Wisconsin Breast Cancer Data Project (K-means clustering)

By Lionel Ngobesing Alangeh

1.0) Introduction

K-Means Clustering is an unsupervised machine learning algorithm which basically means we will just have input, not the corresponding output label. In this article, we will see it's implementation using python.

K Means Clustering tries to cluster your data into clusters based on their similarity. In this algorithm, we have to specify the number of clusters (which is a hyperparameter) we want the data to be grouped into. Hyperparameters are the variables whose value need to be set before applying value to the dataset. Hyperparameters are adjustable parameters you choose to train a model that carries out the training process itself.

2.0) K-means algorithm

The algorithm starts with an initial set of cluster centers, chosen at random or according to some heuristic procedure. In each iteration, each instance is assigned to its nearest cluster center according to the Euclidean distance between the two. Then the cluster centers are re-calculated. The center of each cluster is calculated as the mean of all the instances belonging to that cluster:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q$$

Where Nk is the number of instances belonging to cluster k and ¹k is the mean of the cluster k. A number of convergence conditions are possible. For example, the search may stop when the partitioning error is not reduced by the relocation of the centers. This indicates that the present partition is locally optimal. Other stopping criteria can be used also such as exceeding a pre-defined number of iterations.

The algorithm is as follows:

```
Input: $S$ (instance set), $K$ (number of cluster)
Output: clusters
1: Initialize $K$ cluster centers.
2: while termination condition is not satisfied do
3:     Assign instances to the closest cluster center.
4:     Update cluster centers based on the assignment.
5: end while
```

The K-means algorithm may be viewed as a gradient-decent procedure, which begins with an initial set of K cluster-centers and iteratively updates it so as to decrease the error function. A rigorous proof of the finite convergence of the K-means type algorithms is given in (Selim and Ismail, 1984). The complexity of T iterations of the K-means algorithm performed on a sample size of m instances, each characterized by N attributes, is: O(T * K * m * N). This linear complexity is one of the reasons for the popularity of the K-means algorithms.

3.0) Wisconsin Breast Cancer Data Project (k-means kümeleme uygulanması)

3.1) Proje Tanımı

This project implements the K-means clustering algorithm on UCI machine learning dataset for classifying Benign and Malign cells using Python and Machine learning.

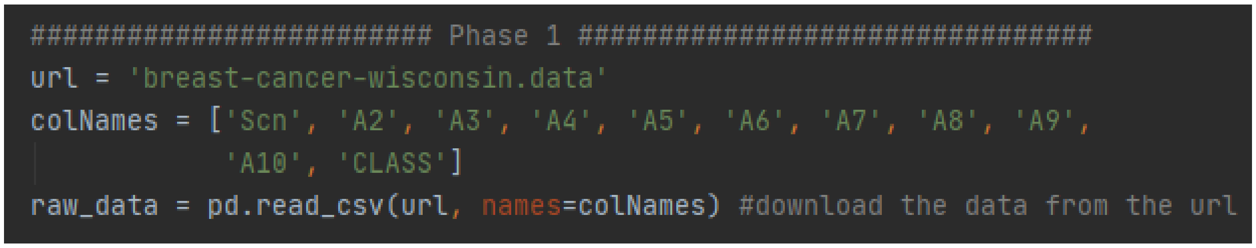
The project is divided into 3 phases:

i. Phase 1 – Data Analysis

The first phase is basically the data analysis phase where the data is gotten and using various big data visualization tools is analyzed. In this case we use the histogram to analyze the data. The data analysis phase involves calculating the median, the mean, the standard deviation and variance of each of the 9 attributes of each data element in the dataset (Labelled from A2 to A10). The different attributes for breast cancer are: Clump thickness, Uniformity of Cell size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell size, Bare Nuclei, Bland Chromatin, Normal Nucleoli and mitosis.

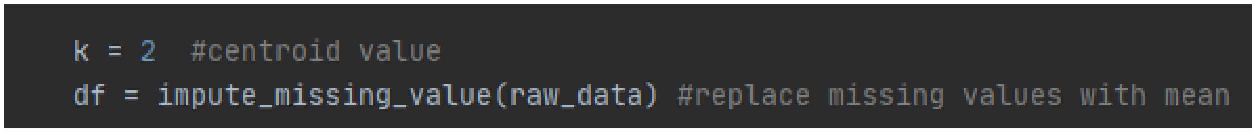
The code is implemented as follows:

- Get the dataset data from breast-cancer-wisconsin.data file and load the dataset into python using pandas library.



```
######################### Phase 1 ################################
url = 'breast-cancer-wisconsin.data'
colNames = ['Scn', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9',
            'A10', 'CLASS']
raw_data = pd.read_csv(url, names=colNames) #download the data from the url
```

- Impute missing values using mean imputation techniques

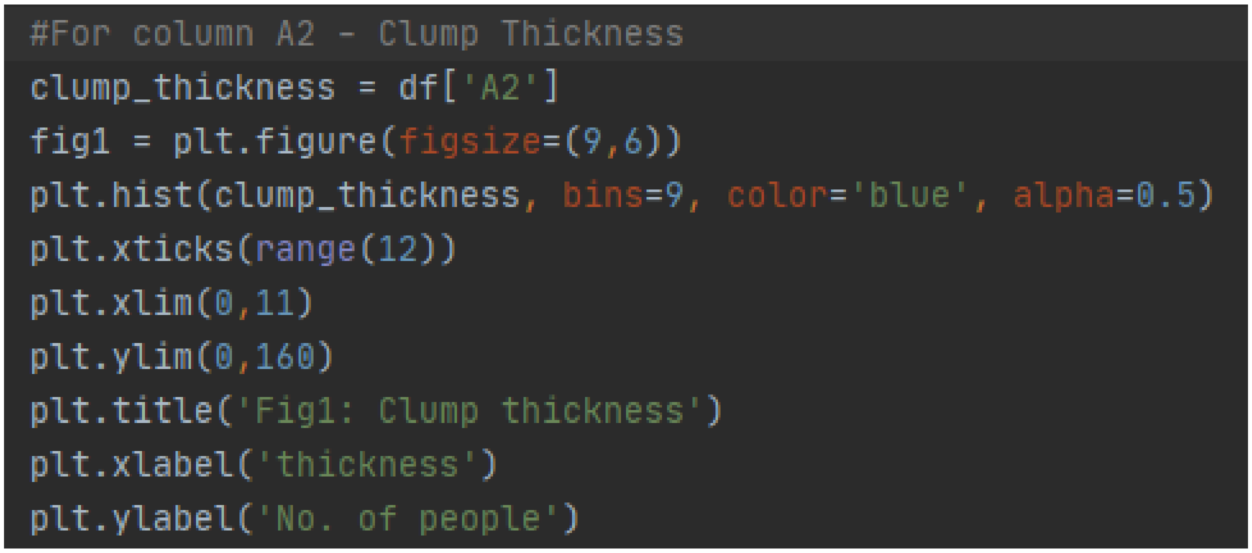


```
  k = 2  #centroid value
  df = impute_missing_value(raw_data) #replace missing values with mean
```

- Plot histograms using matplotlib.

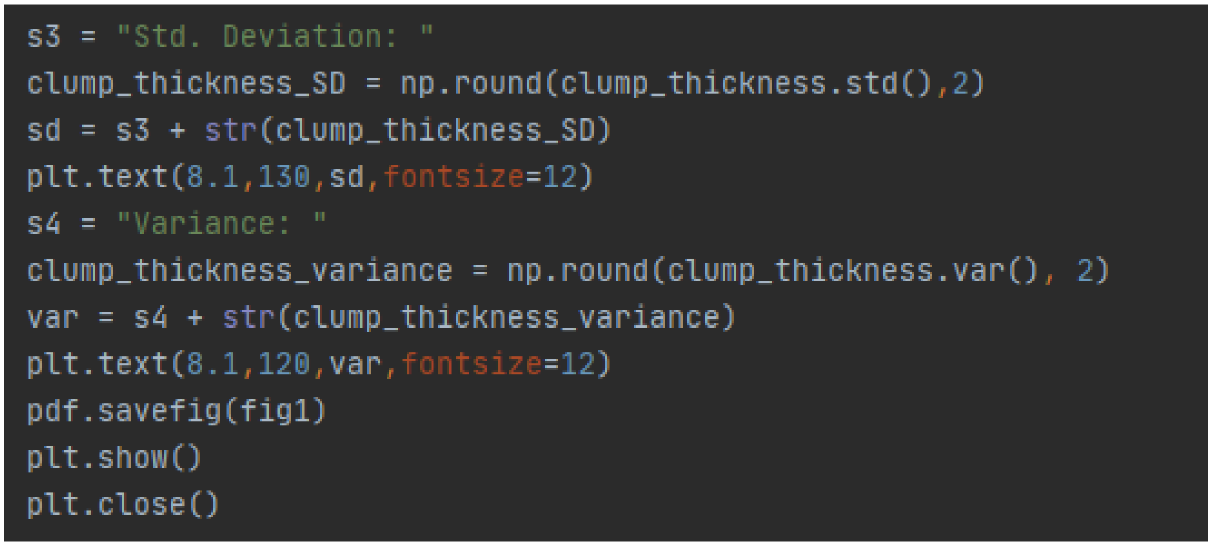
For each attribute we will plot the histogram using the matploid library and assigning each diagram a different color, for readability and differentiability.

Each histogram plots a curve of the Number of People (y-axis) against the respective attribute (x-axis).



```
#For column A2 - Clump Thickness
clump_thickness = df['A2']
fig1 = plt.figure(figsize=(9,6))
plt.hist(clump_thickness, bins=9, color='blue', alpha=0.5)
plt.xticks(range(12))
plt.xlim(0,11)
plt.ylim(0,160)
plt.title('Fig1: Clump thickness')
plt.xlabel('thickness')
plt.ylabel('No. of people')
```

Next we compute the mean, standard deviation, variance and Median for each attribute and display to the histogram as follows:

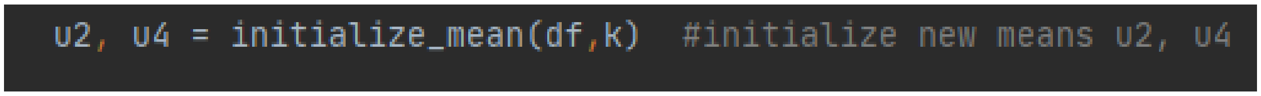


```
s3 = "Std. Deviation: "
clump_thickness_SD = np.round(clump_thickness.std(),2)
sd = s3 + str(clump_thickness_SD)
plt.text(8.1,130,sd,fontsize=12)
s4 = "Variance: "
clump_thickness_variance = np.round(clump_thickness.var(), 2)
var = s4 + str(clump_thickness_variance)
plt.text(8.1,120,var,fontsize=12)
pdf.savefig(fig1)
plt.show()
plt.close()
```

This is repeated for all the attributes which gives a total of 9 different histograms for the 9 different attributes.

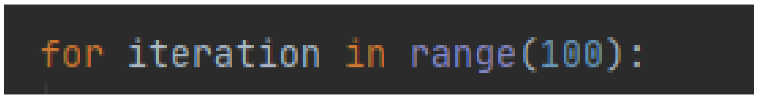
ii. Phase 2 – Implementing K-means algorithm

In phase 2 of the program, we implement the k-means algorithm using the dataset which has been imputed with the missing values. We initialize k = 2, giving two clusters, u2 and u4 (representing the cluster centers)



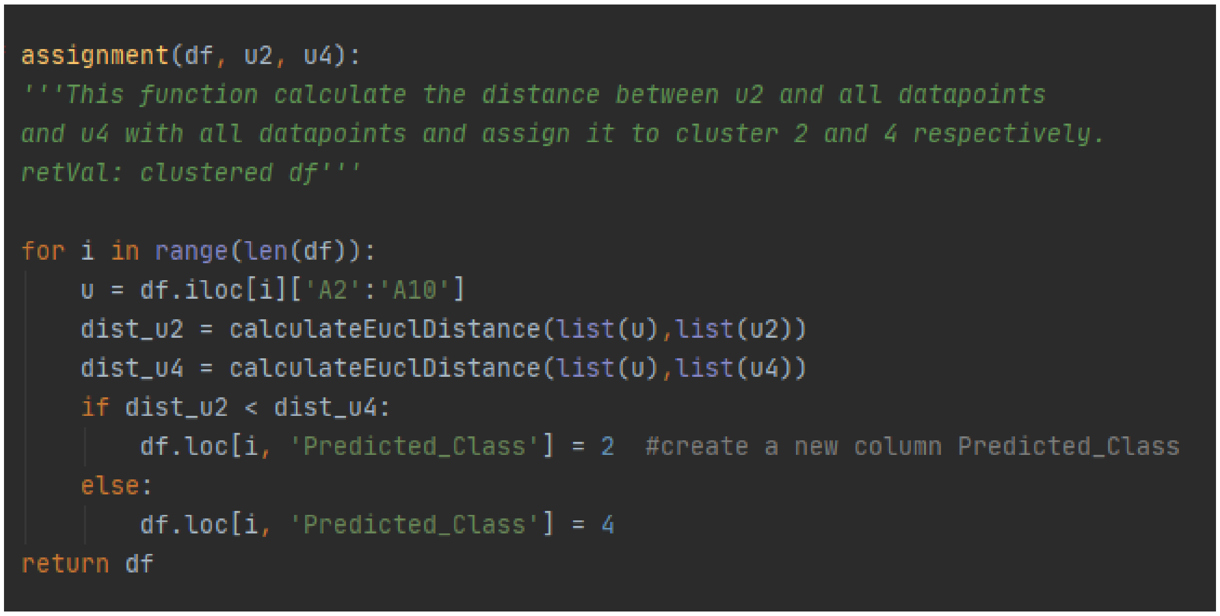
```
u2, u4 = initialize_mean(df,k)  #initialize new means u2, u4
```

We set our code to iterate the k-means algorithm over 100 times for simplicity. The number of iterations can be increased.



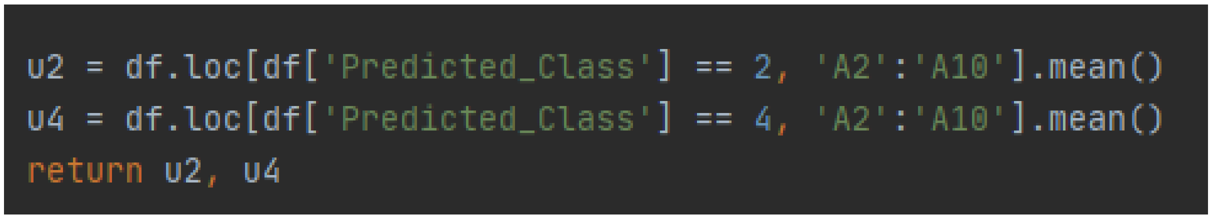
```
for iteration in range(100):
```

Using the “assignment” method, we can assign each data point to the respective cluster depending on the distance from cluster centers.



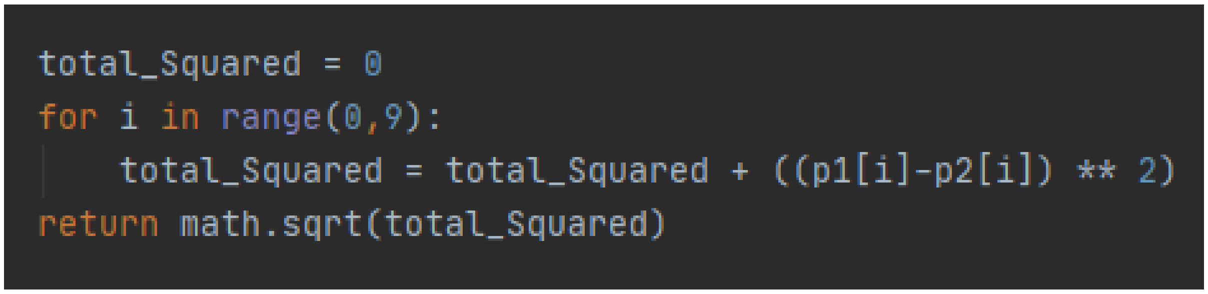
```
assignment(df, u2, u4):
'''This function calculate the distance between u2 and all datapoints
and u4 with all datapoints and assign it to cluster 2 and 4 respectively.
retVal: clustered df'''

for i in range(len(df)):
    u = df.iloc[i]['A2':'A10']
    dist_u2 = calculateEuclDistance(list(u),list(u2))
    dist_u4 = calculateEuclDistance(list(u),list(u4))
    if dist_u2 < dist_u4:
        df.loc[i, 'Predicted_Class'] = 2  #create a new column Predicted_Class
    else:
        df.loc[i, 'Predicted_Class'] = 4
return df
```

This method implement the function to calculate the Euclidean distance between each of the points and the cluster centers and assign them to the respective clusters. After this the new cluster centers are recalculated and reassigned.



```
u2 = df.loc[df['Predicted_Class'] == 2, 'A2':'A10'].mean()
u4 = df.loc[df['Predicted_Class'] == 4, 'A2':'A10'].mean()
return u2, u4
```

The Euclidean distance is calculated as:

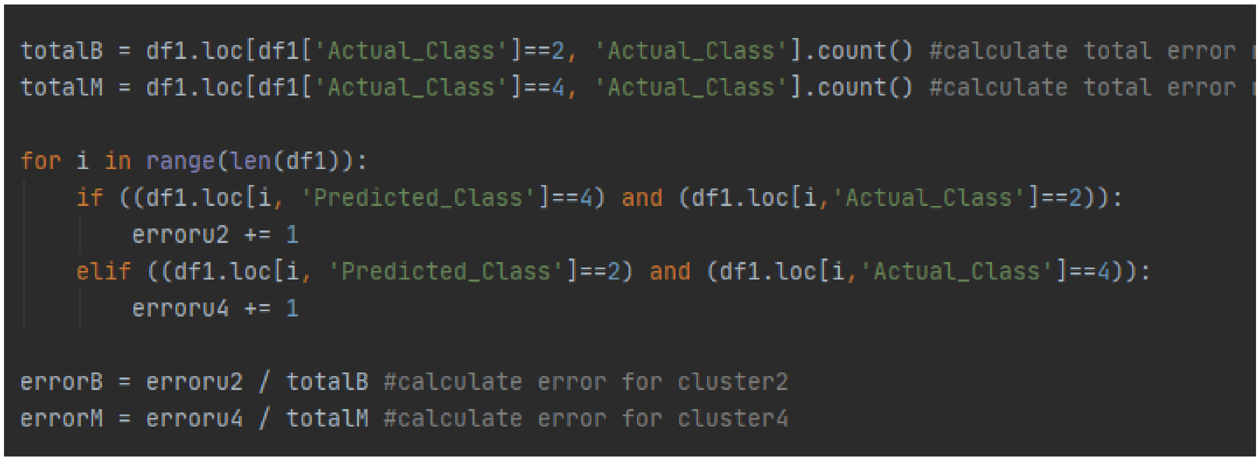


```
total_Squared = 0
for i in range(0,9):
    total_Squared = total_Squared + ((p1[i]-p2[i]) ** 2)
return math.sqrt(total_Squared)
```

Finally we print the final means for both clusters, u2 and u4, and display 20 of the data points and their assignments to the clusters as shown in the results section.

iii. Phase 3 – SSE error calculation

In the last phase we calculate the squared error of the 2 clusters to analyze the quality of the centroids and of the partition.

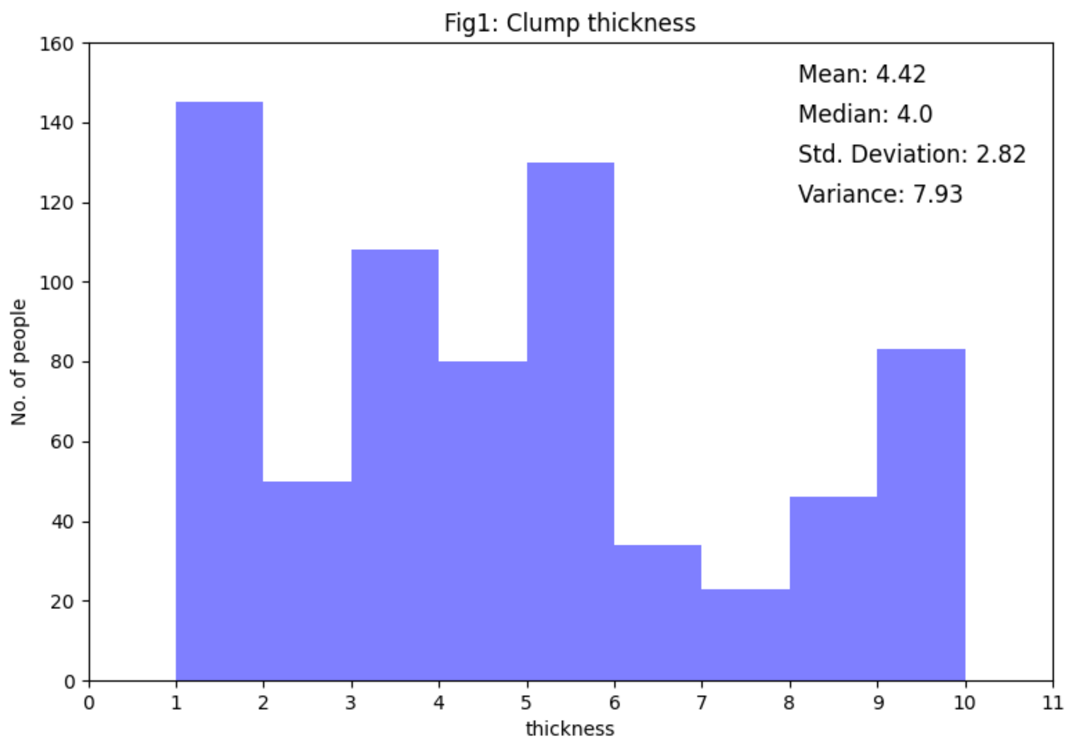


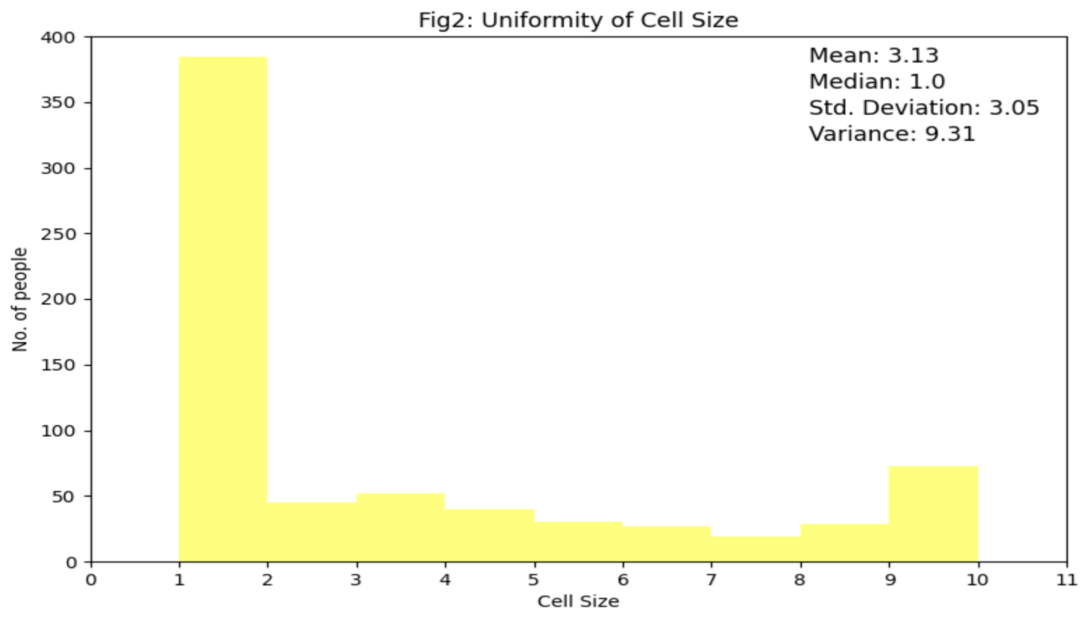
```
totalB = df1.loc[df1['Actual_Class']==2, 'Actual_Class'].count() #calculate total error
totalM = df1.loc[df1['Actual_Class']==4, 'Actual_Class'].count() #calculate total error

for i in range(len(df1)):
    if ((df1.loc[i, 'Predicted_Class']==4) and (df1.loc[i,'Actual_Class']==2)):
        erroru2 += 1
    elif ((df1.loc[i, 'Predicted_Class']==2) and (df1.loc[i,'Actual_Class']==4)):
        erroru4 += 1

errorB = erroru2 / totalB #calculate error for cluster2
errorM = erroru4 / totalM #calculate error for cluster4
```

The error here is a calculation of the number of datapoints wrongly classified. This is a function of the number of False Positives and False Negatives.

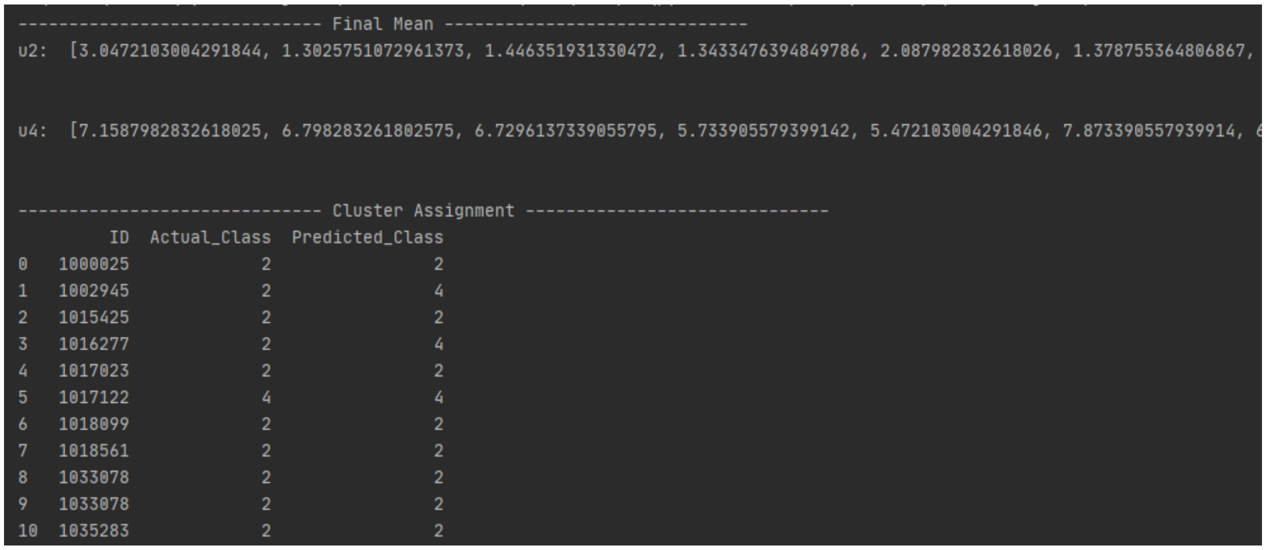
4.0) Results

Sample histogram for attributes

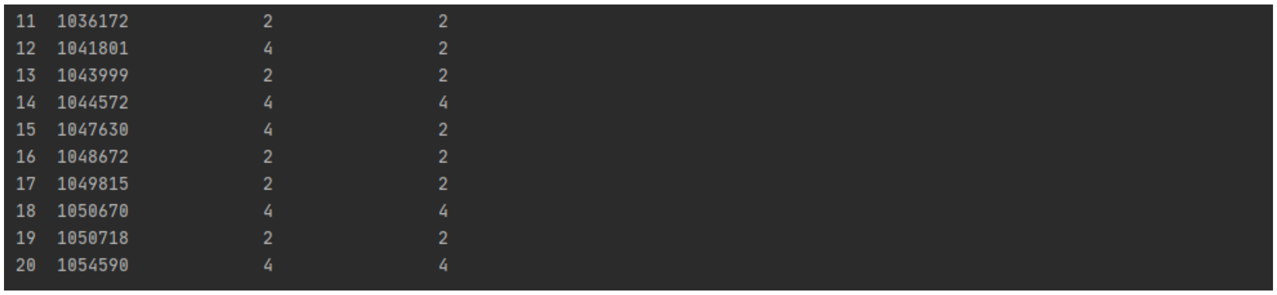




Phase 2 results showing final means and the clustering of 20 data points with the actual class and predicted classes.

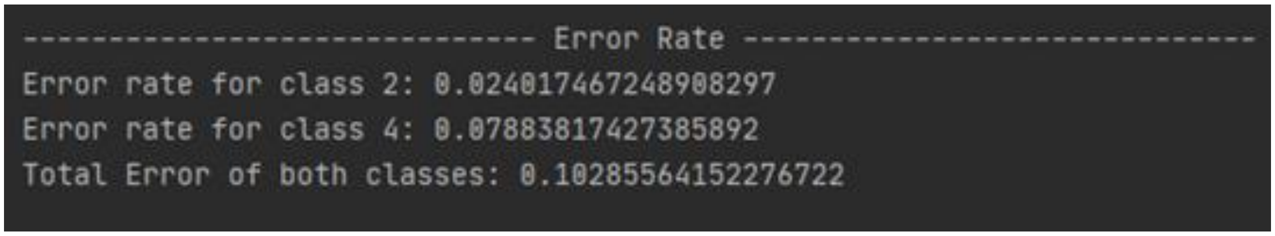


```
----------------------------- Final Mean -----------------------------
u2:  [3.0472103004291844, 1.3025751072961373, 1.446351931330472, 1.3433476394849786, 2.087982832618026, 1.378755364806867,


u4:  [7.1587982832618025, 6.798283261802575, 6.7296137339055795, 5.733905579399142, 5.472103004291846, 7.873390557939914,


----------------------------- Cluster Assignment -----------------------------
         ID  Actual_Class  Predicted_Class
0   1000025             2                2
1   1002945             2                4
2   1015425             2                2
3   1016277             2                4
4   1017023             2                2
5   1017122             4                4
6   1018099             2                2
7   1018561             2                2
8   1033078             2                2
9   1033078             2                2
10  1035283             2                2
```



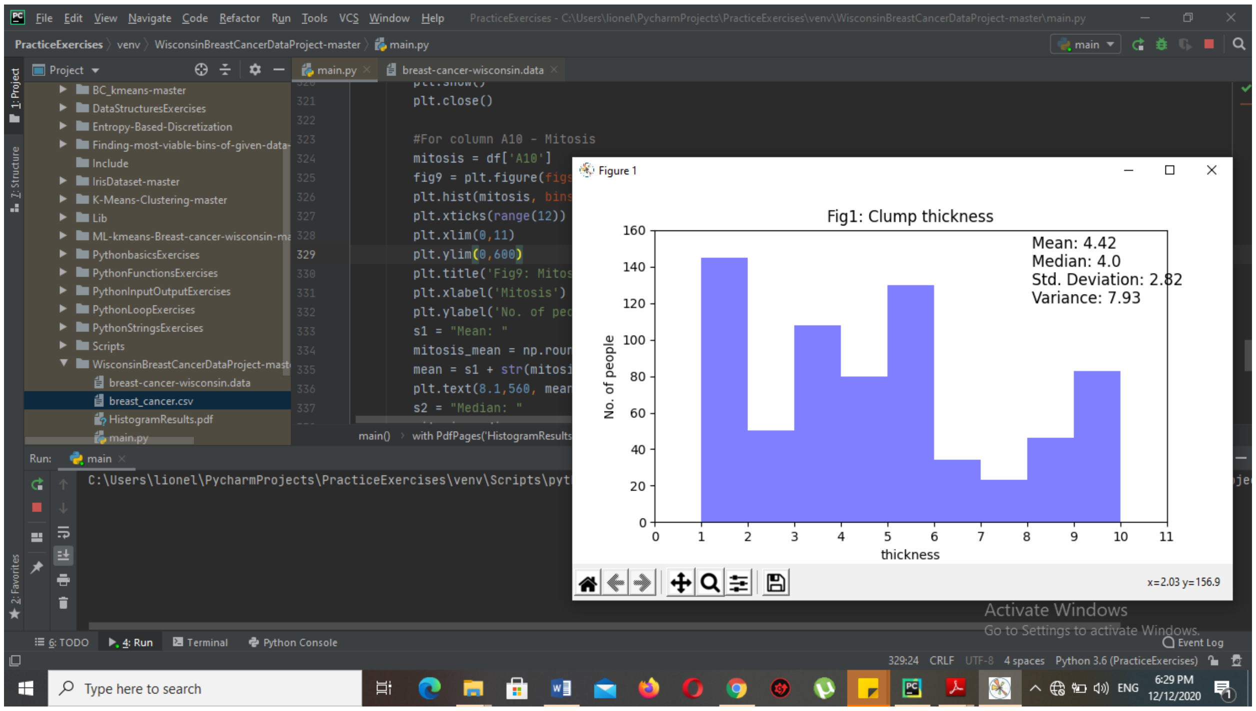
```
11  1036172             2                2
12  1041801             4                2
13  1043999             2                2
14  1044572             4                4
15  1047630             4                2
16  1048672             2                2
17  1049815             2                2
18  1050670             4                4
19  1050718             2                2
20  1054590             4                4
```

Phase 3 results show the error rate for each cluster and the total error rate



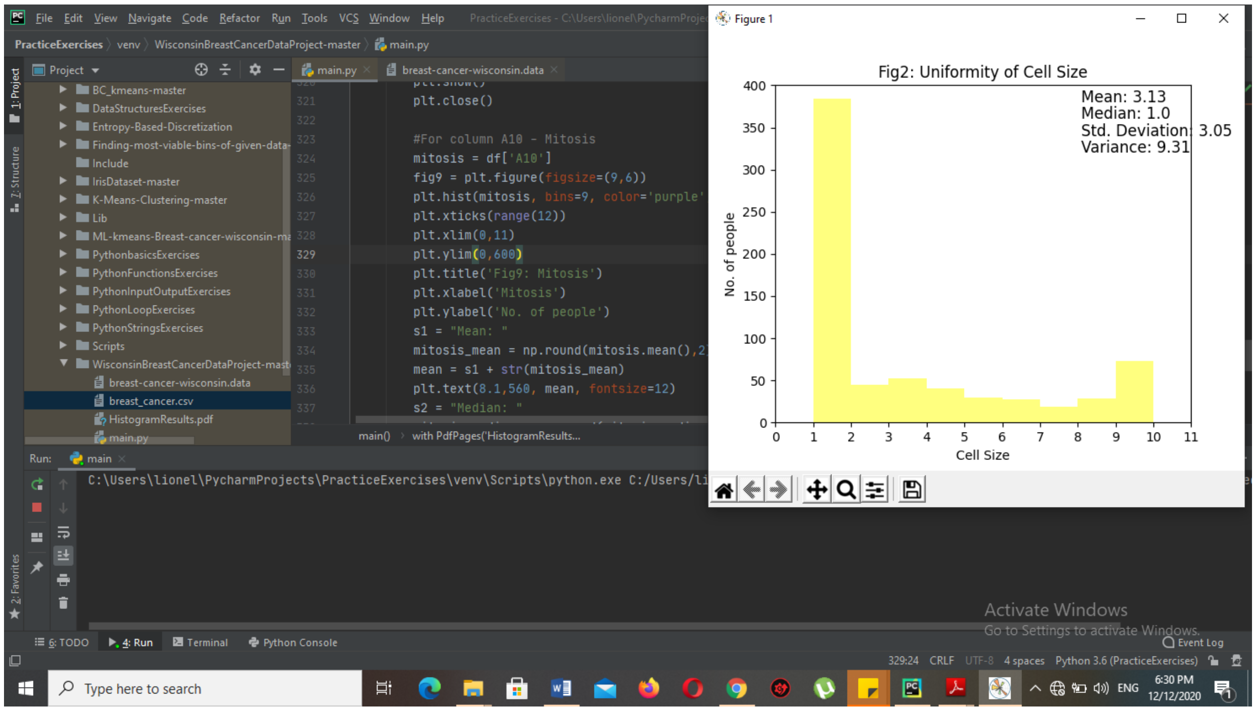
```
------------------------------ Error Rate ------------------------------
Error rate for class 2: 0.024017467248908297
Error rate for class 4: 0.07883817427385892
Total Error of both classes: 0.10285564152276722
```

Other results



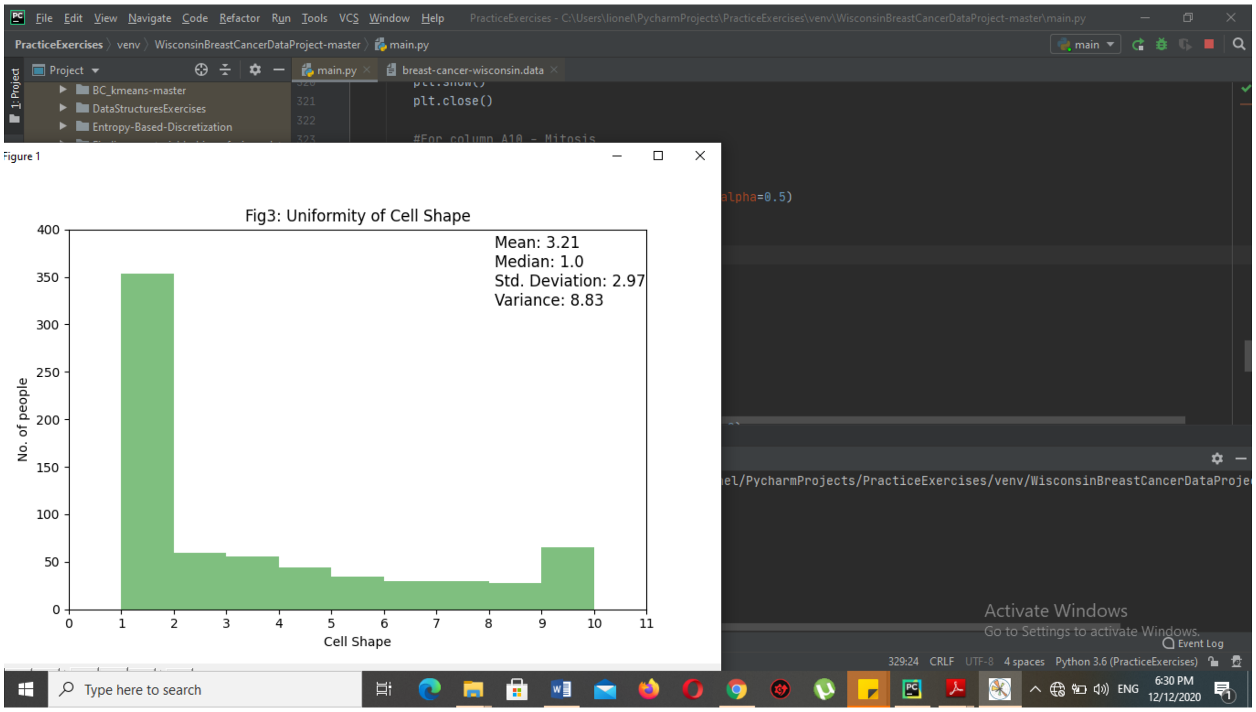
```
plt.close()

#For column A10 - Mitosis
mitosis = df['A10']
fig9 = plt.figure(figs
plt.hist(mitosis, bins
plt.xticks(range(12))
plt.xlim(0,11)
plt.ylim(0,600)
plt.title('Fig9: Mitos
plt.xlabel('Mitosis')
plt.ylabel('No. of peo
s1 = "Mean: "
mitosis_mean = np.roun
mean = s1 + str(mitosi
plt.text(8.1,560, mean
s2 = "Median: "
```

```
C:\Users\lionel\PycharmProjects\PracticeExercises\venv\Scripts\pyt
```



```
plt.close()

#For column A10 - Mitosis
mitosis = df['A10']
fig9 = plt.figure(figsize=(9,6))
plt.hist(mitosis, bins=9, color='purple'
plt.xticks(range(12))
plt.xlim(0,11)
plt.ylim(0,600)
plt.title('Fig9: Mitosis')
plt.xlabel('Mitosis')
plt.ylabel('No. of people')
s1 = "Mean: "
mitosis_mean = np.round(mitosis.mean(),2
mean = s1 + str(mitosis_mean)
plt.text(8.1,560, mean, fontsize=12)
s2 = "Median: "
```

```
C:\Users\lionel\PycharmProjects\PracticeExercises\venv\Scripts\python.exe C:/Users/li
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
PracticeExercises - C:\Users\lionel\PycharmProjects\PracticeExercises\venv\WisconsinBreastCancerDataProject-master\main.py
PracticeExercises 〉 venv 〉 WisconsinBreastCancerDataProject-master 〉 main.py
main
Project
main.py
breast-cancer-wisconsin.data
BC_kmeans-master
DataStructuresExercises
Entropy-Based-Discretization
Figure 1
Fig3: Uniformity of Cell Shape
Mean: 3.21
Median: 1.0
Std. Deviation: 2.97
Variance: 8.83
No. of people
Cell Shape
0
50
100
150
200
250
300
350
400
0
1
2
3
4
5
6
7
8
9
10
11
Activate Windows
Go to Settings to activate Windows.
Event Log
329:24 CRLF UTF-8 4 spaces Python 3.6 (PracticeExercises)
Type here to search
ENG
6:30 PM
12/12/2020

```
plt.close()

#For column A10 - Mitosis
```

```
alpha=0.5)
```

```
el/PycharmProjects/PracticeExercises/venv/WisconsinBreastCancerDataProje
```