Exercise 4

##Exercise 4-1 Color-histograms and distance functions (1 point)

```
import math
import numpy as np

P = np.array([2, 3, 5])
Q = np.array([4, 7, 8])

distance = math.sqrt(sum([(a-b)**2 for a, b in zip(P, Q)]))
print("Euclidean distance from P to Q:", distance)
```

```
Euclidean distance from P to Q: 5.385164807134504
Euclidean distance from P to Q: 5.385164807134504
```

```
P = np.array([2, 3, 5])
Q = np.array([4, 7, 8])
distance2 = sum(abs(a-b) for a, b in zip(P, Q))

print("Manhatten distance form P to Q:", distance2)
```

```
Manhatten distance form P to Q: 9
```

```
P = np.array([2, 3, 5])
Q = np.array([4, 7, 8])

distance3 = max(abs(a-b) for a, b in zip(P, Q))

print("Maximum distance form P to Q:", distance3)
```

```
Maximum distance form P to Q: 4
```

```
import numpy as np

P = np.array([2, 3, 5])
Q = np.array([4, 7, 8])
w = np.array([1, 1.5, 2.5])

def weightedL2(a, b, c):
    q = a-b
    return np.sqrt((c*q*q).sum())


print("Weighted Euclidean form P to Q:", weightedL2(P, Q, w))
```

```
Weighted Euclidean form P to Q: 7.106335201775948
```

```python
P = np.array([2, 3, 5])
Q = np.array([4, 7, 8])
M1 = np.matrix([[1,0,0], [0,1,0], [0,0,1]])
M2 = np.matrix([[1,0.9,0.7], [0.9,1,0.8], [0.7,0.8,1]])

def QuadraticForm(a, b, c):
    q = a-b
    p = np.dot((q), c)
    return np.sqrt((np.dot(p, q.T)).sum())

print("Quadratic form form P to Q:", QuadraticForm(P, Q, M1))
print("Quadratic form form P to Q:", QuadraticForm(P, Q, M2))



##TIl en af slide fra svarmulighederne
print("form P to Quadratic form Q:", QuadraticForm(np.array([0,0]),
np.array([0,2]), np.matrix([[2,0],[0,4]])))

print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([4,0]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([np.sqrt(8), 0]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([8, 0]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([np.sqrt(2), 0]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([np.sqrt(2),np.sqrt(3)]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([1, 0]), np.matrix([[2,0],[0,4]])))
print("Quadratic form form P to Q:", QuadraticForm(np.array([0,0]),
np.array([0,-2]), np.matrix([[2,0],[0,4]])))
```

```
Quadratic form form P to Q: 5.385164807134504
Quadratic form form P to Q: 8.426149773176359
form P to Quadratic form Q: 4.0
Quadratic form form P to Q: 5.656854249492381
Quadratic form form P to Q: 4.0
Quadratic form form P to Q: 11.313708498984761
Quadratic form form P to Q: 2.0
Quadratic form form P to Q: 4.0
Quadratic form form P to Q: 1.4142135623730951
Quadratic form form P to Q: 4.0
```

a) Extract from each picture a color histogram with the bins red, orange, and blue (the white pixels are ignore

```python
q = np.array([1, 8, 7])
a1 = np.array([1, 4, 4])
```

```
b1 = np.array([8, 1, 7])
c1 = np.array([2, 4, 10])
d1 = np.array([1, 2, 13])
```

(b) Which pictures are most similar to the query q, using Euclidean distance? Give a ranking according to similarity to

```
plot1 = math.sqrt(sum([(a-b)**2 for a, b in zip(q, a1)]))
plot2 = math.sqrt(sum([(a-b)**2 for a, b in zip(q, b1)]))
plot3 = math.sqrt(sum([(a-b)**2 for a, b in zip(q, c1)]))
plot4 = math.sqrt(sum([(a-b)**2 for a, b in zip(q, d1)]))

print("What distance are most similar to the query q:", list([plot1,
plot2, plot3, plot4]))

#so the first plot by figure a is the most similar
```

```
What distance are most similar to the query q: [5.0,
9.899494936611665, 5.0990195135927845, 8.48528137423857]
```

(c) The results are not entirely satisfactory. What could you change in the feature extraction or in the distance function to get better results? Report the improved feature extraction and features or the improved distance function

Debatably, picture b is more similar to q than a or d are. The problem is that the Euclidean distance takes each color individually to compute the distance but does not take similarity between different colors (i.e., bins in the histogram) into account.

A solution would be to use the quadratic form (a.k.a. Mahalanobis-) distance. We need a similarity matrix to define the (subjective) similarity of bins with each other:

```
M3 = np.matrix([[1, 0.9, 0], [0.9, 1, 0], [0,0,1]]) #havde matrix
været med (1,0,0);(0,1,0);(0,0,1) havde svaret været præcis det samme
som før (kig på slide 131)

print("figure a from P to Quadratic form Q:", QuadraticForm(q, a1,
M3))
print("figure b from P to Quadratic form Q:", QuadraticForm(q, b1,
M3))
print("figure c from P to Quadratic form Q:", QuadraticForm(q, c1,
M3))
print("figure d from P to Quadratic form Q:", QuadraticForm(q, d1,
M3))

#with the Quadratic form distance the figure b is the most similar to
figure q
```

```
figure a from P to Quadratic form Q: 5.0
figure b from P to Quadratic form Q: 3.130495168499706
figure c from P to Quadratic form Q: 4.33589667773576
figure d from P to Quadratic form Q: 8.48528137423857
```

Exercise 4-3 Distances on a database

```
X = np.array([0, 1, 0, 1, 2, 3])
Y = np.array([1, 1, 1, 1, 2, 3])

distance = math.sqrt(sum([(a-b)**2 for a, b in zip(X, Y)]))
print("Euclidean distance from X to Y:", distance)
```

```
Euclidean distance from X to Y: 1.4142135623730951
```

Exercise 4-5 k-means 1-dimensional Example (1 point)

```
points = [2, 3, 4, 10, 11, 12, 20, 25, 30]
k = 3
u1 = 2
u2 = 4
u3 = 6
#First step is the assignment of the clusters. Each point is assigned to the clusters which is its nearest. The initial cluster assignment is given by:
c1 = [2,3]
c2 = [4]
c3 = [10, 11, 12, 20, 25, 30]

#Next step is the cluster update. The mean of the new clusters are updated by finding mean of the points present in it. For cluster 1,

cluster1 = sum(c1)/len(c1)
print(cluster1)

#cluster 2
cluster2 = sum(c2)/len(c2)
print(cluster2)

#cluster 3
cluster3 = sum(c3)/len(c3)
print(cluster3)

#This new means will be considered for cluster assignment in iteration 2. so cluster_i is the new 3 means

#the new clusters
c1_1 = [2,3]
c2_2 = [4,10,11]
c3_3 = [12,20,25,30]

#cluster 1_1
cluster1_1 = sum(c1_1)/len(c1_1)
print(cluster1_1)
```

```
#cluster 2_2
cluster1_2 = sum(c2_2)/len(c2_2)
print(cluster1_2)

# cluster 3_3
cluster1_3 = sum(c3_3)/len(c3_3)
print(cluster1_3)

#the next iteration
#the new clusters
c1_1_1 = [2,3,4]
c2_2_2 = [10, 11, 12]
c3_3_3 = [20,25,30]

#cluster 1_1_1
cluster1_1_1 = sum(c1_1_1)/len(c1_1_1)
print(cluster1_1_1)

#cluster 1_1_1
cluster2_2_2 = sum(c2_2_2)/len(c2_2_2)
print(cluster2_2_2)

#cluster 1_1_1
cluster3_3_3 = sum(c3_3_3)/len(c3_3_3)
print(cluster3_3_3)

#now convergence has been meet and the iterations stops. we get the
following 3 new centroids u1 = 3, u2 = 11 and u3 = 25
```

```
2.5
4.0
18.0
2.5
8.333333333333334
21.75
3.0
11.0
25.0
```

Exercise 5: Clustering: k-means and Silhouette

Exercise 5-1 Silhouette Coefficient (1 point)

```
centroid1x = np.array([4])
centroid1y = np.array([3.25])
centroid2x = np.array([6.75])
centroid2y = np.array([8])
clu1x = np.array([10, 2, 3, 1])
clu1y = np.array([1, 3, 4, 5])
clu2x = np.array([7, 6, 7, 7])
```

```python
clu2y = np.array([7, 8, 8, 9])

#med en funktion kan vi beregne a(o) og b(o) ud for hvert punkt
def get_total_distance(a, b, c, d):
    for i in b:
        for j in a:
            p = b - a
    for x in d:
        for y in c:
            q = d - c
    return np.array(np.sqrt((p*p) + (q*q)))

s_a = get_total_distance(clu1x, centroid1x, clu1y, centroid1y)
s_b = get_total_distance(clu1x, centroid2x, clu1y, centroid2y)
s_a2 = get_total_distance(clu2x, centroid2x, clu2y, centroid2y)
s_b2 = get_total_distance(clu2x, centroid1x, clu2y, centroid1y)

##herefter beregner vi hver s(o) og summere dem sammen for at få the
simplified silhoutte coefficient
tester2 = sum([(b-a)/max(a,b) for a, b in zip(s_a, s_b)])
tester3 = sum([(b-a)/max(a,b) for a, b in zip(s_a2, s_b2)])
print("The simplified silhoutte coefficient is:",(tester2+tester3)/8)
```

```
The simplified silhoutte coefficient is: 0.6937939026296913
```

```python
#datapoints in figure 1
centroid1x = np.array([4])
centroid1y = np.array([3.25])
centroid2x = np.array([6.75])
centroid2y = np.array([8])
clu1x = np.array([10, 2, 3, 1])
clu1y = np.array([1, 3, 4, 5])
clu2x = np.array([7, 6, 7, 7])
clu2y = np.array([7, 8, 8, 9])

#datapoints in figure 2
centroid1x_2 = np.array([10])
centroid1y_2 = np.array([1])
centroid2x_2 = np.array([4.75])
centroid2y_2 = np.array([6.3])
clu1x_2 = np.array([10])
clu1y_2 = np.array([1])
clu2x_2 = np.array([2, 3, 1, 7, 6, 7, 7])
clu2y_2 = np.array([3, 4, 5, 7, 8, 8, 9])


#datapoints in figure 3
centroid1x_3 = np.array([2])
centroid1y_3 = np.array([4])
centroid2x_3 = np.array([7.4])
```

```
centroid2y_3 = np.array([6.6])
clu1x_3 = np.array([2, 3, 1])
clu1y_3 = np.array([3, 4, 5])
clu2x_3 = np.array([10, 7, 6, 7, 7])
clu2y_3 = np.array([1, 7, 8, 8, 9])

#TD^2 for the 3 figures
def get_total_distance(a, b, c, d):
    for i in b:
        for j in a:
            p = b - a
    for x in d:
        for y in c:
            q = d - c
    return (p*p).sum() + (q*q).sum()

print("The measure of compactness for a cluster C aka Sum of squares
figure 1:", get_total_distance(clu1x, centroid1x, clu1y, centroid1y)
+get_total_distance(clu2x, centroid2x, clu2y, centroid2y))
print("The measure of compactness for a cluster C aka Sum of squares
figure 2:", get_total_distance(clu1x_2, centroid1x_2, clu1y_2,
centroid1y_2)+get_total_distance(clu2x_2, centroid2x_2, clu2y_2,
centroid2y_2))
print("The measure of compactness for a cluster C aka Sum of squares
figure 3:", get_total_distance(clu1x_3, centroid1x_3, clu1y_3,
centroid1y_3)+get_total_distance(clu2x_3, centroid2x_3, clu2y_3,
centroid2y_3))
```

```
The measure of compactness for a cluster C aka Sum of squares figure
1: 61.5
The measure of compactness for a cluster C aka Sum of squares figure
2: 72.8675
The measure of compactness for a cluster C aka Sum of squares figure
3: 54.400000000000006
```

```
##TIL MCQ 4

#for S1
clusterm1 = np.array([1, 3, 5])
clusterm2 = np.array([7, 10, 11, 12])
centroidm1 = np.array([np.sum(clusterm1)/ len(clusterm1)])
centroidm2 = np.array([np.sum(clusterm2)/ len(clusterm2)])

#for S2

s21 = np.array([1, 3])
s22 = np.array([5, 7])
s23 = np.array([10, 11, 12])
sct21 = np.array([np.sum(s21)/ len(s21)])
sct22 = np.array([np.sum(s22)/ len(s22)])
```

```
sct23 = np.array([np.sum(s23)/ len(s23)])

#for S3

s31 = np.array([1, 3, 5, 7])
s32 = np.array([10, 11, 12])
sct31 = np.array([np.sum(s31)/ len(s31)])
sct32 = np.array([np.sum(s32)/ len(s32)])

def get_total_distanceMSC(a, b):
    for i in b:
        for j in a:
            p = b - a
    return np.array(np.sqrt((p*p)))


#for S1
s_am = get_total_distanceMSC(clusterm1, centroidm1)
s_bm = get_total_distanceMSC(clusterm1, centroidm2)
s_am2 = get_total_distanceMSC(clusterm2, centroidm2)
s_bm2 = get_total_distanceMSC(clusterm2, centroidm1)

##herefter beregner vi hver s(o) og summere dem sammen for at få the
simplified silhoutte coefficient
testerm1 = sum([(b-a)/max(a,b) for a, b in zip(s_am, s_bm)])
testerm2 = sum([(b-a)/max(a,b) for a, b in zip(s_am2, s_bm2)])
print("The simplified silhoutte coefficient for S1 is:",
(testerm1+testerm2)/(len(clusterm1)+len(clusterm2)))

#for S2 #regn den ud manuelt svært, når der er tale om 3
s_am21 = get_total_distanceMSC(s21, sct21)
s_bm21 = get_total_distanceMSC(s21, sct22)
print(np.sum(s_bm21)/2) #pick this one in the code below its the
lowest
s_bm212 = get_total_distanceMSC(s21, sct23)
print(np.sum(s_bm212)/2)

s_am22 = get_total_distanceMSC(s22, sct22)
s_bm22 = get_total_distanceMSC(s22, sct21)
print(np.sum(s_bm22)/2) #pick this one in the code below its the
lowest
s_bm222 = get_total_distanceMSC(s22, sct23)
print(np.sum(s_bm222)/2)

s_am223 = get_total_distanceMSC(s23, sct23)
s_bm223 = get_total_distanceMSC(s23, sct21)
print(np.sum(s_bm223)/2)
s_bm2232 = get_total_distanceMSC(s23, sct22)
print(np.sum(s_bm2232)/2) #pick this one in the code below its the
```

```
lowest

##herefter beregner vi hver s(o) og summere dem sammen for at få the
simplified silhoutte coefficient
testerm12 = sum([(b-a)/max(a,b) for a, b in zip(s_am21, s_bm21)])
testerm22 = sum([(b-a)/max(a,b) for a, b in zip(s_am22, s_bm22)])
testerm233 = sum([(b-a)/max(a,b) for a, b in zip(s_am223, s_bm2232)])

print("The simplified silhoutte coefficient for S2 is:",
(testerm12+testerm22+testerm233)/(len(s21)+len(s22)+len(s23)))

#for S3
s_am3 = get_total_distanceMSC(s31, sct31)
s_bm3 = get_total_distanceMSC(s31, sct32)
s_am23 = get_total_distanceMSC(s32, sct32)
s_bm23 = get_total_distanceMSC(s32, sct31)

##herefter beregner vi hver s(o) og summere dem sammen for at få the
simplified silhoutte coefficient
testerm13 = sum([(b-a)/max(a,b) for a, b in zip(s_am3, s_bm3)])
testerm23 = sum([(b-a)/max(a,b) for a, b in zip(s_am23, s_bm23)])
print("The simplified silhoutte coefficient for S3 is:",
(testerm13+testerm23)/(len(s31)+len(s32)))
```

```
The simplified silhoutte coefficient for S1 is: 0.7543650793650792
4.0
9.0
4.0
5.0
13.5
7.5
The simplified silhoutte coefficient for S2 is: 0.7880952380952382
The simplified silhoutte coefficient for S3 is: 0.7666666666666667
```

```
#DEFINITION s = CLUSTER og sct = centroid (DETTE ER TIL MCQ3)

#for S1

clusterm1 = np.array([1, 3, 5])
clusterm2 = np.array([7, 10, 11, 12])
centroidm1 = np.array([np.sum(clusterm1)/ len(clusterm1)])
centroidm2 = np.array([np.sum(clusterm2)/ len(clusterm2)])

#for S2

s21 = np.array([1, 3])
s22 = np.array([5, 7])
s23 = np.array([10, 11, 12])
sct21 = np.array([np.sum(s21)/ len(s21)])
sct22 = np.array([np.sum(s22)/ len(s22)])
```

```python
sct23 = np.array([np.sum(s23)/ len(s23)])

#for S3

s31 = np.array([1, 3, 5, 7])
s32 = np.array([10, 11, 12])
sct31 = np.array([np.sum(s31)/ len(s31)])
sct32 = np.array([np.sum(s32)/ len(s32)])

#TD^2 for the 3 figures
def get_total_distanceMSQ(a, b):
    for i in b:
        for j in a:
            p = b - a
    return (p*p).sum()


print("TD1 for S1:", get_total_distanceMSQ(clusterm1, centroidm1) +
get_total_distanceMSQ(clusterm2, centroidm2))
print("TD2 for S2:", get_total_distanceMSQ(s21, sct21) +
get_total_distanceMSQ(s22, sct22) + get_total_distanceMSQ(s23, sct23))
print("TD3 for S3:", get_total_distanceMSQ(s31, sct31) +
get_total_distanceMSQ(s32, sct32))
```

```
TD1 for S1: 22.0
TD2 for S2: 6.0
TD3 for S3: 22.0
```