> length(unique(user\_artists$userID))

[1] 1892

> length(unique(user\_artists$artistID))

[1] 17632

> length(unique(user\_taggedartists$userID))

[1] 1892

> length(unique(user\_taggedartists$artistID))

[1] 12523

> View(artists)

> user\_artists <- read.delim("G:/BITS/4-2/Information retrieval/my\_assignment/last\_fm\_dataset/user\_artists.dat")

> View(user\_artists)

> memory.size()

[1] 49.23

<https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/> - knn from scratch

<https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html> -> R and python code comparison

<https://github.com/letiantian/kmedoids/blob/master/kmedoids.py> ->kmedoids code

# 3 points in dataset

data = np.array([[1,1],

[2,2],

[10,10]])

# distance matrix

D = pairwise\_distances(data, metric='euclidean')

# split into 2 clusters

M , C= kMedoids(D, 2)

print('medoids:')

for point\_idx in M:

print( data[point\_idx] )

print('')

print('clustering result:')

for label in C:

for point\_idx in C[label]:

print('label {0}:　{1}'.format(label, data[point\_idx]))

<https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/>

Not using bfs since the graphs may be disconnected. And choosing weight even if I implement it, needs a research in itself.