ENGR 1330 – Computational Thinking and Data Science

Red Wine Quality Final Project - Background

In this project, a dataset related to red wine samples from the northwest region of Portugal will be analyzed. The quality of a red wine that is determined via a sensory test is dependent on many different physicochemical properties, namely, fixed acidity, volatile acidity, pH value, density, etc. A file named 'winequality-red.csv' contains information about different varieties of red wine and their quality that depends on several physicochemical properties like the ones mentioned above. Specifically, in the dataset, there is a quality score (QS) ranging from 3 to 8 that is given to each variety of red wine depending on 11 different properties. For this project, consider that a good wine is one with a quality score of QS ≥ 6 and a bad wine is one with a quality score of QS ≤ 5. The objective of this problem is to classify whether the wine is good or bad depending on the 11 different properties that are in the dataset.

Required Tasks:

(a) Literature review: 1) P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009. [Get the research paper from Web of Science at TTU].

(b) Data acquisition: 1) Get the dataset required for this project from the following Kaggle website: https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009

(c) Exploratory data analysis 1) Perform exploratory data analysis (getting information about the dataset, making plots, etc.) 2) Modify the dataset as needed for performing the analysis

(d) Classification model 1) Implement a classification algorithm from scratch as well as using the data science library to classify good wines and bad wines 2) Evaluate the model by computing the necessary evaluation metrics from scratch as well as using the data science library

Database Acquisition

- Get the database from the zip file from BlackBoard or Kaggle website: https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009
- Supply links to any additional databases discovered during the literature research

Exploratory Data Analysis

- Describe (in words) the database.
- Reformat as needed (column headings perhaps) the database for subsequent analysis.

Model Building

- Build data model
- Assess data model quality
- Build the input data interface for using the model

Documentation

- Training and Project management video on how to use your tool, and demonstrate the tool(s) as they are run
- Interim report (see deliverables below); this document must be rendered as a .pdf.
- Final ipynb file (see deliverables below)

Deliverables:

Part 1 Interim Report (due April 29th):

A report that briefly describes the project.Use the Interim Report Template in BlackBoard.

- Break down each task into manageable subtasks and describe how you intend to solve the subtasks and how you will test each task. (Perhaps make a simple Gantt Chart) or list of meeting times.
- Address the responsibilities of each team member for tasks completed and tasks to be completed until the end of the semester. (Perhaps make explicit subtask assignments)

Part 2 Final Report (due May 7th):

- A well-documented JupyterLab (using a python kernel), use markdown cells and commenting for explanations and text.
- A how-to video demonstrating performance and description of problems that you were not able to solve and also talk about project management such as who did what. Active participation of every single group member is mandatory in the presentation.
- A final peer evaluation report, where each group member should rate the participation and contribution of the other members.

Above items can reside in a single video; but structure the video into the two parts; use an obvious transition when moving from "how to ..." into the project management portion. Keep the total video length to less than 10 minutes; submit as an *unlisted* YouTube video, and just supply the link (someone on each team is likely to have a YouTube creator account). Keep in mind a 10 minute video can approach 100MB file size before compression, so it won't upload to Blackboard and cannot be emailed.

In [146…
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
# Loading the image
img = mpimg.imread('redwine.png')
# Set the figure size
plt.figure(figsize=(10, 30))  # Adjust the width and height as needed
# Displaying the image
plt.imshow(img)
plt.axis('off')  # Turn off axis
plt.show()
```

In [147…

```
#The Libraries needed for this Project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import math
```

In [148…

```
#Read redwinequality.csv file and putting in 'Redwine' variable
Redwine=pd.read_csv("redwinequality.csv")
Redwine.head(17)
```

Out[148]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.9986	3.17	0.93	
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	0.75	

In [149…

```
#Adds a fresh column named 'Quality Check' in the DataFrame. It assigns a bina
Redwine["QualityCheck"]=(Redwine['quality']>=6).astype('uint8')
#Printing first 17 rows of dataframe
Redwine.head(17)
```

Out[149]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.9986	3.17	0.93	
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	0.75	

In [150…
```
#Check RedWine.Shape()
Redwine.shape
```

Out[150]:
```
(1599, 13)
```

In [151…
```
#Obtaining fundamental statistical insights from the 'Redwine' dataset.
Redwine.describe()
```

Out[151]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	tota
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.

In [152…
```
#Showing the specifics of the 'Redwine' dataset.
Redwine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
 12  QualityCheck          1599 non-null   uint8  
dtypes: float64(11), int64(1), uint8(1)
memory usage: 151.6 KB
```

In [153…
```
#Determining the quantity of missing values present in the 'Redwine' dataset.
Redwine.isnull()
```

Out[153]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	
1594	False	False	False	False	False	False	False	False	False	False	
1595	False	False	False	False	False	False	False	False	False	False	
1596	False	False	False	False	False	False	False	False	False	False	
1597	False	False	False	False	False	False	False	False	False	False	
1598	False	False	False	False	False	False	False	False	False	False	

1599 rows × 13 columns

In [154…
```
#Examining each column in the DataFrame "Redwine" for any missing values and ag
Redwine.isnull().sum()
```

Out[154]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
QualityCheck            0
dtype: int64
```

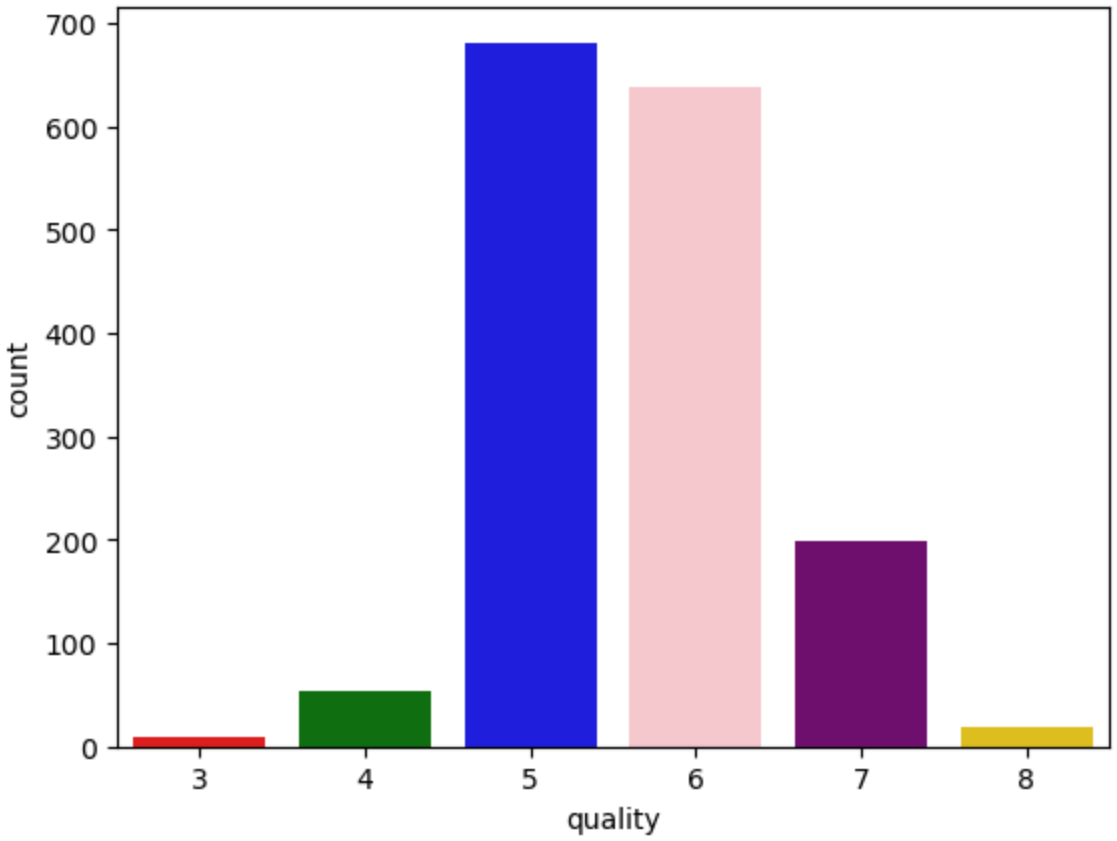
In [155…
```
#This code produces a pairplot utilizing the Seaborn library for the DataFrame
sns.pairplot(Redwine, hue='QualityCheck')
```

```
/Users/samirchand/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[155]:
```
<seaborn.axisgrid.PairGrid at 0x29ee64710>
```

In [156…
```
#Showing total count of quality scores of 'Redwine'
sns.countplot(x=Redwine['quality'], palette=['red', 'green', 'blue', 'pink', '
```

Out[156]:
```
<Axes: xlabel='quality', ylabel='count'>
```



In [157…
```
#Create a copy of 'Redwine' and put it in'Wine_Equality'
Wine_Equality=Redwine.copy()
#Playing with the 'quality' column from the 'Wine_Equality' variable
Wine_Equality=Wine_Equality.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,12]]
#Printing first 17 row for 'Wine_Equality'
Wine_Equality.head(17)
```

Out[157]:

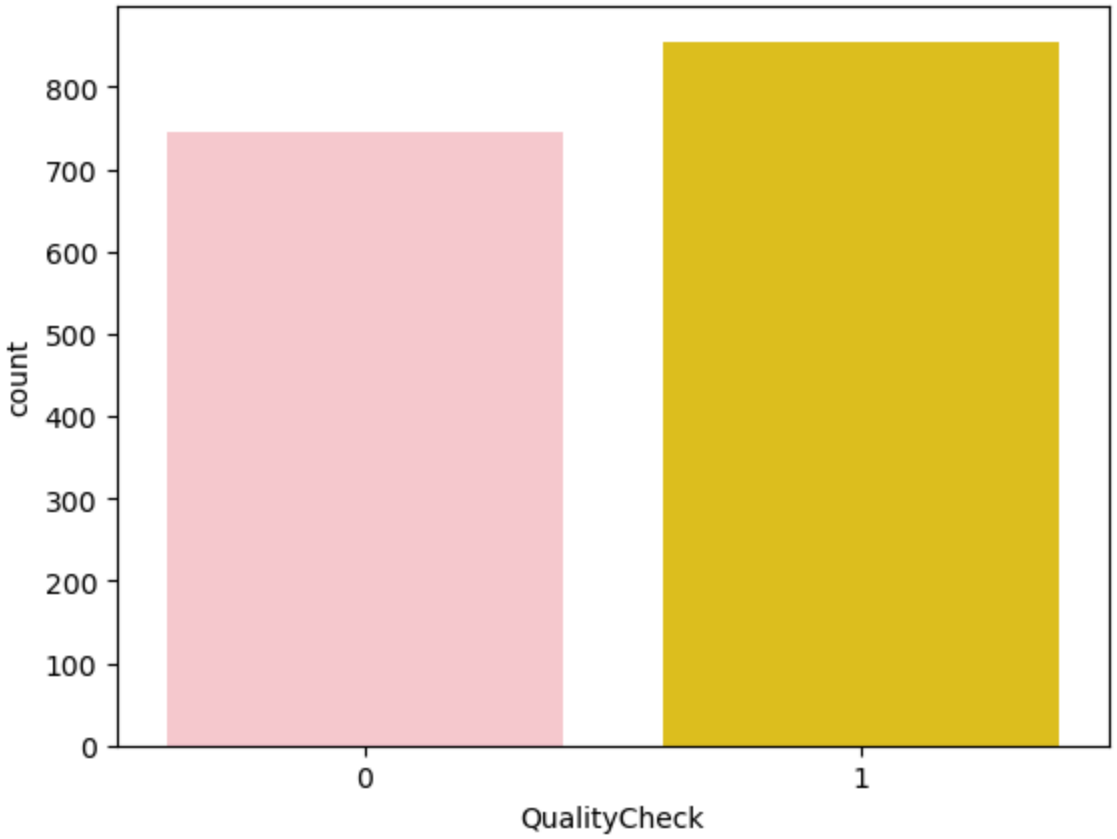
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.9986	3.17	0.93	
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	0.75	

In [158…

```
#Showing the total count of HIGH and LOW-quality red wine samples.
sns.countplot(x=Wine_Equality['QualityCheck'], palette=['pink', 'gold'])
```

Out[158]:

```
<Axes: xlabel='QualityCheck', ylabel='count'>
```

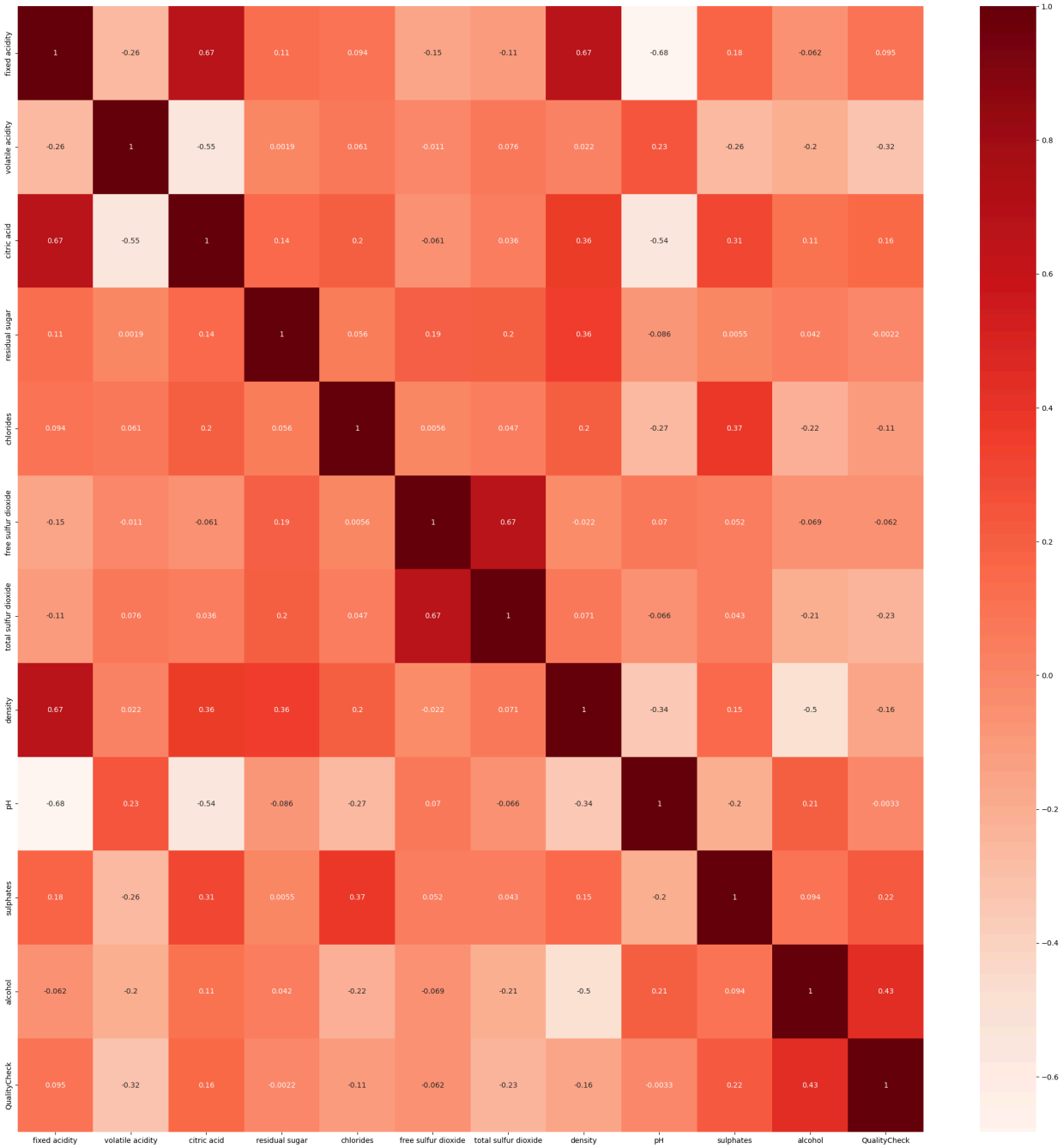


In [159…
```
#Computing the correlation coefficient among various parameters of 'Wine_Equal
correlation=Wine_Equality.corr()
correlation
```

Out[159]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-
QualityCheck	0.095093	-0.321441	0.159129	-0.002160	-0.109494	-0.061757	-0.231963	

In [160…
```
#Creating a heatmap using the seaborn library function.
plt.figure(figsize=(28,28))
heatmap = sns.heatmap(correlation, annot=True, cmap='Reds')
```



In [161…
```
#Changing fixed acidity into standard units.
Wine_Equality['fixed acidity (standard)']=(Wine_Equality['fixed acidity']-np.me

#Changing volatile acidity into standard units.
Wine_Equality['volatile acidity (standard)']=(Wine_Equality['volatile acidity'

#Changing citric acid into standard units.
Wine_Equality['citric acid (standard)']=(Wine_Equality['citric acid']-np.mean(W

#Changing residual sugar into standard unit
Wine_Equality['residual sugar (standard)']=(Wine_Equality['residual sugar']-np
#Changing chlorids into standard unit
Wine_Equality['chlorides (standard)']=(Wine_Equality['chlorides']-np.mean(Wine_
```

```
#Changing free sulfur dioxide into standatd unit
Wine_Equality['free sulfur dioxide (standard)']=(Wine_Equality['free sulfur di

#Changing total sulfur dioxide into standard unit
Wine_Equality['total sulfur dioxide (standard)']=(Wine_Equality['total sulfur

#Changing density into standard unit
Wine_Equality['density (standard)']=(Wine_Equality['density']-np.mean(Wine_Equa

#Changing pH into standard unit
Wine_Equality['pH (standard)']=(Wine_Equality['pH']-np.mean(Wine_Equality['pH'

#Changing sulphates into standard unit
Wine_Equality['sulphates (standard)']=(Wine_Equality['sulphates']-np.mean(Wine_

#Changing alcohol into standard unit
Wine_Equality['alcohol (standard)']=(Wine_Equality['alcohol']-np.mean(Wine_Equa

#17 rows of Wine_Equality printed
Wine_Equality.head(17)
```

Out[161]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	...
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	...
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	...
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	...
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	...
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	...
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	...
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	...
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	...
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	...
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	...
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	...
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	...
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	...
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	...
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	...
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.9986	3.17	0.93	...
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	0.75	...

17 rows × 23 columns

In [162…

```
#Randomizing the order of rows in the 'Wine_Equality' dataset.
Randomized=Wine_Equality.sample(len(Wine_Equality.axes[0]), replace = False)
```

```
#Splitting data into train set(1599=1119)
Set_Train=Randomized.iloc[0:1119, :]
#Splitting data into test set(1599=480)
Set_Test=Randomized.iloc[1119:, :]
```

In [163…

```
#17 rows of Set of the Train
Set_Train.head(17)
```

Out[163]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	.
610	8.8	0.24	0.54	2.50	0.083	25.0	57.0	0.99830	3.39	0.54	
1146	7.8	0.50	0.12	1.80	0.178	6.0	21.0	0.99600	3.28	0.87	
1548	11.2	0.40	0.50	2.00	0.099	19.0	50.0	0.99783	3.10	0.58	
475	9.6	0.68	0.24	2.20	0.087	5.0	28.0	0.99880	3.14	0.60	
1044	6.4	0.39	0.33	3.30	0.046	12.0	53.0	0.99294	3.36	0.62	
768	7.1	0.59	0.02	2.30	0.082	24.0	94.0	0.99744	3.55	0.53	
71	7.7	0.67	0.23	2.10	0.088	17.0	96.0	0.99620	3.32	0.48	
1156	8.5	0.18	0.51	1.75	0.071	45.0	88.0	0.99524	3.33	0.76	
1509	7.9	0.18	0.40	1.80	0.062	7.0	20.0	0.99410	3.28	0.70	
1059	8.9	0.48	0.53	4.00	0.101	3.0	10.0	0.99586	3.21	0.59	
762	8.8	0.70	0.00	1.70	0.069	8.0	19.0	0.99701	3.31	0.53	
1561	7.8	0.60	0.26	2.00	0.080	31.0	131.0	0.99622	3.21	0.52	
388	7.8	0.46	0.26	1.90	0.088	23.0	53.0	0.99810	3.43	0.74	
456	8.9	0.59	0.39	2.30	0.095	5.0	22.0	0.99860	3.37	0.58	
895	7.1	0.59	0.01	2.30	0.080	27.0	43.0	0.99550	3.42	0.58	
751	8.3	0.65	0.10	2.90	0.089	17.0	40.0	0.99803	3.29	0.55	
10	6.7	0.58	0.08	1.80	0.097	15.0	65.0	0.99590	3.28	0.54	

17 rows × 23 columns

In [164…

```
#17 rows of Set of the Test
Set_Test.head(17)
```

Out[164]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	.
1215	8.8	0.270	0.46	2.10	0.095	20.0	29.0	0.99488	3.26	0.56	
733	7.0	0.450	0.34	2.70	0.082	16.0	72.0	0.99800	3.55	0.60	
1060	11.6	0.230	0.57	1.80	0.074	3.0	8.0	0.99810	3.14	0.70	
526	7.3	0.365	0.49	2.50	0.088	39.0	106.0	0.99660	3.36	0.78	
410	9.0	0.430	0.34	2.50	0.080	26.0	86.0	0.99870	3.38	0.62	
7	7.3	0.650	0.00	1.20	0.065	15.0	21.0	0.99460	3.39	0.47	
1534	6.6	0.560	0.14	2.40	0.064	13.0	29.0	0.99397	3.42	0.62	
55	7.7	0.620	0.04	3.80	0.084	25.0	45.0	0.99780	3.34	0.53	
207	7.8	0.570	0.31	1.80	0.069	26.0	120.0	0.99625	3.29	0.53	
402	12.2	0.480	0.54	2.60	0.085	19.0	64.0	1.00000	3.10	0.61	
1251	7.5	0.580	0.14	2.20	0.077	27.0	60.0	0.99630	3.28	0.59	
522	8.2	0.390	0.49	2.30	0.099	47.0	133.0	0.99790	3.38	0.99	
501	10.4	0.440	0.73	6.55	0.074	38.0	76.0	0.99900	3.17	0.85	
450	11.9	0.390	0.69	2.80	0.095	17.0	35.0	0.99940	3.10	0.61	
1042	8.9	0.500	0.21	2.20	0.088	21.0	39.0	0.99692	3.33	0.83	
1070	9.3	0.330	0.45	1.50	0.057	19.0	37.0	0.99498	3.18	0.89	
43	8.1	0.660	0.22	2.20	0.069	9.0	23.0	0.99680	3.30	1.20	

17 rows × 23 columns

In [165…

```
# TRAINING and TESTING's length
print('Total rows of Set of the Train is ', len(Set_Train))
print('Total rows of Set of the Test is ', len(Set_Test))
```

```
Total rows of Set of the Train is  1119
Total rows of Set of the Test is  480
```

In [166…

```
#Selecting the independent features related to 'Wine_Equality' as X_Train and X
X_Train=Set_Train.iloc[:, 12:].values
X_Test=Set_Test.iloc[:, 12:].values

#Choosing the target variable 'Wine_Equality' as Y_Train and Y_Test.
Y_Train=Set_Train.iloc[:, 11].values
Y_Test=Set_Test.iloc[:, 11].values

#Reshaping the target variable of 'Wine_Equality'
Y_Train=np.reshape(Y_Train, (len(Y_Train), 1))
Y_Test=np.reshape(Y_Test, (len(Y_Test), 1))

#Showing the shape of the matrix of the TRAIN set.
print("X_train_Shape:", np.shape(X_Train))
print("Y_train_Shape:", np.shape(Y_Train))

#Showing the shape of the matrix of the TEST set.
```

```
print("X_test_Shape:", np.shape(X_Test))
print("Y_test_Shape:", np.shape(Y_Test))
```

```
X_train_Shape: (1119, 11)
Y_train_Shape: (1119, 1)
X_test_Shape: (480, 11)
Y_test_Shape: (480, 1)
```

In [167…

```
#Transposing the X_Train and X_Test
X_Train_Trans=np.transpose(X_Train)
X_Test_Trans=np.transpose(X_Test)

#Adding rows of ones vertically to the X_Train and X_Test datasets.
X_Train_Aug=np.vstack((np.ones((1,len(X_Train))),X_Train_Trans))
X_Test_Aug=np.vstack((np.ones((1,len(X_Test))),X_Test_Trans))

#Showing the shape of the augmented matrix of X_Train_Aug and X_Test_Aug.
print("Shape of X_train_Aug: ", np.shape(X_Train_Aug))
print("Shape of X_test_Aug: ", np.shape(X_Test_Aug))
```

```
Shape of X_train_Aug:  (12, 1119)
Shape of X_test_Aug:  (12, 480)
```

In [168…

```
#Defining theta as a 2D array initialized with zeros.
theta=np.zeros((12,1))
print("Shape of θ is: ", np.shape(theta))
```

```
Shape of θ is:  (12, 1)
```

In [169…

```
#Total iteration value
no_of_iteration=np.arange(1,50001)

#The learning Set
alpha=0.005

#Getting the X-train's length
Length_Train=len(X_Train)

#Getting the X-test's length
Length_Test=len(X_Test)

#Establishing an empty list to store the values of the cost function in each i
Func_Cost=[]

#Running the loop for gradient descent iterations.
for i in no_of_iteration:
#Hypothesis function (A matrix of size 1 * 1119).
    z=np.transpose(theta)@X_Train_Aug

    #Sigmoid Function Equation:
    p = (1/(1+np.exp(-z)))

    #Cost Function if predicted value(wrong presiction) is 1
    ft = ((np.log10((1/(1+np.exp(-z)))))@Y_Train)[0,0]

    #Func_Cost if predicted wrong value(wrong presiction) is 0
    st = ((np.log10(1-(1/(1+np.exp(-z)))))@(1-Y_Train))[0,0]

    #Finding the Func_Cost
    cf = (1/Length_Train)*(-ft-st)
```

```
    #Appending the Func_Cost function to empty list[]
    Func_Cost.append(cf)

    #Calculating derivative of Func_Cost
    delthetaj = (1/Length_Train)*((X_Train_Aug)@(np.transpose(p)-Y_Train))

    #Changing the values of theta after each loop
    theta = theta-(alpha*delthetaj)

#The length of cost function printed
print(len(Func_Cost))

#Final value of theta printed
print(theta)
```

```
50000
[[ 0.31614381]
 [ 0.28994692]
 [-0.63909148]
 [-0.30285646]
 [ 0.11179967]
 [-0.21776326]
 [ 0.24636498]
 [-0.54256798]
 [-0.078142  ]
 [ 0.02555867]
 [ 0.56638527]
 [ 0.8461667 ]]
```

In [170…

```
#Assessing the model's performance on the training set.
Pred_Train_Yaxis = np.zeros((Length_Train,1))
YTheta = np.transpose(theta)@X_Train_Aug
Y_Theta_Trans = np.transpose(YTheta)

#Evaluating the consistency between the predicted outcomes and the actual resu
for j in range(Length_Train):
    if  1/(1+np.exp(-Y_Theta_Trans[j])) >= 0.5:
        Pred_Train_Yaxis[j] = [1]
    else:
        Pred_Train_Yaxis[j] = [0]

#The training dataset printed .
print("The training dataset : ", Y_Train)

#The predicted training dataset printed .
print("The predicted training dataset : ", Pred_Train_Yaxis)
```

```
The training dataset :  [[0]
 [1]
 [0]
 ...
 [1]
 [1]
 [1]]
The predicted training dataset :  [[0.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
```

In [171…

```
#Testing the model on set of "test"
Pred_Test_Y = np.zeros((Length_Test,1))
YTheta = np.transpose(theta)@X_Test_Aug
Y_Theta_Trans = np.transpose(YTheta)

#Contrasting the predicted results with the actual results.
for j in range(Length_Test):
    if  1/(1+np.exp(-Y_Theta_Trans[j])) >= 0.5:
        Pred_Test_Y[j] = [1]
    else:
        Pred_Test_Y[j] = [0]
#Displaying our predicted testing samples.
print("Test Sample Guess: ", Y_Test)

#Displaying our predicted testing samples.
print("Test Sample Guess: ", Pred_Test_Y)
```

```
Test Sample Guess:  [[1]
 [0]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
 [1]
```

```
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
```

```
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
```

```
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
```

```
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
```

```
[1]
[1]
[0]
[1]
[1]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
```

```
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
```

```
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[0]]
```

```
Test Sample Guess:  [[1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
```

```
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
```

```
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
```

```
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
```

```
[1.]
[1.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
```

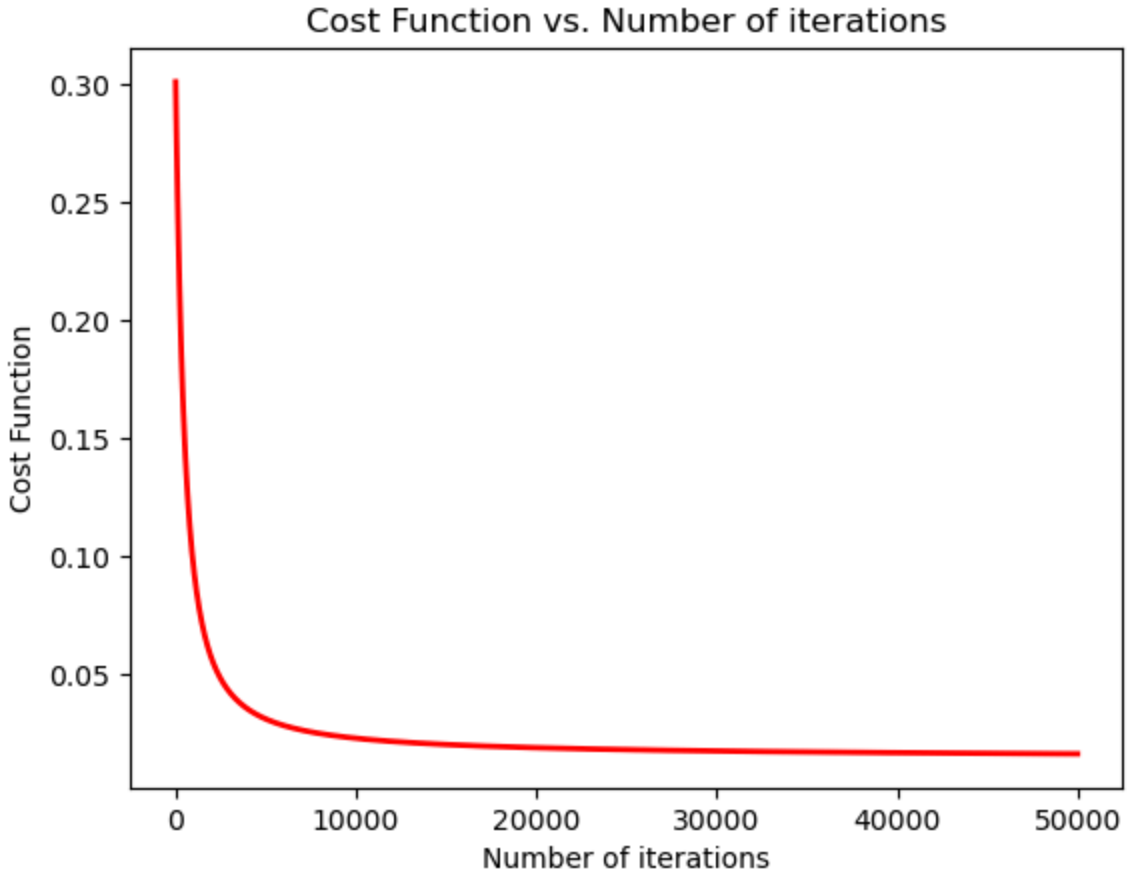
```
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
```

```
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
```

```
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]]
```

In [172…
```
#Graphing the cost function against the number of iterations.
plt.plot(no_of_iteration,costfunc, color='red', linewidth='2')
plt.xlabel("Number of iterations")
plt.ylabel("Cost Function")
plt.title("Cost Function vs. Number of iterations")
```

Out[172]:
```
Text(0.5, 1.0, 'Cost Function vs. Number of iterations')
```



In [173…
```
#Printing the length of training(70%) and testing(30%) sets
print('Number of rows in training set is: ', len(Y_Train))
print('Number of rows in testing set is: ', len(X_Test))
```

```
Number of rows in training set is:  1119
Number of rows in testing set is:  480
```

In [174…
```
#Evaluating the model on the training set.
#Positive
count_TP=0
for TP in range(Length_Train):
    if (Pred_Train_Yaxis[TP]==1) & (Y_Train[TP]==1):
        count_TP=count_TP+1
print("True_Positives: ", count_TP)

#False Positive
count_FP=0
for FP in range(Length_Train):
    if (Pred_Train_Yaxis[FP]==1) & (Y_Train[FP]==0):
        count_FP=count_FP+1
print("False Positives: ", count_FP)

#True Negative
```

```
count_TN=0
for TN in range(Length_Train):
    if (Pred_Train_Yaxis[TN]==0) & (Y_Train[TN]==0):
        count_TN=count_TN+1
print("True Negatives: ", count_TN)

#Negative
count_FN=0
for FN in range(Length_Train):
    if (Pred_Train_Yaxis[FN]==0) & (Y_Train[FN]==1):
        count_FN=count_FN+1
print("False Negatives: ", count_FN)

#Determining the accuracy of our train module.
Accuracy=(count_TP+count_TN)/Length_Train
print("Accuracy of model is: ", Accuracy)

#Computing the clarity of our train module.
Precision=count_TP/(count_TP+count_FP)
print("Precision of model is: ", Precision)

#Determining the recall of our train module.
Recall=count_TP/(count_TP+count_FN)
print("Recall of model is: ", Recall)

#Computing the F1 score of our train module.
F1_Score=(2*Precision*Recall)/(Precision+Recall)
print("F1_Score of model is:", F1_Score)
```

```
True_Positives:  471
False Positives:  143
True Negatives:  361
False Negatives:  144
Accuracy of model is:  0.743521000893655
Precision of model is:  0.7671009771986971
Recall of model is:  0.7658536585365854
F1_Score of model is: 0.7664768104149716
```

In [179…

```
#Evaluating the model on the training set.
#Positive
count_TP=0
for TP in range(Length_Test):
    if (Pred_Train_Yaxis[TP]==1) & (Y_Train[TP]==1):
        count_TP=count_TP+1
print("True_& Positives: ", count_TP)

#Positive
count_FP=0
for FP in range(Length_Test):
    if (Pred_Train_Yaxis[FP]==1) & (Y_Train[FP]==0):
        count_FP=count_FP+1
print("False & Positives: ", count_FP)


#Negative
count_TN=0
for TN in range(Length_Test):
    if (Pred_Train_Yaxis[TN]==0) & (Y_Train[TN]==0):
        count_TN=count_TN+1
print("True & Negatives: ", count_TN)
```

```
#Negative
count_FN=0
for FN in range(Length_Train):
    if (Pred_Train_Yaxis[FN]==0) & (Y_Train[FN]==1):
        count_FN=count_FN+1
print("False & Negatives: ", count_FN)

#Determining the accuracy of our trained module.
Accuracy=(count_TP+count_TN)/Length_Train
print("Model's accuracy is:  ", Accuracy)

#Computing the clarity of our trained module.
Precision=count_TP/(count_TP+count_FP)
print("Model's precision is: ", Precision)

#Determining the recall of our trained module.
Recall=count_TP/(count_TP+count_FN)
print("Model's recall is: ", Recall)

#Computing the F1 score of our trained module.
F1_Score=(2*Precision*Recall)/(Precision+Recall)
print("Model's f1_score is :", F1_Score)
```

```
True_& Positives:  196
False & Positives:  64
True & Negatives:  156
False & Negatives:  144
Model's accuracy is:   0.3145665773011618
Model's precision is:  0.7538461538461538
Model's recall is:  0.5764705882352941
Model's f1_score is : 0.6533333333333333
```

In [180…

```
#Importing module for training and testing samples
from sklearn.model_selection import train_test_split
#Importing module for model development and prediction
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
#Importing modules for evaluating performance of classification model
from sklearn import metrics
#Importing module for showing text report of classification mode
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [181…

```
#Choosing the independent features related to 'winequality'.
X=Redwine[['fixed acidity','volatile acidity','citric acid','residual sugar','
#Selecting dependent or target variable of 'winequality'
Y=Redwine.QualityCheck
```

In [182…

```
#Transforming features into their standard units using the Sklearn library fun
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = sc_X.fit_transform(X.astype(float))
X
```

Out[182]:
```
array([[-0.52835961,  0.96187667, -1.39147228, ...,  1.28864292,
        -0.57920652, -0.96024611],
       [-0.29854743,  1.96744245, -1.39147228, ..., -0.7199333 ,
         0.1289504 , -0.58477711],
       [-0.29854743,  1.29706527, -1.18607043, ..., -0.33117661,
        -0.04808883, -0.58477711],
       ...,
       [-1.1603431 , -0.09955388, -0.72391627, ...,  0.70550789,
         0.54204194,  0.54162988],
       [-1.39015528,  0.65462046, -0.77526673, ...,  1.6773996 ,
         0.30598963, -0.20930812],
       [-1.33270223, -1.21684919,  1.02199944, ...,  0.51112954,
         0.01092425,  0.54162988]])
```

In [183…
```
#Dividing our independent feature (X) and dependent variable (Y) into training
X_Train,X_test,Y_Train,Y_test=train_test_split(X,Y,train_size=0.70,test_size=0
```

In [184…
```
#Displaying the size of the training and testing datasets.
print('Number of rows in training set is: ', len(X_Train))
print('Number of rows in testing set is: ', len(X_Test))
```

```
Number of rows in training set is:  1119
Number of rows in testing set is:  480
```

In [185…
```
#Importing module for training and testing samples
from sklearn.model_selection import train_test_split
#Importing module for model development and prediction
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
#Importing modules for evaluating performance of classification model
from sklearn import metrics
#Importing module for showing text report of classification mode
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#Creating a logistic regression classifier object using logistic regression fu
model=LogisticRegression(max_iter=50000, solver = 'lbfgs')
#Training a model
model.fit(X_Train,Y_Train)
```

Out[185]:
```
▾ LogisticRegression
LogisticRegression(max_iter=50000)
```

In [186…
```
#Conducting testing on a separate training dataset.
Pred_Train_Yaxis = model.predict(X_Train)
Pred_Train_Yaxis
```

Out[186]:
```
array([1, 1, 0, ..., 0, 1, 0], dtype=uint8)
```

In [187…
```
#Conducting testing on a separate test dataset.
Pred_Test_Y = model.predict(X_Test)
Pred_Test_Y
```

Out[187]:
```
array([1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1], dtype=uint8)
```

In [188…
```
#Assessing the performance of the model on the training set.
print(classification_report(Y_Train,Pred_Train_Yaxis))
print(confusion_matrix(Y_Train,Pred_Train_Yaxis))
```

```
              precision    recall  f1-score   support

           0       0.73      0.72      0.73       518
           1       0.76      0.77      0.77       601

    accuracy                           0.75      1119
   macro avg       0.75      0.75      0.75      1119
weighted avg       0.75      0.75      0.75      1119

[[374 144]
 [139 462]]
```

In [189…
```
##Assessing the performance of the model on the testing set.
print(classification_report(Y_Test,Pred_Test_Y))
print(confusion_matrix(Y_Test,Pred_Test_Y))
```

```
              precision    recall  f1-score   support

           0       0.78      0.73      0.76       240
           1       0.75      0.80      0.77       240

    accuracy                           0.76       480
   macro avg       0.77      0.76      0.76       480
weighted avg       0.77      0.76      0.76       480

[[175  65]
 [ 48 192]]
```

In [190…
```
#Determining the accuracy of our trained model.
print("Accuracy of model is: ", metrics.accuracy_score(Y_Test, Pred_Test_Y))

#Assessing the recall of our trained model.
print("Precision of model is: ", metrics.precision_score(Y_Test, Pred_Test_Y))
```

```
#Assessing the recall of our trained model.
print("Recall of model is: ", metrics.recall_score(Y_Test, Pred_Test_Y))

#Evaluating our trained model by computing its F1 score.
print("F1-score of model is: ", metrics.f1_score(Y_Test, Pred_Test_Y))
```

```
Accuracy of model is:  0.7645833333333333
Precision of model is:  0.7470817120622568
Recall of model is:  0.8
F1-score of model is:  0.772635814889336
```

In [191…

```
#Presenting readings of various parameters from the 'Wine_Equality' dataset.
Input_Readings=(7.4,0.7,0,1.9,0.076,11,34,0.9978,3.51,0.56,9.4)
#Converting the input to an array
Input_Readings=np.array(Input_Readings)

#Changing the input array specifically for this set of data.
Input_Readings=Input_Readings.reshape(1,-1)

#Guessing for the Graph
Guess = model.predict(Input_Readings)
print(Guess)
```

```
[0]
```

In [192…

```
print("Thankyou")
```

```
Thankyou
```