MACHINE LEARNING

(Wine Quality Prediction)

Summer Internship Report Submitted in partial

fulfillment of the requirement for undergraduate degree

of

Bachelor of Technology
In
Computer Science and Engineering

By
Dheeraj Bamar

221710314065

Under the Guidance

of

Mr/Mrs

Assistant Professor

GITAM (DEEMED TO BE UNIVERSITY)
VISAKHAPATNAM HYDERABAD BENGALURU

Department Of Computer Science and Engineering

GITAM School of Technology

GITAM (Deemed to be University) Hyderabad-502329

June 2020

DECLARATION

I submit this industrial training work entitled **"WINE QUALITY PREDICTION**" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science and Engineering**". I declare that it was carried out independently by me under the guidance of **Mr** , Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD

Date:

Student Name: DHEERAJ BAMAR

Student RollNo:221710314065

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Date:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **“WINE QUALITY PREDICTION”** is being submitted by Dheeraj Bamar(221710314065) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science and Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19

It is faithful record work carried out by him at the **Computer Science and Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Dr.S.Phani Kumar

Assistant Professor | Professor and HOD

Department of CSE | Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM HYDERABAD

I would like to thank respected **Dr. S. Phani Kumar,** Head of the Department of Electronics and Communication Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

StudentName :DHEERAJ BAMAR

StudentRollNo:221710314065

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predi ct the values is the primary task to be performed on Cereals data set My perception of understanding the given data set has been in the view of undertaking a client's requirement of overcoming the stagnant point of sales of the products being manufactured by client.

To get a better understanding and work on a strategical approach for solution of the client, I have adapted the view point of looking at ratings of the products and for further deep understanding of the problem, I have taken the stance of a consumer and reasoned out the various factors of choice of the products and they purchase , and my primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to ratings of products and draft out an outcome report to client regarding the various accepts of a product manufacturing , marketing and sale point determination

TABLE OF CONTENTS

LIST OF FIGURES:

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

Figure 1: The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

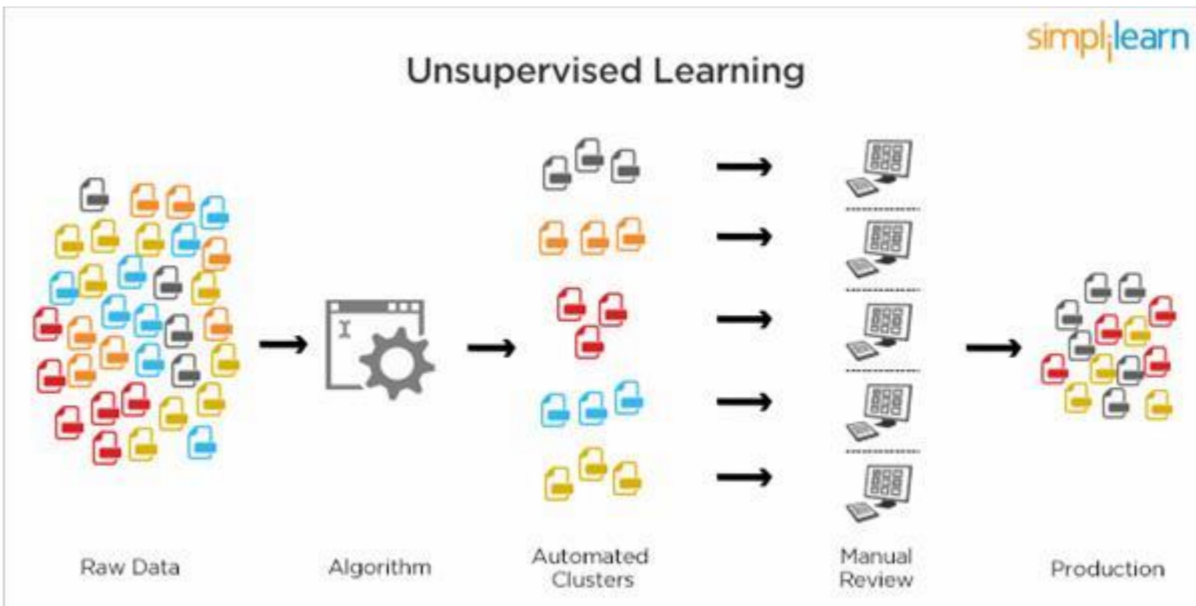


Figure 2: Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

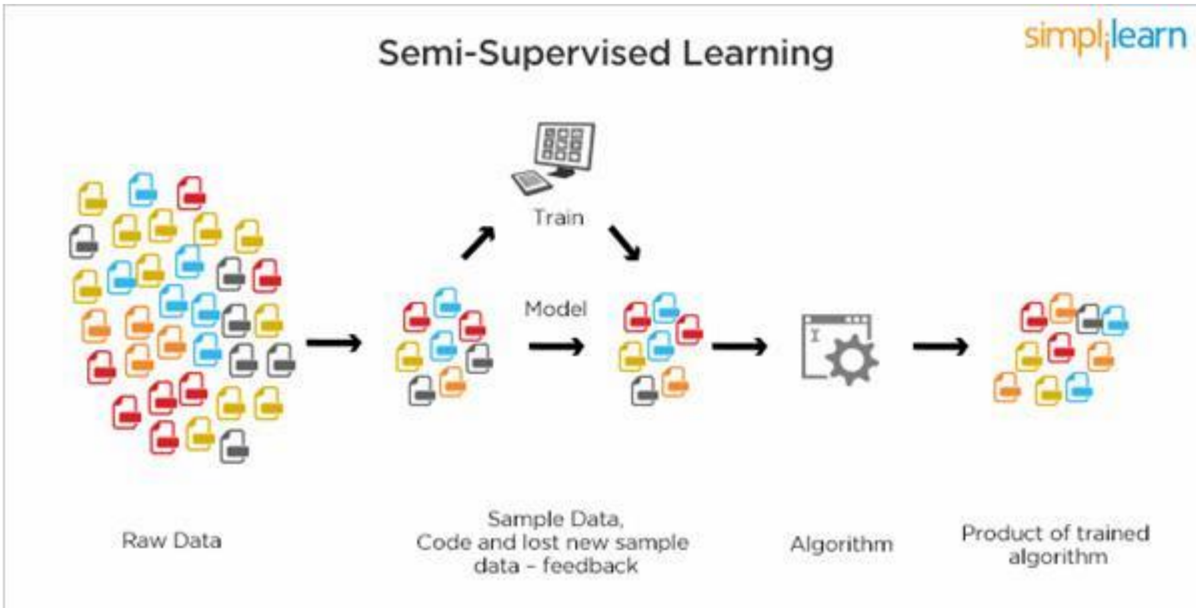


Figure 3: Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly definedsyntax,This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the yes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc.,is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.

- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

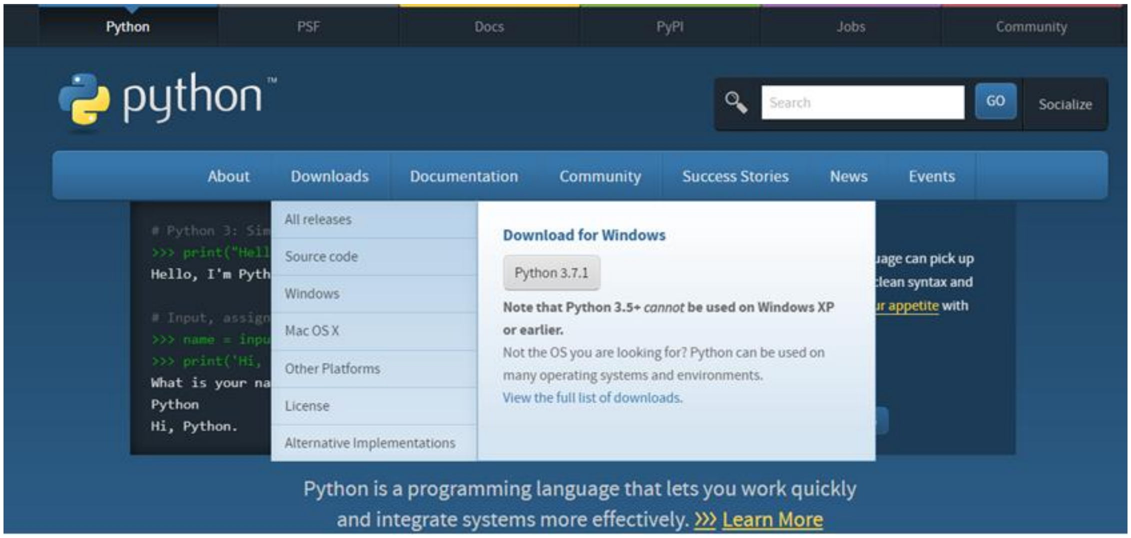


Figure 4 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.
- In WINDOWS:
- In windows
  - Step 1: Open Anaconda.com/downloads in web browser.
  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
  - Step 3: select installation type(all users)

- Step 4: Select path(i.e.add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
- Step 5: Open jupyter notebook (it opens in default browser)

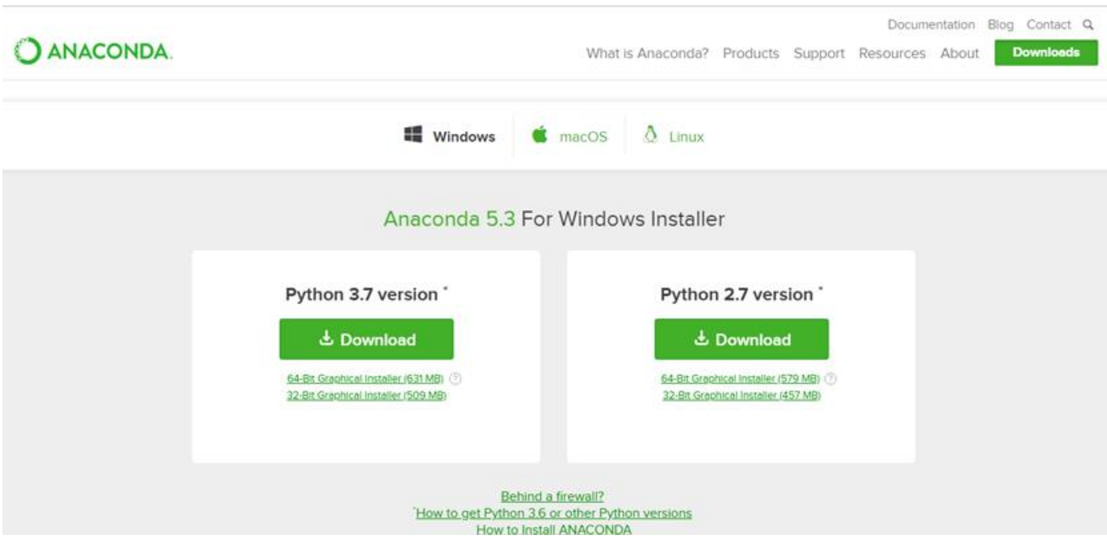


Figure 5 : Anaconda download

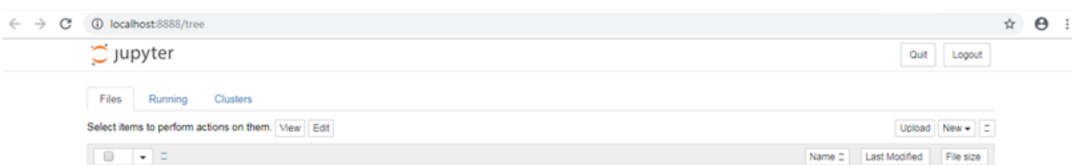


Figure 6 :Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types–
  - Numbers
  - Strings
  - Lists
  - Tuples
  - Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented inthe quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3Python Lists:

- Lists are the most versatile of Python's compound datatypes.
- A list contains items separated by commas and enclosed within square brackets([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different datatype.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end-1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists ,however,

tuples are enclosed within parentheses.

- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of alist.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what itis.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variable sand instance variables) and methods, accessed via dot notation.

- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variable sare.

- Data member: A class variable or instance variable that holds data associated witha class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
    - We define a class in a very similar way how we define a function.
    - Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7 : Defining a Class

2.7.2 __init__ method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores:__init__().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

To predict the amount of purchases in Retail shop using Machine Learning algorithm called MULTIPLE LINEAR REGRESSION

3.2 DATA SET:

The given data set consists of the following parameters:

A - fixed_acidity

B - volatile_acidity

C - citric_acid

D - residual_sugar

E - chlorides

F - free_sulfur_dioxide

G - total_sulfur_dioxide

H - density

I - pH

J - sulphates

K - alcohol

L –quality

M- color

N- dtype: object

3.3 OBJECTIVE OF THE CASE STUDY:

To get a better understanding and chalking out a plan of action for solution of the client, we have adapted the view point of looking at product categories and for further deep understanding of the problem, we have also considered gender age of the customer and reasoned out the various factors of choice of the products and they purchase , and our primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to product categories and draft out an outcome report to client regarding the various accepts of a product purchases.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

We can get the data set from the database or

we can get the data from client.

4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.

IMPORT THE LIBRARIES

```
]: import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   import seaborn as sns
```

Figure 8 : Importing Libraries

4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed.It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaNvalue have to be cleaned.

```
In [3]: #READ THE DATA
        df= pd.read_csv("C:\\Users\\DELL\\Desktop\\Project\\wine.csv")

In [4]: df.head()
```

Out[4]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red

Figure 9 : Reading the dataset

4.1.4 SHAPE OF DATA:

This displays the no.of coloumns and no.of rows in the given data set

```
In [5]: df.shape

Out[5]: (6497, 13)
```

NUMBER OF COLUMNS

```
In [6]: df.columns

Out[6]: Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
               'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
               'pH', 'sulphates', 'alcohol', 'quality', 'color'],
              dtype='object')
```

Figure 10: Numbering the columns

DATA TYPES IN THE DATA:

This displays the data types of the data in the given datset.From the given data we can observe that all are in integer type except the color

DATA TYPES IN THE DATA

```
In [7]: df.dtypes

Out[7]: fixed_acidity           float64
        volatile_acidity        float64
        citric_acid             float64
        residual_sugar          float64
        chlorides               float64
        free_sulfur_dioxide     float64
        total_sulfur_dioxide    float64
        density                 float64
        pH                      float64
        sulphates               float64
        alcohol                 float64
        quality                   int64
        color                    object
        dtype: object
```

Figure 11 : Defining data type

CHECKING FOR ANY MISSING DATA:

```
In [8]: df.isnull().sum()

Out[8]: 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
        color                   0
        dtype: int64
```

Figure 12 : Checking for missing data

There are no missing values in the data we can observe from the data

GETTING THE INFORMATION OF ALL UNIQUE DATA

There are many unique values in the dataset

```
In [9]: df.nunique()

Out[9]: fixed_acidity            106
        volatile_acidity         187
        citric_acid               89
        residual_sugar           316
        chlorides                214
        free_sulfur_dioxide      135
        total_sulfur_dioxide     276
        density                  998
        pH                       108
        sulphates                111
        alcohol                  111
        quality                    7
        color                      2
        dtype: int64
```

Figure 13 : Information of unique data

DESCRIBE THE DATA

From this describe function we can observe that that data is described in many ways

```
In [11]: df.describe()
Out[11]:
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	115.744574	0.994697	3.218501	0.531268
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	56.521855	0.002999	0.160787	0.148806
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000

Figure 14: Describe data

CHECK THE NUMBER OF QUALITY RANGES

From this we can observe the different number of quality of divided acoording to there quality content as we are taking quality content as the the main data to predict.

```
In [12]: df['quality'].value_counts()

Out[12]: 6    2836
         5    2138
         7    1079
         4     216
         8     193
         3      30
         9       5
         Name: quality, dtype: int64
```

Figure 15: Quality check

NUMBER OF WHITE AND RED WINE

Counting the number of white wine and red wine we can see there are many of the white wine

```
In [13]: df['color'].value_counts()

Out[13]: white    4898
         red      1599
         Name: color, dtype: int64
```

Figure 16: Counting total

PLOTTING NUMBER OF RED AND WHITE WINE ON GRAPH

We are then plotting the the number of red wine and white wine on the bargraph from good understanding

In [14]:
```
plt.figure(figsize=(5,5))
sns.countplot(df['color'])
```

Out[14]: `<matplotlib.axes._subplots.AxesSubplot at 0x2d00ead8508>`

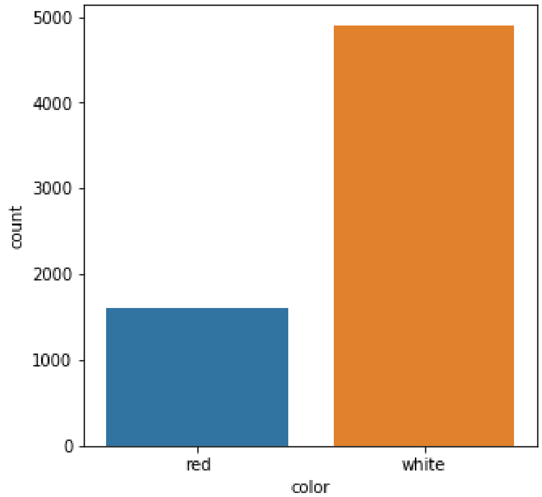


Figure 17: Plotting number in bar graph

PLOTTING QUALITY OF RED AND WHITE WINE ON GRAPH

From this graph we can show that quality 6 is having is highest number of wine quality

In [21]:
```
plt.figure(figsize=(5,5))
sns.countplot(df['quality'])
```

Out[21]: `<matplotlib.axes._subplots.AxesSubplot at 0xb874908>`

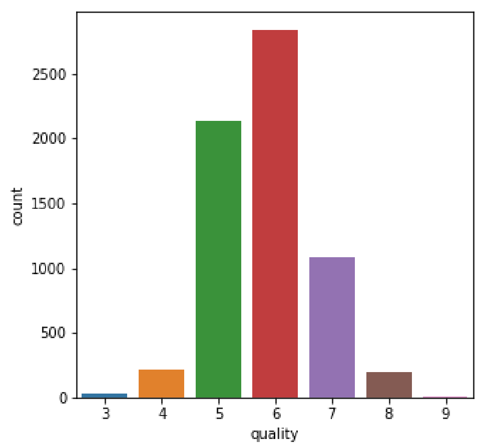


Figure 18 : Plotting quality in bar graph

KNOWING THE MISSING VALUES

Checking for missing values by plotting missing graph

```
In [24]: import missingno as msno
         msno.matrix(df)

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0xb93e208>
```

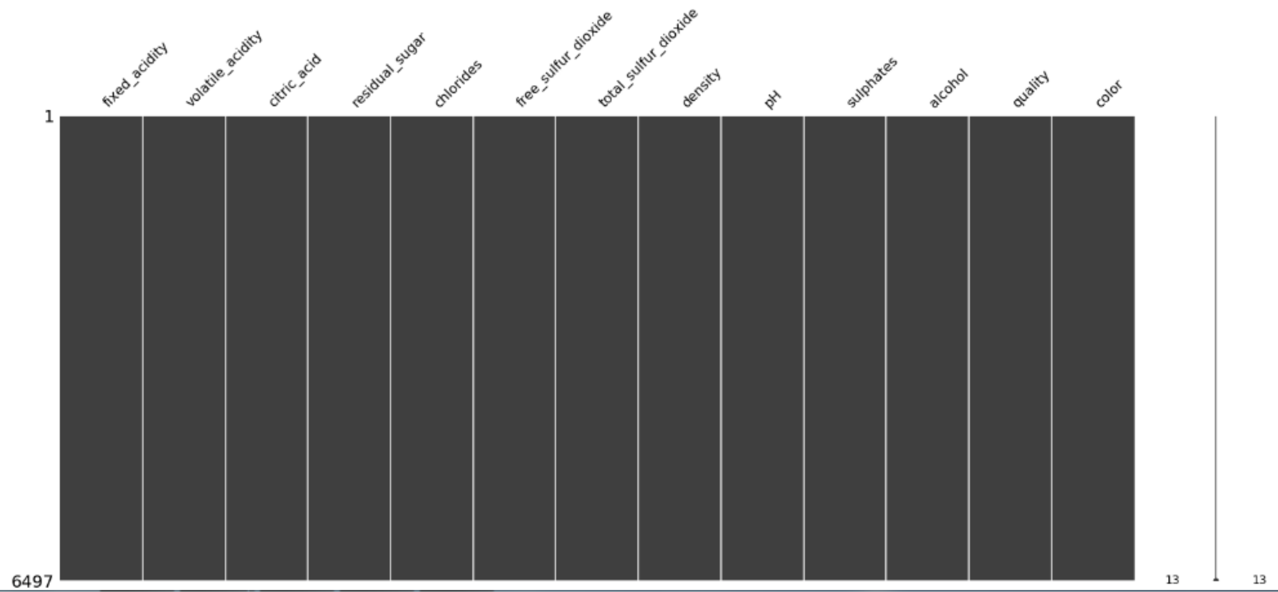


Figure 19: Checking for missing value

MISSING VALUES ON BAR GRAPH

```
In [25]: msno.bar(df)
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0xb64dc88>
```

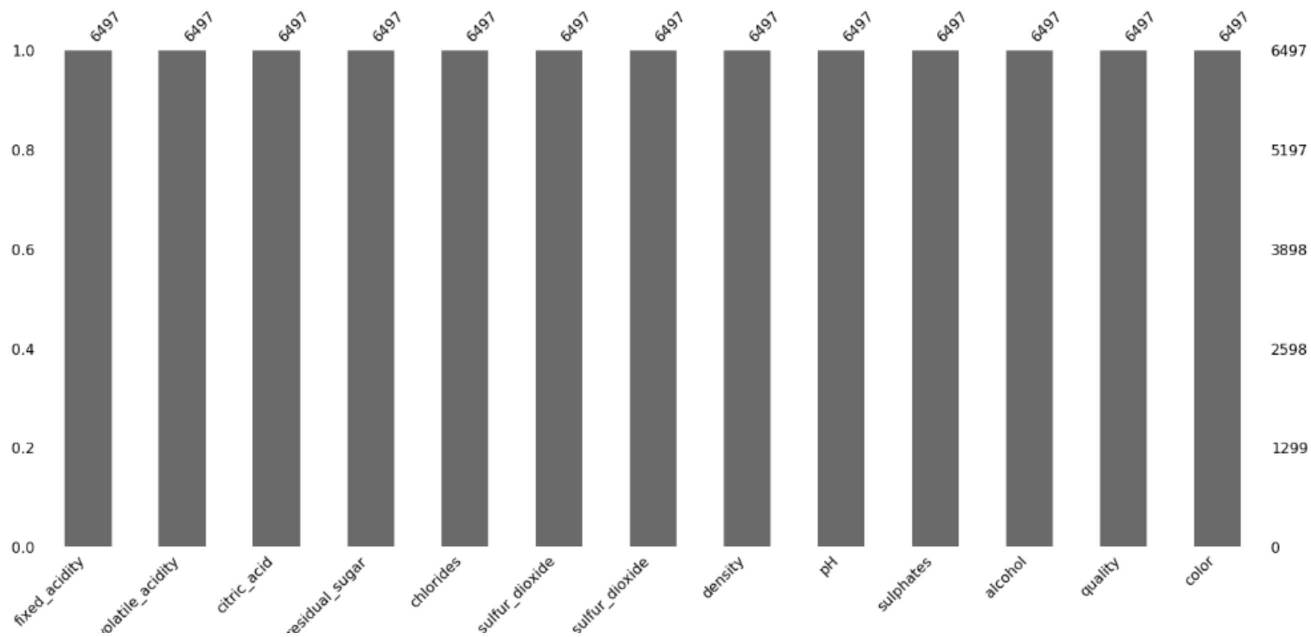


Figure 20: Missing value on bar graph

GRAPH BETWEEN COMPARISION BETWEEN BOTH THE WINES

Comparing between the quality by there color and white wine has more number quality wine

```
In [26]: sns.barplot(x='color',y='quality',data=df)
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0xb906788>
```

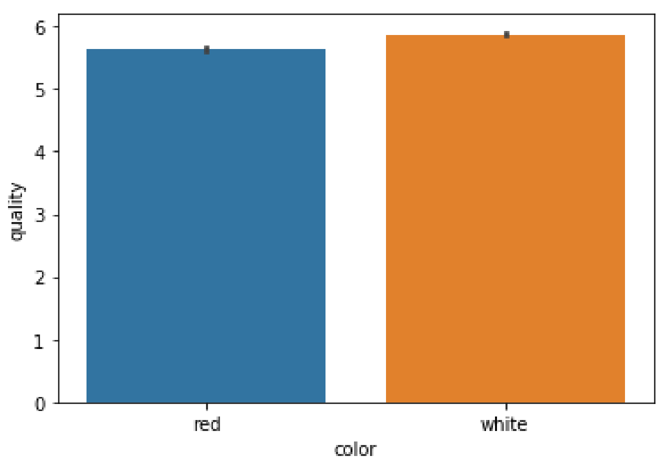
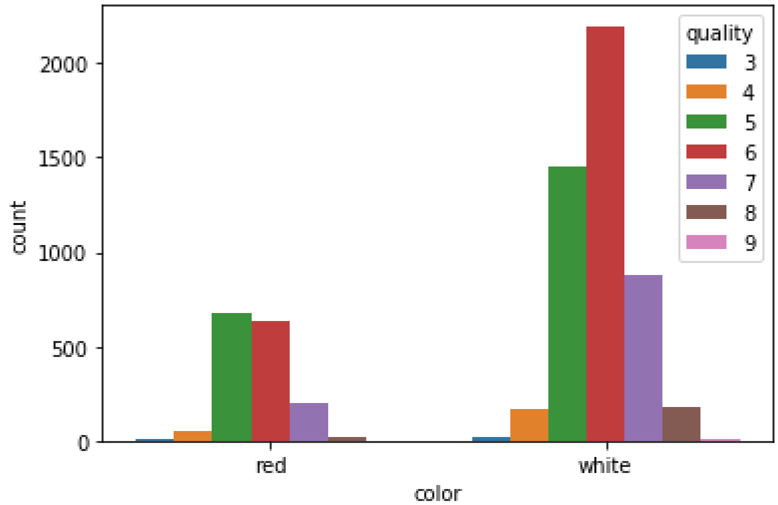


Figure 21: Comparisons between wines

SHOWING THE DATA ON GRAPH

Plotting countplot between the number of quality on there color. As white wine is more it shows more number of count

```
In [27]: sns.countplot(x='color',hue='quality',data=df)

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xb4ddf08>
```



DRAWING A HISTOGRAM

Histogram is drawn to show the graph comparison between the all the other columns of the data

```
In [28]: df.hist(figsize=(20,20))

Out[28]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B73F588>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B75DD88>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B795EC8>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B7D2048>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B98B148>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B9C4248>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BC2C2C8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BC653C8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BC69F88>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BEBB188>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF1E708>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF57788>]],
               dtype=object)
```

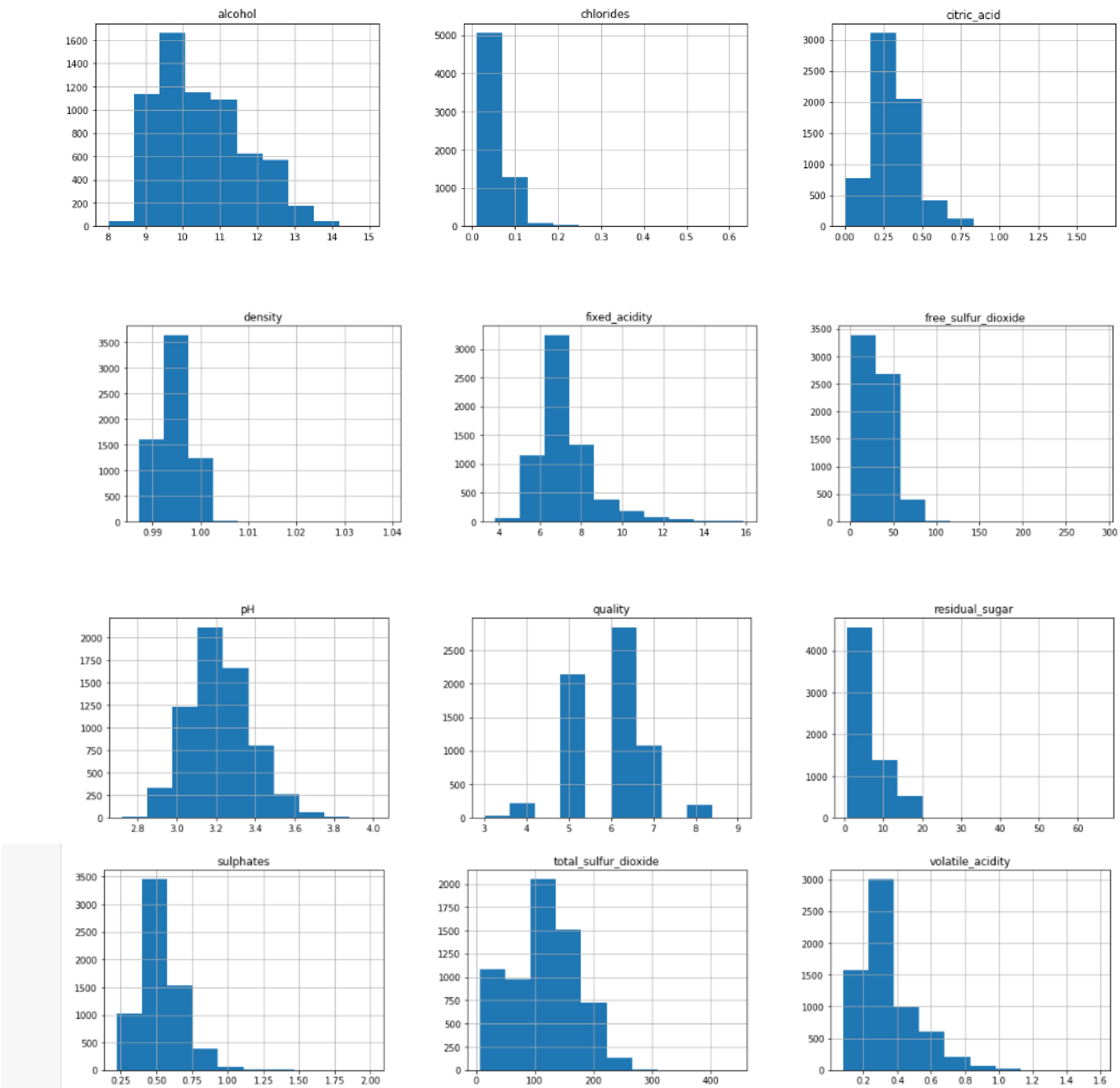


Figure 22: Histogram of the data

HEATMAP

It graphical representation of data where the individual values contained in a matrix are represented as colors

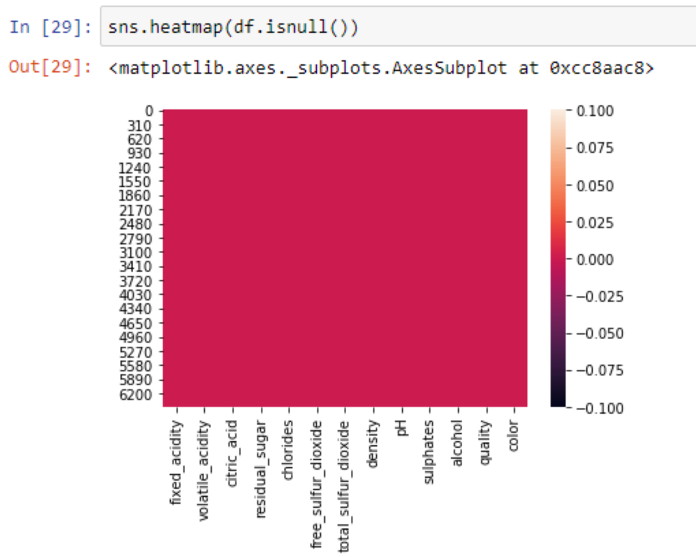


Figure 23: Heat map

4.1.5 CATEGORICAL DATA:

- Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.
- Categorical Variables are of two types: Nominal and Ordinal
- **Nominal**: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them.Examples: Male or Female, any colour
- **Ordinal**: The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

- Categorical data can be handled by using dummy variables, which are also called as indicator variables.

HISTOGRAM OF QUALITY

The quality of the data set is represented in the form of histogram

```
In [30]: df['quality'].hist()
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0xcd27248>
```

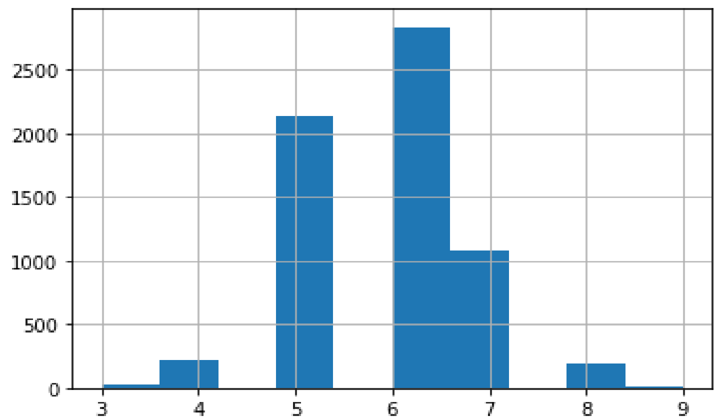


Figure 24: Histogram of quality

DISTPLOT IS DRAWN TO KNOW THE QUALITY

The values are plotted on the Distplot for the better understanding if the data set

```
sns.distplot(df['quality'],kde=False)
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x2d018addd08>

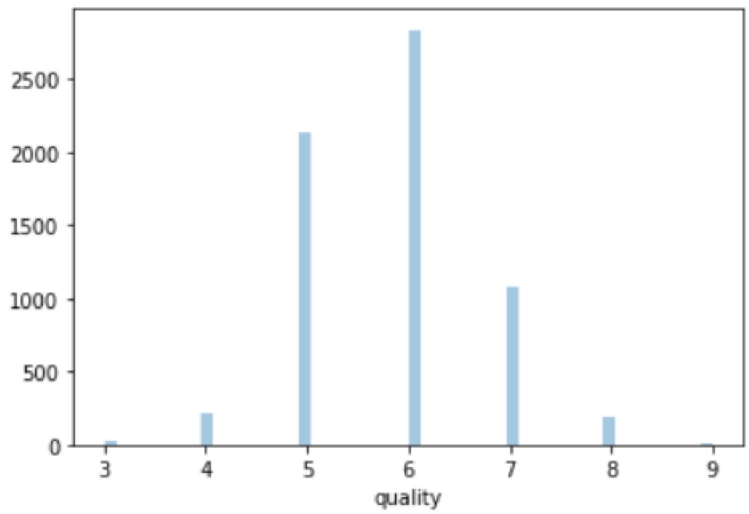


Figure 25 :Displot for quality

USING THE HEATMAP TO GET THE BETTER UNDERSTANDING OF THE DATA WE CAN USE THE DATA TO COMPAIR THE VALUES

```
plt.figure(figsize=(14,14))
sns.heatmap(df.corr(),annot=True,cmap='plasma')
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x2d014231548>

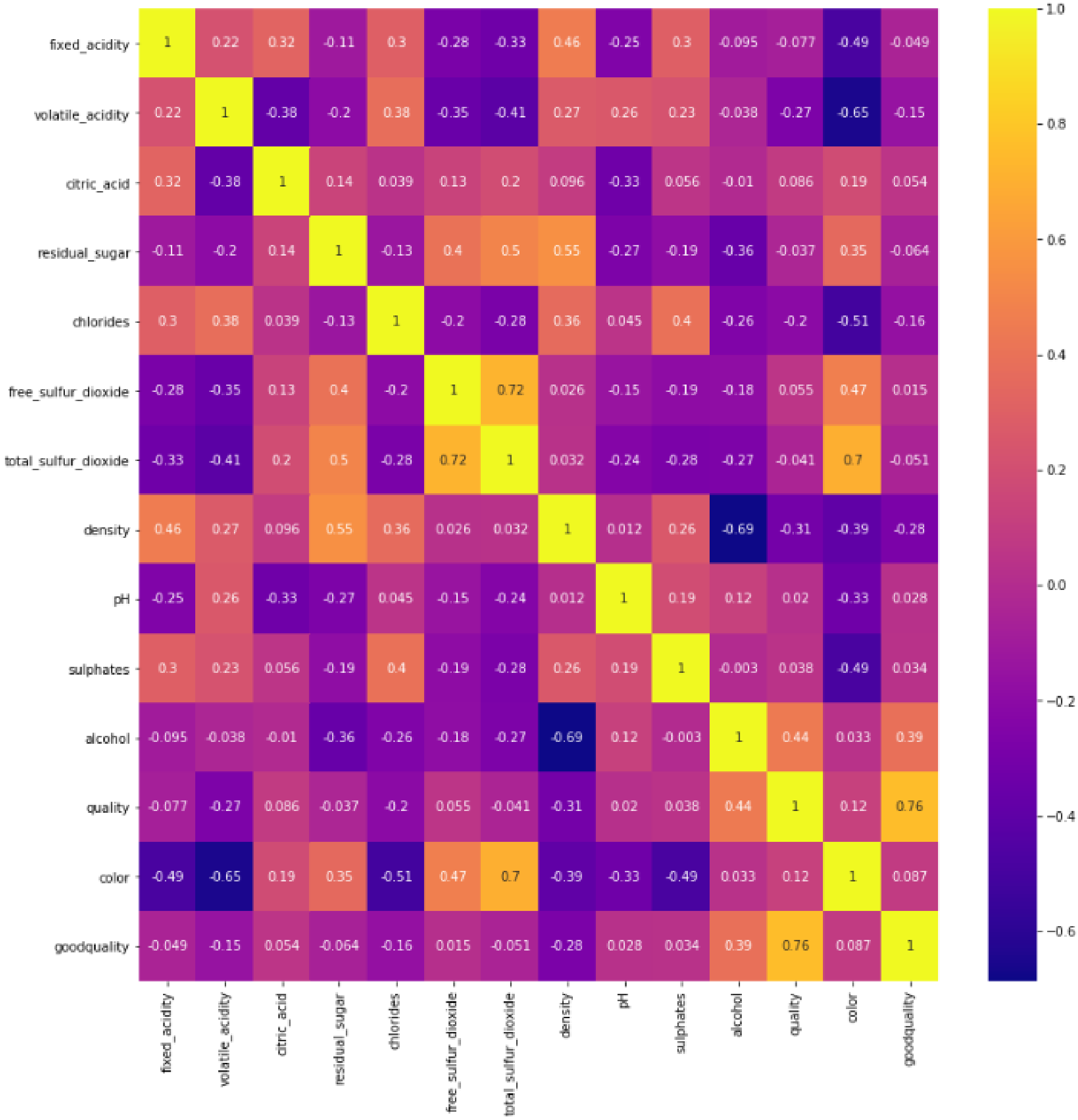
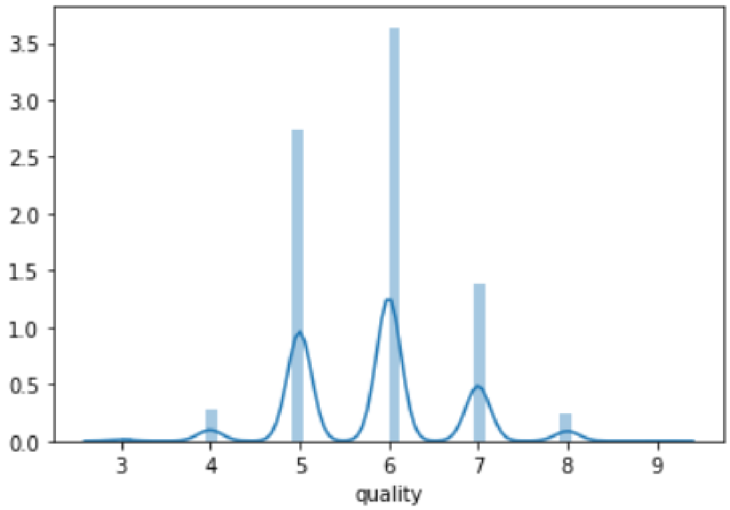


Figure 26: Heat map

```
sns.distplot(df['quality'])
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x2d014402508>



CATPLOT BETWEEN COLOR AND QUALITY

This plots show relationship between numerical variable and one or more categorical variables, like boxplot, stripplot and so

```
In [31]: sns.catplot("color",'quality',data=df,kind="box")
Out[31]: <seaborn.axisgrid.FacetGrid at 0xd516a88>
```

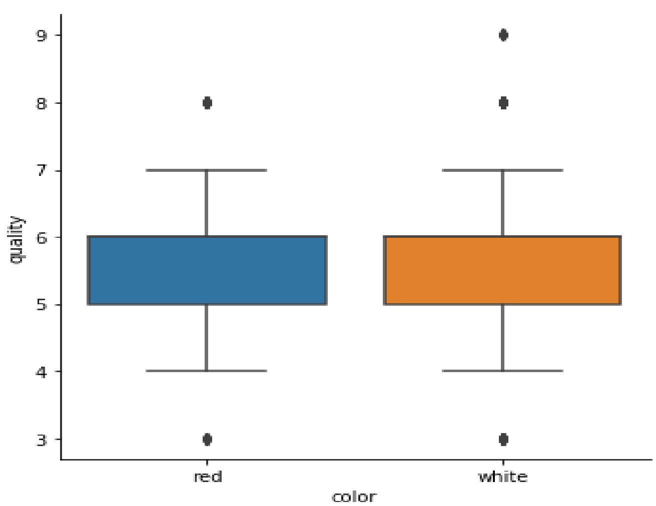


Figure 27:Catplot

CHECKING THE QUALITY MORE THAN 7

```
In [32]: df['quality'] >7

Out[32]: 0       False
         1       False
         2       False
         3       False
         4       False
                 ...
         6492    False
         6493    False
         6494    False
         6495    False
         6496    False
         Name: quality, Length: 6497, dtype: bool
```

Figure 28: Checking quality more than 7

DIFFRENTIATING THE DATA ON GOOD OR BAD

We are here differentiating the data for better understanding.

So here I made good wine quality as more than 7 and bad less than 7 is bad. Here I made clarity of the data by giving checking them

```
In [33]: quality = [6495,7]
         def quality_predict(quality):
             if quality[1]>7:
                 return 'good'
             else:
                 return "bad"

In [34]: quality_predict(quality)

Out[34]: 'bad'

In [35]: quality_predict([6495,7])

Out[35]: 'bad'

In [36]: df['quality'].shape

Out[36]: (6497,)
```

```
In [37]: df[['quality','color']]
```

Out[37]:

	quality	color
0	5	red
1	5	red
2	5	red
3	6	red
4	5	red
...	...	...
6492	6	white
6493	5	white
6494	6	white
6495	7	white
6496	6	white

6497 rows × 2 columns

Figure 29: Differentiating good or bad

DIFFERENTIATING THE QUALITY

```
In [104]: df['goodquality'] = [1 if x >= 7 else 0 for x in df['quality']]
# Separate feature variables and target variable
X = df.drop(['quality','goodquality'], axis = 1)
y = df['goodquality']
```

IF QUALITY MORE THAN 7 '1 ELSE '0

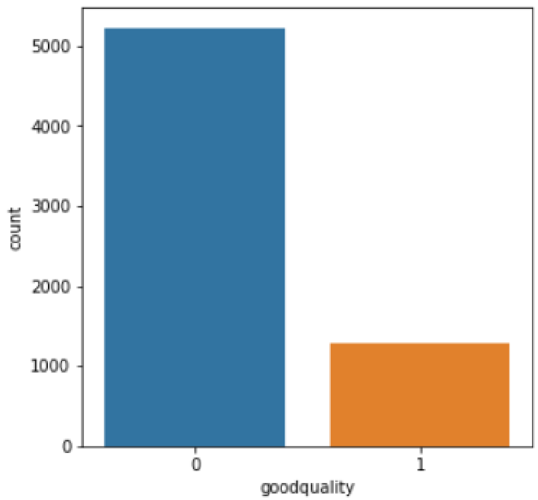
```
In [42]: # See proportion of good vs bad wines
df['goodquality'].value_counts()

Out[42]: 0    5220
1    1277
Name: goodquality, dtype: int64
```

Figure 30: Differentiating the quality

PLOTTING WINE AFTER DIFFRENTIATING THEN ON THE BASIS OF GOODQUALITY

```
#plotting wine
plt.figure(figsize=(5,5))
sns.countplot(df['goodquality'])
```

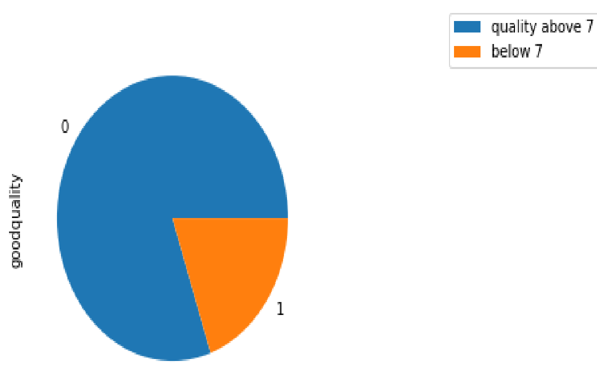
```
: <matplotlib.axes._subplots.AxesSubplot at 0x2c1a36bdf48>
```



PLOTTING THEM ON PIE CHART

```
In [43]: df.goodquality.value_counts().plot.pie().legend(labels=['quality above 7','below 7'], loc='center right', bbox_to_anchor=(2,1))
```

```
Out[43]: <matplotlib.legend.Legend at 0xf4cb908>
```

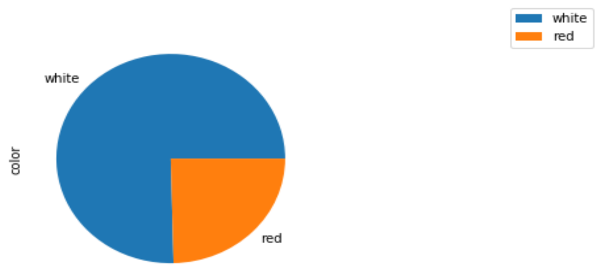


COUNT OF COLOR

```
In [44]: df['color'].value_counts()

Out[44]: white    4898
         red      1599
         Name: color, dtype: int64
```

CHANGED THE WHITE AS '1 RED AS'0

```
In [45]: df.color.value_counts().plot.pie().legend(labels=['white','red'], loc='center right', bbox_to_anchor=(2,1))

Out[45]: <matplotlib.legend.Legend at 0xdb785c8>
```




```
In [46]: df.color.replace(to_replace=['red', 'white'], value=[0, 1],inplace=True)

In [47]: X=df.drop('quality',axis=1)
         y=df.quality
```

Figure 31 :Plotting wine in pie chart after differentiating

Here we made X and Y by dropping the quality and we use this to check

CHECKING THE DATATYPES

```
In [48]: X.dtypes

Out[48]: fixed_acidity           float64
         volatile_acidity        float64
         citric_acid             float64
         residual_sugar          float64
         chlorides               float64
         free_sulfur_dioxide     float64
         total_sulfur_dioxide    float64
         density                 float64
         pH                      float64
         sulphates               float64
         alcohol                 float64
         color                     int64
         goodquality               int64
         dtype: object
```

4.2 TRAINING AND TESTING THE DATA MODEL:

In this we are taking the data of quality in the data set.Here we didn't make any differentiation between quality we are taking the whole quality and testing the data for our better understanding

- Import the train_test_split from model_selection package from scikitlearn library
- Then assigning the output to four different variables, before assigning we have to mention the train size or test size as a parameter to train_test_split. Then this method will split according to the size and assigns it to four variables.

In [49]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

In [50]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(5197, 13)
(5197,)
(1300, 13)
(1300,)
```

Figure 32 : Training and Testing data

Here we are using decision tree classifier and understanding the data here we gave criterion as 'entropy' and we are checking the classification report

```
In [51]: # Apply the Decision Tree Algorithm
         from sklearn.tree import DecisionTreeClassifier

         # intialization of object
         dtree = DecisionTreeClassifier(criterion = 'entropy')

         #Applying the classifier to the dataset
         dtree.fit(X_train,y_train)

Out[51]: DecisionTreeClassifier(criterion='entropy')
```

```
In [52]: # Predict on training data
         y_train_pred=dtree.predict(X_train)
         y_train_pred

Out[52]: array([7, 5, 5, ..., 6, 6, 6], dtype=int64)
```

```
In [46]: df.color.replace(to_replace=['red', 'white'], value=[0, 1],inplace=True)
```

```
In [47]: X=df.drop('quality',axis=1)
         y=df.quality
```

```
In [53]: # Classification Report
         from sklearn.metrics import classification_report,confusion_matrix
         print(classification_report(y_train,y_train_pred))

              precision    recall  f1-score   support

           3       1.00      1.00      1.00        26
           4       1.00      1.00      1.00       168
           5       1.00      1.00      1.00      1703
           6       1.00      1.00      1.00      2259
           7       1.00      1.00      1.00       876
           8       1.00      1.00      1.00       161
           9       1.00      1.00      1.00         4

    accuracy                           1.00      5197
   macro avg       1.00      1.00      1.00      5197
weighted avg       1.00      1.00      1.00      5197
```

Figure 33: Data for Decision Tree Classifier

From this confusion matrix we can check the number of correct and indirect predictions

```
In [54]: confusion_matrix(y_train,y_train_pred)

Out[54]: array([[  26,    0,    0,    0,    0,    0,    0],
               [   0,  168,    0,    0,    0,    0,    0],
               [   0,    0, 1703,    0,    0,    0,    0],
               [   0,    0,    0, 2259,    0,    0,    0],
               [   0,    0,    0,    0,  876,    0,    0],
               [   0,    0,    0,    0,    0,  161,    0],
               [   0,    0,    0,    0,    0,    0,    4]], dtype=int64)
```

Figure 34 : Training and testing

```
In [55]: sns.heatmap(confusion_matrix(y_train,y_train_pred),annot=True)

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0xf64bf48>
```

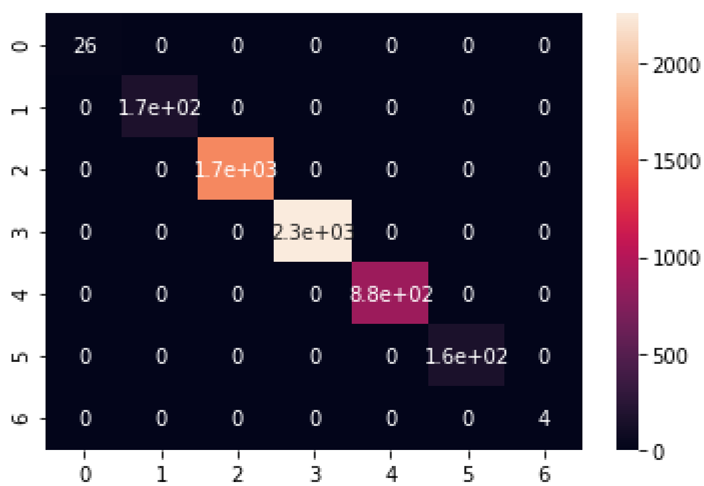


Figure 35: Heat map

From this heat map we can observe that my data are predicted,

Preparing Data for Modeling

Standardizing Feature Variables

```
In [53]: # Create Classification version of target variable
         df['goodquality'] = [1 if x >= 7 else 0 for x in df['quality']]
         # Separate feature variables and target variable
         X = df.drop(['quality','goodquality'], axis = 1)
         y = df['goodquality']

In [54]: # See proportion of good vs bad wines
         df['goodquality'].value_counts()

Out[54]: 0    5220
         1    1277
         Name: goodquality, dtype: int64

In [55]: # Normalize feature variables
         from sklearn.preprocessing import StandardScaler
         X_features = X
         X = StandardScaler().fit_transform(X)
```

Figure 36 : Data for Standardizing feature variables

Split data

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.

- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)
- First we need to identify the input and output variables and we need to separate the input set and output set
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

```
In [56]: # Splitting the data
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state=0)
```

Figure 37 : Splitting data

5. MODELLING

5.1:MODEL 1-DecisionTreeClassifier

We first use model 1 and check the classification report and we can choose weather is this model best for our prediction or no

- Then we need to import DecisionTreeClassifier
- We need to train the model based on our train set (that we have obtained from splitting)

- Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.
- Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Classification And Regression Tree (CART) is the general term for this.

```
In [57]: from sklearn.metrics import classification_report
         from sklearn.tree import DecisionTreeClassifier
         model1 = DecisionTreeClassifier(random_state=1)
         model1.fit(X_train, y_train)
         y_pred1 = model1.predict(X_test)
         print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90      1301
           1       0.59      0.60      0.59       324

    accuracy                           0.84      1625
   macro avg       0.74      0.75      0.74      1625
weighted avg       0.84      0.84      0.84      1625
```

Figure 38 : Decision Tree Classifier

FINDING THE TEST AND THE TRAIN SCORE

```
In [60]: from sklearn.metrics import accuracy_score

In [61]: print('test',accuracy_score(y_pred1,y_test))
         print('train',accuracy_score(y_train,model1.predict(X_train)))

         test 0.8356923076923077
         train 1.0
```

Figure 39: Finding Test and train score

FINDING THE SCORE AND THE F1 SCORE

```
In [155]: print("score",model1.score(X_test,y_test))
          print("F1_score:{}".format(f1_score(y_test,y_pred1)))

          score 0.8356923076923077
          F1_score:0.5923664122137405
```

Figure 40: Finding f1 score

FINDING THE MEAN SCORE

```
In [63]: from sklearn.model_selection import cross_val_score
         score=cross_val_score(model1,X_train,y_train,cv=5)
         np.mean(score)

Out[63]: 0.8265622071289422
```

Figure 41: Finding mean score

As we got the all the values of this model but we didn't get the most nearest value so we should try different models for best score

5.2:MODEL 2-RandomForestClassifier

We first use model 2 and check the classification report and we can choose weather is this model best for our prediction or no

- Then we need to importRandomForestClassifier
- We need to train the model based on our train set (that we have obtained from splitting)
- A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- Random forest uses gini importance or mean decrease in impurity (MDI) to calculate the importance of each feature. Gini importance is also known as the total decrease in node impurity.
- Random forests also offers a good feature selection indicator. Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction

```
In [64]: from sklearn.ensemble import RandomForestClassifier
         model2 = RandomForestClassifier(random_state=1)
         model2.fit(X_train, y_train)
         y_pred2 = model2.predict(X_test)
         print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      1301
           1       0.80      0.54      0.64       324

    accuracy                           0.88      1625
   macro avg       0.84      0.75      0.79      1625
weighted avg       0.87      0.88      0.87      1625
```

Figure 42: Random tree Classifier

FINDING THE TEST AND THE TRAIN SCORE

```
In [65]: print('test',accuracy_score(y_pred2,y_test))
         print('train',accuracy_score(y_train,model2.predict(X_train)))

test 0.8806153846153846
train 1.0
```

Figure 43: Test and train score

FINDING THE SCORE AND THE F1 SCORE

```
In [132]: print("score",model2.score(X_test,y_test))
          print("F1_score:{}".format(f1_score(y_test,y_pred2)))

score 0.8806153846153846
F1_score:0.6433823529411764
```

Figure 44 : Finding f1 score

FINDING THE MEAN SCORE

```
In [67]: from sklearn.model_selection import cross_val_score
         score=cross_val_score(model2,X_train,y_train,cv=5)
         np.mean(score)

Out[67]: 0.8754109408729533
```

Figure 45 : Finding mean score

5.3: MODEL 3-AdaBoostClassifier

We first use model 3 and check the classification report and we can choose weather is this model best for our prediction or no

- Then we need to importAdaBoostClassifier
- We need to train the model based on our train set (that we have obtained from splitting)
- An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
- AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations

```
In [68]: from sklearn.ensemble import AdaBoostClassifier
         model3 = AdaBoostClassifier(random_state=1)
         model3.fit(X_train, y_train)
         y_pred3 = model3.predict(X_test)
         print(classification_report(y_test, y_pred3))

              precision    recall  f1-score   support

           0       0.86      0.93      0.89      1301
           1       0.59      0.39      0.47       324

    accuracy                           0.82      1625
   macro avg       0.72      0.66      0.68      1625
weighted avg       0.81      0.82      0.81      1625
```

Figure 46 : AdaBoost Classifier

FINDING THE TEST AND THE TRAIN SCORE

```
In [163]: print('test',accuracy_score(y_pred3,y_test))
          print('train',accuracy_score(y_train,model3.predict(X_train)))

          test 0.824
          train 0.8327175697865353
```

Figure 47 : Test and train score

FINDING THE SCORE AND THE F1 SCORE

```
In [156]: print("score",model3.score(X_test,y_test))
          print("F1_score:{}".format(f1_score(y_test,y_pred3)))

          score 0.824
          F1_score:0.46840148698884765
```

Figure 48 : Finding f1 score

FINDING THE MEAN SCORE

```
In [71]: from sklearn.model_selection import cross_val_score
         score=cross_val_score(model3,X_train,y_train,cv=5)
         np.mean(score)

Out[71]: 0.820405623124309
```

Figure 49 : Finding mean score

5.4: MODEL 4- Gradient Boosting Classifier

We first use model 3 and check the classification report and we can choose weather is this model best for our prediction or no

- Then we need to importGradient BoostingClassifier
- GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.
- Gradient boosting classifiers are specific types of algorithms that are used for classification tasks, as the name suggests.
- Because the labels contain the target values for the machine learning classifier, when training a classifier you should split up the data into training and testing sets.

We need to train the model based on our train set (that we have obtained from splitting

```
In [72]: from sklearn.ensemble import GradientBoostingClassifier
         model4 = GradientBoostingClassifier(random_state=1)
         model4.fit(X_train, y_train)
         y_pred4 = model4.predict(X_test)
         print(classification_report(y_test, y_pred4))

              precision    recall  f1-score   support

           0       0.87      0.95      0.91      1301
           1       0.68      0.41      0.52       324

    accuracy                           0.84      1625
   macro avg       0.78      0.68      0.71      1625
weighted avg       0.83      0.84      0.83      1625
```

Figure 50 : Gradient Boosting Classifier

FINDING THE TEST AND THE TRAIN SCORE

```
In [73]: print('test',accuracy_score(y_pred4,y_test))
         print('train',accuracy_score(y_train,model4.predict(X_train)))

         test 0.8449230769230769
         train 0.8754105090311987
```

Figure 51 : Test and train score

FINDING THE SCORE AND THE F1 SCORE

```
In [84]: print("score",model4.score(X_test,y_test))
         print("F1_score:{}".format(f1_score(y_test,y_pred4)))

         score 0.8449230769230769
         F1_score:0.5153846153846154
```

Figure 52 : Finding f1 score

FINDING THE MEAN SCORE

```
In [75]: from sklearn.model_selection import cross_val_score
         score=cross_val_score(model4,X_train,y_train,cv=5)
         np.mean(score)

Out[75]: 0.8376448165113463
```

Figure 53 : Finding mean score

5.5: MODEL-5-XGBClassifier

We first use model 3 and check the classification report and we can choose weather is this model best for our prediction or no

- Then we need to importXGBClassifier
- **XGBoost** is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework
- XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.
- XGBoost is an advanced version of gradient boosting
- XGBoost is a more advanced version of the gradient boosting method. The main aim of this algorithm is to increase speed and to increase the efficiency of your competitions

We need to train the model based on our train set (that we have obtained from splitting

In [77]:
```
import xgboost as xgb
model5 = xgb.XGBClassifier(random_state=1)
model5.fit(X_train, y_train)
```

Out[77]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [78]:
```
y_pred5 = model5.predict(X_test)
print(classification_report(y_test, y_pred5))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      1301
           1       0.72      0.57      0.64       324

    accuracy                           0.87      1625
   macro avg       0.81      0.76      0.78      1625
weighted avg       0.86      0.87      0.87      1625
```

Figure 54 : XGB Classifier

FINDING THE TEST AND THE TRAIN SCORE

In [79]:
```
print('test',accuracy_score(y_pred5,y_test))
print('train',accuracy_score(y_train,model5.predict(X_train)))
```

```
test 0.8707692307692307
train 0.9979474548440066
```

Figure 55 : Test and train score

FINDING THE SCORE AND THE F1 SCORE

```
In [164]: from sklearn.metrics import f1_score
          model5.fit(X_train,y_train)
          print("score",model5.score(X_test,y_test))
          print("F1_score:{}".format(f1_score(y_test,y_pred5)))

          score 0.8707692307692307
          F1_score:0.6391752577319588
```

Figure 56 : Finding f1 score

FINDING THE MEAN SCORE

```
In [81]: # Kfold CV:
         from sklearn.model_selection import cross_val_score
         a =cross_val_score(model5,X_train,y_train,cv=5)

In [82]: np.mean(a)

Out[82]: 0.8725385141894382
```

Figure 57 : Finding mean score

AFTER GETTING ALL THE VALUES CHECKING THE BEST MEAN BY TAKING THE BARGRAPH

FROM THE BARGRAPH WE CAN OBSERVE THAT MODEL 2 WHICH IS RANDOM FORESTCLASSIFIER =0.875 HAS HIGHEST ACCURACY

```
In [214]: models=['model1', 'model2', 'model3', 'model4', 'model5','grid_search']
          mean=[0.826,0.875,0.820,0.837,0.872,0.811]
          plt.bar(models,mean,color="GREEN")
          plt.xlabel("MODELS")
          plt.ylabel("accuracy score")
          plt.show()
```

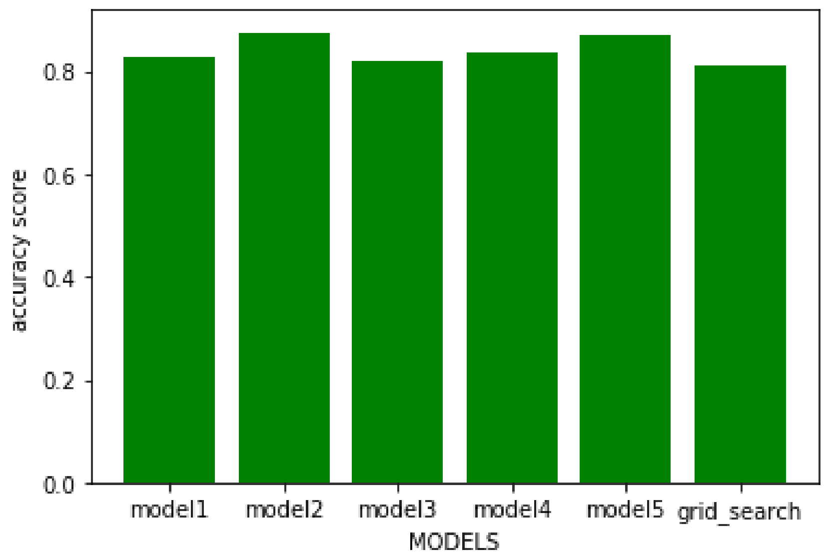


Figure 58 : Best mean score in bar Graph

- **AS THE DATA IS UNBALANCED DATA WE SHOULD CHECK THE F1 SCORE AND CHOOSE THE MODEL**
- **AFTER GETTING ALL THE VALUES CHECKING THE BEST F1 SCORE BY TAKING THE BARGRAPH**

FROM THE BARGRAPH WE CAN OBSERVE THAT MODEL 2 WHICH IS RANDOM FOREST CLASSIFICATION HAS HIGHEST F1=0.64

In [168]:
```
models=['model1', 'model2', 'model3', 'model4', 'model5',]
mean=[0.592,0.64,0.46,0.1,0.63,]
plt.bar(models,mean,color="RED")
plt.xlabel("MODELS")
plt.ylabel("F1 score")
plt.show()
```

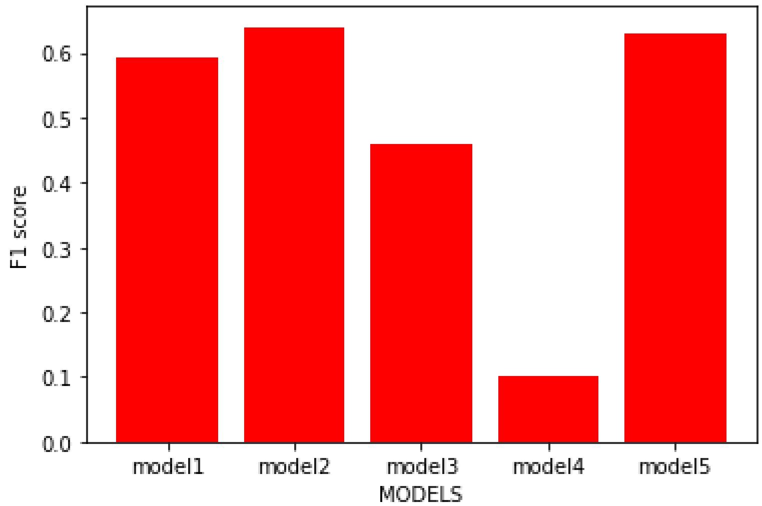


Figure 59 : Highest result

5.6: APPLYING GRIDSEARCHCV TO INCREASE THE ACCURACY

- GridSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.
- The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.
- Two generic approaches to sampling search candidates are provided in scikit-learn: for given values,gridsearchcv exhaustively considers all parameter combinations

```
In [92]: from sklearn.model_selection import GridSearchCV
         #initialization
```

```
In [93]: # Hyperparameters
         # GridSearchCV--> find optimum parameters
         grid_param={'criterion' : ['gini','entropy'],
                     'max_depth' : range(3,5,7),
                     'min_samples_leaf' : range(1,10,12)}
```

```
In [94]: clf = RandomForestClassifier()
         grid_search = GridSearchCV(estimator=model2,scoring="accuracy",param_grid = grid_param)
```

Figure 60 : Importing grid search

APPLYING GRIDSEARCH ONTO DATA SET

```
In [189]: grid_search=grid_search.fit(X_train,y_train)
```

```
In [190]: accuracy=grid_search.best_score_
```

```
In [191]: y_pred6 = grid_search.predict(X_test)
          print(classification_report(y_test, y_pred6))

              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1301
           1       0.60      0.03      0.05       324

    accuracy                           0.80      1625
   macro avg       0.70      0.51      0.47      1625
weighted avg       0.76      0.80      0.72      1625
```

Figure 61 : Applying grid search on dataset

THE ACCURACY WE OBSERVE FROM GRID SEARCHCV

```
In [97]: accuracy

Out[97]: 0.8117822355604696

In [98]: grid_search.best_params_

Out[98]: {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1}
```

Figure 62 : Accuracy from grid search

- Even after applying gridsearchcv the accuracy is not increased so we don’t need this

THE TRAIN AND TEST VALUES

```
In [225]: acc = {'model1_train':y_train,'model1_test':X_test,'model2_train':y_train,'model2_test':X_test,
                 'model3_train':y_train,'model3_test':X_test,
                 'model4_train':y_train,'model4_test':X_test,
                 'model5_train':y_train,'model5_test':X_test}
          x = acc.keys()
          y = acc.values()

In [226]: x

Out[226]: dict_keys(['model1_train', 'model1_test', 'model2_train', 'model2_test', 'model3_train', 'model3_test', 'model4_train', 'model4
          _test', 'model5_train', 'model5_test'])
```

```
In [227]: y

Out[227]: dict_values([3731    0
          1651    1
          5060    1
          1303    0
          474     0
                 ..
          4931    1
          3264    1
          1653    0
          2607    1
          2732    0
          Name: goodquality, Length: 4872, dtype: int64, array([[-0.70607349, -0.96988404, -0.54115347, ..., -0.07573051,
                  -1.08316218,  0.57136659],
                 [-0.32037042, -1.21286227, -0.95407324, ...,  0.05868313,
                   0.59381798,  0.57136659],
                 [ 0.06533265, -0.72690581,  1.24816549, ...,  0.86516498,
                  -1.2508602 ,  0.57136659],
                 ...,
                 [ 0.8367388 , -0.6054167 , -0.33469359, ...,  0.99957862,
```

Figure 63 : Test and train values

We have to build the model with best parameters so we are using Random Forest Classifier

From the comparison of the f1 score we decide that random forest classifier is the best model to apply for the prediction

Prediction on test data

```
In [99]: clf = RandomForestClassifier(criterion='gini',max_depth=12,min_samples_leaf=1)

         # We need to fit the model to the data
         clf.fit(X_train,y_train)

Out[99]: RandomForestClassifier(max_depth=12)
```

```
In [100]: pred_test = clf.predict(X_test)

          #Classification Report of actual values
          print(classification_report(y_test,pred_test))

                       precision    recall  f1-score   support

                    0       0.89      0.97      0.93      1301
                    1       0.83      0.52      0.64       324

             accuracy                           0.88      1625
            macro avg       0.86      0.75      0.79      1625
         weighted avg       0.88      0.88      0.87      1625
```

```
In [106]: print("F1_score:{}".format(f1_score(y_test,pred_test)))

          F1_score:0.6415094339622641
```

Figure 64 : Prediction on test set

Building a model using Random Forest Classifier

After training the model, we need to test the performance of it with some unseen data.

```
In [139]: from sklearn.ensemble import RandomForestClassifier
          rfc = RandomForestClassifier(n_estimators = 100)
          rfc.fit(X_train,y_train)

Out[139]: RandomForestClassifier()
```

Figure 65 : Building a model using random tree classifier

5.7: PREDICTING THE DATA BY GIVING UNKNOWN VALUES

We need to give all the new observations to check

Here we can clearly observe that array[0] as our output.

```
In [193]: input_sample = [[7.4 ,0.59 ,0.08 ,4.4,0.086,6,29,0.9974,3.38,0.5,9,1]]
          rfc.predict(input_sample)

Out[193]: array([0], dtype=int64)
```

Figure 66 : Predicting by unknown value

Why Random Forest Classifier is chosen?

- We took RandomForestclassifier and predicted for the unknown values
- We took this model because from all the model's we tried we got the best f1 score and accuracy for this model so we choose this model.

CHAPTER 5

CONCLUSION

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) and its come to point of getting the solution for the problem statement being , that prediction of wine quality is clearly determined by using RandomForestlasifier . We have predicted which quality of wine is of goodquality by performing the various test models and choose the best one. By this I can conclude that my problem statement is solved.

CHAPTER 6

REFERENCES

[1] https://www.kaggle.com/semakulapaul/cereals-dataset

[2] https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676

[3] https://en.wikipedia.org/wiki/Machine_learning

GitHub link: https://github.com/dheerajbamar/Project-Winequality