ISL Project Team - 3

Group members:

Suryansh

Rajendra Gudimetla

Teja Immaneni

University of Missouri-Kansas City

2024SP-COMP_SCI-5565-0003

Professor Adu Baffour

Intro to Statistical Learning

Technical Report:

A branch of machine learning and statistics called "statistical learning" is concerned with comprehending and creating algorithms for drawing conclusions or predictions from data. It involves analysing patterns, correlations, and structures in data using statistical models and methodologies. Creating models that can precisely predict or estimate an output variable based on input features or variables is the primary objective of statistical learning. It can be categorised into Supervised and Unsupervised Learning.

Supervised Learning:

In supervised learning, a labelled dataset is used to train the algorithm, and each sample is linked to a target variable. The intention is to enable the algorithm to predict on new, unknown data by learning a mapping from input attributes to the target variable.

E.g.: - Random Forest, Decision Tress, Linear Regression, Logistic Regression etc.,

Unsupervised Learning:

Unsupervised learning aims to find patterns, structures, or correlations in data without the need for human intervention. The algorithm is trained on an unlabelled dataset. Common problems in unsupervised learning include clustering, dimensionality reduction, and association rule mining.

E.g.: - K - Means Clustering, Hierarchical Clustering etc.,

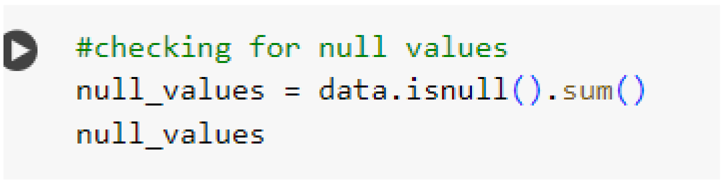
1.Data Description:

To start with a model, we need to collect some data, here we are using some data called dry beans data which carries different types of features and values. It has a vast amount of data that contains the shape of 13611 samples and 17 features.

The dry bean data contains 16 feature roles, 1 target role, 14 continuous types, 2 integer types and 1categorial type.

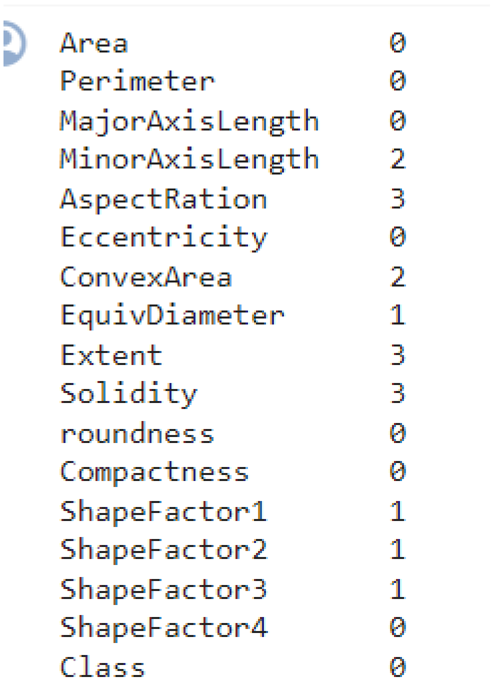
2.Data Preprocessing:

The data needs to be split up into training data and testing data and before splitting up we need to find whether the data has any null values as they may cause miscalculations in finding accuracy.



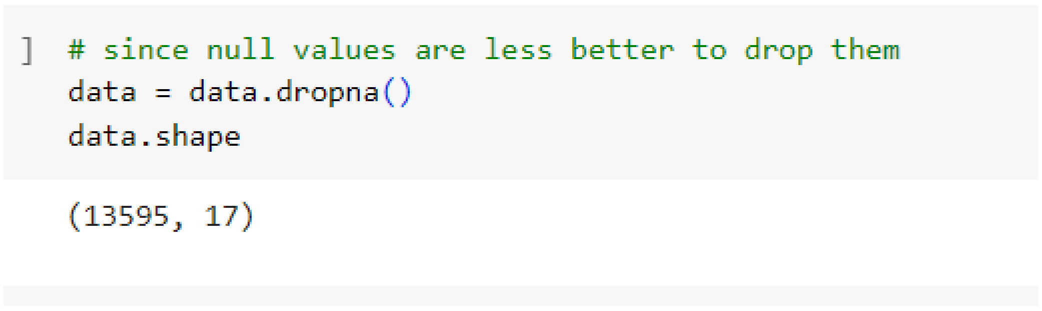
```
#checking for null values
null_values = data.isnull().sum()
null_values
```

The above python command is used to get the values of null values in the data. This data has some null values, and we need to get rid of null values as we need to get more appropriate accuracy and f1 score. By using the above command, we got the null values as described below.



```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    2
AspectRation       3
Eccentricity       0
ConvexArea         2
EquivDiameter      1
Extent             3
Solidity           3
roundness          0
Compactness        0
ShapeFactor1       1
ShapeFactor2       1
ShapeFactor3       1
ShapeFactor4       0
Class              0
```

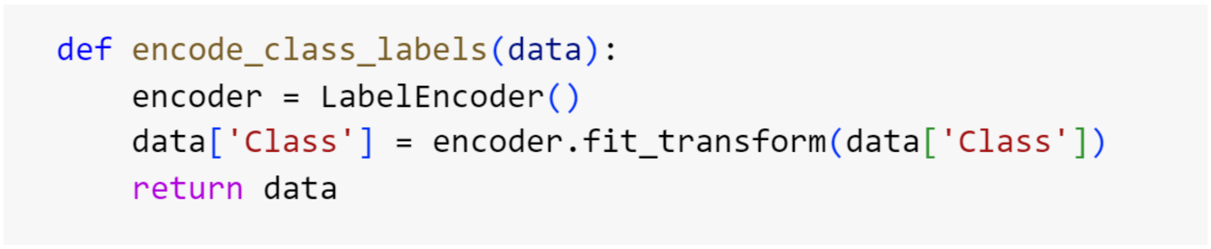
From the above results, we have 3 null values in Aspect Ratio, Extent and Solidity, 2 null values in Minor Axis Length and ConverArea, 1 each in EquivDiameter, ShapeFactor1, ShapeFactor2 and ShapeFactor3.



```
] # since null values are less better to drop them
  data = data.dropna()
  data.shape

  (13595, 17)
```

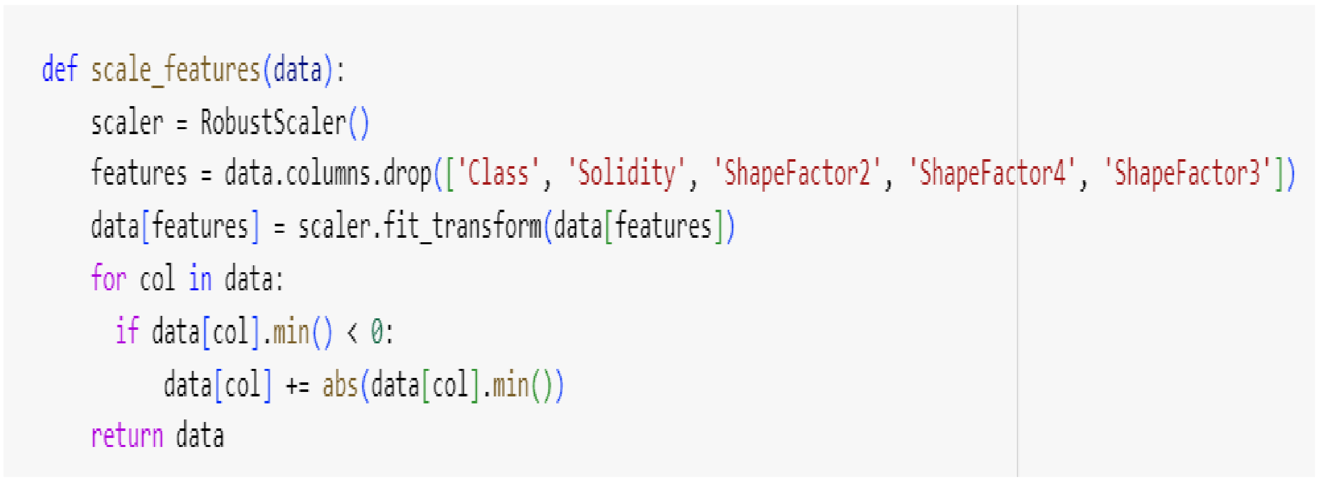
After removing the null values, the data came to the shape that contains 13595 samples and 17 features. Then performed the class labels encoding by following the steps as below:

1. An instance of Label Encoder is created.
2. Then by using the Fitting and transforming internally assigns a unique numerical value to each unique string in column.
3. The original string labels in the ‘Class column is replaced with numerical, then returned the updated labels.

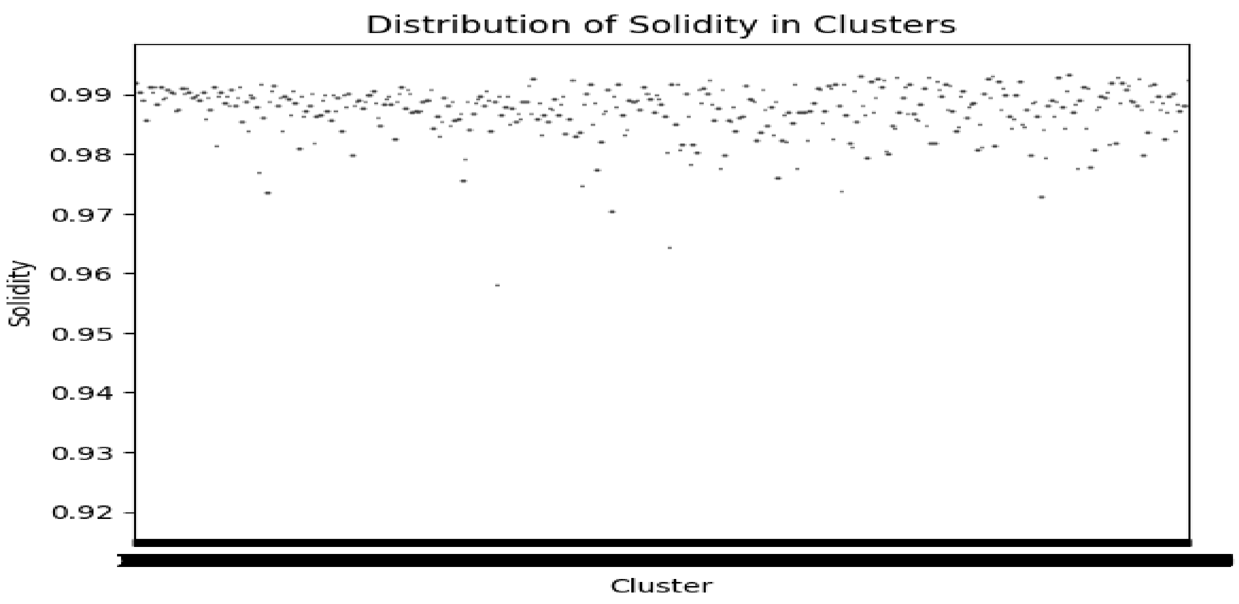


```
def encode_class_labels(data):
    encoder = LabelEncoder()
    data['Class'] = encoder.fit_transform(data['Class'])
    return data
```

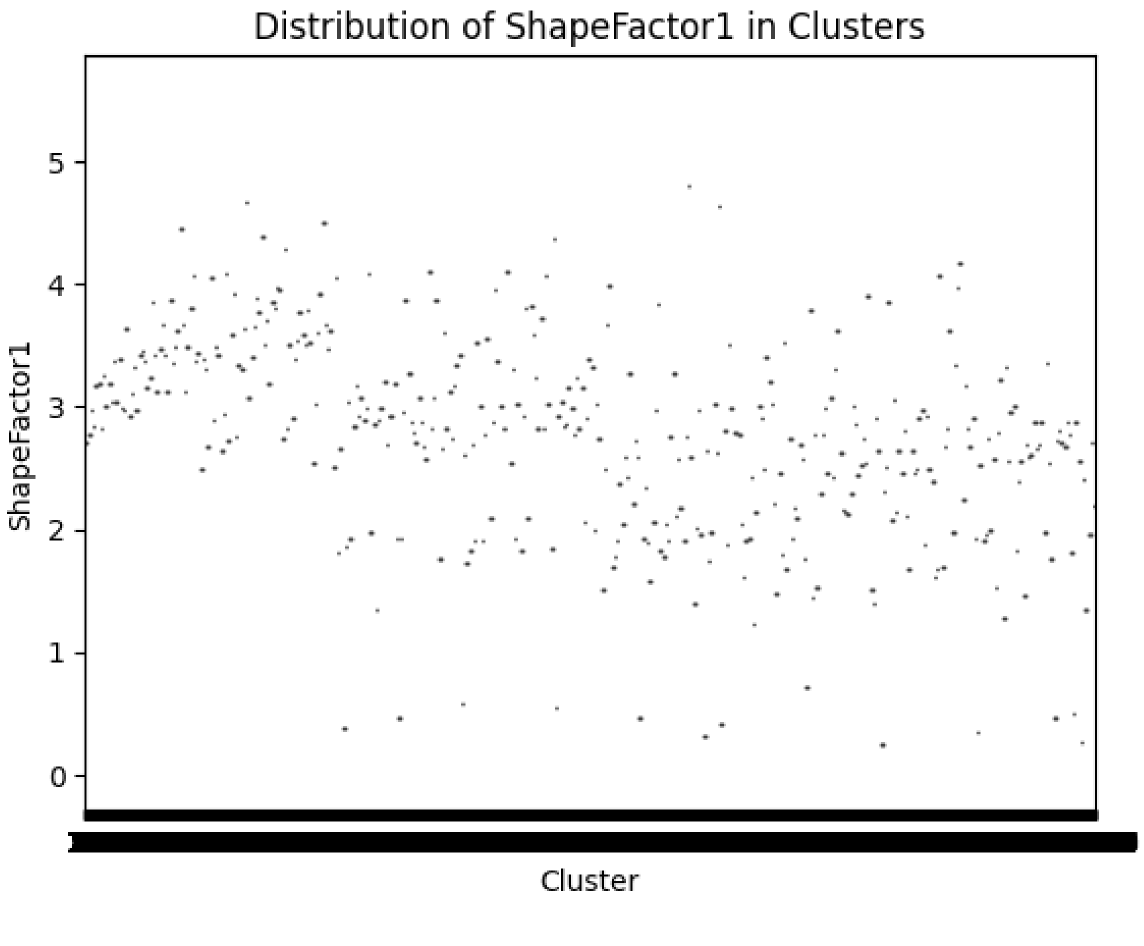
As many machine learning algorithms are sensitive to the scale of the input features. We need to do scaling to our data and to do scaling we have used robust scaler which is most effective our case.



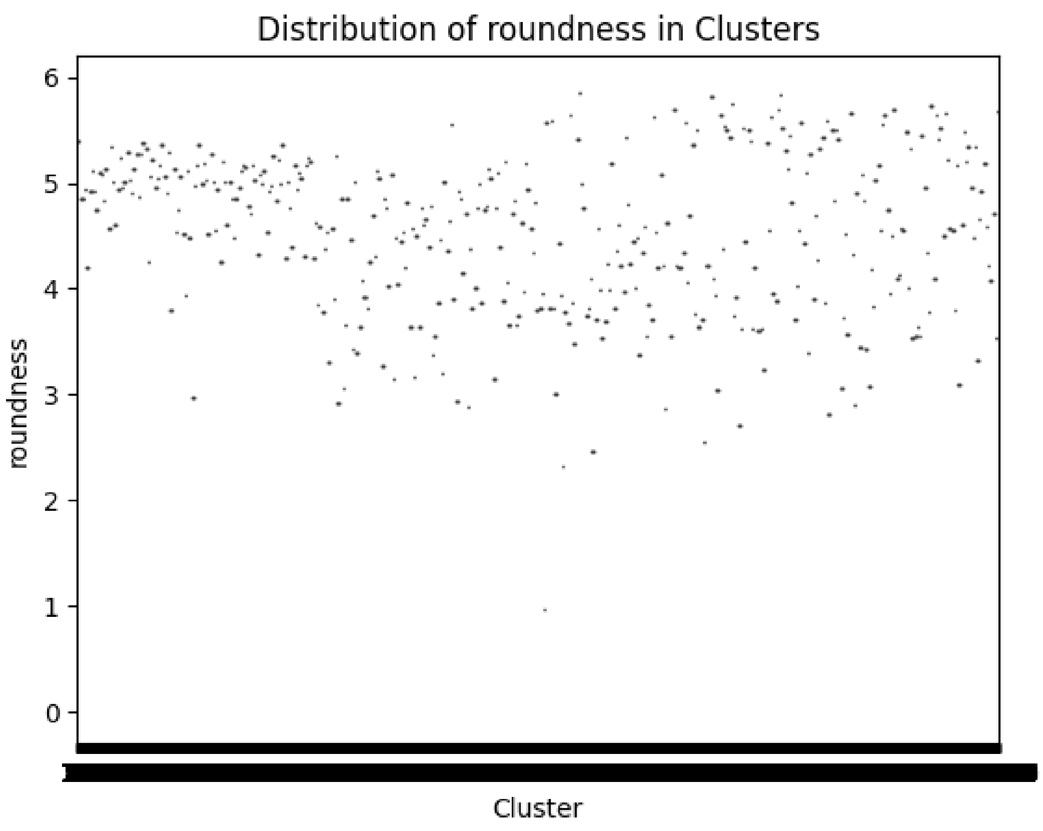
```
def scale_features(data):
    scaler = RobustScaler()
    features = data.columns.drop(['Class', 'Solidity', 'ShapeFactor2', 'ShapeFactor4', 'ShapeFactor3'])
    data[features] = scaler.fit_transform(data[features])
    for col in data:
      if data[col].min() < 0:
          data[col] += abs(data[col].min())
    return data
```



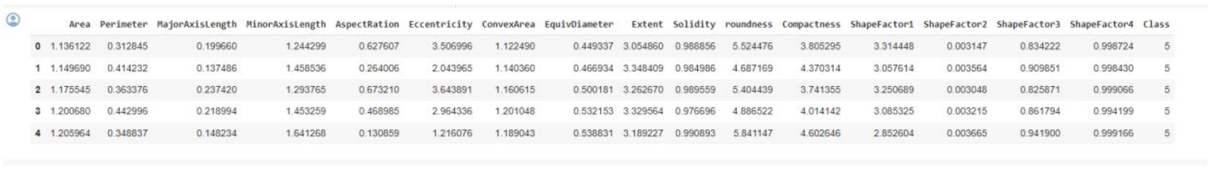
Distribution of Solidity in Clusters
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Solidity
Cluster



Distribution of ShapeFactor1 in Clusters
5
4
3
2
1
0
ShapeFactor1
Cluster

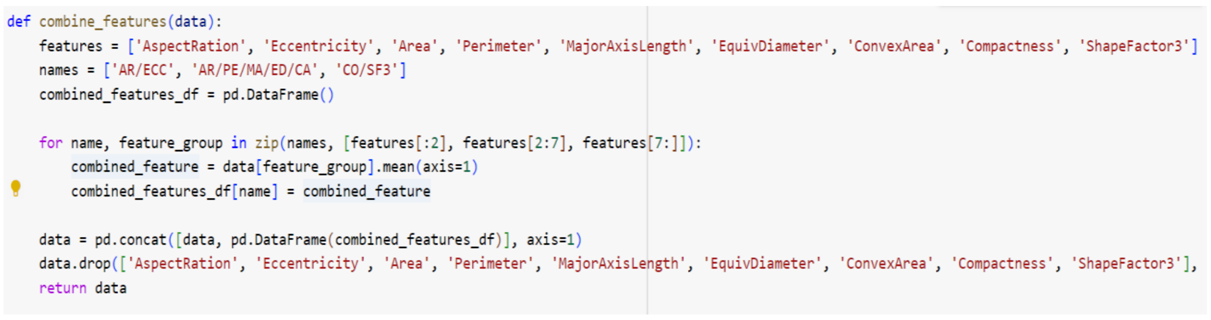


The above commands are used to scale the data using robust scaler. After scaling the data, we got the data as follows:

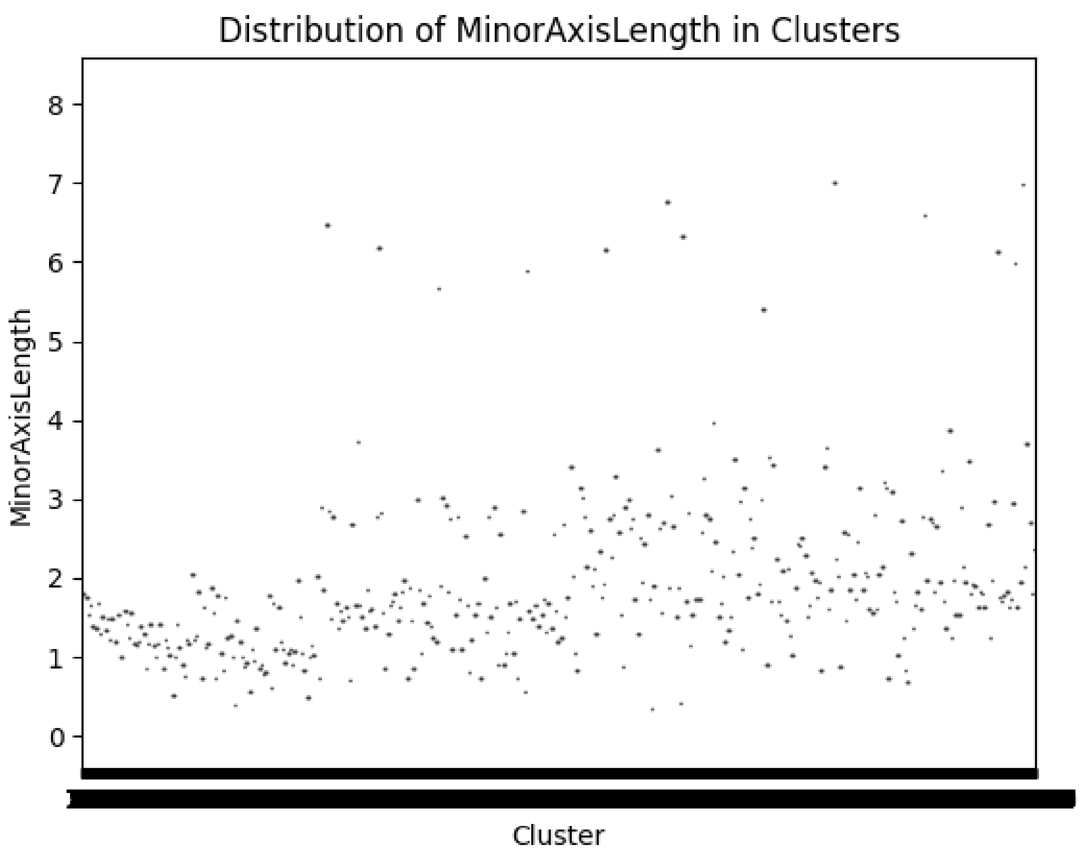


	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3	ShapeFactor4	Class
0	1.136122	0.312845	0.199660	1.244299	0.627607	3.506996	1.122490	0.449337	3.054860	0.988856	5.524476	3.805295	3.314448	0.003147	0.834222	0.998724	5
1	1.149690	0.414232	0.137486	1.458536	0.264006	2.043965	1.140360	0.466934	3.348409	0.984986	4.687169	4.370314	3.057614	0.003564	0.909851	0.998430	5
2	1.175545	0.363376	0.237420	1.293765	0.673210	3.643891	1.160615	0.500181	3.262670	0.989559	5.404439	3.741355	3.250689	0.003048	0.825871	0.999066	5
3	1.200680	0.442996	0.218994	1.453259	0.468985	2.964336	1.201048	0.532153	3.329564	0.976696	4.886522	4.014142	3.085325	0.003215	0.861794	0.994199	5
4	1.205964	0.348837	0.148234	1.641268	0.130859	1.216076	1.189043	0.538831	3.189227	0.990893	5.841147	4.602646	2.852604	0.003665	0.941900	0.999166	5

From the incoming Data Frame data, the function aggregates certain feature groups to create new combined features. It adds new columns with these merged features after calculating the mean of each feature group. Lastly, it removes the original characteristics from the Data Frame and adds these new ones.



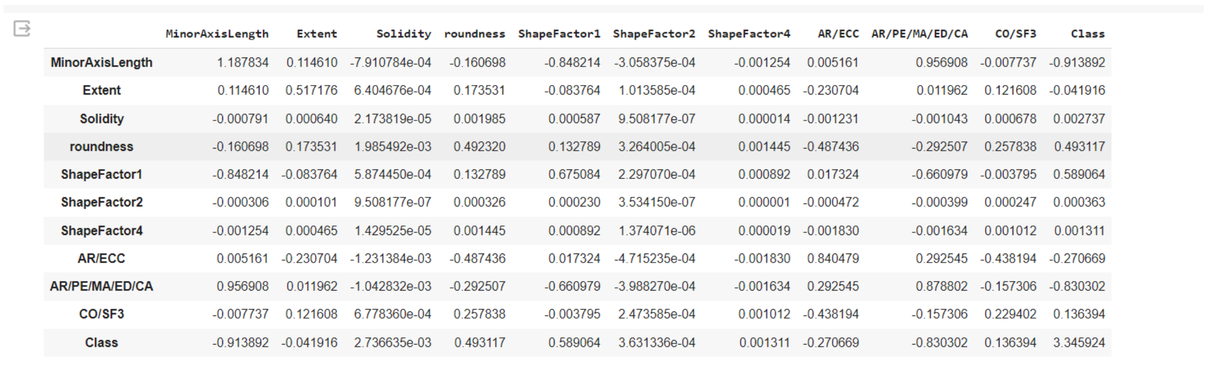
```
def combine_features(data):
    features = ['AspectRation', 'Eccentricity', 'Area', 'Perimeter', 'MajorAxisLength', 'EquivDiameter', 'ConvexArea', 'Compactness', 'ShapeFactor3']
    names = ['AR/ECC', 'AR/PE/MA/ED/CA', 'CO/SF3']
    combined_features_df = pd.DataFrame()

    for name, feature_group in zip(names, [features[:2], features[2:7], features[7:]]):
        combined_feature = data[feature_group].mean(axis=1)
        combined_features_df[name] = combined_feature

    data = pd.concat([data, pd.DataFrame(combined_features_df)], axis=1)
    data.drop(['AspectRation', 'Eccentricity', 'Area', 'Perimeter', 'MajorAxisLength', 'EquivDiameter', 'ConvexArea', 'Compactness', 'ShapeFactor3'],
    return data
```



New combined features:

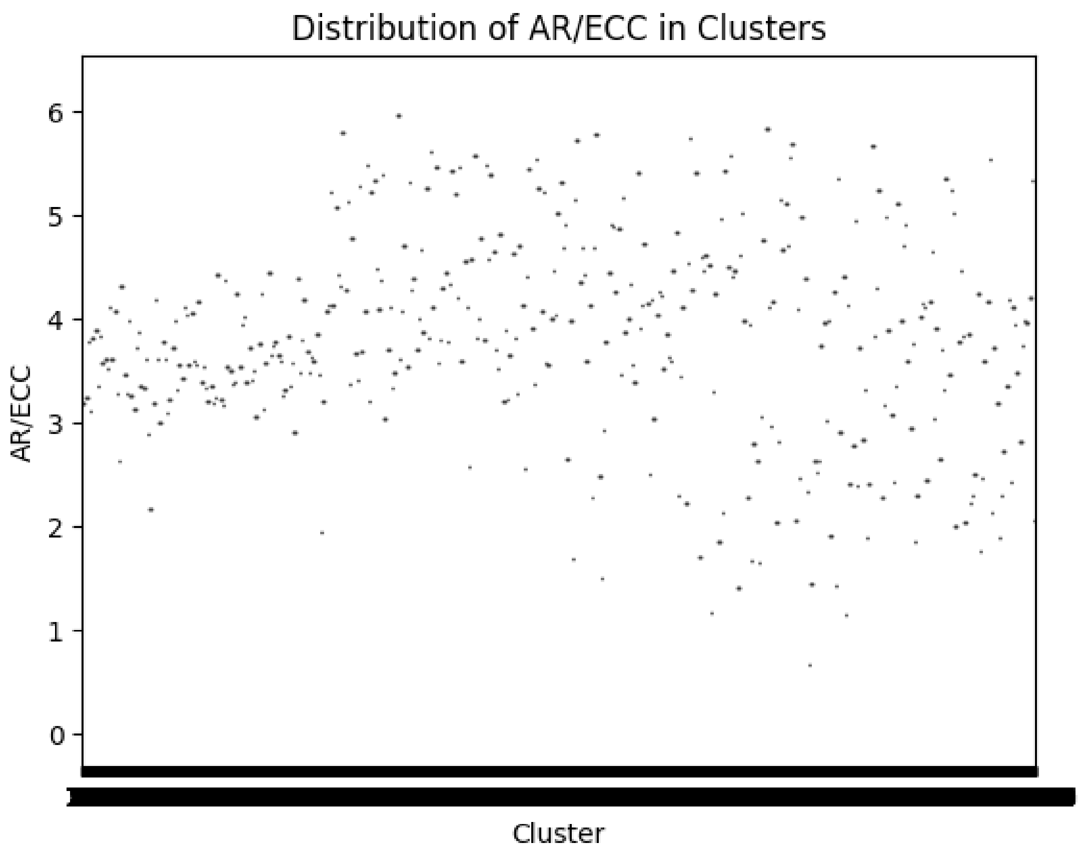
	MinorAxisLength	Extent	Solidity	roundness	ShapeFactor1	ShapeFactor2	ShapeFactor4	Class	AR/ECC	AR/PE/MA/ED/CA	CO/SF3
0	1.244299	3.054860	0.988856	5.524476	3.314448	0.003147	0.998724	5	2.067302	0.644091	2.319759
1	1.458536	3.348409	0.984986	4.687169	3.057614	0.003564	0.998430	5	1.153985	0.661740	2.640082
2	1.293765	3.262670	0.989559	5.404439	3.250689	0.003048	0.999066	5	2.158551	0.687428	2.283613
3	1.453259	3.329564	0.976696	4.886522	3.085325	0.003215	0.994199	5	1.716661	0.719174	2.437968
4	1.641268	3.189227	0.990893	5.841147	2.852604	0.003665	0.999166	5	0.673468	0.686182	2.772273

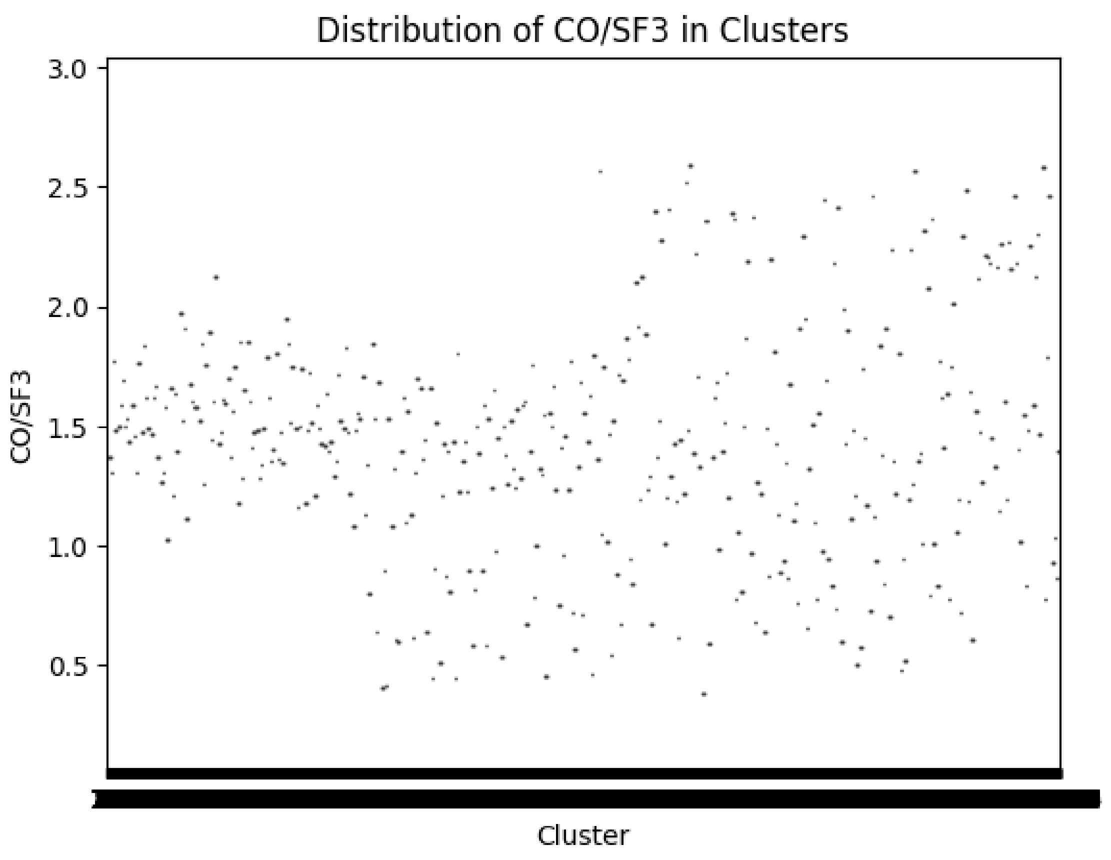
Covariance Matrix:

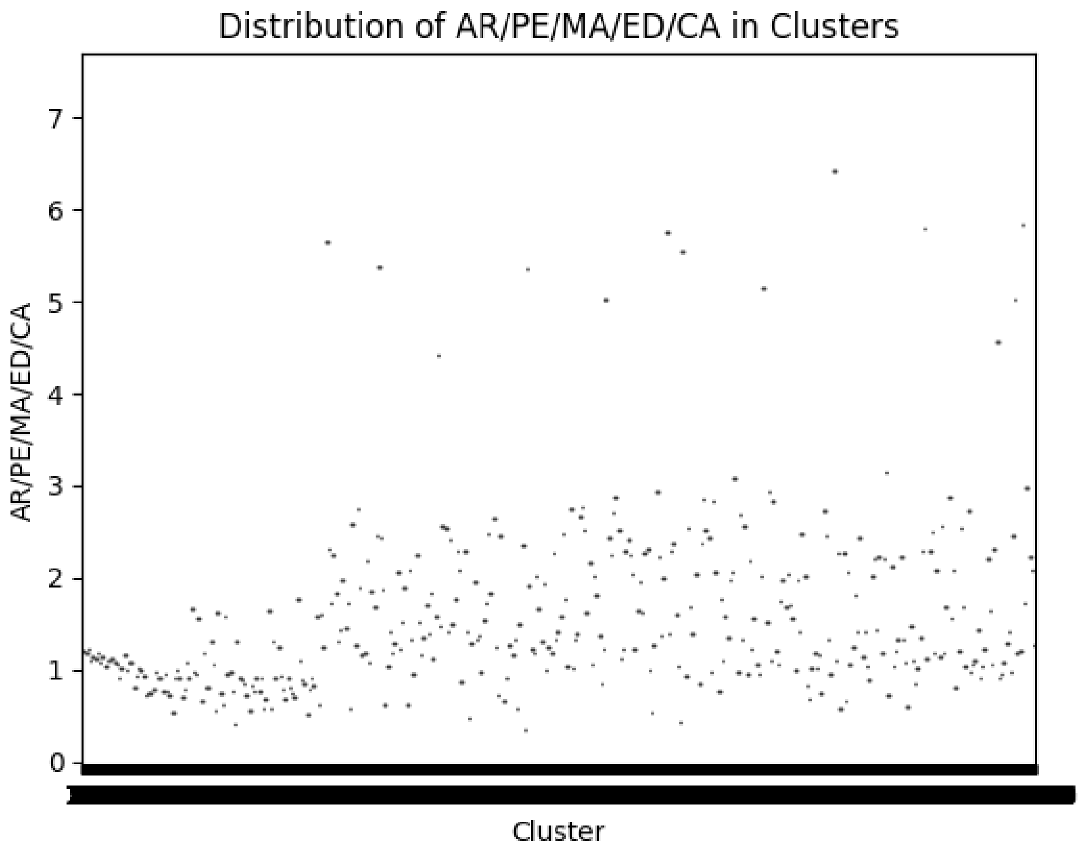


	MinorAxisLength	Extent	Solidity	roundness	ShapeFactor1	ShapeFactor2	ShapeFactor4	AR/ECC	AR/PE/MA/ED/CA	CO/SF3	Class
MinorAxisLength	1.187834	0.114610	-7.910784e-04	-0.160698	-0.848214	-3.058375e-04	-0.001254	0.005161	0.956908	-0.007737	-0.913892
Extent	0.114610	0.517176	6.404676e-04	0.173531	-0.083764	1.013585e-04	0.000465	-0.230704	0.011962	0.121608	-0.041916
Solidity	-0.000791	0.000640	2.173819e-05	0.001985	0.000587	9.508177e-07	0.000014	-0.001231	-0.001043	0.000678	0.002737
roundness	-0.160698	0.173531	1.985492e-03	0.492320	0.132789	3.264005e-04	0.001445	-0.487436	-0.292507	0.257838	0.493117
ShapeFactor1	-0.848214	-0.083764	5.874450e-04	0.132789	0.675084	2.297070e-04	0.000892	0.017324	-0.660979	-0.003795	0.589064
ShapeFactor2	-0.000306	0.000101	9.508177e-07	0.000326	0.000230	3.534150e-07	0.000001	-0.000472	-0.000399	0.000247	0.000363
ShapeFactor4	-0.001254	0.000465	1.429525e-05	0.001445	0.000892	1.374071e-06	0.000019	-0.001830	-0.001634	0.001012	0.001311
AR/ECC	0.005161	-0.230704	-1.231384e-03	-0.487436	0.017324	-4.715235e-04	-0.001830	0.840479	0.292545	-0.438194	-0.270669
AR/PE/MA/ED/CA	0.956908	0.011962	-1.042832e-03	-0.292507	-0.660979	-3.988270e-04	-0.001634	0.292545	0.878802	-0.157306	-0.830302
CO/SF3	-0.007737	0.121608	6.778360e-04	0.257838	-0.003795	2.473585e-04	0.001012	-0.438194	-0.157306	0.229402	0.136394
Class	-0.913892	-0.041916	2.736635e-03	0.493117	0.589064	3.631336e-04	0.001311	-0.270669	-0.830302	0.136394	3.345924

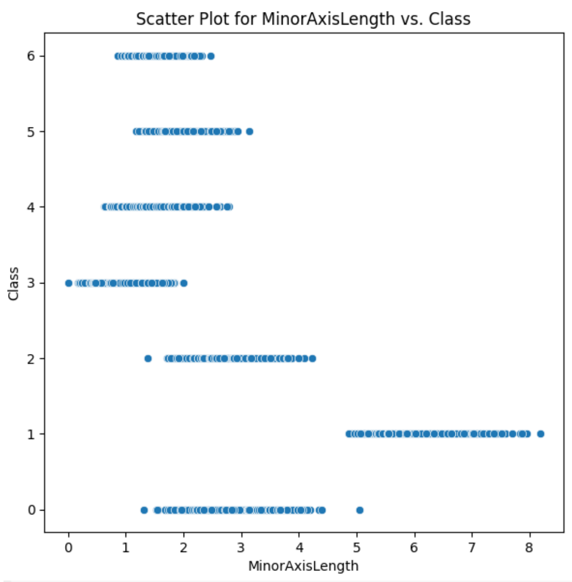
Visualization of combined features

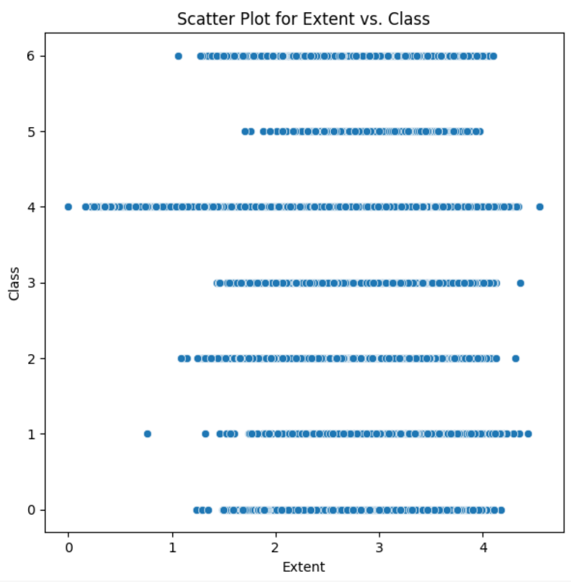


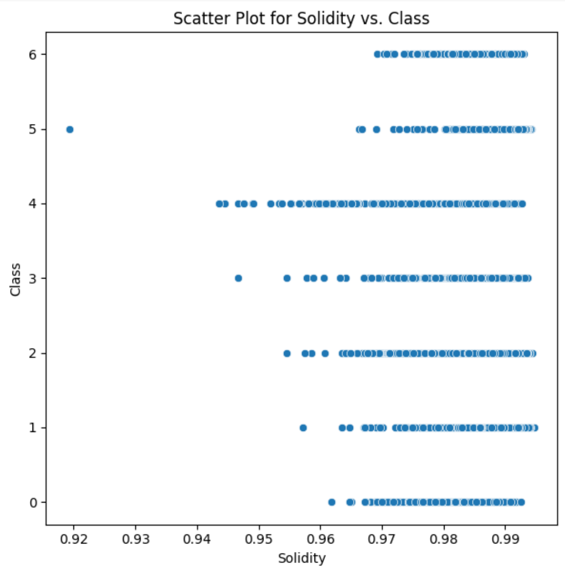


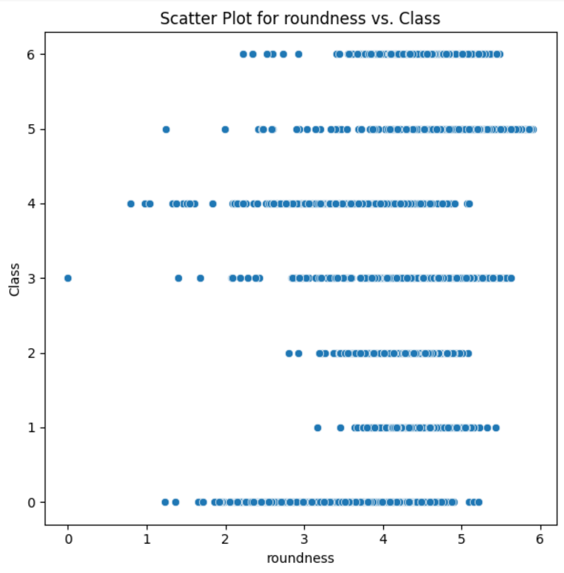


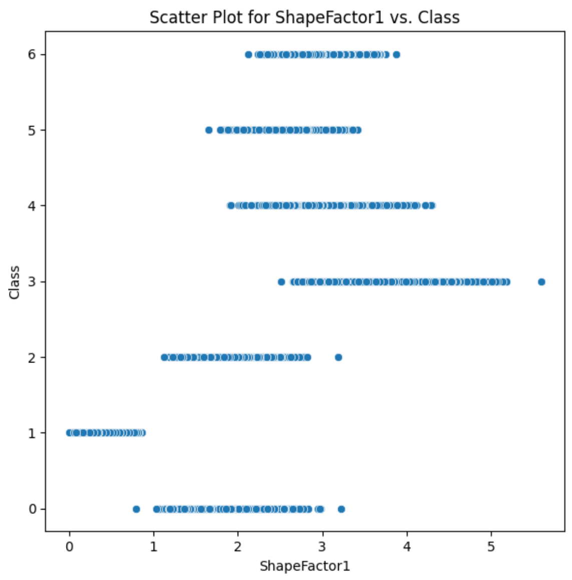
3.Exploratory data analysis:

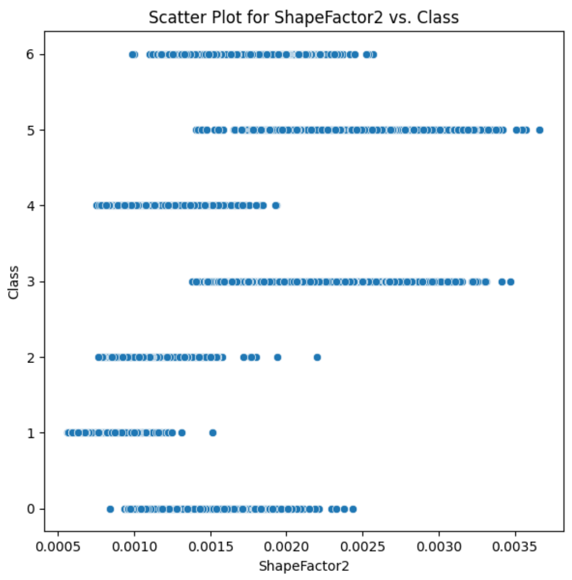

Scatter Plot for MinorAxisLength vs. Class
Class
MinorAxisLength

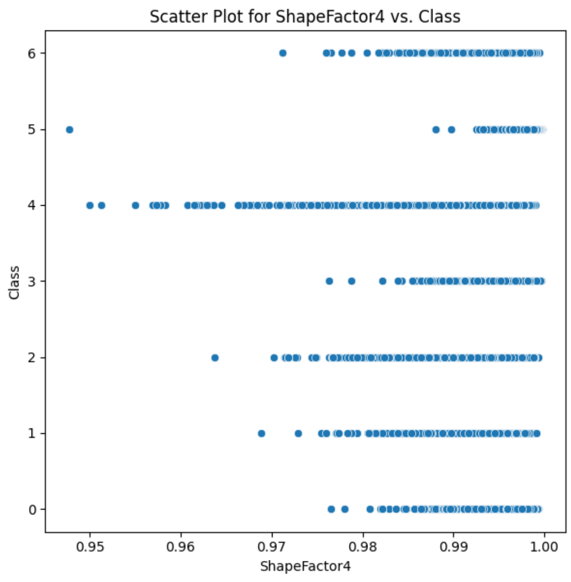

Scatter Plot for Extent vs. Class
Class
Extent

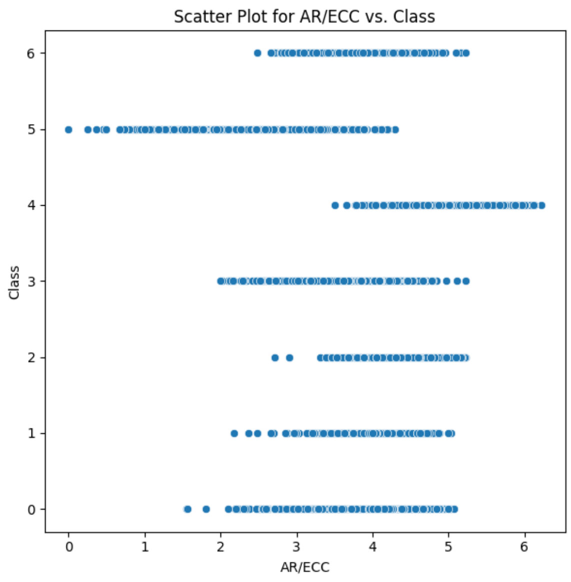

Scatter Plot for Solidity vs. Class
Class
Solidity

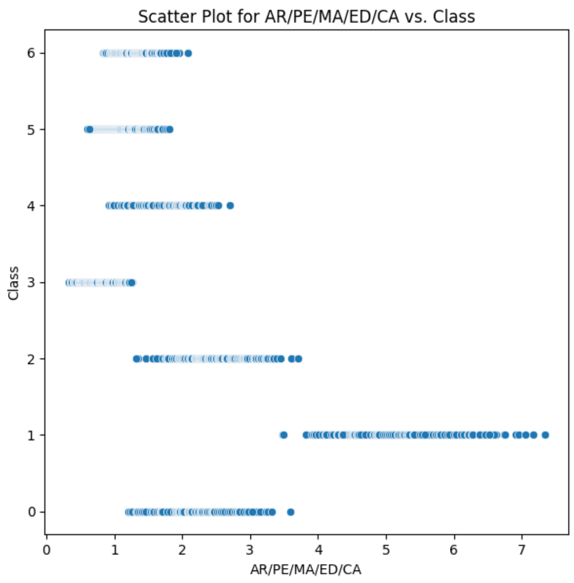

Scatter Plot for roundness vs. Class
Class
roundness

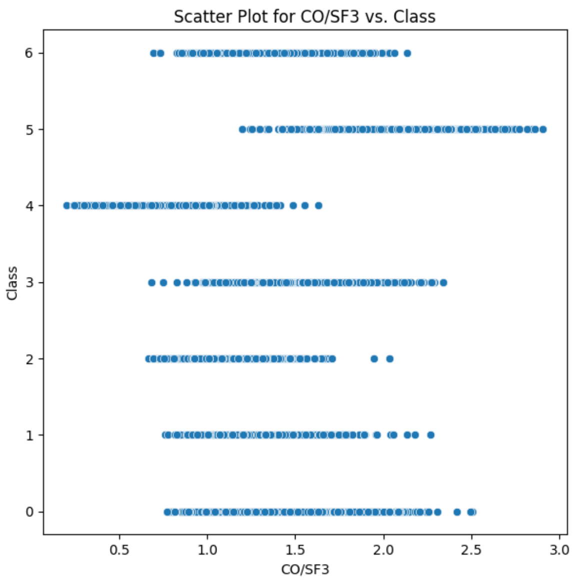

Scatter Plot for ShapeFactor1 vs. Class
Class
ShapeFactor1

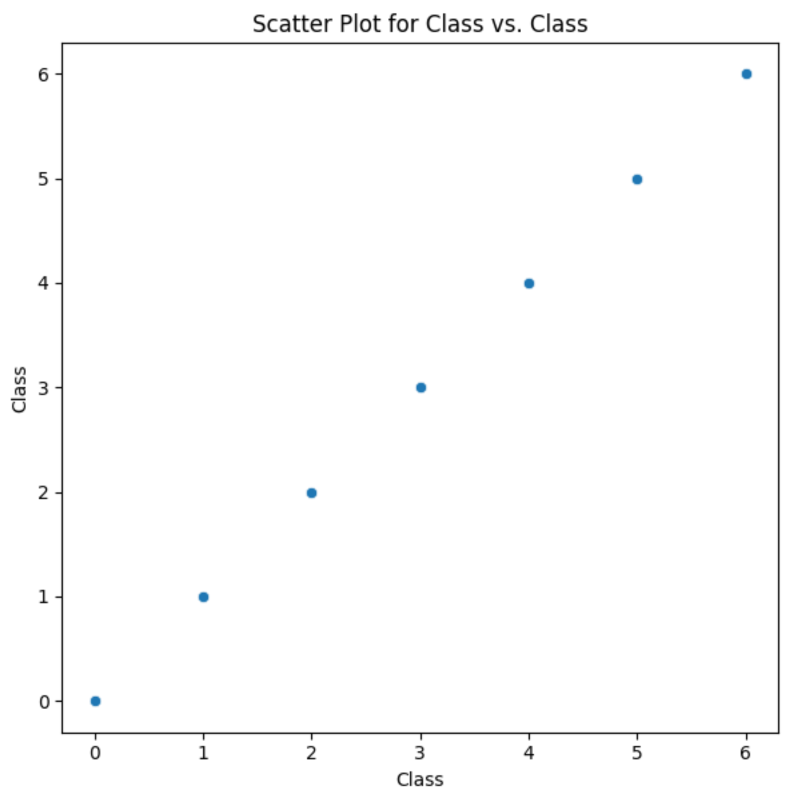

Scatter Plot for ShapeFactor2 vs. Class
Class
ShapeFactor2


Scatter Plot for ShapeFactor4 vs. Class
Class
ShapeFactor4


Scatter Plot for AR/ECC vs. Class
Class
AR/ECC


Scatter Plot for AR/PE/MA/ED/CA vs. Class
Class
AR/PE/MA/ED/CA


Scatter Plot for CO/SF3 vs. Class
Class
CO/SF3



4.Model Development:

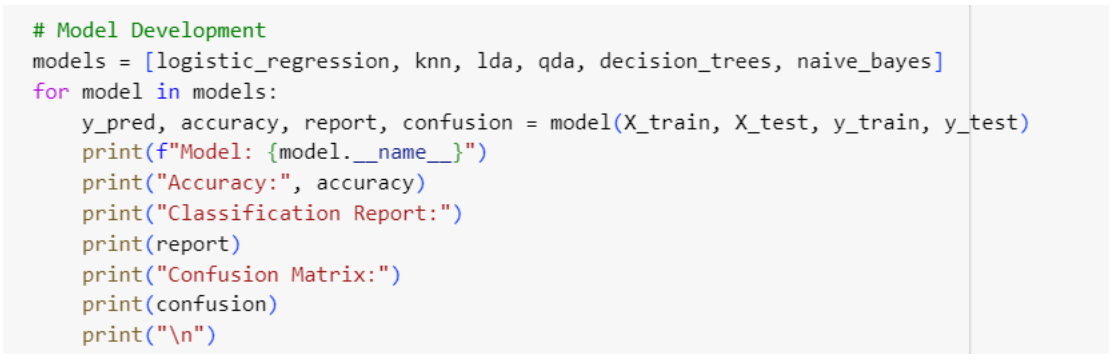
To train and test the model, we need to process the data which we did in earlier steps and now we need to select models which gives us more accuracy and f1 score. In Supervised Learning, we selected 2 models namely, KNN and Logistic Regression and coming to Unsupervised Learning, we took K – Means.

4.1 Supervised Learning:

1. Logistic Regression:

Logistic regression is a type of regression analysis used for predicting the probability of a binary outcome based on one or more predictor variables. It's commonly used for classification tasks where the outcome variable is categorical and binary.

```
def logistic_regression(X_train, X_test, y_train, y_test):
    logistic_reg = LogisticRegression(max_iter=100, multi_class='multinomial', solver='lbfgs')
    logistic_reg.fit(X_train, y_train)
    y_pred = logistic_reg.predict(X_test)
    return y_pred, accuracy_score(y_test, y_pred), classification_report(y_test, y_pred), confusion_matrix(y_test, y_pred)
```

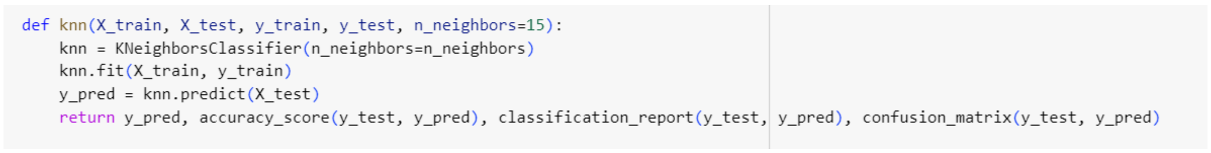


```python
# Model Development
models = [logistic_regression, knn, lda, qda, decision_trees, naive_bayes]
for model in models:
    y_pred, accuracy, report, confusion = model(X_train, X_test, y_train, y_test)
    print(f"Model: {model.__name__}")
    print("Accuracy:", accuracy)
    print("Classification Report:")
    print(report)
    print("Confusion Matrix:")
    print(confusion)
    print("\n")
```

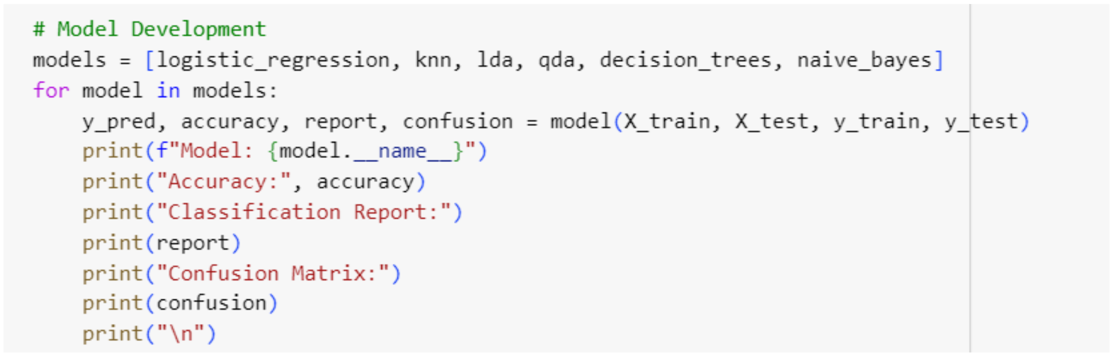
To obtain the expected results in Statistical Learning, first we need to split up the data and we need to train and test the data to transform a model as per our requirement. Here, we have given the max iteration value and multi class value to get the model into logistic regression.

2.KNN:

K-Nearest Neighbors (KNN) is a simple, yet powerful algorithm used for both classification and regression tasks. It is a non-parametric and instance-based learning algorithm, meaning it does not make assumptions about the underlying distribution of the data and instead relies on the data itself to make predictions.



```python
def knn(X_train, X_test, y_train, y_test, n_neighbors=15):
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return y_pred, accuracy_score(y_test, y_pred), classification_report(y_test, y_pred), confusion_matrix(y_test, y_pred)
```



```python
# Model Development
models = [logistic_regression, knn, lda, qda, decision_trees, naive_bayes]
for model in models:
    y_pred, accuracy, report, confusion = model(X_train, X_test, y_train, y_test)
    print(f"Model: {model.__name__}")
    print("Accuracy:", accuracy)
    print("Classification Report:")
    print(report)
    print("Confusion Matrix:")
    print(confusion)
    print("\n")
```

KNN makes predictions by identifying the K nearest data points (neighbors) to a given query point in the feature space. For classification, the predicted class label is typically the majority class among the K nearest neighbors. For regression, the predicted value is typically the mean or median of the target values of the K nearest neighbors.

4.2 Unsupervised Learning:

1. K – Means:

K-Means is a popular unsupervised learning algorithm used for clustering data into groups or clusters based on similarity. It aims to partition the data into K distinct clusters, where each data point belongs to the cluster with the nearest mean (centroid).

```python
def kmeans_clustering(X_train, X_test, n_clusters=7):
    kmeans = KMeans(n_clusters=n_clusters, random_state=33, n_init="auto").fit(X_train)
    kmeans_labels = kmeans.predict(X_test)
    silhouette = silhouette_score(X_test, kmeans_labels)
    return kmeans_labels, silhouette
```

```python
# Clustering
kmeans_labels, silhouette_kmeans = kmeans_clustering(X_train, X_test)
print("K-Means Clustering:")
print("Silhouette Score:", silhouette_kmeans)
print("\n")
```

K-Means begins by randomly initializing K centroids in the feature space. It then iteratively assigns each data point to the nearest centroid and updates the centroids based on the mean of the data points assigned to each cluster. The process continues until convergence, where the centroids stabilize, and no further changes occur.

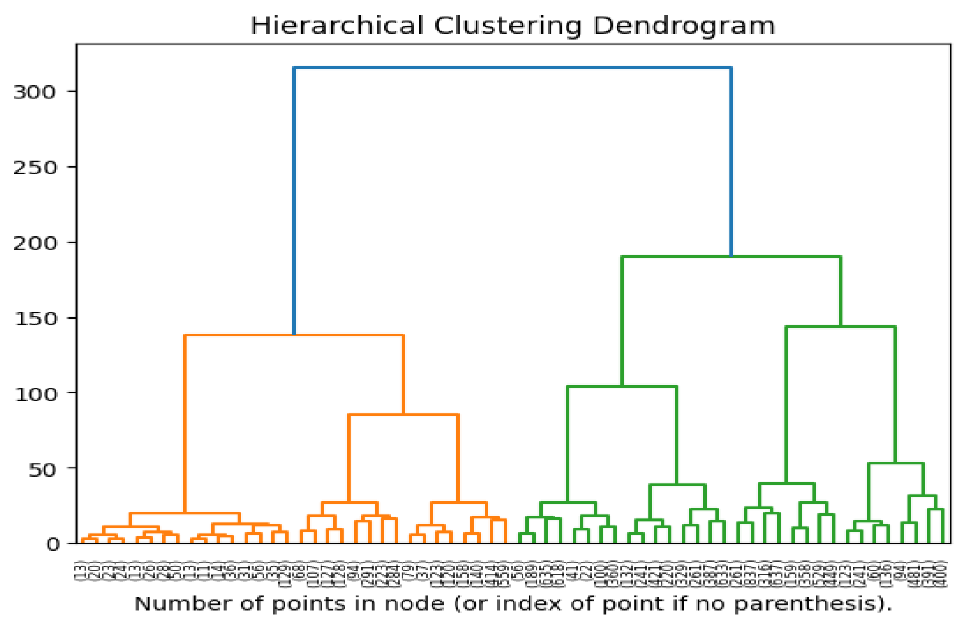
2. Hierarchical Clustering:

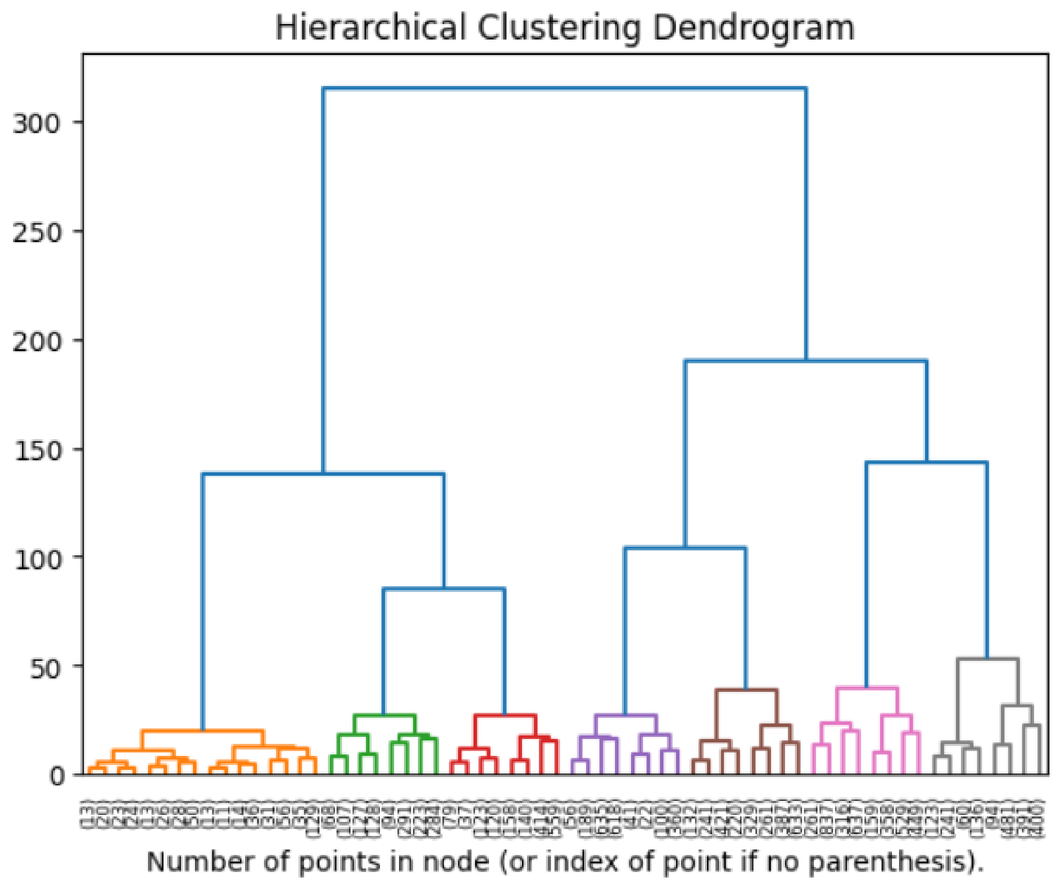
Hierarchical Clustering is an unsupervised learning algorithm that starts by treating each data point as a separate cluster and merges the closest pairs of clusters until we get the single cluster. The hierarchy can be represented by dendrogram. Hierarchical Clustering can be agglomerative which is a bottom-up approach.

```python
def hierarchical_clustering(X, n_clusters=7):
    model = AgglomerativeClustering(n_clusters=n_clusters)
    model = model.fit(X)
    test_labels = model.fit_predict(X)
    silhouette_hc = silhouette_score(X, test_labels)
    return test_labels, silhouette_hc
```

```python
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


X = data
# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)
```

```
plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=5 , color_threshold=55)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```





Here, we have used the agglomerative clustering, where each observation started in its own cluster and pairs of clusters are merged in the hierarchy. But the choice of merging of clusters is based on the distance between clusters, linkage etc.

```
hc_labels, silhouette_hc = hierarchical_clustering(data[:-1])
print("Hierarchical Clustering:")
print("Silhouette Score:", silhouette_hc)
print("\n")
# evaluate_feature_relevance(X_test , hc_labels)
print("\n")
```

5.Performance Evaluation:

The following results show the evaluations done with different types of algorithms and based on those results we have decided on the best model to continue. All the results are based on the Sklearn method.

1.Supervised Learning:

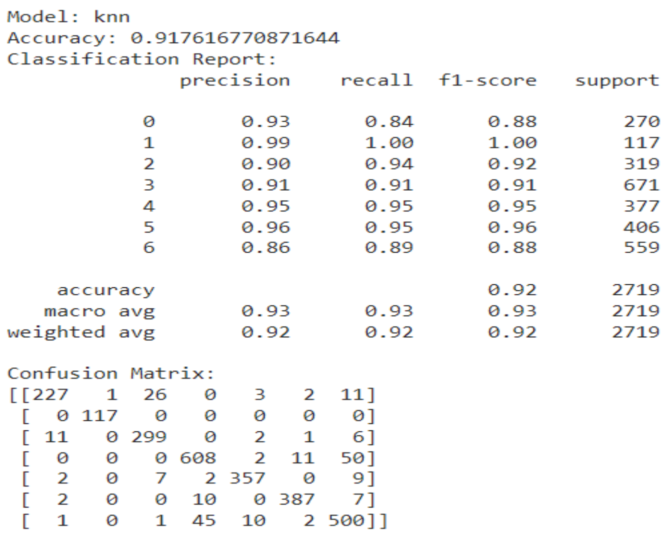
Logistic regression:

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model: logistic_regression
Accuracy: 0.9124678190511217
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.86      0.89       270
           1       0.99      1.00      1.00       117
           2       0.92      0.92      0.92       319
           3       0.91      0.90      0.90       671
           4       0.95      0.95      0.95       377
           5       0.95      0.93      0.94       406
           6       0.85      0.89      0.87       559

    accuracy                           0.91      2719
   macro avg       0.93      0.92      0.92      2719
weighted avg       0.91      0.91      0.91      2719

Confusion Matrix:
[[233   1  19   0   3   2  12]
 [  0 117   0   0   0   0   0]
 [ 17   0 295   0   2   1   4]
 [  0   0   0 603   1  11  56]
 [  0   0   7   4 358   0   8]
 [  4   0   0  13   0 379  10]
 [  0   0   0  43  14   6 496]]
```

From the above results we have

- Accuracy – 91%
- Precision – on a macro average of 93%
- Recall – on a macro average of 92%.
- F1 – score – with an accuracy of 92%

KNN:



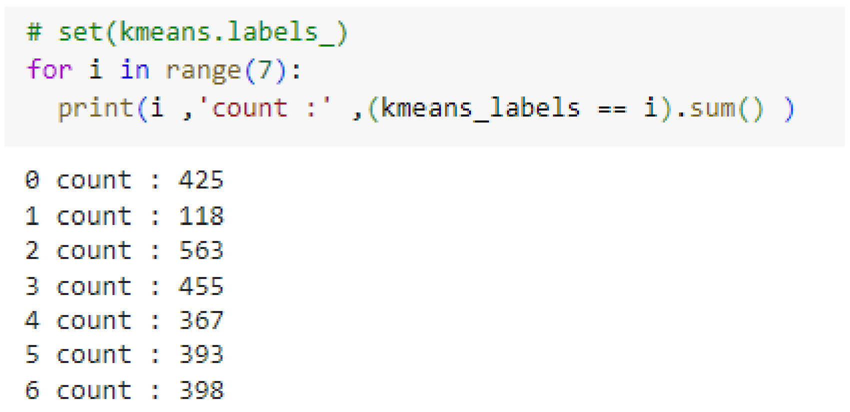
```
Model: knn
Accuracy: 0.917616770871644
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.84      0.88       270
           1       0.99      1.00      1.00       117
           2       0.90      0.94      0.92       319
           3       0.91      0.91      0.91       671
           4       0.95      0.95      0.95       377
           5       0.96      0.95      0.96       406
           6       0.86      0.89      0.88       559

    accuracy                           0.92      2719
   macro avg       0.93      0.93      0.93      2719
weighted avg       0.92      0.92      0.92      2719

Confusion Matrix:
[[227   1  26   0   3   2  11]
 [  0 117   0   0   0   0   0]
 [ 11   0 299   0   2   1   6]
 [  0   0   0 608   2  11  50]
 [  2   0   7   2 357   0   9]
 [  2   0   0  10   0 387   7]
 [  1   0   1  45  10   2 500]]
```

From the above results we have

- Accuracy – 92%
- Precision – on a macro average of 93%
- Recall – on a macro average of 93%.
- F1 – score – on a macro average of 93%

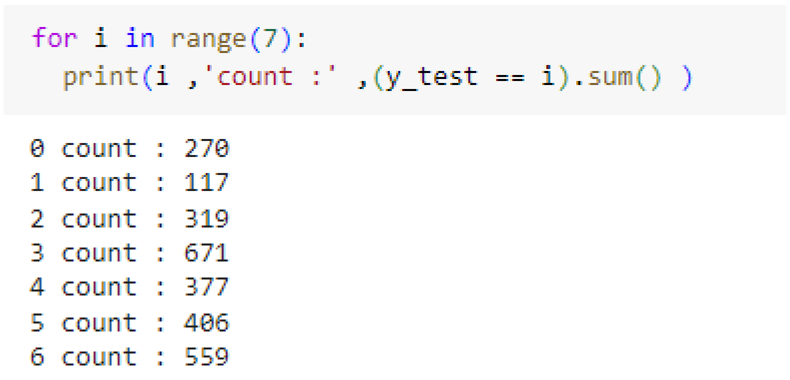
2.Unsupervised Learning:

1.K – Means:



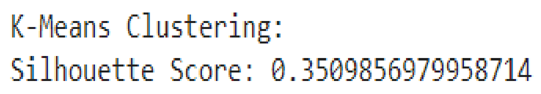
```
# set(kmeans.labels_)
for i in range(7):
  print(i ,'count :' ,(kmeans_labels == i).sum() )
```

```
0 count : 425
1 count : 118
2 count : 563
3 count : 455
4 count : 367
5 count : 393
6 count : 398
```

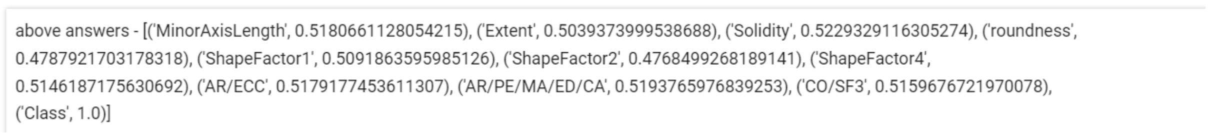


```
for i in range(7):
  print(i ,'count :' ,(y_test == i).sum() )
```

```
0 count : 270
1 count : 117
2 count : 319
3 count : 671
4 count : 377
5 count : 406
6 count : 559
```



```
K-Means Clustering:
Silhouette Score: 0.3509856979958714
```



above answers - [('MinorAxisLength', 0.5180661128054215), ('Extent', 0.5039373999538688), ('Solidity', 0.5229329116305274), ('roundness', 0.4787921703178318), ('ShapeFactor1', 0.5091863595985126), ('ShapeFactor2', 0.4768499268189141), ('ShapeFactor4', 0.5146187175630692), ('AR/ECC', 0.5179177453611307), ('AR/PE/MA/ED/CA', 0.5193765976839253), ('CO/SF3', 0.5159676721970078), ('Class', 1.0)]

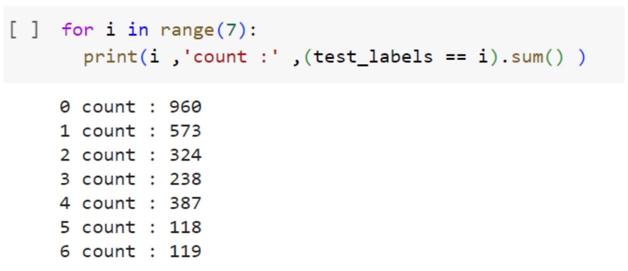
GPT interpretation

A silhouette score of 0.35 suggests that the clusters in your data are reasonably well-separated. Here's how to interpret the silhouette score:

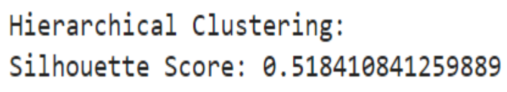
The silhouette score ranges from -1 to 1. A score close to 1 indicates that the data points are well-clustered, with each point being closer to its own cluster than to other clusters. A score close to 0 suggests overlapping clusters or clusters with similar densities. A negative score indicates that the data points might have been assigned to the wrong clusters. In your case, a silhouette score of 0.35 indicates that the clusters are somewhat well-separated, but there may still be some degree of overlap or suboptimal clustering. It's generally a decent score, but it's always essential to consider the context of your specific dataset and problem domain when interpreting clustering results. Additionally, visual inspection of the clusters can provide more insights into their quality and structure.

2.Hierarchical Clustering:

In this the performance is based on the silhouette score, ass per the model analysis the test results are generated as:



```
[ ]  for i in range(7):
       print(i ,'count :' ,(test_labels == i).sum() )

     0 count : 960
     1 count : 573
     2 count : 324
     3 count : 238
     4 count : 387
     5 count : 118
     6 count : 119
```



```
Hierarchical Clustering:
Silhouette Score: 0.518410841259889
```

6.Interpretation:

Both models function very well, with KNN marginally exceeding logistic regression in terms of accuracy, precision, recall, and F1-score, according to the assessment findings for the KNN and logistic regression algorithms that have been presented.

Accuracy: When compared to logistic regression (91%), KNN (92%) has a little greater accuracy. This suggests that all things considered, KNN could be more accurate in classifying cases. On a macro average, both models have the same accuracy of 93%, indicating that their performance in correctly detecting positive cases out of all instances projected as positive is comparable.

Recall: Once more, the recall of both models is the same (93% on a macro average), meaning that their ability to capture every positive case among all real positive instances is equal.

F1-score: The harmonic mean of recall and precision, or F1-score, is the same for both models (93% on a macro average), suggesting that their memory and accuracy ratios are comparable.

Several aspects might be considered when evaluating why one model would perform better than the other:

1.Model Complexity: Compared to KNN, which computes the distances between data points, logistic regression is a simpler model. Though in this example, KNN seems to handle the complexity effectively, simpler models may generalize well in other situations.
The selection and applicability of characteristics have a significant influence on the performance of the model. It's conceivable that the KNN method performs better at class discrimination because the characteristics utilized in this assessment are better suited for it.

2.Data Distribution: Since KNN depends on the separation between data points, it could function better when the data is organized into distinct groups or clusters. Since logistic regression is a linear model, it may not perform well if the data cannot be separated linearly.

3.Hyperparameters: Choosing the right number of neighbors (k) can have a significant impact on how well a KNN performs. Choosing an ideal value for k throughout the model's tuning process could help it perform better.

7. Conclusions:

In this project, we explored how to use supervised and unsupervised statistical learning approaches to evaluate and forecast data from the dry bean dataset. We prepared the dataset for model building by performing several data preparation stages, such as managing null values, encoding categorical characteristics, and scaling. KNN and logistic regression algorithms in supervised learning; KNN performed marginally better. K-Means clustering, Hierarchical clustering was used for unsupervised learning; a silhouette score indicated appropriate grouping. All things considered, the study showed how well supervised and unsupervised learning methods can analyse large, complicated datasets, with KNN turning out to be the best model for classification tasks in this situation. Subsequent research on feature design and hyperparameter tuning may improve the predictive power of models.