Serie 1

Aufgabe 1.1

Der Datensatz der OECD enthält Messgrössen, die das Wohlergehen von Kindern in den Mitgliedsstaaten ermitteln sollen. Im Jahr 2009 wurde abgefragt:

- Einkommen (Average disposable income): das durchschnittliche Einkommen der Eltern [in tausend US Dollar pro Kind].
- Armut (Children in poor homes): der Anteil [immer in Prozent] an Kindern in einem armen Elternhaus.
- Bildung (Educational Deprivation): der Anteil an Kindern, die ohne Grundausstattung (Bücher, Schreibtisch, Computer, Internet) für Bildung auskommen müssen.
- Wenig Raum (Overcrowding): der Anteil an Kindern, die auf zu wenig Raum wohnen.
- Umwelt (Poor environmental conditions): der Anteil an Kindern, die unter schlechten Umweltbedingungen leben.
- Lesen (Average mean literacy score): mittlerer PISA-Score zur Lesefähigkeit.
- Geburtsgewicht (Low birth weight): der Anteil an Kindern, die bei der Geburt weniger als 2.5 kg wiegen.
- Säuglingssterblichkeit (Infant mortality): Säuglingssterblichkeit (< 1 Jahr) [x in Tausend].
- Sterblichkeit (Mortality rates): Sterblichkeit (< 20 Jahre) [x in 100 000].
- Selbstmord (Suicide rates): Selbstmord von Jugendlichen im Alter von 15 bis 19 [x in 100 000].
- Bewegung (Physical activity): der Anteil an 11, 13 und 15 jährigen Jugendlichen, die sich regelmässig bewegen.
- Rauchen (Smoking): der Anteil an 15 jährigen Jugendlichen, die mindestens einmal die Woche rauchen.
- Alkohol (Drunkenness): der Anteil an 13-15 jährigen Jugendlichen, die mindestens zweimal betrunken waren.
- Bullying (Bullying): der Anteil an Kindern, die angeben, in der Schule bedroht zu werden.

- Schule (Liking school): der Anteil an Kindern, die angeben die Schule zu mögen.

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(r"*child.csv", sep=",", index_col=0)
```

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

Das Argument **`sep=","`** braucht es, weil die Kolonnen im File `child.csv` durch Kommata getrennt sind.

Das Argument **`index_col=0`** erreicht, dass die 1. Spalte des Files als Index erkannt wird.

Mögliche Schwierigkeiten beim Einlesen:

- Fehlermeldung: „... file not found..." → Falscher Pfad
- Fehlermeldung: „... unicodexxxx ..." → Leerzeichen in Ordnernamen (vermeiden Sie diese)
- Fehlermeldung: „... list has no end ..." → File wurde falsch eingelesen (nicht Ihr Fehler). Versuchen Sie es nochmals.
- Achtung: Die Datei *nicht* mit Excel oder einem anderen Spreadsheet öffenen und abspeichern!

b) Überprüfen Sie mit dem Attribut **`.shape`** die Dimension der Daten.

c) Bestimmen Sie den Mittelwert und Median der einzelnen Variablen mit dem Python-Attribut **`.describe()`**.

d) Überprüfen Sie, ob die Niederlande in der Länderliste des Datensatzes auftaucht. Gibt es auch einen Eintrag für China? Die Zeilennamen ermitteln Sie mit dem Attribut **`.index`**.

Wie erhalten Sie die Spaltennamen?

e) In welchen fünf Ländern waren die meisten Jugendlichen mindestens zweimal betrunken? Wie hoch ist der maximale Prozentsatz? Benützen Sie die Methode **`.sort_values(by=..., ascending=...)`**. Verwenden Sie die Internetseiten oben, um die Punkte auszufüllen.

f) In welchem Land ist die Säuglingssterblichkeit am geringsten? Wie hoch ist sie in diesem Land? Benützen Sie die Methode **`.nsmallest(...)`**

g) In welchen Ländern ist der Prozentsatz an Jugendlichen, die sich regelmässig bewegen, kleiner als der Durchschnitt? Benützen Sie das Attribut **`.mean()`** und

```
data.loc[... < ..., :]
```

Aufgabe 1.2

Das Dataframe `d.fuel` enthält die Daten verschiedener Fahrzeuge aus einer amerikanischen Untersuchung der 80er-Jahre. Jede Zeile (row) enthält die Daten eines Fahrzeuges (ein Fahrzeug entspricht einer Beobachtung).

a) Lesen Sie die auf Ilias abgelegte Datei `d.fuel.dat` ein mit dem folgenden **`Pandas`**-Befehl:

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

fuel =  pd.read_table(r"*d.fuel.dat", sep=",", index_col=0)
```

Die Spalten (columns) enthalten die folgenden Variablen:

weight:	Gewicht in Pounds (1 Pound = 0.453 59 kg)
mpg:	Reichweite in Miles Per Gallon (1 gallon = 3.789 l; 1 mile = 1.6093 km)
type:	Autotyp

b) Wählen Sie nur die fünfte Zeile des Dataframe **`d.fuel`** aus. Welche Werte stehen in der fünften Zeile? Verwenden Sie das Attribut **`.loc`** (siehe Aufgabe 1)).

c) Wählen Sie nun die erste bis fünfte Beobachtung des Datensatzes aus. So lässt sich übrigens bei einem unbekannten Datensatz ein schneller Überblick über die Art des Dataframe gewinnen.

d) Berechnen Sie den Mittelwert der Reichweiten aller Autos in Miles/Gallon.

`Python` **-Hinweis**: Methode **`.mean()`**

e) Berechnen Sie den Mittelwert der Reichweite der Autos 7 bis 22.

f) Erzeugen Sie einen neuen Vektor `t_kml`, der alle Reichweiten in km/l, und einen Vektor `t_kg`, der alle Gewichte in kg enthält.

g) Berechnen Sie den Mittelwert der Reichweiten in km/l und denjenigen der Fahrzeuggewichte in kg.

Aufgabe 1.3

Bei der Ermittlung der landwirtschaftlichen Nutzfläche von Bauernhöfen in einem Bezirk ergaben sich folgende Werte (in ha):

```
2.1,2.4,2.8,3.1,4.2,4.9,5.1,6.0,6.4,7.3,10.8,12.5,13.0,13.7,14.8,17.6,19.6,
23.0,25.0,35.2,39.6
```

a) Berechnen Sie die Summen $\sum x_i$ und $\sum x_i^2$.

   `Python` -Hinweis: Verwenden Sie die Methode **`.sum()`**

b) Berechnen Sie den Mittelwert und die Standardabweichung (ohne die in **`pandas`** implementierten Funktionen, sondern aufgrund der Definition der Grössen).

   `Python` -Hinweis: Verwenden Sie die Methode **`.size()`**

c) Bestimmen Sie den Median (ohne die in **`pandas`** implementierte Funktion, sondern aufgrund der Definition der Grössen).

   `Python` -Hinweis: Verwenden Sie zum Sortieren die Methode **`.sort_values()`** und den Befehl **`np.round()`** zum Runden.

d) Bestimmen Sie nun Mittelwert, Standardabweichung, Median und das 75 % Quantil mit den Methoden **`.mean()`**, **`.std()`**, **`.median()`**.

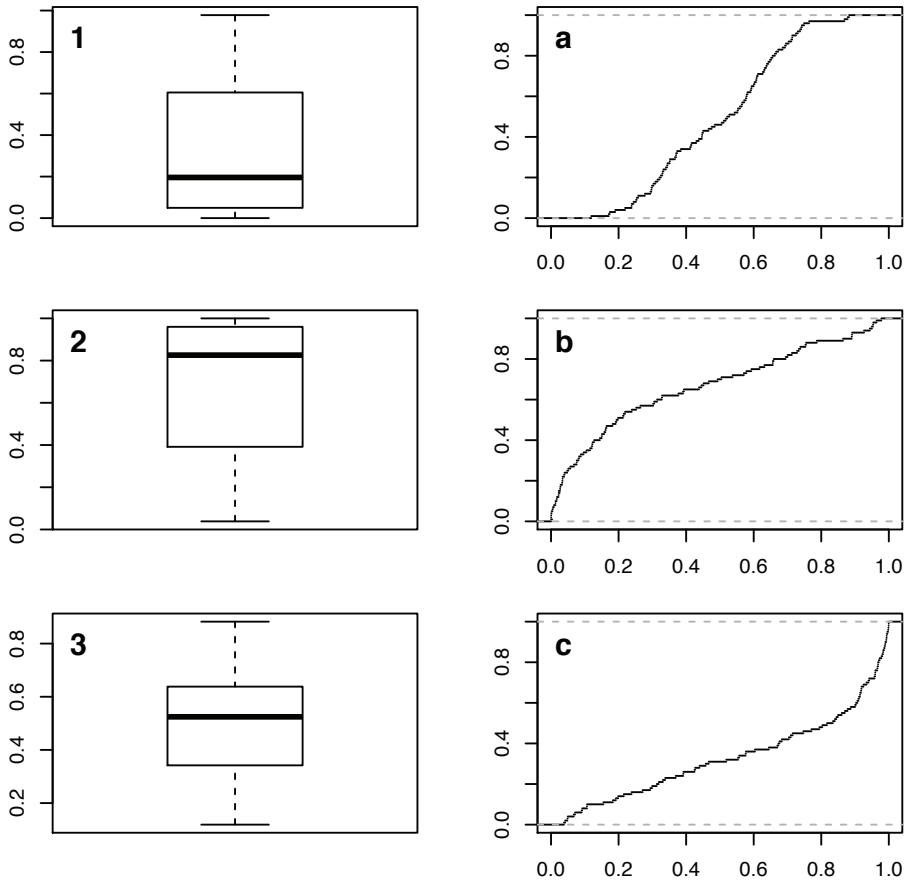
e) Überprüfen Sie aufgrund des Datenvektors mit den landwirtschaftlichen Nutzflächen, dass das arithmetische Mittel der **standardisierten** Variablen

$$z_i = \frac{x_i - \overline{x}}{s_x} \quad \text{mit} \quad i = 1, \ldots, n$$

   gleich null und die empirische Standardabweichung von z_i gleich 1 ist.

Aufgabe 1.4

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 1.5

Der Geysir Old Faithful im Yellowstone National Park ist eine der bekanntesten heissen Quellen. Für die Zuschauer und den Nationalparkdienst ist die Zeitspanne zwischen zwei Ausbrüchen und die Eruptionsdauer von grossem Interesse. Auf Ilias sind die Messungen in der Datei `geysir.dat` vom 1.8.1978 - 8.8.1978 in 3 Spalten abgelegt: `Tag`, `Zeitspanne` und `Eruptionsdauer`.

a) Zeichnen Sie Histogramme von der Zeitspanne zwischen zwei Ausbrüchen. Lesen Sie zuerst die Datei ein

```
from pandas import Series, DataFrame
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

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

Für * geben Sie wieder den gesamten Dateipfad der Datei ein.

Mit der Methode **`.head()`** können Sie überprüfen, ob die Datei richtig eingelesen wurde:

```
geysir.head()

##    Tag  Zeitspanne  Eruptionsdauer
## 1    1          78             4.4
## 2    1          74             3.9
## 3    1          68             4.0
## 4    1          76             4.0
## 5    1          80             3.5
```

Wir zeichnen nun die Histogramme. Um die Histogramme besser mit einander vergleichen zu können, zeichnen wir diese in Teilplots. Die geschieht mit dem Befehl

```
plt.subplot(221)
```

Dies erzeugt vier Fenster in der Graphik, nämlich zwei Zeilen und zwei Spalten. Die 1 am Schluss bedeutet, dass der nächste Plot in das erste Fenster gezeichnet wird.

```
plt.subplot(221)
geysir["Zeitspanne"].plot(kind="hist", edgecolor="black")
plt.xlabel("10 Klassen")

plt.subplot(222)
geysir["Zeitspanne"].plot(kind="hist",
                          bins=20,
                          edgecolor="black")
plt.xlabel("20 Klassen")

plt.subplot(223)
geysir["Zeitspanne"].plot(kind="hist",
                          bins=np.arange(41,107,11),
                          edgecolor="black")
plt.xlabel("Klassengrenzen 41, 52 , 63, 74 , 85, 96")

plt.show()
```

Was fällt auf? Was ist der Unterschied zwischen den drei Histogrammen?

b) Zeichnen Sie Histogramme (Anzahl Klassen variieren) von der Eruptionsdauer:

```
geysir["Eruptionsdauer"].plot(kind="hist")
```

Was fällt auf? Vergleichen Sie mit der ersten Teilaufgabe.

c) Zeichnen Sie die empirische kumulative Verteilungsfunktion von der Eruptionsdauer von Old Faithful Geysir. Untersuchen Sie, wie viel Prozent der Eruptionen höchstens 2 Minuten gedauert haben, sowie welche Eruptionsdauer der 60 % Eruptionen, die am längsten gedauert haben, mindestens gedauert haben.

```
geysir["Eruptionsdauer"].plot(kind="hist",
                              normed=True,
                              cumulative=True)
```

Kurzlösungen vereinzelter Aufgaben

A 1.3:

a) 269.1 , 5729.27

b) 12.81, 114 , 10.68

c) 10.8

d) 4.9, 17.6

Musterlösungen zu Serie 1

Lösung 1.1

a) (zu **R**)

Mit dem Attribut **`.head()`** können wir überprüfen, ob die Datei richtig eingelesen wurde. Dabei werden standardmässig die ersten fünf Zeilen ausgegeben.

```
data.head()

##                 Average.disposable.income      ...        Liking.school
## Australia                       20.813221      ...                  NaN
## Austria                         22.162446      ...                 38.1
## Belgium                         21.401153      ...                 21.6
## Canada                          25.606245      ...                 29.5
## Czech Republic                  10.849270      ...                 11.7
##
## [5 rows x 21 columns]
```

Sie sehen noch, dass Werte **`NaN`** vorkommen. 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**)

Die Dimension ermitteln wir dann mit

```
data.shape

## (30, 21)
```

Der Datensatz enthält also 30 Zeilen und 21 Spalten.

c) (zu **R**)

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

```
data.describe()

##        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]
```

Die Mittelwerte können wir auch wie folgt herauslesen: (zu **R**)

```
data.mean()
```

```
## Average.disposable.income          18.847713
## Children.in.poor.homes             12.372193
## Educational.Deprivation             2.673333
## Overcrowding                       31.950163
## Poor.environmental.conditions      25.217498
## Average.mean.literacy.score       496.317000
## Literacy.inequality                 1.665085
## Youth.NEET.rate                     7.377778
## Low.birth.weight                    6.643333
## Infant.mortality                    5.446667
## Breastfeeding.rates                86.027586
## Vaccination.rates..pertussis.      93.775862
## Vaccination.rates.measles.         91.517241
## Physical.activity                  20.134615
## Mortality.rates                    24.598966
## Suicide.rates                       6.856272
## Smoking                            16.512500
## Drunkenness                        15.225000
## Teenage.births                     15.500000
## Bullying                           10.979167
## Liking.school                      27.172000
## dtype: float64
```

Entsprechend gilt für den Median (zu **R**)

```
data.median()
```

```
## Average.disposable.income          21.107187
## Children.in.poor.homes             11.659053
## Educational.Deprivation             1.500000
## Overcrowding                       21.574977
## Poor.environmental.conditions      25.487116
## Average.mean.literacy.score       501.335000
## Literacy.inequality                 1.682739
## Youth.NEET.rate                     6.200000
## Low.birth.weight                    6.750000
## Infant.mortality                    4.200000
## Breastfeeding.rates                91.000000
## Vaccination.rates..pertussis.      95.800000
## Vaccination.rates.measles.         94.000000
## Physical.activity                  19.300000
## Mortality.rates                    23.150000
## Suicide.rates                       6.784772
## Smoking                            16.600000
## Drunkenness                        14.550000
```

```
## Teenage.births                          10.600000
## Bullying                                 9.650000
## Liking.school                           25.600000
## dtype: float64
```

d) (zu **R**)

Wir wollen die Zeilennamen unseres Datensatzes ermitteln

```
data.index

## Index(['Australia', 'Austria', 'Belgium', 'Canada', 'Czech Republic',
##        'Denmark', 'Finland', 'France', 'Germany', 'Greece', 'Hungary',
##        'Iceland', 'Ireland', 'Italy', 'Japan', 'Korea', 'Luxembourg', 'Mexico',
##        'Netherlands', 'New Zealand', 'Norway', 'Poland', 'Portugal',
##        'Slovak Republic', 'Spain', 'Sweden', 'Switzerland', 'Turkey',
##        'United Kingdom', 'United States'],
##       dtype='object')
```

Die Niederlande ist in dieser Liste enthalten, China hingegen nicht.

Wir können dies auch mit einem Befehl überprüfen: (zu **R**)

```
"China" in data.index
"Netherlands" in data.index

## False
## True
```

Das **in** hat die Bedeutung eines Elementzeichens $\in$ für Mengen.

Mit dem Attribut **.columns** erhalten wir die Spaltennamen. (zu **R**)

```
data.columns

## Index(['Average.disposable.income', 'Children.in.poor.homes',
##        'Educational.Deprivation', 'Overcrowding',
##        'Poor.environmental.conditions', 'Average.mean.literacy.score',
##        'Literacy.inequality', 'Youth.NEET.rate', 'Low.birth.weight',
##        'Infant.mortality', 'Breastfeeding.rates',
##        'Vaccination.rates..pertussis.', 'Vaccination.rates.measles.',
##        'Physical.activity', 'Mortality.rates', 'Suicide.rates', 'Smoking',
##        'Drunkenness', 'Teenage.births', 'Bullying', 'Liking.school'],
##       dtype='object')
```

e) (zu **R**)

Aus der Teilaufgabe vorher sehen wir, dass Betrunkenheit als Spalte *Drunkenness* aufgeführt wird.

Um zu ermitteln, in welchen 5 Ländern die meisten Jugendlichen mindestens zweimal betrunken sind, ordnen wir den Datensatz nach **Drunkenness**

```
drunk = data.sort_values(by="Drunkenness", ascending=False)

drunk["Drunkenness"].head()

## Denmark           24.8
## Finland           22.4
## United Kingdom    22.1
## Poland            19.9
## Canada            18.8
## Name: Drunkenness, dtype: float64
```

Hier wurde mit

```
drunk = data.sort_values(by="Drunkenness", ascending=False)
```

eine neue Tabelle erzeugt. Diese ist nach **`Drunkenness`** absteigend (**`ascending = False`**) geordnet und wird dem Namen **`drunk`** zugeordnet wurde.

```
drunk["Drunkenness"]
```

gibt die Spalte **`Drunkenness`** aus. Mit **`.head()`** wird diese Ausgabe auf die ersten 5 Zeilen beschränkt.

In Dänemark sind die meisten Jugendlichen mindestens zweimal betrunken, nämlich

```
data.loc["Denmark","Drunkenness"]

## 24.8
```

Prozent der dänischen Jugendlichen.

f) (zu **`R`**)

Die Spalte, in der der Wert mit der kleinsten Säuglingssterblichkeit steht, lautet *Infant.mortality*

```
infant = data.nsmallest(n=1, columns = "Infant.mortality")

infant.index

## Index(['Iceland'], dtype='object')
```

Hier wurde mit

```
infant = data.nsmallest(n=1, "Infant.mortality")
```

eine neue Tabelle erzeugt. Diese ist nach **`Infant.Mortality`** aufsteigend (**`ascending = True`**) geordnet und enthält nur eine Zeile. Sie wird dem Namen **`infant`** zugeordnet.

g) (zu **R**)

Der Mittelwert der Anzahl an Jugendlichen, die sich regelmässig bewegen, lautet

```
data["Physical.activity"].mean()

## 20.13461538461539
```

Also in folgenden Ländern ist die Anzahl an Jugendlichen, die sich regelmässig bewegen, kleiner als im OECD Durchschnitt (zu **R**)

```
mean_phys = data["Physical.activity"].mean()

data.loc[data["Physical.activity"] < mean_phys,:].index

## Index(['Austria', 'Belgium', 'France', 'Germany', 'Greece', 'Hungary', 'Italy',
##        'Luxembourg', 'Mexico', 'Norway', 'Poland', 'Portugal', 'Sweden',
##        'Switzerland', 'Turkey', 'United Kingdom'],
##       dtype='object')
```

Der Befehl

```
data.loc[data["Physical.activity"] < mean_phys,:]
```

erzeugt eine neue Tabelle, wo nur die Zeilen aufgeführt sind, bei denen der Wert in **`Physical.activity`** kleiner als der OECD-Durchschnitt ist. Das Attribut **`.index`** gibt dann den Index dieser Tabelle aus.

Lösung 1.2

a) (zu **R**)

Siehe Aufgabenstellung.

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

fuel =  pd.read_table("*d.fuel.dat",sep=",",index_col=0)
```

Um die Daten in Tabellenform zu sehen, tippt man den Namen des Objektes ein

```
fuel

##     weight  mpg     type
## X
## 1     2560   33    Small
## 2     2345   33    Small
## 3     1845   37    Small
## 4     2260   32    Small
## 5     2440   32    Small
## 6     2285   26    Small
```

```
## 7     2275   33    Small
## 8     2350   28    Small
## 9     2295   25    Small
## 10    1900   34    Small
## 11    2390   29    Small
## 12    2075   35    Small
## 13    2330   26    Small
## 14    3320   20   Sporty
## 15    2885   27   Sporty
## 16    3310   19   Sporty
## 17    2695   30   Sporty
## 18    2170   33   Sporty
## 19    2710   27   Sporty
## 20    2775   24   Sporty
## 21    2840   26   Sporty
## 22    2485   28   Sporty
## 23    2670   27  Compact
## 24    2640   23  Compact
## 25    2655   26  Compact
## 26    3065   25  Compact
## 27    2750   24  Compact
## 28    2920   26  Compact
## 29    2780   24  Compact
## 30    2745   25  Compact
## 31    3110   21  Compact
## 32    2920   21  Compact
## 33    2645   23  Compact
## 34    2575   24  Compact
## 35    2935   23  Compact
## 36    2920   27  Compact
## 37    2985   23  Compact
## 38    3265   20   Medium
## 39    2880   21   Medium
## 40    2975   22   Medium
## 41    3450   22   Medium
## 42    3145   22   Medium
## 43    3190   22   Medium
## 44    3610   23   Medium
## 45    2885   23   Medium
## 46    3480   21   Medium
## 47    3200   22   Medium
## 48    2765   21   Medium
## 49    3220   21   Medium
## 50    3480   23   Medium
```

```
## 51    3325   23    Large
## 52    3855   18    Large
## 53    3850   20    Large
## 54    3195   18      Van
## 55    3735   18      Van
## 56    3665   18      Van
## 57    3735   19      Van
## 58    3415   20      Van
## 59    3185   20      Van
## 60    3690   19      Van
```

b) (zu **R**)

Auswählen der fünften Beobachtung:

```
fuel.loc[5,:]

## weight     2440
## mpg          32
## type      Small
## Name: 5, dtype: object
```

c) (zu **R**)

Auswählen der 1. bis 5. Beobachtung:

```
fuel.loc[1:5,:]

##    weight  mpg   type
## X
## 1    2560   33  Small
## 2    2345   33  Small
## 3    1845   37  Small
## 4    2260   32  Small
## 5    2440   32  Small
```

Alternativ kann man sich eine Übersicht verschaffen mit Hilfe des Attributes **.head()**

```
fuel.head()

##    weight  mpg   type
## X
## 1    2560   33  Small
## 2    2345   33  Small
## 3    1845   37  Small
## 4    2260   32  Small
## 5    2440   32  Small
```

d) (zu **R**)

Die Werte der Reichweiten stehen in der dritten Spalte, die **mpg** heisst. Zur Berechnung des Mittelwertes gibt es verschiedene Möglichkeiten, welche sich in der Art der Datenselektion unterscheiden:

```
fuel["mpg"].mean()

## 24.583333333333332
```

e) (zu **R**)

Auch hier gibt es wieder verschiedene Möglichkeiten. Eine davon ist:

```
fuel.loc[7:22,"mpg"].mean()

## 27.75
```

f) (zu **R**)

Umrechnung der Miles Per Gallon in Kilometer pro Liter und der Pounds in Kilogramm:

```
t_kml = fuel["mpg"]*1.6093/3.789

t_kml

## X
## 1     14.016073
## 2     14.016073
## 3     15.714991
## 4     13.591343
## 5     13.591343
## 6     11.042966
## 7     14.016073
## 8     11.892425
## 9     10.618237
## 10    14.440802
## 11    12.317155
## 12    14.865532
## 13    11.042966
## 14     8.494590
## 15    11.467696
## 16     8.069860
## 17    12.741884
## 18    14.016073
## 19    11.467696
## 20    10.193508
## 21    11.042966
```

```
## 22    11.892425
## 23    11.467696
## 24     9.768778
## 25    11.042966
## 26    10.618237
## 27    10.193508
## 28    11.042966
## 29    10.193508
## 30    10.618237
## 31     8.919319
## 32     8.919319
## 33     9.768778
## 34    10.193508
## 35     9.768778
## 36    11.467696
## 37     9.768778
## 38     8.494590
## 39     8.919319
## 40     9.344049
## 41     9.344049
## 42     9.344049
## 43     9.344049
## 44     9.768778
## 45     9.768778
## 46     8.919319
## 47     9.344049
## 48     8.919319
## 49     8.919319
## 50     9.768778
## 51     9.768778
## 52     7.645131
## 53     8.494590
## 54     7.645131
## 55     7.645131
## 56     7.645131
## 57     8.069860
## 58     8.494590
## 59     8.494590
## 60     8.069860
## Name: mpg, dtype: float64
```

```
t_kg = fuel["weight"]*0.45359

t_kg
```

```
## X
## 1     1161.19040
## 2     1063.66855
## 3      836.87355
## 4     1025.11340
## 5     1106.75960
## 6     1036.45315
## 7     1031.91725
## 8     1065.93650
## 9     1040.98905
## 10     861.82100
## 11    1084.08010
## 12     941.19925
## 13    1056.86470
## 14    1505.91880
## 15    1308.60715
## 16    1501.38290
## 17    1222.42505
## 18     984.29030
## 19    1229.22890
## 20    1258.71225
## 21    1288.19560
## 22    1127.17115
## 23    1211.08530
## 24    1197.47760
## 25    1204.28145
## 26    1390.25335
## 27    1247.37250
## 28    1324.48280
## 29    1260.98020
## 30    1245.10455
## 31    1410.66490
## 32    1324.48280
## 33    1199.74555
## 34    1167.99425
## 35    1331.28665
## 36    1324.48280
## 37    1353.96615
## 38    1480.97135
## 39    1306.33920
## 40    1349.43025
## 41    1564.88550
## 42    1426.54055
## 43    1446.95210
```

```
## 44    1637.45990
## 45    1308.60715
## 46    1578.49320
## 47    1451.48800
## 48    1254.17635
## 49    1460.55980
## 50    1578.49320
## 51    1508.18675
## 52    1748.58945
## 53    1746.32150
## 54    1449.22005
## 55    1694.15865
## 56    1662.40735
## 57    1694.15865
## 58    1549.00985
## 59    1444.68415
## 60    1673.74710
## Name: weight, dtype: float64
```

Lösung 1.3

Erzeugen eines Vektors `x` aus den Daten: (zu **R**)

```
from pandas import Series
import pandas as pd
import numpy as np

x  = Series([2.1, 2.4, 2.8, 3.1, 4.2, 4.9, 5.1, 6.0, 6.4, 7.3,
10.8, 12.5, 13.0, 13.7, 14.8, 17.6, 19.6, 23.0, 25.0, 35.2, 39.6])
```

a) (zu **R**)

$\sum x_i =$

```
x.sum()

## 269.1
```

(zu **R**)

$\sum x_i^2 =$

```
(x*x).sum()

## 5729.27
```

b) (zu **R**)

Mittelwert: $\frac{1}{n}\sum x_i =$

```
n = x.size
mean_x = 1/n * x.sum()
mean_x

## 12.814285714285715
```

(zu **R**)

Varianz:

```
var_x = 1/(n-1) * ((x-mean_x)**2).sum()
var_x

## 114.04728571428574
```

(zu **R**)

Standardabweichung

```
np.sqrt(var_x)

## 10.679292378911898
```

c) (zu **R**)

Median:

```
x_sorted = x.sort_values()
n = x.size
k = np.round(0.5*n+0.5)-1
x_sorted[k]

## 10.8
```

Nach der Rundung muss noch 1 subtrahiert werden, da der Index der Series `x_sorted` bei 0 beginnt.

d) (zu **R**)

Mit den **pandas**-Befehlen erhalten wir für die obigen Resultate:

```
x.mean()

## 12.814285714285715
```

```
x.std()

## 10.679292378911898
```

```
x.median()

## 10.8
```

e) (zu R)

Wir stellen aufgrund unseres Beispieldatensatzes für den arithmetischen Mittelwert und die Standardabweichung fest, dass

```
z = (x-x.mean())/x.std()

z.mean()

## -2.114710523095536e-17
```

was gerundet 0 ergibt und

```
z.std()

## 1.0
```

Im Allgemeinen ergibt sich für den arithmetischen Mittelwert von z_i

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n} z_i &= \frac{1}{n}\sum_{i=1}^{n}\frac{x_i-\overline{x}}{s_x} \\
&= \frac{1}{n\cdot s_x}\sum_{i=1}^{n}(x_i-\overline{x}) \\
&= \frac{1}{n\cdot s_x}\sum_{i=1}^{n}\left(\frac{n}{n}\cdot x_i-\overline{x}\right) \\
&= \frac{1}{n\cdot s_x}\left(\frac{n}{n}\cdot\sum_{i=1}^{n}x_i-\sum_{i=1}^{n}\overline{x}\right) \\
&= \frac{1}{n\cdot s_x}(n\overline{x}-n\overline{x}) \\
&= 0\,.
\end{aligned}$$

Für die Varianz von z_i erhalten wir

$$\begin{aligned}
\frac{1}{n-1}\sum_{i=1}^{n}(z_i-\overline{z})^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(z_i-0)^2 \\
&= \frac{1}{(n-1)}\sum_{i=1}^{n}\frac{(x_i-\overline{x})^2}{s_x^2} \\
&= \frac{1}{s_x^2}\frac{1}{(n-1)}\sum_{i=1}^{n}(x_i-\overline{x})^2
\end{aligned}$$

$$= \frac{1}{s_x^2} s_x^2$$
$$= 1 .$$

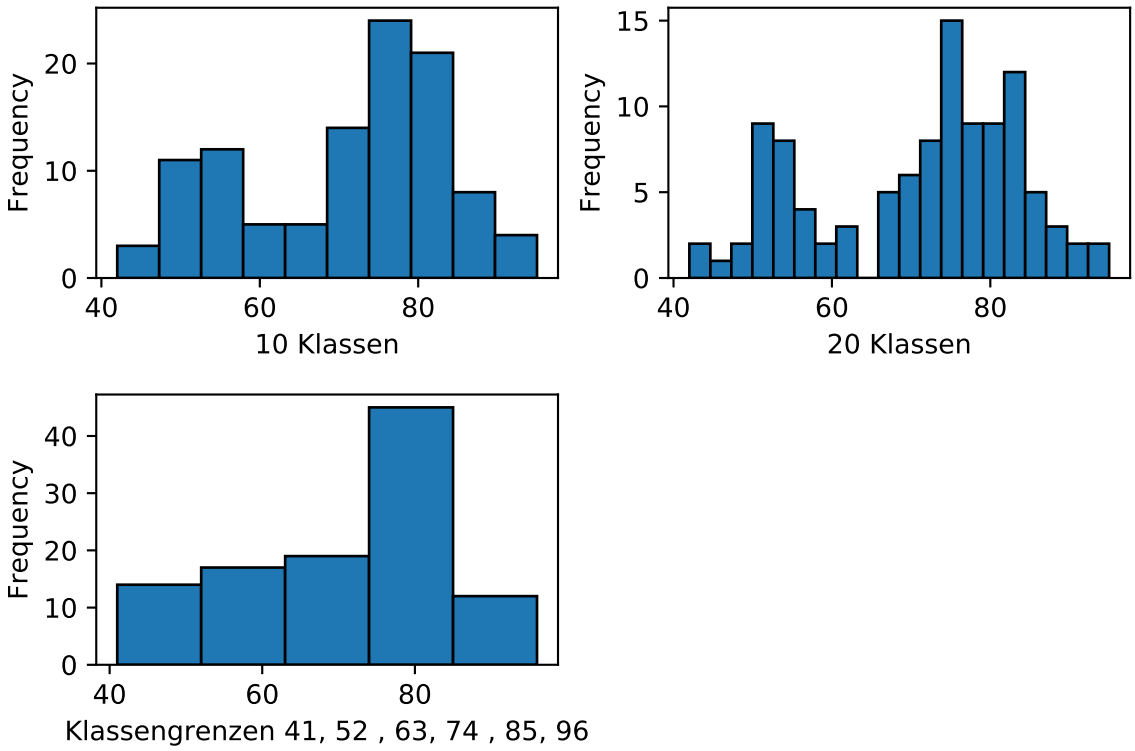
Lösung 1.4

1b, 2c, 3a

Lösung 1.5

a) (zu R)

Die Plots sehen wie folgt aus:

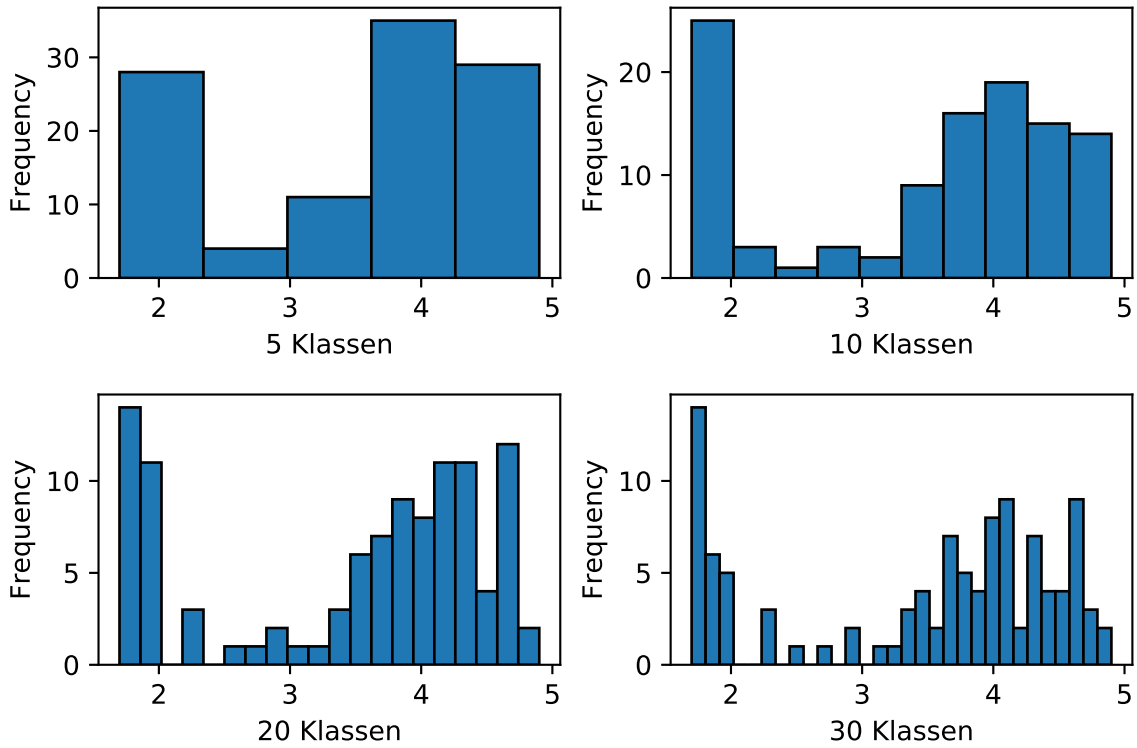


Die drei Histogramme in der Abbildung zeigen die Intervalle zwischen zwei Ausbrüchen von Old Faithful. Auffallend ist, dass Zeitspannen um 55 Minuten aber auch zwischen 70 und 85 Minuten häufiger vorkommen als andere Intervalle. So eine Verteilung mit zwei Gipfeln heisst auch *bimodal*.

Werden die Klassenbreiten ungeschickt gewählt, entdeckt man diese Besonderheit der Geysirdaten nicht. Das ist im dritten Histogramm passiert. Das Beispiel illustriert, dass die richtige Wahl der Klassenbreiten- bzw. grenzen wohlüberlegt sein muss.

b) (zu R)

Die Histogramme mit 5, 10, 20 und 30 Klassen.



Diese vier Histogramm (mit 5, 10, 20, 30 Balken) schliesslich zeigen die Häufigkeiten verschiedener Eruptionsdauern. Hier sind die beiden Gipfel sehr deutlich erkennbar: „Entweder ist der Ausbruch sofort wieder vorbei, oder er dauert mindestens dreieinhalb Minuten".

Ob die Dauer eines Ausbruchs aber etwas zu tun hat mit der Dauer des vorangegangenen Ruheintervalls (mit anderen Worten: ob die Gipfel des Histogramms aus Teilaufgabe b) den Gipfeln der Histogramme aus Teilaufgabe a) entsprechen), kann man aufgrund dieser Darstellungen nicht sagen.

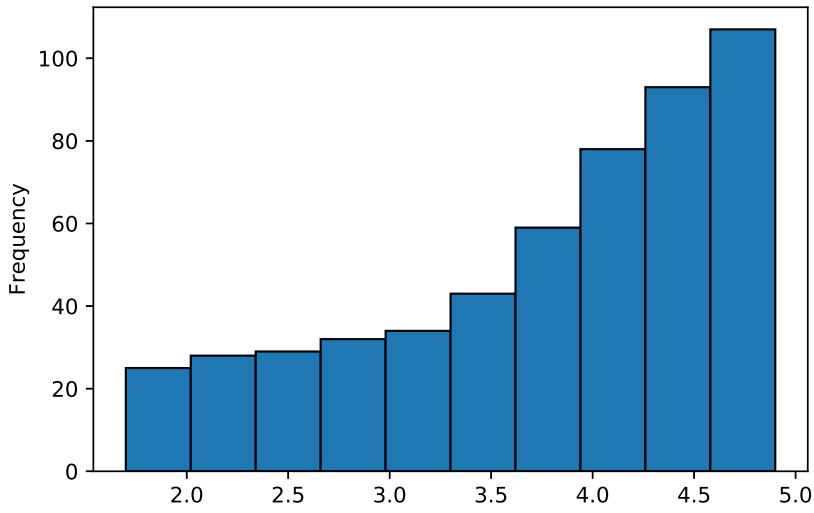
c) (zu **R**)

Die *kumulative Verteilungsfunktion* ist definiert als

$$F_n(x) = \frac{1}{n}\text{Anzahl}\{i \mid x_i \leq x\}.$$

Der Funktionswert von $F_n(x)$ springt also bei jeder Beobachtung $x_i \leq x$ um den Wert $1/n$. Mit **Python** können wir den Funktionsgraphen der kumulativen

Verteilungsfunktion folgendermassen zeichnen:



Aus der Graphik der kumulativen Verteilungsfunktion sehen wir, dass gut 20 % der Eruptionszeiten weniger als 2 Minuten dauern. Rund 60 % der Eruptionen dauern mindestens 3.7 Minuten.

R-Code

Aufgabe 1.1

a) (zu **Python**)

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

b) (zu **Python**)

```
dim(data)

## [1] 30 21
```

c) (zu **Python**)

```
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
```

(zu **Python**)

```
apply(data, 2, mean, na.rm = T)
```

```
##     Average.disposable.income
##                     18.847713
##        Children.in.poor.homes
##                     12.372193
##       Educational.Deprivation
##                      2.673333
##                  Overcrowding
##                     31.950163
## Poor.environmental.conditions
##                     25.217498
##   Average.mean.literacy.score
##                    496.317000
##           Literacy.inequality
##                      1.665085
##               Youth.NEET.rate
##                      7.377778
##             Low.birth.weight
##                      6.643333
##             Infant.mortality
##                      5.446667
##           Breastfeeding.rates
##                     86.027586
## Vaccination.rates..pertussis.
```

```
##                      93.775862
##    Vaccination.rates.measles.
##                      91.517241
##             Physical.activity
##                      20.134615
##               Mortality.rates
##                      24.598966
##                 Suicide.rates
##                       6.856272
##                        Smoking
##                      16.512500
##                   Drunkenness
##                      15.225000
##                Teenage.births
##                      15.500000
##                       Bullying
##                      10.979167
##                 Liking.school
##                      27.172000
```

(zu **Python**)

```
apply(data, 2, median, na.rm = T)
```

```
##     Average.disposable.income
##                      21.107187
##        Children.in.poor.homes
##                      11.659053
##       Educational.Deprivation
##                       1.500000
##                   Overcrowding
##                      21.574977
## Poor.environmental.conditions
##                      25.487116
##   Average.mean.literacy.score
##                     501.335000
##           Literacy.inequality
##                       1.682739
##               Youth.NEET.rate
##                       6.200000
##             Low.birth.weight
##                       6.750000
##             Infant.mortality
##                       4.200000
##           Breastfeeding.rates
```

```
##                      91.000000
## Vaccination.rates..pertussis.
##                      95.800000
##    Vaccination.rates.measles.
##                      94.000000
##             Physical.activity
##                      19.300000
##               Mortality.rates
##                      23.150000
##                 Suicide.rates
##                       6.784772
##                        Smoking
##                      16.600000
##                   Drunkenness
##                      14.550000
##                Teenage.births
##                      10.600000
##                       Bullying
##                       9.650000
##                 Liking.school
##                      25.600000
```

d) (zu **Python**)

```
rownames(data)
```

```
##  [1] "Australia"       "Austria"
##  [3] "Belgium"         "Canada"
##  [5] "Czech Republic"  "Denmark"
##  [7] "Finland"         "France"
##  [9] "Germany"         "Greece"
## [11] "Hungary"         "Iceland"
## [13] "Ireland"         "Italy\t"
## [15] "Japan\t"         "Korea\t"
## [17] "Luxembourg"      "Mexico"
## [19] "Netherlands"     "New Zealand"
## [21] "Norway"          "Poland"
## [23] "Portugal"        "Slovak Republic"
## [25] "Spain\t"         "Sweden"
## [27] "Switzerland"     "Turkey"
## [29] "United Kingdom"  "United States\t"
```

(zu **Python**)

```
is.element("China", rownames(data))
```

```
## [1] FALSE

is.element("Netherlands", rownames(data))

## [1] TRUE
```

(zu **Python**)

```
colnames(data)

##  [1] "Average.disposable.income"
##  [2] "Children.in.poor.homes"
##  [3] "Educational.Deprivation"
##  [4] "Overcrowding"
##  [5] "Poor.environmental.conditions"
##  [6] "Average.mean.literacy.score"
##  [7] "Literacy.inequality"
##  [8] "Youth.NEET.rate"
##  [9] "Low.birth.weight"
## [10] "Infant.mortality"
## [11] "Breastfeeding.rates"
## [12] "Vaccination.rates..pertussis."
## [13] "Vaccination.rates.measles."
## [14] "Physical.activity"
## [15] "Mortality.rates"
## [16] "Suicide.rates"
## [17] "Smoking"
## [18] "Drunkenness"
## [19] "Teenage.births"
## [20] "Bullying"
## [21] "Liking.school"
```

e) (zu **Python**)

Um zu ermitteln, in welchen 5 Ländern die meisten Jugendlichen mindestens zweimal betrunken sind, benützen wir

```
order(...)
```

um die Zeilen in aufsteigender Reihenfolge zu ermitteln

```
order(data[, "Drunkenness"], na.last = F)[26:30]

## [1]  4 22 29  7  6
```

Dabei stellt das Argument `na.last=F` die Zeilen mit den nicht vorhandenen Datenpunkten am Anfang des geordneten Datenvektors. Die Ländernamen lauten dann

```
rownames(data[order(data[, "Drunkenness"], na.last = F)[26:30],
    ])
```

```
## [1] "Canada"         "Poland"
## [3] "United Kingdom" "Finland"
## [5] "Denmark"
```

In Dänemark sind die meisten Jugendlichen mindestens zweimal betrunken, nämlich

```
data["Denmark", "Drunkenness"]
```

```
## [1] 24.8
```

Prozent der dänischen Jugendlichen.

f) (zu **Python**)

Die Zeile, in der der Wert mit der kleinsten Säuglingssterblichkeit steht, lautet

```
which.min(data[, "Infant.mortality"])
```

```
## [1] 12
```

Das betreffende Land ist

```
rownames(data[which.min(data[, "Infant.mortality"]),
    ])
```

```
## [1] "Iceland"
```

g) (zu **Python**)

```
mean(data[, "Physical.activity"], na.rm = T)
```

```
## [1] 20.13462
```

(zu **Python**)

```
mean.physical.activity <- mean(data[, "Physical.activity"],
    na.rm = T)
which(data[, "Physical.activity"] < mean.physical.activity)
```

```
##  [1]  2  3  8  9 10 11 14 17 18 21 22 23 26 27 28
## [16] 29
```

```
rownames(data[which(data[, "Physical.activity"] <
    mean.physical.activity), ])
```

```
##  [1] "Austria"        "Belgium"
##  [3] "France"         "Germany"
##  [5] "Greece"         "Hungary"
##  [7] "Italy\t"        "Luxembourg"
##  [9] "Mexico"         "Norway"
## [11] "Poland"         "Portugal"
## [13] "Sweden"         "Switzerland"
## [15] "Turkey"         "United Kingdom"
```

Aufgabe 1.2

a) (zu **Python**)

Siehe Aufgabenstellung.

```
d.fuel <- read.table("./Daten/d.fuel.dat", header = T,
    sep = ",")
```

Um die Daten in Tabellenform zu sehen, tippt man den Namen des Objektes ein

```
d.fuel
```

```
##     X weight mpg    type
## 1   1   2560  33   Small
## 2   2   2345  33   Small
## 3   3   1845  37   Small
## 4   4   2260  32   Small
## 5   5   2440  32   Small
## 6   6   2285  26   Small
## 7   7   2275  33   Small
## 8   8   2350  28   Small
## 9   9   2295  25   Small
## 10 10   1900  34   Small
## 11 11   2390  29   Small
## 12 12   2075  35   Small
## 13 13   2330  26   Small
## 14 14   3320  20  Sporty
## 15 15   2885  27  Sporty
## 16 16   3310  19  Sporty
## 17 17   2695  30  Sporty
## 18 18   2170  33  Sporty
## 19 19   2710  27  Sporty
## 20 20   2775  24  Sporty
```

```
## 21 21   2840  26  Sporty
## 22 22   2485  28  Sporty
## 23 23   2670  27 Compact
## 24 24   2640  23 Compact
## 25 25   2655  26 Compact
## 26 26   3065  25 Compact
## 27 27   2750  24 Compact
## 28 28   2920  26 Compact
## 29 29   2780  24 Compact
## 30 30   2745  25 Compact
## 31 31   3110  21 Compact
## 32 32   2920  21 Compact
## 33 33   2645  23 Compact
## 34 34   2575  24 Compact
## 35 35   2935  23 Compact
## 36 36   2920  27 Compact
## 37 37   2985  23 Compact
## 38 38   3265  20  Medium
## 39 39   2880  21  Medium
## 40 40   2975  22  Medium
## 41 41   3450  22  Medium
## 42 42   3145  22  Medium
## 43 43   3190  22  Medium
## 44 44   3610  23  Medium
## 45 45   2885  23  Medium
## 46 46   3480  21  Medium
## 47 47   3200  22  Medium
## 48 48   2765  21  Medium
## 49 49   3220  21  Medium
## 50 50   3480  23  Medium
## 51 51   3325  23   Large
## 52 52   3855  18   Large
## 53 53   3850  20   Large
## 54 54   3195  18     Van
## 55 55   3735  18     Van
## 56 56   3665  18     Van
## 57 57   3735  19     Van
## 58 58   3415  20     Van
## 59 59   3185  20     Van
## 60 60   3690  19     Van
```

b) (zu **Python**)

Auswählen der fünften Beobachtung:

```
d.fuel[5, ]

##   X weight mpg  type
## 5 5   2440  32 Small
```

c) (zu **Python**)

Auswählen der 1. bis 5. Beobachtung:

```
d.fuel[1:5, ]

##   X weight mpg  type
## 1 1   2560  33 Small
## 2 2   2345  33 Small
## 3 3   1845  37 Small
## 4 4   2260  32 Small
## 5 5   2440  32 Small
```

Alternativ kann man sich eine Übersicht verschaffen mit Hilfe der R-Funktion `head(...)`

```
head(d.fuel)

##   X weight mpg  type
## 1 1   2560  33 Small
## 2 2   2345  33 Small
## 3 3   1845  37 Small
## 4 4   2260  32 Small
## 5 5   2440  32 Small
## 6 6   2285  26 Small
```

d) (zu **Python**)

Die Werte der Reichweiten stehen in der dritten Spalte, die `mpg` heisst. Zur Berechnung des Mittelwertes gibt es verschiedene Möglichkeiten, welche sich in der Art der Datenselektion unterscheiden:

```
mean(d.fuel[, 3])

## [1] 24.58333

mean(d.fuel[, "mpg"])

## [1] 24.58333

mean(d.fuel$mpg)

## [1] 24.58333
```

e) (zu **Python**)

Auch hier gibt es wieder verschiedene Möglichkeiten. Eine davon ist:

```
mean(d.fuel[7:22, "mpg"])

## [1] 27.75
```

f) (zu **Python**)

Umrechnung der Miles Per Gallon in Kilometer pro Liter und der Pounds in Kilogramm:

```
t_kml <- d.fuel[, "mpg"] * 1.6093/3.789
t_kml

##  [1] 14.016073 14.016073 15.714991 13.591343
##  [5] 13.591343 11.042966 14.016073 11.892425
##  [9] 10.618237 14.440802 12.317155 14.865532
## [13] 11.042966  8.494590 11.467696  8.069860
## [17] 12.741884 14.016073 11.467696 10.193508
## [21] 11.042966 11.892425 11.467696  9.768778
## [25] 11.042966 10.618237 10.193508 11.042966
## [29] 10.193508 10.618237  8.919319  8.919319
## [33]  9.768778 10.193508  9.768778 11.467696
## [37]  9.768778  8.494590  8.919319  9.344049
## [41]  9.344049  9.344049  9.344049  9.768778
## [45]  9.768778  8.919319  9.344049  8.919319
## [49]  8.919319  9.768778  9.768778  7.645131
## [53]  8.494590  7.645131  7.645131  7.645131
## [57]  8.069860  8.494590  8.494590  8.069860

t_kg <- d.fuel[, "weight"] * 0.45359
t_kg

##  [1] 1161.1904 1063.6686  836.8736 1025.1134
##  [5] 1106.7596 1036.4532 1031.9172 1065.9365
##  [9] 1040.9890  861.8210 1084.0801  941.1993
## [13] 1056.8647 1505.9188 1308.6072 1501.3829
## [17] 1222.4251  984.2903 1229.2289 1258.7123
## [21] 1288.1956 1127.1711 1211.0853 1197.4776
## [25] 1204.2814 1390.2533 1247.3725 1324.4828
## [29] 1260.9802 1245.1046 1410.6649 1324.4828
## [33] 1199.7456 1167.9942 1331.2867 1324.4828
## [37] 1353.9661 1480.9714 1306.3392 1349.4302
## [41] 1564.8855 1426.5405 1446.9521 1637.4599
## [45] 1308.6072 1578.4932 1451.4880 1254.1763
## [49] 1460.5598 1578.4932 1508.1868 1748.5894
```

```
## [53] 1746.3215 1449.2200 1694.1587 1662.4073
## [57] 1694.1587 1549.0098 1444.6842 1673.7471
```

Aufgabe 1.3

(zu **Python**)

```
x <- c(2.1, 2.4, 2.8, 3.1, 4.2, 4.9, 5.1, 6, 6.4, 7.3, 10.8,
    12.5, 13, 13.7, 14.8, 17.6, 19.6, 23, 25, 35.2, 39.6)
```

a) (zu **Python**)

```
sum(x)

## [1] 269.1
```

(zu **Python**)

```
sum(x^2)

## [1] 5729.27
```

b) (zu **Python**)

```
n <- length(x)
mean.x <- 1/n * sum(x)
mean.x

## [1] 12.81429
```

(zu **Python**)

```
var.x <- 1/(n - 1) * sum((x - mean.x)^2)
var.x

## [1] 114.0473
```

(zu **Python**)

```
sqrt(var.x)

## [1] 10.67929
```

c) (zu **Python**)

```r
x.sorted <- sort(x)
0.5 * length(x)

## [1] 10.5
```

```r
k <- round(0.5 * length(x) + 0.5)
k

## [1] 11

x.sorted[k]

## [1] 10.8
```

d) (zu **Python**)

```r
mean(x)

## [1] 12.81429

sd(x)

## [1] 10.67929

median(x)

## [1] 10.8
```

e) (zu **Python**)

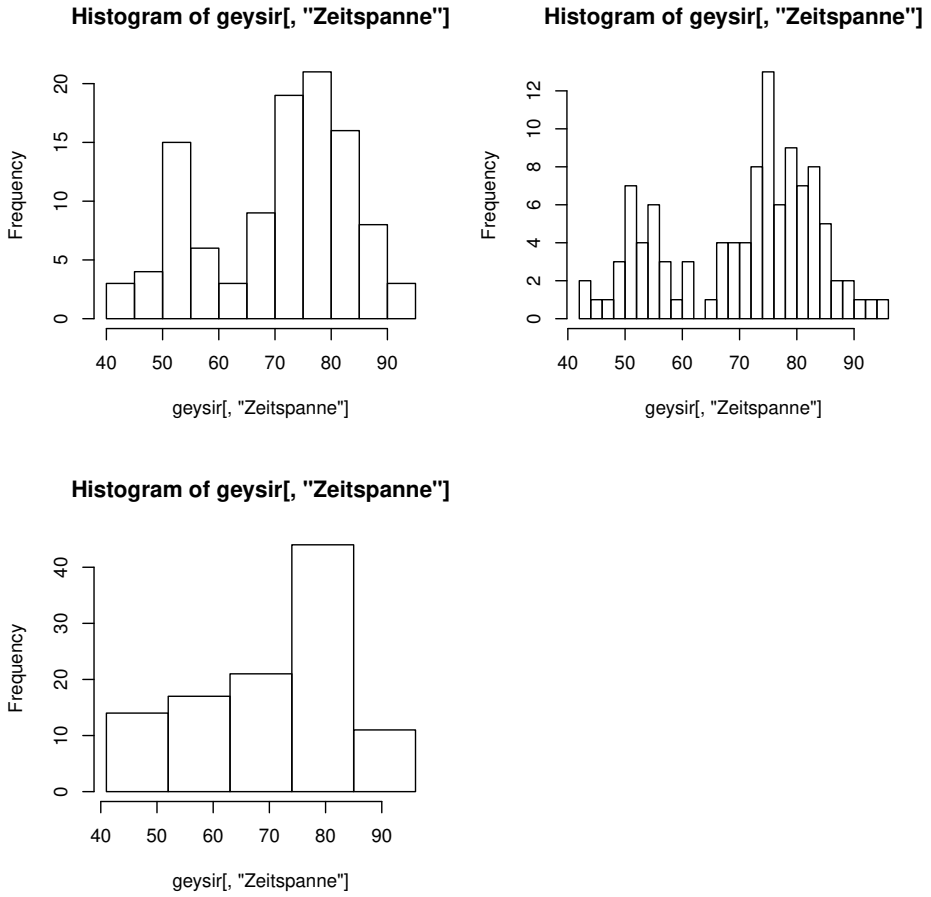
```r
z <- (x - mean(x))/sd(x)
mean(z)

## [1] -2.614476e-17

sd(z)

## [1] 1
```

Aufgabe 1.5

a) (zu **Python**)

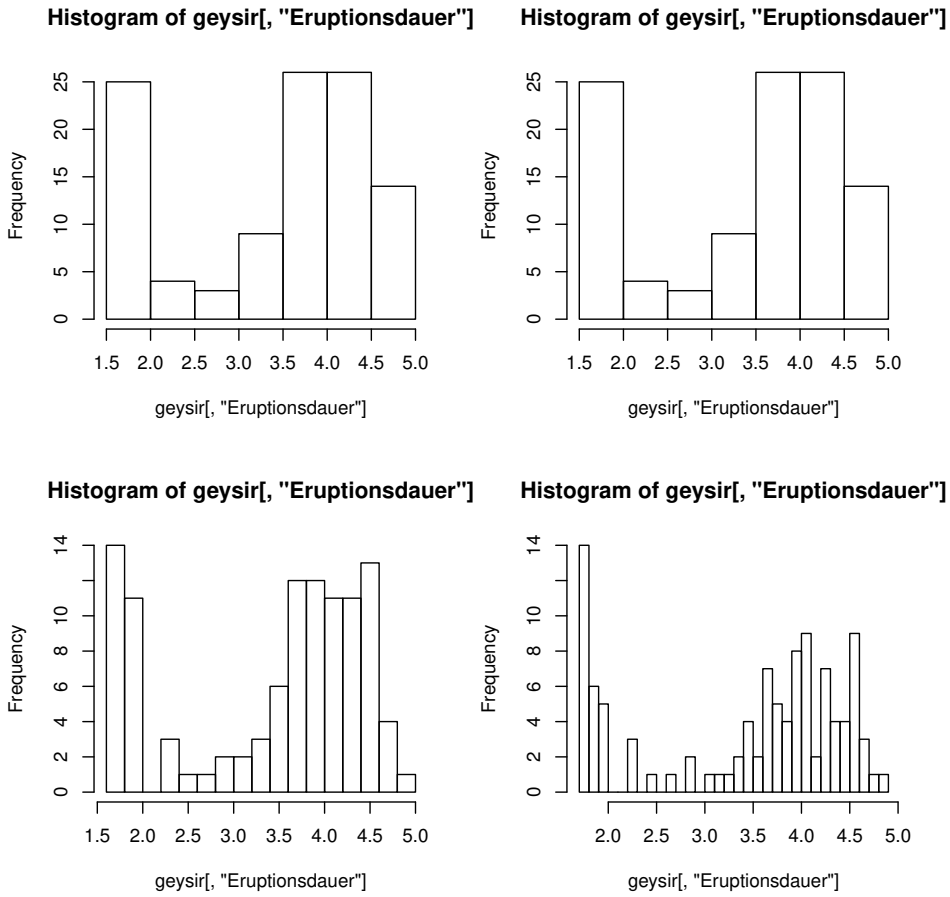
```r
# Datensatz einlesen
geysir <- read.table("./Daten/geysir.dat", header = TRUE)
par(mfrow = c(2, 2))  # 4 Grafiken im Grafikfenster
# Histogramme zeichnen
hist(geysir[, "Zeitspanne"])
```

```r
hist(geysir[, "Zeitspanne"], breaks = 20)
hist(geysir[, "Zeitspanne"], breaks = seq(41, 96, by = 11))
```



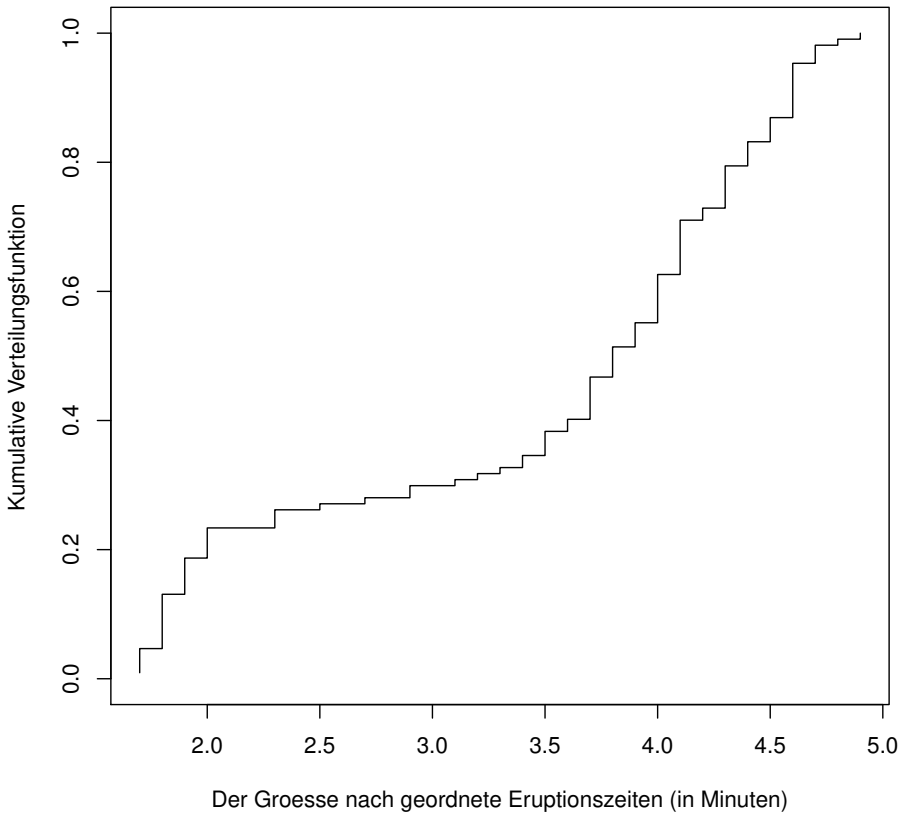
b) (zu **Python**)

```r
# Datensatz einlesen
geysir <- read.table("./Daten/geysir.dat", header = TRUE)
par(mfrow = c(2, 2))  # 4 Grafiken im Grafikfenster
# Histogramme zeichnen
hist(geysir[, "Eruptionsdauer"], breaks = 5)
hist(geysir[, "Eruptionsdauer"], breaks = 10)
hist(geysir[, "Eruptionsdauer"], breaks = 20)
hist(geysir[, "Eruptionsdauer"], breaks = 30)
```



c) (zu **Python**)

```
eruptionsdauern <- geysir[, "Eruptionsdauer"]
n <- length(eruptionsdauern)
plot(sort(eruptionsdauern), (1:n)/n,
    type = "s", ylim = c(0, 1),
    ylab = "Kumulative Verteilungsfunktion",
    xlab = "Der Groesse nach geordnete Eruptionszeiten (in Minuten)",
    main = "")
```


Kumulative Verteilungsfunktion
0.0
0.2
0.4
0.6
0.8
1.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Der Groesse nach geordnete Eruptionszeiten (in Minuten)