1)
a)

```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import numpy as np
```

```
scaler = StandardScaler()
```

```
data = np.array([[5.7,64], [4.7,58], [6.1,56], [4.6,64], [5.4,84], [4.9,60], [5.0,62], [6.4,62], [5.1,76], [6.0,60]]
scaled_data = preprocessing.scale(data)
print(np.round(scaled_data, 2))
the_mean = data.mean(axis=0)
the_std = data.std(axis=0)
```

```
[[ 0.52 -0.07]
 [-1.15 -0.8 ]
 [ 1.19 -1.04]
 [-1.32 -0.07]
 [ 0.02  2.35]
 [-0.82 -0.56]
 [-0.65 -0.32]
 [ 1.69 -0.32]
 [-0.49  1.38]
 [ 1.02 -0.56]]
```

b)
We scale the centroids.

```
: initial_centroids = [[5.6, 60], [5.9, 60], [5.2, 75]]
  scaled_centroids = [[(initial_centroids[pair][0]-the_mean[0])/the_std[0], (initial_centroids[pair][1]-the_mean[1])/t
   for pair in range(len(initial_centroids))]
  scaled_centroids = np.round(np.array(scaled_centroids),2)
  scaled_centroids
```

```
: array([[ 0.35, -0.56],
         [ 0.85, -0.56],
         [-0.32,  1.26]])
```

After one iteration we get.

```
: output = []
  for i in range(len(scaled_data)):
      temp = []
      for j in range(len(initial_centroids)):
          temp.append(round(manDist(scaled_data[i], scaled_centroids[j]),2))
      temp.append(min(temp))
      if(temp.index(min(temp)) == 0):
          temp.append("\u03bc1")
      elif(temp.index(min(temp)) == 1):
          temp.append("\u03bc2")
      else:
          temp.append("\u03bc3")
      output.append(temp)
  output
```

```
: [[0.66, 0.82, 2.17, 0.66, 'μ1'],
   [1.75, 2.25, 2.9, 1.75, 'μ1'],
   [1.32, 0.82, 3.81, 0.82, 'μ2'],
   [2.16, 2.66, 2.34, 2.16, 'μ1'],
   [3.25, 3.75, 1.43, 1.43, 'μ3'],
   [1.17, 1.67, 2.32, 1.17, 'μ1'],
   [1.25, 1.75, 1.91, 1.25, 'μ1'],
   [1.59, 1.09, 3.59, 1.09, 'μ2'],
   [2.78, 3.28, 0.29, 0.29, 'μ3'],
   [0.67, 0.17, 3.16, 0.17, 'μ2']]
```

```
: new_centroids = [(scaled_data[0] + scaled_data[1] + scaled_data[3] + scaled_data[5] + scaled_data[6])/5]
  new_centroids = np.append(new_centroids, [(scaled_data[2] + scaled_data[7] + scaled_data[9])/3], axis=0)
  new_centroids = np.append(new_centroids, [(scaled_data[4] + scaled_data[8])/2], axis=0)
  new_centroids = np.round(new_centroids, 2)
  print(new_centroids)
```

```
[[-0.69 -0.36]
 [ 1.3  -0.64]
 [-0.23  1.87]]
```

After second iteration we get.

```
output = []
for i in range(len(scaled_data)):
    temp = []
    for j in range(len(new_centroids)):
        temp.append(round(manDist(scaled_data[i], new_centroids[j]),2))
    temp.append(min(temp))
    if(temp.index(min(temp)) == 0):
        temp.append("\u03bc1")
    elif(temp.index(min(temp)) == 1):
        temp.append("\u03bc2")
    else:
        temp.append("\u03bc3")
    output.append(temp)
output
```

```
[[1.5, 1.35, 2.69, 1.35, 'μ2'],
 [0.91, 2.62, 3.6, 0.91, 'μ1'],
 [2.56, 0.51, 4.33, 0.51, 'μ2'],
 [0.92, 3.19, 3.04, 0.92, 'μ1'],
 [3.42, 4.28, 0.73, 0.73, 'μ3'],
 [0.33, 2.2, 3.02, 0.33, 'μ1'],
 [0.08, 2.28, 2.61, 0.08, 'μ1'],
 [2.43, 0.72, 4.11, 0.72, 'μ2'],
 [1.95, 3.81, 0.74, 0.74, 'μ3'],
 [1.91, 0.36, 3.68, 0.36, 'μ2']]
```

```
new_centroids2 = [(scaled_data[1] + scaled_data[3]+ scaled_data[5] + scaled_data[6])/4]
new_centroids2 = np.append(new_centroids2, [(scaled_data[0] + scaled_data[2] + scaled_data[7] + scaled_data[9])/4],
new_centroids2 = np.append(new_centroids2, [(scaled_data[4] + scaled_data[8])/2], axis=0)
new_centroids2 = np.round(new_centroids2, 2)
new_centroids2
```

```
array([[-0.99, -0.44],
       [ 1.1 , -0.5 ],
       [-0.23,  1.87]])
```

Center of second cluster [1.1, -0.5].

c)

```
final = []
for i in range(len(scaled_data)):
    temp = []
    for j in range(len(new_centroids2)):
        temp.append(round(manDist(scaled_data[i], new_centroids2[j]),2))
    temp.append(min(temp))
    if(temp.index(min(temp)) == 0):
        temp.append("\u03bc1")
    elif(temp.index(min(temp)) == 1):
        temp.append("\u03bc2")
    else:
        temp.append("\u03bc3")
    final.append(temp)
final
```

```
[[1.88, 1.01, 2.69, 1.01, 'μ2'],
 [0.53, 2.56, 3.6, 0.53, 'μ1'],
 [2.78, 0.63, 4.33, 0.63, 'μ2'],
 [0.7, 2.85, 3.04, 0.7, 'μ1'],
 [3.8, 3.94, 0.73, 0.73, 'μ3'],
 [0.29, 1.98, 3.02, 0.29, 'μ1'],
 [0.46, 1.94, 2.61, 0.46, 'μ1'],
 [2.81, 0.78, 4.11, 0.78, 'μ2'],
 [2.33, 3.47, 0.74, 0.74, 'μ3'],
 [2.13, 0.14, 3.68, 0.14, 'μ2']]
```

The points did not change, therefore converged. The centroids are the same as before.
So center of third cluster is [-0.23, 1.87].

d)
The points don't change after 3 iterations. Therefore, 3 iterations are required for cluster to converge.

2)
We pick the plot in each set where the points are assigned to the color of the cluster that is closest. The other plot in the set does not do that.
a) A2

b) B2
c) C2
d) D1
e) E2
f) F2

3)
a)
complete linkage (maximum distance)
two farthest points are (4.6,2.9) and (6.7,3.1).
$d = ((6.7 - 4.6)^2 + (3.1 - 2.9)^2)^{0.5} = 2.11$
You also see it is the highest from the python code below where I calculated distance between all points.

b)
single linkage (minimum distance)
two closest points are (5,3) and (5.9,3.2).
$d = ((5.9 - 5)^2 + (3.2 - 3)^2)^{0.5} = 0.92$
You also see it is the lowest rom the python code below where I calculated distance between all points.

c)
average link is 1.41

```
import math

red_points = [[4.7,3.2], [4.9,3.1], [5.0,3.0], [4.6,2.9]]
blue_points = [[5.9,3.2], [6.0,3.0], [6.7,3.1], [6.2,2.8]]
```

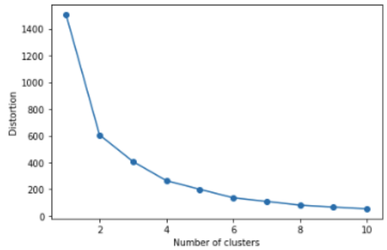
```
def calcDist(p1, p2):
    return math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )

result = []
for i in range(len(red_points)):
    temp = []
    for j in range(len(blue_points)):
        temp.append(red_points[i])
        temp.append(blue_points[j])
        temp.append(calcDist(red_points[i], blue_points[j]))
        result.append(temp)
        temp = []

total = 0
for i in range(len(result)):
    total = total + result[i][2]
print()
print("average linkage = " + str(round(total/16,2)))
result
```

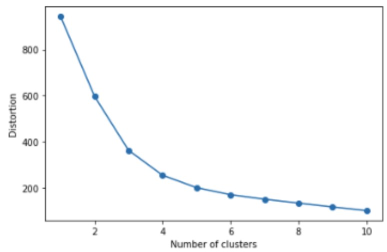
```
average linkage = 1.41

[[[4.7, 3.2], [5.9, 3.2], 1.2000000000000002],
 [[4.7, 3.2], [6.0, 3.0], 1.3152946437965904],
 [[4.7, 3.2], [6.7, 3.1], 2.0024984394500787],
 [[4.7, 3.2], [6.2, 2.8], 1.5524174696260025],
 [[4.9, 3.1], [5.9, 3.2], 1.004987562112089],
 [[4.9, 3.1], [6.0, 3.0], 1.1045361017187258],
 [[4.9, 3.1], [6.7, 3.1], 1.7999999999999998],
 [[4.9, 3.1], [6.2, 2.8], 1.3341664064126333],
 [[5.0, 3.0], [5.9, 3.2], 0.9219544457292891],
 [[5.0, 3.0], [6.0, 3.0], 1.0],
 [[5.0, 3.0], [6.7, 3.1], 1.7029386365926404],
 [[5.0, 3.0], [6.2, 2.8], 1.216552506059644],
 [[4.6, 2.9], [5.9, 3.2], 1.3341664064126342],
 [[4.6, 2.9], [6.0, 3.0], 1.4035668847618203],
 [[4.6, 2.9], [6.7, 3.1], 2.109502310972899],
 [[4.6, 2.9], [6.2, 2.8], 1.6031219541881403]]
```

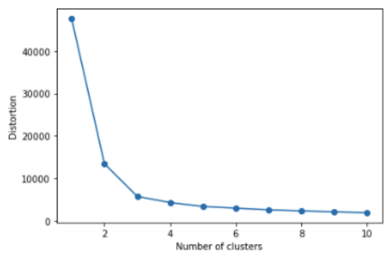
d)
the average link is clearly the most robust to noise, but it takes longer to compute.

4) (From Jupyter notebook)

```
# TODO :: run the kmeans on the other datasets and use elbow method to select the number of clusters.
plot_distortions(noisy_moons[0]) # 4 clusters
plot_distortions(noisy_circles[0]) # 3 clusters
plot_distortions(varied[0]) # 3 clusters
```







Question 1

Based on the code above, what is the difference between the two models? Which one performs better and why?

The difference between the two models is that the first model had the same color on different parts of the strip and the second does not have that. The second one with connectivity constraints performs better since all points of the same cluster are close together when it is unrolled.

Question 2

eps and min_samples are two important parameters for the DBSCAN model. What are those two parameters? Tune the parameters of the model for noisy_circles and noisy_moon dataset to make it separate the clusters perfectly.

eps is the threshold distance where the two points are considered neighbors. min_samples is minimum number of neighbors a given point should have in order to be classified as a core point.

for noisy_circles eps=0.2, min_samples=7 separates the clusters. for noisy_moon eps=0.3, min_samples=5 separates the clusters.

I used k means clustering on this data set I found. From the elbow method we see it's 3 or 4 clusters. I picked 3.

Question 4

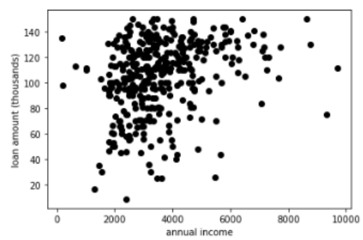
Apply any of these algorithms to your favorite dataset. Possible applications of clustering algorihtm will include but not be limited to image segmentation and outlier detection.

```
import pandas as pd
data = pd.read_csv('clustering.csv')
data
```

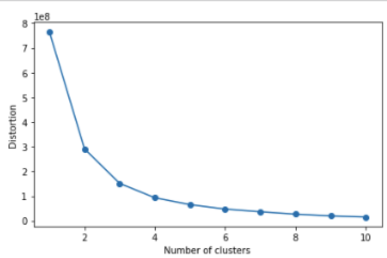
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
1	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
2	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
3	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
4	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
376	LP002953	Male	Yes	3+	Graduate	No	5703	0.0	128.0	360.0	1.0
377	LP002974	Male	Yes	0	Graduate	No	3232	1950.0	108.0	360.0	1.0
378	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
379	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
380	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

381 rows × 13 columns

```
the_data = data[["LoanAmount","ApplicantIncome"]]

plt.scatter(the_data["ApplicantIncome"],the_data ["LoanAmount"],c='black')
plt.xlabel('annual income')
plt.ylabel('loan amount (thousands)')
plt.show()
```




```
plot_distortions(the_data) #
```




```
km2 = KMeans(n_clusters=3,
             init='random',
             n_init=10,
             max_iter=300,
             tol=1e-04,
             random_state=0)
```

```
d1 = [coor[1] for coor in the_data.to_numpy()]
d2 = [coor[0] for coor in the_data.to_numpy()]
res = []
for i in range(len(the_data.to_numpy())):
       res.append([d1[i],d2[i]])
res
print_cluster(km2, 3, np.array(res))
```