Hochschule Esslingen
University of Applied Sciences

Rechnerübungen Statistik mit Python / Beispiele für die Einführungsstunde

Für alle Beispiele benötigen Sie eine Datei einf_daten.jpynb.
Für Beispiel 1 ist zusätzlich die Datei einf_bl.txt **erforderlich.**

Wenn Sie die Lösungswege nachvollziehen wollen:

- **In der Datei** einf_beispiele_mit_lösungsweg.pdf **stehen ebenfalls die unten genannten Beispiele zusammen mit Lösungsvorschlägen.**

Damit keine Missverständnisse auftreten: Die hier genannten Beispiele sind nicht die Testat-Aufgaben sr_aufg_1 bis sr_aufg_3, die Sie selbständig bearbeiten sollen.

In den hier vorliegenden Beispielen werden einige wichtige Statistik- Funktionen der Software Python erläutert. Es können aber nicht alle Funktionen und Optionen angesprochen werden, die bei der Bearbeitung der Testataufgaben benötigt werden.

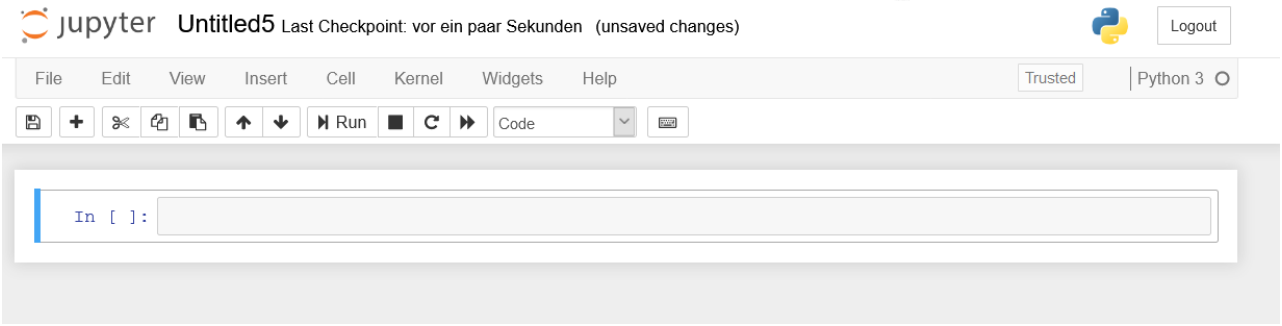
1. Installation *Python* und *Jupyter* Notebooks

Ein Jupyter Notebook vereint ausführbaren Python-Code und seine Ausgaben und weiteren Text in einem einzigen Dokument. Zusätzlich können in einem solchen Dokument auch Graphiken, beschreibender Text, mathematische Gleichungen und andere Bestandteile vorkommen. Ein Beispiel für ein Jupyter Notebook finden Sie unter einf_daten.ipynb.

Jupyter Notebooks sind kostenlos und lassen sich am Besten über das „Anaconda data science toolkit“ herunterladen. Nach einem erfolgreichen Herunterladen des „Anaconda data science toolkits“ sollten Sie unter Ihrem Eingangsmenü unter dem Menüpunkt „Anaconda“ auch den Eintrag „Jupyter Notebook“ sehen.

Durch Anklicken öffnet sich im Browser ein Menü Ihrer Dateien:

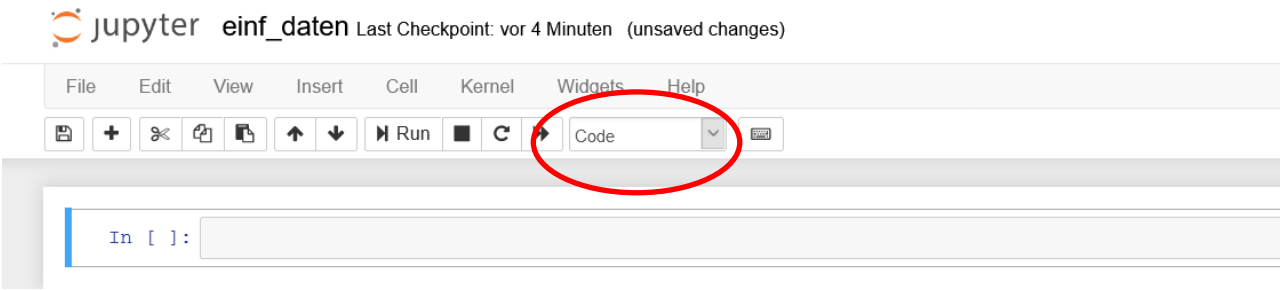
Über den Menüpunkt New kann ein neues Notebook mit Python 3 aufgerufen werden. Ein neues Notebook hat zu Beginn die folgende Form:



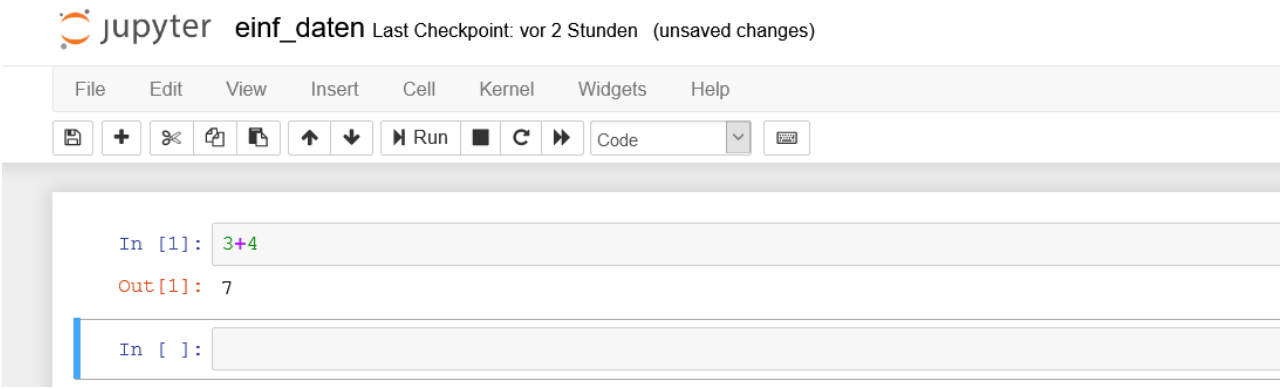
Speichern Sie ihr Jupyter Notebook unter dem Namen einf_daten.jpynb in einem entsprechenden Laufwerk ab.

Ein Jupyter Notebook stellt innerhalb einer Webanwendung alle Inhalte eines Python Programms dar, dazu gehören Eingabedaten, Ausgabedaten, erklärender Text, mathematische Formeln oder Bilder. Die unterschiedlichen Eingabefelder können, wie folgt beschrieben werden:

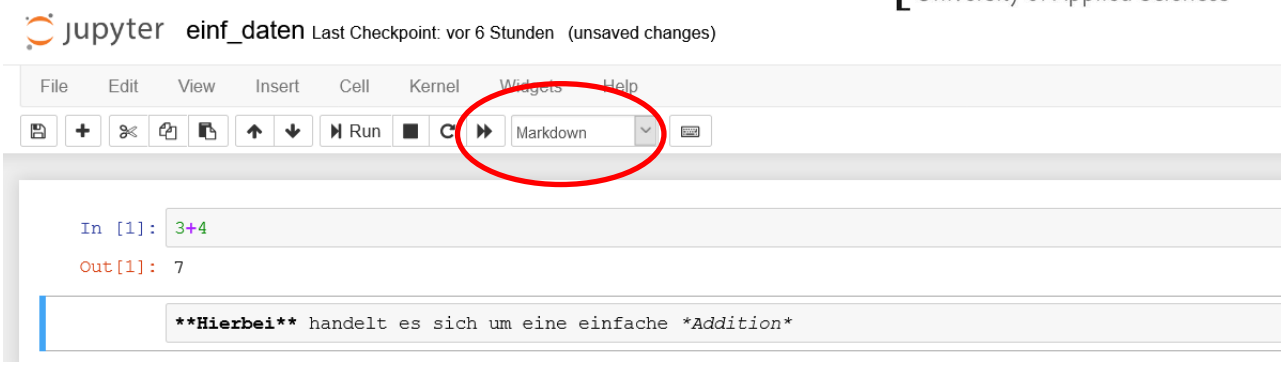
Code Cells: Eine Code Cell erlaubt es Code zu schreiben und darzustellen und später auch auszuführen.



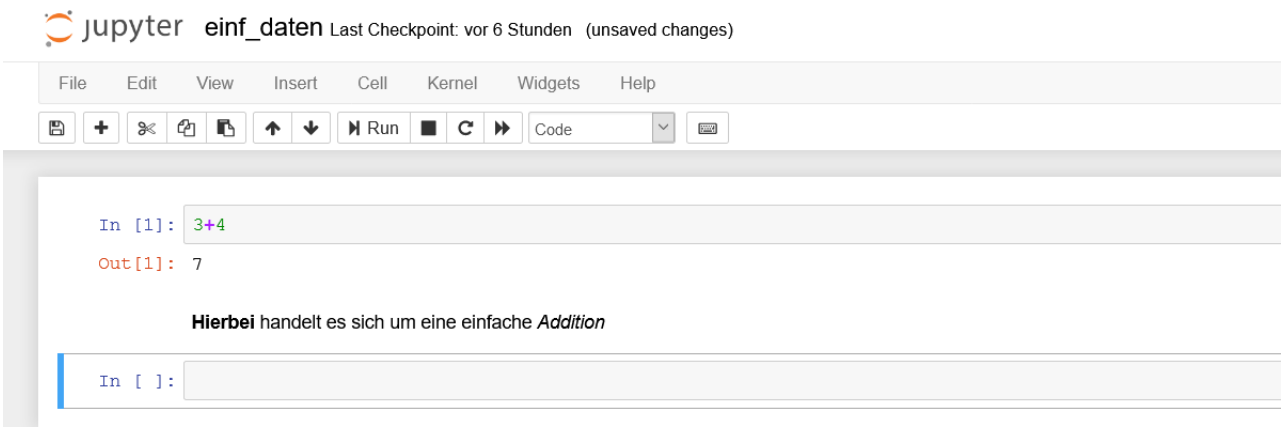
Das Resultat einer Code Cell wird nach der Ausführung über den Run Button in einer Output Cell dargestellt.



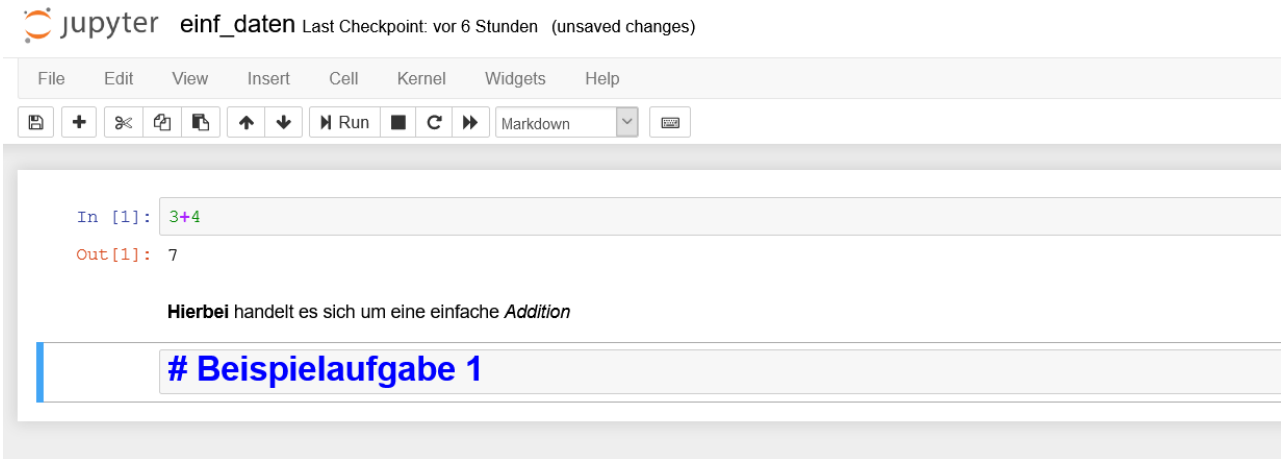
Markdown Cells: In einer Markdown Cell können Text, Erklärung und Erläuterungen stehen.

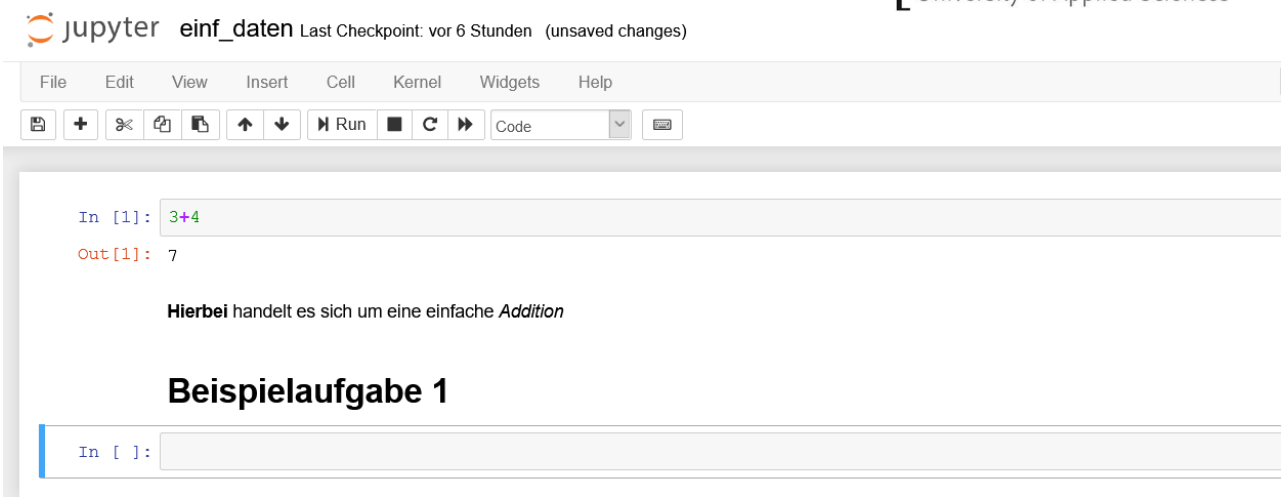


Durch Betätigen des **Run** Buttons werden sie in normalen Text umgewandelt.

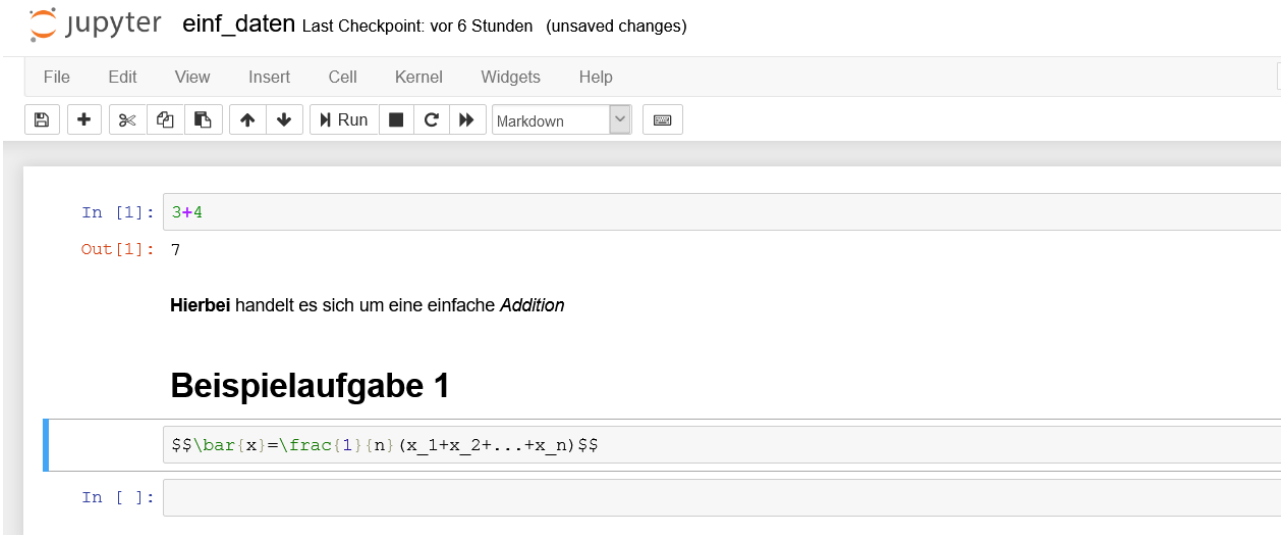


Um Überschriften in das **Jupyter** Notebook einzubinden können in **Markdown Cells** auch 1 bis 6 Hashtags # verwendet werden.

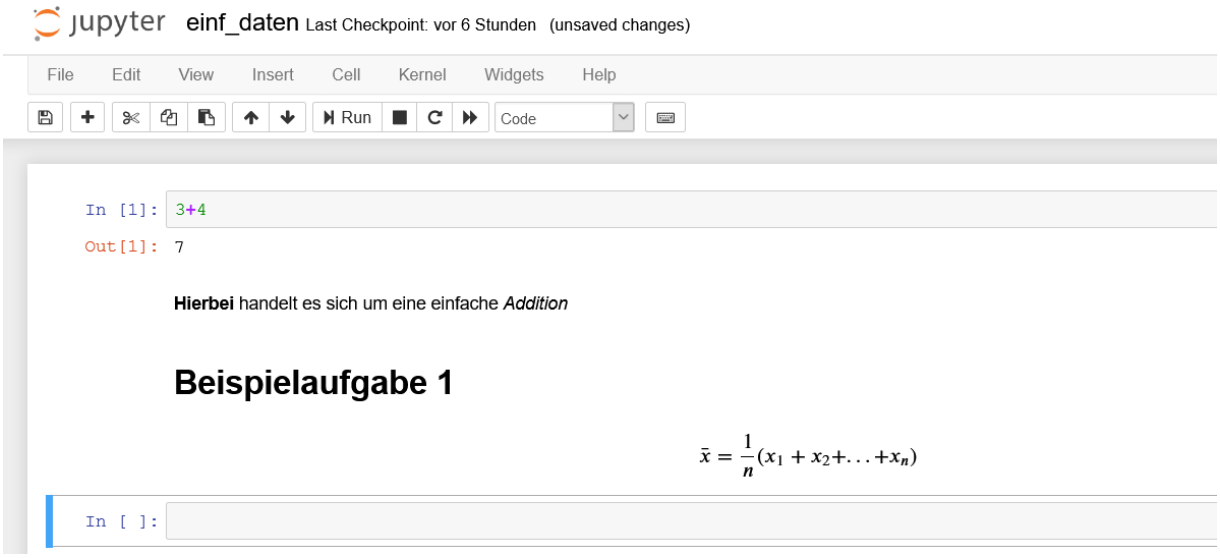




Außerdem können für mathematische Formeln auch LaTeX-Befehle verwendet werden.



Durch Drücken des **Run** Buttons erhält man die folgende Ausgabe.



Im Folgenden werden vier Beispielaufgaben behandelt, deren Bearbeitung ihnen auch für die eigentlichen Laboraufgaben helfen kann.

Beispiel 1
In diesem Beispiel sollen die Noten und Punktzahlen einer Klausur ausgewertet werden. Die dazu erforderlichen Daten liegen noch in der Datei einf_b1.txt in drei Feldern (laufende Nummer, Punktzahl, Note), die durch Leerzeichen getrennt sind. Die Daten sollen in das Jupyter Notebook einf_daten.ipynb geladen werden und dort weiterbearbeitet werden. Folgende Schritte sollen durchgeführt werden:

Textdatei einlesen

a) Öffnen Sie das Notebook einf_daten.ipynb

b) Lesen Sie die Daten aus der Datei einf_b1.txt in das Notebook ein. Verwenden Sie dazu die Bibliothek Pandas.

Die Daten werden in einem sogenannten pandas.DataFrame gespeichert. Es kann in Python wie eine Tabelle verwendet werden, es hat Spaltennamen und kann unterschiedliche Daten Typen speichern. Auf die einzelnen Elemente kann unkompliziert zugegriffen werden.

Das Ergebnis sieht wie folgt aus:

```
In [25]: data = pandas.read_csv('C:\\Users\guehring\Documents\Vorlesungen\SoSe_2021\Labor Statistik\einf_b1.txt',
                                sep=' ',
                                na_values=".",
                                header=None, names=['Nr.', 'Punkte','Note'])

In [26]: data
```

Out[26]:

	Nr.	Punkte	Note
0	1	110	1
1	2	90	1,7
2	3	108	1
3	4	95	1,3
4	5	80	2,3
5	6	74	2,7
6	7	52	3,7
7	8	78	2,3
8	9	39	4,7
9	10	102	1
10	11	79	2,3
11	12	88	1,7
12	13	58	3,3

Häufigkeiten und Prozentanteile berechnen

c) Berechnen Sie die Häufigkeiten der Noten „sehr gut“ (1,0 und 1,3), „gut“ (1,7 bis 2,3), „befriedigend“ (2,7 bis 3,3), „ausreichend“ (3,7 und 4,0) und „mangelhaft“ (4,7 und 5,0).

Zunächst wird eine Notenintervallskala festgelegt, die auch mit den entsprechenden sprechenden Notenausdrücken belegt wird. Noten_bin gibt hier die Grenzen der jeweiligen Notenunterteilung an. Mit der Funktion cut() werden die sprechenden Noten den einzelnen Notenintervallen zugeteilt.

```
In [21]: Noten_bin=[0,1.3,2.3,3.3,4.0,5.0]
         Noten=['sehr gut', 'gut', 'befriedigend', 'ausreichen', 'mangelhaft']
         Stud_Noten = pandas.cut(data['Note'], Noten_bin, labels=Noten)
         data['Schreibnoten']=Stud_Noten
         data
```

Out[21]:

	Nr.	Punkte	Note	Schreibnoten
0	1	110	1.0	sehr gut
1	2	90	1.7	gut
2	3	108	1.0	sehr gut
3	4	95	1.3	sehr gut
4	5	80	2.3	gut
5	6	74	2.7	befriedigend
6	7	52	3.7	ausreichen
7	8	78	2.3	gut
8	9	39	4.7	mangelhaft
9	10	102	1.0	sehr gut
10	11	70	2.3	gut

Die Funktion value_counts() kann dann die einzelnen Notenwerte abzählen.

```
In [22]: pandas.value_counts(data['Schreibnoten'], sort=False)
```

```
Out[22]: sehr gut        10
         gut             20
         befriedigend     9
         ausreichen       2
         mangelhaft       1
         Name: Schreibnoten, dtype: int64
```

Um weiter auf diese Häufigkeitstabelle zu zugreifen, sollte sie wieder in ein DataFrame umgewandelt werden.

```
In [6]: pandas.value_counts(data['Schreibnoten'], sort=False).rename_axis('Schulnoten').reset_index(name='Häufigkeit')
```

Out[6]:

	Schulnoten	Häufigkeit
0	sehr gut	10
1	gut	20
2	befriedigend	9
3	ausreichend	2
4	mangelhaft	1

d) Ergänzen Sie bei c) die Summe.

```
In [12]: Hf=pandas.value_counts(data['Schreibnoten'], sort=False).rename_axis('Schulnoten').reset_index(name='Häufigkeit')
         Total=Hf['Häufigkeit'].sum()
         Hf.loc[5]=['Summe', Total]
         Hf
```

Out[12]:

	Schulnoten	Häufigkeit
0	sehr gut	10
1	gut	20
2	befriedigend	9
3	ausreichend	2
4	mangelhaft	1
5	Summe	42

e) Berechnen Sie, welche prozentualen Anteile auf die Notenstufen „sehr gut“ bis „mangelhaft“ entfallen. Geben Sie die Prozentzahlen als ganze Zahlen (ohne Nachkommastellen) an.

Es wird zunächst eine neue Spalte mit der Überschrift „rel. Häufigkeit in %“ hinzugefügt.

```
In [21]: Hf['rel. Häufigkeit in %']=Hf['Häufigkeit']/Total*100
```

Dann wird diese Spalte noch auf das richtige Format gebracht mit der Funktion map().

```
]: Hf['rel. Häufigkeit in %'] = Hf['rel. Häufigkeit in %'].map('{:,.0f}'.format)
   Hf
```

]:

	Schulnoten	Häufigkeit	rel. Häufigkeit in %
0	sehr gut	10	24
1	gut	20	48
2	befriedigend	9	21
3	ausreichend	2	5
4	mangelhaft	1	2
5	Summe	42	100

f) Säulendiagramm erstellen

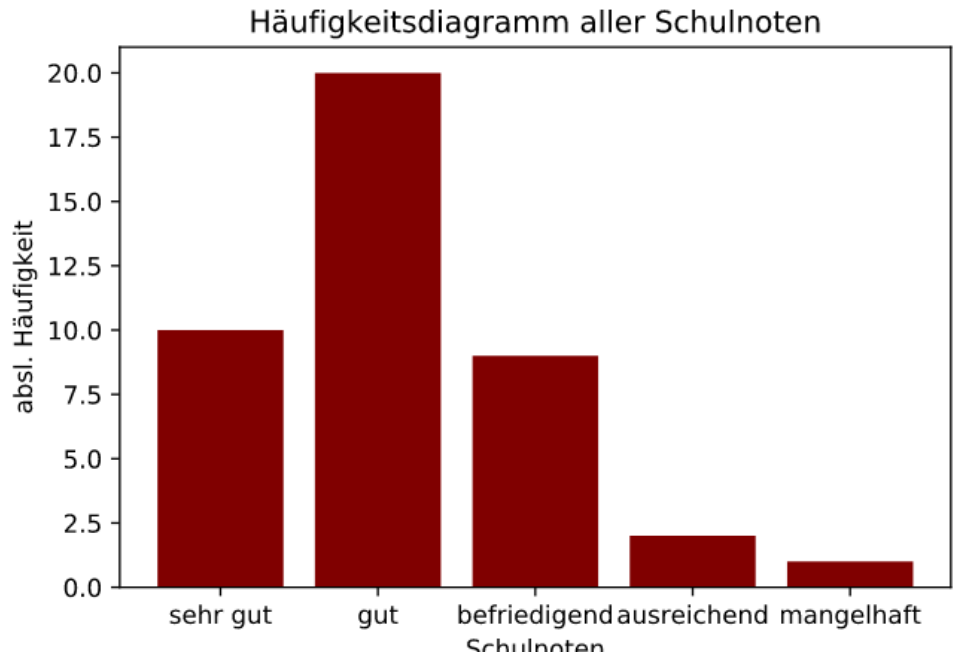
Stellen Sie die Häufigkeiten aus c) mit einem Säulendiagramm dar. Geben Sie dem Diagramm einen passenden Titel und beschriften Sie die Achsen.

Um Diagramme zu zeichnen wird die Python Bibliothek matplotlib und dort die Unterbibliothek matplotlib.pyplot verwendet. Sie beinhaltet eine Sammlung von Funktionen, die speziell zur grafischen Darstellung von Daten verwendet werden können (z.B. Grafiken darstellen, Teilfunktionsgebiete erstellen, Linien zeichnen, Bezeichnungen hinzufügen, etc).

Für eine bessere Auflösung der Daten in der Grafik empfiehlt sich der Befehl:

%config InlineBackend.figure_format='svg'

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_format='svg'
plt.bar(Hf['Schulnoten'][0:5], Hf['Häufigkeit'][0:5], color='maroon')
plt.xlabel('Schulnoten')
plt.ylabel('absl. Häufigkeit')
plt.title('Häufigkeitsdiagramm aller Schulnoten')
plt.show()
```

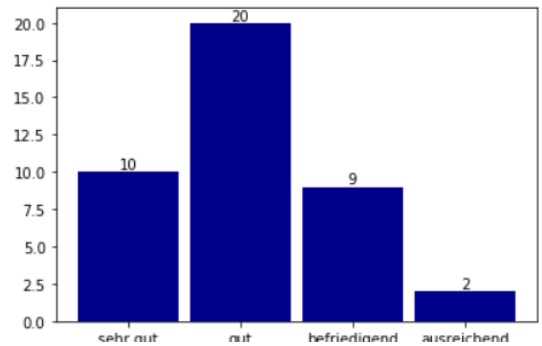


g) Führen Sie in dem Diagramm die folgenden Umformatierungen durch:

g1) Die Farbe der Säulen soll dunkelblau sein.

g2) Jede Säule soll mit der zugehörigen Häufigkeit (wie oft gab es diese Note?) beschriftet sein.

```
In [14]: plt.bar(Hf['Schulnoten'][0:4], Hf['Häufigkeit'][0:4],width=0.9,color='darkblue')
         for i in range(len(Hf['Häufigkeit'][0:4])):
             plt.annotate(str(Hf['Häufigkeit'][i]),xy=(Hf['Schulnoten'][i],Hf['Häufigkeit'][i]),
                          ha='center',va='bottom')
         plt.show()
```



Kennzahlen berechnen

h) Berechnen Sie Mittelwert, empirische Varianz und empirische Standardabweichung

der Punktzahlen und geben Sie sie mit 4 Nachkommastellen an.

```
In [10]: print('Mittelwert: %8.4f, Standardabweichung: %8.4f, Varianz: %8.4f'
              %(data['Punkte'].mean(), data['Punkte'].var(), data['Punkte'].std()))

         Mittelwert:  82.2381, Standardabweichung: 250.2834, Varianz:  15.8203
```

i) Berechnen Sie Median und Spannweite der Punktzahlen, ohne die Punktzahlenliste zu sortieren. (Spannweite = größter Datenwert minus kleinster Datenwert.)

```
print('Median: %8.4f, Spannweite: %8.4f'
      %(data['Punkte'].median(), data['Punkte'].max()-data['Punkte'].min()))

Median:  83.5000, Spannweite:  71.0000
```

j) Speichern Sie die geänderte Datei einf_daten.xls in Ihr persönliches Verzeichnis ab.

Beispiel 2

Die Daten, die diesem Beispiel zugrunde liegen, sind Angaben über die Weltproduktion von Mais (Körnermais) in Millionen Tonnen. Sie stehen in der Datei Maisproduktion.txt in Moodle im Ordner Einführung (Quelle der Daten: Deutsches Maiskomitee; Stand: Oktober 2005.).

Streudiagramm zeichnen

a) Erstellen Sie ein Streudiagramm der Maisdaten. Legen Sie dabei die Jahreszahlen auf die x-Achse und die Maisproduktion auf die y-Achse. Geben Sie dem Diagramm einen passenden Titel und beschriften Sie Achsen.

Zunächst werden die Daten in eingelesen.

```
import pandas
mais_daten=pandas.read_csv('C:\\Users\guehring\Documents\Vorlesungen\SoSe_2021\Labor_Statistik\Maisproduktion_ohneText
                           sep="\t")
mais_daten
```

	Jahr	Maisproduktion	Unnamed: 2	Unnamed: 3
0	1960.0	220.0	NaN	NaN
1	1970.0	300.0	NaN	NaN
2	1980.0	420.0	NaN	NaN
3	1990.0	520.0	NaN	NaN
4	2000.0	590.0	NaN	NaN
5	2001.0	614.0	NaN	NaN
6	2002.0	602.0	NaN	NaN
7	2003.0	640.0	NaN	NaN
8	2004.0	705.0	NaN	NaN
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN

Dabei waren in der Textdatei wohl noch weitere Spalten und Zeilen durch die Tab-Taste getrennt. Sie werden im Anschluss wieder aus dem Pandas DataFrame gelöscht. Es wird zunächst die Methode del verwendet, die zur Standarddistribution von Python gehört.

```
del mais_daten['Unnamed: 2']
del mais_daten['Unnamed: 3']
```

Mit der Pandas Funktion drop() ist es ebenso möglich, sie wird hier zum Löschen der unnötigen Zeilen verwendet.

```
mais_daten=mais_daten.drop(mais_daten.index[10])
mais_daten=mais_daten.drop(mais_daten.index[9])
```

```
mais_daten
```

	Jahr	Maisproduktion
0	1960.0	220.0
1	1970.0	300.0
2	1980.0	420.0
3	1990.0	520.0
4	2000.0	590.0
5	2001.0	614.0
6	2002.0	602.0
7	2003.0	640.0
8	2004.0	705.0

Die Zahlen in der Tabelle sollten anschließend noch so formatiert werden, dass die Jahre als Integer ohne Nachkommastellen dargestellt werden.

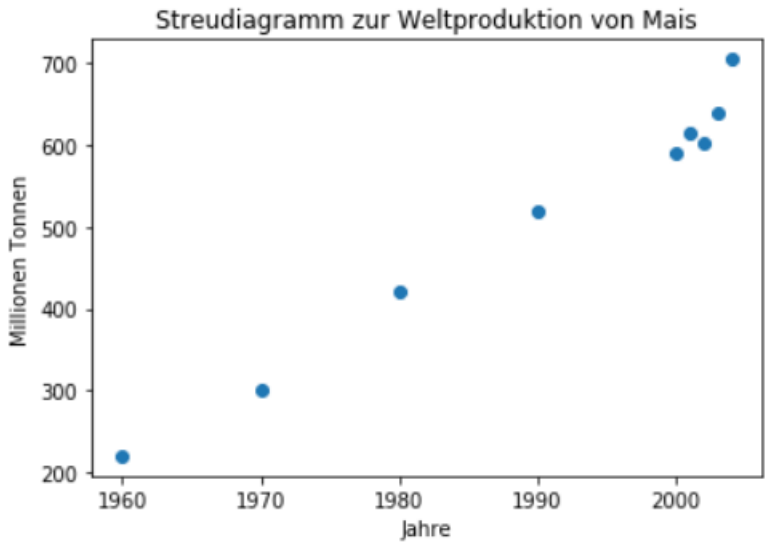
```
mais_daten['Jahr']=mais_daten['Jahr'].round().astype(int)
mais_daten
```

	Jahr	Maisproduktion
0	1960	220.0
1	1970	300.0
2	1980	420.0
3	1990	520.0
4	2000	590.0
5	2001	614.0
6	2002	602.0
7	2003	640.0
8	2004	705.0

Das Streudiagramm kann dann wie folgt gezeichnet werden.

```
import matplotlib.pyplot as plt
plt.scatter(mais_daten['Jahr'], mais_daten['Maisproduktion'],marker='o')
plt.xlabel('Jahre')
plt.ylabel('Millionen Tonnen')
plt.title('Streudiagramm zur Weltproduktion von Mais')
```

```
Text(0.5, 1.0, 'Streudiagramm zur Weltproduktion von Mais')
```



Regressionsgerade einzeichnen

b) Zeichnen Sie in Ihr Diagramm aus a) die lineare Regressionsgerade ein. Geben Sie die

Gleichung der linearen Regressionsgerade und das zugehörige Bestimmtheitsmaß R^2 an

Zur Berechnung einer linearen Regressionsgerade werden zwei weitere Software-Bibliotheken NumPy und Scikit-learn verwendet. Während NumPy eine einfache Handhabung von Vektoren und Matrizen sowie viele numerische Berechnungen ermöglicht, baut Scikit-learn auf NumPy auf und bietet viele Funktionen und Algorithmen aus dem Bereich des maschinellen Lernens an. Insbesondere bietet NumPy auch Algorithmen für Regressionsverfahren an.

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

Es wird dann zunächst eine Hülle für das Regressionsmodell angelegt.

```
mais_model=LinearRegression()
```

Anschließend wird die erste Spalte des Pandas DataFrames mais_daten in ein NumPy-Array konvertiert. Nur in NumPy gibt es die Funktion reshape(), die für die Darstellung der 1. Spalte der Jahreszahlen als Matrix verwendet werden muss. Zur Berechnung der Regressionskoeffizienten verwendet Scikit-learn Matrizenrechnung, die 1. Spalte des Pandas DataFrames mais_daten muss, deshalb in eine (nx1)–Matrix konvertiert werden.

```
x=mais_daten['Jahr']
x=x.to_numpy() #konvertiert die Spalte Jahr in ein NumPy Array
x=x.reshape(-1,1)
```

Mit der Funktion fit() können dann die Regressionskoeffizienten berechnet werden. Sie liefert zwei Attribute als Rückgabewerte intercept_ für den Achsenabschnitt und coef_ für die Steigung.

```
mais_model.fit(x, mais_daten['Maisproduktion'])
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=
False)
```

Mit dem Befehl print() und dem String-Modulo Operator kann die folgende Ausgabe erzeugt werden:

```
In [89]: print('Die Gleichung der Regressionsgeraden lautet: y = %4.2ft%4.2f'
               %(mais_model.coef_, mais_model.intercept_))
```

```
Die Gleichung der Regressionsgeraden lautet: y = 9.98t-19357.77
```

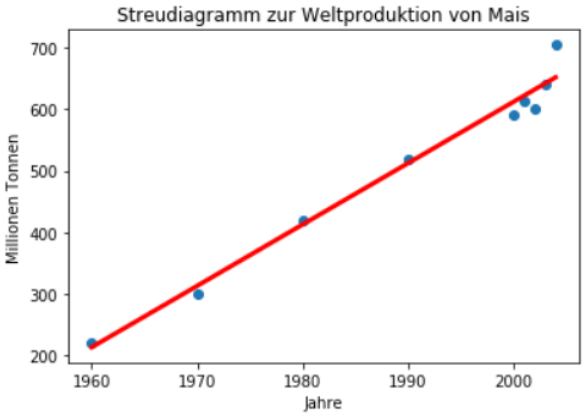
Mit score() erhält man das zugehörige R^2.

```
r_sqr=mais_model.score(x,mais_daten['Maisproduktion'])
print('Das Bestimmtheitsmaß lautet: %1.4f' % r_sqr)
```

```
Das Bestimmtheitsmaß lautet: 0.9788
```

Um die Regressionsgerade in das Diagramm zu zeichnen verwendet man die Funktion predict() angewendet auf das zuvor spezifizierte Modell. Indie Funktion predict() werden die x-Werte der Regression eingesetzt um die y-Werte der Regressionsgeraden zu erhalten.

```
]: plt.scatter(mais_daten['Jahr'], mais_daten['Maisproduktion'],marker='o')
   plt.xlabel('Jahre')
   plt.ylabel('Millionen Tonnen')
   plt.title('Streudiagramm zur Weltproduktion von Mais')
   mais_predict=mais_model.predict(x)
   plt.plot(x,mais_predict,color='red',linewidth=3)
]: [<matplotlib.lines.Line2D at 0x21f5515d048>]
```



Korrelationskoeffizienten berechnen

c) Berechnen Sie den empirischen Korrelationskoeffizienten *r* zwischen Jahr und produzierter Maismenge. Geben Sie *r* mit 4 Nachkommastellen an.

Zur Berechnung des Korrelationskoeffizienten kann man die NumPy Funktion corrcoef() verwenden. Ihre Ausgabe besteht standardmäßig aus einer ganzen Korrelationsmatrix. Den Korrelationskoeffizienten erhält man z.B. durch Auswahl der 0. Zeile und 1. Spalte der Korrelationsmatrix.

```
: print('Der Korrelationskoeffizient lautet: %1.4f'
        %np.corrcoef(mais_daten['Jahr'], mais_daten['Maisproduktion'])[0,1])
  Der Korrelationskoeffizient lautet: 0.9894
```

d) Speichern Sie die geänderte Datei einf_daten.ipynb in Ihr persönliches Verzeichnis ab.

Andere Regressionskurven ausprobieren

e) Ändern Sie den Typ der Regressionskurve von linear in quadratisch.

Um die Regressionskurve von linear in quadratisch umzuändern, wird die Scikit-learn Funktion PolynomialFeatures() verwendet. Sie wandelt Daten in ihre Potenzen um. Dabei handelt es sich um ein Preprocessing der Daten.

```
from sklearn.preprocessing import PolynomialFeatures
quad_features = PolynomialFeatures(degree=2)
quad_mais=quad_features.fit_transform(x)
```

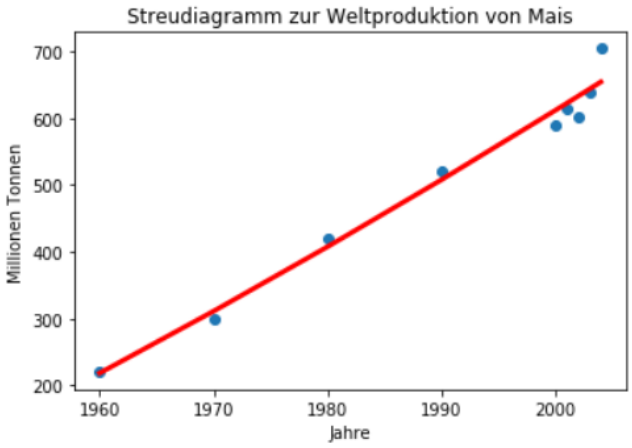
Mit den neuen Daten wird jetzt die Regressionskurve $y = a_0 + a_1 t + a_2 t^2$ geschätzt.

```
quad_mais_model = LinearRegression()
quad_mais_model.fit(quad_mais, mais_daten['Maisproduktion'])

  # predicting on quadratic polynomial
quad_mais_predict = quad_mais_model.predict(quad_mais)
plt.scatter(mais_daten['Jahr'], mais_daten['Maisproduktion'],marker='o')
plt.xlabel('Jahre')
plt.ylabel('Millionen Tonnen')
plt.title('Streudiagramm zur Weltproduktion von Mais')
plt.plot(x,quad_mais_predict,color='red',linewidth=3)

r_sqr_q=quad_mais_model.score(quad_mais,mais_daten['Maisproduktion'])
print('Das Bestimmtheitsmaß lautet: %1.4f' % r_sqr_q)
```

```
Das Bestimmtheitsmaß lautet: 0.9792
```



f) Das Bestimmtheitsmaß R^2 gibt an, wie gut die Regressionskurve die Punktewolke beschreibt (0 = gar nicht, 1 = alle Datenpunkte liegen auf der Regressionskurve). Bei quadratischer Regression ist R^2 größer als bei linearer Regression. Warum ist bei diesem Datensatz trotzdem eine lineare Regression sinnvoller als eine quadratische?

Die quadratische Regression liefert zwar ein besseres Bestimmtheitsmaß R^2, es sind aber mehr Parameter zu schätzen, die Regressionsgleichung wird damit komplizierter. Außerdem müssen mit der gleichen Anzahl von Daten mehr Parameter geschätzt werden.

g) Probieren Sie außerdem eine Regression mit einem Polynom sechsten Grades. Was stellen Sie hier fest?

Um eine Regression mit einem Polynom 6. Grades mit den vorgegebenen Datenpunkte durchzuführen ebenfalls mit der Scikit-learn Funktion PolynomialFeatures() gearbeitet werden, man ersetzt im Code unter e) in der Funktion PolynomialFeatures() den Wert für degree von 2 durch 6. Auf diese Art und Weise erhält man ein Polynom vom Grad 6 mit R^2 =0,9800 das an die Punktewolke mit dem kleinsten quadratischen Abstand angepasst wird.

Ein etwas anderes Polynom 6. Grades, das an die Punktewolke angepasst wird,

erhält man durch Normalisierung bzw. Skalierung der Eingangsparameter, also der Jahreszahlen. In diesem Fall bedeutet eine Skalierung der Jahreszahlen, dass der Mittelwert μ der angegebenen Jahreszahlen und die Standardabweichung σ der angegebenen Jahreszahlen berechnet wird und jede Jahreszahl x durch $x_{scaled} = \frac{x-\mu}{\sigma}$ ersetzt wird.

Das Polynom 6. Grades wird dann auf Basis der Werte x_{scaled} geschätzt. In Python ist das Skalieren der Inputwerte mit StandardScaler() aus der Unterbibliothek Preprocessing von Scikit-learn möglich.

```
:  from sklearn.preprocessing import StandardScaler
   scaler = StandardScaler()
   x_scaled = scaler.fit_transform(x)
```

Schätzt man basierend auf diesen skalierten Werten ein Polynom 6. Grades, so erhält man ein R^2 von 0,9982.

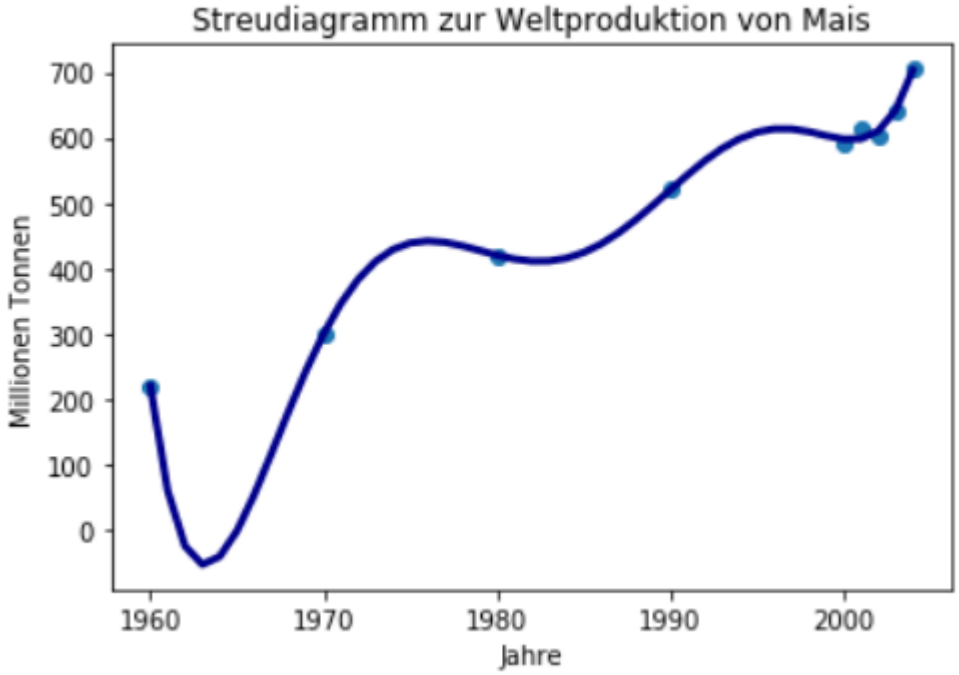
```
2]: from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    x_scaled = scaler.fit_transform(x)
    six_mais = PolynomialFeatures(degree=6, include_bias=False).fit_transform(x_scaled)
    six_mais_model = LinearRegression()
    six_mais_model.fit(six_mais, mais_daten['Maisproduktion'])
    #Zeichnen des Diagramms auf allen Daten von 1960 bis 2005
    y=np.arange(1960,2005).reshape(-1,1)
    y_scaled =(y-np.mean(x))/np.std(x)
    six_mais_y = PolynomialFeatures(degree=6, include_bias=False).fit_transform(y_scaled)
      # predicting on polynomial of sixth order
    six_mais_predict = six_mais_model.predict(six_mais_y)
    plt.scatter(mais_daten['Jahr'], mais_daten['Maisproduktion'],marker='o')
    plt.xlabel('Jahre')
    plt.ylabel('Millionen Tonnen')
    plt.title('Streudiagramm zur Weltproduktion von Mais')
    #plt.plot(x,mais_predict,color='red',linewidth=3)
    #plt.plot(x,quad_mais_predict,color='green',linewidth=3)
    plt.plot(y,six_mais_predict,color='darkblue',linewidth=3)

    r_sqr_s=six_mais_model.score(six_mais,mais_daten['Maisproduktion'])
    print('Das Bestimmtheitsmaß lautet: %1.4f' % r_sqr_s)
```

```
Das Bestimmtheitsmaß lautet: 0.9982
```

Um das geschätzte Polynom auf allen Werten von 1960 bis 2004 zu zeichnen, wurde in obigem Code noch der Vektor y definiert, der alle Werte von 1960 bis 2005 annimmt. Für eine Prognose dieser Daten muss auch y mit μ und σ skaliert werden.

Man erhält das folgende Bild:



Obwohl hier R^2 noch größer ist als bei der quadratischen Regression, sieht man auch, dass durch die Regression sehr leicht ein Overfittung bzw. eine Nonsens Regression zustande kommt.

Eine nichtlineare Regression kann auch mit der Scikit-learn Bibliothek optimize und dort der Funktion curve_fit() durchgeführt werden. Auch hierfür können die x-Werte skaliert werden.

```
]: from scipy.optimize import curve_fit
   import math
   def f(x,a0,a1,a2,a3,a4,a5,a6): return a0+a1*x+a2*x**2+a3*x**3+a4*x**4+a5*x**5+a6*x**6
   scaler = StandardScaler()
   x_scaled = scaler.fit_transform(x)
   coefs6,cov6=curve_fit(f,(mais_daten['Jahr']-np.mean(mais_daten['Jahr']))/np.std(mais_daten['Jahr
                         mais_daten['Maisproduktion'])
   coefs6
```

```
]: array([ 519.85571635,  362.08072413,  -24.42071974, -814.45277321,
        -118.40802546,  664.9165609 ,  271.35441476])
```

Als Ergebnis der Methode erhält man sowohl die Koeffizienten der zugehörigen Funktion, als auch eine Kovarianzmatrix der Koeffizienten.

Leider liefert die Funktion curve_fit() kein R^2, das Bestimmtheitsmaß muss berechnet werden. Dazu muss die 2. Spalte der Tabelle mais_daten wieder auf eine (nx1)-Matrix transferiert werden.

```
residuals=mais_daten['Maisproduktion'].to_numpy().reshape(-1,1)-f(x_scaled,*coefs6)
ss_res = np.sum(residuals**2)
ss_tot = np.sum((mais_daten['Maisproduktion']-np.mean(mais_daten['Maisproduktion']))**2)
r_squared = 1 - (ss_res / ss_tot)
```

```
r_squared
```

```
0.9982251406121202
```

Beispiel 3
In diesem Beispiel lernen Sie einige Statistik-Funktionen kennen, mit denen man Berechnungen der wichtigsten **diskreten Wahrscheinscheinlichkeitsverteilungen** (hypergeometrische Verteilung, Binomialverteilung, Poissonverteilung) durchführen kann. Genaueres über diese Verteilungen erfahren Sie in der Vorlesung.

Hypergeometrische Verteilung

a) Sie erhalten eine Lieferung von 50 elektronischen Bauteilen. Daraus entnehmen Sie eine Stichprobe von 20 Bauteilen und testen diese 20 Bauteile auf Funktionsfähigkeit. Die Zufallsvariable X gebe die Anzahl der defekten Bauteile unter den 20 Bauteilen der Stichprobe an. Angenommen, in der Lieferung sind 5 defekte elektronische Bauteile. Unter diesen Annahmen folgt X einer so genannten hypergeometrischen Verteilung $X \sim H(20; 50; 5)$. Berechnen Sie hierfür

a1) die Wahrscheinlichkeit, dass in Ihrer Stichprobe kein defektes Bauteil ist;

Um Werte für statistische Verteilungen zu berechnen soll die Statistik Bibliothek scipy.stats verwendet werden. Danach wird zunächst eine hypergeometrische Verteilung mit den entsprechenden Parametern angelegt.

```
from scipy.stats import hypergeom
rv = hypergeom(50, 5, 20)
```

Die geforderten Wahrscheinlichkeiten erhält man, indem man die entsprechende Wahrscheinlichkeitsverteilung (engl. *probability mass function*) .pmf() auswertet.

```
[53]: print('Wahrscheinlichkeit, dass kein Bauteil der Stichprobe defekt ist: p=%1.4f'
            % rv.pmf(0))

      Wahrscheinlichkeit, dass kein Bauteil der Stichprobe defekt ist: p=0.0673
```

a2) die Wahrscheinlichkeit, dass in Ihrer Stichprobe genau 1 defektes Bauteil ist;

```
[55]: print('Wahrscheinlichkeit, dass 1 Bauteil der Stichprobe defekt ist: p=%1.4f'
            % rv.pmf(1))

      Wahrscheinlichkeit, dass 1 Bauteil der Stichprobe defekt ist: p=0.2587
```

a3) die Wahrscheinlichkeit, dass in Ihrer Stichprobe genau 2 defekte Bauteile sind;

```
[56]: print('Wahrscheinlichkeit, dass 2 Bauteile der Stichprobe defekt sind: p=%1.4f'
            % rv.pmf(2))

      Wahrscheinlichkeit, dass 2 Bauteile der Stichprobe defekt sind: p=0.3641
```

a4) die Wahrscheinlichkeit, dass in Ihrer Stichprobe genau 3 defekte Bauteile sind;

```
[57]: print('Wahrscheinlichkeit, dass 3 Bauteile der Stichprobe defekt sind: p=%1.4f'
            % rv.pmf(3))

      Wahrscheinlichkeit, dass 3 Bauteile der Stichprobe defekt sind: p=0.2341
```

a5) die Wahrscheinlichkeit, dass in Ihrer Stichprobe höchstens 3 defekte Bauteile sind.

An dieser Stelle werden kumulative Wahrscheinlichkeiten oder die Wahrscheinlichkeitsverteilung (engl. *cumulative distribution function*) .cdf() benötigt.

```
[58]: print('Wahrscheinlichkeit, dass höchstens 3 Bauteile der Stichprobe defekt sind: p=%1.4f'
            % rv.cdf(3))

      Wahrscheinlichkeit, dass höchstens 3 Bauteile der Stichprobe defekt sind: p=0.9241
```

Binomialverteilung

b) Bei der Massenproduktion bestimmter elektronischer Kleinteile entsteht eine Ausschussquote von 10 %. Sie entnehmen der laufenden Produktion eine Stichprobe vom Umfang 20. Man kann davon ausgehen, dass hierbei verschiedene Stichprobenteile unabhängig voneinander defekt sind. Die Zufallsvariable X gebe die Anzahl der defekten Kleinteile unter diesen 20 Teilen an. Unter den genannten Annahmen folgt X einer so genannten Binomialverteilung $X \sim B(20; 0.1)$. Berechnen Sie hierfür

b1) die Wahrscheinlichkeit, dass in Ihrer Stichprobe genau 3 defekte Kleinteile sind;

Jetzt verwenden wir aus der Statistik Bibliothek die Binomialverteilung.

```
[60]: from scipy.stats import binom
      rv = binom(20,0.1)
```

Und benutzen dann wieder die Funktion .pmf().

```
[62]: print('Wahrscheinlichkeit, dass genau 3 elektrische Kleinteile der Stichprobe defekt sind: p=%1.
            % rv.pmf(3))

      Wahrscheinlichkeit, dass genau 3 elektrische Kleinteile der Stichprobe defekt sind: p=0.1901
```

b2) die Wahrscheinlichkeit, dass in Ihrer Stichprobe höchstens 3 defekte Kleinteile sind;

```
[63]: print('Wahrscheinlichkeit, dass höchstens 3 elektrische Kleinteile der Stichprobe defekt sind: p
            % rv.cdf(3))

      Wahrscheinlichkeit, dass höchstens 3 elektrische Kleinteile der Stichprobe defekt sind: p=0.8
      670
```

Poissonverteilung

c) Bei der Produktion einer bestimmten Textilart entstehen zufallsbedingt Gewebefehler. Im Mittel sind es 2 Gewebefehler auf 1 m^2. Sie entnehmen zufällig ein Textilstück von 1 m^2 und zählen, wie viele Gewebefehler auf diesem Stück sind. Die Zufallsvariable X gebe die Anzahl festgestellter Gewebefehler an. Unter den genannten Annahmen folgt X einer so genannten Poissonverteilung $X \sim Po(2)$, dabei ist $\lambda = 2$ der Erwartungswert von X (mittlere, d. h. erwartete Anzahl von Fehlern). Berechnen Sie hierfür

c1) die Wahrscheinlichkeit, dass auf einem Textilstück genau 3 Gewebefehler sind;

```
[65]: from scipy.stats import poisson
      rv = poisson(2)

[66]: print('Wahrscheinlichkeit, dass genau 3 Gewebefehler auf einem Textilstück sind: p=%1.4f'
            % rv.pmf(3))

      Wahrscheinlichkeit, dass genau 3 Gewebefehler auf einem Textilstück sind: p=0.1804
```

c2) die Wahrscheinlichkeit, dass auf Ihrem Textilstück höchstens 3 Gewebefehler sind.

```
[67]: print('Wahrscheinlichkeit, dass höchstens 3 Gewebefehler auf einem Textilstück sind: p=%1.4f'
            % rv.cdf(3))

      Wahrscheinlichkeit, dass höchstens 3 Gewebefehler auf einem Textilstück sind: p=0.8571
```

Beispiel 4

In diesem Beispiel lernen Sie einige Statistik-Funktionen kennen, mit denen man Berechnungen bei der wichtigsten stetigen Wahrscheinlichkeitsverteilung, nämlich der Normalverteilung, durchführen kann. Genaueres über die Normalverteilungen erfahren Sie später in der Vorlesung.

Eine Maschine füllt Zucker in Packungen. Die Füllmenge variiert zufällig. Die Zufallsvariable X gebe die Füllmenge [in g] einer zufällig ausgewählten Zuckerpackung an. Wir gehen in diesem Beispiel davon aus dass die Zufallsvariable X einer Normalverteilung X~N(1000;9) folgt. In diesem Beispiel ist also der Erwartungswert der Füllmenge μ=1.000 [g], Varianz der Füllmenge σ^2=9 [g²] und Standardabweichung der Füllmenge σ=3 [g].

a) Berechnen Sie die Wahrscheinlichkeit, dass die Füllmenge einer zufällig ausgewählten Zuckerpackung bei höchstens 994 g liegt.

```
In [1]: from scipy.stats import norm
        rv=norm(1000,3)# Python schreibt hier die Standardabweichung in die Funktion norm(), nicht die Varianz
        print('Wahrscheinlichkeit, dass höchstens 994 g in einer Zuckerpackung sind: p=%1.4f'
              % rv.cdf(994))

        Wahrscheinlichkeit, dass höchstens 994 g in einer Zuckerpackung sind: p=0.0228
```

b) Berechnen Sie das 1-%-Quantil der Normalverteilung . Das ist diejenige Füllmenge, die von einer zufällig ausgewählten Zuckerpackung nur mit einer Wahrscheinlichkeit von 0,01 unterschritten wird.

```
23]: norm.ppf(0.01,loc=1000,scale=3)
23]: 993.0209563778775

26]: print('Das 1-Prozent-Quantil der Zuckerpackungen beträgt: z= %1.4f'
           % norm.ppf(0.01, loc=1000, scale=3))
     #ppf steht für percent point function

     Das 1-Prozent-Quantil der Zuckerpackungen beträgt: z= 993.0210
```

c) Mit welcher Funktion können die Quantile der so genannten t-Verteilung berechnen kann? (Die Quantile der t-Verteilung werden in Kapitel 5 der Vorlesung genauer erläutert.)

```
In [26]: print('Das 1-Prozent-Quantil der Zuckerpackungen beträgt: z= %1.4f'
               % norm.ppf(0.01, loc=1000, scale=3))
         #ppf steht für percent point function

Das 1-Prozent-Quantil der Zuckerpackungen beträgt: z= 993.0210
```

d) Zeichnen Sie die Dichtefunktion der Zufallsvariablen X im Intervall [990, 1010].

```
In [19]: import matplotlib.pyplot as plt
         import numpy as np
         x_axis = np.arange(990, 1010, 0.05)
         plt.plot(x_axis, 100*norm.pdf(x_axis, 1000, 3),color='red')

Out[19]: [<matplotlib.lines.Line2D at 0x258d6dc7520>]
```