CI7520 – Machine Learning and Artificial Intelligence

Topic: CLASSIC MACHINE LEARNING

Name: NILENDU SAHA

Stream: Mechatronic Systems

KU Number: K1943079

Date of Submission: 23/02/2020

Number of words: 4919

Technical Report on Classic Machine Learning

1. INTRODUCTION

This assignment is all about implementation of Classic Machine Learning techniques. The assigned dataset is the popular Wisconsin Breast Cancer dataset. In order to carry on with the assignment, a certain pipeline is followed.

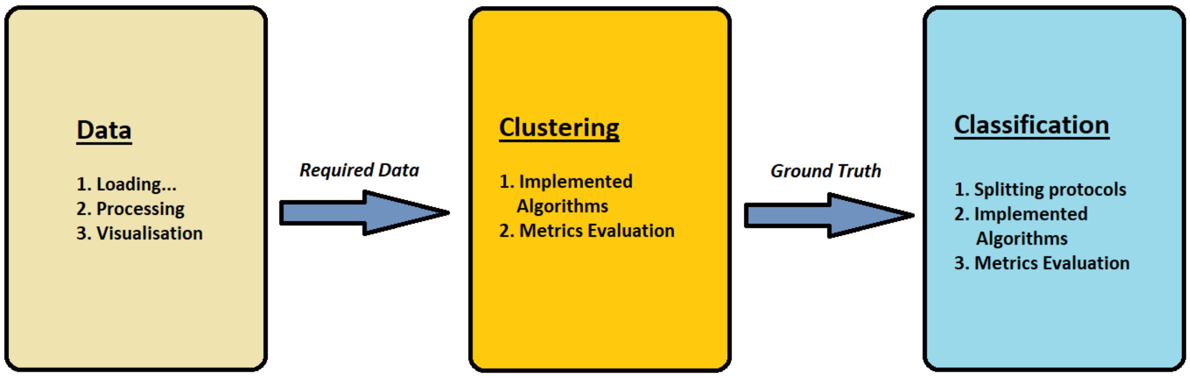


Fig. 1.1

Figure 1.1 briefly explains the extent and the aim of the assignment. All the codes have been written in Geany IDE with a Python compiler of version 3.6.0.

2. DATA

For the assignment, the Wisconsin Breast Cancer Dataset is taken into consideration. This is an extremely popular data set where classic machine learning algorithms can be applied. This simple classification dataset contains two major classes named as Malignant(M) and Benign(B). Both the classes determine the diagnosis status of a patient. If the patient has developed breast cancer, then the diagnosis status falls into Malignant Class. Otherwise, it falls into the Benign Class. The current dataset has 30 features or dimensions which are taken into consideration. Among the 30 features, the 10 real valued features for each nucleus are:

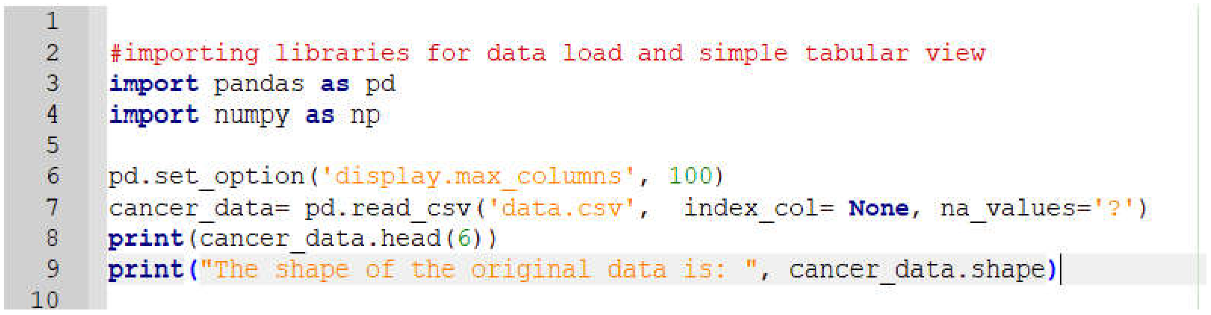
- Radius
- Texture
- Perimeter
- Area
- Smoothness
- Compactness
- Concavity
- Concave Points
- Symmetry

- Fractal Dimension

The **mean**, **standard error** and **worst** were computed for each of the above-mentioned features, which eventually resulted into 30 features. For example, column 3 of the dataset is **radius_mean**, then column 13 is **radius_se** and column 23 is **radius_worst**. Each instances of this dataset are already separated into two above mentioned classes. Later on, these labels should be marked as true labels and put in a variable, **y_true**. The **y_true** values will only be used for evaluation purposes, in clustering.

2.1. DATA LOADING

In order to work with the dataset, certain libraries in python will help in loading and viewing the dataset.

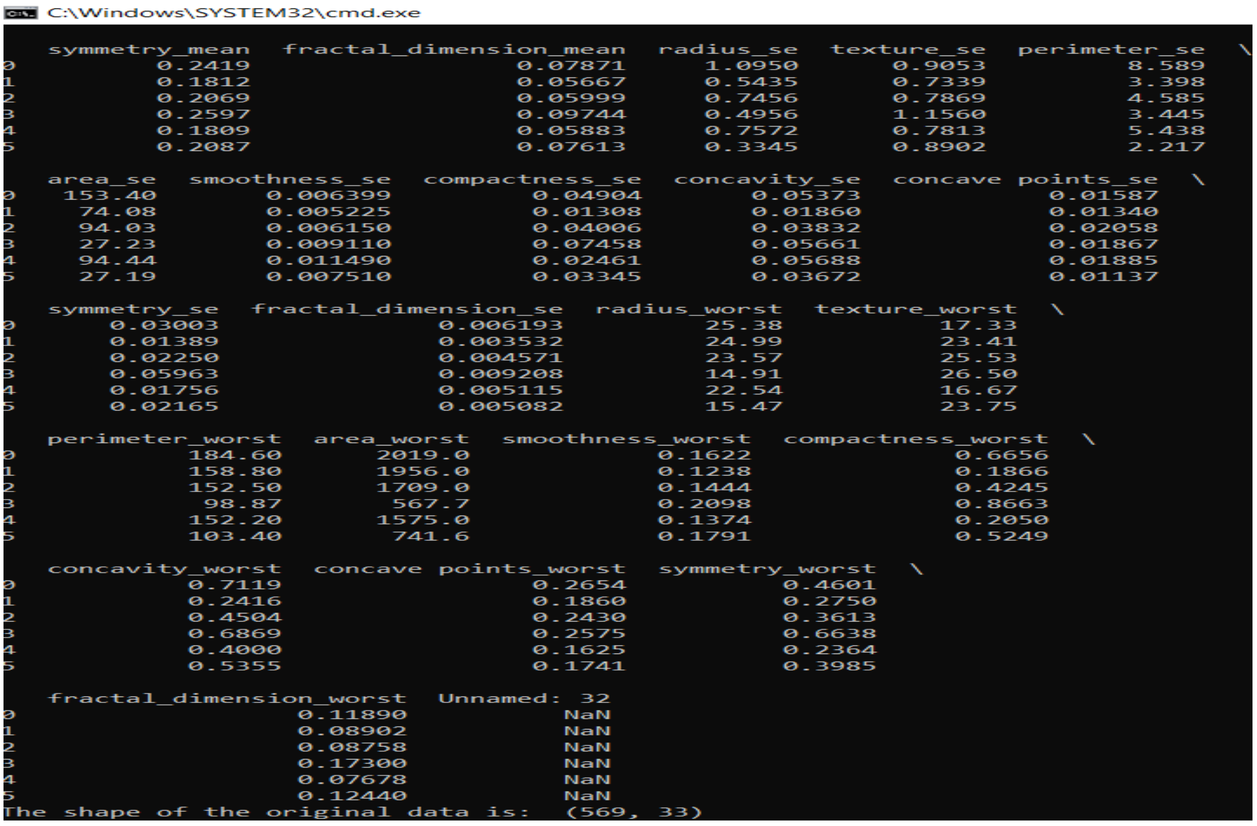


```
 1
 2  #importing libraries for data load and simple tabular view
 3  import pandas as pd
 4  import numpy as np
 5
 6  pd.set_option('display.max_columns', 100)
 7  cancer_data= pd.read_csv('data.csv',  index_col= None, na_values='?')
 8  print(cancer_data.head(6))
 9  print("The shape of the original data is: ", cancer_data.shape)
10
```

Fig. 2.1

Since the dataset is in csv format, pandas should be imported in order to access the dataset. Numpy library is imported for carrying out all the numerical operations throughout the coursework. The command written on the eighth line, is for printing first 6 rows of the dataset. The ninth line indicates the shape of the original dataset.

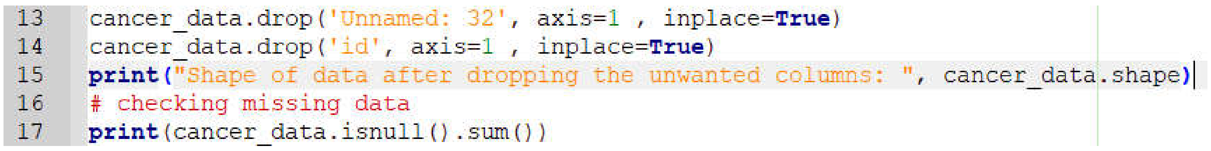
Output:



```
C:\Windows\SYSTEM32\cmd.exe
   symmetry_mean  fractal_dimension_mean  radius_se  texture_se  perimeter_se  \
0         0.2419                 0.07871     1.0950      0.9053         8.589
1         0.1812                 0.05667     0.5435      0.7339         3.398
2         0.2069                 0.05999     0.7456      0.7869         4.585
3         0.2597                 0.09744     0.4956      1.1560         3.445
4         0.1809                 0.05883     0.7572      0.7813         5.438
5         0.2087                 0.07613     0.3345      0.8902         2.217

   area_se  smoothness_se  compactness_se  concavity_se  concave points_se  \
0   153.40       0.006399         0.04904       0.05373            0.01587
1    74.08       0.005225         0.01308       0.01860            0.01340
2    94.03       0.006150         0.04006       0.03832            0.02058
3    27.23       0.009110         0.07458       0.05661            0.01867
4    94.44       0.011490         0.02461       0.05688            0.01885
5    27.19       0.007510         0.03345       0.03672            0.01137

   symmetry_se  fractal_dimension_se  radius_worst  texture_worst  \
0      0.03003              0.006193         25.38          17.33
1      0.01389              0.003532         24.99          23.41
2      0.02250              0.004571         23.57          25.53
3      0.05963              0.009208         14.91          26.50
4      0.01756              0.005115         22.54          16.67
5      0.02165              0.005082         15.47          23.75

   perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0           184.60      2019.0            0.1622             0.6656
1           158.80      1956.0            0.1238             0.1866
2           152.50      1709.0            0.1444             0.4245
3            98.87       567.7            0.2098             0.8663
4           152.20      1575.0            0.1374             0.2050
5           103.40       741.6            0.1791             0.5249

   concavity_worst  concave points_worst  symmetry_worst  \
0           0.7119                0.2654          0.4601
1           0.2416                0.1860          0.2750
2           0.4504                0.2430          0.3613
3           0.6869                0.2575          0.6638
4           0.4000                0.1625          0.2364
5           0.5355                0.1741          0.3985

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN
5                  0.12440          NaN
The shape of the original data is:  (569, 33)
```

Fig. 2.2

As per the output, dimensions of the dataset is 33. This is because the exisiting dataset conatains 2 unwanted columns (id and Unnamed: 32) and the other column is the diagnosis (later on which forms the **y_true**).

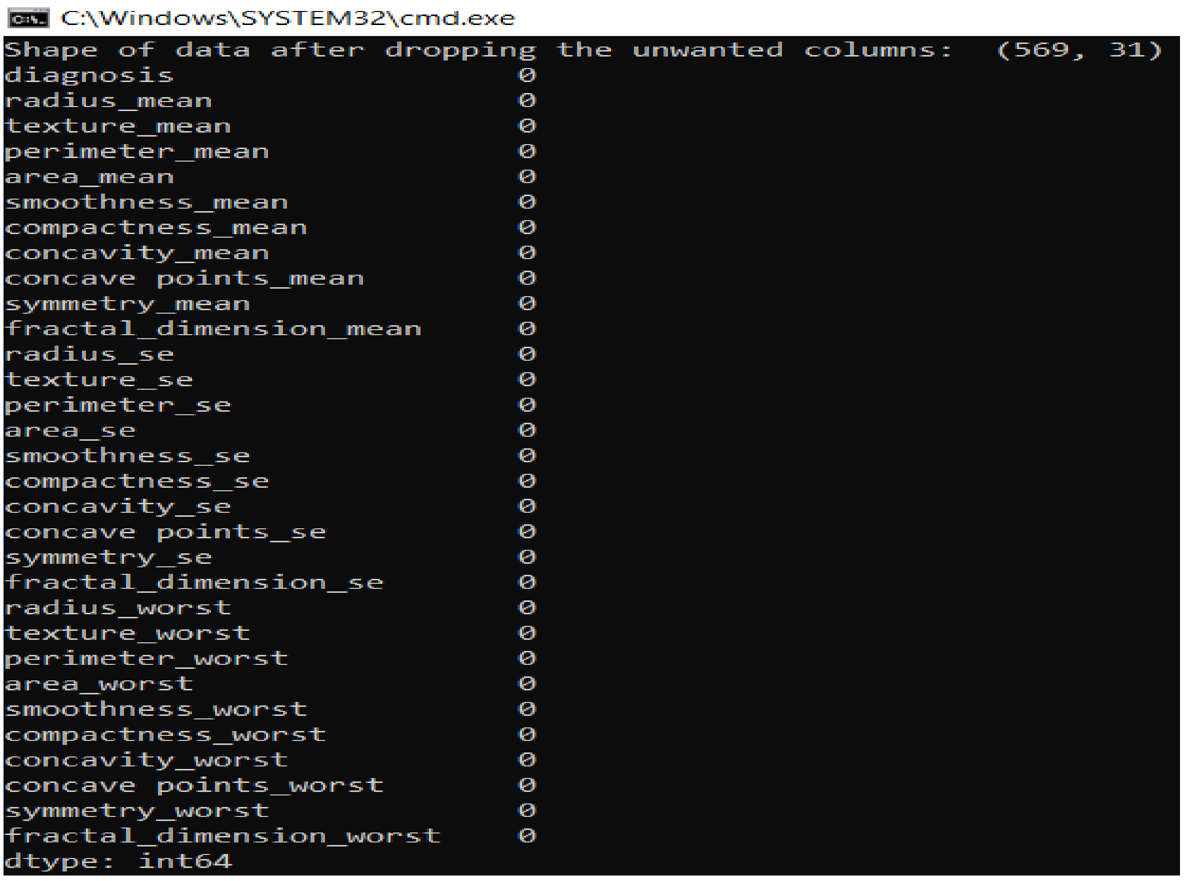


```
13  cancer_data.drop('Unnamed: 32', axis=1 , inplace=True)
14  cancer_data.drop('id', axis=1 , inplace=True)
15  print("Shape of data after dropping the unwanted columns: ", cancer_data.shape)
16  # checking missing data
17  print(cancer_data.isnull().sum())
```

Fig. 2.3

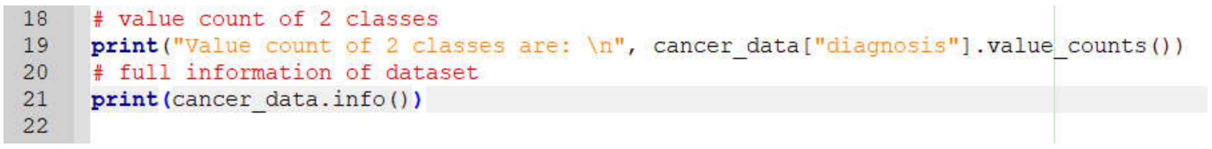
Lines 13 and 14 drops out the unwanted columns from the dataset. Line 17 checks if there is any missing value in the overall dataset. In this case, there is no data missing.

Output:



```
Shape of data after dropping the unwanted columns:  (569, 31)
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

Fig. 2.4

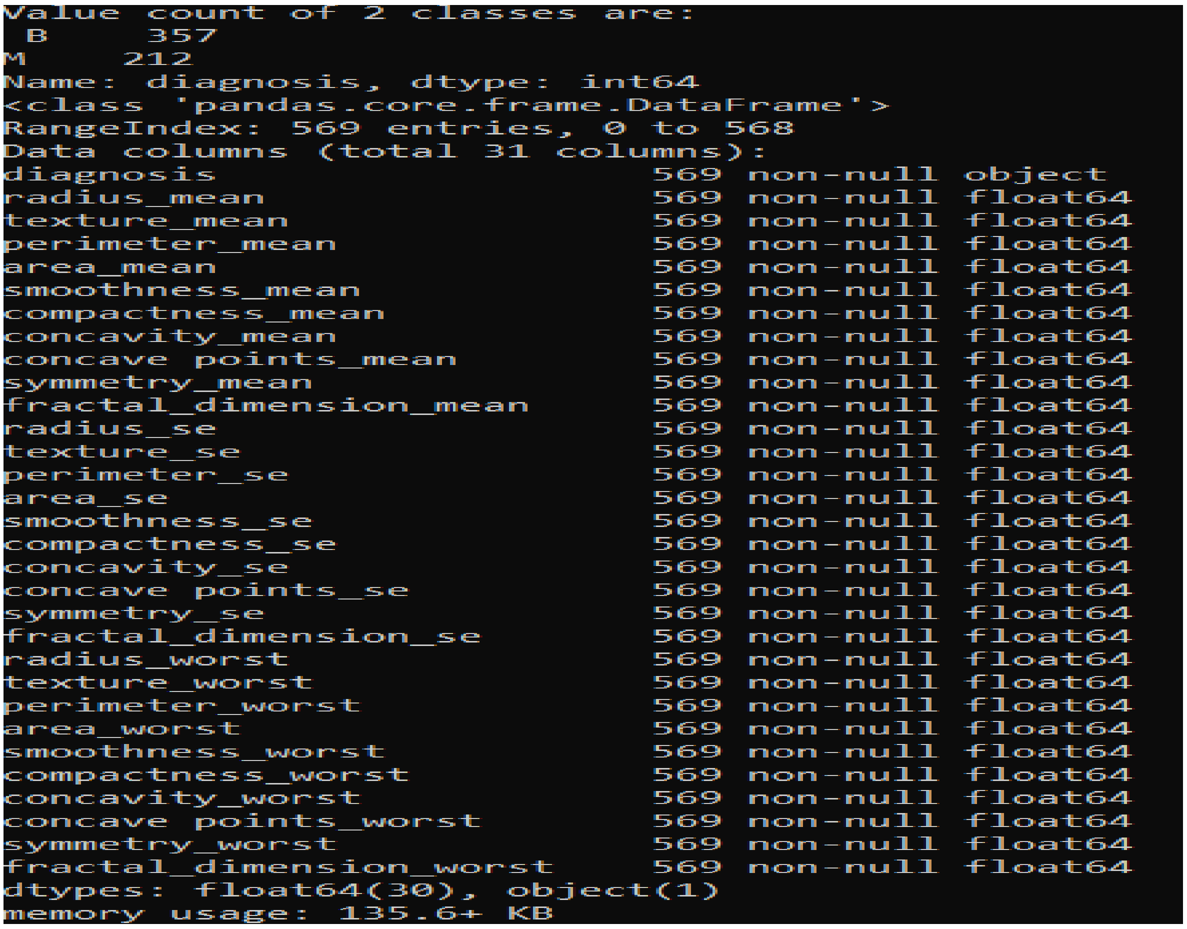


```
18  # value count of 2 classes
19  print("Value count of 2 classes are: \n", cancer_data["diagnosis"].value_counts())
20  # full information of dataset
21  print(cancer_data.info())
22
```

Fig. 2.5

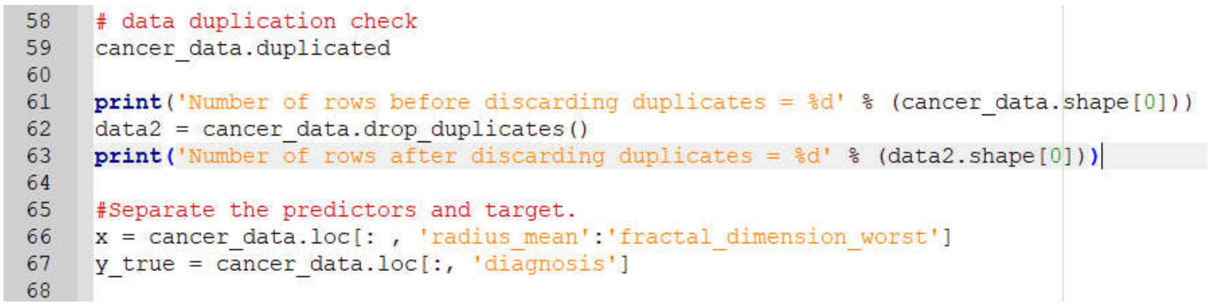
Line 19 counts the total number of **Malignant(M)** and **Benign(B)** patients. In this dataset there are, 357 Benign and 212 Malignant cases. Line 21 gives the information about datatypes under every column. All the 30 features have float64 type datatype. Whereas, the diagnosis column has object datatype. It has to be changed to numbers before feeding it into any of the machine learning algorithms.

Output:



```
Value count of 2 classes are:
 B     357
M     212
Name: diagnosis, dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
dtypes: float64(30), object(1)
memory usage: 135.6+ KB
```

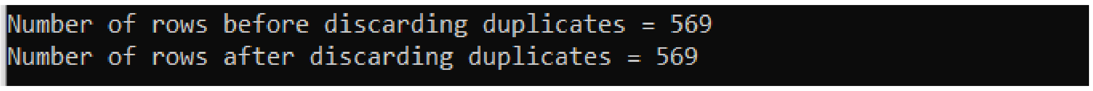
Fig. 2.6



```
58  # data duplication check
59  cancer_data.duplicated
60
61  print('Number of rows before discarding duplicates = %d' % (cancer_data.shape[0]))
62  data2 = cancer_data.drop_duplicates()
63  print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))
64
65  #Separate the predictors and target.
66  x = cancer_data.loc[: , 'radius_mean':'fractal_dimension_worst']
67  y_true = cancer_data.loc[:, 'diagnosis']
68
```

Fig. 2.7

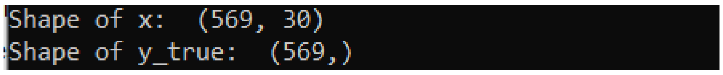
Output:



```
Number of rows before discarding duplicates = 569
Number of rows after discarding duplicates = 569
```

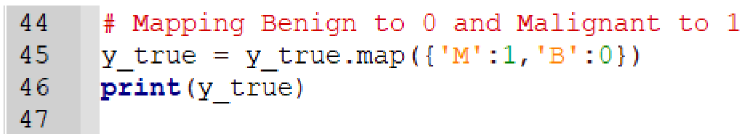
Fig. 2.8

Lines 59 to 63 quickly checks for any duplicated value in the given dataset. Fortunately, this dataset does not contain any duplicate values. The dataset is separated to x and y_true. Hence, the shapes of x and y_ true are (569,30) and (569,) respectively.



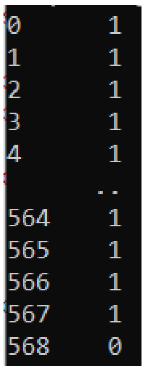
```
Shape of x:  (569, 30)
Shape of y_true:  (569,)
```

Fig. 2.9



```
44  # Mapping Benign to 0 and Malignant to 1
45  y_true = y_true.map({'M':1,'B':0})
46  print(y_true)
47
```

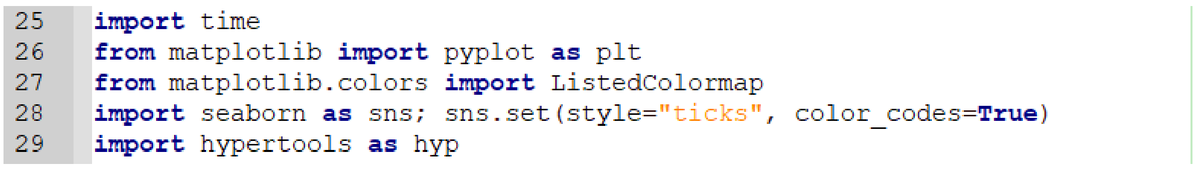
The datatypes of all the instances under y_true are objects. In machine learning, each and every working data is changed into numbers. These object datatypes can be changed to integers. The set only has 2 classes and 0 and 1 can be used for that. Hence, the code in line 45 converts the objects to integers with the help of mapping. The desired output is given below.



```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
```

Fig. 2.10

2.2. DATA VISUALISATION:

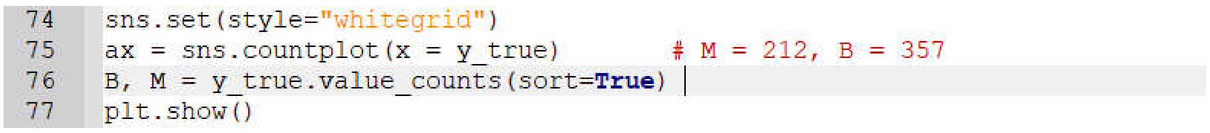


```
import time
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns; sns.set(style="ticks", color_codes=True)
import hypertools as hyp
```

Fig. 2.11

Visualising the data is a very important aspect in Classic Machine Learning. It helps to compare among the output data from any of the machine learning algorithms. It also helps to give feedbacks and draw inferences on any type of dataset. The above-mentioned libraries help to plot any graphs based on the output acquired from python. Based on the Wisconsin breast cancer dataset, various graphs will be plotted which will shape the data into pictures. It will help me to visualise the features and make critical comments.

- ## Bar Graph



```
sns.set(style="whitegrid")
ax = sns.countplot(x = y_true)          # M = 212, B = 357
B, M = y_true.value_counts(sort=True)
plt.show()
```

Fig. 2.12

The above set of codes helps to plot a bar graph. This bar graph shows the count of each classes that have been mentioned earlier. The blue coloured bar indicates the **Malignant(M)** class and the brown bar represents the **Benign(B)** class.

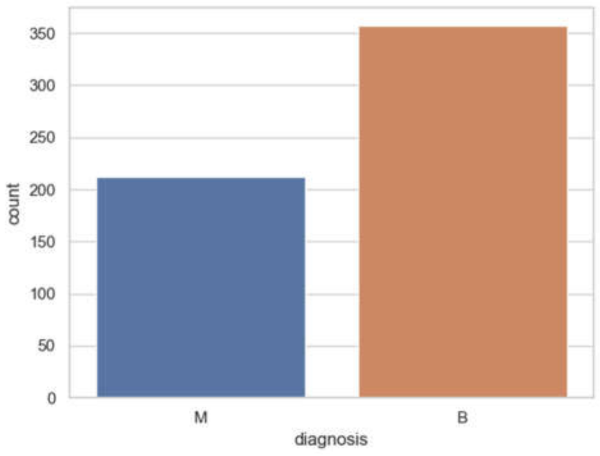
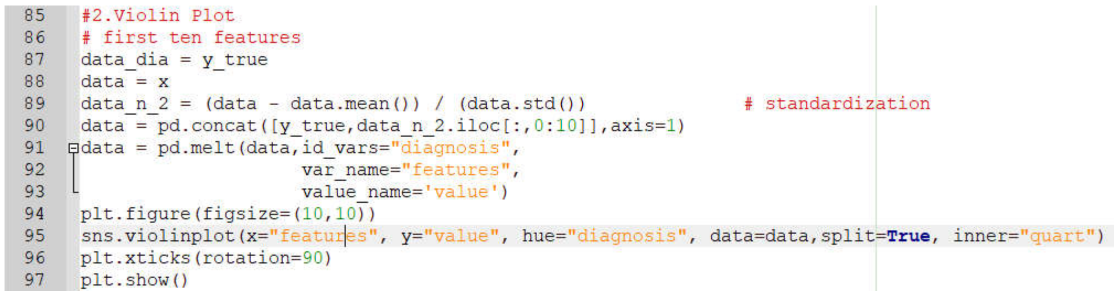


Fig. 2.13

- ## Violin Plot



```
85  #2.Violin Plot
86  # first ten features
87  data_dia = y_true
88  data = x
89  data_n_2 = (data - data.mean()) / (data.std())          # standardization
90  data = pd.concat([y_true,data_n_2.iloc[:,0:10]],axis=1)
91  data = pd.melt(data,id_vars="diagnosis",
92                      var_name="features",
93                      value_name='value')
94  plt.figure(figsize=(10,10))
95  sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="quart")
96  plt.xticks(rotation=90)
97  plt.show()
```

Fig. 2.14

The features of the dataset are going to be plotted and visualised. The set has 30 features to study. For sake of simplicity, it is grouped them into 3 groups having 10 features each. The above given code helps to plot a violin plot for the first ten features. For plotting the second ten plots, the value in line 90 has to be changed from [:,0:10] to [:,10:20]. Eventually, for plotting the third set, the value is changed to [:,20:31].

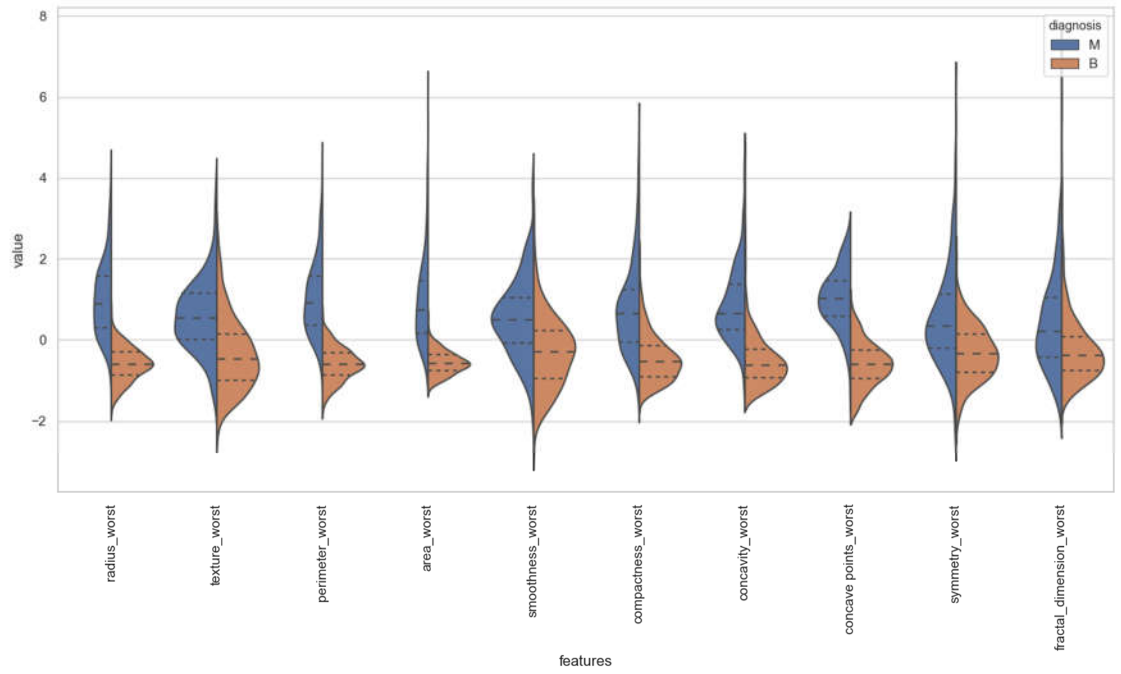
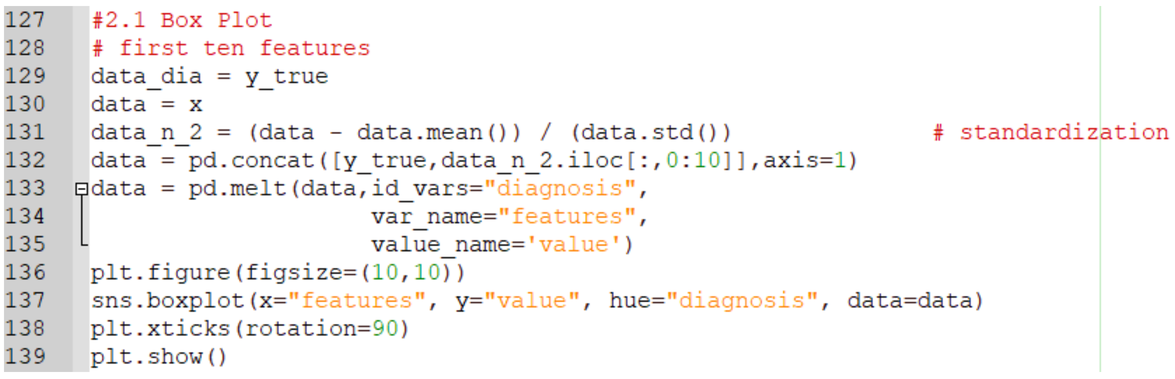


Fig. 2.15

For instance, the above diagram shows the plot for last ten features of the data set. If we look carefully, the Benign and Malignant medians of the **perimeter_worst** and **concave points_worst** are very well separated. This indicates that these two features will show very good results in my classification task. Whereas, such medians of **symmetry_worst** and **fractal_dimension_worst** are not far enough and will hinder good performances in classifications.

- ## Box Plot



```
127  #2.1 Box Plot
128  # first ten features
129  data_dia = y_true
130  data = x
131  data_n_2 = (data - data.mean()) / (data.std())            # standardization
132  data = pd.concat([y_true,data_n_2.iloc[:,0:10]],axis=1)
133  data = pd.melt(data,id_vars="diagnosis",
134                      var_name="features",
135                      value_name='value')
136  plt.figure(figsize=(10,10))
137  sns.boxplot(x="features", y="value", hue="diagnosis", data=data)
138  plt.xticks(rotation=90)
139  plt.show()
```

Fig. 2.16

The features of the dataset are going to be plotted and visualised. The set has 30 features to study. For sake of simplicity, it is grouped them into 3 groups having 10 features each. The above given code helps to plot a violin plot for the first ten features. For plotting the second ten plots, the value in line 133 has to be changed from [:,0:10] to [:,10:20]. Eventually, for plotting the third set, the value is changed to [:,20:31].

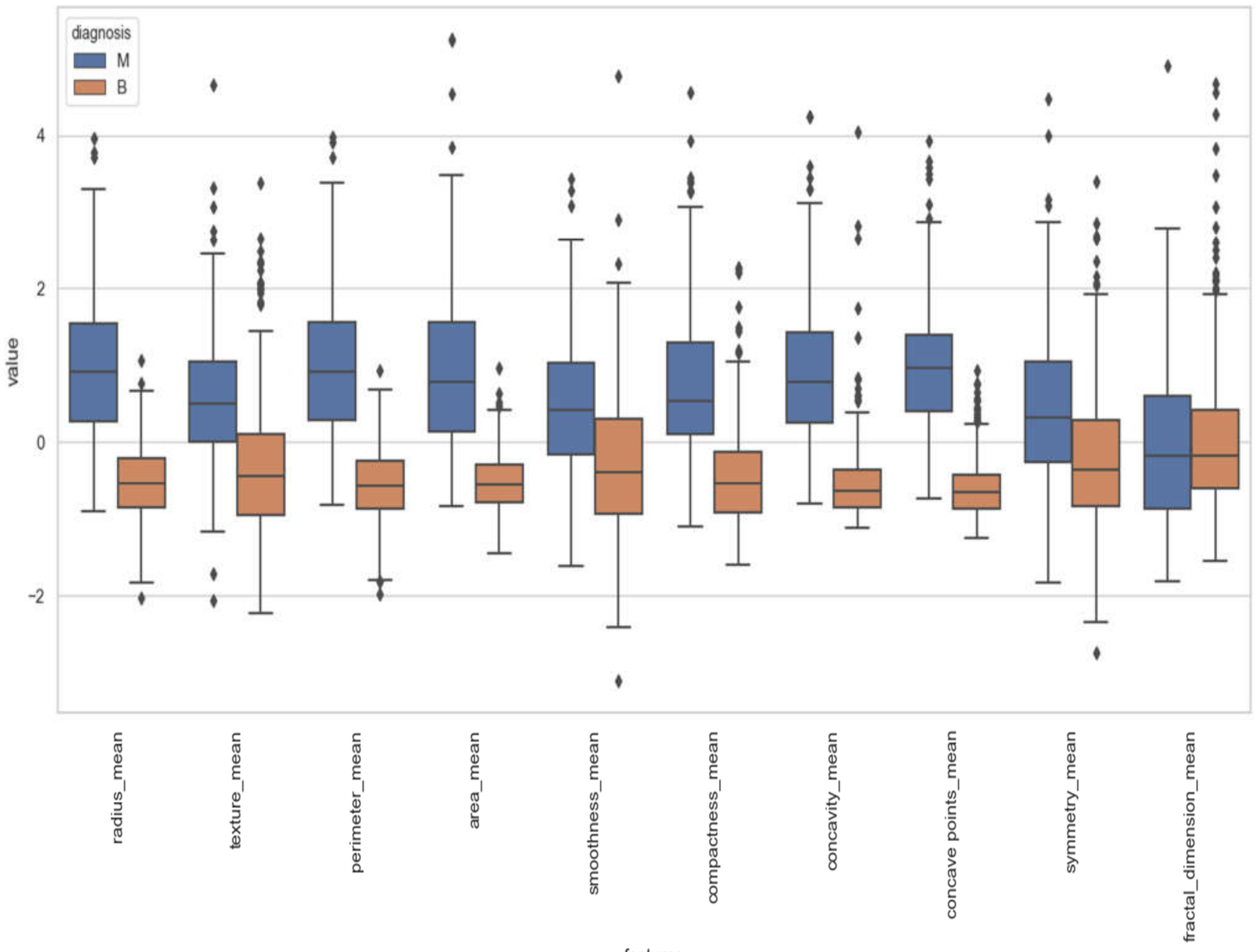
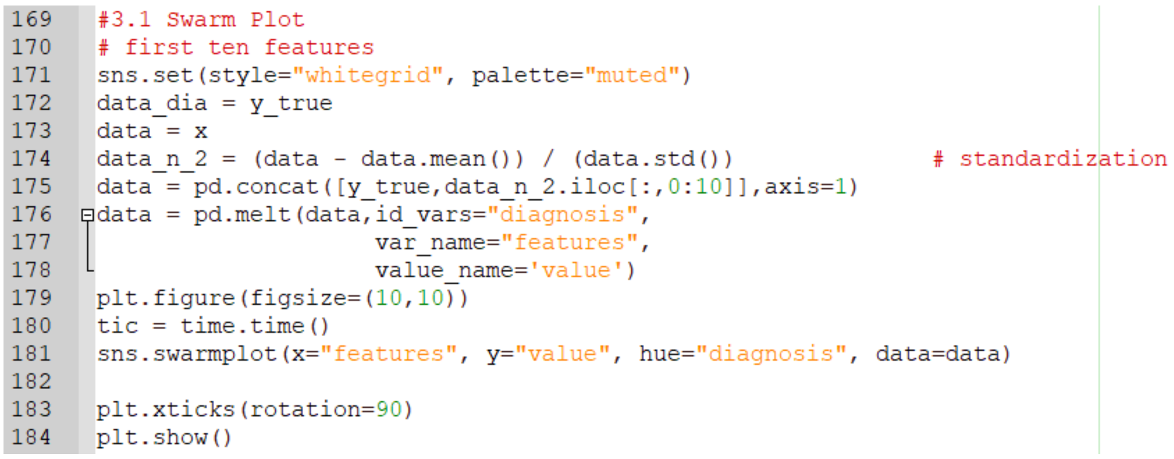


Fig. 2.17

In box plots, the classes for each of the features are visualized as boxes. Length of the boxes determines the density of the datapoints for that class in a particular feature. For this figure, the Benign of **concavity_mean** and **concave points_mean** are almost the same. Hence, they will cause hindrance in classification. Similar observations can be commented about **perimeter_mean** and **area_mean**. Among the features, such plots are really important for judging the extent of a particular class in terms of length.

- ## Swarm Plot



```
169 #3.1 Swarm Plot
170 # first ten features
171 sns.set(style="whitegrid", palette="muted")
172 data_dia = y_true
173 data = x
174 data_n_2 = (data - data.mean()) / (data.std())          # standardization
175 data = pd.concat([y_true,data_n_2.iloc[:,0:10]],axis=1)
176 data = pd.melt(data,id_vars="diagnosis",
177                     var_name="features",
178                     value_name='value')
179 plt.figure(figsize=(10,10))
180 tic = time.time()
181 sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
182 
183 plt.xticks(rotation=90)
184 plt.show()
```

Fig. 2.18

The features of the dataset are going to be plotted and visualised. The set has 30 features to study. For sake of simplicity, it is grouped them into 3 groups having 10 features each. The above given code helps to plot a violin plot for the first ten features. For plotting the second ten plots, the value in line 175 has to be changed from [:,0:10] to [:,10:20]. Eventually, for plotting the third set, the value is changed to [:,20:31].

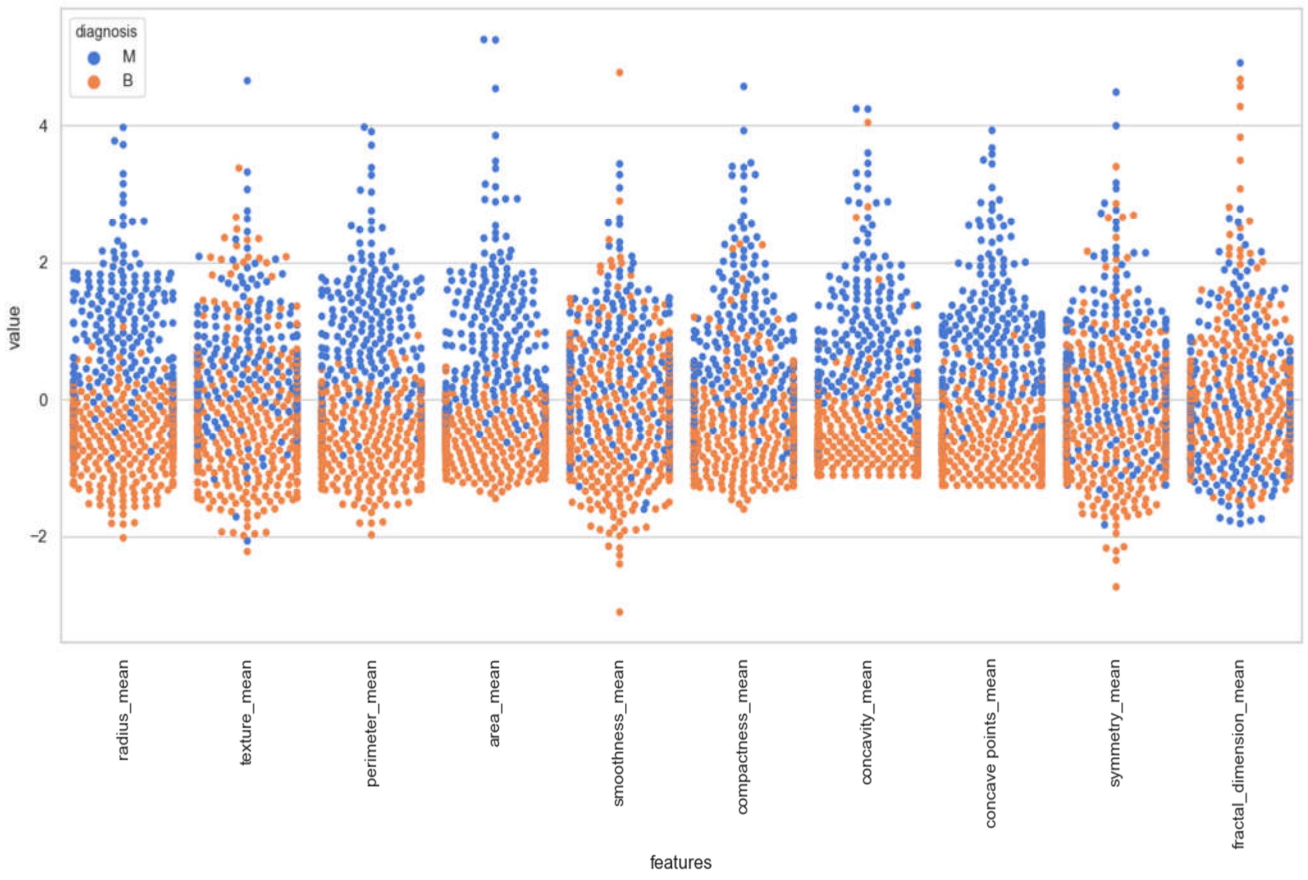


Fig. 2.19

The swarm plots give a proper idea about variances in the features of a dataset. Out of the other two plots, this plot gives a clear vision about the classification tasks related to the dataset. As per the above figure, the classes of **concavity_mean** and **area_mean** are mostly fit for a classification task. Whereas, the datapoints of **smoothness_mean** and **symmetry_mean** is totally in a haphazard form, in terms of class.

3. CLUSTERING

The concept of clustering falls under unsupervised machine learning. There might be some times when we do not have the data separated into classes. At that time, the algorithms under clustering helps to gather the unknown data points into set of clusters. Hence, the machine is not fed with the labels of any dataset and as an output it gets predicted labels. In this assignment, the **y_true** variable is avoided from the dataset. Only the values of **x** are fed to the clustering algorithms, in order to get new set of labels. The output which is generated from any clustering method is technically termed as the ground truth. There are a lot of algorithms under clustering, for example K-Means, Affinity Propagation, Birch, DBSCAN and many more.

3.1. Comparison of Clustering Algorithm

There are many types of clustering algorithms. The dataset is used to plot the scatter plots for all types. Visually judging the scatter plots can help choosing the best method to carry out and evaluate further. This will also tell about the execution time of the algorithm on the dataset. It is the first step to go forward with the right set of algorithms.

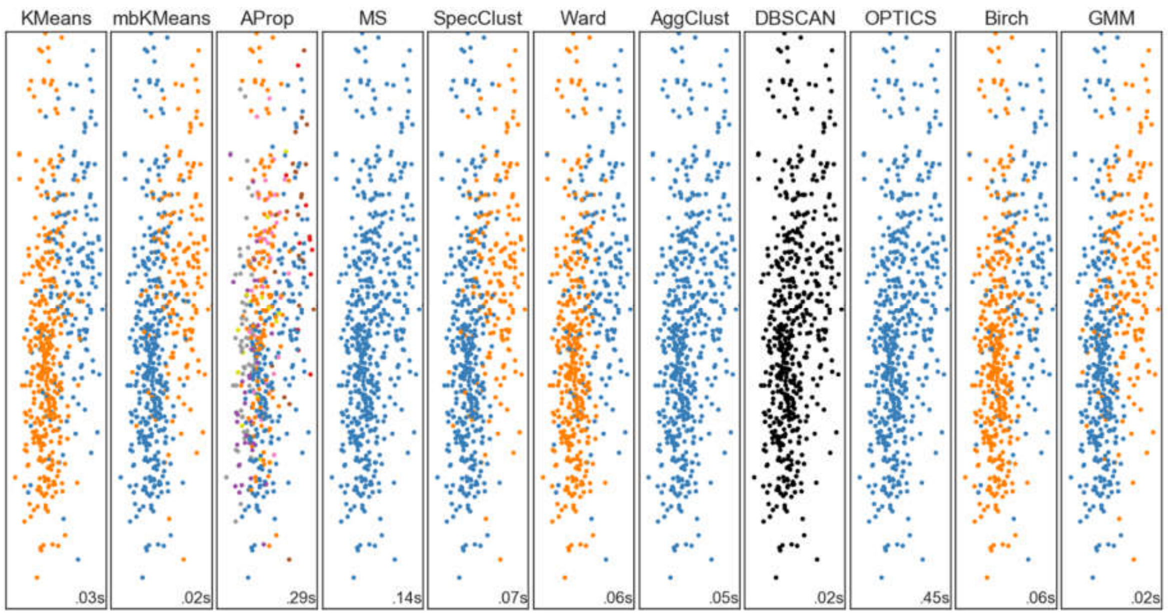


Fig. 3.1

Figure 2.0 is an output of the **clustering_algos.py** python file. The figure shows the scatter plots of all the clustering algorithms applied on the breast cancer dataset. In the flat type clustering methods, the number of desired clusters are already set to 2. This is because it is already known that the purpose of our whole dataset is to classify an instance as Malignant(M) or Benign(B). Ample amount of observations can be made from the graphs. Hence, they are as follows:

- **K-Means:** From the graph, it is easy to distinguish between the orange and the blue coloured clusters. Even the density of the orange clusters is pretty higher than the blue ones. There are pretty less data points which fall into the regions of the different clusters. The execution time of this algorithm is 0.03 seconds. Its pretty fast than the others. Hence, further investigation can be carried out with K-Means clustering method.
- **Mini Batch K-Means:** This method is just an extension of the K-Means which executes the code in much less time than the original algorithm. This is because it executes in small batches of data termed as mini batches. This algorithm also has an output similar to the previous one. So, this can be used for further evaluation due to less execution time.
- **Affinity Propagation:** In this case, the number of clusters cannot be set from before. Hence, from the graph it can be seen that different coloured clusters have formed.

This algorithm did not provide the required output at all. Hence, this method can be skipped.

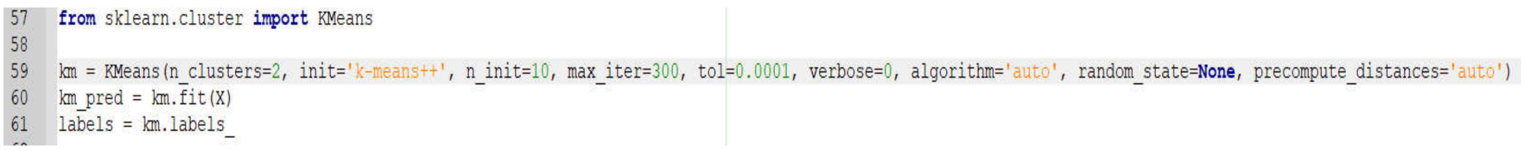
- **Mean Shift:** From the above graph, it can be observed that the algorithm was not able to separate the datapoints into the required number of clusters. Hence, again we can discard this algorithm.
- **Spectral Clustering, Ward, Birch, Gaussian Mixture:** From the graph, it is easy to distinguish the plotted clusters very clearly. The execution time is 0.07, 0.06, 0.06 and 0.02 seconds respectively, which is fair enough. Hence, any of these methods can be chosen
- **Agglomerative Clustering, DBSCAN, OPTICS:** These methods fail to distinguish the clusters within the given datapoints. Hence, these methods cannot be further included in our machine learning pipeline.

3.2. K-Means

K-Means is one of the flat clustering methods in which the user has to define the number of clusters from before. This algorithm tries to group up data points having similar variance and also tries to reduce its inertia. The algorithm can be implemented on a large number of samples and is widely used in large number of applications. It has an objective to choose the centroids that minimise the inertia. Hence, it can be mathematically defined as,

$$\sum_{i=0}^{n} \min_{\mu_j \epsilon C}(||x_i - \mu_j||^2)$$

3.2.1. Implementing on the given dataset

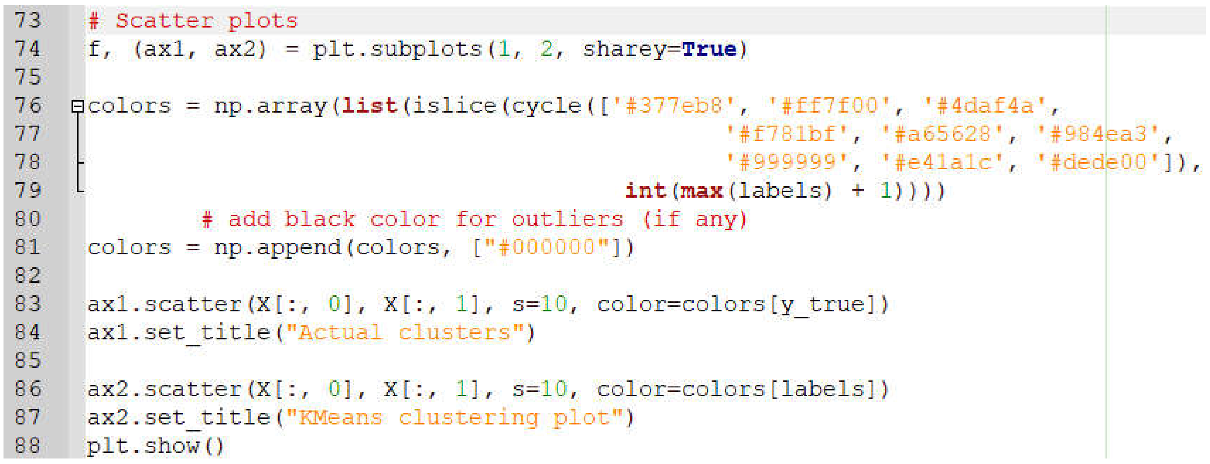


```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, algorithm='auto', random_state=None, precompute_distances='auto')
km_pred = km.fit(X)
labels = km.labels_
```

Fig. 3.2

The above set of codes are written in order to implement the K-Means algorithm on the given dataset. Previously I had mentioned that the dataset was divided into two-parts x and y_true. The list x contains the data of different features and the y_true contains the diagnosis status or the labels for x. It is obvious that the list y_true should not be considered in any of the clustering algorithms. It will only be required during the evaluation.

In line 57, the K-Means library is being imported from sklearn. The parameters for the algorithm can be fed with arguments, through line 59. The code at line 60 helps us to use a method known as fit. This means that the dataset is being fitted in the algorithm and the k-means algorithm is being implemented. The list variable **labels** store the predicted labels which serves as the output for our entire operation.



```
# Scatter plots
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                      '#f781bf', '#a65628', '#984ea3',
                                      '#999999', '#e41a1c', '#dede00']),
                               int(max(labels) + 1))))
        # add black color for outliers (if any)
colors = np.append(colors, ["#000000"])

ax1.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_true])
ax1.set_title("Actual clusters")

ax2.scatter(X[:, 0], X[:, 1], s=10, color=colors[labels])
ax2.set_title("KMeans clustering plot")
plt.show()
```

Fig. 3.3

Now the algorithm has been implemented and the output labels have been temporarily stored. The library matplotlib will help us plotting the results in a scatter plot. The above written code helps us to plot the actual clusters and the clusters obtained from K-means.

3.2.2. Observations and Scoring

The K-Means algorithm is applied on the given dataset. First, the model was run with the default configuration and the ground truth, graphs and the scores are obtained. Parameters were changed to bring fruitful results. The following observations are written below.

TABLE 3.1

K-Means Configuration	Scores (Clustering Metrics)

n_clusters=2, init='k-means++', n_init=10, max_iter=200, tol=0.0001, verbose=0, algorithm='auto', random_state=None, precompute_distances='auto' (***default***)	Accuracy Score = **0.9050**
	Adjusted Rand index = **0.654**
	Homogeneity = **0.525**
	Completeness = **0.54**
	V-Measure = **0.53**

n_clusters=2, init='k-means++', n_init=90, max_iter=300, tol=0.0001, verbose=0, algorithm='auto', random_state=None, precompute_distances='auto' (***best***)	Accuracy Score = **0.9103**
	Adjusted Rand index = **0.671**
	Homogeneity = **0.544**
	Completeness = **0.565**
	V-Measure = **0.555**

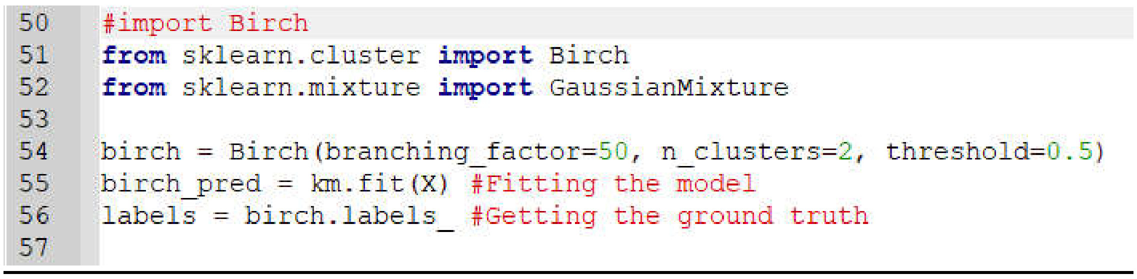
The n_init and max_iter parameters were changed to bring worthy results. The second attempt did really increase the accuracy and the adjust rand index value.

Furthermore, discussions have been under the Discussion and Conclusions heading in part 5 of the report.

3.3. Birch

The Birch algorithm works by building a tree known as the Clustering Feature Tree (CFT) for the given dataset. The dataset is compressed to Clustering Nodes (CN). Data tends to get lost in this process. The Clustering Nodes also have sub clusters called Clustering Features (CF) sub clusters. The algorithm has two parameters, the threshold and the branching factor. The branching factor limits the number of sub clusters in a node and the threshold limits the distance between the entering sample and the existing sub clusters.

3.3.1. Implementing on the given dataset



```
50  #import Birch
51  from sklearn.cluster import Birch
52  from sklearn.mixture import GaussianMixture
53  
54  birch = Birch(branching_factor=50, n_clusters=2, threshold=0.5)
55  birch_pred = km.fit(X) #Fitting the model
56  labels = birch.labels_ #Getting the ground truth
57  
```

Fig. 3.4

The above set of codes are written to implement the Birch algorithm on our existing dataset. Just like the previous method, the same protocol of splitting the data into x and y_true is obeyed. Line 54 helps to set the parameters for this algorithm. As it was introduced earlier, this algorithm has got only 2 parameters to play with. Keeping in mind that the number of desired clusters are 2. The **labels** variable in line 56 will temporarily store the predicted ground truth from the model.

3.3.2. Observations and Scoring

Just like the previous task, the Birch algorithm is applied on the given dataset. First, the model was run with the default configuration and the ground truth, graphs and the scores are obtained. Parameters were changed to bring fruitful results.The following observations are written below.

TABLE 3.2

Birch Configurations	Scores (Clustering Metrics)

n_clusters=2, branching_factor=50, threshold=0.5, copy=True (***default***)	Accuracy Score = **0.1195**
	Adjusted Rand index = **0.575**
	Homogeneity = **0.446**
	Completeness = **0.468**
	V-Measure = **0.457**

n_clusters=2, branching_factor=50, threshold=0.5, copy=True (***configuration1***)	Accuracy Score = **0.7785**
	Adjusted Rand index = **0.287**
	Homogeneity = **0.262**
	Completeness = **0.408**
	V-Measure = **0.319**

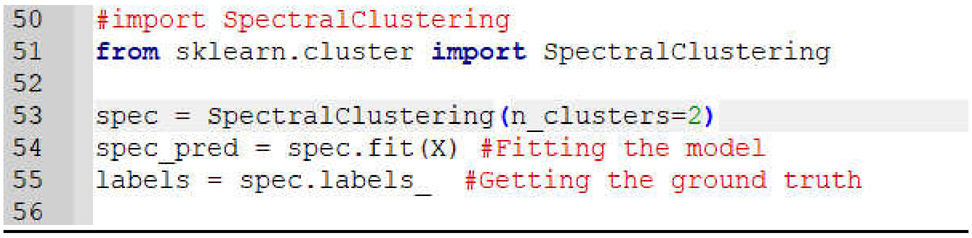
Both the observations have poor results. Furthermore, discussions have been made under the Discussion and Conclusions heading in part 5 of the report.

3.4. Spectral Clustering

This algorithm performs a low-dimension embedding of the affinity matrix between the given samples, followed by a simple clustering, for example by K-Means, of the components of the eigenvectors in the low dimension space. It is especially computationally efficient if the affinity matrix is sparse and the amg solver is used for the eigenvalue problem.

The existing version of Spectral Clustering requires the number of clusters to be specified in advance. It works well for a small number of clusters, but is not advised for a large number of clusters. In our case, we will only be using 2.

3.4.1. Implementing on the given dataset



```
#import SpectralClustering
from sklearn.cluster import SpectralClustering

spec = SpectralClustering(n_clusters=2)
spec_pred = spec.fit(X) #Fitting the model
labels = spec.labels_  #Getting the ground truth

```

Fig. 3.5

The above set of codes are written to implement the Spectral Clustering algorithm on our existing dataset. Just like the previous other methods, I have followed the same protocols of splitting the data into x and y_true.

3.4.2. Observations and Scoring

Just like the previous clustering tasks, the observations are given below.

TABLE 3.3

Spectral Clustering Configurations	Scores (Clustering Metrics)

n_clusters=2, eigen_solver=None, n_components=None, random_state=None, n_init=10, gamma=0.7, affinity='nearest_neighbors', n_neighbors=30, eigen_tol=0.001, assign_labels='kmeans', degree=5, coef0=2, kernel_params=None, n_jobs=None (***default***)	Accuracy Score = **0.63**
	Adjusted Rand index = **0.005**
	Homogeneity = **0.005**
	Completeness =**0.149**
	V-Measure = **0.010**

n_clusters=2, eigen_solver=None, n_components=None, random_state=None, n_init=10, gamma=1.0, affinity='rbf', n_neighbors=10, eigen_tol=0.0, assign_labels='kmeans', degree=3, coef0=1, kernel_params=None, n_jobs=None(***best***)	Accuracy Score = **0.938**
	Adjusted Rand index = **0.767**
	Homogeneity = **0.653**
	Completeness = **0.673**
	V-Measure = **0.663**

For better results, the gamma, affinity, n_neighbors, eigen_tol, degree and coef0 parameters have been altered.

Eventually, the later model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

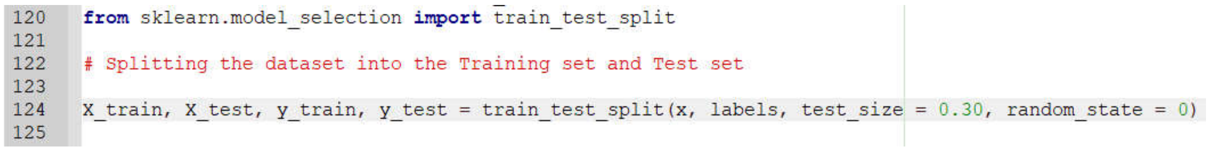
4. CLASSIFICATION

Classification is one of the major categories under supervised machine learning. It is a problem when the desired output is in the form of a category or a class such as “black” and “white” or “diagnosis positive” and “diagnosis negative”. The assignment data set is based on a medical field, the classes are Malignant(M) and Benign(B). A classification model helps to draw a conclusion based on observing values. Hence, the model is passed with both the x and the labels. The classification model is generally termed as a classifier. There are a number of classification algorithms such as Support Vector Machines (SVM), Decision Tree, Random Forests and many more. Since a certain pipeline is obeyed for this report. The ground truth will be taken from the best selected Clustering method and feed it as the input in the classifier.

Before fitting it into the classifier, the dataset and ground truth need to be split into two parts, the **train** and the **test** data. The classifier is only fed with the train dataset and further tested on the test dataset. While training, the model should not be exposed to the test dataset. Exposing the test dataset is forbidden and it is termed as cheating. This concept is just like a learning process in a school. In the school, the students are taught certain lessons. The students train themselves from those lessons and then they are allowed to give a test based on those lessons. The questions appearing on the class test are never exposed during the training process. This protocol is followed in Machine Learning because in this developing world, data needs to be assessed correctly and in an unbiased way. Or else, it can cost anybody’s life or any valuable thing can get stolen without knowing.

The train-test split can be done in two ways, they are as follows:

- **The train_test_split method:** In this case, the data is divided in terms of percentage. The python syntax for this approach is given below

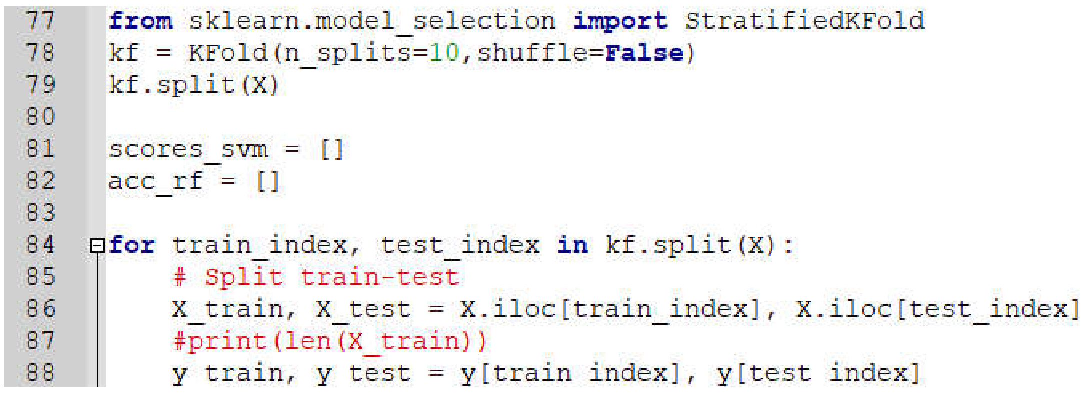


```
from sklearn.model_selection import train_test_split

# Splitting the dataset into the Training set and Test set

X_train, X_test, y_train, y_test = train_test_split(x, labels, test_size = 0.30, random_state = 0)

```

Fig. 4.1

In the assignment, the train and the test values will be 70% and 30% respectively.

- **K-Folds Cross Validation split:** In this case, the data is divided in K number of sets or folds. The first set is kept for validation while the (K-1) sets are kept for training. This process is repeated K times to get K number of accuracies. At last the average accuracy is taken into consideration. The python syntax for this approach is given below



```
from sklearn.model_selection import StratifiedKFold
kf = KFold(n_splits=10,shuffle=False)
kf.split(X)

scores_svm = []
acc_rf = []

for train_index, test_index in kf.split(X):
    # Split train-test
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    #print(len(X_train))
    y train, y test = y[train index], y[test index]
```

Fig. 4.2

For this assignment, the K is set to 10.

4.1. Support Vector Machines

4.1.1. Observations and Results

- **Using Train-Test Split:**

The SVM classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.1

SVM Configuration	Metrics

C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=True, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None (***default***)	Accuracy = **0.9766**
	Balanced Accuracy = **0.9704**
	F1-Score = **0.9766**
	Recall = **0.9704**
	Precision = **0.9781**
	ROC AUC = **0.9972**
	Confusion Matrix = $\begin{bmatrix} \mathbf{110} & \mathbf{1} \\ \mathbf{3} & \mathbf{57} \end{bmatrix}$

C=1.0, kernel='sigmoid', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=True, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',	Accuracy = **0.9415**
	Balanced Accuracy = **0.9319**
	F1-Score = **0.9415**
	Recall = **0.9319**
	Precision = **0.9389**
	ROC AUC = **0.9854**

break_ties=False, random_state=None (***another configuration)***	Confusion Matrix = $\begin{bmatrix} \mathbf{107} & \mathbf{4} \\ \mathbf{6} & \mathbf{54} \end{bmatrix}$

To check for better scores, the kernel parameter was altered. All the kernels have almost the same result to the default, except for sigmoid. The rbf kernel model has a more ROC score and a better confusion matrix.

Eventually, the default model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

- ## Using K-Folds:

The SVM classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.2

SVM Configuration	**Metrics**

C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=True, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None (***default***)	Accuracy = **0.8999**
	Balanced Accuracy = **0.8713**
	F1-Score = **0.8472**
	Recall = **0.77**
	Precision = **0.9517**
	ROC AUC = **0.9599**

C=1.0, kernel='sigmoid', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=True, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None (***another configuration)***	Accuracy = **0.8999**
	Balanced Accuracy = **0.8713**
	F1-Score = **0.8472**
	Recall = **0.77**
	Precision = **0.9517**
	ROC AUC = **0.9601**

To check for better scores, the kernel parameter was altered. All the kernels have almost the same result to the default, except for sigmoid. The sigmoid kernel model has a more ROC score.

Eventually, the other model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

4.2. AdaBoost

4.2.1. Observations and Results

- **Using Train-Test Split:**

The AdaBoost classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.3

AdaBoost Configuration	Metrics

base_estimator=None, n_estimators=10, learning_rate=1.0, algorithm='SAMME.R', random_state=None (***default***)	Accuracy = **0.9766**
	Balanced Accuracy = **0.9704**
	F1-Score = **0.9766**
	Recall = **0.9704**
	Precision = **0.9781**
	ROC AUC = **0.9954**
	Confusion Matrix = $\begin{bmatrix} \mathbf{110} & \mathbf{1} \\ \mathbf{3} & \mathbf{57} \end{bmatrix}$

base_estimator=None, n_estimators=100, learning_rate=0.5, algorithm='SAMME', random_state=None (***best***)	Accuracy = **0.9766**
	Balanced Accuracy = **0.966**
	F1-Score = **0.9766**
	Recall = **0.966**
	Precision = **0.9826**
	ROC AUC = **0.9975**
	Confusion Matrix = $\begin{bmatrix} \mathbf{111} & \mathbf{0} \\ \mathbf{4} & \mathbf{56} \end{bmatrix}$

To check for better scores, the algorithm, n_estimators and learning_rate parameters was altered. This change improved the ROC score to **0.9975** and gave a better confusion matrix.

Hence, the later model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

- ## Using K-Folds:

The AdaBoost classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.4

AdaBoost Configuration K = 10	Metrics
base_estimator=None, n_estimators=10, learning_rate=1.0, algorithm='SAMME.R', random_state=None (***default***)	Avg. Accuracy = **0.9719** Avg. Balanced Accuracy = **0.9702** Avg. F1-Score = **0.9586** Avg. Recall = **0.9555** Avg. Precision = **0.9646** Avg. ROC AUC = **0.9950**
base_estimator=None, n_estimators=90, learning_rate=0.5, algorithm='SAMME', random_state=None (***best***)	Avg. Accuracy = **0.9753** Avg. Balanced Accuracy = **0.9686** Avg. F1-Score = **0.96** Avg. Recall = **0.9473** Avg. Precision = **0.9748** Avg. ROC AUC = **0.9981**

To check for better scores, the algorithm, n_estimators and learning_rate parameters was altered. This change improved the ROC score to **0.9975**.

Hence, the later model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

4.3. Gaussian Naive Bayes (Gaussian NB)

4.3.1. Observations and Results

- **Using Train-Test Split:**

The Gaussian NB classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.5

Gaussian NB Configuration	Metrics

var_smoothing = 1e-9 (***default***)	Accuracy = **0.9649**
	Balanced Accuracy = **0.9691**
	F1-Score = **0.9649**
	Recall = **0.9691**
	Precision = **0.9562**
	ROC AUC = **0.9972**
	Confusion Matrix = $\begin{bmatrix} \mathbf{59} & \mathbf{1} \\ \mathbf{5} & \mathbf{106} \end{bmatrix}$

Changing the value of var_smoothing gave similar scores.

Hence, the default model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

- **Using K-Folds:**

The Gaussian NB classifier is implemented on the given dataset. First, the default configuration is used to generate the results and the metric scores. Then the scores are improved by changing the parameters of the configuration. The observations are given below.

TABLE 4.6

Gaussian NB Configuration (K = 10)	**Metrics**
var_smoothing = 1e-9 (***default***)	Avg. Accuracy = **0.9667** Avg. Balanced Accuracy = **0.9676** Avg. F1-Score = **0.9488** Avg. Recall = **0.9656** Avg. Precision = **0.9385** Avg. ROC AUC = **0.9948**
var_smoothing = 1e-13 (***best***)	Avg. Accuracy = **0.9736** Avg. Balanced Accuracy = **0.9790** Avg. F1-Score = **0.9788** Avg. Recall = **0.9620** Avg. Precision = **0.9967** Avg. ROC AUC = **0.9949**

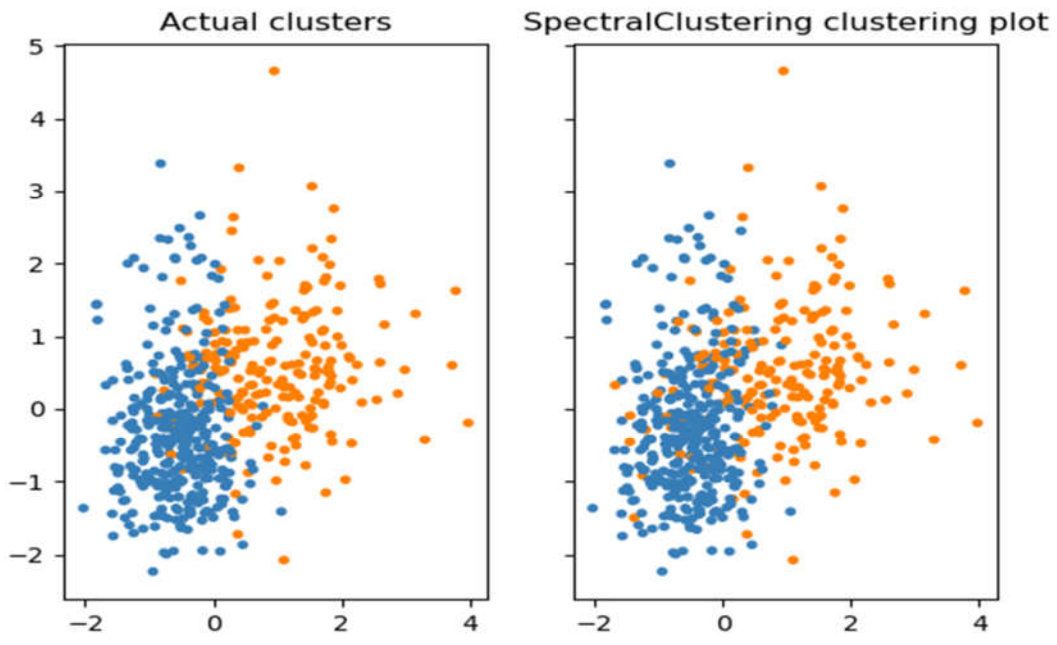
Changing the value of var_smoothing changed the scores. This change improved the ROC score to **0.9949** and accuracy to **0.9736**.

Hence, the later model is chosen for further discussion under the Classification section of Discussion and Conclusion heading.

5. DISCUSSION AND CONCLUSION

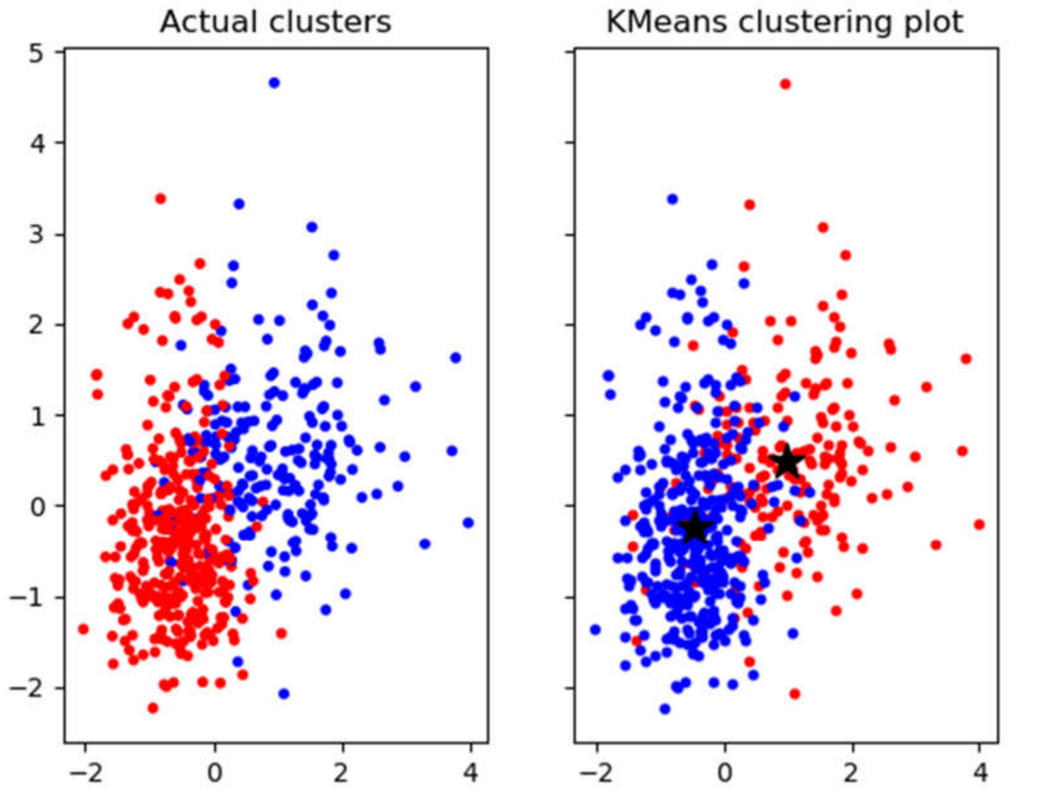
5.1. Clustering

The given dataset can majorly have two kind of classes termed as Benign and Malignant. Eventually the number of clusters in the assignment will be taken as two right from the beginning. Observations have already been tabulated for three different clustering methods and the best configuration is selected out of each. The birch method should undergo rejection if the scores of the best configurations are compared. This is because all the vital scores of birch method are very poor based on this dataset. Birch method shows good results for datasets having large number of instances and data points. It causes clustering problems in densely populated clustered regions. Hence, birch is not a good option for this dataset. Comparison of the scatter plots of the other two methods are done below.



Actual clusters
SpectralClustering clustering plot

Fig. 5.1



Actual clusters
KMeans clustering plot

Fig. 5.2

From the above output plots, it is really hard to select the best clustering plot. This is because in both the plots, the boundary region of both the classes has a mixture of varied datapoints. Both the clustering methods can be compared using the scores. As per the table and table, the scores for spectral clustering are much better. Hence, the best configuration of Spectral clustering can be taken as the best method for this dataset. Eventually the ground truth is passed to the next stage of the pipeline.

5.2. Classification

It is known from the basics that tasks under classification requires the feature data and the ground truth. For this assignment a pipeline is followed, and the ground truth is obtained from the best clustering configuration. As per the previous discussion, the best configuration of Spectral Clustering was selected. This ground truth is much fair enough for classification. This task has been performed in two parts. One is the train-test split protocol and the other is the K-folds split. The first type is being considered and the results are obtained for three types of classification. This assignment as three types of classifiers, SVM, AdaBoost and Gaussian NB. Each classifier undergoes evaluation processes and can be compared in terms of scores. At least its performance can be plotted down as a Receiver Operating Characteristic curve. The ROC curves of the 3 classifiers with their best configurations are given below.

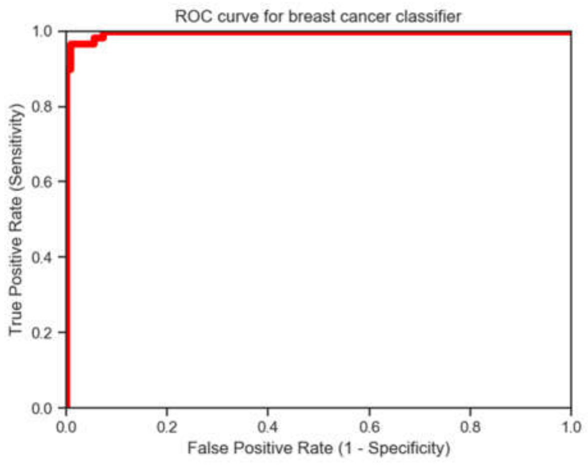


Fig. 5.3

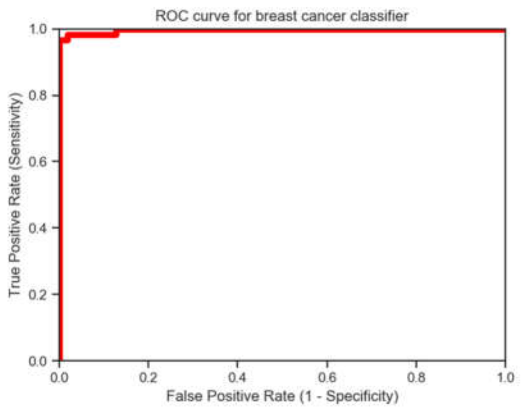


Fig. 5.4

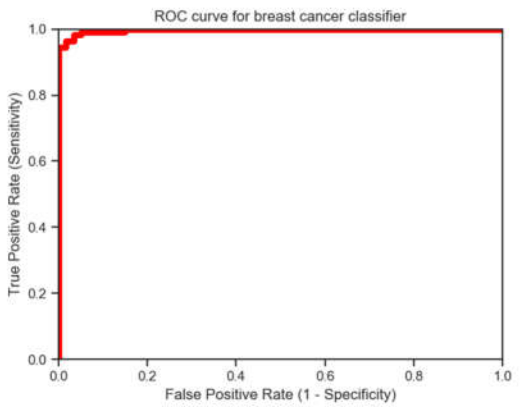


Fig. 5.5

The above figures represent the ROC plots for the best configurations for each of the selected classifiers. Figure 5.3 is the plot for SVM, Figure 5.4 is for AdaBoost and the last is for Gaussian NB. As per theory, the perfect curve should have a proper inverted 'L' shape with the area under curve as 1.0. It is much visible that, the area outside the curve is the most for SVM, less for Gaussian NB and least for AdaBoost. Hence, the performance for AdaBoost is the best for this assignment.

For the K-Fold type of split, 10 values of answers are obtained for each of the scores of a classifier. This is because the number of K folds selected are 10. Hence, the average of the 10 values forms the final score. In reference to table 1, 2 and 3, the scores types are same as the previous type of split. The best configuration of SVM has obtained accuracy and ROC AUC score as **0.8999** and **0.9601** respectively. The best configuration of Gaussian NB has obtained accuracy and ROC AUC score as **0.9736** and **0.9949** respectively. Similarly, the best configuration of the second type has accuracy and ROC AUC scores as **0.9806** and **0.9981**. From the above scores, it is sure that the particular configuration for the AdaBoost classifier has the best performance on this dataset. Hence, the predictions of this model will be **99.81%** correct.

5.3. Conclusion

The classic machine techniques have put a great effect on this data set. At first, the data was loaded from a csv file. Then it was checked and processed for the next stage of the pipeline. All the three selected clustering methods were able to show competitive results for the data sets. But the Spectral clustering method had much better results. To maintain the ethics of the pipeline, the ground truth was passed with the feature data into classification. This third stage was done in two methods. In both the types the AdaBoost method had a well above satisfactory ROC scores and simple confusion matrices for every observation. Hence, this type of classifier can accurately make predictions on new real time instances of data.

6. REFERENCES

Allen, M. (2020). 80. Grouping unlabelled data with k-means clustering. [online] Python for healthcare modelling and data science. Available at: https://pythonhealthcare.org/2018/05/01/80-grouping-unlabelled-data-with-k-means-clustering/ [Accessed 12 Feb. 2020].

Cse.msu.edu. (2020). tutorial4. [online] Available at: http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial4/tutorial4.html [Accessed 11 Feb. 2020].

Kaggle.com. (2020). Breast cancer Prediction using PCA. [online] Available at: https://www.kaggle.com/shreedhar091/breast-cancer-prediction-using-pca [Accessed 11 Feb. 2020].

Kaggle.com. (2020). Breast Cancer Wisconsin (Diagnostic) Data Set. [online] Available at: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data [Accessed 10 Feb. 2020].

Kaggle.com. (2020). Feature Selection and Data Visualization. [online] Available at: https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization#Data-Analysis [Accessed 10Feb. 2020].

Kassani, S. (2020). sara-kassani/Python-Machine-Learning-Codes. [online] GitHub. Available at: https://github.com/sara-kassani/Python-Machine-LearningCodes/blob/master/Visualization%20-%20Breast%20Cancer%20Wisconsin%20(Diagnostic)%20Data%20Set.ipynb [Accessed 10 Feb. 2020].

Learn, S. (2020). Comparing different clustering algorithms on toy datasets â€” scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikitlearn.org/stable/auto_examples/cluster/plot_cluster_comparison.html [Accessed 11 Feb. 2020].

Learn, S. (2020). sklearn.cluster.Birch â€" scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html [Accessed 11 Feb. 2020].
Learn, S. (2020). sklearn.cluster.KMeans â€" scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans [Accessed 11Feb. 2020].
Learn, S. (2020). sklearn.cluster.SpectralClustering â€" scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html [Accessed 11 Feb. 2020].

Learn, S. (2020). 2.3. Clustering â€" scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/clustering.html [Accessed 11 Feb. 2020].
Learn, S. (2020). 2.3. Clustering â€" scikit-learn 0.22.1 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/clustering.html#k-means [Accessed 11 Feb. 2020].
Parekh, V. (2020). Cluster analysis of Breast Cancer dataset. [online] Kaggle.com. Available at: https://www.kaggle.com/vishwaparekh/cluster-analysis-of-breast-cancer-dataset [Accessed 12 Feb. 2020].

Roncen, T. (2020). Breast Cancer data set feature selection. [online] Kaggle.com. Available at: https://www.kaggle.com/quantumofronron/breast-cancer-data-set-feature-selection#II.-Data-Analysis [Accessed 12 Feb. 2020].
Sinha, S. (2020). Visualising the Breast Cancer Wisconsin Data Set. [online] Kaggle.com. Available at: https://www.kaggle.com/djokester/visualising-the-breast-cancer-wisconsin-data-set [Accessed 10 Feb.2020].

codebasics (2019a). *Machine Learning Tutorial Python 12 - K Fold Cross Validation*. *YouTube*. Available at: https://www.youtube.com/watch?v=gJo0uNL-5Qw [Accessed 19 Feb. 2020].

dhavalsays (2019b). *codebasics/py*. [online] GitHub. Available at: https://github.com/codebasics/py/blob/master/ML/12_KFold_Cross_Validation/12_k_fold.ipynb [Accessed 19 Feb. 2020].

financetrain (2020). *K-Fold Cross Validation Example Using Python scikit-learn*. [online] Finance Train. Available at: https://financetrain.com/k-fold-cross-validation-example-python-scikit-learn/ [Accessed 18 Feb. 2020].

geeksforgeeks (2017). *Regression and Classification | Supervised Machine Learning - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ [Accessed 15 Feb. 2020].

Kindson (2019c). *How to Perform K Means Clustering in Python(Step by Step)*. *YouTube*. Available at: https://www.youtube.com/watch?v=fl0PH6uQDIw [Accessed 14 Feb. 2020].

Maklin, C. (2019). *BIRCH Clustering Algorithm Example In Python*. [online] Medium. Available at: https://towardsdatascience.com/machine-learning-birch-clustering-algorithm-clearly-explained-fb9838cbeed9 [Accessed 13 Feb. 2020].

Raheel Shaikh (2018). *Cross Validation Explained: Evaluating estimator performance.* [online] Medium. Available at: https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85 [Accessed 18 Feb. 2020].

Scikit (2018a). *1.4. Support Vector Machines — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/svm.html [Accessed 15 Feb. 2020].

Scikit (2018b). *1.4. Support Vector Machines — scikit-learn 0.22.1 documentation*. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/svm.html#classification [Accessed 15 Feb. 2020].

Scikit (2018c). *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation*. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html [Accessed 17 Feb. 2020].

scikit (2018d). *sklearn.ensemble.AdaBoostClassifier — scikit-learn 0.22.1 documentation*. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html [Accessed 17 Feb. 2020].

scikit (2018e). *sklearn.naive_bayes.GaussianNB — scikit-learn 0.22.1 documentation*. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html [Accessed 17 Feb. 2020].

scikit (2019d). *1.11. Ensemble methods — scikit-learn 0.22.1 documentation*. [online] scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/ensemble.html#adaboost [Accessed 15 Feb. 2020].

scikit (2019e). *sklearn.svm.SVC — scikit-learn 0.22 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC [Accessed 15 Feb. 20202].

stackabuse (2019f). *Understanding ROC Curves with Python*. [online] Stack Abuse. Available at: https://stackabuse.com/understanding-roc-curves-with-python/ [Accessed 19 Feb. 2020].

VanderPlas, J. (2019). *In Depth: k-Means Clustering | Python Data Science Handbook*. [online] Github.io. Available at: https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html [Accessed 12 Feb. 2020].

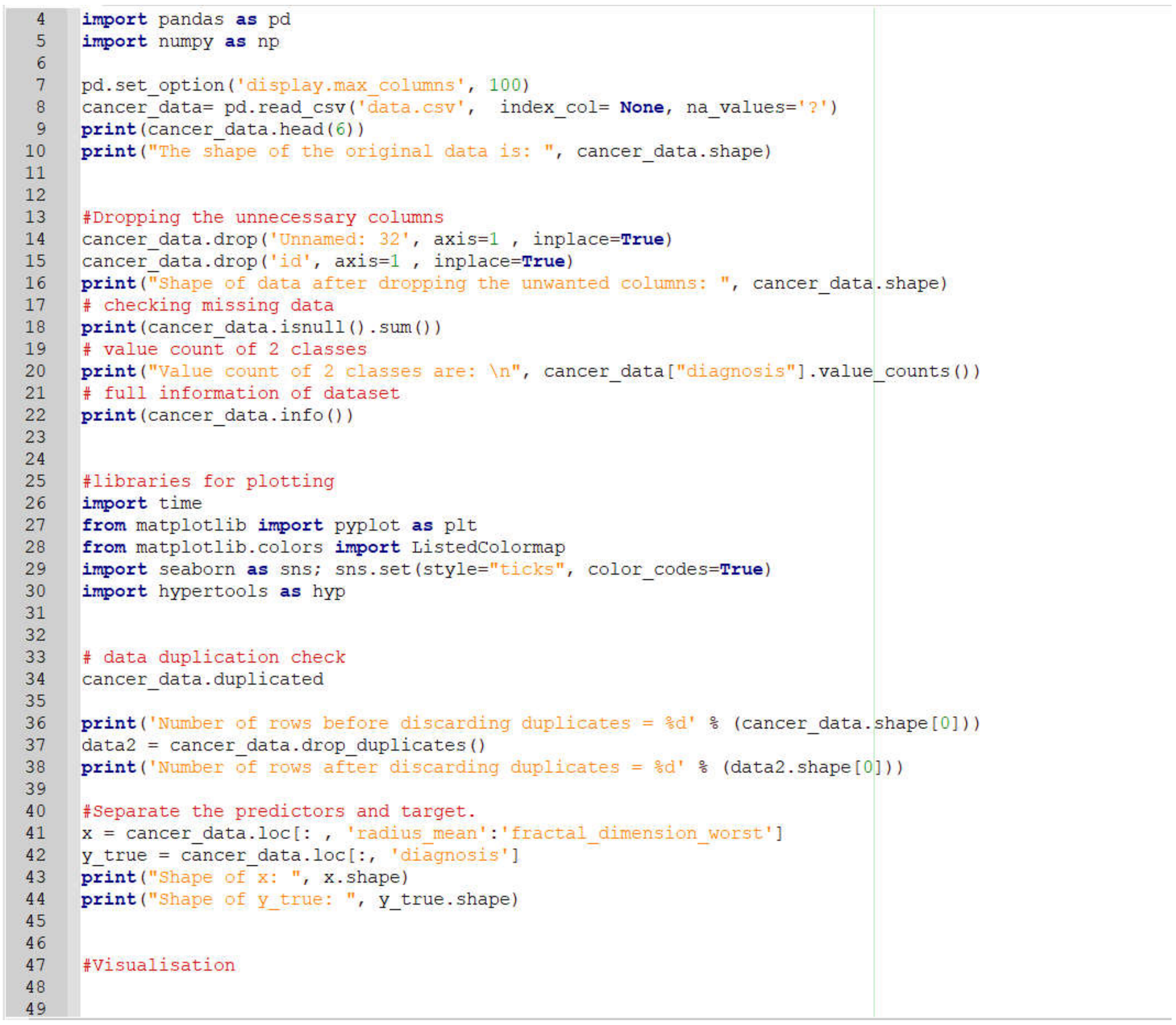
Veda, A. (2020). *ashokveda/youtube_ai_ml*. [online] GitHub. Available at: https://github.com/ashokveda/youtube_ai_ml/blob/master/roc_auc_ml_classification_metrics.ipynb [Accessed 22 Feb. 2020].

vishabh goel (2018). *Building a Simple Machine Learning Model on Breast Cancer Data*. [online] Medium. Available at: https://towardsdatascience.com/building-a-simple-machine-learning-model-on-breast-cancer-data-eca4b3b99fa3 [Accessed 16 Feb. 2020].

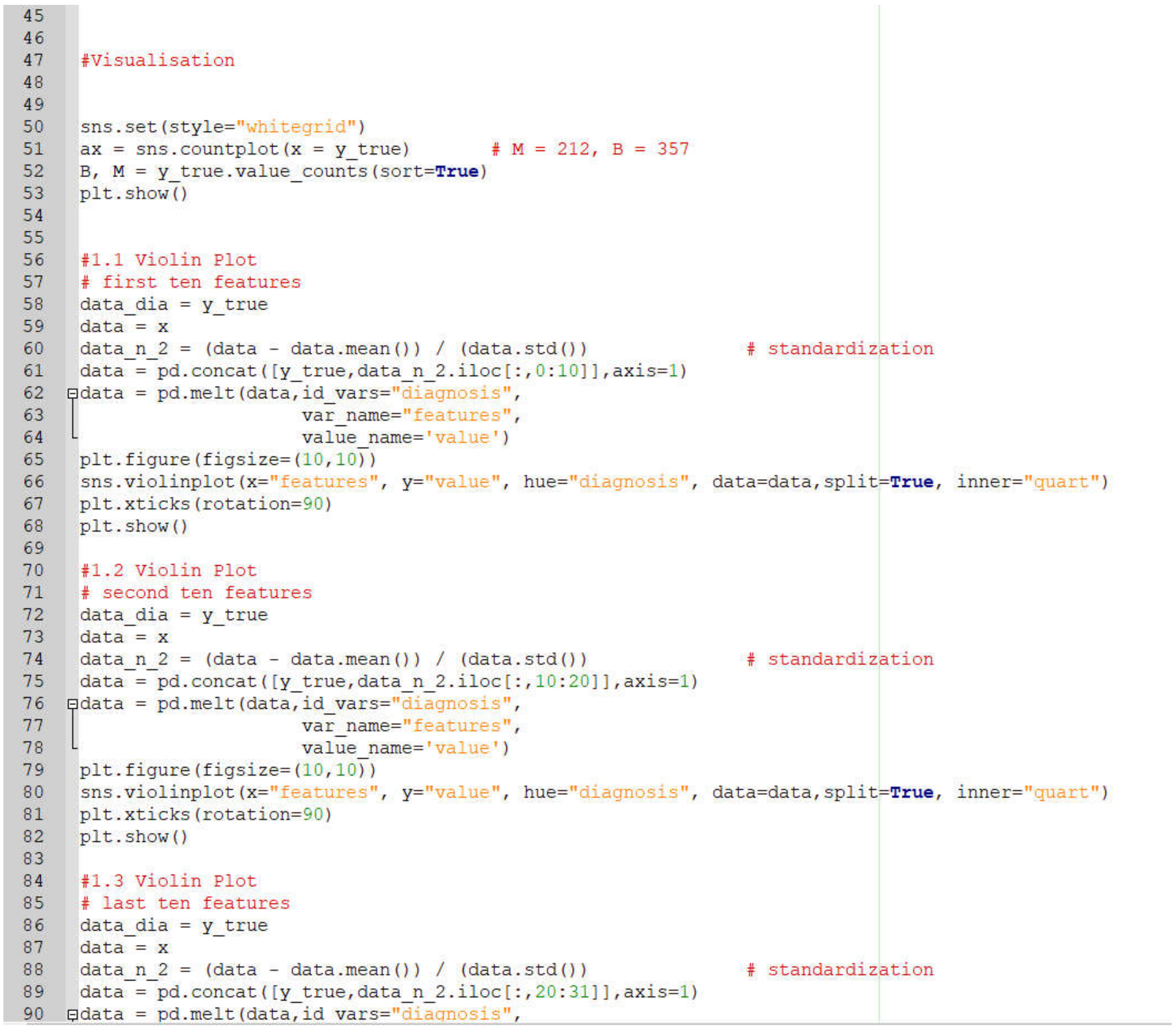
VRINDA (2018f). *Analysis of Breast Cancer Wisconsin Data Set. YouTube*. Available at: https://www.youtube.com/watch?v=SWE9pi4tM9o [Accessed 12 Feb. 2020].

7. APPENDIX

- **PART 1:**

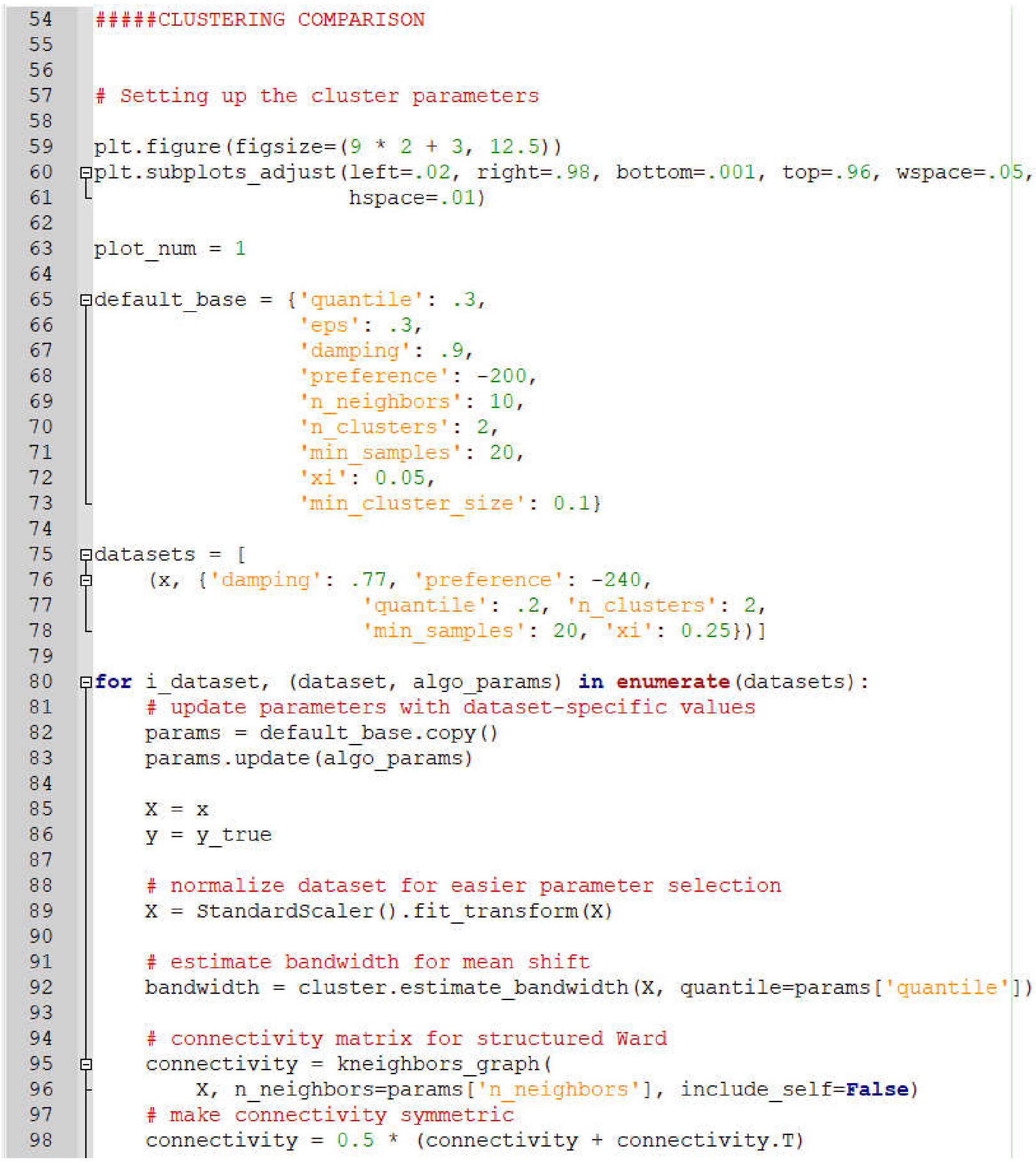


```
import pandas as pd
import numpy as np

pd.set_option('display.max_columns', 100)
cancer_data= pd.read_csv('data.csv',  index_col= None, na_values='?')
print(cancer_data.head(6))
print("The shape of the original data is: ", cancer_data.shape)


#Dropping the unnecessary columns
cancer_data.drop('Unnamed: 32', axis=1 , inplace=True)
cancer_data.drop('id', axis=1 , inplace=True)
print("Shape of data after dropping the unwanted columns: ", cancer_data.shape)
# checking missing data
print(cancer_data.isnull().sum())
# value count of 2 classes
print("Value count of 2 classes are: \n", cancer_data["diagnosis"].value_counts())
# full information of dataset
print(cancer_data.info())


#libraries for plotting
import time
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns; sns.set(style="ticks", color_codes=True)
import hypertools as hyp


# data duplication check
cancer_data.duplicated

print('Number of rows before discarding duplicates = %d' % (cancer_data.shape[0]))
data2 = cancer_data.drop_duplicates()
print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))

#Separate the predictors and target.
x = cancer_data.loc[: , 'radius_mean':'fractal_dimension_worst']
y_true = cancer_data.loc[:, 'diagnosis']
print("Shape of x: ", x.shape)
print("Shape of y_true: ", y_true.shape)


#Visualisation


```

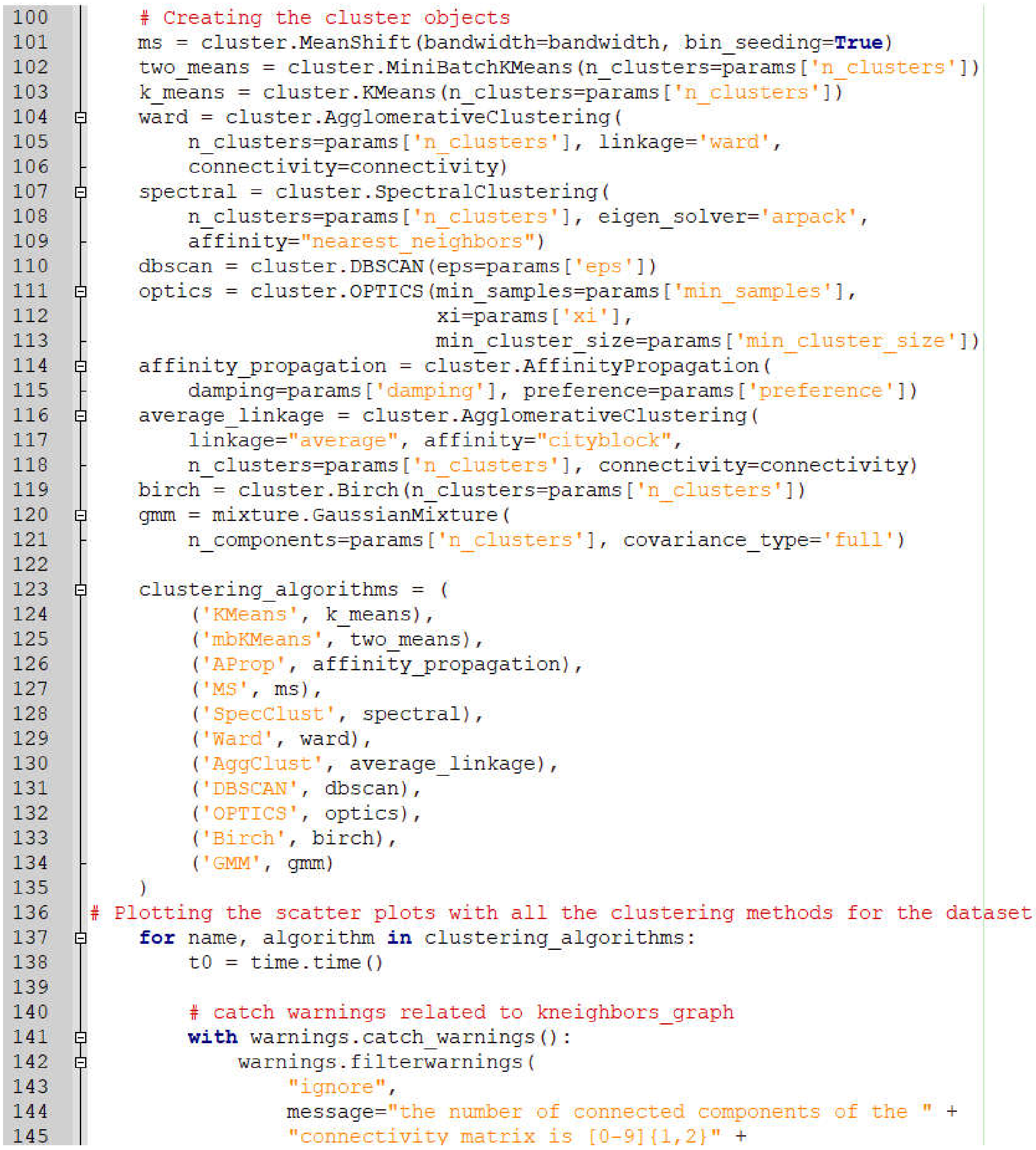


```python


#Visualisation


sns.set(style="whitegrid")
ax = sns.countplot(x = y_true)        # M = 212, B = 357
B, M = y_true.value_counts(sort=True)
plt.show()


#1.1 Violin Plot
# first ten features
data_dia = y_true
data = x
data_n_2 = (data - data.mean()) / (data.std())              # standardization
data = pd.concat([y_true,data_n_2.iloc[:,0:10]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="quart")
plt.xticks(rotation=90)
plt.show()

#1.2 Violin Plot
# second ten features
data_dia = y_true
data = x
data_n_2 = (data - data.mean()) / (data.std())              # standardization
data = pd.concat([y_true,data_n_2.iloc[:,10:20]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="quart")
plt.xticks(rotation=90)
plt.show()

#1.3 Violin Plot
# last ten features
data_dia = y_true
data = x
data_n_2 = (data - data.mean()) / (data.std())              # standardization
data = pd.concat([y_true,data_n_2.iloc[:,20:31]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
```

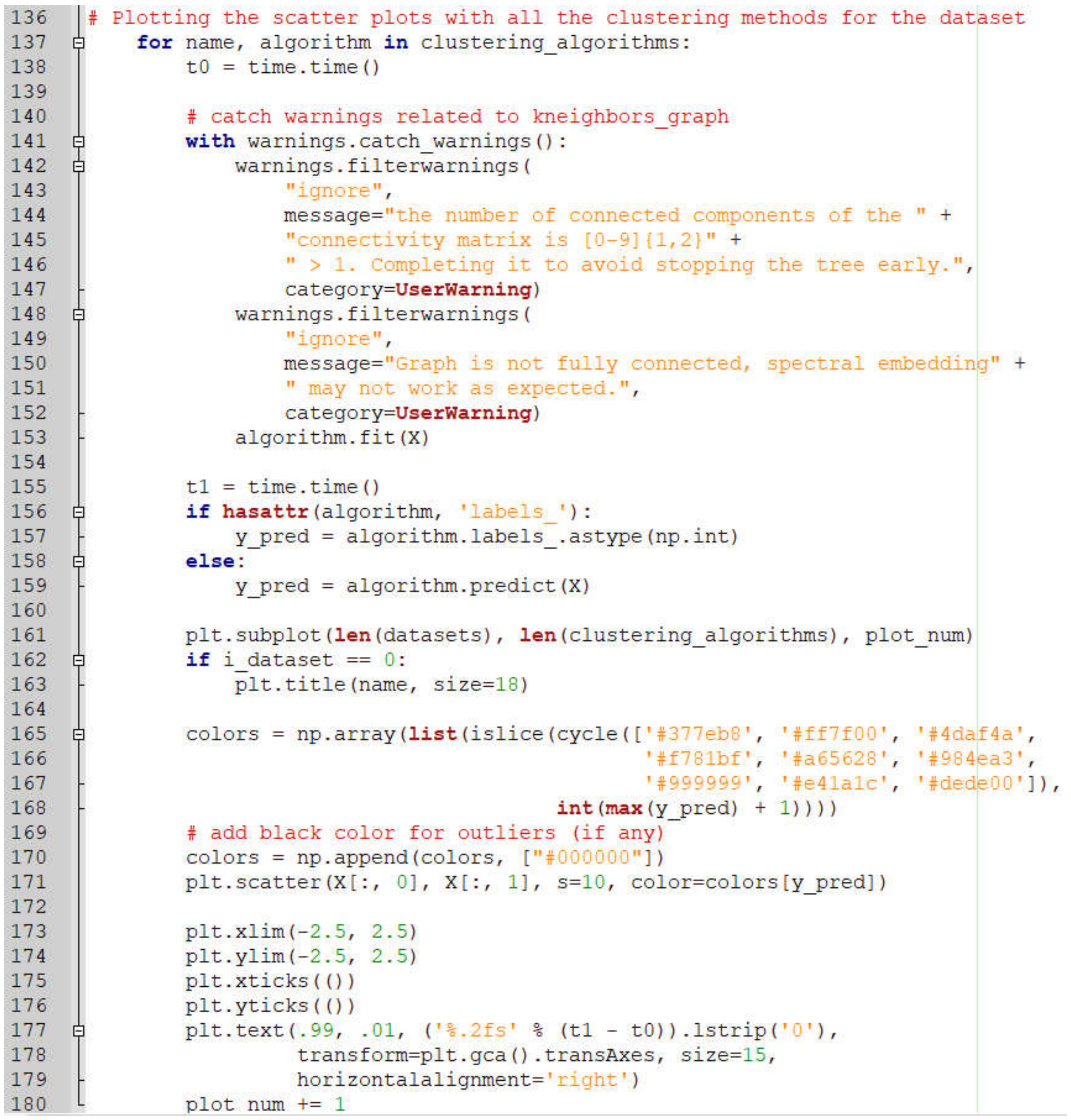
- **PART 2:**



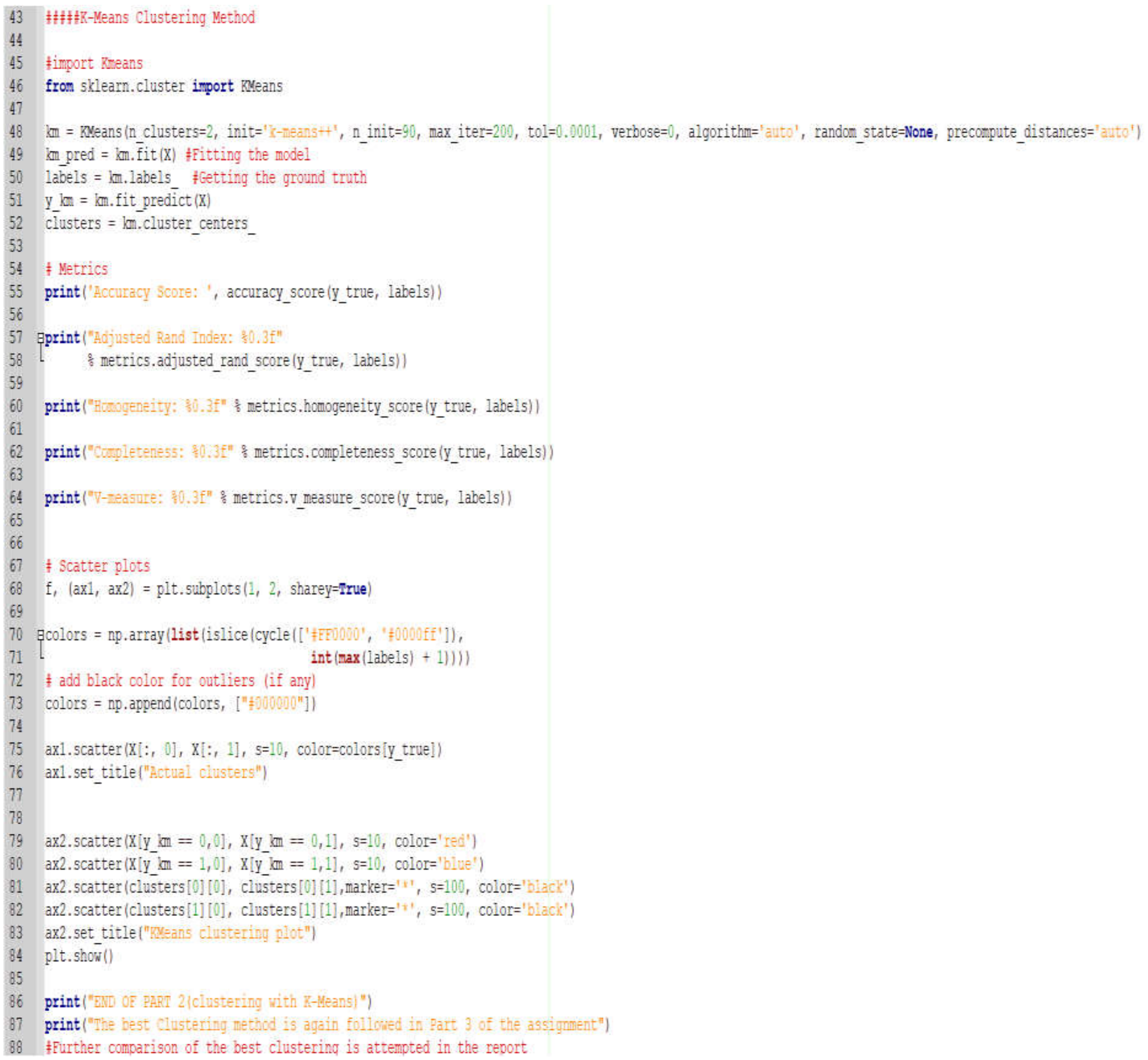
```python
#####CLUSTERING COMPARISON


# Setting up the cluster parameters

plt.figure(figsize=(9 * 2 + 3, 12.5))
plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
                    hspace=.01)

plot_num = 1

default_base = {'quantile': .3,
                'eps': .3,
                'damping': .9,
                'preference': -200,
                'n_neighbors': 10,
                'n_clusters': 2,
                'min_samples': 20,
                'xi': 0.05,
                'min_cluster_size': 0.1}

datasets = [
    (x, {'damping': .77, 'preference': -240,
         'quantile': .2, 'n_clusters': 2,
         'min_samples': 20, 'xi': 0.25})]

for i_dataset, (dataset, algo_params) in enumerate(datasets):
    # update parameters with dataset-specific values
    params = default_base.copy()
    params.update(algo_params)

    X = x
    y = y_true

    # normalize dataset for easier parameter selection
    X = StandardScaler().fit_transform(X)

    # estimate bandwidth for mean shift
    bandwidth = cluster.estimate_bandwidth(X, quantile=params['quantile'])

    # connectivity matrix for structured Ward
    connectivity = kneighbors_graph(
        X, n_neighbors=params['n_neighbors'], include_self=False)
    # make connectivity symmetric
    connectivity = 0.5 * (connectivity + connectivity.T)
```



```python
    # Creating the cluster objects
    ms = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
    two_means = cluster.MiniBatchKMeans(n_clusters=params['n_clusters'])
    k_means = cluster.KMeans(n_clusters=params['n_clusters'])
    ward = cluster.AgglomerativeClustering(
        n_clusters=params['n_clusters'], linkage='ward',
        connectivity=connectivity)
    spectral = cluster.SpectralClustering(
        n_clusters=params['n_clusters'], eigen_solver='arpack',
        affinity="nearest_neighbors")
    dbscan = cluster.DBSCAN(eps=params['eps'])
    optics = cluster.OPTICS(min_samples=params['min_samples'],
                            xi=params['xi'],
                            min_cluster_size=params['min_cluster_size'])
    affinity_propagation = cluster.AffinityPropagation(
        damping=params['damping'], preference=params['preference'])
    average_linkage = cluster.AgglomerativeClustering(
        linkage="average", affinity="cityblock",
        n_clusters=params['n_clusters'], connectivity=connectivity)
    birch = cluster.Birch(n_clusters=params['n_clusters'])
    gmm = mixture.GaussianMixture(
        n_components=params['n_clusters'], covariance_type='full')

    clustering_algorithms = (
        ('KMeans', k_means),
        ('mbKMeans', two_means),
        ('AProp', affinity_propagation),
        ('MS', ms),
        ('SpecClust', spectral),
        ('Ward', ward),
        ('AggClust', average_linkage),
        ('DBSCAN', dbscan),
        ('OPTICS', optics),
        ('Birch', birch),
        ('GMM', gmm)
    )
# Plotting the scatter plots with all the clustering methods for the dataset
    for name, algorithm in clustering_algorithms:
        t0 = time.time()

        # catch warnings related to kneighbors_graph
        with warnings.catch_warnings():
            warnings.filterwarnings(
                "ignore",
                message="the number of connected components of the " +
                "connectivity matrix is [0-9]{1,2}" +
```

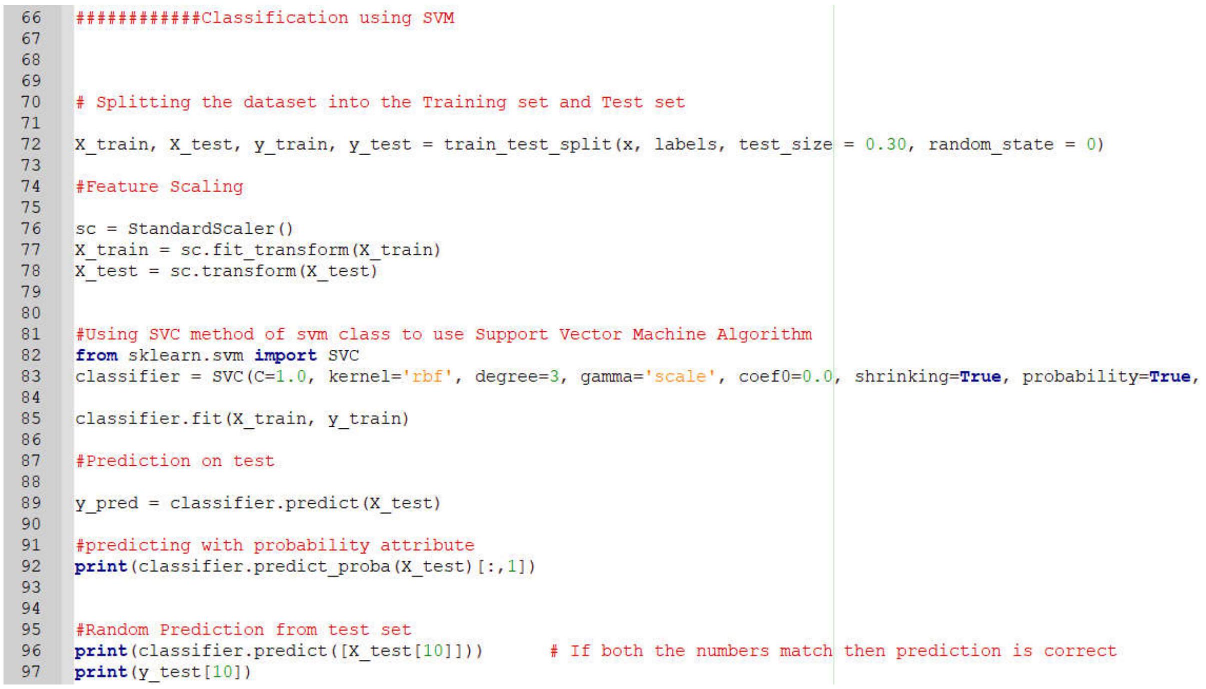


```python
# Plotting the scatter plots with all the clustering methods for the dataset
    for name, algorithm in clustering_algorithms:
        t0 = time.time()

        # catch warnings related to kneighbors_graph
        with warnings.catch_warnings():
            warnings.filterwarnings(
                "ignore",
                message="the number of connected components of the " +
                "connectivity matrix is [0-9]{1,2}" +
                " > 1. Completing it to avoid stopping the tree early.",
                category=UserWarning)
            warnings.filterwarnings(
                "ignore",
                message="Graph is not fully connected, spectral embedding" +
                " may not work as expected.",
                category=UserWarning)
            algorithm.fit(X)

        t1 = time.time()
        if hasattr(algorithm, 'labels_'):
            y_pred = algorithm.labels_.astype(np.int)
        else:
            y_pred = algorithm.predict(X)

        plt.subplot(len(datasets), len(clustering_algorithms), plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                             '#f781bf', '#a65628', '#984ea3',
                                             '#999999', '#e41a1c', '#dede00']),
                                      int(max(y_pred) + 1))))
        # add black color for outliers (if any)
        colors = np.append(colors, ["#000000"])
        plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_pred])

        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
                 transform=plt.gca().transAxes, size=15,
                 horizontalalignment='right')
        plot_num += 1
```

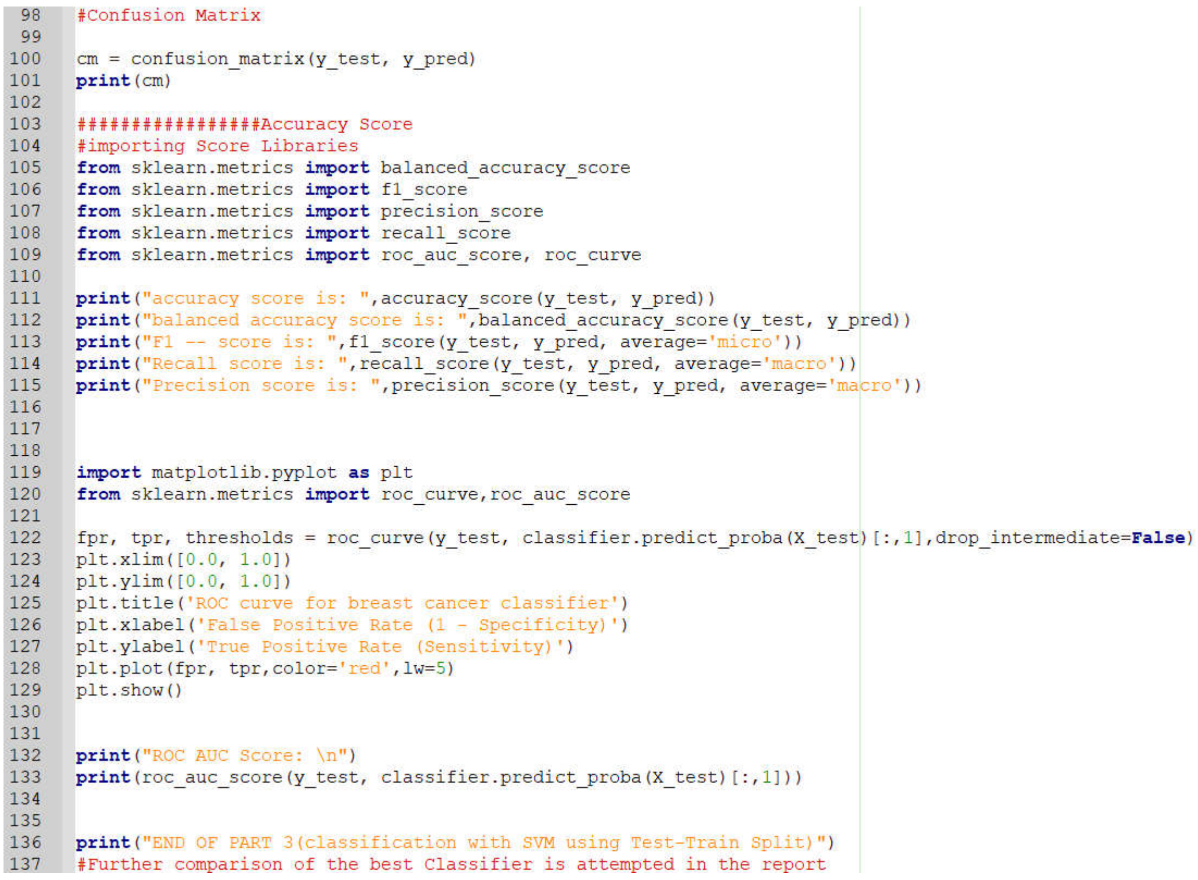


```
#####K-Means Clustering Method

#import Kmeans
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2, init='k-means++', n_init=90, max_iter=200, tol=0.0001, verbose=0, algorithm='auto', random_state=None, precompute_distances='auto')
km_pred = km.fit(X) #Fitting the model
labels = km.labels_  #Getting the ground truth
y_km = km.fit_predict(X)
clusters = km.cluster_centers_

# Metrics
print('Accuracy Score: ', accuracy_score(y_true, labels))

print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(y_true, labels))

print("Homogeneity: %0.3f" % metrics.homogeneity_score(y_true, labels))

print("Completeness: %0.3f" % metrics.completeness_score(y_true, labels))

print("V-measure: %0.3f" % metrics.v_measure_score(y_true, labels))


# Scatter plots
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)

colors = np.array(list(islice(cycle(['#FF0000', '#0000ff']),
                                     int(max(labels) + 1))))
# add black color for outliers (if any)
colors = np.append(colors, ["#000000"])

ax1.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_true])
ax1.set_title("Actual clusters")


ax2.scatter(X[y_km == 0,0], X[y_km == 0,1], s=10, color='red')
ax2.scatter(X[y_km == 1,0], X[y_km == 1,1], s=10, color='blue')
ax2.scatter(clusters[0][0], clusters[0][1],marker='*', s=100, color='black')
ax2.scatter(clusters[1][0], clusters[1][1],marker='*', s=100, color='black')
ax2.set_title("KMeans clustering plot")
plt.show()

print("END OF PART 2(clustering with K-Means)")
print("The best Clustering method is again followed in Part 3 of the assignment")
#Further comparison of the best clustering is attempted in the report
```

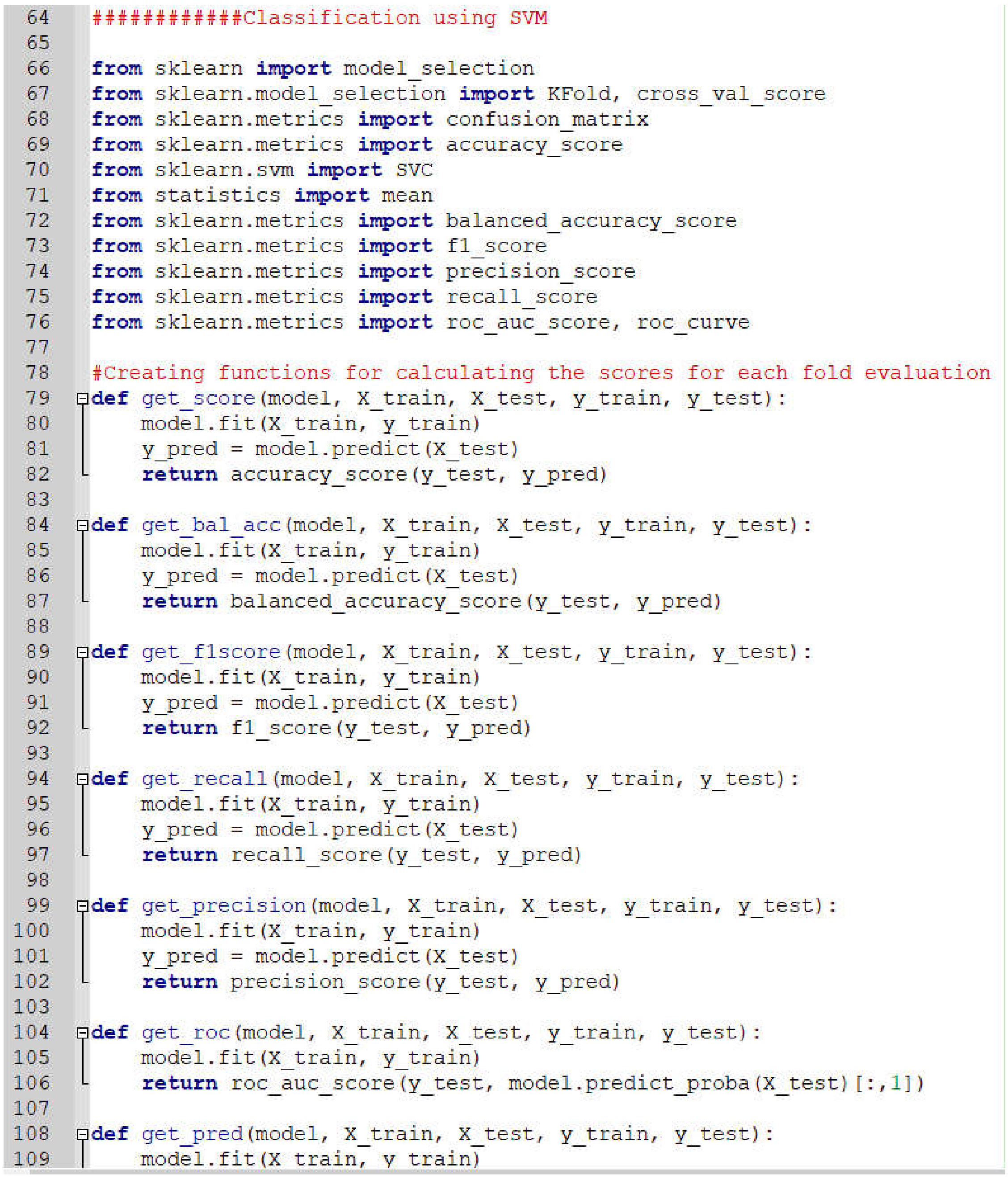
- **PART 3:**



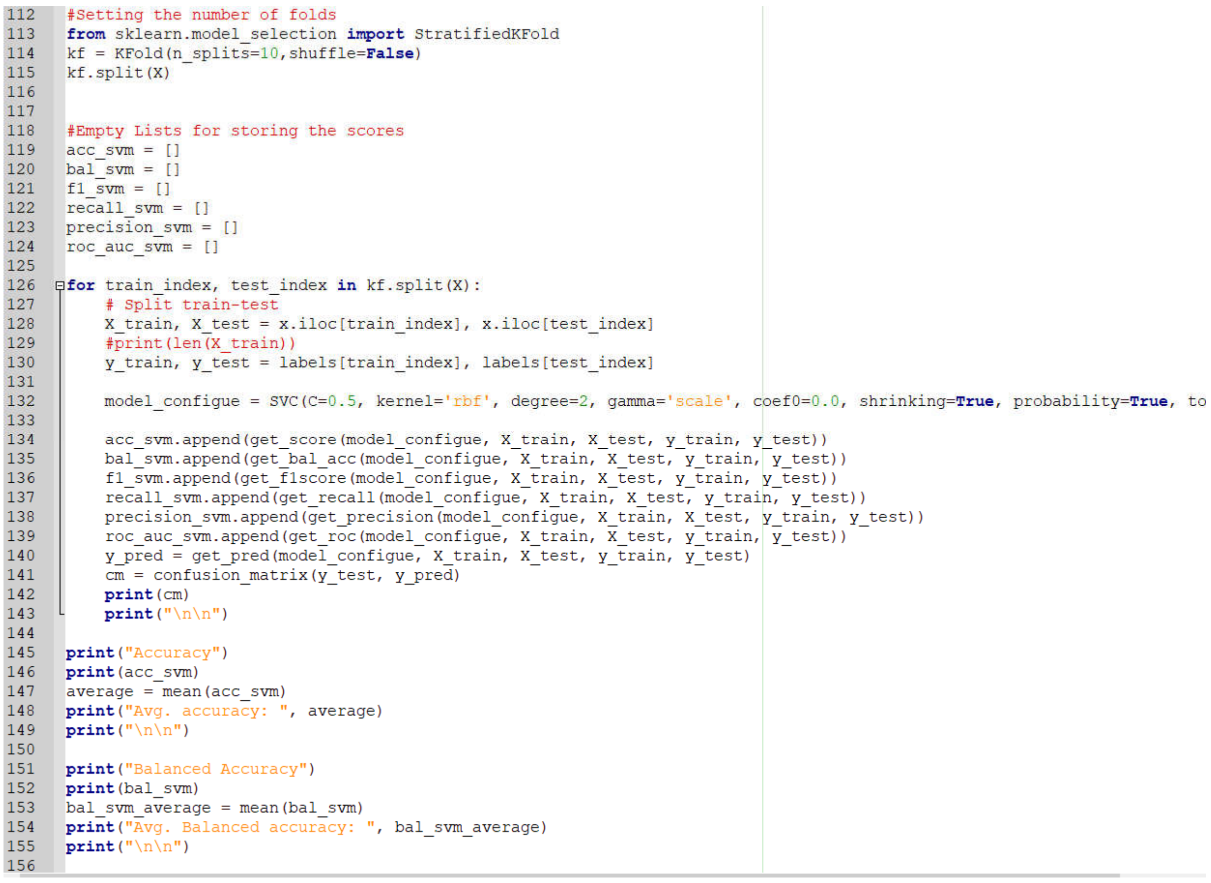
```python
############Classification using SVM



# Splitting the dataset into the Training set and Test set

X_train, X_test, y_train, y_test = train_test_split(x, labels, test_size = 0.30, random_state = 0)

#Feature Scaling

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


#Using SVC method of svm class to use Support Vector Machine Algorithm
from sklearn.svm import SVC
classifier = SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=True,

classifier.fit(X_train, y_train)

#Prediction on test

y_pred = classifier.predict(X_test)

#predicting with probability attribute
print(classifier.predict_proba(X_test)[:,1])


#Random Prediction from test set
print(classifier.predict([X_test[10]]))       # If both the numbers match then prediction is correct
print(y_test[10])
```



```python
#Confusion Matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)

#################Accuracy Score
#importing Score Libraries
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score, roc_curve

print("accuracy score is: ",accuracy_score(y_test, y_pred))
print("balanced accuracy score is: ",balanced_accuracy_score(y_test, y_pred))
print("F1 -- score is: ",f1_score(y_test, y_pred, average='micro'))
print("Recall score is: ",recall_score(y_test, y_pred, average='macro'))
print("Precision score is: ",precision_score(y_test, y_pred, average='macro'))



import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, classifier.predict_proba(X_test)[:,1],drop_intermediate=False)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for breast cancer classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.plot(fpr, tpr,color='red',lw=5)
plt.show()


print("ROC AUC Score: \n")
print(roc_auc_score(y_test, classifier.predict_proba(X_test)[:,1]))


print("END OF PART 3(classification with SVM using Test-Train Split)")
#Further comparison of the best Classifier is attempted in the report
```



```python
############Classification using SVM

from sklearn import model_selection
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from statistics import mean
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score, roc_curve

#Creating functions for calculating the scores for each fold evaluation
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)

def get_bal_acc(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return balanced_accuracy_score(y_test, y_pred)

def get_f1score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return f1_score(y_test, y_pred)

def get_recall(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return recall_score(y_test, y_pred)

def get_precision(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return precision_score(y_test, y_pred)

def get_roc(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return roc_auc_score(y_test, model.predict_proba(X_test)[:,1])

def get_pred(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
```



```python
#Setting the number of folds
from sklearn.model_selection import StratifiedKFold
kf = KFold(n_splits=10,shuffle=False)
kf.split(X)


#Empty Lists for storing the scores
acc_svm = []
bal_svm = []
f1_svm = []
recall_svm = []
precision_svm = []
roc_auc_svm = []

for train_index, test_index in kf.split(X):
    # Split train-test
    X_train, X_test = x.iloc[train_index], x.iloc[test_index]
    #print(len(X_train))
    y_train, y_test = labels[train_index], labels[test_index]

    model_configue = SVC(C=0.5, kernel='rbf', degree=2, gamma='scale', coef0=0.0, shrinking=True, probability=True, to

    acc_svm.append(get_score(model_configue, X_train, X_test, y_train, y_test))
    bal_svm.append(get_bal_acc(model_configue, X_train, X_test, y_train, y_test))
    f1_svm.append(get_f1score(model_configue, X_train, X_test, y_train, y_test))
    recall_svm.append(get_recall(model_configue, X_train, X_test, y_train, y_test))
    precision_svm.append(get_precision(model_configue, X_train, X_test, y_train, y_test))
    roc_auc_svm.append(get_roc(model_configue, X_train, X_test, y_train, y_test))
    y_pred = get_pred(model_configue, X_train, X_test, y_train, y_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print("\n\n")

print("Accuracy")
print(acc_svm)
average = mean(acc_svm)
print("Avg. accuracy: ", average)
print("\n\n")

print("Balanced Accuracy")
print(bal_svm)
bal_svm_average = mean(bal_svm)
print("Avg. Balanced accuracy: ", bal_svm_average)
print("\n\n")

print("\n\n")

print("Balanced Accuracy")
print(bal_svm)
bal_svm_average = mean(bal_svm)
print("Avg. Balanced accuracy: ", bal_svm_average)
print("\n\n")

print("F1 Score")
print(f1_svm)
f1_svm_average = mean(f1_svm)
print("Avg. F1 Score: ", f1_svm_average)
print("\n\n")

print("Recall")
print(recall_svm)
recall_svm_average = mean(recall_svm)
print("Avg. Recall: ", recall_svm_average)
print("\n\n")

print("Precision")
print(precision_svm)
precision_svm_average = mean(precision_svm)
print("Avg. Precision: ", precision_svm_average)
print("\n\n")

print("ROC")
print(roc_auc_svm)
roc_svm_average = mean(roc_auc_svm)
print("Avg. ROC: ", roc_svm_average)
print("\n\n")

print("done")

print("END OF PART 3(classification with SVM using K-Folds Split)")
#Further comparison of the best Classifier is attempted in the report
```