Capstone Project: Dry Bean Dataset

Author :
Angel Enrique MEDINA PEREZ

Year : 2024

Introduction

This report aims to put into practice the knowledge acquired during the course of the MLOps bootcamp, through a small but enriching exercise that emulates the deployment of an ML pipeline in a professional environment. First, a dataset (Dry Bean Dataset) will be analyzed through an Exploratory Data Analysis. Subsequently, we will explore the challenges and questions that can be addressed and solved by leveraging ML models. Python will act as the main technological conduit to perform our analysis and execution tasks.

Contents

Chapter 1

Exploratory Data Analysis

In this first phase we will focus on knowing the characteristics of our data, such as its type, the completeness of each category, finding possible outliers and finally exploring possible correlations.

1.1 Loading the data

The data will be loaded from the $xlsx$ file found in the dataset repository.

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import math
%matplotlib inline

file_path = "data/DryBeanDataset/Dry_Bean_Dataset.xlsx"
data = pd.read_excel(file_path)
```

1.2 Dataset characteristics

Some of the characteristics of the dataset that were analyzed will be summarized below. The goal is to find possible records with missing columns or null values and verify that the data types are appropriate for the next steps.

1.2.1 Shape

Rows: 13611, Columns: 17

1.2.2 Missing values

No missing values were found in this dataset after executing:

```
data.isnull().sum()
```

1.2.3 Data types

The following data types were obtained by executing:

```
data.dtypes
```

There are 17 features, of which 16 are numerical and 1 is a string, representing the class of the row.

Area	int64
Perimeter	float64
MajorAxisLength	float64
MinorAxisLength	float64
AspectRation	float64
Eccentricity	float64
ConvexArea	int64
EquivDiameter	float64
Extent	float64
Solidity	float64
roundness	float64
Compactness	float64
ShapeFactor1	float64
ShapeFactor2	float64
ShapeFactor3	float64
ShapeFactor4	float64
Class	object

Table 1.1: Data types obtained with $describe$ method.

1.2.4 Summary

This dataset is composed by 13,611 entries and 17 features, of which 16 are numerical and 1 is categorical (string). It also does not include any Null value. Therefore, no transformation to the dataset is required at this point of the project.

1.3 Distribution

As can be seen in the histogram below, some of the numerical distributions have long tails, which implies that the data are loaded on one of the extremes of the range. In some cases, it is even observed that there are two modes, which probably indicates that some bean classes are different from the rest in that particular feature.

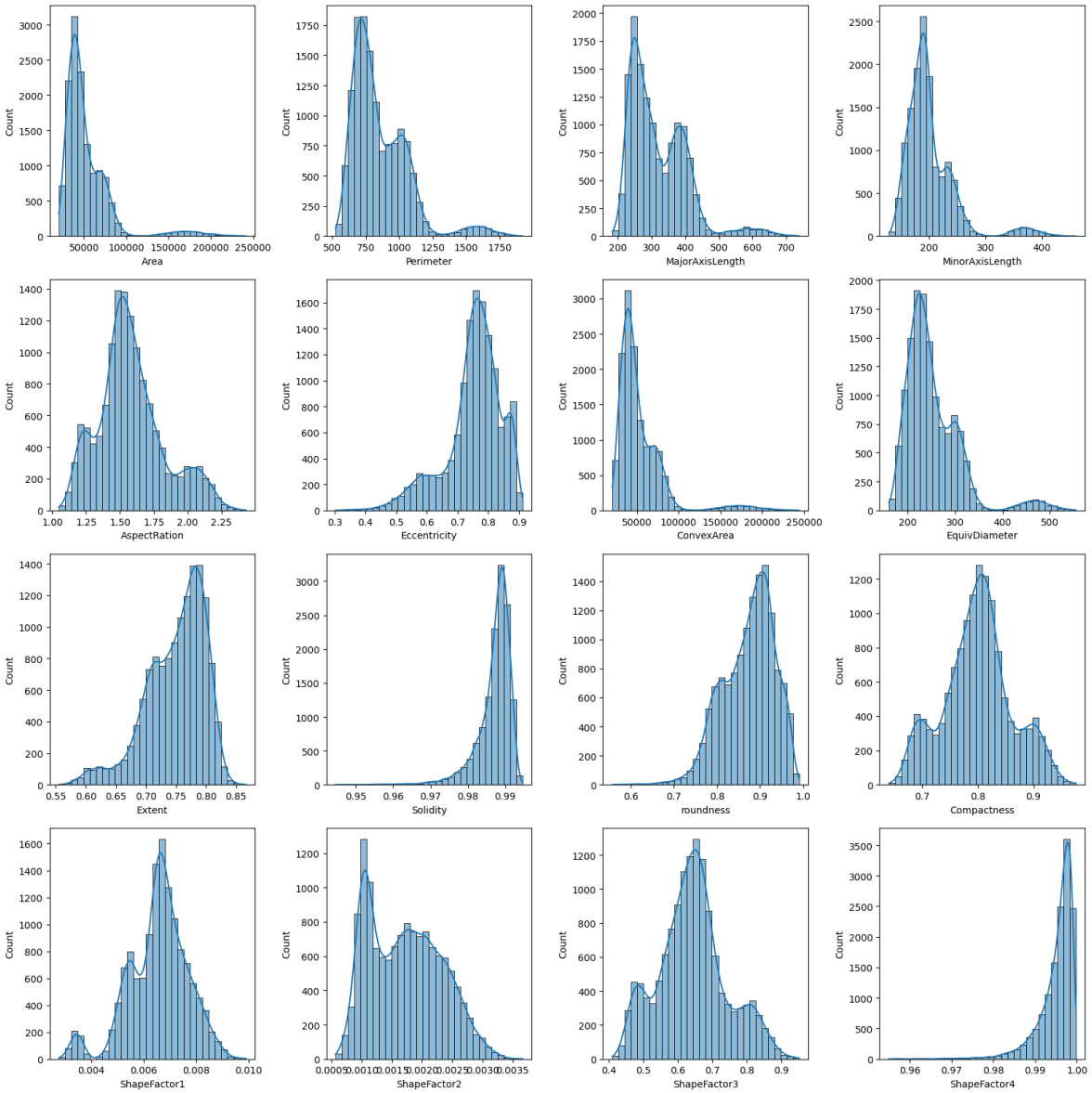

Figure 1.1: Histogram per feature.

1.4 Possible Outliers

In this stage, we will try to identify possible outliers in each of the features whose data type is $int64$ or $float64$. Those observation points distant from other observations can be visually identified using a box plot, where the minimum, first quartile, median, third quartile, and maximum values are drawn.

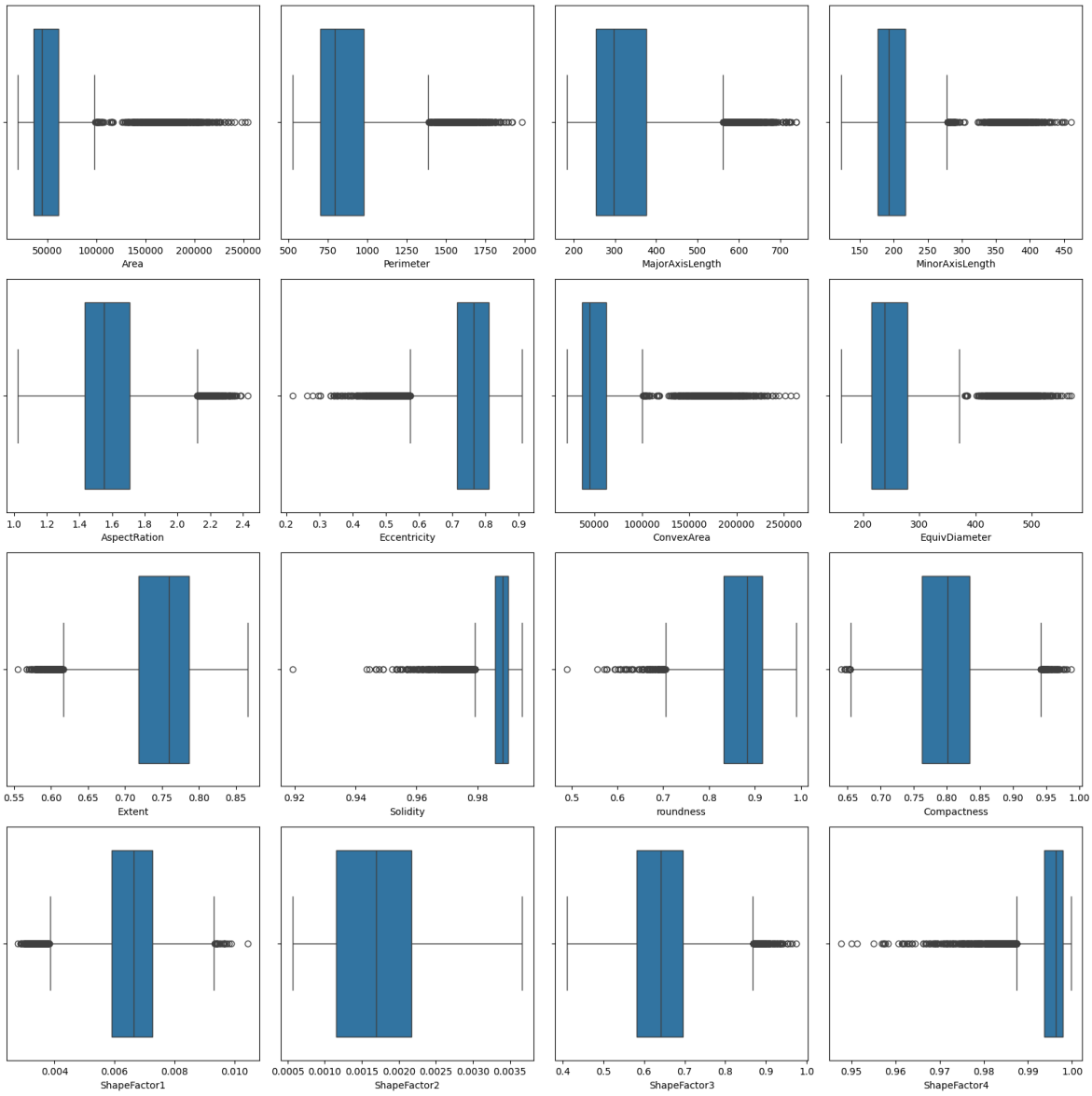

Figure 1.2: Box plot per category.

In order not to affect the size of the dataset or cause the loss of potentially valuable information, only those observations that are furthest from the set of the rest of the observations will be eliminated, even if this set exceeds the value $max = Q3 + 1.5IQR$. Based on the box plots and the dataset, the limits to consider for filtering outliers are:

- Eccentricity >= 0.3
- ConvexArea <= 250000
- Solidity >= 0.94
- Roundness >= 0.51
- ShapeFactor1 <= 0.0104
- ShapeFactor4 >= 0.954

1.5 Possible correlations

The correlations between features were analyzed thanks to a correlation matrix. Visually, a heatmap was used to denote the pairs of features with the highest correlations.

```
corr_matrix = data.corr(numeric_only=True)
sb.heatmap(corr_matrix, annot=True, cmap='YlOrBr')
plt.show()
```

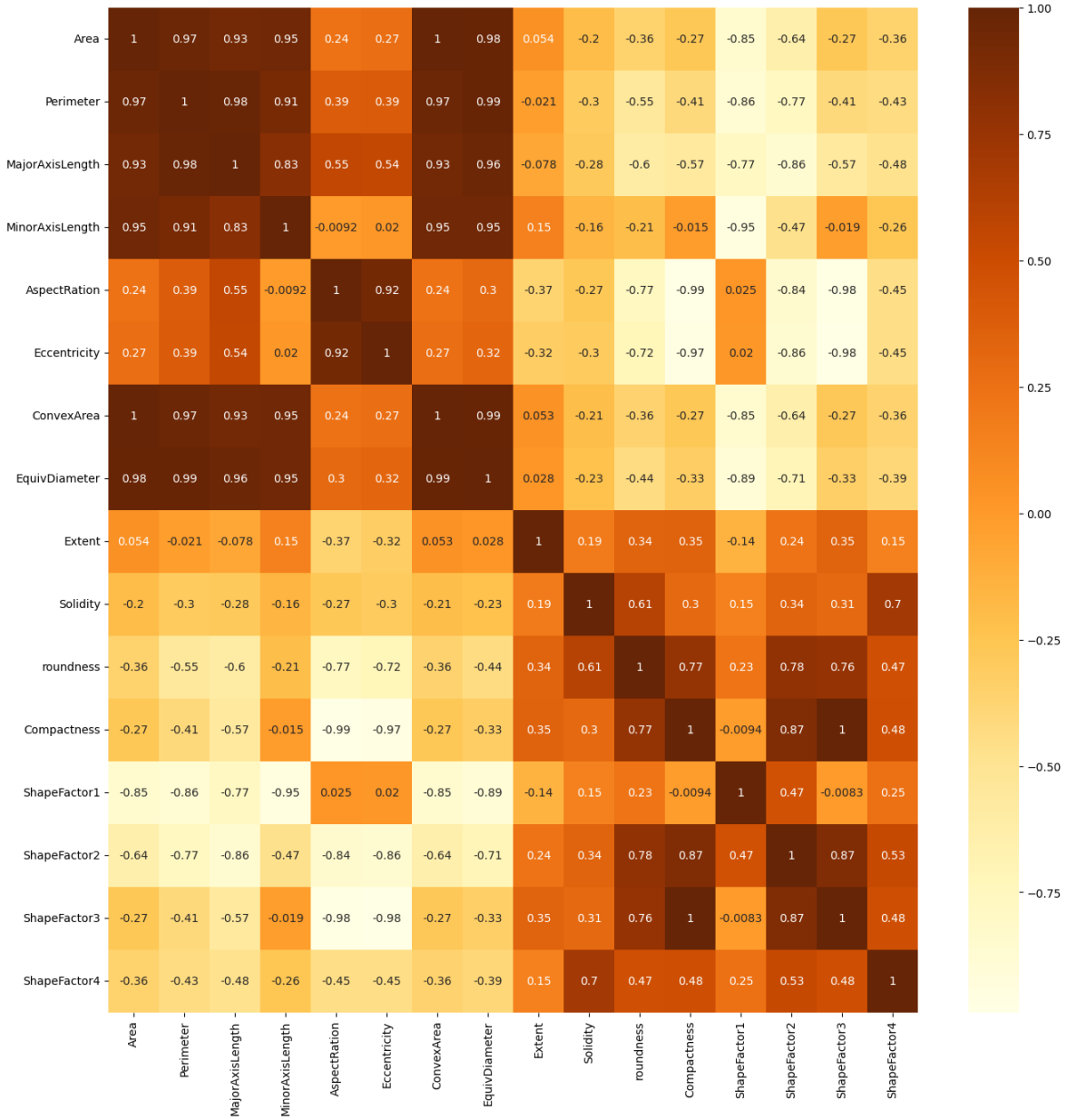

Figure 1.3: Correlation heatmap.

As can be seen in the previous results, there are several highly related features, mainly due to the fact that these are characteristics that describe the physical appearance of the bean, such as area and perimeter. Likewise, those physical characteristics in which bean dimensions are

involved present a high correlation, such as the duples $(Area|Perimeter, MajorAxisLength)$, $(Area|Perimeter, MinorAxisLength)$, $(Area|Perimeter, EquivDiameter)$, $(Area|Perimeter, ConvexArea)$, etc.

Said the above, $Area$ and $Perimeter$ will be dropped from the train dataset as removal of correlated features helps improve a model's ability to generalize to new data, which reduces the risk of overfitting and makes the model more robust. Similarly, $Compactness$ and $ShapeFactor3$ present a high correlation between them ($correlation = 1$), so both features will be removed from the train dataset.

1.6 Data Cleaning

At this stage, the data will be cleaned acoording to the results of the EDA.

```
# Removing outliers

initial_rows = data.shape[0]

data = data[data['Eccentricity'] >= 0.3]
data = data[data['ConvexArea'] <= 250000]
data = data[data['Solidity'] >= 0.94]
data = data[data['roundness'] >= 0.51]
data = data[data['ShapeFactor1'] <= 0.0104]
data = data[data['ShapeFactor4'] >= 0.954]

final_rows = data.shape[0]

print(f'Initial rows = {initial_rows}, final rows = {final_rows},
    difference = {initial_rows - final_rows}')

>> Result: Initial rows = 13611, final rows = 13599, difference = 12.
```

A total of 12 rows were removed from the dataset, which does not represent a number of observations that could be detrimental to the model, considering the initial size of the dataset.

Chapter 2

Problem definition

In Mexico, common beans are the second most produced and consumed crop after maize[1]. During recent years, attempts have been made to develop policies and methods that improve the competitiveness and productivity of beans in the Mexican national territory[2]. However, it is interesting to explore the possibility of using machine learning techniques to improve some quality criteria in the cultivation and harvesting of beans. The classification of bean types thanks to the extraction of their physical characteristics and the use of computer vision systems becomes relevant in the context of quality control.

Bean classification allows bean seeds to be separated based on quality, size, shape and other attributes. This ensures that the seeds used for sowing are in optimal condition and have a high probability of germination and healthy plant development. High quality seeds tend to produce more uniform crops with higher yields.

Bean classification can also help identify and remove weed seeds or other impurities that might be present in the batch. This is crucial to prevent the spread of weeds and diseases that could compete with crops or reduce their yield.

In summary, dry bean classification is essential to ensure optimal crop quality and yield, as well as to minimize problems associated with weeds, diseases and uneven planting.

Having said the above, the problem to be addressed will consist of implementing a dry bean classifier that can be deployed in an agricultural production context, aimed at increasing the quality metrics of the bean crop and harvest.

2.1 Why MLOps for this scenario?

MLOps ensures consistency, availability and standardization of data throughout the entire design, implementation, testing, monitoring and life cycle management [3]. On the one hand, ML models depend on data quality, availability and relevance. After performing the EDA, it is clear that our dry bean dataset is a good candidate to be implemented in an MLOps pipeline; it is sufficient and quality data to be subjected to an ML process. On the other hand, although it is true that the success of the model will depend largely on the reliability and robustness of the computer vision system that reports the observations of the beans, it will be assumed that it works adequately for this exercise.

As established in the previous section, the main objective of this ML model implementation exercise is to improve the quality and effectiveness with which beans are classified, and of course to automate the classification process, which in an agricultural production context , can be laborious and subject to human error.

Chapter 3

Architecture

The architecture to follow for this problem will be online and divided into 5 stages, EDA, Data Preparation, Training, Source and version control and Running Environment. The idea will be to do the EDA with the provided dataset and in the same way feed the Data Preparation process with this same data source (in case a re-training is required without going through the EDA stage). On the other hand, when training a model, there is the possibility of performing iterative hyper parameter tuning until the model meets the required validation. Subsequently, the model is packaged, versioned and saved in the common repository and then deployed together with the service that will host the model. Once the prediction service has been deployed in a running environment, client applications can consume this service and obtain predictions. In parallel, a monitoring service will be responsible for collecting the necessary metrics to determine if the model needs retraining.

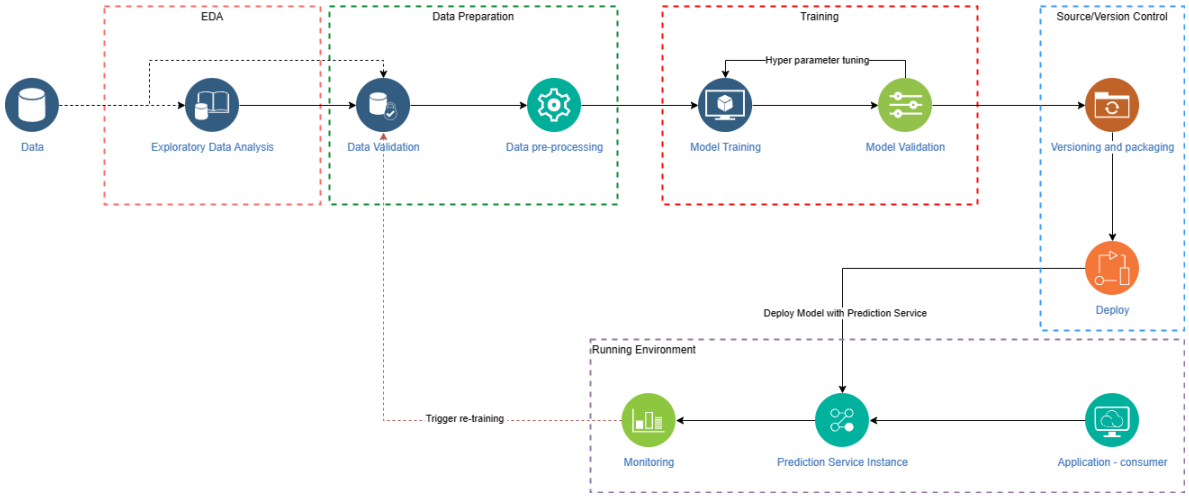

Figure 3.1: Proposed pipeline architecture.

Chapter 4

Choosing a model

4.1 Data Pre-processing

At this stage, the data will be cleaned and prepared according to the results of the EDA.

4.1.1 Removing Outliers

In this step, the objective is to remove those values that, according to the EDA, are outside the normal range of values of the sample set. The following code fragment shows the ranges of values that will be removed from the general dataset for each feature.

```
# Removing outliers

initial_rows = data.shape[0]

data = data[data['Eccentricity'] >= 0.3]
data = data[data['ConvexArea'] <= 250000]
data = data[data['Solidity'] >= 0.94]
data = data[data['roundness'] >= 0.51]
data = data[data['ShapeFactor1'] <= 0.0104]
data = data[data['ShapeFactor4'] >= 0.954]

final_rows = data.shape[0]

print(f'Initial rows = {initial_rows}, final rows = {final_rows},
    difference = {initial_rows - final_rows}')

>> Result: Initial rows = 13611, final rows = 13599, difference = 12.
```

A total of 12 rows were removed from the dataset, which does not represent a number of observations that could be detrimental to the model, considering the initial size of the dataset.

4.1.2 Data split

Dataset will be split into random train and test subsets. A 80-20 ratio is used in this dataset split. By using 80% of the data to train the model and the remaining 20% to evaluate it, a balance can be achieved between having enough data for the model to learn relevant patterns (training) and having enough data to evaluate the performance of the model on unseen data (test).

```
features = data.drop(columns=['Class']).columns

X = data[features]
y = data['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    train_size=0.8, random_state=None, shuffle=True)
```

4.1.3 Scaling

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data [4].

The StandardScaler from sklearn is implemented in this exercise in order to standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as $z = (x - u)/s$, where u is the mean of the training samples, and s is the standard deviation of the training samples.

Since StandardScaler is sensitive to outliers, and the features may scale differently from each other in the presence of outliers, these where removed in the previous section.

```
columns: list[str] = X_train.columns.to_list()

standard_scaler = StandardScaler()
X_train[columns] =
    pd.DataFrame(standard_scaler.fit_transform(X_train[columns]),
    index=X_train.index)
X_test[columns] =
    pd.DataFrame(standard_scaler.fit_transform(X_test[columns]),
    index=X_test.index)
```

4.1.4 Removing correlated features

As stated in the EDA, $Area$ and $Perimeter$ as well as $Compactness$ and $ShapeFactor3$ as these features present a high correlation between them, so the 4 features will be removed from the train dataset.

```
# Removing correlated features
dropped_columns = ['Area', 'Perimeter', 'Compactness', 'ShapeFactor3']
X_train = X_train.drop(columns=dropped_columns)
X_test = X_test.drop(columns=dropped_columns)
```

4.1.5 Models

The list of models to be tested after the data pre-processing can be found here below. These classification models where chosen according to the following criteria: the dry bean dataset is a small dataset with around 13K records and there is a relatively low number of features.

```
models = {
```

```
  "LogisticRegression": LogisticRegression(max_iter=1000),
  "DecisionTreeClassifier": DecisionTreeClassifier(),
  "SVC": SVC(max_iter=1000),
  "LinearSVC": LinearSVC(max_iter=1000, dual='auto'),
  "KNeighborsClassifier": KNeighborsClassifier(),
  "RandomForestClassifier": RandomForestClassifier()
}
```

The next step is to train each of the selected models. For this, we iterate over our list of models and train them using the split training datasets (X_train and y_train).

```
for name, model in models.items():
    model.fit(X_train, y_train)
```

The last step is to measure the performance of each trained model. For this, the classification of our X_test set will be predicted and the accuracy of each model will be calculated, comparing the predicted class and the expected class (y_pred and y_test). The model to be chosen will be the one with the best accuracy.

```
# Model Performance

print("Model Performance:")
for name, model in models.items():
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name}:")
    print(classification_report(y_test, y_pred))
    disp = ConfusionMatrixDisplay.from_predictions(y_true=y_test,
        y_pred=y_pred)
    disp.ax_.set_title(name)

plt.show()
```

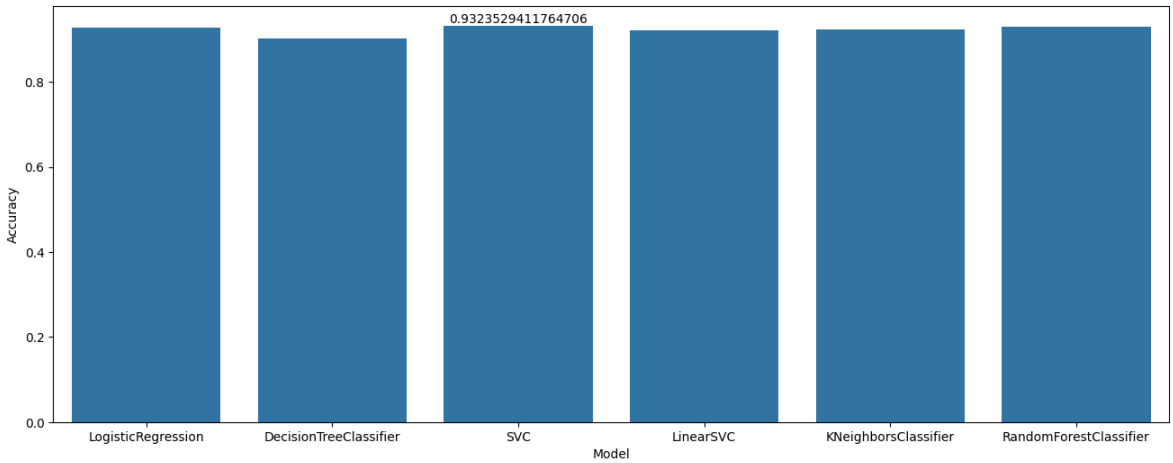

Figure 4.1: Model performance.

According to the results, the model that obtained the best score was SVC, so it will be the

model chosen to carry out the next step of this exercise. Below is the confusion matrix for the SVC model.

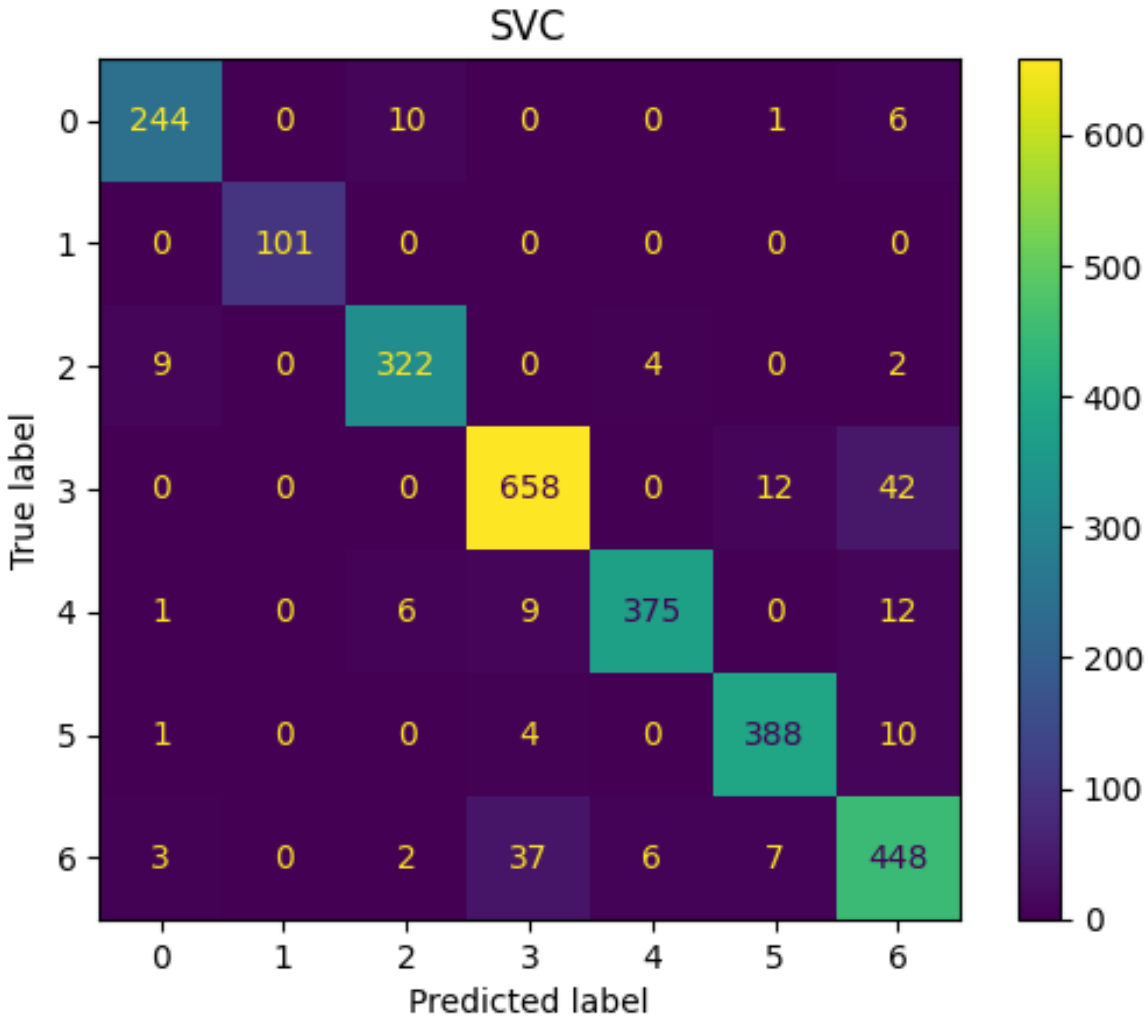

Figure 4.2: SVC Confusion Matrix.

Appendix A

Model performance

A.1 Performance results

LogisticRegression:

precision	recall	f1-score	support	
0	0.94	0.93	0.94	261
1	1.00	1.00	1.00	101
2	0.93	0.96	0.95	337
3	0.93	0.92	0.92	712
4	0.97	0.92	0.95	403
5	0.95	0.96	0.95	403
6	0.86	0.88	0.87	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

DecisionTreeClassifier:

precision	recall	f1-score	support	
0	0.86	0.92	0.89	261
1	1.00	0.99	1.00	101
2	0.92	0.93	0.92	337
3	0.91	0.89	0.90	712
4	0.96	0.89	0.93	403
5	0.94	0.95	0.95	403
6	0.82	0.85	0.83	503
accuracy	0.90	2720		
macro avg	0.92	0.92	0.92	2720
weighted avg	0.90	0.90	0.90	2720

SVC:

precision	recall	f1-score	support	
0	0.95	0.93	0.94	261
1	1.00	1.00	1.00	101
2	0.95	0.96	0.95	337
3	0.93	0.92	0.93	712
4	0.97	0.93	0.95	403
5	0.95	0.96	0.96	403
6	0.86	0.89	0.88	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

LinearSVC:

precision	recall	f1-score	support	
0	0.95	0.91	0.93	261
1	1.00	1.00	1.00	101
2	0.94	0.96	0.95	337
3	0.93	0.90	0.91	712
4	0.97	0.92	0.95	403
5	0.96	0.95	0.96	403
6	0.81	0.89	0.85	503
accuracy	0.92	2720		
macro avg	0.94	0.93	0.93	2720
weighted avg	0.92	0.92	0.92	2720

KNeighborsClassifier:

precision	recall	f1-score	support	
0	0.93	0.93	0.93	261
1	1.00	1.00	1.00	101
2	0.94	0.95	0.94	337
3	0.91	0.92	0.92	712
4	0.97	0.92	0.94	403
5	0.95	0.95	0.95	403
6	0.85	0.87	0.86	503
accuracy	0.92	2720		
macro avg	0.94	0.93	0.93	2720
weighted avg	0.92	0.92	0.92	2720

RandomForestClassifier:

precision	recall	f1-score	support	
0	0.94	0.91	0.93	261
1	1.00	1.00	1.00	101
2	0.93	0.96	0.94	337
3	0.92	0.94	0.93	712
4	0.97	0.93	0.95	403
5	0.95	0.96	0.95	403
6	0.88	0.88	0.88	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

Appendix B

References

1 Ramírez-Jaspeado, R., Palacios-Rojas, N., Nutti, M. R., & Pérez, S. (2020). ESTADOS POTENCIALES EN MÉXICO PARA LA PRODUCCIÓN y CONSUMO DE FRIJOL BIOFORTIFICADO CON HIERRO y ZINC.
Revista Fitotecnia Mexicana, 43(1), 11. https://doi.org/10.35196/rfm.2020.1.11

2 Salazar, J. A. G., Licea, G. R., Torres, A. S., & Rebollar, S. R. (2006). Políticas para mejorar la competitividad de la producción de maíz y frijol en México. Revista Fitotecnia Mexicana, 29(Es2), 115-121.

3 MLOps & Quality Data: The Path to AI Transformation - Spiceworks. (2023, 9 agosto). Spiceworks. https://www.spiceworks.com/tech/artificial-intelligence/guest-article/mlops-best-practices-optimizing-ai-with-quality-data/

4 sklearn.preprocessing.StandardScaler. (s.f.). Scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html