Team 9

Introduction to Statistical Learning

Project: Supervised and Unsupervised Models on Dry Bean Dataset.

Description: This Report gives a brief summary on Dry Beans Dataset and Implementing Supervised and Unsupervised methods and different metrics and

- **Data description**: how many samples? How many features? What type of features?

  Below are some of details about Dry bean Dataset:

  Samples: In given dataset, there are 13,611 Samples.

  Features: The Dry Bean Dataset consists of 16 features.

  Type of Features: The features are of mix of real and Integer types.

  Area: Area is the size of a bean zone as well as the number of pixels within its bounds.
  The length of a bean's border is used to calculate its circumference.

  Major axis length: The distance between the ends of the longest line drawn from a bean.

  Minor axis length: The longest line drawn from the bean while standing perpendicular to the main axis.

  Aspect ratio: Defines the relationship between the MajorAxisLength and the MinorAxisLength.

  Eccentricity: The ellipse's eccentricity is the same as that of the region.

ISL_Project_ | ISL Project_1 | ChatGPT | (3) WhatsAp | Team Projec | Team Projec | Grades for D | Paraphrasin

colab.research.google.com/drive/1wgX12SoVLI9A899902DC99yY-kkHqOm_#scrollTo=68jP4loOMj3B

ISL Project_16352878.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Colab AI

```
import pandas as pd

# Load your dataset
data = Dry_Bean_Dataset

# Display basic information about the dataset
print("Number of samples:", data.shape[0])  # Number of rows (samples)
print("Number of features:", data.shape[1])  # Number of columns (features)
print("\nData types of features:")
print(data.dtypes)  # Display data types of each feature
```

```
Number of samples: 13611
Number of features: 17

Data types of features:
Area              int64
Perimeter         float64
MajorAxisLength   float64
MinorAxisLength   float64
AspectRation      float64
Eccentricity      float64
ConvexArea        float64
EquivDiameter     float64
Extent            float64
Solidity          float64
roundness         float64
Compactness       float64
ShapeFactor1      float64
ShapeFactor2      float64
ShapeFactor3      float64
ShapeFactor4      float64
Class              object
dtype: object
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_c
  and should_run_async(code)
```

0s completed at 9:32 PM

ISL_Project_HC | ISL Project_1635 | (3) WhatsApp | Team Project Sp | Team Project Sp | Grades for Durg | Paraphrasing Te

colab.research.google.com/drive/1wgX12SoVLI9A899902DC99yY-kkHqOm_#scrollTo=OfYcsWRnNY8i

ISL Project_16352878.ipynb

File Edit View Insert Runtime Tools Help All changes saved

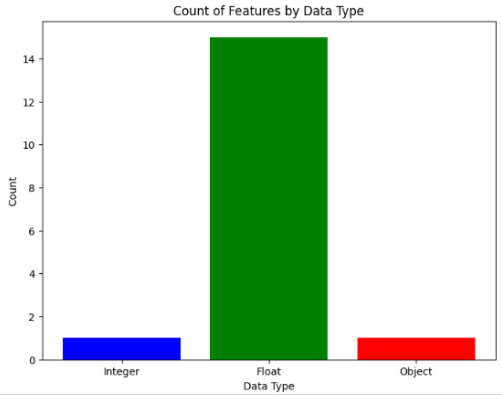
Comment Share

+ Code + Text

RAM Disk Colab AI

```
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_c
  and should_run_async(code)
```



0s completed at 9:47 PM

- **Data preprocessing**: are there any null values? How did you deal with them? How did you handle scaling?

  Yes, there are some Null Values for given Dry beans Dataset below images shows the number of null values present in each features:

(i)Finding the Null Values

```
#Finding number of Null Values
Null_number = Dry_Bean_Dataset.isnull().sum()
print("No. of  Null  Values:")
print(Null_number)
```

```
No. of  Null  Values:
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
dtype: int64
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkern
  and should_run_async(code)
```

<u>Dealing with Null Values:</u>

"**Dry_Bean_Dataset. Fillna (Dry_Bean_Dataset. Mean (), inplace=True)**"

The above code fills missing values in the Data Frame "Dry_Bean_Dataset" with the mean of each column. The function **Fillna ()** with the **Mean ()** method calculates the mean for each column and replaces missing values with their respective column means, modifying the DataFrame in place **inplace = True**

After processing the data, we have removed null values:

```
[14] Processed_data = Dry_Bean_Dataset.isnull().sum()
     print("No. of  Null  Values:")
     print(Processed_data)
```

```
No. of  Null  Values:
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
/usr/local/lib/python3.10/dist-packages/ipykernel/i
  and should_run_async(code)
```

Scaling:

Scaling is important because many machine learning algorithms are sensitive to the scale of the input features. Features with larger scales can dominate those with smaller scales, leading to biased model predictions. By scaling the features to a common range, MinMaxScaler helps to mitigate this issue and ensures that all features contribute equally to the model's learning process.

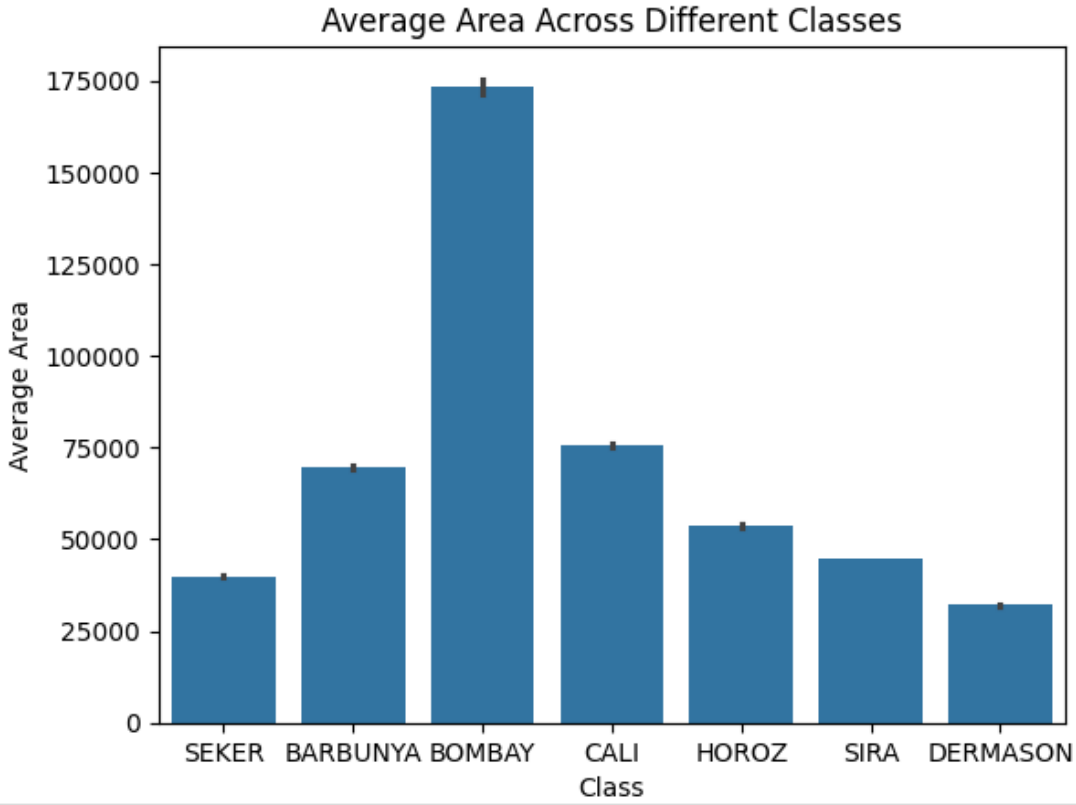
We have used **Min Max scaling** for our model.

The MinMaxScaler is a preprocessing technique used to scale numerical features to a specified range, typically between 0 and 1. Scaling is an important step in many machine learning algorithms because it helps to normalize the features and ensure that they have a similar scale, which can improve the performance of the model and the convergence of optimization algorithms.

- **Exploratory data Analysis**: visualize the data with graphs and describe your findings.

  We have have shown different visualizations for given dataset,

  Plot_1: Bar Plot: The Bar Plot below represents the average area of seven different classes or Beans: SEKER, BARBUNYA, BOMBAY, CALI, HOROZ, SIRA, and DERMASON.

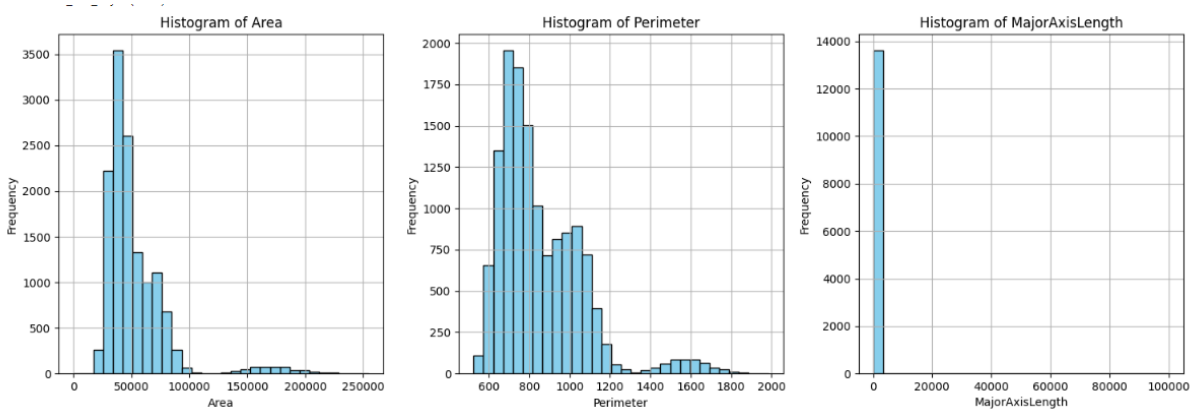


The Plot above summarizes that the Class or bean BOMBAY has a higher average area compared to other classes with 173485. The other classes have lower average areas when compared with other classes with average of

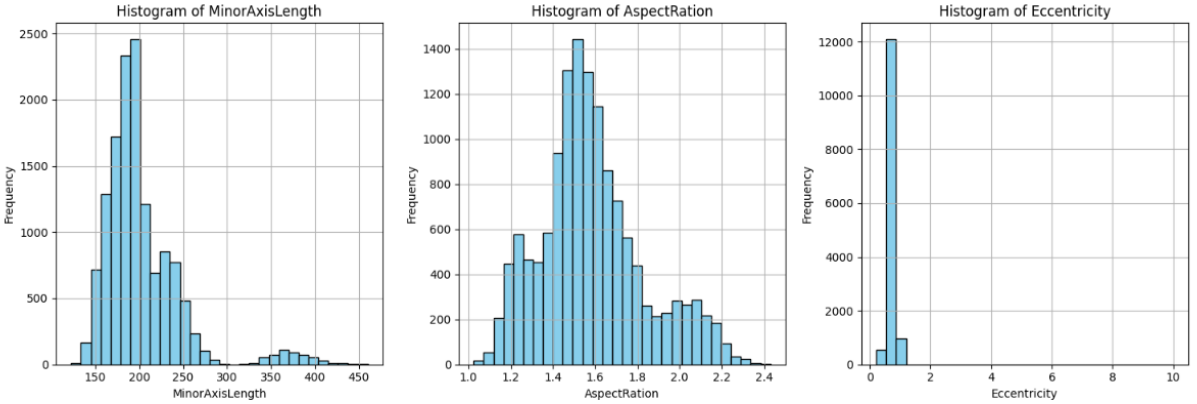
BARBUNYA - 69804
CALI - 75538
DERMASON 32118
HOROZ 53648
SEKER 39863
SIRA 44729

Plot_2: Histogram: Below is Histogram plot of different features in given Dataset vs. Frequency.

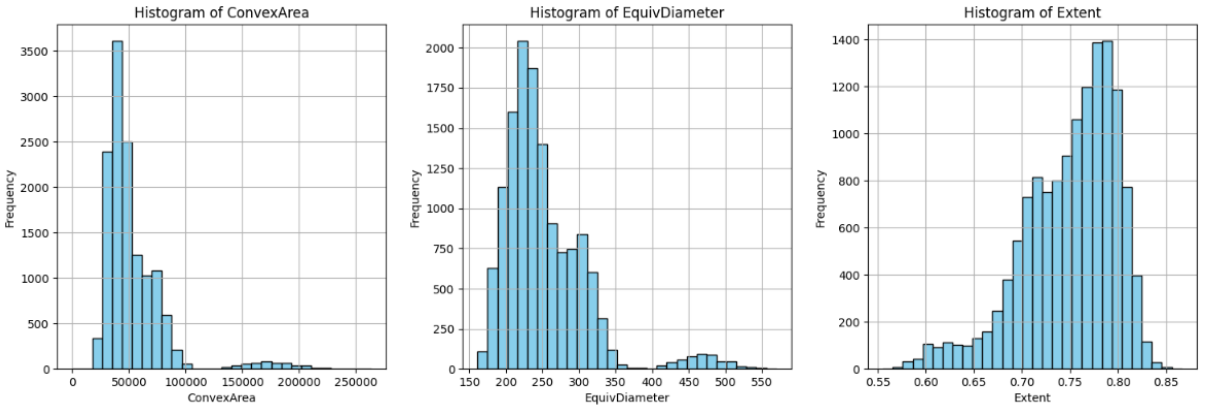
Plot2.1: The Histogram shows the distribution of Area, Perimeter, Major Axis Length and frequency



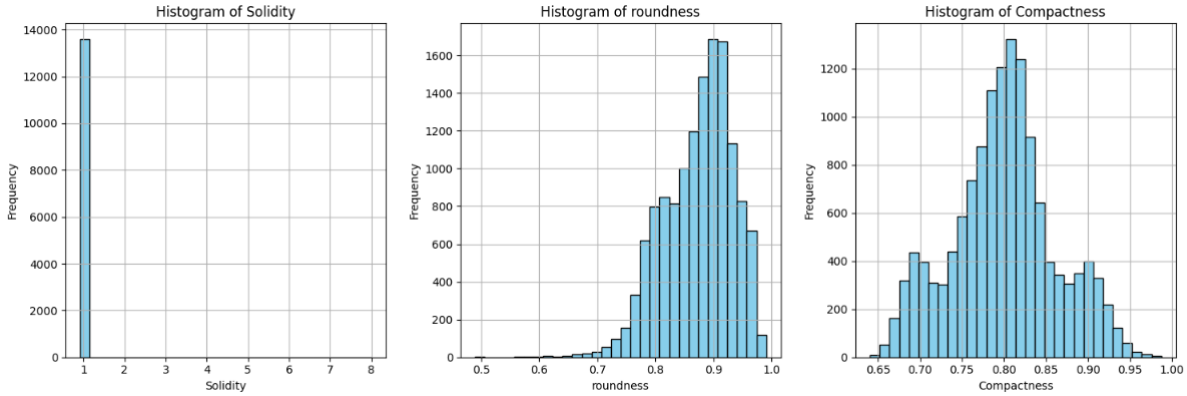
Plot 2.2: The Plot below shows the distribution of MinorAxisLength, Aspect Ratio, Eccentricity Vs. Frequency



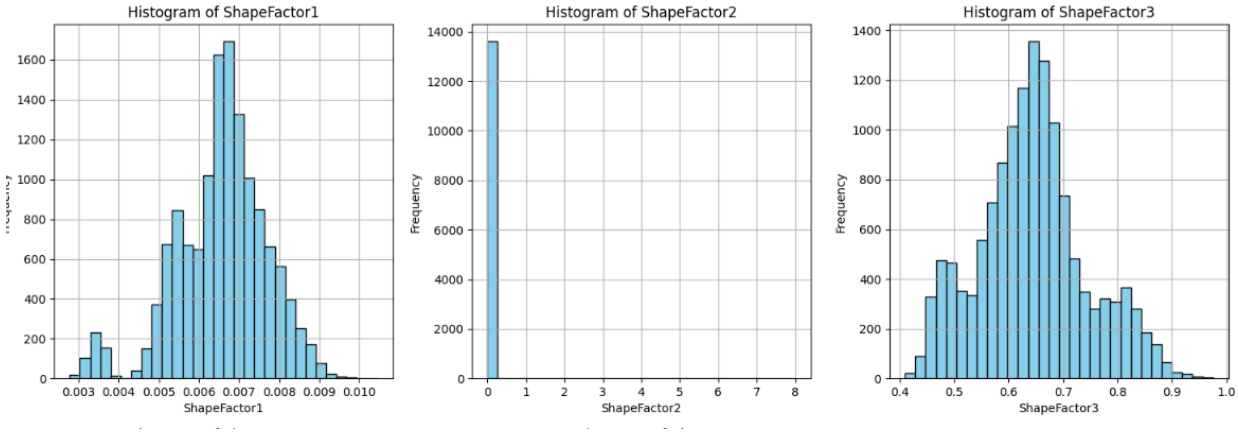
Plot 2.3: The Plot below shows the distribution of Convex Area, EquivDiameter, Extent Vs. Frequency



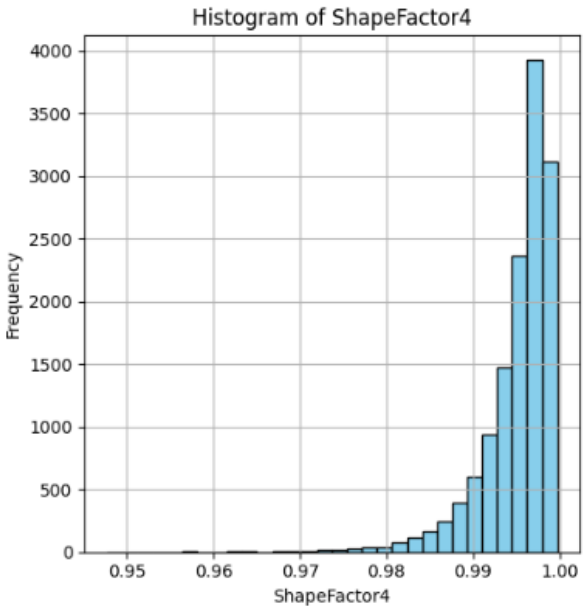
Plot 2.4: The Plot below shows the distribution of Solidity, Roundness, Compactness Vs. Frequency

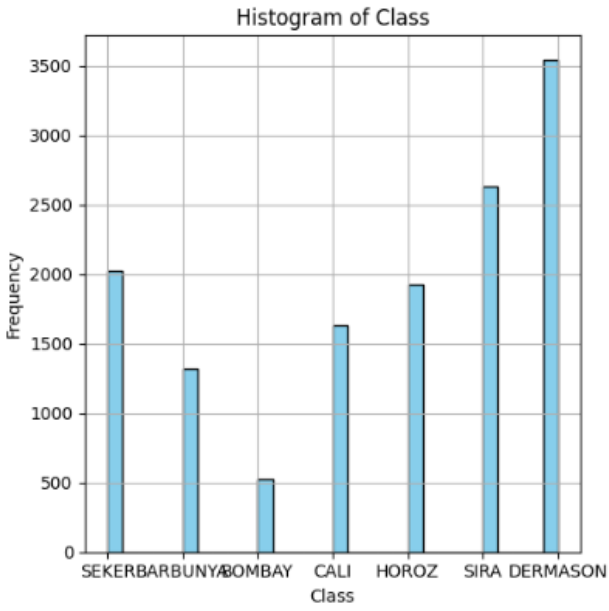


Plot 2.5: The Plot below shows the distribution of ShapeFactor1, ShapeFactor2, ShapeFactor3 Vs. Frequency



Plot 2.6: The Plot below shows the distribution of ShapeFactor4, Class vs. Frequency





Count of Total Classes:

The below Table shows number of classes with their counts, in the given dataset we have seven types of classes, Dermason class highest count of 3546 among all classes and Bombay has least count of 522

Total Counts of Each Class

Class	Count
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522

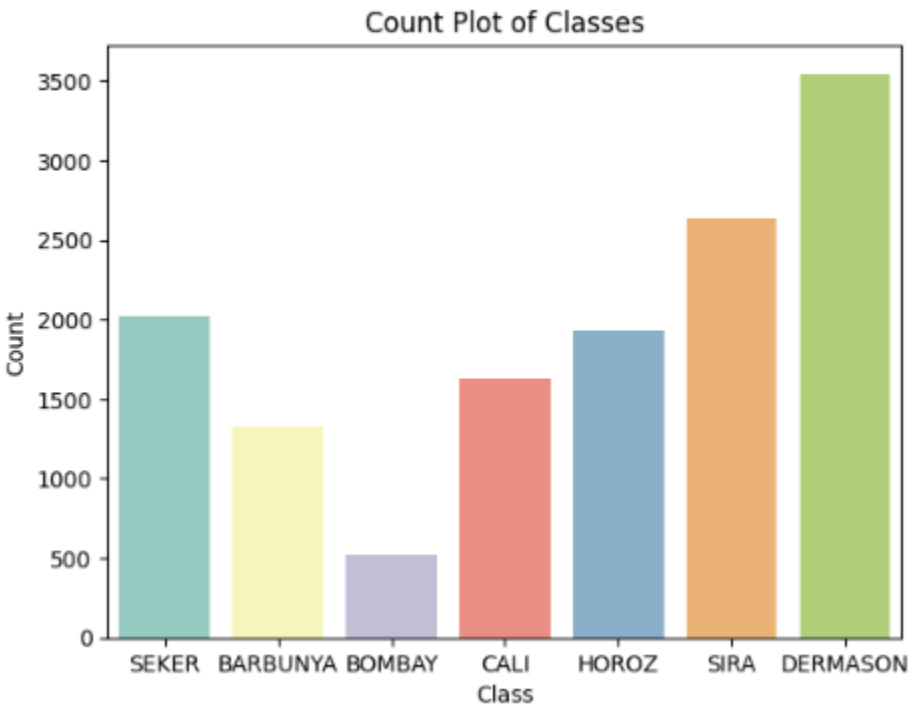
Plot_3: Count Plot of Classes:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'Dry_Bean_Dataset' contains the dataset
# Let's say 'Class' is a categorical feature


# Creating a count plot
sns.countplot(x='Class', data=Dry_Bean_Dataset, palette='Set3')
plt.title('Count Plot of Classes')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

The Below plot shows count plot of different classes with
Dermason (Green bar) has maximum count of 3500
SIRA (Orange bar) has count of 2500
HOROZ (Blue bar) has count of 1800
CALI (Orange bar) has count of 1500
SEKER (teal bar) has count of 2000
BARBUNYA (Yellow bar) has count of 1400
Bombay (Purple Bar) has lowest count of 500



Plot_4: Heat Map:

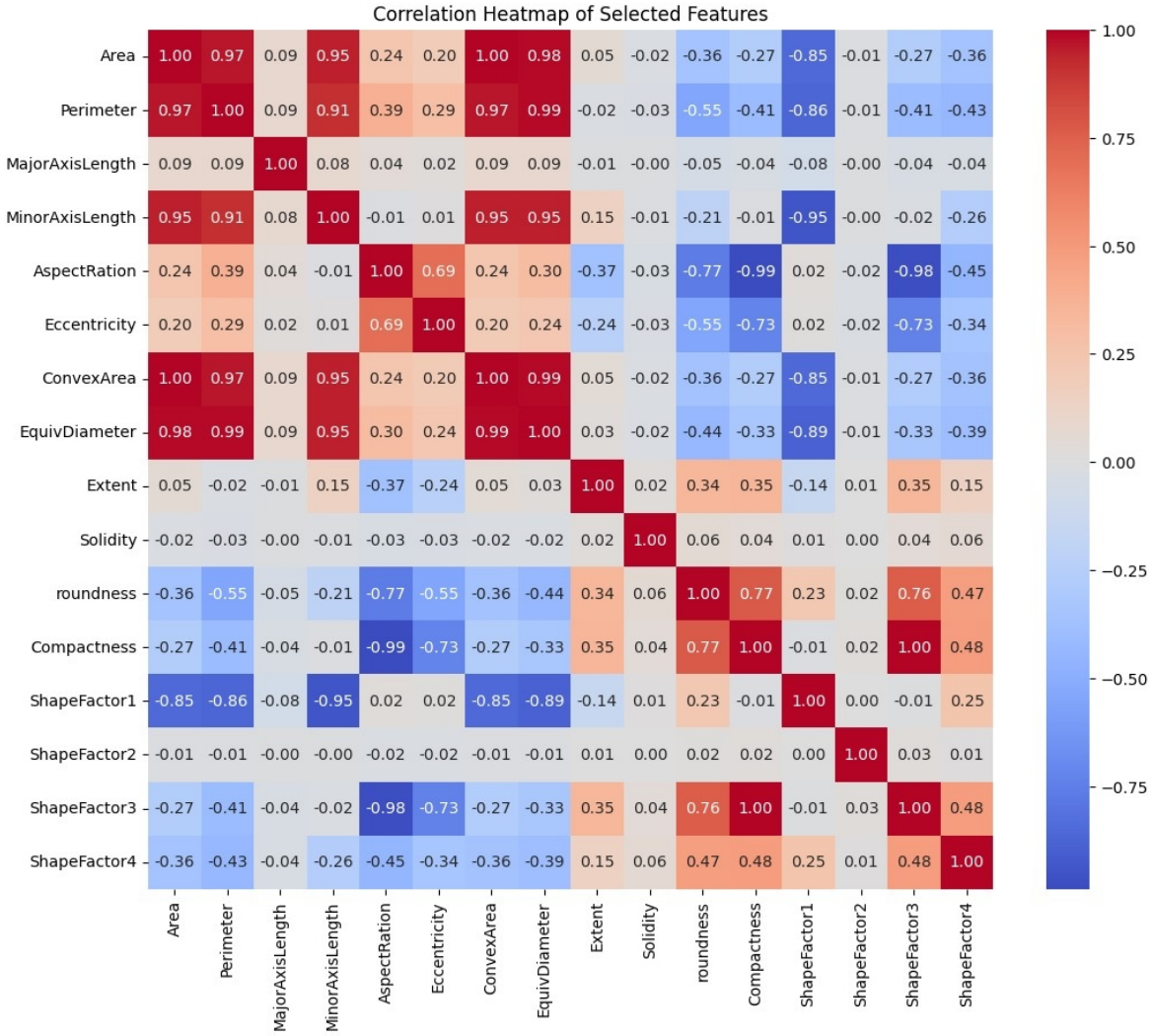
```
[ ]  import seaborn as sns
     import matplotlib.pyplot as plt

     # Selecting features
     selected_features = ['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength',
                          'AspectRation', 'Eccentricity', 'ConvexArea', 'EquivDiameter',
                          'Extent', 'Solidity', 'roundness', 'Compactness', 'ShapeFactor1',
                          'ShapeFactor2', 'ShapeFactor3', 'ShapeFactor4', 'Class']

     # Create a DataFrame with only the selected features
     selected_data = Dry_Bean_Dataset[selected_features]

     # Calculate the correlation matrix
     correlation_matrix = selected_data.corr()

     # Plot the heatmap
     plt.figure(figsize=(12, 10))
     sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
     plt.title("Correlation Heatmap of Selected Features")
     plt.show()
```



The above heat map generated represents the correlation matrix for 17 features present in our data set, where each cell in above heatmap represents correlation between two variables.

The range of values for this correlation coefficient is between “-1.0 to “1.0”, If the correlation is greater than “0” , It has a positive relationship if it is less than “0” it has negative relationship, if the value is equal to zero has no relation between the two variables.

Colour Representation in Heat Map:

Blue colors indicate Positive correlations, it represents that if one feature increases then other increases they are dependent.

Red colors represents negative correlations, it represents that if one feature increases then other decreases.

Note: The darker the color, the stronger the correlation, it means a dark blue cell indicates a strong positive correlation.

The numeric values in the cell represents the calculates correlation coefficients for each pair of features

Model development: state the hyper parameters selected for the models and how/why you selected those hyper parameters

Logistic regression:

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Define the logistic regression model with the "multinomial" strategy for multi-class classification
logistic_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)

# Train the model
logistic_model.fit(X_train, y_train)

# Predict on the test set
y_pred = logistic_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)
```

For the above Logistic Regression model, we have set random state as a hyper parameter, which is set to 43, it is used for initializing the random number generator , which will decide the splitting of data into train and test.

- **Performance evaluation**:

  Logistic Regression:

```
Accuracy: 0.8721841332027425
Classification Report:
              precision    recall  f1-score   support

    BARBUNYA       0.90      0.72      0.80       400
      BOMBAY       0.99      1.00      0.99       150
        CALI       0.85      0.94      0.89       500
    DERMASON       0.88      0.90      0.89      1107
       HOROZ       0.92      0.87      0.89       579
       SEKER       0.92      0.88      0.90       610
        SIRA       0.78      0.84      0.81       738

    accuracy                           0.87      4084
   macro avg       0.89      0.88      0.88      4084
weighted avg       0.87      0.87      0.87      4084

Confusion Matrix:
[[290   0  63   0   8   0  39]
 [  0 150   0   0   0   0   0]
 [ 19   2 470   0   8   1   0]
 [  0   0   0 992   0  35  80]
 [  0   0  21  47 501   0  10]
 [ 11   0   0  16   2 539  42]
 [  3   0   2  74  25  14 620]]
```

Below are metrics that we achieved for logistic regression,

Accuracy: for this model, we have achieved an accuracy of 0.87% or 87% which shows that model correctly predicted the class 92% of the time

Precision: It is ratio of correctly predicted positive observations to the total predicted positives. It means that high precision relates to low positive rate. In our case, the class “bar bunya” is 0.90, where 90% of “bar bunya” are correctly predicted.

Recall: It states that ratio of correctly predicted +ve classes observations to all observations in actual class, the recall for “bar bunya” is 0.72, it means that our model correctly predicted 72% of all “bar bunya” classes.

F1-Score: It is weighted average of precision and recall, it takes both false positives and false negatives into account.

Confusion Matrix:

The confusion matrix is a table that shows the performance of a classification model.

The numbers going from the top left to the bottom right (290, 150, 470, 992, 501, 539, and 620) are the times the model guessed right. Each of these numbers is a score for how many times the model got each type of bean right.

The other numbers in the table are the times the model made mistakes. These are the times when the model guessed one type of bean, but it was actually a different type.

Model Development –KNN :

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors
knn.fit(X_train, y_train)

# Predict on the test set
y_pred = knn.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Compute precision
precision = precision_score(y_test, y_pred, average='weighted')  # You can change 'average' parameter as needed
print("Precision:", precision)

# Compute recall
recall = recall_score(y_test, y_pred, average='weighted')  # You can change 'average' parameter as needed
print("Recall:", recall)

# Compute F1-score
f1 = f1_score(y_test, y_pred, average='weighted')  # You can change 'average' parameter as needed
print("F1-score:", f1)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Performance evaluation – KNN:

```
Accuracy: 0.9214102093279471
Precision: 0.9223651314721464
Recall: 0.9214102093279471
F1-score: 0.9216504475764851
Confusion Matrix:
[[233   0  19   0   1   1   7]
 [  0 117   0   0   0   0   0]
 [  6   0 304   0   4   1   2]
 [  0   0   0 610   2   7  52]
 [  1   0   9   4 387   0   7]
 [  3   0   0  12   0 388  10]
 [  1   0   0  55   6   4 470]]
```

Accuracy: The percentage of true predictions made by the model out of all forecasts. Your model's accuracy is roughly 0.9214, or 92.14%, indicating that it correctly predicted the class 92.14% of the time.
Precision is the fraction of true positive predictions (properly predicted positives) to all predicted positives. The precision of your model is roughly 0.9224, or 92.24%.
Recall: This is the percentage of genuine positive forecasts among all actual positives. The recall of your model is roughly 0.9214, or 92.14%.
The F1-score is the harmonic mean of precision and recall, resulting in a balanced assessment of precision and recall. An F1 score achieves its maximum value at 1 (perfect precision and recall) and its lowest at 0.
Confusion Matrix:

The diagonal elements reflect the number of points for which the predicted label is equal to the true label, and the off-diagonal elements are those that are mislabeled by the classifier.

Model Development – K_Means Implementations:

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score


kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the K-Means model to the data
kmeans.fit(X)

# Get the cluster labels for each sample
cluster_labels = kmeans.labels_

# Add the cluster labels to the dataframe
data['Cluster'] = cluster_labels

# Visualize the clusters (assuming 2D data)
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=cluster_labels, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-Means Clustering')
plt.colorbar(label='Cluster')
plt.show()

# Calculate silhouette score
silhouette_avg = silhouette_score(X, cluster_labels)
print("Silhouette Score:", silhouette_avg)
```

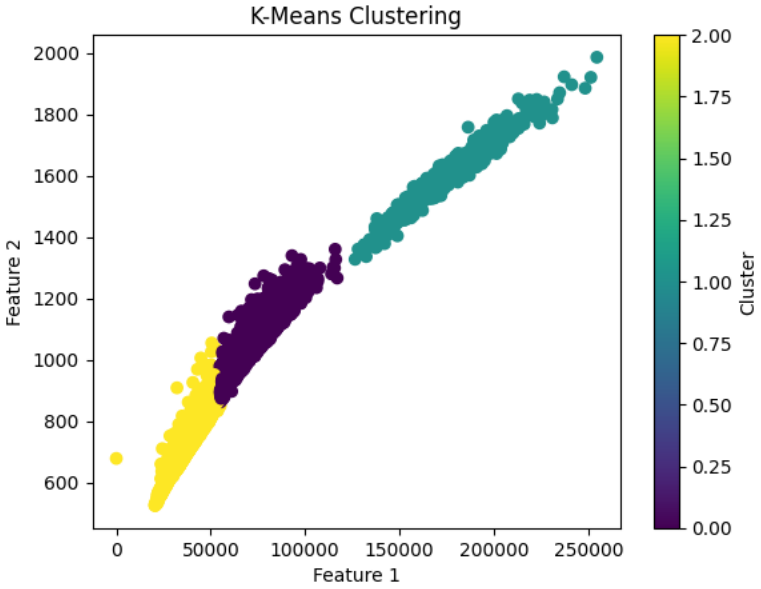
The above code if implementation for K-means clustering model is trained, the hyper parameters used are clusters and random state.

Clusters: It states that no. of clusters to form and no. of centroids to generate, in our model we have set to 3, which means the k-means algorithm will group the data into 3 clusters

Random state: it is used for initializing the internal random number generator which decide the random initialization of the centroids.

Selecting clusters is major step in K-means clustering ,When we do clustering, which is like grouping similar things together, we often don't know how many groups (or "clusters") we should make. So, we have to guess and check with different numbers of groups.

We make a bunch of models, each with a different number of groups, and then see which one does the best job grouping our data. We decide which one is best by using a score, kind of like in a game. In this case, the score is called the "silhouette score".



Performance evaluation - Kmeans:

We have evaluated the performance of our model using the silhouette score score, which measures how much an object is similar to its own when compared to other clusters.

The graph is clear that the model was clearly able to classify the group of beans.

The silhouette score is like a report for our grouping. It ranges between -1 and 1.If it's close to 1, that means our model is best. The items in the same group are very similar to each other, and very different from items in other groups.

If it's close to -1 that means our model didn't do a good job. The items in the same group aren't that similar to each other, and they might be similar to items in other groups.

If the score is around 0, that means the items are kind of mixed up and it's hard to tell the groups apart.

We got a silhouette score 0.6643623692147601, that is the model performs well at this setting. We tested with different cluster counts but count = 3, gave us best results.

Model Development – Hierarchical Clustering Implementations:

Hierarchical Clustering

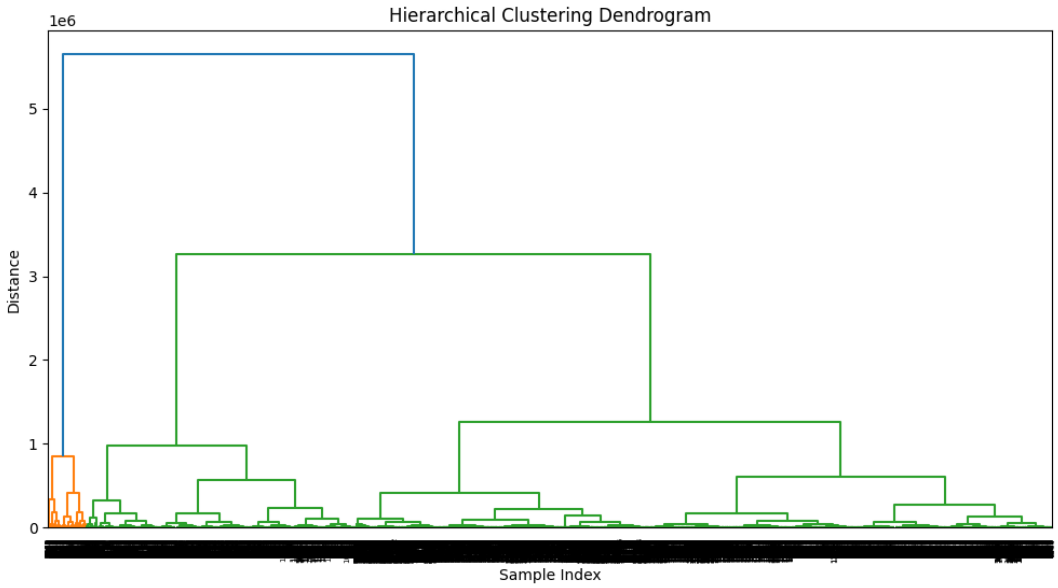
```
import pandas as pd
from sklearn.impute import SimpleImputer
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform hierarchical clustering using AgglomerativeClustering
agg_cluster = AgglomerativeClustering(n_clusters=5)

# Fit the AgglomerativeClustering model to the data
cluster_labels = agg_cluster.fit_predict(X)

# Plot the dendrogram for hierarchical clustering
Z = linkage(X, method='ward')
plt.figure(figsize=(12, 6))
dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

Performance evaluation



We used hierarchical clustering to group our data into 5 clusters using the AgglomerativeClustering method.

This method starts by considering each data point as its own cluster and then merges similar clusters iteratively until the desired number of clusters is reached.

We fit the hierarchical clustering model to our data and obtain cluster labels for each data point.

We create a dendrogram, which is a tree-like diagram showing how clusters are merged based on their similarity. By using the Ward as linkage method, we performed the clustering.

Performance evaluation - Hierarchical Clustering:
The silhouette score ranges from -1 to 1.
A score of 1 indicates perfectly clustered data, where each point is far away from other clusters.
A score of 0 means overlapping clusters or clusters that are too close to each other.
A negative score suggests that data points may be assigned to the wrong cluster.
Interpretation of 0.531:
We got a silhouette score of 0.531 is closer to 1 than to 0, which is a positive sign.
It suggests that the clusters are reasonably well-separated and distinct.
However, there is still room for improvement to achieve more clearly separated clusters.

Interpretation

1. Accuracy, Precision, Recall, and F1-score: These metrics provide an overall assessment of the model's performance. Higher values indicate better performance. Model 2 appears to have the highest accuracy, precision, recall, and F1-score among the models provided.

2. Confusion Matrix: This matrix provides insights into the model's performance for each class. We can observe how well the model is correctly classifying instances for each class and where misclassifications are occurring.

3. Classification Report: This report provides a detailed breakdown of precision, recall, and F1-score for each class. It helps us understand how well the model is performing for each class individually.

4. Silhouette Score: For Model 3 and Model 4, the silhouette score is provided. This score measures how similar an object is to its own cluster compared to other clusters. Higher silhouette scores indicate better-defined clusters.

Based on these considerations:

- Model 2 seems to have the best overall performance with the highest accuracy, precision, recall, and F1-score. It also has high precision, recall, and F1-score values for each class, indicating balanced performance across all classes.

- Model 3has a relatively high silhouette score, indicating well-defined clusters in the data. However, without additional information about the model's performance on unseen data, it's challenging to assess its overall effectiveness.

-Model 4 has a lower silhouette score compared to Model 3, suggesting less distinct clusters. This could indicate that the clustering algorithm used in Model 4 may not be as effective in separating the data into meaningful clusters.

Regarding the most significant features, without access to the feature importance or coefficients from the models, it's challenging to determine the most significant features definitively. However, feature

importance analysis or examining coefficients from models like logistic regression or linear SVM could provide insights into which features contribute most to the model's predictions.

Conclusions:.

We as team would like to conclude that the classification models (Knn and Logistic) performed well in predicting class labels, while Model –Kmeans exhibited superior clustering quality compared to Model Hierarchical clustering.