**IDS PROJECT**

Classification and Prediction of Wine Quality

**─**

**Group Members**

Anupam Sharma 19UCC140

Shwetank 19UCC134

Satyam goyal 19UCC013

Pranjal Sharma 19UCC144

**Project Objective**

The goal of this project is to apply appropriate machine learning algorithms to the data collection and derive inferences from it. We would do a preliminary analysis of the dataset after pre-processing and categorise our data into appropriate categories with sufficient validation.

**Source of Dataset**

The dataset was sourced from the UCI Machine Learning repository and can be found [here](https://archive.ics.uci.edu/ml/datasets/Wine+Quality).

**Introduction to Dataset**

The two datasets are related to white variant of the Portuguese "Vinho Verde" wine.

Under this project, we would be classifying the wine data in one of three categories where the categories indicate the three different cultivators.

**Data Description**

Following are the details of our dataset.

● Data Set Characteristic: Multivariate

● No. of Attributes: 12

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density



9 - pH

10 - sulphates

11 - alcohol

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

● Attribute Characteristics: Integer, Real

● Presence of Missing Values: False

**Implementation**

Importing the libraries

We start with importing the following libraries.

1. The **Numpy** and **Pandas** libraries provide high-performance data structures that are simple to use. These libraries are critical for working with the dataset, which is essentially a 178x13 matrix.

2. **Matplotlib** and **Seaborn** are libraries that help with data visualisation by providing a large range of curves and maps.

3. The **SciKit** Learn library is the major library used in the project for implementing the preprocessing and cross-validation procedures as well as applying the classification models.

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

%matplotlib inline

warnings.filterwarnings("ignore")

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

Importing the dataset

To make the code run faster and more efficiently, we used Google Colab.Uploaded the file and the read csv tool in pandas was used to read the wines.csv dataset, which was then saved as a DataFrame.

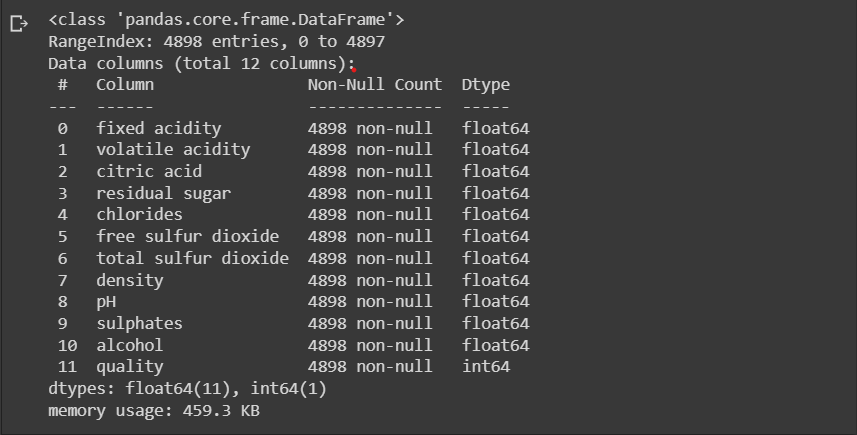
df = pd.read\_csv("winequality.csv")

Understanding the dataset

Using the df.info() command, we were able to determine the type and count of the attributes in the dataset we were working with.

#datatype information

df.info()



We also used the df.head() command to get a quick overview of the dataset.

df.head()

We also used df.describe() to figure out what our attributes' ranges and value distributions were. We were able to acquire a good picture of the dataset because all of our attributes were numerical.

#statistical info

df.describe()

Visualizing the dataset

We were able to get the most out of the dataset by employing EDA techniques. This aided us in recognising and dealing with outliers and abnormalities.

* **Boxplot**

Boxplot is another important tool that allows us to display the five statistical measures of a characteristic (Minimum, Maximum, Median and the two quartiles).

#box plots for the given dataset features

fig, ax = plt.subplots(ncols= 6,nrows=2,figsize=(17,8))

index = 0

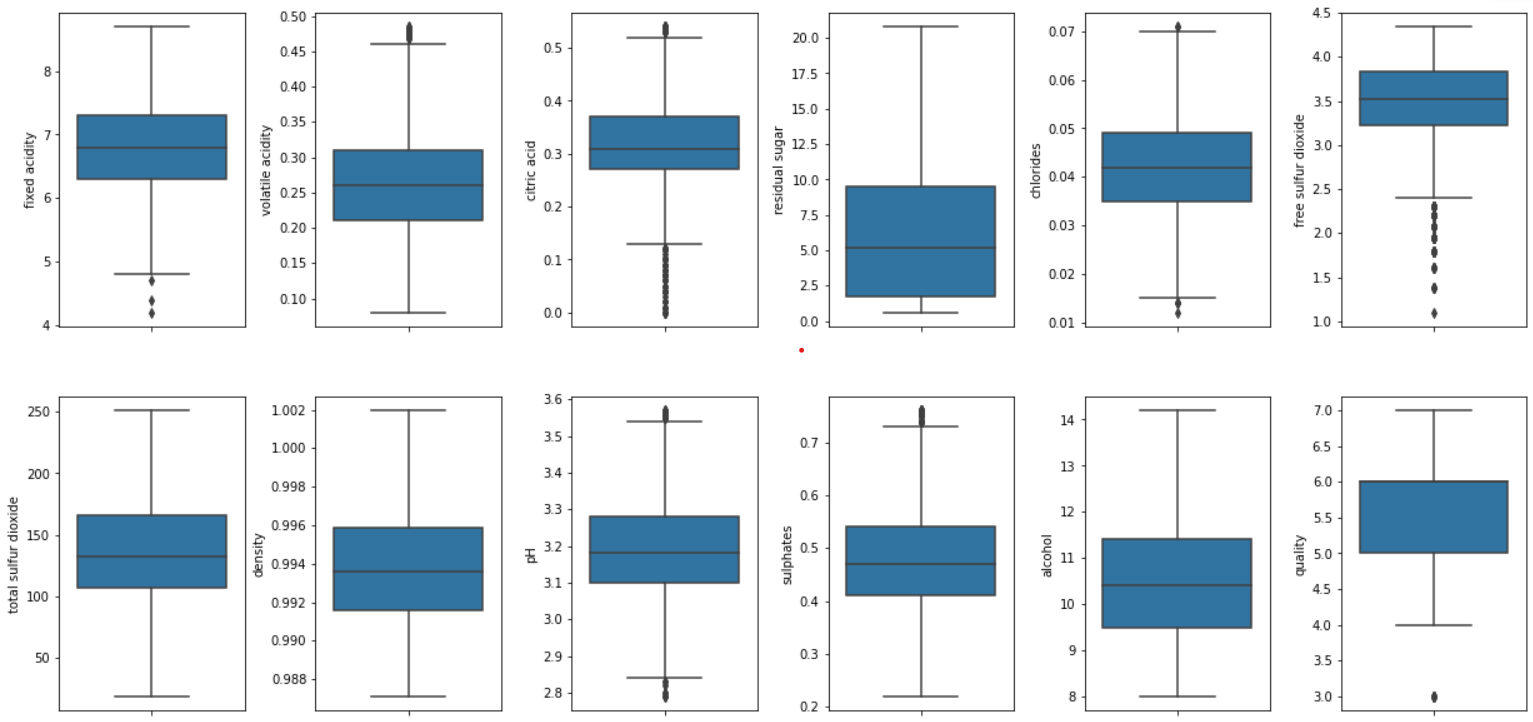
ax = ax.flatten()

for col, value in df.items():

sns.boxplot(y = col, data = df , ax = ax[index])

index += 1

plt.tight\_layout(pad = 0.5, w\_pad = 0.7, h\_pad = 5.0)



These two quartiles were employed to deal with the outliers in our dataset. We excluded outliers from our data because they can cause the forecasts to go haywire.

Handling Outliers

#handling outliers

for j,attr in enumerate(list(df.columns.values)):

Q1=df[attr].quantile(0.25)

Q3=df[attr].quantile(0.75)

IQR=Q3-Q1

Lower\_Whisker = Q1-1.5\*IQR

Upper\_Whisker = Q3+1.5\*IQR

df = df[df[attr]< Upper\_Whisker]

#box plots for the given dataset features after removing the outliers

fig, ax = plt.subplots(ncols= 6,nrows=2,figsize=(17,8))

index = 0

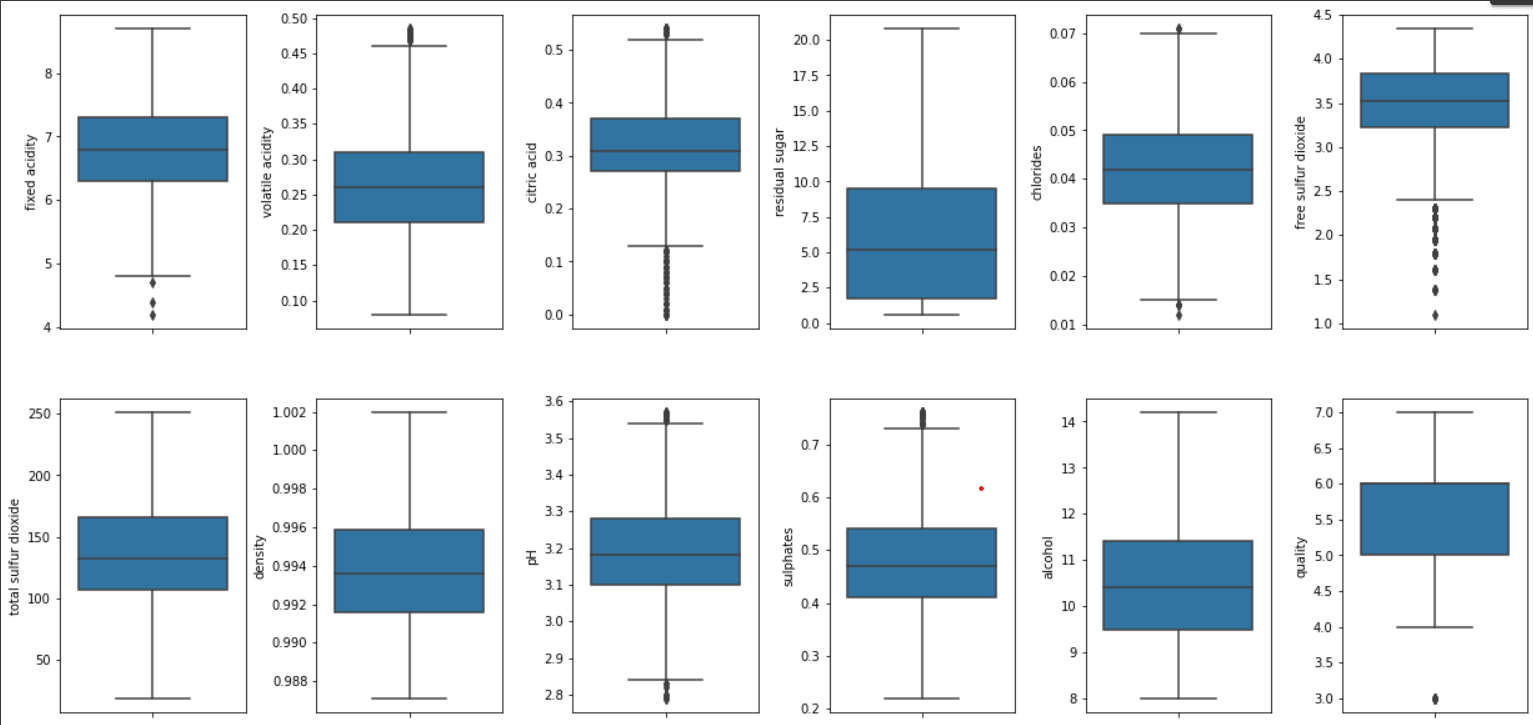
ax = ax.flatten()

for col, value in df.items():

sns.boxplot(y = col, data = df , ax = ax[index])

index += 1

plt.tight\_layout(pad = 0.5, w\_pad = 0.7, h\_pad = 5.0)



● **Dist-plot**

**#dist plot for the given dataset features**

**fig, ax = plt.subplots(ncols= 6,nrows=2,figsize=(20,10))**

**index = 0**

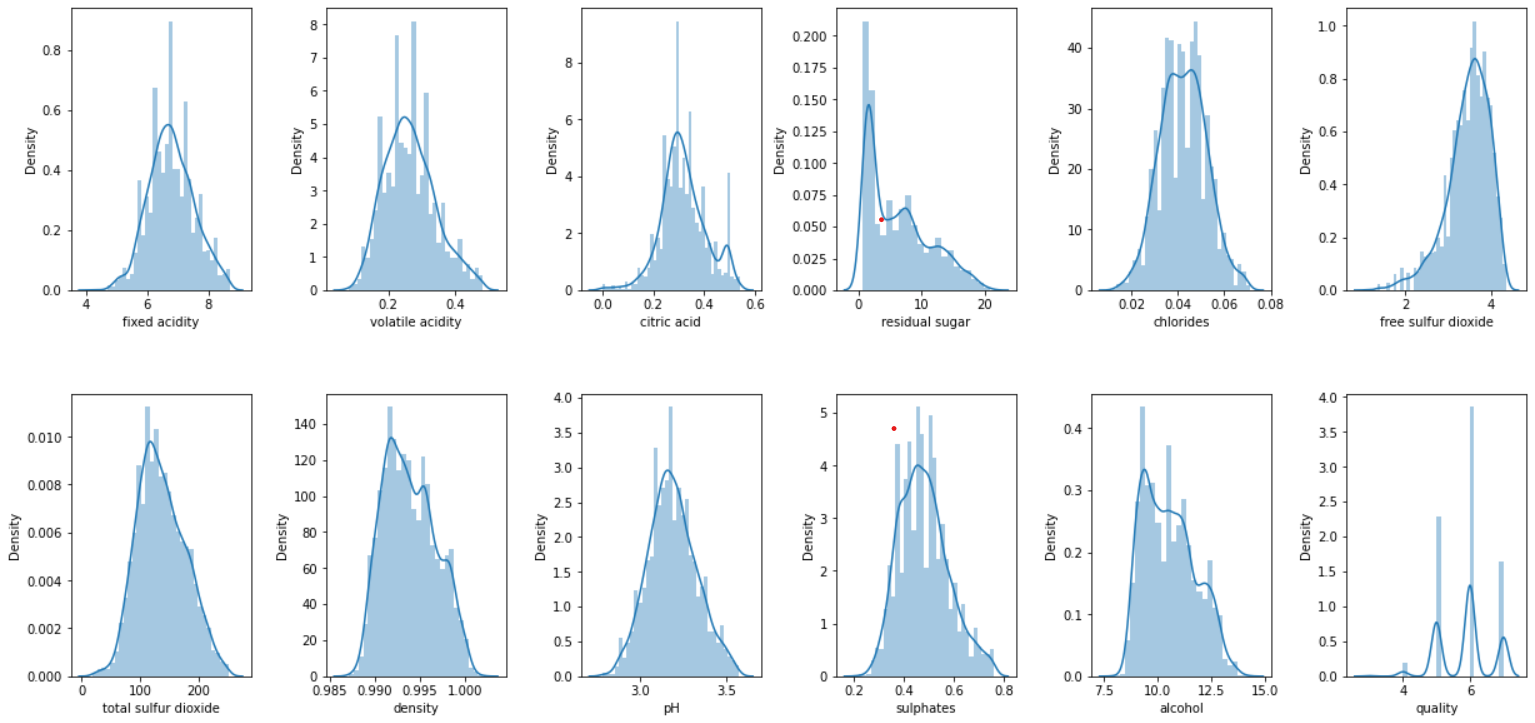
**ax = ax.flatten()**

**for col, value in df.items():**

**sns.distplot(value, ax = ax[index])**

**index += 1**

**plt.tight\_layout(pad = 0.5, w\_pad = 0.7, h\_pad = 5.0)**

****

● **Heatmap**

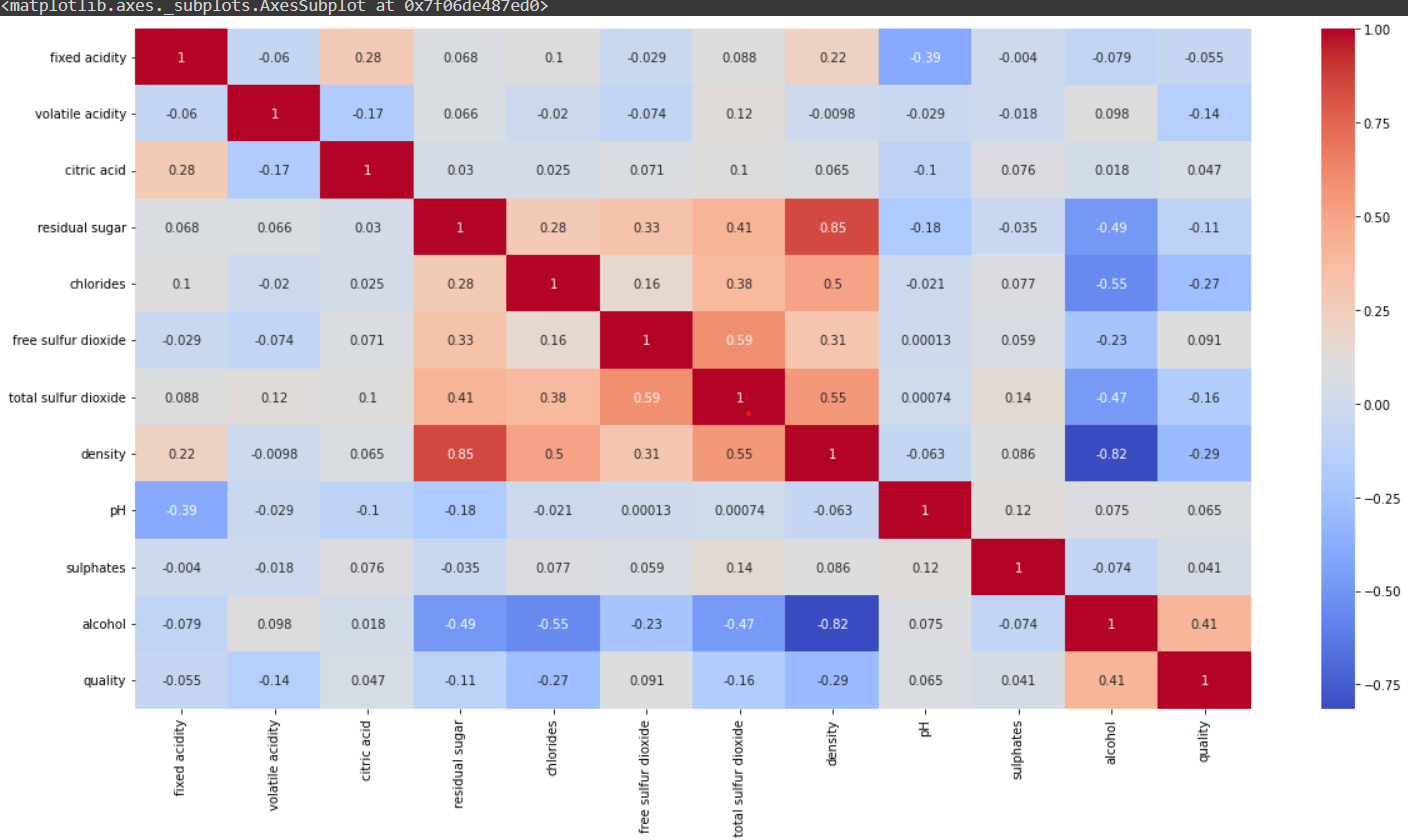
This correlation was plotted as a heatmap to make the analysis easier. The normalised correlation between the attributes is the core notion of heatmaps. The stronger the link, the lighter the colour of their box.

#Correlation Matrix

corr = df.corr()

plt.figure(figsize=(20,10))

sns.heatmap(corr, annot = True, cmap = 'coolwarm')



We can see that alcohol and density are negatively correlated so we can drop the feature “density”

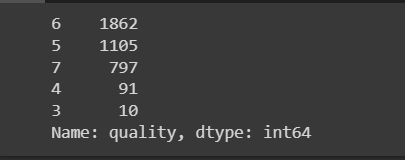
As ‘quality’ is the output variable we dropped it from ‘X’ and defined it as a new variable ‘y .

X = df.drop(columns = ['quality', 'density'])

y = df['quality']

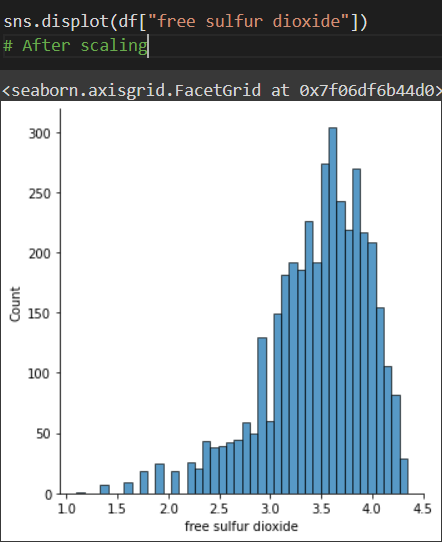
value\_counts() commands were used to determine the number of instances in each category. The.sort index() command was used to sort them in the correct numerical order.

y.value\_counts()



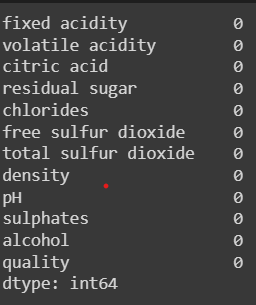
Adjusting the features to appropriate scale

# Scaling the feature "free sulfur dioxide"

df["free sulfur dioxide"] = np.log(1+ df["free sulfur dioxide"])

#check if the dataset has null values

df.isnull().sum()



Function “classify()”

#Classify function

from sklearn.model\_selection import cross\_val\_score , train\_test\_split

def classify(model, X, y):

x\_train,x\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state = 42)

#model training

model.fit(x\_train,y\_train)

print("Accuracy", model.score(x\_test, y\_test)\*100)

* **Splitting the dataset**

To measure the performance of various algorithms on the data, we split our data into training and testing datasets. We would apply the algorithms to the training data and then assess their accuracy on the testing data, giving us a clear picture of their performance.

x\_train,x\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state = 42)

* **Model training**

model.fit(x\_train,y\_train)

print("Accuracy", model.score(x\_test, y\_test)\*100)

Training the dataset

We started with the KNN model, using K=3.

We are using K nearest neighbors classification as it is quite easy to implement and during the training phase, it does not learn anything. The training data isn't used to derive any discriminative functions. In other words, it does not require any training. It saves the training dataset and uses it only when making real-time predictions to learn from it. This makes the KNN method much faster than other training-based algorithms like SVM and Linear Regression.Due to this any new data can be added seamlessly without affecting the accuracy of our model.

model = KNeighborsClassifier(n\_neighbors = 3)

classify(model, X, y)

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski', metric\_params=None, n\_jobs=None, n\_neighbors=3, p=2, weights='uniform')

The predictions showed an accuracy 76.03092783505154%

Secondly, we moved on to **Naive Bayes**.

We use the Naive Bayes Classification as it handles the continuous data as well as the discrete data well.It can handle a large number of predictors and data points.It is unaffected by non-essential characteristics.Also,predicting the test data set's class is simple and quick. It's also good at multi-class prediction. When the assumption of independence is met, a Naive Bayes classifier outperforms alternative models such as logistic regression and requires less training data.

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

classify(model, X,y)

GaussianNB(priors=None, var\_smoothing=1e-09)

The predictions showed an accuracy of 50.47250859106529%.

Finally, we do the **Linear Discriminant Analysis** on our dataset.

The benefit of LDA is that it takes information from both features to construct a new axis, reducing variance and increasing class distance between the two variables.The purpose of LDA is to project features from a higher-dimensional space onto a lower-dimensional space, avoiding the dimensionality curse while simultaneously saving resources and reducing dimensional costs.With dimensionality reduction, you may plot multi-dimensional data in just 2 or 3 dimensions. It enables the data to be presented in a clear and understandable manner for a layman.

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

model = LinearDiscriminantAnalysis()

classify(model, X , y)

The prediction showed an accuracy of 48.840206185567006%

**Conclusion**

In our research, we discovered that the data we got is a classification dataset. We discovered this by comparing and contrasting various visualisation techniques and classification algorithms. Based on the results, we can infer that KNN is the most appropriate classification algorithm for the dataset.