PREDICTING QUALITY OF WINE

CLASSIFICATION METHOD

GROUP 10

D. Niharika

Shivam Singh

Syed Suhel

Mihir Razvi

Bhavans Vivekananda College

B.sc Hons Data Science

Steps involved in Machine Learning Analysis

01 DATA PREPROCESSING

02. EXPLORATORY DATA ANALYSIS

03 TRAINING THE MODEL

04 DETERMINING MODEL ACCURACY

05 COMPARING THE RESULTS WITH OTHER MODELS

06 CONCLUSION AND INSIGHTS

ABOUT THE DATA SET

The red wine dataset typically refers to a dataset specially focused on red wine, providing information on attributes like types of acids, sulfur, pH, alcohol, quality. Such datasets are often utilized for analysis and modeling in the field of Data Science.

OBJECTIVE

- Understand the data set and clean up
- Build classification models to predict the wine quality
- Also fine tune the hyperparameters
- Comparing the evaluation metrics of various classification algorithms

THE PATH

Implenting multiple machine learning models to fit the best model for the dataset

Data and Data Quality Check

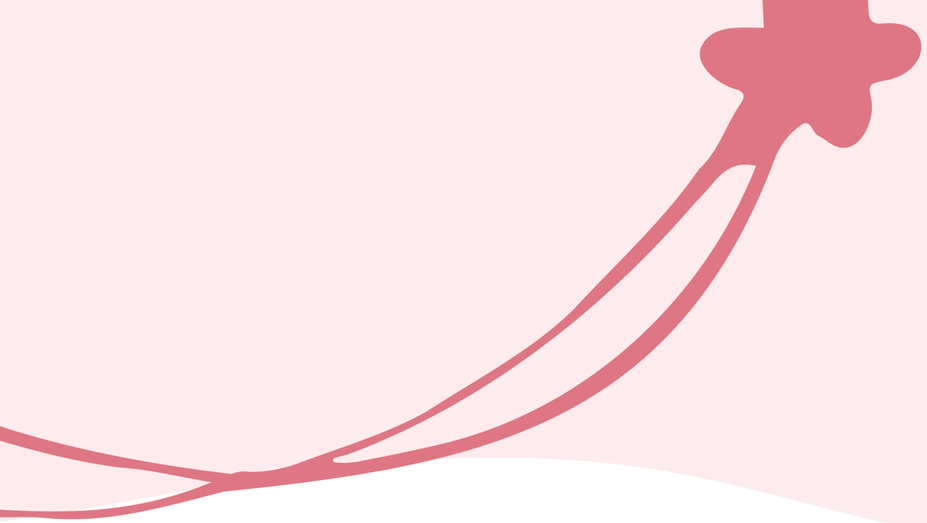
Data

There are 1599 rows and 12 columns present in the dataset. There are no missing values or the null values in the dataset.

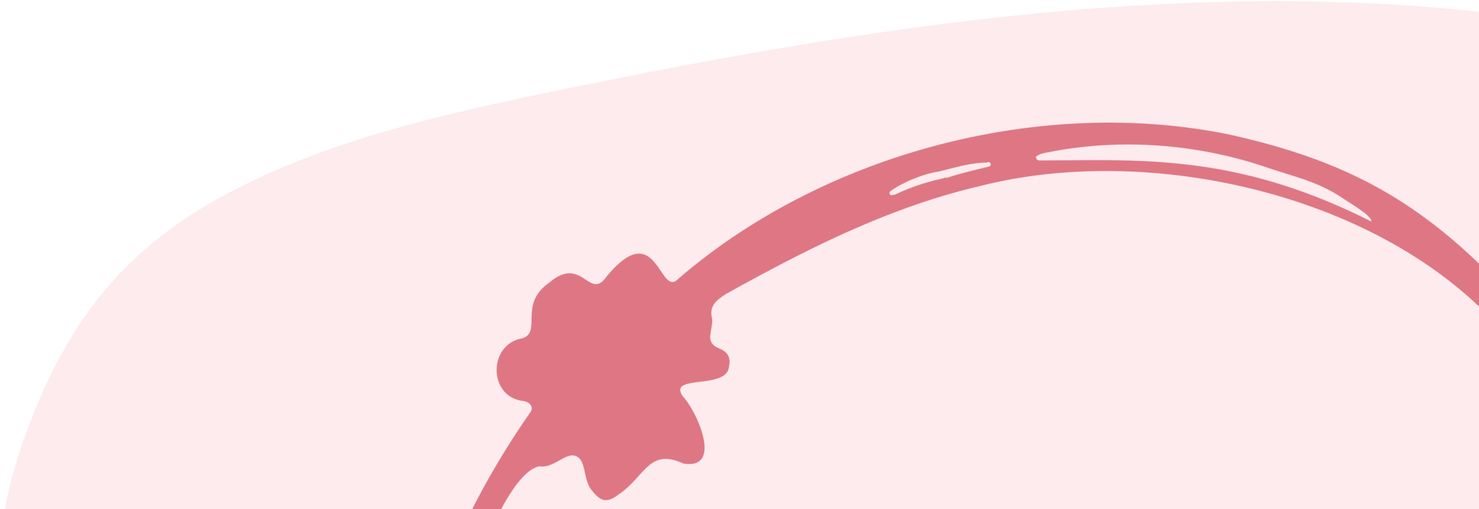
Attribute Description

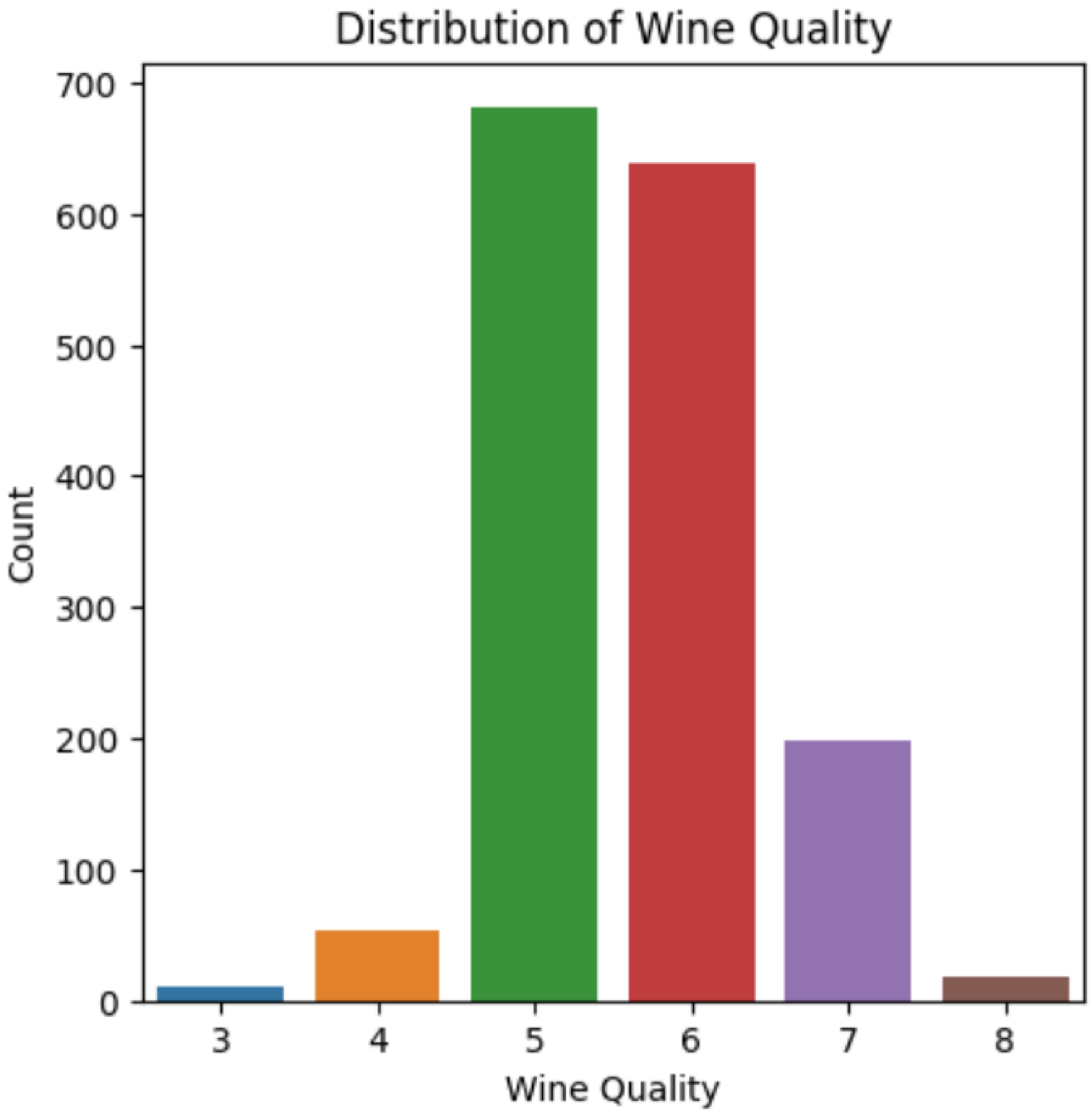
Attribute	Description
fixed acidity	Acidity is a fundamental property of wine, imparting sourness and resistance to microbial infection. Fixed acidity is the no. of grams of tartaric acid per dm^3
volatile acidity	Wine spoilage is legally defined by volatile acidity which is calculated as no. of grams of acetic acid per dm^3 of wine
citric acid	It is the no. of grams of citric acid per dm^3 of wine
residual sugar	Residual sugar refers to the sugar remaining after fermentation stops. Given as no. of grams per dm^3
chlorides	It is the no. of grams of sodium chloride per dm^3 of wine
free sulfur dioxide	It is the no. of grams of free sulfites per dm3 of wine

Attribute	Description
total sulfur dioxide	It is the no. of grams of total sulfite (free sulfite+ bound sulfite) in per dm^3 of wine
density	It gives the density of the wine in grams per cm^3
pH	It gives the pH of the wines. pH is used to measure ripeness in relation to acidity
sulphates	It gives the no. of grams of potassium sulphate per dm^3 of wine
alcohol	It gives the volume of alcohol in percentage
quality	This is the target variable. Here the wine is rated from 1-10 based on the quality

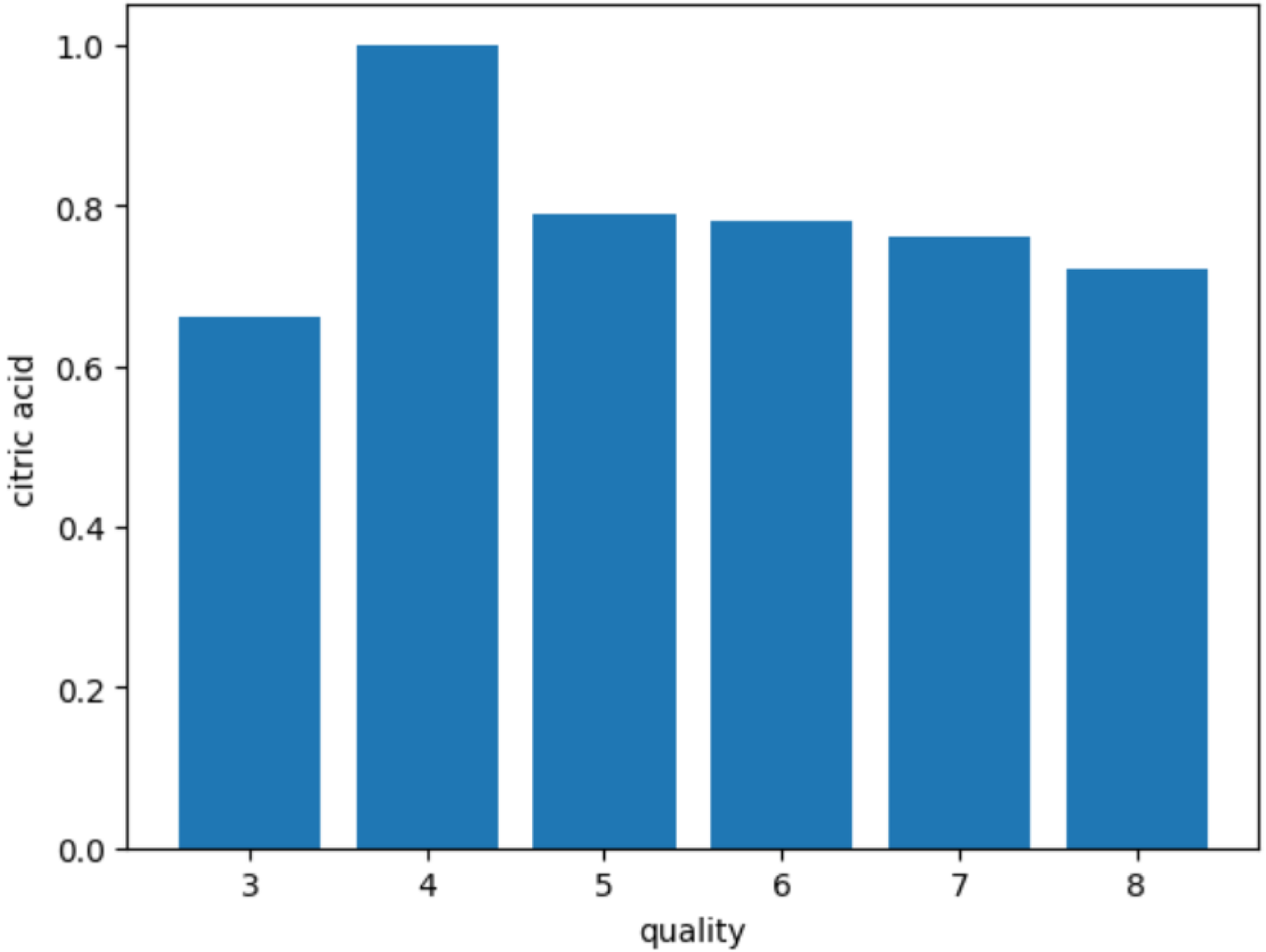


EXPLORATORY DATA ANALYSIS

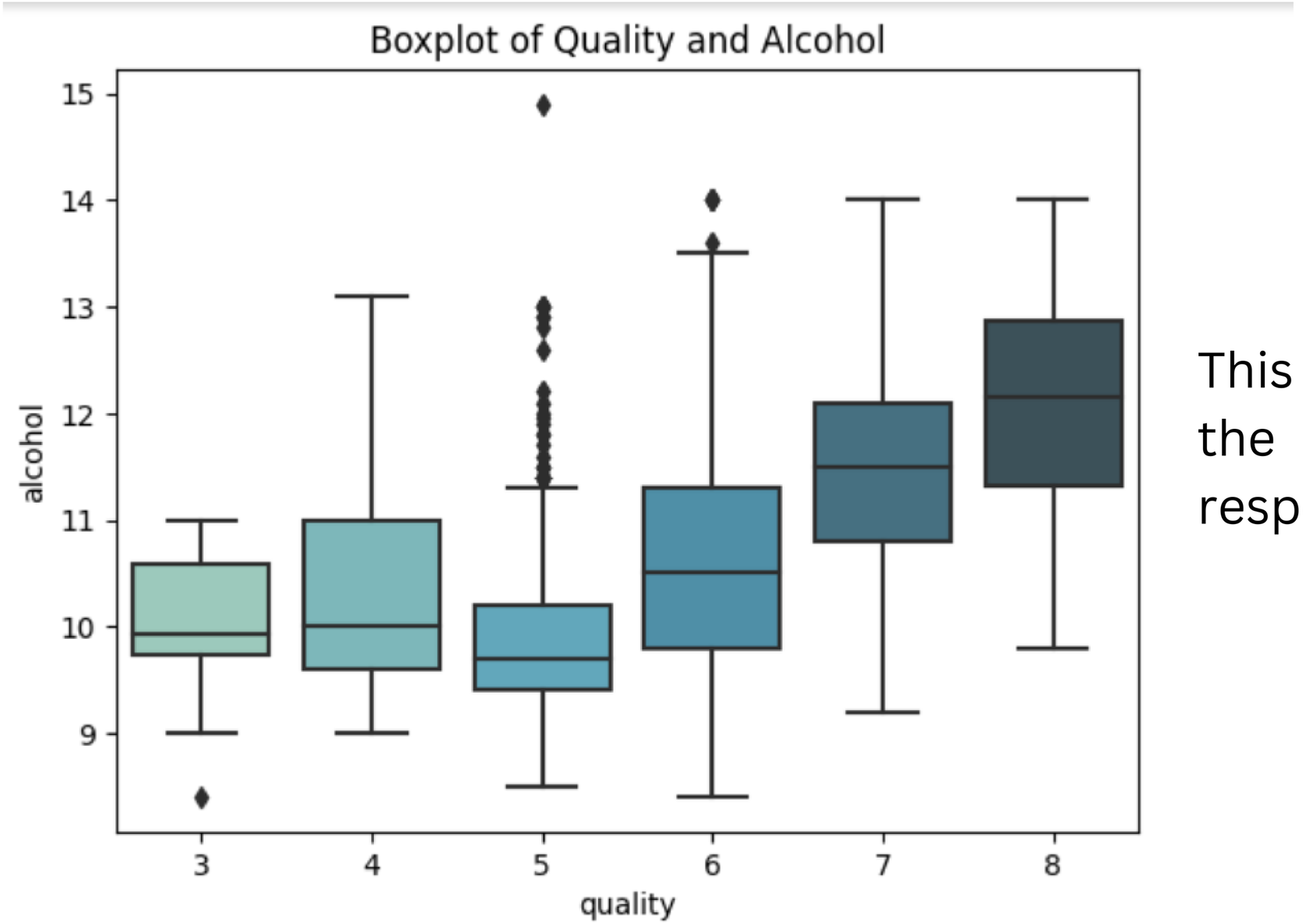




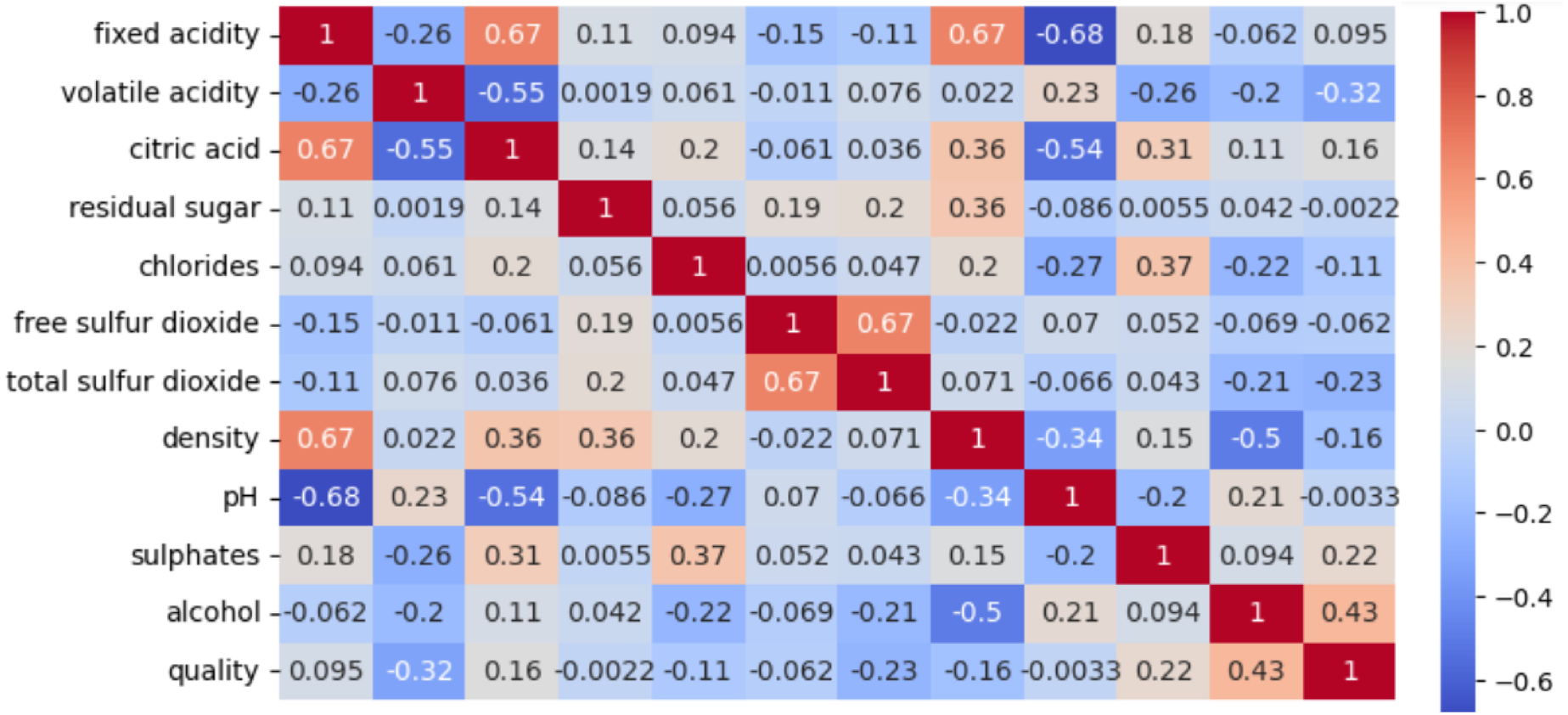
This bar graph represents the no. of instances in each quality.



Bar plot represents the level of citric acid present in respective quality



This box plot represents the alchol present in the respective quality



This shows the relationship between different variables. Dark colors represents the strong relationships and the light colors represents the weak relationships.

VARIANCE INFLATION FACTOR

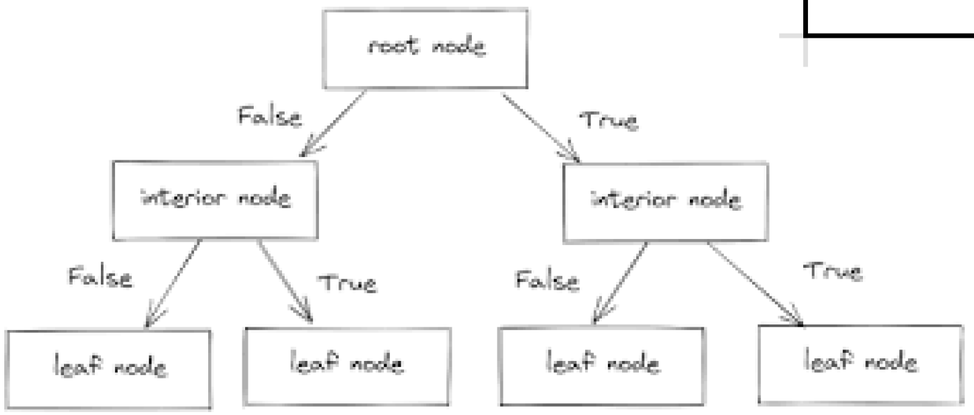
	variables	VIF
0	fixed acidity	74.452265
1	volatile acidity	17.060026
2	citric acid	9.183495
3	residual sugar	4.662992
4	chlorides	6.554877
5	free sulfur dioxide	6.442682
6	total sulfur dioxide	6.519699
7	density	1479.287209
8	pH	1070.967685
9	sulphates	21.590621
10	alcohol	124.394866
11	type_encoded	NaN

ALGORITHMS USED IN MODELING THE DATASET

- Decision Tree Classifier
- Random Forest Classifier
- Support Vector Classifier
- Naive Bayes
- K_Nearest Neighborst
- Bagging Classifier
- XG Boost
- Gradient Boostingt

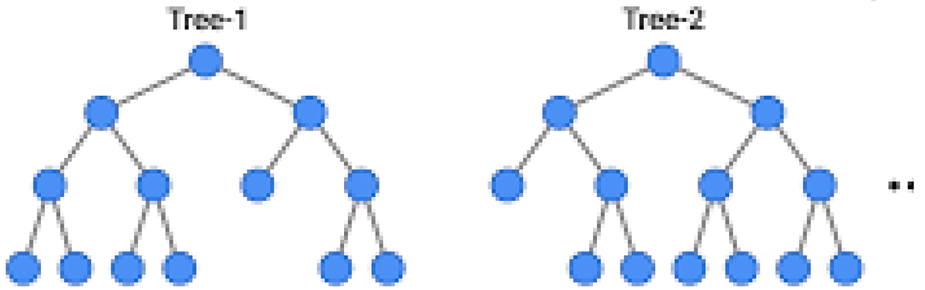
DECISION TREE CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.83
70-30	0.73
75-25	0.8
65-35	0.78



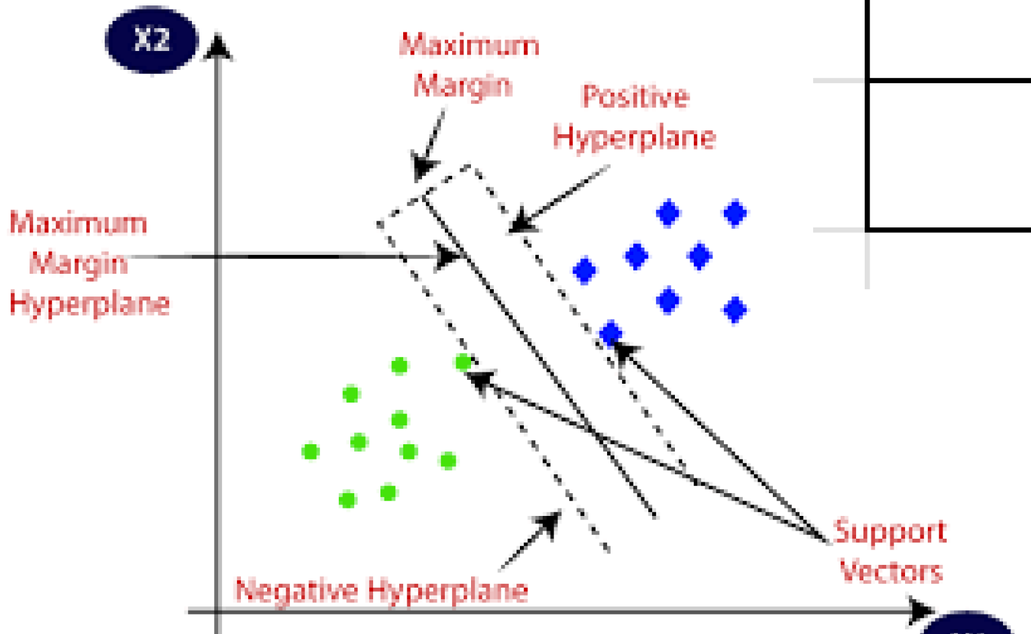
RANDOM FOREST CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.8
70-30	0.79
75-25	0.81
65-35	0.78

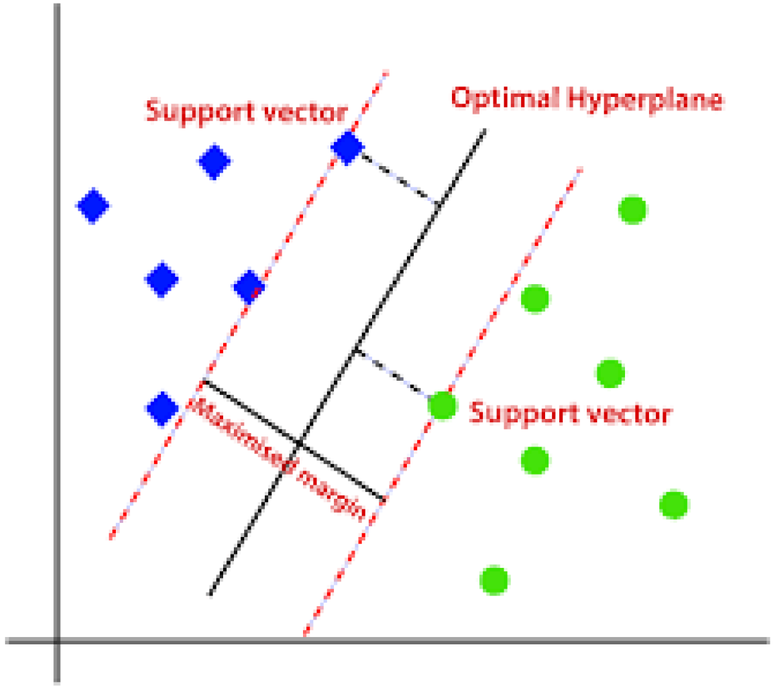


SUPPORT VECTOR CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.73
70-30	0.72
75-25	0.71
65-35	0.72



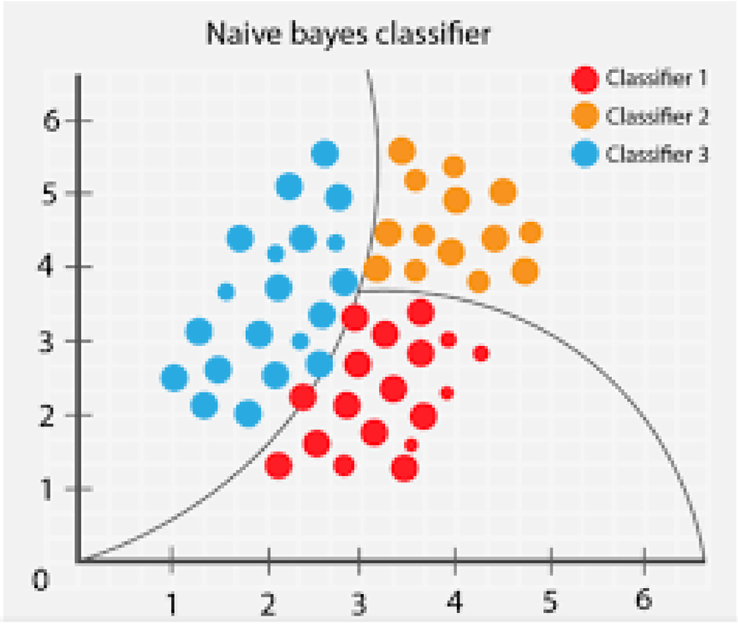
K_ NEAREST NEIGHBOR CLASSIFIER



Train-Test-Split	Accuracy
80-20	0.69
70-30	0.66
75-25	0.67
65-35	0.72

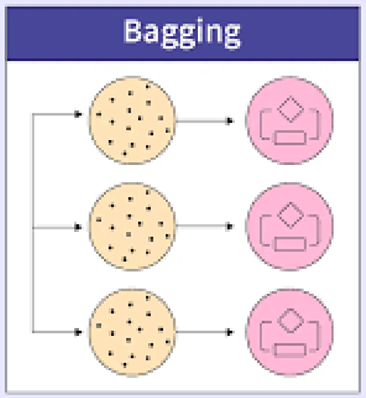
NAIVE BAYES CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.71
70-30	0.72
75-25	0.73
65-35	0.72



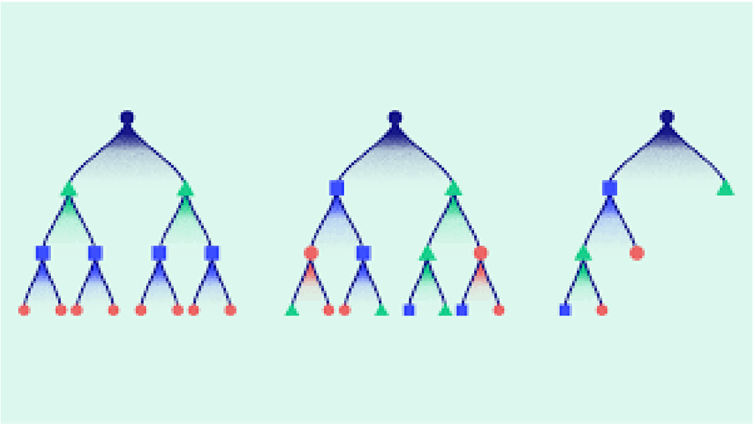
BAGGING CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.78
70-30	0.77
75-25	0.81
65-35	0.76



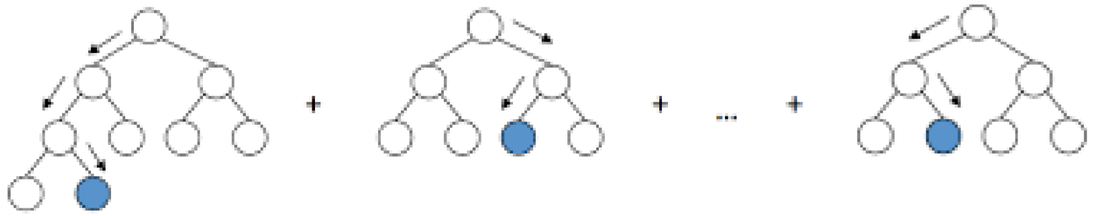
XG BOOST CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.8
70-30	0.78
75-25	0.77
65-35	0.76

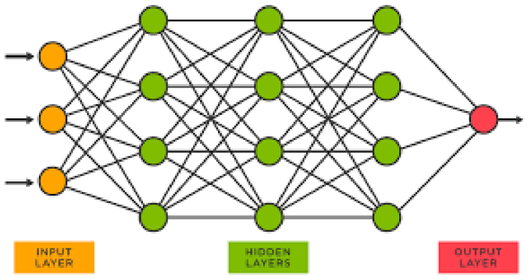


GRADIENT BOOSTING CLASSIFIER

Train-Test-Split	Accuracy
80-20	0.72
70-30	0.76
75-25	0.74
65-35	0.75



NEURAL NETWORKS



Train-Test Ratio	Architectures	Optimizer	Epochs	Accuracy
80-20	64-32-1	Adam	10	76.24
70-30	65-30-1	Adam	25	74.79
75-25	75-38-1	Adam	30	74.25
65-35	100-50-1	Adam	8	75.35

TRAIN AND TEST RATIO

Algorithm	80-20	70-30	75-25	65-35
Decision Tree Classifier	0.83	0.73	0.8	0.79
Random Forest Classifier	0.8	0.79	0.81	0.78
Support Vector Classifier	0.78	0.72	0.71	0.72
Naive Bayes	0.71	0.72	0.73	0.72
K_Nearest Neighbors	0.69	0.66	0.67	0.72
Bagging Classifier	0.78	0.77	0.81	0.76
XG Boost Classifier	0.8	0.78	0.77	0.76
Gradient Boosting	0.72	0.76	0.74	0.75
Neural Networks	76.24	74.79	74.25	75.35

COMPARISION OF ALGORITHMS

Algorithms	Accuracy
Decision Tree Classifier	0.83
Random Forest Classifier	0.81
Support Vector Classifier	0.73
Naive Bayes	0.73
K_Nearest Neighbors	0.72
Bagging Classifier	0.81
XG Boost Classifier	0.8
Gradient Boosting	0.76
Neural Networks	76.24

with the ratio of 80-20

CONCLUSION

- For the Red Wine Quality dataset, we performed nine types of machine learning algorithms : Decision Tree, Random Forest , K_Nearest Neighbors, Support Vector Machine, Naive Bayes, Bagging, XG Boost, Gradient Boosting and Neural Networks.
- From all the analysis and implementations of the algorithms, we found the best results in “Decision Tree”.
- Decision Tree Classifier performed best on this data set with an accuracy of 0.83.

APPLICATIONS

- These results will be used by the wine manufactures to improve the quality of future wines.
- Results can be used to make wine selection guides for wine magazines.
- Results can be used by consumers for wine selection.

Thank You!

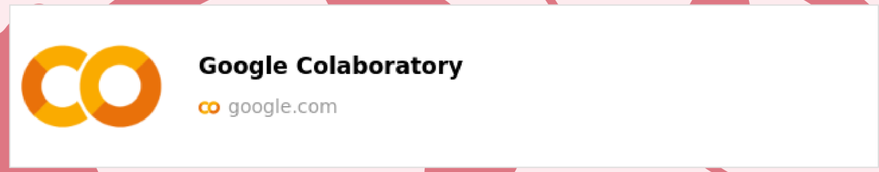
PRESENTED BY:

D. Niharika

Shivam Singh

Suhel

Mihir



APPENDIX

DECISION TREE CLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

```
y_pred=model.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.785
```

RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
```

```
X_train.shape,X_test.shape
```

```
model = RandomForestClassifier()
model.fit(X_train,y_train)
```

```
y_pred=model.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.785
```

SUPPORT VECTOR CLASSIFIER

```
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
model=svm.SVC(kernel='linear')
model.fit(X_train,y_train)
svm.SVC(kernel='linear')
```

```
y_pred=model.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.715
```

NAIVE BAYES

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
gnb=GaussianNB()
gnb.fit(X_train,y_train)
```

```
y_pred=gnb.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.72
```

K_NEAREST NEIGHBORS

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
model=KNeighborsClassifier(n_neighbors=10)
model.fit(X_train,y_train)
KNeighborsClassifier()
```

```
y_pred=model.predict(X_test)
y_pred
```

```
df=pd.DataFrame({'Actual':y_test,'Predict':y_pred})
df
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.6775
```

BAGGING CLASSIFIER

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
model=DecisionTreeClassifier()
```

```
num_classifiers = 10
```

```
bagging_classifier = BaggingClassifier(model, n_estimators=num_classifiers, random_state=100)
```

```
bagging_classifier=bagging_classifier.fit(X_train, y_train)
```

```
y_pred=bagging_classifier.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.815
```

XG BOOST CLASSIFIER

```
from xgboost import XGBClassifier
xgboost= XGBClassifier(n_estimators=1000, learning_rate=0.05)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
xgboost.fit(X_train, y_train)
```

```
y_pred = xgboost.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.7775
```

GRADIENT BOOSTING

```
from sklearn.ensemble import GradientBoostingClassifier
gbr = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
gbr.fit(X_train, y_train)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
X_train.shape,X_test.shape
```

```
y_pred = gbr.predict(X_test)
y_pred
```

```
print("Accuracy.",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy. 0.7475
```

NEURAL NETWORKS

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```python
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
loss, accuracy=model.evaluate(X_test,y_test)
print(f"Accuracy: {accuracy*100:2f}%")
```

```
Epoch 1/10
33/33 [==============================] - 4s 18ms/step - loss: 0.6055 - accuracy: 0.7103 - val_loss: 0.5642 - val_accuracy: 0.7268
Epoch 2/10
33/33 [==============================] - 0s 7ms/step - loss: 0.5257 - accuracy: 0.7421 - val_loss: 0.5493 - val_accuracy: 0.7107
Epoch 3/10
33/33 [==============================] - 0s 6ms/step - loss: 0.5049 - accuracy: 0.7584 - val_loss: 0.5275 - val_accuracy: 0.7286
Epoch 4/10
33/33 [==============================] - 0s 7ms/step - loss: 0.4886 - accuracy: 0.7671 - val_loss: 0.5220 - val_accuracy: 0.7286
Epoch 5/10
33/33 [==============================] - 0s 10ms/step - loss: 0.4798 - accuracy: 0.7738 - val_loss: 0.5152 - val_accuracy: 0.7304
Epoch 6/10
33/33 [==============================] - 0s 12ms/step - loss: 0.4748 - accuracy: 0.7767 - val_loss: 0.5122 - val_accuracy: 0.7393
Epoch 7/10
33/33 [==============================] - 0s 12ms/step - loss: 0.4705 - accuracy: 0.7786 - val_loss: 0.5065 - val_accuracy: 0.7411
Epoch 8/10
33/33 [==============================] - 0s 6ms/step - loss: 0.4627 - accuracy: 0.7892 - val_loss: 0.5093 - val_accuracy: 0.7375
Epoch 9/10
33/33 [==============================] - 0s 7ms/step - loss: 0.4590 - accuracy: 0.7844 - val_loss: 0.5168 - val_accuracy: 0.7375
Epoch 10/10
33/33 [==============================] - 0s 5ms/step - loss: 0.4527 - accuracy: 0.7834 - val_loss: 0.5062 - val_accuracy: 0.7321
18/18 [==============================] - 0s 3ms/step - loss: 0.5062 - accuracy: 0.7321
Accuracy: 73.214287%
```