#necessary libraries

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

wine\_file = pd.read\_csv('20101539\_Umme Abira Azmary\_CSE422\_11\_Assignment05\_Summer2024.csv') #loading the dataset

print(wine\_file.shape) #checking the shape(row x col) of the dataset

print(wine\_file.columns) #checking the features

wine\_file.isnull().sum() #checking the total number of null values in features.

#Removing null values

#It is observed that the number of null values is relatively small; therefore, removing the rows containing null values is preferable to removing columns. As a result, the .dropna() method should be applied.

print("Before removing null values:", wine\_file.shape)

wine\_file = wine\_file.dropna(axis = 0, subset = ['fixed acidity']) #removing null rows for the 'fixed acidity' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['volatile acidity']) #removing null rows for the 'volatile acidity' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['citric acid']) #removing null rows for the 'critic acid' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['residual sugar']) #removing null rows for the 'residual sugar' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['chlorides']) #removing null rows for the 'chlorides' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['pH']) #removing null rows for the 'pH' feature.

wine\_file = wine\_file.dropna(axis = 0, subset = ['sulphates']) #removing null rows for the 'sulphates' feature.

print("After removing null values:", wine\_file.shape)

wine\_file.isnull().sum()

# Removing Duplicates

# Duplicate values can distort the representation of data. These duplicates may skew the data, leading to inaccurate or misleading results in the model

dup = wine\_file.duplicated() # identifying duplicate rows

sum\_dup = dup.sum() #total number of duplicate rows

print(f"Number of duplicate values present are:",sum\_dup)

wine\_file2 = wine\_file.drop\_duplicates() #removes duplicate rows

print('Duplicates removed!')

dup2 = wine\_file2.duplicated()

sum\_dup2 = dup2.sum()

print(f"Number of duplicate values present are:",sum\_dup2)

wine\_file2['type'].unique() # retrieves the unique values from the 'type' column

# As the retrieval values are Nominal, instead of scaling, encoding methods need to be applied. This is why, applying a one-hot-encoding method.

wine\_file2 = pd.get\_dummies(wine\_file2, columns=['type'], drop\_first=True) #creating dummy variables for categorical data

wine\_file2.head()

wine\_file2['type\_white'].unique() #Instead of having non-ordered 'type' values, the values are now represented with conditions or states.

# Feature Scaling

# Separating the target column from the features as scaling brings all the input features to a common scale so that feature contributes equally to the learning. The target should not be affected by this as it is what the model tries to predict.

target\_val = wine\_file2['quality']

vals = wine\_file2.drop(columns=['quality'])

print(vals.describe())

# Observing the dataset, it is vigilant that residual sugar, total sulfur dioxide, chlorides columns contain a significant variance and different scale units. So, scaling these features may help to understand these features more.

sel\_cols = ['residual sugar','chlorides', 'total sulfur dioxide', 'free sulfur dioxide']

scaler = StandardScaler() #as the dataset contains features of different scales with high variance.

vals[sel\_cols] = scaler.fit\_transform(vals[sel\_cols])

print(vals.describe())

wine\_file3 = pd.concat([vals, target\_val], axis=1)

print(wine\_file3.head())

wine\_file3.shape

#Correlation helps to understand how much a certain feature is similar to other features. If two features display high correlation, they affect the model similarly. So, one of it can be removed while considering it redundant.

corr\_matx = wine\_file3.corr() #applying correlation

print(corr\_matx)

import seaborn as sns

sns.heatmap(corr\_matx, cmap = 'YlGnBu')

# Dropping one of the highly correlated values based on the generated heatmap

wine\_file4 = wine\_file3.drop('free sulfur dioxide', axis=1)

# Checking the dataset after removing correlated columns

print(wine\_file4.head())

corr\_matx2 = wine\_file4.corr() #applying correlation

print(corr\_matx2)

# Working with Linear Regression

from sklearn.model\_selection import train\_test\_split

X = wine\_file4.drop(columns=['quality']) # features

y = wine\_file4['quality'] # target

# Split the dataset: 80% training, 20% testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

X\_train.head()

y\_train.head()

#Linear Regression

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

v1 = LinearRegression()

v1.fit(X\_train,y\_train)

yPredict = v1.predict(X\_test)

yPredict

# R-squared value explains how much dependent variance is expalined by the independent variables

r2\_val = r2\_score(y\_test, yPredict)

print(f'R-squared:', r2\_val)

#Mean-Square-Error indicates the average squared difference between the actual and predicted values

mse\_val = mean\_squared\_error(y\_test, yPredict)

print(f'Mean Squared Error:', mse\_val)

import matplotlib.pyplot as plt

import numpy as np

# Sorting by indices to visualize better

y\_test\_sort = np.sort(y\_test)

y\_pred\_sort = np.sort(yPredict)

plt.figure(figsize=(10, 6))

plt.plot(y\_test\_sort, label='Actual (y\_test)', color='blue')

plt.plot(y\_pred\_sort, label='Predicted (y\_Prediction)', color='green')

plt.xlabel('Index (Order)')

plt.ylabel('Quantity')

plt.title('Actual vs Predicted Quantities')

plt.legend()

plt.show()

# Logistic Regression

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score

wine\_file4.head()

#used 6 as a threshold to create a binary classification for the wine quality. This value divides the target into two groups: wines with a quality higher than 6 are considered high-quality, and the rest are low/average-quality.

wine\_file4['quality\_class'] = np.where(wine\_file4['quality'] > 6, 1, 0)

X = wine\_file4.drop(['quality', 'quality\_class'], axis=1)

y = wine\_file4['quality\_class']

X.head()

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X, y, stratify = y, test\_size=0.3, random\_state=42)

log\_reg = LogisticRegression(max\_iter = 1000)

log\_reg.fit(X\_train1, y\_train1)

y\_pred1 = log\_reg.predict(X\_test1)

#print(y\_pred1)

accuracy\_val = accuracy\_score(y\_test1, y\_pred1)

precision\_val = precision\_score(y\_test1, y\_pred1)

recall\_val = recall\_score(y\_test1, y\_pred1)

f1\_score = f1\_score(y\_test1, y\_pred1)

print(f'Accuracy: {accuracy\_val \* 100:.3f}%')

print(f'Precision: {precision\_val:.3f}')

print(f'Recall: {recall\_val:.3f}')

print(f'F1 Score: {f1\_score:.3f}')

print(wine\_file4['quality'].value\_counts())

#From the result, it is visible that, even if the accuracy value is good, F1\_score is not good. This indicates a huge class imbalance,handling that can help to attain a better F1 score.