Serie 2

Aufgabe 2.1

In einer Klasse wurden in einer Statistik-Prüfung folgende Noten geschrieben:

4.2, 2.3, 5.6, 4.5, 4.8, 3.9, 5.9, 2.4, 5.9, 6, 4, 3.7, 5, 5.2, 4.5, 3.6, 5, 6, 2.8, 3.3, 5.5, 4.2, 4.9, 5.1

a) Ändern Sie drei Noten im Datensatz so ab, dass der Median gleich bleibt, aber der Mittelwert sich stark ändert.

b) Erstellen Sie zu den beiden Datensätzen je ein Histogramm und einen Boxplot. Verwenden Sie **`plt.subplot(...)`** aus Aufgabe 1.

Aufgabe 2.2

21 Labors bestimmten den Kupfergehalt von 9 verschiedenen Klärschlammproben. Die Daten stehen in der auf Ilias abgelegten Datei `klaerschlamm.dat` zur Verfügung. Die erste Spalte bezeichnet das Labor, die restlichen 9 Spalten sind die verschiedenen Klärschlammproben. Die Daten (in mg/kg) können mit dem Befehl

```
schlamm = pd.read_table(r"*klaerschlamm.dat", sep=" ", index_col=0)
```

eingelesen werden (für * wieder der Dateipfad). Die erste Spalte `Labor` wollen wir noch entfernen, da sie uns nicht interessiert.

```
schlamm = schlamm.drop("Labor",1)
schlamm.head()
```

a) Erstellen Sie für jede Probe einen Boxplot, und berechnen Sie jeweils das arithmetische Mittel und den Median. Bei welchen Proben gibt es Ausreisser, und wo unterscheiden sich arithmetisches Mittel und Median wesentlich? Bei welchen der 9 Proben ist es plausibel, dass die wahre Konzentration unter 400 mg/kg liegt?

`Python`-Hinweise:

```
schlamm.describe()
schlamm.plot(kind="box")
```

b) Erstellen Sie für jedes Labor einen Boxplot der Messfehler. Unter dem Messfehler eines Labors bei einer Probe verstehen wir den gemessenen Wert minus den Median über alle Labors. Welche der 21 Labors haben systematische Fehler in ihrem Analyseverfahren? Welche haben grosse Zufallsfehler, und bei welchen Labors ist die Qualität der Analysen besonders gut?

Python-Hinweise: Wir ziehen zunächst von jeder Spalte den Median ab

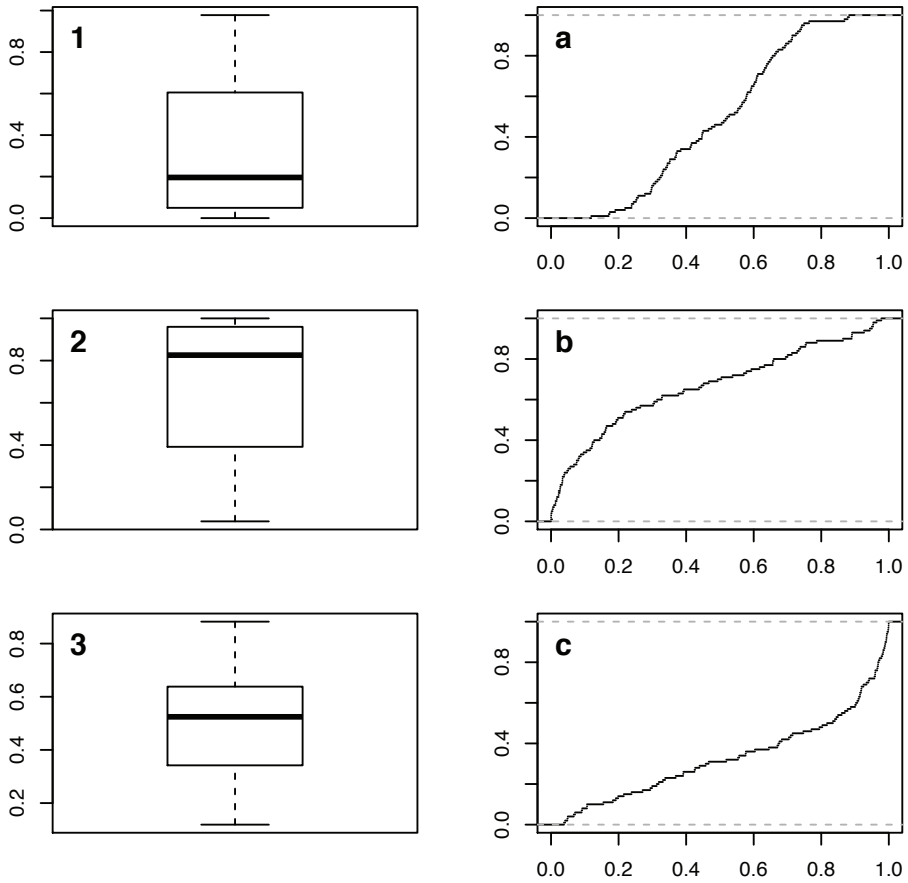
```
schlamm_centered = schlamm - schlamm.median()
```

und zeichnen den Boxplot mit vertauschten Zeilen und Spalten. Dies geschieht mit dem Attribut **.T**

```
schlamm_centered.T.plot(kind="box")
```

Aufgabe 2.3

Für drei Stichproben vom Umfang $n = 100$ wurden je ein Boxplot und die empirische Verteilungsfunktion gezeichnet. Ordnen Sie die drei Boxplots den entsprechenden empirischen Verteilungsfunktionen zu:



Aufgabe 2.4

Edwin Hubble untersuchte seit 1920 am Mount Wilson Observatory die Eigenschaften von Galaxien ausserhalb der Milchstrasse. Mit Überraschung bemerkte er einen Zusammenhang zwischen der Distanz einer Galaxie zur Erde und dessen Geschwindigkeit, sich von der Erde fortzubewegen (Fluchtgeschwindigkeit). Hubbles ursprüngliche Daten von 24 galaktischen Nebeln (E. Hubble, "'Proceedings of the National Academy of Science 15 (1929): 168-73"') sind in Tabelle 1 gezeigt. Die Fluchtgeschwindigkeit ist in Kilometer pro Sekunde angegeben und konnte aufgrund der Rotverschiebung im Lichtspektrum der Galaxien mit grosser Genauigkeit bestimmt werden. Die Distanz einer Galaxie zur Erde wird in Megaparsec (Mpc) gemessen: ein Megaparsec entspricht etwa 3.09×10^{10} m. Die Distanzen werden durch Vergleich der mittleren Luminosität von Galaxien mit der Luminosität von bestimmten bekannten Sternen bestimmt, wobei diese Methode relativ ungenau ist.

Nebel	Geschwindigkeit (km/s)	Distanz (Mpc)
S. Mag.	170	0.032
L. Mag. 2	290	0.034
NGC 6822	-130	0.214
NGC 598	-70	0.263
NGC 221	-185	0.275
NGC 224	-220	0.275
NGC 5457	200	0.450
NGC 4736	290	0.500
NGC 5194	270	0.500
NGC 4449	200	0.630
NGC 4214	300	0.800
NGC 3031	-30	0.900
NGC 3627	650	0.900
NGC 4626	150	0.900
NGC 5236	500	0.900
NGC 1068	920	1.000
NGC 5055	450	1.100
NGC 7331	500	1.100
NGC 4258	500	1.400
NGC 4151	960	1.700
NGC 4382	500	2.000
NGC 4472	850	2.000
NGC 4486	800	2.000
NGC 4649	1090	2.000

Table 1: Zusammenhang zwischen Distanz und Fluchtgeschwindigkeit von Galaxien.

a) Erstellen Sie von den Daten in Tabelle 1 ein Streudiagramm, in dem Sie die Distanz versus Fluchtgeschwindigkeit aufzeichnen.

   Lesen Sie dazu die Datei ein:

```
hubble = pd.read_table("*/hubble.txt", sep=" ")
```

b) Bestimmen Sie mit dem Befehl **`np.polyfit(...)`** (siehe Skript) die Koeffizienten β_0 und β_1 für die Regressionsgerade

$$y = \beta_0 + \beta_1 x$$

   wobei y die Distanz und x die Fluchtgeschwindigkeit bezeichnet.

c) Bestimmen Sie noch den Korrelationskoeffizienten und interpretieren Sie diesen.

Aufgabe 2.5

Wir betrachten eine Studie, die 1979 in den Vereinigten Staaten durchgeführt wurde (National Longitudinal Study of Youth, NLSY79): von 2584 Amerikanern im Jahr 1981 wurde der Intelligenzquotient (gemäss AFQT - armed forces qualifying test score) gemessen; 2006 wurden dieselben Personen nach ihrem jährlichen Einkommen im Jahr 2005 und der Anzahl Jahre Schulbildung befragt. Uns interessiert hier natürlich, ob ein hoher IQ oder eine lange Schulbildung zu einem höheren Einkommen führen. In der auf Ilias abgelegten Datei `income.dat` finden Sie den Datensatz mit dem Einkommen, der Anzahl Jahre abgeschlossener Schulbildung und den ermittelten Intelligenzquotienten von 2584 Amerikanern.

a) Lesen Sie den Datensatz `income.dat` ein

```
income = pd.read_table(r"*income.dat", sep=" ")
```

und generieren Sie Streudiagramme, in welchen das Einkommen versus Anzahl Jahre Schulbildung und Einkommen versus Intelligenzquotient aufgetragen sind.

b) Bestimmen Sie die Parameter a und b des linearen Modells $y = a + bx$, wobei y das Einkommen bezeichnet und x die Anzahl Jahre Schulbildung. Zeichnen Sie die Regressionsgerade wie im Skript

```
x = np.linspace(... , ...)

plt.plot(x, a+b*x, c="orange")
```

Wie interpretieren Sie die Parameter a und b?

c) Berechnen Sie die Korrelation zwischen Einkommen und Anzahl Jahre Schulbildung. Wie angebracht ist ein Regressionsmodell für diesen Datensatz?

Aufgabe 2.6

In dieser Aufgabe betrachten wir 4 Datensätze, die von Anscombe konstruiert wurden.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5])
y1 = np.array([8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68])
y2 = np.array([9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74])
y3 = np.array([7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73])
x4 = np.array([8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8])
y4 = np.array([6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89])
```

In jedem der Datensätze gibt es eine Zielvariable y und eine erklärende Variable x.

a) Stellen Sie jeden der 4 Datensätze als Streudiagramm dar, zeichnen Sie die Regressionsgerade ein und kommentieren Sie die Ergebnisse. Verwenden Sie wieder **`plt.subplot(...)`** aus Aufgabe 1 und **`plt.scatter(...)`**.

b) Vergleichen Sie die Schätzungen von β_0 und β_1, wobei $y = \beta_0 + \beta_1 x$.

Die Schätzungen für die Koeffizienten β_0 und β_1 des linearen Regressionsmodells kann man mit **`np.polyfit(...)`** berechnen und numerisch auswerten.

c) Berechnen Sie die Korrelationskoeffizienten mit **`np.corrcoef(...)`**.

Aufgabe 2.7

Der Datensatz der OECD enthält Messgrössen, die das Wohlergehen von Kindern in den Mitgliedsstaaten ermitteln sollen.

a) Speichern Sie die Datei `child.csv`, und lesen Sie den Datensatz `child.csv` mit der Funktion

```
from pandas import Series,DataFrame
import pandas as pd

data = pd.read_csv("../child.csv", sep=",", index_col=0)
```

Achtung: Für *../ muss* der gesamte Pfad angegeben werden, wo sich ihr File `child.csv` befindet.

b) Bestimmen Sie den Mittelwert und Median der einzelnen Variablen mit dem Python-Attribut **`.describe()`**. Gibt es **`NaN`**'s im Datensatz?

c) Entfernen Sie alle Länder (Zeilen), die **`NaN`** enthalten.

d) Entfernen Sie alle Messgrössen (Spalten), die mehr als zwei **`NaN`** enthalten.

e) Ersetzen Sie nun alle **`NaN`**'s mit Werten, und zwar mit einer Methode Ihrer Wahl. Erklären Sie die von Ihnen benützte Methode.

f) Warum fehlen Datenwerte? Handelt es sich wohl um MCAR, MAR oder MNAR?

Kurzlösungen vereinzelter Aufgaben

A 2.5:

b) $a = -40200$ und $b = 6451$ c) $r = 0.346$

A 2.6:

b) $\widehat{\beta}_0 \approx 3.00$ und $\widehat{\beta}_1 \approx 0.500$

c) Alle ungefähr 0.816 (auf dritte Stelle nach Komma)

Musterlösungen zu Serie 2

Lösung 2.1

a) (zu R) Der ursprüngliche Datensatz hat für den Median und Mittelwert folgende Werte:

```
from pandas import Series
n1 = Series([4.2, 2.3, 5.6, 4.5, 4.8, 3.9, 5.9, 2.4, 5.9,
             6, 4, 3.7, 5, 5.2, 4.5, 3.6, 5, 6, 2.8, 3.3,
             5.5, 4.2, 4.9, 5.1])
n1.median()
n1.mean()

## 4.65
## 4.5125
```

Zuerst ordnen wir die Datenwerte der Grösse nach: (zu R)

```
n2 = n1.sort_values()
n2

## 1     2.3
## 7     2.4
## 18    2.8
## 19    3.3
## 15    3.6
## 11    3.7
## 5     3.9
## 10    4.0
## 21    4.2
## 0     4.2
## 14    4.5
## 3     4.5
## 4     4.8
## 22    4.9
## 12    5.0
## 16    5.0
## 23    5.1
## 13    5.2
## 20    5.5
## 2     5.6
## 8     5.9
## 6     5.9
## 17    6.0
## 9     6.0
```

```
## dtype: float64
```

Da die Anzahl Noten gerade ist, (zu **R**)

```
n2 = n1.sort_values()
n2.size

## 24
```

wird der Median aus dem Mittelwert von $x_{(12)}$ und $x_{(13)}$ gebildet. Wenn wir also Noten kleiner als $x_{(12)}$ abändern, wird sich der Median nicht ändern. Dementsprechend ändern wir die Notenwert $x_{(9)}, x_{(10)}, x_{(11)}$ zu einer eins. Dies lässt den Median unverändert, lässt den Mittelwert aber maximal schrumpfen.

Die **Series** `n2` sieht wie folgt aus:

```
n2.head()

## 1     2.3
## 7     2.4
## 18    2.8
## 19    3.3
## 15    3.6
## dtype: float64
```

Der Index wurde also mit sortiert. Damit wir auf die 9., 10. und 11. Werte von `n2` zugreifen können, müssen wird den Index von `n2` ändern.

```
n2.index = np.arange(1, n2.size+1)
n2.head()

## 1    2.3
## 2    2.4
## 3    2.8
## 4    3.3
## 5    3.6
## dtype: float64
```

Nun können wir den 9., 10. und elften Wert auf 1 setzen:

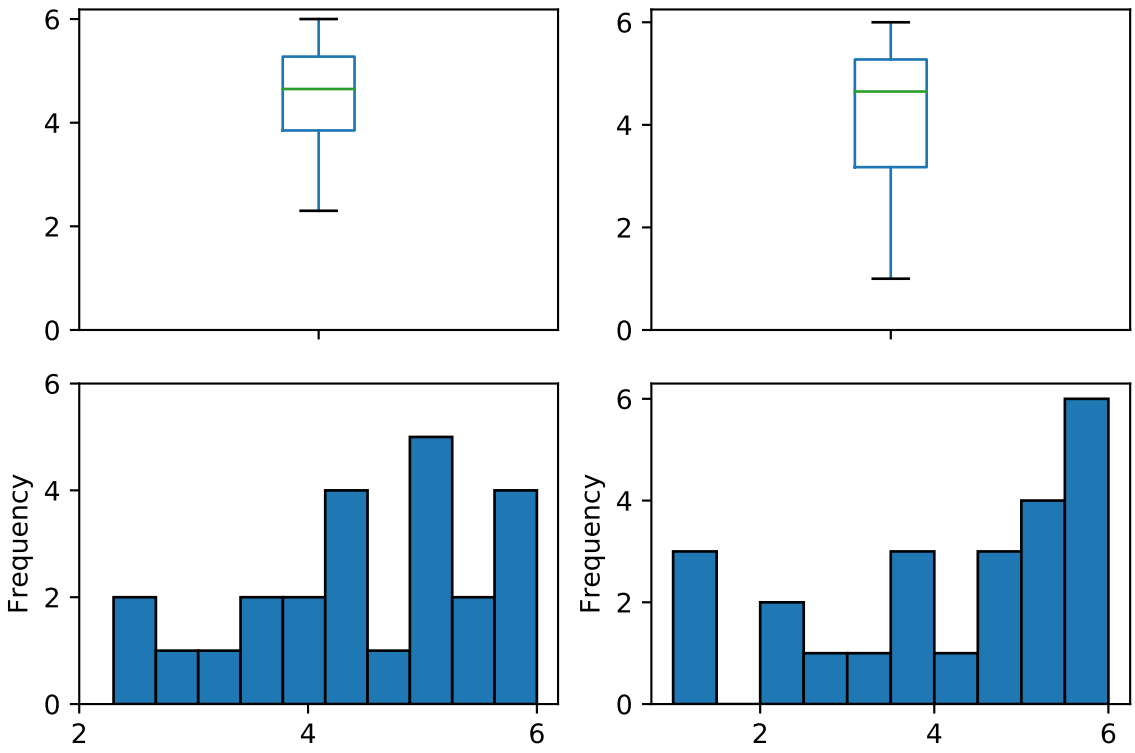
```
n2[11], n2[10], n2[9] = 1, 1, 1
n2.head(n=12)

## 1     2.3
## 2     2.4
## 3     2.8
## 4     3.3
## 5     3.6
```

```
## 6     3.7
## 7     3.9
## 8     4.0
## 9     1.0
## 10    1.0
## 11    1.0
## 12    4.5
## dtype: float64
```

Nun ist (zu R)

```
n2.median()
n2.mean()

## 4.65
## 4.1000000000000005
```

b) Der Plot (die Skizzen wurden hier noch skaliert).

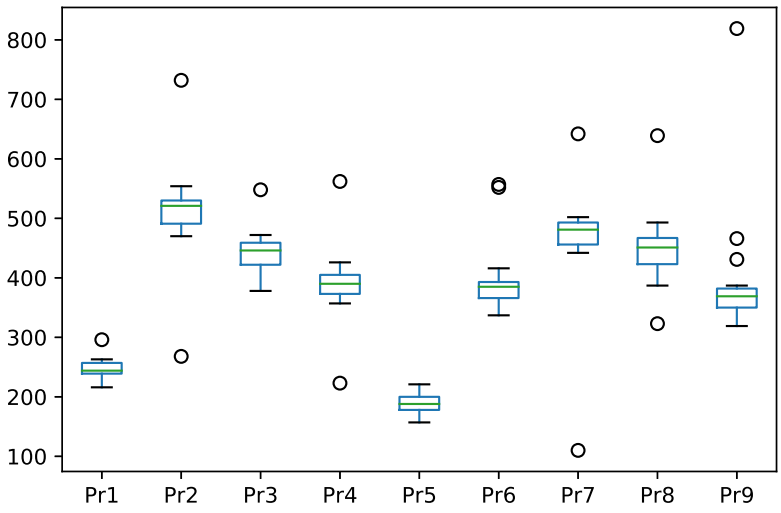



```
plt.subplot(221)
n1.plot(kind="box")
```

```
plt.subplot(222)
n2.plot(kind="box")

plt.subplot(223)
n1.plot(kind="hist",edgecolor="black")

plt.subplot(224)
n2.plot(kind="hist",edgecolor="black")

plt.show()
```

Lösung 2.2

a) Aus den Boxplots erkennen wir, dass es vor allem bei den Proben 2, 4, 6, 7, 8 und 9 Ausreisser gibt. Das arithmetische Mittel und der Median unterscheiden wesentlich bei den Proben 2, 6, 7 und 9.



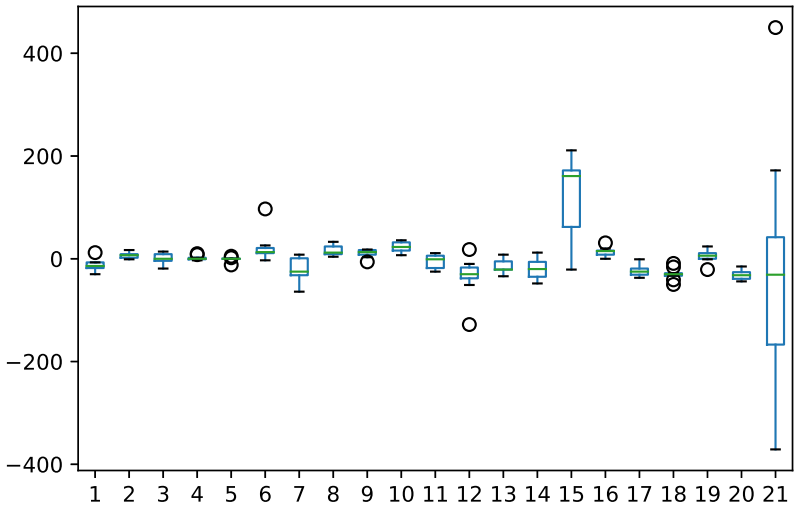
(Zu (zu R))

```
schlamm.describe()
```

Bei den Proben 1 und 5 ist es plausibel, dass die Konzentration unter 400 mg/kg liegt, während wir bei Probe 2, 3, 7 und 8 eher dazu tendieren, den Grenzwert 400 mg/kg als überschritten zu betrachten.

Die übrigen Proben, Probe 4, 6 und 9 sind eher Grenzfälle. Die Konzentrationen scheinen zwar unter 400 mg/kg zu liegen, die drei Proben weisen jedoch jeweils extreme Ausreisser über dem Grenzwert auf.

b) Als erstes stechen die Messungen der Labors 15 und 21 ins Auge. Beide haben sowohl eine grosse Standardabweichung als auch systematische Fehler. Die Labors 6 und 12 haben beide Ausreisser zu verzeichnen. Die Labors 1, 7, 12, 13, 14, 17, 18, 20 und 21 geben systematisch zu kleine Werte an, während die Labors 6, 8, 10 und 15 zu grosse Werte erhalten. Die Labors 2, 3, 4, 5 und 19 scheinen zuverlässige Untersuchungen durchzuführen. Sowohl systematische wie auch Zufallsfehler scheinen sich hier in Grenzen zu halten (zu (zu **R**))



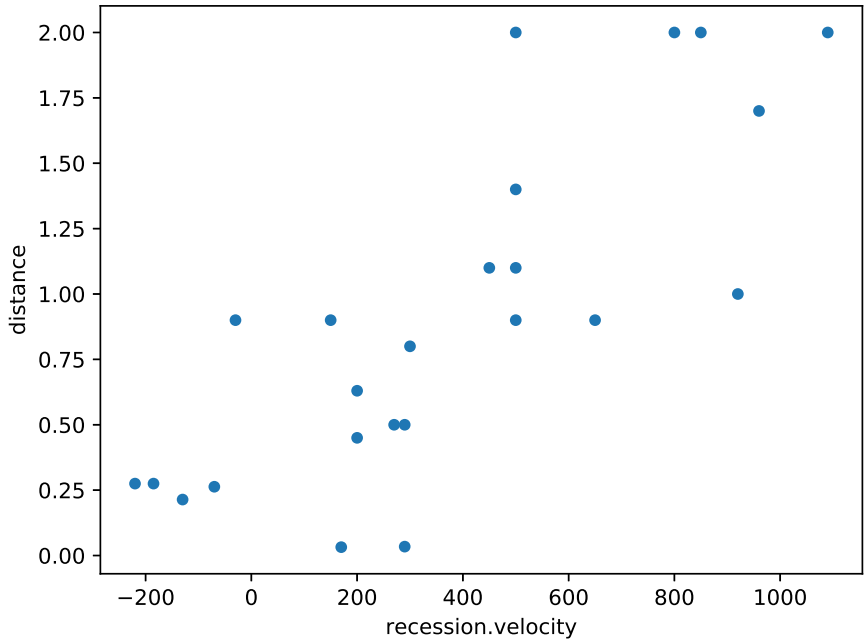
Lösung 2.3

1b, 2c, 3a

Lösung 2.4

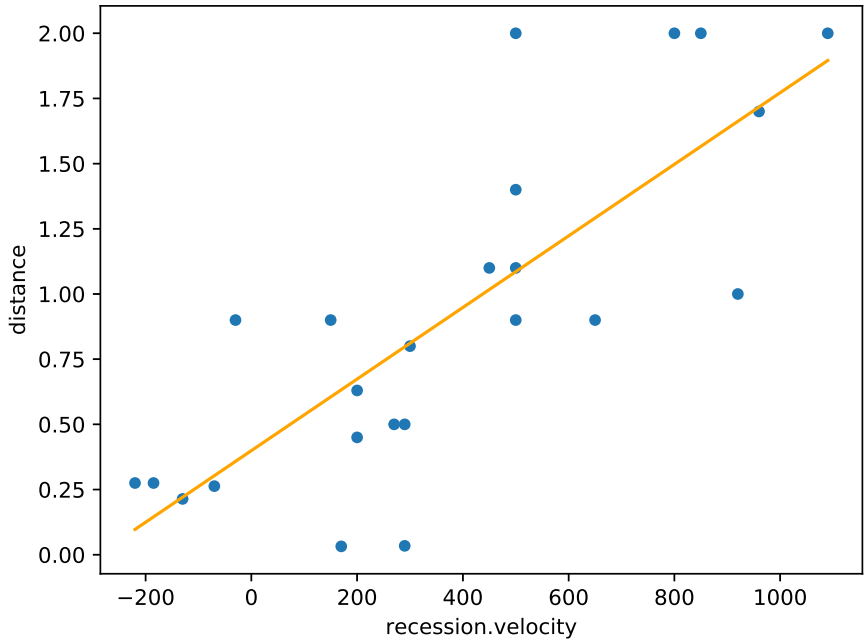
a) Streudiagramm in **Python** (zu (zu **R**))

```
from pandas import Series
import matplotlib.pyplot as plt
hubble.plot(kind="scatter",
            x= "distance",
            y="recession.velocity")
```



b) Die Koeffizienten der Regressionsgerade berechnen sich mit (siehe (zu R))

```
beta1, beta0 = np.polyfit(y= hubble["distance"], x=hubble["recession.velocity
print(beta0, beta1)

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
## 0.3990982158435929 0.0013729361049417948
```




```
hubble.plot(kind="scatter",
            x= "distance",
            y="recession.velocity")

beta1, beta0 = np.polyfit(y= hubble["distance"],
                          x=hubble["recession.velocity"],
                          deg=1)

x = np.linspace(hubble["recession.velocity"].min(),
                hubble["recession.velocity"].max())

plt.plot(x, beta0 + beta1*x, color="orange")

plt.show()
```

c) Der Regressionskoeffizient ist 0.79 und damit recht hoch (nahe bei 1), was auf einen linearen Zusammenhang von Abstand und Fluchtgeschwindigkeit der Galaxien hindeutet (siehe (zu **R**)).

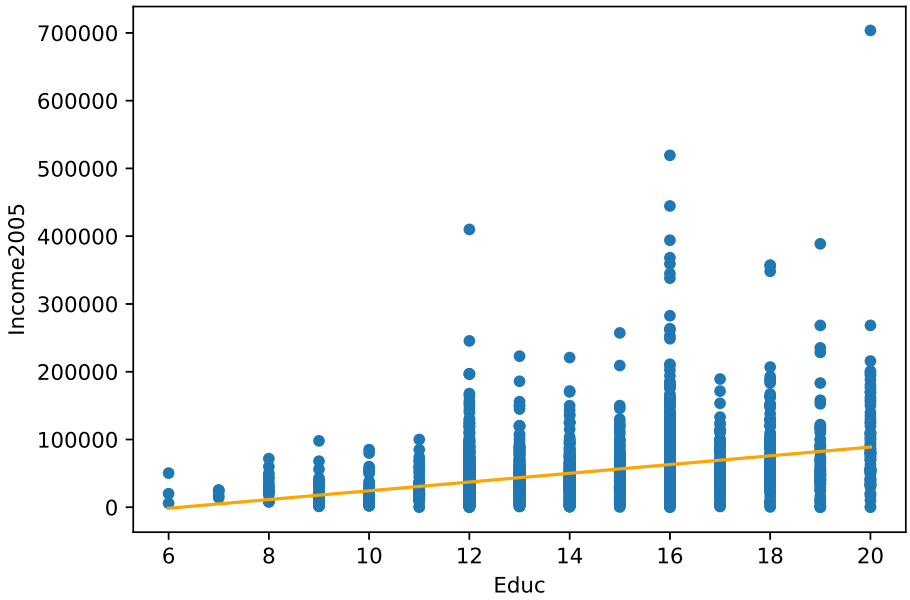
```
hubble.corr()

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
##                      distance  recession.velocity
```

```
## distance             1.000000             0.789639
## recession.velocity   0.789639             1.000000
```

Lösung 2.5

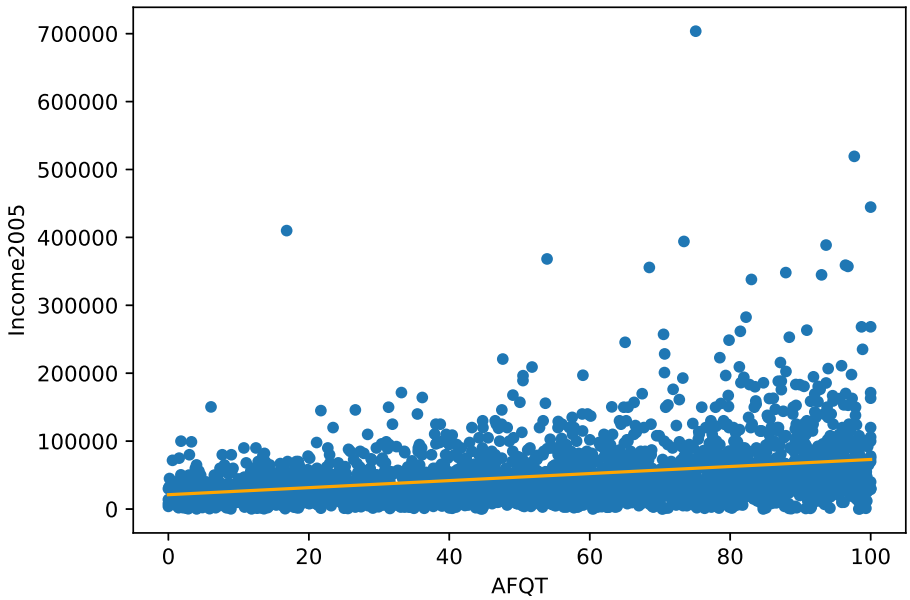
a) (zu R)

Streudiagramme




```
income.plot(kind="scatter", x="Educ", y="Income2005")

x = np.linspace(income["Educ"].min(), income["Educ"].max())

a, b = np.polyfit(income["Educ"], income["Income2005"], deg=1)

plt.plot(x,a*x+b,c="orange")

plt.show()
```

und



b) (zu **R**)

Mit **Python** ermitteln wir für a und b

```
b, a = np.polyfit(income["AFQT"], income["Income2005"], deg=1)
print(a, b)

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
## 21181.656863527787 518.6820790195897
```

```
b, a = np.polyfit(income["Educ"], income["Income2005"], deg=1)
print(a, b)

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
## -40199.57535260002 6451.4745559458015
```

Wir finden also die Werte $a = -40'200$ und $b = 6451$ für den Fall von Einkommen gegen Anzahl Jahre Schulbildung (und $a = 21'182$ und $b = 518.68$ für den betrachteten Fall Einkommen gegen Intelligenzquotient). Mit jedem zusätzlichen Jahr Schulbildung geht also eine jährliche Einkommenszunahme von 6451 USD einher.

Nun ist allerdings Vorsicht geboten: jemand ohne Schulbildung würde ein Einkommen von $-40'200$ USD haben. Dies macht natürlich keinen Sinn. Wann immer man in Bereiche extrapoliert, wo keine Datenpunkte vorhanden waren, ist Vorsicht bei der Interpretation geboten.

c) Für die *empirische Korrelation* erhalten wir dann

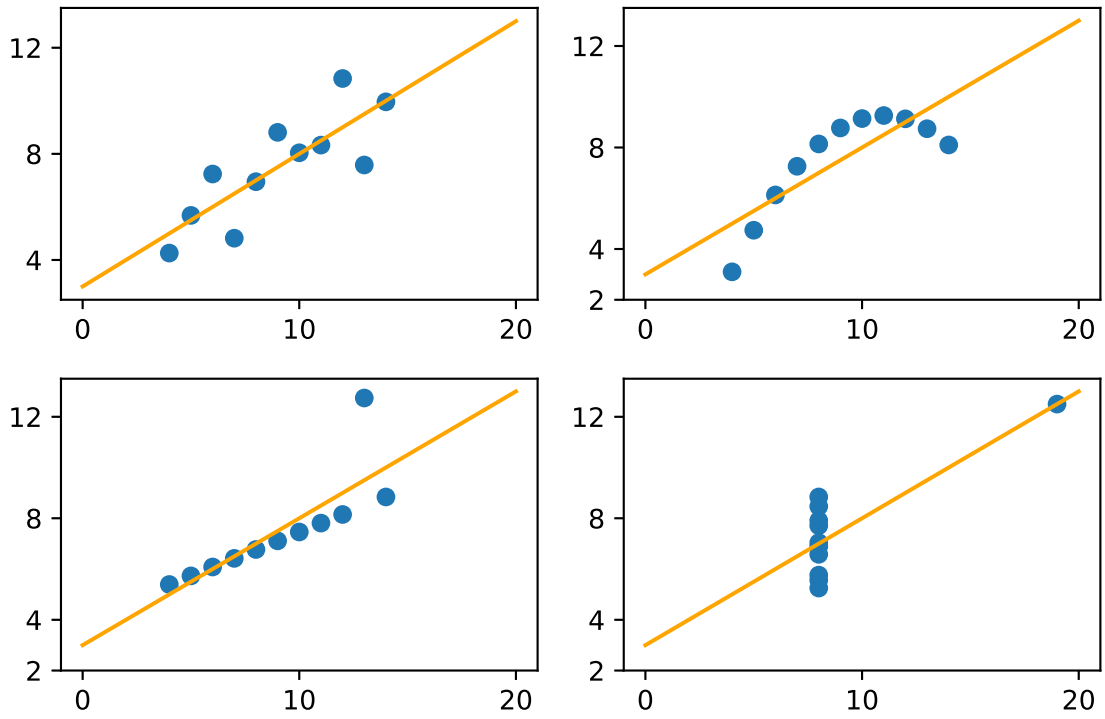
```
income.corr()

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
##                  AFQT      Educ  Income2005
## AFQT         1.000000  0.595160    0.308153
## Educ         0.595160  1.000000    0.345647
## Income2005   0.308153  0.345647    1.000000
```

Da der Korrelationskoeffizient mit 0.346 relativ klein ist, scheint ein Modell beruhend auf einem linearen Zusammenhang zwischen Einkommen und Anzahl Jahre Schulbildung nicht angebracht zu sein.

Lösung 2.6

a) (zu **R**)

Betrachtet man die vier Streudiagramme (hier noch skaliert), so sieht man, dass nur im ersten Fall eine lineare Regression korrekt ist. Im zweiten Fall ist die Beziehung zwischen X und Y nicht linear, sondern quadratisch. Im dritten Fall gibt es einen Ausreisser, welcher die geschätzten Parameter stark beeinflusst. Im vierten Fall wird die Regressionsgerade durch einen einzigen Punkt bestimmt.



b) (zu **R**)

Bei allen vier Modellen sind die Schätzungen des Achsenabschnitts β_0 und der Steigung β_1 fast identisch. Für das erste Beispiel erhalten wir

```
np.polyfit(x, y1, deg=1)

## [0.50009091 3.00009091]
```

	Modell 1	Modell 2	Modell 3	Modell 4
Achsenabschnitt ($\widehat{\beta}_0$)	3.000	3.001	3.002	3.002
Steigung ($\widehat{\beta}_1$)	0.500	0.500	0.500	0.500

Alle Streudiagramme haben praktisch dieselbe Regressionsgerade, obwohl die Streudiagramme sehr unterschiedlich aussehen.

Fazit: Es genügt **nicht**, nur $\widehat{\beta}_0$ und $\widehat{\beta}_1$ anzuschauen. In allen Modellen sind diese Schätzungen fast gleich, aber die Datensätze sehen ganz unterschiedlich aus. Eine (graphische) Überprüfung der Modellannahmen ist also unumgänglich.

c) (zu **R**)

Für das erste Beispiel erhalten wir

```
np.corrcoef(x, y1)

## [[1.         0.81642052]
##  [0.81642052 1.        ]]
```

Lösung 2.7

a) (zu **R**)

Eine Zusammenfassung lässt sich mit **Python** folgendermassen erhalten:

```
data.describe()

## -c:3: FutureWarning: read_table is deprecated, use read_csv instead.
##        Average.disposable.income  ...  Liking.school
## count                  30.000000  ...       25.00000
## mean                   18.847713  ...       27.17200
## std                     7.597219  ...       10.39926
## min                     3.839462  ...       11.70000
## 25%                    16.617877  ...       21.40000
## 50%                    21.107187  ...       25.60000
## 75%                    22.642722  ...       34.90000
## max                    34.241822  ...       57.40000
##
## [8 rows x 21 columns]
```

Wie Sie feststellen können, kommen noch die Werte **NaN** vor. Dies steht für „not a number" und steht in solchen Untersuchungen für Werte, die nicht erhoben wurden oder unbekannt sind. Für weitere Berechnungen werden diese **NaN** ignoriert.

b) (zu **R**)

Mit **df.dropna()** können alle Zeilen mit **NaN** entfernt werden:

```
data = pd.read_table("./Daten/child.csv",sep=",",index_col=0)
data.dropna()
print(data.dropna().shape)

## -c:4: FutureWarning: read_table is deprecated, use read_csv instead.
##                 Average.disposable.income  ...  Liking.school
## Austria                         22.162446  ...           38.1
## Belgium                         21.401153  ...           21.6
## Czech Republic                  10.849270  ...           11.7
## Denmark                         23.175894  ...           25.6
## Finland                         22.027651  ...           16.1
## France                          18.960382  ...           21.4
## Germany                         19.894067  ...           34.9
## Greece                          17.183647  ...           25.6
## Hungary                          9.463130  ...           27.6
## Ireland                         22.364689  ...           24.0
## Italy                           17.180761  ...           12.8
## Netherlands                     25.041012  ...           39.7
## Norway                          28.574371  ...           41.7
## Portugal                         3.839462  ...           22.8
## Spain                           16.430249  ...           23.9
## Sweden                          19.916998  ...           24.1
## United Kingdom                  22.697062  ...           35.7
## United States                   29.196531  ...           26.4
##
## [18 rows x 21 columns]
## (18, 21)
```

c) (zu **R**)

Mit **df.dropna(axis=1, thresh=28)** können alle Spalten entfernt werden, die mehr als zwei **NaN** enthalten (resp. werden die Spalten behalten, die mindestens 28 nicht-fehlende Werte enthalten). In diesem Fall braucht es in jeder Spalte folglich mindestens 28 Werte, die verschieden von **NaN** sind, ansonsten wird die Spalte gelöscht.

```
data = pd.read_table("./Daten/child.csv",sep=",",index_col=0)
data.dropna()
print(data.dropna(axis=1, thresh=28).shape)

## -c:4: FutureWarning: read_table is deprecated, use read_csv instead.
##                  Average.disposable.income  ...  Teenage.births
## Australia                        20.813221  ...            14.3
## Austria                          22.162446  ...            12.3
## Belgium                          21.401153  ...             7.8
## Canada                           25.606245  ...            13.2
## Czech Republic                   10.849270  ...            11.4
## Denmark                          23.175894  ...             6.6
## Finland                          22.027651  ...             9.7
## France                           18.960382  ...             6.7
```

```
## Germany                          19.894067  ...            9.8
## Greece                           17.183647  ...            8.7
## Hungary                           9.463130  ...           20.7
## Iceland                          22.286852  ...           16.9
## Ireland                          22.364689  ...           13.5
## Italy                            17.180761  ...            6.8
## Japan                            22.479705  ...            3.7
## Korea                            21.651918  ...            3.7
## Luxembourg                       34.241822  ...            8.6
## Mexico                            5.335074  ...           65.8
## Netherlands                      25.041012  ...            4.7
## New Zealand                      17.197108  ...           23.4
## Norway                           28.574371  ...            9.4
## Poland                            7.939399  ...           14.5
## Portugal                          3.839462  ...           18.1
## Slovak Republic                   7.797596  ...           20.0
## Spain                            16.430249  ...            9.1
## Sweden                           19.916998  ...            6.8
## Switzerland                      24.651815  ...            4.5
## Turkey                            5.071860  ...           39.7
## United Kingdom                   22.697062  ...           24.8
## United States                    29.196531  ...           49.8
##
## [30 rows x 13 columns]
## (30, 13)
```

d) (zu **R**)

Mit **`KNN(k=x).fit_transform(data)`** werden die fehlenden Werte aufgrund der k nächstliegenden Werte (Nachbarn) ersetzt. Die Anzahl k der zu betrachtenden nächsten Nachbarn muss festgelegt werden. Standardmässig werden alle k Werte gleich (**`uniform`**) zur Berechnung des Ersatzwertes gewichtet. Es gibt aber auch andere Gewichtungsmethoden.

```
from fancyimpute import KNN
values = data.values
data_imputed = DataFrame(KNN(k=3).fit_transform(values))
data_imputed
print(data_imputed)

## Using TensorFlow backend.
## -c:5: FutureWarning: read_table is deprecated, use read_csv instead.
## Imputing row 1/30 with 5 missing, elapsed time: 0.001
##            0          1     2   ...    18         19         20
## 0   20.813221  11.791352   2.2  ...  14.3  10.693183  27.154130
## 1   22.162446   6.166094   0.6  ...  12.3  15.600000  38.100000
## 2   21.401153   9.974720   1.0  ...   7.8  12.200000  21.600000
## 3   25.606245  15.057585   2.1  ...  13.2  14.000000  29.500000
## 4   10.849270  10.270000   1.2  ...  11.4   5.500000  11.700000
## 5   23.175894   2.740000   0.7  ...   6.6   8.000000  25.600000
## 6   22.027651   4.170000   1.0  ...   9.7   8.000000  16.100000
## 7   18.960382   7.640000   1.2  ...   6.7  13.600000  21.400000
## 8   19.894067  16.289270   0.5  ...   9.8  13.900000  34.900000
## 9   17.183647  13.230296   6.1  ...   8.7  22.000000  25.600000
## 10   9.463130   8.724203   2.1  ...  20.7   6.600000  27.600000
## 11  22.286852   8.250000   0.4  ...  16.9   5.400000  36.600000
## 12  22.364689  16.299416   2.9  ...  13.5   8.600000  24.000000
## 13  17.180761  15.500000   1.2  ...   6.8   7.900000  12.800000
## 14  22.479705  13.688033   5.6  ...   3.7  12.079283  31.706649
## 15  21.651918  10.746834   1.8  ...   3.7   9.686507  27.502170
```

```
## 16  34.241822  12.390000   1.1  ...   8.6  13.800000  20.700000
## 17   5.335074  22.164514  13.7  ...  65.8  23.175605  47.635140
## 18  25.041012  11.526755   0.6  ...   4.7   8.500000  39.700000
## 19  17.197108  15.000000   2.2  ...  23.4  13.876911  31.188436
## 20  28.574371   4.600000   1.3  ...   9.4   8.300000  41.700000
## 21   7.939399  21.500000   2.1  ...  14.5   9.600000  21.100000
## 22   3.839462  16.550398   1.4  ...  18.1  14.100000  22.800000
## 23   7.797596  10.930000   3.8  ...  20.0   7.092834  13.000000
## 24  16.430249  17.300000   0.9  ...   9.1   4.700000  23.900000
## 25  19.916998   3.969106   1.6  ...   6.8   4.200000  24.100000
## 26  24.651815   9.433293   0.7  ...   4.5  12.100000  27.300000
## 27   5.071860  24.590000  13.6  ...  39.7  25.300000  57.400000
## 28  22.697062  10.080000   1.8  ...  24.8   9.700000  35.700000
## 29  29.196531  20.593932   4.8  ...  49.8  11.900000  26.400000
##
## [30 rows x 21 columns]
```

e) Gehen wir davon aus, dass OECD Staaten nichts zu verbergen haben, so dürfen wir MAR annehmen.

R-Code

Aufgabe 2.1

a) (zu **Python**)

```
noten.1 <- c(4.2, 2.3, 5.6, 4.5, 4.8, 3.9, 5.9, 2.4, 5.9,
    6, 4, 3.7, 5, 5.2, 4.5, 3.6, 5, 6, 2.8, 3.3, 5.5, 4.2,
    4.9, 5.1)
median(noten.1)

## [1] 4.65

mean(noten.1)

## [1] 4.5125
```

(zu **Python**)

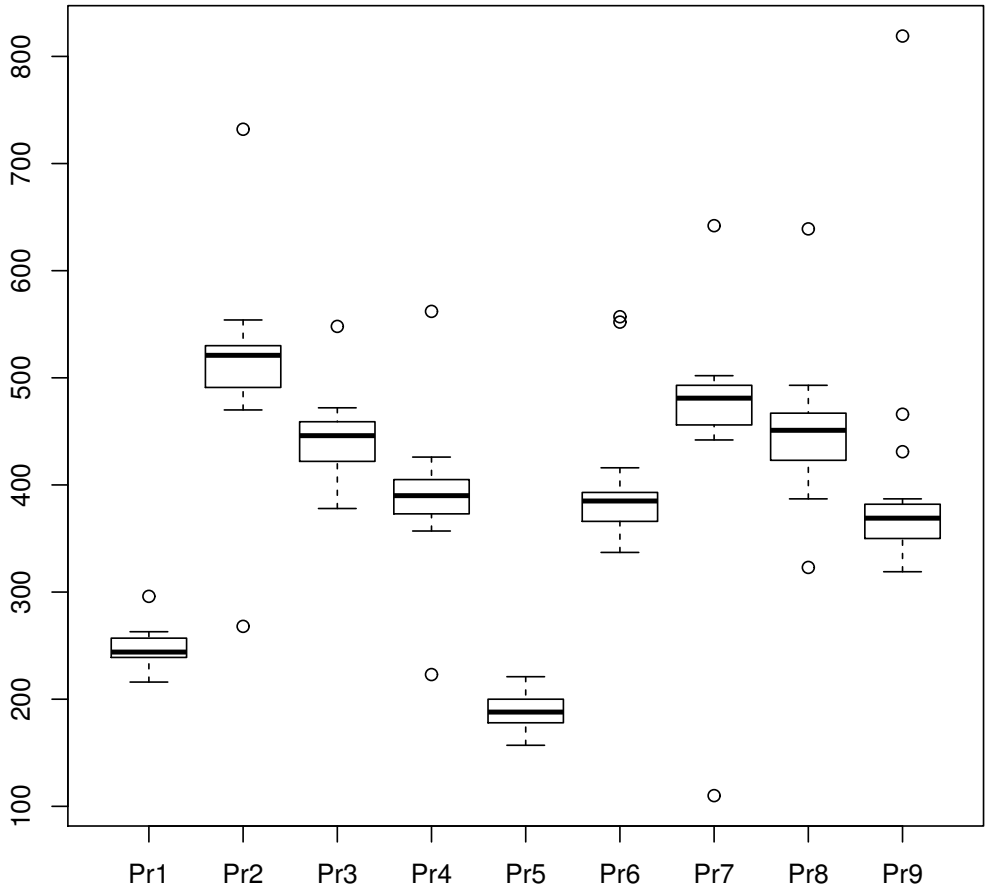
```
sort(noten.1)

##  [1] 2.3 2.4 2.8 3.3 3.6 3.7 3.9 4.0 4.2 4.2 4.5
## [12] 4.5 4.8 4.9 5.0 5.0 5.1 5.2 5.5 5.6 5.9 5.9
## [23] 6.0 6.0
```

```
noten.2 <- c(1, 2.3, 5.6, 4.5, 4.8, 3.9, 5.9, 2.4, 5.9,
    6, 4, 3.7, 5, 5.2, 1, 3.6, 5, 6, 2.8, 3.3, 5.5, 1, 4.9,
    5.1)
median(noten.2)

## [1] 4.65

mean(noten.2)

## [1] 4.1
```

a) (zu **Python**)

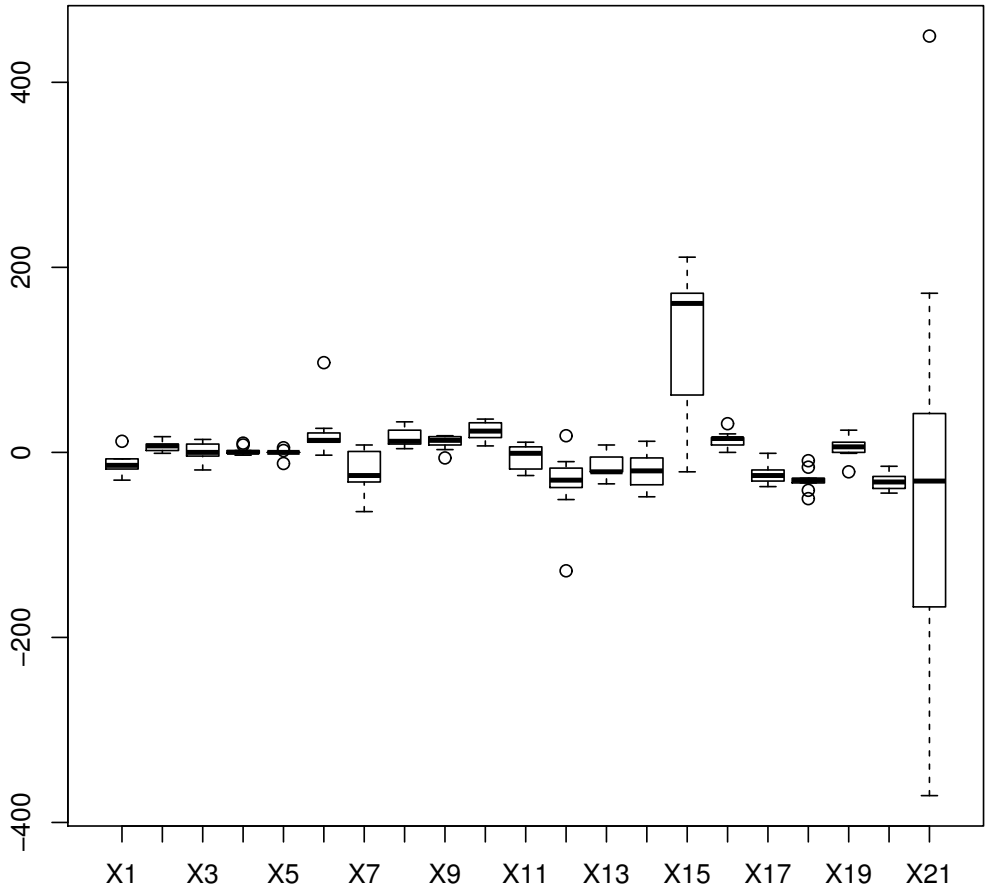
```
schlamm.all <- read.table(file = "./Daten/klaerschlamm.dat",
    header = TRUE)
schlamm <- schlamm.all[, -1]  # Labor-Spalte entfernen
summary(schlamm)

##       Pr1             Pr2             Pr3
##  Min.   :216.0   Min.   :268.0   Min.   :378.0
##  1st Qu.:239.0   1st Qu.:491.0   1st Qu.:422.0
##  Median :244.0   Median :521.0   Median :446.0
##  Mean   :246.1   Mean   :511.4   Mean   :443.4
```

```
##  3rd Qu.:257.0   3rd Qu.:530.0   3rd Qu.:459.0
##  Max.   :296.0   Max.   :732.0   Max.   :548.0
##       Pr4             Pr5             Pr6
##  Min.   :223.0   Min.   :157.0   Min.   :337.0
##  1st Qu.:373.0   1st Qu.:178.0   1st Qu.:366.0
##  Median :390.0   Median :188.0   Median :385.0
##  Mean   :389.2   Mean   :188.2   Mean   :394.9
##  3rd Qu.:405.0   3rd Qu.:200.0   3rd Qu.:393.0
##  Max.   :562.0   Max.   :221.0   Max.   :557.0
##       Pr7             Pr8           Pr9
##  Min.   :110.0   Min.   :323   Min.   :319.0
##  1st Qu.:456.0   1st Qu.:423   1st Qu.:350.0
##  Median :481.0   Median :451   Median :369.0
##  Mean   :465.5   Mean   :450   Mean   :388.9
##  3rd Qu.:493.0   3rd Qu.:467   3rd Qu.:382.0
##  Max.   :642.0   Max.   :639   Max.   :819.0

boxplot(schlamm)
```



b)

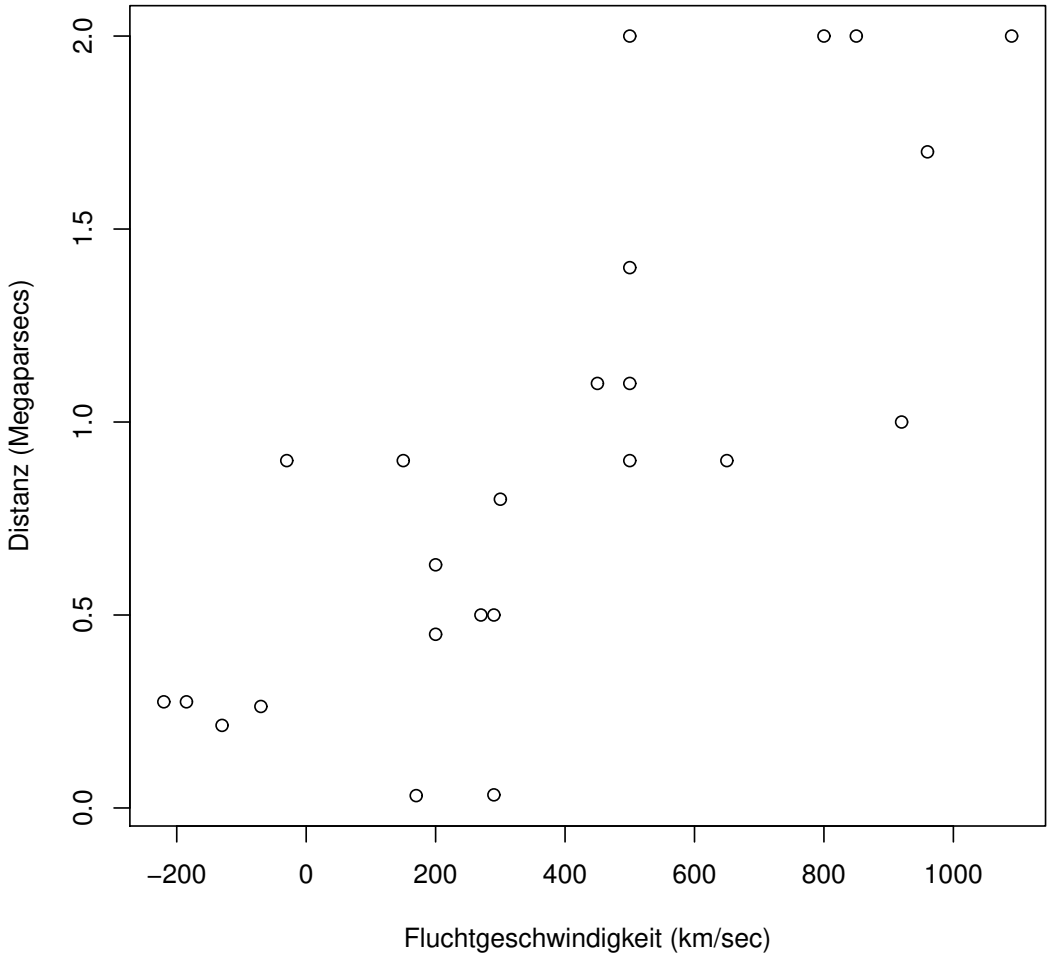
```
# Fuer jede Spalte Median berechnen
med <- apply(schlamm, 2, median)
# Median von jeder *Spalte* abziehen
schlamm.centered <- scale(schlamm, scale = FALSE, center = med)
# Boxplot zeichnen. Dazu zuerst data-frame
# transponieren
boxplot(data.frame(t(schlamm.centered)))
```



2.4

a) (zu **Python**)

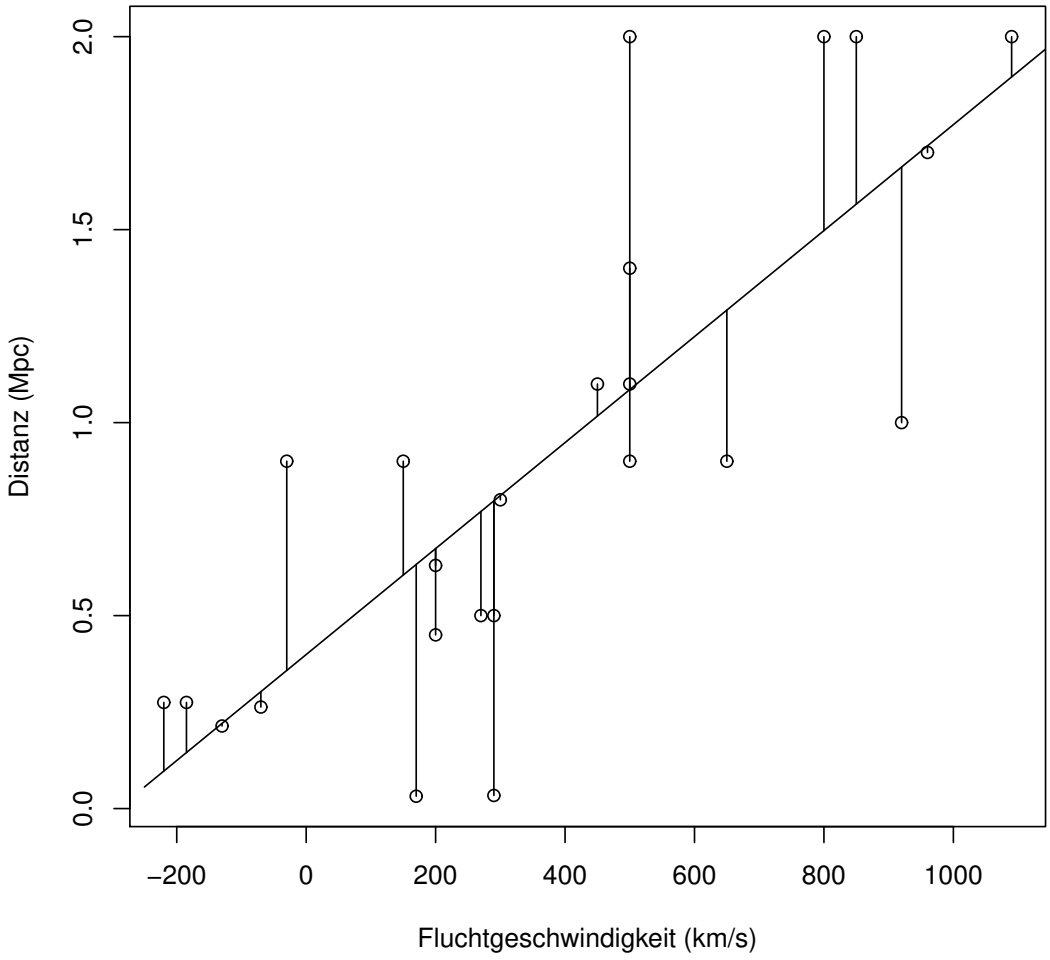
```
recession.velocity <- c(170, 290, -130, -70, -185,
    -220, 200, 290, 270, 200, 300, -30, 650, 150, 500,
    920, 450, 500, 500, 960, 500, 850, 800, 1090)
distance <- c(0.032, 0.034, 0.214, 0.263, 0.275, 0.275,
    0.45, 0.5, 0.5, 0.63, 0.8, 0.9, 0.9, 0.9, 0.9,
    1, 1.1, 1.1, 1.4, 1.7, 2, 2, 2, 2)
plot(recession.velocity, distance, ylab = "Distanz (Megaparsecs)",
    xlab = "Fluchtgeschwindigkeit (km/sec)")
```



b) (zu **Python**)

```
regr_model <- lm(distance ~ recession.velocity)
beta_0 <- regr_model$coefficients[1]
beta_0

## (Intercept)
##   0.3990982

beta_1 <- regr_model$coefficients[2]
beta_1

## recession.velocity
##        0.001372936
```

```r
plot(recession.velocity, distance, ylab = "Distanz (Mpc)",
    xlab = "Fluchtgeschwindigkeit (km/s)")
lines(-250:1200, beta_0 + beta_1 * (-250:1200), type = "l",
    new = TRUE)
segments(recession.velocity, beta_0 + beta_1 * (recession.velocity),
    recession.velocity, distance)
```



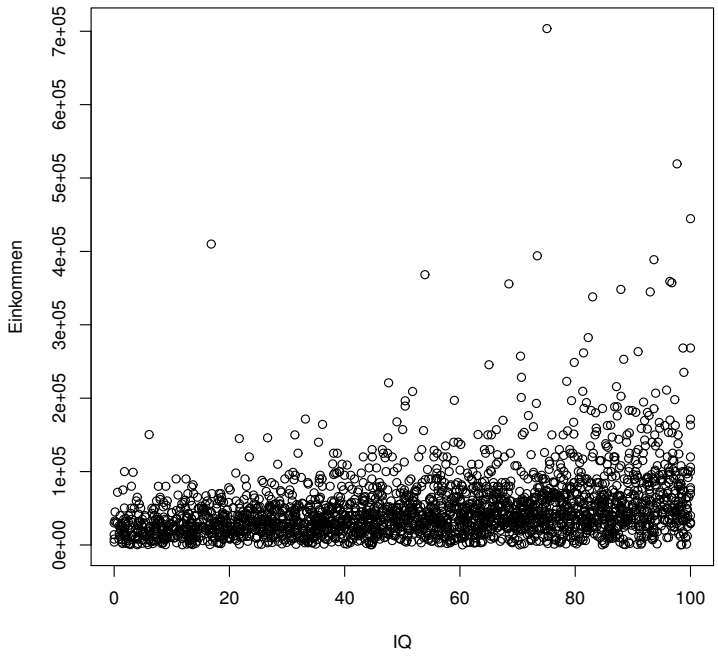
Aufgabe 2.5

a) (zu **Python**)

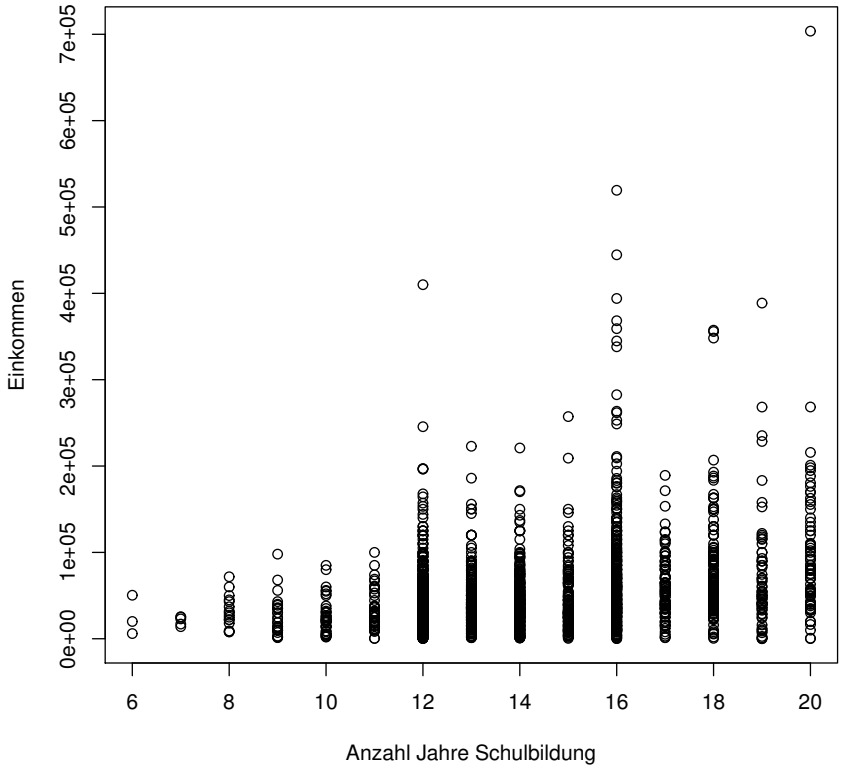
```r
income <- read.table(file = "./Daten/income.dat", header = TRUE)
head(income)
```

```
##      AFQT Educ Income2005
## 1   6.841   12       5500
## 2  99.393   16      65000
## 3  47.412   12      19000
## 4  44.022   14      36000
## 5  59.683   14      65000
## 6  72.313   16       8000
```

```
iq <- income[, 1]
anzahl.jahre.schule <- income[, 2]
einkommen <- income[, 3]
plot(iq, einkommen, type = "p", xlab = "IQ", ylab = "Einkommen")
```




```
plot(anzahl.jahre.schule, einkommen, type = "p",
    xlab = "Anzahl Jahre Schulbildung", ylab = "Einkommen")
```



b) (zu **Python**)

```
lm(einkommen ~ iq)

##
## Call:
## lm(formula = einkommen ~ iq)
##
## Coefficients:
## (Intercept)           iq
##     21181.7        518.7

lm(einkommen ~ anzahl.jahre.schule)

##
## Call:
## lm(formula = einkommen ~ anzahl.jahre.schule)
##
## Coefficients:
##         (Intercept)  anzahl.jahre.schule
##               -40200                 6451
```

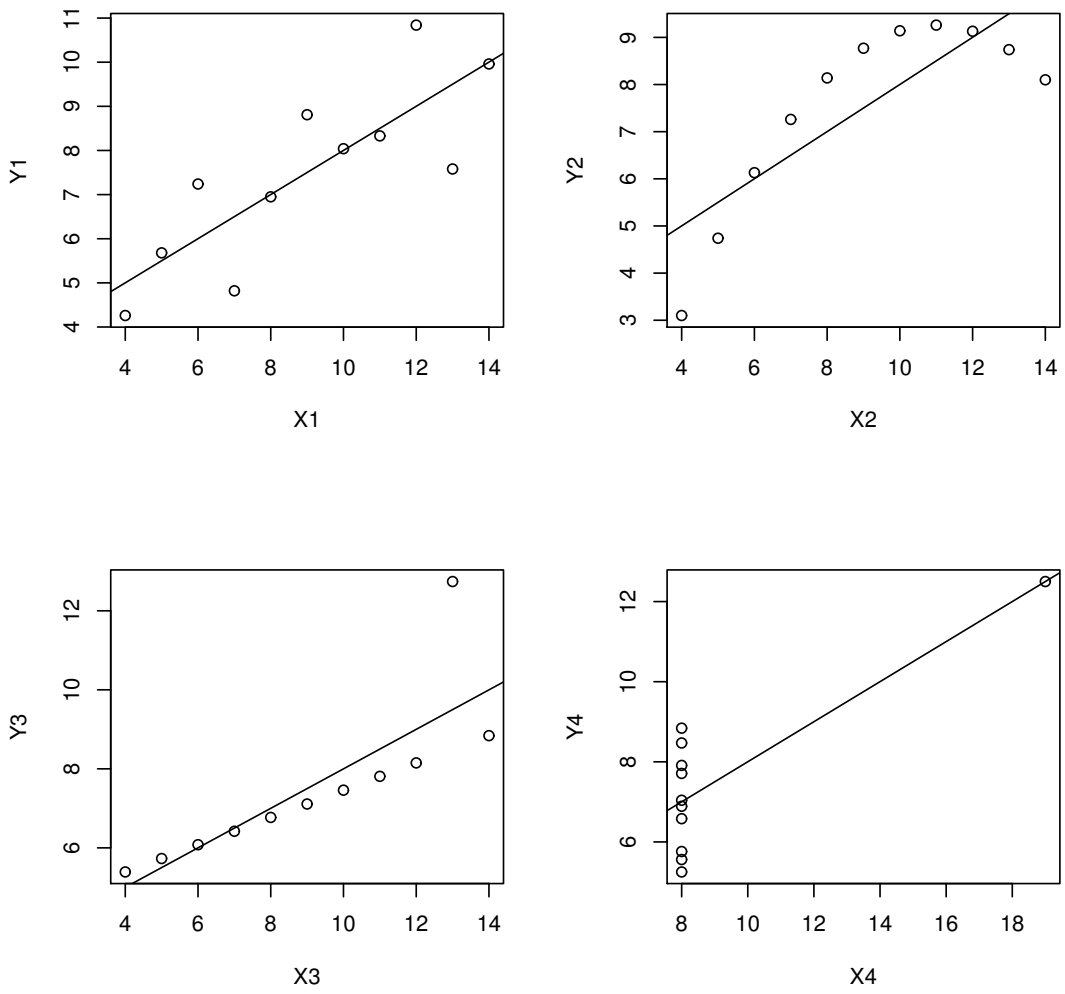
c) (zu **Python**)

```
cor(anzahl.jahre.schule, einkommen)

## [1] 0.3456474
```

Aufgabe 2.6

a) (zu **Python**)

```
data(anscombe)  ## Einlesen des Datensatzes

reg <- lm(anscombe$y1 ~ anscombe$x1)
reg2 <- lm(anscombe$y2 ~ anscombe$x2)
reg3 <- lm(anscombe$y3 ~ anscombe$x3)
reg4 <- lm(anscombe$y4 ~ anscombe$x4)
par(mfrow = c(2, 2))
plot(anscombe$x1, anscombe$y1, ylab = "Y1", xlab = "X1")
abline(reg)
plot(anscombe$x2, anscombe$y2, ylab = "Y2", xlab = "X2")
abline(reg2)
plot(anscombe$x3, anscombe$y3, ylab = "Y3", xlab = "X3")
abline(reg3)
plot(anscombe$x4, anscombe$y4, ylab = "Y4", xlab = "X4")
abline(reg4)
```



b) (zu **Python**)

```
lm(anscombe$y1 ~ anscombe$x1)

##
## Call:
## lm(formula = anscombe$y1 ~ anscombe$x1)
##
## Coefficients:
## (Intercept)  anscombe$x1
##      3.0001       0.5001
```

c) (zu **Python**)

```
cor(anscombe$y1, anscombe$x1)
```

```
## [1] 0.8164205
```

Aufgabe 2.7

a) (zu **Python**)

```
data <- read.table(file = "./Daten/child.txt", header = TRUE,
    sep = ",")
summary(data)
```

```
##  Average.disposable.income Children.in.poor.homes
##  Min.   : 3.839            Min.   : 2.740
##  1st Qu.:16.618            1st Qu.: 8.901
##  Median :21.107            Median :11.659
##  Mean   :18.848            Mean   :12.372
##  3rd Qu.:22.643            3rd Qu.:16.092
##  Max.   :34.242            Max.   :24.590
##
##  Educational.Deprivation  Overcrowding
##  Min.   : 0.400          Min.   :10.33
##  1st Qu.: 1.000          1st Qu.:17.06
##  Median : 1.500          Median :21.57
##  Mean   : 2.673          Mean   :31.95
##  3rd Qu.: 2.200          3rd Qu.:44.39
##  Max.   :13.700          Max.   :73.96
##                          NA's   :4
##  Poor.environmental.conditions
##  Min.   :10.50
##  1st Qu.:20.15
##  Median :25.49
##  Mean   :25.22
##  3rd Qu.:30.24
##  Max.   :38.71
##  NA's   :6
##  Average.mean.literacy.score Literacy.inequality
##  Min.   :408.7               Min.   :1.475
##  1st Qu.:482.8               1st Qu.:1.623
##  Median :501.3               Median :1.683
##  Mean   :496.3               Mean   :1.665
##  3rd Qu.:512.8               3rd Qu.:1.719
##  Max.   :552.7               Max.   :1.756
##
##  Youth.NEET.rate  Low.birth.weight
```

```
##  Min.   : 1.700   Min.   : 3.900
##  1st Qu.: 4.550   1st Qu.: 5.150
##  Median : 6.200   Median : 6.750
##  Mean   : 7.378   Mean   : 6.643
##  3rd Qu.: 8.400   3rd Qu.: 7.500
##  Max.   :37.700   Max.   :11.300
##  NA's   :3
##  Infant.mortality Breastfeeding.rates
##  Min.   : 2.300   Min.   :41.00
##  1st Qu.: 3.525   1st Qu.:79.00
##  Median : 4.200   Median :91.00
##  Mean   : 5.447   Mean   :86.03
##  3rd Qu.: 5.250   3rd Qu.:96.00
##  Max.   :23.600   Max.   :99.00
##                   NA's   :1
##  Vaccination.rates..pertussis.
##  Min.   :78.00
##  1st Qu.:91.00
##  Median :95.80
##  Mean   :93.78
##  3rd Qu.:97.80
##  Max.   :99.80
##  NA's   :1
##  Vaccination.rates.measles. Physical.activity
##  Min.   :74.00              Min.   :13.10
##  1st Qu.:88.00              1st Qu.:15.80
##  Median :94.00              Median :19.30
##  Mean   :91.52              Mean   :20.13
##  3rd Qu.:96.30              3rd Qu.:21.80
##  Max.   :99.80              Max.   :42.10
##  NA's   :1                  NA's   :4
##  Mortality.rates Suicide.rates       Smoking
##  Min.   :14.84   Min.   : 1.263   Min.   : 8.10
##  1st Qu.:21.17   1st Qu.: 5.037   1st Qu.:14.62
##  Median :23.15   Median : 6.785   Median :16.60
##  Mean   :24.60   Mean   : 6.856   Mean   :16.51
##  3rd Qu.:25.75   3rd Qu.: 8.864   3rd Qu.:19.50
##  Max.   :50.23   Max.   :15.950   Max.   :27.10
##  NA's   :1       NA's   :1        NA's   :6
##   Drunkenness    Teenage.births     Bullying
##  Min.   :10.00   Min.   : 3.70   Min.   : 4.200
##  1st Qu.:11.35   1st Qu.: 7.05   1st Qu.: 7.975
##  Median :14.55   Median :10.60   Median : 9.650
##  Mean   :15.22   Mean   :15.50   Mean   :10.979
```

```
##  3rd Qu.:17.93   3rd Qu.:17.80   3rd Qu.:13.825
##  Max.   :24.80   Max.   :65.80   Max.   :25.300
##  NA's   :6                       NA's   :6
##  Liking.school
##  Min.   :11.70
##  1st Qu.:21.40
##  Median :25.60
##  Mean   :27.17
##  3rd Qu.:34.90
##  Max.   :57.40
##  NA's   :5
```

b) (zu **Python**)

```
library(tidyr)
summary(drop_na(data))

##  Average.disposable.income Children.in.poor.homes
##  Min.   : 3.839            Min.   : 2.740
##  1st Qu.:17.181            1st Qu.: 6.535
##  Median :20.659            Median :10.175
##  Mean   :19.464            Mean   :10.868
##  3rd Qu.:22.614            3rd Qu.:16.092
##  Max.   :29.197            Max.   :20.594
##  Educational.Deprivation  Overcrowding
##  Min.   :0.500           Min.   :10.33
##  1st Qu.:0.925           1st Qu.:15.53
##  Median :1.200           Median :20.15
##  Mean   :1.717           Mean   :28.17
##  3rd Qu.:1.750           3rd Qu.:33.50
##  Max.   :6.100           Max.   :73.31
##  Poor.environmental.conditions
##  Min.   :11.99
##  1st Qu.:20.67
##  Median :25.62
##  Mean   :26.17
##  3rd Qu.:31.20
##  Max.   :38.71
##  Average.mean.literacy.score Literacy.inequality
##  Min.   :464.0               Min.   :1.475
##  1st Qu.:482.9               1st Qu.:1.623
##  Median :501.3               Median :1.692
##  Mean   :496.8               Mean   :1.671
##  3rd Qu.:504.8               3rd Qu.:1.725
##  Max.   :552.7               Max.   :1.753
```

```
##  Youth.NEET.rate  Low.birth.weight
##  Min.   : 2.500   Min.   :4.100
##  1st Qu.: 4.475   1st Qu.:5.225
##  Median : 6.150   Median :6.800
##  Mean   : 6.317   Mean   :6.550
##  3rd Qu.: 8.025   3rd Qu.:7.500
##  Max.   :11.200   Max.   :8.800
##  Infant.mortality Breastfeeding.rates
##  Min.   :2.400    Min.   :41.00
##  1st Qu.:3.525    1st Qu.:74.90
##  Median :3.950    Median :86.05
##  Mean   :4.156    Mean   :83.27
##  3rd Qu.:4.625    3rd Qu.:95.95
##  Max.   :6.800    Max.   :99.00
##  Vaccination.rates..pertussis.
##  Min.   :83.00
##  1st Qu.:91.10
##  Median :95.40
##  Mean   :93.97
##  3rd Qu.:97.60
##  Max.   :99.80
##  Vaccination.rates.measles. Physical.activity
##  Min.   :74.00              Min.   :13.50
##  1st Qu.:87.47              1st Qu.:15.80
##  Median :92.10              Median :19.30
##  Mean   :90.88              Mean   :19.64
##  3rd Qu.:96.22              3rd Qu.:21.80
##  Max.   :99.80              Max.   :31.10
##  Mortality.rates Suicide.rates       Smoking
##  Min.   :19.27   Min.   : 1.263   Min.   : 8.10
##  1st Qu.:21.30   1st Qu.: 3.115   1st Qu.:15.22
##  Median :23.04   Median : 6.077   Median :17.80
##  Mean   :24.06   Mean   : 6.072   Mean   :17.08
##  3rd Qu.:24.83   3rd Qu.: 8.657   3rd Qu.:19.88
##  Max.   :34.60   Max.   :12.159   Max.   :27.10
##   Drunkenness    Teenage.births     Bullying
##  Min.   :10.00   Min.   : 4.70   Min.   : 4.200
##  1st Qu.:11.43   1st Qu.: 7.05   1st Qu.: 7.925
##  Median :14.55   Median : 9.55   Median : 8.550
##  Mean   :15.35   Mean   :13.15   Mean   :10.183
##  3rd Qu.:17.48   3rd Qu.:13.20   3rd Qu.:13.250
##  Max.   :24.80   Max.   :49.80   Max.   :22.000
##  Liking.school
##  Min.   :11.70
```

```
##  1st Qu.:21.90
##  Median :24.85
##  Mean   :26.32
##  3rd Qu.:33.08
##  Max.   :41.70

dim(drop_na(data))

## [1] 18 21
```

c) (zu **Python**)

(zu **Python**)

```
summary(data[, !(nrow(data) - colSums(!is.na(data)) >
    2)])

##  Average.disposable.income Children.in.poor.homes
##  Min.   : 3.839            Min.   : 2.740
##  1st Qu.:16.618            1st Qu.: 8.901
##  Median :21.107            Median :11.659
##  Mean   :18.848            Mean   :12.372
##  3rd Qu.:22.643            3rd Qu.:16.092
##  Max.   :34.242            Max.   :24.590
##
##  Educational.Deprivation
##  Min.   : 0.400
##  1st Qu.: 1.000
##  Median : 1.500
##  Mean   : 2.673
##  3rd Qu.: 2.200
##  Max.   :13.700
##
##  Average.mean.literacy.score Literacy.inequality
##  Min.   :408.7               Min.   :1.475
##  1st Qu.:482.8               1st Qu.:1.623
##  Median :501.3               Median :1.683
##  Mean   :496.3               Mean   :1.665
##  3rd Qu.:512.8               3rd Qu.:1.719
##  Max.   :552.7               Max.   :1.756
##
##  Low.birth.weight Infant.mortality
##  Min.   : 3.900   Min.   : 2.300
##  1st Qu.: 5.150   1st Qu.: 3.525
##  Median : 6.750   Median : 4.200
##  Mean   : 6.643   Mean   : 5.447
```

```
##  3rd Qu.: 7.500   3rd Qu.: 5.250
##  Max.   :11.300   Max.   :23.600
##
##  Breastfeeding.rates
##  Min.   :41.00
##  1st Qu.:79.00
##  Median :91.00
##  Mean   :86.03
##  3rd Qu.:96.00
##  Max.   :99.00
##  NA's   :1
##  Vaccination.rates..pertussis.
##  Min.   :78.00
##  1st Qu.:91.00
##  Median :95.80
##  Mean   :93.78
##  3rd Qu.:97.80
##  Max.   :99.80
##  NA's   :1
##  Vaccination.rates.measles. Mortality.rates
##  Min.   :74.00              Min.   :14.84
##  1st Qu.:88.00              1st Qu.:21.17
##  Median :94.00              Median :23.15
##  Mean   :91.52              Mean   :24.60
##  3rd Qu.:96.30              3rd Qu.:25.75
##  Max.   :99.80              Max.   :50.23
##  NA's   :1                  NA's   :1
##  Suicide.rates    Teenage.births
##  Min.   : 1.263   Min.   : 3.70
##  1st Qu.: 5.037   1st Qu.: 7.05
##  Median : 6.785   Median :10.60
##  Mean   : 6.856   Mean   :15.50
##  3rd Qu.: 8.864   3rd Qu.:17.80
##  Max.   :15.950   Max.   :65.80
##  NA's   :1

dim(data[, !(nrow(data) - colSums(!is.na(data)) >
    2)])

## [1] 30 13
```

d) (zu **Python**)

(zu **Python**)

```r
library(DMwR)

## Loading required package:  lattice

## Loading required package:  grid

knnOutput <- knnImputation(data[, !names(data) %in%
    "medv"], k = 3)
summary(knnOutput)

##  Average.disposable.income Children.in.poor.homes
##  Min.   : 3.839            Min.   : 2.740
##  1st Qu.:16.618            1st Qu.: 8.901
##  Median :21.107            Median :11.659
##  Mean   :18.848            Mean   :12.372
##  3rd Qu.:22.643            3rd Qu.:16.092
##  Max.   :34.242            Max.   :24.590
##  Educational.Deprivation  Overcrowding
##  Min.   : 0.400          Min.   :10.33
##  1st Qu.: 1.000          1st Qu.:17.05
##  Median : 1.500          Median :20.89
##  Mean   : 2.673          Mean   :30.77
##  3rd Qu.: 2.200          3rd Qu.:39.99
##  Max.   :13.700          Max.   :73.96
##  Poor.environmental.conditions
##  Min.   :10.50
##  1st Qu.:20.47
##  Median :25.70
##  Mean   :25.33
##  3rd Qu.:29.75
##  Max.   :38.71
##  Average.mean.literacy.score Literacy.inequality
##  Min.   :408.7               Min.   :1.475
##  1st Qu.:482.8               1st Qu.:1.623
##  Median :501.3               Median :1.683
##  Mean   :496.3               Mean   :1.665
##  3rd Qu.:512.8               3rd Qu.:1.719
##  Max.   :552.7               Max.   :1.756
##  Youth.NEET.rate  Low.birth.weight
##  Min.   : 1.700   Min.   : 3.900
##  1st Qu.: 4.400   1st Qu.: 5.150
##  Median : 6.200   Median : 6.750
##  Mean   : 7.145   Mean   : 6.643
##  3rd Qu.: 8.150   3rd Qu.: 7.500
##  Max.   :37.700   Max.   :11.300
```

```
##  Infant.mortality Breastfeeding.rates
##  Min.   : 2.300   Min.   :41.00
##  1st Qu.: 3.525   1st Qu.:79.53
##  Median : 4.200   Median :91.50
##  Mean   : 5.447   Mean   :86.24
##  3rd Qu.: 5.250   3rd Qu.:95.95
##  Max.   :23.600   Max.   :99.00
##  Vaccination.rates..pertussis.
##  Min.   :78.00
##  1st Qu.:91.10
##  Median :95.90
##  Mean   :93.91
##  3rd Qu.:97.77
##  Max.   :99.80
##  Vaccination.rates.measles. Physical.activity
##  Min.   :74.00              Min.   :13.10
##  1st Qu.:88.00              1st Qu.:16.55
##  Median :93.35              Median :18.90
##  Mean   :91.46              Mean   :19.98
##  3rd Qu.:96.22              3rd Qu.:21.05
##  Max.   :99.80              Max.   :42.10
##  Mortality.rates Suicide.rates       Smoking
##  Min.   :14.84   Min.   : 1.263   Min.   : 8.10
##  1st Qu.:21.18   1st Qu.: 4.939   1st Qu.:13.07
##  Median :23.29   Median : 6.784   Median :16.45
##  Mean   :24.73   Mean   : 6.733   Mean   :16.19
##  3rd Qu.:27.54   3rd Qu.: 8.776   3rd Qu.:19.38
##  Max.   :50.23   Max.   :15.950   Max.   :27.10
##   Drunkenness    Teenage.births     Bullying
##  Min.   :10.00   Min.   : 3.70   Min.   : 4.200
##  1st Qu.:11.47   1st Qu.: 7.05   1st Qu.: 7.352
##  Median :14.55   Median :10.60   Median : 9.650
##  Mean   :15.06   Mean   :15.50   Mean   :10.648
##  3rd Qu.:17.48   3rd Qu.:17.80   3rd Qu.:13.384
##  Max.   :24.80   Max.   :65.80   Max.   :25.300
##  Liking.school
##  Min.   :11.70
##  1st Qu.:21.90
##  Median :26.00
##  Mean   :27.74
##  3rd Qu.:33.38
##  Max.   :57.40

dim(knnOutput)
```

```
## [1] 30 21
```