Assignment 2.

This Assignment consist of 2 parts. First part related with KNN, second is K-Means Clusteing.

Part 1. KNN.

In this task you are going to predict text.csv with KNN algorithm. You will work with Breast Cancer Wisconsin (Diagnostic) Data Set. More info here.

The data set is already spited to the train and test files. On the moodle you can find 4 files(train.csv, test.csv, train_answers.csv, test_answers.csv). Please open the files and start to explore. train.csv and test.csv contains 31 columns, here is the description of the 10 columns:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)
etc………

train_answers.csv and test_answers.csv they labels for the train and test set. There is two columns id and Diagnosis (m=malignant, b=benign)

Dataset is Standardized no need to scale them. Do not touch the test_answers.csv at the beginning . You will use it at the final step, when you start to calculate the accuracy score.

Step to do:

Please open the files

```
import pandas as pd

train = pd.read_csv("train.csv", index_col='Unnamed: 0')
y = pd.read_csv("train_answers.csv", index_col='Unnamed: 0')
test = pd.read_csv("test.csv", index_col='Unnamed: 0')
answers = pd.read_csv("real_answers_for_test.csv", index_col='Unnamed: 0')
```

Look at their sizes with data.shape. How many features we have? What is the sizes of train, test, y and answers?

Start to predicting test with help of train and y. NOTE: you will not implement KNN algorithm from packages. For the calculating the Euclidean distance you can use numpy.linalg.norm(a-b).

Example of predicting first row of test with one neighbor:

I. Calculate the distance to all train set.

```
# record the distance
distLst = []
for i in range(len(train)):
    #calcualte the distance
    distance = np.linalg.norm(test.iloc[0]-train.iloc[i])
    # record the distance
    distLst.append(distance)
```

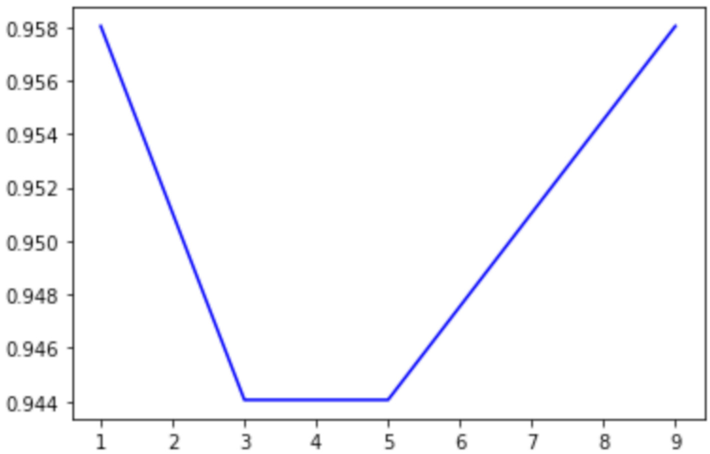
II. Find the Minimum distance and predict to according it's label.

```
print("Minimum distance: ", min(distLst))
print("The index of the record with minimum dist: ", distLst.index(min(distLst)))
print("The label at that index: ", y.iloc[13])
```

```
Minimum distance:  3.06778195989615
The index of the record with minimum dist:  13
The label at that index:  Diagnosis    B
```

In that example above we predicted that first example of test is B (benign).

Your task is to predict for all other records of test with 1, 3, 5, 6, 7, 9 nearest neighbors. Calculate the accuracy score (without using package) for each neighbors result with test_answers.csv. and plot the graph. Accuracy VS number of the neighbors.



This the the result that you should get finally.

Part 2. K-means.

In this part you will create two functions. We know that we have two important steps in iteration:
1. Finding the closest centroid (assign it)
2. Compute the mean. (recomputing at each iteration)

Algorithm 1 k-means algorithm

Specify the number k of clusters to assign.
Randomly initialize k centroids.
repeat
  expectation: Assign each point to its closest centroid.
  maximization: Compute the new centroid (mean) of each cluster.
until The centroid positions do not change.

```
for i in range(iterations):

    idx = findClosestCentroids(X, centroids)

    centroids = computeMeans(X, idx, K)
```

Your task is to create function findClosestCentroids.

```python
def findClosestCentroids(X, centroids):
    """
```

X is data matrix, centroids it is location of all centroids. Output of should be array that holds the index (a value in {1,…,*K*} K is total number of centroids) of the closest centroid to every training example.

This is the result of function :

```python
# Load an example dataset that we will be using
data = loadmat('ex7data2.mat')
X = data['X']
initial_centroids = np.array([[3, 3], [6, 2], [8, 5]])

# Find the closest centroids for the examples using the initial_centroids
idx = findClosestCentroids(X, initial_centroids)

print('Closest centroids for the first 3 examples:')
print(idx[:3])
print('(the closest centroids should be 0, 2, 1 respectively)')
```

```
Closest centroids for the first 3 examples:
[[0]
 [2]
 [1]]
(the closest centroids should be 0, 2, 1 respectively)
```

Next task is is to create function computeCentroids.

```python
def computeCentroids(X, idx, K):
```

idx is the A vector (size m) of centroid assignments (i.e. each entry in range [0 ... K-1]) example. K is the number of clusters. and you should return centroids A matrix of size (K, n), n is the number of features.

This is the result of function :

```python
centroids = computeCentroids(X, idx, K)

print('Centroids computed after initial finding of closest centroids:')
print(centroids)
print('\nThe centroids should be')
print('   [ 2.428301 3.157924 ]')
print('   [ 5.813503 2.633656 ]')
print('   [ 7.119387 3.616684 ]')
```

```
Centroids computed after initial finding of closest centroids:
[[2.42830111 3.15792418]
 [5.81350331 2.63365645]
 [7.11938687 3.6166844 ]]

The centroids should be
   [ 2.428301 3.157924 ]
   [ 5.813503 2.633656 ]
   [ 7.119387 3.616684 ]
```

If you correctly completed that two function above,
please run the following code

```
from scipy.io import loadmat
import os
import numpy as np

data = loadmat(os.path.join('Data', 'ex7data2.mat'))
X = data['X']
max_iters = 10
initial_centroids = np.array([[3, 3], [6, 2], [8, 5]])


for i in range(max_iters):
        idx = findClosestCentroids(X, centroids)
        centroids = computeCentroids(X, idx, K)
```

And plot the results.