MACHINE LEARNING

(WINE QUALITY PREDICTION)

Summer Internship Report submitted in partial fulfilment of the requirement of for the undergraduate Degree of

Bachelor of Technology

In

Computer Science Engineering

By

Dubba Srikanth Reddy

221710315052

Under the guidance of

GITAM

(DEEMED TO BE UNIVERSITY)

VISAKHAPATNAM ✺ HYDERABAD ✺ BENGALURU

Department Of Computer Science Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
July 2020

DECLARATION

I submit this industrial training work entitled “WINE QUALITY PREDICATION” 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 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

StudentName

Dubba Srikanth Reddy

StudentRollNo

221710315052

GITAM (DEEMED TO BE UNIVERSITY)

UN Hyderabad-502329, India

CERTIFICATE

This is to certify that the Industrial Training Report entitled "WINE QUALITY PREDICTION" is being submitted by Dubba Srikanth Reddy (221710315052) in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-2020.

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

ABSTRACT

Machine learning is one of the emerging areas of research. Many algorithms of data mining have already been used on wine quality dataset to analyze the wine attributes such as quality or class. The quality of wine is not only based on the quantity of alcohol but it also depends on various attributes, these attributes changes with time and so the quality of wine also refines. In this report, machine learning techniques are utilized to analyze those attributes. Firstly data pre-processing takes place i.e. making data appropriate for the models that are built for prediction. Defining independent and dependent variables, missing data handling, feature scaling and data splitting is done to improve the data standard. Then, Decision Tree, Logistic regression and Random forest classifier are performed individually on data to predict the test data values.

Table of Contents

LIST OF FIGURES

CHAPTER 1 MACHINE LEARNING

1.1 INTRODUCTION:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

1.2 IMPORTANCE OF MACHINE LEARNING:

Machine learning has several very practical applications that drive the kind of real business results – such as time and money savings – that have the potential to dramatically impact the future of your organization. At Interactions in particular, we see tremendous impact occurring within the customer care industry, whereby machine learning is allowing people to get things done more quickly and efficiently. Through Virtual Assistant solutions, machine learning automates tasks that would otherwise need to be performed by a live agent – such as changing a password or checking an account balance. This frees up valuable agent time that can be used to focus on the kind of customer care that humans perform best: high touch, complicated decision-making that is not as easily handled by a machine. At Interactions, we further

Improve the process by eliminating the decision of whether a request should be sent to a human or a machine: unique Adaptive Understanding technology, the machine learns to be aware of its limitations, and bail out to humans when it has a low confidence in providing the correct solution.

Machine learning has made dramatic improvements in the past few years, but we are still very far from reaching human performance. Many times, the machine needs the assistance of human to complete its task. At Interactions, we have deployed Virtual Assistant solutions that seamlessly blend artificial with true human intelligence to deliver the highest level of accuracy and understanding.

1.3 MACHINE LEARNING VS TRADITIONAL PROGRAMMING

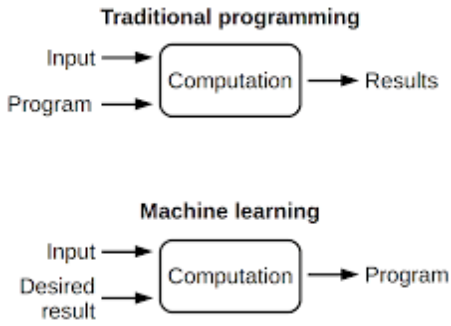


FIGURE 1- 1.3.1 MACHINE LEARNING VS TRADITIONAL PROGRAMMING

• Traditional programming is a manual process i.e. a programmer writes the program and passes the input to get the desired output.

• Here the programmer has to first develop an algorithm and then implement that algorithm in to the code and then use it by passing input values to get the output.

• Whereas in Machine Learning, we don't need to write the program but we need to collect the data that is used to build the model and now pass the input to the model and it gives the computed output.

• So, basically the difference between traditional programming and machine learning is that without anyone programming the logic, In Traditional programming one has to manually formulate/code rules while in Machine Learning the algorithms automatically formulate the rules from the data, which is very powerful.

1.4 USES OF MACHINE LEARNING:

Artificial Intelligence is everywhere, there is a possibility that we are using it in one way or another and we don't even realize it. It has many applications in the real world and some of them are listed below:

• **Virtual Assistants**: Machine learning is an important part of these personal assistants as they collect and refine the information on the basis of your previous involvement with them.

• **Predictions while commuting:** *Traffic Predictions*: We all have been using GPS navigation services. While we do that, our current locations and velocities are being saved at a central server for managing traffic. This data is then used to build a map of current traffic. Machine learning in such scenarios helps to estimate the regions where congestion can be found on the basis of daily experiences.

• **Email spam and Malware filtering:** ML can be used to filter spam messages from your inbox. It can also be used to detect malware programs as each piece of malware code is about 97-98% similar to its previous versions.

• **Online customer support:** Companies can use AI to answer the queries of the customers.

• **Product recommendations:** It can also recommend us what content and products we might use based on our history of usage.

• **Online fraud Detection:** It can also be used to detect and track any of the fraudulent transactions happening in the internet.

1.5 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.

1.5.1 SUPERVISED LEARNING:

"*The outcome or output for the given input is known before itself*" and the machine must be able to map or assign the given input to the output. Multiple images of a cat, dog, orange, apple etc here the images are labeled. It is fed into the machine for training and the machine must identify the same. Just like a human child is shown a cat and told so, when it sees a completely different cat among others still identifies it as a cat, the same method is employed here. It is called supervised learning because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance. Supervised learning problems can be further grouped into regression and classification problems.

• **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.

• **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

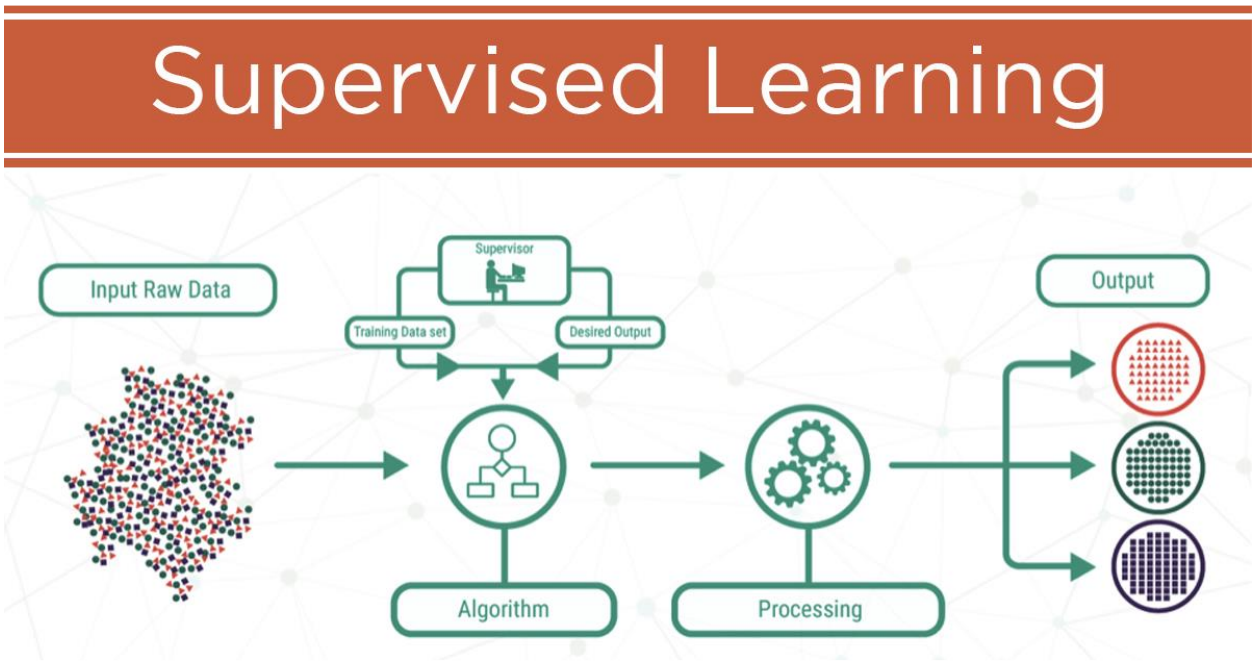


FIGURE 2- 1.5.1.1 SUPERVISED LEARNING

1.5.2 UNSUPERVISED LEARNING:

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabelled data by our-self.

For instance, suppose it is given an image having both dogs and cats which have not seen ever. Thus the machine has no idea about the features of dogs and cat so we can’t categories it in dogs and cats. But it can categories them according to their similarities, patterns, and differences i.e., we can easily categories the above picture into two parts. First first may contain all pics having

dogs in it and second part may contain all pics having **cats** in it. Here, you didn't learn anything before, means no training data or examples.

Unsupervised learning classified into two categories of algorithms:

• **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

• **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

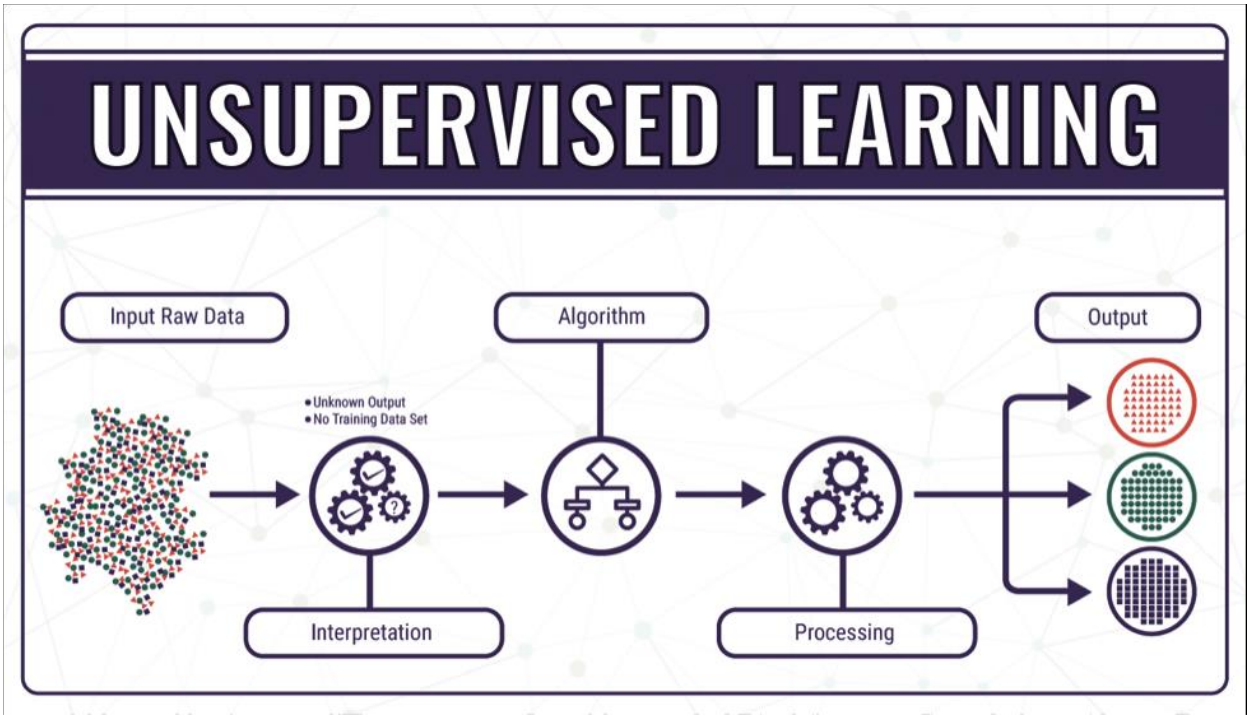


FIGURE 3- 1.5.2.1 UNSUPERVISED LEARNING

1.5.3 SEMI SUPERVISED LEARNING

The most basic disadvantage of any Supervised Learning algorithm is that the dataset has to be hand-labeled either by a Machine Learning Engineer or a Data Scientist. This is a very *costly process*, especially when dealing with large volumes of data. The most basic disadvantage of any Unsupervised Learning is that its application spectrum is limited.

To counter these disadvantages, the concept of Semi-Supervised Learning was introduced. In this type of learning, the algorithm is trained upon a combination of labeled and unlabelled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabelled data. The basic procedure involved is that first, the programmer will cluster similar data using an unsupervised learning algorithm and then use the existing labeled data to label the rest of the unlabelled data. The typical use cases of such type of algorithm have a common property among them – The acquisition of unlabelled data is relatively cheap while labeling the said data is very expensive.

A Semi-Supervised algorithm assumes the following about the data

1. Continuity Assumption: The algorithm assumes that the points which are closer to each other are more likely to have the same output label.

2. Cluster Assumption: The data can be divided into discrete clusters and points in the same cluster are more likely to share an output label.

3. Manifold Assumption: The data lie approximately on a manifold of much lower dimension than the input space. This assumption allows the use of distances and densities which are defined on a manifold.

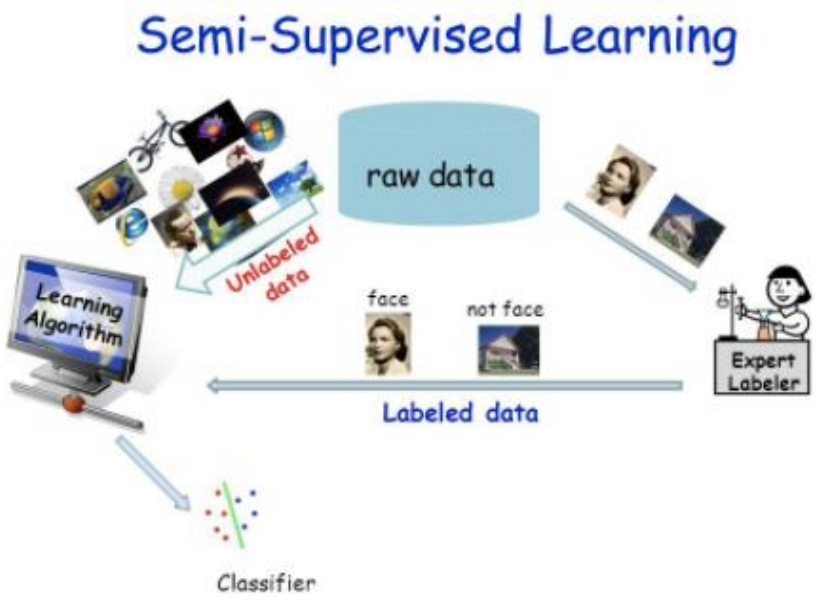


FIGURE 4- 1.5.3.1 SEMI SUPERVISED LEARNING

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

Data mining is used on an existing dataset (like a data warehouse) to find patterns. Machine learning, on the other hand, is trained on a 'training' data set, which teaches the computer how to make sense of data, and then to make predictions about new data sets.

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 PYTHON:

- 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 code within objects.

2.2 HISTORY 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.
- In January 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.
- Python 2.0 was released on 16 October 2000 with many major new features
- Python 3.0 was released on 3 December 2008

2.3 FEATURES OF PYTHON:

- Python uses dynamic data type and a combination of reference counting and a cycle-detecting garbage collector for memory management.

- It also features dynamic name resolution, which binds method and variable names during program execution.

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax,

- This allows the student to pick up the language quickly.

- Easy-to-read: Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain: Python source code is fairly easy-to-maintaining.

- Databases: Python provides interface to all major commercial database.

- 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

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS. 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

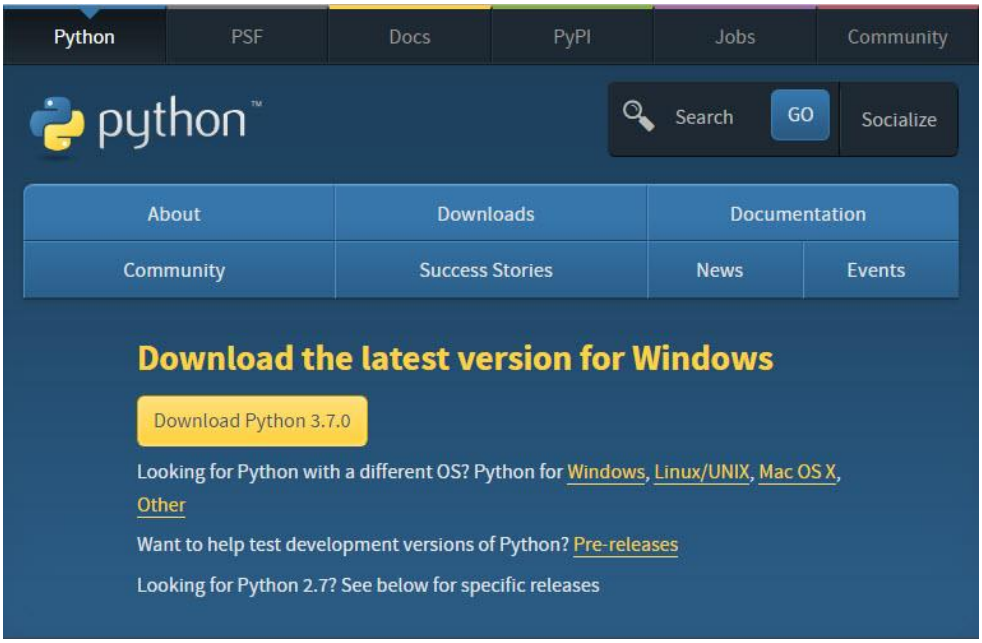


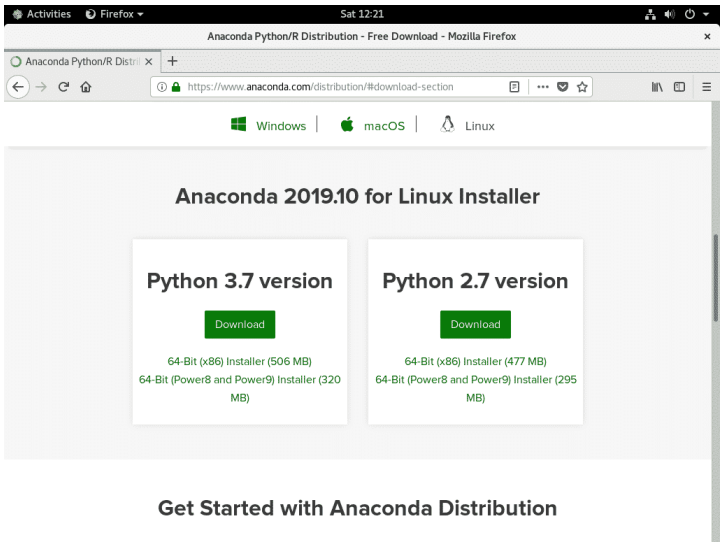
FIGURE 5- 2.4.1.1 INSTALLATION (USING PYTHON IDLE)

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 Jupiter notebook (it opens in default browser)



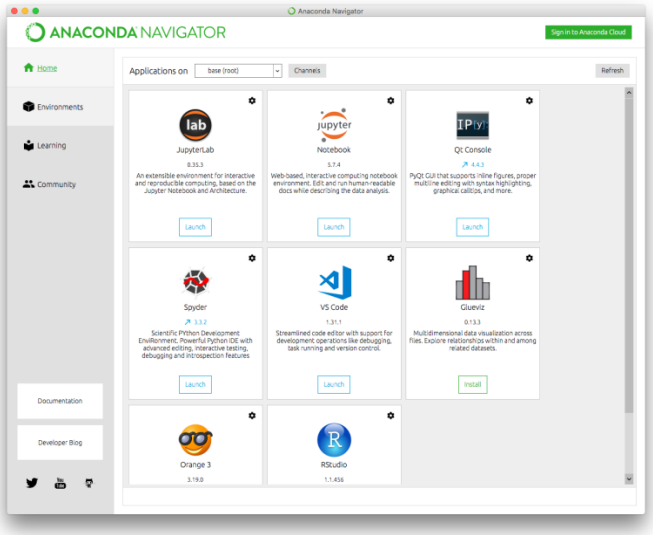


FIGURE 6- 2.4.2.1 INSTALLATION (USING ANACONDA)

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 PYTHYON NUMBERS:

- Number data types store numeric values. Number objects are created when you assign a 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 in the 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.

2.5.3 PYTHON LISTS:

- Lists are the most versatile of Python's compound data types.

- 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 data type.

- 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 a list.

- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

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 variables and 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 variables are.

- Data member: A class variable or instance variable that holds data associated with a 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.

```
>>> class Math:
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def add(self):
                return self.x + self.y
        def subtract(self):
                return self.x - self.y

>>> class Math2:
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def multiply(self):
                return self.x * self.y
        def divide(self):
                return self.x / self.y

>>>
```

FIGURE 7- 2.7.1.1 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:

In this project I wanted to compare several classification algorithms to predict wine quality which has a score between 0 and 10.I decided to compare and select an algorithm to find out what makes a good wine by using wine quality –wine.csv data sourced from the Kaggle.
We try to find the feature and quality of wine, to help manufacture to select the Best quality of Wine Quality Labels
0-5 = low
5-7 = medium
8-10 = high
To predict the accuracy of the wine quality data recognized by different algorithms and choose the best one in it.

3.2 DATA SET:

The inputs are physicochemical information, such as PH values, and the output is based on sensory data which is the median of at least 3 evaluations made by wine experts. Each expert graded the wine quality between 0 (very bad) and 10 (excellent).

The dataset provider alleges that due to privacy and logistic issues, only physicochemical and sensory variables are available.

There's no information about how the dataset was created, such as the distribution of grape types, wine brands and whether the same experts graded all wines. This information is important to identify possible bias that may distort the analysis results. For example, the physicochemical properties may vary among different wines of the same type, affecting the distribution in the dataset.

HOW IT WORKS?

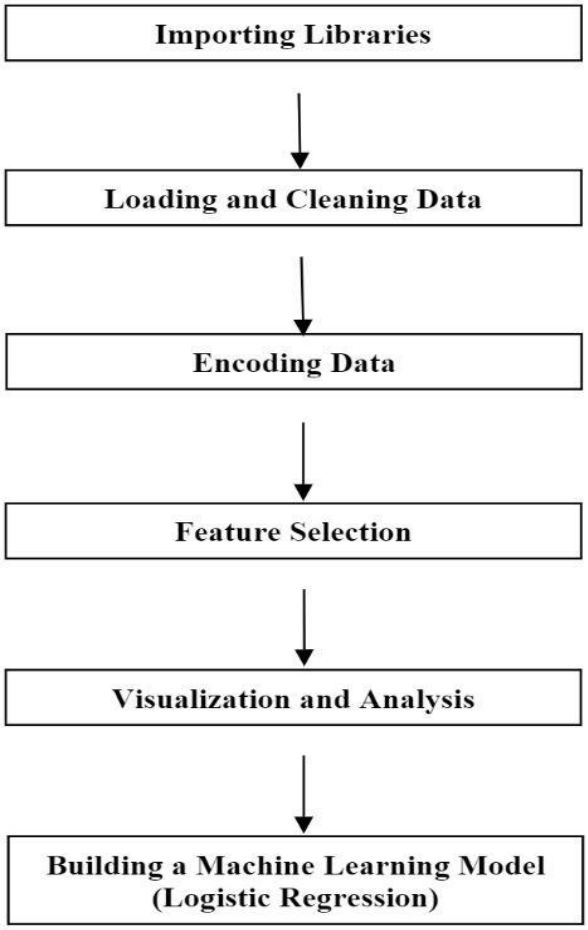


FIGURE 8- 3.2.1 HOW IT WORKS?

3.3 OBJECTIVE OF THE CASE STUDY:

Input variables:

1 - **Fixed acidity**: - The total acidity is divided into two groups: the volatile acids and the nonvolatile or fixed acids. The value of this variable is represented by in gm/dm3 in the data sets.

2 - **Volatile acidity**: - The volatile acidity is a process of wine turning into vinegar. In this data sets, the volatile acidity is expressed in gm/dm3

3 - **Citric acid**: - Citric acid is one of the fixed acids in wines. It's expressed in g/dm3 in the data sets.

4 - **Residual sugar:** - Residual Sugar is the sugar remaining after fermentation stops, or is stopped. It's expressed in g/dm3 in the data set.

5 – **Chlorides**: - can be an important contributor to saltiness in wine. The value of this variable is represented by in gm/dm3 in the data sets.

6 - **Free sulfur dioxide**: - It is the part of the sulfur dioxide that is added to a wine. The value of this variable is represented by in gm/dm3 in the data sets.

7 - **Total sulfur dioxide**:- It is the sum of the bound and the free sulfur dioxide. The value of this variable is represented by in gm/dm3 in the data sets.

8 – **Density**: - the density of water is close to that of water depending on the percent alcohol and sugar content

9 – **PH**: - describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the

10 – **Sulphates**:- a wine additive which can contribute to sulfur dioxide gas (S02) levels, which acts as an antimicrobial and

11 – **Alcohol**:- the percent alcohol content of the wine

Output variable (based on sensory data):
12 - **quality** (score between 0 and 10

CHAPTER 4 MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:
Label Encoding is used to convert the labels into numeric form so as to convert it into the machine-readable form. It is an important pre-processing step for the structured dataset in supervised learning. We have used label encoding to label the quality of data as low, medium, high Assigning 0 to 5 as low. 6 to 7 as medium, 8 to 10 as high

4.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from client.

The Dataset which are considered in this notebook file are taken from Kaggle. The obtained dataset has been randomly partitioned into two sets, where 70% is training data and 30% is testing data.

The dataset that is used in this project has been taken from kaggle from the following link:-
https://drive.google.com/file/d/1yx3Re0jQwGY7Gb_tTvIHc9ot5nBn_kCN/view?usp=sharing

4.1.2 IMPORTING THE LIBRARIES:

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

- Importing the necessary packages and modules
- numpy package can be used to perform mathematical operations like 'mean'.
- pandas package can be used to process dataframes.
- Sea born package can be used to visualise data in the form of various effective graphs and plots.
- Sklearn is the main package which is used for machine learning.
- LabelEncoder is used to encode the non-numeric data into numericals so that machine learning model can be built.
- train_test_split module is used to split the data into training and testing sets.
- LinearRegression module is used to fit a LinearRegression model.
- Sklearn.metrics can be used to calculate statistical results like mean squared error, root mean squared error, etc.

Installing Libraries

```
[68] import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline
```

FIGURE 9- 4.1.2.1 IMPORTING THE LIBRARIES

4.1.3 IMPORTING OF DATASET:

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 NaN value have to be cleaned.

```
[69] df = pd.read_csv("/content/drive/My Drive/WINE QUALITY PREDICTION PROJECT/winequality-red.csv")
```

```
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

FIGURE 10- 4.1.3.1 IMPORTING OF DATASET

Reading the dataset using read_csv(“).

4.1.4 SHAPE OF DATASET:

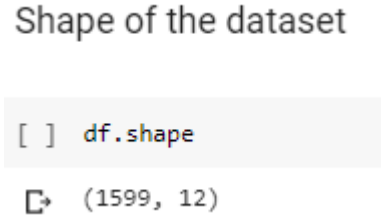


FIGURE 11- 4.1.4.1 SHAPE OF DATASET

In the above dataset we have 1599 rows and 12 columns.

4.1.5 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

(A) dropna():- dropna() is a function which drops all the rows and columns which are having the missing values (i.e. NaN)

- dropna() function has a parameter called how which works as follows
- if how = ’all’ is passed then it drops the rows where all the columns of the particular row are missing
- if how = ’any’ is passed then it drops the rows where all the columns of the particular row are missing

(B)fillna():- fillna() is a function which replaces all the missing values using different ways.

- fillna() also have parameters called method and axis
- if we use method = ’ffill’ where ffill is a method called forward fill, which carry forwards the previous row’s value
- if we use method = ‘bfill’ where bfill is a method called backward fill, which carry backward the next row’s value
- if we use method = ‘ffill’ , axis = ‘columns’ then it carry forwards the previous column’s value
- if we use method = ‘bfill’ , axis = ‘columns’ then it carry backward the next column’s value

(C) Interpolate ():

- interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values

(D) Mean imputation and median imputation:

- Mean and median imputation can be performed by using fillna().
- Mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.
- Median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

4.1.6CHECKING THE NULL VALUES

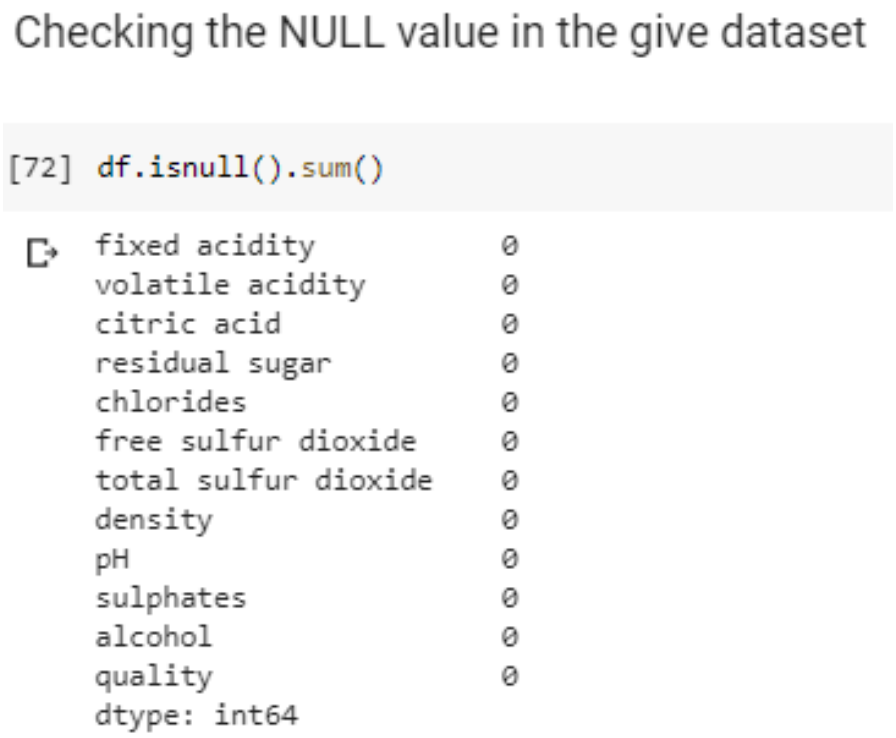


FIGURE 12- 4.1.6.1 CHECKING THE NULL VALUES

- The red wine dataset doesn't have any missing values/rows/cells for any of the variables/feature.
- It seems that data has been collected neatly or prior cleaning has been performed before publishing the dataset

Scaling:-a dataset usually produces better dataset and more accurate predictions.
First we check the range (the min and the max) for each of the datasets.
Let's try using the .describe () method and lets exclude the activity column which is the last column.
It gives the max, min, mean, count, 25%, 50%, 75%.

Transpose of a Description Dataset:

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

FIGURE 13- 4.1.6.2 Transpose of a Description Dataset

4.1.7 INFORMATION AND DATATYPE

Information of the given data

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 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  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

FIGURE 14- 4.1.7.1 INFORMATION AND DATATYPE

It gives the information of the every column in the dataset and gives its data type.

4.1.8 Counting the Quality:-

```
plt.figure(figsize=(10, 6))
sns.countplot(df["quality"])
df["quality"].value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

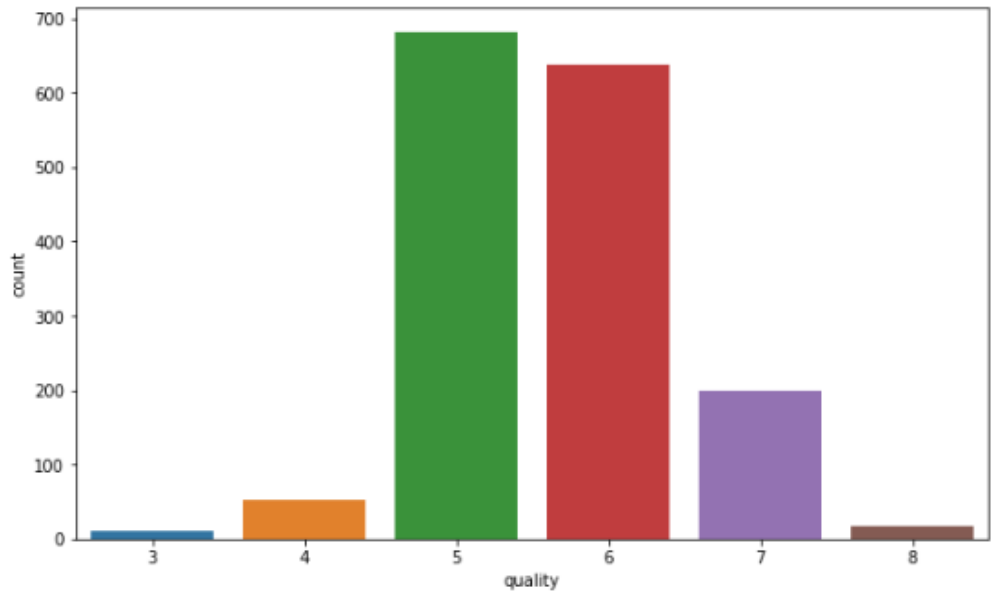


FIGURE 15- 4.1.8.1 Counting the Quality

There 6 wine quality values based from the results above. Minimum is 3 and maximum is 8. We can create 3 wine quality categories namely **poor quality**, **normal quality**, **excellent quality**.
if quality < 5 - **low quality**
if quality = 6 OR 7- **medium quality**
if quality > 7 - **high quality**

In the above plot we can see 5, 6 have high count compared to other.
We have 18 in 8 quality, we have 199 in 7 quality.
We have 638 in 6 quality, we have 681 in 5 quality.
We have 53 in 4 quality, we have 10 in 3 quality.

4.1.9 HEAT MAP

```
fig = plt.subplots(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True,annot=True)
plt.show()
```

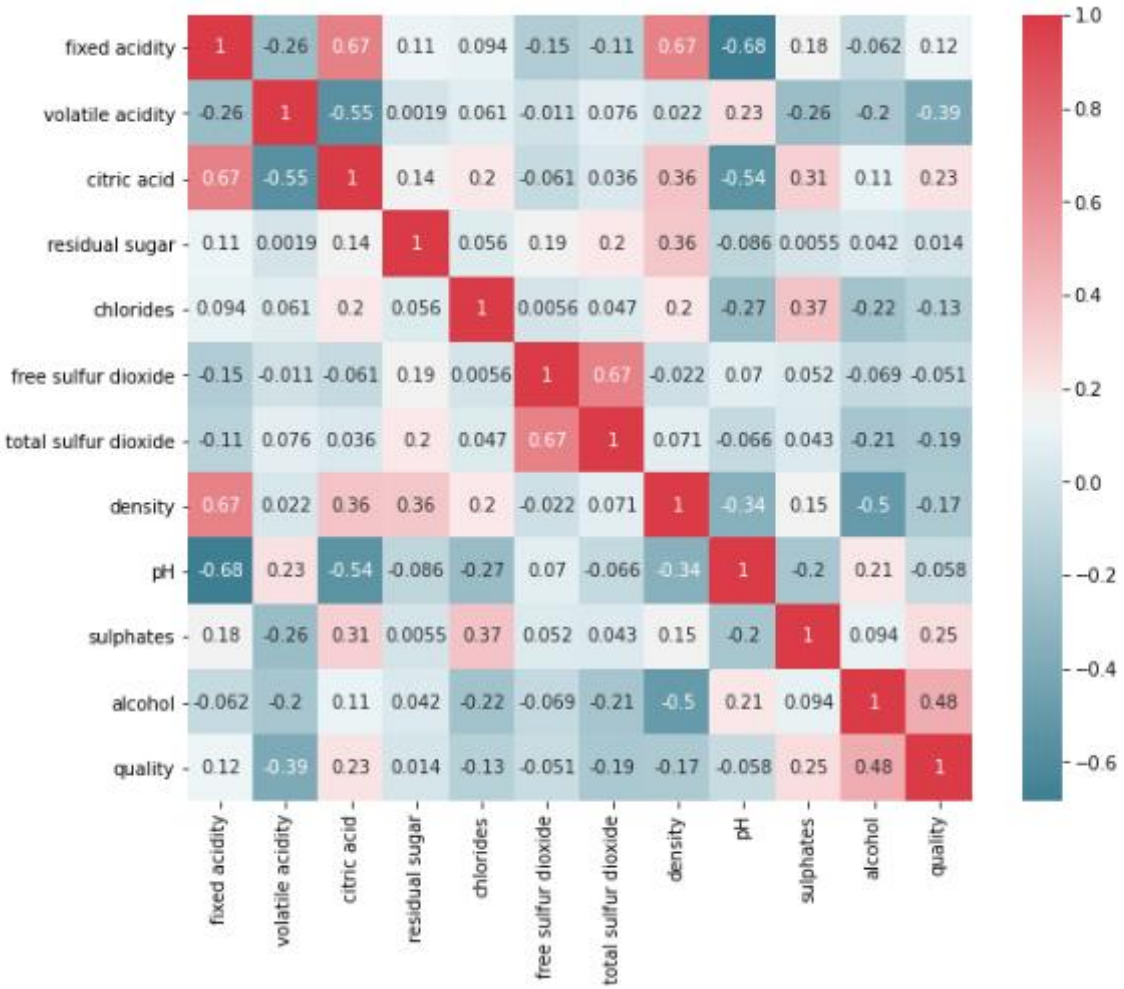


FIGURE 16- 4.1.9.1 HEAT MAP

When I checked the correlation between columns I can see that some of the features are strongly correlated with quality while some of them are not.

From the above correlation plot for the given dataset for wine quality prediction, we can easily see which items are related strongly with each other and which items are related weekly with each other. For Example,

The strongly correlated items are:
1. fixed acidity and citric acid.
2. Free sulfur dioxide and total sulfur dioxide.
3. fixed acidity and density.

Alcohol and quality.
So, from above points there is a clear inference that alcohol is the most important characteristic to determine the quality of wine.

The weekly correlated items are:
1. Citric acid and volatile acidity.
2. fixed acidity and ph.
3. Density and alcohol.
These are some relations which do not depend on each other at all.

4.2 ADDING A QUALITY LABEL COLUMN:-

```
# # Set a new field quality_label by transforming the existing quality column
# df['quality_label'] = df['quality'].apply(lambda value: 'low' if value <= 5
#                                                          else 'medium' if value <= 7
#                                                          else 'high')
# # Convert new field to categorical type
# df['quality_label'] = pd.Categorical(df['quality_label'],
#                                      categories=['low', 'medium', 'high'])
bins = [1, 5, 6, 8]
df['quality_label'] = pd.cut(df.quality, bins, labels=['low', 'medium', 'high'], include_lowest=True)
```

FIGURE 17-4.2.1 ADDING A QUALITY LABEL COLUMN

We are dividing the quality into 3 qualities:-low, medium, high.

- Low is <5.
- Medium is 6 – 7.
- High >7.

```
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	quality_label
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	low
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	low
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	low
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	medium
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	low
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5	low
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6	medium
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6	medium
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5	low
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6	medium

1599 rows × 13 columns

Dataset after adding the quality label column.

```
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'quality_label'],
      dtype='object')
```

FIGURE 18- 4.2.3 COLUMNS

Columns of the dataset.

4.2.1 COUNTING THE VALUE IN THE QUALITY LABELS:-

Counting the no of low,medium,high in the dataset

```
[92] df['quality_label'].value_counts()
```

```
low       744
medium    638
high      217
Name: quality_label, dtype: int64
```

FIGURE 19- 4.2.1.1 COUNTING THE VALUE IN THE QUALITY LABELS

We have 744 – low, 638- medium, 217—high.

```
df.groupby('quality_label').mean()
```

quality_label	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
low	8.142204	0.589503	0.237755	2.542070	0.092989	16.567204	54.645161	0.997068	3.311653	0.618535	9.926478	4.901882
medium	8.347179	0.497484	0.273824	2.477194	0.084956	15.711599	40.869906	0.996615	3.318072	0.675329	10.629519	6.000000
high	8.847005	0.405530	0.376498	2.708756	0.075912	13.981567	34.889401	0.996030	3.288802	0.743456	11.518049	7.082949

4.2.2 PLOTTING OF THE PIE CHART AND BARPLOT FOR QUALITY LABELS:-

```
plt.figure(figsize=(8,8))
label = ["low",'medium','high']
values=df['quality_label'].value_counts().values
plt.pie(values,labels=label,colors=['cyan','orange','red'],autopct='%1.2f%%')
plt.show()
```

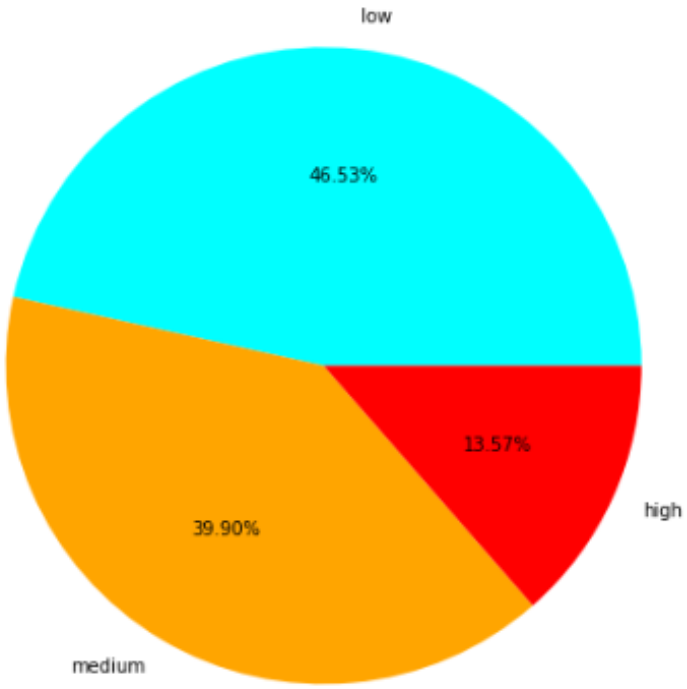


FIGURE 20- 4.2.2.1 PLOTTING OF THE PIE CHART FOR QUALITY LABELS

In the above pie chart we can observe, we have

- Low= 46.53%
- Medium=39.90%
- High=13.57%

```
plt.figure(figsize=(6, 6))
sns.countplot(df['quality_label'], palette="muted")
plt.show()
```

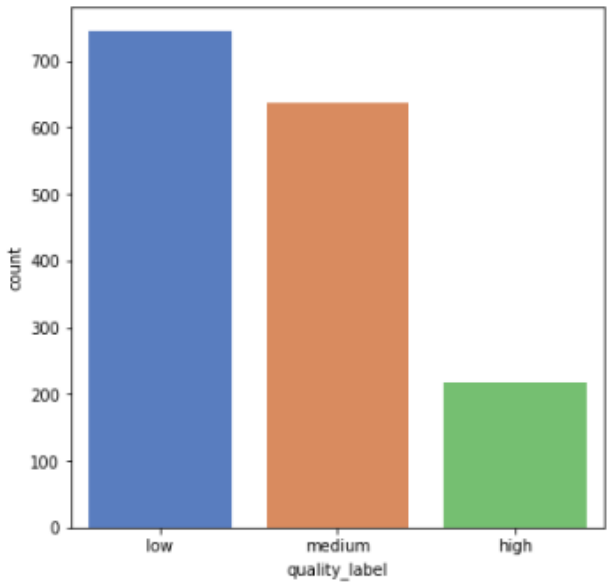


FIGURE 21- 4.2.2.2 BARPLOT FOR QUALITY LABELS

We can observe that in the given dataset we have high number (more than 700) of low quality data. We have low number (we have 200) of high quality data.

4.2.3 GROUPING OF QUALITY LABELS IN DIFFERENT DATAFRAMES:-

```
#### Filtering df for only low quality
df_temp = df[df['quality_label']=='low']
print(df_temp)
```

```
      fixed acidity  volatile acidity  ...  quality  quality_label
0               7.4             0.700  ...        5            low
1               7.8             0.880  ...        5            low
2               7.8             0.760  ...        5            low
4               7.4             0.700  ...        5            low
5               7.4             0.660  ...        5            low
...             ...               ...  ...      ...            ...
1582            6.1             0.715  ...        5            low
1583            6.2             0.460  ...        5            low
1589            6.6             0.725  ...        5            low
1594            6.2             0.600  ...        5            low
1597            5.9             0.645  ...        5            low

[744 rows x 13 columns]
```

FIGURE 22- 4.2.3.1 GROUPING OF LOW QUALITY LABELS IN DATAFRAMES

In df_temp we have stored low quality.

```
#### Filtering df for only medium quality
df_temp2 = df[df['quality_label']=='medium']
print(df_temp2)
```

```
      fixed acidity  volatile acidity  ...  quality  quality_label
3              11.2             0.280  ...        6         medium
19              7.9             0.320  ...        6         medium
20              8.9             0.220  ...        6         medium
24              6.9             0.400  ...        6         medium
29              7.8             0.645  ...        6         medium
...             ...               ...  ...      ...            ...
1592            6.3             0.510  ...        6         medium
1593            6.8             0.620  ...        6         medium
1595            5.9             0.550  ...        6         medium
1596            6.3             0.510  ...        6         medium
1598            6.0             0.310  ...        6         medium

[638 rows x 13 columns]
```

FIGURE 23- 4.2.3.2 GROUPING OF MEDIUM QUALITY LABELS IN DATAFRAMES

In df_temp2 we have stored medium quality

```
#### Filtering df for only high quality
df_temp3 = df[df['quality_label']== 'high']
print(df_temp3)
```

```
      fixed acidity  volatile acidity  ...  quality  quality_label
7               7.3              0.65  ...        7           high
8               7.8              0.58  ...        7           high
16              8.5              0.28  ...        7           high
37              8.1              0.38  ...        7           high
62              7.5              0.52  ...        7           high
...             ...               ...  ...      ...            ...
1541            7.4              0.25  ...        7           high
1544            8.4              0.37  ...        7           high
1549            7.4              0.36  ...        8           high
1555            7.0              0.56  ...        7           high
1584            6.7              0.32  ...        7           high

[217 rows x 13 columns]
```

FIGURE 24- 4.2.3.3 GROUPING OF HIGH QUALITY LABELS IN DATAFRAMES

In df_temp3 we have stored high quality

4.3 VISUALIZING OF THE DATASET:-

4.3.1 ANALYSIS OF FEATURE WITH QUALITY_LABEL:-

```
#visualizing  the variation of Alcohol in the different qualitys of wine
df.groupby('quality_label').alcohol.plot.hist(title='Alcohol by quality group',
                                              alpha=0.5, legend=True)
plt.show()
```

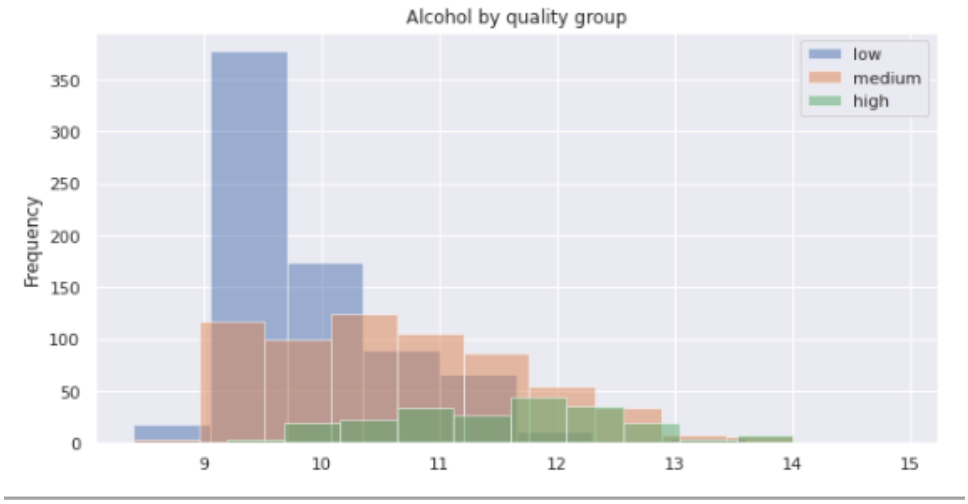


FIGURE 25- 4.3.1.1 ANALYSIS OF FEATURE WITH QUALITY_LABEL

In high quality wine Alcohol rate will be more. Confirms trend of low quality having low amounts of alcohol.

Analysis of pH & wine ratings:-

```
#visualizing  the variation of pH in the different qualitys of wine
sns.set(rc={'figure.figsize':(10,5)})
sns.boxplot(y='pH', x='quality_label', hue='quality_label', data=df)
plt.show()
```

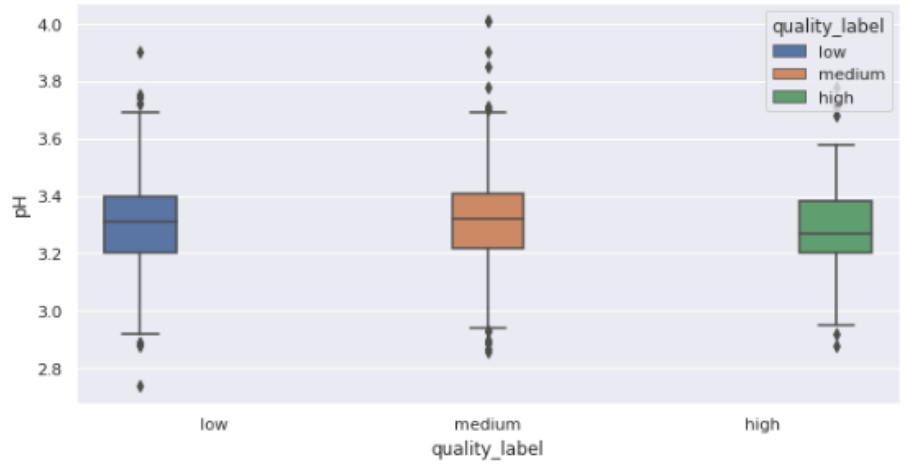


FIGURE 26- 4.3.1.2 ANALYSIS OF PH & WINE RATINGS

Analysis of fixed acidity & wine ratings:-

```
#visualizing  the variation of fixed acidity in the different qualitys of wine
sns.set(rc={'figure.figsize':(10,5)})
sns.boxplot(y='fixed acidity', x='quality_label', hue='quality_label', data=df)
plt.show()
```

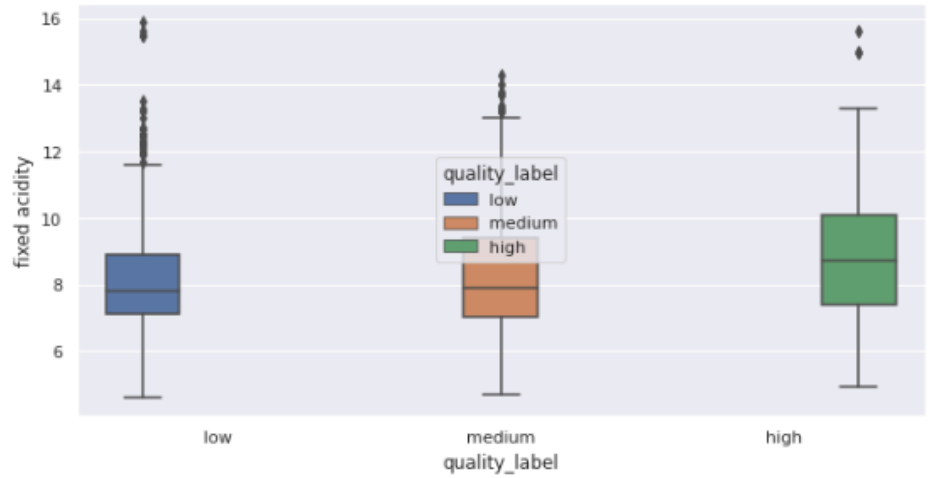


FIGURE 27- 4.3.1.3 ANALYSIS OF FIXED ACIDITY & WINE RATINDS

Analysis of sulphates & wine ratings:-

```
#visualizing  the variation of sulphates in the different qualitys of wine
sns.set(rc={'figure.figsize':(10,5)})
sns.boxplot(y='sulphates', x='quality_label', hue='quality_label', data=df)
plt.show()
```

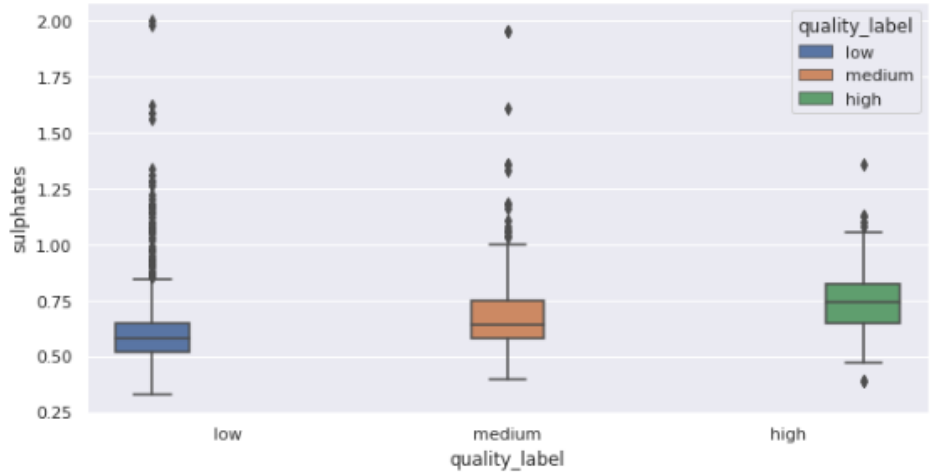


FIGURE 28- 4.3.1.4 ANALYSIS OF SULPHATES & WINE RATINGS

From the above plots

- fixed acidity has no significant effect on quality
- increase in alcohol also increases the wine quality
- increase in sulphates also increases the wine quality
- increase in pH also the wine quality

4.3.2 PLOTS OF THE DATASET

SCATTERPLOT OF ALCOHOL AND PH

```
#Scatter plot of Alcohol and the PH
plt.figure(figsize=(12,10))
sns.scatterplot('alcohol','pH',data=df,color='red')
plt.show()
```

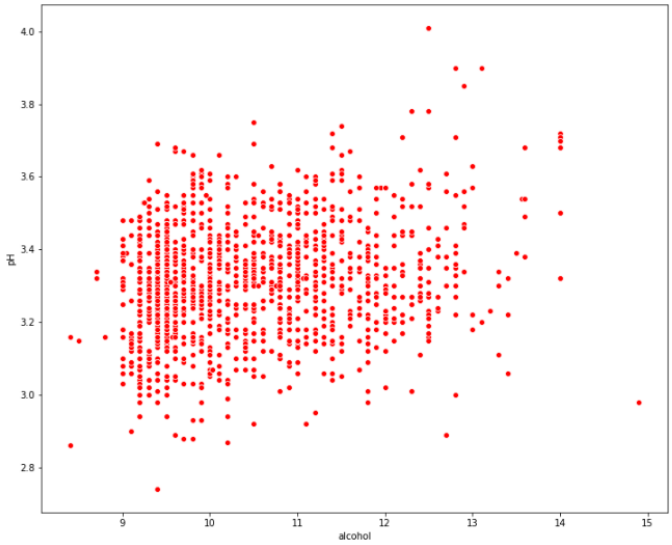


FIGURE 29- 4.3.2.1 SCATTERPLOT OF ALCOHOL AND PH

RELATION OF ALCOHOL WITH WINE

```
# checking the variation of fixed acidity in the different qualities of wine

sns.barplot(df['quality_label'], df['alcohol'], palette='Reds')
plt.title('relation of alcohol with wine')
plt.xlabel('quality')
plt.ylabel('alcohol')
plt.show()
```

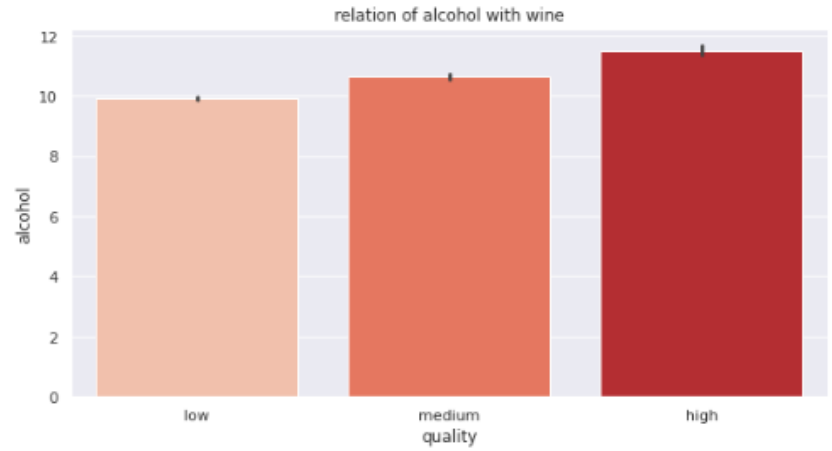


FIGURE 30- 4.3.2.2 RELATION OF ALCOHOL WITH WINE

It seems there's positive relationship between alcohol and quality. High quality wines are more likely to have high percentage of alcohol.

RELATION BETWEEN DENSITY AND ALCOHOL:-

```
#density  in alcohol
plt.figure(figsize=(12,6))
sns.jointplot(y=df["density"],x=df["alcohol"],kind="hex")
plt.show()
```

```
<Figure size 864x432 with 0 Axes>
```

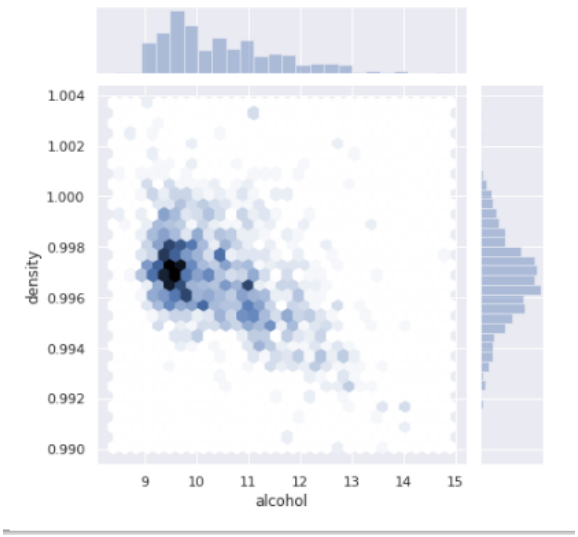


FIGURE 31- 4.3.2.3 RELATION BETWEEN DENSITY AND ALCOHOL

RELATION BETWEEN QUALITY AND RESIDUAL SUGAR:-

```
#quality with residual sugar
#It does not effect the wine quality
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality_label', y = 'residual sugar', data = df)
plt.show()
```

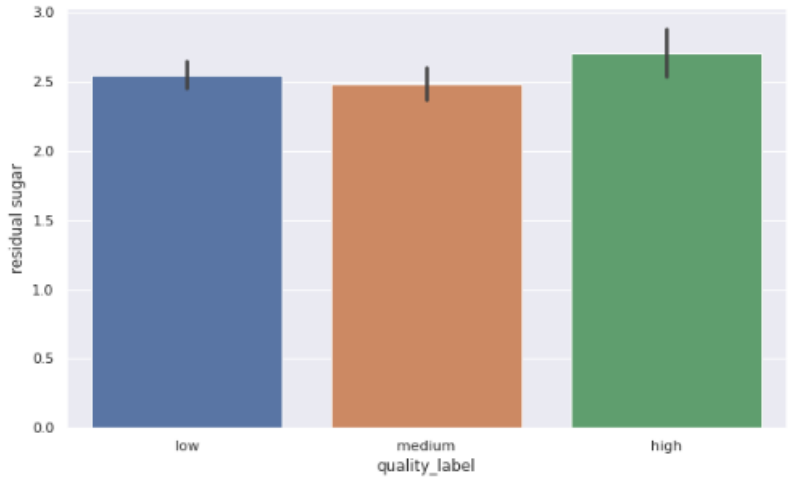


FIGURE 32- 4.3.2.4 RELATION BETWEEN QUALITY AND RESIDUAL SUGAR

As we can clearly see, residual sugar is not very impact full of the quality of wine. Hence we can eliminate these features. Though we are selecting these features, they will change according to the domain experts

RELATION BETWEEN QUALITY AND CITRIC ACID:-

```
#quality with citric acid
# Composition of citric acid go higher as we go higher in the quality of the wine
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality_label', y = 'citric acid', data = df)
plt.show()
```

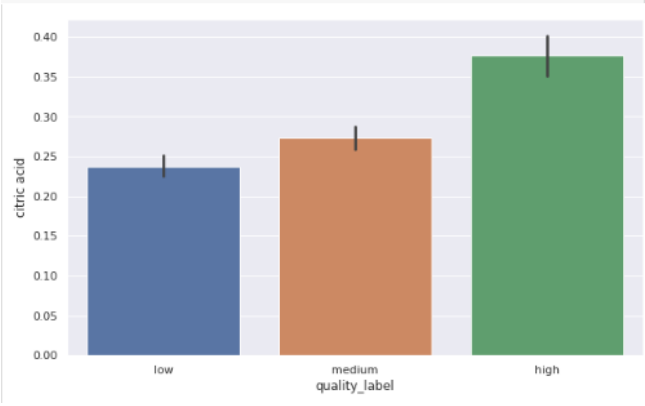
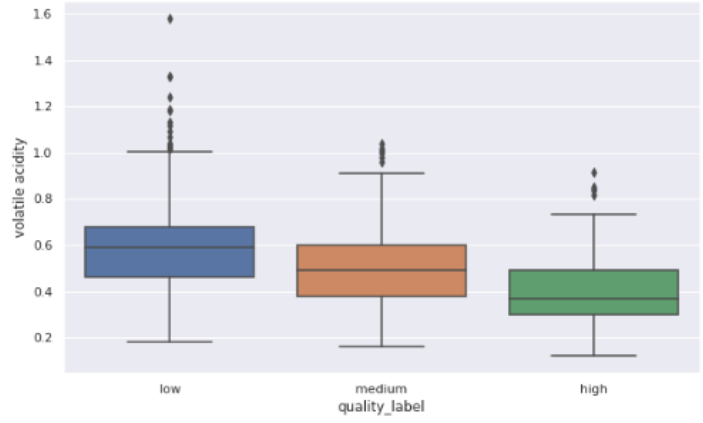


FIGURE 33- 4.3.2.5 RELATION BETWEEN QUALITY AND CITRIC ACID

This bar plot shows a directly proportional relation between citric acid and quality. As the quality of wine increases the amount of citric acid also increases which shows that citric acid is the important feature on which quality of wine depends.

RELATION BETWEEN QUALITY AND VOLATILE ACIDITY:-

```
#quality with volatile acidity
#increasing the volatile it decrease the wine quality
fig = plt.figure(figsize = (10,6))
sns.boxplot(x = 'quality_label', y = 'volatile acidity', data = df)
plt.show()
```



.

FIGURE 34- 4.3.2.6 RELATION BETWEEN QUALITY AND VOLATILE ACIDITY

Now we can see a negative association between these two attributes. While alcohol is increasing with quality, volatile acidity is negatively associated with quality. The correlation coefficient R2=0.152R2=0.152, which means volatile acidity can explain only 15.2% the variation of quality.

RELATION BETWEEN QUALITY AND CITRIC ACID

```
#quality with citric acid
# Composition of citric acid go higher as we go higher in the quality of the wine
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality_label', y = 'citric acid', data = df)
plt.show()
```

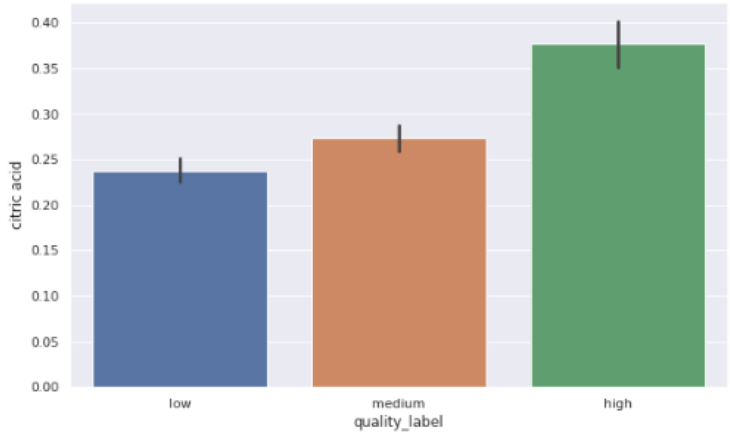


FIGURE 35- 4.3.2.7 RELATION BETWEEN QUALITY AND CITRIC ACID

Free sulfur dioxide is greatly contributing to the quality of wine; this bar plot gives us a clearer picture.

Histogram of the red wine data

```
# Histogram
df.hist(figsize=(10,10))
plt.show()
```

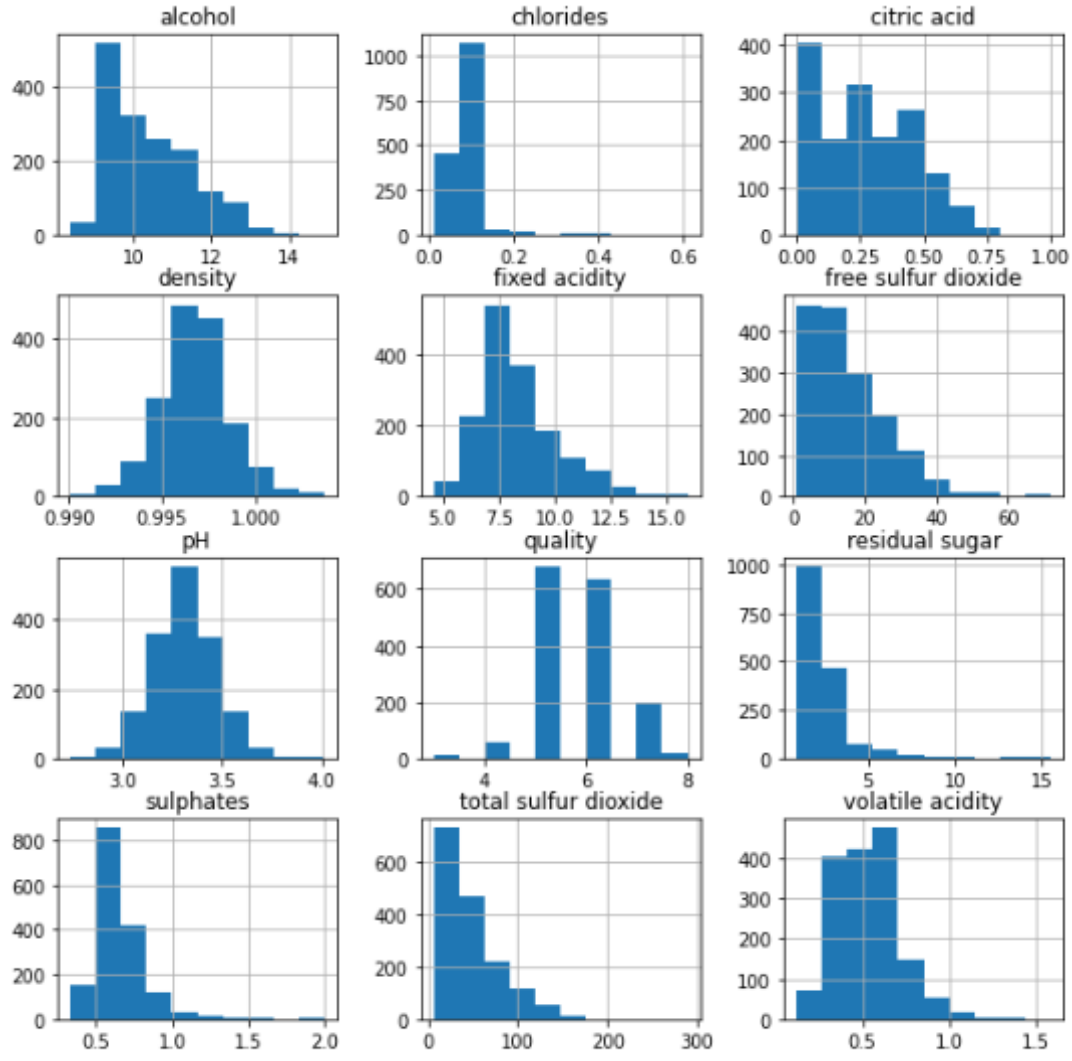


FIGURE 36- 4.3.2.8 HISTOGRAM

4.4 Splitting the data into train and validation:

In this step of the analysis I defined the features to train and test the machine learning model and the target to predict which ‘quality’ is. And then I did standardization (also called z-score normalization) for the features because different scales of features may impact the performance of the machine learning models. For this purpose, I used StandardScaler () function defined in Scikit-learn. And finally I split the dataset into training and test sets 80% and 20% respectively.

```
# dividing the dataset into dependent and independent variables
X = df.drop(['quality','quality_label'], axis = 1)
y = df['quality_label']
# determining the shape of x and y.
print(X.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

FIGURE 37- 4.4.1 DIVIDING INTO A AND Y VARIABLES

Now we need to import train_test_split from the sklearn.model_selection library where sklearn is the main package name and the model_selection is the sub package.

```
#dividing the dataset in training and testing set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 44)
```

FIGURE 38- 4.4.2 TRAIN AND TESTING SET

Here we used the X_train, X_test, y_train, y_test variables to store the testing and training inputs. The train_test_split splits the data according to the test_size attribute. Here I used 20% for the testing data and 80% for the training data. The random_state is used to shuffle the records.

```
# determining the shapes of training and testing sets
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(1199, 11)
(1199,)
(400, 11)
(400,)
```

FIGURE 39- 4.4.3 SHAPE OF TRAIN AND TEST

Now we used shape to view the number of columns and rows in the training and testing data.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Scaling for training data
scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
scaled_X_train

# Scaling for test data
scaled_X_test = pd.DataFrame(scaler.transform(X_test),columns=X_test.columns)
scaled_X_test
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	-0.644910	0.051016	-0.683192	-0.310468	-0.504370	-0.062753	-0.508882	-1.856471	0.266530	-0.290408	1.568472
1	0.090923	-1.919411	1.222880	-0.542153	-0.352382	2.829266	1.342243	-0.797302	0.133736	0.618096	1.287543
2	-0.418500	1.205122	-1.301378	-0.045686	-0.352382	-0.544756	-0.979507	-0.723529	1.328886	0.315261	1.193900
3	0.487140	-0.511963	1.274395	-0.178077	-0.091830	-0.159154	-0.697132	0.446299	0.266530	-0.290408	0.819329
4	0.373935	0.051016	1.119849	-0.442860	1.818878	-0.930359	-0.634382	0.393604	-0.264648	-0.290408	-0.959885
...	...	...	...	...	...	...	...	...	...	...	...
395	-0.984525	1.008080	-0.992285	0.020510	0.407559	-0.159154	-0.571632	-0.054303	0.864106	-0.472109	-0.210742
396	-0.305295	0.332505	-1.301378	-0.376664	-0.308956	-0.641157	-0.854007	0.024739	0.332928	-0.532676	-0.866242
397	-0.248692	-1.243836	2.098643	3.793664	-0.808346	2.106262	7.303492	-1.893357	-1.990976	-0.896078	1.755757
398	-0.361897	0.332505	-1.352893	-0.509055	0.016733	-0.351955	-0.854007	-0.565445	0.067339	-0.593243	0.070187
399	-0.192090	0.445101	-0.992285	0.020510	-0.678071	-1.219561	-1.199132	-2.051442	-0.596634	-1.744015	2.411257

400 rows × 11 columns

FIGURE 40- 4.4.4 SCALING INPUT USING STANDARD SCALER

The training and the testing data inputs has been scaled by using the standard scaler of the sklearn package.

4.5 Model Building and Evaluation

Now we use various algorithms to build the models and choose the best out those algorithms

4.5.1 DECISION TREE

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.

Decision trees are a popular model, used in operations research, strategic planning, and machine learning. Each square above is called a node, and the more nodes you have, the more accurate your decision tree will be (generally). The last nodes of the decision tree, where a decision is made, are called the leaves of the tree. Decision trees are intuitive and easy to build but fall short when it comes to accuracy.

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any Boolean function on discrete attributes using the decision tree.

Approach to make decision tree:-

While making decision tree, at each node of tree we ask different type of questions. Based on the asked question we will calculate the information gain corresponding to it.

Information Gain:-

- Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes. A commonly used measure of purity is called information. For each node of the tree, the information value measures how much information a feature gives us about the class. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.

Gini Impurity

- First let’s understand the meaning of Pure and Impure.
- Pure means, in a selected sample of dataset all data belongs to same class (PURE).
- Impure means, data is mixture of different classes.

Definition of Gini Impurity: - Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.

Advantage of Decision Tree

- Easy to use and understand.
- Can handle both categorical and numerical data.
- Resistant to outliers, hence require little data preprocessing.

Disadvantage of Decision Tree

- Prone to over fitting.
- Require some kind of measurement as to how well they are doing.
- Need to be careful with parameter tuning.
- Can create biased learned trees if some classes dominate.

```
# Apply the decision tree algorithm
from sklearn.tree import DecisionTreeClassifier
# create an object
rtree = DecisionTreeClassifier(criterion='entropy')
# Applying the classifier to the dataset
rtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

FIGURE 41- 4.5.1.1 BUILDING THE MODEL

Predicting the training data

```
#predict on training data
y_train_pred = rtree.predict(X_train)
y_train_pred
```

```
array(['medium', 'low', 'medium', ..., 'medium', 'low', 'low'],
      dtype=object)
```

FIGURE 42- 4.5.1.2 PREDICT TRAIN DATA

```
#Classification report on training data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred))
confusion_matrix(y_train,y_train_pred)
```

```
              precision    recall  f1-score   support

        high       1.00      1.00      1.00       168
         low       1.00      1.00      1.00       540
      medium       1.00      1.00      1.00       491

    accuracy                           1.00      1199
   macro avg       1.00      1.00      1.00      1199
weighted avg       1.00      1.00      1.00      1199

array([[168,   0,   0],
       [  0, 540,   0],
       [  0,   0, 491]])
```

FIGURE 43- 4.5.1.3 TRAIN CLASSIFICATION REPORT

Classification report of training data.

```
## predicting the testdata
y_test_pred = rtree.predict(X_test)
y_test_pred
```

```
array(['medium', 'medium', 'high', 'medium', 'medium', 'low', 'medium',
       'low', 'low', 'low', 'low', 'low', 'medium', 'low', 'medium',
       'high', 'low', 'high', 'low', 'low', 'low', 'low', 'low', 'low',
       'medium', 'medium', 'medium', 'low', 'medium', 'medium', 'low',
       'high', 'medium', 'low', 'low', 'high', 'medium', 'medium', 'low',
       'low', 'medium', 'low', 'low', 'low', 'low', 'low', 'low', 'low',
       'medium', 'medium', 'medium', 'medium', 'medium', 'low', 'low',
       'high', 'low', 'medium', 'low', 'low', 'low', 'low', 'low', 'high'
       'low', 'low', 'low', 'low', 'low', 'low', 'medium', 'medium',
       'low', 'high', 'medium', 'low', 'low', 'low', 'low', 'low', 'low',
       'medium', 'medium', 'low', 'low', 'medium', 'medium', 'low', 'low'
       'medium', 'high', 'medium', 'low', 'high', 'low', 'low', 'medium',
       'low', 'high', 'medium', 'low', 'low', 'high', 'medium', 'medium',
       'low', 'medium', 'low', 'medium', 'low', 'medium', 'high',
```

FIGURE 44- 4.5.1.4 PREDICT TEST DATA

Predicting the test data.

```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

        high       0.54      0.55      0.55        49
         low       0.80      0.76      0.78       204
      medium       0.64      0.67      0.66       147

    accuracy                           0.70       400
   macro avg       0.66      0.66      0.66       400
weighted avg       0.71      0.70      0.71       400
```

FIGURE 45- 4.5.1.5 TEST CLASSIFICATION REPORT

Now we can also use cross validation score to maybe further increase the accuracy Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
We can apply this by importing the cross_val_score of the scikit learn package as shown

```
# Visualisation of the decision tree
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
```

```
from sklearn.model_selection import cross_val_score
cross_val_score(rtree,X_train,y_train,cv=5)
```

```
array([0.65833333, 0.625     , 0.63333333, 0.65416667, 0.65690377])
```

FIGURE 46- 4.5.1.6 CROSS VAL SCORE

Hyperparameters

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed.

In any machine learning algorithm, these parameters need to be initialized before training a model. Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model is being trained. A good choice of hyperparameters can really make an algorithm shine.

The downside of the hyper parameters is that it is trial and error, it is very difficult to find out which hyperparameters will give the optimal value so, it is trial and error but we can use grid search cv to make it easy for us. It is an Exhaustive search over specified parameter values for an estimator. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

So for us to use this hyper parameter tuning we will first define a range of values for the parameter of the random forest classifier and the grid search will find the best parameter out of those.

```
## Hyperparameters
grid_param = {'criterion':['gini','entropy'],
              'max_depth': range(2,10,1),
              'min_samples_leaf' : range(1,10,1)}
```

FIGURE 47- 4.5.1.7 HYPER PARAMETRERS

First we need to create a dictionary with all the parameters that we want to change
Now we need to import the GridSearchCV from the sklearn package and pass the model name and the dictionary into the object and then fit the model with the training data as shown

```
] from sklearn.model_selection import GridSearchCV
  grid_search = GridSearchCV(estimator=rtree, param_grid=grid_param)
  grid_search.fit(X_train,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='entropy',
                                              max_depth=None, max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'min_samples_leaf': range(1, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

FIGURE 48- 4.5.1.8 GRID SEARCH CV

Now that we have built the model let's check what the optimal parameters were. It can be done by using the best_params_

```
# returns the optimal parameters.
grid_search.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 7, 'min_samples_leaf': 2}
```

FIGURE 49- 4.5.1.9 BEST PARAMETERS

Here as we can see that the best parameter that we can use in building our random forest model are Criterion as entropy, max_depth as 7 and min_samples_leaf as 2

```
# Apply the decision tree algorithm
from sklearn.tree import DecisionTreeClassifier
# create an object
rtree = DecisionTreeClassifier(criterion='entropy',max_depth=7,min_samples_leaf=2)
# Applying the classifier to the dataset
rtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=7, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
y_train_pred = rtree.predict(X_train)
y_train_pred
```

```
array(['low', 'low', 'medium', ..., 'low', 'low', 'high'], dtype=object)
```

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print(classification_report(y_train,y_train_pred))
dec_train = accuracy_score(y_train,y_train_pred)
```

```
              precision    recall  f1-score   support

        high       0.79      0.76      0.77       168
         low       0.77      0.89      0.82       540
      medium       0.80      0.68      0.73       491

    accuracy                           0.78      1199
   macro avg       0.79      0.77      0.78      1199
weighted avg       0.78      0.78      0.78      1199
```

FIGURE 50- 4.5.1.10 TRAINING ACCURACY

From above we got Training data accuracy as: - 0.78 or 78%

```
clf = DecisionTreeClassifier(criterion='entropy',max_depth=7,min_samples_leaf=2)
clf.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=7, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
pred_test = clf.predict(X_test)
print(classification_report(y_test,pred_test))
dec_test = accuracy_score(y_test,pred_test)
```

```
              precision    recall  f1-score   support

        high       0.55      0.55      0.55        49
         low       0.70      0.83      0.76       204
      medium       0.61      0.45      0.52       147

    accuracy                           0.66       400
   macro avg       0.62      0.61      0.61       400
weighted avg       0.65      0.66      0.64       400
```

FIGURE 51- 4.5.1.11 TESTING ACCURACY

From above we got Testing data accuracy as:- 0.66 or 66%

Form the above algorithm we got train and test f1-score.
Train=0.78
Test=0.66

CONFUSION MATRIX:-

```
from sklearn import metrics
dtc_conf_matrix = pd.DataFrame(data=metrics.confusion_matrix(y_test,y_test_pred)
,index=[['low','medium','high']], columns=[['low','medium','high']])
print("Confusion Matrix: ")
dtc_conf_matrix
```

```
Confusion Matrix:
```

	low	medium	high
low	27	8	14
medium	7	156	41
high	16	32	99

```
#heatmap of confusion matrix:
sns.heatmap(dtc_conf_matrix,annot=True)
plt.title('CONFUSION MARTIX')
plt.show()
```

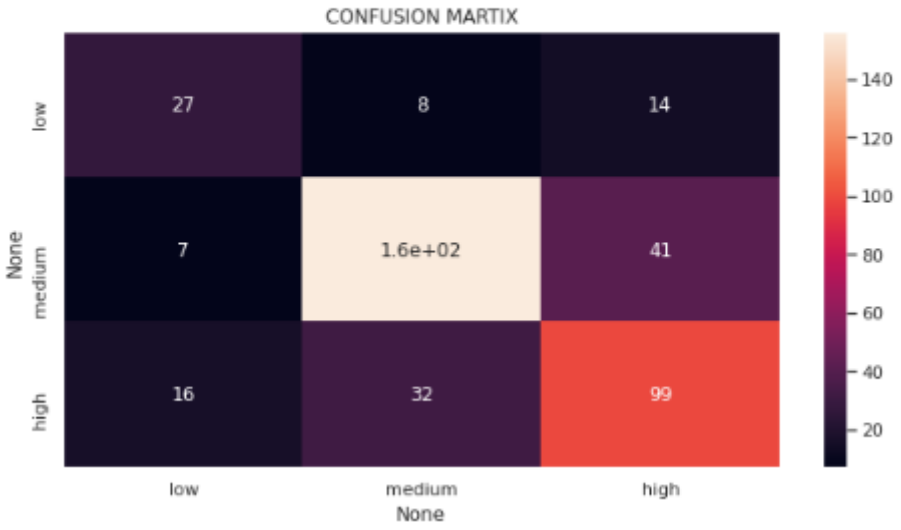


FIGURE 52- 4.5.1.12 CONFUSION MATRIX

4.5.2 LOGISTIC REGRESSION

Classification in Machine Learning is a technique of learning, where an instance is mapped to one of many labels. The machine learns patterns from data in such a way that the learned representation successfully maps the original dimension to the suggested label/class without any intervention from a human expert.
Logistic regression predicts categorical outcomes (binomial / multinomial values of y)
It predicts the probability associated with each dependent variable category.
It uses the sigmoid function
It finds a linear relationship between the independent variables and a link function of its probabilities. Then the link ruction that provides the best goodness of fit for the given is chosen.

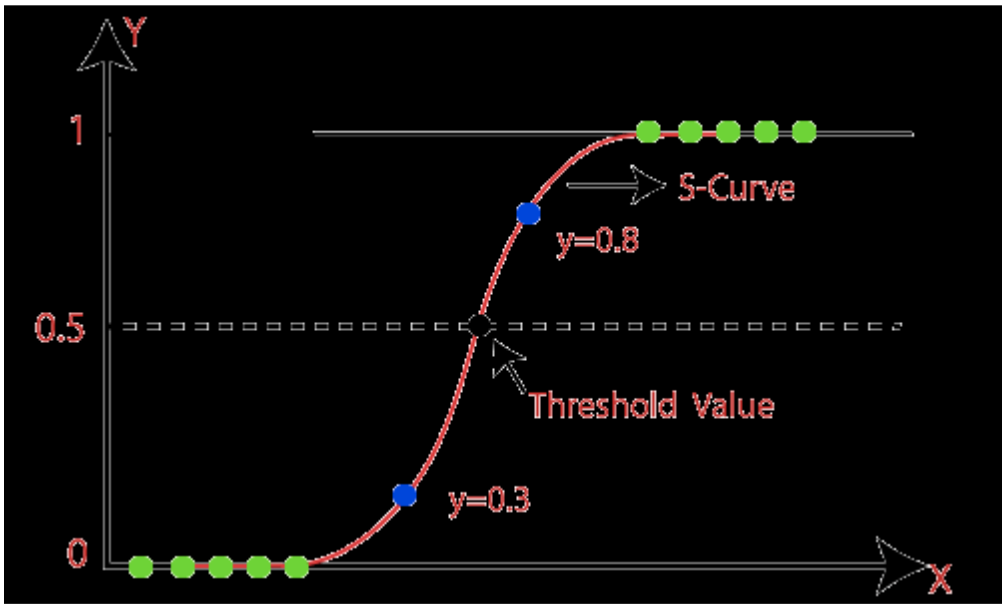


FIGURE 53- 4.5.2.1 LOGISTIC REGRESSION

Here it creates one threshold and the values above it are classified as 1 and below it are 0. In case of multiple classes, then it creates more threshold values. So in order for us to use this algorithm, first we need to import the package and then create an object or instance for that class and then fit the training data and build the model.

```
#importing the model class
from sklearn.linear_model import LogisticRegression
# creating an object for Logistic Regression
log_reg = LogisticRegression()
#fitting the input and output of training data to the object and building the model
log_reg.fit(scaled_X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

FIGURE 54- 4.5.2.2 BUILDING THE MODEL

Now that we have built the model, next step is to check the accuracy of the training data and in order to do that we need to predict the training data output and compare them to the original values. We can do that by using the predict function

```
# predicting the output of the training data
y_train_pred = log_reg.predict(scaled_X_train)
y_train_pred
```

```
array(['low', 'low', 'medium', ..., 'low', 'low', 'medium'], dtype=object)
```

```
from sklearn.metrics import accuracy_score
print("Training data accuracy:",accuracy_score(y_train,y_train_pred))
log_train = accuracy_score(y_train,y_train_pred)
```

```
Training data accuracy: 0.6430358632193495
```

FIGURE 55- 4.5.2.3 PREDICTING THE OUTPUT AND TRAINING ACCURACY

```
y_test_pred = log_reg.predict(scaled_X_test)
y_test_pred
```

```
array(['medium', 'medium', 'medium', 'medium', 'low', 'low', 'medium',
       'low', 'low', 'low', 'medium', 'low', 'medium', 'medium', 'low',
       'low', 'low', 'medium', 'low', 'low', 'low', 'low', 'medium',
       'low', 'high', 'medium', 'low', 'medium', 'low', 'low', 'medium',
       'high', 'low', 'medium', 'low', 'medium', 'medium', 'medium',
       'low', 'medium', 'medium', 'low', 'medium', 'medium', 'low', 'low',
       'low', 'low', 'medium', 'medium', 'medium', 'medium', 'medium',
       'low', 'low', 'high', 'low', 'low', 'medium', 'medium', 'low',
       'low', 'low', 'high', 'low', 'low', 'low', 'low', 'low', 'medium',
       'low', 'low', 'low', 'high', 'low', 'low', 'low', 'low', 'low',
```

FIGURE 56- 4.5.2.4 PREDICTING THE TEST OUTPUT

```
print("Testing data accuracy:",accuracy_score(y_test,y_test_pred))
log_test = accuracy_score(y_test,y_test_pred)
```

```
Testing data accuracy: 0.6225
```

FIGURE 57- 4.5.2.5 TESTING ACCURACY

As we can see that we have got an accuracy score of 0.64 or 64% on the training data. The accuracy that we got for the testing data is 0.62 or 62%.

CONFUSION MATRIX:-

```
from sklearn import metrics
log_conf_matrix = pd.DataFrame(data=metrics.confusion_matrix(y_test,y_test_pred),
                               index=[['low','medium','high']], columns=[['low','medium','high']])
print("Confusion Matrix:")
log_conf_matrix
```

```
Confusion Matrix:
```

	low	medium	high
low	14	6	29
medium	2	153	49
high	14	51	82

```
#heatmap of confusion matrix:
sns.heatmap(log_conf_matrix,annot=True)
plt.title('CONFUSION MARTIX')
plt.show()
```

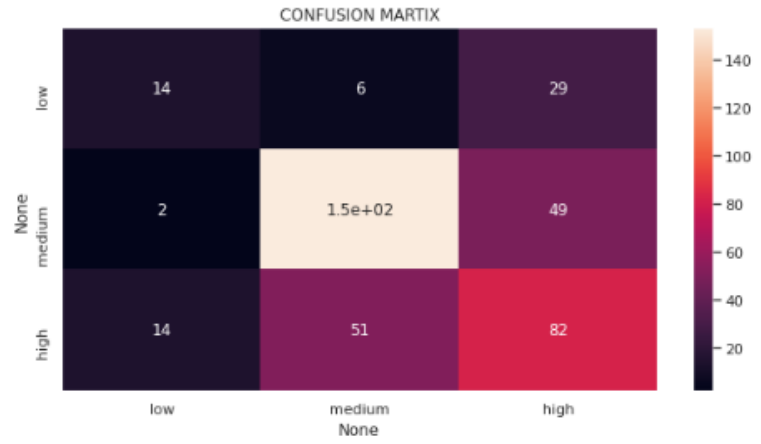


FIGURE 58- 4.5.2.6 LOGISTIC REGRESSION CONFUSION MATRIX

4.5.3 RANDOM FOREST

Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

Suppose training set is given as : [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

• [X1, X2, X3]

• [X1, X2, X4]

• [X2, X3, X4]

So finally, it predicts based on the majority of votes from each of the decision trees made.

This works well because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results as shown in the figure.

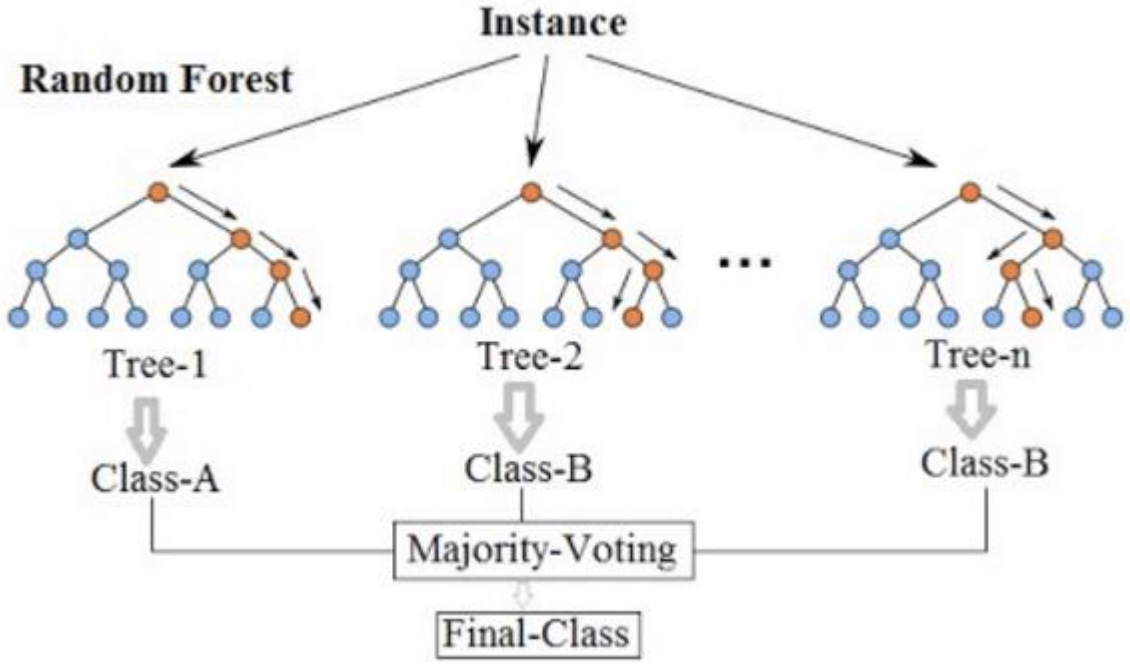


FIGURE 59- 4.5.3.1 RANDOM FOREST

So in order for us to use this algorithm, first we need to import the package and then create an object or instance for that class and then fit the training data and build the model

```
#import , initialize and fit
#import the RFC from sklearn
from sklearn.ensemble import RandomForestClassifier
#initialize the object for RFC
rfc = RandomForestClassifier(n_estimators=200)
#fit the RFC to the dataset
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

FIGURE 60- 4.5.3.2 BUILDING THE MODEL

Now that we have built the model, next step is to check the accuracy of the training data and in order to do that we need to predict the training data output and compare them to the original values. We can do that by using the predict function

```
#predictions on training data
#syntax: objectname.predict(inputvalues)
y_pred_train = rfc.predict(X_train)
from sklearn.metrics import confusion_matrix ,classification_report
print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

        high       1.00      1.00      1.00       168
         low       1.00      1.00      1.00       540
      medium       1.00      1.00      1.00       491

    accuracy                           1.00      1199
   macro avg       1.00      1.00      1.00      1199
weighted avg       1.00      1.00      1.00      1199
```

FIGURE 61- 4.5.3.3 TRAIN CLASSIFICATION REPORT

Here I predicted the training values and the classification report of the prediction is generated by using the classification_metrics of the sklearn.metrics sub package

```
#prediction on test data(unseen data)
y_pred_test = rfc.predict(X_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

        high       0.61      0.57      0.59        49
         low       0.82      0.82      0.82       204
      medium       0.68      0.69      0.68       147

    accuracy                           0.74       400
   macro avg       0.70      0.69      0.70       400
weighted avg       0.74      0.74      0.74       400
```

FIGURE 62- 4.5.3.4 TEST CLASSIFICATION REPORT

So, the accuracy of the testing data is calculated to be 1.0 or 100%

So, the accuracy of the testing data is calculated to be 0.74 or 74%

Now we can also use cross validation score to maybe further increase the accuracy Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
We can apply this by importing the cross_val_score of the scikit learn package as shown

```
] #CrossValidationScore
  from sklearn.model_selection import cross_val_score
  scores = cross_val_score(rfc,X_train,y_train,cv=5)
  np.mean(scores)
```

```
0.7056101813110182
```

FIGURE 63- 4.5.3.5 CROSS VAL SCORE

It has given us 5 different score as put CV=5, and we need to calculate the mean of the scores to get mean accuracy of all the cross validations. Here we passed training data as input and we get the score of the test data as output. So the accuracy of the test data using the cross validation score is 0.681 or 68.1% which is slightly better than the normal calculated test accuracy.

Hyperparameters

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed.

In any machine learning algorithm, these parameters need to be initialized before training a model. Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model is being trained. A good choice of hyperparameters can really make an algorithm shine.

The downside of the hyper parameters is that it is trial and error, it is very difficult to find out which hyperparameters will give the optimal value so, it is trial and error but we can use grid search cv to make it easy for us. It is an Exhaustive search over specified parameter values for an estimator. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

So for us to use this hyper parameter tuning we will first define a range of values for the parameter of the random forest classifier and the grid search will find the best parameter out of those.

```
# using hyperparameters
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,10,1),
    'min_samples_leaf' : range(1,10,1)
    }
```

FIGURE 64- 4.5.3.6 HYPER PARAMETERS

First we need to create a dictionary with all the parameters that we want to change

Now we need to import the GridSearchCV from the sklearn package and pass the model name and the dictionary into the object and then fit the model with the training data as shown

```
] #Import the GridSearchCV
  from sklearn.model_selection import GridSearchCV

  # initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
  clf = RandomForestClassifier()
  grid_search = GridSearchCV(estimator=clf, param_grid=grid_param)

  # applying gridsearch onto dataset
  grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'min_samples_leaf': range(1, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

FIGURE 65- 4.5.3.7 GRID SEARCH CV

Now that we have built the model lets check what the optimal parameters were. It can be done by using the best_params_

```
grid_search.best_params_
```

```
{'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 1}
```

FIGURE 66- 4.5.3.8 BEST PARAMETERS

Here as we can see that the best parameter that we can use in building our random forest model are Criterion as gini, max_depth as 9 and min_samples_leaf as 1

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(criterion= 'gini', max_depth= 9, min_samples_leaf= 1)
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

So now the next step is to build the model using these parameters and check if the performance has increased or not

```
y_pred_train = rfc.predict(X_train)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

        high       1.00      1.00      1.00       168
         low       1.00      1.00      1.00       540
      medium       1.00      1.00      1.00       491

    accuracy                           1.00      1199
   macro avg       1.00      1.00      1.00      1199
weighted avg       1.00      1.00      1.00      1199
```

FIGURE 67- 4.5.3.9 TRAINING ACCURACY

Here we can observe that the accuracy of the training data is 100% So now lets check for the test data.

```
y_pred_test = rfc.predict(X_test)
print(classification_report(y_test,y_pred_test))
ran_train = accuracy_score(y_train,y_pred_train)
```

```
              precision    recall  f1-score   support

        high       0.57      0.47      0.52        49
         low       0.80      0.81      0.81       204
      medium       0.63      0.67      0.65       147

    accuracy                           0.71       400
   macro avg       0.67      0.65      0.66       400
weighted avg       0.71      0.71      0.71       400
```

FIGURE 68- 4.5.3.10 TEST ACCURACY

Form the Random Forest algorithm we got train and test accuracy score
Train=1.0
Test=0.71

CONFUSION MATRIX:-

```
] rfc_conf_matrix = pd.DataFrame(data=metrics.confusion_matrix(y_test,y_pred_test),
                                 index=[['low','medium','high']], columns=[['low','medium','high']])
  print("Confusion Matrix: ")
  rfc_conf_matrix
```

```
Confusion Matrix:
```

	low	medium	high
low	29	6	14
medium	1	169	34
high	14	35	98

```
#heatmap of confusion matrix:
sns.heatmap(rfc_conf_matrix,annot=True)
plt.title('CONFUSION MARTIX')
plt.show()
```

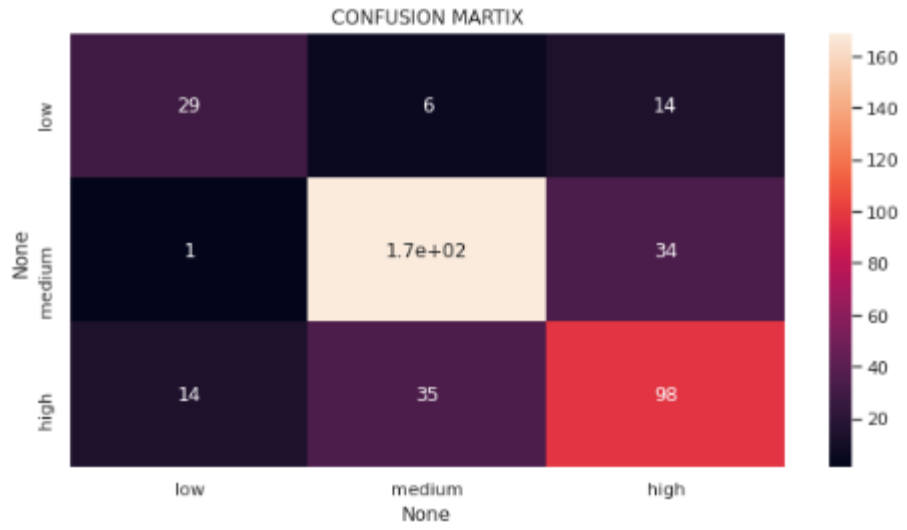


FIGURE 69- 4.5.3.11 RANDOM FOREST CONFUSION MATRIX

4.5.4 Gradient Boosting classifier

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

Gradient boosting involves three elements:
• A loss function to be optimized.

• A weak learner to make predictions.

• An additive model to add weak learners to minimize the loss function

Gradient boosting is a greedy algorithm and can overfit a training dataset quickly. It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.

We can use gradient boosting by importing the GradientBoostingClassifier from sklearn.ensemble Then we can build the model and evaluate its performance as shown

Here I also used the parameters such as learning_rate and n_estimators. The values for those parameters are calculated by using trail and error method.

```
from sklearn.ensemble import GradientBoostingClassifier
gdc = GradientBoostingClassifier()
gdc.fit(X_train,y_train)
y_pred = gdc.predict(X_test)
print("train accuracy: ",accuracy_score(y_train,gdc.predict(X_train)))
print("test accuracy: ",accuracy_score(y_pred,y_test))
grad_train = accuracy_score(y_train,gdc.predict(X_train))
grad_test = accuracy_score(y_pred,y_test)
```

```
train accuracy:  0.878231859883236
test accuracy:  0.6725
```

FIGURE 70- 4.5.4.1 GBC ACCURACY

Here we can observe the performance of the gradient boosting classifier model.
We got the accuracy of the train data as 0.87 or 87%
We got the accuracy of the test data as 0.67 or 67%

CONFUSION MATRIX:-

```
gdc_conf_matrix = pd.DataFrame(data=metrics.confusion_matrix(y_test,y_pred),
                               index=[['low','medium','high']], columns=[['low','medium','high']])
print("Confusion Matrix: ")
gdc_conf_matrix
```

```
Confusion Matrix:
```

	low	medium	high
low	25	9	15
medium	3	161	40
high	23	41	83

```
#heatmap of confusion matrix:
sns.heatmap(gdc_conf_matrix,annot=True)
plt.title('CONFUSION MARTIX')
plt.show()
```

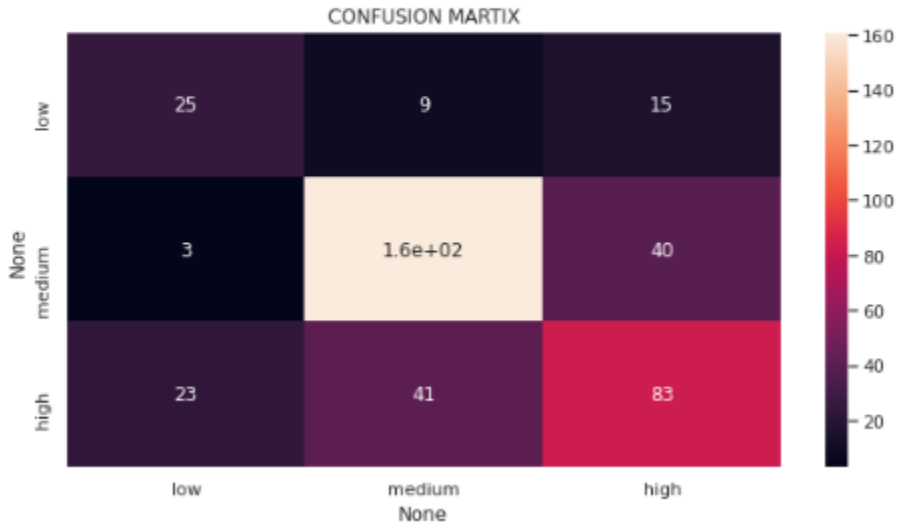


FIGURE 71- 4.5.4.2 GBC CONFUSION MATRIX

4.5.5 XGBoost Classifier

XGBoost (eXtreme Gradient Boosting) is an efficient and easy to use algorithm which delivers high performance and accuracy as compared to other algorithms. XGBoost is also known as regularized version of GBM.

The features of XGBoost are as follows

1. **Regularization**: XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting. That is why, XGBoost is also called the regularized form of GBM (Gradient Boosting Machine).

2. **Parallel Processing**: XGBoost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model. While using Scikit Learn library, nthread hyper-parameter is used for parallel processing. nthread represents number of CPU cores to be used. If you want to use all the available cores, don't mention any value for nthread and the algorithm will detect automatically.

3. **Handling Missing Values**: XGBoost has an in-built capability to handle missing values. When XGBoost encounters a missing value at a node, it tries both the left and right hand split and learns the way leading to higher loss for each node. It then does the same when working on the testing data.

4**. Cross Validation**: XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

5. **Effective Tree Pruning**: A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand make splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

In order for us to use the XGBoost classifier, first we need to import the XGBClassifier from the xgboost package.
Then build the model and find the performance of that model as shown below.

```
# using xgboost classifier
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train,y_train)
preds = xgb.predict(X_test)
print("train accuracy: ",accuracy_score(y_train,xgb.predict(X_train)))
print("test accuracy: ",accuracy_score(preds,y_test))
xg_train = accuracy_score(y_train,xgb.predict(X_train))
xg_test = accuracy_score(preds,y_test)
```

```
train accuracy:  0.8381984987489575
test accuracy:  0.6825
```

FIGURE 72- 4.5.5.1 XGB ACCURACY

Here we can observe the performance of the gradient boosting classifier model.
We got the accuracy of the train data as 0.83 or 83%
We got the accuracy of the test data as 0.68 or 68%

By comparing the above models, the random forest and GradientBoosting seems to yield the highest level of accuracy. However, since GradientBoosting has a better f1-score for predicting good quality wines

CONFUSION MATRIX:-

```
XG_conf_matrix = pd.DataFrame(data=metrics.confusion_matrix(y_test,preds),
                              index=[['low','medium','high']], columns=[['low','medium','high']])
print("Confusion Matrix: ")
XG_conf_matrix
```

```
Confusion Matrix:
```

	low	medium	high
low	24	9	16
medium	1	161	42
high	21	38	88

```
#heatmap of confusion matrix:
sns.heatmap(XG_conf_matrix,annot=True)
plt.title('CONFUSION MARTIX')
plt.show()
```

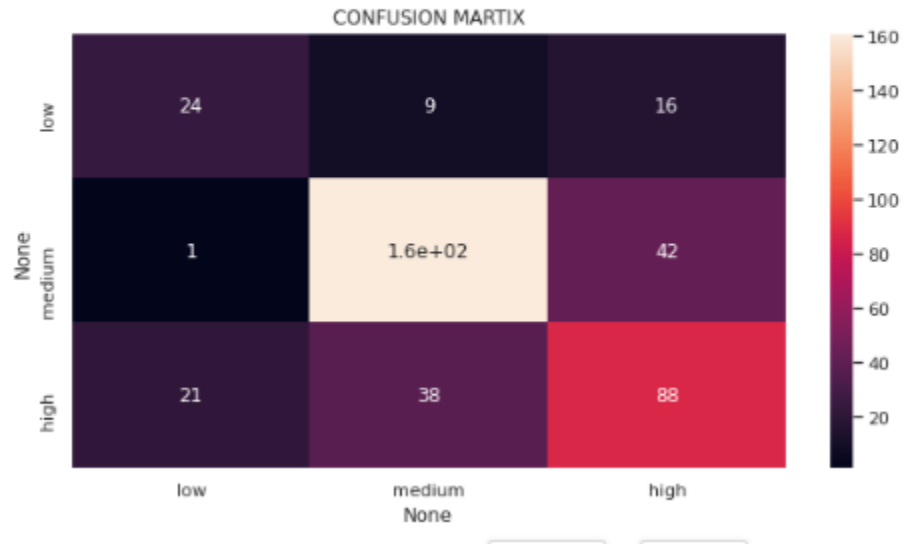


FIGURE 73- 4.5.5.2 XGB CONFUSION MATRIX

4.6 Evaluating all the models performance

Now we need to compare all the models performance and select the best model which gives us the best accuracy for the training and testing data. I have created a dictionary of all the accuracies and labels and now we use bar plot to visualize it as shown.

From all the above models, visualizing the best model.

```
[76] acc = {'Decision_train':dec_train,
            'Logistic_train':log_train,
            'RandomForest_train':ran_train,
            'GradientBoost_train':grad_train,
            'XGBoost_train':xg_train,}
     x = acc.keys()
     y = acc.values()
```

```
plt.figure(figsize=(12,9))
plt.xlabel("CLASSIFIER")
plt.ylabel("ACCURACY")
plt.title("TRAINING")
plt.bar(list(x),list(y))
plt.show()
```

FIGURE 74- 4.6.1 MAKING DICTIONARIES FOR ACCURACIES

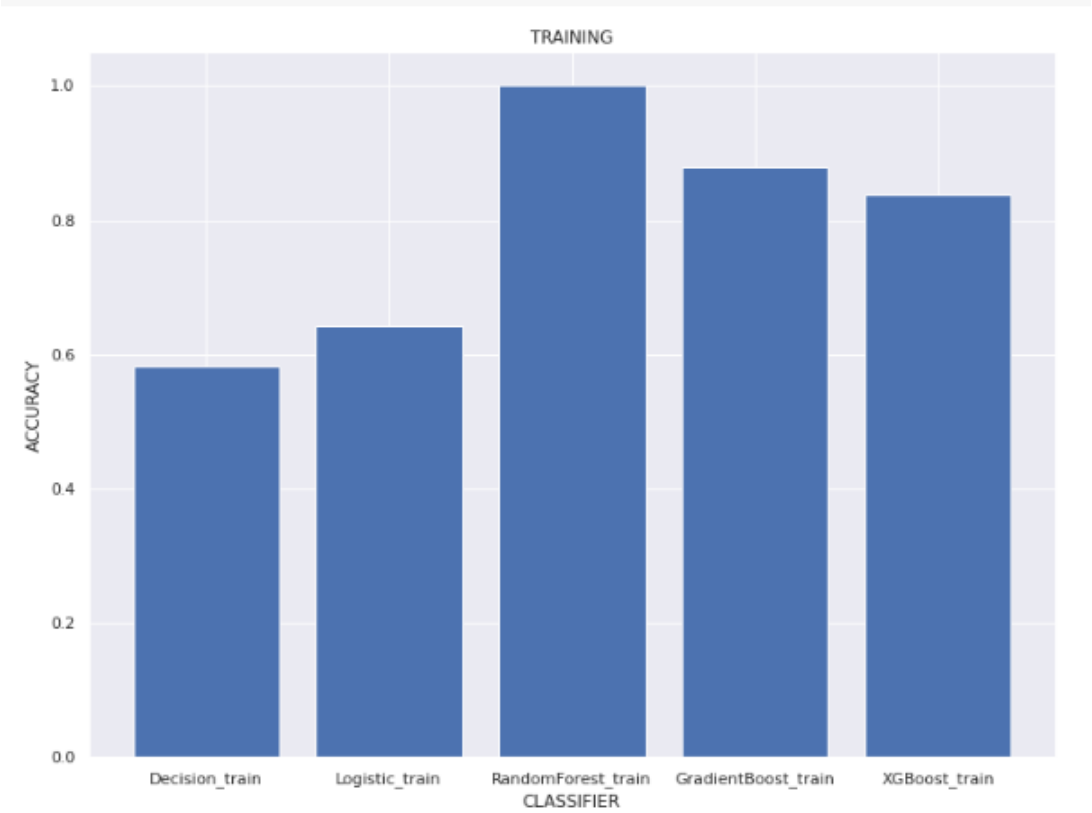


FIGURE 75- 4.6.2 TRAIN MODELS ACCURACY

TESTING

```
acc = {'Decision_test':dec_test,
       'Logistic_test':log_test,
       'RandomForest_test':ran_test,
       'GradientBoost_test':grad_test,
       'XGBoost_test':xg_test}
x1 = acc.keys()
y2 = acc.values()
```

```
[86] plt.figure(figsize=(12,9))
     plt.xlabel("CLASSIFIER")
     plt.ylabel("ACCURACY")
     plt.title("TESTING")
     plt.bar(list(x1),list(y2))
     plt.show()
```

FIGURE 76- 4.6.3 MAKING DICTIONARIES FOR ACCURACIES

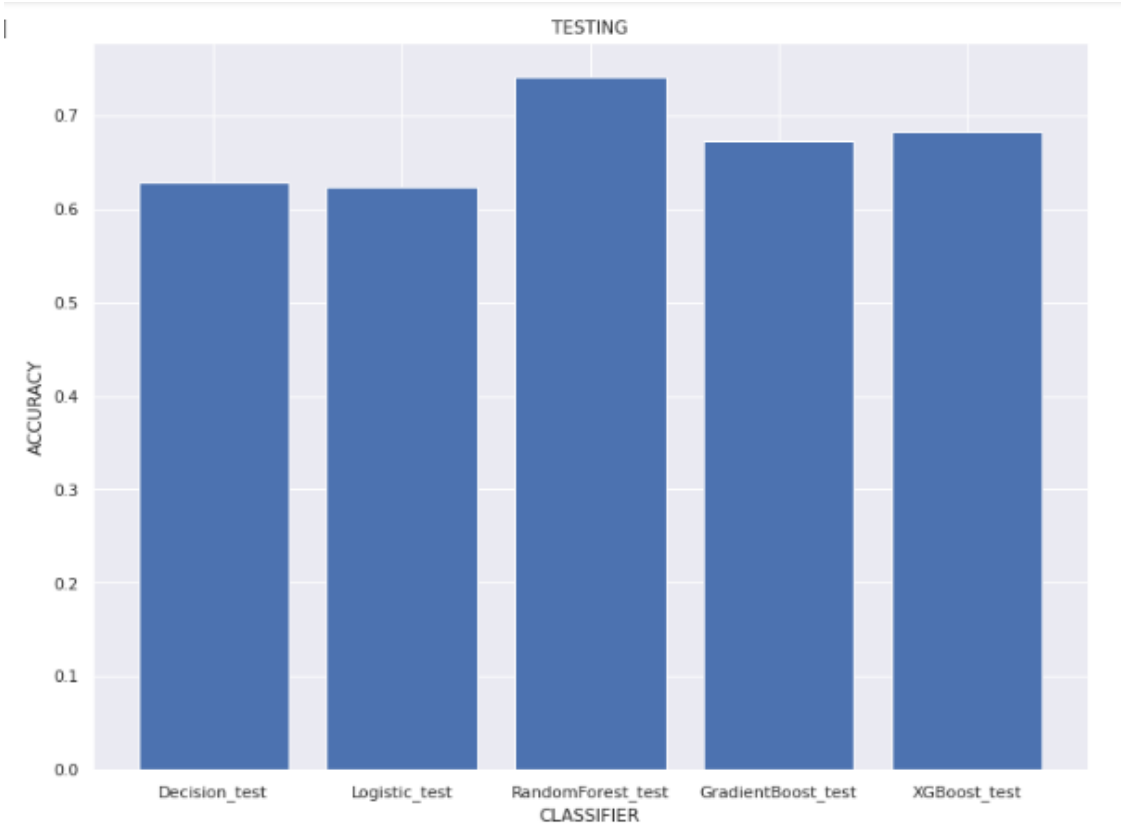


FIGURE 77- 4.6.4 TRAIN MODELS ACCURACY

From above we can observe that Random Forest performed best among the other entire algorithm.
So, we perform passing a sample to Random Forest classifier to see the output of the given data.

4.7 PASSING A SAMPLE INPUT TO THE MODEL:-

```
'] ## buliding a model
  from sklearn.ensemble import RandomForestClassifier
  rfc = RandomForestClassifier(n_estimators = 100)
  rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

FIGURE 78- 4.7.1 BUILD THE BEST MODEL

We have passed a sample input to the Random Forest classifier.

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

```
array(['low'], dtype=object)
```

+ Code + Text

we predicted an output of is low quality wine for the sample data that we have given to the model.

FIGURE 79- 4.7.2 PREDICTING THE QUALITY FOR A SAMPLE INPUT

After passing the input we got the output.
We got low quality wine

```
[414] input_sample = [[7.9 ,0.35 ,0.46 ,3.6,0.078,15,37,0.997,3.35,0.86,12.8]]
      rfc.predict(input_sample)
```

```
array(['high'], dtype=object)
```

we predicted an output of high quality wine for the sample data that we have given

FIGURE 80- 4.7.3 PREDICTING THE QUALITY FOR A SAMPLE INPUT

After passing the input we got the output.
We got high quality win

4.8 CONCLUSION:

After trying many models and optimizing their hyper-parameters, this study reaches a conclusion that random forest model performs the best for prediction and for classification. For predicting wine quality, RF model gives the best of test and train as 0.71 and 1.0, which is not bad considering that the wine quality could range from 0 to 10. For classifying high-quality wine, RF model gives the best overall accuracy.

4.9 REFERENCES:

- https://towardsdatascience.com/understanding-random-forest-58381e0602d2
- https://expertsystem.com/machine-learning-definition/
- https://www.kdnuggets.com/2018/02/logistic-regression-concise-technical-overview.html
- https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

GITHUB REFERENCE LINK:- https://github.com/dubba1212/WINE-QUALITY-PREDICTION