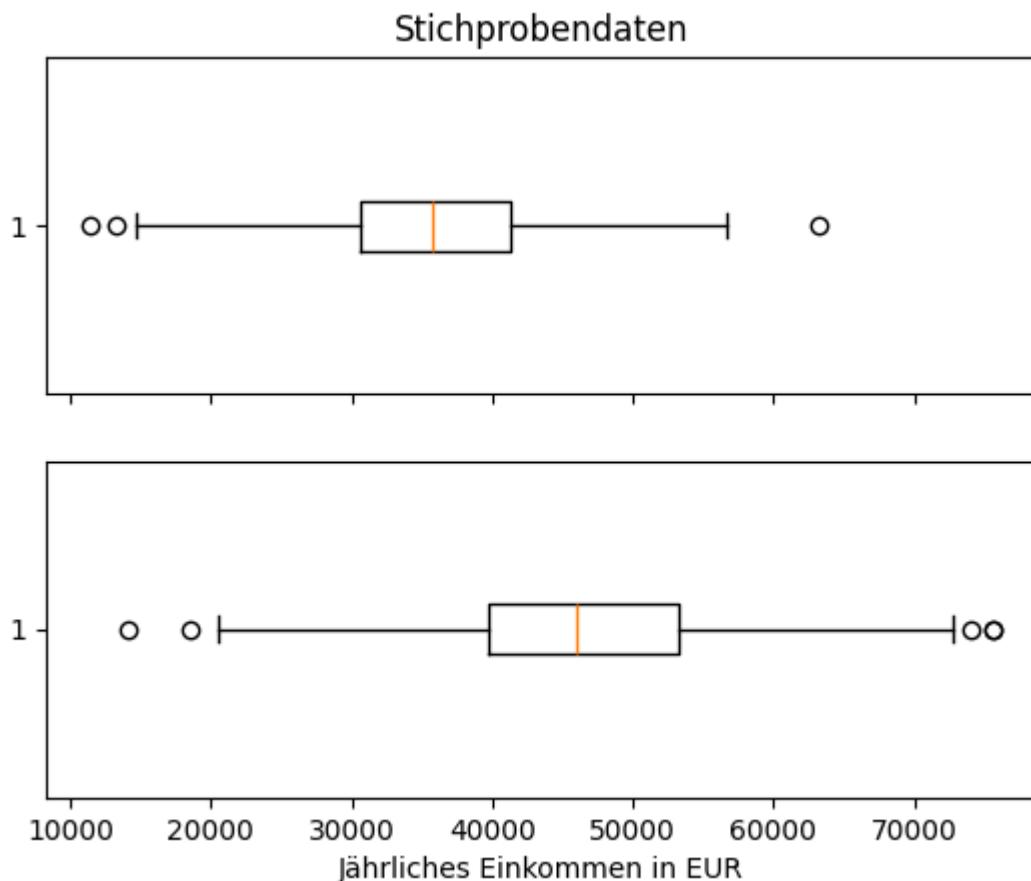
```
np.float64(1.2498481129790953)
```

Das Verhältnis liegt bei etwa 1,249, so dass wir zu dem Schluss kommen, dass das Kriterium der gleichen Standardabweichungen der Bevölkerung erfüllt ist. Eine einfache Visualisierungstechnik zur Bewertung der Streuung einer Variablen ist die Darstellung eines Boxplots.

```
fig1, (ax1, ax2) = plt.subplots(2, sharex="all")

ax1.set_title("Stichprobendaten")
ax2.boxplot(male["salary"], vert=False)
ax1.boxplot(female["salary"], vert=False)
ax2.set_xlabel("Jährliches Einkommen in EUR")
```

Text(0.5, 0, 'Jährliches Einkommen in EUR')



Überprüfung der Hypothesen

Wir führen den Zweistichproben-*t*-Test für unabhängige Stichproben durch, indem wir das schrittweise Durchführungsverfahren für Hypothesentests befolgen.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass das durchschnittliche Jahresgehalt der männlichen Absolventen (μ_1) gleich dem durchschnittlichen Jahresgehalt der weiblichen Absolventen (μ_2) ist.

$$H_0 : \mu_1 = \mu_2$$

Es sei daran erinnert, dass die Formulierung der Alternativhypothese vorgibt, ob wir einen zweiseitigen, einen links- oder einen rechtsseitigen Hypothesentest durchführen. Wir wollen prüfen,

ob das Gehalt der männlichen Absolventen (μ_1) höher ist als das durchschnittliche Jahresgehalt der weiblichen Absolventen (μ_2), daher wird die Alternativhypothese wie folgt formuliert

Alternative Hypothese

$$H_A : \mu_1 > \mu_2$$

Aus dieser Formulierung ergibt sich ein rechtsseitiger Hypothesentest.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,01$$

```
alpha = 0.01
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die Gleichung für die Teststatistik von oben.

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Wenn H_0 wahr ist, dann ist $\mu_1 - \mu_2 = 0$ und somit vereinfacht sich die Gleichung zu

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}},$$

wobei s_g gleich

$$s_g = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

ist.

```
# Berechne Teststatistik
n1 = len(male_sample)
n2 = len(female_sample)
s1 = np.std(male_sample, ddof=1)
s2 = np.std(female_sample, ddof=1)
x1_bar = np.mean(male_sample)
x2_bar = np.mean(female_sample)

sg = np.sqrt(((n1 - 1) * s1**2 + (n2 - 1) * s2**2) / (n1 + n2 - 2))
tw = (x1_bar - x2_bar) / (sg * np.sqrt(1 / n1 + 1 / n2))
tw
```

```
np.float64(5.591470501655584)
```

Der numerische Wert der Teststatistik ist 5,59147.

Um den p -Wert zu berechnen, wenden wir die Funktion `t.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet.

$$df = n_1 + n_2 - 2 = 50 + 50 - 2 = 98$$

```
# Berechne den p-Wert
df = n1 + n2 - 2
p = 1 - t.cdf(tw, df=df)
p
```

```
np.float64(1.0169262409931434e-07)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,01; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 1%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 1,0169 \cdot 10^{-7}$; Bei einem Signifikanzniveau von 1% lassen die Daten den Schluss zu, dass das Durchschnittsgehalt männlicher Studenten höher ist als das Durchschnittsgehalt weiblicher Studenten.

Hypothesentests in Python

Wir haben soeben manuell einen Zweistichproben- t -Test für unabhängige Stichproben in Python durchgeführt. Bitte beachten Sie jedoch, dass wir die volle Leistungsfähigkeit von Python nutzen können, um das gleiche Ergebnis wie oben in nur einer Zeile Code zu erhalten!

Um einen Zweistichproben- t -Test für unabhängige Stichproben in Python durchzuführen, verwenden wir die Funktion `ttest_ind()`. Wir geben zwei Vektoren als Dateneingabe an und setzen `equal_var=True`, um explizit anzugeben, dass wir die gepoolte Version des t -Tests anwenden, und wir setzen das Argument `alternative` auf `alternative="greater"`, um H_A widerzuspiegeln:
 $\mu_1 > \mu_2$

```
ttest_ind(male_sample, female_sample, equal_var=True, alternative="greater")
```

```
TtestResult(statistic=np.float64(5.591470501655584), pvalue=np.float64(1.01692624127)
```

Ein großer Erfolg! Vergleichen Sie die Ausgabe der Funktion `ttest_ind()` mit unserem Ergebnis von oben. Sie stimmen perfekt überein! Auch hier können wir schlussfolgern, dass bei einem Signifikanzniveau von 1% die Daten einen sehr starken Hinweis darauf liefern, dass das Durchschnittsgehalt der männlichen Absolventen höher ist als das der weiblichen Absolventen.

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
The autoreload extension is already loaded. To reload it, use:  
  %reload_ext autoreload  
The lab_black extension is already loaded. To reload it, use:  
  %reload_ext lab_black  
-----  
Working on the host: imarevic-pc  
-----  
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]  
-----  
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

```
<Figure size 640x480 with 0 Axes>
```

Inferenz für zwei Grundgesamtheitsmittelwerte bei unabhängigen Stichproben; $s_1 \neq s_2$

```
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from random import sample  
from scipy.stats import t, ttest_ind  
import statsmodels.api as smi
```

In den Fällen, in denen die Mittelwerte zweier Grundgesamtheiten getestet werden sollen und die Standardabweichungen zwischen den beiden Grundgesamtheiten unterschiedlich sind, wird der so genannte **nicht gepoolte t -Test** oder [**Welch's \$t\$ -Test**](#) angewendet.

Der **Welch's t -Test** ist dem **2-Stichproben t -Test für unabhängige Stichproben** sehr ähnlich, mit Ausnahme der Teststatistik t und bei der Berechnung der Freiheitsgrade (df). Die Teststatistik nutzt nicht s_g , die gewichtete Standardabweichung, und wird geschrieben als

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}.$$

Der Nenner der obigen Gleichung ist der Schätzer der Standardabweichung von $\bar{x}_1 - \bar{x}_2$ und gegeben durch

$$s_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}.$$

Die Teststatistik t folgt einer t -Verteilung und die Freiheitsgrade (df) sind gegeben durch

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1-1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2-1}}.$$

Runden Sie die Freiheitsgrade auf die nächste ganze Zahl ab, wenn Sie Wahrscheinlichkeitstabellen verwenden.

Der **nicht gepoolte t -Test** ist robust gegenüber mäßigen Verstößen gegen die Normalverteilungsannahme, aber er ist weniger robust gegenüber Ausreißern (s.403).

Intervall-Schätzung von $\mu_1 - \mu_2$

Das $100(1 - \alpha)\%$ -Konfidenzintervall für $\mu_1 - \mu_2$ ist

$$(\bar{x}_1 - \bar{x}_2) \pm t^* \times \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

wobei der Wert von t aus der t -Verteilung für das gegebene Konfidenzniveau ermittelt wird. Die Freiheitsgrade (df) und die Standardabweichung ($s_{\bar{x}_1 - \bar{x}_2}$) ergeben sich aus der obigen Gleichung.

Der Welch's t -Test: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den Welch's t -Test in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `Name`, `Geschlecht`, `Alter`, `Größe`, `Gewicht`, `Religion`, `nc_score`, `Semester`, `Hauptfach`, `Nebenfach`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Zur Veranschaulichung des **nicht gepoolten t -Tests** untersuchen wir das mittlere Jahresgehalt (in Euro) von Absolventinnen in Abhängigkeit von ihrem Studienfach. Die erste Grundgesamtheit besteht aus Studentinnen mit dem Hauptfach Politikwissenschaft und die zweite Grundgesamtheit aus Studentinnen mit dem Hauptfach Sozialwissenschaften. Wir wollen testen, **ob es einen Unterschied im mittleren Gehalt dieser beiden Gruppen gibt.**

Vorbereitung der Daten

Wir beginnen mit der Datenaufbereitung.

- Wir unterteilen den Datensatz anhand der Variablen `gender` und `graduated`.
- Dann unterteilen wir die Teilmenge in Absolventen der Politikwissenschaften und der Sozialwissenschaften (Variable `major`).
- Dann ziehen wir aus jeder Gruppe 50 Studenten und extrahieren die Variable von Interesse, das mittlere Jahresgehalt (in Euro), das in der Spalte `salary` gespeichert ist. Wir ordnen diese beiden Vektoren den Variablen `PS` und `SS` zu.

```

female_graduates = students.loc[
    (students["graduated"] == 1) & (students["gender"] == "Female")
]
subset_PS = female_graduates.loc[female_graduates["major"] == "Political Science"]
subset_SS = female_graduates.loc[female_graduates["major"] == "Social Sciences"]

PS = subset_PS["salary"].sample(n=50, random_state=2)
SS = subset_SS["salary"].sample(n=50, random_state=2)

```

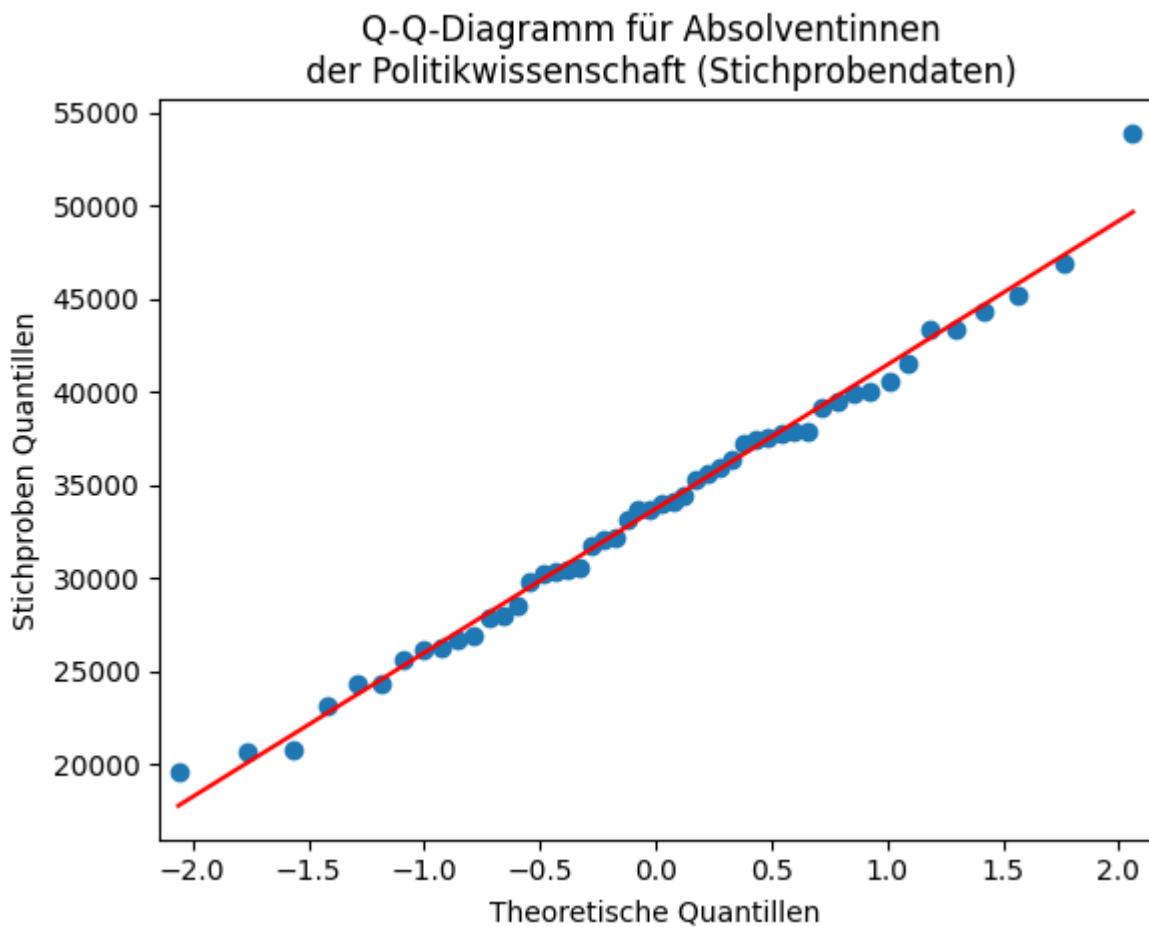
Außerdem überprüfen wir, ob die Daten normalverteilt sind, indem wir ein Q-Q-Diagramm erstellen. In Python können wir die Funktion `qqplot()` verwenden, um Q-Q-Plots zu erstellen.

```

# Erzeuge Q-Q Plot
qqp = smi.qqplot(PS, line="r")
ax = qqp.gca()
ax.set_title(
    "Q-Q-Diagramm für Absolventinnen \n der Politikwissenschaft (Stichprobendaten)"
)
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")

```

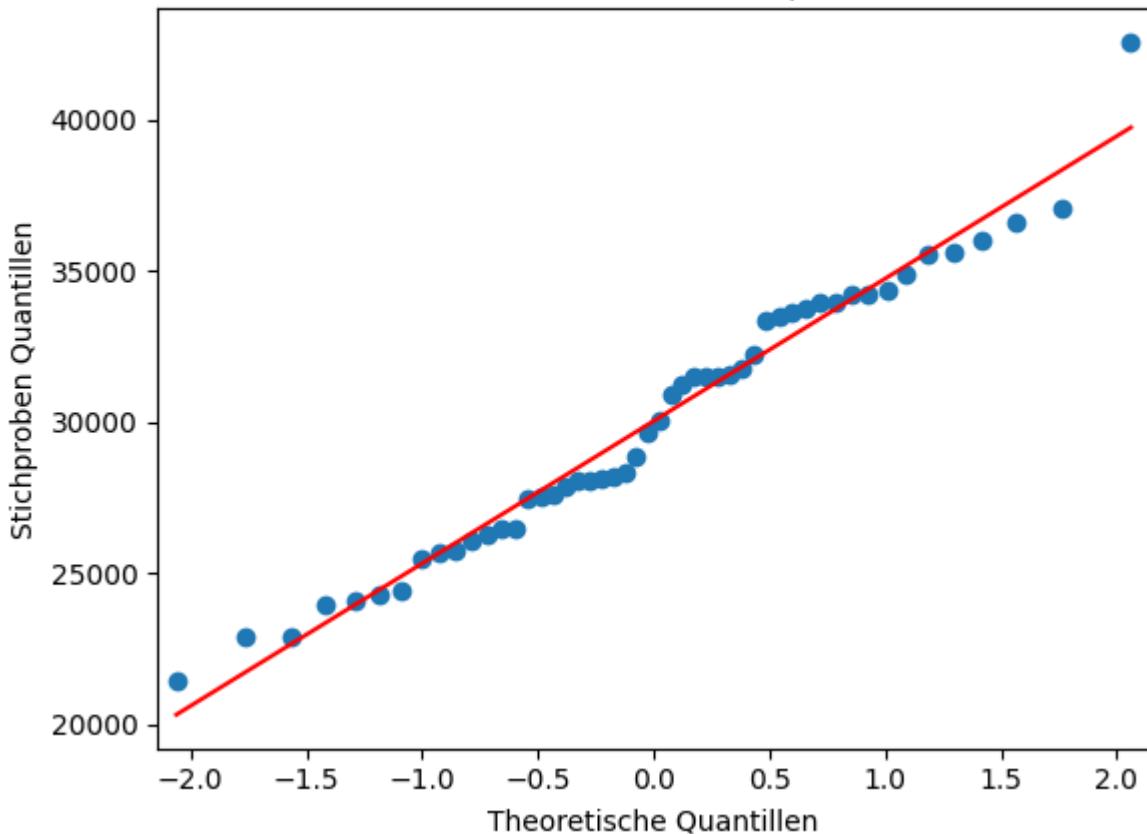
```
Text(0, 0.5, 'Stichproben Quantillen')
```



```
# Erzeuge Q-Q Plot
qqp2 = smi.qqplot(ss, line="r")
ax = qqp2.gca()
ax.set_title(
    "Q-Q-Diagramm für Absolventinnen \n der Sozialwissenschaft (Stichprobendaten)"
)
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")
```

```
Text(0, 0.5, 'Stichproben Quantillen')
```

Q-Q-Diagramm für Absolventinnen
der Sozialwissenschaft (Stichprobendaten)



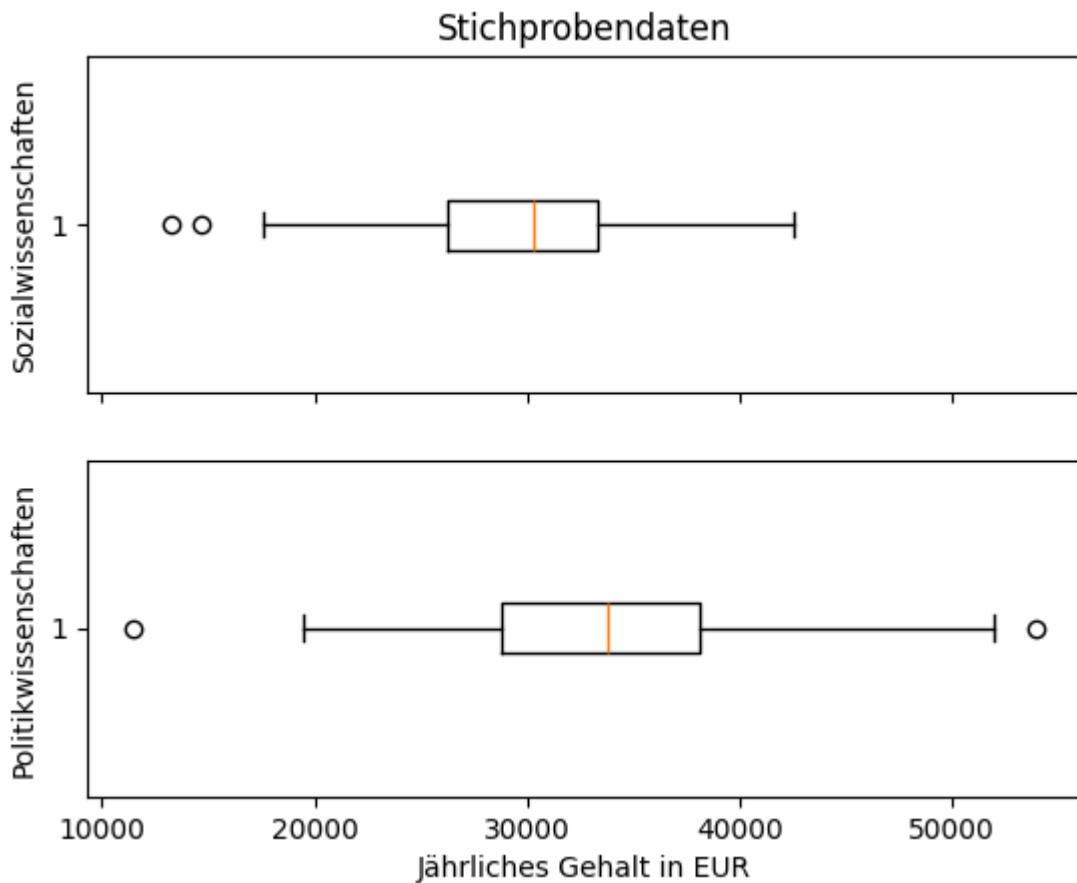
Wir sehen, dass die Daten beider Stichproben ungefähr auf einer Geraden liegen.

Gehen wir davon aus, dass die Daten des `students` Datensatzes eine gute Annäherung an die Grundgesamtheit darstellen. Dann können wir visuell überprüfen, ob sich die Standardabweichungen der beiden Grundgesamtheiten tatsächlich voneinander unterscheiden, indem wir ein Boxplot aufzeichnen.

```
fig1, (ax1, ax2) = plt.subplots(nrows=2, sharex=True)

ax1.set_title("Stichprobendaten")
ax1.set_ylabel("Sozialwissenschaften", size=10)
ax2.set_ylabel("Politikwissenschaften", size=10)
ax2.set_xlabel("Jährliches Gehalt in EUR")
ax2.boxplot(subset_PS["salary"], vert=False)
ax1.boxplot(subset_SS["salary"], vert=False)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x79a96ddca8c0>,
   <matplotlib.lines.Line2D at 0x79a96ddcab60>],
 'caps': [<matplotlib.lines.Line2D at 0x79a96ddcae00>,
   <matplotlib.lines.Line2D at 0x79a96ddcb0a0>],
 'boxes': [<matplotlib.lines.Line2D at 0x79a96ddca620>],
 'medians': [<matplotlib.lines.Line2D at 0x79a96ddcb340>],
 'fliers': [<matplotlib.lines.Line2D at 0x79a96ddcb5e0>],
 'means': []}
```



Auf der Grundlage des grafischen Auswertungsansatzes kommen wir zu dem Schluss, dass die Daten annähernd normalverteilt sind und dass die Standardabweichungen voneinander abweichen.

Überprüfung der Hypothesen

Erinnern Sie sich an die Forschungsfrage. **Lassen die Daten den Schluss zu, dass sich das mittlere Jahresgehalt von Absolventinnen mit einem Hauptfach in Politikwissenschaft vom mittleren Jahresgehalt von Absolventinnen mit einem Hauptfach in Sozialwissenschaften unterscheidet?**

Um den nicht zusammengefassten t -Test durchzuführen, folgen wir dem schrittweisen Durchführungsverfahren für Hypothesentests.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass das durchschnittliche Jahresgehalt von Absolventinnen mit dem Hauptfach Politikwissenschaft (μ_1) gleich dem durchschnittlichen Jahresgehalt von Absolventinnen mit dem Hauptfach Sozialwissenschaften (μ_2) ist.

$$H_0 : \mu_1 = \mu_2$$

Alternative Hypothese

$$H_A : \mu_1 \neq \mu_2$$

Diese Formulierung führt zu einem zweiseitigen Hypothesentest.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Wir erinnern uns an die Gleichungen für die Teststatistik von oben.

```
# Berechne die Teststatistik
n1 = len(PS)
n2 = len(SS)
s1 = np.std(PS, ddof=1)
s2 = np.std(SS, ddof=1)
x1_bar = np.mean(PS)
x2_bar = np.mean(SS)

tw = (x1_bar - x2_bar) / (np.sqrt(s1**2 / n1 + s2**2 / n2))
tw
```

```
np.float64(3.054456171492353)
```

Der numerische Wert der Teststatistik ist 3,05446.

Um den p -Wert zu berechnen, wenden wir die Funktion `t.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet.

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2} \right)^2}{\frac{\left(\frac{s_1^2}{n_1} \right)^2}{n_1-1} + \frac{\left(\frac{s_2^2}{n_2} \right)^2}{n_2-1}},$$

```
# Berechne df
df_numerator = (s1**2 / n1 + s2**2 / n2) ** 2
df_denominator = (s1**2 / n1) ** 2 / (n1 - 1) + (s2**2 / n2) ** 2 / (n2 - 1)
df = df_numerator / df_denominator
df
```

```
np.float64(81.31178520706362)
```

```
# Berechne p-Wert
# wir verwenden einen zweiseitigen Test
upper = 1 - t.cdf(abs(tw), df=df)
lower = t.cdf(-abs(tw), df=df)
p = upper + lower
p
```

```
np.float64(0.0030479813074539763)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,05; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 5%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 0,003047987$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass sich das durchschnittliche Jahresgehalt von Absolventinnen der Politikwissenschaft vom durchschnittlichen Jahresgehalt von Absolventinnen der Sozialwissenschaften unterscheidet.

Hypothesentests in Python

Wir haben gerade einen Welch's t -Test in Python manuell durchgeführt. Jetzt nutzen wir die volle Leistung von Python, um das gleiche Ergebnis wie oben mit nur einer Zeile Code zu erhalten!

Um einen Welch's t -Test in Python durchzuführen, verwenden wir die Funktion `ttest_ind()`. Wir geben zwei Vektoren als Dateneingabe an und setzen `equal_var = False`, um explizit anzugeben, dass wir die nicht gepoolte Version des t -Tests anwenden. Das Argument `alternative` muss nicht gesetzt werden, da der Standardwert unserer Alternativhypothese entspricht $H_A : \mu_1 \neq \mu_2$

```
ttest_ind(PS, SS, equal_var=False)
```

```
TtestResult(statistic=np.float64(3.054456171492353), pvalue=np.float64(0.00304798136))
```

Stark! Vergleichen Sie die Ausgabe der Funktion `ttest_ind()` mit unserem Ergebnis von oben. Sie stimmen perfekt überein! Auch hier können wir schlussfolgern, dass die Daten bei einem Signifikanzniveau von 5% einen sehr starken Hinweis darauf liefern, dass sich das durchschnittliche Jahresgehalt von Absolventinnen der Politikwissenschaft vom durchschnittlichen Jahresgehalt von Absolventinnen der Sozialwissenschaften unterscheidet.

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

The autoreload extension is already loaded. To reload it, use:

```
%reload_ext autoreload
```

The lab_black extension is already loaded. To reload it, use:

```
%reload_ext lab_black
```

Working on the host: imarevic-pc

Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]

Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti

<Figure size 640x480 with 0 Axes>

Inferenz für zwei Grundgesamtheitsmittelwerte bei gepaarten Stichproben

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
from scipy.stats import t, ttest_rel
import statsmodels.api as smi
```

Wenden wir uns nun einem Hypothesentestverfahren für die Differenz zwischen zwei Grundgesamtheitsmittelwerte zu, wenn die Stichproben **abhängig** sind. Wenn beispielsweise zwei

Datenwerte aus derselben Quelle (oder demselben Teilsystem) stammen, werden diese als **gepaarte** oder **abhängige** Stichproben bezeichnet.

Sehr häufig werden diese Verfahren für die **Before-After-Control-Impact (BACI)** eingesetzt. Stellen Sie sich einen Fall vor, in dem Sie die Wirksamkeit eines Filtersystems zur Entfernung von Luftschadstoffen, die von einer Fabrik freigesetzt werden, bewerten sollen. In diesem Fall besteht eine Grundgesamtheit aus Messungen der Luftqualität vor der Einführung oder Erneuerung des Filtersystems und die andere Grundgesamtheit aus Messungen der Luftqualität nach der Installation des neuen Filtersystems. In diesem Fall hat man es mit gepaarten Stichproben zu tun, da die beiden Datensätze von derselben Quelle, der Fabrik, erhoben werden.

Bei gepaarten Stichproben wird die Differenz zwischen den Datenwerten der beiden Stichproben mit d bezeichnet, oft auch als **gepaarte Differenz** bezeichnet. Beachten Sie, dass der Stichprobenumfang n für jede Stichprobe gleich ist. Der Mittelwert der gepaarten Differenzen für die Stichproben wird als \bar{d} bezeichnet.

$$\bar{d} = \frac{\sum d}{n}$$

Die Standardabweichung der gepaarten Unterschiede für zwei Stichproben, s_d , wird wie folgt berechnet

$$s_d = \sqrt{\frac{\sum d^2 - \frac{(\sum d)^2}{n}}{n - 1}}$$

Angenommen, die gepaarte Differenzvariable d ist normalverteilt, dann wird die gepaarte t-Statistik wie folgt ausgedrückt

$$t = \frac{\bar{d} - (\mu_1 - \mu_2)}{\frac{s_d}{\sqrt{n}}},$$

was sich vereinfacht zu

$$t = \frac{\bar{d}}{\frac{s_d}{\sqrt{n}}},$$

wenn $\mu_1 - \mu_2 = 0$. Die Teststatistik t für gepaarte Stichproben folgt einer t -Verteilung mit $df = n - 1$.

Intervall-Schätzung von μ_d

Das $100(1 - \alpha)\%$ -Konfidenzintervall für μ_d ist

$$\bar{d} \pm t \times \frac{s_d}{\sqrt{n}}$$

wobei sich der Wert von t aus der t -Verteilung für das gegebene Konfidenzniveau und $n - 1$ Freiheitsgrade ergibt.

Der 2-Stichproben t -Test für gepaarte (abhängige) Stichproben: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den **gepaarten t -Test** in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id, name, gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary`.

Um den gepaarten t -Test für abhängige Stichproben zu veranschaulichen, **interessieren wir uns für die Frage, ob ein Online-Tutorium zum Erlernen von Statistik den Studenten hilft, ihre Noten zu verbessern**. Es gibt drei Variablen im Datensatz der Studenten, die von Interesse sind. Die Variable

`online_tutorial` ist eine binäre Variable, die den Wert 1 annimmt, wenn der Student das Online-Tutorium zum Thema Statistik absolviert hat, und ansonsten den Wert 0. Die Variablen `score1` und `score2` geben die Noten (0 – 100) für zwei Klausuren in Mathematik und Statistik an. Je höher der Wert ist, desto besser hat der jeweilige Studierende abgeschnitten. Bitte beachten Sie, dass die erste Prüfung stattfindet, bevor die Studierenden das Online-Tutorium zum Thema Statistik besucht haben. Die Teilnahme am Online-Tutorium zur Statistik ist nicht verpflichtend, die beiden Prüfungen sind jedoch für alle Studierenden obligatorisch. Die erste Prüfung (`score1`) findet zu Beginn des 3. Semesters statt, die zweite Prüfung (`score2`) am Ende des 3. Semesters.

Im Wesentlichen sind zwei Forschungsfragen von Interesse. **Erstens wollen wir untersuchen, ob die Gruppe der Studierenden, die das Online-Tutorial zum Statistiklernen besucht hat, in der zweiten Prüfung besser abschneidet als in den ersten Prüfungen. Zweitens wird untersucht, wie die Gruppe der Studierenden, die nicht am Online-Tutorial für Statistik teilgenommen hat, bei beiden Prüfungen abgeschnitten hat.**

Vorbereitung der Daten

Wir beginnen mit der ersten Forschungsfrage und konzentrieren uns auf die Studierenden, die das Online-Tutorium zum Thema Statistik besucht haben.

Für die Datenaufbereitung unterteilen wir den Datensatz anhand der Variable `online_tutorial`, die angibt, ob der Studierende das Tutorium besucht hat oder nicht (1=ja, 0=nein). Dann ziehen wir eine Zufallsstichprobe von 65 Studenten aus dem Datensatz und extrahieren die beiden Variablen von Interesse, `score1` und `score2`. Wir speichern jede von ihnen in einem Vektor mit den Namen `score1_sample` und `score2_sample`.

```
tutorial = students.loc[students["online_tutorial"] == 1]
n = 65
score1_sample = tutorial["score1"].sample(n, random_state=1)
score2_sample = tutorial["score2"].sample(n, random_state=1)
```

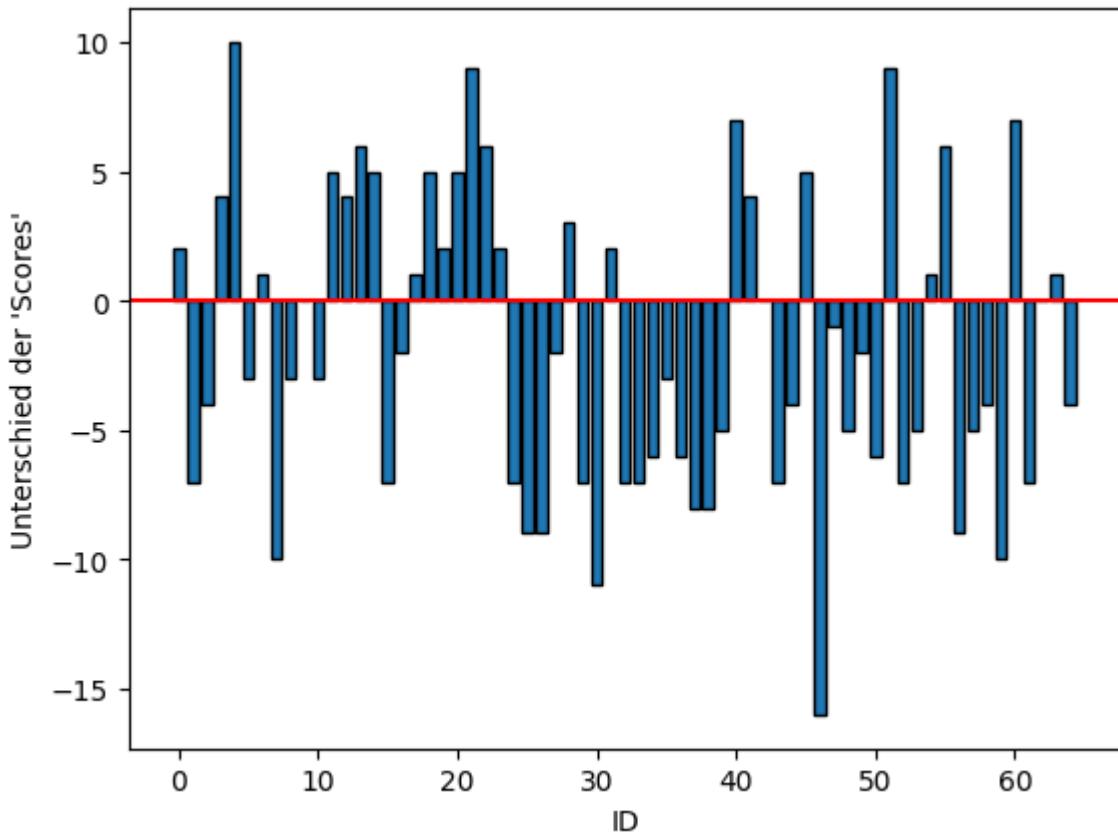
Nun berechnen wir die gepaarten Differenzen, d , und stellen sie dar.

```

d = score1_sample - score2_sample
x = np.arange(n)
fig, ax = plt.subplots()
ax.axhline(y=0, color="r")
ax.bar(x, d, edgecolor="k")
ax.set_xlabel("ID")
ax.set_ylabel("Unterschied der 'Scores'")

```

Text(0, 0.5, "Unterschied der 'Scores'")

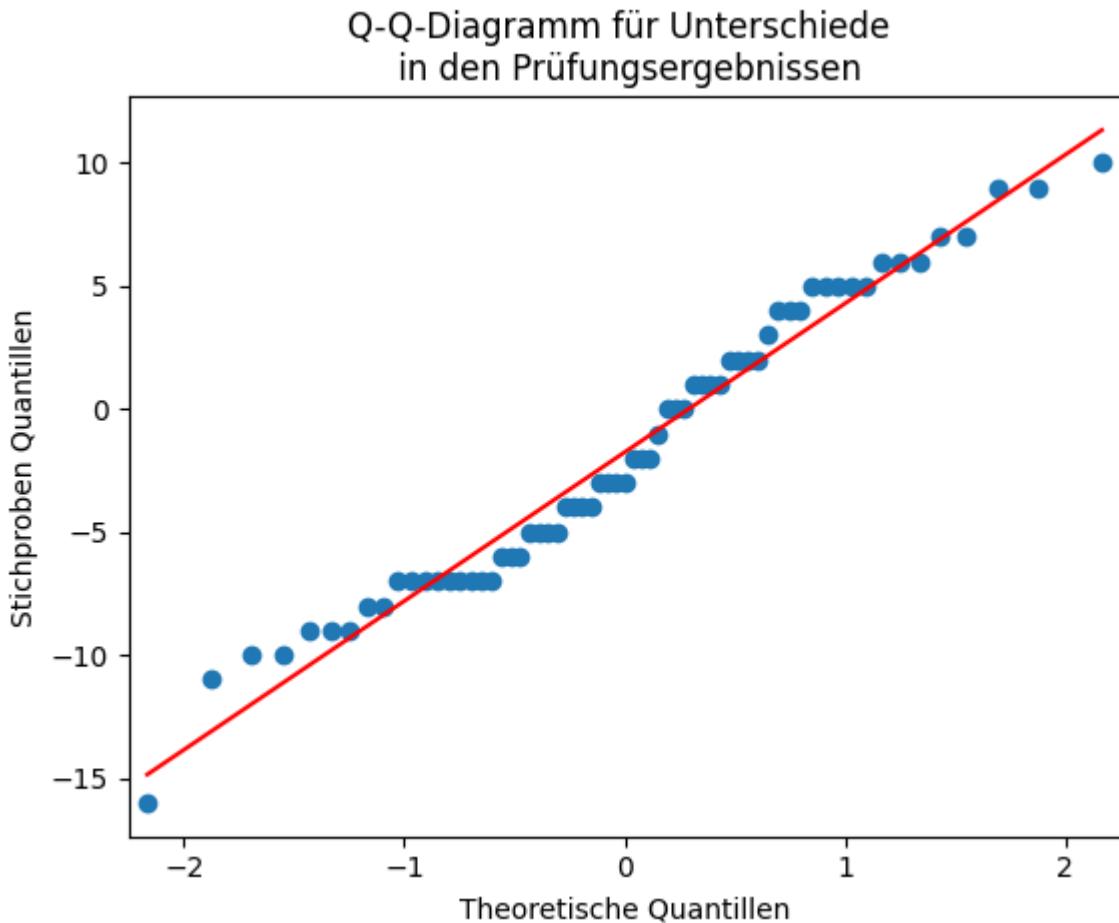


Das Diagramm sieht wie erwartet aus. Einige Studierende schneiden in der ersten Prüfung besser ab als in der zweiten Prüfung und umgekehrt.

Um die Normalverteilungsannahme zu überprüfen, stützen wir uns wiederum auf eine visuelle Inspektion eines [Q-Q-Plots](#). Wenn die Variable normalverteilt ist, sollte das Q-Q-Diagramm ungefähr linear sein. In Python können wir die Funktion `qqplot()` verwenden, um Q-Q-Plots zu erstellen.

```
# Erzeuge Q-Q Plot
qqp = smi.qqplot(d, line="r")
ax = qqp.gca()
ax.set_title("Q-Q-Diagramm für Unterschiede \n in den Prüfungsergebnissen")
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")
```

```
Text(0, 0.5, 'Stichproben Quantillen')
```



Die Daten sind nicht sehr genau und etwas verrauscht, aber sie scheinen ungefähr normalverteilt zu sein.

Wir berechnen weiter \bar{d} den Mittelwert der gepaarten Differenzen

$$\bar{d} = \frac{\sum d}{n},$$

und s_d , die Standardabweichung der gepaarten Unterschiede für zwei Stichproben

$$s_d = \sqrt{\frac{\sum d^2 - \frac{(\sum d)^2}{n}}{n - 1}}.$$

```
# gepaarte Differenz
d_bar = sum(d) / len(d)
d_bar
```

-1.7538461538461538

```
# Standardabweichung
s_d = np.sqrt((sum(d**2) - (sum(d) ** 2 / len(d))) / (n - 1))
s_d
```

np.float64(5.836391139947831)

Überprüfung der Hypothesen

Jetzt sind wir bereit, den **gepaarten *t*-Test** anzuwenden. Erinnern Sie sich an unsere erste Forschungsfrage: **Lassen die Daten den Schluss zu, dass sich die durchschnittlichen Prüfungsergebnisse verbessern, wenn die Studierenden ein Online-Tutorial zum Thema Statistik besuchen?**

Wir folgen dem schrittweisen Implementierungsverfahren für Hypothesentests.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass es keinen Unterschied im Mittelwert der Prüfungsnoten der einen Prüfung im Vergleich zur anderen gibt.

$$H_0 : \mu_1 = \mu_2$$

Erinnern Sie sich daran, dass die Formulierung der Alternativhypothese vorgibt, ob wir einen zweiseitigen, einen links- oder einen rechtsseitigen Hypothesentest durchführen.

Alternative Hypothese

$$H_A : \mu_1 < \mu_2$$

Diese Formulierung führt zu einem linksseitigen Hypothesentest und besagt, dass die Studenten im Durchschnitt bei der zweiten Prüfung besser abschneiden.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die obige Gleichungsform:

$$t = \frac{\bar{d} - (\mu_1 - \mu_2)}{\frac{s_d}{\sqrt{n}}}$$

Wenn H_0 wahr ist, dann ist $\mu_1 - \mu_2 = 0$ und somit vereinfacht sich die Gleichung zu

$$t = \frac{\bar{d}}{\frac{s_d}{\sqrt{n}}}.$$

```
# Berechne Teststatistik

# gepaarte Differenz
d_bar = sum(d) / len(d)

# Standardabweichung
s_d = np.sqrt((sum(d**2) - (sum(d) ** 2 / len(d))) / (n - 1))

# Teststatistik
tw = d_bar / (s_d / np.sqrt(len(d)))
tw
```

```
np.float64(-2.4227231184673745)
```

Der numerische Wert der Teststatistik ist $-2,422723$.

Um den p -Wert zu berechnen, wenden wir die Funktion `t.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet.

$$df = n - 1 = 64$$

```
# Berechne p-Wert
df = len(d) - 1
p = t.cdf(tw, df=df)
p
```

```
np.float64(0.009122595661033035)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von $0,05$; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 5% -Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 0,009123$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass sich die Prüfungsnoten der Studierenden nach der Teilnahme an einem Online-Tutorium für Statistik verbessern.

Hypothesentests in Python

Wir haben soeben manuell einen gepaarten t -Test in Python durchgeführt. Das ist gut, aber jetzt nutzen wir die ganze Macht von Python, um das gleiche Ergebnis wie oben mit nur einer Zeile Code zu erhalten!

Um einen gepaarten t -Test in Python durchzuführen, verwenden wir die Funktion `ttest_rel()`. Wir geben zwei Vektoren als Dateneingabe an und wir setzen das Argument `alternative` auf `'less'`, um auf $HA : \mu_1 < \mu_2$ zu testen

```
ttest_rel(score1_sample, score2_sample, alternative="less")
```

```
TtestResult(statistic=np.float64(-2.4227231184673745), pvalue=np.float64(0.009122595
```

Großartig! Vergleichen Sie die Ausgabe der Funktion `ttest_rel()` mit unserem Ergebnis von oben. Sie stimmen perfekt überein! Auch hier können wir schlussfolgern, dass die Daten bei einem Signifikanzniveau von 5% einen sehr starken Hinweis darauf liefern, dass sich die Prüfungsnoten der Studenten nach der Teilnahme an einem Online-Tutorium für Statistik verbessern.

Bevor wir fortfahren, muss noch eine Forschungsfrage beantwortet werden. Was ist, wenn es andere Gründe für die besseren Noten in der zweiten Prüfung gibt? Was ist, wenn die zweite Prüfung viel einfacher war? Was wäre, wenn die Studenten einen tollen Dozenten hatten und sich dadurch im Laufe des Semesters verbessert haben? Wir testen diese Hypothese, indem wir einen gepaarten t -Test durchführen, und zwar explizit für die Studierenden, die nicht am Online-Tutorium für Statistik teilgenommen haben. Da wir mit der Python-Maschinerie bestens vertraut sind, führen wir einen gepaarten t -Test mit nur wenigen Zeilen Code durch.

```
no_tutorial = students.loc[students["online_tutorial"] == 0]

n = 65

score1_no_tutorial = no_tutorial["score1"].sample(n, random_state=1)
score2_no_tutorial = no_tutorial["score2"].sample(n, random_state=1)

# führe paired t-test durch
statistics, pvalue = ttest_rel(
    score1_no_tutorial, score2_no_tutorial, nan_policy="omit", alternative="less"
)
ttest_rel(score1_no_tutorial, score2_no_tutorial, nan_policy="omit", alternative="less")
```

```
TtestResult(statistic=np.float64(2.1307181194375913), pvalue=np.float64(0.9771439852)
```

```
pvalue <= alpha
```

```
np.False_
```

Der p -Wert ist größer als das angegebene Signifikanzniveau von 0,05; wir verwerfen H_0 nicht. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant und liefern keinen ausreichenden Beweis gegen die Nullhypothese.

Bei einem Signifikanzniveau von 5% liefern die Daten keine ausreichende Evidenz für die Schlussfolgerung, dass sich die Prüfungsnoten der Studierenden, die nicht am Online-Tutorium teilgenommen haben, verbessert haben.

Inferenz für die Standardabweichung der Grundgesamtheit

Bisher haben wir Methoden der Inferenzstatistik erörtert, die sich auf Aussagen über einen oder mehrere Grundgesamtheitsmittelwerte konzentrieren. In den folgenden Abschnitten betrachten wir Methoden der Inferenzstatistik, die Aussagen über die Varianzen (oder Standardabweichungen) der Grundgesamtheit liefern. Wir erinnern daran, dass die Varianz (σ^2) und die Standardabweichung (σ) ein Maß für die Streuung und Variabilität einer Variablen sind.

Inferenz für die Standardabweichung einer Grundgesamtheit

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
import statsmodels.api as smi
```

Inferenz für eine Grundgesamtheitsstandardabweichung basiert auf der [Chi-Quadrat \(\$\chi^2\$ \)-Verteilung](#). Eine χ^2 -Verteilung ist eine rechtsschiefe Wahrscheinlichkeitsdichtekurve. Die Form der χ^2 -Kurve wird durch ihre Freiheitsgrade (df) bestimmt.

► Show code cell content

Um einen Hypothesentest für eine Populationsstandardabweichung durchzuführen, wird ein χ^2 -Wert mit einer bestimmten Fläche unter einer χ^2 -Kurve in Beziehung gesetzt. Entweder wir ziehen eine χ^2 -Tabelle, um diesen Wert nachzuschlagen, oder wir machen von der Python-Maschinerie Gebrauch.

Bei gegebenen α , wobei α einer Wahrscheinlichkeit zwischen 0 und 1 entspricht, bezeichnet χ^2_α den χ^2 -Wert, der die Fläche α zu seiner Rechten unter einer χ^2 -Kurve hat.

► Show code cell content

Intervall-Schätzung von σ

Das $100(1 - \alpha)\%$ -Konfidenzintervall für σ beträgt

$$\sqrt{\frac{n-1}{\chi^2_{\alpha/2}}} \leq \sigma \leq \sqrt{\frac{n-1}{\chi^2_{1-\alpha/2}}}$$

χ^2 -Test auf Standardabweichung

Das Hypothesentestverfahren für eine Standardabweichung wird als **χ^2 -Test auf Standardabweichung** bezeichnet. Hypothesentests für Varianzen folgen demselben schrittweisen Verfahren wie Hypothesentests für den Mittelwert.

-
- Schritt 1 Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an.
 - Schritt 2 Legen Sie das Signifikanzniveau, α fest.
 - Schritt 3 Berechnen Sie den Wert der Teststatistik.
 - Schritt 4 Bestimmen Sie den p-Wert.
 - Schritt 5 Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht ablehnen.
 - Schritt 6 Interpretieren Sie das Ergebnis des Hypothesentests.
-

Die Teststatistik für einen Hypothesentest mit der Nullhypothese $H_0 : \sigma = \sigma_0$ für eine normalverteilte Variable ist gegeben durch

$$\chi^2 = \frac{n-1}{\sigma_0^2} s^2$$

wobei n der Stichprobenumfang und s die Standardabweichung der Stichprobendaten ist.

Die Variable folgt einer χ^2 -Verteilung mit $n - 1$ Freiheitsgraden.

Beachten Sie, dass der Test auf eine Standardabweichung χ^2 -Test nicht robust gegenüber Abweichungen von der Normalverteilung ist (Weiss 2010).

χ^2 -Test auf Standardabweichung : Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den χ^2 -Test mit einer Standardabweichung in einer Übung an. Dazu laden wir den **students** Datensatz. Sie können die Datei **students.csv** [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Um den χ^2 -Test auf Standardabweichung zu demonstrieren, untersuchen wir die Streuung der Körpergröße in cm der Studentinnen und vergleichen sie mit der Streuung der Körpergröße aller Studenten (unserer Grundgesamtheit). **Wir wollen testen, ob die Standardabweichung der Körpergröße der Studentinnen kleiner ist als die Standardabweichung der Körpergröße aller Studenten.**

Vorbereitung der Daten

Wir beginnen mit der Datenaufbereitung.

- Zunächst definieren wir die Standardabweichung der Grundgesamtheit. In unserem Beispiel entspricht die Grundgesamtheit der Körpergröße aller 8239 Studenten im Datensatz. Wir berechnen die Standardabweichung für die Variable `height` und weisen ihr den Variablenamen `sigma0` zu.
- Zweitens unterteilen wir den Datensatz anhand der Variable `gender`.
- Drittens nehmen wir eine Stichprobe von 30 Studentinnen und extrahieren die interessierende Statistik, die Standardabweichung der Größe der Studentinnen in unserer Stichprobe.

```
sigma0 = students["height"].std()
sigma0
```

```
np.float64(11.077529134763823)
```

Die Standardabweichung der interessierenden Grundgesamtheit (σ_0) beträgt $\approx 11,08$ cm.

```

female = students.loc[students["gender"] == "Female"]

n = 30
female_sample = female["height"].sample(n=30, random_state=1)
sample_std = female_sample.std()

```

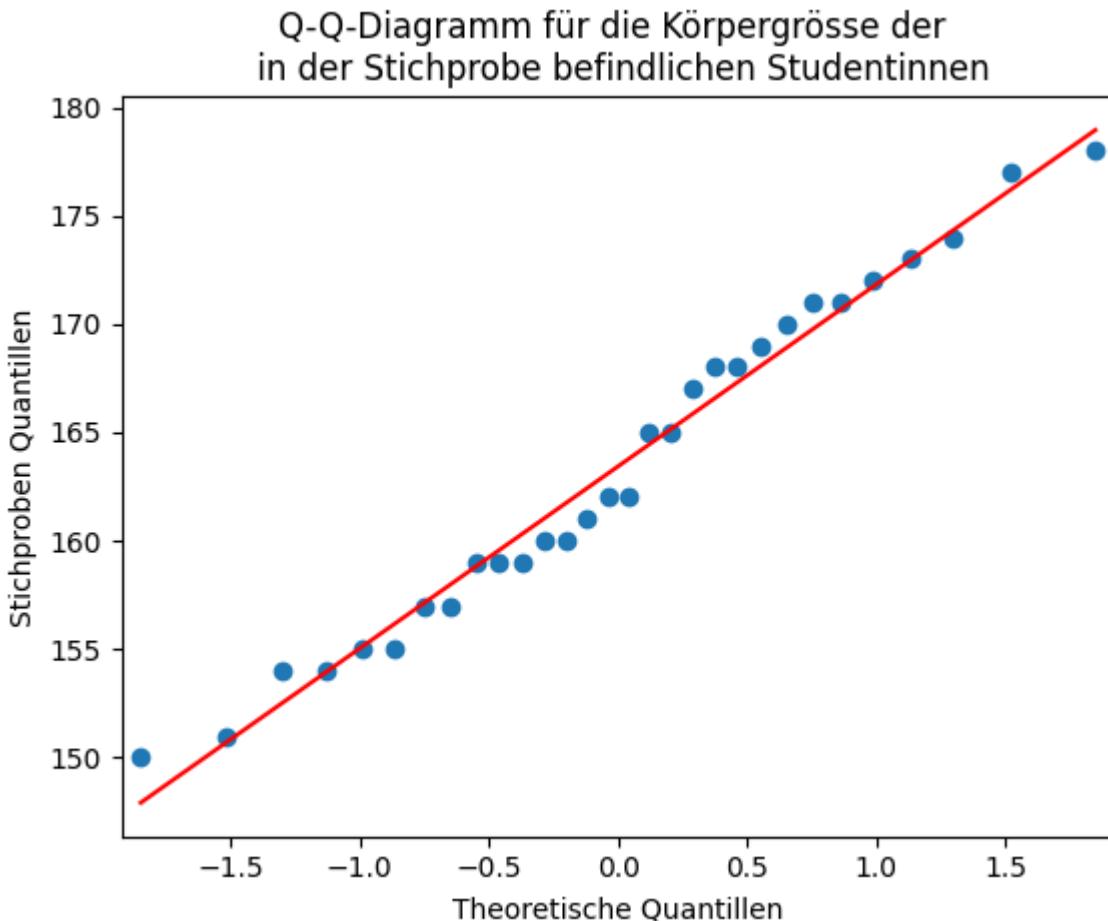
Außerdem überprüfen wir die Normalverteilungsannahme, indem wir ein [Q-Q-Diagramm](#) erstellen. In Python können wir die Funktion `qqplot()` verwenden, um Q-Q-Plots zu erstellen.

```

# Erzeuge Q-Q Plot
qqp = smi.qqplot(female_sample, line="r")
ax = qqp.gca()
ax.set_title(
    "Q-Q-Diagramm für die Körpergrösse der \n in der Stichprobe befindlichen Studenten"
)
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")

Text(0, 0.5, 'Stichproben Quantillen')

```



Wir sehen, dass die Daten ungefähr auf einer Geraden liegen. Auf der Grundlage des grafischen Auswertungsansatzes kommen wir zu dem Schluss, dass die interessierende Variable ungefähr normalverteilt ist.

Überprüfung der Hypothesen

Zur Durchführung des **χ^2 -Tests auf Standardabweichung** folgen wir dem Verfahren der schrittweisen Durchführung von Hypothesentests.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass die Standardabweichung der Körpergröße der Studentinnen (σ) gleich der Standardabweichung der Grundgesamtheit ($\sigma_0 \approx 11,08$ cm) ist.

$$H_0 : \sigma = \sigma_0$$

Alternative Hypothese

$$H_A : \sigma < \sigma_0$$

Diese Formulierung führt zu einem linksseitigen Hypothesentest.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die Gleichung für die Teststatistik von oben:

$$\chi^2 = \frac{n-1}{\sigma_0^2} s^2$$

```
# Berechne Teststatistik
n = len(female_sample)
s_2 = np.var(female_sample, ddof=1)
sigma0_2 = np.var(students["height"], ddof=1)
x2 = ((n - 1) / sigma0_2) * s_2
x2
```

```
np.float64(14.272211660048107)
```

Der numerische Wert der Teststatistik ist 14,27221166.

Um den p -Wert zu berechnen, wenden wir die Funktion `chi2.ppf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet:

$$df = n - 1$$

```
# Berechne df
df = n - 1

# Berechne p-Wert
p = chi2.cdf(x2, df=df)
p
```

```
np.float64(0.010089951471801363)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,05; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 5%-Niveau und liefern starke Beweise gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 0,01008995$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass die Standardabweichung der Körpergröße von Studentinnen weniger als 11 cm beträgt.

Hypothesentests in Python

Wir haben gerade einen χ^2 -Test mit einer Standardabweichung in Python manuell durchgeführt. Meines Wissens bietet Python keine eingebaute Funktion zur Berechnung eines Standardabweichung χ^2 -Test. Wir können jedoch eine solche Funktion selbst implementieren. Unsere Funktion `simple_chi2_test()` nimmt als Eingabe einen Stichprobenvektor `x`, die Standardabweichung der Grundgesamtheit `sigma0`, das Signifikanzniveau `alpha` und die angegebene Methode, `right`, `left` und `two-sided`.

```
def simple_chi2_test(x, sigma0, alpha, method="two-sided"):
    df = len(x) - 1
    v = np.var(x, ddof=1)
    # Berechne Teststatistik
    testchi = df / (sigma0**2) * v

    # linksseitiger Test
    if method == "left":
        p = chi2.cdf(x=testchi, df=df)
    # rechtsseitiger Test
    elif method == "right":
        p = 1 - chi2.cdf(x=testchi, df=df)

    # beidseitiger Test (default)
    else:
        p_upper = 1 - chi2.cdf(x=testchi, df=df)
        p_lower = chi2.cdf(x=testchi, df=df)
        if p_upper * 2 > 1:
            p = p_lower * 2
        else:
            p = p_upper * 2
    # evaluiere p < alpha
    if p < alpha:
        reject = True
    else:
        reject = False

    # Ausgabe
    print("Significance level:", alpha)
    print("Degrees of freedom:", df)
    print("Test statistic:", round(testchi, 4))
    print("p-value:", p)
    print("Reject H0:", reject)
```

Wenden wir nun unsere selbst erstellte Funktion `simple_x2_test()` auf die obigen Beispieldaten an.

```
simple_chi2_test(x=female_sample, sigma0=sigma0, alpha=0.05, method="left")
```

```
Significance level: 0.05
Degrees of freedom: 29
Test statistic: 14.2722
p-value: 0.010089951471801363
Reject H0: True
```

Perfekt! Vergleichen Sie die Ausgabe der Funktion `simple_chi2_test()` mit unserem Ergebnis von oben. Auch hier können wir zu dem Schluss kommen, dass die Daten bei einem Signifikanzniveau von 5% starke Anhaltspunkte dafür liefern, dass die Standardabweichung der Körpergröße von Studentinnen weniger als 11 cm beträgt.

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
The autoreload extension is already loaded. To reload it, use:  
  %reload_ext autoreload  
The lab_black extension is already loaded. To reload it, use:  
  %reload_ext lab_black  
-----  
Working on the host: imarevic-pc  
-----  
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]  
-----  
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

```
<Figure size 640x480 with 0 Axes>
```

Inferenz für Standardabweichungen zweier Grundgesamtheiten

```
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from random import sample  
from scipy.stats import f  
import statsmodels.api as smi
```

In diesem Abschnitt werden Hypothesentests für die Standardabweichungen zweier Grundgesamtheiten behandelt. Oder anders ausgedrückt, wir erörtern Methoden der Inferenz für die Standardabweichungen einer Variablen aus zwei verschiedenen Grundgesamtheiten. Diese Methoden beruhen auf der [F-Verteilung](#), benannt zu Ehren von [Sir Ronald Aylmer Fisher](#).

Die *F*-Verteilung ist eine rechtsschiefe Wahrscheinlichkeitsdichteverteilung mit zwei Formparametern, v_1 und v_2 , den Freiheitsgraden für den Zähler (v_1) und den Freiheitsgraden für den Nenner (v_2).

$$df = (v_1, v_2)$$

Wie bei jeder anderen Dichtekurve entspricht die Fläche unter der Kurve der F -Verteilung den Wahrscheinlichkeiten. Die Fläche unter der Kurve und damit die Wahrscheinlichkeit für ein gegebenes Intervall und einen gegebenen df -Wert wird mittels Software berechnet. Alternativ kann man sie auch in einer [Tabelle](#) nachschlagen. In diesen Tabellen werden im Allgemeinen die Freiheitsgrade für den Zähler (v_1) am oberen Rand angezeigt, während die Freiheitsgrade für den Nenner (v_2) in den äußeren Spalten auf der linken Seite angezeigt werden.

Um einen Hypothesentest für zwei Grundgesamtheitsstandardabweichungen durchzuführen, wird der Wert berechnet, der einer bestimmten Fläche unter einer F -Kurve entspricht, berechnet.

Für gegebenes α , wobei α einer Wahrscheinlichkeit zwischen 0 und 1 entspricht, bezeichnet F_α den Wert, der eine Fläche α zu seiner Rechten unter einer F -Kurve hat.

► Show code cell content

In der obigen Abbildung ergibt $F_{0,05}$ für $df = (9, 14)$ den Wert $\approx 2,6458$.

Eine interessante Eigenschaft von F -Kurven ist die **reziproke Charakteristik**. Sie besagt, dass für eine F -Kurve mit $df = (v_1, v_2)$ der F -Wert mit der Fläche α auf der linken Seite gleich dem Kehrwert des F -Wertes mit der Fläche α auf der rechten Seite für eine F -Kurve mit $df = (v_2, v_1)$ ist (s.282). Übertragen auf das obige Beispiel, bei dem $F_{0,05}$ für $df = (9, 14) \approx 2,6458$ beträgt, bedeutet dies, dass $F_{0,95}$ für $df = (14, 9) ; \frac{1}{2,6458} = 0,378$ beträgt.

► Show code cell content

Intervall-Schätzung von $\sigma_1 - \sigma_2$

Das $100(1 - \alpha)\%$ -Konfidenzintervall für σ beträgt

$$\frac{1}{\sqrt{F_{\alpha/2}}} \times \frac{s_1}{s_2} \leq \sigma \leq \frac{1}{\sqrt{F_{1-\alpha/2}}} \times \frac{s_1}{s_2},$$

wobei s_1 und s_2 die Standardabweichungen der Stichprobe sind.

F-Test für zwei Standardabweichungen

Das Hypothesentestverfahren für die Standardabweichung wird als F-Test für zwei Standardabweichungen bezeichnet. Der Hypothesentest für zwei Standardabweichungen der Grundgesamtheit folgt demselben schrittweisen Verfahren wie andere Hypothesentests.

-
- Schritt 1 Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an.
 - Schritt 2 Legen Sie das Signifikanzniveau, α fest.
 - Schritt 3 Berechnen Sie den Wert der Teststatistik.
 - Schritt 4 Bestimmen Sie den p-Wert.
 - Schritt 5 Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht ablehnen.
 - Schritt 6 Interpretieren Sie das Ergebnis des Hypothesentests.
-

Die Teststatistik für einen Hypothesentest für eine normalverteilte Variable und für unabhängige Stichproben der Größen n_1 und n_2 ist gegeben durch

$$F = \frac{s_1^2/\sigma_1^2}{s_2^2/\sigma_2^2},$$

mit $df = (n_1 - 1, n_2 - 1)$.

Wenn $H_0 : \sigma_1 = \sigma_2$ wahr ist, dann vereinfacht sich die Gleichung zu

$$F = \frac{s_1^2}{s_2^2}$$

F-Test für zwei Standardabweichungen : Ein Beispiel

Um einige praktische Erfahrungen zu sammeln, wenden wir den ***F-Test für zwei Standardabweichungen*** in einer Übung an. Dazu laden wir den [students](#) Datensatz. Sie können

die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id, name, gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary*.

Um den **F-Test für zwei Standardabweichungen** zu zeigen, untersuchen wir die Streuung der Körpergröße in cm der Studentinnen und vergleichen sie mit der Streuung der Körpergröße aller Studenten (unserer Grundgesamtheit). **Wir wollen testen, ob sich die Standardabweichung der Körpergröße der weiblichen Studenten (σ_1) von der Standardabweichung der Körpergröße der männlichen Studenten (σ_2) unterscheidet.**

Vorbereitung der Daten

- Wir unterteilen den Datensatz anhand der Variable `gender`.
- Dann nehmen wir 25 weibliche und 25 männliche Studenten in die Stichprobe auf.
- Dann berechnen wir die Standardabweichungen der interessierenden Variable (Körpergröße in cm) für beide Stichproben und weisen ihnen die Variablen `std_female` und `std_male` zu.

```
# Unterteile Datensatz nach Variable `gender`
male = students.loc[students["gender"] == "Male"]
female = students.loc[students["gender"] == "Female"]

# Entnehme Probe von jeweils 25 Studenten
n = 25
male_sample = male["height"].sample(n=25, random_state=1)
female_sample = female["height"].sample(n=25, random_state=1)
```

```
std_female = np.std(female_sample, ddof=1)
std_female
```

```
np.float64(7.009041779492163)
```

```
std_male = np.std(male_sample, ddof=1)  
std_male
```

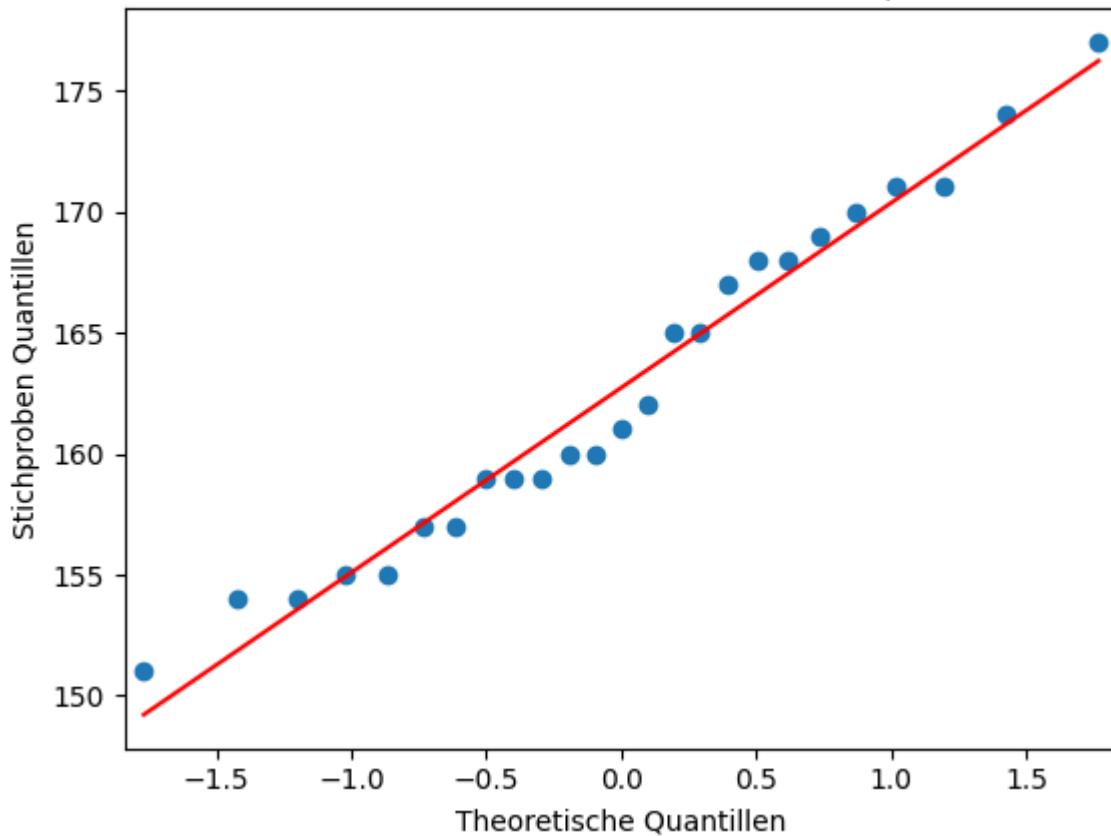
```
np.float64(8.203251387915241)
```

Außerdem überprüfen wir die Normalverteilungsannahme, indem wir ein [Q-Q-Diagramm](#) erstellen. In Python können wir die Funktion `qqplot()` verwenden, um Q-Q-Plots zu erstellen.

```
# Erzeuge Q-Q Plot  
qqp = smi.qqplot(female_sample, line="r")  
ax = qqp.gca()  
ax.set_title(  
    "Q-Q-Diagramm für die Körpergrösse der \n in der weiblichen Studentinnen (Stichprobe)")  
ax.set_xlabel("Theoretische Quantillen")  
ax.set_ylabel("Stichproben Quantillen")
```

```
Text(0, 0.5, 'Stichproben Quantillen')
```

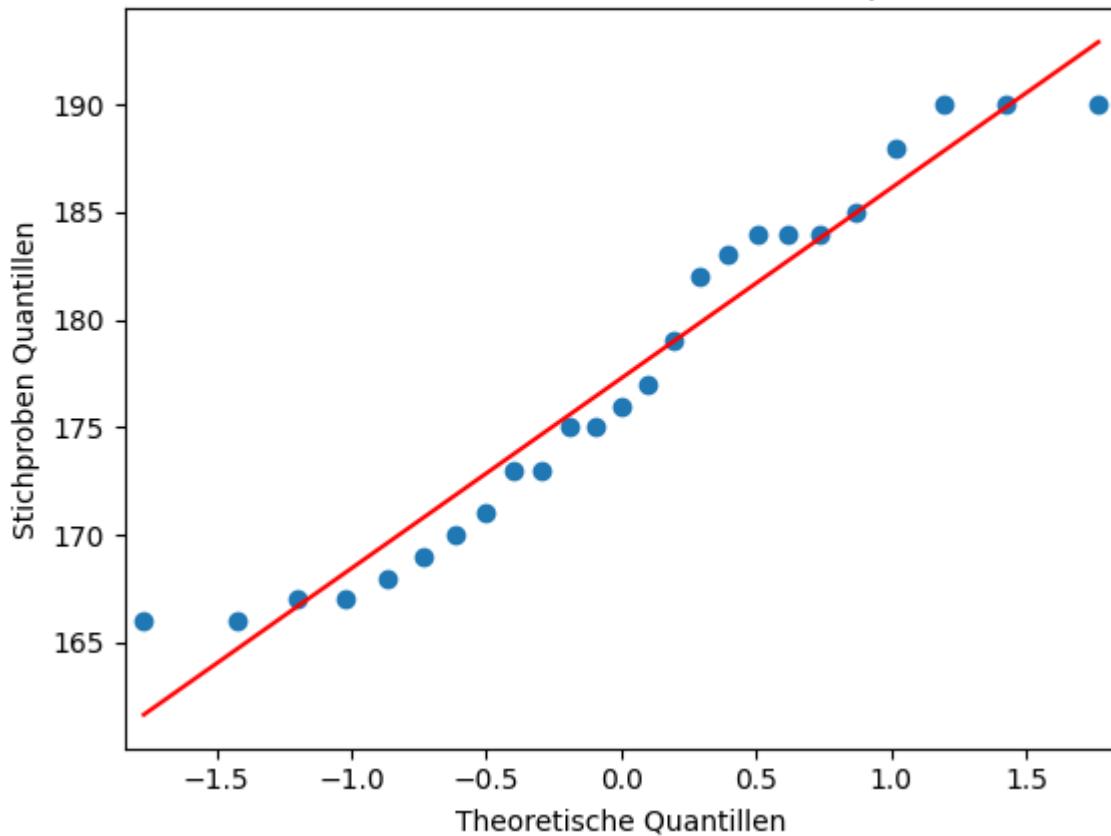
Q-Q-Diagramm für die Körpergrösse der
in der weiblichen Studentinnen (Stichprobe)



```
# Erzeuge Q-Q Plot
qqp = smi.qqplot(male_sample, line="r")
ax = qqp.gca()
ax.set_title(
    "Q-Q-Diagramm für die Körpergrösse der \n in der männlichen Studenten (Stichprobe")
)
ax.set_xlabel("Theoretische Quantillen")
ax.set_ylabel("Stichproben Quantillen")
```

```
Text(0, 0.5, 'Stichproben Quantillen')
```

Q-Q-Diagramm für die Körpergrösse der
in der männlichen Studenten (Stichprobe)



Wir sehen, dass die Daten ungefähr auf einer Geraden liegen. Auf der Grundlage des grafischen Auswertungsansatzes kommen wir zu dem Schluss, dass die interessierende Variable ungefähr normalverteilt ist.

Überprüfung der Hypothesen

Zur Durchführung des **F-Tests für zwei Standardabweichungen** folgen wir dem Verfahren der schrittweisen Durchführung von Hypothesentests.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass die Standardabweichung der Körpergröße der weiblichen Studenten (σ_1) gleich der Standardabweichung der Körpergröße der männlichen Studenten (σ_2) ist.

$$H_0 : \sigma_1 = \sigma_2$$

Alternative Hypothese

$$H_A : \sigma_1 \neq \sigma_2$$

Diese Formulierung führt zu einem zweiseitigen Hypothesentest.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die Gleichung für die Teststatistik von oben:

$$F = \frac{s_1^2}{s_2^2}$$

```
# Berechne die Teststatistik
Ftest = std_female**2 / std_male**2
Ftest
```

```
np.float64(0.7300376461264116)
```

Der numerische Wert der Teststatistik beträgt $\approx 0,73$.

Um den p -Wert zu erhalten, wenden wir die Funktion `f.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet.

$$df = (n_1 - 1, n_2 - 1)$$

```
# Berechne df
df1 = len(female_sample) - 1
df2 = len(male_sample) - 1
```

Den p -Wert berechnen wir nun mit Hilfe der `f.cdf()` Funktion und bilden davon die **Survival Funktion** `1 - f.cdf()`. Dies entspricht einem Vergleich gegenüber der linken Seite der F-Verteilung.

```
p = 1 - f.cdf(Ftest, df1, df2)
print(f"p-value: {p}")
print(f"Sig. result: {p<alpha}")
```

```
p-value: 0.7767376526940193
Sig. result: False
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.False_
```

Der p -Wert ist größer als das angegebene Signifikanzniveau von 0,05; wir verwerfen H_0 nicht. Die Testergebnisse sind auf dem 5%-Niveau statistisch signifikant und liefern keinen ausreichenden Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 0,77673765$. Bei einem Signifikanzniveau von 5% liefern die Daten keine ausreichenden Beweise für die Schlussfolgerung, dass die Standardabweichungen der Körpergröße von weiblichen und männlichen Studenten unterschiedlich sind.

Hypothesentests in Python

Wir haben gerade einen F -Test für zwei Standardabweichungen in Python manuell durchgeführt. OK, wir haben eine Menge gelernt, aber jetzt nutzen wir die Mittel von Python, um das gleiche Ergebnis

wie oben mit nur einer Zeile Code zu erhalten!

Um einen F -Test für zwei Standardabweichungen in Python durchzuführen, verwenden wir unsere Funktion `simple_x2_test()` und verändern sie geringfügig. Wir geben zwei Vektoren als Dateneingabe an: `female_sample` und `male_sample`. Das Argument `alternative` muss nicht angegeben werden, da `alternative = 'two-sided'` die Vorgabe ist.

```
def simple_f_test(x, y, dfn, dfd, alpha, method="two-sided"):
    df1 = len(female_sample) - 1
    df2 = len(male_sample) - 1
    std_male = np.std(male_sample, ddof=1)
    std_female = np.std(female_sample, ddof=1)
    # Berechne Teststatistik
    Ftest = std_female**2 / std_male**2

    # linksseitiger Test
    if method == "left":
        p = scipy.stats.f.cdf(x=Ftest, dfn=df1, dfd=df2)
    # rechtsseitiger Test
    elif method == "right":
        p = 1 - scipy.stats.f.cdf(x=Ftest, dfn=df1, dfd=df2)

    # beidseitiger Test (default)
    else:
        p_lower = 1 - f.cdf(Ftest, dfn=df1, dfd=df2)
        p_upper = f.cdf(Ftest, dfn=df1, dfd=df2)
        p = p_lower

    # evaluiere p < alpha
    if (p_lower < alpha) & (p_upper > (1 - alpha)):
        reject = True
    else:
        reject = False

    # Ausgabe
    print("Significance level:", alpha)
    print("Degrees of freedom:", df1, df2)
    print("Test statistic:", round(Ftest, 4))
    print("p-value:", p)
    print("Reject H0:", reject)
```

```
simple_f_test(female_sample, male_sample, df1, df2, alpha=alpha)
```

```
Significance level: 0.05
Degrees of freedom: 24 24
Test statistic: 0.73
p-value: 0.7767376526940193
Reject H0: False
```

Es hat gut funktioniert! Vergleichen Sie die Ausgabe der Funktion `simple_f_test()` mit unserem Ergebnis von oben. Auch hier können wir zu dem Schluss kommen, dass die Daten bei einem Signifikanzniveau von 5% keine ausreichenden Beweise dafür liefern, dass die Standardabweichungen der Körpergröße von weiblichen und männlichen Studenten unterschiedlich sind.

Chi-Quadrat-Tests

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
from scipy.stats import chi2, chisquare
```

Neben Inferenzverfahren für Hypothesentests für Grundgesamtheitsparameter wie den Mittelwert μ und die Standardabweichung σ gibt es statistische Verfahren, um Rückschlüsse auf die Verteilung einer Variablen zu ziehen. Diese Schlussfolgerungsmethoden beruhen auf der Chi-Quadrat-Verteilung (χ^2) und werden daher als χ^2 -Tests bezeichnet.

Im folgenden Abschnitt werden der **Chi-Quadrat-Anpassungstest**, ein Hypothesentest, der angewandt wird, um Rückschlüsse auf die Verteilung einer Variablen zu ziehen, und der **Chi-Quadrat-Unabhängigkeitstest**, ein Hypothesentest, der angewandt wird, um zu entscheiden, ob ein Zusammenhang zwischen zwei Variablen einer Grundgesamtheit besteht, erörtert.

χ^2 -Verteilung

Grundlegende Eigenschaften von χ^2 -Kurven (s.279)

- Die Gesamtfläche unter einer χ^2 -Kurve ist gleich 1.

- Eine χ^2 -Kurve beginnt bei 0 auf der horizontalen Achse und erstreckt sich unendlich weit nach rechts, wobei sie sich der horizontalen Achse nähert, diese aber nie berührt.
- Eine χ^2 -Kurve ist rechtsschief.
- Mit zunehmender Anzahl von Freiheitsgraden sehen χ^2 -Kurven zunehmend wie normale Kurven aus.

▶ Show code cell content

Chi-Quadrat-Test auf Anpassungsgüte

Der χ^2 -**Anpassungstest** wird angewandt, um Hypothesentests über die Verteilung einer qualitativen (kategorialen) Variable oder einer diskreten quantitativen Variable, die nur endlich viele mögliche Werte hat, durchzuführen.

Die grundlegende Logik des χ^2 -Anpassungstest besteht darin, die Häufigkeiten von zwei Variablen zu vergleichen. Wir vergleichen die **beobachteten Häufigkeiten** einer Stichprobe mit den **erwarteten Häufigkeiten**.

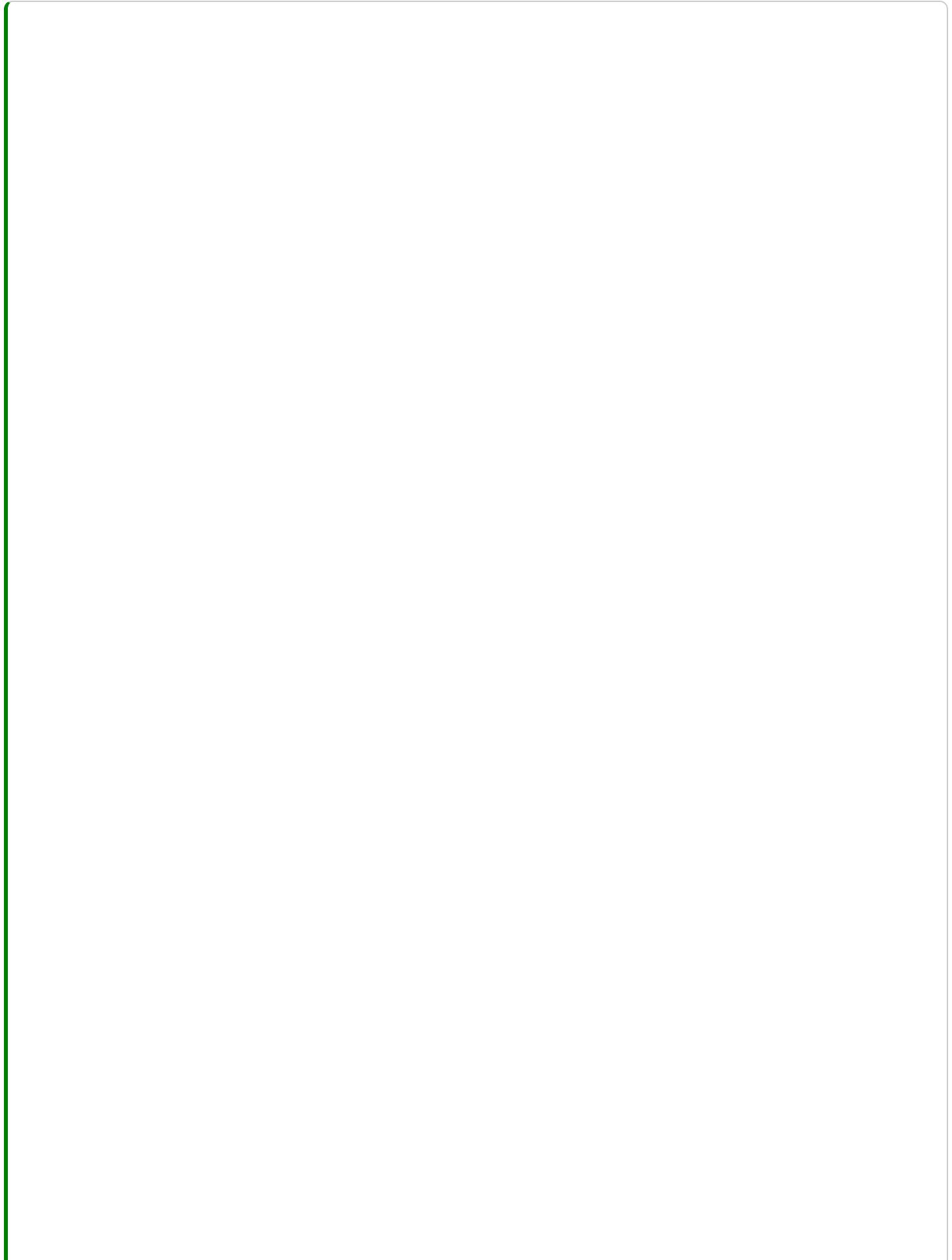
Betrachten wir ein einfaches Beispiel:

Am 22. September 2013 fand in Deutschland die [Bundestagswahl 2013](#) statt. Mehr als 44 Millionen Menschen gingen zur Wahl. 41,5% der deutschen Wähler entschieden sich für die Christlich Demokratische Union (CDU) und 25,7% für die Sozialdemokratische Partei (SPD). Der Einfachheit halber fassen wir den verbleibenden Prozentsatz der Stimmen (32,8%) als Sonstige zusammen.

Auf der Grundlage dieser Daten können wir eine Häufigkeitstabelle erstellen.

Partei	Prozent	Relative Häufigkeit
CDU	41,5	0,415
SPD	25,7	0,257
Sonstige	32,8	0,328
	100	1

Die dritte Spalte der obigen Tabelle entspricht den **relativen Häufigkeiten** in der deutschen Bevölkerung/Wählerschaft. Für diese Übung nehmen wir eine Zufallsstichprobe. Wir befragen 123 Studierende der FU Berlin nach ihrer Parteizugehörigkeit und halten die folgenden Antworten fest.



```
sample = [
    "SPD",
    "CDU",
    "Sonstige",
    "CDU",
    "CDU",
    "CDU",
    "CDU",
    "CDU",
    "CDU",
    "SPD",
    "CDU",
    "Sonstige",
    "SPD",
    "CDU",
    "SPD",
    "Sonstige",
    "CDU",
    "Sonstige",
    "CDU",
    "CDU",
    "Sonstige",
    "Sonstige",
    "SPD",
    "CDU",
    "CDU",
    "CDU",
    "Sonstige",
    "CDU",
    "CDU",
    "CDU",
    "SPD",
    "Sonstige",
    "CDU",
    "CDU",
    "CDU",
    "CDU",
    "SPD",
    "CDU",
    "Sonstige",
    "CDU",
    "CDU",
    "Sonstige",
    "SPD",
    "Sonstige",
    "CDU",
    "SPD",
    "SPD",
    "Sonstige",
    "SPD",
```



```

"SPD",
"SPD",
"CDU",
"Sonstige",
"CDU",
"CDU",
"SPD",
"CDU",
"Sonstige",
"CDU",
"CDU",
"CDU",
"CDU",
"CDU",
"Sonstige",
"Sonstige",
"SPD",
]

```

Im nächsten Schritt zählen wir das Auftreten der einzelnen Kategorien (Parteien) in unserer Stichprobe. Diese Größen sind die **beobachteten Häufigkeiten**.

```
[f"{l}: {sample.count(l)}" for l in set(sample)]
```

```
['CDU: 57', 'SPD: 26', 'Sonstige: 40']
```

Im nächsten Schritt berechnen wir die **erwartete Häufigkeit (E)** für jede Kategorie.

$$E = n \times p,$$

wobei n der Stichprobenumfang und p die entsprechende relative Häufigkeit aus der obigen Tabelle ist.

$$E_{CDU} = n \times p = 123 \times 0,415 = 51,045$$

$$E_{SPD} = n \times p = 123 \times 0,257 = 31,611$$

$$E_{Sonstige} = n \times p = 123 \times 0,382 = 46,986$$

Beachten Sie, dass es sich zwar um einzelne Zählungen handelt, die durch ganzzahlige Werte dargestellt werden, die **erwartete Häufigkeit**, E eine Fließkommazahl ist. Das ist in Ordnung.

Nun werden die **beobachteten Häufigkeiten** und die **erwarteten Häufigkeiten** in einer Tabelle zusammengeführt.

Partei	Beobachtete Häufigkeit	Erwartete Häufigkeit
CDU	57	51,045
SPD	26	31,611
Sonstige	40	46,986
	123	129,642

Großartig! Sobald wir die erwarteten Häufigkeiten kennen, müssen wir auf zwei Annahmen prüfen. Erstens müssen wir sicherstellen, dass alle erwarteten Häufigkeiten 1 oder größer sind, und zweitens, dass höchstens 20% der erwarteten Häufigkeiten kleiner als 5 sind. Durch einen Blick auf die Tabelle können wir bestätigen, dass beide Annahmen erfüllt sind.

Jetzt haben wir alle Zutaten, die wir brauchen, außer der Teststatistik, um einen χ^2 -Anpassungstest durchzuführen.

Die χ^2 Teststatistik für die Anpassungsgüte ist gegeben durch

$$\chi^2 = \sum \frac{(O - E)^2}{E},$$

wobei O den beobachteten Häufigkeiten und E den erwarteten Häufigkeiten entspricht. Die Teststatistik χ^2 approximiert eine *Chi-Quadrat*-Verteilung, wenn die Nullhypothese wahr ist. Die Zahl der Freiheitsgrade ist um 1 kleiner als die Zahl der möglichen Werte (Kategorien) für die betrachtete Variable.

$$df = c - 1$$

Ausgehend von den in der obigen Tabelle angegebenen beobachteten und erwarteten Häufigkeiten lässt sich der χ^2 -Wert relativ einfach berechnen. Um das Berechnungsverfahren jedoch

übersichtlicher zu gestalten, haben wir alle erforderlichen Berechnungsschritte in einer Tabelle zusammengefasst.

Partei	Beobachtete Häufigkeit	Erwartete Häufigkeit	Differenz $O - E$	Quadrat der Differenz $(O - E)^2$	χ^2 Zwischensumme $(O - E)^2/E$
CDU	57	51,045	5,955	35,462025	0,6947208
SPD	26	31,611	-5,611	31,483321	0,9959609
Sonstige	40	46,986	-6,986	48,804196	1,0386965
	123	129,642	-6,642		

In unserem Beispiel ergibt die χ^2 -Teststatistik für die Anpassungsgüte den Wert

$$\chi^2 = \sum \frac{(O - E)^2}{E} \approx 2,729$$

Wenn die Nullhypothese wahr ist, sind die beobachtete und die erwartete Häufigkeit ungefähr gleich. Dies führt zu einem kleinen Wert der χ^2 -Teststatistik und unterstützt somit H_0 . Ist der Wert der χ^2 -Teststatistik jedoch groß, liefern die Daten Beweise gegen H_0 .

χ^2 -Anpassungstest: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den χ^2 Anpassungstest in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

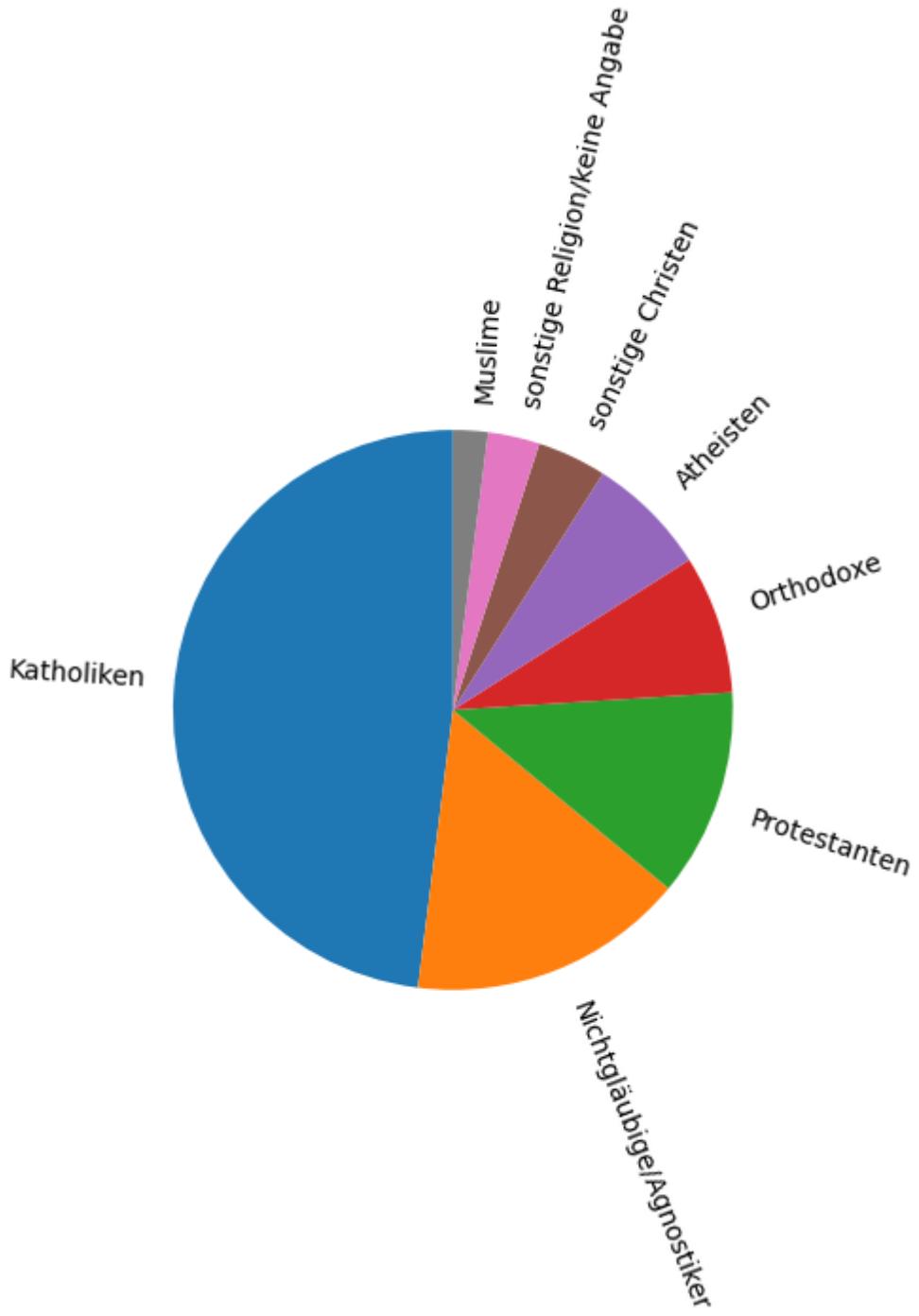
Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`,

gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary.

Es sei daran erinnert, dass χ^2 -Anpassungstests für qualitative (kategoriale) Variablen oder diskrete quantitative Variablen angewendet werden. Der `students` Datensatz enthält mehrere kategoriale Variablen, wie z. B. `gender`, `religion`, `major`, `minor` und `graduated`.

Um den χ^2 -Anpassungstest zu veranschaulichen, untersuchen wir, ob die Religion unter den Studierenden gleichmäßig verteilt ist, verglichen mit der Verteilung der Religion in der Bevölkerung der Europäischen Union. Die Daten auf kontinentaler Ebene stammen aus dem Bericht "Discrimination in the EU in 2012" ([European Union: European Commission, Special Eurobarometer, 393, p. 233](#)). Der Bericht enthält Daten für 8 Kategorien: 48% der Menschen werden als Katholiken, 16% als Nichtgläubige/Agnostiker, 12% als Protestanten, 8% als Orthodoxe, 7% als Atheisten, 4% als sonstige Christen, 3% als sonstige Religion/keine Angabe und 2% als Muslime eingestuft. Zum besseren Verständnis stellen wir die Daten in Form eines Tortendiagramms dar.

```
# Erzeuge Pie chart
y = np.array([48, 16, 12, 8, 7, 4, 3, 2])
mylabels = [
    "Katholiken",
    "Nichtgläubige/Agnostiker",
    "Protestanten",
    "Orthodoxe",
    "Atheisten",
    "sonstige Christen",
    "sonstige Religion/keine Angabe",
    "Muslime",
]
plt.pie(y, labels=mylabels, startangle=90, labeldistance=1.1, rotatelabels=True)
plt.show()
```



Vorbereitung der Daten

Wir beginnen mit der Datenexploration und der Datenaufbereitung.

Zunächst wollen wir wissen, welche Kategorien im Datensatz vorhanden sind. Dazu wenden wir die Funktion `unique()` an, die die einzelnen Spaltennamen (Kategorien) ausgibt.

```
students["religion"].unique()
```

```
array(['Muslim', 'Other', 'Protestant', 'Catholic', 'Orthodox'],  
      dtype=object)
```

Offensichtlich gibt es in den *Studentendaten* 5 weniger Kategorien, im Vergleich zu den 8 Kategorien, die im EU-Bericht angegeben sind. Um Vergleiche anstellen zu können, kodieren wir die Kategorien der Religionsvariable um, um schließlich 5 Kategorien zu erhalten: "Katholisch", "Muslimisch", "Orthodox", "Protestantisch" und "Sonstige". Achten Sie bei diesem Schritt darauf, dass Sie die Kategorien nicht verwechseln!

```
# Lege Kategorien zusammen  
raw_data = [  
    ("Other", (16 + 7 + 4 + 3)),  
    ("Catholic", 48),  
    ("Protestant", 12),  
    ("Orthodox", 8),  
    ("Muslim", 2),  
]  
  
data = pd.DataFrame(raw_data, columns=["Religion", "absoulte Häufigkeit"])  
data["relative Häufigkeit"] = data["absoulte Häufigkeit"] / 100  
data = data.set_index("Religion")  
data
```

	absoulte Häufigkeit	relative Häufigkeit
Religion		
Other	30	0.30
Catholic	48	0.48
Protestant	12	0.12
Orthodox	8	0.08
Muslim	2	0.02

Nun nehmen wir eine Zufallsstichprobe. Wir wählen 256 Studenten nach dem Zufallsprinzip aus und zählen mit der Funktion `value_counts()` die Anzahl der Studenten in jeder bestimmten Kategorie der Variablen `religion`. Wir erinnern uns, dass diese Menge den **beobachteten Häufigkeiten** entspricht.

```

n = 256
students_sample = students["religion"].sample(n, random_state=1)
sample_frequencies = students_sample.value_counts()
sample_frequencies

```

```

religion
Other      85
Catholic   77
Protestant 60
Orthodox   27
Muslim     7
Name: count, dtype: int64

```

Mit einer einzigen Code-Zeile fügen wir die beobachteten Häufigkeiten in `data` ein, den `dataframe`, den wir oben konstruiert haben.

```

data["beobachtete Häufigkeit"] = sample_frequencies
data

```

	absoulte Häufigkeit	relative Häufigkeit	beobachtete Häufigkeit
Religion			
Other	30	0.30	85
Catholic	48	0.48	77
Protestant	12	0.12	60
Orthodox	8	0.08	27
Muslim	2	0.02	7

Im nächsten Schritt berechnen wir die erwarteten Häufigkeiten. Erinnern Sie sich an die Gleichung:

$$E = n \times p$$

Wir fügen die erwarteten Häufigkeiten als neue Spalte in `data` ein.

```

data["erwartete Häufigkeit"] = n * data["relative Häufigkeit"]
data

```

	absoulte Häufigkeit	relative Häufigkeit	beobachtete Häufigkeit	erwartete Häufigkeit
Religion				
Other	30	0.30	85	76.80
Catholic	48	0.48	77	122.88
Protestant	12	0.12	60	30.72
Orthodox	8	0.08	27	20.48
Muslim	2	0.02	7	5.12

Sobald wir die erwarteten Häufigkeiten kennen, müssen wir zwei Annahmen überprüfen. Erstens müssen wir sicherstellen, dass alle erwarteten Häufigkeiten 1 oder größer sind, und zweitens, dass höchstens 20% der erwarteten Häufigkeiten kleiner als 5 sind. Durch einen Blick auf die Tabelle können wir bestätigen, dass beide Annahmen erfüllt sind.

Perfekt, jetzt sind wir fertig! Der Datensatz ist bereit für die Analyse mit dem χ^2 Anpassungstest zu analysieren. Erinnern Sie sich an die Frage, an der wir interessiert sind: **Ist die Religion unter den Studierenden gleichmäßig verteilt, verglichen mit der Verteilung der Religion in der Bevölkerung der Europäischen Union?**

Überprüfung der Hypothesen

Zur Durchführung des χ^2 Anpassungstests folgen wir dem schrittweisen Durchführungsverfahren für Hypothesentests. Der χ^2 Anpassungstest folgt demselben schrittweisen Verfahren wie Hypothesentests für den Grundgesamtheitsmittelwert.

-
- Schritt 1 Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an.
 - Schritt 2 Legen Sie das Signifikanzniveau, α fest.
 - Schritt 3 Berechnen Sie den Wert der Teststatistik.
 - Schritt 4 Bestimmen Sie den p-Wert.
 - Schritt 5 Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht ablehnen.
 - Schritt 6 Interpretieren Sie das Ergebnis des Hypothesentests.
-

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass die Religion unter den Studenten gleich verteilt ist, verglichen mit der Verteilung der Religion in der Bevölkerung der Europäischen Union.

H_0 : Die Variable hat die angegebene Verteilung

Alternative Hypothese

H_A : Die Variable hat nicht die angegebene Verteilung

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,01$$

```
alpha = 0.01
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die Gleichung für die Teststatistik von oben:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

```
# Berechne Teststatistik
x2 = sum(
    (data["beobachtete Häufigkeit"] - data["erwartete Häufigkeit"]) ** 2
    /
    data["erwartete Häufigkeit"]
)
x2
```

```
48.679361979166664
```

Der numerische Wert der Teststatistik beträgt $\approx 48,67936$.

Um den p -Wert zu berechnen, verwenden wir die Funktion `chi2.cdf()`. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet:

$$df = (c - 1)$$

```
# Berechne df
df = len(data) - 1

# Berechne p-Wert
p = chi2.sf(x2, df=df)
p
```

```
np.float64(6.811017529549072e-10)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,01; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 1%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 6,8110175 \cdot 10^{-10}$. Bei einem Signifikanzniveau von 1% lassen die Daten den Schluss zu, dass die Religionsverteilung unter den Studenten von der Religionsverteilung der Bevölkerung der Europäischen Union abweicht.

Hypothesentests in Python

Wir haben gerade einen χ^2 Anpassungstest in Python manuell durchgeführt. Sehr cool, aber jetzt wiederholen wir dieses Beispiel und nutzen die Python-Maschinerie, um das gleiche Ergebnis wie oben mit nur einer Zeile Code zu erhalten!

Für die Durchführung eines χ^2 Anpassungsgüte-Test in Python durchzuführen, verwenden wir die Funktion `chisquare()`. Wir geben zwei Vektoren als Daten ein: `data['beobachtete Häufigkeit']` und `data['erwartete Häufigkeit']`.

```
# Führe Chi-Square Goodness of Fit Test durch
chisquare(f_obs=data["beobachtete Häufigkeit"], f_exp=data["erwartete Häufigkeit"])
```

```
Power_divergenceResult(statistic=np.float64(48.679361979166664), pvalue=np.float64(6.25e-11))
```

Es hat gut funktioniert! Vergleichen Sie die Ausgabe der Funktion `chisquare()` mit unserem Ergebnis von oben. Auch hier können wir feststellen, dass die Daten bei einem Signifikanzniveau von 1% sehr starke Hinweise darauf liefern, dass sich die Religionsverteilung unter den Schülern von der Religionsverteilung der Bevölkerung der Europäischen Union unterscheidet.

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
The autoreload extension is already loaded. To reload it, use:
```

```
  %reload_ext autoreload
```

```
The lab_black extension is already loaded. To reload it, use:
```

```
  %reload_ext lab_black
```

```
-----
```

```
Working on the host: imarevic-pc
```

```
-----
```

```
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

```
-----
```

```
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

```
<Figure size 640x480 with 0 Axes>
```

Der Chi-Quadrat-Unabhängigkeitstest

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
from scipy.stats import chi2, chi2_contingency
```

Der χ^2 **Unabhängigkeitstest** ist eine inferentielle Methode, um zu entscheiden, ob ein Zusammenhang zwischen zwei Variablen besteht. Ähnlich wie bei anderen Hypothesentests besagt die Nullhypothese, dass die beiden Variablen nicht miteinander verbunden sind. Im Gegensatz dazu besagt die Alternativhypothese, dass die beiden Variablen miteinander verbunden sind.

Es sei daran erinnert, dass statistisch **abhängige Variablen** als **assoziierte Variablen** bezeichnet werden. Im Gegensatz dazu werden nicht-assoziierte Variablen als statistisch unabhängige Variablen bezeichnet. Erinnern Sie sich auch an das Konzept der [Kontingenztabellen](#) (auch als Zwei-Wege-Tabellen oder Kreuztabellen bezeichnet), in denen die Häufigkeitsverteilungen von bivariaten Daten dargestellt werden.

χ^2 -Unabhängigkeitstest

Die Grundidee des χ^2 -**Unabhängigkeitstests** besteht darin, die **beobachteten Häufigkeiten** in einer Kontingenztabelle mit den **erwarteten Häufigkeiten** zu vergleichen, unter der Annahme das die Nullhypothese der Nicht-Assoziation wahr ist. Die erwartete Häufigkeit für jede Zelle einer Kontingenztabelle ist gegeben durch

$$E = \frac{R \times C}{n},$$

wobei R die Zeilensumme ist, C die Spaltensumme und n der Stichprobenumfang ist.

Lassen Sie uns zum besseren Verständnis ein Beispiel konstruieren. Wir betrachten eine Exit Poll in Form einer Kontingenztabelle, die das Alter von $n = 1189$ Personen in den Kategorien $18 - 29$, $30 - 44$, $45 - 64$ und > 65 Jahre und ihre politische Zugehörigkeit, d. h. “Konservativ”, “Sozialistisch” und “Andere” aufgliedert. Diese Tabelle entspricht den beobachteten Häufigkeiten.

Beobachtete Häufigkeiten:

	Konservativ	Sozialistisch	Andere	Insgesamt
18-29	141	68	4	213
30-44	179	159	7	345
45-64	220	216	4	440
65 & älter	86	101	4	191
Insgesamt	626	544	19	1189

Auf der Grundlage der oben genannten Gleichung berechnen wir die erwartete Häufigkeit für jede Zelle.

Erwartete Häufigkeiten:

	Konservativ	Sozialistisch	Andere	Insgesamt
18-29	$\frac{213 \times 626}{1189} \approx 112,14$	$\frac{213 \times 544}{1189} \approx 97,45$	$\frac{213 \times 19}{1189} \approx 3,4$	213
30-44	$\frac{345 \times 626}{1189} \approx 181,64$	$\frac{345 \times 544}{1189} \approx 157,85$	$\frac{345 \times 19}{1189} \approx 5,51$	345
45-64	$\frac{440 \times 626}{1189} \approx 231,66$	$\frac{440 \times 544}{1189} \approx 201,31$	$\frac{440 \times 19}{1189} \approx 7,03$	440
65 & älter	$\frac{191 \times 626}{1189} \approx 100,56$	$\frac{191 \times 544}{1189} \approx 87,39$	$\frac{191 \times 19}{1189} \approx 3,05$	191
Insgesamt	626	544	19	1189

Sobald wir die erwarteten Häufigkeiten kennen, müssen wir zwei Annahmen überprüfen. Erstens müssen wir sicherstellen, dass alle erwarteten Häufigkeiten 1 oder größer sind, und zweitens, dass höchstens 20% der erwarteten Häufigkeiten kleiner als 5 sind. Ein Blick auf die Tabelle bestätigt, dass beide Annahmen erfüllt sind.

Der eigentliche Vergleich erfolgt auf der Grundlage der χ^2 Teststatistik für die beobachtete Häufigkeit und die erwartete Häufigkeit. Die -Teststatistik folgt der χ^2 -Verteilung und ist gegeben durch

$$\chi^2 = \sum \frac{(O - E)^2}{E},$$

wobei O für die beobachtete Häufigkeit und E für die erwartete Häufigkeit steht. Bitte beachten Sie, dass $\frac{(O-E)^2}{E}$ für jede Zelle ausgewertet und dann aufsummiert wird.

Die Anzahl der Freiheitsgrade ist gegeben durch

$$df = (r - 1) \times (c - 1),$$

wobei r und c die Anzahl der möglichen Werte für die beiden betrachteten Variablen sind.

Übertragen auf das obige Beispiel führt dies zu einem etwas langen Ausdruck, der der Kürze halber nur für die erste und die letzte Zeile der interessierenden Kontingenztafeln angegeben wird.

$$\chi^2 = \frac{(141 - 112, 14)^2}{112, 14} + \frac{(68 - 97, 45)^2}{97, 45} + \frac{(4 - 3, 4)^2}{3, 4} + \dots + \frac{(86 - 100, 56)^2}{100, 56} + \dots$$

Wenn die Nullhypothese wahr ist, sind die beobachtete und die erwartete Häufigkeit ungefähr gleich, was zu einem kleinen Wert der χ^2 -Teststatistik führt und somit H_0 unterstützt. Ist der Wert der χ^2 -Teststatistik jedoch groß, liefern die Daten Beweise gegen H_0 . In den nächsten Abschnitten wird weiter erörtert, wie der Wert der χ^2 -Teststatistik im Rahmen der Hypothesentests zu bewerten ist.

χ^2 Unabhängigkeitstest: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den χ^2 Unabhängigkeitstest in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

In dieser Übung wollen wir untersuchen, ob es einen Zusammenhang zwischen den Variablen `gender` und `major` gibt, oder mit anderen Worten, wir wollen wissen, ob männliche Studenten andere Studienfächer bevorzugen als weibliche Studenten.

Vorbereitung der Daten

Wir beginnen mit der Datenaufbereitung. Da wir nicht den gesamten Datensatz von 8239 Einträgen bearbeiten wollen, wählen wir zufällig 865 Studenten aus dem Datensatz aus. Der erste Schritt der Datenvorbereitung besteht darin, die interessierenden Daten in Form einer Kontingenztabelle darzustellen. Pandas bietet die Funktion `crosstab()`, die diese Aufgabe übernehmen wird.

```
n = 865
data = students.sample(n, random_state=1)
pd.crosstab(index=data["major"], columns=data["gender"])
```

gender	Female	Male
major		
Biology	89	65
Economics and Finance	55	79
Environmental Sciences	99	96
Mathematics and Statistics	39	92
Political Science	112	40
Social Sciences	70	29

Außerdem bestimmen wir die Spaltensummen und die Zeilensummen. Python bietet das Argument `margins = True`, das die Zeilensummen und die Spaltensummen für uns hinzufügt.

```
observed_frequencies = pd.crosstab(
    index=data["major"], columns=data["gender"], margins=True
)
observed_frequencies
```

gender	Female	Male	All
major			
Biology	89	65	154
Economics and Finance	55	79	134
Environmental Sciences	99	96	195
Mathematics and Statistics	39	92	131
Political Science	112	40	152
Social Sciences	70	29	99
All	464	401	865

Im nächsten Schritt konstruieren wir die **erwarteten Häufigkeiten**. Erinnern Sie sich an die Gleichung von oben:

$$E = \frac{R \times C}{n},$$

wobei R die Zeilensumme ist, C die Spaltensumme und n der Stichprobenumfang ist.

Wir berechnen die erwarteten Häufigkeiten mit der Numpy-Funktion `outer`. Das Ergebnis wandeln wir dann in eine Pandas `DataFrame` um.

```
rows = observed_frequencies["All"].values
cols = observed_frequencies.loc["All"].values
n = observed_frequencies.loc["All", "All"]
```

```
expected_frequencies = pd.DataFrame(
    np.outer(rows, cols) / n,
    index=observed_frequencies.index,
    columns=observed_frequencies.columns,
)
```

```
expected_frequencies
```

gender	Female	Male	All
major			
Biology	82.608092	71.391908	154.0
Economics and Finance	71.879769	62.120231	134.0
Environmental Sciences	104.601156	90.398844	195.0
Mathematics and Statistics	70.270520	60.729480	131.0
Political Science	81.535260	70.464740	152.0
Social Sciences	53.105202	45.894798	99.0
All	464.000000	401.000000	865.0

Sobald wir die erwarteten Häufigkeiten kennen, müssen wir zwei Annahmen überprüfen. Erstens müssen wir sicherstellen, dass alle erwarteten Häufigkeiten 1 oder größer sind, und zweitens, dass höchstens 20% der erwarteten Häufigkeiten kleiner als 5 sind. Durch einen Blick auf die Tabelle können wir bestätigen, dass beide Annahmen erfüllt sind.

Jetzt haben wir alle Daten, die wir für einen χ^2 Unabhängigkeitstest durchzuführen.

Überprüfung der Hypothesen

Zur Durchführung des **χ^2 -Unabhängigkeitstests** folgen wir dem Verfahren der schrittweisen Durchführung von Hypothesentests. Der **χ^2 -Unabhängigkeitstest** folgt demselben schrittweisen Verfahren wie in den vorangegangenen Abschnitten beschrieben.

-
- Schritt 1 Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an.
 - Schritt 2 Legen Sie das Signifikanzniveau, α fest.
 - Schritt 3 Berechnen Sie den Wert der Teststatistik.
 - Schritt 4 Bestimmen Sie den p-Wert.
 - Schritt 5 Wenn $p \leq \alpha$, H_0 ablehnen ; ansonsten H_0 nicht ablehnen.
 - Schritt 6 Interpretieren Sie das Ergebnis des Hypothesentests.
-

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass es keinen Zusammenhang zwischen dem Geschlecht und dem Hauptstudienfach der Studierenden gibt.

H_0 : Kein Zusammenhang zwischen Geschlecht und Studienschwerpunkt

Alternative Hypothese

H_A : Es gibt einen Zusammenhang zwischen dem Geschlecht und dem Hauptstudienfach

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,05$$

```
alpha = 0.05
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die Gleichung für die Teststatistik von oben:

$$\chi^2 = \sum \frac{(O - E)^2}{E},$$

wobei O für die beobachtete Häufigkeit und E für die erwartete Häufigkeit steht.

```
chi_2 = (
    (observed_frequencies - expected_frequencies) ** 2 / expected_frequencies)
    .sum(axis=1)
    .sum()
)
chi_2
```

```
np.float64(76.42971653913816)
```

Der numerische Wert der Teststatistik beträgt $\approx 76,43$.

Um den p -Wert zu berechnen, verwenden wir die Funktion `chi2.cdf()`. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet:

$$df = (r - 1) \times (c - 1),$$

wobei r und c die Anzahl der möglichen Werte für die beiden betrachteten Variablen sind.

```
df = (observed_frequencies.index.drop("All").nunique() - 1) * (observed_frequencies.columns.drop("All").nunique() - 1)  
df
```

5

```
p = chi2.sf(chi_2, df)  
p
```

`np.float64(4.67888864647207e-15)`

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

`np.True_`

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,05; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 5%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 4,6788886 \cdot 10^{-15}$. Bei einem Signifikanzniveau von 5% lassen die Daten den Schluss zu, dass es einen Zusammenhang zwischen dem Geschlecht und dem Hauptstudienfach gibt

Hypothesentests in Python

Wir haben gerade einen χ^2 Unabhängigkeitstest manuell durchgeführt. Wir können dasselbe in Python mit nur einer Zeile Code tun!

Dazu wenden wir die Funktion `chi2_contingency()` an. Wir achten darauf die Spalte und die Reihe `All` zu exkludieren.

```
stat, p, dof, expected = chi2_contingency(  
    observed_frequencies.drop("All").drop("All", axis=1)  
)  
print("p- Wert : ", p)  
print("Freiheitsgrade : ", dof)  
print("chi^2 : ", stat)
```

```
p- Wert : 4.678888646472395e-15  
Freiheitsgrade : 5  
chi^2 : 76.42971653913814
```

Perfekt! Vergleichen Sie die Ausgabe der Funktion `chi2_contingency()` mit unserem Ergebnis von oben. Auch hier können wir zu dem Schluss kommen, dass die Daten bei einem Signifikanzniveau von 5% sehr starke Anhaltspunkte dafür liefern, dass es einen Zusammenhang zwischen dem Geschlecht und dem Hauptstudienfach gibt.

Inferenzmethoden in Regression und Korrelation

Das lineare Modell ergibt sich aus der Gleichung

$$y = \beta_0 + \beta_1 x + e,$$

Wobei β_0 der Achsenabschnitt, β_1 der Regressionskoeffizient und e der Fehlerterm ist. Die beste Regressionsgerade wird durch Anwendung der [Methode der kleinsten Quadrate](#) gefunden, um die **Summe der quadratischen Fehler (SSE)** zu minimieren, d. h. die quadratische Differenz zwischen der gemessenen Reaktionsvariablen y und der Modellvorhersage \hat{y} zu minimieren, die wie folgt lautet

$$SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y - \hat{y})^2.$$

Im Abschnitt über die *lineare Regression* finden Sie weitere Einzelheiten zum linearen Modell.

Wie auch immer, wir müssen anerkennen, dass wir unsere Modelle, in diesem Fall unser lineares Regressionsmodell, auf Beobachtungsdaten aufbauen. Die Daten stammen also aus einer Grundgesamtheit und deren entsprechenden statistischen Eigenschaften, die uns im Allgemeinen unbekannt sind. Jede Beobachtung stellt also eine Ausprägung der Population dar, die mit dem Begriff Zufallsvariable bezeichnet wird.

Betrachten wir ein Beispiel, wie es in der folgenden Abbildung dargestellt ist. In diesem Beispiel sind die Parameter der Grundgesamtheit bekannt, so dass wir ein lineares Regressionsmodell der Form $y = \beta_0 + \beta_1 x = 1 + 0,25x$ erstellen können

► Show code cell content

Wenn wir jedoch eine Zufallsstichprobe aus der Grundgesamtheit nehmen und ein lineares Modell auf der Grundlage der Stichprobendaten erstellen, wird die Regressionslinie der Stichprobe nicht mit der Regressionslinie der Grundgesamtheit übereinstimmen. In der folgenden Abbildung haben wir vier Stichproben mit dem Stichprobenumfang 10 (blaue Punkte) gezogen. Wir sehen sofort, dass die Regressionslinie der Stichprobe (blaue gestrichelte Linie) nicht mit der Regressionslinie der Grundgesamtheit (graue Linie) übereinstimmt. Um dieser Variabilität Rechnung zu tragen, die auf den Zufallsstichprobenprozess zurückzuführen ist, wird eine Statistik berechnet, indem die folgende Gleichung angewendet wird

$$s_e = \sqrt{\frac{SSE}{n - 2}},$$

Wobei SSE für die **Summe der quadratischen Fehler** und n für den Stichprobenumfang steht. Die Statistik s_e wird als **Standardfehler der Schätzung** (s_e) oder als **Reststandardfehler** bezeichnet.

► Show code cell content

Wie oben dargestellt, variiert die Regressionsgerade der Stichprobe von einer Stichprobe zur anderen und ist daher eine Zufallsvariable. Ihre Verteilung wird als **Stichprobenverteilung der Steigung der Regressionsgeraden** bezeichnet.

Inferenz über die Steigung - Der t -Test des Regressionskoeffizienten

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
from scipy.stats import t, linregress
```

Der **t -Test des Regressionskoeffizienten** wird angewendet, um zu prüfen, ob die Steigung β_1 der Regressionslinie der Grundgesamtheit gleich 0 ist. Auf der Grundlage dieses Tests können wir entscheiden, ob x ein nützlicher (linearer) Vorhersagewert für y ist.

Die Teststatistik folgt einer t -Verteilung mit $df = n - 2$ und kann geschrieben werden als

$$t = \frac{\beta_1}{s_b} = \frac{\beta_1}{s_e / \sqrt{\sum(x - \bar{x})^2}},$$

wobei β_1 dem Stichprobenregressionskoeffizienten und s_e dem **Reststandardfehler** entspricht ($s_e = \sqrt{\frac{SSE}{n-2}}$ und $SSE = \sum_{i=1}^n e_i^2$)

Intervall-Schätzung von β_1

Das $100(1 - \alpha)\%$ -Konfidenzintervall für β_1 ist gegeben durch

$$\beta_1 \pm t_{\alpha/2} \times \frac{s_e}{\sqrt{\sum(x - \bar{x})^2}},$$

wobei s_e dem **Reststandardfehler** (auch als **Standardfehler der Schätzung** bezeichnet) entspricht.

Der Wert von t ergibt sich aus der t -Verteilung für das gegebene Konfidenzniveau und $n - 2$ Freiheitsgrade.

Der *t*-Test des Regressionskoeffizienten: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den **Regressions-*t*-Test** in einer Übung an.

Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen.

Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Um den *t*-Test des Regressionskoeffizienten zu veranschaulichen, untersuchen wir die Beziehung zwischen zwei Variablen, der Körpergröße der Studenten als Vorhersagevariable und dem Gewicht der Studenten als Antwortvariable. **Die Frage ist, ob die Vorhersagevariable `height` nützlich ist, um Vorhersagen über das Gewicht der Studierenden zu treffen.**

Vorbereitung der Daten

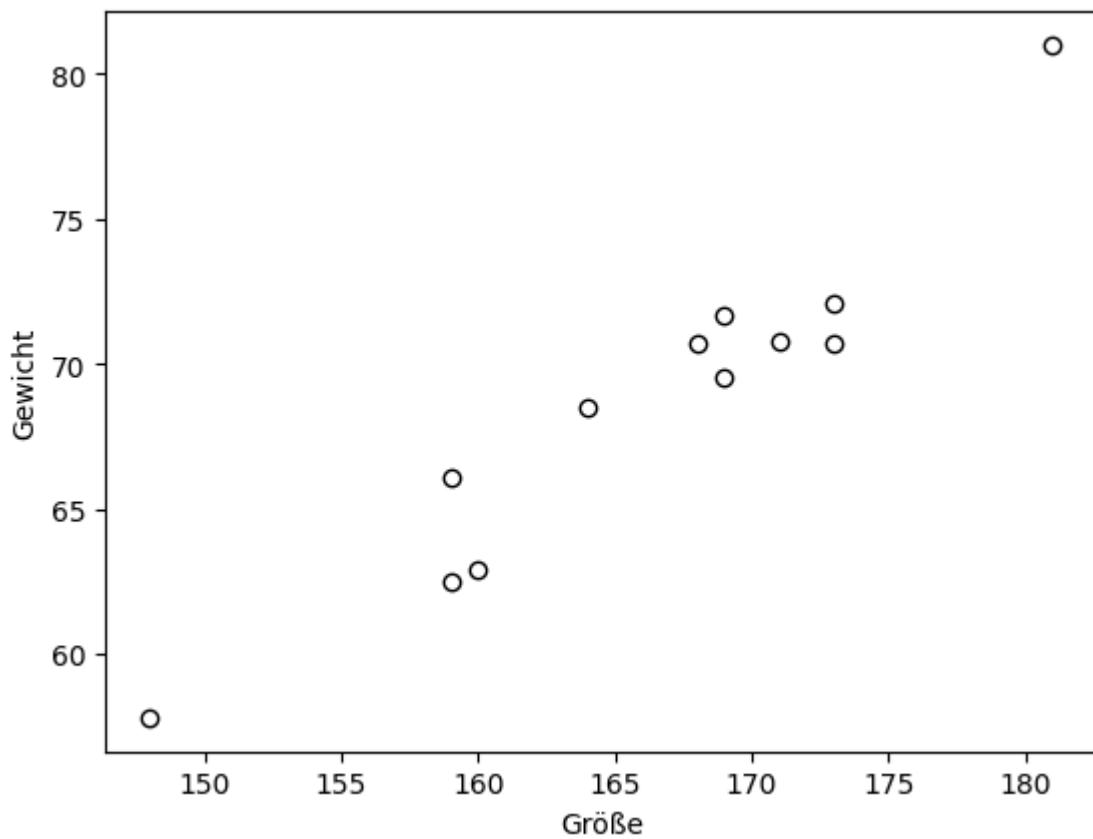
Zur Datenaufbereitung ziehen wir eine Zufallsstichprobe von 12 Studenten aus dem Datensatz und erstellen einen `dataframe` mit den zwei Variablen von Interesse (`height` und `weight`). Außerdem stellen wir die Daten in Form eines Streudiagramms dar, um die zugrunde liegende lineare Beziehung zwischen den beiden Variablen zu visualisieren.

```
# Nehme Stichprobe
n = 12
data = students[["height", "weight"]].sample(n, random_state=2)
data
```

	height	weight
stud_id		
769350	169	71.7
886785	181	81.0
648615	148	57.8
501070	173	72.1
649588	168	70.7
822135	159	66.1
261234	169	69.5
249763	159	62.5
211480	173	70.7
934141	164	68.5
399893	171	70.8
413775	160	62.9

```
# Erzeuge Streudiagramm
fig, ax = plt.subplots()
ax.scatter(data["height"], data["weight"], color="white", edgecolor="k")
ax.set_xlabel("Größe")
ax.set_ylabel("Gewicht")
```

```
Text(0, 0.5, 'Gewicht')
```



Die visuelle Inspektion bestätigt unsere Vermutung, dass die Beziehung zwischen der Größe und der Gewichtsvariable ungefähr linear ist. Mit anderen Worten: Mit zunehmender Größe neigt der einzelne Student dazu, ein höheres Gewicht zu haben.

Überprüfung der Hypothesen

Zur Durchführung des ***t*-Test des Regressionskoeffizienten** folgen wir dem schrittweisen Durchführungsverfahren für Hypothesentests. Der ***t*-Test des Regressionskoeffizienten** folgt demselben schrittweisen Verfahren wie in den vorangegangenen Abschnitten beschrieben.

Schritt 1	Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an
Schritt 2	Legen Sie das Signifikanzniveau, α fest
Schritt 3	Berechnen Sie den Wert der Teststatistik
Schritt 4	Bestimmen Sie den p-Wert
Schritt 5	Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen
Schritt 6	Interpretieren Sie das Ergebnis des Hypothesentests

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass es keinen linearen Zusammenhang zwischen der Größe und dem Gewicht der Personen im Datensatz der Studierenden gibt.

$H_0 : \beta_1 = 0$ (die Vorhersagevariable ist für die Erstellung von Vorhersagen nicht geeignet)

Alternative Hypothese

$H_A : \beta_1 \neq 0$ (die Vorhersagevariable ist für die Erstellung von Vorhersagen geeignet)

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,01$$

```
alpha = 0.01
```

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p-Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die obige Gleichung.

$$t = \frac{\beta_1}{s_b} = \frac{\beta_1}{s_e / \sqrt{\sum(x - \bar{x})^2}}$$

wobei $\beta_1 = \frac{cov(x,y)}{var(x)}$, und

$$s_e = \sqrt{\frac{SSE}{n - 2}},$$

wobei $SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y - \hat{y})^2$. Die Teststatistik folgt einer t -Verteilung mit $df = n - 2$. Um $\hat{y} = \beta_0 + \beta_1 x$ zu berechnen, müssen wir β_0 kennen, das als $\beta_0 = \bar{y} - \beta_1 \bar{x}$ berechnet wird.

Um nicht durch die verschiedenen Berechnungsschritte verwirrt zu werden, führen wir einen Schritt nach dem anderen durch.

- Erstellen Sie das lineare Modell durch Berechnung des Achsenabschnitts (β_0) und des Regressionskoeffizienten (β_1).

```
y_bar = data["weight"].mean()
x_bar = data["height"].mean()

# Lineares Modell
beta1 = np.cov(data["height"], data["weight"]) / data["height"].var()
beta1[0][1]
```

```
np.float64(0.6507615230460922)
```

```
beta0 = y_bar - beta1[0][1] * x_bar
beta0
```

```
np.float64(-39.44320641282566)
```

Berechnen Sie die Summe der quadrierten Fehler (SSE) und den Reststandardfehler (s_e).

```
# Berechne SSE
y_hat = beta0 + beta1[0][1] * data["height"]
SSE = sum((data["weight"] - y_hat) ** 2)
SSE
```

```
30.38618436873755
```

```
# Berechne Reststandardfehler
se = np.sqrt(SSE / (n - 2))
se
```

```
np.float64(1.743163341994592)
```

- Berechnen Sie den Wert der Teststatistik.

```
# Berechne Teststatistik
tw = beta1[0][1] / (se / np.sqrt(sum((data["height"] - x_bar) ** 2)))
tw
```

```
np.float64(10.766100516207496)
```

Der numerische Wert der Teststatistik ist $\approx 10,7661$.

Um den p -Wert zu berechnen, wenden wir die Funktion `t.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet.

$$df = n - 2 = 10$$

```
# Berechne p-Wert
df = data.shape[0] - 2

# zweiseitiger Test
p_upper = t.sf(abs(tw), df=df)
p_lower = t.cdf(-abs(tw), df=df)
p = p_upper + p_lower
p
```

```
np.float64(8.048424329265031e-07)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,01; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 1%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 8,0484243 \cdot 10^{-7}$. Bei einem Signifikanzniveau von 1% lassen die Daten den Schluss zu, dass die Variable Größe ein guter Schätzer für das Gewicht der Studenten ist.

Hypothesentests in Python

Wir haben gerade einen **t -Test des Regressionskoeffizienten** in Python manuell berechnet. Das ist in Ordnung, aber wir können dasselbe in Python mit nur ein paar Zeilen Code tun!

Daher müssen wir die Funktion `linregress()` auf unsere Antwortvariable `weight` und unsere Vorhersagevariable `height` anwenden.

```
gradient, intercept, r_value, p_value, stderr = linregress(  
    data["height"], data["weight"]  
)  
print("p-value", p_value)
```

```
p-value 8.048424329265105e-07
```

Für den Vergleich mit den manuell berechneten Werten extrahieren wir den Achsenabschnitt β_0 und die Steigung β_1

```
print(f"Achsenabschnitt: {intercept}")
print(f"Steigung: {gradient}")
```

```
Achsenabschnitt: -39.44320641282562
Steigung: 0.650761523046092
```

Vergleichen Sie die Ausgabe mit unseren Ergebnissen von oben. Sie passen perfekt!

Inferenzstatistik - Der lineare Korrelationskoeffizienten

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import sample
from scipy.stats import t, pearsonr
```

Um zu prüfen, ob zwei Variablen linear korreliert sind, d. h. ob eine lineare Beziehung zwischen den beiden Variablen besteht, können wir den so genannten ***t*-Test zur Prüfung des Korrelationskoeffizienten** anwenden. Der **lineare Korrelationskoeffizient der Grundgesamtheit**, ρ , misst die lineare Korrelation zweier Variablen auf die gleiche Weise wie der **lineare Korrelationskoeffizient der Stichprobe**, r , die lineare Korrelation zweier Variablen einer Stichprobe von Paaren misst. Sowohl ρ als auch r beschreiben die Stärke der linearen Beziehung zwischen zwei Variablen; r ist jedoch eine Schätzung von ρ der aus Stichprobendaten gewonnen wird.

Der lineare Korrelationskoeffizient von zwei Variablen liegt zwischen -1 und 1 . Bei $\rho = 0$ sind die Variablen linear unkorreliert, es besteht also keine lineare Beziehung zwischen den Variablen. Bei $\rho \neq 0$ sind die Variablen linear korreliert. Wenn $\rho > 0$, sind die Variablen positiv linear korreliert, und wenn $\rho < 0$ sind die Variablen negativ linear korreliert.

Eine häufig verwendete Statistik zur Berechnung der linearen Beziehung zwischen quantitativen Variablen ist der [Pearson-Produkt-Moment-Korrelationskoeffizient](#). Er ist gegeben durch

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{s_{xy}}{s_x s_y},$$

wobei s_{xy} die Kovarianz von x und y ist und s_x und s_y die Standardabweichungen von x bzw. y sind.

Da der lineare Korrelationskoeffizient der Stichprobe, r eine Schätzung des linearen Korrelationskoeffizienten der Grundgesamtheit, ρ , ist, können wir r für einen Hypothesentest für ρ verwenden. Die Teststatistik für einen Korrelationstest hat eine t -Verteilung mit $n - 2$ Freiheitsgraden und kann wie folgt geschrieben werden

$$t = \frac{r}{\sqrt{\frac{1-r^2}{n-2}}}.$$

Der t -test zur Prüfung des Korrelationskoeffizienten: Ein Beispiel

Um den t -Test für den Korrelationskoeffizienten zu üben, laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Um den t -Test für die Korrelation zu veranschaulichen, untersuchen wir die Beziehung zwischen den Variablen `score1` und `score2`, die das Ergebnis von zwei obligatorischen statistischen Prüfungen

darstellen. **Die Frage ist, ob es eine lineare Beziehung zwischen den Noten von zwei aufeinanderfolgenden statistischen Prüfungen gibt?**

Vorbereitung der Daten

Wir beginnen mit der Datenvorbereitung.

- Wir unterteilen den Datensatz anhand der Variablen `score1` und `score2`. Durch Anwendung der Funktion `dropna()` lassen wir alle `nan`-Werte im Datensatz aus.
- Dann ziehen wir aus jeder Teilmenge 50 Studierende und extrahieren die Variablen von Interesse.

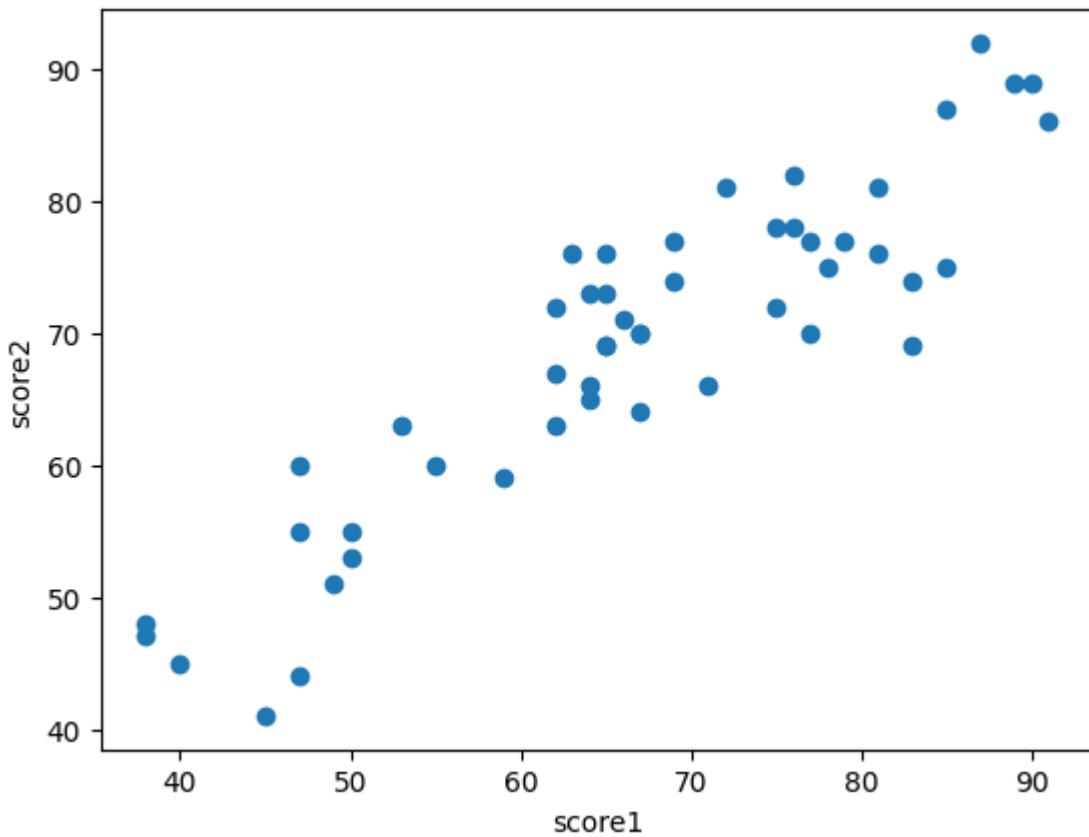
```
n = 50
data = students.dropna()[["score1", "score2"]].sample(n, random_state=1)

score1 = data["score1"]
score2 = data["score2"]
```

Zum Zweck der visuellen Kontrolle stellen wir die Stichprobe in Form eines Streudiagramms dar.

```
# Erzeuge Streudiagramm
fig, ax = plt.subplots()
ax.scatter(x=score1, y=score2)
ax.set_xlabel("score1")
ax.set_ylabel("score2")
```

```
Text(0, 0.5, 'score2')
```



Die visuelle Inspektion zeigt, dass eine positive lineare Beziehung zwischen den Variablen `score1` und `score2` besteht.

Überprüfung der Hypothesen

Zur Durchführung des t -Tests für die Korrelation folgen wir dem schrittweisen Durchführungsverfahren für Hypothesentests. Der t -Test für die Korrelation folgt demselben schrittweisen Verfahren wie in den vorangegangenen Abschnitten beschrieben.

Schritt 1	Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an
Schritt 2	Legen Sie das Signifikanzniveau, α fest
Schritt 3	Berechnen Sie den Wert der Teststatistik
Schritt 4	Bestimmen Sie den p-Wert
Schritt 5	Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen
Schritt 6	Interpretieren Sie das Ergebnis des Hypothesentests

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass es keinen linearen Zusammenhang zwischen den Noten von zwei aufeinanderfolgenden Statistikprüfungen gibt.

$$H_0 : r = 0$$

Erinnern Sie sich daran, dass die Formulierung der Alternativhypothese vorgibt, ob wir einen zweiseitigen, einen links- oder einen rechtsseitigen Hypothesentest durchführen.

Alternative Hypothese

$$H_A : r \neq 0$$

Diese Formulierung führt zu einem zweiseitigen Hypothesentest.

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,01$$

alpha = 0.01

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p-Wert

Zur Veranschaulichung berechnen wir die Teststatistik manuell in Python. Erinnern Sie sich an die obige Gleichungsform:

$$t = \frac{r}{\sqrt{\frac{1-r^2}{n-2}}}$$

```
n = len(score1)

# Berechne Teststatistik
# Berechne PearsonKorrelationskoeffizient r
r = np.corrcoef(score1, score2)[0][1]

# Teststatistik
tw = r / np.sqrt((1 - r**2) / (n - 2))
tw
```

```
np.float64(14.846357121994247)
```

Der numerische Wert der Teststatistik ist 14,846357.

Um den p -Wert zu berechnen, wenden wir die Funktion `t.cdf()` an. Erinnern Sie sich daran, wie man die Freiheitsgrade berechnet:

$$df = n - 2 = 48$$

```
# Berechne den p-Wert
df = len(score1) - 2

# zweiseitiger Test
p_upper = t.sf(abs(tw), df=df)
p_lower = t.cdf(-abs(tw), df=df)
p = p_upper + p_lower
p
```

```
np.float64(1.4401962006630608e-19)
```

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,01; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 1%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 1,4401962 \cdot 10^{-19}$. Bei einem Signifikanzniveau von 1% lassen die Daten den Schluss zu, dass die Prüfungsnoten der Studenten linear korreliert sind.

Hypothesentests in Python

Wir haben soeben einen t -test zur Prüfung des Korrelationskoeffizienten in Python manuell durchgeführt. Wir können dasselbe in Python mit nur einer Zeile Code tun!

Dazu wenden wir die Funktion `pearsonr()` an. Für die Funktion verwenden wir die zwei Vektoren, `score1` und `score2`, als Dateneingabe.

```
r, p_value = pearsonr(score1, score2)
print(f"Korrelationskoeffizient: {r}")
print(f"p - Wert: {p_value}")
```

```
Korrelationskoeffizient: 0.906185403227985
p - Wert: 1.4401962006630623e-19
```

Perfekt! Vergleichen Sie die Ausgabe der Funktion `pearsonr()` mit unserem Ergebnis von oben. Darüber hinaus gibt die Funktion den Pearson-Korrelationskoeffizienten für die Beispieldaten aus. Aus der Ausgabe der Funktion `pearsonr()` können wir schließen, dass die Daten bei einem Signifikanzniveau von 1% sehr starke Hinweise darauf liefern, dass die Prüfungsnoten der Studenten linear korreliert sind.

Wahrscheinlichkeits-Tabellen

Die folgenden Wahrscheinlichkeitsverteilungen werden behandelt:

- z -Verteilung
- t -Verteilung
- χ^2 -Verteilung
- F -Verteilung ($\alpha = 1$ und $\alpha = 5$)

Die z -Verteilung

Quantilen der Standard-Normal-(z)-Verteilung

- $\phi(-z) \rightarrow$ unter Mittelwert
- $\phi(z) \rightarrow$ über Mittelwert
- $D(z) \rightarrow$ symmetrisches Intervall

z	$\phi(-z)$	$\phi(z)$	$D(z)$
0,01	0,496	0,504	0,007979
0,02	0,492	0,508	0,01596
0,03	0,488	0,512	0,02393
0,04	0,484	0,516	0,03191
0,05	0,4801	0,5199	0,03988
0,06	0,4761	0,5239	0,04784
0,07	0,4721	0,5279	0,05581
0,08	0,4681	0,5319	0,06376
0,09	0,4641	0,5359	0,07171
0,1	0,4602	0,5398	0,07966
0,11	0,4562	0,5438	0,08759
0,12	0,4522	0,5478	0,09552
0,13	0,4483	0,5517	0,1034
0,14	0,4443	0,5557	0,1113
0,15	0,4404	0,5596	0,1192
0,16	0,4364	0,5636	0,1271
0,17	0,4325	0,5675	0,135
0,18	0,4286	0,5714	0,1428
0,19	0,4247	0,5753	0,1507
0,2	0,4207	0,5793	0,1585

0,21	0,4168	0,5832	0,1663
0,22	0,4129	0,5871	0,1741
0,23	0,409	0,591	0,1819
0,24	0,4052	0,5948	0,1897
0,25	0,4013	0,5987	0,1974
0,26	0,3974	0,6026	0,2051
0,27	0,3936	0,6064	0,2128
0,28	0,3897	0,6103	0,2205
0,29	0,3859	0,6141	0,2282
0,3	0,3821	0,6179	0,2358
0,31	0,3783	0,6217	0,2434
0,32	0,3745	0,6255	0,251
0,33	0,3707	0,6293	0,2586
0,34	0,3669	0,6331	0,2661
0,35	0,3632	0,6368	0,2737
0,36	0,3594	0,6406	0,2812
0,37	0,3557	0,6443	0,2886
0,38	0,352	0,648	0,2961
0,39	0,3483	0,6517	0,3035
0,4	0,3446	0,6554	0,3108
0,41	0,3409	0,6591	0,3182

0,42	0,3372	0,6628	0,3255
0,43	0,3336	0,6664	0,3328
0,44	0,33	0,67	0,3401
0,45	0,3264	0,6736	0,3473
0,46	0,3228	0,6772	0,3545
0,47	0,3192	0,6808	0,3616
0,48	0,3156	0,6844	0,3688
0,49	0,3121	0,6879	0,3759
0,5	0,3085	0,6915	0,3829
0,51	0,305	0,695	0,3899
0,52	0,3015	0,6985	0,3969
0,53	0,2981	0,7019	0,4039
0,54	0,2946	0,7054	0,4108
0,55	0,2912	0,7088	0,4177
0,56	0,2877	0,7123	0,4245
0,57	0,2843	0,7157	0,4313
0,58	0,281	0,719	0,4381
0,59	0,2776	0,7224	0,4448
0,6	0,2743	0,7257	0,4515
0,61	0,2709	0,7291	0,4581
0,62	0,2676	0,7324	0,4647

0,63	0,2643	0,7357	0,4713
0,64	0,2611	0,7389	0,4778
0,65	0,2578	0,7422	0,4843
0,66	0,2546	0,7454	0,4907
0,67	0,2514	0,7486	0,4971
0,68	0,2483	0,7517	0,5035
0,69	0,2451	0,7549	0,5098
0,7	0,242	0,758	0,5161
0,71	0,2389	0,7611	0,5223
0,72	0,2358	0,7642	0,5285
0,73	0,2327	0,7673	0,5346
0,74	0,2296	0,7704	0,5407
0,75	0,2266	0,7734	0,5467
0,76	0,2236	0,7764	0,5527
0,77	0,2206	0,7794	0,5587
0,78	0,2177	0,7823	0,5646
0,79	0,2148	0,7852	0,5705
0,8	0,2119	0,7881	0,5763
0,81	0,209	0,791	0,5821
0,82	0,2061	0,7939	0,5878
0,83	0,2033	0,7967	0,5935

0,84	0,2005	0,7995	0,5991
0,85	0,1977	0,8023	0,6047
0,86	0,1949	0,8051	0,6102
0,87	0,1922	0,8078	0,6157
0,88	0,1894	0,8106	0,6211
0,89	0,1867	0,8133	0,6265
0,9	0,1841	0,8159	0,6319
0,91	0,1814	0,8186	0,6372
0,92	0,1788	0,8212	0,6424
0,93	0,1762	0,8238	0,6476
0,94	0,1736	0,8264	0,6528
0,95	0,1711	0,8289	0,6579
0,96	0,1685	0,8315	0,6629
0,97	0,166	0,834	0,668
0,98	0,1635	0,8365	0,6729
0,99	0,1611	0,8389	0,6778
1	0,1587	0,8413	0,6827
1,01	0,1562	0,8438	0,6875
1,02	0,1539	0,8461	0,6923
1,03	0,1515	0,8485	0,697
1,04	0,1492	0,8508	0,7017

1,05	0,1469	0,8531	0,7063
1,06	0,1446	0,8554	0,7109
1,07	0,1423	0,8577	0,7154
1,08	0,1401	0,8599	0,7199
1,09	0,1379	0,8621	0,7243
1,1	0,1357	0,8643	0,7287
1,11	0,1335	0,8665	0,733
1,12	0,1314	0,8686	0,7373
1,13	0,1292	0,8708	0,7415
1,14	0,1271	0,8729	0,7457
1,15	0,1251	0,8749	0,7499
1,16	0,123	0,877	0,754
1,17	0,121	0,879	0,758
1,18	0,119	0,881	0,762
1,19	0,117	0,883	0,766
1,2	0,1151	0,8849	0,7699
1,21	0,1131	0,8869	0,7737
1,22	0,1112	0,8888	0,7775
1,23	0,1093	0,8907	0,7813
1,24	0,1075	0,8925	0,785
1,25	0,1056	0,8944	0,7887

1,26	0,1038	0,8962	0,7923
1,27	0,102	0,898	0,7959
1,28	0,1003	0,8997	0,7995
1,29	0,09853	0,9015	0,8029
1,3	0,0968	0,9032	0,8064
1,31	0,0951	0,9049	0,8098
1,32	0,09342	0,9066	0,8132
1,33	0,09176	0,9082	0,8165
1,34	0,09012	0,9099	0,8198
1,35	0,08851	0,9115	0,823
1,36	0,08691	0,9131	0,8262
1,37	0,08534	0,9147	0,8293
1,38	0,08379	0,9162	0,8324
1,39	0,08226	0,9177	0,8355
1,4	0,08076	0,9192	0,8385
1,41	0,07927	0,9207	0,8415
1,42	0,0778	0,9222	0,8444
1,43	0,07636	0,9236	0,8473
1,44	0,07493	0,9251	0,8501
1,45	0,07353	0,9265	0,8529
1,46	0,07215	0,9279	0,8557

1,47	0,07078	0,9292	0,8584
1,48	0,06944	0,9306	0,8611
1,49	0,06811	0,9319	0,8638
1,5	0,06681	0,9332	0,8664
1,51	0,06552	0,9345	0,869
1,52	0,06426	0,9357	0,8715
1,53	0,06301	0,937	0,874
1,54	0,06178	0,9382	0,8764
1,55	0,06057	0,9394	0,8789
1,56	0,05938	0,9406	0,8812
1,57	0,05821	0,9418	0,8836
1,58	0,05705	0,9429	0,8859
1,59	0,05592	0,9441	0,8882
1,6	0,0548	0,9452	0,8904
1,61	0,0537	0,9463	0,8926
1,62	0,05262	0,9474	0,8948
1,63	0,05155	0,9484	0,8969
1,64	0,0505	0,9495	0,899
1,65	0,04947	0,9505	0,9011
1,66	0,04846	0,9515	0,9031
1,67	0,04746	0,9525	0,9051

1,68	0,04648	0,9535	0,907
1,69	0,04551	0,9545	0,909
1,7	0,04457	0,9554	0,9109
1,71	0,04363	0,9564	0,9127
1,72	0,04272	0,9573	0,9146
1,73	0,04182	0,9582	0,9164
1,74	0,04093	0,9591	0,9181
1,75	0,04006	0,9599	0,9199
1,76	0,0392	0,9608	0,9216
1,77	0,03836	0,9616	0,9233
1,78	0,03754	0,9625	0,9249
1,79	0,03673	0,9633	0,9265
1,8	0,03593	0,9641	0,9281
1,81	0,03515	0,9649	0,9297
1,82	0,03438	0,9656	0,9312
1,83	0,03362	0,9664	0,9328
1,84	0,03288	0,9671	0,9342
1,85	0,03216	0,9678	0,9357
1,86	0,03144	0,9686	0,9371
1,87	0,03074	0,9693	0,9385
1,88	0,03005	0,9699	0,9399

1,89	0,02938	0,9706	0,9412
1,9	0,02872	0,9713	0,9426
1,91	0,02807	0,9719	0,9439
1,92	0,02743	0,9726	0,9451
1,93	0,0268	0,9732	0,9464
1,94	0,02619	0,9738	0,9476
1,95	0,02559	0,9744	0,9488
1,96	0,025	0,975	0,95
1,97	0,02442	0,9756	0,9512
1,98	0,02385	0,9761	0,9523
1,99	0,0233	0,9767	0,9534
2	0,02275	0,9772	0,9545
2,01	0,02222	0,9778	0,9556
2,02	0,02169	0,9783	0,9566
2,03	0,02118	0,9788	0,9576
2,04	0,02068	0,9793	0,9586
2,05	0,02018	0,9798	0,9596
2,06	0,0197	0,9803	0,9606
2,07	0,01923	0,9808	0,9615
2,08	0,01876	0,9812	0,9625
2,09	0,01831	0,9817	0,9634

2,1	0,01786	0,9821	0,9643
2,11	0,01743	0,9826	0,9651
2,12	0,017	0,983	0,966
2,13	0,01659	0,9834	0,9668
2,14	0,01618	0,9838	0,9676
2,15	0,01578	0,9842	0,9684
2,16	0,01539	0,9846	0,9692
2,17	0,015	0,985	0,97
2,18	0,01463	0,9854	0,9707
2,19	0,01426	0,9857	0,9715
2,2	0,0139	0,9861	0,9722
2,21	0,01355	0,9864	0,9729
2,22	0,01321	0,9868	0,9736
2,23	0,01287	0,9871	0,9743
2,24	0,01255	0,9875	0,9749
2,25	0,01222	0,9878	0,9756
2,26	0,01191	0,9881	0,9762
2,27	0,0116	0,9884	0,9768
2,28	0,0113	0,9887	0,9774
2,29	0,01101	0,989	0,978
2,3	0,01072	0,9893	0,9786

2,31	0,01044	0,9896	0,9791
2,32	0,01017	0,9898	0,9797
2,33	0,009903	0,9901	0,9802
2,34	0,009642	0,9904	0,9807
2,35	0,009387	0,9906	0,9812
2,36	0,009137	0,9909	0,9817
2,37	0,008894	0,9911	0,9822
2,38	0,008656	0,9913	0,9827
2,39	0,008424	0,9916	0,9832
2,4	0,008198	0,9918	0,9836
2,41	0,007976	0,992	0,984
2,42	0,00776	0,9922	0,9845
2,43	0,007549	0,9925	0,9849
2,44	0,007344	0,9927	0,9853
2,45	0,007143	0,9929	0,9857
2,46	0,006947	0,9931	0,9861
2,47	0,006756	0,9932	0,9865
2,48	0,006569	0,9934	0,9869
2,49	0,006387	0,9936	0,9872
2,5	0,00621	0,9938	0,9876
2,51	0,006037	0,994	0,9879

2,52	0,005868	0,9941	0,9883
2,53	0,005703	0,9943	0,9886
2,54	0,005543	0,9945	0,9889
2,55	0,005386	0,9946	0,9892
2,56	0,005234	0,9948	0,9895
2,57	0,005085	0,9949	0,9898
2,58	0,00494	0,9951	0,9901
2,59	0,004799	0,9952	0,9904
2,6	0,004661	0,9953	0,9907
2,61	0,004527	0,9955	0,9909
2,62	0,004396	0,9956	0,9912
2,63	0,004269	0,9957	0,9915
2,64	0,004145	0,9959	0,9917
2,65	0,004025	0,996	0,992
2,66	0,003907	0,9961	0,9922
2,67	0,003793	0,9962	0,9924
2,68	0,003681	0,9963	0,9926
2,69	0,003573	0,9964	0,9929
2,7	0,003467	0,9965	0,9931
2,71	0,003364	0,9966	0,9933
2,72	0,003264	0,9967	0,9935

2,73	0,003167	0,9968	0,9937
2,74	0,003072	0,9969	0,9939
2,75	0,00298	0,997	0,994
2,76	0,00289	0,9971	0,9942
2,77	0,002803	0,9972	0,9944
2,78	0,002718	0,9973	0,9946
2,79	0,002635	0,9974	0,9947
2,8	0,002555	0,9974	0,9949
2,81	0,002477	0,9975	0,995
2,82	0,002401	0,9976	0,9952
2,83	0,002327	0,9977	0,9953
2,84	0,002256	0,9977	0,9955
2,85	0,002186	0,9978	0,9956
2,86	0,002118	0,9979	0,9958
2,87	0,002052	0,9979	0,9959
2,88	0,001988	0,998	0,996
2,89	0,001926	0,9981	0,9961
2,9	0,001866	0,9981	0,9963
2,91	0,001807	0,9982	0,9964
2,92	0,00175	0,9982	0,9965
2,93	0,001695	0,9983	0,9966

2,94	0,001641	0,9984	0,9967
2,95	0,001589	0,9984	0,9968
2,96	0,001538	0,9985	0,9969
2,97	0,001489	0,9985	0,997
2,98	0,001441	0,9986	0,9971
2,99	0,001395	0,9986	0,9972
3	0,00135	0,9987	0,9973

Die t -Verteilung

Quantilen der Studentschen t-Verteilung

Si	0,75	0,9	0,95	0,975	0,99	0,995	0,999	0,9995
df/Alpha	0,25	0,1	0,05	0,025	0,01	0,005	0,001	5e-04
1	1	3,08	6,31	12,71	31,82	63,66	318,3	636,6
2	0,82	1,89	2,92	4,3	6,96	9,92	22,33	31,6
3	0,76	1,64	2,35	3,18	4,54	5,84	10,21	12,92
4	0,74	1,53	2,13	2,78	3,75	4,6	7,17	8,61
5	0,73	1,48	2,02	2,57	3,36	4,03	5,89	6,87
6	0,72	1,44	1,94	2,45	3,14	3,71	5,21	5,96
7	0,71	1,41	1,89	2,36	3	3,5	4,79	5,41

Si	0,75	0,9	0,95	0,975	0,99	0,995	0,999	0,9995
8	0,71	1,4	1,86	2,31	2,9	3,36	4,5	5,04
9	0,7	1,38	1,83	2,26	2,82	3,25	4,3	4,78
10	0,7	1,37	1,81	2,23	2,76	3,17	4,14	4,59
11	0,7	1,36	1,8	2,2	2,72	3,11	4,02	4,44
12	0,7	1,36	1,78	2,18	2,68	3,05	3,93	4,32
13	0,69	1,35	1,77	2,16	2,65	3,01	3,85	4,22
14	0,69	1,35	1,76	2,14	2,62	2,98	3,79	4,14
15	0,69	1,34	1,75	2,13	2,6	2,95	3,73	4,07
16	0,69	1,34	1,75	2,12	2,58	2,92	3,69	4,01
17	0,69	1,33	1,74	2,11	2,57	2,9	3,65	3,97
18	0,69	1,33	1,73	2,1	2,55	2,88	3,61	3,92
19	0,69	1,33	1,73	2,09	2,54	2,86	3,58	3,88
20	0,69	1,33	1,72	2,09	2,53	2,85	3,55	3,85
21	0,69	1,32	1,72	2,08	2,52	2,83	3,53	3,82
22	0,69	1,32	1,72	2,07	2,51	2,82	3,5	3,79
23	0,69	1,32	1,71	2,07	2,5	2,81	3,48	3,77
24	0,68	1,32	1,71	2,06	2,49	2,8	3,47	3,75
25	0,68	1,32	1,71	2,06	2,49	2,79	3,45	3,73
26	0,68	1,31	1,71	2,06	2,48	2,78	3,43	3,71
27	0,68	1,31	1,7	2,05	2,47	2,77	3,42	3,69
28	0,68	1,31	1,7	2,05	2,47	2,76	3,41	3,67

Si	0,75	0,9	0,95	0,975	0,99	0,995	0,999	0,9995
29	0,68	1,31	1,7	2,05	2,46	2,76	3,4	3,66
30	0,68	1,31	1,7	2,04	2,46	2,75	3,39	3,65
35	0,68	1,31	1,69	2,03	2,44	2,72	3,34	3,59
40	0,68	1,3	1,68	2,02	2,42	2,7	3,31	3,55
45	0,68	1,3	1,68	2,01	2,41	2,69	3,28	3,52
50	0,68	1,3	1,68	2,01	2,4	2,68	3,26	3,5
60	0,68	1,3	1,67	2	2,39	2,66	3,23	3,46
70	0,68	1,29	1,67	1,99	2,38	2,65	3,21	3,44
80	0,68	1,29	1,66	1,99	2,37	2,64	3,2	3,42
90	0,68	1,29	1,66	1,99	2,37	2,63	3,18	3,4
100	0,68	1,29	1,66	1,98	2,36	2,63	3,17	3,39
150	0,68	1,29	1,66	1,98	2,35	2,61	3,15	3,36
200	0,68	1,29	1,65	1,97	2,35	2,6	3,13	3,34
300	0,68	1,28	1,65	1,97	2,34	2,59	3,12	3,32
500	0,67	1,28	1,65	1,96	2,33	2,59	3,11	3,31
1000	0,67	1,28	1,65	1,96	2,33	2,58	3,1	3,3
1e+09	0,67	1,28	1,64	1,96	2,33	2,58	3,09	3,29

Die Chi-Quadrat-Verteilung

χ^2 -Quantilen

Si	0,01	0,025	0,05	0,1	0,25	0,5	0,75	0,9	0,95	0,975	0,99
df/Alpha	0,99	0,975	0,95	0,9	0,75	0,5	0,25	0,1	0,05	0,025	0,01
1	0	0	0	0,02	0,1	0,45	1,32	2,71	3,84	5,02	6,63
2	0,02	0,05	0,1	0,21	0,58	1,39	2,77	4,61	5,99	7,38	9,21
3	0,11	0,22	0,35	0,58	1,21	2,37	4,11	6,25	7,81	9,35	11,34
4	0,3	0,48	0,71	1,06	1,92	3,36	5,39	7,78	9,49	11,14	13,28
5	0,55	0,83	1,15	1,61	2,67	4,35	6,63	9,24	11,07	12,83	15,09
6	0,87	1,24	1,64	2,2	3,45	5,35	7,84	10,64	12,59	14,45	16,81
7	1,24	1,69	2,17	2,83	4,25	6,35	9,04	12,02	14,07	16,01	18,48
8	1,65	2,18	2,73	3,49	5,07	7,34	10,22	13,36	15,51	17,53	20,09
9	2,09	2,7	3,33	4,17	5,9	8,34	11,39	14,68	16,92	19,02	21,67
10	2,56	3,25	3,94	4,87	6,74	9,34	12,55	15,99	18,31	20,48	23,21
11	3,05	3,82	4,57	5,58	7,58	10,34	13,7	17,28	19,68	21,92	24,72
12	3,57	4,4	5,23	6,3	8,44	11,34	14,85	18,55	21,03	23,34	26,22
13	4,11	5,01	5,89	7,04	9,3	12,34	15,98	19,81	22,36	24,74	27,69
14	4,66	5,63	6,57	7,79	10,17	13,34	17,12	21,06	23,68	26,12	29,14
15	5,23	6,26	7,26	8,55	11,04	14,34	18,25	22,31	25	27,49	30,58
16	5,81	6,91	7,96	9,31	11,91	15,34	19,37	23,54	26,3	28,85	32
17	6,41	7,56	8,67	10,09	12,79	16,34	20,49	24,77	27,59	30,19	33,41

Si	0,01	0,025	0,05	0,1	0,25	0,5	0,75	0,9	0,95	0,975	0,99
18	7,01	8,23	9,39	10,86	13,68	17,34	21,6	25,99	28,87	31,53	34,81
19	7,63	8,91	10,12	11,65	14,56	18,34	22,72	27,2	30,14	32,85	36,19
20	8,26	9,59	10,85	12,44	15,45	19,34	23,83	28,41	31,41	34,17	37,57
21	8,9	10,28	11,59	13,24	16,34	20,34	24,93	29,62	32,67	35,48	38,93
22	9,54	10,98	12,34	14,04	17,24	21,34	26,04	30,81	33,92	36,78	40,29
23	10,2	11,69	13,09	14,85	18,14	22,34	27,14	32,01	35,17	38,08	41,64
24	10,86	12,4	13,85	15,66	19,04	23,34	28,24	33,2	36,42	39,36	42,98
25	11,52	13,12	14,61	16,47	19,94	24,34	29,34	34,38	37,65	40,65	44,31
26	12,2	13,84	15,38	17,29	20,84	25,34	30,43	35,56	38,89	41,92	45,64
27	12,88	14,57	16,15	18,11	21,75	26,34	31,53	36,74	40,11	43,19	46,96
28	13,56	15,31	16,93	18,94	22,66	27,34	32,62	37,92	41,34	44,46	48,28
29	14,26	16,05	17,71	19,77	23,57	28,34	33,71	39,09	42,56	45,72	49,59
30	14,95	16,79	18,49	20,6	24,48	29,34	34,8	40,26	43,77	46,98	50,89
35	18,51	20,57	22,47	24,8	29,05	34,34	40,22	46,06	49,8	53,2	57,34
40	22,16	24,43	26,51	29,05	33,66	39,34	45,62	51,81	55,76	59,34	63,69
45	25,9	28,37	30,61	33,35	38,29	44,34	50,98	57,51	61,66	65,41	69,96
50	29,71	32,36	34,76	37,69	42,94	49,33	56,33	63,17	67,5	71,42	76,15
60	37,48	40,48	43,19	46,46	52,29	59,33	66,98	74,4	79,08	83,3	88,38
70	45,44	48,76	51,74	55,33	61,7	69,33	77,58	85,53	90,53	95,02	100,4
80	53,54	57,15	60,39	64,28	71,14	79,33	88,13	96,58	101,9	106,6	112,3
90	61,75	65,65	69,13	73,29	80,62	89,33	98,65	107,6	113,2	118,1	124,1

Si	0,01	0,025	0,05	0,1	0,25	0,5	0,75	0,9	0,95	0,975	0,99
100	70,06	74,22	77,93	82,36	90,13	99,33	109,1	118,5	124,3	129,6	135,8
150	112,7	118	122,7	128,3	138	149,3	161,3	172,6	179,6	185,8	193,2
200	156,4	162,7	168,3	174,8	186,2	199,3	213,1	226	234	241,1	249,4
300	246	253,9	260,9	269,1	283,1	299,3	316,1	331,8	341,4	349,9	359,9
500	429,4	439,9	449,1	459,9	478,3	499,3	521	540,9	553,1	563,9	576,5
1000	898,9	914,3	927,6	943,1	969,5	999,3	1030	1058	1075	1090	1107

Die F -Verteilung

Quantilen der F -Verteilung ($\alpha = 5\%$)

	1	2	3	4	5	6	7	8	9	10	15	20
1	161,4	199,5	215,7	224,6	230,2	234	236,8	238,9	240,5	241,9	245,9	248
2	18,51	19	19,16	19,25	19,3	19,33	19,35	19,37	19,38	19,4	19,43	19,
3	10,13	9,55	9,28	9,12	9,01	8,94	8,89	8,85	8,81	8,79	8,7	8,6
4	7,71	6,94	6,59	6,39	6,26	6,16	6,09	6,04	6	5,96	5,86	5,8
5	6,61	5,79	5,41	5,19	5,05	4,95	4,88	4,82	4,77	4,74	4,62	4,5
6	5,99	5,14	4,76	4,53	4,39	4,28	4,21	4,15	4,1	4,06	3,94	3,8
7	5,59	4,74	4,35	4,12	3,97	3,87	3,79	3,73	3,68	3,64	3,51	3,4
8	5,32	4,46	4,07	3,84	3,69	3,58	3,5	3,44	3,39	3,35	3,22	3,1
9	5,12	4,26	3,86	3,63	3,48	3,37	3,29	3,23	3,18	3,14	3,01	2,9
10	4,96	4,1	3,71	3,48	3,33	3,22	3,14	3,07	3,02	2,98	2,85	2,7
11	4,84	3,98	3,59	3,36	3,2	3,09	3,01	2,95	2,9	2,85	2,72	2,6
12	4,75	3,89	3,49	3,26	3,11	3	2,91	2,85	2,8	2,75	2,62	2,5
13	4,67	3,81	3,41	3,18	3,03	2,92	2,83	2,77	2,71	2,67	2,53	2,4
14	4,6	3,74	3,34	3,11	2,96	2,85	2,76	2,7	2,65	2,6	2,46	2,3
15	4,54	3,68	3,29	3,06	2,9	2,79	2,71	2,64	2,59	2,54	2,4	2,3
16	4,49	3,63	3,24	3,01	2,85	2,74	2,66	2,59	2,54	2,49	2,35	2,2
17	4,45	3,59	3,2	2,96	2,81	2,7	2,61	2,55	2,49	2,45	2,31	2,2
18	4,41	3,55	3,16	2,93	2,77	2,66	2,58	2,51	2,46	2,41	2,27	2,1

	1	2	3	4	5	6	7	8	9	10	15	20
19	4,38	3,52	3,13	2,9	2,74	2,63	2,54	2,48	2,42	2,38	2,23	2,1
20	4,35	3,49	3,1	2,87	2,71	2,6	2,51	2,45	2,39	2,35	2,2	2,1
22	4,3	3,44	3,05	2,82	2,66	2,55	2,46	2,4	2,34	2,3	2,15	2,0
24	4,26	3,4	3,01	2,78	2,62	2,51	2,42	2,36	2,3	2,25	2,11	2,0
26	4,23	3,37	2,98	2,74	2,59	2,47	2,39	2,32	2,27	2,22	2,07	1,9
28	4,2	3,34	2,95	2,71	2,56	2,45	2,36	2,29	2,24	2,19	2,04	1,9
30	4,17	3,32	2,92	2,69	2,53	2,42	2,33	2,27	2,21	2,16	2,01	1,9
35	4,12	3,27	2,87	2,64	2,49	2,37	2,29	2,22	2,16	2,11	1,96	1,8
40	4,08	3,23	2,84	2,61	2,45	2,34	2,25	2,18	2,12	2,08	1,92	1,8
50	4,03	3,18	2,79	2,56	2,4	2,29	2,2	2,13	2,07	2,03	1,87	1,7
70	3,98	3,13	2,74	2,5	2,35	2,23	2,14	2,07	2,02	1,97	1,81	1,7
100	3,94	3,09	2,7	2,46	2,31	2,19	2,1	2,03	1,97	1,93	1,77	1,6
200	3,89	3,04	2,65	2,42	2,26	2,14	2,06	1,98	1,93	1,88	1,72	1,6
1000	3,85	3	2,61	2,38	2,22	2,11	2,02	1,95	1,89	1,84	1,68	1,5
1e+09	3,84	3	2,6	2,37	2,21	2,1	2,01	1,94	1,88	1,83	1,67	1,5

Quantilen der F -Verteilung ($\alpha = 1\%$)

	1	2	3	4	5	6	7	8	9	10	15	20
1	4052	5000	5403	5625	5764	5859	5928	5981	6022	6056	6157	620
2	98,5	99	99,17	99,25	99,3	99,33	99,36	99,37	99,39	99,4	99,43	99,
3	34,12	30,82	29,46	28,71	28,24	27,91	27,67	27,49	27,35	27,23	26,87	26,
4	21,2	18	16,69	15,98	15,52	15,21	14,98	14,8	14,66	14,55	14,2	14,
5	16,26	13,27	12,06	11,39	10,97	10,67	10,46	10,29	10,16	10,05	9,72	9,5
6	13,75	10,92	9,78	9,15	8,75	8,47	8,26	8,1	7,98	7,87	7,56	7,4
7	12,25	9,55	8,45	7,85	7,46	7,19	6,99	6,84	6,72	6,62	6,31	6,1
8	11,26	8,65	7,59	7,01	6,63	6,37	6,18	6,03	5,91	5,81	5,52	5,3
9	10,56	8,02	6,99	6,42	6,06	5,8	5,61	5,47	5,35	5,26	4,96	4,8
10	10,04	7,56	6,55	5,99	5,64	5,39	5,2	5,06	4,94	4,85	4,56	4,4
11	9,65	7,21	6,22	5,67	5,32	5,07	4,89	4,74	4,63	4,54	4,25	4,1
12	9,33	6,93	5,95	5,41	5,06	4,82	4,64	4,5	4,39	4,3	4,01	3,8
13	9,07	6,7	5,74	5,21	4,86	4,62	4,44	4,3	4,19	4,1	3,82	3,6
14	8,86	6,51	5,56	5,04	4,69	4,46	4,28	4,14	4,03	3,94	3,66	3,5
15	8,68	6,36	5,42	4,89	4,56	4,32	4,14	4	3,89	3,8	3,52	3,3
16	8,53	6,23	5,29	4,77	4,44	4,2	4,03	3,89	3,78	3,69	3,41	3,2
17	8,4	6,11	5,18	4,67	4,34	4,1	3,93	3,79	3,68	3,59	3,31	3,1
18	8,29	6,01	5,09	4,58	4,25	4,01	3,84	3,71	3,6	3,51	3,23	3,0

	1	2	3	4	5	6	7	8	9	10	15	20
19	8,18	5,93	5,01	4,5	4,17	3,94	3,77	3,63	3,52	3,43	3,15	3
20	8,1	5,85	4,94	4,43	4,1	3,87	3,7	3,56	3,46	3,37	3,09	2,9
22	7,95	5,72	4,82	4,31	3,99	3,76	3,59	3,45	3,35	3,26	2,98	2,8
24	7,82	5,61	4,72	4,22	3,9	3,67	3,5	3,36	3,26	3,17	2,89	2,7
26	7,72	5,53	4,64	4,14	3,82	3,59	3,42	3,29	3,18	3,09	2,81	2,6
28	7,64	5,45	4,57	4,07	3,75	3,53	3,36	3,23	3,12	3,03	2,75	2,6
30	7,56	5,39	4,51	4,02	3,7	3,47	3,3	3,17	3,07	2,98	2,7	2,5
35	7,42	5,27	4,4	3,91	3,59	3,37	3,2	3,07	2,96	2,88	2,6	2,4
40	7,31	5,18	4,31	3,83	3,51	3,29	3,12	2,99	2,89	2,8	2,52	2,3
50	7,17	5,06	4,2	3,72	3,41	3,19	3,02	2,89	2,78	2,7	2,42	2,2
70	7,01	4,92	4,07	3,6	3,29	3,07	2,91	2,78	2,67	2,59	2,31	2,1
100	6,9	4,82	3,98	3,51	3,21	2,99	2,82	2,69	2,59	2,5	2,22	2,0
200	6,76	4,71	3,88	3,41	3,11	2,89	2,73	2,6	2,5	2,41	2,13	1,9
1000	6,66	4,63	3,8	3,34	3,04	2,82	2,66	2,53	2,43	2,34	2,06	1,9
1e+09	6,63	4,61	3,78	3,32	3,02	2,8	2,64	2,51	2,41	2,32	2,04	1,8

Übungsaufgaben

Fehler 1-ter und 2-ter Art

1. Erklären Sie was man unter Fehler 1-ter und 2-ter Art versteht.
2. Warum kann man α nicht beliebig genau wählen?

3. Eine Maschine produziert im Schnitt 2% Ausschuss. Unter 100 Stück finden sich 5 Defekte. Die Nullhypothese ist das die Maschine 2% Ausschuss hat. Berechnen Sie Wahrscheinlichkeit des α -Fehlers (also bei 5 Stück oder mehr Ausfällen fälschlicherweise die Nullhypothese abzulehnen) unter Annahme von binomialverteilten Abweichungen.

Frage 3 ...

Lösungen

► Show code cell content

Hypothesentest - unabhängige Stichproben, $\sigma_1 \approx \sigma_2$

Bei 2 Stichproben aus 2 Grundgesamtheiten erhalten Sie folgende Mittelwerte und Standardabweichungen: $\mu_1 = 61, \sigma_1 = 15,5, n_1 = 15, \mu_2 = 48,4 \sigma_2 = 18,1, n_2 = 12$. Welchen Hypothesentest müssen Sie anwenden um zu prüfen ob $\mu_1 > \mu_2$ gilt?

1. Formulieren Sie die geeignete Null- und Alternativhypothese.
2. Berechnen Sie die Teststatistik.
3. Berechnen Sie den kritischen Wert (entweder mit Python oder Wahrscheinlichkeitstabelle) bei einem Signifikanzniveau $\alpha = 0,01$. Wird H_0 abgelehnt? Hierbei ergibt sich der kritische Wert bei $\alpha = 0,01$ für einen rechtseitigen Test $\mu_1 > \mu_2$ mit der t -Verteilung $(1 - \alpha, t_{df})$.
4. Interpretieren Sie das Ergebnis

Die gewichtete Standardabweichung ergibt sich zu:

$$s_g = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}},$$

Die Teststatistik ist t -verteilt und ergibt sich für $(\mu_1 - \mu_2) = 0$ zu:

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{s_g \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Frage 3 ...

Frage 4 ...

Lösung

► Show code cell content

► Show code cell content

2-Stichproben t -Test

Führen Sie einen 2-Stichproben t -Test für folgende unabhängige Daten mit gleicher Standardabweichung bei einem Signifikanzniveau $\alpha = 0,01$ aus:

```
from scipy.stats import norm

n = 100
a = norm.rvs(loc=0, scale=2, size=n, random_state=1)
b = norm.rvs(loc=1, scale=2, size=n, random_state=1)
```

Überprüfen Sie die Nullhypothese:

$$H_0 : \mu_1 = \mu_2$$

und

alternative Hypothese:

$$H_A : \mu_1 \neq \mu_2$$

1. für den p -Wert Ansatz
2. für den kritischen Wert
3. Interpretieren Sie das Ergebnis

Lösungen

► Show code cell content

► Show code cell content

Lernziele

Varianzanalyse - ANOVA

- Das Grundkonzept von Varianzanalyse wird am Beispiel des einfaktoriellen ANOVA-Hypothesentests erklärt
- Maße der Variabilität der beobachteten Variable (SST) zwischen Gruppen (SSG) und innerhalb der Gruppen (SSE) werden definiert
- Das Problem der Alphafehler-Kumulierung bei multiplen Testprozeduren wird demonstriert und mögliche Korrekturen wie Bonferroni-Korrektur und Tukey's Test werden vermittelt
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Varianzanalyse , einfaktorieller ANOVA-Hypothesentest , Multiples Testproblem , Alphafehler-Kumulierung , Bonferroni-Korrektur , Tukey's Test

Varianzanalyse - ANOVA

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import f, f_oneway, ttest_ind
import statsmodels.api as smi
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

In diesem Abschnitt wird eine Methode namens [Varianzanalyse](#) (englisch analysis of variance, kurz ANOVA) erörtert, mit der mehrere Mittelwerte einer Grundgesamtheit verglichen werden können.

Genauer gesagt geht es um die sogenannte einfaktorielle Varianzanalyse (s.478). Bei dieser Art der ANOVA werden die Mittelwerte einer Variablen verglichen, die sich aus der Klassifizierung durch *eine* andere Variable, dem sogenannten **Faktor**, ergeben. Um das Konzept zu veranschaulichen, wollen wir ein einfaches Beispiel entwerfen. Angenommen, wir vergleichen die mittleren Jahresgehälter der in Deutschland lebenden Personen, gruppiert nach Bundesländern, dann erhalten wir 16 Mittelwerte der Jahresgehälter, einen für jedes Bundesland. In diesem Beispiel ist die Bundeslandvariable also eine Klassifikationsvariable, der so genannte Faktor.

Bitte beachten Sie, dass die einfaktorielle Varianzanalyse die Verallgemeinerung des 2-Stichproben *t*-Tests auf mehr als zwei Grundgesamtheiten ist.

Einfaktorielle ANOVA

Die grundlegende Logik einer einfaktoriellen ANOVA besteht darin, aus jeder Gruppe unabhängige Zufallsstichproben zu ziehen, dann die Stichprobenmittelwerte für jede Gruppe zu berechnen und anschließend die Variation der Stichprobenmittelwerte zwischen den Gruppen mit der Variation innerhalb der Gruppen zu vergleichen. Schließlich wird auf der Grundlage einer Teststatistik entschieden, ob die Mittelwerte der Gruppen gleich sind oder nicht.

Auf der Grundlage dieser Logik benötigen wir quantitative **Maße für die Variabilität**. Daher teilen wir die Gesamtvariabilität in zwei Segmente auf: Eines, das die **Variabilität zwischen den Gruppen** berücksichtigt, und das andere, das die **Variabilität innerhalb der Gruppen** berücksichtigt.

Maße der Variabilität

Wir führen drei quantitative Maße für die Variation ein:

- Summe der Gesamtquadrate (SST)
- Summe der Gruppenquadrate (SSG)
- Summe der Fehlerquadrate (SSE)

Die **Summe der Gesamtquadrate (SST)** ist ein Maß für die Gesamtvariabilität der Variablen. Sie ist gegeben durch

$$SST = \sum_{i=1}^n (x_i - \bar{x})^2,$$

wobei x_i den Beobachtungen in den Stichproben entspricht und \bar{x} für den Gesamtmittelwert aller Stichproben steht.

Die **Summe der Gruppenquadrate (SSG)** ist ein Maß für die Variabilität zwischen Gruppen und entspricht der quadrierten Abweichung der Gruppenmittelwerte vom Gesamtmittelwert, gewichtet mit dem Stichprobenumfang.

$$SSG = \sum_{i=1}^n n_j (\bar{x}_i - \bar{x})^2$$

Dabei steht n_j für den Stichprobenumfang der Gruppe j , \bar{x}_i für den Mittelwert der Gruppe j und \bar{x} für den Gesamtmittelwert der Stichprobe.

Die **Summe der Fehlerquadrate (SSE)** schließlich ist ein Maß für die Variabilität innerhalb der Gruppen. Sie steht im Zusammenhang mit der unerklärten Variabilität, d. h. der Variabilität, die nicht durch die Gruppenvariable erklärt werden kann. Die Summe der Fehlerquadrate ist gegeben durch

$$SSE = \sum_{i=1}^n (n_j - 1) s_j^2,$$

wobei n_j den Stichprobenumfang für Gruppe j und s_j^2 die Varianz von Gruppe j bezeichnet. Alternativ kann man SSE sowie die Differenz von SST und SSG berechnen

$$SSE = SST - SSG.$$

Maße der mittleren Variabilität

Bisher haben wir Maße für die Gesamtvariabilität (SST), die Variabilität zwischen Gruppen (SSG) und die Variabilität innerhalb von Gruppen (SSE) berechnet. Um eine durchschnittliche Variabilität

zu erhalten, skalieren wir im nächsten Schritt diese Variabilitätsmaße mit dem Stichprobenumfang (genauer gesagt mit den Freiheitsgraden, df).

Die **Freiheitsgrade** werden für jede Unterteilung der Variabilität (Gesamtvariabilität, Variabilität zwischen Gruppen und Variabilität innerhalb von Gruppen) definiert.

- Gesamtvariabilität

$$df_T = n - 1,$$

wobei n den Gesamtumfang der Stichprobe bezeichnet. den Gesamtumfang der Stichprobe bezeichnet.

- Variabilität zwischen den Gruppen

$$df_G = k - 1,$$

wobei k die Anzahl der Gruppen bezeichnet. den Gesamtumfang der Stichprobe bezeichnet.

- Variabilität innerhalb der Gruppe

$$df_E = n - k.$$

Nun können wir die **mittleren Quadrate** für die Variabilität zwischen den Gruppen und die Variabilität innerhalb der Gruppen berechnen. Die durchschnittliche Variabilität zwischen und innerhalb der Gruppen wird als die Gesamtvariabilität, skaliert mit den zugehörigen Freiheitsgraden, berechnet.

- Mittlere Variabilität zwischen den Gruppen

$$MSG = \frac{SSG}{df_G}$$

- Mittlere Variabilität innerhalb der Gruppe

$$MSE = \frac{SSE}{df_E}$$

Teststatistik und p -Wert

Schließlich vergleichen wir die mittlere Variation zwischen den Gruppen, MSG , mit der Variation innerhalb der Gruppe, MSE . Daher berechnen wir das Verhältnis zwischen der durchschnittlichen Variation zwischen den Gruppen (MSG) und der Variation innerhalb der Gruppen (MSE), das mit F bezeichnet wird.

$$F = \frac{MSG}{MSE}$$

Die F -Statistik hat die F -Verteilung (benannt nach [Sir Ronald A. Fisher](#)) mit

$$df = (k - 1, n - k),$$

wobei k für die Anzahl der Gruppen und n für den Stichprobenumfang steht. Große Werte der F -Werte zeigen an, dass die Variation zwischen den Stichprobenmittelwerten der Gruppen im Verhältnis zur Variation innerhalb der Gruppe groß ist. Darüber hinaus können wir den p -Wert für jeden gegebenen F -Wert berechnen. Wenn der p -Wert klein ist, liefern die Daten überzeugende Beweise dafür, dass sich mindestens ein Paar von Gruppenmittelwerten voneinander unterscheidet. Ist der p -Wert groß, liefern die Daten keinen überzeugenden Beweis dafür, dass sich zumindest ein Paar von Gruppenmittelwerten voneinander unterscheidet, und die beobachteten Unterschiede in den Stichprobenmittelwerten sind somit auf Stichprobenvariabilität (oder Zufall) zurückzuführen.

Einfaktorielle ANOVA-Tabellen

Wie oben dargestellt, umfasst die einfache Varianzanalyse mehrere Analyseschritte. Dabei ist eine gängige Methode zur Darstellung einer einfachen ANOVA die so genannte **einfaktorielle ANOVA-Tabelle**. Der allgemeine Aufbau einer solchen Tabelle ist unten dargestellt.

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	$k - 1$	SSG	$MSG = \frac{SSG}{k-1}$	$F = \frac{MSG}{MSE}$	p
Fehler/Residuen	$n - k$	SSE	$MSE = \frac{SSE}{n-k}$		
Insgesamt	$n - 1$	SST			

Einfaktorieller ANOVA-Hypothesentest

Der Zweck eines **einfaktoriellen ANOVA-Hypothesentests** ist der Vergleich von k Grundgesamtheits-/Gruppenmittelwerten, $\mu_1, \mu_2, \dots, \mu_k$. Die folgenden Annahmen müssen erfüllt sein, damit eine einseitige ANOVA angewendet werden kann (s.485):

- Zufällige Stichproben
- Unabhängige Stichproben
- Für jede Grundgesamtheit ist die betrachtete Variable normalverteilt.
- Die Standardabweichungen der betrachteten Variable sind für alle Grundgesamtheiten gleich.

Ein einfaktorieller ANOVA-Hypothesentest folgt demselben schrittweisen Verfahren wie andere Hypothesentests.

Schritt 1	Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an
Schritt 2	Legen Sie das Signifikanzniveau, α fest
Schritt 3	Berechnen Sie den Wert der Teststatistik
Schritt 4	Bestimmen Sie den p-Wert
Schritt 5	Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen
Schritt 6	Interpretieren Sie das Ergebnis des Hypothesentests

Einfaktorieller ANOVA-Hypothesentest: Ein Beispiel

Um praktische Erfahrungen zu sammeln, wenden wir den **einfaktoriellen ANOVA-Hypothesentest** in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein
students = pd.read_csv("../data/students.csv")
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: *stud_id, name, gender, age, height, weight, religion, nc_score, semester, major, minor, score1, score2, online_tutorial, graduated, salary*.

Um den **einfaktoriellen ANOVA-Hypothesentest** zu veranschaulichen, untersuchen wir das mittlere Jahresgehalt der Absolventen, gruppiert nach ihrem Hauptstudienfach. Zur Verdeutlichung: In diesem Beispiel erfolgt die Klassifizierung/Gruppierung durch eine Variable, die Hauptvariable, den so genannten Faktor, wir führen also eine einfaktorielle ANOVA durch. In dieser Übung **wollen wir testen, ob sich das mittlere Jahresgehalt der Absolventen zwischen den Absolventen verschiedener Studienfächer unterscheidet**.

Datenexploration und Aufbereitung

Wir beginnen unsere Datenanalyse mit dem Zufallsstichprobenverfahren. Wir ziehen eine Zufallsstichprobe von 275 Studenten aus dem Datensatz mit Hilfe der `sample()` Methode in Python. Wir wollen sicherstellen, dass wir nur Studenten mit abgeschlossenem Studium in die Stichprobe aufnehmen, deshalb unterteilen wir die Daten des Dataframes zuvor in Python. Des Weiteren reduzieren wir unseren Datensatz auf die zwei Variablen von Interesse, die kategoriale Variable `major` und die numerische Variable `salary`. Dann zeigen wir die ersten 10 Zeilen des Datensatzes an.

```

n = 275
data = students.loc[students["graduated"] == 1, ["major", "salary"]].sample(
    n, random_state=1
)
data.head(10)

```

	major	salary
5018	Biology	49976.94
4476	Political Science	27821.99
240	Environmental Sciences	37668.89
7165	Biology	60958.21
2488	Economics and Finance	59920.48
1193	Mathematics and Statistics	46116.22
241	Social Sciences	35881.74
2968	Economics and Finance	48456.79
1475	Mathematics and Statistics	61384.55
6764	Biology	61819.85

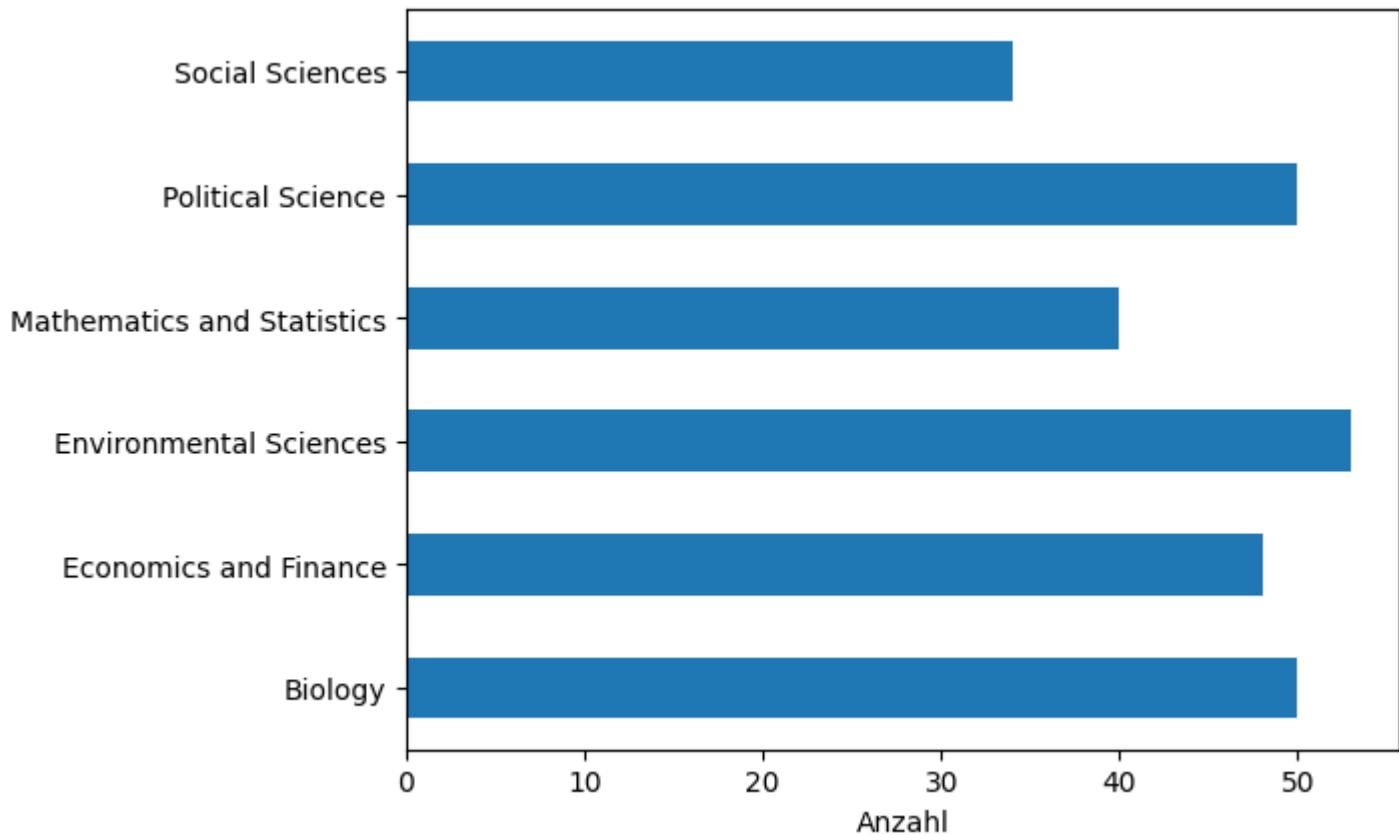
Aus reinem Interesse visualisieren wir die Zählungen für jede der 6 verschiedenen Studienteilnehmer in unserer Stichprobe, indem wir ein Balkendiagramm erstellen. Wir verwenden die `barh()` Funktion aus der `matplotlib` Bibliothek für die Darstellung.

```

fig, ax = plt.subplots()
data.groupby("major")["salary"].count().plot.barh(ax=ax)
ax.set_ylabel("")
ax.set_xlabel("Anzahl")

```

```
Text(0.5, 0, 'Anzahl')
```

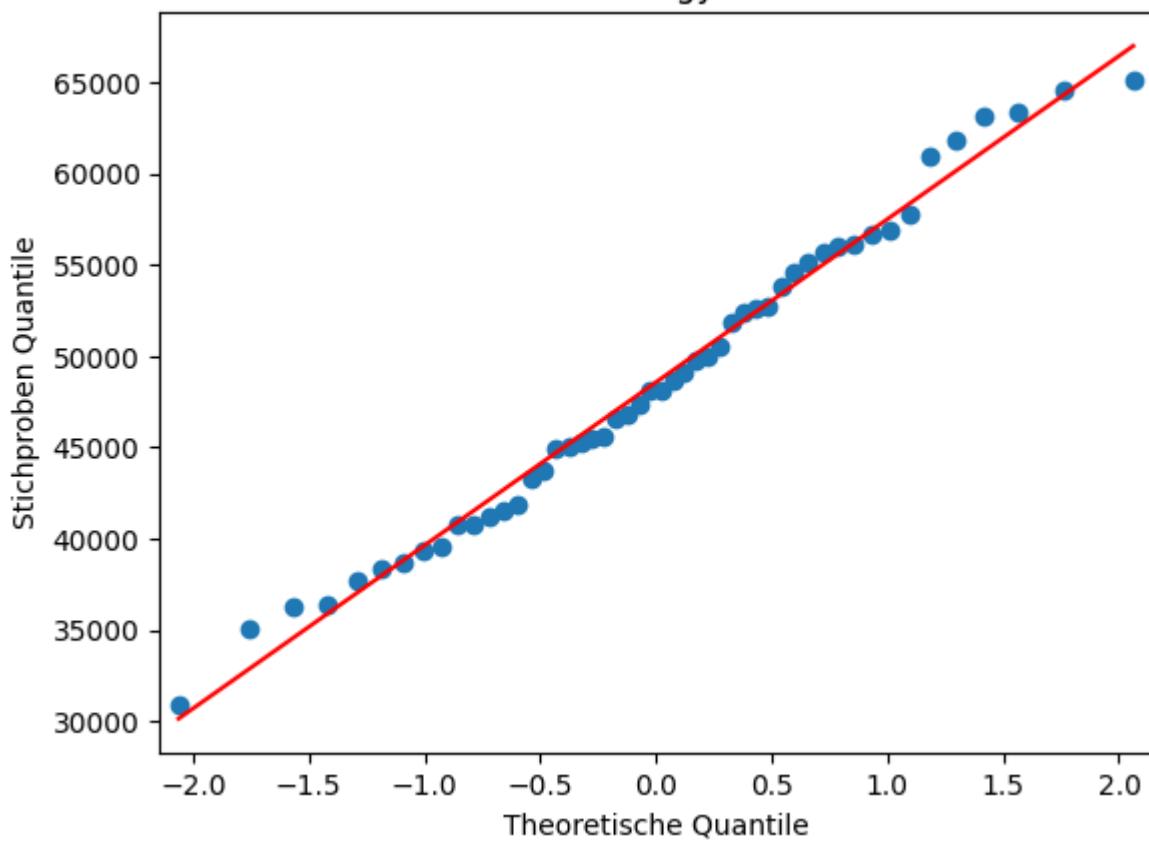


Wir sehen, dass die verschiedenen Studienteilnehmer in unserer Stichprobe nicht gleich verteilt sind, aber das ist in Ordnung.

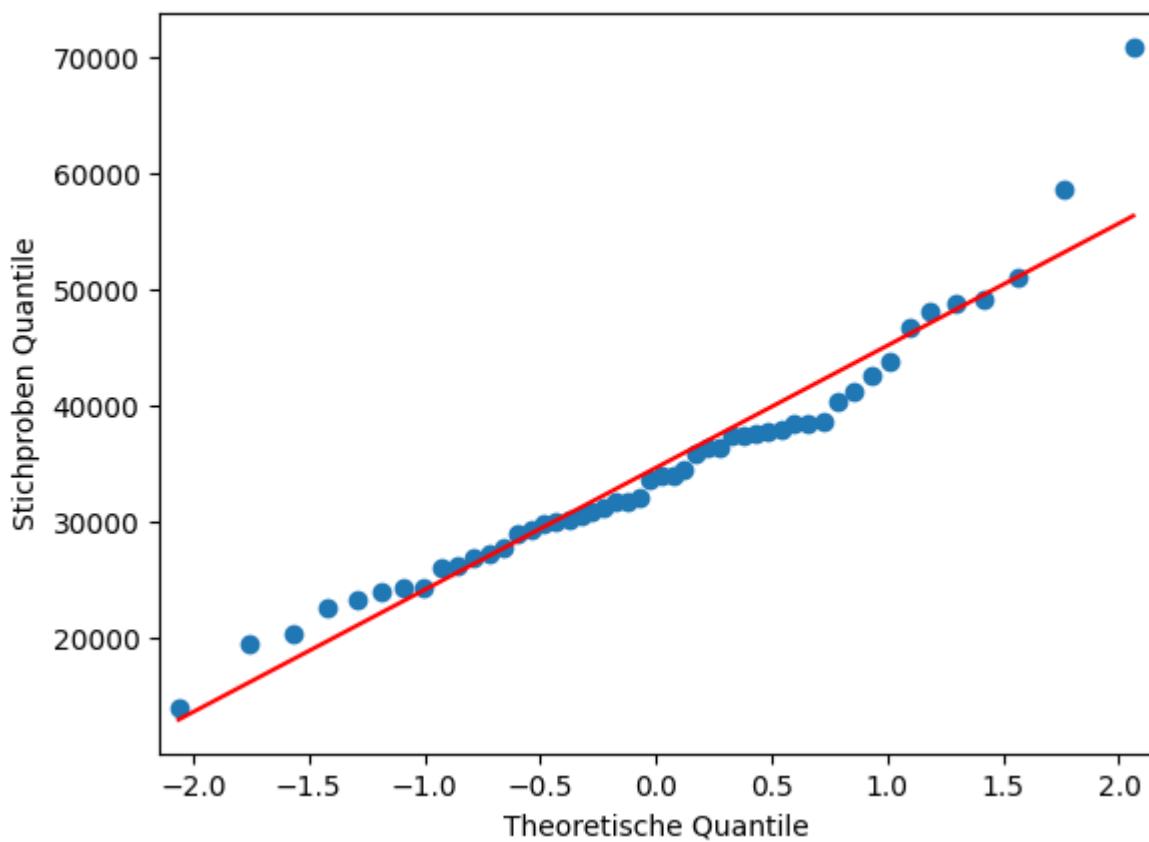
Bevor wir mit dem Hypothesentest beginnen, überprüfen wir, ob die Annahmen für den einfaktoriellen ANOVA-Hypothesentest erfüllt sind. Die Stichproben sind Zufallsstichproben und unabhängig. Das ist in Ordnung. Wir überprüfen die Normalitätsannahme, indem wir für jede gruppierte Variable ein Normalwahrscheinlichkeitsdiagramm ([Q-Q-Diagramm](#)) erstellen.

```
# Erzeuge Q-Q Plot
for _major in data.major.unique():
    _ = smi.qqplot(data.loc[data["major"] == _major, "salary"], line="r")
    ax = plt.gca()
    ax.set_title(_major)
    ax.set_xlabel("Theoretische Quantile")
    ax.set_ylabel("Stichproben Quantile")
```

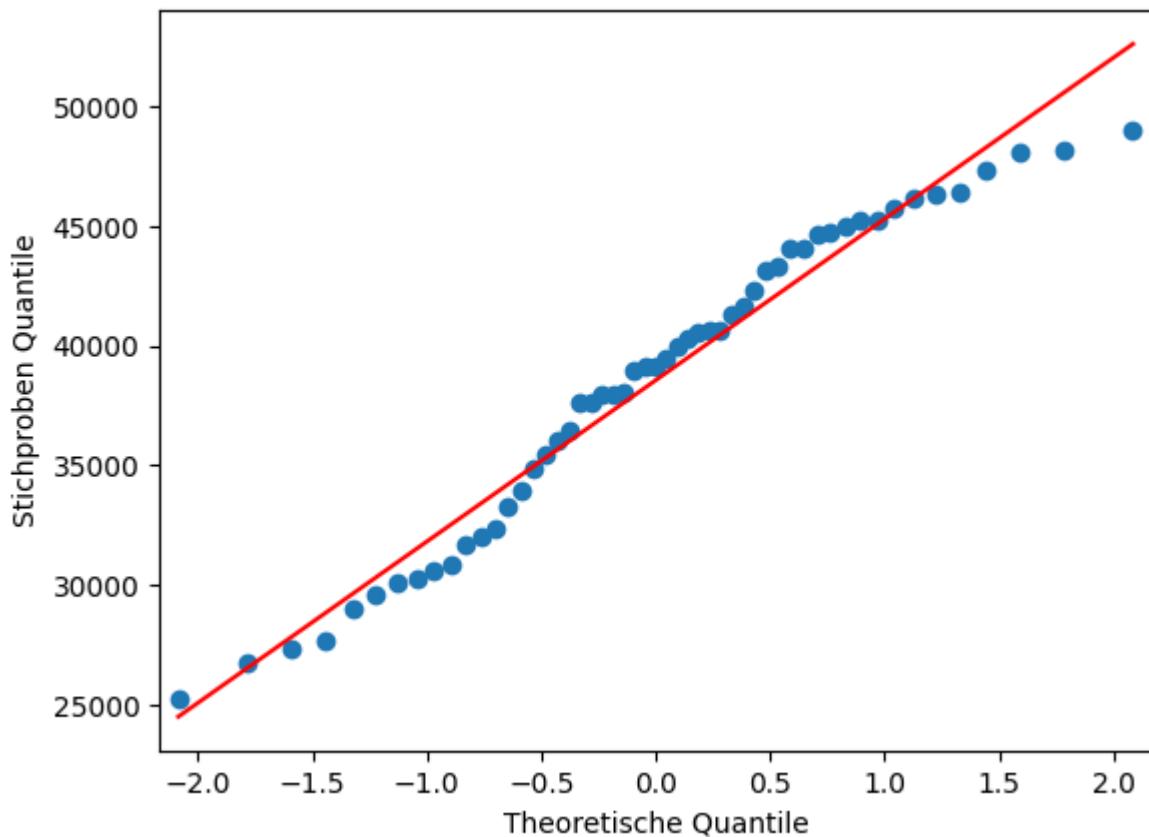
Biology



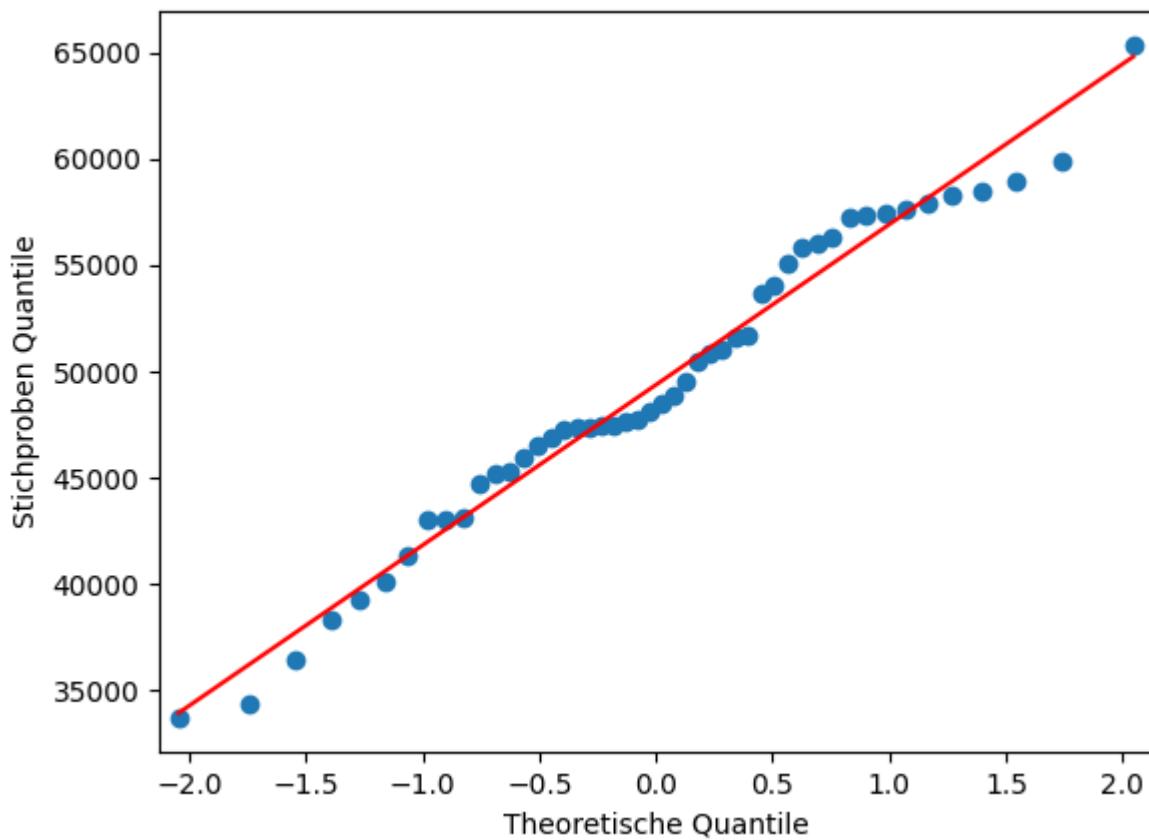
Political Science



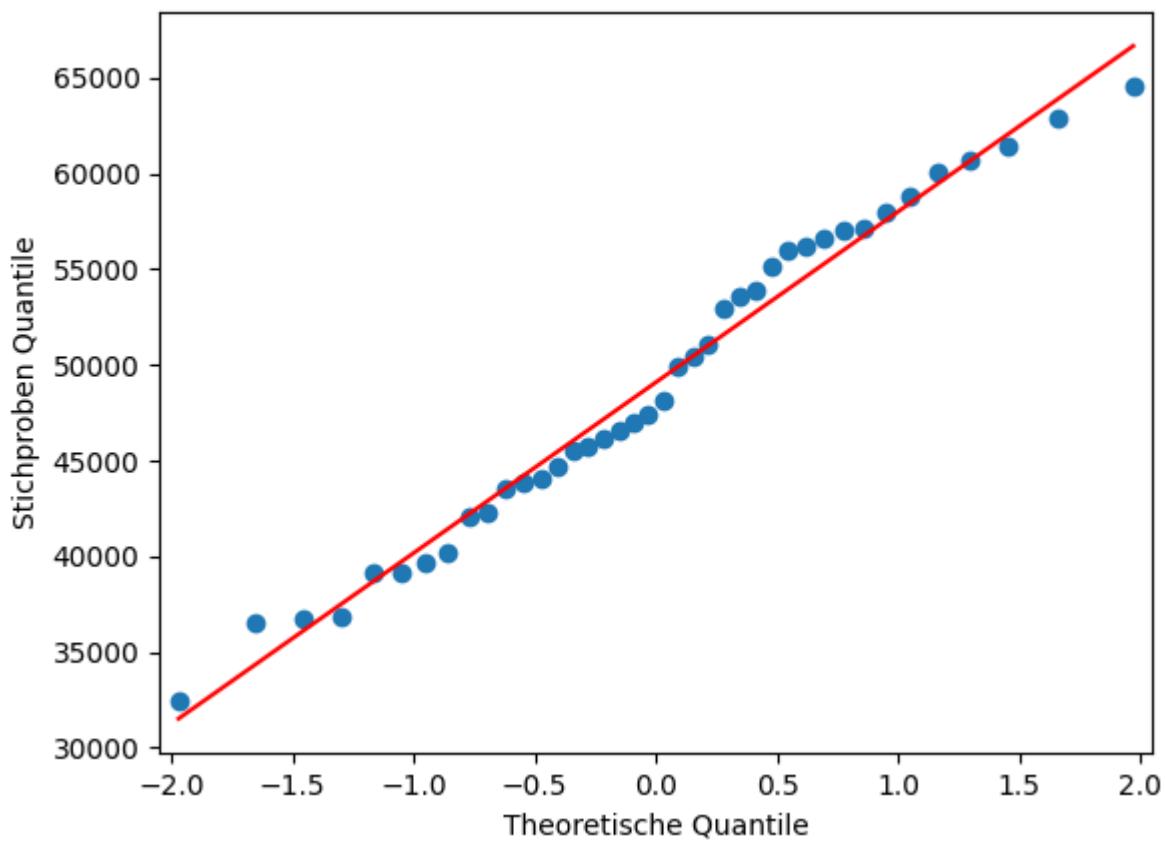
Environmental Sciences



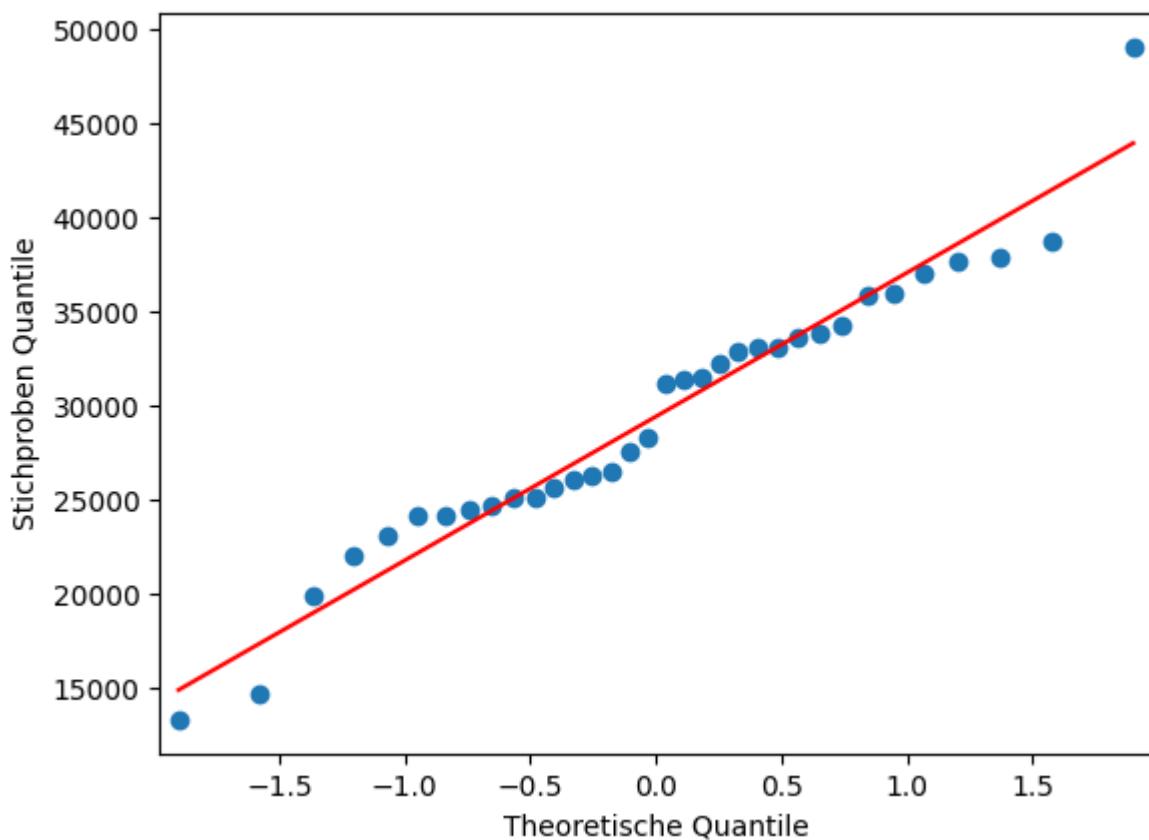
Economics and Finance



Mathematics and Statistics



Social Sciences



Als nächstes testen wir die Annahme gleicher Standardabweichungen. Dazu berechnen wir die Standardabweichung für jede Gruppe. Die Programmiersprache Python bietet einige ausgezeichnete Funktionen zur Berechnung statistischer Parameter für verschiedene Gruppierungen eines Datensatzes. Für unser Beispiel verwenden wir die Funktion `pivot_table()`, um die Standardabweichungen für jede Gruppe zu berechnen.

```
data.head()
```

	major	salary
5018	Biology	49976.94
4476	Political Science	27821.99
240	Environmental Sciences	37668.89
7165	Biology	60958.21
2488	Economics and Finance	59920.48

```
table = data.groupby("major")["salary"].std()  
table
```

```
major  
Biology           8470.015961  
Economics and Finance 7175.592211  
Environmental Sciences 6460.464104  
Mathematics and Statistics 8384.558228  
Political Science      10273.903364  
Social Sciences        7227.381320  
Name: salary, dtype: float64
```

Als Faustregel gilt die Annahme gleicher Standardabweichungen als erfüllt, wenn das Verhältnis der größten zur kleinsten Standardabweichung der Stichprobe kleiner als 2 ist.

```
ratio = table.max() / table.min()  
ratio
```

```
np.float64(1.590273268094179)
```

Das Verhältnis zwischen der größten und der kleinsten Standardabweichung der Stichprobe beträgt 1,59. Das liegt nahe am Schwellenwert von 2, ist aber für unser Musterbeispiel noch akzeptabel. Daraus können wir schließen, dass die Annahmen erfüllt sind.

Beachten Sie, dass die einseitige ANOVA robust gegenüber moderaten Verstößen gegen die Normalitätsannahme und die Annahme gleicher Standardabweichungen ist (s.486).

Überprüfung der Hypothesen

Um den **einfaktoriellen ANOVA-Hypothesentest** durchzuführen, folgen wir dem schrittweisen Durchführungsverfahren für Hypothesentests.

Schritt 1 : Geben Sie die Nullhypothese H_0 und alternative Hypothese H_A an

Die Nullhypothese besagt, dass das mittlere Jahresgehalt bei allen Gruppen von Absolventen gleich ist.

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 = \mu_6$$

Alternative Hypothese

$$H_A : \text{Nicht alle Mittelwerte sind gleich}$$

Schritt 2: Legen Sie das Signifikanzniveau, α fest

$$\alpha = 0,01$$

alpha = 0.01

Schritt 3 und 4: Berechnen Sie den Wert der Teststatistik und den p -Wert

Um die F -Test-Statistik zu berechnen, müssen wir zuvor mehrere Größen bestimmen. Zur Veranschaulichung berechnen wir die F -Teststatistik manuell in Python. Schritt für Schritt füllen wir die ANOVA-Tabelle aus, bis wir schließlich die F -Teststatistik und folglich den p -Wert erhalten.

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F -Statistik	p -Wert
Gruppe/Klasse	$k - 1$	SSG	$MSG = \frac{SSG}{k-1}$	$F = \frac{MSG}{MSE}$	p
Fehler/Residuen	$n - k$	SSE	$MSE = \frac{SSE}{n-k}$		
Insgesamt	$n - 1$	SST			

Wir beginnen mit der ersten Spalte und berechnen die Freiheitsgrade.

```
k = data["major"].nunique()
n = data.shape[0]

df_G = k - 1
df_G
```

5

```
df_E = n - k
df_E
```

269

```
df_T = n - 1
df_T
```

274

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	5	SSG	$MSG = \frac{SSG}{k-1}$	$F = \frac{MSG}{MSE}$	p
Fehler/Residuen	269	SSE	$MSE = \frac{SSE}{n-k}$		
Insgesamt	274	SST			

Außerdem berechnen wir die drei Summen der Quadrate, SST , SSG und SSE . Erinnern Sie sich an die Gleichungen von oben.

$$SST = \sum_{i=1}^n (x_i - \bar{x})^2,$$

wobei x_i den Beobachtungen in den Stichproben und \bar{x} dem Gesamtmittelwert aller Stichproben entspricht.

```
# Berechne Gesamtmittelwert
x_bar = data["salary"].mean()

# Beobachtungen
xi = data["salary"]

# Berechne SST
SST = np.sum((xi - x_bar) ** 2)
SST
```

```
np.float64(33169010775.846596)
```

$$SSG = \sum_{i=1}^n n_j (\bar{x}_i - \bar{x})^2$$

Dabei steht n_j für den Stichprobenumfang der Gruppe j , \bar{x}_i für den Mittelwert der Gruppe j und \bar{x} für den Gesamtmittelwert aller Stichproben.

```
n_j = data.groupby("major")["salary"].count()  
n_j
```

```
major  
Biology                  50  
Economics and Finance    48  
Environmental Sciences    53  
Mathematics and Statistics 40  
Political Science         50  
Social Sciences            34  
Name: salary, dtype: int64
```

```
# Berechne Stichprobengrösse für alle Gruppen  
n_j = data.groupby("major")["salary"].count()  
  
# Berechne Mittelwert für alle Gruppen  
xi_bar = data.groupby("major")["salary"].mean()  
  
# Berechne SSG  
SSG = np.sum(n_j * (xi_bar - x_bar) ** 2)  
SSG
```

```
np.float64(15425759982.382113)
```

$$SSE = \sum_{i=1}^n (n_j - 1)s_j^2,$$

wobei n_j den Stichprobenumfang für die Gruppe j und s_j^2 die Varianz der Gruppe j bezeichnet.

```
# Berechne Standardabweichung für die einzelnen Gruppen  
s2_j = data.groupby("major")["salary"].std()  
  
# Berechne SSE  
SSE = np.sum((n_j - 1.0) * s2_j**2.0)  
SSE
```

```
np.float64(17743250793.46448)
```

```
# alternativ kann SSE auch so berechnet werden
SSE2 = SST - SSG
SSE2
```

```
np.float64(17743250793.464485)
```

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	5	$1,542576 \cdot 10^{10}$	$MSG = \frac{SSG}{k-1}$	$F = \frac{MSG}{MSE}$	p
Fehler/Residuen	269	$1,774325 \cdot 10^{10}$	$MSE = \frac{SSE}{n-k}$		
Insgesamt	274	$3,316901 \cdot 10^{10}$			

Nun berechnen wir die beiden Maße für die mittlere Variabilität, MSG und MSE .

```
# Berechne MSG
MSG = SSG / (k - 1)
MSG
```

```
np.float64(3085151996.4764223)
```

```
# Berechne MSE
MSE = SSE / (n - k)
MSE
```

```
np.float64(65960040.124403276)
```

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	5	$1,542576 \cdot 10^{10}$	$3,085152 \cdot 10^9$	$F = \frac{MSG}{MSE}$	p
Fehler/Residuen	269	$1,774325 \cdot 10^{10}$	$6,596004 \cdot 10^7$		
Insgesamt	274	$3,316901 \cdot 10^{10}$			

Schließlich erhalten wir die F -Statistiken durch das Verhältnis von MSG und MSE .

```
# Berechne Teststatistik
Fstat = MSG / MSE
Fstat
```

```
np.float64(46.773046084533945)
```

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	5	$1,542576 \cdot 10^{10}$	$3,085152 \cdot 10^9$	46,77304	p
Fehler/Residuen	269	$1,774325 \cdot 10^{10}$	$6,596004 \cdot 10^7$		
Insgesamt	274	$3,316901 \cdot 10^{10}$			

Im letzten Schritt berechnen wir den p -Wert durch Aufruf der Funktion `f.cdf()` in Python. Erinnern Sie sich, wie man die Freiheitsgrade berechnet.

$$df = (k - 1, n - k)$$

Da die Nullhypothese nur dann abgelehnt wird, wenn die Teststatistik F zu groß ist, ist ein einfaktorieller ANOVA-Test immer rechtsschief.

```
df1 = k - 1
df2 = n - k

p_value = f.sf(Fstat, dfn=df1, dfd=df2)
p_value
```

```
np.float64(1.0932037477708007e-34)
```

Quelle	df	Summe der Quadrate (SS)	Mittlere Quadrate (MS)	F-Statistik	p-Wert
Gruppe/Klasse	5	$1,542576 \cdot 10^{10}$	$3,085152 \cdot 10^9$	46,77304	$1,093212 \cdot 10^{-34}$
Fehler/Residuen	269	$1,774325 \cdot 10^{10}$	$6,596004 \cdot 10^7$		
Insgesamt	274	$3,316901 \cdot 10^{10}$			

Schritt 5: Wenn $p \leq \alpha$, H_0 ablehnen; ansonsten H_0 nicht ablehnen

```
p_value <= alpha
```

```
np.True_
```

Der p -Wert ist kleiner als das angegebene Signifikanzniveau von 0,01; wir verwerfen H_0 . Die Testergebnisse sind statistisch signifikant auf dem 1%-Niveau und liefern einen sehr starken Beweis gegen die Nullhypothese.

Schritt 6: Interpretieren Sie das Ergebnis des Hypothesentests

$p = 1,093212 \cdot 10^{-34}$. Bei einem Signifikanzniveau von 1% liefern die Daten sehr starke Hinweise darauf, dass sich mindestens ein Paar von Gruppenmittelwerten voneinander unterscheidet.

Hypothesentests in Python

Wir haben gerade einen einfaktoriellen ANOVA-Hypothesentest in Python manuell durchgeführt. Toll, aber jetzt wiederholen wir das Beispiel und nutzen die Möglichkeiten von Python, um das gleiche Ergebnis wie oben mit nur wenigen Zeilen Code zu erhalten!

Um einen einseitigen ANOVA-Hypothesentest in Python durchzuführen, verwenden wir die Funktion `f_oneway()`. Die `f_oneway()`-Funktion erwartet die Eingabe der zu vergleichenden Untergruppen. Hierfür verwenden wir eine `list-comprehension`. Außerdem lesen wir die *F*-Teststatistik `statistics` und *p*-Wert `p_value` aus.

```
dat = [
    data.loc[data["major"] == major, "salary"].values
    for major in data["major"].unique()
]
statistics, pvalue = f_oneway(*dat)

print(f"Wert der F-Statistik: {statistics}")
print(f"p-Wert: {pvalue}")
```

```
Wert der F-Statistik: 46.77304608453393
p-Wert: 1.0932037477708007e-34
```

Es hat gut funktioniert! Vergleichen Sie die Ausgabe der Funktion `f_oneway()` mit unserem Ergebnis von oben. Auch hier können wir zu dem Schluss kommen, dass die Daten bei einem Signifikanzniveau von 1% sehr starke Hinweise darauf liefern, dass sich mindestens ein Paar von Gruppenmitteln voneinander unterscheidet.

Multiples Testproblem

Eine Einschränkung der ANOVA besteht darin, dass wir, wenn wir die Nullhypothese ablehnen, feststellen, dass die Mittelwerte der betrachteten Populationen nicht alle gleich sind. Wir können jedoch weder entscheiden, welche Mittelwerte unterschiedlich sind, noch, in welchem Verhältnis die Mittelwerte zueinander stehen.

Um diese Frage zu klären, wenden wir Methoden an, die als **multiples Testen** oder **Mehrfachvergleiche** bezeichnet werden. Das Problem bei Mehrfachvergleichen ist, dass je mehr Hypothesen für einen bestimmten Datensatz getestet werden, desto wahrscheinlicher ist es, dass die Nullhypothese fälschlicherweise zurückgewiesen wird. Daher erfordern die Methoden des Mehrfachvergleichs eine höhere Signifikanzschwelle (α) für einzelne Vergleiche, um die Anzahl der gezogenen Schlüsse zu kompensieren.

Alphafehler-Kumulierung (engl. Family Wise Error Rate - FWER)

Eine **Testfamilie** ist der Fachbegriff für eine Reihe von Tests, die an einem Datensatz durchgeführt werden. Die [Alphafehler-Kumulierung](#) (engl. Family Wise Error Rate) ist die Wahrscheinlichkeit, dass bei der Durchführung von Mehrfachhypothesentests ein oder mehrere falsche Zurückweisungen der Null-Hypothese oder Fehler vom [Typ I](#) gemacht werden.

Es sei daran erinnert, dass bei einem Signifikanzniveau von $\alpha = 0,05$ die Wahrscheinlichkeit, einen Fehler vom Typ I zu machen, $0,05$ oder 5% beträgt. Folglich ist die Wahrscheinlichkeit, keinen Fehler vom Typ I zu machen, $1 - \alpha = 1 - 0,05 = 0,95$. Außerdem ist die Wahrscheinlichkeit, zwei unabhängige Ereignisse zu beobachten, das Produkt ihrer Wahrscheinlichkeiten. Wenn wir also zwei unabhängige Tests durchführen, ist die Wahrscheinlichkeit, beim ersten und beim zweiten Test keinen Fehler vom Typ I zu machen

$$(1 - \alpha) \times (1 - \alpha) = (1 - \alpha)^2$$

Wenn $\alpha = 0,05$ ist, ergibt sich eine Wahrscheinlichkeit, dass beim ersten und zweiten Test kein Fehler vom Typ I auftritt, von

$$(1 - \alpha)^2 = (1 - 0,05)^2 = 0,95^2 \approx 0,902$$

Für eine Familie von C -Tests ist die Wahrscheinlichkeit, dass kein Fehler vom Typ I für die gesamte Familie auftritt, formal ausgedrückt

$$(1 - \alpha)^C.$$

Betrachten wir nun $C = 10$ und $\alpha = 0,05$. Wir führen also 10 Mehrfachvergleiche mit einem Datensatz durch. Die Wahrscheinlichkeit, keinen Fehler vom Typ I in der Familie zu machen, ist dann

$$(1 - \alpha)^C = (1 - 0,05)^{10} \approx 0,599$$

Folglich ist die Wahrscheinlichkeit, dass **ein oder mehrere Fehler vom Typ I** in der Testfamilie auftreten

$$1 - (1 - \alpha)^C$$

Für unser Beispiel finden wir

$$1 - (1 - \alpha)^C = 1 - (1 - 0,05)^{10} \approx 0,401$$

Somit ist bei $\alpha = 0,05$ für jeden der 10 Mehrfachvergleiche die Wahrscheinlichkeit, dass die Nullhypothese falsch zurückgewiesen wird, 0,401 oder 40,1%.

Um diesem Problem Rechnung zu tragen, gibt es [mehrere statistische Methoden](#). In diesem Abschnitt werden die [Bonferroni-Korrektur](#) und die [Tukey-Test](#), auch bekannt als **Tukey's HSD-Test (honestly significant difference)**, behandelt.

Beispiel-Daten

In diesem Abschnitt wiederholen wir das Beispiel aus dem vorherigen Abschnitt. Dort haben wir eine einfaktorielle ANOVA angewandt, um zu **testen, ob sich das mittlere Jahresgehalt der Absolventen zwischen den Absolventen verschiedener Studienfächer unterscheidet**. Dieses Mal werden wir jedoch mehrere Vergleiche durchführen, um die Beziehung zwischen allen Gruppenmittelwerten zu analysieren.

Laden Sie den [students](#) Datensatz erneut (Sie können die Datei [students.csv](#) [hier](#) herunterladen).

```
# Lese Datei students.csv als Dataframe ein
students = pd.read_csv("../data/students.csv")
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Aus dem Datensatz der Studierenden ziehen wir eine Zufallsstichprobe von 275 Absolventen und reduzieren den Datensatz auf die beiden interessierenden Variablen, die kategoriale Variable `major` und die numerische Variable `salary`. Für eine bessere Lesbarkeit in der folgenden Analyse ersetzen wir die Namen der Studienfächer durch entsprechende Abkürzungen.

```
n = 275
data = students.loc[students["graduated"] == 1, ["major", "salary"]].sample(
    n, random_state=300
)

lookup = {
    "Biology": "Bio",
    "Political Science": "PoS",
    "Economics and Finance": "EcFi",
    "Environmental Sciences": "EnS",
    "Mathematics and Statistics": "MaSt",
    "Social Sciences": "SoS",
}
data["major"] = data["major"].apply(lambda x: lookup[x])

data.head(10)
```

	major	salary
3825	MaSt	57849.34
794	MaSt	56558.16
2126	PoS	46945.17
1291	Bio	40093.46
5912	EnS	35106.96
1870	EnS	33216.12
1852	EnS	30606.20
1483	PoS	37240.11
5618	MaSt	50913.04
2298	EcFi	50365.10

Des Weiteren führen wir einen einfaktoriellen ANOVA-Hypothesentest in Python durch, indem wir die Funktion `f_oneway()` anwenden.

```
dat = [
    data.loc[data["major"] == major, "salary"].values
    for major in data["major"].unique()
]
statistics, pvalue = f_oneway(*dat)

print(f"V Wert der F-Statistik: {statistics}")
print(f"p-Wert: {pvalue}")
```

V Wert der F-Statistik: 41.14261117043315
 p-Wert: 2.2839018706608163e-31

Bonferroni-Korrektur

Die Bonferroni-Korrektur kompensiert die erhöhte Wahrscheinlichkeit, dass eine Nullhypothese aufgrund von Mehrfachvergleichen fälschlicherweise abgelehnt wird (Fehler vom Typ I), indem das Signifikanzniveau α in folgender Form angepasst wird

$$\alpha = \frac{\alpha}{m},$$

wobei m der Anzahl der Vergleiche entspricht, die gegeben ist durch

$$m = \frac{k(k-1)}{2},$$

wobei k den Ebenen des Faktors entspricht, der die kategoriale Klassifikationsvariable ist.

```
def bonferroni(alpha, k):  
    """Computes the boferroni correction"""  
    m = k * (k - 1) / 2  
    bonf = alpha / m  
    return bonf
```

Der paarweiser t -Test in Python: Bonferroni-Korrektur

Wie im vorigen Abschnitt erwähnt, ist eine einfaktorielle Varianzanalyse die Verallgemeinerung des 2-Stichproben t -Tests auf mehr als zwei Grundgesamtheiten. Die Python-Funktion zur Durchführung von Mehrfachvergleichen ist `ttest_ind()`. Die Funktion `ttest_ind()` vergleicht die jeweiligen Gehaltsverteilungen (`salary`) nach Studienfächer gruppiert (`major`) miteinander.

Zunächst führen wir einen paarweisen t -Test ohne Anpassung durch, wodurch sich die Wahrscheinlichkeit erhöht, dass die Nullhypothese falsch zurückgewiesen wird.

```
from itertools import combinations  
  
major = list(combinations(data.major.unique(), 2))  
major
```

```
[('MaSt', 'PoS'),
 ('MaSt', 'Bio'),
 ('MaSt', 'EnS'),
 ('MaSt', 'EcFi'),
 ('MaSt', 'SoS'),
 ('PoS', 'Bio'),
 ('PoS', 'EnS'),
 ('PoS', 'EcFi'),
 ('PoS', 'SoS'),
 ('Bio', 'EnS'),
 ('Bio', 'EcFi'),
 ('Bio', 'SoS'),
 ('EnS', 'EcFi'),
 ('EnS', 'SoS'),
 ('EcFi', 'SoS')]
```

```
alpha = 0.05

for major1, major2 in major:
    _, p_value = ttest_ind(
        data.loc[data["major"] == major1, "salary"],
        data.loc[data["major"] == major2, "salary"],
    )
    print(f"{major1} vs. {major2}")
    print(f"p-value: {p_value}")
    print(f"Reject H0 (p-value <= {alpha}): {p_value <= alpha}\n")
```


MaSt vs. PoS
p-value: 5.285766661005193e-11
Reject H0 (p-value <= 0.05): True

MaSt vs. Bio
p-value: 0.23955822802713908
Reject H0 (p-value <= 0.05): False

MaSt vs. EnS
p-value: 2.873918020813552e-10
Reject H0 (p-value <= 0.05): True

MaSt vs. EcFi
p-value: 0.6139625494797094
Reject H0 (p-value <= 0.05): False

MaSt vs. SoS
p-value: 1.0518797210598569e-10
Reject H0 (p-value <= 0.05): True

PoS vs. Bio
p-value: 4.8406921036979525e-15
Reject H0 (p-value <= 0.05): True

PoS vs. EnS
p-value: 0.12016488500152636
Reject H0 (p-value <= 0.05): False

PoS vs. EcFi
p-value: 2.6490333956019106e-12
Reject H0 (p-value <= 0.05): True

PoS vs. SoS
p-value: 0.23097756920298104
Reject H0 (p-value <= 0.05): False

Bio vs. EnS
p-value: 1.7902286073326123e-14
Reject H0 (p-value <= 0.05): True

Bio vs. EcFi
p-value: 0.5127687451099829
Reject H0 (p-value <= 0.05): False

Bio vs. SoS
p-value: 6.474696427603517e-14
Reject H0 (p-value <= 0.05): True

EnS vs. EcFi
p-value: 8.151863146368181e-12
Reject H0 (p-value <= 0.05): True

EnS vs. SoS

```
p-value: 0.0055758247214054745
Reject H0 (p-value <= 0.05): True
```

```
EcFi vs. SoS
p-value: 7.145673135006227e-12
Reject H0 (p-value <= 0.05): True
```

Der paarweise *t*-Test zeigt, dass bei einem Signifikanzniveau von 5% die Mittelwerte für 5 Kombinationen **nicht** statistisch signifikant unterschiedlich sind. Diese Kombinationen sind Bio-EcFi, Bio-MaSt, EcFi-MaSt, EnS-PoS, PoS-SoS mit *p*-Werten von 0, 5128, 0, 2396, 0, 61396, 0, 1202, 0, 635057 bzw. 0, 230977. Für die restlichen 10 Kombinationen verwerfen wir H_0 ; d. h. für 10 Kombinationen sind die Mittelwerte bei einem Signifikanzniveau von 5% unterschiedlich!

Zweitens führen wir einen **paarweisen *t*-Test mit der Bonferroni-Anpassung** durch.

```
bonf = bonferroni(alpha=0.05, k=data.major.nunique())
bonf
```

```
0.0033333333333333335
```

```
for major1, major2 in major:
    _, p_value = ttest_ind(
        data.loc[data["major"] == major1, "salary"],
        data.loc[data["major"] == major2, "salary"],
    )
    print(f"{major1} vs. {major2}")
    print(f"p-value: {p_value}")
    print(f"Reject H0 (p-value <= {np.round(bonf, 5)}): {p_value <= bonf}\n")
```


MaSt vs. PoS
p-value: 5.285766661005193e-11
Reject H₀ (p-value <= 0.00333): True

MaSt vs. Bio
p-value: 0.23955822802713908
Reject H₀ (p-value <= 0.00333): False

MaSt vs. EnS
p-value: 2.873918020813552e-10
Reject H₀ (p-value <= 0.00333): True

MaSt vs. EcFi
p-value: 0.6139625494797094
Reject H₀ (p-value <= 0.00333): False

MaSt vs. SoS
p-value: 1.0518797210598569e-10
Reject H₀ (p-value <= 0.00333): True

PoS vs. Bio
p-value: 4.8406921036979525e-15
Reject H₀ (p-value <= 0.00333): True

PoS vs. EnS
p-value: 0.12016488500152636
Reject H₀ (p-value <= 0.00333): False

PoS vs. EcFi
p-value: 2.6490333956019106e-12
Reject H₀ (p-value <= 0.00333): True

PoS vs. SoS
p-value: 0.23097756920298104
Reject H₀ (p-value <= 0.00333): False

Bio vs. EnS
p-value: 1.7902286073326123e-14
Reject H₀ (p-value <= 0.00333): True

Bio vs. EcFi
p-value: 0.5127687451099829
Reject H₀ (p-value <= 0.00333): False

Bio vs. SoS
p-value: 6.474696427603517e-14
Reject H₀ (p-value <= 0.00333): True

EnS vs. EcFi
p-value: 8.151863146368181e-12
Reject H₀ (p-value <= 0.00333): True

EnS vs. SoS

```
p-value: 0.0055758247214054745
Reject H0 (p-value <= 0.00333): False
```

```
EcFi vs. SoS
p-value: 7.145673135006227e-12
Reject H0 (p-value <= 0.00333): True
```

Der paarweise t -Test mit der Bonferroni-Anpassung zeigt, dass bei einem Signifikanzniveau von 5% die Mittelwerte für 6 Kombinationen nicht statistisch signifikant unterschiedlich sind. Bei diesen Kombinationen handelt es sich um Bio-EcFi, Bio-MaSt, EcFi-MaSt, EnS-PoS, PoS-SoS, EnS-SoS mit p -Werten von 0,5128, 0,2396, 0,61397, 0,1202, 0,230978 bzw. 0,00558. (beim Bonferroni-Verfahren wird α durch die Anzahl der Tests geteilt bzw. der p -Wert mit dieser Anzahl multipliziert und auf 1 gekürzt, wenn das Ergebnis über 1 liegt und somit keine Wahrscheinlichkeit darstellt). Für die verbleibenden 9 Kombinationen haben wir H_0 abgelehnt, d. h. für 9 Kombinationen sind die Mittelwerte bei einem Signifikanzniveau von $\approx 0,0067\%$ unterschiedlich!

Tukey-Mehrfach-Vergleichsmethode

Der [Tukey-Test](#), auch bekannt als **Tukey's HSD-Test (honest significant difference)**, basiert auf der [studentized range-Verteilung](#), die manchmal auch als q -Verteilung bezeichnet wird. Die q -Verteilung ist eine rechtsschiefe Wahrscheinlichkeitsdichtekurve mit zwei Parametern, κ und ν , die ihre Form beschreiben. Diese Parameter sind gegeben durch

$$\kappa = k$$

und

$$\nu = n - k,$$

wobei n die Gesamtzahl der Beobachtungen ist und k die Anzahl der Gruppen/Klassen.

Der Tukey-Test vergleicht die Mittelwerte jeder Gruppe mit den Mittelwerten jeder anderen Gruppe. Er liefert das Konfidenzintervall für jedes

$$\mu_i - \mu_j.$$

Wenn das Konfidenzintervall für einen paarweisen Vergleich 0 einschließt, wird H_0 nicht verworfen, es wird nicht angenommen, dass sie signifikant unterschiedlich sind. Alle anderen Paare, für die das

Konfidenzintervall nicht 0 einschließt, sind signifikant unterschiedlich, H_0 wird also verworfen.

Tukey's Test in Python

In Python werden Tukey's HSD Tests durch die `pairwise_tukeyhsd()` Funktion berechnet. Die `pairwise_tukeyhsd()`-Funktion erwartet als Eingabe `endog`, die zu vergleichende Größe und `groups` die Aufteilung der Gruppen. Um die Breite der Konfidenzintervalle festzulegen, geben wir der Funktion das Konfidenzniveau mit dem Argument `alpha` an.

```
# Tukey's test ausführen
tukey = pairwise_tukeyhsd(endog=data["salary"], groups=data["major"], alpha=0.05)
tukey.summary()
```

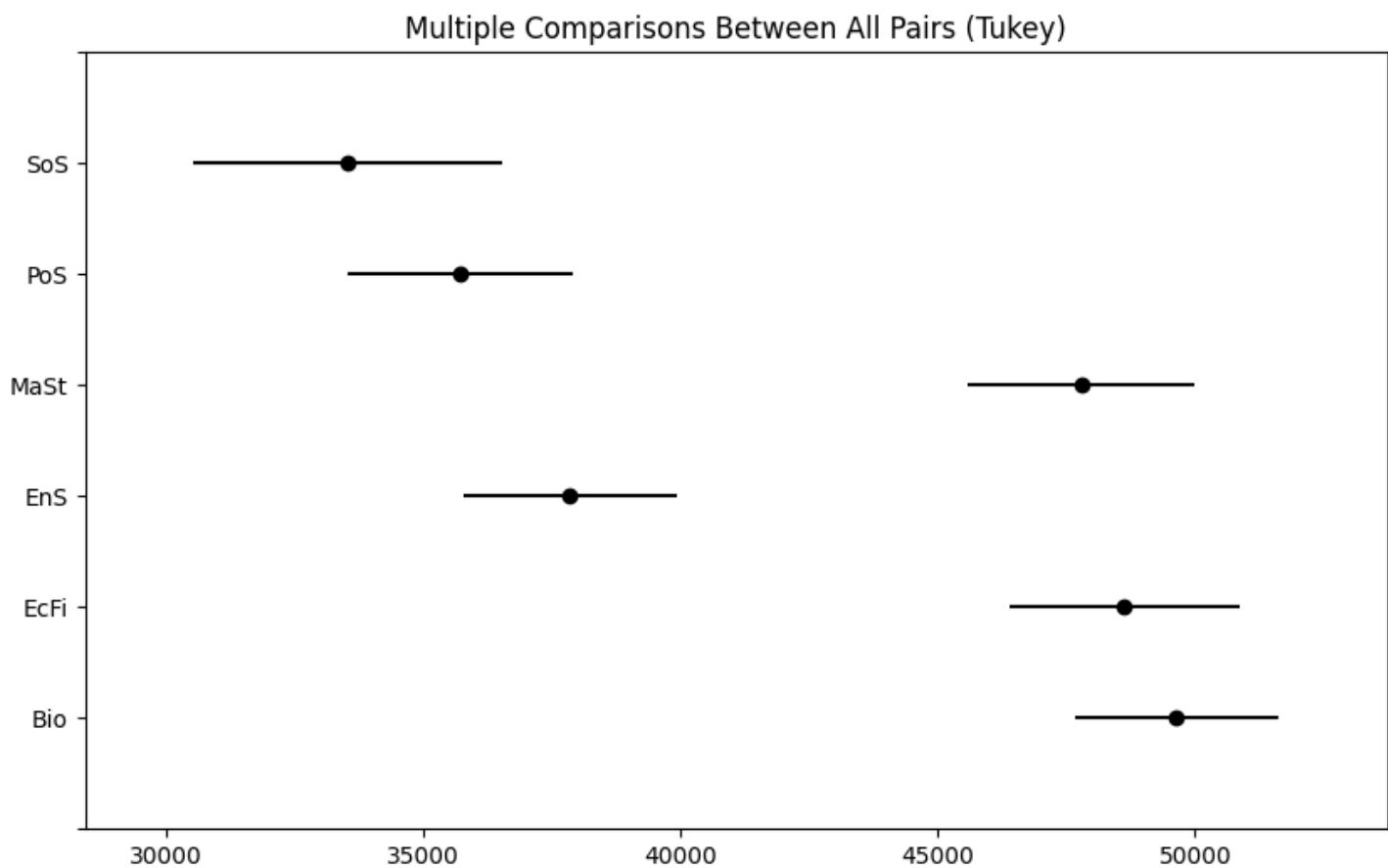
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Bio	EcFi	-1012.3074	0.983	-5228.5838	3203.969	False
Bio	EnS	-11777.6591	0.0	-15810.771	-7744.5472	True
Bio	MaSt	-1846.3676	0.8039	-6036.7579	2344.0227	False
Bio	PoS	-13913.1332	0.0	-18078.5881	-9747.6782	True
Bio	SoS	-16091.4847	0.0	-21100.6999	-11082.2695	True
EcFi	EnS	-10765.3517	0.0	-15067.6405	-6463.0629	True
EcFi	MaSt	-834.0602	0.9946	-5284.1241	3616.0036	False
EcFi	PoS	-12900.8257	0.0	-17327.4172	-8474.2343	True
EcFi	SoS	-15079.1773	0.0	-20307.5551	-9850.7995	True
EnS	MaSt	9931.2915	0.0	5654.3682	14208.2148	True
EnS	PoS	-2135.474	0.7016	-6387.9695	2117.0214	False
EnS	SoS	-4313.8256	0.1475	-9395.65	767.9988	False
MaSt	PoS	-12066.7655	0.0	-16468.7078	-7664.8232	True
MaSt	SoS	-14245.1171	0.0	-19452.6422	-9037.5919	True
PoS	SoS	-2178.3516	0.834	-7365.8329	3009.1297	False

Bitte beachten Sie, dass wir im Fall ohne p -Wert-Anpassung die Nullhypothese für 10 Kombinationen abgelehnt haben. Beim paarweisen t -Test mit der Tukey-Anpassung wurde die Nullhypothese für 9 Kombinationen verworfen.

Die Tabelle zeigt die Differenz zwischen den einzelnen Paaren, die 95%-Konfidenzintervalle und den p -Wert der paarweisen Vergleiche. Schauen Sie sich die Tabelle genau an, und Sie werden sehen, dass für alle 6 Vergleiche, bei denen das Konfidenzintervall 0 einschließt, der p -Wert höher ist als das Signifikanzniveau α . Wenn $p > \alpha$ ist, verwerfen wir H_0 nicht, d. h. es gibt keinen statistisch signifikanten Unterschied zwischen den Mittelwerten dieser beiden Gruppen. Für alle Paare, bei denen $p \leq \alpha$ ist, verwerfen wir dagegen H_0 und stellen fest, dass ein statistisch signifikanter Unterschied zwischen den Mittelwerten dieser Paare besteht. Die `tukey.plot_simultaneous()`-Funktion bietet eine nette Plot-Funktion, die die Konfidenzintervalle für jedes Paar visualisiert.

```
_ = tukey.plot_simultaneous()
```



Übungsaufgaben

Einfaktorielle ANOVA Grundbegriffe

Beschreiben Sie die Grundlagen der einfaktoriellen ANOVA.

1. Wonach kann mittels ANOVA getestet werden? Nennen Sie mögliche praktische Anwendungen in der Wissenschaft.
 2. Welche Bedingungen müssen für das Durchführen einer ANOVA gegeben sein?
 3. Was sind die Einschränkungen einer ANOVA?
-

Lösungen

Einfaktorielle ANOVA

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import seed
from numpy.random import normal

import pylab
from scipy.stats import t

from scipy.stats import uniform
from scipy import stats
from scipy.stats import f_oneway
```

1. Führen Sie jeweils eine schrittweise einfaktorielle Varianzanalyse für die folgenden Daten durch:

- `sample_dat1`
- `sample_dat2`
- `sample_dat3`
- `sample_dat4`
- `sample_dat5`
- `sample_dat6`

\begin{array}{|l|} \hline \text{Schritt 1} & \text{Geben Sie die Nullhypothese } H_0 \text{ und alternative Hypothese } H_A \text{ an.} \\ \hline \text{Schritt 2} & \text{Legen Sie das Signifikanzniveau, } \alpha \text{ fest.} \\ \hline \text{Schritt 3} & \text{Berechnen Sie den Wert der Teststatistik.} \\ \hline \text{Schritt 4} & \text{Bestimmen Sie den p-Wert.} \\ \hline \text{Schritt 5} & \text{Wenn } p \leq \alpha \text{, } H_0 \text{ ablehnen; ansonsten } H_0 \text{ nicht ablehnen.} \\ \hline \text{Schritt 6} & \text{Interpretieren Sie das Ergebnis des Hypothesentests.} \\ \hline \end{array}

- Benutzen Sie für Schritte 3 und 4 die Funktion `f_oneway()` die Sie mit `from scipy.stats import f_oneway` importieren können

2. Interpretieren Sie das Ergebnis

```
from scipy.stats import norm, t

rs = 1
sample_dat1 = norm.rvs(loc=0, scale=1, size=25, random_state=rs)
sample_dat2 = norm.rvs(loc=0.01, scale=1.1, size=30, random_state=rs)
sample_dat3 = norm.rvs(loc=-0.01, scale=1.1, size=28, random_state=rs)
sample_dat4 = t.rvs(df=33, loc=0.8, scale=0.8, size=34, random_state=rs)
sample_dat5 = t.rvs(df=26, loc=0.5, scale=1.22, size=27, random_state=rs)
sample_dat6 = norm.rvs(loc=0, scale=1, size=25, random_state=rs)
```

Frage 1 ...

Lösungen

► Show code cell content

Bonferroni Korrektur

1. Führen Sie einen post-hoc Mehrfachhypothesentests mit den Datensätzen `sample_dat1`, `sample_dat2` und `sample_dat3` durch um zu bestimmen welcher Datensatz sich von den anderen unterscheidet. Berechnen Sie hierfür die Bonferroni Korrektur bei einem $\alpha = 0.05$. Die Bonferroni Korrektur ergibt sich zu:

$$\alpha = \frac{\alpha}{m},$$

$$m = \frac{k(k - 1)}{2},$$

$$\alpha = \frac{0,05}{3}$$

2. Interpretieren Sie das Ergebnis

```
from scipy.stats import norm, t

rs = 1
sample_dat1 = t.rvs(df=33, loc=0.8, scale=0.8, size=34, random_state=rs)
sample_dat2 = t.rvs(df=26, loc=0.5, scale=1.22, size=27, random_state=rs)
sample_dat3 = norm.rvs(loc=0, scale=1, size=25, random_state=rs)
```

```
# Frage 1 ...
```

Lösungen

▶ Show code cell content

Lernziele

Lineare Regression

- Die Grundbegriffe der linearen Regression werden erläutert und anhand des probabilistischen Modells der linearen Einfachregression, $y = a + bx + \epsilon$ veranschaulicht
- Am Beispiel des Anscombe Datensatzes werden Methoden der Regressionsdiagnostik zur Bewertung und Auswahl von Modellen wie : Bestimmtheitsmaß , Methode der gewöhnlichen kleinsten Quadrate und RMSE besprochen
- Weiters werden Methoden zum Umgang mit Heelpunkten und Ausreißern behandelt
- Schließlich wird die Polynomiale Regression als Erweiterung zur linearen Regression vorgestellt
- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Lineare Regression , Lineare Einfachregression , Parameterschätzung - Methode der gewöhnlichen kleinsten Quadrate , Modelldiagnose , Regressionsdiagnostik , Anscombe-Quartett , Bestimmtheitsmaß , Heelpunkte und Ausreißer , Multiple lineare Regression - Polynomiale Regression

Lineare Regression

```
import math

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import linregress
import statsmodels.api as sm
```

Die [Regressionsanalyse](#) ist ein statistisches Verfahren zur Schätzung der Beziehungen zwischen zwei oder mehr Variablen. Die Beziehung wird als $y \sim x$ oder $y = f(x)$ modelliert. Beide Modellbeschreibungen besagen, dass die Variable y eine Funktion von x ist. Daher wird die Variable y als **Antwortvariable** oder **abhängige Variable** bezeichnet, während die Variable x als **Prädikatorvariable** oder **unabhängige Variable** bezeichnet wird.

Einfache lineare Regression

In diesem Abschnitt wird eine spezielle Art der Regression behandelt, die als [einfache lineare Regression](#) bezeichnet wird. In diesem speziellen Fall der Regressionsanalyse wird die Beziehung zwischen der Antwortvariablen y und der Prädikatorvariablen x in Form einer **linearen** Gleichung dargestellt

$$y = a + bx,$$

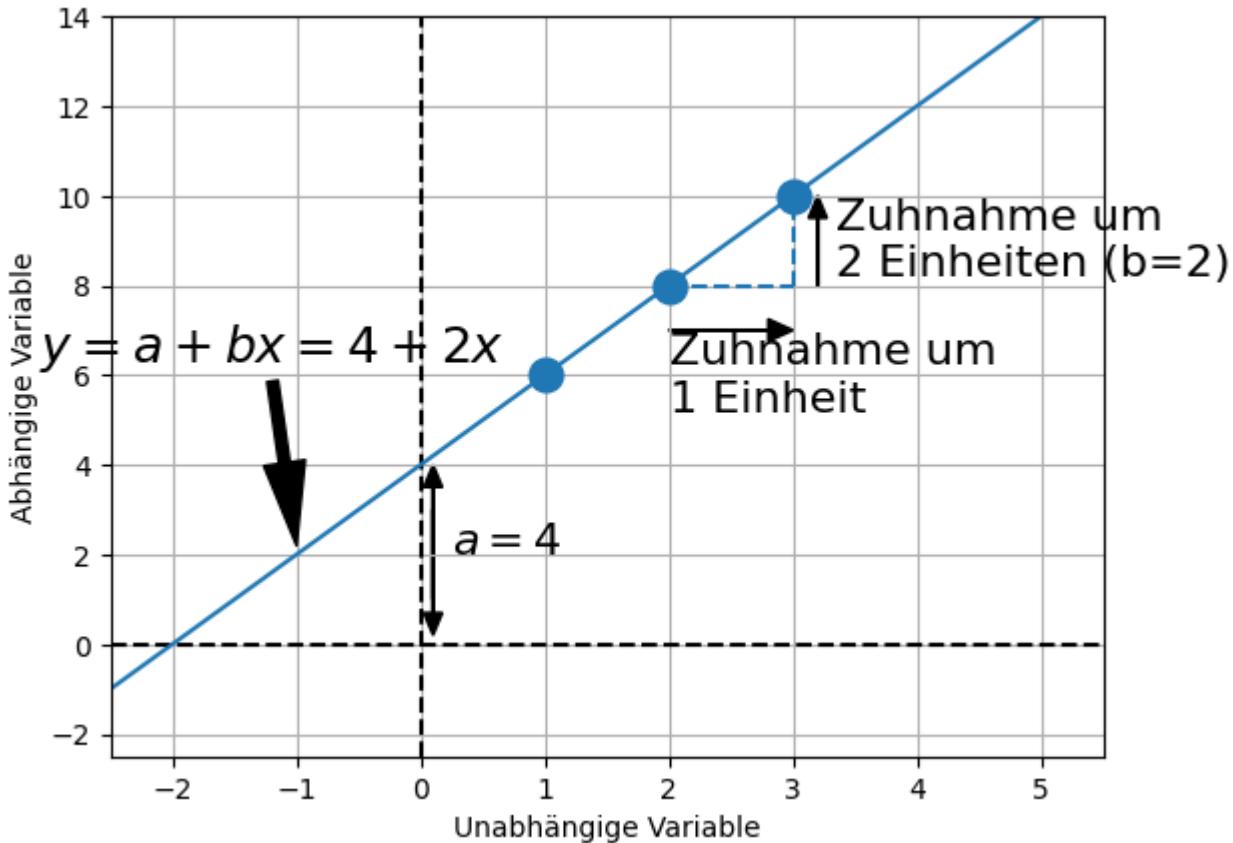
wobei a und b Konstanten sind. Die Zahl a wird als **Achsenabschnitt** bezeichnet und definiert den Schnittpunkt der Regressionslinie mit der y -Achse ($x = 0$). Die Zahl b wird als **Regressionskoeffizient** bezeichnet. Er ist ein Maß für die Steigung der **Regressionsgeraden**. So gibt b an, um wie viel sich der y -Wert ändert, wenn sich der x -Wert um 1 Einheit erhöht. Das Adjektiv **einfach** bezieht sich auf die Tatsache, dass die Ergebnisvariable mit einem einzigen Vorhersagewert verknüpft ist. Das Modell wird als **deterministisches Modell** betrachtet, da es eine genaue Beziehung zwischen x und y herstellt.

Lassen Sie uns ein einfaches Beispiel betrachten. Gegeben ist eine Grundgesamtheit von $n = 3$ Punkten mit kartesischen Koordinaten (x_i, y_i) von $(1, 6)$, $(2, 8)$ und $(3, 10)$. Diese Punkte liegen auf

einer Geraden und können daher durch ein lineares Gleichungsmodell in der Form $y = a + bx$ beschrieben werden, wobei der Schnittpunkt $a = 4$ und $b = 2$ ist.

► Show code cell source

```
Text(0, 0.5, 'Abhängige Variable')
```



In vielen Fällen ist die Beziehung zwischen zwei Variablen x und y jedoch nicht exakt. Das liegt daran, dass die Antwortvariable y von anderen unbekannten und/oder zufälligen Prozessen beeinflusst wird, die von der Prädiktorvariable x nicht vollständig erfasst werden. In einem solchen Fall liegen die Datenpunkte nicht auf einer Geraden. Die Daten können jedoch immer noch einer zugrunde liegenden linearen Beziehung folgen. Um diese Unbekannten zu berücksichtigen, wird der linearen Modellgleichung ein **Zufallsfehlerterm**, bezeichnet mit ϵ , hinzugefügt, was im Gegensatz zum oben beschriebenen deterministischen Modell zu einem **probabilistischen Modell** führt.

$$y = a + bx + \epsilon$$

wobei angenommen wird, dass der Fehlerterm ϵ_i aus unabhängigen normalverteilten Werten besteht, $\epsilon_i \sim N(0, \sigma^2)$.

Bei der linearen Regressionsmodellierung werden folgende Annahmen über das Modell getroffen (s.439,).

- Der zufällige Fehlerterm ϵ hat für jedes x einen Mittelwert gleich Null.
- Die mit verschiedenen Beobachtungen verbundenen Fehler sind unabhängig.
- Für jedes gegebene x ist die Verteilung der Fehler normal.
- Die Verteilung der Fehler für jedes x hat die gleiche (konstante) Standardabweichung, die mit σ_ϵ bezeichnet wird.

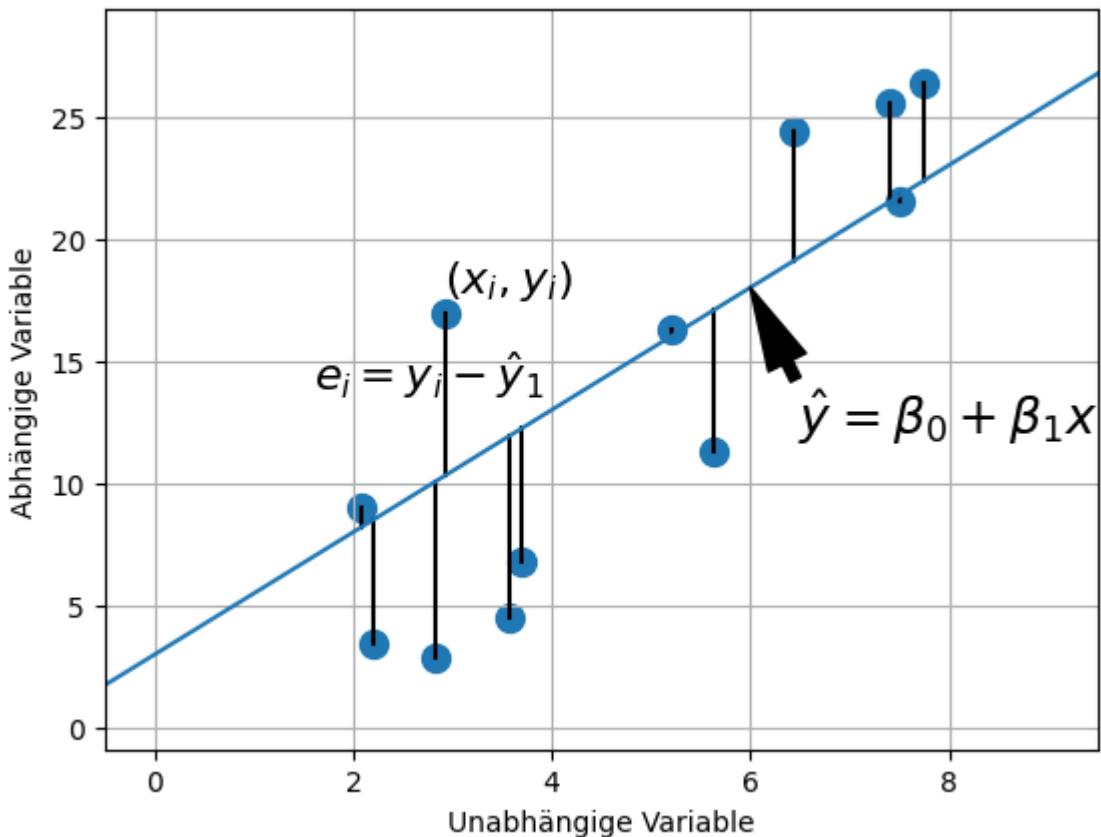
Betrachten wir ein weiteres Beispiel. Diesmal nehmen wir eine Zufallsstichprobe mit dem Stichprobenumfang $n = 8$ aus einer Grundgesamtheit. Um zu betonen, dass die Werte des Abschnitts und der Steigung aus Stichprobendaten berechnet werden, werden a und b mit β_0 bzw. β_1 bezeichnet. Außerdem wird der Fehlerterm ϵ als e bezeichnet. β_0 , β_1 und e sind also Schätzungen auf der Grundlage von Stichprobendaten für die Grundgesamtheitsparameter a , b und ϵ .

$$\hat{y} = \beta_0 + \beta_1 x + e,$$

wobei \hat{y} der **geschätzte oder vorhergesagte Wert** von y für einen bestimmten Wert von x ist.

► Show code cell source

Text(0, 0.5, 'Abhängige Variable')



Der Fehler e_i für jedes einzelne Wertepaar (x_i, y_i) , auch **Residuum** genannt, wird aus der Differenz zwischen dem beobachteten Wert y_i und dem durch \hat{y}_i gegebenen vorhergesagten Wert errechnet.

$$e_i = y_i - \hat{y}_i$$

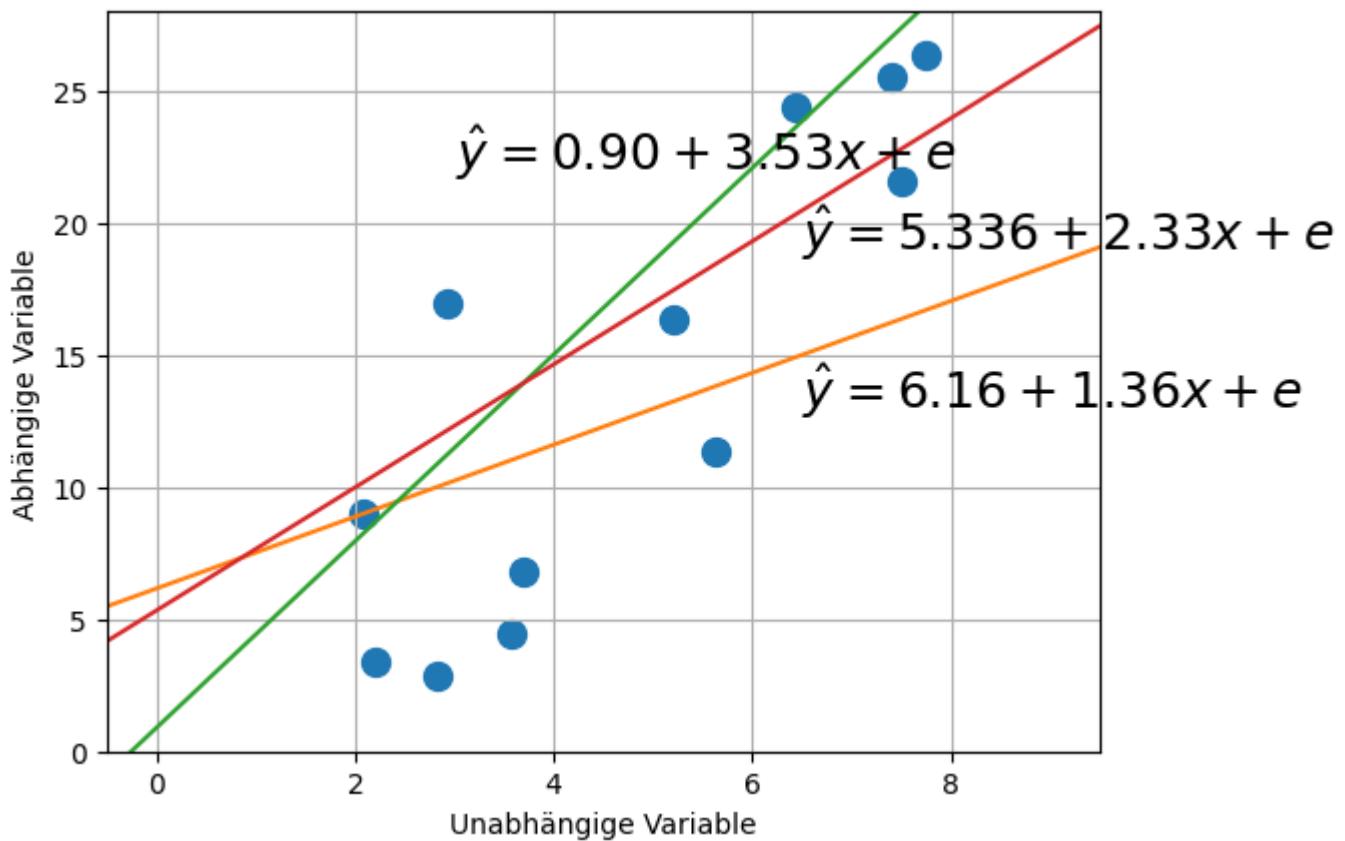
Je nach Datenlage ist e_i eine negative Zahl, wenn y_i unterhalb der Regressionslinie liegt, oder eine positive Zahl, wenn y_i oberhalb der Regressionslinie liegt.

Parameterschätzung - Methode der gewöhnlichen kleinsten Quadrate (OLS)

Da wir nun die Beschränkungen des deterministischen Modells gelockert und einen Fehlerterm ϵ eingeführt haben, stoßen wir auf ein weiteres Problem. Es gibt unendlich viele Regressionsgeraden, die die Spezifikationen des probabilistischen Modells erfüllen.

► Show code cell source

Text(0, 0.5, 'Abhängige Variable')



Offensichtlich brauchen wir eine Strategie, um diejenige Regressionsgerade auszuwählen, die das beste Modell zur Beschreibung der Daten darstellt. In diesem Abschnitt befassen wir uns mit einer der gängigsten Methoden zur Erfüllung dieser Aufgabe, der so genannten Methode der [gewöhnlichen kleinsten Quadrate](#) (englisch ordinary least squares, kurz: *OLS*).

Wie im vorigen Abschnitt erwähnt, wird für jedes bestimmte Wertepaar (x_1, y_1) wird der Fehler e_i durch $y_1 - \hat{y}$ berechnet. Um die beste Anpassungsgerade für die gegebenen Daten zu erhalten, wird die **Summe der Fehlerquadrate**, bezeichnet als *SSE*, minimiert.

$$\min SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y - \hat{y})^2$$

Für das einfache lineare Modell gibt es eine analytische Lösung für β_1

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n ((x_i - \bar{x})(y_i - \bar{y}))}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{cov(x, y)}{var(x)},$$

und β_0 :

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Die *OLS* liefert die Maximum-Likelihood-Schätzung für $\hat{\beta}$, wenn die Parameter die gleiche Varianz haben und unkorreliert sind und die Residuen ϵ unkorreliert sind und einer Gaußschen Verteilung folgen ([Homoskedastizität](#)).

Einfache lineare Regression - Ein Beispiel

Um einige praktische Erfahrungen zu sammeln, wenden wir die einfache lineare Regression in einer Übung an. Dazu laden wir den `students` Datensatz. Sie können die Datei `students.csv` [hier](#) herunterladen. Importieren Sie den Datensatz und geben Sie ihm einen passenden Namen.

```
# Lese Datei students.csv als Dataframe ein
students = pd.read_csv("../data/students.csv")
```

Der `students` Datensatz besteht aus 8239 Zeilen, von denen jede einen bestimmten Studenten repräsentiert, und 16 Spalten, von denen jede einer Variable/einem Merkmal entspricht, das sich auf diesen bestimmten Studenten bezieht. Diese selbsterklärenden Variablen sind: `stud_id`, `name`, `gender`, `age`, `height`, `weight`, `religion`, `nc_score`, `semester`, `major`, `minor`, `score1`, `score2`, `online_tutorial`, `graduated`, `salary`.

Um die **einfache lineare Regression** zu veranschaulichen, untersuchen wir die Beziehung zwischen zwei Variablen, `height` der Studenten als Prädiktorvariable und `weight` der Studierenden als Antwortvariable.

Vorbereitung der Daten

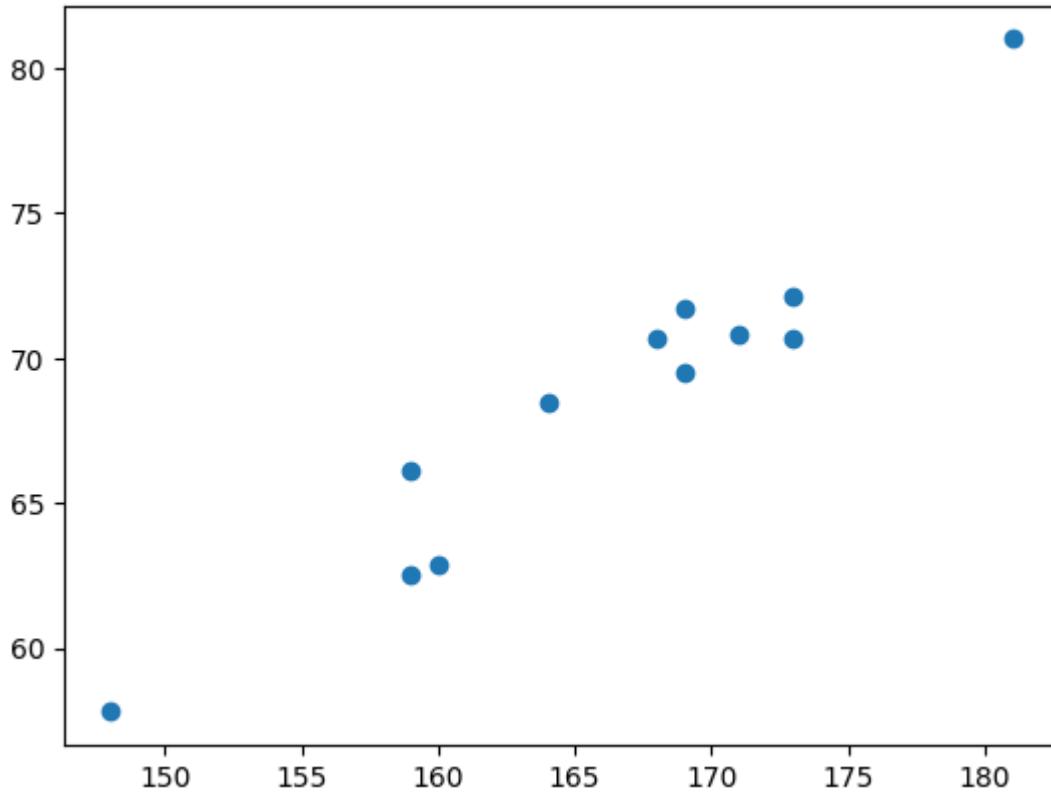
Zur Datenaufbereitung ziehen wir eine Zufallsstichprobe von 12 Studenten aus dem Datensatz und erstellen einen Datensatz mit den zwei Variablen von Interesse (`height` und `weight`). Außerdem

stellen wir die Daten in Form eines Streudiagramms dar, um die zugrunde liegende lineare Beziehung zwischen den beiden Variablen zu visualisieren.

```
n = 12
data = students[["weight", "height"]].sample(n, random_state=2).sort_index()

fig, ax = plt.subplots()
ax.scatter(data["height"], data["weight"])
```

```
<matplotlib.collections.PathCollection at 0x7cd036886080>
```



Die visuelle Inspektion bestätigt unsere Vermutung, dass die Beziehung zwischen der Größe und der Gewichtsvariable ungefähr linear ist. Mit anderen Worten: Mit zunehmender Größe neigt der einzelne Studierende dazu, ein höheres Gewicht zu haben.

Schätzung der Parameter

Lösen für β_0 und β_1 analytisch in Python

Wie im vorherigen Abschnitt gezeigt, können die Parameter β_0 und β_1 eines einfachen linearen Modells analytisch berechnet werden. Erinnern Sie sich an die Gleichung für ein lineares Modell aus Stichprobendaten

$$\hat{y} = \beta_0 + \beta_1 x + e,$$

für β_1

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n ((x_i - \bar{x})(y_i - \bar{y}))}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{cov(x, y)}{var(x)},$$

und für β_0

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Zum besseren Verständnis verwenden wir Python, um die einzelnen Terme zu berechnen

```
# Berechne b1
x = data["height"]
y = data["weight"]
x_bar = np.mean(x)
y_bar = np.mean(y)

b1 = np.sum((x - x_bar) * (y - y_bar)) / np.sum((x - x_bar) ** 2)
b1
```

```
np.float64(0.6507615230460924)
```

Die Steigung des Regressionsmodells beträgt ungefähr 0,65. Zur Überprüfung der Korrektheit berechnen wir das Verhältnis der Kovarianz von x und y mit der Funktion `cov()` und die Varianz von x mit der Funktion `var()` und vergleichen es mit dem Ergebnis von oben.

```
np.cov(x, y, ddof=0)[0][1] / np.var(x)
```

```
np.float64(0.6507615230460924)
```

Eine perfekte Übereinstimmung!

Weiter berechnen wir β_0 .

```
# Berechne b0
b0 = y_bar - b1 * x_bar
b0
```

```
np.float64(-39.443206412825674)
```

Der Achsenabschnitt β_0 des Regressionsmodells beträgt ungefähr $-39,44$.

Wir können also das Regressionsmodell aufschreiben

$$\text{Gewicht} = -39,44 + 0,65 \times \text{Höhe}$$

Auf der Grundlage dieser Gleichung können wir nun das Gewicht eines Studenten anhand seiner Größe bestimmen. Lassen Sie uns zum Spaß das Gewicht von Studierenden mit einer Größe von 156, 178 und 192 cm vorhersagen.

$$\text{Gewicht}_{156} = -39,44 + 0,65 \times 156 \approx 62 \text{ kg}$$

$$\text{Gewicht}_{178} = -39,44 + 0,65 \times 178 \approx 76 \text{ kg}$$

$$\text{Gewicht}_{192} = -39,44 + 0,65 \times 192 \approx 85 \text{ kg}$$

Verwenden Sie die Funktion `linregress()` bzw `OLS()` in Python, um β_0 zu berechnen und β_1

Zum einen können wir die Funktion `linregress()` nutzen um β_0 und β_1 zu berechnen.

```
gradient, intercept, r_value, p_value, stderr = linregress(  
    data["height"], data["weight"]  
)  
print(f"beta_0 = {intercept}")  
print(f"beta_1 = {gradient}")
```

```
beta_0 = -39.44320641282566  
beta_1 = 0.6507615230460922
```

Eine andere Möglichkeit ist es das Paket `statsmodels` und die Funktion `OLS()` zu nutzen, die zusätzlich zur Berechnung von β_0, β_1 viele weitere Möglichkeiten bietet

```
x = sm.add_constant(x)  
model = sm.OLS(y, x).fit()  
  
model.params
```

```
const      -39.443206  
height      0.650762  
dtype: float64
```

Die Ausgabe über die Methode `params` liefert den Achsenabschnitt und den Regressionskoeffizienten. Weiters kann die Methode `summary()` nützlich sein, wenn Sie auf andere Eigenschaft des Modellobjekts zugreifen möchten.

```
model.summary()
```

```
/home/imarevic/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10
    return hypotest_fun_in(*args, **kwds)
```

OLS Regression Results

Dep. Variable: weight **R-squared:** 0.921
Model: OLS **Adj. R-squared:** 0.913
Method: Least Squares **F-statistic:** 115.9
Date: Tue, 22 Oct 2024 **Prob (F-statistic):** 8.05e-07

```

Time: 21:36:24 Log-Likelihood: -22.602
No. Observations: 12 AIC: 49.20
Df Residuals: 10 BIC: 50.17
Df Model: 1
Covariance Type: nonrobust

            coef  std err      t  P>|t|  [0.025  0.975]
const    -39.4432  10.057  -3.922  0.003  -61.851  -17.036
height     0.6508   0.060  10.766  0.000    0.516    0.785

Omnibus: 2.161 Durbin-Watson: 1.778
Prob(Omnibus): 0.339 Jarque-Bera (JB): 0.923
Skew: 0.110 Prob(JB): 0.630
Kurtosis: 1.659 Cond. No. 3.33e+03

```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 3.33e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Die Methode `conf_int()` gibt das Konfidenzintervall für die Modellkoeffizienten für ein bestimmtes Konfidenzniveau zurück (Standardeinstellung entspricht $\alpha = 0, 05$).

```
model.conf_int(alpha=0.05)
```

	0	1
const	-61.850735	-17.035678
height	0.516081	0.785442

Eine weitere nützliche Extraktormethode ist die Funktion `resid()`.

```
model.resid
```

```
338      0.815271
2123     -1.037014
2387      1.218317
2726     -1.527876
2766     -1.035491
3593      1.164509
4151      2.655371
4432      2.072124
5099     -1.038537
7098     -2.438537
7135      0.930501
8169     -1.778637
dtype: float64
```

Wir können sofort überprüfen, ob die Summe der Residuen ($\sum e$) nahe bei Null liegt.

```
sum(model.resid)
```

```
7.815970093361102e-14
```

Cool, nahe an Null!

Um die erhaltene Regressionslinie zu zeichnen, verwenden wir die Funktion $y = \beta_0 + \beta_1 x$, die Linien auf der Grundlage des Achsenabschnitts (β_0) und der Steigung (β_1) zeichnet. Wir verwenden den Syntax `plot([x_1, x_2], [y_1, y_2])` um die Regressionsgerade zu plotten.

Eine weitere besonders interessante Extraktormethode ist `predict()`. Wenn sie nicht spezifiziert ist, gibt die Methode `predict()` \hat{y}_i für jedes einzelne x_i zurück; ähnlich wie die Methode `fittedvalues`. Man kann jedoch auch neue Daten angeben und die Funktion `predict()` wird \hat{y}_i für jedes gegebene x_i vorhersagen. Beachten Sie, dass die neuen Daten ein Data-Frame-Objekt oder eine Liste sein müssen.

```
print(model.predict(x))
```

```
338      69.884729
2123     71.837014
2387     67.281683
2726     64.027876
2766     70.535491
3593     70.535491
4151     78.344629
4432     64.027876
5099     73.138537
7098     73.138537
7135     56.869499
8169     64.678637
dtype: float64
```

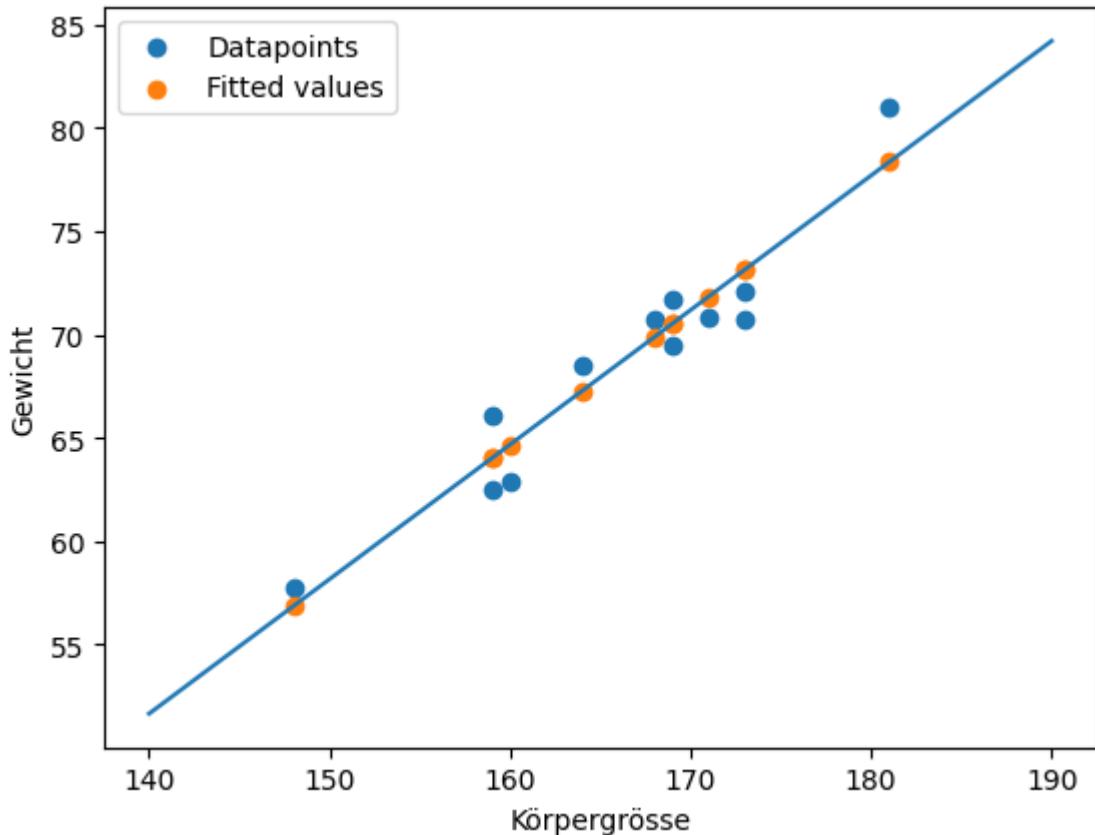
```
model.fittedvalues
```

```
338      69.884729
2123     71.837014
2387     67.281683
2726     64.027876
2766     70.535491
3593     70.535491
4151     78.344629
4432     64.027876
5099     73.138537
7098     73.138537
7135     56.869499
8169     64.678637
dtype: float64
```

```
fig, ax = plt.subplots()
ax.scatter(data["height"], data["weight"], label="Datapoints")
ax.scatter(data["height"], model.fittedvalues, label="Fitted values")
# create regression line
x_axis = np.linspace(140, 190, 100)
_x_axis = sm.add_constant(x_axis)
ax.plot(x_axis, model.predict(_x_axis))

ax.set_ylabel("Gewicht")
ax.set_xlabel("Körpergrösse")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7cd03682b0a0>
```



Darüber hinaus bietet Python einen einfachen Ansatz zur Erstellung von Fehlerbereichen um die angepasste Regressionslinie. Es gibt zwei Arten von Bändern, die als *schmale* und *breite* Bänder bezeichnet werden. Die schmalen Bänder, die so genannten [Konfidenzbänder](#), spiegeln die Unsicherheit über die Linie selbst wider. Die Bänder sind schmal, wenn es viele Beobachtungen gibt, und spiegeln eine gut bestimmte Linie wider. Die breiten Banden, die so genannten Prognosebänder (s.27), beinhalten die Unsicherheit über zukünftige Beobachtungen. Diese Bänder erfassen die Mehrheit der beobachteten Punkte und fallen nicht zu einer Linie zusammen, wenn die Anzahl der Beobachtungen zunimmt.

Um diese Unsicherheitsbänder zu berechnen, wenden wir die Methode `get_prediction()` an und fügen die Methode `summary_frame()` hinzu, um den Vektor der vorhergesagten Werte zu erhalten, der mit Grenzwerten ergänzt wird.

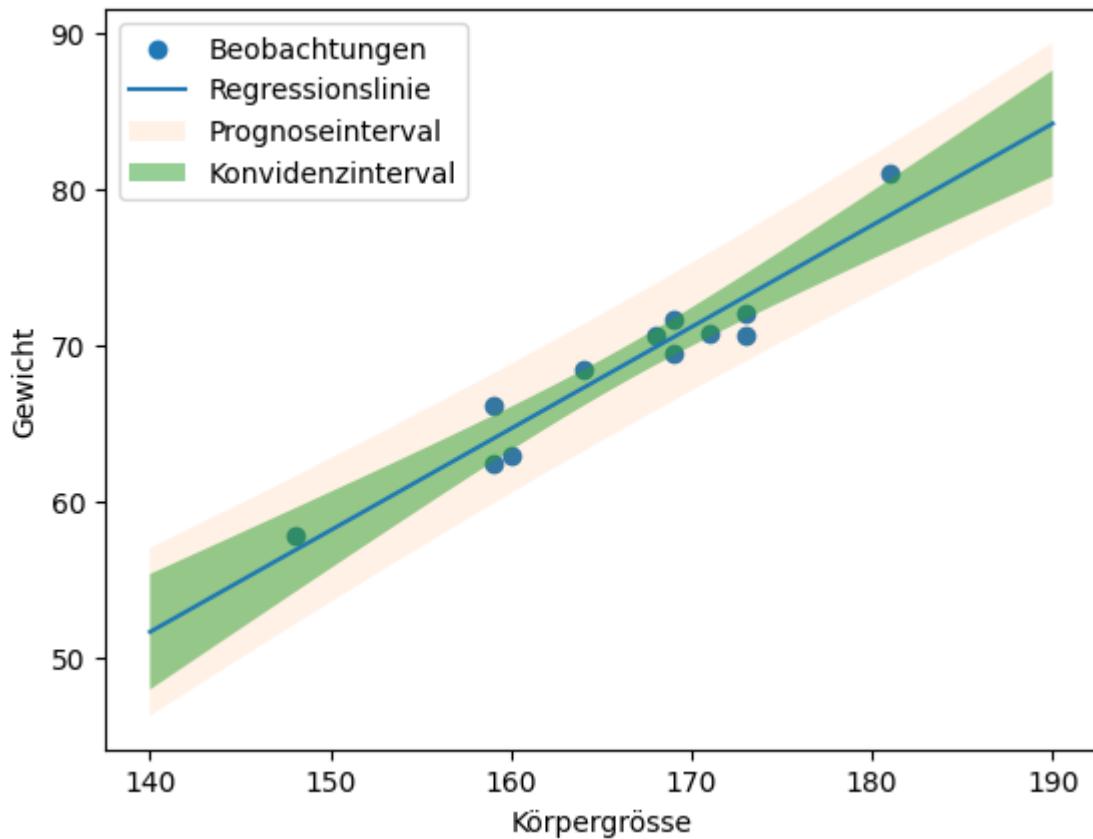
```
pred = np.linspace(140, 190)
pred = sm.add_constant(pred)
model_predictions = model.get_prediction(pred).summary_frame()

fig, ax = plt.subplots()
ax.scatter(data["height"], data["weight"], label="Beobachtungen")

# create regression line
x_axis = np.linspace(140, 190)

ax.plot(x_axis, model_predictions["mean"], label="Regressionslinie")
ax.fill_between(
    x_axis,
    model_predictions["obs_ci_lower"],
    model_predictions["obs_ci_upper"],
    alpha=0.1,
    label="Prognoseintervall",
)
ax.fill_between(
    x_axis,
    model_predictions["mean_ci_lower"],
    model_predictions["mean_ci_upper"],
    alpha=0.5,
    label="Konfidenzintervall",
)
ax.set_ylabel("Gewicht")
ax.set_xlabel("Körpergrösse")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7cd0363784c0>
```



Es gibt viele weitere Attribute und Methoden eines `OLS`-Objekts auf die zugegriffen werden kann. Die Funktion `dir()` gibt einen Überblick über die Struktur des Modellobjekts.

```
dir(model)
```



```
['HC0_se',
 'HC1_se',
 'HC2_se',
 'HC3_se',
 '_HCCM',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abat_diagonal',
 '_cache',
 '_data_attr',
 '_data_in_cache',
 '_get_robustcov_results',
 '_get_wald_nonlinear',
 '_is_nested',
 '_transform_predict_exog',
 '_use_t',
 '_wexog_singular_values',
 'aic',
 'bic',
 'bse',
 'centered_tss',
 'compare_f_test',
 'compare_lm_test',
 'compare_lr_test',
 'condition_number',
 'conf_int',
 'conf_int_el',
 'cov_HC0',
 'cov_HC1',
```

```
'cov_HC2',
'cov_HC3',
'cov_kwds',
'cov_params',
'cov_type',
'df_model',
'df_resid',
'diagn',
'eigenvals',
'el_test',
'ess',
'f_pvalue',
'f_test',
'fittedvalues',
'fvalue',
'get_influence',
'get_prediction',
'get_robustcov_results',
'info_criteria',
'initialize',
'k_constant',
'llf',
'load',
'model',
'mse_model',
'mse_resid',
'mse_total',
'nobs',
'normalized_cov_params',
'outlier_test',
'params',
'predict',
'pvalues',
'remove_data',
'resid',
'resid_pearson',
'rsquared',
'rsquared_adj',
'save',
'scale',
'ssr',
'summary',
'summary2',
't_test',
't_test_pairwise',
'tvalues',
'uncentered_tss',
'use_t',
'wald_test',
'wald_test_terms',
'wresid']
```

Modelldiagnose

Regressionsdiagnostik beinhaltet eine Reihe von Verfahren, die zur Bewertung der numerischen Ergebnisse einer Regressionsanalyse angewandt werden. Die Verfahren umfassen Methoden der grafischen und quantitativen Analyse oder formale statistische Hypothesentests. In diesem Abschnitt konzentrieren wir uns auf die beiden wichtigsten Methoden, die grafische und die quantitative Analyse. Statistische Hypothesentests für Regressionsprobleme finden Sie im Abschnitt über *Hypothesentests*.

Bestimmtheitsmaß

Der Bestimmtheitsmaß, auch als R^2 bezeichnet, ist der Anteil der Variation der beobachteten Werte, der durch die Regressionsgleichung erklärt wird. Mit anderen Worten: R^2 ist ein statistisches Maß dafür, wie gut die Regressionsgerade die realen Datenpunkte annähert; es ist also ein Maß für die Anpassungsfähigkeit des Modells.

Die Gesamtvariation der Antwortvariablen y basiert auf der Abweichung jedes beobachteten Wertes y_i vom Mittelwert \bar{y} . Diese Größe wird als **Gesamtsumme der Quadrate**, SST , bezeichnet und ist gegeben durch

$$SST = \sum (y_i - \bar{y})^2.$$

Diese Gesamtsumme der Quadrate (SST) kann in zwei Teile zerlegt werden: die durch die Regressionslinie erklärte Abweichung $\hat{y}_i - \bar{y}$ und die verbleibende unerklärte Abweichung $y_i - \hat{y}_i$. Folglich wird der Anteil der Variation, der durch die Regression erklärt wird, als **Summe der Quadrate aufgrund der Regression**, SSR , bezeichnet und ist gegeben durch

$$SSR = \sum (\hat{y}_i - \bar{y})^2.$$

Das Verhältnis zwischen der Summe der Quadrate aufgrund der Regression (SSR) und der Gesamtsumme der Quadrate (SST) wird als Bestimmtheitsmaß bezeichnet und mit R^2 angegeben.

$$R^2 = \frac{SSR}{SST}$$

R^2 liegt zwischen 0 und 1. Ein Wert nahe 0 deutet darauf hin, dass die Regressionsgleichung nicht in der Lage ist, die Daten zu erklären. Ein R^2 von 1 zeigt an, dass die Regressionsgerade perfekt zu den Daten passt.

Der Vollständigkeit halber wird die Variation in den beobachteten Werten der Reaktionsvariablen, die nicht durch die Regression erklärt wird, als **Summe der quadrierten Fehler der Vorhersage (SSE)** bezeichnet und ist gegeben durch

$$SSE = \sum (y_i - \hat{y}_i)^2.$$

Erinnern Sie sich, dass die SSE -Größe minimiert wird, um die beste Regressionslinie zur Beschreibung der Daten zu erhalten, auch bekannt als die [Methode der gewöhnlichen kleinsten Quadrate \(OLS\)](#).

Die Methode `summary()`

Eine grundlegendes Mittel für die Regressionsdiagnose in Python ist die Methode `summary()`. Die Funktion `OLS()` gibt ein Modellobjekt zurück. Dieses `OLS()`-Objekt enthält die Modelleigenschaften, die durch Anwendung der Methode `summary()` untersucht werden können.

Zu Demonstrationszwecken wird dasselbe Modell wie im vorherigen Abschnitt verwendet.

```
# Lese Datei students.csv als Dataframe ein; Indexspalte wird übersprungen
students = pd.read_csv("../data/students.csv", index_col=0)

n = 12

data = students[["height", "weight"]].sample(n, random_state=2)
x = data["height"]
y = data["weight"]

x = sm.add_constant(x)
model = sm.OLS(y, x).fit()
predictions = model.predict(x) # make the predictions by the model

# Lese Modellwerte aus
model.summary()
```

```
/home/imarevic/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10
return hypotest_fun_in(*args, **kwds)
```

OLS Regression Results

Dep. Variable: weight **R-squared:** 0.921
Model: OLS **Adj. R-squared:** 0.913
Method: Least Squares **F-statistic:** 115.9
Date: Tue, 22 Oct 2024 **Prob (F-statistic):** 8.05e-07
Time: 21:36:25 **Log-Likelihood:** -22.602
No. Observations: 12 **AIC:** 49.20
Df Residuals: 10 **BIC:** 50.17
Df Model: 1
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	-39.4432	10.057	-3.922	0.003	-61.851	-17.036
height	0.6508	0.060	10.766	0.000	0.516	0.785
Omnibus:	2.161	Durbin-Watson:		1.442		
Prob(Omnibus):	0.339	Jarque-Bera (JB):		0.923		
Skew:	0.110	Prob(JB):		0.630		
Kurtosis:	1.659	Cond. No.		3.33e+03		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 3.33e+03. This might indicate that there are strong multicollinearity or other numerical problems.

- Die Ausgabe der Methode `summary()` beginnt mit einer Wiederholung der abhängigen Variable und der angewandten Methode (in diesem Fall *OLS*).
- Die nächste Zeile zeigt R^2 , den quadrierten [Pearson Korrelationskoeffizienten](#), auch bekannt als [Bestimmtheitsmaß](#) und das angepasste R^2 , ein statistisches Maß, das für die Merkmalsauswahl

bei der Regressionsanalyse mit mehreren Prädiktoren (multiple Regression) verwendet werden kann.

- Die darauf folgenden Zeilen zeigen die F -Statistik, die Anzahl der Beobachtungen (Datenpunkte) und Freiheitsgrade.
- In den nächsten Zeilen werden der Regressionskoeffizient (unter `height`) und der Achsenabschnitt (unter `const`) angegeben, außerdem für jeden von ihnen der Standardfehler, die t -Werte und die p -Werte.

Diagnostische Plots

Es ist wichtig zu wissen, dass Sie eine lineare Regressionsanalyse mit dem Softwarepaket Python oder einer anderen Statistiksoftware durchführen können, die eine Reihe von Zahlen, einschließlich eines p -Werts, ergibt, so dass Sie sofort feststellen können, ob die Ergebnisse signifikant waren (oder nicht). Sind wir mit der Angabe der Signifikanz der Ergebnisse fertig?

Nehmen wir einen sehr berühmten Datensatz, das so genannte [Anscombe-Quartett](#). Das Anscombe-Quartett besteht aus vier Datensätzen und hat die folgende Form:

x1	y1	x2	y2	x3	y3	x4	y4
10	8,04	10	9,14	10	7,46	8	6,58
8	6,95	8	8,14	8	6,77	8	5,76
13	7,58	13	8,74	13	12,74	8	7,71
9	8,81	9	8,77	9	7,11	8	8,84
11	8,33	11	9,26	11	7,81	8	8,47
14	9,96	14	8,1	14	8,84	8	7,04
6	7,24	6	6,13	6	6,08	8	5,25
4	4,26	4	3,1	4	5,39	19	12,5
12	10,84	12	9,13	12	8,15	8	5,56

7	4,82	7	7,26	7	6,42	8	7,91
5	5,68	5	4,74	5	5,73	8	6,89

Das Anscombe-Quartett wird oft verwendet um die Unterschiede zwischen grafischer und statistischer Auswertung hervorzuheben. Wir geben den Datensatz in Python ein.

```

x1 = [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]
y1 = [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]

x2 = [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]
y2 = [9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74]

x3 = [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]
y3 = [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73]

x4 = [8, 8, 8, 8, 8, 8, 19, 8, 8, 8]
y4 = [6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89]

X = [x1, x2, x3, x4]
Y = [y1, y2, y3, y4]

```

Nun berechnen wir einige deskriptive statistische Maße für jedes der vier (x, y) -Paare. Zunächst berechnen wir den Mittelwert für jedes einzelne x und y im Datensatz.

```

for e, (x, y) in enumerate(zip(X, Y)):
    print(f"mean x{e+1}: {round(np.mean(x), 3)} | mean y{e+1}: {round(np.mean(y), 3)}")

```

```

mean x1: 9.0 | mean y1: 7.501
mean x2: 9.0 | mean y2: 7.501
mean x3: 9.0 | mean y3: 7.5
mean x4: 9.0 | mean y4: 7.501

```

Die Werte stimmen entweder perfekt überein oder liegen sehr nahe beieinander!!

Jetzt berechnen wir die Varianz jedes (x, y) Paars.

```

for e, (x, y) in enumerate(zip(X, Y)):
    print(
        f"variance x{e+1}: {round(np.var(x), 3)} | variance y{e+1}: {round(np.var(y), 3)}")

```

```
variance x1: 10.0 | variance y1: 3.752
variance x2: 10.0 | variance y2: 3.752
variance x3: 10.0 | variance y3: 3.748
variance x4: 10.0 | variance y4: 3.748
```

Sie sind zwar nicht exakt gleich, aber definitiv sehr nahe beieinander. Schließlich erstellen wir mit der Funktion `linregress()` ein lineares Modell für jede Teilmenge und berechnen die Koeffizienten des Modells und R^2 , das Bestimmtheitsmaß.

```
for e, (x, y) in enumerate(zip(X, Y)):
    slope, intercept, r_value, p_value, std_err = linregress(x, y)
    print(
        f"{e+1}: {round(slope,2)}x + {round(intercept,2)}, p-value: {round(p_value,2)}")
```

```
1: 0.5x + 3.0, p-value: 0.00217, Pearson correlation coefficient: 0.816, Standard error: 0.00217
2: 0.5x + 3.0, p-value: 0.00218, Pearson correlation coefficient: 0.816, Standard error: 0.00218
3: 0.5x + 3.0, p-value: 0.00218, Pearson correlation coefficient: 0.816, Standard error: 0.00218
4: 0.5x + 3.0, p-value: 0.00216, Pearson correlation coefficient: 0.817, Standard error: 0.00216
```

Erstaunlich! Sie sind fast identisch! Und jetzt R^2 :

```
for e, (x, y) in enumerate(zip(X, Y)):
    _, _, r_value, _, _ = linregress(x, y)
    print(
        f"r2: {round(r_value**2,3)}",
    )
```

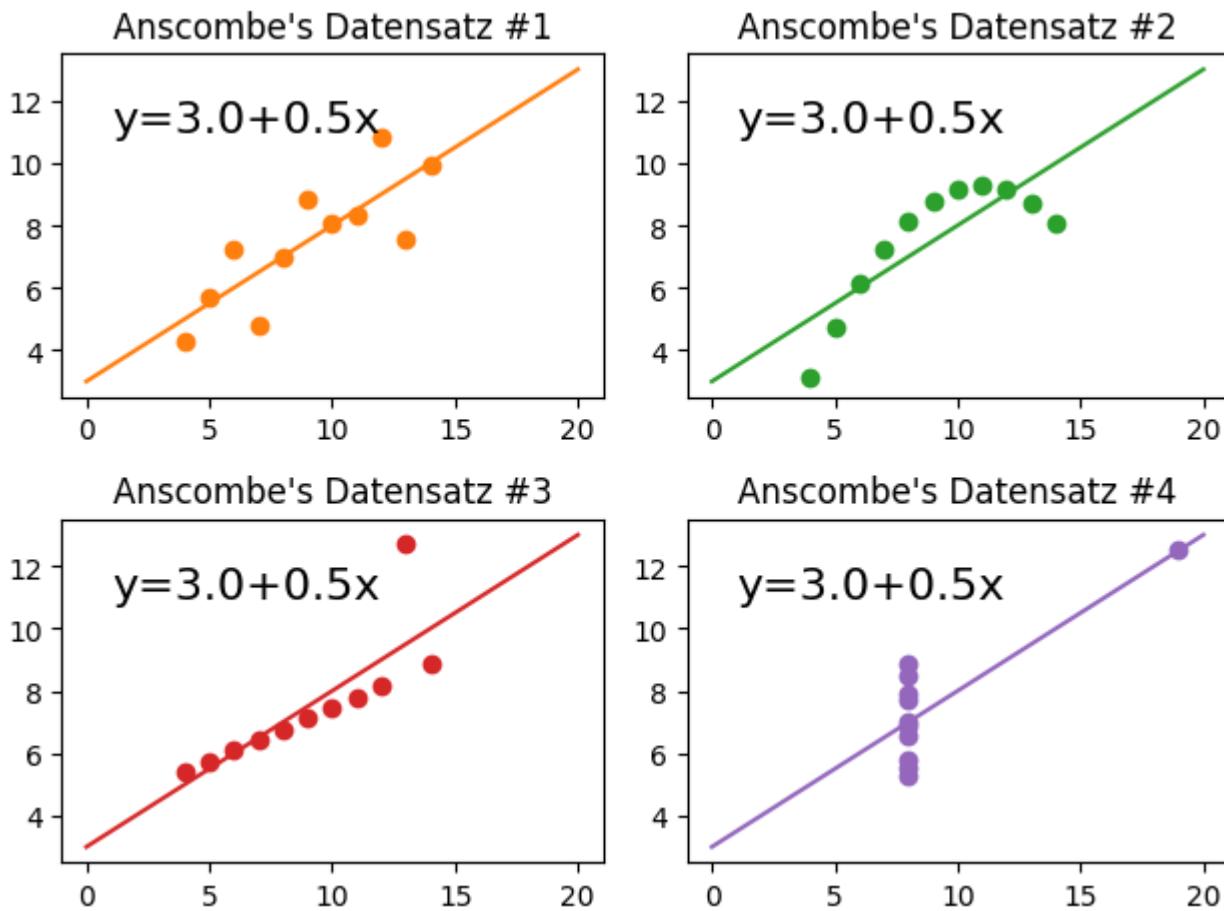
```
r2: 0.667
r2: 0.666
r2: 0.666
r2: 0.667
```

Wow, was für eine Analyse! Wir haben eine Menge verschiedener statistischer Methoden auf die vier Datensätze angewandt, und ehrlich gesagt, sie sehen einander sehr ähnlich.

Sind wir jetzt mit unserer Analyse fertig? Nein, noch nicht! Egal was wir tun, wir sollten immer überprüfen, ob das Modell für die Daten gut funktioniert. Eine einfache Möglichkeit, dies zu tun, ist die

Visualisierung der Daten. Lassen Sie uns den Anscombe-Datensatz einschließlich der Regressionslinie grafisch darstellen.

```
fig, ax = plt.subplots(ncols=2, nrows=2)
ax = np.ravel(ax)
xaxis = np.linspace(0, 20, 100)
for e, (x, y) in enumerate(zip(X, Y)):
    slope, intercept, _, _, _ = linregress(x, y)
    regline = intercept + slope * xaxis
    ax[e].plot(xaxis, regline, color=f"C{e+1}")
    ax[e].scatter(x, y, color=f"C{e+1}")
    ax[e].set_title(f"Anscombe's Datensatz #{e+1}")
    ax[e].text(s=f"y={round(intercept,2)}+{round(slope,3)}x", x=1, y=11, size=16)
fig.tight_layout()
```



Was für eine Überraschung! Das wichtigste Ergebnis der Übung ist die Erkenntnis, dass wir auf viele verschiedene Arten prüfen müssen, ob ein Modell für Daten gut funktioniert. Wir achten auf Regressionsergebnisse wie Steigungskoeffizienten, p -Werte oder R^2 , die uns sagen, wie gut ein Modell die gegebenen Daten darstellt. Das ist jedoch nicht die ganze Geschichte. Wir müssen auch visuelle Diagnosen anwenden. Die visuelle Inspektion hilft bei der Bewertung, ob die Annahmen der linearen Regression erfüllt sind, oder bei der Ermittlung von [Ausreißern](#) und/oder statistisch

bedeutsame Beobachtungen und so genannten [Hebelwerte](#), die das numerische Ergebnis der Regressionsanalyse beeinflussen.

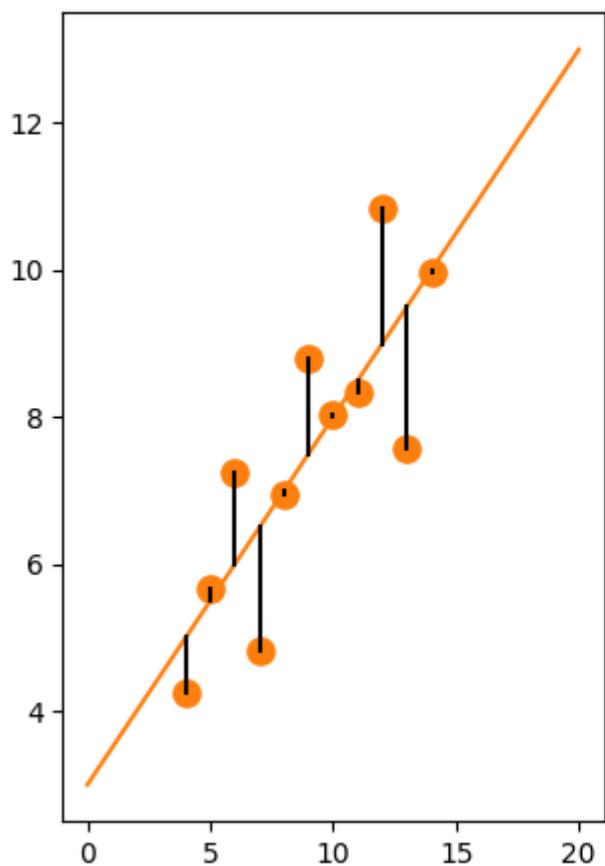
Analyse der Residuen

Ein [Residuum](#) eines beobachteten Wertes ist die Differenz zwischen dem beobachteten Wert und dem geschätzten Wert ($y_i - \hat{y}_i$). Es handelt sich um die Residuen, die nach der Anpassung eines Modells an die Daten übrig bleiben. Die **Summe der quadrierten Vorhersagefehler (SSE)**, auch bekannt als die **Summe der quadrierten Residuen** oder die **Fehlersumme der Quadrate**, ist ein Indikator dafür, wie gut ein Modell die Daten darstellt.

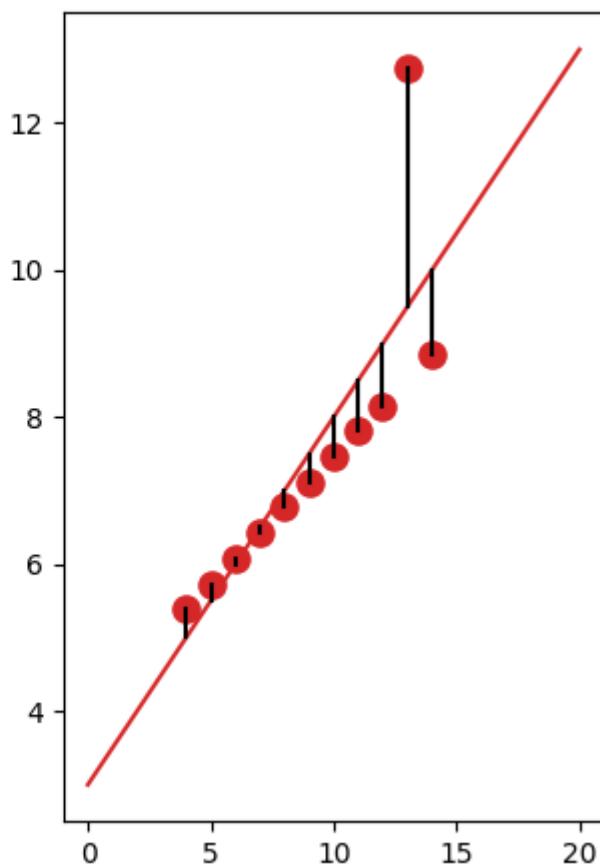
Wenn die absoluten Residuen, definiert für die Beobachtung x_i als $e_i = y_i - \hat{y}_i$ definiert sind, ungewöhnlich groß sind, kann es sein, dass die Beobachtung aus einer anderen Grundgesamtheit stammt oder dass bei der Durchführung oder Aufzeichnung der Beobachtung ein Fehler aufgetreten ist.

```
fig, ax = plt.subplots(ncols=2)
xaxis = np.linspace(0, 20, 100)
colors = ["C1", "C3"]
for e, (X, Y) in enumerate(zip((X[0], X[2]), (Y[0], Y[2]))):
    slope, intercept, _, _, _ = linregress(X, Y)
    regline = intercept + slope * xaxis
    ax[e].plot(xaxis, regline, color=colors[e])
    ax[e].scatter(X, Y, s=90, color=colors[e])
    for x, y in zip(X, Y):
        yhat = intercept + slope * x
        ax[e].plot((x, x), (y, yhat), color="k")
    ax[e].set_title(f"Anscombe's Datensatz #{colors[e][1:]}")
fig.tight_layout()
```

Anscombe's Datensatz #1



Anscombe's Datensatz #3



Die beiden obigen Diagramme zeigen, dass ein Datenpunkt in Anscombes Datensatz Nr. 3 (rechtes Diagramm) ein ungewöhnlich großes Residuum aufweist. Ein solcher Datenpunkt erfordert besondere Aufmerksamkeit, da er die Regressionsanalyse beeinflusst. Es gibt keine allgemeingültige Regel, wie mit Ausreißern umzugehen ist, aber je nach den Fachkenntnissen des Forschers kann es Fälle geben, in denen man beschließt, einen solchen Ausreißer aus der Analyse auszuschließen.

Darüber hinaus können wir die Residuen analysieren, um zu prüfen, ob die Annahmen der linearen Regression erfüllt sind. Regressionsresiduen sollten annähernd normalverteilt sein, d. h. die Regression sollte die Struktur erklären, und was übrig bleibt, sollte nur Rauschen sein, das durch Messfehler oder viele kleine unkorrelierte Faktoren verursacht wird. Die Normalität der Residuen kann grafisch überprüft werden, indem man die Residuen gegen die Werte der Prädiktorvariablen aufträgt. In einem solchen **Residuen-Plot** sollten die Residuen zufällig um 0 streuen und die Variation um 0 sollte gleich sein.

Vor der Darstellung der Residuen ist es üblich, die Residuen zu standardisieren. Python bietet die Möglichkeit mit `get_influence()` auf die standardisierten Residuen zuzugreifen (`influence = model.get_influence()` und `standardized_residuals =`

`influence.resid_studentized_internal`) und alternativ kann man mit `stud_res = model.outlier_test()` die studentisierten Residuen (s.152) berechnen.

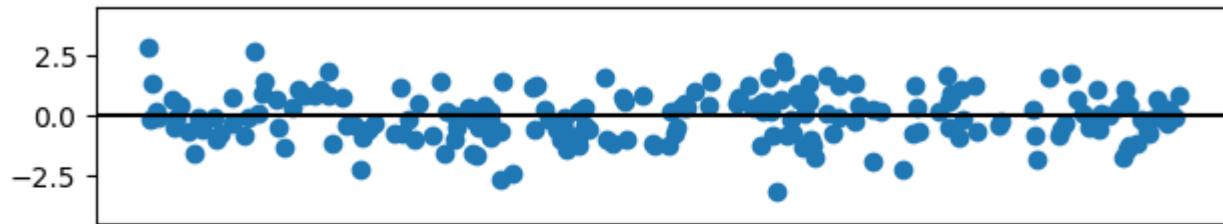
Wenn die Annahmen für Regressionsschlussfolgerungen erfüllt sind, sollten die folgenden zwei Bedingungen gelten (s.443):

- Eine Darstellung der Residuen (Residuenplot) gegen die Werte der Prädiktorvariablen sollte ungefähr in ein horizontales Band fallen, das um die x -Achse zentriert und symmetrisch ist.
- Eine Normalwahrscheinlichkeitsdarstellung der Residuen sollte in etwa linear sein.

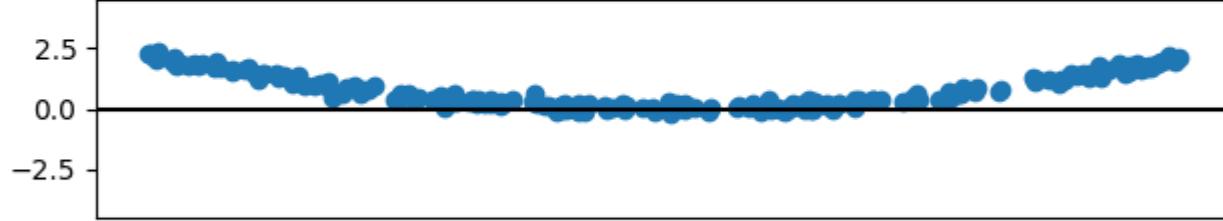
► Show code cell source

Residuen-Plots

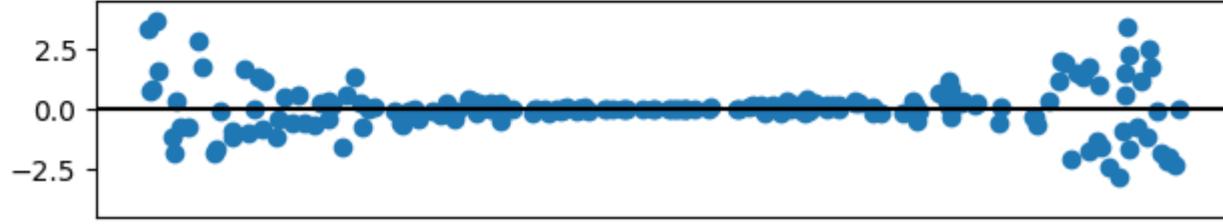
Keine Verletzung



Verletzung der Linearität



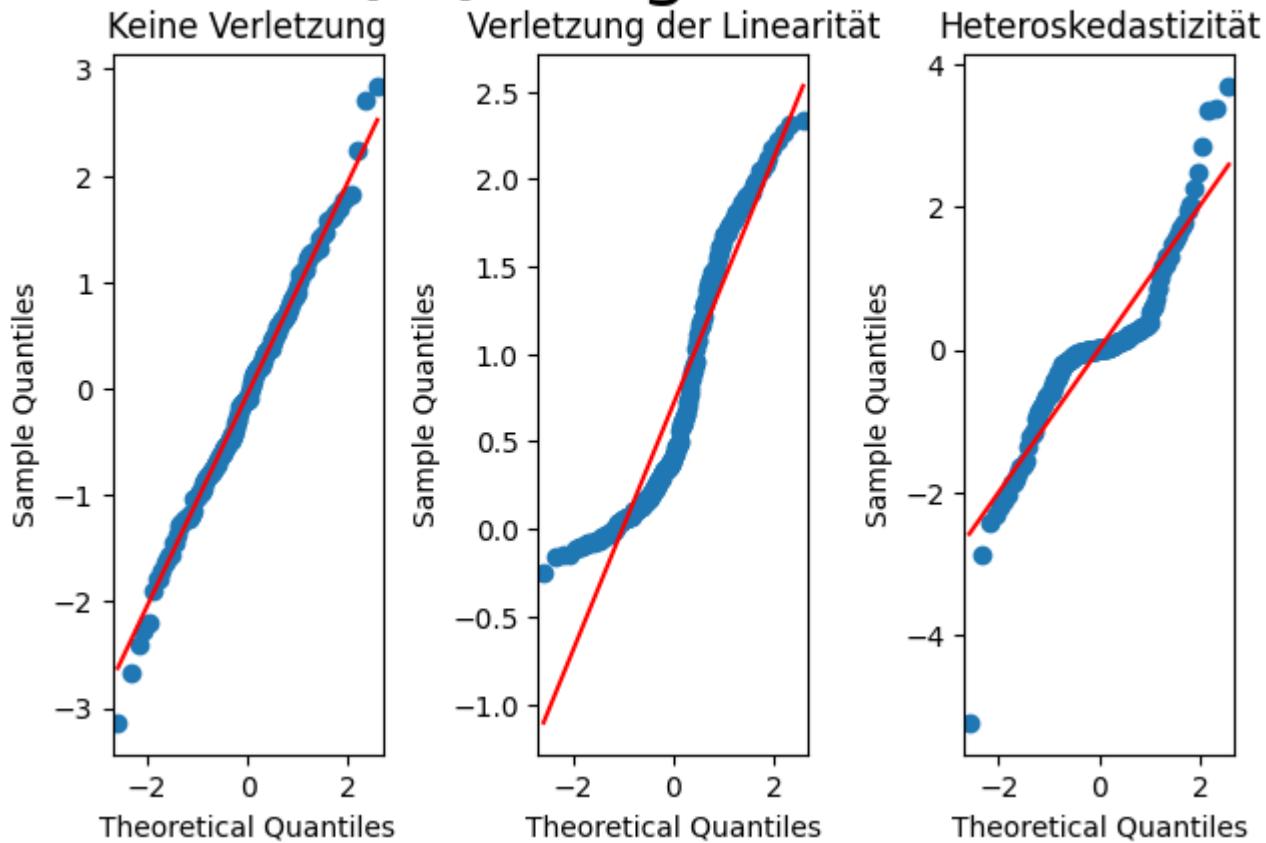
Verletzung der konstanten Standardabweichung (Heteroskedastizität)



Nur in der obersten Grafik sind die Residuen relativ gut um den Nullpunkt verteilt, während dies in den beiden unteren Grafiken nicht der Fall ist, was darauf hindeutet, dass die linearen Modellannahmen für dieses Modell nicht erfüllt sind.

► Show code cell source

Q-Q-Diagramm



Die Normalwahrscheinlichkeitsdiagramme, die oft als [Q-Q-Diagramme](#) bezeichnet werden, zeigen, dass nur im linken Diagramm die Datenpunkte in etwa auf eine gerade Linie fallen. Dies ist bei den anderen Diagrammen nicht der Fall, was darauf hindeutet, dass die Annahmen des linearen Modells nicht erfüllt sind.

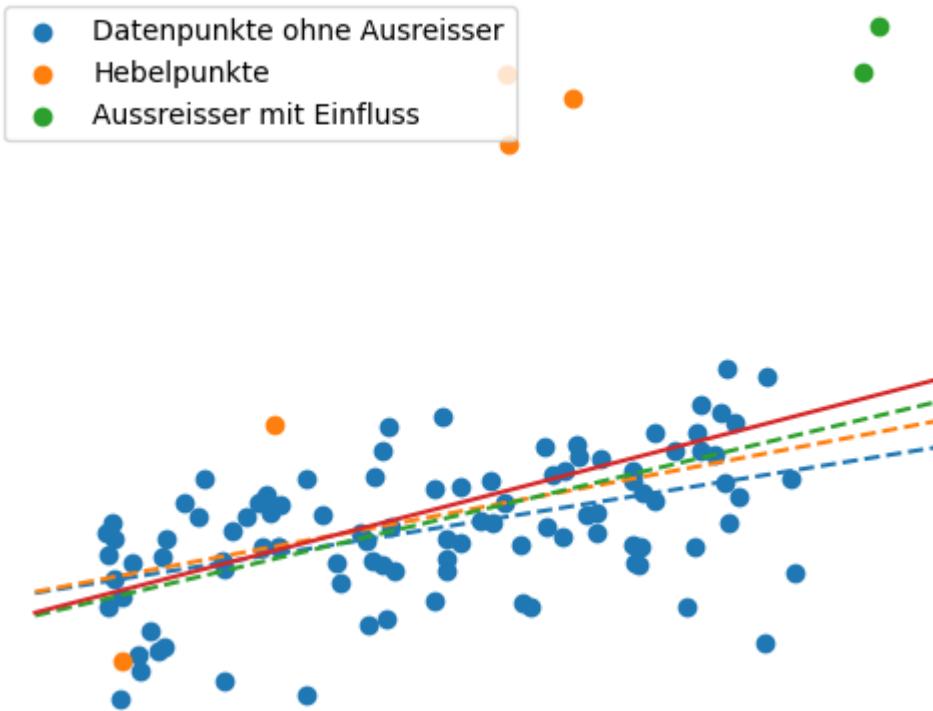
Ausreißer und einflußreiche Beobachtungen

Ausreißer sind Punkte, die aus der Wolke der Datenpunkte herausfallen. Ausreißer, die horizontal von der Mitte der Wolke wegfallen und die Neigung der Regressionslinie nicht beeinflussen, werden als **Leverage-Werte** (Hebelwerte) bezeichnet. Ausreißer, die die Steigung der Regressionsgeraden tatsächlich beeinflussen, werden als **einflußreiche Beobachtungen** bezeichnet, bei denen es sich in der Regel um hohe Leverage-Punkte handelt.

Wir wollen einen Beispieldatensatz erstellen, um das Konzept der statistisch bedeutsamen Beobachtungen zu untersuchen.

► Show code cell source

```
(np.float64(-1.65),
 np.float64(12.65),
 np.float64(-23.52561113057294),
 np.float64(125.83725556438257))
```



Die obige Abbildung zeigt deutlich die Auswirkungen der verschiedenen Arten von Ausreißern. Die blaue gestrichelte Linie zeigt die Regressionslinie ohne Ausreißer, die orange gestrichelte Linie zeigt die Regressionslinie, wenn die orangen Hebelelemente enthalten sind, die grüne gestrichelte Linie zeigt die Regressionslinie, wenn die grünen Ausreißer enthalten sind, und die rote Linie zeigt die Regressionslinie, wenn alle Daten enthalten sind. Offensichtlich haben die grünen Punkte den größten Einfluss auf die Steigung der Regressionslinie!

Leverage

Die [Leverage](#) einer Beobachtung zeigt an, ob sie das Regressionsmodell beeinflussen kann. Diese Beobachtungen sind nicht notwendigerweise ein Fehler, aber sie sollten identifiziert und überprüft werden. Die Leverage wird durch den *H*-Wert gemessen, der den Gesamteinfluss einer einzelnen Beobachtung auf die Modellvorhersagen misst. Der *H*-Wert nimmt Werte zwischen 0 und 1 an. Ein Punkt mit einer Hebelwirkung von Null hat keinen Einfluss auf das Regressionsmodell. Je höher der *H*-Wert ist, desto größer ist der Einfluss des betreffenden Punktes auf das Regressionsmodell.

Cook-Abstand

Eine weitere Methode zur Erfassung einflussreicher Ausreißer ist der [Cook-Abstand](#). Das Maß ist eine Kombination aus Leverage und Residuen der einzelnen Beobachtungen. Je höher die Hebelwirkung und der Rückstand, desto größer ist der Cook-Abstand. Normalerweise werden Punkte mit einem Cook-Abstand von mehr als 1 als einflussreich eingestuft. In Python wird der Cook-Abstand mit dem Attribut `cooks.distance` berechnet.

Andere nützliche Regressionsdiagnosen

Weitere nützliche Werkzeuge für die Regressionsanalysediagnose sind DFFITS (“difference in fit(s)”), die angibt, wie sehr eine Beobachtung den zugehörigen angepassten Wert beeinflusst, und die DFBETAS, die die Änderung der geschätzten Parameter angibt, wenn eine Beobachtung im Verhältnis zu ihrem Standardfehler ausgeschlossen wird.

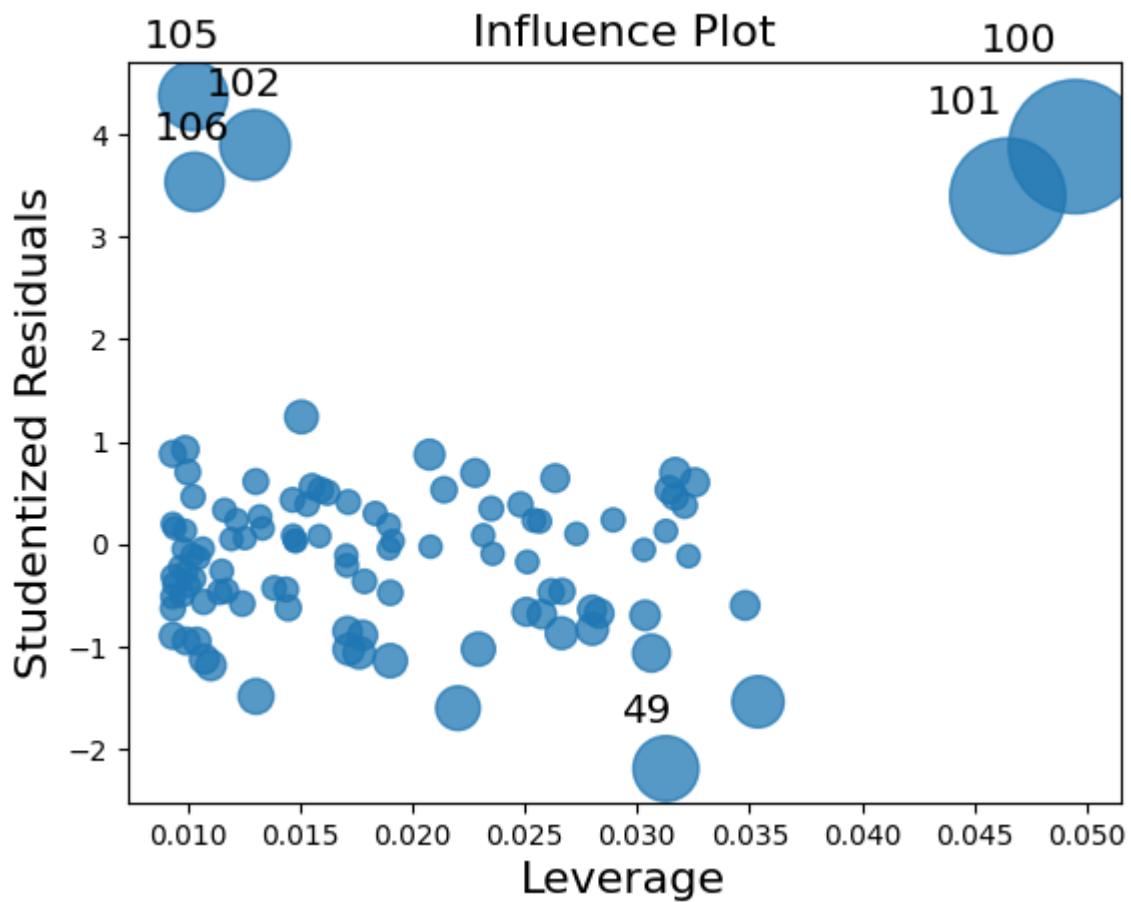
In Paket statsmodels kann auf diese die genannten und weitere diagnostische Verfahren sehr leicht über die Methdode `get_influence().summary_frame()` zurückgegriffen werden.

```
full_model.get_influence().summary_frame().sample(12)
```

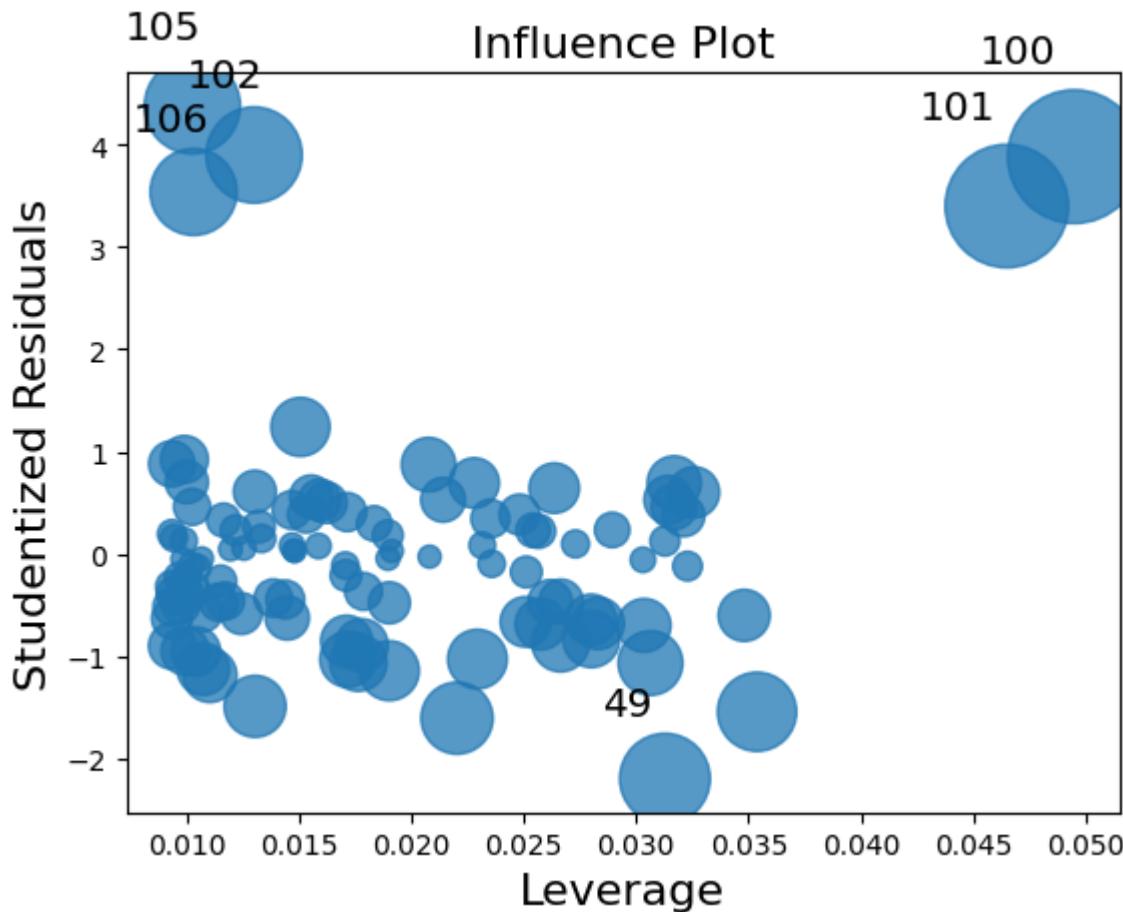
	dfb_const	dfb_x1	cooks_d	standard_resid	hat_diag	dffits_internal	student_r
17	-0.008556	-0.006683	0.000378	-0.274503	0.009938	-0.027502	-0.271
5	-0.000263	0.009673	0.000158	0.152658	0.013345	0.017754	0.151
19	0.066637	-0.022027	0.004280	0.924806	0.009908	0.092515	0.924
86	-0.075890	0.031410	0.004754	-0.949855	0.010429	-0.097511	-0.949
39	0.003107	-0.002012	0.000006	0.027135	0.014813	0.003327	0.021
82	0.023955	-0.091890	0.009254	-1.028980	0.017179	-0.136041	-1.029
21	0.027092	-0.097669	0.010127	-1.061836	0.017647	-0.142317	-1.061
36	0.040591	-0.033427	0.000833	0.236372	0.028942	0.040807	0.239
64	-0.068953	0.023327	0.004514	-0.948047	0.009946	-0.095021	-0.941
69	0.009371	-0.006311	0.000049	0.077862	0.015872	0.009888	0.071
47	0.039709	-0.028573	0.000839	0.299574	0.018354	0.040963	0.298
13	0.125702	-0.105563	0.007940	0.696475	0.031701	0.126018	0.694

Ebenso stehen Visualisierungsmethoden zur Auswahl.

```
_ = sm.graphics.influence_plot(full_model, criterion="cooks")
```



```
_ = sm.graphics.influence_plot(full_model, criterion="DFFITS")
```



Polynomiale Regression

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

Die [polynomiale Regression](#) ist eine spezielle Art der linearen Regression, bei der die Beziehung zwischen der unabhängigen Variablen x und der abhängigen Variablen y durch ein Polynom n -ten Grades in x modelliert wird. Mit anderen Worten, wir nehmen neben dem ursprünglichen linearen Term auch Potenzen zweiter Ordnung und höherer Potenzen einer Variablen in das Modell auf. Die Einbeziehung von Polynomen n -ten Grades führt zu einer nichtlinearen Beziehung zwischen y und x , aber das Modell ist immer noch ein lineares Modell, da die Beziehung zwischen den Koeffizienten (

β_i) und den erwarteten Beobachtungen linear ist. Somit kann die Modellgleichung wie folgt geschrieben werden

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_k x^k + \epsilon.$$

Die Werte der Koeffizienten werden durch Anpassung des Polynoms an die Beobachtungsdaten (y) bestimmt. Wie bei der einfachen linearen Regression, die im vorigen Abschnitt behandelt wurde, geschieht dies durch Minimierung der **Summe der quadrierten Fehler (SSE)**, die durch folgende Gleichung gegeben ist

$$SSE = \sum e^2 = \sum (\hat{y} - y)^2.$$

Bei der Anpassung eines Polynoms an Beobachtungen stellt sich das Problem der Wahl der Ordnung k des Polynoms. Wie man die richtige Zahl für das Polynom wählt, ist eine Frage eines wichtigen Konzepts, das **Modellauswahl** oder [**Informationskriterium**](#) genannt wird. Der Einfachheit halber verwenden wir die Wurzel des mittleren quadratischen Fehlers ([root-mean-square error, RMSE](#)), definiert durch

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (\hat{y} - y)^2}{n}}$$

um die Eignung des Modells zu bewerten.

Polynomiale Regression

Wir beginnen mit unserer Übung, indem wir Daten sammeln. Die Daten werden von einer Funktion generiert, die Sie nicht kennen. Diese Vorbedingung macht das Beispiel realistischer, da wir in realen Anwendungen die genauen Spezifikationen des zugrunde liegenden Datenerzeugungsprozesses nicht kennen. Am Ende dieses Abschnitts lüften wir das Geheimnis des Datengenerierungsprozesses.

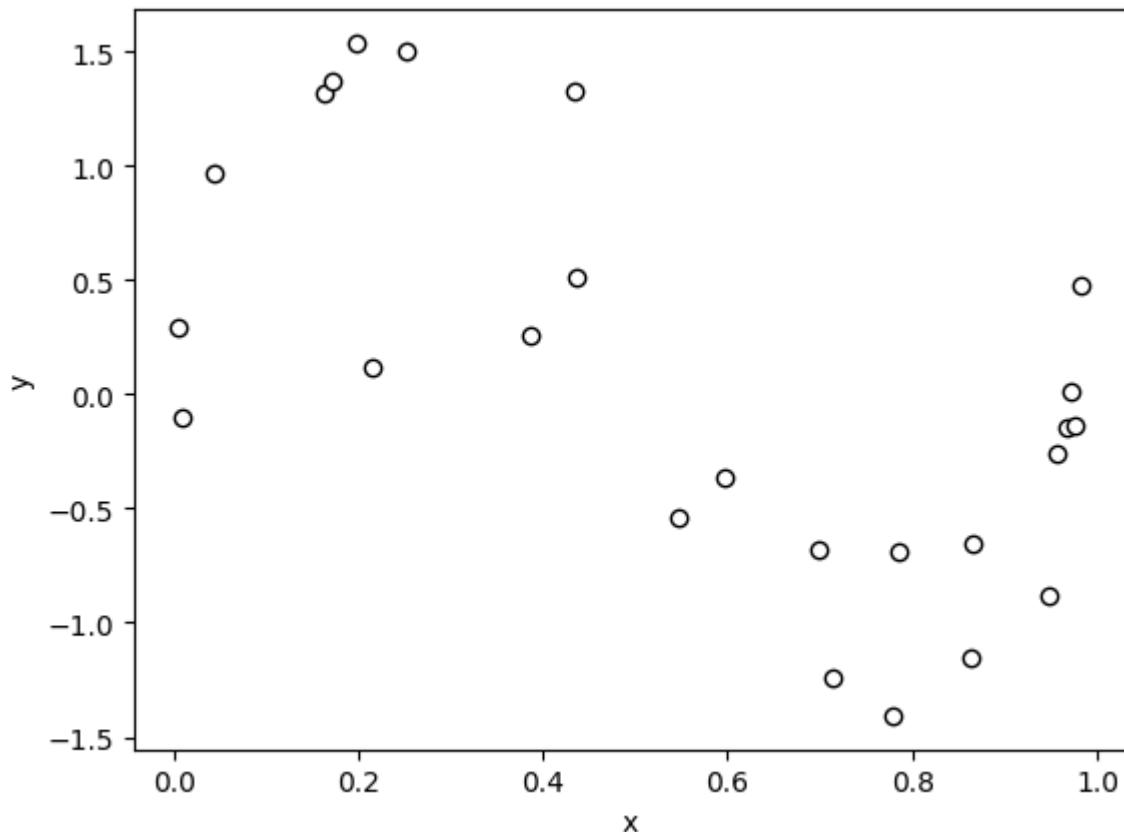
Dies sind die Daten, unsere Beobachtungen, in tabellarischer Form. Wir haben 25 Datenpunkte, jeder Datenpunkt ist ein (x, y) Paar.

```
n = 25
np.random.seed(4)
x = np.random.uniform(0, 1, n)
y = np.sin(2 * np.pi * x) + np.random.normal(0, 0.35, n)
poly_data = pd.DataFrame({"x": x, "y": y})
```

Hier sind die Daten in Form eines Streudiagramms dargestellt:

```
fig, ax = plt.subplots()
X = poly_data["x"]
y = poly_data["y"]
ax.scatter(X, y, edgecolor="k", color="white")
ax.set_xlabel("x")
ax.set_ylabel("y")
```

```
Text(0, 0.5, 'y')
```



Anpassen einer Kurve in Python: Die Notation in Python

Python bietet leistungsstarke Funktionen zur Anpassung eines Polynoms an Daten. Um ein k -dimensionales Polynom anzupassen, verwenden wir `linear_model` aus dem `sklearn` Paket und fügen dem Funktionsaufruf `LinearRegression()` zusätzliche `PolyomialFeatures` hinzu. Darüber hinaus gibt es zwei verschiedene Möglichkeiten, eine polynomiale Regression zu kodieren.

Für ein Polynom 2. Ordnung besteht die erste Möglichkeit darin, `model.predict(X-werte)` einzugeben und zu plotten, und die zweite Möglichkeit ist die Koeffizienten mit `model.coef_` aus dem Model auszulesen und daraus äquivalent zu $\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_k x^k + \epsilon$,

```
model.coef_[0][0] + model.coef_[0][1] * X-werte + model.coef_[0][2] * (X-werte)**2 + ...
```

zu konstruieren.

Um die Verwirrung zu lindern, zeigen wir ein Beispiel in Python. Wir konstruieren zwei Polynome der Ordnung 2 für `poly_data`.

```
X = poly_data["x"].values.reshape(-1, 1)
y = poly_data["y"].values.reshape(-1, 1)

# Polynomialer Fit
poly = PolyomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

model = LinearRegression()
model.fit(X_poly, y)
```

▼ `LinearRegression` ⓘ ⓘ
`LinearRegression()`

Zum Auslesen der Koeffizienten und des Achsenabschnitts des Modells verwenden wir das Attribut `coef_` und `intercept_`.

Wir überprüfen die Ergebnisse des Modells.

```
print(f"Achsenabschnitt: {model.intercept_}")
print(f"Koeffizienten: {model.coef_}")
```

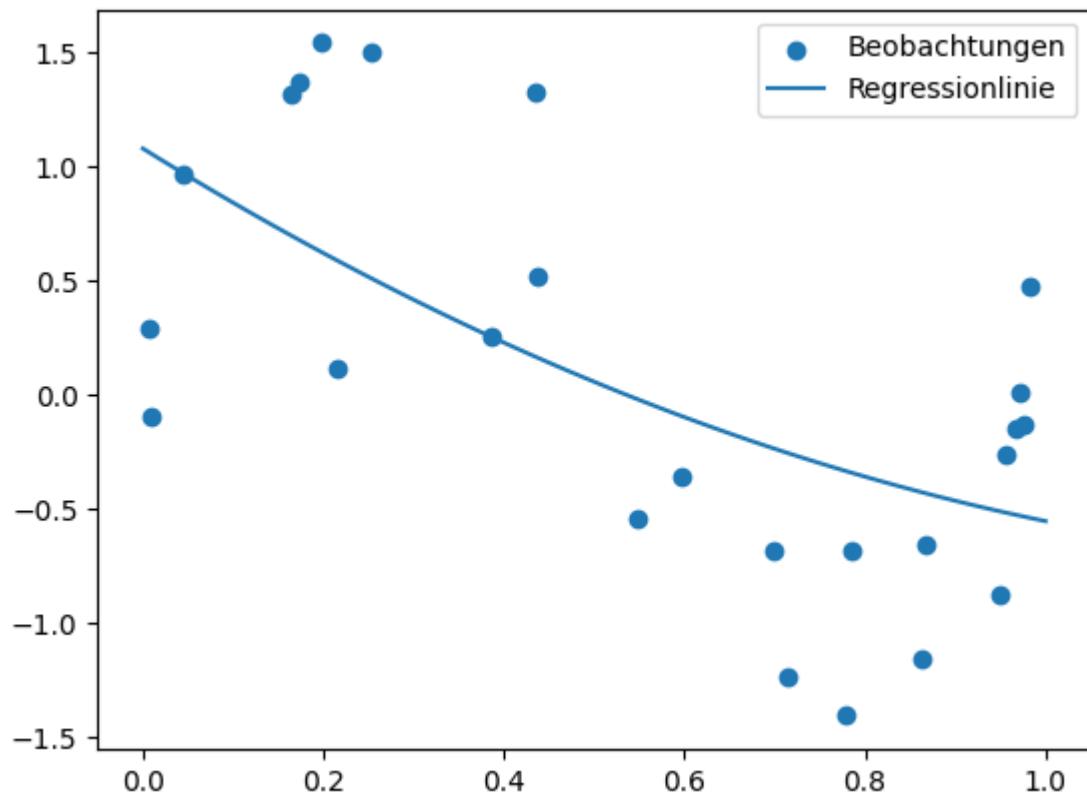
```
Achsenabschnitt: [1.07678298]
Koeffizienten: [[ 0.          -2.45381544  0.82240915]]
```

Bei der Ausgabe der Koeffizienten dient der nullte Koeffizient als Platzhalter für den Achsenabschnitt und kann daher ignoriert werden. Setzen wir nun die Parameter in die Gleichung ein, erhalten wir die Regressionslinie.

```
# Definiere x- Achse zur Vorhersage von Datenpunkten
x_axis = np.linspace(0, 1, 50)
# Konstruiere Regressionslinie
regline = (
    model.intercept_ + model.coef_[0][1] * x_axis + model.coef_[0][2] * x_axis**2
)

fig, ax = plt.subplots()
ax.scatter(poly_data["x"], poly_data["y"], label="Beobachtungen")
ax.plot(x_axis, regline, label="Regressionslinie")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7fb29eeb6b00>
```

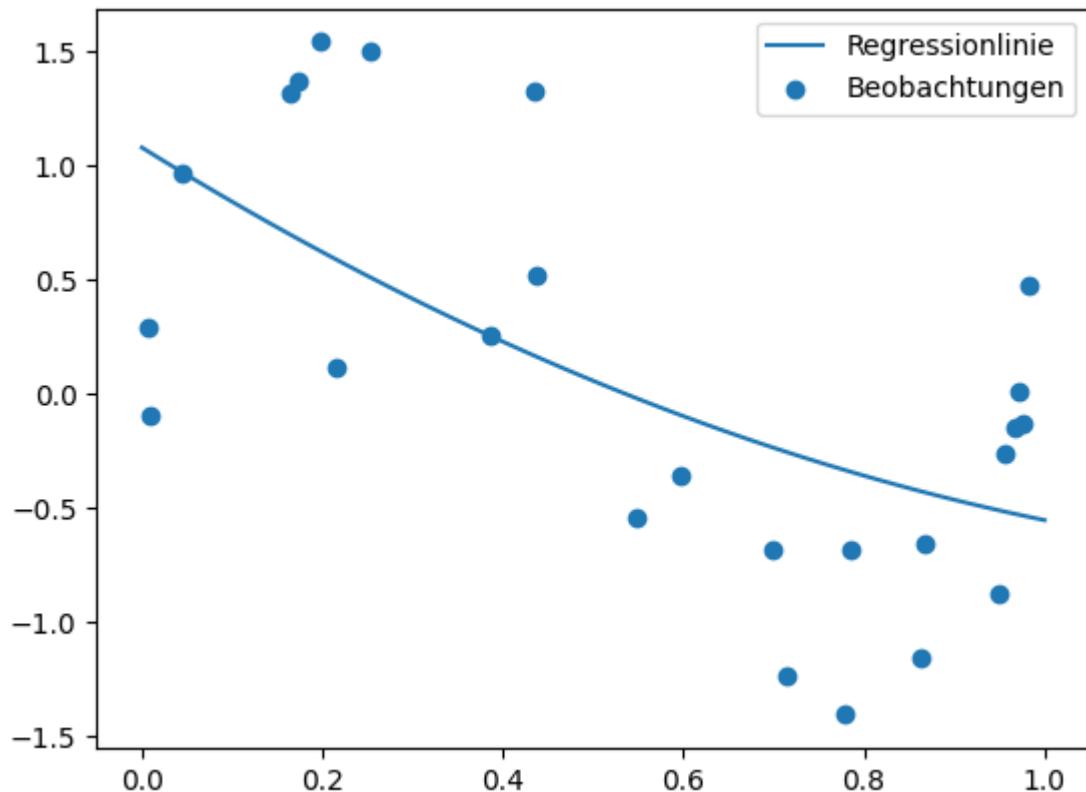


Andererseits können wir die Regressionslinie auch mit der Methode `predict()` für beliebige x -Werte erzeugen.

```
X_poly = poly.transform(x_axis.reshape(-1, 1))
regline = model.predict(X_poly)

fig, ax = plt.subplots()
ax.plot(x_axis, regline, label="Regressionslinie")
ax.scatter(x=x, y=y, label="Beobachtungen")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7fb29ee9ffa0>
```



Anpassen einer Kurve in Python (Fortsetzung)

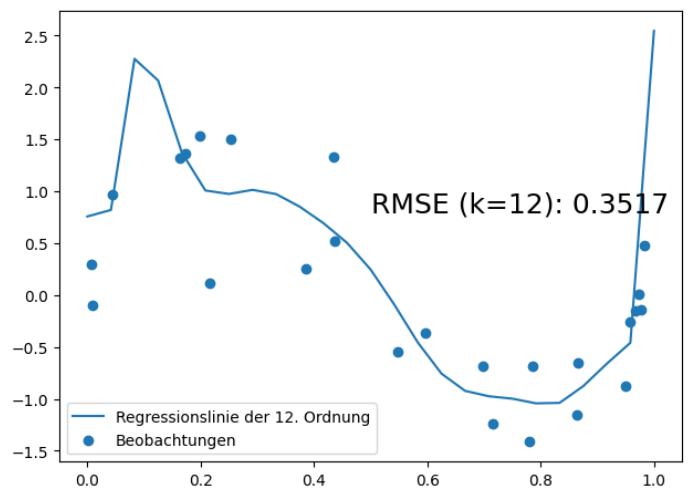
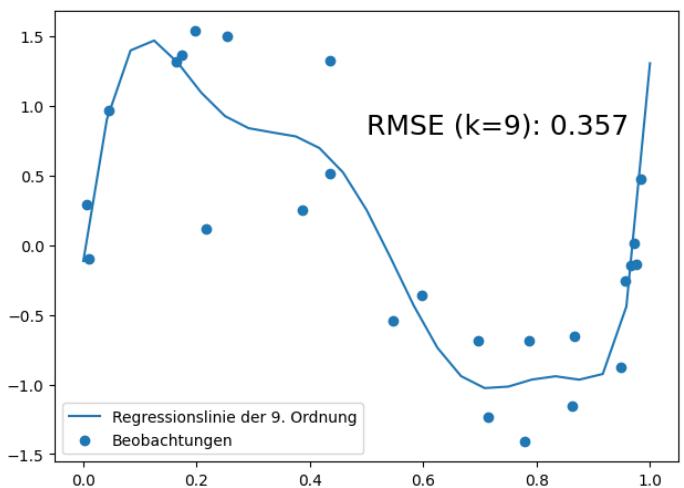
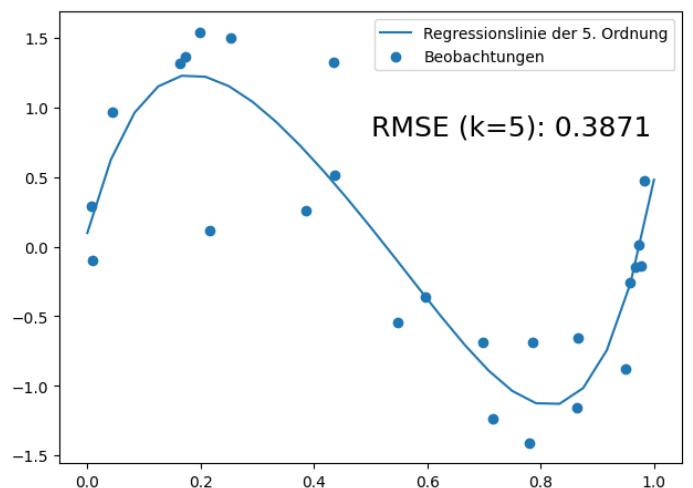
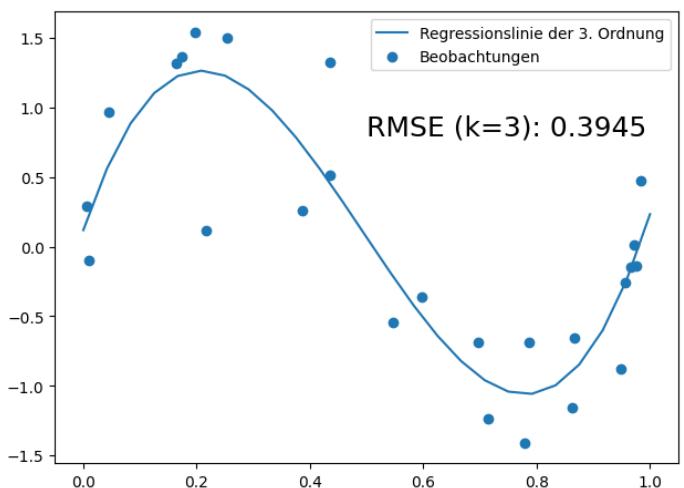
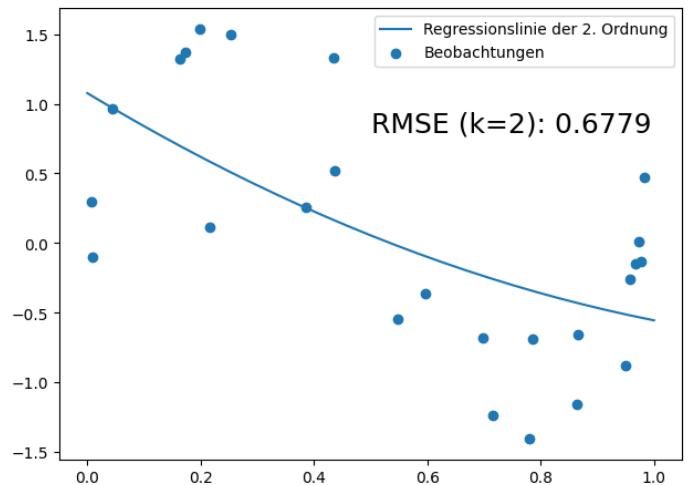
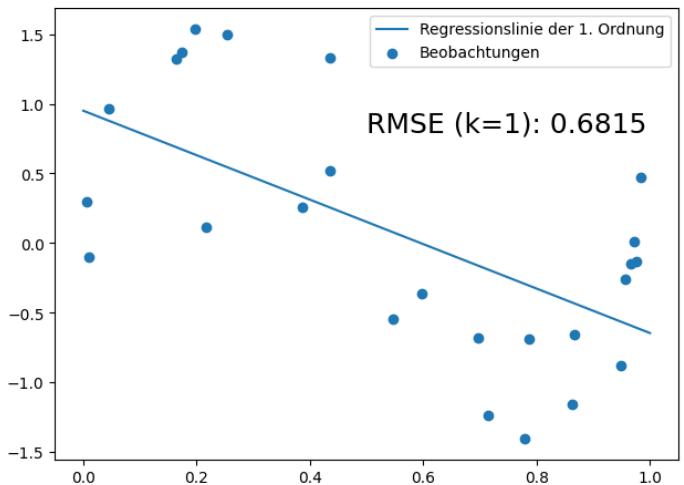
Da wir nun die Notation in Python kennen, beginnen wir mit der Erstellung von 6 verschiedenen Modellen, mit $k = 1, 2, 3, 5, 9, 12$. Für jedes Modell berechnen wir den *RMSE*. Schließlich stellen wir die Daten zusammen mit der Regressionslinie dar, die durch jedes einzelne Modell gegeben ist. Der Einfachheit halber konstruieren wir eine Schleife, um den Kodierungsaufwand zu verringern.

```
RMSE = []
X = poly_data["x"].values.reshape(-1, 1)
y = poly_data["y"].values.reshape(-1, 1)
x_axis = np.linspace(0, 1, 25)

orders = [1, 2, 3, 5, 9, 12]

fig, ax = plt.subplots(figsize=(16, 18), ncols=2, nrows=3)
ax = np.ravel(ax)
for e, order in enumerate(orders):
    poly = PolynomialFeatures(degree=order)
    X_poly = poly.fit_transform(X)
    model = LinearRegression()
    model.fit(X_poly, y)
    y_hat = model.predict(X_poly)
    mse = mean_squared_error(y, y_hat)
    rmse = np.sqrt(mse)
    RMSE.append(rmse)

    X_poly = poly.transform(x_axis.reshape(-1, 1))
    regline = model.predict(X_poly)
    ax[e].plot(x_axis, regline, label=f"Regressionslinie der {order}. Ordnung")
    ax[e].scatter(x=X, y=y, label="Beobachtungen")
    ax[e].text(s=f"RMSE (k={order}): {np.round(rmse, 4)}", y=0.8, x=0.5, size=18)
    ax[e].legend()
```

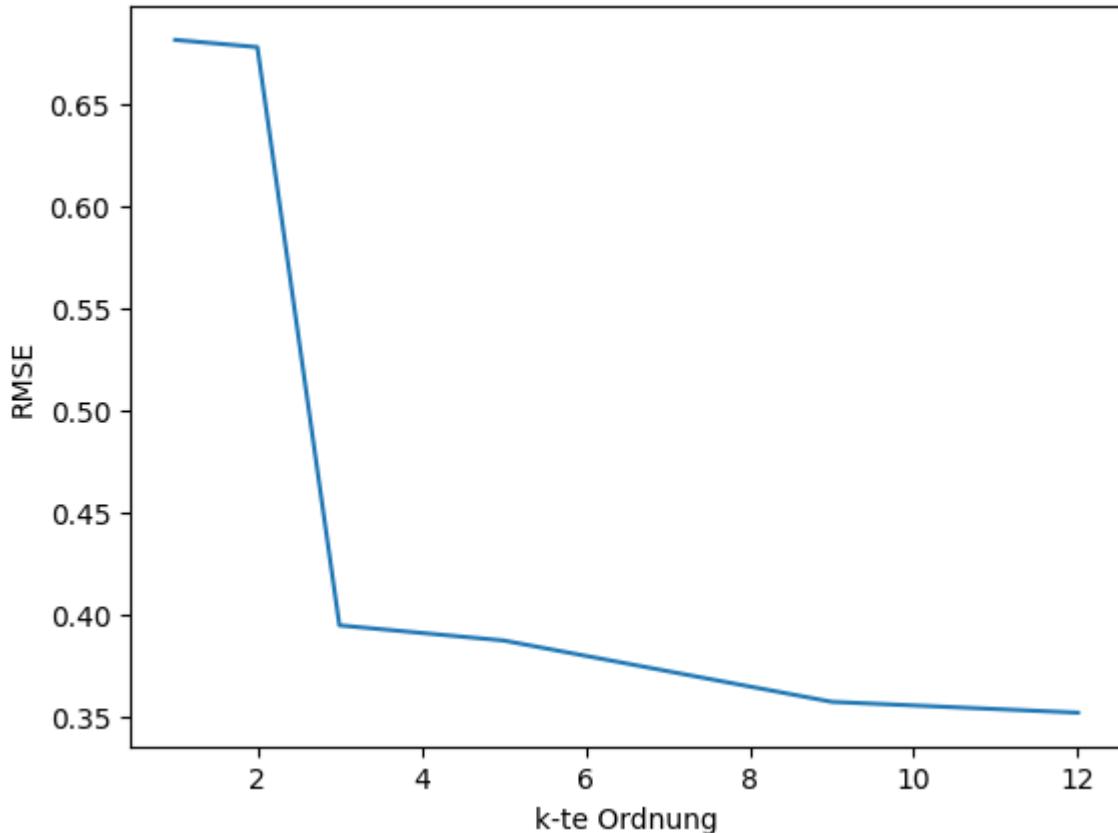


Fantastische Diagramme! Die Abbildung zeigt, dass wenn wir k , die Ordnung des Polynoms, erhöht, wird die Kurve flexibler und passt immer besser zu den Daten. Je besser die Daten angepasst werden, desto geringer wird der Fehler, $RMSE$.

Der Einfachheit halber plotten wir den $RMSE$ gegen k .

```
fig, ax = plt.subplots()
ax.plot(orders, RMSE)
ax.set_ylabel("RMSE")
ax.set_xlabel("k-te Ordnung")
```

```
Text(0.5, 0, 'k-te Ordnung')
```



Daher stellt sich wieder einmal die Frage, welches Polynom am besten zu den Daten passt. Glauben wir, dass das Polynom der Ordnung 9 am besten zu dem zugrunde liegenden Datenerzeugungsprozess passt? Obwohl wir eine hervorragende Anpassung an die Beobachtungsdaten erhalten, indem wir die Ordnung des Polynoms erhöhen, bleibt es fraglich, ob das Polynom hoher Ordnung gut verallgemeinert werden kann. Stellen Sie sich vor, wir führen eine neue Messreihe durch und erhalten neue Daten. Glauben Sie, dass die wild schwingende Kurve eines Polynoms hoher Ordnung immer noch gut zu den Daten passt? Nein, wahrscheinlich nicht!

Dieses Verhalten wird als Überanpassung bezeichnet. Erinnern Sie sich daran, dass das Ziel darin besteht, die Parameter aus den Daten zu lernen. Wir sind also daran interessiert, eine gute Verallgemeinerung des Modells zu erreichen und nicht unbedingt perfekt angepasste Beobachtungsdaten.

Aus den Daten lernen

Wie können wir das Problem lösen? Wie bestimmen wir das beste Polynom n -ter Ordnung für unseren Datensatz? Nun, es gibt viele Methoden und Strategien, um einer Überanpassung entgegenzuwirken. In diesem Abschnitt verfolgen wir einen einfachen Ansatz. Zunächst teilen wir den Datensatz in zwei Teile auf. Einen Teil nennen wir **Trainingsmenge**, den anderen Teil nennen wir **Validierungsmenge**. Dann verwenden wir alle Daten des Trainingssatzes, um die Modellparameter β_i zu lernen, und zwar auf die gleiche Weise wie oben. Danach wenden wir das gelernte Modell an, um die Daten des Validierungssatzes vorherzusagen, und bewerten die Leistung des Modells, indem wir den $RMSE$ berechnen. Wir verwenden also die Validierungsmenge, um die, durch k gegebene, Komplexität des Modells zu optimieren.

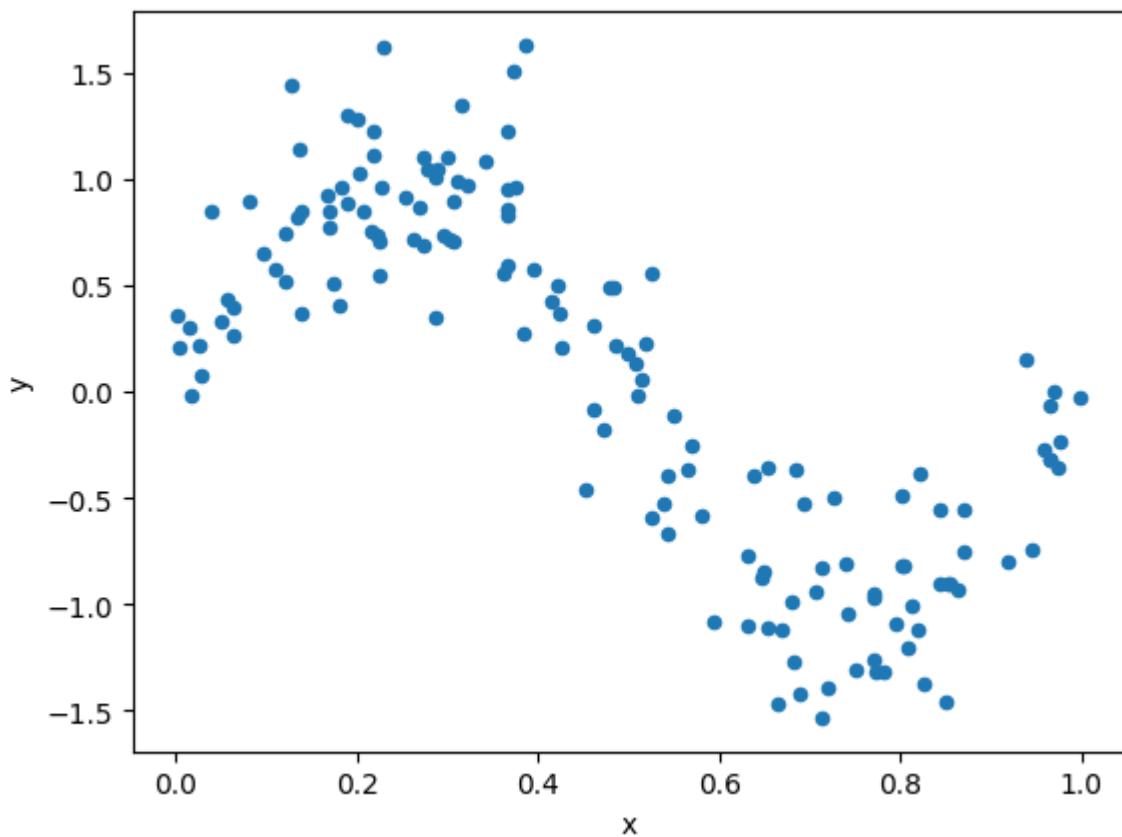
Leider brauchen wir, wenn wir aus Daten lernen wollen, letztendlich auch Daten, aus denen wir lernen können. Bislang haben wir mit 25 Beobachtungen gearbeitet. Das ist nicht viel. In realen Anwendungen müssten wir wahrscheinlich neue Beobachtungen durch eine neue Messreihen gewinnen. In unserer Übung können wir jedoch relativ leicht mehr Daten erzeugen. Daher setzen wir dieses Beispiel mit einem neuen Datensatz von 150 Beobachtungen fort.

Lassen Sie uns die Daten plotten!

```
n = 150
np.random.seed(415)
x = np.random.uniform(0, 1, n)
y = np.sin(2 * np.pi * x) + np.random.normal(0, 0.3, n)
new_poly_data = pd.DataFrame({"x": x, "y": y})
```

```
new_poly_data.plot.scatter(x="x", y="y")
```

```
<Axes: xlabel='x', ylabel='y'>
```



Trainings- und Validierungsmenge

Nun sind wir bereit, unsere Trainings- und Validierungsmenge zu erstellen. Dazu verwenden wir die Funktion `train_test_split()` aus dem `sklearn`-Paket. Wir teilen die Daten so auf, dass 65% der Daten dem Trainingsset und der Rest, 35% der Daten, dem Validationsset zugewiesen werden.

```
X = new_poly_data.x.values.reshape(-1, 1)
y = new_poly_data.y.values.reshape(-1, 1)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.35, random_state=42)
```

```
# Überprüfen der Dimension von Training und Validierungsdatensatz
print("Trainingset")
print(f"X_train: {np.shape(X_train)}")
print(f"y_train: {np.shape(y_train)}")

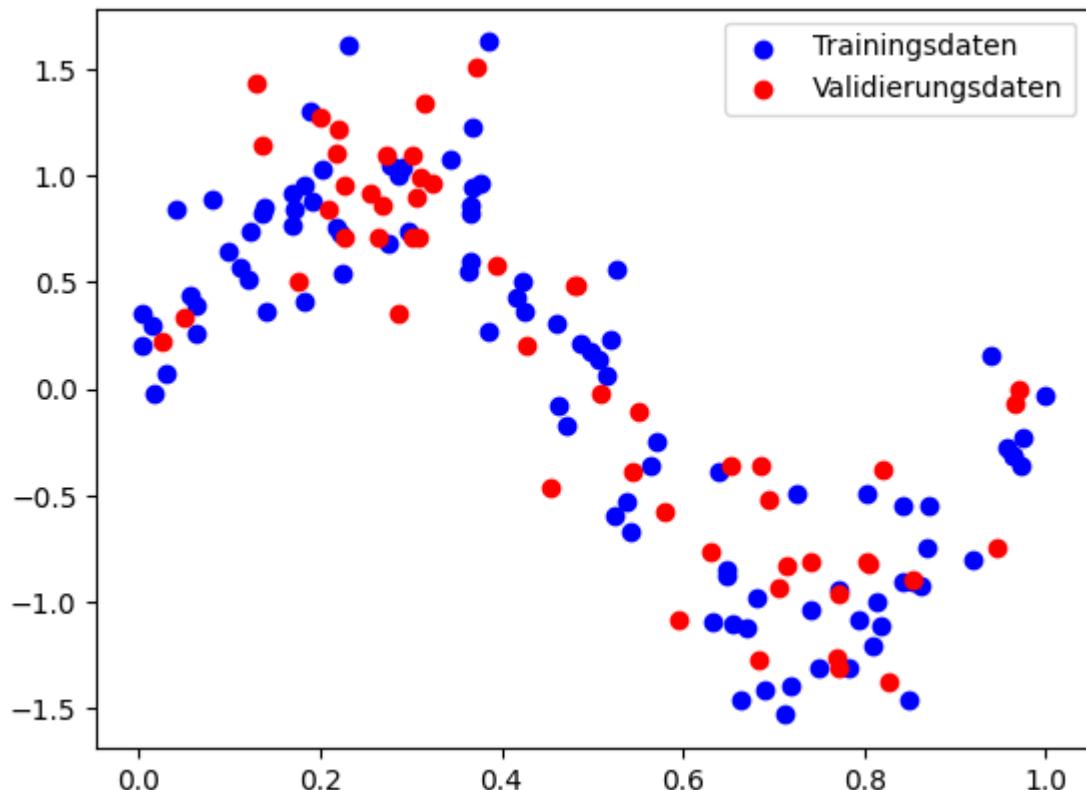
print("\nValidierungsset")
print(f"X_val: {np.shape(X_val)}")
print(f"y_val: {np.shape(y_val)}")
```

```
Trainingset
X_train: (97, 1)
y_train: (97, 1)

Validierungsset
X_val: (53, 1)
y_val: (53, 1)
```

```
# Plot
plt.scatter(X_train, y_train, label="Trainingsdaten", color="blue")
plt.scatter(X_val, y_val, label="Validierungsdaten", color="red")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb29e376230>
```



Modellbildung und Modellbewertung

In dieser Übung werden wir 10 Modelle mit $k = 1, 2, \dots, 10$ erstellen. Wir bewerten jedes der 10 polynomialen Regressionsmodelle, indem wir den $RMSE$ auf dem Trainingssatz berechnen. Nach der Erstellung des Modells und der Ermittlung der Modellparameter β_i verwenden wir das Modell, um die Antwortvariable in der Validierungsmenge vorherzusagen. Auch hier stützen wir uns auf den $RMSE$, um die Vorhersagen für den Validierungssatz zu bewerten. Schließlich stellen wir den $RMSE$ jedes Modells sowohl für die Trainingsmenge als auch für die Validierungsmenge dar. Anhand des $RMSE$ bewerten wir die Generalisierung des Modells.

```
rmse_train = []
rmse_val = []

for order in range(1, 11):
    # Polynomial Fit
    poly = PolynomialFeatures(degree=order)
    X_train_poly = poly.fit_transform(X_train)  # .reshape(-1, 1)
    X_val_poly = poly.fit_transform(X_val)  # .reshape(-1, 1)
    model = LinearRegression()
    model.fit(X=X_train_poly, y=y_train)

    # Berechne RMSE - Trainingsset
    mse = mean_squared_error(y_train, model.predict(X_train_poly))
    rmse = np.sqrt(mse)
    print(f"RMSE train (k={order}): {np.round(rmse, 3)}")
    rmse_train.append(rmse)

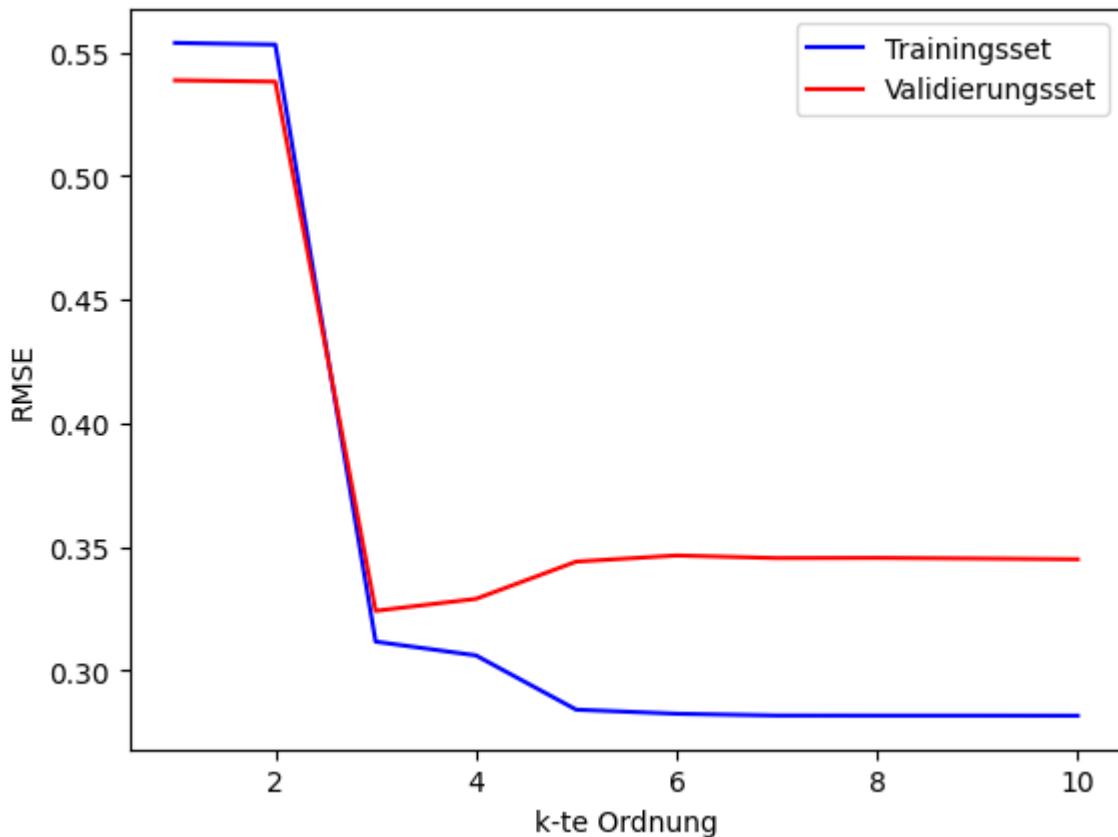
    # Berechne RMSE - Validierungsset
    mse = mean_squared_error(y_val, model.predict(X_val_poly))
    rmse = np.sqrt(mse)
    print(f"RMSE val (k={order}): {np.round(rmse, 3)}")
    rmse_val.append(rmse)
    print("-----")
```

```
RMSE train (k=1): 0.554
RMSE val (k=1): 0.539
-----
RMSE train (k=2): 0.553
RMSE val (k=2): 0.538
-----
RMSE train (k=3): 0.312
RMSE val (k=3): 0.324
-----
RMSE train (k=4): 0.306
RMSE val (k=4): 0.329
-----
RMSE train (k=5): 0.284
RMSE val (k=5): 0.344
-----
RMSE train (k=6): 0.282
RMSE val (k=6): 0.346
-----
RMSE train (k=7): 0.282
RMSE val (k=7): 0.345
-----
RMSE train (k=8): 0.282
RMSE val (k=8): 0.345
-----
RMSE train (k=9): 0.282
RMSE val (k=9): 0.345
-----
```

```
RMSE train (k=10): 0.282
RMSE val (k=10): 0.345
-----
```

```
k = range(1, 11)
fig, ax = plt.subplots()
ax.plot(k, rmse_train, label="Trainingsset", color="blue")
ax.plot(k, rmse_val, label="Validierungsset", color="red")
ax.set_ylabel("RMSE")
ax.set_xlabel("k-te Ordnung")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7fb29df181f0>
```



Die Abbildung zeigt, dass der Fehler bei den Trainingsdaten (blaue Linie) stetig abnimmt. Das macht durchaus Sinn, denn je komplexer das Modell wird, indem k erhöht wird, desto besser passt das Modell zu den Trainingsdaten. Das gleiche Verhalten haben wir im obigen Abschnitt beobachtet, als wir unser Modell mit nur 25 Beobachtungen trainiert haben. Wenn wir uns den $RMSE$ für den Validierungssatz (rote Linie) ansehen, sehen wir, dass mit zunehmendem k und damit zunehmender Modellkomplexität der Fehler abnimmt. Es gibt jedoch einen "Sweet Spot", der durch den niedrigsten $RMSE$ angezeigt wird, an dem das Modell gerade komplex genug ist, um auf den bisher ungesiehenen Validierungsdaten gut zu generalisieren. Wenn die Modellkomplexität weiter zunimmt, beginnt auch der $RMSE$ zu steigen. Dies deutet auf eine Überanpassung des Modells hin. Das Modell merkt sich also die Daten in der Trainingsmenge gut, aber die Vorhersagekraft des Modells für bisher ungesiehene Daten, wie die Daten der Validierungsmenge, wird schlechter. Ein Blick auf die obige Abbildung zeigt, dass der niedrigste Fehler im Validierungssatz, der so genannte Sweet Spot, für ein Regressionsmodell 3-ter Ordnung erreicht wird.

Vorstellung des Modells

Im vorangegangenen Abschnitt haben wir festgestellt, dass ein polynomiales Regressionsmodell der Ordnung 3 in der Validierungsgruppe am besten funktioniert. Nun stellen wir dieses Modell auf dem gesamten Datensatz dar, um seine Qualität visuell zu bewerten. Außerdem stellen wir die Funktion dar, die dem Prozess der Datengenerierung zugrunde liegt.

Datenerzeugung: Die Eingabewerte x_n für die zugrundeliegende Funktion werden gleichmäßig im Bereich $U(0, 1)$ erzeugt, und die entsprechenden Zielwerte y erhält man, indem man zunächst die entsprechenden Werte der Funktion $\sin(\pi x)$ berechnet und dann zufälliges Rauschen mit einer Gaußschen Verteilung mit einer Standardabweichung von 0,35 hinzufügt.

```
# Erzeuge Daten
n = 100
np.random.seed(4)
x = np.random.uniform(0, 1, n)
y = np.sin(2 * np.pi * x) + np.random.normal(0, 0.35, n)
raw_data = pd.DataFrame({"x": x, "y": y})
```

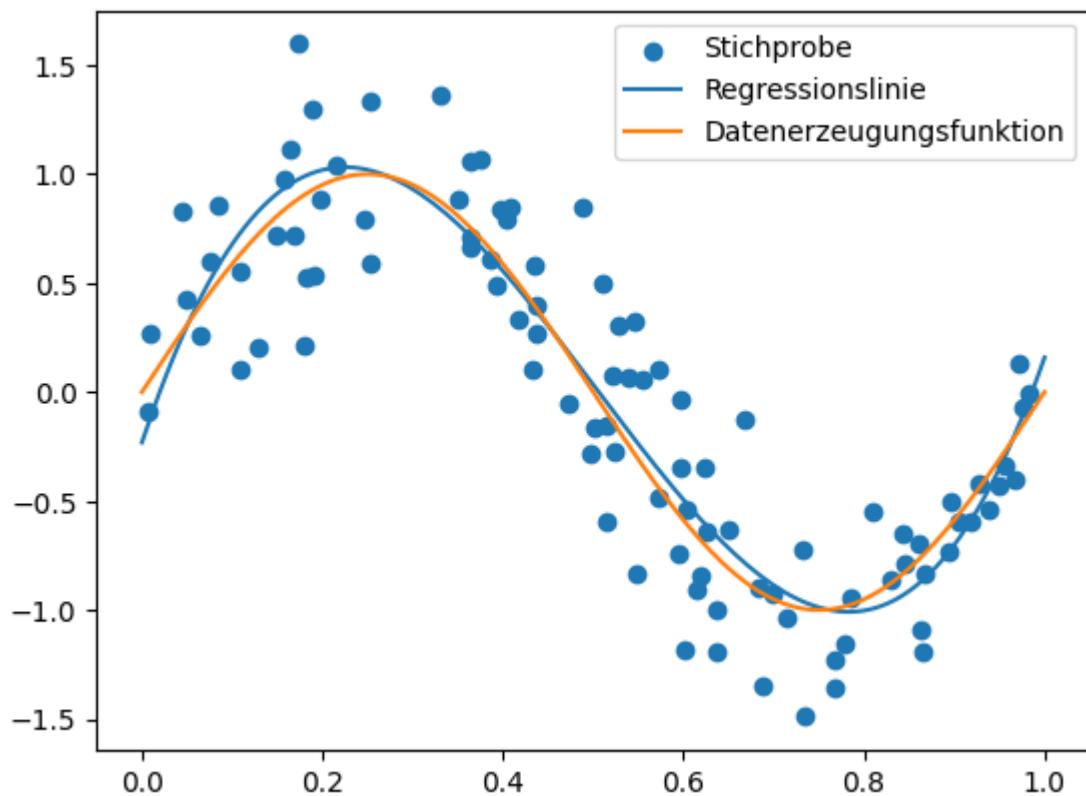
```
X = raw_data["x"].values.reshape(-1, 1)
y = raw_data["y"].values.reshape(-1, 1)

# Polynomial Fit
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)

x_axis = np.linspace(0, 1, 150)
x_axis_poly = poly.fit_transform(x_axis.reshape(-1, 1))
y_reg_line = model.predict(x_axis_poly)

fig, ax = plt.subplots()
ax.scatter(X, y, label="Stichprobe")
ax.plot(x_axis, y_reg_line, label="Regressionslinie")
ax.plot(x_axis, np.sin(2 * np.pi * x_axis), label="Datenerzeugungsfunktion")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7fb29df83010>
```



Die Abbildung zeigt, dass unser Modell die Daten gut abbildet und wir daher recht zufrieden damit sein können.

Übungsaufgaben

Lineare Regression - Grundbegriffe

1. Was versteht man unter Residuen?
2. Was versteht man unter Homoskedastizität?
3. Was sind Heelpunkte?

Lösungen

Lineare Regression

1. Führen Sie eine lineare Regression für die folgenden Daten (x, y) durch und stellen Sie die Regressionsgerade und die Daten graphisch dar.

```
import numpy as np

n = 10
noise = np.random.normal(0, 1.4, n)
x = np.arange(0, n, 1)
y = 2 * x + noise
```

```
# Frage 1 ...
```

Lösungen

► Show code cell content

Polynomiale Regression

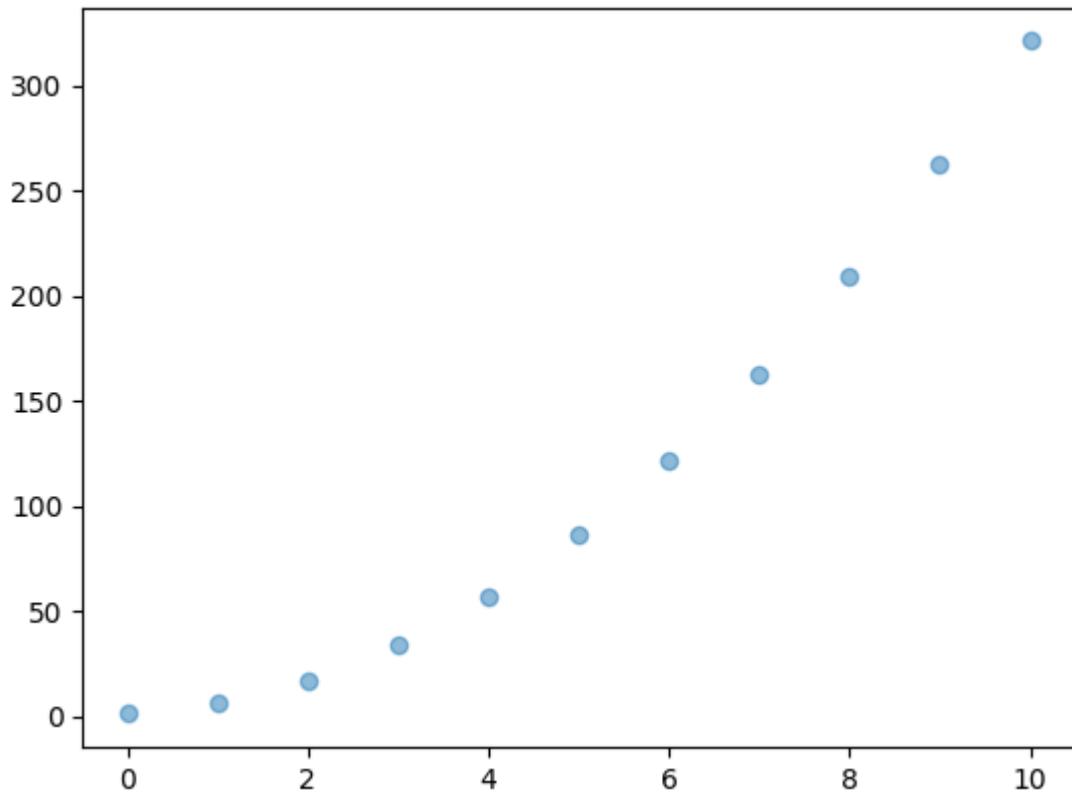
1. Führen Sie eine polynomiale Regression 2-ten Grades für die folgenden Daten (x , y) durch und stellen Sie die Regressionsgerade und die Daten graphisch dar.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([1, 6, 17, 34, 57, 86, 121, 162, 209, 262, 321])

fig, ax = plt.subplots()

ax.scatter(
    x,
    y,
    alpha=0.5,
)
plt.show()
```



```
# Frage 1 ...
```

Lösungen

► Show code cell content

Lernziele

Logistische Regression

- Den Studierenden wird das Modell der logistischen Regression vorgestellt
- Die Logit Funktion und ihr Zusammenhang mit Odds und der Sigmoidfunktion wird besprochen
- Die Studierenden lernen anhand eines Beispiels die praktische Anwendung des logistischen Regressionsmodells

- Zusammengefasst werden folgende Begriffe und Methoden besprochen : Logistische Regression , Logit-Funktion , Odds - Log Odds , Sigmoidfunktion

Logistische Regression

```
import math
import folium
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

Die [logistische Regression](#), auch bekannt als **Logit-Regression** oder **Logit-Modell**, ist ein probabilistisches lineares Modell für dichotome Daten. Die Antwortvariable ist eine binäre Variable (Nominalvariable), d. h. die Variable hat zwei Kategorien oder zwei Werte: Wahr vs. Falsch oder 1 vs. 0 oder Erfolg vs. Misserfolg, mit den Wahrscheinlichkeiten π bzw. $1 - \pi$. Somit folgt die Antwortvariable einer Binomialverteilung, die wie folgt geschrieben wird

$$y \sim B(\eta, \pi),$$

wobei η der binomische Nenner ist, der für eine binäre Variable 0 oder 1 ist, und π die Erfolgswahrscheinlichkeit ist.

Die Logit-Funktion

Das Ergebnis einer logistischen Regression ist eine Wahrscheinlichkeit (π), also ein Wert zwischen 0 und 1. Außerdem ist dieses Ergebnis eine lineare Funktion bekannter Kovariaten x_i , was nur ein anderes Wort für die Beobachtungen in unserem Datensatz ist.

$$\pi = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$$

Für ein einfaches logistisches Regressionsmodell mit einer Vorhersagevariablen vereinfacht sich die obige Gleichung zu

$$\pi = \beta_0 + \beta_1 x_1.$$

Der rechte Term der Gleichung kann jedoch jeden reellen Wert annehmen, während der linke Term der Gleichung eine Wahrscheinlichkeit auf der Skala von 0 bis 1 ist. Um die Skala der Daten (rechter Term) in eine Wahrscheinlichkeit zwischen 0 und 1 abzubilden wenden wir eine so genannte **Verknüpfungsfunktion** an.

Für das logistische Regressionsmodell ist diese Verknüpfungsfunktion die [Logit-Funktion](#). Die Logit-Funktion bildet Wahrscheinlichkeiten aus dem Bereich (0, 1) auf den gesamten reellen Zahlenbereich $(-\infty, \infty)$. Sie wird geschrieben als

$$\eta = \text{logit}(\pi),$$

wobei π die Wahrscheinlichkeit ist.

Um den Logit zu verstehen, führen wir zunächst die [Chance](#) oder kurz **Odds** ein. Die Odds (o) können geschrieben werden als

$$o = \frac{\pi}{1 - \pi},$$

wobei π die Wahrscheinlichkeit ist, dass ein Ereignis eintritt. Wenn die Wahrscheinlichkeit eines Ereignisses 0,5 beträgt, ist die Wahrscheinlichkeit eins zu eins oder gerade $\left(\frac{0.5}{1-0.5} = 1\right)$. Wenn die Wahrscheinlichkeit $1/3$ beträgt, ist die Wahrscheinlichkeit eins zu zwei $\left(\frac{1/3}{1-1/3} = 1/2\right)$. Die Odds können jeden positiven Wert annehmen und unterliegen daher keiner Beschränkung nach oben $[0, \infty]$. Daher definieren wir weiter die **Log-Odds**, die der Logarithmus der Odds ist:

$$\eta = \text{logit}(\pi) = \log \left(\frac{\pi}{1 - \pi} \right)$$

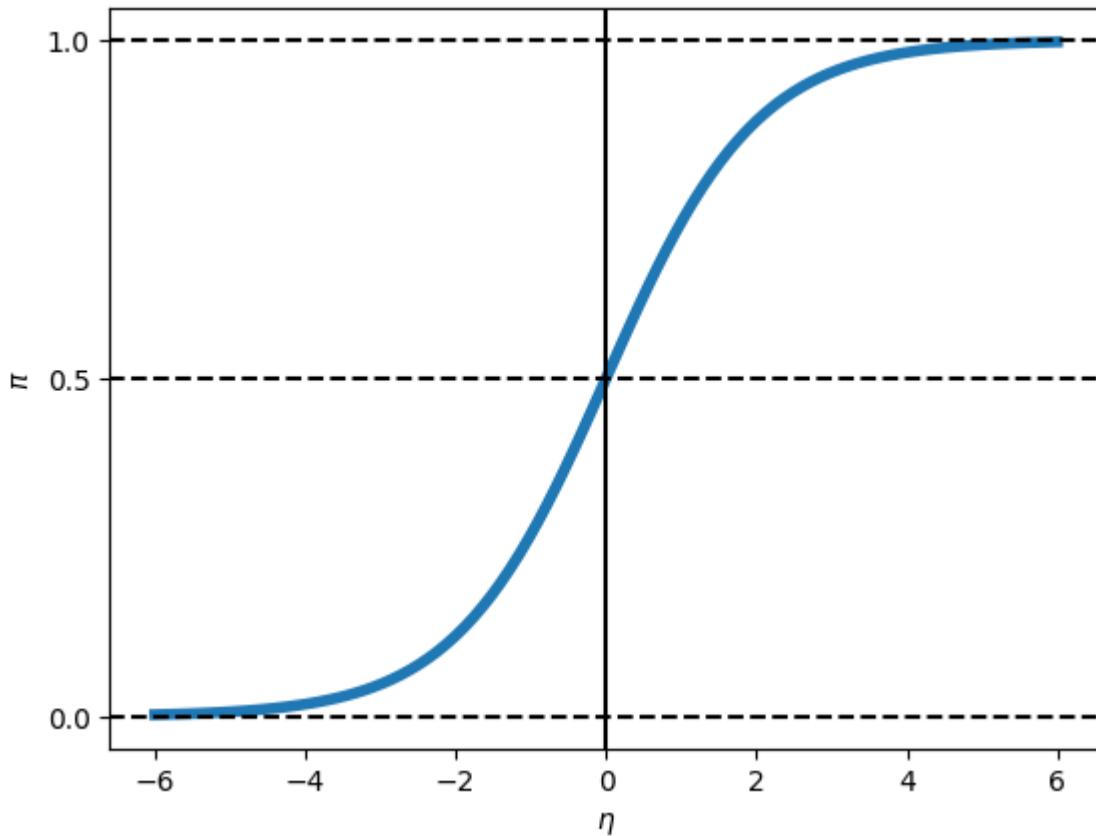
Diese logarithmische Funktion bewirkt, dass die Beschränkung der Untergrenze aufgehoben wird, so dass die Funktion, die [Logit-Funktion](#), unsere Verknüpfungsfunktion, Werte im Bereich von 0 bis 1 in Werte über den gesamten realen Zahlenbereich $]-\infty, +\infty[$ umwandelt. Wenn die Wahrscheinlichkeit $1/2$ beträgt sind die Odds gerade und der Logit ist Null. Negative Logits stehen für Wahrscheinlichkeiten unter der Hälfte und positive Logits für Wahrscheinlichkeiten über der Hälfte.

Die umgekehrte Form der Logitfunktion wird auch als [logistische Funktion](#) bezeichnet, die aufgrund ihrer charakteristischen *S*-Form manchmal einfach als [Sigmoidfunktion](#) abgekürzt wird. Sie ermöglicht es uns, von Logits auf Wahrscheinlichkeiten umzuschalten.

$$\pi = \text{logit}^{-1}(\eta) = \frac{e^\eta}{1 + e^\eta} = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k)}}$$

Die logistische Funktion für das Intervall $[-6, 6]$ ist unten dargestellt. Für Werte von η im Bereich von $-\infty$ bis ∞ liegt π im Bereich von 0 bis 1.

► Show code cell source



Das logistische Regressionsmodell

Die Logit-Funktion bildet Wahrscheinlichkeiten auf Werte über den gesamten realen Zahlenbereich ab. Die Wahrscheinlichkeit, dass ein Ereignis/Ergebnis/Erfolg wahr ist ($y = 1$), wenn die Menge der unabhängigen Variablen x_i , unsere Daten, gegeben ist, wird also geschrieben als

$$\text{logit}(P(y=1|x_i)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k,$$

Der Einfachheit halber drücken wir das Inverse der obigen Funktion wie folgt aus

$$\phi(\eta) = \frac{1}{1 + e^{-\eta}},$$

wobei η die Linearkombination von Koeffizienten (β_i) und unabhängiger Variablen (x_i) ist, berechnet als $\eta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$.

Die Parameter (β_i) des Logit-Modells werden nach der [Methode der maximalen Wahrscheinlichkeit](#) geschätzt. Es gibt jedoch keine geschlossene Lösung, so dass die Maximum-Likelihood-Schätzungen u. a. durch iterative Algorithmen wie [Newton-Raphson](#), iterativ neu gewichtete kleinste Quadrate oder [Gradientenverfahren](#) ermittelt werden.

Das Ergebnis der Sigmoidfunktion wird als die Wahrscheinlichkeit interpretiert, dass eine bestimmte Beobachtung zur Klasse 1 gehört. Sie wird geschrieben als $\phi(\eta) = P(y=1|x_i, \beta_i)$, die Erfolgswahrscheinlichkeit ($y=1$) bei den durch die Koeffizienten β_i parametrisierten Prädiktorvariablen x_i . Wenn wir beispielsweise für eine bestimmte Beobachtung $\phi(\eta) = 0,65$ berechnen, bedeutet dies, dass die Wahrscheinlichkeit, dass diese Beobachtung zur Klasse 1 gehört, 65% beträgt. In ähnlicher Weise wird die Wahrscheinlichkeit, dass diese Beobachtung zu Klasse 2 gehört, berechnet als $\phi(\eta) = P(y=0|x_i, \beta_i) = 1 - P(y=1|x_i, \beta_i) = 1 - 0,65 = 0,35$ oder 35 %. Für die Klassenzuordnung wird die vorhergesagte Wahrscheinlichkeit dann über eine Einheitssprungfunktion in ein binäres Ergebnis umgewandelt:

$$\hat{y} = \begin{cases} 1, & \text{wenn } \phi(\eta) \geq 0,5 \\ 0, & \text{sonst} \end{cases}$$

Logistische Regression in Python - ein Beispiel

Die logistische Regressionsanalyse gehört zur Klasse der [verallgemeinerten linearen Modelle](#). In Python werden verallgemeinerte lineare Modelle mit der Funktion `GLM()` verarbeitet. Die Funktion wird als `GLM(y=Antwort, X=Prädiktor, family=sm.families.Binomial())` geschrieben. Bitte beachten Sie, dass `logit` die Vorgabe für binomial ist; wir müssen es also nicht explizit eingeben.

Die Funktion `GLM()` gibt ein Modellobjekt zurück, auf das wir Extraktormethoden wie `summary()`, `fitted()` oder `predict()` anwenden können.

Einführung und explorative Datenanalyse

Dieses Beispiel ist inspiriert von der Arbeit von [James B. Elsner](#) und seinen Kollegen ([Elsner et al. 1996](#) und [Kimberlain und Elsner 1998](#)), die an der **Klassifizierung der nordatlantischen Hurrikane** auf der Grundlage von Entstehungs- und Entwicklungsmechanismen gearbeitet haben. Die Klassifizierung ergibt drei verschiedene Gruppen: tropische Wirbelstürme, Wirbelstürme unter baroklinem Einfluss und Wirbelstürme mit baroklinem Ursprung. Der Begriff "baroklin" bezieht sich auf die Tatsache, dass diese Hurrikane von Störungen der äußeren Tropen beeinflusst werden oder sogar in den äußeren Tropen entstehen. Die stärkeren tropischen Wirbelstürme entwickeln sich weiter südlich und treten hauptsächlich im August und September auf. Die schwächeren außertropischen Wirbelstürme treten während einer längeren Saison auf. Der ursprüngliche Datensatz zur objektiven Hurrikan-Klassifizierung kann [hier](#) abgerufen werden. Die Analyse von James B. Elsner kann [hier](#) eingesehen werden.

Ziel der Übung ist es, ein Modell zu erstellen, das die Gruppenzugehörigkeit eines Hurrikans, entweder tropisch oder ausser tropisch, auf der Grundlage des Breitengrades seiner Entstehung vorhersagt.

Wir beginnen die Analyse mit dem Laden des Datensatzes. Laden Sie die Daten auf Ihren Computer [herunter](#) und öffnen Sie die Datei. Bitte beachten Sie, dass es sich bei der Datei um ein *Excel*-Tabelle handelt. Um mit *Excel*-Tabellen umgehen zu können benutzen wir `read_excel()` aus dem [Pandas](#) Paket.

```
hurricanes = pd.read_excel("../data/hurricanes.xlsx", index_col=0)
hurricanes.head(5)
```

	Number	Name	Year	Type	FirstLat	FirstLon	MaxLat	MaxLon	L
RowNames									
1	430	NOTNAMED	1944	1	30.2	-76.1	32.1	-74.8	
2	432	NOTNAMED	1944	0	25.6	-74.9	31.0	-78.1	
3	433	NOTNAMED	1944	0	14.2	-65.2	16.6	-72.2	

	Number	Name	Year	Type	FirstLat	FirstLon	MaxLat	MaxLon	L
RowNames									
4	436	NOTNAMED	1944	0	20.8	-58.0	26.3	-72.3	
5	437	NOTNAMED	1944	0	20.0	-84.2	20.6	-84.9	

Zunächst untersuchen wir die Struktur des Datensatzes, indem wir die Methode `info()` anwenden.

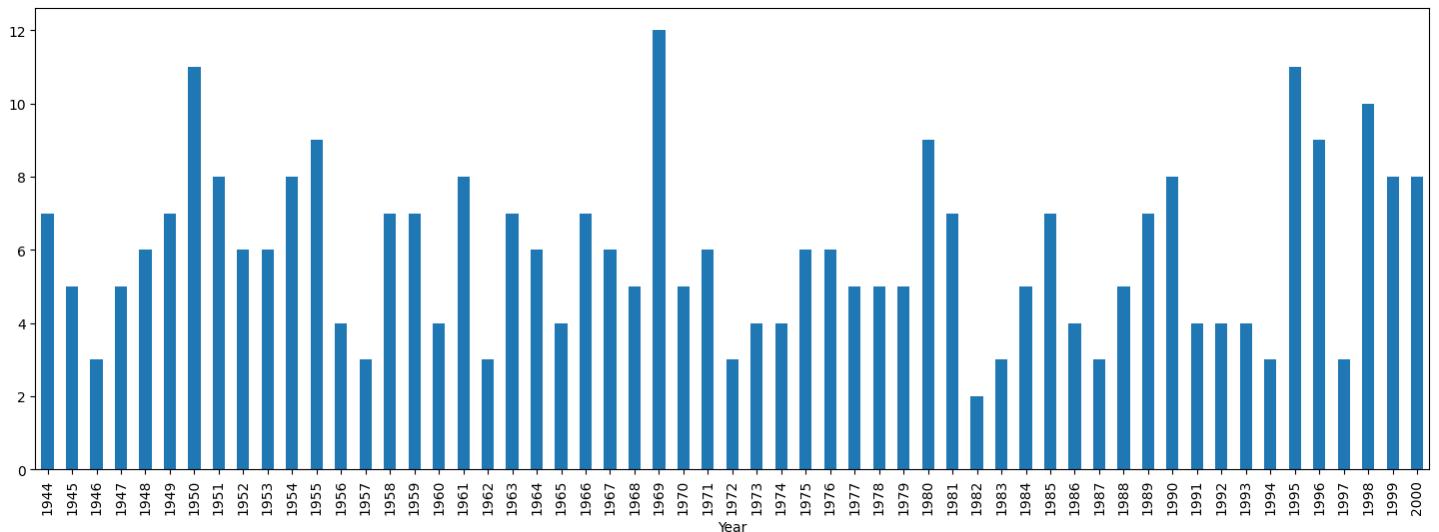
```
hurricanes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 337 entries, 1 to 337
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Number      337 non-null    int64  
 1   Name        337 non-null    object  
 2   Year        337 non-null    int64  
 3   Type        337 non-null    int64  
 4   FirstLat    337 non-null    float64 
 5   FirstLon    337 non-null    float64 
 6   MaxLat     337 non-null    float64 
 7   MaxLon     337 non-null    float64 
 8   LastLat    337 non-null    float64 
 9   LastLon    337 non-null    float64 
 10  MaxInt     337 non-null    int64  
dtypes: float64(6), int64(4), object(1)
memory usage: 31.6+ KB
```

Der Datensatz besteht aus 337 Beobachtungen und 12 Variablen. Wir interessieren uns in erster Linie für die Variable `Type`, die unsere Antwortvariable ist, und für die Variable `FirstLat`, die dem Breitengrad der Entstehung entspricht und somit unsere Prädiktorvariable ist. Um jedoch ein Gefühl für den Datensatz zu bekommen, stellen wir die Anzahl der Hurrikane für jedes Jahr als Balkendiagramm mit dem `matplotlib`-Paket dar.

```
hurricanes["Year"].value_counts().sort_index().plot.bar(figsize=(18, 6))
```

```
<Axes: xlabel='Year'>
```

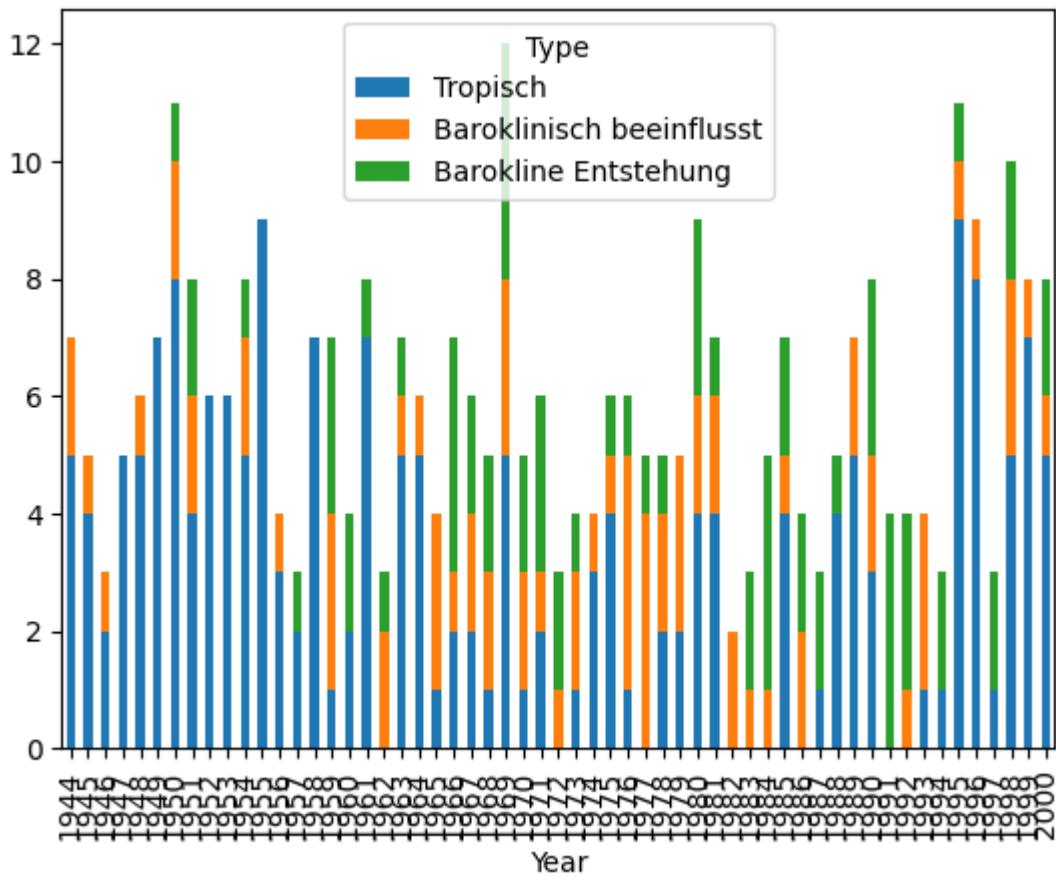


Das ist eine schöne Darstellung. Indem wir den Datensatz umformen können wir die Balken zusätzlich entsprechend der Variable `Type` einfärben.

```
hurricane_types = (
    hurricanes.groupby("Type")["Year"].value_counts().unstack().transpose()
)
hurricane_types = hurricane_types.rename(
    columns={0: "Tropisch", 1: "Baroklinisch beeinflusst", 3: "Barokline Entstehung"}
)
```

```
hurricane_types.plot.bar(stacked=True)
```

```
<Axes: xlabel='Year'>
```



Für eine numerische Darstellung der Hurricane-Klassen verwenden wir die Methode `sum()`.

```
print(
    f"Klasse 0 (tropische Wirbelstürme) Hurrikans: {hurricane_types['Tropisch'].sum()
)
print(
    f"Klasse 1 (barokline Einflüsse) Hurrikans: {hurricane_types['Baroklinisch beeinflusst'].sum()
)
print(
    f"Klasse 3 (barokline Auslösung) Hurrikans: {hurricane_types['Barokline Entstehung'].sum()
)
```

```
Klasse 0 (tropische Wirbelstürme) Hurrikans: 187.0
Klasse 1 (barokline Einflüsse) Hurrikans: 77.0
Klasse 3 (barokline Auslösung) Hurrikans: 73.0
```

Für die **Klasse 0, tropische Wirbelstürme**, gibt es 187 Beobachtungen, für die **Klasse 1, barokline Einflüsse**, gibt es 77 Beobachtungen und für die **Klasse 3, barokline Auslösung**, gibt es 73

Beobachtungen. Bei der logistischen Regression haben wir es mit dichotomen Daten zu tun; der Einfachheit halber kodieren wir die Klassen neu und weisen der Klasse 1 und der Klasse 3, die beide von den äußeren Tropen beeinflusst werden, die Bezeichnung **ausser tropisch** zu.

```
hurricanes["Origins"] = hurricanes["Type"].replace(  
    {0: "tropisch", 1: "ausser tropisch", 3: "ausser tropisch"}  
)
```

Wir haben nun eine binäre Antwortvariable mit zwei Klassen: Tropische und ausser tropische Wirbelstürmen.

```
hurricanes["Origins"].value_counts()
```

```
Origins  
tropisch      187  
ausser tropisch 150  
Name: count, dtype: int64
```

Um diesen Teil der explorativen Datenanalyse abzuschließen, stellen wir die Daten auf einer interaktiven Karte dar, indem wir das **folium**-Paket verwenden.

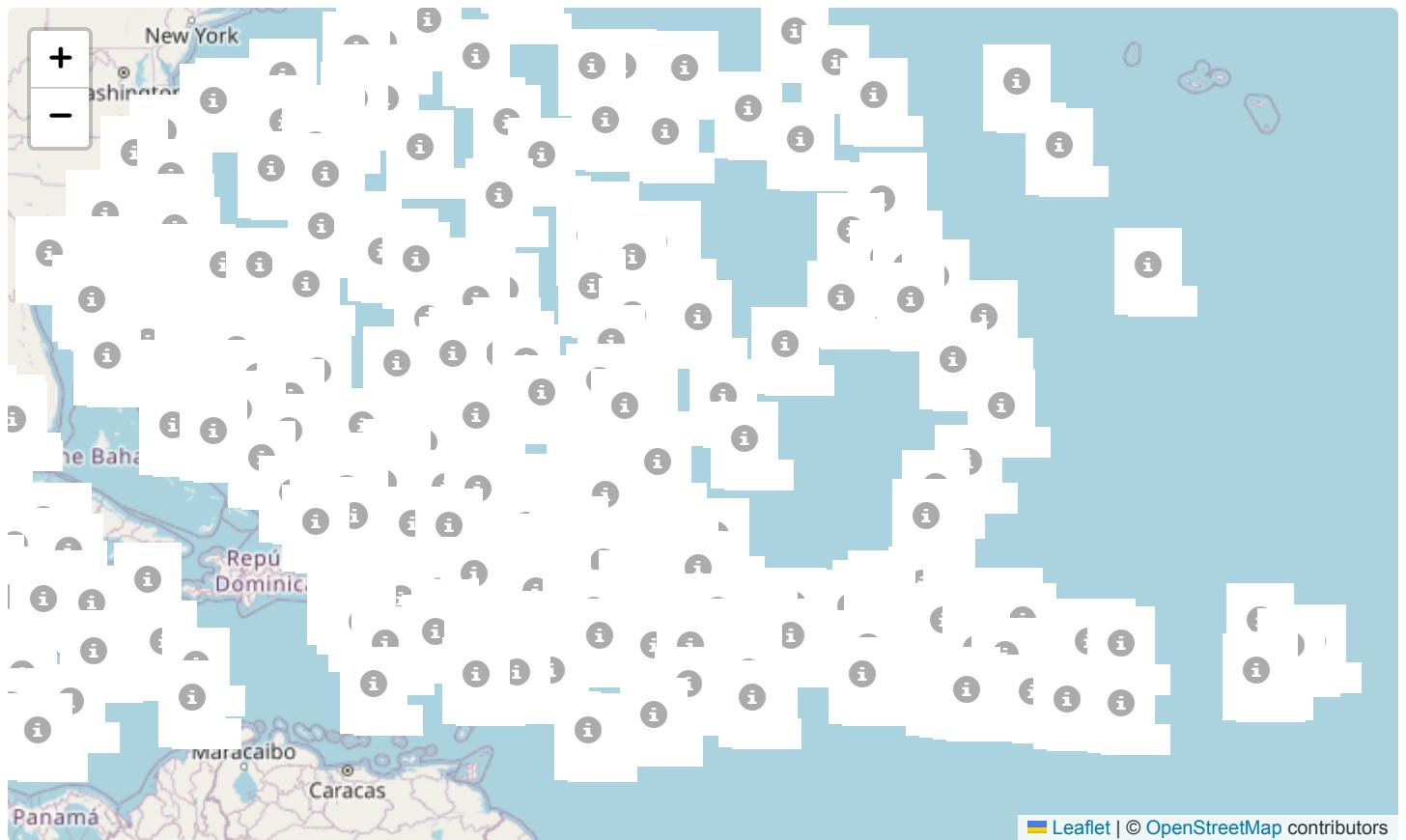
```
hurricanes.head()
```

	Number	Name	Year	Type	FirstLat	FirstLon	MaxLat	MaxLon	L
RowNames									
1	430	NOTNAMED	1944	1	30.2	-76.1	32.1	-74.8	
2	432	NOTNAMED	1944	0	25.6	-74.9	31.0	-78.1	
3	433	NOTNAMED	1944	0	14.2	-65.2	16.6	-72.2	
4	436	NOTNAMED	1944	0	20.8	-58.0	26.3	-72.3	
5	437	NOTNAMED	1944	0	20.0	-84.2	20.6	-84.9	

```

# Create the map
hurricane_map = folium.Map(location=[25, -50], zoom_start=4)
# add marker one by one on the map
colors = {"aussertropisch": "green", "tropisch": "red"}
for row in hurricanes.itertuples():
    m = folium.Marker(
        location=[row.FirstLat, row.FirstLon], icon=folium.Icon(colors[row.Origins])
    )
    m.add_to(hurricane_map)
# Show the map
hurricane_map

```



Logistische Regression: Modellanpassung

Erinnern Sie sich an das Ziel der Übung: Wir wollen ein Modell erstellen, das die Gruppenzugehörigkeit eines Hurrikans vorhersagt, entweder tropisch (0) oder aussertropisch (1), basierend auf dem Breitengrad der Entstehung des Hurrikans. Die Antwortvariable ist die binäre Variable `Origins` und die Prädiktorvariable ist `FirstLat`. Wir erstellen ein Logit-Modell, indem wir die Funktion `GLM()` anwenden. Für das logistische Regressionsmodell geben wir `family = sm.families.Binomial()` an. Statsmodel erwartet numerischen Inputvariablen, deshalb wandeln wir `tropisch` zu 0 und `aussertropisch` zu 1 um.

```
X = hurricanes["FirstLat"].values
X = sm.add_constant(X)
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values
log_model = sm.GLM(y, X, family=sm.families.Binomial()).fit()
```

```
/tmp/ipykernel_21065/101292342.py:3: FutureWarning: Downcasting behavior in `replace`  
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values
```

Wir verwenden die Extraktormethode `summary()`, um die Modelleigenschaften zu überprüfen.

```
log_model.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	y	No. Observations:	337			
Model:	GLM	Df Residuals:	335			
Model Family:	Binomial	Df Model:	1			
Link Function:	Logit	Scale:	1.0000			
Method:	IRLS	Log-Likelihood:	-116.02			
Date:	Tue, 22 Oct 2024	Deviance:	232.03			
Time:	21:36:37	Pearson chi2:	430.			
No. Iterations:	6	Pseudo R-squ. (CS):	0.4963			
Covariance Type:	nonrobust					
	coef	std err	z	P> z	[0.025	0.975]
const	-9.0826	0.961	-9.446	0.000	-10.967	-7.198
x1	0.3728	0.039	9.447	0.000	0.295	0.450

Schauen wir uns die Koeffizienten genauer an:

```
log_model.params
```

```
array([-9.08263355,  0.37282953])
```

Beachten Sie, dass die Ausgabe des logistischen Modells auf der Link-Skala (*logit*) erfolgt; die numerische Ausgabe des Modells entspricht also den **Log-Odds!** Der Einfachheit halber schreiben wir das logistische Regressionsmodell mit dem berechneten Achsenabschnitt und Koeffizienten auf.

$$\phi(\eta) = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} = \frac{1}{1 + e^{-(9,0826335 + 0,3728295x)}}$$

Konzentrieren wir uns zunächst auf den Koeffizienten des Achsenabschnitts. Der Koeffizient des Achsenabschnitts entspricht der logarithmischen Wahrscheinlichkeit der Beobachtung eines Hurrikans auf dem Breitengrad Null, der als Äquator bekannt ist. Um die Wahrscheinlichkeit der Beobachtung eines ausertropischen Hurrikans bei einem Breitengrad von Null zu berechnen, exponentizieren wir die log-odds $e^{-9,0826335} \approx 1,1362198 \times 10^{-4}$.

Dies ist eine sehr niedrige Zahl! Und sie macht durchaus Sinn, da es sehr unwahrscheinlich ist, einen ausertropischen Hurrikan am Äquator zu beobachten.

Der Koeffizient der Variable für die Bildungsbreite (`FirstLat`) hat einen numerischen Wert von 0,3728295. Das positive Vorzeichen dieses Wertes zeigt an, dass die Wahrscheinlichkeit, einen außertropischen Wirbelsturm zu beobachten, mit dem Breitengrad zunimmt. Die Größe des Koeffizienten bedeutet, dass die Log-Odds für jedes Grad Zunahme der geografischen Breite im Durchschnitt um konstant 0,3728295 Einheiten zunimmt. Wenn man den Koeffizientenwertes als Exponenten nimmt, erhält man das Odds Verhältnis.

```
math.exp(log_model.params[1])
```

```
1.451836826600862
```

```
log_model.params[1]
```

```
np.float64(0.3728295317136198)
```

Somit erhöht sich das Chancenverhältnis für jedes Grad Zunahme der Bildungsbreite im Durchschnitt um einen konstanten Faktor von 1,4518368 (oder 45%). Die Interpretation gilt nur für den Bereich

der Breitengrade in den Daten und ist für Breitengrade außerhalb des Bereichs, in dem Hurrikane auftreten, physikalisch bedeutungslos.

Die obige Koeffiziententabelle enthält einen Standardfehler und den p -Wert. Je kleiner der p -Wert ist, desto geringer ist die Unterstützung für die Nullhypothese angesichts der Daten und des Modells. Zur Erinnerung: Die Nullhypothese besagt, dass der Koeffizient gleich 0 ist, mit anderen Worten, die Nullhypothese besagt, dass es keinen Zusammenhang zwischen dem Auftreten eines nichttropischen Hurrikans und der Entstehungsbreite gibt. Der sehr kleine p -Wert stützt die Nullhypothese nicht und erlaubt uns, das Modell zu akzeptieren.

Wir haben gelernt, dass jede Punktschätzung von einem Konfidenzniveau bezüglich dieser Punktschätzung begleitet sein sollte. Daher konstruieren wir ein 95%-Konfidenzintervall für den geschätzten Modellkoeffizienten. Wir exponentizieren das Ergebnis, um das Odds-Ratio zu erhalten, eine Größe, die leichter zu interpretieren ist als die Log-Odds.

```
lci = log_model.conf_int()[1][0]
uci = log_model.conf_int()[1][1]
```

```
print(f"2,5%: {round(lci,4)}")
print(f"97,5%: {round(uci,4)}")
```

```
2,5%: 0.2955
97,5%: 0.4502
```

```
print(f"2,5%: {round(math.exp(lci),4)}")
print(f"97,5%: {round(math.exp(uci),4)}")
```

```
2,5%: 1.3438
97,5%: 1.5686
```

Wir können also mit 95%iger Sicherheit sagen, dass für jedes Grad Zunahme der geographischen Breite die Beobachtung eines aussertropischen Hurrikans im Durchschnitt zwischen 34 und 57% wahrscheinlicher wird. Macht Sinn, oder? In anderen Worten je weiter nördlich Wirbestürme entstehen, umso eher ist dieser aussertropischen Einflüssen unterworfen.

Logistische Regression: Modellvorhersage

Im vorangegangenen Abschnitt haben wir ein logistisches Regressionsmodell für die Beziehung zwischen der Entstehungsbreite und der Art des Hurrikans (tropisch/aussertropisch) erstellt. In diesem Abschnitt wenden wir dieses Modell an, um Vorhersagen über die Wahrscheinlichkeit der Beobachtung eines aussertropischen Hurrikans in Abhängigkeit von der Entstehungsbreite zu machen.

Um die Wahrscheinlichkeit vorherzusagen, dass ein zufällig ausgewählter Hurrikan, der sich auf einem bestimmten Breitengrad bildet, vom Typ aussertropisch ist, wenden wir die Funktion `predict()` an. Berechnen wir die Wahrscheinlichkeit, einen aussertropischen Hurrikan zu beobachten, der sich bei folgenden Breitengraden gebildet hat: 10° , $23,5^\circ$ und 30°

```
pred = [10.0, 23.5, 30.0] #
pred = sm.add_constant(pred)
predictions = log_model.get_prediction(exog=pred).summary_frame()
predictions
```

	mean	mean_se	mean_ci_lower	mean_ci_upper
0	0.004705	0.002701	0.001524	0.01443
1	0.420398	0.040645	0.343423	0.50145
2	0.891122	0.028045	0.822841	0.93516

Die Ergebnisse machen durchaus Sinn. Mit zunehmender geografischer Breite nimmt die Wahrscheinlichkeit, die Bildung eines nichttropischen Hurrikans zu beobachten, deutlich zu. Bei einem Breitengrad von 10° beträgt die Wahrscheinlichkeit der Beobachtung eines außertropischen Hurrikans 0,005, während sie bei einem Breitengrad von 30° 0,891 beträgt.

Zum besseren Verständnis wollen wir die Wahrscheinlichkeit der Beobachtung eines nichttropischen Hurrikans, der sich bei einem Breitengrad von $23,5^\circ$ von Hand. Erinnern wir uns an die Gleichung für das Regressionsmodell von oben:

$$\phi(\eta) = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_1)}}$$

$$= \frac{1}{1+e^{-(9,0826335+0,3728295 \times 23,5)}}$$

$$= \frac{1}{1+e^{-(-0,3211396)}}$$

$$= \frac{1}{2,378698} = 0,4203981$$

Zur Veranschaulichung unserer Ergebnisse erstellen wir ein Diagramm der Vorhersagen über eine Reihe von Breitengraden. Dazu erstellen wir zunächst einen Vektor der Breitengrade, `lats`. Wir geben eine Schrittweite von 0,1 Grad an, damit die resultierende Vorhersagekurve glatt ist. Wir wenden die Funktion `predict()` an und verwenden die Methode `get_prediction()`, um den **Standardfehler** der Vorhersage zu erhalten. Die Kenntnis des Standardfehlers ermöglicht die Berechnung der **Fehlermarge** und damit des **Konfidenzintervalls**. Mit der Methode `summary_frame()` können wir die Werte in einem Dataframe ausgeben und darauf zugreifen. Schließlich zeichnen wir die Datenpunkte (Beobachtungen) bei 0 und 1 ein.

```

# Berechne Vorhersagen für lats
X = hurricanes["FirstLat"].values
X = sm.add_constant(X)
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values

lats_grid = np.linspace(min(hurricanes["FirstLat"]), max(hurricanes["FirstLat"]))

log_model = sm.GLM(y, X, family=sm.families.Binomial()).fit()
predictions = log_model.get_prediction(exog=sm.add_constant(lats_grid)).summary_frame()

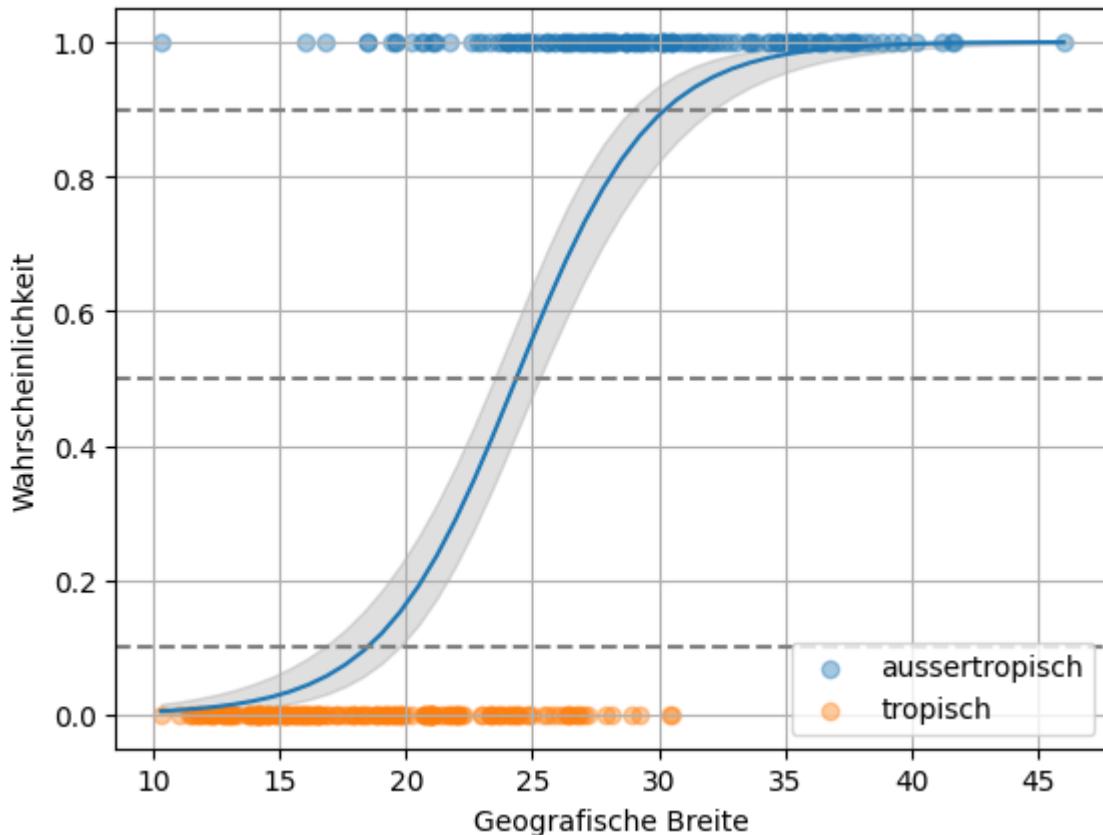
# Ploting
fig, ax = plt.subplots()
ax.plot(lats_grid, predictions["mean"])
ax.fill_between(
    lats_grid,
    predictions["mean_ci_lower"],
    predictions["mean_ci_upper"],
    color="grey",
    alpha=0.25,
)
outer = hurricanes.loc[hurricanes.Origins == "aussertropisch", "FirstLat"].values
ax.scatter(outer, np.repeat(1, repeats=len(outer)), label="aussertropisch", alpha=0.5)
tropical = hurricanes.loc[hurricanes.Origins == "tropisch", "FirstLat"].values
ax.scatter(tropical, np.repeat(0, repeats=len(tropical)), label="tropisch", alpha=0.5)

ax.grid()
for _y in [0.1, 0.5, 0.9]:
    ax.axhline(y=_y, color="gray", linestyle="dashed")
ax.set_xlabel("Geografische Breite")
ax.set_ylabel("Wahrscheinlichkeit")
ax.legend()

```

```
/tmp/ipykernel_21065/345546926.py:4: FutureWarning: Downcasting behavior in `replace`  
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values
```

```
<matplotlib.legend.Legend at 0x7b3ce644e980>
```



Die Beobachtungen tropischer und aussertropischer Hurrikane sind entlang der Linien $y = 0$ bzw. $y = 1$ Linien dargestellt. Das graue Band ist das 95%ige punktweise Konfidenzintervall.

Die Modellvorhersagen zeigen, dass die Wahrscheinlichkeit eines nichttropischen Hurrikans bei Breitengraden südlich von $20^\circ N$ weniger als 10% beträgt. Bis zum Breitengrad $26^\circ N$ übersteigt die Wahrscheinlichkeit jedoch 50% und bis zum Breitengrad $33^\circ N$ übersteigt die Wahrscheinlichkeit 90%. Das Chancenverhältnis ist konstant, aber die Wahrscheinlichkeit ist eine nichtlineare Funktion des Breitengrads.

Übungsaufgaben

Logistische Regression - Grundbegriffe

1. Was sind Odds, Log-Odds (*Logit*)? Auf welchen Wertebereich bilden sie ab?
 2. Vergleichen Sie lineare und einfache logistische Regression. Worin unterscheiden sie sich?
 3. Was ist die logistische Funktion? Wie hängt Sie mit der *logit*-Funktion zusammen?
-

Lösungen

Logistische Funktion

1. Zeigen Sie dass die logistische Funktion $logit^{-1}(\eta) = \frac{e^\eta}{1+e^\eta}$ die inverse Funktion zur *logit*-Funktion $\eta = logit(\pi) = \log\left(\frac{\pi}{1-\pi}\right)$ ist.
-

Lösungen

Odds und Log-Odds

Zeigen Sie, dass wenn die Odds und das einfache (binomiale) logistische Regressionsmodell (π) gegeben sind durch:

$$\text{Odds} = \frac{\pi}{1 - \pi}$$

$$\pi = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1)}}$$

die Log-Odds gleich

$$\text{Log-Odds} = \log\left(\frac{\pi}{1 - \pi}\right) = \beta_0 + \beta_1 x_1 \text{ sind.}$$

Lösungen

Einfaches logistisches Regressionsmodell

1. Erstellen Sie ein einfaches logistische Regressionsmodell für die folgenden Daten in Python und stellen Sie das logistische Modell graphisch dar.

```
x = [  
    29,  
    15,  
    33,  
    28,  
    39,  
    44,  
    31,  
    19,  
    9,  
    24,  
    32,  
    31,  
    37,  
    35,  
    8,  
    4,  
    11,  
    12,  
    33,  
    45,  
    20,  
    25,  
    27,  
    26,  
    29,  
]  
y = [0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1]
```

Lösungen

► Show code cell content

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
-----  
Working on the host: imarevic-pc
```

```
-----  
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

```
-----  
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

Entscheidungsbäume

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_squared_error,
    confusion_matrix,
    accuracy_score,
    ConfusionMatrixDisplay,
)
from sklearn.preprocessing import StandardScaler
import dtreeviz
```

In Kapitel 8 hatten wir die lineare Regression und die polynomiale Regression, als Verfahren zur Vorhersage kontinuierlicher Variablen kennen gelernt. Für die Vorhersage kategorialer Variablen wurde in Kapitel 9 die logistische Regression dargestellt. Im Folgenden werden wir ein Verfahren kennenlernen, das sowohl für Regressions- als auch Klassifikationsprobleme verwendet werden kann, die Entscheidungsbäume.

Entscheidungsbäume sind baumbasierte Verfahren, die sowohl für Regressions-, als auch Klassifikationsprobleme genutzt werden können. Hierbei wird der Vorhersageraum in eine bestimmte Anzahl Regionen **segmentiert** und kann somit als Baumstruktur visualisiert werden (s.303–314).

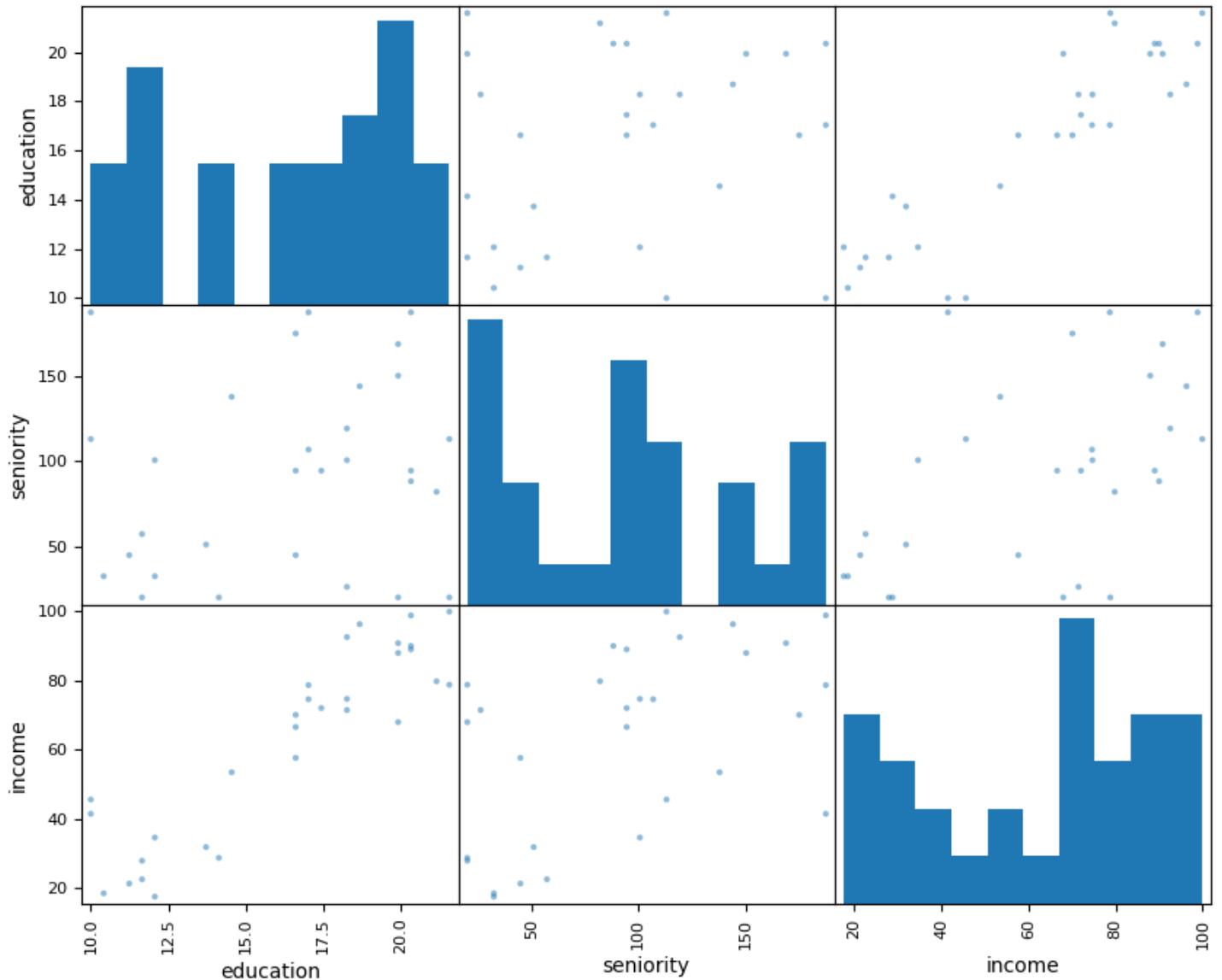
Regressions-Bäume

Um Entscheidungsbäume besser zu verstehen, wenden wir uns zunächst dem Fall der Regression zu und behandeln **Regressions-Bäume**. Als Beispiel verwenden wir einen fiktiven Datensatz indem Einkommensdaten gelistet sind. Wir werden versuchen das Einkommen (`income`) anhand der Variablen Anzahl Jahre in der Schule (`education`) und Berufserfahrung (`seniority`) vorherzusagen. Zunächst laden wir den Datensatz:

```
df = pd.read_csv("../data/income_data.csv")
```

Eine kurze Visualisierung der Daten zeigt, dass ein Zusammenhang zwischen `income` und den beiden Prädiktoren `education` und `seniority` besteht, wobei der Zusammenhang für `education` etwas stärker ist als für `seniority`:

```
pd.plotting.scatter_matrix(df, figsize=(10, 8))  
plt.show()
```



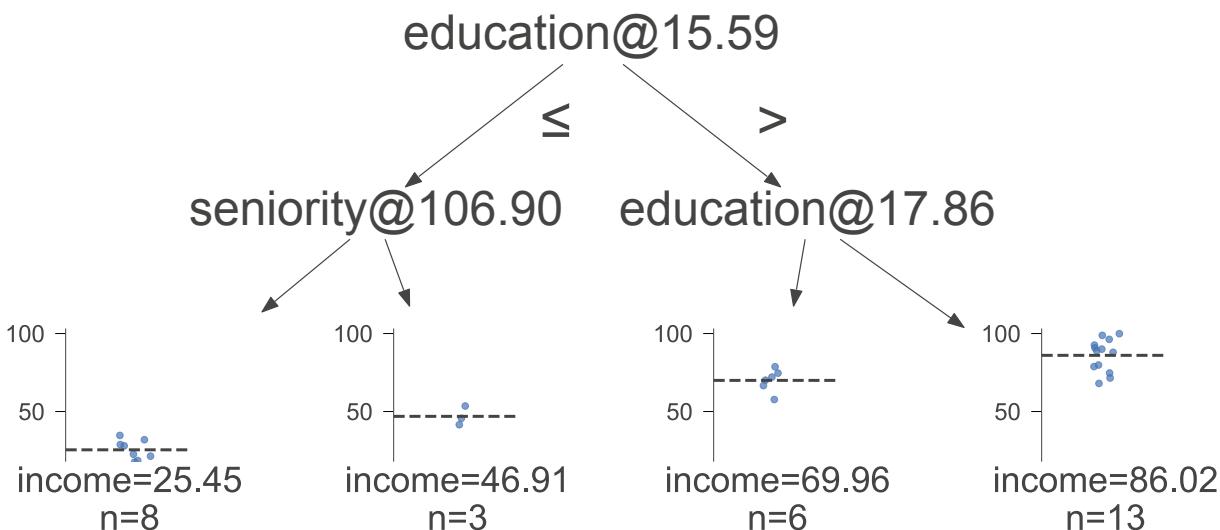
Wir greifen nun etwas voraus und fitten einen einfachen Entscheidungsbaum, unter Nutzung der Bibliothek `scikit-learn`, an die Daten und visualisieren den Baum anhand der Splits (hierfür verwenden wir die Bibliothek `dtreeviz`), die für die Variablen `education` und `seniority` im Entscheidungsbaum angewendet wurden. Hier zunächst die Visualisierung:

```

# fit a regression tree in order to create
# the regressor object
features = ["education", "seniority"]
target = "income"
X = df[features]
y = df[target]
reg = DecisionTreeRegressor(max_leaf_nodes=4, max_depth=2)
model = reg.fit(X, y)
# create viz model object
viz_model = dtreeviz.model(
    reg,
    X_train=df[features],
    y_train=df[target],
    feature_names=features,
    target_name=target,
)
# plot view
viz_model.view(fancy=False, scale=2)

```

/home/imarevic/Documents/teaching/SRH/content/statistik/statistik-env/lib/python3.10



In der erzeugten Visualisierung ist schön zu sehen, dass der Entscheidungsbaum zur Vorhersage des Einkommens bei einer Tiefe von 2 den ersten Split bei 15.59 Jahren in Bezug auf die Anzahl Jahre in der Schule einführt. Eine Ebene darunter wird dann auf Basis der selben Variablen bei 17.86 Jahren, sowie bei einer Seniority von 106.90 gesplittet. Der in jedem "Blatt" des Baumes angezeigte Wert gibt das mittlere Einkommen für die Beobachtungen an, die zu dem jeweiligen Blatt gehören.

Verallgemeinert lässt sich also festhalten, dass in einem Entscheidungsbaum die Ergebnisse eines Splits für Prädiktoren X_j mit $j = 1, 2, 3, \dots, n$ wie folgt definiert sind:

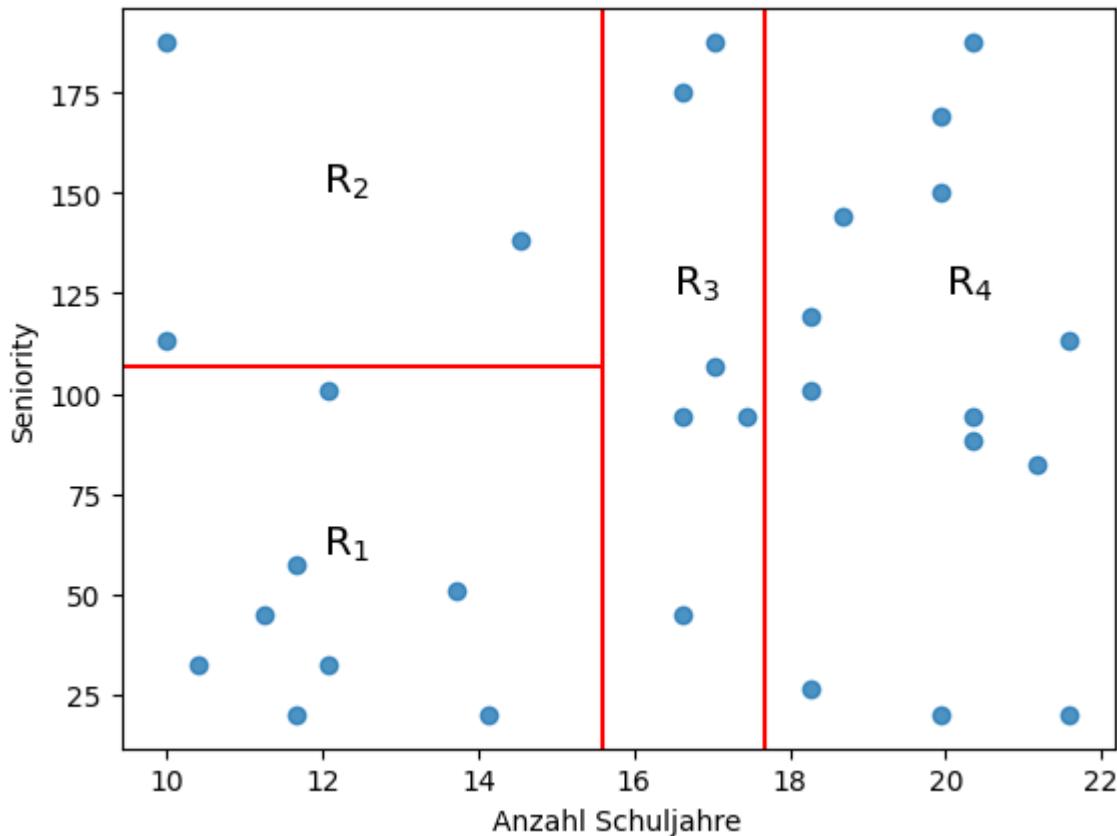
$$\text{LinkerZweig} : X_j < t_k$$

$$\text{RechterZweig} : X_j > t_k$$

wobei t_k den genauen Splitwert bezeichnet.

Würden wir die Daten, bei einer Baumtiefe von 2, in einem Streudiagramm visualisieren, dann ließen sich für jeden der Blätter im Baum eine **Region** einzeichnen. Diese werden anhand der **Splitwerte** bestimmt:

```
# plot regions in predictor space
params = {"mathtext.default": "regular"}
plt.rcParams.update(params)
plt.scatter(df["education"], df["seniority"], alpha=0.8)
plt.axvline(x=15.59, color="r", linestyle="-")
plt.axhline(y=106.9, xmax=0.481, color="r", linestyle="-")
plt.axvline(x=17.68, color="r", linestyle="-")
plt.text(x=12, y=60, s="$R_{1}$", fontsize="x-large")
plt.text(x=12, y=150, s="$R_{2}$", fontsize="x-large")
plt.text(x=16.5, y=125, s="$R_{3}$", fontsize="x-large")
plt.text(x=20, y=125, s="$R_{4}$", fontsize="x-large")
plt.xlabel("Anzahl Schuljahre")
plt.ylabel("Seniority")
plt.show()
```



Für jede der vier Regionen können wir den Mittelwert des Einkommens berechnen. Diese sind, wie oben beschrieben, in den Blättern des Baumes angeführt. Die vier Regionen (auch **Endknoten** oder **Blätter** genannt) R_1 - R_4 unterteilen den Prädiktorenraum nach folgenden Eigenschaften:

- $R_1 = \{X | education < 15.59, seniority < 106.90\} \rightarrow$ Personen mit weniger als 15.59 Schuljahren und einer Seniority von höchstens 106.9
- $R_2 = \{X | education < 15.59, seniority > 106.90\} \rightarrow$ Personen mit weniger als 15.59 Schuljahren und einer Seniority von mindestens 106.9
- $R_3 = \{X | 15.59 < education < 17.86\} \rightarrow$ Personen zwischen 15.59 und 17.86 Schuljahren
- $R_4 = \{X | education > 17.86\} \rightarrow$ Personen mit mehr als 17.86 Schuljahren

Anhand der Regionen und der Baumstruktur lassen sich Entscheidungsbäume sehr gut in Worten beschrieben und können somit bei komplexen Datensätzen leichter als klassische Regressionsmodelle interpretiert werden.

Die Hauptelemente eines Entscheidungsbaums lassen sich also wie folgt zusammenfassen:

Baumelement	Definition
Blatt/Endknoten	Unterste Ebene des Entscheidungsbaumes nachdem alle Splits vollzogen wurden.
Interne Knoten	Knoten, die zwischen dem Ursprung und den Blättern liegen. Entlang dieser Knoten wird gesplittet.
Zweige	Segmente des Baums, welche die Knoten verbinden.

Beim betrachten eines Entscheidungsbaumes wird schnell deutlich, dass theoretisch beliebig viele Regionen R_n erzeugt werden könnten. Es stellt sich also die Frage wie eine Entscheidungsbaum effizient und möglichst aussagekräftig auf gegebenen Daten erzeugt werden kann. Hierzu sind im Allgemeinen folgende 2 Schritte durchzuführen:

1. Wir teilen den Prädiktorenraum, also die Menge aller möglichen Werte für X_1, X_2, \dots, X_j in J distinkte und sich nicht überlappende Regionen R_1, R_2, \dots, R_J
2. Für jede Beobachtung, die in die Region R_j fällt, machen wir dieselbe Vorhersage anhand des Mittelwerts der Trainingsdaten der entsprechenden Region.

Wie wählen wir aber die Regionen R_1, \dots, R_J ?

Hierbei ist die Unterteilung in die obigen vier Regionen eine starke Vereinfachung. In der Praxis werden die Regionen so gewählt, dass die **Summe der Fehlerquadrate (SSE)** minimiert wird.

$$SSE = \sum_{j=1}^J \sum_{i \in \mathbb{R}} (y_i - \hat{y}_{R_j})^2$$

wobei \hat{y}_{R_j} den Mittelwert der Trainingsbeobachtungen in der j-ten Region angibt.

Es wird schnell deutlich, dass es in der Praxis keinen Sinn macht jede mögliche Partitionierung in Regionen durchzuführen um am Ende die beste Konfiguration zu wählen. Daher wird der Entscheidungsbaum in der Praxis mit einem *top-down greedy* Verfahren aufgebaut. Dieses Verfahren wird auch als **rekursives binäres Splitting** bezeichnet. Hierbei beginnt man oben am Baum und splittet immer in sukzessive 2 Regionen, wobei der Splitpunkt s immer so gewählt wird, dass der Split zur größtmöglichen Reduktion im SSE führt. Das Verfahren ist **greedy**, da bei jedem Split des Baumaufbaus immer nur der beste Split zu diesem bestimmten Zeitpunkt definiert wird. Es wird also

nicht berücksichtigt ob der Split zum Zeitpunkt t_1 ein guter Split für den finalen Baum war, der zu einem späteren Zeitpunkt t_5 entstehen könnte. Dieses Verfahren wird so lange wiederholt, bis ein Stopkriterium erfüllt ist (z.B. weniger als 5 Datenpunkte in einer Region).

Der **Top-Down Greedy Entscheidungsbaum Algorithmus** lässt sich also wie folgt formalisieren:

1. Starte mit dem gesamten Prädiktorraum und splitte diesen in 2 Regionen unter Selektion des Prädiktors X_j und dem Splitpunkt s , so dass der Raum in die beiden Regionen $\{X|X_j < s\}$ und $\{X|X_j \geq s\}$ unterteilt wird und zur größt möglichen Reduktion im SSE führt. Die Notation $\{X|X_j < s\}$ bedeutet in diesem Kontext, dass die Region des Prädiktorraums in dem X_j einen Wert kleiner s annimmt. Das bedeutet, wir berücksichtigen alle Prädiktoren X_1, \dots, X_p und alle möglichen Werte für den Splitpunkt s für jeden der Prädiktoren. Formal können wir notieren:

Für jedes j und s definieren wir das Paar an Halbebenen

$$R_1(j, s) = \{X|X_j < s\}$$

und

$$R_2(j, s) = \{X|X_j \geq s\}$$

und wir suchen Werte für j und s , welche folgende Gleichung minimieren:

$$\sum_{i:x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2$$

2. Wiederhole dieses Vorgehen für den nächsten Split, wobei der SSE nur für diesen Split zum Zeitpunkt t_j minimiert wird.
3. Der Prozess endet sobald ein Stopkriterium K erreicht ist, wobei K die Anzahl Datenpunkte in einer Region sein kann. Zum Beispiel $K = 5$.
4. Sobald die Regionen R_1, \dots, R_J erstellt wurden, wird der Mittelwert zu jeder Beobachtung des Testdatensatzes, die in diese Region fällt, ausgegeben.

Regressionsbäume in Python

Lassen Sie uns zur Veranschaulichung den soeben beschriebenen Algorithmus in Python implementieren:

Wir werden den Entscheidungsbaum Regressor als Python Klasse implementieren. Details zu objektorientierter Programmierung sind hier nicht entscheidend und können getrost ignoriert werden. Wichtig ist nur, dass wir die Hauptmethoden der Klasse verstehen. Bevor wir mit der Implementierung starten, benötigen wir ein paar Standardfunktionen, die wir der Klasse hinzufügen:

- `__init__()`: wird bei der Instantierung des `MyDecisionTreeRegressor()` Objekts ausgeführt.
- `calculate_sse()` zur Berechnung des SSE
- `get_overall_sse()` zur Berechnung des SSE für beide Hälften des Splits. Dieser overall SSE wird mimiert.

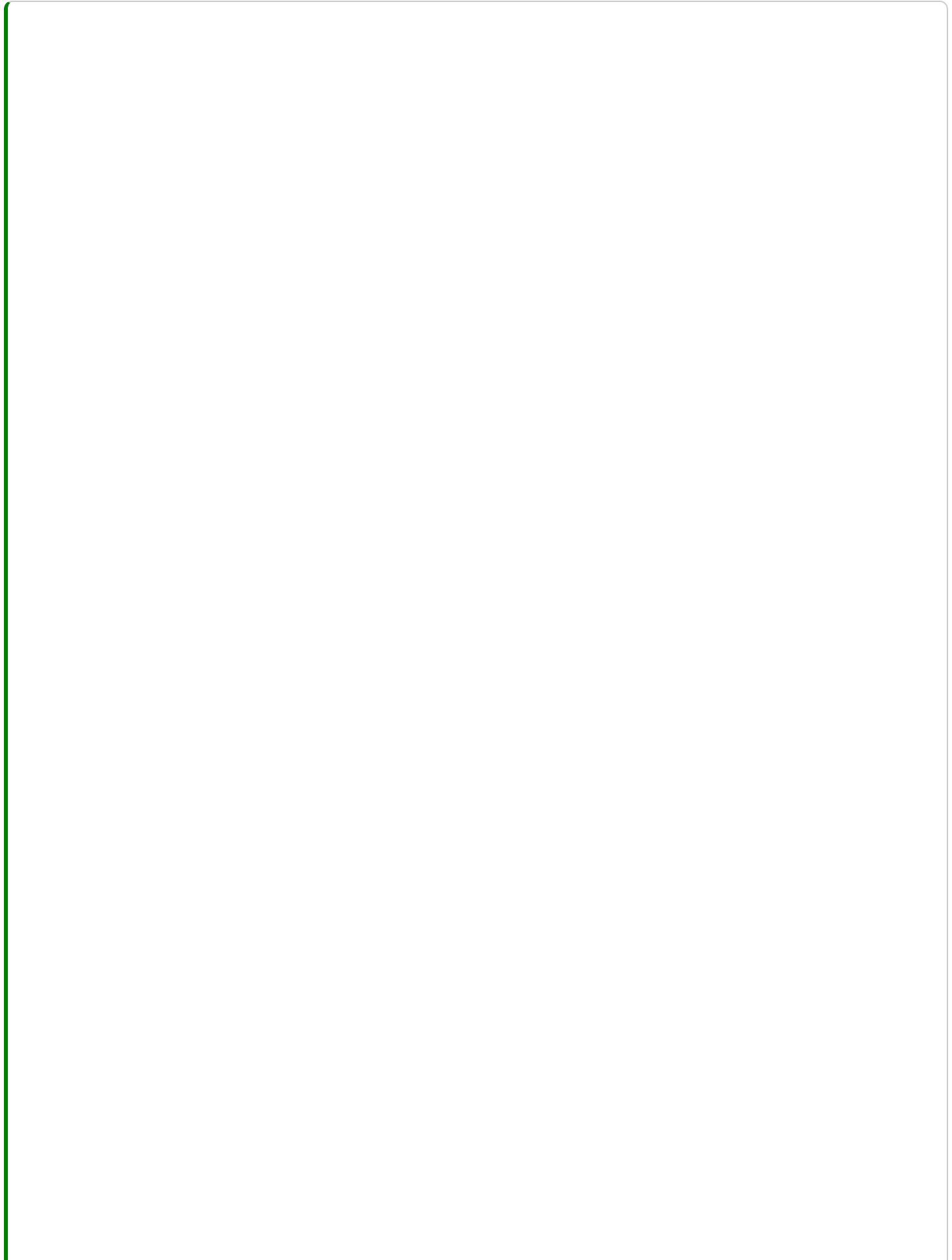
```
import numpy as np

class MyDecisionTreeRegressor:
    def __init__(self, max_depth=None, max_n_leaf_observations=1):
        self.max_depth = max_depth
        self.max_n_leaf_observations = max_n_leaf_observations
        self.tree = None

    def calculate_sse(self, y):
        return np.sum((y - np.mean(y)) ** 2)

    def get_overall_sse(self, left_y, right_y):
        left_sse = calculate_sse(left_y)
        right_sse = calculate_sse(right_y)
        return left_sse + right_sse
```

Nun erweitern wir die Implementierung um die `fit()` Methode, welche vom Nutzer verwendet werden kann und die interne Methode `fit_tree()`, in welcher die eigentliche rekursive Implementierung steckt. In dieser Methode ist das Ziel, den besten Split zu bestimmen und dieses Vorgehen rekursiv anzuwenden, bis der beste Split gefunden wurde.



```

import numpy as np

class MyDecisionTreeRegressor:
    def __init__(self, max_depth=None, max_n_leaf_observations=1):
        self.max_depth = max_depth
        self.max_n_leaf_observation = max_n_leaf_observation
        self.tree = None

    def calculate_sse(self, y):
        return np.sum((y - np.mean(y)) ** 2)

    def get_overall_sse(self, left_y, right_y):
        left_sse = calculate_sse(left_y)
        right_sse = calculate_sse(right_y)
        return left_sse + right_sse

    def fit_tree(self, X, y, depth):
        n_samples, n_features = X.shape
        unique_y = np.unique(y)

        # Stopkriterien
        if len(unique_y) == self.max_n_leaf_observation:
            return {"value": unique_y[0]}
        if depth == self.max_depth:
            return {"value": np.mean(y)}

        # Greedy algorithm um den beste Split zu finden
        best_split = None
        best_sse = float("inf")

        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = X[:, feature] > threshold

                if np.any(left_indices) and np.any(right_indices):
                    left_y = y[left_indices]
                    right_y = y[right_indices]

                    sse = self.get_overall_sse(left_y, right_y)

                    if sse < best_sse:
                        best_sse = sse
                        best_split = {
                            "feature": feature,
                            "threshold": threshold,
                            "left": (X[left_indices], y[left_indices]),
                            "right": (X[right_indices], y[right_indices]),
                        }

        if best_split is None:
            return {"value": np.mean(y)}
        else:
            return best_split

```

```

        return {"value": np.mean(y)}

    # rekursiver Aufruf der fit Methode
    left_tree = fit_tree(*best_split["left"], depth + 1)
    right_tree = fit_tree(*best_split["right"], depth + 1)

    return {
        "feature": best_split["feature"],
        "threshold": best_split["threshold"],
        "left": left_tree,
        "right": right_tree,
    }

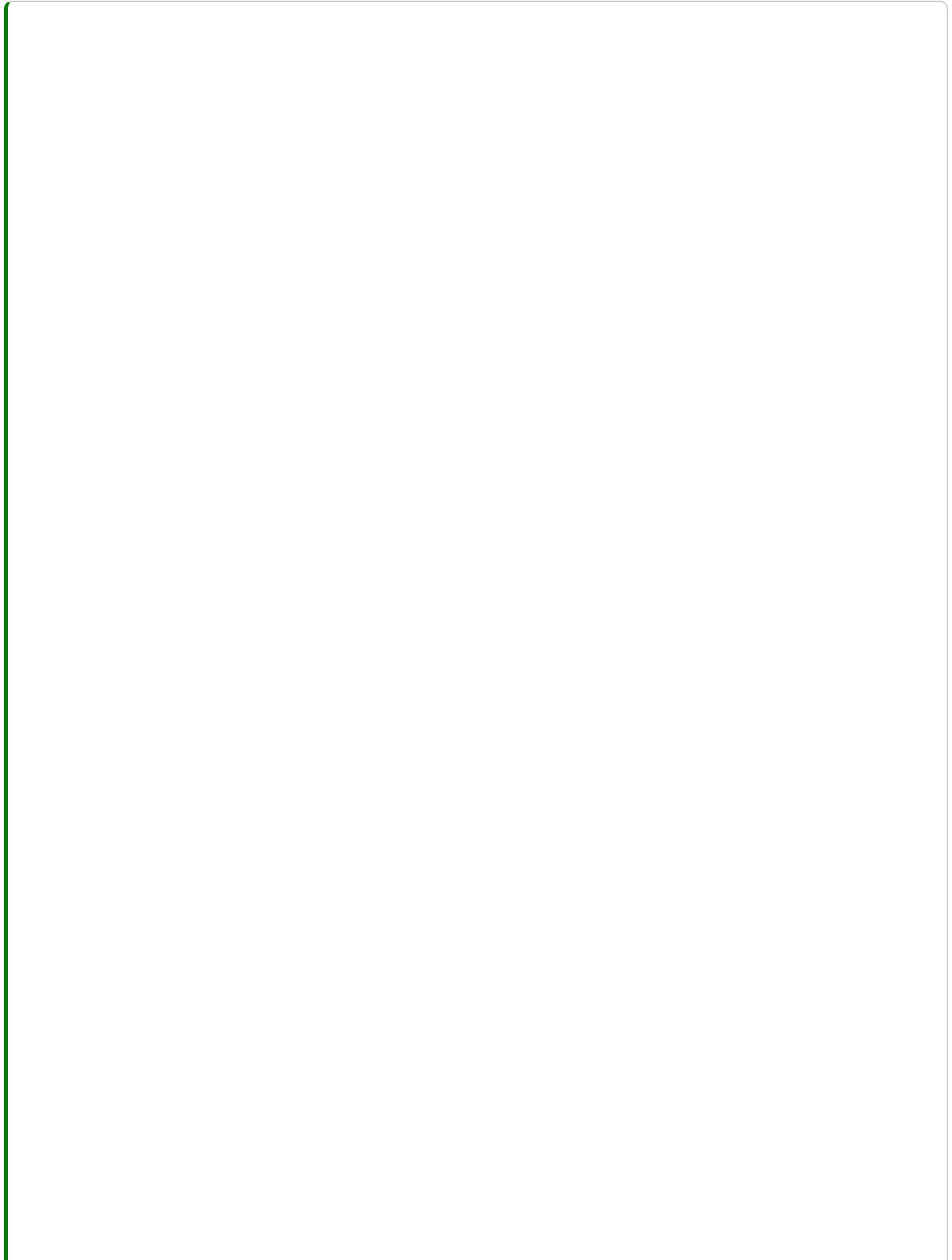
def fit(self, X, y):
    # wir updaten das Baum Attribut der Klasse als letzten Schritt
    self.tree = fit_tree(X, y, depth=0)

```

Folgendes ist bei der Implementierung der `fit()` Methode zu beachten:

- Zunächst werden die Stoppkriterien geprüft
- Dann wird über jedes Feature/Prädiktor iteriert und eine `threshold` S so gesucht, dass der SSE minimiert wird.
- Ist dies erfolgt, dann wird der beste Split als `best_split` Dictionary gesetzt
- Zum Schluss, in der eigentlich vom User aufgerufenen Methode, wird das Attribut `self.tree` der Entscheidungsbaum Klasse gesetzt (hier wird dann zur Laufzeit die rekursive Methode `fit_tree()` aufgerufen), sobald der beste Baum mit den besten Splits gefunden wurde. Die Speicherung in `self.tree` ist nützlich, da wir so den gefundenen Baum in der noch zu implementierenden `predict()` Methode verwenden können.

Lassen Sie uns diese `predict()` Methode als nächstes implementieren. Auch hier splitten wir die Methode wieder in 2 Methoden: Eine, die der User für die gesamten Daten aufruft (`predict()`) und die andere (`predict_sample()`), welche die Implementierung für eine Beobachtung verarbeitet:



```

import numpy as np

class MyDecisionTreeRegressor:
    def __init__(self, max_depth=None, max_n_leaf_observations=1):
        self.max_depth = max_depth
        self.max_n_leaf_observations = max_n_leaf_observations
        self.tree = None

    def calculate_sse(self, y):
        return np.sum((y - np.mean(y)) ** 2)

    def get_overall_sse(self, left_y, right_y):
        left_sse = calculate_sse(left_y)
        right_sse = calculate_sse(right_y)
        return left_sse + right_sse

    def fit_tree(self, X, y, depth):
        n_samples, n_features = X.shape
        unique_y = np.unique(y)

        # Stopkriterien
        if len(unique_y) == self.max_n_leaf_observations:
            return {"value": unique_y[0]}
        if depth == self.max_depth:
            return {"value": np.mean(y)}

        # Greedy algorithm um den beste Split zu finden
        best_split = None
        best_sse = float("inf")

        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = X[:, feature] > threshold

                if np.any(left_indices) and np.any(right_indices):
                    left_y = y[left_indices]
                    right_y = y[right_indices]

                    sse = self.get_overall_sse(left_y, right_y)

                    if sse < best_sse:
                        best_sse = sse
                        best_split = {
                            "feature": feature,
                            "threshold": threshold,
                            "left": (X[left_indices], y[left_indices]),
                            "right": (X[right_indices], y[right_indices]),
                        }

        if best_split is None:
            return {"value": np.mean(y)}
        else:
            return best_split

```

```

        return {"value": np.mean(y)}

    # rekursiver Aufruf der fit Methode
    left_tree = fit_tree(*best_split["left"], depth + 1)
    right_tree = fit_tree(*best_split["right"], depth + 1)

    return {
        "feature": best_split["feature"],
        "threshold": best_split["threshold"],
        "left": left_tree,
        "right": right_tree,
    }

def fit(self, X, y):
    # wir updaten das Baum Attribut der Klasse als letzten Schritt
    self.tree = self.fit_tree(X, y, depth=self.max_depth)

def predict_sample(self, x, tree):
    if "value" in tree:
        return tree["value"]
    if x[tree["feature"]] <= tree["threshold"]:
        return predict_sample(x, tree["left"])
    else:
        return predict_sample(x, tree["right"])

def predict(self, X):
    return np.array([self.predict_sample(x, self.tree) for x in X])

```

Es wird deutlich, dass wir den in der `fit()` Methode erzeugten Entscheidungsbaum hier nun nutzen und die darin gespeicherten Splitwerte für jeden Datenpunkt anwenden. Ebenfalls hervorzuheben ist, dass zu Beginn der `predict_sample()` Methode zunächst geprüft wird ob `"value" in tree`. Diese einfache Prüfung geht auf das Setzen des Strings "value" als der Default für die Stopkriterien in der `fit()` Methode zurück.

Hier ein Beispielauf des soeben implementierten Models:

```

# Beispieldaten
X = np.array([[2, 3], [10, 15], [6, 8], [1, 2], [7, 10]])
y = np.array([1, 4, 2, 1, 3])

# Instantierung des Models und Modelfitting
model = MyDecisionTreeRegressor(max_depth=3, max_n_leaf_observations=5)
model.fit(X, y)

# Nach Modelfit rufen wir die Vorhersagemethode auf
test_data = np.array([[1, 14], [2, 12]])
predictions = model.predict(test_data)
print("Predictions:", predictions)

```

Predictions: [2.2 2.2]

Anhand dieser Implementierung lassen sich Entscheidungsbäume gut nachvollziehen. Bitte beachten Sie jedoch, dass dies **nur ein Beispiel zur Veranschaulichung** ist und das Ergebnis von professionellen Implementierungen wie in der [scikit-learn](#) Bibliothek abweichen kann. Gründe hierfür sind unter anderen folgende:

- sklearn implementiert Entscheidungsbäume effizient und nutzt bessere Algorithmen als unser naives Beispiel (siehe Tree Pruning im nächsten Abschnitt)
- sklearn nutzt den MSE anstatt des SSE.

Nichts desto trotz können wir aus obigem Beispiel die Grundlagen des Greedy Entscheidungsbäume Algorithmus verstehen. Im nächsten Abschnitt werden wir auf eine mögliche Erweiterung hinweisen (Tree Pruning), die in professionellen Implementierungen häufig Anwendung findet und zu weit aus besseren Ergebnissen führt als die einfache Greedy Entscheidungsbäume Variante.

Tree Pruning

Das soeben beschriebene Vorgehen kann sehr gute Vorhersagen auf dem Trainingsdatensatz liefern, kann aber dazu neigen auf zuvor ungesehnenen Validierungs- oder Testdatensätzen schlechte Vorhersagen zu liefern. Dies liegt daran, dass der greedy Ansatz dazu neigt möglichst große Entscheidungsbäume aufzuspannen und somit die Trainingsdaten zu **overfitten**. Ein kleinerer Baum, der weniger Spalten und somit weniger Regionen $R_1 \dots R_n$ enthält, kann aufgrund geringerer Varianz zu besseren generalisierten Vorhersagen auf den Testdaten führen.

Eine Lösung des Problems könnte sein, weniger Splits beim Aufbau des Entscheidungsbäumes zuzulassen, indem man eine Grenze für die relative Verbesserung des SSE vorgibt und somit ab einer bestimmten marginalen Verbesserung des SEE keine weiteren Splits zulässt. Nachteil dieses Verfahrens ist jedoch, dass eventuell "bedeutungslose" Splits zu Beginn des Aufbaus auftreten könnten, wodurch eine spätere Verbesserung des SSE, durch einen weiteren Split, der theoretisch gut wäre, aber aufgrund des Abbruchkriteriums nicht mehr zustande kommt, verhindert wird. Daher zieht man es in der Praxis vor, einen sehr **tiefen Entscheidungsbäume** zu generieren und diesen dann zu **trimmen** (zurück schneiden) um einen **Sub-Baum** zu erhalten.

Das hierfür verwendete Verfahren wird in der Literatur auch **Cost-complexity pruning** oder **weakest link pruning** genannt und kann wie folgt beschrieben werden:

1. Baue eine Entscheidungsbaum durch einaches binäres Splitten auf, bis ein Stoppkriterium (z.B. $N < 5$) erreicht ist (siehe **Top-Down Greedy Entscheidungsbaum Algorithmus**). Dieser tiefe Baum wird als T_0 bezeichnet.
2. Wende **Cost-complexity pruning** auf den tiefen Entscheidungsbaum an um eine Sequenz an Subbäumen T in Abhängigkeit eines Skalierungsparameters α zu erhalten. Zu jedem α gehört also ein Subbaum T , sodass gilt:

$$\$ \sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \$,$$

wobei $|T|$ das Minimum der Anzahl Blätter von T darstellt, R_m die Region des Prädiktorraums, die zum m -ten Blatt gehört, und \hat{y}_{R_m} stellt den vorhergesagten Wert dar, der zur Region R_m gehört, also der Mittelwert der Trainingsdaten in dieser Region.

Daraus folgt:

Wenn $\alpha = 0$, dann ist Subbaum $T = T_0$.

Wenn $\alpha > 0$, dann wird $|T|$ kleiner und folglich auch der Subbaum T . Dies erlaubt ein kontrolliertes **Trimmen** (engl. pruning) des Baumes in Abhängigkeit von α unter Nutzung von Validierungsdaten.

3. Nutze Krossvalidierung um α zu wählen, sodass der mittlere Vorhersagefehler über alle Validierungssätze minimal ist.
4. Kehre zum Subbaum aus Schritt 2 zurück, der zu dem gewählten α gehört.

Regressionsbäume in Python und Scikit-Learn

Im folgenden können wir so einen Entscheidungsbaum auf den **Hitters-Datenatz** anwenden. Dieser Datensatz enthält Features von Baseball Spielern anhand derer das Gehalt der Spieler vorhergesagt werden soll. Wir verwenden bei der Implementierung `scikit-learn` Bibliothek. Das Beispiel soll zeigen, wie der Testfehler nach einer bestimmten Anzahl an Blättern nicht mehr besser wird:

Laden wir zunächst die Daten:

```
data = pd.read_csv("../data/hitters.csv")
data = data.dropna()
data.head(5)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
1	315	81	7	24	38	39	14	3449	835	69	321
2	479	130	18	66	72	76	3	1624	457	63	224
3	496	141	20	65	78	37	11	5628	1575	225	828
4	321	87	10	39	42	30	2	396	101	12	48
5	594	169	4	74	51	35	11	4408	1133	19	501

Als nächstes dummy-encodieren wir die kategorialen Variablen und standarisieren alle metrischen Prädiktoren:

```
# Enkodierung der Daten
data = pd.get_dummies(data, drop_first=True)

# Features und Target definieren
X_df = data.drop("Salary", axis=1)
y_df = data["Salary"]
X = X_df.values
y = y_df.values

# Standardisierung der Daten
scaler_X = StandardScaler()
X_standardized = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_standardized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()
```

Im letzten Schritt können wir nun die Daten in **Trainings-, Validierungs-, und Testdaten** aufteilen und den Entscheidungsbaum trainieren.

```

# Splitten der Daten in train, validatio und test set
X_train, X_temp, y_train, y_temp = train_test_split(
    X_standardized, y_standardized, test_size=0.4, random_state=42
)
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42
)

# Leere Listen zur Speicherung der Ergebnisse
train_mse = []
valid_mse = []
test_mse = []

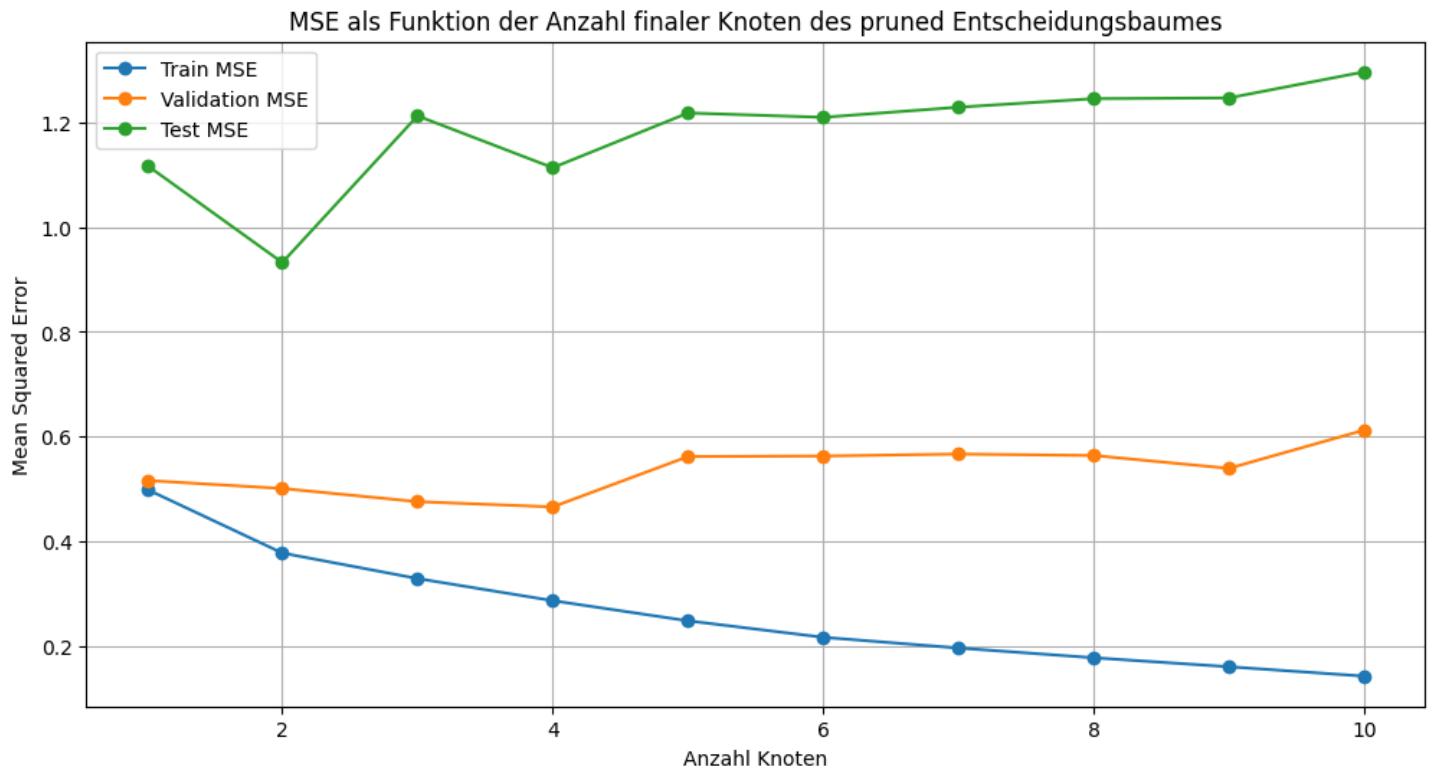
# Fitten der "pruned" Entscheidungsbäume
for n_nodes in range(2, 12):
    tree = DecisionTreeRegressor(max_leaf_nodes=n_nodes, random_state=42)
    tree.fit(X_train, y_train)

    y_train_pred = tree.predict(X_train)
    y_valid_pred = tree.predict(X_valid)
    y_test_pred = tree.predict(X_test)

    train_mse.append(mean_squared_error(y_train, y_train_pred))
    valid_mse.append(mean_squared_error(y_valid, y_valid_pred))
    test_mse.append(mean_squared_error(y_test, y_test_pred))

# Erzeuge Plot
plt.figure(figsize=(12, 6))
plt.plot(range(1, 11), train_mse, label="Train MSE", marker="o")
plt.plot(range(1, 11), valid_mse, label="Validation MSE", marker="o")
plt.plot(range(1, 11), test_mse, label="Test MSE", marker="o")
plt.xlabel("Anzahl Knoten")
plt.ylabel("Mean Squared Error")
plt.title("MSE als Funktion der Anzahl finaler Knoten des pruned Entscheidungsbaumes")
plt.legend()
plt.grid(True)
plt.show()

```



In dem obigen Plot sehen wir, dass **Pruning** durchaus Sinn macht, da zu tiefe Bäume zu Overfitting neigen: Der Entscheidungsbaum beschreibt die Trainingsdaten sehr gut, generalisiert aber nicht auf das Validierungs- und Test-Set der Daten. Nach 4-5 Knoten im Baum reduziert sich zwar der Fehler (in unserem Beispiel der Mean-Squared-Error (MSE)) auf den Trainingsdaten, jedoch nicht auf den Testdaten.

Klassifikations-Bäume

Klassifikationsbäume sind Regressionsbäumen sehr ähnlich, mit dem Unterschied, dass hier eine kategoriale Variable vorhergesagt wird und nicht eine intervall skalierte oder metrische Variable. Das bedeutet, dass bei der Vorhersage nicht der Mittelwert, der zu der Region und assoziierten Blättern des Entscheidungsbaumes gehörenden Beobachtungen zurück geliefert wird, sondern die am **häufigsten auftretende Klasse in den Trainingsdaten** der jeweiligen Region.

Es wird somit auch schnell deutlich, dass wir für Klassifikationsbäume den SSE nicht mehr verwenden können. Anstelle des SSE kommt hier die **Klassifikations-Fehlerrate** zum Einsatz. Diese definiert sich als der Anteil an Trainingsdaten in einer bestimmten Region R_n , die nicht zur häufigsten Klasse dieser Region gehören.

Formal können wir dies wie folgt schreiben:

$$E = 1 - \max_k(\hat{p}_{mk})$$

\hat{p}_{mk} steht in dieser Gleichung also für den Anteil der Trainingdaten oder Trainingsbeobachtungen, die der m-ten Region der k-ten Klasse angehören.

In der Praxis hat sich herausgestellt, dass dieser Klassifikationsfehler nicht sensitiv genug ist um Klassifikationsbäume aufzubauen. Daher kommen in der Praxis zwei bessere Maße zum Einsatz:

- der **Gini Index**
- das Maß der **Cross-Entropy**

Der [Gini-Index](#) ist wie folgt definiert:

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Eine Alternative zum Gini Index ist die [Cross-Entropy](#). Sie ist wie folgt definiert:

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log p_{mk},$$

da $0 \leq \hat{p}_{mk} \leq 1$, folgt dass $0 \leq -\hat{p}_{mk} \log p_{mk}$. Die Cross-Entropy D nimmt also nahe null an wenn die \hat{p}_{mk} der Entscheidungsbäume nahe 0 oder exakt 0 sind. Daher nimmt die Cross-Entropy, analog dem Gini-Index kleinere Werte annimmt, wenn der m-te Knoten des Baumes "pur" ist, also größtenteils Beobachtungen nur einer Klasse enthält.

Klassifikationsbäume in Python

Lassen Sie uns analog den Regressions-Bäumen auch einen einfachen Klassifikationsbaum in Python implementieren.

Zunächst benötigen wir die soeben erläuterten Funktionen zur Berechnung des Gini-Index (wir werden diese der Einfachheit halber verwenden), sowie ein paar Hilfsfunktionen zur Bestimmung der Anzahl Labels. Folgende Methoden fügen wir unserer Klasse `MyDecisionTreeClassifier` zunächst hinzu:

- `__init__()`: wird bei der Instantierung des `MyDecisionTreeClassifier()` Objekts ausgeführt.
- `calculate_gini()` und `gini_impurity()`: berechnen den Gini-Index und die “purheit” an einem bestimmten Knoten.
- `count_labels()` und `most_common_label()`: zählen Labels und geben das häufigste Label zurück.

```
class MyDecisionTreeClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.tree = None
        self.n_features = None

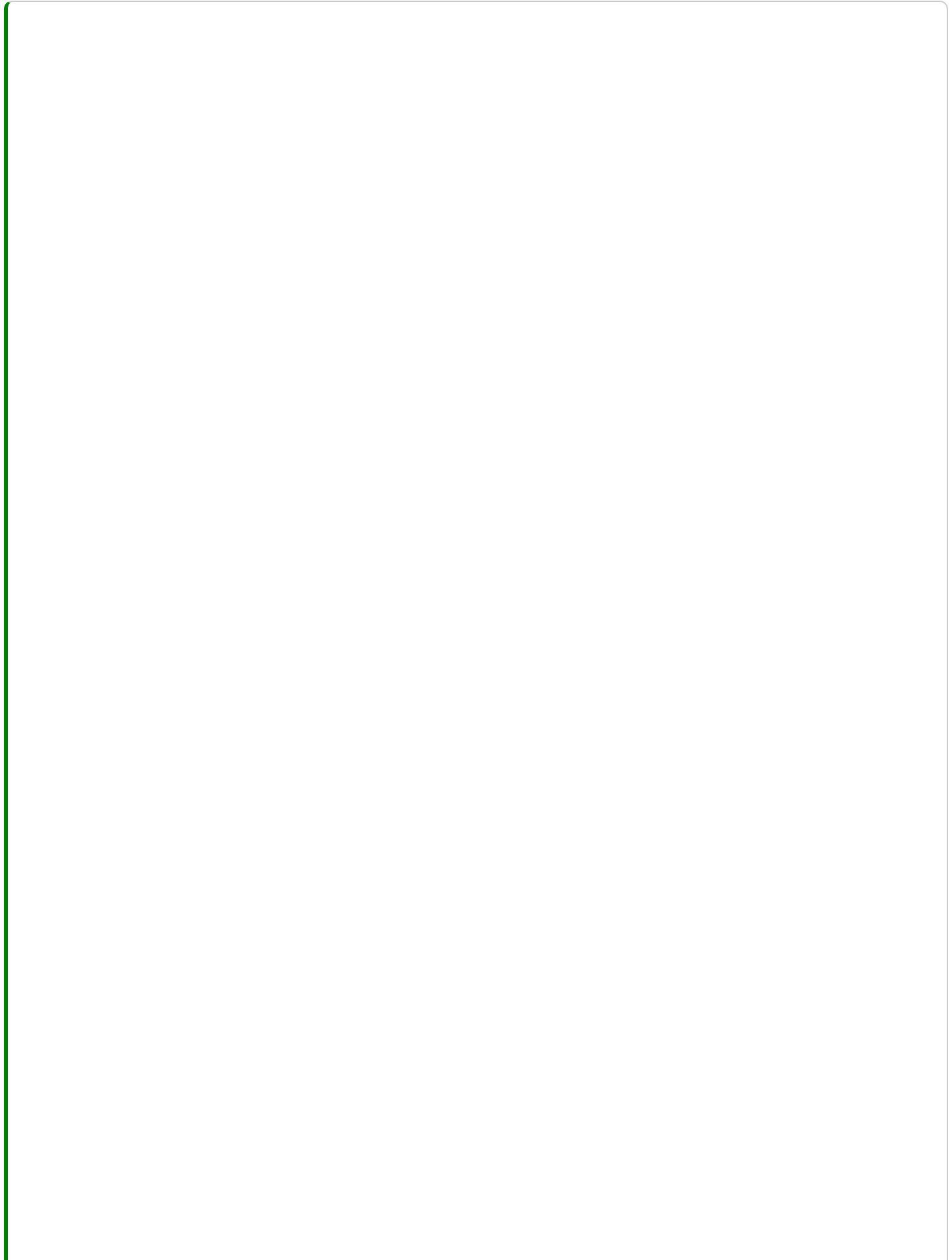
    def calculate_gini(self, left_y, right_y):
        left_impurity = self.gini_impurity(left_y)
        right_impurity = self.gini_impurity(right_y)
        total_impurity = (
            len(left_y) * left_impurity + len(right_y) * right_impurity
        ) / (len(left_y) + len(right_y))
        return total_impurity

    def gini_impurity(self, y):
        counts = self.count_labels(y)
        impurity = 1 - sum((count / len(y)) ** 2 for count in counts.values())
        return impurity

    def count_labels(self, y):
        counts = {}
        for label in y:
            if label in counts:
                counts[label] += 1
            else:
                counts[label] = 1
        return counts

    def most_common_label(self, y):
        counts = self.count_labels(y)
        most_common = max(counts.items(), key=lambda item: item[1])
        return most_common[0]
```

Nun erweitern wir die Implementierung um die Methode `grow_tree()`, welche den Baum aufspannt, sowie die Methode `best_split()`, die den besten Split bestimmt und diesen zusammen mit den besten Features zurück gibt. Die `grow_tree()` Methode wird final in der `fit()` Methode aufgerufen, welche auf vom Endnutzer aufgerufen wird:



```

class MyDecisionTreeClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.tree = None
        self.n_features = None

    def fit(self, X, y):
        self.n_features = X.shape[1]
        self.tree = self.grow_tree(X, y, 0)

    def grow_tree(self, X, y, depth):
        # Prüfe ob alle Label identisch sind
        if len(set(y)) == 1:
            return y[0]
        if self.max_depth is not None and depth >= self.max_depth:
            return self.most_common_label(y)

        # Finde den besten Split
        best_feature, best_threshold = self.best_split(X, y)
        if best_feature is None:
            return self.most_common_label(y)

        left_indices = X[:, best_feature] <= best_threshold
        right_indices = X[:, best_feature] > best_threshold

        left_subtree = self.grow_tree(X[left_indices], y[left_indices], depth + 1)
        right_subtree = self.grow_tree(X[right_indices], y[right_indices], depth + 1)

        return (best_feature, best_threshold, left_subtree, right_subtree)

    def best_split(self, X, y):
        best_gini = float("inf")
        best_feature = None
        best_threshold = None

        for feature in range(self.n_features):
            thresholds = set(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = X[:, feature] > threshold

                if len(set(y[left_indices])) == 1 and len(set(y[right_indices])) == 1:
                    continue

                gini = self.calculate_gini(y[left_indices], y[right_indices])
                if gini < best_gini:
                    best_gini = gini
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold

    def calculate_gini(self, left_y, right_y):

```

```

left_impurity = self.gini_impurity(left_y)
right_impurity = self.gini_impurity(right_y)
total_impurity = (
    len(left_y) * left_impurity + len(right_y) * right_impurity
) / (len(left_y) + len(right_y))
return total_impurity

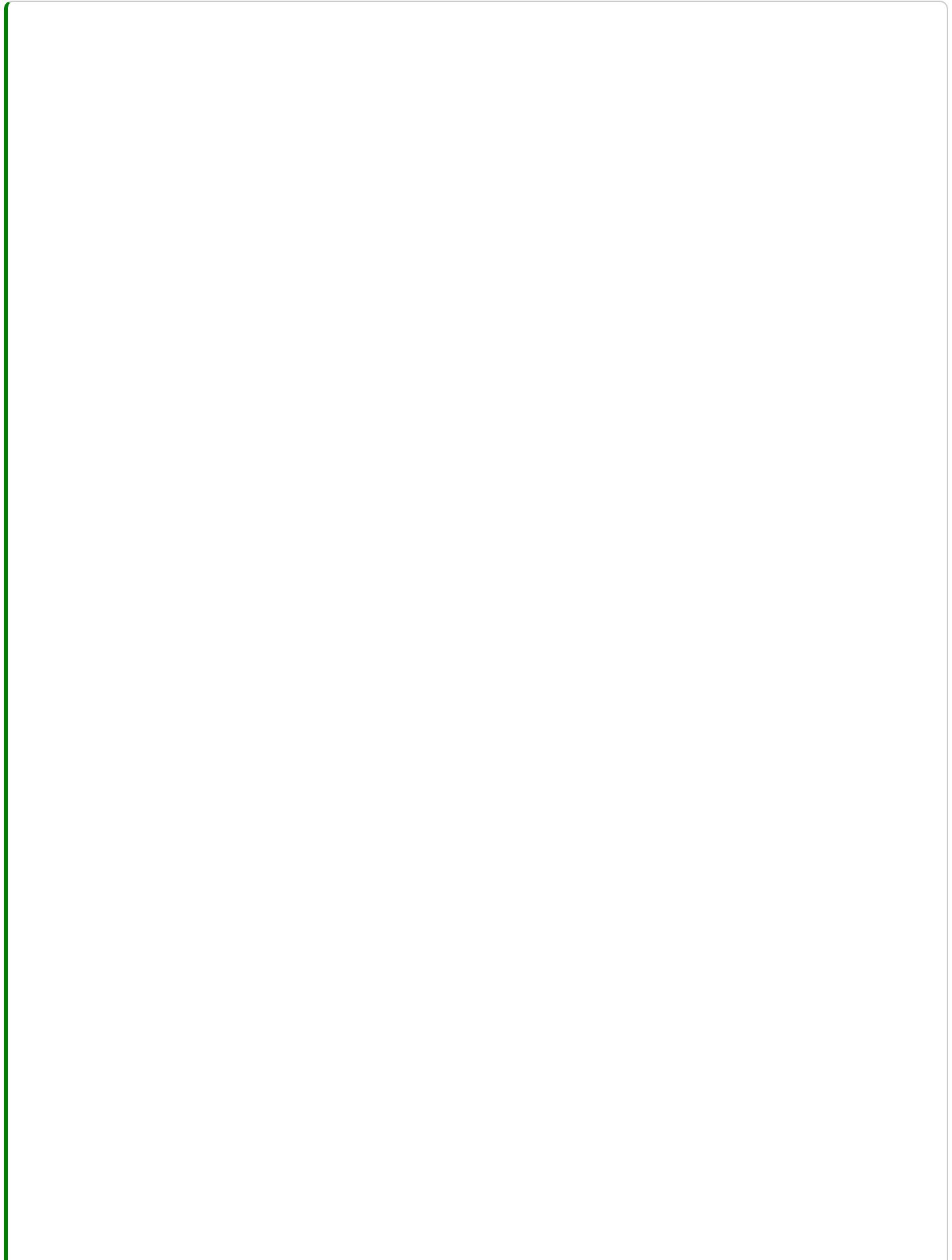
def gini_impurity(self, y):
    counts = self.count_labels(y)
    impurity = 1 - sum((count / len(y)) ** 2 for count in counts.values())
    return impurity

def count_labels(self, y):
    counts = {}
    for label in y:
        if label in counts:
            counts[label] += 1
        else:
            counts[label] = 1
    return counts

def most_common_label(self, y):
    counts = self.count_labels(y)
    most_common = max(counts.items(), key=lambda item: item[1])
    return most_common[0]

```

Zuletzt fügen wir noch die vom Nutzer verwendete Methode `predict()` hinzu, welche intern die Methode `predict_sample()` verwendet:



```

class MyDecisionTreeClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.tree = None
        self.n_features = None

    def fit(self, X, y):
        self.n_features = X.shape[1]
        self.tree = self.grow_tree(X, y, self.max_depth)

    def grow_tree(self, X, y, depth):
        # Prüfe ob alle Label identisch sind
        if len(set(y)) == 1:
            return y[0]
        if self.max_depth is not None and depth >= self.max_depth:
            return self.most_common_label(y)

        # Finde den besten Split
        best_feature, best_threshold = self.best_split(X, y)
        if best_feature is None:
            return self.most_common_label(y)

        left_indices = X[:, best_feature] <= best_threshold
        right_indices = X[:, best_feature] > best_threshold

        left_subtree = self.grow_tree(X[left_indices], y[left_indices], depth + 1)
        right_subtree = self.grow_tree(X[right_indices], y[right_indices], depth + 1)

        return (best_feature, best_threshold, left_subtree, right_subtree)

    def best_split(self, X, y):
        best_gini = float("inf")
        best_feature = None
        best_threshold = None

        for feature in range(self.n_features):
            thresholds = set(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = X[:, feature] > threshold

                if len(set(y[left_indices])) == 1 and len(set(y[right_indices])) == 1:
                    continue

                gini = self.calculate_gini(y[left_indices], y[right_indices])
                if gini < best_gini:
                    best_gini = gini
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold

    def calculate_gini(self, left_y, right_y):

```

```

        left_impurity = self.gini_impurity(left_y)
        right_impurity = self.gini_impurity(right_y)
        total_impurity = (
            len(left_y) * left_impurity + len(right_y) * right_impurity
        ) / (len(left_y) + len(right_y))
        return total_impurity

    def gini_impurity(self, y):
        counts = self.count_labels(y)
        impurity = 1 - sum((count / len(y)) ** 2 for count in counts.values())
        return impurity

    def count_labels(self, y):
        counts = {}
        for label in y:
            if label in counts:
                counts[label] += 1
            else:
                counts[label] = 1
        return counts

    def most_common_label(self, y):
        counts = self.count_labels(y)
        most_common = max(counts.items(), key=lambda item: item[1])
        return most_common[0]

    def predict(self, X):
        predictions = []
        for sample in X:
            predictions.append(self.predict_sample(sample, self.tree))
        return np.array(predictions)

    def predict_sample(self, sample, node):
        if isinstance(node, tuple):
            feature, threshold, left_subtree, right_subtree = node
            if sample[feature] <= threshold:
                return self.predict_sample(sample, left_subtree)
            else:
                return self.predict_sample(sample, right_subtree)
        else:
            return node

```

Folgende Punkte sind zur Implementierung hervorzuheben:

- die Methode `grow_tree()` wird rekursiv aufgerufen, sodass sobald ein Split generiert wurde an diesem Knoten erneut ein Entscheidungsbaum der Tiefe 1 generiert wird. Daraufhin wird beim nächsten rekursiven Aufruf die Tiefe `depth` um 1 inkrementiert. Dies geschieht so lange bis ein Abbruchkriterium erfüllt ist und es wird das Tuple `(best_feature, best_threshold, left_subtree, right_subtree)` zurückgeliefert.

- die Methode `best_split()` nutzt den Gini-Index um den besten Splitpunkt zu liefern. Der einfachehalt halber wird hier auch das beste feature ebenso zurück geliefert: `best_feature, best_threshold`.
- in der Methode `fit()` ist zu sehen, wie das Attribut `self.tree` der Klasse gesetzt wird, sobald der Klassifikationsbaum vollständig rekursiv generiert wurde. Dieses Attribut wird beim Aufruf von `predict()` dann später verwendet.

Auch bei dieser Implementierung ist hervorzuheben, dass es sich nur um ein **anschauliches Beispiel** handelt, anhend dessen wir Klassifikationsbäume besser verstehen können.

Hier ein Beispiellauf des eben implementierten Models:

```
# Beispiel
X = np.array([[2, 3], [1, 1], [4, 5], [3, 4], [5, 6]])
y = np.array([0, 0, 1, 1, 1])

# Instantierung des Klassifikators
clf = MyDecisionTreeClassifier(max_depth=6)
clf.fit(X, y)

# Vorhersage auf Test-Daten
predictions = clf.predict(X)
print(f"Predictions: {predictions}")
```

Predictions: [1 1 1 1 1]

Klassifikationsbäume in Python und Scikit-Learn

Unter Verwendung der Bibliothek `scikit-learn` können wir auch für Klassifikationsbäume eine Anwendungsbeispiel betrachten. Wir nehmen hierzu den **Hurricanes** Datensatz, der Ihnen schon aus dem Kapitel zur logistischen Regression bekannt ist, und wenden den in `scikit-learn` zur Verfügung stehenden `DecisionTreeClassifier()` an. Lassen Sie uns zunächst die Daten laden:

```
hurricanes = pd.read_excel("../data/hurricanes.xlsx", index_col=0)
hurricanes.head(5)
```

	Number	Name	Year	Type	FirstLat	FirstLon	MaxLat	MaxLon	L
RowNames									
1	430	NOTNAMED	1944	1	30.2	-76.1	32.1	-74.8	
2	432	NOTNAMED	1944	0	25.6	-74.9	31.0	-78.1	
3	433	NOTNAMED	1944	0	14.2	-65.2	16.6	-72.2	
4	436	NOTNAMED	1944	0	20.8	-58.0	26.3	-72.3	
5	437	NOTNAMED	1944	0	20.0	-84.2	20.6	-84.9	

Wir werden versuchen die binäre Antwortvariable `Origins` anhand der Prädiktorvariablen `FirstLat` vorherzusagen.

```
# Benamung der Origin
hurricanes["Origins"] = hurricanes["Type"].replace(
    {0: "tropisch", 1: "aussertropisch", 3: "aussertropisch"})
# Erstellung von X und y Datensätzen
X = hurricanes["FirstLat"].values
X = sm.add_constant(X)
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values
```

```
/tmp/ipykernel_21123/1118630555.py:8: FutureWarning: Downcasting behavior in `replac
```

```
# test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Fitten das Klassifikators an die Trainingsdaten
clf = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
clf.fit(X_train, y_train)
```



DecisionTreeClassifier

[i](#) [?](#)

```
DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
```

Nachdem das Modell an die Trainingsdaten gefitted wurde, können wir durch Aufruf der `predict()` Methode Vorhersagen für die Klasse erhalten.

```
y_pred = clf.predict(X_test)  
y_pred
```

```
array([1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,  
1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,  
1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1,  
1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0])
```

Abschliessend können wir, die vom Modell vorhergesagten Werte, mit den echten Werten im Testdatensatz vergleichen um die Performanz des Modells zu evaluieren. Dies können wir anhand der **Accuracy** des Models machen.

Diese ist definiert als

$$\text{Accuracy} = \frac{\text{korrekte Klassifizierungen}}{\text{gesamte Klassifizierungen}}$$

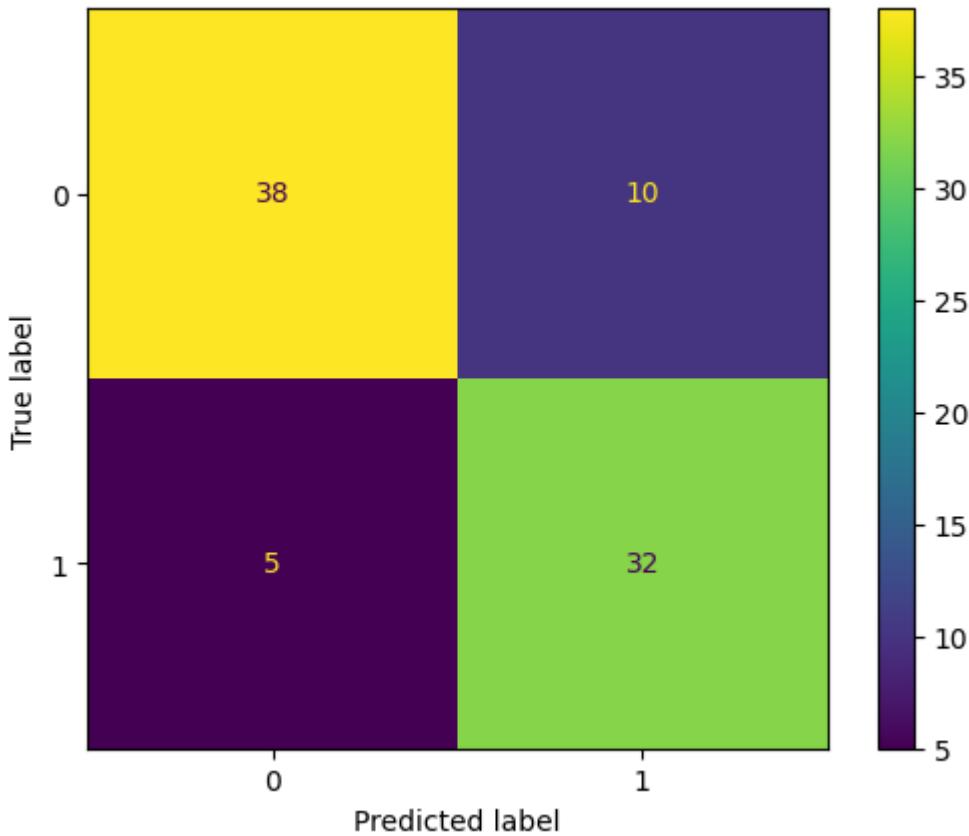
In `scikit-learn` lässt sich die Accuracy wie folgt berechnen:

```
accuracy_score(y_test, y_pred)
```

```
0.8235294117647058
```

Viel genauer sehen wir jedoch, die korrekt klassifizierten Beobachtungen und Missklassifizierungen anhand einer **Konfusionsmatrix**. Bei der [Konfusionsmatrix](#) werden die relativen Häufigkeiten **korrekter und inkorrektler Vorhersagen** der vom Modell vorhergesagten Werte und echten Werte gegeneinander in einer Matrix geplottet.

```
confusion_matrix = confusion_matrix(y_test, y_pred)  
cm_display = ConfusionMatrixDisplay(  
    confusion_matrix=confusion_matrix, display_labels=[0, 1]  
)  
  
cm_display.plot()  
plt.show()
```



Wir sehen also, das unser Modell ganz gute Vorhersagen liefert, da die meisten Beobachtungen auf der Hauptdiagonalen liegen.

Entscheidungsbäume vs. Lineare Modelle

Regressions- und Klassifikationsbäume sind eine fundamental andere Herangehensweise an ein Machine Learning Problem im Vergleich zu klassischen Verfahren wie die lineare Regression. Wenn wir uns das lineare Regressionsmodell ansehen, dann kann es wie folgt formalisiert werden:

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j$$

Im Gegensatz dazu hat ein Entscheidungsbaum folgende Form:

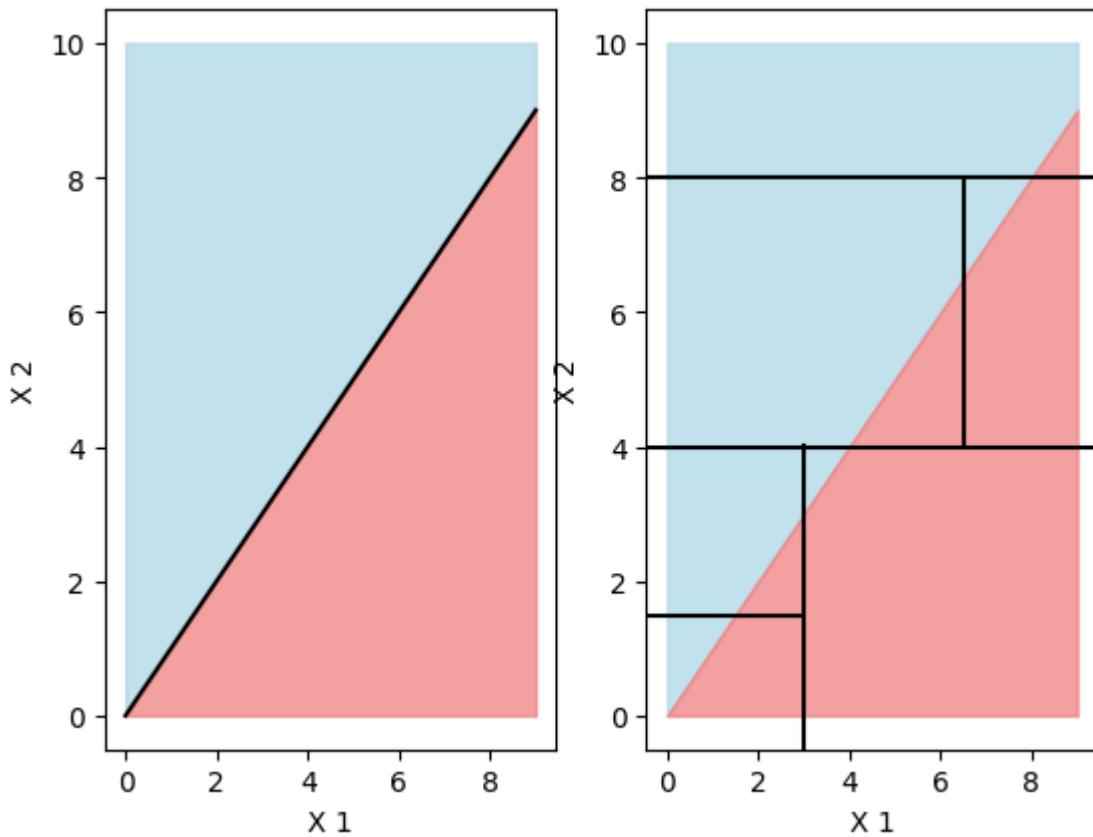
$$f(X) = \sum_{m=1}^M c_m \cdot 1_{X \in R_m}$$

wobei R_1, \dots, R_M eine Partition des Prädiktorraumes repräsentiert.

Wenn man sich nun fragt welches der beiden Modelle besser ist, bleibt einem nur festzustellen, dass es auf das Problem ankommt. Wenn der Zusammenhang zwischen den Prädiktoren und der Antwortvariablen gut durch ein lineares Modell approximiert werden kann, dann wird ein lineares Modell besser als ein Entscheidungsbaum funktionieren. Wenn aber ein non-linearer oder komplexer Zusammenhang zwischen den Prädiktoren und der Antwortvariablen besteht, dann ist ein Entscheidungsbaum besser geeignet. Ein illustratives Beispiel hierzu kann mit folgender Visualisierung gegeben werden:

```
fig, ax = plt.subplots(nrows=1, ncols=2)
x = range(10)
# left plot
ax[0].plot(x, color="k")
ax[0].fill_between(x=x, y1=10, y2=x, color="lightblue", interpolate=True, alpha=0.7)
ax[0].fill_between(x=x, y1=x, y2=0, color="lightcoral", interpolate=True, alpha=0.7)
ax[0].set_xlabel("X 1")
ax[0].set_ylabel("X 2")
# right plot
ax[1].axhline(y=1.5, xmax=0.34, color="k", linestyle="--")
ax[1].axvline(x=3, ymax=0.41, color="k", linestyle="--")
ax[1].axhline(y=4, color="k", linestyle="--")
ax[1].axvline(x=6.5, ymin=0.41, ymax=0.77, color="k", linestyle="--")
ax[1].axhline(y=8, color="k", linestyle="--")
ax[1].fill_between(x=x, y1=10, y2=x, color="lightblue", interpolate=True, alpha=0.7)
ax[1].fill_between(x=x, y1=x, y2=0, color="lightcoral", interpolate=True, alpha=0.7)
ax[1].set_xlabel("X 1")
ax[1].set_ylabel("X 2")
```

Text(0, 0.5, 'X 2')

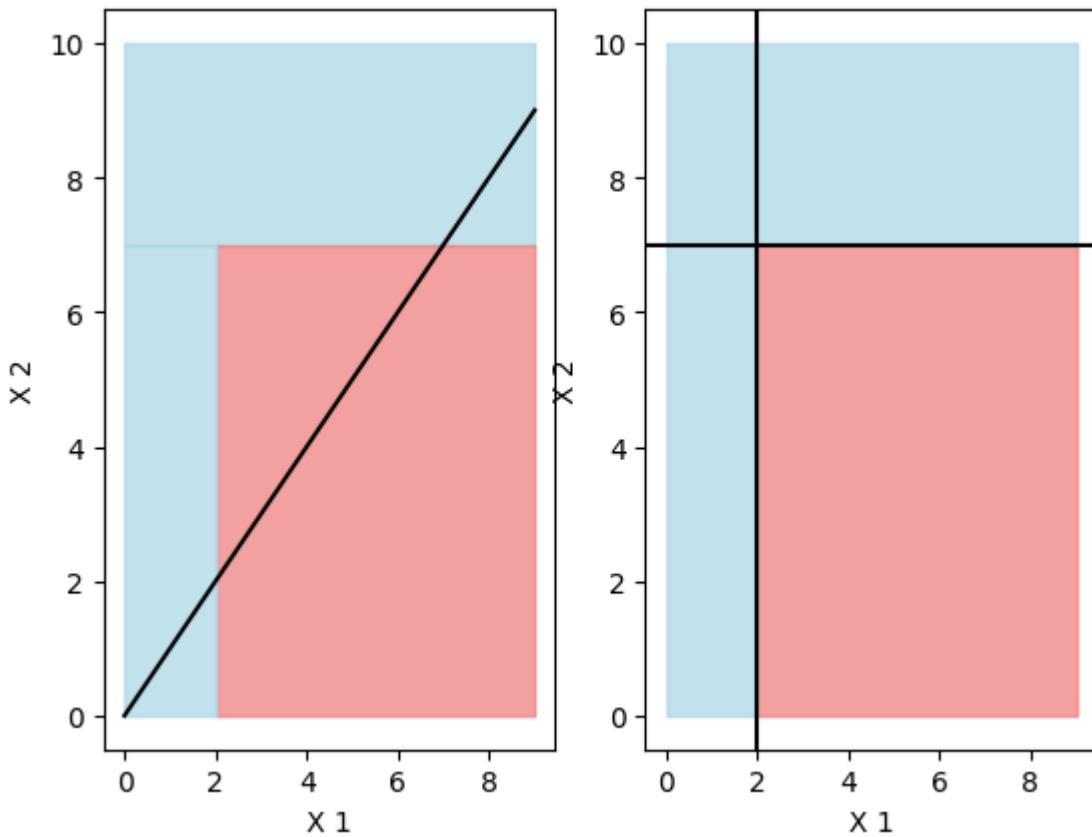


Hier ist auf der linken Seite ein linearer Zusammenhang zwischen x_1 und x_2 farblich kodiert. Dieser Zusammenhang kann somit auch durch ein lineares Modell wie die lineare Regression beschrieben werden. Ein Entscheidungsbaum auf der rechten Seite hat hier Schwierigkeiten, da die Anzahl Regionen sehr hoch sein muss um diese linearen Zusammenhang gut zu beschreiben.

Analog lässt sich das Beispiel auch auf nicht-lineare Zusammenhänge zwischen x_1 und x_2 veranschaulichen (siehe untenstehende Visualisierung). Hier liefert ein lineares Modell auf der linken Seite keine gute Approximation, jedoch ein nicht-lineares Modell, wie der Entscheidungsbaum, eine sehr gute (siehe rechte Seite).

```
fig, ax = plt.subplots(nrows=1, ncols=2)
x = range(10)
# left plot
ax[0].plot(x, color="k")
ax[0].fill_between(x=x, y1=10, y2=7, color="lightblue", interpolate=True, alpha=0.75)
ax[0].fill_between(
    x=range(2, 10), y1=7, y2=0, color="lightcoral", interpolate=True, alpha=0.75
)
ax[0].fill_betweenx(
    y=range(8), x1=2, x2=0, color="lightblue", interpolate=True, alpha=0.75
)
ax[0].set_xlabel("X 1")
ax[0].set_ylabel("X 2")
# right plot
ax[1].fill_between(x=x, y1=10, y2=7, color="lightblue", interpolate=True, alpha=0.75)
ax[1].fill_between(
    x=range(2, 10), y1=7, y2=0, color="lightcoral", interpolate=True, alpha=0.75
)
ax[1].fill_betweenx(
    y=range(8), x1=2, x2=0, color="lightblue", interpolate=True, alpha=0.75
)
ax[1].axvline(x=2, color="k", linestyle="--")
ax[1].axhline(y=7, color="k", linestyle="--")
ax[1].set_xlabel("X 1")
ax[1].set_ylabel("X 2")
```

Text(0, 0.5, 'X 2')



Vor- und Nachteile von Entscheidungsbäumen

Entscheidungsbäume für Regressions- und Klassifikationsprobleme haben einige Vor- und Nachteile gegenüber den Verfahren der vorherigen Kapitel. Diese wollen wir hier auflisten:

Vorteile:

- Entscheidungsbäume lassen sich sehr leicht erklären und sind intuitiver als andere Verfahren.
- Manche Menschen sind überzeugt, dass Entscheidungsbäume näher an menschlicher Entscheidungsfindung sind als andere Verfahren.
- Entscheidungsbäume können grafisch dargestellt werden und sind leicht zu interpretieren. Dies ermöglicht auch Nicht-Experten einen Zugang zu den damit untersuchten Problemen.
- Entscheidungsbäume können qualitative Prädiktoren vararbeiten ohne Dummy Variablen einführen zu müssen.

Nachteile:

- In manchen Fällen können Entscheidungsbäume nicht die selbe Vorhersagegenauigkeit aufweisen wie andere Verfahren, insbesondere bei linearen Zusammenhängen.
- Entscheidungsbäume können fragil sein gegenüber kleinen Änderungen in den Daten. Das bedeutet das kleine Änderungen an den Daten relativ große Abweichungen in der Vorhersage produzieren können.

Die soeben angeführten Nachteile lassen sich jedoch durch Erweiterungen der Entscheidungsbäume (z.B. durch Bagging, Boosting) kompensieren, was uns auf Entscheidungsbäume angewendet zu *Random Forests* führen wird. Diesen Themen werden wir uns in den nächsten beiden Kapiteln widmen.

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

Working on the host: imarevic-pc

Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]

Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti

Bagging und Boosting bei

Entscheidungsbäumen

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statistics
import statsmodels.api as sm
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_squared_error,
    confusion_matrix,
    accuracy_score,
    ConfusionMatrixDisplay,
)
from sklearn.preprocessing import StandardScaler
```

Im vorherigen Kapitel haben wir zur Reduktion der Varianz das Trimmen oder Pruning von Entscheidungsbäumen kennengelernt. Eine weitere Methode, die unabhängig vom verwendeten Modell (Entscheidungsbäume, Regressionmodell, etc.) ist, ist das Bagging.

Bagging

Das [Bagging](#) ist eine Methode, die durch wiederholtes Sampling der Daten Vorhersagen verbessert. Die Fundamentale Idee stützt sich auf das simple Konzept der Berechnung eines Maßes der zentralen Tendenz (z. B. Mittelwert, Median, Modus), nur dass beim Bagging diese Maße auf Vorhersagen von Modellen angewendet werden:

Nehmen wir ein Set der Größe n an **unabhängigen Beobachtungen** an (z.B. mehrere Datensätze der gleichen Art) und bezeichnen diese als Z_1, \dots, Z_n , wobei jeder Datensatz die Varianz σ besitzt und den Mittelwert \hat{Z} . Dann ist die Varianz der Mittelwerte \hat{Z} gegeben durch σ^2/n . Anschaulich bedeutet dies, dass die Bildung des Mittelwerts über mehrere Beobachtungen/Datensätze die Varianz **reduziert**.

Angewendet auf die Nutzung von Machine Learning Modellen, können somit mehrere Modell auf jeweils einer Stichprobe des Datensatzes trainiert werden und die resultierenden Vorhersagen

gemittelt werden.

Formal:

Wir berechnen $\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$, unter Nutzung B verschiedener Trainingsdatensätze. Dann lässt sich ein statistische Model mit geringer Varianz erzeugen durch

$$\hat{f}_{\text{gemittelt}} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

In der Praxis stehen uns nicht mehrere unabhängige Trainingsdatensätze zur Verfügung und daher generieren wir aus dem Ursprungsdatensatz durch zufälliges wiederholtes Ziehen von Stichproben mehrere **bootstrapped Trainingsdaten** (das Ziehen mehrerer Stichproben mit Zurücklegen wird auch [Bootstrap Sampling](#) genannt) um $\hat{f}^{*b}(x)$ zu berechnen und final durch Mitteln folgendes Model zu erhalten:

$$\hat{f}_{\text{bagged}} = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

Dieses Verfahren, lässt sich auf jedes statistische Modell anwenden, jedoch findet es im Kontext der Entscheidungsbäume sehr häufig Anwendung, da es hier zu besonders guten Modellen führt und die Flexibilität von Entscheidungsbäumen sehr gut ausnutzt. Zum Beispiel können nicht getrimmte Bäume verwendet werden, die sehr tief sind um dennoch mithilfe von Bagging sehr genaue Vorhersagen zu erhalten.

Im Falle einer nicht metrischen Vorhersagenvariablen (z.B. im Klassifikationskontext), kann Bagging ebenfalls wie folgt angewendet werden: Man fitted B Klassifikationsbäume an die bootstrapped Trainingsdaten und merkt sich für jedes Model die am besten vorhergesagte Klasse. Aggregiert wird dann indem ein **Majority-Vote** vollzogen wird. Dies bedeutet, dass die finale Vorhersage, die am meisten vorkommende Vorhersage der B Modelle ist.

Lassen Sie uns nun zur Veranschaulichung Bagging in Python demonstrieren.

Bagging für Regressionsbäume in Python

Wir werden nun wieder die **Baseball Hitter-Daten** verwenden um Bagging im Regressionfall zu demonstrieren. Zunächst lesen wir den Datensatz ein.

```
data = pd.read_csv("../data/hitters.csv")
data = data.dropna()
data.head(5)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
1	315	81	7	24	38	39	14	3449	835	69	321
2	479	130	18	66	72	76	3	1624	457	63	224
3	496	141	20	65	78	37	11	5628	1575	225	828
4	321	87	10	39	42	30	2	396	101	12	48
5	594	169	4	74	51	35	11	4408	1133	19	501

Als nächstes wollen wir zum Vergleich einen Regressionsbaum fitten, der auf den gesamten Daten trainiert wurde. Dieser soll als Baseline dienen, gegen die wir dann unser Baggin-Verfahren evaluieren können.

```
# Enkodierung der Daten
data = pd.get_dummies(data, drop_first=True)

# Features und Target definieren
X_df = data.drop("Salary", axis=1)
y_df = data["Salary"]
X = X_df.values
y = y_df.values

# Standartisierung der Daten
scaler_X = StandardScaler()
X_standardized = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_standardized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()
```

```

# Splitten der Daten in train, validatio und test set
X_train, X_temp, y_train, y_temp = train_test_split(
    X_standardized, y_standardized, test_size=0.4, random_state=42
)
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42
)

# Fitten des Entscheidungsbaumes
tree = DecisionTreeRegressor(random_state=42)
tree.fit(X_train, y_train)
y_test_pred = tree.predict(X_test)

mse = mean_squared_error(y_test, y_test_pred)
print(f"Der MSE auf den Testdaten für einen globalen Regressionsbaum: {mse}")

```

Der MSE auf den Testdaten für einen globalen Regressionsbaum: 1.359673782288005

Lassen Sie uns nun Bagging mit $B = 10000$ anwenden und den finalen Testfehler vergleichen. Wir werden immer 80% der Daten bootstrap samplen, also `samples_fraction = 0.8`.

```

B = 10000
samples_fraction = 0.8
models = []
test_preds = []
# Training loop
n_samples = int(len(X) * samples_fraction)
for b in range(1, B + 1):
    # ziehe Stichprobe mit Zurücklegen
    indices = np.random.choice(len(X_train), size=n_samples, replace=True)
    X_sample = X_train[indices]
    y_sample = y_train[indices]
    # fitte Entscheidungsbäume und speichere jedes Model
    tree = DecisionTreeRegressor(random_state=42)
    tree.fit(X_sample, y_sample)
    models.append(tree)

# prediction loop
predictions = np.zeros((X_test.shape[0], B))
# Erzeuge Vorhersage auf Testdaten und speichere in Liste
for i, model in enumerate(models):
    predictions[:, i] = model.predict(X_test)

# Berechne Mittelwert über alle Vorhersagen
bagged_preds = np.mean(predictions, axis=1)

# Berechne MSE
mse = mean_squared_error(y_test, bagged_preds)
print(f"Der MSE auf den Testdaten für einen bagged Regressionsbaum: {mse}")

```

Der MSE auf den Testdaten für einen bagged Regressionsbaum: 0.7818952432442009

Wir sehen also, dass der Testfehler geringer ausfällt und wir somit ein besseres Model zur Vorhersage durch Bagging erhalten haben.

Bagging für Klassifikationsbäume in Python

Im Folgenden werden wir, der Vollständigkeit halber, Bagging auch auf Klassifikationsbäume anwenden. Die Implementierung ist im Prinzip identisch zum Regressionsfall. Wir werden analog zum vorherigen Kapitel nun ebenfalls den **Hurricane-Datensatz** verwenden:

```

hurricanes = pd.read_excel("../data/hurricanes.xlsx", index_col=0)
hurricanes.head(5)

```

	Number	Name	Year	Type	FirstLat	FirstLon	MaxLat	MaxLon	L
RowNames									
1	430	NOTNAMED	1944	1	30.2	-76.1	32.1	-74.8	
2	432	NOTNAMED	1944	0	25.6	-74.9	31.0	-78.1	
3	433	NOTNAMED	1944	0	14.2	-65.2	16.6	-72.2	
4	436	NOTNAMED	1944	0	20.8	-58.0	26.3	-72.3	
5	437	NOTNAMED	1944	0	20.0	-84.2	20.6	-84.9	

```

# Benamung der Origin
hurricanes["Origins"] = hurricanes["Type"].replace(
    {0: "tropisch", 1: "aussertropisch", 3: "aussertropisch"})
)
# Erstellung von X und y Datensätzen
X = hurricanes["FirstLat"].values
X = sm.add_constant(X)
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values

# test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

```

```

/tmp/ipykernel_21189/1862976890.py:8: FutureWarning: Downcasting behavior in `replace` has changed in pandas 1.3.0
y = hurricanes["Origins"].replace({"tropisch": 0, "aussertropisch": 1}).values

```

Nun haben wir Test- und Trainingsdaten und können mit der Implementierung der Bagging Variante des Klassifikationsbaumes fortfahren. Die Implementierung sieht wie folgt aus:

```

B = 10000
samples_fraction = 0.8
models = []
test_preds = []
# Training loop
n_samples = int(len(X) * samples_fraction)
for b in range(1, B + 1):
    # ziehe Stichprobe mit Zurücklegen
    indices = np.random.choice(len(X_train), size=n_samples, replace=True)
    X_sample = X_train[indices]
    y_sample = y_train[indices]
    # fitte Entscheidungsbäume und speichere jedes Model
    tree = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
    tree.fit(X_sample, y_sample)
    models.append(tree)

# prediction loop
predictions = np.zeros((X_test.shape[0], B), dtype=int)
# Erzeuge Vorhersage auf Testdaten und speichere in Liste
for i, model in enumerate(models):
    predictions[:, i] = model.predict(X_test)

# Berechne Majority Vote über alle Vorhersagen
bagged_preds = np.array(
    [np.bincount(predictions[i]).argmax() for i in range(predictions.shape[0])])
)

```

Wir berechnen auch hier die Accuracy:

```

# Berechne Accuracy
acc = accuracy_score(y_test, bagged_preds)
print(
    f"Der Accuracy Score auf den Testdaten für einen bagged Klassifikationsbaum: {acc}")

```

Der Accuracy Score auf den Testdaten für einen bagged Klassifikationsbaum: 0.8235294

Boosting

Genauso wie das Bagging, kann [Boosting](#) generell bei allen machine learning Verfahren eingesetzt werden. Es ist vom Prinzip her nicht auf Entscheidungsbäume beschränkt, da es aber bei einer Variante der Random Forests zum Einsatz kommt und somit im Kontext der Entscheidungsbäume

sehr verbreitet ist, werden wir Boosting ebenfalls unter Verwendung von Entscheidungsbäumen vorstellen.

Wir erinnern uns, dass beim Bagging das *Bootstrap Sampling* zum Einsatz kam um für jedes Model eine neue Stichprobe zu ziehen und die Vorhersagen am Ende global zu aggregieren. Im Gegensatz hierzu involviert das Boosting nicht wiederholtes Ziehen eines Bootstrap Samples, sondern es ist ein **sequentielles Verfahren**, dass Informationen von zuvor aufgespannten Entscheidungsbäumen in das Lernverfahren einbezieht und somit jeder Entscheidungsbaum auf einer modifizierten Version der Originaldaten trainiert wird.

Im Gegensatz zum Bagging ist das Boosting ein **langsame Verfahren**. Es wird so bezeichnet, da hier ein Entscheidungsbaum zunächst auf die Daten gefittet wird und danach immer wieder auf die Residuen der Vorhersagen des vorherigen Models. Es werden also zu jeder Iteration die **Residuen** als Y betrachtet und ein Model darauf trainiert. Der resultierende Baum wird dann nach jeder Iteration zurück in die gefittete Funktion integriert um die Residuen zu updaten. Die verwendeten Bäume können hierbei sehr klein sein. Beim Boosting wird diese Tiefe durch den Parameter d kontrolliert. Durch Nutzung sehr kleiner Bäume, verbessern wir die Vorhersage sehr *langsam* und verbessern somit die finale Funktion \hat{f} , die das Gesamtmodel über die Iterationen beschreibt besonders in den Bereichen, in denen das Model schlechte Vorhersagen macht. Das sehr langsame Lernen wird noch weiter verlangsamt, wenn wir einen Parameter λ einbauen, der die Lernrate verringert und somit erlaubt verschiedene Bäume aufzubauen, da dann insgesamt mehr Bäume möglich sind.

Formal lässt sich der Boosting-Algorithmus wie folgt beschreiben:

1. Setze $\hat{f}(x) = 0$ und $r_i = y_i$ für alle Beobachtungen i im Trainingsdatensatz.
2. Für $b = 1, 2, \dots, B$, wiederhole folgendes:
 - A. Fitte einen Baum \hat{f}^b mit d Splits ($d+1$ finale Knoten) an die Trainingsdaten.
 - B. Update \hat{f} durch hinzufügen einer kleineren Version des vorherigen Baumes:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

- C. Update die Residuen r :

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Gebe das geboostete Model zurück:

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$$

Im generellen Boosting-Algorithmus sind folgende Parameter involviert:

Parameter	Beschreibung
B	Die Anzahl an Bäumen. Wenn B zu groß gewählt wird, kann eine Boosting-Algorithmus overfitten. Daher wird B häufig durch Krossvalidierung auf einem Validierungsdatensatz gewählt
λ	Dieser Parameter wird Shrinkage Parameter genannt, da dieser die Lernrate kontrolliert, mit der der Boosting-Algorithmus lernt. Er sorgt dafür dass die Bäume sich über mehrere Iterationen nicht zu sehr ähneln.
d	Dieser Parameter kontrolliert die Komplexität der Bäume. Beim Boosting werden häufig nur sehr geringe Tiefen gewählt. Meistens reicht eine Tiefe von $d = 1$ vollkommen aus (nur 1 Split).

Boosting in Python

In Python lässt sich Boosting bei Entscheidungsbäumen gut mit der Bibliothek `scikit-learn` implementieren. Hierbei ist zu beachten, dass verschiedene Varianten des oben beschriebenen Boosting-Algorithmus existieren:

- AdaBoost (Adaptive Boosting)
- Gradient Boost
- XGBoost

Jede dieser Varianten. Zum Beispiel werden bei AdaBoost die Gewichte des Models bei jeder Iteration geupdated und bei Gradient Boosting wird oben beschriebenes Update Verfahren über die

Residuen gewählt. XGBoost ermöglicht noch weitere Regularisierung des Models. Die Details jedes dieser Verfahren sind jedoch nicht Gegenstand dieses Moduls.

Im Folgenden werden wir anschaulich Gradient Boosting aus `scikit-learn` in Python verwenden. Wir nutzen hierfür wieder den Baseball Hitter Datensatz:

```
data = pd.read_csv("../..../data/hitters.csv")
data = data.dropna()
data.head(5)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
1	315	81	7	24	38	39	14	3449	835	69	321
2	479	130	18	66	72	76	3	1624	457	63	224
3	496	141	20	65	78	37	11	5628	1575	225	828
4	321	87	10	39	42	30	2	396	101	12	48
5	594	169	4	74	51	35	11	4408	1133	19	501

```
# Enkodierung der Daten
data = pd.get_dummies(data, drop_first=True)

# Features und Target definieren
X_df = data.drop("Salary", axis=1)
y_df = data["Salary"]
X = X_df.values
y = y_df.values

# Standardisierung der Daten
scaler_X = StandardScaler()
X_standardized = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_standardized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

# Splitten der Daten in train, validatio und test set
X_train, X_temp, y_train, y_temp = train_test_split(
    X_standardized, y_standardized, test_size=0.4, random_state=42
)
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42
)
```

Nun werden wir den `GradientBoostingRegressor()` auf die Trainisdaten anwenden und den MSE auf dem Testdatensatz ausgeben. Es ist zu bachten, dass wir im Dictionary `params` ein paar

Hyperparameter setzen, die das Training kontrollieren. In unserem Beispiel wählen wir die verbreiteten Settings (z.B. `n_estimators : 500` oder `learning_rate : 0.01`) und erlauben nur sehr kleine Bäume (`max_depth : 4`):

```
# Setzen der Lernparameter
params = {
    "n_estimators": 500,
    "max_depth": 4,
    "min_samples_split": 5,
    "learning_rate": 0.01,
    "loss": "squared_error",
}

# Modelfitting
reg = GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

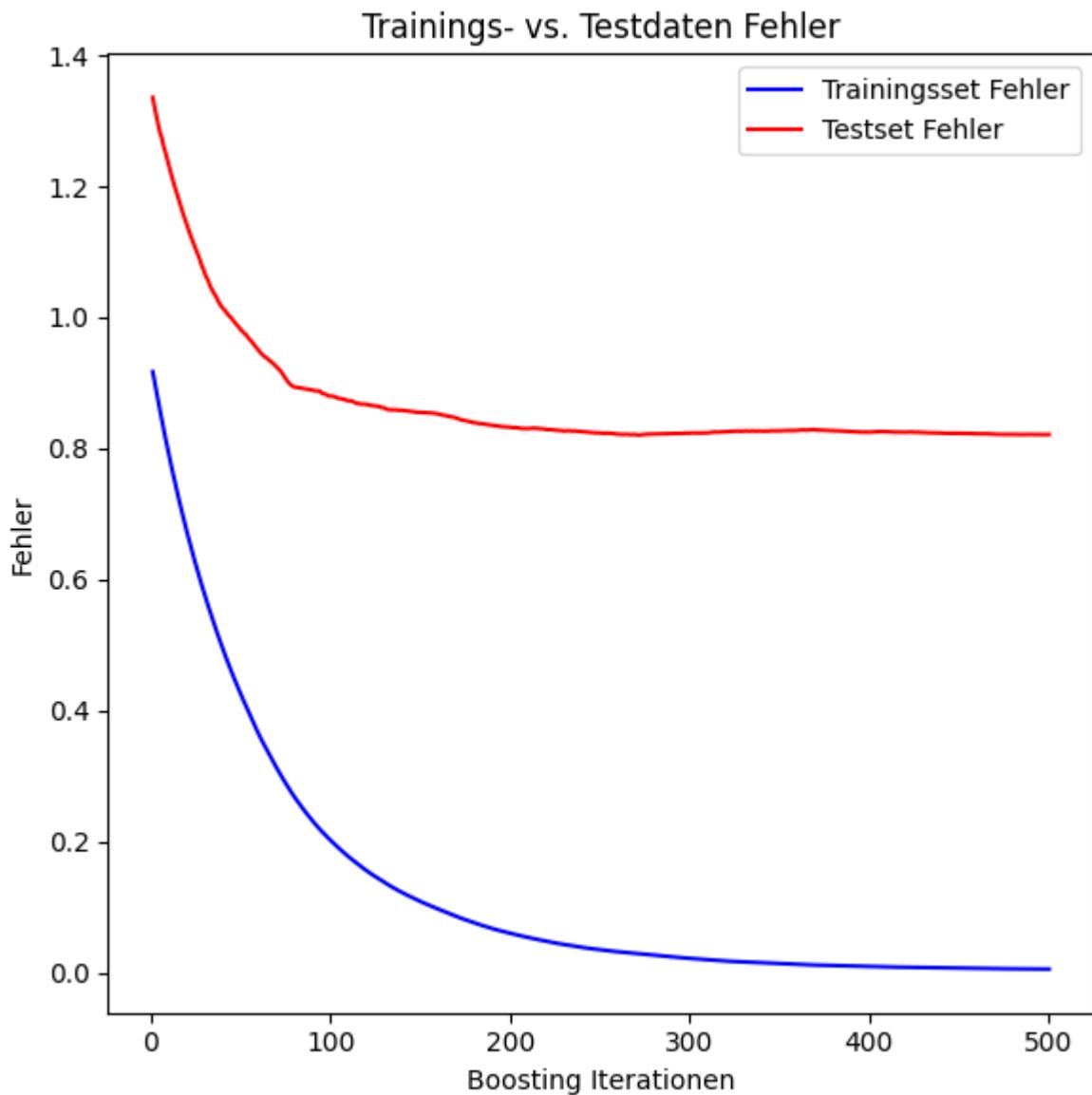
# MSE
mse = mean_squared_error(y_test, reg.predict(X_test))
print("The mean squared error (MSE) auf den Testdaten: {:.4f}".format(mse))
```

```
The mean squared error (MSE) auf den Testdaten: 0.8215
```

Um zu veranschaulichen wie der Boosting-Algorithmus **langsam** auf den Trainingsdaten lernt und bei jeder Iteration die **vorherige Vorhersage etwas verbessert**, werden wir nun den Trainings- und Testfehler für jede Iteration $B = 1, 2, \dots, N$ plotten.

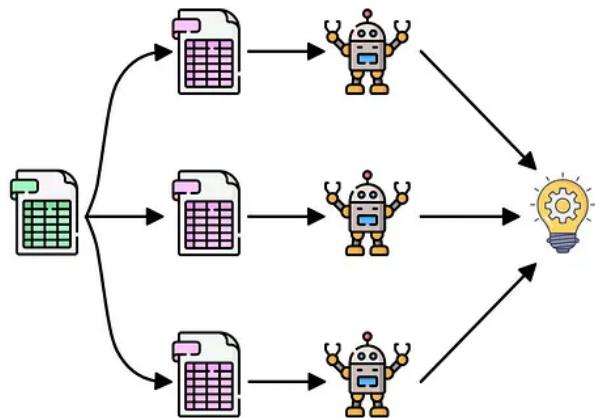
```
test_score = np.zeros((params["n_estimators"])), dtype=np.float64)
for i, y_pred in enumerate(reg.staged_predict(X_test)):
    test_score[i] = mean_squared_error(y_test, y_pred)

fig = plt.figure(figsize=(6, 6))
plt.subplot(1, 1, 1)
plt.title("Trainings- vs. Testdaten Fehler")
plt.plot(
    np.arange(params["n_estimators"]) + 1,
    reg.train_score_,
    "b-",
    label="Trainingsset Fehler",
)
plt.plot(
    np.arange(params["n_estimators"]) + 1, test_score, "r-", label="Testset Fehler"
)
plt.legend(loc="upper right")
plt.xlabel("Boosting Iterationen")
plt.ylabel("Fehler")
fig.tight_layout()
plt.show()
```

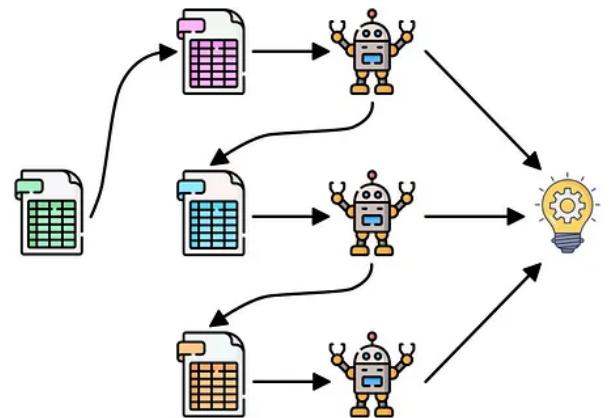


Zum Abschluss lässt sich Bagging und Boosting wie folgt zusammenfassen:

Bagging



Boosting



Parallel

Sequential

Bagging:

- Ziehe N Trainingsdatensätze mit Zurücklegen
- Trainiere auf jedem der Datensätze ein Modell
- Aggregiere die Vorhersagen der Modelle um eine genauere finale Vorhersage zu erhalten

Boosting:

- Trainiere ein Modell auf den Trainingsdaten
- Fitte das Modell auf die Residuen des vorherigen Trainingslaufs und update die Residuen
- Füge bei jeder Iteration sequentiell das erhaltene Modell dem Ursprungsmodell hinzu

```
%matplotlib inline
# Load the "autoreload" extension
%load_ext autoreload
# always reload modules
%autoreload 2
# black formatter for jupyter notebooks
# %load_ext nb_black
# black formatter for jupyter lab
%load_ext lab_black

%run ../../src/notebook_env.py
```

```
-----  
Working on the host: imarevic-pc
```

```
-----  
Python version: 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

```
-----  
Python interpreter: /home/imarevic/Documents/teaching/SRH/content/statistik/statisti
```

Random Forest

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_squared_error,
    confusion_matrix,
    accuracy_score,
    ConfusionMatrixDisplay,
)
from sklearn.preprocessing import StandardScaler
```

Wir haben in den vorherigen Kapiteln sehr viel über Entscheidungsbäume und Wege zur Optimierung dieser (z.B. Bagging und Boosting) gelernt. Dieses Wissen bietet im Prinzip schon das gesamte Wichtige ab, dass wir verstehen müssen um [Random Forests](#) zu verstehen und anwenden zu können. Dieses Kapitel wird daher ein sehr kurzes Kapitel, da wir die Grundlagen hierfür schon in den vorherigen beiden Kapiteln erarbeitet haben.

Random Forests bieten gegenüber Bagging die Vorteile, dass die Entscheidungsbäume, die an die Daten gefittet werden und über deren Vorhersagen am Ende “gebagged” (gemittelt) wird, **dekorrelierte** Bäume sind. Was bedeutet jedoch dekorreliert?

Dekorrelierte Bäume werden bei Random Forests erzeugt, indem bei jeder Iteration nur ein Bruchteil der Prädiktoren im Trainingsdatensatz verwendet werden. Lassen Sie uns ein Beispiel machen: Nehmen wir an wir haben einen Datensatz mit den 5 Prädiktoren A, B, C, D, und E. Wenn wir einfaches Bagging durchführen, dann würden wir alle 5 Prädiktoren in jeder Iteration B des Bagging-Verfahrens verwenden. Wenn nun aber der Prädiktor C ein sehr guter Prädiktor im Vergleich zu den anderen ist, dann würde dies bedeuten, dass wir B sehr stark korrelierte Bäume als Modell trainiert haben und die resultierenden Bäume sich sehr ähneln werden. Daher wird bei Random Forests immer nur ein Bruchteil $m \approx \sqrt{p}$ der insgesamt zur Verfügung stehenden p Prädiktoren im Training verwendet. Dies stellt somit sicher, dass kein dominanter Prädiktor in jedem Baum, der in den B Iterationen trainiert wurde, vorkommt und somit die Bäume **dekorreliert** sind.

Feature Importance

Da wir bei Random Forests immer nur einen Teil der Prädiktoren verwenden, lässt sich die **Feature Importance**, also die Güte hinsichtlich jedes Prädiktors abschätzen. Dies geschieht in dem der Abfall der Summe der Fehlerquadrate (SSE), oder ein anderes Fehlermaß, über alle gebagten Entscheidungsbäume in Abhängigkeit der Splits und Prädiktoren, die in jedem Split vorhanden waren, berechnet wird. Als Maß für die Feature Importance wird häufig der **Mean Impurity Index (MDI)** verwendet. Dieser ist in [scikit-learn](#) als Feature Importance Algorithmus für Random Forests implementiert und berechnet sich wie folgt:

1. Für jeden Entscheidungsbaum und Knoten im Entscheidungsbaum berechne die **Gini-Impurity** G :
$$G = 1 - \sum_{i=1}^k (p_i)^2$$
 wobei mit k die Anzahl Kategorien in die gesplitted wurde bezeichnet wird und p_i der Anteil Beobachtungen die zur Kategorie i an dem jeweiligen Knoten gehören.
2. Die Gini-Impurity wird einmal für den gesamten Entscheidungsbaum und alle Bäume im Forest berechnet. Diese wird $G_{initial}$ genannt.
3. Die Gini-Impurity wird dann für jeden Knoten bei dem ein Feature/Prädiktor involviert war ebenfalls berechnet. Für Features $F_1, F_2, F_3, \dots, F_n$ wird also separat jeweils respektive ein $G_{F_1}, G_{F_2}, G_{F_3}, \dots, G_{F_n}$, berechnet.

4. Im Anschluss wird dann für jedes Feature die Differenz $D_{reduction}$ zur initialen Gini-Impurity $G_{initial}$ berechnet, sodass

$$D_{reduction} = G_{initial} - G_{F_i}, \forall i \in \{1, 2, 3, \dots, N\}$$

5. Zuletzt wird dann über alle Bäume und Knoten die MDI berechnet:

$$MDI = \frac{\sum_{i=F_1}^{F_N} D_{reduction}}{N_{features}}$$

In der folgenden Implementierung werden wir sehen, wie wir die Feature Importance für jeden Prädiktor für Random Forests ausgeben und plotten können.

Random Forests in Python

Im Folgenden werden wir Random Forests mit Hilfe der [scikit-learn](#) Bibliothek in Python implementieren. Wir werden uns hier auf den Regressionsfall beschränken, der Klassifikationsfall funktioniert jedoch analog.

```
data = pd.read_csv("../data/hitters.csv")
data = data.dropna()
data.head(5)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
1	315	81	7	24	38	39	14	3449	835	69	321
2	479	130	18	66	72	76	3	1624	457	63	224
3	496	141	20	65	78	37	11	5628	1575	225	828
4	321	87	10	39	42	30	2	396	101	12	48
5	594	169	4	74	51	35	11	4408	1133	19	501

```

# Enkodierung der Daten
data = pd.get_dummies(data, drop_first=True)

# Features und Target definieren
X_df = data.drop("Salary", axis=1)
y_df = data["Salary"]
X = X_df.values
y = y_df.values

# Standardisierung der Daten
scaler_X = StandardScaler()
X_standardized = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_standardized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

# Splitten der Daten in train, validation und test set
X_train, X_temp, y_train, y_temp = train_test_split(
    X_standardized, y_standardized, test_size=0.4, random_state=42
)
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42
)

```

```

# Model Training
regr_rf = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=42)
regr_rf.fit(X_train, y_train)

# Vorhersage neuer Beobachtungen
y_rf = regr_rf.predict(X_test)

# MSE
mse = mean_squared_error(y_test, y_rf)
print("The mean squared error (MSE) auf den Testdaten: {:.4f}".format(mse))

```

The mean squared error (MSE) auf den Testdaten: 0.7893

Zuletzt werden wir die Feature Importance ausgeben um zu inspizieren, welche der Prädiktoren die meiste Varianz im Modell aufklären:

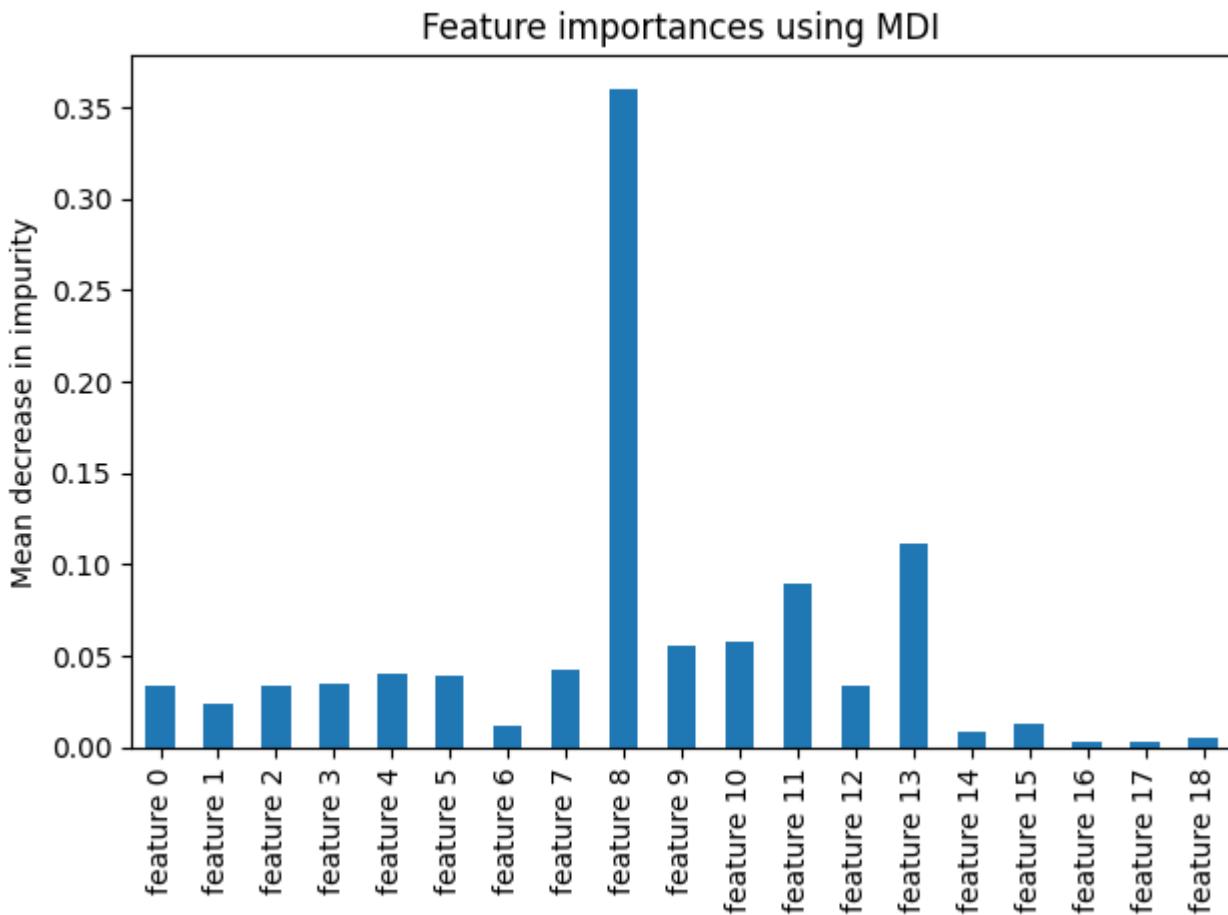
```

# Extraktion der Feature Importances aus dem Model Objekt
importances = regr_rf.feature_importances_

# Feature Namen und STD werden generiert
feature_names = [f"feature {i}" for i in range(X_test.shape[1])]
forest_importances = pd.Series(importances, index=feature_names)
std = np.std([regr_rf.feature_importances_ for tree in regr_rf.estimators_], axis=0)

# Plotten der Feature Importance
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()

```



Da wir die Prädiktoren im Vorfeld dummy-kodiert haben, sehen wir nun genau diese im Plot. Es ist deutlich zu erkennen, dass **feature 8** die höchste Varianzaufklärung besitzt, gefolgt von **feature 13** und **feature 11**. Das bedeutet, dass ein Model mit nur den Features mit höchster Feature Importance sehr wahrscheinlich ausreichend wäre, und die anderen Features keinen prädiktiven Mehrwert haben.

Übungsaufgaben

Entscheidungsbäume

1. Wie wird beim “greedy” Aufbau von Entscheidungsbäumen entschieden wann keine Splits mehr durchgeführt werden?
 2. Was ist “cost-complexity pruning” und warum wird es bei Entscheidungsbäumen eingesetzt?
 3. Nennen Sie die Vor- und Nachteile von Entscheidungsbäumen gegenüber anderen Regressionsverfahren.
-

Lösungen

Bagging & Random Forests

1. Nehmen wir 10 bootstrapped Samples aus einem Datensatz als gegeben an, die “rote” und “grüne” Klassen als Vorhersagevariable beinhalten. Wir wenden nun 10 Entscheidungsbäume auf den Datensatz X an und erhalten 10 Wahrscheinlichkeiten $P(Klasse \text{ ist } rot | X)$:

$$P(red|X) = \{0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75\}$$

Wenden sie das Majority-Vote an um die finale Klassifizierung zu erhalten.

2. Wir berechnen die Feature Importance für einen Random Forest mit 10 Features ($F_1 - F_{10}$) als Input. Die Mean-Decrease Impurity (MDI) zeigt für die Features $F_4, F_7, \text{ und } F_9$ folgende Werte an:

- $F_4 = 0.33$
- $F_7 = 0.30$
- $F_9 = 0.14$

Alle anderen Features haben MDIs < 0.05

Was bedeutet dies für das Random Forest Model?

Lösungen

Literaturverzeichnis