FINAL REPORT

I590 – PYTHON PROGRAMMING

OBJECTIVE

Implement “k- means” algorithm for Wisconsin Breast Cancer data using Python.

PROBLEM STATEMENT

Breast cancer is a rising issue among women. A cancer’s stage is a crucial factor in deciding what treatment options to recommend, and in determining the patient’s prognosis. Today, in the United States, approximately one in eight women over their lifetime has a risk of developing breast cancer. An analysis of the most recent data has shown that the survival rate is 88% after 5 years of diagnosis and 80% after 10 years of diagnosis. With early detection and treatment, it is possible that this type of cancer will go into remission. In such a case, the worse fear of a cancer patient is the recurrence of the cancer.

Implement one of the most popular data mining technique- k means clustering on a very famous Wisconsin Breast Cancer Data. Classify benign and malign cells in two different groups.

DATA DESCRIPTION

Wolberg’s breast cancer data can be found here. Samples arrive periodically as Dr. Wolberg reports his clinical cases. There are total 11 columns or features in this dataset.

Column/Attribute Information:

1. Sample code number: id number
2. Clump Thickness: 1 - 10
3. Uniformity of Cell Size: 1 - 10
4. Uniformity of Cell Shape: 1 - 10
5. Marginal Adhesion: 1 - 10
6. Single Epithelial Cell Size: 1 - 10
7. Bare Nuclei: 1 - 10
8. Bland Chromatin: 1 - 10
9. Normal Nucleoli: 1 - 10
10. Mitoses: 1 - 10
11. Class: (2 for benign, 4 for malignant)

PHASE 1

In this phase, focus was on data download and data analysis. The missing values in the data set were imputed and various basic graphs were plotted for different attributes. The value of the attribute A7 was missing for some of the rows, it was imputed by the mean value of the column A7.

As per the Phase 1 code in Appendix, libraries pandas and matplotlib are imported. Pandas read_csv is used to read the file from the link, column names are provided in names clause and '?' is treated as the Null value. The function fillna is used to fill the column A7, with the mean of A7. To plot the graphs a figure is created.

create_hist()

This function accepts the parameters col, plot_num, title and xlabel is created. This function adds the subplot to the figure for the col passed. col is the parameter for attributes like A2, A3 etc. The parameter plot_num, title and xlabel are also passed to the function to create plot and to set the xlabel and title of the graph. This function creates the histogram for attributes A2 to A10 as per Phase 1 Result in appendix.

main()

In the main function, the create_hist function is called for each attribute from A2 to A10 and 9 histograms are generated in the same figure. To calculate the mean, median, standard deviation and variance, a key value pair is created. This key value pair uses the pandas data frame functions: mean, median, std and var. A data frame is created using the key value pairs and meaning full names are added to the index. The result is finally printed from the data frame.

PHASE 2

In this phase, the implementation of K-means algorithm is done. As per the Phase 2 Code in Appendix, pandas, numpy, math and time libraries are implemented. The below functions are created as per the requirements.

create_cluster()

This function accepts 5 parameters – m2, m4, cluster2, cluster4 and df. Parameters m2 and m4 are the means to create cluster2 and cluster4 respectively. The data frame df stores the downloaded data. The next line shows the input_row which stores the row of the data frame based on the loop index. The variables input_row_cal, m2_cal and m4_cal store the data for columns A2 to A10. The variable d2 and d4 calculates the euclidian distance of input_row_cal with both the means m2_cal and m4_cal. If d4 distance is less than d2 distance, the input_row will go to cluster4 else to cluster2. This way cluster2 and cluster4 are created and returned.

cal_new_mean()

This function accepts cluster2 and cluster4 to calculate their means and return. The mean series is added with dummy columns like 222 and 444 for Scn and Class as 1000. These values are added in the mean to be in consistent with the data frame columns. In the later stage these mean values are removed from the final cluster data frames.

main()

This function first download the data set in to a data frame and impute the missing values of column A7 with means. This code is inherited from Phase 1.

- Initialization
  This section of the code uses the np.random.randint() function to generate the random number between 0 and 699. Two random numbers are stored in variable rn1 and rn2. Using these random numbers two records using the iloc function are taken form the data frame df. These records are stored in variable m2 and m4 and are considered as first two random means to start the algorithm. m2 and m4 are added as the first record in cluster 2 and cluster 4 respectively.

- Assignment
  The function create_cluster is called to assign all the rows of the data frames df to cluster2 and cluster4 based on the means passed to the function.

- Recalculation
  The new means as new_m2 and new_m4 are calculated for the clusters created in the above assignment step. In the next step a loop is used from 0 to 1500 to iterate and create new clusters as cluster2_new and cluster4_new using create_clusters function as explained above. The variables prev_m2 and prev_m4 stores means of previous clusters and new_m2 and new_m4 stores the means of new cluster. If previous means and new means matches, the loop is breaking and clusters are finalized. This is the convergence step.

Once the above steps are done, the latest means are printed on the console. Now we have cluster2_new and cluster4_new as the latest clusters. The column PREDICTED_CLASS as 2 is added to cluster2_new and as 4 to cluster4_new. The data type of both the clusters are changed to int64. The mean value in the cluster which has a class of 1000 is also filtered out. Finally, we concatenate the cluster2_new and cluster4_new in to the final data frame final_output and sort on index. The columns of data frame final_output is renamed to match the required names. The data frame final_output is printed as cluster assignment.

PHASE 3

In this phase, the error rate is calculated for the clusters created in Phase 2. All the Phase2 code is taken for Phase 3

error_rate()

This function accepts cluster and class_value. The error rate is calculated by dividing the no error records with the total no of records in the data frame. The error rate varies for different runs, the best error rate which I got is 0.083 as shown in appendix Phase 3 Results.

FINAL CODE

The final code as shown in appendix is created by merging the steps done in Phase1, Phase2 and Phase3. This is the final python script which when executed via main performs all the activities from Phase 1 to Phase 3.

CONCLUSION

K-Means clustering algorithm is implemented successfully to classify the Wisconsin Breast Cancer Data. K-means clustering showed different results for different runs based on the first two random values selected. Finally, the error rate was observed which ranges from 0.08 to 1.91. The best error rate is **0.083446.**

APPENDIX

Phase 1 Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 14 07:34:31 2017

@author: vipinsingh
"""
# Import the pandas and matplotlib

import pandas as pd
import matplotlib.pyplot as plt

# Read the data in to a data frame using the pandas read_csv
# Consider the '?' as na_values
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
```

```
                    names = ['Scn', 'A2', 'A3', 'A4','A5','A6', 'A7',
'A8','A9','A10','CLASS'],na_values=['?'])

# Using the fillna to fill the NaN values with mean. This will fill the NaN
# values with mean of the column values excluding the NaN values
df = df.fillna(df.mean())


# Creating the figure
fig = plt.figure()

# A function to add the subplot, create histograms along with labels
# This function will be called for each column from A2 to A10
# Input parameters are col - dataframe column name A2, A3 ect, plot_num-
1,2 etc
# title and xlabel of the histogram
def create_hist(col,plot_num,title,xlabel):
    sp = fig.add_subplot(3,3,plot_num)
    sp.hist(df[col], bins = 20, color = "b")
    sp.set_xticks([1,2,3,4,5,6,7,8,9,10])
    sp.set_yticks([0,100,200,300,400,500,600])
    sp.set_title(title,fontsize='small')
    sp.set_xlabel(xlabel,fontsize='small')
    sp.set_ylabel('Frequency', fontsize='small')

# main function calling the create_hist function for each
# column value from A2 to A10
def main():

# Generating histogram for A2
    title = 'Histogram of Clump Thickness'
    xlabel = 'Clump Thickness'
    create_hist('A2',1,title,xlabel)

# Generating histogram for A3
    title = 'Histogram of Uniformity of Cell Size'
    xlabel = 'Uniformity of Cell Size'
    create_hist('A3',2,title,xlabel)

# Generating histogram for A4
    title = 'Histogram of Uniformity of Cell Shape'
    xlabel = 'Uniformity of Cell Shape'
    create_hist('A4',3,title,xlabel)

# Generating histogram for A5
    title = 'Histogram of Marginal Adhesion'
    xlabel = 'Marginal Adhesion'
    create_hist('A5',4,title,xlabel)

# Generating histogram for A6
    title = 'Histogram of Single Epithelial Cell Size'
    xlabel = 'Single Epithelial Cell Size'
    create_hist('A6',5,title,xlabel)

# Generating histogram for A7
    title ='Histogram of Bare Nuclei'
    xlabel ='Bare Nuclei'
```

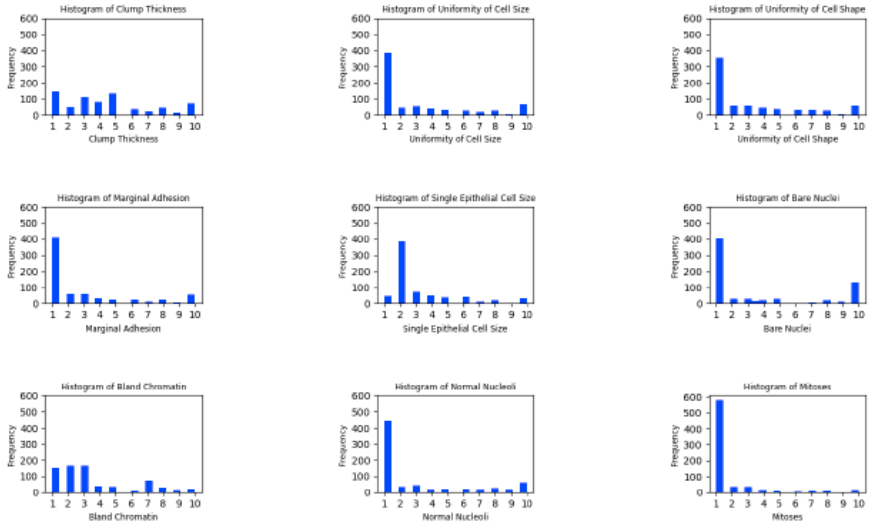
```
    create_hist('A7',6,title,xlabel)

# Generating histogram for A8
    title ='Histogram of Bland Chromatin'
    xlabel ='Bland Chromatin'
    create_hist('A8',7,title,xlabel)

# Generating histogram for A9
    title ='Histogram of Normal Nucleoli'
    xlabel = 'Normal Nucleoli'
    create_hist('A9',8,title,xlabel)

# Generating histogram for A10
    title = 'Histogram of Mitoses'
    xlabel = 'Mitoses'
    create_hist('A10',9,title,xlabel)

# Adjusting the space between subplots
    plt.tight_layout()


# Create the Keyvalue pair of mean, median, standard deviation and variance
    data = {"Mean" : [df['A2'].mean(),df['A3'].mean(),df['A4'].mean(),
                      df['A5'].mean(),df['A6'].mean(),df['A7'].mean(),
                      df['A8'].mean(),df['A9'].mean(),df['A10'].mean()],

            "Median" :
[df['A2'].median(),df['A3'].median(),df['A4'].median(),

df['A5'].median(),df['A6'].median(),df['A7'].median(),

df['A8'].median(),df['A9'].median(),df['A10'].median()],

            "Standard Deviation" :
[df['A2'].std(),df['A3'].std(),df['A4'].std(),
                        df['A5'].std(),df['A6'].std(),df['A7'].std(),
                        df['A8'].std(),df['A9'].std(),df['A10'].std()],

            "Variance" : [df['A2'].var(),df['A3'].var(),df['A4'].var(),
                        df['A5'].var(),df['A6'].var(),df['A7'].var(),
                        df['A8'].var(),df['A9'].var(),df['A10'].var()]
           }
# Converting the key value pair data in to a data frame and assigning the
index
    df1 = pd.DataFrame(data, index = ["Clump Thickness","Uniformity of Cell
Size",
                                      "Uniformity of Cell Shape","Marginal
Adhesion",
                                      "Single Epithelial Cell Size","Bare
Nuclei",
                                      "Bland Chromatin","Normal Nucleoli",
                                      "Mitoses"])

# Print the data frame for mean, median, standard deviation and variance
    print(df1)

# Calling main
```

```
main()
```

Phase 1 Results

Final Project Phase 1

Histograms of columns A2 to A10



Mean, Median, Standard Deviation and Variance of columns A2 to A10

```
>>> runfile('/Users/vipinsingh/Desktop/PYTHON/Assignment11/main.py', wdir='/Users/vipinsingh/Desktop/PYTHON/Assignment11')
                              Mean  Median  Standard Deviation   Variance
Clump Thickness           4.417740     4.0            2.815741   7.928395
Uniformity of Cell Size   3.134478     1.0            3.051459   9.311403
Uniformity of Cell Shape  3.207439     1.0            2.971913   8.832265
Marginal Adhesion         2.806867     1.0            2.855379   8.153191
Single Epithelial Cell Size 3.216023   2.0            2.214300   4.903124
Bare Nuclei               3.544656     1.0            3.601852  12.973335
Bland Chromatin           3.437768     3.0            2.438364   5.945620
Normal Nucleoli           2.866953     1.0            3.053634   9.324680
Mitoses                   1.589413     1.0            1.715078   2.941492
>>>
```

Phase 2 Code

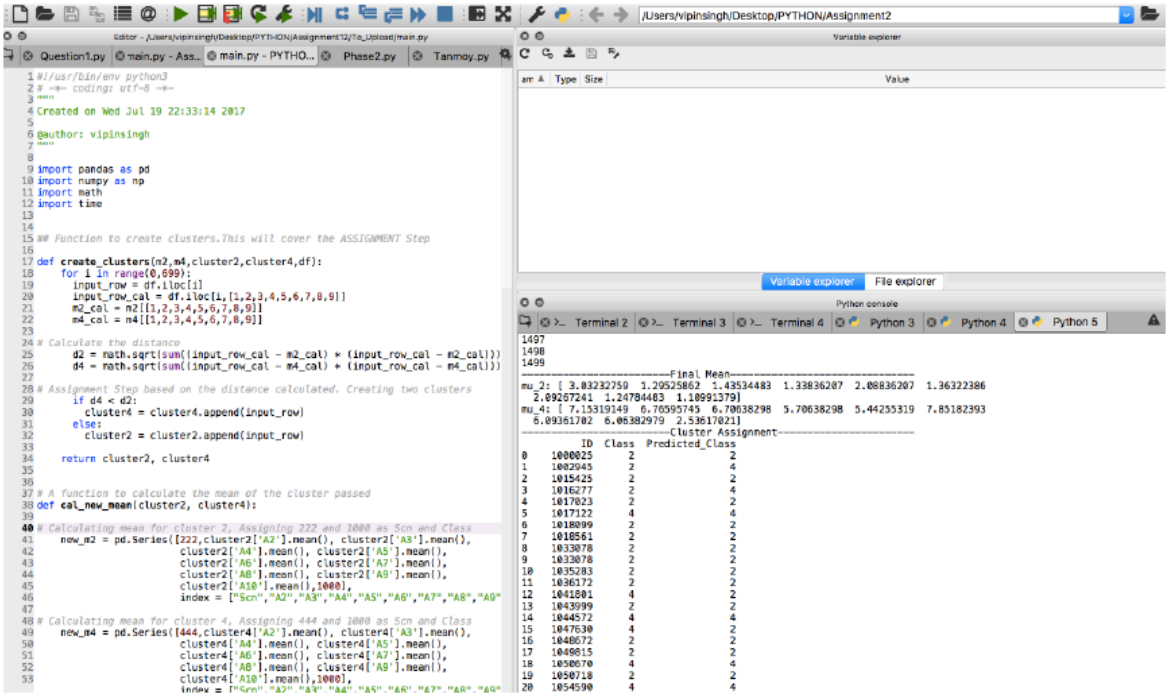
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 19 22:33:14 2017

@author: vipinsingh
"""

import pandas as pd
import numpy as np
import math
```

```
import time


## Function to create clusters.This will cover the ASSIGNMENT Step

def create_clusters(m2,m4,cluster2,cluster4,df):
    for i in range(0,699):
      input_row = df.iloc[i]
      input_row_cal = df.iloc[i,[1,2,3,4,5,6,7,8,9]]
      m2_cal = m2[[1,2,3,4,5,6,7,8,9]]
      m4_cal = m4[[1,2,3,4,5,6,7,8,9]]

# Calculate the distance
      d2 = math.sqrt(sum((input_row_cal - m2_cal) * (input_row_cal -
m2_cal)))
      d4 = math.sqrt(sum((input_row_cal - m4_cal) * (input_row_cal -
m4_cal)))

# Assignment Step based on the distance calculated. Creating two clusters
      if d4 < d2:
        cluster4 = cluster4.append(input_row)
      else:
        cluster2 = cluster2.append(input_row)

    return cluster2, cluster4


# A function to calculate the mean of the cluster passed
def cal_new_mean(cluster2, cluster4):

# Calculating mean for cluster 2, Assigning 222 and 1000 as Scn and Class
    new_m2 = pd.Series([222,cluster2['A2'].mean(), cluster2['A3'].mean(),
                        cluster2['A4'].mean(), cluster2['A5'].mean(),
                        cluster2['A6'].mean(), cluster2['A7'].mean(),
                        cluster2['A8'].mean(), cluster2['A9'].mean(),
                        cluster2['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

# Calculating mean for cluster 4, Assigning 444 and 1000 as Scn and Class
    new_m4 = pd.Series([444,cluster4['A2'].mean(), cluster4['A3'].mean(),
                        cluster4['A4'].mean(), cluster4['A5'].mean(),
                        cluster4['A6'].mean(), cluster4['A7'].mean(),
                        cluster4['A8'].mean(), cluster4['A9'].mean(),
                        cluster4['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

    return new_m2, new_m4


def main():

# Read the data in to a data frame using the pandas read_csv
# Consider the '?' as na_values
    df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
```

```
                    names = ['Scn', 'A2', 'A3', 'A4','A5','A6', 'A7',
'A8','A9','A10','CLASS'],na_values=['?'])

# Using the fillna to fill the NaN values with mean. This will fill the NaN
# values with mean of the column values excluding the NaN values
    df = df.fillna(df.mean())

## Step a INITIALIZATION for K = 2. It means two random means
# Selecting the two random means between 0 and 699
    rn1 = np.random.randint(0,699)
    rn2 = np.random.randint(0,699)
    print('Index of random row1',rn1)
    print('Index of random row2',rn2)

# Selecting the initial means m2 and m4. The following m2 and m4 are series
    m2 = df.iloc[rn1]
    m4 = df.iloc[rn2]

# Assigning the random means to the specific clusters2 and 4
    cluster2 = pd.DataFrame(m2).transpose()
    cluster4 = pd.DataFrame(m4).transpose()

# Creating the clusters using the first two random means
    cluster2, cluster4 = create_clusters(m2,m4,cluster2,cluster4,df)

# Now calculate the mean of the clusters created by random means
    new_m2, new_m4 = cal_new_mean(cluster2, cluster4)

# Run the loop to continiously build the clusters again and agian by
calculating
# the mean, RECALCULATION Step
    counter = 0
    for i in range(0,1500):
# Create new cluster by assigning the calculated means
        cluster2_new = pd.DataFrame(new_m2).transpose()
        cluster4_new = pd.DataFrame(new_m4).transpose()
# Call Create Cluster Function
        cluster2_new, cluster4_new =
create_clusters(new_m2,new_m4,cluster2_new,cluster4_new,df)
        prev_m2, prev_m4 = new_m2, new_m4
        new_m2, new_m4 = cal_new_mean(cluster2_new, cluster4_new)
        counter = counter + 1
#        print(i)

## Convergence, I have tested and commented, but as per requirement i will
run 1500 iterations
        if (
prev_m2[[1,2,3,4,5,6,7,8,9]].equals(new_m2[[1,2,3,4,5,6,7,8,9]]) and

prev_m4[[1,2,3,4,5,6,7,8,9]].equals(new_m4[[1,2,3,4,5,6,7,8,9]]) ):
            break;

    print('Ran for',counter,'iterations and previous means and current
means matched')
#   Print the Final mean
    print('-------------------------Final Mean-----------------------------
--')
```

```
    new_mean2 = new_m2[[1,2,3,4,5,6,7,8,9]]
    new_mean4 = new_m4[[1,2,3,4,5,6,7,8,9]]
    print('mu_2:',new_mean2.values)
    print('mu_4:',new_mean4.values)

# Assign the Predicted class to the cluster 2 as 2 and filtering out the
# mean record which has class of 1000
# Changed the data type to int
    cluster2_new['PREDICTED_CLASS'] = 2
    cluster2_new = cluster2_new.astype('int64')
    cluster2_new = cluster2_new[cluster2_new['CLASS'] != 1000]
#    print(cluster2_new)
# Assign the predicted class to the cluster 4 as 4 and filter out
# the mean record which was added as a part of the clustering
# Change the data type to int
    cluster4_new['PREDICTED_CLASS'] = 4
    cluster4_new = cluster4_new.astype('int64')
    cluster4_new = cluster4_new[cluster4_new['CLASS'] != 1000]
#    print(cluster4_new)

# Concatenate both clusters - cluster 2 and cluster 4
# Sort on index and dice and rename the columns
    final_output = pd.concat([cluster2_new, cluster4_new])
    final_output = final_output.sort_index()
    final_output = final_output[['Scn','CLASS','PREDICTED_CLASS']]
    final_output = final_output.rename(columns={'Scn': 'ID', 'CLASS':
'Class',
                       'PREDICTED_CLASS': 'Predicted_Class'})
# Printing the Final Output
    print('-------------------------Cluster Assignment--------------------
--')
    print(final_output)
    print('End time')
    print(time.ctime())


# Calling main
main()
```

Phase 2 Results



Phase 3 Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 19 22:33:14 2017

@author: vipinsingh
"""

import pandas as pd
import numpy as np
import math



## Function to create clusters.This will cover the ASSIGNMENT Step

def create_clusters(m2,m4,cluster2,cluster4,df):
    for i in range(0,699):
      input_row = df.iloc[i]
      input_row_cal = df.iloc[i,[1,2,3,4,5,6,7,8,9]]
      m2_cal = m2[[1,2,3,4,5,6,7,8,9]]
      m4_cal = m4[[1,2,3,4,5,6,7,8,9]]

# Calculate the distance
      d2 = math.sqrt(sum((input_row_cal - m2_cal) * (input_row_cal -
m2_cal)))
      d4 = math.sqrt(sum((input_row_cal - m4_cal) * (input_row_cal -
m4_cal)))
```

```
# Assignment Step based on the distance calculated. Creating two clusters
      if d4 < d2:
        cluster4 = cluster4.append(input_row)
      else:
        cluster2 = cluster2.append(input_row)

    return cluster2, cluster4


# A function to calculate the mean of the cluster passed
def cal_new_mean(cluster2, cluster4):

# Calculating mean for cluster 2, Assigning 222 and 1000 as Scn and Class
    new_m2 = pd.Series([222,cluster2['A2'].mean(), cluster2['A3'].mean(),
                        cluster2['A4'].mean(), cluster2['A5'].mean(),
                        cluster2['A6'].mean(), cluster2['A7'].mean(),
                        cluster2['A8'].mean(), cluster2['A9'].mean(),
                        cluster2['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

# Calculating mean for cluster 4, Assigning 444 and 1000 as Scn and Class
    new_m4 = pd.Series([444,cluster4['A2'].mean(), cluster4['A3'].mean(),
                        cluster4['A4'].mean(), cluster4['A5'].mean(),
                        cluster4['A6'].mean(), cluster4['A7'].mean(),
                        cluster4['A8'].mean(), cluster4['A9'].mean(),
                        cluster4['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

    return new_m2, new_m4

def error_rate(cluster,class_value):
    return len(cluster[cluster['CLASS'] == class_value]) / cluster.shape[0]



def main():

# Read the data in to a data frame using the pandas read_csv
# Consider the '?' as na_values
    df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                     names = ['Scn', 'A2', 'A3', 'A4','A5','A6', 'A7',
'A8','A9','A10','CLASS'],na_values=['?'])

# Using the fillna to fill the NaN values with mean. This will fill the NaN
# values with mean of the column values excluding the NaN values
    df = df.fillna(df.mean())

## Step a INITIALIZATION for K = 2. It means two random means
# Selecting the two random means between 0 and 699
    rn1 = np.random.randint(0,300)
    rn2 = np.random.randint(300,699)
    print('Index of random row1 - ',rn1)
    print('Index of random row2 - ',rn2)
```

```
# Selecting the initial means m2 and m4. The following m2 and m4 are series
    m2 = df.iloc[rn1]
    m4 = df.iloc[rn2]

# Assigning the random means to the specific clusters2 and 4
    cluster2 = pd.DataFrame(m2).transpose()
    cluster4 = pd.DataFrame(m4).transpose()

# Creating the clusters using the first two random means
    cluster2, cluster4 = create_clusters(m2,m4,cluster2,cluster4,df)

# Now calculate the mean of the clusters created by random means
    new_m2, new_m4 = cal_new_mean(cluster2, cluster4)

# Run the loop to continiously build the clusters again and agian by
calculating
# the mean, RECALCULATION Step
    counter = 0
    for i in range(0,50):
# Create new cluster by assigning the calculated means
        cluster2_new = pd.DataFrame(new_m2).transpose()
        cluster4_new = pd.DataFrame(new_m4).transpose()
# Call Create Cluster Function
        cluster2_new, cluster4_new =
create_clusters(new_m2,new_m4,cluster2_new,cluster4_new,df)
        prev_m2, prev_m4 = new_m2, new_m4
        new_m2, new_m4 = cal_new_mean(cluster2_new, cluster4_new)
        counter = counter + 1
        print(i)

## Convergence, I have tested and commented, but as per requirement i will
run 1500 iterations
        if (
prev_m2[[1,2,3,4,5,6,7,8,9]].equals(new_m2[[1,2,3,4,5,6,7,8,9]]) and

prev_m4[[1,2,3,4,5,6,7,8,9]].equals(new_m4[[1,2,3,4,5,6,7,8,9]]) ):
            break;

    print('\n\nRan for',counter,'iterations and previous means and current
means matched\n\n')
#   Print the Final mean
    print('\n\n-------------------------Final Mean-------------------------
------\n\n')
    new_mean2 = new_m2[[1,2,3,4,5,6,7,8,9]]
    new_mean4 = new_m4[[1,2,3,4,5,6,7,8,9]]
    print('mu_2:',new_mean2.values)
    print('mu_4:',new_mean4.values)

# Assign the Predicted class to the cluster 2 as 2 and filtering out the
# mean record which has class of 1000
# Changed the data type to int
    cluster2_new['PREDICTED_CLASS'] = 2
    cluster2_new = cluster2_new.astype('int64')
    cluster2_new = cluster2_new[cluster2_new['CLASS'] != 1000]
#    print(cluster2_new)
# Assign the predicted class to the cluster 4 as 4 and filter out
```

```
# the mean record which was added as a part of the clustering
# Change the data type to int
    cluster4_new['PREDICTED_CLASS'] = 4
    cluster4_new = cluster4_new.astype('int64')
    cluster4_new = cluster4_new[cluster4_new['CLASS'] != 1000]
#    print(cluster4_new)

# Concatenate both clusters - cluster 2 and cluster 4
# Sort on index and dice and rename the columns
    final_output = pd.concat([cluster2_new, cluster4_new])
    final_output = final_output.sort_index()
    final_output = final_output[['Scn','CLASS','PREDICTED_CLASS']]
    final_output = final_output.rename(columns={'Scn': 'ID', 'CLASS':
'Class',
                        'PREDICTED_CLASS': 'Predicted_Class'})
# Printing the Final Output
    print('\n\n------------------------Cluster Assignment----------------
----')
    print(final_output)
    print('\n\n------------------------Error Rate------------------------
----')

#    print('Error rows in cluster2',len(cluster2_new[cluster2_new['CLASS']
== 4]))
#    print('Size of Cluster2',cluster2_new.shape[0])
#    print('Error rows in cluster4',len(cluster4_new[cluster4_new['CLASS']
== 2]))
#    print('Size of Cluster4',cluster4_new.shape[0])

    err_b = error_rate(cluster2_new,4)
    err_m = error_rate(cluster4_new,2)

    print('\nError Rate - ',err_b + err_m)



# Calling main
main()
```

Phase 3 Results

```
------------------------Error Rate-----------------------------

Error Rate -  0.08344644167278062
>>>


------------------------Error Rate-----------------------------

Error Rate -  1.9120171673819741
>>>
```

FINAL CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 19 22:33:14 2017

@author: vipinsingh
"""

import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt


# A function to add the subplot, create histograms along with labels
# This function will be called for each column from A2 to A10
# Input parameters are col - dataframe column name A2, A3 ect, plot_num-
1,2 etc
# title and xlabel of the histogram


def create_hist(col,plot_num,title,xlabel,df,fig):
    sp = fig.add_subplot(3,3,plot_num)
    sp.hist(df[col], bins = 20, color = "b")
    sp.set_xticks([1,2,3,4,5,6,7,8,9,10])
    sp.set_yticks([0,100,200,300,400,500,600])
    sp.set_title(title,fontsize='small')
    sp.set_xlabel(xlabel,fontsize='small')
    sp.set_ylabel('Frequency', fontsize='small')

## Function to create clusters.This will cover the ASSIGNMENT Step

def create_clusters(m2,m4,cluster2,cluster4,df):
    for i in range(0,699):
      input_row = df.iloc[i]
      input_row_cal = df.iloc[i,[1,2,3,4,5,6,7,8,9]]
      m2_cal = m2[[1,2,3,4,5,6,7,8,9]]
      m4_cal = m4[[1,2,3,4,5,6,7,8,9]]

# Calculate the distance
      d2 = math.sqrt(sum((input_row_cal - m2_cal) * (input_row_cal -
m2_cal)))
      d4 = math.sqrt(sum((input_row_cal - m4_cal) * (input_row_cal -
m4_cal)))

# Assignment Step based on the distance calculated. Creating two clusters
      if d4 < d2:
        cluster4 = cluster4.append(input_row)
      else:
        cluster2 = cluster2.append(input_row)

    return cluster2, cluster4


# A function to calculate the mean of the cluster passed
def cal_new_mean(cluster2, cluster4):
```

```
# Calculating mean for cluster 2, Assigning 222 and 1000 as Scn and Class
    new_m2 = pd.Series([222,cluster2['A2'].mean(), cluster2['A3'].mean(),
                        cluster2['A4'].mean(), cluster2['A5'].mean(),
                        cluster2['A6'].mean(), cluster2['A7'].mean(),
                        cluster2['A8'].mean(), cluster2['A9'].mean(),
                        cluster2['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

# Calculating mean for cluster 4, Assigning 444 and 1000 as Scn and Class
    new_m4 = pd.Series([444,cluster4['A2'].mean(), cluster4['A3'].mean(),
                        cluster4['A4'].mean(), cluster4['A5'].mean(),
                        cluster4['A6'].mean(), cluster4['A7'].mean(),
                        cluster4['A8'].mean(), cluster4['A9'].mean(),
                        cluster4['A10'].mean(),1000],
                        index =
["Scn","A2","A3","A4","A5","A6","A7","A8","A9","A10","CLASS"])

    return new_m2, new_m4

def error_rate(cluster,class_value):
    return len(cluster[cluster['CLASS'] == class_value]) / cluster.shape[0]



def main():

# Read the data in to a data frame using the pandas read_csv
# Consider the '?' as na_values
    df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                     names = ['Scn', 'A2', 'A3', 'A4','A5','A6', 'A7',
'A8','A9','A10','CLASS'],na_values=['?'])

# Using the fillna to fill the NaN values with mean. This will fill the NaN
# values with mean of the column values excluding the NaN values
    df = df.fillna(df.mean())

# main function calling the create_hist function for each
# column value from A2 to A10
# Creating the figure
    fig = plt.figure()


# Generating histogram for A2
    title = 'Histogram of Clump Thickness'
    xlabel = 'Clump Thickness'
    create_hist('A2',1,title,xlabel,df,fig)

# Generating histogram for A3
    title = 'Histogram of Uniformity of Cell Size'
    xlabel = 'Uniformity of Cell Size'
    create_hist('A3',2,title,xlabel,df,fig)

# Generating histogram for A4
    title = 'Histogram of Uniformity of Cell Shape'
```

```
    xlabel = 'Uniformity of Cell Shape'
    create_hist('A4',3,title,xlabel,df,fig)

# Generating histogram for A5
    title = 'Histogram of Marginal Adhesion'
    xlabel = 'Marginal Adhesion'
    create_hist('A5',4,title,xlabel,df,fig)

# Generating histogram for A6
    title = 'Histogram of Single Epithelial Cell Size'
    xlabel = 'Single Epithelial Cell Size'
    create_hist('A6',5,title,xlabel,df,fig)

# Generating histogram for A7
    title ='Histogram of Bare Nuclei'
    xlabel ='Bare Nuclei'
    create_hist('A7',6,title,xlabel,df,fig)

# Generating histogram for A8
    title ='Histogram of Bland Chromatin'
    xlabel ='Bland Chromatin'
    create_hist('A8',7,title,xlabel,df,fig)

# Generating histogram for A9
    title ='Histogram of Normal Nucleoli'
    xlabel = 'Normal Nucleoli'
    create_hist('A9',8,title,xlabel,df,fig)

# Generating histogram for A10
    title = 'Histogram of Mitoses'
    xlabel = 'Mitoses'
    create_hist('A10',9,title,xlabel,df,fig)

# Adjusting the space between subplots
    plt.tight_layout()


# Create the Keyvalue pair of mean, median, standard deviation and variance
    data = {"Mean" : [df['A2'].mean(),df['A3'].mean(),df['A4'].mean(),
                      df['A5'].mean(),df['A6'].mean(),df['A7'].mean(),
                      df['A8'].mean(),df['A9'].mean(),df['A10'].mean()],

            "Median" :
[df['A2'].median(),df['A3'].median(),df['A4'].median(),

df['A5'].median(),df['A6'].median(),df['A7'].median(),

df['A8'].median(),df['A9'].median(),df['A10'].median()],

            "Standard Deviation" :
[df['A2'].std(),df['A3'].std(),df['A4'].std(),
                       df['A5'].std(),df['A6'].std(),df['A7'].std(),
                       df['A8'].std(),df['A9'].std(),df['A10'].std()],

            "Variance" : [df['A2'].var(),df['A3'].var(),df['A4'].var(),
                       df['A5'].var(),df['A6'].var(),df['A7'].var(),
                       df['A8'].var(),df['A9'].var(),df['A10'].var()]
```

```
            }
# Converting the key value pair data in to a data frame and assigning the
index
    df1 = pd.DataFrame(data, index = ["Clump Thickness","Uniformity of Cell
Size",
                                      "Uniformity of Cell Shape","Marginal
Adhesion",
                                      "Single Epithelial Cell Size","Bare
Nuclei",
                                      "Bland Chromatin","Normal Nucleoli",
                                      "Mitoses"])

# Print the data frame for mean, median, standard deviation and variance
    print(df1)

## Step a INITIALIZATION for K = 2. It means two random means
# Selecting the two random means between 0 and 699
    rn1 = np.random.randint(0,300)
    rn2 = np.random.randint(300,699)
    print('Index of random row1 - ',rn1)
    print('Index of random row2 - ',rn2)

# Selecting the initial means m2 and m4. The following m2 and m4 are series
    m2 = df.iloc[rn1]
    m4 = df.iloc[rn2]

# Assigning the random means to the specific clusters2 and 4
    cluster2 = pd.DataFrame(m2).transpose()
    cluster4 = pd.DataFrame(m4).transpose()

# Creating the clusters using the first two random means
    cluster2, cluster4 = create_clusters(m2,m4,cluster2,cluster4,df)

# Now calculate the mean of the clusters created by random means
    new_m2, new_m4 = cal_new_mean(cluster2, cluster4)

# Run the loop to continiously build the clusters again and agian by
calculating
# the mean, RECALCULATION Step
    counter = 0
    for i in range(0,1500):
# Create new cluster by assigning the calculated means
        cluster2_new = pd.DataFrame(new_m2).transpose()
        cluster4_new = pd.DataFrame(new_m4).transpose()
# Call Create Cluster Function
        cluster2_new, cluster4_new =
create_clusters(new_m2,new_m4,cluster2_new,cluster4_new,df)
        prev_m2, prev_m4 = new_m2, new_m4
        new_m2, new_m4 = cal_new_mean(cluster2_new, cluster4_new)
        counter = counter + 1
        print(i)

# Convergence iterations
        if (
prev_m2[[1,2,3,4,5,6,7,8,9]].equals(new_m2[[1,2,3,4,5,6,7,8,9]]) and

prev_m4[[1,2,3,4,5,6,7,8,9]].equals(new_m4[[1,2,3,4,5,6,7,8,9]]) ):
```

```
            break;

    print('\n\nRan for',counter,'iterations and previous means and current
means matched\n\n')
#    Print the Final mean
    print('\n\n-------------------------Final Mean-------------------------
------\n\n')
    new_mean2 = new_m2[[1,2,3,4,5,6,7,8,9]]
    new_mean4 = new_m4[[1,2,3,4,5,6,7,8,9]]
    print('mu_2:',new_mean2.values)
    print('mu_4:',new_mean4.values)

# Assign the Predicted class to the cluster 2 as 2 and filtering out the
# mean record which has class of 1000
# Changed the data type to int
    cluster2_new['PREDICTED_CLASS'] = 2
    cluster2_new = cluster2_new.astype('int64')
    cluster2_new = cluster2_new[cluster2_new['CLASS'] != 1000]
#     print(cluster2_new)
# Assign the predicted class to the cluster 4 as 4 and filter out
# the mean record which was added as a part of the clustering
# Change the data type to int
    cluster4_new['PREDICTED_CLASS'] = 4
    cluster4_new = cluster4_new.astype('int64')
    cluster4_new = cluster4_new[cluster4_new['CLASS'] != 1000]
#     print(cluster4_new)

# Concatenate both clusters - cluster 2 and cluster 4
# Sort on index and dice and rename the columns
    final_output = pd.concat([cluster2_new, cluster4_new])
    final_output = final_output.sort_index()
    final_output = final_output[['Scn','CLASS','PREDICTED_CLASS']]
    final_output = final_output.rename(columns={'Scn': 'ID', 'CLASS':
'Class',
                      'PREDICTED_CLASS': 'Predicted_Class'})
# Printing the Final Output
    print('\n\n-------------------------Cluster Assignment-----------------
----')
    print(final_output)
    print('\n\n-------------------------Error Rate-------------------------
----')

#     print('Error rows in cluster2',len(cluster2_new[cluster2_new['CLASS']
== 4]))
#     print('Size of Cluster2',cluster2_new.shape[0])
#     print('Error rows in cluster4',len(cluster4_new[cluster4_new['CLASS']
== 2]))
#     print('Size of Cluster4',cluster4_new.shape[0])

    err_b = error_rate(cluster2_new,4)
    err_m = error_rate(cluster4_new,2)

    print('\nError Rate - ',err_b + err_m)
```

```
# Calling main
main()
```