Übung Hooke'sches Gesetz

Inhaltsverzeichnis

Häufig liegen Sensordaten in mehreren Dateien vor. Mögliche Gründe dafür können sein, dass die Messung

- von unterschiedlichen Personen,
- an unterschiedlichen Standorten,
- zu unterschiedlichen Zeiten,
- mit verschiedenen Geräten oder
- für unterschiedliche Messgrößen durchgeführt wurden.

Im Ordner '01-daten/hooke' liegen mehrere txt-Dateien mit Messdaten zur Federausdehnung. In diesem Kapitel sollen Sie das bisher Gelernte anwenden und die folgenden Fragen beantworten.

1. Liegen ungültige Messungen vor?
2. Welche Werte können für die Federkonstanten ermittelt werden?
3. Wurden die Messungen mit der gleichen Feder durchgeführt, wenn als Vertrauenswahrscheinlichkeit 90 % bzw. 95 % angenommen werden soll?

Im Abschnitt Kapitel 0.1 finden Sie Hinweise und im Abschnitt Kapitel 0.2 eine Musterlösung zum Einlesen der Dateien. Anschließend sollen Sie die Aufgabenstellung eigenständig bearbeiten. Ab dem Abschnitt Kapitel **??** finden Sie eine Musterlösung.

Hinweis: Je nach gewähltem Vorgehen ergeben sich unterschiedliche Ergebnisse.

Dafür wäre die Einführung der Fehlerrechnung bzw. des Größtfehlers sinnvoll, weil hier auf das Vorliegen grober Fehler geprüft wird.

Wir prüfen 'automatisiert', ob mehrere Datensätze Fehlmessungen enthalten. Anschließend bestimmen wir die Federkonstanten und die Konfidenzintervalle.

- einlesen mit glob - aufgaben/01-daten
  - man könnte eine Spalte mit dem teamnamen anlegen und dann mit groupby arbeiten. Das macht auch mehr Sinn, falls es tatsächlich sehr viele Datensätze sind. :)
- z-Werte (studentisiert) nach Gewicht bestimmen
- Federkonstante im Konfidenzintervall ausgeben

0.1 Dateien einlesen

Für das Einlesen der Dateien können Sie das Modul glob verwenden (siehe **Querverweis auf m-Einlesen strukturierter Datensätze**).

Zunächst kann der Funktion `glob.glob()` im Argument `pathname = *` der Platzhalter `*` für eine beliebige Zeichenfolge (außer Dateipfadelemente wie `/` oder `.`) übergeben werden, sodass die Namen aller im angegebenen Ordner gespeicherten Dateien ausgelesen werden. Auf diese Weise kann die Anzahl der Dateien und die Dateiendung bestimmt werden, falls dies noch unbekannt ist.

```
ordnerpfad = '01-daten/hooke'

pfadliste = glob.glob(pathname = '*', root_dir = ordnerpfad, recursive = False)
print(pfadliste)
print(f"Anzahl Dateien: {len(pfadliste)}")
```

```
['team_kreativkoepfe.txt', 'team_die_ahnungslosen.txt', 'team_ma.txt', 'team_fabi.txt']
Anzahl Dateien: 4
```

Mit den Dateipfaden können die Dateien mit Hilfe einer Schleife in eine Liste eingelesen werden. Zunächst werden nur die jeweils ersten 3 Zeilen eingelesen, um einen Eindruck vom Aufbau der Dateien zu erhalten.

```
list_of_files = []
for pfad in pfadliste:
  zwischenspeicher = pd.read_csv(filepath_or_buffer = ordnerpfad + '/' + pfad, nrows = 3)
  list_of_files.append(zwischenspeicher)
  print(pfad, "\n", zwischenspeicher, "\n", sep = '')
```

```
team_kreativkoepfe.txt
   10:33:02\t109.64 cm\t0
0  10:33:05\t109.62 cm\t0
1  10:33:08\t109.64 cm\t0
2  10:33:11\t109.62 cm\t0

team_die_ahnungslosen.txt
   11:03:23\t109.66 cm\t0
0  11:03:23\t109.62 cm\t0
1  11:03:23\t109.73 cm\t0
2  11:03:23\t109.55 cm\t0

team_ma.txt
   10:09:38\t109.26 cm\t0
0  10:09:41\t109.26 cm\t0
1  10:09:44\t109.28 cm\t0
2  10:09:47\t109.18 cm\t0

team_fabi.txt
   09:17:54\t110.31 cm\t0
0  09:17:57\t110.29 cm\t0
1  09:18:03\t110.74 cm\t0
2  09:18:06\t109.95 cm\t0
```

Die Dateien beinhalten keine Spaltenbeschriftung und verwenden den Tabulator ”\t“ als Trennzeichen. Die erste Spalte enthält einen Zeitstempel, die zweite die gemessene Federausdehnung und die dritte (vermutlich) das angehängte Gewicht.

0.2 Aufgabe Dateien einlesen

Lesen Sie die Dateien nun ein. Prüfen Sie dabei:

- ob die Datentypen korrekt eingelesen werden und
- auf fehlende Werte.

Sie können:

a) Jede Datei einzeln einlesen.
b) Mit dem Modul glob die Dateien automatisch einlesen und jeweils in einem separaten Objekt speichern (*Hinweis:* Dieses Vorgehen wird im **Methodenbaustein Einlesen strukturierter Datensätze** gezeigt).
c) Die Dateien mit dem Modul glob automatisch einlesen und zu einer Datei zusammenführen.

Die verschiedenen Möglichkeiten sind mit zunehmend mehr Aufwand beim Programmieren verbunden. Je mehr separate Dateien Sie auswerten möchten, desto mehr Automatisierung ist gefragt. Da bei der Auswertung von Sensordaten häufig zahlreiche Dateien ausgewertet werden müssen, wird in der Musterlösung Variante c) gezeigt.

Schrittweises Vorgehen

Das Einlesen der Dateien wird voraussichtlich der aufwändigste und fehleranfälligste Arbeitsschritt sein. Entwickeln Sie Ihre Lösung Schritt für Schritt. Beginnen Sie mit der Variante a). Wenn Sie die Dateien eingelesen haben, können Sie sich durch die Weiterentwicklung zur Variante b) mit dem Modul glob vertraut machen. Darauf aufbauend können Sie mit der Variante c) die Automatisierung für beliebig viele Dateien umsetzen.

Musterlösung Dateien einlesen

Der erste Versuch, die Dateien einzulesen, scheitert mit einer Fehlermeldung. Die Anweisungen werden deshalb in die Struktur zur Ausnahmebehandlung eingebettet und die verursachende Datei abgefangen. (Sollten mehrere Dateien Fehler aufwerfen, müssten die Dateien in einer Liste gespeichert und später - falls möglich - mit einer Schleife weiter behandelt werden.)

```
hooke = pd.DataFrame(columns = ['Zeit', 'Abstand', 'Gewicht', 'Team']) # ein leerer DataFra

for pfad in pfadliste:

  try:
    zwischenspeicher = pd.read_csv(filepath_or_buffer = ordnerpfad + '/' + pfad, sep = '\t

    # Dateiname als Spalte einfügen
    zwischenspeicher['Team'] = pfad[5:-4]

    hooke = pd.concat([hooke, zwischenspeicher], ignore_index = True)

  except Exception as error:
    print(pfad, error, sep = "\n")
    pfad_problem_datei = pfad

print(hooke.info(), "\n")
print("Erfolgreich einglesen:\n", hooke['Team'].unique(), sep = '')
```

```
team_ma.txt
Error tokenizing data. C error: Expected 3 fields in line 135, saw 4

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359 entries, 0 to 358
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Zeit     355 non-null    object
 1   Abstand  355 non-null    object
 2   Gewicht  355 non-null    float64
 3   Team     359 non-null    object
dtypes: float64(1), object(3)
memory usage: 11.3+ KB
None

Erfolgreich einglesen:
['kreativkoepfe' 'die_ahnungslosen' 'fabi']
```

Der Fehlermeldung zufolge besteht Zeile 135 aus 4 statt aus 3 Spalten. Die fehlerverursachende Datei wird deshalb zeilenweise durchlaufen und jede Zeile ausgegeben, die mehr als 3 Einträge hat. Zur Kontrolle werden auch die ersten 5 Zeilen ausgegeben.

```
# einen leeren DataFrame mit 3 Spalten erstellen
df = pd.DataFrame(data = [], columns = ['Zeit', 'Abstand', 'Gewicht'])

dateiobjekt_problem_datei = open(file = ordnerpfad + '/' + pfad_problem_datei, mode = 'r')

index = 0
for zeile in dateiobjekt_problem_datei:
  try:
    zwischenspeicher = zeile.split(sep = "\t")
    if len(zwischenspeicher) > 3:
      print("Index =", index, ":", zwischenspeicher)
    elif index <= 5:
      print(zeile)
    index += 1
  except Exception as error:
    print(error)

dateiobjekt_problem_datei.close()
```

```
10:09:38    109.26 cm   0



10:09:41    109.26 cm   0



10:09:44    109.28 cm   0



Index = 134 : ['10:29:27', '105.49 cm', '300', '\n']
```

Jede zweite Zeile ist leer. In Zeile 134 wird ein Zeilenumbruch '\n' eingelesen. Die Datei wird deshalb mit einer angepassten Schleife erneut durchlaufen. Aus der betreffenden Zeile wird der zusätzliche Zeilenumbruch '\n' entfernt. Leere Zeilen werden übersprungen. Die korrekten Zeilen werden an den DataFrame hooke angefügt.
Der Code muss ggf. noch angepasst werden, weil vermutlich so leere zeilen angefügt werden

```
dateiobjekt_problem_datei = open(file = ordnerpfad + '/' + pfad_problem_datei, mode = 'r')

for zeile in dateiobjekt_problem_datei:
  try:
    zwischenspeicher = zeile.split(sep = "\t")
    if len(zwischenspeicher) > 3:
      zwischenspeicher = zwischenspeicher[:3]
    elif len(zwischenspeicher) < 3: # leere Zeilen überspringen
      continue
    # Dateinamen anfügen
    zwischenspeicher.append(pfad[5:-4])

    hooke.loc[len(hooke)] = pd.Series(zwischenspeicher).values

  except Exception as error:
    print(error)
    print(pd.Series(zwischenspeicher).values)

dateiobjekt_problem_datei.close()

print("Erfolgreich einglesen:\n", hooke['Team'].unique(), "\n", sep = '')
print(hooke.info())
```

```
Erfolgreich einglesen:
['kreativkoepfe' 'die_ahnungslosen' 'fabi']

<class 'pandas.core.frame.DataFrame'>
Index: 537 entries, 0 to 536
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Zeit     533 non-null    object
 1   Abstand  533 non-null    object
 2   Gewicht  533 non-null    object
 3   Team     537 non-null    object
dtypes: object(4)
memory usage: 21.0+ KB
None
```

Anschließend werden zum einen die Zeilen mit Nullwerten betrachtet …

```
print(hooke.loc[hooke.apply(pd.isna).any(axis = 1), :])
```

```
    Zeit Abstand Gewicht              Team
3    NaN      NaN     NaN     kreativkoepfe
23   NaN      NaN     NaN     kreativkoepfe
28   NaN      NaN     NaN     kreativkoepfe
212  NaN      NaN     NaN  die_ahnungslosen
```

… und entfernt.

```
hooke.drop(np.where(hooke.apply(pd.isna).any(axis = 1))[0], inplace = True)
```

Zum anderen werden die Datentypen kontrolliert.

- Die Zeit kann als string stehen bleiben, da sie für die Auswertung nicht benötigt wird.
- Der gemessene Abstand ist mit ' cm' notiert - diese Zeichenkette wird entfernt. Anschließend sollte die Spalte als numerisch erkannt werden.
- Das Gewicht sollte numerische Werte enthalten, wird aber als Datentyp object eingelesen und muss weiter untersucht werden.
- Der Spalte Team könnte der Pandas Datentyp category zugewiesen werden, notwendig ist es aber nicht.

String ' cm' entfernen.

```
hooke.replace(' cm', '', regex = True, inplace = True)
```

Ob alle Elemente einer Zelle numerisch sind, kann mit der Pandas-Methode `pd.Series.str.isnumeric()` überprüft werden. Ein Blick auf die Daten zeigt die Ursache.

```
print(hooke['Gewicht'].str.isnumeric().sum())

print(hooke['Gewicht'].head())
print(hooke['Gewicht'].tail())
```

```
1
0    0.0
1    0.0
2    0.0
4    0.0
5    0.0
```

```
Name: Gewicht, dtype: object
532    850\n
533    850\n
534    850\n
535    850\n
536    850\n
Name: Gewicht, dtype: object
```

Die Zeilenumbrüche werden ebenfalls entfernt.

```
hooke.replace('\n', '', regex = True, inplace = True)
print(hooke['Gewicht'].str.isnumeric().sum())
print(hooke['Gewicht'].tail())
```

```
178
532    850
533    850
534    850
535    850
536    850
Name: Gewicht, dtype: object
```

```
print(hooke.info(), "\n")

# explizite Zuweisung
hooke['Abstand'] = hooke['Abstand'].astype('float')
hooke['Gewicht'] = hooke['Gewicht'].astype('float')

print(hooke.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 533 entries, 0 to 536
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Zeit     533 non-null    object
 1   Abstand  533 non-null    object
 2   Gewicht  533 non-null    object
 3   Team     533 non-null    object
dtypes: object(4)
memory usage: 20.8+ KB
```

```
None

<class 'pandas.core.frame.DataFrame'>
Index: 533 entries, 0 to 536
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Zeit     533 non-null    object
 1   Abstand  533 non-null    float64
 2   Gewicht  533 non-null    float64
 3   Team     533 non-null    object
dtypes: float64(2), object(2)
memory usage: 20.8+ KB
None
```

Das Ergebnis könnte so aussehen:

```
hooke.groupby(by = ['Team', 'Gewicht'])['Abstand'].describe()
```

Team	Gewicht	coun
die_ahnungslosen	0.0	10.0
	50.0	8.0
	100.0	12.0
	150.0	19.0
	200.0	18.0
	250.0	18.0
	300.0	11.0
	350.0	14.0
	400.0	11.0
	450.0	12.0
fabi	0.0	20.0
	50.0	22.0
	100.0	10.0
	101.0	9.0
	150.0	11.0
	152.0	10.0
	200.0	9.0
	201.0	10.0
	250.0	10.0
	253.0	11.0

Team	Gewicht	coun
	300.0	10.0
	302.0	12.0
	350.0	10.0
	353.0	11.0
	400.0	10.0
	403.0	14.0
	450.0	10.0
	455.0	11.0
	500.0	10.0
	550.0	10.0
	600.0	10.0
	650.0	10.0
	700.0	10.0
	750.0	9.0
	800.0	11.0
	850.0	8.0
	0.0	9.0
	50.0	8.0
	100.0	10.0
	150.0	10.0
	200.0	10.0
	250.0	10.0
kreativkoepfe	300.0	10.0
	350.0	10.0
	400.0	9.0
	450.0	9.0
	500.0	8.0
	550.0	9.0

0.3 Daten aufbereiten

Im nächsten Schritt werden die Daten geprüft und ggf. bereinigt. Dies umfasst folgende Schritte:

- auf Ausreißer prüfen (studentisierte z-Werte und grafisch) und ggf. bereinigen,
- Normierung der Abstandsmessung auf den Nullpunkt und Umrechnung der verwendeten Einheiten und
- grafische Darstellung.

1. **Liegen ungültige Messungen vor?**

Auf Ausreißer prüfen

Auf Ausreißer kann (unter anderem) mit studentisierten z-Werten und grafisch geprüft werden.

0.4 Ausreißer bestimmen

```
Anzahl der studentisierten z-Werte mit Betrag  3: 0

Anzahl der studentisierten z-Werte mit Betrag  2.5: 4
         Zeit  Abstand  Gewicht              Team
79   10:45:39    97.45    350.0     kreativkoepfe
94   10:47:14    95.80    450.0     kreativkoepfe
99   10:47:29    95.37    500.0     kreativkoepfe
231  11:13:56    99.45    400.0  die_ahnungslosen

Kombinationen aus Gewicht & Team bestimmen
kreativkoepfe
79    350.0
94    450.0
99    500.0
Name: Gewicht, dtype: float64

die_ahnungslosen
231    400.0
Name: Gewicht, dtype: float64
```

0.5 Gemeinsame Ausgabe der Messreihen

```
kreativkoepfe 350.0
kreativkoepfe 450.0
kreativkoepfe 500.0
die_ahnungslosen 400.0
         Zeit  Abstand  z-Werte Abstand  Gewicht              Team
70   10:44:49   100.84        -0.554549    350.0     kreativkoepfe
71   10:44:52   100.82         1.421519    350.0     kreativkoepfe
72   10:44:55   100.77         1.378561    350.0     kreativkoepfe
73   10:45:21    98.25        -0.468633    350.0     kreativkoepfe
74   10:45:24    97.43        -0.984129    350.0     kreativkoepfe
75   10:45:27    97.79         0.906023    350.0     kreativkoepfe
76   10:45:30    99.13         0.605317    350.0     kreativkoepfe
```

77	10:45:33	97.50	-0.984129	350.0	kreativkoepfe
78	10:45:36	98.06	-1.284835	350.0	kreativkoepfe
79	10:45:39	97.45	0.605317	350.0	kreativkoepfe
89	10:46:46	96.71	-0.640465	450.0	kreativkoepfe
90	10:46:49	95.65	1.383500	450.0	kreativkoepfe
91	10:46:52	95.75	1.369910	450.0	kreativkoepfe
92	10:46:55	95.83	1.335934	450.0	kreativkoepfe
93	10:47:11	94.63	-0.376453	450.0	kreativkoepfe
94	10:47:14	95.80	-0.933659	450.0	kreativkoepfe
95	10:47:17	97.24	-0.689032	450.0	kreativkoepfe
96	10:47:20	94.12	0.221523	450.0	kreativkoepfe
97	10:47:23	93.91	-0.886093	450.0	kreativkoepfe
98	10:47:26	94.91	-0.505562	500.0	kreativkoepfe
99	10:47:29	95.37	-0.920069	500.0	kreativkoepfe
100	10:47:32	94.81	1.070335	500.0	kreativkoepfe
101	10:47:35	96.52	0.120475	500.0	kreativkoepfe
102	10:47:38	96.69	0.210084	500.0	kreativkoepfe
103	10:47:41	97.69	0.281772	500.0	kreativkoepfe
104	10:48:43	93.40	-0.793541	500.0	kreativkoepfe
105	10:48:46	93.30	0.254889	500.0	kreativkoepfe
226	11:13:41	99.11	1.545265	400.0	die_ahnungslosen
227	11:13:44	99.57	-1.250550	400.0	die_ahnungslosen
228	11:13:47	99.56	-1.438729	400.0	die_ahnungslosen
229	11:13:50	99.13	-0.272487	400.0	die_ahnungslosen
230	11:13:53	99.01	0.021575	400.0	die_ahnungslosen
231	11:13:56	99.45	-0.336413	400.0	die_ahnungslosen
232	11:13:59	99.38	0.756729	400.0	die_ahnungslosen
233	11:14:02	99.01	0.865404	400.0	die_ahnungslosen
234	11:14:05	98.94	1.504669	400.0	die_ahnungslosen
235	11:14:08	99.38	-1.237776	400.0	die_ahnungslosen
236	11:14:11	99.09	-1.301702	400.0	die_ahnungslosen

0.6 Code

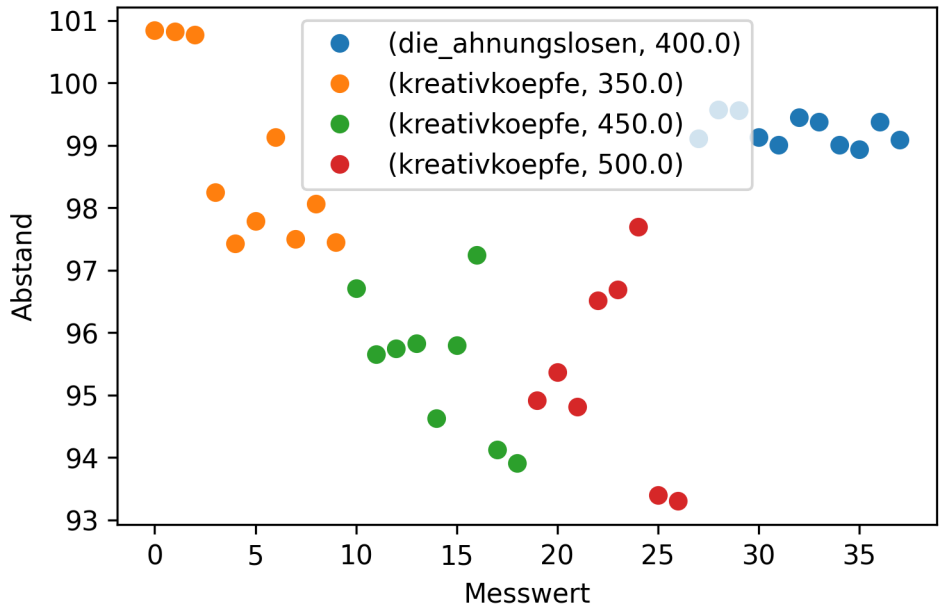
```
# z-Werte größer gleich abs(3) finden
z_values_ge3_sum = hooke.groupby(by = ['Team', 'Gewicht'])['Abstand'].apply(lambda x: scipy.s
print("Anzahl der studentisierten z-Werte mit Betrag  3:", z_values_ge3_sum)

# z-Werte größer gleich abs(2.5) finden
z_values_ge25_sum = hooke.groupby(by = ['Team', 'Gewicht'])['Abstand'].apply(lambda x: scipy
print("Anzahl der studentisierten z-Werte mit Betrag  2.5:", z_values_ge25_sum)
```

```
# Die Zeilen mit z-Werten größer abs(2.5) ausgeben
bool_index = hooke.groupby(by = ['Team', 'Gewicht'])['Abstand'].apply(lambda x: scipy.stats.z
z_values_ge_25 = hooke.iloc[bool_index , :]
print(z_values_ge_25, "\n")

# Kombinationen aus Gewicht & Team bestimmen

teams = z_values_ge_25['Team'].unique()

## teams durchlaufen und jeweils die Gewichte speichern
team_gewichte = [] # leere liste
for i in range(len(teams)):
  print(teams[i])
  print(z_values_ge_25.loc[z_values_ge_25['Team'] == teams[i], 'Gewicht'], "\n")
  team_gewichte.append(z_values_ge_25.loc[z_values_ge_25['Team'] == teams[i], 'Gewicht'].valu

print(team_gewichte, "\n")

# Messreihen auswählen
messreihen = pd.DataFrame()
for i in range(len(teams)):
  for j in range(len(team_gewichte[i])):

    print(teams[i], team_gewichte[i][j])

    messreihen = pd.concat([messreihen, hooke.loc[ (hooke['Team'] == teams[i]) & (hooke['Gewi

# studentisierte z-Werte der Messreihen bilden
messreihen_z_scores = messreihen.groupby(by = ['Team', 'Gewicht'])['Abstand'].apply(lambda x

# gemeinsame Ausgabe der Daten
messreihen.insert(loc = 2, column = 'z-Werte Abstand', value = messreihen_z_scores.values)
print(messreihen)
```

Die Werte können mit der Pandas-Methode `pd.plot()` mit wenig Aufwand dargestellt werden. Die Methode ist jedoch nicht so flexibel, wie das Paket matplotlib. So ist das Punktdiagramm (`kind = 'scatter'`) nur für DataFrames, nicht aber für groupby-Objekte verfügbar. Dies wird durch das Setzen eines Markers und die Einstellung der Liniendicke auf 0 kompensiert.

```
messreihen.reset_index(drop = True).groupby(by = ['Team', 'Gewicht'])['Abstand'].plot(marker
```

```
plt.xlabel('Messwert')
plt.ylabel('Abstand')
plt.legend()

plt.show()
```



Die Werte, die betragsmäßig studentisierte z-Werte $\geq 2{,}5$ aufweisen, könnten als Ausreißer entfernt werden. In diesem Fall wird darauf verzichtet.

Umwandlung der Rechengrößen

Im nächsten Schritt wird die Abstandsmessung auf den Nullpunkt normiert, um die Federausdehnung abzubilden. Ebenso wird das Gewicht in g in die wirkende Kraft in N umgerechnet.

Tipp 1: Musterlösung

Abstandsmessung auf Meter und auf den Nullpunkt normieren
Abstandsmessung auf den Nullpunkt normieren. Die Spalte Abstand wird in Abständsänderung umbenannt.

```
nullpunkte = hooke.loc[hooke['Gewicht'] == 0, : ].groupby(by = 'Team')['Abstand'].mean()

print(nullpunkte)

teams = nullpunkte.index

for i in range(len(teams)):

  hooke.loc[hooke['Team'] == teams[i] , 'Abstand'] = hooke.loc[hooke['Team'] == teams[i] ,

hooke.rename(columns = {'Abstand': 'Abstandsänderung'}, inplace = True)
hooke['Abstandsänderung'] = hooke['Abstandsänderung'].div(100)
```

```
Team
die_ahnungslosen    109.759000
fabi                109.812000
kreativkoepfe       109.676667
Name: Abstand, dtype: float64
```

Gewicht in wirkende Kraft umrechnen
Gewicht in g in die wirkende Kraft in N umrechnen. Die Spalte wird in den Datensatz eingefügt. umbenannt.

```
hooke['Kraft'] = hooke['Gewicht'].div(1000).mul(9.81)
```

Das Ergebnis könnte so aussehen. Die Spalte Abstand wurde in Abständsänderung umbenannt.

```
hooke.groupby(by = ['Team', 'Kraft'])['Abstandsänderung'].describe()
```

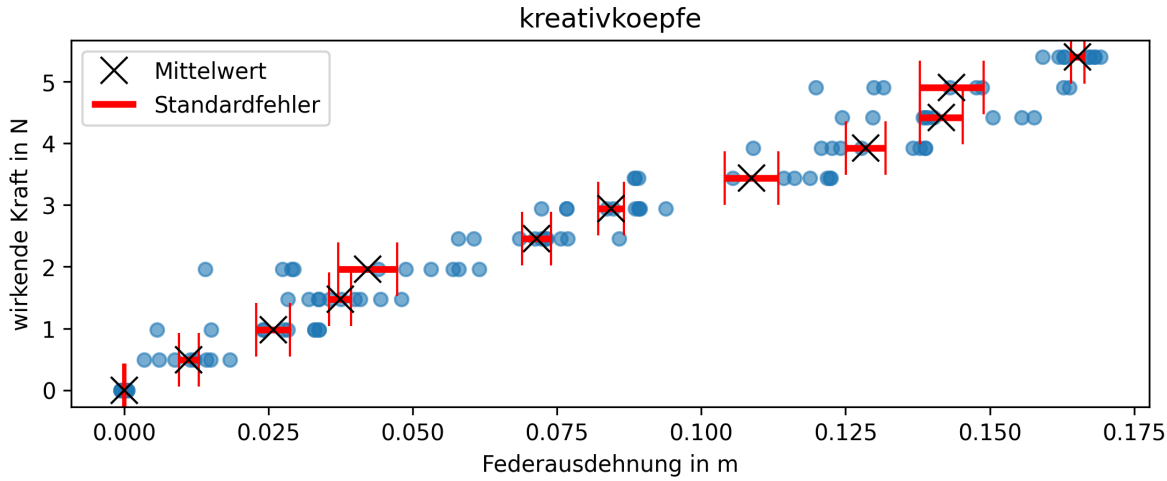
Team	Kraft	count
die_ahnungslosen	0.00000	10.0
	0.49050	8.0
	0.98100	12.0
	1.47150	19.0
	1.96200	18.0
	2.45250	18.0
	2.94300	11.0
	3.43350	14.0

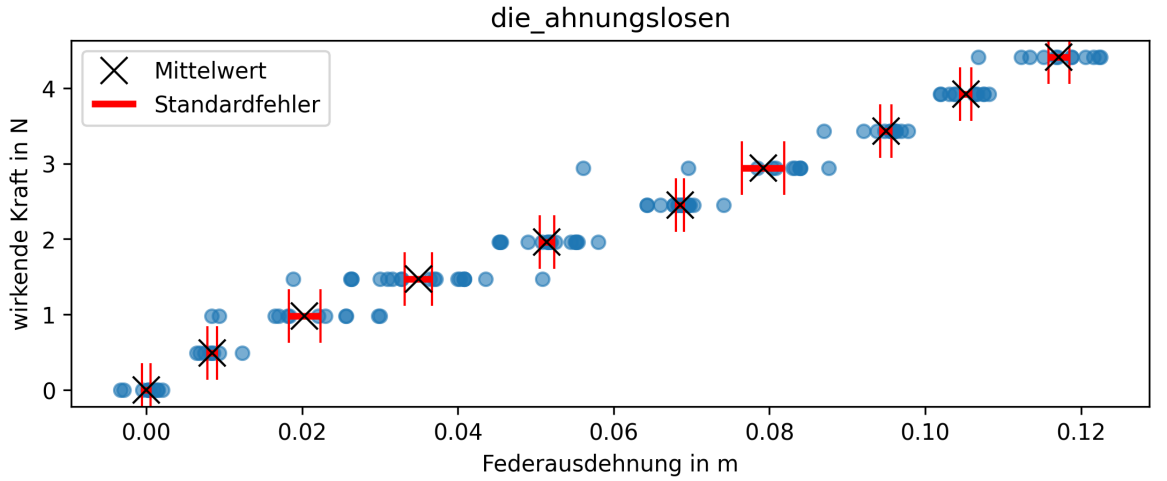
Team	Kraft	count
fabi	3.92400	11.0
	4.41450	12.0
	0.00000	20.0
	0.49050	22.0
	0.98100	10.0
	0.99081	9.0
	1.47150	11.0
	1.49112	10.0
	1.96200	9.0
	1.97181	10.0
	2.45250	10.0
	2.48193	11.0
	2.94300	10.0
	2.96262	12.0
	3.43350	10.0
	3.46293	11.0
	3.92400	10.0
	3.95343	14.0
	4.41450	10.0
	4.46355	11.0
	4.90500	10.0
	5.39550	10.0
	5.88600	10.0
	6.37650	10.0
	6.86700	10.0
	7.35750	9.0
	7.84800	11.0
	8.33850	8.0
kreativkoepfe	0.00000	9.0
	0.49050	8.0
	0.98100	10.0
	1.47150	10.0
	1.96200	10.0
	2.45250	10.0
	2.94300	10.0
	3.43350	10.0
	3.92400	9.0
	4.41450	9.0
	4.90500	8.0

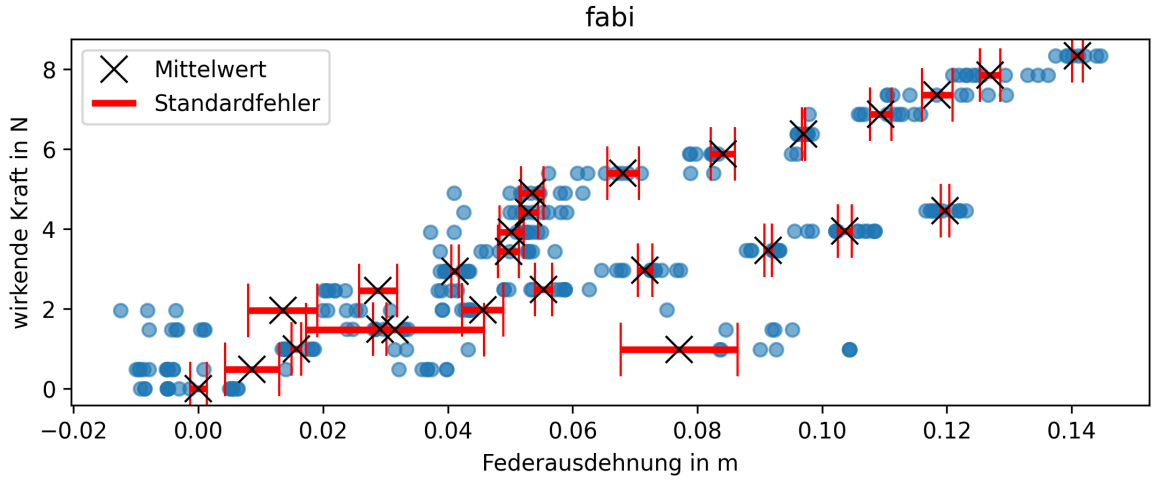
Grafische Darstellung

Da es nur vier Teams gibt, können die Messreihen grafisch dargestellt werden. Eine mögliche Darstellung können Sie dem ersten Reiter, die Zwischenschritte und Schlussfolgerungen den folgenden Reitern entnehmen.

0.7 Grafische Darstellung







0.8 Mittelwerte und Standardfehler berechnen

Die Ausgabe ist aus Platzgründen auf die ersten Zeilen beschränkt.

```
Team              Kraft
die_ahnungslosen  0.0000   -1.420739e-16
                  0.4905    8.465000e-03
                  0.9810    2.033167e-02
                  1.4715    3.493211e-02
                  1.9620    5.142889e-02
Name: Federausdehnung, dtype: float64
Team              Kraft
die_ahnungslosen  0.0000    0.000571
                  0.4905    0.000640
                  0.9810    0.002034
                  1.4715    0.001742
                  1.9620    0.000926
Name: Standardfehler, dtype: float64
```

0.9 Code

```
# Mittelwerte der Teams nach Kraft
distance_means_by_team_and_force = hooke.groupby(by = [hooke['Team'], hooke['Kraft']])['Absta
distance_means_by_team_and_force.name = 'Federausdehnung'

print(distance_means_by_team_and_force.head())

# Standardfehler der Teams nach Kraft
distance_stderrors_by_team_and_force = hooke.groupby(by = [hooke['Team'], hooke['Kraft']])['A
distance_stderrors_by_team_and_force.name = 'Standardfehler'

print(distance_stderrors_by_team_and_force.head())

# grafische Darstellung
anzahl_teams = hooke['Team'].unique().size

plt.figure(figsize = (7.5, 12))
for i in range(anzahl_teams):

  plt.subplot(4, 1, i + 1) # plt.subplot zählt ab 1
```

```
  # Punktdiagramm
  plotting_data = hooke.loc[hooke['Team'] == hooke['Team'].unique()[i], :]
  plt.scatter(x = plotting_data['Abstandsänderung'], y = plotting_data['Kraft'], alpha = 0.6)

  plt.title(label = hooke['Team'].unique()[i])
  plt.xlabel("Federausdehnung in m")
  plt.ylabel("wirkende Kraft in N")

  # # Fehlerbalken
  distance_means_by_force = plotting_data.groupby(by = plotting_data['Kraft'])['Abstandsänder
  distance_stderrors_by_force = plotting_data.groupby(by = plotting_data['Kraft'])['Abstandsä

  errorbar_container = plt.errorbar(x = distance_means_by_force, y = distance_means_by_force
  linestyle = 'none', marker = 'x', color = 'black', markersize = 12, elinewidth = 3, ecolor

  # siehe: https://matplotlib.org/stable/api/container_api.html#matplotlib.container.Errorbar
  plt.legend([errorbar_container.lines[0], errorbar_container.lines[2][0]],
             ['Mittelwert', 'Standardfehler'],
             loc = 'upper left')

plt.tight_layout()
plt.show()
```

0.10 Auswertung

- Die Messreihen des Teams die_ahnungslosen entsprechen dem erwarteten linearen Trend.
- Bei Team fabi scheint für das erste angehängte Gewicht (50 Gramm) ein Fehler bei der Datenerhebung vorzuliegen. Vermutlich wurde hier mit 0 Gramm gemessen.
- Die Messreihen des Teams kreativköpfe entsprechen weitgehend dem erwarteten linearen Trend.
- Die Messreihen des Teams ma scheinen wenigstens für die ersten vier angehängten Gewichten durch grobe Messfehler geprägt zu sein.

Die Messreihe des Teams ma wird wegen grober Messfehler aus dem Datensatz entfernt. Aus der Messreihe des Teams fabi wird die Messung für das Gewicht 50 Gramm entfernt.

```
hooke.drop(index = hooke.loc[hooke['Team'] == 'ma', :].index, inplace = True)
hooke.drop(index = hooke.loc[(hooke['Team'] == 'fabi') & (hooke['Gewicht'] == 50), :].index,
```

Tipp 2: Vorgehen bei vielen Datensätzen

Bei einer großen Anzahl an Datensätzen kann auch die grafische Kontrolle an Grenzen stoßen. In diesem Fall empfiehlt es sich, die visuelle und kennzahlenbasierende Methoden zusammen zu nutzen, um Muster zu identifizieren und für eine große Zahl von Messungen zu überprüfen. Beispielsweise könnten nach einer visuellen Inspektion von Messreihen mit Extremwerten bzw. Ausreißern alle Messreihen daraufhin überprüft werden, ob mit zunehmenden Gewicht stets auch die mittlere Federausdehnung größer als für leichtere Gewichte ist. Abweichende Messreihen könnten dann grafisch kontrolliert werden.

0.11 Federkonstanten bestimmen

Im nächsten Schritt können die Federkonstanten mittels linearer Regression bestimmt werden.

2. **Welche Werte können für die Federkonstanten ermittelt werden?**

3. **Wurden die Messungen mit der gleichen Feder durchgeführt, wenn als Vertrauenswahrscheinlichkeit 90 % bzw. 95 % angenommen werden soll?**

0.12 $\alpha = 0.10$

```
kreativkoepfe Konfidenzniveau: 0.9
y = 0.3180 + 29.7643 * x
r = 0.9773 R2 = 0.9552 p = 0.0000
Standardfehler des Anstiegs: 0.6148
28.744    29.764   30.784

die_ahnungslosen Konfidenzniveau: 0.9
y = 0.1798 + 34.9183 * x
r = 0.9886 R2 = 0.9773 p = 0.0000
Standardfehler des Anstiegs: 0.4651
34.148    34.918   35.689

fabi Konfidenzniveau: 0.9
y = 0.7549 + 44.8858 * x
r = 0.7990 R2 = 0.6384 p = 0.0000
Standardfehler des Anstiegs: 2.0790
41.454    44.886   48.317
```

0.13 $\alpha = 0.05$

```
kreativkoepfe Konfidenzniveau: 0.95
y = 0.3180 + 29.7643 * x
r = 0.9773 R2 = 0.9552 p = 0.0000
Standardfehler des Anstiegs: 0.6148
28.546    29.764   30.983

die_ahnungslosen Konfidenzniveau: 0.95
y = 0.1798 + 34.9183 * x
r = 0.9886 R2 = 0.9773 p = 0.0000
Standardfehler des Anstiegs: 0.4651
33.998    34.918   35.838

fabi Konfidenzniveau: 0.95
y = 0.7549 + 44.8858 * x
r = 0.7990 R2 = 0.6384 p = 0.0000
Standardfehler des Anstiegs: 2.0790
40.792    44.886   48.979
```

0.14 Code

```
anzahl_teams = hooke['Team'].unique().size
alpha = 0.10

for i in range(anzahl_teams):

  reg_data = hooke.loc[hooke['Team'] == hooke['Team'].unique()[i], :]
  x = reg_data['Abstandsänderung']
  y = reg_data['Kraft']
  n = len(x)

  print("\n", hooke['Team'].unique()[i], " Konfidenzniveau: ", 1 - alpha, sep = '')

  slope, intercept, rvalue, pvalue, slope_stderr = scipy.stats.linregress(x, y)
  print(f"y = {intercept:.4f} + {slope:.4f} * x\n",
        f"r = {rvalue:.4f} R2 = {rvalue ** 2:.4f} p = {pvalue:.4f}\n",
        f"Standardfehler des Anstiegs: {slope_stderr:.4f}", sep = '')

  print(f"{slope - scipy.stats.t.ppf(q = 1 - alpha / 2, df = n - 2) * slope_stderr:.3f}    {s
```

```
alpha = 0.05

for i in range(anzahl_teams):

  reg_data = hooke.loc[hooke['Team'] == hooke['Team'].unique()[i], :]
  x = reg_data['Abstandsänderung']
  y = reg_data['Kraft']
  n = len(x)

  print("\n", hooke['Team'].unique()[i], " Konfidenzniveau: ", 1 - alpha, sep = '')

  slope, intercept, rvalue, pvalue, slope_stderr = scipy.stats.linregress(x, y)
  print(f"y = {intercept:.4f} + {slope:.4f} * x\n",
        f"r = {rvalue:.4f} R2 = {rvalue ** 2:.4f} p = {pvalue:.4f}\n",
        f"Standardfehler des Anstiegs: {slope_stderr:.4f}", sep = '')

  print(f"{slope - scipy.stats.t.ppf(q = 1 - alpha / 2, df = n - 2) * slope_stderr:.3f}    {s
```

0.15 Auswertung

Die Punktschätzung der Federkonstante von Team fabi 34.105 liegt im 95-%-Konfidenzintervall der Messung von Team die_ahnungslosen 33.998 34.918 35.838. Die Punktschätzung der Federkonstante von Team fabi 34.105 liegt **aber nicht im 90-%-Konfidenzintervall** der Messung von Team die_ahnungslosen 34.148 34.918 35.689.

Unabhängig vom gewählten Vertrauensniveau liegt die Punktschätzung der Federkonstante von Team kreativkoepfe 29.764 nicht in den Konfidenzintervallen der beiden übrigen Teams.

> **Warning 1: Ergebnisse**
>
> Abhängig vom gewählten Vorgehen sind andere Ergebnisse möglich, beispielsweise durch das Entfernen von als Ausreißern eingestuften Einzelwerten oder einer anderen Behandlung der Messreihe vom Team fabi für das angehängte Gewicht 50 Gramm.