MILK QUALITY PREDICTION

1. INTRODUCTION

This project aims to develop an advanced algorithm that accurately predicts milk quality based on factors such as PH, Temperature, Taste, Odor, Fat, Turbidity, Color and Grade. The remaining section provides a comprehensive overview of the project, including objectives, methodology, implementation details, and evaluation results.

2. OBJECTIVES

The main objectives of the Milk Quality Prediction project are as follows:

1. Develop a predictive model that accurately estimates milk quality based on measurable parameters.
2. Enhance consumer satisfaction and safety by ensuring the production of high-quality milk products.

3. METHODOLOGY

The following method was used to achieve the objectives of the project:

3.1. Data Collection: The milk dataset was collected from Kaggle. The data include eight attributes: PH, Temperature, Taste, Odor, Fat, Turbidity, Color and Grade. It also contains 1059 rows.

```
# Displaying the first 5 Rows from the dataset
df.head()
```

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium

3.2. Exploratory Data Analysis: Visualization o the dataset was done using a bar chart, box plot, his plot, count plot, and pair plot to identify trends, distributions, outliers, and potential relationships between variables to know the key factors influencing milk production.

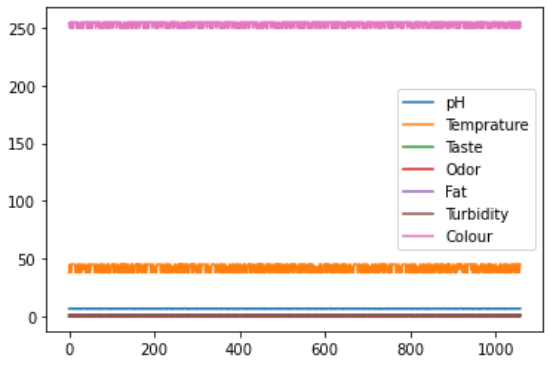


Fig1: Plotting the whole dataset

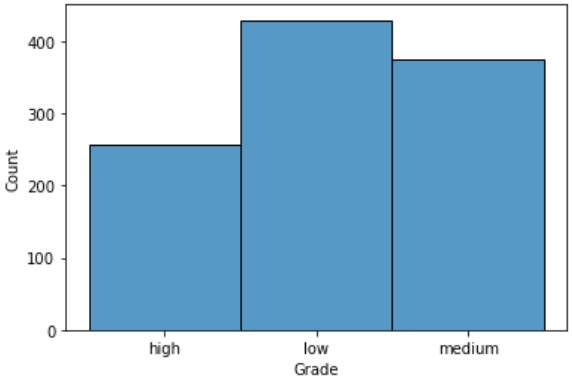


fig2: Grade by value count

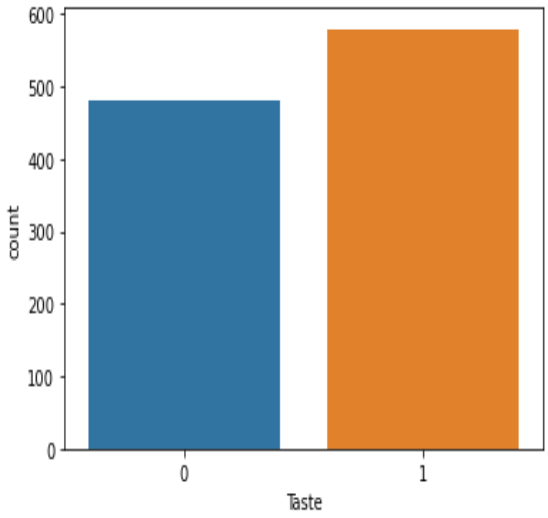


Fig3: Taste by value count

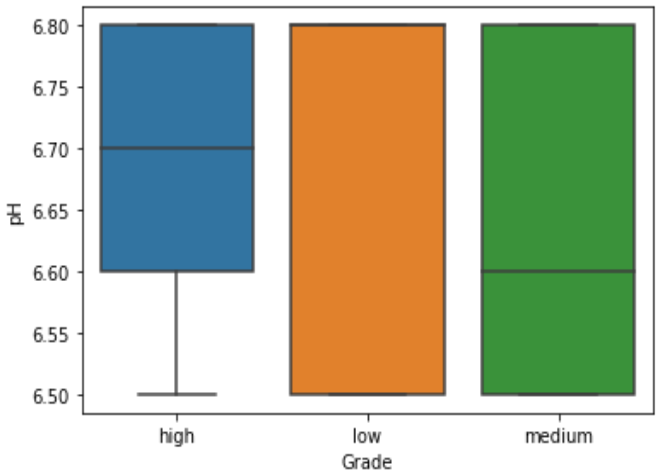


Fig4: Grade by PH

3.3. Data Preprocessing: The collected dataset was cleaned and prepared by removing outliers normalizing the dataset, and performing a label encoder on the Grade attribute.

#The following libraries were imported

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")
```

LABELENCODER

#Adjusting Temperature values to mitigate outliers

```
quantile1=df["Temprature"].quantile(0.25)

quantile2=df["Temprature"].quantile(0.75)

df["Temprature"]=np.where(df["Temprature"]<quantile1,quantile1,df["Temprature"])

df["Temprature"]=np.where(df["Temprature"]>quantile2,quantile2,df["Temprature"])
```

#Adjusting Colour values to mitigate outliers

```
quantile1=df["Colour"].quantile(0.25)

quantile2=df["Colour"].quantile(0.75)

df["Colour"]=np.where(df["Colour"]<quantile1,quantile1,df["Colour"])

df["Colour"]=np.where(df["Colour"]>quantile2,quantile2,df["Colour"])
```

#Adjusting PH values to mitigate outliers

```
quantile1=df["pH"].quantile(0.25)

quantile2=df["pH"].quantile(0.75)

df["pH"]=np.where(df["pH"]<quantile1,quantile1,df["pH"])

df["pH"]=np.where(df["pH"]>quantile2,quantile2,df["pH"])
```

NORMALIZATION

```
from sklearn.preprocessing import StandardScaler # importing the library

scaler = StandardScaler() # Create a StandardScaler object

x = scaler.fit_transform(x) # Standardize the data using the StandardScaler object
```

3.4. Feature Selection: Relevant features that can influence milk production were selected to improve the accuracy of the predictive model, and in this case, all the features were selected, which are PH, Temperature, Taste, Odor, Fat, Turbidity, Color and Grade

```
# importing the library.

from sklearn.preprocessing import LabelEncoder

le =LabelEncoder()# Create a LabelEncoder object

# The encoded values will replace the original values in the "Grade" column

df["Grade"] =le.fit_transform(df["Grade”]

x = df.drop("Grade", axis= 1)

y = df["Grade"]
```

3.5. Data Splitting and Training: The dataset is divided into 70% training and 30% testing. Machine-earning algorithms such as hierarchical clustering, KNN, K-Means, CART, and ML Class were applied to predict milk quality. They were trained to enable them to learn patterns and relationships in data and make accurate predictions or decisions.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7, random_state=42)

# Split the data into training and testing sets

# train_size=0.7 means that 70% of the data will be used for training and 30% for testing

# random_state=42 sets the random seed for reproducibility
```

4. MODEL IMPLEMENTATION AND RESULT

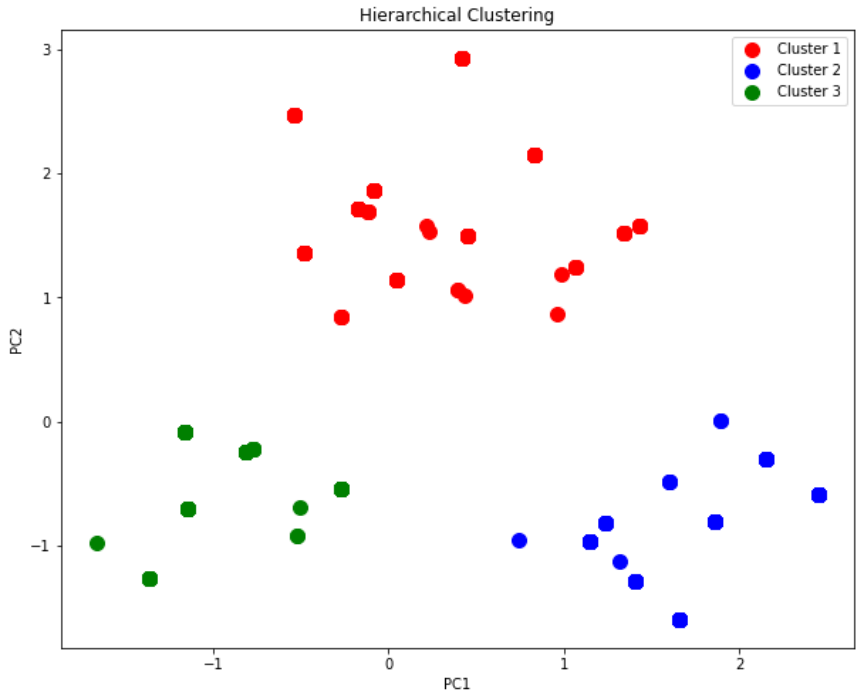
The project's implementation involved using six (6) different algorithms, each with its own characteristics and outcomes. The algorithms are described below:

i. **Hierarchical Clustering:** This algorithm was used to group similar milk samples based on their quality attributes. The principal Component Analysis (PCA) is applied to the data "X" to reduce its dimensionality to 2. The sil_coeff was used to calculate the Silhouette Coefficient and shows the average silhouette score for each cluster size in the range. The obtained outcome indicates an improved clustering performance, with a value of 0.6115. Also, the visualization shows that the clustering algorithm successfully separated the data into well-defined clusters. The number of clustering chosen was seven (7) which shows that Each data point was assigned to one of the seven clusters based on its similarity to other data points.

CODE:

```
From sklearn.cluster import KMeans #importing the library
#importing the libraries
from sklearn.cluster import AgglomerativeClustering
From sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
# Perform PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(x)
# Perform hierarchical clustering
hc = AgglomerativeClustering(n_clusters=7, affinity='euclidean', linkage='ward')
y_hc = hc.fit_predict(reduced_data)
# Determine the optimal number of clusters
def sil_coeff(no_clusters):
    # Apply your clustering algorithm of choice to the reduced data
    clusterer_1 = AgglomerativeClustering(n_clusters=no_clusters, affinity='euclidean', linkage='ward')
    clusterer_1.fit(reduced_data)
    # Predict the cluster for each data point
```

```
    labels_1 = clusterer_1.labels_
    # Calculate the Silhouette Coefficient for each sample
    silhouette_avg = silhouette_score(reduced_data, labels_1)
    print("For n_clusters =", no_clusters, "The average silhouette_score is :", silhouette_avg)
# Calculate the Silhouette Coefficient for different cluster sizes
clusters_range = range(2, 15)
for i in clusters_range:
    sil_coeff(i)
# VISUALIZATION OF HIERARCHICAL CLUSTERING
fig, ax = plt.subplots(figsize=(10, 8))
ax.scatter(reduced_data[y_hc == 0, 0], reduced_data[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
ax.scatter(reduced_data[y_hc == 1, 0], reduced_data[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
ax.scatter(reduced_data[y_hc == 2, 0], reduced_data[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
ax.set_title('Hierarchical Clustering')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend()
plt.show()
```



 # Add the cluster labels to the original data

df['cluster'] = y_hc

 # Group the data points by cluster

clusters = df.group by('cluster')

 # Calculate the mean values for each feature in each cluster

means = clusters.mean()

 # Print the mean values for each feature in each cluster

print(means)

VISUALIZATION TO SHOW THE OPTIMAL NUMBER OF CLUSTERS

importing the libraries

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette_score

from sklearn.decomposition import PCA

 # Perform PCA

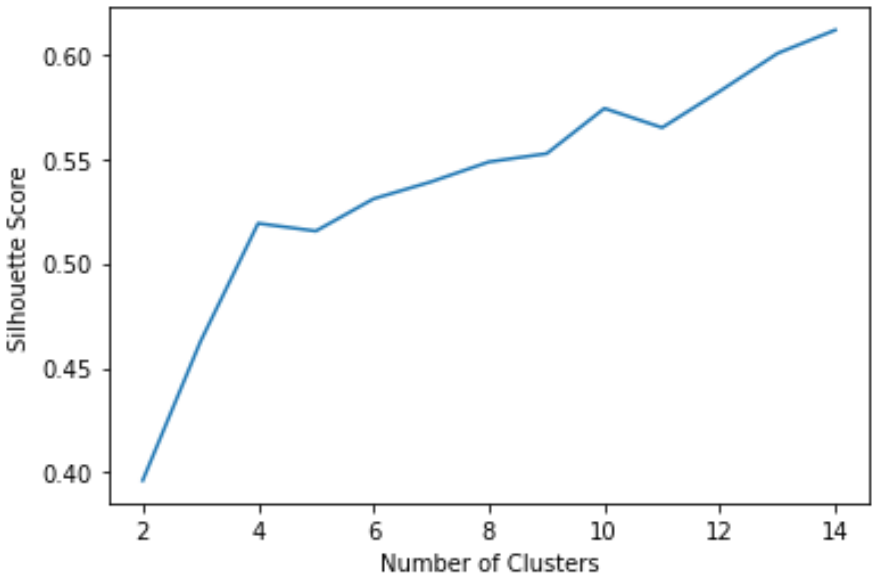
pca = PCA(n_components=2)

reduced_data = pca.fit_transform(x)

Determine the optimal number of clusters

silhouette_scores = []

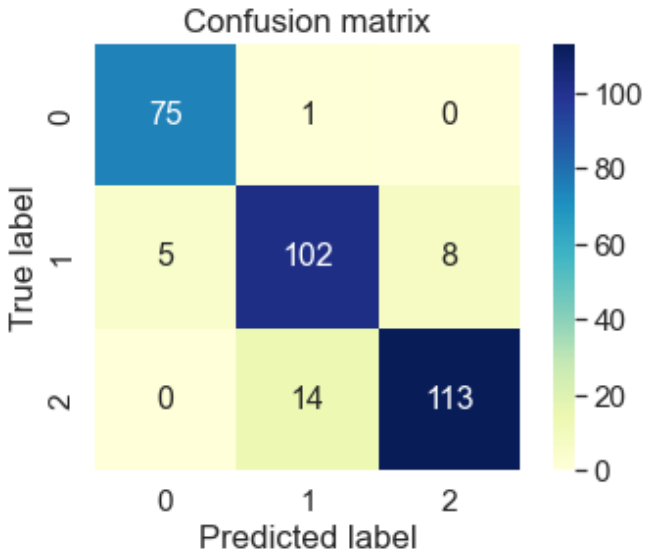
```
clusters_range = range(2, 15)
for i in clusters_range:
    # Perform hierarchical clustering
    hc = AgglomerativeClustering(n_clusters=i, affinity='euclidean', linkage='ward')
    y_hc = hc.fit_predict(reduced_data)
    # Calculate the Silhouette score
    silhouette_scores.append(silhouette_score(reduced_data, y_hc))
# Plot the Silhouette scores
plt.plot(clusters_range, silhouette_scores)
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



ii. **KNN:** In this algorithm, seven (7) nearest neighbours were considered in making predictions to avoid overfitting due to the large dataset for the model to capture localized patterns and adapt to the specific characteristics of the milk data. The K-Nearest Neighbors Classifier was trained on the milk dataset and used to predict labels for the test data, which gave an accuracy of 91.1%. The Confusion matrix provided insights into the model's performance for each class. It demonstrates that the model correctly predicted 75 instances from class 0, 102 from class 1, and 113 from class 2.

CODE:

```
#importing the libraries
from sklearn.neighbors import KNeighborsClassifier
# Create a KNN classifier object with k=7
knn = KNeighborsClassifier(n_neighbors=4)
# Train the classifier on the training set
knn.fit(x_train, y_train)
# Predict the classes of the testing set
knn_y_pred = knn.predict(x_test)
# accuracy on X_test
accuracy = knn.score(x_test, y_test)
print(accuracy)
#Importing the necessary libraries and generating a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, knn_y_pred))
from sklearn.metrics import confusion_matrix
# creating a confusion matrix
cm_knn = confusion_matrix(y_test, knn_y_pred )
#VISUALIZE THE CONFUSION MATRIX USING SEABORN
sns.set(font_scale=1.4)
sns.heatmap(cm_knn, annot=True, cmap="YlGnBu", fmt='g', annot_kws={"size": 16},
square=True)
# Set axis labels and title
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()
```



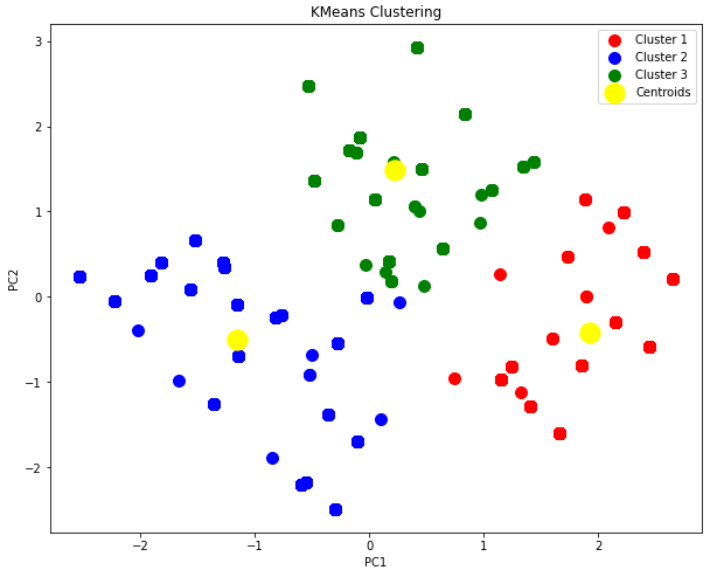
iii. **K-Means:** K-means is an unsupervised learning algorithm that partitions datasets into k clusters based on attribute similarity. It was used in the prediction of milk in order to identify natural groupings or clusters among the milk samples. The value of k was chosen to be 3 to capture the overall patterns and variations in milk quality attributes. The result shows that the average Silhouette Coefficient increases as clusters increase from 2 to 4. It reaches its peak at five and remains relatively high until around 9. After that, it fluctuates, indicating that the clusters may become less well-defined or overlapping. Also, the elbow plot shows that the WCSS, calculated for cluster sizes ranging from 1 to 14, decreases as the number of clusters increases, but the rate of decrease slows down after a certain point.

CODE:

```
from sklearn.cluster import KMeans #importng the libraries
# Perform KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(reduced_data)
# Determine the optimal number of clusters
def sil_coeff(no_clusters):
    # Apply your clustering algorithm of choice to the reduced data
    clusterer_2 = KMeans(n_clusters=no_clusters, random_state=42)
    clusterer_2.fit(reduced_data)
    # Predict the cluster for each data point
```

```
    labels_2 = clusterer_2.labels_

    # Calculate the Silhouette Coefficient for each sample
    silhouette_avg = silhouette_score(reduced_data, labels_2)
    print("For n_clusters =", no_clusters, "The average silhouette_score is :", silhouette_avg)
# Calculate Silhouette Coefficient for different cluster sizes
clusters_range = range(2, 15)
for i in clusters_range:
    sil_coeff(i)
# VISUALIZE OF THE KMEANS CLUSTERS
fig, ax = plt.subplots(figsize=(10, 8))
ax.scatter(reduced_data[y_kmeans == 0, 0], reduced_data[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
ax.scatter(reduced_data[y_kmeans == 1, 0], reduced_data[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
ax.scatter(reduced_data[y_kmeans == 2, 0], reduced_data[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
ax.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
ax.set_title('KMeans Clustering')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend()
plt.show()
```

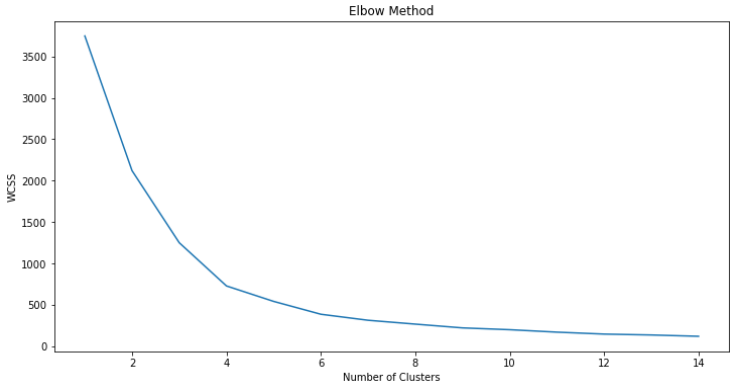


 # Calculate within-cluster sum of squares (WCSS) for different cluster sizes

```
wcss = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=42)
    kmeans.fit(reduced_data)
    wcss.append(kmeans.inertia_)
```

 # PLOT THE ELBOW GRAPH

```
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(range(1, 15), wcss)
ax.set_title('Elbow Method')
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('WCSS')
plt.show()
```



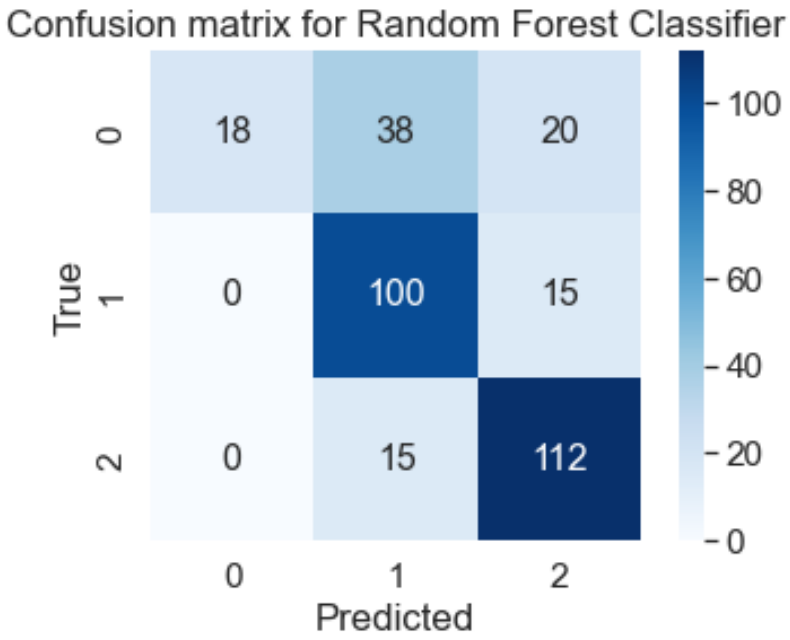
iv. **Random Forest:** It is a combination of multiple decision trees. The number estimators were chosen to be 100 to capture complex patterns of many trees. Max_depth was chosen to be 3 to prevent overfitting and have a simpler model to capture a simple relationship in the milk quality data. The Random Forest Classifier was trained and used to predict labels for the test data. The result shows that the classifier on the test data gave an accuracy of 72.3%.

CODE:

```
from sklearn.ensemble import RandomForestClassifier #importing the libraries
# Create a Random Forest Classifier with 100 trees
rfc = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=42)
# Fit the classifier to the data
rfc.fit(x_train, y_train)
# Predict the labels of the test data
rfc_y_pred = rfc.predict(x_test)
# accuracy on X_test
accuracy = rfc.score(x_test, y_test)
print(accuracy)
rfc_cm = confusion_matrix(y_test, rfc_y_pred)

# CREATE A HEATMAP TO VISUALIZE THE CONFUSION MATRIX
sns.set(font_scale=1.4)
sns.heatmap(rfc_cm , annot=True, cmap="Blues", fmt='g', annot_kws={"size": 16}, square=True)
```

```
# Set axis labels and title
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion matrix for Random Forest Classifier')
plt.show()
```

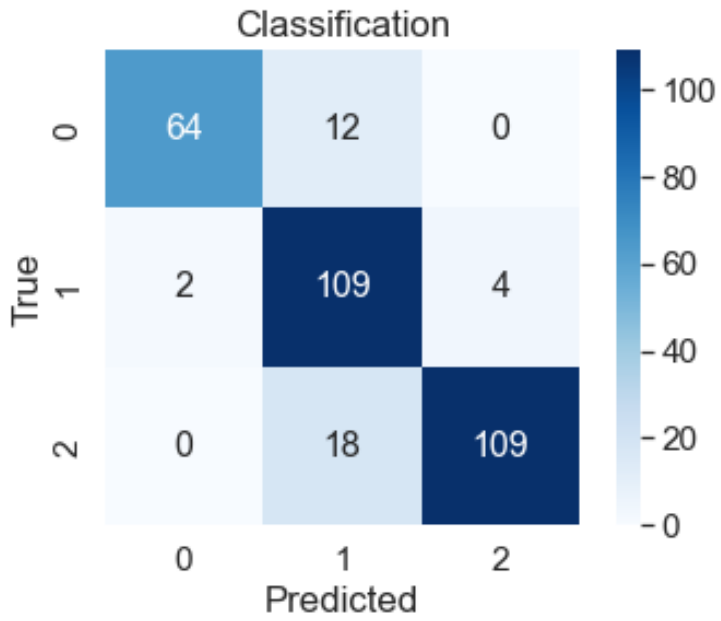


v. **CART (Classification and Regression Tree):** This decision tree algorithm is used for classification or regression tasks. It is a supervised learning algorithm that can be applied to classification and regression tasks. The classification tasks assign categorical labels to instances based on their attribute values. The algorithm splits the dataset into smaller subsets based on attribute values, creating a tree-like structure where each internal node represents a test on an attribute, each branch corresponds to an attribute value, and each leaf node represents a predicted outcome or class label. On the other hand, for regression tasks, the algorithm predicts continuous numerical values instead of discrete labels. Hence the DecisionTreeClassifier achieved an accuracy of 88.6% while DecisionTreeRegressor achieved an accuracy of 88.3% which shows that the decision tree algorithm is performing well on the given datasets.

CODE (DECISIONTREECLASSIFIER)

```
from sklearn.tree import DecisionTreeClassifier #improting the libraries
```

```
# Create a Decision Tree classifier
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
# Predict the classes for new data
clf_prediction = clf.predict(x_test)
# accuracy on X_test
accuracy = clf.score(x_test, y_test)
print(accuracy)
# print the confusion matrix and classification report
print(classification_report(y_test, clf_prediction)
clf_cm = confusion_matrix(y_test, clf_prediction)
# CREATE A HEATMAP TO VISUALIZE THE CONFUSION MATRIX
sns.set(font_scale=1.4)
sns.heatmap(clf_cm, annot=True, cmap="Blues", fmt='g', annot_kws={"size": 16}, square=True)
# Set axis labels and title
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Classification')
plt.show()
```



CODE (DECISIONTREEREGRESSOR)

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor #importing the libraries

 # Create a decision tree regressor

reg = DecisionTreeRegressor()

 # Fit the regressor to the training data

reg.fit(x_train, y_train)

 # Make predictions on the testing data

reg_pred = reg.predict(x_test)

 # accuracy on X_test

accuracy = reg.score(x_test, y_test)

print(accuracy)

from sklearn.metrics import mean_squared_error

 # Calculate mean squared error

mse = mean_squared_error(y_test, reg_pred)

print("Mean Squared Error:", mse)

vi. **ML Class:** This artificial neural network is used in milk prediction to learn complex relationships between input attributes and milk quality. The model was evaluated using a confusion matrix showing 63 true positives for the first class, 12 false negatives, one false positive, 109 true positives for the second class, five false negatives, 0 false positives for the third class, and 17 false positives. The hidden_layer_size was chosen to be 100 because It will help capture the complex relationships in the milk dataset. The max_iter was chosen to be 1000 to allow the network more training time and potentially improve its performance.

vii. **CODE:**

#importing the libraries

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score

 # Create an MLP classifier

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)

 # Train the MLP classifier

mlp.fit(x_train, y_train)

 # Predict on the test set

mlp_y_pred = mlp.predict(x_test)

 # Print the confusion matrix

cm = confusion_matrix(y_test, mlp_y_pred)

print("Confusion Matrix:")

print(cm)

 # Calculate accuracy

accuracy = mlp.score(x_test, y_test)

print("Accuracy:", accuracy)

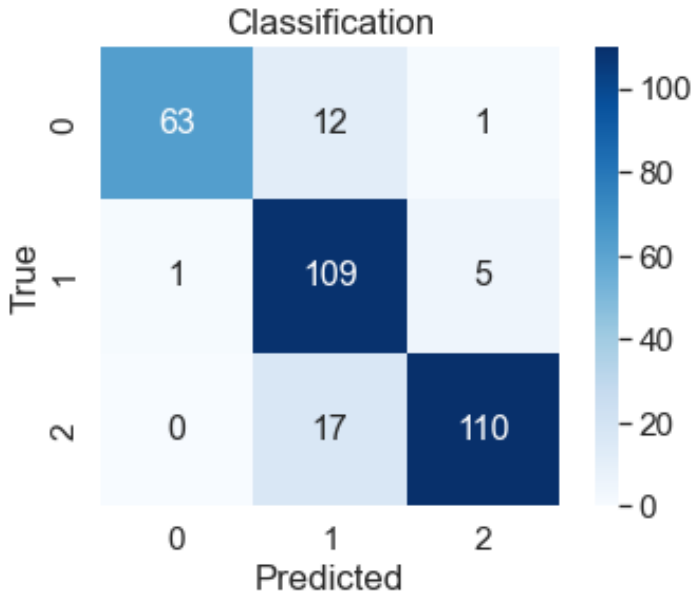
mlp_cm = confusion_matrix(y_test, mlp_y_pred)

 # CREATE A HEATMAP TO VISUALIZE THE CONFUSION MATRIX

sns.set(font_scale=1.4)

```
sns.heatmap(mlp_cm, annot=True, cmap="Blues", fmt='g', annot_kws={"size": 16}, square=True)

# Set axis labels and title
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Classification')
plt.show()
```



5.0. RESULT

There were eight attributes from the dataset related to milk quality: pH, Temperature, Taste, Odor, Fat, Turbidity, Color, and Grade. Exploratory data analysis was performed to gain insights into the dataset, uncover patterns, distributions, outliers, and correlations among the variables. Several machine learning algorithms were implemented and evaluated using the dataset. The accuracy of each algorithm was assessed using the confusion matrix, precision, recall, and F1-score. Among the algorithms, the K-Nearest Neighbors (KNN) algorithm achieved the highest accuracy of 91.1% when predicting milk quality, indicating that the KNN algorithm performed well in capturing the patterns and characteristics of the milk data.

6.0. CONCLUSION

In conclusion, this project aimed to predict milk quality using various machine learning algorithms such as Hierarchical Clustering, K-Nearest Neighbors (KNN), K-Means, Random Forest, CART (Classification and Regression Tree), and ML Class (Artificial Neural Network). The dataset was collected from Kaggle and consisted of eight attributes related to milk quality (PH, Temperature, Taste, Odor, Fat, Turbidity, Color and Grade). The results demonstrated that the K-Nearest Neighbors (KNN) algorithm outperformed other algorithms, achieving an accuracy of 91.1%. Future work can be done by exploring ensemble methods further to improve the accuracy of the milk quality prediction model.