Serie 1

Aufgabe 1.1

In dieser Aufgabe werden einige einfache `Python`-Befehle besprochen. Schauen Sie im Zweifelsfall im Dokument `Intro_python.pdf` auf ILIAS nach.

a) Bilden Sie einen Vektor `x` mit den Zahlen $4, 2, 1, 3, 35, 7$.

b) Wählen Sie mit `Python` den dritten Wert aus.

c) Wählen Sie mit `Python` den ersten und vierten Wert aus.

d) Bestimmen Sie die Länge des Vektors `x`.

e) Was macht der Befehl `x+2`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

f) Was macht der Befehl `np.sum(x+2)`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

g) Was macht der Befehl `x <= 3`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

h) Was macht der Befehl `x[x <= 3]`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

i) Was macht der Befehl `np.sort(x)`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

j) Was macht der Befehl `np.argsort(x)`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus. Vergleichen Sie dabei die Werte von `np.argsort(x)+1` mit den Werten von `x`.

k) Sie möchten den Wert des 4. Eintrages durch die Zahl 8 ersetzen. Wie machen Sie das?

Aufgabe 1.2

Gegeben sind folgende Temperaturen in Grad Fahrenheit (°F)

$$51.9, \quad 51.8, \quad 51.9, \quad 53$$

a) Bilden Sie einen Vektor `fahrenheit` mit diesen Werten.

b) Berechnen Sie diese Temperaturen in Grad Celsius (°C) um. Die Umrechnungsformel lautet

$$C = \frac{5}{9}(F - 32)$$

Bilden Sie dazu einen Vektor **`celsius`**.

c) Gegeben sind weitere Temperaturen

$$48, \quad 48.2, \quad 48, \quad 48.7$$

Bestimmen Sie die Differenz zu den ursprünglichen Temperaturen. Benützen Sie wieder Vektoren.

Aufgabe 1.3

Wir haben von 6 Personen die Körpergrösse (kg)

$$60,\ 72,\ 57,\ 90,\ 95,\ 72$$

und das Körpergewicht (in m)

$$1.75,\ 1.80,\ 1.65,\ 1.90,\ 1.74,\ 1.91$$

gegeben.

Nun wollen wir den Body Mass Index (BMI) berechnen. Dieser berechnet sich wie folgt

$$\text{BMI} = \frac{\text{Gewicht}}{\text{Grösse}^2}$$

a) Erzeugen Sie zwei Vektoren **`weight`** und **`height`**.

b) Berechnen Sie den BMI dieser 6 Personen gleichzeitig. Erzeugen Sie dazu einen Vektor **`bmi`**.

Aufgabe 1.4

Diese Aufgabe befasst sich mit dem Datensatz `weather.csv`, den wir in der Einführung kennengelernt haben.

Schauen Sie im Zweifelsfall im Dokument **`Intro_python.pdf`** auf ILIAS nach.

a) Laden Sie den Datensatz und speichern Sie diesen unter der Variable **`data`** ab.

b) Wählen Sie den Wert der zweiten Zeile und dritten Spalte aus (Feb und Chur).

c) Wählen Sie die 4. Zeile aus (Apr)?

d) Wählen Sie die 1. und die 4. Spalte aus. Verwenden Sie dazu die Spaltennamen.

e) Speichern Sie obige Data unter dem Namen **`data1`** ab und speichern Sie dies unter dem Namen `weather2.csv`.

f) Wie können Sie herausfinden (mit **`Python`** natürlich), welches der Name der 3. Spalte ist?

g) Wir möchten den Spaltennamen **`Basel`** durch **`Genf`** ersetzen. Wie würden Sie vorgehen? Der Befehl **`data.columns.values`** gibt die Spaltennamen aus.

h) Ordnen Sie die Daten nach den Werten von Zurich. Verwenden Sie den Befehl **`data.sort_values()`**.

i) Ordnen Sie die Daten nach den Werten von Zurich aufsteigend. Googeln Sie dazu den entsprechenden Befehl.

Aufgabe 1.5

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) Lesen Sie den Datensatz `child.csv` mit der folgenden Funktion ein:

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

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

Achtung: Für `*` muss der Pfad angegeben werden, wo sich ihr File `child.csv` befindet. Entweder geben Sie den absoluten Pfad an oder den relativen Pfad ab dem Arbeitsverzeichnis, das im `File Explorer` angezeigt wird.

Das Argument `sep=","` definiert das Trennzeichen der Spalten. In einer `csv`-Datei (*c*omma *s*eparated *f*ile) ist dies standardmässig das Komma. Die Option kann in diesem Fall also auch weggelassen werden.

Das Argument `index_col=0` bewirkt, dass die 1. Spalte des Files als Index verwendet wird.

Mögliche Schwierigkeiten beim Einlesen:

- Fehlermeldung `...file not found...` → Falscher Pfad
- Fehlermeldung `...unicodexxxx...` → Leerzeichen in Ordnernamen (vermeiden Sie diese)
- Fehlermeldung: Pcom...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 öffnen und speichern!

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 Zeilenamen 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=...)`**.

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

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

Aufgabe 1.6

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_csv("*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.

Kurzlösungen vereinzelter Aufgaben

Musterlösungen zu Serie 1

Lösung 1.1

a) Ein Vektor wird mit dem Befehl **`np.array(...)`** gebildet.

```
import numpy as np
x = np.array([4, 2, 1, 3, 3, 5,7])
x
print(x)

## [4 2 1 3 3 5 7]
```

b)
```
x[2]

## 1
```

Beachten Sie, dass die Indexierung in **`Python`** bei 0 beginnt.

c)
```
x[[0, 3]]

## [4 3]
```

d)
```
x.size

## 7
```

e)
```
x+2

## [6 4 3 5 5 7 9]
```

Zu jeder Komponente wird 2 addiert.

f)
```
np.sum(x+2)

## 39
```

Hier werden alle Werte in **`x+2`** aufaddiert.

g)
```
x <= 3

## [False  True  True  True  True False False]
```

Der Befehl erzeugt einen Vektor der Länge von **`x`**. Für alle Werte die kleiner oder gleich als 3 sind, wird **`TRUE`**, für die anderen **`FALSE`**.

h)
```
x[x <= 3]

## [2 1 3 3]
```

Die Konstruktion **`x[...]`** wählt Elemente aus dem Vektor **`x`** aus. Die Auswahl geschieht nun mit **`x <= 3`** aus g). Es werden alle Werte ausgewählt, die den Wert **`TRUE`** haben.

i)
```
np.sort(x)

## [1 2 3 3 4 5 7]
```

Die Werte von **`x`** werden der Grösse nach aufsteigend geordnet.

j)
```
x
np.argsort(x)+1

## [4 2 1 3 3 5 7]
## [3 2 4 5 1 6 7]
```

- Der ersten Wert von **`np.argsort(x)+1`** ist 3. Der zugehörige Wert von **`x`** ist 1.
- Der zweite Wert von **`np.argsort(x)+1`** ist 2. Der zugehörige Wert von **`x`** ist 2.
- Der dritte Wert von **`np.argsort(x)+1`** ist 4. Der zugehörige Wert von **`x`** ist 3.
- Der vierte Wert von **`np.argsort(x)+1`** ist 5. Der zugehörige Wert von **`x`** ist 3.
- usw. ...

Der Befehl **`np.argsort(x)+1`** gibt also die Stellen an, *wo* sich die Werte von **`x`** befinden.

k)
```
x[3] = 8
x

## [4 2 1 8 3 5 7]
```

Lösung 1.2

a) Vector **`fahrenheit`**

```
import numpy as np
fahrenheit = np.array([51.9, 51.8, 51.9, 53])

fahrenheit

## [51.9 51.8 51.9 53. ]
```

b) Temperatures in degree celsius (°C)

```
celsius = 5/9 * (fahrenheit - 32)

celsius

## [11.05555556 11.         11.05555556 11.66666667]
```

c) Weitere Temperaturen

```
import numpy as np
fahrenheit = np.array([51.9, 51.8, 51.9, 53])
fahrenheit_2 = np.array([48, 48.2, 48, 48.7])

fahrenheit_3 = fahrenheit - fahrenheit_2

fahrenheit_3
print(fahrenheit_3)

## [3.9 3.6 3.9 4.3]
```

Lösung 1.3

a)
```
import numpy as np

weight = np.array([60, 72, 57, 90, 95, 72])
height = np.array([1.75, 1.80, 1.65, 1.90, 1.74, 1.91])
```

b)
```
bmi = weight / height**2

np.round(bmi, 3)

## [19.592 22.222 20.937 24.931 31.378 19.736]
```

Somit haben wir den BMI für alle 6 Personen gleichzeitig berechnet!

Lösung 1.4

a)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
print(data)

##      Luzern  Basel  Chur  Zurich
## Jan       2      5    -3       4
## Feb       5      6     1       0
## Mar      10     11    13       8
## Apr      16     12    14      17
## May      21     23    21      20
## Jun      25     21    23      27
```

Ihr Pfad wird natürlich anders lauten. Für Windows-User: Sie müssen die \ durch / ersetzen.

b)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
print(data.loc["Feb", "Chur"])

## 1
```

Nochmals: Der erste Wert von **`data[..., ...]`** bezieht sich *immer* auf die Zeile und der zweite Wert auf die Spalte.

c)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
print(data.loc["Apr", :])

## Luzern    16
## Basel     12
## Chur      14
## Zurich    17
## Name: Apr, dtype: int64
```

d)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
data.loc[:, ["Luzern", "Zurich"]]
print(data.loc[:, ["Luzern", "Zurich"]])

##      Luzern  Zurich
## Jan       2       4
## Feb       5       0
## Mar      10       8
## Apr      16      17
## May      21      20
## Jun      25      27
```

e)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
data1 = data.loc[:, ["Luzern", "Zurich"]]
data1.to_csv("../../../Software_R_Python/Python/weather2.csv")

data2 = pd.read_csv("../../../Software_R_Python/Python/weather2.csv")
data2
print(data2)

##   Unnamed: 0  Luzern  Zurich
## 0        Jan       2       4
## 1        Feb       5       0
```

```
## 2        Mar      10       8
## 3        Apr      16      17
## 4        May      21      20
## 5        Jun      25      27
```

f)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
data.columns.values

print(data.columns.values)
data.columns.values[2]
print(data.columns.values[2])

## ['Luzern' 'Basel' 'Chur' 'Zurich']
## Chur
```

Der Befehl **`.columns.values`** erzeugt einen Vektor mit den Spaltennamen der Datei **`data`**. Mit **`...[2]`** wird der dritte Wert ausgewählt.

g)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")

data.columns.values[1] = "Genf"
print(data.columns.values)

## ['Luzern' 'Genf' 'Chur' 'Zurich']
```

h)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
data3 = data.sort_values(by="Zurich")
print(data3)

##      Luzern  Basel  Chur  Zurich
## Feb       5      6     1       0
## Jan       2      5    -3       4
## Mar      10     11    13       8
## Apr      16     12    14      17
## May      21     23    21      20
## Jun      25     21    23      27
```

i)
```
import pandas as pd
data = pd.read_csv("../../../Software_R_Python/R/weather.csv")
data3 = data.sort_values(by="Zurich", ascending = False)
print(data3)

##      Luzern  Basel  Chur  Zurich
## Jun      25     21    23      27
```

```
## May      21      23     21       20
## Apr      16      12     14       17
## Mar      10      11     13        8
## Jan       2       5     -3        4
## Feb       5       6      1        0
```

Lösung 1.5

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()
print(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]
```

Wie Sie feststellen können, kommen noch die Werte **NaN** vor. Dies bedeutet „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]
```

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

```
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
```

(zu R) Entsprechend gilt für den Median

```
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 also im Datensatz enthalten, China hingegen nicht. Wir können dies auch mit folgendem Befehl überprüfen: (zu **R**)

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

```
## China False
## Netherlands True
```

Das Schlüssenwort **in** hat die Bedeutung des 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 in der 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, nach **`Drunkenness`** absteigend sortiert und dem Namen **`drunk`** zugewiesen. Der Befehl

```
drunk["Drunkenness"].head()
```

gibt die ersten 5 Zeilen der Spalte **`Drunkenness`** aus.

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 Säuglingssterblichkeitsrate steht, lautet **`Infant.mortality`**. Den kleinsten Wert in dieser Spalte erhalten wir mit

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

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

g) (zu **`R`**) Der Mittelwert der Anzahl an Jugendlichen, die sich regelmässig bewegen, lautet

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

## 20.134615384615383
```

Also ist in folgenden Ländern 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
```

```
## 20.134615384615383
## 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,:].index
```

erzeugt eine neue Tabelle, die nur die Zeilen enthält, deren Wert von **`Physical.activity`** kleiner als **`mean_phys`** ist. Das Attribut **`.index`** gibt den Index dieser Tabelle aus.

Lösung 1.6

a) (zu **R**)

Siehe Aufgabenstellung.

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

fuel =  pd.read_csv("*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
```

R-Code