T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

CSE 555 - STATISTICAL DATA ANALYSIS

FINAL PROJECT

EMIRKAN BURAK YILMAZ

SUPERVISOR
HABIL KALKAN

GEBZE
2025

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE 555 - STATISTICAL DATA ANALYSIS
FINAL PROJECT

EMIRKAN BURAK YILMAZ

SUPERVISOR
HABIL KALKAN

2025
GEBZE

ABSTRACT

This study provides an in-depth analysis of the Dry Bean dataset, applying a combination of statistical methods, machine learning, and dimensionality reduction techniques to investigate feature discriminability, clustering behavior, and the impact of outliers. The analysis begins with exploratory data examination, identifying key geometric features such as EquivDiameter, Perimeter, and MinorAxisLength as the most discriminative, based on Fisher Score assessments. Dimensionality reduction comparisons show that Principal Component Analysis (PCA) outperforms Linear Discriminant Analysis (LDA) in separating classes (silhouette scores of 0.3347 vs. 0.2105), indicating better preservation of non-linear patterns. Four clustering methods (K-Means, DBSCAN, t-SNE, and Self-Organizing Maps) are evaluated, with t-SNE yielding the most distinct visual cluster separation. Additionally, applying Isolation Forest for outlier removal improves both feature discriminability (e.g., EquivDiameter Fisher Score rising from 3.2778 to 3.6325) and clustering performance (silhouette score increasing from 0.2565 to 0.2781). Overall, the results emphasize the critical role of geometric features in classifying bean types, the strengths of unsupervised learning approaches for this dataset, and the benefits of outlier detection during preprocessing.

Keywords: Machine Learning, Dry Bean Dataset, Feature Selection, Dimensionality Reduction, Clustering Analysis, Outlier Detection.

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
DBSCAN	:	Density-based Spatial Clustering of Applications with Noise
LDA	:	Linear Discriminant Analysis
PCA	:	Principal Component Analysis
SOM	:	Self Organizing Map
t-SNE	:	t-distributed Stochastic Neighbor Embedding

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This project explores the application of statistical and machine learning techniques to the Dry Bean dataset, which contains samples from seven distinct types of beans, each described by sixteen measurable features.

The main goal of this work is to investigate and analyze the dataset through a structured pipeline that includes data normalization, dimensionality reduction, clustering, and outlier detection. Techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and various clustering algorithms (including K-Means, DBSCAN, t-SNE, and Self-Organizing Maps) are applied to assess how effectively they can cluster the Dry Bean dataset.

The report is organized as follows: Chapter 2 presents the methods and results, structured step-by-step according to the project tasks. Chapter 3 summarizes the key findings and conclusions. All code used throughout the experiments is provided in the appendices.

2. ANALYSIS AND RESULTS

This chapter presents the statistical analysis steps applied to the Dry Bean dataset. Each task specified in the project instructions is addressed in a separate section, with corresponding visualizations and interpretations.

2.1. Boxplot Analysis of Attributes

The distribution of each feature was visualized using boxplots, as illustrated in Figure 2.1. This analysis allowed us to identify skewness, variability, and potential outliers.

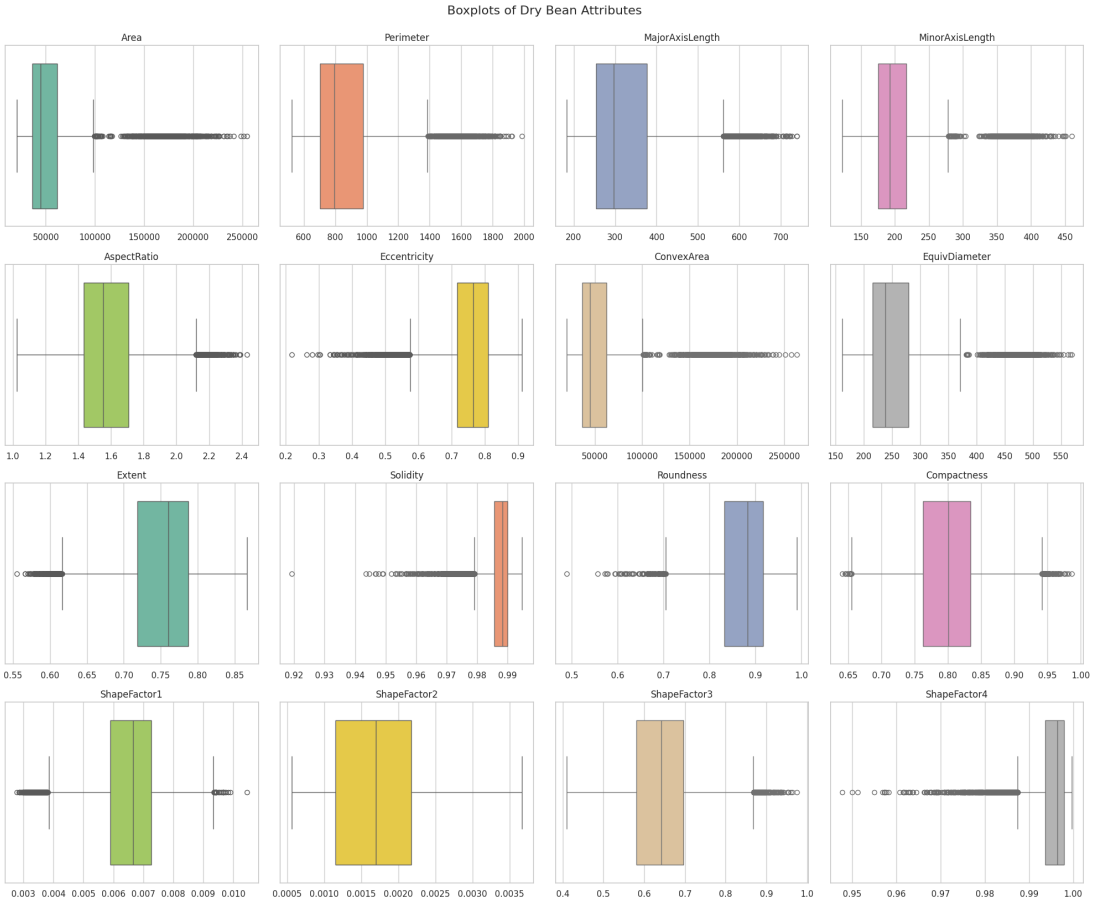

Figure 2.1: Boxplot of the attributes

At first glance, all features except ShapeFactor2 contain numerous outliers. These

outliers are mostly located at the upper end of the range, particularly in area-related features such as AspectRatio, ConvexArea, and EquivDiameter, while for features like Eccentricity, Extent, Solidity, and Roundness, the outliers are more pronounced at the lower end.

All features exhibit a narrow interquartile range (IQR), indicating that the majority of the data points are densely clustered, with relatively few extreme values.

Regarding the distribution patterns, many features display approximately normal distributions but with varying means and variances. Some features show noticeable skewness either to the left or right, suggesting asymmetric data spread and the presence of tails on one side of the distribution.

These insights provide an initial understanding of the dataset's variability, symmetry, and potential anomalies, which are important when selecting and applying further preprocessing and modeling techniques.

2.2. Z-Score Normalization and Fisher Distance

To begin, the data were normalized using Z-score normalization, ensuring that each feature has a mean of zero and a variance of one. This standardization step is essential because the Fisher Distance metric is sensitive to the scale of the input attributes. Without such normalization, features with larger scales could disproportionately influence the distance calculations.

Given that the dataset comprises seven distinct classes, the Fisher Distance was adapted to handle this multi-class setting by employing the generalized Fisher Distance approach proposed by Li et al. (2017) [1]. While the classical Fisher Distance is designed for two-class problem, the generalized method extends this framework to multi-class problems by computing the average of all pairwise Fisher Distances across class pairs. This approach allows for a comprehensive assessment of class separability in multi-class datasets. The generalized Fisher Distance FD_j for feature is computed as:

$$FD_j = \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{k=i+1}^{C} \frac{|\mu_{ij} - \mu_{kj}|}{\sqrt{\sigma_{ij}^2 + \sigma_{kj}^2}}$$

where:

- C: number of classes
- μ_{ij}: mean of feature j in class i

- σ_{ij}^2: variance of feature j in class i

Table 2.1: Fisher Scores of Features (Descending Order)

Feature	**Fisher Score**
EquivDiameter	3.2778
Perimeter	3.2119
MinorAxisLength	3.1344
ShapeFactor1	3.1215
MajorAxisLength	2.9482
Area	2.9035
ConvexArea	2.9027
ShapeFactor2	2.3974
Compactness	1.8473
ShapeFactor3	1.8223
AspectRatio	1.8027
Eccentricity	1.5937
Roundness	1.5048
ShapeFactor4	0.7290
Solidity	0.5141
Extent	0.3750

The results in Table 2.1 indicate that EquivDiameter, Perimeter, and MinorAxisLength have the highest Fisher Scores, suggesting they are the most discriminative features for class separation in this dataset. These geometric features likely capture significant shape or size variations across classes, which contribute to their high class separability. In contrast, features like ShapeFactor4, Solidity, and Extent exhibit very low Fisher Scores, implying they contribute minimally to class discrimination.

2.3. Principal Component Analysis (PCA)

This section presents the application of PCA to the dataset [2]. Each principal component was treated as a new feature, and its Fisher Distance was calculated to assess class separability. The results, summarized in Table 2.2, reveal the relationship between the explained variance ratio and the Fisher Score for each component.

Table 2.2: Fisher Scores and Explained Variance of PCA Components

Principal Component	**Explained Variance Ratio**	**Fisher Score**
PC1	0.5547	2.8558
PC2	0.2643	2.4563
PC3	0.0801	1.1114
PC4	0.0511	0.7795
PC5	0.0274	0.7703
PC6	0.0115	0.4922
PC7	0.0070	0.4819
PC8	0.0033	0.4768
PC9	0.0005	0.3897
PC10	0.0001	0.3297
PC11	0.0001	0.2757
PC12	0.0000	0.2565
PC13	0.0000	0.2350
PC14	0.0000	0.1734
PC15	0.0000	0.1302
PC16	0.0000	0.0754

The first principal component (PC1), which accounts for approximately 55.47% of the total variance, achieved the highest Fisher Score of 2.86, indicating that it captures both substantial variability in the data and a meaningful level of class-discriminative information. Similarly, PC2, which explains 26.43% of the variance, attained a Fisher Score of 2.46, further underscoring the discriminative relevance of the leading components.

Beyond PC2, a sharp decline is observed in both explained variance and Fisher Scores. For instance, PC3 and PC4 account for 8.01% and 5.11% of the variance, respectively, with corresponding Fisher Scores of 1.11 and 0.78. This trend continues, with later components contributing progressively less to both variance and class separation. Notably, components PC10 through PC16 contribute no variance and exhibit very low Fisher Scores suggesting negligible utility in distinguishing between classes.

The results in Table 2.2 further reveal that the Fisher Scores for PCA-transformed features are generally lower than those computed for the original features, as shown in Table 2.1. While PCA is effective for dimensionality reduction and data compression, it does not inherently optimize for class separability. As an unsupervised technique, PCA seeks to maximize variance without accounting for class labels, which can result

in components that do not capture class-specific structure effectively.

2.4. Scatter Plot of Principal Components

To further assess the effectiveness of the principal components in capturing data structure and enabling class discrimination, two scatter plots were analyzed: one displaying the projection of the dataset onto the first two principal components (PC1 and PC2) and the other onto the last two components (PC15 and PC16). These visualizations are presented in Figure 2.2.

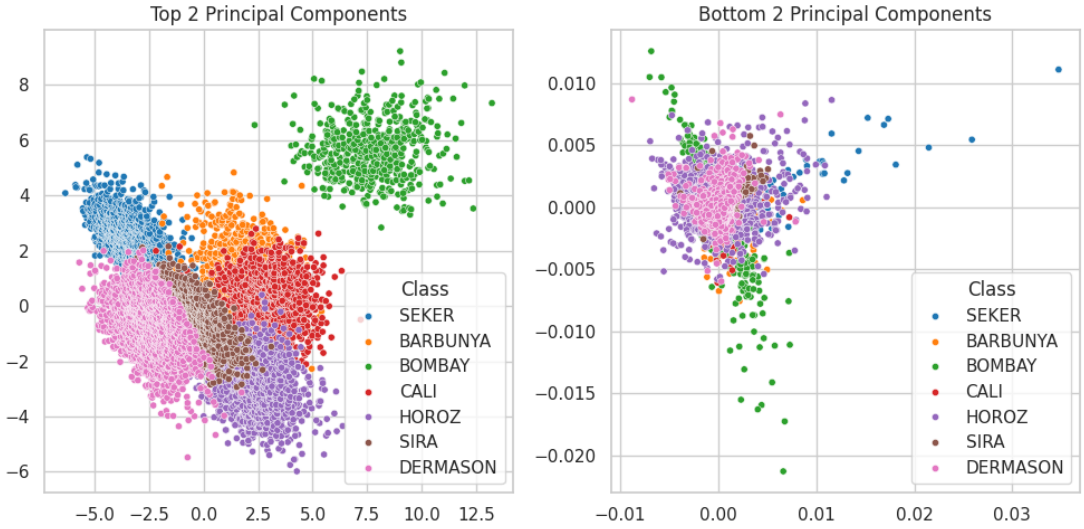

Figure 2.2: Scatter plot using top 2 and bottom 2 PCA components

The projection onto PC1 and PC2 captures approximately 80.9% of the total variance (55.47% and 26.43%, respectively), indicating that these components retain the majority of the original data's structural information. The scatter plot reveals distinct clustering and directional separation among different bean classes, suggesting a high degree of class separability in this reduced two-dimensional space.

These visual observations are consistent with the numerical results reported in Table 2.2, where PC1 and PC2 achieved the highest Fisher Scores of 2.86 and 2.46, respectively. The combination of high variance capture and strong class discrimination confirms the value of these components for both data representation and supervised learning tasks.

In contrast, the projection onto PC15 and PC16 explains virtually no variance in the dataset. Correspondingly, the Fisher Scores for these components are minimal (0.13 for

PC15 and 0.075 for PC16) indicating that these dimensions carry negligible discriminative information. The scatter plot illustrates this, displaying a highly overlapping and noisy distribution with no discernible class structure, thereby reinforcing the conclusion that these components contribute little to classification performance.

2.5. Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique designed to maximize class separability by projecting data along directions that increase between-class variance and reduce within-class variance [3]. In contrast, PCA is an unsupervised method that identifies directions of maximum overall variance without considering class labels. LDA was applied to reduce the dataset to two dimensions, and the resulting projection is visualized in Figure 2.3, alongside the PCA projection from Section 2.4 for comparison.

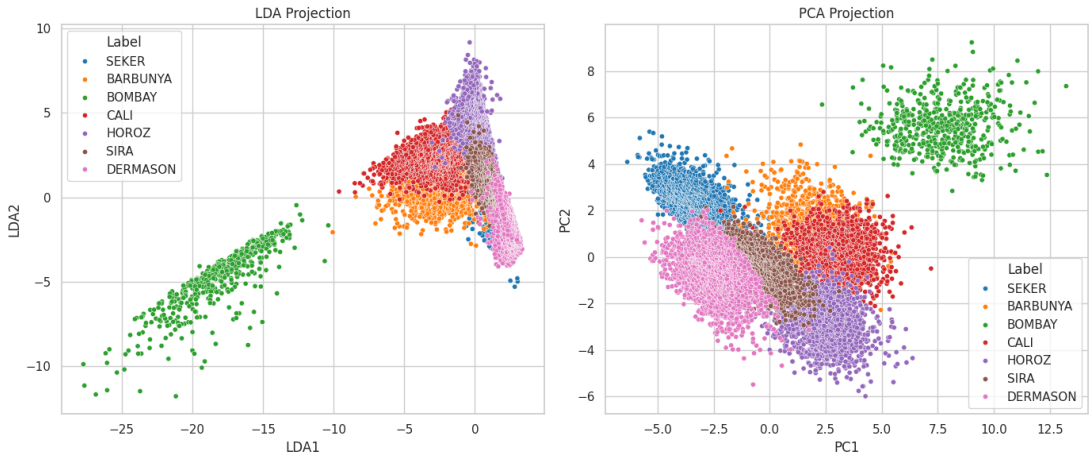

Figure 2.3: LDA vs PCA

Visually, the BOMBAY class (shown in green in the plot) is clearly distinguishable in both projections. However, the PCA projection (right plot) shows better separation among several major classes compared to the LDA projection (left plot), where multiple classes remain tightly clustered with significant overlap.

To quantify the quality of class separation, the silhouette score was used [4]. This score measures how similar each point is to its own cluster compared to other clusters, with values ranging from -1 (poor separation) to $+1$ (well-separated clusters). As shown in Table 2.3, the PCA projection achieved a higher average silhouette score (0.3347) compared to the LDA projection (0.2105), indicating better class separation

with PCA.

Table 2.3: Average Silhouette Scores for PCA and LDA Projections

Projection	**Silhouette Score**
PCA	0.3347
LDA	0.2105

The higher silhouette score for PCA (0.3347) compared to LDA (0.2105) can be explained by several factors. First, the Dry Bean dataset likely contains non-linear relationships between features. LDA, which is optimized for linear class separation, may struggle when classes are not perfectly linearly separable. In contrast, PCA is an unsupervised method that focuses on maximizing variance, capturing the overall data structure, even if it's not perfectly aligned with class boundaries.

Additionally, LDA is more sensitive to noise. If the dataset has overlapping classes or irrelevant features, this can hinder LDAs performance. In contrast, PCA is less sensitive to noise and focuses on retaining the overall variance in the data, which may contribute to better class separation in some cases.

In conclusion, while LDA is designed for class separability, PCA performed better in this dataset, likely due to the non-linear nature of the data, the dimensionality constraints of LDA, and its sensitivity to noise.

2.6. Clustering Algorithms

In this section, the Dry Bean dataset is grouped using four different clustering algorithms: K-Means, DBSCAN, t-SNE, and SOM. The goal is to explore the structure of the data through unsupervised learning and visualize the results. For the K-Means and DBSCAN clustering algorithms, the first two principal components obtained from the PCA transformation are used as the reduced feature set for clustering. On the other hand, t-SNE and SOM are applied directly to the original dataset without any dimensionality reduction.

2.6.1. K-Means

The K-Means algorithm partitions the data into k clusters by minimizing the within-cluster variance [5]. In this analysis, the number of clusters was set to $k = 7$, based

on prior knowledge of the dataset, specifically reflecting the known number of distinct classes in the Dry Bean dataset. Figure 2.4 illustrates the clustering results obtained using the K-Means algorithm with a comparison to the original classes, applied to the first two principal components extracted through PCA.

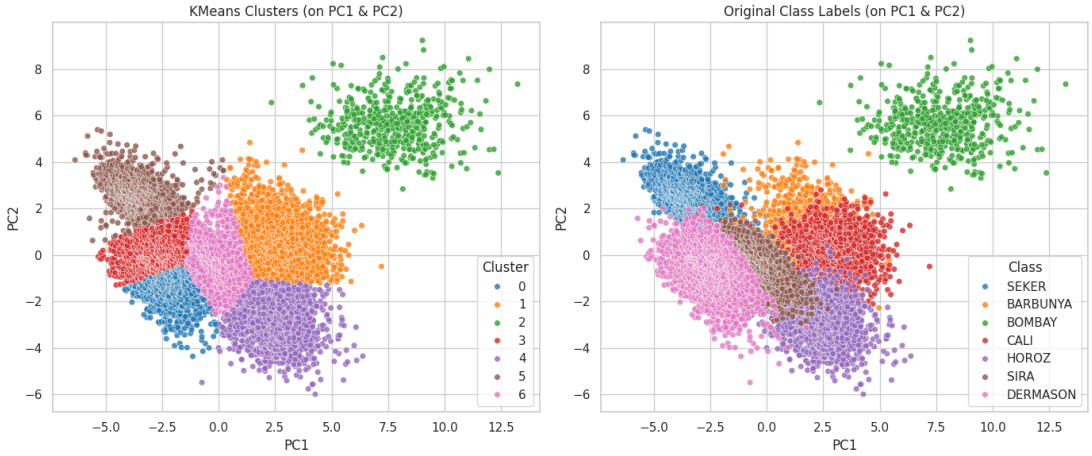

Figure 2.4: Clustering result using K-Means (PCA-reduced data)

The figure demonstrates that K-Means separates the data into distinct clusters; however, the proximity of several cluster boundaries suggests that the algorithm struggles to differentiate certain classes, particularly in our case where the clusters are not well-separated in the reduced PCA space. This observation highlights the sensitivity of the algorithm to the chosen value of k. Without prior knowledge of the actual number of classes, selecting a smaller k such as 2 or 3 may have been more appropriate, as the current configuration shows clusters positioned so closely that minor noise could easily cause a data point to shift from one cluster to another.

2.6.2. DBSCAN

Unlike K-Means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) does not require a predefined number of clusters and can detect clusters of varying shapes and densities [6]. In Figure 2.5, the results of the DBSCAN algorithm illustrates the identified two primary clusters, as indicated by the clear separation in the plot. This result suggests that DBSCAN is more flexible than K-Means, as it can identify clusters without prior knowledge of number of classes and also detect noise. DBSCAN's ability to find dense regions and mark the rest as noise shows its advantage when dealing with unevenly distributed data.

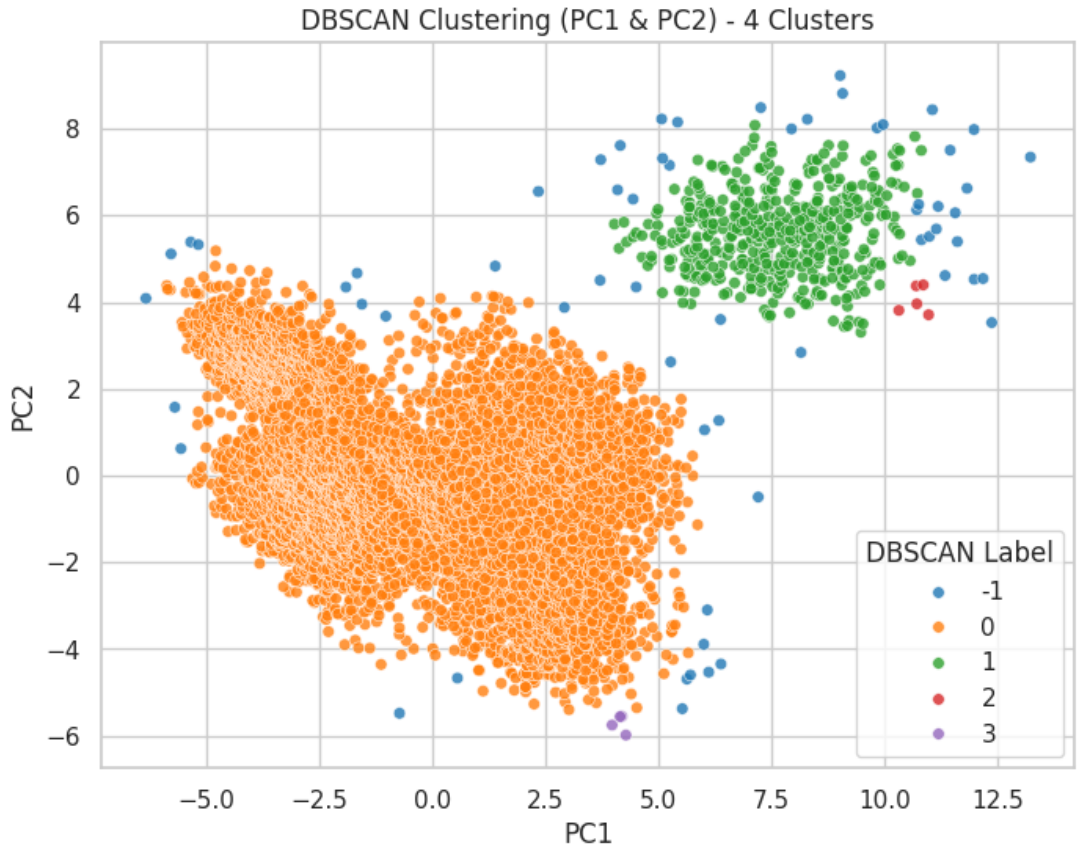

Figure 2.5: Clustering result using DBSCAN (PCA-reduced data)

2.6.3. t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique that is particularly useful for visualizing high-dimensional data in lower dimensions [7]. The algorithm works by preserving local data structures, making it easier to identify clusters within the data. In this figure, the clusters are better separated compared to the PCA-based result. Figure 2.6 shows clearer boundaries between most of the classes compared to PCA, suggesting that t-SNE is effective in uncovering the structure of the data that may not be as apparent in higher dimensions.

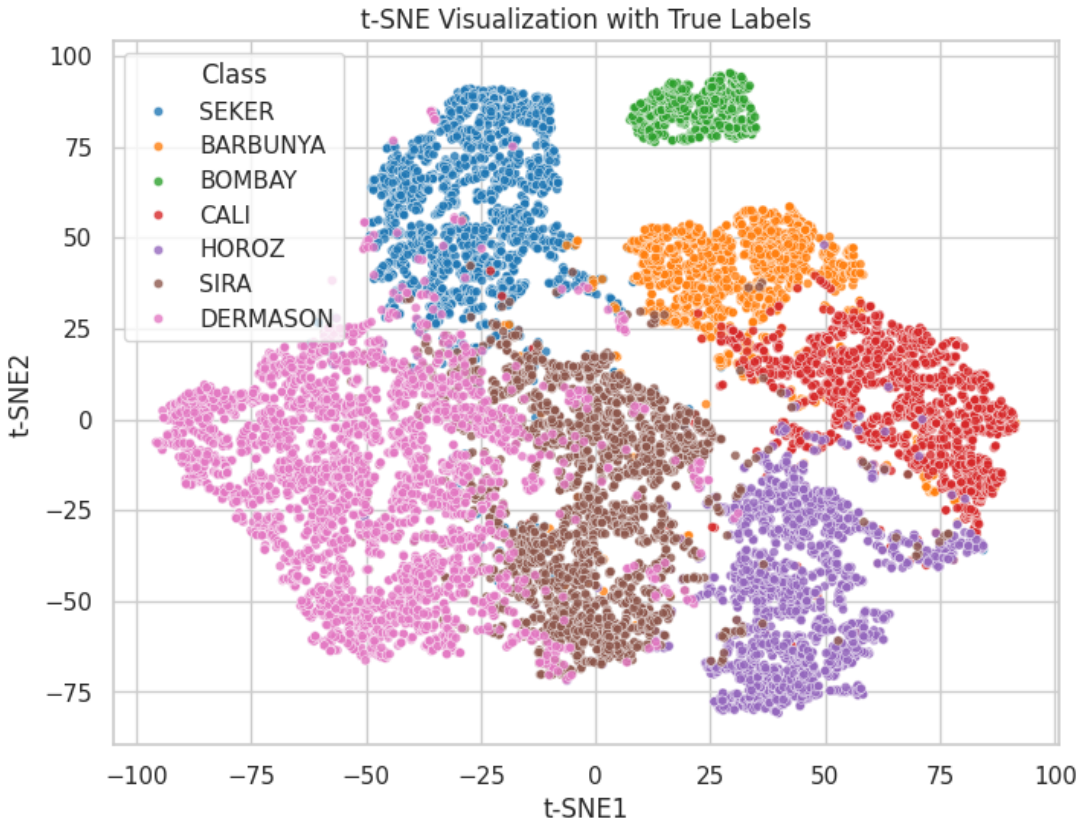

Figure 2.6: t-SNE 2D visualization

2.6.4. SOM

SOM (Self-Organizing Map) is an unsupervised artificial neural network that projects high-dimensional data onto a typically two-dimensional grid, preserving the topological relationships among data points [8]. During training, the SOM adjusts its neurons to represent the input space, effectively grouping similar instances into neighboring grid cells. In this case, a 10×10 grid was used to map the Dry Bean dataset.

Figure 2.7 reveals that similar data points are clustered around certain grid regions, with some neurons capturing the boundaries between different classes. While the SOM effectively identifies general class groupings, some grid cells contain data points from multiple classes, indicating classification or boundary errors. This is partly due to the competitive and neighborhood-based learning process of SOMs, where neighboring neurons influence each other's updates.

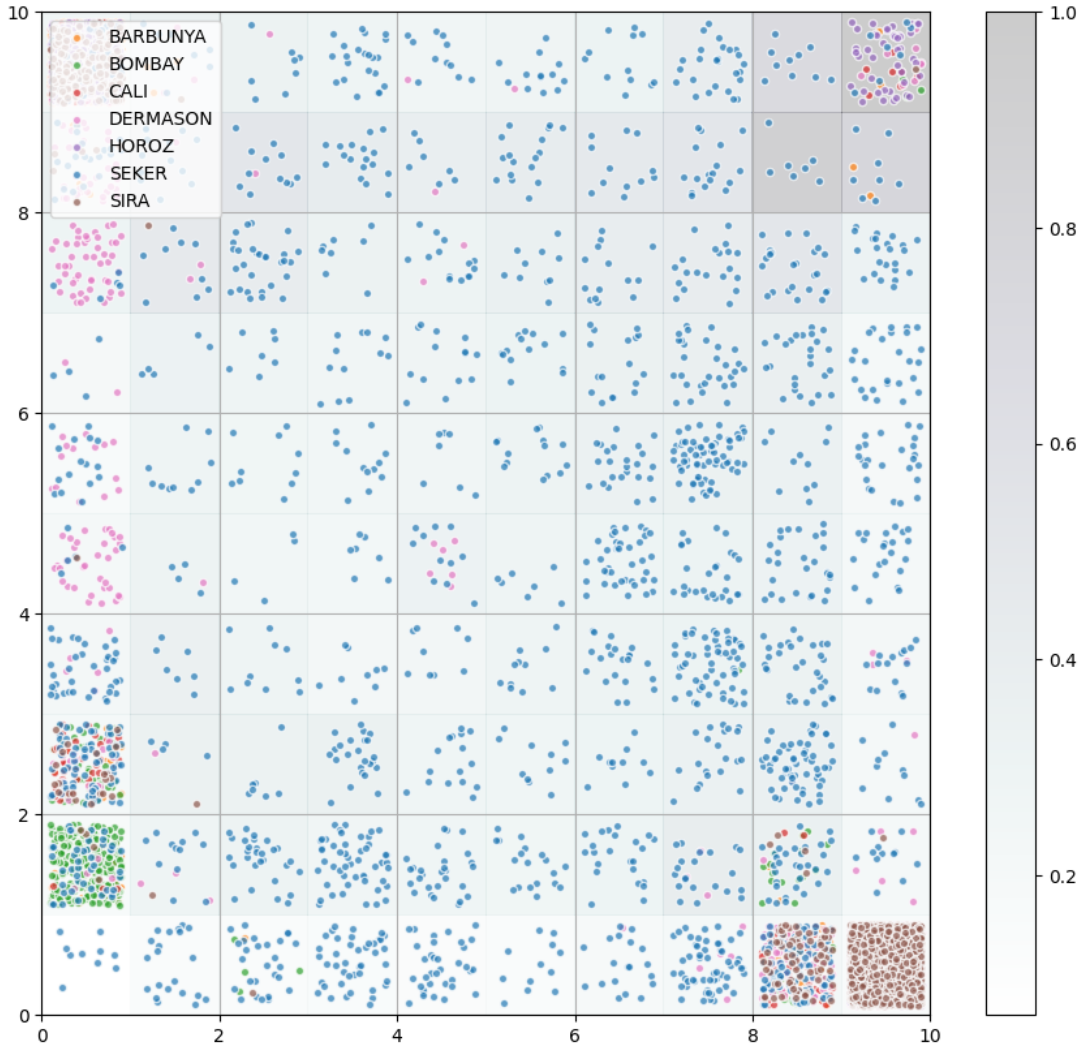

Figure 2.7: SOM clusters

The SOM algorithm is particularly valuable for high-dimensional datasets because it provides a non-linear, topology-preserving mapping, allowing to visualize complex patterns and relationships that may not be easily captured through linear techniques like PCA. However, it is important to note that the interpretability of SOM results can depend on factors such as grid size, training iterations, and neighborhood radius, which influence how well the map captures the true class structure.

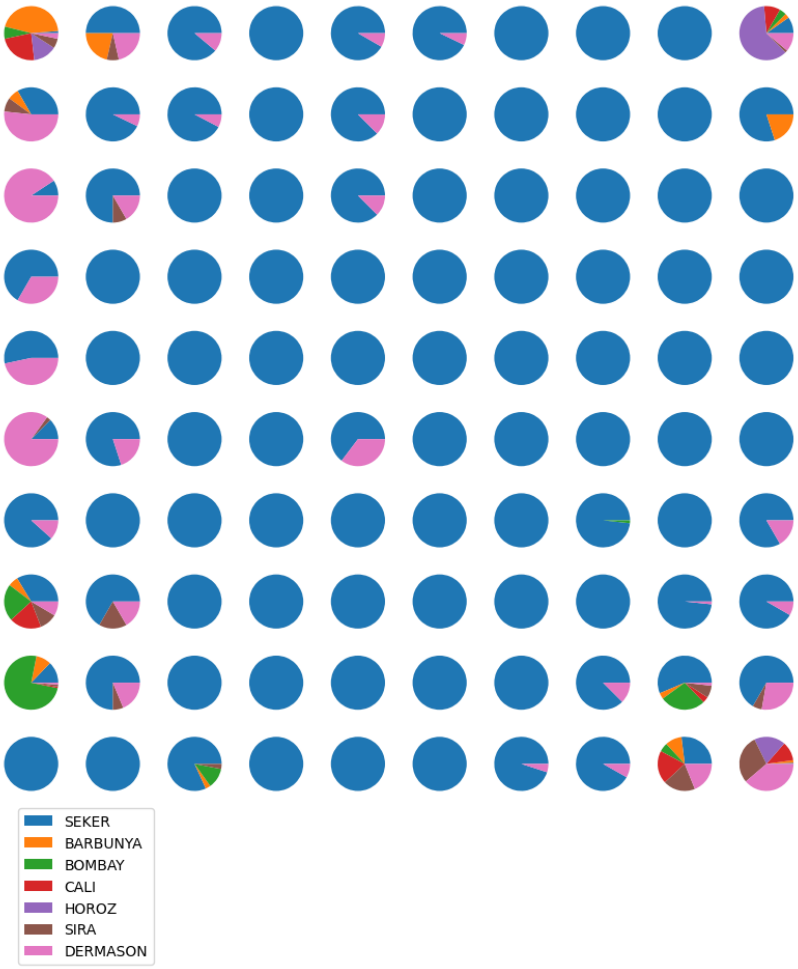

Figure 2.8: SOM neuron frequencies

Figure 2.8 presents SOM visualization that depicts the distribution of classes across the grid's neurons. In this map, each neuron is visualized using a pie-chart-like color scheme, where the proportion of colors reflects the percentage of different classes assigned to that neuron. The visualization reveals that many neurons are entirely dominated by the SEKER class (shown in blue), while several other neurons exhibit mixed-class compositions, highlighting regions where the algorithm clusters similar but not perfectly separable samples. This outcome is consistent with observations from the PCA and t-SNE analyses, where certain class boundaries were also found to be ambiguous or overlapping. The SOM's clustering behavior here reflects the inherent complexity of the dataset and the presence of confusing or borderline examples, underscoring a common challenge faced across multiple clustering techniques.

2.7. Outlier Detection

For outlier detection and noise cleaning in the Dry Bean dataset, the Isolation Forest algorithm was chosen. Isolation Forest is a tree-based ensemble method designed specifically for anomaly detection. It isolates anomalies instead of profiling normal data points, which makes it efficient for high-dimensional datasets like ours with 16 features [9]. Additionally, Isolation Forest does not assume any underlying data distribution, making it suitable for datasets with complex, non-linear feature interactions. Its ability to handle large datasets efficiently and its effectiveness in identifying anomalies with few hyperparameters make it a practical choice for our cleaning task.

The Isolation Forest algorithm was applied to the original dataset with a contamination parameter set to 0.01, meaning that approximately 1% of the data points were flagged as outliers. These detected outliers were then removed from the dataset before further analysis. The goal was to reduce noise and improve the quality of clustering and class separability without removing too many valid samples.

After removing the identified outliers, the Fisher scores were recalculated to assess the effect on feature discriminability. Table 2.4 summarizes the Fisher scores before and after outlier removal.

Table 2.4: Fisher Scores Before and After Outlier Removal Using Isolation Forest

Feature	**Fisher Score Before**	**Fisher Score After**
EquivDiameter	3.2778	3.6325
Perimeter	3.2119	3.5138
MinorAxisLength	3.1344	3.4840
ConvexArea	2.9027	3.3320
Area	2.9035	3.3265
ShapeFactor1	3.1215	3.1560
MajorAxisLength	2.9482	3.1193
ShapeFactor2	2.3974	2.3655
Compactness	1.8473	1.8491
ShapeFactor3	1.8223	1.8237
AspectRatio	1.8027	1.8069
Eccentricity	1.5937	1.5960
Roundness	1.5048	1.5145
ShapeFactor4	0.7290	0.7310
Solidity	0.5141	0.5235
Extent	0.3750	0.3847

Notably, the most discriminative features, such as EquivDiameter, Perimeter, and MinorAxisLength, showed a marked increase in Fisher scores after cleaning, indicating improved class separability. This suggests that outlier removal reduced the within-class variance or clarified the differences between classes.

Table 2.5: Silhouette Scores Before and After Outlier Removal

Condition	**Silhouette Score**
Before Outlier Removal	0.2565
After Outlier Removal	0.2781

Furthermore, the silhouette score for clustering increased from 0.2565 before cleaning to 0.2781 after cleaning, as presented in Table 2.5. Although the increase is moderate, it indicates an improvement in clustering quality, attributed to the reduction of noise and anomalous samples that previously obscured cluster boundaries.

To visualize the overall structure of the dataset and the effect of outlier removal, PCA was performed, reducing the 16-dimensional feature space to two principal components.

Figure 2.9 shows the 2D PCA scatter plot of the cleaned dataset, revealing clearer class groupings and less noise compared to the original data.

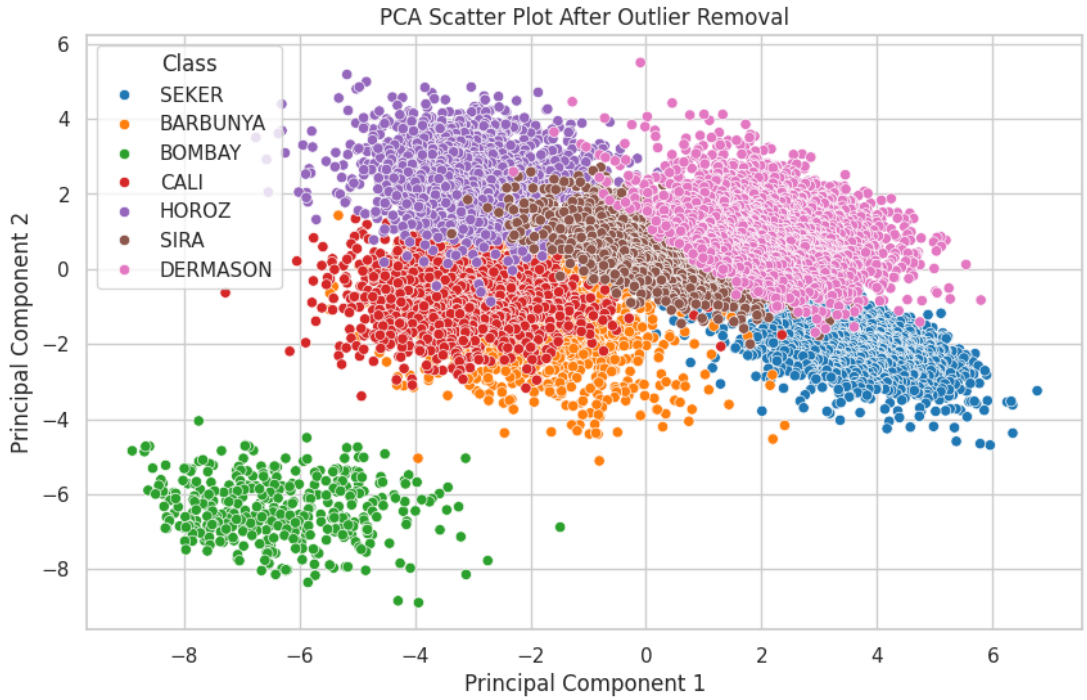

Figure 2.9: PCA plot after outlier removal

Overall, these improvements highlight the importance of outlier detection and removal in enhancing the quality and interpretability of statistical analyses.

3. CONCLUSION

This project explored the Dry Bean dataset through a comprehensive set of statistical analyses, dimensionality reduction techniques, clustering algorithms, and outlier detection methods, each revealing key insights into the structure and separability of the data.

The boxplot analysis highlighted that, although the dataset contains numerous outliers particularly in area-related features, the interquartile ranges were narrow, suggesting a compact distribution of most data points. Z-score normalization enabled fair comparison across features, and the generalized Fisher Distance identified EquivDiameter, Perimeter, and MinorAxisLength as the most discriminative features for class separation.

Applying Principal Component Analysis (PCA) revealed that the first two principal components captured over 80% of the dataset's variance while retaining meaningful class-discriminative information, as confirmed by both Fisher Scores and scatter plots. However, PCA, being an unsupervised technique, did not explicitly optimize for class separation. When compared to Linear Discriminant Analysis (LDA), PCA surprisingly achieved a higher silhouette score, suggesting that the underlying class structure may not be perfectly aligned with linear boundaries and that variance-preserving projections can sometimes outperform class-focused methods in complex datasets.

Clustering experiments using K-Means, DBSCAN, t-SNE, and SOM further illustrated the dataset's rich internal structure. While K-Means and DBSCAN, applied in the PCA-reduced space, provided reasonable but imperfect clusters, t-SNE and SOM, applied to the full dataset, offered superior visual separation of classes, highlighting their strength in capturing non-linear patterns and topological relationships.

Finally, applying the Isolation Forest algorithm for outlier detection and noise removal resulted in notable improvements in feature Fisher Scores and silhouette scores, confirming that removing even a small proportion of anomalous points can enhance class separability and clustering performance.

In summary, this project demonstrated the importance of combining multiple analytical approaches ranging from statistical summaries and supervised methods to unsupervised dimensionality reduction, clustering, and outlier detection to comprehensively understand and improve the performance of classification and clustering tasks on real-world datasets. The findings suggest that for datasets like Dry Bean, where non-linear re-

lationships and noise are present, unsupervised methods such as PCA and t-SNE, complemented by robust cleaning techniques, can be highly effective, sometimes even outperforming supervised alternatives.

BIBLIOGRAPHY

[1] J. Li, K. Cheng, S. Wang, *et al.*, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[2] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.

[3] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[4] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[5] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, University of California press, vol. 5, 1967, pp. 281–298.

[6] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, 1996, pp. 226–231.

[7] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[8] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[9] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, IEEE, 2008, pp. 413–422.

APPENDICES

The following Python code is an automatic conversion from a Jupyter notebook to a '.py' script, used for reproducing the experiments and analysis described in this work. To reproduce the results, and better understanding of the code, please check out the given Jupyter notebook.

```
# -*- coding: utf-8 -*-
"""cse555_final_project.ipynb

Automatically generated by Colab.

# 0. Setup
"""

!pip install ucimlrepo
!pip install minisom

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import
   LinearDiscriminantAnalysis as LDA
from sklearn.cluster import KMeans, DBSCAN
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score
from sklearn.ensemble import IsolationForest
from minisom import MiniSom
from ucimlrepo import fetch_ucirepo

# Fetch and load the Dry Bean dataset using ucimlrepo
dry_bean = fetch_ucirepo(id=602)
X = dry_bean.data.features
y = dry_bean.data.targets.iloc[:, 0]
class_labels = y.unique()

```

```
dry_bean.variables

"""# 1. Plot and Interpret Boxplots"""

import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="whitegrid")
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(20, 16))
axes = axes.flatten()
palette = sns.color_palette("Set2")

# Plot each attribute's boxplot horizontally
for i, column in enumerate(X.columns):
    sns.boxplot(x=X[column], ax=axes[i], color=palette[i % len
        (palette)])
    axes[i].set_title(column, fontsize=12)
    axes[i].set_xlabel("")

plt.tight_layout()
plt.suptitle("Boxplots of Dry Bean Attributes", fontsize=16, y
    =1.02)
plt.show()

"""# 2. Z-Score Normalization and Fisher Distance Analysis"""

def calculate_fisher_distance(X, y):
    class_labels = y.unique()
    C = len(class_labels)
    fisher_scores = {}

    for i in range(X.shape[1]):
        total_fd = 0
        valid_pairs = 0
        for j in range(C):
            for k in range(j + 1, C):
                group_j = X.loc[y == class_labels[j], X.
                    columns[i]]
                group_k = X.loc[y == class_labels[k], X.
                    columns[i]]

```

```
                # Skip if either group is empty
                if group_j.empty or group_k.empty:
                    continue

                mu_j = group_j.mean()
                mu_k = group_k.mean()
                var_j = group_j.var()
                var_k = group_k.var()

                # Skip if variance sum is zero (avoid division by zero)
                std_sum = np.sqrt(var_j + var_k)
                if std_sum == 0 or np.isnan(std_sum):
                    continue

                mu_diff = abs(mu_j - mu_k)
                total_fd += mu_diff / std_sum
                valid_pairs += 1

        # Avoid division by zero if no valid pairs found
        if valid_pairs == 0:
            fisher_scores[X.columns[i]] = np.nan
        else:
            fisher_scores[X.columns[i]] = total_fd / valid_pairs

    # Sort scores in descending order, ignoring NaNs
    return dict(sorted(
        ((k, v) for k, v in fisher_scores.items() if not np.isnan(v)),
        key=lambda item: item[1], reverse=True))

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
fisher_scores = calculate_fisher_distance(X_scaled_df, y)

print("Fisher Scores:")
for k, v in fisher_scores.items():
    print(f"{k}: {v:.4f}")

```

```
"""# 3. PCA Transformation and Fisher Distance Comparison"""

from sklearn.decomposition import PCA

# PCA transformation
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Fisher Distance on PCA-transformed features
fisher_pca = calculate_fisher_distance(pd.DataFrame(X_pca), y)

# Compare Fisher Scores with Eigenvalues
explained_var = pca.explained_variance_ratio_

# Print both for comparison
print(f"{'PC':<6} {'Explained Var':<15} {'Fisher Score':<15}")
for i, (ev, fs) in enumerate(zip(explained_var, fisher_pca.
    values())):
     print(f"PC{i+1:<3} {ev:<15.4f} {fs:<15.4f}")

"""# 4. Scatter Plot Visualization"""

# Top 2 components
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y, palette="
    tab10", s=20)
plt.title("Top 2 Principal Components")

# Bottom 2 components
plt.subplot(1,2,2)
sns.scatterplot(x=X_pca[:,-2], y=X_pca[:,-1], hue=y, palette="
    tab10", s=20)
plt.title("Bottom 2 Principal Components")
plt.tight_layout()
plt.show()

"""# 5. Dimensionality Reduction with LDA"""

import seaborn as sns
import pandas as pd
```

```
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

lda_df = pd.DataFrame(X_lda, columns=["LDA1", "LDA2"])
lda_df["Label"] = y

plt.figure(figsize=(8, 6))
sns.scatterplot(data=lda_df, x="LDA1", y="LDA2", hue="Label",
    palette="tab10", s=20)

plt.title("LDA Projection of Dry Bean Dataset (2D)")
plt.xlabel("LDA1")
plt.ylabel("LDA2")
plt.legend(title="Bean Type")
plt.grid(True)
plt.tight_layout()
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import silhouette_score

# Prepare DataFrames for Seaborn
lda_df = pd.DataFrame(X_lda, columns=["LDA1", "LDA2"])
lda_df["Label"] = y

pca_df = pd.DataFrame(X_pca[:, :2], columns=["PC1", "PC2"])
pca_df["Label"] = y

# Set up side-by-side subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

sns.scatterplot(data=lda_df, x="LDA1", y="LDA2", hue="Label",
    palette="tab10", s=20, ax=axs[0])
axs[0].set_title("LDA Projection")
axs[0].grid(True)
```

```
183
184 sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="Label",
       palette="tab10", s=20, ax=axs[1])
185 axs[1].set_title("PCA␣Projection")
186 axs[1].grid(True)
187
188 plt.tight_layout()
189 plt.show()
190
191 sil_lda = silhouette_score(X_lda, y)
192 sil_pca = silhouette_score(X_pca[:, :2], y)
193
194 print(f"Silhouette␣Score␣(LDA):␣{sil_lda:.4f}")
195 print(f"Silhouette␣Score␣(PCA):␣{sil_pca:.4f}")
196
197 """# 6. Clustering with K-Means, DBSCAN, t-SNE, and SOM
198
199 ## K-Means
200 """
201
202 import numpy as np
203 import matplotlib.pyplot as plt
204 import seaborn as sns
205 from sklearn.cluster import KMeans
206
207 # Fit KMeans on PC1 and PC2
208 kmeans = KMeans(n_clusters=7, random_state=0)
209 k_labels = kmeans.fit_predict(X_pca[:, :2])
210
211 # Create side-by-side plots
212 fig, axes = plt.subplots(1, 2, figsize=(14, 6))
213
214 sns.scatterplot(
215     x=X_pca[:, 0], y=X_pca[:, 1], hue=k_labels, palette="tab10
        ",
216     s=30, edgecolor='w', alpha=0.8, ax=axes[0]
217 )
218 axes[0].set_title("KMeans␣Clusters␣(on␣PC1␣&␣PC2)")
219 axes[0].set_xlabel("PC1")
220 axes[0].set_ylabel("PC2")
221 axes[0].legend(title="Cluster")
```

```
222
223 sns.scatterplot(
224     x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette="tab10",
225     s=30, edgecolor='w', alpha=0.8, ax=axes[1]
226 )
227 axes[1].set_title("Original Class Labels (on PC1 & PC2)")
228 axes[1].set_xlabel("PC1")
229 axes[1].set_ylabel("PC2")
230 axes[1].legend(title="Class")
231
232 plt.tight_layout()
233 plt.show()
234
235 """## DBSCAN"""
236
237 from sklearn.cluster import DBSCAN
238
239 # eps is the maximum distance for a point to be considered as
    part of a neighborhood
240 dbscan = DBSCAN(eps=0.5, min_samples=5)
241 y_db = dbscan.fit_predict(X_pca[:, :2])
242
243 df_dbscan = pd.DataFrame({
244     'PC1': X_pca[:, 0],
245     'PC2': X_pca[:, 1],
246     'Cluster': y_db
247 })
248
249 plt.figure(figsize=(8, 6))
250 sns.scatterplot(
251     data=df_dbscan,
252     x='PC1',
253     y='PC2',
254     hue='Cluster',
255     palette=sns.color_palette("tab10", len(set(y_db))),
256     s=30,
257     edgecolor='w',
258     alpha=0.8
259 )
260
261 plt.title(f"DBSCAN Clustering (PC1 & PC2) - {len(set(y_db)) -
```

```
    (1 if -1 in y_db else 0)} Clusters")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="DBSCAN Label")
plt.show()

sil_dbscan = silhouette_score(X_pca[:, :2], y_db)
print(f"Silhouette Score (DBSCAN): {sil_dbscan:.4f}")

"""## t-SNE"""

from sklearn.manifold import TSNE

# Perform t-SNE
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
sns.scatterplot(x=X_tsne[:,0], y=X_tsne[:,1], hue=y, palette="
    tab10", alpha=0.8, s=20)
plt.title("t-SNE Visualization with True Labels")
plt.xlabel("t-SNE1")
plt.ylabel("t-SNE2")
plt.show()

sil_tsne = silhouette_score(X_tsne, y)
print(f"Silhouette Score (t-SNE): {sil_tsne:.4f}")

"""## SOM"""

from minisom import MiniSom

# Define SOM parameters
som_shape = (10, 10)  # grid size

# Initialize SOM
som = MiniSom(som_shape[0], som_shape[1], X_scaled.shape[1],
    sigma=1.5, learning_rate=.5)

# Initialize weights randomly
som.random_weights_init(X_scaled)
```

```
# som.pca_weights_init(X_scaled)

# Train the SOM
som.train_random(X_scaled, 1000)  # 1000 iterations

w_x, w_y = zip(*[som.winner(x) for x in X_scaled])
w_x = np.array(w_x)
w_y = np.array(w_y)

# Define the color palette
unique_classes = y.unique()
palette = sns.color_palette("tab10", n_colors=len(
    unique_classes))
colors_in_order = [palette[i] for i, cls in enumerate(
    unique_classes)]

# Create a mapping of classes to colors
class_colors = {cls: palette[i] for i, cls in enumerate(
    unique_classes)}

plt.figure(figsize=(11, 10))
plt.pcolor(som.distance_map().T, cmap='bone_r', alpha=.2)
plt.colorbar()

for c in np.unique(y):
    idx_target = y == c
    plt.scatter(w_x[idx_target] + 0.5 + (np.random.rand(np.sum
        (idx_target)) - 0.5) * 0.8,
                w_y[idx_target] + 0.5 + (np.random.rand(np.sum
                    (idx_target)) - 0.5) * 0.8,
                s=20, label=c, alpha=0.7, edgecolor='w', color
                    =class_colors[c])

plt.legend(loc='upper left')
plt.grid()
plt.show()

import matplotlib.gridspec as gridspec

labels_map = som.labels_map(X_scaled, y)

```

```
fig = plt.figure(figsize=(10, 10))
the_grid = gridspec.GridSpec(som_shape[0], som_shape[1], fig)
for position in labels_map.keys():
    label_fracs = [labels_map[position][y] for y in y.unique()
        ]
    plt.subplot(the_grid[som_shape[1] - 1 - position[1],
        position[0]], aspect=1)
    patches, texts = plt.pie(label_fracs, colors=
        colors_in_order)

plt.legend(labels=y.unique(), loc="best", bbox_to_anchor=(0,
    -0.05), ncol=1)
plt.show()

"""# 7. Outlier Detection and Data Cleaning"""

from sklearn.ensemble import IsolationForest

# Normalize the features using z-score
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize and fit the Isolation Forest
iso_forest = IsolationForest(contamination=0.01, random_state
    =42)
outlier_preds = iso_forest.fit_predict(X_scaled)

# Outlier mask: -1 indicates outlier, 1 indicates inlier
outliers = outlier_preds == -1
inliers = outlier_preds == 1
mask = outlier_preds != -1
X_clean = X_scaled[inliers]
y_clean = y[inliers]

print(f"Original␣sample␣size:␣{X.shape[0]}")
print(f"Cleaned␣sample␣size:␣{X_clean.shape[0]}")
print(f"Number␣of␣outliers␣detected␣and␣removed:␣{X.shape[0]␣-
    ␣X_clean.shape[0]}")

# Re-run PCA on cleaned data
X_pca_clean = pca.fit_transform(X_clean)
```

```
370
371 plt.figure(figsize=(10,6))
372 sns.scatterplot(x=X_pca_clean[:,0], y=X_pca_clean[:,1], hue=
       y_clean, palette='tab10', legend='full')
373 plt.title('PCA Scatter Plot After Outlier Removal')
374 plt.xlabel('Principal Component 1')
375 plt.ylabel('Principal Component 2')
376 plt.show()
377
378 X_clean_df = pd.DataFrame(X_clean, columns=X.columns).
       reset_index(drop=True)
379 y_clean_reset = y_clean.reset_index(drop=True)
380
381 # Recalculate Fisher Scores on cleaned data
382 fisher_clean = calculate_fisher_distance(X_clean_df,
       y_clean_reset)
383
384 print("Fisher Scores AFTER Outlier Removal:")
385 for k, v in fisher_clean.items():
386     print(f"{k}: {v:.4f}")
387
388 from sklearn.metrics import silhouette_score
389 from sklearn.cluster import KMeans
390
391 # Before cleaning
392 kmeans_before = KMeans(n_clusters=7, random_state=0).fit(
       X_scaled)
393 sil_before = silhouette_score(X_scaled, kmeans_before.labels_)
394
395 # After cleaning
396 kmeans_after = KMeans(n_clusters=7, random_state=0).fit(
       X_clean)
397 sil_after = silhouette_score(X_clean, kmeans_after.labels_)
398
399 print(f"Silhouette Score BEFORE Cleaning: {sil_before:.4f}")
400 print(f"Silhouette Score AFTER Cleaning:  {sil_after:.4f}")
```

Listing 3.1: Project Python Code