

Artificial Intelligence

K-Means Clustering

Name : Ziad Ashraf Shreet
ID : 3947

K-means clustering → is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K . The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

Objective:

Machine learning is to derive techniques for unsupervised learning on data. This kind of data analysis is very helpful in many applications that require classification of data, such as identifying cancerous cells within a large sample, clustering words with similar definitions for better search engine accuracy, identifying outliers in student's academic performance for better refinement of habits, or even for detecting landmines in a battlefield.

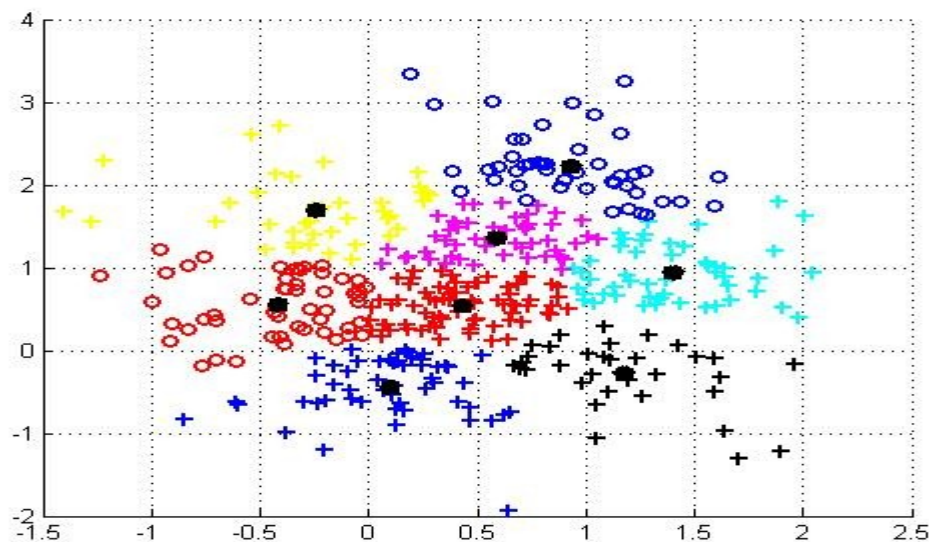
The results of the K-means clustering algorithm are:

1. The centroids of the K clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have

formed organically. The "Choosing K" section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.



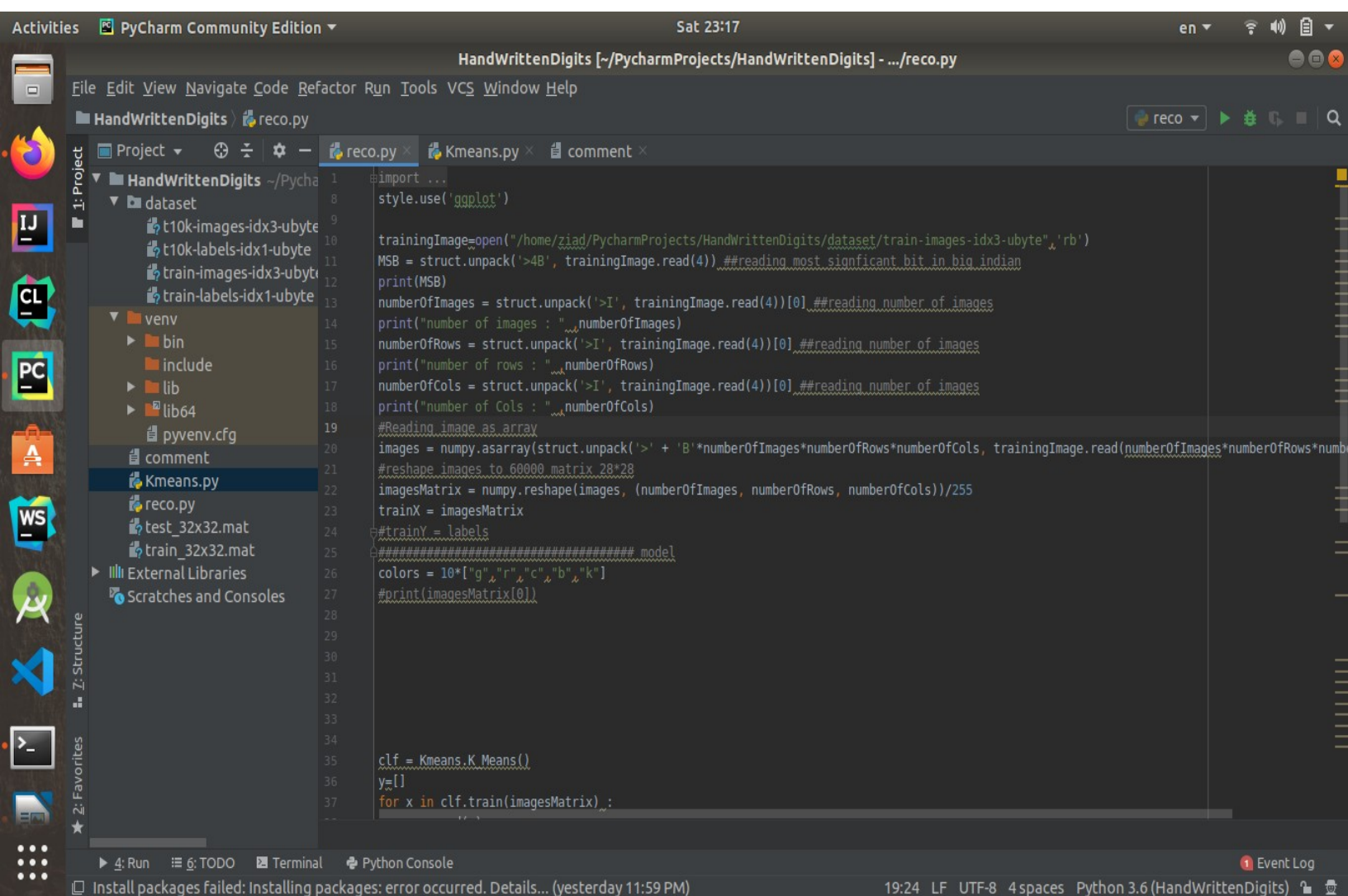
K-means algorithm iteratively minimizes the distances between every data point and its centroid in order to find the most optimal solution for all the data points.

1. Random points of the data-set are chosen to be the initial centroids.
2. Distances between every data point and the centroids are calculated and stored.
3. Based on distance calculates, each point is assigned to the nearest cluster
4. New cluster centroid positions are updated: similar to finding the mean
5. If the centroid locations changed, the process repeats from step 2, until the calculated new center stays the same, which signals that the clusters' members and centroids are now set and will be constant movement.

I chose the *MINIST Handwritten Digits Data-Set*

The Data-Set is a batch of 60,000 images

First I began with extracting and loading the Data-set file to nd-array and reshaped it into Matrix then adjusting the attributes to be in the interval [0,1].



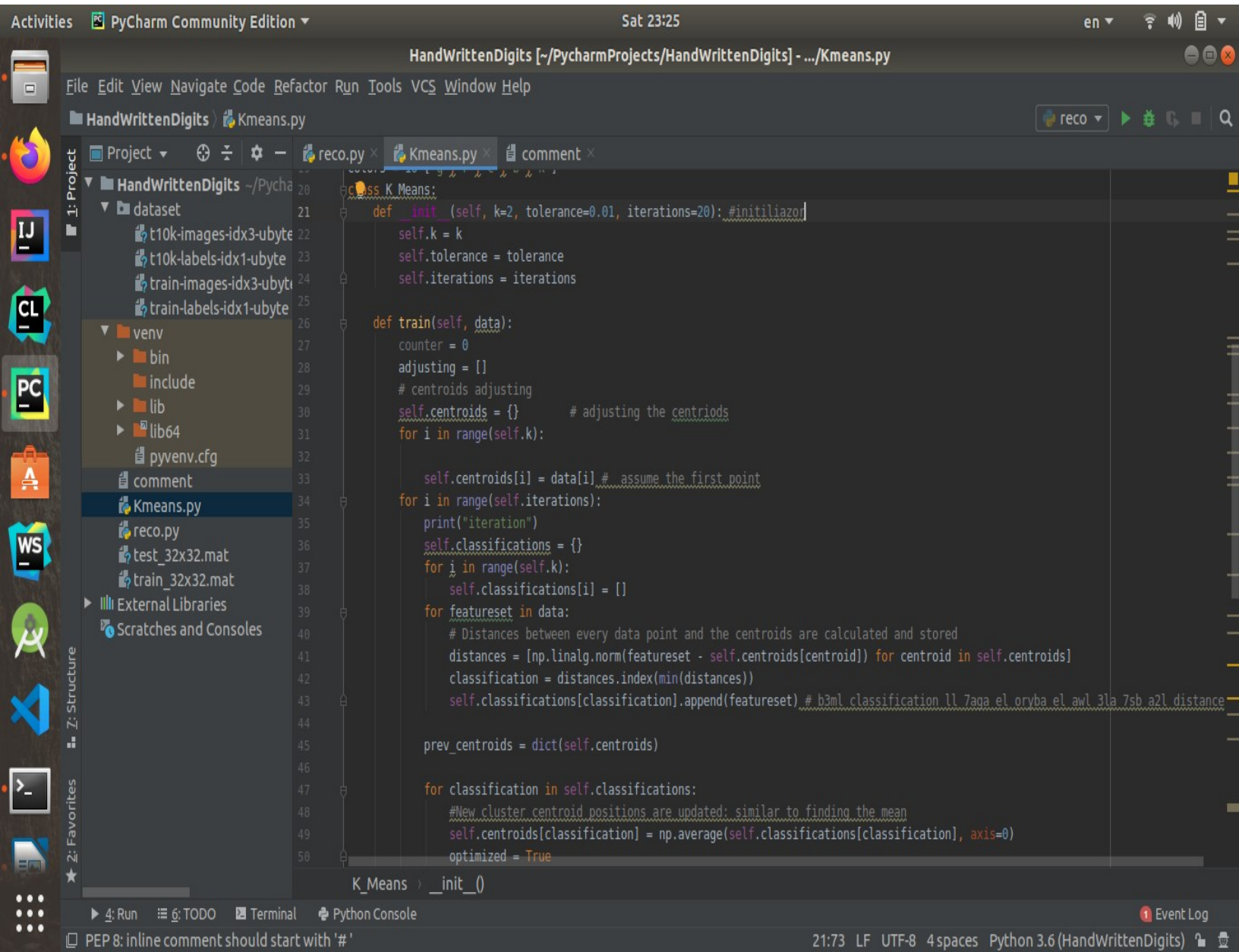
The screenshot displays the PyCharm Community Edition interface. The left sidebar shows the project structure for 'HandWrittenDigits', including a 'dataset' folder with training and test image and label files, and a 'venv' folder. The main editor window shows the 'reco.py' file with the following code:

```
1 import ...
2 style.use('ggplot')
3
4 trainingImage=open("/home/ziad/PycharmProjects/HandWrittenDigits/dataset/train-images-idx3-ubyte", 'rb')
5 MSB = struct.unpack('>4B', trainingImage.read(4)) ##reading most significant bit in big indian
6 print(MSB)
7 numberOfImages = struct.unpack('>I', trainingImage.read(4))[0] ##reading number of images
8 print("number of images : ", numberOfImages)
9 numberOfRows = struct.unpack('>I', trainingImage.read(4))[0] ##reading number of images
10 print("number of rows : ", numberOfRows)
11 numberOfCols = struct.unpack('>I', trainingImage.read(4))[0] ##reading number of images
12 print("number of Cols : ", numberOfCols)
13
14 ##Reading image as array
15 images = numpy.asarray(struct.unpack('>' + 'B'*numberOfImages*numberOfRows*numberOfCols, trainingImage.read(numberOfImages*numberOfRows*numberOfCols)))
16 ##reshape images to 60000 matrix 28*28
17 imagesMatrix = numpy.reshape(images, (numberOfImages, numberOfRows, numberOfCols))/255
18 trainX = imagesMatrix
19
20 #trainY = labels
21
22 ##### model
23 colors = 10*["g", "r", "c", "b", "k"]
24 #print(imagesMatrix[0])
25
26
27
28
29
30
31
32
33
34
35 clf = Kmeans.K Means()
36 y=[]
37 for x in clf.train(imagesMatrix):
```

The bottom status bar indicates the current configuration: 4: Run, 6: TODO, Terminal, Python Console, 19:24, LF, UTF-8, 4 spaces, Python 3.6 (HandWrittenDigits).

I used `struct.unpack()` to *unpack the* data from strings using format specifiers made up of characters representing the type of the data and optional count and endianness indicators.

K-Means Implementation with comments and the steps are the same as I explained in the introduction :



```
class K_Means:
    def __init__(self, k=2, tolerance=0.01, iterations=20):
        self.k = k
        self.tolerance = tolerance
        self.iterations = iterations

    def train(self, data):
        counter = 0
        adjusting = []
        # centroids adjusting
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i] # assume the first point

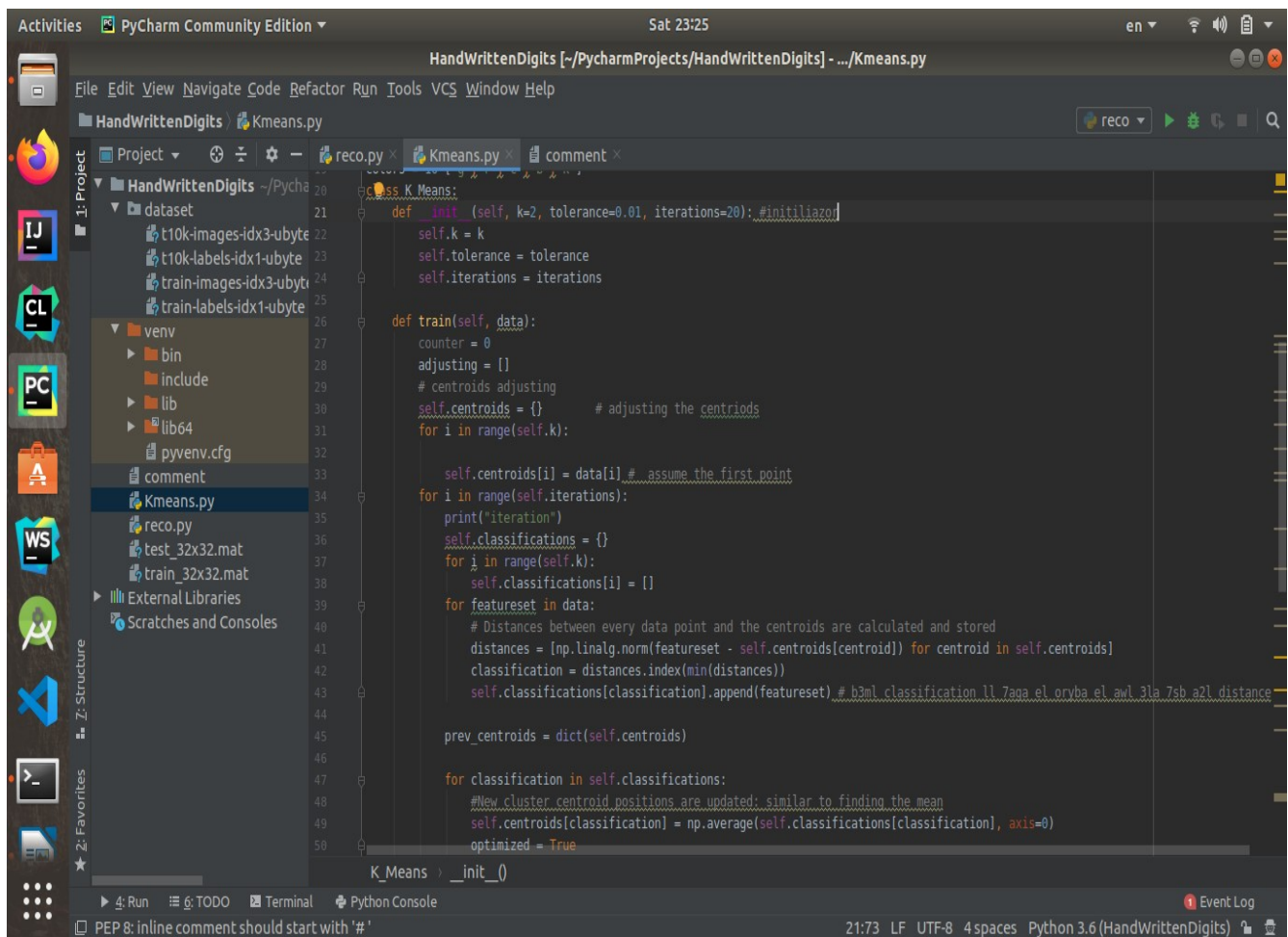
        for i in range(self.iterations):
            print("iteration")
            self.classifications = {}
            for j in range(self.k):
                self.classifications[j] = []
            for featureset in data:
                # Distances between every data point and the centroids are calculated and stored
                distances = [np.linalg.norm(featureset - self.centroids[centroid]) for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classifications[classification].append(featureset) # b3ml classification ll 7aga el oryba el awl 3la 7sb a2l distance

            prev_centroids = dict(self.centroids)

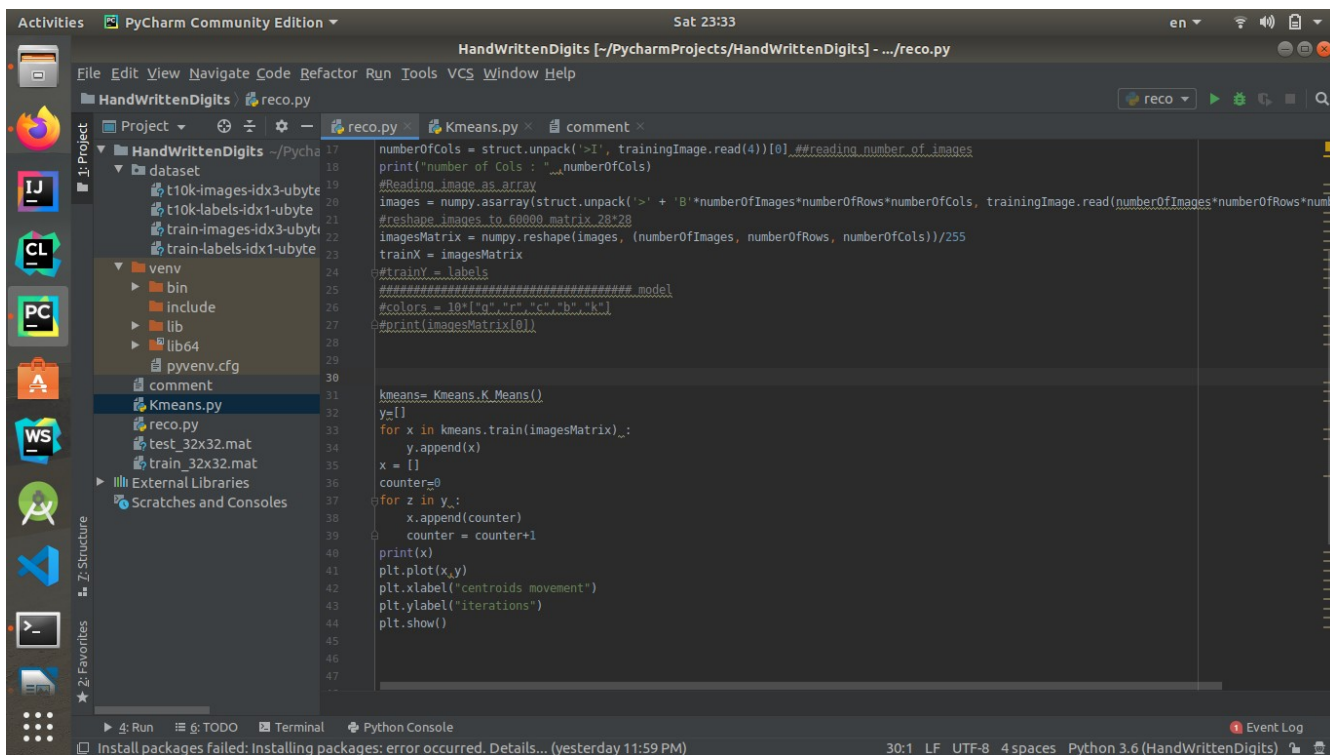
            for classification in self.classifications:
                # New cluster centroid positions are updated: similar to finding the mean
                self.centroids[classification] = np.average(self.classifications[classification], axis=0)

            optimized = True
```

PEP 8: inline comment should start with '#'

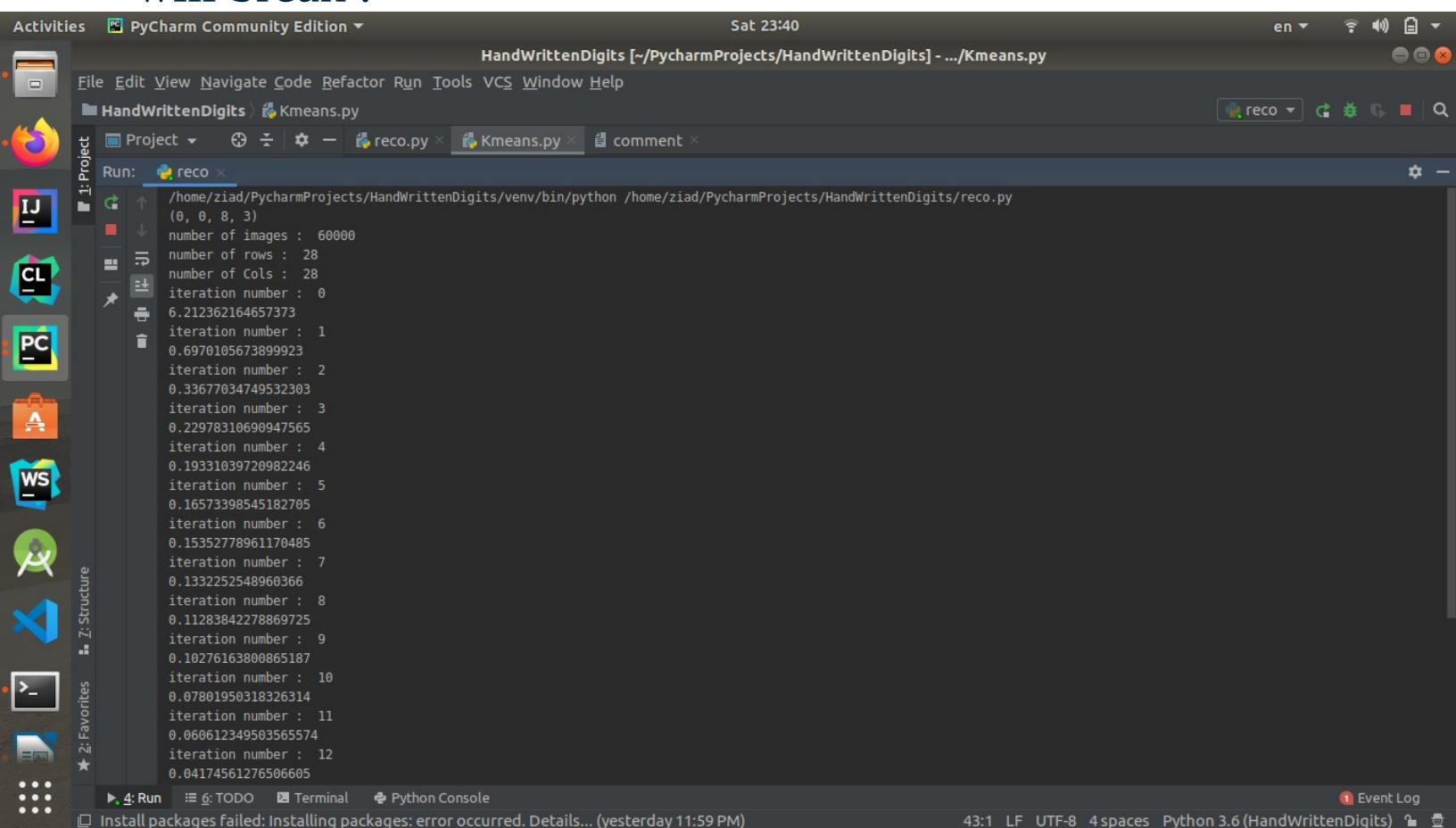


First I tested the algorithm for small data-set then I tested it for the Big Data-set:



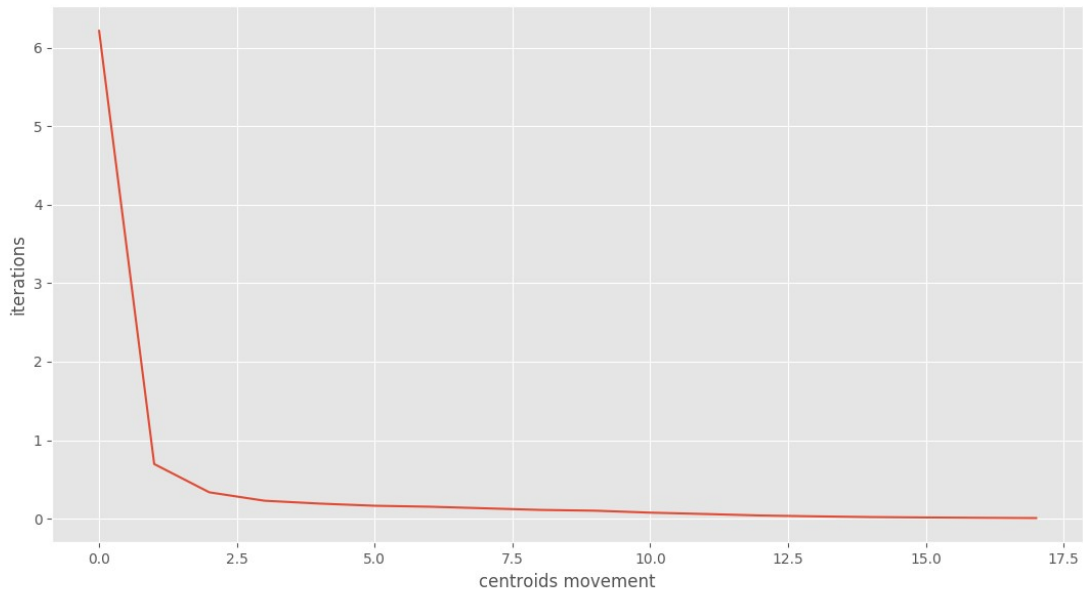
for k = 2 :

choose random 2 images to be the initial centriods then calculate the distances from the images to the centriods the nearest centriod will be the cluster that the image will belong to if it exceeds the tolerance the algorithm will break .

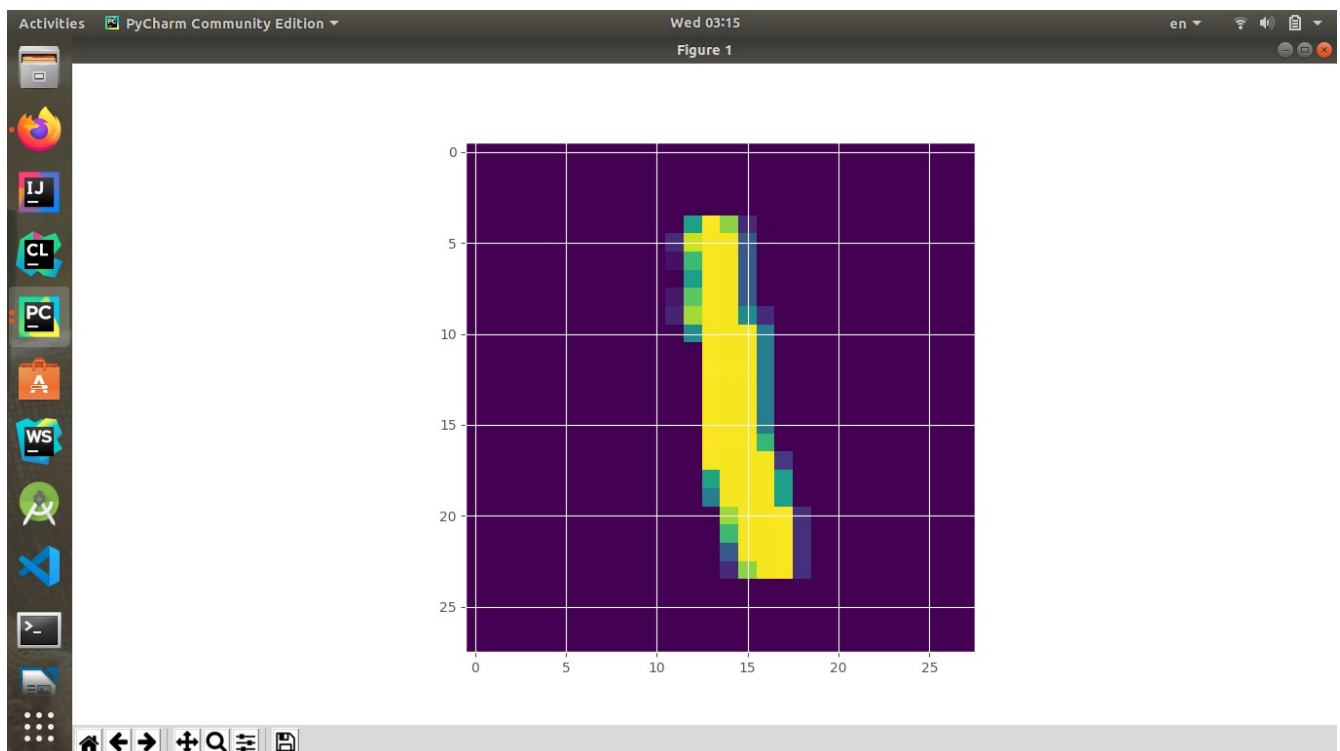


```
HandWrittenDigits [-/PycharmProjects/HandWrittenDigits] - .../Kmeans.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
HandWrittenDigits Kmeans.py
Project reco.py Kmeans.py comment x
Run: reco x
/home/ziad/PycharmProjects/HandWrittenDigits/venv/bin/python /home/ziad/PycharmProjects/HandWrittenDigits/reco.py
(0, 0, 8, 3)
number of images : 60000
number of rows : 28
number of Cols : 28
iteration number : 0
6.212362164657373
iteration number : 1
0.6970105673899923
iteration number : 2
0.33677034749532303
iteration number : 3
0.22978310690947565
iteration number : 4
0.19331039720982246
iteration number : 5
0.16573398545182705
iteration number : 6
0.15352778961170485
iteration number : 7
0.1332252548960366
iteration number : 8
0.11283842278869725
iteration number : 9
0.10276163800865187
iteration number : 10
0.07801950318326314
iteration number : 11
0.060612349503565574
iteration number : 12
0.04174561276506605
4: Run 6: TODO Terminal Python Console
Install packages failed: Installing packages: error occurred. Details... (yesterday 11:59 PM) 43:1 LF UTF-8 4 spaces Python 3.6 (HandWrittenDigits) Event Log
```

Plot the K-Means objective function (distortion measure) as a function of iteration and verify that it never increases.
The objective function :



some representative images :



for $k = 4$:

The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for Activities, PyCharm Community Edition, and the current date and time (Sat 23:54). The main window displays the file `HandWrittenDigits` with the `Kmeans.py` file open. The code in `Kmeans.py` is as follows:

```
1 import matplotlib.pyplot as plt
2 from matplotlib import style
3 style.use('ggplot')
4 import numpy as np
5
6 K_Means = KMeans
7
8 def __init__(self):
9     pass
```

The left sidebar shows the Project view with the following structure:

- HandWrittenDigits
 - dataset
 - t10k-images-idx3-ubyte
 - t10k-labels-idx1-ubyte
 - train-images-idx3-ubyte
 - train-labels-idx1-ubyte

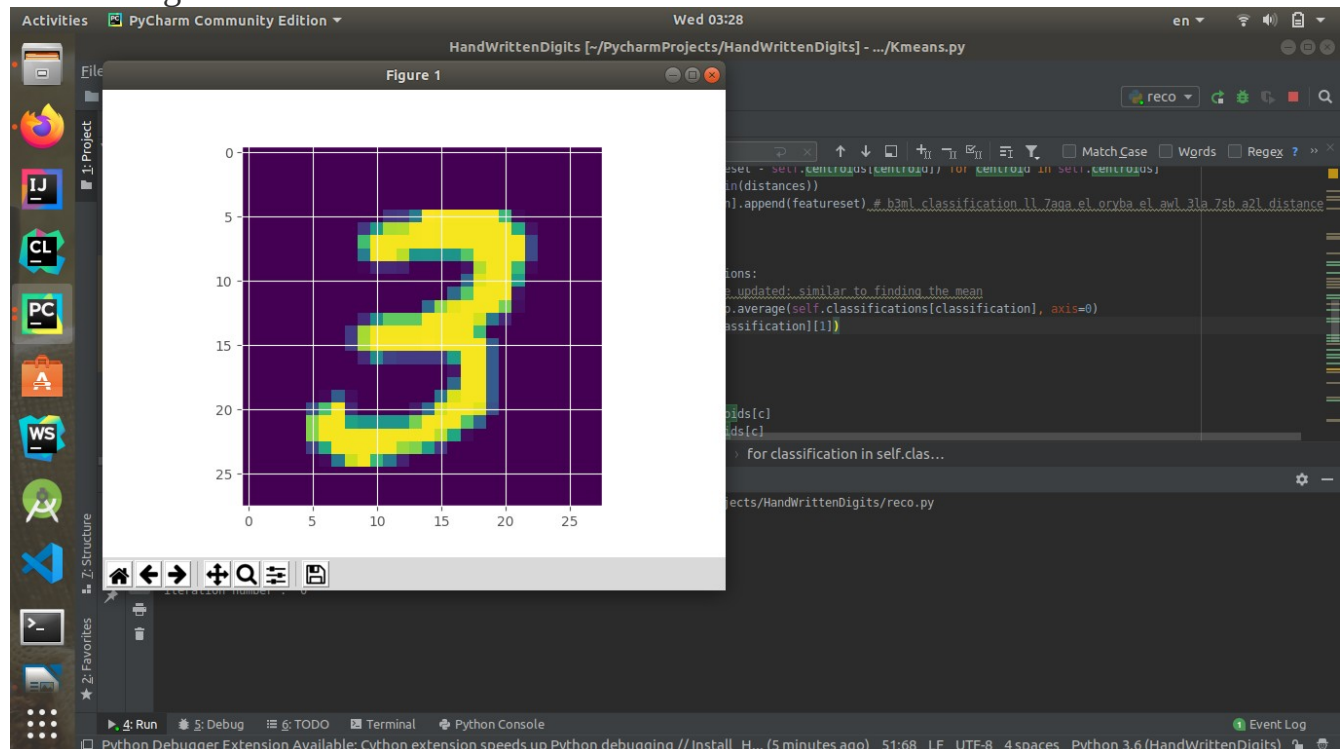
The bottom pane shows the Run output for the `reco` configuration. The output is as follows:

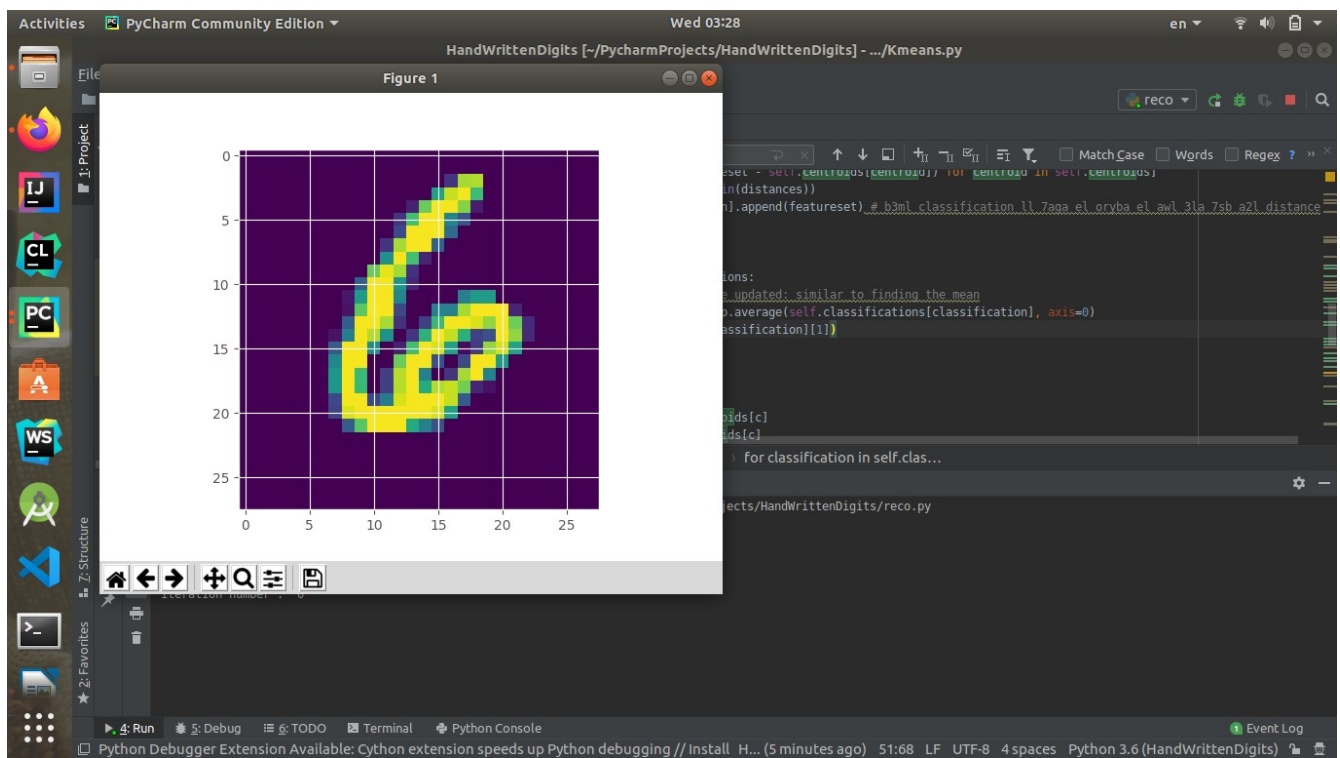
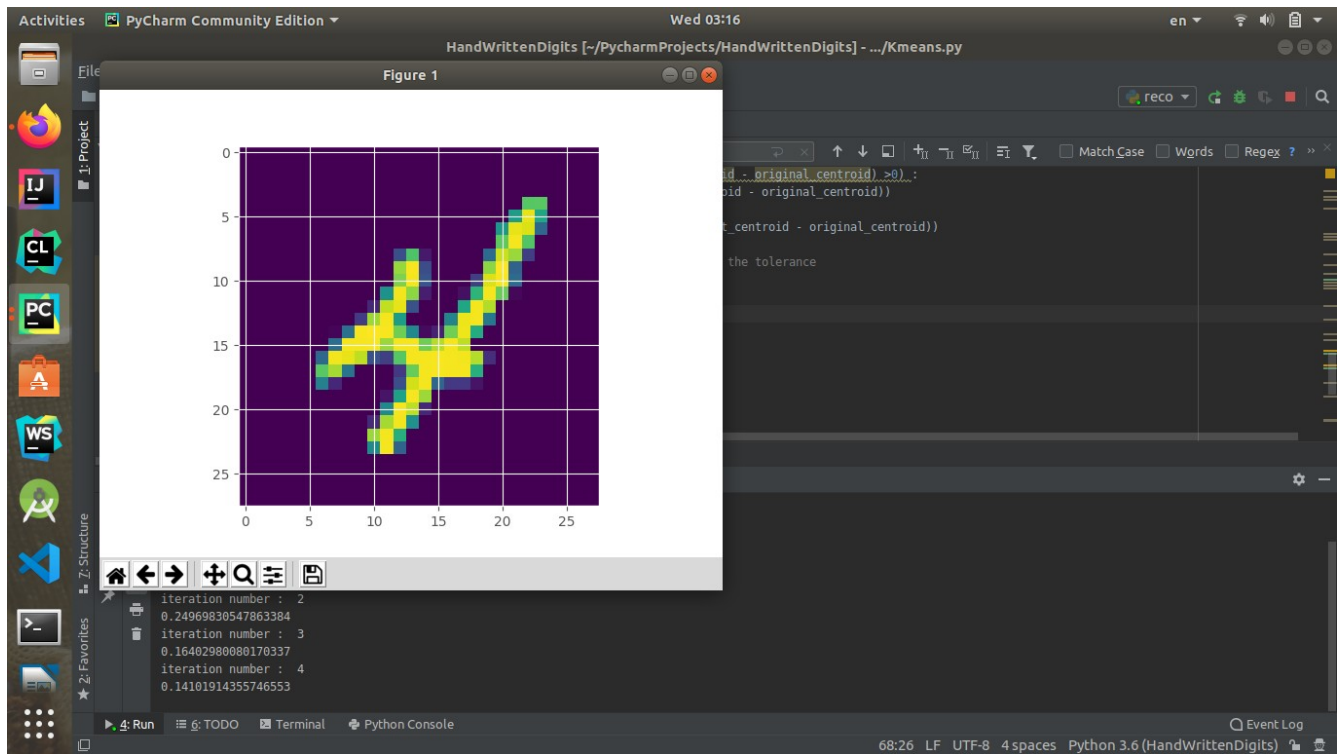
```
/home/ziad/PycharmProjects/HandWrittenDigits/venv/bin/python /home/ziad/PycharmProjects/HandWrittenDigits/reco.py
(0, 0, 0, 3)
number of images : 60000
number of rows : 28
number of Cols : 28
iteration number : 0
5.421926596397055
iteration number : 1
0.778241383638248
iteration number : 2
0.41301664991405124
iteration number : 3
0.178607283961256
iteration number : 4
0.08639967089975004
iteration number : 5
0.04549826443215987
iteration number : 6
0.07491545851974624
iteration number : 7
0.1150889032420232
iteration number : 8
0.12999948629669036
iteration number : 9
0.12770000000000000
```

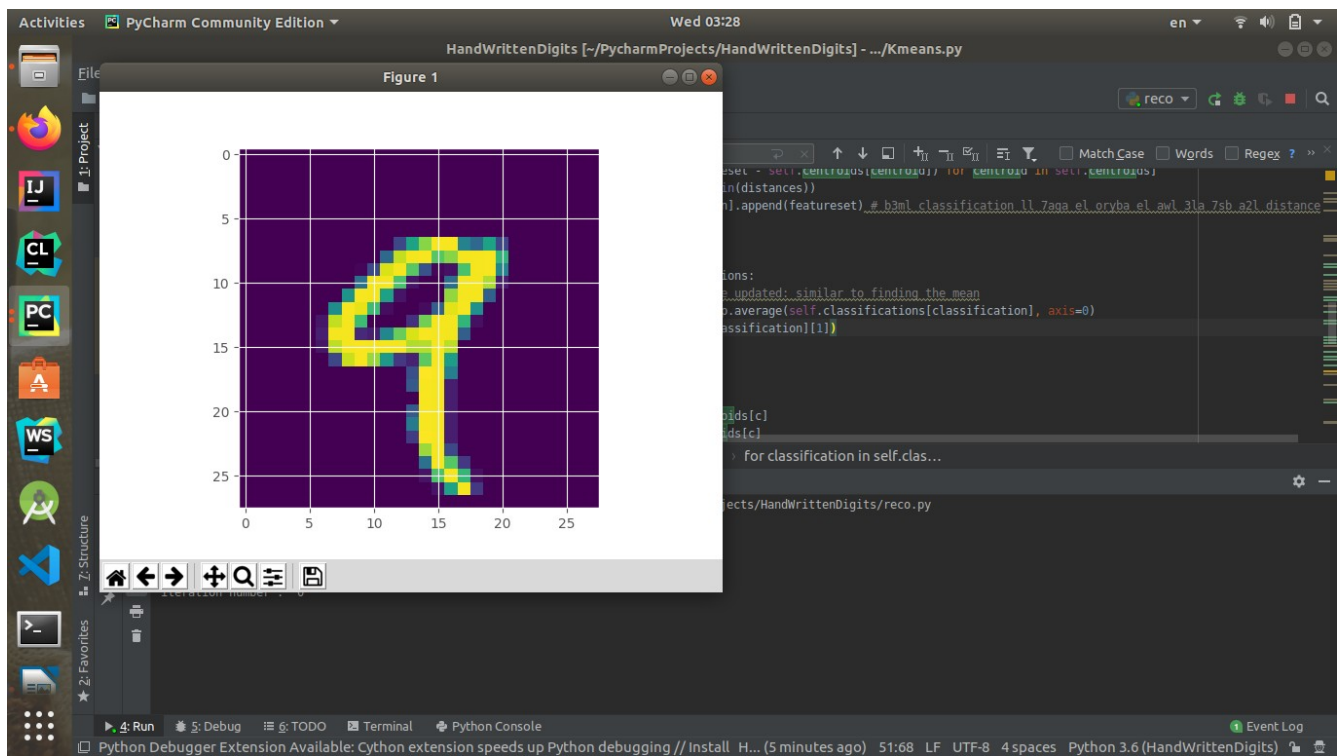
The bottom status bar shows the Run configuration (Run, TODO, Terminal, Python Console) and the Event Log.

Train Accuracy = 0.4719, Test Accuracy = 0.4632

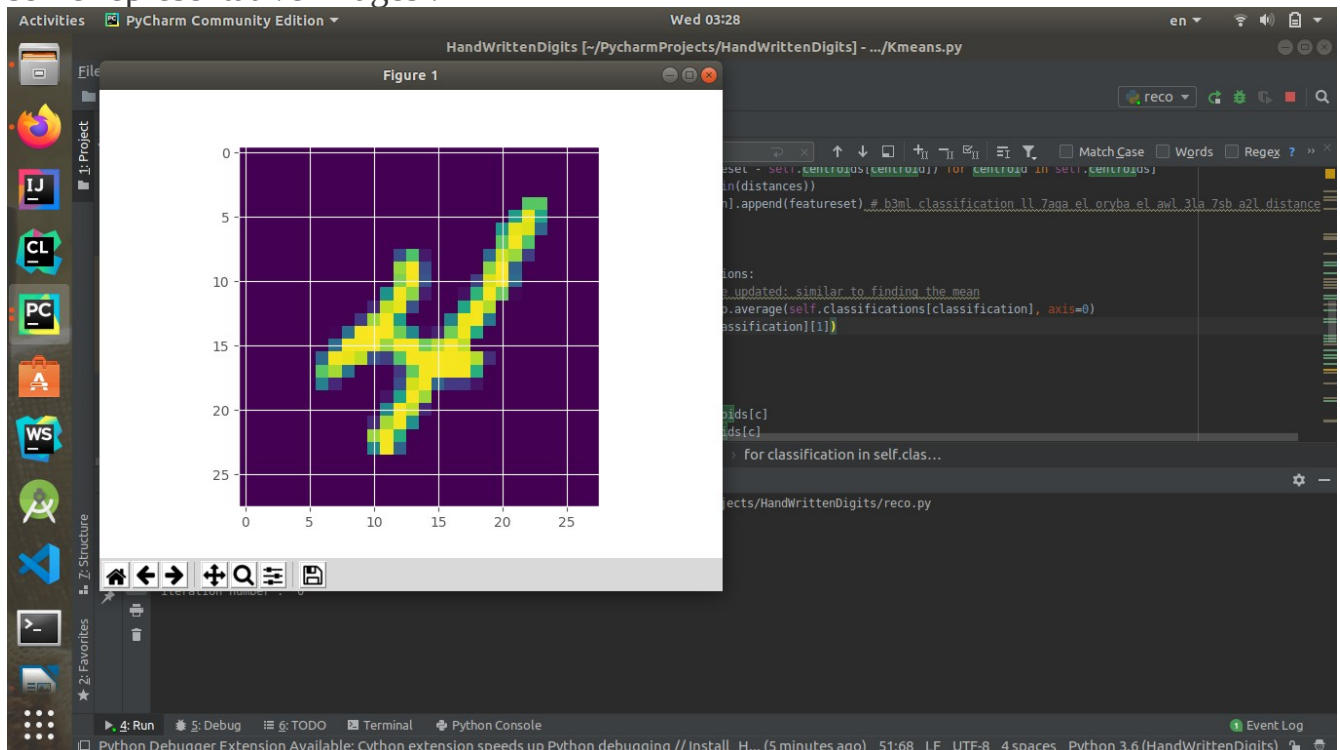
Mean Images :

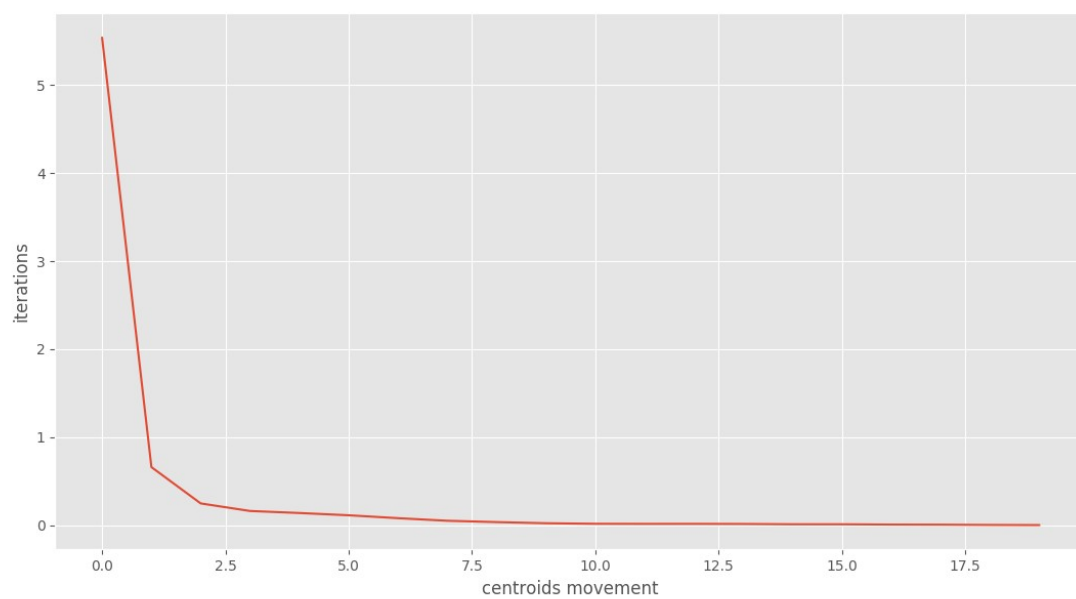




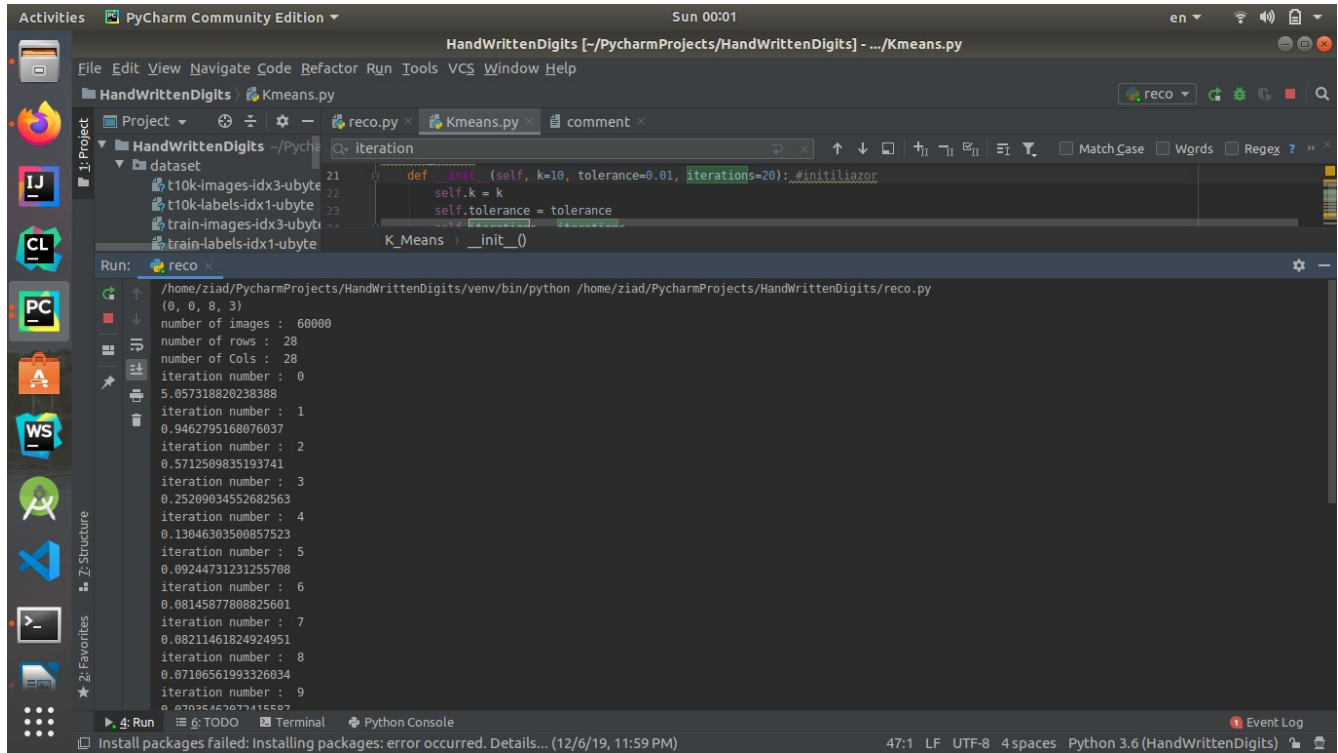


some representative images :





for $K = 10$:



The screenshot shows the PyCharm Community Edition interface. The main editor displays the `Kmeans.py` file with the following code:

```
def __init__(self, k=10, tolerance=0.01, iterations=20): #initilazor
    self.k = k
    self.tolerance = tolerance
    self.iterations = iterations
    self._init_()
```

The Run console shows the output of the program:

```
/home/ziad/PycharmProjects/HandWrittenDigits/venv/bin/python /home/ziad/PycharmProjects/HandWrittenDigits/reco.py
(0, 0, 8, 3)
number of images : 60000
number of rows : 28
number of Cols : 28
iteration number : 0
5.057318820238388
iteration number : 1
0.9462795168076037
iteration number : 2
0.5712509835193741
iteration number : 3
0.25209034552682563
iteration number : 4
0.13046303500857523
iteration number : 5
0.09244731231255708
iteration number : 6
0.08145877808825601
iteration number : 7
0.08211461824924951
iteration number : 8
0.07106561993326034
iteration number : 9
0.07035462072415597
```

Train Accuracy = 0.5193, Test Accuracy = 0.5092

The objective function :

