Dry Beans Classification

IE 7300 Statistical Learning Final Project

Group 3

Harsha Bhargav Yarramsetty - yarramsetty.h@northeastern.edu
Abhigna Reddy Musku - musku.ab@northeastern.edu
Shamhith Kamasani - kamasani.s@northeastern.edu

Percentage of contribution by Harsha Bhargav Yarramsetty: 33.33%
Percentage of contribution by Abhigna Reddy Musku: 33.33%
Percentage of contribution by Shamhith Kamasani: 33.33%

Signature of Harsha Bhargav: Harsha Bhargav Yarramsetty
Signature of Abhigna Reddy Musku: Abhigna Reddy Musku
Signature of Shamhith Kamasani: Shamhith Kamasani

Submission Date: 04/17/2023

Abstract:

This study focuses on the classification of a dry beans dataset using three machine learning algorithms: logistic regression, support vector machine (SVM), and neural network (NN). The goal of the project is to evaluate the performance of these algorithms on a challenging classification task in the agricultural industry.

To achieve this goal, the researchers first preprocessed the data and split it into training and testing sets. They then trained each algorithm on the training set and evaluated their performance using various performance metrics such as accuracy, precision, and recall. The results showed that SVM outperformed both logistic regression and NN with an accuracy of 0.976.

This project has important implications for the field of machine learning, as it demonstrates the effectiveness of different algorithms for classification tasks in complex and diverse datasets. Additionally, it highlights the importance of preprocessing data and selecting appropriate performance metrics to accurately evaluate the performance of machine learning models.

Overall, this project provides valuable insights into the use of machine learning algorithms for classification tasks in real-world applications and can serve as a foundation for future research in this field.

Introduction:

Dry beans are one of the most widely produced and consumed legume crops in the world, with a significant impact on global agriculture and food security. However, the quality of dry beans can vary greatly depending on various factors, such as genetic diversity, growing conditions, and harvesting methods. This has led to a growing interest in developing effective methods for seed classification and quality control, which are essential for ensuring sustainable agricultural systems and providing consumers with high-quality products.

In this project, we aim to explore the use of various machine learning algorithms and techniques to classify the 7 types of Dry Beans and evaluate their performance.

Problem Definition:

The Dry Beans dataset contains features for seven different types of dry beans, and the goal is to develop an effective classification model that can accurately identify and classify each type based on their physical characteristics and properties. The challenge is to find a machine learning

algorithm or technique that can handle the complex and high-dimensional nature of the dataset, while also achieving high accuracy and robustness in the classification task.

Data Resources:

The data for the analysis is taken from UC Irvine's Machine Learning Repository.
https://archive-beta.ics.uci.edu/dataset/602/dry+bean+dataset

Data Description:

The dataset contains 16 features, 13,611 instances, and 7 outcome classes of different types of dry beans. Each record in the Dry Beans dataset represents a single bean zone in an image.

Independent Features:

The features of this dataset are

1) Area (A): The area of a bean zone and the number of pixels within its boundaries.
2) Perimeter (P): Bean circumference is defined as the length of its border.
3) Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4) Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5) Aspect ratio (K): Defines the relationship between L and l.
6) Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7) Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8) Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9) Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10) Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11) Roundness (R): Calculated with the following formula: (4piA)/(P^2)
12) Compactness (CO): Measures the roundness of an object: Ed/L
13) ShapeFactor1 (SF1)
14) ShapeFactor2 (SF2)
15) ShapeFactor3 (SF3)
16) ShapeFactor4 (SF4)

Dependent Feature:

The dependent variable or target variable in this dataset is the "Class" column, which indicates the type of the dry bean. There are 7 types of dry beans represented in this dataset, which are: BARBUNYA, BOMBAY, CALI, DERMASON, HOROZ, SEKER and SIRA.

Data Understanding & Pre-processing:

From analyzing the data, we can see that out of the 16 variables in the dataset, variable - Class is the dependent target variable we are trying to predict, and it is a categorical variable.
A view of features and target variable, type, unique values and missing values and duplicates:

	Dtype	Unique	Duplicated	Mean	Median	Skewness
Area	int64	12011	68	53048.285	44652.000	2.952931
Perimeter	float64	13416	68	855.283	794.941	1.626124
MajorAxisLength	float64	13543	68	320.142	296.883	1.357815
MinorAxisLength	float64	13543	68	202.271	192.432	2.238211
AspectRation	float64	13543	68	1.583	1.551	0.582573
Eccentricity	float64	13543	68	0.751	0.764	-1.062824
ConvexArea	int64	12066	68	53768.200	45178.000	2.941821
EquivDiameter	float64	12011	68	253.064	238.438	1.948958
Extent	float64	13535	68	0.750	0.760	-0.895348
Solidity	float64	13526	68	0.987	0.988	-2.550093
roundness	float64	13543	68	0.873	0.883	-0.635749
Compactness	float64	13543	68	0.800	0.801	0.037115
ShapeFactor1	float64	13543	68	0.007	0.007	-0.534141
ShapeFactor2	float64	13543	68	0.002	0.002	0.301226
ShapeFactor3	float64	13543	68	0.644	0.642	0.242481
ShapeFactor4	float64	13543	68	0.995	0.996	-2.759483

This view gives us information about non-Null values and the Datatype of values in each feature. We have three datatypes for values in this dataset: - int, float and object.

```
Shape of the data (13611, 17)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13611 entries, 0 to 13610
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             13611 non-null  int64
 1   Perimeter        13611 non-null  float64
 2   MajorAxisLength  13611 non-null  float64
 3   MinorAxisLength  13611 non-null  float64
 4   AspectRation     13611 non-null  float64
 5   Eccentricity     13611 non-null  float64
 6   ConvexArea       13611 non-null  int64
 7   EquivDiameter    13611 non-null  float64
 8   Extent           13611 non-null  float64
 9   Solidity         13611 non-null  float64
 10  roundness        13611 non-null  float64
 11  Compactness      13611 non-null  float64
 12  ShapeFactor1     13611 non-null  float64
 13  ShapeFactor2     13611 non-null  float64
 14  ShapeFactor3     13611 non-null  float64
 15  ShapeFactor4     13611 non-null  float64
 16  Class            13611 non-null  object
dtypes: float64(14), int64(2), object(1)
memory usage: 1.8+ MB
None
```

The size of the data is 13611X17

We have 13,611 rows and 16 columns in the data set, on which we will perform different data explorations and visualizations to see the correlations and see how we can impute or fill the null values.

Data Exploration & Visualization:

1)Finding Null values in the dataset

This dataset has doesn't have any null values

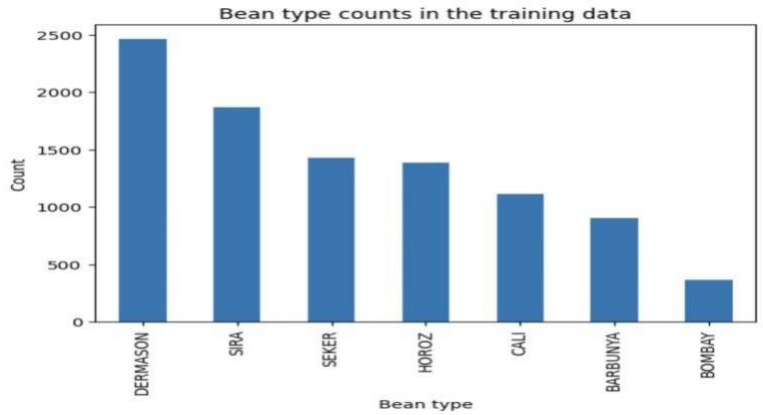
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
```

Numerical Features:

- This table gives info about the count, standard deviation, minimum, maximum and the 4-percentile cut off values for the 16 numerical features

	count	mean	std	min	25%	50%	75%	max
Area	13611.0	53048.2845	29324.0957	20420.0000	36328.0000	44652.0000	61332.0000	254616.0000
Perimeter	13611.0	855.2835	214.2897	524.7360	703.5235	794.9410	977.2130	1985.3700
MajorAxisLength	13611.0	320.1419	85.6942	183.6012	253.3036	296.8834	376.4950	738.8602
MinorAxisLength	13611.0	202.2707	44.9701	122.5127	175.8482	192.4317	217.0317	460.1985
AspectRation	13611.0	1.5832	0.2467	1.0249	1.4323	1.5511	1.7071	2.4303
Eccentricity	13611.0	0.7509	0.0920	0.2190	0.7159	0.7644	0.8105	0.9114
ConvexArea	13611.0	53768.2002	29774.9158	20684.0000	36714.5000	45178.0000	62294.0000	263261.0000
EquivDiameter	13611.0	253.0642	59.1771	161.2438	215.0680	238.4380	279.4465	569.3744
Extent	13611.0	0.7497	0.0491	0.5553	0.7186	0.7599	0.7869	0.8662
Solidity	13611.0	0.9871	0.0047	0.9192	0.9857	0.9883	0.9900	0.9947
roundness	13611.0	0.8733	0.0595	0.4896	0.8321	0.8832	0.9169	0.9907
Compactness	13611.0	0.7999	0.0617	0.6406	0.7625	0.8013	0.8343	0.9873
ShapeFactor1	13611.0	0.0066	0.0011	0.0028	0.0059	0.0066	0.0073	0.0105
ShapeFactor2	13611.0	0.0017	0.0006	0.0006	0.0012	0.0017	0.0022	0.0037
ShapeFactor3	13611.0	0.6436	0.0990	0.4103	0.5814	0.6420	0.6960	0.9748
ShapeFactor4	13611.0	0.9951	0.0044	0.9477	0.9937	0.9964	0.9979	0.9997

The graph below gives us information about the counts of each class. Dermason is the most frequent class with ‘2496’ beans and Bombay is the least frequent class with ‘365’ beans. This huge difference between the 2 classes, should be considered while building a model, otherwise there might be a risk of the model being biased.



This view gives us a description of all the variables included is available, along with their respective mean and median values, which can be used to assess the degree of skewness in the

data. This information can aid in determining the distribution of the data and identifying potential outliers.

	Dtype	Unique	Duplicated	Mean	Median	Skewness
Area	int64	12011	68	53048.285	44652.000	2.952931
Perimeter	float64	13416	68	855.283	794.941	1.626124
MajorAxisLength	float64	13543	68	320.142	296.883	1.357815
MinorAxisLength	float64	13543	68	202.271	192.432	2.238211
AspectRation	float64	13543	68	1.583	1.551	0.582573
Eccentricity	float64	13543	68	0.751	0.764	-1.062824
ConvexArea	int64	12066	68	53768.200	45178.000	2.941821
EquivDiameter	float64	12011	68	253.064	238.438	1.948958
Extent	float64	13535	68	0.750	0.760	-0.895348
Solidity	float64	13526	68	0.987	0.988	-2.550093
roundness	float64	13543	68	0.873	0.883	-0.635749
Compactness	float64	13543	68	0.800	0.801	0.037115
ShapeFactor1	float64	13543	68	0.007	0.007	-0.534141
ShapeFactor2	float64	13543	68	0.002	0.002	0.301226
ShapeFactor3	float64	13543	68	0.644	0.642	0.242481
ShapeFactor4	float64	13543	68	0.995	0.996	-2.759483

We find that there are 68 duplicate values in this dataset.

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	Shape
5505	33518	702.956	277.571399	154.305581	1.798842	0.831240	34023	206.582775	0.808383	0.985157	0.852377	0.744251	
5509	33954	716.750	277.368480	156.356326	1.773951	0.825970	34420	207.922042	0.799482	0.986461	0.830549	0.749624	
5548	38427	756.323	306.533886	160.591784	1.908777	0.851782	38773	221.193978	0.796976	0.991076	0.844174	0.721597	
5554	38891	791.343	319.499996	156.869619	2.036723	0.871168	39651	222.525412	0.650025	0.980833	0.780422	0.696480	
5599	40804	790.802	323.475648	163.287717	1.981016	0.863241	41636	227.932592	0.787570	0.980017	0.819931	0.704636	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7263	63408	1005.966	412.551649	196.337705	2.101235	0.879494	64200	284.136539	0.798791	0.987664	0.787385	0.688730	
7278	63882	1004.206	411.263403	198.765453	2.069089	0.875452	64663	285.196579	0.754705	0.987922	0.796054	0.693465	
7285	63948	996.497	412.297178	198.877557	2.073121	0.875971	64641	285.343867	0.777909	0.989279	0.809254	0.692083	
7340	65766	1035.842	406.416622	207.242369	1.961069	0.860218	66698	289.371512	0.792295	0.986027	0.770237	0.712007	
7342	65781	1039.257	409.713859	204.992832	1.998674	0.865834	66762	289.404510	0.642549	0.985306	0.765358	0.706358	

68 rows × 17 columns

After the Data exploration, we perform these two tasks: -

1) Dropping the duplicated rows

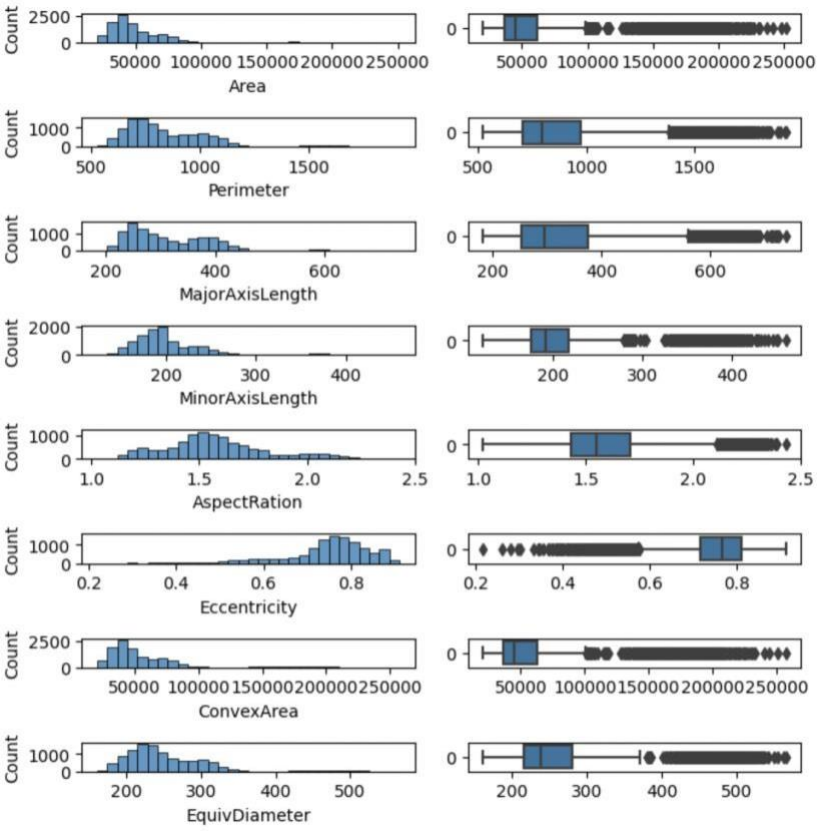
2) Setting aside 20% of the data for testing which we will not use in model building

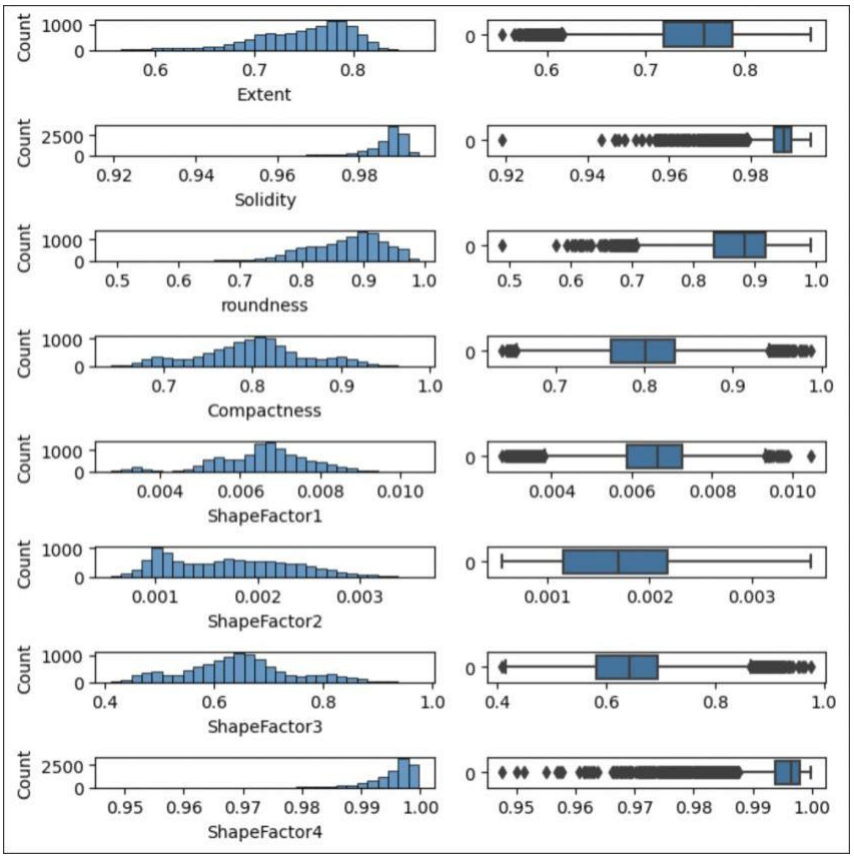
	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1
0	36292	707.300	263.729391	175.674695	1.501237	0.745847	36678	214.961414	0.759120	0.989476	0.911619	0.815083	0.007267
1	69845	1050.030	416.157251	218.251925	1.906775	0.851444	72211	298.210355	0.783604	0.967235	0.796053	0.716581	0.005958
2	33682	691.390	263.216456	163.669292	1.608221	0.783172	34106	207.087552	0.712756	0.987568	0.885445	0.786758	0.007815
3	20942	530.683	191.176525	139.586402	1.369593	0.683293	21191	163.291710	0.742361	0.988250	0.934453	0.854141	0.009129
4	45047	796.038	298.762324	192.785476	1.549714	0.763946	45589	239.490338	0.738730	0.988111	0.893322	0.801608	0.006632

ShapeFactor2	ShapeFactor3	ShapeFactor4	Class
0.001978	0.664361	0.997363	DERMASON
0.000969	0.513488	0.979106	CALI
0.001847	0.618988	0.995469	DERMASON
0.002997	0.729557	0.999196	DERMASON
0.001689	0.642576	0.995809	SIRA

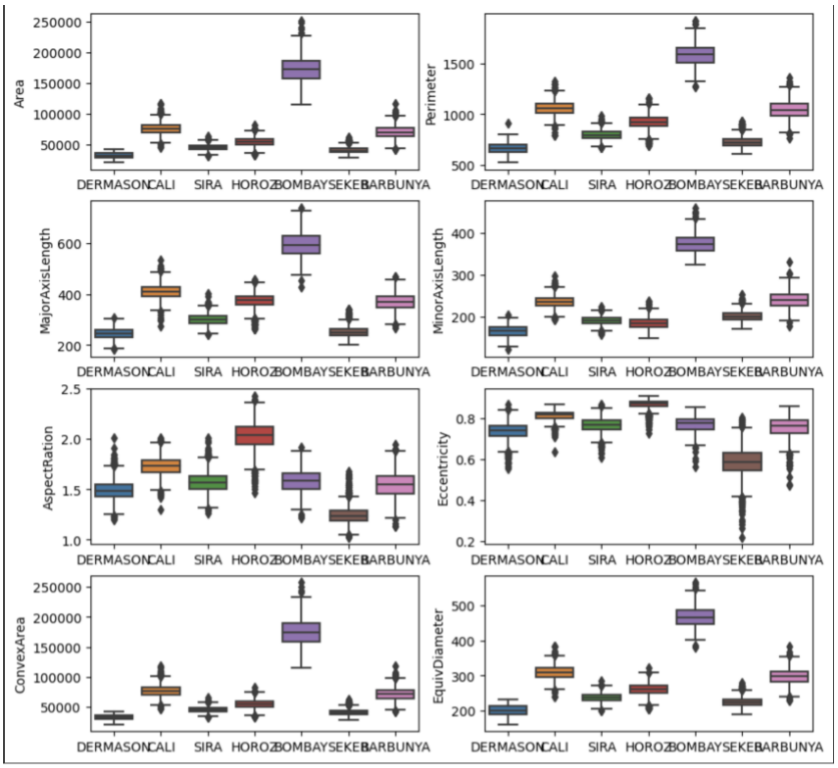
Univariate Analysis: -

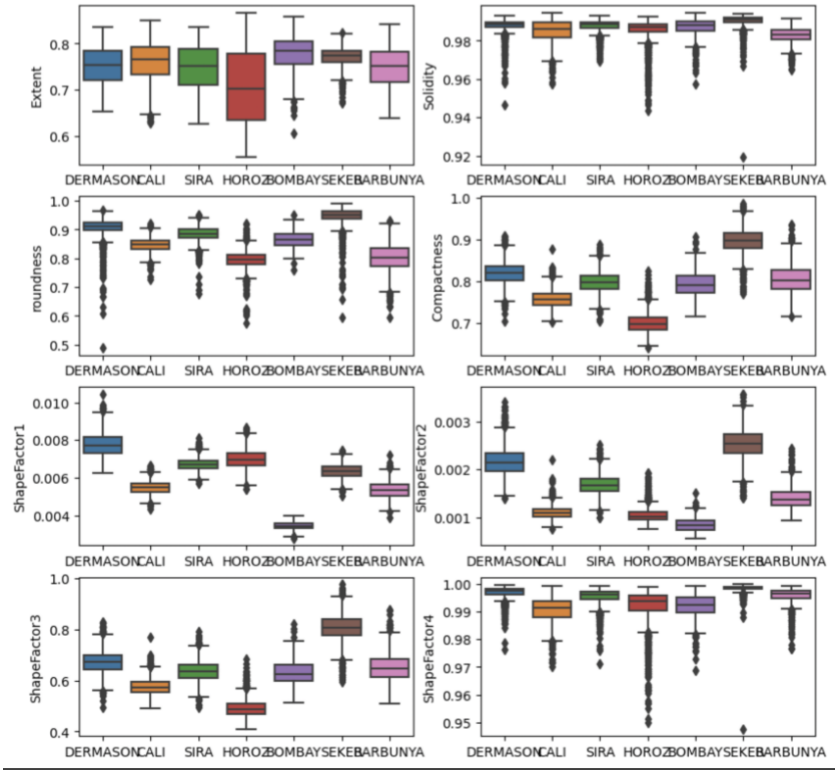
Univariate analysis is important in machine learning because it provides a detailed understanding of individual variables in a dataset, allowing for the identification of trends, patterns, and anomalies. This information is crucial for selecting appropriate modeling techniques and developing accurate predictive models.





These boxplots provide important insights into the distribution of each feature, allowing us to identify any potential outliers, skewness or significant differences in the distributions between the two groups, which can help guide data cleaning, feature selection and machine learning modeling decisions.


Area
Perimeter
MajorAxisLength
MinorAxisLength
AspectRation
Eccentricity
ConvexArea
EquivDiameter
DERMASON
CALI
SIRA
HOROZ
BOMBAY
SEKER
BARBUNYA


Extent
Solidity
roundness
Compactness
ShapeFactor1
ShapeFactor2
ShapeFactor3
ShapeFactor4
DERMASON
CALI
SIRA
HOROZ
BOMBAY
SEKER
BARBUNYA

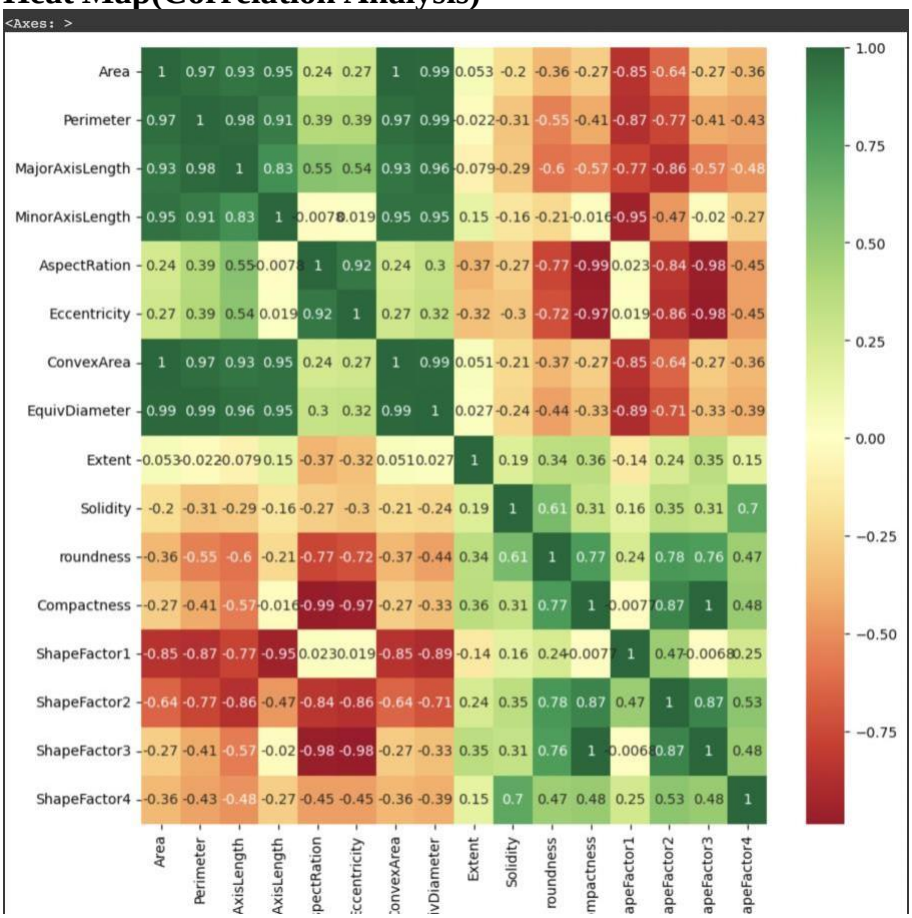
Insights from Univariate Analysis: -

- Most of the features are left or right skewed and have a lot of outliers (long tail in eccentricity, solidity, roundness, shape factor2, shape factor4)
- W.r.t area related features (Area, perimeter, convex area, equidistance, major axis), we can differentiate the 'Bombay' class
- Both Barbunya class and Cali class have similar distributions and values in many features (area, minor axis length, equivalent diameter, extent, shape factor1), which may lead to mislabeling one as the other.
- Dermason class is similar to Seker class in some features, and Sira class in other features. It may be a difficult class to label accurately!

Multivariate Analysis

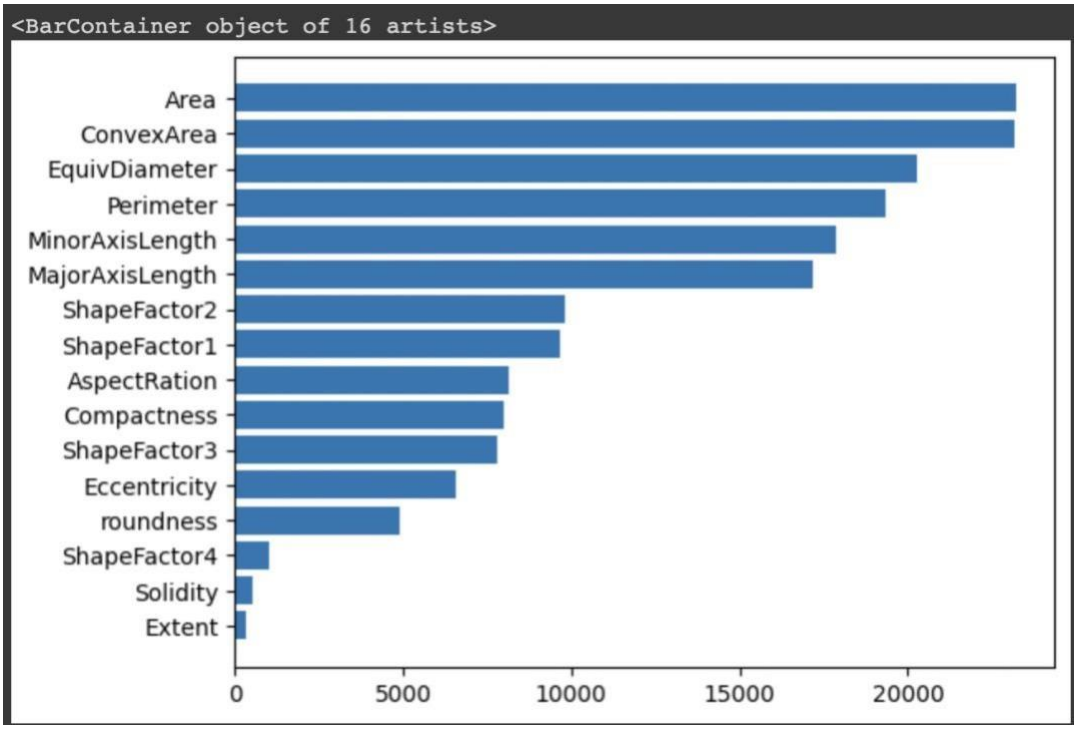
Multivariate analysis is important in machine learning because it allows us to analyze and model the relationships between multiple variables simultaneously. By using multivariate analysis, we identify the important variables and understand how they are related to each other. This understanding helps us to develop more accurate and effective machine learning models.

Heat Map(Correlation Analysis)

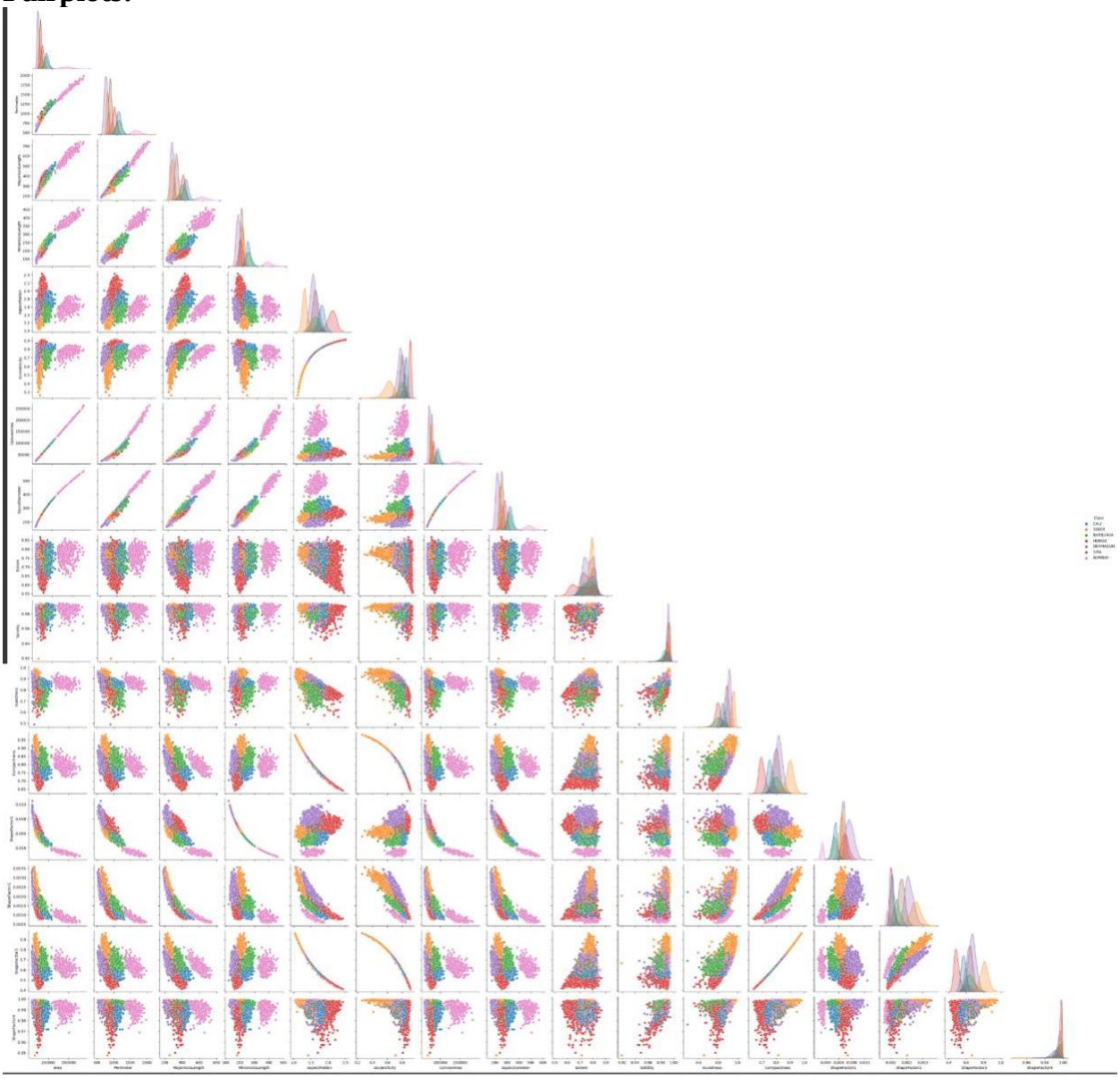


Anova/F Test to see the dependence of features with the target variable

F-test estimates the degree of linear dependency between feature and target variable.



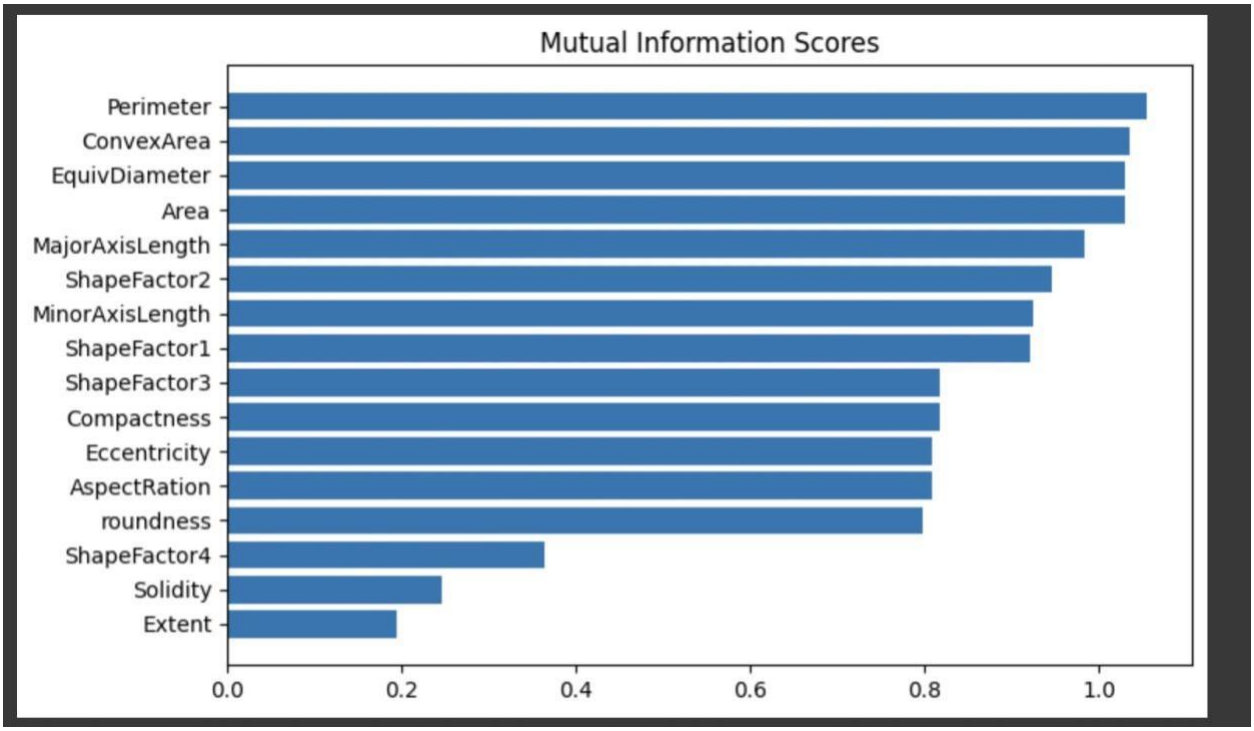
Pairplots:-



Mutual Information Scores

Mutual information between two random variables is a non-negative value, which measures the dependency between the variables.

It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.



Dropping columns with high correlation

After performing, multivariate Analysis, we drop the features with high correlation.In this case

```
columns we are dropping since there is >= .95 correlation :
['Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'ConvexArea', 'EquivDiameter', 'ShapeFactor3']

columns we are dropping since there is =< -0.95 correlation :
['Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'ConvexArea', 'EquivDiameter', 'ShapeFactor3']
```

Insights from Multivariate Analysis:

After plotting the correlation plots, we find the following pairs have the highest correlation with each other:

1. Area & convex area: 1.00
2. Compactness & shape factor 3: 1.00
3. Equivalent diameter & perimeter: 0.99
4. Equivalent diameter & convex area: 0.99
5. Major axis length & perimeter: 0.98
6. Area & perimeter: 0.97

7. Convex area & perimeter: 0.97
8. Major axis length & equivalent diameter: 0.96
9. Minor axis length & equivalent diameter: 0.95
10. Minor axis length & convex area: 0.95
11. Minor axis length & shape factor 1: -0.95
12. Eccentricity & compactness: -0.97
13. eccentricity & shape factor 3: -0.98
14. aspect ratio & shape factor 3: -0.98
15. aspect ratio & compactness: -0.99

- From Mutual Information & F-test, we can see that following variables have least dependency with respect to the response variable:

1. ShapeFactor4
2. Solidity
3. Extent

Hence, we can conclude that, the above highly correlated features are not useful for our analysis and can be dropped. Similarly, features that offer very low information to the dependent variable can also be dropped.
So, we are dropping

1. Perimeter
2. Major Axis Length
3. Minor Axis Length
4. Convex Area
5. Equivalent Diameter
6. ShapeFactor3
7. Compactness
8. Extent
9. Solidity
10. ShapeFactor4

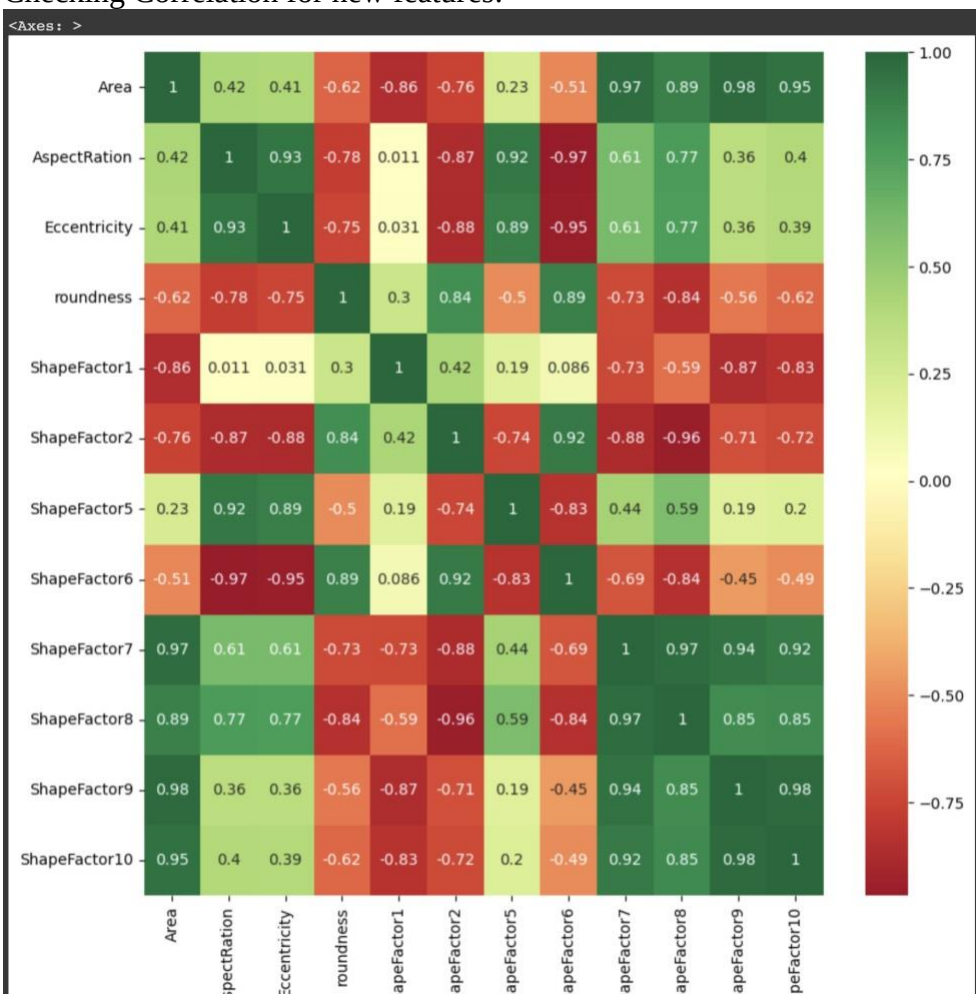
FEATURE ENGINEERING: -

We created 6 new features from the existing features, in order to reduce variance.
They are

1. ShapeFactor5
2. ShapeFactor6
3. ShapeFactor7
4. ShapeFactor8
5. ShapeFactor9
6. ShapeFactor10

We calculate them using the existing features:-
X_new[‘ShapeFactor5’] = X[‘MajorAxisLength’] / X[‘Perimeter’]
X_new[‘ShapeFactor6’] = X[‘MinorAxisLength’] / X[‘Perimeter’]
X_new[‘ShapeFactor7’] = X[‘Eccentricity’] * X[‘Area’]
X_new[‘ShapeFactor8’] = X[‘Eccentricity’] * X[‘Perimeter’]
X_new[‘ShapeFactor9’] = X[‘Extent’] * X[‘Area’]
X_new[‘ShapeFactor10’] = X[‘Extent’] * X[‘Perimeter’]

Checking Correlation for new features: -



After adding new features, we found out there is a huge correlation between the new features, hence we decided to go eliminate the new ones and proceed with the existing features.

For the outliers to not affect the model performance, we eliminated the outliers found.

Dimensionality Reduction: -

Principal Component Analysis

Principal Component Analysis (PCA) is a method that identifies the components with the highest variance in a dataset. It achieves this by projecting values onto the eigenvectors of the corresponding covariance matrices.

The process of finding these eigenvectors involves decomposing the matrix into its eigenvalues and eigenvectors. If the data set is not a square matrix, it can be decomposed using singular value decomposition.

To calculate the maximum variance captured by the first n components in singular value decomposition, we use a formula that involves the sum of the first n eigenvalues and the sum of all eigenvalues. To perform PCA, we first center the values around their mean. Then, we find the covariance matrix, eigenvalues, and eigenvectors of the matrix. After sorting the eigenvectors, we choose the top n eigenvectors and project the data onto them.

The first n components from the singular values decomposition, when calculated the maximum variance captured from the formula:

$$\frac{\sum_{i=1}^{n} s_i}{\sum_{i=1}^{m} s_i}$$

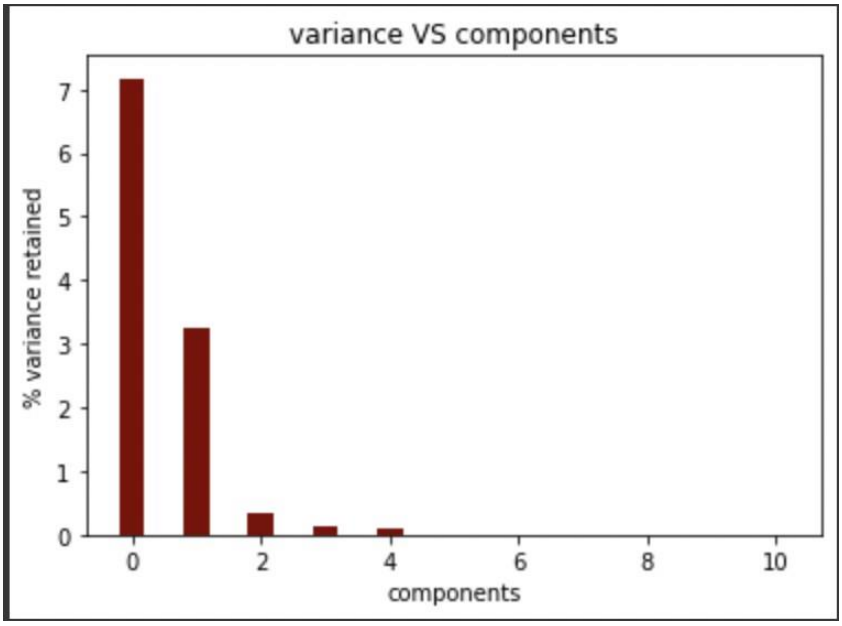
n = first n eigenvalues

M = sum of all eigenvalues

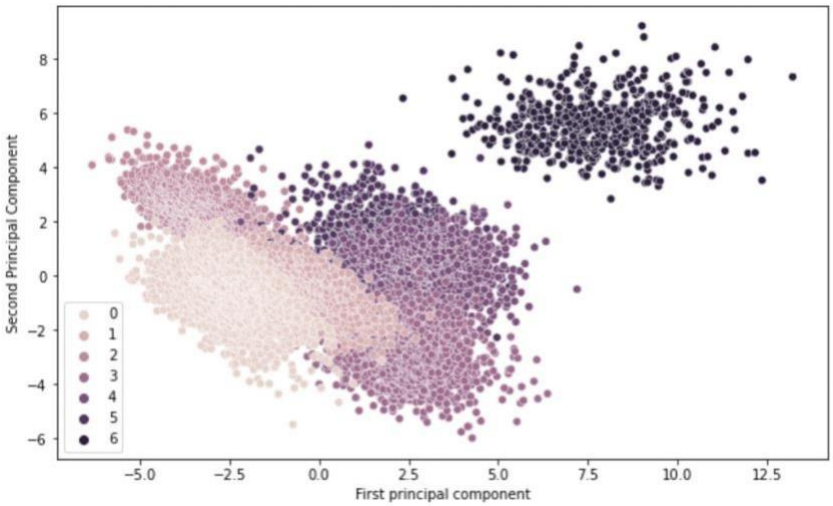
These are the steps to perform PCA: -

1. Center the values in the dataset around their mean.
2. Calculate the covariance matrix of the centered dataset.
3. Find the eigenvalues and eigenvectors of the covariance matrix.
4. Sort the eigenvectors in descending order of their corresponding eigenvalues.
5. Choose the top n eigenvectors based on the desired number of principal components to retain.
6. Project the original data onto the selected eigenvectors to transform it into a new k-dimensional feature space.

The plot illustrates the relationship between the percentage of variance captured and the number of components in a dataset. It shows that the first 2-3 components contain a large proportion of the variance, meaning that they are the most significant features of the dataset. The data presented in the plot was selected using univariate feature selection methods, which involve analyzing each feature independently and selecting the ones with the highest predictive power.



Based on this finding, it was decided to move forward with univariate featureselection methods. A plot of the first principal component (PC1) versus the second principal component (PC2) is shown in the charts below.



MODEL SELECTION: -

Model Candidates:

The response (dependent) variable we are trying to predict through the dataset is a categorical variable, hence we will be using classification techniques to predict the class. We have taken the following machine learning classification models to perform on our dataset:

1. **Multi Nominal Logistic Regression:** This is a simple and widely used classification algorithm that is effective for classification problems involving more than two classes

2. **Support Vector Machines (SVM):** This algorithm uses a hyperplane to separate data into different classes and tries to maximize the margin between the hyperplane and the closest data points.

3. **Neural Networks:** This is a deep learning algorithm that uses layers of interconnected nodes to learn complex representations of data and make predictions

We will be using the above-mentioned models to perform the classification on the train data and using the matrices such as precision and recall on the train and validation data, we will be selectin the best candidate to predict the outcome for our test data.

MultiNominal Logistic Regression:

Multinomial logistic regression is a logistic regression extension that enables multi-class classification using SoftMax activation. It is used to predict probabilities of belonging to different classes when there are more than two classes involved.

Steps Involved:

1) In cases where the dependent variable in logistic regression involves more than two classes, a commonly used technique is to apply the SoftMax function. This approach was also utilized in our particular scenario where we had to predict among seven different classes.

```
def softmax(self,z):
    z = z - np.max(z)
    z = np.exp(z)
    return np.divide(z , z.sum(axis=0))
```

2) The categorical cross-entropy loss function is a way to measure the difference between the predicted probabilities and the true probabilities of the classes in a classification problem. It is commonly used in deep learning for multi-class classification tasks.

In order to calculate the categorical cross-entropy loss for a single example, the function computes the sum of the negative logarithm of the predicted probability of the true class. This can be expressed as:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

```
def costFunction(self,X,y):
    y_hat = self.softmax(X.dot(self.w)) # y_hat = predicted labels
    cost  = - np.sum(np.log(y_hat) * y, axis=1) # - sum(  y * log ( y_hat )  )
    if self.reg:
      cost  = cost + (self.reg_param * ( np.sum(np.square(self.w)) ) )
    return 0.5* np.mean(cost)
```

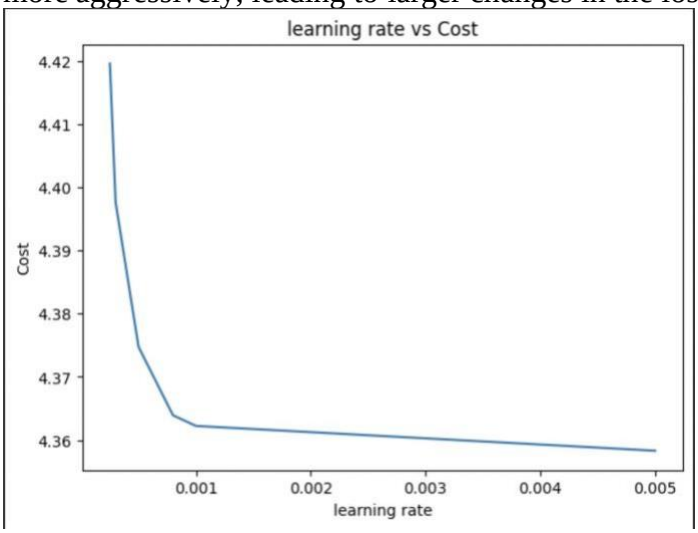
3)In multi-class classification, the labels are typically one-hot encoded, which means that each class is represented by a binary vector where only one element is 1 (positive class) and the rest are 0s (negative classes).

When using the categorical cross-entropy loss function for multi-class classification with one-hot encoded labels, only the term corresponding to the positive class Cp contributes to the loss. This is because all other terms in the sum of the loss function are multiplied by 0 (since the corresponding elements in the one-hot encoded target vector are 0s).

Mathematically, we can express this as follows:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

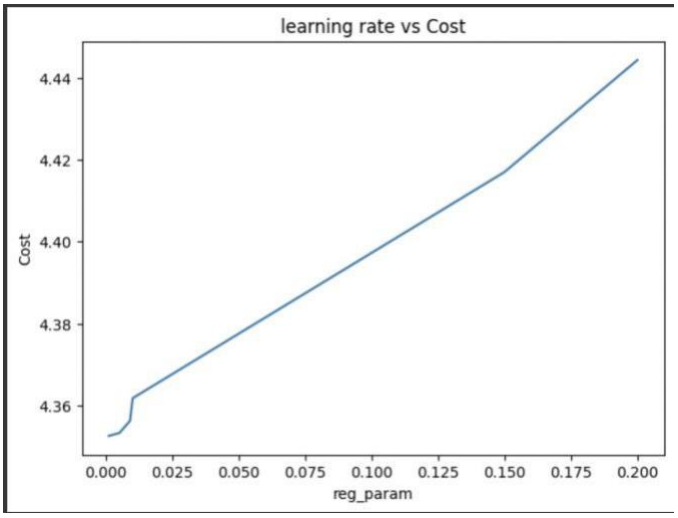
4) The plot displays the relationship between the learning rate and the average loss across the iterations. It shows that as the learning rate increases, the average loss decreases. The learning rate is a hyperparameter that determines the step size in updating the model parameters during the training process. A higher learning rate means that the model parameters will be updated more aggressively, leading to larger changes in the loss function.



5) The plot shows the relationship between the regularization parameter value (L2 penalization) and the cost. The L2 penalization is a regularization technique used to prevent overfitting by adding a penalty term to the loss function that discourages the model from relying too heavily on any one feature. The penalty term is proportional to the square of the weights of the model, which means that as the regularization parameter value increases, the weight vector is forced to become smaller, resulting in a simpler and more generalizable model.

The plot indicates that as the value of the regularization parameter increases, the cost also increases. This is because the greater the penalty term, the more the model is discouraged from relying on any one feature, which makes the model more biased and less likely to overfit the training data.

On the other hand, if the regularization parameter value is very low, the cost is also very low, which indicates that the model is overfitting the data. In this case, the model is relying too heavily on the training data and is not generalizing well to new, unseen data.



SUPPORT VECTOR MACHINES

A support vector machine (SVM) is a popular supervised learning algorithm used for classification and regression analysis. SVMs are used to sort data into one of two categories by constructing a decision boundary that maximizes the margins between the two categories. In the case of Soft Margin SVM, the SVM algorithm allows for some data points to be misclassified in order to achieve a wider margin.

In the case of multi-class classification, the one-vs-all (OVA) approach is often used with SVMs. The OVA approach involves training multiple binary SVM classifiers, each one trained to distinguish one class from all the others. During prediction, the class with the highest score from any of the binary classifiers is selected as the predicted class.

Overall, SVMs are powerful and versatile machine learning algorithms that can be used for a variety of tasks, such as image classification, natural language processing, and fraud detection. The choice of SVM variant and approach depends on the specific problem and dataset at hand, as well as the desired balance between accuracy and complexity.

Steps Involved: -

1) The soft margin support vector machine (SVM) allows some data points to be misclassified using a hyperparameter C. If the value of C is large, the SVM becomes a hard margin classifier, which aims to correctly classify all data points. On the other hand, if C is small, the SVM becomes a soft margin classifier, allowing for some misclassification.

To keep alpha values in a specific range, we can use complementary slackness, which is the fourth rule of the Karush-Kuhn-Tucker (KKT) conditions for SVM optimization.
By satisfying these conditions, we can ensure that alpha values stay within a specific range, and the SVM can be used effectively for classification tasks.

$$\alpha_i (1 - y_i f(x_i) - \varepsilon) = 0$$

$$(\frac{c}{n} - \alpha_i)\varepsilon = 0$$

- If $y_i f(x_i) > 1$ then the margin loss is $\varepsilon = 0$ and we get $\alpha_i = 0$
- If $y_i f(x_i) < 1$ then the margin loss is $\varepsilon > 0$ so $\alpha_i = \frac{c}{n}$
- If $\alpha_i = 0$ then $\varepsilon = 0$ which implies no loss, so $y_i f(x_i) >= 1$
- If $\alpha_i \epsilon (0, \frac{c}{n})$ then $\varepsilon = 0$ which implies $1 - y_i f(x_i) = 0$

2) To handle nonlinearity in the data, we utilized the Gaussian radial basis function as a kernel. This function measures the Gaussian distance between data points in a nonlinear space, which can be interpreted as a measure of similarity between data points.

$$K(X, Y) = exp(-\frac{||x-y||2}{2\sigma^2})$$

```
def GRBF(x1, x2):
    diff = x1 - x2
    return np.exp(-np.dot(diff, diff) * len(x1) / 2)
```

3) To handle the scenario of multiple classes, we implemented the One-vs-All approach, where each class is taken as the positive class, labeled as '1', and all other classes are labeled as '-1'.

4) Due to longer execution time of support vector machines, we have implemented One-vs-All approach by sampling the data for each class one at a time. The results of this approach are as follows:

```
one_vs_all(X_train_UV_pca,y_train_UV,X_test_UV_pca, y_test_UV)
96 corrcted out of 100:
```

NEURAL NETWORKS: -

We explored different strategies for optimizing Neural Networks and prioritizing speed. To achieve this, we have decided to use mini batches of size 64 to simplify matrix multiplication. We have provided the training and testing times in table nn1.1. Additionally, we are using different optimization algorithms such as Adam and RMSprop to converge faster through gradient descent. RMSprop uses a combination of gradient descent with momentum and exponentially weighted averages to adjust the learning rate.

Momentum is a technique used in gradient descent optimization to speed up convergence. It helps the gradient descent algorithm to move faster in the relevant direction and slow down in irrelevant directions. The momentum term is calculated by taking a weighted average of the previous gradients.

$$V_{dw} = \beta * V_{dw} + (1 - \beta) d_w$$

$$V_{db} = \beta * V_{db} + (1 - \beta) d_b$$

$$W = W - \alpha * V_{dw}$$

$$b = b - \alpha * V_{db}$$

2) In essence, this approach helps to make the update steps more consistent by using RMSprop to implement the same smoothing process.

$$S_{dw} = \beta * S_{dw} + (1 - \beta) d_w^2$$

$$S_{db} = \beta * S_{db} + (1 - \beta) d_b^2$$

$$W = W - \alpha * \frac{d_w}{\sqrt{S_{dw}}}$$

$$b = b - \alpha * \frac{d_b}{\sqrt{S_{db}}}$$

3) According to the original research recommendation, we set the value of β to 0.99 and α as the learning rate as usual. To make the update slower in the vertical direction, we divide the update of b by a relatively larger number compared to W because the slope is larger in the b direction.

Sometimes, we also add a small value of ε in the denominator under the square root to ensure numerical stability in case the denominator is 0.

4. Adam Optimization (Adaptive moment estimation): Adam performs both momentum and RMSprop and combines them as follows.

$$V_{dw} = \beta_1 * V_{dw} + (1 - \beta_1) d_w$$

$$V_{db} = \beta_1 * V_{db} + (1 - \beta_1) d_b$$

$$S_{dw} = \beta_2 * S_{dw} + (1 - \beta_2) d_w^2$$

$$S_{dw} = \beta_2 * S_{dw} + (1 - \beta_2) d_w^2$$

In a standard method such like Adam, a bias correction is applied prior to performing a gradient update.

$$V_{dw}^{corrected} = \frac{V_{dw}}{1 - \beta_1^t}$$

$$V_{db}^{corrected} = \frac{V_{db}}{1 - \beta_1^t}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{1 - \beta_2^t}$$

$$S_{db}^{corrected} = \frac{S_{db}}{1 - \beta_2^t}$$

$$W = W - \alpha * \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$$

$$b = b - \alpha * \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

This algorithm, which combines gradient descent with momentum and gradient descent with RMSprop, is a commonly used approach for faster convergence in many types of neural networks. There are several hyperparameters in this algorithm. The most commonly used parameter values include:

- α = needs to be tuned
- β (Moving averages for dw, computes past 10 values averaged) 1 = 0. 9
- β (Moving averages of dw2) 2 = 0. 999

- ε = 10 (Usually not required to use)

To accelerate the learning process of algorithms, one effective approach is to implement learning rate decay. This involves gradually decreasing the learning rate over time as the number of mini-batches and iterations increase. The reasoning behind this method is that at the start of the learning process, taking larger steps can be beneficial, but as more iterations are completed, it becomes more advantageous to slow down the learning process to approach the optimal solution more accurately.

Here, we have implemented neural networks from scratch using NumPy and optimized various parameters to achieve the maximum accuracy on the test results. In addition to the above-mentioned hyperparameters and methods, we also utilized techniques to avoid getting stuck at the saddle point, which is one of the local optima during the learning process. To prevent this, we used Adam optimizer, which helps to slip off the saddle point during learning. We also fine-tuned the decay rate hyperparameter to optimize the learning rate decay, which gradually reduces the learning rate as the number of mini-batches and iterations increases. Overall, our approach helped to speed up the learning process and avoid getting stuck at local optima.

RESULTS

1) Logistic Regression Results

- train error: 4.257356647636495
- accuracy: 0.7994613124387855
- test error: 3.8359508144175183

2) SVM Results

- 96 corrected out of 100:
- Accuracy of SVM on 100 samples = 96%

3) Neural Network Model Comparison

Accuracy on test data for different methods and Train Time

	Accuracy using PCA and Univariate FS	Train Time
Adam_using_weight_decay	86.75%	0:00:06.429996
RMSprop_using_L2	86.83%	0:00:07.193343
Base_model_without_optim_and_minibatches	58.5%	0:09:26.794456

The notation "Adam_using_weight_decay" represents the use of mini-batch gradient descent with Adam optimization. On the other hand, "RMSprop_using_L2" uses RMSprop as the optimizer and L2 regularization. However, all the Adam methods mentioned in the context use the Weight Decay technique, not L2 regularization. The key difference between L2 regularization and weight decay is that L2 regularization changes the gradients to include lambda

times the weight parameters, whereas weight decay doesn't modify the gradients. Instead, it subtracts the product of learning rate, lambda, and the weight parameters from the weights in the update step.

Conclusion:

- From the results above, we can conclude that SVM and Neural Networks give us the best resultsin this classifying task of dry beans into seven different classes.
- For Neural Networks(using RMSprop & Adam), the time taken is least whilst also providing great results.
- When it comes to SVM, it provided us with the best accuracy i.e., of 96%