Abschlussbeispiel

UK Computational Statistics

Angabe

Schreiben Sie ein R-Programm und ein Python Programm zur Implementation und Demonstration ***des k-means Algorithmus in einem 2-dimensionalen Datenraum***.

Ziel dieses Algorithmus ist es eine vorliegende Stichprobe in k disjunkte Teilmengen aufzuteilen.

Dazu wird folgendes Vorgehensmodell verwendet:

1) Wähle zufällig k Punkte aus der beobachteten Datenmatrix als Startlösung für die Gruppenmittelwerte (Es ist dabei sicherzustellen, dass sich unter den k Punkten keine identischen Beobachtungspaare befinden).

2) Bestimme für jeden Datenpunkt seine Euklid‘schen Distanzen zu den k aktuellen Gruppenmittelwerten und ordne den Datenpunkt zu jener Gruppe zu, für die die Distanz minimiert wird.

3) Bestimme aufgrund der aktuellen Gruppenzugehörigkeit der Datenpunkte für jede der k Gruppen durch Anwendung der Funktion mean neue Gruppenmittelwerte

4) Wiederhole die Schritte 2 und 3 solange bis entweder, die Zahl der maximalen Iterationsschritte erreicht ist, oder aber sich die Summe der Distanzen über alle Punkte zu ihrem nächsten Zentrum nur mehr geringfügig ändert.

Abbruchbedingung: falls die Summe der Distanzen im letzten Schritt mit Si-1 und im aktuellen Schritt mit Si bezeichnet werden, soll gestoppt werden, falls abs(Si-1 – Si)/Si-1 < change, wobei change als Inputparameter festgelegt werden können soll (siehe unten)..

Konkrete Aufgabenstellung:

a) Schreiben Sie ein Funktion mit folgenden ***Inputparametern***:

x … Datenmatrix oder Dataframe der Dimension n x 2 (n Beobachtungen von jeweils 2 Variablen)

k … Anzahl der Teilgruppen, die gebildet werden sollen

trace=F .. falls True soll für jeden Zwischenschritt eine Grafik produziert werden (siehe b)

maxiter=10 … maximale Zahl der Iterationsschritte

change=0.001 … Abbruchswert für die relative Änderung

Die Funktion soll als ***Output-Objekt*** eine Liste mit folgenden Komponenten erzeugen:

iter .. Zahl der durchgeführten Iterationsschritte

Zentren … Matrix der Dimension k x 2, welche für jede Teilmenge die Mittelwerte der beiden Variablen enthält

Index … Vektor der Länge n, welcher für jeden Datenpunkt die Info enthält zu welcher Teilmenge er gehört

Distanz … Vektor der Länge n, welcher für jeden Datenpunkt die Distanz zu dem Zentrum der Teilmenge, zu der er gehört, enthält

b) Falls der Inputparameter trace auf True gesetzt wird, soll pro Iterationsschritt ein Streudiagramm ausgegeben werden, welches die k Gruppenmittelwerte durch Farbmarkierung und Symbolik hervorhebt und die Datenpunkte je nach der Zugehörigkeit zu der Gruppe farblich und symbolisch unterschiedlich darstellt.

c) Überprüfen Sie die Funktionalität mit dem in R enthaltenen Datensatz „faithful".

d) Überprüfen Sie die Funktionalität mit einem simulierten Datensatz:
generieren Sie 4 Stichproben mit je 25 Beobachtungen (insgesamt n=100), und folgenden Mittelwerten (-1,1), (-1,-1), (1,1), (1,-1), die Werte für die beiden Variablen sollen jeweils um den Mittelwert normalverteilt mit Standardabweichung 1 sein.

Optionale Mehrleistungen:

• Steuerung des Programms über eine interaktive Eingabemaske

• Ermittlung der Silhouetten-Werte für jede Beobachtung und der Silhouettenkoeffizienten für jeden Cluster https://de.wikipedia.org/wiki/Silhouettenkoeffizient

• Implementierung eines Silhouetten-Plots

In dieser Dokumentation wird der lauffähige Python-Code für die Lösung des Abschlussbeispiels erklärt und an den jeweiligen Stellen werden die Analyseergebnisse besprochen. Für den gesamten Code zur Ausführung siehe Python-File, das ebenfalls auf Moodle hochgeladen wurde.

Lösung

Im Folgenden wir der Python-Code erklärt und dokumentiert:

Zunächst werden die notwendigen Libraries und Funktionen importiert.

Dann werden die Testdaten, d.h. der *faithful*-Datensatz, aus einem CSV-File mittels Dateipfad eingelesen.

```
# -*- coding: utf-8 -*-
"""
@author: cordula eggerth, dariga ramazanova, lusine yeghiazaryan

abschlussbeispiel: k-means

"""
import pandas as pd
from itertools import repeat
import random as rd
import math
from statistics import mean
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np


# ------------------------------------------------------------------------
# TEST-DATEN AUS CSV-FILE faithful.csv EINLESEN & DATAFRAME ANLEGEN
# ------------------------------------------------------------------------
# daten: eruptions (col1), waiting (col2)
path = "C:/Users/cordu/Desktop/faithful.csv"
daten = pd.read_csv(path, sep=",", decimal=".")

```

Untenstehend wird die k-means-Funktion in kommentierter Version vorgestellt:

```
34
35 # ------------------------------------------------------------------------------
36 # K-MEANS FUNKTION
37 # ------------------------------------------------------------------------------
38
39 '''
40   input-parameter:
41     x ... dataframe (dimension nx2, d.h. n rows, 2 cols)
42     k ... anzahl der zu bildenden teilgruppen
43     trace=FALSE ... falls TRUE, dann soll für jeden zwischenschritt eine grafik produziert werden
44     maxiter=10 ... maximale anzahl von iterationsschritten
45     change=0.001 ... abbruchswert für die relative änderung
46
47   output (list):
48     iter ... anzahl d er durchgeführten iterationsschritte
49     zentren ... matrix der dimension kx2 (enthält für jede teilmenge die mittelwerte der beiden variablen)
50     index ... vektor der länge n (enthält für jeden datenpunkt info, zu welcher teilmenge er gehört)
51     distanz ... vektor der länge n (enthält für jeden datenpunkt die distanz zum zentrum seiner teilmenge)
52 '''
53
54 def kmeans(x=daten, k=3, trace=False, maxiter=10, change=0.001):
55   # OUTPUT INITIALISIERUNG:
56   iter = 0
57   zentren = None
58   index = list(repeat(-1, len(x)))
59   distanz = list(repeat(-1, len(x)))
60   colnames = x.columns.values.tolist()
61   outputliste = list()
62
```

In der Funktion werden die Output-Variablen initialisiert. Diese sind *iter*, *zentren*, *index*, *distanz* und die *outputliste*, wobei in letzterer die Output-Variablen gespeichert werden.

```
# weitere intialisierungen:
distanzensumme = 0
relativeAenderung_DistanzenSumme = change+1
index_groupMeans = list(range(0, k, 1))
cols_groupMeans = ["xvalue", "yvalue"]
group_means = pd.DataFrame(index=index_groupMeans, columns=cols_groupMeans)
group_means = group_means.fillna(0) # intialisiere mit 0

if (trace):
    mpl.style.use('seaborn')
    number_of_colors = k

    color = ["#"+''.join([rd.choice('0123456789ABCDEF') for j in range(6)])
           for i in range(number_of_colors)]
    print(color)
    #colours = ["xkcd:black", "xkcd:red", "xkcd:green", "xkcd:light yellow", "xkcd:grey", "xkcd:pir
     #        "xkcd:lavender", "xkcd:aquamarine", "xkcd:dark pink", "xkcd:light red", "xkcd:chartre
      #       "xkcd:turquoise", "xkcd:teal", "xkcd:orange"]
    colours = np.array(color)


# SCHRITT 1:
# wähle zufällig k punkte aus beobachtungen als startlösung für die gruppenmittelwerte
# stelle sicher, dass keine identischen beobachtungspaare unter den k ausgewählten punkten
if(k > len(x)):
    try:
        raise Exception("k muss kleiner als anzahl der beobachtungen sein!")
    except Exception as e:
        k=x-1


x_unique = x.drop_duplicates() # duplikate herausnehmen
randomStartingRowIndices = rd.sample(list(range(1, len(x_unique), 1)), k=k)
randomStartingPoints = x_unique.loc[randomStartingRowIndices, ]
randomStartingPoints = randomStartingPoints.reset_index() # ACHTUNG: clusternamen sind 0 bis (k-1)
```

Falls der Parameter *trace* auf True gesetzt ist, werden die Datenpunkte des gewählten Datensatzes in der unbearbeiteten Version angezeigt.

Falls die Anzahl *k* der Gruppen größer als die Anzahl der Datenpunkte im Datensatz ist, wird eine Exception geworfen.

Die Daten werden auf Duplikate hin überprüft, sodass dann nur mehr die Datenpunkte, die „*unique*“ sind, übrig bleiben. Daraus werden zufällig Datenpunkte gezogen.

```
# plot wenn trace ist TRUE
 if (trace):
     plt.scatter(x.iloc[:, 0], x.iloc[:, 1], c = "xkcd:mauve",marker = "p",alpha = 0.6)
     plt.scatter(randomStartingPoints.iloc[:, 1], randomStartingPoints.iloc[:, 2],c = colours[0:k], marker = "X", s= 100, alpha = 0.9)
     plt.title("Starting points")
     plt.show()

```

```
while(iter < maxiter and relativeAenderung_DistanzenSumme >= change): # check abbruchbedingungen
                                                                      # (i.e. schritt 4)
    for i in range(0,len(x)): # i ... anzahl der beobachtungen

        distanzenZuClustern_proBeobachtung = list(repeat(float(0), k))

        for j in range(0,k): # j ... anzahl der k zu bildenden gruppen

            ## berechne euklidsche distanzen
            if(iter==0):
                distanzenZuClustern_proBeobachtung[j] = math.sqrt( (float(x.iloc[i,0])-float(randomStartingPoints.iloc[j,1]))**2 + (float(x.iloc[i,1])-float(randomStartingPoints.iloc[j,2]))**2 )
            else:
                distanzenZuClustern_proBeobachtung[j] = math.sqrt( (float(x.iloc[i,0])-float(group_means.iloc[j,0]))**2 + (float(x.iloc[i,1])-float(group_means.iloc[j,1]))**2 )

        ## setze distanz (distanz zum gewählten cluster-mittelpunkt)
        distanz[i] = min(distanzenZuClustern_proBeobachtung)

        ## setze index (clusterzuordnung gemäß minimaler distanz)
        bool_isMin = False
        minDistanzClusterNummer_proBeobachtung = -1

        for l in range(0,k):
            bool_isMin = distanzenZuClustern_proBeobachtung[l] == min(distanzenZuClustern_proBeobachtung)
            if(bool_isMin):
                minDistanzClusterNummer_proBeobachtung = l
                break
        index[i] = minDistanzClusterNummer_proBeobachtung # ACHTUNG: clusternummern starten bei 0
        # CHECK: print("index i", index[i])

```

Solange nun die Abbruchkriterien nicht erfüllt sind, d.h. die maximale Anzahl an Iterationen oder die relative Änderung nicht erreicht sind, läuft der Algorithmus und berechnet neue Zentren und passt die Gruppenzugehörigkeiten an.

```
        ## distanzsumme und relative änderung davon in laufender iteration
        if(iter!=0):
            relativeAenderung_DistanzenSumme = abs(distanzensumme - sum(distanz)) / distanzensumme

        distanzensumme = sum(distanz)

        ## SCHRITT 3: bestimme aufgrund von aktueller gruppenzugehörigkeit der datenpunkte für jede
        ##            der k gruppen durch anwendung von "mean" neue gruppenmittelwerte

        # neue col mit index (i.e. info über clusternummer pro beobachtung) an dataframe dranhängen
        index_series = pd.Series(index)
        x['cluster'] = index_series.values

        # pro cluster neuen mittelwert bilden
        for a in range(0,k):
            try:
                group_means.iloc[a,0] = mean((x[x.cluster == a]).iloc[::,0])
                group_means.iloc[a,1] = mean((x[x.cluster == a]).iloc[::,1])
            except:
                group_means.iloc[a,0] = 0
                group_means.iloc[a,1] = 0
```

```
158
159         if (trace):
160             plt.scatter(x.iloc[:, 0], x.iloc[:, 1], c = colours[index], marker = "p", alpha = 0.6)
161             plt.scatter(group_means.iloc[:, 0], group_means.iloc[:, 1],
162                         c = colours[0:k], marker = "X", linewidths = 3, s= 100, alpha = 0.9)
163             plt.title("Iteration" + str(iter + 1))
164             plt.show()
165
166         ## iterationsschritte-anzahl erhöhen
167         iter = iter + 1
168
169
170     # setze inhalte der outputliste
171     zentren = group_means
172     outputliste.append(iter)
173     outputliste.append(zentren)
174     outputliste.append(index)
175     outputliste.append(distanz)
```

```
176
177     # ergebnisse ausgebenv
178     print("\n Iterationsschritte: ", outputliste[0], "\n \n",
179         "Zentren: ", outputliste[1], "\n  \n",
180         "Index (Cluster): ", outputliste[2], "\n \n",
181         "Distanzen: ", outputliste[3], "\n  \n")
182     print("Anmerkung: Falls Clusternummern ohne Punkte vorkommen, werden die Zentre
183
184
185     ## RETURN list of output
186     return outputliste
```

Wenn alle notwendigen Output-Variablen berechnet wurden und eines der Abbruchkriterien schlagend wurde, werden die Elemente der Outputliste gesetzt, die Ergebnisse werden auf die Console gedruckt, und die *outputliste* wird als Return-Wert zurückgegeben.

```
189 # TEST AUFRUFE DER FUNCTION kmeans
190 ergebnis_fall0 = kmeans()
191 ergebnis_fall1 = kmeans(daten, 10, False, 10, 0.001)
192 ergebnis_fall2 = kmeans(daten, 4, False, 10, 0.01)
193
194 ergebnis_fall3 = kmeans(daten, 7, True, 10, 0.001)
195
```

Mit der *kmeans*-Funktion wurden einige Testaufrufe gemacht. Zwei Aufrufe davon werden untenstehend vorgestellt.

Output der Aufrufe zum Test der Funktion:

ergebnis_fall0 = kmeans()

Die Funktion *kmeans* wird hier mit den Default-Parametern aufgerufen.

Als Output wird angezeigt, dass 4 Iterationsschritte notwendig waren, um die endgültigen Werte zu finden. Die Zentren der 3 Cluster (k=3) werden jeweils mit *xvalue* und *yvalue* angegeben. Die Liste *index* gibt die Zugehörigkeit der jeweiligen Datenpunkte zur den Clustern 0, 1, oder 2 an. Die Liste *distanz* enthält für jeden Datenpunkt die minimale Distanz zum Clustermittelpunkt, zu dem er gehört.

```
In [3]: ergebnis_fall0 = kmeans()

 Iterationsschritte:  4

 Zentren:       xvalue     yvalue
0  4.29793  80.284884
1  0.00000   0.000000
2  2.09433  54.750000

 Index (Cluster):  [0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 2, 0, 2, 0, 2, 2, 0,
0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 2, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2, 2, 0, 2, 0, 0, 2,
0, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0,
2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0,
0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0,
0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0,
0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0,
2, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
2, 0, 2, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2,
0, 2, 2, 0, 0, 2, 0, 2, 0]
```

(Ausschnitt aus den Distanzwerten der einzelnen Datenpunkte:)

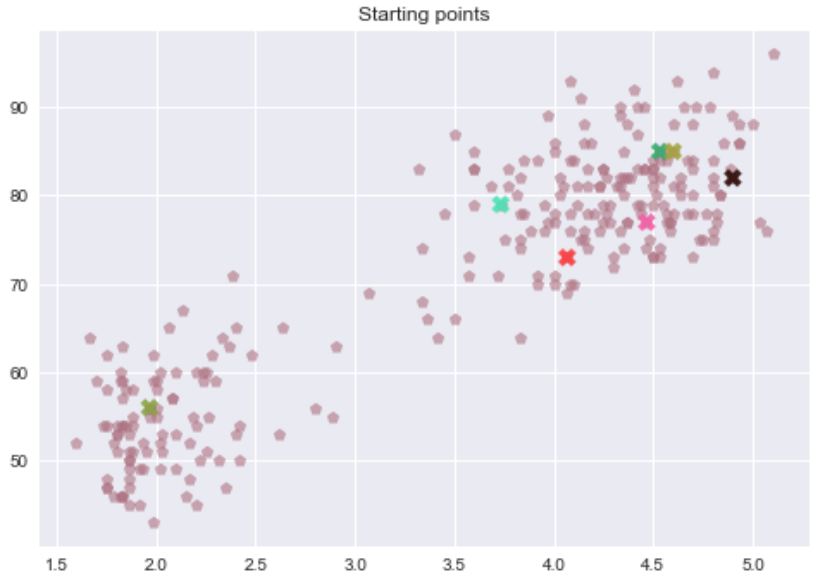
```
 Distanzen:  [1.4622013492779287, 0.8056861354770851, 6.358526066575385,
7.252454506503298, 4.7209722855269804, 0.8273453746169132, 7.725586016442894,
4.766490127406888, 3.752776458690285, 4.715403777602733, 0.794225011504926,
3.7345945976887434, 2.2869814053946538, 7.757645464243645, 2.7447252151646033,
2.75096000132681, 7.258172163079352, 3.7488882376497, 2.7940762603944798,
1.2857773848939067, 3.761532951989122, 7.757645464243645, 2.4371457275793165,
11.351818789618779, 6.289278271882958, 2.803384208129218, 0.28055824511142063,
4.290270796437034, 2.3283760674366163, 1.2919636288953082, 7.284884014959447,
3.289231771446843, 11.321757325119629, 0.3890331124395044, 6.302057085330193,
2.7510870449515044, 6.753826983932887, 0.6061834627250604, 4.258026933792223,
9.72721835856132, 0.28960316837442845, 3.2568635784908158, 3.724847313214377,
3.2681895827659693, 7.288675368202344, 2.886880761309028, 9.411985623071255,
1.750009185375894, 1.7475398707278884, 4.251046712152196, 5.308678742882504,
9.72410749870111, 0.794225011504926, 0.6061834627250604, 0.8325018732111056,
2.777438215630385, 9.303039591783719, 9.259865599937182, 3.2958852528241493,
0.7153704974493058, 4.252261676907949, 3.7206076328905144, 6.758776749449563,
1.7870920239621757, 5.257319842743068, 11.715560919970967, 2.2886319808937703,
```

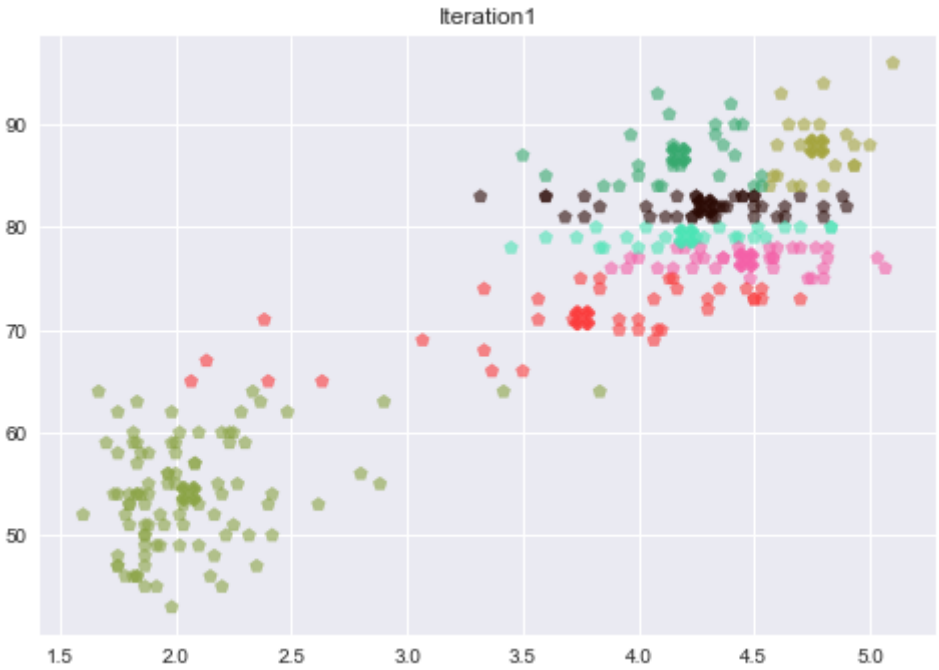
ergebnis_fall3 = kmeans(daten, 7, True, 10, 0.001)

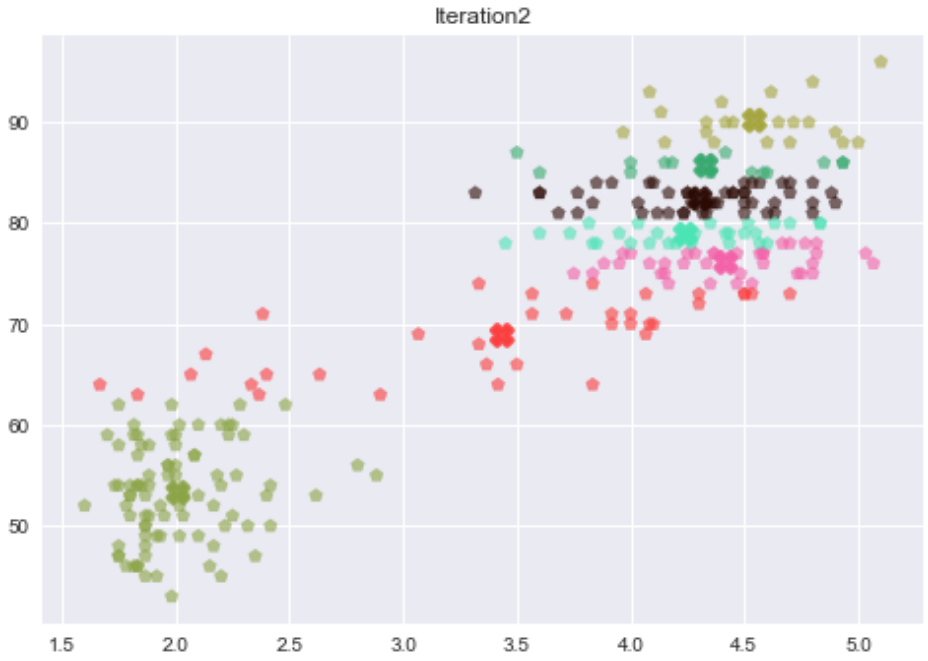
Die Funktion *kmeans* wird hier mit den Parametern daten, *k*=7, *trace*=True, *maxiter*=10 und *change*=0.001 aufgerufen. Da *trace* auf True gesetzt ist, werden anfangs und danach in jedem Iterationsschritt Plots ausgegeben.

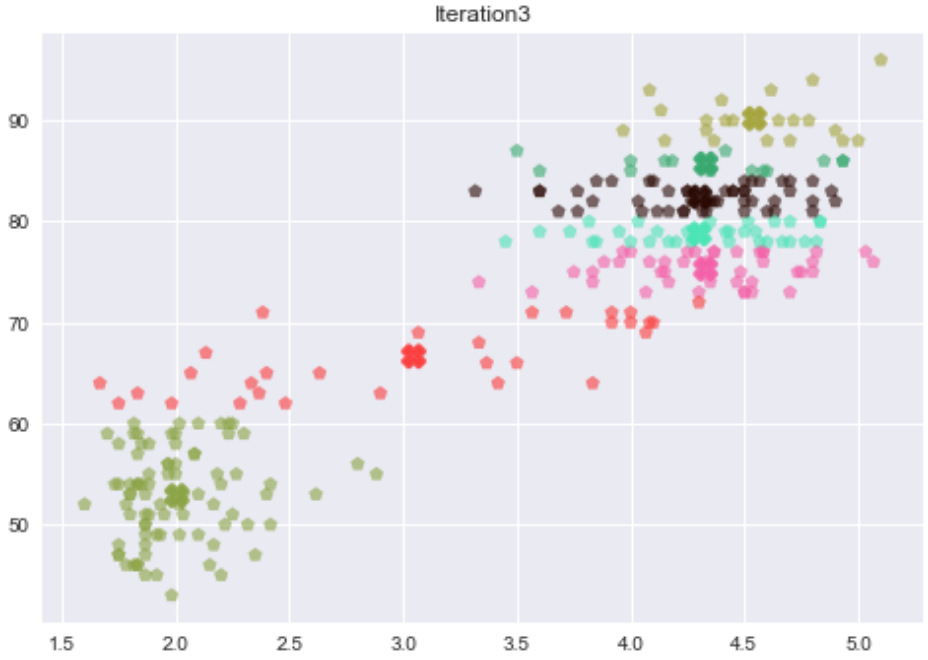
In diesem Beispielaufruf sind 9 Iterationsschritte notwendig, bis der Algorithmus zum Abbruch kommt.

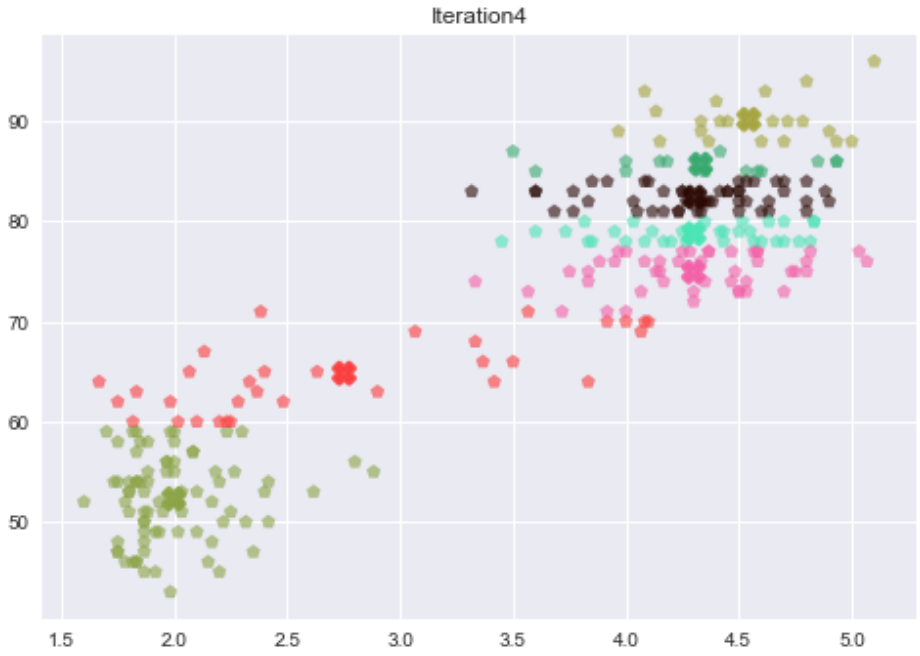
```
In [6]: ergebnis_fall3 = kmeans(daten, 7, True, 10, 0.001)
['#36A96E', '#2B0A04', '#4AE4B5', '#8BA445', '#FC3C3C', '#F560A8', '#A5A53D']
```

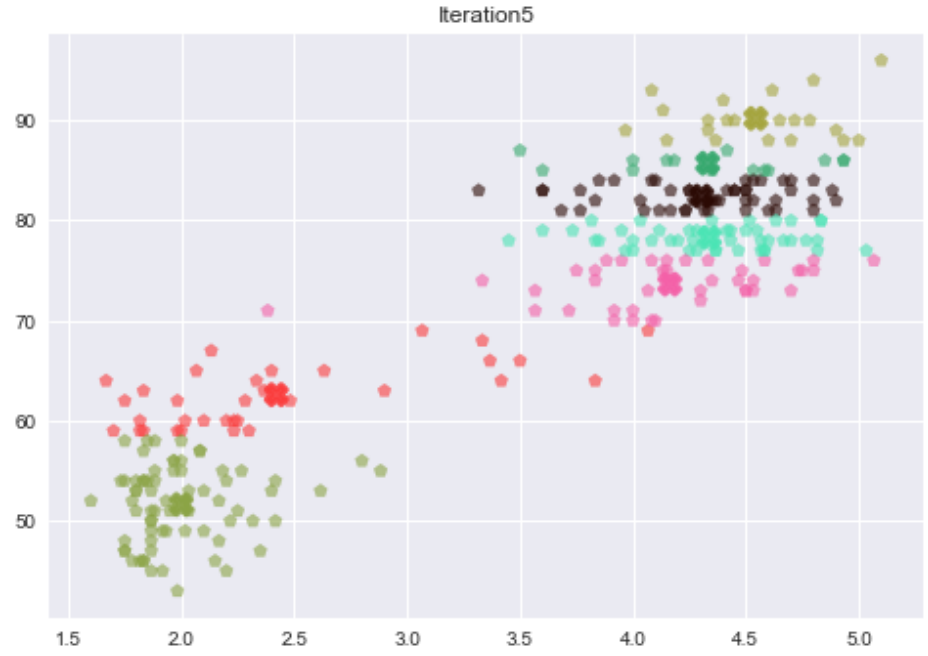


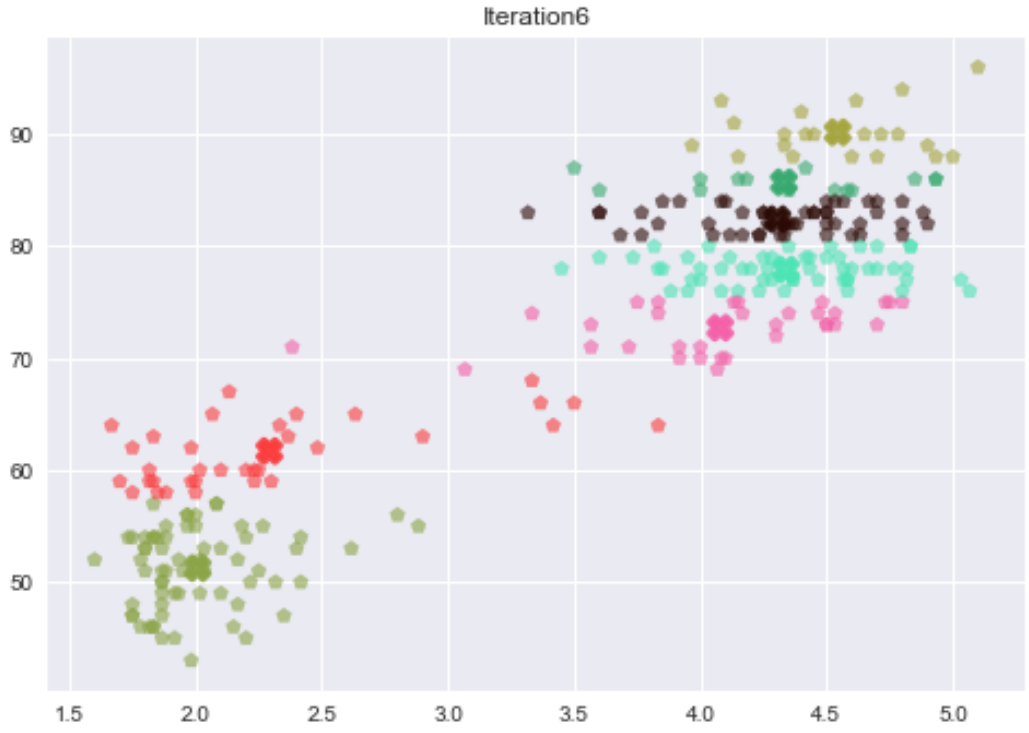


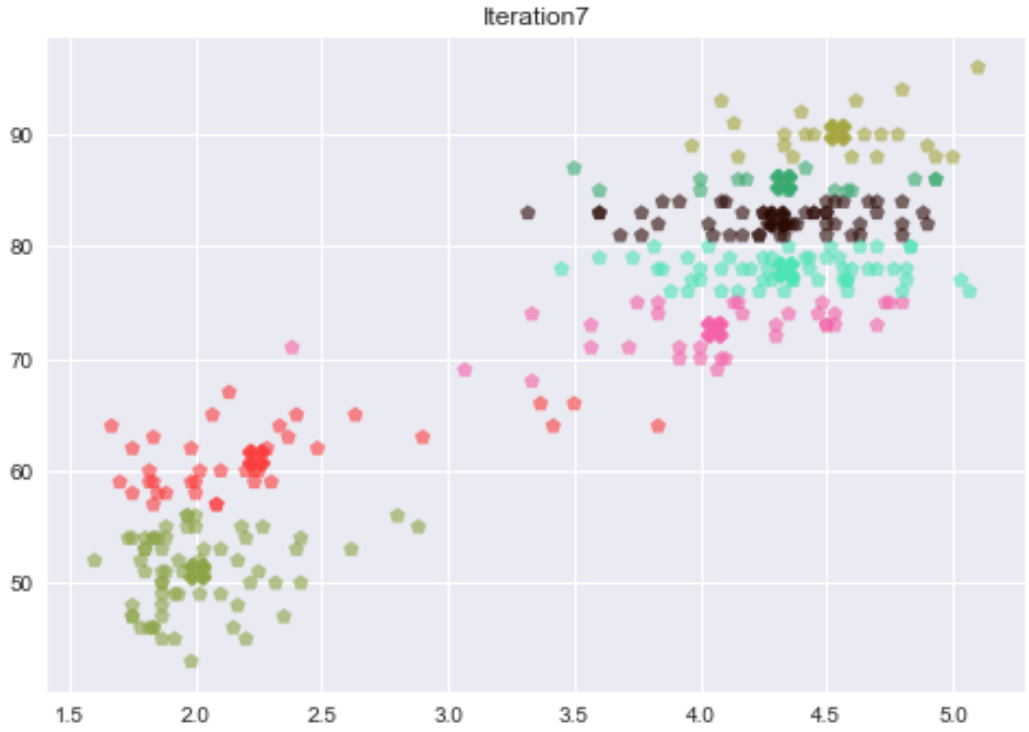

Iteration2
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

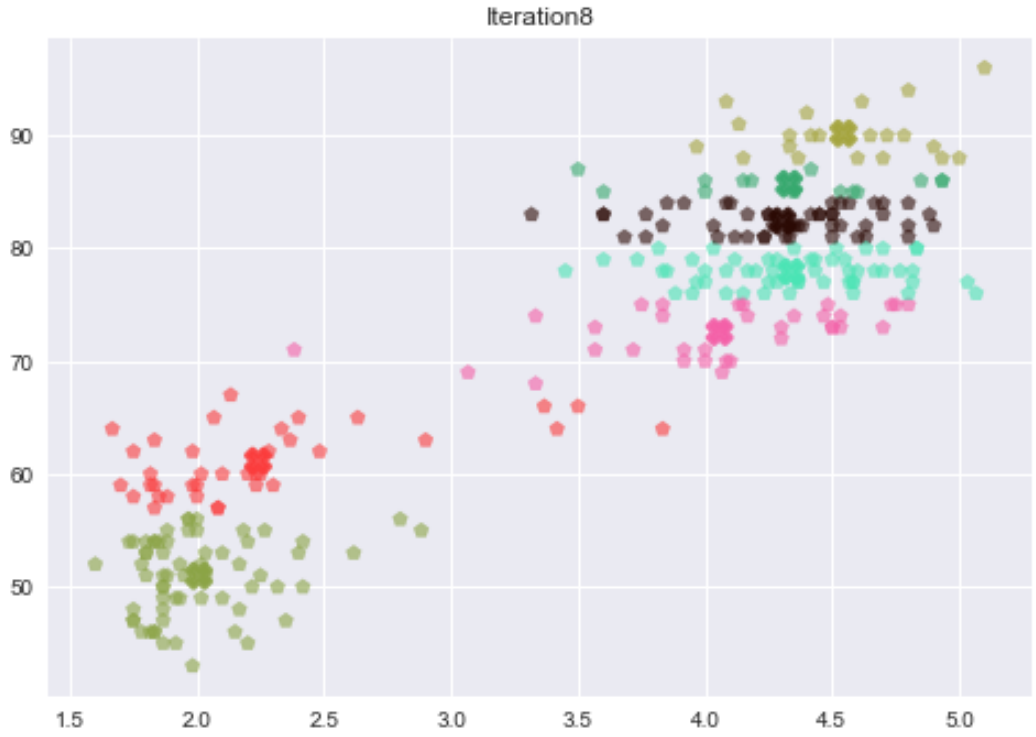

Iteration3
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

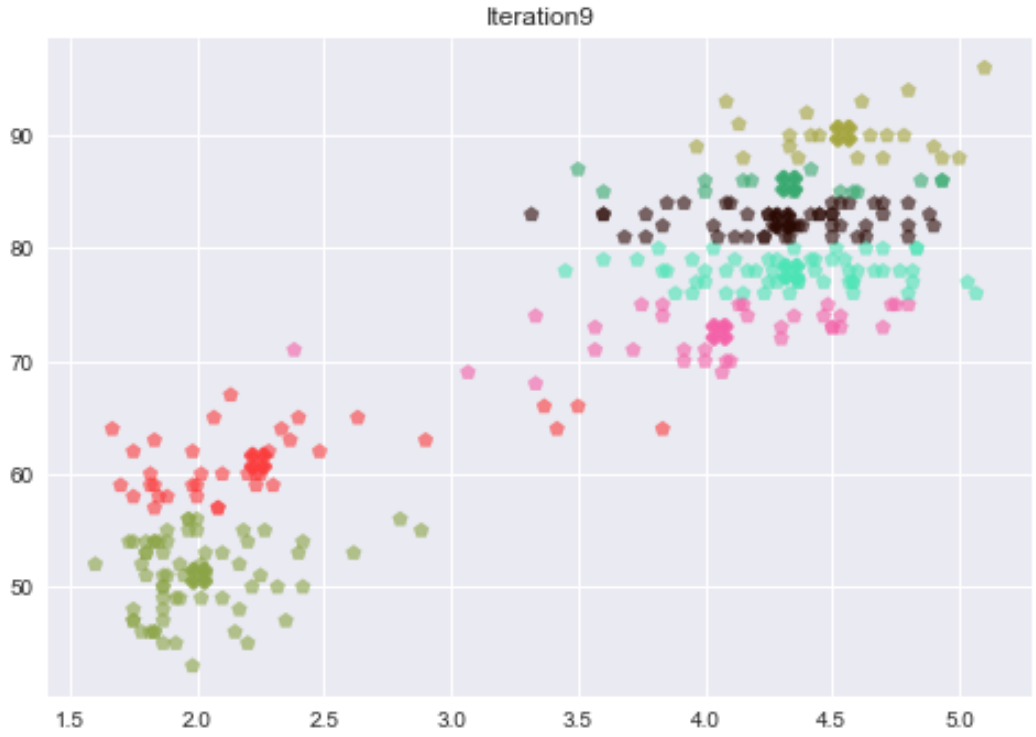

Iteration4
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration5
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration6
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration7
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration8
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration9
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

```
 Iterationsschritte:  9

 Zentren:       xvalue     yvalue
0  4.330857  85.714286
1  4.304755  82.428571
2  4.339148  77.925926
3  2.008238  50.984127
4  2.240919  61.162162
5  4.056853  72.588235
6  4.544381  90.190476

 Index (Cluster):  [2, 3, 5, 4, 0, 3, 6, 0, 3, 0, 3, 1, 2, 3, 1, 3, 4, 1, 3, 2, 3, 3, 2,
5, 5, 1, 3, 2, 2, 2, 5, 2, 4, 2, 5, 3, 3, 2, 4, 6, 2, 4, 1, 4, 5, 1, 4, 3, 1, 4, 5, 6, 3,
2, 3, 1, 5, 4, 2, 1, 4, 1, 3, 1, 4, 6, 2, 2, 4, 5, 1, 3, 2, 5, 4, 2, 4, 2, 2, 1, 5, 1, 5,
4, 5, 6, 2, 2, 3, 0, 4, 6, 3, 2, 4, 5, 1, 5, 3, 1, 4, 6, 3, 1, 1, 3, 1, 3, 0, 1, 5, 4, 6,
2, 4, 1, 3, 0, 4, 0, 3, 5, 2, 3, 6, 1, 3, 1, 3, 6, 3, 1, 3, 6, 3, 1, 3, 0, 3, 2, 1, 4, 1,
2, 2, 4, 2, 3, 6, 3, 2, 2, 4, 1, 5, 5, 1, 6, 3, 6, 3, 0, 4, 2, 4, 2, 4, 6, 3, 6, 3, 4, 2,
5, 1, 1, 5, 3, 0, 5, 3, 2, 1, 1, 3, 2, 1, 3, 1, 3, 1, 4, 2, 1, 2, 1, 0, 2, 3, 2, 4, 1, 6,
3, 2, 3, 2, 1, 3, 1, 5, 2, 3, 5, 4, 2, 3, 6, 3, 2, 3, 1, 3, 5, 2, 2, 2, 2, 5, 2, 5, 3, 0,
3, 6, 3, 3, 2, 2, 4, 5, 3, 0, 4, 0, 1, 4, 1, 4, 5, 3, 1, 5, 5, 6, 2, 5, 1, 3, 2, 2, 1, 4,
1, 3, 4, 5, 1, 3, 6, 3, 5]
```

(Auszug aus den Distanzwerten:)

```
 Distanzen:  [1.3038309328701019, 3.0230536138446853, 1.5865190402969953,
0.8388939503284887, 0.7423380741458181, 4.110041906068567, 2.195997092216868,
1.0219374955925389, 0.060362474848754934, 0.7145421825282509, 3.0209598537374998,
1.618561637461088, 0.15763621279068638, 3.9924872873286583, 0.6948015123815297,
1.0282039324874088, 0.9710682918648432, 1.6476210923803556, 1.0948316431226182,
1.0777673723565433, 0.2088421819012986, 3.9924872873286583, 0.8922283327743027,
3.7222629368580753, 1.4898978509956213, 0.907309410343273, 4.016084742646242,
1.942885108788822, 0.4947250542338582, 1.0781666321557657, 0.47819521664787645,
0.9347111405796591, 4.96716554449157, 2.0965471502832407, 1.4294018763274479,
1.0159108008846056, 2.987467498892879, 2.1320583754493545, 2.2003052197133317,
0.30531987983906866, 2.074102463099365, 3.182353766055899, 1.5931604255716552,
3.2000423317141595, 0.6294967788930097, 1.1411357298115594, 3.2539277439120315,
2.0179604216362907, 0.539831624142187, 2.1755429531449875, 2.523663318982513,
0.2563869666032008, 3.0209598537374998, 2.1320583754493545, 3.028406521083541,
0.8129561945521553, 1.6241894504974725, 2.895290403285515, 0.9535487856916907,
1.4286239057416927, 2.1621766636338013, 1.583511390954017, 2.9952797484086506,
0.6549358581894554, 1.2370643237030219, 1.8152747110682705, 0.1874085199791312,
0.36837620367613727, 3.8417765499697847, 0.7636676713621794, 0.5074685259924712,
5.016042532900012, 1.0860517643474707, 1.5892525307204641, 0.8766381301592397,
2.0588732332025197, 1.1835373139074439, 0.23959013928425815, 1.9792073679159068,
0.907309410343273, 2.412966508492657, 0.4295011567487118, 2.5885949096761762,
3.8578137388528138, 0.41188971316742623, 2.2246821582454013, 1.9648478193897765,
2.0816855060151065, 2.9883472354795746, 0.4371488328256406, 1.162882302336278,
0.2845401310321622, 0.9942102999030696, 0.4835590561329365, 1.882563614489678,
0.6365070725924525, 1.6126404190641799, 2.4312070508388253, 0.14212723939973773,
0.7334779873549006, 0.8721097937359547, 2.197646455486108, 1.9862477530022182,
0.6038635461956513, 1.451108710106326, 3.9866296573917217, 1.6203722672474987,
1.04054321578944, 0.5925723240108978, 1.5580101839989378, 2.504752251560118,
2.162969206814434, 1.2424566258506493, 1.0768918364603042, 2.228797544042564,
```

Generieren eines simulierten Datensatzes mit den vorgegebenen Mittelwerten:

```
##  d) Überprüfen Sie die Funktionalität mit einem simulierten Datensatz:
#generieren Sie 4 Stichproben mit je 25 Beobachtungen (insgesamt n=100), und folgenden
#Mittelwerten (-1,1), (-1,-1), (1,1), (1,-1), die Werte für die beiden Variablen sollen jeweils
#um den Mittelwert normalverteilt mit Standardabweichung 1 sein.
#ok = {"xvalue":np.random.normal(-1,1,25),"yvalue": np.random.normal(1,1,25)}
stichprobe1 = pd.DataFrame(data = {"xvalue":np.random.normal(-1,1,25),"yvalue": np.random.normal(1,1,25)})
stichprobe2 = pd.DataFrame(data = {"xvalue":np.random.normal(-1,1,25),"yvalue": np.random.normal(-1,1,25)})
stichprobe3 = pd.DataFrame(data = {"xvalue":np.random.normal(1,1,25),"yvalue": np.random.normal(1,1,25)})
stichprobe4 = pd.DataFrame(data = {"xvalue":np.random.normal(1,1,25),"yvalue": np.random.normal(-1,1,25)})
#rbind wie im R
temp1 =  stichprobe1.append(stichprobe2, ignore_index = True)
temp2 = temp1.append(stichprobe3, ignore_index =True)
RandomData = temp2.append(stichprobe4, ignore_index= True)

ergebniss_mitRandomData = kmeans(RandomData, 18, True, 10, 0.001)

```

Für den Datensatz wurden 4 Stichproben mit *np.random.normal* generiert, um normalverteilte Realisierungen für die vorgegebenen Zufallsvariablen zu erhalten. Die *kmeans*-Funktion wird dann nochmals für den simulierten Datensatz aufgerufen mit *trace*=True.

Beispiele der generierten Simulationsdaten für die Stichproben:

```
In [12]: stichprobe1
Out[12]:
      xvalue    yvalue
0   0.669832 -0.005516
1   0.176020  0.629248
2  -2.488960  1.379063
3   0.313360 -0.362582
4  -3.058682  0.064280
5  -1.354055  1.218541
6   0.299990 -0.782648
7  -1.245503  1.053920
8  -1.528713  1.511013
9   0.720649  1.341368
10 -0.101534  1.883023
11  1.393897  1.221952
12 -0.165364  2.089727
13 -3.194703 -0.301262
14 -0.128200  0.491918
15 -0.138023  0.670272
16 -2.017918 -0.500455
17 -1.386056  1.430432
18 -1.327392  0.895073
19  0.396751  0.578294
20 -0.362478  0.365806
21  0.024391  2.000523
22 -3.187617 -0.003606
23  0.511568 -0.409913
24  0.165327  0.056433
```

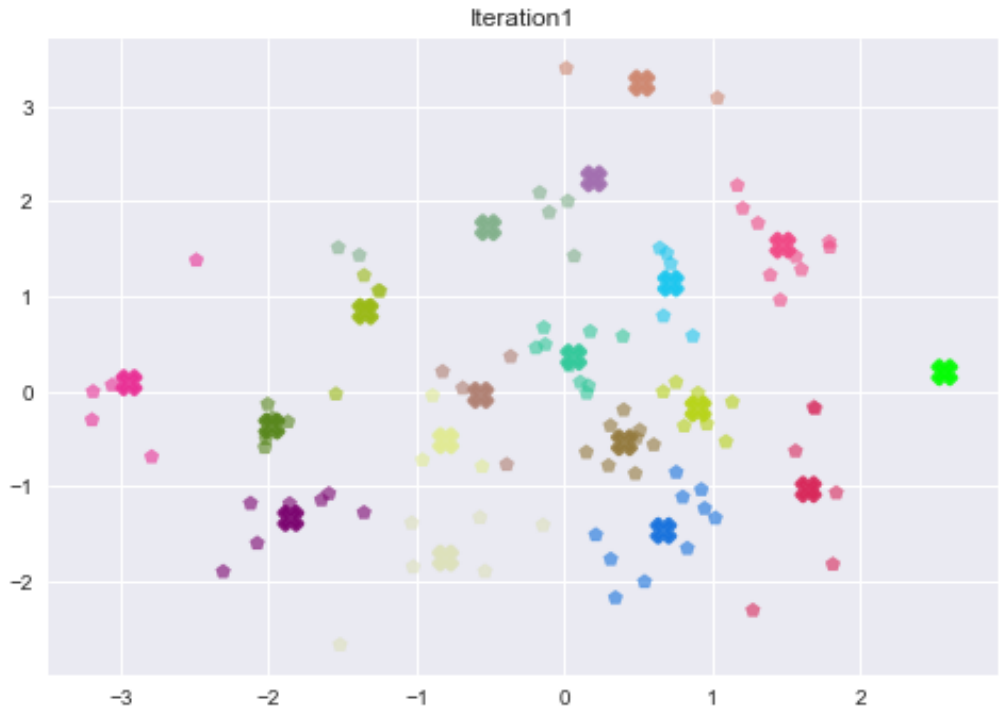
```
In [13]: stichprobe2
Out[13]:
      xvalue    yvalue
0  -1.030969 -1.385482
1  -0.143145 -1.408052
2  -2.791136 -0.689064
3  -1.869157 -0.321599
4  -0.555174 -0.792140
5  -0.535882 -1.894255
6  -1.591671 -1.076972
7  -0.958653 -0.723590
8  -1.021636 -1.847810
9  -1.516355 -2.667113
10 -2.005733 -0.135880
11 -0.686403  0.032827
12 -2.121895 -1.178856
13 -1.354483 -1.276548
14 -2.025415 -0.590994
15 -0.569776 -1.329131
16 -2.076871 -1.599370
17 -1.857554 -1.182470
18 -1.642321 -1.143248
19  0.402527 -0.195957
20 -2.307254 -1.896725
21 -1.257148  1.058972
22 -1.974677 -0.281854
23  1.094152 -0.530266
24 -1.544521 -0.029313
```

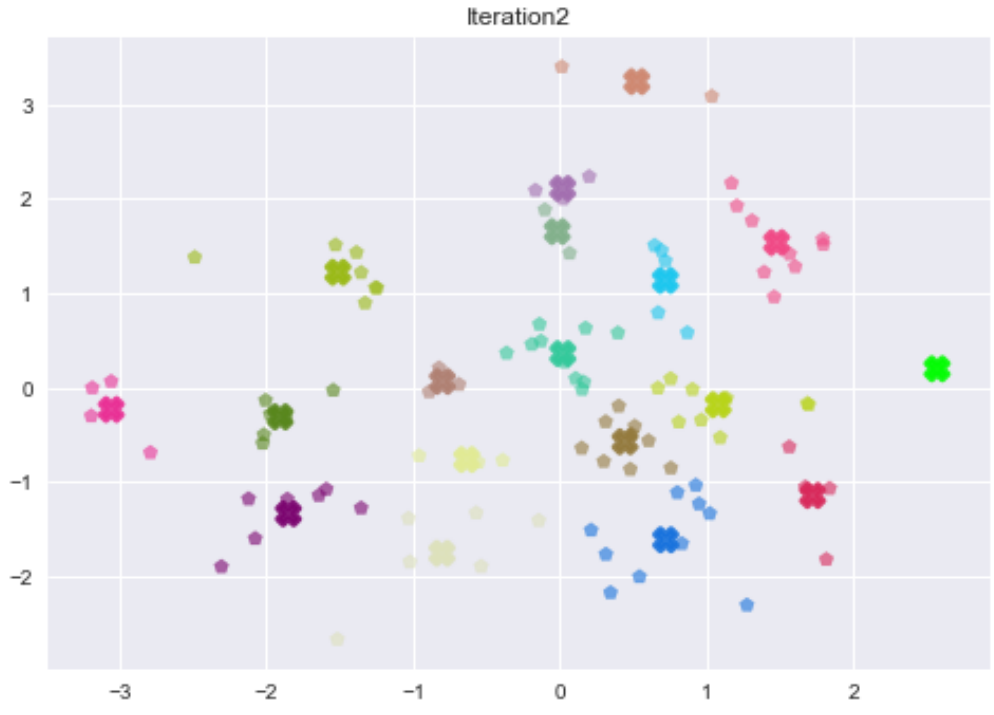
```
In [14]: stichprobe3
Out[14]:
      xvalue    yvalue
0   0.905160 -0.019799
1   1.568469  1.415220
2   1.797345  1.514676
3   1.036384  3.085314
4  -0.191017  0.457793
5   0.965178 -0.345091
6   0.870112  0.580844
7   1.171414  2.165169
8   0.152522 -0.018413
9   0.757964 -0.853201
10  0.488576 -0.492377
11  0.067297  1.422375
12  1.311190  1.766930
13  1.207980  1.924468
14  0.032772  0.270546
15  1.605605  1.281072
16  0.671587  0.792753
17 -0.389220 -0.771057
18  0.200906  2.235356
19  1.687988 -0.167546
20  0.013775  3.397694
21  0.646865  1.506345
22  0.543914 -2.003803
23  0.694967  1.450949
24 -0.823052  0.207731
```

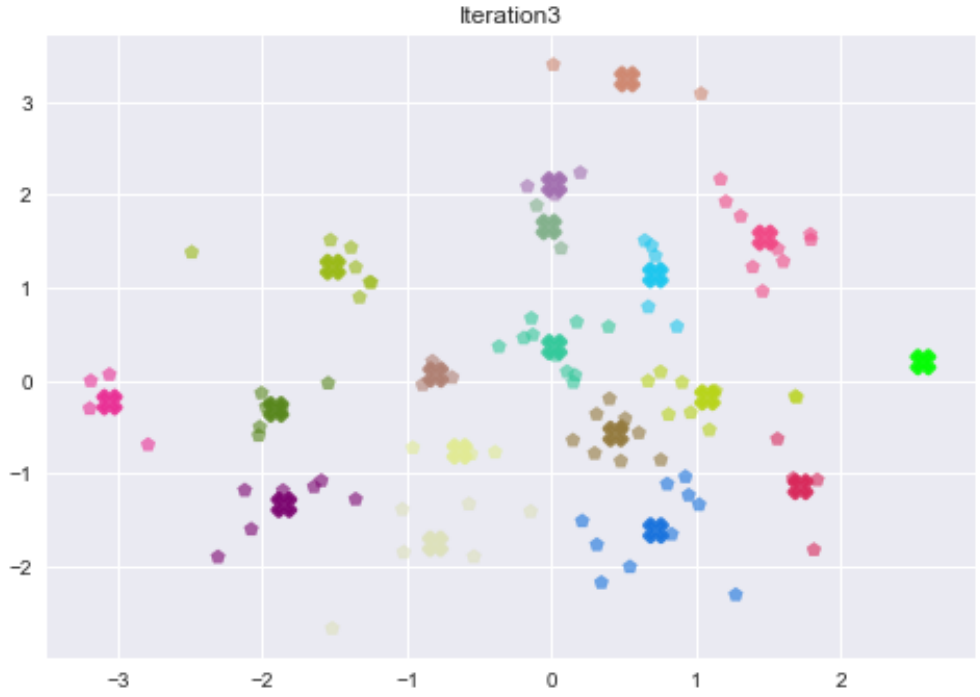
```
In [15]: stichprobe4
Out[15]:
      xvalue    yvalue
0   2.573315  0.199132
1   1.461945  0.959736
2   1.674437 -1.053427
3   1.793877  1.572347
4   1.022639 -1.334023
5   0.928132 -1.035202
6   1.565189 -0.629113
7   0.481265 -0.865453
8   1.698691 -0.183284
9   0.950499 -1.233313
10  0.109248  0.095836
11  0.346569 -2.173102
12  0.605711 -0.563670
13  0.832676 -1.652732
14  0.756558  0.094079
15  1.137459 -0.113208
16  0.811987 -0.366066
17  1.276598 -2.304554
18  1.817905 -1.820649
19  0.802013 -1.112764
20  1.840551 -1.067301
21 -0.890667 -0.045668
22  0.213918 -1.510988
23  0.315013 -1.766380
24  0.149952 -0.641438
```

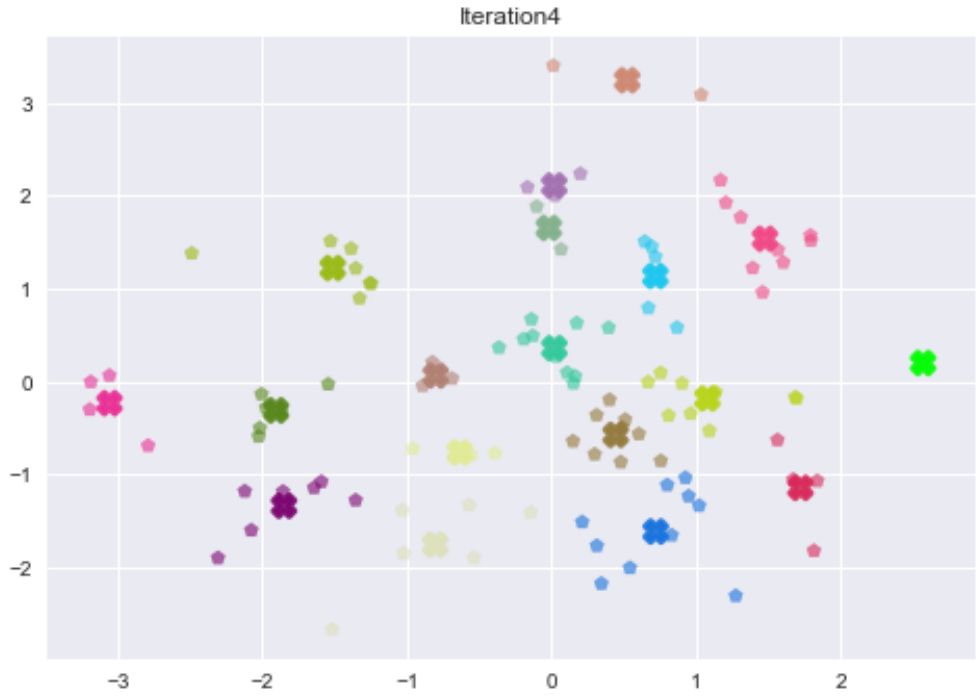
Output von k-means mit generiertem Random Data:

```
In [20]: ergebniss_mitRandomData = kmeans(RandomData, 18, True, 10, 0.001)
['#1AC7EE', '#58871C', '#DDE1BB', '#82B18B', '#E92F95', '#9ABA15', '#B08072', '#1772DD',
'#947A3D', '#33C99B', '#D8285A', '#E1EB94', '#F14885', '#B8D418', '#A36FB0', '#7B036F',
'#CF8870', '#03FD03']
```


Iteration1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


Iteration2
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


Iteration3
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


Iteration4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

```
 Iterationsschritte:  4

 Zentren:        xvalue    yvalue
0   0.720836  1.134452
1  -1.906237 -0.310016
2  -0.802961 -1.755307
3  -0.017118  1.652699
4  -3.058035 -0.232413
5  -1.512547  1.221002
6  -0.800040  0.064963
7   0.723197 -1.612686
8   0.445657 -0.574138
9   0.021292  0.359773
10  1.724520 -1.142623
11 -0.634349 -0.762262
12  1.479080  1.535730
13  1.080778 -0.181855
14  0.019978  2.108535
15 -1.850293 -1.336313
16  0.525080  3.241504
17  2.573315  0.199132

 Index (Cluster):  [13, 9, 5, 8, 4, 5, 8, 5, 5, 0, 3, 12, 14, 4, 9, 9, 1, 5, 5, 9, 9, 14,
4, 8, 9, 2, 2, 4, 1, 11, 2, 15, 11, 2, 2, 1, 6, 15, 15, 1, 2, 15, 15, 15, 8, 15, 5, 1, 13,
1, 13, 12, 12, 16, 9, 13, 0, 12, 9, 8, 8, 3, 12, 12, 9, 12, 0, 11, 14, 13, 16, 0, 7, 0, 6,
17, 12, 10, 12, 7, 7, 10, 8, 13, 7, 9, 7, 8, 7, 13, 13, 13, 7, 10, 7, 10, 6, 7, 7, 8]

 Distanzen:  [0.4471831073327317, 0.31073702114979984, 0.9891236976559491,
0.24951586117957406, 0.2966937684495529, 0.1585108360696725, 0.25435295238191186,
0.31500600655645056, 0.29046137663027405, 0.2069166390132019, 0.24530639654502898,
0.3251353767043738, 0.18629399083111187, 0.15303065320416437, 0.19952443388423266,
0.34898503924739355, 0.22077111941747757, 0.24466475412098715, 0.3748494132656599,
0.4344195858914189, 0.3838174368397748, 0.10810210178472568, 0.2629531862775721,
0.17695777584891784, 0.3357993505908983, 0.4344634340654042, 0.7456155088876653,
0.5289281261334146, 0.03884644486617131, 0.08462465960553264, 0.3010604190124594,
0.3662551628860265, 0.3266012814618378, 0.2374355782434657, 1.1577228322694408,
0.20055597385771226, 0.11809440812662501, 0.3139434638841894, 0.499398578994049,
0.3052078961016041, 0.4857996931470026, 0.3471841744544979, 0.1540138465606289,
0.2837717660161836, 0.3806322829663853, 0.7231011456619469, 0.3024605271746114,
0.07400766687950644, 0.34866708617912956, 0.45785648503298326, 0.23896479836996612,
0.15004316685174374, 0.31895992838828546, 0.534628439639729, 0.23384420618228177,
0.20002366096814977, 0.5733798073234616, 0.7006085489759915, 0.4003070414917724,
0.41882233408469405, 0.09234103240479175, 0.24530639654502878, 0.28572799896526224,
0.4739329854633801, 0.0899626226262032, 0.2843576507238804, 0.3452297921429757,
0.24528636698270004, 0.22094941825434603, 0.6073783248443888, 0.5346284396397291,
0.37917811922098427, 0.4302502615394706, 0.31755221923633153, 0.14461056948589499, 0.0,
0.5762488759656286, 0.10229458972405606, 0.3169195606591161, 0.40904608003690907,
0.6127692195581756, 0.5376606486371338, 0.29348334055393366, 0.6179144619686661,
0.44225574758711395, 0.27820682526542323, 0.6752145445265775, 0.16039585035177967,
0.1165732031676424, 0.4257446768169896, 0.08902288635011682, 0.3258563146741214,
0.885964680732381, 0.6844275466666184, 0.5060971278212751, 0.13833412744959428,
0.14301236949852053, 0.5193335342768186, 0.43616029189576533, 0.3032669081205717]
```

Optionale Mehrleistung

- Ermittlung der Silhouetten-Werte für jede Beobachtung und der Silhouettenkoeffizienten für jeden Cluster
- Implementierung eines Silhouetten-Plots

Lösung

Für Lösung dieser Aufgabestellung wurde eine neue Funktion `silhouetten` geschrieben.nDiese Funktion berechnet die Silhouettenwerte und Silhouettenkoeffizienten von einem gruppierten Datensatz. Die Silhouettenwerte werden mithilfe eines Silhouettenplots dargestellt.

Silhouettenwerte werden folgenderweise berechnet[1]:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$, wobei

- *a(i)* – durchschnittliche Distanz zwischen jedem Punkt und allen restlichen Punkten im selben Cluster (Variable `a` im Code).
- *b(i)* – minimale mittlere Distanz von jedem Punkt i zu allen anderen Punkten in einem anderen Cluster, in dem i nicht liegt (Variable `b` im Code).

Um Funktion ausführen zu können, braucht man folgende Input-ParameterÖ

- *x* – dataframe (dimension nx2, d.h. n rows, 2 cols)
- *erg* – list mit Ergebnissen vom KMEANS-Algorithmus

Ausgegeben wird eine Liste mit folgenden Elementen:

- Vektor der Länge n mit Silhouettenwerten
- Summary von Silhouettenwerten
- Silhouettenkoeffizienten von Clusters

Dazu werden auch folgende Plots dargestellt:

- barplot von Silhouettenwerten
- plot von geclusterten Punkten

[1] Quelle: https://en.wikipedia.org/wiki/Silhouette_(clustering)

```
244 def silhouetten(x = daten, erg = ergebnis):
245
246     outputliste = list()
247
248     # Anzahl der Punkte
249     n = len(x.index)
250     # Vektor von Clusternummern
251     ind = np.array(erg[2])
252     # Anzahl der Cluster
253     k = len(np.unique(ind))
254     # Initialisierung
255     silhouetten_werte = np.zeros(n)
256
```

Aus den Input-Parametern können Anzahl der Beobachtungen im gegebenen Datensatz (*n*), Anzahl der Clustern (*k*) und Clusters von den Punkten (*ind*) herausgelesen werden. Für Output und weitere Berechnung der Silhouettenwerte (*silhouetten_werte*) sollte eine Liste bzw. ein Vektor lauter 0 erstellt werden. Mittels zuvor erwähnter Formel werden die Silhouettenwerte in einer for-Schleife berechnet.

```
for i in range(0, n):
    # mittlere Distanz von einem Punkt bis zur anderen in demselben Cluster
    bool_i = ind == ind[i]

    dist_within = ( (x.iloc[i,0] - x.iloc[bool_i,0])**2 + (x.iloc[i,1] - x.iloc[bool_i,1])**2 ) ** 0.5
    a = mean(dist_within[dist_within != 0])

    # kleinste mittlere Distanz von einem Punkt bis zur anderen in unterscheidenden Clustern
    dist_nextgroup = np.zeros(k)
    vek = np.array(range(0, k))

    for j in vek[vek != ind[i]]:
        bool_j = ind == j
        dist_nextgroup[j]  = mean(( (x.iloc[i,0] - x.iloc[bool_j,0])**2 + (x.iloc[i,1] - x.iloc[bool_j,1])**2 ) ** 0.5)

    b = min(dist_nextgroup[dist_nextgroup != 0])

    silhouetten_werte[i] = (b - a)/max(a, b)
```

Ein Dataframe (*yy*) mit Clusters (*ind*) von Punkten und Silhouettenwerten dient dazu, Silhouettenkoeffizienten zu berechnen. Silhouettenkoeffizient eines Clusters wird als Mittelwert aller Silhouettenwerte dieses Clusters definiert.[2]

```
277     # Dataframe fuer Plot und Berechnung von Koeffizienten
278     yy = pd.DataFrame({"cluster": ind, "werte": silhouetten_werte})
279     yy = yy.sort_values(by = ['cluster', 'werte'])
280
281     # Silhouettenkoeffiziente
282     n_in_cluster = yy.groupby(["cluster"]).agg({'werte': np.size})
283     sil_koef = yy.groupby(["cluster"]).agg({'werte': np.mean})
284
285     # Vorbereitung fuer Plots
286     mpl.style.use('seaborn')
287     color = ["#"+''.join([rd.choice('0123456789ABCDEF') for j in range(6)])
288                 for i in range(k)]
289     print(color)
290     colours = np.array(color)
291
```

[2] https://de.wikipedia.org/wiki/Silhouettenkoeffizient

Weiters werden 2 Plots erstellt: Silhouetten-Plot als horizontaler Barplot von Silhouettenwerten und Scatterplot der geclusterten Beobachtungen und deren Mittelwerten.

```
    lbs = list()
    for i in range(k):
        txt = "{0} cluster: {1} | {2}".format(i+1, n_in_cluster.iloc[i, 0], round(sil_koef.iloc[i, 0], 3))
        lbs.append(txt)

    #Silhouettenplot
    fig, (ax1, ax2) = plt.subplots(1, 2, sharey = False, sharex = False, figsize = (16,8))

    y_pos = np.arange(n)
    for i in range(k):
        bool_1 = yy['cluster'] == i
        ax1.barh(y_pos[bool_1], yy['werte'].loc[bool_1], color = colours[i], linewidth = 0,
                    label = lbs[i], alpha = 0.7)

    ax1.set_title("Silhouettenplot", fontsize = 14)
    ax1.set_xlabel("Silhouettenwerte", fontsize = 12)
    ax1.set_xlim(min(yy['werte']), 1.5)
    ax1.legend(title = "Silhouettenkoeffizienten")

    # Geclusterte Punkte
    zentren = erg[1]
    ax2.scatter(x.iloc[:, 0], x.iloc[:, 1], c = colours[ind], marker = "p", alpha = 0.6)
    ax2.scatter(zentren.iloc[:, 0], zentren.iloc[:, 1],
                                c = colours[0:k], marker = "X", linewidths = 3, s= 100, alpha = 0.9)
    ax2.set_title("Clustering points", fontsize = 14)

    plt.show()
```

Am Ende der Funktion werden die Silhouettenwerte, deren Summary und Silhouettenkoeffizienten auf die Konsole gedruckt und in der Outputliste gespeichert.

```
    print("\n Silhouettenwerte: \n ", silhouetten_werte, "\n \n",
      "Summary von Silhouettenwerten: \n ", yy['werte'].describe(), "\n  \n",
      "Silhouettenkoeffizienten: \n", sil_koef, "\n \n")

    outputliste.append(silhouetten_werte)
    outputliste.append(yy['werte'].describe())
    outputliste.append(sil_koef)
    return outputliste
```

Output der Aufrufe zum Testen der Funktion:

ergebnis_faith = kmeans(daten, 7, False, 10, 0.001)

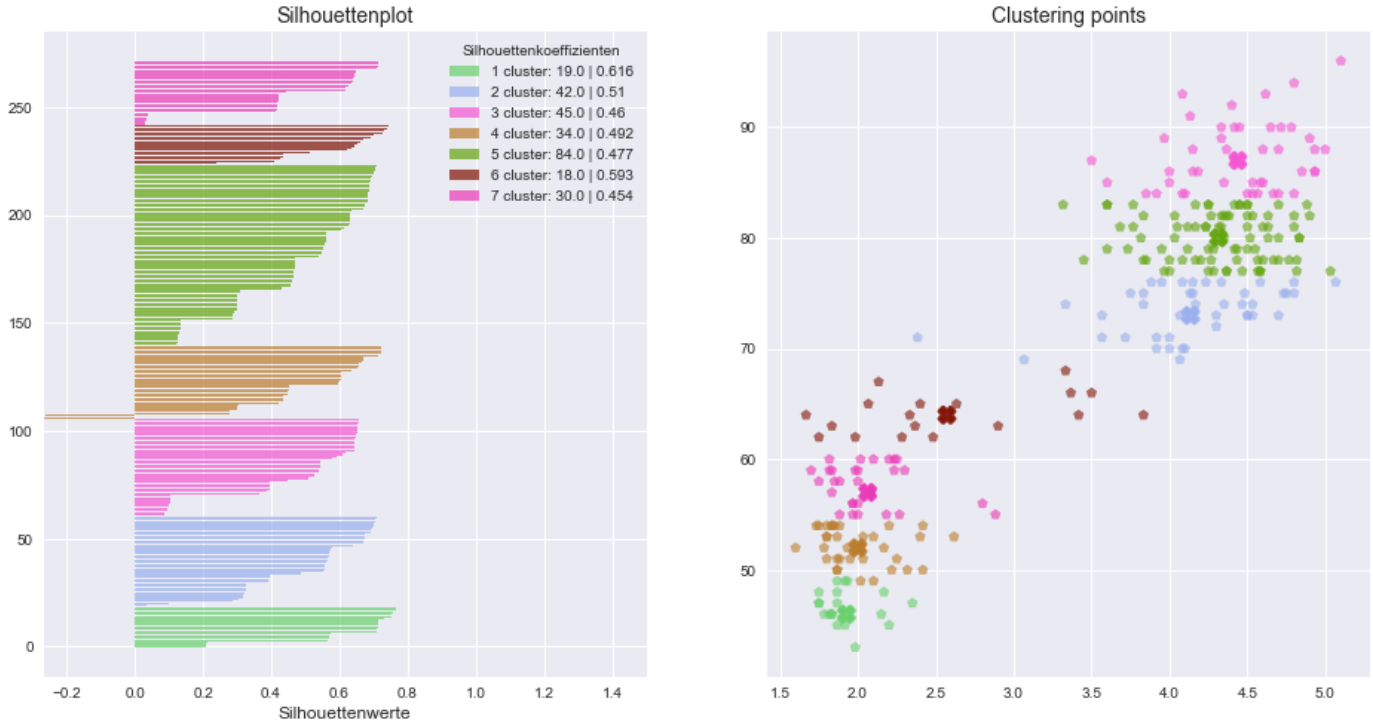
silhouetten(erg = ergebnis_faith)

Vor der Verwendung der Funktion *silhouetten* muss die Funktion *kmeans* ausgeführt werden. Die Funktion *silhouetten* wird hier mit einem Default-Parameter *x* aufgerufen (zur Erinnerung: *daten* ist er Datensatz faithful).

Zuerst werden 2 Plots angezeigt:

1. Silhouetten-Plot: aus dem Plot kann man nicht nur die Silhouettenwerte, sondern auch die Anzahl der Beobachtungen und Silhouettenkoeffizienten jedes Clusters sehen.
2. Scatterplot der geclusterten Punkte und deren Mittelwerte (als Kreuze).

Danach werden eine Liste von berechneten Silhouettenwerten, deren deskriptive Statistiken und Silhouettenkoeffizienten auf die Konsole gedruckt.



Ausschnitt aus den Silhouettenwerten der einzelnen Beobachtungen des eingegebenen Datensatzes:

```
Silhouettenwerte:
 [ 0.60654663  0.45098151  0.64086189  0.43615759  0.39709971  0.02866375
  0.65415681  0.36553076  0.60536175  0.39647809  0.43483445  0.08845881
  0.46994854  0.71198267  0.30018939  0.72034743  0.40691944  0.09750356
  0.71147567  0.63281622  0.60096348  0.71198267  0.42912169  0.03546409
  0.67112964  0.28707168  0.03436583  0.32487064  0.45662548  0.63255482
  0.70799845  0.13428765  0.65980962  0.7023754   0.66845483  0.72390298
  0.57191771  0.6931892   0.64536239  0.64222328  0.70693948  0.71438882
  0.10294945  0.71043635  0.70457935  0.28466238  0.67115098  0.66732348
  0.55422086  0.64685997  0.55293875  0.64335294  0.43483445  0.6931892
  0.44748309  0.29277017  0.57013863  0.69344704  0.13418802  0.68771302
  0.63953483  0.1027107   0.56938768  0.54676823  0.41688023  0.57705913
  0.46930521  0.46468909  0.73046314  0.69739436  0.55440611  0.42234698
  0.6312722   0.57628426  0.42448064  0.28756117  0.42099681  0.4688376
  0.31875163  0.28707168  0.57082675  0.56041019  0.39426634  0.74285755
  0.70661989  0.6492933   0.32143482  0.70597454  0.56357539  0.53804215
  0.41816323  0.64358533  0.27788044  0.45886803  0.62108376  0.69351467
  0.10168165  0.5573459   0.60411718  0.54123664  0.43650854  0.65446296
 -0.26188274  0.29834081  0.68379392  0.72916779  0.10090899  0.72057637
  0.53953749  0.66793447  0.55734628  0.63578486  0.65321084  0.63274615
  0.63925395  0.68159335  0.30184891  0.39608754  0.64487782  0.61752712
  0.63373004  0.10074924  0.13137814  0.42234698  0.65508216  0.67236309
  0.71443209  0.55837305  0.03783453  0.64405956  0.71403354  0.3076481
  0.41155861  0.65725693  0.75274843  0.56028568  0.60458682  0.52614423
  0.66876227  0.61473529  0.68375871  0.41666952  0.55754453  0.1293243
  0.32376937  0.64700585  0.7037958  -0.26711072  0.4490285   0.65598591
  0.12377477  0.12364325  0.74167835  0.68269438  0.56421245  0.39218109
```

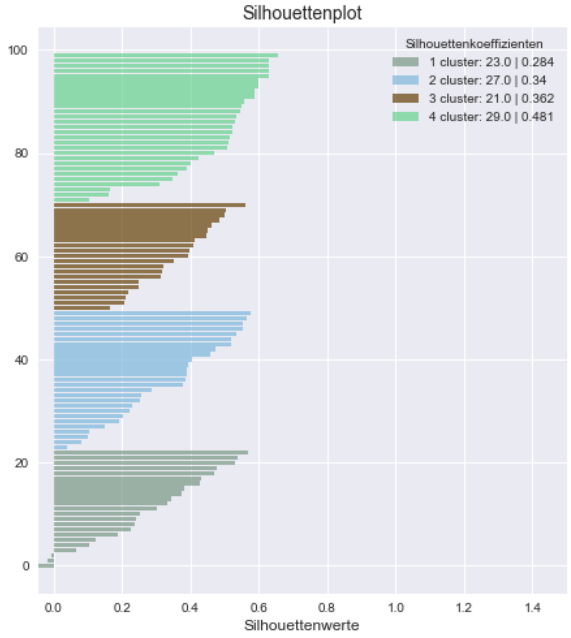
```
 Summary von Silhouettenwerten
  count    272.000000
mean       0.495962
std        0.207154
min       -0.267111
25%        0.393582
50%        0.560027
75%        0.655986
max        0.766195
Name: werte, dtype: float64

 Silhouettenkoeffizienten:
            werte
cluster
0        0.615775
1        0.510116
2        0.460066
3        0.491933
4        0.476725
5        0.593478
6        0.454026
```

<u>Output von k-means mit generiertem Random Data:</u>

Folgender Code wurde ausgeführt:

```
ergebnis_RD = kmeans(RandomData, 4, False, 10, 0.001)
silhouetten(x = RandomData, erg = ergebnis_RD)
```




```
Silhouettenwerte:
 [ 0.34519424  0.30767177  0.39890172  0.42389438  0.6304136   0.50939943
  0.51856318  0.5977561   0.62781973  0.5285677   0.15875809  0.63008103
  0.62965543 -0.01694618  0.53228119  0.59831806  0.38940361  0.55536334
  0.50760621  0.52285794  0.51408013  0.23797911  0.65586973  0.54624898
  0.54736241  0.25086285  0.4605608   0.06479301  0.22413949  0.37185713
  0.52154465  0.58879891  0.39677032  0.5317784  -0.00621004  0.53732742
  0.36175396 -0.04598191  0.1034159   0.42802632  0.16336652  0.18719495
  0.46733282  0.56763276  0.34144152  0.33287687  0.30201287  0.10283927
  0.42934321  0.4773159   0.45774115  0.56542171  0.14816528  0.19068701
  0.38746393  0.58693545  0.25114863  0.39230146  0.3146009   0.2017165
  0.40426579  0.10506639  0.55164242  0.47198068  0.37873608  0.28615665
  0.23007352  0.16352981  0.51947906  0.57449391  0.53484656  0.03736106
  0.55136256  0.22214541  0.2396224   0.24736655  0.09953335  0.45144132
  0.38305105  0.35265847  0.12135509  0.21941773  0.46989541  0.20874155
  0.38960319  0.4993198   0.5618443   0.3217425   0.25647208  0.41088269
  0.50239266  0.31491253  0.39194022  0.40777706  0.20500244  0.38421652
  0.24975464  0.48355767  0.44672932  0.08014546]
```

```
 Summary von Silhouettenwerten:
  count    100.000000
mean       0.372846
std        0.172013
min       -0.045982
25%        0.239212
50%        0.392121
75%        0.519995
max        0.655870
Name: werte, dtype: float64

 Silhouettenkoeffizienten:
             werte
cluster
0        0.284034
1        0.340400
2        0.362418
3        0.481043
```